



Développement et déploiement de modules



Développement et déploiement de modules

Important

Avant d'utiliser les informations de ce document, lisez les informations générales figurant à la section Remarques qui se trouve à la fin du présent document.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- <http://www.fr.ibm.com> (serveur IBM en France)
- <http://www.can.ibm.com> (serveur IBM au Canada)
- <http://www.ibm.com> (serveur IBM aux Etats-Unis)

*Compagnie IBM France
Direction Qualité
Tour Descartes
92066 Paris-La Défense Cedex 50*

© Copyright IBM France 2009. Tous droits réservés.

© **Copyright International Business Machines Corporation 2005, 2009.**

Manuels PDF et Centre de documentation

Les manuels PDF sont fournis pour votre convenance afin de les imprimer et de les consulter hors ligne. Pour obtenir les informations les plus récentes, consultez le Centre de documentation en ligne.

Dans l'ensemble, les manuels PDF contiennent les mêmes informations que le Centre de documentation.

La documentation PDF est disponible au plus tard un trimestre après une édition majeure du centre de documentation, comme Version 6.0 ou Version 6.1.

La documentation PDF est moins fréquemment mise à jour que le Centre de documentation mais plus fréquemment que les Redbooks. En général, les manuels PDF sont mis à jour lorsqu'il existe un nombre suffisant de modifications.

Les liens du manuel PDF pointant vers des rubriques sont dirigés vers le Centre de documentation sur le Web. Les liens pointant vers des cibles sont marqués par des icônes qui indiquent si la cible est un manuel PDF ou une page Web.

Tableau 1. Icônes liant à des rubriques situées en dehors de ce manuel



Icône	Description
	<p>Lien vers une page Web, y compris une page du Centre de documentation.</p> <p>Les liens vers le Centre de documentation passent par un service d'adressage indirect de telle sorte qu'ils sont toujours valides même si la rubrique cible est déplacée vers un emplacement différent.</p> <p>Si vous voulez trouver une page liée dans un centre de documentation local, vous pouvez effectuer une recherche sur le titre du lien. Vous pouvez également effectuer une recherche sur l'ID de la rubrique. Si la recherche donne plusieurs résultats dans différentes rubriques pour des différents produits, vous pouvez utiliser les commandes Grouper par du résultat de la recherche pour identifier l'instance de la rubrique que vous souhaitez afficher. Par exemple :</p> <ol style="list-style-type: none">1. Copiez l'URL du lien. Par exemple, cliquez avec le bouton droit de la souris sur le lien puis sélectionnez Copier l'emplacement du lien. Par exemple : <code>http://www14.software.ibm.com/webapp/wsbroker/redirect?version=wbpm620&product=wesb-dist&topic=tins_apply_service</code>2. Copiez l'ID de la rubrique après <code>&topic=</code>. Par exemple : <code>tins_apply_service</code>3. Dans la zone de recherche de votre centre de documentation local, collez l'ID de la rubrique. Si la fonction de documentation est installée localement, le résultat de la recherche affichera la rubrique. Par exemple : <div data-bbox="613 1577 1455 1772" style="border: 1px solid black; border-radius: 10px; padding: 10px;"><p>1 résultat(s) trouvé pour</p><p>Grouper par : Néant Plateforme Version Produit</p><p>Afficher le récapitulatif</p><p>Installation de groupes de correctifs et de groupes de mises à jour avec Update Installer</p></div>
	Lien vers un manuel PDF.

Table des matières

Manuels PDF et Centre de documentation	iii
--	-----

Figures	vii
-------------------	-----

Tableaux	ix
--------------------	----

Partie 1. Développement d'applications 1

Chapitre 1. Développement de solutions d'intégration métier 3

Modèle de programmation pour l'intégration métier	6
Architecture et modèles d'intégration métier	8
Scénarios d'intégration métier	9
Rôles, produits et défis techniques	10
Infrastructure d'objets métier	12
Architecture de composants de service	14
Processus métier	19
Tâches utilisateur	20
Création d'applications d'intégration métier	21

Chapitre 2. Développement de modules de service 23

Présentation du développement de modules	24
Développement de composants de service	26
Appel de composants	28
Appel dynamique d'un composant	30
Présentation de l'isolement des modules et des cibles	31
Liaisons HTTP	35

Chapitre 3. Guides et techniques de programmation 37

Programmation d'objets métier	37
Tableaux dans les objets métier	38
Création d'objets métier imbriqués	40
Objets métier : renforcement du schéma et prise en charge du schéma industriel	44
Remplacement d'une conversion d'objet SDO en Java	66
Remplacement de l'implémentation d'architecture SCA générée	67
Règles en exécution de la conversion de Java en objets SDO	68
Validation de document XML	70
Propagation d'en-tête de protocole à partir de liaisons d'exportation non SCA	72
Gestion des règles métier	74
Modèle de programmation	75
Exemples	111
Classes d'opérations communes	188

Chapitre 4. Développement d'applications client pour les tâches et processus métier 199

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur	200
Requêtes portant sur les données des processus métier et des tâches	203
Comparaison des interfaces de programmation destinées à l'extraction de données de processus et de tâche	204
Tables de requêtes dans Business Process Choreographer	206
API de requête EJB de Business Process Choreographer	270
Développement d'applications client EJB pour des processus métier et des tâches utilisateur	286
Accès aux API EJB	289
Requête sur des objets liés aux processus métier et aux tâches	295
Développement d'applications pour les processus métier	301
Développement d'applications pour des tâches utilisateur	328
Développement d'applications pour les processus métier et les tâches utilisateur	350
Gestion des exceptions et des erreurs	357
Développement d'applications API de service Web	361
Composants de service Web et séquence de contrôle	362
Présentation des API des services Web	363
Exigences en termes de processus métier et de tâches utilisateur	364
Développement d'applications client	364
Copie d'artefacts	365
Développement d'applications client dans l'environnement de services Web Java	375
Développement d'applications client dans l'environnement .NET	387
Requêtes sur des objets liés aux processus métier et aux tâches	393
Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer	397
Exigences des processus métier	398
Autorisation pour les affichages JMS	399
Accès à l'interface JMS	399
Copie d'artefacts pour les applications client JMS	403
Vérification du message de réponse pour les exceptions de métier	404
Exemple : exécution d'un processus de longue durée à l'aide de l'API JMS de Business Process Choreographer	404

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF.	405
Composants Exemples de Business Process Choreographer Explorer.	410
Traitement des erreurs dans les composants JSF	411
Convertisseurs et intitulés par défaut d'objets de modèle client	412
Ajout du composant List à une application JSF	413
Ajout du composant Details à une application JSF	421
Ajout du composant CommandBar à une application JSF	423
Ajout du composant Message à une application JSF	428
Développement des pages JSP pour les messages de tâche et de processus.	432
Fragments JSP définis par l'utilisateur	433
Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur	435
Création de gestionnaires d'événements d'API	436
Création de gestionnaire d'événements de notification	439
Installation des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification	441
Enregistrement des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification avec des modèles de tâche et des tâches	442
Création, installation et exécution de plug-ins en vue du post-traitement des résultats des requêtes sur les utilisateurs.	443

Partie 2. Déploiement des applications 449

Chapitre 5. Présentation de la préparation et de l'installation de modules 451

Présentation des bibliothèques et des fichiers JAR	452
Présentation du fichier EAR	455
Préparation au déploiement sur un serveur	456
Remarques concernant l'installation d'applications de service sur des clusters	458

Chapitre 6. Déploiement d'un module 461

Installation de modules SCA versionnés dans un environnement de production	462
Installation d'un module SCA avec la console	463
Création d'un fichier EAR installable via serviceDeploy	465
Déploiement d'applications à l'aide des tâches ANT Apache	465

Chapitre 7. Installation des applications de tâche utilisateur et de processus métier. 469

Installation d'applications de processus métier et de tâches utilisateur dans un environnement de déploiement réseau	470
Déploiement des processus métier et des tâches utilisateur	471
Installation d'applications de processus métier et de tâche utilisateur en mode interactif	472
Configuration de la source de données d'une application de processus et des paramètres de référence d'ensemble	473
Désinstallation d'applications de processus métier et de tâche utilisateur à l'aide de la console d'administration	474
Désinstallation d'applications de processus métier et de tâches humaines à l'aide d'une commande d'administration	475

Chapitre 8. Adaptateurs et installation 479

Chapitre 9. Identification et résolution des incidents lors d'un échec de déploiement 481

Suppression des spécifications d'activation J2C	482
Suppression des destinations SIBus	483

Partie 3. Annexes 485

Remarques 487

Figures

1. Les outils IBM prolongent le cycle de vie complet de la gestion des processus métier, ce qui vous permet de concevoir, assembler, déployer et gérer vos processus.	5
2. Structure à base de composants WebSphere Process Server.	15
3. SCA dans WebSphere Process Server	16
4. Diagramme d'assemblage.	17
5. Modèle d'appel simple.	32
6. Appel de service unique par des applications multiples	33
7. Modèle d'appel isolé du service UpdateCalculateFinal	34
8. Modèle d'appel isolé du service UpdatedCalculateFinal.	35
9. Propagation de contexte comprenant un en-tête de protocole	73
10. Diagramme de classes de BusinessRuleGroup et classes associées	80
11. Diagramme de classes de Property et classes associées	82
12. Diagramme de classes de Operation et classes associées	85
13. Diagramme de classes de BusinessRule et classes associées	87
14. Diagramme de classes de BusinessRule et classes associées	90
15. Diagramme de classes de DecisionTable et classes associées	92
16. Diagramme de classes de TreeNode et classes associées	95
17. Diagramme de classes de TreeAction et classes associées	99
18. Diagramme de classes de DecisionTableRule et classes associées.	100
19. Diagramme de classes de Template et de Parameter, et classes associées	102
20. Diagramme de classes de BusinessRuleManager et module	104
21. Diagramme de classes de QueryNodeFactory et classes associées.	106
22. Diagramme de classes de BusinessRuleManagementException et classes associées	108
23. Tables de requêtes dans Business Process Choreographer	207
24. Contenu des tables de requêtes composites	216
25. Table de requêtes composite avec critères de sélection	218
26. Filtres dans les tables de requêtes composites	222
27. Filtres et critères de sélection dans des expressions	230
28. Autorisation par instance pour les tables de requêtes	236
29. Relations entre module, composants et bibliothèques	453

Tableaux

1. Icônes liant à des rubriques situées en dehors de ce manuel	iii	30. Paramètres utilisateur destinés à l'API de table de requêtes	258
2. Abstractions des données et implémentations correspondantes	13	31. Propriétés d'un ensemble de résultats d'entités renvoyé par l'API de table de requêtes	259
3. Conversion de type WSDL en classe Java	70	32. Propriétés d'une entité renvoyée par l'API de table de requêtes	259
4. Problèmes liés aux groupes de règles métier	108	33. Propriétés d'un ensemble de résultats de lignes renvoyé par l'API de table de requêtes .	260
5. Problèmes liés aux ensembles de règles et aux tables de décisions	109	34. Méthodes pour l'extraction de métadonnées des tables de requêtes.	261
6. Propriétés des tables de requêtes prédéfinies	210	35. Métadonnées relatives à la structure d'une table de requêtes	262
7. Tables de requêtes prédéfinies contenant des données d'instance.	211	36. Métadonnées relatives à l'internationalisation d'une table de requêtes	263
8. Tables de requêtes prédéfinies contenant des données de modèle	212	37. Les options applicables aux tables de requêtes composites et leur impact sur les performances des requêtes	266
9. Propriétés des tables de requêtes supplémentaires	214	38. Les options de l'API de table de requêtes et leur impact sur les performances des requêtes	267
10. Contenus valides d'une table de requêtes composite.	219	39. Performances des tables de requêtes - Autres considérations	269
11. Contenus non valides d'une table de requêtes composite.	219	40.	271
12. Propriétés des tables de requêtes composites	219	41. Méthodes API pour les modèles de processus	325
13. Etapes de développement de tables de requêtes	224	42. Les méthodes API sont liées au démarrage des instances de processus.	325
14. Les attributs des tables de requêtes et leur utilisation dans les expressions.	229	43. Méthodes API pour le contrôle du cycle de vie des instances de processus	326
15. Types d'autorisation pour les tables de requêtes	235	44. Méthodes API pour le contrôle du cycle de vie des instances d'activité	327
16. Types d'éléments de travail	237	45. Méthodes API pour les variables et les propriétés personnalisées	327
17. Éléments de travail et critères d'affectation de personnes.	238	46. Méthodes API pour les modèles de tâches	347
18. Types d'attribut	240	47. Méthodes API pour les modèles de tâches	348
19. Correspondance entre types des bases de données et types d'attribut	241	48. Méthodes API de gestion des escalades	348
20. Exemple de mappage entre types de base de données et types d'attribut	242	49. Méthodes API pour les variables et les propriétés personnalisées	349
21. Correspondance entre types d'attribut et valeurs littérales	243	50. Mappage des liaisons de référence aux noms JNDI	408
22. Correspondance entre types d'attribut et valeurs de paramètre utilisateur	244	51. Mappage d'interfaces de Business Process Choreographer avec des objets de modèle client	413
23. Correspondance entre types d'attribut et types d'objet Java	245	52. Attributs bpe:list	420
24. Compatibilité entre types d'attribut	246	53. Attributs bpe:column	420
25. Méthodes pour les requêtes exécutées sur les tables de requêtes	249	54. Attributs bpe:details	423
26. Paramètres de l'API de table de requêtes	251	55. Attributs bpe:property	423
27. Paramètres de l'API de table de requêtes : options de filtrage	253	56. Attributs bpe:commandbar	427
28. Paramètres de l'API de table de requêtes : options d'autorisation par défaut pour l'autorisation par instance	256	57. Attributs bpe:command	428
29. Paramètres de l'API de table de requêtes : AdminAuthorizationOptions	257	58. Attributs bpe:form	431

Partie 1. Développement d'applications

Chapitre 1. Développement de solutions d'intégration métier

Cette section évoque les principes du modèle de programmation pour l'intégration métier. Elle présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier.

L'intégration métier est la discipline permettant aux entreprises d'identifier, consolider et optimiser des processus métier. Son objectif est d'améliorer la productivité et maximiser l'efficacité de l'entreprise. L'intérêt grandissant que suscite l'intégration métier est dû aux fusions et aux consolidations d'entreprises et au fait qu'elles développent des bibliothèques d'éléments d'actif informationnel divers. Ces éléments d'actif manquent souvent de cohérence et de coordination, ce qui crée des "îlots d'informations".

L'intégration métier est étroitement liée à la gestion des processus métier (BPM - Business Process Management) et à l'architecture SOA (Service-Oriented Architecture). En fonction du type d'entreprise et de l'étendue des besoins d'intégration, l'intégration métier impose diverses exigences aux services informatiques. Certains projets peuvent seulement être confrontés à quelques aspects de ces exigences, alors que des projets de plus grande envergure peuvent regrouper un grand nombre d'entre elles. Vous trouverez ci-dessous quelques aspects, parmi les plus courants, composant des projets d'intégration métier :

- **Intégration d'applications** est une exigence courante. La complexité des projets d'intégration d'applications varie selon qu'il s'agit de situations simples, dans lesquelles vous devez garantir qu'un nombre réduit d'applications peut partager des informations, ou de situations plus complexes, dans lesquelles des transactions et des échanges de données doivent apparaître simultanément dans plusieurs applications dorsales. L'intégration d'applications complexe exige souvent une gestion de l'unité de travail compliquée, mais aussi de la transformation et du mappage.
- **Automatisation des processus** est un autre aspect clé qui assure que les activités exercées par une personne ou une entreprise déclenchent automatiquement des conséquences ailleurs. Ceci garantit l'accomplissement du processus métier global. Par exemple, lorsqu'une entreprise embauche un employé, les informations de la feuille de paie doivent être mises à jour, le service de sécurité doit appliquer des actions adéquates, les outils requis doivent être mis à la disposition de l'employé, etc. Certaines activités composant un processus peuvent capturer les entrées des utilisateurs et leur interaction, alors que d'autres peuvent appeler des scripts sur des systèmes dorsaux et d'autres services présents dans l'environnement.
- **Connectivité** est un aspect abstrait, et pourtant essentiel, pour une entreprise et les partenaires commerciaux. Par "connectivité", nous voulons parler du flux d'informations échangées entre les entreprises ou les sociétés et la capacité à accéder à des services informatiques distribués.

Certains défis techniques liés aux implémentations d'intégrations métier peuvent se résumer de la façon suivante :

- Traiter différents formats de données et ne pas être en mesure d'effectuer une transformation efficace des données
- Traiter différents protocoles et mécanismes pour accéder à des services informatiques qui ont pu être développés via des technologies très différentes

- Organiser différents services informatiques qui peuvent être distribués géographiquement ou offerts par différentes entreprises
- Fournir des règles et des mécanismes pour classer et gérer les services qui sont disponibles (gouvernance)

L'intégration métier en tant que telle regroupe de nombreux thèmes et éléments qui sont également communs à l'architecture SOA. La vision d'IBM concernant l'intégration métier se fonde sur de nombreux concepts de base identiques figurant dans l'architecture SOA. L'une des conséquences directes de cette vision est que les solutions d'intégration métier peuvent exiger le recours à plusieurs produits pour leur élaboration. IBM® fournit toute une gamme d'outils et de plateformes d'exécution afin de prendre en charge les différentes étapes et aspects opérationnels.

Pour paraphraser la vision d'IBM concernant l'intégration métier, elle doit permettre aux entreprises de définir, créer, fusionner, consolider et simplifier les processus métier à l'aide d'applications exécutées sous une infrastructure informatique SOA. Le travail d'intégration métier se base véritablement sur des rôles. Au niveau macro, ceci implique la conception, le développement, la gouvernance, la gestion et la surveillance des applications de processus métier. Grâce à l'utilisation d'outils et de procédures adéquats, vous pouvez automatiser les processus métier impliquant des personnes et des systèmes hétérogènes à l'intérieur, mais aussi à l'extérieur, de l'entreprise. L'un des points clés de l'intégration métier est la possibilité d'optimiser vos activités commerciales afin qu'elles soient suffisamment efficaces, évolutives, fiables et flexibles pour gérer des modifications.

L'intégration métier exige des outils de développement, des serveurs d'exécution, des outils de surveillance, un référentiel de services, des boîtes à outils et des modèles de processus. Etant donné que l'intégration métier se compose de nombreux aspects différents, vous allez découvrir que plusieurs outils de développement doivent être utilisés pour développer une solution. Ces outils permettent aux développeurs d'intégration d'assembler des solutions métier complexes. Un serveur est un moteur d'activités à hautes performances ou un conteneur de services qui exécute des applications complexes. La direction veut toujours être informée de l'attribution des tâches au sein de l'entreprise et c'est à ce moment-là que les outils de surveillance interviennent. A mesure que les entreprises créent des processus ou services métier, la gouvernance, la classification et le stockage de ces services deviennent essentiels. Cette fonction est mise à disposition par un référentiel de services. Des boîtes à outils spéciales permettant de créer des éléments spécifiques à la solution, comme les connecteurs ou les adaptateurs de systèmes existants, sont souvent requises.

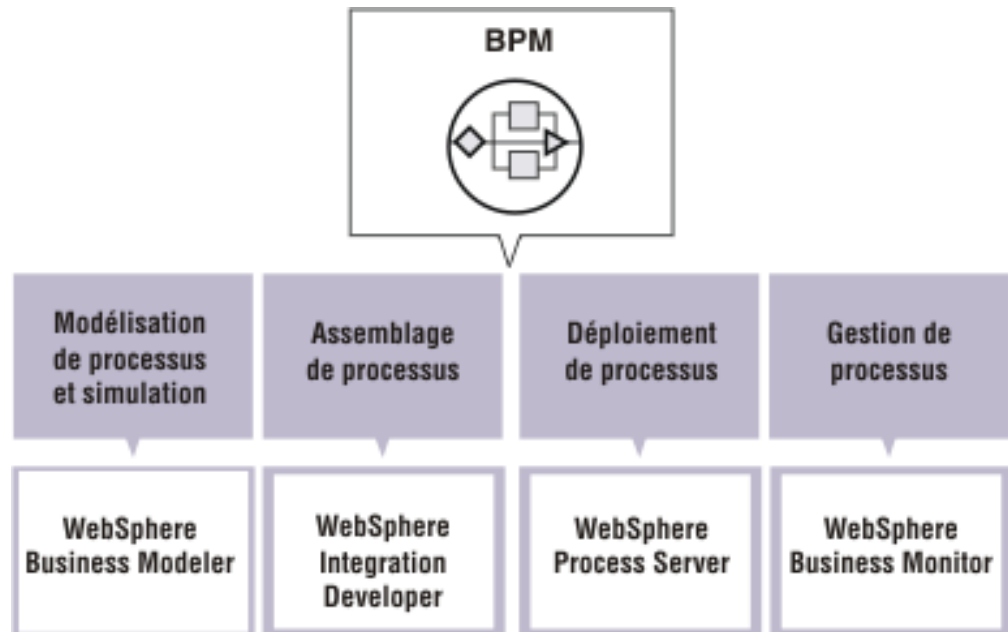


Figure 1. Les outils IBM prolongent le cycle de vie complet de la gestion des processus métier, ce qui vous permet de concevoir, assembler, déployer et gérer vos processus.

L'intégration métier ne se base pas sur un produit unique. Elle mobilise la quasi-totalité du personnel et tous les aspects commerciaux d'une et de plusieurs entreprises. L'intégration métier regroupe de nombreux services et éléments figurant dans l'architecture de référence SOA.

Pour plus de détails sur ces concepts et pour consulter des exemples de programmation, reportez-vous à :

- *WebSphere Business Integration Primer : Process Server, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1 : Development*, IBM Redbooks, SG24-7608-00, June 2008.

Concepts associés

Modèle de programmation pour l'intégration métier

L'intégration métier n'est pas une tâche facile. Il existe un très grand nombre de technologies et de méthodes permettant de représenter ou d'interagir avec des données. Par conséquent, le fait de réussir une intégration se révèle être une tâche difficile. Si vous considérez les trois aspects composant un modèle de programmation (données, appel et composition) et que vous appliquez les nouveaux paradigmes d'une approche orientée services, le nouveau modèle de programmation d'une architecture SOA commence à prendre forme.

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

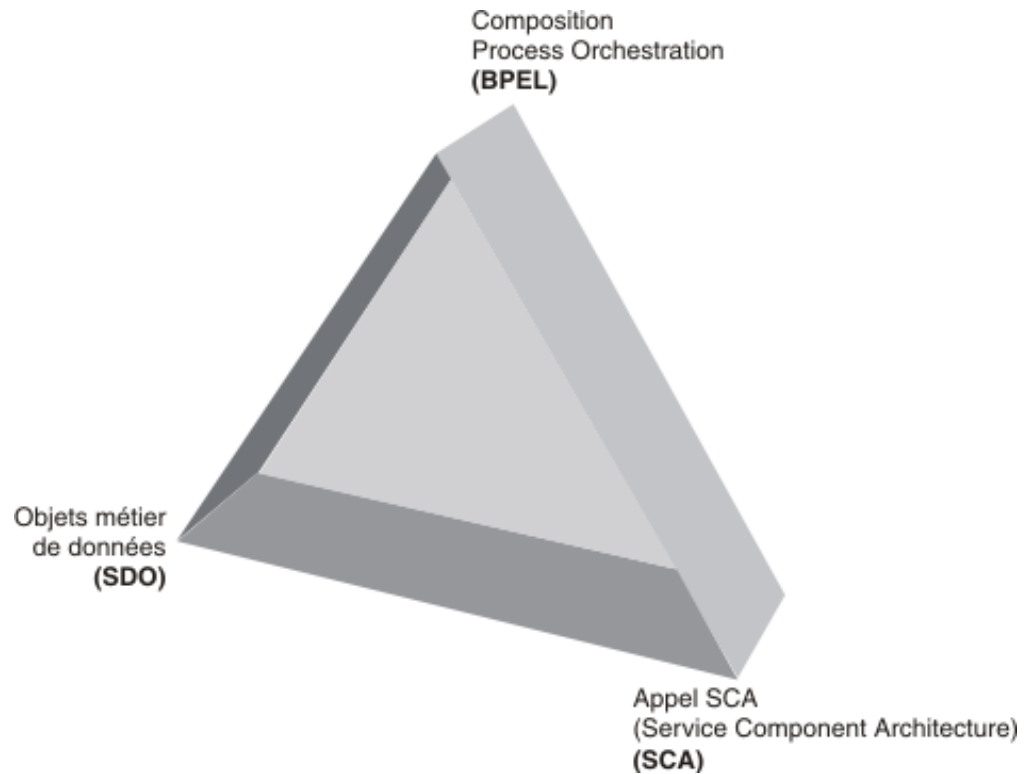
Création d'applications d'intégration métier

Intégration métier implique l'intégration d'applications, de données et de processus dans une ou plusieurs entreprises. L'intégration implique également le développement de processus, car il existe une certaine logique dans la séquence d'applications assemblées, afin de les intégrer. WebSphere Integration Developer permet de créer des applications d'intégration métier.

Modèle de programmation pour l'intégration métier

L'intégration métier n'est pas une tâche facile. Il existe un très grand nombre de technologies et de méthodes permettant de représenter ou d'interagir avec des données. Par conséquent, le fait de réussir une intégration se révèle être une tâche difficile. Si vous considérez les trois aspects composant un modèle de programmation (données, appel et composition) et que vous appliquez les nouveaux paradigmes d'une approche orientée services, le nouveau modèle de programmation d'une architecture SOA commence à prendre forme.

Tout d'abord, nous constatons que le langage XML (Extensible Markup Language) est principalement utilisé pour représenter des données et que leur programmation est effectuée à l'aide d'objets SDO (Service Data Objects) ou de fonctions XML natives, telle que XPath ou XSLT (Extensible Stylesheet Language Transformation). Ensuite, un appel de service effectue un mappage vers l'architecture SCA (Service Component Architecture). Pour finir, la composition est intégrée à l'orchestration des processus à l'aide du langage BPEL (Business Process Execution Language). Le schéma suivant illustre les trois aspects de ce nouveau modèle de programmation.



Architecture SCA

En plus de fournir une syntaxe cohérente et un mécanisme d'appel des services, l'architecture SCA sert de cadre d'appel et permet aux développeurs d'encapsuler les implémentations de services dans des composants réutilisables. Elle permet aux développeurs de définir des interfaces, des implémentations et des références indépendamment du point de vue technologique, vous donnant ainsi la possibilité d'associer des éléments à la technologie de votre choix. L'architecture SCA distingue la logique métier de l'infrastructure afin que les programmeurs d'application puissent se consacrer à la résolution de problèmes métier.

Concepts associés

Développement de solutions d'intégration métier

Cette section évoque les principes du modèle de programmation pour l'intégration métier. Elle présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier.

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Cette section présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier. L'utilisation de modèles semblent se généraliser dans notre vie quotidienne. Les modèles de patrons, les modèles d'apprentissage personnalisé ("Think-and-learn") destinés aux enfants, les modèles pour la construction de maisons individuelles, les modèles de sculpture sur bois, les modèles de vols, les modèles de configuration des vents, les modèles de pratique en médecine, les modèles d'achat des clients, les modèles de flux de travaux, les modèles de conception en informatique, et beaucoup d'autres encore.

Les modèles se révèlent être utiles pour les concepteurs et développeurs de solutions. Il n'est donc pas surprenant de voir des modèles d'intégration métier et d'intégration d'entreprise. Il existe toute une gamme de modèles pouvant être appliqués à l'intégration métier, y compris des modèles de demande et de réponse pour le routage, des modèles de canaux (publication/abonnement) et beaucoup d'autres encore. Les modèles abstraits fournissent un modèle de résolution appliqués à une certaine catégorie de problèmes, alors que les modèles concrets fournissent des indications plus précises concernant la méthode d'implémentation d'une solution spécifique. Cette section traite des modèles devant traiter l'appel de données et de services qui constituent la structure du modèle de programmation de la stratégie des logiciels IBM pour l'intégration métier de WebSphere.

Concepts associés

Développement de solutions d'intégration métier

Cette section évoque les principes du modèle de programmation pour l'intégration métier. Elle présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier.

Scénarios d'intégration métier

Les entreprises disposent de nombreux systèmes logiciels différents qui sont utilisés dans le cadre de leur activité. De plus, l'intégration de ces composants métier est propre à chaque entreprise.

Rôles, produits et défis techniques

La réussite de projets d'intégration métier dépend de l'association de rôles de développement spécialisés, de techniques de programmation et de suites d'outils.

Infrastructure d'objets métier

L'industrie du logiciel a développé plusieurs modèles et infrastructures de programmation permettant aux développeurs d'encapsuler des informations sur un objet métier (BO - Business Object). L'infrastructure d'un BO doit généralement assurer l'indépendance de la base de données, mapper de façon transparente les objets métier personnalisés par l'administrateur vers les tables de la base de données ou vers les structures de données dans les systèmes d'information d'entreprise, mais aussi associer les objets métier aux interfaces utilisateur. Les schémas XML sont récemment devenus les schémas les plus courants et les plus acceptés pour représenter la structure d'un objet métier.

Architecture de composants de service

L'architecture SCA est un concept que vous pouvez implémenter de différentes manières. Elle n'exige aucune technologie, langage de programmation, protocole d'appel ou mécanisme de transport particulier. Les composants SCA sont décrits à l'aide du langage SCDL (Service Component Definition Language) qui est un langage basé sur XML.

Processus métier

Processus métier, plus précisément, processus métier basés sur BPEL, qui constituent la base des composants de service dans l'architecture SCA.

Tâches utilisateur

Une tâche utilisateur est un composant impliquant l'interaction des personnes et des services.

Scénarios d'intégration métier

Les entreprises disposent de nombreux systèmes logiciels différents qui sont utilisés dans le cadre de leur activité. De plus, l'intégration de ces composants métier est propre à chaque entreprise.

Les deux scénarios d'intégration de processus métier les plus répandus sont les suivants :

- **Courtier d'intégration** : Dans ce scénario, la solution d'intégration métier agit comme intermédiaire entre plusieurs d'applications dorsales. Par exemple, vous devez vous assurer que lorsqu'un client passe une commande à l'aide de l'application de gestion de commandes en ligne, la transaction met à jour les informations correspondantes dans votre application dorsale CRM (Customer Relationship Management). Dans ce scénario, la solution d'intégration doit pouvoir capturer et éventuellement transformer les informations requises de l'application de gestion de commandes et appeler les services correspondants dans l'application CRM.
- **Automatisation des processus** : Dans ce scénario, la solution d'intégration sert de lien entre les différents services informatiques qui, dans le cas contraire,

n'auraient aucun point commun. Par exemple, lorsqu'une entreprise embauche un employé, la série d'actions suivante doit se dérouler :

- Les informations relatives à l'employé sont ajoutées au système de feuille de paie.
- L'employé doit pouvoir accéder physiquement aux infrastructures et un badge doit lui être fourni.
- L'entreprise peut fournir un ensemble de ressources matérielles à l'employé (espace bureau, ordinateur, etc.).
- Le service informatique doit créer un profil utilisateur pour l'employé et autoriser l'accès à toute une série d'applications.

L'automatisation de ce processus est courante dans un scénario d'intégration métier. Dans ce cas, la solution implémente un flux automatisé qui est déclenché du fait de l'ajout de l'employé au système de feuille de paie. Par conséquent, le flux déclenche les étapes suivantes en créant des éléments de travail pour les preneurs de décisions ou en appelant les services correspondants.

Dans ces deux scénarios, la solution d'intégration doit accomplir les actions suivantes :

1. Utiliser des sources d'informations diverses et des formats de données différents, mais aussi pouvoir convertir des informations entre différents formats.
2. Pouvoir appeler plusieurs services, en utilisant éventuellement différents mécanismes et protocoles d'appel.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Rôles, produits et défis techniques

La réussite de projets d'intégration métier dépend de l'association de rôles de développement spécialisés, de techniques de programmation et de suites d'outils.

Les projets d'intégration métier exigent quelques éléments de base :

- Une séparation claire des rôles au sein de l'entreprise en charge du développement, afin de favoriser la spécialisation, ce qui améliore généralement la qualité des composants individuels qui sont développés.
- Un modèle d'objet métier (BO - Business Object) commun qui permet aux informations métier d'être représentées dans un modèle logique commun.
- Un modèle de programmation qui sépare clairement les interfaces des implémentations, qui prend en charge un mécanisme d'appel de service générique totalement indépendant de l'implémentation et qui concerne uniquement les interfaces.
- Un ensemble d'outils et de produits intégrés qui prend en charge les rôles de développement et empêche leur séparation.

Les sections suivantes détaillent chacun de ces éléments.

Séparation claire des rôles

Un projet d'intégration métier a besoin de personnel pour quatre rôles de collaboration clairement distincts :

- **Analyste métier** : Il s'agit d'experts de domaine en charge de capturer les aspects métier d'un processus et de créer un modèle de processus qui représente correctement le processus même. Leur mission est d'optimiser les performances financières d'un processus. Ils ne s'intéressent pas aux aspects techniques de l'implémentation de processus.
- **Développeur de composants** : Ils sont en charge de l'implémentation de services et de composants individuels. Leur mission consiste à utiliser une technologie spécifique pour l'implémentation. Ce rôle exige une formation solide en programmation.
- **Spécialiste en intégration** : Ce rôle relativement nouveau consiste à assembler un ensemble de composants existants dans une solution d'intégration métier plus grande. Les développeurs d'intégration n'ont pas besoin de connaître les détails techniques de chaque composant et service qu'ils réutilisent et connectent entre eux. Théoriquement, ils doivent uniquement s'intéresser à comprendre les interfaces des services qu'ils assemblent. Ils doivent utiliser les outils d'intégration pour le processus d'assemblage.
- **Déploieur de solutions** : Les déploieurs et les administrateurs de solutions se chargent de rendre les solutions d'intégration métier disponibles aux utilisateurs finaux. En théorie, un déploieur de solutions se charge principalement de lier une solution aux ressources physiques prêtes à la faire fonctionner (bases de données, gestionnaires de files d'attente, etc.) et non pas de comprendre le fonctionnement interne d'une solution. Sa mission première est la qualité de service (QoS - Quality of Service).

Un modèle d'objet métier commun

Comme nous l'avons mentionné précédemment, les aspects clés d'un projet d'intégration métier incluent la capacité à coordonner l'appel de plusieurs composants et à gérer l'échange de données entre eux. Plus particulièrement, différents composants peuvent utiliser différentes techniques pour représenter des éléments métier, comme les données d'une commande, les informations relatives à un client, etc. Par exemple, il se peut que vous ayez à intégrer une application Java™ qui utilise des EJB (Enterprise Java Beans) d'entité pour représenter des éléments métier et une application existante qui organise les informations dans des fichiers de stockage COBOL. Par conséquent, une plateforme dont l'objectif est de simplifier la création de solutions d'intégration doit également fournir une méthode générique pour représenter des éléments métier, en faisant abstraction des techniques utilisées par les systèmes dorsaux pour la gestion des données. WebSphere Process Server et WebSphere Enterprise Service Bus permettent d'y parvenir grâce à la *structure d'objets métier*.

Cette dernière permet aux développeurs d'utiliser des schémas XML afin de définir la structure des données métier, mais aussi d'accéder et de manipuler les instances de ces structures de données (objets métier) via un code XPath ou Java. L'infrastructure d'objets métier se base sur la norme SDO (Service Data Object).

Modèle de programmation de l'architecture SCA (Service Component Architecture)

Le modèle de programmation SCA représente la base de toute solution à développer sur WebSphere Process Server et WebSphere Enterprise Service Bus.

L'architecture SCA permet aux développeurs d'encapsuler les implémentations de services dans des composants réutilisables. Elle permet de définir des interfaces, des implémentations et des références indépendamment du point de vue technologique, vous donnant ainsi la possibilité d'associer des éléments à la technologie de votre choix. Il existe également un modèle de programmation client SCA qui permet d'appeler ces composants. Plus particulièrement, il permet aux infrastructures d'exécution basées sur Java d'interagir avec des exécutions non Java. L'architecture SCA utilise des objets métier comme éléments de données pour l'appel d'un service.

Outils et produits

IBM WebSphere Integration Developer est l'environnement de développement intégré qui dispose de tous les outils nécessaires pour créer et composer des solutions d'intégration métier basées sur les technologies susmentionnées. Ces solutions sont généralement déployées dans WebSphere Process Server ou, dans certains cas de figure, dans WebSphere Enterprise Service Bus.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Infrastructure d'objets métier

L'industrie du logiciel a développé plusieurs modèles et infrastructures de programmation permettant aux développeurs d'encapsuler des informations sur un objet métier (BO - Business Object). L'infrastructure d'un BO doit généralement assurer l'indépendance de la base de données, mapper de façon transparente les objets métier personnalisés par l'administrateur vers les tables de la base de données ou vers les structures de données dans les systèmes d'information d'entreprise, mais aussi associer les objets métier aux interfaces utilisateur. Les schémas XML sont récemment devenus les schémas les plus courants et les plus acceptés pour représenter la structure d'un objet métier.

D'un point de vue outils, WebSphere Integration Developer fournit aux développeurs un modèle de BO commun, afin de représenter plusieurs sortes d'entités issues de domaines différents. Pendant la phase de développement, WebSphere Integration Developer représente les objets métier sous la forme de schémas XML. Toutefois, au moment de l'exécution, ces mêmes objets métier sont représentés dans la mémoire par l'instance Java d'un objet SDO. Un objet SDO est une spécification standard que les systèmes IBM et BEA ont développé conjointement et qu'ils ont convenu d'utiliser. IBM a étendu la spécification SDO en incluant des services supplémentaires qui facilitent la manipulation des données dans les objets métier.

Avant de nous intéresser à l'infrastructure d'un BO, penchons-nous un instant sur les types basiques de données qui sont manipulées :

- **Données d'instance** correspond aux données réelles et aux structures de données pouvant contenir de simples objets de base dotés de propriétés scalaires jusqu'à des hiérarchies d'objets volumineux et complexes. Des définitions de

données sont également incluses, comme la description de types d'attributs de base, des informations de type complexe, la cardinalité et des valeurs par défaut.

- **Métadonnées d'instance** correspond à des données propres à une instance. Des informations incrémentielles sont ajoutées aux données de base, comme le suivi des modifications (aussi appelé "récapitulatif des modifications"), les informations de contexte associées à la manière dont l'objet ou les données ont été créé(e)s, ainsi que les en-têtes et pieds de page d'un message.
- **Métadonnées type** correspond généralement aux informations propres à une application, comme les mappages de niveau attribut vers des colonnes de données (par exemple, le mappage du nom d'une zone d'un BO en un nom de colonne de table SAP) de systèmes d'information d'entreprise (EIS - Enterprise Information System) cibles.
- **Services** correspond généralement aux services auxiliaires qui obtiennent et définissent les données, modifient les récapitulatifs ou fournissent un accès type pour la définition de données.

Le tableau suivant indique comment les types de données de base sont implémentés dans la plateforme WebSphere.

Tableau 2. Abstractions des données et implémentations correspondantes

Abstraction des données	Implémentation
Données d'instance	Objet métier (SDO)
Métadonnées d'instance	Graphique métier
Métadonnées type	Métadonnées d'entreprise, métadonnées type de l'objet métier
Services	Services de l'objet métier

Utilisation de l'infrastructure d'objets métier IBM

Comme nous l'avons vu précédemment, la structure d'un BO WebSphere Process Server correspond à une extension de la norme SDO. Par conséquent, les objets métier échangés entre les composants WebSphere Process Server correspondent à des instances de la classe `commonj.sdo.DataObject`. Toutefois, la structure d'un BO WebSphere Process Server ajoute plusieurs services et fonctions qui simplifient et enrichissent la fonctionnalité `DataObject` de base.

Pour faciliter la création et la manipulation d'objets métier, la structure d'un BO WebSphere étend les spécifications d'un objet SDO en fournissant un ensemble de services Java. Ces services font partie du package `com.ibm.websphere.bo` :

- **BOFactory** : Service clé qui indique plusieurs méthodes permettant de créer des instances d'objets métier.
- **BOXMLSerializer** : Indique plusieurs méthodes permettant de "développer" un objet métier depuis un flux ou d'écrire le contenu d'un objet métier, au format XML, dans un flux.
- **BOCopy** : Indique plusieurs méthodes permettant de copier des objets métier (sémantique "profonde" et "superficielle").
- **BODataObject** : Vous permet d'accéder aux aspects de l'objet de données d'un objet métier, comme le récapitulatif des modifications, le graphique métier et le récapitulatif de l'événement.
- **BOXMLDocument** : Avant-guichet du service qui vous permet de manipuler l'objet métier comme un document XML.

- **BOChangeSummary** et **BOEventSummary** : Facilite l'accès et la manipulation du récapitulatif des modifications et de la partie récapitulative d'un événement dans un objet métier.
- **BOEquality** : Service qui vous permet de déterminer si deux objets métier contiennent des informations identiques. Il prend en charge l'égalité profonde et superficielle.
- **BOType** et **BOTypeMetaData** : Ces services matérialisent des instances de `commonj.sdo.Type` et vous permettent de manipuler les métadonnées associées. Les instances `Type` peuvent ensuite être utilisées pour créer des objets métier "par type".
- **BOInstanceValidator** : Valide les données composant un objet métier, afin de vérifier s'il est conforme aux éléments XSD.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Architecture de composants de service

L'architecture SCA est un concept que vous pouvez implémenter de différentes manières. Elle n'exige aucune technologie, langage de programmation, protocole d'appel ou mécanisme de transport particulier. Les composants SCA sont décrits à l'aide du langage SCDL (Service Component Definition Language) qui est un langage basé sur XML.

Un composant SCA dispose des caractéristiques suivantes :

- Il encapsule un artefact d'implémentation qui contient la logique que le composant peut exécuter.
- Il offre une ou plusieurs interfaces.
- Il peut offrir une ou plusieurs références aux autres composants. La logique d'implémentation détermine si un composant affiche une référence. Si l'implémentation exige l'appel d'autres services, le composant SCA doit afficher une référence.

Les informations suivantes traitent essentiellement de l'implémentation de l'architecture SCA que WebSphere Process Server met à disposition et de l'outil WebSphere Integration Developer qui permet de créer et de combiner des composants SCA. WebSphere Process Server et WebSphere Integration Developer prennent en charge les artefacts d'implémentation suivants :

- Objets Java simples
- Processus métier
- Machines d'état métier
- Tâches utilisateur
- Règles métier
- Flux de médiation

L'architecture SCA distingue la logique métier de l'infrastructure afin que les programmeurs d'application puissent se consacrer à la résolution d'incidents métier. WebSphere Process Server d'IBM se base sur ce même principe. La figure 2 affiche le modèle architectural de WebSphere Process Server.

Une structure à base de composants pour tous les styles d'intégration.

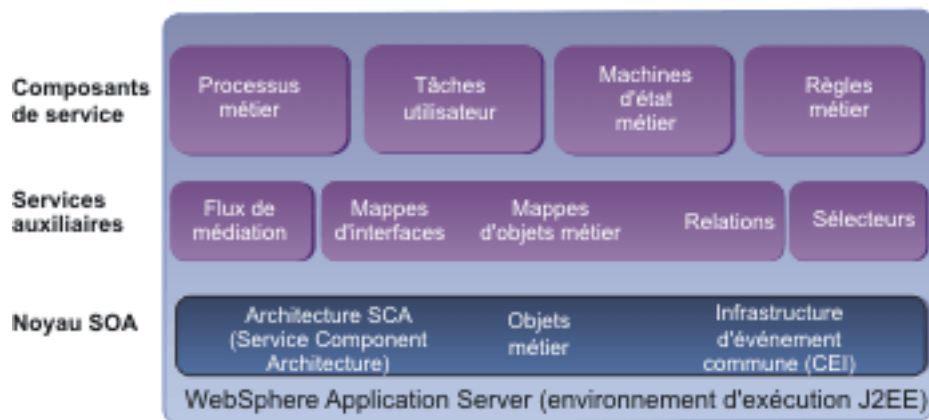


Figure 2. Structure à base de composants WebSphere Process Server

Dans l'environnement WebSphere, la structure SCA se base sur l'environnement d'exécution Java 2 Platform, Enterprise Edition (J2EE) de WebSphere Application Server. La structure générale de WebSphere Process Server se compose d'un noyau SOA, de services auxiliaires et de composants de service. La même structure dotée d'un sous-ensemble de cette capacité globale, axée plus spécifiquement sur les besoins de connectivité et d'intégration des applications de l'intégration métier, est disponible dans WebSphere Enterprise Service Bus.

L'interface d'un composant SCA, comme illustrée dans la figure 3, à la page 16, peut être représentée de l'une des manières suivantes :

- Une interface Java
- Un type de port WSDL (dans WSDL 2.0, le type de port correspond à une interface)

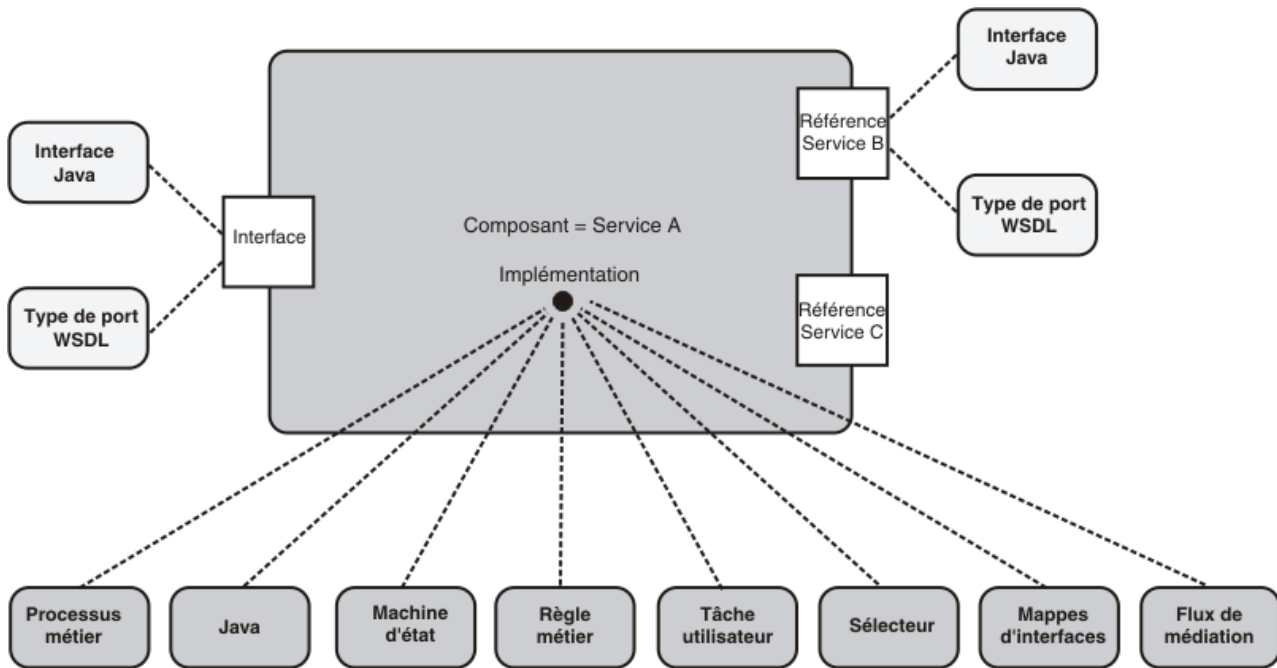


Figure 3. SCA dans WebSphere Process Server

Un module SCA est un groupe de composants connectés les uns aux autres après avoir directement relié les références et les implémentations. Dans WebSphere Integration Developer, chaque module SCA dispose d'un diagramme d'assemblage qui lui est associé et qui représente l'application métier intégrée contenant des composants SCA et des fils de connexion. L'une des principales missions du développeur d'intégration est de créer le diagramme d'assemblage en connectant les composants qui forment la solution. WebSphere Integration Developer fournit un éditeur d'assemblage graphique pour vous aider dans cette tâche. Lors de la création d'un diagramme d'assemblage, le développeur d'intégration dispose de deux moyens d'action :

- **Approche descendante** définit les composants, leurs interfaces et leurs interactions avant de créer l'implémentation. Le développeur d'intégration peut définir la structure du processus, identifier les composants requis et leurs types d'implémentation, puis générer un squelette d'implémentation.
- **Approche ascendante** associe des composants existants. Dans ce cas, le développeur d'intégration doit uniquement faire glisser et déposer les implémentations existantes dans le diagramme d'assemblage.

L'approche ascendante est plus couramment utilisée lorsque les clients disposent de services existants qu'ils veulent réutiliser et combiner. Lorsque vous souhaitez créer de nouveaux objets métier, vous utiliserez certainement l'approche descendante.

Modèle de programmation SCA : Principes

Le concept d'un *composant* logiciel est à la base du modèle de programmation SCA. Comme nous l'avons mentionné précédemment, un composant est une unité qui implémente une logique et qui la rend disponible aux autres composants via une interface. Un composant peut également exiger les services rendus disponibles par les autres composants. Dans ce cas, le composant affiche une *référence* pour ces services.

Dans SCA, chaque composant doit afficher au moins une interface. Le diagramme d'assemblage de la figure 4 dispose de trois composants : C1, C2 et C3. Chacun d'entre eux dispose d'une interface représentée par la lettre I entourée d'un cercle. Un composant peut également se référer à d'autres composants. Les références sont représentées par la lettre R entourée d'un carré. Les références et les interfaces sont ensuite liées dans un diagramme d'assemblage. En règle générale, le développeur d'intégration "résout" les références en les connectant aux interfaces de composants qui implémentent la logique requise.

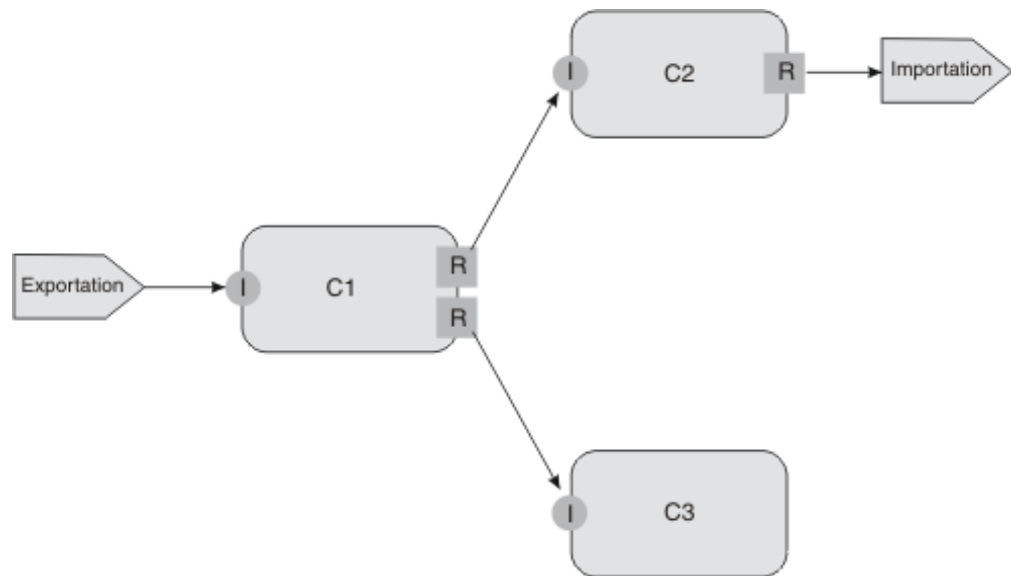


Figure 4. Diagramme d'assemblage

Appel de composants SCA

Pour fournir un accès aux services à appeler, le modèle de programmation SCA contient une classe *ServiceManager* qui permet aux développeurs de rechercher les services disponibles par nom. Voici un fragment de code Java type illustrant la recherche d'un service. La classe *ServiceManager* permet d'obtenir une référence au service *BOFactory* qui est un service fourni par le système :

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
    smgr.locateService("com/ibm/websphere/bo/BOFactory");
```

Remarque : Le package de la classe *ServiceManager* est `com.ibm.websphere.sca`.

Les développeurs peuvent utiliser un mécanisme similaire pour obtenir les références de leurs propres services en spécifiant le nom du service référence dans la méthode *locateService*. Après avoir obtenu la référence d'un service à l'aide de la classe *ServiceManager*, vous pouvez invoquer n'importe quelle opération disponible sur ce service indépendamment du protocole d'appel et du type d'implémentation.

Pour appeler des composants SCA, il existe trois styles d'appel différents :

- **Appel synchrone** : Lorsque vous utilisez ce style d'appel, l'appelant attend de façon synchrone que la réponse soit envoyée. Il s'agit du mécanisme d'appel classique.
- **Appel asynchrone** : Ce mécanisme permet à l'appelant d'appeler un service sans attendre l'envoi immédiat d'une réponse. Au lieu d'obtenir une réponse, l'appelant obtient un "ticket" qui peut être utilisé ultérieurement pour récupérer la réponse. L'appelant récupère la réponse en appelant une opération spéciale qui doit être fournie par l'appelé dans ce cas de figure.
- **Appel asynchrone avec rappel** : Ce style d'appel est identique au précédent, mais l'appelé est chargé de l'envoi de la réponse. L'appelant doit afficher une opération spéciale (opération de rappel) que l'appelé peut appeler lorsque la réponse est prête.

Importations

Parfois, les composants ou les fonctions disponibles sur des systèmes externes indiquent la logique métier, comme les applications existantes ou d'autres implémentations externes. Dans ce cas, le développeur d'intégration ne peut pas résoudre la référence en connectant une référence à un composant qui contient l'implémentation dont il/elle a besoin pour connecter la référence à un composant qui "pointe vers" l'implémentation externe. Ce composant est appelé *importation*. Lors de la définition d'une importation, vous devez spécifier la méthode d'accès à un service externe (emplacement), ainsi que le protocole d'appel.

Exportations

De même, si l'accès à votre composant s'effectue via des applications externes, ce qui est souvent le cas, vous devez le rendre accessible. Pour cela, utilisez un composant spécial qui affiche votre logique au "monde externe". Ce composant est appelé *exportation*. Il peut être appelé de façon synchrone ou asynchrone.

Références autonomes

Dans WebSphere Process Server, un module de service SCA est intégré comme fichier EAR J2EE qui contient plusieurs autres sous-modules J2EE. Les éléments J2EE, comme un fichier WAR, peuvent être intégrés au module SCA. Les artefacts autres que SCA, comme les JSP, peuvent également être intégrés à un module de service SCA. Ce dernier leur permet d'appeler des services SCA à l'aide du modèle de programmation client SCA grâce à un type de composant spécial appelé "référence autonome".

Le modèle de programmation SCA est hautement déclaratif. Les développeurs d'intégration peuvent configurer des aspects, comme le comportement transactionnel des appels, la propagation des données d'identification de sécurité, si un appel doit être synchrone ou asynchrone de façon déclarative, directement dans le diagramme d'assemblage. L'exécution SCA, non pas les développeurs, doit se charger de l'implémentation du comportement spécifié dans ces modificateurs. La flexibilité déclarative de SCA est l'une des fonctions les plus puissantes de ce modèle de programmation. Les développeurs peuvent se consacrer à implémenter la logique métier, au lieu de répondre aux aspects techniques, comme faciliter les mécanismes d'appel asynchrone. Tous ces aspects sont automatiquement gérés par l'exécution SCA.

Qualificateurs

Les qualificateurs régissent l'interaction entre un client de service et un service cible. Des qualificateurs peuvent être spécifiés dans les références de composant de service, les interfaces et les implémentations. Ils sont généralement externes à une implémentation.

Les différentes catégories de qualificateurs sont les suivantes :

- Transaction, qui spécifie la manière dont les contextes transactionnels sont gérés dans un appel SCA.
- Session d'activité, qui spécifie la manière dont les contextes de session d'activité sont propagés.
- Sécurité, qui spécifie les autorisations.
- La fiabilité asynchrone fournit des règles pour la distribution de messages asynchrones.

SCA autorise l'application de ces qualificateurs de qualité de service (QoS - Quality of Service) aux composants de façon déclarative (sans aucune programmation ou modification du code d'implémentation des services). Ceci peut être effectué dans WebSphere Integration Developer. En général, les qualificateurs QoS sont appliqués lorsque vous êtes prêt à envisager le déploiement d'une solution. Pour plus d'informations, reportez-vous à la section Référence du qualificateur pour la qualité de service.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Processus métier

Processus métier, plus précisément, processus métier basés sur BPEL, qui constituent la base des composants de service dans l'architecture SCA.

Qu'il s'agisse d'une simple approbation de commande ou d'un processus de fabrication complexe, les entreprises ont toujours utilisé des processus métier. Un *processus métier* est un ensemble d'activités, associées à une activité commerciale, où chacune est appelée dans un ordre spécifique pour atteindre un objectif commercial. Dans le cadre de l'intégration métier, un processus métier est défini avec un langage de balisage.

Ces processus métier peuvent avoir recours à d'autres services auxiliaires ou d'autres composants de service, comme des machines d'état métier, des tâches utilisateur, des règles métier ou des mappes de données. Puis, une fois développés, ces processus peuvent être rapidement terminés ou exécutés pendant une durée prolongée. Il arrive parfois que ces processus soient exécutés pendant plusieurs années.

Comme la plupart des composants utilisés dans J2EE, les processus métier sont exécutés dans un conteneur. Dans la plateforme IBM WebSphere, ce conteneur

spécial porte le nom de "Business Process Choreographer". Dans WebSphere Process Server, Business Process Choreographer est chargé de l'exécution des processus métier et des tâches humaines.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Tâches utilisateur

Une tâche utilisateur est un composant impliquant l'interaction des personnes et des services.

Certaines tâches utilisateur correspondent à des tâches à effectuer. Elles peuvent être lancées par un utilisateur ou par un service automatique. Les tâches utilisateur permettent notamment d'intégrer des activités dans des processus métier nécessitant une intervention humaine, par exemple le traitement et l'approbation manuels d'une exception. D'autres tâches utilisateur permettent d'appeler un service, ou de coordonner la collaboration entre les utilisateurs. Toutefois, quel que soit le mode de lancement d'une tâche, c'est une personne membre d'un groupe et à laquelle la tâche est affectée qui effectue le travail associé à la tâche.

Les tâches utilisateur sont affectées soit de manière statique, soit à l'aide de critères (par exemple rôle ou groupe) qui sont résolus lors de l'exécution à partir d'un répertoire d'utilisateurs. Dans d'autres cas, les données en entrée d'une tâche utilisateur ou d'un processus métier sont utilisées pour rechercher les personnes qualifiées pour travailler sur une tâche précise.

Concepts associés

Architecture et modèles d'intégration métier

Un projet d'intégration métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Création d'applications d'intégration métier

Intégration métier implique l'intégration d'applications, de données et de processus dans une ou plusieurs entreprises. L'intégration implique également le développement de processus, car il existe une certaine logique dans la séquence d'applications assemblées, afin de les intégrer. WebSphere Integration Developer permet de créer des applications d'intégration métier.

Cette section fournit des informations générales concernant le processus de développement d'un module d'intégration métier.

Le flux de développement classique des modules et des modules de médiation est le suivant :

1. Démarrez WebSphere Integration Developer et ouvrez un espace de travail.
2. Basculez vers la perspective Business Integration pour effectuer le développement.
3. Créez une bibliothèque pour stocker des artefacts, comme les objets métier et les interfaces qui sont partagés entre plusieurs modules.
4. Créez un nouveau module ou un module de médiation.
5. Créez des objets métier pour contenir des données d'application, par exemple : des données client ou les données d'une commande.
6. Créez l'interface et définissez les opérations d'interface pour chaque composant. L'interface détermine les données pouvant être transférées d'un composant à un autre.
7. Créez et implémentez des composants de service.
8. Créez un assemblage de modules en ajoutant des composants de service, des importations et des exportations au diagramme d'assemblage. Connectez les composants. Associez les importations et les exportations à un protocole.
9. Testez le module dans l'environnement de tests intégré.
10. Déployez le module dans WebSphere Process Server.
11. Partagez le module testé avec les autres membres de l'équipe en le plaçant dans un référentiel.

Concepts associés

Développement de solutions d'intégration métier

Cette section évoque les principes du modèle de programmation pour l'intégration métier. Elle présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier.

Chapitre 2. Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Avant de commencer

Cette tâche suppose qu'une analyse des exigences montre que l'implémentation d'un composant de service que d'autres modules utiliseront est avantageuse.

A propos de cette tâche

Après avoir analysé vos besoins, vous pouvez décider que la fourniture et l'utilisation de composants de service constituent un moyen efficace de traiter les informations. Si vous déterminez que des composants de service réutilisables présenteraient un avantage pour votre environnement, créez un module de service destiné à contenir les composants de service.

Procédure

1. Identifiez les composants de service que d'autres modules peuvent utiliser.
Une fois que vous avez identifié les composants de service, passez à la section «Développement de composants de service».
2. Identifiez des composants de service dans une application qui pourraient utiliser des composants de service dans d'autres modules de service.
Une fois que vous avez identifié les composants de service et les composants cibles correspondants, passez à la section «Appel de composants» ou «Appel dynamique des composants »
3. Reliez les composants client aux composants cible par le biais de connexions.

Concepts associés

Présentation du développement de modules

Un module est une unité de déploiement de base pour une application WebSphere Process Server. Un module peut contenir des composants, des bibliothèques et des modules de transfert utilisés par l'application.

Présentation de l'isolement des modules et des cibles

Lors du développement de modules, vous êtes amené à identifier des services exploités par plusieurs modules. Cette méthode d'optimisation des services permet de raccourcir le cycle de développement et de réduire les coûts. Lorsqu'un service est utilisé par de nombreux modules, il convient d'isoler les modules appelants de la cible afin que, dans le cas où la mise à niveau d'une cible est effectuée, le basculement sur le nouveau service puisse s'effectuer de manière transparente vis-à-vis du module appelant. La présente rubrique compare le modèle d'appel simple et le modèle d'appel isolé, en illustrant par un exemple les avantages offerts par la technique d'isolement. Bien que l'exemple décrit soit spécifique, il existe d'autres méthodes pour isoler les modules et les cibles.

Liaisons HTTP

La liaison HTTP permet de relier une architecture SOA à HTTP. Cela permet d'intégrer les applications HTTP existantes ou récemment développées aux environnements d'architecture SOA (Service Oriented Architecture).

Tâches associées

Développement de composants de service

Développez des composants de service pour fournir une logique réutilisable à plusieurs applications dans votre serveur.

Appel de composants

Les composants avec modules peuvent utiliser des composants sur n'importe quel noeud d'un cluster WebSphere Process Server.

Appel dynamique d'un composant

Lorsqu'un module appelle un composant qui a une interface de type de port WSDL (Web Service Descriptor Language), il doit appeler le composant de façon dynamique à l'aide de la méthode `invoke()`.

Présentation du développement de modules

Un module est une unité de déploiement de base pour une application WebSphere Process Server. Un module peut contenir des composants, des bibliothèques et des modules de transfert utilisés par l'application.

Le développement de modules consiste notamment à assurer que les composants, les modules de transfert et les bibliothèques (collections d'artefacts référencés par le module) requis par l'application sont disponibles sur le serveur de production.

WebSphere Integration Developer est l'outil principal de développement des modules destinés à être déployés sur WebSphere Process Server. Bien qu'il soit possible de développer des modules dans d'autres environnements, il est préférable d'utiliser WebSphere Integration Developer.

WebSphere Process Server prend en charge les modules de service métier et les modules de médiation. Les deux modules et les modules de médiation sont des types de module SCA (Service Component Architecture). Un module de communication permet les communications entre applications en transformant l'appel de service dans un format compris par la cible, en transmettant la demande à la cible et en renvoyant le résultat au module émetteur. Un module de service

métier implémente la logique d'un processus métier. Toutefois, un module peut également inclure la même logique de médiation qui peut être intégrée à un module de médiation.

Les sections suivantes décrivent comment implémenter et mettre à jour des modules sous WebSphere Process Server.

Composants

Les modules SCA contiennent des composants qui forment la structure de base permettant d'encapsuler la logique métier réutilisable. Les composants fournissent et consomment des services qui sont associés à des interfaces, des références et des implémentations. L'interface définit un contrat entre un composant de service et un composant appelant.

Avec WebSphere Process Server, un module peut soit exporter un composant de service pour qu'il soit utilisé par d'autres modules, soit importer un composant de service pour l'utiliser. Pour appeler un composant de service, un module appelant fait référence à l'interface du composant de service. Les références aux interfaces sont résolues à travers la configuration des références du module appelant à leurs interfaces respectives.

Pour développer un module, vous devez effectuer les activités suivantes :

1. Définir ou identifier des interfaces pour les composants du module.
2. Définir ou manipuler des objets métier utilisés par les composants.
3. Définir ou modifier des composants via leurs interfaces.

Remarque : Un composant est défini par le biais de son interface.

4. Facultatif : Exporter ou importer des composants de service.
5. Créez un fichier d'archive d'entreprise (EAR - Enterprise Archive) à déployer dans la phase d'exécution. Créez le fichier à l'aide de la fonction EAR d'exportation dans WebSphere Integration Developer ou de la commande `serviceDeploy`.

Types de développement

WebSphere Process Server comprend un modèle de programmation de composant afin de faciliter un paradigme de programmation orientée services. Pour utiliser ce modèle, un fournisseur exporte les interfaces d'un composant de service de façon à ce qu'un client puisse importer ces interfaces et utiliser le composant de service comme s'il était local. Un développeur utilise soit des interfaces fortement typées, soit des interfaces dynamiquement typées pour implémenter ou appeler le composant de service. Les interfaces et leurs méthodes sont décrites dans la section Références de ce centre de documentation.

Après avoir installé des modules de service sur vos serveurs, vous pouvez utiliser la console d'administration pour modifier le composant cible pour une référence d'une application. La nouvelle cible doit accepter le même type d'objet métier et effectuer la même opération que ce que la référence de l'application demande.

Remarques concernant le développement de composants de service

Lorsque vous développez un composant de service, posez-vous les questions suivantes :

- Ce composant de service va-t-il être exporté et utilisé par un autre module ?
Si c'est le cas, assurez-vous que la définition portant sur le composant va pouvoir être utilisée par un autre module.
- L'exécution de ce composant de service prendra-t-elle relativement longtemps ?
Si c'est le cas, envisagez d'implémenter une interface asynchrone pour le composant de service.
- Est-ce avantageux de décentraliser le composant de service ?
Si c'est le cas, envisagez de placer une copie du composant de service dans un module de service qui est déployé sur un cluster de serveurs afin de bénéficier du traitement parallèle.
- L'application nécessite-t-elle un mélange de ressources à une phase et à deux phases ?
Si c'est le cas, assurez-vous d'activer le support du dernier participant pour l'application.

Remarque : Si vous créez votre application à l'aide de WebSphere Integration Developer ou créez le fichier EAR installable à l'aide de la commande `serviceDeploy`, ces outils activent automatiquement le support pour l'application. Consultez la rubrique consacrée à l'«utilisation de ressources de validation à une phase et de ressources de validation à deux phases dans la même transaction» dans le centre de documentation de WebSphere Application Server Network Deployment.

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Développement de composants de service

Développez des composants de service pour fournir une logique réutilisable à plusieurs applications dans votre serveur.

Avant de commencer

Cette tâche suppose que vous avez déjà développé et identifié le traitement qui est utile pour plusieurs modules.

A propos de cette tâche

Plusieurs modules peuvent utiliser un composant de service. L'exportation d'un composant de service rend celui-ci disponible pour les autres modules qui se réfèrent à lui par le biais d'une interface. Cette tâche explique comment compiler le composant de service de manière à ce que d'autres modules puissent l'utiliser.

Remarque : Un composant de service unique peut contenir plusieurs interfaces.

Procédure

1. Définir l'objet de données permettant de déplacer des données entre l'appelant et le composant de service.

L'objet de données et son type font partie de l'interface entre les appelants et le composant de service.

2. Définir une interface que les appelants utiliseront pour référencer le composant de service.

La définition de cette interface nomme le composant de service et répertorie toutes les méthodes disponibles dans ce composant de service.

3. Générer la classe implémentant l'appel du service.
4. Développer l'implémentation de la classe générée.
5. Sauvegarder les interfaces et les implémentations du composant dans des fichiers dotés d'une extension .java.
6. Empaqueter le module de service et les ressources nécessaires dans un fichier JAR.

Reportez-vous à la section «Déploiement d'un module sur un serveur de production» de ce centre de documentation pour obtenir une description des étapes 6 à 8.

7. Exécuter la commande `serviceDeploy` pour créer un fichier EAR installable contenant l'application.
8. Installer l'application sur le noeud du serveur.
9. Facultatif : Configurer les connexions entre les appelants et le composant de service correspondant, en cas d'appel d'un composant de service d'un autre module de service.

La section «Administration» de ce centre de documentation explique comment configurer ces connexions.

Exemples de développement de composants

Cet exemple montre un composant de service synchrone qui implémente une méthode unique, `CustomerInfo`. La première section définit l'interface du composant de service qui implémente une méthode appelée `getCustomerInfo`.

```
public interface CustomerInfo {  
    public Customer getCustomerInfo(String  
customerID);  
}
```

Le bloc de code suivant implémente le composant de service.

```
public class CustomerInfoImpl implements CustomerInfo {  
    public Customer getCustomerInfo(String customerID) {  
        Customer cust = new Customer();  
  
        cust.setCustNo(customerID);  
        cust.setFirstName("Victor");  
        cust.setLastName("Hugo");  
        cust.setSymbol("IBM");  
        cust.setNumShares(100);  
        cust.setPostalCode(10589);  
        cust.setErrorMsg("");  
  
        return cust;  
    }  
}
```

x

La section suivante est l'implémentation de la classe associée à `StockQuote`.

```
public class StockQuoteImpl implements StockQuote {  
  
    public float getQuote(String symbol) {
```

```
        return 100.0f;
    }
}
```

Que faire ensuite

Appelez le service.

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Appel de composants

Les composants avec modules peuvent utiliser des composants sur n'importe quel noeud d'un cluster WebSphere Process Server.

Avant de commencer

Avant d'appeler un composant, assurez-vous que le module qui contient le composant est installé sur WebSphere Process Server.

A propos de cette tâche

Les composants peuvent utiliser n'importe quel composant de service disponible dans un cluster WebSphere Process Server en utilisant le nom du composant et en transférant le type de données qu'attend le composant. L'appel d'un composant dans cet environnement implique la localisation, puis la création de la référence vers le composant nécessaire.

Remarque : Un composant de module peut appeler un composant à l'intérieur du même modèle : cette opération s'appelle un appel intra-module. Implémentez les appels externes (appels inter-modules) en exportant l'interface dans le composant fournisseur et en important l'interface dans le composant appelant.

Important : Lors de l'appel d'un composant résidant sur un serveur autre que le serveur sur lequel s'exécute le module appelant, vous devez apporter des modifications de configuration à ces deux serveurs. Les configurations requises dépendent du mode d'appel du composant (appel asynchrone ou appel synchrone). La procédure de configuration spécifique des serveurs d'applications est décrite dans les tâches associées.

Procédure

1. Déterminer les composants requis par le module appelant.
Notez le nom de l'interface dans un composant et le type de données dont l'interface a besoin.
2. Définir un objet de données.
Bien que l'entrée ou le retour puisse être une classe Java, l'idéal est un objet de données de service.
3. Localiser le composant.
 - a. Utiliser la classe ServiceManager pour obtenir les références disponibles pour le module appelant.

- b. Utiliser la méthode `locateService()` pour trouver le composant.
En fonction du composant, l'interface peut être soit un type de port WSDL (Web Service Descriptor Language), soit une interface Java.
4. Appeler le composant de manière synchrone.
Vous pouvez soit appeler le composant par le biais d'une interface Java, soit utiliser la méthode `invoke()` pour appeler le composant de manière dynamique.
5. Traiter le retour.
Le composant peut générer une exception, aussi le client doit-il être capable de traiter cette possibilité.

Exemple d'appel d'un composant

L'exemple suivant permet de créer une classe `ServiceManager`.

```
ServiceManager serviceManager = new ServiceManager();
```

Cet exemple utilise la classe `ServiceManager` pour obtenir une liste de composants à partir d'un fichier contenant les références des de composants.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

Le code suivant localise un composant qui implémente l'interface Java `StockQuote`.

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

Le code suivant localise un composant qui implémente soit une interface Java, soit une interface de type de port WSDL. Le module appelant utilise l'interface `Service` afin d'interagir avec le composant.

Conseil : Si le composant implémente une interface Java, il peut être appelé à l'aide de l'interface ou de la méthode `invoke()`.

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

L'exemple suivant illustre le code `MyValue`, qui appelle un autre composant.

```
public class MyValueImpl implements MyValue {

    public float myValue throws MyValueException {

        ServiceManager serviceManager = new ServiceManager();

        // variables
        Customer customer = null;
        float quote = 0;
        float value = 0;

        // invoke
        CustomerInfo cInfo = (CustomerInfo)serviceManager.locateService("customerInfo");
        customer = cInfo.getCustomerInfo(customerID);

        if (customer.getErrorMsg().equals("")) {

            // invoke
            StockQuote sQuote =
            (StockQuote)serviceManager.locateService("stockQuote");
            Ticket ticket = sQuote.getQuote(customer.getSymbol());
            // ... do something else ...
            quote = sQuote.getQuoteResponse(ticket, Service.WAIT);

            // assign
            value = quote * customer.getNumShares();
        }
    }
}
```

```

    } else {
        // throw
        throw new MyValueException(customer.getErrorMsg());
    }
    // reply
    return value;
}
}

```

Que faire ensuite

Configurez les connexions entre les références de module appelant et les interfaces de composant.

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Appel dynamique d'un composant

Lorsqu'un module appelle un composant qui a une interface de type de port WSDL (Web Service Descriptor Language), il doit appeler le composant de façon dynamique à l'aide de la méthode `invoke()`.

Avant de commencer

Cette tâche suppose qu'un composant appelant appelle un composant de façon dynamique.

A propos de cette tâche

Avec une interface de type de port WSDL, un composant appelant doit utiliser la méthode `invoke()` pour appeler le composant. Un composant appelant peut également appeler un composant ayant une interface Java de cette façon.

Procédure

1. Déterminez le module qui contient le composant nécessaire.
2. Déterminez le tableau dont le composant a besoin.
Le tableau d'entrée peut être de l'un des trois types suivants :
 - Des types Java haut de casse primitifs ou des tableaux de ce type
 - Des classes Java ordinaires ou des tableaux de ces classes
 - Service Data Objects (SDO)
3. Définissez un tableau pour contenir la réponse du composant.
Le tableau de réponse peut être des mêmes types que le tableau d'entrée.
4. Utilisez la méthode `invoke()` pour appeler le composant nécessaire et transférer l'objet tableau vers le composant.
5. Traitez le résultat.

Exemples d'appel dynamique d'un composant

Dans l'exemple suivant, un module utilise la méthode `invoke()` pour appeler un composant qui utilise des types de données Java haut de casse primitives.

```

Service service = (Service)serviceManager.locateService("multiParamInf");

Reference reference = service.getReference();

OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

Type t = methodMultiType.getInputType();

BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

DataObject paramObject = boFactory.createbyType(t);

paramObject.set(0,"input1")
paramObject.set(1,"input2")
paramObject.set(2,"input3")

service.invoke("methodMultiParamater",paramObject);

```

L'exemple suivant utilise la méthode d'appel via une interface de port WSDL configurée en tant que cible.

```

Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
dob.setString("attribute1", stringArg);

DataObject wrapBo = factory.createByElement
    ("http://MultiCallWSServerOne/wsd1/ServerOneInf", "methodOne");
wrapBo.set("input1", dob); //wrapBo encapsule tous les paramètres de methodOne
wrapBo.set("input2", "XXXX");
wrapBo.set("input3", "yyyy");

DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);

```

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Présentation de l'isolement des modules et des cibles

Lors du développement de modules, vous êtes amené à identifier des services exploités par plusieurs modules. Cette méthode d'optimisation des services permet de raccourcir le cycle de développement et de réduire les coûts. Lorsqu'un service est utilisé par de nombreux modules, il convient d'isoler les modules appelants de la cible afin que, dans le cas où la mise à niveau d'une cible est effectuée, le basculement sur le nouveau service puisse s'effectuer de manière transparente vis-à-vis du module appelant. La présente rubrique compare le modèle d'appel simple et le modèle d'appel isolé, en illustrant par un exemple les avantages offerts par la technique d'isolement. Bien que l'exemple décrit soit spécifique, il existe d'autres méthodes pour isoler les modules et les cibles.

Modèle d'appel simple

Lors du développement d'un module, vous pouvez être amené à utiliser des services situés dans d'autres modules. Pour ce faire, vous devez importer le service dans le module, puis appeler ce service. Le service importé est «connecté» au service exporté via l'autre module, soit sous WebSphere Integration Developer, soit par l'établissement d'une liaison avec le service via la console d'administration. Le modèle d'appel simple illustre cette configuration.

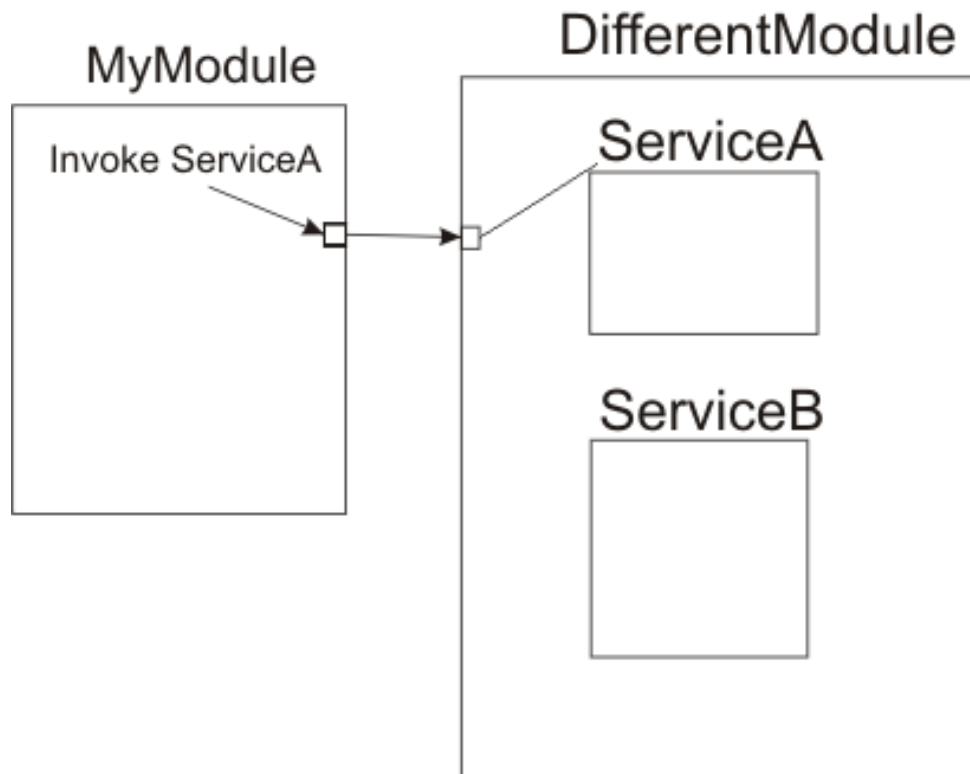


Figure 5. Modèle d'appel simple

Modèle d'appel isolé

Pour changer la cible d'appel sans impliquer l'arrêt des modules d'appel, vous pouvez isoler ces derniers de la cible concernée par l'appel. Ceci permet aux modules de poursuivre le traitement durant le changement de cible, puisque le changement affecte non pas le module lui-même, mais la cible située en aval. La figure Exemple d'isolement d'applications indique comment l'isolement permet de modifier la cible sans influencer sur l'état du module appelant.

Exemple d'isolement d'applications

Lorsque le modèle d'appel simple est appliqué, l'appel d'un même service par plusieurs modules équivaut pratiquement à un Appel de service unique par des applications multiples. Les modules MODA, MODB et MODC appellent conjointement CalculateFinalCost.

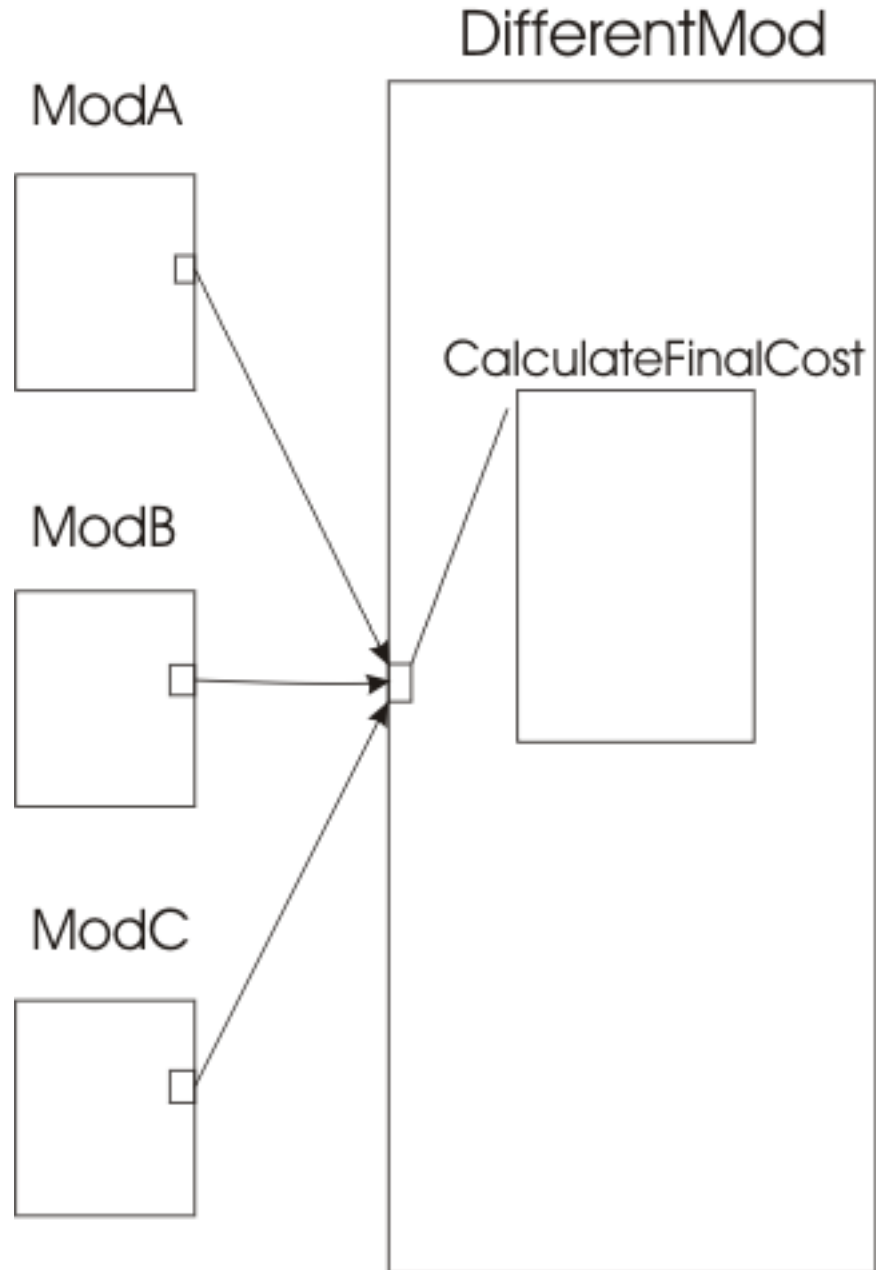


Figure 6. Appel de service unique par des applications multiples

Le service fourni par CalculateFinalCost nécessite une mise à jour, de sorte que les nouveaux coûts soient reflétés dans tous les modules exploitant ce service. L'équipe de développement met au point et teste un nouveau service (UpdatedCalculateFinal) visant à incorporer les modifications. Le nouveau service est dès lors prêt à entrer en phase de production. Si aucun isolement n'est effectué, vous devez mettre à jour l'ensemble des modules appelant CalculateFinalCost, afin de définir l'appel de UpdateCalculateFinal. Grâce à l'isolement, la seule modification nécessaire porte sur la liaison entre le module tampon et la cible.

Remarque : En utilisant cette méthode pour modifier le service, vous pouvez continuer à fournir le service d'origine aux autres modules ayant besoin de l'exploiter.

L'isolement permet de créer un module tampon entre les applications et la cible (voir Modèle d'appel isolé du service UpdateCalculateFinal).

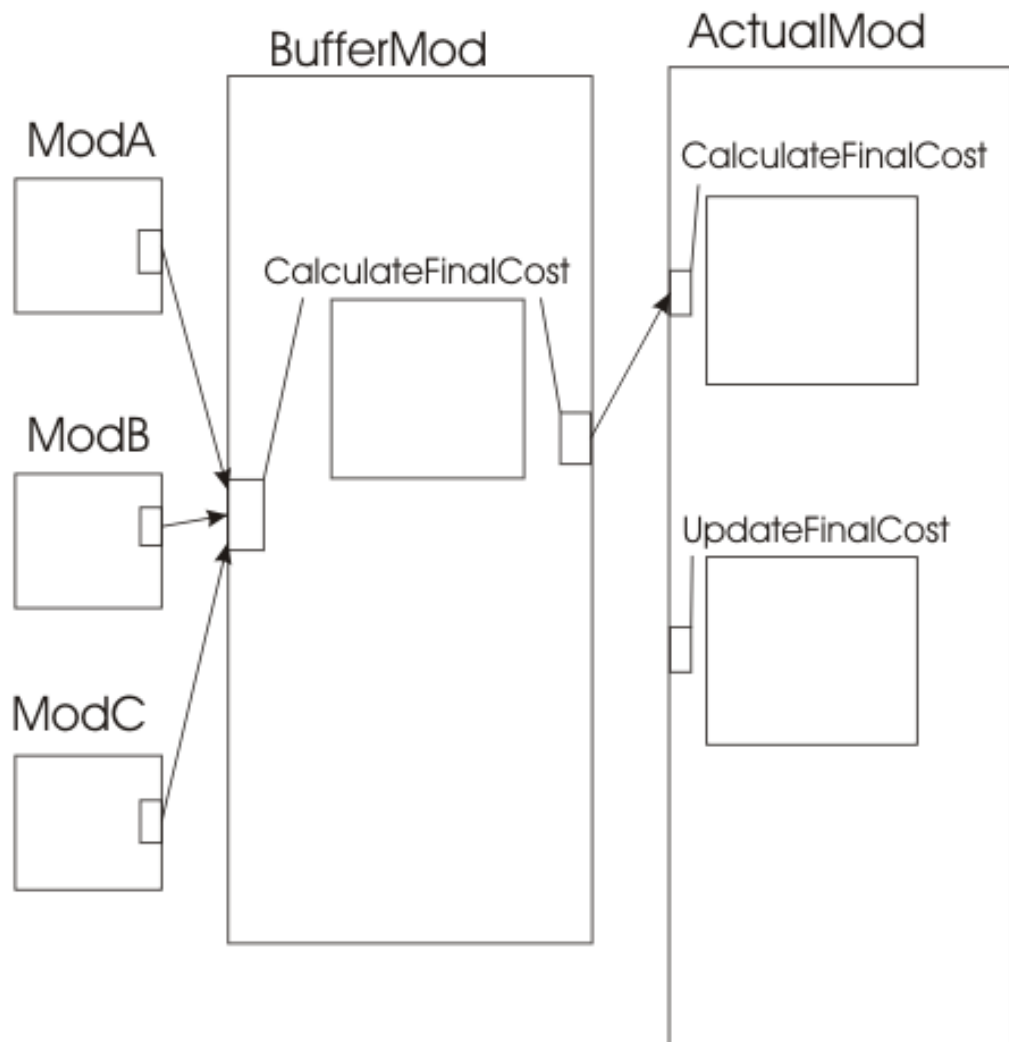


Figure 7. Modèle d'appel isolé du service UpdateCalculateFinal

Suivant ce modèle, les modules d'appel restent inchangés, la seule modification portant sur la liaison entre l'interface d'importation du module tampon et la cible (voir Modèle d'appel isolé du service UpdatedCalculateFinal).

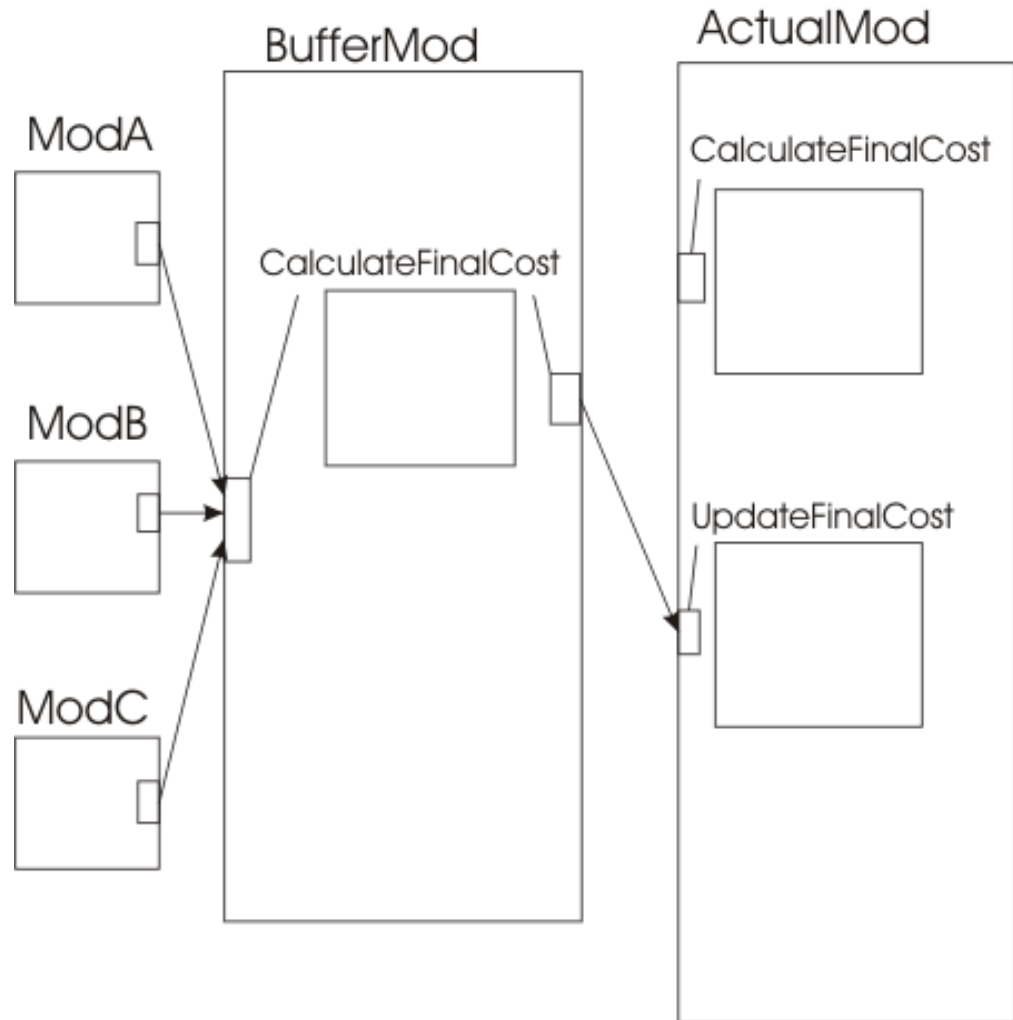


Figure 8. Modèle d'appel isolé du service UpdatedCalculateFinal

Si le module tampon procède à l'appel synchrone de la cible, le résultat renvoyé vers l'application d'origine lors du redémarrage du module tampon (qu'il s'agisse d'un module de médiation ou d'un service pour module métier) provient de la nouvelle cible. En cas d'appel asynchrone de la cible par le module tampon, les résultats renvoyés vers l'application d'origine proviendront de la nouvelle cible dès l'appel suivant.

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Liaisons HTTP

La liaison HTTP permet de relier une architecture SOA à HTTP. Cela permet d'intégrer les applications HTTP existantes ou récemment développées aux environnements d'architecture SOA (Service Oriented Architecture).

De plus, un réseau d'environnements d'exécution SCA peuvent communiquer via une infrastructure HTTP existante.

La liaison HTTP offre plusieurs fonctions HTTP :

- Dans les messages présentés sur les composants de communication, le format HTTP et les informations de l'en-tête sont conservés. Ce mode d'affichage correspond à ce que les programmeurs d'applications HTTP, les utilisateurs et les administrateurs ont l'habitude de voir.
- Une structure de liaisons de données existante développée selon les conventions HTTP permet de mapper les messages SCA aux en-têtes de message HTTP et aux corps des messages.
- Les importations et les exportations peuvent être configurées de façon à prendre en charge un ensemble de fonctions HTTP courantes.
- Lorsque vous installez un module SCA contenant des importations ou des exportations HTTP, l'environnement d'exécution est automatiquement configuré pour permettre la connectivité vers HTTP.

Vous trouverez des instructions détaillées sur la création d'importations et d'exportations HTTP dans le centre de documentation dans **WebSphere Integration Developer > Développement des applications d'intégration > Liaisons de données HTTP**.

Tâches associées

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Chapitre 3. Guides et techniques de programmation

Cette section comprend des guides et des exemples de programmation.

Les sous-rubriques ci-après fournissent des informations pour la programmation de divers composants, applications et solutions d'intégration métier.

Important : Voir la section Référence du centre de documentation pour obtenir des détails sur les API (interfaces de programme d'application) et les SPI a(interfaces de programmation de système) qui sont prises en charge par WebSphere Process Server et WebSphere Enterprise Service Bus.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Validation de document XML

Il est possible de valider les documents XML et les objets métier à l'aide du service de validation.

Propagation d'en-tête de protocole à partir de liaisons d'exportation non SCA

Le service de contexte est chargé de la propagation du contexte (y compris les en-têtes de protocole comme l'en-tête JMS et le contexte utilisateur comme l'ID de compte) tout au long du chemin d'appel SCA (Service Component Architecture). Le service de contexte offre un ensemble d'API et de paramètres configurables.

Gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Les sous-rubriques ci-après contiennent des informations pour la programmation d'objets métier et SDO.

Concepts associés

Guides et techniques de programmation

Cette section comprend des guides et des exemples de programmation.

Tableaux dans les objets métier

Vous pouvez définir des tableaux pour un élément dans un objet métier afin que cet élément puisse contenir plus d'une instance de données.

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Objets métier : renforcement du schéma et prise en charge du schéma industriel

La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Règles en exécution de la conversion de Java en objets SDO

Pour une substitution correcte du code généré ou l'identification des éventuelles exceptions d'exécution liées aux conversions de Java en SDO (Service Data Object), il est important de bien comprendre les règles en jeu. La plus grande partie des conversions se font directement, mais il existe des cas complexes où l'environnement d'exécution offre les meilleures possibilités de conversion du code généré.

Tâches associées

Remplacement d'une conversion d'objet SDO en Java

Il se peut que la conversion d'un objet SDO (Service Data Object) en objet de type Java effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation par défaut par celle de votre choix.

Remplacement de l'implémentation d'architecture SCA générée

Il se peut que la conversion de code Java en objet SDO (Service Data Object) effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation d'architecture SCA (Service Component Architecture) par défaut par celle de votre choix.

Tableaux dans les objets métier

Vous pouvez définir des tableaux pour un élément dans un objet métier afin que cet élément puisse contenir plus d'une instance de données.

Vous pouvez utiliser une Liste pour créer un tableau pour un seul élément nommé dans un objet métier. Vous pourrez ainsi utiliser cet élément pour contenir des instances multiples de données. Par exemple, vous pouvez utiliser un tableau pour stocker plusieurs numéros de téléphone dans un élément nommé `telephone` et défini en tant que chaîne dans l'encapsuleur d'objet métier. Vous pouvez également définir la taille du tableau en précisant le nombre d'instances de données. Pour cela, vous utiliserez la valeur `maxOccurs`. L'exemple de code suivant montre comment créer un tel tableau comportant trois instances de données pour cet élément :

```
<xsd:element name="telephone" type="xsd:string" maxOccurs="3"/>
```

Cela va créer un index pour l'élément `telephone` qui peut contenir jusqu'à trois instances de données. Vous pouvez également utiliser la valeur `minOccurs` si vous envisagez d'avoir un élément dans le tableau.

Le tableau créé se compose de deux éléments :

- son contenu
- le tableau lui-même.

Pour créer ce tableau, cependant, vous devez effectuer une opération intermédiaire consistant à définir un encapsuleur. Celui-ci remplace en effet la propriété de l'élément par un objet tableau. Dans l'exemple ci-dessus, vous pouvez créer un objet `ArrayOfTelephone` pour définir l'élément `telephone` en tant que tableau. L'exemple de code suivant indique comment accomplir cette tâche :

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Customer">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="ArrayOfTelephone" type="ArrayOfTelephone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="ArrayOfTelephone">
      <xsd:sequence maxOccurs="3">
        <xsd:element name="telephone" type="xsd:string" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
```

L'élément `telephone` apparaît maintenant en tant qu'enfant de l'objet encapsuleur `ArrayOfTelephone`.

Vous remarquerez que dans l'exemple ci-dessus, l'élément `telephone` comprend la propriété `nillable`. Vous pouvez définir cette valeur sur `true` si vous voulez que certains éléments dans l'indice de tableau ne contiennent aucune donnée. L'exemple de code suivant indique comment les données d'un tableau peuvent être représentées :

```
<Customer>
  <name>Bob</name>
  <ArrayOfTelephone>
    <telephone>111-1111</telephone>
    <telephone xsi:nil="true"/>
    <telephone>333-3333</telephone>
  </ArrayOfTelephone>
</Customer>
```

Dans ce cas, le premier et le troisième éléments dans l'indice de tableau de l'élément `telephone` contiennent des données contrairement au deuxième. Si vous n'aviez pas utilisé la propriété `nillable` pour l'élément `telephone`, vous auriez alors les deux premiers éléments qui contiennent des données.

Vous pouvez utiliser les API de séquence Service Data Object (SDO) dans WebSphere Process Server comme alternative au traitement des séquences dans les tableaux d'objet métier. L'exemple de code suivant permet de créer un tableau pour l'élément `telephone` avec des données identiques à celles indiquées plus haut :

```
DataObject customer = ...
customer.setString("name", "Bob");

DataObject tele_array = customer.createDataObject("ArrayOfTelephone");
Sequence seq = tele_array.getSequence(); // The array is sequenced
seq.add("telephone", "111-1111");
seq.add("telephone", null);
seq.add("telephone", "333-3333");
```

Vous pouvez renvoyer les données d'un indice de tableau d'élément donné en utilisant un code semblable à l'exemple ci-dessous :

```
String tele3 = tele_array.get("telephone[3]"); // tele3 = "333-3333"
```

Dans cet exemple, la chaîne tele3 va renvoyer les données "333-3333".

Vous pouvez remplir les éléments données du tableau dans l'index en utilisant une largeur fixe ou des données délimitées placées dans une file d'attente de messages JMS ou MQ. Vous pouvez également accomplir cette tâche en utilisant un fichier texte à plat contenant les données correctement formatées.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Vous pouvez créer des objets métier imbriqués à partir d'un objet métier parent sans écrire de code détaillant les objets enfant intermédiaires. Par exemple, vous pouvez créer un objet métier imbriqué deux niveaux sous l'objet parent sans avoir à définir un objet métier dépendant entre les deux, c'est-à-dire un niveau sous l'objet parent. La fonction `setWithCreate` permet d'accomplir cette tâche pour :

- une seule instance
- plusieurs instances
- une valeur de caractère générique
- un groupe de modèles

Les rubriques suivantes expliquent comment procéder.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Tâches associées

Instance unique d'un objet métier imbriqué

La fonction `setWithCreate` permet de créer une instance unique d'un objet métier imbriqué.

Création de plusieurs instances d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des instances multiples d'un objet métier imbriqué.

Utilisation d'un objet métier imbriqué défini par un caractère générique

Vous pouvez spécifier le type `xsd:any` dans un objet parent pour indiquer un objet enfant, mais uniquement si cet objet enfant existe déjà.

Utilisation des objets métier dans les groupes de modèles

Vous devriez utiliser les modèles de chemin de groupe de modèles lorsque vous utilisez des objets métier imbriqués faisant partie d'un groupe de modèles.

Instance unique d'un objet métier imbriqué

La fonction `setWithCreate` permet de créer une instance unique d'un objet métier imbriqué.

Avant de commencer

L'exemple suivant montre comment vous devriez normalement créer du code pour un objet intermédiaire (enfant) à partir d'un objet de niveau plus élevé (parent) afin de créer un objet de troisième niveau (grand-enfant). Le fichier XSD aurait la forme suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

A propos de cette tâche

Si vous utilisiez la méthode traditionnelle "descendante" pour définir les données d'objet métier, il vous faudrait traiter le code suivant précisant les objets enfant et grand-enfant avant de définir les données dans l'objet grand-enfant :

```
DataObject parent = ...
DataObject child = parent.createDataObject("child");
DataObject grandchild = child.createDataObject("grandChild");
grandchild.setString("name", "Bob");
```

Il existe une méthode plus efficace qui consiste à utiliser la fonction `setWithCreate`. Celle-ci permet en effet de définir simultanément l'objet grand-enfant et ses données, sans avoir à préciser l'objet enfant intermédiaire. L'exemple de code suivant indique comment accomplir cette tâche :

```
DataObject parent = ...
parent.setString("child/grandchild/name", "Bob");
```

Résultats

L'objet métier de niveau inférieur est défini sans avoir à définir l'objet métier de niveau intermédiaire. Une exception est émise toutefois si le chemin est incorrect.

Concepts associés

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Création de plusieurs instances d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des instances multiples d'un objet métier imbriqué.

Avant de commencer

L'exemple suivant représente un fichier XSD contenant des objets imbriqués se trouvant un niveau (enfant) et deux niveaux (petit-enfant) sous l'objet métier supérieur (parent) :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child" maxOccurs="5"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Vous remarquerez que l'objet parent peut avoir jusqu'à cinq objets enfant, comme l'indique la valeur `maxOccurs`.

A propos de cette tâche

Vous pouvez créer une liste avec une règle plus rigoureuse ne permettant pas que des séquences soient absentes d'un tableau. Vous pouvez utiliser la méthode `setWithGet` et, en même temps, préciser les données qui apparaîtront dans un élément de l'index de liste particulier :

```
DataObject parent = ...
parent.setString("child[3]/grandchild/name", "Bob");
```

Dans cet exemple, vous obtenez un tableau de taille trois, mais les valeurs des éléments de l'index de liste `child[1]` et `child[2]` ne sont pas définies. Vous voudrez peut-être leur attribuer la valeur `null` ou la valeur d'une donnée associée. Dans le scénario ci-dessus, une exception sera émise car la valeur des deux premiers éléments du tableau n'est pas définie.

Vous pouvez remédier à cette situation en définissant ces valeurs dans l'index de la liste. Si l'élément de l'index fait référence à un élément existant du tableau et que la valeur de cet élément n'est pas `null` (c'est-à-dire qu'il contient des données), celui-ci sera utilisé. Si sa valeur est `null`, il sera créé puis utilisé. Si l'index de la liste est plus grand que la taille de celle-ci, une nouvelle valeur sera créée et ajoutée. L'exemple suivant illustre le fonctionnement dans une liste de taille deux, où l'élément `child[1]` est désigné comme `null` et l'élément `child[2]` contient des données :

```
DataObject parent = ...
// child[1] = null
// child[2] = existing Child
// Ce code fonctionne car l'élément child[1] est null et sera créé.
```

```

parent.setString("child[1]/grandchild/name", "Bob");

// Ce code fonctionne car l'élément child[2] existe et sera utilisé.
parent.setString("child[2]/grandchild/name", "Dan");

// Ce code fonctionne car la liste enfant est de taille 2 et l'ajout
// d'un élément de liste supplémentaire va accroître la taille de la liste.
parent.setString("child[3]/grandchild/name", "Sam");

```

Résultats

Vous avez remplacé les valeurs des deux éléments existants et ajouté un troisième à l'index de la liste. Néanmoins, si vous ajoutez un autre élément qui n'est pas de taille quatre, ou qui est plus grand que la taille précisée dans `maxOccurs`, une exception sera émise. La règle plus rigoureuse de cette méthode est démontrée dans l'exemple suivant.

Remarque : Le code qui suit est ajouté au code utilisé ci-dessus :

```

// Ce code entraîne une exception car la liste est de taille 3
// et vous n'avez pas créé d'élément pour augmenter la taille à 4.
parent.setString("child[5]/grandchild/name", "Billy");

```

Concepts associés

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Utilisation d'un objet métier imbriqué défini par un caractère générique

Vous pouvez spécifier le type `xsd:any` dans un objet parent pour indiquer un objet enfant, mais uniquement si cet objet enfant existe déjà.

A propos de cette tâche

La fonction `setWithCreate` utilisée pour définir des objets métier imbriqués pour une seule ou plusieurs instances ne fonctionne pas si vous utilisez la valeur générique `xsd:any` dans l'objet de données de service. L'exemple suivant illustre cette situation :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>

```

Résultats

Une exception sera émise si l'objet données enfant n'existe pas.

Concepts associés

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Utilisation des objets métier dans les groupes de modèles

Vous devriez utiliser les modèles de chemin de groupe de modèles lorsque vous utilisez des objets métier imbriqués faisant partie d'un groupe de modèles.

A propos de cette tâche

Les groupes de modèles utilisent la balise `xsd:choice` que vous pouvez utiliser pour créer des objets métier à partir d'un objet métier parent. Eclipse Modeling Framework (EMF), cependant, peut entraîner des conflits de dénomination qui peuvent alors générer une exception. L'exemple suivant illustre comment une telle situation peut se produire :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Vous remarquerez qu'il peut y avoir plusieurs instances des éléments "child1" et "child2",

Vous devez utiliser les modèles de chemin Service Data Object (SDO) pour les groupes de modèles pour résoudre ces conflits.

Résultats

Vous obtiendrez des tableaux qui utilisent le modèle de chemin SDO utilisé pour traiter les groupes de modèles, comme indiqué dans l'exemple de code ci-dessous :

```
set("child1/grandchild/name", "Bob");
```

```
set("child11/grandchild/name", "Joe");
```

Concepts associés

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Objets métier : renforcement du schéma et prise en charge du schéma industriel

La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Ce manuel contient des informations sur les incidents relatifs à la gestion des constructions de schéma pour certaines fonctions. Pour obtenir des informations sur la procédure de définition d'un objet métier, des instructions de développement d'objets métier et sur l'utilisation des API de programmation d'objets métier, reportez-vous aux articles de la section "Informations connexes" ci-dessous.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Différenciation d'éléments portant le même nom

Vous devez donner des noms uniques aux éléments et attributs d'objet de données.

Différenciation de propriétés portant le même nom

Lorsque plusieurs XSD avec le même espace de nom définissent des types portant le même nom, un type incorrect peut être accidentellement référencé.

Résolution de noms de propriétés contenant des points

Les noms des propriétés dans un XSD peuvent contenir un point (".") comme un des nombreux caractères valides, alors que, dans un SDO, ils sont également utilisés pour montrer l'indexation dans une propriété à cardinalité multiple. Dans certaines situations, ceci peut entraîner des problèmes de résolution.

Utilisation de l'objet de séquence pour définir l'ordre des données

Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Utilisation de AnySimpleType pour les types simples

AnySimpleType est traité de la même manière que les autres types simples (chaîne, int, booléen, etc.) par les API SDO.

Utilisation de AnyType pour les types complexes

La balise anyType est traitée de la même manière que les autres types complexes par les API SDO.

Utilisation de la balise Any pour définir des éléments globaux de types complexes

Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

Utilisation de AnyAttribute pour définir les attributs globaux de types complexes

La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Information associée

➡ Web Services Description Language (WSDL) 1.1

➡ Introduction aux objets SDO (Service Data Objects)

➡ Examen des objets métier dans WebSphere Process Server

Différenciation d'éléments portant le même nom

Vous devez donner des noms uniques aux éléments et attributs d'objet de données.

Dans l'infrastructure SDO, les éléments et les attributs sont créés en tant que propriétés. Dans les exemples de code suivants, les XSD créent des types comportant une propriété nommée foo :

```
<xsd:complexType name="ElementFoo">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
```

```

</xsd:complexType>

<xsd:complexType name="AttributeFoo">
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>

```

Dans ces cas-là, vous pouvez accéder à la propriété en utilisant le langage XML Path (XPath). Cependant, les types de schéma valides peuvent comporter un attribut et un élément qui portent le même nom, comme dans l'exemple suivant :

```

<xsd:complexType name="DuplicateNames">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>

```

Dans XPath, vous devez pouvoir différencier des éléments portant le même nom des attributs. Pour cela, on ajoute au début des noms le symbole (@). Le fragment suivant montre comment accéder à un élément et un attribut portant le même nom :

```

1 DataObject duplicateNames = ...

2 // Affiche "elem_value"
3 System.out.println(duplicateNames.get("foo"));

4 // Affiche "attr_value"
5 System.out.println(duplicateNames.get("@foo"));

```

Utilisez ce schéma de désignation pour toutes les méthodes prenant une valeur de chaîne dans un XPath SDO.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
 La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Prise en charge de groupes de modèles (tous, choix, séquence et références de groupes)

La spécification SDO nécessite que les groupes de modèles (tous, choix, séquence et références de groupes) soient développés et ne décrit pas les types ni les propriétés.

Prise en charge de groupes de modèles (tous, choix, séquence et références de groupes) :

La spécification SDO nécessite que les groupes de modèles (tous, choix, séquence et références de groupes) soient développés et ne décrit pas les types ni les propriétés.

Pratiquement, cela signifie que toutes les structures qui se trouvent dans les mêmes structures sont "mises à plat". Cette "mise à plat" met tous les enfants de ces structures au même niveau. Ceci peut entraîner des problèmes de noms dupliqués dans un SDO dont la structure est dérivée des données mises à plat. Lorsqu'un XSD ne met pas à plat les groupes, les noms dupliqués contenus par des parents différents restent séparés.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>

```

```

<xsd:choice>
  <xsd:element name="option1" type="xsd:string"/>
  <xsd:element name="option2" type="xsd:string"/>
</xsd:choice>
<xsd:element name="separator" type="xsd:string"/>
<xsd:choice>
  <xsd:element name="option1" type="xsd:string"/>
  <xsd:element name="option2" type="xsd:string"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

Les occurrences multiples de option1 et option2 se trouvant dans des blocs de choix distincts, et comportant même un élément de séparation entre eux, le XSD et le XML les distingue sans problème. Mais lorsque le SDO met à plat ces groupes, toutes les propriétés d'option sont maintenant sous le même conteneur de groupe multiple.

Même sans noms dupliqués, la mise à plat de ces groupes entraîne un problème d'ordre sémantique. Par exemple, pour le XSD suivant :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://SimpleChoice">
  <xsd:complexType name="SimpleChoice">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Demander à l'utilisateur de renommer les noms dupliqués ou d'ajouter des annotations spéciales aux XSD n'est pas possible dans beaucoup de cas, comme les schémas de normes ou industriels, car l'utilisateur ne contrôle pas les XSD avec lesquels il travaille.

Pour que toutes les propriétés soient cohérentes, les objets métier incluent une méthode pour accéder à chaque occurrence individuelle des propriétés portant le même nom via la balise XPath. Selon la convention de dénomination EMF, le chiffre non utilisé suivant sera ajouté à tous les noms dupliqués trouvés ; par exemple, le XSD suivant :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://TieredGroup">
  <xsd:complexType name="TieredGroup">
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:sequence>
          <xsd:element name="low" minOccurs="1"
            maxOccurs="1" type="xsd:string"/>
          <xsd:choice minOccurs="0">
            <xsd:element name="width" minOccurs="0"
              maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
              maxOccurs="1" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
        <xsd:element name="high" minOccurs="1"
          maxOccurs="1" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

```

        <xsd:sequence>
          <xsd:element name="width" minOccurs="1"
maxOccurs="1" type="xsd:string"/>
          <xsd:element name="high" minOccurs="0"
maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="center" minOccurs="1"
maxOccurs="1" type="xsd:string"/>
          <xsd:element name="width" minOccurs="0"
maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Le XSD précédent produit le modèle d'objet de données suivant :

```

DataObject - TieredGroup
  Property[0] - low - string
  Property[1] - width - string
  Property[2] - high - string
  Property[3] - high1 - string
  Property[4] - width1 - string
  Property[5] - high2 - string
  Property[6] - center - string
  Property[7] - width2 - string

```

Où **width**, **width1** et **width2** sont les noms des propriétés nommées "width" en commençant par la première dans le XSD et ainsi de suite, et de même pour **high**, **high1**, **high2**.

Les nouveaux noms des propriétés sont les noms utilisés pour référence et XPath et n'affectent pas le contenu sérialisé. Les noms "vrais" de chacune de ces propriétés apparaissant dans le XML sérialisé sont les valeurs données dans le XSD. Ainsi, pour l'instance XML :

```

<?xml version="1.0" encoding="UTF-8"?>
<p:TieredGroup xsi:type="p:TieredGroup"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://TieredGroup">
  <width>foo</width>
  <high>bar</high>
</p:TieredGroup>

```

Pour accéder à ces propriétés, vous devez utiliser le code suivant :

```

DataObject tieredGroup = ...

// Affiche "foo"
System.out.println(tieredGroup.get("width1"));

// Affiche "bar"
System.out.println(tieredGroup.get("high2"));

```

Concepts associés

Différenciation d'éléments portant le même nom

Vous devez donner des noms uniques aux éléments et attributs d'objet de données.

Différenciation de propriétés portant le même nom

Lorsque plusieurs XSD avec le même espace de nom définissent des types portant le même nom, un type incorrect peut être accidentellement référencé.

Address1.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="city" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Address2.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="state" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Les objets métier ne prennent pas en charge les noms dupliqués pour des structures XSD globales (telles que `complexType`, `simpleType`, `element`, `attribute`, etc.) par le biais des API `BOFactory.create()`. Il est cependant possible de créer ces structures globales dupliquées comme enfants d'autres structures si les API correctes sont utilisées, comme indiqué dans les exemples suivants

Customer1.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://Customer1"
  targetNamespace="http://Customer1">
  <xsd:import schemaLocation="./Address1.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Customer2.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://Customer2"
  targetNamespace="http://Customer2">
  <xsd:import schemaLocation="./Address2.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Lorsque les champs "Customer address" sont renseignés et que l'API `BOFactory.create()` est appelée pour créer l'adresse, les types d'objets métier enfants qui en résultent peuvent être définis d'une manière incorrecte. Pour éviter cela, vous pouvez appeler l'API `createDataObject("address")` sur l'objet de données "Customer". Un enfant de type correct sera ainsi créé, car les objets métier correspondront à l'emplacement de schéma de l'importation.

```
DataObject customer1 = ...
```

```

// Manière incorrecte de créer un enfant "Address"
// Un type d'adresse Address1.xsd ou Address2.xsd risquerait d'être créé
DataObject incorrect = boFactory.create("", "Address");
customer1.set("address", incorrect);

```

```
// Manière correcte de créer un enfant "Address"  
// Le type d'adresse Address1.xsd sera ainsi forcément créé  
customer1.createDataObject("address");
```

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Résolution de noms de propriétés contenant des points

Les noms des propriétés dans un XSD peuvent contenir un point (".") comme un des nombreux caractères valides, alors que, dans un SDO, ils sont également utilisés pour montrer l'indexation dans une propriété à cardinalité multiple. Dans certaines situations, ceci peut entraîner des problèmes de résolution.

Les noms des propriétés dans les objets de données de service (SDO) sont basés sur les noms des éléments et de l'attribut à partir desquels ils sont générés dans le XSD. Les objets métier traiteront le caractère "." correctement, avec une exception : si un XSD comporte une propriété à cardinalité unique dont le nom est "<name>.<#>" et une propriété à cardinalité multiple dont le nom est "<name>".

Une balise XPath telle que "foo.0" ne sera pas résolue correctement s'il y a une propriété à cardinalité unique nommée "foo.0" et une propriété à cardinalité multiple appelée "foo". Dans ce cas, la propriété à cardinalité unique portant le nom "foo.0" est celle qui sera résolue. Bien que cela ne risque de se produire que rarement, vous pouvez l'éviter entièrement si vous utilisez la syntaxe "foo[1]" pour accéder à leur propriété à cardinalité multiple. Les SDO ne prendront pas en charge la syntaxe "." pour l'indexation, et vous devez donc utiliser "[]" pour l'indexation.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Sérialisation et désérialisation d'unions portant xsi:type

Dans le XSD, une union est un moyen de fusionner les espaces lexicaux de plusieurs types de données simples connus comme membres.

Sérialisation et désérialisation d'unions portant xsi:type :

Dans le XSD, une union est un moyen de fusionner les espaces lexicaux de plusieurs types de données simples connus comme membres.

L'exemple de XSD suivant montre une union comportant les membres d'un nombre entier et d'une date.

```
<xsd:simpleType name="integerOrDate">  
  <xsd:union memberTypes="xsd:integer xsd:date"/>  
</xsd:simpleType>
```

Cette saisie multiple peut entraîner une confusion lors de la désérialisation et de la manipulation des données.

Les objets métier prennent en charge les SDO utilisant xsi:type pour la sérialisation et suivront le même algorithme pour déterminer le type lors d'une désérialisation si le xsi:type n'est pas présent dans les données XML.

Ainsi, pour garantir que les données (le nombre "42" dans cet exemple) seront désérialisées comme un nombre entier, vous pouvez utiliser le `xsi:type` spécifié dans le XML d'entrée. Vous pouvez également ordonner la liste des membres de l'union dans le XSD de telle sorte que le nombre entier soit avant la chaîne. L'exemple suivant montre comment les deux méthodes sont mises en oeuvre :

```
<integerOrString xsi:type="xsd:integer">42</integerOrString>
```

```
<xsd:simpleType name="integerOrString">  
  <xsd:union memberTypes="xsd:integer xsd:string"/>  
</xsd:simpleType>
```

De même, si l'utilisateur souhaitait que les données soient désérialisées en tant que chaîne, l'une ou l'autre des modifications suivantes entraînerait le comportement suivant :

```
<integerOrString xsi:type="xsd:string">42</integerOrString>
```

```
<xsd:simpleType name="integerOrString">  
  <xsd:union memberTypes="xsd:string xsd:integer"/>  
</xsd:simpleType>
```

Remarque : si un type de chaîne est le premier membre de l'union, aucune de ses informations n'est jamais perdue. Il peut également contenir toutes les données qui seront toujours choisies par l'algorithme `no xsi:type`. Si vous souhaitez utiliser un autre type qu'une chaîne, vous devez soit utiliser `xsi:type` dans le XML soit réorganiser les types de membre dans le XSD pour donner aux autres membres la possibilité d'accepter les données.

Concepts associés

Résolution de noms de propriétés contenant des points

Les noms des propriétés dans un XSD peuvent contenir un point (".") comme un des nombreux caractères valides, alors que, dans un SDO, ils sont également utilisés pour montrer l'indexation dans une propriété à cardinalité multiple. Dans certaines situations, ceci peut entraîner des problèmes de résolution.

Utilisation de l'objet de séquence pour définir l'ordre des données

Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Par exemple, l'ordre est important dans les XSD si le contenu est mixte. Si les données de texte apparaissent avant ou après un élément, la signification peut être différente que si elles apparaissent dans un autre emplacement. Pour ces situations, le SDO génère un objet connu sous le nom de Séquence, qui est utilisé pour définir les données d'une manière ordonnée.

Les séquences SDO ne doivent pas être confondues avec les séquences XSD. Les séquences XSD ne sont que des groupes de modèles mis à plat avant la génération du modèle SDO. La présence d'une séquence XSD n'a pas de rapport avec la présence d'une séquence SDO.

Avec les conditions suivantes, un XSD dans une séquence SDO est généré :

Un type complexe avec du contenu mixte :

```
<?xml version="1.0" encoding="UTF-8"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:tns="http://MixedContent"  
  targetNamespace="http://MixedContent">  
  <xsd:complexType name="MixedContent" mixed="true">
```

```

    <xsd:sequence>
      <xsd:element name="element1" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element2" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element3" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MixedContent" type="tns:MixedContent"/>
</xsd:schema>

```

Un schéma comportant 1 ou plusieurs balises <any/> :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://AnyElemAny"
  targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <xsd:any/>
      <xsd:element name="marker1" type="xsd:string"/>
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Un tableau de groupes de modèles (tous, choix, séquence ou référence de groupe avec maxOccurs > 1) :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence maxOccurs="3">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Une balise <all/> de maxOccurs <= 1 contenant plusieurs éléments :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://All">
  <xsd:complexType name="All">
    <xsd:all>
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>

```

Vous trouverez des informations spécifiques sur l'utilisation de <any/> avec une séquence dans la rubrique référencée en bas de cette page. Les informations d'ordre général qui suivent dans le reste de cette section expliquent comment travailler avec les autres conditions de séquence, mais s'appliquent également à <any/>.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Comment savoir si mon objet de données possède une séquence ?

Deux API simples permettent de déterminer si un objet de données est mis en séquence : `DataObject noSequence` et `DataObject withSequence`.

Pourquoi dois-je savoir si un objet de données possède une séquence ?

Si vous travaillez sur un objet de données comportant une séquence, il est important de connaître l'ordre dans lequel les données sont définies. Vous devez donc faire attention à l'ordre dans lequel les données sont définies.

Comment utiliser des contenus mixtes ?

Pour les contenus mixtes, la séquence comporte une API spécifique pour ajouter du texte : `addText(...)`.

Comment utiliser un tableau de groupes de modèles ?

Un tableau de groupes de modèles est créé lorsque la valeur `maxOccurs` d'un groupe de modèles est > 1 .

Comment savoir si mon objet de données possède une séquence ? :

Deux API simples permettent de déterminer si un objet de données est mis en séquence : `DataObject noSequence` et `DataObject withSequence`.

Vous pouvez utiliser `DataObject noSequence` et `DataObject withSequence` de la manière indiquée dans l'exemple suivant :

```
DataObject noSequence = ...
DataObject withSequence = ...

// Affiche la valeur faux
System.out.println(noSequence.getType().isSequenced());

// Affiche la valeur vrai
System.out.println(withSequence.getType().isSequenced());

// Affiche la valeur vrai
System.out.println(noSequence.getSequence() == null);

// Affiche la valeur faux
System.out.println(withSequence.getSequence() == null);
```

Concepts associés

Utilisation de l'objet de séquence pour définir l'ordre des données

Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Pourquoi dois-je savoir si un objet de données possède une séquence ? :

Si vous travaillez sur un objet de données comportant une séquence, il est important de connaître l'ordre dans lequel les données sont définies. Vous devez donc faire attention à l'ordre dans lequel les données sont définies.

Un objet de données qui n'est pas mis en séquence permet l'accès à un ensemble dans un ordre aléatoire. Le fonctionnement est identique à un mappage dans lequel toutes les clés sont définies sur les mêmes valeurs. L'ordre dans lequel les clés sont définies n'a pas d'importance, les données au sein du mappage étant identiques et étant sérialisées en XML d'une manière identique.

Lorsqu'un objet de données est mis en séquence, l'ordre dans lequel les données ont été définies est enregistré dans la séquence, comme s'il s'agissait d'ajouter des données à une liste. Ainsi, deux manières d'accéder aux données sont possibles : par paires nom/valeur (les API d'objet de données) et selon l'ordre dans lequel elles ont été définies (les API de séquence). Vous pouvez utiliser les API d'objet de données `set(...)` ou de séquence `add(...)` pour conserver la structure. Cet ordre a un incidence sur la manière dont le XML est sérialisé.

Prenons par exemple le XSD de la balise `<all/>` ci-dessous. Lorsque les méthodes `set` sont appelées dans l'ordre suivant, le XML suivant est produit lorsqu'il est sérialisé :

```
DataObject all = ...
all.set("element1", "foo");
all.set("element2", "bar");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:p="http://A11">
  <element1>foo</element1>
  <element2>bar</element2>
</p:All>
```

Si, à la place, les méthodes `set` sont appelées dans l'ordre inverse, le XML suivant est produit lorsque l'objet métier est sérialisé :

```
DataObject all = ...
all.set("element2", "bar");
all.set("element1", "foo");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:p="http://A11">
  <element2>bar</element2>
  <element1>foo</element1>
</p:All>
```

Si l'ordre de la séquence doit être modifié, la classe de la séquence a les méthodes `add`, `remove` et `move` pour permettre à l'utilisateur de modifier l'ordre de la séquence.

Concepts associés

Utilisation de l'objet de séquence pour définir l'ordre des données

Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Comment utiliser des contenus mixtes ? :

Pour les contenus mixtes, la séquence comporte une API spécifique pour ajouter du texte : `addText(...)`.

Toutes les autres API fonctionnent de la même manière avec du texte comme avec les propriétés. L'API `getProperty(int)` renverra la valeur `null` pour les données de texte avec des contenus mixtes. L'exemple suivant de code de contenu mixte peut être utilisé pour imprimer tout le texte avec des contenus mixtes depuis un objet de données :

```
DataObject mixedContent = ...
Sequence seq = mixedContent.getSequence();

for (int i=0; i < seq.size(); i++)
```

```

{
    Property prop = seq.getProperty(i);
    Object value = seq.getValue(i);

    si (prop == null)
    {
        System.out.println("Found mixed content text: "+value);
    }
    else
    {
        System.out.println("Found Property "+prop.getName()+": "+value);
    }
}

```

Concepts associés

Utilisation de l'objet de séquence pour définir l'ordre des données
Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Comment utiliser un tableau de groupes de modèles ? :

Un tableau de groupes de modèles est créé lorsque la valeur maxOccurs d'un groupe de modèles est > 1.

Les groupes de modèles étant mis à plat et n'étant pas exprimés dans un objet de données, les propriétés au sein du groupe de modèles deviennent des propriétés à cardinalité multiple et leurs méthodes isMany() renvoient la valeur vrai si elles ne l'ont pas déjà. Leurs facettes minOccurs et maxOccurs sont alors multipliées par celles du groupe de modèles qui les contient. Le choix multipliera la facette maxOccurs de la même manière que les autres groupes de modèles, mais utilisera toujours 0 comme valeur de multiplication pour minOccurs, car toutes les données dans le choix peuvent ne pas être sélectionnées.

Par exemple, le XSD suivant comporte un tableau de groupes de modèles :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
minOccurs="0" maxOccurs="3"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Comme indiqué, **element1** et **element2** seront maintenant une cardinalité multiple et un mécanisme d'accès get(...) renverra donc une liste. **Element1** a par défaut la valeur minOccurs 1 et la valeur maxOccurs 1. **Element2** a la valeur minOccurs 0 et la valeur maxOccurs 3. Dans l'exemple suivant, leurs nouvelles valeurs minOccurs et maxOccurs seront les suivantes :

```

Data Object - ModelGroupArray
Property[0] - element1 - minOccurs=(2*1)=2 - maxOccurs=(5*1)=5
Property[1] - element2 - minOccurs=(2*0)=0 - maxOccurs=(5*3)=15

```

Si le type était Choix :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:choice minOccurs="2" maxOccurs="5">

```

```

        <xsd:element name="element1" type="xsd:string"/>
        <xsd:element name="element2" type="xsd:string"
            minOccurs="0" maxOccurs="3"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

La valeur minOccurs suivante serait générée, en raison de l'exclusion du choix indiquant que seul **element1** puisse être extrait à chaque fois ou que seul **element2** puisse être extrait à chaque fois, et les deux doivent pouvoir avoir 0 occurrence pour réussir la validation :

```

DataObject - ModelGroupArray
Property[0] - element1 - minOccurs=(0*1)=0 - maxOccurs=(5*1)=5
Property[1] - element2 - minOccurs=(0*0)=0 - maxOccurs=(5*3)=15

```

Concepts associés

Utilisation de l'objet de séquence pour définir l'ordre des données
Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Utilisation de AnySimpleType pour les types simples

AnySimpleType est traité de la même manière que les autres types simples (chaîne, int, booléen, etc.) par les API SDO.

Les seules différences entre anySimpleType et les autres types simples sont dans ses données d'instance et la sérialisation/désérialisation. Elles doivent être des concepts internes pour les objets métier uniquement, et elles sont utilisées pour déterminer si les données mappées vers ou depuis le champ sont valides. Si une méthode set(...) devait être appelée sur un type de chaîne, les données seraient d'abord converties en une chaîne, et les données d'origine seraient perdues :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://StringType">
    <xsd:complexType name="StringType">
        <xsd:sequence>
            <xsd:element name="foo" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>

```

```
DataObject stringType = ...
```

```
// Définir les données sur une chaîne
stringType.set("foo", "bar");
```

```
// Les données d'instance seront toujours du type chaîne,
// quelles que soient les données définies
// Affiche "java.lang.String"
System.out.println(stringType.get("foo").getClass().getName());
```

```
// Définir les données sur un nombre entier
stringType.set("foo", new Integer(42));
```

```
// Les données d'instance seront toujours du type chaîne,
// quelles que soient les données définies
// Affiche "java.lang.String"
System.out.println(stringType.get("foo").getClass().getName());
```

Un élément anySimpleType à la place ne perd pas le type de données d'origine de ce qui est défini :

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://AnySimpleType">
  <xsd:complexType name="AnySimpleType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:anySimpleType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

```

DataObject anySimpleType = ...

// Définir les données sur une chaîne
stringType.set("foo", "bar");

// Les données d'instance seront toujours du type date utilisé dans l'ensemble
// Affiche "java.lang.String"
System.out.println(stringType.get("foo").getClass().getName());

// Définir les données sur un nombre entier
stringType.set("foo", new Integer(42));

// Les données d'instance seront toujours du type date utilisé dans l'ensemble
// Affiche "java.lang.Integer"
System.out.println(stringType.get("foo").getClass().getName());

```

Ce type de données est également préservé lors de la sérialisation et désérialisation par `xsi:type`. En conséquence, à chaque fois que vous sérialisez un élément `anySimpleType`, il aura un `xsi:type` qui correspond à celui défini dans la spécification SDO en fonction de son type Java :

Dans l'exemple suivant, vous sérialisez l'objet métier ci-dessus de telle sorte que les données ressembleront à :

```

<?xml version="1.0" encoding="UTF-8"?>
<p:StringType xsi:type="p:StringType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:p="http://StringType">
  <foo xsi:type="xsd:int">42</foo>
</p:StringType></p:StringType>

```

Le `xsi:type` sera utilisé lors de la désérialisation pour charger les données comme classe d'instance Java appropriée. Si aucun `xsi:type` n'est spécifié, le type de désérialisation par défaut sera une chaîne.

Pour les autres types simples, déterminer la mappabilité est une constante. Par exemple, un élément booléen peut toujours mapper une chaîne. `AnySimpleType` peut contenir n'importe quel type simple, mais un mappage peut être possible ou non, en fonction des données d'instance dans le champ.

Utilisez le type de propriété URI et Nom pour déterminer si une propriété est du type `anySimpleType`. Il s'agira de "commonj.sdo" et "Object". Pour déterminer si des données sont valides pour être insérées dans `anySimpleType`, vérifiez s'il ne s'agit pas d'une instance d'un objet de données. Toutes les données pouvant être représentées sous la forme d'une chaîne et n'étant pas un objet de données peuvent être définies dans un champ `anySimpleType`.

Les règles de mappage sont donc les suivantes :

- `anySimpleType` peut toujours être mappé sur `anySimpleType`.
- n'importe quel autre type simple peut toujours être mappé sur `anySimpleType`.

- anySimpleType peut toujours être mappé sur une chaîne car tous les types simples doivent pouvoir être convertis en une chaîne.
- anySimpleType peut ou ne peut pas être mappé sur un des autres types simples, en fonction de sa valeur dans l'objet métier. Cela signifie que ce mappage ne peut pas être déterminé au moment de la conception, mais uniquement lors de l'exécution.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Information associée

➡ Affectation depuis et vers xs:any

Utilisation de AnyType pour les types complexes

La balise anyType est traitée de la même manière que les autres types complexes par les API SDO.

Les seules différences entre anyType et les autres types complexes sont dans leurs données d'instance et la sérialisation/désérialisation, qui doivent être des concepts internes pour l'objet métier uniquement, et déterminant si les données mappées vers ou depuis le champ sont valides. Les types complexes sont limités à un type unique : Client, Adresse, etc. La balise anyType, cependant, permet n'importe quel objet de données quel que soit le type. Si maxOccurs > 1, chaque objet de données de la liste peut être d'un type différent.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://AnyType">
  <xsd:complexType name="AnyType">
    <xsd:sequence>
      <xsd:element name="person" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://Customer">
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://Employee" targetNamespace="http://Employee">
  <xsd:complexType name="Employee">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
DataObject anyType = ...
DataObject customer = ...
DataObject employee = ...
```

```
// Définir la personne sur Customer
anyType.set("person", customer);
```

```

// Les données d'instance seront un client
// Affiche "Customer"
System.out.println(anyType.getDataObject("person").getName());

// Définir la personne sur Employee
anyType.set("person", employee);

// Les données d'instance seront un employé
// Affiche "Employee"
System.out.println(anyType.getDataObject("person").getName());

```

Comme `anySimpleType`, `anyType` utilise l'élément `xsi:type` lors de la sérialisation afin d'assurer que le type d'objet de données voulu est conservé lorsqu'il est désérialisé. Ainsi, si vous le définissez sur "Customer", le XML se présente comme suit :

```

<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:customer="http://Customer"
  xmlns:p="http://AnyType">
  <person xsi:type="customer:Customer">
    <name>foo</name>
  </person>
</p:AnyType>

```

Et, si vous le définissez sur "Employee" :

```

<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:employee="http://Employee"
  xmlns:p="http://AnyType">
  <person xsi:type="employee:Employee">
    <id>foo</id>
  </person>
</p:AnyType>

```

La balise `AnyType` permet également de définir des valeurs de type simple par le biais d'objets de données encapsuleurs. Ces objets de données encapsuleurs possèdent une propriété unique appelée "value" (élément) qui contient la valeur de type simple. Les API SDO ont été écrasées pour encapsuler et désencapsuler automatiquement ces objets de données de types simples et encapsuleurs lorsque les API de `<Type>get/set<>` sont utilisées. Les API `get/set` de transtypage non-type n'effectueront pas cet encapsulage.

```
DataObject anyType = ...
```

```

// Appeler une API de <Type> set sur une propriété anyType entraîne la création
// automatique d'un objet de données encapsuleur
anyType.setString("person", "foo");

```

```

// Les API get/set classiques ne sont pas écrasées, et renverront donc
// l'objet de données encapsuleur
DataObject wrapped = anyType.get("person");

```

```

// L'objet de données encapsulé aura la propriété "value"
// Affiche "foo"
System.out.println(wrapped.getString("value"));

```

```

// L'API de <Type> get désencapsulera automatiquement l'objet de données
// Affiche "foo"
System.out.println(anyType.getString("person"));

```

Lorsque l'objet de données encapsuleur est sérialisé, il est sérialisé de la même manière qu'un mappage anySimpleType de classes d'instance Java en types XSD dans le champ xsi:type. Ce paramètre doit donc être sérialisé de la manière suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:p="http://AnyType">
  <person xsi:type="xsd:string">foo</person>
</p:AnyType>
```

Si aucun élément xsi:type n'est donné ou si un élément xsi:type incorrect est donné, une exception est émise. En plus de l'encapsulage automatique, l'encapsuleur peut être créé manuellement pour être utilisé avec l'API set() via BOFactory createDataTypeWrapper(Type, Object), où Type est le type simple de SDO des données à encapsuler et Object représente les données à encapsuler.

```
Type stringType = boType.getType("http://www.w3.org/2001/XMLSchema", "string");
DataObject stringType = boFactory.createByMessage(stringType, "foo");
```

Pour déterminer si un objet de données est du type encapsuleur, l'élément BOType isDataTypeWrapper(Type) peut être appelé.

```
DataObject stringType = ...
boolean isWrapper = boType.isDataTypeWrapper(stringType.getType());
```

Pour les autres types complexes, pour pouvoir déplacer les données d'un champ à l'autre, les données doivent être du même type. La balise AnyType peut contenir n'importe quel type complexe, mais un déplacement direct sans mappage peut être basé sur les données d'instance dans le champ ou non.

Vous pouvez utiliser l'URI et le Nom du type de propriété pour déterminer si une propriété est du type anyType. Il s'agira de "commonj.sdo" et "DataObject". Toutes les données sont valides pour être insérées dans une balise anyType. Les règles de mappage sont donc les suivantes :

- anyType peut toujours être mappé sur anyType.
- n'importe quel type complexe peut toujours être mappé sur anyType.
- n'importe quel type simple peut toujours être mappé sur anyType.
- anyType peut ou ne peut pas être mappé sur un des autres types simples ou complexes, en fonction de sa valeur dans l'instance d'objet métier. Cela signifie que ce mappage ne peut pas être déterminé au moment de la conception, mais uniquement au moment de l'exécution.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Utilisation de la balise Any pour définir des éléments globaux de types complexes

Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

Avec une occurrence de la balise any, les méthodes DataObject Type isOpen() et isSequenced() renvoient la valeur vrai. Si la valeur de maxOccurs est > 1 sur une balise any, cela n'a aucune incidence sur la structure de l'objet de données ; elle est utilisée uniquement comme information lors de la validation. De la même manière,

L'occurrence de balises any multiples dans un type ne modifie pas la structure de l'objet de données ; elles sont utilisées uniquement pour valider l'emplacement des données ouvertes qui ont été définies.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Comment savoir si mon objet de données possède une balise ?

Vous pouvez déterminer facilement si des valeurs sont définies au sein d'un objet de données en vérifiant leurs propriétés pour voir si des propriétés ouvertes sont des attributs.

Comment obtenir/définir des valeurs ?

Vous pouvez exécuter une instruction get sur des données qui ont été définies dans un champ de la même manière que pour une autre valeur d'élément si le nom est connu.

Quels sont les mappages de données valides pour la valeur Any ?

Une balise <any/> est un ensemble de paires nom/valeur. Le seul mappage valide pouvant être déterminé au moment de la conception pour <any/> est une autre balise <any/> ou anyType ayant la même valeur maxOccurs.

Comment savoir si mon objet de données possède une balise ? :

Vous pouvez déterminer facilement si des valeurs sont définies au sein d'un objet de données en vérifiant leurs propriétés pour voir si des propriétés ouvertes sont des attributs.

L'objet de données ne possède pas de mécanisme permettant de déterminer si un type d'objet de données comporte des balises. Les objets de données possèdent uniquement le concept "ouvert" qui s'applique à la balise any et à la balise anyAttribute, et qui permet d'ajouter librement des propriétés. Alors que la présence d'une balise implique qu'un objet de données a la valeur isOpen() = vrai et isSequenced() = vrai, il peut comporter uniquement une balise anyAttribute et une des raisons pour lesquelles il est mis en séquence décrite dans la rubrique Séquences. L'exemple suivant explique ces concepts :

```
DataObject dobj = ...
```

```
// Vérifiez si le type est "ouvert" ; dans le cas contraire, aucune valeur ne peut  
// être définie pour cet objet de données.  
boolean isOpen = dobj.getType().isOpen();
```

```
si (!isOpen) renvoie la valeur faux ; // Aucune valeur n'est définie pour cet  
objet de données
```

```
// Les propriétés ouvertes sont ajoutées à la liste des propriétés de l'instance,  
// mais pas la liste des propriétés. Ainsi, comparer leurs tailles permet de  
// déterminer facilement si des données ouvertes sont définies  
int instancePropertyCount = dobj.getInstanceProperties().size();  
int definedPropertyCount = dobj.getType().getProperties().size();
```

```
// Si elles sont égales, aucun contenu ouvert n'est défini  
si (instancePropertyCount == definedPropertyCount) renvoie la valeur faux ;
```

```
// Vérifiez les propriétés du contenu ouvertes pour déterminer si certaines d'entre  
elles sont des éléments  
for (int i=definedPropertyCount; i < instancePropertyCount; i++)  
{  
    Property prop = (Property)dobj.getInstanceProperties().get(i);  
    si (boXsdHelper.isElement(prop))  
    {
```

```

    }
    renvoie la valeur vrai ; // Une valeur any a été trouvée
}

```

```
renvoie la valeur faux ; // Aucune valeur n'est définie
```

Concepts associés

Utilisation de la balise Any pour définir des éléments globaux de types complexes
 Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

Comment obtenir/définir des valeurs ? :

Vous pouvez exécuter une instruction get sur des données qui ont été définies dans un champ de la même manière que pour une autre valeur d'élément si le nom est connu.

Vous pouvez envoyer une instruction get avec la balise XPath "<name>" pour la résoudre. Si le nom est inconnu, il est possible de trouver la valeur en vérifiant les propriétés de l'instance comme ci-dessus. S'il y a plusieurs balises any, ou une balise any avec maxOccurs > 1, la séquence de l'objet de données devra être utilisée à la place s'il est important de déterminer quelle balise any est à l'origine des données.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://AnyElemAny"
  targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <!-- Handle all these any one way -->
      <xsd:any maxOccurs="3"/>
      <xsd:element name="marker1" type="xsd:string"/>
      <!-- Handle this any in another -->
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

La balise <any/> entraînant la mise en séquence de l'objet de données, il est possible de déterminer quelle valeur any a été définie en vérifiant dans la séquence la position des propriétés any.

Vous pouvez déterminer à quelle balise any les données d'instance appartiennent pour le XSD suivant en utilisant le code suivant :

```

DataObject anyElemAny = ...
Sequence seq = anyElemAny.getSequence();

// Jusqu'à ce que l'élément marker1 ait été trouvé, toutes les données ouvertes
// trouvées appartiennent à la première balise any
boolean foundMarker1 = false;

for (int i=0; i<seq.size(); i++)
{
  Property prop = seq.getProperty(i);

  // Vérifiez si la propriété est une propriété ouverte
  si (prop.isOpenContent())
  {
    si (!foundMarker1)
    {
      // Doit être la première balise any car elle survient
      // avant l'élément marker1
    }
  }
}

```

```

        System.out.println("Found first any data: "+seq.getValue(i));
    }
    else
    {
        // Doit être la seconde balise any car elle survient
        // après l'élément marker1
        System.out.println("Found second any data: "+seq.getValue(i));
    }
}
else
{
    // Doit être l'élément marker1
    System.out.println("Found marker1 data: "+seq.getValue(i));
    foundMarker1 = true;
}
}
}

```

Définir une valeur `<any/>` est effectué en créant une propriété d'élément global et en ajoutant cette valeur à la séquence.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://GlobalElems"
  targetNamespace="http://GlobalElems">
  <xsd:element name="globalElement1" type="xsd:string"/>
  <xsd:element name="globalElement2" type="xsd:string"/>
</xsd:schema>

```

```

DataObject anyElemAny = ...
Sequence seq = anyElemAny.getSequence();

// Obtenir la propriété de l'élément global pour globalElement1
Property globalProp1 = boXsdHelper.getGlobalProperty(http://GlobalElems,
"globalElement1", true);

// Obtenir la propriété de l'élément global pour globalElement2
Property globalProp2 = boXsdHelper.getGlobalProperty(http://GlobalElems,
"globalElement2", true);

// Ajouter les données à la séquence pour la première balise any
seq.add(globalProp1, "foo");
seq.add(globalProp1, "bar");

// Ajouter les données pour le marker1
seq.add("marker1", "separator"); // ou anyElemAny.set("marker1", "separator")

// Ajouter les données à la séquence pour la seconde balise any
seq.add(globalProp2, "baz");

// Il est maintenant possible d'accéder aux données avec une instruction get
System.out.println(dobj.get("globalElement1"); // Affiche "[foo, bar]"
System.out.println(dobj.get("marker1"); // Affiche "separator"
System.out.println(dobj.get("globalElement2"); // Affiche "baz"

```

Concepts associés

Utilisation de la balise Any pour définir des éléments globaux de types complexes
 Vous pouvez utiliser la balise `<any/>` pour définir des éléments globaux sur un type complexe.

Quels sont les mappages de données valides pour la valeur Any ? :

Une balise `<any/>` est un ensemble de paires nom/valeur. Le seul mappage valide pouvant être déterminé au moment de la conception pour `<any/>` est une autre balise `<any/>` ou `anyType` ayant la même valeur `maxOccurs`.

Individuellement, les valeurs contenues dans une instance d'un objet de données pour la balise any sont des types complexes de base respectant les règles d'un mappage de type complexe. Certains de ces types complexes peuvent être des types simples encapsulés, et ils suivront les règles du mappage de type simple.

Concepts associés

Utilisation de la balise Any pour définir des éléments globaux de types complexes
Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

Utilisation de AnyAttribute pour définir les attributs globaux de types complexes

La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Comme pour la balise <any/>, l'occurrence de la balise <anyAttribute/> entraîne le renvoi par la méthode DataObject Type isOpen() de la valeur true. Toutefois, contrairement à la balise <any/>, <anyAttribute/> n'implique pas le séquençement de l'objet données, car les attributs de XSD ne sont pas des constructions ordonnées.

Concepts associés

Objets métier : renforcement du schéma et prise en charge du schéma industriel
La structure SDO (Service Data Objects) fournit la base pour les données d'objet métier utilisées par WebSphere Process Server

Comment savoir si mon objet de données possède une balise AnyAttribute ?

Vous pouvez aisément déterminer si des instances d'un objet données comportent un ensemble de valeurs anyAttribute en vérifiant leurs propriétés pour savoir si les propriétés ouvertes représentent des attributs.

Comment obtenir/définir des valeurs AnyAttribute ?

Définir une valeur <anyAttribute/> est effectué de la même manière que pour une balise <any/>, mais un attribut global est utilisé à la place d'un élément global.

Quels sont les mappages de données valides pour la valeur AnyAttribute ?

La balise AnyAttribute est similaire à la balise any, et comprend un ensemble de paires nom/valeur. En conséquence, le seul mappage valide pour anyAttribute est une autre balise anyAttribute.

Comment savoir si mon objet de données possède une balise AnyAttribute ? :

Vous pouvez aisément déterminer si des instances d'un objet données comportent un ensemble de valeurs anyAttribute en vérifiant leurs propriétés pour savoir si les propriétés ouvertes représentent des attributs.

L'objet données ne prévoit pas de mécanisme pour déterminer si un type d'objet données inclut une balise anyAttribute. Seuls les objets données comportent un concept d'ouverture qui s'applique aux balises any et <anyAttribute/> et qui permet l'ajout de propriétés supplémentaires. S'il est vrai que si un objet données a défini isOpen() = true et isSequenced() = false, il doit inclure une balise anyAttribute, si isOpen() = true et isSequenced() = true, le type d'objet données pouvant ou non inclure une balise anyAttribute.

L'objet données fournit des méthodes d'interrogation des métadonnées pour répondre à l'aide d'un programme à cette question et bien d'autres sur la structure XSD qui a servi à le générer. Le modèle InfoSet peut être interrogé pour déterminer si nécessaire l'existence de la balise anyAttribute. Parce que la balise anyAttribute est unique et que sa valeur peut être ou non true, les objets métier fournissent

également une méthode `BOXSDHelper hasAnyAttribute(Type)` pour déterminer si la définition d'un attribut ouvert sur cet objet données produira un résultat valide. L'exemple de code suivant illustre ces concepts :

```
DataObject dobj = ...

// Vérifiez si le type est ouvert. S'il ne l'est pas, aucune
// valeur anyAttribute ne peut y être définie.
boolean isOpen = dobj.getType().isOpen() ;

si (!isOpen) return false ; // Aucune valeur anyAttribute définie

// Les propriétés ouvertes sont ajoutées à la liste des propriétés de l'instance,
// mais pas la liste des propriétés. Par conséquent, la comparaison de leurs tailles
// peut permettre de déterminer facilement si des données ouvertes sont définies
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// Si leur taille est identique, aucun contenu ouvert n'est défini
si (instancePropertyCount == definedPropertyCount) return false ;

// Vérifiez les propriétés du contenu ouvert pour déterminer l'une d'elles
//   constituant des attributs
pour (int i=definedPropertyCount; i<instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    si (boXsdHelper.isAttribute(prop))
    {
        return true ; // Valeur anyAttribute trouvée
    }
}

return false ; // Aucune valeur anyAttribute définie
```

Concepts associés

Utilisation de `AnyAttribute` pour définir les attributs globaux de types complexes
La balise `<anyAttribute/>` permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Comment obtenir/définir des valeurs `AnyAttribute` ? :

Définir une valeur `<anyAttribute/>` est effectué de la même manière que pour une balise `<any/>`, mais un attribut global est utilisé à la place d'un élément global.

Exécuter une instruction `get` sur des données qui ont été définies dans un champ `anyAttribute` peut être effectué de la même manière que pour une autre valeur d'attribut si le nom est connu. Vous pouvez envoyer une instruction `get` avec la balise XPath `"@<name>"` pour la résoudre. Si le nom est inconnu, vous pouvez utiliser le code ci-dessus pour itérer les valeurs et y accéder une par une.

L'exemple de code suivant montre comment procéder :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://AnyAttrOnlyMixed"
  targetNamespace="http://AnyAttrOnly">
  <xsd:complexType name="AnyAttrOnly">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```

targetNamespace="http://GlobalAttrs">
  <xsd:attribute name="globalAttribute" type="xsd:string"/>
</xsd:schema>

Data Object dobj = ...

// Obtenir la propriété de l'attribut global qui va être défini
Property globalProp = boXsdHelper.getGlobalProperty(http://GlobalAttrs,
"globalAttribute", false);

// Définir la valeur sur l'objet de données, comme n'importe quelle autre donnée
dobj.set(globalProp, "foo");

// Il est maintenant possible d'accéder aux données avec une instruction get
System.out.println(dobj.get("@globalAttribute")); // Affiche "foo"

```

Concepts associés

Utilisation de AnyAttribute pour définir les attributs globaux de types complexes
La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Quels sont les mappages de données valides pour la valeur AnyAttribute ? :

La balise AnyAttribute est similaire à la balise any, et comprend un ensemble de paires nom/valeur. En conséquence, le seul mappage valide pour anyAttribute est une autre balise anyAttribute.

Individuellement, les valeurs contenues dans les données anyAttribute sont des types simples de base respectant les règles du mappage de type simple

Concepts associés

Utilisation de AnyAttribute pour définir les attributs globaux de types complexes
La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Remplacement d'une conversion d'objet SDO en Java

Il se peut que la conversion d'un objet SDO (Service Data Object) en objet de type Java effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation par défaut par celle de votre choix.

Avant de commencer

Vérifiez que vous avez généré la conversion de type WSDL vers Java à l'aide de WebSphere Integration Developer ou la commande genMapper.

A propos de cette tâche

Pour remplacer un composant généré qui mappe un type WSDL à un type Java, remplacez le code généré par le code qui répond à vos besoins. Vous pouvez utiliser votre propre mappe si vous avez défini vos propres classes Java. Suivez cette procédure pour effectuer les modifications.

Procédure

1. Localisez le composant généré. Le nom du composant est *java_classMapper.component*.
2. Editez le composant dans un éditeur de texte.
3. Mettez en commentaires le code généré et insérez votre méthode.

Ne modifiez pas le nom du fichier qui contient l'implémentation du composant.

Exemple

Voici un exemple de composant généré à remplacer :

```
private Object dataToJava_get_customerAcct(DataObject myCustomerID,
    String integer)
{
    // Vous pouvez remplacer ce code par un mappage personnalisé.
    // Mettez en commentaire ce code et écrivez le code personnalisé.

    // Vous pouvez également changer le type Java transmis au
    // convertisseur que le convertisseur tente de convertir.

    return SDOJavaObjectMediator.data2Java(customerID, integer) ;
}
```

Que faire ensuite

Copiez le composant et les autres fichiers dans le répertoire où se trouve le module conteneur et connectez le composant dans WebSphere Integration Developer ou générez un fichier EAR à l'aide de la commande `serviceDeploy`.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Remplacement de l'implémentation d'architecture SCA générée

Il se peut que la conversion de code Java en objet SDO (Service Data Object) effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation d'architecture SCA (Service Component Architecture) par défaut par celle de votre choix.

Avant de commencer

Vérifiez que vous avez généré la conversion de type Java vers WSDL (Web Services Definition Language) en utilisant WebSphere Integration Developer ou la commande `genMapper`.

A propos de cette tâche

Pour remplacer un composant généré qui mappe un type Java à un type WSDL, remplacez le code généré par le code qui répond à vos besoins. Vous pouvez utiliser votre propre mappe si vous avez défini vos propres classes Java. Suivez cette procédure pour effectuer les modifications.

Procédure

1. Localisez le composant généré. Le nom du composant est `java_classMapper.component`.
2. Editez le composant dans un éditeur de texte.
3. Mettez en commentaires le code généré et insérez votre méthode.

Ne modifiez pas le nom du fichier qui contient l'implémentation du composant.

Exemple

Voici un exemple de composant généré à remplacer :

```
private DataObject javatodata_setAccount_output(Object myAccount) {  
  
    // Vous pouvez remplacer ce code par un mappage personnalisé.  
    // Mettez en commentaire ce code et écrivez le code personnalisé.  
  
    // Vous pouvez également changer le type Java transmis au  
    // convertisseur que le convertisseur tente de convertir.  
  
    return SDOJavaObjectMediator.java2Data(myAccount);  
  
}
```

Que faire ensuite

Copiez le composant et les autres fichiers dans le répertoire où se trouve le module conteneur et connectez le composant dans WebSphere Integration Developer ou générez un fichier EAR à l'aide de la commande `serviceDeploy`.

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Règles en exécution de la conversion de Java en objets SDO

Pour une substitution correcte du code généré ou l'identification des éventuelles exceptions d'exécution liées aux conversions de Java en SDO (Service Data Object), il est important de bien comprendre les règles en jeu. La plus grande partie des conversions se font directement, mais il existe des cas complexes où l'environnement d'exécution offre les meilleures possibilités de conversion du code généré.

Types et classes de base

L'environnement d'exécution effectue une conversion directe entre Service Data Objects et types et classes Java de base. Types et classes de base :

- Char ou `java.lang.Character`
- Boolean
- `Java.lang.Boolean`
- Byte ou `java.lang.Byte`
- Short ou `java.lang.Short`
- Int ou `java.lang.Integer`
- Long ou `java.lang.Long`
- Float ou `java.lang.Float`
- Double ou `java.lang.Double`
- `Java.lang.String`
- `Java.math.BigInteger`
- `Java.math.BigDecimal`
- `Java.util.Calendar`
- `Java.util.Date`
- `Java.xml.namespace.QName`
- `Java.net.URI`
- `Byte[]`

Classes et tableaux Java définis par l'utilisateur

Lors de la conversion d'une classe ou d'un tableau (array) Java en SDO, l'environnement d'exécution crée un objet de données possédant un URI généré en inversant le nom de package du type Java et ayant un type égal au nom de la classe Java. Par exemple, si la classe Java `com.ibm.xsd.Customer` est convertie en objet SDO, l'URI est `http://xsd.ibm.com` et le type est `Customer`. L'environnement d'exécution inspecte ensuite le contenu des membres de la classe Java et affecte les valeurs aux propriétés du SDO.

Lors de la conversion d'un SDO en un type Java, l'environnement d'exécution génère le nom du package en inversant l'URI et le nom du type est égal au type du SDO. Par exemple, l'objet de données de type `Customer` et dont l'URI est `http://xsd.ibm.com` génère une instance du module Java `com.ibm.xsd.Customer`. L'environnement d'exécution extrait ensuite les valeurs des propriétés du SDO et affecte ces propriétés aux zones de l'instance de la classe Java.

Lorsque la classe Java est une interface définie par l'utilisateur, vous devez substituer le code généré et offrir une classe concrète que l'environnement d'exécution puisse instancier. Si l'environnement d'exécution ne peut créer de classe concrète, une exception se produira.

Java.lang.Object

Si le type Java est `java.lang.Object`, le type généré est `xsd:anyType`. Un module peut appeler cette interface avec tout objet SDO. L'environnement d'exception tente d'instancier une classe concrète de la même façon que pour les classes et tableaux (arrays) Java définis par l'utilisateur lorsqu'il ne peut trouver cette classe. Autrement, l'environnement d'exécution passe le SDO à l'interface Java.

Même si la méthode renvoie un objet `java.lang.Object`, l'environnement d'exécution effectuera la conversion seulement en un SDO, si la méthode renvoie un type concret. L'environnement d'exécution emploie une conversion semblable à celle des classes et tableaux Java définis par l'utilisateur en SDO, tel que décrit dans le paragraphe suivant.

Lors de la conversion d'une classe ou d'un tableau (array) Java en SDO, l'environnement d'exécution crée un objet de données possédant un URI généré en inversant le nom de package du type Java et ayant un type égal au nom de la classe Java. Par exemple, si la classe Java `com.ibm.xsd.Customer` est convertie en objet SDO, l'URI est `http://xsd.ibm.com` et le type est `Customer`. L'environnement d'exécution inspecte ensuite le contenu des membres de la classe Java et affecte les valeurs aux propriétés du SDO.

Dans un cas ou l'autre, si l'environnement d'exécution est incapable d'accomplir la conversion, une exception se produit.

Classes de conteneur Java.util

Lors de la conversion en une classe de conteneur Java concrète telle que `Vector`, `HashMap`, `HashSet` et autres du même genre, l'environnement d'exécutioninstanciera la classe de conteneur appropriée. L'environnement d'exécution emploie une méthode semblable à celle des classes et tableaux Java définis par l'utilisateur pour renseigner la classe de conteneur. Si l'environnement d'exécution ne peut localiser de classe Java concrète, l'environnement d'exécution injectera la classe de conteneur dans le SDO.

Lors de la conversion de classes de conteneur Java concrètes en SDO, l'environnement d'exécution utilise les schémas générés montrés dans la conversion de «Java en XML.»

Interfaces Java.util

Pour certaines interfaces de conteneur du package java.util, l'environnement d'exécution instancie les classes concrètes suivantes :

Tableau 3. Conversion de type WSDL en classe Java

Interface	Classes concrètes par défaut
Collection	HashSet
Map	HashMap
List	ArrayList
Set	HashSet

Concepts associés

Programmation d'objets métier

L'infrastructure d'objets métier (BO) WebSphere Process Server est une extension de la norme SDO (Service Data Object). Cette section fournit des informations sur la programmation d'objets métier et SDO.

Validation de document XML

Il est possible de valider les documents XML et les objets métier à l'aide du service de validation.

Par ailleurs, d'autres services requièrent un certain nombre de standards minimum sinon une exception d'exécution est émise. `BOXMLSerializer` est l'un de ces services.

Vous pouvez utiliser `BOXMLSerializer` pour valider des documents XML avant qu'ils ne soient traités par une demande de service. `BOXMLSerializer` valide la structure des documents XML pour déterminer s'il comporte l'un des types d'erreur suivants :

- Documents XML non valides, comme ceux pour lesquels il manque certaines balises d'éléments.
- Documents XML syntaxiquement incorrects, comme ceux pour lesquels il manque des balises fermantes.
- Documents contenant des erreurs d'analyse syntaxique, comme les erreurs dans les déclarations d'entité.

Lorsque `BOXMLSerializer` identifie une erreur, une exception est émise avec une analyse détaillée de l'incident.

Vous pouvez valider l'importation et/ou l'exportation des documents XML pour les services suivants :

- HTTP
- Services Web JAXRPC
- Services Web JAX-WS
- Services JMS
- Services MQ

Pour les services HTTP, JAXRPC et JAX-WS, `BOXMLSerializer` génère les exceptions de la manière suivante :

- Importations –
 1. Le composant SCA appelle le service.
 2. Le service appelle l'URL d'une destination.
 3. L'URL cible répond avec une exception XML non valide.
 4. Le service échoue avec une exception d'exécution et un message.
- Exportations –
 1. Le client de service appelle l'exportation de service.
 2. Le client de service envoie un XML non valide
 3. L'exportation échoue pour le service et génère une exception et un message.

Pour les services de messagerie JMS et MQ, les exceptions sont générées de la manière suivante :

- Importations –
 1. L'importation appelle le service JMS ou MQ.
 2. Le service renvoie une réponse.
 3. Le service renvoie une exception XML non valide.
 4. L'importation échoue et génère un message.
- Exportations –
 1. Le client MQ ou JMS appelle une exportation.
 2. Le client envoie un XML non valide.
 3. L'exportation échoue et génère une exception et un message.

Vous pouvez afficher les journaux pour tout message généré par une exception de validation XML. Les exemples ci-dessous représentent des messages générés par du code XML incorrect qui a été validé par `BOXMLSerializer`

- Importation JAXWS

```
javax.xml.ws.WebServiceException: org.apache.axiom.om.OMException:
  javax.xml.stream.XMLStreamException: Element type "TestResponse" must be
  followed by either attribute specifications, ">" or "/>".
```

```
javax.xml.ws.WebServiceException: org.apache.axiom.soap.SOAPProcessingException:
  First Element must contain the local name, Envelope
```

- Importation JAXRPC

```
[9/11/08 15:16:27:417 CDT] 0000003e ExceptionUtil E
CNTR0020E: EJB threw an unexpected (non-declared)
exception during invocation of method
"transactionNotSupportedActivitySessionNotSupported" on bean
"BeanId(WXMLValidationApp#WXMLValidationEJB.jar#Module, null)".
Exception data: WebServicesFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
faultString: org.xml.sax.SAXParseException: Element type "TestResponse"
must be followed by either
attribute specifications, ">" or "/>". Message being parsed:
  <?xml version="1.0"?><TestResponse
  xmlns="http://WXMLValidation"><firstName>Bob</firstName>
  <lastName>Smith</lastName></TestResponse>
faultActor: null
faultDetail:
[9/11/08 15:16:35:135 CDT] 0000003f ExceptionUtil E CNTR0020E: EJB threw an
unexpected (non-declared) exception during invocation of method
"transactionNotSupportedActivitySessionNotSupported" on bean
"BeanId(WXMLValidationApp#WXMLValidationEJB.jar#Module, null)".
Exception data: WebServicesFault
```

```
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
faultString: org.xml.sax.SAXException: WSWS3066E: Error: Expected 'envelope'
but found TestResponse
Message being parsed: <?xml version="1.0"?><TestResponse
xmlns="http://WSXMLValidation">
<firstName>Bob</firstName><middleName>John</middleName>
<lastName>Smith</lastName>
</TestResponse>
faultActor: null
faultDetail:
```

- Exportation JAXRPC/JAXWS
[9/11/08 15:35:13:401 CDT] 00000064 WebServicesSe E
com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
getSoapAction WSWS3112E:
Error: Generating WebServicesFault due to missing SOAPAction.
WebServicesFault
faultCode: Client.NoSOAPAction
faultString: WSWS3147E: Error: no SOAPAction header!
faultActor: null
faultDetail:

Pour plus d'informations sur les services de validation, voir l'interface de `B0InstanceValidator` dans la documentation sur l'interface SPI et l'interface de programme d'application générées de la section Référence.

Concepts associés

Guides et techniques de programmation

Cette section comprend des guides et des exemples de programmation.

Propagation d'en-tête de protocole à partir de liaisons d'exportation non SCA

Le service de contexte est chargé de la propagation du contexte (y compris les en-têtes de protocole comme l'en-tête JMS et le contexte utilisateur comme l'ID de compte) tout au long du chemin d'appel SCA (Service Component Architecture). Le service de contexte offre un ensemble d'API et de paramètres configurables.

Lorsque la propagation du service de contexte est bidirectionnelle, le contexte de réponse écrasera systématiquement le contexte en cours. Lorsque vous exécutez un appel d'un composant SCA à un autre, la réponse présentera un contexte différent. Un composant de service aura un contexte entrant, mais si vous appelez un autre service, celui-ci écrasera le contexte sortant d'origine. Le contexte de réponse deviendra alors le nouveau contexte.

Lorsque la propagation du service de contexte est unidirectionnelle, le contexte d'origine reste à l'identique.

Le cycle de vie du service de contexte est associé à un appel. Une demande dispose d'un contexte associé et le cycle de vie de ce contexte est lié au traitement de cette demande particulière. Lorsque le traitement de cette demande se termine, le cycle de vie de ce contexte s'achève.

Dans le cas d'un processus BPEL (Business Process Execution Language) à court terme, le contexte de réponse écrasera le contexte de demande. Il récupérera le contexte de réponse auprès de la première demande et le passera à la demande suivante. Dans le cas d'un processus BPEL au long cours, le contexte de réponse est éliminé par le framework BPEL. Celui-ci stocke le contexte d'origine et utilise ce contexte pour faire d'autres appels sortants.

Exemple

Exemple : un contexte incluant un en-tête de protocole est propagé à travers tout le chemin d'appel partant d'une demande entrant dans BPEL en provenance d'un service web SOAP. BPEL traite ce contexte, et les appels en provenance de BPEL sont effectués séquentiellement vers une liaison de service web sortante, puis une autre liaison de service web sortante. Une demande issue du service web SOAP utilise le service de contexte pour transmettre l'en-tête de protocole. Le service de contexte est extrait de la demande entrante et l'en-tête de protocole est transmis à l'extérieur.

Vous pourrez voir le même type de comportement avec un autre composant SCA en lieu et place du BPEL de cet exemple.

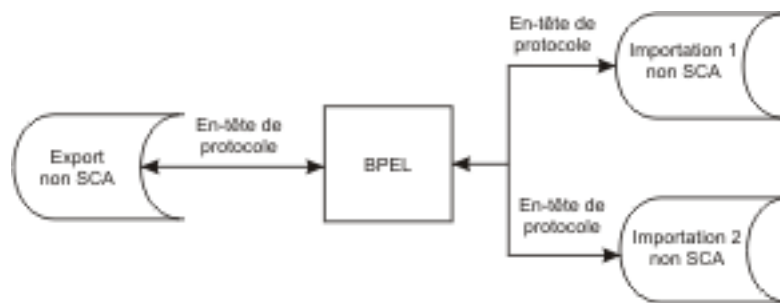


Figure 9. Propagation de contexte comprenant un en-tête de protocole

Voici un exemple de code.

```
//Import the necessary classes;
import com.ibm.bpm.context.ContextService;
import com.ibm.websphere.sca.ServiceManager;
import com.ibm.bpm.context.cobo.ContextObject;
import com.ibm.bpm.context.cobo.ContextObjectFactory;
import com.ibm.bpm.context.cobo.HeaderInfoType;
import com.ibm.bpm.context.cobo.UserDefinedContextType;

//Locate ContextService;
ContextService contextService = (ContextService)ServiceManager.INSTANCE.locateService
("com/ibm/bpm/context/ContextService");

// Get header info
HeaderInfo headerInfo = contextService.getHeaderInfo();
// Get user defined context in current execution context
UserDefinedContextType userDefinedContext = contextService.getUserDefinedContext();
if(userDefinedContext == null){ // create a new context if context is null
userDefinedContext = ContextObjectFactory.eINSTANCE.createUserDefinedContextType()
}

// Do some modification to header info and userDefinedContext

// Set user defined context back to the current execution context.
contextService.setUserDefinedContext(userDefinedContext);

// Set header info back to the current execution context.
contextService.setHeaderInfo(headerInfo);
```

Remarque : Dans le composant de flux de médiation, les API ContextService ne doivent pas être employées. Utilisez le modèle de programmation SMO pour accéder au contexte.

Les services de contexte possèdent des règles et des tables configurables qui dictent le comportement de la liaison. Pour plus d'informations, voir la documentation API et SPI générées disponible à la section Référence. Lors du développement dans WebSphere Integration Developer, vous pouvez définir le service de contexte sur les propriétés d'importation ou d'exportation. Pour plus de détails, reportez-vous aux informations relatives aux liaisons d'importation et d'exportation dans le centre de documentation de WebSphere Integration Developer.

Concepts associés

Guides et techniques de programmation

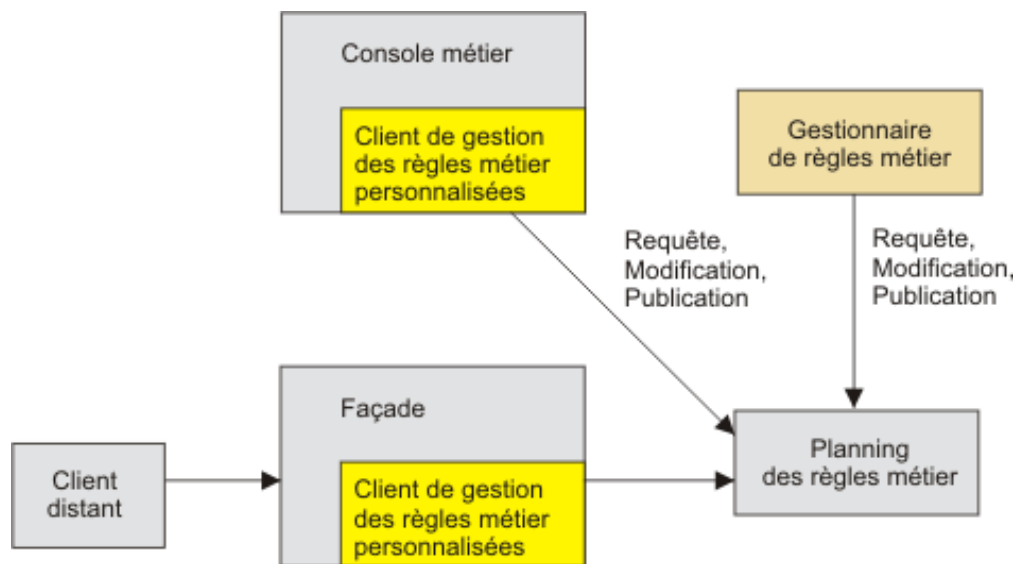
Cette section comprend des guides et des exemples de programmation.

Gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Les classes de gestion des règles métier peuvent être utilisées dans une application Web, où elles sont combinées à d'autres capacités de gestion pour des processus métier ou des tâches utilisateur, afin de gérer tous les composants d'un même client. Vous pouvez utiliser tout client de gestion personnalisé avec l'application Web Business Rule Manager contenue dans WebSphere Process Server. Les classes peuvent également être utilisées pour l'automatisation des modifications apportées aux règles métier au sein d'une application. Par exemple, certaines règles métier peuvent être modifiées si les résultats d'un processus métier utilisant ces règles dépassent un seuil ou une limite spécifique.

Les classes de gestion des règles métier doivent être utilisées dans une application installée sur WebSphere Process Server. Les classes n'incluent pas d'interface distante, mais elles peuvent être encapsulées dans une façade, qui est ensuite exposée via un protocole spécifique, à des fins d'exécution à distance.



Ce guide de programmation se compose de deux sections principales et d'une annexe. La première section explique le modèle de programmation, et indique comment utiliser les différentes classes. Des diagrammes de classes sont fournis pour illustrer les relations existant entre les classes. La deuxième section contient des exemples d'utilisation des classes pour l'exécution d'opérations telles que la recherche de groupes de règles métier, la planification de la destination d'une nouvelle règle, ou encore la modification d'un ensemble de règles ou d'une table de décision. L'annexe contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations courantes, et d'autres exemples de création de requêtes complexes servant à rechercher des groupes de règles métier en utilisant des caractères génériques.

Ce guide de programmation consacré aux classes est également disponible au format HTML Javadoc inclus dans WebSphere Process Server v6.1 et dans l'environnement de test de WebSphere Integration Developer v6.1. Cette documentation Javadoc est figure dans le répertoire `{Répertoire d'installation de WebSphere Process Server}\web\apidocs` ou dans `{Répertoire d'installation de WebSphere Integration Developer}\runtimes\bi_v61\web\apidocs`. Les packages `com.ibm.wbiserver.brules.mgmt.*` contiennent toutes les informations.

Concepts associés

Guides et techniques de programmation

Cette section comprend des guides et des exemples de programmation.

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Référence associée

Classes d'opérations communes

Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Le partage d'un même modèle a été jugé essentiel pour des raisons de maintenance future, et également dans le but d'offrir à l'utilisateur final un modèle de programmation cohérent. Le partage de ce modèle a nécessité des compromis entre les besoins d'outils, l'exécution et la création : en effet, tous ces aspects possèdent leurs propres exigences en fonction de leur environnement respectif ; or, ces exigences entraînent parfois en conflit. Les artefacts décrits ci-dessous en tant que partie intégrante du modèle de programmation global représentent un équilibre entre toutes les exigences de ces environnements différents.

La modification des règles métier est limitée aux seuls éléments définis à l'aide de modèles dans les ensembles de règles, dans les tables de décision et dans la table de sélection des opérations (dates d'entrée en vigueur et cibles). La création de nouveaux ensembles de règles et de nouvelles tables de décisions n'est prise en charge que via la copie d'un ensemble de règles ou d'une table de décision existant(e). Le composant de groupe de règle métier lui-même ne peut pas être créé dynamiquement lors de l'exécution, à l'exception des propriétés définies par l'utilisateur et des valeurs de description. Pour apporter les modifications requises au composant (ajout d'une nouvelle opération, par exemple), vous devez utiliser WebSphere Integration Developer, puis redéployer ces modifications ou les réinstaller sur le serveur.

Concepts associés

Gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Groupe de règles métier

La classe `BusinessRuleGroup` représente le composant de groupe de règles métier. Cette classe peut être considérée comme l'objet racine contenant les ensembles de règles et les tables de décision.

Propriétés de groupes de règles métier

Les propriétés des groupes de règles métier servent à gérer ces groupes. Les propriétés définies dans les groupes de règles métier peuvent être utilisées dans les requêtes, pour renvoyer uniquement un sous-ensemble de groupes de règles métier à afficher puis à modifier.

Opération

Les opérations représentent le point de départ d'accès aux ensembles de règles et aux tables de décisions à modifier. Les opérations d'un groupe de règles métier correspondent aux opérations répertoriées dans le langage WSDL associé au composant de groupes de règles métier.

Règle métier

Les classes `RuleSet` et `DecisionTable` sont basées sur une classe générique `BusinessRule` et contiennent des méthodes fournissant les informations disponibles dans les ensembles de règles et les tables de décision.

Ensemble de règles

Un ensemble de règles constitue un type de règle métier. Les ensembles de règles sont généralement utilisés lorsque plusieurs règles doivent être exécutées sur la base de différentes valeurs conditionnelles. Les ensembles de règles se composent d'un bloc de règles et de modèles de règles. Le bloc de règles (`RuleBlock`) contient les différentes règles if-then et action qui composent la logique de l'ensemble de règles.

table de décision

Les tables de décision représentent un autre type de règle métier que vous pouvez gérer et modifier. Elles sont généralement utilisées lorsque de nombreuses conditions doivent être évaluées et qu'un ensemble spécifique d'actions doivent être émises une fois les conditions remplies.

Modèles et paramètres

Les modèles inclus dans les ensembles de règles et dans les tables de décision prennent comme base une définition commune. Les modèles possèdent des paramètres et une présentation de l'utilisateur. Les valeurs de paramètres inclus dans les modèles sont définis pour permettre d'apporter des modifications à la règle une fois que celle-ci a été déployée.

Validation

Parmi les objets principaux, nombreux sont ceux qui possèdent une méthode de validation ; elle permet de vérifier si les artefacts sont corrects et complets avant leur publication.

Suivi des modifications

Pour tous les objets, vous pouvez utiliser une méthode `hasChanges` afin de vérifier si des modifications ont été apportées à l'objet et aux objets qu'il contient.

`BusinessRuleManager`

La classe `BusinessRuleManager` est la principale classe d'utilisation des groupes de règles, des ensembles de règles et des tables de décision.

Traitement des exceptions

Des exceptions peuvent être générées lors d'un appel de validation pour un

artefact ou lors de sa publication. En cas d'erreur de validation, l'exception `ValidationException` est générée ; elle s'accompagne de la liste des problèmes rencontrés. Si un problème survient au cours de la publication car une autre transaction publie les mêmes artefacts, l'exception `ChangeConflictException` est générée. A chaque détection de la modification d'un artefact par une autre transaction, l'exception `ChangeConflictException` est générée.

Autorisation

Les classes ne prennent en charge aucun niveau d'autorisation. L'application client utilisant les classes doit ajouter sa propre méthode d'autorisation.

Groupe de règles métier

La classe `BusinessRuleGroup` représente le composant de groupe de règles métier. Cette classe peut être considérée comme l'objet racine contenant les ensembles de règles et les tables de décision.

Les ensembles de règles et les tables de décision sont accessibles uniquement par le groupe de règles métier auquel elles sont associées. La classe contient des méthodes permettant d'extraire les informations liées au groupe de règles métier et d'accéder aux ensembles de règles et aux tables de décision. Les méthodes permettent d'extraire les informations suivantes :

- Espace de nom cible
- Nom de groupe de règles métier
- Nom affiché
- Synchronisation nom/nom affiché
- Description
- Fuseau horaire de présentation indiquant si les dates doivent être affichées au format UTC (temps universel coordonné) ou en local sur le système
- Opérations définies dans l'interface associée au groupe de règles métier
- Propriétés personnalisées définies dans le groupe de règles métier

Les différents ensembles de règles et tables de décision associés au groupe de règles métier sont accessibles par l'opération du groupe de règles métier.

Des méthodes permettent également de mettre à jour les informations dans le groupe de règles métier. Les informations suivantes peuvent être mises à jour via les méthodes :

- Description
- Nom affiché
- Synchronisation nom/nom affiché
- Propriétés personnalisées définies dans le groupe de règles métier

Le nom affiché du groupe de règles métier peut être défini de manière explicite ou sur la valeur du nom à l'aide de la méthode `setDisplaynameIsSynchronizedToName`.

Les autres valeurs ne peuvent pas être modifiées puisqu'elles font partie de la définition du composant de groupe de règles métier. Leur modification nécessiterait un redéploiement ainsi qu'une réinstallation.

La classe du groupe de règles métier offre également une méthode d'actualisation. Cette méthode effectue un appel vers la mémoire persistante ou le référentiel dans lesquels les règles métier sont stockées et renvoie le groupe de règles métier ainsi que tous les ensembles de règles et les tables de décision avec les informations conservées. Le groupe de règles métier renvoyé représente la dernière copie et l'objet précédent devient obsolète.

La méthode `isShell` permet de dire si la version d'une instance de groupe de règles métier est prise en charge par l'exécution en cours. Par exemple, si un client Web a été créé avec les classes de gestion de règles métier en cours, et que de nouvelles fonctions ajoutées ultérieurement au groupe de règles métier ne sont pas prises en charge par les classes, un groupe de règles métier interpréteur de commandes est créé une fois le groupe de règles métier récupéré. Cela permet au client Web de continuer à utiliser les règles métier prises en charge et à récupérer les groupes de règles métier avec des fonctions et des attributs limités. Lorsque la méthode `isShell` est vraie, seules les méthodes `getName`, `getTargetNameSpace`, `getProperties`, `getPropertyValue` et `getProperty` renvoient des valeurs. Toutes les autres méthodes conduisent à l'exception `UnsupportedOperationException`. Outre l'utilisation de la méthode `isShell`, le type de `BusinessRuleGroup` peut également être vérifié s'il s'agit d'une instance de `BusinessRuleGroupShell`, afin de déterminer si la version est prise en charge.

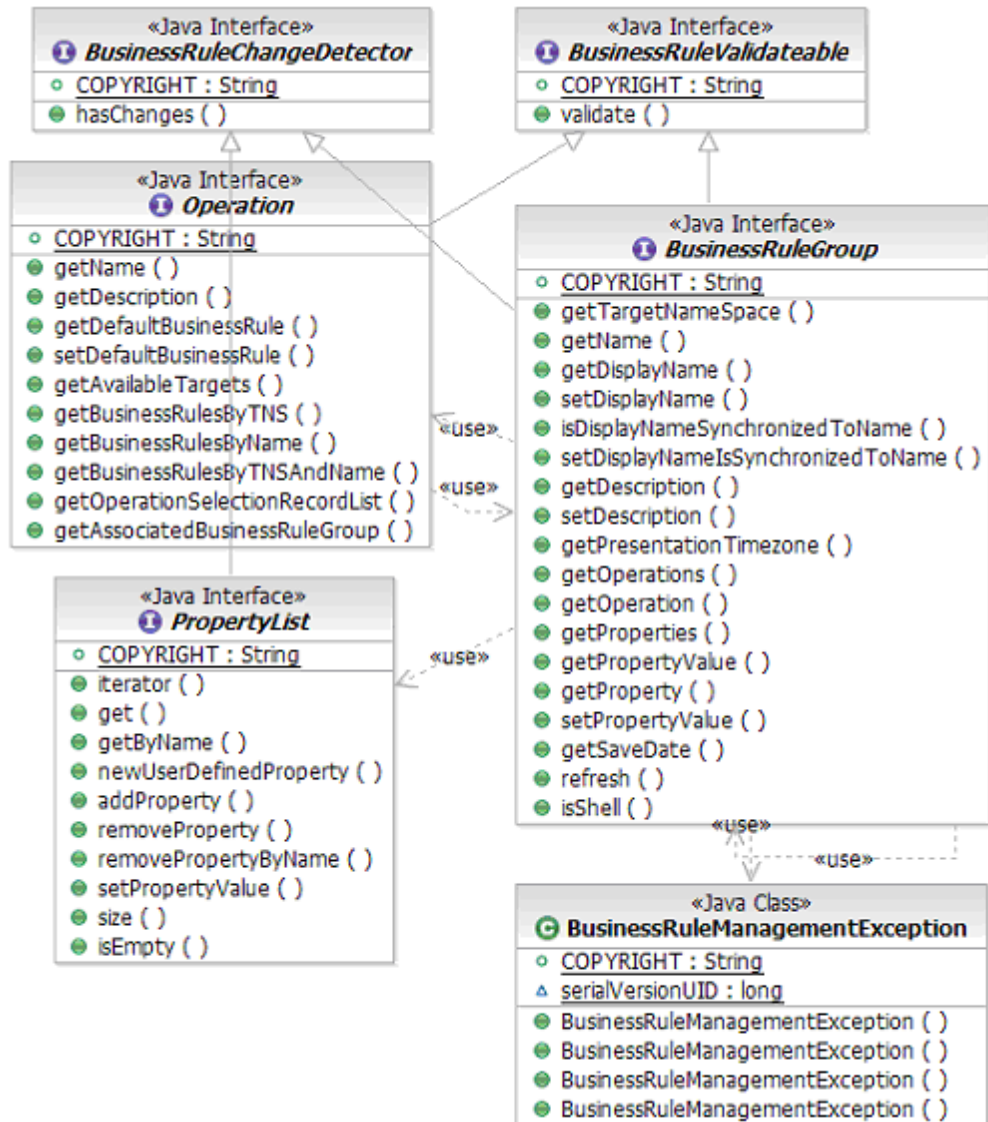


Figure 10. Diagramme de classes de BusinessRuleGroup et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Propriétés de groupes de règles métier

Les propriétés des groupes de règles métier servent à gérer ces groupes. Les propriétés définies dans les groupes de règles métier peuvent être utilisées dans les requêtes, pour renvoyer uniquement un sous-ensemble de groupes de règles métier à afficher puis à modifier.

Toutes les propriétés sont du type chaîne et sont définies en tant que paires valeur-nom. Chaque propriété ne peut être définie qu'une seule fois dans un groupe de règles métier. Pour chaque propriété définie, une valeur doit également lui être définie. La valeur de propriété peut être une chaîne vide ou de longueur zéro, mais pas NULL. Définir une propriété sur NULL revient à la supprimer.

Les propriétés d'un groupe de règles métier sont également accessibles dans un ensemble de règles ou une table de décision au moment de l'exécution. Cela permet à une valeur unique, à définir dans le groupe de règles métier, d'être utilisée au sein de plusieurs ensembles de règles ou de tables de décision du groupe de règles métier. Seules les propriétés définies dans le groupe de règles métier sont disponibles pour les ensembles de règles et les tables de décision joints.

Il existe deux types de propriétés, système et définies par l'utilisateur. Le nombre de propriétés système ou de propriétés définies par l'utilisateur n'est pas limité dans un groupe de règles métier. Les propriétés système permettent de détenir des informations spécifiques liées à un composant telles que la version du modèle de règle utilisée lors de la définition de la logique de règle. Ces informations système apparaissent dans les propriétés pour permettre les requêtes sur ces zones. Les propriétés système commencent par un préfixe IBMSysSystem et sont en lecture seule dans le groupe de règles métier et les classes de propriétés. Les propriétés système peuvent être ajoutées, modifiées ou supprimées. Voici un exemple de propriété système :

Nom de la propriété	Valeur de la propriété
IBMSysSystemVersion	6.2.0

Remarque : les valeurs du nom, de l'espace de nom et du nom affiché d'un groupe de règles métier sont traitées en tant que propriétés système dans le cadre de requêtes, et font partie de la liste de propriétés à extraire pour un groupe de règles métier à l'aide de la méthode `getProperties`. Toutefois, ces propriétés ne sont pas définies en tant qu'éléments de propriétés en cours dans l'artefact de groupe de règles métier et n'apparaissent pas comme propriétés dans WebSphere Integration Developer, dans la mesure où elles sont définies avec des éléments uniques et distincts dans le groupe de règles métier. Elles sont fournies uniquement pour offrir davantage d'options de requête.

Les propriétés définies par l'utilisateur peuvent être utilisées pour détenir des informations spécifiques aux utilisateurs, ainsi que dans les requêtes relatives aux groupes de règles métier. Ces propriétés sont disponibles en lecture-écriture.

Les propriétés d'un groupe de règles métier peuvent être extraites individuellement ou sous forme de liste (objet `PropertyList`). Avec l'onglet `PropertyList`, les méthodes de récupération des propriétés individuelles, d'ajout et de suppression des propriétés définies par l'utilisateur sont fournies.

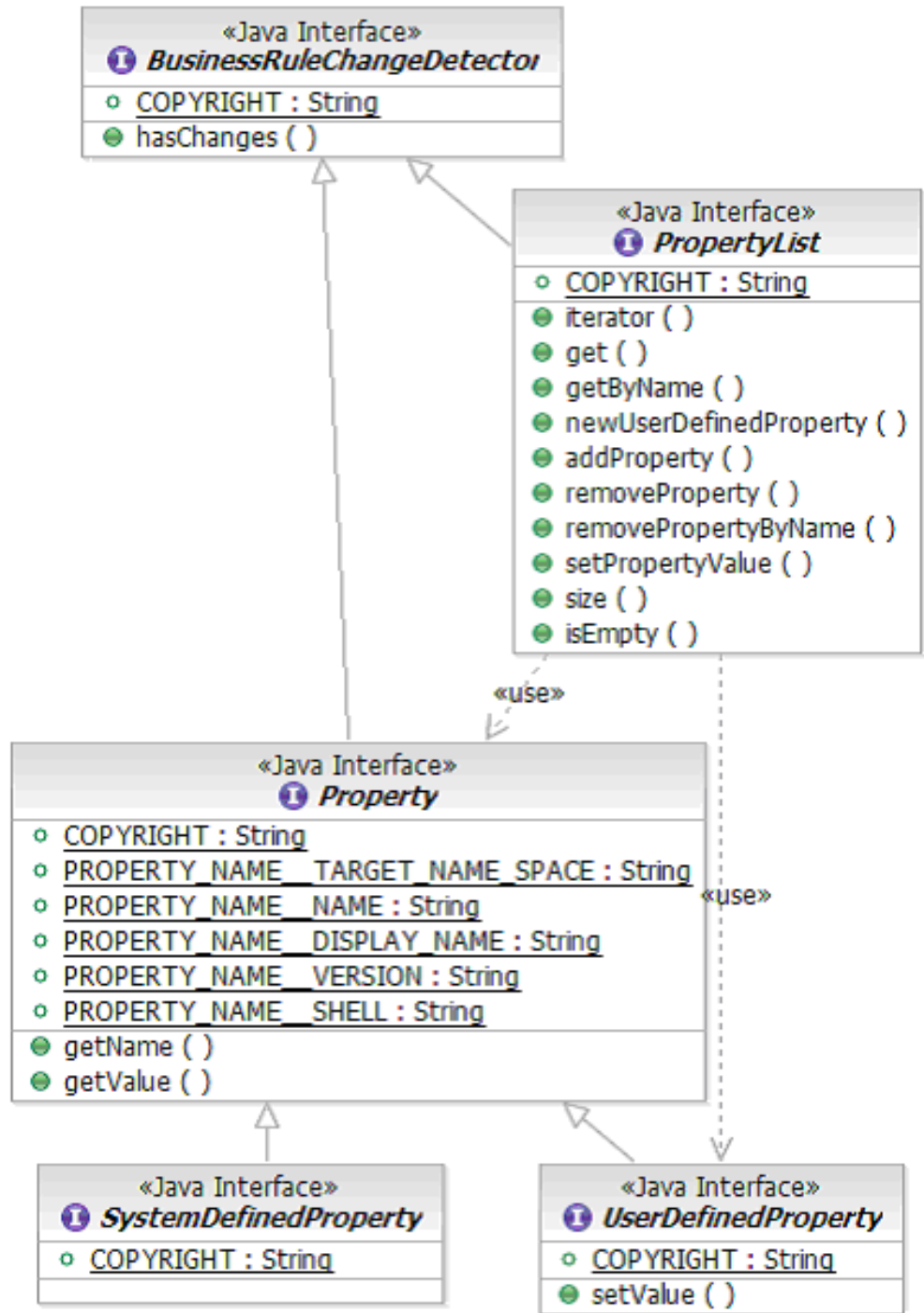


Figure 11. Diagramme de classes de Property et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Opération

Les opérations représentent le point de départ d'accès aux ensembles de règles et aux tables de décisions à modifier. Les opérations d'un groupe de règles métier correspondent aux opérations répertoriées dans le langage WSDL associé au composant de groupes de règles métier.

Pour chaque opération, il existe différentes cibles, chacune d'entre elles constituant une règle métier (ensemble de règles ou table de décision) :

- Cible par défaut (facultatif)
- Liste des cibles planifiées par plages de date/heure (OperationSelectionRecord)
- Liste de toutes les cibles disponibles pouvant être utilisées pour cette opération

Pour chaque opération, une cible de règle métier doit être spécifiée au minimum. Cette cible peut être OperationSelectionRecord et comporter une date de début et une date de fin spécifiques, correspondant à la période d'activation planifiée de la cible. Une cible unique par défaut peut également être définie pour l'opération, puis utilisée au cours de l'exécution si aucune cible de règle métier planifiée correspondante n'est trouvée. La classe Operation fournit des méthodes d'extraction et de définition de cible de règle métier par défaut, ainsi que des méthodes d'extraction de la liste (OperationSelectionRecordList) des cibles de règles métier planifiées. Outre la cible de règle métier par défaut et les cibles de règles métier planifiées, il existe une liste de toutes les cibles de règles métier disponibles pour l'opération. Cette liste répertorie les cibles de règles métier planifiées, la cible de règle métier par défaut, ainsi que les autres ensembles de règles ou tables de décisions non planifiés pour cette opération. Un ensemble de règles ou une table de décision non planifié(e) est associé(e) à l'opération via la liste des cibles disponibles, car elle partage implicitement les informations relatives à l'opération. Toutes les cibles de règles métier doivent prendre en charge les messages entrants et sortants de leur opération. Chaque opération étant unique sur une interface donnée, les ensembles de règles et les tables de décisions d'une opération sont uniques.

Vous pouvez planifier l'activation des ensembles de règles et tables de décisions de la liste des cibles disponibles via la création d'une méthodeOperationSelectionRecord. Dans ce cas, vous devez spécifier une date de début et une date de fin pour chaque ensemble de règles ou table de décision de la liste des cibles disponibles. La date de début doit être antérieure à la date de fin. Ces dates peuvent représenter une période incluant la date du jour, ou encore une période passée ou future. La période indiquée par ces dates ne peut pas chevaucher une autre période spécifiée par OperationSelectionRecords, une fois ajoutée à OperationSelectionRecordList et publiée. Les valeurs de date de début et de date de fin sont de type java.util.Date. Les valeurs spécifiées seront considérées comme des valeurs UTC, selon la classe java.util.Date. Une fois OperationSelectionRecord terminée, elle peut être ajoutée à OperationSelectionRecordList en vue d'être planifiée avec d'autres cibles de règles métier. Il peut exister des écarts entre les périodes spécifiées par différentes méthodes OperationSelectionRecords. Lorsqu'un écart est constaté au cours de

l'exécution, la cible par défaut est utilisée. Si aucune cible par défaut n'a été spécifiée, une exception est générée. Il est recommandé de toujours spécifier une cible de règle métier par défaut.

Une cible de règle métier par défaut peut être supprimée de la liste des cibles planifiées, via la suppression de la méthode `OperationSelectionRecord` de `OperationSelectionRecordList`. Si vous supprimez un élément `OperationSelectionRecord`, cela ne supprime pas la cible de règle métier correspondante de la liste des cibles de règles métier disponibles, et cela ne supprime pas non plus les autres éléments `OperationSelectionRecords` portant la même cible de règle métier planifiée.

Outre l'extraction d'un ensemble de règles ou d'une table de décision via la méthode `OperationSelectionRecordList` ou via la liste des cibles disponibles, la classe `Operation` permet également d'extraire les cibles de règles métier par nom et par valeur de propriété d'espace de nom cible. Grâce aux méthodes de la classe `Operation`, les ensembles de règles et tables de décisions qui figurent parmi les cibles disponibles pour cette opération peuvent faire l'objet d'une requête. Les ensembles de règles et tables de décisions susceptibles de porter des valeurs de nom et d'espace de nom cible correspondantes, mais qui font partie des listes des cibles disponibles d'autres opérations ne sont pas inclus dans l'ensemble de résultats. Les méthodes `getBusinessRulesByName`, `getBusinessRulesByTNS` et `getBusinessRulesByTNSAndName` sont fournies pour simplifier l'extraction d'ensembles de règles et de tables de décisions spécifiques.

La classe `Operation` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom de l'opération
- Extraction de la description de l'opération
- Extraction et définition de la cible de règle métier par défaut
- Extraction des cibles de règles métier planifiées (`OperationSelectionRecordList`)
- Extraction de la liste de toutes les cibles de règles métier disponibles
- Extraction d'un ensemble de règles ou d'une table de décision de la liste des cibles disponibles, par nom ou par espace de nom cible
- Extraction du groupe de règles métier associé à l'opération

La classe `OperationSelectionRecordList` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction d'un élément de la classe `OperationSelectionRecord` par valeur d'index
- Suppression d'un élément spécifique de la classe `OperationSelectionRecord` par valeur d'index
- Ajout d'un nouvel élément de la classe `OperationSelectionRecord` à la liste

La classe `OperationSelectionRecord` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction et définition de la date de début
- Extraction et définition de la date de fin
- Extraction et définition de la cible de règle métier
- Extraction de l'opération à laquelle l'élément de la classe `OperationSelectionRecord` est associé

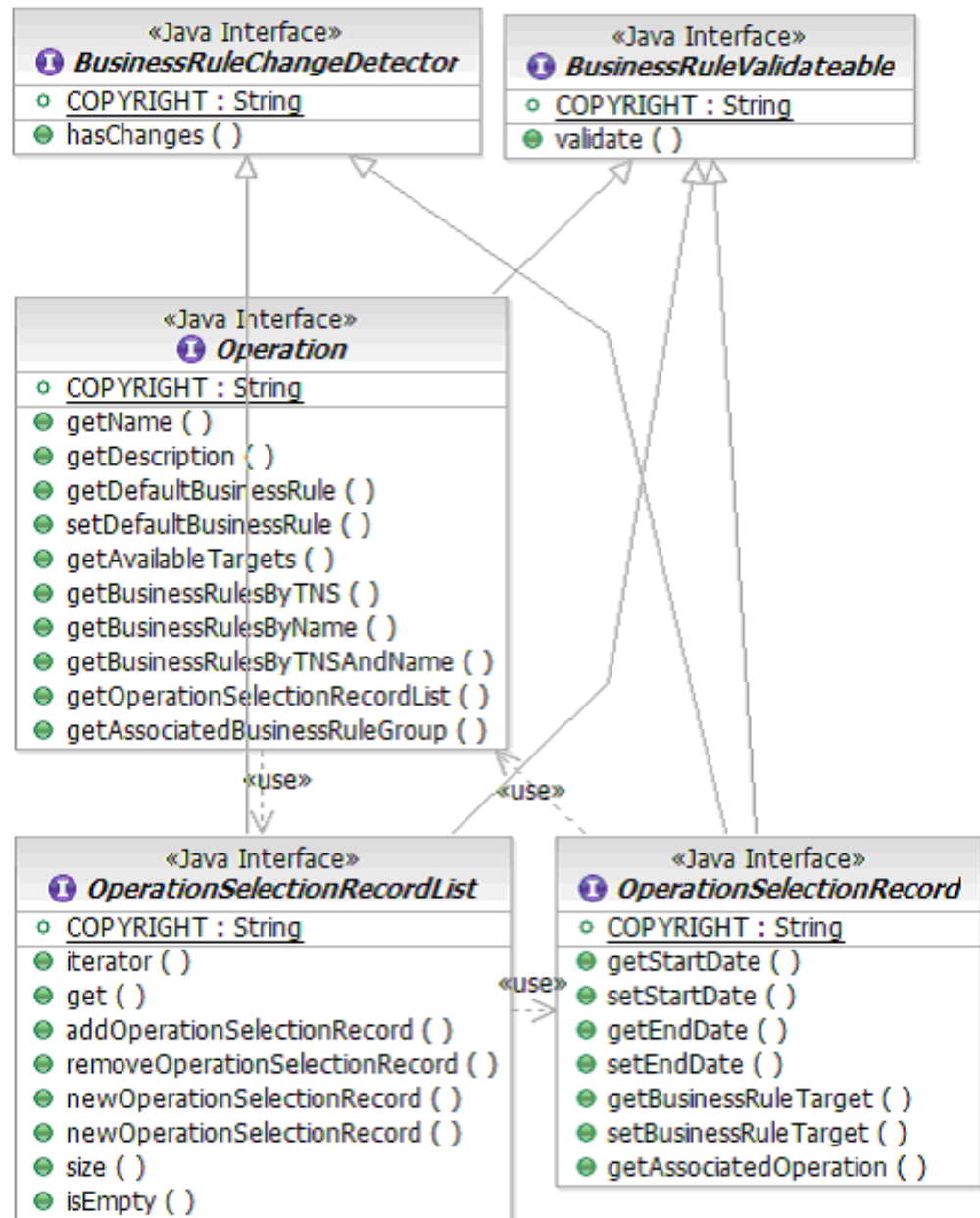


Figure 12. Diagramme de classes de Operation et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Règle métier

Les classes RuleSet et DecisionTable sont basées sur une classe générique BusinessRule et contiennent des méthodes fournissant les informations disponibles dans les ensembles de règles et les tables de décision.

A l'instar des artefacts de groupe de règles métier, les ensembles de règles et les tables de décision possèdent un nom et un espace de nom cible. La combinaison de ces valeurs doit être unique par rapport aux autres ensembles de règles et tables

de décision. Par exemple, deux ensembles de règles peuvent partager la même valeur d'espace de nom cible, mais leur nom doit être différent. De même, un ensemble de règles et une table de décision peuvent porter le même nom mais ils doivent détenir des valeurs d'espace de nom cible différentes.

La copie d'une règle métier peut être réalisée à partir d'une règle métier existante lorsqu'une règle similaire doit être planifiée à une heure spécifique, avec différentes valeurs de paramètres pour les règles construites à partir de modèles. Dans la mesure où une classe de sauvegarde Java est nécessaire à l'implémentation de la règle métier, les règles ne peuvent pas être créés à partir de rien. La classe de sauvegarde Java est créée seulement au moment du déploiement. Lors de la création d'une règle, cette dernière est ajoutée à la liste des cibles disponibles pour l'opération associée à la règle d'origine. Toutefois, la règle additionnelle n'est pas conservée sauf en cas de publication du groupe de règles métier auquel l'opération est associée.

La nouvelle règle métier doit comporter un espace de nom cible ou un nom différent de la règle d'origine. Le nom affiché de la nouvelle règle métier peut rester identique à celui de la règle d'origine puisque la combinaison du nom et de l'espace de nom fournissent une valeur clé permettant d'identifier la règle métier. Dans le cadre de la règle métier, les différentes valeurs de paramètre, précédemment définies avec un modèle, peuvent être modifiées. La planification de la règle métier à une heure spécifique peut être réalisée avec `OperationSelectionRecordList` ou en tant que destination par défaut avec l'Opération associée à la règle métier.

La classe `BusinessRule` fournit des méthodes permettant de :

- Extraire l'espace de nom cible
- Extraire le nom de l'ensemble de règles ou la table de décision
- Extraire et définir le nom affiché de l'ensemble de règles ou de la table de décision
- Extraire le type de la règle métier : ensemble de règles ou table de décision
- Extraire et définir la description de la règle métier
- Extraire l'opération à laquelle la règle métier est associée.
- Créer une copie de la règle métier avec un nom et/ou un espace de nom cible différent

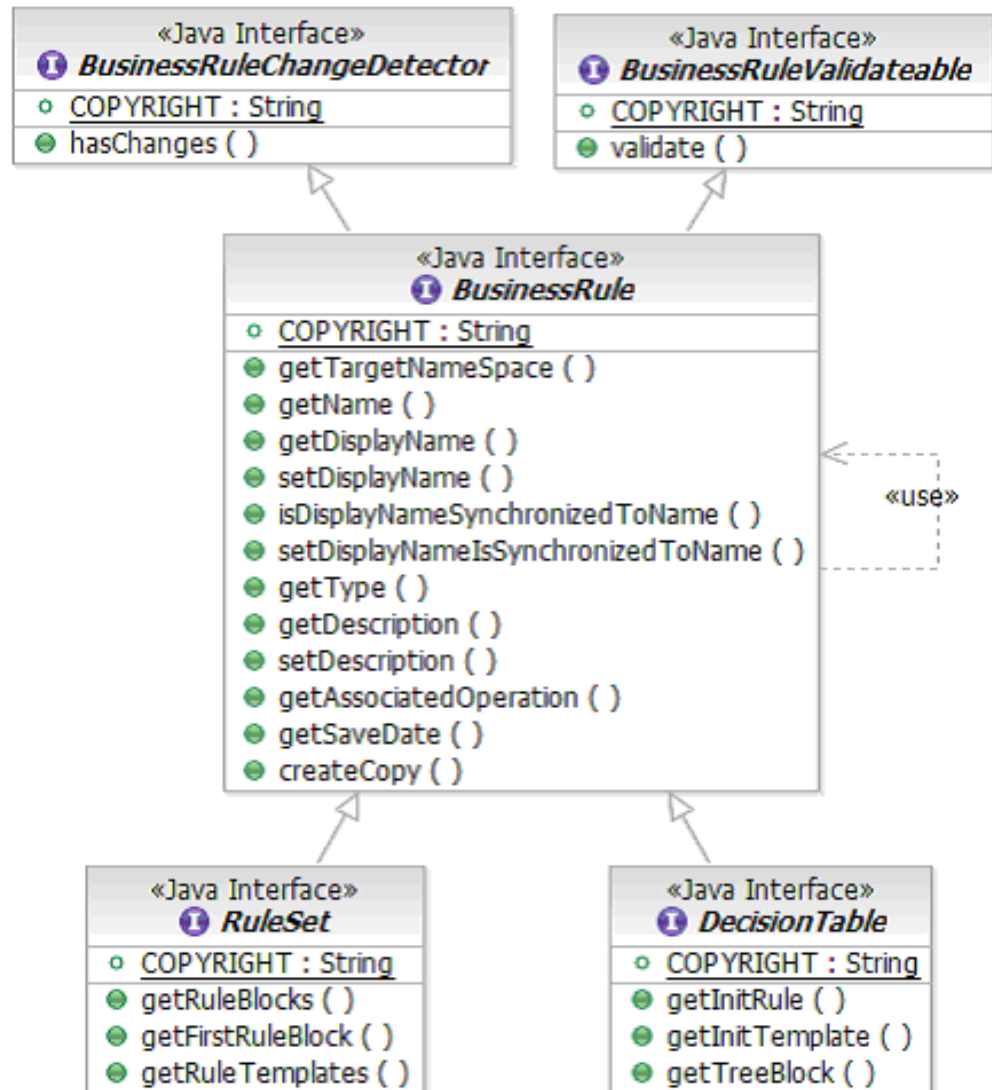


Figure 13. Diagramme de classes de BusinessRule et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Ensemble de règles

Un ensemble de règles constitue un type de règle métier. Les ensembles de règles sont généralement utilisés lorsque plusieurs règles doivent être exécutées sur la base de différentes valeurs conditionnelles. Les ensembles de règles se composent d'un bloc de règles et de modèles de règles. Le bloc de règles (RuleBlock) contient les différentes règles if-then et action qui composent la logique de l'ensemble de règles.

La classe RuleSet fournit les méthodes qui prennent en charge les aspects suivants :

- Extraction d'une liste de blocs de règles pour l'ensemble de règles

- Extraction d'une liste de modèles de règles définis dans l'ensemble de règles

A l'heure actuelle, chaque ensemble de règles ne peut contenir qu'un bloc de règles, tandis que plusieurs modèles de règles peuvent être définis dans l'ensemble de règles. Le bloc de règles contient l'ensemble de règles qui sera exécuté lors de l'appel de l'ensemble de règles. Le bloc de règles permet de modifier l'ordre des règles. Un bloc de règles doit contenir au minimum une règle définie. Les règles (`Rule`) peuvent être définies comme des règles d'instance de modèle (`TemplateInstanceRule`) ou codées en dur. Si une règle if-then ou une règle action a été définie avec un modèle, elle peut être supprimée du bloc de règles. Si une nouvelle instance de règle a été créée avec un modèle, elle peut être ajoutée au bloc de règles.

Si une règle est codée en dur et qu'elle n'a pas été définie avec un modèle, elle ne peut être ni modifiée, ni supprimée du bloc de règles. Ces règles ont été conçues pour faire systématiquement partie de la logique des ensembles de règles et ne doivent pas être modifiées ou répétées au sein de cette logique.

Lorsqu'une nouvelle règle est créée avec un modèle, elle doit porter une valeur de nom unique. La liste des règles existantes peut être extraite et vérifiée avant la création de la règle.

Pour les règles codées en dur if-then et action, seuls le nom et la présentation peuvent être extraits. La présentation représente une chaîne que vous pouvez utiliser pour afficher les informations relatives à la règle dans les applications client. Pour les règles if-then ou action définies avec un modèle, vous pouvez extraire le nom et la présentation, ainsi que des informations supplémentaires. Les valeurs de paramètres spécifiques peuvent être extraites et modifiées. Si un modèle (`RuleSetRuleTemplate`) a été défini dans l'ensemble de règles, vous pouvez créer une autre instance de la règle au sein de l'ensemble de règles et définir des valeurs de paramètres. Par exemple, une règle peut indiquer qu'un client d'un niveau spécifique doit recevoir une remise d'un montant donné. Cette logique peut être définie avec un modèle de règle unique, puis répétée en modifiant les valeurs des paramètres de niveau de client (or, argent, bronze, etc.), et de montant de la remise (15 %, 10 %, 5 %, etc.).

Les paramètres d'une règle ayant été définis avec un modèle sont propres à l'instance de règle correspondante. Le modèle définit uniquement une présentation standard, ainsi que le nombre de paramètres applicables à la règle. Chaque règle définie avec un modèle peut posséder des valeurs différentes, comme l'explique l'exemple de remises appliquées à différents niveaux de clients.

La classe `RuleBlock` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction d'une règle par index
- Ajout d'une règle définie avec un modèle
- Suppression d'une règle définie avec un modèle
- Modification de l'ordre établi (d'une place ou à un emplacement d'index spécifique)

La classe `RuleSetRule` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom de la règle
- Extraction du nom affiché de la règle

- Extraction de la présentation de l'utilisateur
- Extraction du bloc de règle

La classe `RuleSetRuleTemplate` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Création d'une instance de modèle de règle à partir de la définition de modèle correspondante
- Extraction de l'ensemble de règles parent

La classe `TemplateInstanceRule` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction des paramètres de la règle
- Extraction de la définition de modèle qui a permis de définir la règle

La classe `Template` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

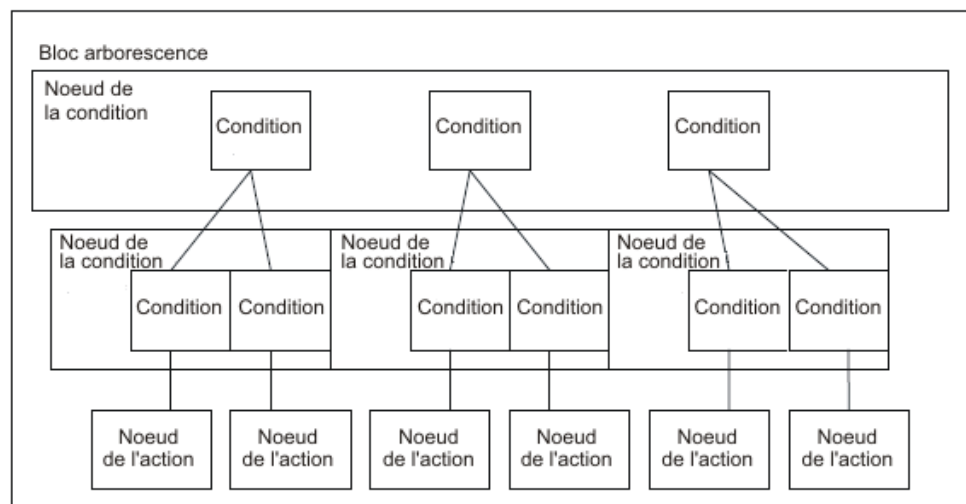
- Extraction de l'ID de modèle
- Extraction du nom
- Extraction et définition du nom affiché
- Extraction et définition de la description
- Extraction des paramètres de ce modèle
- Extraction de la présentation de l'utilisateur

Table de décision

Les tables de décision représentent un autre type de règle métier que vous pouvez gérer et modifier. Elles sont généralement utilisées lorsque de nombreuses conditions doivent être évaluées et qu'un ensemble spécifique d'actions doivent être émises une fois les conditions remplies.

Les tables de décision sont semblables aux arborescences de décision, mais elles sont équilibrées. Elles comportent toujours le même nombre de conditions à évaluer et d'actions à exécuter, quelles que soient les branches résolues sur true. Une arborescence de décision peut comporter une branche ayant plus de conditions à évaluer qu'une autre.

Les tables de décision sont structurées sous la forme d'une arborescence de noeuds et sont définies par un `TreeBlock`. Différents `TreeNode` composent le `TreeBlock`. Les `TreeNode` peuvent être des noeuds de condition ou d'action. Les noeuds de condition sont les branches d'évaluation. A la fin des branches, les noeuds d'action contiennent les actions d'arborescence appropriées à émettre et pour lesquelles toutes les conditions doivent avoir pour résultat true. Le nombre de niveaux de noeuds de condition est illimité, mais il ne peut y avoir qu'un seul niveau de noeuds d'action.



Les tables de décision peuvent également comporter une règle d'initialisation (règle init) qui peut être émise avant vérification des conditions de la table.

La classe `DecisionTable` contient des méthodes permettant de :

- Extraire le bloc d'arborescence de noeuds d'arborescences (noeuds de condition et d'action)
- Extraire l'instance de règle init
- Extraire le modèle de règle s'il est défini

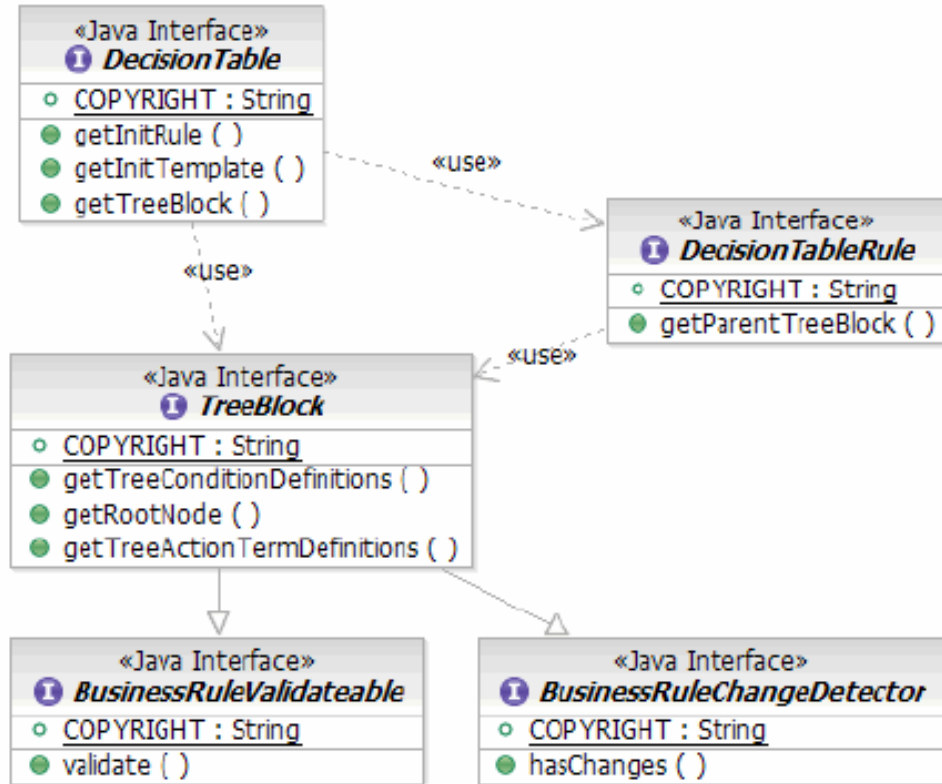
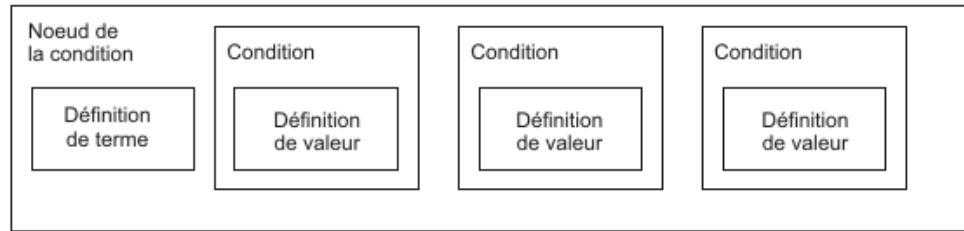


Figure 15. Diagramme de classes de DecisionTable et classes associées

Le TreeBlock d'une table de décision contient les différents noeuds de condition et d'action. Chaque noeud de condition (ConditionNode) abrite une définition de terme (TreeConditionTermDefinition) et de un à n résultats de cas (CaseEdge). La définition de terme contient l'opérande situé à gauche de l'expression de condition. Les résultats de cas contiennent les définitions de valeurs représentant les opérandes situés à droite, à utiliser dans l'expression de condition. Par exemple, dans l'expression (statut == "or") la définition de terme est "statut" et "or" est la définition de valeur dans le résultat de cas. L'ensemble des résultats de cas d'un noeud de condition partagent la définition du terme et diffèrent uniquement par leur valeur (TreeConditionValueDefinition). Pour poursuivre avec cet exemple, un autre résultat de cas dans le noeud de condition peut présenter la valeur "argent". Elle est alors également utilisée dans une expression (statut == "argent"). La seule exception à ce comportement est la définition d'une clause OTHERWISE dans le noeud de condition. Avec cette clause, il n'y a aucune définition de valeur puisqu'elle est utilisée si tous les autres résultats de cas dans le noeud de condition ont pour résultat false. Bien qu'une OTHERWISE ne soit pas un résultat de cas, elle possède un TreeNode impossible à extraire.



En ce qui concerne la définition de terme, la présentation utilisateur peut être extraite et utilisée dans les applications client. La présentation de la définition de terme est généralement une simple représentation de l'opérande situé à gauche (statut, dans notre exemple) et ne contient aucune marque de réservation. En ce qui concerne les résultats de cas, un modèle peut être utilisé pour définir la définition de valeur (`TreeConditionValueTemplate`). Une instance de définition de valeur de modèle (`TemplateInstanceExpression`) contient les valeurs de paramètres utilisées pour l'exécution, qui sont modifiables. Si une tentative de récupération de la définition de modèle de valeur est réalisée pour `TreeConditionValueDefinition`, non défini avec un modèle, une valeur NULL est renvoyée. Si aucun modèle n'a été utilisé pour définir la condition de valeur, une présentation utilisateur peut toujours être extraite et utilisée dans les applications client si cela a été spécifié lors de la création.

La classe `TreeBlock` contient des méthodes permettant de :

- Extraire le noeud racine de l'arborescence
- Extraire les définitions de terme de condition pour le bloc d'arborescence
- Extraire les définitions de terme d'action du bloc d'arborescence

Le noeud racine de l'arborescence est du type `TreeNode` et il représente le point de départ de la navigation dans la table de décision. La classe `TreeNode` contient des méthodes permettant de :

- Déterminer si un noeud est une clause OTHERWISE
- Extraire le noeud parent du noeud d'arborescence en cours (noeud de condition ou d'action)
- Extraire le noeud racine de l'arborescence contenant le noeud d'arborescence en cours

La classe `ConditionNode` contient des méthodes permettant de :

- Extraire les résultats de cas
- Extraire la définition de terme
- Extraire le cas OTHERWISE
- Extraire les modèles des conditions de valeur des résultats de cas pour le noeud de condition
- Ajouter au noeud une valeur de condition basée sur un modèle
- Supprimer une valeur de condition basée sur un modèle

La classe `CaseEdge` contient des méthodes permettant de :

- Extraire la liste des modèles de valeur disponibles pour la définition de valeur
- Extraire le noeud enfant (noeud de condition ou d'action)
- Extraire l'instance de la définition de modèle associée à la définition de valeur
- Extraire directement la définition de valeur sans extraire le modèle

- Définir la valeur de la définition pour utiliser une définition d'instance de modèle spécifique

La classe `TreeConditionTermDefinition` contient des méthodes permettant de :

- Extraire les modèles de définition de valeur définis pour le noeud de condition
- Extraire la présentation utilisateur du terme de condition

La classe `TreeConditionDefinition` contient des méthodes permettant de :

- Extraire la définition de terme du noeud de condition
- Extraire les définition de valeur de condition pour le noeud de condition, à partir de tous les résultats de cas
- Extraire l'orientation (ligne ou colonne)

La classe `TreeConditionValueDefinition` contient des méthodes permettant de :

- Extraire l'expression d'instance de modèle spécifique définie pour la valeur
- Extraire l'utilisateur

La classe `Template` contient des méthodes permettant de :

- Extraire l'ID système du modèle
- Extraire le nom du modèle
- Extraire les paramètres définis pour le modèle
- Extraire la présentation du modèle

La classe `TreeConditionValueTemplate` contient une méthode permettant de :

- Créer une nouvelle instance de valeur de condition de modèle

La classe `TemplateInstanceExpression` contient des méthodes permettant de :

- Extraire les paramètres de l'instance de modèle
- Extraire le modèle (`TreeConditionValueTemplate` dans le cas d'un résultat de cas dans une table de décision) utilisé pour définir l'instance

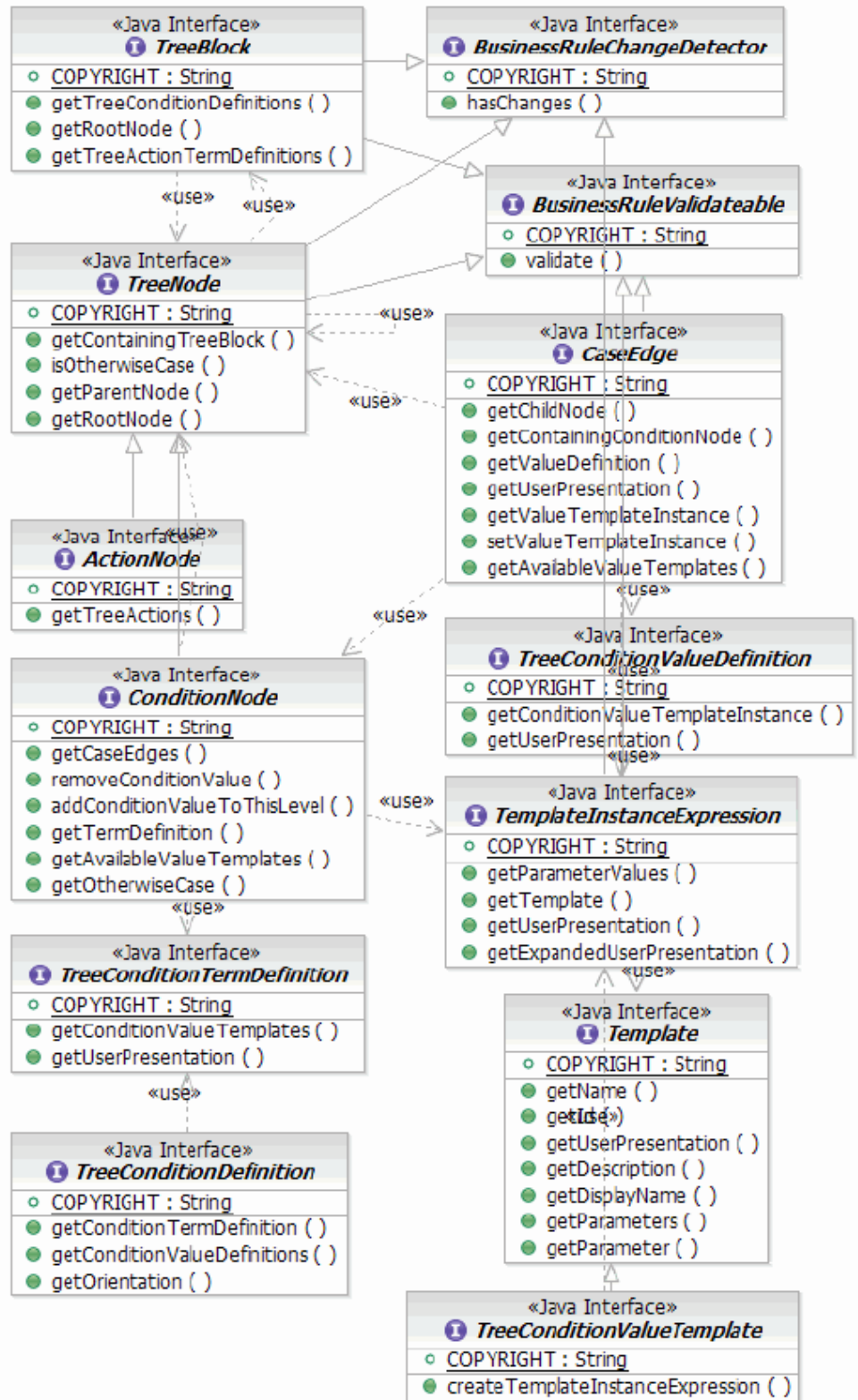


Figure 16. Diagramme de classes de *TreeNode* et classes associées

Lorsqu'un nouveau résultat de cas est ajouté à un noeud de condition, il doit utiliser un modèle pour la définition de la valeur. Par exemple, si un nouveau cas "bronze" doit être ajouté pour la vérification de 'statut', le modèle approprié

(TreeConditionValueTemplate) devra être utilisé pour créer un nouveau TemplateInstanceExpression, en définissant la valeur de paramètre sur "bronze".

Lors de l'ajout d'un nouveau résultat de cas, un noeud de condition enfant lui est également ajouté automatiquement. Ce noeud de condition enfant contient des résultats de cas basés sur les définitions de résultats de cas définies pour les noeuds de condition situés au même niveau. Si des modèles ou des valeurs codées en dur sont utilisés dans les résultats de cas, ils sont ensuite également utilisés dans les résultats de cas du noeud de condition enfant. Ce noeud, ajouté automatiquement, possède également ses propres noeuds de condition enfant, créés automatiquement. Ces noeuds de condition enfant possèdent à leur tour des noeuds de condition enfant et ainsi de suite, jusqu'à ce que tous les niveaux de noeuds de condition aient été recréés.

Outre les noeuds de condition, une table de décision et plus spécifiquement un bloc d'arborescence contient également un niveau de noeuds d'action (ActionNode). Les noeuds d'action sont des noeuds terminaux qui résident à la fin de la branche de noeuds de condition et des résultats de cas. Si toutes les valeurs de condition d'une ligne de résultats de cas ont pour résultat true, un noeud d'action est atteint. Le noeud d'action possède au moins une action (TreeAction) définie. Cette action a une définition de terme et une définition de valeur. A l'instar des noeuds de condition, la définition de terme (TreeActionTermDefinition) se situe à gauche de l'expression et la définition de valeur (TemplateInstanceExpression) à droite. Par exemple, pour les différents noeuds de condition procédant à une vérification du statut, des actions peuvent définir la remise. Si la condition est (statut == "or"), l'action peut être (valeurRemise = 0.90). Pour l'action, "valeurRemise" est la définition de terme, et "= 0.90" est la définition de valeur.

La définition de terme d'une action d'arborescence est partagée avec d'autres actions d'arborescence dans d'autres noeuds d'action. Dans la mesure où chaque branche de résultats de cas accède à une action, les mêmes définitions de terme sont utilisées. Toutefois, ces dernières peuvent différer par l'action d'arborescence et le noeud d'action. Par exemple, la valeurRemise avec le statut "or" peut être "0.90", et "0.95" pour un statut "argent".

Les noeuds d'action peuvent comporter plusieurs actions d'arborescence avec une définition de terme distincte et une définition de valeur distincte. Par exemple, si la remise est fixée pour un véhicule de location, outre la définition de valeurRemise, vous pouvez également affecter un niveau de véhicule spécifique. Une autre action d'arborescence peut être créée pour définir le terme "qualitéVéhicule" sur "haut de gamme" si le statut est "or", et "valeurRemise" sur "0.90".

La définition de valeur dans une action d'arborescence peut être créée à partir d'un modèle (TreeActionValueTemplate). La définition de modèle contient une expression (TemplateInstanceExpression) contenant des paramètres.

En plus des paramètres, la définition de valeur entière peut être modifiée par une nouvelle instance de définition de valeur, créée avec un autre modèle défini pour l'action d'arborescence.

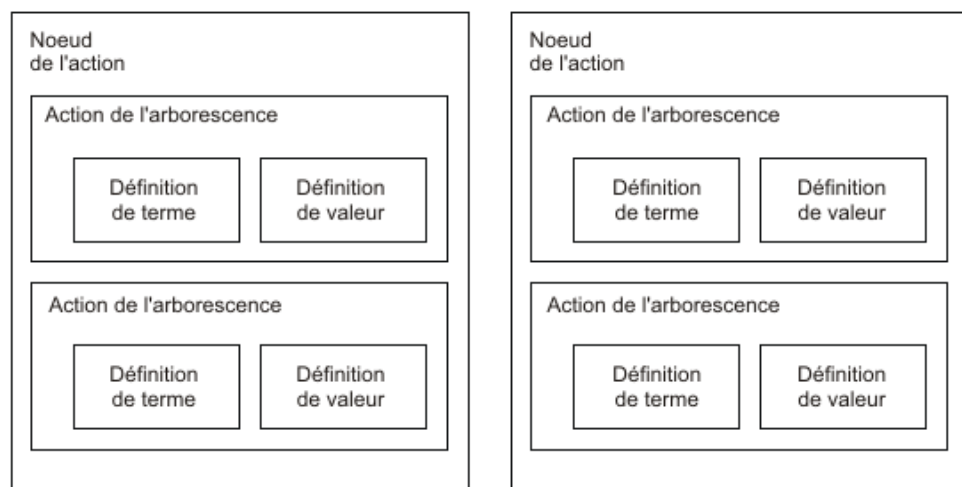
Si une définition de valeur n'est pas créée à partir d'un modèle, elle ne peut pas être modifiée. En ce qui concerne les applications client, la présentation utilisateur peut être utilisée dans l'affichage si cela a été précisé au moment de la création.

Pour les définitions de terme des actions d'arborescence, si une présentation utilisateur a été spécifiée, elle peut également être utilisée par les applications client.

Lorsqu'un nouveau résultat de cas est ajouté à un noeud de condition et que différents noeuds de condition enfant sont créés, des noeuds d'action sont également créés. Contrairement aux noeuds de condition enfant et aux résultats de cas créés en fonction de la définition des résultats de cas déjà définis pour ce niveau, les noeuds d'action n'héritent pas automatiquement d'une conception existante. Seuls les marques de réservation `TreeActions` vides sont créées dans le noeud d'action. Un modèle (`TreeActionValueTemplate`) doit être utilisé pour compléter la définition d'action en créant une `TemplateInstanceExpression` pour au moins une définition de terme du noeud d'action. Avant que l'action d'arborescence soit définie avec `TemplateInstanceExpression`, elle possède des valeurs `NULL` spécifiées pour la valeur de présentation utilisateur et la valeur d'instance de modèle.

Lors de la création d'une nouvelle condition ayant pour résultat de nouveaux `ActionNodes`, les noeuds d'action sont ajoutés à droite des actions existantes pour le noeud de condition parent immédiat. Par exemple, si un statut "rubis" est ajouté à la table de décision et est censé disposer d'une remise spécifique, la condition de vérification du statut est ajoutée à droite de "or", "argent" et "bronze". Le noeud d'action de la remise associée à "rubis" est ajouté à droite des noeuds d'action correspondant aux résultats de cas "or", "argent" et "bronze".

Lors de la définition de nouvelles actions d'arborescence pour des noeuds d'action, un algorithme basé sur le noeud d'action de droite du dernier résultat de cas renvoie le noeud d'action avec une action d'arborescence vide. Vous pouvez également vérifier si l'action d'arborescence possède des valeurs `NULL` pour la valeur de présentation utilisateur et la valeur d'instance de modèle. Une fois l'action d'arborescence obtenue, elle peut être définie avec l'instance adéquate de `TreeActionValueTemplate`.



La classe `ActionNode` contient une méthode permettant de :

- Extraire la liste des actions d'arborescences définies

La classe `TreeAction` contient des méthodes permettant de :

- Extraire la liste des modèles de valeurs disponibles, définies pour l'action d'arborescence
- Extraire la définition de terme
- Extraire l'instance de modèle de valeur définie pour l'action d'arborescence
- Extraire la présentation utilisateur de la valeur si un modèle de valeur n'a pas été utilisé
- Vérifier si l'action est un appel de service SCA (méthode `isValueNotApplicable`)
- Remplacer l'instance de modèle de valeur par une nouvelle instance

La classe `TreeActionTermDefinition` contient des méthodes permettant de :

- Extraire la présentation utilisateur pour la définition de valeur de terme
- Extraire la liste des modèles de valeurs disponibles pour l'action d'arborescence
- Vérifier si l'action est un appel de service SCA (méthode `isTermNotApplicable`)

La classe `Template` contient des méthodes permettant de :

- Extraire l'ID système du modèle
- Extraire le nom du modèle
- Extraire les paramètres définis pour le modèle
- Extraire la présentation du modèle

La classe `TreeActionValueTemplate` contient une méthode permettant de :

- Créer une nouvelle instance de modèle de valeur à partir de la définition de modèle

La classe `TemplateInstanceExpression` contient des méthodes permettant de :

- Extraire les paramètres de l'instance de modèle
- Extraire le modèle (`TreeActionValueTemplate` dans le cas d'une action d'arborescence dans une table de décision) utilisé pour définir l'instance

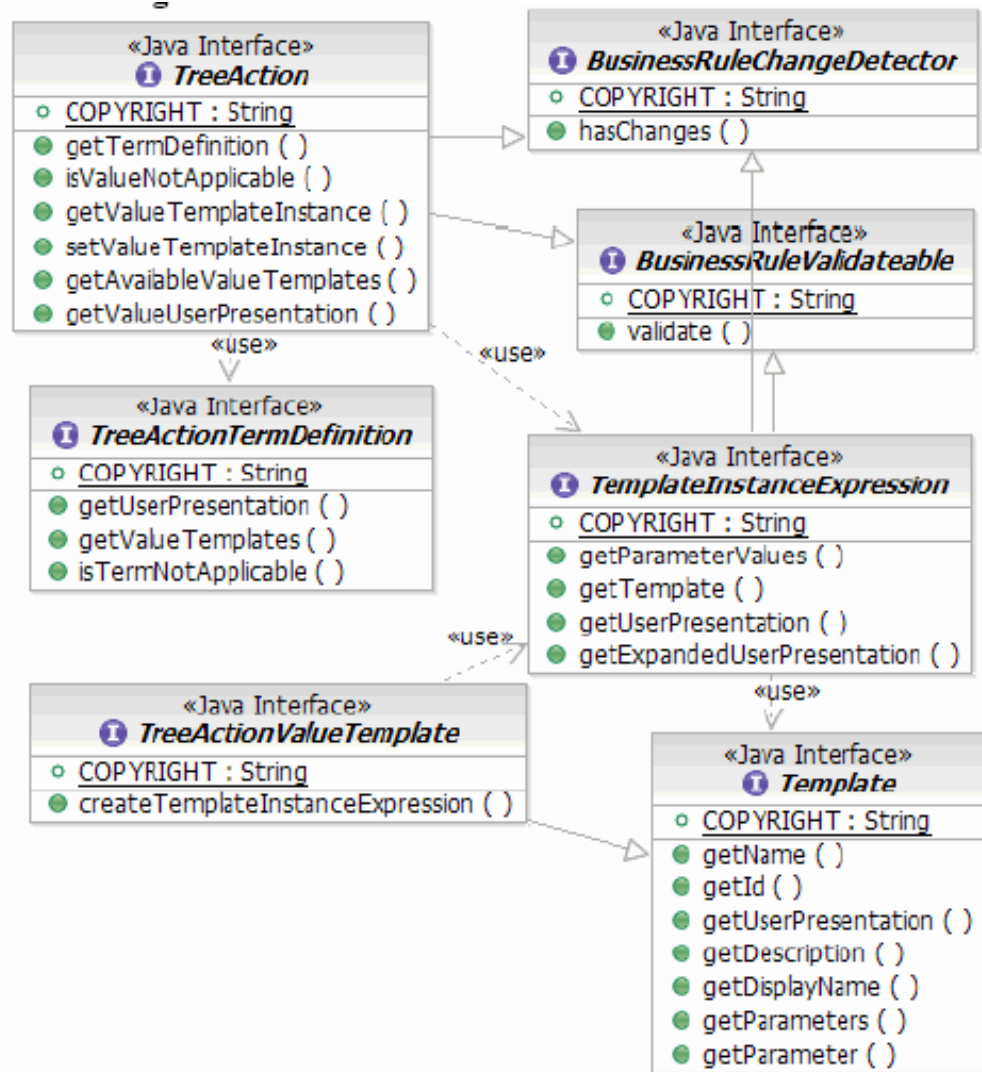


Figure 17. Diagramme de classes de TreeAction et classes associées

La définition d'une règle init pour une table de décision suit la même structure que celle d'un ensemble de règles. La règle init peut être définie avec un modèle (DecisionTableRuleTemplate).

Si une règle init n'a pas été créée au moment de la création, elle ne peut pas être ajoutée une fois la règle déployée.

La classe Rule contient des méthodes permettant de :

- Extraire le nom de la règle
- Extraire la présentation utilisateur pour la règle
- Extraire la présentation utilisateur pour la règle avec les différents paramètres définis de la règle

La classe DecisionTableRule contient une méthode permettant de :

- Extraire le bloc d'arborescence contenant la règle init

La classe DecisionTableRuleTemplate contient une méthode permettant de :

- Extraire la table de décision contenant le modèle

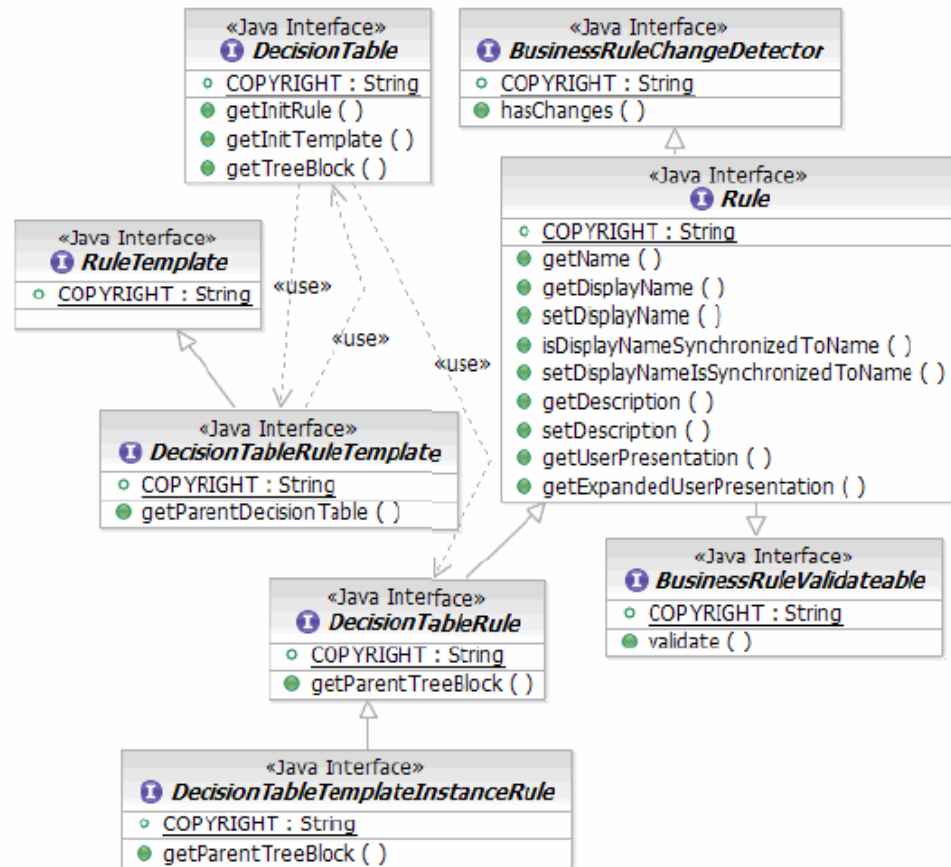


Figure 18. Diagramme de classes de DecisionTableRule et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l’aide de deux outils de création différents, et sont émises par le dispositif d’exécution de règles. Tous trois partagent le même modèle d’artefacts de règles métier.

Modèles et paramètres

Les modèles inclus dans les ensembles de règles et dans les tables de décision prennent comme base une définition commune. Les modèles possèdent des paramètres et une présentation de l’utilisateur. Les valeurs de paramètres inclus dans les modèles sont définis pour permettre d’apporter des modifications à la règle une fois que celle-ci a été déployée.

La valeur de présentation utilisateur fournit une valeur de chaîne qui peut être utilisée pour l’affichage de la règle et de différents paramètres de façon conviviale. Cette présentation sous forme de chaîne possède des marques de réservation qui permettent le remplacement des différentes valeurs de paramètres, ainsi que leur affichage. Ces marques de réservation figurent au format (<paramètre index>). Par exemple, si la chaîne de présentation de la règle init est “Base discount is {0} %” (la remise de base s’élève à x), la marque de réservation {0} doit être remplacée par la

valeur de paramètre correspondante. La chaîne de présentation ne peut pas être modifiée pour la règle ou la définition de modèle. Toutefois, les valeurs de marque de réservation peuvent être modifiées avec les valeurs de paramètre figurant dans une application client, selon la définition figurant dans le modèle. Les différents modèles incluent une méthode de simplification (`getExpandedUserPresentation`) qui renvoie une chaîne contenant toutes les valeurs de paramètre, correctement placées dans la chaîne.

Toutes les valeurs de paramètres possèdent un type de données spécifique ; toutefois, lors de l'extraction et de la définition d'une valeur de paramètre, un objet string est utilisé. La valeur de paramètre peut être considérée comme une chaîne lors du remplacement de la valeur dans la présentation utilisateur, ou encore lors de l'affectation d'une nouvelle valeur au paramètre. Le paramètre est converti dans le type de données approprié au moment de l'exécution, afin d'émettre la règle correctement. Au cours de la validation, la valeur de paramètre est comparée au type de données afin de vérifier qu'il est correct. Par exemple, si un paramètre est de type boolean et porte la valeur "T", la validation ne reconnaît pas cette valeur et renvoie un message d'erreur.

Dans la définition de modèle, les valeurs de paramètres peuvent être limitées par des contraintes. Ces contraintes peuvent être définies sous forme de plage ou d'énumération. Les contraintes du paramètre seront mises en oeuvre une fois la règle validée. Si aucun modèle n'a été utilisé pour définir la valeur, seule une présentation utilisateur sera disponible. Une définition de valeur ne peut pas utiliser à la fois un modèle et une présentation utilisateur. En cas d'utilisation d'un modèle, la présentation issue de la définition de modèle est la seule présentation disponible.

La classe `Template` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction de l'ID de modèle
- Extraction du nom
- Extraction des paramètres
- Extraction de la présentation de l'utilisateur

La classe `Parameter` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom du paramètre
- Extraction du type de données du paramètre
- Extraction de la contrainte associée au paramètre
- Extraction du modèle définissant le paramètre
- Création d'une valeur de paramètre

La classe `ParameterValue` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom du paramètre
- Extraction de la valeur du paramètre
- Définition de la valeur du paramètre

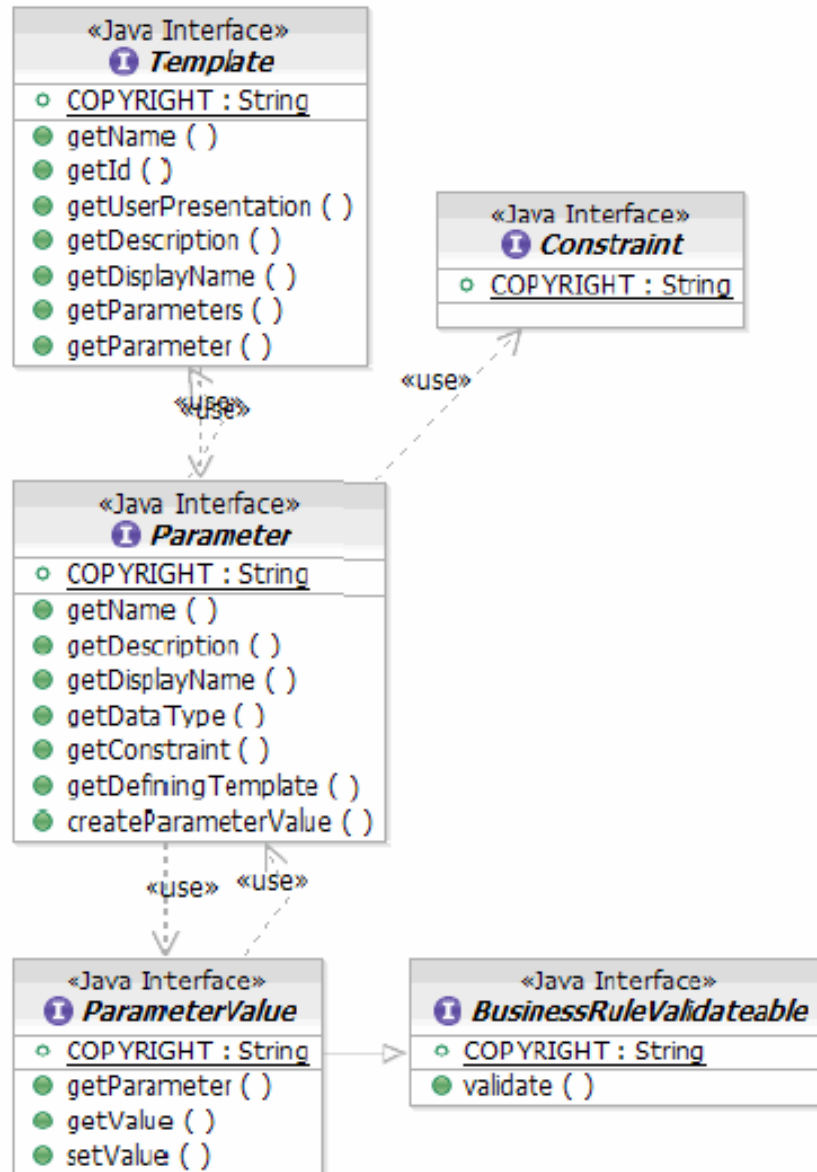


Figure 19. Diagramme de classes de Template et de Parameter, et classes associées

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Validation

Parmi les objets principaux, nombreux sont ceux qui possèdent une méthode de validation ; elle permet de vérifier si les artefacts sont corrects et complets avant leur publication.

La validation qui a lieu au moment de l'apport de modifications via les classes d'API ne représente qu'un sous-ensemble de la validation globale effectuée lors de l'exécution de la commande serviceDeploy ou au moment de l'édition des artefacts

dans WebSphere Integration Developer. Cela est dû aux contraintes déjà associées au groupe de règles métier (limitation des aspects modifiables au moment de l'exécution). L'utilisateur des classes peut valider la table de sélection des groupes de règles métier, la table des ensembles de règles ou la table de décision chaque fois que nécessaire (le composant de groupes de règles lui-même n'est pas modifiable au moment de l'exécution). Lorsqu'un groupe de règles métier est publié, la table de sélection de groupes de règles, la table d'ensembles de règles et la table de décision sont validées avant leur publication dans le référentiel.

Si les artefacts sont incorrects, une exception `ValidationException` est générée, accompagnée de la liste des problèmes de validation rencontrés. Les différents problèmes de validation rencontrés sont documentés dans la section consacrée au traitement des exceptions.

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Suivi des modifications

Pour tous les objets, vous pouvez utiliser une méthode `hasChanges` afin de vérifier si des modifications ont été apportées à l'objet et aux objets qu'il contient.

Cette méthode peut être utilisée pour vérifier les modifications et pour publier un groupe de règles métier (uniquement si certains de ses éléments ont été modifiés).

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

BusinessRuleManager

La classe `BusinessRuleManager` est la principale classe d'utilisation des groupes de règles, des ensembles de règles et des tables de décision.

La classe `BusinessRuleManager` comporte des méthodes permettant d'extraire les groupes de règles métier par nom, par espace de nom cible ou par propriétés personnalisées. Elle contient également une méthode de publication des modifications apportées aux groupes de règles métier, aux ensembles de règles ou aux tables de décision.

La classe `BusinessRuleManager` contient des méthodes permettant de :

- Extraire tous les groupes de règles métier
- Extraire les groupes de règles métier d'un espace de nom cible spécifique
- Extraire les groupes de règles métier d'un nom spécifique
- Extraire les groupes de règles métier d'un espace de nom cible et d'un nom spécifiques
- Extraire les groupes de règles métier contenant une propriété spécifique
- Extraire les groupes de règles métier contenant des propriétés spécifiques
- Publier des groupes de règles métier

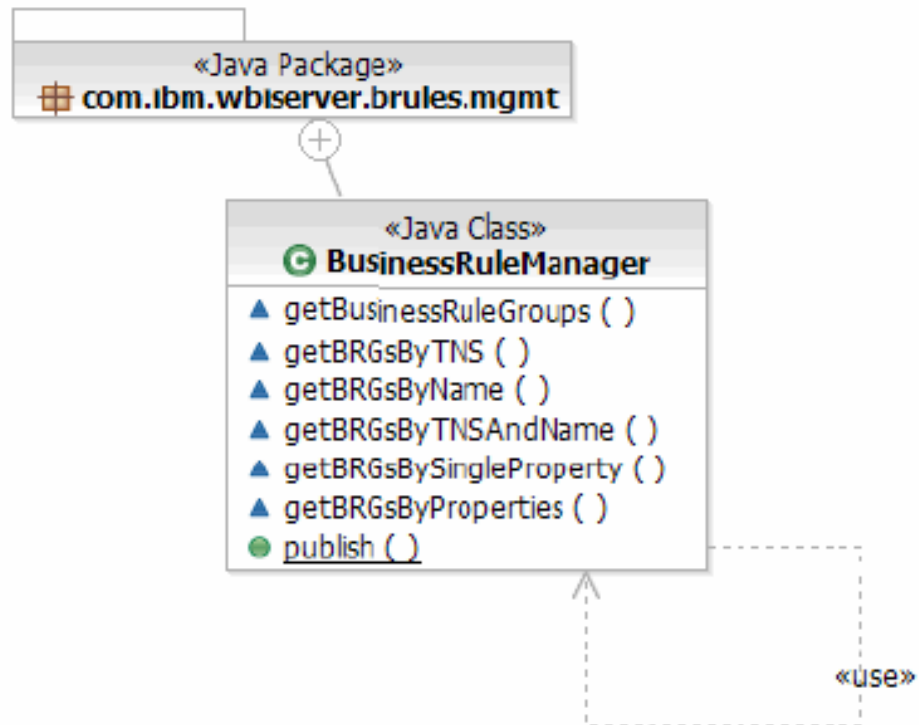


Figure 20. Diagramme de classes de BusinessRuleManager et module

Requête du composant de groupe de règles

Le composant de groupe de règles peut posséder des propriétés définies par l'utilisateur (paires nom/valeur) permettant d'affiner la liste des groupes de règles métier renvoyés par la classe. Les zones utilisables dans la requête et dans toute combinaison sont les suivantes :

- Espace de nom cible du composant de groupe de règles métier
- Nom du composant de groupe de règles métier
- Nom de propriété
- Valeur de propriété

Chaque nom de propriété ne peut être défini qu'une seule fois pour chaque composant de groupe de règles métier.

La fonction de requête prise en charge par cette classe représente un petit sous-ensemble du langage SQL entier. L'utilisateur ne fournit pas l'instruction SQL mais les valeurs des paramètres d'une propriété unique ou d'une structure arborescente contenant les informations liées à une requête à propriétés multiples sous forme de noeuds. Des noeuds d'opérateur logique et des noeuds de requête de propriété implémentent l'interface QueryNode. Les noeuds d'opérateur logique spécifient les opérateurs booléens (AND, OR, NOT). Ces derniers sont créés via QueryNodeFactory. Dans le cadre de la création de ces noeuds d'opérateur logique, les côtés gauche et droit de l'opérateur doivent être spécifiés avec des classes QueryNode supplémentaires. Ces noeuds peuvent être soit un noeud de requête de propriété, soit un autre noeud d'opérateur logique. Si un noeud de requête de propriété est transmis, il contient le nom, la valeur et l'opérateur (EQUAL (==),

NOT_EQUAL (!=), LIKE ou NOTLIKE) de propriété. L'ensemble de l'interface QueryNode est analysée par la classe et une requête est effectuée sur les données sous-jacentes de la mémoire persistante.

Les recherches génériques sont prises en charge lors de l'utilisation des opérateurs LIKE et NOTLIKE. Les caractères '%' et '_' sont pris en charge dans les recherches génériques. Le caractère '%' est utilisé lorsqu'un nombre infini de caractères sont inconnus ou ne doivent pas être pris en compte dans la recherche. Par exemple, si une recherche doit être lancée pour tous les groupes de règles métier possédant une propriété avec un nom de Service et une valeur commençant par "Nord", la valeur doit être indiquée comme suit : "Nord%". Autre exemple, supposons que tous les services dont la valeur se termine par "Région" sont souhaités. La valeur sera alors "%Région". Le caractère '%' peut également être utilisé au milieu d'une chaîne. Par exemple, dans le cas de groupes de règles métier dont les propriétés ont les valeurs "RégionCentreNord", "RégionEstNord" et "RégionOuestNord", vous pouvez spécifier la valeur "Région%Nord".

Le caractère '_' est utilisé lorsqu'un seul caractère est inconnu ou qu'il ne doit pas être pris en compte dans la recherche. Par exemple, si une recherche porte sur tous les groupes de règles métier pour lesquels les propriétés Service ont les valeurs "Srv1Nord", "Srv2Nord", "Srv3Nord" et "Srv4Nord", la valeur "Srv_Nord" peut être spécifiée, et les 4 groupes de règles métier comportant ces propriétés sont renvoyés. Le caractère '_' peut être utilisé plusieurs fois dans une valeur de recherche, et chaque instance indique un caractère unique à ignorer. Le caractère '_' peut être utilisé au début ou à la fin d'une valeur. Par exemple, si deux caractères doivent être ignorés dans une valeur, vous pouvez utiliser deux '_' comme dans "Svc__ud".

Pour pouvoir traiter '%' et '_' en tant que caractères littéraux plutôt qu'en tant que caractères génériques, spécifiez le caractère d'échappement '\' avant '%' ou '_'. Par exemple, si le nom de propriété est "%Remise", pour pouvoir l'utiliser dans une requête, spécifiez "\\%Remise". Si le caractère '\' doit être utilisé en tant que caractère littéral, un autre caractère d'échappement '\' doit être utilisé, comme dans "Commandes\\Client". Si un seul caractère '\' est placé, sans être suivi de '%', '_' ou '\', une exception IllegalArgumentException est émise.

Les caractères génériques peuvent être utilisés uniquement du côté gauche (valeur de propriété). Ils ne peuvent pas être utilisés dans un nom de propriété.

Au cours de recherches portant sur la valeur d'une propriété spécifique ou lors d'une recherche de valeurs, si aucune propriété n'est renvoyée, l'artefact est ignoré de la recherche. Par exemple, si parmi 3 groupes de règles métier (A, B et C), seulement deux (A et B) ont une propriété intitulée "Service" avec des valeurs différentes (respectivement "Comptabilité" et "Expédition"), et qu'une recherche est lancée sur tous les groupes de règles métier ne comportant pas de propriété "Service" définie sur "Comptabilité", seul le groupe de règles métier dont la propriété "Service" est définie, mais n'équivaut pas à "Comptabilité" (groupe de règles métier B), est renvoyé. Le groupe de règles métier (C), qui ne comporte pas de propriété "Service", n'est pas renvoyé puisque cette propriété n'est pas définie.

En cas d'utilisation de propriétés de recherche, deux propriétés spéciales intitulées *IBMSysName* et *IBMSysTargetNameSpace* servent aux recherches basées sur le nom et l'espace de nom d'un artefact. Ces valeurs peuvent également être extraites à l'aide des méthodes *getName* et *getTargetNameSpace*.

La classe prend en charge les méthodes de requête suivantes :

```

List getBRGsByTNS (string tNSName, Operator op, int skip, int threshold)
List getBRGByName(string Name, Operator op, int skip, int threshold)
List getBRGsByTNSAndName (string tNSName, Operator, tNSOp, string
    name, Operator nameOp, int skip, int threshold)
List getBRGsBySingleProperty (string propertyName, string propertyValue,
    Operator op, int skip, int threshold)
List getBRGsByProperties (QueryNode queryTree, int skip, int threshold)

```

Les paramètres 'skip' et 'threshold' permettent à l'utilisateur d'extraire une liste partielle de résultats jusqu'au seuil spécifié. La valeur zéro pour ces deux paramètres renvoie la liste entière de résultats. Le curseur n'est pas maintenu dans l'ensemble de résultats à partir d'un appel de requête. Si une valeur de saut est utilisée, il est possible que des ajouts ou des suppressions aient été apportés à l'ensemble de résultats et qu'une demande ultérieure renvoie alors des groupes de règles métier situés dans un précédent ensemble de résultats.

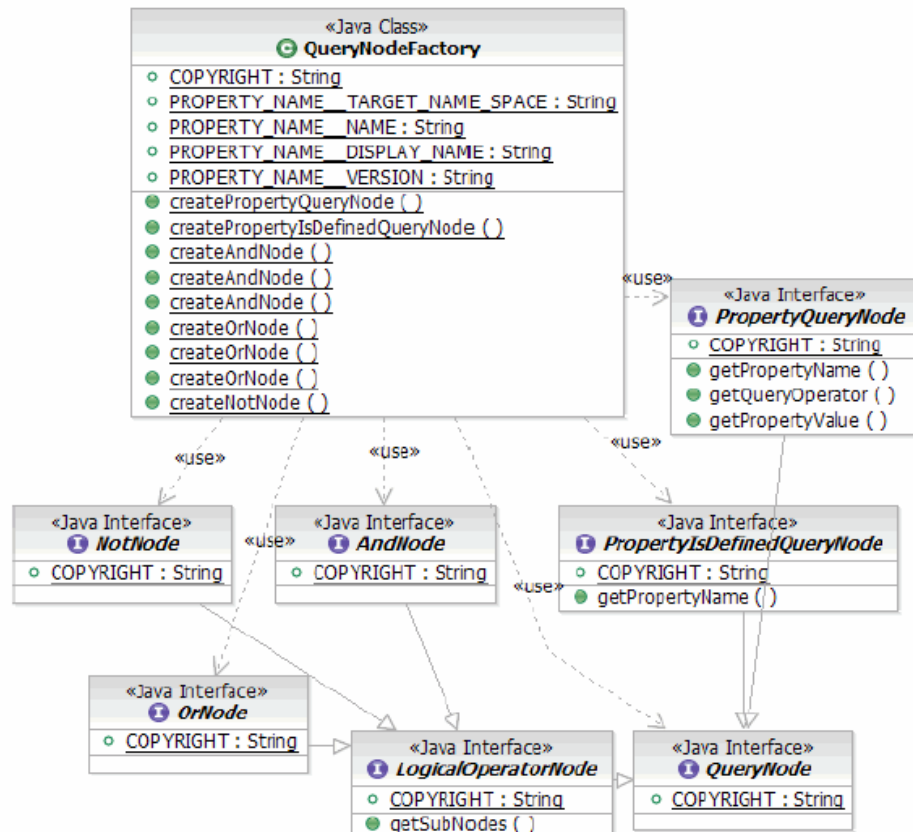


Figure 21. Diagramme de classes de QueryNodeFactory et classes associées

Les noeuds de l'arborescence permettent à l'utilisateur de spécifier une expression de recherche à l'aide des opérateurs booléens, des caractères génériques (% et échappement) et de la paire propriété/valeur. L'opérateur est valide uniquement pour les valeurs, l'opérateur de la propriété est toujours représenté par les signes égal (==).

Publication

La publication des modifications de règles métier est réalisée au niveau du composant de groupe de règles métier. L'utilisateur peut publier 1...n composants de groupe de règles métier. Avant l'exécution d'une opération de publication, une action de validation est réalisée sur le groupe de règles métier et sur les différents objets présents dans ce groupe (table de sélection d'opération, ensembles de règles, tables de décision, etc). Chaque demande de publication survient dans une transaction unique. En cas d'exception au cours de la validation ou de la publication de la base de données, la transaction est annulée et aucune modification de groupe de règles métier n'est publiée dans le référentiel. Cela permet aux modifications dépendantes les unes des autres dans un composant unique (par exemple, la table de sélection d'opération et un ensemble de règles) ou aux dépendances entre des composants de survenir au sein d'une opération atomique.

Au moment de la publication, une vérification est réalisée pour veiller à ce que les éléments à publier n'aient pas été modifiés par une autre transaction. Pour réduire les risques de conflit, la méthode de publication offre à l'utilisateur la possibilité de publier tous les artefacts, qu'ils soient modifiés ou non, ou uniquement les artefacts modifiés dans le groupe de règles métier. Le comportement par défaut instaure la publication de tous les artefacts. Si l'option est définie sur la publication de tous les artefacts et qu'une autre transaction a modifié les artefacts entre-temps, une exception `ChangeConflictException` est émise. Pour réduire le risque de conflit, spécifiez la publication des artefacts modifiés uniquement. En procédant ainsi, il est possible que deux utilisateurs apportent des modifications au référentiel pour deux artefacts différents dans un groupe de règles métier (par exemple, deux ensembles de règles), ce qui peut insérer des modifications incompatibles dans le groupe de règles métier. En raison de l'éventualité de cette situation, cette option doit être utilisée avec précaution.

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Traitement des exceptions

Des exceptions peuvent être générées lors d'un appel de validation pour un artefact ou lors de sa publication. En cas d'erreur de validation, l'exception `ValidationException` est générée ; elle s'accompagne de la liste des problèmes rencontrés. Si un problème survient au cours de la publication car une autre transaction publie les mêmes artefacts, l'exception `ChangeConflictException` est générée. A chaque détection de la modification d'un artefact par une autre transaction, l'exception `ChangeConflictException` est générée.

Par ailleurs, une exception `SystemPropertyNotChangeableException` est générée en cas de tentative de modification d'une propriété qui duplique un nom de propriété système. En effet, les propriétés système ne peuvent pas être modifiées.

Une exception `ChangesNotAllowedException` est générée en cas de tentative d'exécution de l'opération `set` sur un artefact pendant sa publication.

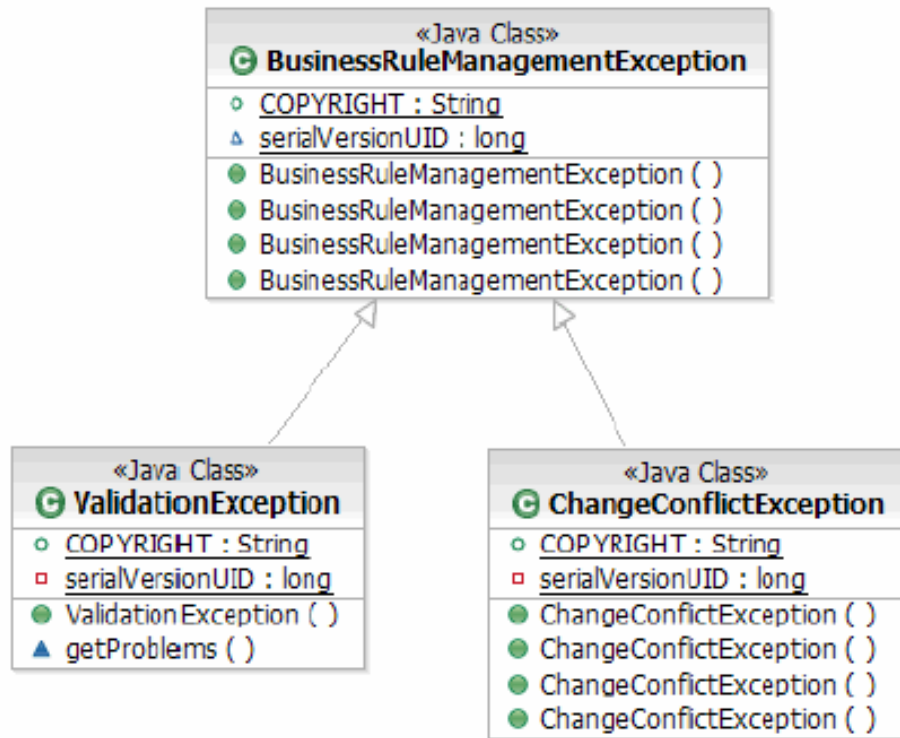


Figure 22. Diagramme de classes de BusinessRuleManagementException et classes associées

Problèmes liés aux groupes de règles métier

Ces problèmes peuvent se poser lorsqu'un groupe de règles métier est validé ou qu'une tentative de publication de ce groupe de règles métier a lieu alors qu'une partie de ce groupe est incorrecte.

Tableau 4. Problèmes liés aux groupes de règles métier

Exception	Description
ProblemBusRuleNotInAvailTargetList	Ce problème se pose lorsqu'une règle est spécifiée en tant que règle métier par défaut pour une table de sélection d'opération, mais que l'artefact correspondant à cette règle ne figure pas dans la liste des cibles disponibles pour cette opération. Pour éviter ce problème, spécifiez une règle métier valide parmi la liste des cibles disponibles pour cette opération.
ProblemDuplicatePropertyName	Ce problème se pose en cas de tentative de création d'une propriété qui représente le double d'une propriété définie par l'utilisateur pour un groupe de règles métier spécifique. Pour éviter ce problème, vous devez utiliser un nom unique de propriété.
ProblemOperationContainsNoTargets	Ce problème se pose lorsqu'une opération n'est pas associée à une destination de règle par défaut ou à un ensemble de destinations de règle planifié. Pour éviter ce problème, vous devez définir l'opération en spécifiant au minimum une destination de règle en tant que valeur par défaut ou en tant que période planifiée.

Tableau 4. Problèmes liés aux groupes de règles métier (suite)

Exception	Description
ProblemOverlappingRanges	Ce problème se pose lorsque la date de début ou la date de fin d'un enregistrement de sélection d'opération chevauche la plage correspondante d'un autre enregistrement de sélection d'opération. Ce chevauchement de plages de dates empêche la localisation de la destination de règle à appeler. Pour éviter ce problème, vous devez vérifier la date de début ou la date de fin des autres enregistrements de sélection d'opération afin de vous assurer qu'il n'existe pas de chevauchement.
ProblemStartDateAfterEndDate	Ce problème se pose lorsque la date de début d'un enregistrement de sélection d'opération est ultérieure à la date de fin de cet enregistrement. Ce problème peut se poser pour tous les enregistrements de sélection d'opération, à l'exception de l'enregistrement par défaut, qui ne possède ni date de début, ni date de fin. Pour éviter ce problème, vous devez spécifier une date de début après avoir spécifié la date de fin d'un enregistrement de sélection d'opération.
ProblemTargetBusRuleNotSet	Ce problème se pose lorsque la règle spécifiée dans un enregistrement de sélection d'opération ne figure pas dans la liste des règles cibles disponibles. Pour éviter ce problème, vous devez spécifier une règle figurant dans la liste des cibles disponibles.
ProblemTNSAndNameAlreadyInUse	Ce problème se pose lorsqu'une nouvelle règle métier est créée et qu'elle porte un nom et un espace de nom cible déjà utilisé par un ensemble de règles ou par une table de décision. Dans ce cas, un contrôle est effectué au niveau de tous les ensembles de règles et de toutes les tables de décisions associés au groupe de règles métier en cours d'utilisation, et au niveau de tous les artefacts de règles stockés dans le référentiel. Pour éviter ce problème, vous devez utiliser un autre nom ou espace de nom cible.
ProblemWrongOperationForOpSelectionRecord	Ce problème se pose lorsqu'un nouvel enregistrement de sélection d'opération est ajouté à une liste d'enregistrements de sélection d'opération et que le fonctionnement du nouvel enregistrement ne correspond pas à celui des autres enregistrements de la liste. Pour éviter ce problème, vous devez créer une nouvelle opération à l'aide de la méthode <code>newOperationSelectionRecord</code> au niveau de l'objet approprié de la liste des enregistrements de sélection d'opération.

Problèmes liés aux ensembles de règles et aux tables de décisions

Tableau 5. Problèmes liés aux ensembles de règles et aux tables de décisions

Exception	Description
ProblemInvalidBooleanValue	Ce problème se pose lorsqu'un paramètre de modèle de règle figurant dans un ensemble de règles ou qu'une valeur d'action ou de condition figurant dans une table de décision reçoit une valeur autre que "true" ou "false" alors qu'il s'agit d'un paramètre de type booléen. Par exemple, il peut s'agir d'une valeur "T" ou "F". Pour éviter ce problème, vous devez utiliser les valeurs "true" ou "false" lorsque vous recourez à un paramètre de type booléen.
ProblemParmNotDefinedInTemplate	Ce problème se pose lorsqu'une valeur est spécifiée pour un paramètre de modèle et que ce paramètre n'est pas défini dans la liste des paramètres valides pour ce modèle. Les paramètres doivent être vérifiés avant la configuration du modèle. Cela peut se produire pour les modèles <code>RuleTemplate</code> , <code>TreeActionValueTemplate</code> , ou encore <code>TreeConditionValueTemplate</code> .

Tableau 5. Problèmes liés aux ensembles de règles et aux tables de décisions (suite)

Exception	Description
ProblemParmValueListContainsUnexpectedValue	Ce problème se pose lorsque des paramètres valides sont transmis avec un modèle, mais que le nombre de paramètres soit trop élevé. Dans ce cas, le nombre de paramètres doit être diminué. Cela peut se produire pour les modèles RuleTemplate, TreeActionValueTemplate, ou encore TreeConditionValueTemplate.
ProblemRuleBlockContainsNoRules	Ce problème se pose lorsque toutes les règles d'un bloc d'ensemble de règles sont supprimées et qu'une tentative de validation ou de publication de cet ensemble de règles a lieu. Dans ce cas, le bloc de règles de cet ensemble doit comporter au minimum une règle.
ProblemTemplateNotAssociatedWithRuleSet	Ce problème se pose en cas de tentative d'ajout d'une règle à un ensemble de règles, alors que cette règle a été créée avec un modèle non défini au sein de cet ensemble. Pour éviter ce problème, lorsque vous créez une nouvelle règle, vous devez utiliser un modèle défini au sein de l'ensemble de règles correspondant.
ProblemRuleNameAlreadyInUse	Ce problème se pose en cas de tentative d'ajout d'une règle à un bloc d'ensemble de règles et que cette règle porte le même nom qu'une règle déjà existante au sein de ce bloc de règles. Pour éviter ce problème, vous devez vérifier les noms des règles avant l'ajout de nouvelles règles.
ProblemTemplateParameterNotSpecified	Ce problème se pose lorsqu'un paramètre est absent lors d'une mise à jour de modèle pour l'une des règles d'un ensemble de règles ou d'une valeur d'action ou de condition d'une table de décision. Pour éviter ce problème, vous devez spécifier tous les paramètres d'un modèle.
ProblemTypeConversionError	Ce problème se pose lorsqu'un paramètre de modèle ne peut pas être converti dans le type approprié ; tous les paramètres sont considérés comme des objets String, puis convertis dans le type du paramètre (boolean, byte, short, int, long, float et double). Si la chaîne de la valeur de paramètre ne peut pas être convertie dans le type spécifié pour ce paramètre, cette erreur se produit. Pour éviter ce problème, vous devez spécifier une chaîne pouvant être convertie dans le type du paramètre (boolean, byte, short, int, long, float et double).
ProblemValueViolatesParmConstraints	Ce problème se pose lorsqu'un paramètre ne se trouve pas dans l'énumération ou dans la plage de valeurs définie dans le modèle de ce paramètre. Ce problème peut concerner les paramètres limités au niveau des énumérations ou des plages (modèles de règles d'un ensemble de règles ou valeur d'action ou de condition d'une table de décision). Pour éviter ce problème, vous devez utiliser une valeur contenue dans la plage d'énumération.
ProblemInvalidActionValueTemplate	Ce problème se pose en cas de tentative de définition d'une instance de modèle dans une opération de l'arborescence, mais que le modèle correspondant n'est pas disponible pour cette opération. Pour éviter ce problème, vous devez utiliser le modèle approprié lors de la création d'une définition de valeur pour une opération de l'arborescence.
ProblemInvalidConditionValueTemplate	Ce problème se pose en cas de tentative de définition d'une instance de modèle pour une condition de cas, alors que le modèle correspondant n'est pas disponible pour ce cas. Pour éviter ce problème, utilisez le modèle approprié lors de la création d'une définition de condition pour un cas spécifique.

Tableau 5. Problèmes liés aux ensembles de règles et aux tables de décisions (suite)

Exception	Description
ProblemTreeActionIsNull	Ce problème se pose lorsqu'une valeur de condition est créée et qu'aucune action n'a pas été définie avec une instance de modèle. Dans ce cas, vous devez utiliser un modèle provenant de ActionNode, créer une nouvelle instance de modèle et la définir dans la liste TreeActions.

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Autorisation

Les classes ne prennent en charge aucun niveau d'autorisation. L'application client utilisant les classes doit ajouter sa propre méthode d'autorisation.

Concepts associés

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Ils contiennent plusieurs projets.

- **BRMgmtExamples** – Projet de module contenant des artefacts de règles métier utilisés dans les différents exemples.
- **BRMgmt** – Projet Java dont les exemples figurent dans le package `com.ibm.websphere.sample.brules.mgmt`.
- **BRMgmtDriverWeb** – Projet Web avec interface pour l'exécution des exemples.

Les exemples sont également fournis sous forme de fichier EAR (`BRMgmtExamples.ear`) qui peuvent être émis après leur installation dans WebSphere Process Server. Une interface Web est fournie avec les exemples. Cette interface est volontairement simple, car les exemples concernent l'utilisation des classes pour l'extraction d'artefacts, l'apport de modifications et la publication de celles-ci. Elle n'est pas destinée à être une interface Web hautes performances. Les classes peuvent cependant être facilement utilisées pour l'élaboration d'interfaces Web robustes, ou encore dans d'autres applications Java pour la modification des règles métier.

Remarque : Vous pouvez télécharger les fichiers d'échange de projet (.zip) et EAR des exemples à partir de la page Business Rule Management Programming Guide for WebSphere Process Server V6.1.

L'application d'exemples peut être installée sur WebSphere Process Server v6.1 ; la page d'index est accessible à l'adresse suivante :

`http://<nom_hôte>:<port>/BRMgmtDriverWeb/`

Par exemple : `http://localhost:9080/BRMgmtDriverWeb/`

Au fur et à mesure de l'émission des exemples, des modifications seront apportées aux artefacts des règles. Si tous les exemples sont émis, l'application devra être réinstallée afin d'afficher de nouveau les mêmes résultats pour tous les exemples.

Les exemples sont détaillés, avec des exemples de code et le résultat tel qu'il s'affiche dans un navigateur Web.

Des classes supplémentaires ont été créées pour l'exécution d'opérations courantes et pour faciliter l'affichage des informations dans l'exemple d'application Web. Pour plus d'informations sur les classes `Formatter` et `RuleArtifactUtility`, voir l'annexe.

Pour bien comprendre ces exemples, il convient d'examiner les différents artefacts contenus dans WebSphere Integration Developer.

Concepts associés

Gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Exemple 1 : extraction et impression de l'ensemble des groupes de règles métier
Cet exemple présente l'extraction de tous les groupes de règles métier et l'impression des attributs, des propriétés et des opérations de chaque groupe de règles métier.

Exemple 2 : Extraire et afficher tous les groupes de règles métier, les jeux de règles et les tables de décision

Outre la fonction de l'exemple 1, cet exemple permet d'imprimer la table de sélection pour chaque opération, puis la destination des règles métier par défaut (jeu de règles ou table de décision) et les autres règles métier planifiées pour l'opération. Il imprime à la fois le jeu de règles et les tables de décision.

Exemple 3 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur AND

Cet exemple est également similaire à l'exemple 1, mais il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur "accounting", ainsi que la propriété RuleType et la valeur "regulatory".

Exemple 4 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur OR

Cet exemple est similaire à l'exemple 3 ; toutefois, il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur "accounting", ou encore la propriété RuleType et la valeur "monetary".

Exemple 5 : extraction de groupes de règles métier à l'aide d'une requête complexe
Cet exemple constitue une combinaison des exemples 3 et 4 ; il a pour but d'illustrer la création de requêtes plus complexes. Dans cet exemple, une recherche est effectuée à l'aide d'une requête qui associe 2 conditions de requête. La première condition de requête consiste à extraire les groupes de règles métier possédant la propriété Department et la valeur "General", ou encore la propriété MissingProperty et la valeur "somevalue". Cette condition de requête est ensuite associée, à l'aide d'un opérateur AND, à une condition contenant la propriété RuleType et la valeur "messages".

Exemple 6 : mise à jour d'une propriété de groupe de règles métier et publication du groupe de règles métier

Dans cet exemple, l'une des propriétés d'un groupe de règles métier est mise à jour, puis le groupe de règles métier correspondant est publié.

Exemple 7 : mise à jour des propriétés contenues dans plusieurs groupes de règles métier et publication des groupes de règles métier correspondants.

Dans cet exemple, les propriétés de plusieurs groupes de règles métier sont mises à jour avant la publication des groupes de règles métier correspondants.

Exemple 8 : modification de la règle métier par défaut d'un groupe de règles métier

Dans cet exemple, la règle métier par défaut est remplacée par une autre règle métier faisant partie de la liste de cibles disponibles d'une opération spécifique.

Exemple 9 : planification d'une autre règle d'opération au sein d'un groupe de règles métier

Dans cet exemple, une règle métier est planifiée en vue d'être active pendant une durée d'une heure à compter de l'heure de la publication d'une opération spécifique.

Exemple 10 : modification d'une valeur de paramètre dans un modèle d'un ensemble de règles

Dans cet exemple, une instance de règle définie avec un modèle est modifiée en changeant une valeur de paramètre, puis publiée.

Exemple 11 : Ajouter une nouvelle règle depuis un modèle vers un jeu de règles
Dans cet exemple, une nouvelle règle est ajoutée à un jeu de règles, à partir d'un modèle. Avant la création de l'instance de règle, des paramètres sont définis pour cette instance.

Exemple 12 : Modifier et publier un modèle d'une table de décision en changeant la valeur d'un paramètre

Dans cet exemple, une condition et une action (toutes deux définies avec des modèles) sont modifiées dans une table de décision, en changeant les valeurs des paramètres avant publication.

Exemple 13 : Ajout d'une valeur de condition et d'actions dans une table de décision

Dans cet exemple, une valeur de condition et une action vont être ajoutées à une table de décision. Pour ajouter une valeur de condition à une table de décision, vous pouvez utiliser un modèle.

Exemple 14 : Gestion des erreurs dans un jeu de règles

Cet exemple explique comment identifier des incidents dans un jeu de règles et déterminer la nature de l'incident, afin d'afficher le message approprié ou de mettre en oeuvre l'action nécessaire pour corriger la situation.

Exemple 15 : Gestion des erreurs dans un groupe de règles métier

Cet exemple est similaire à l'exemple 14, car il montre comment gérer les incidents qui peuvent se produire lors de la publication d'un groupe de règles métier. Il montre comment déterminer la nature de l'incident afin d'imprimer le message correspondant ou d'exécuter l'action appropriée.

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Exemple 1 : extraction et impression de l'ensemble des groupes de règles métier

Cet exemple présente l'extraction de tous les groupes de règles métier et l'impression des attributs, des propriétés et des opérations de chaque groupe de règles métier.

```
package com.ibm.websphere.sample.brules.mgmt;
```

```
import java.util.Iterator;  
import java.util.List;
```

En ce qui concerne les classes de gestion des règles métier, veillez à les utiliser dans le module `com.ibm.wbiserver.brules.mgmt`, et pas dans le module `com.ibm.wbiserver.brules` ou dans tout autre module. Ces autres modules concernent les classes internes IBM.

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;  
import  
com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;  
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;  
import com.ibm.wbiserver.brules.mgmt.Operation;  
import com.ibm.wbiserver.brules.mgmt.Property;  
import com.ibm.wbiserver.brules.mgmt.PropertyList;
```

```
public class Example1 {  
    static Formatter out = new Formatter();  
    static public String executeExample1()
```

```

{
try
{
out.clear();

```

La classe `BusinessRuleManager` est la principale classe d'extraction des groupes de règles métier et de publication des modifications des groupes de règles métier. Cela inclut l'utilisation et la modification des artefacts de règle tels que les ensembles de règles et les tables de décision. La classe `BusinessRuleManager` comporte de nombreuses méthodes permettant de simplifier l'extraction de groupes de règles métier spécifiques par nom, espace de nom et propriétés.

```

// Récupérer tous les groupes de règles métier
List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBusinessRuleGroups(0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Procéder à une itération via la liste des groupes de règles métier
while (iterator.hasNext())
{
    brg = iterator.next();
    // Sortir les attributs de chaque groupe de règles métier
    out.printlnBold("Business Rule Group");

```

Les attributs de base du groupe de règles métier peuvent être extraits et affichés.

```

out.println("Nom : " + brg.getName());
out.println("Espace de nom : " +
brg.getTargetNameSpace());
out.println("Nom affiché : " +
brg.getDisplayName());
out.println("Description : " + brg.getDescription());
out.println("Fuseau horaire de présentation : "
    + brg.getPresentationTimezone());
out.println("Date de sauvegarde : " + brg.getSaveDate());

```

Les propriétés du groupe de règles métier peuvent également être extraites et modifiées.

```

PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Sortir des valeurs et des noms de propriétés
while (propIterator.hasNext())
{
    prop = propIterator.next();
    out.println("Nom de propriété : " +
prop.getName());
    out.println("Valeur de propriété : " +
prop.getValue());
}

```

Les opérations du groupe de règles métier sont également disponibles, et permettent d'extraire les artefacts de règles métier tels que les ensembles de règles et les tables de décision.

```

List<Operation> opList = brg.getOperations();

Iteration<Operation> opIterator = opList.iterator();
Operation op = null;
// Sortir les opérations du groupe de règles métier
while (opIterator.hasNext())

```

```

{
op = opIterator.next();
out.println("Opération : " + op.getName());
}
out.println("");}
} catch (BusinessRuleManagementException e)
{
e.printStackTrace();
out.println(e.getMessage());
}
return out.toString();
}
}

```

Sortie du navigateur Web pour l'exemple 1.

Exécution de l'exemple 1

Groupe de règles métier

Nom : ApprovalValues
 Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
 Nom affiché : ApprovalValues
 Description : null
 Fuseau horaire de présentation : LOCAL
 Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
 Nom de propriété : IBMSysVersion
 Valeur de propriété : 6.2.0
 Nom de propriété : Department
 Valeur de propriété : Accounting
 Nom de propriété : RuleType
 Valeur de propriété : regulatory
 Nom de propriété : IBMSysTargetNameSpace
 Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
 Nom de propriété : IBMSysName
 Valeur de propriété : ApprovalValues
 Nom de propriété : IBMSysDisplayName
 Valeur de propriété : ApprovalValues
 Opération : getApprover

Groupe de règles métier

Nom : ConfigurationValues
 Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
 Nom affiché : ConfigurationValues
 Description : null
 Fuseau horaire de présentation : LOCAL
 Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
 Nom de propriété : IBMSysVersion
 Valeur de propriété : 6.2.0
 Nom de propriété : Department
 Valeur de propriété : General
 Nom de propriété : RuleType
 Valeur de propriété : messages
 Nom de propriété : IBMSysTargetNameSpace
 Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
 Nom de propriété : IBMSysName
 Valeur de propriété : ConfigurationValues
 Nom de propriété : IBMSysDisplayName
 Valeur de propriété : ConfigurationValues
 Opération : getMessages

Groupe de règles métier

Nom : DiscountRules
 Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
 Nom affiché : DiscountRules
 Description : null
 Fuseau horaire de présentation : LOCAL
 Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008

Nom de propriété : Department
 Valeur de propriété : Accounting
 Nom de propriété : IBMSystemVersion
 Valeur de propriété : 6.2.0
 Nom de propriété : RuleType
 Valeur de propriété : monetary
 Nom de propriété : IBMSystemTargetNameSpace
 Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
 Nom de propriété : IBMSystemName
 Valeur de propriété : DiscountRules
 Nom de propriété : IBMSystemDisplayName
 Valeur de propriété : DiscountRules
 Opération : calculateOrderDiscount
 Opération : calculateShippingDiscount

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 2 : Extraire et afficher tous les groupes de règles métier, les jeux de règles et les tables de décision

Outre la fonction de l'exemple 1, cet exemple permet d'imprimer la table de sélection pour chaque opération, puis la destination des règles métier par défaut (jeu de règles ou table de décision) et les autres règles métier planifiées pour l'opération. Il imprime à la fois le jeu de règles et les tables de décision.

La majeure partie de l'exemple est identique, mais répétée à des fins d'exhaustivité.

```

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
public class Example2
{
    status Formatter out = new Formatter();
    static public String executeExample2()
    {
        try
        {
            out.clear();

```

Un groupe de règles métier spécifique est extrait par son nom pour cet exemple.

```

// Extraction de tous les groupes de règles métier
List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByName("DiscountRules",
        QueryOperator.EQUAL, 0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

```

```

BusinessRuleGroup brg = null;
// Itération dans la liste des groupes de règles métier
while (iterator.hasNext())
{
    brg = iterator.next();
    // Extraction des attributs pour chaque groupe de règles métier
    out.printlnBold("Groupe de règles métier");
    out.println("Nom: " + brg.getName());
    out.println("Espace de nom: " +
        brg.getTargetNameSpace());
    out.println("Nom affiché: " +
        brg.getDisplayName());
    out.println("Description: " + brg.getDescription());
    out.println("Fuseau horaire de présentation: "
        + brg.getPresentationTimezone());
    out.println("Date d'enregistrement: " + brg.getSaveDate());

    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
propList.iterator();
    Property prop = null;
    // Extraction des noms et des valeurs des propriétés
    while (propIterator.hasNext())
    {
        prop = propIterator.next();
        out.println("Nom de la propriété: " +
            prop.getName());
        out.println("Valeur de la propriété: " +
            prop.getValue());
    }
}

```

Pour chaque opération, une table de sélection comporte une liste des différents artefacts de règle et leurs périodes d'activité planifiées. Une règle métier par défaut peut être spécifiée pour chaque opération. Il n'est pas obligatoire de spécifier une règle métier par défaut ou d'avoir une règle métier planifiée. Toutefois, vous devez avoir au moins une règle métier par défaut ou une règle métier spécifiée. Par conséquent, il est conseillé de rechercher les valeurs null avant d'utiliser la règle métier par défaut, ou de vérifier la taille de la liste `OperationSelectionRecordList`.

```

List<Operation> opList = brg.getOperations();

Iterator<Operation> opIterator = opList.iterator();
Operation op = null;
out.println("");
out.printlnBold("Opérations");
// Extraction des opérations pour le groupe de règles métier
while (opIterator.hasNext())
{
    op = opIterator.next();
    out.printBold("Opération: ");
    out.println(op.getName());

    // Extraction de la règle métier par défaut pour l'opération
    BusinessRule defaultRule =
op.getDefaultBusinessRule();
    // Si la règle par défaut est localisée, imprimer cette règle
    // à l'aide de la méthode appropriée pour le type de règle
    if (defaultRule != null)
    {
        out.printlnBold("Destination par défaut:");
    }
}

```

La règle métier par défaut est de type `RuleSet` ou `DecisionTable`, et peut être convertie dans le type approprié pour traiter l'artefact de règle.

```

        if (defaultRule instanceof RuleSet)
            out.println(RuleArtifactUtility.
                intRuleSet(defaultRule));
        else
            out.print(RuleArtifactUtility.
                tDecisionTable(defaultRule));
    }
    OperationSelectionRecordList
    opSelectionRecordList = op
        .getOperationSelectionRecordList()
        ;

    Iterator<OperationSelectionRecord>
    opSelRecordIterator = opSelectionRecordList
        .iterator();
    OperationSelectionRecord record = null;

```

L'élément OperationSelectionRecord est composé de l'artefact de règle et des périodes d'activité de cet artefact de règle.

```

    while (opSelRecordIterator.hasNext())
    {
        out.printlnBold("Destination
            planifiée:");
        record = opSelRecordIterator.next();

        out.println("Date de début: " +
            record.getStartDate()
            + " - Date de fin: " +
            record.getEndDate());
        BusinessRule ruleArtifact = record
            .getBusinessRuleTarget();

        if (ruleArtifact instanceof RuleSet)
            out.println(RuleArtifactUtility.pr
                intRuleSet(ruleArtifact));
        else
            out.print(RuleArtifactUtility.prin
                tDecisionTable(ruleArtifact));
    }
}
}
}
out.println("");
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
    return out.toString();
}
}
}

```

Exemple

Sortie du navigateur Web pour l'exemple 2.

Groupe de règles métier

```

Nom : DiscountRules
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : DiscountRules
Description : null
Fuseau horaire de présentation : LOCAL
Date d'enregistrement : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : RuleType

```

Valeur de la propriété : monetary
Nom de la propriété : IBMSYSTEMTARGETNAMESPACE
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSYSTEMNAME
Valeur de la propriété : DiscountRules
Nom de la propriété : IBMSYSTEMDISPLAYNAME
Valeur de la propriété : DiscountRules

Opérations

Opération : calculateOrderDiscount

Destination par défaut :

Jeu de règles

Nom : calculateOrderDiscount

Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Règle : CopyOrder

Nom affiché : CopyOrder

Description : null

Présentation utilisateur détaillée : null

Présentation utilisateur : null

Règle : FreeGiftInitialization

Nom affiché : FreeGiftInitialization

Description : null

Présentation utilisateur détaillée : ID produit pour la cadeau gratuit = 5001AE80

Quantité = 1 Coût =

Description 0.0 = Cadeau gratuit pour une commande avec remise

Présentation utilisateur : ID produit pour le cadeau gratuit = {0}

Quantité = {1} Coût = {2}

Description = {3}Parameter Name: param0

Valeur du paramètre : 5001AE80

Nom du paramètre : param1

Valeur du paramètre : 1

Nom du paramètre : param2

Valeur du paramètre : 0.0

Nom du paramètre : param3

Valeur du paramètre : Cadeau gratuit pour une commande avec remise

Règle : Rule1

Nom affiché : Rule1

Description : null

Présentation utilisateur détaillée : Si le client a le statut gold, appliquer une remise de 20,0 et ajouter un cadeau gratuit

Présentation utilisateur : Si le client a le statut {0}, appliquer une remise de {1} et ajouter un cadeau gratuit

Nom du paramètre : param0

Valeur du paramètre : gold

Nom du paramètre : param1

Valeur du paramètre : 20.0

Règle : Rule2

Nom affiché : Rule2

Description : null

Présentation utilisateur détaillée : Si customer.status == silver, appliquer une remise de 15,0

Présentation utilisateur : Si customer.status == {0}, appliquer une remise de {1}

Nom du paramètre : param0

Valeur du paramètre : silver

Nom du paramètre : param1

Valeur du paramètre : 15.0

Règle : Rule3

Nom affiché : Rule3

Description : Modèle pour les clients qui n'ont pas le statut gold

Présentation utilisateur détaillée : Si customer.status == bronze, appliquer une remise de 10,0

Présentation utilisateur : Si customer.status == {0}, appliquer une remise de {1}

Nom du paramètre : param0

Valeur du paramètre : bronze

Nom du paramètre : param1

Valeur du paramètre : 10.0

Opération : calculateShippingDiscount
Destination par défaut :
Table de décision
Nom : calculateShippingDiscount
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Règle d'initialisation : Rule1
Nom affiché : Rule1
Description : null
Présentation utilisateur détaillée : null
Présentation utilisateur : null

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 3 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur AND

Cet exemple est également similaire à l'exemple 1, mais il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur "accounting", ainsi que la propriété RuleType et la valeur "regulatory".

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example3
{
    static Formatter out = new Formatter();
    static public String executeExample3()
    {
        try
        {
            out.clear();

```

Les requêtes de groupes de règles métier sont composées de noeuds de requêtes qui suivent une arborescence. Chaque noeud de requête contient un terme gauche et un terme droit. Chacun de ces termes peut représenter un autre noeud de requête. Dans cet exemple, le groupe de règles métier est extrait via la combinaison de deux valeurs de propriété.

```
// Extrait les groupes de règles métier sur la base de deux conditions
// Crée des noeuds PropertyQueryNodes pour chaque condition
PropertyQueryNode propertyNode1 = QueryNodeFactory
    .createPropertyQueryNode("Department",
        QueryOperator.EQUAL,"Accounting");
PropertyQueryNode propertyNode2 = QueryNodeFactory
    .createPropertyQueryNode("RuleType", QueryOperator.EQUAL,
        "regulatory");
// Associe les deux noeuds PropertyQueryNodes à un noeud AND
```

```

AndNode andNode =
QueryNodeFactory.createAndNode(propertyNode1, propertyNode2);

// Utilise andNode lors des recherches de groupes de règles métier
List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByProperties(andNode, 0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Effectue une itération dans la liste des groupes de règles métier
while (iterator.hasNext())
{
    brg = iterator.next();
    // Permet d'obtenir les attributs de sortie de chaque groupe de règles métier
    out.printlnBold("Business Rule Group");
    out.println("Name: " + brg.getName());
    out.println("Namespace: " +
brg.getTargetNameSpace());
    out.println("Display Name: " + brg.getDisplayName());
    out.println("Description: " + brg.getDescription());
    out.println("Presentation Time zone: "
+ brg.getPresentationTimezone());
    out.println("Save Date: " + brg.getSaveDate());

    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
propList.iterator();
    Property prop = null;
    // Permet d'obtenir les noms et valeurs des propriétés
    while (propIterator.hasNext())
    {
        prop = propIterator.next();
        out.println("\t Property Name: " +
prop.getName());
        out.println("\t Property Value: " +
prop.getValue());
    }
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Résultat de navigateur Web pour l'exemple 3.

Exécution de l'exemple 3

```

Groupe de règles métier
Nom : ApprovalValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ApprovalValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSysVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : RuleType

```

Valeur de la propriété : regulatory
 Nom de la propriété : IBMSystemTargetNameSpace
 Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
 Nom de la propriété : IBMSystemName
 Valeur de la propriété : ApprovalValues
 Nom de la propriété : IBMSystemDisplayName
 Valeur de la propriété : ApprovalValues

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 4 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur OR

Cet exemple est similaire à l'exemple 3 ; toutefois, il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur "accounting", ou encore la propriété RuleType et la valeur "monetary".

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example4
{
    static Formatter out = new Formatter();
    static public String executeExample4()
    {
        try
        {
            out.clear();
```

Différentes propriétés composent la requête et permettent de renvoyer différents groupes de règles métier.

```
// Retrieve business rule groups based on two conditions
// Crée des noeuds PropertyQueryNodes pour chaque condition
PropertyQueryNode propertyNode1 = QueryNodeFactory
    .createPropertyQueryNode("Department",
        QueryOperator.EQUAL, "Accounting");
PropertyQueryNode propertyNode2 = QueryNodeFactory
    .createPropertyQueryNode("RuleType",
        QueryOperator.EQUAL, "monetary");
// Associe les deux noeuds PropertyQueryNodes à un noeud OR
OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1,
        propertyNode2);
// Utilise orNode dans les recherches de groupes de règles métier
List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByProperties(orNode, 0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();
```

```

BusinessRuleGroup brg = null;
// Effectue une itération dans la liste des groupes de règles métier
while (iterator.hasNext())
{
    brg = iterator.next();
    // Permet d'obtenir les attributs de chaque groupe de règles métier
    out.printlnBold("Business Rule Group");
    out.println("Name: " + brg.getName());
    out.println("Namespace: " +
        brg.getTargetNameSpace());
    out.println("Display Name: " + brg.getDisplayName());
    out.println("Description: " + brg.getDescription());
    out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
    out.println("Save Date: " + brg.getSaveDate());

    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
        propList.iterator();
    Property prop = null;
    // Permet d'obtenir les noms et valeurs de propriétés
    while (propIterator.hasNext())
    {
        prop = propIterator.next();
        out.println("\t Property Name: " +
            prop.getName());
        out.println("\t Property Value: " +
            prop.getValue());
    }
    out.println("");
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Résultat de navigateur Web pour l'exemple 4.

Exécution de l'exemple 4

Groupe de règles métier

```

Nom : ApprovalValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ApprovalValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSysVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : RuleType
Valeur de la propriété : regulatory
Nom de la propriété : IBMSysTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSysName
Valeur de la propriété : ApprovalValues
Nom de la propriété : IBMSysDisplayName
Valeur de la propriété : ApprovalValues

```


Groupe de règles métier

Nom : DiscountRules
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : DiscountRules
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : IBMSYSTEMVERSION
Valeur de la propriété : 6.2.0
Nom de la propriété : RuleType
Valeur de la propriété : monetary
Nom de la propriété : IBMSYSTEMTARGETNAMESPACE
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSYSTEMNAME
Valeur de la propriété : DiscountRules
Nom de la propriété : IBMSYSTEMDISPLAYNAME
Valeur de la propriété : DiscountRules

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 5 : extraction de groupes de règles métier à l'aide d'une requête complexe

Cet exemple constitue une combinaison des exemples 3 et 4 ; il a pour but d'illustrer la création de requêtes plus complexes. Dans cet exemple, une recherche est effectuée à l'aide d'une requête qui associe 2 conditions de requête. La première condition de requête consiste à extraire les groupes de règles métier possédant la propriété Department et la valeur "General", ou encore la propriété MissingProperty et la valeur "somevalue". Cette condition de requête est ensuite associée, à l'aide d'un opérateur AND, à une condition contenant la propriété RuleType et la valeur "messages".

D'autres exemples de requêtes de groupes de règles métier figurent dans l'Annexe.

```
package com.ibm.websphere.sample.brules.mgmt;
```

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example5
{
    static Formatter out = new Formatter();
    static public String executeExample5()
    {
        try
```

```

{
    out.clear();

    // Extrait les groupes de règles métier sur la base de trois conditions ;
    // deux de celles-ci sont combinées au sein d'un noeud OR
    // Crée des noeuds PropertyQueryNodes pour chaque condition du noeud OR
    PropertyQueryNode propertyNode1 = QueryNodeFactory
        .createPropertyQueryNode("Department",
            QueryOperator.EQUAL, "General");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
        .createPropertyQueryNode("MissingProperty",
            QueryOperator.EQUAL, "SomeValue");
    // Combine les deux PropertyQueryNodes au sein d'un noeud OR
    OrNode orNode =
        QueryNodeFactory.createOrNode(propertyNode1, propertyNode2);
    // Crée le troisième noeud PropertyQueryNode
    PropertyQueryNode propertyNode3 = QueryNodeFactory
        .createPropertyQueryNode("RuleType",
            QueryOperator.EQUAL, "messages");

```

La partie gauche de la condition est combinée à la partie droite à l'aide d'un noeud AND. AndNode constitue la racine de l'arborescence de requête.

```

// Combine le noeud OR avec le troisième PropertyQueryNode à l'aide de :
AndNode andNode =
    QueryNodeFactory.createAndNode(propertyNode3, orNode);

List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByProperties(andNode, 0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Effectue une itération dans la liste des groupes de règles métier
while (iterator.hasNext())
{
    brg = iterator.next();
    // Permet d'obtenir les attributs de chaque groupe de règles métier
    out.printlnBold("Business Rule Group");
    out.println("Name: " + brg.getName());
    out.println("Namespace: " +
        brg.getTargetNameSpace());
    out.println("Display Name: " + brg.getDisplayName());
    out.println("Description: " + brg.getDescription());
    out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
    out.println("Save Date: " + brg.getSaveDate());
    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
        propList.iterator();
    Property prop = null;
    // Permet d'obtenir les noms et valeurs de propriétés
    while (propIterator.hasNext())
    {
        prop = propIterator.next();
        out.println("\t Property Name: " +
            prop.getName());
        out.println("\t Property Value: " +
            prop.getValue());
    }
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}

```

```

    }
    return out.toString();
  }
}

```

Exemple

Résultat de navigateur Web pour l'exemple 5.

Exécution de l'exemple 5

Groupe de règles métier

```

Nom : ConfigurationValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ConfigurationValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSysVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : General
Nom de la propriété : RuleType
Valeur de la propriété : messages
Nom de la propriété : IBMSysTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSysName
Valeur de la propriété : ConfigurationValues
Nom de la propriété : IBMSysDisplayName
Valeur de la propriété : ConfigurationValues

```

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 6 : mise à jour d'une propriété de groupe de règles métier et publication du groupe de règles métier

Dans cet exemple, l'une des propriétés d'un groupe de règles métier est mise à jour, puis le groupe de règles métier correspondant est publié.

```
package com.ibm.websphere.sample.brules.mgmt;
```

```
import java.util.ArrayList;
import java.util.List;
```

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
```

```
public class Example6
```

```
{
    static Formatter out = new Formatter();
```

```
    static public String executeExample6()
```

```
{
    try
    {
```

```
        out.clear();
```

```
        out.printlnBold("Business Rule Group before publish:");
```

```
        // Extrait les groupes de règles métier à l'aide d'une seule valeur de propriété
```

```

List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL,"General", 0, 0);

if (brgList.size() > 0)
{
    // Extrait le premier groupe de règles métier de la liste
    BusinessRuleGroup brg = brgList.get(0);
    // Extrait la propriété du groupe de règles métier
    UserDefinedProperty userDefinedProperty =
        (UserDefinedProperty) brg
            .getProperty("Department");

    out.println("Business Rule Group: " + brg.getName());
    out.println("Department Property value: "
        + brg.getProperty("Department").getValue());
}

```

La méthode `getProperty` renvoie une propriété par référence ; les modifications apportées à la propriété sont directement répercutées au niveau du groupe de règles métier.

```

// Modification de la valeur de propriété du groupe de règles métier
// Cela permet de mettre à jour la valeur de la propriété directement dans
// l'objet du groupe de règles métier
userDefinedProperty.setValue("GeneralConfig");
// Utilise la liste d'origine ou crée une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();
// Ajoute le groupe de règles métier modifié à la liste
publishList.add(brg);

```

La classe `BusinessRuleManager` est utilisée pour la publication des modifications apportées à un groupe de règles métier. Pour publier ces modifications, une liste est transférée à la méthode de publication `BusinessRuleManager`, même si un seul élément est publié.

```

// Publie la liste contenant le groupe de règles métier modifié
BusinessRuleManager.publish(publishList, true);

out.println("");

// Extrait de nouveau le groupe de règles métier pour vérifier que les
// modifications ont été publiées
out.printlnBold("Business Rule Group after publish:");
brgList = BusinessRuleManager
    .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL, "GeneralConfig", 0, 0);

brg = brgList.get(0);

out.println("Business Rule Group: " + brg.getName());
// Affiche la valeur de propriété pour indiquer la modification apportée
out.println("Department Property value: "
    + brg.getProperty("Department").getValue());
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Résultat de navigateur Web pour l'exemple 6.

Exécution de l'exemple 6

Groupe de règles métier avant la publication :
Groupe de règles métier : ConfigurationValues
Valeur de la propriété Department : General

Groupe de règles métier après la publication :
Groupe de règles métier : ConfigurationValues
Valeur de la propriété Department : GeneralConfig

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 7 : mise à jour des propriétés contenues dans plusieurs groupes de règles métier et publication des groupes de règles métier correspondants.

Dans cet exemple, les propriétés de plusieurs groupes de règles métier sont mises à jour avant la publication des groupes de règles métier correspondants.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example7
{
    static Formatter out = new Formatter();

    static public String executeExample7()
    {
        try
        {
            out.clear();
            out.printlnBold("Business Rule Group before publish:");
            // Extrait les groupes de règles métier à l'aide d'une seule valeur de propriété
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsBySingleProperty("Department",
                    QueryOperator.EQUAL, "Accounting", 0, 0);

            Iterator<BusinessRuleGroup> iterator = brgList.iterator();

            BusinessRuleGroup brg = null;

            // Utilise la liste d'origine ou crée une nouvelle liste
            // de groupes de règles métier
            List<BusinessRuleGroup> publishList = new
                ArrayList<BusinessRuleGroup>();

            // Effectue une itération au sein de tous les groupes de règles métier et
            // modifie la propriété

```

```

while (iterator.hasNext())
{
    // Extrait la propriété du groupe de règles métier
    brg = iterator.next();

    out.println("Business Rule Group: " + brg.getName());

    // Extrait la propriété du groupe de règles métier
    UserDefinedProperty prop = (UserDefinedProperty) brg
        .getProperty("Department");
    out.println("Department Property value: "
        +
        brg.getProperty("Department").getValue())
        ;

    // Modifie la valeur de propriété dans le groupe de règles métier
    // Cela permet de mettre à jour la valeur de la propriété directement dans
    // l'objet du groupe de règles métier
    prop.setValue("Finance");
}

```

Chaque groupe de règles métier modifié est ajouté à la liste.

```

// Ajoute le groupe de règles métier modifié à la liste
publishList.add(brg);
}

// Publie la liste contenant le groupe de règles métier
// modifié
BusinessRuleManager.publish(publishList, true);

out.println("");

// Extrait de nouveau les groupes de règles métier afin de vérifier que
// les modifications ont été publiées
out.printlnBold("Business Rule Group after
publish:");

brgList = BusinessRuleManager
    .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL,
        "Finance", 0, 0);
iterator = brgList.iterator();

while (iterator.hasNext())
{
    brg = iterator.next();
    out.println("Business Rule Group: " +
        brg.getName());
    out.println("Department Property value: "
        +
        brg.getProperty("Department").getVa
        lue());
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Résultat de navigateur Web pour l'exemple 7.

Exécution de l'exemple 7

Groupe de règles métier avant la publication :

Groupe de règles métier : ApprovalValues
Valeur de la propriété Department : Accounting
Groupe de règles métier : DiscountRules
Valeur de la propriété Department : Accounting

Groupe de règles métier après la publication :

Groupe de règles métier : ApprovalValues
Valeur de la propriété Department : Finance
Groupe de règles métier : DiscountRules
Valeur de la propriété Department : Finance

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 8 : modification de la règle métier par défaut d'un groupe de règles métier

Dans cet exemple, la règle métier par défaut est remplacée par une autre règle métier faisant partie de la liste de cibles disponibles d'une opération spécifique.

```
package com.ibm.websphere.sample.brules.mgmt;
```

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

```
import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
```

```
public class Example8
{
    static Formatter out = new Formatter();

    static public String executeExample8()
    {
        try
        {
            out.clear();

            // Extrait un groupe de règles métier par espace de nom et nom cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere
                    /sample/brules",
                    QueryOperator.EQUAL,
                    "DiscountRules",
                    QueryOperator.EQUAL, 0, 0);

            if (brgList.size() > 0)
            {
                out.printlnBold("Business Rule Group before publish:");
                // Extrait le premier groupe de règles métier de la liste
                // Il doit s'agir du seul groupe de règles métier de la liste, car
                // la combinaison d'espace de nom et de nom cible est unique
                BusinessRuleGroup brg = brgList.get(0);
            }
        }
    }
}
```

```

out.print("Business Rule Group: ");
out.println(brg.getName());

// Extrait le fonctionnement du groupe de règles métier dont
// la règle métier par défaut doit être mise à jour
Operation op =
brg.getOperation("calculateShippingDiscount");

```

La règle métier par défaut est extraite avant d'être mise à jour à l'aide d'une autre règle métier faisant partie de la liste de cibles disponibles de l'opération. Les ensembles de règles et les tables de décisions sont spécifiques aux opérations ; seuls les artefacts de règles métier relatifs à une opération peuvent être définis en tant qu'artefacts par défaut ou être programmés à un autre moment pour cette opération.

```

// Extrait la règle métier par défaut de l'opération
BusinessRule defaultRule =
op.getDefaultBusinessRule();
out.print("Default Rule: ");
out.println(defaultRule.getName());

// Extrait la liste des règles métier disponibles pour cette
opération
List<BusinessRule> ruleList =
op.getAvailableTargets();

Iterator<BusinessRule> iterator =
ruleList.iterator();
BusinessRule rule = null;

// Recherche une règle métier différente de la règle
en cours d'utilisation
// règle métier
// par défaut
while (iterator.hasNext())
{
    rule = iterator.next();
    if
    (!defaultRule.getName().equals(rule.getName()))
    {

```

La règle métier par défaut est définie pour l'objet de l'opération. L'affectation de la valeur Null à la règle métier par défaut a pour effet de supprimer la règle métier par défaut de l'opération ; toutefois, il est recommandé de spécifier une règle métier par défaut pour chaque opération.

```

// Définit une autre règle
// métier par défaut
// Cette modification concerne directement
// l'objet de l'opération
op.setDefaultBusinessRule(rule);
break;
}
}
// Utilise la liste d'origine ou crée une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();
// Ajoute le groupe de règles métier modifié à la liste
publishList.add(brg);
// Publie la liste contenant le groupe de règles
métier modifié
BusinessRuleManager.publish(publishList, true);

out.println("");

```



```

// Extrait de nouveau les groupes de règles métier, afin de vérifier que
// les modifications ont été publiées

out.printlnBold("Business Rule Group after publish:");
brgList = BusinessRuleManager
.getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere/sample/brules",
    QueryOperator.EQUAL, "DiscountRules",
    QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
out.println("Business Rule Group: " + brg.getName());
op = brg.getOperation("calculateShippingDiscount");

// Extrait la règle métier par défaut de l'opération
defaultRule = op.getDefaultBusinessRule();
out.print("Default Rule: ");
out.println(defaultRule.getName());
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Résultat de navigateur Web pour l'exemple 8.

Exécution de l'exemple 8

Groupe de règles métier avant la publication :

Groupe de règles métier : DiscountRules
Règle par défaut : calculateShippingDiscount

Groupe de règles métier après la publication :

Groupe de règles métier : DiscountRules
Règle par défaut : calculateShippingDiscountHoliday

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 9 : planification d'une autre règle d'opération au sein d'un groupe de règles métier

Dans cet exemple, une règle métier est planifiée en vue d'être active pendant une durée d'une heure à compter de l'heure de la publication d'une opération spécifique.

```

package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;

```

```

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example9 {
    static Formatter out = new Formatter();

    static public String executeExample9()
    {
        try
        {
            out.clear();

            // Extrait un groupe de règles métier par espace de nom et nom cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere
                    /sample/brules",
                    QueryOperator.EQUAL,
                    "DiscountRules",
                    QueryOperator.EQUAL, 0, 0);

            if (brgList.size() > 0)
            {
                out.println("");
                out.printlnBold("Business Rule Group before publish:");
                // Extrait le premier groupe de règles métier de la liste
                // Il doit s'agir du seul groupe de règles métier de la liste, car
                // la
                // combinaison d'espace de nom et de nom cible est unique
                BusinessRuleGroup brg = brgList.get(0);

                // Extrait le fonctionnement du groupe de règles métier dont
                // une nouvelle règle métier doit être planifiée
                Operation op =
                    brg.getOperation("calculateShippingDiscount");

                printOperationSelectionRecord(op);
                // Extrait la liste des règles métier disponibles pour cette opération
                List<BusinessRule> ruleList =
                    op.getAvailableTargets();

                // Extrait la première règle de la liste, qui sera planifiée
                // pour l'opération
                BusinessRule rule = ruleList.get(0);

                // Extrait la liste des règles métier planifiées
                OperationSelectionRecordList opList = op
                    .getOperationSelectionRecordList();
                // Crée une date de fin pour la règle métier
                Date future = new Date();
                long futureTime = future.getTime() + 3600000;
            }
        }
    }
}

```

Pour les nouvelles règles planifiées, il est possible de spécifier une date de début et une date de fin. Si une valeur Null est affectée pour la date de début, cela indique que la règle sera active immédiatement au moment de la publication. Si une valeur Null est affectée à la date de fin, la règle ne comportera pas de date de fin. Les chevauchements de planification ne sont pas autorisés et peuvent être contrôlés via l'appel de la méthode `validate` au niveau de l'opération.

```

// Crée la nouvelle règle métier planifiée en indiquant la date
// actuelle, ce qui signifie que cette règle deviendra immédiatement active
// au moment de la
// publication, ainsi que la date future.
newOperationSelectionRecord(new Date(),
    new Date(futureTime), rule);
// Ajoute la nouvelle règle métier planifiée à la liste des
// règles planifiées
opList.addOperationSelectionRecord(newRecord);

```

Validation de l'opération afin de vérifier qu'il n'existe aucun chevauchement.

```

// Valide la liste afin de vérifier l'absence de chevauchements
List<Problem> problems = op.validate();
if (problems.size() == 0)
{
    // Utilise la liste d'origine ou crée une nouvelle liste
    // de groupes de règles métier
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();
    // Ajoute le groupe de règles métier modifié à la liste
    publishList.add(brg);
    // Publie la liste contenant le groupe de règles
    métier mis à jour
    BusinessRuleManager.publish(publishList, true);
    out.println("");

    // Extrait de nouveau les groupes de règles métier afin de
    vérifier que les
    // modifications ont été publiées
    out.printlnBold("Business Rule Group after
    publish:");

    brgList =
    BusinessRuleManager.getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "DiscountRules",
        QueryOperator.EQUAL, 0, 0);
    brg = brgList.get(0);

    op =
    brg.getOperation("calculateShippingDiscount");

    printOperationSelectionRecord(op);
}
// Gère l'erreur de validation
}
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
/*
Méthode d'impression de l'enregistrement de sélection d'opération. La
date de début et la date de fin sont imprimées, ainsi que le nom de l'artefact
de règle correspondant à l'heure planifiée.
*/
private static void printOperationSelectionRecord(Operation op)
{
    OperationSelectionRecordList opSelectionRecordList = op
        .getOperationSelectionRecordList();
    Iterator<OperationSelectionRecord> opSelRecordIterator =
    opSelectionRecordList

```

```

        .iterator();
        OperationSelectionRecord record = null;
        while (opSelRecordIterator.hasNext())
        {
            out.printlnBold("Scheduled Destination:");
            record = opSelRecordIterator.next();
            out.println("Start Date: " + record.getStartDate()
                + " - End Date: " + record.getEndDate());
            BusinessRule ruleArtifact = record.getBusinessRuleTarget();
            out.println("Rule: " + ruleArtifact.getName());
        }
    }
}

```

Exemple

Résultat de navigateur Web pour l'exemple 9.

Exécution de l'exemple 9

Groupe de règles métier avant la publication :

Destination planifiée :

Date de début : Thu Dec 01 00:00:00 CST 2005 -

Date de fin : Sun Dec 25 00:00:00 CST 2005

Règle : calculateShippingDiscountHoliday

Groupe de règles métier après la publication :

Destination planifiée :

Date de début : Thu Dec 01 00:00:00 CST 2005 -

Date de fin : Sun Dec 25 00:00:00 CST 2005

Règle : calculateShippingDiscountHoliday

Destination planifiée :

Date de début : Mon Jan 07 21:08:31 CST 2008 -

Date de fin : Mon Jan 07 22:08:31 CST 2008

Règle : calculateShippingDiscount

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 10 : modification d'une valeur de paramètre dans un modèle d'un ensemble de règles

Dans cet exemple, une instance de règle définie avec un modèle est modifiée en changeant une valeur de paramètre, puis publiée.

```

package com.ibm.websphere.sample.brules.mgmt;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;

```

```

public class Example10
{
static Formatter out = new Formatter();

static public String executeExample10()
{
    try
    {
        out.clear();

        // Extraire un groupe de règles métier par espace de nom cible et
        // par nom
        List<BusinessRuleGroup> brgList = BusinessRuleManager
            .getBRGsByTNSAndName(
                "http://BRSamples/com/ibm/websphere
                /sample/brules",
                QueryOperator.EQUAL,
                "ApprovalValues",
                QueryOperator.EQUAL, 0, 0);
        if (brgList.size() > 0)
        {
            // Obtenir le premier groupe de règles métier depuis la liste
            // Il doit être le seul groupe de règles métier de la
            // liste puisque
            // la combinaison de l'espace de nom cible et du nom est
            // unique
            BusinessRuleGroup brg = brgList.get(0);
            // Obtenir l'opération du groupe de règles métier comportant
            // la règle métier à modifier puisque
            // les règles métier sont associées à une opération
            // spécifique
            Operation op = brg.getOperation("getApprover");

            // Obtenir la règle métier de l'opération qui
            // sera modifiée
            List<BusinessRule> ruleList =
                op.getBusinessRulesByName(
                    "getApprover", QueryOperator.EQUAL, 0,
                    0);

            if (ruleList.size() > 0)
            {
                out.println("");
                out.printlnBold("Rule set before publish:");
                // Obtenir la règle à modifier. Les règles sont
                // uniques par
                // espace de nom cible et par nom, mais cet
                // exemple
                // comporte une seule règle métier intitulée
                // "getApprover"
                RuleSet ruleSet = (RuleSet) ruleList.get(0);
                out.print(RuleArtifactUtility.printRuleSet(rule
                    Set));
            }
        }
    }
}

```

Toutes les règles d'un ensemble de règles sont dans un bloc de règles. Un seul bloc de règles est pris en charge et la méthode `getFirstRuleBlock` doit être utilisée pour extraire le bloc de règles.

```

// Un ensemble de règles comporte toutes les règles définies dans un
// bloc de règles
RuleBlock ruleBlock =
    ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
    ruleBlock.iterator();

// Procéder à l'itération via les règles du bloc de règles

```

```

pour trouver
// l'instance de règle intitulée "LargeOrderApprover"
while (ruleIterator.hasNext())
{
    RuleSetRule rule = ruleIterator.next();

```

Si une règle n'est pas définie avec un modèle de règle, seule sa présentation Web peut être extraite. Aucune mise à jour ne peut être réalisée sur une règle non définie avec un modèle. Si le nom de la règle est inconnu, il est recommandé de vérifier si elle a été définie avec un modèle.

```

// La règle doit avoir été définie avec un
modèle
// pour pouvoir être modifiée. Vérifier
si la règle en
// cours est basée sur un modèle.
if (rule instanceof
RuleSetTemplateInstanceRule)
{

```

Utilisez l'objet `TemplateInstance` pour créer la règle.

```

// Obtenir l'instance du modèle de règle
RuleSetTemplateInstanceRule
templateInstance =
(RuleSetTemplateInstanceRule) rule;

// Rechercher l'instance de règle
correspondant
// à la règle à modifier
if
(templateInstance.getName().equals(
"LargeOrderApprover"))
{

```

Pour l'instance de modèle, seules les valeurs de paramètre peuvent être modifiées. Les paramètres sont modifiés en extrayant `ParameterValue` et en le définissant sur la valeur appropriée. Dans la mesure où `ParameterValue` est validé par référence, la mise à jour est effectuée directement sur la règle, l'ensemble de règles et le groupe de règles métier.

```

// Obtenir le paramètre de
l'instance de règle
ParameterValue parameter =
templateInstance
.getParameterValue("par
am2");

// Modifier la valeur du
paramètre
parameter.setValue("superviso
r");
break;
}
}

// Utiliser la liste d'origine ou créer une nouvelle liste de
// groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajouter le groupe de règles métier modifié à la liste
publishList.add(brg);

// Publier la liste avec le groupe de règles métier mis à
jour

```

```

        BusinessRuleManager.publish(publishList, true);

        out.println("");
        // Extraire de nouveau les groupes de règles métier pour vérifier
        que les
        // modifications ont été publiées
        out.printlnBold("Rule set after publish:");

        brgList = BusinessRuleManager
            .getBRGsByTNSAndName(
                "http://BRSSamples/com/ibm/websphere/sample/brules",
                QueryOperator.EQUAL, "ApprovalValues",
                QueryOperator.EQUAL, 0, 0);

        brg = brgList.get(0);
        op = brg.getOperation("getApprover");
        ruleList = op.getBusinessRulesByName(
            "getApprover", QueryOperator.EQUAL, 0,0);

        ruleSet = (RuleSet) ruleList.get(0);
        out.print(RuleArtifactUtility.printRuleSet(ruleSet));
    }
} catch (BusinessRuleManagementException e)
{
    e.printStackTrace();
    out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Sortie du navigateur Web pour l'exemple 10.

Exécution de l'exemple 10

Ensemble de règles avant la publication :

Ensemble de règles

Nom : getApprover

Espace de nom : http://BRSSamples/com/ibm/websphere/sample/brules

Règle : LargeOrderApprover

Nom affiché : LargeOrderApprover

Description : null

Présentation utilisateur détaillée : si le nombre d'éléments Commande est supérieur à 10 et que la commande dépasse 5000 \$, l'approbation du responsable est nécessaire

Présentation utilisateur : si le nombre d'éléments Commande est supérieur à {0} et que la commande dépasse {1} \$, l'approbation de {2} est nécessaire

Nom de paramètre : param0

Valeur de paramètre : 10

Nom de paramètre : param1

Valeur de paramètre : 5000

Nom de paramètre : param2

Valeur de paramètre : manager

Règle : DefaultApprover

Nom affiché : DefaultApprover

Description : null

Présentation utilisateur détaillée : approuver = peer

Présentation utilisateur : approuver = {0}

Nom de paramètre : param0

Valeur de paramètre : peer

Ensemble de règles une fois terminé :

Ensemble de règles

Nom : getApprover

Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Règle : LargeOrderApprover

Nom affiché : LargeOrderApprover

Description : null

Présentation utilisateur détaillée : si le nombre d'éléments Commande est supérieur à 10 et que la commande dépasse 5000 \$, l'approbation du superviseur est nécessaire

Présentation utilisateur : si le nombre d'éléments Commande est supérieur à {0} et que la commande dépasse {1} \$, l'approbation de {2} est nécessaire

Nom de paramètre : param0

Valeur de paramètre : 10

Nom de paramètre : param1

Valeur de paramètre : 5000

Nom de paramètre : param2

Valeur de paramètre : supervisor

Règle : DefaultApprover

Nom affiché : DefaultApprover

Description : null

Présentation utilisateur détaillée : approver = peer

Présentation utilisateur : approver = {0}

Nom de paramètre : param0

Valeur de paramètre : peer

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 11 : Ajouter une nouvelle règle depuis un modèle vers un jeu de règles

Dans cet exemple, une nouvelle règle est ajoutée à un jeu de règles, à partir d'un modèle. Avant la création de l'instance de règle, des paramètres sont définis pour cette instance.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example11
{
    static Formatter out = new Formatter();

    static public String executeExample11()
    {
        try
        {
            out.clear();
        }
    }
}
```



```

// Extraction d'un groupe de règles métier par nom et espace de nom
cible
List<BusinessRuleGroup> brgList = BusinessRuleManager
.getBRGsByTNSAndName(
"http://BRSamples/com/ibm/websphere
/sample/brules",
QueryOperator.EQUAL,
"ApprovalValues",
QueryOperator.EQUAL, 0, 0);

if (brgList.size() > 0)
{
// Extraction du premier groupe de règles métier de la liste
// Cela doit être le seul groupe de règles métier de la
liste car
// la combinaison de nom et d'espace de nom cible est
unique
BusinessRuleGroup brg = brgList.get(0);
// Extraction de l'opération du groupe de règles métier qui comporte
// la règle métier qui sera modifiée lorsque les
// règles métier seront associées à une opération
// spécifique
Operation op = brg.getOperation("getApprover");

// Extraction de la règle métier pour l'opération qui
sera modifiée
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
"getApprover", QueryOperator.EQUAL, 0,0);

if (ruleList.size() > 0)
{
out.println("");
out.printlnBold("Jeu de règles avant publication:");
// Extraction de la règle à modifier. Les règles sont uniques par
// nom et espace de nom cible, mais cet exemple utilise
// une seule règle métier appelée
"getApprover"
RuleSet ruleSet = (RuleSet) ruleList.get(0);
out.print(RuleArtifactUtility.printRuleSet(rule
Set));
}
}

```

Pour ajouter une nouvelle règle au jeu de règles, le modèle approprié doit être identifié dans le jeu de règles et une instance doit être créée à partir de ce modèle. Le modèle peut être localisé grâce à son nom.

```

// Extraction de la liste des modèles de règles
ListRuleSetRuleTemplate> ruleTemplates =
ruleSet
.getRuleTemplates();

Iterator<RuleSetRuleTemplate> templateIterator
= ruleTemplates
.iterator();

while (templateIterator.hasNext())
{
RuleSetRuleTemplate template =
templateIterator.next();

// Localisation du modèle à utiliser pour créer une
nouvelle règle
if
(template.getName().equals("Template_Larg
eOrder"))
{

```

Pour une instance de modèle, une liste de paramètres doit être créée.

```
// Création d'une liste pour les paramètres
de cette instance de règle
// modèle
List<ParameterValue> paramList =
new ArrayList<ParameterValue>();

// A partir de la définition de modèle,
extraction d'un paramètre spécifique
// et définition d'une valeur
Parameter param =
template.getParameter("param0");
ParameterValue paramValue = param
.createParameterValue("
20");

// Ajout d'un paramètre à la liste
paramList.add(paramValue);

// Extraction du paramètre suivant et définition
de la valeur
param = template.getParameter("param1");
paramValue =
param.createParameterValue("7500");

// Ajout d'un paramètre à la liste
paramList.add(paramValue);

// Extraction du paramètre suivant et définition
de la valeur
param =
template.getParameter("param2");
paramValue = param
.createParameterValue("
Responsable de niveau 2");

// Ajout d'un paramètre à la liste
paramList.add(paramValue);
```

A partir des paramètres créés, l'instance de modèle peut être créée.

```
// Création de l'instance de règle
modèle avec la liste de
// paramètres
RuleSetTemplateInstanceRule
templateInstance = template
.createRuleFromTemplate
("ExtraLargeOrder",
paramList);
// Extraction du bloc de règles correspondant au jeu
de règles
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();
```

Une fois l'instance de modèle créée, elle peut être ajoutée au bloc de règles. Elle peut ensuite être organisée parmi les autres instances de règle modèle.

```
// Ajout de la règle de modèle au
bloc de règle
ruleBlock.addRule(templateInstance)
;

break;
}
}

// Utilisation de la liste d'origine ou création d'une nouvelle liste
```

```

// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajout du groupe de règles métier modifié à la
liste
publishList.add(brg);

// Publication de la liste avec le groupe de règles métier
mis à jour
BusinessRuleManager.publish(publishList, true);

out.println("");

// Extraction des groupes de règles métier pour
s'assurer que
// les modifications ont été publiées
out.printlnBold("Jeu de règles après publication:");

brgList = BusinessRuleManager
.getBRGsByTNSAndName(
"http://BRSamples/com/ibm/websphere
/sample/brules",
QueryOperator.EQUAL,
"ApprovalValues",
QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
op = brg.getOperation("getApprover");
ruleList = op.getBusinessRulesByName(
"getApprover", QueryOperator.EQUAL,
0, 0);

ruleSet = (RuleSet) ruleList.get(0);
out.print(RuleArtifactUtility.printRuleSet(rule
Set));
}
}
} catch (BusinessRuleManagementException e)
{
e.printStackTrace();
out.println(e.getMessage());
}
return out.toString();
}
}

```

Exemple

Sortie du navigateur Web pour l'exemple 11.

Exécution de l'exemple 11

Jeu de règles avant publication :

Jeu de règles

Nom : getApprover

Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Règle : LargeOrderApprover

Nom affiché : LargeOrderApprover

Description : null

Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 10 et que la commande excède 5 000 \$, l'approbation du superviseur est nécessaire

Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que la commande excède {1} \$, l'approbation du {2} est nécessaire

Nom du paramètre : param0

Valeur du paramètre : 10

Nom du paramètre : param1

Valeur du paramètre : 5000
Nom du paramètre : param2
Valeur du paramètre : superviseur
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom du paramètre : param0
Valeur du paramètre : peer

Jeu de règles après publication :

Jeu de règles

Nom : getApprover
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : LargeOrderApprover
Nom affiché : LargeOrderApprover
Description : null
Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 10 et que la commande excède 5 000 \$, l'approbation du superviseur est nécessaire
Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que la commande excède {1} \$, l'approbation du {2} est nécessaire
Nom du paramètre : param0
Valeur du paramètre : 10
Nom du paramètre : param1
Valeur du paramètre : 5000
Nom du paramètre : param2
Valeur du paramètre : superviseur
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom du paramètre : param0
Valeur du paramètre : peer
Règle : ExtraLargeOrder
Nom affiché :
Description : null
Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 20 et que la commande excède 7 500 \$, l'approbation du responsable de niveau 2 est nécessaire
Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que la commande excède {1} \$, l'approbation du {2} est nécessaire
Nom du paramètre : param0
Valeur du paramètre : 20
Nom du paramètre : param1
Valeur du paramètre : 7500
Nom du paramètre : param2
Valeur du paramètre : responsable de niveau 2

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 12 : Modifier et publier un modèle d'une table de décision en changeant la valeur d'un paramètre

Dans cet exemple, une condition et une action (toutes deux définies avec des modèles) sont modifiées dans une table de décision, en changeant les valeurs des paramètres avant publication.

La méthode la plus simple pour modifier des conditions et des actions dans une table de décision consiste à utiliser des noms uniques pour les modèles à chaque niveau de condition et pour chaque action. Cela permet d'effectuer des recherches sur les noms uniques, puis d'apporter des modifications aux instances de modèle définies à partir de ce modèle. Lorsque des modifications sont apportées à une instance d'un modèle particulier, toutes les valeurs de condition définies avec ce modèle à ce niveau seront mises à jour. Pour les expressions d'action, chaque instance est unique et les modifications apportées à une instance n'affectent pas les autres instances.

Pour cet exemple, un certain nombre de méthodes supplémentaires ont été créées pour simplifier la localisation d'un cas spécifique pour mise à jour, la recherche de la valeur de paramètre spécifique, et la recherche de l'expression d'action définie avec un modèle spécifique.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example12 {
    static Formatter out = new Formatter();

    static public String executeExample12()
    {
        try
        {
            out.clear();
            // Extraction d'un groupe de règles métier par nom et espace de nom
            // cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere
                    /sample/brules",
                    QueryOperator.EQUAL,
                    "ConfigurationValues",
                    QueryOperator.EQUAL, 0, 0);

            if (brgList.size() > 0)
            {
                // Extraction du premier groupe de règles métier de la liste
                // Ce doit être le seul groupe de règles métier de la
                // liste car
                // les combinaisons nom/espace de nom sont
```

```

uniques
BusinessRuleGroup brg = brgList.get(0);

// Extraction de l'opération du groupe de règles métier qui
// la règle métier qui sera modifiée lorsque les
// règles métier seront associées à une opération
// spécifique
Operation op = brg.getOperation("getMessages");

// Extraction de toutes les règles métier disponibles pour cette
opération
List<BusinessRule> ruleList =
op.getAvailableTargets();

// Pour cette opération, il n'existe qu'une seule règle métier
et
// il s'agit de celle que nous souhaitons mettre à jour
DecisionTable decisionTable = (DecisionTable)
ruleList.get(0);
out.println("");
out.printlnBold("Table de décision avant publication:");
out
.print(RuleArtifactUtility
.printDecisionTable(decisionT
able));

```

La règle, les conditions et les actions sont contenues dans une arborescence. Il est possible d'extraire le noeud racine de l'arborescence.

```

// Extraction de l'arborescence contenant toutes les
conditions
// et les actions pour la table de décision
TreeBlock treeBlock = decisionTable.getTreeBlock();
// Dans l'arborescence, extraction du noeud qui
constitue
// le point de départ pour la navigation dans la table de
décision
TreeNode treeNode = treeBlock.getRootNode();

```

La condition à mettre à jour a été définie à partir d'un modèle appelé "Condition Value Template 2.1". La méthode `getCaseEdge` permet d'effectuer des recherches récursives à partir du noeud jusqu'au niveau cas, afin de localiser le modèle. Cette méthode suppose que le niveau auquel le modèle est défini soit connu, ainsi que le niveau actuel. Elle peut être utilisée pour rechercher le cas associé à un modèle donné, au cas où un même nom soit utilisé pour différents cas.

```

// Extraction du cas au niveau 1 sous la racine, associé
// à un modèle spécifique avec une valeur de paramètre portant un nom
// spécifique. Etant donné que nous partons d'en haut,
// la profondeur actuelle est 0
CaseEdge caseEdge = getCaseEdge(treeNode, "param0",
"Condition Value Template 2.1", 1, 0);

```

A partir du cas trouvé, il est possible d'extraire l'objet `ConditionValueTemplateInstance` pour la condition.

```

if (caseEdge != null)
{
// Cas localisé. Extraction de la
définition de valeur
// du cas
TreeConditionValueDefinition condition =
caseEdge
.getValueDefinition();
// Extraction de l'expression de condition définie à l'aide d'un
correspondant

```

```

TemplateInstanceExpression conditionExpression
= condition
.getConditionValueTemplateInstance(
);

```

Avec l'objet `ConditionValueTemplateInstance`, la valeur de paramètre appropriée peut être extraite, puis mise à jour à l'aide de la méthode `getParameterValue`.

```

// Extraction du modèle pour l'expression
Template conditionTemplate =
conditionExpression
.getTemplate();

// Vérification du modèle car il est possible
d'avoir
// plusieurs modèles pour une valeur de condition,
mais un seul peut être
// appliqué
if (conditionTemplate.getName().equals(
"Condition Value Template 2.1"))
{
// Extraction de la valeur de paramètre
ParameterValue parameterValue =
getParameterValue("param0",
conditionExpression);

// Définition de la nouvelle valeur de paramètre
parameterValue.setValue("info");
}

```

Il est alors possible d'extraire les différentes expressions d'action définies à l'aide de modèles, afin de les mettre à jour. La méthode `getActionExpressions` renvoie toutes les actions définies avec le modèle `Action Value Template 1`.

```

ConditionNode conditionNode = (ConditionNode)
treeNode;

// Extraction de l'arborescence de cas
List<CaseEdge> caseEdges =
conditionNode.getCaseEdges();

// Création d'une liste contenant toutes les expressions
d'action qui devront
// également être mises à jour. Etant donné que chaque
action est
// indépendante des autres actions même si elles partagent
le même modèle,
// toutes les actions doivent être mises à jour.
List<TemplateInstanceExpression> expressions =
new Vector<TemplateInstanceExpression>();

// Extraction de toutes les expressions
pour (CaseEdge edge : caseEdges)
{
getActionExpressions("Action Value
Template 1", edge,
expressions);
}

```

Avec la liste des expressions d'action, chaque élément peut être mis à jour. Pour les expressions d'action définies à partir de modèles, la valeur de paramètre appropriée peut être mise à jour.

```

// Mise à jour du paramètre approprié dans chaque
expression
pour (TemplateInstanceExpression expression
expressions)

```

```

    {
        for (ParameterValue parameterValue :
            expression
                .getParameterValues())
        {
            // Vérification du paramètre
            bien qu'il n'y ait
            // qu'un seul paramètre dans
            notre modèle
            if
            (parameterValue.getParameter().getName().equals("param0")) {
                String value =
                    parameterValue.getValue();
                parameterValue.setValue("Info
                    "
                    +
                    value.substring(value.
                        indexOf(":"),
                        value.length()));
            }
        }
    }
    // Une fois la valeur de condition et les actions
    mises à jour, le
    // groupe de règles métier peut être publié.
    // Utilisez la liste d'origine ou créez une liste de
    // groupes de règles métier
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Ajout du groupe de règles métier modifié à la
    liste
    publishList.add(brg);

    // Publication de la liste avec le groupe de règles métier
    mis à jour
    BusinessRuleManager.publish(publishList, true);

    out.println("");

    // Extraction des groupes de règles métier pour
    s'assurer que
    // les modifications ont été publiées
    out.printlnBold("Table de décision après
    publication:");

    brgList =
    BusinessRuleManager.getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "ConfigurationValues",
        QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    op = brg.getOperation("getMessages");
    ruleList = op.getAvailableTargets();

    decisionTable = (DecisionTable)
    ruleList.get(0);
    out.print(RuleArtifactUtility
        .printDecisionTable(decisionTable))
    ;
    }
} catch (BusinessRuleManagementException e)

```



```

    {
        e.printStackTrace();
        out.println(e.getMessage());
    }
    return out.toString();
}

/*
Méthode permettant de naviguer de façon récursive dans une table de décision et de
localiser un cas associé à un modèle portant un nom spécifique et contenant un
paramètre spécifique à modifier. Cette méthode suppose que le niveau (depth) auquel
se trouve la valeur à modifier dans la table de décision est connu, et que
le niveau actuel (currentDepth) est connu aussi *
*/
static private CaseEdge getCaseEdge(TreeNode node, String pName,
    String templateName, int depth, int currentDepth)
{
    // Vérification de l'activité du noeud actuel. Ceci indique que
    // cette branche de la table de décision a été entièrement analysée dans le cadre
    // de la recherche de cas
    if (node instanceof ActionNode)
    {
        return null;
    }

    // Extraction des cas pour ce noeud
    List<CaseEdge> caseEdges = ((ConditionNode) node).getCaseEdges();
    for (CaseEdge caseEdge : caseEdges)
    {

        // Vérification afin de savoir si le niveau approprié a été atteint
        if (currentDepth < depth)
        {
            // Descente d'un niveau et appel de getCaseEdge
            pour
            // traiter ce niveau
            currentDepth++;
            return getCaseEdge(caseEdge.getChildNode(), pName,
                templateName, depth, currentDepth);
        } else
        {
            // Le niveau approprié a été atteint. Extraction
            de la condition pour
            // vérifier si les modèles de cette condition
            correspondent
            // au modèle recherché
            TreeConditionValueDefinition condition = caseEdge
                .getValueDefinition();

            // Extraction de l'expression pour la condition qui a
            été définie
            // avec un modèle
            TemplateInstanceExpression expression = condition
                .getConditionValueTemplateInstance();
            // Extraction du modèle dans l'expression
            Template template = expression.getTemplate();

            // Vérification afin de déterminer si le modèle trouvé est bien celui recherché
            if (template.getName().equals(templateName))
            {
                // Le modèle trouvé est bien celui recherché
                return caseEdge;
            } else
            {
                caseEdge = null;
            }
        }
    }
    return null;
}

```

```

}

/*
Cette méthode permet de rechercher une expression dans les différentes valeurs de
paramètre et si cette expression est trouvée, de renvoyer la valeur de paramètre
concernée.
*/
private static ParameterValue getParameterValue(String pName,
    TemplateInstanceExpression expression)
{
    // Vérification pour s'assurer que l'expression n'est pas nulle, car une valeur
    // nulle indiquerait que l'expression qui a été transmise n'a probablement pas
    // été définie avec un modèle et qu'il n'y a donc aucun paramètre à // vérifier.
    if (expression != null) {
        // Extraction des valeurs de paramètre pour l'expression
        List<ParameterValue> parameterValues = expression
            .getParameterValues();

        for (ParameterValue parameterValue : parameterValues)
        {
            // Vérification pour s'assurer que les différents paramètres
            // correspondent à la valeur
            // de paramètre recherchée

            if
                (parameterValue.getParameter().getName().equals(pName
                ))
            {
                // Retour de la valeur de paramètre appropriée
                return parameterValue;
            }
        }
    }
    return null;
}
/*
Cette méthode permet de trouver toutes les expressions d'action définies
avec un modèle spécifique. Elle fonctionne de manière récursive
et ajoute les expressions d'action qui correspondent au
paramètre d'expression.
*/

private static void getActionExpressions(String templateName,
    CaseEdge next, List<TemplateInstanceExpression>
    expressions)
{
    ActionNode actionNode = null;
    TreeNode treeNode = next.getChildNode();

    // Vérification de l'activité du noeud actuel.
    if (treeNode instanceof ConditionNode)
    {
        List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
            .getCaseEdges();

        Iterator<CaseEdge> caseEdgesIterator =
            caseEdges.iterator();

        // Analyse de tous les cas pour trouver les expressions
        // d'action
        while (caseEdgesIterator.hasNext())
        {
            getActionExpressions(templateName,
                caseEdgesIterator.next(),
                expressions);
        }
    } else {

```


Lorsque vous ajoutez une valeur de condition à un noeud de condition, vous ajoutez un cas. Ce nouveau cas est ajouté à la fin de la liste de cas. Pour la valeur de condition, vous devez spécifier une expression d'instance de modèle qui présente les valeurs de paramètre appropriées. Pour spécifier l'expression d'instance de modèle, vous devez utiliser un modèle spécifique. Il est recommandé de choisir des noms uniques pour les modèles à chaque niveau de noeud de condition, afin de pouvoir retrouver les modèles appropriés pour chaque type de condition. Si une définition de modèle unique est utilisée, il peut s'avérer difficile de déterminer le niveau auquel la condition est ajoutée.

Lorsque vous définissez une valeur de condition pour un noeud de condition, vous ajoutez une valeur de condition avec la même instance de modèle pour tous les noeuds de condition de même niveau. Cela est effectué dans le cadre de l'équilibrage de la table de décision. Lorsqu'une valeur de condition est ajoutée, de nouveaux noeuds d'action sont également ajoutés. Ces noeuds d'action comportent trois actions, qui ont des valeurs null pour la présentation utilisateur et pour l'expression d'instance de modèle. Etant donné que la valeur de condition peut être ajoutée à un noeud de condition qui n'a pas de noeud d'action en tant que noeud enfant, l'ajout d'un noeud de condition peut entraîner la création d'un grand nombre de noeuds d'action. Le nombre de noeuds d'action est basé sur le niveau auquel le noeud de condition est ajouté, et sur le nombre de noeuds de condition à ce niveau ainsi que sur le niveau et le nombre de noeuds de condition au niveau enfant.

Pour localiser les noeuds d'action qui ont été créés, vous pouvez effectuer une recherche sur les noeuds d'action avec des actions d'arborescence qui ont des valeurs null pour les présentations utilisateur et les expressions d'instance de modèle. La méthode `TreeActionValueTemplate` peut être utilisée pour créer une expression qui peut être définie dans `TreeAction`. Cette opération doit être répétée pour tous les nouveaux noeuds d'action.

Dans cet exemple, deux méthodes sont fournies pour définir les nouvelles actions d'arborescence. La méthode `getEmptyActionNode` permet de rechercher de façon récursive un noeud d'action vide à partir du noeud de condition en cours et la méthode `getParameterValue` permet de renvoyer la valeur d'un paramètre qui a été spécifié par son nom.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
```

```

import com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example13
{
    static Formatter out = new Formatter();

    static public String executeExample13()
    {
        try
        {
            out.clear();

            // Extraction d'un groupe de règles métier par nom et espace de nom
            // cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere/sample/brules",
                    QueryOperator.EQUAL, "ConfigurationValues",
                    QueryOperator.EQUAL, 0, 0);

            if (brgList.size() > 0)
            {
                // Extraction du premier groupe de règles métier de la
                // liste. Ce doit être le seul groupe de règles métier
                // de la liste car les combinaisons nom/espace
                // de nom sont uniques
                BusinessRuleGroup brg = brgList.get(0);

                // Extraction de l'opération du groupe de règles métier
                // qui comporte la règle métier qui sera
                // modifiée lors de l'association des règles métier
                // avec une opération spécifique
                Operation op = brg.getOperation("getMessages");

                // Extraction de toutes les règles métier disponibles pour
                // cette opération
                List<BusinessRule> ruleList =
                    op.getAvailableTargets();

                // Pour cette opération, il n'existe qu'une seule règle
                // métier et il s'agit de celle que nous souhaitons
                // mettre à jour

                DecisionTable decisionTable = (DecisionTable)
                    ruleList.get(0);
                out.printlnBold("Table de décision avant
                    publication:");
                out.print(RuleArtifactUtility
                    .printDecisionTable(decisionTable));
            }
        }
    }
}

```

Vous devez localiser le niveau auquel la valeur de condition va être ajoutée. Cette information est généralement transmise en tant que paramètre, afin que l'interface utilisateur ou l'application qui utilise les classes sache où ajouter la condition.

```

// Extraction du bloc d'arborescence contenant toutes les
// conditions et les actions pour la table de
// décision
TreeBlock treeBlock =
    decisionTable.getTreeBlock();

// Dans le bloc d'arborescence, extraction du noeud qui
// constitue le point de départ pour la navigation dans
// la table de décision
ConditionNode conditionNode = (ConditionNode)

```

```

treeBlock.getRootNode();

// Extraction des cas pour ce noeud, qui est
// le premier niveau de conditions
List<CaseEdge> caseEdges =
    conditionNode.getCaseEdges();

// Extraction du cas auquel la nouvelle condition
// sera ajoutée
CaseEdge caseEdge = caseEdges.get(0);

// Pour le cas, extraction du noeud de condition afin
// d'extraire les modèles pour la
// condition
conditionNode = (ConditionNode)
    caseEdge.getChildNode();

// Extraction des modèles pour la condition
List<TreeConditionValueTemplate>
treeValueConditionTemplates = conditionNode
    .getAvailableValueTemplates();

Iterator<TreeConditionValueTemplate>
treeValueConditionTemplateIterator =
    treeValueConditionTemplates.iterator();

TreeConditionValueTemplate conditionTemplate =
    null;

```

En utilisant des noms de modèle uniques pour chaque niveau de noeud de condition dans la table de décision, vous pouvez vous assurer que la valeur de condition est ajoutée au noeud de condition approprié.

```

// Recherche du modèle à utiliser
while
(treeValueConditionTemplateIterator.hasNext())
{
    conditionTemplate =
        treeValueConditionTemplateIterator
            .next();
    if (conditionTemplate.getName().equals(
        "Condition Value Template
        2.1"))
    {
        // Modèle trouvé
        break;
    }
    conditionTemplate = null;
}
if (conditionTemplate != null)
{

```

Une fois que vous avez trouvé le modèle approprié, une instance peut être créée et la valeur de paramètre appropriée peut être définie avant l'ajout au noeud de condition.

```

// Extraction de la définition de paramètre à partir
// du modèle
Parameter conditionParameter =
    conditionTemplate.getParameter("param0");

// Création d'une instance de valeur de paramètre à
// utiliser dans une nouvelle instance de modèle
// de condition
ParameterValue conditionParameterValue =
    conditionParameter
        .createParameterValue("fatal");

```

```

List<ParameterValue>
    conditionParameterValues = new
        ArrayList<ParameterValue>();

// Ajout de la valeur de paramètre à une liste

conditionParameterValues
    .add(conditionParameterValue);

// Création d'une instance de modèle de condition
// avec cette valeur de paramètre
TemplateInstanceExpression
newConditionValue =
    conditionTemplate
        .createTemplateInstanceExpression(c
            onditionParameterValues);
// Ajout de l'instance de modèle de condition à
// ce noeud de condition
conditionNode

.addConditionValueToThisLevel(newConditionValue);
// Lorsqu'un noeud de condition est ajouté,
// de nouveaux noeuds d'action vides sont
// créés. Il faut ensuite leur ajouter des
// instances de modèle d'action. En exécutant
// une recherche sur les noeuds d'action vides
// à partir du niveau parent, vous pouvez localiser
// tous les nouveaux noeuds d'action vides.
conditionNode = (ConditionNode)
    conditionNode.getParentNode();

```

Une fois la valeur de condition ajoutée au noeud de condition, les actions d'arborescence dans les nouveaux noeuds d'action doivent être définies via la méthode `TreeActionValueTemplate`. Tous d'abord, vous devez localiser le noeud d'action vide pour chaque cas. Utilisez le noeud de condition parent pour vous assurer que, lors des itérations dans les différents noeuds de condition, vous récupérerez tous les noeuds d'action.

```

// Extraction des cas pour le noeud parent
caseEdges = conditionNode.getCaseEdges();

Iterator<CaseEdge> caseEdgesIterator =
    caseEdges.iterator();

while (caseEdgesIterator.hasNext())
{
    // Pour chaque cas, extraction d'un
    // noeud d'action vide s'il en existe un
    ActionNode actionNode =
        getEmptyActionNode(caseEdgesIterator
            .next());

    // Vérification pour s'assurer que toutes les actions sont remplies
    if (actionNode != null)
    {

```

Lorsqu'un noeud d'action avec des actions d'arborescence vides est localisé, l'action d'arborescence doit être définie via la méthode `TreeActionValueTemplate`. Tout d'abord, localisez le modèle, puis spécifiez les paramètres avant de créer une instance de modèle. Une fois l'instance de modèle créée, l'action d'arborescence peut être mise à jour. Pour cet exemple, le paramètre a été défini sur une valeur issue d'une autre action d'arborescence d'un autre noeud d'action, sous le même noeud de condition. Pour les autres tables de décision pour lesquelles une autre action d'arborescence n'aura peut-être pas une valeur susceptible d'être utilisée

pour créer les nouvelles valeurs de paramètre, cette valeur devra être transmise en tant que paramètre à partir de l'application.

```

// Extraction de la liste
// d'actions d'arborescence. Il ne
// s'agit pas des actions
// elles-mêmes, mais
// des marques de
// réservation
List<TreeAction>
treeActionList = actionNode
    .getTreeActions();

List<TreeActionTermDefinition>
treeActionTermDefinitions =
    treeBlock
    .getTreeActionTermDefinitions();

List<TreeActionValueTemplate>
treeActionValueTemplates =
    treeActionTermDefinitions
    .get(0).getValueTemplates();

TreeActionValueTemplate
actionTemplate = null;

for (TreeActionValueTemplate
tempActionTemplate :
    treeActionValueTemplates)
{

    if
    (tempActionTemplate.get
    Name().equals(
    "Action Value
    Template 1"))
    {
        actionTemplate =
        tempActionTemplate;
        break;
    }
}

if (actionTemplate != null)
{
    // Extraction d'une autre action
    // qui se trouve sous
    // le noeud de condition
    // parent afin
    // d'utiliser la valeur comme
    // base pour le
    // message d'erreur dans
    // le nouveau
    // noeud d'action. Remontez
    // d'abord jusqu'au
    // noeud de condition
    // parent
    ConditionNode
    parentNode =
    (ConditionNode)
    actionNode
    .getParentNode();

    // Extraction du premier
    // cas du noeud
    // parent, car cette
    // action sera

```



```

// remplie au fur et à mesure
// de l'ajout de nouvelles
// actions à la fin
// de la liste de
// cas.
CaseEdge caseE =
parentNode.getCas
eEdges().get(
0);

// Le noeud enfant est un
// noeud d'action
// et se trouve au même
// niveau que le nouveau
// noeud d'action.
ActionNode aNode =
(ActionNode) caseE
.getChildNode();

// Extraction de la liste d'actions
// d'arborescence
TreeAction
existingTreeAction =
aNode
.getTreeActions()
.get(0);

// Extraction de l'expression
// d'instance
// de modèle pour
// l'action d'arborescence
// à partir de laquelle
// vous pouvez extraire le
// paramètre

TemplateInstanceExpression
existingExpression =
existingTreeAction
.getValueTemplateInstance();

ParameterValue
existingParameterValue =
getParameterValue(
"param0",
existingExpression);

String actionValue =
existingParameterValue
.getValue();

// Création du nouveau
// message à partir du
// message de
// l'action d'arborescence
// l'action d'arborescence
actionValue = "Fatal"
+
actionValue.substring(actionValue
.indexOf(":"), actionValue
.length());
Parameter
actionParameter =
actionTemplate
.getParameter("param0");

// Extraction du paramètre
// à partir du modèle

```

```

        ParameterValue
        actionParameterValue =
            actionParameter
            .createParameterValue(actionValue);

        // Ajout du paramètre à
        // une liste de modèles
        List<ParameterValue>
        actionParameterValues = new
        ArrayList<ParameterValue>();

        actionParameterValues.add(actionParameterValue);

        // Création d'une nouvelle
        // instance d'action d'arborescence

        TemplateInstanceExpression
        treeAction = actionTemplate
        .createTemplateInstanceExpression(actionParameterValues);

        // Définition de l'action d'arborescence
        // dans le noeud d'action
        // en la définissant dans la
        // liste d'actions d'arborescence

```

Ici, l'action d'arborescence dans le noeud d'action est mise à jour.

```

        treeActionList.get(0)
        .setValueTemplateInstance(
        treeAction);
    }
}
}
// Une fois la valeur de condition et les actions
// mises à jour, le groupe de règles métier peut être
// publié.
// Utilisez la liste d'origine ou créez une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajout du groupe de règles métier modifié à la
// liste
publishList.add(brg);

// Publication de la liste contenant le groupe de règles
// métier mis à jour

BusinessRuleManager.publish(publishList, true);

brgList =
BusinessRuleManager.getBRGsByTNSAndName(
"http://BRSamples/com/ibm/websphere/sample/brules",
QueryOperator.EQUAL, "ConfigurationValues",
QueryOperator.EQUAL, 0, 0);
brg = brgList.get(0);
op = brg.getOperation("getMessages");
ruleList = op.getAvailableTargets();
decisionTable = (DecisionTable)
ruleList.get(0);
out.printlnBold("Table de décision après
publication:");
out
.print(RuleArtifactUtility
.printDecisionTable(decisionTable));
}

```

```

    } catch (ValidationException e)
    {
        List<Problem> problems = e.getProblems();

        out.println("Incident = " +
            problems.get(0).getErrorType().name());

        e.printStackTrace();
        out.println(e.getMessage());
    } catch (BusinessRuleManagementException e)
    {
        e.printStackTrace();
        out.println(e.getMessage());
    }
    return out.toString();
}

/*
 * Cette méthode permet de rechercher le cas actuel pour tous
 * les noeuds d'action qui ont des actions d'arborescence vides. Pour trouver
 * un noeud d'action vide, vous devez examiner la fin de la liste
 * de cas et vérifier si le noeud d'action comporte des actions d'arborescence
 * qui ont des présentations utilisateur et des expressions
 * TemplateInstanceExpression nulles.
 */
private static ActionNode getEmptyActionNode(CaseEdge next)
{
    ActionNode actionNode = null;
    TreeNode treeNode = next.getChildNode();

    if (treeNode instanceof ConditionNode)
    {
        List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
            .getCaseEdges();

        if (caseEdges.size() > 1)
        {
            // Extraction du cas situé complètement à droite en tant que
            // nouvelle condition. Les actions vides se situent donc complètement à droite
            // des cas
            actionNode = getEmptyActionNode(caseEdges
                .get(caseEdges.size() - 1));

            if (actionNode != null)
            {
                return actionNode;
            }
        }
        else
        {
            actionNode = (ActionNode) treeNode;

            List<TreeAction> treeActions =
                actionNode.getTreeActions();

            if (!treeActions.isEmpty())
            {
                if
                ((treeActions.get(0).getValueUserPresentation() == null)
                    &&
                    (treeActions.get(0).getValueTemplateInstance() == null))
                {
                    return actionNode;
                }
            }
            actionNode = null;
        }
    }
}

```

```

    return actionNode;
}
/*
 * Cette méthode permet de rechercher une expression dans les différentes valeurs
 * de paramètre et si cette expression est trouvée, de renvoyer la valeur de
 * paramètre concernée.
 */
private static ParameterValue getParameterValue(String pName,
    TemplateInstanceExpression expression)
{
    ParameterValue parameterValue = null;

    // Vérification pour s'assurer que l'expression n'est pas nulle,
    // car une valeur nulle indiquerait
    // que l'expression qui a été transmise n'a probablement pas été
    // définie avec un modèle et qu'il n'y a donc aucun
    // paramètre à vérifier.
    if (expression != null)
    {
        // Extraction des valeurs de paramètre pour l'expression
        List<ParameterValue> parameterValues = expression
            .getParameterValues();
        Iterator<ParameterValue> parameterIterator =
            parameterValues
                .iterator();

        // Vérification pour s'assurer que les différents paramètres
        // correspondent à la valeur de paramètre recherchée
        while (parameterIterator.hasNext())
        {
            parameterValue = parameterIterator.next();

            if
                (parameterValue.getParameter().getName().equals(pName))
            {
                // Retour de la valeur de paramètre
                // appropriée
                return parameterValue;
            }
        }
    }
    return parameterValue;
}
}

```

Exemple

Sortie du navigateur Web pour l'exemple 13.

Exécution de l'exemple 13

Table de décision avant publication :

Table de décision

Nom : getMessages

Espace de nom : <http://BRSamples/com/ibm/websphere/sample/brules>

Table de décision après publication :

Table de décision

Nom : getMessages

Espace de nom : <http://BRSamples/com/ibm/websphere/sample/brules>

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 14 : Gestion des erreurs dans un jeu de règles

Cet exemple explique comment identifier des incidents dans un jeu de règles et déterminer la nature de l'incident, afin d'afficher le message approprié ou de mettre en oeuvre l'action nécessaire pour corriger la situation.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.problem.ValidationError;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example14 {
    static Formatter out = new Formatter();

    static public String executeExample14() {
        try {
            out.clear();

            // Extraction d'un groupe de règles métier par nom et espace de nom
            // cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere
                    /sample/brules",
                    QueryOperator.EQUAL,
                    "ApprovalValues",
                    QueryOperator.EQUAL, 0, 0);

            if (brgList.size() > 0) {
                // Extraction du premier groupe de règles métier de la liste
                // Cela doit être le seul groupe de règles métier de la
                // liste car
                // les combinaisons nom/espace de nom sont
                // uniques
                BusinessRuleGroup brg = brgList.get(0);
                out.println("Groupe de règles métier extrait");

                // Extraction de l'opération du groupe de règles métier qui
                // comporte la règle métier qui sera modifiée lorsque les
```

```

// règles métier seront associées à une opération
// spécifique
Operation op = brg.getOperation("getApprover");

// Extraction d'une règle spécifique par son nom
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
    "getApprover", QueryOperator.EQUAL, 0,
    0);

// Extraction de la règle spécifique
RuleSet ruleSet = (RuleSet) ruleList.get(0);
out.println("Jeu de règles extrait");

RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

// Recherche de la règle à
// modifier
while (ruleIterator.hasNext()) {
    RuleSetRule rule = ruleIterator.next();

    // Vérification pour s'assurer que la règle a été définie avec un
    // modèle
    // afin de permettre les modifications.
    if (rule instanceof
RuleSetTemplateInstanceRule) {
        // Extraction de l'instance de règle du modèle
        RuleSetTemplateInstanceRule
        templateInstance =
        (RuleSetTemplateInstanceRule) rule;
        // Vérification pour s'assurer qu'il s'agit de l'instance de règle de modèle
        // appropriée
        if (templateInstance.getName().equals(
            "LargeOrderApprover")) {

```

Pour provoquer un incident, cet exemple définit pour un paramètre une valeur qui n'est pas valide pour l'expression. En effet, le paramètre attend un entier, mais une chaîne est spécifiée.

```

// Extraction du paramètre de l'instance de
// modèle
ParameterValue parameter =
templateInstance
    .getParameterValue("par
    am1");

// Définition d'une valeur incorrecte pour ce
// paramètre
// Cela provoque une erreur de
// validation
parameter.setValue("3500 $");
out.println("Valeur incorrecte saisie
pour un paramètre");
break;
    }
}
// Il n'est pas possible d'accéder à ce code en raison
// de l'erreur
// introduite
// ci-dessus

// Une fois la valeur de condition et les actions mises à jour, le
// groupe de

```

```

// règles
// métier peut être publié.
// Utilisez la liste d'origine ou créez une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajout du groupe de règles métier modifié à la liste
publishList.add(brg);

// Publication de la liste avec le groupe de règles métier
mis à jour
BusinessRuleManager.publish(publishList, true);
}

```

Une erreur `ValidationException` est émise et à partir de cette exception, les incidents peuvent être extraits. Pour chaque incident, il est alors possible de déterminer la nature de l'erreur. Un message peut être imprimé ou une action appropriée peut être exécutée.

```

} catch (ValidationException e) {
out.println("Erreur de validation");

List<Problem> problems = e.getProblems();

Iterator<Problem> problemIterator = problems.iterator();

// Recherche de l'erreur concernée dans la liste des incidents et
// exécution de l'action appropriée (signaler l'erreur, corriger
// l'erreur, etc.)
while (problemIterator.hasNext()) {
Problem problem = problemIterator.next();
ValidationError error = problem.getErrorType();

// Identification de la valeur de l'erreur
if (error == ValidationError.TYPE_CONVERSION_ERROR) {
// Gestion de l'erreur en signalant
l'incident
.println("Incident : Valeur incorrecte
saisie pour un paramètre");
return out.toString();
}
// else if....
// Possibilité de rechercher d'autres erreurs et d'imprimer
// le message correspondant ou d'exécuter l'action
appropriée pour
// corriger la situation
}
} catch (BusinessRuleManagementException e) {
out.println("Erreur");
e.printStackTrace();
}
return out.toString();
}
}

```

Exemple

Sortie du navigateur Web pour l'exemple 14.

Exécution de l'exemple 14

```

Groupe de règles métier extrait
Jeu de règles extrait
Erreur de validation
Incident : Valeur incorrecte saisie pour un paramètre

```

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Exemple 15 : Gestion des erreurs dans un groupe de règles métier

Cet exemple est similaire à l'exemple 14, car il montre comment gérer les incidents qui peuvent se produire lors de la publication d'un groupe de règles métier. Il montre comment déterminer la nature de l'incident afin d'imprimer le message correspondant ou d'exécuter l'action appropriée.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example15
{
    static Formatter out = new Formatter();

    static public String executeExample15()
    {
        try
        {
            out.clear();

            // Extraction d'un groupe de règles métier par nom et espace de nom
            // cible
            List<BusinessRuleGroup> brgList = BusinessRuleManager
                .getBRGsByTNSAndName(
                    "http://BRSamples/com/ibm/websphere
                    /sample/brules",
                    QueryOperator.EQUAL,
                    "ApprovalValues",
                    QueryOperator.EQUAL, 0, 0);
            if (brgList.size() > 0)
            {
                // Extraction du premier groupe de règles métier de la liste
                // Cela doit être le seul groupe de règles métier de la
                // liste car
```



```

// la combinaison de nom et d'espace de nom cible est
unique
BusinessRuleGroup brg = brgList.get(0);
out.println("Groupe de règles métier extrait");

// Extraction de l'opération du groupe de règles métier qui comporte
// la règle métier qui sera modifiée lorsque les
// règles métier seront associées à une opération
// spécifique
Operation op = brg.getOperation("getApprover");

// Extraction d'une règle spécifique par son nom
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
    "getApprover", QueryOperator.EQUAL, 0,
    0);

// Extraction de la règle spécifique
RuleSet ruleSet = (RuleSet) ruleList.get(0);
out.println("Jeu de règles extrait");

RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
    ruleBlock.iterator();

// Recherche de la règle à
modifier
while (ruleIterator.hasNext())
{
    RuleSetRule rule = ruleIterator.next();

    // Vérification pour s'assurer que la règle a été définie avec un
modèle
// afin de permettre les modifications.
if (rule instanceof
RuleSetTemplateInstanceRule)
{
    // Extraction de l'instance de règle du modèle
RuleSetTemplateInstanceRule
templateInstance =
(RuleSetTemplateInstanceRule) rule;

    // Vérification pour s'assurer qu'il s'agit de l'instance de règle de modèle
appropriée
if (templateInstance.getName().equals(
    "LargeOrderApprover"))
{
    // Extraction du paramètre de l'instance de
modèle
ParameterValue parameter =
templateInstance
.getParameterValue("par
am1");

    // Définition de la valeur de ce paramètre
// Cette valeur est au format
approprié et ne
// provoquera pas d'erreur de validation
parameter.setValue("4000");
out.println("Valeur de paramètre de jeu de règle
définie correctement");
break;
}
}
}
}

```

Pour vérifier si un jeu de règles est correct, vous pouvez appeler la méthode `validate`. La méthode `validate` est disponible pour tous les objets et renvoie une liste d'incidents permettant d'identifier les erreurs. Lorsque vous appelez la méthode `validate` pour un objet, elle est également exécutée pour tous les sous-objets qu'il contient.

```
// Vérification des modifications apportées au jeu de règles
List<Problem> problems = ruleSet.validate();
out.println("Jeu de règles validé");

// Normalement, ce jeu d'essai ne contient aucune erreur,
// mais recherchez quand même les éventuels problèmes, puis
// prenez les mesures nécessaires pour corriger ou signaler
// l'erreur
if (problems != null)
{
    Iterator<Problem> problemIterator =
    problems.iterator();

    while (problemIterator.hasNext())
    {
        Problem problem = problemIterator.next();

        if (problem instanceof
        ProblemStartDateAfterEndDate)
        {
            out
            .println("Valeur
            incorrecte saisie pour un
            paramètre");
            return out.toString();
        }
    }
} else
{
    out.println("Aucun incident détecté pour le jeu de
    règles");
}
// Extraction de la liste des règles cible disponibles
List<BusinessRule> ruleList2 =
op.getAvailableTargets();

// Extraction de la première règle planifiée
l'élément comportant une
BusinessRule rule = ruleList2.get(0);

// Pour créer une condition d'erreur, nous allons définir l'heure de fin
d'une règle
// planifiée 1 heure avant l'heure de début
// Cela provoque une erreur de validation
Date future = new Date();
long futureTime = future.getTime() - 360000;

// Extraction de la liste de sélection d'opération pour ajouter
l'élément comportant une
// erreur de planification
OperationSelectionRecordList oplist = op
.getOperationSelectionRecordList();

// Création d'une nouvelle instance de règle planifiée
// Aucune erreur n'est renvoyée jusqu'à la validation ou la publication,
// car d'autres modifications peuvent être apportées
OperationSelectionRecord newRecord = oplist
.newOperationSelectionRecord(new Date(),
new Date(
futureTime), rule);
```

Lorsque l'enregistrement est ajouté avec des dates incorrectes, aucune erreur n'est renvoyée. Des chevauchements peuvent se produire ou aucun enregistrement de sélection n'est défini pour l'opération, tandis que des modifications sont en cours. L'erreur sera identifiée lors de la publication du groupe de règles métier comportant l'enregistrement de sélection d'opération en question. La méthode `validate` est appelée avant la publication des objets et des exceptions sont émises si des erreurs sont identifiées.

```

// Ajout de l'instance de règle planifiée à l'opération
// Aucune erreur identifiée
opList.addOperationSelectionRecord(newRecord);
out.println("Nouvel enregistrement de sélection ajouté avec
une planification incorrecte");

// Une fois la valeur de condition et les actions mises à jour, le
groupe
// de règles
// métier peut être publié.
// Utilisez la liste d'origine ou créez une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajout du groupe de règles métier modifié à la liste
publishList.add(brg);

// Publication de la liste avec le groupe de règles métier
mis à jour
BusinessRuleManager.publish(publishList, true);
}
} catch (ValidationException e) {
out.println("Erreur de validation");

List<Problem> problems = e.getProblems();

Iterator<Problem> problemIterator = problems.iterator();
// Il peut y avoir plusieurs incidents
// Passez en revue tous les incidents, et traitez chacun d'entre eux ou
// signalez l'incident
while (problemIterator.hasNext())
{
Problem problem = problemIterator.next();

// Chaque incident est de type différent et il est possible de les
comparer
if (problem instanceof ProblemStartDateAfterEndDate)
{
out
.println("La planification de la règle est
incorrecte. La date de début est postérieure à la date de
fin.");
return out.toString();
}
// else if...
// Possibilité de rechercher d'autres erreurs et d'imprimer
// le message correspondant ou d'exécuter l'action
appropriée pour
// corriger la situation
}
} catch (BusinessRuleManagementException e)
{
out.println("Erreur");
e.printStackTrace();
}

```

```
}
return out.toString();
}
}
```

Exemple

Sortie du navigateur Web pour l'exemple 15.

Exécution de l'exemple 15

Groupe de règles métier extrait
Jeu de règles extrait
Valeur de paramètre de jeu de règle définie correctement
Jeu de règles validé
Erreur de validation
Planification incorrecte de la règle.
La date de début est postérieure à la date de fin.

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Dans ces exemples, différentes propriétés et caractères génériques ('_', '%') sont utilisés avec différents opérateurs (AND, OR, LIKE, NOT_LIKE, EQUAL et NOT_EQUAL).

Exemple

Pour les besoins de ces exemples, les requêtes renverront différentes combinaisons de 4 groupes de règles métier. Il est important de bien comprendre les différents attributs et propriétés des groupes de règles métier, car ils sont utilisés dans les requêtes.

Nom : BRG1
Espace de nom cible : http://BRG1/com/ibm/br/rulegroup
Propriétés :
organization, 8JAA
department, claims
ID, 00000567
region: SouthCentralRegion
manager: Joe Bean

Nom : BRG2
Espace de nom cible : http://BRG2/com/ibm/br/rulegroup
Propriétés :
organization, 7GAA
department, accounting
ID, 0000047
ID_cert45, ABC
region: NorthRegion

Nom : BRG3
Espace de nom cible : http://BRG3/com/ibm/br/rulegroup

Propriétés :
organization, 7FAB
department, finance
ID, 0000053
ID_app45, DEF
region: NorthCentralRegion

Nom : BRG4
Espace de nom cible : <http://BRG4/com/ibm/br/rulegroup>
Propriétés :
organization, 7HAA
department, shipping
ID, 0000023
ID_app45, GHI
region: SouthRegion

Concepts associés

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Référence associée

Interrogation par une propriété unique

Ceci est un exemple d'interrogation par une propriété unique.

Interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur

Interrogation de groupes de règles métier par des propriétés et un caractère générique ('_')

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et un caractère générique ('_').

Interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')

Ceci est un exemple d'interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')

Interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_')

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_').

Interrogation de groupes de règles métier par l'opérateur NOT_EQUAL

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_EQUAL.

Interrogation de groupes de règles métier par PropertyIsDefined

Ceci est un exemple d'interrogation de groupes de règles métier par PropertyIsDefined.

Interrogation de groupes de règles métier par NOT PropertyIsDefined

Ceci est un exemple d'interrogation de groupes de règles métier par NOT PropertyIsDefined.

Interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec

des opérateurs AND et OR

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud combiné avec un opérateur AND

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur AND.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR.

Interrogation par une propriété unique :

Ceci est un exemple d'interrogation par une propriété unique.

```
List<BusinessRuleGroup> brgList = null;

brgList = BusinessRuleManager.getBRGsBySingleProperty(
    "department", QueryOperator.EQUAL,
    "accounting", 0, 0);
// Returns BRG2
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur :

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur

```
// Query Prop AND Prop
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
    "region", QueryOperator.LIKE,
    "%Region");

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode(
    "ID", QueryOperator.LIKE,
    "000005%");

QueryNode queryNode =
QueryNodeFactory.createAndNode(leftNode,
    rightNode);

brgList =
BusinessRuleManager.getBRGsByProperties(queryNode, 0, 0);
// Returns BRG1 and BRG3
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par des propriétés et un caractère générique ('_') :

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et un caractère générique ('_').


```
brgList = BusinessRuleManager.getBRGsBySingleProperty("ID",
QueryOperator.LIKE, "00000_3", 0, 0);
```

```
// Returns BRG3 and BRG4
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%') :

Ceci est un exemple d'interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("region",
QueryOperator.LIKE, "__uth%Region",
0, 0);
```

```
// Returns BRG1 and BRG4
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_') :

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_').

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
"7__A", 0, 0);
```

```
// Returns BRG1 and BRG3
```

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
"7%", 0, 0);
```

```
// Returns BRG1
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par l'opérateur NOT_EQUAL :

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_EQUAL.

```
brgList =
BusinessRuleManager.getBrgsBySingleProperty("department",
QueryOperator.NOT_EQUAL,
"claims", 0, 0);
// Returns BRG1
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par PropertyIsDefined :

Ceci est un exemple d'interrogation de groupes de règles métier par PropertyIsDefined.

```
PropertyIsDefinedQueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);
```

```
brgList = BusinessRuleManager.getBrgsByProperties(node, 0,
0);
```

```
// Returns BRG1
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par NOT PropertyIsDefined :

Ceci est un exemple d'interrogation de groupes de règles métier par NOT PropertyIsDefined.

```
// NOT Prop
QueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);
```

```
NotNode notNode = QueryNodeFactory.createNotNode(node);
```

```
brgList = BusinessRuleManager.getBrgsByProperties(notNode,
0, 0);
```

```
// Returns BRG1
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique.

```
// Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
```

```
    QueryOperator.EQUAL, "accounting");

    NotNode notNode =
    QueryNodeFactory.createNotNode(rightNode);

    QueryNode leftNode =
    QueryNodeFactory.createPropertyQueryNode("ID",
        QueryOperator.LIKE, "00000%");

    AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
        notNode);

    brgList = BusinessRuleManager.getBRGsByProperties (andNode,
    0, 0);

    // Returns BRG2
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND.

```
// NOT Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE, "cla%");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

AndNode andNode = QueryNodeFactory.createAndNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBrgsByProperties(andNode,
0, 0);

// Returns BRG1 and BRG2
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR.

```
// NOT Prop OR NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE, "acc%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
"department", QueryOperator.EQUAL,
"claims");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND.

```
// (Prop AND Prop) AND (Prop AND Prop)
QueryNode rightNode =
    QueryNodeFactory.createPropertyQueryNode("department",
        QueryOperator.LIKE, "acc%");

QueryNode leftNode =
    QueryNodeFactory.createPropertyQueryNode("organization",
        QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
    QueryNodeFactory.createAndNode(leftNode, rightNode);

QueryNode rightNode2 =
    QueryNodeFactory.createPropertyQueryNode("ID",
        QueryOperator.LIKE, "000004_");

QueryNode leftNode2 =
    QueryNodeFactory.createPropertyQueryNode("region",
        QueryOperator.EQUAL,
        "NorthRegion");

AndNode andNodeRight =
    QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNode =
    QueryNodeFactory.createAndNode(andNodeLeft, andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
    0, 0);

// Returns BRG2
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR.

```
// (Prop AND Prop) OR (Prop AND NOT Prop)
QueryNode rightNode =
    QueryNodeFactory.createPropertyQueryNode("department",
        QueryOperator.LIKE, "acc%");

QueryNode leftNode =
    QueryNodeFactory.createPropertyQueryNode("organization",
        QueryOperator.EQUAL, "7GAA");
```

```

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode,rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.EQUAL, "8JAA");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE, "%1Region");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, notNode);

OrNode orNode = QueryNodeFactory.createOrNode(andNodeLeft,
andNodeRight);

brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);

// Returns BRG2 and BRG3

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT.

```

// Prop AND NOT (Prop AND Prop)
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE, "000005%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.EQUAL,
"8JAA");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
"%1Region");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
notNode);

brgList = BusinessRuleManager.getBrgsByProperties(andNode,
0, 0);

// Returns BRG3

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR.

```
// NOT (Prop AND Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "8_A_");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
    "%1Region");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
rightNode);

brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);

// Returns BRG3
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```
// Prop AND (Prop AND (Prop AND Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7%");
```

```

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode, andNodeRight);

PropertyIsDefinedQueryNode node2 =
QueryNodeFactory.createPropertyIsDefinedQueryNode("ID_cert4
5");

AndNode andNode = QueryNodeFactory.createAndNode(node2,
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);
// Returns BRG2

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```

// (Prop AND (Prop AND Prop)) AND Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region", QueryOper
ator.LIKE,
"__thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.LIKE,
"7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE,
"%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode, andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_app45", QueryOp
erator.LIKE, "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

```



```
brgList = BusinessRuleManager.getBrgsByProperties (andNode,  
0, 0);
```

```
// Returns BRG4
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT.

```
// Prop AND (Prop AND (Prop AND NOT Prop))  
QueryNode rightNode =  
QueryNodeFactory.createPropertyQueryNode("organization",  
    QueryOperator.LIKE,  
    "7%");  
  
QueryNode rightNode2 =  
QueryNodeFactory.createPropertyQueryNode("region",  
    QueryOperator.LIKE,  
    "%1Region");  
  
NotNode notNode =  
QueryNodeFactory.createNotNode(rightNode2);  
  
QueryNode leftNode2 =  
QueryNodeFactory.createPropertyQueryNode("department",  
    QueryOperator.LIKE,  
    "%ing");  
  
AndNode andNodeRight =  
QueryNodeFactory.createAndNode(leftNode2, notNode);  
  
AndNode andNodeLeft =  
QueryNodeFactory.createAndNode(rightNode, andNodeRight);  
  
QueryNode leftNode =  
QueryNodeFactory.createPropertyQueryNode("ID_cert45",  
    QueryOperator.LIKE,  
    "AB_");  
  
AndNode andNode = QueryNodeFactory.createAndNode(leftNode,  
andNodeLeft);  
  
brgList = BusinessRuleManager.getBrgsByProperties (andNode,  
0, 0);  
  
// Returns BRG2
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```
// (Prop AND (Prop AND Prop)) AND Prop - Return empty
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "__thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode, andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
    QueryOperator.LIKE,
    "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBrgsByProperties(andNode,
0, 0);

//Returns no BRGs
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

```
// (Prop OR (Prop OR Prop)) OR Prop

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "__thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");
```

```

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2, rightNode2);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode, orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
    QueryOperator.LIKE,
    "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);

// Returns BRG1

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

```

// (Prop OR (Prop OR NOT Prop)) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "__thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2, notNode);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode, orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
    QueryOperator.LIKE,
    "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

```

```
brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);
```

```
// Returns BRG3
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

```
// Prop OR NOT(Prop OR Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "__thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
    "organization",
    QueryOperator.LIKE,
    "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
    "department",
    QueryOperator.LIKE,
    "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,
    rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft = QueryNodeFactory.createOrNode(leftNode,
notNode);

brgList =
BusinessRuleManager.getBrgsByProperties(orNodeLeft, 0, 0);

// Returns BRG3
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

```
// NOT(Prop OR Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
```

```

        QueryOperator.LIKE,
        "%1Region");

    QueryNode rightNode2 =
    QueryNodeFactory.createPropertyQueryNode(
        "organization",
        QueryOperator.LIKE,
        "7%");

    QueryNode leftNode =
    QueryNodeFactory.createPropertyQueryNode(
        "department",
        QueryOperator.LIKE,
        "%ing");

    OrNode orNodeRight =
    QueryNodeFactory.createOrNode(rightNode2, rightNode);

    NotNode notNode =
    QueryNodeFactory.createNotNode(orNodeRight);

    OrNode orNodeLeft =
    QueryNodeFactory.createOrNode(notNode, leftNode);

    brgList =
    BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

    // Returns BRG2 and BRG4

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND.

```

// AND list
List<QueryNode> list = new ArrayList<QueryNode>();

    QueryNode rightNode =
    QueryNodeFactory.createPropertyQueryNode("region",
        QueryOperator.LIKE,
        "%thRegion");

    list.add(rightNode);

    QueryNode rightNode2 =
    QueryNodeFactory.createPropertyQueryNode("organization",
        QueryOperator.LIKE,
        "7%");

    list.add(rightNode2);

    QueryNode leftNode =
    QueryNodeFactory.createPropertyQueryNode("department",
        QueryOperator.LIKE,
        "%ing");

    list.add(leftNode);

    QueryNode leftNode2 =

```

```

QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "7H%");

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
    0, 0);

// Returns BRG4

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud combiné avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur AND.

```

// AND list with a notNode
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "8%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
    0, 0);

// Return BRG4

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR.

```
// OR list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "8%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

list.add(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBrgsByProperties(orNode,
0, 0);

//Returns BRG3
```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR.

```
// OR list with Not node
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
    QueryOperator.LIKE,
    "%thRegion");

list.add(rightNode);
```

```

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "8%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
    QueryOperator.LIKE,
    "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
    QueryOperator.LIKE,
    "8%");

list.add(leftNode2);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4

```

Concepts associés

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Classes d'opérations communes

Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Concepts associés

Gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Référence associée

Classe `Formatter`

Cette classe fournit diverses méthodes permettant d'afficher les différents exemples. Elle ajoute diverses balises HTML pour formater la sortie.

Classe `RuleArtifactUtility`

Cette classe utilitaire comporte deux méthodes publiques. La première sert à imprimer une table de décision. Cette méthode exploite une méthode privée qui utilise la récursivité pour imprimer les conditions et les actions de la table de décision. La seconde méthode publique sert à imprimer un jeu de règles.

Classe `Formatter`

Cette classe fournit diverses méthodes permettant d'afficher les différents exemples. Elle ajoute diverses balises HTML pour formater la sortie.


```

package com.ibm.websphere.sample.brules.mgmt;

public class Formatter {

private StringBuffer buffer;

public Formatter()
{
    buffer = new StringBuffer();
}

public void println(Object o)
{
    buffer.append(o);
    buffer.append("<br>\n");
}

public void print(Object o)
{
    buffer.append(o);
}

public void printlnBold(Object o)
{
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b><br>\n");
}

public void printBold(Object o)
{
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b>");
}

public String toString()
{
    return buffer.toString();
}

public void clear()
{
    buffer = new StringBuffer();
}
}

```

Référence associée

Classes d'opérations communes

Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Classe RuleArtifactUtility

Cette classe utilitaire comporte deux méthodes publiques. La première sert à imprimer une table de décision. Cette méthode exploite une méthode privée qui utilise la récursivité pour imprimer les conditions et les actions de la table de décision. La seconde méthode publique sert à imprimer un jeu de règles.

```

package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;

```

```

import com.ibm.wbiserver.brules.mgmt.RuleTemplate;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableRule;
import
com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionTermDefinition;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class RuleArtifactUtility
{
    static Formatter out = new Formatter();

    /*
    Method to print out a decision table with the conditions and
    actions printed out in a HTML tabular format. The conditions
    and actions are printed out with a separate method that
    recursively works through the case edges of the decision
    tables.
    */

    public static String printDecisionTable(BusinessRule
ruleArtifact)
    {
        out.clear();
        out.printlnBold("Decision Table");
        DecisionTable decisionTable = (DecisionTable)
ruleArtifact;
        out.println("Name: " +
decisionTable.getName());
        out.println("Namespace: " +
decisionTable.getTargetNameSpace());

        // Output the init rule for the decision table
        before
        // working through the table of conditions and
        actions
        DecisionTableRule initRule =
decisionTable.getInitRule();
        if (initRule != null)
        {
            out.printBold("Init Rule: ");
            out.println(initRule.getName());
            out.println("Display Name: " +
initRule.getDisplayName());
            out.println("Description: " +
initRule.getDescription());
            // The expanded user presentation
            will automatically populate the
            // presentation with the parameter
            values and can be used for

```

```

// display if the init rule was
// defined with a template. If no
// template was defined the
// expanded user presentation
// is the same as the regular
// presentation.
out.println("Extended User
Presentation: "
+
initRule.getExpandedUse
rPresentation());
// The regular user presentation
// will have placeholders in the
// string where the
// parameter can be substituted if
// the init rule was defined with a
// template
// If the rule was not defined with
// a template, the user
// presentation will only
// be a string without
// placeholders. The placeholders are
// of a
// format of {n} where
// n is the index (zero-based) of
// the parameter in the template. This
// value
// can be used to create an
// interface for editing where there
// are
// fields with
// the parameter values available
// for editing
out.println("User Presentation: " +
initRule.getUserPresentation());
// Init rules might be defined with
// or without a template
// Check to make sure a template
// was used before trying
// to access the parameters
if (initRule instanceof
DecisionTableTemplateInstanceRule)
{
    DecisionTableTemplateIn
    stanceRule
    templateInstance =
    (DecisionTableTemplateI
    nstanceRule) initRule;

    RuleTemplate template =
    templateInstance.getRul
    eTemplate();

    List<Parameter>
    parameters =
    template.getParameters(
    );
    Iterator<Parameter>
    paramIterator =
    parameters.iterator();

    Parameter parameter =
    null;

    while
    (paramIterator.hasNext(
    )) {

```

```

        parameter =
        paramIterator.next();

        out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
+
templateInstance.getPar
ameterValue(parameter
.getName()));
    }
}
// For the rest of the decision table, start at
the root and
// recursively work through the different case
edges and
// actions
TreeBlock treeBlock =
decisionTable.getTreeBlock();
TreeNode treeNode = treeBlock.getRootNode();

printDecisionTableConditionsAndActions(treeNode
, 0);
out.println("");
return out.toString();
}
/*Method to recursively work through the case edges and print
out the conditions and actions.
*/
static private void printDecisionTableConditionsAndActions(
TreeNode treeNode, int indent)
{
out.print("<table border=\"1\">");
if (treeNode instanceof ConditionNode)
{
// Get the case edges for the
current TreeNode
// and for each case edge print out
the conditions
ConditionNode conditionNode =
(ConditionNode) treeNode;

List<CaseEdge> caseEdges =
conditionNode.getCaseEdges();
Iterator<CaseEdge> caseEdgeIterator
= caseEdges.iterator();

CaseEdge caseEdge = null;

while (caseEdgeIterator.hasNext())
{
out.print("<tr>");
// If this is the start
of the conditions for the
// condition node,
print out the condition term
if (indent == 0)
{
out.print("<td>");

TreeConditionTermDefinition
termDefinition =
conditionNode
.getTermDefinition();

```

```

out.print(termDefinition.getUserPresentation()
);
out.print("</td>");
indent++;
} else {
// After the condition
term has been printed
for a
// case edge skip for
the rest of the case
edges
out.print("<td></td>");
}

caseEdge =
caseEdgeIterator.next()
;

out.print("<td>");

// Check if the
caseEdge is defined by
a template
if
(caseEdge.getValueDefin
ition() != null)
{
TemplateInstanceExpress
ion templateInstance =
caseEdge
.getValueTemplateInstan
ce();

out.println(templateIns
tance.getExpandedUserPr
esentation());

TreeConditionValueDefin
ition valueDef =
caseEdge
.getValueDefinition();

out.println(valueDef.ge
tUserPresentation());

Template template =
templateInstance.getTem
plate();

// Get the parameters
for the template
definition and
// print out the
parameter names and
values
List<Parameter>
parameters =
template.getParameters(
);
Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =

```

```

templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
(paramIterator.hasNext(
) &&
paramValues.hasNext())
{
parameter =
paramIterator.next();
parameterValue =
paramValues.next();

out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
+
parameterValue.getValue
());
}
}

out.print("</td><td>");
// Print the child node
for the caseEdge
printDecisionTableCondi
tionsAndActions(caseEdg
e.getChildNode(),
0);

out.print("</td></tr>")
;
}

// Add Otherwise condition if it
exists
TreeNode otherwise =
conditionNode.getOtherwiseCase();

if (otherwise != null)
{
out.print("<tr><td></td>
<td>Otherwise</td><td>
");
// Print the Otherwise
ConditionNode
printDecisionTableCondi
tionsAndActions(otherwi
se, 0);
out.print("</td></td>")
;
}
out.print("</table>");
} else {
// ActionNode has been found and

```

```

different logic is needed
// to print out the TreeActions
ActionNode actionNode =
(ActionNode) treeNode;
List<TreeAction> treeActions =
actionNode.getTreeActions();

Iterator<TreeAction>
treeActionIterator =
treeActions.iterator();

TreeAction treeAction = null;

// The ActionNode can contain
multiple TreeActions to
// print out
while
(treeActionIterator.hasNext())
{
    out.print("<tr>");
    treeAction =
treeActionIterator.next
();

    TreeActionTermDefinitio
n treeActionTerm =
treeAction
.getTermDefinition();

    if (indent == 0) {
out.print("<td>");
out.print(treeActionTer
m.getUserPresentation()
);
out.print("</td>");
}
out.print("<td>");
TemplateInstanceExpress
ion templateInstance =
treeAction
.getValueTemplateInstan
ce();

// Check that a
template was specified
for
// the TreeAction
before working with the
// parameter name and
values
if (templateInstance !=
null) {
out.println(templateIns
tance.getExpandedUserPr
esentation());

    Template template =
templateInstance.getTem
plate();

    List<Parameter>
parameters =
template.getParameters(
);

    Iterator<Parameter>
paramIterator =

```

```

        parameters.iterator();

        List<ParameterValue>
        parameterValues =
        templateInstance
        .getParameterValues();
        Iterator<ParameterValue
        > paramValues =
        parameterValues
        .iterator();

        Parameter parameter =
        null;
        ParameterValue
        parameterValue = null;

        while
        (paramIterator.hasNext(
        ) &&
        paramValues.hasNext())
        {
        {parameter =
        paramIterator.next();
        parameterValue =
        paramValues.next();

        out.println(" Parameter
        Name: " +
        parameter.getName());
        out.println(" Parameter
        Value: "
        +
        parameterValue.getValue
        ());

        }
        } else
        {
        // If a template was
        not used, the only item
        that is
        // available is the
        UserPresentation if it
        was
        // specified when the
        rule was created
        out.print(treeAction.ge
        tValueUserPresentation(
        ));
        }

        out.print("</td></tr>")
        ;
    }
    out.print("</table>");
}
}
/*
 * Method to print out a rule set
 */
public static String printRuleSet(BusinessRule
ruleArtifact)
{
    out.clear();
    out.printlnBold("Rule Set");
    RuleSet ruleSet = (RuleSet) ruleArtifact;
    out.println("Name: " + ruleSet.getName());
}

```



```

out.println("Namespace: " +
ruleSet.getTargetNameSpace());

// The rules in a rule set are contained in a
rule block
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

RuleSetRule rule = null;

// Iterate through the rules in the rule block.
while (ruleIterator.hasNext())
{
    rule = ruleIterator.next();
    out.printBold("Rule: ");
    out.println(rule.getName());
    out.println("Display Name: " +
rule.getDisplayName());
    out.println("Description: " +
rule.getDescription());
    // The expanded user presentation
will automatically populate the
// presentation with the parameter
values and can be used for
// display if the rule was defined
with a template. If no
// template was defined the
expanded user presentation
// is the same as the regular
presentation.
    out.println("Expanded User
Presentation: "
+
rule.getExpandedUserPre
sentation());
    // The regular user presentation
will have placeholders in the
// string where the parameter can
be substituted if the rule
// was defined with a template. If
the rule was not defined with
// a template, the user
presentation will only be a string
// without placeholders. The
placeholders are of a format of {n}
// where n is the index (zerobased)
of the parameter in the
// template. This value can be used
to create an interface for
// editing where there are fields
with the parameter values
// available for editing
    out.println("User Presentation: " +
rule.getUserPresentation());

    // Check if the rule was defined
with a template
    if (rule instanceof
RuleSetTemplateInstanceRule) {
        RuleSetTemplateInstance
Rule templateInstance =
(RuleSetTemplateInstanc
eRule) rule;

```

```

RuleSetRuleTemplate
template =
templateInstance
.getRuleSetRuleTemplate
();

List<Parameter>
parameters =
template.getParameters(
);
Iterator<Parameter>
paramIterator =
parameters.iterator();

Parameter parameter =
null;

// Retrieve all of the
parameters and output
the name and value
while
(paramIterator.hasNext(
))
{
parameter =
paramIterator.next();

out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
+
templateInstance.getPar
ameterValue(
parameter.getName()).ge
tValue());
}
}
}
out.println("");
return out.toString();
}
}

```

Référence associée

Classes d'opérations communes

Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Chapitre 4. Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

A propos de cette tâche

Ces clients peuvent être des clients EJB (Enterprise JavaBeans™), des clients de service Web ou encore des clients Web exploitant les composants JSF (JavaServer Faces) de Business Process Choreographer Explorer. Ce dernier fournit des API EJB (Enterprise JavaBeans) et des interfaces pour les services Web pour vous permettre de développer ces clients. L'API EJB est accessible via n'importe quelle application Java, y compris une autre application EJB. Il est possible d'accéder aux interfaces des services Web à partir des environnements Java ou Microsoft® .Net.

Concepts associés

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches utilisateur. Chacune de ces interfaces présente des caractéristiques différentes.

Requêtes portant sur les données des processus métier et des tâches

Les données d'instance des processus métier et des tâches humaines à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche humaine grâce à une interface de requête.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Développement des pages JSP pour les messages de tâche et de processus Business Process Choreographer Explorer fournit des formulaires d'entrée et de sortie par défaut pour afficher et saisir les données métier. Vous pouvez utiliser des pages JSP pour créer des formulaires d'entrée et de sortie définis par l'utilisateur.

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches utilisateur. Chacune de ces interfaces présente des caractéristiques différentes.

L'interface de programmation que vous choisissez dépend de plusieurs facteurs, dont la fonctionnalité devant être fournie par votre application client, le fait que vous disposez ou non d'une infrastructure de client final existante, ou encore que vous souhaitiez ou non traiter les flux de tâches utilisateur. Pour faciliter la sélection de l'interface appropriée, le tableau suivant compare les caractéristiques des interfaces de programmation EJB, JMS, REST et de services Web.

	Interface EJB	Interface de service Web	Interface de message JMS	Interface REST
Fonctionnalité	Cette interface est disponible à la fois pour les processus métier et les tâches utilisateur. Utilisez cette interface pour créer des clients fonctionnant de manière générique avec des processus et des tâches.	Cette interface est disponible à la fois pour les processus métier et les tâches utilisateur. Utilisez cette interface pour créer des clients destinés à un ensemble connu de processus et de tâches.	Cette interface est disponible uniquement pour les processus métier. Utilisez cette interface pour créer des clients de messagerie destinés à un ensemble connu de processus.	Cette interface est disponible à la fois pour les processus métier et les tâches utilisateur. Utilisez cette interface pour créer des clients de type Web 2.0 conçus pour un ensemble connu de processus et de tâches.
Traitement des données	Prend en charge le chargement de schémas d'artefacts distants pour accéder aux métadonnées des objets métier. Si l'application client EJB est exécutée dans la même cellule que l'instance de WebSphere Process Server à laquelle elle est connectée, les schémas requis par les objets métier des processus et des tâches ne doivent pas nécessairement être disponibles au niveau du client et peuvent être chargés depuis le serveur via le chargeur d'artefacts distants RAL (Remote Artifact Loader). Le chargeur RAL peut également être appliqué à plusieurs cellules si l'application client s'exécute sur une installation serveur complète de WebSphere Process Server. toutefois, le chargeur RAL n'est pas utilisable dans une configuration inter-cellules dans laquelle l'application client s'exécute dans une installation client de WebSphere Process Server.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.
Environnement client	Une installation de WebSphere Process Server ou une installation client de WebSphere Process Server.	Tout environnement d'exécution prenant en charge les appels de services Web, y compris les environnements Microsoft .NET.	Tout environnement d'exécution prenant en charge les clients JMS, y compris les modules SCA utilisant des importations JMS SCA.	Tout environnement d'exécution prenant en charge les clients REST.

	Interface EJB	Interface de service Web	Interface de message JMS	Interface REST
Sécurité	Sécurité Java 2, Enterprise Edition (J2EE).	Sécurité des services Web.	Sécurité Java 2, Enterprise Edition (J2EE) pour l'installation WebSphere Process Server. Vous pouvez également sécuriser les files d'attente dans lesquelles l'application client JMS place les messages d'interface API, par exemple via les mécanismes de sécurité de WebSphere MQ.	Les applications client appelant les méthodes REST doivent utiliser un mécanisme d'authentification HTTP adapté.

Une opération peut être exposée par plusieurs protocoles. Tenez compte des remarques générales suivantes si vous utilisez la même opération dans différents protocoles.

- Dans les interfaces Services Web et REST, tous les identificateurs d'objet, tels que PIID, AIID et TKIID, sont représentés par un type chaîne (string). Seule l'interface API EJB attend un ID d'objet à type sécurisé.
- La surcharge d'opération est seulement utilisée pour les méthodes EJB et non pour les opérations WSDL. Dans certains cas, il existe plusieurs opérations WSDL distinctes ; dans d'autres cas, il n'en existe qu'une seule, qui autorise toutes les variantes de paramètres soit par omission (`minOccurs="0"`), soit par l'emploi de valeurs Null (`nillable="true"`).
- Dans certaines méthodes EJB, les espaces de noms XML et les noms locaux sont passés dans des paramètres distincts. La plupart des opérations WSDL utilisent le type XML Schema QName pour passer ces paramètres.
- Les interactions asynchrones avec les opérations de demande-réponse WSDL à exécution longue, telles que l'opération `callWithReplyContext` dans l'interface EJB ou l'opération `callAsync` dans l'interface WSDL, sont représentées par l'opération `call` dans l'interface JMS.
- L'interface EJB renvoie un ensemble d'objets d'API, qui exposent les méthodes d'accès `get` et `set` des champs contenus. Les interfaces Services Web et REST renvoient des documents de types complexes (XML ou JSON) au client.
- Certains services de Human Task Manager opérant sur des tâches humaines sont également disponibles comme services Business Flow Manager opérant sur des activités qui appellent une tâche humaine.

Tâches associées

Développement d'applications client pour les tâches et processus métier
Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Requêtes portant sur les données des processus métier et des tâches

Les données d'instance des processus métier et des tâches humaines à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche humaine grâce à une interface de requête.

Les interfaces EJB d'interrogation, l'API de requête et l'API de table de requêtes, sont disponibles avec Business Process Choreographer.

En fonction des clients qui accèdent aux données relatives aux processus ou aux tâches, vous pouvez utiliser une ou plusieurs de ces interfaces. Les API REST et Web Services (services Web) sont fournies dans Business Process Choreographer pour vous permettre d'interroger les données des listes de tâches et de processus. Cependant, pour des raisons de performances, si vous devez interroger des listes de processus et de tâches particulièrement volumineuses, utilisez l'API de table de requêtes EJB ou l'API de table de requêtes REST de Business Process Choreographer.

Concepts associés

Comparaison des interfaces de programmation destinées à l'extraction de données de processus et de tâche

Business Process Choreographer fournit deux interfaces de programmation (API) spécialisées dans l'extraction des données de processus et de table : l'API de table de requêtes et l'API de requête. Chacune de ces interfaces présente des caractéristiques différentes.

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

API de requête EJB de Business Process Choreographer

Les méthodes query ou queryAll du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Comparaison des interfaces de programmation destinées à l'extraction de données de processus et de tâche

Business Process Choreographer fournit deux interfaces de programmation (API) spécialisées dans l'extraction des données de processus et de table : l'API de table de requêtes et l'API de requête. Chacune de ces interfaces présente des caractéristiques différentes.

L'interface choisie dépendra de plusieurs facteurs, dont la fonctionnalité attendue de votre application cliente, la disponibilité ou non d'une infrastructure de client pour utilisateur final existante et, bien sûr, le niveau de performances visé. Pour vous aider à faire le bon choix, le tableau suivant compare les caractéristiques des API de table de requêtes et de requête.

Caractéristique	API de table de requêtes	API de requête
Disponibilité	L'API de table de requêtes est disponible pour l'interface EJB de Business Flow Manager et pour l'interface de programmation REST.	L'API de requête est disponible pour les interfaces de programmation EJB, services Web, JMS et REST.

Caractéristique	API de table de requêtes	API de requête
Méthodes dédiées à l'extraction de contenus	L'API fournit les méthodes suivantes : <ul style="list-style-type: none"> • queryEntities • queryEntityCount • queryRows • queryRowCount 	L'API fournit les méthodes suivantes : <ul style="list-style-type: none"> • query • queryAll
Méthodes dédiées à l'extraction de métadonnées	L'API fournit les méthodes suivantes : <ul style="list-style-type: none"> • getQueryTableMetaData • findQueryTableMetaData • queryProcessTemplates • queryTaskTemplates 	
Nom de la table de requêtes	Spécifie la table de requêtes sur laquelle l'API de table de requêtes est exécutée. Il n'est possible d'interroger qu'une seule table de requêtes à la fois. Par exemple, queryEntities("CUST.TASKS", ...).	La clause SELECT spécifie les colonnes et les vues de base de données prédéfinies sur lesquelles la requête est exécutée. Cette spécification est similaire à une clause SQL SELECT. Par exemple, query("TASK.TKIID, TASK.STATE, WORK_ITEM.REASON", ...).
Clause SELECT et attributs sélectionnés	Utilisez les options de filtrage de l'API de table de requête pour spécifier les attributs que la requête doit renvoyer dans ses résultats. Comme la requête s'exécute sur une seule table de requêtes, les attributs sont identifiables sans équivoque par leurs noms respectifs.	Utilisez la clause SELECT pour spécifier les attributs. Un nom d'attribut est spécifié avec une syntaxe de la forme <i>nom_vue.nom_attribut</i> . Par exemple, pour rechercher des états de tâche, spécifiez TASK.STATE dans votre requête.
Clause WHERE et filtres	Avec l'API de table de requêtes, utilisez la propriété queryCondition pour filtrer encore plus les résultats de vos requêtes. Les tables de requêtes fournissent un contenu préfiltré si, dans leur définition, des filtres de table de requête principale, des filtres d'autorisation ou des filtres de table de requêtes ont été spécifiés.	Utilisez la clause WHERE pour filtrer vos requêtes.
Clause WHERE et critères de sélection	La clause WHERE de l'API de requête n'est pas nécessaire sous cette forme dans l'API de table de requêtes. Avec l'API de table de requêtes, utilisez la propriété queryCondition pour appliquer un filtrage additionnel. Les critères de sélection spécifiés dans la définition de table de requêtes sélectionnent une propriété particulière de la table de requêtes attachée. Ce comportement est obtenu en plus du filtrage exercé par la clause WHERE dans l'API de requête.	Les critères de sélection ne sont pas disponibles pour l'API de requête. Cependant, ils sont similaires à la partie de la clause WHERE qui définit, par exemple, le nom (colonne NAME) ou l'environnement local (colonne LOCALE) de la table ou vue QUERY_PROPERTY, TASK_CPROP ou TASK_DESC. Par exemple, une clause WHERE telle que QUERY_PROPERTY.NAME='xyz' revient au même que de spécifier NAME='xyz' comme critère de sélection dans la définition de la table de requêtes attachée QUERY_PROPERTY.
Éléments de travail et autorisation	Utilisez la table de requêtes WORK_ITEM pour accéder aux éléments de travail. Vous pouvez personnaliser l'emploi des éléments de travail dans la définition d'une table de requêtes au moment où vous la développez, ainsi que dans l'API de table de requêtes, en utilisant l'objet AuthorizationOptions ou l'objet AdminAuthorizationOptions. Par exemple, pour exclure les éléments de travail 'everybody' lors de l'interrogation de la table de requêtes TASK, spécifiez WI.EVERYBODY=0 pour la propriété queryCondition, ou alors spécifiez setUseEverybody(Boolean.FALSE) sur l'objet AuthorizationOptions.	Utilisez la vue WORK_ITEM pour accéder aux éléments de travail. Les quatre types d'éléments de travail sont pris en compte pour les résultats d'une requête : 'everybody', 'individual', 'groups' et 'inherited'. Pour filtrer les éléments de travail d'après un type spécifique, personnalisez la clause WHERE. Par exemple, pour exclure les éléments de travail 'everybody', spécifiez WORK_ITEM.EVERYBODY=0 dans la clause WHERE.
Paramètres	Vous pouvez utiliser des paramètres dans les filtres et les critères de sélection pour les tables de requêtes composites.	Les paramètres ne sont pas disponibles pour l'API de requête, sauf si des requêtes stockées sont utilisées.

Caractéristique	API de table de requêtes	API de requête
Requêtes stockées et tables de requêtes	La différence entre une requête stockée et une table de requêtes est que la première est définie pour une requête particulière, tandis que la table de requêtes est définie pour un ensemble particulier de requêtes. Par exemple, la définition d'une table de requêtes n'autorise pas la spécification d'une clause ORDER BY, car les informations correspondantes ne sont généralement disponibles qu'au moment où la requête est exécutée.	Les requêtes stockées permettent d'interroger les données en utilisant des ensembles prédéfinis d'options.
Vues matérialisées	Les vues matérialisées ne sont pas disponibles pour l'API de table de requêtes.	Les vues matérialisées utilisent des technologies propres aux bases de données pour améliorer les performances des requêtes.
Tables personnalisées	Les tables de requêtes supplémentaires offrent la même fonctionnalité que les tables personnalisées.	Les tables personnalisées servent à inclure, dans les requêtes, des données qui sont externes au schéma de base de données de Business Process Choreographer.
queryAll et options d'autorisation	La fonctionnalité queryAll est fournie par l'objet AdminAuthorizationOptions, lequel peut être passé à l'API de table de requêtes à la place de l'objet AuthorizationOptions. L'appelant doit faire partie du rôle J2EE BPESystemAdministrator.	La méthode queryAll permet aux utilisateurs ayant le rôle J2EE BPESystemAdministrator d'obtenir tous les objets dans les résultats d'une requête sans être limités par les éléments de travail d'un utilisateur ou d'un groupe particulier.
Internationalisation	Lors de l'utilisation de tables de requêtes, il est possible de choisir la langue dans laquelle doivent être présentés les noms et les descriptions des tables en question et de leurs attributs.	Les noms des colonnes des vues sélectionnées sont affichés tels qu'ils apparaissent dans la base de données.

Concepts associés

Requêtes portant sur les données des processus métier et des tâches
 Les données d'instance des processus métier et des tâches humaines à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche humaine grâce à une interface de requête.

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Il existe trois types de tables de requêtes : les tables prédéfinies, les tables supplémentaires et les tables composites.

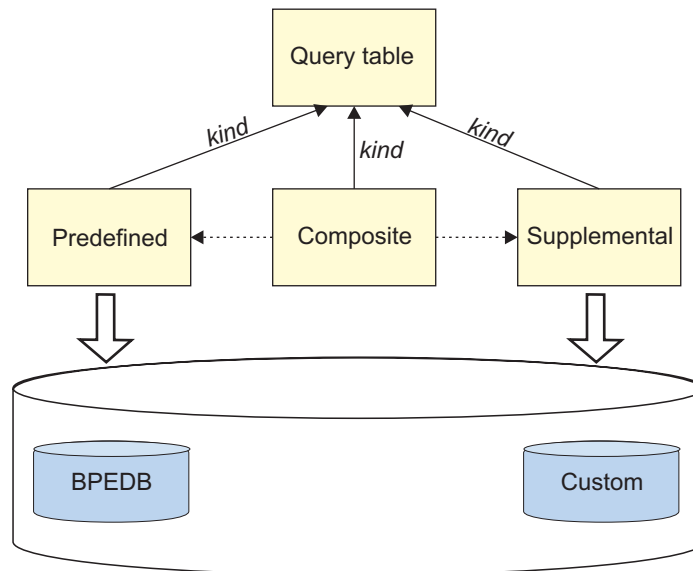


Figure 23. Tables de requêtes dans Business Process Choreographer

Les tables de requêtes sont représentées au moyen de modèles similaires dans l'environnement d'exécution et peuvent être interrogées à l'aide de l'API de table de requête. Tandis que les tables de requêtes prédéfinies et supplémentaires pointent directement sur des tables ou des vues dans la base de données, les tables composites sont constituées de parties de ces données, qu'elles mettent à disposition sous forme de table unique.

Les tables de requêtes étendent les possibilités offertes par les vues de base de données prédéfinies et les interfaces de requête existantes de Business Process Choreographer. Leurs points forts sont les suivants :

- Elles sont optimisées pour exécuter des requêtes de liste de processus et de tâches en utilisant des modèles d'accès eux-mêmes optimisés pour délivrer les meilleures performances.
- Elles simplifient et consolident l'accès aux informations nécessaires.
- Elles permettent de configurer très précisément les options d'autorisation et de filtrage.

Il est possible de personnaliser des tables de requêtes. Les options de configuration peuvent par exemple stipuler qu'une table de requête doit contenir uniquement les tâches ou les instances de processus relatives à un scénario particulier. Lorsque les performances sont primordiales, par exemple dans le cas de requêtes concernant une liste de processus ou de tâches volumineuse, utilisez des tables de requêtes.

L'outil Query Table Builder est fourni sous forme de plug-in Eclipse pour vous permettre :

- de développer des tables de requêtes composites et supplémentaires ;
- d'importer et d'exporter des définitions de tables de requêtes au format XML.

L'outil Query Table Builder peut être téléchargé à partir du site des SupportPacs WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Concepts associés

Requêtes portant sur les données des processus métier et des tâches

Les données d'instance des processus métier et des tâches humaines à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche humaine grâce à une interface de requête.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

Filtres et critères de sélection des tables de requêtes

Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requêtes.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Requêtes sur des tables de requêtes pour l'extraction de métadonnées

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes. Des méthodes sont disponibles pour permettre l'extraction de métadonnées des tables de requêtes.

Internationalisation pour les métadonnées des tables de requêtes

L'internationalisation est prise en charge pour les métadonnées des tables de requêtes.

Tables de requêtes et performances des requêtes

Les tables de requêtes offrent un nouveau modèle de programmation propre, conçu pour le développement d'applications clientes qui extraient des listes de tâches humaines et de processus métier dans Business Process Choreographer. Les tables de requêtes ont un effet bénéfique sur les performances des requêtes. Vous trouverez ci-après une description des options applicables aux tables de requêtes et des paramètres de l'API de tables de requêtes qui ont une incidence sur les performances des requêtes. Des informations sont également fournies à propos d'autres facteurs qui influent sur les performances.

Tâches associées

☞ Administration des tables de requêtes

Utilisez le script `wsadmin manageQueryTable.py` pour administrer les tables de requêtes de Business Process Choreographer qui ont été développées à l'aide de Query Table Builder. Contrairement aux tables de requêtes prédéfinies, qui sont prêtes à l'emploi, les tables de requêtes composites et supplémentaires doivent être déployées sur WebSphere Process Server avant de pouvoir être utilisées avec l'API de table de requêtes.

☞ Déploiement de tables de requêtes

Utilisez le script `manageQueryTable.py` pour déployer des tables de requêtes composites et supplémentaires dans Business Process Choreographer. Les tables de requêtes doivent être déployées sur un serveur autonome en cours d'exécution ou dans un cluster dont au moins un des membres est en cours d'exécution. L'annulation du déploiement des tables supplémentaires et composites doit également être réalisée sur les serveurs en cours d'exécution. Pour les tables de requêtes supplémentaires, les objets physiques de base de données associés (vue de base de données ou table de base de données, par exemple) doivent le cas échéant être créés avant l'utilisation de la table de requêtes.

Référence associée

☞ Vue de la base de données Business Process Choreographer

Ces informations de référence décrivent les colonnes disponibles dans les vues de base de données prédéfinies.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que `TASK` ou `PROCESS_INSTANCE`.

Les tables de requêtes prédéfinies utilisent les mêmes données physiques sous-jacentes que les vues de base de données prédéfinies et présentent donc la même structure. Cependant, elles étendent les fonctionnalités des vues de base de données prédéfinies et en améliorent les performances, car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

Les tables de requêtes prédéfinies peuvent être interrogées directement à l'aide de l'API de table de requêtes. Lorsque vous accédez aux tables en utilisant cette API, vous avez un plus grand choix d'options de configuration qu'avec l'API de requête. Vous pouvez développer une table de requêtes composite qui contient toutes les informations à récupérer lors de l'exécution de la requête, et non seulement les informations d'une seule table.

L'autorisation est activée pour tous les éléments de travail : éléments de travail everybody, individual, group et inherited. Sauf mention du contraire, pour les tables de requêtes prédéfinies contenant des données d'instance, l'API de table de requêtes utilise par défaut les éléments de travail everybody, individual et group.

Propriétés

Les tables de requêtes prédéfinies ont les propriétés suivantes :

Tableau 6. Propriétés des tables de requêtes prédéfinies

Propriété	Description
Nom	Le nom de la table de requêtes est le nom, en majuscules, de l'une des vues prédéfinies de la base de données ; par exemple, TASK.
Attributs	<p>Les attributs des tables de requêtes prédéfinies définissent les éléments d'information disponibles pour les requêtes. Il s'agit des noms en majuscules des colonnes spécifiées par les vues prédéfinies de la base de données.</p> <p>Les attributs sont définis avec un nom et un type. Le type est l'un des suivants :</p> <ul style="list-style-type: none"> • Boolean : une valeur booléenne • Decimal : un nombre en virgule flottante • ID : un ID d'objet, tel que le TKIID de la table de requêtes TASK • Number : un entier, court (type short) ou long (type long) • String : une chaîne • Timestamp : un horodatage
Autorisation	<p>Les tables de requêtes prédéfinies utilisent soit l'autorisation par instance, soit l'autorisation par rôle.</p> <ul style="list-style-type: none"> • Pour les tables de requêtes prédéfinies qui contiennent des données d'instance, l'autorisation par instance est obligatoire. Cela signifie que la requête renvoie seulement les objets avec des éléments de travail destinés à l'utilisateur qui exécute la requête. Cependant, en utilisant l'objet AdminAuthorizationOptions, vous pouvez limiter cette vérification à un simple contrôle de l'existence d'un élément de travail pour tout utilisateur. L'utilisateur doit avoir le rôle J2EE BPESystemAdministrator pour les requêtes ainsi configurées. • Pour les tables de requêtes prédéfinies qui contiennent des données de modèle, l'autorisation par rôle est obligatoire ; cela signifie que leur contenu n'est accessible qu'aux utilisateurs qui font partie du rôle J2EE BPESystemAdministrator.

Tables de requêtes prédéfinies avec des données d'instance

Le tableau suivant présente les tables de requêtes prédéfinies qui contiennent des données d'instance. Ces tables de requêtes se caractérisent par ce qui suit :

- Elles peuvent être utilisées comme table de requête principale d'une table composite.
- Elles utilisent l'autorisation par instance si elles sont interrogées directement. La technique utilisée à cet effet est une jointure (SQL-) avec la vue qui stocke les informations d'autorisation, c'est-à-dire la vue ou la table de requêtes prédéfinie WORK_ITEM.
- Elles contiennent des données d'instance ; par exemple, celles d'instances de tâche ou d'instances de processus.

Tableau 7. Tables de requêtes prédéfinies contenant des données d'instance

Données d'instance	Nom de la table de requêtes
Informations sur les activités d'une instance de processus.	ACTIVITY
	ACTIVITY_ATTRIBUTE
	ACTIVITY_SERVICE
Informations sur les escalades appartenant aux tâches humaines.	ESCALATION
	ESCALATION_CPROP
	ESCALATION_DESC
Informations sur les instances de processus.	PROCESS_ATTRIBUTE
	PROCESS_INSTANCE
	QUERY_PROPERTY
Informations sur les tâches humaines.	TASK
	TASK_CPROP
	TASK_DESC

La table de requêtes WORK_ITEM contient aussi des données d'instance, mais elle n'est pas utilisable comme table de requêtes principale ni comme table de requêtes attachée. Les données d'élément de travail sont disponibles implicitement lors de l'interrogation des tables de requêtes qui utilisent l'autorisation par instance. Autrement dit, lorsque vous interrogez une table de requêtes utilisant l'autorisation par instance, vous pouvez utiliser comme critères les attributs de la table WORK_ITEM, même s'ils ne sont pas explicitement spécifiés par la table que vous interrogez.

Tables de requêtes prédéfinies avec des données de modèle

Pour les tables de requêtes prédéfinies qui contiennent des données de modèle, l'autorisation par rôle est obligatoire. Elles ne peuvent être interrogées que par les administrateurs, à l'aide de l'objet AdminAuthorizationOptions.

Le tableau suivant présente les tables de requêtes prédéfinies qui contiennent des données de modèle. Ces tables de requêtes se caractérisent par ce qui suit :

- Elles peuvent être utilisées comme table de requête principale d'une table composite.
- Elles utilisent l'autorisation par rôle si elles sont interrogées directement. Cela signifie que l'appelant doit faire partie du rôle J2EE BPESystemAdministrator et que l'objet AdminAuthorizationOptions doit être utilisé.
- Elles contiennent des données de modèle ; par exemple, celles des modèles de tâche ou des modèles de processus.

Tableau 8. Tables de requêtes prédéfinies contenant des données de modèle

Données de modèle	Nom de la table de requêtes
Informations sur les composants d'application.	APPLICATION_COMP
Informations sur les modèles d'escalade.	ESC_TEMPL
	ESC_TEMPL_CPROP
	ESC_TEMPL_DESC
Informations sur les modèles de processus.	PROCESS_TEMPLATE
	PROCESS_TEMPL_ATTR
Informations sur les modèles de tâche.	TASK_TEMPL
	TASK_TEMPL_CPROP
	TASK_TEMPL_DESC

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Référence associée

 Vue de la base de données Business Process Choreographer

Ces informations de référence décrivent les colonnes disponibles dans les vues de base de données prédéfinies.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données

externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Les tables de requêtes supplémentaires sont associées à des tables ou des vues dans la base de données de Business Process Choreographer. Il s'agit de tables de requêtes qui contiennent des données métier supplémentaires, tenues à jour par des applications externes (celles du client). Les tables de requêtes supplémentaires fournissent des informations dans une table composite qui vient en complément du contenu d'une table de requêtes prédéfinie.

Les tables de requêtes supplémentaires ont les propriétés suivantes :

Tableau 9. Propriétés des tables de requêtes supplémentaires

Propriété	Description
Nom	<p>Le nom d'une table de requêtes doit être unique au sein d'une même installation Business Process Choreographer. A l'exécution de la requête, c'est ce nom qui est utilisé pour identifier la table de requêtes interrogée.</p> <p>Une table de requêtes est identifiée de manière unique par son nom, qui est de la forme <i>préfixe.nom</i>. La longueur de <i>préfixe.nom</i> ne doit pas dépasser 28 caractères. Le préfixe 'IBM' est réservé et ne doit pas être utilisé ; par exemple, 'COMPANY.BUS_DATA' est un nom correct.</p>
Nom de base de données	Le nom de la table ou vue associée dans la base de données. Seules des lettres majuscules sont acceptées.
Schéma de base de données	Le schéma de la table ou vue associée dans la base de données. Seules des lettres majuscules sont acceptées. Ce schéma doit être différent de celui de la base de données de Business Process Choreographer. Cependant, la table ou la vue doit être accessible avec la même source de données JDBC que celle qui est utilisée pour accéder à la base de données de Business Process Choreographer.
Attributs	<p>Les attributs des tables de requêtes supplémentaires définissent les éléments d'information disponibles pour les requêtes. Ces attributs doivent avoir le même nom que les colonnes auxquelles ils correspondent dans la table ou la vue de base de données.</p> <p>Les attributs sont définis avec un nom et un type. Le nom est défini en majuscules. Le type est l'un des suivants :</p> <ul style="list-style-type: none"> • Boolean : une valeur booléenne • Decimal : un nombre en virgule flottante • ID : un ID d'objet d'une longueur de 16 octets, tel que le TKIID de la table de requêtes TASK • Number : un entier, court (type short) ou long (type long) • String : une chaîne • Timestamp : un horodatage
Jointures	<p>Des jointures doivent être définies sur les tables supplémentaires si elles sont attachées à une table dite "principale" pour former des tables de requêtes composites. Une jointure définit quels attributs sont utilisés pour corréler les informations de la table de requêtes supplémentaire avec celles de la table de requêtes principale.</p> <p>Lorsqu'une jointure est définie, l'attribut source et l'attribut cible doivent être du même type.</p>

Tableau 9. Propriétés des tables de requêtes supplémentaires (suite)

Propriété	Description
Autorisation	Aucun contrôle d'autorisation n'est spécifié pour les tables de requêtes supplémentaires ; par conséquent, tous les utilisateurs authentifiés peuvent voir leur contenu.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Les tables de requêtes composites autorisent une configuration fine des filtres et des options d'autorisation, permettant ainsi d'optimiser l'accès aux données lors de l'exécution des requêtes. Elles n'ont pas de représentation spécifique de données dans la base de données ; elles accèdent simplement au contenu correspondant aux tables prédéfinies et supplémentaires dont elles sont constituées. Les tables de requêtes composites sont créées au moyen de code SQL, qui est optimisé pour les requêtes sur les listes de tâches et de processus.

Les tables de requêtes composites sont conçues par les développeurs d'applications clientes. Dans les scénarios de production, leur utilisation est à privilégier par rapport à l'emploi des API de requêtes standard de Business Process Choreographer, car elles fournissent une abstraction de l'implémentation proprement dite des requêtes et permettent donc d'optimiser celles-ci. En outre, les tables de requêtes composites autorisent des changements à l'exécution sans que cela nécessite de redéployer le client qui y accède.

La figure suivante offre une vue d'ensemble du contenu des tables de requêtes composites :

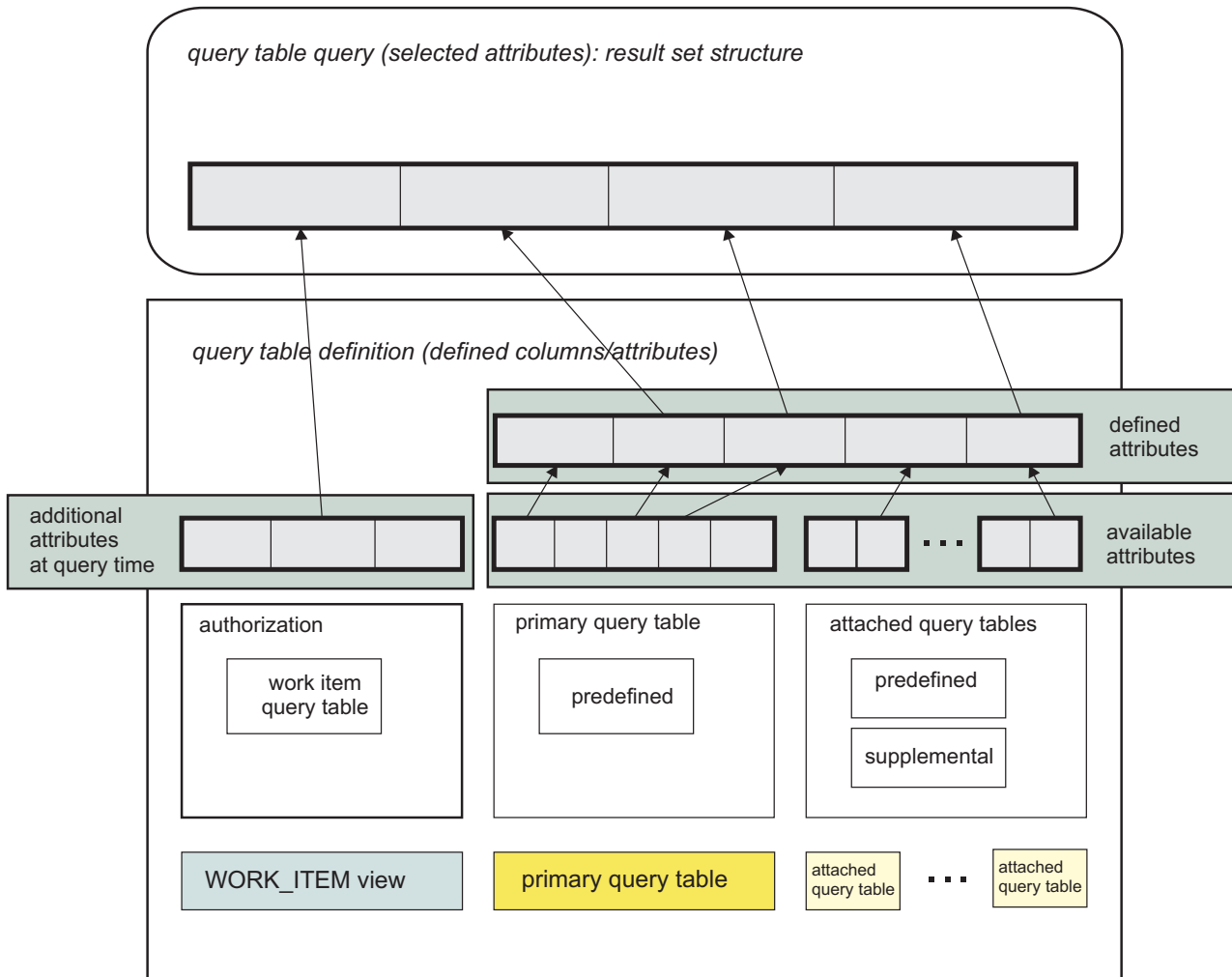


Figure 24. Contenu des tables de requêtes composites

Toutes les tables de requêtes composites sont définies avec une table de requêtes principale et zéro, une ou plusieurs tables de requêtes attachées.

La table de requêtes principale :

- Constitue l'information principale contenue dans une table de requêtes composite.
- Doit être l'une des tables de requêtes prédéfinies.
- Identifie de manière unique, par sa clé primaire, chaque objet contenu dans la table de requêtes composites. Par exemple, dans le cas de la table de requêtes prédéfinie TASK, il s'agit de l'ID de tâche TKIID.
- Contrôle l'accès au contenu d'une table de requêtes en utilisant les éléments de travail qui figurent dans la table de requêtes WORK_ITEM, si l'autorisation par instance est utilisée.
- Détermine la liste des objets retournés comme lignes d'une table lorsque la table composite est interrogée.

Les tables de requêtes attachées :

- Peuvent être des tables de requêtes prédéfinies ou supplémentaires déjà déployées sur le système.
- Ont pour objectif de fournir des informations complémentaires de celles qui sont fournies par la table de requêtes principale. Par exemple, si TASK est la table de requêtes principale, la description de chaque tâche fournie dans la table de requêtes TASK_DESC peut être ajoutée au contenu de la table de requêtes composite.

En général, la table de requêtes principale est choisie en fonction de l'objectif de la table de requêtes composite.

- Si la table composite doit décrire une liste de tâches, la table de requêtes TASK sera désignée comme table principale.
- Si la table composite doit décrire une liste de processus, la table de requêtes PROCESS_INSTANCE sera désignée comme table principale.
- Les listes d'activités sont obtenues en utilisant ACTIVITY comme table de requêtes principale.
- Les listes d'escalades de tâches humaines sont obtenues en utilisant ESCALATION comme table de requêtes principale.

La relation entre table de requêtes principale et tables attachées

La relation entre la table principale et une table attachée est du type un à un ou un à zéro, ce qui signifie que, dans la table principale, chaque ligne doit correspondre au plus à une ligne de la table attachée. Si le type de relation un à un ou un à zéro n'est pas respecté, une exception d'exécution (RuntimeException) se produira à l'exécution de la requête.

La corrélation entre table principale et chaque table attachée est réalisée par un attribut de jointure qui est défini sur la table attachée. Pour les tables de requêtes prédéfinies, cet attribut de jointure ne peut pas être changé, car il décrit la relation entre les données dans les différentes tables de requêtes de Process Choreographer. Cet attribut de jointure est généralement suffisant pour maintenir la relation un à un ou un à zéro. Par exemple, l'attribut CONTAINMENT_CTX_ID est utilisé sur la table de requêtes TASK pour attacher les données de l'instance de processus associée, laquelle est identifiée par l'attribut PIID dans la table de requêtes PROCESS_INSTANCE. Cependant, lorsque la relation créée par la jointure est du type un à plusieurs, un critère additionnel doit être spécifié afin retrouver une relation du type un à un ou un à zéro. C'est ce qu'on appelle le critère de sélection.

Les critères de sélection sont spécifiés pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder. Ils sont utilisés dans la définition des tables de requêtes pour choisir un élément d'information parmi les multiples éléments d'une relation un à plusieurs. Par exemple, il peut s'agir de l'expression "LOCALE='fr_FR' ". Une tâche peut avoir plusieurs descriptions identifiées chacune par un code d'environnement local différent.

Exemple 1 :

La figure suivante illustre l'emploi de critères de sélection spécifiés sur les tables de requêtes attachées :

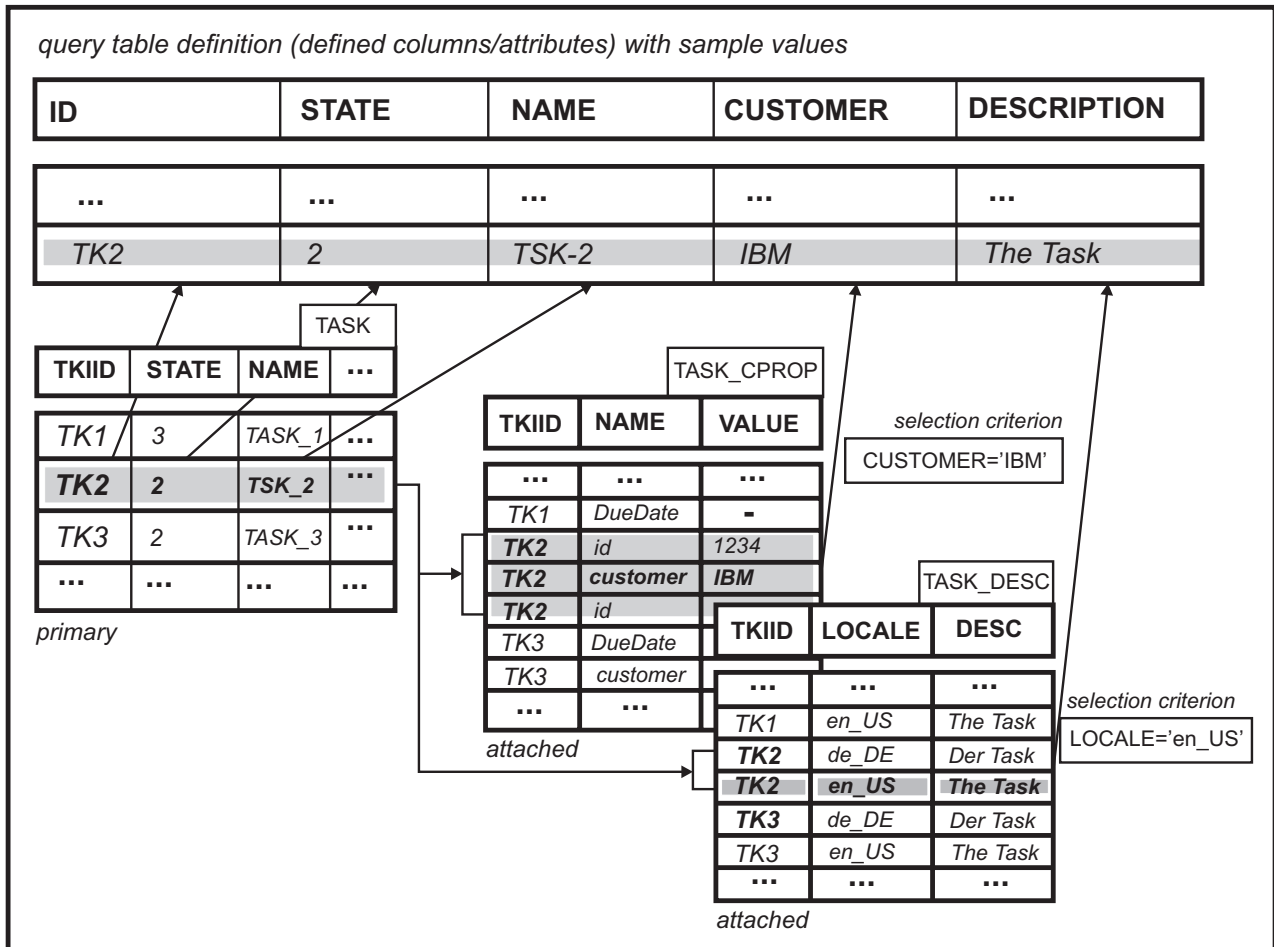


Figure 25. Table de requêtes composite avec critères de sélection

La table de requêtes composite contient les attributs ID, STATE, NAME, CUSTOMER et DESCRIPTION.

- Les attributs ID (le TKIID), STATE et NAME sont fournis par la table de requêtes principale TASK.
- CUSTOMER est une propriété personnalisée définie sur la table TASK. Les propriétés personnalisées sont stockées dans la table de requêtes TASK_CPROP. Pour une tâche particulière, une propriété personnalisée est identifiée de manière unique par son nom. Ce point est reflété dans le critère de sélection "CUSTOMER=' IBM' ".

- DESCRIPTION est la description de la tâche ; elle est stockée dans la table de requêtes TASK_DESC. La description de tâche associée à une instance de tâche particulière est identifiée de manière unique par son environnement local. Ce point est reflété dans le critère de sélection "LOCALE='en_US'".

Exemple 2 :

Si TASK est la table de requêtes principale et que TASK_DESC y est attachée, un environnement local particulier doit être choisi ; il s'agit ici de l'attribut LOCALE de la table TASK_DESC. L'objectif de cet exemple est d'illustrer la relation entre la table principale et la table attachée en utilisant TASK comme table principale et TASK_DESC comme table attachée. Le tableau suivant montre des exemples de contenus d'une table composite, avec un critère de sélection valide spécifié pour la table attachée TASK_DESC.

Tableau 10. Contenus valides d'une table de requêtes composite

Table principale TASK	Table attachée TASK_DESC	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_two	en_US	This is a description.
...

Le tableau suivant présente des contenus hypothétiques qui ne sont pas valides (en **gras**) si le critère de sélection n'est pas défini correctement, ce qui signifie que la relation un à un ou un à zéro n'est pas respectée.

Tableau 11. Contenus non valides d'une table de requêtes composite

TASK (table de requêtes principale)	TASK_DESC (table de requêtes attachée)	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_one	de_DE	Das ist eine Beschreibung.
...

Propriétés

Les tables de requêtes composites ont les propriétés suivantes :

Tableau 12. Propriétés des tables de requêtes composites

Propriété	Description
Nom	<p>Le nom d'une table de requêtes doit être unique au sein d'une même installation Business Process Choreographer. A l'exécution de la requête, c'est ce nom qui est utilisé pour identifier la table de requêtes interrogée.</p> <p>Une table de requêtes est identifiée de manière unique par son nom, qui est de la forme <i>préfixe.nom</i> dans le cas d'une table de requêtes composite. La longueur de <i>préfixe.nom</i> ne doit pas dépasser 28 caractères. Le préfixe 'IBM' est réservé et ne doit pas être utilisé ; par exemple, 'COMPANY.TODO_TASK_LIST' est un nom correct.</p>

Tableau 12. Propriétés des tables de requêtes composites (suite)

Propriété	Description
Attributs	<p>Les attributs des tables de requêtes composites définissent les éléments d'information disponibles pour les requêtes.</p> <p>Les attributs sont définis avec un nom en majuscules. Leur type est hérité de celui de l'attribut auquel ils font référence, qui est l'un des suivants :</p> <ul style="list-style-type: none"> • Boolean : une valeur booléenne • Decimal : un nombre en virgule flottante • ID : un ID d'objet, tel que le TKIID de la table de requêtes TASK • Number : un entier, court (type short) ou long (type long) • String : une chaîne • Timestamp : un horodatage <p>Les attributs d'une table de requêtes composite sont définis via une référence aux attributs de la table principale ou des tables attachées. Les attributs des tables de requêtes composites héritent des types et des constantes des attributs auxquels ils font référence.</p> <p>Outre les attributs qui font partie de la définition de la table de requêtes, les données d'élément de travail peuvent être interrogées à l'exécution. Les conditions à remplir sont que la table de requêtes principale contienne des données d'instance, (c'est le cas des tables de requêtes TASK et PROCESS_INSTANCE) et que l'autorisation par instance soit utilisée sur la table de requêtes composite. Par exemple, la requête peut être définie de manière à renvoyer uniquement les tâches humaines pour lesquelles l'utilisateur est un propriétaire potentiel.</p>

Tableau 12. Propriétés des tables de requêtes composites (suite)

Propriété	Description
Autorisation	<p>Chaque table de requêtes composite définit si l'autorisation par instance ou par rôle est utilisée lorsque des requêtes sont exécutées dessus (ou s'il n'y a pas de contrôle d'autorisation).</p> <p>Si l'autorisation par instance est définie, la requête renvoie seulement les objets avec des éléments de travail destinés à l'utilisateur qui exécute la requête. Cependant, en utilisant l'objet <code>AdminAuthorizationOptions</code>, vous pouvez limiter cette vérification à un simple contrôle de l'existence d'un élément de travail pour tout utilisateur. L'utilisateur doit faire partie du rôle <code>J2EE BPESystemAdministrator</code> pour les requêtes ainsi configurées, et l'objet <code>AdminAuthorizationOptions</code> doit être passé à l'API de table de requêtes.</p> <p>Si l'autorisation par rôle est définie, l'utilisateur doit faire partie du rôle <code>J2EE BPESystemAdministrator</code> pour les requêtes ainsi configurées, et l'objet <code>AdminAuthorizationOptions</code> doit être passé à l'API de table de requêtes.</p> <p>Si aucun contrôle d'autorisation n'est défini, l'exécution de la requête a lieu sans vérification préalable de l'existence des éléments de travail des objets associés dans la table de requête. Tous les utilisateurs authentifiés peuvent voir le contenu de la table de requêtes.</p> <p>L'autorisation par instance peut être définie si la table de requêtes principale contient des données d'instance ; l'autorisation par rôle peut être définie si la table de requêtes principale contient des données de modèle. L'absence de contrôle d'autorisation peut être définie sur les tables de requêtes composites, quelle que soit leur table principale.</p>

Filtres

Les filtres servent à limiter le nombre d'objets ou de lignes contenus dans une table de requêtes composite.

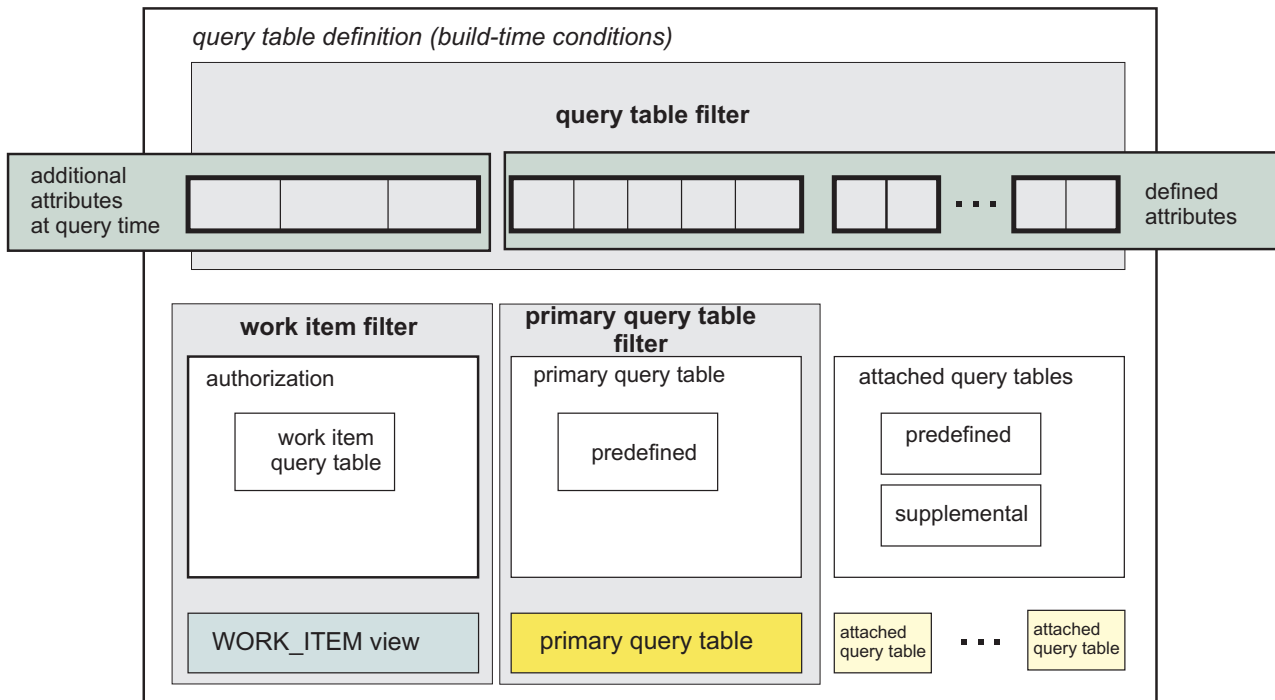


Figure 26. Filtres dans les tables de requêtes composites

Pendant le développement d'une table de requêtes composite, des filtres peuvent être définis et appliqués :

- à la table de requêtes principale, comme filtre de la table de requêtes principale ;
- à la table de requêtes WORK_ITEM, laquelle est implicitement disponible et assure le contrôle d'accès (autorisation) si la table de requêtes principale contient des données d'instance. Ce filtre est appelé filtre d'autorisation ; il est disponible uniquement si la table composite est configurée pour utiliser l'autorisation par instance.
- à la table de requêtes composite, comme filtre de table de requêtes.

Les filtres sont définis durant le développement des tables de requêtes. Par exemple, avec une table de requêtes composite dont la table principale est TASK, il est possible de filtrer les tâches de manière à exclure toutes celles qui ne sont pas à l'état prêt (en utilisant l'expression "STATE=STATE_READY" comme filtre de la table de requêtes principale).

Autorisation

L'autorisation d'accès au contenu d'une table de requêtes composite est similaire à celle qui régit l'accès à la table utilisée comme table principale. La différence est que la table composite peut être configurée pour être plus restrictive.

- Si l'autorisation par instance est utilisée, les données contenues dans la table composite sont vérifiées par rapport aux éléments de travail dans la table de requêtes WORK_ITEM. Cette vérification s'effectue par rapport à la table de requêtes principale. Les éléments de travail 'everybody', 'individual', 'group' et 'inherited' sont utilisés pour la vérification, selon la configuration de la table de requêtes composite. Si des éléments de travail 'inherited' sont spécifiés, les objets ayant comme parent une instance de processus (par exemple, une tâche humaine participante) et qui sont liés à un élément de travail 'everybody', 'individual' ou

'group' (en fonction de la configuration), sont présents dans la table de requêtes composite. En général, les éléments de travail 'inherited' ne sont utiles qu'aux administrateurs.

- Les tables composites dont la table principale contient des données de modèle ne doivent pas être configurées pour utiliser l'autorisation par instance. Si l'autorisation par rôle est utilisée, les requêtes ne peuvent être exécutées que par les utilisateurs faisant partie du rôle J2EE BPESystemAdministrator, et l'objet AdminAuthorizationOptions doit être utilisé.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du

développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

L'outil Query Table Builder, disponible sous forme de plug-in Eclipse, peut être téléchargé sur le site des SupportPacs WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

L'utilisation de tables de requêtes a une incidence sur le développement et le déploiement des applications. Les étapes suivantes décrivent les rôles mis en jeu lorsque vous concevez et développez une application Business Process Choreographer utilisant des tables de requêtes.

Tableau 13. *Étapes de développement de tables de requêtes*

Étape	Rôle	Description
1. Analyse	Analyste métier, développeur de client	Analysez les besoins de l'application cliente et déterminez quelles tables de requêtes sont nécessaires. Posez-vous les questions suivantes : <ul style="list-style-type: none"> • Combien de listes de tâches ou de processus faut-il fournir à l'utilisateur ? Y a-t-il des listes de tâches ou de processus qui pourraient partager la même table de requêtes ? • Quel type d'autorisation est utilisé ? Par instance, par rôle ou aucun ? • Existe-t-il, dans le système, d'autres tables de requêtes déjà prêtes à l'emploi qui pourraient être réutilisées ? • Les tables de requêtes doivent-elles fournir le contenu en plusieurs langues ? Si oui, les critères de sélection sur les tables de requêtes attachées doivent être <code>LOCALE=\$LOCALE</code>.
2. Développement des tables de requêtes	Développeur de client, analyste métier	Développez les tables de requêtes à utiliser dans l'application cliente. Efforcez-vous de spécifier leur définition de manière à optimiser les performances obtenues avec les requêtes qu'elles fournissent.
3. Déploiement des tables de requêtes	Administrateur	Les tables de requêtes doivent être déployées dans l'environnement d'exécution avant de pouvoir être utilisées. Cette étape est réalisée au moyen de la commande <code>wsadmin manageQueryTable.py</code> .
4. Requêtes sur les tables de requêtes	Développeur de client	La dernière étape du développement consiste à exécuter des requêtes sur les tables. Le développeur de l'application cliente doit connaître le nom de la table à interroger et ses attributs.

L'exemple de code ci-dessous utilise l'API de table de requêtes pour interroger une table de requêtes. Pour des raisons de simplification, les exemples 1 et 2 fournis interrogent la table de requêtes prédéfinie TASK. Les exemples 3 et 4 interrogent une table de requêtes composite et supposent que celle-ci a été préalablement déployée sur le système. Dans le cadre du développement d'applications, il est recommandé d'utiliser des tables de requêtes composites au lieu d'interroger directement les tables de requêtes prédéfinies.

Exemple 1

```
// permet d'obtenir le contexte d'affectation de nom et de rechercher l'interface
// EJB home de Business Flow Manager ; cette interface
// doit être mise en cache pour des raisons de performances
// nous supposons également qu'il existe une référence EJB
// à l'EJB local de Business Flow Manager
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
    (LocalBusinessFlowManagerHome)
    ctx.lookup("java:comp/env/ejb/BFM");

// crée le module de remplacement de Business Flow Manager côté client
LocalBusinessFlowManager bfm = home.create();

// *****
// ***** exemple 1 *****
// *****

// exécute une requête sur la table de requêtes prédéfinie
// TASK ; il s'agit ici d'une simple liste Mes tâches
EntityResultSet ers = null;
ers = bfm.queryEntities("TASK", null, null, null);

// imprime le résultat dans STDOUT
EntityInfo entityInfo = ers.getEntityInfo();
List attList = entityInfo.getAttributeInfo();
int attSize = attList.size();

Iterator iter = ers.getEntities().iterator();
while( iter.hasNext() ) {
    System.out.print("Entity: ");
    Entity entity = (Entity) iter.next();
    for (int i = attSize - 1; i >= 0; i--) {
        AttributeInfo ai = (AttributeInfo) attList.get(i);
        System.out.print(
            entity.getAttributeValue(ai.getName()));
    }
    System.out.println();
}
```

Exemple 2

```
// *****
// ***** exemple 2 *****
// *****

// identique à l'exemple 1, mais utilise des requêtes
// par ligne
RowResultSet rrs = null;
rrs = bfm.queryRows("TASK", null, null, null);

attList = rrs.getAttributeInfo();
attSize = attList.size();

// imprime le résultat dans STDOUT
while (rrs.next()) {
    System.out.print("Row: ");
```

```

for (int i = attSize - 1; i >= 0; i--) {
    AttributeInfo ai = (AttributeInfo) attList.get(i);
    System.out.print(
        rrs.getAttributeValue(ai.getName()));
    }
System.out.println();
}

```

Exemple 3

```

// *****
// ***** exemple 3 *****
// *****

// exécute une requête sur une table de requêtes composite
// préalablement déployée sur le système ;
// on suppose que le nom est COMPANY.TASK_LIST
ers = bfm.queryEntities(
    "COMPANY.TASK_LIST", null, null, null);
^
// imprime le résultat dans STDOUT ...

```

Exemple 4

```

// *****
// ***** exemple 4 *****
// *****

// requête sur la même table de requêtes que dans l'exemple 3,
// mais utilise des options personnalisées
FilterOptions fo = new FilterOptions();

// renvoie uniquement les objets à l'état Prêt
fo.setQueryCondition("STATE=STATE_READY");

// trie les objets en fonction de leur ID
fo.setSortAttributes("ID");

// limite à 50 le nombre d'entités
fo.setThreshold(50);

// récupère uniquement un sous-ensemble des attributs définis
// au niveau de la table de requêtes
fo.setSelectedAttributes("ID, STATE, DESCRIPTION");

AuthorizationOptions ao = new AuthorizationOptions();

// ne renvoie pas les objets que tous les utilisateurs sont
// autorisés à voir
ao.setEverybodyUsed(Boolean.FALSE);

ers = bfm.queryEntities(
    "COMPANY.TASK_LIST", fo, ao, null);

// imprime le résultat dans STDOUT ...

```

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Filtres et critères de sélection des tables de requêtes

Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requêtes.

Tâches associées

☞ Administration des tables de requêtes

Utilisez le script `wsadmin manageQueryTable.py` pour administrer les tables de requêtes de Business Process Choreographer qui ont été développées à l'aide de Query Table Builder. Contrairement aux tables de requêtes prédéfinies, qui sont prêtes à l'emploi, les tables de requêtes composites et supplémentaires doivent être déployées sur WebSphere Process Server avant de pouvoir être utilisées avec l'API de table de requêtes.

☞ Déploiement de tables de requêtes

Utilisez le script `manageQueryTable.py` pour déployer des tables de requêtes

composites et supplémentaires dans Business Process Choreographer. Les tables de requêtes doivent être déployées sur un serveur autonome en cours d'exécution ou dans un cluster dont au moins un des membres est en cours d'exécution. L'annulation du déploiement des tables supplémentaires et composites doit également être réalisée sur les serveurs en cours d'exécution. Pour les tables de requêtes supplémentaires, les objets physiques de base de données associés (vue de base de données ou table de base de données, par exemple) doivent le cas échéant être créés avant l'utilisation de la table de requêtes.

Filtres et critères de sélection des tables de requêtes

Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requêtes.

Pour toute information sur l'installation de l'outil Query Table Builder, consultez le site des SupportPacs de WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Attributs

Lorsqu'il est question d'attributs dans une expression, il s'agit de ceux des tables de requêtes. Selon l'endroit où se situe l'expression, différents attributs sont disponibles. Le développeur d'applications clientes ne peut utiliser des expressions que dans les filtres de requête qu'il passe à l'API de table de requêtes. Le développeur de tables de requêtes composites a quant à lui la possibilité d'utiliser des expressions en des endroits plus variés. Le tableau suivant décrit les attributs disponibles et en quels endroits ils sont utilisables.

Tableau 14. Les attributs des tables de requêtes et leur utilisation dans les expressions

Où	Expression	Attributs disponibles
API de table de requêtes	Filtre de requête	<ul style="list-style-type: none"> Tous les attributs définis dans la table de requêtes. Si l'autorisation par instance est utilisée, tous les attributs définis dans la table de requêtes WORK_ITEM, préfixés avec 'WI.' . <p>Exemples :</p>
Table de requêtes composite	Filtre de table de requêtes	<ul style="list-style-type: none"> STATE=STATE_READY, si la table de requêtes contient un attribut STATE et si une constante STATE_READY est définie pour cet attribut STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER, si la table de requêtes contient un attribut STATE et si elle utilise l'autorisation par instance
	Filtre de la table de requêtes principale	<ul style="list-style-type: none"> Tous les attributs définis pour la table de requêtes principale. <p>Exemple :</p> <ul style="list-style-type: none"> STATE=STATE_READY, si la table de requêtes contient un attribut STATE et si une constante STATE_READY est définie pour cet attribut
	Filtre d'autorisation	<ul style="list-style-type: none"> Tous les attributs définis dans la table de requêtes prédéfinie WORK_ITEM, préfixés avec 'WI.' . <p>Exemple :</p> <ul style="list-style-type: none"> WI.REASON=REASON_POTENTIAL_OWNER
	Critère de sélection	<ul style="list-style-type: none"> Tous les attributs définis dans la table de requêtes attachée et associée. <p>Exemple :</p> <ul style="list-style-type: none"> LOCALE='en_US', si la table de requêtes attachée contient un attribut LOCALE, ce qui est le cas de la table TASK_DESC

La figure suivante montre les différents endroits où les filtres et critères de sélection peuvent être utilisés dans des expressions ; elle inclut également des exemples :

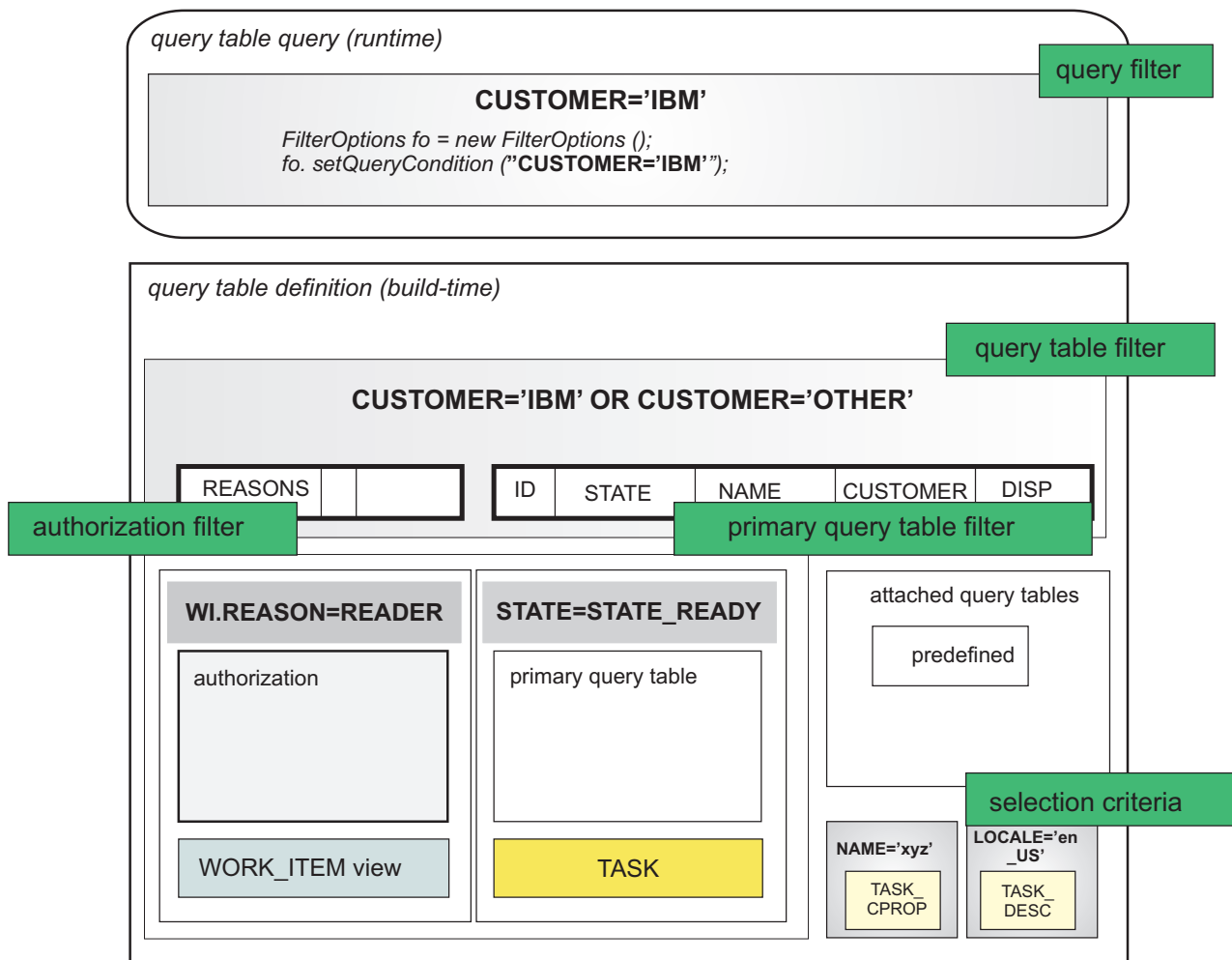


Figure 27. Filtres et critères de sélection dans des expressions

Expressions

Les expressions ont la syntaxe suivante :

```

expression ::= attribut op_binaire valeur |
             attribut op_unaire |
             attribut op_liste liste |
             (expression) |
             expression AND expression |
             expression > OR expression
  
```

Les règles suivantes s'appliquent :

- L'opérateur AND est évalué avant l'opérateur OR. Les sous-expressions peuvent être reliées par des opérateurs AND et OR.
- Les expressions peuvent être groupées au moyen de parenthèses, qui doivent être appariées.

Exemples :

- STATE = STATE_READY
- NAME IS NOT NULL
- STATE IN (2, 5, STATE_FINISHED)
- ((PRIORITY=1) OR (WI.REASON=2)) AND (STATE=2)

Une expression est prise en compte et évaluée dans une portée précise, qui détermine les attributs valides pour cette expression. Les critères de sélection ou les filtres de requête sont pris en compte et évalués dans la portée de la table de requêtes sur laquelle la requête est exécutée.

L'exemple suivant s'applique à une requête exécutée sur la table de requêtes prédéfinie TASK :

```
'(STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER)
OR (WI.REASON=REASON_OWNER)'
```

Opérateurs binaires

Les opérateurs binaires suivants sont disponibles :

op_binaire ::= = | < | > | <> | <= | >= | LIKE | NOT LIKE

Les règles suivantes s'appliquent :

- L'opérande côté gauche d'un opérateur binaire doit faire référence à un attribut d'une table de requêtes.
- L'opérande côté droit d'un opérateur binaire doit être une valeur littérale, une valeur constante ou un paramètre.
- Les opérateurs LIKE et NOT LIKE sont utilisables uniquement avec les attributs du type d'attribut STRING.
- Les opérandes côtés gauche et droit doivent avoir des types d'attribut compatibles.
- Les paramètres utilisateur doivent être compatibles avec le type d'attribut de l'attribut côté gauche.

Exemples :

- STATE > 2
- NAME LIKE 'start%'
- STATE <> PARAM(theState)

Opérateurs unaires

Les opérateurs unaires suivants sont disponibles :

op_unaire ::= IS NULL | IS NOT NULL

Les règles suivantes s'appliquent :

- L'opérande côté gauche d'un opérateur unaire doit faire référence à un attribut d'une table de requêtes. Les attributs valides dépendent de l'endroit où le filtre ou le critère de sélection est utilisé.
- Tous les attributs peuvent être testés pour déterminer s'ils sont NULL ; par exemple : CUSTOMER IS NOT NULL.

Exemple :

```
DESCRIPTION IS NOT NULL
```

Opérateurs de liste

Les opérateurs de liste suivants sont disponibles :

op_liste ::= IN | NOT IN

Les règles suivantes s'appliquent :

- L'opérande côté droit d'un opérateur de liste ne doit pas être remplacé par un paramètre utilisateur.
- Des paramètres utilisateurs peuvent apparaître dans la liste de l'opérande côté droit.

Exemple :

```
STATE IN (STATE_READY, STATE_RUNNING, PARAM(st), 1)
```

Les listes sont représentées comme suit :

```
liste ::= valeur [, liste]
```

Les règles suivantes s'appliquent :

- L'opérande côté droit d'un opérateur de liste ne doit pas être remplacé par un paramètre utilisateur.
- Des paramètres utilisateurs peuvent apparaître dans la liste de l'opérande côté droit.

Exemples :

- (2, 5, 8)
- (STATE_READY, STATE_CLAIMED)

Valeurs

Dans les expressions, une valeur peut être l'une des suivantes :

- **Constante** : une valeur constante, définie pour l'attribut concerné d'une table de requêtes prédéfinie. Par exemple, STATE_READY est une constante définie pour l'attribut STATE de la table prédéfinie TASK.
- **Littéral** : toute valeur codée en dur.
- **Paramètre** : un paramètre est remplacé par une valeur spécifique lors de l'exécution de la requête.

Des **constantes** sont disponibles pour certains attributs des tables de requêtes prédéfinies. Pour plus de détails sur ces constantes, reportez-vous à la description des vues prédéfinies. Seules les constantes qui définissent des valeurs entières sont exposées dans les tables de requêtes. Il est également possible d'utiliser, à la place des constantes, les valeurs littérales correspondantes ou des paramètres.

Exemples :

- STATE_READY, constante propre à l'attribut STATE de la table de requêtes TASK, peut être utilisée dans un filtre pour déterminer si la tâche est à l'état prêt.
- REASON_POTENTIAL_OWNER, constante propre à l'attribut REASON de la table de requêtes WORK_ITEM, peut être utilisée dans un filtre pour tester si l'utilisateur qui exécute la requête sur une table de requêtes est un propriétaire potentiel.
- Le filtre de requête STATE=STATE_READY est identique à STATE=2, si la requête est exécutée sur la table de requêtes TASK.

Des **littéraux** peuvent aussi être utilisés dans les expressions. Une syntaxe spéciale doit être employée pour les horodatages et les ID.

Exemples :

- STATE=1
- NAME='theName'

- `CREATED > TS ('2008-11-26 T12:00:00')`
- `TKTID=ID('_TKT:801a011e.9d57c52.ab886df6.1fcc0000')`

L'utilisation de **paramètres** dans les expressions permet de donner un caractère dynamique aux tables de requêtes composites. Il existe des paramètres utilisateur et des paramètres système :

- Un paramètre utilisateur est spécifié avec le format `PARAM (nom)`. Ce paramètre (sa valeur) doit être fourni au moment où la requête est exécutée. Il est passé à l'API de table de requêtes sous forme d'instance de la classe `com.ibm.bpe.api.Parameter`.
- Les paramètres système sont ainsi appelés parce que leurs valeurs sont fournies par l'environnement d'exécution (runtime) de table de requêtes, sans être spécifiées lors de l'exécution de la requête. Les paramètres système disponibles sont `$USER` et `$LOCALE`.
 - `$USER` est une chaîne identifiant l'utilisateur qui exécute la requête.
 - `$LOCALE` est une chaîne spécifiant le code standard de l'environnement local en vigueur au moment où la requête est exécutée. 'fr_FR' est un exemple de valeur utilisable pour le paramètre `$LOCALE`.

Vous pouvez spécifier un paramètre dans le critère de sélection d'une table de requêtes attachée afin de sélectionner un environnement local spécifique. Par exemple, si `TASK` est la table principale d'une table de requêtes composite et si `TASK_DESC` est l'une des tables attachées, vous pouvez faire en sorte d'obtenir la description de la tâche dans une langue particulière. Voici quelques exemples d'utilisation de paramètres :

- `STATE=PARAM(theState)`
- `LOCALE=$LOCALE`
- `OWNER=$USER`

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Le type d'autorisation utilisé lorsqu'une requête est exécutée sur une table de requêtes est défini sur la table elle-même.

- L'autorisation par instance indique que les objets dans la table de requêtes sont autorisés moyennant l'utilisation d'un élément de travail. A cet effet, un contrôle de l'existence d'un élément de travail adéquat est effectué.
- L'autorisation par rôle se fonde sur les rôles J2EE. Cela signifie que l'appelant doit faire partie du rôle J2EE BPESystemAdministrator pour voir le contenu de la table de requêtes. Ce type d'autorisation est disponible pour les tables de requêtes prédéfinies avec des données de modèle et pour les tables composites dont la table principale contient des données de modèle. Les objets contenus dans ces tables de requêtes n'ont pas d'élément de travail associé.
- Si aucun contrôle d'autorisation n'est spécifié, tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requêtes, après application des filtres.

Les types d'autorisation appliqués aux tables de requêtes prédéfinies, ainsi que les types d'autorisation qu'il est possible de configurer sur les tables composites et sur les tables supplémentaires, sont décrits dans le tableau suivant.

Tableau 15. Types d'autorisation pour les tables de requêtes

Table de requêtes	Autorisation par instance	Autorisation par rôle	Absence de contrôle d'autorisation
Prédéfinie	Requise pour les tables de requêtes prédéfinies qui contiennent des données d'instance.	Requise pour les tables de requêtes prédéfinies qui contiennent des données de modèle.	N/A
Composite	<p>Peut être désactivée, auquel cas il n'y a pas de contrôle d'autorisation et les contraintes de sécurité sont passées outre. Cela signifie que chaque utilisateur authentifié peut utiliser la table de requêtes pour extraire des données, qu'il soit ou non autorisé à accéder aux objets correspondants.</p> <p>Les tables composites dont la table principale contient des données de modèle ne doivent pas être configurées pour utiliser l'autorisation par instance.</p>	<p>Peut être désactivée, par exemple pour les tables composites dont la table principale contient des données de modèle. Dans ce cas, il n'y a pas de contrôle d'autorisation et les contraintes de sécurité sont passées outre. Cela signifie que chaque utilisateur authentifié peut utiliser la table de requêtes pour extraire des données, qu'il soit ou non autorisé à accéder aux objets correspondants.</p> <p>Les tables composites dont la table principale contient des données d'instance ne doivent pas être configurées pour utiliser l'autorisation par rôle.</p>	Tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requêtes, après application des filtres.
Supplémentaire	Les tables de requêtes supplémentaires ne doivent pas être configurées pour utiliser l'autorisation par instance, car Business Process Choreographer ne gère pas ces tables lui-même et il ne dispose donc pas d'informations sur les autorisations d'accès à leur contenu.	Les tables de requêtes supplémentaires ne doivent pas être configurées pour utiliser l'autorisation par rôle.	Tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requêtes, après application des filtres.

La figure suivante offre une vue d'ensemble des options disponibles pour les types d'autorisation, en fonction du type de table de requêtes. Elle illustre aussi les différents comportements obtenus avec l'API de table de requêtes et ses options d'autorisation.

Composite query table	primary query table with instance data	all	primary query table with template data
Predefined query tables	instance data	n/a	template data
Supplemental query tables	n/a	business data	n/a
Authorization	Instance-based authorization	None	Role-based authorization
Query with AuthorizationOptions	(A) Query result contains objects with work items related to the caller.	(C) Query result contains all objects that are in this query table.	n/a
Query with AdminAuthorizationOptions*	(B) Query result contains all objects that are in this query table.	(C) Query result contains all objects that are in this query table.	(D) Query result contains all objects that are in this query table.

Figure 28. Autorisation par instance pour les tables de requêtes

*) Si le paramètre onBehalfUser est défini explicitement, le comportement (A) s'applique

L'autorisation par instance utilisant les éléments de travail et s'exerçant sur les objets renvoyés dans les résultats d'une requête dépend du paramètre d'autorisation qui est passé à l'API de table de requêtes, mais aussi de la valeur (true ou false) de l'option Autorisation par instance spécifiée dans la définition de la table de requêtes interrogée.

- (A) Les requêtes ciblant des tables prédéfinies ou composites et utilisant l'objet AuthorizationOptions renvoient les entités qui concordent avec un élément de travail approprié à l'utilisateur désigné. C'est également le cas si l'objet AdminAuthorizationOptions est utilisé et si le paramètre onBehalfUser est défini avec une valeur explicite (l'ID d'un utilisateur). Les clients standard, qui

présentent des listes de tâches ou de processus aux utilisateurs, emploient généralement cette combinaison de types de table de requêtes et de paramètres d'API.

- (B) Le contenu intégral d'une table de requêtes est constitué des entités qui ont un élément de travail correspondant, tel que configuré avec l'autorisation par instance de la table en question. L'autorisation par instance considère quatre types d'éléments de travail : 'everybody', 'individual', 'group' et 'inherited'. L'appelant doit faire partie du rôle J2EE BPESystemAdministrator. Cette combinaison de types de tables de requêtes et de paramètres d'API est destinée aux scénarios d'administration, dans lesquels la liste complète des tâches ou processus disponibles doit pouvoir être affichée ou parcourue.
- (C) Les requêtes ciblant des tables de requêtes qui n'utilisent pas l'autorisation par instance ou par rôle renvoient les mêmes résultats que si l'objet AdminAuthorizationOptions ou AuthorizationOptions était passé à l'API de table de requêtes. Ce comportement est disponible pour les tables de requêtes supplémentaires ainsi que pour les tables composites. Aucune vérification des éléments de travail ou des rôles J2EE n'est effectuée ; par conséquent, tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table interrogée. Les clients qui ne veulent pas restreindre la visibilité des objets en appliquant les contraintes d'autorisation par instance ou par rôle fournies par Business Process Choreographer peuvent désactiver les vérifications d'autorisation lors du développement des tables de requêtes. En revanche, pour les opérations de réclamation et d'achèvement de tâches, les utilisateurs doivent avoir un élément de travail approprié.
- (D) Seule l'autorisation par rôle permet d'accéder aux données de modèle dans les tables de requêtes prédéfinies ou dans les tables composites configurées avec l'autorisation par rôle. L'appelant doit faire partie du rôle J2EE BPESystemAdministrator. L'API de table de requêtes peut être utilisée à la place de l'API de requête pour accéder aux données de modèle.

Éléments de travail et autorisation par instance

Le mécanisme d'autorisation par instance fourni par Business Process Choreographer est basé sur des éléments de travail. Chaque élément de travail décrit qui a quels droits et sur quels objets. Cette information est accessible via la table de requêtes WORK_ITEM, si l'autorisation par instance est utilisée.

Le tableau suivant décrit les différents types d'éléments de travail pris en considération si l'autorisation par instance est utilisée lors de l'exécution d'une requête sur une table de requêtes :

Tableau 16. Types d'éléments de travail

Type d'élément de travail	Description
everybody	Les éléments de travail du type 'everybody' autorisent tous les utilisateurs à accéder à un objet spécifique tel qu'une tâche ou une instance de processus. Dans ce cas, l'attribut EVERYBODY de l'élément de travail concerné est mis à TRUE.
individual	Les éléments de travail du type 'individual' sont créés pour des utilisateurs particuliers. L'ID d'un utilisateur spécifique est affecté comme valeur à l'attribut OWNER_ID de l'élément de travail concerné. Pour un même objet (tel qu'une tâche), il peut exister plusieurs éléments de travail, chacun avec un attribut OWNER_ID différent.

Tableau 16. Types d'éléments de travail (suite)

Type d'élément de travail	Description
group	Les éléments de travail du type 'group' sont créés pour les utilisateurs d'un groupe particulier. Le nom d'un groupe spécifique est affecté comme valeur à l'attribut GROUP_NAME de l'élément de travail concerné.
inherited	Les lecteurs et les administrateurs des instances de processus sont également autorisés à hériter de l'accès aux tâches humaines (y compris celles pour lesquelles il y a escalade) qui appartiennent à ces instances de processus. Dans les requêtes sur les tâches, les vérifications concernant les éléments de travail hérités sont exécutées au moyen de jointures SQL complexes ; ces jointures sont réalisées à l'exécution et ont une incidence sur les performances.

Les éléments de travail sont créés par Business Process Choreographer dans différents cas de figure. Par exemple, à la création d'une tâche, des éléments de travail sont créés pour les différents rôles, tels que ceux de lecteur et de propriétaire potentiel, si des critères d'affectation de personnes ont été spécifiés.

Le tableau suivant décrit les différents types d'éléments de travail créés en fonction des critères d'affectation de personnes définis, si l'autorisation par instance est utilisée lors de l'exécution d'une requête sur une table de requêtes. Les éléments de travail hérités n'apparaissent pas dans ce tableau, car ils reflètent des relations qui ne peuvent être modélisées explicitement dans la phase de développement.

Tableau 17. Eléments de travail et critères d'affectation de personnes

Type d'élément de travail	Critères d'affectation de personnes associés
everybody	Tous les utilisateurs
individual	Tous les critères d'affectation de personnes excepté les instructions <i>Personne</i> (Nobody), <i>Tous les utilisateurs</i> (Everybody) et <i>Groupe</i> (Group)
group	Groupe

Filtre d'autorisation sur les tables de requêtes composites

Pour les tables de requêtes composites, un filtre d'autorisation peut être spécifié si l'autorisation par instance est utilisée. Ce filtre limite les éléments de travail utilisables pour l'autorisation en fonction de certains de leurs attributs. Par exemple, le filtre d'autorisation "WI.REASON=REASON_POTENTIAL_OWNER" appliqué à une table de requêtes composite dont la table principale est TASK limite à certains types les tâches qui peuvent être renvoyées lorsqu'une personne exécute une requête. Seules sont renvoyées les tâches que cette personne peut avoir à effectuer ; autrement dit, le résultat est limité aux tâches que la personne est autorisée à réclamer. Ce filtre peut aussi être spécifié comme filtre de table de requêtes ou comme filtre de requête. Il a aussi un effet bénéfique sur les performances des requêtes lorsqu'il est spécifié comme filtre d'autorisation.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Options d'autorisation pour l'API de table de requêtes

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requêtes.

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Pour définir le type d'un attribut dans une table de requêtes, on a recours à un sous-ensemble des types caractéristiques du langage de programmation Java et des types propres aux technologies de base de données. Les types d'attribut sont une abstraction des types Java concrets ou des types de données propres aux bases de données. Pour les tables de requêtes supplémentaires, vous devez utiliser un mappage valide entre types de base de données et types d'attribut.

Le tableau suivant décrit les types d'attribut :

Tableau 18. Types d'attribut

Type d'attribut	Description
ID	L'ID servant à identifier une tâche humaine (TKIID), une instance de processus (PIID) ou d'autres objets. Par exemple, pour réclamer ou effectuer une tâche humaine particulière, l'utilisateur doit spécifier celle-ci en l'identifiant par son attribut TKIID.
STRING	Les descriptions des tâches ou les propriétés des requêtes peuvent être représentées par des chaînes.
NUMBER	Les nombres sont utilisés pour les attributs tels que le niveau de priorité d'une tâche.
TIMESTAMP	Les horodatages décrivent un instant précis ; par exemple, la date et l'heure de création d'une tâche humaine ou la date et l'heure de fin d'exécution d'une instance de processus.
DECIMAL	Le type DECIMAL peut être utilisé pour les propriétés d'une requête ; par exemple, pour définir une propriété de requête avec une variable du type XSD 'double'.
BOOLEAN	Les booléens peuvent prendre une valeur parmi deux : true ou false. Par exemple, les tâches humaines ont un attribut, autoClaim, qui précise si oui ou non une tâche est réclamée automatiquement lorsqu'il n'existe qu'un seul utilisateur qui puisse être son propriétaire.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Correspondance entre types des bases de données et types d'attribut

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Correspondance entre types d'attribut et représentations littérales

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et représentations littérales.

Correspondance entre types d'attribut et paramètres

Utilisez des types d'attribut lorsque vous définissez des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Correspondance entre types d'attribut et types d'objet Java

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et types d'objet Java.

Compatibilité entre types d'attribut

Utilisez des types d'attribut lorsque vous définissez des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Correspondance entre types des bases de données et types d'attribut :

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant décrit les types propres aux bases de données et indique leur correspondance avec les types d'attribut :

Tableau 19. Correspondance entre types des bases de données et types d'attribut

Type de base de données	Type d'attribut
Type binaire sur 16 octets. Il s'agit du type utilisé pour les ID tels que le TKIID des tâches dans les tables de Business Process Choreographer.	ID
Type caractère. Sa longueur dépend de la colonne de table de base de données qui est référencée par l'attribut dans la table de requêtes.	STRING

Tableau 19. Correspondance entre types des bases de données et types d'attribut (suite)

Type de base de données	Type d'attribut
Type d'entier de base de données, tel qu'un 'integer', un 'short' ou un 'long'.	NUMBER
Type d'horodatage de base de données.	TIMESTAMP
Type décimal, tel qu'un 'float' ou un 'double'.	DECIMAL
Type convertible en valeur booléenne telle qu'un nombre. 1 est interprété comme <i>true</i> (vrai) et tous les autres nombres, comme <i>false</i> (faux).	BOOLEAN

Exemple :

Prenons le cas d'un environnement DB2. Une table appelée CUSTOM.ADDITIONAL_INFO doit être représentée en tant que table de requêtes supplémentaire dans Business Process Choreographer. L'instruction SQL suivante crée la table de base de données :

```
CREATE TABLE CUSTOM.ADDITIONAL_INFO
(
  PIID      CHAR(16) FOR BIT DATA,
  INFO      VARCHAR(220),
  COUNT     INTEGER
);
```

La correspondance suivante, entre les types des colonnes côté base de données et les types d'attribut côté table de requêtes, est utilisé pour définir une table de requêtes supplémentaire chargée de représenter la table CUSTOM.ADDITIONAL_INFO.

Tableau 20. Exemple de mappage entre types de base de données et types d'attribut

Colonne et type de base de données	Attribut et type de table de requête
PIID CHAR(16) FOR BIT DATA	PIID (ID)
INFO VARCHAR(220)	INFO (STRING)
COUNT INTEGER	COUNT (NUMBER)

Généralement, les tables de requêtes supplémentaires font référence à des tables et des vues existantes de la base de données ; lorsque c'est le cas, elles ne nécessitent pas la création de tables ou de vues spécifiques.

Concepts associés

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Correspondance entre types d'attribut et représentations littérales :

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et représentations littérales.

Des valeurs littérales peuvent être utilisées dans les expressions pour définir des critères de filtrage et de sélection, comme dans les filtres des tables de requêtes composites ainsi que dans les filtres passés à l'API de table de requêtes.

Le tableau suivant décrit les types d'attribut et indique leur correspondance avec les valeurs littérales. Les marques de réservation figurent en *italique*. Notez que les types d'attribut ID et TIMESTAMP, qui peuvent être passés à l'API de table de requêtes, utilisent une syntaxe spéciale, également employée par l'API de requête.

Tableau 21. Correspondance entre types d'attribut et valeurs littérales

Type d'attribut	Syntaxe et utilisation comme valeur littérale dans les expressions
ID	ID ('représentation chaîne d'un ID') Lors du développement d'applications clientes, les ID sont représentés soit comme des chaînes, soit comme des instances de l'interface com.ibm.bpe.api.OID. Il est possible d'obtenir la représentation sous forme de chaîne d'une instance de l'interface com.ibm.bpe.api.OID en utilisant sa méthode toString. La chaîne doit être encadrée d'apostrophes.
STRING	'la chaîne' La chaîne doit être encadrée d'apostrophes.
NUMBER	nombre Le nombre sous forme de texte, non délimité par des apostrophes. Des constantes sont définies et utilisables pour certains attributs du type NUMBER dans les tables de requêtes prédéfinies.
TIMESTAMP	TS ('AAAA-MM-JJThh:mm:ss ') L'horodatage doit être spécifié au format suivant : <ul style="list-style-type: none"> • AAAA : les quatre chiffres de l'année • MM : les deux chiffres du mois de l'année • DD : les deux chiffres du jour du mois • hh : les deux chiffres de l'heure (sur 24 heures) • mm : les deux chiffres de la minute • ss : les deux chiffres de la seconde. L'interprétation de l'horodatage se fait par rapport au fuseau horaire de l'utilisateur.
DECIMAL	nombre.fraction Le nombre décimal sous forme de texte, non délimité par des apostrophes ; la partie .fraction est optionnelle.
BOOLEAN	true, false La valeur booléenne sous forme de texte.

Exemples :

- filterOptions.setQueryCondition("STATE=2");
- filterOptions.setQueryCondition("STATE=STATE_READY");
- un critère de sélection sur une table de requêtes attachée TASK_DESC :
"LOCALE='en_US'"
- filterOptions.setQueryCondition("PTID=ID('_PT:8001011e.1dee8e51.247d6df6.29a60000')");

Concepts associés

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Correspondance entre types d'attribut et paramètres :

Utilisez des types d'attribut lorsque vous définissez des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant décrit les types d'attribut et leur correspondance avec les valeurs de paramètre qui peuvent être utilisées dans les expressions pour définir des critères de filtrage et de sélection, comme dans les filtres des tables de requêtes composites ainsi que dans les filtres passés à l'API de table de requêtes.

Tableau 22. Correspondance entre types d'attribut et valeurs de paramètre utilisateur

Type d'attribut	Utilisation comme valeur de paramètre dans les expressions
ID	PARAM(<i>nom</i>) Lors du développement d'applications clientes, les ID sont représentés soit comme des chaînes, soit comme des instances de l'interface com.ibm.bpe.api.OID. Les deux représentations sont valides et acceptées comme paramètres. Un tableau d'octets (byte[]) reflétant un OID valide peut également être utilisé.
STRING	PARAM(<i>nom</i>) La représentation chaîne de l'objet qui, à l'exécution, est passée à l'API de table de requêtes par la méthode toString.
NUMBER	PARAM(<i>nom</i>) Une représentation du nombre sous forme de java.lang.Long, java.lang.Integer, java.lang.Short ou java.lang.String est passée à l'API de table de requêtes. Les noms des constantes définies pour certains attributs des tables de requêtes prédéfinies peuvent aussi être passés.
TIMESTAMP	PARAM(<i>nom</i>) Les représentations suivantes sont valides : <ul style="list-style-type: none">• Une représentation java.lang.String de l'horodatage• Instances de com.ibm.bpe.api.UTCDate• Instances de java.util.Calendar
DECIMAL	PARAM(<i>nom</i>) Une représentation du nombre décimal sous forme de java.lang.Long, java.lang.Integer, java.lang.Short, java.lang.Double, java.lang.Float ou java.lang.String est passée à l'API de table de requêtes.

Tableau 22. Correspondance entre types d'attribut et valeurs de paramètre utilisateur (suite)

Type d'attribut	Utilisation comme valeur de paramètre dans les expressions
BOOLEAN	PARAM(<i>nom</i>) Les valeurs admises sont : <ul style="list-style-type: none"> • Une représentation java.lang.String du booléen • Un java.lang.Short, java.lang.Integer ou java.lang.Long avec des valeurs appropriées ; 0 (pour false) ou 1 (pour true) • Un objet java.lang.Boolean

Exemple :

```

...
// l'exemple suivant illustre l'exécution d'une requête sur une table de
// requêtes composite, COMP.TASKS, avec un paramètre "client"
java.util.List params = new java.util.ArrayList();

list.add(new com.ibm.bpe.api.Parameter("client", "IBM");
bfm.queryEntities("COMP.TASKS", null, null, params);
...

```

Concepts associés

Types d'attribut pour les tables de requêtes
 Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Correspondance entre types d'attribut et types d'objet Java :

Utilisez des types d'attribut pour définir des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et types d'objet Java.

Le tableau suivant décrit les types d'attribut et indique leur correspondance avec les types d'objet Java dans les ensembles de résultats des requêtes.

Tableau 23. Correspondance entre types d'attribut et types d'objet Java

Type d'attribut	Type d'objet Java associé
ID	com.ibm.bpe.api.OID
STRING	java.lang.String
NUMBER	java.lang.Long
TIMESTAMP	java.util.Calendar
DECIMAL	java.lang.Double
BOOLEAN	java.lang.Boolean

Exemple :

```

...
// l'exemple suivant illustre l'exécution d'une requête sur une table de
// requêtes composite appelée COMP.TA ; l'attribut "STATE" est du type d'attribut
NUMBER
...

```

```

// exécution de la requête
EntityResultSet rs = bfm.queryEntities("COMP.TA",null,null,params);

// obtenir les entités et itérer dessus
List entities = rs.getEntities();
for (int i = 0 ; i < entities.size(); i++) {

    // manipuler une entité particulière
    Entity en = (Entity) entities.get(i);

    // notez que le code suivant pourrait être écrit d'une
    // façon moins spécifique en utilisant les objets d'informations
    // d'attributs renvoyés par la méthode ei.getAttributeInfo()

    // obtenir l'attribut STATE
    Long state = (Long) en.getAttributeValue("STATE");
    ...
}
...

```

Concepts associés

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Compatibilité entre types d'attribut :

Utilisez des types d'attribut lorsque vous définissez des tables de requêtes dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant dresse la liste des types d'attribut et indique leur compatibilité. Utilisez ces informations pour définir des filtres et des critères de sélection dans les tables de requêtes. La compatibilité entre types d'attribut est signalée par un X.

Tableau 24. Compatibilité entre types d'attribut

Type d'attribut	ID	STRING	NUMBER	TIMESTAMP	DECIMAL	BOOLEAN
ID	X					
STRING		X				
NUMBER			X		X	
TIMESTAMP				X		
DECIMAL			X		X	
BOOLEAN						X

Dans les expressions de table de requêtes qui spécifient des filtres et des critères de condition, les types des attributs ou des valeurs comparés doivent être compatibles. Par exemple, WI.OWNER_ID=1 est un filtre non valide, car l'opérande côté gauche est du type STRING alors que l'opérande côté droit est du type NUMBER.

Concepts associés

Types d'attribut pour les tables de requêtes

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requêtes, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Une requête s'exécute toujours sur une seule table de requêtes. Le contenu des tables de requêtes est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes. Des paramètres (ou arguments) d'entrée sont passés aux méthodes de l'API de table de requêtes.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Méthodes de l'API de table de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes. Le contenu des tables de requêtes est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes.

Paramètres de l'API de table de requêtes

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Résultats des requêtes exécutées sur les tables de requêtes

Les méthodes de l'API de table de requêtes s'utilisent lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer. Le résultat de la méthode `queryEntityCount` ou de la méthode `queryRowCount` est un simple nombre. En revanche, les méthodes `queryEntities` et `queryRows` renvoient des ensembles de résultats.

Tables de requêtes prédéfinies

Les tables de requêtes prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requêtes, des vues de la base de données de Business Process Choreographer, telles que `TASK` ou `PROCESS_INSTANCE`.

Tables de requêtes supplémentaires

Les tables de requêtes supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requêtes un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requêtes supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requêtes prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches humaines.

Tables de requêtes composites

Dans Business Process Choreographer, les tables de requêtes composites sont conçues à partir de tables de requêtes prédéfinies et de tables de requêtes supplémentaires. Elles regroupent des données issues de tables ou de vues existantes. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Développement des tables de requêtes

Dans Business Process Choreographer, le développement des tables de requêtes supplémentaires et des tables de requêtes composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requêtes prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process

Choreographer.

Filtres et critères de sélection des tables de requêtes

Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requêtes.

Méthodes de l'API de table de requêtes :

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes. Le contenu des tables de requêtes est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes.

Les méthodes suivantes sont disponibles dans l'API de table de requêtes pour permettre l'exécution de requêtes sur les tables de requêtes dans Business Process Choreographer :

Tableau 25. Méthodes pour les requêtes exécutées sur les tables de requêtes

Fonction	Méthodes
Interrogation de contenu	<ul style="list-style-type: none">• queryEntities• queryRows <p>Les deux méthodes renvoient du contenu de la table de requêtes. La méthode queryEntities renvoie un contenu basé sur des entités et la méthode queryRows, un contenu basé sur des lignes.</p>
Interrogation du nombre d'objets	<ul style="list-style-type: none">• queryEntityCount• queryRowCount <p>Les deux méthodes renvoient le nombre d'objets présents dans la table de requêtes, mais ce nombre peut varier en fonction de l'approche adoptée (requête par entité ou par ligne).</p>

Les requêtes par entité, exécutées à l'aide des méthodes queryEntities et queryEntityCount, supposent qu'une table de requêtes contient des entités identifiables de façon unique, tel que défini par la clé primaire dans la table de requêtes principale.

Les requêtes par ligne, exécutées à l'aide des méthodes queryRows et queryRowCount, renvoient un ensemble de résultats comme avec JDBC ; cet ensemble de résultats est également à base de lignes et navigable au moyen des méthodes first et next. L'ensemble de résultats obtenu lorsque vous exécutez une requête sur une table de requêtes en utilisant l'API de table de requêtes est comparable aux ensembles QueryResultSet renvoyés par l'API de requêtes. En général, le nombre de lignes est supérieur au nombre d'entités contenues dans une table de requêtes. Une même entité, par exemple une tâche humaine identifiée par son ID de tâche TKIID, peut apparaître plusieurs fois dans l'ensemble de résultats de lignes.

Une instance particulière contenue dans une table de requêtes prédéfinie n'existe qu'en un seul exemplaire dans un environnement Business Process Choreographer. Les tâches humaines et les processus métier sont des exemples de telles instances. Chaque instance est identifiée de manière unique par un ID ou un ensemble d'ID. Il s'agit du TKIID pour les instances de tâche humaine et du PIID pour les instances de processus.

Les tables de requêtes composites comprennent une table de requêtes principale et zéro, une ou plusieurs tables de requêtes attachées. L'identification des objets dans une table composite se fait par les ID unique des objets de la table de requêtes principale. C'est donc la table de requêtes principale qui détermine le type d'entité d'une table composite. Par exemple, une table de requêtes composite dont la table principale est TASK contiendra obligatoirement des entités du type TASK. La relation un à un ou un à zéro qui existe entre la table principale et les tables attachées garantit que les tables attachées n'introduisent pas d'entités en double.

Les requêtes par entité tirent parti du fait qu'une table de requêtes contient des entités identifiables de façon unique, tel que défini par la clé primaire dans la table de requêtes principale. La garantie d'unicité des instances et d'absence de doublons est une qualité particulièrement appréciée par les développeurs d'applications clientes, notamment ceux qui sont chargés de la partie interface utilisateur. Par exemple, il est essentiel qu'une tâche humaine ne soit affichée qu'une seule fois dans l'interface utilisateur. Des instances uniques sont renvoyées si l'API de table de requêtes par entité est utilisée.

Les requêtes par ligne peuvent renvoyer des doubles des lignes de la table de requêtes principale si l'autorisation par instance est utilisée.

- Les informations de la table de requêtes WORK_ITEM sont récupérées avec la requête. Par exemple, si l'attribut WI.REASON est extrait en plus des attributs définis dans la table de requêtes, plusieurs lignes sont susceptibles d'être renvoyées comme résultat. En effet, cet attribut stocke le motif d'accès à une entité telle qu'une tâche ou une instance de processus ; or, un utilisateur peut accéder à une telle entité pour plusieurs raisons.
- L'autorisation par instance est utilisée et l'opérateur 'distinct' n'est pas spécifié. Bien qu'il n'y ait pas d'extraction des données d'élément de travail, plusieurs lignes peuvent être renvoyées si l'autorisation par instance est utilisée.

Si l'API de table de requêtes par entité est utilisée :

- Les requêtes par entité sont toujours exécutées avec l'opérateur SQL 'distinct'.
- Les requêtes par entité renvoient des résultats dans lesquels les informations relatives aux éléments de travail peuvent être fournies sous forme de valeurs de tableau (array).

Concepts associés

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Paramètres de l'API de table de requêtes :

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Les paramètres d'entrée suivants sont passés aux méthodes de l'API de table de requêtes :

Tableau 26. Paramètres de l'API de table de requêtes

Paramètre	Optionnel	Type et description
Nom de la table de requêtes	Non	java.lang.String
		Nom unique de la table de requêtes.
Options de filtrage	Oui	com.ibm.bpe.api.FilterOptions
		Options utilisables pour définir la requête. Par exemple, un seuil peut être fixé dans ce paramètre pour limiter le nombre de résultats renvoyés.
Options d'autorisation	Oui	com.ibm.bpe.api.AuthorizationOptions ou com.ibm.bpe.api.AdminAuthorizationOptions
		Si l'autorisation par instance est utilisée, ce paramètre permet d'imposer des contraintes encore plus serrées. Pour les tables de requêtes nécessitant une autorisation par rôle, une instance d'objet AdminAuthorizationOptions doit être passée.
Paramètres	Oui	Une liste (java.util.List) d'objets com.ibm.bpe.api.Parameter
		Ce paramètre sert à passer des paramètres utilisateur qui sont spécifiés dans un critère de filtrage ou de sélection sur une table de requêtes composite.

Une requête s'exécute toujours sur une seule table de requêtes. La relation entre plusieurs tables de requêtes est définie au moyen d'une table de requêtes composite. Ce concept particulier à l'API de table de requêtes correspond à la notion de vues de base de données dans l'API de requête.

Les filtres et les critères de sélection sont spécifiés dans des expressions pendant la phase de développement des tables de requêtes, à l'aide de l'outil Query Table Builder. Pour plus de détails à ce propos, consultez la rubrique du centre de documentation relative aux tables de requêtes composites, ainsi que la rubrique consacrée aux filtres et aux critères de recherche des tables de requêtes. Pour toute information sur l'outil Query Table Builder, consultez le site des SupportPacs de WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Concepts associés

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Nom de la table de requêtes

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, le nom de cette table est passé comme paramètre d'entrée aux méthodes de l'API de table de requêtes.

Options de filtrage

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des options de filtrage en guise de paramètres d'entrée aux méthodes de l'API de table de requêtes.

Options d'autorisation pour l'API de table de requêtes

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requêtes.

Paramètres

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des paramètres utilisateur aux méthodes de l'API de table de requêtes. Dans une définition de table de requêtes, vous pouvez spécifier des paramètres dans les filtres appliqués à la table de requête principale, aux autorisations et à la table de requête elle-même. Des paramètres peuvent aussi être spécifiés dans les critères de sélection sur les tables de requêtes attachées.

Nom de la table de requêtes :

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, le nom de cette table est passé comme paramètre d'entrée aux méthodes de l'API de table de requêtes.

Le nom de la table de requêtes dont il est question ici est celui de la table constituant la cible de l'exécution de la requête.

- Lorsque la cible est une table de requêtes prédéfinie, le nom spécifié est donc celui de la table prédéfinie en question.
- Dans le cas d'une table composite ou supplémentaire, il s'agit du nom de la table de requêtes qui a été spécifié lors de la modélisation de la table. Le nom d'une table composite ou supplémentaire est de la forme *préfixe.nom* ; la partie *préfixe* ne doit pas être 'IBM'.

Le nom de la table de requêtes et le préfixe doivent tous les deux être en lettres majuscules. Le nom de la table de requêtes ne doit pas dépasser 28 caractères.

Concepts associés

Paramètres de l'API de table de requêtes

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Options de filtrage :

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des options de filtrage en guise de paramètres d'entrée aux méthodes de l'API de table de requêtes.

Une instance de la classe **com.ibm.bpe.api.FilterOptions** peut être passée à l'API de table de requêtes. Les options de filtrage permettent de configurer une requête en agissant sur les caractéristiques suivantes :

- Un seuil et un décalage (skipCount)
- Des attributs de tri (similaires à la clause ORDER BY dans une requête SQL)
- Un filtre de requête additionnel
- L'ensemble d'attributs renvoyé, y compris les données d'élément de travail
- Autre

L'ensemble de résultats qui peut être obtenu d'une table de requêtes est spécifié par la définition de cette table. Cependant, vous pouvez avoir besoin de spécifier des options supplémentaires au moment d'exécuter la requête. Le tableau suivant décrit les options de filtrage qui peuvent être spécifiées au moyen de l'objet **com.ibm.bpe.api.FilterOptions**.

Tableau 27. Paramètres de l'API de table de requêtes : options de filtrage

Option	Type	Description
Attributs sélectionnés	java.lang.String	<ul style="list-style-type: none"> • Une liste (séparée par des virgules) des attributs de la table de requêtes à renvoyer dans l'ensemble de résultats. • Si l'autorisation par instance est utilisée, vous pouvez extraire les données d'élément de travail en spécifiant les attributs de la table de requêtes WORK_ITEM, préfixés avec 'WI.'. Par exemple, WI.REASON. • Si 'null' est spécifié, tous les attributs de la table de requêtes sont renvoyés, sans les données d'élément de travail.
Filtre de requête	java.lang.String	Le filtre de requête, qui complète l'action des filtres et des critères de sélection déjà définis pour la table de requête.
Attributs de tri	java.lang.String	Une liste (séparée par des virgules) des attributs de la table de requêtes éventuellement suivis de ASC ou DESC pour spécifier un tri par ordre croissant ou décroissant, respectivement. Cette liste est similaire à la clause SQL ORDER BY : <i>sortAttributes ::= attribut [ASC DESC] [, sortAttributes]</i> . Si ASC ou DESC n'est pas spécifié, l'ordre de tri croissant (ASC) est appliqué par défaut. L'ordre dans lequel les attributs de tri sont spécifiés a son importance. Ainsi, dans cet exemple, les tâches de la table de requêtes TASK sont d'abord triées par état et dans l'ordre décroissant (DESC), puis par nom et dans l'ordre croissant (ASC) au sein de chaque groupe de tâches ayant le même état : "STATE DESC, NAME ASC".

Tableau 27. Paramètres de l'API de table de requêtes : options de filtrage (suite)

Option	Type	Description
Seuil	java.lang.Integer	<p>Définit une limite qui peut être :</p> <ul style="list-style-type: none"> Le nombre maximum de lignes renvoyées si queryRows est utilisée. Le nombre maximum d'entités renvoyées si queryEntities est utilisée. Le nombre réel d'entités disponibles dans la table de requêtes concernée peut dépasser la limite définie pour la requête, même si l'ensemble de résultats ne contient pas autant d'entités que le nombre correspondant à cette limite. Cela est dû à des impératifs techniques qui entrent en jeu lorsque les données d'élément de travail sont sélectionnées. Le compte renvoyé si queryRowCount ou queryEntityCount est utilisée. <p>La valeur par défaut est 'null', ce qui signifie qu'aucun seuil n'est défini.</p>
Nombre d'éléments à sauter (skipCount)	java.lang.Integer	<p>Définit le nombre de lignes (requêtes par ligne) ou le nombre d'entités (requêtes par entité) à sauter. Comme pour le paramètre de seuil, skipCount peut manquer d'exactitude dans le cas des requêtes par entité.</p> <p>Le rôle du paramètre skipCount est de permettre la pagination lorsque de gros ensembles de résultats sont renvoyés. La valeur par défaut est 'null', ce qui signifie que skipCount n'est pas défini.</p>
Fuseau horaire	java.util.TimeZone	<p>Le fuseau horaire utilisé lors de la conversion des horodatages. L'attribut CREATED de la table requêtes prédéfinie TASK est un exemple d'horodatage affecté par ce paramètre. S'il n'est pas spécifié (null), le fuseau horaire utilisé est celui du serveur.</p>
Environnement local	java.util.Locale	<p>L'environnement local utilisé pour calculer la valeur du paramètre système \$LOCALE. Exemple d'utilisation de \$LOCALE dans un critère de sélection : 'LOCALE=\$LOCALE'.</p>
Lignes distinctes	java.lang.Boolean	<p>S'applique uniquement aux requêtes par ligne. Si la valeur est true, les requêtes par ligne renvoient des lignes distinctes. Cela n'implique pas que des lignes uniques sont renvoyées en raison de l'éventuelle multiplicité des données d'élément de travail.</p>

Tableau 27. Paramètres de l'API de table de requêtes : options de filtrage (suite)

Option	Type	Description
Condition de requête	setQueryCondition	Applique un filtrage supplémentaire à l'ensemble de résultats. Les attributs définis dans la table de requêtes peuvent être référencés si une autorisation est requise. Les colonnes définies dans la table de requêtes WORK_ITEM peuvent également être référencées à l'aide du préfixe 'WI.' Exemple : WI.REASON=REASON_POTENTIAL_OWNER.

Concepts associés

Paramètres de l'API de table de requêtes

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Options d'autorisation pour l'API de table de requêtes :

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requêtes.

Utilisez une instance de la classe `com.ibm.bpe.api.AuthorizationOptions` ou de la classe `com.ibm.bpe.api.AdminAuthorizationOptions` pour spécifier des options d'autorisation supplémentaires lors de l'exécution de la requête.

Si l'autorisation par instance est utilisée, les instances de la classe `com.ibm.bpe.api.AuthorizationOptions` permettent de spécifier les types d'éléments de travail identifiant les instances qui peuvent être renvoyées par la requête.

Une instance de la classe `com.ibm.bpe.api.AuthorizationOptions` peut être passée à l'API de table de requêtes si la requête est exécutée sur une table de requête prédéfinie qui contient des données d'instance. Elle peut aussi être passée à l'API si la requête est exécutée sur une table composite dont la table principale contient des données d'instance et si l'autorisation par instance est configurée pour être utilisée. Si la requête est exécutée sur une table prédéfinie avec des données de modèle ou sur une table composite avec l'autorisation par rôle configurée, une exception `EngineNotAuthorizedException` est émise. Dans tous les autres cas, les options d'autorisation passées à l'API de table de requêtes sont ignorées.

Les tables de requêtes composites peuvent restreindre les types d'éléments de travail pris en compte lors de l'identification des objets (ou entités) qu'elles contiennent. Par exemple, si les options d'autorisation passées à l'API de table de requêtes sont configurées pour utiliser les éléments de travail 'everybody', elles ne sont prises en compte que si l'utilisation de tels éléments de travail est prévue dans la définition de la table de requêtes composite. Voici une règle simple à retenir : l'API de table de requêtes ne peut pas forcer la prise en compte d'un type d'élément de travail dont l'utilisation n'est pas spécifiée dans la définition de la table de requêtes ; en revanche, l'API n'est pas forcée de prendre en compte un type d'élément de travail dont l'utilisation est spécifiée dans la définition de la table de requêtes. De même, l'API de table de requêtes ne peut pas spécifier un type d'autorisation différent de celui qui est défini pour une table de requêtes composite ou prédéfinie.

En fonction du type de la table de requêtes interrogée, différentes options d'autorisation par défaut s'appliquent si l'objet d'autorisation n'est pas spécifié ou si, comme c'est le cas par défaut, les attributs associés (everybody, individual, group ou inherited) sont mis à 'null'.

Le tableau suivant indique les options d'autorisation appliquées par défaut à l'autorisation par instance, pour chaque combinaison de type de table de requête et de type d'élément de travail.

Tableau 28. Paramètres de l'API de table de requêtes : options d'autorisation par défaut pour l'autorisation par instance

Type de table de requêtes	Élément de travail 'everybody'	Élément de travail 'individual'	Élément de travail 'group'	Élément de travail 'inherited'
Prédéfinie avec des données d'instance	TRUE	TRUE	TRUE	FALSE
Prédéfinie avec des données de modèle	N/A	N/A	N/A	N/A
Composite avec une table principale contenant des données d'instance	TRUE	TRUE	TRUE	TRUE
Composite avec une table principale contenant des données de modèle	N/A	N/A	N/A	N/A
Supplémentaire	N/A	N/A	N/A	N/A

N/A (non applicable) signifie que l'autorisation par instance n'est pas utilisée et que, par conséquent, toute option concernant les éléments de travail dans l'objet d'autorisation est ignorée.

Si TRUE est spécifié, la requête résultante ne prend en compte le type d'élément de travail concerné que si la table de requêtes est définie pour utiliser ce type d'élément de travail. Cette règle est valable pour toutes les tables de requêtes prédéfinies qui contiennent des données d'instance ; elle peut ne pas l'être pour une table de requêtes composite. En ce qui concerne l'élément de travail 'group', celui-ci doit être activé sur le conteneur de tâches humaines. Un exemple de cas où l'élément de travail est réglé à TRUE est celui où l'administrateur d'une instance de processus peut voir les instances de tâches humaines participantes qui sont créées pour cette instance de processus.

Spécifiez une instance de la classe `com.ibm.bpe.api.AdminAuthorizationOptions` à la place d'une instance de `com.ibm.bpe.api.AuthorizationOptions` dans les cas suivants :

- Une requête est exécutée sur une table de requêtes configurée avec l'autorisation par rôle. Les tables de requêtes prédéfinies avec des données de modèle

requièrent l'autorisation par rôle ; de même, les tables composites dont la table principale contient des données de modèle peuvent être configurées pour exiger l'autorisation par rôle.

- Une requête est exécutée sur une table de requêtes contenant des données d'instance ou sur une table composite dont la table principale contient des données d'instance. Le contenu de cette table de requêtes doit être renvoyé, sans considération des restrictions liées à l'autorisation d'un utilisateur particulier. Ce comportement particulier à l'API de table de requêtes correspond à l'utilisation de la méthode queryAll dans l'API de requête.
- Une requête doit être exécutée pour le compte d'un autre utilisateur.

Le tableau suivant décrit de quelle manière les comportements décrits plus haut sont obtenus :

Tableau 29. Paramètres de l'API de table de requêtes : AdminAuthorizationOptions

Situation	Description
Valeur de onBehalfUser = null	<ul style="list-style-type: none"> • Si une requête est exécutée sur une table de requêtes configurée avec l'autorisation par rôle, l'intégralité de son contenu est renvoyée. • Si la requête est exécutée sur une table de requêtes qui utilise l'autorisation par instance, les objets qu'elle contient ne sont pas vérifiés par rapport aux éléments de travail autorisés pour tel ou tel utilisateur. La requête renvoie donc tous les objets contenus dans la table de requêtes.
Valeur de onBehalfUser = un utilisateur particulier	La requête est exécutée avec l'autorité de l'utilisateur spécifié, et les objets contenus dans la table de requêtes cible sont vérifiés par rapport aux éléments de travail autorisés pour cet utilisateur, si l'autorisation par instance est utilisée.

Si vous spécifiez com.ibm.bpe.api.AdminAuthorizationOptions, l'appelant doit faire partie du rôle J2EE BPESystemAdministrator.

Concepts associés

Paramètres de l'API de table de requêtes

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Autorisation pour les tables de requêtes

L'exécution de requêtes sur des tables de requêtes peut s'effectuer avec plusieurs types d'autorisation : autorisation par instance, autorisation par rôle ou absence de contrôle d'autorisation.

Paramètres :

Lorsque vous exécutez une requête sur une table de requêtes dans Business Process Choreographer, vous pouvez passer des paramètres utilisateur aux méthodes de l'API de table de requêtes. Dans une définition de table de requêtes, vous pouvez spécifier des paramètres dans les filtres appliqués à la table de requête principale, aux autorisations et à la table de requête elle-même. Des paramètres peuvent aussi être spécifiés dans les critères de sélection sur les tables de requêtes attachées.

Les paramètres système \$USER et \$LOCALE sont remplacés à l'exécution dans les filtres et les critères de sélection et n'ont pas besoin d'être passés à l'API de table de requêtes. La valeur d'entrée utilisée pour le calcul du paramètre système \$LOCALE est fournie en définissant l'environnement local dans les options de filtre.

Les paramètres utilisateur doivent être passés à l'API de table de requêtes au moment où la requête est exécutée. Cette étape est réalisée en passant une liste d'instances de la classe com.ibm.bpe.api.Parameter.

Les propriétés suivantes doivent être spécifiées dans un objet Parameter :

Tableau 30. Paramètres utilisateur destinés à l'API de table de requêtes

Propriété	Description
Nom	Le nom du paramètre tel qu'il est spécifié dans la définition de table de requêtes. Le nom est sensible à la casse.
Valeur	La valeur du paramètre. Le type du paramètre doit être compatible avec le type de l'opérande côté gauche et de tous les filtres et critères de sélection où ce paramètre est utilisé. Les constantes définies pour certains attributs des tables de requêtes prédéfinies peuvent être passés comme des chaînes ; par exemple, STATE_READY.

Voici quelques exemples d'utilisation de paramètres :

```
// permet d'obtenir le contexte d'affectation de nom et de rechercher l'interface
// EJB home de Business Flow Manager ; cette interface
// doit être mise en cache pour des raisons de performances
// nous supposons également qu'il existe une référence EJB
// à l'EJB local de Business Flow Manager
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
(LocalBusinessFlowManagerHome)
ctx.lookup("java:comp/env/ejb/BFM");

// crée le module de remplacement de Business Flow Manager côté client
LocalBusinessFlowManager bfm = home.create();

// exécute une requête sur une table de requêtes composite
// CUST.CPM avec le filtre de table de requêtes principale
// 'STATE=PARAM(theState)'
EntityResultSet ers = null;
List parameterList = new ArrayList();
parameterList.add(new Parameter
("theState", new Integer(2)));

ers = bfm.queryEntities
("CUST.CPM", null, null, parameterList);

// manipuler l'ensemble de résultats
// ...
```

Concepts associés

Paramètres de l'API de table de requêtes

Les méthodes de l'API de table de requêtes permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer.

Résultats des requêtes exécutées sur les tables de requêtes :

Les méthodes de l'API de table de requêtes s'utilisent lors de l'exécution de requêtes sur une table de requêtes dans Business Process Choreographer. Le

résultat de la méthode `queryEntityCount` ou de la méthode `queryRowCount` est un simple nombre. En revanche, les méthodes `queryEntities` et `queryRows` renvoient des ensembles de résultats.

EntityResultSet

Une instance de la classe `com.ibm.bpe.api.EntityResultSet` est renvoyée par la méthode `queryEntities`. Un ensemble de résultats d'entités a les propriétés suivantes :

Tableau 31. Propriétés d'un ensemble de résultats d'entités renvoyé par l'API de table de requêtes

Propriété	Description
<code>queryTableName</code>	Nom de la table de requêtes sur laquelle la requête a été exécutée.
<code>entityTypeName</code>	<ul style="list-style-type: none"> • Si la requête a été exécutée sur une table de requêtes composite, cette propriété est le nom de la table de requêtes principale. • Si la requête a été exécutée sur une table de requêtes prédéfinie ou supplémentaire, cette propriété est le nom de la table en question ; autrement dit, elle a la même valeur que la propriété <code>queryTableName</code>.
<code>entityInfo</code>	Cette propriété contient les méta-informations des entités contenues dans l'ensemble de résultats. Une liste <code>java.util.List</code> d'objets <code>com.ibm.bpe.api.AttributeInfo</code> peut être extraite de cet objet. Cette liste contient les noms et les types d'attribut des informations contenues dans les entités de l'ensemble de résultats. Elle contient également les méta-informations relatives aux attributs qui constituent la clé de ces entités.
<code>entities</code>	Une liste <code>java.util.List</code> d'objets <code>Entity</code> .
<code>locale</code>	L'environnement local calculé pour le paramètre système <code>\$LOCALE</code> .

Les instances de la classe `com.ibm.bpe.api.Entity` contiennent les informations extraites de la requête lancée sur la table de requêtes. Une entité représente un objet identifiable de manière unique ; par exemple, une tâche, une instance de processus, une activité ou une escalade. Les propriétés suivantes sont disponibles pour les entités :

Tableau 32. Propriétés d'une entité renvoyée par l'API de table de requêtes

Propriété	Description
<code>entityInfo</code>	L'objet <code>entityInfo</code> qui est également contenu dans l'ensemble de résultats d'entités.
<code>attributeValue (attributeName)</code>	La valeur de l'attribut spécifié qui est extrait pour cette entité. Le type est contenu dans l'objet <code>AttributeInfo</code> correspondant de cet attribut.
<code>attributeValuesOfArray (attributeName)</code>	Un tableau de valeurs. Utilisez cette propriété si la valeur de la propriété <code>array</code> de l'objet <code>AttributeInfo</code> est 'true', ce qui est actuellement le cas uniquement si l'attribut fait référence à des données d'élément de travail.

Le nombre d'entités contenues dans l'ensemble de résultats est obtenu en exécutant la méthode `size()` de la liste d'entités.

Exemple : Requête utilisant l'API de requête par entité :

```

...
// L'exemple suivant illustre une requête exécutée sur la
// table prédéfinie TASK et utilisant l'API par entité

...
// exécution de la requête
EntityResultSet rs = bfm.queryEntities("TASK", null, null, null);

// obtenir les méta-informations des entités
EntityInfo ei = rs.getEntityInfo();
List atts = ei.getAttributeInfo();

// obtenir les entités et itérer dessus
Iterator entitiesIter = rs.getEntities().iterator();
while (entitiesIter.hasNext()) {

    // manipuler une entité particulière
    Entity en = (Entity) entitiesIter.next();

    for (int i = 0; i < atts.size(); i++) {
        AttributeInfo ai = (AttributeInfo) atts.get(i);
        Serializable value = en.getAttributeValue(ai.getName());

        // traiter...
    }
}
...

```

RowResultSet

Une instance de la classe `com.ibm.bpe.api.RowResultSet` est renvoyée par la méthode `queryRows`. Ce type d'ensemble de résultats est similaire à un ensemble de résultats JDBC. Un ensemble de résultats de lignes a les propriétés suivantes :

Tableau 33. Propriétés d'un ensemble de résultats de lignes renvoyé par l'API de table de requêtes

Propriété	Description
<code>primaryQueryTableName</code>	<ul style="list-style-type: none"> Si la requête a été exécutée sur une table de requêtes composite, cette propriété est le nom de la table de requêtes principale. Si la requête a été exécutée sur une table de requêtes prédéfinie ou supplémentaire, cette propriété est le nom de la table en question ; autrement dit, elle a la même valeur que la propriété <code>queryTableName</code>.
<code>attributeInfo</code>	Cette propriété contient la liste des objets <code>com.ibm.bpe.api.AttributeInfo</code> qui décrivent les méta-informations de cet ensemble de résultats. Chaque objet <code>AttributeInfo</code> contient le nom et le type d'un attribut. Il ne contient pas de métadonnées relatives aux clés, car les ensembles de résultats de lignes n'ont pas de clé.
<code>attributeValue</code>	La valeur de l'attribut spécifié qui a été extrait pour cette ligne. Le type est contenu dans l'objet <code>AttributeInfo</code> correspondant de cet attribut.

Tableau 33. Propriétés d'un ensemble de résultats de lignes renvoyé par l'API de table de requêtes (suite)

Propriété	Description
next, first, last, previous	La navigation parmi les lignes de l'ensemble de résultats se fait au moyen de ces méthodes. Comparez leur utilisation à celle des itérateurs, des énumérations ou des ensembles de résultats JDBC.

Le nombre de lignes contenues dans l'ensemble de résultats est obtenu en exécutant la méthode `size()` de la liste de lignes.

Exemple : Requête utilisant l'API de requête par ligne :

```

...
// L'exemple suivant illustre une requête exécutée sur la
// table prédéfinie TASK et utilisant l'API par ligne
...
// exécution de la requête
ResultSet rs = bfm.queryRows("TASK", null, null, null);

// obtenir les méta-informations des entités
List atts = rs.getAttributeInfo();

// obtenir les entités et itérer dessus
while (rs.next()) {

    // manipuler une ligne particulière
    for (int i = 0; i < atts.size(); i++) {
        AttributeInfo ai = (AttributeInfo) atts.get(i);
        Serializable value = rs.getAttributeValue(ai.getName());

        // traiter...
    }
}
...

```

Concepts associés

Requêtes sur des tables de requêtes

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Requêtes sur des tables de requêtes pour l'extraction de métadonnées

L'exécution de requêtes sur les tables de requêtes définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requêtes. Des méthodes sont disponibles pour permettre l'extraction de métadonnées des tables de requêtes.

Les méthodes suivantes sont disponibles pour l'extraction de métadonnées lorsque vous exécutez des requêtes sur les tables de requêtes dans Business Process Choreographer en utilisant l'API de table de requêtes :

Tableau 34. Méthodes pour l'extraction de métadonnées des tables de requêtes

Objectif	Méthode
Renvoyer les métadonnées d'une table de requêtes spécifique	<code>getQueryTableMetaData</code>
Renvoyer la liste des métadonnées de table de requêtes ayant des propriétés spécifiques	<code>findQueryTableMetaData</code>

Tableau 34. Méthodes pour l'extraction de métadonnées des tables de requêtes (suite)

Objectif	Méthode
Renvoyer le contenu d'une table de requêtes en fonction d'entités et d'un sous-ensemble des métadonnées des attributs sélectionnés	queryEntities
Renvoyer le contenu d'une table de requêtes en fonction de lignes et d'un sous-ensemble des métadonnées des attributs sélectionnés	queryRows

Les métadonnées des tables de requêtes comprennent des informations en rapport avec la structure des tables et d'autres informations en rapport avec l'internationalisation.

Le tableau suivant présente les métadonnées relatives à la structure d'une table de requêtes.

Tableau 35. Métadonnées relatives à la structure d'une table de requêtes

Métadonnées	Description	Renvoyées par getQuery- TableMetaData	Renvoyées par findQuery- TableMetaData	Renvoyées par queryEntities	Renvoyées par queryRows
Nom de la table de requêtes	Le nom de la table de requêtes	Oui	Oui	Oui	Oui
Nom de la table de requêtes principale	Le nom de la table de requêtes lorsqu'il s'agit d'une table prédéfinie ou supplémentaire, ou le nom de la table principale lorsque la table interrogée est une table de requêtes composite	Oui	Oui	Oui	Oui
Type	Le type de table de requêtes : composite, prédéfinie ou supplémentaire.	Oui	Oui	Non	Non
Autorisation	L'autorisation qui est définie sur la table de requêtes : <ul style="list-style-type: none"> Utilisation d'éléments de travail Autorisation par instance ou par rôle ou absence de contrôle d'autorisation 	Oui	Oui	Non	Non
Attributs définis	Les métadonnées des attributs qui sont définis sur la table de requêtes	Oui	Oui	Non. Les métadonnées des attributs sélectionnés sont renvoyées.	Non. Les métadonnées des attributs sélectionnés sont renvoyées.
Attributs de clé	Les attributs de clé de la table de requêtes	Oui	Oui	Oui	Non. Ne concerne pas les requêtes par ligne.

Le tableau suivant présente les métadonnées relatives à l'internationalisation d'une table de requêtes.

Tableau 36. Métadonnées relatives à l'internationalisation d'une table de requêtes

Métadonnées	Description	Renvoyées par getQuery- TableMetaData	Renvoyées par findQuery- TableMetaData	Renvoyées par queryEntities	Renvoyées par queryRows
locales[]	Environnements locaux pour lesquels les noms d'affichage et les descriptions de la table de requêtes et des attributs sont définis.	Oui	Oui	Non	Non
Environnement local	Valeur du paramètre système \$LOCALE qui résulte de l'environnement local qui est transmis à l'API.	Oui	Oui	Oui	Oui
Nom d'affichage et description de la table de requêtes	Les noms d'affichage et les descriptions de la table de requêtes, fournis pour tous les environnements locaux définis.	Oui	Oui	Non	Non
Noms d'affichage et descriptions des attributs	Les noms d'affichage et les descriptions des attributs, fournis pour tous les environnements locaux définis.	Oui	Oui	Non	Non

Toutes les méthodes de l'API de table de requêtes EJB qui renvoient des métadonnées de table de requêtes acceptent un paramètre d'environnement local ; par exemple, `FilterOptions.setLocale` et `MetaDataOptions.setLocale`. Ce paramètre doit recevoir pour valeur l'environnement local Java que le client utilise pour présenter les informations à l'utilisateur. Ce paramètre d'environnement local sert à calculer la valeur du paramètre système \$LOCALE, lequel peut être utilisé dans les filtres et les critères de sélection. L'environnement local renvoyé contient l'environnement local Java qui est utilisé pour \$LOCALE.

Si votre requête extrait les noms d'affichage et les descriptions d'une table de requêtes spécifique, passez `getLocale` aux méthodes concernées afin d'obtenir ces noms et ces descriptions dans le même environnement local que celui des descriptions des tâches. Par exemple, les descriptions suivantes sont attachées au moyen du critère de sélection '`LOCALE=$LOCALE`'.

Exemple :

```
// l'exemple suivant montre comment extraire
// les métadonnées d'une table de requêtes composite particulière

...
// exécution de la requête
MetaDataOptions mdo = new MetaDataOptions("TASK", null, false, new Locale("en_US"));
List list = bfm.findQueryTableMetaData(mdo);

// pour obtenir les métadonnées d'une table de requêtes spécifique
// utiliser bfm.getQueryTableMetaData(...)

// itérer sur la liste des tables de requêtes dont la table principale est TASK
// => au moins une table de requêtes est renvoyée : la table prédéfinie TASK

Iterator iter = list.iterator();
while (iter.hasNext()) {
    QueryTableMetaData md = (QueryTableMetaData) iter.next();
    Locale effectiveLocale = md.getLocale();
}
```

```

String queryTableDisplayName = md.getDisplayName(effectiveLocale);
System.out.println("found query table: " + queryTableDisplayName);
List attributesList = md.getAttributeMetaData();
Iterator attrIter = attributesList.iterator();
while (attrIter.hasNext()) {
    AttributeMetaData amd = (AttributeMetaData) attrIter.next();
    String attributeDisplayName = amd.getDisplayName(effectiveLocale);
    System.out.println("\tattribute:" + attributeDisplayName);
}
}

```

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d’interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l’implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d’une API spécifique, appelée API de table de requête.

Internationalisation pour les métadonnées des tables de requêtes

L’internationalisation est prise en charge pour les métadonnées des tables de requêtes.

Les noms d’affichage et les descriptions des tables de requêtes composites peuvent être fournis en différentes langues correspondant à différents codes d’environnement local. Par exemple, il est possible de définir, pour une table de requêtes composite, un nom d’affichage en anglais (environnement local en_US), en allemand (environnement de) et dans la langue de l’environnement local par défaut. Cette configuration est réalisée durant le développement de la table de requêtes, à l’aide de Query Table Builder. Pour déployer une table de requêtes avec des noms d’affichage et des descriptions en différentes langues, vous devez utiliser l’option `-deploy jarFile` lorsque vous déployez la table dans le conteneur Business Process Choreographer.

En termes de traitement des environnements locaux, le comportement des méthodes de l’API de table de requêtes, `queryEntities` et `queryRows`, et des méthodes de manipulation des métadonnées fournies par cette même API, `getQueryTableMetaData` et `findQueryTableMetaData`, est similaire à celui qui est obtenu avec les regroupements de ressources Java.

Pour que les noms d’affichage et les descriptions des métadonnées d’une table de requêtes restent cohérents avec le contenu de ladite table, la valeur du paramètre système `$LOCALE` dépend des environnements locaux pour lesquels les noms d’affichage et les descriptions sont spécifiés dans la table de requêtes.

Exemple :

Dans le scénario suivant, un client affiche des listes de tâches et des listes de processus et crée une demande pour interroger une table de requêtes.

- Le client n'a pas spécifié d'environnement local particulier à utiliser pour présenter l'information à l'utilisateur. Il est probable que l'application n'est pas conçue pour fonctionner en différentes langues.
 - Un environnement local par défaut est spécifié pour les noms d'affichage et les descriptions des tables de requêtes. C'est le cas pour toutes les tables composites et supplémentaires construites avec la version actuelle de Query Table Builder. Par conséquent la valeur de \$LOCALE est default.
 - Pour l'environnement local par défaut, aucun nom d'affichage ou description n'est spécifié sur la table de requêtes. C'est le cas pour toutes les tables de requêtes prédéfinies ainsi que pour toutes les tables de requêtes déployées avec l'option `-deploy qtdFile`. La valeur de \$LOCALE est basée sur la méthode du regroupement de ressources Java.
- Le client a spécifié l'environnement local à utiliser pour présenter l'information à l'utilisateur. Par exemple, c'est le cas lorsque l'API REST pour les tables de requêtes est utilisée.
 - Les noms d'affichage et les descriptions sont spécifiés sur la table de requêtes. La méthode du regroupement de ressources Java est utilisée pour calculer la valeur de \$LOCALE d'après l'environnement local qui est transmis par le client.
 - Les noms d'affichage et les descriptions ne sont pas spécifiés sur la table de requêtes. La variable \$LOCALE est réglée à la valeur qui est transmise par le client.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Tables de requêtes et performances des requêtes

Les tables de requêtes offrent un nouveau modèle de programmation propre, conçu pour le développement d'applications clientes qui extraient des listes de tâches humaines et de processus métier dans Business Process Choreographer. Les tables de requêtes ont un effet bénéfique sur les performances des requêtes. Vous trouverez ci-après une description des options applicables aux tables de requêtes et des paramètres de l'API de tables de requêtes qui ont une incidence sur les performances des requêtes. Des informations sont également fournies à propos d'autres facteurs qui influent sur les performances.

Les temps de réponse des requêtes exécutées sur les tables de requêtes dépendent principalement des options d'autorisation, des filtres et des critères de sélection utilisés. Voici une liste de points à prendre en considération lorsqu'il est question de performances.

- Les options d'autorisation influent de façon importante sur les performances. Activez l'autorisation en utilisant le moins possible d'options et utilisez principalement les éléments de travail de personne et de groupe. Evitez d'utiliser des éléments de travail `inherited`. Les options d'autorisation peuvent être restreintes de manière plus importante au moment de l'exécution de la

requête. En outre, si cela n'est pas nécessaire, indiquez que l'autorisation par le biais des éléments de travail n'est pas requise.

- Si une autorisation par éléments de travail est requise, définissez un filtre d'autorisation. Par exemple, pour autoriser uniquement les objets de la table de requêtes qui contiennent un élément de travail de propriétaire potentiel, utilisez `WI.REASON=REASON_POTENTIAL_OWNER`.
- Le filtrage de la table de requêtes principale est par exemple utile pour autoriser uniquement les tâches à l'état Prêt de la table de requêtes lorsque TASK est la table de requêtes principale.
- En termes de performances, les filtres de table de requêtes et les filtres de requête (filtres transmis à l'exécution de la requête) sont moins efficaces que les filtres principaux.
- Lorsque cela est possible, évitez d'utiliser des paramètres dans les filtres et les critères de sélection.
- Évitez également d'utiliser des opérateurs LIKE dans les filtres et les critères de sélection.

Définitions des tables de requêtes composites

Le tableau suivant décrit l'impact qu'ont les options définies pour les tables de requêtes composites sur les performances des requêtes. Il traite également d'autres sujets en rapport avec les définitions des tables de requêtes composites. L'indication figurant dans la colonne Impact sur les performances est une moyenne ; les impacts réellement observés peuvent varier.

Tableau 37. Les options applicables aux tables de requêtes composites et leur impact sur les performances des requêtes

Objet ou sujet	Impact sur les performances	Description
Filtre de table de requêtes	Négatif	Les filtres appliqués aux tables de requêtes sont ceux qui ont l'impact le plus négatif sur les performances des requêtes. Généralement, ils ne peuvent pas utiliser les index définis dans la base de données.
Filtre de la table de requêtes principale	Positif	Un filtre appliqué à la table principale intervient très tôt dans le calcul des résultats de la requête et se révèle donc particulièrement efficace en termes de performances. Pour cette raison, il est conseillé de restreindre le contenu de la table de requêtes au moyen d'un filtre appliqué à la table de requêtes principale.
Filtre d'autorisation	Positif	Un filtre d'autorisation peut améliorer les performances de la requête dans des proportions comparables à celles du filtre appliqué à la table de requêtes principale. Dans la mesure du possible, un filtre d'autorisation doit être appliqué. Par exemple, si les éléments de travail reader (lecteur) ne doivent pas être pris en compte, spécifiez <code>WI.REASON=REASON_READER</code> .
Critères de sélection	Aucun	Pour certaines relations entre la table principale et les tables attachées, il faut définir un critère de sélection afin de garantir que ces relations sont bien du type un à un ou un à zéro. Généralement, un critère de sélection n'a qu'un impact légèrement négatif sur les performances, car il est évalué pour un petit nombre de lignes uniquement.
Paramètres	Aucun	Actuellement, l'emploi de paramètres dans les tables de requêtes n'a pas d'impact négatif sur les performances. Ils ne doivent cependant être utilisés qu'en cas de nécessité.

Tableau 37. Les options applicables aux tables de requêtes composites et leur impact sur les performances des requêtes (suite)

Objet ou sujet	Impact sur les performances	Description
Autorisation par instance	Négatif	Si l'autorisation par instance est utilisée, l'existence d'un élément de travail doit être vérifiée pour chaque objet dans la table de requêtes. Les éléments de travail sont représentés par les entrées de la table de requêtes WORK_ITEM. Cette vérification affecte les performances.
Autorisation par instance : <ul style="list-style-type: none"> • everybody • individuals • groups • inherited 	Négatif	Chaque type d'élément de travail dont l'utilisation est spécifiée dans la table de requêtes a un impact sur les performances. Les applications ayant à soumettre d'importants volumes de requêtes doivent, si possible, se limiter à l'emploi des éléments de travail 'individuals' et 'groups', voire à un seul de ces éléments. Les éléments de travail 'inherited' ne sont généralement pas nécessaires, en particulier pour la définition de listes de tâches renvoyant des tâches humaines qui représentent des tâches à effectuer. Ils doivent en revanche être utilisés lorsqu'il est clair qu'ils sont indispensables ; par exemple, pour retourner des listes de tâches qui appartiennent à un processus métier, si une personne, compte tenu de son autorisation, ne bénéficie que d'un accès en lecture au processus métier englobant les tâches en question.
Autorisation par rôle ou absence de contrôle d'autorisation	Aucun	Si l'autorisation par rôle est utilisée, ou si aucun contrôle d'autorisation n'est exercé, les vérifications par rapport aux éléments de travail n'ont pas lieu.
Nombre d'attributs définis	Actuellement aucun	Le nombre d'attributs contenus dans une table de requêtes n'a pas d'incidence sur les performances. Néanmoins, seuls les attributs véritablement nécessaires doivent faire partie d'une table de requêtes.

API de table de requêtes

Le tableau suivant décrit l'impact qu'ont les options définies pour l'API de table de requêtes sur les performances des requêtes. L'indication figurant dans la colonne Impact sur les performances est une moyenne ; les impacts réellement observés peuvent varier.

Tableau 38. Les options de l'API de table de requêtes et leur impact sur les performances des requêtes

Option	Impact sur les performances	Description
Attributs sélectionnés	Négatif (moins il y en a, mieux c'est)	Le nombre d'attributs sélectionnés lorsqu'une requête est exécutée sur une table de requêtes détermine directement le nombre de données qui devront être traitées à la fois par la base de données et par l'environnement d'exécution de table de requêtes de Business Process Choreographer. De plus, pour les tables de requêtes composites, les informations des tables attachées n'ont besoin d'être extraites que si elles sont spécifiées par les attributs sélectionnés ou référencées par le filtre de table de requêtes ou par le filtre de requête.

Tableau 38. Les options de l'API de table de requêtes et leur impact sur les performances des requêtes (suite)

Option	Impact sur les performances	Description
Filtre de requête	Négatif	S'il est spécifié, le filtre de requête a le même impact sur les performances que le filtre de table de requêtes. Cependant, il est préférable de spécifier les filtres sur les tables de requêtes plutôt que de les passer à l'API de table de requêtes.
Attributs de tri	Négatif	Le tri des résultats d'une requête est coûteux en temps de traitement ; de plus, dès lors que le tri est utilisé, les optimisations de la base de données sont restreintes. Le tri doit être évité s'il n'est pas indispensable. La plupart des applications en ont toutefois besoin.
Seuil	Positif	La spécification d'un seuil peut améliorer significativement les performances des requêtes. Il est recommandé de toujours spécifier un seuil.
Nombre d'éléments à sauter (skipCount)	Négatif	Le saut d'un nombre particulier d'objets dans l'ensemble de résultats d'une requête est coûteux en temps de traitement ; il ne doit être réalisé qu'en cas d'absolue nécessité (par exemple, pour la pagination des résultats).
Fuseau horaire	Aucun	Le réglage de fuseau horaire n'a pas d'incidence sur les performances.
Environnement local	Aucun	Le réglage d'environnement local n'a pas d'incidence sur les performances.
Lignes distinctes	Négatif	L'emploi de l'opérateur 'distinct' dans les requêtes a un certain impact sur les performances, mais il est parfois incontournable pour éviter l'extraction de lignes en double. Cette option concerne uniquement les requêtes par ligne ; elle est ignorée dans les autres cas.
Requêtes de comptage	Positif	Si seul le nombre total d'entités ou le nombre total de lignes d'une requête particulière est nécessaire (autrement dit, s'il n'est pas nécessaire d'extraire le contenu de toutes les entrées de la table de requêtes), il convient d'utiliser la méthode queryEntityCount ou la méthode queryRowCount, respectivement. L'environnement d'exécution de Business Process Choreographer peut appliquer des optimisations valides uniquement pour les requêtes de comptage.

Autres considérations

Les autres facteurs à prendre en considération en ce qui concerne les performances sont les suivants :

Tableau 39. Performances des tables de requêtes - Autres considérations

Elément	Description
Nombre de tables de requêtes sur le système	<p>Le nombre de tables de requêtes déployées dans un conteneur Business Process Choreographer n'influe pas sur les performances des requêtes exécutées sur ces tables. Il est également sans conséquence sur la navigation des instances de processus métier et n'a pas non plus d'impact sur les opérations de réclamation et d'achèvement des tâches humaines. Maintenez-le quand même à un niveau raisonnable, sous peine de compliquer la maintenance de votre environnement. Généralement, une table de requêtes particulière représente une seule et même liste de tâches ou liste de processus affichée dans l'interface utilisateur.</p>
Réglage de la base de données	<p>Même si l'accès au contenu des tables de requêtes est réalisé au moyen de code SQL optimisé, les meilleures pratiques de réglage des bases de données restent de mise lorsqu'il est question de la base de données de Business Process Choreographer :</p> <ul style="list-style-type: none">• La mémoire réservée à la base de données doit être réglée au maximum, en tenant compte des besoins en mémoire des autres processus exécutés sur le serveur de base de données et des limites propres au matériel.• Les statistiques relatives à la base de données doivent être le plus à jour possible ; elles doivent donc être actualisées à intervalles réguliers. Généralement, les procédures à cet effet sont déjà implémentées dans les grandes topologies. Par exemple, prévoyez une collecte hebdomadaire des statistiques destinées à l'optimiseur pour refléter les changements des données dans la base de données.• Les systèmes de gestion de bases de données fournissent des outils qui permettent de réorganiser (ou défragmenter) les conteneurs de données. L'agencement physique des données dans la base de données peut aussi influencer sur les performances des requêtes et les voies d'accès qu'elles empruntent.• L'optimisation des index est l'un des facteurs clés en matière de performance des requêtes. Business Process Choreographer est fourni avec des index prédéfinis, optimisés pour conférer les meilleures performances à la navigation des processus et aux requêtes dans des scénarios types. Dans les environnements personnalisés, d'autres index peuvent être nécessaires afin de soutenir les requêtes sur les listes de tâches ou de processus qui génèrent ou manipulent de gros volumes de données. Utilisez les outils fournis par la base de données pour prendre en charge les requêtes exécutées sur une table de requêtes.

Concepts associés

Tables de requêtes dans Business Process Choreographer

Les tables de requêtes permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches humaines et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requêtes fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requêtes sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

API de requête EJB de Business Process Choreographer

Les méthodes `query` ou `queryAll` du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

La méthode `query` peut être appelée par tous les utilisateurs, elle renvoie les propriétés des objets pour lesquels les éléments de travail existent. La méthode `queryAll` ne peut être appelée que par les utilisateurs avec les rôles J2EE suivants : `BPESystemAdministrator`, `TaskSystemAdministrator`, `BPESystemMonitor` ou `TaskSystemMonitor`. Cette méthode renvoie les propriétés de tous les objets qui sont stockés dans la base de données.

Toutes les requêtes API sont mappées avec les requêtes SQL. La forme de la requête SQL résultante dépend des aspects suivants :

- Si la requête a été appelée par une personne ayant l'un des rôles J2EE.
- Les objets qui sont interrogés. Des vues prédéfinies des bases de données sont disponibles pour vous permettre de rechercher les propriétés de l'objet.
- L'insertion d'une clause `From`, de conditions de jointure et de conditions propres à l'utilisateur pour le contrôle d'accès.

Les requêtes peuvent inclure à la fois des propriétés personnalisées et des propriétés de variable. Si vous ajoutez plusieurs propriétés personnalisées ou propriétés de variables à votre requête, des jointures automatiques sont créées dans la table de base de données correspondante. Suivant le système de base de données utilisé, les appels de `query()` peuvent avoir des implications diverses sur les performances.

Vous pouvez également stocker des requêtes dans la base de données Business Process Choreographer à l'aide de la méthode `createStoredQuery`. Vous fournissez les critères de requête lors de la définition de la requête stockée. Les critères sont appliqués lors de l'exécution de la requête stockée, ce qui signifie que les données sont regroupées durant cette période. Si la requête stockée contient des paramètres, ils sont également résolus lors de son exécution.

Pour plus d'informations sur les interfaces API de Business Process Choreographer, consultez Javadoc dans le package `com.ibm.bpe.api` pour les méthodes relatives aux processus et dans le package `com.ibm.task.api` pour les méthodes relatives aux tâches.

Concepts associés

Requêtes portant sur les données des processus métier et des tâches
Les données d'instance des processus métier et des tâches humaines à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche humaine grâce à une interface de requête.

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Conditions d'accès propres à l'utilisateur

Les conditions d'accès propres à l'utilisateur sont ajoutées lorsque l'instruction SQL SELECT est générée par la requête API. Ces conditions garantissent que seuls ces objets sont renvoyés à l'appelant parce qu'ils sont conformes à la condition spécifiée par l'appelant et rendus accessibles à ce dernier.

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Référence associée

 Vue de la base de données Business Process Choreographer

Ces informations de référence décrivent les colonnes disponibles dans les vues de base de données prédéfinies.

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

La syntaxe de la requête dépend du type d'objet. Le tableau suivant présente la syntaxe correspondant aux différents types d'objet.

Tableau 40.

Objet	Syntaxe
Modèle de processus	<code>ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);</code>
Modèle de tâche	<code>TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);</code>
Données relatives aux processus métier et aux tâches	<code>QueryResultSet query (java.lang.String selectClause, java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer skipTuples, java.lang.Integer threshold, java.util.TimeZone timezone);</code>

Concepts associés

API de requête EJB de Business Process Choreographer

Les méthodes `query` ou `queryAll` du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

Clause `Select`

La clause `SELECT` de la fonction identifie les propriétés de l'objet qui doivent être renvoyées par une requête.

Clause `Where`

La clause `WHERE` de la fonction de requête décrit les critères de filtrage à appliquer au domaine de la requête.

Clause `Order-by`

La clause `ORDER BY` de la fonction de requête spécifie les critères de tri pour l'ensemble de résultats de la requête.

Paramètre `Skip-tuples`

Le paramètre `skip-tuples` spécifie le nombre de tuples dans l'ensemble de résultats de requête, en partant du début, à ignorer et à ne pas renvoyer à l'appelant dans l'ensemble des résultats de requête.

Paramètre `Threshold`

Le paramètre `threshold` de la fonction de requête restreint le nombre d'objets renvoyés du serveur au client dans l'ensemble de résultats de requête.

Paramètre `Timezone`

Le paramètre `time-zone` de la fonction de requête définit le fuseau horaire des constantes d'horodatage de la requête.

Paramètres des requêtes stockées

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

Résultats d'interrogation

Un ensemble de résultats de requête contient les résultats d'une requête d'API de Business Process Choreographer.

Clause `Select` :

La clause `SELECT` de la fonction identifie les propriétés de l'objet qui doivent être renvoyées par une requête.

La clause `SELECT` décrit le résultat de la requête. Cette clause spécifie une liste de noms identifiant les propriétés des objets (colonnes du résultat) à renvoyer. Sa syntaxe est identique à celle de la clause `SELECT` de SQL ; utilisez la virgule pour séparer les différentes parties de la clause. Chaque partie de la clause doit spécifier une colonne d'une des vues prédéfinies. Les colonnes doivent être entièrement spécifiées par le nom de la vue et le nom de la colonne. Les colonnes renvoyées dans l'objet `QueryResultSet` sont affichées dans le même ordre que les colonnes spécifiées dans la clause `Select`.

La clause `SELECT` ne prend pas en charge des fonctions d'agrégation SQL telles `AVG()`, `SUM()`, `MIN()` ou `MAX()`.

Pour sélectionner les propriétés de plusieurs paires nom-valeur, telles que des propriétés personnalisées ou des propriétés de variables pouvant être interrogées, ajoutez un compteur à un chiffre au nom de la vue. Ce compteur peut adopter une valeur comprise de 1 à 9.

Exemples de clauses SELECT

- "WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON"
Obtient les type des objets associés et les motifs d'attribution des éléments de travail.
- "DISTINCT WORK_ITEM.OBJECT_ID"
Obtient tous les ID des objets, sans les doublons, pour lesquels l'appelant a un élément de travail.
- "ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON"
Obtient les noms des activités pour lesquelles l'appelant a des éléments de travail, ainsi que leurs motifs d'attribution.
- "ACTIVITY.STATE, PROCESS_INSTANCE.STARTER"
Obtient les états des activités et les initiateurs des instances de processus y associés.
- "DISTINCT TASK.TKIID, TASK.NAME"
Obtient tous les ID et les noms de tâches, sans les doublons, pour lesquels l'appelant a un élément de travail.
- "TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE"
Obtient les valeurs des propriétés personnalisées qui sont spécifiées dans la clause WHERE.
- "QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE"
Extrait les valeurs des propriétés de variables pouvant être interrogées. Ces parties sont ensuite spécifiées dans la clause Where.
- "COUNT(DISTINCT TASK.TKIID)"
Compte le nombre de éléments de travail pour les tâches uniques qui satisfont la clause WHERE.

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Clause Where :

La clause WHERE de la fonction de requête décrit les critères de filtrage à appliquer au domaine de la requête.

La syntaxe de la clause Where est identique à celle de la clause SQL WHERE. Vous n'avez pas besoin d'ajouter explicitement une clause SQL à partir d'une clause ou des prédicats de jointure à la clause Where de l'API, car ces constructions sont ajoutées automatiquement lors de l'exécution de la requête. Si vous ne désirez pas appliquer de critères de filtre, spécifiez null comme valeur de la clause WHERE.

La syntaxe de la clause WHERE prend en charge :

- Mots clés : AND, OR, NOT
- Opérateurs de comparaison : =, <=, <, <>, >, >=, LIKE
L'opération LIKE prend en charge les caractères génériques définis pour la base de données interrogée.
- Opération SET : IN

Les règles suivantes s'appliquent également :

- Spécifiez les constantes ID d'objet comme ID('string-rep-of-oid').
- Spécifiez les constantes binaires comme BIN('UTF-8 string').
- Utilisez des constantes symboliques au lieu des énumérations d'entiers. Par exemple, au lieu de spécifier une expression d'état d'activitéACTIVITY.STATE=2, spécifiez ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY.
- Si la valeur de la propriété de l'instruction de comparaison contient des guillemets simples ('), doublez ces guillemets ; par exemple, "TASK_CPROP.STRING_VALUE='d''automatisation'".
- Faites référence aux propriétés de plusieurs paires nom-valeur, telles que des propriétés personnalisées, en ajoutant un suffixe à un chiffre au nom de la vue. Par exemple : "TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'"
- Spécifiez les constantes d'horodatage comme TS('yyy-mm-ddThh :mm :ss'). Pour faire référence à la date actuelle, spécifiez CURRENT_DATE comme horodatage.
Au moins une valeur de date ou d'heure doit être spécifiée dans l'horodatage.
 - Si vous spécifiez uniquement une date, la valeur de l'heure sera zéro.
 - Si vous spécifiez uniquement une heure, la valeur de la date sera la date actuelle.
 - Si vous spécifiez une date, l'année doit consister d'au moins quatre chiffres ; les valeurs du mois et du jour sont optionnelles. Les valeurs du jour et du mois manquantes seront remplacées par 01. Par exemple, TS('2003') et identique à TS('2003-01-01T00 :00 :00').
 - Si vous spécifiez une heure, cette valeur sera convertie en format 24 heures. Par exemple, si la date actuelle est le premier janvier 2003, TS('T16 :04') ou TS('16 :04') est identique à TS('2003-01-01T16 :04 :00').

Exemples de clauses WHERE

- Comparaison d'un ID d'objet avec un ID existant

```
"WORK_ITEM.WIID =
ID('_WI :800c00ed.df8d7e7c.feffff80.38')"
```

Ce type de clause WHERE est d'habitude créé de façon dynamique avec un ID d'objet existant, obtenu d'un appel antérieur. Si cet ID d'objet est stocké dans une variable *wiid1*, la clause peut être générée comme :

```
"WORK_ITEM.WIID = ID('" + wiid1.String() +
"')"
```

- Utilisation des horodatages

```
"ACTIVITY.STARTED >= TS('2002-06-1T16.00.00')"
```

- Utilisation des constantes symboliques

```
"WORK_ITEM.REASON =
WORK_ITEM.REASON.REASON_OWNER"
```

- Utilisation des valeurs booléennes vrai et faux

```
"ACTIVITY.BUSINESS_RELEVANCE = TRUE"
```

- Utilisation de propriétés personnalisées

```
"TASK_CPROP1.NAME = 'prop1' AND " TASK_CPROP1.STRING_VALUE = 'v1' AND
TASK_CPROP2.NAME = 'prop2' AND " TASK_CPROP2.STRING_VALUE = 'v2'"
```

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Clause Order-by :

La clause ORDER BY de la fonction de requête spécifie les critères de tri pour l'ensemble de résultats de la requête.

Vous pouvez spécifier la liste des colonnes à partir des vues servant de base de tri du résultat. Ces colonnes doivent être entièrement qualifiées par le nom de la vue et de la colonne.

La syntaxe de la clause Order-by est similaire à la syntaxe d'une clause SQL Order-by. Utilisez une virgule pour séparer chaque partie de la clause. Vous pouvez également spécifier la commande ASC pour trier les colonnes dans l'ordre croissant et la commande DESC pour les trier dans l'ordre décroissant. Si vous ne désirez pas trier l'ensemble de résultats, spécifiez la valeur null pour la clause ORDER BY.

Des critères de tri sont appliqués au serveur ; en fait, ce sont les paramètres régionaux du serveur qui sont utilisés pour le tri. Si la requête spécifie plusieurs propriétés, l'ensemble de résultats est trié par les valeurs de la première colonne et ensuite par les valeurs de la deuxième propriété, et ainsi de suite. Contrairement à la requête SQL, il est impossible de spécifier les colonnes dans la clause Order-by par position.

Exemples de clauses ORDER BY

- "PROCESS_TEMPLATE.NAME"

Trie les résultats de la requête alphabétiquement par le nom du modèle de processus.

- "PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC"

Trie les résultats de la requête par date de création, et pour une date spécifique, trie les résultats alphabétiquement par le nom de l'instance du processus en ordre inverse.

- "ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE"

Trie les résultats de la requête par le propriétaire de l'activité, ensuite par le nom du modèle d'activité et ensuite par l'état de l'activité.

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Paramètre Skip-tuples :

Le paramètre skip-tuples spécifie le nombre de tuples dans l'ensemble de résultats de requête, en partant du début, à ignorer et à ne pas renvoyer à l'appelant dans l'ensemble des résultats de requête.

Utilisez ce paramètre avec le paramètre threshold pour implémenter la pagination dans une application client, par exemple, pour extraire les 20 premiers éléments, puis les 20 éléments suivants, etc.

Si ce paramètre a pour valeur null et que le paramètre threshold n'est pas défini, tous les tuples correspondants sont renvoyés.

Exemple de paramètre skip-tuples

- new Integer(5)

Spécifie que les cinq premiers tuples correspondants ne seront pas renvoyés.

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Paramètre Threshold :

Le paramètre threshold de la fonction de requête restreint le nombre d'objets renvoyés du serveur au client dans l'ensemble de résultats de requête.

Dans un environnement de production, les ensembles de résultats d'une requête peuvent contenir des milliers voire des millions d'éléments. Pour cette raison, il est recommandé de toujours définir un seuil. Si vous définissez le paramètre threshold correctement, la requête dans la base de données est plus rapide et moins de données sont transférées à partir du serveur vers le client. Le paramètre threshold peut s'avérer utile, par exemple, dans une interface utilisateur graphique où seul un petit nombre d'éléments peuvent être affichés en même temps.

Si ce paramètre a pour valeur null et que le paramètre skip-tuples n'est pas défini, tous les objets correspondants sont renvoyés.

Exemple de paramètre threshold

- new Integer(50)

Spécifie que 50 tuples correspondants doivent être renvoyés.

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Paramètre Timezone :

Le paramètre time-zone de la fonction de requête définit le fuseau horaire des constantes d'horodatage de la requête.

Le fuseau horaire du client qui lance la requête peut différer de celui du serveur qui traite la requête. Utilisez le paramètre time-zone pour spécifier le fuseau horaire des constantes d'horodatage dans la clause WHERE utilisées, par exemple, pour spécifier l'heure locale. Les dates renvoyées dans l'ensemble de résultats de la requête sont dans le fuseau horaire spécifié pour la requête.

Si le paramètre a pour valeur null, les valeurs par défaut des constantes d'horodatage sont en temps universel UTC.

Exemples de paramètres time-zone

- ```
process.query("ACTIVITY.AIID",
 "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
 (Chaîne)null,
 (Entier)null,
 java.util.TimeZone.getDefault());
```

Renvoie les ID d'objet pour les activités démarrées après 17h40 heure locale, le premier janvier 2005.

- ```
process.query("ACTIVITY.AIID",  
             "ACTIVITY.STARTED > TS('2005-01-01T17:40')",  
             (Chaîne)null, (Entier)null, (FuseauHoraire)null);
```

Renvoie les ID d'objet pour les activités démarrées après 17h40 UTC, le premier janvier 2005. Cette spécification est décalée de 6 heures en heure EST (Eastern Standard Time).

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Paramètres des requêtes stockées :

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

Il existe par exemple des propriétés personnalisées pour stocker les noms de client. Vous pouvez définir des requêtes visant à renvoyer les tâches associées à un client donné, ACME Co. Pour faire la demande de ces informations, la clause where de votre requête devrait ressembler à ce qui est indiqué dans l'exemple suivant :

```
String whereClause =  
"TASK.STATE = TASK.STATE.STATE_READY  
AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER  
AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = 'ACME Co.'";
```

Pour rendre cette requête réutilisable afin de permettre également la recherche du client BCME Ltd, vous pouvez configurer des paramètres pour les valeurs de la propriété personnalisée. Si vous ajoutez des paramètres à la requête, celle-ci se peut présenter comme suit :

```
String whereClause =  
"TASK.STATE = TASK.STATE.STATE_READY  
AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER  
AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = '@param1'";
```

Le paramètre @param1 est résolu au moment de l'exécution à partir de la liste des paramètres transmis à la méthode query. Les règles suivantes s'appliquent lors de l'utilisation de paramètres dans les requêtes :

- Les paramètres sont utilisables uniquement dans la clause where.
- Les paramètres sont de type Chaîne.
- Les paramètres sont remplacés au moment de l'exécution via une substitution de chaînes. Si des caractères spéciaux sont nécessaires, vous devez les spécifier dans la clause where ou les insérer au moment de l'exécution en tant que partie du paramètre.

- Les noms de paramètre sont constitués de la chaîne @param concaténée avec un nombre entier. La valeur la plus faible est 1, ce qui renvoie au premier élément de la liste des paramètres transmis à l'API de la requête au moment de l'exécution.
- Un paramètre peut être réutilisé plusieurs fois au sein d'une clause where ; toutes les occurrences du paramètre sont remplacées par la même valeur.

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Tâches associées

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Résultats d'interrogation :

Un ensemble de résultats de requête contient les résultats d'une requête d'API de Business Process Choreographer.

Les éléments de l'ensemble de résultats sont les propriétés des objets qui sont conformes à la clause Where fournie par l'appelant et que ce dernier est autorisé à voir. Vous pouvez lire les éléments d'une manière relative à l'aide de la méthode next de l'API ou d'une manière absolue à l'aide des méthodes first et last. Le curseur implicite d'un ensemble de résultats de requête étant positionné, au départ, avant le premier élément, vous devez appeler la méthode first ou next avant de lire un élément. Vous pouvez utiliser la méthode size pour déterminer le nombre d'éléments d'un ensemble.

Un élément de l'ensemble de résultats de la recherche comprend les attributs sélectionnés des éléments de travail et les objets référencés y associés, tels que les instances d'activité et les instances de processus. Le premier attribut (colonne) d'un élément ResultSet spécifie la valeur du premier attribut spécifié dans la clause SELECT de la demande de requête. Le deuxième attribut (colonne) d'un élément ResultSet spécifie la valeur du deuxième attribut spécifié dans la clause SELECT de la demande de requête et ainsi de suite.

Vous pouvez extraire les valeurs des attributs en appelant une méthode compatible avec le type d'attribut et en spécifiant l'indice de colonne correspondant. La numérotation des indices de colonnes commence à 1.

Type d'attribut	Méthode
Chaîne	getString
OID	getOID
Horodatage	getTimestamp getString getTimestampAsLong

Type d'attribut	Méthode
Entier	getInteger getShort getLong getString getBoolean
Booléen	getBoolean getShort getInteger getLong getString
byte[]	getBinary

Exemple :

La requête suivante est exécutée :

```
QueryResultSet resultSet = process.query("ACTIVITY.STARTED,
                                         ACTIVITY.TEMPLATE_NAME AS NAME,
                                         WORK_ITEM.WIID, WORK_ITEM.REASON",
                                         (String)null, (String)null,
                                         (Integer)null, (TimeZone)null);
```

L'ensemble de résultats renvoyé a quatre colonnes :

- La colonne 1 est l'horodatage
- La colonne 2 est une chaîne
- La colonne 3 est un ID d'objet
- La colonne 4 est un entier

Les méthodes suivantes vous permettent d'obtenir les valeurs des attributs :

```
while (resultSet.next())
{
    java.util.Calendar activityStarted = resultSet.getTimestamp(1);
    String templateName = resultSet.getString(2);
    WIID wiid = (WIID) resultSet.getOID(3);
    Integer reason = resultSet.getInteger(4);
}
```

Vous pouvez utiliser les noms affichés de l'ensemble de résultats, par exemple, en tant qu'en-têtes d'un tableau imprimé. Ces noms sont les noms de colonnes de la vue ou du nom défini par la clause AS dans la requête. Cet exemple illustre l'utilisation de la méthode suivante pour obtenir les noms affichés :

```
resultSet.getColumnDisplayName(1) returns "STARTED"
resultSet.getColumnDisplayName(2) returns "NAME"
resultSet.getColumnDisplayName(3) returns "WIID"
resultSet.getColumnDisplayName(4) returns "REASON"
```

Concepts associés

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

Conditions d'accès propres à l'utilisateur

Les conditions d'accès propres à l'utilisateur sont ajoutées lorsque l'instruction SQL SELECT est générée par la requête API. Ces conditions garantissent que seuls ces objets sont renvoyés à l'appelant parce que conformes à la condition spécifiée par l'appelant et rendus accessibles à ce dernier.

La condition d'accès n'est ajoutée que si l'utilisateur est un administrateur système.

Requêtes appelées par les utilisateurs autres que les administrateurs système

La clause SQL générée WHERE combine l'API avec la clause dotée d'une condition de contrôle d'accès qui est propre à l'utilisateur. La requête n'extrait que les objets auxquels l'utilisateur est autorisé à accéder, autrement dit, uniquement les objets pour lesquels l'utilisateur dispose d'un élément de travail. Un élément de travail représente l'affectation du rôle d'autorisation d'un objet métier à un utilisateur ou un groupe, comme une tâche ou un processus. Par exemple, si l'utilisateur, John Smith, est un membre doté du rôle de propriétaire potentiel d'une tâche donnée, un objet élément de travail existe pour représenter cette relation.

Par exemple, si un utilisateur autre qu'un administrateur système, requiert des tâches, la condition d'accès suivante est ajoutée à la clause WHERE si les éléments de travail de groupe ne sont pas activés :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'user'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

De ce fait, si John Smith souhaite obtenir la liste des tâches dont il est propriétaire potentiel, l'API contenant la clause se présentera comme suit :

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

Cette API contenant la clause génère la condition d'accès suivante dans l'instruction SQL :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JohnSmith'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

Cela signifie également que si John Smith souhaite voir les activités et les tâches dont il est lecteur de processus ou administrateur de processus et pour lesquelles il dispose d'un élément de travail, une propriété provenant de la vue PROCESS_INSTANCE doit être ajoutée à la clause Select, Where, ou Order-by de la requête, telle que PROCESS_INSTANCE.PIID.

Si les éléments de travail de groupe sont activés, une autre condition d'accès est ajoutée à la clause WHERE qui permet à un utilisateur d'accéder aux objets auxquels le groupe a accès.

Requêtes appelées par les administrateurs système

Les administrateurs système peuvent appeler la méthode query pour extraire des objets dotés d'éléments de travail associés. Dans ce cas, un joint à la vue WORK_ITEM est ajouté à la requête SQL générée, mais sans condition de contrôle d'accès pour WORK_ITEM.OWNER_ID.

Dans ce cas, la requête SQL des tâches contient ce qui suit :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
```

queryAll queries

Ce type de requête ne peut être appelé que par les administrateurs système ou les contrôleurs système. Ni les conditions de contrôle d'accès, ni un joint à la vue WORK_ITEM ne sont ajoutés. Ce type de requête renvoie toutes les données de tous les objets.

Concepts associés

API de requête EJB de Business Process Choreographer

Les méthodes query ou queryAll du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Concepts associés

API de requête EJB de Business Process Choreographer

Les méthodes query ou queryAll du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

Exemple : requête de tâches à l'état Prêt

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté peut exploiter.

Exemple : requête de tâches à l'état Réclamé

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté a réclamées.

Exemple : interrogation d'escalades

Cet exemple indique comment utiliser la méthode query pour extraire les escalades pour l'utilisateur connecté.

Exemple : utilisation de la méthode queryAll

Cet exemple indique comment utiliser la méthode queryAll pour extraire toutes les activités propres à un modèle de processus.

Exemple : ajout de propriétés de requête à une requête

Cet exemple indique comment utiliser la méthode query pour extraire les tâches propres à un processus métier. Le processus dispose de propriétés de requête spécifiques que vous pouvez inclure à la recherche.

Exemple : ajout de propriétés personnalisées à une requête

Cet exemple montre comment utiliser la méthode query pour extraire les tâches dotées de propriétés personnalisées.

Exemple : requête de tâches à l'état Prêt :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté peut exploiter.

John Smith souhaite obtenir la liste des tâches qui lui ont été affectées. Pour qu'un utilisateur puisse travailler sur une tâche, celle-ci doit être à l'état Prêt. L'utilisateur connecté doit également avoir l'élément de travail d'un propriétaire potentiel de la tâche. Le fragment de code suivant affiche l'appel de méthode query pour cette requête :

```
query( "DISTINCT TASK.TKIID",  
      "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )  
      AND " +  
      "TASK.STATE = TASK.STATE.STATE_READY AND " +  
      "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",  
      (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que TASK.STATE.STATE_READY sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID
  FROM   TASK TA, WORK_ITEM WI,
  WHERE  WI.OBJECT_ID = TA.TKIID
  AND    TA.KIND IN ( 101, 105 )
  AND    TA.STATE = 2
  AND    WI.REASON = 1
  AND    ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Pour restreindre la requête API aux tâches d'un processus spécifique, par exemple, sampleProcess, la requête ressemble à ce qui suit :

```
query( "DISTINCT TASK.TKIID",
       "PROCESS_TEMPLATE.NAME = 'sampleProcess' AND "+
       "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )
       AND " +
       "TASK.STATE = TASK.STATE.STATE_READY AND " +
       "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Exemple : requête de tâches à l'état Réclamé :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté a réclamées.

L'utilisateur, John Smith, souhaite rechercher des tâches qu'il a réclamées et qui sont toujours à l'état Réclamé. La condition qui spécifie "réclamé par John Smith" est TASK.OWNER = 'JohnSmith'. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT TASK.TKIID",
       "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
       "TASK.OWNER = 'JohnSmith'",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID
  FROM   TASK TA, WORK_ITEM WI,
  WHERE  WI.OBJECT_ID = TA.TKIID
  AND    TA.STATE = 8
  AND    TA.OWNER = 'JohnSmith'
  AND    ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Lorsqu'une tâche est réclamée, les éléments de travail sont créés pour le propriétaire de la tâche. Ainsi, l'autre façon de former la requête pour les tâches réclamées de John Smith consiste à ajouter la condition suivante à la requête au lieu d'utiliser TASK.OWNER = 'JohnSmith' :

```
WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER
```

Alors la requête ressemble au fragment de code suivant :

```
query( "DISTINCT TASK.TKIID",  
      "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +  
      "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER",  
      (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que TASK.STATE.STATE_READY sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID  
FROM   TASK TA, WORK_ITEM WI,  
WHERE  WI.OBJECT_ID = TA.TKIID  
AND    TA.STATE = 8  
AND    WI.REASON = 4  
AND    ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

John est sur le point de partir en congés, donc son responsable d'équipe, Anne Grant, souhaite évaluer sa charge de travail actuelle. Anne dispose des droits d'administration. La requête qu'elle appelle est la même que celle appelée par John. Cependant, l'instruction SQL qui est générée est différente car Anne est administrateur. Le fragment de code suivant indique l'instruction SQL générée :

```
SELECT DISTINCT TASK.TKIID  
FROM   TASK TA, WORK_ITEM WI,  
WHERE  TA.TKIID = WI.OBJECT_ID =  
AND    TA.STATE = 8  
AND    TA.OWNER = 'JohnSmith')
```

Du fait qu'Anne est administrateur, une condition de contrôle d'accès n'est pas ajoutée à la clause WHERE.

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Exemple : interrogation d'escalades :

Cet exemple indique comment utiliser la méthode query pour extraire les escalades pour l'utilisateur connecté.

Lorsqu'une tâche est escaladée, un élément de travail récepteur d'escalade est créé. L'utilisateur Mary Jones souhaite voir la liste des tâches qui lui ont été escaladées. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT ESCALATION.ESIID, ESCALATION.TKIID",  
      "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_ESCALATION_RECEIVER",  
      (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que TASK.STATE.STATE_READY sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant indique l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT ESCALATION.ESIID, ESCALATION.TKIID
FROM   ESCALATION ESC, WORK_ITEM WI
WHERE  ESC.ESIID = WI.OBJECT_ID
AND    WI.REASON = 10
AND
( WI.OWNER_ID = 'MaryJones' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Exemple : utilisation de la méthode queryAll :

Cet exemple indique comment utiliser la méthode queryAll pour extraire toutes les activités propres à un modèle de processus.

La méthode queryAll est disponible uniquement pour les utilisateurs avec des droits d'administrateur système ou de contrôleur système. Le fragment de code suivant indique l'appel de méthode queryAll pour la requête permettant d'extraire toutes les activités propres au modèle de processus, sampleProcess :

```
queryAll( "DISTINCT ACTIVITY.AIID",
          "PROCESS_TEMPLATE.NAME = 'sampleProcess'",
          (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Le fragment de code suivant montre la requête SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT ACTIVITY.AIID
FROM   ACTIVITY AI, PROCESS_TEMPLATE PT
WHERE  AI.PTID = PT.PTID
AND    PT.NAME = 'sampleProcess'
```

Du fait que l'appel est invoqué par un administrateur, une condition de contrôle d'accès n'est pas ajoutée à l'instruction SQL générée. Un joint à la vue WORK_ITEM n'est pas ajouté non plus. Cela signifie que la requête extrait toutes les activités du modèle de processus, y compris les activités sans élément de travail.

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Exemple : ajout de propriétés de requête à une requête :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches propres à un processus métier. Le processus dispose de propriétés de requête spécifiques que vous pouvez inclure à la recherche.

Par exemple, vous souhaitez rechercher toutes les tâches utilisateur à l'état Prêt qui sont propres à un processus métier. Le processus fournit la propriété de requête **customerID** qui est dotée de la valeur CID_12345 et d'un espace de nom. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
        PROCESS_INSTANCE.NAME",
        " QUERY_PROPERTY.NAME = 'customerID' AND " +
        " QUERY_PROPERTY.STRING_VALUE = 'CID_12345' AND " +
        " QUERY_PROPERTY.NAMESPACE =
        'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
        " TASK.KIND IN
        ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
        " TASK.STATE = TASK.STATE.STATE_READY ",
        (String)null, (String)null, (Integer)null, (TimeZone)null );
```

A présent, si vous souhaitez ajouter une deuxième propriété de requête à la requête, comme par exemple, **Priority**, avec un espace de nom donné, l'appel de méthode query de la requête ressemble à ce qui suit :

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
        PROCESS_INSTANCE.NAME",
        " QUERY_PROPERTY1.NAME = 'customerID' AND " +
        " QUERY_PROPERTY1.STRING_VALUE = 'CID_12345' AND " +
        " QUERY_PROPERTY1.NAMESPACE =
        'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
        " QUERY_PROPERTY2.NAME = 'Priority' AND " +
        " QUERY_PROPERTY2.NAMESPACE =
        'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
        " TASK.KIND IN
        ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
        " TASK.STATE = TASK.STATE.STATE_READY ",
        (String)null, (String)null, (Integer)null, (TimeZone)null );
```

Si vous ajoutez plusieurs propriétés de requête à la requête, vous devez numéroter chaque propriété que vous ajoutez comme indiqué dans le fragment de code. Cependant, l'interrogation des propriétés personnalisées a une répercussion sur les performances, car elles se réduisent du fait du nombre de propriétés personnalisées dans la requête.

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Exemple : ajout de propriétés personnalisées à une requête :

Cet exemple montre comment utiliser la méthode query pour extraire les tâches dotées de propriétés personnalisées.

Par exemple, vous souhaitez rechercher toutes les tâches utilisateur à l'état Prêt avec la propriété personnalisée **customerID** et la valeur CID_12345. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT TASK.TKIID",
        " TASK_CPROP.NAME = 'customerID' AND " +
        " TASK_CPROP.STRING_VALUE = 'CID_12345' AND " +
        " TASK.KIND IN
        ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
        " TASK.STATE = TASK.STATE.STATE_READY ",
        (String)null, (String)null, (Integer)null, (TimeZone)null );
```

A présent, si vous souhaitez extraire les tâches et leurs propriétés personnalisées, l'appel de méthode query de la requête ressemble à ce qui suit :

```

query ( " DISTINCT TASK.TKIID, TASK_CPROP.NAME, TASK_CPROP.STRING_VALUE",
      " TASK.KIND IN
        ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
      " TASK.STATE = TASK.STATE.STATE_READY ",
      (String)null, (String)null, (Integer)null, (TimeZone)null );

```

L'instruction SQL qui est générée à partir de cette requête API s'affiche dans le fragment de code suivant :

```

SELECT DISTINCT TA.TKIID , TACP.NAME , TACP.STRING_VALUE
FROM TASK TA LEFT JOIN TASK_CPROP TACP ON (TA.TKIID = TACP.TKIID),
WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND TA.KIND IN ( 101, 105 )
AND TA.STATE = 2
AND (WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID IS NULL AND WI.EVERYBODY = 1 )

```

Cette instruction SQL contient un joint extérieur entre la vue TASK et la vue TASK_CPROP. Cela signifie que les tâches qui répondent à la clause WHERE sont extraites même si elles ne comportent pas de propriété personnalisée.

Concepts associés

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

A propos de cette tâche

Ces API EJB (Enterprise JavaBeans) permettent de créer des applications client pour effectuer les opérations suivantes :

- Gérer le cycle de vie des processus et des tâches, depuis leur lancement jusqu'à leur suppression finale
- Réparer des activités et des processus
- Gérer et distribuer la charge de travail entre les membres d'un groupe de travail

Les API EJB sont fournies sous forme de deux beans enterprise session sans état :

- L'interface BusinessFlowManagerService fournit les méthodes pour les applications de processus métier.
- L'interface HumanTaskManagerService fournit les méthodes pour les applications basées sur des tâches.

Pour plus d'informations concernant les API EJB, voir la documentation Java dans le package com.ibm.bpe.api et le package com.ibm.task.api.

La procédure suivante offre un aperçu des actions à entreprendre pour développer une application client EJB.

Procédure

1. Déterminez les fonctionnalités que l'application doit offrir.
2. Décidez quels beans session vous souhaitez utiliser.

En fonction des scénarios que vous souhaitez implémenter à l'aide de votre application, vous pouvez choisir l'un des beans session ou les deux.

3. Déterminez quels sont les droits requis par les utilisateurs de l'application.

Les utilisateurs de votre application doivent disposer des rôles d'autorisation appropriés pour pouvoir appeler les méthodes que vous incluez dans celle-ci et pour visualiser les objets et les attributs des objets renvoyés par ces méthodes. Si une instance du bean session approprié est créée, WebSphere Application Server associe un contexte à cette instance. Le contexte contient des informations relatives à l'ID principal de l'appelant, à la liste d'appartenance au groupe et aux rôles. Ces informations sont utilisées à la vérification des droits d'accès de l'appelant pour chaque appel.

Les informations d'autorisation relatives à chacune des méthodes sont décrites dans Javadoc.

4. Déterminez de quelle façon rendre l'application.

Les interfaces API EJB peuvent être appelées à distance ou localement.

5. Développez l'application.

a. Accédez à l'API EJB.

b. Utilisez l'API EJB pour interagir avec les processus ou les tâches.

- Recherchez les données.
- Utilisez les données.

Concepts associés

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches utilisateur. Chacune de ces interfaces présente des caractéristiques différentes.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans entreprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean entreprise de session approprié via l'interface home du bean.

Requête sur des objets liés aux processus métier et aux tâches

Les applications client fonctionnent avec des objets liés à des processus métier et à des tâches. Vous pouvez effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données afin d'extraire les propriétés spécifiques de ces objets.

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Développement d'applications pour les processus métier et les tâches utilisateur

La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction humaine lorsque le processus est démarré ou géré ou lorsque des activités humaines sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

Référence associée

 Vue de la base de données Business Process Choreographer

Ces informations de référence décrivent les colonnes disponibles dans les vues de base de données prédéfinies.

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans entreprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean entreprise de session approprié via l'interface home du bean.

A propos de cette tâche

L'interface `BusinessFlowManagerService` fournit les méthodes pour les applications de processus métier et l'interface `HumanTaskManagerService` fournit les méthodes pour les applications basées sur des tâches. Il peut s'agir de n'importe quelle application Java, y compris une autre application Enterprise JavaBeans (EJB).

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Accès à l'interface distante du bean session

Une application client EJB de processus métier ou de tâches utilisateur accède à l'interface distante du bean session par le biais de l'interface home distante du bean.

Accès à l'interface locale du bean session

Une application client EJB de processus métier ou de tâches utilisateur accède à l'interface locale du bean session par le biais de l'interface home locale du bean.

Accès à l'interface distante du bean session

Une application client EJB de processus métier ou de tâches utilisateur accède à l'interface distante du bean session par le biais de l'interface home distante du bean.

A propos de cette tâche

Le bean session peut être soit le bean session `BusinessFlowManager` pour les applications de processus, soit le bean session `HumanTaskManager` pour les applications de tâche.

Procédure

1. Ajoutez à l'interface distante du bean session une référence pointant vers le descripteur de déploiement d'applications. Ajoutez la référence à l'un des fichiers suivants :
 - Le fichier `application-client.xml` pour une application client Java 2 Platform Enterprise Edition (J2EE)
 - Le fichier `web.xml` pour une application Web
 - Le fichier `ejb-jar.xml` pour une application Enterprise JavaBeans (EJB)

La référence à l'interface home distante des applications de processus est illustrée dans l'exemple suivant :

```
<ejb-ref>
  <ejb-ref-name>ejb/BusinessFlowManagerHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
  <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
```

La référence à l'interface home locale des applications de tâche est illustrée dans l'exemple suivant :

```
<ejb-ref>
  <ejb-ref-name>ejb/HumanTaskManagerHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.task.api.HumanTaskManagerHome</home>
  <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
```

Si vous utilisez WebSphere Integration Developer pour ajouter la référence EJB au descripteur de déploiement, la liaison de la référence EJB est automatiquement créée lors du déploiement de l'application. Pour plus d'informations concernant l'ajout de références EJB, consultez la documentation WebSphere Integration Developer.

2. Intégrez les substituts générés dans votre application.
 - a. Pour les applications de processus, intégrez les fichiers contenus dans le fichier `<racine_installation>/ProcessChoreographer/client/bpe137650.jar` et le fichier d'archive d'entreprise (EAR) de votre application.
 - b. Pour les applications de tâche, intégrez le fichier `<racine_installation>>/ProcessChoreographer/client/task137650.jar` avec le fichier EAR de votre application.
 - c. Définissez le paramètre de chemin de classes dans le fichier manifeste du module d'application afin d'y inclure le fichier JAR.
Le module d'application peut être une application J2EE, une application Web ou une application EJB.
3. Décidez de la méthode adoptée pour fournir les définitions des objets métier.
Pour utiliser des objets métier dans une application client distante, vous devez avoir accès aux schémas correspondants pour les objets métier (fichiers XSD ou WSDL) utilisés pour l'interaction avec un processus ou une tâche. L'accès à ces fichiers est possible de l'une des manières suivantes :
 - Si l'application client n'est pas exécutée dans un environnement géré J2EE, incluez les fichiers dans le fichier EAR de l'application client.
 - Si l'application client est une application Web ou un client EJB exécuté dans un environnement géré J2EE, vous pouvez soit inclure les fichiers dans le fichier EAR de l'application client, soit bénéficier du chargement des artefacts distants.
 - a. Utilisez l'interface API EJB `createMessage` de Business Process Choreographer et les méthodes `ClientObjectWrapper.getObject` pour charger les définitions d'objet métier distantes de l'application correspondante vers le serveur, de façon transparente.
 - b. Utilisez l'interface de programmation Service Data Object pour créer ou consulter un objet métier en tant que partie d'un objet métier déjà instancié. Pour cela, utilisez les méthodes `commonj.sdo.DataObject.createDataObject` ou `getDataObject` sur l'interface `DataObject`.
 - c. Lorsque vous souhaitez créer un objet métier en tant que valeur de propriété d'un objet métier saisie à l'aide du schéma XML `any` ou `anyType`, utilisez les services Business Object pour créer ou lire votre objet métier. Pour cela, vous devez définir le contexte de RAL de manière à pointer vers l'application à partir de laquelle les schémas seront chargés. Vous pouvez ensuite les services des objets métier appropriés.
Créez par exemple un objet métier dans lequel "ApplicationName" est le nom de l'application qui contient les définitions de vos objets métier.

```

BOFactory bofactory = (BOFactory) new
    ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");

com.ibm.wsspi.al.ALContext.setContext
    ("RALTemplateName", "ApplicationName");
try {
    DataObject dataObject = bofactory.create("uriName", "typeName" );
} finally {
    com.ibm.wsspi.al.ALContext.unset();
}

```

Lisez par exemple une entrée XML dans laquelle "ApplicationName" est le nom de l'application qui contient les définitions de vos objets métier.

```

BOXMLSerializer serializerService =
    (BOXMLSerializer) new ServiceManager().locateService
        ("com/ibm/websphere/bo/BOXMLSerializer");
ByteArrayInputStream input = new ByteArrayInputStream("<?xml?>..");

com.ibm.wsspi.al.ALContext.setContext
    ("RALTemplateName", "ApplicationName");
try {
    BOXMLDocument document = serializerService.readXMLDocument(input);
    DataObject dataObject = document.getDataObject();
} finally {
    com.ibm.wsspi.al.ALContext.unset();
}

```

- Localisez l'interface home distante du bean session dans l'interface JNDI (Java Naming and Directory Interface).

L'exemple suivant illustre cette étape pour une application de processus :

```

// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

// Rechercher l'interface home distante du bean BusinessFlowManager
Object result =
    initialContext.lookup("java :comp/env/ejb/BusinessFlowManagerHome");

// Convertir le résultat de la recherche dans le type approprié
BusinessFlowManagerHome processHome =
    (BusinessFlowManagerHome) javax.rmi.PortableRemoteObject.narrow
        (result, BusinessFlowManagerHome.class);

```

L'interface home distante du bean session contient une méthode de création pour les objets EJB. Cette méthode renvoie l'interface distante du bean session.

- Accédez à l'interface distante du bean session.

L'exemple suivant illustre cette étape pour une application de processus :

```
BusinessFlowManager process = processHome.create();
```

L'accès au bean session ne garantit pas que l'appelant puisse effectuer toutes les actions sur un certain processus ; l'appelant doit être également autorisé à effectuer l'action. Lorsqu'une instance du bean session est créée, elle est associée à un contexte du bean session. Le contexte contient l'ID principal de l'appelant, la liste d'appartenance au groupe et indique si l'appelant est titulaire d'un des rôles J2EE de Business Process Choreographer. Le contexte est utilisé pour vérifier l'autorisation de l'appelant pour chaque appel, même lorsque la sécurité administrative n'est pas configurée. Si la sécurité administrative n'est pas configurée, la valeur de l'ID principal de l'appelant est UNAUTHENTICATED.

- Appelez les fonctions métier exposées par l'interface de service.

L'exemple suivant illustre cette étape pour une application de processus :

```
process.initiate("MyProcessModel", input);
```

Les appels venant des applications sont exécutés comme des transactions. Une transaction est établie et terminée de l'une des façons suivantes :

- Automatiquement par WebSphere Application Server (le descripteur de déploiement spécifie TX_REQUIRED).
- De manière explicite par l'application. Vous pouvez regrouper les appels d'application à l'intérieur d'une seule transaction :

```
// Obtenir l'interface de transaction utilisateur
UserTransaction transaction=
    (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

// Commencer une transaction
transaction.begin();

// Appels d'applications ...

// En cas d'aboutissement, valider la transaction
transaction.commit();
```

Conseil : Afin d'éviter tout conflit de verrouillage de la base de données, évitez d'exécuter en parallèle des instructions similaires à la suivante :

```
// Obtenir l'interface de transaction utilisateur
UserTransaction transaction=
    (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

transaction.begin();

//Lire l'instance d'activité
process.getActivityInstance(aiid);
//Réclamer l'instance d'activité
process.claim(aiid);

transaction.commit();
```

La méthode getActivityInstance ainsi que d'autres opérations de lecture définissent un verrou en lecture. Dans cet exemple, un verrou en lecture sur l'instance d'activité est mis à niveau vers un verrou U sur l'instance d'activité. Ceci peut provoquer un blocage de la base de données lorsque ces transactions sont exécutées en parallèle.

Exemple

Voici un exemple illustrant les étapes 3 à 5 pour une application de tâche.

```
// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

// Rechercher l'interface home distante du bean HumanTaskManager
Object result =
    initialContext.lookup("java:comp/env/ejb/HumanTaskManagerHome");

// Convertir le résultat de la recherche dans le type approprié
HumanTaskManagerHome taskHome =
    (HumanTaskManagerHome)javax.rmi.PortableRemoteObject.narrow
    (result,HumanTaskManagerHome.class);

...
//Accéder à l'interface distante du bean session
HumanTaskManager task = taskHome.create();

...
//Appeler les fonctions métier exposées par l'interface de service
task.callTask(tkiid,input);
```


Tâches associées

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans entreprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean entreprise de session approprié via l'interface home du bean.

Accès à l'interface locale du bean session

Une application client EJB de processus métier ou de tâches utilisateur accède à l'interface locale du bean session par le biais de l'interface home locale du bean.

A propos de cette tâche

Le bean session peut être soit le bean session BusinessFlowManager pour les applications de processus, soit le bean session HumanTaskManager pour les applications de tâche utilisateur.

Procédure

1. Ajoutez à l'interface locale du bean session une référence pointant vers le descripteur de déploiement d'applications. Ajoutez la référence à l'un des fichiers suivants :

- Le fichier application-client.xml pour une application client Java 2 Platform Enterprise Edition (J2EE)
- Le fichier web.xml pour une application Web
- Le fichier ejb-jar.xml pour une application Enterprise JavaBeans (EJB)

La référence à l'interface home locale des applications de processus est illustrée dans l'exemple suivant :

```
<ejb-local-ref>
  <ejb-ref-name>ejb/LocalBusinessFlowManagerHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
  <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
</ejb-local-ref>
```

La référence à l'interface home locale des applications de tâche est illustrée dans l'exemple suivant :

```
<ejb-local-ref>
  <ejb-ref-name>ejb/LocalHumanTaskManagerHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
  <local>com.ibm.task.api.LocalHumanTaskManager</local>
</ejb-local-ref>
```

Si vous utilisez WebSphere Integration Developer pour ajouter la référence EJB au descripteur de déploiement, la liaison de la référence EJB est automatiquement créée lors du déploiement de l'application. Pour plus d'informations concernant l'ajout de références EJB, consultez la documentation WebSphere Integration Developer.

2. Localisez l'interface home locale du bean session dans l'interface JNDI (Java Naming and Directory Interface).

L'exemple suivant illustre cette étape pour une application de processus :

```
// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

// Lookup the local home interface of the BusinessFlowManager bean

LocalBusinessFlowManagerHome processHome =
    (LocalBusinessFlowManagerHome) initialContext.lookup
    ("java :comp/env/ejb/LocalBusinessFlowManagerHome");
```

L'interface home locale du bean session contient une méthode de création pour les objets EJB. Cette méthode renvoie l'interface locale du bean session.

3. Accédez à l'interface locale du bean session.

L'exemple suivant illustre cette étape pour une application de processus :

```
LocalBusinessFlowManager process = processHome.create();
```

L'accès au bean session ne garantit pas que l'appelant puisse effectuer toutes les actions sur un certain processus ; l'appelant doit être également autorisé à effectuer l'action. Lorsqu'une instance du bean session est créée, elle est associée à un contexte du bean session. Le contexte contient l'ID principal de l'appelant, la liste d'appartenance au groupe et indique si l'appelant est titulaire d'un des rôles J2EE de Business Process Choreographer. Le contexte est utilisé pour vérifier l'autorisation de l'appelant pour chaque appel, même lorsque la sécurité administrative n'est pas configurée. Si la sécurité administrative n'est pas configurée, la valeur de l'ID principal de l'appelant est UNAUTHENTICATED.

4. Appelez les fonctions métier exposées par l'interface de service.

L'exemple suivant illustre cette étape pour une application de processus :

```
process.initiate("MyProcessModel", input);
```

Les appels venant des applications sont exécutés comme des transactions. Une transaction est établie et terminée de l'une des façons suivantes :

- Automatiquement par WebSphere Application Server (le descripteur de déploiement spécifie TX_REQUIRED).
- De manière explicite par l'application. Vous pouvez regrouper les appels d'application à l'intérieur d'une seule transaction :

```
// Obtenir l'interface de transaction utilisateur
UserTransaction transaction=
    (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

// Commencer une transaction
transaction.begin();

// Appels d'applications ...

// En cas d'aboutissement, valider la transaction
transaction.commit();
```

Conseil : Afin d'éviter tout blocage de la base de données, évitez d'exécuter en parallèle des instructions similaires à la suivante :

```
// Obtain user transaction interface
UserTransaction transaction=
    (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

transaction.begin();

//read the activity instance
process.getActivityInstance(aiid);
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

La méthode getActivityInstance ainsi que d'autres opérations de lecture définissent un verrou en lecture. Dans cet exemple, un verrou en lecture sur l'instance d'activité est mis à niveau vers un verrou U sur l'instance d'activité. Ceci peut provoquer un blocage de la base de données lorsque ces transactions sont exécutées en parallèle

Exemple

Voici un exemple illustrant les étapes 2 à 4 pour une application de tâche.

```
//Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the local home interface of the HumanTaskManager bean
LocalHumanTaskManagerHome taskHome =
    (LocalHumanTaskManagerHome)initialContext.lookup
    ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Access the local interface of the session bean
LocalHumanTaskManager task =
taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

Tâches associées

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans entreprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean entreprise de session approprié via l'interface home du bean.

Requête sur des objets liés aux processus métier et aux tâches

Les applications client fonctionnent avec des objets liés à des processus métier et à des tâches. Vous pouvez effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données afin d'extraire les propriétés spécifiques de ces objets.

A propos de cette tâche

Durant la configuration de Business Process Choreographer, une base de données relationnelle est associée au conteneur de processus métier et au conteneur de tâche. La base de données stocke toutes les données de modèle et d'instance (programme d'exécution) nécessaires à la gestion des processus métier et des tâches. Utilisez une syntaxe SQL pour rechercher ces données.

Vous pouvez effectuer une requête unique pour extraire une propriété particulière d'un objet. Vous pouvez également enregistrer les requêtes que vous utilisez souvent et inclure ces requêtes stockées dans votre application.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Filtrage de données à l'aide de variables définies dans des requêtes

Un résultat de requête renvoie l'objet répondant aux critères de la recherche. Vous pouvez filtrer ces résultats selon les valeurs des variables.

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Référence associée

 Vue de la base de données Business Process Choreographer

Ces informations de référence décrivent les colonnes disponibles dans les vues de base de données prédéfinies.

Filtrage de données à l'aide de variables définies dans des requêtes

Un résultat de requête renvoie l'objet répondant aux critères de la recherche. Vous pouvez filtrer ces résultats selon les valeurs des variables.

A propos de cette tâche

Vous pouvez définir des variables utilisées par un processus lors de l'exécution dans son modèle de processus. Vous pouvez, pour ces variables, déclarer sur quelles parties porte la requête.

Voici un exemple : John Smith appelle sa société d'assurance afin de connaître le statut de sa demande d'indemnisation suite à un accident de la circulation. L'administrateur des demandes d'indemnisation recherche le dossier du client par le biais de son ID client.

Procédure

1. Facultatif : Répertoirez les propriétés des variables dans un processus pouvant faire l'objet d'une requête.

Identifiez le processus par le biais de l'ID du modèle de processus. Vous pouvez omettre cette étape si vous connaissez les variables pouvant faire l'objet de requêtes.

```
List variableProperties = process.getQueryProperties(ptid);
for (int i = 0; i < variableProperties.size(); i++)
{
    QueryProperty queryData = (QueryProperty)variableProperties.get(i);
    String variableName = queryData.getVariableName();
    String name          = queryData.getName();
    int mappedType      = queryData.getMappedType();
    ...
}
```

2. Dressez la liste des instances de processus contenant les variables conformes aux critères de filtrage.

Pour ce processus, l'ID client est modélisé en tant que partie de la variable customerClaim pouvant être soumise à la requête. Ainsi, vous pouvez rechercher la demande d'indemnisation par l'intermédiaire de l'ID client.

```
QueryResultSet result = process.query
    ("PROCESS_INSTANCE.NAME, QUERY_PROPERTY.STRING_VALUE",
     "QUERY_PROPERTY.VARIABLE_NAME = 'customerClaim' AND " +
     "QUERY_PROPERTY.NAME = 'customerID' AND " +
     "QUERY_PROPERTY.STRING_VALUE like 'Smith%'",
     (String)null, (Integer)null,
     (Integer)null, (TimeZone)null );
```

Cette opération renvoie un ensemble de résultats de requête contenant les noms d'instance de processus et les valeurs d'ID des clients dont l'identifiant commence par 'Smith'.

Tâches associées

Requête sur des objets liés aux processus métier et aux tâches

Les applications client fonctionnent avec des objets liés à des processus métier et à des tâches. Vous pouvez effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données afin d'extraire les propriétés spécifiques de ces objets.

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

A propos de cette tâche

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Une requête privée et une requête publique peuvent être sauvegardées sous le même nom. Les requêtes enregistrées par différents utilisateurs peuvent également avoir un nom identique.

Vous pouvez avoir stocké des requêtes pour des objets de processus métier, des objets de tâche ou une combinaison de ces deux types d'objets.

Concepts associés

Paramètres des requêtes stockées

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

Tâches associées

Requête sur des objets liés aux processus métier et aux tâches

Les applications client fonctionnent avec des objets liés à des processus métier et à des tâches. Vous pouvez effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données afin d'extraire les propriétés spécifiques de ces objets.

Gestion des requêtes stockées publiques

Les requêtes stockées publiques sont créées par l'administrateur système. Ces requêtes sont accessibles à tous les utilisateurs.

Gestion des requêtes stockées privées d'autres utilisateurs

Tout utilisateur peut créer des requêtes privées. Seul le propriétaire d'une requête et l'administrateur système peuvent les utiliser.

Gestion des requêtes stockées privées

Si vous n'êtes pas un administrateur système, vous pouvez créer, exécuter et supprimer vos propres requêtes stockées privées. Vous pouvez également utiliser les requêtes stockées publiques créées par l'administrateur système.

Gestion des requêtes stockées publiques :

Les requêtes stockées publiques sont créées par l'administrateur système. Ces requêtes sont accessibles à tous les utilisateurs.

A propos de cette tâche

En tant qu'administrateur système, vous pouvez créer, visualiser et supprimer des requêtes stockées publiques. Si vous ne spécifiez aucun ID utilisateur dans l'appel d'API, on suppose que la requête stockée est une requête stockée publique.

Procédure

1. Créez une requête stockée publique.

Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous le nom `CustomerOrdersStartingWithA`.

```
process.createStoredQuery("CustomerOrdersStartingWithA",
    "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
    "PROCESS_INSTANCE.NAME LIKE 'A%'",
    "PROCESS_INSTANCE.NAME",
    (Integer)null, (TimeZone)null);
```

Le résultat de la requête stockée consiste en une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).

2. Exécutez la requête définie par la requête stockée.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",
    new Integer(0), null);
```

Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Répertoirez les requêtes stockées publiques disponibles.

Le fragment de code suivant vous permet de restreindre aux requêtes publiques la liste des requêtes renvoyées.

```
String[] storedQuery = process.getStoredQueryNames(StoredQueryData.KIND_PUBLIC);
```

4. Facultatif : Vérifiez la requête définie par une requête stockée spécifique.

Une requête stockée privée peut porter le même nom qu'une requête stockée publique. Si ces noms sont identiques, la requête stockée renvoyée est la requête privée. Le fragment de code suivant montre comment renvoyer la requête publique portant le nom spécifié. Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez `StoredQuery` au lieu de `StoredQueryData` pour l'objet renvoyé.

```
StoredQueryData storedQuery = process.getStoredQuery(
    StoredQueryData.KIND_PUBLIC, "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();
```

5. Supprimez une requête stockée publique.

Le fragment de code suivant montre comment supprimer la requête stockée que vous avez créée à l'étape 1.

```
process.deleteStoredQuery("CustomerOrdersStartingWithA");
```

Tâches associées

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Gestion des requêtes stockées privées d'autres utilisateurs :

Tout utilisateur peut créer des requêtes privées. Seul le propriétaire d'une requête et l'administrateur système peuvent les utiliser.

A propos de cette tâche

En tant qu'administrateur système, vous pouvez gérer des requêtes stockées privées qui appartiennent à un utilisateur spécifique.

Procédure

1. Créez une requête stockée privée pour l'ID utilisateur Smith.

Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous le nom `CustomerOrdersStartingWithA` pour l'ID utilisateur Smith.

```
process.createStoredQuery("Smith", "CustomerOrdersStartingWithA",
    "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
    "PROCESS_INSTANCE.NAME LIKE 'A%'",
    "PROCESS_INSTANCE.NAME",
    (Integer)null, (TimeZone)null,
    (List)null, (String)null);
```

La requête stockée renvoie une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).

2. Exécutez la requête définie par la requête stockée.

```

QueryResultSet result = process.query
    ("Smith", "CustomerOrdersStartingWithA",
     (Integer)null, (Integer)null, (List)null);
new Integer(0));

```

Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Accédez à la liste des noms des requêtes privées appartenant à un utilisateur donné.

Par exemple, le fragment de code suivant montre comment obtenir la liste des requêtes privées appartenant à l'utilisateur Smith.

```
String[] storedQuery = process.getStoredQueryNames("Smith");
```

4. Affichez les détails d'une requête spécifique.

Le fragment de code suivant montre comment afficher les détails de la requête CustomerOrdersStartingWithA qui appartient à l'utilisateur Smith.

```

StoredQueryData storedQuery = process.getStoredQuery
    ("Smith", "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();

```

Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez StoredQuery au lieu de StoredQueryData pour l'objet renvoyé.

5. Supprimez une requête stockée privée.

Le fragment de code suivant montre comment supprimer une requête privée qui appartient à l'utilisateur Smith.

```
process.deleteStoredQuery("Smith", "CustomerOrdersStartingWithA");
```

Tâches associées

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Gestion des requêtes stockées privées :

Si vous n'êtes pas un administrateur système, vous pouvez créer, exécuter et supprimer vos propres requêtes stockées privées. Vous pouvez également utiliser les requêtes stockées publiques créées par l'administrateur système.

Procédure

1. Créez une requête stockée privée.

Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous un nom spécifique. Si aucun ID utilisateur n'est spécifié, on suppose que la requête stockée est une requête stockée privée de l'utilisateur connecté.

```

process.createStoredQuery("CustomerOrdersStartingWithA",
    "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
    "PROCESS_INSTANCE.NAME LIKE 'A%'",
    "PROCESS_INSTANCE.NAME",
    (Integer)null, (TimeZone)null);

```


Cette requête renvoie une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).

2. Exécutez la requête définie par la requête stockée.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",  
    new Integer(0));
```

Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Extrayez une liste des noms de requêtes stockées auxquelles l'utilisateur connecté peut accéder.

Le fragment de code suivant montre comment extraire les requêtes stockées auxquelles l'utilisateur connecté peut accéder.

```
String[] storedQuery = process.getStoredQueryNames();
```

4. Affichez les détails d'une requête spécifique.

Le fragment de code suivant montre comment afficher les détails de la requête CustomerOrdersStartingWithA dont l'utilisateur Smith est le propriétaire.

```
StoredQueryData storedQuery = process.getStoredQuery  
    ("CustomerOrdersStartingWithA");  
String selectClause = storedQuery.getSelectClause();  
String whereClause = storedQuery.getWhereClause();  
String orderByClause = storedQuery.getOrderByClause();  
Integer threshold = storedQuery.getThreshold();  
String owner = storedQuery.getOwner();
```

Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez StoredQuery au lieu de StoredQueryData pour l'objet renvoyé.

5. Supprimez une requête stockée privée.

Le fragment de code suivant indique comment supprimer une requête stockée privée.

```
process.deleteStoredQuery("CustomerOrdersStartingWithA");
```

Tâches associées

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

A propos de cette tâche

Un processus métier peut être soit un microflux, soit un processus de longue durée :

- Les microflux sont des processus métier de courte durée exécutés de manière synchrone. Après un court moment, le résultat est renvoyé à l'appelant.
- Les processus interruptibles de longue durée sont exécutés en tant que séquences d'activités chaînées. L'utilisation de certaines constructions dans un processus engendre des interruptions dans le flux de processus, notamment

l'appel d'une tâche utilisateur, d'un service utilisant une liaison synchrone ou encore l'utilisation d'activités automatiques.

Les branches parallèles du processus sont généralement accessibles de manière asynchrone, ce qui signifie que les activités des branches parallèles sont exécutées simultanément. En fonction du type et du paramètre de transaction de l'activité, une activité peut être exécutée au sein de sa propre transaction.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Traitement des activités humaines

Les activités humaines sont attribuées aux différentes personnes de votre organisation par l'intermédiaire des tâches élémentaires. Au démarrage d'un processus, des éléments de travail sont créés pour les propriétaires potentiels.

Traitement d'un flux de travaux par une seule personne

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. L'API `completeAndClaimSuccessor` prend en charge le traitement de ce type de flux de travaux.

Envoi d'un message à une activité en attente

Les activités de messages entrants (également appelées activités de réception, `onMessage` dans des activités de sélection, `onEvent` dans les gestionnaires d'événements) peuvent être utilisées pour synchroniser un processus d'exécution avec des événements du "monde extérieur". Par exemple, la réception d'un courrier électronique provenant d'un client en réponse à une demande d'informations peut correspondre à ce type d'événement.

Gestion des événements

L'ensemble d'un processus métier et chacune de ses portées peuvent être associés à des gestionnaires d'événements qui sont appelés si l'événement associé se produit. Les gestionnaires d'événements sont similaires aux activités de réception ou de sélection en cela qu'un processus peut fournir des opérations de service Web à l'aide de gestionnaires d'événements.

Analyse des résultats d'un processus

Un processus peut afficher des opérations de services Web modélisées sous forme d'opérations WSDL (Web Services Description Language) asynchrones ou de type requête-réponse. Les résultats des processus interruptibles à interface unidirectionnelle ne peuvent être obtenus par la méthode `getOutputMessage`, car ces processus ne produisent pas de résultat. Cependant, vous pouvez interroger le contenu des variables.

Réparation d'activités

Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

Rôles nécessaires pour effectuer des actions sur des instances de processus

L'accès à l'interface `BusinessFlowManager` ne garantit pas que l'appelant puisse effectuer toutes les actions sur un processus donné. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

Le tableau suivant indique les actions qu'un rôle spécifique peut effectuer sur une instance de processus.

Action	Rôle principal de l'appelant		
	Lecteur	Initiateur	Administrateur
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspend			x
transferWorkItem			x

Rôles nécessaires pour effectuer des actions sur les activités de processus métier

L'accès à l'interface BusinessFlowManager ne garantit pas que l'appelant puisse effectuer toutes les actions sur une activité donnée. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

Le tableau suivant indique les actions qu'un rôle spécifique peut effectuer sur une instance d'activité.

Action	Rôle principal de l'appelant				
	Lecteur	Editeur	Propriétaire potentiel	Propriétaire	Administrateur
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x
getCustomProperties	x	x	x	x	x
getCustomProperty	x	x	x	x	x
getCustomPropertyNames	x	x	x	x	x
getFaultMessage	x	x	x	x	x
getFaultNames	x	x	x	x	x
getInputMessage	x	x	x	x	x
getOutputMessage	x	x	x	x	x
getVariable	x	x	x	x	x
getVariableNames	x	x	x	x	x
getInputVariableNames	x	x	x	x	x
getOutputVariableNames	x	x	x	x	x
getWorkItems	x	x	x	x	x
setCustomProperty		x		x	x
setFaultMessage		x		x	x
setOutputMessage		x		x	x
setVariable					x
transferWorkItem				x Réservé au propriétaires ou administrateurs potentiels	x

Gestion du cycle de vie d'un processus métier

Une instance de processus est créé lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

A propos de cette tâche

Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour les actions de cycle de vie typiques sur les processus.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Démarrage de processus métier

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

Mise en suspens et reprise d'un processus métier

Vous pouvez mettre en suspens une instance de processus de niveau supérieur de longue durée pendant qu'elle est en cours d'exécution, puis la relancer ultérieurement.

Redémarrage d'un processus métier

Vous pouvez redémarrer une instance de processus se trouvant à l'état terminé, arrêté, échoué ou compensé.

Arrêt d'une instance de processus

Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administrateur de processus d'arrêter une instance de processus de niveau supérieur dans un état irrécupérable. Etant donné qu'une instance de processus se termine immédiatement, sans attendre l'arrêt de sous processus ou d'activités en cours, vous ne devez terminer une instance de processus que dans des situations exceptionnelles.

Suppression d'instances de processus

Les instances de processus terminées sont automatiquement supprimées de la base de données de Business Process Choreographer si la propriété correspondante est définie pour le modèle de processus dans le modèle de processus. Vous pouvez choisir de conserver les instances de processus dans votre base de données, par exemple, pour rechercher des données relatives aux instances de processus qui ne sont pas consignées dans le journal d'audit. Cependant, les données d'instance de processus stockées n'ont pas seulement une incidence sur l'espace disque et les performances mais elles empêchent la création d'instances de processus utilisant les mêmes valeurs d'ensembles de corrélation. Vous devez par conséquent supprimer régulièrement les données d'instances de processus de la base de données.

Démarrage de processus métier :

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

A propos de cette tâche

Des exemples sont fournis pour illustrer la façon dont vous pouvez développer des applications pour les scénarios de démarrage habituels des microflux et des processus longue durée.

Tâches associées

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Exécution d'un microflux contenant un service de démarrage unique

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage est unique si le microflux démarre avec une activité de réception ou lorsque l'activité de sélection n'a qu'une définition onMessage.

Exécution d'un microflux contenant un service de démarrage non unique

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage n'est pas unique si le microflux démarre avec une activité de sélection possédant plusieurs définitions onMessage.

Démarrage d'un processus de longue durée contenant un service de démarrage unique

Si le service de démarrage est unique, vous pouvez utiliser la méthode de déclenchement et transmettre le nom du modèle de processus en tant que paramètre. C'est le cas lorsque le processus de longue durée démarre avec une activité de sélection ou de réception unique et lorsque l'activité de sélection unique n'a qu'une définition onMessage.

Démarrage d'un processus de longue durée contenant un service de démarrage non unique

Un processus de longue durée peut être lancé par le biais de plusieurs activités de sélection ou de réception déclenchantes. Vous pouvez utiliser la méthode de déclenchement pour lancer le processus. Si le service de démarrage n'est pas unique, par exemple si le processus démarre avec plusieurs activités de réception ou de sélection ou avec une activité de sélection possédant plusieurs définitions onMessage, vous devez identifier le service à appeler.

Exécution d'un microflux contenant un service de démarrage unique :

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage est unique si le microflux démarre avec une activité de réception ou lorsque l'activité de sélection n'a qu'une définition onMessage.

A propos de cette tâche

Si le microflux implémente une opération de requête-réponse, c'est à dire si le processus contient une réponse, vous pouvez utiliser la méthode d'appel pour exécuter le processus transmettant le nom de modèle de processus comme paramètre d'appel.

Si le micro-flux est une opération unidirectionnelle, exécutez le processus via la méthode sendMessage. Cette méthode n'est pas traitée dans l'exemple.

Procédure

1. Facultatif : Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez exécuter.

Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
    PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_MICROFLOW",
"PROCESS_TEMPLATE.NAME",
    new Integer(50),
    (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés par la méthode d'appel.

2. Lancez le processus avec un message de sortie du type approprié.

Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ProcessTemplateData template = processTemplates[0];
//create a message for the single starting receive activity
ClientObjectWrapper input = process.createMessage
    (template.getID(),
    template.getInputMessageType());
DataObject myMessage = null;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)input.getObject();
    //set the strings in the message, for example, a customer name
    myMessage.setString("CustomerName", "Smith");
}

//run the process
ClientObjectWrapper output = process.call(template.getName(), input);
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myOutput = (DataObject)output.getObject();
    int order = myOutput.getInt("OrderNo");
}
```

Cette opération crée une instance du modèle de processus, CustomerTemplate, et transfère quelques données client. L'opération renvoie uniquement lorsque le processus est terminé. Le résultat du processus, OrderNo, est renvoyé à l'appelant.

Tâches associées

Démarrage de processus métier

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

Exécution d'un microflux contenant un service de démarrage non unique :

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage n'est pas unique si le microflux démarre avec une activité de sélection possédant plusieurs définitions onMessage.

A propos de cette tâche

Si le microflux implémente une opération de requête-réponse, c'est à dire si le processus contient une réponse, vous pouvez utiliser la méthode d'appel pour exécuter le processus transmettant l'ID du service de démarrage comme paramètre d'appel.

Si le micro-flux est une opération unidirectionnelle, exécutez le processus via la méthode `sendMessage`. Cette méthode n'est pas traitée dans l'exemple.

Procédure

1. Facultatif : Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez exécuter.

Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
    PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_MICROFLOW",
"PROCESS_TEMPLATE.NAME",
new Integer(50),
(TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés en tant que microflux.

2. Déterminez le service de démarrage à appeler.

Cet exemple utilise le premier modèle trouvé.

```
ProcessTemplateData template = processTemplates[0];
ActivityServiceTemplateData[] startActivities =
    process.getStartActivities(template.getID());
```

3. Lancez le processus avec un message de sortie du type approprié.

Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ActivityServiceTemplateData activity = startActivities[0];
//create a message for the service to be called
ClientObjectWrapper input =
    process.createMessage(activity.getServiceTemplateID(),
        activity.getActivityTemplateID(),
        activity.getInputMessageType());
DataObject myMessage = null;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)input.getObject();
    //set the strings in the message, for example, a customer name
    myMessage.setString("CustomerName", "Smith");
}
//run the process
ClientObjectWrapper output = process.call(activity.getServiceTemplateID(),
        activity.getActivityTemplateID(),
        input);
//check the output of the process, for example, an order number
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myOutput = (DataObject)output.getObject();
    int order = myOutput.getInt("OrderNo");
}
```

Cette opération crée une instance du modèle de processus, `CustomerTemplate`, et transfère quelques données client. L'opération renvoie uniquement lorsque le processus est terminé. Le résultat du processus, `OrderNo`, est renvoyé à l'appelant.

Tâches associées

Démarrage de processus métier

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

Démarrage d'un processus de longue durée contenant un service de démarrage unique :

Si le service de démarrage est unique, vous pouvez utiliser la méthode de déclenchement et transmettre le nom du modèle de processus en tant que paramètre. C'est le cas lorsque le processus de longue durée démarre avec une activité de sélection ou de réception unique et lorsque l'activité de sélection unique n'a qu'une définition onMessage.

Procédure

1. Facultatif : Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez lancer.

Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
"PROCESS_TEMPLATE.NAME",
new Integer(50),
(TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés par la méthode de déclenchement.

2. Lancez le processus avec un message de sortie du type approprié.

Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message. Si vous spécifiez un nom d'instance de processus, il ne doit pas commencer par un trait de soulignement. Si aucun nom d'instance de processus n'est spécifié, l'identifiant d'instance de processus (PIID) au format chaîne est utilisé en tant que nom.

```
ProcessTemplateData template = processTemplates[0];
//create a message for the single starting receive activity
ClientObjectWrapper input = process.createMessage
(template.getID(),
template.getInputMessageTypeName());

DataObject myMessage = null;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)input.getObject();
    //set the strings in the message, for example, a customer name
    myMessage.setString("CustomerName", "Smith");
}
//start the process
PIID piid = process.initiate(template.getName(), "CustomerOrder", input);
```

Cette opération crée une instance, CustomerOrder, et transfère quelques données client. Lorsque le processus démarre, l'opération renvoie à l'appelant l'identifiant objet de la nouvelle instance de processus.

L'initiateur de l'instance de processus est défini pour l'appelant de la requête. Cette personne reçoit un élément de travail pour l'instance de processus. Les administrateurs du processus, les lecteurs et les éditeurs de l'instance de processus sont déterminés et reçoivent des éléments de travail pour l'instance

de processus. Les instances d'activité suivie sont déterminées. Elles sont lancées automatiquement, ou, si ce sont des activités manuelles, de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

Tâches associées

Démarrage de processus métier

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

Démarrage d'un processus de longue durée contenant un service de démarrage non unique :

Un processus de longue durée peut être lancé par le biais de plusieurs activités de sélection ou de réception déclenchantes. Vous pouvez utiliser la méthode de déclenchement pour lancer le processus. Si le service de démarrage n'est pas unique, par exemple si le processus démarre avec plusieurs activités de réception ou de sélection ou avec une activité de sélection possédant plusieurs définitions `onMessage`, vous devez identifier le service à appeler.

Procédure

1. Facultatif : Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez lancer.

Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_LONG_RUNNING",
"PROCESS_TEMPLATE.NAME",
new Integer(50),
(TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés en tant que processus de longue durée.

2. Déterminez le service de démarrage à appeler.

```
ProcessTemplateData template = processTemplates[0];
ActivityServiceTemplateData[] startActivities =
process.getStartActivities(template.getID());
```

3. Lancez le processus avec un message de sortie du type approprié.

Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message. Si vous spécifiez un nom d'instance de processus, il ne doit pas commencer par un trait de soulignement. Si aucun nom d'instance de processus n'est spécifié, l'identifiant d'instance de processus (PIID) au format chaîne est utilisé en tant que nom.

```
ActivityServiceTemplateData activity = startActivities[0];
//create a message for the service to be called
ClientObjectWrapper input = process.createMessage
(activity.getServiceTemplateID(),
activity.getActivityTemplateID(),
activity.getInputMessageType());

DataObject myMessage = null;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
myMessage = (DataObject)input.getObject();
//set the strings in the message, for example, a customer name
```

```

        myMessage.setString("CustomerName", "Smith");
    }
    //start the process
    PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                                   activity.getActivityTemplateID(),
                                   input);

```

Cette opération crée une instance et transfère quelques données client. Lorsque le processus démarre, l'opération renvoie à l'appelant l'identifiant objet de la nouvelle instance de processus.

L'initiateur de l'instance de processus est défini pour l'appelant de la requête et reçoit un élément de travail pour l'instance de processus. Les administrateurs du processus, les lecteurs et les éditeurs de l'instance de processus sont déterminés et reçoivent des éléments de travail pour l'instance de processus. Les instances d'activité suivie sont déterminées. Elles sont lancées automatiquement, ou, si ce sont des activités manuelles, de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

Tâches associées

Démarrage de processus métier

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

Mise en suspens et reprise d'un processus métier :

Vous pouvez mettre en suspens une instance de processus de niveau supérieur de longue durée pendant qu'elle est en cours d'exécution, puis la relancer ultérieurement.

Avant de commencer

L'appelant doit être un administrateur de l'instance de processus ou un administrateur de processus métier. Pour qu'une instance de processus puisse être mise en suspens, elle doit se trouver à l'état exécution en cours ou échec en cours.

A propos de cette tâche

Vous pouvez avoir besoin de mettre en suspens une instance de processus, par exemple, pour pouvoir configurer l'accès à un système dorsal qui est utilisé ultérieurement dans le processus. Une fois que les conditions prérequis pour le processus sont remplies, vous pouvez reprendre l'instance de processus. Vous pouvez également souhaiter interrompre un processus afin de résoudre un problème engendrant l'échec de l'instance de processus, puis le reprendre une fois le problème résolu.

Procédure

1. Obtenez le processus en cours d'exécution, CustomerOrder, que vous souhaitez mettre en suspens.

```

ProcessInstanceData processInstance =
    process.getProcessInstance("CustomerOrder");

```

2. Mettez l'instance de processus en suspens.

```

PIID piid = processInstance.getID();
process.suspend( piid );

```

Cette action suspend l'instance de processus de niveau supérieur spécifiée. L'instance de processus passe à l'état mis en suspens. Les sous-processus dont l'attribut `autonomy` est défini sur `enfant` (child) sont également suspendus, s'ils étaient en cours d'exécution, en état d'échec, terminés ou en cours de compensation. Les tâches en ligne associées à cette instance de processus sont également interrompues, ce qui n'est pas le cas des tâches autonomes.

Dans cet état, des activités lancées peuvent être terminées mais aucune nouvelle activité n'est activée, par exemple, une activité humaine associée à l'état réclamé peut être terminée.

3. Reprenez l'instance de processus.

```
process.resume( piid );
```

Cette action met l'instance de processus et ses sous processus dans l'état où ils se trouvaient avant d'être mis en suspens.

Tâches associées

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Redémarrage d'un processus métier :

Vous pouvez redémarrer une instance de processus se trouvant à l'état terminé, arrêté, échoué ou compensé.

Avant de commencer

L'appelant doit être un administrateur de l'instance de processus ou un administrateur de processus métier.

A propos de cette tâche

Le redémarrage d'une instance de processus est similaire au démarrage initial d'une instance de processus. Toutefois, lorsqu'une instance de processus est redémarrée, l'identifiant de l'instance de processus est connu et le message d'entrée pour l'instance est disponible.

Si le processus possède plusieurs activités de réception ou activités de sélection (également appelées activités de choix de réception) capables de créer l'instance de processus, tous les messages qui appartiennent à ces activités sont utilisés pour le redémarrage de l'instance de processus. Si l'une de ces activités implémentent une opération de requête-réponse, la réponse est envoyée à nouveau lors du survol de l'activité de réponse associée.

Procédure

1. Obtenez le processus que vous souhaitez redémarrer.

```
ProcessInstanceData processInstance =  
    process.getProcessInstance("CustomerOrder");
```

2. Redémarrez l'instance de processus.

```
PIID piid = processInstance.getID();  
process.restart( piid );
```

Cette action redémarrez l'instance de processus spécifiée.

Tâches associées

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Arrêt d'une instance de processus :

Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administrateur de processus d'arrêter une instance de processus de niveau supérieur dans un état irrécupérable. Étant donné qu'une instance de processus se termine immédiatement, sans attendre l'arrêt de sous processus ou d'activités en cours, vous ne devez terminer une instance de processus que dans des situations exceptionnelles.

Procédure

1. Procédez à l'extraction de l'instance de processus devant être arrêtée.

```
ProcessInstanceData processInstance =  
    process.getProcessInstance("CustomerOrder");
```

2. Arrêtez l'instance de processus.

Si vous arrêtez une instance de processus, vous pouvez arrêter l'instance de processus avec ou sans compensation.

Pour arrêter l'instance de processus avec compensation :

```
PIID piid = processInstance.getID();  
process.forceTerminate(piid, CompensationBehaviour.INVOKE_COMPENSATION);
```

Pour arrêter l'instance de processus sans compensation :

```
PIID piid = processInstance.getID();  
process.forceTerminate(piid);
```

Si vous arrêtez l'instance de processus avec compensation, la compensation du processus est exécutée comme si une erreur était survenue sur la portée de niveau supérieur. Si vous arrêtez l'instance de processus sans compensation, l'instance de processus est arrêtée aussitôt sans attendre que les activités en cours, les tâches à effectuer ou les tâches d'appel intégrées ne se terminent normalement.

Les applications démarrées par le processus et les tâches autonomes liées au processus ne sont pas arrêtées par la requête d'arrêt forcé. Si l'arrêt de ces applications est prévu, vous devez ajouter à l'application du processus les déclarations destinées à mettre fin explicitement aux applications initiées par le processus.

Tâches associées

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Suppression d'instances de processus :

Les instances de processus terminées sont automatiquement supprimées de la base de données de Business Process Choreographer si la propriété correspondante est définie pour le modèle de processus dans le modèle de processus. Vous pouvez

choisir de conserver les instances de processus dans votre base de données, par exemple, pour rechercher des données relatives aux instances de processus qui ne sont pas consignées dans le journal d'audit. Cependant, les données d'instance de processus stockées n'ont pas seulement une incidence sur l'espace disque et les performances mais elles empêchent la création d'instances de processus utilisant les mêmes valeurs d'ensembles de corrélation. Vous devez par conséquent supprimer régulièrement les données d'instances de processus de la base de données.

A propos de cette tâche

Pour supprimer une instance de processus, vous devez traiter les droits d'administrateur et l'instance de processus doit être une instance de processus de niveau supérieur.

L'exemple suivant montre comment supprimer toutes les instances de processus terminées.

Procédure

1. Répertoriez les instances de processus qui sont terminées.

```
QueryResultSet result =
    process.query("DISTINCT PROCESS_INSTANCE.PIID",
                 "PROCESS_INSTANCE.STATE =
                 PROCESS_INSTANCE.STATE.STATE_FINISHED",
                 (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête qui répertorie les instances de processus terminées.

2. Supprimez les instances de processus terminées.

```
while (result.next() )
{
    PIID piid = (PIID) result.getOID(1);
    process.delete(piid);
}
```

Cette action supprime l'instance de processus sélectionnée et ses tâches en ligne de la base de données.

Tâches associées

Gestion du cycle de vie d'un processus métier

Une instance de processus est créée lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

Traitement des activités humaines

Les activités humaines sont attribuées aux différentes personnes de votre organisation par l'intermédiaire des tâches élémentaires. Au démarrage d'un processus, des éléments de travail sont créés pour les propriétaires potentiels.

A propos de cette tâche

Lorsqu'une activité humaine est activée, une instance d'activité et une tâche à effectuer associée sont créées en même temps. Le traitement de l'activité humaine et la gestion de l'élément de travail sont délégués à l'application Human Task Manager. Toute modification d'état au niveau de l'instance d'activité est reflétée dans l'instance d'activité et inversement.

Un propriétaire potentiel réclame l'activité. Cette personne est responsable de fournir les informations pertinentes et de mener l'activité à terme.

Procédure

1. Répertoriez les activités appartenant à une personne connectée et qui sont prêtes à être traitées :

```
QueryResultSet result =
    process.query("ACTIVITY.AIID",
        "ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
        ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
        WORK_ITEM.REASON =
        WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
        (String)null, (Integer)null, (TimeZone)null);
```

Cette action renvoie un ensemble de résultats de requête contenant les activités pouvant être gérées par la personne connectée.

2. Réclamez l'activité à gérer :

```
if (result.size() > 0)
{
    result.first();
    AIID aaid = (AIID) result.getOID(1);
    ClientObjectWrapper input = process.claim(aaid);
    DataObject activityInput = null ;
    if ( input.getObject() != null && input.getObject() instanceof DataObject )
    {
        activityInput = (DataObject)input.getObject();
        // lire les valeurs
        ...
    }
}
```

Une fois l'activité réclamée, le message d'entrée de l'activité est renvoyé.

3. Une fois la gestion de l'activité terminée, terminez celle-ci. L'activité peut se terminer correctement, ou produire un message d'erreur. En cas de succès de l'activité, un message de sortie est transmis. En cas d'échec de l'activité, celle-ci est mise en état d'échec ou d'arrêt et un message d'erreur est transmis. Vous devez créer les messages appropriés pour ces opérations. Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

- a. Pour terminer l'activité correctement, créez un message de sortie.

```
ActivityInstanceData activity = process.getActivityInstance(aaid);
ClientObjectWrapper output =
    process.createMessage(aaid, activity.getOutputMessageType());
DataObject myMessage = null ;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myMessage = (DataObject)output.getObject();
    //définir les parties du message d'erreur, par exemple un numéro d'ordre
    myMessage.setInt("OrderNo", 4711);
}

//fin de l'activité
process.complete(aaid, output);
```

Cette opération définit un message de sortie contenant le numéro de commande.

- b. Pour terminer l'activité lorsque se produit une erreur, créez un message d'erreur.


```

//retrieve the faults modeled for the human task activity
List faultNames = process.getFaultNames(aiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
    process.createMessage(aiid, faultNames.get(0));

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)myFault.getObject();
    //set the parts in the message, for example, a customer name
    myMessage.setInt("error",1304);
}

process.complete(aiid, myFault,(String) faultNames.get(0) );

```

Cette action définit l'activité comme ayant l'état en échec ou arrêté. Si le paramètre **continueOnError** de l'activité contenue dans le modèle de processus est défini sur la valeur true, l'activité est mise en état d'échec et la navigation se poursuit. Si le paramètre **continueOnError** est défini sur false et que l'erreur n'est pas traitée dans la portée environnante, l'activité est mise à l'état arrêté. Lorsque l'activité se trouve dans cet état, elle peut être réparée via un arrêt ou un redémarrage forcé.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Traitement d'un flux de travaux par une seule personne

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. L'API `completeAndClaimSuccessor` prend en charge le traitement de ce type de flux de travaux.

A propos de cette tâche

Dans une librairie en ligne, l'acheteur accomplit une série d'actions afin de commander un ouvrage. Cette séquence d'actions peut être implémentée comme une série d'activités humaines (tâches à accomplir). Si l'acheteur décide de commander plusieurs livres, cela équivaut à réclamer l'activité humaine suivante. Ce type de flux de travaux est également appelé *flux de pages* du fait que les définitions d'interface sont associées aux activités de contrôle portant sur le flux des boîtes de dialogue dans l'interface utilisateur.

L'API `completeAndClaimSuccessor` effectue une activité humaine et demande la suivante dans la même instance de processus pour l'utilisateur connecté. L'API renvoie ensuite les informations sur l'activité réclamée suivante, y compris le message d'entrée à traiter. L'activité suivante étant disponible dans la même transaction que celle de l'activité terminée, le comportement transactionnel de toutes les activités humaines du modèle de processus doit être défini sur `participates`.

Comparez cet exemple avec celui qui utilise à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

Procédure

1. Réclamez la première activité dans la séquence d'activités.

```
//
//Requête portant sur la liste des activités pouvant être réclamées par
//l'utilisateur connecté
QueryResultSet result =
    process.query("ACTIVITY.AIID",
        "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
        ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
        ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
        WORK_ITEM.REASON =
            WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
        (String)null, (Integer)null, (TimeZone)null);
...
//
//Réclamer la première activité
//
if (result.size() > 0)
{
    result.first();
    AIID aaid = (AIID) result.getOID(1);
    ClientObjectWrapper input = process.claim(aaid);
    DataObject activityInput = null ;
    if ( input.getObject() != null && input.getObject() instanceof DataObject )
    {
        activityInput = (DataObject)input.getObject();
        // lire les valeurs
        ...
    }
}
```

Une fois l'activité réclamée, le message d'entrée de l'activité est renvoyé.

2. Une fois la gestion de l'activité terminée, terminez celle-ci et réclamez l'activité suivante.

Pour terminer l'activité, un message de sortie est créé. Lorsque vous créez le message de sortie, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ActivityInstanceData activity = process.getActivityInstance(aaid);
ClientObjectWrapper output =
    process.createMessage(aaid, activity.getOutputMessageType());
DataObject myMessage = null ;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myMessage = (DataObject)output.getObject();
    //définir les parties du message d'erreur, par exemple un numéro d'ordre
    myMessage.setInt("OrderNo", 4711);
}

//Fin de l'activité et réclamation de la suivante
CompleteAndClaimSuccessorResult successor =
    process.completeAndClaimSuccessor(aaid, output);
```

Cette opération définit un message de sortie contenant le numéro de commande et réclame l'activité suivante de la séquence. Si `AutoClaim` est défini pour les activités de succession et que plusieurs chemins d'accès peuvent être utilisés, toutes les activités de succession sont réclamées et une activité aléatoire est renvoyée en tant qu'activité suivante. Si aucune activité de succession supplémentaire ne peut être affectée à cet utilisateur, la valeur `Null` est renvoyée.

Si le processus contient des chemins parallèles pouvant être suivis, que ces chemins contiennent des activités humaines et que l'utilisateur connecté est le

propriétaire potentiel de plusieurs de ces activités, une activité aléatoire est automatiquement réclamée et renvoyée comme activité suivante.

3. Traitement de l'activité suivante.

```
String name = successor.getActivityName();

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject() !=
    null && nextInput.getObject() instanceof DataObject )
{
    activityInput = (DataObject)input.getObject();
    // lire les valeurs
    ...
}

aiid = successor.getAIID();
```

4. Poursuivez à l'étape 2 pour terminer l'activité.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Traitement par une seule personne d'un flux de travaux contenant des tâches utilisateur

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple démontre comment implémenter sous forme d'une série d'activités humaines (tâches à effectuer) la séquence d'actions nécessaires pour commander un livre. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

Envoi d'un message à une activité en attente

Les activités de messages entrants (également appelées activités de réception, `onMessage` dans des activités de sélection, `onEvent` dans les gestionnaires d'événements) peuvent être utilisées pour synchroniser un processus d'exécution avec des événements du "monde extérieur". Par exemple, la réception d'un courrier électronique provenant d'un client en réponse à une demande d'informations peut correspondre à ce type d'événement.

A propos de cette tâche

Vous pouvez utiliser des tâches d'origine pour envoyer le message à l'activité.

Procédure

1. Répertoriez les modèles de services d'activité attendant un message de l'utilisateur connecté dans une instance de processus avec un ID d'instance de processus spécifique.

```
ActivityServiceTemplateData[] services = process.getWaitingActivities(piid);
```

2. Envoyez un message au premier service en attente.

On suppose que le premier service est celui que vous souhaitez servir.

L'appelant doit être un démarreur potentiel de l'activité recevant le message ou un administrateur de l'instance de processus.

```
VTID vtid = services[0].getServiceTemplateID();
ATID atid = services[0].getActivityTemplateID();
String inputType = services[0].getInputMessageType();
```

```

// créer un message pour le service à appeler
ClientObjectWrapper message =
    process.createMessage(vtid,atid,inputMessageTypeName);
DataObject myMessage = null;
if ( message.getObject() != null && message.getObject() instanceof DataObject )
{
    myMessage = (DataObject)message.getObject();
    //set the strings in the message, for example, chocolate is to be ordered
    myMessage.setString("Order", "chocolate");
}

// envoi du message à l'activité en attente
process.sendMessage(vtid, atid, message);
}

```

Cette opération envoie le message spécifié au service d'activité en attente et transfère certaines données de commande.

Vous pouvez également spécifier l'identifiant de l'instance de processus afin de veiller à ce que le message soit envoyé à l'instance de processus spécifiée. Si l'identifiant de l'instance de processus n'est pas spécifié, le message est envoyé au service d'activité et à l'instance de processus identifiée par les valeurs de corrélation du message. Si l'identifiant de l'instance de processus est spécifié, l'instance de processus trouvée à l'aide des valeurs de corrélation est vérifiée afin de veiller à ce qu'elle possède bien l'identifiant de l'instance de processus spécifiée.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Gestion des événements

L'ensemble d'un processus métier et chacune de ses portées peuvent être associés à des gestionnaires d'événements qui sont appelés si l'événement associé se produit. Les gestionnaires d'événements sont similaires aux activités de réception ou de sélection en cela qu'un processus peut fournir des opérations de service Web à l'aide de gestionnaires d'événements.

A propos de cette tâche

Vous pouvez appeler un gestionnaire d'événements autant de fois que vous le souhaitez tant que la portée correspondante est en cours d'exécution. Par ailleurs, plusieurs instances d'un gestionnaire d'événements peuvent être activées en même temps.

Le fragment de code suivant montre comment obtenir les gestionnaires d'événements actifs pour une instance de processus donnée et comment envoyer un message d'entrée.

Procédure

1. Déterminez les données de l'identifiant d'instance de processus et répertoriez les gestionnaires d'événements actifs pour le processus.

```

ProcessInstanceData processInstance =
    process.getProcessInstance( "CustomerOrder2711");
EventHandlerTemplateData[] events = process.getActiveEventHandlers(
    processInstance.getID() );

```

2. Envoyez le message d'entrée.

Cet exemple utilise le premier gestionnaire d'événements trouvé.

```
EventHandlerTemplateData event = null;
if ( events.length > 0 )
{
    event = events[0];

    // créer un message pour le service à appeler
    ClientObjectWrapper input = process.createMessage(
        event.getID(), event.getInputMessageType());

    if (input.getObject() != null && input.getObject() instanceof DataObject )
    {
        DataObject inputMessage = (DataObject)input.getObject();
        // définir le contenu du message, par exemple, un nom de client, numéro
        de commande
        inputMessage.setString("CustomerName", "Smith");
        inputMessage.setString("OrderNo", "2711");

        // envoyer le message
        process.sendMessage( event.getProcessTemplateName(),
            event.getPortTypeNamespace(),
            event.getPortTypeName(),
            event.getOperationName(),

            input );
    }
}
```

Cette opération envoie le message spécifié au gestionnaire d'événements actif pour le processus.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Analyse des résultats d'un processus

Un processus peut afficher des opérations de services Web modélisées sous forme d'opérations WSDL (Web Services Description Language) asynchrones ou de type requête-réponse. Les résultats des processus interruptibles à interface unidirectionnelle ne peuvent être obtenus par la méthode `getOutputMessage`, car ces processus ne produisent pas de résultat. Cependant, vous pouvez interroger le contenu des variables.

A propos de cette tâche

Les résultats du processus ne sont stockés dans la base de données que si le modèle de processus dont dérive l'instance de processus ne spécifie pas une suppression automatique des instances de processus dérivées.

Procédure

Analysez les résultats des processus. Vérifiez par exemple le numéro de commande.

```
QueryResultSet result = process.query
    ("PROCESS_INSTANCE.PIID",
     "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
     PROCESS_INSTANCE.STATE =
     PROCESS_INSTANCE.STATE.STATE_FINISHED",
     (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
```

```

{
    result.first();
    PIIID piid = (PIIID) result.getOID(1);
    ClientObjectWrapper output = process.getOutputMessage(piid);
    DataObject myOutput = null;
    if ( output.getObject() != null && output.getObject() instanceof DataObject )
    {
        myOutput = (DataObject)output.getObject();
        int order = myOutput.getInt("OrderNo");
    }
}

```

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Réparation d'activités

Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

A propos de cette tâche

L'API de Business Process Choreographer propose les méthodes de réparation d'activité `forceRetry` et `forceComplete`. Plusieurs exemples illustrent l'ajout et la réparation d'actions pour des activités de vos applications.

Tâches associées

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

Forcer une activité à se terminer

Les activités situées dans des processus de longue durée rencontrent parfois des erreurs. Si ces erreurs ne sont pas interceptées par un gestionnaire d'erreurs dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Dans cet état, vous pouvez forcer l'activité à se terminer.

Nouvelle tentative d'exécution d'une activité arrêtée

Si une activité d'un processus de longue durée rencontre une erreur non interceptée dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Vous pouvez tenter d'exécuter à nouveau l'activité.

Forcer une activité à se terminer :

Les activités situées dans des processus de longue durée rencontrent parfois des erreurs. Si ces erreurs ne sont pas interceptées par un gestionnaire d'erreurs dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Dans cet état, vous pouvez forcer l'activité à se terminer.

A propos de cette tâche

Vous pouvez également forcer l'achèvement des activités en cours d'exécution si, par exemple, une activité ne répond pas.

Des exigences supplémentaires existent pour certains types d'activités.

Activités humaines

Vous pouvez transmettre des paramètres dans l'appel forcer à terminer, comme le message qui aurait dû être envoyé ou l'erreur qui aurait dû être détectée.

Activités de script

Vous ne pouvez pas transmettre de paramètres dans l'appel forcer à terminer. Cependant, vous devez définir les variables qui doivent être réparées.

Activités d'appel

Vous pouvez également forcer l'achèvement des activités d'appel appelant un service asynchrone qui n'est pas un sous-processus si ces activités sont dans l'état en cours d'exécution. Vous pouvez en avoir besoin, par exemple, si le service asynchrone est appelé et ne répond pas.

Procédure

1. Répertoriez les activités arrêtées qui se trouvent à l'état arrêté.

```
QueryResultSet result =
    process.query("DISTINCT ACTIVITY.AIID",
                 "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                 PROCESS_INSTANCE.NAME='CustomerOrder'",
                 (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie les activités arrêtées pour l'instance de processus CustomerOrder.

2. Achevez l'activité ; une activité humaine arrêtée, par exemple.

Dans cet exemple, un message de sortie est transmis.

```
if (result.size() > 0)
{
    result.first();
    AIID aiid = (AIID) result.getOID(1);
    ActivityInstanceData activity = process.getActivityInstance(aiid);
    ClientObjectWrapper output =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
    DataObject myMessage = null;
    if ( output.getObject() != null && output.getObject() instanceof DataObject )
    {
        myMessage = (DataObject)output.getObject();
        //set the parts in your message, for example, an order number
        myMessage.setInt("OrderNo", 4711);
    }

    boolean continueOnError = true;
    process.forceComplete(aiid, output, continueOnError);
}
```

Cette action effectue l'activité. Si une erreur survient, le paramètre **continueOnError** détermine l'action à entreprendre en cas d'erreur lors du traitement de la requête forceComplete.

Dans l'exemple, **continueOnError** est vrai. Cette valeur signifie que si une erreur se produit, l'activité est mise à l'état d'échec. L'erreur se propage aux

portées de l'activité jusqu'à ce qu'elle soit gérée ou que la portée du processus soit atteinte. Le processus est alors mis à l'état d'échec en cours avant d'atteindre finalement l'état d'échec.

Tâches associées

Réparation d'activités

Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

Nouvelle tentative d'exécution d'une activité arrêtée :

Si une activité d'un processus de longue durée rencontre une erreur non interceptée dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Vous pouvez tenter d'exécuter à nouveau l'activité.

A propos de cette tâche

Vous pouvez définir des variables utilisées par l'activité. à l'exception des activités de script, vous pouvez également transmettre des paramètres dans l'appel forcer la nouvelle tentative, comme le message qui était attendu par l'activité.

Procédure

1. Répertoriez les activités arrêtées.

```
QueryResultSet result =
    process.query("DISTINCT ACTIVITY.AIID",
        "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
        PROCESS_INSTANCE.NAME='CustomerOrder'",
        (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie les activités arrêtées pour l'instance de processus CustomerOrder.

2. Tentez à nouveau d'exécuter l'activité, une activité humaine, par exemple.

```
if (result.size() > 0)
{
    result.first();
    AIID aaid = (AIID) result.getOID(1);
    ActivityInstanceData activity = process.getActivityInstance(aaid);
    ClientObjectWrapper input =
        process.createMessage(aaid, activity.getOutputMessageType());
    DataObject myMessage = null;
    if ( input.getObject() != null && input.getObject() instanceof DataObject )
    {
        myMessage = (DataObject)input.getObject();
        //set the strings in your message, for example, chocolate is to be ordered
        myMessage.setString("OrderNo", "chocolate");
    }

    boolean continueOnError = true;
    process.forceRetry(aaid, input, continueOnError);
}
```

Cette opération tente à nouveau d'exécuter l'activité. Si une erreur se produit, le paramètre **continueOnError** détermine l'action à entreprendre en cas d'erreur lors du traitement de la requête forceRetry.

Dans l'exemple, `continueOnError` est vrai. Cela signifie que si une erreur se produit durant le traitement de la requête `forceRetry`, l'activité est mise en état échec. L'erreur se propage aux portées de l'activité jusqu'à ce qu'elle soit gérée ou que la portée du processus soit atteinte. Le processus est alors mis à l'état d'échec en cours, puis un gestionnaire d'erreur au niveau du processus est exécuté avant que le processus n'atteigne l'état d'échec.

Tâches associées

Réparation d'activités

Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

Interface `BusinessFlowManagerService`

L'interface `BusinessFlowManagerService` permet l'accès aux fonctions de processus métier pouvant être appelées par une application client.

Les méthodes pouvant être appelées par l'intermédiaire de l'interface `BusinessFlowManagerService` varient selon l'état du processus ou de l'activité et des droits d'accès de l'utilisateur de l'application qui contient la méthode. Les méthodes principales de manipulation des objets de processus métier sont répertoriées dans cette rubrique. Plus d'information sur ces méthodes et d'autres méthodes fournies par l'interface `BusinessFlowManagerService`, consultez Javadoc dans le package `com.ibm.bpe.api`.

Modèles de processus

Le modèle de processus est un exemple de processus mis à niveau, déployé et installé contenant la spécification d'un processus métier. Vous pouvez l'instancier et le démarrer en lançant les demandes appropriées, par exemple, `sendMessage()`. L'exécution de l'instance de processus est automatiquement gérée par le serveur.

Tableau 41. Méthodes API pour les modèles de processus

Méthode	Description
<code>getProcessTemplate</code>	Extrait le modèle de processus spécifié.
<code>queryProcessTemplates</code>	Extrait des modèles de processus stockés dans la base de données.

Traitement d'instances

Les méthodes API suivantes sont liées au démarrage des instances de processus.

Tableau 42. Les méthodes API sont liées au démarrage des instances de processus.

Méthode	Description
<code>call</code>	Crée et exécute un microflux.
<code>callWithReplyContext</code>	Crée et exécute un microflux avec un service à démarrage unique ou un processus longue durée provenant du modèle de processus spécifié. L'appel attend le renvoi du résultat en mode asynchrone.

Tableau 42. Les méthodes API sont liées au démarrage des instances de processus. (suite)

Méthode	Description
callWithUISettings	Crée et exécute un processus et renvoie le message de sortie et les paramètres de l'interface utilisateur (UI) du client.
initiate	Crée et exécute une instance de processus et démarre son traitement. Cette méthode est adaptée aux processus longue durée. Vous pouvez également appliquer cette méthode aux microflux destinés à être déclenchés, puis laissés sans surveillance.
sendMessage	Envoie le message spécifié au service d'activité et à l'instance de processus spécifiés. Si une instance de processus possédant les mêmes valeurs que l'ensemble de corrélations n'existe pas, celle-ci est créée. Le processus peut posséder des services de démarrage uniques ou non.
getStartActivities	Renvoie des informations sur les activités qui peuvent démarrer une instance de processus à partir du modèle de processus spécifié.
getActivityServiceTemplate	Extrait le modèle de service de l'activité spécifiée.

Tableau 43. Méthodes API pour le contrôle du cycle de vie des instances de processus

Méthode	Description
suspend	Met en suspens l'exécution d'une instance de processus de longue durée, de niveau supérieur se trouvant à l'état d'échec en cours ou d'exécution en cours.
resume	Reprend l'exécution d'une instance de processus de longue durée, de niveau supérieur se trouvant à l'état mis en suspens.
restart	Redémarre une instance de processus de longue durée, de niveau supérieur se trouvant à l'état terminé, échoué ou arrêté.
forceTerminate	Termine l'instance de processus de niveau supérieur spécifiée, ses sous-processus avec autonomie enfant et ses activités en cours d'exécution, réclamées, ou en attente
delete	Supprime l'instance de processus de niveau supérieur spécifiée et ses sous-processus avec autonomie enfant.
query	Extrait à partir de la base de données les propriétés correspondant aux critères de recherche.

Activités

Pour les activités d'appel, vous pouvez spécifier dans le modèle de processus que ces activités doivent continuer dans des situations d'erreur. Si l'indicateur

continueOnError est défini sur false et qu'une erreur non gérée survient, l'activité passe à l'état arrêté. L'administrateur du processus peut ensuite réparer l'activité. L'indicateur continueOnError et les fonctions de réparation associées peuvent être utilisés, par exemple, pour un processus de longue durée où les activités d'appel échouent occasionnellement mais où l'effort requis pour modéliser la compensation et la gestion des erreurs est trop important.

Les méthodes suivantes sont disponibles pour l'utilisation et la réparation des activités.

Tableau 44. Méthodes API pour le contrôle du cycle de vie des instances d'activité

Méthode	Description
claim	Réclame une instance d'activité prête pour permettre à un utilisateur d'utiliser l'activité.
cancelClaim	Annule la réclamation de l'instance d'activité.
complete	Termine l'instance d'activité.
completeAndClaimSuccessor	Effectue une instance d'activité et demande la suivante dans la même instance de processus pour l'utilisateur connecté.
forceComplete	Force l'exécution des éléments suivants : <ul style="list-style-type: none"> • Une instance d'activité se trouvant à l'état en cours d'exécution ou arrêté. • Une activité de tâche utilisateur se trouvant à l'état prêt ou réclamé. • Une attente d'attente se trouvant à l'état en attente.
forceRetry	Force la répétition des éléments suivants : <ul style="list-style-type: none"> • Une instance d'activité se trouvant à l'état en cours d'exécution ou arrêté. • Une activité de tâche utilisateur se trouvant à l'état prêt ou réclamé.
query	Extrait à partir de la base de données les propriétés correspondant aux critères de recherche.

Variables et propriétés personnalisées

L'interface fournit une méthode get et une méthode set pour l'extraction et la définition de valeurs pour les variables. Vous pouvez aussi associer les propriétés mentionnées aux instances de processus et d'activité et les en extraire. Le noms de propriétés personnalisées et des valeurs doivent être de type java.lang.String.

Tableau 45. Méthodes API pour les variables et les propriétés personnalisées

Méthode	Description
getVariable	Extrait la variable spécifiée.
setVariable	Définit la variable spécifiée.
getCustomProperty	Extrait la propriété personnalisée indiquée de l'activité ou instance de processus indiqué.

Tableau 45. Méthodes API pour les variables et les propriétés personnalisées (suite)

Méthode	Description
getCustomProperties	Extrait les propriétés personnalisées de l'activité ou de l'instance de processus indiquée.
getCustomPropertyNames	Extrait les noms des propriétés personnalisées pour l'instance d'activité ou de processus spécifiée.
setCustomProperty	Stocke les valeurs spécifiques aux propriétés personnalisées correspondant à l'instance d'activité ou de processus spécifiée.

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

A propos de cette tâche

Pour plus d'informations concernant l'API de Human Task Manager, voir la documentation Java dans le package `com.ibm.task.api`.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Démarrage d'une tâche d'appel qui appelle une interface synchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel synchrone que si le composant SCA associé peut être appelé de manière synchrone.

Démarrage d'une tâche d'appel qui appelle une interface asynchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel asynchrone que si le composant SCA associé peut être appelé de manière asynchrone.

Création et lancement d'une instance de tâche

Ce scénario indique comment créer une instance de modèle de tâche permettant de définir une tâche de collaboration (également appelée *tâche utilisateur* et de démarrer l'instance de tâche.

Traitement des tâches à effectuer ou des tâches de collaboration

Les tâches à effectuer (également appelées *tâches de participation* dans l'API) ou les tâches de collaboration (également appelées *tâches utilisateur* dans l'API) sont attribuées à diverses personnes de votre organisation par le biais des éléments de travail. Les tâches à effectuer et leurs éléments de travail associés sont créés, par exemple, lorsqu'un processus navigue jusqu'à une activité humaine.

Mise en suspens et reprise d'une instance de tâche

Vous pouvez interrompre les instances de tâche de collaboration (également appelées *tâches utilisateur* dans l'API) ou les instances de tâche à effectuer (également appelées *tâches de participation* dans l'API).

Analyse des résultats d'une tâche

Une tâche à effectuer (également appelée *tâche de participation* dans l'API) ou une tâche de collaboration (également appelée *tâche utilisateur* dans l'API) fonctionne de manière asynchrone. Si un gestionnaire de réponses est indiqué lors du démarrage d'une tâche, le message de sortie est automatiquement retourné à la fin de celle-ci. Dans le cas contraire, le message doit être extrait explicitement.

Arrêt d'une instance de tâche

Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administration d'arrêter une instance de tâche dans un état irrécupérable. Etant donné qu'une instance de tâche s'arrête instantanément, cette opération ne doit être exécutée que dans des situations exceptionnelles.

Suppression d'instances de tâche

Les instances de tâche ne sont automatiquement supprimées que lorsqu'elles sont terminées, à condition que cela soit spécifié dans le modèle de tâche associé dont dérivent les instances. Cet exemple montre comment supprimer toutes les instances de tâche qui sont terminées mais ne sont pas supprimées automatiquement.

Libération d'une tâche réclamée

Lorsqu'un propriétaire potentiel réclame une tâche, il lui incombe de mener la tâche à son terme. Toutefois, certaines tâches réclamées doivent être libérées pour afin qu'un autre propriétaire potentiel puisse la réclamer à son tour.

Gestion des tâches élémentaires

Durant la durée de vie d'une instance d'activité ou de tâche, l'ensemble des personnes associées à l'objet peut changer, par exemple, si une personne est en

congé, si de nouvelles personnes sont engagées ou si la charge de travail doit être redistribuée. Pour autoriser ces modifications, vous devez développer des applications afin de créer, supprimer ou transférer les tâches élémentaires.

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

Référence associée

Interface HumanTaskManagerService

L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Démarrage d'une tâche d'appel qui appelle une interface synchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel synchrone que si le composant SCA associé peut être appelé de manière synchrone.

A propos de cette tâche

Un tel composant SCA peut, par exemple, être implémenté en tant que microflux ou en tant que classe Java simple.

Ce scénario crée une instance d'un modèle de tâche et transmet certaines données client. La tâche reste à l'état actif jusqu'à la fin de l'opération bidirectionnelle. Le résultat de la tâche, OrderNo, est renvoyé à l'appelant.

Procédure

1. Facultatif : Répertoriez les modèles de tâche pour trouver le nom de la tâche d'appel que vous voulez exécuter.

Cette étape est facultative si vous connaissez déjà le nom de la tâche.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
 "TASK_TEMPL.NAME",
  new Integer(50),
  (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles d'origine classés.

2. Créez un message d'entrée pour le type approprié.

```
TaskTemplate template = taskTemplates[0];

// créer un a message pour la tâche sélectionnée
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //définir les parties du message, par exemple, un nom de client
  myMessage.setString("CustomerName", "Smith");
}
```

3. Créez la tâche et exécutez la tâche de façon synchrone.

Pour qu'une tâche s'exécute de façon synchrone, il doit s'agir d'une opération bidirectionnelle. L'exemple utilise la méthode `createAndCallTask` pour créer et exécuter la tâche.

```
ClientObjectWrapper output = task.createAndCallTask( template.getName(),
                                                    template.getNamespace(),
                                                    input);
```

4. Analysez le résultat de la tâche.

```
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myOutput = (DataObject)output.getObject();
    int order = myOutput.getInt("OrderNo");
}
```

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Démarrage d'une tâche d'appel qui appelle une interface asynchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel asynchrone que si le composant SCA associé peut être appelé de manière asynchrone.

A propos de cette tâche

Un tel composant SCA peut, par exemple, être implémenté en tant que processus à long terme ou en tant qu'opération unidirectionnelle.

Ce scénario crée une instance d'un modèle de tâche et transmet certaines données client.

Procédure

1. Facultatif : Répertoriez les modèles de tâche pour trouver le nom de la tâche d'appel que vous voulez exécuter.

Cette étape est facultative si vous connaissez déjà le nom de la tâche.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.ORIGINATING",
 "TASK_TEMPL.NAME",
 new Integer(50),
 (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles d'origine classés.

2. Créez un message d'entrée pour le type approprié.

```
TaskTemplate template = taskTemplates[0];

// créer un a message pour la tâche sélectionnée
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)input.getObject();
    //définir les parties du message, par exemple, un nom de client
    myMessage.setString("CustomerName", "Smith");
}
```

3. Créez la tâche et exécutez-la de façon asynchrone.

L'exemple utilise la méthode `createAndStartTask` pour créer et exécuter la tâche.

```
task.createAndStartTask( template.getName(),
                        template.getNamespace(),
                        input,
                        (ReplyHandlerWrapper)null);
```

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Création et lancement d'une instance de tâche

Ce scénario indique comment créer une instance de modèle de tâche permettant de définir une tâche de collaboration (également appelée *tâche utilisateur* et de démarrer l'instance de tâche.

Procédure

1. Facultatif : Répertoriez les modèles de tâche pour trouver le nom de la tâche de collaboration que vous voulez exécuter.

Cette étape est facultative si vous connaissez déjà le nom de la tâche.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_HUMAN",
 "TASK_TEMPL.NAME",
 new Integer(50),
 (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles de tâche classés.

2. Créez un message d'entrée pour le type approprié.

```
TaskTemplate template = taskTemplates[0];

// créer un a message pour la tâche sélectionnée
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)input.getObject();
    //définir les parties du message, par exemple, un nom de client
    myMessage.setString("CustomerName", "Smith");
}
```

3. Création et démarrage de la tâche de collaboration (aucun gestionnaire de réponse n'est spécifié dans cet exemple).

L'exemple utilise la méthode `createAndStartTask` pour créer et démarrer la tâche.

```
TKIID tkiid = task.createAndStartTask( template.getName(),
                                       template.getNamespace(),
                                       input,
                                       (ReplyHandlerWrapper)null);
```

Des éléments de travail sont créés pour les personnes concernées par l'instance de tâche. Un propriétaire potentiel, par exemple, peut réclamer la nouvelle instance de tâche.

4. Réclamation de l'instance de tâche.

```
ClientObjectWrapper input2 = task.claim(tkiid);
DataObject taskInput = null ;
if ( input2.getObject() != null && input2.getObject() instanceof DataObject )
```



```

{
    taskInput = (DataObject)input2.getObject();
    // lire les valeurs
    ...
}

```

Une fois l'instance de tâche réclamée, le message d'entrée de la tâche est renvoyé.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Traitement des tâches à effectuer ou des tâches de collaboration

Les tâches à effectuer (également appelées *tâches de participation* dans l'API) ou les tâches de collaboration (également appelées *tâches utilisateur* dans l'API) sont attribuées à diverses personnes de votre organisation par le biais des éléments de travail. Les tâches à effectuer et leurs éléments de travail associés sont créés, par exemple, lorsqu'un processus navigue jusqu'à une activité humaine.

A propos de cette tâche

L'un des propriétaires potentiels réclame la tâche associée à l'élément de travail. Cette personne est responsable de fournir les informations pertinentes et de mener la tâche à terme.

Procédure

1. Répertoriez les tâches appartenant à une personne connectée qui sont prêtes à être effectuées.

```

QueryResultSet result =
    task.query("TASK.TKIID",
              "TASK.STATE = TASK.STATE.STATE_READY AND
              (TASK.KIND = TASK.KIND.KIND_PARTICIPATING OR
              TASK.KIND = TASK.KIND.KIND_HUMAN)AND
              WORK_ITEM.REASON =
              WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
              (String)null, (Integer)null, (TimeZone)null);

```

Cette opération renvoie un ensemble de résultats de requête contenant les tâches pouvant être effectuées par la personne connectée.

2. Réclamez la tâche à effectuer.

```

if (result.size() > 0)
{
    result.first();
    TKIID tkiid = (TKIID) result.getOID(1);
    ClientObjectWrapper input = task.claim(tkiid);
    DataObject taskInput = null ;
    if ( input.getObject() != null && input.getObject() instanceof DataObject )
    {
        taskInput = (DataObject)input.getObject();
        // lire les valeurs
        ...
    }
}

```

Une fois la tâche réclamée, le message d'entrée de la tâche est renvoyé.

3. Une fois le travail de la tâche effectué, terminez la tâche.

La tâche peut se terminer correctement ou par un message d'erreur. Si la tâche s'exécute correctement, un message de sortie est transmis. Si la tâche ne

s'exécute pas correctement, un message d'erreur est transmis. Vous devez créer les messages appropriés pour ces opérations.

- a. Pour terminer la tâche correctement, créez un message de sortie.

```
ClientObjectWrapper output =
    task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myMessage = (DataObject)output.getObject();
    //set the parts in your message, for example, an order number
    myMessage.setInt("OrderNo", 4711);
}

//fin de la tâche
task.complete(tkiid, output);
```

Cette opération définit un message de sortie contenant le numéro de commande. La tâche est mise à l'état terminé.

- b. Pour terminer la tâche lorsque se produit une erreur, créez un message d'erreur.

```
//retrieve the faults modeled for the task List faultNames =
task.getFaultNames(tkiid);
ListfaultNames input = task.getFaultNames(tkiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
    task.createFaultMessage(tkiid, (String)faultNames.get(0));

// définir les parties du message d'erreur, par exemple un numéro d'erreur
DataObject myMessage = null ;
if ( myFault.getObject() != null && input.getObject() instanceof DataObject )
{
    myMessage = (DataObject)myFault.getObject();
    //définir les parties du message, par exemple, un nom de client
    myMessage.setInt("error",1304);
}

task.complete(tkiid, (String)faultNames.get(0), myFault);
```

Cette action définit un message d'erreur qui contient le code d'erreur. La tâche est mise à l'état d'échec.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Mise en suspens et reprise d'une instance de tâche

Vous pouvez interrompre les instances de tâche de collaboration (également appelées *tâches utilisateur* dans l'API) ou les instances de tâche à effectuer (également appelées *tâches de participation* dans l'API).

Avant de commencer

L'instance de tâche peut se trouver à l'état prêt ou réclamé. Elle peut être transférée à un niveau supérieur. L'appelant doit être le propriétaire, l'émetteur ou l'administrateur de l'instance de tâche.

A propos de cette tâche

Vous pouvez mettre une instance de tâche en suspens durant son exécution. Il peut également être souhaitable d'effectuer cette opération dans le but de recueillir des informations nécessaires pour achever la tâche. Une fois ces informations disponibles, vous pouvez reprendre l'exécution de l'instance de tâche.

Procédure

1. Obtention de la liste des tâches réclamées par l'utilisateur connecté.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",  
    "TASK.STATE = TASK.STATE.STATE_CLAIMED",  
    (String)null,  
    (Integer)null,  
    (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête contenant une liste des tâches réclamées par l'utilisateur connecté.

2. Met en suspens l'instance de tâche.

```
if (result.size() > 0)  
{  
    result.first();  
    TKIID tkiid = (TKIID) result.getOID(1);  
    task.suspend(tkiid);  
}
```

Cette action met en suspens l'instance de tâche spécifiée. L'instance de tâche est placée dans l'état Interrompu.

3. Reprise de l'instance de processus.

```
task.resume( tkiid );
```

Cette action remet l'instance de tâche dans l'état où elle se trouvait avant sa mise en suspens.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Analyse des résultats d'une tâche

Une tâche à effectuer (également appelée tâche de *participation* dans l'API) ou une tâche de collaboration (également appelée *tâche utilisateur* dans l'API) fonctionne de manière asynchrone. Si un gestionnaire de réponses est indiqué lors du démarrage d'une tâche, le message de sortie est automatiquement retourné à la fin de celle-ci. Dans le cas contraire, le message doit être extrait explicitement.

A propos de cette tâche

Les résultats de la tâche ne sont stockés dans la base de données que si le modèle de tâche dont dérive l'instance de tâche ne spécifie pas une suppression automatique des instances de tâche dérivées.

Procédure

Analysez les résultats de la tâche.

L'exemple illustre le contrôle du numéro d'ordre d'une tâche effectuée avec succès.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",  
    "TASK.NAME = 'CustomerOrder' AND  
    TASK.STATE = TASK.STATE.STATE_FINISHED",
```

```

                                (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
    result.first();
    TKIID tkiid = (TKIID) result.getOID(1);
    ClientObjectWrapper output = task.getOutputMessage(tkiid);
    DataObject myOutput = null;
    if ( output.getObject() != null && output.getObject() instanceof DataObject)
    {
        myOutput = (DataObject)output.getObject();
        int order = myOutput.getInt("OrderNo");
    }
}

```

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Arrêt d'une instance de tâche

Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administration d'arrêter une instance de tâche dans un état irrécupérable. Etant donné qu'une instance de tâche s'arrête instantanément, cette opération ne doit être exécutée que dans des situations exceptionnelles.

Procédure

1. Procédez à l'extraction de l'instance de tâche devant être arrêtée.

```
Task taskInstance = task.getTask(tkiid);
```

2. Arrêtez l'instance de tâche.

```
TKIID tkiid = taskInstance.getID();
task.terminate(tkiid);
```

L'instance de tâche est arrêtée aussitôt sans attendre les tâches en instance.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Suppression d'instances de tâche

Les instances de tâche ne sont automatiquement supprimées que lorsqu'elles sont terminées, à condition que cela soit spécifié dans le modèle de tâche associé dont dérivent les instances. Cet exemple montre comment supprimer toutes les instances de tâche qui sont terminées mais ne sont pas supprimées automatiquement.

Procédure

1. Répertoriez les instances de tâche qui sont terminées.

```
QueryResultSet result =
    task.query("DISTINCT TASK.TKIID",
              "TASK.STATE = TASK.STATE.STATE_FINISHED",
              (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête qui répertorie les instances de tâche terminées.

2. Supprimez les instances de tâche terminées.

```

while (result.next() )
{
    TKIID tkiid = (TKIID) result.getOID(1);
    task.delete(tkiid);
}

```

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Libération d'une tâche réclamée

Lorsqu'un propriétaire potentiel réclame une tâche, il lui incombe de mener la tâche à son terme. Toutefois, certaines tâches réclamées doivent être libérées pour afin qu'un autre propriétaire potentiel puisse la réclamer à son tour.

A propos de cette tâche

Il s'avère parfois nécessaire pour un utilisateur disposant de droits d'administration de libérer une tâche réclamée. Cette situation peut se produire, par exemple, lorsqu'une tâche doit être effectuée en l'absence du propriétaire de la tâche. Le propriétaire de la tâche peut également libérer une tâche réclamée.

Procédure

1. Répertoriez les tâches réclamées possédées par une personne spécifique, par exemple, Smith.

```

QueryResultSet result =
    task.query("DISTINCT TASK.TKIID",
              "TASK.STATE = TASK.STATE.STATE_CLAIMED AND
              TASK.OWNER = 'Smith'",
              (String)null, (Integer)null, (TimeZone)null);

```

Cette opération renvoie un ensemble de résultats de requête répertoriant les tâches réclamées par cette personne, Smith.

2. Libérez la tâche réclamée.

```

if (result.size() > 0)
{
    result.first();
    TKIID tkiid = (TKIID) result.getOID(1);
    task.cancelClaim(tkiid, true);
}

```

Cette opération renvoie la tâche à l'état prêt de manière à ce qu'elle puisse être réclamée par l'un des autres propriétaires éventuels. Toute donnée de sortie définie par le propriétaire d'origine est maintenue.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Gestion des tâches élémentaires

Durant la durée de vie d'une instance d'activité ou de tâche, l'ensemble des personnes associées à l'objet peut changer, par exemple, si une personne est en congé, si de nouvelles personnes sont engagées ou si la charge de travail doit être redistribuée. Pour autoriser ces modifications, vous devez développer des applications afin de créer, supprimer ou transférer les tâches élémentaires.

A propos de cette tâche

Une tâche élémentaire correspond à l'affectation d'un objet à un utilisateur ou à un groupe d'utilisateurs pour un motif particulier. Cet objet est généralement une instance d'activité humaine, une instance de processus ou une instance de tâche. Les motifs sont dérivés du rôle conféré à l'utilisateur pour l'objet. Un objet peut comporter plusieurs éléments de travail étant donné qu'un utilisateur peut avoir différents rôles associés à l'objet, et qu'un élément de travail est créé pour chacun de ces rôles. Une instance de tâche à effectuer peut par exemple avoir un élément de travail administrateur, lecteur, éditeur et propriétaire en même temps.

Les actions pouvant être menées pour gérer les tâches élémentaires dépendent du rôle de l'utilisateur : par exemple, un administrateur peut créer, supprimer et transférer des tâches élémentaires, alors que le propriétaire de la tâche ne peut que transférer des tâches élémentaires.

- Créez une tâche élémentaire.

```
// query the task instance for which an additional
// administrator is to be specified
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                   (String)null, (Integer)null,
                                   (TimeZone)null);

if (result.size() > 0)
{
    result.first();
    // create the work item
    task.createWorkItem((TKIID)(result.getOID(1)),
                       WorkItem.REASON_ADMINISTRATOR, "Smith");
}
```

Cette opération crée une tâche élémentaire pour l'utilisateur Smith qui a un rôle d'administration.

- Supprimez une tâche élémentaire.

```
// query the task instance for which a work item is to be deleted
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                   (String)null, (Integer)null,
                                   (TimeZone)null);

if (result.size() > 0)
{
    result.first();
    // delete the work item
    task.deleteWorkItem((TKIID)(result.getOID(1)),
                       WorkItem.REASON_READER, "Smith");
}
```

Cette opération supprime la tâche élémentaire pour l'utilisateur Smith qui a un rôle de lecteur.

- Transférez une tâche élémentaire.

```
// query the task that is to be rescheduled
QueryResultSet result =
    task.query("DISTINCT TASK.TKIID",
              "TASK.NAME='CustomerOrder' AND
              TASK.STATE=TASK.STATE.STATE_READY AND
              WORK_ITEM.REASON=WORK_ITEM.REASON.POTENTIAL_OWNER AND
              WORK_ITEM.OWNER_ID='Miller'",
              (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
    result.first();
    // transfer the work item from user Miller to user Smith
}
```

```

// so that Smith can work on the task
task.transferWorkItem((TKIID)(result.getOID(1)),
                    WorkItem.REASON_POTENTIAL_OWNER,"Miller","Smith");
}

```

Cette opération transfère la tâche élémentaire à l'utilisateur Smith de manière à ce qu'il puisse travailler avec.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

A propos de cette tâche

Cette opération peut être nécessaire, par exemple, quand la définition de tâche n'est pas disponible lors du déploiement de l'application, quand les tâches d'une procédure ne sont pas encore connues ou quand une tâche est requise pour mener à bien une collaboration ad hoc dans un groupe.

Vous pouvez modéliser les tâches à effectuer ou les tâches de collaboration ad-hoc en créant des instances de la classe `com.ibm.task.api.TaskModel`, et les utiliser pour créer un modèle de tâche réutilisable ou créer directement une instance de tâche à exécution unique. Pour créer une instance de la classe `TaskModel`, un ensemble de méthodes de fabrique est disponible dans la classe de fabrique `com.ibm.task.api.ClientTaskFactory`. La modélisation des tâches utilisateur lors de l'exécution se base sur EMF (Eclipse Modeling Framework).

Procédure

1. Créez un ensemble de ressources `org.eclipse.emf.ecore.resource.ResourceSet` à l'aide de la méthode de fabrique `createResourceSet`.
2. Facultatif : Si vous avez l'intention d'utiliser des types de message complexes, vous pouvez soit les définir à l'aide de `org.eclipse.xsd.XSDFactory`, que vous pouvez obtenir grâce à la méthode de fabrique `getXSDFactory()`, soit importer directement un schéma XML existant à l'aide de la méthode de fabrique `loadXSDSchema`.

Pour rendre les types complexes disponibles au serveur WebSphere Process Server, déployez-les dans le cadre d'une application d'entreprise.

3. Créez ou importez une définition WSDL (Web Services Definition Language) du type `javax.wsdl.Definition`.

Vous pouvez créer une nouvelle définition WSDL à l'aide de la méthode `createWSDLDefinition`. Puis vous pouvez lui ajouter un type de port et une opération. Vous pouvez également importer directement une définition WSDL existante à l'aide de la méthode de fabrique `loadWSDLDefinition`.

4. Créez la définition de tâche à l'aide de la méthode de fabrique `createTTask`.

Si vous voulez ajouter ou manipuler des éléments de tâche plus complexes, vous pouvez utiliser la classe `com.ibm.wbit.tel.TaskFactory` que vous pouvez récupérer à l'aide de la méthode de fabrique `getTaskFactory`.

5. Créez le modèle de tâche en utilisant la méthode de fabrique `createTaskModel`, puis envoyez-lui le regroupement de ressources que vous avez créé à l'étape 1 et qui rassemble tous les autres artefacts que vous avez créés depuis lors.
6. Facultatif : Validez le modèle à l'aide de la méthode `TaskModel.validate`.

Résultats

Utilisez l'une des méthodes `create` de l'API EJB Human Task Manager dont le paramètre `TaskModel` permet de créer un modèle de tâche réutilisable ou de créer directement une instance de tâche à exécution unique.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Création de tâches d'exécution utilisant des types Java simples

Cet exemple crée une tâche d'exécution utilisant des types Java simples, comme un objet `String`, dans son interface.

Création de tâches d'exécution utilisant des types complexes

Cet exemple crée une tâche d'exécution utilisant des types complexes dans son interface. Les types complexes sont déjà définis, c'est-à-dire que le système de fichiers local du client possède des fichiers XSD contenant la description des types complexes.

Création de tâches d'exécution utilisant une interface existante

Cet exemple crée une tâche d'exécution utilisant une interface déjà définie, c'est-à-dire que le système de fichiers local possède un fichier contenant la description de l'interface.

Création de tâches d'exécution utilisant une interface à partir d'une application d'appel

Cet exemple crée une tâche d'exécution utilisant une interface appartenant à l'application d'appel. Par exemple, une tâche d'exécution est créée dans un fragment de code Java d'un processus métier et utilise une interface à partir de l'application de processus.

Création de tâches d'exécution utilisant des types Java simples :

Cet exemple crée une tâche d'exécution utilisant des types Java simples, comme un objet `String`, dans son interface.

A propos de cette tâche

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

Procédure

1. Accédez à `ClientTaskFactory` et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Créez la définition WSDL et ajoutez les descriptions des opérations.


```

// Création de l'interface WSDL
Definition definition = factory.createWSDLDefinition
    ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

// Création d'un type de port
PortType portType = factory.createPortType( definition, "doItPT" );

// Création d'une opération ; les messages d'entrée et de sortie sont de type
Chaîne :
// aucun message d'erreur n'est spécifié
Operation operation = factory.createOperation
    ( definition, portType, "doIt",
      new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
      new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
      (Map)null );

```

3. Créez le modèle EMF de la nouvelle tâche utilisateur.

Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```

TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );

```

Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche utilisateur.

```

// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

```

```

// Extraction de la fabrique de tâches pour créer ou modifier les éléments de
tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

```

```

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

```

```

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

```

```

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);

```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

```

TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );

```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```

ValidationProblem[] validationProblems = taskModel.validate();

```

7. Créez l'instance ou le modèle de tâche d'exécution.

L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Du fait que l'application utilise des types Java simples uniquement, il est inutile de spécifier un nom d'application.

- Le fragment de code suivant crée une instance de tâche :


```
atask.createTask( taskModel, (String)null, "HTM" );
```
- Le fragment de code suivant crée un modèle de tâche :

```
task.createTaskTemplate( taskModel, (String)null );
```

Résultats

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Tâches associées

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

Création de tâches d'exécution utilisant des types complexes :

Cet exemple crée une tâche d'exécution utilisant des types complexes dans son interface. Les types complexes sont déjà définis, c'est-à-dire que le système de fichiers local du client possède des fichiers XSD contenant la description des types complexes.

A propos de cette tâche

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

Procédure

1. Accédez à ClientTaskFactory et créer un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Ajoutez les définitions XSD de vos types complexes à l'ensemble de ressources pour les mettre à votre disposition lors de la définition d'opérations.

Les fichiers sont relatifs à l'emplacement d'exécution du code.

```
factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
```

3. Créez la définition WSDL et ajoutez les descriptions des opérations.

```
// Création de l'interface WSDL
Definition definition = factory.createWSDLDefinition
    ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );
```

```
// Création d'un type de port
PortType portType = factory.createPortType( definition, "doItPT" );
```

```
// Création d'une opération ; le message d'entrée est un objet InputBO,
// le message de sortie un objet OutputBO ;
// aucun message d'erreur n'est spécifié
Operation operation = factory.createOperation
    ( definition, portType, "doIt",
      new QName( "http://Input", "InputBO" ),
      new QName( "http://Output", "OutputBO" ),
      (Map)null );
```

4. Créez le modèle EMF de la nouvelle tâche utilisateur.

Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

5. Modifiez les propriétés du modèle de tâche utilisateur.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package,
// par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// Extraction de la fabrique de tâches pour créer ou modifier les éléments de
// tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. Créer le modèle de tâche contenant toutes les définitions de ressources.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

7. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

8. Créez l'instance ou le modèle de tâche d'exécution.

L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles. L'application doit également contenir une tâche ou un processus factice permettant son chargement par Business Process Choreographer.

- Le fragment de code suivant crée une instance de tâche :
task.createTask(taskModel, "B0application", "HTM");
- Le fragment de code suivant crée un modèle de tâche :
task.createTaskTemplate(taskModel, "B0application");

Résultats

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Tâches associées

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

Création de tâches d'exécution utilisant une interface existante :

Cet exemple crée une tâche d'exécution utilisant une interface déjà définie, c'est-à-dire que le système de fichiers local possède un fichier contenant la description de l'interface.

A propos de cette tâche

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

Procédure

1. Accédez à ClientTaskFactory et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Accédez à la définition WSDL et aux descriptions des opérations.

La description d'interface est relative à l'emplacement d'exécution du code.

```
Definition definition = factory.loadWSDLDefinition(
    resourceSet, "interface.wsdl" );
PortType portType = definition.getPortType(
    new QName( definition.getTargetNamespace(), "doItPT" ) );
Operation operation = portType.getOperation(
    "doIt", (String)null, (String)null);
```

3. Créez le modèle EMF de la nouvelle tâche utilisateur.

Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```
TTask humanTask = factory.createTTask( resourceSet,
    TTaskKinds.HTASK_LITERAL,
    "TestTask",
    new UTCDate( "2005-01-01T00:00:00" ),
    "http://www.ibm.com/task/test/",
    portType,
    operation );
```

Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche utilisateur.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );
```

```
// Extraction de la fabrique de tâches pour créer ou modifier les éléments de
tâches composites
TaskFactory taskFactory = factory.getTaskFactory();
```

```
// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");
```

```
// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);
```

```
// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. Créez l'instance ou le modèle de tâche d'exécution.

L'interface `HumanTaskManagerService` permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles. L'application doit également contenir une tâche ou un processus factice permettant son chargement par `Business Process Choreographer`.

- Le fragment de code suivant crée une instance de tâche :

```
task.createTask( taskModel, "B0application", "HTM" );
```

- Le fragment de code suivant crée un modèle de tâche :

```
task.createTaskTemplate( taskModel, "B0application" );
```

Résultats

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Tâches associées

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme `WebSphere Integration Developer`, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans `WebSphere Process Server` et créez des instances à partir de ces modèles en utilisant, par exemple, `Business Process Choreographer Explorer`. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

Création de tâches d'exécution utilisant une interface à partir d'une application d'appel :

Cet exemple crée une tâche d'exécution utilisant une interface appartenant à l'application d'appel. Par exemple, une tâche d'exécution est créée dans un fragment de code Java d'un processus métier et utilise une interface à partir de l'application de processus.

A propos de cette tâche

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

Procédure

1. Accédez à `ClientTaskFactory` et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```

ClientTaskFactory factory = ClientTaskFactory.newInstance();

// Spécification du chargeur de classe de contexte pour rechercher les ressources
suivantes
ResourceSet resourceSet = factory.createResourceSet
    ( Thread.currentThread().getContextClassLoader() );

```

2. Accédez à la définition WSDL et aux descriptions des opérations.

Indiquez le chemin d'accès à l'intérieur du fichier JAR de package contenant.

```

Definition definition = factory.loadWSDLDefinition( resourceSet,
    "com/ibm/workflow/metaflow/interface.wsdl" );
PortType portType = definition.getPortType(
    new QName( definition.getTargetNamespace(), "doItPT" ) );
Operation operation = portType.getOperation
    ("doIt", (String)null, (String)null);

```

3. Créez le modèle EMF de la nouvelle tâche utilisateur.

Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```

TTask humanTask = factory.createTTask( resourceSet,
    TTaskKinds.HTASK_LITERAL,
    "TestTask",
    new UTCDate( "2005-01-01T00:00:00" ),
    "http://www.ibm.com/task/test/",
    portType,
    operation );

```

Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche utilisateur.

```

// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

```

```

// Extraction de la fabrique de tâches pour créer ou modifier les éléments de
tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

```

```

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

```

```

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

```

```

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);

```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

```

TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );

```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```

ValidationProblem[] validationProblems = taskModel.validate();

```

7. Créez l'instance ou le modèle de tâche d'exécution.

L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles.

- Le fragment de code suivant crée une instance de tâche :

```
task.createTask( taskModel, "WorkflowApplication", "HTM" );
```
- Le fragment de code suivant crée un modèle de tâche :

```
task.createTaskTemplate( taskModel, "WorkflowApplication" );
```

Résultats

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Tâches associées

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche utilisateur ou de participation lors de l'exécution.

Interface HumanTaskManagerService

L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Différentes méthodes peuvent être appelées selon l'état de la tâche et les droits d'accès de l'utilisateur de l'application contenant la méthode en question. Les méthodes principales de manipulation des objets de tâche sont répertoriées dans cette rubrique. Plus plus d'information sur ces méthodes et d'autres méthodes fournies par l'interface HumanTaskManagerService, consultez Javadoc dans le package com.ibm.task.api.

Modèles de tâches

Les méthodes suivantes sont disponibles pour les modèles de tâches.

Tableau 46. Méthodes API pour les modèles de tâches

Méthode	Description
getTaskTemplate	Extrait le modèle de tâche spécifié.
createAndCallTask	Crée et exécute une instance de tâche à partir du modèle de tâche et attend le résultat de façon synchrone.
createAndStartTask	Crée et démarre une instance de tâche à partir du modèle de tâche spécifié.
createTask	Crée une instance de tâche à partir du modèle de tâche spécifié.
createInputMessage	Crée un message d'entrée pour le modèle de tâche indiqué. Par exemple, crée un message pouvant servir à démarrer une tâche.
queryTaskTemplates	Extrait des modèles de tâche stockés dans la base de données.

Instances de tâches

Les méthodes suivantes sont disponibles pour les instances de tâches.

Tableau 47. Méthodes API pour les modèles de tâches

Méthode	Description
getTask	Extrait une instance de tâche ; l'instance de tâche peut se trouver dans n'importe quel état.
callTask	Démarre une tâche d'appel en mode synchrone.
startTask	Démarre une tâche qui a déjà été créée.
suspend	Interrompt la tâche de collaboration ou la tâche à effectuer.
resume	Reprend la tâche de collaboration ou la tâche à effectuer.
terminate	Arrête l'instance de tâche spécifiée. Si une tâche d'appel est arrêtée, cette action n'a aucun impact sur le service appelé.
delete	Supprime l'instance de tâche spécifiée.
claim	Réclame la tâche en vue de son traitement.
update	Met à jour l'instance de tâche.
complete	Termine l'instance de tâche.
cancelClaim	Libère une instance de tâche réclamée afin de permettre son traitement par un autre propriétaire potentiel.
createWorkItem	Crée un élément de travail pour l'instance de tâche.
transferWorkItem	Transfère l'élément de travail à un propriétaire spécifié.
deleteWorkItem	Supprime l'élément de travail.

Escalades

Les méthodes suivantes sont disponibles pour les escalades.

Tableau 48. Méthodes API de gestion des escalades

Méthode	Description
getEscalation	Extrait l'instance d'escalade spécifiée.

Propriétés personnalisées

Les tâches, les modèles de tâche et les escalades peuvent tous posséder des propriétés personnalisées. L'interface fournit une méthode `get` et une méthode `set` pour l'extraction et la définition de valeurs des propriétés personnalisées. Vous pouvez aussi associer les propriétés mentionnées aux instances de tâche et les en extraire. Le noms de propriétés personnalisées et des valeurs doivent être de type `java.lang.String`. Les méthodes suivantes sont adaptées aux tâches, modèles de tâche et escalades.

Tableau 49. Méthodes API pour les variables et les propriétés personnalisées

Méthode	Description
getCustomProperty	Extrait la propriété personnalisée mentionnée de l'instance de tâche spécifiée.
getCustomProperties	Extrait les propriétés personnalisées de l'instance de tâche spécifiée.
getCustomPropertyNames	Extrait les noms des propriétés personnalisées pour l'instance de tâche.
setCustomProperty	Stocke les valeurs spécifiques aux propriétés personnalisées correspondant à l'instance de tâche spécifiée.

Tâches associées

Développement d'applications pour des tâches utilisateur

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches utilisateur sont fournis.

Référence associée

Actions autorisées pour les tâches

Les actions pouvant être effectuées sur une tâche varient selon qu'il s'agit d'une tâche à effectuer, d'une tâche collaborative, d'une tâche d'appel ou d'une tâche d'administration.

Actions autorisées pour les tâches :

Les actions pouvant être effectuées sur une tâche varient selon qu'il s'agit d'une tâche à effectuer, d'une tâche collaborative, d'une tâche d'appel ou d'une tâche d'administration.

Vous ne pouvez pas utiliser toutes les actions disponibles à travers l'interface HumanTaskManager sur tous les types de tâche. Le tableau suivant indique les actions que vous pouvez effectuer sur chaque type de tâche.

Action	Type de tâche			
	Tâche à effectuer	Tâche de collaboration	Tâche d'appel	Tâche d'administration
callTask			X	
cancelClaim	X	X ¹		
claim	X	X ¹		
complete	X	X ¹		X
completeWithFollowOnTask ⁴	X	X ¹		
completeWithFollowOnTask ⁵		X ³	X ³	
createFaultMessage	X	X	X	X
createInputMessage	X	X	X	X
createOutputMessage	X	X	X	X
createWorkItem	X	X ¹	X	X
delete	X ¹	X ¹	X	X ¹
deleteWorkItem	X	X ¹	X	X
getCustomProperty	X	X ¹	X	X

Action	Type de tâche			
	Tâche à effectuer	Tâche de collaboration	Tâche d'appel	Tâche d'administration
getDocumentation	X	X ¹	X	X
getFaultNames	X	X ¹		
getFaultMessage	X	X ¹	X	
getInputMessage	X	X ¹	X	
getOutputMessage	X	X ¹	X	
getUsersInRole	X	X ¹	X	X
getTask	X	X ¹	X	X
getUISettings	X	X ¹	X	X
resume	X	X ¹		
setCustomProperty	X	X ¹	X	X
setFaultMessage	X	X ¹		
setOutputMessage	X	X ¹		
startTask	X ¹	X ¹	X	X
startTaskAsSubtask ⁶	X	X ¹		
startTaskAsSubtask ⁷		X ³	X ³	
suspend	X	X ¹		
suspendWithCancelClaim	X	X ¹		
terminate	X ¹	X ¹	X ¹	
transferWorkItem	X	X ¹	X	X
update	X	X ¹	X	X
Remarques :				
1. Uniquement pour les tâches autonomes, ad-hoc et les modèles de tâches				
2. Uniquement pour les tâches autonomes, en ligne intégrées aux processus métier et ad-hoc				
3. Uniquement pour les tâches autonomes et ad-hoc				
4. Les types de tâches pouvant comporter des tâches de suivi				
5. Les types de tâches pouvant être utilisés en tant que tâches de suivi				
6. Les types de tâches pouvant posséder des sous-tâches				
7. Les types de tâches pouvant être utilisés en tant que sous-tâches				

Référence associée

Interface HumanTaskManagerService

L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Développement d'applications pour les processus métier et les tâches utilisateur

La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction humaine lorsque le processus est démarré ou géré ou lorsque des activités humaines sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

A propos de cette tâche

Pour impliquer des personnes dans des scénarios de processus métier, vous pouvez inclure les types de tâche suivants dans le processus métier :

- Une tâche d'appel en ligne (également appelée *tâche de départ* dans l'API).
Vous pouvez fournir une tâche d'appel pour chaque activité de réception, pour chaque élément `onMessage` de l'activité de sélection et pour chaque élément `onEvent` du gestionnaire d'événements. Cette tâche peut alors contrôler les utilisateurs autorisés à démarrer un processus ou à communiquer avec une instance de processus en cours d'exécution.
- Une tâche d'administration.
Vous pouvez fournir une tâche d'administration afin d'indiquer qui est autorisé à administrer le processus ou à effectuer des opérations d'administration sur les activités du processus qui ont échoué.
- Une tâche à effectuer (également appelée *tâche de participation* dans l'API).
Les tâches à effectuer implémentent une activité humaine. Ce type d'activité vous permet de faire participer des utilisateurs au processus.

Les activités humaines du processus métier représentent les tâches à effectuer réalisées par les utilisateurs dans le scénario de processus métier. Pour réaliser de tels scénarios, vous pouvez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

- Le processus métier est le conteneur de toutes les activités appartenant au processus, y compris les activités humaines qui sont représentées par les tâches à effectuer. Lorsqu'une instance de processus est créée, un ID objet unique (PIID) lui est affecté.
- Lorsqu'une activité humaine est activée au cours de l'exécution de l'instance de processus, une instance d'activité est créée, qui est identifiée par son ID objet (AIID) unique. En même temps, une instance de tâche à effectuer en ligne est également créée, qui est identifiée par son ID objet (TKIID). La relation entre l'activité humaine et l'instance de tâche est créée par le biais des ID objet :
 - L'ID tâche à effectuer de l'instance d'activité est défini en fonction du TKIID de la tâche à effectuer associée.
 - L'ID de contexte de confinement de l'instance de tâche est défini en fonction de l'instance de processus qui contient l'instance d'activité associée.
 - L'ID de contexte parent de l'instance de tâche est défini en fonction de l'AIID de l'instance d'activité associée.
- Les cycles de vie de toutes les instances de tâche à effectuer en ligne sont gérés par l'instance de processus. Lorsque l'instance de processus est supprimée, les instances de tâches le sont également. En d'autres termes, toutes les tâches dont l'ID de contexte de confinement est défini en fonction du PIID de l'instance de processus sont automatiquement supprimées.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Déterminer les modèles de processus ou les activités pouvant être démarrés

Un processus métier peut être démarré en appelant les méthodes call, initiate ou sendMessage de l'API de Business Flow Manager. Si le processus n'a qu'une seule activité de démarrage, vous pouvez utiliser la signature de méthode dont le paramètre doit être un nom de modèle de processus. Si le processus comporte plusieurs activités de démarrage, vous devez identifier l'activité de démarrage de manière explicite.

Traitement par une seule personne d'un flux de travaux contenant des tâches utilisateur

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple démontre comment implémenter sous forme d'une série d'activités humaines (tâches à effectuer) la séquence d'actions nécessaires pour commander un livre. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

Déterminer les modèles de processus ou les activités pouvant être démarrés

Un processus métier peut être démarré en appelant les méthodes call, initiate ou sendMessage de l'API de Business Flow Manager. Si le processus n'a qu'une seule activité de démarrage, vous pouvez utiliser la signature de méthode dont le paramètre doit être un nom de modèle de processus. Si le processus comporte plusieurs activités de démarrage, vous devez identifier l'activité de démarrage de manière explicite.

A propos de cette tâche

Lorsqu'un processus métier est modélisé, le modélisateur peut décider que seul un sous-ensemble d'utilisateurs est autorisé à créer une instance de processus à partir du modèle de processus. Ceci est effectué en associant une tâche d'appel en ligne à une activité de démarrage du processus, puis en précisant les restrictions d'autorisation appliquées à cette tâche. Seuls les utilisateurs qui sont des démarreurs ou des administrateurs potentiels de la tâche sont autorisés à créer une instance de la tâche, et par conséquent, une instance du modèle de processus.

Si aucune tâche d'appel en ligne n'est associée à l'activité de démarrage, ou si les restrictions d'autorisation ne sont pas indiquées pour la tâche, tous les utilisateurs peuvent créer une instance de processus à l'aide de l'activité de démarrage.

Un processus peut avoir plusieurs activités de démarrage, chacune avec différentes requêtes d'utilisateurs pour des démarreurs ou des administrateurs potentiels. Cela signifie qu'un utilisateur peut être autorisé à démarrer un processus avec l'activité A, mais pas avec l'activité B.

Procédure

1. Utilisez l'API de Business Flow Manager pour créer la liste des versions courantes des modèles de processus qui sont à l'état démarré.

Conseil : La méthode `queryProcessTemplates` exclut uniquement les modèles de processus qui font partie des applications n'ayant pas encore démarré. Par conséquent, si vous utilisez cette méthode sans filtrer les résultats, vous obtiendrez toutes les versions des modèles de processus indépendamment de l'état dans lequel ils se trouvent.

```
// current timestamp in UTC format, converted to yyyy-mm-ddThh:mm:ss
String now = (new UTCDate()).toXsdString();
String whereClause = "PROCESS_TEMPLATE.STATE =
                    PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                    PROCESS_TEMPLATE.VALID_FROM =
                    (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                     WHERE NAME=PROCESS_TEMPLATE.NAME AND
                     VALID_FROM <= TS('" + now + "'))";

ProcessTemplateData[] processTemplates = process.queryProcessTemplates
    ( whereClause,
      "PROCESS_TEMPLATE.NAME",
      (Entier)null, (FuseauHoraire)null);
```

Les résultats sont triés par nom de modèle de processus.

2. Créez la liste des modèles de processus et celle des activités de démarrage pour lesquelles l'utilisateur est autorisé.

La liste des modèles de processus contient les modèles de processus ayant une activité de démarrage unique. Soit ces activités sont non protégées, soit l'utilisateur connecté est autorisé à les démarrer. Sinon, vous pouvez regrouper les modèles de processus qui peuvent être démarrés par au moins une activité de démarrage.

Conseil : Un administrateur de processus peut également créer une instance de processus. Pour obtenir la liste complète des modèles, vous devez aussi lire le modèle de tâche d'administration qui est associé au modèle de processus, puis vérifier si l'utilisateur est connecté en tant qu'administrateur.

```
List authorizedProcessTemplates = new ArrayList();
List authorizedActivityServiceTemplates = new ArrayList();
```

3. Déterminez les activités de démarrage pour chacun des modèles de processus.

```
for( int i=0; i<processTemplates.length; i++ )
{
    ProcessTemplateData template = processTemplates[i];
    ActivityServiceTemplateData[] startActivities =
        process.getStartActivities(template.getID());
```

4. Pour chaque activité de démarrage, récupérez l'ID du modèle de tâche d'appel en ligne associé.

```
for( int j=0; j<startActivities.length; j++ )
{
    ActivityServiceTemplateData activity = startActivities[j];
    TKTID tktid = activity.getTaskTemplateID();
```

- a. Si un modèle de tâche d'appel n'existe pas, cela signifie que le modèle de processus n'est pas sécurisé par cette activité de démarrage.

Dans pareil cas, tout utilisateur peut créer une instance de processus à l'aide de cette activité de démarrage.

```
boolean isAuthorized = false;
    if ( tktid == null )
    {
        isAuthorized = true;
        authorizedActivityServiceTemplates.add(activity);
    }
```

- b. Si un modèle de tâche d'appel existe, utilisez l'API de Human Task Manager pour vérifier les autorisations dont dispose l'utilisateur connecté.

Dans l'exemple, l'utilisateur connecté s'appelle Smith. Il est impératif que l'utilisateur connecté soit un démarreur potentiel de la tâche d'appel ou un administrateur.

```
if ( tktid != null )
{
    isAuthorized =
        task.isUserInRole
            (tkid, "Smith", WorkItem.REASON_POTENTIAL_STARTER) ||
            task.isUserInRole(tktid, "Smith", WorkItem.REASON_ADMINISTRATOR);

    if ( isAuthorized )
    {
        authorizedActivityServiceTemplates.add(activity);
    }
}
```

Si l'utilisateur correspond au rôle indiqué ou si les critères d'affectation des utilisateurs pour ce rôle ne sont pas définis, la méthode `isUserInRole` renvoie la valeur `true`.

5. Vérifiez s'il est possible de démarrer le processus à l'aide du nom du modèle de processus uniquement.

```
if ( isAuthorized && startActivities.length == 1 )
{
    authorizedProcessTemplates.add(template);
}
```

6. Arrêtez les boucles.

```
    } // end of loop for each activity service template
} // end of loop for each process template
```

Tâches associées

Développement d'applications pour les processus métier et les tâches utilisateur. La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction humaine lorsque le processus est démarré ou géré ou lorsque des activités humaines sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

Traitement par une seule personne d'un flux de travaux contenant des tâches utilisateur

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple démontre comment implémenter sous forme d'une série d'activités humaines (tâches à effectuer) la séquence d'actions nécessaires pour commander un livre. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

A propos de cette tâche

Dans une librairie en ligne, l'acheteur accomplit une série d'actions afin de commander un ouvrage. Cette séquence d'actions peut être implémentée comme une série d'activités humaines (tâches à accomplir). Si l'acheteur décide de commander plusieurs livres, cela équivaut à réclamer l'activité humaine suivante. Les informations sur la séquence de tâches sont gérées par le Business Flow Manager, alors que les tâches elles-mêmes sont gérées par le Human Task Manager.

Comparez cet exemple avec celui qui utilise uniquement l'API de Business Flow Manager.

Procédure

1. Utilisez l'API de Business Flow Manager pour accéder à l'instance de processus que vous voulez traiter.

Dans cet exemple, il s'agit d'une instance du processus CustomerOrder.

```
ProcessInstanceData processInstance =  
    process.getProcessInstance("CustomerOrder");  
String piid = processInstance.getID().toString();
```

2. Utilisez l'API de Human Task Manager pour interroger les tâches à effectuer prêtes (de type tâche de participation) qui font partie de l'instance de processus indiquée.

Utilisez l'ID de contexte de confinement de la tâche pour spécifier l'instance du processus de confinement. Pour un flux de travaux exécuté par une seule personne, la requête renvoie la tâche à effectuer qui est associée à la première activité manuelle dans la séquence d'activités manuelles.

```
//  
// Query the list of to-do tasks that can be claimed by the logged-on user  
// for the specified process instance  
//  
QueryResultSet result =  
    task.query("DISTINCT TASK.TKIID",  
              "TASK.CONTAINMENT_CTX_ID = ID('" + piid + "') AND  
              TASK.STATE = TASK.STATE.STATE_READY AND  
              TASK.KIND = TASK.KIND.KIND_PARTICIPATING AND  
              WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",  
              (String)null, (Integer)null, (TimeZone)null);
```

3. Réclamez la tâche à effectuer qui est renvoyée.

```
if (result.size() > 0)  
{  
    result.first();  
    TKIID tkiid = (TKIID) result.getOID(1);  
    ClientObjectWrapper input = task.claim(tkiid);  
    DataObject activityInput = null ;  
    if ( input.getObject() != null && input.getObject() instanceof DataObject )  
    {  
        taskInput = (DataObject)input.getObject();  
        // read the values  
        ...  
    }  
}
```

Une fois la tâche réclamée, le message d'entrée de la tâche est renvoyé.

4. Déterminez l'activité humaine qui est associée à la tâche à effectuer.

Pour établir une corrélation entre les activités et les tâches correspondantes, vous pouvez utiliser l'une des méthodes suivantes.

- La méthode `task.getActivityID` :

```
AIID aiid = task.getActivityID(tkiid);
```

- L'ID de contexte parent qui fait partie de l'objet tâche :

```
AIID aiid = null;  
Task taskInstance = task.getTask(tkiid);
```

```
OID oid = taskInstance.getParentContextID();  
if ( oid != null and oid instanceof AIID )  
{  
    aiid = (AIID)oid;  
}
```

5. Lorsque vous avez terminé de traiter la tâche, utilisez l'API de Business Flow Manager pour terminer la tâche ainsi que l'activité humaine qui lui est associée, puis réclamez l'activité humaine suivante dans l'instance de processus.

Pour terminer l'activité humaine, un message de sortie est transmis. Lorsque vous créez le message de sortie, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
    process.createMessage(aiid, activity.getOutputMessageType());
DataObject myMessage = null ;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
    myMessage = (DataObject)output.getObject();
    //set the parts in your message, for example, an order number
    myMessage.setInt("OrderNo", 4711);
}

//complete the human task activity and its associated to-do task,
// and claim the next human task activity
CompleteAndClaimSuccessorResult successor =
    process.completeAndClaimSuccessor(aiid, output);
```

Cette opération définit un message de sortie contenant le numéro de commande et réclame l'activité humaine suivante de la séquence. Si `AutoClaim` est défini pour les activités de succession et que plusieurs chemins d'accès peuvent être utilisés, toutes les activités de succession sont réclamées et une activité aléatoire est renvoyée en tant qu'activité suivante. Si aucune activité de succession supplémentaire ne peut être affectée à cet utilisateur, la valeur `Null` est renvoyée.

Si le processus contient des chemins parallèles pouvant être suivis, que ces chemins contiennent des activités humaines et que l'utilisateur connecté est le propriétaire potentiel de plusieurs de ces activités, une activité aléatoire est automatiquement réclamée et renvoyée comme activité suivante.

6. Exécutez l'activité humaine suivante.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject() !=
    null && nextInput.getObject() instanceof DataObject )
{
    activityInput = (DataObject)input.getObject();
    // read the values
    ...
}

aiid = successor.getAIID();
```

7. Passez à l'étape 5 afin de terminer l'activité humaine et de récupérer l'activité humaine suivante.

Tâches associées

Développement d'applications pour les processus métier et les tâches utilisateur
La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction humaine lorsque le processus est démarré ou géré ou lorsque des activités humaines sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

Traitement d'un flux de travaux par une seule personne

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. L'API `completeAndClaimSuccessor` prend en charge le traitement de ce type de flux de travaux.

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

A propos de cette tâche

Les erreurs BPEL (Business Process Execution Language) proviennent des éléments suivants :

- Appels de service Web (erreurs WSDL (Web Services Description Language))
- Activités d'émission
- Erreurs standard BPEL reconnues par Business Process Choreographer

Il existe des mécanismes pour gérer ces erreurs : Pour résoudre les erreurs liées à une instance de processus, utilisez l'un des mécanismes suivants :

- Transférez le contrôle aux gestionnaires d'erreur correspondants
- Effectuez une compensation du travail précédent du processus
- Arrêtez le processus afin de laisser quelqu'un d'autre remédier à la situation (forcer la nouvelle tentative, forcer à terminer)

Un processus BPEL peut également renvoyer des erreurs à l'appelant d'une opération fournie par le processus. Vous pouvez modéliser l'erreur dans le processus sous forme d'activité de réponse avec un nom d'erreur et des données d'erreur. Ces erreurs sont renvoyées à l'appelant API sous forme d'exceptions vérifiées.

Si un processus BPEL ne gère pas d'erreurs BPEL ou si une exception API survient, une exception d'exécution est renvoyée à l'appelant de l'API. Par exemple, une exception API est lancée lorsque le modèle de processus à partir duquel une instance doit être créée n'existe pas.

La gestion des erreurs et des exceptions est décrite dans les tâches suivantes.

Tâches associées

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches utilisateur installées sur WebSphere Process Server.

Gestion des exceptions de l'API EJB de Business Process Choreographer

Si une méthode de l'interface BusinessFlowManagerService ou HumanTaskManagerService ne se termine pas correctement, une exception est générée, indiquant la cause de l'erreur. Vous pouvez gérer cette exception de manière spécifique pour guider l'appelant.

Vérification de l'erreur définie pour une activité de tâche utilisateur

Lorsqu'une activité de tâche utilisateur est traitée, elle peut s'exécuter correctement. Dans ce cas, vous pouvez transmettre un message de sortie. Si l'activité de tâche utilisateur ne se termine pas correctement, vous pouvez transmettre un message d'erreur.

Vérification d'une erreur survenue lors d'une activité d'appel arrêtée

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par des gestionnaires d'erreur. Vous pouvez extraire les informations relatives à l'exception ou à l'erreur qui s'est produite pour une activité d'appel provenant de l'instance d'activité.

Vérification de l'erreur ou de l'exception non gérée survenue lors de l'échec d'une instance de processus

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par un gestionnaire d'erreur. Si le processus implémente une opération bi-directionnelle, vous pouvez extraire des informations sur une erreur ou une exception gérée à partir de la propriété du nom de l'erreur de l'objet de l'instance de processus. Pour les erreurs, vous pouvez également extraire le message d'erreur correspondant à l'aide de l'API getFaultMessage.

Gestion des exceptions de l'API EJB de Business Process Choreographer

Si une méthode de l'interface BusinessFlowManagerService ou HumanTaskManagerService ne se termine pas correctement, une exception est générée, indiquant la cause de l'erreur. Vous pouvez gérer cette exception de manière spécifique pour guider l'appelant.

A propos de cette tâche

Cependant, il est de coutume de gérer uniquement un sous-ensemble des exceptions de manière spécifique et de fournir un guide général pour les autres exceptions potentielles. Toutes les exceptions spécifiques héritent d'une classe générique ProcessException ou TaskException. Interceptez les exceptions génériques avec une instruction finale catch(ProcessException) ou catch(TaskException). Cette instruction permet de veiller à la compatibilité ascendante de votre programme d'application car elle prend en compte toutes les autres exceptions qui peuvent survenir.

Tâches associées

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

Vérification de l'erreur définie pour une activité de tâche utilisateur

Lorsqu'une activité de tâche utilisateur est traitée, elle peut s'exécuter correctement. Dans ce cas, vous pouvez transmettre un message de sortie. Si l'activité de tâche utilisateur ne se termine pas correctement, vous pouvez transmettre un message d'erreur.

A propos de cette tâche

Vous pouvez lire le message d'erreur pour déterminer la cause de l'erreur.

Procédure

1. Répertoirez les activités de tâche se trouvant à l'état d'échec ou arrêté.

```
QueryResultSet result =
    process.query("ACTIVITY.AIID",
        "(ACTIVITY.STATE = ACTIVITY.STATE.STATE_FAILED OR
        ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED) AND
        ACTIVITY.KIND=ACTIVITY.KIND.KIND_STAFF",
        (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête contenant des activités en échec ou arrêtées.

2. Lisez le nom de l'erreur.

```
if (result.size() > 0)
{
    result.first();
    AIID aaid = (AIID) result.getOID(1);
    ClientObjectWrapper faultMessage = process.getFaultMessage(aaid);
    DataObject fault = null ;
    if ( faultMessage.getObject() != null && faultMessage.getObject() instanceof DataObject )
    {
        fault = (DataObject) faultMessage.getObject();
        Type type = fault.getType();
        String name = type.getName();
        String uri = type.getURI();
    }
}
```

Cela renvoie le nom de l'erreur. Vous pouvez aussi analyser l'exception non prise en charge d'une activité arrêtée au lieu d'extraire le nom de l'erreur.

Tâches associées

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

Vérification d'une erreur survenue lors d'une activité d'appel arrêtée

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par des gestionnaires d'erreur. Vous pouvez extraire les informations relatives à l'exception ou à l'erreur qui s'est produite pour une activité d'appel provenant de l'instance d'activité.

A propos de cette tâche

Si une activité entraîne une erreur, le type d'erreur détermine les actions que vous pouvez effectuer pour réparer l'activité.

Procédure

1. Répertoriez les activités humaines qui sont en état arrêté.

```
QueryResultSet result =
    process.query("ACTIVITY.AIID",
        "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
        ACTIVITY.KIND=ACTIVITY.KIND.KIND_INVOKE",
        (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête contenant des activités d'appel arrêtées.

2. Lisez le nom de l'erreur.

```
if (result.size() > 0)
{
    result.first();
    AIID aaid = (AIID) result.getOID(1);
    ActivityInstanceData activity = process.getActivityInstance(aaid);

    ProcessException excp = activity.getUnhandledException();
    if ( excp instanceof ApplicationFaultException )
    {
        ApplicationFaultException fault = (ApplicationFaultException)excp;
        String faultName = fault.getFaultName();
    }
}
```

Tâches associées

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

Vérification de l'erreur ou de l'exception non gérée survenue lors de l'échec d'une instance de processus

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par un gestionnaire d'erreur. Si le processus implémente une opération bi-directionnelle, vous pouvez extraire des informations sur une erreur ou une exception gérée à partir de la propriété du nom de l'erreur de l'objet de l'instance de processus. Pour les erreurs, vous pouvez également extraire le message d'erreur correspondant à l'aide de l'API `getFaultMessage`.

A propos de cette tâche

Si une instance de processus échoue parce qu'une exception n'est pas gérée par l'un des gestionnaire d'erreur, vous pouvez extraire des informations sur l'exception non gérée à partir de l'objet de l'instance de processus. En revanche, si une erreur est interceptée par un gestionnaire d'erreur, les informations sur l'erreur ne sont pas disponibles. Vous pouvez, cependant, extraire le message et le nom de l'erreur et les renvoyer à l'appelant à l'aide de l'exception `FaultReplyException`.

Procédure

1. Répertoriez les instances de processus présentant l'état Echoué.

```
QueryResultSet result =
    process.query("PROCESS_INSTANCE.PIID",
        "PROCESS_INSTANCE.STATE =
        PROCESS_INSTANCE.STATE.STATE_FAILED",
        (String)null, (Integer)null, (TimeZone)null);
```

Cette opération renvoie un ensemble de résultats de requête contenant les instances de processus ayant échoué.

2. Prenez connaissances des informations concernant l'exception non gérée.

```
if (result.size() > 0)
{
    result.first();
    PIID piid = (PIID) result.getOID(1);
    ProcessInstanceData pInstance = process.getProcessInstance(piid);

    ProcessException excp = pInstance.getUnhandledException();
    if ( excp instanceof RuntimeFaultException )
    {
        RuntimeFaultException xcp = (RuntimeFaultException)excp;
        Throwable cause = xcp.getRootCause();
    }
    else if ( excp instanceof StandardFaultException )
    {
        StandardFaultException xcp = (StandardFaultException)excp;
        String faultName = xcp.getFaultName();
    }
    else if ( excp instanceof ApplicationFaultException )
    {
        ApplicationFaultException xcp = (ApplicationFaultException)excp;
        String faultName = xcp.getFaultName();
    }
}
```

Résultats

Utilisez ces informations pour rechercher le nom de l'erreur ou la cause principale du problème.

Tâches associées

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

A propos de cette tâche

Vous pouvez développer des applications client dans n'importe quel environnement client de service Web, y compris les services Web Java et Microsoft .NET.

Concepts associés

Copie d'artefacts

Un certain nombre doivent être copiés depuis l'environnement WebSphere afin de créer des applications client.

Développement d'applications client dans l'environnement de services Web Java

Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Composants de service Web et séquence de contrôle

Un certain nombre de composants côté client et côté serveur font partie de la séquence de contrôle qui représente une requête et une réponse de service Web.

Présentation des API des services Web

Les API des services Web permettent de développer des applications client qui accèdent aux processus métier et aux tâches utilisateur s'exécutant en environnement Business Process Choreographer à l'aide de services Web.

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches utilisateur. Chacune de ces interfaces présente des caractéristiques différentes.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Développement d'applications client

Le processus de développement d'applications client comprend un certain nombre d'étapes.

Requêtes sur des objets liés aux processus métier et aux tâches

Vous pouvez utiliser les API de services Web pour effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données Business Process Choreographer, afin d'extraire les propriétés spécifiques de ces objets.

Composants de service Web et séquence de contrôle

Un certain nombre de composants côté client et côté serveur font partie de la séquence de contrôle qui représente une requête et une réponse de service Web.

Une séquence de contrôle typique se présente comme suit.

1. Côté client :

- a. Une application client (fournie par l'utilisateur) émet une requête de service Web.

- b. Un client proxy (également fourni par l'utilisateur, mais pouvant être généré automatiquement par des utilitaires côté client) encapsule la requête de service dans une enveloppe de requête SOAP.
 - c. L'infrastructure de développement côté client réachemine la requête vers une adresse URL définie en tant que noeud final du service Web.
2. Le réseau transmet la requête au noeud final de service Web via le protocole HTTP ou HTTPS.
3. Côté serveur :
 - a. L'API de service Web générique reçoit la requête et la décode.
 - b. La requête est soit gérée directement par les composants génériques Business Flow Manager ou Human Task Manager, soit transmise au processus métier ou à la tâche utilisateur spécifiés.
 - c. Les données renvoyées sont encapsulées dans une enveloppe de réponse SOAP.
4. Le réseau transmet la réponse à l'environnement côté-client via le protocole HTTP ou HTTPS.
5. De retour côté client :
 - a. L'infrastructure de développement côté client décode l'enveloppe de réponse SOAP.
 - b. Le client proxy extrait les données de la réponse SOAP et les transmet à l'application client.
 - c. L'application client traite les données renvoyées selon les nécessités.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Présentation des API des services Web

Les API des services Web permettent de développer des applications client qui accèdent aux processus métier et aux tâches utilisateur s'exécutant en environnement Business Process Choreographer à l'aide de services Web.

L'API des services Web Business Process Choreographer dispose de deux interfaces de services Web distinctes (types de port WSDL) :

- API Business Flow Manager. Elle permet aux applications client d'avoir une interaction avec des microflux et des processus longue durée, par exemple :
 - Créer des modèles et des instances de processus
 - Réclamer des processus existants
 - Rechercher un processus à partir de son ID

Pour consulter la liste complète des actions possibles, voir «Développement d'applications pour les processus métier», à la page 301.

- API Human Task Manager. Elle permet aux applications client d'effectuer les opérations suivantes :
 - Créer et lancer des tâches
 - Réclamer des tâches existantes
 - Exécuter des tâches
 - Rechercher une tâche à partir de son ID
 - Rechercher un ensemble de tâches.

Pour consulter la liste complète des actions possibles, voir «Développement d'applications pour des tâches utilisateur», à la page 328.

Les applications client peuvent utiliser l'une des interfaces de service Web ou les deux.

Exemple

La structure suivante peut convenir pour une application client qui accède à l'API du service Web Human Task Manager afin de traiter une tâche utilisateur de participation :

1. L'application client envoie un appel de service Web query au serveur WebSphere Process Server demandant la liste des tâches de participation sur lesquelles un utilisateur devra travailler.
2. La liste des tâches de participation est renvoyée dans une enveloppe de réponse SOAP/HTTP.
3. L'application client envoie alors un appel de service Web claim pour demander l'une des tâches de participation.
4. WebSphere Process Server renvoie le message d'entrée de la tâche.
5. L'application client envoie un appel de service Web complete pour achever la tâche par un message de sortie ou d'erreur.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Exigences en termes de processus métier et de tâches utilisateur

Les processus métier et les tâches utilisateur développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via les API de services Web.

Les exigences sont les suivantes :

1. Les interfaces des processus métier et des tâches utilisateur doivent être définies à l'aide du style "document/literal wrapped" défini dans l'API Java pour la spécification XML-RPC (JAX-RPC 1.1). Il s'agit du style par défaut défini pour l'ensemble des processus métier et des tâches utilisateur développés avec l'ID de poste de travail.
2. Les messages d'erreur accessibles aux processus métier et aux tâches utilisateur des opérations de service Web doivent comprendre un seul composant de message WSDL défini au moyen d'un élément de schéma XML. Par exemple :

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

Information associée

 Page de téléchargement d'API Java pour XML-RPC (JAX-RPC)

 Quel style de langage WSDL dois-je utiliser ?

Développement d'applications client

Le processus de développement d'applications client comprend un certain nombre d'étapes.

Procédure

1. Décidez quelle API de services Web votre application client doit utiliser : l'API de Business Flow Manager, l'API de Human Task Manager ou les deux.
2. Exportez les fichiers nécessaires depuis l'environnement de WebSphere Process Server. Vous pouvez également copier les fichiers depuis le CD client WebSphere Process Server.
3. Dans l'environnement de développement d'applications client que vous avez sélectionné, générez un *client proxy* à l'aide des artefacts exportés.
4. Facultatif : Générez des *classes auxiliaires*. Les classes auxiliaires sont requises si votre application client interagit directement avec des tâches ou des processus concrets présents sur le serveur WebSphere. Elles ne sont toutefois pas obligatoires si votre application client est uniquement destinée à exécuter des tâches génériques telles que l'émission de requêtes.
5. Développez le code de votre application client.
6. Ajoutez les mécanismes de sécurité nécessaires à votre application client.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Copie d'artefacts

Un certain nombre doivent être copiés depuis l'environnement WebSphere afin de créer des applications client.

Deux méthodes permettent d'obtenir ces artefacts :

- Publiez et exportez-les depuis l'environnement WebSphere Process Server.
- Copiez les fichiers depuis le CD client WebSphere Process Server.

Concepts associés

Utilisation de fichiers sur le CD du client

Une solution alternative visant à exporter des artefacts depuis l'environnement du serveur WebSphere consiste à copier les fichiers requis pour la génération d'une application client à partir du CD du client WebSphere Process Server.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Publication et exportation d'artefacts depuis l'environnement de serveurs

Avant d'être en mesure de développer des applications client pour accéder aux API de services Web, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere.

Publication et exportation d'artefacts depuis l'environnement de serveurs

Avant d'être en mesure de développer des applications client pour accéder aux API de services Web, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere.

A propos de cette tâche

Les artefacts à exporter sont les suivants :

- Fichiers WSDL (Web Service Definition Language) décrivant les types de port et les opérations qui génèrent les API de services Web.

- Fichiers XSD (XML Schema Definition) contenant des définitions de types de données référencés par des services et des méthodes dans les fichiers WSDL.
- Fichiers XSD et WSDL supplémentaires décrivant des objets métier. Les objets métier décrivent des tâches utilisateur ou des processus métier concrets s'exécutant sur le serveur WebSphere. Ces fichiers supplémentaires sont requis uniquement si votre application client doit interagir directement avec les tâches utilisateur ou les processus métier concrets via les API de services Web. Ils ne sont pas nécessaires si votre application client est uniquement destinée à exécuter des tâches génériques, tels que l'émission de requêtes.

Une fois ces artefacts publiés, vous devez les copier dans votre environnement de programmation client, dans lequel ils sont utilisés pour générer un client proxy et des classes auxiliaires.

Concepts associés

Copie d'artefacts

Un certain nombre doivent être copiés depuis l'environnement WebSphere afin de créer des applications client.

Tâches associées

Spécification de l'adresse du noeud final de service Web

L'adresse du noeud final de service Web est l'adresse URL qu'une application client doit spécifier pour accéder aux API de services Web. L'adresse du noeud final est inscrite dans le fichier WSDL que vous exportez pour générer un client proxy pour votre application client.

Publication des fichiers WSDL

Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Vous devez d'abord publier ces fichiers WSDL, puis les copier de l'environnement WebSphere vers votre environnement de développement, où ils serviront à générer un client proxy.

Exportation des objets métier

Les processus métier et les tâches utilisateur disposent d'interfaces bien définies les rendant accessibles depuis l'extérieur en tant que services Web. Si ces interfaces font référence à des objets métier, vous devez exporter les définitions d'interface et les objets métier vers votre environnement de programmation client.

Spécification de l'adresse du noeud final de service Web :

L'adresse du noeud final de service Web est l'adresse URL qu'une application client doit spécifier pour accéder aux API de services Web. L'adresse du noeud final est inscrite dans le fichier WSDL que vous exportez pour générer un client proxy pour votre application client.

A propos de cette tâche

L'adresse du noeud final de service Web à utiliser dépend de la configuration de votre serveur WebSphere :

- Scénario 1. Un seul serveur WebSphere. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur, par exemple **host1:9080**.
- Scénario 2 : Un cluster WebSphere est composé de plusieurs serveurs. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur hébergeant les API de services Web, par exemple **host2:9081**.

- Scénario 3 : Un serveur Web est configuré en tant que système frontal. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur Web, par exemple : **host:80**.

Par défaut, l'adresse du noeud final de service Web adopte le format *protocole://hôte:port/racine_contexte/chemin d'accès fixe*. Où :

- *protocole*. Protocole de communication utilisé entre l'application client et le serveur WebSphere. Le protocole par défaut est HTTP. Vous pouvez également utiliser le protocole HTTPS (HTTP sur SSL), plus sécurisé. Il est recommandé d'utiliser HTTPS.
- *hôte:port*. Nom d'hôte et numéro de port d'accès au système hébergeant les API de service Web. Ces valeurs varient selon la configuration du serveur WebSphere ; si, par exemple, votre application client accède à l'application directement ou par l'intermédiaire d'un serveur Web frontal.
- *racine_contexte*. Vous pouvez affecter n'importe quelle valeur à la racine de contexte. La valeur choisie doit néanmoins être unique dans chaque cellule WebSphere. La valeur par défaut utilise un suffixe "node_server/cluster" pour éliminer les risques de conflit entre les noms.
- *chemin_accès_fixe* correspond à /sca/com/ibm/bpe/api/BFMWS (pour l'API de Business Flow Manager) ou à /sca/com/ibm/task/api/HTMWS (pour l'API de Human Task Manager) et ne peut pas être modifié.

L'adresse du noeud final de service Web est initialement spécifiée lors de la configuration du conteneur de processus métier ou du conteneur de tâche utilisateur :

Procédure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Sélectionnez **Applications** → **Modules SCA**.

Remarque : Vous pouvez également sélectionner **Applications** → **Applications d'entreprise** pour afficher la liste de toutes les applications d'entreprise disponibles.

3. Sélectionnez **BPEContainer** (pour le conteneur de processus métier) ou **TaskContainer** (pour le conteneur de tâches utilisateur) dans la liste des modules ou applications SCA.
4. Sélectionnez l'option **Fournir les informations URL du noeud final HTTP** (Fournir les informations URL du noeud final HTTP) dans la liste **Propriétés supplémentaires**.
5. Sélectionnez l'un des préfixes par défaut dans la liste ou entrez un préfixe personnalisé. Utilisez un préfixe issu de la liste de préfixes par défaut si vos applications client doivent se connecter directement au serveur d'applications hébergeant l'API de services Web. Sinon, indiquez un préfixe personnalisé.
6. Cliquez sur **Appliquer** pour copier le préfixe sélectionné dans le module SCA.
7. Cliquez sur **OK**. Les données URL sont sauvegardées dans votre espace de travail.

Résultats

Vous pouvez afficher la valeur en cours dans la console d'administration (par exemple pour le conteneur de processus métier : **Applications d'entreprise** → **BPEContainer** → **Afficher le descripteur de déploiement**).

Dans le fichier WSDL exporté, l'attribut `location` de l'élément `soap:address` contient l'adresse spécifiée pour le noeud final de services Web. Par exemple :

```
<wsdl:service name="BFMWSservice">
  <wsdl:port name="BFMWSport" binding="this:BFMWSbinding">
    <soap:address location="https://myserver:9080/WebServicesAPIs/sca/com/ibm/bpe/api/BFMWS"/>
  </wsdl:port>
</wsdl:service>
```

Tâches associées

Publication et exportation d'artefacts depuis l'environnement de serveurs
Avant d'être en mesure de développer des applications client pour accéder aux API de services Web, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere.

Publication des fichiers WSDL :

Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Vous devez d'abord publier ces fichiers WSDL, puis les copier de l'environnement WebSphere vers votre environnement de développement, où ils serviront à générer un client proxy.

Avant de commencer

Avant de publier les fichiers, assurez-vous que l'adresse du noeud final de services Web correcte est spécifiée. Il s'agit de l'adresse URL qu'une application client utilise pour accéder aux API de services Web.

A propos de cette tâche

La publication des fichiers WSDL n'est nécessaire qu'une fois.

Remarque : Si vous disposez du CD client WebSphere Process Server, vous pouvez copier les fichiers directement depuis cet emplacement vers votre environnement de programmation client.

Tâches associées

Publication et exportation d'artefacts depuis l'environnement de serveurs
Avant d'être en mesure de développer des applications client pour accéder aux API de services Web, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere.

Publication du WSDL des processus métier

La console d'administration permet de publier le fichier WSDL.

Publication du WSDL des tâches utilisateur

La console d'administration permet de publier le fichier WSDL.

Publication du WSDL des processus métier :

La console d'administration permet de publier le fichier WSDL.

Procédure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Sélectionnez **Applications** → **Modules SCA**

Remarque : Vous pouvez également sélectionner **Applications** → **Applications d'entreprise** pour afficher la liste de toutes les applications d'entreprise disponibles.

3. Choisissez l'application **BPEContainer** dans la liste des applications ou modules SCA.
4. Sélectionnez l'option **Publier des fichiers WSDL** dans la liste des **Propriétés supplémentaires**
5. Cliquez sur le fichier zip dans la liste.
6. Dans la fenêtre de téléchargement de fichiers qui s'affiche, cliquez sur **Enregistrer**.
7. Accédez à un dossier local et cliquez sur **Enregistrer**.

Résultats

Le fichier zip exporté est nommé BPEContainer_WSDLFiles.zip. Il contient un fichier WSDL qui décrit les services Web, ainsi que tous les fichiers XSD référencés dans le fichier WSDL.

Tâches associées

Publication des fichiers WSDL

Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Vous devez d'abord publier ces fichiers WSDL, puis les copier de l'environnement WebSphere vers votre environnement de développement, où ils serviront à générer un client proxy.

Publication du WSDL des tâches utilisateur :

La console d'administration permet de publier le fichier WSDL.

Procédure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Sélectionnez **Applications** → **Modules SCA**

Remarque : Vous pouvez également sélectionner **Applications** → **Applications d'entreprise** pour afficher la liste de toutes les applications d'entreprise disponibles.

3. Choisissez l'application **TaskContainer** dans la liste des applications ou modules SCA.
4. Sélectionnez l'option **Publier des fichiers WSDL** dans la liste des **Propriétés supplémentaires**
5. Cliquez sur le fichier zip dans la liste.
6. Dans la fenêtre de téléchargement de fichiers qui s'affiche, cliquez sur **Enregistrer**.
7. Accédez à un dossier local et cliquez sur **Enregistrer**.

Résultats

Le fichier zip exporté est nommé TaskContainer_WSDLFiles.zip. Il contient un fichier WSDL qui décrit les services Web, ainsi que tous les fichiers XSD référencés dans le fichier WSDL.

Tâches associées

Publication des fichiers WSDL

Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Vous devez d'abord publier ces fichiers WSDL, puis les copier de l'environnement WebSphere vers votre environnement de développement, où ils serviront à générer un client proxy.

Exportation des objets métier :

Les processus métier et les tâches utilisateur disposent d'interfaces bien définies les rendant accessibles depuis l'extérieur en tant que services Web. Si ces interfaces font référence à des objets métier, vous devez exporter les définitions d'interface et les objets métier vers votre environnement de programmation client.

A propos de cette tâche

Cette procédure doit être répétée pour chaque objet métier avec lequel votre application client entre en interaction.

Dans WebSphere Process Server, les objets métier définissent le format des messages de requête, de réponse et d'erreur qui interagissent avec les processus métier ou les tâches utilisateur. Ces messages peuvent également contenir les définitions des types de données complexes.

Par exemple, pour créer et démarrer une tâche utilisateur, les éléments d'information suivants doivent être transmis à l'instance de tâche :

- Le nom du modèle de tâche
- L'espace de nom du modèle de tâche.
- Un message d'entrée contenant les données métier mises en forme
- Un encapsuleur de réponse pour le renvoi du message de réponse
- Un message d'erreur pour le renvoi des erreurs et des exceptions

Ces éléments sont encapsulés dans un objet métier unique. Toutes les opérations de l'interface du service Web sont modélisées sous forme d'opération "document/littéral encapsulé". Les paramètres d'entrée et de sortie relatifs à ces opérations sont encapsulés dans des documents d'encapsulation. Les autres objets métier définissent la réponse correspondante et les formats des messages d'erreur.

Pour permettre la création et le démarrage du processus métier ou de la tâche utilisateur via un service Web, l'application client côté client doit pouvoir accéder à ces objets d'encapsulation.

Cette configuration est réalisée en exportant les objets métier depuis l'environnement WebSphere sous forme de fichiers WSDL (Web Service Definition Language) et XSD (XML Schema Definition), en important les définitions des types de données dans l'environnement de programmation client, puis en les convertissant en classes auxiliaires en vue de leur utilisation par l'application client.

Procédure

1. Lancez l'espace de travail WebSphere Integration Developer s'il n'est pas déjà en cours d'exécution.

2. Sélectionnez le module de bibliothèque contenant les objets métier à exporter.
Un module de bibliothèque est un fichier compressé contenant les objets métier requis.
3. Exportez le module de bibliothèque.
4. Copiez les fichiers exportés vers votre environnement de développement d'applications client.

Exemple

En supposant qu'un processus métier expose l'opération de service Web suivante :

```
<wsdl:operation name="updateCustomer">
  <wsdl:input message="tns:updateCustomerRequestMsg" name="updateCustomerRequest"/>
  <wsdl:output message="tns:updateCustomerResponseMsg" name="updateCustomerResponse"/>
  <wsdl:fault message="tns:updateCustomerFaultMsg" name="updateCustomerFault"/>
</wsdl:operation>
```

avec les messages WSDL définis comme suit :

```
<wsdl:message name="updateCustomerRequestMsg">
  <wsdl:part element="types:updateCustomer" name="updateCustomerParameters"/>
</wsdl:message>
<wsdl:message name="updateCustomerResponseMsg">
  <wsdl:part element="types:updateCustomerResponse" name="updateCustomerResult"/>
</wsdl:message>
<wsdl:message name="updateCustomerFaultMsg">
  <wsdl:part element="types:updateCustomerFault" name="updateCustomerFault"/>
</wsdl:message>
```

Les éléments *concrets* définis par l'utilisateur `types:updateCustomer`, `types:updateCustomerResponse` et `types:updateCustomerFault` doivent être transmis vers et depuis les API de services Web au moyen des paramètres `<xsd:any>` dans toutes les opérations *génériques* (`call`, `sendMessage` etc.) exécutées par l'application client. Ces éléments définis par le client sont créés, sérialisés et désérialisés côté application client à l'aide des classes auxiliaires générées par les fichiers XSD exportés.

Tâches associées

Publication et exportation d'artefacts depuis l'environnement de serveurs
Avant d'être en mesure de développer des applications client pour accéder aux API de services Web, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere.

Utilisation de fichiers sur le CD du client

Une solution alternative visant à exporter des artefacts depuis l'environnement du serveur WebSphere consiste à copier les fichiers requis pour la génération d'une application client à partir du CD du client WebSphere Process Server.

Dans ce cas, vous devez modifier manuellement l'adresse de noeud final des services Web par défaut des API Business Flow Manager API ou Human Task Manager.

Si l'application client doit pouvoir accéder aux deux API, vous devez éditer l'adresse de noeud final par défaut pour les deux API.

Concepts associés

Copie d'artefacts

Un certain nombre doivent être copiés depuis l'environnement WebSphere afin de créer des applications client.

Tâches associées

Copie de fichiers depuis le CD client

Les fichiers requis pour accéder aux API de services Web sont disponibles sur le CD client WebSphere Process Server.

Changement manuel d'adresse du noeud final de service Web

Si vous copiez les fichiers depuis le CD-ROM du client, vous devez remplacer l'adresse du noeud final du service Web spécifiée dans les fichiers WSDL par celle du serveur hébergeant les API des services Web.

Copie de fichiers depuis le CD client :

Les fichiers requis pour accéder aux API de services Web sont disponibles sur le CD client WebSphere Process Server.

Procédure

1. Accédez au CD client et au répertoire ProcessChoreographer\client.
2. Copiez les fichiers nécessaires à votre environnement de développement d'applications client.

Pour l'API de Business Flow Manager, copiez :

BFMWS.wsdl

Décrit les services Web disponibles dans l'API de services Web Business Flow Manager. Ce fichier contient l'adresse du noeud final.

BFMIF.wsdl

Décrit les paramètres et le type de données pour chaque service Web dans l'API de services Web Business Flow Manager.

BFMIF.xsd

Décrit les types de données utilisés dans l'API de services Web Business Flow Manager.

BPCGEN.xsd

Contient des types de données communs entre les API de services Web Business Flow Manager et Human Task Manager.

Pour l'API de Human Task Manager, copiez :

HTMWS.wsdl

Décrit les services Web disponibles dans l'API de services Web Human Task Manager. Ce fichier contient l'adresse du noeud final.

HTMIF.wsdl

Décrit les paramètres et le type de données pour chaque service Web dans l'API de services Web Human Task Manager.

HTMIF.xsd

Décrit les types de données utilisés dans l'API de services Web Human Task Manager.

BPCGEN.xsd

Contient des types de données communs entre les API de services Web Business Flow Manager et Human Task Manager.

Remarque : Le fichier BPCGen.xsd est commun aux deux API.

Que faire ensuite

Après avoir copié les fichiers, vous devez modifier manuellement l'adresse du noeud final de l'API de services Web dans les fichiers BFMWS.wsdl ou HTMWS.wsdl par celle du serveur d'applications WebSphere hébergeant les API de services Web.

Concepts associés

Utilisation de fichiers sur le CD du client

Une solution alternative visant à exporter des artefacts depuis l'environnement du serveur WebSphere consiste à copier les fichiers requis pour la génération d'une application client à partir du CD du client WebSphere Process Server.

Changement manuel d'adresse du noeud final de service Web :

Si vous copiez les fichiers depuis le CD-ROM du client, vous devez remplacer l'adresse du noeud final du service Web spécifiée dans les fichiers WSDL par celle du serveur hébergeant les API des services Web.

A propos de cette tâche

Vous pouvez utiliser la console d'administration pour définir l'adresse du noeud final de service Web avant d'exporter les fichiers WSDL. Si, toutefois, vous copiez les fichiers WSDL depuis le CD-ROM du client WebSphere Process Server, vous devez modifier manuellement l'adresse par défaut du noeud final de service Web.

L'adresse du noeud final de service Web à utiliser dépend de la configuration de votre serveur WebSphere :

- Scénario 1 : Une instance unique du serveur WebSphere est configurée. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur, par exemple **host1:9080**.
- Scénario 2 : Un cluster WebSphere est composé de plusieurs serveurs. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur hébergeant les API de services Web, par exemple **host2:9081**.
- Scénario 3 : Un serveur Web est configuré en tant que système frontal. L'adresse du noeud final WebSphere à spécifier est le nom d'hôte et le numéro de port du serveur Web, par exemple : **host:80**.

Concepts associés

Utilisation de fichiers sur le CD du client

Une solution alternative visant à exporter des artefacts depuis l'environnement du serveur WebSphere consiste à copier les fichiers requis pour la génération d'une application client à partir du CD du client WebSphere Process Server.

Tâches associées

Modification du noeud final de l'API de Business Flow Manager

Si vous copiez les fichiers de l'API de Business Flow Manager depuis le CD-ROM WebSphere Process Server, vous devez modifier manuellement l'adresse par défaut du noeud final.

Modification du noeud final de l'API de Human Task Manager

Si vous copiez les fichiers de l'API de Human Task Manager depuis le CD-ROM WebSphere Process Server, vous devez modifier manuellement l'adresse par défaut du noeud final.

Modification du noeud final de l'API de Business Flow Manager :

Si vous copiez les fichiers de l'API de Business Flow Manager depuis le CD-ROM WebSphere Process Server, vous devez modifier manuellement l'adresse par défaut du noeud final.

Procédure

1. Accédez au répertoire contenant les fichiers copiés depuis le CD-ROM du client.
2. Ouvrez le fichier BFMWS.wsdl dans un éditeur de texte ou un éditeur XML.
3. Localisez l'élément `soap:address` (vers la fin du fichier).
4. Remplacez la valeur de l'attribut `location` par l'URL HTTP du serveur sur lequel l'API du service Web fonctionne. Pour cela :
 - a. Vous pouvez remplacer `http` par `https` afin d'utiliser le protocole HTTPS, plus sécurisé.
 - b. Remplacez `localhost` par l'adresse IP ou le nom d'hôte associé à l'adresse de noeud final du serveur de l'API des services Web.
 - c. Remplacez `9080` par le numéro de port du serveur d'applications.
 - d. Remplacez `BPEContainer_N1_server1` par la racine de contexte de l'application exécutant l'API des services Web. La racine de contexte par défaut est composée comme suit :
 - `BPEContainer`. Nom de l'application.
 - `N1`. Nom du noeud.
 - `server1`. Nom du serveur.
 - e. Ne modifiez pas la partie fixe de l'URL (`/sca/com/ibm/bpe/api/BFMWS`) .

Par exemple, si l'application s'exécute sur le serveur `s1.n1.ibm.com` et que le serveur accepte les requêtes SOAP/HTTP au port `9080`, modifiez l'élément `soap:address` comme suit :

```
<soap:address location="http://s1.n1.ibm.com:9080/  
BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

Tâches associées

Changement manuel d'adresse du noeud final de service Web

Si vous copiez les fichiers depuis le CD-ROM du client, vous devez remplacer l'adresse du noeud final du service Web spécifiée dans les fichiers WSDL par celle du serveur hébergeant les API des services Web.

Modification du noeud final de l'API de Human Task Manager :

Si vous copiez les fichiers de l'API de Human Task Manager depuis le CD-ROM WebSphere Process Server, vous devez modifier manuellement l'adresse par défaut du noeud final.

Procédure

1. Accédez au répertoire contenant les fichiers copiés depuis le CD-ROM du client.
2. Ouvrez le fichier HTMWS.wsdl dans un éditeur de texte ou un éditeur XML.
3. Localisez l'élément `soap:address` (vers la fin du fichier).
4. Remplacez la valeur de l'attribut `location` par l'adresse de noeud final correcte. Pour cela :
 - a. Vous pouvez remplacer `http` par `https` afin d'utiliser le protocole HTTPS, plus sécurisé.
 - b. Remplacez `localhost` par l'adresse IP ou le nom d'hôte associé à l'adresse de noeud final du serveur de l'API des services Web.

- c. Remplacez *9080* par le numéro de port du serveur d'applications.
- d. Remplacez *HTMContainer_N1_server1* par la racine de contexte de l'application exécutant l'API des services Web. La racine de contexte par défaut est composée comme suit :
 - *HTMContainer*. Nom de l'application.
 - *N1*. Nom du noeud.
 - *server1*. Nom du serveur.
- e. Ne modifiez pas la partie fixe de l'URL (*/sca/com/ibm/task/api/HTMWS*).

Par exemple, si l'application s'exécute sur le serveur **s1.n1.ibm.com** et que le serveur accepte les requêtes SOAP/HTTPS au port **9081**, modifiez l'élément `soap:address` comme suit :

```
<soap:address location="https://s1.n1.ibm.com:9081/  
HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

Tâches associées

Changement manuel d'adresse du noeud final de service Web

Si vous copiez les fichiers depuis le CD-ROM du client, vous devez remplacer l'adresse du noeud final du service Web spécifiée dans les fichiers WSDL par celle du serveur hébergeant les API des services Web.

Développement d'applications client dans l'environnement de services Web Java

Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Génération d'un client proxy (services Web Java)

Les applications client de service Web utilisent un *client proxy* pour gérer l'interaction avec les API de services Web.

Création de classes auxiliaires pour les processus BPEL (services Web Java)

Les objets métier référencés dans les requêtes d'API concrètes (par exemple, `sendMessage`, ou `call`) nécessitent que les applications client utilisent les éléments de style "document/literal wrapped". Les applications client requièrent des classes auxiliaires pour leur permettre de générer les éléments d'encapsulation nécessaires.

Création d'une application client (services Web Java)

Une application client envoie des requêtes et reçoit des réponses vers et depuis les API de services Web. En utilisant un client proxy pour gérer les communications et des classes auxiliaires pour formater les types de données, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Renforcement de la sécurité (services Web Java)

Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Ajout d'un support de transaction (services Web Java)

Les applications client de service Web Java peuvent être configurées pour permettre au traitement de la requête côté serveur de participer à la transaction client, en transmettant un contexte d'application client en tant que requête de service. Ce support de transaction atomique est défini dans la spécification Web Services-Atomic Transaction (WS-AT).

Génération d'un client proxy (services Web Java)

Les applications client de service Web utilisent un *client proxy* pour gérer l'interaction avec les API de services Web.

A propos de cette tâche

Un client proxy destiné aux services Web Java contient un certain nombre de classes de Bean Java qui sont appelées par l'application client pour exécuter des demandes de services Web. Le client proxy gère l'assemblage des paramètres de services sous forme de messages SOAP, envoie des messages SOAP au service Web via HTTP, reçoit des réponses du service Web et transmet toutes les données renvoyées à l'application client.

Par conséquent, un client proxy permet à une application d'appeler un service Web comme s'il s'agissait d'une fonction locale.

Remarque : La génération d'un client proxy n'est nécessaire qu'une fois. Toutes les applications client accédant aux mêmes API de services Web peuvent alors utiliser le même client proxy.

Dans l'environnement de services Web IBM, il existe deux façons de générer un client proxy :

- A l'aide des environnements de développement intégrés Rational Application Developer ou WebSphere Integration Developer.
- A l'aide de l'outil de ligne de commande WSDL2Java.

Les autres environnements de développement de services Web Java comprennent généralement l'outil WSDL2Java ou des fonctions de génération d'applications client de propriétés.

Concepts associés

Développement d'applications client dans l'environnement de services Web Java
Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Tâches associées

Utilisation de Rational Application Developer pour générer un client de proxy
L'environnement de développement intégré Rational Application Developer vous permet de générer un client proxy pour votre application client.

Utilisation de WSDL2Java pour générer un client proxy
WSDL2Java est un outil de ligne de commande qui génère un client proxy. Un client proxy simplifie la programmation des applications client.

Utilisation de Rational Application Developer pour générer un client de proxy :

L'environnement de développement intégré Rational Application Developer vous permet de générer un client proxy pour votre application client.

Avant de commencer

Avant de générer un client proxy, vous devez avoir préalablement exporté les fichiers WSDL décrivant les API de services Web pour les processus métier et les tâches utilisateur depuis l'environnement WebSphere (ou le CD client WebSphere Process Server), puis les avoir copiés dans votre environnement de programmation client.

Procédure

1. Ajoutez à votre projet le fichier WSDL approprié.
 - Pour les processus métier :
 - a. Décompressez le fichier d'exportation `BPEContainer_nomnoeud_nomserveur_WSDLFiles.zip` dans un répertoire temporaire.
 - b. Importez le sous-répertoire META-INF à partir du répertoire décompressé `BPEContainer_nomnoeud_nomserveur.ear/b.jar`.
 - Pour les tâches utilisateur:
 - a. Décompressez le fichier d'exportation `TaskContainer_nomnoeud_nomserveur_WSDLFiles.zip` dans un répertoire temporaire.
 - b. Importez le sous-répertoire META-INF à partir du répertoire décompressé `TaskContainer_nomnoeud_nomserveur.ear/h.jar`.

Un nouveau répertoire wsdl et une structure de sous-répertoire sont créés dans votre projet.
2. Modifiez les propriétés de l'assistant de Service Web :
 - a. Dans Rational Application Developer, sélectionnez **Préférences** → **Services Web** → **Génération de code** → **Programme d'exécution IBM WebSphere**.
 - b. Sélectionnez l'option **Générer Java à partir de WSDL en style non encapsulé** (Generate Java from WSDL using the no wrapped style).

Remarque : Si vous n'êtes pas en mesure de sélectionner l'option **Web services (services web)** dans le menu **Preferences (Préférences)**, vous devez d'abord activer les fonctions requises comme suit : **Window (Fenêtre) → Preferences (Préférences) → Workbench (Workbench) → (Capabilities (Fonctions))**. Cliquez sur **Web Service Developer (Développeur de services web)**, puis sur **OK**. Ensuite, ouvrez une nouvelle fois la fenêtre **Preferences (Préférences)**, puis modifiez l'option **Code Generation (Génération de code)**.

3. Sélectionnez le fichier BFMWS.WSDL ou le fichier HTMWS.WSDL situé dans le répertoire wsdl nouvellement créé.
4. Cliquez avec le bouton droit et sélectionnez **Web Services (Services web) → Generate client (Générer un client)**.
Avant d'entamer le reste de la procédure, assurez-vous que le serveur a démarré.
5. Dans la fenêtre **Web services (Services web)**, cliquez sur **Next (Suivant)** afin d'accepter toutes les valeurs par défaut.
6. Dans la fenêtre **Web Service Selection (Sélection des services web)**, cliquez également sur **Next (Suivant)** afin d'accepter toutes les valeurs par défaut.
7. Dans la fenêtre **Client Environment Configuration (Configuration de l'environnement client)** :
 - a. Cliquez sur **Edit (Editer)**, puis sélectionnez la valeur **IBM WebSphere** pour l'option **Web service runtime (Exécution des services web)**
 - b. Sélectionnez la valeur **1.4** pour l'option **J2EE Version (Version J2EE)**.
 - c. Cliquez sur **OK**.
 - d. Cliquez sur **Suivant**.
8. Cette étape est nécessaire uniquement si vous devez générer un client de services comportant à la fois des API de Business Process et des API de Human Task Web Services, puisqu'il existe des méthodes en double dans les deux fichiers WSDL.
 - a. Dans la fenêtre **Proxy des services web**, sélectionnez **Define custom mapping for namespace to package (Définir le mappage personnalisé pour l'espace de nom à compresser)**, puis cliquez sur **OK**.
 - b. Dans la fenêtre de mappage de l'espace nom **Web Service Client (Client de service web)** à compresser, ajoutez les espaces de nom et package suivants :
Pour BFMWS.wsdl :

Espace de nom	Package
http://www.ibm.com/xmlns/prod/websphere/business-process/types/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0/Binding	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.bpe

Pour HTMWS.wsdl :

Espace de nom	Package
http://www.ibm.com/xmlns/prod/websphere/human-task/types/6.0	com.ibm.sca.task

Espace de nom	Package
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0/Binding	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.task

Si vous êtes invité à confirmer l'écrasement, cliquez sur **YesToAll (OuiPourTous)**.

9. Cliquez sur **Finish (Terminer)**.

Résultats

Un client proxy contenant un certain nombre de classes Java proxy, locator et helper est généré et ajouté à votre projet. Le descripteur de déploiement est également mis à jour.

Tâches associées

Génération d'un client proxy (services Web Java)

Les applications client de service Web utilisent un *client proxy* pour gérer l'interaction avec les API de services Web.

Utilisation de WSDL2Java pour générer un client proxy :

WSDL2Java est un outil de ligne de commande qui génère un client proxy. Un client proxy simplifie la programmation des applications client.

Avant de commencer

Avant de générer un client proxy, vous devez avoir préalablement exporté les fichiers WSDL décrivant les API de services Web pour les processus métier ou les tâches utilisateur depuis l'environnement WebSphere (ou le CD client WebSphere Process Server), puis les avoir copiés dans votre environnement de programmation client.

A propos de cette tâche

Procédure

1. Utilisation de l'outil WSDL2Java pour générer un client proxy : Entrez :

wSDL2java *options* *WSDLfilepath*

Où :

- *options* comprend :

-noWrappedOperations (-w)

Désactive la détection des opérations encapsulées. Des beans Java sont générés pour les messages de requête et de réponse.

Remarque : Il ne s'agit pas de la valeur par défaut.

-role (-r)

Spécifiez la valeur **client** pour générer les fichiers et établir des liaisons de développement côté client.

-container (-c)

Conteneur côté client à utiliser. Les arguments admis sont les suivants :

client Conteneur client.

ejb Conteneur d'EJB (Enterprise JavaBeans).

none Aucun conteneur.

web Conteneur Web.

-output (-o)

Dossier dans lequel sont stockés les fichiers générés.

Pour obtenir la liste complète des paramètres WSDL2Java, utilisez le commutateur de ligne de commande **-help** ou reportez-vous à l'aide en ligne relative à l'outil WSDL2Java dans WID/RAD.

- *WSDLfilepath* désigne le chemin d'accès et le nom du fichier WSDL exporté depuis l'environnement WebSphere ou copié depuis le CD client.

L'exemple suivant permet de générer un client proxy pour l'API de services Web 'Human Task Activities' :

```
call wsd12java.bat -r client -c client -noWrappedOperations  
-output c:\ws\proxyClient c:\ws\bin\HTMWS.wsd1
```

2. Incluez à votre projet les fichiers classe générés.

Tâches associées

Génération d'un client proxy (services Web Java)

Les applications client de service Web utilisent un *client proxy* pour gérer l'interaction avec les API de services Web.

Création de classes auxiliaires pour les processus BPEL (services Web Java)

Les objets métier référencés dans les requêtes d'API concrètes (par exemple, `sendMessage`, ou `call`) nécessitent que les applications client utilisent les éléments de style "document/literal wrapped". Les applications client requièrent des classes auxiliaires pour leur permettre de générer les éléments d'encapsulation nécessaires.

Avant de commencer

Pour créer des classes auxiliaires, vous devez avoir préalablement exporté le fichier WSDL de l'API des services Web depuis l'environnement WebSphere Process Server.

A propos de cette tâche

Les opérations `call()` et `sendMessage()` des API de services Web permettent l'interaction avec les processus BPEL de WebSphere Process Server. Le message d'entrée de l'opération `call()` attend l'indication de l'encapsuleur document/littéral figurant dans le message d'entrée du processus.

Il existe différentes techniques permettant de générer des classes auxiliaires pour une tâche utilisateur ou un processus BPEL, notamment :

1. Utilisez l'objet `SoapElement`.

Dans l'environnement Rational Application Developer disponible dans WebSphere Integration Developer, le moteur de service Web prend en charge JAX-RPC 1.1. Dans JAX-RPC 1.1, l'objet `SoapElement` étend un élément DOM (Document Object Model), de sorte qu'il est possible d'utiliser l'API DOM pour créer, lire, charger et enregistrer des messages SOAP.

Supposons, par exemple, que le fichier WSDL contienne le message d'entrée suivant pour un processus de flux de travaux ou une tâche utilisateur :

```
<xsd:element name="operation1">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="input1" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Le fichier WSDL est créé lorsque vous développez un module de processus ou de tâche utilisateur.

Pour créer le message SOAP correspondant dans votre application client à l'aide de l'API du DOM :

```
SOAPFactory soapfactoryinstance = SOAPFactory.newInstance();
SOAPElement soapmessage = soapfactoryinstance.createElement
    ("operation1", namespaceprefix, interfaceURI);
SOAPElement inputelement = soapfactoryinstance.createElement("input1");
inputelement.addTextNode( message value);
soapmessage.addChildElement(outputelement);
```

L'exemple suivant illustre comment créer des paramètres d'entrée pour l'opération `sendMessage` dans votre application client :

```
SendMessage inWsend = new SendMessage();
inWsend.setProcessTemplateName(processtemplatename);
inWsend.setPortType(porttype);
inWsend.setOperation(operationname);
inWsend.set_any(soapmessage);
```


2. Utilisez la fonction de liaison de données personnalisée de WebSphere.


Cette technique est décrite dans les articles `developerWorks` suivants :

- How to choose a custom mapping technology for Web services (Choix d'une technologie de mappage personnalisée pour les services Web)
- Developing Web Services with EMF SDOs for complex XML schema (Développement de services Web à l'aide d'objets SDO pour un schéma XML complexe)

Concepts associés

Développement d'applications client dans l'environnement de services Web Java
Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

 [Interoperability With Patterns and Strategies for Document-Based Web Services \(Interopérabilité avec modèles et stratégies pour des services Web basés sur des documents\)](#)

 [Web Services support for Schema/WSDL\(s\) containing optional JAX-RPC 1.0/1.1 XML Schema Types \(Prise en charge des services Web pour des schémas/WSDL contenant des types de schéma XML JAX-RPC 1.0/1.1 facultatifs\)](#)

Création d'une application client (services Web Java)

Une application client envoie des requêtes et reçoit des réponses vers et depuis les API de services Web. En utilisant un client proxy pour gérer les communications et des classes auxiliaires pour formater les types de données, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Avant de commencer

Avant de commencer à créer une application client, générez le client proxy et les classes auxiliaires éventuellement requises.

A propos de cette tâche

Vous pouvez développer des applications client à l'aide de n'importe quel outil compatible avec les services Web, tel que IBM Rational Application Developer (RAD). Vous pouvez créer tous types d'applications de services Web pour appeler les API de services Web génériques.

Procédure

1. Créez un projet d'application client.
2. Générez le client proxy et ajoutez les classes auxiliaires Java dans votre projet.
3. Codez votre application client.
4. Générez le projet.
5. Exécutez l'application client.

Exemple

L'exemple suivant illustre comment utiliser l'API de services Web Business Flow Manager.

```
// create the proxy
    BFMIFProxy proxy = new BFMIFProxy();
// prepare the input data for the operation
    GetProcessTemplate iW = new GetProcessTemplate();
    iW.setIdentifier(your_process_template_name);

// invoke the operation
    GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

// process output of the operation
    ProcessTemplateType ptd = oW.getProcessTemplate();
    System.out.println("getName= " + ptd.getName());
    System.out.println("getPtid= " + ptd.getPtid());
```

Concepts associés

Développement d'applications client dans l'environnement de services Web Java
Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Renforcement de la sécurité (services Web Java)

Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

A propos de cette tâche

WebSphere Application Server prend en charge les mécanismes de sécurité suivants pour les API des services Web :

- Le jeton de nom d'utilisateur
- LTPA (Lightweight Third Party Authentication)

Concepts associés

Développement d'applications client dans l'environnement de services Web Java
Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Tâches associées

Implémentation du jeton du nom d'utilisateur

Le mécanisme de sécurité relatif au jeton du nom d'utilisateur fournit une autorisation d'accès via un nom d'utilisateur et un mot de passe.

Implémentation du mécanisme de sécurité LTPA

Le mécanisme de sécurité LTPA (Lightweight Third Party Authentication) peut être utilisé lorsque l'application client s'exécute au sein d'un contexte de sécurité précédemment établi.

Implémentation du jeton du nom d'utilisateur :

Le mécanisme de sécurité relatif au jeton du nom d'utilisateur fournit une autorisation d'accès via un nom d'utilisateur et un mot de passe.

A propos de cette tâche

Le mécanisme de sécurité relatif au jeton du nom d'utilisateur vous permet d'implémenter différents *gestionnaires d'appel*. Selon le choix que vous avez effectué :

- Vous êtes invité à indiquer un nom d'utilisateur et un mot de passe chaque fois que vous exécutez l'application client.
- Le nom d'utilisateur et le mot de passe sont inscrits dans le descripteur de déploiement.

Dans tous les cas, le nom d'utilisateur et le mot de passe doivent correspondre à ceux d'un rôle autorisé dans le conteneur de tâches utilisateur ou de processus métier correspondant.

Le nom d'utilisateur et le mot de passe sont encapsulés dans l'enveloppe du message de la requête, et apparaissent ainsi "en clair" dans l'en-tête du message SOAP. Il est, par conséquent, vivement recommandé de configurer l'application client afin qu'elle utilise le protocole de communication HTTPS (HTTP via SSL). Toutes les communications sont alors cryptées. Vous pouvez sélectionner le protocole de communication HTTPS lorsque vous spécifiez l'adresse URL du noeud final de l'API du service Web.

Pour définir un jeton de nom d'utilisateur :

Procédure

1. Créez un jeton de sécurité :
 - a. Ouvrez l'**Editeur de déploiement** de votre module
 - b. Cliquez sur l'onglet **Extension de service web**.
 - c. Sous **Références aux services**, les références aux services web suivantes peuvent apparaître :
 - service/BFMWSService pour les processus métier
 - service/HTMWSService pour les tâches utilisateur

Ce qui apparaît dépend de si BFMWS.wsdl (pour le processus métier), HTMWWS.wsdl (pour la tâche utilisateur) ou les deux, ont été ajoutés au moment de générer le client de proxy.

- d. Pour les deux références aux services :
 - 1) Sélectionnez l'une des **Références aux services**.
 - 2) Développez la section **Configuration du générateur de demande**.
 - 3) Développez la sous-section **Jeton de sécurité**.
 - 4) Cliquez sur **Ajouter**. La fenêtre Jeton de sécurité apparaît.
 - 5) Dans la zone **Nom**, entrez le nom du nouveau jeton de sécurité : **UserNameTokenBFM** ou **UserNameTokenHTM** .
 - 6) Dans la zone de liste déroulante **Type de jeton**, sélectionnez **Nom d'utilisateur**. (La zone **Nom local** est automatiquement renseignée avec une valeur par défaut.)
 - 7) Laissez le champ **URI** vide. Les jetons de nom d'utilisateur ne nécessitent pas de valeur URI.
 - 8) Cliquez sur **OK**.
2. Créez un générateur de jeton :
 - a. Ouvrez l'**Editeur de déploiement** de votre module
 - b. Cliquez sur l'onglet **Liaison de service web**
 - c. Sous les **Références aux services**, les mêmes références aux services web sont mentionnées à l'étape précédente :
 - service/BFMWSService pour les processus métier
 - service/HTMWSService pour les tâches utilisateur
 - d. Pour les deux références aux services :
 - 1) Sélectionnez l'une des **Références aux services**.
 - 2) Développez la section **Configuration de la sécurité de liaison du générateur de demande**.
 - 3) Développez la sous-section **Générateur de jeton**.
 - 4) Cliquez sur **Ajouter**. La fenêtre Générateur de jeton apparaît.
 - 5) Dans la zone **Nom**, tapez le nom du nouveau générateur de jeton, par exemple "UserNameTokenGeneratorBFM" ou "UserNameTokenGeneratorHTM".
 - 6) Dans la zone **Classe du générateur de jeton**, assurez-vous que la classe de générateur de jeton suivante est sélectionnée : **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**.
 - 7) Dans la zone de liste déroulante **Jeton de sécurité**, sélectionnez le jeton de sécurité approprié que vous avez créé antérieurement.
 - 8) Cochez la case **Use Value Type (Utiliser le type de valeur)**.
 - 9) Dans le champ **Value Type (Type de valeur)**, sélectionnez **Username Token (Jeton nom d'utilisateur)**. (La zone **Local name (Nom local)** est automatiquement renseignée avec le **Username Token (Jeton utilisateur)** que vous avez choisi.)
 - 10) Dans la zone **Call back handler (Gestionnaire des rappels)**, saisissez "com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler" (qui vous invite à fournir le nom d'utilisateur et le mot de passe lorsque vous lancez l'application client) ou "com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler".

- 11) Si vous choisissez **NonPromptCallbackHandler**, vous devez indiquer un nom d'utilisateur et un mot de passe valides dans le champ correspondant du descripteur de déploiement.
- 12) Cliquez sur **OK**.

Tâches associées

Renforcement de la sécurité (services Web Java)

Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Information associée

☞ IBM WebSphere Developer - Journal technique : Sécurité des services Web avec WebSphere Application Server V6

Implémentation du mécanisme de sécurité LTPA :

Le mécanisme de sécurité LTPA (Lightweight Third Party Authentication) peut être utilisé lorsque l'application client s'exécute au sein d'un contexte de sécurité précédemment établi.

A propos de cette tâche

Le mécanisme de sécurité LTPA est disponible uniquement si votre application client s'exécute au sein d'un environnement sécurisé dans lequel un contexte de sécurité a déjà été établi. Par exemple, si votre application client s'exécute dans un conteneur EJB (Enterprise JavaBeans), le client EJB doit se connecter avant de pouvoir appeler l'application client. Un contexte de sécurité est alors établi. Si l'application client EJB appelle le service Web, le gestionnaire d'appel LTPA extrait le jeton LTPA du contexte de sécurité, puis l'ajoute au message de la requête SOAP. Côté serveur, le jeton LTPA est géré par le mécanisme LTPA.

Pour implémenter le mécanisme de sécurité LTPA :

Procédure

1. Dans l'environnement Rational Application Developer disponible dans WebSphere Integration Developer, choisissez **Liaison de service Web** → **Configuration de la sécurité de liaison du générateur de requête** → **Générateur de jeton**.
2. Créez un jeton de sécurité :
 - a. Ouvrez l'**Editeur de déploiement** de votre module
 - b. Cliquez sur l'onglet **WS Extension (Extension de service web)**.
 - c. Sous **Service References (Références aux services)**, les **références aux services web** suivantes peuvent apparaître :
 - service/BFMWSService pour les processus métier
 - service/HTMWSService pour les tâches utilisateurCe qui apparaît dépend de si BFMWS.wsdl (pour le processus métier), HTMWS.wsdl (pour la tâche utilisateur) ou les deux, ont été ajoutés au moment de générer le client de proxy.
 - d. Pour les deux références aux services :
 - 1) Sélectionnez l'une des **Références aux services**.
 - 2) Développez la section **Request Generator Configuration (Demander la configuration du générateur)**.
 - 3) Développez la sous-section **Username Token (Jeton de sécurité)**.

- 4) Cliquez sur **Add (Ajouter)**. La fenêtre Security Token (Jeton de sécurité) apparaît.
 - 5) Dans la zone **Name (Nom)**, entrez le nom du nouveau jeton de sécurité : **LTPATokenBFM** ou **LTPATokenHTM** .
 - 6) Dans la zone de liste déroulante **Token Type (Type de jeton)**, sélectionnez **LTPAToken (Jeton LTPA)**. (Les zones **URI** et **Local Name (Nom local)** sont automatiquement renseignées avec les valeurs par défaut.)
 - 7) Cliquez sur **OK**.
3. Créez un générateur de jeton :
- a. Ouvrez le **Deployment Editor (Editeur de déploiement)** de votre module
 - b. Cliquez sur l'onglet **WS Binding (Liaison de service web)**
 - c. Sous les **Services References (Références aux services)**, les mêmes références aux services web sont mentionnées à l'étape précédente :
 - service/BFMWSService pour les processus métier
 - service/HTMWSService pour les tâches utilisateur
 - d. Pour les deux références aux services :
 - 1) Sélectionnez l'une des **Références aux services**.
 - 2) Développez la section **Security Request Generator Binding Configuration (Configuration de la sécurité de liaison du générateur de requête)**.
 - 3) Développez la sous-section **Token Generator (Générateur de jeton)**.
 - 4) Cliquez sur **Add (Ajouter)**. La fenêtre Générateur de jeton apparaît.
 - 5) Dans la zone **Nom**, tapez le nom du nouveau générateur de jeton, par exemple "LTPATokenGeneratorBFM" ou "LTPATokenGeneratorHTM".
 - 6) Dans la zone **Token Generatr Class (Classe du générateur de jeton)**, assurez-vous que la classe de générateur de jeton suivante est sélectionnée : **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**.
 - 7) Dans la zone de liste déroulante **Security Token (Jeton de sécurité)**, sélectionnez le jeton de sécurité approprié que vous avez créé antérieurement.
 - 8) Cochez la case **Use Value Type (Utiliser le type de valeur)**.
 - 9) Dans le champ **Value Type (Type de valeur)**, sélectionnez **LTPAToken (Jeton LTPA)**. (Les zones **URI** et **Local Name (Nom local)** sont automatiquement renseignées avec le **LTPA Token (Jeton LTPA)** que vous avez choisi.)
 - 10) Dans la zone **Call back handler (Gestionnaire des rappels)**, saisissez "com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler".
 - 11) Cliquez sur **OK**.

Résultats

Lors de l'exécution, l'élément **LTPATokenCallbackHandler** extrait le jeton LTPA du contexte de sécurité existant et l'ajoute au message de la requête SOAP.

Tâches associées

Renforcement de la sécurité (services Web Java)

Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Ajout d'un support de transaction (services Web Java)

Les applications client de service Web Java peuvent être configurées pour permettre au traitement de la requête côté serveur de participer à la transaction client, en transmettant un contexte d'application client en tant que requête de service. Ce support de transaction atomique est défini dans la spécification Web Services-Atomic Transaction (WS-AT).

A propos de cette tâche

WebSphere Application Server exécute chaque requête d'API de services Web en tant que transaction atomique distincte. Les applications client peuvent être configurées en vue d'utiliser un support de transaction pour :

- Participer à la transaction. Le traitement des requêtes côté serveur est effectué dans le contexte de transaction de l'application client. Si, par la suite, le serveur rencontre un problème alors que la requête d'API de services Web est en cours d'exécution et est invalidée, la requête de l'application client est également invalidée.
- Ne pas utiliser de prise en charge de la transaction. WebSphere Application Server crée néanmoins une transaction afin d'exécuter la requête mais le traitement de la requête côté serveur n'est pas effectué au moyen du contexte de transaction de l'application client.

Concepts associés

Développement d'applications client dans l'environnement de services Web Java
Vous pouvez utiliser n'importe quel environnement de développement Java compatible avec les services Web Java pour développer des applications client destinées aux API de service Web.

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Génération d'un client proxy (.NET)

Les applications client .NET utilisent un *client proxy* pour gérer l'interaction avec les API de service Web. Un client proxy permet d'isoler les applications client hors de la complexité du protocole de messagerie de service Web.

Création de classes auxiliaires pour les processus BPEL (.NET)

Certaines opérations d'API de services Web nécessitent que les applications client utilisent des éléments encapsulés "document/littéral". Les applications client requièrent des classes auxiliaires pour leur permettre de générer les éléments d'encapsulation nécessaires.

Création d'une application client (.NET)

Une application client envoie des requêtes et reçoit des réponses vers et depuis les API de services Web. En utilisant un client proxy pour gérer les communications et des classes auxiliaires pour formater les types de données, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Renforcement de la sécurité (.NET)

Vous pouvez sécuriser les communications des services Web en intégrant des mécanismes de sécurité à vos applications client.

Génération d'un client proxy (.NET)

Les applications client .NET utilisent un *client proxy* pour gérer l'interaction avec les API de service Web. Un client proxy permet d'isoler les applications client hors de la complexité du protocole de messagerie de service Web.

Avant de commencer

Pour créer un client proxy, vous devez avoir préalablement exporté les fichiers WSDL depuis l'environnement WebSphere et les avoir copiés dans votre environnement de programmation client.

Remarque : Si vous disposez du CD client WebSphere Process Server, vous pouvez également copier les fichiers depuis cet emplacement.

A propos de cette tâche

Un client proxy comprend un ensemble de classes de bean C#. Chaque classe contient l'ensemble des méthodes et objets exposés par le biais d'un service Web unique. Les méthodes du service gèrent l'assemblage des paramètres sous forme de messages SOAP complets, envoient les messages SOAP au service Web via le protocole HTTP, reçoivent les réponses émises par le service Web et traitent les données éventuellement renvoyées.

Remarque : La génération d'un client proxy n'est nécessaire qu'une fois. Toutes les applications client accédant aux API de service Web peuvent utiliser le même client proxy.

Procédure

1. Utilisez la commande WSDL pour générer un client proxy : Entrez :

`wSDL options WSDLfilepath`

Où :

- *options* comprend :

/language

Permet de spécifier le langage utilisé pour créer la classe proxy. L'option par défaut est C#. Vous pouvez également spécifier **VB** (Visual Basic), **JS** (JScript) ou **VJS** (Visual J#) comme argument de langage.

/output

Nom du fichier de sortie qualifié par le suffixe approprié. Par exemple : proxy.cs

/protocol

Protocole mis en oeuvre dans la classe proxy. Le paramètre par défaut est **SOAP**.

Pour obtenir une liste complète des paramètres **WSDL.exe**, utilisez le commutateur de ligne de commande */?* ou reportez-vous à l'aide en ligne relative à l'outil WSDL dans Visual Studio.

- *WSDLfilepath* désigne le chemin d'accès et le nom du fichier WSDL exporté depuis l'environnement WebSphere ou copié depuis le CD client.

L'exemple suivant permet de générer un client proxy pour l'API de services Web Human Task Manager :

```
wsdl /language:cs /output:proxycient.cs c:\ws\bin\HTMWS.wsdl
```

2. Compilez le client proxy sous forme de bibliothèque de liaison dynamique (DLL).

Concepts associés

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Création de classes auxiliaires pour les processus BPEL (.NET)

Certaines opérations d'API de services Web nécessitent que les applications client utilisent des éléments encapsulés "document/littéral". Les applications client requièrent des classes auxiliaires pour leur permettre de générer les éléments d'encapsulation nécessaires.

Avant de commencer

Pour créer des classes auxiliaires, vous devez avoir préalablement exporté le fichier WSDL de l'API des services Web depuis l'environnement WebSphere Process Server.

A propos de cette tâche

Les opérations `call()` et `sendMessage()` des API de services Web déclenchent le lancement des processus BPEL dans WebSphere Process Server. Le message d'entrée de l'opération `call()` attend l'indication de l'encapsuleur document/littéral figurant dans le message d'entrée du processus BPEL. Pour générer les beans et les classes nécessaires aux processus BPEL, copiez l'élément `<wsdl:types>` dans un nouveau fichier XSD, puis utilisez l'outil `xsd.exe` pour générer des classes auxiliaires.

Procédure

1. Exportez le fichier WSDL de l'interface de processus BPEL depuis WebSphere Integration Developer, si vous n'avez pas déjà effectué cette opération.

2. Ouvrez le fichier WSDL dans un éditeur de texte ou un éditeur XML.
3. Copiez le contenu des éléments enfants de l'élément `<wsdl:types>` et copiez-le dans un nouveau fichier squelette XSD.
4. Appliquez l'outil `xsd.exe` au fichier XSD :

call xsd.exe file.xsd /classes /o

Où :

file.xsd

Fichier de définitions de schéma XML à convertir.

/classes (/c)

Génère des classes auxiliaires correspondant au contenu du ou des fichier(s) XSD spécifié(s).

/output (/o)

Spécifie le répertoire de sortie des fichiers générés. Si ce répertoire est omis, le répertoire par défaut est le répertoire en cours.

Par exemple :

call xsd.exe ProcessCustomer.xsd /classes /output:c:\temp

5. Ajout du fichier classe généré à votre application client. Si vous utilisez Visual Studio, par exemple, vous pouvez effectuer cette opération avec l'option de menu **Projet → Ajouter élément existant (Add Existing Item)**.

Exemple

Si le fichier `ProcessCustomer.wsdl` contient les éléments suivants :

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://com/ibm/bpe/unittest/sca"
  xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  name="ProcessCustomer"
  targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
  <wsdl:types>
    <xsd:schema targetNamespace="http://ProcessTypes/bpel/ProcessCustomer"
      xmlns:bons1="http://com/ibm/bpe/unittest/sca"
      xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
        schemaLocation="xsd-includes/http.com.ibm.bpe.unittest.sca.xsd"/>
      <xsd:element name="doit">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input1" nillable="true" type="bons1:Customer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="doitResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="output1" nillable="true" type="bons1:Customer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="doitRequestMsg">
    <wsdl:part element="tns:doit" name="doitParameters"/>
  </wsdl:message>
  <wsdl:message name="doitResponseMsg">
    <wsdl:part element="tns:doitResponse" name="doitResult"/>
  </wsdl:message>
</wsdl:definitions>
```

```

</wsdl:message>
  <wsdl:portType name="ProcessCustomer">
    <wsdl:operation name="doit">
      <wsdl:input message="tns:doitRequestMsg" name="doitRequest"/>
      <wsdl:output message="tns:doitResponseMsg" name="doitResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>

```

Le fichier XSD résultant contient les éléments suivants :

```

<xsd:schema xmlns:bons1="http://com/ibm/bpe/unittest/sca"
             xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
  <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
             schemaLocation="Customer.xsd"/>
  <xsd:element name="doit">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="input1" type="bons1:Customer" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="doitResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="output1" type="bons1:Customer" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>


```

Concepts associés

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Information associée

 Documentation Microsoft relative à l'outil XSD (XML Schema Definition, XSD.EXE)

Création d'une application client (.NET)

Une application client envoie des requêtes et reçoit des réponses vers et depuis les API de services Web. En utilisant un client proxy pour gérer les communications et des classes auxiliaires pour formater les types de données, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Avant de commencer

Avant de commencer à créer une application client, générez le client proxy et les classes auxiliaires éventuellement requises.

A propos de cette tâche

Vous pouvez développer des applications client .NET à l'aide de n'importe quel outil de développement compatible avec .NET, comme par exemple Visual Studio .NET. Vous pouvez créer tout type d'application .NET afin d'appeler les API de services Web génériques.

Procédure

1. Créez un projet d'application client. Vous pouvez par exemple créer une **application WinFX Windows®** dans Visual Studio.
2. Dans les options du projet, ajoutez une référence au fichier DLL (Dynamic Link Library) du client proxy. Ajoutez à votre projet toutes les classes auxiliaires contenant les définitions d'objets métier. Visual Studio, par exemple, vous pouvez effectuer cette opération avec l'option de menu **Projet → Ajouter élément existant (Add existing item)**.
3. Créez un objet client proxy. Par exemple :

```
HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService service =
    new HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService();
```

4. Déclarez tout type de données d'objet métier utilisé dans les messages transmis vers et depuis le service Web. Par exemple :

```
HTMClient.HTMReference.TKIID id = new HTMClient.HTMReference.TKIID();
```

```
ClipBG bg = new ClipBG();
Clip clip = new Clip();
```

5. Appelez les fonctions de service Web spécifiques et spécifiez les paramètres obligatoires éventuels. Par exemple, pour créer et démarrer une tâche utilisateur :

```
HTMClient.HTMReference.createAndStartTask task =
    new HTMClient.HTMReference.createAndStartTask();
HTMClient.HTMReference.StartTask sTask =
    new HTMClient.HTMReference.StartTask();
```

```
sTask.taskName = "SimpleTask";
sTask.taskNamespace = "http://myProcess/com/acme/task";
sTask.inputMessage = bg;
task.inputTask = sTask;
```

```
id = service.createAndStartTask(task).outputTask;
```

6. Les processus et les tâches distants sont identifiés par des ID persistants (id dans l'exemple précédent). Par exemple, pour réclamer une tâche utilisateur précédemment créée :

```
HTMClient.HTMReference.claimTask claim = new HTMClient.HTMReference.claimTask();
claim.inputTask = id;
```

Concepts associés

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Renforcement de la sécurité (.NET)

Vous pouvez sécuriser les communications des services Web en intégrant des mécanismes de sécurité à vos applications client.

A propos de cette tâche

Ces mécanismes de sécurité peuvent inclure le jeton de nom d'utilisateur (nom d'utilisateur et mot de passe) ou des jetons de sécurité binaires personnalisés et XML.

Procédure

1. Téléchargez et installez le module WSE (Web Services Enhancements) 2.0 SP3 pour Microsoft .NET. Ce module est accessible à l'adresse suivante : <http://www.microsoft.com/downloads/details.aspx?familyid=1ba1f631-c3e7-420a-bc1e-ef18bab66122&displaylang=en>

2. Modifiez comme suit le code client proxy généré.

Modifiez :

```
public class Export1_MyMicroflowHttpService : System.Web.Services.Protocols.SoapHttpClientProtocol {  
    En :  
public class Export1_MyMicroflowHttpService : Microsoft.Web.Services2.WebServicesClientProtocol {
```

Remarque : Ces modifications seront perdues si vous régénérez le client proxy en exécutant l'outil WSDL.exe.

3. Modifiez le code de l'application client en ajoutant les lignes suivantes en début de fichier :

```
using System.Web.Services.Protocols;  
using Microsoft.Web.Services2;  
using Microsoft.Web.Services2.Security.Tokens;  
...
```

4. Ajoutez le code de mise en oeuvre du mécanisme de sécurité souhaité. Le code suivant, par exemple, ajoute une protection par nom d'utilisateur et mot de passe :

```
string user = "U1";  
string pwd = "password";  
UsernameToken token = new UsernameToken(user, pwd, PasswordOption.SendPlainText);  
  
me._proxy.RequestSoapContext.Security.Tokens.Clear();  
me._proxy.RequestSoapContext.Security.Tokens.Add(token);
```

Concepts associés

Développement d'applications client dans l'environnement .NET

Microsoft .NET offre un puissant environnement de développement permettant de connecter des applications via des services Web.

Requêtes sur des objets liés aux processus métier et aux tâches

Vous pouvez utiliser les API de services Web pour effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données Business Process Choreographer, afin d'extraire les propriétés spécifiques de ces objets.

A propos de cette tâche

La base de données Business Process Choreographer stocke les données de modèle (model) et d'instance (runtime) nécessaires à la gestion des processus métier et des tâches.

Les applications client peuvent, par l'intermédiaire des API de services Web, extraire de la base de données des informations relatives aux processus métier et aux tâches.

Les applications client vous permettent d'effectuer une requête unique pour extraire une propriété particulière d'un objet. Vous pouvez sauvegarder les requêtes que vous exécutez le plus souvent. Ces requêtes stockées peuvent ensuite être extraites et utilisées par votre application client.

Concepts associés

Requêtes portant sur des objets liés aux processus métier et aux tâches à l'aide des API de services Web

L'interface de requête des API de service Web vous permet d'obtenir des informations stockées relatives aux processus métier et aux tâches.

Tâches associées

Développement d'applications API de service Web

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches utilisateur via des API de services Web.

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Requêtes portant sur des objets liés aux processus métier et aux tâches à l'aide des API de services Web

L'interface de requête des API de service Web vous permet d'obtenir des informations stockées relatives aux processus métier et aux tâches.

Les applications client utilisent une syntaxe de type SQL pour interroger la base de données.

Exemple de services Java Web

```
string processTemplateName = "ProcessCustomerLR";
query query1 = new query();
query1.selectClause = "DISTINCT PROCESS_INSTANCE.STARTED, PROCESS_INSTANCE.PIID";
query1.whereClause =
    "PROCESS_INSTANCE.TEMPLATE_NAME = '" + processTemplateName + "'";
query1.orderByClause = "PROCESS_INSTANCE.STARTED";
query1.threshold = null;
query1.timeZone = "UTC"; query1.skipTuples = null;
queryResponse queryResponse1 = proxy.query(query1);
```

Les informations extraites de la base de données sont renvoyées via les API de service Web sous forme d'*ensemble de résultats de requête*.

Par exemple :

```
QueryResultSetType queryResultSet = queryResponse1.queryResultSet;
if (queryResultSet != null) {
    Console.WriteLine("--> QueryResultSetType");
    Console.WriteLine(" . size= " + queryResultSet.size);
    Console.WriteLine(" . numberColumns= " + queryResultSet.numberColumns);
    string indent = " . ";

    // -- the query column info
    QueryColumnInfoType[] queryColumnInfo = queryResultSet.QueryColumnInfo;
    if (queryColumnInfo.Length > 0) {
        Console.WriteLine();
        Console.WriteLine("= . QueryColumnInfoType size= " + queryColumnInfo.Length);
        Console.WriteLine(" | tableName ");
        for (int i = 0; i < queryColumnInfo.Length ; i++) {
            Console.WriteLine(" | " + queryColumnInfo[i].tableName.PadLeft(20) );
        }
        Console.WriteLine();
        Console.WriteLine(" | columnName ");
        for (int i = 0; i < queryColumnInfo.Length ; i++) {
            Console.WriteLine(" | " + queryColumnInfo[i].columnName.PadLeft(20) );
        }
        Console.WriteLine();
    }
}
```

```

        Console.WriteLine( " | data type ");
        for (int i = 0; i < queryColumnInfo.Length ; i++) {
            QueryColumnInfoType tt = queryColumnInfo[i].type;
            Console.WriteLine( " | " + tt.ToString());
        }
        Console.WriteLine();
    }
    else {
        Console.WriteLine("--> queryColumnInfo= <null>");
    }
}

// - the query result values
string[][] result = queryResultSet.result;
if (result !=null) {
    Console.WriteLine();
    Console.WriteLine("= . result size= " + result.Length);
    for (int i = 0; i &lt; result.Length; i++) {
        Console.WriteLine(indent + i );
        string[] row = result[i];
        for (int j = 0; j &lt; row.Length; j++ ) {
            Console.WriteLine(" | " + row[j]);
        }
        Console.WriteLine();
    }
}
else {
    Console.WriteLine("--> result= <null>");
}
}
else {
    Console.WriteLine("--> QueryResultSetType= <null>");
}
}

```

La fonction de requête renvoie des éléments en fonction des droits d'accès de l'appelant. L'ensemble de résultats de requête contient uniquement les propriétés des objets que l'appelant est autorisé à consulter.

Des vues prédéfinies des bases de données sont disponibles pour vous permettre de rechercher les propriétés de l'objet. Pour les modèles de processus, la fonction de requête possède la syntaxe suivante :

```

ProcessTemplateData[] queryProcessTemplates
    (java.lang.String whereClause,
     java.lang.String orderByClause,
     java.lang.Integer threshold,
     java.util.TimeZone timezone);

```

Pour les modèles de tâches, la fonction de requête présente la syntaxe suivante :

```

TaskTemplate[] queryTaskTemplates
    (java.lang.String whereClause,
     java.lang.String orderByClause,
     java.lang.Integer threshold,
     java.util.TimeZone timezone);

```

Pour d'autres objets liés aux processus métier et aux tâches, la fonction de requête a la syntaxe suivante :

```

QueryResultSet query (java.lang.String selectClause,
                     java.lang.String whereClause,
                     java.lang.String orderByClause,
                     java.lang.Integer skipTuples
                     java.lang.Integer threshold,
                     java.util.TimeZone timezone);

```

L'interface de requête contient également une méthode queryAll. Vous pouvez utiliser cette méthode pour extraire toutes les données pertinentes concernant un objet, par exemple, à des fins de contrôle. L'appelant de la méthode queryAll doit disposer de l'un des rôles Java 2 Platform, Enterprise Edition (J2EE) suivants : BPESystemAdministrator, BPESystemMonitor, TaskSystemAdministrator ou TaskSystemMonitor. Le contrôle de l'autorisation à l'aide de l'élément de travail correspondant de l'objet n'est pas appliqué.

Exemple pour .NET

```
ProcessTemplateType[] templates = null;

try {
    queryProcessTemplates iW = new queryProcessTemplates();
    iW.whereClause = "PROCESS_TEMPLATE.STATE=PROCESS_TEMPLATE.STATE.STATE_STARTED";
    iW.orderByClause = null;
    iW.threshold = null;
    iW.timeZone = null;

    Console.WriteLine("--> queryProcessTemplates ... ");
    Console.WriteLine("--> query: WHERE " + iW.whereClause + " ORDER BY " +
        iW.orderByClause + " THRESHOLD " + iW.threshold + " TIMEZONE " + iW.timeZone);

    templates = proxy.queryProcessTemplates(iW);

    if (templates.Length < 1) {
        Console.WriteLine("--> No templates found :-(");
    }
    else {
        for (int i = 0; i < templates.Length ; i++) {
            Console.WriteLine("--> found template with ptid: " + templates[i].ptid);
            Console.WriteLine(" and name: " + templates[i].name);
            /* ... other properties of ProcessTemplateType ... */
        }
    }
}
catch( Exception e ) {
    Console.WriteLine("exception= " + e);
}
```

Tâches associées

Requêtes sur des objets liés aux processus métier et aux tâches
Vous pouvez utiliser les API de services Web pour effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données Business Process Choreographer, afin d'extraire les propriétés spécifiques de ces objets.

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

A propos de cette tâche

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Une requête privée et une requête publique peuvent être sauvegardées sous le même nom. Les requêtes enregistrées par différents utilisateurs peuvent également avoir un nom identique.

Vous pouvez avoir stocké des requêtes pour des objets de processus métier, des objets de tâche ou une combinaison de ces deux types d'objets.

Tâches associées

Requêtes sur des objets liés aux processus métier et aux tâches

Vous pouvez utiliser les API de services Web pour effectuer des requêtes de données sur les objets liés aux processus métier et aux tâches dans la base de données Business Process Choreographer, afin d'extraire les propriétés spécifiques de ces objets.

Gestion des requêtes stockées publiques

Les requêtes stockées publiques sont créées par l'administrateur système. Ces requêtes sont accessibles à tous les utilisateurs.

Gestion de requêtes stockées privées pour d'autres utilisateurs

Tout utilisateur peut créer des requêtes privées. Seul le propriétaire d'une requête et l'administrateur système peuvent les utiliser.

Gestion des requêtes stockées privées

Si vous n'êtes pas un administrateur système, vous pouvez créer, exécuter et supprimer vos propres requêtes stockées privées. Vous pouvez également utiliser les requêtes stockées publiques créées par l'administrateur système.

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

A propos de cette tâche

Les applications client JMS échangent des messages de demande et de réponse avec l'API JMS. Pour créer un message de demande, l'application client remplit le corps du message JMS TextMessage avec un élément XML représentant l'encapsuleur document/littéral de l'opération correspondante.

Concepts associés

Exigences des processus métier

Les processus métier développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via l'API JMS.

Autorisation pour les affichages JMS

Pour autoriser l'accès à l'interface JMS, des paramètres de sécurité doivent être activés dans WebSphere Application Server.

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches utilisateur

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches utilisateur. Chacune de ces interfaces présente des caractéristiques différentes.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Accès à l'interface JMS

Pour envoyer et recevoir des messages par le biais de l'interface JMS, une application doit d'abord créer une connexion au bus BPC.cellname.Bus, créer une session, puis générer des expéditeurs et des destinataires de message.

Copie d'artefacts pour les applications client JMS

Un certain nombre d'artefacts peuvent être copiés à partir de l'environnement WebSphere Process Server pour faciliter la création d'applications client JMS.

Vérification du message de réponse pour les exceptions de métier

Les applications client JMS doivent vérifier l'en-tête de message de tous les messages de réponse pour les exceptions de métier.

Exemple : exécution d'un processus de longue durée à l'aide de l'API JMS de Business Process Choreographer

Cet exemple montre comment créer une application client générique utilisant l'API JMS pour exploiter des processus de longue durée.

Exigences des processus métier

Les processus métier développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via l'API JMS.

Les exigences sont les suivantes :

1. Les interfaces des processus métier doivent être définies à l'aide du style "document/literal wrapped" défini dans l'API Java pour la spécification XML-RPC (JAX-RPC 1.1). Il s'agit du style par défaut défini pour l'ensemble des processus métier et des tâches utilisateur développés avec WebSphere Integration Developer.
2. Les messages d'erreur accessibles aux processus métier et aux tâches utilisateur des opérations de service Web doivent comprendre un seul composant de message WSDL défini au moyen d'un élément de schéma XML. Par exemple :

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Information associée

➡ Page de téléchargement d'API Java pour XML-RPC (JAX-RPC)

➡ Quel style de langage WSDL dois-je utiliser ?

Autorisation pour les affichages JMS

Pour autoriser l'accès à l'interface JMS, des paramètres de sécurité doivent être activés dans WebSphere Application Server.

Lorsque le conteneur de processus métier est installé, le rôle **JMSAPIUser** doit être mappé avec un ID utilisateur. Cet ID utilisateur permet d'émettre toutes les demandes de l'API JMS. Par exemple, si **JMSAPIUser** est mappé avec "Utilisateur A", toutes les demandes de l'API JMS apparaissent dans le moteur de processus avec pour origine "Utilisateur A".

Le rôle **JMSAPIUser** doit être affecté aux autorités suivantes :

Demande	Autorisation requise
forceTerminate	Administrateur de processus
sendEvent	Propriétaire potentiel d'activité ou administrateur de processus

Remarque : Pour toutes les demandes, aucune autorisation spéciale n'est requise.

L'autorité spéciale est accordée à une personne avec le rôle d'administrateur de processus métier. Un administrateur de processus métier est un rôle spécial. Il est différent de celui de l'administrateur de processus d'une instance de processus. Il dispose de tous les privilèges.

Vous ne pouvez pas supprimer l'ID utilisateur du lanceur de processus à partir de votre registre des utilisateurs alors que l'instance du processus existe. Si vous supprimez cet ID utilisateur, la navigation dans ce processus ne peut se poursuivre. Vous recevrez l'exception suivante dans le fichier journal du système :
no unique ID for: <ID utilisateur>

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Accès à l'interface JMS

Pour envoyer et recevoir des messages par le biais de l'interface JMS, une application doit d'abord créer une connexion au bus BPC.cellname.Bus, créer une session, puis générer des expéditeurs et des destinataires de message.

A propos de cette tâche

Le serveur de processus accepte les messages Java Message Service (JMS) qui suivent le paradigme point-à-point. Une application qui envoie ou qui reçoit des messages JMS doit exécuter les actions suivantes.

L'exemple suivant suppose que le client JMS est exécuté dans un environnement géré (Enterprise JavaBeans, client d'application ou conteneur de client Web). Si vous voulez exécuter le client JMS dans un environnement J2SE, consultez la rubrique "Client IBM pour JMS sur J2SE avec IBM WebSphere Application Server" à la page <http://www-1.ibm.com/support/docview.wss?uid=swg24012804>.

Procédure

1. Créez une connexion au `BPC.nomcellule.Bus`. Il n'existe pas de fabrique de connexions préconfigurée pour les requêtes d'une application client : l'application client peut soit utiliser la commande `ReplyConnectionFactory` de l'API JMS, soit créer sa propre fabrique de connexions, auquel cas elle peut utiliser la recherche JNDI (Java Naming and Directory Interface) pour récupérer la fabrique de connexions. Le nom de recherche JNDI doit être identique au nom indiqué lors de la configuration de la file d'attente des demandes externes de Business Process Choreographer. L'exemple suivant suppose que l'application client crée sa propre fabrique de connexions nommée "jms/clientCF".

```
//Obtain the default initial JNDI context.
Context initialContext = new InitialContext();

// Look up the connection factory.
// Create a connection factory that connects to the BPC bus.
// Call it, for example, "jms/clientCF".
// Also configure an appropriate authentication alias.
ConnectionFactory connectionFactory =
    (ConnectionFactory)initialContext.lookup("jms/clientCF");

// Create the connection.
Connection connection = connectionFactory.createConnection();
```

2. Créez une session afin de pouvoir créer les expéditeurs et les destinataires de message.

```
// Create a transaction session using auto-acknowledgement.
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. Créez un expéditeur de message pour envoyer les messages. Le nom de recherche JNDI doit être identique au nom indiqué lors de la configuration de la file d'attente des demandes externes de Business Process Choreographer.

```
// Look up the destination of the Business Process Choreographer input queue to
// send messages to.
Queue sendQueue = (Queue) initialContext.lookup("jms/BFMJMSAPIQueue");
```

```
// Create a message producer.
MessageProducer producer = session.createProducer(sendQueue);
```

4. Créez un destinataire de message pour recevoir les réponses. Le nom de recherche JNDI de la destination de la réponse peut indiquer une destination définie par l'utilisateur, mais il peut également indiquer la destination de la réponse par défaut (définie par Business Process Choreographer) `jms/BFMJMSReplyQueue`. Dans les deux cas, la destination de la réponse doit être basée sur `BPC.<cellname>.Bus`.

```

// Look up the destination of the reply queue.
Queue replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

// Create a message consumer.
MessageConsumer consumer = session.createConsumer(replyQueue);

```

5. Envoyez un message.

```

// Start the connection.
connection.start();

// Create a message - see the task descriptions for examples - and send it.
// This method is defined elsewhere ...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Set mandatory JMS header.
// targetFunctionName is the operation name of JMS API
// (for example, getProcessTemplate, sendMessage)
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Set the reply queue; this is mandatory if the replyQueue
// is not the default queue (as it is in this example).
requestMessage.setJMSReplyTo(replyQueue);

// Send the message.
producer.send(requestMessage);

// Get the message ID.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();

```

6. Recevez la réponse.

```

// Receive the reply message and analyse the reply.
TextMessage replyMessage = (TextMessage) consumer.receive();

// Get the payload.
String payload = replyMessage.getText();

session.commit();

```

7. Mettez fin à la connexion, puis libérez les ressources.

```

// Final housekeeping; free the resources.
session.close();
connection.close();

```

Remarque : Vous n'êtes pas obligé de mettre fin à la connexion après chaque transaction. Une fois la connexion démarrée, vous pouvez échanger n'importe quel nombre de messages de demande et de réponse avant de mettre fin à la connexion. L'exemple illustre un cas simple avec un appel unique au sein d'une méthode métier unique.

Concepts associés

Structure d'un message JMS de Business Process Choreographer
L'en-tête et le corps d'un message JMS doivent avoir une structure prédéfinie.

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Structure d'un message JMS de Business Process Choreographer

L'en-tête et le corps d'un message JMS doivent avoir une structure prédéfinie.

Un message JMS (Java Message Service) se compose des éléments suivants :

- Un en-tête de message pour l'identification du message et l'acheminement de l'information.
- Le corps (charge) du message qui renferme le contenu.

Business Process Choreographer ne prend en charge que les formats de message texte.

En-tête de message

JMS permet aux clients d'accéder à certains champs d'en-tête de message.

Les champs d'en-tête suivants peuvent être définis par un client JMS de Business Process Choreographer :

- **JMSReplyTo**

Destination à laquelle est envoyée la réponse à une requête. Si ce champ n'est pas spécifié dans le message de requête, la réponse est alors envoyée à la destination de réponse par défaut de l'interface d'exportation (l'exportation correspond à l'affichage de l'interface client d'un composant de processus métier). Il est possible d'obtenir cette destination à l'aide de `initialContext.lookup("jms/BFMJMSReplyQueue")`;

- **TargetFunctionName**

Le nom de l'opération WSDL pourrait être "queryProcessTemplates", par exemple. Ce champ doit toujours être défini. Notez que TargetFunctionName spécifie l'opération de l'interface du message JMS générique décrite ici. A ne pas confondre avec les opérations fournies par des tâches ou des processus concrets pouvant être appelés indirectement à l'aide de l'opération **call** ou **sendMessage**, par exemple.

Un client Business Process Choreographer peut également accéder aux champs d'en-tête suivants :

- **JMSMessageID**

Identifie un message de manière unique. Défini par le fournisseur JMS lorsque le message est envoyé. Si le client définit le champ JMSMessageID avant l'envoi du message, il est systématiquement remplacé par le fournisseur JMS. Si l'ID du message est requis à des fins d'authentification, le client peut alors obtenir le paramètre JMSMessageID après l'envoi du message.

- **JMSCorrelationID**

Relie les messages. Ne pas définir ce champ. Un message de réponse Business Process Choreographer contient toujours le champ JMSMessageID du message de requête.

Chaque message de réponse contient les champs d'en-tête JMS suivants :

- **IsBusinessException**

"False" pour les messages de sortie WSDL ou "True" pour les messages d'erreur WSDL.

Les exceptions `ServiceRuntimeExceptions` ne sont pas renvoyées aux applications client asynchrones. Lorsqu'une exception sévère se produit lors du traitement d'un message de requête JMS, une erreur d'exécution est générée, ce qui provoque l'annulation de la transaction en cours de traitement. Le message de requête JMS est alors relivré. Si l'erreur se produit prématurément dans la phase d'exportation SCA du traitement du message (par exemple, lors de sa désérialisation), de nouvelles tentatives sont exécutées jusqu'au nombre maximum de livraisons échouées spécifié par la destination de réception de la fonction d'exportation SCA. Une fois ce nombre atteint, le message de requête est ajouté à la destination d'exception système du bus Business Process Choreographer. Cependant, si l'échec se produit lors du traitement réel de la requête par le composant SCA de Business Flow Manager, le message de requête échoué est géré par l'infrastructure de gestion des événements en échec de WebSphere Process Server, autrement dit, on se retrouve dans la base de données de gestion des événements échoués si les tentatives ne permettent pas de résoudre la situation exceptionnelle.

Corps du message

Le corps du message JMS est une chaîne contenant un document XML représentant l'élément encapsuleur du document/littéral de l'opération.

Voici l'exemple simple d'un corps de message de requête valide :

```
<?xml version="1.0" encoding="UTF-8"?>
<_6:queryProcessTemplates xmlns:_6="http://www.ibm.com/xmlns/prod/
  websphere/business-process/services/6.0">
<whereClause>PROCESS_TEMPLATE.STATE IN (1)</whereClause>
</_6:queryProcessTemplates>
```

Tâches associées

Accès à l'interface JMS

Pour envoyer et recevoir des messages par le biais de l'interface JMS, une application doit d'abord créer une connexion au bus `BPC.cellname.Bus`, créer une session, puis générer des expéditeurs et des destinataires de message.

Copie d'artefacts pour les applications client JMS

Un certain nombre d'artefacts peuvent être copiés à partir de l'environnement WebSphere Process Server pour faciliter la création d'applications client JMS.

A propos de cette tâche

Ces artefacts sont obligatoires uniquement si vous utilisez `BOXMLSerializer` pour créer le corps du message JMS. Pour l'API JMS, ces artefacts sont:

- BFMIF.wsdl
- BFMIF.xsd
- BPCGen.xsd
- wsa.xsd

Vous pouvez obtenir ces artefacts de différentes manières :

- Publiez et exportez les artefacts à partir de l'environnement WebSphere Process Server.

Ces artefacts client se trouvent dans le répertoire *racine_installation*\ProcessChoreographer\client.

- Copiez les fichiers qui se trouvent dans le répertoire *racine_installation*\ProcessChoreographer\client du CD du client WebSphere Process Server.

Résultats

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Vérification du message de réponse pour les exceptions de métier

Les applications client JMS doivent vérifier l'en-tête de message de tous les messages de réponse pour les exceptions de métier.

A propos de cette tâche

Une application client JMS doit d'abord vérifier la propriété **IsBusinessException** de l'en-tête du message de réponse.

Par exemple :

Exemple

```
// receive response message
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject()).receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
    // strResponse is a bussiness fault
    // any api can end w/a processFaultMsg
    // the call api also w/a businessFaultMsg
}
else {
    // strResponse is the output message
}
}
```

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Exemple : exécution d'un processus de longue durée à l'aide de l'API JMS de Business Process Choreographer

Cet exemple montre comment créer une application client générique utilisant l'API JMS pour exploiter des processus de longue durée.

Procédure

1. Configurez l'environnement JMS, comme décrit dans «Accès à l'interface JMS», à la page 399.
2. Obtenez une liste des définitions de processus installées.
 - Envoyez la commande `queryProcessTemplates`.
 - Cette commande renvoie une liste d'objets `ProcessTemplate`.

3. Obtenez une liste d'activités de démarrage (activités de réception ou de sélection avec `createInstance="yes"`).
 - Envoyez `getStartActivities`.
 - Cette commande renvoie une liste d'objets `InboundOperationTemplate`.
4. Créez un message d'entrée. Ce message est propre à l'environnement et peut nécessiter l'emploi d'artefacts prédéployés, propres à chaque processus.
5. Créez une instance de processus.
 - Emettez une instruction `sendMessage`.

Grâce à l'API JMS, vous pouvez également utiliser l'opération `call` pour l'interaction avec des opérations de demande-réponse de longue durée fournies par un processus métier. Cette opération renvoie le résultat ou l'erreur d'opération à la destination de réponse spécifiée, même après une longue période. Par conséquent, si vous utilisez l'opération `call`, il n'est pas nécessaire d'utiliser les opérations `query` et `getOutputMessage` pour que le message de sortie de processus ou d'erreur s'affiche.
6. Facultatif : Obtenez les messages de sortie des instances de processus en répétant la procédure suivante :
 - a. Emettez la commande `query` pour obtenir l'état achevé de l'instance de processus.
 - b. Emettez la commande `getOutputMessage`.
7. Facultatif : Travaillez maintenant sur les opérations supplémentaires présentées par le processus :
 - a. Envoyez les commandes `getWaitingActivities` ou `getActiveEventHandlers` pour obtenir une liste des objets `InboundOperationTemplate`.
 - b. Créez des messages d'entrée.
 - c. Envoyez les messages à l'aide de la commande `sendMessage`.
8. Facultatif : Obtenez et définissez des propriétés personnalisées définies sur le processus ou les activités qu'il contient, en utilisant les commandes `getCustomProperties` et `setCustomProperties`.
9. Terminez le travail sur l'instance de processus :
 - a. Envoyez `delete` et `terminate` pour mettre fin au processus de longue durée.

Tâches associées

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

A propos de cette tâche

WebSphere Integration Developer permet de générer une application Web. Pour les applications comprenant des tâches utilisateurs, vous pouvez générer un client JSF personnalisé. Pour plus d'informations sur la génération d'un client JSF, consultez le centre de documentation de WebSphere Integration Developer.

Vous pouvez également développer votre client Web à l'aide des composants JSF fournis par Business Process Choreographer.

Procédure

1. Créez un projet dynamique et modifiez les propriétés Web Project Features pour inclure les composants de base JSF.
Pour plus d'informations sur la création d'un projet Web, accédez au centre de documentation de WebSphere Integration Developer.
2. Ajoutez les fichiers archive Java (JAR) préalables de Business Process Choreographer Explorer.

Ajoutez les fichiers suivants au répertoire WEB-INF/lib de votre projet :

- bpcclientcore.jar
- bfmclientmodel.jar
- htmclientmodel.jar
- bpcjsfcomponents.jar

Si vous déployez votre application web sur un serveur distant, ajoutez également les fichiers suivants. Ces fichiers sont nécessaires pour accéder à distance aux API de Business Process Choreographer.

- bpe137650.jar
- task137650.jar

Dans WebSphere Process Server, ces fichiers se trouvent tous dans le répertoire suivant :

- Sous Windows : *racine_installation*\ProcessChoreographer\client
- Sur les systèmes UNIX[®], Linux[®] et i5/OS : *racine_installation*/ProcessChoreographer/client

3. Ajoutez les références EJB requises pour le descripteur de déploiement d'applications Web, le fichier web.xml.

```
<ejb-ref id="EjbRef_1">
  <ejb-ref-name>ejb/BusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
  <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
<ejb-ref id="EjbRef_2">
  <ejb-ref-name>ejb/HumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.task.api.HumanTaskManagerHome</home>
  <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
<ejb-local-ref id="EjbLocalRef_1">
  <ejb-ref-name>ejb/LocalBusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
  <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
</ejb-local-ref>
<ejb-local-ref id="EjbLocalRef_2">
  <ejb-ref-name>ejb/LocalHumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
  <local>com.ibm.task.api.LocalHumanTaskManager</local>
</ejb-local-ref>
```

4. Ajoutez les composants JSF de Business Process Choreographer Explorer à l'application JSF.
 - a. Ajoutez les références de bibliothèque de balises requises pour les applications dans les fichiers JavaServer Pages (JSP). En généralement, les

ressources requises sont les bibliothèques de balises JSF et HTML et la bibliothèque de balises requise par les composants JSF.

- `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>`
- `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`
- `<%@ taglib uri="http://com.ibm.bpe.jsf/taglib" prefix="bpe" %>`

b. Ajoutez une balise `<f:view>` au corps de la page JSP et une balise `<h:form>` à la balise `<f:view>`.

c. Ajoutez les composants JSF aux fichiers JSP.

Selon votre application, ajoutez les composants List, Details, CommandBar ou Message aux fichiers JSP. Vous pouvez ajouter plusieurs instances à chaque composant.

d. Configurez les beans gérés dans le fichier de configuration JSF.

Le fichier de configuration par défaut est `faces-config.xml`. Ce fichier réside dans le répertoire `WEB-INF` de l'application Web.

Selon le composant que vous ajoutez à votre fichier JSP, vous devez également ajouter les références à la requête et aux objets d'encapsulation au fichier de configuration JSF. Pour s'assurer d'un traitement correct des erreurs, vous devez également définir un bean d'erreur et une cible de navigation pour la page d'erreur dans le fichier de configuration JSF. Utilisez `BPCError` comme nom pour le bean d'erreur et `error` comme nom pour la cible de navigation de la page d'erreur.

```
<faces-config>
...
<managed-bean>
  <managed-bean-name>BPCError</managed-bean-name>
  <managed-bean-class>com.ibm.bpc.clientcore.util.ErrorBeanImpl
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

...
<navigation-rule>
...
<navigation-case>
<description>
Page générale des erreurs.
</description>
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

e. Implémentez le code personnalisé requis pour la prise en charge des composants JSF.

5. Déployez l'application.

Si vous déployez l'application dans un environnement de déploiement réseau, modifiez les noms JNDI (Java Naming and Directory Interface) des ressources cible avec des valeurs permettant de trouver les API de Business Flow Manager et Human Task Manager dans votre cellule.

- Si vos conteneurs de processus métier sont configurés sur un autre serveur au sein de la même cellule gérée, les noms se présentent de la manière suivante :

cellule/noeuds/nomnoeud/serveurs/nomserveur/com/ibm/bpe/api/BusinessManagerHome
cellule/noeuds/nomnoeud/serveurs/nomserveur/com/ibm/task/api/HumanTaskManagerHome

- Si vos conteneurs de processus métier sont configurés sur un serveur au sein de la même cellule, les noms se présentent de la manière suivante :

cellule/clusters/nomcluster/com/ibm/bpe/api/BusinessFlowManagerHome
cellule/clusters/nomcluster/com/ibm/task/api/HumanTaskManagerHome

Mappez les références EJB avec les noms JNDI ou ajoutez manuellement les références au fichier `ibm-web-bnd.xml`.

Le tableau suivant dresse la liste des liaisons de référence et leurs mappages par défaut.

Tableau 50. Mappage des liaisons de référence aux noms JNDI

Liaison de référence	Nom JNDI	Commentaires
<code>ejb/BusinessProcessHome</code>	<code>com/ibm/bpe/api/BusinessFlowManagerHome</code>	Bean session distant
<code>ejb/LocalBusinessProcessHome</code>	<code>com/ibm/bpe/api/BusinessFlowManagerHome</code>	Bean session local
<code>ejb/HumanTaskManagerEJB</code>	<code>com/ibm/task/api/HumanTaskManagerHome</code>	Bean session distant
<code>ejb/LocalHumanTaskManagerEJB</code>	<code>com/ibm/task/api/HumanTaskManagerHome</code>	Bean session local

Résultats

Votre application Web déployée contient les fonctionnalités fournies par les composants de Business Process Choreographer Explorer.

Que faire ensuite

Si vous utilisez des JSP personnalisés pour les messages de processus et de tâche, vous devez mapper les modules web qui sont utilisés pour déployer les JSP avec les mêmes serveurs que ceux avec lesquels est mappé le client JSF personnalisé.

Concepts associés

Composants Exemples de Business Process Choreographer Explorer

Les composants Business Process Choreographer Explorer constituent un ensemble d'éléments réutilisables configurables basés sur la technologie JavaServer Faces (JSF). Vous pouvez imbriquer ces éléments dans des applications Web. Les applications Web peuvent alors accéder à des applications de processus métier et de tâches utilisateur installées.

Traitement des erreurs dans les composants JSF

Les composants JavaServer Faces (JSF) exploitent un bean géré prédéfini, `BPCError`, pour le traitement des erreurs. Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Ajout du composant List à une application JSF

Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Ajout du composant Details à une application JSF

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, de tâches élémentaires, d'instances de processus et de modèles de processus.

Ajout du composant CommandBar à une application JSF

Utilisez le composant CommandBar de Business Process Choreographer Explorer pour permettre l'affichage d'une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d'un objet ou des objets sélectionnés d'une liste.

Ajout du composant Message à une application JSF

Le composant Message de l'explorateur du Chorégraphe de processus métier permet d'afficher des objets de données et des types de primitive dans une application JavaServer Faces (JSF).

Référence associée

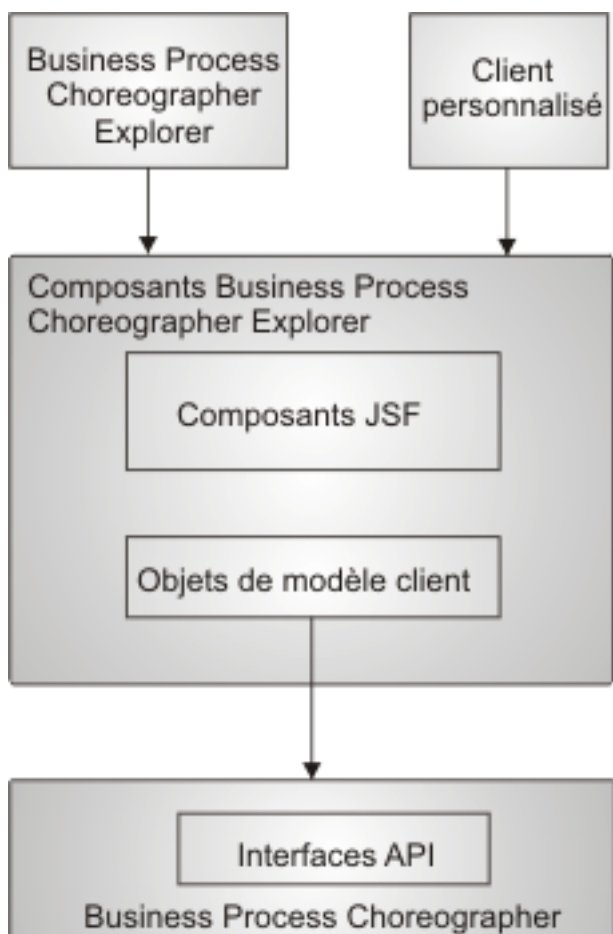
Convertisseurs et intitulés par défaut d'objets de modèle client

Les objets de modèle client implémentent les interfaces correspondantes de l'API de Business Process Choreographer.

Composants Exemples de Business Process Choreographer Explorer

Les composants Business Process Choreographer Explorer constituent un ensemble d'éléments réutilisables configurables basés sur la technologie JavaServer Faces (JSF). Vous pouvez imbriquer ces éléments dans des applications Web. Les applications Web peuvent alors accéder à des applications de processus métier et de tâches utilisateur installées.

Les composants consistent en un ensemble de composants JSF et un ensemble d'objets modèle client. La relation entre les composants et Business Process Choreographer, Business Process Choreographer Explorer et d'autres clients personnalisés est représentée dans la figure suivante.



Composants JSF

Les composants de Business Process Choreographer Explorer comprennent les composants JSF suivants. Ces composants JSF sont insérés dans les fichiers JavaServer Pages (JSP) lorsque vous générez des applications Web de gestion des processus métier et tâches utilisateur.

- Composant List

Le composant List affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades. Ce composant possède un gestionnaire de liste associé.

- **Composant Details**
Le composant Details permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus. Ce composant possède un gestionnaire de détails associé.
- **Composant CommandBar**
Le composant CommandBar permet d'afficher une barre avec boutons de commande. Ces boutons représentent des commandes qui agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste. Ces objets sont fournis par un gestionnaire de listes ou un gestionnaire de détails.
- **Composant Message**
Le composant Message affiche un message pouvant contenir un objet SDO (Service Data Object) ou un type simple.

Objets de modèle client

Les objets de modèle client sont utilisés avec les composants JSF. Les objets implémentent certaines interfaces de l'API de Business Process Choreographer sous-jacent et encapsule l'objet d'origine. Les objets de modèle client fournissent un support multilingue pour les libellés et les convertisseurs de certaines propriétés.

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Traitement des erreurs dans les composants JSF

Les composants JavaServer Faces (JSF) exploitent un bean géré prédéfini, `BPCError`, pour le traitement des erreurs. Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

Ce bean met en oeuvre l'interface `com.ibm.bpc.clientcore.util.ErrorBean`. L'affichage de la page d'erreur a lieu dans les cas suivants :

- Lorsqu'une erreur se produit durant l'exécution d'une requête définie pour un gestionnaire de listes, et que cette erreur est générée en tant qu'erreur `ClientException` par la méthode `execute` d'une commande
- Lorsqu'une erreur `ClientException` est émise par la méthode `execute` d'une commande et qu'il ne s'agit pas d'une erreur `ErrorsInCommandException`, ou qu'elle ne met pas en oeuvre l'interface `CommandBarMessage`
- Si un message d'erreur est affiché dans le composant et que vous suivez l'hyperlien lié au message

Une mise en oeuvre par défaut de l'interface `com.ibm.bpc.clientcore.util.ErrorBeanImpl` est disponible.

L'interface est définie comme suit :

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * Cette méthode d'accès set permet de transmettre l'environnement
     * local et l'exception. Ainsi, les méthodes getMessage
     */
}
```

```

    * peuvent renvoyer des chaînes localisées
    *
    */
    public void setException(Exception ex, Locale locale);

    public Exception getException();
    public String getStack();
    public String getNestedExceptionMessage();
    public String getNestedExceptionStack();
    public String getRootExceptionMessage();
    public String getRootExceptionStack();

    /*
    * Cette méthode renvoie le message d'exception
    * concaténé de façon récursive avec les messages de
    * toutes les exceptions imbriquées.
    */
    public String getAllExceptionMessages();

    /*
    * Cette méthode renvoie la pile d'exceptions
    * concaténée de façon récursive avec les piles
    * toutes les exceptions imbriquées.
    */
    public String getAllExceptionStacks();
}

```

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Convertisseurs et intitulés par défaut d'objets de modèle client

Les objets de modèle client implémentent les interfaces correspondantes de l'API de Business Process Choreographer.

Les composants List et Details fonctionnent sur tout type de bean. Vous pouvez afficher toutes les propriétés d'un bean. Toutefois, si vous voulez définir les convertisseurs et les intitulés utilisés pour les propriétés d'un bean, vous devez utiliser soit la balise column du composant List, soit la balise property du composant Details. Au lieu de définir les convertisseurs et les intitulés, vous pouvez définir des convertisseurs et des intitulés par défaut pour les propriétés en définissant les méthodes statiques suivantes. Vous pouvez définir les méthodes statiques suivantes :

```

static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
    getConverter(String property);

```

Le tableau suivant répertorie les objets de modèle client qui implémentent les classes d'API Business Flow Manager et Human Task Manager et fournissent les intitulés et le convertisseur par défaut pour leurs propriétés. Cet encapsulage des interfaces fournit des intitulés sensibles et des convertisseurs pour un ensemble de propriétés. Le tableau suivant répertorie les correspondances entre les interfaces de Business Process Choreographer et les objets de modèle client.

Tableau 51. Mappage d'interfaces de Business Process Choreographer avec des objets de modèle client

Interface de Business Process Choreographer	Classe d'objet de modèle client
com.ibm.bpe.api.ActivityInstanceData	com.ibm.bpe.clientmodel.bean.ActivityInstanceBean
com.ibm.bpe.api.ActivityServiceTemplateData	com.ibm.bpe.clientmodel.bean.ActivityServiceTemplateBean
com.ibm.bpe.api.ProcessInstanceData	com.ibm.bpe.clientmodel.bean.ProcessInstanceBean
com.ibm.bpe.api.ProcessTemplateData	com.ibm.bpe.clientmodel.bean.ProcessTemplateBean
com.ibm.task.api.Escalation	com.ibm.task.clientmodel.bean.EscalationBean
com.ibm.task.api.Task	com.ibm.task.clientmodel.bean.TaskInstanceBean
com.ibm.task.api.TaskTemplate	com.ibm.task.clientmodel.bean.TaskTemplateBean

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Ajout du composant List à une application JSF

Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Procédure

1. Ajoutez le composant List au fichier JavaServer Pages (JSP).

Ajoutez la balise `bpe:list` à la balise `h:form`. La balise `bpe:list` doit contenir un attribut de modèle. Ajoutez des balises `bpe:column` à la balise `bpe:list` pour ajouter les propriétés des objets qui doivent figurer à chaque ligne de la liste.

L'exemple suivant illustre l'ajout d'un composant List afin d'afficher des instances de tâche.

```
<h:form>

    <bpe:list model="#{TaskPool}">
        <bpe:column name="name" action="taskInstanceDetails" />
        <bpe:column name="state" />
        <bpe:column name="kind" />
        <bpe:column name="owner" />
        <bpe:column name="originator" />
    </bpe:list>

</h:form>
```

L'attribut de modèle fait référence à un bean géré, TaskPool. Le bean géré fournit la liste d'objets Java traités par itération, puis affichés dans des lignes individuelles.

2. Configurez le bean géré référencé par la balise `bpe:list`.

Pour le composant List, ce bean géré doit être une instance de la classe `com.ibm.bpe.jsf.handler.BPCListHandler`.

L'exemple suivant illustre l'ajout d'un bean géré TaskPool au fichier de configuration.

```
<managed-bean>
<managed-bean-name>TaskPool</managed-bean-name>
<managed-bean-class>com.ibm.bpe.jsf.handler.BPCListHandler</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
<managed-property>
```

```

        <property-name>query</property-name>
        <value>#{TaskPoolQuery}</value>
    </managed-property>
    <managed-property>
        <property-name>type</property-name>
        <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
    </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>TaskPoolQuery</managed-bean-name>
<managed-bean-class>sample.TaskPoolQuery</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>type</property-name>
        <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
    </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>htmConnection</managed-bean-name>
<managed-bean-class>com.ibm.task.clientmodel.HTMConnection</managed-bean-class>
<managed-bean-scope>application</managed-bean-scope>
    <managed-property>
        <property-name>jndiName</property-name>
        <value>java:comp/env/ejb/LocalHumanTaskManagerEJB</value>
    </managed-property>
</managed-bean>

```

L'exemple indique que TaskPool possède deux propriétés configurables : query et type. La valeur de la propriété query fait référence à un autre bean géré, TaskPoolQuery. La valeur de la propriété type indique la classe de bean dont les propriétés s'affichent dans les colonnes de la liste affichée. L'instance de requête associée possède également un type de propriété. Si un type de propriété est indiqué, il doit être identique au type indiqué pour le gestionnaire de liste.

Vous pouvez ajouter n'importe quel type de logique de requête à l'application JSF à partir du moment où le résultat de la requête peut être représenté sous forme de liste de beans fortement typés. Par exemple, la requête TaskPoolQuery est implémentée à l'aide d'une liste d'objets com.ibm.task.clientmodel.bean.TaskInstanceBean.

3. Ajoutez le code personnalisé du bean géré figurant en référence dans le gestionnaire de liste.

L'exemple suivant illustre l'ajout de code personnalisé du bean géré TaskPool.

```

public class TaskPoolQuery implements Query {

    public List execute throws ClientException {

        // Rechercher dans le fichier faces-config le bean géré "htmConnection".
        //
        FacesContext ctx = FacesContext.getCurrentInstance();
        Application app = ctx.getApplication();
        ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
        htmConnection = (HTMConnection) htmVb.getValue(ctx);
        HumanTaskManagerService taskService =
            htmConnection.getHumanTaskManagerService();

        // Appel de la méthode de requête effective sur le service Human Task Manager.
        //
        // Ajouter à l'instruction de sélection les colonnes de base de données
        // de toutes les propriétés à afficher dans la liste
        //
        QueryResultSet queryResult = taskService.query(
            "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND, TASK.STATE, TASK.TYPE,"

```

```

+ "TASK.STARTER, TASK.OWNER, TASK.STARTED, TASK.ACTIVATED, TASK.DUE,
  TASK.EXPIRES, TASK.PRIORITY",
"TASK.KIND IN(101,102,105) AND TASK.STATE IN(2)
  AND WORK_ITEM.REASON IN (1)",
  (Chaîne)null,
  (Integer)null,
  (TimeZone)null);
List applicationObjects = transformToTaskList ( queryResult );
return applicationObjects ;
}

```

```

private List transformToTaskList(QueryResultSet result) {

ArrayList array = null;
int entries = result.size();
array = new ArrayList( entries );

// Transformation de chaque ligne de QueryResultSet en bean d'instance de tâche.
for (int i = 0; i < entries; i++) {
  result.next();
  array.add( new TaskInstanceBean( result, connection ));
}
return array ;
}
}

```

Le bean `TaskPoolQuery` interroge les propriétés des objets Java. Ce bean doit implémenter l'interface `com.ibm.bpc.clientcore.Query`. Quand il actualise son contenu, le gestionnaire de liste appelle la méthode `execute` de la requête. L'appel renvoie une liste d'objets Java. La méthode `getType` doit renvoyer le nom de classe des objets Java renvoyés.

Résultats

Votre application JSF contient à présent une page `JavaServer` affichant les propriétés de la liste d'objets demandée : état, type, propriétaire et émetteur des tâches d'instance disponibles, par exemple.

Concepts associés

Mode de traitement des listes

Chaque instance du composant List est associée à une instance de la classe `com.ibm.bpe.jsf.handler.BPCListHandler`.

Informations de fuseau horaire propres à l'utilisateur

Les composants JavaServer Faces (JSF) offrent un utilitaire de gestion des informations de fuseau horaire propre à l'utilisateur dans le composant List.

Traitement des erreurs dans le composant List

Lorsque vous utilisez le composant List pour afficher des listes dans votre application JSF, vous pouvez tirer parti des fonctions de traitement d'erreurs fournies par la classe `com.ibm.bpe.jsf.handler.BPCListHandler`.

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Référence associée

Composant List : définitions de balises

Le composant List de Business Process Choreographer Explorer affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades.

Mode de traitement des listes

Chaque instance du composant List est associée à une instance de la classe `com.ibm.bpe.jsf.handler.BPCListHandler`.

Le gestionnaire de listes effectue le suivi des éléments sélectionnés dans la liste associée et fournit un mécanisme de notification pour associer les entrées de liste aux pages de détails des différents types d'éléments. Le gestionnaire de listes est lié au composant List via l'attribut **model** contenu dans la balise `bpe:list`.

Le système de notification du gestionnaire de listes est mis en oeuvre via l'interface `com.ibm.bpe.jsf.handler.ItemListener`. Des implémentations de cette interface peuvent être enregistrées dans le fichier de configuration de votre application JSF (JavaServer Faces).

La notification est déclenchée en cas d'activation d'un lien dans la liste. Les liens de toutes les colonnes pour lesquelles l'attribut **action** est défini, s'affichent. La valeur de l'attribut **action** est soit une cible de navigation JSF, soit une méthode d'action JSF qui renvoie une cible de navigation JSF.

La classe `BPCListHandler` fournit également une méthode `refreshList`. Vous pouvez appliquer cette méthode à des liaisons de méthodes JSF afin de mettre en oeuvre un contrôle d'interface utilisateur visant à réexécuter la requête.

Mises en oeuvre de requêtes

Le gestionnaire de listes peut être utilisé pour afficher toutes sortes d'objets, ainsi que les propriétés de ces derniers. Le contenu de la liste affichée dépend de la liste des objets renvoyés par la mise en oeuvre de l'interface `com.ibm.bpc.clientcore.Query` configurée pour le gestionnaire de listes. Vous

pouvez définir la requête par voie de programme via la méthode `setQuery` de la classe `BPCListHandler`, ou la configurer dans les fichiers de configuration JSF de l'application.

L'exécution de requêtes peut concerner non seulement les API de Business Process Choreographer, mais également toute autre source d'informations accessible par le biais de votre application, telle qu'un système de gestion de contenus ou une base de données. La seule condition requise est que le résultat de la requête soit renvoyé sous forme d'une liste `java.util.List` contenant les objets de la méthode exécutée.

Le type des objets renvoyés doit garantir que les méthodes d'accès `get` appropriées sont disponibles pour toutes les propriétés affichées dans les colonnes de la liste faisant l'objet de la requête. Pour vous assurer que le type d'objet renvoyé correspond bien aux définitions de la liste, vous pouvez utiliser le nom de classe qualifié complet des objets renvoyés en tant que valeur de propriété du type concerné dans l'instance `BPCListHandler` définie par le fichier de configuration JSF. Ce nom peut être renvoyé dans l'appel `getType` de la mise en oeuvre de la requête. Lors de l'exécution, le gestionnaire de listes contrôle que les types d'objet sont bien conformes aux définitions.

Pour créer un mappage entre des messages d'erreur et des entrées spécifiques d'une liste, les objets renvoyés par la requête doivent mettre en oeuvre une méthode comportant la signature `public Object getID()`.

Convertisseurs et intitulés par défaut

Les éléments renvoyés par une requête doivent être des beans et leurs classes doivent correspondre à la classe spécifiée comme le type dans la définition de la classe `BPCListHandler` ou de l'interface `com.ibm.bpc.clientcore.Query`. De plus, le composant `List` vérifie si la classe d'éléments ou une superclasse implémente les méthodes suivantes :

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
    getConverter(String property);
```

Si ces méthodes sont définies pour les beans, le composant `List` utilise l'intitulé comme intitulé par défaut pour la liste et `SimpleConverter` comme convertisseur par défaut pour la propriété. Vous pouvez remplacer ces paramètres par les attributs `label` et `converterID` de la balise `bpe:list`. Pour plus d'informations sur l'interface `SimpleConverter` et `ColumnTag` class, reportez-vous à la documentation Java.

Tâches associées

Ajout du composant `List` à une application JSF

Le composant `List` de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Informations de fuseau horaire propres à l'utilisateur

Les composants JavaServer Faces (JSF) offrent un utilitaire de gestion des informations de fuseau horaire propre à l'utilisateur dans le composant `List`.

La classe `BPCListHandler` utilise l'interface `com.ibm.bpc.clientcore.util.User` pour obtenir des informations sur le fuseau horaire et l'environnement local de chaque utilisateur. Pour les besoins du composant `List` la mise en oeuvre de l'interface doit être configurée de sorte que `user` soit le nom du bean géré défini dans le fichier

fichier de configuration JSF (JavaServer Faces). Si cette entrée est absente du fichier de configuration, la valeur renvoyée est celle du fuseau horaire dans lequel WebSphere Process Server est exécuté.

L'interface `com.ibm.bpc.clientcore.util.User` est définie comme suit :

```
public interface User {  
  
    /**  
     * Environnement local utilisé par le client de l'utilisateur.  
     * @return Locale.  
     */  
    public Locale getLocale();  
    /**  
     * Fuseau horaire utilisé par le client de l'utilisateur.  
     * @return TimeZone.  
     */  
    public TimeZone getTimeZone();  
  
    /**  
     * Nom de l'utilisateur.  
     * @return nom de l'utilisateur.  
     */  
    public String getName();  
}
```

Tâches associées

Ajout du composant List à une application JSF

Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Traitement des erreurs dans le composant List

Lorsque vous utilisez le composant List pour afficher des listes dans votre application JSF, vous pouvez tirer parti des fonctions de traitement d'erreurs fournies par la classe `com.ibm.bpc.jsf.handler.BPCListHandler`.

Erreurs se produisant lors de l'exécution de requêtes ou de commandes

Si une erreur se produit lors de l'exécution d'une requête, la classe `BPCListHandler` fait une distinction entre les erreurs dues à des droits d'accès insuffisants et les autres exceptions. Pour intercepter les erreurs dues à des droits d'accès insuffisants, le paramètre `rootCause` de l'exception `ClientException` lancée par la méthode `execute` de la requête doit être une exception de type `com.ibm.bpc.api.EngineNotAuthorizedException` ou `com.ibm.task.api.NotAuthorizedException`. Le composant List affiche le message d'erreur à la place du résultat de la requête.

Si l'erreur n'est pas provoquée par des droits d'accès insuffisants, la classe `BPCListHandler` transmet l'objet exception à la mise en oeuvre d'interface `com.ibm.bpc.clientcore.util.ErrorBean` qui est définie par la clé `BPCError` dans le fichier de configuration de l'application JSF. Une fois l'exception définie, la cible de navigation de l'erreur est appelée.

Erreurs se produisant lors du traitement d'entités affichées dans une liste

La classe `BPCListHandler` met en oeuvre l'interface `com.ibm.bpc.jsf.handler.ErrorHandler`. Vous pouvez fournir des informations sur

ces erreurs via le paramètre de mappage de type `java.util.Map` inclus dans la méthode `setErrors`. Dans cette mappe, des identifiants sont associés à des clés et des exceptions sont associées à des valeurs. Les identifiants doivent obligatoirement être les valeurs renvoyées par la méthode `getID` de l'objet ayant provoqué l'erreur. Si la mappe est définie et qu'un ID correspond à l'une des entités de la liste, le gestionnaire de listes ajoute automatiquement à la liste une colonne contenant le message d'erreur.

Pour éviter que la liste ne contienne des messages d'erreur périmés, réinitialisez la mappe d'erreurs. La mappe est initialisée automatiquement dans les cas suivants :

- La classe `BPCListHandler` de la méthode `refreshList` est appelée.
- Une nouvelle requête est envoyée à la classe `BPCListHandler`.
- Le composant `CommandBar` est utilisé pour déclencher des actions concernant les entités contenues dans la liste. Le composant `CommandBar` utilise ce mécanisme comme méthode de traitement des erreurs.

Tâches associées

Ajout du composant `List` à une application JSF

Le composant `List` de `Business Process Choreographer Explorer` permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Composant List : définitions de balises

Le composant `List` de `Business Process Choreographer Explorer` affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades.

Le composant `List` comprend deux balises de composant JSF : `bpe:list` et `bpe:column`. La balise `bpe:column` est un sous-élément de la balise `bpe:list`.

Classe de composants

`com.ibm.bpe.jsf.component.ListComponent`

Syntaxe exemple

```
<bpe:list model="#{ProcessTemplateList}">
  rows="20"
  styleClass="list"
  headerStyleClass="listHeader"
  rowClasses="normal">

  <bpe:column name="name" action="processTemplateDetails"/>
  <bpe:column name="validFromTime"/>
  <bpe:column name="executionMode" label="Execution mode"/>
  <bpe:column name="state" converterID="my.state.converter"/>
  <bpe:column name="autoDelete"/>
  <bpe:column name="description"/>

</bpe:list>
```

Attributs de balise

Le corps de la balise `bpe:list` ne peut contenir que des balises `bpe:column`. Quand la table s'affiche, le composant `List` effectue une itération sur sa liste d'objets d'application et affiche toutes les colonnes de chaque objet.

Tableau 52. Attributs *bpe:list*

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de styles CSS pour l'affichage des boutons dans la zone de pied de page.
cellStyleClass	non	Classe de styles CSS pour l'affichage de cellules de tableau.
checkbox	non	Détermine si la case à cocher de sélection multiple est affichée. L'attribut possède la valeur true ou false. Si la valeur est définie sur true, la colonne de case à cocher est affichée.
headerStyleClass	non	Classe de styles CSS pour l'affichage de l'entête de tableau.
model	oui	Liaison de valeur d'un bean géré de la classe <code>com.ibm.bpe.jsf.handler.BPCListHandler</code> .
rows	non	Nombre de lignes affichées par page. Si le nombre d'éléments est supérieur au nombre de lignes, des boutons de pagination s'affichent à la fin du tableau. Les expressions de valeur ne sont pas prises en charge pour cet attribut.
rowClasses	non	Classe de styles CSS pour l'affichage des lignes du tableau.
selectAll	non	Si cet attribut est défini à true, tous les éléments de la liste sont sélectionnés par défaut.
styleClass	non	Classe de styles CSS pour l'affichage du tableau contenant les titres, les lignes et les boutons de pagination.

Tableau 53. Attributs *bpe:column*

Attribut	Obligatoire	Description
action	non	Si cet attribut est indiqué, un lien s'affiche dans cette colonne. Quand vous cliquez sur ce lien, cela provoque le déclenchement d'une méthode d'action JavaServer Faces ou de la cible de navigation Faces. Une méthode d'action JavaServer Faces possède la signature : <code>String method()</code> .
converterID	non	L'identificateur du convertisseur Faces utilisé pour convertir la valeur de la propriété. Si cet attribut n'est pas défini, l'identificateur du convertisseur Faces fourni par le modèle pour cette propriété est utilisé.
label	non	Expression littérale ou de liaison de valeur utilisée en tant qu'intitulé de l'en-tête de la colonne ou de la cellule de la ligne d'en-tête de table. Si cet attribut n'est pas défini, l'intitulé fourni par le modèle pour cette propriété est utilisé.

Tableau 53. Attributs `bpe:column` (suite)

Attribut	Obligatoire	Description
name	oui	Nom de la propriété qui est affichée dans cette colonne.

Tâches associées

Ajout du composant List à une application JSF

Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

Ajout du composant Details à une application JSF

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, de tâches élémentaires, d'instances de processus et de modèles de processus.

Procédure

1. Ajoutez le composant Details au fichier JavaServer Pages (JSP).

Ajoutez la balise `bpe:details` à la balise `<h:form>`. La balise `bpe:details` doit contenir un attribut de **modèle**. Vous pouvez ajouter des propriétés au composant Details à l'aide de la balise `bpe:property`.

L'exemple suivant illustre l'ajout d'un composant Details afin d'afficher quelques-unes des propriétés d'une instance de tâche.

```
<h:form>

    <bpe:details model="#{TaskInstanceDetails}">
        <bpe:property name="displayName" />
        <bpe:property name="owner" />
        <bpe:property name="kind" />
        <bpe:property name="state" />
        <bpe:property name="escalated" />
        <bpe:property name="suspended" />
        <bpe:property name="originator" />
        <bpe:property name="activationTime" />
        <bpe:property name="expirationTime" />
    </bpe:details>

</h:form>
```

L'attribut de **modèle** fait référence à un bean géré, `TaskInstanceDetails`. Le bean fournit les propriétés de l'objet Java.

2. Configurez le bean géré référencé par la balise `bpe:details`.

Pour le composant Details, ce bean géré doit être une instance de la classe `com.ibm.bpe.jsf.handler.BPCDetailsHandler`. Cette classe de gestionnaire encapsule un objet Java et expose ses propriétés publiques au composant Details.

L'exemple suivant illustre l'ajout d'un bean géré `TaskInstanceDetails` au fichier de configuration.

```
<managed-bean>
    <managed-bean-name>TaskInstanceDetails</managed-bean-name>
    <managed-bean-class>com.ibm.bpe.jsf.handler.BPCDetailsHandler</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>type</property-name>
        <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
    </managed-property>
</managed-bean>
```

L'exemple montre que le bean `TaskInstanceDetails` bean possède une propriété type configurable. La valeur de la propriété type indique la classe de bean (`com.ibm.task.clientmodel.bean.TaskInstanceBean`) dont les propriétés s'affichent dans les lignes de détail générées. La classe de bean peut correspondre à n'importe quelle classe JavaBeans. Si le bean fournit des intitulés de conversion et de propriété par défaut, le convertisseur et l'intitulé sont utilisés pour le rendu de la même manière que le composant `List`.

Résultats

Votre application JSF contient à présent une page JavaServer affichant les détails de l'objet spécifié (une instance de tâche, par exemple).

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Référence associée

Composant Details : définitions de balises

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus.

Composant Details : définitions de balises

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus.

Le composant Details comprend deux balises de composant JSF : `bpe:details` et `bpe:property`. La balise `bpe:property` est un sous-élément de la balise `bpe:details`.

Classe de composants

`com.ibm.bpe.jsf.component.DetailsComponent`

Syntaxe exemple

```
<bpe:details model="#{MyActivityDetails}">
  <bpe:property name="name"/>
  <bpe:property name="owner"/>
  <bpe:property name="activated"/>
</bpe:details>

<bpe:details model="#{MyActivityDetails}" style="style" styleClass="cssStyle">
  style="style"
  styleClass="cssStyle"
</bpe:details>
```

Attributs de balise

Les balises `bpe:property` permettent d'indiquer à la fois le sous-ensemble d'attributs affichés et l'ordre d'affichage de ces attributs. Si la balise `details` ne contient pas de balise d'attribut, elle affiche tous les attributs disponibles de l'objet modèle.

Tableau 54. Attributs *bpe:details*

Attribut	Obligatoire	Description
columnClasses	non	Liste des classes de style CSS séparées par des virgules et utilisées pour l’affichage de colonnes.
id	non	ID du composant JavaServer Faces.
model	oui	Liaison de valeur d’un bean géré de la classe <code>com.ibm.bpe.jsf.handler.BPCDetailsHandler</code> .
rowClasses	non	Liste des classes de style CSS séparées par des virgules et utilisées pour l’affichage des lignes.
styleClass	non	Classe CSS utilisée pour l’affichage de l’élément HTML.

Tableau 55. Attributs *bpe:property*

Attribut	Obligatoire	Description
converterID	non	Identificateur utilisé pour l’enregistrement du convertisseur dans le fichier de configuration JavaServer Faces (JSF).
label	non	Libellé de la propriété. Si cet attribut n’est pas défini, un libellé par défaut est fourni par la classe de modèle client.
name	oui	Nom de la propriété à afficher. Ce nom doit correspondre à une propriété nommée définie dans la classe de modèle client correspondant.

Tâches associées

Ajout du composant Details à une application JSF

Le composant Details de Business Process Choreographer Explorer permet d’afficher les propriétés de tâches, de tâches élémentaires, d’instances de processus et de modèles de processus.

Ajout du composant CommandBar à une application JSF

Utilisez le composant CommandBar de Business Process Choreographer Explorer pour permettre l’affichage d’une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d’un objet ou des objets sélectionnés d’une liste.

A propos de cette tâche

Quand l’utilisateur clique sur un bouton dans l’interface, la commande correspondante est exécutée sur les objets sélectionnés. Vous pouvez ajouter et étendre le composant CommandBar dans votre application JSF (JavaServer Faces).

Procédure

1. Ajoutez le composant CommandBar au fichier JavaServer Pages (JSP).
Ajoutez la balise `bpe:commandbar` à la balise `<h:form>`. La balise `bpe:commandbar` doit contenir un attribut de modèle.

L'exemple suivant illustre l'ajout d'un composant CommandBar, ce dernier fournissant des commandes de régénération et de réclamation pour une liste d'instances de tâches.

```
<h:form>

    <bpe:commandbar model="#{TaskInstanceList}">
        <bpe:command commandID="Refresh" >
            action="#{TaskInstanceList.refreshList}"
            label="Refresh"/>

        <bpe:command commandID="MyClaimCommand" >
            label="Claim" >
                commandClass="<customcode>"/>
    </bpe:commandbar>

</h:form>
```

L'attribut **model** fait référence à un bean géré. Ce bean doit implémenter l'interface `ItemProvider` et fournir les objets Java sélectionnés. Le composant `CommandBar` est généralement utilisé soit avec le composant `List`, soit avec le composant `Details` dans le même fichier JSP. En général, le modèle spécifié dans la balise correspond à celui qui est indiqué dans le composant `List` ou `Details` sur la même page. Pour un composant `List`, la commande agit donc sur les éléments sélectionnés dans la liste.

Dans cet exemple, l'attribut **model** fait référence au bean géré `TaskInstanceList`. Ce bean fournit les objets sélectionnés dans la liste des instances de tâches. Il doit implémenter l'interface `ItemProvider`. Cette interface est implémentée par les classes `BPCListHandler` et `BPCDetailsHandler`.

2. Facultatif : Configurez le bean géré référencé par la balise `bpe:commandbar`.

Si l'attribut **model** de `CommandBar` fait référence à un bean géré qui est déjà configuré, par exemple dans le cas d'une liste ou d'un gestionnaire de détails, aucune configuration complémentaire n'est requise. Si vous n'utilisez ni la classe `BPCListHandler`, ni la classe `BPCDetailsHandler` pour le modèle, vous devez faire référence à un autre objet comportant une classe qui implémente l'interface `ItemProvider`.

3. Ajoutez le code implémentant les commandes personnalisés vers l'application JSF.

Le fragment de code ci-dessous montre comment écrire une classe de commandes qui implémente l'interface `Command`. Cette classe de commandes (`MyClaimCommand`) est désignée par la balise `bpe:command` dans le fichier JSP.

```
public class MyClaimCommand implements Command {

    public String execute(List selectedObjects) throws ClientException {
        if( selectedObjects != null && selectedObjects.size() > 0 ) {
            try {
                // Déterminer HumanTaskManagerService à partir d'un bean HTMLConnection.
                // Configurer le bean dans le fichier faces-config.xml pour faciliter
                // l'accès à l'application JSF.
                FacesContext ctx = FacesContext.getCurrentInstance();
                ValueBinding vb =
                    ctx.getApplication().createValueBinding("#{htmlConnection}");
                HTMLConnection htmConnection = (HTMLConnection) htmVB.getValue(ctx);
                HumanTaskManagerService htm =
                    htmConnection.getHumanTaskManagerService();

                Iterator iter = selectedObjects.iterator() ;
                while( iter.hasNext() ) {
                    try {
                        TaskInstanceBean task = (TaskInstanceBean) iter.next() ;
                        TKIID tiid = task.getID() ;
```

```

        htm.claim( tiid ) ;
        task.setState( new Integer( TaskInstanceBean.STATE_CLAIMED ) ) ;
    }
    catch( Exception e ) {
        ; // Erreur lors de l'itération ou réclamation d'une instance
          de tâche.
          // Ignorer pour mieux comprendre l'exemple.
    }
}
}
}
catch( Exception e ) {
    ; // Erreur de configuration ou de communication.
    // Ignorer pour mieux comprendre l'exemple.
}
}
return null;
}

// Implémentations par défaut
public boolean isMultiSelectEnabled() { return false; }
public boolean[] isApplicable(List itemsOnList) {return null; }
public void setContext(Object targetModel) {; // Non utilisé ici }
}

```

La commande est traitée ainsi :

- a. Une commande est appelée quand un utilisateur clique sur le bouton correspondant dans la barre de commandes. Le composant `CommandBar` extrait les éléments sélectionnés depuis le fournisseur d'éléments indiqué dans l'attribut **model** et transmet la liste d'objets sélectionnés à la méthode `execute` de l'instance `commandClass`.
- b. Facultatif : L'attribut **commandClass** fait référence à une implémentation de commande personnalisée mettant en oeuvre l'interface `Command`. Cela signifie que la commande doit implémenter la méthode `public String execute(List selectedObjects) throws ClientException`. Elle renvoie le résultat permettant de déterminer la prochaine règle de navigation de l'application JSF.
- c. Facultatif : Après l'exécution de la commande, le composant `CommandBar` évalue l'attribut **action**. L'attribut **action** peut être une chaîne statique ou une liaison de méthode vers une méthode d'action ayant la signature `public String Method()`. L'attribut **action** permet de remplacer le résultat d'une classe de commandes ou d'indiquer explicitement un résultat pour les règles de navigation. L'attribut **action** n'est pas traité si la commande génère une exception autre que `ErrorsInCommandException`.
- d. Si aucune classe de commandes n'est spécifiée pour l'attribut **commandClass**, l'action est immédiatement appelée. Par exemple, pour la commande `refresh` utilisée dans l'exemple, c'est l'expression de valeur JSF `#{TaskInstanceList.refreshList}` qui est appelée au lieu d'une commande.

Résultats

Votre application JSF contient à présent une page JSP implémentant une barre de commandes personnalisée.

Concepts associés

Mode de traitement des commandes

Utilisez le composant `CommandBar` pour intégrer des boutons d'action à votre application. Le composant crée les boutons qui correspondent aux actions dans l'interface utilisateur et traite les événements générés lors du clic sur un bouton.

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants `JavaServer Faces (JSF)`. Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Référence associée

Composant `CommandBar` : définitions de balises

Le composant `CommandBar` de `Business Process Choreographer Explorer` permet d'afficher une barre comportant des boutons de commande. Ces boutons agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste.

Mode de traitement des commandes

Utilisez le composant `CommandBar` pour intégrer des boutons d'action à votre application. Le composant crée les boutons qui correspondent aux actions dans l'interface utilisateur et traite les événements générés lors du clic sur un bouton.

Ces boutons déclenchent des fonctions agissant sur les objets renvoyés par une interface `com.ibm.bpe.jsf.handler.ItemProvider` tels que la classe `BPCListHandler`, ou encore la classe `BPCDetailsHandler`. Le composant `CommandBar` utilise le fournisseur d'éléments défini par la valeur de l'attribut **model** contenu dans la balise `bpe:commandbar`.

Lorsqu'un clic est effectué sur un bouton situé dans la section dédiée à la barre de commandes dans l'interface utilisateur de l'application, l'événement associé est traité comme suit par le composant `CommandBar`.

1. Le composant `CommandBar` identifie la mise en oeuvre de l'interface `com.ibm.bpc.clientcore.Command` spécifiée pour le bouton ayant généré l'événement.
2. Si le modèle associé au composant `CommandBar` met en oeuvre l'interface `com.ibm.bpe.jsf.handler.ErrorHandler`, la méthode `clearErrorMap` est appelée pour effacer les messages d'erreur consécutifs aux événements antérieurs.
3. La méthode `getSelectedItems` de l'interface `ItemProvider` est appelée. La liste des entités renvoyées est transmise à la méthode `execute` de la commande, puis cette dernière est appelée.
4. Le composant `CommandBar` détermine la cible de navigation JSF (`JavaServer Faces`). Si aucun attribut **action** n'est spécifié dans la balise `bpe:commandbar`, la cible de navigation est spécifiée par la valeur renvoyée de la méthode `execute`. Si l'attribut **action** est défini sur une liaison de méthode JSF, la chaîne renvoyée par la méthode est interprétée comme étant la cible de navigation. L'attribut **action** peut également spécifier une cible de navigation explicite.

Tâches associées

Ajout du composant CommandBar à une application JSF

Utilisez le composant CommandBar de Business Process Choreographer Explorer pour permettre l'affichage d'une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d'un objet ou des objets sélectionnés d'une liste.

Composant CommandBar : définitions de balises

Le composant CommandBar de Business Process Choreographer Explorer permet d'afficher une barre comportant des boutons de commande. Ces boutons agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste.

Le composant CommandBar comprend deux balises de composant JSF :

bpe:commandbar et bpe:command. La balise bpe:command est un sous-élément de la balise bpe:commandbar.

Classe de composants

com.ibm.bpe.jsf.component.CommandBarComponent

Syntaxe exemple

```
<bpe:commandbar model="#{TaskInstanceList}">

    <bpe:command
        commandID="Work on"
        label="Work on..."
        commandClass="com.ibm.bpc.explorer.command.WorkOnTaskCommand"
        context="#{TaskInstanceDetailsBean}" />

    <bpe:command
        commandID="Cancel"
        label="Cancel"
        commandClass="com.ibm.task.clientmodel.command.CancelClaimTaskCommand"
        context="#{TaskInstanceList}" />

</bpe:commandbar>
```

Attributs de balise

Tableau 56. Attributs bpe:commandbar

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de styles CSS pour l'affichage des boutons de la barre de commandes.
id	non	ID du composant JavaServer Faces.
model	oui	Expression de liaison de valeur vers un bean géré implémentant une interface ItemProvider. Ce bean géré est généralement la classe com.ibm.bpe.jsf.handler.BPCListHandler ou la classe com.ibm.bpe.jsf.handler.BPCDetailsHandler utilisée par le composant List ou Details dans le même fichier JavaServer Pages (JSP) que le composant CommandBar.
styleClass	non	Classe de styles CSS pour l'affichage de la barre de commandes.

Tableau 57. Attributs *bpe:command*

Attribut	Obligatoire	Description
action	non	Méthode d'action JavaServer Faces ou cible de navigation Faces qui est déclenchée par le bouton de commande. La cible de navigation qui est renvoyée par l'action écrase toutes les autres règles de navigation. L'action est appelée lorsqu'une exception n'est pas émise ou lorsqu'une exception <code>ErrorsInCommandEvent</code> est émise par la commande.
commandClass	non	Le nom de la classe de commande. Une instance de la classe est créée par le composant <code>CommandBar</code> , puis elle est exécutée lorsque le bouton de commande est sélectionné.
commandID	oui	ID de la commande.
context	non	Un objet qui fournit du contexte pour les commandes qui sont spécifiées à l'aide de l'attribut <code>commandClass</code> . L'objet de contexte est extrait lors du premier accès à la barre de commandes.
immediate	non	Indique le moment du déclenchement de la commande. Si la valeur de cet attribut est définie sur <code>true</code> , la commande est déclenchée avant le traitement de l'entrée de la page. La valeur par défaut est <code>false</code> .
label	oui	Libellé du bouton affiché dans la barre de commandes.
rendu	non	Détermine si un bouton a été rendu. La valeur de l'attribut peut être une valeur booléenne ou une expression de valeur.
styleClass	non	Classe CSS utilisée pour l'affichage du bouton. Ce style se substitue au style de bouton défini pour la barre de commandes.

Tâches associées

Ajout du composant `CommandBar` à une application JSF

Utilisez le composant `CommandBar` de `Business Process Choreographer Explorer` pour permettre l'affichage d'une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d'un objet ou des objets sélectionnés d'une liste.

Ajout du composant Message à une application JSF

Le composant `Message` de l'explorateur du Chorégraphe de processus métier permet d'afficher des objets de données et des types de primitive dans une application JavaServer Faces (JSF).

A propos de cette tâche

Si le message est de type primitif, un libellé et un champ de saisie sont affichés. Si le type de message est un objet de données, le composant traverse l'objet et affiche les éléments à l'intérieur de l'objet.

Procédure

1. Ajoutez le composant Message au fichier JavaServer Pages (JSP).

Ajoutez la balise `bpe:form` à la balise `<h:form>`. La balise `bpe:form` doit contenir un attribut `model`.

L'exemple suivant illustre l'ajout d'un composant Message.

```
<h:form>

    <h:outputText value="Input Message" />
    <bpe:form model="#{MyHandler.inputMessage}" readOnly="true" />

    <h:outputText value="Output Message" />
    <bpe:form model="#{MyHandler.outputMessage}" />

</h:form>
```

L'attribut **model** du composant Message fait référence à un objet `com.ibm.bpc.clientcore.MessageWrapper`. Cet objet encapsuleur enveloppe un objet SDO (Service Data Object) ou une primitive de type Java, par exemple de type `int` ou `boolean`. Dans l'exemple, le message est fourni par une propriété du bean géré `MyHandler`.

2. Configurez le bean géré référencé par la balise `bpe:form`.

L'exemple suivant illustre l'ajout d'un bean géré `MyHandler` au fichier de configuration.

```
<managed-bean>
  <managed-bean-name>MyHandler</managed-bean-name>
  <managed-bean-class>com.ibm.bpc.sample.jsf.MyHandler</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>

  <managed-property>
    <property-name>type</property-name>
    <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
  </managed-property>

</managed-bean>
```

3. Ajoutez du code personnalisé à l'application JSF.

L'exemple suivant illustre l'implémentation de messages d'entrée et de sortie.

```
public class MyHandler implements ItemListener {

    private TaskInstanceBean taskBean;
    private MessageWrapper inputMessage, outputMessage

    /* Listener method, e.g. when a task instance was selected in a list handler.
     * Ensure that the handler is registered in the faces-config.xml or manually.
     */
    public void itemChanged(Object item) {
        if( item instanceof TaskInstanceBean ) {
            taskBean = (TaskInstanceBean) item ;
        }
    }

    /* Get the input message wrapper
     */
    public MessageWrapper getInputMessage() {
        try{
            inputMessage = taskBean.getInputMessageWrapper() ;
        }
        catch( Exception e ) {
            ; //...ignore errors for simplicity
        }
        return inputMessage;
    }
}
```

```

/* Get the output message wrapper
*/
public MessageWrapper getOutputMessage() {
    Extraction du message du bean. Si aucun message n'existe, créez-en
    // un si la tâche a été réclamée par l'utilisateur. Assurez-vous que
    // seuls les propriétaires (potentiels ou non) peuvent manipuler le message
    // de sortie.
    try{
        outputMessage = taskBean.getOutputMessageWrapper();
        if( outputMessage == null
            && taskBean.getState() == TaskInstanceBean.STATE_CLAIMED ) {
            HumanTaskManagerService htm = getHumanTaskManagerService();
            outputMessage = new MessageWrapperImpl();
            outputMessage.setMessage(
                htm.createOutputMessage( taskBean.getID() ).getObject()
            );
        }
    }
    catch( Exception e ) {
        ; //...ignore errors for simplicity
    }
    return outputMessage
}
}

```

Le bean géré MyHandler implémente l'interface `com.ibm.jsf.handler.ItemListener` pour permettre son enregistrement en tant qu'écouteur d'éléments du gestionnaire de listes. Quand l'utilisateur clique sur un élément dans la liste, le bean MyHandler est informé sur l'élément sélectionné via la méthode `itemChanged(Object item)`. Le gestionnaire contrôle le type d'élément, puis stocke une référence à l'objet `TaskInstanceBean` associé. Pour utiliser cette interface, ajoutez une entrée dans la liste `itemListener` du gestionnaire de listes approprié, qui se trouve dans le fichier `faces-config.xml`.

Le bean MyHandler fournit les méthodes `getInputMessage` et `getOutputMessage`. Ces deux méthodes retournent un objet `MessageWrapper`. Les méthodes délèguent les appels du bean d'instance de tâche référencé. Si l'instance de tâche renvoie la valeur null, par exemple parce qu'un message n'est pas défini, le gestionnaire crée et stocke un nouveau message vide. Le composant Message affiche les messages fournis par le bean MyHandler.

Résultats

Votre application JSF contient à présent une page JSP permettant d'afficher des objets de données et des types primitifs.

Tâches associées

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches utilisateur à des applications Web.

Référence associée

Composant Message : définitions de balises

Le composant Message de Business Process Choreographer Explorer affiche des objets `commonj.sdo.DataObject` et des types de primitive, tels que des entiers et des chaînes, dans une application JavaServer Faces (JSF).

Composant Message : définitions de balises

Le composant Message de Business Process Choreographer Explorer affiche des objets `commonj.sdo.DataObject` et des types de primitive, tels que des entiers et des chaînes, dans une application JavaServer Faces (JSF).

Le composant Message comprend la balise de composant JSF : `bpe:form`.

Classe de composants

`com.ibm.bpe.jsf.component.MessageComponent`

Syntaxe exemple

```
<bpe:form model="#{TaskInstanceDetailsBean.inputMessageWrapper}"
  simplification="true" readOnly="true"
  styleClass4table="messageData"
  styleClass4output="messageDataOutput">
</bpe:form>
```

Attributs de balise

Tableau 58. Attributs `bpe:form`

Attribut	Obligatoire	Description
<code>id</code>	non	ID du composant JavaServer Faces.
<code>model</code>	oui	Expression de liaison de valeur qui fait référence à un objet <code>commonj.sdo.DataObject</code> ou à un objet <code>com.ibm.bpc.clientcore.MessageWrapper</code> .
<code>readOnly</code>	non	Si cet attribut est réglé sur <code>true</code> , un formulaire s'affiche en lecture seule. Par défaut, cet attribut est réglé sur <code>false</code> .
<code>simplification</code>	non	Si cet attribut est réglé sur <code>true</code> , les propriétés contenant des types simples et ayant une cardinalité de 0 ou de 1 sont affichées. Par défaut, cet attribut est défini sur <code>true</code> .
<code>style4validinput</code>	non	Style CSS (feuille de styles en cascade) pour l'affichage de valeur d'entrée valide.
<code>style4invalidinput</code>	non	Style CSS pour l'affichage de valeur d'entrée incorrecte.
<code>styleClass4invalidInput</code>	non	Nom de classe de style CSS pour l'affichage de valeur d'entrée incorrecte.

Tableau 58. Attributs bpe:form (suite)

Attribut	Obligatoire	Description
styleClass4output	non	Nom de classe de styles CSS pour l'affichage d'éléments sortants.
styleClass4table	non	Nom de classe du style de tableau CSS pour l'affichage des tableaux affichés par le composant de message.
styleClass4validInput	non	Nom de classe de style CSS pour l'affichage de valeur d'entrée correcte.

Tâches associées

Ajout du composant Message à une application JSF

Le composant Message de l'explorateur du Chorégraphe de processus métier permet d'afficher des objets de données et des types de primitive dans une application JavaServer Faces (JSF).

Développement des pages JSP pour les messages de tâche et de processus

Business Process Choreographer Explorer fournit des formulaires d'entrée et de sortie par défaut pour afficher et saisir les données métier. Vous pouvez utiliser des pages JSP pour créer des formulaires d'entrée et de sortie définis par l'utilisateur.

A propos de cette tâche

Pour inclure des pages JSP (JavaServer Pages) définies par l'utilisateur dans le client Web, vous devez les indiquer lorsque vous modélisez une tâche utilisateur dans WebSphere Integration Developer. Par exemple, vous pouvez fournir des pages JSP pour une tâche spécifique et pour les messages d'entrée et de sortie associés, ainsi que pour un rôle utilisateur spécifique ou pour tous les rôles utilisateur. Lors de l'exécution, les pages JSP définies par l'utilisateur sont incluses dans l'interface utilisateur pour afficher les données de sortie et collecter les données d'entrée.

Les formulaires personnalisés ne sont pas des pages Web autonomes ; il s'agit de fragments de code HTML que Business Process Choreographer Explorer intègre dans un formulaire HTML (par exemple, les fragments pour tous les libellés et les zones d'entrée d'un message).

Lorsqu'un utilisateur clique sur un bouton de la page contenant les formulaires personnalisés, les données d'entrée sont soumises et validées dans Business Process Choreographer Explorer. La validation dépend du type des propriétés fournies et des paramètres locaux utilisés dans le navigateur. Si les données d'entrée ne peuvent pas être validées, la même page s'affiche de nouveau et les informations relatives aux erreurs de validation sont fournies dans l'attribut de demande `messageValidationErrors`. Les informations sont fournies sous forme d'un plan qui mappe l'expression XPath (XML Path Expression) des propriétés non valides avec les exceptions de validation qui ont eu lieu.

Pour ajouter des formulaires personnalisés à Business Process Choreographer Explorer, exécutez les opérations suivantes à l'aide de WebSphere Integration Developer :

Procédure

1. Créez les formulaires personnalisés.

Les pages JSP définies par l'utilisateur pour les formulaires d'entrée et de sortie utilisés dans l'interface Web doivent accéder aux données de messages. Utilisez les fragments Java d'un JSP ou le langage d'exécution JSP pour accéder aux données du message. Les données contenues dans les formulaires sont accessibles via le contexte de requête.

2. Affectez les pages JSP à une tâche.

Ouvrez la tâche utilisateur dans l'éditeur de tâches utilisateur. Dans les paramètres client, indiquez l'emplacement des pages JSP définies par l'utilisateur et le rôle auquel s'applique le formulaire personnalisé (par exemple, administrateur). Les paramètres client de l'explorateur du Chorégraphe de processus métier sont stockés dans le modèle de tâche. Lors de l'exécution, ces paramètres sont extraits avec le modèle de tâche.

3. Compressez les pages JSP définies par l'utilisateur dans une archive Web (fichier WAR).

Vous pouvez inclure le fichier WAR dans le fichier EAR (Enterprise Archive) avec le module contenant les tâches ou déployer le fichier WAR séparément. Si les JSP sont déployés séparément, faites en sorte qu'ils soient disponibles sur le serveur où est déployé Business Process Choreographer Explorer ou le client défini par l'utilisateur.

Si vous utilisez des JSP personnalisés pour les messages de processus et de tâche, vous devez mapper les modules web qui sont utilisés pour déployer les JSP avec les mêmes serveurs que ceux avec lesquels est mappé le client JSF personnalisé.

Résultats

Les formulaires personnalisés s'affichent dans Business Process Choreographer Explorer lors de l'exécution.

Concepts associés

Fragments JSP définis par l'utilisateur

Les fragments JSP (JavaServer Pages) définis par l'utilisateur sont intégrés à une balise de formulaire HTML. Lors de l'exécution, Business Process Choreographer Explorer inclut ces fragments dans la page affichée.

Tâches associées

Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Fragments JSP définis par l'utilisateur

Les fragments JSP (JavaServer Pages) définis par l'utilisateur sont intégrés à une balise de formulaire HTML. Lors de l'exécution, Business Process Choreographer Explorer inclut ces fragments dans la page affichée.

Le fragment JSP défini par l'utilisateur du message d'entrée est intégré avant le fragment JSP du message de sortie.

```

<html....>
...
<form...>
    Message JSP d'entrée (affichage du message d'entrée de la tâche)

    Message JSP de sortie (affichage du message de sortie de la tâche)

</form>
...
</html>

```

Les fragments JSP définis par l'utilisateur étant intégrés à une balise de formulaire HTML, vous pouvez ajouter des éléments d'entrée. Le nom de l'élément d'entrée doit correspondre à l'expression XPath (XML Path Language) de l'élément de données. Il est important de faire précéder de la valeur de préfixe fournie le nom de l'élément d'entrée :

```

<input id="address"
      type="text"
      name="{prefix}/selectPromotionalGiftResponse/address"
      value="{messageMap['/selectPromotionalGiftResponse/address']}"
      size="60"
      align="left" />

```

La valeur de préfixe est fournie sous forme d'attribut de demande. L'attribut garantit l'unicité du nom d'entrée dans le formulaire d'inclusion. Le préfixe est généré par Business Process Choreographer Explorer et ne doit pas être modifié :

```
String prefix = (String)request.getAttribute("prefix");
```

L'élément de préfixe est défini uniquement si le message peut être modifié dans le contexte spécifié. Les données de sortie peuvent s'afficher de différentes façons selon l'état de la tâche utilisateur. Par exemple, si l'état de la tâche est Réclamé, les données de sortie peuvent être modifiées. Toutefois, si l'état de la tâche est Terminé, les données peuvent uniquement être affichées. Dans votre fragment JSP, vous pouvez vérifier si l'élément de préfixe existe et afficher le message en conséquence. L'instruction JSTL suivante montre comment vérifier si l'élément de préfixe est défini :

```

...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<c:choose>
  <c:when test="{not empty prefix}">
    <!--Read/write mode-->
  </c:when>
  <c:otherwise>
    <!--Read-only mode-->
  </c:otherwise>
</c:choose>

```

Tâches associées

Développement des pages JSP pour les messages de tâche et de processus Business Process Choreographer Explorer fournit des formulaires d'entrée et de sortie par défaut pour afficher et saisir les données métier. Vous pouvez utiliser des pages JSP pour créer des formulaires d'entrée et de sortie définis par l'utilisateur.

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

A propos de cette tâche

Vous pouvez créer des modules d'extension pour des événements liés à des API de tâche utilisateur et à des notifications d'escalade. Vous pouvez également créer un plug-in qui traite les résultats renvoyés par la résolution des utilisateurs. Vous pouvez par exemple, lors de pics périodes, ajouter des utilisateurs à la liste de résultats afin de rééquilibrer la charge de travail.

avant de pouvoir utiliser le module d'extension, vous devez les installer et les enregistrer. Vous pouvez enregistrer le module d'extension pour permettre le post-traitement des résultats des requêtes de personnel avec l'application TaskContainer. Dans ce cas, le module d'extension est disponible pour toutes les tâches.

Tâches associées

Développement d'applications client pour les tâches et processus métier
Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

Création de gestionnaires d'événements d'API

Un événement d'API se produit lorsqu'une méthode d'API manipule une tâche utilisateur. Utilisez l'interface SPI du plug-in de gestionnaire d'événements d'API pour créer des plug-in permettant de gérer les événements de tâche envoyés par l'API ou par les événements internes ayant des événements API équivalents.

Création de gestionnaire d'événements de notification

Les événements de notification surviennent lors de l'escalade de tâches utilisateur. Business Process Choreographer fournit des fonctionnalités permettant la gestion des escalades, telles que la création d'éléments de travail d'escalade ou l'envoi de messages électroniques. Vous pouvez créer des gestionnaires d'événements de notification pour personnaliser le mode de traitement des escalades.

Installation des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification

Pour pouvoir utiliser un module d'extension de gestionnaire d'événements d'API ou de notification, vous devez l'installer de sorte qu'il soit accessible au conteneur de tâches.

Enregistrement des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification avec des modèles de tâche et des tâches
Vous pouvez enregistrer les modules d'extension pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches et des modèles de tâche à différentes occasions : lors de la création d'une tâche ad-hoc, de la mise à jour d'une tâche existante, de la création d'un modèle de tâche ou de la définition d'un modèle de tâche.

Création, installation et exécution de plug-ins en vue du post-traitement des résultats des requêtes sur les utilisateurs

La résolution d'utilisateurs renvoie une liste des utilisateurs auxquels un rôle spécifique est affecté, par exemple, le propriétaire potentiel d'une tâche. Vous pouvez créer un plug-in pour modifier les résultats des requêtes d'utilisateurs renvoyés par la résolution des utilisateurs. Par exemple, pour améliorer l'équilibrage de charge, vous pourriez avoir un plug-in qui supprime les utilisateurs du résultat de la requête s'ils ont déjà une charge de travail élevée.

Création de gestionnaires d'événements d'API

Un événement d'API se produit lorsqu'une méthode d'API manipule une tâche utilisateur. Utilisez l'interface SPI du plug-in de gestionnaire d'événements d'API pour créer des plug-in permettant de gérer les événements de tâche envoyés par l'API ou par les événements internes ayant des événements API équivalents.

A propos de cette tâche

Exécutez les étapes suivantes pour créer un gestionnaire d'événements d'API

Procédure

1. Rédigez une classe qui implémente l'interface `APIEventHandlerPlugin3` ou étend la classe d'implémentation `APIEventHandler`. Cette classe peut appeler les méthodes d'autres classes.
 - Si vous utilisez l'interface `APIEventHandlerPlugin3`, vous devez implémenter toutes les méthodes de l'interface `APIEventHandlerPlugin3` et de l'interface `APIEventHandlerPlugin`.
 - Si vous étendez la classe d'implémentation `APIEventHandler`, remplacez les méthodes selon vos besoins.

Cette classe s'exécute dans le contexte d'une application d'entreprise EJB J2EE (Enterprise Java 2 Enterprise Edition). Assurez-vous que cette classe et ses classes auxiliaires suivent la spécification EJB.

Remarque : Pour appeler l'interface `HumanTaskManagerService` à partir de cette classe, n'appellez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action peut entraîner une incohérence des données de tâche dans la base de données.

2. Assemblez la classe du plug-in et ses classes auxiliaires dans un fichier JAR. Pour rendre le fichier JAR disponible, vous pouvez procéder de l'une des manières suivantes :
 - En tant que fichier JAR d'utilitaire dans le fichier EAR de l'application.
 - En tant que bibliothèque partagée installée avec le fichier EAR de l'application.
 - En tant que bibliothèque partagée installée avec l'application `TaskContainer`. Dans ce cas, le module d'extension est disponible pour toutes les tâches.
3. Créez un fichier de configuration de fournisseur de services pour le plug-in dans le répertoire `META-INF/services/` du fichier JAR.

Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le plug-in. Ce fichier est conforme à la spécification de l'interface du fournisseur de services Java 2.

- a. Créez un fichier portant le nom `com.ibm.task.spi.nom_module_extensionAPIEventHandlerPlugin`, où `nom_module_extension` est le nom du plug-in.

Par exemple, si votre plug-in s'appelle `Customer` et qu'il implémente l'interface `com.ibm.task.spi.APIEventHandlerPlugin3`, le nom du fichier de configuration est `com.ibm.task.spi.CustomerAPIEventHandlerPlugin`.

- b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire (c'est-à-dire commençant par le signe `#`) ni une ligne vide, doit spécifier le nom qualifié complet de la classe de module d'extension créée à l'étape 1.

Par exemple, si la classe de votre plug-in est `MyAPIEventHandler` et se trouve dans le module `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante :

```
com.customer.plugins.MyAPIEventHandler.
```

Résultats

Vous avez un fichier JAR installable qui contient un plug-in gérant les événements d'API et un fichier de configuration du fournisseur de services pouvant être utilisé pour charger le plug-in.

Remarques : Vous ne disposez que d'une propriété `eventHandlerName` pour enregistrer à la fois les gestionnaires d'événements d'API et les gestionnaires d'événements de notification. Pour utiliser à la fois un gestionnaire d'événement

d'API If et un gestionnaire d'événement de notification, il est nécessaire que les implémentations des plug-ins portent le même nom (Customer comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux plug-ins à l'aide d'une seule classe ou de classes distinctes. Dans les deux cas, vous devez créer deux fichiers dans le répertoire META-INF/services/ de votre fichier JAR (par exemple, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin et com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

Regroupez l'implémentation du plug-in et les classes auxiliaires dans un seul fichier JAR.

Pour rendre effective une modification de l'implémentation, remplacez le fichier JAR contenu dans la bibliothèque partagée, déployez à nouveau le fichier EAR associé et redémarrez le serveur.

Que faire ensuite

Vous devez maintenant installer et enregistrer le module d'extension afin de le rendre disponible pour le conteneur de tâches utilisateur lors de l'exécution. Vous pouvez enregistrer des gestionnaires d'événements liés à l'API avec une instance de tâche, un modèle de tâche ou un composant d'application.

Concepts associés

Gestionnaires d'événements d'API

Les événements d'API surviennent lorsqu'une tâche utilisateur est modifiée ou change d'état. Pour permettre le traitement de ces événements d'API, le gestionnaire d'événements est appelé directement avant la modification de la tâche (méthode pré-événement) et juste après le renvoi de l'appel API (méthode post-événement).

Tâches associées

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Gestionnaires d'événements d'API

Les événements d'API surviennent lorsqu'une tâche utilisateur est modifiée ou change d'état. Pour permettre le traitement de ces événements d'API, le gestionnaire d'événements est appelé directement avant la modification de la tâche (méthode pré-événement) et juste après le renvoi de l'appel API (méthode post-événement).

Si la méthode pré-événement génère une exception `ApplicationVetoException`, l'action de l'API n'est pas exécutée, l'exception est renvoyée à l'appelant de l'API et la transaction associée à l'événement est annulée. Si la méthode pré-événement a été déclenchée par un événement interne et qu'une exception `ApplicationVetoException` est générée, l'événement interne (par exemple une réclamation automatique) n'est pas exécuté mais une exception est renvoyée à l'application client. Dans ce cas, un message d'information est enregistré dans le

fichier SystemOut.log. Si la méthode d'API génère une exception au cours du traitement, celle-ci est interceptée et transmise à la méthode post-événement. L'exception est de nouveau transmise à l'appelant lorsque la méthode post-événement est renvoyée.

Les règles suivantes s'appliquent aux méthodes pré-événement :

- Les méthodes pré-événement reçoivent les paramètres de la méthode d'API ou de l'événement interne associé(e).
- Les méthodes pré-événement peuvent générer une exception `ApplicationVetoException` pour empêcher la poursuite du traitement.

Les règles suivantes s'appliquent aux méthodes post-événement :

- Les méthodes post-événement reçoivent les paramètres fournis à l'appel d'API, puis renvoient les valeurs. Si une exception est émise par l'implémentation d'une méthode d'API, la méthode post-événement reçoit également l'exception.
- Les méthodes post-événement ne modifient pas les valeurs renvoyées.
- Les méthodes post-événement ne peuvent pas générer d'exceptions. Les exceptions d'exécution sont consignées, mais ignorées.

Pour implémenter les gestionnaires d'événements d'API, vous pouvez au choix faire appel à l'interface `APIEventHandlerPlugin3`, qui étend l'interface `APIEventHandlerPlugin`, ou bien étendre la classe d'implémentation SPI par défaut `com.ibm.task.spi.APIEventHandler`. Si votre gestionnaire d'événements hérite de la classe d'implémentation par défaut, il implémente toujours la version la plus récente de l'interface SPI. Si vous effectuez une mise à niveau vers une version plus récente de Business Process Choreographer, quelques modifications doivent être apportées si vous souhaitez utiliser de nouvelles méthodes d'interface SPI.

Si un gestionnaire d'événements de notification et un gestionnaire d'événements d'API sont présents simultanément, ils doivent tous deux porter le même nom, car il n'est possible de nommer qu'un seul gestionnaire.

Tâches associées

Création de gestionnaires d'événements d'API

Un événement d'API se produit lorsqu'une méthode d'API manipule une tâche utilisateur. Utilisez l'interface SPI du plug-in de gestionnaire d'événements d'API pour créer des plug-in permettant de gérer les événements de tâche envoyés par l'API ou par les événements internes ayant des événements API équivalents.

Création de gestionnaire d'événements de notification

Les événements de notification surviennent lors de l'escalade de tâches utilisateur. Business Process Choreographer fournit des fonctionnalités permettant la gestion des escalades, telles que la création d'éléments de travail d'escalade ou l'envoi de messages électroniques. Vous pouvez créer des gestionnaires d'événements de notification pour personnaliser le mode de traitement des escalades.

A propos de cette tâche

Pour implémenter les gestionnaires d'événements de notification, vous pouvez soit faire appel à l'interface `NotificationEventHandlerPlugin`, soit dériver la classe d'implémentation SPI par défaut `com.ibm.task.spi.NotificationEventHandler`.

Suivez la procédure ci-après pour créer un gestionnaire d'événements de notification.

Procédure

1. Générez une classe qui implémente l'interface `NotificationEventHandlerPlugin` ou étend la classe d'implémentation `NotificationEventHandler`. Cette classe permet d'appeler les méthodes des autres classes.

Si vous utilisez l'interface `NotificationEventHandlerPlugin`, vous devez implémenter toutes les méthodes de cette interface. Si vous étendez la classe d'implémentation `SPI`, remplacez les méthodes selon vos besoins.

Cette classe s'exécute dans le contexte d'une application d'entreprise EJB J2EE (Enterprise Java 2 Enterprise Edition). Assurez-vous que cette classe et ses classes auxiliaires suivent les la spécification EJB.

Le module d'extension est appelé avec les droits d'accès associés au rôle `EscalationUser`. Ce rôle est défini lorsque le conteneur des tâches utilisateur est configuré.

Remarque : Pour appeler l'interface `HumanTaskManagerService` à partir de cette classe, n'appellez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action peut entraîner une incohérence des données de tâche dans la base de données.

2. Assemblez la classe du plug-in et ses classes auxiliaires dans un fichier JAR.

Pour rendre le fichier JAR disponible, vous pouvez procéder de l'une des manières suivantes :

- En tant que fichier JAR d'utilitaire dans le fichier EAR de l'application.
- En tant que bibliothèque partagée installée avec le fichier EAR de l'application.
- En tant que bibliothèque partagée installée avec l'application `TaskContainer`. Dans ce cas, le module d'extension est disponible pour toutes les tâches.

3. Assemblez la classe du module d'extension et ses classes auxiliaires dans un fichier JAR.

Si les classes auxiliaires sont utilisées par plusieurs applications J2EE, vous pouvez les regrouper dans un fichier JAR distinct que vous enregistrez sous forme de bibliothèque partagée.

4. Créez un fichier de configuration de fournisseur de services pour le module d'extension dans le répertoire `META-INF/services/` de votre fichier JAR.

Le fichier de configuration fournit le mécanisme d'identification et de chargement du module d'extension. Ce fichier est conforme à la spécification de l'interface du fournisseur de services Java 2.

- a. Créez un fichier nommé

`com.ibm.task.spi.nom_module_extensionNotificationEventHandlerPlugin`, ou `nom_module_extension` est le nom du module d'extension.

Si, par exemple, votre module d'extension est nommé `HelpDeskRequest` (nom du gestionnaire d'événements) et qu'il implémente l'interface `com.ibm.task.spi.NotificationEventHandlerPlugin`, le fichier de configuration porte le nom

`com.ibm.task.spi.HelpDeskRequestNotificationEventHandlerPlugin`.

- b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire (c'est-à-dire commençant par le signe #) ni une ligne vide, doit spécifier le nom qualifié complet de la classe de module d'extension créée à l'étape 1.

Si par exemple la classe de module d'extension porte le nom `MyEventHandler` et est incluse dans le package `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante : `com.customer.plugins.MyEventHandler`.

Résultats

Vous disposez d'un fichier JAR installable contenant un module d'extension qui gère les événements de notification et d'un fichier de configuration de fournisseur de services pouvant servir à charger le module d'extension. Vous pouvez enregistrer des gestionnaires d'événements liés à l'API avec une instance de tâche, un modèle de tâche ou un composant d'application.

Remarques : Vous ne disposez que d'une propriété `eventHandlerName` pour enregistrer à la fois les gestionnaires d'événements d'API et les gestionnaires d'événements de notification. Pour utiliser à la fois un gestionnaire d'événement d'API et un gestionnaire d'événement de notification, il est nécessaire que les implémentations des plug-ins portent le même nom (Customer comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux plug-ins à l'aide d'une seule classe ou de classes distinctes. Dans les deux cas, vous devez créer deux fichiers dans le répertoire `META-INF/services/` de votre fichier JAR (par exemple, `com.ibm.task.spi.CustomerNotificationEventHandlerPlugin` et `com.ibm.task.spi.CustomerAPIEventHandlerPlugin`).

Regroupez l'implémentation du plug-in et les classes auxiliaires dans un seul fichier JAR.

Pour rendre effective une modification de l'implémentation, remplacez le fichier JAR contenu dans la bibliothèque partagée, déployez à nouveau le fichier EAR associé et redémarrez le serveur.

Que faire ensuite

Vous devez maintenant installer et enregistrer le module d'extension afin de le rendre disponible pour le conteneur de tâches utilisateur lors de l'exécution. Vous pouvez enregistrer des gestionnaires d'événements de notification avec une instance de tâche, un modèle de tâche ou un composant d'application.

Tâches associées

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Installation des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification

Pour pouvoir utiliser un module d'extension de gestionnaire d'événements d'API ou de notification, vous devez l'installer de sorte qu'il soit accessible au conteneur de tâches.

A propos de cette tâche

La façon dont vous installez le plug-in dépend de si le plug-in doit être utilisé par une seule application J2EE (Java 2 Enterprise Edition) ou par plusieurs applications.

Procédez de l'une des manières suivantes pour installer un module d'extension.

- Installez un module d'extension pour qu'il soit utilisé par une seule application J2EE.

Ajoutez le fichier JAR du module d'extension au fichier JAR de l'application. Dans l'éditeur du descripteur de déploiement de WebSphere Integration Developer, installez le fichier JAR de votre plug-in en tant que fichier JAR d'utilitaire de projet pour l'application J2EE du module EJB d'entreprise (entreprise JavaBeans) principal.

- Installez un module d'extension pour qu'il soit utilisé par plusieurs applications J2EE.

Placez le fichier JAR dans une bibliothèque partagée de WebSphere Application Server et associez la bibliothèque aux applications devant accéder au plug-in. Pour rendre le fichier JAR accessible dans un environnement de déploiement réseau, distribuez manuellement le fichier JAR sur chaque noeud hébergeant un serveur ou un membre de cluster sur lequel l'une de vos applications est déployée. Vous pouvez utiliser la portée de la cible de déploiement de vos applications, c'est-à-dire le serveur ou le cluster sur lequel les applications sont déployées, ou bien la portée de cellule. Souvenez-vous que les classes des modules d'extension sont alors visibles dans toute la portée de déploiement sélectionnée.

Que faire ensuite

Vous pouvez, maintenant, enregistrer le module d'extension.

Tâches associées

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Enregistrement des modules d'extension du gestionnaire d'événements d'API et du gestionnaire d'événements de notification avec des modèles de tâche et des tâches

Vous pouvez enregistrer les modules d'extension pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches et des modèles de tâche à différentes occasions : lors de la création d'une tâche ad-hoc, de la mise à jour d'une tâche existante, de la création d'un modèle de tâche ou de la définition d'un modèle de tâche.

A propos de cette tâche

Vous pouvez enregistrer des modules d'extension pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches à différents niveaux :

Modèle de tâche

Toutes les tâches créées à l'aide du modèle utilisent les mêmes gestionnaires

Modèle de tâche ad-hoc

Les tâches créées à l'aide du modèle utilisent les mêmes gestionnaires

Tâche ad-hoc

La tâche créée utilise les gestionnaires spécifiés

Tâche existante

La tâche utilise les gestionnaires spécifiés

Vous pouvez enregistrer un module d'extension en suivant l'une des procédures suivantes.

- Pour les modèles de tâches modélisés dans WebSphere Integration Developer, spécifiez le module d'extension dans le modèle de tâche.
- Pour les tâches ad-hoc ou modèles de tâches ad-hoc, spécifiez le module d'extension au moment de la création du tâche ou du modèle de tâche.
Utilisez la méthode `setEventHandlerName` de la classe `TTask` pour enregistrer le nom du gestionnaire d'événements.
- Modifiez le gestionnaire d'événements pour une instance de tâche lors de l'exécution.
La méthode `update(Task task)` vous permet d'utiliser un autre gestionnaire d'événements pour une instance de tâche lors de l'exécution. L'appelant doit disposer de droit d'accès administrateur pour mettre à jour cette propriété.

Tâches associées

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Création, installation et exécution de plug-ins en vue du post-traitement des résultats des requêtes sur les utilisateurs

La résolution d'utilisateurs renvoie une liste des utilisateurs auxquels un rôle spécifique est affecté, par exemple, le propriétaire potentiel d'une tâche. Vous pouvez créer un plug-in pour modifier les résultats des requêtes d'utilisateurs renvoyés par la résolution des utilisateurs. Par exemple, pour améliorer l'équilibrage de charge, vous pourriez avoir un plug-in qui supprime les utilisateurs du résultat de la requête s'ils ont déjà une charge de travail élevée.

A propos de cette tâche

Vous ne pouvez avoir qu'un seul plug-in de post-traitement : autrement dit, le plug-in doit gérer les résultats des requêtes sur les utilisateurs provenant de toutes les tâches. Votre plug-in peut ajouter ou supprimer des utilisateurs, ou modifier les informations d'utilisateur ou de groupe. Il peut également modifier le type de résultat, par exemple, provenant d'une liste d'utilisateurs à un groupe, ou à tout le monde.

Du fait que l'exécution des plug-in n'a lieu qu'après la résolution des utilisateurs, toutes les règles de confidentialité ou de sécurité éventuellement définies ont déjà été appliquées. Le plug-in reçoit des informations sur les utilisateurs qui ont été supprimés pendant la résolution des utilisateurs (dans la clé de mappe `HTM_REMOVED_USERS`). Vous devez vous assurer que le plug-in utilise ces informations de contexte pour préserver les règles de confidentialité ou de sécurité dont vous disposez éventuellement.

Pour implémenter le post-traitement des résultats de requête d'utilisateur, vous utilisez l'interface `StaffQueryResultPostProcessorPlugin`. L'interface contient des méthodes permettant de modifier les résultats de requête pour les tâches, les escalades, les modèles de tâche et les composants d'application.

Exécutez les étapes suivantes pour créer un plug-in permettant le post-traitement des résultats d'une requête d'utilisateur.

Procédure

1. Ecrivez une classe implémentant l'interface `StaffQueryResultPostProcessorPlugin`.

Cette classe s'exécute dans le contexte d'une application d'entreprise EJB J2EE (Enterprise Java 2 Enterprise Edition). Cette classe peut appeler les méthodes d'autres classes. Assurez-vous que cette classe et ses classes auxiliaires suivent les la spécification EJB.

Remarque : Pour appeler l'interface `HumanTaskManagerService` à partir de cette classe, n'appellez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action peut entraîner une incohérence des données de tâche dans la base de données.

Vous devez implémenter toutes les méthodes dans l'interface. Ces méthodes incluent des informations relatives aux critères d'affectation d'utilisateurs au modèle de tâche, à la tâche ou au rôle d'escalade en question.

- La définition des critères d'affectation d'utilisateurs est spécifiée sous forme d'entrée dans le paramètre **context** du type `Map`. Pour accéder à ces informations, procédez comme suit :

```
Map pacAsMap = (Map) context.get("HTM_VERB");

// extrait le nom des critères d'affectation d'utilisateurs
String pacName = (String) pacAsMap.get("HTM_VERB_NAME");

// extrait les noms de paramètre des critères d'affectation d'utilisateurs
Set paramNames = pacAsMap.keySet();

// extrait la valeur d'un paramètre spécifique
String paramValue = (String) pacAsMap.get(paramName);
```


- Les variables de substitution spécifiées en tant que valeurs pour le paramètre des critères d'affectation d'utilisateurs sont des entrées du paramètre **context** du type Map. Pour accéder à ces informations, procédez comme suit :

```
Object replVarObj = pacAsMap.get(replVarName);
if (replVarObj instanceof String)
    String replVarValue = (String) replVarObj;
if (replVarObj instanceof String[])
    String[] replVarValues = (String[]) replVarObj;
```

- L'objet StaffQueryResult est créé en accédant à un annuaire de personnes au cours de la résolution des utilisateurs, par exemple en accédant à l'annuaire de personnes de Virtual Member Manager.

L'objet StaffQueryResult contient des informations sur les entrées d'utilisateur extraites lors de la résolution des utilisateurs. Pour plus d'informations, consultez les informations de référence Javadoc concernant l'interface StaffQueryResultPostProcessorPlugin.

- La liste des utilisateurs explicitement exclus lors de la résolution des utilisateurs est stockée en tant qu'entrée du paramètre **context** du type Map. Pour accéder à ces informations, procédez comme suit :

```
String[] removedUserIDs = (String[]) context.get("HTM_REMOVED_USERS");
```

L'exemple suivant indique comment modifier le rôle d'éditeur d'une tâche appelée SpecialTask.

```
public StaffQueryResult processStaffQueryResult
    (StaffQueryResult originalStaffQueryResult,
     Task task,
     int role,
     Map context)
{
    StaffQueryResult newStaffQueryResult = originalStaffQueryResult;
    StaffQueryResultFactory staffResultFactory =
        StaffQueryResultFactory.newInstance();
    if (role == com.ibm.task.api.WorkItem.REASON_EDITOR &&
        task.getName() != null &&
        task.getName().equals("SpecialTask"))
    {
        UserData user = staffResultFactory.newUserData
            ("SuperEditor",
             new Locale("en-US"),
             "SuperEditor@company.com");
        ArrayList userList = new ArrayList();
        userList.add(user);

        newStaffQueryResult = staffResultFactory.newStaffQueryResult(userList);
    }
    return(newStaffQueryResult);
}
```

2. Assemblez la classe du plug-in et ses classes auxiliaires dans un fichier JAR. Vous pouvez rentre le fichier JAR disponible dans une bibliothèque partagée et l'associer avec le conteneur de tâches. Dans ce cas, le module d'extension devient disponible pour toutes les tâches.
3. Créez un fichier de configuration de fournisseur de services pour le plug-in dans le répertoire META-INF/services/ du fichier JAR.

Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le plug-in. Ce fichier est conforme à la spécification de l'interface du fournisseur de services Java 2.

- a. Créez un fichier portant le nom `com.ibm.task.spi.nom_plug-inStaffQueryResultPostProcessorPlugin`, où `nom_plug-in` correspond au nom du plug-in.

Par exemple, si votre plug-in s'appelle MyHandler et qu'il implémente l'interface `com.ibm.task.spi.StaffQueryResultPostProcessorPlugin`, le nom du fichier de configuration sera `com.ibm.task.spi.MyHandlerStaffQueryResultPostProcessorPlugin`.

- b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire (c'est-à-dire commençant par le signe #) ni une ligne vide, doit spécifier le nom qualifié complet de la classe de module d'extension créée à l'étape 1.

Par exemple, si la classe de votre plug-in est `StaffPostProcessor` et se trouve dans le module `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante :

`com.customer.plugins.StaffPostProcessor`. Vous avez un fichier JAR installable qui contient un plug-in assurant le post-traitement des résultats de requête d'utilisateur et un fichier de configuration du fournisseur de services pouvant être utilisé pour charger le plug-in.

4. Installez le module d'extension.

Vous ne pouvez avoir qu'un seul plug-in de post-traitement pour les résultats de requête d'utilisateur. Vous devez installer le plug-in en tant que bibliothèque partagée.

- a. Définissez une bibliothèque partagée de WebSphere Application Server pour le module d'extension. Définissez la bibliothèque partagée dans la portée du serveur ou cluster sur lequel Business Process Choreographer est configuré. Ensuite, associez cette bibliothèque partagée à l'application `TaskContainer`. Cette étape ne doit être effectuée qu'une seule fois.
- b. Rendez le fichier JAR de plug-in disponible pour chaque installation de WebSphere Process Server affectée qui héberge un serveur ou un membre de cluster.

5. Enregistrez le plug-in.

- a. Dans la console d'administration, accédez à la page Propriétés personnalisées de Human Task Manager.

Cliquez sur **Serveurs** → **Serveurs d'applications** → *nom_serveur* dans un environnement autonome, sur **Serveurs** → **Clusters** → *nom_cluster* si Business Process Choreographer est configuré dans un cluster. Sous **Intégration métier**, sélectionnez **Human Task Manager**. Dans **Propriétés supplémentaires**, cliquez sur **Propriétés personnalisées**.

- b. Ajoutez une propriété personnalisée nommée **Staff.PostProcessorPlugin** et ainsi que la valeur du nom que vous avez donné à votre plug-in (MyHandler dans cet exemple).

Le module d'extension est désormais disponible pour effectuer le post-traitement des résultats de requête du personnel. Si vous modifiez le fichier JAR, remplacez le fichier dans la bibliothèque partagée, puis redémarrez le serveur.

6. Exécutez le plug-in. Le plug-in de post-traitement est appelé une fois que les opérations d'affectation et de substitution des utilisateurs sont terminées. Le plug-in est appelé en utilisant les informations spécifiées par l'interface `StaffQueryResultPostProcessorPlugin`.

Tâches associées

Création de modules d'extension pour personnaliser les fonctionnalités des tâches utilisateur

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches utilisateur.

L'application des modules d'extension est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des modules d'extension personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Partie 2. Déploiement des applications

Chapitre 5. Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Remarque : Le processus d'installation d'applications dans un environnement de production est similaire au processus décrit dans la rubrique «Développement et déploiement d'applications» présente dans le centre de documentation de WebSphere Application Server Network Deployment, version 6. Si vous ne connaissez pas ces rubriques, reportez-vous y en premier.

Avant d'installer un module dans un environnement de production, vérifiez à chaque fois les modifications dans un environnement de test. Pour installer des modules dans un environnement de test, utilisez WebSphere Integration Developer (voir le centre de documentation WebSphere Integration Developer pour plus d'informations). Pour installer des modules dans un environnement de production, utilisez WebSphere Process Server.

Cette rubrique décrit les concepts et les tâches nécessaires à la préparation et à l'installation de modules dans un environnement de production. Les autres rubriques décrivent les fichiers contenant les objets que votre module utilise et vous aide à déplacer ce module de l'environnement de test vers l'environnement de production. Il est important de comprendre ces fichiers et leur contenu pour être sûr d'avoir installé vos modules correctement.

Concepts associés

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques. Les bibliothèques et les artefacts sont inclus dans les fichiers d'archive Java (JAR) que vous identifiez lors du déploiement d'un module.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques. Les bibliothèques et les artefacts sont inclus dans les fichiers d'archive Java (JAR) que vous identifiez lors du déploiement d'un module.

Lors du développement d'un module, il est possible d'identifier certaines ressources ou composants qui peuvent être utilisés par différentes parties du module. Ces ressources ou composants peuvent être des objets créés lors du développement du module ou des objets existants se trouvant dans une bibliothèque déjà déployée sur le serveur. Cette rubrique décrit les bibliothèques et les fichiers dont vous aurez besoin lors de l'installation d'une application.

Bibliothèque

Une bibliothèque contient des objets ou des ressources utilisés par plusieurs modules dans WebSphere Integration Developer. Les artefacts peuvent se trouver dans des fichiers JAR, des fichiers archive de ressources (RAR) ou des fichiers archive de services (WAR). Ces artefacts sont notamment :

- des interfaces ou des descripteurs de services Web (fichiers ayant une extension .wsdl) ;
- des définitions de schéma XML d'objets métier (fichiers ayant une extension .xsd) ;
- des mappes d'objets métier (fichiers ayant une extension .map) ;
- des définitions de relations et de rôles (fichiers ayant une extension .rel et .rol).

Lorsqu'un module doit utiliser un artefact, le serveur recherche cet artefact à partir du chemin d'accès aux classes EAR et le charge, s'il n'est pas déjà chargé dans la mémoire. A partir de ce moment, toute requête portant sur l'artefact utilise cette copie jusqu'à son remplacement. La figure 29, à la page 453 illustre les composants et les bibliothèques d'une application.

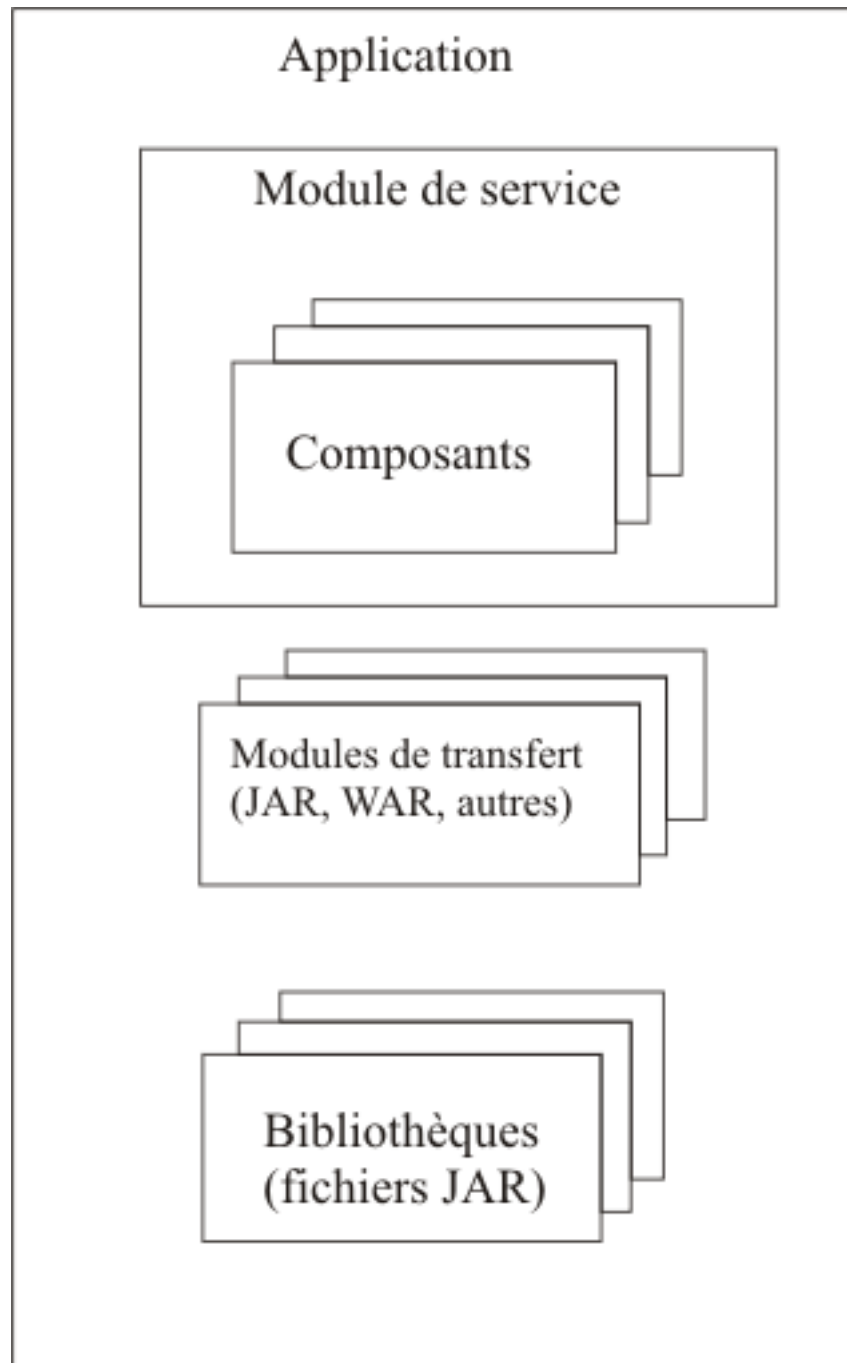


Figure 29. Relations entre module, composants et bibliothèques

Fichiers JAR, RAR et WAR

Un certain nombre de fichiers peuvent contenir des composants d'un module. Ces fichiers sont décrits en détails dans la spécification Java Platform, Enterprise Edition (J2EE). Une description détaillée des fichiers JAR est disponible dans la spécification JAR.

Dans WebSphere Process Server, un fichier JAR contient également une application qui est la version assemblée du module comprenant toutes les références de prise en charge et les interfaces vers tous les autres composants de service utilisés par le module. Pour installer l'application complète, vous avez besoin de ce fichier JAR et de toutes autres bibliothèques : fichiers JAR, fichiers WAR (archive Web), fichiers RAR (archive de ressources), fichiers JAR de bibliothèques de transfert (EJB - Enterprise Java Beans) ou de toutes autres archives, et vous devez créer un fichier EAR installable à l'aide de la commande `serviceDeploy` .

Conventions de dénomination pour les modules de transfert

Dans la bibliothèque, des conventions de dénomination s'appliquent aux noms des modules de transfert. Ces noms sont uniques pour un module spécifique. Nommez les autres modules requis pour déployer l'application en veillant à éviter tout conflit avec les noms des modules de transfert. Pour un module nommé *myService*, les noms de modules de transfert sont les suivants :

- *myServiceApp*
- *myServiceEJB*
- *myServiceEJBClient*
- *myServiceWeb*

Remarque : La commande `serviceDeploy` crée le module de transfert *myServiceWeb* uniquement si le service inclut un service de type de port WSDL.

Remarques concernant l'utilisation de bibliothèques

L'utilisation de bibliothèques assure la cohérence des objets métier et celle du traitement entre les différents modules étant donné que chaque module appelant dispose de sa propre copie d'un composant spécifique. Pour empêcher les incohérences et les erreurs, il est important de veiller à ce que les modifications apportées aux composants et aux objets métiers utilisés par les modules appelants soient coordonnées avec l'ensemble des modules appelants. Pour mettre les modules appelants à jour, procédez comme suit :

1. copiez le module et la copie la plus récente des bibliothèques sur le serveur de production ;
2. recréez le fichier EAR installable à l'aide de la commande `serviceDeploy` ;
3. arrêtez l'application en cours d'exécution qui contient le module appelant et réinstallez-la ;
4. redémarrez l'application qui contient le module appelant.

Concepts associés

Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Un fichier d'archive d'entreprise (EAR) est un fichier compressé qui contient les bibliothèques, les beans enterprise et les fichiers JAR nécessaires au déploiement de l'application.

Les fichiers JAR sont créés lors de l'exportation des modules d'application à partir de WebSphere Integration Developer. Ce fichier JAR et toutes autres bibliothèques d'artefacts ou objets sont utilisés en tant qu'entrées dans le processus d'installation. La commande `serviceDeploy` crée un fichier EAR à partir des fichiers d'entrée contenant les descriptions des composants et le code Java qui forment l'application.

Concepts associés

Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques. Les bibliothèques et les artefacts sont inclus dans les fichiers d'archive Java (JAR) que vous identifiez lors du déploiement d'un module.

Tâches associées

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Avant de commencer

Avant de commencer, vous devez avoir développé et testé vos modules sur un serveur de test et résolu les incidents et les problèmes liés aux performances.

Important : Pour éviter de remplacer une application ou un module s'exécutant déjà dans un environnement de déploiement, assurez-vous que le nom du module ou de l'application est différent de celui déjà installé.

A propos de cette tâche

Cette tâche vérifie que toutes les pièces nécessaires d'une application sont disponibles et rassemblées dans les bons fichiers pour être amenées vers le serveur de production.

Remarque : Vous pouvez également exporter un fichier d'archive d'entreprise (EAR) à partir de WebSphere Integration Developer et installer ce fichier directement dans WebSphere Process Server.

Important : Si les services internes d'un composant utilisent une base de données, installez l'application sur un serveur connecté directement à une base de données.

Procédure

1. Localisez le dossier contenant les composants du module que vous souhaitez déployer.

- Le dossier contenant les composants doit porter le nom *module-nomet* contenir un fichier nommé *module.module* correspondant au module de base.
2. Vérifiez que tous les composants contenus dans le module se trouvent dans les sous-dossiers de composant sous le dossier du module.
Pour faciliter l'utilisation, nommez le sous-dossier de la façon suivante *module/composant*.
 3. Vérifiez que tous les fichiers comprenant chacun des composants font partie du bon sous-dossier de composant et ont un nom ressemblant à *composant-fichier-nom.composant*.
Les fichiers de composants contiennent les définitions de chaque composant individuel à l'intérieur du module.
 4. Vérifiez que tous les autres composants et artefacts se trouvent bien dans les sous-dossiers de composants qui exigent leur présence.
Lors de cette étape, vous allez vérifier que toutes les références à des outils nécessaires à un composant sont disponibles. Les noms de composants ne doivent pas entrer en conflit avec les noms que la commande `serviceDeploy` utilise pour hiérarchiser les modules. Voir convention de dénomination des modules de transfert.
 5. Vérifiez que le fichier de références, *module.references*, existe bien dans le dossier module de l'étape 1, à la page 456.
Le fichier de références définit les références et les interfaces à l'intérieur du module.
 6. Vérifiez que le fichier câblage, *module.wires*, existe bien dans le dossier composant.
Le fichier câblage complète les connexions entre les références et les interfaces du module.
 7. Vérifiez que le fichier manifeste, *module.manifest*, existe bien dans le dossier composant.
Le manifeste liste les composants contenus dans le module. Il contient également une instruction de chemin de classes afin de permettre à la commande `serviceDeploy` de localiser tout autre module nécessaire au module.
 8. Créez un fichier compressé ou un fichier JAR du module représentant l'entrée de la commande `serviceDeploy` que vous utiliserez afin de préparer l'installation du module vers le serveur de production.

Exemple de structure de dossier pour un module MyValue avant déploiement

Ce qui suit illustre la structure de répertoire du module `MyValueModule` comprenant les composants `MyValue`, `CustomerInfo` et `StockQuote`.

```
MyValueModule
  MyValueModule.manifest
  MyValueModule.references
  MyValueModule.wiring
  MyValueClient.jsp
process/myvalue
  MyValue.component
  MyValue.java
  MyValueImpl.java
service/customerinfo
  CustomerInfo.component
  CustomerInfo.java
  Customer.java
  CustomerInfoImpl.java
service/stockquote
```

```
StockQuote.component
StockQuote.java
StockQuoteAsynch.java
StockQuoteCallback.java
StockQuoteImpl.java
```

Que faire ensuite

Installez le module sur les systèmes de production comme décrit à la rubrique Installation d'un module sur un serveur de production.

Concepts associés

Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques. Les bibliothèques et les artefacts sont inclus dans les fichiers d'archive Java (JAR) que vous identifiez lors du déploiement d'un module.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Les clusters apportent de nombreux avantages à votre environnement de traitement grâce aux économies d'échelle, ce qui permet d'équilibrer la charge des requêtes entre serveurs et fournit un niveau de disponibilité pour les clients des applications. Avant d'installer une application contenant des services sur un cluster, tenez compte des points suivants :

- Les utilisateurs de l'application ont-ils besoin de la puissance et de la disponibilité de traitement des clusters ?
Si c'est le cas, la mise en cluster est la solution adéquate. La mise en cluster augmente la disponibilité et la capacité de vos applications.
- Le cluster est-il préparé correctement pour les applications de service ?
Vous devez configurer le cluster correctement avant d'installer et de démarrer la première application contenant un service. Le cluster doit être configuré correctement pour que les requêtes soient traitées correctement.
- Un cluster de secours est-il installé ?
Vous devez installer l'application sur le cluster de secours également.

Concepts associés

Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques. Les bibliothèques et les artefacts sont inclus dans les fichiers d'archive Java (JAR) que vous identifiez lors du déploiement d'un module.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Chapitre 6. Déploiement d'un module

Vous pouvez déployer un module ou un module de médiation généré par WebSphere Integration Developer, dans un environnement de production WebSphere Process Server en suivant la procédure ci-dessous.

Avant de commencer

Avant de déployer une application de service sur un serveur de production, assemblez et testez l'application sur un serveur test. A l'issue du test, exportez les fichiers adéquats comme cela est décrit dans *Préparation du déploiement sur un serveur* dans le fichier PDF Développement et déploiement de modules et transférez les fichiers sur le système de production en vue du déploiement. Pour plus d'informations, consultez les centres de documentation WebSphere Integration Developer et WebSphere Application Server Network Deployment.

Procédure

1. Copiez le module et d'autres fichiers sur le serveur de production.
Les modules et ressources (fichiers EAR, JAR, RAR et WAR) requis par l'application sont transférés sur votre environnement de production.
2. Exécutez la commande `serviceDeploy` pour créer un fichier EAR installable.
Cette étape définit le module auprès du serveur en préparation de l'installation de l'application en production.
 - a. Localisez le fichier JAR qui contient le module à déployer.
 - b. Exécutez la commande `serviceDeploy` en utilisant le fichier JAR de l'étape précédente comme entrée.
3. Installez le fichier EAR à partir de l'étape 2. Le mode d'installation des applications dépend de la destination : serveur autonome ou serveur dans une cellule.

Remarque : Vous pouvez utiliser la console d'administration ou un script pour installer l'application. Pour plus d'informations, voir le centre de documentation WebSphere Application Server.

4. Sauvegardez la configuration. Le module est installé en tant qu'application.
5. Lancez l'application.

Résultats

L'application est active ; le flux de travail doit circuler via le module.

Que faire ensuite

Contrôlez l'application pour vous assurer que le serveur traite correctement les demandes.

Tâches associées

Installation de modules SCA versionnés dans un environnement de production
Vous pouvez déployer des modules SCA versionnés dans la phase d'exécution. Chaque version d'un module existe parallèlement aux autres versions actuellement installées sur le serveur ou dans la cellule.

Installation d'un module SCA avec la console

Avant de démarrer l'exécution d'un module ou d'un module de médiation, vous devez le déployer sur un serveur ou un cluster. Le déploiement implique la création d'un fichier EAR installable et l'installation du fichier EAR sur le serveur ou le cluster.

Création d'un fichier EAR installable via serviceDeploy

Pour installer une application dans l'environnement de production, utilisez les fichiers copiés sur le serveur de production et créez un fichier EAR installable.

Déploiement d'applications à l'aide des tâches ANT Apache

Les tâches permettent de définir le déploiement de plusieurs applications sur WebSphere Process Server et de les exécuter sans surveillance sur un serveur.

Installation de modules SCA versionnés dans un environnement de production

Vous pouvez déployer des modules SCA versionnés dans la phase d'exécution. Chaque version d'un module existe parallèlement aux autres versions actuellement installées sur le serveur ou dans la cellule.

Avant de commencer

Assurez-vous d'effectuer les opérations suivantes avant d'installer un module SCA versionné dans votre environnement de production :

- Dans WebSphere Integration Developer, précisez que le module est versionné et exportez-le en vue d'un déploiement sur la ligne de commande. Pour plus d'informations, voir la rubrique sur la création de bibliothèques et de modules versionnés.
- Déterminez si vous souhaitez déployer simultanément différentes versions du module sur un seul serveur ou si vous souhaitez déployer simultanément plusieurs instances du même module versionné sur divers clusters de la même cellule.

A propos de cette tâche

Pour installer des modules versionnés, procédez comme suit :

Procédure

1. Exécutez serviceDeploy sur le module versionné que vous avez exporté pour générer un fichier EAR installable.

```
serviceDeploy nomModule.zip
```

La commande serviceDeploy renvoie un fichier EAR installable dont le nom inclut la version, et parfois, des informations sur l'ID de la cellule.

2. Installez le module d'une des manières suivantes :
 - Dans la console d'administration, cliquez sur Modules SCA et cliquez sur le bouton **Installer** de la page Modules SCA.
 - Dans la console d'administration, cliquez sur Applications > Installation d'une nouvelle application.

- Exécutez la commande `wsadmin AdminApp.install`.
3. Si vous souhaitez installer un module versionné sur plusieurs serveurs ou clusters dans une cellule, effectuez l'opération suivante pour chaque instance de module désirée :
 - a. Exécutez la commande `createVersionedSCAModule` pour créer une instance du module.


```
createVersionedSCAModule -archiveAbsolutePath rép_archive_entrée
-workingDirectory rép_travail -uniqueCellID ID_cellule
```
 - b. Installez le fichier EAR résultant comme décrit dans l'étape 2, à la page 462.
 4. Facultatif : Exécutez la commande `validateSCAImportExportInformation` pour confirmer l'existence de toutes les liaisons SCA et liaisons d'exportation du sélecteur dans le fichier EAR spécifié.

Résultats

Votre environnement de production compte désormais une ou de plusieurs applications versionnées. Ces dernières peuvent toutes être administrées via la console d'administration ou à l'aide des commandes d'administration correspondantes.

Remarque : Pour conserver les informations de gestion des versions, le processus d'installation modifie automatiquement le nom du module pour s'assurer qu'il est unique au sein du serveur ou de la cellule via l'exécution de la commande `serviceDeploy` ou `createVersionedSCAModule`. Ces commandes ajoutent le numéro de version, un ID de cellule unique ou le nom du module d'origine.

nomModule_vvaleurVersion_iDCellule unique

Par exemple, si vous avez suivi les étapes contenues dans cette section, le déploiement de la version 1.0.1 du module `processusFacturation` entraîne la création d'un module appelé `processusFacturation_v1_0_1` et d'une application de service installée appelée `processusFacturation_v1_0_1App`. Si vous indiquez également un ID de cellule unique (exemple : `Cellule5`), le module porte le nom `processusFacturation_v1_0_1_Cellule5` et l'application de service installée, le nom `processusFacturation_v1_0_1_Cellule5App`.

Tâches associées

Déploiement d'un module de

Vous pouvez déployer un module ou un module de médiation généré par WebSphere Integration Developer, dans un environnement de production WebSphere Process Server en suivant la procédure ci-dessous.

Installation d'un module SCA avec la console

Avant de démarrer l'exécution d'un module ou d'un module de médiation, vous devez le déployer sur un serveur ou un cluster. Le déploiement implique la création d'un fichier EAR installable et l'installation du fichier EAR sur le serveur ou le cluster.

Avant de commencer

Si vous avez exporté votre module ou module de médiation vers un fichier JAR, utilisez la commande `serviceDeploy` pour créer un fichier EAR installable à partir du fichier JAR. Pour plus d'informations, reportez-vous au Chapitre 6, «Déploiement d'un module», à la page 461.

A propos de cette tâche

Vous devez installer le fichier EAR sur un serveur ou un cluster avant de pouvoir lancer l'exécution du module ou du module de médiation.

Au lieu d'utiliser la console d'administration, vous pouvez utiliser d'autres méthodes pour installer le fichier EAR, telles que la commande `AdminApp.install` ou `AdminApp.installinteractive` à l'aide de l'outil `wsadmin`.

Important : Si, après avoir commencé la procédure, vous ne souhaitez plus installer l'application, cliquez sur **Annuler** : ne vous contentez pas de passer sur une autre page de la console d'administration.

Procédure

1. A partir de la console d'administration, cliquez sur **Applications** → **Installer une nouvelle application** dans le panneau de navigation de la console. La première des deux pages de la **préparation de l'installation de l'application** s'affiche.
2. Dans la première page **Préparation de l'installation de l'application** :
 - a. Indiquez le chemin d'accès complet du fichier EAR. Pour plus d'informations, voir *Installation des applications à l'aide de la console*.
 - b. Indiquez si vous acceptez les valeurs par défaut ou indiquez de nouvelles valeurs :
 - Me prévenir uniquement lorsque des informations supplémentaires sont requises.**
Affiche uniquement la phase de mappage du module et les autres phases dans lesquelles vous devez fournir des informations.
 - Me montrer toutes les options et tous les paramètres d'installation.**
Affiche toutes les phases de l'installation. Pour utiliser **Générer les liaisons par défaut**, qui permet d'entrer les valeurs par défaut pour les liaisons non renseignées, sélectionnez cette option.
 - c. Cliquez sur **Suivant**.
3. L'installation d'un fichier EAR contenant un flux de médiation est identique à l'installation de tout autre fichier EAR d'application d'entreprise dans WebSphere Application Server. Pour obtenir des informations détaillées pour renseigner la seconde page de la **préparation de l'installation de l'application** et préciser les options dans les dernières étapes de l'assistant, voir *Installation des applications à l'aide de la console*.
4. Lorsque vous installez un module de médiation ou un module contenant un flux de médiation, vous pouvez effectuer une étape supplémentaire. Dans le panneau **Modifier les propriétés du module**, vous pouvez modifier les valeurs des propriétés du module. Si les propriétés appartiennent à un groupe, elles s'affichent dans une section à développer sinon vous les voyez immédiatement.

Résultats

Vous pouvez à présent démarrer le module ou module de médiation.

Tâches associées

Déploiement d'un module de
Vous pouvez déployer un module ou un module de médiation généré par WebSphere Integration Developer, dans un environnement de production WebSphere Process Server en suivant la procédure ci-dessous.

Création d'un fichier EAR installable via serviceDeploy

Pour installer une application dans l'environnement de production, utilisez les fichiers copiés sur le serveur de production et créez un fichier EAR installable.

Avant de commencer

Avant de commencer cette tâche, vous devez disposer d'un fichier JAR contenant le module et les services que vous déployez sur le serveur. Pour plus d'informations, voir «Préparation du déploiement vers un serveur».

A propos de cette tâche

La commande serviceDeploy utilise un fichier JAR, d'autres fichiers EAR, JAR, RAR, WAR et ZIP dépendants et crée un fichier EAR que vous pouvez installer sur un serveur.

Procédure

1. Localisez le fichier JAR qui contient le module à déployer.
2. Exécutez la commande serviceDeploy en utilisant le fichier JAR de l'étape précédente comme entrée.
Cette étape crée un fichier EAR.

Remarque : Suivez la procédure suivante sur une console d'administration.

3. Sélectionnez le fichier EAR à installer dans la console d'administration du serveur.
4. Cliquez sur **Sauvegarder** pour installer le fichier EAR.

Tâches associées

Déploiement d'un module de
Vous pouvez déployer un module ou un module de médiation généré par WebSphere Integration Developer, dans un environnement de production WebSphere Process Server en suivant la procédure ci-dessous.

Déploiement d'applications à l'aide des tâches ANT Apache

Les tâches permettent de définir le déploiement de plusieurs applications sur WebSphere Process Server et de les exécuter sans surveillance sur un serveur.

Avant de commencer

Cette tâche suppose que :

- Les applications déployées ont déjà été développées et testées.
- Les applications doivent être installées sur le(s) même(s) serveur(s).
- Vous avez certaines connaissances en tâches ANT Apache.
- Vous comprenez le processus de déploiement.

Des informations sur le développement et le test d'applications se trouvent dans le centre de documentation WebSphere Integration Developer.

La section de référence Documentation sur les interfaces API et SPI générées fournit des informations détaillées sur les interfaces de programme d'application. Les tâches ANT Apache sont décrites dans le module `com.ibm.websphere.ant.tasks`. Pour la présente rubrique, les tâches utilisées sont `ServiceDeploy` et `InstallApplication`.

A propos de cette tâche

Si vous devez installer des applications multiples simultanément, développez une tâche ANT Apache, avant le déploiement. La tâche ANT Apache peut alors déployer et installer les applications sur les serveurs sans que vous n'ayez à intervenir.

Procédure

1. Identifiez les applications à déployer.
2. Créez un fichier JAR pour chaque application.
3. Copiez les fichiers JAR sur les serveurs cible.
4. Créez une tâche ANT Apache pour exécuter la commande `ServiceDeploy` afin de créer un fichier EAR pour chaque serveur.
5. Créez une tâche ANT Apache pour exécuter la commande `InstallApplication` pour chaque fichier EAR depuis l'étape 4 sur les serveurs concernés.
6. Exécutez la tâche ANT `ServiceDeploy` Apache pour créer le fichier EAR pour les applications.
7. Exécutez la tâche ANT `InstallApplication` Apache afin d'installer les fichiers EAR à partir de l'étape 6.

Résultats

Les applications sont correctement déployées sur les serveurs cible.

Exemple de déploiement automatique d'une application

Dans cet exemple de déploiement d'une application sans surveillance, une tâche ANT Apache est contenue dans un fichier `myBuildScript.xml`.

```
<?xml version="1.0">
<project name="OwnTaskExample" default="main" basedir=".">
  <taskdef name="servicedeploy"
    classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
  <target name="main" depends="main2">
    <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
      ignoreErrors="true"
      outputApplication="c:/synctest/SyncTargetEAREAR"
      workingDirectory="c:/synctest"
      noJ2eeDeploy="true"
      cleanStagingModules="true"/>
  </target>
</project>
```

Cette instruction montre comment appeler la tâche ANT Apache.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

Conseil : Plusieurs applications peuvent être déployées automatiquement en ajoutant des instructions de projet supplémentaires au fichier.

Que faire ensuite

Utilisez la console d'administration pour vérifier que les applications nouvellement installées sont démarrées et traitent le flux de travail correctement.

Tâches associées

Déploiement d'un module de

Vous pouvez déployer un module ou un module de médiation généré par WebSphere Integration Developer, dans un environnement de production WebSphere Process Server en suivant la procédure ci-dessous.

Chapitre 7. Installation des applications de tâche utilisateur et de processus métier

Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Avant de commencer

Vérifiez que Business Flow Manager et Human Task Manager sont installés et configurés pour chaque serveur d'applications ou cluster sur lequel vous souhaitez installer l'application.

A propos de cette tâche

Vous pouvez installer des applications de processus métier et de tâche à partir de la console d'administration ou de la ligne de commande, ou en exécutant un script d'administration.

Résultats

Après l'installation d'une application de processus métier ou de tâche utilisateur, tous les modèles de processus métier et de tâche utilisateur passent à l'état "Démarré". Vous pouvez créer des instances de processus et de tâche à partir de ces modèles.

Que faire ensuite

Pour pouvoir créer des instances de processus ou de tâche, vous devez démarrer l'application.

Concepts associés

Installation d'applications de processus métier et de tâches utilisateur dans un environnement de déploiement réseau

Lorsque des modèles de processus ou de tâches utilisateur sont installés dans un environnement de déploiement réseau, les actions suivantes sont automatiquement exécutées par le programme d'installation des applications.

Déploiement des processus métier et des tâches utilisateur

Lorsque WebSphere Integration Developer ou le déploiement de service génère du code de déploiement pour votre processus ou votre tâche, chaque composant de processus ou composant de tâche est mappé avec un bean entreprise de session. L'ensemble du code de déploiement est mis en forme dans le fichier d'application d'entreprise (EAR). De plus, pour chaque processus, une classe Java représentant le code Java dans ce processus est générée et imbriquée dans le fichier EAR au cours de l'installation de l'application d'entreprise. Chaque nouvelle version d'un modèle devant être déployé doit être mise en forme dans une nouvelle application d'entreprise.

Tâches associées

Installation d'applications de processus métier et de tâche utilisateur en mode interactif

Vous pouvez installer une application en mode interactif lors son exécution à l'aide de l'outil wsadmin et du script installInteractive. Vous pouvez utiliser le script pour modifier les paramètres qui ne sont pas modifiables si vous utilisez la console d'administration pour installer l'application.

Désinstallation d'applications de processus métier et de tâche utilisateur à l'aide de la console d'administration

Vous pouvez utiliser la console d'administration pour désinstaller des applications contenant des processus métier ou des tâches utilisateur.

Désinstallation d'applications de processus métier et de tâches humaines à l'aide d'une commande d'administration

L'utilisation du script bpcTemplates.jacl est une alternative à l'emploi de la console d'administration pour désinstaller des applications qui contiennent des processus métier ou des tâches humaines.

Installation d'applications de processus métier et de tâches utilisateur dans un environnement de déploiement réseau

Lorsque des modèles de processus ou de tâches utilisateur sont installés dans un environnement de déploiement réseau, les actions suivantes sont automatiquement exécutées par le programme d'installation des applications.

L'application est installée par étapes. Chaque étape doit être exécutée avec succès pour que la suivante puisse débiter.

1. L'installation d'application démarre sur le gestionnaire de déploiement.

Au cours de cette étape, les modèles de processus métier et de tâche utilisateur sont configurés dans le référentiel de configuration WebSphere. L'application est également validée. Si des erreurs se produisent, elles sont consignées dans les fichiers System.out et System.err, ou en tant qu'entrées FFDC dans le gestionnaire de déploiement.

2. L'installation de l'application se poursuit sur l'agent de noeud.

Au cours de cette étape, l'installation de l'application sur une instance de serveur d'applications est déclenchée. Cette instance de serveur d'applications est soit la cible de déploiement, soit fait partie de celle-ci. Si la cible de déploiement est un cluster comprenant plusieurs membres, l'instance du

serveur est choisie arbitrairement parmi les membres de ce cluster. Si des erreurs se produisent au cours de cette étape, elles sont consignées dans les fichiers SystemOut.log et SystemErr.log, ou en tant qu'entrées FFDC sur l'agent de noeud.

3. L'application s'exécute sur l'instance de serveur.

Au cours de cette étape, les modèles de processus métier et de tâche utilisateur sont déployés dans la base de données de Business Process Choreographer sur la cible de déploiement. Si des erreurs se produisent, elles sont consignées dans les fichiers System.out et SystemErr.log ou en tant qu'entrées FFDC sur l'instance de serveur.

Tâches associées

Installation des applications de tâche utilisateur et de processus métier

Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Déploiement des processus métier et des tâches utilisateur

Lorsque WebSphere Integration Developer ou le déploiement de service génère du code de déploiement pour votre processus ou votre tâche, chaque composant de processus ou composant de tâche est mappé avec un bean entreprise de session. L'ensemble du code de déploiement est mis en forme dans le fichier d'application d'entreprise (EAR). De plus, pour chaque processus, une classe Java représentant le code Java dans ce processus est générée et imbriquée dans le fichier EAR au cours de l'installation de l'application d'entreprise. Chaque nouvelle version d'un modèle devant être déployé doit être mise en forme dans une nouvelle application d'entreprise.

Lorsque vous installez une application d'entreprise qui contient des processus métier ou des tâches utilisateur, ces derniers sont stockés dans des modèles de processus métier ou des modèles de tâches utilisateur, au sein de la base de données du Business Process Choreographer. Les modèles nouvellement installés sont, par défaut, à l'état démarré. Toutefois, l'application d'entreprise nouvellement installée se trouve à l'état arrêté. Chaque application d'entreprise installée peut être démarrée et arrêtée individuellement.

Vous pouvez déployer de nombreuses versions différentes d'un modèle de processus ou de tâche, chacune dans une application d'entreprise différente. Lorsque vous installez une nouvelle application d'entreprise, la version du modèle qui est installée est déterminée comme suit :

- Si le nom du modèle et l'espace de nom cible n'existent pas, un nouveau modèle est installé.
- Si le nom du modèle et l'espace de nom cible sont identiques à ceux du modèle existant, mais que la date de début de validité est différente, une nouvelle version du modèle existant est installée.

Remarque : Le nom du modèle est dérivé du nom du composant et non du processus métier ou de la tâche utilisateur.

Si vous n'indiquez pas de date de début de validité, la date est déterminée de la façon suivante :

- Si vous utilisez WebSphere Integration Developer, la date de début de validité correspond à la date de modélisation de la tâche utilisateur ou du processus métier.

- Si vous utilisez le déploiement de service, la date de début de validité correspond à la date d'exécution de la commande `serviceDeploy`. Seules les tâches collaboratives affichent la date d'installation de l'application comme date de début de validité.

Tâches associées

Installation des applications de tâche utilisateur et de processus métier
Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Installation d'applications de processus métier et de tâche utilisateur en mode interactif

Vous pouvez installer une application en mode interactif lors son exécution à l'aide de l'outil `wsadmin` et du script `installInteractive`. Vous pouvez utiliser le script pour modifier les paramètres qui ne sont pas modifiables si vous utilisez la console d'administration pour installer l'application.

A propos de cette tâche

Procédez comme suit pour installer des applications de processus métier en mode interactif.

Procédure

1. Démarrez l'outil `wsadmin`.

Dans le répertoire `racine_profil/bin`, entrez `wsadmin`.

2. Installez l'application.

Dans l'invite de ligne de commande, entrez la commande suivante :

```
$AdminApp installInteractive application.ear
```

où *application.ear* désigne le nom qualifié du fichier EAR (Enterprise Archive) contenant votre application de processus. Une série de tâches vous permet de modifier les valeurs définies pour l'application.

3. Sauvegardez les modifications apportées à la configuration.

Dans l'invite de ligne de commande, entrez la commande suivante :

```
$AdminConfig save
```

Vous devez sauvegarder vos modifications afin de transférer les mises à jour au référentiel de configuration maître. Si un processus de scriptage se termine et que vous n'avez pas sauvegardé vos modifications, celles-ci sont supprimées.

Tâches associées

Installation des applications de tâche utilisateur et de processus métier
Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Configuration de la source de données d'une application de processus et des paramètres de référence d'ensemble

Il peut être nécessaire de configurer les applications de processus exécutant des instructions SQL pour une infrastructure de base de données spécifique. Ces instructions SQL peuvent être issues d'activités de service d'information ou peuvent correspondre à des instructions exécutées lors du processus d'installation ou du démarrage d'une instance.

Configuration de la source de données d'une application de processus et des paramètres de référence d'ensemble

Il peut être nécessaire de configurer les applications de processus exécutant des instructions SQL pour une infrastructure de base de données spécifique. Ces instructions SQL peuvent être issues d'activités de service d'information ou peuvent correspondre à des instructions exécutées lors du processus d'installation ou du démarrage d'une instance.

A propos de cette tâche

Lorsque vous installez l'application, vous pouvez spécifier les types de sources de données suivants :

- Sources de données pour l'exécution d'instructions SQL lors de l'installation du processus
- Sources de données pour l'exécution d'instructions SQL lors du démarrage d'une instance de processus
- Sources de données pour l'exécution d'activités de fragments SQL

La source de données requise pour exécuter une activité de fragments SQL est définie dans une variable BPEL de type `tDataSource`. Le schéma de base de données et les noms de table requis pour une activité de fragments SQL sont définis dans des variables BPEL de type `tSetReference`. Vous pouvez configurer les valeurs initiales de ces deux variables.

Vous pouvez spécifier les sources de données à l'aide de l'outil `wsadmin`.

Procédure

1. Installez l'application de processus de manière interactive à l'aide de l'outil `wsadmin`.
2. Parcourez les tâches jusqu'à atteindre celles permettant de mettre à jour des sources de données et des références d'ensemble.

Configurez ces paramètres pour votre environnement. L'exemple suivant présente les paramètres que vous pouvez modifier pour chacune de ces tâches.

3. Enregistrez vos modifications.

Exemple : Mise à jour de sources de données et des références d'ensemble à l'aide de l'outil `wsadmin`

Dans la tâche **Mise à jour des sources de données**, vous pouvez modifier les valeurs des sources de données par des valeurs de variables initiales utilisées lors

de l'installation du processus ou au démarrage de ce dernier. Dans la tâche **Mise à jour des références d'ensemble**, vous pouvez configurer les paramètres liés au schéma de base de données et aux noms de table.

Task[24] : Mise à jour des sources de données

```
//Modifiez les valeurs des sources de données pour les variables initiales lors du
démarrage du processus
```

```
Nom du processus : Test
// Nom du modèle de processus
Démarrage du processus ou heure d'installation : Process start
// Indique si la valeur spécifiée est évaluée
//lors du démarrage ou de l'installation du processus
Instruction ou variable : Variable
// Indique qu'une variable de source de données doit être modifiée
Nom de la source de données : MyDataSource
// Nom de la variable
Nom JNDI :[jdbc/sample] :jdbc/newName
// Définit le nom JNDI sur jdbc/newName
```

Task[25]: Mise à jour des références d'ensemble

```
// Modifiez les valeurs des références d'ensemble utilisées en tant que valeurs
initiales pour les variables BPEL
```

```
Nom du processus : Test
// Nom du modèle de processus
Variable : SetRef
// Nom de la variable BPEL
Nom JNDI :[jdbc/sample] :jdbc/newName
// Définit le nom JNDI de la source de données de référence de l'ensemble sur
jdbc/newName
Nom du schéma : [IISAMPLE]
// Nom du schéma de la base de données
Préfixe de schéma : [] :
// Préfixe du nom du schéma.
// Ce paramètre s'applique uniquement si le nom du schéma est généré.
Nom de table : [SETREFTAB] : NEWTABLE
// Définit le nom de la table de base de données sur NEWTABLE
Préfixe de table : [] :
// Préfixe du nom de table.
// Ce paramètre s'applique uniquement si le nom de la table est généré.
```

Tâches associées

Installation d'applications de processus métier et de tâche utilisateur en mode interactif

Vous pouvez installer une application en mode interactif lors son exécution à l'aide de l'outil wsadmin et du script installInteractive. Vous pouvez utiliser le script pour modifier les paramètres qui ne sont pas modifiables si vous utilisez la console d'administration pour installer l'application.

Désinstallation d'applications de processus métier et de tâche utilisateur à l'aide de la console d'administration

Vous pouvez utiliser la console d'administration pour désinstaller des applications contenant des processus métier ou des tâches utilisateur.

Avant de commencer

Pour désinstaller une application contenant des processus métier ou des tâches humaines, assurez-vous que les conditions suivantes sont remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être démarré et avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un cluster, le gestionnaire de déploiement et au moins un membre du cluster doivent être en cours d'exécution. Le membre de cluster doit avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un serveur géré, le gestionnaire de déploiement et ce serveur doivent être en cours d'exécution. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Il n'existe pas d'instance de modèle de processus métier ou de tâche utilisateur.

A propos de cette tâche

Pour désinstaller une application d'entreprise contenant des processus métier ou des tâches utilisateur, effectuez les opérations suivantes :

Procédure

1. Cliquez sur **Applications** → **Applications d'entreprise** dans le panneau de navigation de la console d'administration.
2. Sélectionnez l'application à désinstaller et cliquez sur **Arrêter**.

Cette étape échoue si des instances de processus ou de tâche existent toujours dans l'application. Vous pouvez soit utiliser Business Process Choreographer Explorer pour supprimer les instances, soit utiliser l'option **-force** décrite à la section «Désinstallation d'applications de processus métier et de tâches humaines à l'aide d'une commande d'administration».

3. Sélectionnez l'application à désinstaller et cliquez sur **Désinstaller**.
4. Cliquez sur **Sauvegarder** pour enregistrer les modifications.

Résultats

L'application est désinstallée.

Tâches associées

Installation des applications de tâche utilisateur et de processus métier
 Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Désinstallation d'applications de processus métier et de tâches humaines à l'aide d'une commande d'administration

L'utilisation du script `bpcTemplates.jacl` est une alternative à l'emploi de la console d'administration pour désinstaller des applications qui contiennent des processus métier ou des tâches humaines.

Avant de commencer

Pour désinstaller une application contenant des processus métier ou des tâches humaines, assurez-vous que les conditions suivantes sont remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être démarré et avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un cluster, le gestionnaire de déploiement et au moins un membre du cluster doivent être en cours d'exécution. Le membre de cluster doit avoir accès à la base de données de Business Process Choreographer.

- Si l'application est installée sur un serveur géré, le gestionnaire de déploiement et ce serveur doivent être en cours d'exécution. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Aucune instance de modèle de processus métier ou de tâche humaine, quel que soit son état, n'est présente, sauf si vous comptez utiliser l'option **-force**.
- Si vous souhaitez utiliser l'option **-force** et que la sécurité administrative est activée, vérifiez que votre ID utilisateur dispose de droits d'administrateur ou d'opérateur.
- Assurez-vous que le processus serveur auquel le client d'administration se connecte est en cours d'exécution. Pour vous assurer que le client d'administration se connecte automatiquement au processus serveur, ne spécifiez pas l'option `-conntype NONE` en tant qu'option de commande.

A propos de cette tâche

Les étapes suivantes expliquent comment utiliser le script `bpcTemplates.jacl` pour désinstaller des applications contenant des modèles de processus métier ou de tâche utilisateur.

Procédure

1. Si des instances de processus ou de tâche sont encore associées aux modèles contenus dans l'application que vous comptez désinstaller, supprimez-les en appliquant l'une des procédures suivantes (ou les deux) :
 - Utilisez Business Process Choreographer Explorer pour supprimer les instances.
 - Utilisez l'option **-force** pour supprimer les instances associées aux modèles, arrêter les modèles et les désinstaller, le tout en une seule opération. Utilisez cette option avec précaution car elle supprime également toutes les données associées aux instances en cours d'exécution.

2. Passez dans le répertoire des scripts d'administration de Business Process Choreographer.

Sous Windows, entrez :

```
cd racine_installation\ProcessChoreographer\admin
```

Sous Linux, UNIX et i5/OS, entrez :

```
cd racine_installation/ProcessChoreographer/admin
```

3. Arrêtez les modèles et désinstallez l'application correspondante.

Sous Windows, entrez :

```
racine_installation\bin\wsadmin -f bpcTemplates.jacl
                                [-user nom_utilisateur]
                                [-password mot_de_passe_utilisateur]
                                -uninstall nom_application
                                [-force]
```

Sous Linux, UNIX et i5/OS, entrez :

```
racine_installation/bin/wsadmin -f bpcTemplates.jacl
                                [-user nom_utilisateur]
                                [-password mot_de_passe_utilisateur]
                                -uninstall nom_application
                                [-force]
```

Où :

nom_utilisateur

Si la sécurité administrative est activée, indiquez l'ID utilisateur à utiliser pour l'authentification.

mot_de_passe_utilisateur

Si la sécurité administrative est activée, indiquez le mot de passe de l'ID utilisateur à utiliser pour l'authentification.

nom_application

Nom de l'application à désinstaller.

-force

Entraîne l'arrêt et la suppression des instances en cours d'exécution avant que l'application ne soit désinstallée. Utilisez cette option avec précaution car elle supprime également toutes les données associées aux instances en cours d'exécution.

Résultats

L'application est désinstallée.

Tâches associées

Installation des applications de tâche utilisateur et de processus métier

Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Chapitre 8. Adaptateurs et installation

Les adaptateurs permettent à votre application de communiquer avec d'autres composants du système d'information d'entreprise.

La procédure d'installation des adaptateurs est décrite dans la rubrique Configuration et utilisation des adaptateurs du centre de documentation de WebSphere Integration Developer.

Chapitre 9. Identification et résolution des incidents lors d'un échec de déploiement

Ce chapitre décrit les étapes nécessaires afin de déterminer la cause d'un problème survenu lors du déploiement d'une application. Il présente également des solutions possibles.

Avant de commencer

Cette rubrique suppose que les conditions suivantes sont remplies :

- Vous comprenez les principes de base du débogage d'un module.
- Les fonctions de journalisation et de trace sont actives pendant le déploiement du module.

A propos de cette tâche

La tâche de résolution des incidents de déploiement commence lorsque vous recevez une notification d'erreur. Lors d'un échec de déploiement, il existe divers symptômes que vous devez inspecter avant d'agir.

Procédure

1. Déterminez si l'installation de l'application a échoué.

Cherchez dans le fichier SystemOut.log des messages qui indiquent la cause de l'échec. Les raisons de l'échec de l'installation d'une application peuvent être notamment les suivantes :

- Vous essayez d'installer une application sur plusieurs serveurs dans la même cellule Network Deployment.
- Une application possède le même nom qu'un module existant de la cellule Network Deployment dans laquelle vous installez l'application.
- Vous essayez de déployer des modules J2EE dans un fichier EAR sur différents serveurs cible.

Important : Si l'installation a échoué et que l'application contient des services, vous devez supprimer toutes les destinations SIBus ou les spécifications d'activation J2C créées avant l'échec et avant la tentative de réinstallation de l'application. Le moyen le plus simple de supprimer ces artefacts est de cliquer sur **Sauvegarder -> Annuler tout** après l'échec. Si vous enregistrez par inadvertance les modifications, vous devez supprimer manuellement les destinations SIBus destinations et les spécifications d'activation J2C (voir les rubriques concernant la suppression des destinations SIBusand et les spécifications d'activation J2C, à la section Administration).

2. Si l'application est installée correctement, examinez-la pour déterminer si elle a été démarrée avec succès.

Si le démarrage de l'application a échoué, l'échec s'est produit lorsque le serveur a tenté d'initier les ressources de l'application.

- a. Cherchez dans le fichier SystemOut.log des messages qui vous indiquent comment continuer.
- b. Déterminez si les ressources requises par l'application sont disponibles et/ou si leur démarrage a réussi.

Les ressources qui n'ont pas démarré empêchent une application de s'exécuter. Cela empêche la perte d'informations. Les raisons pour lesquelles une ressource ne démarre pas incluent :

- Les liaisons sont spécifiées de manière incorrecte
- Les ressources sont configurées de manière incorrecte
- Les ressources ne se trouvent pas dans le fichier RAR (fichier archive de ressources)
- Des ressources Web ne se trouvent pas dans le fichier WAR (fichier archive de services Web)

c. Déterminez si des composants sont manquants.

La raison de l'absence d'un composant est un fichier EAR mal compilé. Assurez-vous que tous les composants requis par le module se trouvent dans les dossiers appropriés du système test sur lequel vous avez compilé le fichier JAR (archive Java). «Préparation du déploiement sur un serveur» contient des informations supplémentaires.

3. Regardez si des informations circulent dans l'application.

Même une application en cours d'exécution peut rencontrer un échec lors du traitement des informations. Les raisons de ce problème sont similaires à celles qui sont mentionnées à l'étape 2b, à la page 481.

- a. Déterminez si l'application utilise des services contenus dans une autre application. Vérifiez que l'autre application est installée et a démarré avec succès.
- b. Déterminez si les liaisons d'importation et d'exportation de tous les services contenus dans d'autres applications utilisées par l'application défaillante sont configurées correctement. Utilisez la console d'administration pour examiner et corriger les liaisons.

4. Corrigez le problème et relancez l'application.

Tâches associées

Suppression des spécifications d'activation J2C

Le système génère des spécifications d'application J2C lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.

Suppression des destinations SIBus

Les destinations de bus d'intégration de services (SIBus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

Suppression des spécifications d'activation J2C

Le système génère des spécifications d'application J2C lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.

Avant de commencer

Si vous supprimez la spécification en raison de l'échec de l'installation d'une application, assurez-vous que le nom JNDI (Java Naming and Directory Interface) du module correspond au nom du module dont l'installation a échoué. La seconde partie du nom JNDI correspond au nom du module qui a implémenté la destination. Par exemple, dans `sca/SimpleBOCrsmA/ActivationSpec`, `SimpleBOCrsmA` correspond au nom du module.

Rôle de sécurité requis pour cette tâche : Lorsque la sécurité et les autorisations par rôle sont activées, vous devez être connecté en tant qu'administrateur ou configurateur pour exécuter cette tâche.

A propos de cette tâche

Supprimez les spécifications d'activation J2C lorsque vous enregistrez par mégarde une configuration après avoir installé une application qui contient des services et ne nécessite aucune spécification.

Procédure

1. Localisez la spécification d'activation à supprimer.
Les spécifications sont contenues dans le panneau relatif aux adaptateurs de ressources. Accédez à ce panneau en cliquant sur **Ressources > Adaptateurs de ressources**.
 - a. Localisez l'**adaptateur de ressources SPI du composant de messagerie de plateforme**.
Pour cela, vous devez vous placer au niveau du **noeud** pour un serveur autonome ou au niveau du **serveur** pour un environnement de déploiement.
2. Affichez les spécifications d'activation J2C associées à l'adaptateur de ressources SPI du composant de messagerie de plateforme.
Cliquez sur le nom de l'adaptateur de ressources, un panneau répertoriant les spécifications associées s'affiche.
3. Supprimez toutes les spécifications dont le **Nom JNDI** correspond à celui du module que vous avez supprimé.
 - a. Cochez la case située en regard de chacune des spécifications concernées.
 - b. Cliquez sur **Supprimer**.

Résultats

Le système supprime les spécifications sélectionnées de l'écran d'affichage.

Que faire ensuite

Sauvegardez les modifications.

Tâches associées

Identification et résolution des incidents lors d'un échec de déploiement

Ce chapitre décrit les étapes nécessaires afin de déterminer la cause d'un problème survenu lors du déploiement d'une application. Il présente également des solutions possibles.

Suppression des destinations SIBus

Les destinations de bus d'intégration de services (SIBus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

Suppression des destinations SIBus

Les destinations de bus d'intégration de services (SIBus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

Avant de commencer

Si vous supprimez la destination en raison de l'échec de l'installation d'une application, assurez-vous que le nom du module de la destination correspond au nom du module dont l'installation a échoué. La seconde partie du nom de la destination correspond au nom du module qui a implémenté la destination. Par exemple, dans `sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/` Customer, **SimpleBOCrsmA** correspond au nom du module.

Rôle de sécurité requis pour cette tâche : Lorsque la sécurité et les autorisations par rôle sont activées, vous devez être connecté en tant qu'administrateur ou configurateur pour exécuter cette tâche.

A propos de cette tâche

Supprimez les destinations SIBus lorsque vous enregistrez par mégarde une configuration après avoir installé une application qui contient des services et n'avez plus besoin des destinations.

Remarque : Cette tâche supprime la destination du bus système SCA uniquement. Vous devez également supprimer les entrées du bus d'application avant de réinstaller une application qui contient des services (voir la rubrique Suppression des spécifications d'activation J2C dans la section relative à l'administration de ce centre de documentation).

Procédure

1. Connectez-vous à la console d'administration.
2. Affichez les destinations sur le bus système SCA.
 - a. Dans la sous-fenêtre de navigation, cliquez sur **Intégration de service** → **bus**
 - b. Dans la sous-fenêtre de contenu, cliquez sur **SCA.SYSTEM.nom_cellule.Bus**
 - c. Dans Ressources de destination, cliquez sur **Destinations**
3. Cochez la case en regard de chaque destination associée à un nom du module correspondant au module en cours de suppression.
4. Cliquez sur **Supprimer**.

Résultats

Le panneau affiche uniquement les destinations restantes.

Que faire ensuite

Supprimez les spécifications d'activation J2C associées au module qui a créé ces destinations.

Tâches associées

Identification et résolution des incidents lors d'un échec de déploiement
Ce chapitre décrit les étapes nécessaires afin de déterminer la cause d'un problème survenu lors du déploiement d'une application. Il présente également des solutions possibles.

Suppression des spécifications d'activation J2C

Le système génère des spécifications d'application J2C lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.

Partie 3. Annexes

Remarques

Ces informations concernent initialement des produits et services fournis aux Etats-Unis.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Contactez votre représentant IBM local pour plus d'informations sur les produits et services actuellement disponibles dans votre pays. Toute référence à un produit, programme ou service IBM n'implique pas que seul ce produit, programme ou service IBM puisse être utilisé. Tout autre produit, programme ou service fonctionnellement équivalent peut être utilisé s'il n'enfreint aucun droit de propriété intellectuelle d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour les demandes relatives aux licences concernant les produits utilisant un jeu de caractères double octet, prenez contact avec le service IBM Intellectual Property Department de votre pays ou envoyez vos questions par écrit à :

*IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan*

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE NON-CONTREFACON ET D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils

contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
1001 Hillsdale Blvd., Suite 400
Foster City, CA 94404
Etats-Unis

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux dispositions de l'ICA, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Toutes données de performance contenues dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut pas confirmer avec exactitude les performances, la compatibilité ou toutes autres déclarations relatives aux produits non fournis par IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

LICENCE DE COPYRIGHT :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits. Ces programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir la fiabilité, la maintenabilité ou le fonctionnement de ces programmes.

Toute copie totale ou partielle de ces programmes exemples et des oeuvres qui en sont dérivées doit comprendre une notice de copyright, libellée comme suit : (c) (votre société) (année). Des segments de codes sont dérivés des Programmes exemples d'IBM Corp. (c) Copyright IBM Corp. _entrez l'année ou les années_. All rights reserved.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Informations relatives à l'interface de programmation

Si elle est fournie, la documentation sur l'interface de programmation aide les utilisateurs à créer des applications en utilisant le produit.

Les interfaces de programmation génériques permettent aux utilisateurs d'écrire des applications, qui bénéficient des services proposés par les outils du produit.

Cependant, cette documentation peut également comporter des informations de diagnostic, de modification et de personnalisation. Les informations de diagnostic, de modification et de personnalisation sont fournies à des fins de débogage de vos applications.

Avertissement : N'utilisez pas les informations de diagnostic, de modification et d'optimisation en guise d'interface de programmation car elles peuvent être modifiées sans préavis.

Marques et marques de service

IBM, le logo IBM et [ibm.com](http://www.ibm.com) sont des marques d'International Business Machines aux Etats-Unis et/ou dans certains autres pays. Si ces marques et d'autres marques d'IBM sont accompagnées d'un symbole de marque (^R ou TM), ces symboles signalent des marques d'IBM aux Etats-Unis à la date de publication de ce document. Ces marques peuvent aussi être des marques déposées ou reconnues comme telles par le droit coutumier sur les marques dans d'autres pays. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web "Copyright and trademark information" à www.ibm.com/legal/copytrade.shtml.

Java est une marque de Sun Microsystems, Inc. aux Etats-Unis et/ou dans certains autres pays.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Ce produit inclut un logiciel développé par Eclipse Project (<http://www.eclipse.org>).



IBM WebSphere Process Server for Multiplatforms, version 6.2

