WebSphere® Process Server for Multiplatforms

Version 6.1.0

Développement et déploiement de modules

WebSphere® Process Server for Multiplatforms

Version 6.1.0

Développement et déploiement de modules

Important
Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section Remarques à la fin de ce document.

édition - mars 2008

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

Table des matières

Figures

Tableaux

Avis aux lecteurs canadiens

Le présent document a été traduit en France. Voici les principales différences et particularités dont vous devez tenir compte.

Illustrations

Les illustrations sont fournies à titre d'exemple. Certaines peuvent contenir des données propres à la France.

Terminologie

La terminologie des titres IBM peut différer d'un pays à l'autre. Reportez-vous au tableau ci-dessous, au besoin.

IBM France	IBM Canada
ingénieur commercial	représentant
agence commerciale	succursale
ingénieur technico-commercial	informaticien
inspecteur	technicien du matériel

Claviers

Les lettres sont disposées différemment : le clavier français est de type AZERTY, et le clavier français-canadien de type QWERTY.

OS/2 et Windows - Paramètres canadiens

Au Canada, on utilise :

- les pages de codes 850 (multilingue) et 863 (français-canadien),
- le code pays 002,
- le code clavier CF.

Nomenclature

Les touches présentées dans le tableau d'équivalence suivant sont libellées différemment selon qu'il s'agit du clavier de la France, du clavier du Canada ou du clavier des États-Unis. Reportez-vous à ce tableau pour faire correspondre les touches françaises figurant dans le présent document aux touches de votre clavier.

France	Canada	Etats-Unis
(Pos1)		Home
Fin	Fin	End
(PgAr)		PgUp
(PgAv)		PgDn
Inser	Inser	Ins
Suppr	Suppr	Del
Echap	Echap	Esc
Attn	Intrp	Break
Impr écran	ImpEc	PrtSc
Verr num	Num	Num Lock
Arrêt défil	Défil	Scroll Lock
(Verr maj)	FixMaj	Caps Lock
AltGr	AltCar	Alt (à droite)

Brevets

Il est possible qu'IBM détienne des brevets ou qu'elle ait déposé des demandes de brevets portant sur certains sujets abordés dans ce document. Le fait qu'IBM vous fournisse le présent document ne signifie pas qu'elle vous accorde un permis d'utilisation de ces brevets. Vous pouvez envoyer, par écrit, vos demandes de renseignements relatives aux permis d'utilisation au directeur général des relations commerciales d'IBM, 3600 Steeles Avenue East, Markham, Ontario, L3R 9Z7.

Assistance téléphonique

Si vous avez besoin d'assistance ou si vous voulez commander du matériel, des logiciels et des publications IBM, contactez IBM direct au 1 800 465-1234.

Partie 1. Développement d'applications

Chapitre 1. Présentation du développement de modules

Un module est une unité de déploiement de base pour une application de base pour une application serveur WebSphere Process Server. Un module contient une ou plusieurs bibliothèques de composants et des modules de transfert utilisés par l'application. Un composant peut faire référence à d'autres composants de service. 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 deux types de modules de service : les modules de services métier et les modules de médiation. Un module de services métier implémente la logique d'un processus. 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.

Les section suivantes décrivent l'implémentation et la mise à jour des module sur WebSphere Process Server.

Synopsis des composants

Un composant est le bloc de construction de base qui permet d'encapsuler la logique métier réutilisable. Un composant de service est associé 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 de service 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 des interfaces pour les composants du module ;
2. définir, modifier ou manipuler les objets métier utilisés par les composants de service ;
3. définir ou modifier les composants de service via leurs interfaces ;

   Remarque : Un composant de service est défini par le biais de son interface.
4. exporter ou importer des composants de service, le cas échéant ;
5. créer un fichier EAR permettant d'installer un module qui utilise des composants. Vous créez le fichier à l'aide de la fonction EAR d'exportation dans WebSphere Integration Developer ou de la commande serviceDeploy afin de créer un fichier EAR permettant d'installer un module qui utilise des composants de service.

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.

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 leurs composants cible, poursuivez le traitement avec Appel de composants.
3. Reliez les composants client aux composants cible par le biais de connexions.

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. Définissez 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. Définissez 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. Développez la classe qui définit l'implémentation.
   - S'il s'agit d'un composant de longue durée (ou asynchrone), passez à l'étape 4.
   - S'il ne s'agit pas d'un composant de longue durée (ou synchrone), passez à l'étape 5.
4. Développez une implémentation asynchrone.

   Important : Une interface de composant asynchrone ne peut pas avoir de propriété joinsTransaction définie sur `true`.

   a. Définissez l'interface qui représente le composant de service synchrone.
   b. Définissez l'implémentation du composant de service.
   c. Passez à l'étape 6.
5. Développez une implémentation synchrone.
   a. Définissez l'interface qui représente le composant de service synchrone.
   b. Définissez l'implémentation du composant de service.
6. Enregistrez les interfaces et les implémentations du composant dans des fichiers dotés d'une extension .java.
7. Regroupez 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 7 à 9.
8. Exécutez la commande serviceDeploy pour créer un fichier EAR installable contenant l'application.
9. Installez l'application sur le noeud du serveur.
10. Facultatif : Configurez 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 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;
 }
}
```

Cet exemple développe un composant de service asynchrone. La première section de code définit l'interface du composant de service qui implémente une méthode appelée getQuote.

```
public interface StockQuote {

 public float getQuote(String symbol);
}
```

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;
 }
}
```

La section de code suivante implémente l'interface asynchrone StockQuoteAsync.

```
public interface StockQuoteAsync {

 // réponse différée
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);

 // rappel
 public Ticket getQuoteAsync(String symbol, StockQuoteCallback callback);
}
```

This section is the interface, StockQuoteCallback, which defines the onGetQuoteResponse method.

```
public interface StockQuoteCallback {

 public void onGetQuoteResponse(Ticket ticket, float quote);
}
```

Que faire ensuite

Appelez le service.

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 un 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 apportez 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 connexes.

Procédure

1. Déterminez 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. Définissez 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. Localisez le composant.
   a. Utilisez la classe ServiceManager afin d'obtenir les références disponibles pour le module appelant.
   b. Utilisez 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. Appelez le composant de manière synchrone ou asynchrone.

   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. Traitez 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;

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

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

         // appeler
       StockQuoteAsync sQuote =
      (StockQuoteAsync)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuoteAsync(customer.getSymbol());
    // ... faire autre chose ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // affecter
          value = quote * customer.getNumShares();
     } else {

         // émettre
         throw new MyValueException(customer.getErrorMsg());
     }
```

```
     // répondre
         return value;
 }
}
```

Que faire ensuite

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

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/wsdl/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);
```

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.

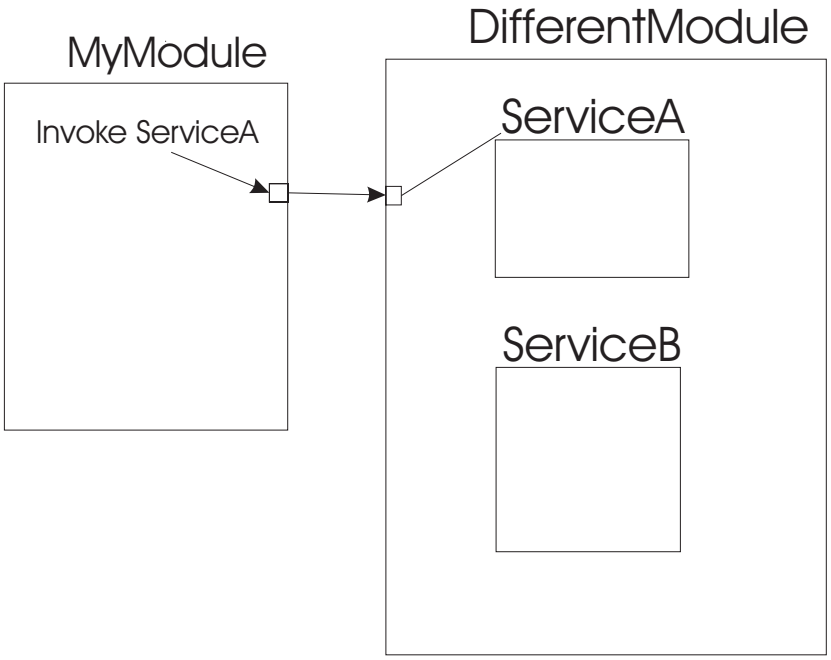


Figure 1. Modèle d'appel simple

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 influer 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.

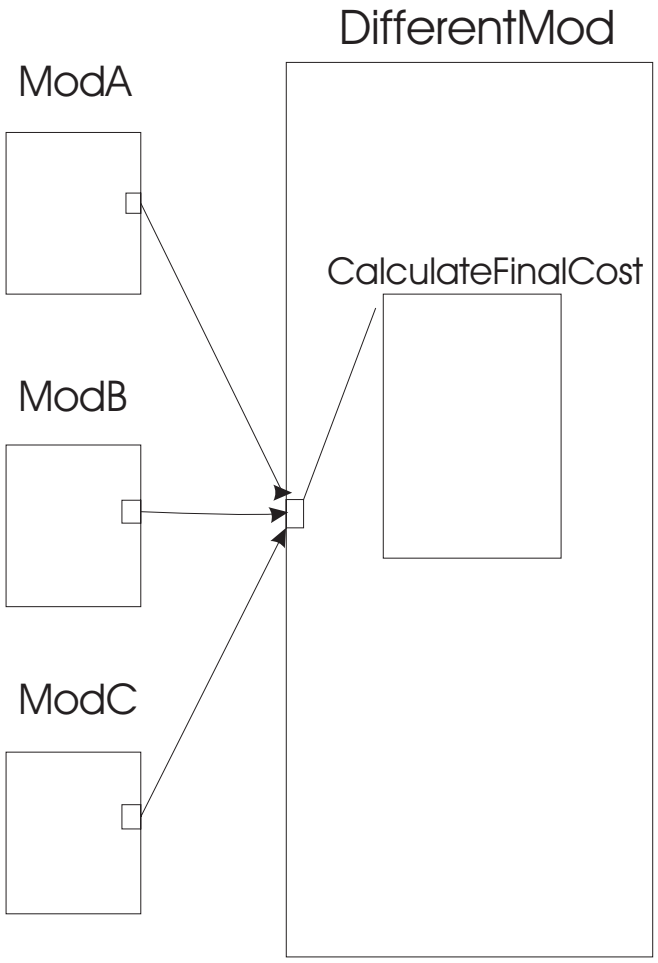


Figure 2. Appel de service unique par des applications multiples

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).

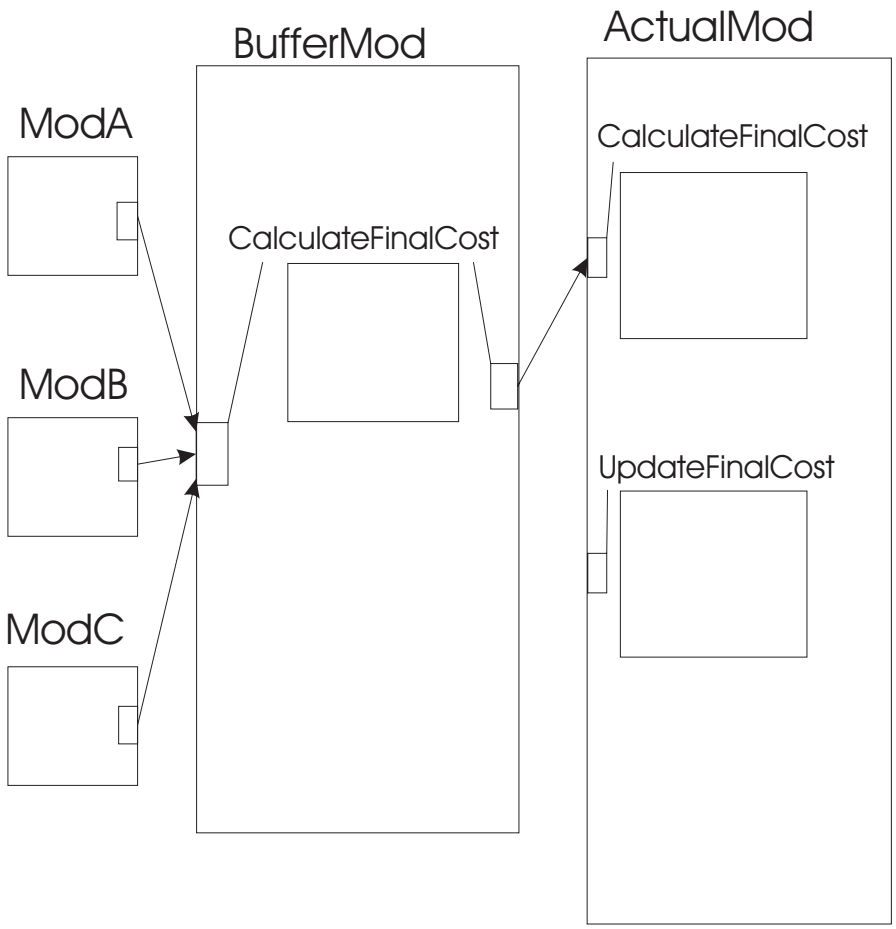


Figure 3. Modèle d'appel isolé du service UpdateCalculateFinal

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).

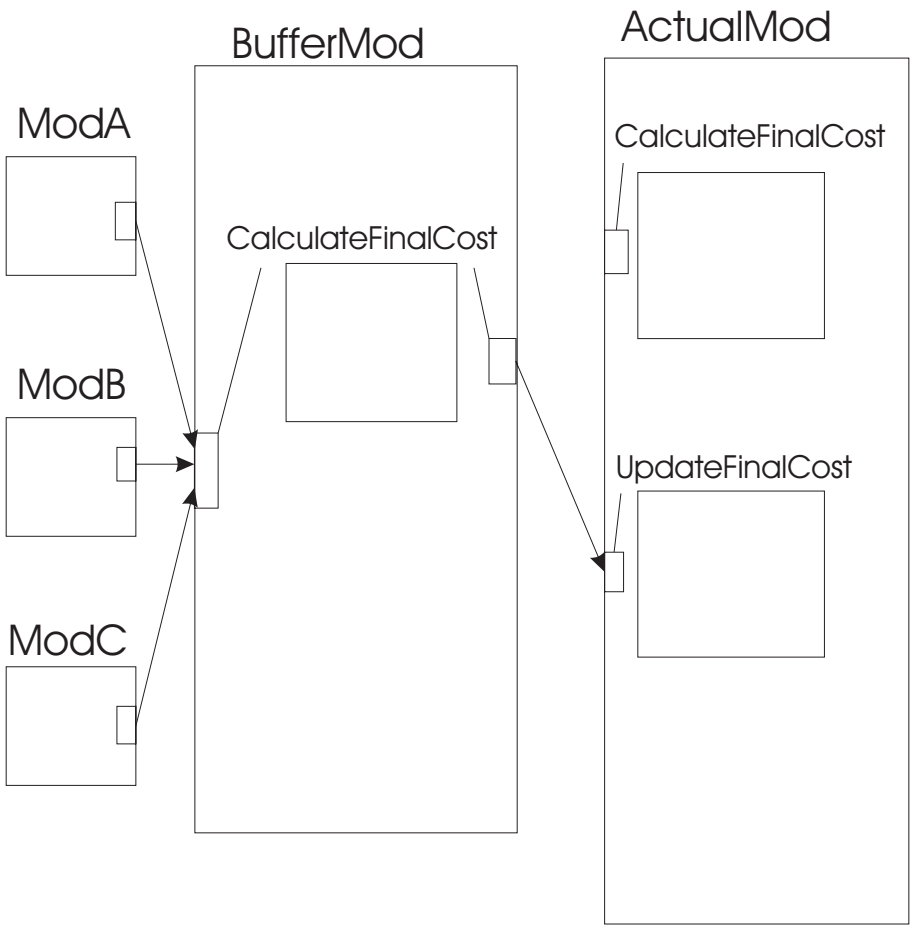


Figure 4. Modèle d'appel isolé du service UpdatedCalculateFinal

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

Changement de cibles

Le changement de la cible d'une référence permet aux applications de bénéficier des avancements effectués au niveau des composants au fur et à mesure sans que vous ayez à recompiler ni réinstaller l'application.

Liaisons HTTP

La liaison HTTP permet de relier une architecture SOA à HTTP. Cela permet d'intégrer les applications existantes ou récemment développées à des environnements 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.

Des instructions détaillées sur la création des importations et des exportations HTTP sont disponibles dans le centre de documentation dans **WebSphereIntegration Developer** > **Developing integration applications (Développement d'applications d'intégration)** > **HTTP data binding (liaison de données HTTP)**.

Tâches associées

Affichage des liaisons HTTP

Après avoir déployé une application, vous souhaiterez peut-être examiner les liaisons HTTP pour vérifier qu'elles sont correctes.

Modification des liaisons d'exportation HTTP

La console d'administration vous permet de modifier la configuration des liaisons d'exportation HTTP sans devoir modifier le code source d'origine puis redéployer l'application.

Modification des liaisons d'importation HTTP

La console d'administration vous permet de modifier la configuration des liaisons d'importation HTTP sans devoir modifier le code source d'origine puis redéployer l'application.

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 de classe d'architecture SCA 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_class*Mapper.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.

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);


}
```

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

«Règles de la conversion de Java en objets SDO durant l'exécution», à la page 19

Pour remplacer le code généré de façon appropriée, ou pour déterminer les éventuelles exceptions relatives à la conversion de Java en objets SDO durant l'exécution, il est important de connaître les règles correspondantes. Les conversions sont en général simples, mais dans certains cas complexes, la conversion de code généré en phase d'exécution permet d'obtenir le meilleur résultat.

Référence associée

Conversion Java vers XML

Le système génère un langage XML basé sur des types Java à l'aide de règles prédéfinies.

Commande genMapper

Utilisez la commande genMapper pour générer un composant qui fasse office de passerelle entre une référence SCA (Service Component Architecture) et une interface Java.

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_class*Mapper.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.

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) ;

}
```

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

«Règles de la conversion de Java en objets SDO durant l'exécution»
Pour remplacer le code généré de façon appropriée, ou pour déterminer les éventuelles exceptions relatives à la conversion de Java en objets SDO durant l'exécution, il est important de connaître les règles correspondantes. Les conversions sont en général simples, mais dans certains cas complexes, la conversion de code généré en phase d'exécution permet d'obtenir le meilleur résultat.

Référence associée


Conversion Java vers XML

Le système génère un langage XML basé sur des types Java à l'aide de règles prédéfinies.


Commande genMapper

Utilisez la commande genMapper pour générer un composant qui fasse office de passerelle entre une référence SCA (Service Component Architecture) et une interface Java.

Règles de la conversion de Java en objets SDO durant l'exécution

Pour remplacer le code généré de façon appropriée, ou pour déterminer les éventuelles exceptions relatives à la conversion de Java en objets SDO durant l'exécution, il est important de connaître les règles correspondantes. Les conversions sont en général simples, mais dans certains cas complexes, la conversion de code généré en phase d'exécution permet d'obtenir le meilleur résultat.

Types et classes de base

Durant l'exécution, une conversion simple est effectuée entre les objets SDO et les types et classes Java de base. Types et classes de base :

- Char or 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 Java en objet SDO durant l'exécution, un objet de données est créé, pour lequel l'URI généré est une inversion du nom du module du type Java et le type correspond à 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. Ensuite le contenu des membres de la classe Java est analysé et les valeurs sont attribuées aux propriétés de l'objet SDO.

Lors de la conversion d'un objet SDO en type Java, le nom du module généré est l'URI inversé et le nom du type correspond au type de l'objet 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. Ensuite, les valeurs des propriétés sont extraites de l'objet SDO et attribuées aux zones de l'instance de la classe Java.

Si la classe Java est une interface définie par l'utilisateur, vous devez remplacer le code généré et fournir une classe concrète qui peut être instanciée durant l'exécution. Si durant l'exécution, la création de la classe concrète échoue, une exception est générée.

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. Si durant l'exécution, la classe concrète est détectée, celle-ci est instanciée de la même manière que les classes et les tableaux Java définis par l'utilisateur. Sinon, l'objet SDO est transféré à l'interface Java.

Même si la méthode renvoie le type java.lang.Object, la conversion en objet SDO est effectuée uniquement si la méthode renvoie un type concret. La conversion appliquée est similaire à la conversion de classes et tableaux Java en objets SDO, comme le décrit le paragraphe suivant.

Lors de la conversion d'une classe ou d'un tableau Java en objet SDO durant l'exécution, un objet de données est créé, pour lequel l'URI généré est une inversion du nom du module du type Java et le type correspond à 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. Ensuite le contenu des membres de la classe Java est analysé et les valeurs sont attribuées aux propriétés de l'objet SDO.

Dans les deux cas, si la conversion échoue, une exception est générée.

Classes de conteneur Java.util

Lors de la conversion en classe de conteneur Java concrète, telle que Vector, HashMap, HashSet, etc. la classe de conteneur appropriée est instanciée. La méthode appliquée est similaire à celle utilisée pour les classes et tableaux Java pour remplir la classe de conteneur. Si la classe Java concrète n'est pas détectée, la classe de conteneur est remplie avec l'objet SDO.

Lors de la conversion d'une classe de conteneur Java concrète en objet SDO, les schémas générés utilisés sont ceux représentés dans la section «Conversion de Java en XML.»

Interfaces Java.util

Pour certaines interfaces de conteneur du module java.util, les classes concrètes suivantes sont instanciées :

Tableau 1. Conversion de type WSDL en classe Java

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

Tâches associées

«Remplacement de l'implémentation d'architecture SCA générée», à la page 17
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 de classe d'architecture SCA par défaut par celle de votre choix.

«Remplacement d'une conversion d'objet SDO en Java», à la page 18
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.

Référence associée

Conversion Java vers XML

Le système génère un langage XML basé sur des types Java à l'aide de règles prédéfinies.

Commande genMapper

Utilisez la commande genMapper pour générer un composant qui fasse office de passerelle entre une référence SCA (Service Component Architecture) et une interface Java.

Chapitre 2. 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. Business Process Choreographer fournit des API EJB (Enterprise JavaBeans) et des interfaces pour les services Web afin de vous permettre de développer ces clients. L'API EJB est accessible via n'importe quelle application Java, y compris une autre application EJB. Les interfaces pour les services Web sont accessibles via un environnement Java ou Microsoft .Net.

Développement d'applications client EJB pour des processus métier et des tâches utilisateur

Les API EJB fournissent un ensemble de méthodes génériques permettant de développer des applications client EJB destinées à fonctionner avec les processus métier et les tâches utilisateur installés sur WebSphere Process Server.

A propos de cette tâche

Grâce à ces API EJB (Enterprise JavaBeans), vous pouvez créer des applications client pour :

- 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. Lorsqu'une instance de bean session appropriée est créée, WebSphere Application Server lui associe un contexte. 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.

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans enterprise de session sans état. Les applications de processus métier et les applications de tâche accèdent au bean enterprise 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).

Accès à l'interface distante du bean session

Une application client EJB 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.

   Si votre application fonctionne sur une machine virtuelle Java (JVM) différente de celle sur laquelle fonctionne l'application BPEContainer ou l'application TaskContainer, effectuez les opérations suivantes :

   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 **Classpath** dans le fichier manifeste du module d'application afin d'inclure le fichier JAR.

      Le module d'application peut être une application J2EE, une application Web ou une application EJB.
   d. Si vous utilisez des types de données complexes dans votre processus métier ou tâche utilisateur et que votre client ne s'exécute pas dans une application EJB ou Web, regroupez les fichiers XSD ou WSDL correspondants avec le fichier EAR de votre application.

3. Localisez l'interface home distante du bean session via l'interface Java Naming and Directory Interface (JNDI).

   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.

4. 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 permet de vérifier les autorisations liées à chaque appel, même lorsque la sécurité globale n'est pas définie. Si la sécurité globale n'est pas définie, l'ID de l'appelant principal possède la valeur `UNAUTHENTICATED`.

5. 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("jta/usertransaction");

     // Commencer une transaction
     transaction.begin();

     // Appels d'applications ...

     // En cas d'aboutissement, valider la transaction
     transaction.commit();
     ```

     Conseil : Pour éviter tout conflit de verrouillage de la base de données, n'exécutez pas en parallèle des instructions identiques à l'exemple ci-dessous :

     ```
     // Obtenir l'interface de transaction utilisateur
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("jta/usertransaction");

     transaction.begin();

     //lire l'instance d'activité
     process.getActivityInstance(aiid);
     //réclamer l'instance d'activité
     process.claim(aiid);

     transaction.commit();
     ```

     La méthode getActivityInstance et les autres opérations de lecture définissent un blocage en lecture. Dans cet exemple, une mise à niveau transforme un blocage en lecture de l'instance d'activité en un blocage de mise à jour. Si ces transactions s'exécutent en parallèle, un blocage de la base de données peut en résulter.

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);
```

Accès à l'interface locale du bean session

Une application client EJB 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 tâches 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 via l'interface Java Naming and Directory Interface (JNDI).

   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 locale du bean BusinessFlowManager

    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 permet de vérifier les autorisations liées à chaque appel, même lorsque la sécurité globale n'est pas définie. Si la sécurité globale n'est pas définie, l'ID de l'appelant principal possède la valeur `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("jta/usertransaction");

     // Commencer une transaction
     transaction.begin();

     // Appels d'applications ...

     // En cas d'aboutissement, valider la transaction
     transaction.commit();
     ```

   Conseil : Pour éviter tout blocage de la base de données, n'exécutez pas en parallèle des instructions identiques à l'exemple ci-dessous :

   ```
   // Obtenir l'interface de transaction utilisateur
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup("jta/usertransaction");

   transaction.begin();
   ```

```
//lire l'instance d'activité
process.getActivityInstance(aiid);
//réclamer l'instance d'activité
process.claim(aiid);

transaction.commit();
```

La méthode getActivityInstance et les autres opérations de lecture définissent un blocage en lecture. Dans cet exemple, une mise à niveau transforme un blocage en lecture de l'instance d'activité en un blocage de mise à jour. Si ces transactions s'exécutent en parallèle, un blocage de la base de données peut en résulter.

Exemple

Voici un exemple illustrant les étapes 2 à 4 pour une application de tâche.

```
// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

//Rechercher l'interface home locale du bean HumanTaskManager
LocalHumanTaskManagerHome taskHome =
        (LocalHumanTaskManagerHome)initialContext.lookup
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Accéder à l'interface locale du bean session
LocalHumanTaskManager task =
taskHome.create();

...
//Appeler les fonctions métier exposées par l'interface de service
task.callTask(tkiid,input);
```

Rechercher 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.

Requêtes de données sur les objets liés aux processus métier et aux tâches

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 des éléments de travail existent. La méthode queryAll peut être appelée uniquement par les utilisateurs dotés de l'un des rôles J2EE suivants : BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor ou TaskSystemMonitor. Elle renvoie les propriétés de tous les objets stockés dans la base de données.

Toutes les requêtes API sont mappées vers des requêtes SQL. La forme de la requête SQL résultant de ce mappage dépend des points suivants :

- La requête a-t-elle été invoquée par un utilisateur disposant de l'un des rôles J2EE ?
- Quels sont les objets faisant l'objet de la requête ? Des vues prédéfinies des bases de données sont disponibles pour vous permettre de rechercher les propriétés de l'objet.
- Une clause from, des conditions d'association et des conditions propres à l'utilisateur relatives au contrôle d'accès sont-elles insérées ?

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 de façon dynamique 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 la documentation Java 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.

Syntaxe de la méthode API de requête :

La syntaxe des requêtes API de Business Process Choreographer est semblable à celle des requêtes SQL. Une requête peut inclure une clause select, une clause where, une clause order-by, un paramètre skip-tuples, un paramètre threshold et un paramètre 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 2.

Objet	Syntaxe
Modèle de processus	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Modèle de tâche	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Données relatives aux processus métier et aux tâches	`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);`

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 similaire à celle de la clause SELECT de SQL ; utilisez des virgules pour séparer les parties de la clause. Chaque partie de la clause doit spécifier une colonne d'une des vues prédéfinies. Les colonnes doivent être clairement indiquées par le nom de la vue et le nom de la colonne. Les colonnes renvoyées dans l'objet QueryResultSet s'affichent 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 d'éléments de travail pour les tâches uniques qui satisfont la clause WHERE.

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 ou des prédicats de jointure à la clause API WHERE, 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'activité`ACTIVITY.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('yyyy-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'"
```

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 indiquer une liste de colonnes à partir de vues par lesquelles le résultat est trié. Ces colonnes sont décrites par le nom et la vue de la colonne. Il est recommandé d'indiquer les colonnes se trouvant dans la clause select.

La syntaxe de la clause order-by est identique à celle de la clause SQL order-by ; utilisez des virgules pour séparer les parties de la clause. `ASC` permet de trier les colonnes dans l'ordre croissant et `DESC` de 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 vous indiquez plusieurs colonnes, l'ensemble de résultats est trié par les valeurs de la première colonne, puis par les valeurs de la deuxième colonne, et ainsi de suite. Dans la clause order-by, vous ne pouvez pas indiquer les colonnes par position comme il est possible de le faire dans une clause SQL.

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 pas 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.

Paramètre skip-tuples :

Le paramètre skip-tuples spécifie le nombre de tuples dans l'ensemble de résultats de la requête, en partant du début, à ignorer et à ne pas renvoyer à l'appelant dans l'ensemble de 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.

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.

Puisque les ensembles de résultats de requête des scénarios de production peuvent contenir des milliers voire des millions d'éléments, il est recommandé de toujours définir un seuil. Le paramètre threshold peut s'avérer utile, par exemple, dans une interface utilisateur graphique où il n'est pas recommandé d'afficher un grand nombre d'éléments en même temps. 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.

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.

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')",
                 (String)null,
                 (Integer)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')",
                 (String)null, (Integer)null, (TimeZone)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).

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.

Tâches associées

«Gestion des requêtes stockées», à la page 63
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.

Les éléments de l'ensemble de résultats sont des propriétés des objets qui satisfont la clause where lancée par l'appelant et que l'appelant est autorisé à voir. Vous pouvez lire les éléments d'une manière relative à l'aide de la méthode API next 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 QueryResultSet spécifie la valeur du premier attribut spécifié dans la clause SELECT de la demande de requête. Le deuxième attribut (colonne) d'un élément QueryResultSet 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
Entier	getInteger getShort getLong getString getBoolean

Type d'attribut	Méthode
Booléen	getBoolean getShort getInteger getLong getString
bit[]	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"
```

Conditions d'accès propres à l'utilisateur :

Des conditions d'accès propres à l'utilisateur sont ajoutées lorsque l'instruction SQL SELECT est générée par la requête API. Elles permettent de ne renvoyer à l'appelant que les objets qui correspondent à la condition indiquée par l'appelant et pour laquelle il dispose d'une autorisation.

La condition d'accès ajoutée varie selon que l'utilisateur est administrateur système ou non.

Requêtes appelées par des utilisateurs qui ne sont pas administrateur système.

La clause SQL WHERE générée combine la clause API where et une condition de contrôle d'accès propre à l'utilisateur. La requête n'extrait que les objets auxquels l'utilisateur a le droit d'accéder, c'est-à-dire uniquement les objets pour lesquels il

dispose d'un élément de travail. Un élément de travail correspond à l'affectation d'un utilisateur ou d'un groupe d'utilisateur à un rôle d'autorisation relatif à un objet métier tel qu'une tâche ou un processus. Par exemple, si l'utilisateur John Smith est l'un des propriétaires potentiels d'une tâche donnée, il existe un objet de type élément de travail représentant cette relation.

Exemple : si un utilisateur qui n'est pas administrateur système exécute une requête sur des tâches, la condition d'accès suivante s'ajoute à la clause WHERE si les éléments de travail du 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 )
```

Ainsi, si John Smith souhaite obtenir une liste des tâches dont il est le propriétaire potentiel, la clause API where peut se présenter comme ceci :

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

Cette clause entraîne 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
```

Ceci signifie également que si John Smith souhaite voir les activités et les tâches pour lesquelles il est lecteur ou administrateur de processus et pour lesquelles il ne dispose pas d'un élément de travail, une propriété de la vue PROCESS_INSTANCE doit être ajoutée à la clause select, where ou order-by de la requête, par exemple PROCESS_INSTANCE.PIID.

Si les éléments de travail du groupe sont activés, une condition d'accès supplémentaire permettant à l'utilisateur d'avoir accès à des objets auxquels le groupe a accès est ajoutée à la clause WHERE.

Requêtes appelées par des administrateurs système

Les administrateurs système peuvent appeler la méthode query pour extraire des objets associés à des éléments de travail. Dans ce cas, une condition d'association à la vue WORK_ITEM s'ajoute à la requête SQL générée, mais aucune condition de contrôle d'accès n'est ajoutée pour WORK_ITEM.OWNER_ID.

Dans ce cas, la requête SQL pour les tâches contient les instructions suivantes :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
```

Requêtes queryAll

Ce type de requête peut être uniquement appelée par les administrateurs ou les contrôleurs système. Aucune condition de contrôle d'accès ou d'association à la vue WORK_ITEM n'est ajoutée. Ce type de requête renvoie toutes les données relatives à tous les objets.

Exemples des méthodes query et queryAll :

Les exemples qui suivent illustrent la syntaxe de diverses requêtes API et les instructions SQL générées lors de leur traitement.

Exemple: requêtes portant sur les tâches à l'état prêt :

Cet exemple illustre comment utiliser une méthode query pour récupérer des tâches que l'utilisateur connecté peut exploiter.

John Smith veut connaître la liste des tâches qui lui ont été affectées. Pour pouvoir être exploitée par un utilisateur, une tâche doit être à l'état prêt. L'utilisateur connecté doit également disposer d'un élément de travail de type propriétaire potentiel pour cette tâche. Le fragment de code suivant illustre l'appel de la 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 initiées lors de la génération de l'instruction SQL SELECT :

- Une condition de contrôle d'accès est ajoutée à la clause where. Cet exemple suppose que les éléments de travail de groupe ne sont pas activés.
- Les constantes telles que `TASK.STATE.STATE_READY` sont remplacées par des valeurs numériques.
- Une clause FROM et des conditions d'association sont ajoutées.

Le fragment de code suivant illustre l'instruction SQL générée par 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 à des tâches relatives à un processus particulier, par exemple sampleProcess, la requête se présente de la façon suivante :

```
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 )
```

Exemple : requêtes portant sur des tâches à l'état réclamé :

Cet exemple illustre comment utiliser une méthode query pour extraire des tâches réclamées par l'utilisateur connecté.

L'utilisateur John Smith veut rechercher les tâches qu'il a réclamées et qui sont toujours à l'état Réclamé. La condition qui signifie ″réclamé par John Smith″ est `TASK.OWNER = 'JohnSmith'`. Le fragment de code suivant illustre l'appel de la 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 illustre l'instruction SQL générée par la requête API :

```
^SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   TA.OWNER = 'JohnSmith'
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Lorsqu'une tâche est réclamée, des éléments de travail sont créés pour le propriétaire de la tâche. Une autre manière de formuler la requête de John Smith est d'ajouter la condition suivante à la requête au lieu d'indiquer simplement `TASK.OWNER = 'JohnSmith'` :

```
WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER
```

La requête se présente alors comme le 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 initiées lors de la génération de l'instruction SQL SELECT :

- Une condition de contrôle d'accès est ajoutée à la clause where. Cet exemple suppose que les éléments de travail de groupe ne sont pas activés.
- Les constantes telles que `TASK.STATE.STATE_READY` sont remplacées par des valeurs numériques.
- Une clause FROM et des conditions d'association sont ajoutées.

Le fragment de code suivant illustre l'instruction SQL générée par 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 s'apprête à partir en vacances et sa collaboratrice, Anne Grant, veut vérifier sa charge de travail du moment. Anne dispose des droits d'administration. La requête qu'elle appelle est la même que celle appelée par John. Cependant, l'instruction SQL générée est différente car Anne est administrateur. Le fragment de code suivant illustre 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')
```

Anne étant administrateur, aucune condition de contrôle d'accès n'est ajoutée à la clause WHERE.

Exemple : requêtes portant sur les escalades :

Cet exemple illustre comment utiliser la méthode query pour extraire les escalades concernant l'utilisateur connecté.

Lorsqu'une tâche fait l'objet d'une escalade, un élément de travail de type récepteur de l'escalade est créé. L'utilisatrice Marie Dupont souhaite afficher une

liste des tâches qui ont été escaladées vers elle. Le fragment de code suivant illustre l'appel de la méthode query pour cette 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 initiées lors de la génération de l'instruction SQL SELECT :

- Une condition de contrôle d'accès est ajoutée à la clause where. Cet exemple suppose que les éléments de travail de groupe ne sont pas activés.
- Les constantes telles que `TASK.STATE.STATE_READY` sont remplacées par des valeurs numériques.
- Une clause FROM et des conditions d'association sont ajoutées.

Le fragment de code suivant illustre l'instruction SQL générée par 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 = 'MarieDupont' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Exemple : utilisation de la méthode queryAll :

Cet exemple illustre comment utiliser la méthode queryAll pour extraire toutes les activités appartenant à un modèle de processus.

La méthode queryAll est disponible uniquement pour les utilisateurs disposant des droits d'un administrateur système ou d'un contrôleur système. Le fragment de code suivant illustre l'appel de la méthode queryAll pour la requête visant à extraire toutes les activités appartenant 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 illustre la requête SQL générée par la requête API :

```
SELECT DISTINCT ACTIVITY.AIID
  FROM   ACTIVITY AI, PROCESS_TEMPLATE PT
  WHERE  AI.PTID = PT.PTID
  AND    PT.NAME = 'sampleProcess'
```

L'appel étant émis par l'administrateur, aucune condition de contrôle d'accès n'est ajoutée à l'instruction SQL générée. Aucune condition d'association avec la vue WORK_ITEM n'est ajoutée. Ceci signifie que la requête extrait toutes les activités du modèle de processus, y compris les activités sans élément de travail.

Exemple : inclusion des propriétés d'une requête dans une requête :

Cet exemple illustre comment utiliser la méthode query pour extraire des tâches appartenant à un processus métier. Des propriétés de requête ont été définies pour le processus et vous souhaitez les inclure à la recherche.

Vous voulez par exemple rechercher toutes les tâches utilisateur à l'état prêt qui appartiennent à un processus métier. Ce processus métier est doté d'une propriété de requête, **customerID**, dont la valeur est CID_12345, et d'un espace de nom. Le fragment de code suivant illustre l'appel de la 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 );
```

Si vous souhaitez maintenant ajouter une seconde propriété à la requête, par exemple **Priority**, et un espace de nom donné, l'appel de la méthode query se présente de la façon suivante :

```
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 à la requête, vous devez attribuer un numéro à chacune d'entre elle, comme le montre le fragment de code ci-dessus. Les requêtes portant sur des propriétés personnalisées ont un impact sur les performances : plus le nombre de propriétés incluses dans la requête est élevé, plus les performances diminuent.

Exemple : inclusion de propriétés personnalisées dans une requête :

Cet exemple illustre comment utiliser la méthode query pour extraire des tâches dotées de propriétés personnalisées.

Vous voulez par exemple rechercher toutes les tâches utilisateur à l'état prêt dotées d'une propriété personnalisée **customerID** dont la valeur est égale à CID_12345. Le fragment de code suivant illustre l'appel de la 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 );
```

Si vous souhaitez maintenant extraire les tâches et leurs propriétés personnalisées, l'appel de la méthode query se présente de la façon suivante :

```
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 générée par cette requête API est illustrée par 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 contient une jointure externe entre la vue TASK et la vue TASK_CPROP. Ceci signifie que les tâches qui satisfont la clause WHERE sont extraites mêmes si elles ne contiennent pas de propriétés personnalisées.

Vues prédéfinies pour les requêtes de données sur les objets de processus métier et de tâches utilisateur :

Des vues prédéfinies des bases de données sont disponibles pour les objets de processus métier et de tâches utilisateur. Utilisez ces vues lors de la recherche de données de référence sur ces objets.

Lorsque vous utilisez les vues prédéfinies, il n'est pas nécessaire d'ajouter explicitement des prédicats de jointure pour les colonnes des vues ; ces constructions sont ajoutées automatiquement à votre place. Vous pouvez utiliser la fonction de requête générique du service API (BusinessFlowManagerService ou HumanTaskManagerService) pour rechercher ces données. Vous pouvez également utiliser la méthode correspondante de l'interface API HumanTaskManagerDelegate ou vos requêtes prédéfinies fournies par vos implémentations de l'interface ExecutableQuery.

Remarque : Les vues peuvent contenir des colonnes non décrites. Elles sont réservées à un usage interne.

Vue ACTIVITY :

Cette vue prédéfinie de la base de données vous permet d'effectuer des requêtes de données sur les activités.

Tableau 3. Colonnes de la vue ACTIVITY

Nom de colonne	Type	Commentaires
PIID	ID	L'ID de l'instance de processus.
AIID	ID	L'ID de l'instance d'activité.
PTID	ID	L'ID du modèle de processus.
ATID	ID	L'ID du modèle d'activité.

Tableau 3. Colonnes de la vue ACTIVITY (suite)

Nom de colonne	Type	Commentaires
KIND	Entier	Le type d'activité. Ses valeurs admises sont : KIND_INVOKE (21) KIND_RECEIVE (23) KIND_REPLY (24) KIND_THROW (25) KIND_RETHROW (46) KIND_TERMINATE (26) KIND_WAIT (27) KIND_COMPENSATE (29) KIND_SEQUENCE (30) KIND_EMPTY (3) KIND_SWITCH (32) KIND_WHILE (34) KIND_PICK (36) KIND_FLOW (38) KIND_SCOPE (40) KIND_SCRIPT (42) KIND_STAFF (43) KIND_ASSIGN (44) KIND_CUSTOM (45) KIND_FOR_EACH_PARALLEL (49) KIND_FOR_EACH_SERIAL (47)
COMPLETED	Horodatage	L'heure à laquelle l'activité s'est terminée.
ACTIVATED	Horodatage	L'heure de l'activation de l'activité.
FIRST_ACTIVATED	Horodatage	L'heure de la première activation de l'activité.
STARTED	Horodatage	L'heure du démarrage de l'activité.
STATE	Entier	L'état de l'activité. Ses valeurs admises sont : STATE_INACTIVE (1) STATE_READY (2) STATE_RUNNING (3) STATE_PROCESSING_UNDO (14) STATE_SKIPPED (4) STATE_FINISHED (5) STATE_FAILED (6) STATE_TERMINATED (7) STATE_CLAIMED (8) STATE_TERMINATING (9) STATE_FAILING (10) STATE_WAITING (11) STATE_EXPIRED (12) STATE_STOPPED (13)
OWNER	Chaîne	ID principal du propriétaire.
DESCRIPTION	Chaîne	Si la description du modèle d'activité contient des espaces réservés, cette colonne contient la description de l'instance de l'activité avec les espaces réservés résolus.
TEMPLATE_NAME	Chaîne	Nom du modèle d'activité associé.

Tableau 3. Colonnes de la vue ACTIVITY (suite)

Nom de colonne	Type	Commentaires
TEMPLATE_DESCR	Chaîne	Description du modèle d'activité associé.
BUSINESS_RELEVANCE	Booléen	Indique si l'activité est une activité professionnelle significative. Ses valeurs admises sont : **TRUE** L'activité est une activité professionnelle significative. Vous pouvez afficher l'état de l'activité dans Business Process Choreographer Explorer. **FALSE** L'activité n'est pas une activité professionnelle significative.
EXPIRES	Horodatage	Date et heure auxquelles l'activité arrive à expiration. Si celle-ci est arrivée à expiration, date et heure de cette échéance.

Vue ACTIVITY_ATTRIBUTE :

Cette vue prédéfinie de la base de données vous permet d'effectuer des requêtes de données sur les propriétés personnalisées d'activités.

Tableau 4. Colonnes de la vue ACTIVITY_ATTRIBUTE

Nom de colonne	Type	Commentaires
AIID	ID	L'ID de l'instance d'activité qui a une propriété personnalisée.
NAME	Chaîne	Le nom de la propriété personnalisée.
VALUE	Chaîne	La valeur de la propriété personnalisée.

Vue ACTIVITY_SERVICE :

Cette vue prédéfinie de la base de données vous permet de rechercher des données sur des services d'activité.

Tableau 5. Colonnes de la vue ACTIVITY_SERVICE

Nom de colonne	Type	Commentaires
EIID	ID	L'ID de l'instance d'événement.
AIID	ID	L'ID de l'instance d'activité qui attend l'événement.
PIID	ID	L'ID de l'instance de processus contenant l'événement.
VTID	ID	L'ID du modèle de service qui décrit l'événement.
PORT_TYPE	Chaîne	Le nom du type de port.
NAME_SPACE_URI	Chaîne	L'URI de l'espace nom.
OPERATION	Chaîne	Le nom d'opération du service.

Vue APPLICATION_COMP :

Cette vue prédéfinie de la base de données vous permet de rechercher des données sur les ID de composants d'applications et sur les valeurs par défaut correspondant aux tâches.

Tableau 6. Colonnes de la vue APPLICATION_COMP

Nom de colonne	Type	Commentaires
ACOID	Chaîne	L'ID du composant d'application.
BUSINESS_ RELEVANCE	Booléen	La règle par défaut concernant le degré d'importance professionnelle des tâches dans le cadre du composant. Cette valeur peut être remplacée par une définition dans le modèle de tâche ou dans la tâche. Cet attribut intervient sur la journalisation dans le journal d'audit. Ses valeurs admises sont : **TRUE** La tâche est une tâche professionnelle significative et fera l'objet d'un audit. **FALSE** La tâche n'est pas une tâche professionnelle significative et ne fera pas l'objet d'un audit.
NAME	Chaîne	Le nom du composant de l'application.
SUPPORT_ AUTOCLAIM	Booléen	La règle par défaut de réclamation automatique applicable au composant. Si cet attribut a pour valeur TRUE, la tâche peut être automatiquement réclamée au cas où le propriétaire potentiel est un utilisateur unique. Cette valeur peut être remplacée par une définition dans le modèle de tâche ou dans la tâche.
SUPPORT_CLAIM_ SUSP	Booléen	Le paramètre par défaut applicable au composant qui spécifie si les tâches mises en suspens peuvent être réclamées. Si cet attribut a pour valeur TRUE, les tâches mises en suspens peuvent être réclamées. Cette valeur peut être remplacée par une définition dans le modèle de tâche ou dans la tâche.
SUPPORT_ DELEGATION	Booléen	La règle par défaut relative à la délégation des tâches applicable au composant. Si cet attribut a pour valeur TRUE, les tâches peuvent être modifiées. Ceci signifie qu'il est possible de créer, supprimer ou transférer des éléments de travail.
SUPPORT_ FOLLOW_ON	Booléen	La règle par défaut de suivi des tâches applicable au composant. Si l'attribut est défini sur TRUE, des tâches de suivi peuvent être créées pour les tâches. Cette valeur peut être remplacée par une définition dans le modèle de tâche ou dans la tâche.

Tableau 6. Colonnes de la vue APPLICATION_COMP (suite)

Nom de colonne	Type	Commentaires
SUPPORT_ SUB_TASK	Booléen	La règle par défaut de sous-tâche applicable au composant. Si cet attribut a pour valeur TRUE, des sous-tâches peuvent être créées pour cette tâche. Cette valeur peut être remplacée par une définition dans le modèle de tâche ou dans la tâche.

Vue ESCALATION :

Cette vue prédéfinie de la base de données vous permet de rechercher des données sur les escalades.

Tableau 7. Colonnes de la vue ESCALATION

Nom de colonne	Type	Commentaires
ESIID	Chaîne	L'ID de l'instance d'escalade.
ACTION	Entier	L'action déclenchée par l'escalade. Ses valeurs admises sont : **ACTION_CREATE_WORK_ITEM (1)** Crée un élément de travail pour chaque destinataire d'escalade. **ACTION_SEND_EMAIL (2)** Envoie un courrier électronique à chaque destinataire d'escalade. **ACTION_CREATE_EVENT (3)** Crée et publie un événement.
ACTIVATION_STATE	Entier	Une instance d'escalade est créée si la tâche correspondante est dans l'un des états suivants : **ACTIVATION_STATE_READY (2)** Spécifie que la tâche utilisateur ou de participation est prête à être réclamée. **ACTIVATION_STATE_RUNNING (3)** Spécifie que la tâche d'origine a démarré et est en cours d'exécution. **ACTIVATION_STATE_CLAIMED (8)** Spécifie que la tâche a été réclamée. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** Spécifie que la tâche attend l'achèvement de sous-tâches.
ACTIVATION_TIME	Horodatage	L'heure de l'activation de l'escalade.

Tableau 7. Colonnes de la vue ESCALATION (suite)

Nom de colonne	Type	Commentaires
AT_LEAST_ EXP_STATE	Entier	L'état de la tâche attendu par l'escalade. Si un délai d'expiration est dépassé, l'état de la tâche est comparé à la valeur de cet attribut. Ses valeurs admises sont : **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** Spécifie que la tâche a été réclamée. **AT_LEAST_EXPECTED_STATE_ENDED (20)** Spécifie que la tâche est dans un état final (FINISHED, FAILED, TERMINATED ou EXPIRED). **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** Spécifie que toutes les sous-tâches de la tâche ont été effectuées.
ESTID	Chaîne	L'ID du modèle d'escalade correspondant.
FIRST_ESIID	Chaîne	L'ID de la première escalade dans la chaîne.
INCREASE_PRIORITY	Entier	Indique la manière dont la priorité de la tâche sera augmentée. Ses valeurs admises sont : **INCREASE_PRIORITY_NO (1)** N'augmente pas la priorité de la tâche. **INCREASE_PRIORITY_ONCE (2)** La priorité de la tâche est augmentée une fois de 1. **INCREASE_PRIORITY_REPEATED (3)** La priorité de la tâche est augmentée de 1 chaque fois que l'escalade est répétée.
NAME	Chaîne	Le nom de l'escalade.
STATE	Entier	L'état de l'escalade. Ses valeurs admises sont : STATE_INACTIVE (1) STATE_WAITING (2) STATE_ESCALATED (3) STATE_SUPERFLUOUS (4)
TKIID	Chaîne	L'ID d'instance de tâche à laquelle appartient l'escalade.

Vue ESCALATION_CPROP :

Utilisez cette vue prédéfinie de la base de données pour rechercher des propriétés personnalisées des escalades.

Tableau 8. Colonnes de la vue ESCALATION_CPROP

Nom de colonne	Type	Commentaires
ESIID	Chaîne	L'ID de l'escalade.
NAME	Chaîne	Nom de la propriété.
DATA_TYPE	Chaîne	Type de la classe pour les propriétés personnalisés autres que de type Chaîne.
STRING_VALUE	Chaîne	La valeur des propriétés personnalisées de type Chaîne.

Vue ESCALATION_DESC :

Utilisez cette vue prédéfinie de la base de données pour rechercher des données descriptives multilingues sur les escalades.

Tableau 9. Colonnes de la vue ESCALATION_DESC

Nom de colonne	Type	Commentaires
ESIID	Chaîne	L'ID de l'escalade.
LOCALE	Chaîne	Le nom du paramètre régional associé à la description ou au nom affiché.
DESCRIPTION	Chaîne	Une description du modèle de tâche.
DISPLAY_NAME	Chaîne	Le nom descriptif de l'escalade.

Vue ESC_TEMPL :

Cette vue prédéfinie de la base de données vous permet de rechercher des données dans les modèles d'escalade.

Tableau 10. Colonnes de la vue ESC_TEMPL

Nom de colonne	Type	Commentaires
ESTID	Chaîne	L'ID du modèle d'escalade.
ACTION	Entier	L'action déclenchée par l'escalade. Ses valeurs admises sont : **ACTION_CREATE_WORK_ITEM (1)** Crée un élément de travail pour chaque destinataire d'escalade. **ACTION_SEND_EMAIL (2)** Envoie un courrier électronique à chaque destinataire d'escalade. **ACTION_CREATE_EVENT (3)** Crée et publie un événement.

Tableau 10. Colonnes de la vue ESC_TEMPL (suite)

Nom de colonne	Type	Commentaires
ACTIVATION_STATE	Entier	Une instance d'escalade est créée si la tâche correspondante est dans l'un des états suivants : **ACTIVATION_STATE_READY (2)** Spécifie que la tâche utilisateur ou de participation est prête à être réclamée. **ACTIVATION_STATE_RUNNING (3)** Spécifie que la tâche d'origine a démarré et est en cours d'exécution. **ACTIVATION_STATE_CLAIMED (8)** Spécifie que la tâche a été réclamée. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** Spécifie que la tâche attend l'achèvement de sous-tâches.
AT_LEAST_ EXP_STATE	Entier	L'état de la tâche attendu par l'escalade. Si un délai d'expiration est dépassé, l'état de la tâche est comparé à la valeur de cet attribut. Les valeurs admises sont : **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** Spécifie que la tâche a été réclamée. **AT_LEAST_EXPECTED_STATE_ENDED (20)** Spécifie que la tâche est dans un état final (FINISHED, FAILED, TERMINATED ou EXPIRED). **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** Spécifie que toutes les sous-tâches de la tâche ont été effectuées.
CONTAINMENT_ CTX_ID	Chaîne	Si le modèle d'escalade appartient à un modèle de tâche en ligne, le contexte de confinement est le modèle de processus. Si le contexte du modèle d'escalade appartient à un modèle de tâche autonome, le contexte de contexte de confinement est le modèle de tâche.
FIRST_ESTID	Chaîne	L'ID du premier modèle d'escalade dans une chaîne de modèles d'escalade.
INCREASE_PRIORITY	Entier	Indique la manière dont la priorité de la tâche sera augmentée. Les valeurs admises sont : **INCREASE_PRIORITY_NO (1)** N'augmente pas la priorité de la tâche. **INCREASE_PRIORITY_ONCE (2)** La priorité de la tâche est augmentée une fois de 1. **INCREASE_PRIORITY_REPEATED (3)** La priorité de la tâche est augmentée de 1 chaque fois que l'escalade est répétée.
NAME	Chaîne	Le nom du modèle d'escalade.
PREVIOUS_ESTID	Chaîne	L'ID du premier modèle d'escalade dans une chaîne de modèles d'escalade.

Tableau 10. Colonnes de la vue ESC_TEMPL (suite)

Nom de colonne	Type	Commentaires
TKTID	Chaîne	L'ID du modèle de tâche auquel appartient le modèle d'escalade.

Vue ESC_TEMPL_CPROP :

Cette vue prédéfinie de la base de données vous permet de rechercher des modèles d'escalade dans les propriétés personnalisées.

Tableau 11. Colonnes de la vue ESC_TEMPL_CPROP

Nom de colonne	Type	Commentaires
ESTID	Chaîne	L'ID du modèle d'escalade.
NAME	Chaîne	Le nom de la propriété.
TKTID	Chaîne	L'ID du modèle de tâche auquel appartient le modèle d'escalade.
DATA_TYPE	Chaîne	Le type de la classe pour les propriétés personnalisés autres que de type Chaîne.
VALUE	Chaîne	La valeur des propriétés personnalisées de type Chaîne.

Vue ESC_TEMPL_DESC :

Utilisez cette vue prédéfinie de la base de données pour rechercher des modèles d'escalade dans les données descriptives multilingues

Tableau 12. Colonnes de la vue ESC_TEMPL_DESC

Nom de colonne	Type	Commentaires
ESTID	Chaîne	L'ID du modèle d'escalade.
LOCALE	Chaîne	Le nom du paramètre régional associé à la description ou au nom affiché.
TKTID	Chaîne	L'ID du modèle de tâche auquel appartient le modèle d'escalade.
DESCRIPTION	Chaîne	Une description du modèle de tâche.
DISPLAY_NAME	Chaîne	Le nom descriptif de l'escalade.

Vue PROCESS_ATTRIBUTE :

Cette vue prédéfinie de la base de données vous permet d'effectuer des requêtes de données sur les propriétés personnalisées de processus.

Tableau 13. Colonnes de la vue PROCESS_ATTRIBUTE

Nom de colonne	Type	Commentaires
PIID	ID	L'ID de l'instance de processus qui a une propriété personnalisée.
NAME	Chaîne	Le nom de la propriété personnalisée.
VALUE	Chaîne	La valeur de la propriété personnalisée.

Vue PROCESS_INSTANCE :

Cette vue prédéfinie de la base de données vous permet de rechercher des données sur les instances de processus.

Tableau 14. Colonnes de la vue PROCESS_INSTANCE

Nom de colonne	Type	Commentaires
PTID	ID	L'ID du modèle de processus.
PIID	ID	L'ID de l'instance de processus.
NAME	Chaîne	Le nom de l'instance de processus.
STATE	Entier	L'état de l'instance de processus. Ses valeurs admises sont : STATE_READY (1) STATE_RUNNING (2) STATE_FINISHED (3) STATE_COMPENSATING (4) STATE_INDOUBT (10) STATE_FAILED (5) STATE_TERMINATED (6) STATE_COMPENSATED (7) STATE_COMPENSATION_FAILED (12) STATE_TERMINATING (8) STATE_FAILING (9) STATE_SUSPENDED (11)
CREATED	Horodatage	L'heure de la création de l'instance de processus.
STARTED	Horodatage	L'heure à laquelle l'instance de processus a démarré.
COMPLETED	Horodatage	L'heure à laquelle l'instance de processus s'est terminée.
PARENT_PIID	ID	L'ID de l'instance du processus parent.
PARENT_NAME	Chaîne	Le nom de l'instance de processus parent.
TOP_LEVEL_PIID	ID	L'ID de l'instance de processus de l'instance du processus de niveau supérieur. S'il n'existe pas d'instance de processus de niveau supérieur, il s'agit de l'ID de l'instance de processus en cours.
TOP_LEVEL_NAME	Chaîne	Le nom de l'instance de processus de niveau supérieur. S'il n'existe pas d'instance de processus de niveau supérieur, il s'agit du nom de l'instance de processus en cours.
STARTER	Chaîne	L'ID principal de l'initiateur de l'instance de processus.
DESCRIPTION	Chaîne	Si la description du modèle de processus contient des espaces réservés, cette colonne contient la description de l'instance du processus avec les espaces réservés résolus.
TEMPLATE_NAME	Chaîne	Le nom du modèle de processus associé.
TEMPLATE_DESCR	Chaîne	La description du modèle de processus associé.
RESUMES	Horodatage	L'heure à laquelle le processus d'instance doit redémarrer automatiquement.

Vue PROCESS_TEMPLATE :

Cette vue prédéfinie de la base de données vous permet de rechercher des données sur les modèles de processus.

Tableau 15. Colonnes de la vue PROCESS_TEMPLATE

Nom de colonne	**Type**	**Commentaires**
PTID	ID	L'ID du modèle de processus.
NAME	Chaîne	Le nom du modèle de processus.
VALID_FROM	Horodatage	L'heure à laquelle le modèle de processus devient disponible pour être instanciée.
TARGET_NAMESPACE	Chaîne	L'espace nom cible du modèle de processus.
APPLICATION_NAME	Chaîne	Le nom de l'application d'entreprise à laquelle appartient le modèle de processus.
VERSION	Chaîne	La version définie par l'utilisateur.
CREATED	Horodatage	L'heure de la création du modèle de processus dans la base de données.
STATE	Entier	Spécifie si le modèle de processus est disponible pour la création des instances de processus. Ses valeurs admises sont : STATE_STARTED (1) STATE_STOPPED (2)
EXECUTION_MODE	Entier	Spécifie la manière dont les instances de processus dérivées de ce modèle de processus peuvent être exécutées. Ses valeurs admises sont : EXECUTION_MODE_MICROFLOW (1) EXECUTION_MODE_LONG_RUNNING (2)
DESCRIPTION	Chaîne	La description du modèle de processus.
COMP_SPHERE	Entier	Spécifie le comportement à la compensation des instances de microflux du modèle de processus ; une sphère de compensation existante est ajoutée ou bien une nouvelle sphère est créée Ses valeurs admises sont : COMP_SPHERE_REQUIRED (2) COMP_SPHERE_SUPPORTS (4)
DISPLAY_NAME	Chaîne	Le nom descriptif du processus.

Vue QUERY_PROPERTY :

Cette vue prédéfinie de la base de données vous permet de lancer des requêtes sur les variables de niveau processus.

Tableau 16. Colonnes de la vue QUERY_PROPERTY

Nom de colonne	Type	Commentaires
PIID	ID	L'ID de l'instance de processus.
VARIABLE_NAME	String (chaîne)	Nom de la variable de niveau processus.
NAME	String (chaîne)	Nom de la propriété de requête.
NAMESPACE	String (chaîne)	Espace de nom de la propriété de requête.
GENERIC_VALUE	String (chaîne)	Une représentation sous forme de chaîne des types de propriétés ne correspondant pas à l'un de ces types définis : STRING_VALUE, NUMBER_VALUE, DECIMAL_VALUE ou TIMESTAMP_VALUE.
STRING_VALUE	String (chaîne)	Si un type de propriété est mappé avec un type Chaîne, il s'agit de la valeur de cette chaîne.
NUMBER_VALUE	Integer (entier)	Si un type de propriété est mappé avec un type Entier, il s'agit de la valeur de cet entier.
DECIMAL_VALUE	Décimal	Si un type de propriété est mappé avec un type virgule flottante, il s'agit de la valeur de la décimale.
TIMESTAMP_VALUE	Horodatage	Si un type de propriété est mappé avec un type horodatage, il s'agit de la valeur de celui-ci.

Vue TASK :

Cette vue prédéfinie de la base de données vous permet de rechercher des données dans les objets de tâche.

Tableau 17. Colonnes de la vue TASK

Nom de colonne	Type	Commentaires
TKIID	ID	L'ID de l'instance de tâche.
ACTIVATED	Horodatage	L'heure de l'activation de l'activité.
APPLIC_ DEFAULTS_ID	ID	L'ID du composant d'application qui spécifie les valeurs par défaut de la tâche.
APPLIC_NAME	Chaîne	Le nom de l'application d'entreprise à laquelle appartient la tâche.

Tableau 17. Colonnes de la vue TASK (suite)

Nom de colonne	Type	Commentaires
BUSINESS_ RELEVANCE	Booléen	Indique si la tâche présente une importance professionnelle significative. Cet attribut intervient sur la journalisation dans le journal d'audit. Ses valeurs admises sont : **TRUE** La tâche est une tâche professionnelle significative et fera l'objet d'un audit. **FALSE** La tâche n'est pas une tâche professionnelle significative et ne fera pas l'objet d'un audit.
COMPLETED	Horodatage	L'heure à laquelle l'activité a été terminée.
CONTAINMENT_ CTX_ID	ID	Le contexte de confinement pour cette tâche. Cet attribut détermine le cycle de vie de la tâche. Lorsque le contexte de confinement d'une tâche est supprimé, la tâche est aussi supprimée.
CTX_ AUTHORIZATION	Entier	Permet au propriétaire de la tâche d'accéder au contexte de la tâche. Ses valeurs admises sont : **AUTH_NONE** Pas de droits sur l'objet de contexte associé. **AUTH_READER** Les opérations sur l'objet de contexte associé nécessitent le droit de lecteur, par exemple, la lecture des propriétés d'une instance de processus.
DUE	Horodatage	L'heure de l'échéance de la tâche.
EXPIRES	Horodatage	La date d'expiration de la tâche.
FIRST_ACTIVATED	Horodatage	L'heure de la première activation de la tâche.
FOLLOW_ON_TKIID	ID	L'ID d'instance de la tâche de suivi.
HIERARCHY_ POSITION	Entier	Ses valeurs admises sont : **HIERARCHY_POSITION_TOP_TASK (0)** Tâche de niveau supérieur dans la hiérarchie de tâches. **HIERARCHY_POSITION_SUB_TASK (1)** La tâche est une sous-tâche dans la hiérarchie de tâches. **HIERARCHY_POSITION_FOLLOW_ON_TASK (2)** La tâche est une tâche de suivi dans la hiérarchie de tâches.
IS_AD_HOC	Booléen	Indique si la tâche a été créée dynamiquement au moment de l'exécution ou via un modèle de tâche.
IS_ESCALATED	Booléen	Indique si un transfert de cette tâche s'est produit.
IS_INLINE	Booléen	Indique si la tâche est une tâche en ligne dans le cadre d'une processus métier.
IS_WAIT_FOR_ SUB_TK	Booléen	Indique si la tâche parent doit attendre qu'une sous-tâche soit entrée dans un état terminal.

Tableau 17. Colonnes de la vue TASK (suite)

Nom de colonne	Type	Commentaires
KIND	Entier	Le type de la tâche. Ses valeurs admises sont : **KIND_HUMAN (101)** Déclare que la tâche est une *tâche de collaboration* créée et traitée par un utilisateur. **KIND_WPC_STAFF_ACTIVITY (102)** Déclare que la tâche est une tâche utilisateur qui est en fait une activité de personnel d'un processus métier WebSphere Business Integration Server Foundation version 5. **KIND_ORIGINATING (103)** Déclare que la tâche est une *tâche d'appel* qui prend en charge les interactions de type utilisateur-à-ordinateur qui permettent à une personne de créer, d'initier et de démarrer des services. **KIND_PARTICIPATING (105)** Déclare que la tâche est une *tâche à effectuer* qui prend en charge les interactions de type ordinateur-à-utilisateur qui permettent à une personne d'implémenter un service. **KIND_ADMINISTRATIVE (106)** Déclare que la tâche est une tâche d'administration.
LAST_MODIFIED	Horodatage	L'heure de la dernière modification de la tâche.
LAST_STATE_ CHANGE	Horodatage	L'heure de la dernière modification de l'état de la tâche.
NAME	Chaîne	Le nom de la tâche.
NAME_SPACE	Chaîne	L'espace nom utilisé pour établir la catégorie de la tâche.
ORIGINATOR	Chaîne	L'ID principal de l'auteur de la tâche.
OWNER	Chaîne	L'ID principal du propriétaire de la tâche.
PARENT_ CONTEXT_ID	Chaîne	Le contexte parent de la tâche. Cet attribut fournit une clé au contexte correspondant dans le composant d'application appelant. Le contexte parent est défini par le composant d'application qui crée la tâche.
PRIORITY	Entier	La priorité de la tâche.
RESUMES	Horodatage	L'heure à laquelle la tâche doit redémarrer automatiquement.
STARTED	Horodatage	L'heure du démarrage de la tâche (STATE_RUNNING, STATE_CLAIMED).
STARTER	Chaîne	L'ID principal de l'initiateur de la tâche.

Tableau 17. Colonnes de la vue TASK (suite)

Nom de colonne	Type	Commentaires
STATE	Entier	L'état de la tâche. Ses valeurs admises sont : **STATE_READY (2)** Déclare que la tâche est prête à être réclamée. **STATE_RUNNING (3)** Déclare que la tâche a été démarrée et qu'elle est en cours d'exécution. **STATE_FINISHED (5)** Déclare que la tâche a abouti. **STATE_FAILED (6)** Déclare que la tâche n'a abouti correctement. **STATE_TERMINATED (7)** Déclare que la tâche a été arrêtée à cause d'une demande interne ou externe. **STATE_CLAIMED (8)** Déclare que la tâche a été réclamée. **STATE_EXPIRED (12)** Déclare que la tâche a été terminée parce qu'elle a dépassé sa durée définie. **STATE_FORWARDED (101)** Indique que la tâche a été effectuée avec une tâche de suivi.
SUPPORT_ AUTOCLAIM	Booléen	Indique si la tâche est automatiquement réclamée ou attribuée à un utilisateur unique.
SUPPORT_CLAIM_ SUSP	Booléen	Indique si la tâche peut être réclamée lorsqu'elle est mise en suspens.
SUPPORT_ DELEGATION	Booléen	Indique si cette tâche prend en charge la délégation du travail par le biais de la création, de la suppression ou du transfert d'éléments de travail.
SUPPORT_ FOLLOW_ON	Booléen	Indique si cette tâche prend en charge la création de tâches de suivi.
SUPPORT_SUB_TASK	Booléen	Indique si cette tâche prend en charge la création de sous-tâches.
SUSPENDED	Booléen	Indique si la tâche a été mise en suspens.
TKTID	ID	L'ID du modèle de la tâche.
TOP_TKIID	ID	L'ID d'instance de tâche parent de niveau supérieur s'il s'agit d'une sous-tâche.
TYPE	Chaîne	Le type utilisé pour établir la catégorie de la tâche.

Vue TASK_CPROP :

Utilisez cette vue prédéfinie de la base de données pour rechercher des propriétés personnalisées des objets de tâche.

Tableau 18. Colonnes de la vue TASK_CPROP

Nom de colonne	Type	Commentaires
TKIID	Chaîne	L'ID d'instance de la tâche.
NAME	Chaîne	Nom de la propriété.
DATA_TYPE	Chaîne	Type de la classe pour les propriétés personnalisés autres que de type Chaîne.
STRING_VALUE	Chaîne	La valeur des propriétés personnalisées de type Chaîne.

Vue TASK_DESC :

Utilisez cette vue prédéfinie de la base de données pour rechercher des données descriptives multilingues sur les objets de tâche.

Tableau 19. Colonne de la vue TASK_DESC

Nom de colonne	Type	Commentaires
TKIID	Chaîne	L'ID d'instance de la tâche.
LOCALE	Chaîne	Le nom du paramètre régional associé à la description ou au nom affiché.
DESCRIPTION	Chaîne	Description de la tâche.
DISPLAY_NAME	Chaîne	Le nom descriptif de la tâche.

Vue TASK_TEMPL :

Cette vue prédéfinie de la base de données contient des données vous permettant d'instancier des tâches.

Tableau 20. Colonnes de la vue TASK_TEMPL

Nom de colonne	Type	Commentaires
TKTID	Chaîne	L'ID du modèle de la tâche.
VALID_FROM	Horodatage	L'heure à laquelle le modèle de tâche devient disponible pour être instancié.
APPLIC_ DEFAULTS_ID	Chaîne	L'ID du composant d'application qui spécifie les valeurs par défaut du modèle de tâche.
APPLIC_NAME	Chaîne	Le nom de l'application d'entreprise à laquelle appartient le modèle de tâche.
BUSINESS_ RELEVANCE	Booléen	Indique si le modèle de tâche présente une importance professionnelle significative. Cet attribut intervient sur la journalisation dans le journal d'audit. Ses valeurs admises sont : **TRUE** La tâche est une tâche professionnelle significative et fera l'objet d'un audit. **FALSE** La tâche n'est pas une tâche professionnelle significative et ne fera pas l'objet d'un audit.

Tableau 20. Colonnes de la vue TASK_TEMPL (suite)

Nom de colonne	Type	Commentaires
CONTAINMENT_ CTX_ID	ID	Le contexte de confinement pour ce modèle de tâche. Cet attribut détermine le cycle de vie du modèle de tâche. Lorsqu'un contexte de confinement est supprimé, le modèle de tâche est aussi supprimé.
CTX_ AUTHORIZATION	Entier	Permet au propriétaire de la tâche d'accéder au contexte de la tâche. Ses valeurs admises sont : **AUTH_NONE** Pas de droits sur l'objet de contexte associé. **AUTH_READER** Les opérations sur l'objet de contexte associé nécessitent le droit de lecteur, par exemple, la lecture des propriétés d'une instance de processus.
DEFINITION_NAME	Chaîne	Le nom de la définition du modèle de tâche dans le fichier TEL (Task Execution Language).
DEFINITION_NS	Chaîne	L'espace de nom de la définition du modèle de tâche dans le fichier TEL.
IS_AD_HOC	Booléen	Indique si ce modèle de tâche a été créé dynamiquement lors de l'exécution ou créé lors du déploiement de la tâche en tant que fichier EAR.
IS_INLINE	Booléen	Indique si le modèle de tâche est défini en tant que tâche d'un processus métier.
KIND	Entier	Le type de tâches dérivées de ce modèle de tâche. Ses valeurs admises sont : **KIND_HUMAN (101)** Déclare que la tâche est une *tâche de collaboration* créée et traitée par un utilisateur. **KIND_ORIGINATING (103)** Déclare que la tâche est une *tâche d'appel* qui prend en charge les interactions de type utilisateur-à-ordinateur qui permettent à une personne de créer, d'initier et de démarrer des services. **KIND_PARTICIPATING (105)** Déclare que la tâche est une *tâche à effectuer* qui prend en charge les interactions de type ordinateur-à-utilisateur qui permettent à une personne d'implémenter un service. **KIND_ADMINISTRATIVE (106)** Déclare que la tâche est une tâche d'administration.
NAME	Chaîne	Le nom du modèle de tâche.
NAMESPACE	Chaîne	L'espace nom utilisé pour établir la catégorie du modèle de tâche.
PRIORITY	Entier	La priorité du modèle de tâche.

Tableau 20. Colonnes de la vue TASK_TEMPL (suite)

Nom de colonne	Type	Commentaires
STATE	Entier	L'état du modèle de la tâche. Ses valeurs admises sont : **STATE_STARTED (1)** Spécifie que le modèle de tâche est disponible pour la création d'instances. **STATE_STOPPED (2)** Spécifie que le modèle de tâche a été arrêté. Les instances de tâches ne peuvent être créées à partir d'un modèle de tâche dans cet état.
SUPPORT_ AUTOCLAIM	Booléen	Indique si les tâches dérivées de ce modèle de tâche peuvent être réclamées automatiquement si elles sont affectées à un seul utilisateur.
SUPPORT_CLAIM_ SUSP	Booléen	Indique si les tâches dérivées de ce modèle de tâche peuvent être réclamées si elles sont en suspens.
SUPPORT_ DELEGATION	Booléen	CORRIGERIndique si les tâches dérivées de ce modèle de tâche prennent en charge la délégation de travail à l'aide de la création, de la suppression ou du transfert d'éléments de travail.
SUPPORT_ FOLLOW_ON	Booléen	Indique si le modèle de tâche prend en charge la création de tâches de suivi.
SUPPORT_SUB_TASK	Booléen	Indique si le modèle de tâche prend en charge la création de sous-tâches.
TYPE	Chaîne	Le type utilisé pour établir la catégorie du modèle de la tâche.

Vue TASK_TEMPL_CPROP :

Cette vue prédéfinie de la base de données vous permet de rechercher des propriétés personnalisées des modèles de tâche.

Tableau 21. Colonnes de la vue TASK_TEMPL_CPROP

Nom de colonne	Type	Commentaires
TKTID	Chaîne	L'ID du modèle de la tâche.
NAME	Chaîne	Nom de la propriété.
DATA_TYPE	Chaîne	Type de la classe pour les propriétés personnalisés autres que de type Chaîne.
STRING_VALUE	Chaîne	La valeur des propriétés personnalisées de type Chaîne.

Vue TASK_TEMPL_DESC :

Utilisez cette vue prédéfinie de la base de données pour rechercher des données descriptives multilingues sur les objets de modèles de tâche.

Tableau 22. Colonnes de la vue TASK_TEMPL_DESC

Nom de colonne	Type	Commentaires
TKTID	Chaîne	L'ID du modèle de la tâche.
LOCALE	Chaîne	Le nom du paramètre régional associé à la description ou au nom affiché.
DESCRIPTION	Chaîne	Une description du modèle de tâche.
DISPLAY_NAME	Chaîne	Le nom descriptif du modèle de tâche.

Vue WORK_ITEM :

Utilisez cette vue prédéfinie de la base de données pour rechercher des données sur les éléments de travail et sur les droits relatifs aux processus, aux tâches et aux escalades.

Tableau 23. Colonnes de la vue WORK_ITEM

Nom de colonne	Type	Commentaires
WIID	ID	L'ID de l'élément de travail.
OWNER_ID	Chaîne	L'ID principal du propriétaire.
GROUP_NAME	Chaîne	Le nom de la liste de travail du groupe associé.
EVERYBODY	Booléen	Spécifie si tous les utilisateurs sont les propriétaires de cet élément.

Tableau 23. Colonnes de la vue WORK_ITEM (suite)

Nom de colonne	Type	Commentaires
OBJECT_TYPE	Entier	Le type de l'objet associé. Ses valeurs admises sont : **OBJECT_TYPE_ACTIVITY (1)** Spécifie que l'élément de travail a été créé pour une activité. **OBJECT_TYPE_PROCESS_INSTANCE (3)** Spécifie que l'élément de travail a été créé pour une instance de processus. **OBJECT_TYPE_TASK_INSTANCE (5)** Spécifie que l'élément de travail a été créé pour une tâche. **OBJECT_TYPE_TASK_TEMPLATE (6)** Spécifie que l'élément de travail a été créé pour un modèle de tâche. **OBJECT_TYPE_ESCALATION_ INSTANCE (7)** Spécifie que l'élément de travail a été créé pour une instance d'escalade. **OBJECT_TYPE_APPLICATION_ COMPONENT (9)** Spécifie que l'élément de travail a été créé pour un composant d'application.
OBJECT_ID	ID	L'ID de l'objet associé, par exemple le processus ou la tâche associée.
ASSOC_OBJECT_TYPE	Entier	Le type d'objet auquel fait référence l'attribut ASSOC_OID, par exemple tâche, processus ou objet externe. Utilisez ces valeurs pour l'attribut OBJECT_TYPE.
ASSOC_OID	ID	L'ID de l'objet associé à l'élément de travail. Par exemple, l'ID (PIID) de l'instance de processus qui contient l'instance d'activité pour laquelle l'élément de travail a été créé.
REASON	Entier	Le motif d'attribution de l'élément de travail. Ses valeurs admises sont : REASON_POTENTIAL_STARTER (5) REASON_POTENTIAL_INSTANCE_ CREATOR (11) REASON_POTENTIAL_STARTER (1) REASON_EDITOR (2) REASON_READER (3) REASON_ORIGINATOR (9) REASON_OWNER (4) REASON_STARTER (6) REASON_ESCALATION_RECEIVER (10) REASON_ADMINISTRATOR (7)
CREATION_TIME	Horodatage	La date et l'heure auxquelles l'élément de travail a été créé.

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 : Répertoriez 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'.

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», à la page 35
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.

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));
   ```

   Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Répertoriez 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 souhaitez exécuter cette requête pour des objets liés à des tâches, spécifiez StoredQuery en tant que type d'objet renvoyé, au lieu de StoredQueryData.

```
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");
   ```

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.

```
StoredQuery 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();
```

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");
   ```

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.

   ```
   StoredQuery storedQuery = process.getStoredQuery
      ("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 privée.

   Le fragment de code suivant indique comment supprimer une requête stockée privée.

   ```
   process.deleteStoredQuery("CustomerOrdersStartingWithA");
   ```

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.

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 certain processus. 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

Action	Rôle principal de l'appelant		
	Lecteur	Initiateur	Administrateur
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 certaine activité. 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

Action	Rôle principal de l'appelant				
	Lecteur	**Editeur**	**Propriétaire potentiel**	**Propriétaire**	**Administrateur**
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.

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.

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.EXCECUTION_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.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");
   }

   //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.

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.EXCECUTION_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.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");
   }
   //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.

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 humaines ou des activités de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

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.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 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.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.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 humaines ou des activités de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

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 prérequises 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 ; les tâches autonomes associées à cette instance de processus ne sont pas interrompues.

   Dans cet état, les activités en cours peuvent toujours être terminées mais aucune nouvelle activité n'est activée. Par exemple, une activité humaine 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.

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.

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 devez 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 s'exécute comme si un incident s'était produit sur la portée de niveau supérieur. Si vous arrêtez l'instance de processus sans compensation, celle-ci est aussitôt arrêtée sans attendre que les activités, les tâches à effectuer ou les tâches d'appel en ligne se terminent normalement.

Les applications démarrées par le processus et les tâches autonomes liées à ce 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.

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.

Traitement des activités humaines

Les activités humaines sont attribuées aux différentes personnes de votre organisation par l'intermédiaire des éléments de travail. 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. Le traitement de l'activité et la gestion de l'élément de travail sont délégués à Human Task Manager. Toute modification d'état de l'instance d'activité se reflète dans l'instance de tâche, 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 aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 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(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
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(aiid, 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.

```
//extraire les erreurs modélisées pour l'activité humaine
List faultNames = process.getFaultNames(aiid);

//créer un message de type approprié
ClientObjectWrapper myFault =
      process.createMessage(aiid, 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);
}

process.complete(aiid, (String)faultNames.get(0), myFault);
```

      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 achèvement ou un redémarrage forcé.

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 mise en oeuvre sous forme d'une série d'activités humaines (tâches à effectuer). Si l'acheteur décide de commander plusieurs ouvrages, ceci équivaut à réclamer la prochaine activité humaine. 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 réclame la suivante, dans la même instance de processus, pour la personne connectée. L'API renvoie ensuite les informations sur l'activité réclamée suivante, y compris le message d'entrée à traiter. Etant donné que l'activité suivante est mise à disposition dans la même transaction que celle de l'activité terminée, la valeur associée au comportement transactionnel de toutes les activités humaines du modèle de processus doit être `participates`.

Comparez cet exemple avec l'exemple qui contient à la fois les API Business Flow Manager et Human Task Manager.

Procédure

1. Réclamez la première activité dans la séquence d'activités.

```
//
//Exécuter une 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 aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 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(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
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(aiid, output);
```

   Cette opération définit un message de sortie contenant la référence de la commande et réclamant 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 et si ces chemins d'accès contiennent des activités humaines et que l'utilisateur connecté soit le propriétaire potentiel de plusieurs d'entre elles, une activité aléatoire est réclamée automatiquement et est renvoyée en tant qu'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

«Traitement par une seule personne d'un flux de travaux contenant des tâches utilisateur», à la page 110
Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple montre la séquence d'actions à exécuter pour décomposer la commande en plusieurs activités humaines (tâches à effectuer). Les interfaces API Flow Manager et Human Task Manager APIs permettent de 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 initiateur potentiel de l'activité recevant le message ou administrateur de l'instance de processus.

   ```
   VTID vtid = services[0].getServiceTemplateID();
   ATID atid = services[0].getActivityTemplateID();
   String inputType = services[0].getInputMessageTypeName();

   // crêer 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.

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.getInputMessageTypeName());

    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.

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 de processus de longue durée à interfaces unidirectionnelles ne peuvent pas être extraits à l'aide de la méthode getOutputMessage car le processus n'a pas de sortie. Vous pouvez cependant exécuter une requête sur 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();
  PIID piid = (PIID) 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");
  }
}
```

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.

Forcer une activité à se terminer : A propos de cette tâche

Les activités situées dans des processus de longue durée rencontrent parfois des erreurs. Si ces erreurs ne sont pas interceptes 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.

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 du 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. Terminez 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();
     //définir les parties du message, par exemple un numéro d'ordre
     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.

Nouvelle tentative d'exécution d'une activité arrêtée : A propos de cette tâche

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.

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 arrêtée, par exemple.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper input =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
    {
      myMessage = (DataObject)input.getObject();
      //définir les chaînes du message, par exemple "il faut commander
        du chocolat"
      myMessage.setString("OrderNo", "chocolate");
    }

   boolean continueOnError = true;
   process.forceRetry(aiid, 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 et un gestionnaire d'erreur au niveau des processus s'exécute avant que le processus ne change d'état.

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 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 24. Méthodes API pour les modèles de processus

Méthode	Description
getProcessTemplate	Extrait le modèle de processus spécifié.
queryProcessTemplates	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 25. Les méthodes API sont liées au démarrage des instances de processus.

Méthode	Description
call	Crée et exécute un microflux.
callWithReplyContext	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.
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.

Tableau 25. Les méthodes API sont liées au démarrage des instances de processus. (suite)

Méthode	Description
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 26. 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 27. 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.

Tableau 27. Méthodes API pour le contrôle du cycle de vie des instances d'activité (suite)

Méthode	Description
cancelClaim	Annule la réclamation de l'instance d'activité.
complete	Termine l'instance d'activité.
completeAndClaimSuccessor	Effectue une activité d'instance et réclame la suivante dans la même instance de processus, pour la personne connectée.
forceComplete	Impose l'achèvement d'une instance d'activité se trouvant à l'état d'exécution en cours ou arrêté.
forceRetry	Impose la répétition d'une instance d'activité se trouvant à l'état d'exécution en cours ou arrêté.
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 28. 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.
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.

Démarrage d'une tâche d'appel qui appelle une interface synchrone

Une tâche d'appel est associée à un composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Démarrez la tâche d'appel de façon synchrone uniquement si le composant SCA associé peut être appelé de façon 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.

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 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");
   }
   ```

Démarrage d'une tâche d'appel qui appelle une interface asynchrone

Une tâche d'appel est associée à un composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Démarrez la tâche d'appel de façon asynchrone uniquement si le composant SCA associé peut être appelé de façon 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.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);
   ```

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* dans l'API) et comment démarrer cette 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 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 et démarrez 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.

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*) sont attribuées à diverses personnes de votre organisation par le biais des éléments de travail. Les tâches à effectuer et les 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();
  //définir les parties du message d'erreur, par exemple un numéro d'ordre
  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.

```
//extraire les erreurs modélisées pour la tâche
ListfaultNames input = task.getFaultNames(tkiid);

//créer un message de type approprié
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.

Mise en suspens et reprise d'une instance de tâche

Vous pouvez mettre en suspens des instances de tâches de collaboration (également appelées *tâches utilisateur* dans l'API) ou des instances de tâches à effectuer (également appelées *tâches de participation*).

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.

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*) s'exécute 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");
  }
}
```

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.

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 automtiquement.

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);
}
```

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.

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 de tâche utilisateur, 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. Par exemple, une instance de tâche à effectuer peut être associée à un élément de travail de types 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.

```
// exécuter une requête sur l'instance de tâche pour laquelle un
// autre administrateur doit être indiqué
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // créer l'élément de travail
  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.

```
// exécuter une requête sur l'instance de tâche pour laquelle un élément de
   travail doit être supprimé
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // supprimer l'élément de travail
  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.

```
// exécuter la tâche devant être replanifée
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.NAME='CustomerOrder' AND
                 TASK.STATE=TASK.STATE.STATE_READY AND
                 WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER AND
                 WORK_ITEM.OWNER_ID='Miller'",
                 (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // transférer l'élément de travail de l'utilisateur Miller à l'utilisateur
     Smith
  // pour que Smith puisse utiliser la tâche
  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.

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 en vue de créer un modèle de tâche réutilisable ou 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 à l'exécution est basée sur EMF (Eclipse Modeling Framework).

Procédure

1. Créez un ensemble 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 messages complexes, vous pouvez soit les définir à l'aide de org.eclipse.xsd.XSDFactory (via la méthode de fabrique getXSDFactory()), soit importer directement un schéma XML existant à l'aide de la fabrique de méthode loadXSDSchema.

   Déployez les types complexes comme des parties d'une application d'entreprise pour qu'ils soient disponibles dans WebSphere Process Server.
3. Créez ou importez une définition WSDL (Web Services Definition Language) du type javax.wsdl.Definition.

Vous pouvez créer une définition WSDL à l'aide de la méthode createWSDLDefinition. Ajoutez-lui un type de port et une opération. Vous pouvez également importer directement une définition WSDL 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 souhaitez ajouter ou manipuler des éléments de tâches plus complexes, utilisez la classe com.ibm.wbit.tel.TaskFactory que vous pouvez extraire de la méthode de fabrique getTaskFactory.
5. Créez le modèle de tâche à l'aide de la méthode de fabrique createTaskModel et transmettez-le au regroupement de ressources créé à l'étape 1 et qui agrège tous les artefacts créés entre-temps.
6. Facultatif : Validez le modèle à l'aide de la méthode validate.

Résultats

Utilisez l'une des méthodes API EJB create de Human Task Manager associée à un paramètre **TaskModel** pour créer ou un modèle de tâche réutilisable ou une instance de tâche à exécution unique.

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.

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, "BOapplication", "HTM" );
     ```

   - Le fragment de code suivant crée un modèle de tâche :

     ```
     task.createTaskTemplate( taskModel, "BOapplication" );
     ```

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.

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, "BOapplication", "HTM" );
     ```

   - Le fragment de code suivant crée un modèle de tâche :

     ```
     task.createTaskTemplate( taskModel, "BOapplication" );
     ```

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.

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.

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 29. 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'entré 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 30. 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 31. 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 32. 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.

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 de collaboration, d'une tâche d'appel ou d'une tâche d'administration.

Vous ne pouvez pas utiliser toutes les actions disponibles via l'interface HumanTaskManager pour toutes les tâches. Le tableau suivant indique les actions que vous pouvez effectuer sur chaque type de tâche.

	Type de tâche			
Action	**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
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

Action	Type de tâche			
	Tâche à effectuer	**Tâche de collaboration**	**Tâche d'appel**	**Tâche d'administration**
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

Développement d'applications pour les processus métier et les tâches utilisateur

Les utilisateurs sont impliqués dans la plupart des scénarios de processus métier. Par exemple, l'intervention d'un utilisateur est indispensable lors du démarrage ou de la gestion d'un processus métier ou lors de l'exécution de tâches utilisateur. Pour prendre en charge ces différents scénarios, vous devrez utilisez les API Business Flow Manager API et Human Task Manager.

A propos de cette tâche

Pour impliquer les utilisateurs dans des scénarios de processus métier, vous pouvez y inclure les types de tâches suivantes :

- Une tâche d'appel en ligne (également appelée *tâche d'origine* dans l'API).

  Vous pouvez fournir une tâche d'appel pour chaque activité de réception, pour chaque élément onMessage d'une activité de sélection et pour chaque élément onEvent d'un gestionnaire d'événements. Cette tâche contrôle alors qui est autorisé 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 pour indiquer qui est autorisé à gérer le processus ou à effectuer des opérations d'administration sur les activités ayant échoué.
- Une tâche à effectuer (également appelée *tâche de participation* dans l'API).

Une tâche à effectuer implémente une activité humaine. Ce type d'activité vous permet d'impliquer des utilisateurs dans le processus.

Les activités humaines du processus métier représentent les tâches à effectuer dans le scénario de processus métier. Vous pouvez utiliser les API Business Flow Manager API et Human Task Manager API pour mettre en oeuvre ces scénarios :

- Le processus métier est un conteneur pour toutes les activités faisant partie d'un processus, y compris les activités humaines représentées par des tâches à effectuer. Lors de la création d'une instance de processus, un ID d'objet unique (le PIID) lui est affecté.
- Lorsqu'une activité humaine est créée au cours de l'exécution de l'instance de processus, une instance d'activité est créée. Elle est identifiée par un ID d'objet unique (l'AIID). Une instance de tâche à effectuer est simultanément créée et identifiée par un ID d'objet (le TKIID). La relation entre l'activité humaine et l'instance de tâche se fait par l'intermédiaire des ID d'objets :
  - La valeur du TKIID de la tâche à effectuer associée est affectée à l'ID de la tâche à effectuer de l'instance d'activité.
  - La valeur du PIID de l'instance de processus contenant l'instance d'activité associée est affectée à l'ID de contexte de confinement de l'instance de tâche.
  - La valeur de l'AIID de l'instance d'activité associée est affectée à l'ID de contexte parent de l'instance de tâche.
- Les cycles de vie de toutes les instances de tâche en ligne sont gérées 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 la valeur du PIID de l'instance de processus est affectée à l'ID de contexte parent sont automatiquement supprimées.

Détermination des modèles de processus ou des activités pouvant démarrer

Un processus métier peut être démarré à l'aide des méthodes call, initiate ou sendMessage de l'API Business Flow Manager. Si le processus ne contient qu'une activité de démarrage, vous pouvez utiliser la signature de méthode qui requiert un nom de modèle de processus comme paramètre. S'il en contient plusieurs, vous devez identifier explicitement l'activité de démarrage.

A propos de cette tâche

Lors de la modélisation d'un processus métier, le modélisateur peut décider que seul un sous-ensemble d'utilisateurs pourront créer une instance de processus à partir d'un modèle de processus. Ceci est possible en associant une tâche d'appel en ligne à une activité de démarrage du processus et en indiquant des restrictions d'autorisation pour cette tâche. Seuls les initiateurs potentiels ou les administrateurs d'une tâche sont autorisés à créer des instances de cette tâche et, par conséquent, à créer une instance du modèle de tâche.

Si aucune tâche d'appel en ligne n'est associée à l'activité de démarrage, ou si aucune restriction d'autorisation n'est associée à la tâche, tous les utilisateurs peuvent créer une instance de processus à l'aide de l'activité de démarrage.

Un processus peut contenir plusieurs activités de démarrage, chacune étant associée à différentes requêtes d'utilisateur pour les initiateurs potentiels ou les administrateurs. Ainsi, un utilisateur peut être autorisé à démarrer un processus à l'aide de l'activité A mais pas de l'activité B.

Procédure

1. Utilisez l'API Business Flow Manager pour créer une liste des versions en cours des modèles de processus qui sont à l'état démarré.

   Conseil : La méthode queryProcessTemplates n'exclut que les modèles de processus faisant partie d'applications n'ayant pas encore démarré. Si vous l'utilisez sans filtrer les résultats, cette méthode renvoie toutes les versions de modèles de processus quel que soit leur état.

   ```
   // horodatage en cours au format UTC, converti en aaaa-mm-jjThh: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",
                                             (Integer)null, (TimeZone)null);
   ```

   Les résultats sont classés par nom de modèle de processus.

2. Créez la liste des modèles de processus et la liste des activités de démarrage pour lesquelles l'utilisateur dispose des autorisations nécessaires.

   La liste des modèles de processus contient les modèles ne contenant qu'une activité de démarrage. Soit ces activités ne sont pas sécurisées, soit l'utilisateur connecté peut les démarrer. Il se peut également que vous souhaitiez rassembler les modèles de processus pouvant être démarrés par l'une des activités de démarrage au moins.

   Conseil : L'administrateur du processus peut aussi démarrer une instance de processus. Pour obtenir une liste complète des modèles, vous devez aussi lire le modèle de tâche d'administration associée au modèle de processus et vérifier si l'utilisateur connecté est administrateur.

   ```
   List authorizedProcessTemplates = new ArrayList();
   List authorizedActivityServiceTemplates = new ArrayList();
   ```

3. Déterminez les activités de démarrage de chaque modèle 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, extrayez l'identificateur 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 aucun modèle de tâche d'appel n'existe, le modèle de processus n'est pas sécurisé par l'activité de démarrage.

      Dans ce cas, tous les utilisateurs peuvent créer une instance de processus à l'aide de cette activité.

```
boolean isAuthorized = false;
       if ( tktid == null )
    {
      isAuthorized = true;
      authorizedActivityServiceTemplates.add(activity);
    }
```

b. Si aucun modèle de tâche d'appel n'existe, utilisez l'API Human Task Manager pour vérifier l'autorisation dont dispose l'utilisateur connecté.

   Dans l'exemple ci-dessous, l'utilisateur connecté s'appelle Smith. Il doit être initiateur potentiel de la tâche d'appel ou 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 a le rôle indiqué, ou si aucun critère d'affectation des utilisateurs n'est indiqué, la méthode isUserInRole renvoie la valeur `true`.

5. Vérifiez si le processus peut être démarré à l'aide du seul nom de modèle de processus.

```
if ( isAuthorized && startActivities.length == 1 )
    {
      authorizedProcessTemplates.add(template);
    }
```

6. Terminez la boucle.

```
  } // terminer la boucle de chaque modèle de service d'activité
}   // fin de la boucle de chaque modèle de 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 montre la séquence d'actions à exécuter pour décomposer la commande en plusieurs activités humaines (tâches à effectuer). Les interfaces API Flow Manager et Human Task Manager APIs permettent de traiter le flux de travaux.

A propos de cette tâche

Dans une librairie en ligne, l'acheteur accomplit une séquence d'actions afin de commander un ouvrage. Ces actions constituent une série d'activités humaines (tâches à effectuer). Si l'acheteur décide de commander plusieurs ouvrages, ceci équivaut à réclamer la prochaine activité humaine. Les informations concernant la séquence de tâches sont gérées par Business Flow Manager alors que les tâches elles-mêmes sont gérées par Human Task Manager.

Comparez cet exemple avec l'exemple qui contient uniquement l'API Business Flow Manager.

Procédure

1. Accédez à l'instance de processus concernée dans l'API Business Flow Manager.

   Cet exemple illustre une instance du processus CustomerOrder.

```
ProcessInstanceData processInstance =
   process.getProcessInstance("CustomerOrder");
String piid = processInstance.getID().toString();
```

2. Utilisez l'API Human Task Manager pour exécuter une requête sur les tâches à effectuer (de type tâche de participation) faisant partie de l'instance de processus indiquée.

   Utilisez l'ID de contexte de confinement de la tâche pour indiquer l'instance de processus contenant les tâches. Dans le cas d'un flux de travaux exécuté par une seule personne, la requête renvoie la tâche à effectuer associée à la première activité humaine de la séquence.

```
//
 // Requête portant sur la liste des tâches à effectuer pouvant être réclamées
    par l'utilisateur connecté
 // pour l'instance de processus indiquée
//
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();
    // lire les valeurs
    ...
  }
}
```

   Une fois la tâche réclamée, le message d'entrée de la tâche est renvoyé.

4. Déterminez quelle activité humaine est associée à la tâche à effectuer.

   Vous pouvez choisir l'une des méthodes suivantes pour mettre en relation les activités et leurs tâches :

   - La méthode task.getActivityID :

     ```
     AIID aiid = task.getActivityID(tkiid);
     ```

   - L'identificateur de contexte parent faisant 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. Une fois le traitement de la tâche terminé, utilisez l'API Business Flow Manager pour terminer la tâche et les activités humaines associées et réclamez l'activité humaine suivante.

   Pour terminer l'activité humaine, 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(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //définir les parties du message, par exemple un numéro d'ordre
  myMessage.setInt("OrderNo", 4711);
}

//terminer l'activité humaine et les tâches associées
// et réclamer l'activité humaine suivante
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

Cette opération définit un message de sortie contenant la référence de la commande et réclamant l'activité humaine suivante. 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 à suivre et si ces chemins d'accès contiennent des activités humaines et que l'utilisateur connecté est le propriétaire potentiel de plusieurs d'entre elles, une activité aléatoire est réclamée automatiquement et est renvoyée en tant qu'activité suivante.

6. Traitez l'activité humaine suivante.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // lire les valeurs
  ...
}

aiid = successor.getAIID();
```

7. Passez à l'étape 5 pour terminer l'activité et pour extraire la prochaine activé humaine.

Tâches associées

«Traitement d'un flux de travaux par une seule personne», à la page 79
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.

Gestion des exceptions API

A propos de cette tâche

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.

Cependant, il est de coutume de gérer uniquement un sous-ensemble des exceptions de manière spécifique et de fournir une aide générale pour les autres exceptions éventuelles. Toutes les exceptions spécifiques découlent d'une ProcessException ou d'une TaskException générique. Il est *plus profitable* d'intercepter les exceptions génériques à l'aide d'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.

Vérification de l'erreur définie pour une activité

Procédure

1. Répertoriez 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 aiid = (AIID) result.getOID(1);
  ClientObjectWrapper faultMessage = process.getFaultMessage(aiid);
  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.

Vérification d'une erreur survenue lors d'une activité d'appel arrêtée

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 à l'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 aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);

  ProcessException excp = activity.getUnhandledException();
  if ( excp instanceof ApplicationFaultException )
  {
```

```
        ApplicationFaultException fault = (ApplicationFaultException)excp;
        String faultName = fault.getFaultName();
      }
    }
```

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.

Introduction aux services Web

Les services Web sont des applications d'entreprise basées sur le Web, qui utilisent des normes XML ouvertes et des protocoles d'échange de données avec des applications client. Les services Web permettent d'exploiter un modèle de programmation indépendant du langage et de l'environnement.

Les services Web utilisent les technologies de base suivantes :

- XML (Extensible Markup Language). Le langage XML permet de résoudre les problèmes d'indépendance des données. Il sert à décrire les données et à les mapper vers et depuis n'importe quelle application ou langage de programmation.
- WSDL (Web Services Description Language). Ce langage reposant sur la norme XML sert à créer une description d'application sous-jacente. C'est cette description qui permet de transformer une application en service Web en faisant office d'interface entre l'application sous-jacente et les autres applications Web.
- SOAP (Simple Object Access Protocol). Le protocole de communication SOAP est le protocole principal du Web et est utilisé par la plupart des services Web pour communiquer.

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.

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 67.
- 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 88.

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.

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/littéral encaspulé" 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 Business Flow Manager, l'API 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.

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.

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.

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 à la machine 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 Business Flow Manager) ou à `/sca/com/ibm/task/api/HTMWS` (pour l'API 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"/>
```

Concepts associés

«Ajout de sécurité (services Web Java)», à la page 133
Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Tâches associées

«Renforcement de la sécurité (.NET)», à la page 142
Vous pouvez sécuriser les communications des services Web en intégrant des mécanismes de sécurité à vos applications 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.

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.

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.

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.

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

«Création de classes auxiliaires pour les processus BPEL (.NET)», à la page 138
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.

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.

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 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 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.

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.

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**.

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 *hôte_local* 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 *ConteneurBPE_N1_serveur1* par la racine de contexte de l'application exécutant l'API des services Web. La racine de contexte par défaut est composée comme suit :
      - *ConteneurBPE*. Nom de l'application.
      - *N1*. Nom du noeud.
      - *serveur1*. 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://si.n1.ibm.com:9080/
                  BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

Concepts associés

«Ajout de sécurité (services Web Java)», à la page 133
Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Tâches associées

«Renforcement de la sécurité (.NET)», à la page 142
Vous pouvez sécuriser les communications des services Web en intégrant des mécanismes de sécurité à vos applications client.

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://si.n1.ibm.com:9081/
                  HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

Concepts associés

«Ajout de sécurité (services Web Java)», à la page 133
Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

Tâches associées

«Renforcement de la sécurité (.NET)», à la page 142
Vous pouvez sécuriser les communications des services Web en intégrant des mécanismes de sécurité à vos 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.

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.

Utilisation de Rational Application Developer pour générer un client 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. Dézippez le fichier exporté BPEContainer_*NomNoeud_NomServeur*_WSDLFiles.zip vers un répertoire temporaire.
     b. Importez le sous-répertoire META-INF à partir du répertoire BPEContainer_*NomNoeud_NomServeur*.ear/b.jar.
   - Pour les tâches utilisateur :
     a. Dézippez le fichier exporté TaskContainer_*NomNoeud_NomServeur*_WSDLFiles.zip vers un répertoire temporaire.
     b. Importez le sous-répertoire META-INF à partir du répertoire dézippé TaskContainer_*NomNoeud_NomServeur*.ear/h.jar.

   Le 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 ne pouvez pas sélectionner l'option **Web services** du menu **Preferences**, vous devez d'abord activer les fonctions nécessaires en procédant comme suit : **Window → Preferences → Workbench → Capabilities**. Cliquez sur **Web Service Developer** puis sur **OK**. Ouvrez à nouveau la fenêtre Preferences et modifiez l'option **Code Generation**.
3. Sélectionnez le fichier BFMWS.WSDL ou HTMWS.WSDL se trouvant dans le nouveau répertoire wsdl.
4. Cliquez avec le bouton droit et sélectionnez **Web Services → Generate client**.

   Avant de passer aux étapes suivantes, vérifiez que le serveur a démarré.
5. Dans la fenêtre Web Services, cliquez sur **Next** pour accepter toutes les valeurs par défaut.
6. Dans la fenêtre Web Service Selection , cliquez sur **Next** pour accepter toutes les valeurs par défaut.
7. Dans la fenêtre Client Environment Configuration :
   a. Cliquez sur **Edit** et pour l'option Web service runtime, choisissez la valeur IBM WebSphere.
   b. Pour l'option J2EE Version, choisissez la valeur 1.4.
   c. Cliquez sur **OK**.
   d. Cliquez sur **Next**.
8. Cette étape est nécessaire uniquement si vous devez créer un client Web Services incluant à la fois l'API Business Process et l'API Human Task Web Services, de la même façon qu'il existe des méthodes en double dans les deux fichiers WSDL.

a. Dans la fenêtre Web Service Proxy, sélectionnez Define custom mapping for namespace to package, puis cliquez sur **OK**.
b. Dans la fenêtre Web Service Client namespace to package mapping, ajoutez les espaces de nom et les packages 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
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 le système vous demande de confirmer le remplacement, cliquez sur **YesToAll**.

9. Cliquez sur **Finish**.

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.

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 wsdl2java.bat -r client -c client -noWrappedOperations
                      -output c:\ws\proxyClient  c:\ws\bin\HTMWS.wsdl
   ```
2. Incluez à votre projet les fichiers classe générés.

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/littéral encapsulé". 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 tâches utilisateur ou un module de processus.

   Pour créer le message SOAP correspondant dans l'application client à l'aide de l'API 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 la création de paramètres d'entrée pour l'opération sendMessage dans l'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)

   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.

L'exemple suivant illustre comment utiliser l'API de services Web Business Flow Manager.

```
// Création du proxy
       BFMIFProxy proxy = new BFMIFProxy();
 // Préparation des données d'entrée pour l'opération
      GetProcessTemplate iW = new GetProcessTemplate();
      iW.setIdentifier(your_process_template_name);

 // Appel de l'opération
      GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

 // Sortie du traitement de l'opération
      ProcessTemplateType ptd = oW.getProcessTemplate();
      System.out.println("getName= " + ptd.getName());
      System.out.println("getPtid= " + ptd.getPtid());
```

Tâches associées

«Génération d'un client proxy (services Web Java)», à la page 127
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)», à la page 130
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/littéral encapsulé″. Les applications client requièrent des classes auxiliaires pour leur permettre de générer les éléments d'encapsulation nécessaires.

Ajout de sécurité (services Web Java)

Vous devez sécuriser les communications du service Web en mettant en oeuvre des mécanismes de sécurité dans l'application client.

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

Rôles d'autorisation des processus métier
Un rôle désigne un groupe de personnes partageant le même niveau d'autorisation. Les actions que vous pouvez mettre en place au niveau des processus métier dépendent de votre rôle d'autorisation. Il peut s'agir d'un rôle J2EE ou d'un rôle d'instance.

Information associée

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/c6task_auth.html

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'**éditeur de déploiement** du module.
   b. Cliquez sur l'onglet **WS Extension**.
   c. Sous **Service References**, la référence Web Service References suivante peut apparaître :
      - service/BFMWSService pour les processus métier
      - service/HTMWSService pour les tâches utilisateur

      La liste des références affichées dépend de si BFMWS.wsdl (pour les processus métier) ou HTMWS.wsdl (pour les tâches utilisateur), ou les deux ont été ajoutés lors de la génération du client proxy.
   d. Pour les deux références de service :
      1) Sélectionnez l'une des **Service References**.
      2) Développez la section **Request Generator Configuration**.
      3) Développez la sous-section **Security Token**.
      4) Cliquez sur **Add**. La fenêtre Security Token s'affiche.
      5) Dans la zone **Nom**, entrez le nom du nouveau jeton de sécurité : **UserNameTokenBFM** ou **UserNameTokenHTM**.
      6) Dans la liste déroulante **Token type**, sélectionnez **Username**. (Une valeur par défaut apparaît automatiquement dans la zone **Local name**.)
      7) N'indiquez aucune valeur dans la zone **URI**. Aucune valeur n'est nécessaire pour un jeton de nom d'utilisateur.
      8) Cliquez sur **OK**.
2. Créez un générateur de jetons :
   a. Ouvrez l'**éditeur de déploiement** du module.
   b. Cliquez sur l'onglet **WS Binding**.
   c. Sous **Service References**, les mêmes références de service Web que dans l'étape précédente apparaissent :
      - service/BFMWSService pour les processus métier
      - service/HTMWSService pour les tâches utilisateur
   d. Pour les deux références de service :
      1) Sélectionnez l'une des **Service References**.
      2) Développez la section **Security Request Generator Binding Configuration**.
      3) Développez la sous-section **Token Generator**.
      4) Cliquez sur **Add**. La fenêtre Token Generator s'ouvre.
      5) Dans la zone **Name**, entrez le nom du nouveau générateur de jetons, par exemple "UserNameTokenGeneratorBFM" ou "UserNameTokenGeneratorHTM".
      6) Dans la zone **Token generator class**, vérifiez que la classe de générateur de jetons suivante est sélectionnée : **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**.
      7) Dans la liste déroulante **Security token**, sélectionnez le jeton de sécurité que vous avez créé précédemment.
      8) Cochez la case **Use value type**.

9) Dans la zone **Value type**, sélectionnez **Username Token**. (La valeur que vous avez choisie pour **Username Token** apparaîtautomatiquement dans la zone **Local name**.)

10) Dans la zone **Gestionnaire de rappels**, entrez ″com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler″ (qui vous invite à saisir un nom d'utilisateur et un mot de passe lorsque vous exécutez l'application client) ou ″com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler″.

11) Si vous optez pour **NonPromptCallbackHandler**, vous devez spécifier un nom d'utilisateur et un mot de passe valides dans le descripteur de déploiement.

12) Cliquez sur **OK**.

Tâches associées

«Spécification de l'adresse du noeud final de service Web», à la page 118
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.

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'**éditeur de déploiement** du module.
   b. Cliquez sur l'onglet **WS Extension**.
   c. Sous **Service References**, la référence **Web Service References** suivante peut apparaître :
      - service/BFMWSService pour les processus métier

- service/HTMWSService pour les tâches utilisateur

La liste des références affichées dépend de si BFMWS.wsdl (pour les processus métier) ou HTMWS.wsdl (pour les tâches utilisateur), ou les deux ont été ajoutés lors de la génération du client proxy.

d. Pour les deux références de service :
   1) Sélectionnez l'une des **Service References**.
   2) Développez la section **Request Generator Configuration**.
   3) Développez la sous-section **Security Token**.
   4) Cliquez sur **Ajouter**. La fenêtre Security Token s'affiche.
   5) Dans la zone **Name**, entrez le nom du nouveau jeton de sécurité : **LTPATokenBFM** ou **LTPATokenHTM**.
   6) Dans la liste déroulante **Token type**, sélectionnez **LTPAToken**. (Les valeurs par défaut apparaissent automatiquement dans les zones **URI** et **Local name**).
   7) Cliquez sur **OK**.

3. Créer un générateur de jetons :
   a. Ouvrez l'**éditeur de déploiement** du module.
   b. Cliquez sur l'onglet **WS Binding**.
   c. Sous **Service References**, les mêmes références de service Web que dans l'étape précédente apparaissent :
      - service/BFMWSService pour les processus métier
      - service/HTMWSService pour les tâches utilisateur
   d. Pour les deux références de service :
      1) Sélectionnez une des **Service References**.
      2) Développez la section **Security Request Generator Binding Configuration**.
      3) Développez la sous-section **Token Generator**.
      4) Cliquez sur **Add**. La fenêtre Token Generator s'ouvre.
      5) Dans la zone **Name**, entrez le nom du nouveau générateur de jetons, par exemple ″LTPATokenGeneratorBFM″ ou ″LTPATokenGeneratorHTM″.
      6) Dans la zone **Token generator class**, vérifiez que la classe de générateur de jetons suivante est sélectionnée : **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**.
      7) Dans la liste déroulante **Security token**, sélectionnez le jeton de sécurité que vous avez créé précédemment.
      8) Cochez la case **Use value type**.
      9) Dans la zone **Value type**, sélectionnez **LTPAToken**. (Les valeurs que vous avez choisies pour le **LTPA Token** apparaissent automatiquement dans les zones **URI** et **Local name**).
      10) Dans la zone **Call back handler**, entrez ″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.

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.

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.

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, envoie les messages SOAP au service Web via le protocole HTTP, reçoit les réponses émises par le service Web et traite 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 la liste complète des paramètres de **WSDL.exe**, spécifiez le symbole de ligne de commande **/?** ou reportez-vous à l'aide en ligne de 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:proxyclient.cs c:\ws\bin\HTMWS.wsdl
   ```

2. Compilez le client proxy sous forme de bibliothèque de liaison dynamique (DLL).

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)**.

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: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>
```

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;
```

Tâches associées

«Génération d'un client proxy (.NET)», à la page 137
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)», à la page 138
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.

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);
```

Rechercher 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.

Requêtes de données sur les objets liés aux processus métier et aux tâches

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 = " . ";

   // -- informations sur la colonne de requête
   QueryColumnInfoType[] queryColumnInfo = queryResultSet.QueryColumnInfo;
   if (queryColumnInfo.Length > 0) {
      Console.WriteLine();
      Console.WriteLine("= . QueryColumnInfoType size= " + queryColumnInfo.Length);
      Console.Write( " | tableName ");
```

```
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].tableName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | columnName ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].columnName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | data type ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         QueryColumnInfoTypeType tt = queryColumnInfo[i].type;
         Console.WriteLine( " | " + tt.ToString());
      }
         Console.WriteLine();
   }
   else {
      Console.WriteLine("--> queryColumnInfo= <null>");
   }

   // - valeurs du résultat de la requête
   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.Write(indent +i );
        string[] row = result[i];
        for (int j = 0; j &lt; row.Length; j++ ) {
           Console.Write(" | " + 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.Write("--> 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);
      }
```

Paramètres des requêtes :

Cchaque requête doit spécifier un certain nombre de clauses et paramètres de type SQL.

Une requête est composée des éléments suivants :
- Clause Select
- Clause Where
- Clause Order-by
- Paramètre Skip-tuples
- Paramètre Threshold
- Paramètre Time-zone

Vues prédéfinies pour les requêtes de données sur les objets de processus métier et de tâches utilisateur

Des vues prédéfinies des bases de données sont disponibles pour les objets de processus métier et de tâches utilisateur.

Utilisez ces vues lors de la recherche de données de référence sur ces objets. Lorsque vous utilisez ces vues, vous n'avez pas besoin d'ajouter explicitement des prédicats de jointure pour les colonnes des vues ; ces constructions sont ajoutées automatiquement à votre place. Vous pouvez utiliser la fonction de requête des API de service Web pour rechercher ces données.

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.

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 JMS

Vous pouvez développer des applications client accédant à des applications de processus métier via l'API JMS (Java Messaging Service).

A propos de cette tâche

Introduction à JMS

WebSphere Process Server version 6.1 prend en charge la messagerie asynchrone comme méthode de communication basée sur l'interface de programmation JMS (Java Messaging Service).

JMS constitue un moyen commun pour tous les clients Java (applications client ou applications J2EE) de créer, d'envoyer, de recevoir et de lire des requêtes en tant que messages JMS.

JMS est une interface asynchrone basée sur les messages dont les caractéristiques sont les suivantes :

- Elle utilise la messagerie **point à point ou de publication/d'abonnement.** . Les structures basées sur les messages peuvent envoyer des informations vers d'autres applications sans que celles-ci n'en aient fait la demande explicite. Les mêmes informations peuvent être envoyées à plusieurs abonnés en parallèle. L'interface JMS de Business Process Choreographer prend en charge la messagerie point à point uniquement.
- Elle offre une grande **indépendance dans le rythme**. Les structures JMS fonctionnent de manière asynchrone mais elles peuvent également simuler un mode de requête/réponse synchrone. Les systèmes source et cibletravaillent ainsi simultanément sans devoir s'attendre l'un l'autre. Cette caractéristique est très utile pour Business Process Choreographer, car elle permet d'interagir de manière asynchrone avec des processus métier de longue durée.
- Elle prend en charge les **transactions**. Les transactions permettent aux applications client applications de gérer des groupes de messages envoyés ou reçus comme unité atomique unique. Les transactions JMS s'exécutent au sein de la transaction du serveur. Pour l'interface JMS de Business Process Choreographer, vous envoyez et recevez généralement un message unique par transaction.
- **Elle garantit la livraison des informations.** . Les structures JMS peuvent gérer des messages en mode transactionnel et garantir la livraison des messages (sans garantie du délai de livraison). Pour Business Process Choreographer, cette fonctionnalité est particulièrement importante car elle concerne les processus métier.
- Garantit l'**interopérabilité entre des structures hétérogènes.** . Les applications source et cible peuvent fonctionner dans des environnements hétérogènes sans devoir gérer les problèmes de communication et d'exécution liés à des structures différentes.
- **Fluidifie les échanges**. Le passage en mode message permet une meilleure granularité des informations échangées.

Exigences en termes de 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 de processus métier doivent être définies à l'aide du style ″document/littéral encapsulé″ 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 ?

Accès à l'interface JMS

Pour pouvoir envoyer et recevoir des messages via l'interface JMS, une application doit d'abord établir une connexion vers le bus BPC.cellname.Bus, créer une session puis générer des expéditeurs et des destinataires de messages.

A propos de cette tâche

Le serveur de processus accepte les messages JMS (Java Message Service) conformes au paradigme point-à-point. Une application qui envoie ou qui reçoit des messages JMS doit exécuter les opérations suivantes :

Dans l'exemple qui suit, le client JMS s'exécute dans un environnement géré (EJB, client d'application ou conteneur de client Web). Si vous souhaitez exécuter le client JMS dans un environnement J2SE, reportez-vous à ″IBM Client for JMS on J2SE with IBM WebSphere Application Server″ à la page http://www-1.ibm.com/support/docview.wss?uid=swg24012804.

Procédure

1. Etablissez une connexion avec le bus BPC.*cellname*.Bus. Aucune fabrique de connexions préconfigurée n'existe pour les requêtes d'une application client : l'application client peut soit utiliser l'API JMS ReplyConnectionFactory, soit créer sa propre fabrique de connexions, auquel cas elle fait appel à la fonction de recherche de l'interface JNDI (Java Naming and Directory Interface) pour extraire 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. Dans l'exemple suivant, l'application client crée sa propre fabrique de connexions appelée ″jms/clientCF″.

   ```
   //Obtenir le contexte JNDI initial par défaut.
   Context initialContext = new InitialContext();

   // Rechercher la fabrique de connexions.
   // Créer une fabrique de connexions qui se connecte au bus BPC.
   // L'appeler par exemple "jms/clientCF".
   // Configurer également un alias d'authentification approprié.
   ConnectionFactory connectionFactory =
               (ConnectionFactory)initialcontext.lookup("jms/clientCF");

   // Etablir la connexion.
   Connection connection = connectionFactory.createConnection();
   ```

2. Créez une session pour pouvoir créer des expéditeurs et des destinataires de messages.

   ```
   // Créer une session de transaction avec accusé de réception automatique.
   Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
   ```

3. Créez un expéditeur de messages qui enverra 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.

   ```
   // Rechercher la destination de la file d'entrée de Business Process
      Choreographer à laquelle
   // envoyer des messages.
   Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");

   // Créer un expéditeur de messages.
   MessageProducer producer = session.createProducer(sendQueue);
   ```

4. Créez un destinataire de messages qui recevra les réponses. Le nom de recherche JNDI de la destination de la réponse peut indiquer une destination

définie par l'utilisateur ou la destination de réponse par défaut jms/BFMJMSReplyQueue. Dans les deux cas, la destination de réponse doit se trouver sur bus BPC.<cellname>.Bus.

```
// Rechercher la destination de la file d'attente de réponses
Queue  replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

// Créer un expéditeur de messages.
MessageConsumer consumer = session.createConsumer(replyQueue);
```

5. Envoyez un message.

```
// Etablir la connexion.
connection.start();

// Créer un message - pour des exemples, voir les descriptions des
   tâches - et l'envoyer.
// Cette méthode est définie ailleurs...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Définir l'en-tête JMS obligatoire.
// targetFunctionName est le nom d'opération de l'API JMS
// (par exemple getProcessTemplate ou sendMessage)
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Définir la file d'attente de réponse ; cette étape est obligatoire si
   replyQueue
// n'est pas la file d'attente par défaut (comme c'est le cas dans
   cet exemple).
requestMessage.setJMSReplyTo(replyQueue);

// Envoyer le message.
producer.send(requestMessage);

// Obtenir l'ID du message.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. Recevez la réponse.

```
// Recevoir le message de réponse et l'analyser.
TextMessage  replyMessage = (TextMessage) consumer.receive();

// Obtenir la charge.
String payload = replyMessage.getText();

session.commit();
```

7. Mettez fin à la connexion et libérez les ressources.

```
 // Gestion finale. Libérer les ressources.
session.close();
connection.close();
```

Remarque : Il n'est pas nécessaire de mettre fin à la connexion après chaque transaction. Une fois qu'une connexion a été établie, vous pouvez échanger autant de messages de requête et de réponse que vous le souhaitez avant de l'interrompre. L'exemple montre un cas simple d'appel unique dans une méthode métier unique.

Structure d'un message JMS de Business Process Choreographer

Une structure doit être prédéfinie pour l'en-tête et le corps de chaque message JMS.

Un message JMS (Java Message Service) se compose de :

- Un en-tête de message destiné à l'identification du message et aux informations de routage.
- Le corps (charge) du message, dans lequel figure le contenu.

Business Process Choreographer prend en charge uniquement les messages au format texte.

En-tête du message

JMS permet aux clients d'avoir accès à un certain nombre de zones d'en-tête de message.

Les zones suivantes peuvent être définies par un client JMS de Business Process Choreographer :

- **JMSReplyTo**

  La destination à laquelle la réponse à la requête doit être envoyée. Si cette zone n'est pas indiquée dans le message de requête, la réponse est envoyée à la destination de réponse de l'interface d'exportation par défaut (une interface d'exportation est une interface client qui affiche un composant de processus métier). Pour connaître cette destination, entrez `initialContext.lookup("jms/BFMJMSReplyQueue");`
- **TargetFunctionName**

  Le nom de l'opération WSDL, par exemple "queryProcessTemplates". Cette zone doit toujours être définie. Notez que le nom TargetFunctionName désigne l'opération de l'interface générique des messages JMS décrite ici. Ne confondez pas cette opération avec les opérations proposées par des tâches ou des processus concrets et pouvant être appelées indirectement telles que les opérations **call** ou **sendMessage**.

Un client Business Process Choreographer peut aussi accéder aux zones d'en-tête suivantes :

- **JMSMessageID**

  Identifie un message de façon unique. Définie par le fournisseur JMS lors de l'envoi du message. Si le client définit l'identificateur JMSMessageID avant d'envoyer le message, le JMSMessageID est remplacé par le fournisseur JMS. Si l'identificateur du message est nécessaire à des fins d'authentification, le client peut extraire le JMSMessageID après avoir envoyé le message.
- **JMSCorrelationID**

  Relie les messages entre eux. Ne définissez pas cette zone. Dans Business Process Choreographer, un message de réponse contient le JMSMessageID du message de requête.

Chaque message de réponse contient les zones d'en-tête JMS suivantes :

- **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 grave se produit au cours du traitement d'un message de requête JMS, il se peut que l'exécution s'interrompe et que la transaction qui traite le message de requête soit annulée. Le message de requête JMS est envoyé à nouveau. Si l'incident se produit tôt, c'est-à-dire pendant la phase de traitement du message de l'exportation SCA (par exemple au cours de la désérialisation du message), il est possible de procéder à d'autres tentatives à concurrence du nombre maximal de tentatives d'envoi indiqué par la destination de réception de l'exportation SCA. Une fois le nombre maximal de tentatives d'envoi atteint, le message de requête s'ajoute à la destination d'exception du système du bus de Business Process Choreographer. Si l'échec se produit pendant le traitement de la requête par le composant SCA de Business Flow Manager, le message est géré par l'infrastructure de gestion des événements ayant échoué, il peut donc aboutir dans la base de données de gestion des événements ayant échoué si les diverses tentatives ne permettent pas de résoudre le problème.

Corps du message

Le corps du message JMS est une chaîne contenant un document XML représentant l'élément encapsuleur document/littéral de l'opération.

Exemple simple d'un corps de message de requête correct :

```
<?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>
```

Autorisation pour l'affichage JMS

Pour pouvoir utiliser l'interface JMS, les paramètres de sécurité doivent être activés dans WebSphere Application Server.

Une fois le conteneur de processus métier installé, le rôle **JMSAPIUser** doit être mappé vers un ID utilisateur. Cet ID utilisateur permet d'émettre toutes les requêtes API JMS. Par exemple, si l'utilisateur **JMSAPIUser** est mappé vers ″Utilisateur A″, toutes les requêtes APIJMS apparaissent comme provenant de cet utilisateur.

Le rôle **JMSAPIUser** doit disposer des droits suivants :

Requête	Autorisation obligatoire
forceTerminate	Administrateur de processus
sendEvent	Propriétaire d'activité ou administrateur de processus potentiel

Remarque : Aucune autorisation spéciale n'est requise pour les autres requêtes.

Une personne dotée du rôle d'administrateur de processus métier dispose de droits spéciaux. L'administrateur de processus métier est un rôle spécial différent 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 le

supprimez, la navigation de ce processus s'interrompt. Vous recevrez l'exception suivante dans le fichier journal du système :

```
no unique ID for: <ID utilisateur>
```

Présentation de l'API JMS

L'interface des messages JMS (appelée ci-dessous ″API JMS″)vous permet de développer des applications client accédant aux processus métier de manière asynchrone dans l'environnement Business Process Choreographer.

L'API JMS permet aux applications client d'interagir en mode asynchrone avec des microflux et des processus de longue durée.

L'API JMS présente la même interface que l'API de services Web, à ces exceptions près :

- Dans l'API des services Web, l'opération `call` ne peut être utilisée que pour appeler des microflux. Dans l'API JMS, l'opération `call` permet en revanche d'appeler des microflux et des processus de longue durée.
- Les opérations qui suivent ne sont pas proposées par l'API JMS :
  - L'opération `callAsync` (ainsi que l'opération de rappel associée).
  - Les opérations `completeAndClaimSuccessor` et `getParticipatingTask`.

Exemple - exécution d'un processus de longue durée

Pour pouvoir exécuter des processus de longue durée sur une application client, procédez comme suit :

1. Configurez l'environnement JMS comme décrit dans «Accès à l'interface JMS», à la page 148.
2. Obtenez la liste des définitions des processus installées en procédant comme suit :
   - Exécutez `queryProcessTemplates`
   - Une liste des objets **ProcessTemplate** est renvoyée.
3. Obtenez la liste des activités de démarrage (de réception ou de sélection associées à `createInstance="yes"`) en procédant comme suit :
   - Exécutez `getStartActivities`.
   - Une liste des objets **InboundOperationTemplate** est renvoyée.
4. Créez un message d'entrée. Des artefacts prédéployés propres au processus peuvent être nécessaires à cette opération propre à chaque environnement.
5. Créez une instance de processus :
   - Exécutez la commande `sendMessage`.

   Dans l'API JMS, vous aurez peut-être besoin de l'opération `call` pour interagir avec les opérations de requête-réponse de longue durée fournies par un processus métier. Cette opération renvoie un résultat ou une erreur à la destination de réponse indiquée, même après un laps de temps très long. Ainsi, dans le cas de l'opération `call`, il est inutile d'ajouter les opérations `query` et `getOutputMessage` pour recevoir le message de sortie ou d'erreur.
6. Facultatif : obtenez les messages de sortie des instances de processus en exécutant à plusieurs reprisesles étapes suivantes :
   - Exécutez `query` pour obtenir l'état terminé de l'instance de processus.
   - Exécutez `getOutputMessage` .

7. Facultatif : exécutez d'autres opérations proposées par le processus :
   - `getWaitingActivities` ou `getActiveEventHandlers` pour obtenir une liste des objets **InboundOperationTemplate**.
   - Créez des messages d'entrée.
   - Envoyez des messages à l'aide de `sendMessage`
8. Facultatif : obtenez et définissez les propriétés personnalisées définies pour le processus ou les activités qu'il contient à l'aide de `getCustomProperties` et de `setCustomProperties`.
9. Facultatif : mettez fin aux activités liées à l'instance de processus :
   - Exécutez `delete` et `terminate` pour mettre fin à l'exécution du processus de longue durée.

Développement d'applications JMS

Les applications client JMS sont développées en Java à l'aide de l'environnement J2EE (Java 2 Enterprise Edition).

A propos de cette tâche

Les applications client JMS échangent des requêtes et des messages de réponse avec l'API JMS. Pour créer un message de requête, l'application client remplit le corps d'un message JMS TextMessage avec un élément XML représentant l'encapsuleur document/littéral de l'opération correspondante.

Copie d'artefacts

Un certain nombre d'artefacts peuvent être copiés depuis l'environnement WebSphere afin de créer des applications client JMS.

L'utilisation de ces artefacts est obligatoire uniquement si le corps du message JMS est créé à l'aide de BOXMLSerializer.

Deux méthodes permettent d'obtenir ces artefacts :

- Publiez et exportez-les depuis l'environnement WebSphere Process Server.

  Dans WebSphere Process Server 6.1, tous les artefacts client se trouvent dans le répertoire *racine_installation*\ProcessChoreographer\client. Pour l'API JMS, ces artefacts sont les suivants :

  BFMIF.wsdl

  BFMIF.xsd

  BPCGen.xsd
- Copiez les fichiers depuis le CD client WebSphere Process Server.

Publication d'artefacts depuis l'environnement de serveurs :

Vous pouvez publier un certain nombre d'artefacts à partir de l'environnement de serveurs WebSphere pour faciliter le développement des applications client accédant à l'API JMS.

A propos de cette tâche

Dans WebSphere Process Server 6.1, tous les artefacts client se trouvent dans le répertoire *was_home*\ProcessChoreographer\client. Pour l'API JMS, ces artefacts sont les suivants :

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

Une fois ces artefacts publiés, copiez-les dans votre environnement de programmation client.

Copie de fichiers depuis le CD client :

Les fichiers requis pour accéder à l'API JMS sont disponibles sur le CD client de WebSphere Process Server.

Procédure

1. Ouvrez le CD client et naviguez jusqu'au répertoire `ProcessChoreographer\client`.
2. Copiez les fichiers nécessaires à votre environnement de développement d'applications client.

   Dans WebSphere Process Server 6.1, tous les artefacts client se trouvent dans le répertoire \ProcessChoreographer\client. Pour l'API JMS, ces artefacts sont les suivants :

   BFMIF.wsdl

   BFMIF.xsd

   BPCGen.xsd

Vérification des exceptions métier dans le message de réponse

Les applications client JMS doivent vérifier la présence d'exceptions métier dans l'en-tête de message de tous les messages de réponse.

A propos de cette tâche

Une application client JMS doit d'abord vérifier la propriété **IsBusinessException** dans l'en-tête du message de réponse.

Par exemple :

```
// recevoir un message de réponse
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject().receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse est un incident métier
  // toutes les api peuvent se terminer avec un message processFaultMsg
  // l'api d'appel se termine aussi avec un message businessFaultMsg
 }
else {
  // strResponse est le message de sortie
}
```

Développement d'applications Web pour les processus métier et tâches utilisateur à l'aide de composants JSF

Business Process Choreographer offre plusieurs composants JSF (JavaServer Faces). 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.

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 l'application est déployée sur un serveur distant, ajoutez les fichiers suivants. Ils sont nécessaires pour accéder à distance aux API de Business Process Choreographer.

   - bpe137650.jar
   - task137650.jar

   Dans WebSphere Process Server, ces fichiers se trouvent dans le répertoire suivant :

   - Sous Windows : *racine_installation*\ProcessChoreographer\client
   - Sous 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 aux bibliothèques de balises requises pour les applications dans les fichiers JSP (JavaServer Pages). 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 que les erreurs soient correctement traitées, vous devez également définir un bean d'erreur et une cible de navigation pour la page d'erreurs du fichier de configuration JSF.

      ```
      <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, changez dans la cellule les noms JNDI (Java Naming and Directory Interface) des ressources cible en des valeurs dans lesquelles les API Business Flow Manager et Human Task Manager peuvent se trouver.

   - Si les conteneurs de processus métier sont configurés sur un autre serveur de la même cellule gérée, les noms ont la structure suivante :

     cellule/noeuds/*nom_noeud*/serveurs/*nom_serveur*/com/ibm/bpe/api/BusinessManagerHome
     cellule/noeuds/*nom_noeud*/serveurs/*nom_serveur*/com/ibm/task/api/HumanTaskManagerHome

   - Si les conteneurs de processus métier sont configurés sur un autre cluster de la même cellule, les noms ont la structure suivante :

     cellule/clusters/*nom_cluster*/com/ibm/bpe/api/BusinessFlowManagerHome
     cellule/clusters/*nom_cluster*/com/ibm/task/api/HumanTaskManagerHome

   Mappez les références EJB avec les noms JNDI ou ajoutez manuellement les références au fichier ibm-web-bnd.xmi.

   Le tableau suivant dresse la liste des liaisons de référence et leurs mappages par défaut.

Tableau 33. Mappage des liaisons de référence aux noms JNDI

Liaison de référence	Nom JNDI	Commentaires
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean session distant
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean session local
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean session distant
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean session local

Résultats

L'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 utilisés pour déployer les JSP vers les mêmes serveurs que ceux vers lesquels le client JSF personnalisé est mappé.

Concepts associés

«Traitement des erreurs dans les composants JSF», à la page 159
Les composants JSF exploitent le 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.

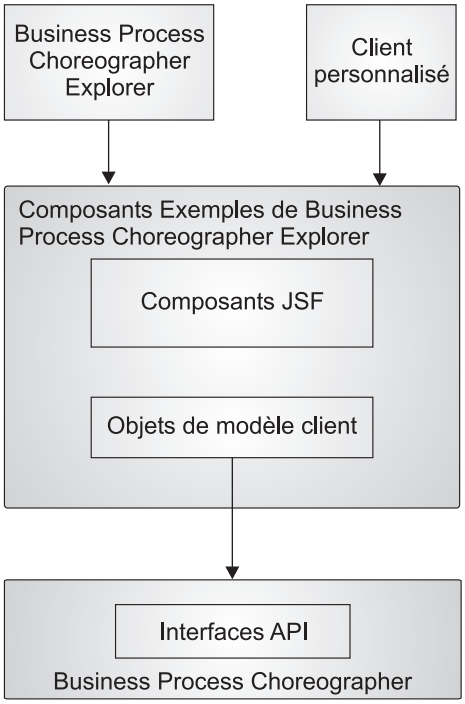
Tâches associées

«Accès à l'interface distante du bean session», à la page 24
Une application client EJB accède à l'interface distante du bean session par le biais de l'interface home distante du bean.

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 listes 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.

Traitement des erreurs dans les composants JSF

Les composants JSF exploitent le 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 getExceptionMessage
     * 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
     * de toutes les exceptions imbriquées.
     */
    public String getAllExceptionStacks();
}
```

Concepts associés

«Traitement des erreurs dans le composant List», à la page 165
Lorsque vous utilisez le composant List pour afficher des listes dans l'application JSF, vous pouvez tirer parti des fonctions de traitement des erreurs fournies par la classe com.ibm.bpe.jsf.handler.BPCListHandler.

Convertisseurs et libellés par défaut pour les objets de modèle client

Les objets de modèle client implémentent les interfaces correspondantes de l'API de Business Process Choreographer.

Le composant List et le composant Details fonctionnent sur tous les beans. Vous pouvez afficher toutes les propriétés d'un bean. Cependant, si vous souhaitez définir les convertisseurs et les libellés associés aux 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 libellés, vous pouvez définir un convertisseur et des libellés par défaut pour les propriétés en définissant les méthodes statiques suivantes. Vous pouvez choisir l'une des méthodes ci-dessous :

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

Les tables suivantes montrent les objets de modèle client qui implémentent les classes API Business Flow Manager et Human Task Manager correspondantes et qui fournissent les libellés et le convertisseur par défaut pour les propriétés. Cet encapsulage des interfaces fournit des libellés 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 34. 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

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 listes.

Vous pouvez ajouter n'importe quel type de logique de requête à l'application JSF tant que le résultat de la requête peut être représenté sous la forme d'une 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 listes.

   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.
    //
    QueryResultSet queryResult = taskService.query(
     "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND, TASK.STATE, TASK.TYPE,"
      + "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)",
      (String)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 listes 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

«Informations de fuseau horaire propres à l'utilisateur», à la page 165
Les composants JSF (JavaServer Faces) comprennent un utilitaire réservé au traitement des informations spécifiques au fuseau horaire de l'utilisateur dans le composant List.

Référence associée

«Composant List : définitions de balises», à la page 166
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 entités sélectionnées dans la liste associée et fournit un système de notification permettant d'associer les entrées de la liste aux pages de détails relatives aux divers 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 se déclenche lorsque l'utilisateur clique sur un lien de la liste. Des liens s'affichent pour toutes les colonnes pour lesquelles l'attribut **action** est défini. La valeur associée à 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 execute.

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 libellés par défaut

Les éléments renvoyés par une requête doivent être des beans et leur classe doit être identique au type indiqué dans la définition 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 le libellé comme libellé par défaut et le convertisseur SimpleConverter comme convertisseur par défaut de la propriété. Vous pouvez remplacer ces paramètres par les attributs **label** et **converterID** de la balise `bpe:list`. Pour plus d'informations, voir la documentation Java relative à l'interface SimpleConverter et à la classe ColumnTag.

Informations de fuseau horaire propres à l'utilisateur

Les composants JSF (JavaServer Faces) comprennent un utilitaire réservé au traitement des informations spécifiques au fuseau horaire de 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 s'exécute.

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();
}
```

Traitement des erreurs dans le composant List

Lorsque vous utilisez le composant List pour afficher des listes dans l'application JSF, vous pouvez tirer parti des fonctions de traitement des erreurs fournies par la classe com.ibm.bpe.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.bpe.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.bpe.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.

Concepts associés

«Traitement des erreurs dans les composants JSF», à la page 159
Les composants JSF exploitent le 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.

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 pour chacun des objets.

Tableau 35. Attributs bpe:list

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de style CSS (feuille de styles en cascade) pour l'affichage des boutons de 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 choisie est `true`, la colonne de la case à cocher s'affiche.
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 com.ibm.bpe.jsf.handler.BPCListHandler.
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 style CSS pour l'affichage du tableau contenant titres, lignes et boutons de pagination.

Tableau 36. Attributs bpe:column

Attribut	Obligatoire	Description
action	non	Si cet attribut est indiqué, un lien s'affiche dans la colonne. Une méthode d'action JSF (JavaServer Faces) ou la Faces cible de navigation se déclenche lors d'un clic sur ce lien. Une méthode d'action JSF possède la signature suivante : String method().
converterID	non	FacesID convertisseur utilisé pour convertir la valeur de la propriété. Si cet attribut n'est pas défini, n'importe quel Faces ID convertisseur fourni par le modèle de cette propriété est utilisé.

Tableau 36. Attributs bpe:column (suite)

Attribut	Obligatoire	Description
label	non	Expression littérale ou expression de liaison de valeur utilisée comme libellé pour l'en-tête de colonne ou pour la cellule de la ligne d'en-tête de tableau. Si cet attribut n'est pas défini, n'importe quel libellé fourni par le modèle de cette propriété est utilisé.
name	oui	Nom de la propriété affiché dans cette colonne.

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 **model**. 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 **model** 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. Il peut s'agir de n'importe quelle classe JavaBeans. Si le bean fournit le convertisseur et les libellés des propriétés par défaut, le convertisseur et le libellé sont utilisés pour l'affichage de la même façon que pour 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).

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 détails ne contient pas de balise d'attribut, elle affiche tous les attributs disponibles de l'objet modèle.

Tableau 37. Attributs bpe:details

Attribut	Obligatoire	Description
columnClasses	non	Liste de classes de styles CSS (feuille de style en cascade) délimitée par des virgules et utilisée pour l'affichage de colonnes.
id	non	ID du composant JavaServer Faces.
model	oui	Liaison de valeur d'un bean géré de la classe com.ibm.bpe.jsf.handler.BPCDetailsHandler.
rowClasses	non	Liste de classes de styles CSS délimitée par des virgules et utilisée pour l'affichage de lignes.
styleClass	non	Classe CSS utilisée pour l'affichage de l'élément HTML.

Tableau 38. 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.

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 l'application JSF.

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 avez modifié la configuration de l'un de ces gestionnaires ou utilisé un autre bean géré, ajoutez un bean géré implémentant l'interface ItemProvider vers le fichier de configuration JSF.

3. Ajoutez le code implémentant les commandes personnalisés vers l'application JSF.

   Le fragment de code suivant 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
            HTMConnection.
         // 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("{htmConnection}");
         HTMConnection htmConnection = (HTMConnection) 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. 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. 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 commande n'est associée à l'attribut **commandClass**, l'action est appelée immédiatement. Dans l'exemple, c'est l'expression de valeur JSF `#{TaskInstanceList.refreshList}` qui est appelée pour la commande refresh à la place d'une commande.

Résultats

Votre application JSF contient à présent une page JSP implémentant une barre de commandes personnalisée.

Référence associée

«Composant CommandBar : définitions de balises», à la page 174
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.

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 39. Attributs bpe:commandbar

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de style CSS (feuille de styles en cascade) 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 style CSS utilisée pour l'affichage de la barre de commandes.

Tableau 40. Attributs bpe:command

Attribut	Obligatoire	Description
action	non	Méthode d'action JSF (JavaServer Faces) ou la Faces cible de navigation à déclencher à l'aide du bouton de commande. La cible de navigation renvoyée par l'action écrase toutes les autres règles de navigation. L'action est appelée lorsqu'aucune exception n'est générée ou lorsqu'une exception ErrorsInCommandException est générée par la commande.
commandClass	non	Nom de la classe de commande. Une instance de la classe est créée par le composant CommandBar. Elle s'exécute en cas de sélection du bouton de commande.
commandID	oui	ID de la commande.
context	non	Objet qui fournit un contexte pour les commandes contenant un attribut **commandClass**. L'objet de contexte est extrait lors du premier accès à la barre de commande.
immediate	non	Indique quand la commande est déclenchée. Si la valeur de cet attribut est `true`, la commande est déclenchée avant que l'entrée de la page ne soit traitée. La valeur par défaut est `false`.
label	oui	Libellé du bouton affiché dans la barre de commandes.
rendered	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 de style CSS utilisée pour l'affichage du bouton. Ce style se substitue au style de bouton défini pour la barre de commandes.

Ajout du composant Message à une application JSF

Le composant Message de Business Process Choreographer Explorer 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.bpe.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 gestionnaires 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 à la liste itemListener dans le gestionnaire de listes approprié du 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.

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 41. Attributs bpe:form

Attribut	Obligatoire	Description
id	non	ID du composant JavaServer Faces.
model	oui	Expression de liaison de valeur qui fait référence à un objet `commonj.sdo.DataObject` ou à un objet `com.ibm.bpc.clientcore.MessageWrapper`.
readOnly	non	Si cet attribut est réglé sur `true`, un formulaire s'affiche en lecture seule. Par défaut, cet attribut est réglé sur `false`.
simplification	non	Si cet attribut est réglé sur `true`, les propriétés qui contiennent des types simples et dont la cardinalité est égale à zéro ou un s'affichent. Par défaut, cet attribut est défini sur `true`.
style4validinput	non	Style CSS (feuille de styles en cascade) pour l'affichage de valeur d'entrée valide.
style4invalidinput	non	Style CSS pour l'affichage de valeur d'entrée incorrecte.
styleClass4invalidInput	non	Nom de la classe de styles CSS pour l'affichage d'une valeur d'entrée incorrecte.
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 la classe de styles CSS pour l'affichage d'une valeur d'entrée correcte.

Développement de 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 les pages JSP pour fournir des formulaires personnalisés d'entrée et de sortie.

A propos de cette tâche

Pour inclure des pages JSP (JavaServer Pages) 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 les messages d'entrée et de sortie associés ainsi que pour un rôle utilisateur spécifique ou 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, des fragments de tous les libellés et de toutes 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 de mappe permettant de mettre en correspondance 'expression XPath XML des propriétés incorrectes vers les exceptions de validation qui se sont produites.

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 pouvoir accéder aux données des messages. Utilisez les fragments de code Java ou le langage d'exécution JSP pour accéder à ces données. Les données des formulaires sont accessibles via le contexte de requête.
2. Affectez les pages JSP à une tâche.

   Ouvrez la tâche manuelle dans l'éditeur de tâches manuelles. 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 Business Process Choreographer Explorer 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 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 pages JSP sont déployées séparément, rendez-les accessibles par le serveur où Business Process Choreographer Explorer ou le client est déployé.

   Si vous utilisez des pages JSP personnalisées pour les messages de processus et de tâche, vous devez mapper les modules Web utilisés pour déployer ces pages vers les mêmes serveurs que ceux vers lesquels le client JSF personnalisé est mappé.

Résultats

Les formulaires personnalisés s'affichent dans Business Process Choreographer Explorer lors de l'exécution.

Fragments JSP définis par l'utilisateur

Les fragments JSP (JavaServer Pages) 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>
```

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 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 module d'extension traitant les résultats renvoyés à partir de 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.

Vous pouvez enregistrer vos modules d'extension sur différents niveaux, pour toutes les tâches sur un niveau global, pour les tâches d'un composant d'application, pour toutes les tâches associées a un modèle de tâche ou pour une seule instance de tâche.

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 module d'extension de gestionnaire d'événements d'API pour créer des modules d'extension 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 APIEventHandlerPlugin2 ou étend la classe d'implémentation APIEventHandler. Cette classe peut appeler les méthodes d'autres classes.
   - Si vous utilisez l'interface APIEventHandlerPlugin2, vous devez implémenter toutes les méthodes de l'interface APIEventHandlerPlugin2 et de l'interface APIEventHandlerPlugin.
   - Si vous étendez la classe d'implémentation de SPI, écrasez les méthodes dont vous avez besoin.

   Cette classe s'exécute dans le contexte d'une application EJB (Enterprise JavaBeans) J2EE (Java 2 Enterprise Edition). Assurez-vous que cette classe et ses classes auxiliaires suivent la spécification EJB.

   Conseil : Pour appeler l'interface HumanTaskManagerService à partir de cette classe, n'appelez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action aboutit à un blocage de base de données.
2. 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 indépendant que vous enregistrez comme une bibliothèque partagée.

3. Créez un fichier de configuration de fournisseur de services pour le module d'extension dans le répertoire META-INF/services/ du fichier JAR.

   Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le module d'extension. 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 extension`*`APIEventHandlerPlugin`, où *nom_module extension* est le nom du module d'extension.

      Par exemple, si votre module d'extension s'appelle `Customer` et qu'il implémente l'interface com.ibm.task.spi.APIEventHandlerPlugin, le nom du fichier de configuration est`com.ibm.task.spi.`**`Customer`**`APIEventHandlerPlugin`.

   b. Dans la première ligne du fichier qui n'est ni une ligne commentaire ni une ligne vide, spécifiez le nom qualifié complet de la classe du module d'extension que vous avez créé à l'étape 1.

      Par exemple, si la classe de votre module d'extension 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 module d'extension gérant les événements d'API et un fichier de configuration du fournisseur de services pouvant être utilisé pour charger le module d'extension.

Conseil : 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 modules d'extension portent le même nom (`Customer` comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux modules d'extension 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 module d'extension et les classes auxiliaires dans un seul fichier JAR.

Que faire ensuite

Vous devez à présent installer et enregistrer le module d'extension pour qu'il soit accessible au conteneur de tâche utilisateur lors de l'exécution. Vous pouvez enregistrer les gestionnaires d'événements d'API avec une instance de tâche, un modèle de tâche ou un composant d'application.

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 APIEventHandlerPlugin2, 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.

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 au choix faire appel à l'interface NotificationEventHandlerPlugin, ou 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 EJB (Enterprise JavaBeans) J2EE (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.

   Conseil : Si vous souhaitez appeler l'interface HumanTaskManagerService à partir de cette classe, n'appelez pas de méthode mettant à jour la tâche ou l'escalade qui a généré l'événement. En effet, une telle opération créerait un blocage de la base de données.
2. 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.
3. 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_extension`*`NotificationEventHandlerPlugin`, 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.`**`HelpDeskRequest`**`NotificationEventHandlerPlugin`.

b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire, 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.

Conseil : 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 modules d'extension portent le même nom (`Customer` comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux modules d'extension 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 module d'extension et les classes auxiliaires dans un seul fichier JAR.

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.

Création de modules d'extension pour le post-traitement des résultats d'une requête utilisateur

La résolution du personnel renvoie une liste d'utilisateurs auxquels un rôle spécifique est affecté, par exemple, le propriétaire potentiel d'une tâche. Vous pouvez créer un module d'extension pour modifier les résultats des requêtes utilisateur renvoyés par la résolution des utilisateurs. Par exemple, pour améliorer l'équilibrage de charge, vous pourriez avoir un module d'extension 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 module d'extension de post-traitement ; autrement dit, le module d'extension doit gérer les résultats des requêtes utilisateur provenant de toutes les tâches. Votre module d'extension 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.

L'exécution des modules d'extension n'ayant lieu qu'après la résolution des utilisateurs, toutes les règles de confidentialité ou de sécurité éventuellement définies ont déjà été appliquées. Le module d'extension reçoit des informations sur les utilisateurs qui ont été supprimés pendant la résolution utilisateur (dans la clé de mappe HTM_REMOVED_USERS). Vous devez vous assurer que le module d'extension utilise ces informations de contexte pour préserver les règles de confidentialité ou de sécurité dont vous disposez éventuellement.

Pour implémenter les résultats des requêtes utilisateur, 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 module d'extension destiné au post-traitement des résultats d'une requête utilisateur.

Procédure

1. Rédigez une classe qui implémente l'interface StaffQueryResultPostProcessorPlugin.

   Vous devez implémenter toutes les méthodes de l'interface. Cette classe peut appeler les méthodes d'autres classes.

   Cette classe s'exécute dans le contexte d'une application EJB (Enterprise JavaBeans) J2EE (Java 2 Enterprise Edition). Assurez-vous que cette classe et ses classes auxiliaires suivent la spécification EJB.

   Conseil : Pour appeler l'interface HumanTaskManagerService à partir de cette classe, n'appelez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action aboutit à un blocage de base de données.

   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 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 indépendant que vous enregistrez comme une bibliothèque partagée.

3. Créez un fichier de configuration de fournisseur de services pour le module d'extension dans le répertoire META-INF/services/ du fichier JAR.

   Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le module d'extension. 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_extension`*`StaffQueryResultPostProcessorPlugin`, où *nom_module_extension_* correspond au nom du module d'extension.

      Par exemple, si votre module d'extension 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.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`.

   b. Dans la première ligne du fichier qui n'est ni une ligne commentaire ni une ligne vide, spécifiez le nom qualifié complet de la classe du module d'extension que vous avez créé à l'étape 1.

      Par exemple, si la classe de votre module d'extension 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 disposez d'un fichier JAR pouvant être installé qui contient un module d'extension qui traite les résultats de la requête utilisateur et d'un fichier de configuration de fournisseur de service qui peut être utilisé pour charger le module d'extension.

4. Installez le module d'extension.

   Vous ne pouvez avoir qu'un seul module d'extension de post-traitement pour les résultats de requête utilisateur. Vous devez installer le module d'extension en tant que bibliothèque partagée.

5. Enregistrez le module d'extension.

   a. Dans la console d'administration, accédez à la page Propriétés personnalisées de Human Task Manager (**Serveurs d'application** → ***nom_serveur*** → **Conteneur de tâches utilisateur** → **Propriétés personnalisées**).

   b. Ajoutez une propriété personnalisée nommée **Staff.PostProcessorPlugin** et ainsi que la valeur du nom que vous avez donné à votre module d'extension (`MyHandler` dans cet exemple).

Installation des modules d'extension

Pour pouvoir utiliser un module d'extension, vous devez l'installer de sorte qu'il soit accessible au conteneur de tâches.

A propos de cette tâche

Le mode d'installation du module d'extension varie selon que celui-ci est exploité uniquement par une 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 de descripteurs de déploiement de WebSphere Integration Developer, installez le fichier JAR pour votre module d'extension en tant que fichier JAR de l'utilitaire de projets pour l'application J2EE du module EJB (Enterprise 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 aux applications qui ont besoin d'accéder au gestionnaire d'événements. Pour rendre le fichier JAR accessible dans un environnement de déploiement réseau, distribuez le fichier JAR sur chaque serveur manuellement, puis installez la bibliothèque partagée une fois pour chaque cellule.

Que faire ensuite

Vous pouvez, maintenant, enregistrer le module d'extension.

Enregistrement des modules d'extension

Vous pouvez enregistrer vos modules d'extension sur différents niveaux dans la hiérarchie de l'artefact du conteneur de la tâche. Par exemple, pour toutes les tâches sur un niveau global, pour les tâches d'un composant d'application, pour toutes les tâches associées à un modèle de tâche ou pour une seule instance de tâche.

A propos de cette tâche

Lorsque vous enregistrez plusieurs modules d'extension, la configuration est prise en charge. Cela signifie qu'un module d'extension enregistré sur un niveau inférieur de la hiérarchie de l'artefact du conteneur de la tâche, comme une instance de tâche, est utilisé à la place du module d'extension enregistré sur un niveau plus élevé, comme un modèle de tâche ou un composant d'application. La configuration est prise en charge pour tous les niveaux de la hiérarchie. Le conteneur de tâche utilise le module d'extension qui est enregistré sur le plus niveau le plus faible de la hiérarchie.

Vous pouvez enregistrer un module d'extension en suivant l'une des procédures suivantes.

- Enregistrez le module d'extension dans le modèle de tâche.

  Dans l'éditeur de tâches de WebSphere Integration Developer, sur la page Détails de la zone relative aux propriétés de la tâche, indiquez le nom du gestionnaire d'événements dans la zone **Nom du gestionnaire d'événements**.
- Enregistrez le module d'extension pour les tâches ad-hoc ou pour les modèles de tâche que vous avez créés lors de l'exécution.

  Utilisez la méthode setEventHandlerName de la classe TTask pour enregistrer le nom du gestionnaire d'événements.
- Modifiez le gestionnaire d'événements enregistré 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.
- Enregistrez le module d'extension dans un niveau global.

  Sur la page Propriétés personnalisées associée au conteneur de la tâche utilisateur, dans la console d'administration, définissez une propriété personnalisée pour le module d'extension. La propriété personnalisée a pour valeur le nom du module d'extension.

Partie 2. Déploiement des applications

Chapitre 3. 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.

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 5 illustre les composants et les bibliothèques d'une application.

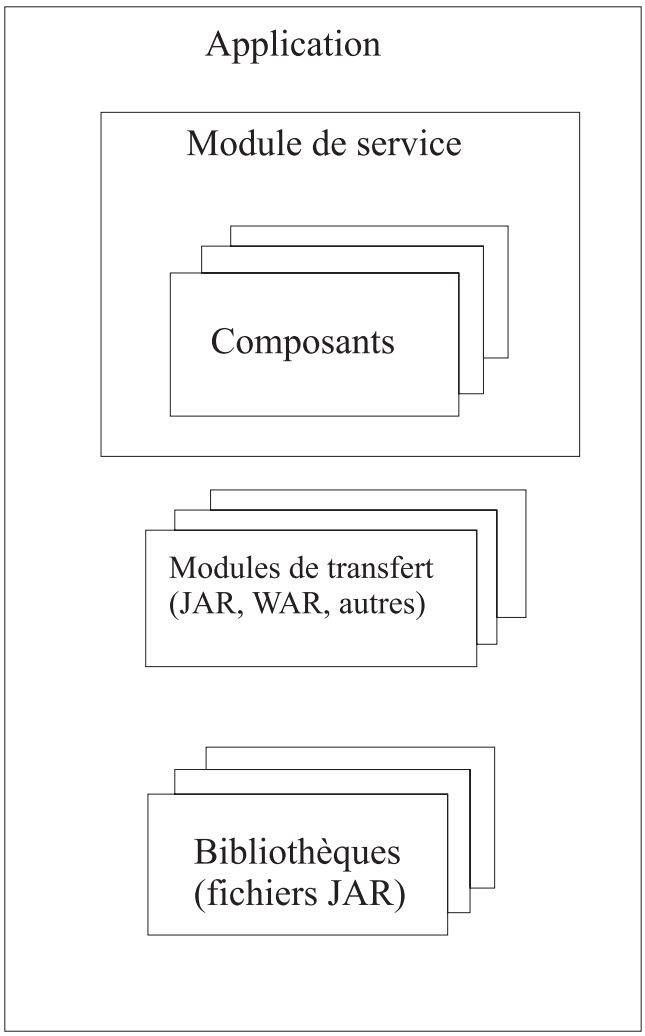


Figure 5. Relations entre module, composants et bibliothèques

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 :

- *myService*App
- *myService*EJB
- *myService*EJBClient
- *myService*Web

Remarque : La commande serviceDeploy crée le module de transfert *myService*Web 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.

Référence associée

Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

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.

Référence associée


Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

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.

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-nom*et 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 196.

   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 un rapport classpath 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
```

Installez le module sur les systèmes de production comme décrit à la rubrique Installation d'un module sur un serveur de productionInstallation d'un module sur un serveur de production.

Référence associée


Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

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'applications 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.

Chapitre 4. Installation d'un module sur un serveur de production

Cette rubrique décrit les étapes liées à l'utilisation d'une application sur un serveur test et à son déploiement dans un environnement de production.

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 à la section *Préparation du déploiement sur un serveur* du document PDF Développement et déploiement de modules transférez les fichiers sur le système de production en vue du déploiement. Pour plus d'informations, consultez les centres de documentation de WebSphere Integration Developer et de 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 commandeserviceDeploy 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 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, consultez le centre de documentation de 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.

Référence associée

Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

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 sur 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 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.

Référence associée


Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

Déploiement d'applications via des tâches Apache Ant

Cette rubrique décrit comment utiliser des tâches Apache Ant pour automatiser le déploiement des applications sur WebSphere Process Server. En utilisant des tâches Apache Ant, vous pouvez définir le déploiement de plusieurs applications et lancer l'exécution 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 connaissez les tâches Apache Ant.
- 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 partie sur les références du centre de documentation de WebSphere Application Server Network Deployment contient une section sur les API. Les tâches Apache Ant 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 plusieurs applications en même temps, développez une tâche Apache Ant avant le déploiement. La tâche Apache Ant peut ensuite déployer et installer les applications sur les serveurs sans intervention de votre part.

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 Apache Ant pour exécuter la commande ServiceDeploy afin de créer le fichier EAR pour chaque serveur.
5. Créez une tâche Apache Ant pour exécuter la commande InstallApplication pour chaque fichier EAR à partir de l'étape 4 sur les serveurs concernés.
6. Exécutez la tâche ServiceDeploy Apache Ant afin de créer le fichier EAR pour les applications.
7. Exécutez la tâche InstallApplication Apache Ant pour 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, une tâche Apache Ant est incluse au 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 indique comment appeler la tâche Apache Ant.

```
${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.

Référence associée

Commande serviceDeploy

Utilisez la commande serviceDeploy pour regrouper les modules compatibles avec l'architecture SCA, tels que les applications Java, qui peuvent être installés sur un serveur. Cette commande est utile pour effectuer des installations par lots avec wsadmin.

Chapitre 5. Installation des applications de tâches utilisateur et de processus métier

Vous pouvez distribuer des modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, à des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Avant de commencer

Vérifiez que Business Flow Manager ou Human Task Manager ou les deux sont installés et configurés pour chaque serveur d'applications sur lequel vous voulez installer votre application.

A propos de cette tâche

Vous pouvez installer des applications de tâche et de processus métier à partir de la console d'administration, à partir de la ligne de commande ou en exécutant un script d'administration, par exemple.

Résultats

Après l'installation d'une application de processus métier ou de tâches utilisateur, tous les modèles de processus métier et de tâche utilisateur sont mis à l'état démarrer. Vous pouvez créer des instances de processus et de tâches à partir de ces modèles.

Que faire ensuite

Avant de pouvoir créer des instances de processus ou de tâches, vous devez lancer l'application.

Concepts associés

«Déploiement de processus métier et de tâches utilisateur», à la page 205
Lorsque WebSphere Integration Developer ou le déploiement de services génère le code de déploiement de votre processus ou de votre tâche, chaque composant de processus ou de tâche est mappé vers un bean enterprise 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 de ce processus est générée et intégrée au 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.

«Installation d'applications de processus métier et de tâches utilisateur dans un environnement de déploiement réseau», à la page 204
Lors de l'installation de modèles de processus ou de modèles de tâches utilisateur dans un environnement de déploiement réseau, les actions suivantes s'exécutent automatiquement :

Installation d'applications de processus métier et de tâches utilisateur dans un environnement de déploiement réseau

Lors de l'installation de modèles de processus ou de modèles de tâches utilisateur dans un environnement de déploiement réseau, les actions suivantes s'exécutent automatiquement :

L'application est installée de manière asynchrone par étapes. Chaque étape doit se terminer sans erreur pour que la suivante puisse commencer.

1. L'installation de l'application démarre sur le gestionnaire de déploiement.

   Au cours de cette étape, les modèles de processus métier et de tâches utilisateur sont configurés dans le référentiel de configuration de WebSphere. L'application est également validée. Si des erreurs se produisent, elles sont consignées dans le fichier System.out, dans le fichier System.err ou sous forme d'entrées FFDC sur le gestionnaire de déploiement.
2. L'installation de l'application se poursuit sur l'agent de noeud.

   Au cours de cette étape, l'installation se déclenche sur un instance du serveur d'applications. Cette instance est la cible de déploiement ou constitue une partie de cette cible. Si la cible de déploiement consiste en un cluster doté de plusieurs membres de cluster, l'instance du serveur est choisie de façon arbitraire parmi les membres du cluster. Si des erreurs se produisent au cours de cette étape, elles sont consignées dans le fichier SystemOut.log, dans le fichier SystemErr.log ou sous forme d'entrées FFDC sur l'agent de noeud.
3. L'application s'exécute sur l'instance du serveur.

   Au cours de cette étape, les modèles de processus et de tâches sont déployés vers la base de données de Business Process Choreographer sur la cible de déploiement. Si des erreurs se produisent, elles sont consignées dans le fichier System.out, dans le fichier SystemErr.log ou sous forme d'entrées FFDC sur cette instance de serveur.

Tâches associées

Chapitre 5, «Installation des applications de tâches utilisateur et de processus métier», à la page 203
Vous pouvez distribuer des modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, à des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Déploiement de processus métier et de tâches utilisateur

Lorsque WebSphere Integration Developer ou le déploiement de services génère le code de déploiement de votre processus ou de votre tâche, chaque composant de processus ou de tâche est mappé vers un bean enterprise 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 de ce processus est générée et intégrée au 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 processus ou ces tâches sont stockés en tant que modèles de processus métier ou modèles de tâches utilisateur, selon le cas, dans la base de données de 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 un application d'entreprise différente. Lors de l'installation d'une nouvelle application d'entreprise, la version du modèle installée est déterminée comme suit :

- Si le nom du modèle et l'espace de nom cible n'existent pas déjà, un nouveau modèle est créé.
- Si le nom du modèle et l'espace de nom cible sont ceux d'un 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 est la date à laquelle la tâche utilisateur ou le processus métier a été modélisé.
- Si vous utilisez le déploiement de services, la date de début de validité est la date à laquelle la commande serviceDeploy a été exécutée. La date d'installation de l'application est uniquement prise en compte comme date de début de validité pour les tâches de collaboration.

Tâches associées

Chapitre 5, «Installation des applications de tâches utilisateur et de processus métier», à la page 203
Vous pouvez distribuer des modules SCA (Service Component Architecture) contenant des processus métier ou des tâches utilisateur, ou les deux, à des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

Installation interactive d'applications de processus métier et de tâches utilisateur

Vous pouvez installer une application de manière interactive 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 de manière interactive.

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 du référentiel de configuration maître. Si un processus de scriptage se termine et que vous n'avez sauvegardé vos modifications, celles-ci sont supprimées.

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.
```

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 utilisateur, les conditions suivantes doivent être préalablement remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être en cours de fonctionnement 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 un membre du cluster au moins doivent être en cours de fonctionnement. Le membre du 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 serveurdoivent être en cours de fonctionnement. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Tous les modèles de processus métier et de tâche utilisateur appartenant à l'application doivent être à l'état arrêté.
- Il n'existe aucune instance de modèle de processus métier ou de tâche utilisateur dans quelque état que ce soit.
- Dans les environnements de serveur autonome utilisés comme environnements de test d'unité et de développement, le serveur peut être configuré pour s'exécuter en mode développement. Il n'est pas nécessaire pour cela que les modèles soient arrêtés et qu'aucune instance n'existe. Cette configuration ne peut toutefois pas s'appliquer aux environnements de production.

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. Arrêtez tous les modèles de tâche et de processus de l'application.

   Cette opération empêche la création d'instances de tâche ou de processus.

   a. Cliquez sur **Applications** → **Modules SCA** dans le panneau de navigation de la console d'administration.

   b. Sélectionnez le module contenant les modules à arrêter.

   c. Dans la section Propriétés supplémentaires, cliquez sur **Processus métier** ou **Tâches utilisateur**, ou sur les deux, selon les besoins.

   d. Sélectionnez tous les modèles de tâche ou de processus en cochant la case appropriée.

   e. Cliquez sur **Arrêter**.

   Répétez cette étape pour tous les modules EJB contenant des modèles de processus métier ou de tâches utilisateur.
2. Assurez-vous que la base de données, qu'au moins un serveur d'applications pour chaque cluster et que le serveur autonome sur lequel l'application est déployée sont en cours d'exécution.

Dans un environnement de déploiement réseau, le gestionnaire de déploiement, tous les serveurs d'applications autonomes gérés et au moins un serveur d'applications doivent fonctionner pour chaque cluster sur lequel est installée l'application.

3. Vérifiez qu'aucune instance de processus métier ou de tâche utilisateur n'existe pour l'application.

   Si nécessaire, un administrateur peut utiliser Business Process Choreographer Explorer pour supprimer des instances de processus ou de processus.

4. Arrêtez et désinstallez l'application :
   a. Cliquez sur **Applications → Applications d'entreprise** dans la sous-fenêtre de navigation de la console d'administration.
   b. Sélectionnez l'application que vous voulez désinstaller et cliquez sur **Arrêter**.

      Cette étape échouera si des instances de processus ou de tâche existent toujours dans l'application.
   c. Sélectionnez de nouveau l'application que vous voulez désinstaller et cliquez sur **Désinstaller**.
   d. Cliquez sur **Sauvegarder** pour enregistrer les modifications.

Résultats

L'application est désinstallée.

Désinstallation d'applications de processus métier et de tâches utilisateur à l'aide de commandes d'administration

Les commandes d'administration offrent une solution alternative à 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 utilisateur, les conditions suivantes doivent être préalablement remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être en cours de fonctionnement 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 un membre du cluster au moins doivent être en cours de fonctionnement. Le membre du 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 serveurdoivent être en cours de fonctionnement. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Tous les modèles de processus métier et de tâche utilisateur appartenant à l'application doivent être à l'état arrêté.
- Il n'existe aucune instance de modèle de processus métier ou de tâche utilisateur dans quelque état que ce soit.

- Dans les environnements de serveur autonome utilisés comme environnements de test d'unité et de développement, le serveur peut être configuré pour s'exécuter en mode développement. Il n'est pas nécessaire pour cela que les modèles soient arrêtés et qu'aucune instance n'existe. Cette configuration ne peut toutefois pas s'appliquer aux environnements de production.

Si la sécurité globale est activée, vérifiez que votre ID utilisateur dispose des droits d'opérateur.

Assurez-vous que le processus serveur auquel le client d'administration est connecté 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 décrivent les instructions d'utilisation du script bpcTemplates.jacl pour désinstaller des applications contenant des modèles de processus métier ou de tâches utilisateur. Vous devez arrêter un modèle de métier avant de pouvoir désinstaller l'application auquel il appartient. Vous pouvez exécuter le script bpcTemplates.jacl pour arrêter et désinstaller les modèles en une seule étape.

Avant de désinstaller des applications, vous pouvez supprimer les instances de processus ou de tâches associées aux modèles contenus dans les applications, à l'aide par exemple de Business Process Choreographer Explorer. Vous pouvez également utiliser l'option **-force** avec le script bpcTemplates.jacl pour supprimer toutes les instances associées aux modèles, pour arrêter les modèles et les désinstaller en une seule étape.

ATTENTION :
Comme l'option -force supprime toutes les données des instances de processus et des instances de tâche, nous vous recommandons de l'utiliser avec précaution.

Procédure

1. Accédez au répertoire d'exemples de Business Process Choreographer.
   Sous Windows, entrez :
   ```
   cd racine_installation\ProcessChoreographer\admin
   ```
   Sous Linux, UNIX et i5/OS, entrez :
   ```
   cd racine_installation/ProcessChoreographer/admin
   ```
2. 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 globale est activée, indiquez l'ID utilisateur aux fins d'authentification.

mot_de_passe_utilisateur
Si la sécurité globale est activée, indiquez le mot de passe utilisateur aux fins d'authentification.

nom_application
Si la sécurité globale est activée, indiquez le mot de passe utilisateur aux fins d'authentification.

Résultats

L'application est désinstallée.

Chapitre 6. Installation des adaptateurs

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 7. Installation d'applications EIS

Un module d'application EIS peut être déployé sur une plateforme J2EE. Le déploiement génère une application encapsulée dans un fichier EAR déployé vers le serveur. Tous les artefacts et ressources sont créés, l'application est configurée et prête pour l'exécution.

A propos de cette tâche

Le déploiement vers la plateforme J2EE crée les artefacts et ressources J2EE suivants :

Tableau 42. Mappages de liaisons avec des artefacts J2EE

Liaison dans le module SCA	Artefacts J2EE générés	Ressources J2EE créées
Importation EIS	Références de ressources générées sur le bean EJB du module session.	ConnectionFactory
Exportation EIS	Bean MDB généré ou déployé selon l'interface d'écoute prise en charge par l'adaptateur de ressources.	ActivationSpec
Importation JMS	Le bean MDB fourni par l'exécutable est déployé, les références de ressources sont générées sur le bean EJB Session du module. Notez que le bean MDB n'est créé que si l'importation possède une destination de réception.	• ConnectionFactory • ActivationSpec • Destinations
Exportation JMS	Le bean MDB fourni par l'exécutable est déployé, les références de ressources sont générées sur le bean EJB Session du module.	• ActivationSpec • ConnectionFactory • Destinations

Quand l'importation ou l'exportation définit une ressource telle qu'une ConnectionFactory, la référence de ressource est générée dans le descripteur de déploiement de l'EJB de session sans état du module. De plus, la liaison appropriée est générée dans le fichier de liaison EJB. Le nom auquel la référence de ressource est liée est soit la valeur de l'attribut cible, le échéant, soit un nom de recherche JNDI par défaut attribué à la ressource, basé sur le nom de module et le nom d'importation.

Lors du déploiement, l'implémentation localise le bean session de module et l'utilise pour interroger les ressources.

Lors du déploiement de l'application vers le serveur, la tâche d'installation EIS vérifie l'existence de la ressource d'élément liée. Si elle n'existe pas et que le fichier SCDL indique au moins une propriété, la ressource sera créée et configurée par la tâche d'installation EIS. Si la ressource n'existe pas, aucune action n'est réalisée et on suppose que la ressource sera créée avant l'exécution de l'application.

Quand l'importation JMS est déployée avec une destination de réception, un bean MDB (Message Driven Bean) est déployé. Il écoute les réponses aux requêtes envoyées. Le MDB est associé à la destination envoyée avec la requête dans la zone JMSreplyTo du message JMS. Lors de l'arrivée du message de réponse, le bean MDB utilise son ID corrélation pour récupérer les informations de rappel stockées dans la destination de rappel, puis appelle l'objet de rappel.

La tâche d'installation crée ConnectionFactory et trois destinations à partir des informations du fichier d'importation. En outre, elle crée ActivationSpec pour activer le bean MDB d'exécution et écouter les réponses sur la destination de réception. Les propriétés de ActivationSpec sont dérivées des propriétés Destination/ConnectionFactory. Si le fournisseur JMS est un adaptateur de ressources SIBus, les destinations SIBus correspondant à la destination JMS sont créées.

Quand l'exportation JMS est déployée, un message MDB (autre que le MDB déployé pour l'importation JMS) est déployé. Il écoute les requêtes entrantes sur la destination de réception et distribue les requêtes que doit traiter le SCA. La tâche d'installation crée l'ensemble de ressources similaire à celui destiné pour l'importation JMS, un objet ActivationSpec, une fabrique de connexions ConnectionFactory pour l'envoi d'une réponse et deux destinations. Toutes les propriétés de ces ressources sont indiquées dans le fichier d'exportation. Si le fournisseur JMS est un adaptateur de ressources SIBus, les destinations SIBus correspondant à la destination JMS sont créées.

Déploiement d'un module d'application EIS vers la plateforme J2SE

Le module EIS peut être déployé sur une plateforme J2SE mais seule l'importation EIS sera prise en charge.

Avant de commencer

Avant de commencer cette tâche, vous devez créer un module d'application EIS avec une liaison d'importation JMS dans l'environnement WebSphere Integration Development.

A propos de cette tâche

Un module d'application EIS sera fourni avec une liaison d'importation lorsque vous souhaiterez accéder de façon asynchrone à des systèmes EIS par l'intermédiaire de files d'attente de messages.

Le déploiement vers la plateforme J2SE est la seule instance où l'implémentation de liaison peut être exécutée dans un noeud non géré. La liaison JMS requiert un support asynchrone et un support JNDI, qui ne sont fournis ni l'un ni l'autre par l'architecture des composants de service de base ou par J2SE. L'architecture JCA (J2EE Connector Architecture) ne prend pas en charge les communications entrantes non gérées, ce qui élimine les exportations EIS.

Quand le module d'application EIS et l'importation EIS sont déployés vers J2SE en plus des dépendances de module, l'adaptateur WebSphere Adapter employé par l'importation doit être indiqué comme dépendance dans le manifeste ou sous toute autre forme prise en charge par SCA.

Déploiement d'un module d'application EIS vers la plateforme J2EE

Le déploiement d'un module EIS vers la plateforme J2EE génère une application encapsulée dans un fichier EAR déployé vers le serveur. Tous les artefacts et ressources sont créés, l'application est configurée et prête pour l'exécution.

Avant de commencer

Avant de commencer cette tâche, vous devez créer un module EIS avec une liaison d'importation JMS dans l'environnement WebSphere Integration Development.

A propos de cette tâche

Le déploiement vers la plateforme J2EE crée les artefacts et ressources J2EE suivants :

Tableau 43. Mappages de liaisons avec des artefacts J2EE

Liaison dans le module SCA	Artefacts J2EE générés	Ressources J2EE créées
Importation EIS	Références de ressources générées sur le bean EJB du module session.	ConnectionFactory
Exportation EIS	Bean MDB généré ou déployé selon l'interface d'écoute prise en charge par l'adaptateur de ressources.	ActivationSpec
Importation JMS	Le bean MDB fourni par l'exécutable est déployé, les références de ressources sont générées sur le bean EJB Session du module. Notez que le bean MDB n'est créé que si l'importation possède une destination de réception.	• ConnectionFactory • ActivationSpec • Destinations
Exportation JMS	Le bean MDB fourni par l'exécutable est déployé, les références de ressources sont générées sur le bean EJB Session du module.	• ActivationSpec • ConnectionFactory • Destinations

Quand l'importation ou l'exportation définit une ressource telle qu'une ConnectionFactory, la référence de ressource est générée dans le descripteur de déploiement de l'EJB de session sans état du module. De plus, la liaison appropriée est générée dans le fichier de liaison EJB. Le nom auquel la référence de ressource est liée est soit la valeur de l'attribut cible, le échéant, soit un nom de recherche JNDI par défaut attribué à la ressource, basé sur le nom de module et le nom d'importation.

Lors du déploiement, l'implémentation localise le bean session de module et l'utilise pour interroger les ressources.

Lors du déploiement de l'application vers le serveur, la tâche d'installation EIS vérifie l'existence de la ressource d'élément liée. Si elle n'existe pas et que le fichier SCDL indique au moins une propriété, la ressource sera créée et configurée par la tâche d'installation EIS. Si la ressource n'existe pas, aucune action n'est réalisée et on suppose que la ressource sera créée avant l'exécution de l'application.

Quand l'importation JMS est déployée avec une destination de réception, un bean MDB (Message Driven Bean) est déployé. Il écoute les réponses aux requêtes envoyées. Le MDB est associé à la destination envoyée avec la requête dans la zone JMSreplyTo du message JMS. Lors de l'arrivée du message de réponse, le bean MDB utilise son ID corrélation pour récupérer les informations de rappel stockées dans la destination de rappel, puis appelle l'objet de rappel.

La tâche d'installation crée ConnectionFactory et trois destinations à partir des informations du fichier d'importation. En outre, elle crée ActivationSpec pour activer le bean MDB d'exécution et écouter les réponses sur la destination de réception. Les propriétés de ActivationSpec sont dérivées des propriétés Destination/ConnectionFactory. Si le fournisseur JMS est un adaptateur de ressources SIBus, les destinations SIBus correspondant à la destination JMS sont créées.

Quand l'exportation JMS est déployée, un message MDB (autre que le MDB déployé pour l'importation JMS) est déployé. Il écoute les requêtes entrantes sur la destination de réception et distribue les requêtes que doit traiter le SCA. La tâche d'installation crée l'ensemble de ressources similaire à celui destiné pour l'importation JMS, un objet ActivationSpec, une fabrique de connexions ConnectionFactory pour l'envoi d'une réponse et deux destinations. Toutes les propriétés de ces ressources sont indiquées dans le fichier d'exportation. Si le fournisseur JMS est un adaptateur de ressources SIBus, les destinations SIBus correspondant à la destination JMS sont créées.

Chapitre 8. 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 indiquant 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 et les spécifications d'activation J2C (voir Suppression de destinations SiBus et Suppression de spécification d'activation J2C dans la section Administration).

2. Si l'application est installée correctement, vérifiez qu'elle a été lancée.

   Si l'application n'a pas été lancée, 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 indiquant comment procéder.
   b. Déterminez si les ressources requises par l'application sont disponibles et/ou ont été lancées.

      Les ressources non lancées empêchent l'exécution d'une application. Cela permet d'éviter 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 composant nécessaires au module se trouvent dans les bons dossiers du système test sur lequel vous avez généré le fichier JAR (archive Java). La section «Préparation au déploiement sur un serveur» contient des informations complémentaires.

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.

   a. Déterminez si l'application utilisent des services contenus dans une autre application. Vérifiez que l'autre application est installée et a été lancée.
   b. Déterminez si les liaisons d'importation et d'exportation de toutes les unités contenues 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.

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.

Suppression des destinations SIBus

Les destinations SIBus sont des connexions qui permettent aux applications d'accéder aux services. Dans certains cas, vous devrez supprimer ces destinations.

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.
   Pour accéder à ce panneau, cliquez sur **Intégration de services > Bus**
3. Sélectionnez les destinations du bus système SCA.
   Dans l'écran, cliquez sur **SCA.SYSTEM.*nom_cellule*.Bus**, où *nom_cellule* correspond au nom de la cellule contenant le module avec les destinations que vous êtes en train de supprimer.
4. Supprimez les destinations qui contiennent le nom du module que vous êtes en train de supprimer.
   a. Cochez la case à cocher située en regard des destinations concernées.
   b. 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.

Partie 3. Annexes

Remarques

Ces informations concernent initialement des produits et services fournis aux Etats-Unis.

Il se peut qu'IBM ne propose pas les produits, services ou fonctions décrits dans ce document dans d'autres pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service 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.

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

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
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

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.

Toute 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 confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non 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 ou aux interfaces de programmation IBM. 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 code 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.

Documentation sur 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, noms de produits et logos

IBM, le logo IBM, developerWorks, WebSphere et z/OS sont des marques d'International Business Machines Corporation aux Etats-Unis et/ou dans certains autres pays.

Adobe est une marque d'Adobe Systems Incorporated aux Etats-Unis et/ou dans certains autres pays.

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques 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 6.1.0