WebSphere® Process Server for Multiplatforms

Versión 6.1.0

Desarrollo y despliegue de módulos

WebSphere® Process Server for Multiplatforms

Versión 6.1.0

Desarrollo y despliegue de módulos

Nota

Antes de utilizar esta información, asegúrese de leer la información general de la sección Avisos al final de este documento.

1 de febrero de 2008

Esta edición se aplica a la versión 6, release 1, modificación 0 de WebSphere Process Server for Multiplatforms (número de producto 5724-L01) y a todos los releases y las modificaciones subsiguientes hasta que se indique lo contrario en nuevas ediciones.

Para enviar comentarios sobre este documento, envíe un mensaje de correo electrónico a doc-comments@us.ibm.com. Esperamos sus comentarios.

Contenido

Figuras

Tablas

Parte 1. Desarrollo de aplicaciones

Capítulo 1. Visión general del desarrollo de módulos

Un módulo es una unidad de despliegue básica para una aplicación de WebSphere Process Server. Un módulo contiene una o más bibliotecas de componentes y módulos intermedios utilizados por la aplicación. Un componente puede hacer referencia a otros componentes de servicio. El desarrollo de módulos implica asegurarse de que los componentes, módulos intermedios y bibliotecas (colecciones de artefactos a los que el módulo hace referencia) que la aplicación necesita están disponibles en el servidor de producción.

WebSphere Integration Developer es la herramienta principal para desarrollar módulos para su despliegue en WebSphere Process Server. Aunque puede desarrollar módulos en otros entornos, es mejor utilizar WebSphere Integration Developer.

WebSphere Process Server da soporte a dos tipos de módulos de servicio; módulos para servicios de empresa y módulos de mediación. Un módulo para servicios de empresa implementa la lógica de un proceso. Un módulo de mediación permite la comunicación entre aplicaciones transformando la invocación de servicio a un formato que sólo entiende el destino, pasando la petición al destino y devolviendo el resultado al originador.

Las secciones siguientes tratan de cómo implementar y actualizar módulos en WebSphere Process Server.

Sinopsis sobre componentes

Un componente es el bloque básico para encapsular una lógica de empresa reutilizable. Un componente de servicio se asocia con interfaces, referencias e implementaciones. La interfaz define un contrato entre un componente de servicio y un componente llamante. Con WebSphere Process Server, un módulo de servicio puede exportar un componente de servicio para su uso por otros módulos o importar un componente de servicio para su uso. Para invocar un componente de servicio, el módulo llamante hace referencia a la interfaz para el componente de servicio. Las referencias a las interfaces se resuelven configurando las referencias del módulo llamante a sus interfaces respectivas.

Para desarrollar un módulo, debe realizar las actividades siguientes:

1. Defina interfaces para los componentes del módulo
2. Defina, modifique o maneje objetos de empresa utilizados por los componentes de servicio
3. Defina o modifique componentes de servicio a través de sus interfaces.

   Nota: Un componente de servicio se define mediante su interfaz.
4. Opcionalmente, exporte o importe componentes de servicio.
5. Cree un archivo EAR que utilizará para instalar un módulo que utiliza componentes. Cree el archivo utilizando la característica EAR de exportación de WebSphere Integration Developer o el mandato serviceDeploy para crear un archivo EAR a fin de instalar un módulo de servicio que utilice componentes de servicio.

Tipos de desarrollo

WebSphere Process Server proporciona un modelo de programación de componentes para facilitar un paradigma de programación orientada a servicios. Para utilizar este modelo, un proveedor exporta interfaces de un componente de servicio para que un consumidor pueda importar esas interfaces y utilizar el componente de servicio como si fuese local. Un desarrollador utiliza interfaces de tipo fuerte o de tipo dinámico para implementar o invocar el componente de servicio. Las interfaces y sus métodos se describen en la sección Referencias de este centro de información.

Después de instalar los módulos de servicio en los servidores, puede utilizar la consola administrativa para cambiar el componente de destino para una referencia de una aplicación. El nuevo destino debe aceptar el mismo tipo de objeto de empresa y realizar la misma operación que solicita la referencia de la aplicación.

Consideraciones sobre el desarrollo de componentes de servicio

Al desarrollar un componente de servicio, formúlese las preguntas siguientes:

- ¿Este componente de servicio se exportará y lo utilizará otro módulo?

  En caso afirmativo, asegúrese de que otro módulo pueda utilizar la interfaz que defina para el componente.
- ¿El componente de servicio tardará relativamente mucho tiempo en ejecutarse?

  Si la respuesta es afirmativa, estudie implementar una interfaz asíncrona al componente de servicio.
- ¿Es ventajoso descentralizar el componente de servicio?

  Si la respuesta es afirmativa, estudie tener una copia del componente de servicio en un módulo desplegado en un clúster de servidores para beneficiarse del proceso paralelo.
- ¿La aplicación requiere una mezcla de recursos de compromiso de 1 fase y de 2 fases?

  En caso afirmativo, asegúrese de habilitar el soporte de último participante para la aplicación.

  Nota: Si crea la aplicación mediante WebSphere Integration Developer o crea el archivo EAR instalable mediante el mandato serviceDeploy, estas herramientas habilitan automáticamente el soporte de la aplicación. Consulte el tema "Utilización de recursos de compromiso de una fase y de dos fases en la misma transacción" en el centro de información de WebSphere Application Server Network Deployment.

Desarrollo de módulos de servicio

Un componente de servicio debe estar contenido en un módulo de servicio. Desarrollar módulos de servicio para contener componentes de servicio es un elemento clave al proporcionar servicios a otros módulos.

Antes de empezar

Esta tarea da por supuesto que un análisis de los requisitos muestra que implementar un componente de servicio para su uso por otros módulos es beneficioso.

Acerca de esta tarea

Después de analizar los requisitos, tal vez decida que proporcionar y utilizar componentes de servicio es una manera eficaz de procesar información. Si determina que los componentes de servicio reutilizables se benefician del entorno, cree un módulo de servicio para contener los componentes de servicio.

Procedimiento

1. Identifique componentes de servicio que otros módulos pueden utilizar.

   Una vez que haya identificado los componentes de servicio, continúe en Desarrollo de componentes de servicio.
2. Identifique componentes de servicio en una aplicación que puedan utilizar componentes de servicio en otros módulos de servicio.

   Una vez que haya identificado los componentes de servicio y sus componentes de destino, continúe en Invocación de componentes.
3. Conecte los componentes de cliente con los componentes de destino mediante cables.

Desarrollo de componentes de servicio

Desarrolle componentes de servicio para proporcionar lógica reutilizable a varias aplicaciones en el servidor.

Antes de empezar

En esta tarea se da por supuesto que ya se ha desarrollado e identificado el proceso que es útil para varios módulos.

Acerca de esta tarea

Varios módulos pueden utilizar un componente de servicio. Al exportar un componente de servicio, queda a disposición de otros módulos que hagan referencia al componente de servicio mediante una interfaz. En esta tarea se describe cómo construir el componente de servicio para que otros modelos puedan utilizarlo.

Nota: Un solo componente de servicio puede contener varias interfaces.

Procedimiento

1. Defina el objeto de datos para mover datos entre el llamante y el componente de servicio.

   El objeto de datos y su tipo forman parte de la interfaz entre los llamantes y el componente de servicio.
2. Defina una interfaz que los llamantes utilizarán para hacer referencia al componente de servicio.

   Esta definición de interfaz indica el componente de servicio y lista los métodos disponibles en el componente de servicio.
3. Desarrolle la clase que define la implementación.
   - Si el componente es de larga ejecución (o asíncrono), siga en el paso 4.
   - Si el componente no es de larga ejecución (o síncrono), siga en el paso 5 en la página 6.
4. Desarrolle una implementación asíncrona.

   Importante: Un componente asíncrono no puede tener una propiedad joinsTransaction establecida en el valor `true`.

a. Defina la interfaz que representa el componente de servicio síncrono.
b. Defina la implementación del componente de servicio.
c. Continúe en el paso 6.

5. Desarrolle una implementación síncrona.
   a. Defina la interfaz que representa el componente de servicio síncrono.
   b. Defina la implementación del componente de servicio.
6. Guarde las interfaces e implementaciones de componentes con una extensión .java.
7. Empaquete el módulo de servicio y los recursos necesarios en un archivo JAR.

   Consulte "Instalación de un módulo en un servidor de producción" en este centro de información para ver una descripción de los pasos 7 a 9.
8. Ejecute el mandato serviceDeploy para crear un archivo EAR instalable que contenga la aplicación.
9. Instale la aplicación en el nodo de servidor.
10. Opcional: Configure los cables entre los llamantes y el componente de servicio correspondiente, si se llama a un componente de servicio de otro módulo de servicio.

    La sección de "Administración" de este centro de información describe la configuración de los cables.

Ejemplos de desarrollo de componentes

Este ejemplo muestra un componente de servicio síncrono que implementa un único método, CustomerInfo. La primera sección define la interfaz para el componente de servicio que implementa un método denominado getCustomerInfo.

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

El siguiente bloque de código implementa el componente de servicio.

```
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;
 }
}
```

Este ejemplo desarrolla un componente de servicio asíncrono. La primera sección del código define la interfaz para el componente de servicio que implementa un método denominado getQuote.

```
public interface StockQuote {

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

La sección siguiente es la implementación de la clase asociada con StockQuote.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

Esta sección siguiente del código implementa la interfaz asíncrona, StockQuoteAsync.

```
public interface StockQuoteAsync {

 // respuesta diferida
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);

 // devolución de llamada
 public Ticket getQuoteAsync(String symbol, StockQuoteCallback callback);
}
```

Esta sección es la interfaz, StockQuoteCallback, que define el método onGetQuoteResponse.

```
public interface StockQuoteCallback {

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

Qué hacer a continuación

Invoque el servicio.

Invocación de componentes

Los componentes con módulos pueden utilizar componentes en cualquier nodo de un clúster de WebSphere Process Server.

Antes de empezar

Antes de invocar un componente, asegúrese de que el módulo que contiene el componente esté instalado en WebSphere Process Server.

Acerca de esta tarea

Los componentes pueden utilizar cualquier componente de servicio disponible en un clúster de WebSphere Process Server utilizando el nombre del componente y pasando el tipo de datos que espera el componente. La invocación de un componente en este entorno implica localizar y, a continuación, crear la referencia en el componente necesario.

Nota: Un componente de un módulo puede invocar un componente en el mismo módulo, conocido como una invocación dentro del módulo. Implemente llamadas externas (invocaciones intermódulos) exportando la interfaz del componente que se proporciona e importando la interfaz en el componente llamante.

Importante: Cuando se invoca un componente que reside en un servidor distinto de aquél donde se ejecuta el módulo llamante, debe realizar configuraciones adicionales en los servidores. Las configuraciones necesarias dependen de si se

llama al componente de forma asíncrona o síncrona. La manera de configurar los servidores de aplicaciones en este caso se describen en tareas relacionadas.

Procedimiento

1. Determine los componentes que necesita el módulo llamante.

   Tome nota del nombre de la interfaz dentro de un componente y el tipo de datos que la interfaz necesita.
2. Defina un objeto de datos.

   Aunque la entrada o el retorno puede ser una clase Java, lo óptimo es un objeto de datos de servicio.
3. Localice el componente.
   a. Utilice la clase ServiceManager para obtener las referencias disponibles en el módulo llamante.
   b. Utilice el método locateService() para encontrar el componente.

      Según el componente, la interfaz puede ser un tipo de puerto WSDL (Web Service Descriptor Language) o una interfaz Java.
4. Invoque el componente de manera síncrona o asíncrona.

   Puede invocar el componente mediante una interfaz Java o utilice el método invoke() para invocar el componente de manera dinámica.
5. Procese la devolución.

   El componente puede generar una excepción, de manera que el cliente tiene que poder procesar esa posibilidad.

Ejemplos de invocación de un componente

En el ejemplo siguiente se crea una clase ServiceManager.

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

El ejemplo siguiente utiliza la clase ServiceManager para obtener una lista de componentes de un archivo que contenga las referencias de componente.

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

El código siguiente localiza un componente que implementa la interfaz StockQuote Java.

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

El código siguiente localiza un componente que implementa una interfaz de tipo de puerto Java o WSDL. El módulo llamante utiliza la interfaz Service para interactuar con el componente.

Consejo: Si el componente implementa una interfaz Java, el componente puede invocarse utilizando la interfaz o el método invoke().

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

En el ejemplo siguiente se muestra MyValue, código que llama a otro componente.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
```

```
        Customer customer = null;
        float quote = 0;
        float value = 0;

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

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

         // invocar
      StockQuoteAsync sQuote =
     (StockQuoteAsync)serviceManager.locateService("stockQuote");
      Ticket ticket =  sQuote.getQuoteAsync(customer.getSymbol());
   // ... hacer otra cosa...
      quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

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

        // lanzar
        throw new MyValueException(customer.getErrorMsg());
    }
    // responder
        return value;
 }
}
```

Qué hacer a continuación

Configure los cables entre las referencias al módulo llamante y las interfaces de componente.

Invocación dinámica de un componente

Cuando un módulo invoca un componente que tiene una interfaz de tipo de puerto WSDL (Service Descriptor Language), el módulo debe invocar el componente de forma dinámica utilizando el método invoke().

Antes de empezar

En esta tarea se da por supuesto que un componente llamante invoca a un componente de forma dinámica.

Acerca de esta tarea

Con una interfaz de tipo de puerto WSDL, un componente llamante debe utilizar el método invoke() para invocar el componente. Un módulo llamante también puede invocar un componente que tiene una interfaz Java de esta manera.

Procedimiento

1. Determine el módulo que contiene el componente necesario.
2. Determine la matriz que el componente necesita.

   La matriz de entrada puede ser de uno de los tres tipos siguientes:
   - Tipos o matrices Java primitivos en mayúsculas de este tipo
   - Clases o matrices Java de las clases
   - SDO (Service Data Objects)
3. Defina una matriz para que contenga la respuesta del componente.

   La matriz de respuesta puede ser de los mismos tipos que la matriz de entrada.

4. Utilice el método invoke() para invocar el componente necesario y pasar el objeto de matriz al componente.
5. Procese el resultado.

Ejemplos de invocación dinámica de un componente

En el ejemplo siguiente, un módulo utiliza el método invoke() para llamar a un componente que utiliza tipos de datos Java primitivos en mayúsculas.

```
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);
```

En el ejemplo siguiente se utiliza el método invoke con una interfaz de tipo de puerto WSDL de destino.

```
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 encapsula todos los parámetros de methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

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

Visión general del aislamiento de módulos y destinos

Al desarrollar módulos, identificará servicios que varios módulos pueden utilizar. Al reforzar los servicios de este modo se minimiza el ciclo de desarrollo y los costes. Si dispone de un servicio que muchos módulos utilizan, aisle los módulos de invocación del destino, de manera que si se actualiza el destino, el cambio al nuevo servicio sea transparente al módulo llamante. En este tema se contrasta el modelo de invocación sencilla y el modelo de invocación aislada y se proporciona un ejemplo de cómo puede resultar de utilidad el aislamiento. Aunque se describe un ejemplo específico, no es el único modo de aislar módulos de destinos.

Modelo de invocación sencilla

Al desarrollar módulos, puede utilizar servicios que se encuentren en otros módulos. Para hacer esto, puede importar el servicio al módulo y luego invocar a ese servicio. El servicio importado se “conecta” con el servicio exportado por el otro módulo, ya sea en WebSphere Integration Developer o enlazando el servicio

en la consola administrativa. El modelo de invocación sencilla ilustra este modelo.

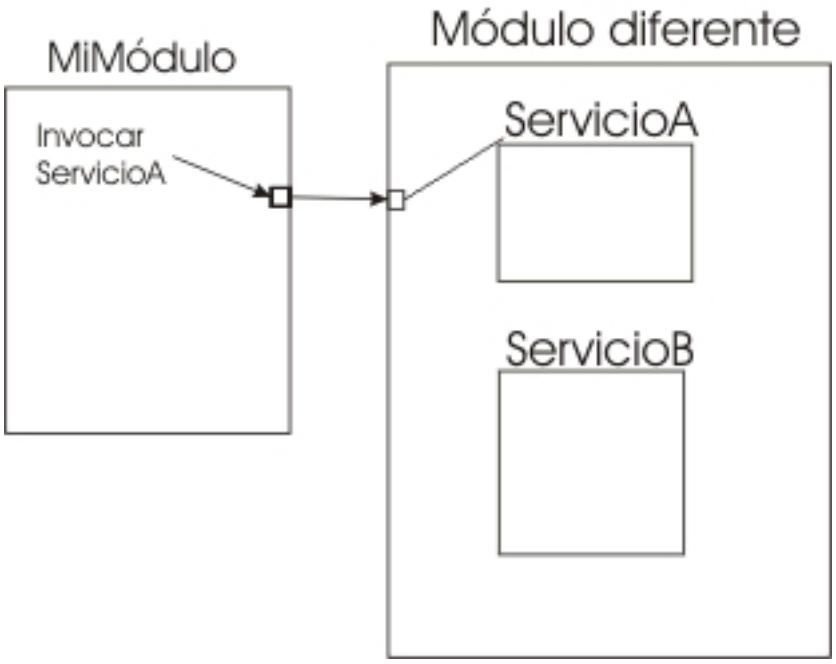


Figura 1. Modelo de invocación sencilla

Modelo de invocación aislada

Para cambiar el destino de una invocación sin detener la invocación de módulos, puede aislar los módulos de invocación del destino de la invocación. Esto permite que los módulos continúen procesando mientras cambia el destino porque no está cambiando el módulo en sí sino el destino en sentido descendente. En el Ejemplo de aislamiento de aplicaciones se muestra cómo el aislamiento permite cambiar el destino sin afectar al estado del módulo de invocación.

Ejemplo de aislamiento de aplicaciones

Utilizando el modelo de invocación sencillo, varios módulos que invocan al mismo servicio se parecerían bastante a varias aplicaciones que invocan a un solo servicio . MODA, MODB y MODC invocan a CalculateFinalCost.

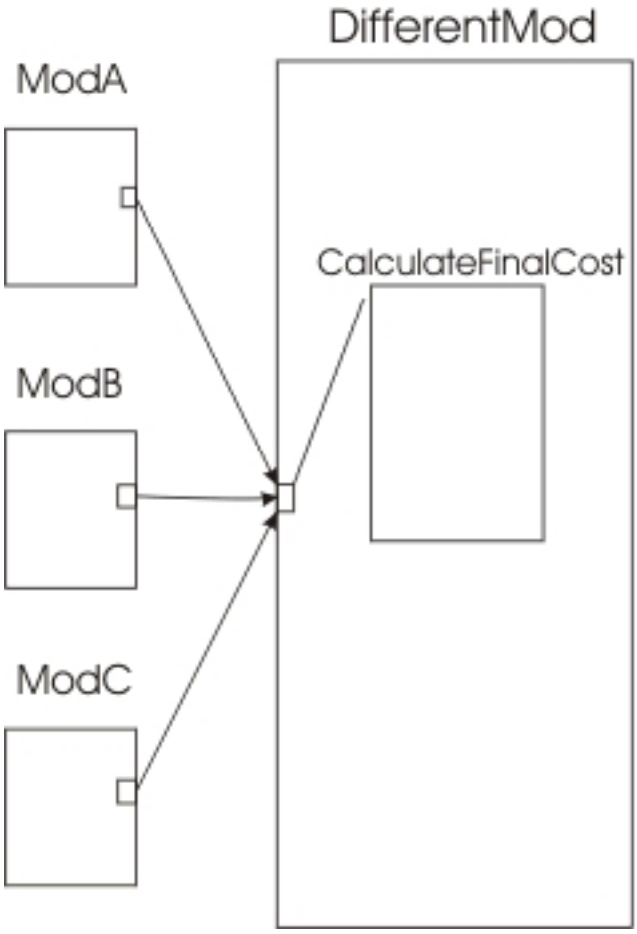


Figura 2. Varias aplicaciones que invocan a un solo servicio

El servicio proporcionado por CalculateFinalCost debe actualizarse de manera que los nuevos costes se reflejen en todos los módulos que utilizan el servicio. El equipo de desarrollo crea y prueba un nuevo servicio UpdatedCalculateFinal para incorporar los cambios. Está preparado para implantar el nuevo servicio en producción. Sin el aislamiento, tendría que actualizar todos los módulos que invocan a CalculateFinalCost para invocar a UpdateCalculateFinal. Con el aislamiento, sólo tiene que cambiar el enlace que conecta el módulo de almacenamiento intermedio con el destino.

Nota: Con el cambio del servicio de esta manera se permite continuar para proporcionar el servicio original a otros módulos que puedan necesitarlo.

Utilizando el aislamiento, puede crear un módulo de almacenamiento intermedio entre las aplicaciones y el destino (consulte Modelo de invocación aislada que invoca a UpdateCalculateFinal).

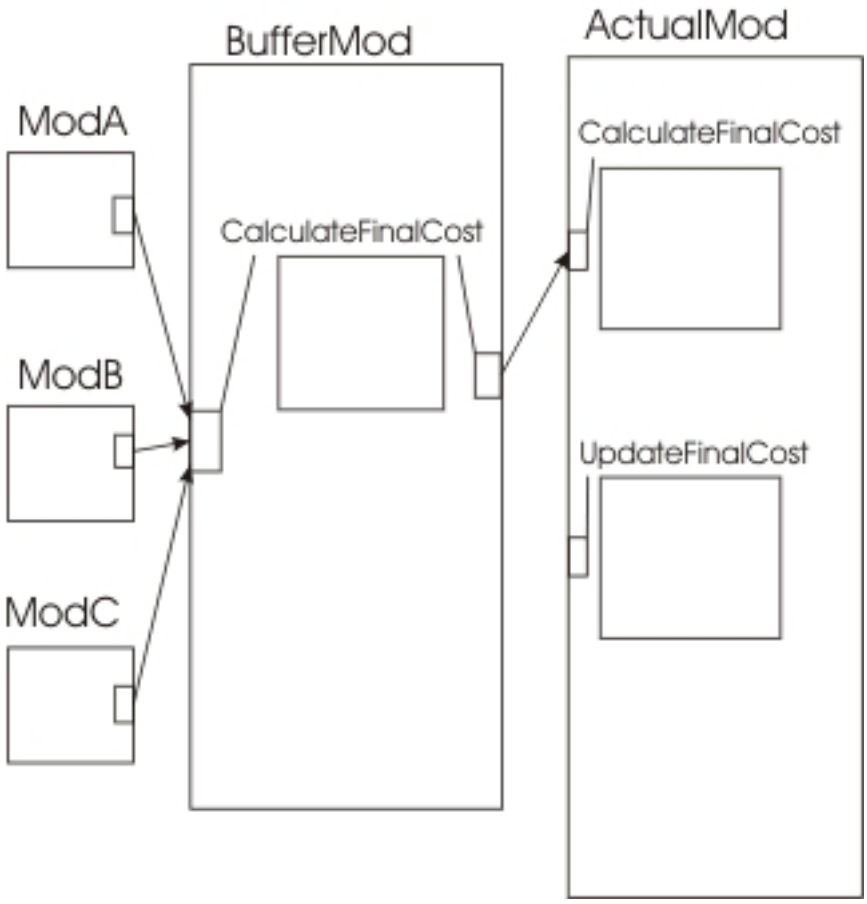


Figura 3. Modelo de invocación aislada que invoca a UpdateCalculateFinal

Con este modelo, los módulos de invocación no cambian, sólo tiene que cambiar el enlace de la importación del módulo de almacenamiento intermedio al destino (consulte Modelo de invocación aislada que invoca a UpdatedCalculateFinal).

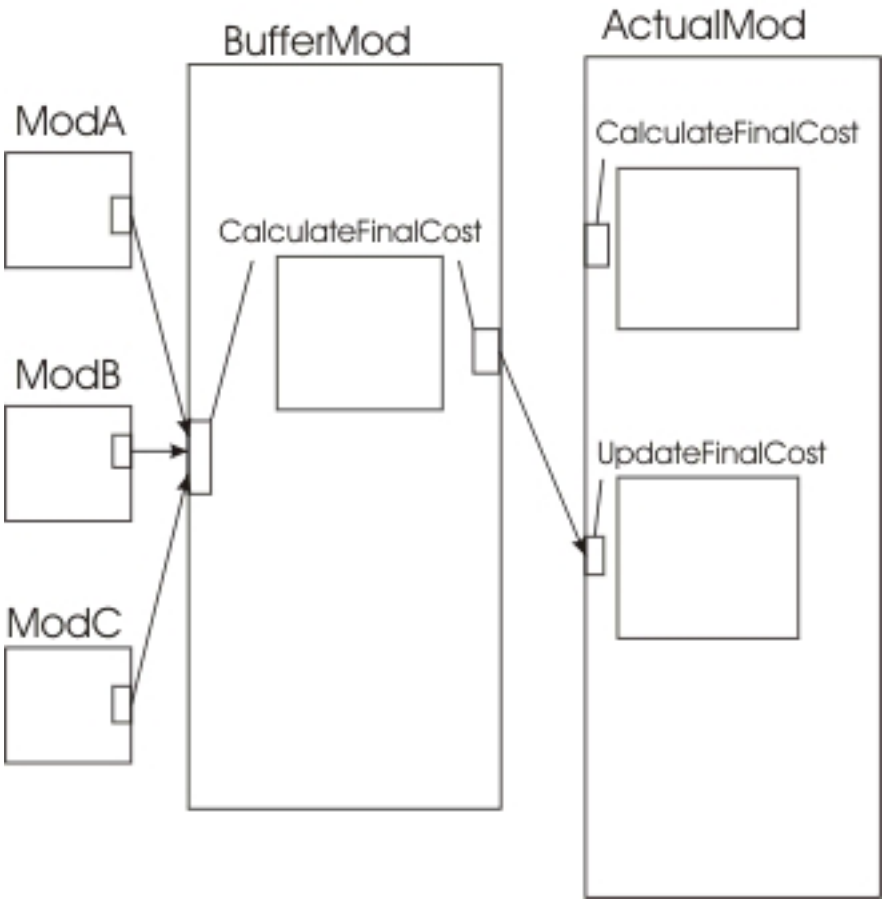


Figura 4. Modelo de invocación aislada que invoca a UpdatedCalculateFinal

Si el módulo de almacenamiento intermedio invoca al destino de forma síncrona, cuando reinicia el módulo de almacenamiento intermedio (ya sea un módulo de mediación o un servicio para un módulo de empresa) los resultados devueltos a la aplicación original proceden del nuevo destino. Si el módulo de almacenamiento intermedio invoca al destino de forma asíncrona, los resultados devueltos a la aplicación original proceden del nuevo destino en la siguiente invocación.

Tareas relacionadas

Cambio de destinos

El cambio del destino de una referencia proporciona a las aplicaciones la flexibilidad de aprovechar los avances en componentes cuando se producen sin volver a compilar e instalar la aplicación.

Enlaces HTTP

El enlace HTTP está diseñado para proporcionar una conectividad SCA (Service Component Architecture) con HTTP. Esto permite a las aplicaciones HTTP existentes o recién desarrolladas participar en entornos de arquitectura orientada a servicios (SOA - Service Oriented Architecture).

Además, una red de entornos de ejecución SCA se comunica en toda una infraestructura HTTP existente.

El enlace HTTP expone varias características HTTP:

- Los mensajes se presentan a los componentes de mediación de una manera que conserva el formato HTTP y la información de cabecera del mensaje. Esto proporciona una vista más familiar para los programadores de aplicaciones HTTP, usuarios y administradores.
- Una infraestructura de enlace de datos existente se amplía para las convenciones HTTP y proporciona una correlación entre mensajes SCA y cabeceras y cuerpos de mensajes HTTP.
- Se pueden configurar importaciones y exportaciones para admitir una gama de características HTTP comunes.
- Al instalar un módulo SCA que contiene importaciones o exportaciones HTTP, se configura automáticamente el entorno de ejecución de forma adecuada para permitir la conectividad con HTTP.

Puede encontrar instrucciones detalladas sobre la creación de importaciones y exportaciones HTTP en el centro de información de en **WebSphere Integration Developer** > **Desarrollo de aplicaciones de integración** > **Enlace de datos HTTP**.

Tareas relacionadas

Visualización de enlaces HTTP

Después de desplegar una aplicación, puede desear examinar los enlaces HTTP para asegurarse de que son correctos.

Modificación de los enlaces de exportación HTTP

La consola administrativa permite cambiar la configuración de enlaces de exportación HTTP sin cambiar el código fuente original y, a continuación, volver a desplegar la aplicación.

Modificación de los enlaces de importación HTTP

La consola administrativa permite cambiar la configuración de enlaces de importación HTTP sin cambiar el código fuente original y, a continuación, volver a desplegar la aplicación.

Alteración temporal de la implementación de la Service Component Architecture generada

Puede que a veces la conversión que el sistema crea entre un código Java y SDO (Service Data Object) no satisfaga sus necesidades. Utilice este procedimiento para sustituir la implementación por omisión de la clase SCA (Service Component Architecture) por la suya propia.

Antes de empezar

Asegúrese de que ha generado la conversión del tipo Java a WSDL (Web Services Definition Language) utilizando WebSphere Integration Developer o el mandato genMapper.

Acerca de esta tarea

Puede alterar temporalmente un componente generado que correlaciona un tipo Java con un tipo WSDL sustituyendo el código generado por código que satisfaga sus necesidades. Considere utilizar su propia correlación si ha definido sus propias clases Java. Utilice este procedimiento para hacer los cambios.

Procedimiento

1. Localice el componente generado. El componente se denomina *java_class*Mapper.component.
2. Edite el componente utilizando un editor de texto.
3. Convierta en comentario el código generado y proporcione su propio método.

   No cambie el nombre de archivo que contiene la implementación del componente.

A continuación figura un ejemplo de un componente generado que se va a sustituir:

```
private DataObject javatodata_setAccount_output(Object myAccount) {

 // Puede alterar temporalmente este código para la correlación personalizada.
 // Convierta en comentario este código y escriba código personalizado.

 // También puede cambiar el tipo Java que se pasa al
 // conversor, que este último intenta crear.


return SDOJavaObjectMediator.java2Data(myAccount);


}
```

Copie el componente y otros archivos al directorio en el que reside el módulo contenedor y conecte el componente de WebSphere Integration Developer o genere un archivo EAR (Enterprise Archive) con el mandato serviceDeploy.

Conceptos relacionados

"Normas de tiempo de ejecución utilizadas para la conversión de Java a Service Data Objects" en la página 18
Para alterar temporalmente el código generado o determinar posibles excepciones en tiempo de ejecución relacionadas con las conversiones de Java a Service Data Object (SDO), es importante tener un conocimiento de las normas correspondientes. La mayoría de las conversiones son directas, pero hay algunos casos complejos en que la ejecución proporciona la mejor posibilidad cuando convierte el código generado.

Referencia relacionada

Conversión de Java a XML

El sistema genera XML basado en tipos Java mediante la utilización de normas predefinidas.

Mandato genMapper

Utilice el mandato genMapper para generar un componente que hace de puente de una referencia de SCA (Service Component Architecture) con una interfaz Java.

Alteración temporal de una conversión de Service Data Object a Java

Puede que a veces la conversión que el sistema crea entre un SDO (Service Data Object) y un objeto de tipo Java no satisfaga sus necesidades. Utilice este procedimiento para sustituir la implementación por omisión por la suya propia.

Antes de empezar

Asegúrese de que ha generado el WSDL para la conversión del tipo Java utilizando WebSphere Integration Developer o el mandato genMapper.

Acerca de esta tarea

Puede alterar temporalmente un componente generado que correlaciona un tipo WSDL con un tipo Java sustituyendo el código generado por código que satisfaga sus necesidades. Considere utilizar su propia correlación si ha definido sus propias clases Java. Utilice este procedimiento para hacer los cambios.

Procedimiento

1. Localice el componente generado. El componente se denomina *java_class*Mapper.component.
2. Edite el componente utilizando un editor de texto.
3. Convierta en comentario el código generado y proporcione su propio método.

   No cambie el nombre de archivo que contiene la implementación del componente.

A continuación figura un ejemplo de un componente generado que se va a sustituir:

```
 private Object datatojava_get_customerAcct(DataObject myCustomerID,
    String integer)
{

  // Puede alterar temporalmente este código para la correlación personalizada.
  // Convierta en comentario este código y escriba código personalizado.

  // También puede cambiar el tipo Java que se pasa al
  // conversor, que este último intenta crear.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

Copie el componente y otros archivos al directorio en el que reside el módulo contenedor y conecte el componente de WebSphere Integration Developer o genere un archivo EAR (Enterprise Archive) con el mandato serviceDeploy.

Conceptos relacionados

"Normas de tiempo de ejecución utilizadas para la conversión de Java a Service Data Objects" en la página 18

Para alterar temporalmente el código generado o determinar posibles excepciones en tiempo de ejecución relacionadas con las conversiones de Java a Service Data Object (SDO), es importante tener un conocimiento de las normas correspondientes. La mayoría de las conversiones son directas, pero hay algunos casos complejos en que la ejecución proporciona la mejor posibilidad cuando convierte el código generado.

Referencia relacionada

Conversión de Java a XML

El sistema genera XML basado en tipos Java mediante la utilización de normas predefinidas.

Mandato genMapper

Utilice el mandato genMapper para generar un componente que hace de puente de una referencia de SCA (Service Component Architecture) con una interfaz Java.

Normas de tiempo de ejecución utilizadas para la conversión de Java a Service Data Objects

Para alterar temporalmente el código generado o determinar posibles excepciones en tiempo de ejecución relacionadas con las conversiones de Java a Service Data Object (SDO), es importante tener un conocimiento de las normas correspondientes. La mayoría de las conversiones son directas, pero hay algunos casos complejos en que la ejecución proporciona la mejor posibilidad cuando convierte el código generado.

Tipos y clases básicos

Durante la ejecución se realiza una conversión directa entre Service Data Objects y tipos y clases Java básicos. Entre los tipos y clases básicos se incluyen:

- Char o java.lang.Character
- Boolean
- Java.lang.Boolean
- Byte o java.lang.Byte
- Short o java.lang.Short
- Int o java.lang.Integer
- Long o java.lang.Long
- Float o java.lang.Float
- Double o 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[]

Clases y matrices Java definidas por el usuario

Cuando se realiza la conversión de una clase o matriz Java a un SDO, se crea un objeto de datos durante la ejecución que tiene un URI generado invirtiendo el nombre de paquete del tipo Java y con un tipo igual al nombre de la clase Java. Por ejemplo, la clase Java com.ibm.xsd.Customer se convierte a un objeto SDO y URI http://xsd.ibm.com con el tipo Customer. Durante la ejecución se inspecciona entonces el contenido de los miembros de la clase Java y se asignan los valores a las propiedades del SDO.

Al convertir un SDO a un tipo Java, durante la ejecución se genera el nombre de paquete invirtiendo el URI y con el nombre del tipo igual al tipo del SDO. Por ejemplo, el objeto de datos con tipo Customer y URI http://xsd.ibm.com genera una instancia del paquete Java com.ibm.xsd.Customer. Durante la ejecución luego se extraen los valores de las propiedades del SDO y se asignan esas propiedades a los campos de la instancia de la clase Java.

Cuando la clase Java es una interfaz definida por el usuario, debe alterar temporalmente el código generado y proporcionar una clase concreta de la que se pueda crear una instancia durante la ejecución. Si durante la ejecución no se puede crear la clase concreta, se produce una excepción.

Java.lang.Object

Cuando un tipo Java es java.lang.Object el tipo generado es xsd:anyType. Un módulo puede invocar esta interfaz con cualquier SDO. Durante la ejecución se intenta crear una instancia de una clase concreta del mismo modo que para las clases y matrices Java definidas por el usuario, si durante la ejecución se puede encontrar esa clase. De lo contrario, se pasa el SDO a la interfaz Java durante la ejecución.

Aun cuando el método devuelva un tipo java.lang.Object, se convertirá durante la ejecución a un SDO sólo si el método devuelve un tipo concreto. Durante la ejecución se utiliza una conversión similar a la que se utiliza para convertir las clases y matrices Java definidas por el usuario a objetos SDO, como se describe en el párrafo siguiente.

Cuando se realiza la conversión de una clase o matriz Java a un SDO, se crea un objeto de datos durante la ejecución que tiene un URI generado invirtiendo el nombre de paquete del tipo Java y con un tipo igual al nombre de la clase Java. Por ejemplo, la clase Java com.ibm.xsd.Customer se convierte a un objeto SDO y URI http://xsd.ibm.com con el tipo Customer. Durante la ejecución se inspecciona entonces el contenido de los miembros de la clase Java y se asignan los valores a las propiedades del SDO.

En cualquier caso, si durante la ejecución no se puede completar la conversión se produce una excepción.

Clases de contenedor Java.util

Al convertir a una clase de contenedor Java concreta, como Vector, HashMap, HashSet y por el estilo, se crea una instancia de la clase de contenedor adecuada durante la ejecución. Durante la ejecución se utiliza un método similar al que se utiliza para las clases y matrices Java definidas por el usuario para llenar la clase de contenedor. Si durante la ejecución no se puede localizar una clase Java concreta, se llena la clase de contenedor con el SDO.

Al convertir clases de contenedor Java concretas a objetos SDO durante la ejecución, se utilizan los esquemas generados que se muestran en la “conversión de Java a XML”.

Interfaces Java.util

Para determinadas interfaces de contenedor del paquete java.util, se crean instancias de las clases concretas siguientes durante la ejecución:

Tabla 1. Conversión del tipo WSDL a la clase Java

Interfaz	**Clases concretas por omisión**
Colección	HashSet
Correlación	HashMap
Lista	ArrayList
Conjunto	HashSet

Tareas relacionadas

“Alteración temporal de la implementación de la Service Component Architecture generada” en la página 15

Puede que a veces la conversión que el sistema crea entre un código Java y SDO (Service Data Object) no satisfaga sus necesidades. Utilice este procedimiento para sustituir la implementación por omisión de la clase SCA (Service Component Architecture) por la suya propia.

"Alteración temporal de una conversión de Service Data Object a Java" en la página 16
Puede que a veces la conversión que el sistema crea entre un SDO (Service Data Object) y un objeto de tipo Java no satisfaga sus necesidades. Utilice este procedimiento para sustituir la implementación por omisión por la suya propia.

Referencia relacionada


Conversión de Java a XML

El sistema genera XML basado en tipos Java mediante la utilización de normas predefinidas.


Mandato genMapper

Utilice el mandato genMapper para generar un componente que hace de puente de una referencia de SCA (Service Component Architecture) con una interfaz Java.

Capítulo 2. Desarrollo de aplicaciones cliente para procesos de empresa y tareas

Puede utilizar una herramienta de creación de modelos para generar y desplegar procesos de empresa y tareas. Se interactúa con estos procesos y tareas durante la ejecución; por ejemplo, se inicia un proceso o las tareas se reclaman y se completan. Puede utilizar Business Process Choreographer Explorer para interactuar con procesos y tareas, o utilizar las API de Business Process Choreographer para desarrollar clientes personalizados para estas interacciones.

Acerca de esta tarea

Estos clientes pueden ser clientes EJB (Enterprise JavaBeans), los clientes de servicios Web o los clientes Web que utilizan los componentes JSF (JavaServer Faces) de Business Process Choreographer Explorer. Business Process Choreographer proporciona las API de EJB (Enterprise JavaBeans) e interfaces para los servicios Web para que desarrolle estos clientes. Cualquier aplicación puede acceder a la API EJB mediante cualquier aplicación Java. Se puede acceder a las interfaces de servicios Web desde cualquier entorno Java o desde cualquier entorno Microsoft .Net.

Desarrollo de aplicaciones de cliente EJB para los procesos empresariales y tareas de usuario

Las API de EJB proporcionan un conjunto de métodos genéricos para desarrollar aplicaciones cliente EJB para trabajar con los procesos de empresa y tareas de usuario instalados en WebSphere Process Server.

Acerca de esta tarea

Con estas API EJB (Enterprise JavaBeans), puede crear aplicaciones de cliente para realizar lo siguiente:

- Gestionar el ciclo de vida de los procesos y tareas desde iniciarlas a suprimirlas cuando finalizan
- Reparar actividades y procesos
- Gestionar y distribuir la carga de trabajo en todos los miembros de un grupo de trabajo

Las API de EJB se proporcionan como dos enterprise bean de sesión sin estado.

- La interfaz BusinessFlowManagerService proporciona los métodos para las aplicaciones de proceso de empresa
- La interfaz HumanTaskManagerService proporciona los métodos para las aplicaciones basadas en tareas.

Para obtener más información sobre las API de EJB, consulte el Javadoc de los paquetes com.ibm.bpe.api y com.ibm.task.api.

En los pasos siguientes se proporciona una visión general de las acciones que es necesario realizar para desarrollar una aplicación de cliente EJB.

Procedimiento

1. Decida sobre la funcionalidad que la aplicación va a proporcionar.
2. Determine los beans de sesión que va a utilizar.

   Según los escenarios que desee implementar con la aplicación, puede utilizar uno de los beans de sesión o ambos.
3. Determine las autorizaciones que necesitan los usuarios de la aplicación.

   A los usuarios de la aplicación se les debe asignar los roles de autorización adecuados para llamar a los métodos que se incluyan en la aplicación y ver los objetos y los atributos de estos objetos que estos métodos devuelvan. Cuando se crea una instancia del bean de sesión adecuado, WebSphere Application Server asocia un contexto a la instancia. El contexto contiene información acerca del ID del principal del proceso que efectúa la llamada, la lista de miembros del grupo y los roles. Esta información se utiliza para comprobar la autorización del llamante para cada llamada.

   El Javadoc contiene información de autorización de todos los métodos.
4. Decida cómo representar la aplicación.

   Se puede llamar a las API de EJB de forma local o remota.
5. Desarrolle la aplicación.
   a. Acceda a la API de EJB.
   b. Utilice la API de EJB para interactuar con procesos o tareas.
      - Consulte los datos.
      - Trabaje con los datos.

Acceso a las API de EJB

Las API de EJB (Enterprise JavaBeans) se proporcionan como dos enterprise beans de sesión sin estado. Las aplicaciones de procesos de empresa y aplicaciones de tareas acceden al enterprise bean de sesión adecuado mediante la interfaz inicial del bean.

Acerca de esta tarea

La interfaz BusinessFlowManagerService proporciona los métodos para las aplicaciones de proceso de empresa y la interfaz HumanTaskManagerService proporciona los métodos para aplicaciones basadas en tareas. La aplicación puede ser cualquier aplicación Java, incluida otra aplicación EJB (Enterprise JavaBeans).

Acceso a la interfaz remota del bean de sesión

Una aplicación de cliente EJB accede a la interfaz remota del bean de sesión mediante la interfaz inicial remota del bean.

Acerca de esta tarea

El bean de sesión puede ser el bean de sesión de BusinessFlowManager para las aplicaciones de proceso o el bean de sesión HumanTaskManager para las aplicaciones de tareas.

Procedimiento

1. Añada una referencia a la interfaz remota del bean de sesión en el descriptor de despliegue de la aplicación. Añada la referencia a uno de los archivos siguientes:
   - El archivo application-client.xml, para una aplicación de cliente J2EE (Java 2 Platform, Enterprise Edition)
   - El archivo web.xml para una aplicación Web

- El archivo ejb-jar.xml para una aplicación EJB (Enterprise JavaBeans)

La referencia a la interfaz inicial remota para aplicaciones de proceso se muestra en el ejemplo siguiente:

```
<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 referencia a la interfaz inicial remota para aplicaciones de tarea se muestra en el ejemplo siguiente:

```
<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 utiliza WebSphere Integration Developer para añadir la referencia EJB al descriptor de despliegue, el enlace para la referencia EJB se crea automáticamente cuando se despliega la aplicación. Para obtener más información sobre cómo añadir referencias EJB, consulte la documentación de WebSphere Integration Developer.

2. Empaquete los módulos de programa generados con la aplicación.

   Si la aplicación se ejecuta en una JVM (Java Virtual Machine) distinta de aquélla en la que se ejecuta la aplicación BPEContainer o la aplicación TaskContainer, realice las acciones siguientes:

   a. Para las aplicaciones de proceso, empaquete el archivo *<raíz_instalación>*/ProcessChoreographer/client/bpe137650.jar con el archivo EAR (Enterprise Archive) de la aplicación.
   b. Para las aplicaciones de tarea, empaquete el archivo *<raíz_instalación>*/ProcessChoreographer/client/task137650.jar con el archivo EAR de la aplicación.
   c. Establezca el parámetro **Classpath** del archivo manifest del módulo de la aplicación para que incluya el archivo JAR.

      El módulo de aplicación puede ser una aplicación J2EE, una aplicación Web o una aplicación EJB.
   d. Si utiliza los tipos de datos complejos en el proceso de empresa o tarea de usuario y el cliente no se ejecuta en una aplicación EJB o en una aplicación Web, empaquete los archivos XSD o WSDL correspondiente con el archivo EAR de la aplicación.

3. Localice la interfaz inicial remota del bean de sesión mediante JNDI (Java Naming and Directory Interface).

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

```
// Obtener el contexto inicial por omisión de JNDI
InitialContext initialContext = new InitialContext();

// Buscar la interfaz inicial remota del bean BusinessFlowManager
Object result =
       initialContext.lookup("java:comp/env/ejb/BusinessFlowManagerHome");

// Convertir el resultado de búsqueda al tipo adecuado
   BusinessFlowManagerHome processHome =
           (BusinessFlowManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,BusinessFlowManagerHome.class);
```

La interfaz inicial remota del bean de sesión contiene un método create para objetos EJB. El método devuelve la interfaz remote del bean de sesión.

4. Acceda a la interfaz remota del bean de sesión.

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

   ```
   BusinessFlowManager process = processHome.create();
   ```

   El acceso al bean de sesión no garantiza que el proceso que efectúa la llamada pueda realizar todas las acciones proporcionadas por el bean, además, dicho proceso debe tener autorización para estas acciones. Cuando se crea una instancia del bean de sesión, se asocia un contexto a la instancia del bean de sesión. El contexto contiene el ID del principal del proceso que efectúa la llamada, la lista de miembros del grupo e indica si dicho proceso tiene uno de los roles J2EE de Business Process Choreographer. El contexto se utiliza para comprobar la autorización para cada llamada del proceso que efectúa la llamada, incluso cuando no se ha establecido la seguridad global. Si no se ha establecido la seguridad global el ID del principal del proceso que efectúa la llamada tiene el valor `UNAUTHENTICATED`.

5. Llame a las funciones de empresa expuestas por la interfaz de servicio.

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

   ```
   process.initiate("MyProcessModel",input);
   ```

   Las llamadas procedentes de las aplicaciones se ejecutarán como transacciones. Se establece y finaliza una transacción de alguna de las formas siguientes:
   - Automáticamente, por parte de WebSphere Application Server (el descriptor de despliegue especifica TX_REQUIRED).
   - Explícitamente, por parte de la aplicación. Puede empaquetar las llamadas de aplicación en una transacción:

     ```
     // Obtener interfaz de transacción de usuario
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("jta/usertransaction");

     // Iniciar una transacción
     transaction.begin();

     // Llamadas de aplicaciones ...

     // Cuando se devuelva de forma satisfactoria, confirmar la transacción
     transaction.commit();
     ```

   Consejo: Para impedir los conflictos de bloqueo en la base de datos, procure no ejecutar las sentencias similares a las siguientes en paralelo:

   ```
   // Obtener interfaz de transacción de usuario
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup("jta/usertransaction");

   transaction.begin();

   //leer la instancia de actividad
   process.getActivityInstance(aiid);
   //reclamar la instancia de actividad
   process.claim(aiid);

   transaction.commit();
   ```

   El método getActivityInstance y otras operaciones de lectura establecen un bloqueo de lectura. En este ejemplo, un bloqueo de lectura en la instancia de actividad se actualiza a un bloqueo de actualización en la instancia de actividad. Esto puede resultar en un punto muerto en la base de datos si estas transacciones se ejecutan en paralelo.

Ejemplo

Éste es un ejemplo de cómo los pasos 3 a 5 pueden buscar una aplicación de tareas.

```
// Obtener el contexto inicial por omisión de JNDI
InitialContext initialContext = new InitialContext();

// Buscar la interfaz inicial remota del bean HumanTaskManager
Object result =
         initialContext.lookup("java:comp/env/ejb/HumanTaskManagerHome");

// Convertir el resultado de búsqueda al tipo adecuado
HumanTaskManagerHome taskHome =
           (HumanTaskManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,HumanTaskManagerHome.class);


...
//Acceder a la interfaz remota del bean de sesión.
HumanTaskManager task = taskHome.create();
...
//Llamar a las funciones de empresa expuestas por la interfaz de servicio
task.callTask(tkiid,input);
```

Acceso a la interfaz local del bean de sesión

Una aplicación de cliente EJB accede a la interfaz local del bean de sesión mediante la interfaz inicial local del bean.

Acerca de esta tarea

El bean de sesión puede ser el bean de sesión de BusinessFlowManager para las aplicaciones de proceso o el bean de sesión HumanTaskManager para las aplicaciones de tareas de usuario.

Procedimiento

1. Añada una referencia a la interfaz local del bean de sesión en el descriptor de despliegue de la aplicación. Añada la referencia a uno de los archivos siguientes:
   - El archivo application-client.xml, para una aplicación de cliente J2EE (Java 2 Platform, Enterprise Edition)
   - El archivo web.xml para una aplicación Web
   - El archivo ejb-jar.xml para una aplicación EJB (Enterprise JavaBeans)

   La referencia a la interfaz inicial local para aplicaciones de proceso se muestra en el ejemplo siguiente:

   ```
   <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 referencia a la interfaz inicial local para aplicaciones de tarea se muestra en el ejemplo siguiente:

   ```
   <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 utiliza WebSphere Integration Developer para añadir la referencia EJB al descriptor de despliegue, el enlace para la referencia EJB se crea

automáticamente cuando se despliega la aplicación. Para obtener más información sobre cómo añadir referencias EJB, consulte la documentación de WebSphere Integration Developer.

2. Localice la interfaz inicial local del bean de sesión mediante JNDI (Java Naming and Directory Interface).

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

   ```
   // Obtener el contexto inicial por omisión de JNDI
   InitialContext initialContext = new InitialContext();

   // Buscar la interfaz inicial local del bean BusinessFlowManager

    LocalBusinessFlowManagerHome processHome =
         (LocalBusinessFlowManagerHome)initialContext.lookup
         ("java:comp/env/ejb/LocalBusinessFlowManagerHome");
   ```

   La interfaz inicial local del bean de sesión contiene un método create para objetos EJB. El método devuelve la interfaz local del bean de sesión.

3. Acceda a la interfaz local del bean de sesión local.

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

   ```
   LocalBusinessFlowManager process = processHome.create();
   ```

   El acceso al bean de sesión no garantiza que el proceso que efectúa la llamada pueda realizar todas las acciones proporcionadas por el bean, además, dicho proceso debe tener autorización para estas acciones. Cuando se crea una instancia del bean de sesión, se asocia un contexto a la instancia del bean de sesión. El contexto contiene el ID del principal del proceso que efectúa la llamada, la lista de miembros del grupo e indica si dicho proceso tiene uno de los roles J2EE de Business Process Choreographer. El contexto se utiliza para comprobar la autorización para cada llamada del proceso que efectúa la llamada, incluso cuando no se ha establecido la seguridad global. Si no se ha establecido la seguridad global el ID del principal del proceso que efectúa la llamada tiene el valor `UNAUTHENTICATED`.

4. Llame a las funciones de empresa expuestas por la interfaz de servicio.

   El ejemplo siguiente muestra este paso para una aplicación de proceso:

   ```
   process.initiate("MyProcessModel",input);
   ```

   Las llamadas procedentes de las aplicaciones se ejecutarán como transacciones. Se establece y finaliza una transacción de alguna de las formas siguientes:
   - Automáticamente, por parte de WebSphere Application Server (el descriptor de despliegue especifica TX_REQUIRED).
   - Explícitamente, por parte de la aplicación. Puede empaquetar las llamadas de aplicación en una transacción:

     ```
     // Obtener interfaz de transacción de usuario
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("jta/usertransaction");

     // Iniciar una transacción
     transaction.begin();

     // Llamadas de aplicaciones ...

     // Cuando se devuelva de forma satisfactoria, confirmar la transacción
     transaction.commit();
     ```

   Consejo: Para impedir los puntos muertos en la base de datos, procure no ejecutar las sentencias similares a las siguientes en paralelo:

```
// Obtener interfaz de transacción de usuario
UserTransaction transaction=
       (UserTransaction)initialContext.lookup("jta/usertransaction");

transaction.begin();

//leer la instancia de actividad
process.getActivityInstance(aiid);
//reclamar la instancia de actividad
process.claim(aiid);

transaction.commit();
```

El método getActivityInstance y otras operaciones de lectura establecen un bloqueo de lectura. En este ejemplo, un bloqueo de lectura en la instancia de actividad se actualiza a un bloqueo de actualización en la instancia de actividad. Esto puede resultar en un punto muerto en la base de datos si estas transacciones se ejecutan en paralelo.

Ejemplo

Éste es un ejemplo de cómo los pasos 2 a 4 pueden buscar una aplicación de tareas.

```
// Obtener el contexto inicial por omisión de JNDI
InitialContext initialContext = new InitialContext();

// Buscar la interfaz inicial local del bean HumanTaskManager
LocalHumanTaskManagerHome taskHome =
        (LocalHumanTaskManagerHome)initialContext.lookup
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Acceder a la interfaz local del bean de sesión local
LocalHumanTaskManager task = taskHome.create();
...
//Llamar a las funciones de empresa expuestas por la interfaz de servicio
task.callTask(tkiid,input);
```

Consulta sobre los objetos de procesos de empresa y relativos a tareas

Las aplicaciones cliente trabajan con objetos de procesos de empresa y relacionados con tareas. Puede consultar objetos de proceso de emprea y relativos a tareas en la base de datos para recuperar propiedades específicas de estos objetos.

Acerca de esta tarea

Durante la configuración de Business Process Choreographer, una base de datos relacional se asocia con el contenedor de procesos de empresa y con el contenedor de tareas. La base de datos almacena todos los datos de plantillas (modelo) e instancias (tiempo de ejecución) para gestionar procesos de empresa y tareas. Utilice una sintaxis análoga a SQL para consultar estos datos.

Puede realizar una consulta específica para recuperar una propiedad concreta de un objeto. También puede guardar consultas que utilice a menudo e incluir estas consultas almacenadas en la aplicación.

Consultas sobre los objetos de procesos de empresa y relativos a tareas

Utilice el método query o el método queryAll de la API de servicio para recuperar la información almacenada acerca de los procesos de empresa y tareas.

Todos los usuarios pueden llamar al método query, que devuelve las propiedades de los objetos para los que existen elementos de trabajo. Solo los usuarios que tengan uno de los siguientes roles J2EE: BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor o TaskSystemMonitor, pueden llamar al método queryAll. Este método devuelve las propiedades de todos los objetos que están almacenados en la base de datos.

Todas las consultas API están correlacionadas con consultas SQL. La forma de la consulta SQL resultante depende de los aspectos siguientes:

- Si alguien con uno de los roles J2EE ha invocado la consulta.
- Los objetos que se consultan. Se proporcionan vistas de bases de datos predefinidas para que se consulten las propiedades de objeto.
- La inserción de una cláusula from, condiciones de unión y condiciones específicas de usuario para el control de acceso.

Puede incluir propiedades personalizadas y propiedades de variables en consultas. Si incluye varias propiedades personalizadas o propiedades de variables en la consulta, se producen uniones automáticas en la tabla de base de datos correspondiente. En función del sistema de base de datos, estas llamadas a `query()` podrían tener implicaciones en el rendimiento.

También puede almacenar consultas en la base de datos de Business Process Choreographer utilizando el método createStoredQuery. Proporcione el criterio de consutla cuando defina la consulta almacenada. El criterio se aplica dinámicamente cuando se ejcuta la consulta almacenada, esto es, los datos se ensamblan durante la ejecución. Si la consulta almacenada contiene parámetros, estos se resuelven también cuando se ejecuta la consulta.

Para obtener más información sobre las API de Business Process Choreographer, consulte el Javadoc en el paquete com.ibm.bpe.api para los métodos relativos a procesos y el paquete com.ibm.task.api para los métodos relativos a tareas.

Sintaxis del método query de la API:

La sintaxis de las consultas de la API de Business Process Choreographer es parecida a la de las consultas SQL. Una consulta puede incluir una cláusula select, una cláusula where, una cláusula order-by, un parámetro skip-tuples, un parámetro threshold y un parámetro time-zone.

La sintaxis de la consulta depende del tipo de objeto. En la tabla siguiente se muestra la sintaxis de cada uno de los distintos tipos de objeto.

Tabla 2.

Objeto	Sintaxis
Plantilla de proceso	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`

Tabla 2. (continuación)

Objeto	Sintaxis
Plantilla de tarea	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Datos de proceso de llamada y datos relacionados con tareas	`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);`

Cláusula select:

La cláusula select de la función de consulta identifica las propiedades de objeto que una consulta ha de devolver.

La cláusula select describe el resultado de la consulta. Especifica una lista de los nombres que identifican las propiedades del objeto (las columnas del resultado) que se han de devolver. Su sintaxis es parecida a la de una cláusula SELECT de SQL; utiliza comas para separar las partes de la cláusula. Cada parte de la cláusula debe especificar una columna de una de las vistas predefinidas. Las columnas deben estar totalmente especificadas por nombre de vista y nombre de columna. Las columnas devueltas en el objeto QueryResultSet aparecerán con el mismo orden que las columnas especificadas en la cláusula select.

La cláusula select no tiene soporte para las funciones de agregación de SQL, como AVG(), SUM(), MIN() o MAX().

Para seleccionar las propiedades de varios pares de nombre y valor como, por ejemplo, las propiedades personalizadas y las propiedades de variables que se pueden consultar, añada un contador de un dígito al nombre de vista. Este contador puede tomar los valores de 1 a 9.

Ejemplos de cláusulas select

- ″WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON″

  Obtiene los tipos de objeto de los objetos asociados y las razones de asignación de los elementos de trabajo.
- ″DISTINCT WORK_ITEM.OBJECT_ID″

  Obtiene todos los ID de objetos, sin duplicados, para los que el llamante tiene un elemento de trabajo.
- ″ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON″

  Obtiene los nombres de las actividades para las que el llamante tiene elementos de trabajo y los motivos de asignación.
- ″ACTIVITY.STATE, PROCESS_INSTANCE.STARTER″

  Obtiene los estados de las actividades y los iniciadores de sus instancias de proceso asociadas.
- ″DISTINCT TASK.TKIID, TASK.NAME″

  Obtiene todos los ID y nombres de tareas, sin duplicados, para los que el llamante tiene un elemento de trabajo.
- ″TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE″

Obtiene los valores de las propiedades personalizadas que se especifican con más detalle en la cláusula where.

- ″QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE

  Obtiene los valores de las propiedades de variables que se pueden consultar. Estas partes se especifican más en la cláusula where.
- ″COUNT(DISTINCT TASK.TKIID)″

  Cuenta el número de elementos de trabajo para las taeras exclusivas que satisfacen la cláusula where.

Cláusula where:

La cláusula where de la función de consulta describe los criterios de filtro que se aplicarán al dominio de consulta.

La sintaxis de una cláusula where es parecida a la de una cláusula WHERE de SQL. No es necesario añadir explícitamente una cláusula from de SQL o predicados de unión a la cláusula where de la API, estas construcciones se añaden automáticamenet cuando se ejecuta la consulta. Si no quiere aplicar criterios de filtro, debe especificar `null` para la cláusula where.

La sintaxis de la cláusula where tiene soporte para:

- Palabras clave: AND, OR, NOT
- Operadores de comparación: =, <=, <, <>, >,>=, LIKE

  La operación LIKE admite los caracteres comodín definidos para la base de datos consultada.
- Operación de definición: IN

También se aplican las normas siguientes:

- Especifique las constantes de ID de objeto como `ID('string-rep-of-oid')`.
- Especifique las constantes binarias como `BIN('UTF-8 string')`.
- Utilice constantes simbólicas en lugar de enumeraciones de enteros. Por ejemplo, en vez de especificar una expresión de estado de actividad `ACTIVITY.STATE=2`, especifique `ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY`.
- Si el valor de la propiedad en la sentencia de comparación contiene apóstrofos (’), doble las comillas; por ejemplo, `"TASK_CPROP.STRING_VALUE='d''automatisation'"`.
- Consulte las propiedades de varios pares de nombre y valor como, por ejemplo, las propiedades personalizadas, añadiendo un sufijo de un dígito al nombre de vista. Por ejemplo: `"TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'"`
- Especifique las constantes de indicación de la hora como `TS('yyyy-mm-ddThh:mm:ss')`. para consultar la fecha actual, especifique `CURRENT_DATE` como la indicación de la hora.

  Debe especificar como mínimo un valor de fecha o de hora en la indicación de la hora:
  - Si especifica solamente una fecha, el valor de hora se establece en cero.
  - Si especifica solamente una hora, la fecha se establece en la fecha actual.
  - Si especifica una fecha, el año debe constar de cuatro dígitos. Los valores de mes y día son opcionales. Si faltan los valores de mes y día se establecen en 01. Por ejemplo, `TS('2003')` es igual que `TS('2003-01-01T00:00:00')`.

– Si especifica una hora, estos valores se expresan en el sistema de 24 horas. Por ejemplo, si la fecha actual es 1 de Enero de 2003, `TS('T16:04')` o `TS('16:04')` es igual que `TS('2003-01-01T16:04:00')`.

Ejemplos de cláusulas where

- Comparación de un ID de objeto con un ID existente

  ```
  "WORK_ITEM.WIID = ID('_WI:800c00ed.df8d7e7c.feffff80.38')"
  ```

  Este tipo de cláusula where suele crearse de forma dinámica con un ID de objeto existente a partir de una llamada anterior. Si este ID de objeto se guarda en una variable *wiid1*, la cláusula podría construirse como:

  ```
  "WORK_ITEM.WIID = ID('" + wiid1.toString() + "')"
  ```

- Utilización de las indicaciones de la hora

  ```
  "ACTIVITY.STARTED >= TS('2002-06-1T16.00.00')"
  ```

- Uso de constantes simbólicas

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

- Uso de los valores booleanos true y false

  ```
  "ACTIVITY.BUSINESS_RELEVANCE = TRUE"
  ```

- Uso de propiedades personalizadas

  ```
  "TASK_CPROP1.NAME = 'prop1' AND " TASK_CPROP1.STRING_VALUE = 'v1' AND
   TASK_CPROP2.NAME = 'prop2' AND " TASK_CPROP2.STRING_VALUE = 'v2'"
  ```

Cláusula order-by:

La cláusula order-by de la función de consulta especifica los criterios de clasificación del conjunto de resultados de la consulta.

Puede especificar una lista de columnas de las vistas según las que se clasifica el resultado. Estas columnas deben estar totalmente calificadas según el nombre de la vista y la columna. Es recomendable especificar columnas que estén en la cláusula select.

La sintaxis de la cláusula order-by es parecida a la de una cláusula order-by de SQL; para separar cada parte de la cláusula debe utilizar comas. También puede especificar `ASC` para clasificar las columnas en orden ascendente y `DESC` para clasificar las columnas en orden descendente. Si no quiere clasificar el conjunto de resultados de consulta, debe especificar `null` para la cláusula order-by.

Se aplican criterios de clasificación en el servidor, es decir, se utiliza el entorno local del servidor para realizar la clasificación. Si especifica más de una columna, el conjunto de resultados de la consulta se clasifica por los valores de la primera columna, luego por los valores de la segunda columna y así sucesivamente. No puede especificar las columnas en la cláusula order-by por posición como sí sucede con una consulta SQL.

Ejemplos de cláusulas order-by

- ″PROCESS_TEMPLATE.NAME″

  Clasifica el resultado de la consulta en orden alfabético por el nombre de la plantilla de proceso.

- ″PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC″

  Clasifica el resultado de la consulta por la fecha de creación y, para una fecha determinada, clasifica el resultado en orden alfabético inverso por el nombre de instancia de proceso.

- "ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE"

  Clasifica el resultado de la consulta por el propietario de la actividad, luego por nombre de plantilla de la actividad y luego por el estado de la actividad.

Parámetro skip-tuples:

El parámetro skip-tuples especifica el número de tuples de tipo query-result-set desde el principio del conjunto de resultados de la consulta que se deben pasar por alto y no devolverse al llamante en el conjunto de resultados de la consulta.

Utilice este parámetro con el parámetro threshold para implementar la paginación de una aplicación de cliente, por ejemplo, para recuperar los primeros 20 elementos, a continuación, los siguientes 20 elementos, etc.

Si este parámetro se establece en `null` y no se establece el parámetro threshold, se devuelven todos los tuples cualificados.

Ejemplo de parámetro skip-tuples

- new Integer(5)

  Especifica que no deben devolverse los cinco primeros tuples calificados.

Parámetro threshold:

El parámetro de umbral (threshold) de la función de consulta restringe el número de objetos devueltos del servidor al cliente en el conjunto de resultados de consulta.

Debido a que los conjuntos de resultados de la consula en los escenarios de producción pueden contener miles o incluso millones de elementos, se recomienda especificar siempre un umbral. El parámetro threshold puede ser útil, por ejemplo, en una interfaz gráfica de usuario en la que solamente deben visualizarse un número reducido de elementos. Si establece el parámetro threshold en consecuencia, la consulta de base de datos es más rápida y es necesario transferir menos datos del servidor al cliente.

Si este parámetro se establece en `null` y no se establece el parámetro skip-tuples, se devuelven todos los objetos cualificados.

Ejemplo de un parámetro threshold

- new Integer(50)

  Especifica que han de devolverse 50 tuples cualificados.

Parámetro timezone:

El parámetro time-zone de la función de consulta define el huso horario para las constantes de indicación de la hora de la consulta.

Los husos horarios del cliente que inicia la consulta y el servidor que la procesa, pueden ser distintos. Utilice el parámetro de huso horario, time-zone, para especificar el huso horario de las constantes de indicación de la hora que se utilizan, por ejemplo, en la cláusula where para especificar la hora local. Las fechas devueltas en el conjunto de resultados de la consulta tienen el mismo huso horario que las especificadas en la consulta.

Si el valor del parámetro se establece en `null`, se presupone que las constantes de timestamp tienen el formato UTC (Coordinated Universal Time).

Ejemplos de parámetros de huso horario

- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null,
                 (Integer)null,
                 java.util.TimeZone.getDefault() );
  ```

  Devuelve los ID de objeto de las actividades que se iniciaron después de las 17:40, hora local, el 1 de enero de 2005.
- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null, (Integer)null, (TimeZone)null);
  ```

  Devuelve los ID de objeto de las actividades que se iniciaron después de las 17:40, hora UTC, el 1 de enero de 2005. Esta especificación es, por ejemplo, 6 horas antes en la hora estándar del Este.

Parámetros de las consultas almacenadas:

Una consulta almacenada es una consulta que se almacena en la base de datos y se identifica con un nombre. Los tuples de calificación se ensamblan dinámicamente cuando se ejecuta la consulta. Para que las consultas almacenadas se puedan volver a utilizar, puede utilizar los parámetros de la definición de consulta que se resuelven durante la ejecución.

Por ejemplo, supongamos que ha definido propiedades personalizadas para almacenar nombres de cliente. Puede definir las consultas para devolver las tareas que se asocian a un cliente determinado, ACME Co. Para consultar esta información, la cláusula where de la consulta sería similar al ejemplo siguiente:

```
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.'";
```

Para hacer que esta consulta sea reutilizable de manera que pueda buscar también el cliente, BCME Ltd, puede utilizar parámetros para los valores de la propiedad personalizada. Si añade parámetros a la consulta de tarea, podría ser similar al ejemplo siguiente:

```
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'";
```

El parámetro `@param1` se resuelve en tiempo de ejecución de la lista de parámetros que se pasa al método query. Se aplican las siguientes normas para el uso de parámetros en las consultas:

- Los parámetros sólo se pueden utilizar en la cláusula where.
- Los parámetros son de tipo string.
- Los parámetros se sustituyen durante la ejecución utilizando la sustitución de la serie. Si necesita caracteres especiales debe especificarlos en la cláusula where o pasarlos durante la ejecución como parte del parámetro.
- Los nombres de parámetro constan de la serie `@param` concatenada con un número entero. El número inferior es 1, que señala al primer elemento de la lista de parámetros que se pasa a la API de consulta durante la ejecución.

- Se puede utilizar un parámetro varias veces en una cláusula where. Todas las apariciones del parámetro se sustituyen por el mismo valor.

  Tareas relacionadas

  "Gestión de consultas almacenadas" en la página 61
  Las consultas almacenadas proporcionan un modo de guardar las consultas que se ejecutan con frecuencia. La consulta almacenada puede ser una consulta que está disponible para todos los usuarios (consulta pública) o una consulta que pertenece a un usuario específico (consulta privada).

Resultados de la consulta:

Un conjunto de resultados de consulta contiene el resultado de una consulta.

Los elementos del conjunto de resultados son propiedades de los objetos que satisfacen la cláusula where proporcionada por el llamante y que este tiene autorización para ver. Puede leer elementos de lectura de forma relativa utilizando el método de API next de modo absoluto utilizando los métodos first y last. Como el cursor implícito de un conjunto de resultados de consulta está inicialmente posicionado antes del primer elemento, debe llamar a los métodos first o next antes de leer un elemento. Puede utilizar el método size para determinar el número de elementos del conjunto.

Un elemento de un conjunto de resultados de consulta comprende los atributos seleccionados de los elementos de trabajo y sus objetos referenciados asociados, como instancias de actividad y de proceso. El primer atributo (columna) de un elemento QueryResultSet especifica el valor del primer atributo especificado en la cláusula select de la petición de consulta. El segundo atributo (columna) de un elemento QueryResultSet especifica el valor del segundo atributo especificado en la cláusula select de la petición de consulta, y así sucesivamente.

Se pueden recuperar los valores de los atributos mediante la invocación de un método que es compatible con el tipo de atributo y mediante la especificación del índice de columna adecuado. La numeración de los índices de columna comienza por 1.

Tipo de atributo	**Método**
String	getString
OID	getOID
Timestamp	getTimestamp getString getTimestampAsLong
Integer	getInteger getShort getLong getString getBoolean
Boolean	getBoolean getShort getInteger getLong getString
byte[]	getBinary

Ejemplo:

Se ejecuta la consulta siguiente:

```
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);
```

El conjunto de resultados de consulta devuelto tiene cuatro columnas:
- La columna 1 es una indicación de la hora
- La columna 2 es una serie
- La columna 3 es un ID de objeto
- La columna 4 es un entero

Puede utilizar los métodos siguientes para recuperar los valores de atributo:

```
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);
}
```

Puede utilizar los nombres de visualización del conjunto de resultados, por ejemplo, como cabeceras para imprimir una tabla. Son los nombres de columna de la vista o el nombre definido mediante la cláusula AS en la consulta. Puede utilizar el método siguiente para recuperar los nombres de visualización del ejemplo:

```
resultSet.getColumnDisplayName(1) returns "STARTED"
resultSet.getColumnDisplayName(2) returns "NAME"
resultSet.getColumnDisplayName(3) returns "WIID"
resultSet.getColumnDisplayName(4) returns "REASON"
```

Condiciones de acceso específicas del usuario:

Las condiciones de acceso específicas del usuario se añaden cuando se genera la declaración de SQL SELECT desde la consulta de API. Estas condiciones garantizan que solo estos objetos se devuelven al llamante para satisfacer la condición especificada por el mismo y para la que está autorizado.

La condición de acceso que se añade depende de si el usuario es un administrador del sistema.

Consultas invocadas por usuarios que no sean administradores del sistema

La cláusula de SQL WHERE generada combina la cláusula where de la API con una condición de control de acceso que sea específica del usuario. La consulta recupera solo los objetos a los que el usuario está autorizado a acceder, es decir, solo aquellos objetos para los que el usuario tiene un elemento de trabajo. Un elemento de trabajo representa la asignación de un usuario o grupo de usuarios a un rol de autorización de un objeto de empresa, como una tarea o proceso. Si, por ejemplo, el usuario Juan Torres es miembro del rol de propietarios potenciales de una tarea determinada, existe un objeto de elemento de trabajo que representa esta relación.

Por ejemplo, si un usuario, que no sea un administrador del sistema, consulta tareas, se añade la siguiente condición de acceso a la cláusula WHERE si no se han habilitado elementos de trabajo de grupo:

```
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 )
```

Por tanto, si Juan Torres desea obtener una lista de tareas para las que es el propietario potencial, la cláusula where de la API podría tener este aspecto:

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

Esta cláusula where de la API resulta en la siguiente condición de acceso en la declaración SQL:

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JuanTorres'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

Esto también significa que si Juan Torres desea ver las actividades y tareas para las que él es un lector de procesos o un administrador de procesos y para las que no tiene un elemento de trabajo, debe añadirse una propiedad de la vista PROCESS_INSTANCE a la cláusula select, where u order-by de la consulta, por ejemplo, PROCESS_INSTANCE.PIID.

Si se habilitan los elementos de trabajo de grupo, se añade una condición de acceso adicional a la cláusula WHERE que permite que un usuario acceda a objetos para los que el grupo tiene acceso.

Consultas invocadas por administradores del sistema

Los administradores del sistema pueden invocar el método query para recuperar objetos que tengan elementos de trabajo asociados. En este caso, se añade una unión con la vista WORK_ITEM a la consulta SQL generada, pero no se añade una condición de control de acceso para WORK_ITEM.OWNER_ID.

En este caso, la consulta SQL para tareas contiene lo siguiente:

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

Consultas queryAll

Este tipo de consulta solo se puede invocar por parte de los administradores del sistema o supervisores del sistema. No se añade ninguna condición para el control de acceso ni una unión a la vista WORK_ITEM. Este tipo de consulta devuelve todos los datos para todos los objetos.

Ejemplos de los métodos query y queryAll:

Estos ejemplos muestran la sintaxis de varias consultas típicas de API y las declaraciones SQL asociadas que se generan al procesar la consulta.

Ejemplo: consulta de tareas en estado preparado:

Este ejemplo muestra cómo utilizar el método query para recuperar tareas con las que puede trabajar el usuario que ha iniciado la sesión.

Juan Torres desea obtener una lista de las tareas que se le han asignado. Para que un usuario pueda trabajar en una tarea, esta debe estar en estado preparado. El

usuario que ha iniciado la sesión también debe tener un elemento de trabajo de propietario potencial para la tarea. El fragmento de código siguiente muestra la llamda al método query para esta consulta:

```
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 )
```

Se adoptan las acciones siguientes cuando se genera la declaración SELECT:

- Se añade una condición para el control de acceso a la cláusula where. En este ejemplo se asume que los elementos de trabajo de grupo no están habilitados.
- Las constantes, como TASK.STATE.STATE_READY, se sustituyen por sus valores numéricos.
- Se añade una cláusula FROM y condiciones de unión.

El fragmento de código siguiente muestra la sentencia SQL que se genera a partir de la consulta de 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 = 'JuanTorres' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Para restringir la consulta de API a tareas para un proceso determinado, por ejemplo, sampleProcess, la consulta tiene el aspecto siguiente:

```
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 )
```

Ejemplo: consulta de tareas en estado reclamado:

Este ejemplo muestra cómo utilizar el método query para recuperar tareas que ha reclamado el usuario que ha iniciado la sesión.

El usuario, Juan Torres, desea buscar tareas que ha reclamado y que todavía están en estado reclamado. La condición que especifica ″reclamado por Juan Torres″ es TASK.OWNER = 'JuanTorres'. El fragmento de código siguiente muestra la llamda al método query para la consulta:

```
query( "DISTINCT TASK.TKIID",
           "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
           "TASK.OWNER = 'JuanTorres'",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

El fragmento de código siguiente muestra la sentencia SQL que se genera a partir de la consulta de API:

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

Cuando se reclama una tarea, se crean elementos de trabajo para el propietario de la tarea. Por tanto, un método alternativo de formular la consulta para las tareas reclamadas de Juan Torres consiste en añadir la condición siguiente a la consulta, en lugar de utilizar `TASK.OWNER = 'JuanTorres'`:

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

La consulta tiene el aspecto del siguiente fragmento de código:

```
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 )
```

Se adoptan las acciones siguientes cuando se genera la declaración SELECT:

- Se añade una condición para el control de acceso a la cláusula where. En este ejemplo se asume que los elementos de trabajo de grupo no están habilitados.
- Las constantes, como `TASK.STATE.STATE_READY`, se sustituyen por sus valores numéricos.
- Se añade una cláusula FROM y condiciones de unión.

El fragmento de código siguiente muestra la sentencia SQL que se genera a partir de la consulta de 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 = 'JuanTorres' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Juan está a punto de irse de vacaciones, por lo que su jefe, Ana García, desea comprobar su carga de trabajo actual. Ana tiene derechos de administrador del sistema. La consulta que invoca es la misma que la que ha invocado Juan. Sin embargo, la declaración SQL que se genera es diferente porque Ana es un administrador. El fragmento de código siguiente muestra la sentencia SQL que se genera:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  TA.TKIID = WI.OBJECT_ID =
   AND    TA.STATE = 8
   AND   TA.OWNER = 'JuanTorres')
```

Dado que Ana es un administrador, no se añade una condición de control de acceso a la cláusula WHERE.

Ejemplo: consulta de escaladas:

Este ejemplo muestra cómo utilizar el método query para recuperar escaladas para el usuario que ha iniciado la sesión.

Cuando se escala una tarea, se crea un elemento de trabajo del destinatario de escalada. El usuario, Mary Jones, desea ver una lista de las tareas que se le han escalado. El fragmento de código siguiente muestra la llamda al método query para la consulta:

```
query( "DISTINCT ESCALATION.ESIID, ESCALATION.TKIID",
         "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_ESCALATION_RECEIVER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Se adoptan las acciones siguientes cuando se genera la declaración SELECT:

- Se añade una condición para el control de acceso a la cláusula where. En este ejemplo se asume que los elementos de trabajo de grupo no están habilitados.
- Las constantes, como TASK.STATE.STATE_READY, se sustituyen por sus valores numéricos.
- Se añade una cláusula FROM y condiciones de unión.

El fragmento de código siguiente muestra la sentencia SQL que se genera a partir de la consulta de API:

```
SELECT DISTINCT ESCALATION.ESIID, ESCALATION.TKIID
  FROM   ESCALATION ESC, WORK_ITEM WI
  WHERE  ESC.ESIID = WI.OBJECT_ID
  AND    WI.REASON  = 10
  AND
 ( WI.OWNER_ID = 'MaryJones' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Ejemplo: uso del método queryAll:

Este ejemplo muestra cómo utilizar el método queryAll para recuperar todas las actividades que pertenecen a una plantilla de proceso.

El método queryAll sólo está disponible para usuarios con derechos de administrador del sistema o de supervisor del sistema. El fragmento de código siguiente muestra la llamada al método queryAll para que la consulta recupere todas las actividades que pertenecen a la plantilla de proceso, sampleProcess:

```
queryAll( "DISTINCT ACTIVITY.AIID",
       "PROCESS_TEMPLATE.NAME = 'sampleProcess'",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

El fragmento de código siguiente muestra la consulta SQL que se genera a partir de la consulta de la API:

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

Dado que un administrador es quien invoca la llamada, no se añade una condición de control de acceso a la declaración SQL generada. Tampoco se añade una unión con la vista WORK_ITEM. Esto significa que la consulta recupera todas las actividades para la plantilla de proceso, incluidas las actividades sin elementos de trabajo.

Ejemplo: inclusión de propiedades de consulta en una consulta:

Este ejemplo muestra cómo utilizar el método query para recuperar tareas que pertenecen a un proceso de empresa. El proceso tiene propiedades de consulta definidas para dicho proceso que se pueden incluir en la búsqueda.

Por ejemplo, si desea buscar todas las tareas de usuario en estado preparado que pertenecen a un proceso de empresa. El proceso tiene una propiedad de consulta, **customerID**, con el valor CID_12345 y un espacio de nombres. El fragmento de código siguiente muestra la llamda al método query para la consulta:

```
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 ahora desea añadir una segunda propiedad de consulta a la consulta, por ejemplo **Priority**, con un espacio de nombres determinado, la llamada de método query para la consulta tiene el aspecto siguiente:

```
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 añade más de una propiedad de consulta a la consulta, debe numerar las propiedades que añada, tal como se muestra en el fragmento de código. No obstante, la consulta de propiedades personalizadas afecta al rendimiento; el rendimiento disminuye a mayor número de propiedades personalizadas en la consulta.

Ejemplo: inclusión de propiedades personalizadas en una consulta:

Este ejemplo muestra cómo utilizar el método query para recuperar tareas que tienen propiedades personalizadas.

Por ejemplo, si desea buscar todas las tareas de usuario en estado preparado que tienen una propiedad personalizada, **customerID**, con el valor `CID_12345`. El fragmento de código siguiente muestra la llamda al método query para la consulta:

```
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 ahora desea recuperar las tareas y sus propiedades personalizadas, la llamada de método query para la consulta tiene el aspecto siguiente:

```
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 );
```

La sentencia SQL que se genera a partir de esta consulta de API se muestra en el siguiente fragmento de código:

```
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 = 'JuanTorres' OR WI.OWNER_ID IS NULL AND WI.EVERYBODY = 1 )
```

Esta declaración SQL contiene una unión exterior entre la vista TASK y la vista TASK_CPROP. Esto significa que las tareas que satisfacen la cláusula WHERE se recuperan incluso si no tienen ninguna propiedad personalizada.

Vistas predefinidas para las consultas sobre los objetos de proceso de empresa y tarea de usuario:

Se proporcionan vistas de bases de datos predefinidas para objetos de proceso de empresa y de tarea de usuario. Utilice estas vistas cuando consulte datos de referencia para dichos objetos.

Cuando utilice estas vistas predefinidas, no es necesario que añada predicados de unión para columnas de vista de forma explícita, estas construcciones se añaden automáticamente. Puede utilizar la función de consulta genérica de la API de servicio (BusinessFlowManagerService o HumanTaskManagerService) para consultar estos datos. También puede utilizar el método correspondiente de la API de HumanTaskManagerDelegate o las consultas predefinidas proporcionadas por las implementaciones de la interfaz ExecutableQuery.

Nota: Las vistas podrían contener columnas que no se describen. Estas columnas sólo son para uso interno.

Vista ACTIVITY:

Utilice esta vista de base de datos predefinida para realizar consultas sobre actividades.

Tabla 3. Columnas de la vista ACTIVITY

Nombre de columna	Tipo	Comentarios
PIID	ID	ID de la instancia de proceso.
AIID	ID	ID de la instancia de actividad.
PTID	ID	ID de plantilla de proceso.
ATID	ID	ID de plantilla de actividad.

Tabla 3. Columnas de la vista ACTIVITY (continuación)

Nombre de columna	Tipo	Comentarios
KIND	Integer	El tipo de actividad. Los valores posibles son: 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	Timestamp	La hora a la que se completó.
ACTIVATED	Timestamp	La hora a la que se activó (la actividad).
FIRST_ACTIVATED	Timestamp	La hora a la que se ha activado la actividad por primera vez.
STARTED	Timestamp	La hora a la que se arrancó.
STATE	Integer	El estado de la actividad. Los valores posibles son: 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	String	ID principal del propietario.
DESCRIPTION	String	Si la descripción de la plantilla de actividad contiene contenedores, esta columna contiene la descripción de la instancia de la actividad con los contenedores resueltos.

Tabla 3. Columnas de la vista ACTIVITY (continuación)

Nombre de columna	Tipo	Comentarios
TEMPLATE_NAME	String	Nombre de la plantilla de actividad asociada.
TEMPLATE_DESCR	String	Descripción de la plantilla de actividad asociada.
BUSINESS_RELEVANCE	Boolean	Especifica si la actividad tiene relevancia empresarial. Los valores posibles son: **TRUE** La actividad está asociada a la empresa. Puede ver el estado de la actividad en Business Process Choreographer Explorer. **FALSE** La actividad no está asociada a la empresa.
EXPIRES	Timestamp	La fecha y la hora en la que debe caducar la actividad. Si la actividad ha caducado, la fecha y hora en la que se ha producido este suceso.

Vista ACTIVITY_ATTRIBUTE:

Utilice esta vista de base de datos predefinida para realizar consultas sobre propiedades personalizadas para actividades.

Tabla 4. Columnas de la vista ACTIVITY_ATTRIBUTE

Nombre de columna	Tipo	Comentarios
AIID	ID	ID de la instancia de actividad que tiene una propiedad personalizada.
NAME	String	Nombre de la propiedad personalizada.
VALUE	String	Valor de la propiedad personalizada.

Vista ACTIVITY_SERVICE:

Utilice esta vista de base de datos predefinida para realizar consultas sobre servicios de actividades.

Tabla 5. Columnas de la vista ACTIVITY_SERVICE

Nombre de columna	Tipo	Comentarios
EIID	ID	ID de la instancia de suceso.
AIID	ID	El ID de la instancia de actividad que está esperando el suceso.
PIID	ID	ID de la instancia de proceso que contiene el suceso.
VTID	ID	ID de la plantilla de servicio que describe el suceso.
PORT_TYPE	String	El nombre del tipo de puerto.

Tabla 5. Columnas de la vista ACTIVITY_SERVICE (continuación)

Nombre de columna	Tipo	Comentarios
NAME_SPACE_URI	String	El URI del espacio de nombres.
OPERATION	String	El nombre de la operación del servicio.

Vista APPLICATION_COMP:

Utilice esta vista de base de datos predefinida para consultar los valores por omisión y de ID de componente de aplicación para tareas.

Tabla 6. Columnas de la vista APPLICATION_COMP

Nombre de columna	Tipo	Comentarios
ACOID	String	ID del componente de aplicación.
BUSINESS_ RELEVANCE	Boolean	La política de relevancia comercial de tareas por omisión del componente. Este valor puede sobrescribirse con una definición de la plantilla de tarea o la tarea. El atributo afecta a la anotación cronológica del seguimiento de supervisión. Los valores posibles son: **TRUE** La tarea es relevante para la empresa y se realiza una auditoría. **FALSE** La tarea no es relevante para la empresa y no se realizará una auditoría.
NAME	String	Nombre del componente de aplicación.
SUPPORT_ AUTOCLAIM	Boolean	Política de reclamación automática por omisión del componente. Si este atributo se establece en TRUE, la tarea puede reclamarse automáticamente si un usuario es el propietario potencial. Este valor puede sobrescribirse con una definición de la plantilla de tarea o tarea.
SUPPORT_CLAIM_ SUSP	Boolean	El valor por omisión del componente que determina si pueden reclamarse tareas suspendidas. Si este atributo se establece en TRUE, pueden reclamarse tareas suspendidas. Este valor puede sobrescribirse con una definición de la plantilla de tarea o la tarea.
SUPPORT_ DELEGATION	Boolean	La política de delegación de tareas por omisión del componente. Si este atributo se establece en TRUE, pueden modificarse las asignaciones de elementos de trabajo para la tarea. Esto quiere decir que los elementos de trabajo pueden crearse, suprimirse o transferirse.
SUPPORT_ FOLLOW_ON	Boolean	La política de tarea de continuación por omisión del componente. Si se establece este atributo en TRUE, se pueden crear tareas de continuación para las tareas. Este valor puede sobrescribirse con una definición de la plantilla de tarea o la tarea.

Tabla 6. Columnas de la vista APPLICATION_COMP (continuación)

Nombre de columna	Tipo	Comentarios
SUPPORT_ SUB_TASK	Boolean	La política de subtarea por omisión del componente. Si este atributo está establecido en TRUE, se pueden crear subtareas de las tareas. Este valor puede sobrescribirse con una definición de la plantilla de tarea o la tarea.

Vista ESCALATION:

Utilice esta vista de base de datos predefinida para consultar datos para escaladas.

Tabla 7. Columnas de la vista ESCALATION

Nombre de columna	Tipo	Comentarios
ESIID	String	ID de la instancia de escalada.
ACTION	Integer	La acción desencadenada por la escalada. Los valores posibles son: **ACTION_CREATE_WORK_ITEM (1)** Crea un elemento de trabajo para cada receptor de escalada. **ACTION_SEND_EMAIL (2)** Envía un correo electrónico a cada receptor de escalada. **ACTION_CREATE_EVENT (3)** Crea y publica un suceso.
ACTIVATION_STATE	Integer	Se crea una instancia de escalada si la tarea correspondiente alcanza uno de los estados siguientes: **ACTIVATION_STATE_READY (2)** Especifica que la tarea de usuario o participativa está lista para reclamarse. **ACTIVATION_STATE_RUNNING (3)** Especifica que la tarea originaria se ha iniciado y está ejecutándose. **ACTIVATION_STATE_CLAIMED (8)** Especifica que la tarea se ha reclamado. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** Especifica que la tarea espera a que se completen las subtareas.
ACTIVATION_TIME	Timestamp	La ahora a la que se activó la escalada.

Tabla 7. Columnas de la vista ESCALATION (continuación)

Nombre de columna	Tipo	Comentarios
AT_LEAST_ EXP_STATE	Integer	Estado de la tarea esperado por la escalada. Si se produce un tiempo de espera excedido, el estado de tarea se compara con el valor de este atributo. Los valores posibles son: **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** Especifica que la tarea se ha reclamado. **AT_LEAST_EXPECTED_STATE_ENDED (20)** Especifica que la tarea está en un estado final (FINISHED, FAILED, TERMINATED o EXPIRED). **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** Especifica que todas las subtareas de la tarea se han completado.
ESTID	String	ID de la plantilla de escalada correspondiente.
FIRST_ESIID	String	ID de la primera escalada de la cadena.
INCREASE_PRIORITY	Integer	Indica cómo se aumentará la prioridad de la tarea. Los valores posibles son: **INCREASE_PRIORITY_NO (1)** No se aumenta la prioridad de tarea. **INCREASE_PRIORITY_ONCE (2)** La prioridad de tarea se aumenta una vez en uno. **INCREASE_PRIORITY_REPEATED (3)** La prioridad de tarea se aumenta en uno cada vez que se repite la escalada.
NAME	String	Nombre de la escalada.
STATE	Integer	Estado de la escalada. Los valores posibles son: STATE_INACTIVE (1) STATE_WAITING (2) STATE_ESCALATED (3) STATE_SUPERFLUOUS (4)
TKIID	String	ID de instancia de tarea al que pertenece la escalada.

Vista ESCALATION_CPROP:

Utilice esta vista de base de datos predefinida para consultar las propiedades personalizadas para escaladas.

Tabla 8. Columnas de la vista ESCALATION_CPROP

Nombre de columna	Tipo	Comentarios
ESIID	String	ID de escalada.
NAME	String	Nombre de la propiedad.
DATA_TYPE	String	El tipo de la clase para las propiedades personalizadas no serie.

Tabla 8. Columnas de la vista ESCALATION_CPROP (continuación)

Nombre de columna	Tipo	Comentarios
STRING_VALUE	String	Valor de las propiedades personalizadas de tipo String.

Vista ESCALATION_DESC:

Utilice esta vista de base de datos predefinida para consultar datos descriptivos multilingües para escaladas.

Tabla 9. Columnas de la vista ESCALATION_DESC

Nombre de columna	Tipo	Comentarios
ESIID	String	ID de escalada.
LOCALE	String	Nombre del entorno local asociado con la descripción o el nombre de pantalla.
DESCRIPTION	String	Descripción de la plantilla de tarea.
DISPLAY_NAME	String	Nombre descriptivo de la escalada.

Vista ESC_TEMPL:

Utilice esta vista de base de datos predefinida para consultar datos para plantillas de escalada.

Tabla 10. Columnas de la vista ESC_TEMPL

Nombre de columna	Tipo	Comentarios
ESTID	String	ID de la plantilla de escalada.
ACTION	Integer	Acción desencadenada por la escalada. Los valores posibles son: **ACTION_CREATE_WORK_ITEM (1)** Crea un elemento de trabajo para cada receptor de escalada. **ACTION_SEND_EMAIL (2)** Envía un correo electrónico a cada receptor de escalada. **ACTION_CREATE_EVENT (3)** Crea y publica un suceso.

Tabla 10. Columnas de la vista ESC_TEMPL (continuación)

Nombre de columna	Tipo	Comentarios
ACTIVATION_STATE	Integer	Se crea una instancia de escalada si la tarea correspondiente alcanza uno de los estados siguientes: **ACTIVATION_STATE_READY (2)** Especifica que la tarea de usuario o participativa está lista para reclamarse. **ACTIVATION_STATE_RUNNING (3)** Especifica que la tarea originaria se ha iniciado y está ejecutándose. **ACTIVATION_STATE_CLAIMED (8)** Especifica que la tarea se ha reclamado. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** Especifica que la tarea espera a que se completen las subtareas.
AT_LEAST_ EXP_STATE	Integer	Estado de la tarea esperado por la escalada. Si se produce un tiempo de espera excedido, el estado de tarea se compara con el valor de este atributo. Los valores posibles son: **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** Especifica que la tarea se ha reclamado. **AT_LEAST_EXPECTED_STATE_ENDED (20)** Especifica que la tarea está en un estado final (FINISHED, FAILED, TERMINATED o EXPIRED). **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** Especifica que todas las subtareas de la tarea se han completado.
CONTAINMENT_ CTX_ID	String	Si la plantilla de escalada pertenece a una plantilla de tarea en línea, el contexto de contenedor es la plantilla de proceso. Si el contexto de la plantilla de escalada pertenece a una plantilla de tarea autónoma, el contexto de contenedor es la plantilla de tarea.
FIRST_ESTID	String	ID de la primera plantilla de escalada en una cadena de plantilla de escalada.
INCREASE_PRIORITY	Integer	Indica cómo se aumentará la prioridad de la tarea. Los valores posibles son: **INCREASE_PRIORITY_NO (1)** No se aumenta la prioridad de tarea. **INCREASE_PRIORITY_ONCE (2)** La prioridad de tarea se aumenta una vez en uno. **INCREASE_PRIORITY_REPEATED (3)** La prioridad de tarea se aumenta en uno cada vez que se repite la escalada.
NAME	String	Nombre de la plantilla de escalada.

Tabla 10. Columnas de la vista ESC_TEMPL (continuación)

Nombre de columna	Tipo	Comentarios
PREVIOUS_ESTID	String	ID de la plantilla de escalada anterior en una cadena de plantilla de escalada.
TKTID	String	ID de plantilla de tarea al que pertenece la plantilla de escalada.

Vista ESC_TEMPL_CPROP:

Utilice esta vista de base de datos predefinida para consultar propiedades personalizadas de consulta para plantillas de escalada.

Tabla 11. Columnas de la vista ESC_TEMPL_CPROP

Nombre de columna	Tipo	Comentarios
ESTID	String	ID de la plantilla de escalada.
NAME	String	Nombre de la propiedad.
TKTID	String	ID de plantilla de tarea al que pertenece la plantilla de escalada.
DATA_TYPE	String	Tipo de la clase para las propiedades personalizadas no serie.
VALUE	String	Valor de las propiedades personalizadas de tipo String.

Vista ESC_TEMPL_DESC:

Utilice esta vista de base de datos predefinida para consultar datos descriptivos multilingües para plantillas de escalada.

Tabla 12. Columnas de la vista ESC_TEMPL_DESC

Nombre de columna	Tipo	Comentarios
ESTID	String	ID de la plantilla de escalada.
LOCALE	String	Nombre del entorno local asociado con la descripción o el nombre de pantalla.
TKTID	String	ID de plantilla de tarea al que pertenece la plantilla de escalada.
DESCRIPTION	String	Descripción de la plantilla de tarea.
DISPLAY_NAME	String	Nombre descriptivo de la escalada.

Vista PROCESS_ATTRIBUTE:

Utilice esta vista de base de datos predefinida para realizar consultas sobre propiedades personalizadas para procesos.

Tabla 13. Columnas de la vista PROCESS_ATTRIBUTE

Nombre de columna	Tipo	Comentarios
PIID	ID	ID de la instancia de proceso que tiene una propiedad personalizada.
NAME	String	Nombre de la propiedad personalizada.

Tabla 13. Columnas de la vista PROCESS_ATTRIBUTE (continuación)

Nombre de columna	Tipo	Comentarios
VALUE	String	Valor de la propiedad personalizada.

Vista PROCESS_INSTANCE:

Utilice esta vista de base de datos predefinida para realizar consultas sobre instancias de procesos.

Tabla 14. Columnas de la vista PROCESS_INSTANCE

Nombre de columna	Tipo	Comentarios
PTID	ID	ID de plantilla de proceso.
PIID	ID	ID de la instancia de proceso.
NAME	String	Nombre de la instancia del proceso.
STATE	Integer	Estado de la instancia de proceso. Los valores posibles son: 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	Timestamp	Hora en la que se ha creado la instancia de proceso.
STARTED	Timestamp	Hora en la que se ha iniciado la instancia de proceso.
COMPLETED	Timestamp	Hora en la que se ha completado la instancia de proceso.
PARENT_PIID	ID	ID de la instancia del proceso padre.
PARENT_NAME	String	Nombre de la instancia del proceso padre.
TOP_LEVEL_PIID	ID	ID de la instancia del proceso de alto nivel. Si no hay ninguna instancia de proceso de nivel superior, es el ID de la instancia de proceso actual.
TOP_LEVEL_NAME	String	Nombre de la instancia del proceso de alto nivel. Si no hay ninguna instancia de proceso de nivel superior, es el nombre de la instancia de proceso actual.
STARTER	String	ID de principal del usuario que ha iniciado la instancia de proceso.
DESCRIPTION	String	Si la descripción de la plantilla de proceso contiene contenedores, esta columna contiene la descripción de la instancia de proceso con los contenedores resueltos.
TEMPLATE_NAME	String	Nombre de la plantilla de proceso asociada.

Tabla 14. Columnas de la vista PROCESS_INSTANCE (continuación)

Nombre de columna	Tipo	Comentarios
TEMPLATE_DESCR	String	Descripción de la plantilla de proceso asociada.
RESUMES	Timestamp	Hora a la que debe reanudarse automáticamente la instancia de proceso.

Vista PROCESS_TEMPLATE:

Utilice esta vista de base de datos predefinida para realizar consultas sobre plantillas de procesos.

Tabla 15. Columnas de la vista PROCESS_TEMPLATE

Nombre de columna	Tipo	Comentarios
PTID	ID	ID de plantilla de proceso.
NAME	String	El nombre de la plantilla de proceso.
VALID_FROM	Timestamp	Hora a partir de la que se puede crear una instancia de plantilla de flujo.
TARGET_NAMESPACE	String	Espacio de nombres de la plantilla de proceso.
APPLICATION_NAME	String	El nombre de la aplicación de empresa a la que pertenece la plantilla de proceso.
VERSION	String	Versión definida por el usuario.
CREATED	Timestamp	Hora a la que se ha creado la plantilla de proceso en la base de datos.
STATE	Integer	Especifica si la plantilla de proceso está disponible para crear instancias de proceso. Los valores posibles son: STATE_STARTED (1) STATE_STOPPED (2)
EXECUTION_MODE	Integer	Especifica cómo se pueden ejecutar las instancias de proceso derivadas de esta plantilla de proceso. Los valores posibles son: EXECUTION_MODE_MICROFLOW (1) EXECUTION_MODE_LONG_RUNNING (2)
DESCRIPTION	String	Descripción de la plantilla de proceso.
COMP_SPHERE	Integer	Especifica el comportamiento de compensación de las instancias de microflujos de la plantilla de proceso; se une una esfera de compensación existente o se crea una esfera de compensación. Los valores posibles son: COMP_SPHERE_REQUIRED (2) COMP_SPHERE_SUPPORTS (4)
DISPLAY_NAME	String	Nombre descriptivo del proceso.

Vista QUERY_PROPERTY:

Utilice esta vista de base de datos predefinida para las consultas de variables a nivel de proceso.

Tabla 16. Columnas de la vista QUERY_PROPERTY

Nombre de columna	Tipo	Comentarios
PIID	ID	ID de la instancia de proceso.
VARIABLE_NAME	String	Nombre de la variable a nivel de proceso.
NAME	String	Nombre de la propiedad de consulta.
NAMESPACE	String	Espacio de nombres de la propiedad de consulta.
GENERIC_VALUE	String	Una representación de serie para los tipos de propiedad que no se conrrelacionan con uno de los tipos definidos: STRING_VALUE, NUMBER_VALUE, DECIMAL_VALUE o TIMESTAMP_VALUE.
STRING_VALUE	String	Si se correlaciona un tipo de propiedad con un tipo String, este es el valor de la serie.
NUMBER_VALUE	Integer	Si se correlaciona un tipo de propiedad con un tipo Integer, este es el valor del entero.
DECIMAL_VALUE	Decimal	Si se correlaciona un tipo de propiedad con un tipo Floating Point, este es el valor del decimal.
TIMESTAMP_VALUE	Timestamp	Si se correlaciona un tipo de propiedad con un tipo Timestamp, este es el valor de la indicación de fecha y hora.

Vista TASK:

Utilice esta vista de base de datos predefinida para realizar consultas en objetos de tarea.

Tabla 17. Columnas de la vista TAREA

Nombre de columna	Tipo	Comentarios
TKIID	ID	ID de la instancia de tarea.
ACTIVATED	Timestamp	Hora a la que se activó la tarea.
APPLIC_ DEFAULTS_ID	ID	ID del componente de aplicación que especifica los valores por omisión de la tarea.
APPLIC_NAME	String	Nombre de la aplicación de empresa a la que pertenece la tarea.
BUSINESS_ RELEVANCE	Boolean	Especifica si la tarea tiene relevancia empresarial. El atributo afecta a la anotación cronológica del seguimiento de supervisión. Los valores posibles son: **TRUE** La tarea es relevante para la empresa y se realiza una auditoría. **FALSE** La tarea no es relevante para la empresa y no se realizará una auditoría.

Tabla 17. Columnas de la vista TAREA (continuación)

Nombre de columna	Tipo	Comentarios
COMPLETED	Timestamp	Hora a la que se completó la tarea.
CONTAINMENT_ CTX_ID	ID	Contexto de contenedor para esta tarea. Este atributo determina el ciclo de vida de la tarea. Cuando se suprime el contexto de contenedor de una tarea, también se suprime la tarea.
CTX_ AUTHORIZATION	Integer	Permite que el propietario de la tarea acceda al contexto de la tarea. Los valores posibles son: **AUTH_NONE** No se tienen derechos de autorización sobre el objeto de contexto asociado. **AUTH_READER** Las operaciones en el objeto de contexto asociado requieren autorización de lector como, por ejemplo, leer las propiedades de una instancia de proceso.
DUE	Timestamp	Hora a la que debe finalizarse la tarea.
EXPIRES	Timestamp	Fecha en la que caduca la tarea.
FIRST_ACTIVATED	Timestamp	Hora a la que se activó la tarea por primera vez.
FOLLOW_ON_TKIID	ID	El identificador de la instancia de la tarea de continuación.
HIERARCHY_ POSITION	Integer	Los valores posibles son: **HIERARCHY_POSITION_TOP_TASK (0)** La tarea de nivel superior de la jerarquía de tareas. **HIERARCHY_POSITION_SUB_TASK (1)** La tarea es una subtarea de la jerarquía de tareas. **HIERARCHY_POSITION_FOLLOW_ON_TASK (2)** La tarea es una tarea de continuación de la jerarquía de tareas.
IS_AD_HOC	Boolean	Indica si esta tarea se ha creado dinámicamente durante la ejecución o a partir de una plantilla de tarea.
IS_ESCALATED	Boolean	Indica si se ha producido una escalada de esta tarea.
IS_INLINE	Boolean	Indica si la tarea es una tarea en línea de un proceso de empresa.
IS_WAIT_FOR_ SUB_TK	Boolean	Indica si la tarea padre está a la espera de una subtarea para alcanzar un estado final.

Tabla 17. Columnas de la vista TAREA (continuación)

Nombre de columna	Tipo	Comentarios
KIND	Integer	Clase de tarea. Los valores posibles son: **KIND_HUMAN (101)** Indica que la tarea es una *tarea de colaboración* que un humano crea y procesa. **KIND_WPC_STAFF_ACTIVITY (102)** Indica que la tarea es una tarea de usuario que es una actividad de personal de un proceso de empresa de WebSphere Business Integration Server Foundation, versión 5. **KIND_ORIGINATING (103)** Indica que la tarea es una *tarea de invocación* que da soporte a la interacción de personas con equipos, lo que permite a un usuario crear, inicializar e iniciar servicios. **KIND_PARTICIPATING (105)** Indica que la tarea es una *tarea a realizar* da soporte a las interacciones de equipos con personas, lo que permite que una persona implemente un servicio. **KIND_ADMINISTRATIVE (106)** Indica que la tarea es una tarea de administración.
LAST_MODIFIED	Timestamp	Hora a la que se modificó la tarea por última vez.
LAST_STATE_ CHANGE	Timestamp	Hora a la que se modificó el estado de la tarea por última vez.
NAME	String	Nombre de la tarea.
NAME_SPACE	String	Espacio de nombres que se utiliza para categorizar la tarea.
ORIGINATOR	String	ID del principal del originador de la tarea.
OWNER	String	ID del principal del propietario de la tarea.
PARENT_ CONTEXT_ID	String	Contexto padre de esta tarea. Este atributo proporciona una clave para el contexto correspondiente en el componente de aplicación llamante. El contexto padre lo establece el componente de aplicación que crea la tarea.
PRIORITY	Integer	Prioridad de la tarea.
RESUMES	Timestamp	Hora a la que debe reanudarse automáticamente la tarea.
STARTED	Timestamp	Hora a la que se inició la tarea (STATE_RUNNING, STATE_CLAIMED).
STARTER	String	ID del principal del iniciador de la tarea.

Tabla 17. Columnas de la vista TAREA (continuación)

Nombre de columna	Tipo	Comentarios
STATE	Integer	Estado de la tarea. Los valores posibles son: **STATE_READY (2)** Indica que la tarea está lista para reclamarse. **STATE_RUNNING (3)** Indica que la tarea se ha iniciado y está ejecutándose. **STATE_FINISHED (5)** Indica que la tarea ha finalizado satisfactoriamente. **STATE_FAILED (6)** Indica que la tarea no ha finalizado satisfactoriamente. **STATE_TERMINATED (7)** Indica que la tarea se ha interrumpido a causa de una solicitud externa o interna. **STATE_CLAIMED (8)** Indica que la tarea se ha reclamado. **STATE_EXPIRED (12)** Indica que la tarea ha finalizado porque se ha agotado su duración especificada. **STATE_FORWARDED (101)** Indica que la tarea se ha completado con una tarea de continuación.
SUPPORT_ AUTOCLAIM	Boolean	Indica si esta tarea se reclama automáticamente si se asigna a un solo usuario.
SUPPORT_CLAIM_ SUSP	Boolean	Indica si esta tarea puede reclamarse si es suspendida.
SUPPORT_ DELEGATION	Boolean	Indica si esta tarea da soporte a la delegación de trabajo mediante la creación, supresión o transferencia de elementos de trabajo.
SUPPORT_ FOLLOW_ON	Boolean	Indica si esta tarea permite crear tareas de continuación.
SUPPORT_SUB_TASK	Boolean	Indica si esta tarea admite la creación de subtareas.
SUSPENDED	Boolean	Indica si la tarea se ha suspendido.
TKTID	ID	ID de plantilla de tarea.
TOP_TKIID	ID	El identificador de instancia de tarea padre si ésta es una subtarea.
TYPE	String	Tipo utilizado para categorizar la tarea.

Vista TASK_CPROP:

Utilice esta vista de base de datos predefinida para consultar propiedades de consulta para objetos de tarea.

Tabla 18. Columnas de la vista TASK_CPROP

Nombre de columna	Tipo	Comentarios
TKIID	String	ID de la instancia de tarea.
NAME	String	Nombre de la propiedad.
DATA_TYPE	String	El tipo de la clase para las propiedades personalizadas no serie.
STRING_VALUE	String	Valor de las propiedades personalizadas de tipo String.

Vista TASK_DESC:

Utilice esta vista de base de datos predefinida para consultar datos descriptivos multilingües para objetos de tarea.

Tabla 19. Columna de la vista TASK_DESC

Nombre de columna	Tipo	Comentarios
TKIID	String	ID de la instancia de tarea.
LOCALE	String	Nombre del entorno local asociado con la descripción o el nombre de pantalla.
DESCRIPTION	String	Descripción de la tarea.
DISPLAY_NAME	String	Nombre descriptivo de la tarea.

Vista TASK_TEMPL:

Esta vista de base de datos predefinida contiene datos que puede utilizar para crear instancias de tareas.

Tabla 20. Columnas de la vista TASK_TEMPL

Nombre de columna	Tipo	Comentarios
TKTID	String	ID de plantilla de tarea.
VALID_FROM	Timestamp	Hora en que la plantilla de tarea queda disponible para la creación de instancias.
APPLIC_ DEFAULTS_ID	String	ID del componente de aplicación que especifica los valores por omisión de la plantilla de tarea.
APPLIC_NAME	String	Nombre de la aplicación de empresa a la que pertenece la plantilla de tarea.
BUSINESS_ RELEVANCE	Boolean	Especifica si la plantilla de tarea tiene relevancia empresarial. El atributo afecta a la anotación cronológica del seguimiento de supervisión. Los valores posibles son: **TRUE** La tarea es relevante para la empresa y se realiza una auditoría. **FALSE** La tarea no es relevante para la empresa y no se realizará una auditoría.
CONTAINMENT_ CTX_ID	ID	Contexto de contenedor para esta plantilla de tarea. Este atributo determina el ciclo de vida de la plantilla de tarea. Cuando se suprime un contexto de contenedor, también se suprime la plantilla de tarea.

Tabla 20. Columnas de la vista TASK_TEMPL (continuación)

Nombre de columna	Tipo	Comentarios
CTX_ AUTHORIZATION	Integer	Permite que el propietario de la tarea acceda al contexto de la tarea. Los valores posibles son: **AUTH_NONE** No se tienen derechos de autorización sobre el objeto de contexto asociado. **AUTH_READER** Las operaciones en el objeto de contexto asociado requieren autorización de lector como, por ejemplo, leer las propiedades de una instancia de proceso.
DEFINITION_NAME	String	Nombre de la definición de plantilla de tarea en el archivo TEL (Task Execution Language).
DEFINITION_NS	String	Espacio de nombres de la definición de plantilla de tarea en el archivo TEL.
IS_AD_HOC	Boolean	Indica si esta plantilla de tarea se ha creado dinámicamente durante la ejecución o cuando se ha desplegado la tarea como parte de un archivo EAR.
IS_INLINE	Boolean	Indica si esta plantilla de tarea está modelada como una tarea en un proceso de empresa.
KIND	Integer	Clase de tareas que se derivan de esta plantilla de tarea. Los valores posibles son: **KIND_HUMAN (101)** Indica que la tarea es una *tarea de colaboración* que un humano crea y procesa. **KIND_ORIGINATING (103)** Indica que la tarea es una *tarea de invocación* que da soporte a la interacción de personas con equipos, lo que permite a un usuario crear, inicializar e iniciar servicios. **KIND_PARTICIPATING (105)** Indica que la tarea es una *tarea a realizar* da soporte a las interacciones de equipos con personas, lo que permite que una persona implemente un servicio. **KIND_ADMINISTRATIVE (106)** Indica que la tarea es una tarea de administración.
NAME	String	El nombre de la plantilla de tarea.
NAMESPACE	String	Espacio de nombres que se utiliza para categorizar la plantilla de tarea.
PRIORITY	Integer	Prioridad de la plantilla de tarea.

Tabla 20. Columnas de la vista TASK_TEMPL (continuación)

Nombre de columna	Tipo	Comentarios
STATE	Integer	Estado de la plantilla de tarea. Los valores posibles son: **STATE_STARTED (1)** Especifica que la plantilla de tarea está disponible para crear instancias de tareas. **STATE_STOPPED (2)** Especifica que la plantilla de tarea se ha detenido. En este estado, las instancias de tareas no pueden crearse a partir de la plantilla de tarea.
SUPPORT_ AUTOCLAIM	Boolean	Indica si las tareas derivadas de esta plantilla de tarea pueden reclamarse automáticamente si se asignan a un solo usuario.
SUPPORT_CLAIM_ SUSP	Boolean	Indica si las tareas derivadas de esta plantilla de tarea pueden reclamarse si se suspenden.
SUPPORT_ DELEGATION	Boolean	Indica si las tareas derivadas de esta plantilla de tarea dan soporte a la delegación de trabajo mediante la creación, supresión o transferencia de elementos de trabajo.
SUPPORT_ FOLLOW_ON	Boolean	Indica si la plantilla de tarea permite crear las tareas de continuación.
SUPPORT_SUB_TASK	Boolean	Indica si la plantilla de tarea da soporte a la creación de subtareas.
TYPE	String	Tipo utilizado para categorizar la plantilla de tarea.

Vista TASK_TEMPL_CPROP:

Utilice esta vista de base de datos predefinida para consultar propiedades personalizadas de consulta para plantillas de tarea.

Tabla 21. Columnas de la vista TASK_TEMPL_CPROP

Nombre de columna	Tipo	Comentarios
TKTID	String	ID de plantilla de tarea.
NAME	String	Nombre de la propiedad.
DATA_TYPE	String	El tipo de la clase para las propiedades personalizadas no serie.
STRING_VALUE	String	Valor de las propiedades personalizadas de tipo String.

Vista TASK_TEMPL_DESC:

Utilice esta vista de base de datos predefinida para consultar datos descriptivos multilingües para objetos de plantilla de tarea.

Tabla 22. Columnas de la vista TASK_TEMPL_DESC

Nombre de columna	Tipo	Comentarios
TKTID	String	ID de plantilla de tarea.

Tabla 22. Columnas de la vista TASK_TEMPL_DESC (continuación)

Nombre de columna	Tipo	Comentarios
LOCALE	String	Nombre del entorno local asociado con la descripción o el nombre de pantalla.
DESCRIPTION	String	Descripción de la plantilla de tarea.
DISPLAY_NAME	String	Nombre descriptivo de la plantilla de tarea.

Vista WORK_ITEM:

Utilice esta vista de base de datos predefinida para realizar consultas sobre elementos de trabajo y datos de autorización para procesos, tareas y escaladas.

Tabla 23. Columnas de la vista WORK_ITEM

Nombre de columna	Tipo	Comentarios
WIID	ID	ID del elemento de trabajo.
OWNER_ID	String	ID del principal del propietario.
GROUP_NAME	String	Nombre de la lista de trabajos de grupos asociada.
EVERYBODY	Boolean	Especifica si todos son propietarios de este elemento de trabajo.
OBJECT_TYPE	Integer	El tipo de objeto asociado. Los valores posibles son: **OBJECT_TYPE_ACTIVITY (1)** Especifica que el elemento de trabajo se ha creado para una actividad. **OBJECT_TYPE_PROCESS_INSTANCE (3)** Especifica que el elemento de trabajo se ha creado para una instancia de proceso. **OBJECT_TYPE_TASK_INSTANCE (5)** Especifica que el elemento de trabajo se ha creado para una tarea. **OBJECT_TYPE_TASK_TEMPLATE (6)** Especifica que el elemento de trabajo se ha creado para una plantilla de tarea. **OBJECT_TYPE_ESCALATION_ INSTANCE (7)** Especifica que el elemento de trabajo se ha creado para una instancia de escalada. **OBJECT_TYPE_APPLICATION_ COMPONENT (9)** Especifica que el elemento de trabajo se ha creado para un componente de aplicación.
OBJECT_ID	ID	ID del objeto asociado, por ejemplo, el proceso o tarea asociado.

Tabla 23. Columnas de la vista WORK_ITEM (continuación)

Nombre de columna	Tipo	Comentarios
ASSOC_OBJECT_TYPE	Integer	Tipo del objeto referenciado por el atributo ASSOC_OID, por ejemplo, tarea, proceso u objetos externos. Utilice los valores para el atributo OBJECT_TYPE.
ASSOC_OID	ID	ID del objeto asociado con el elemento de trabajo. Por ejemplo, el ID (PIID) de la instancia de proceso que contiene la instancia de la actividad para la que se ha creado este elemento de trabajo.
REASON	Integer	El motivo de la asignación del elemento de trabajo. Los valores posibles son: 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	Timestamp	Fecha y hora cuando se creó el elemento de trabajo.

Filtro de los datos con variables en las consultas

El resultado de una consulta devuelve los objetos que cumplen los criterios de la consulta. Quizá desee filtrar estos resultados según los valores de las variables.

Acerca de esta tarea

Puede definir las variables que un proceso utiliza durante la ejecución en el modelo de proceso. Para estas variables, puede declarar qué partes se pueden consultar.

Por ejemplo, Juan Torres, llama al número de servicio de su compañía aseguradora para averiguar el progreso de la reclamación del seguro de su coche dañado. El administrador de reclamaciones utiliza el identificador de cliente para encontrar la reclamación.

Procedimiento

1. Opcional: Enumere las propiedades de las variables de un proceso que se pueden consultar.

   Utilice el identificador de plantilla de proceso para identificar el proceso. Puede omitir este paso si sabe qué variables se pueden consultar.

```
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. Enumere las instancias de proceso con variables que cumplen los criterios de filtro.

   Para este proceso, el identificador de cliente se crea como parte de la variable customerClaim que se puede consultar. Por lo tanto, puede utilizar el identificador de cliente para encontrar la reclamación.

```
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 'Torres%'",
      (String)null, (Integer)null,
      (Integer)null, (TimeZone)null );
```

   Esta acción devuelve un conjunto de resultados de consulta que contiene los nombres de instancia de proceso y los valores de los identificadores de clientes cuyos identificadores empiezan con Torres.

Gestión de consultas almacenadas

Las consultas almacenadas proporcionan un modo de guardar las consultas que se ejecutan con frecuencia. La consulta almacenada puede ser una consulta que está disponible para todos los usuarios (consulta pública) o una consulta que pertenece a un usuario específico (consulta privada).

Acerca de esta tarea

Una consulta almacenada es una consulta que se almacena en la base de datos y se identifica con un nombre. Una consulta almacenada privada y pública puede tener el mismo nombre; las consultas privadas almacenadas de distintos propietarios también pueden tener el mismo nombre.

Puede tener consultas almacenadas para objetos de proceso de empresa, objetos de tarea, o una combinación de estos dos tipos de objetos.

Conceptos relacionados

"Parámetros de las consultas almacenadas" en la página 33
Una consulta almacenada es una consulta que se almacena en la base de datos y se identifica con un nombre. Los tuples de calificación se ensamblan dinámicamente cuando se ejecuta la consulta. Para que las consultas almacenadas se puedan volver a utilizar, puede utilizar los parámetros de la definición de consulta que se resuelven durante la ejecución.

Gestión de consultas almacenadas públicas:

Las consultas almacenadas públicas las crea el administrador del sistema. Estas consultas están disponibles para todos los usuarios.

Acerca de esta tarea

Como administrador del sistema, puede crear, ver y suprimir consultas almacenadas públicas. Si no especifica un ID de usuario en la llamada de la API, se presupone que la consulta almacenada es una consulta almacenada pública.

Procedimiento

1. Cree una consulta almacenada pública.

   Por ejemplo, el fragmento de código siguiente crea una consulta almacenada para las instancias de proceso y lo guarda con el nombre CustomerOrdersStartingWithA.

   ```
   process.createStoredQuery("CustomerOrdersStartingWithA",
               "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
               "PROCESS_INSTANCE.NAME LIKE 'A%'",
               "PROCESS_INSTANCE.NAME",
                (Integer)null, (TimeZone)null);
   ```

   El resultado de la consulta almacenado es una lista ordenada de todos los nombres de instancias de proceso que comienzan por la letra A y sus ID (PIID) de instancia de proceso asociados.

2. Ejecutar la consulta definida mediante la consulta almacenada.

   ```
   QueryResultSet result = process.query("CustomerOrdersStartingWithA",
                new Integer(0));
   ```

   Esta acción devolverá los objetos que cumplan los criterios. En este caso, todos los pedidos de cliente que empiezan por A.

3. Enumere los nombres de las consultas almacenadas públicas disponibles.

   En el fragmento de código siguiente se muestra cómo limitar la lista de consultas devueltas a sólo las consultas públicas.

   ```
   String[] storedQuery = process.getStoredQueryNames(StoredQueryData.KIND_PUBLIC);
   ```

4. Opcional: Compruebe la consulta definida por una consulta almacenada específica.

   Una consulta privada almacenada puede tener el mismo nombre que una consulta pública almacenada. Si estos nombres son iguales, se devolverá la consulta almacenada privada. En el fragmento de código siguiente se muestra cómo devolver sólo la consulta pública con el nombre especificado. Si desea ejecutar esta consulta para los objetos basados en tareas, especifique StoredQuery como el tipo de objeto devuelto en lugar 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. Suprima una consulta almacenada pública.

   El siguiente fragmento de código muestra cómo suprimir la consulta almacenada que creó en el paso 1.

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

Gestión de consultas almacenadas privadas de otros usuarios:

Cualquier usuario puede crear consultas privadas. Estas consultas sólo están disponibles para el propietario de la consulta y el administrador del sistema.

Acerca de esta tarea

Como administrador del sistema, puede gestionar las consultas almacenadas privadas que pertenecen a un usuario determinado.

Procedimiento

1. Cree una consulta almacenada privada para el ID de usuario Smith.

Por ejemplo, el fragmento de código siguiente crea una consulta almacenada para instancias de proceso y la guarda con el nombre CustomerOrdersStartingWithA para el ID de usuario 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);
```

El resultado de la consulta almacenado es una lista ordenada de todos los nombres de instancias de proceso que comienzan por la letra A y sus ID (PIID) de instancia de proceso asociados.

2. Ejecutar la consulta definida mediante la consulta almacenada.

```
QueryResultSet result = process.query
                       ("Smith", "CustomerOrdersStartingWithA",
                         (Integer)null, (Integer)null, (List)null);
new Integer(0));
```

Esta acción devolverá los objetos que cumplan los criterios. En este caso, todos los pedidos de cliente que empiezan por A.

3. Obtenga una lista de los nombres de las consultas privadas que pertenecen a un usuario determinado.

Por ejemplo, el fragmento de código siguiente muestra cómo obtener una lista de consultas privadas que pertenecen al usuario Smith.

```
String[] storedQuery = process.getStoredQueryNames("Smith");
```

4. Visualice los detalles de una consulta determinada.

En el fragmento de código siguiente se muestra cómo visualizar los detalles de la consulta CustomerOrdersStartingWithA cuyo propietario es el usuario 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. Suprimir una consulta almacenada privada.

En el fragmento de código siguiente se muestra cómo suprimir una consulta privada cuyo propietario es el usuario Smith.

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

Cómo trabajar con las consultas almacenadas privadas:

Si usted no es el administrador del sistema, puede crear, ejecutar y suprimir sus propias consultas almacenadas privadas. También puede utilizar las consultas almacenadas públicas que el administrador del sistema ha creado.

Procedimiento

1. Cree una consulta almacenada privada.

Por ejemplo, el fragmento de código siguiente crea una consulta almacenada para instancias de proceso y la guarda con un nombre específico. Si no se especifica un ID de usuario, se presupone que la consulta almacenada es una consulta almacenada privada para el usuario que ha iniciado la sesión.

```
process.createStoredQuery("CustomerOrdersStartingWithA",
             "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
             "PROCESS_INSTANCE.NAME LIKE 'A%'",
             "PROCESS_INSTANCE.NAME",
              (Integer)null, (TimeZone)null);
```

Esta consulta devuelve una lista ordenada de todos los nombres de instancias de proceso que comienzan por la letra A y sus ID (PIID) de instancia de proceso asociados.

2. Ejecutar la consulta definida mediante la consulta almacenada.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",
               new Integer(0));
```

   Esta acción devolverá los objetos que cumplan los criterios. En este caso, todos los pedidos de cliente que empiezan por A.

3. Obtenga una lista de los nombres de las consultas almacenadas a las que el usuario que ha iniciado la sesión tiene acceso.

   En el fragmento de código siguiente se muestra cómo obtener las consultas almacenadas públicas y privadas a las que tiene acceso el usuario.

```
String[] storedQuery = process.getStoredQueryNames();
```

4. Visualice los detalles de una consulta determinada.

   En el fragmento de código siguiente se muestra cómo visualizar los detalles de la consulta CustomerOrdersStartingWithA cuyo propietario es el usuario Smith.

```
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. Suprimir una consulta almacenada privada.

   El fragmento de código siguiente muestra cómo se suprime una consulta almacenada privada.

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

Desarrollo de aplicaciones para procesos de empresa

Un proceso de empresa es un conjunto de actividades relacionadas con la empresa que se invocan en una secuencia específica para alcanzar un objetivo de empresa. Se proporcionan ejemplos que muestran cómo se pueden desarrollar aplicaciones para acciones típicas en procesos.

Acerca de esta tarea

Un proceso de empresa puede ser un microflujo o un proceso de larga ejecución:

- Los microflujos son procesos de empresa de corta ejecución que se ejecutan de forma síncrona. Después de muy poco tiempo, el resultado se devuelve al llamante.
- Los procesos interrumpibles de larga ejecución se ejecutan como una secuencia de actividades encadenadas. El uso de determinadas construcciones en un proceso crea interrupciones en el flujo del proceso, por ejemplo, cuando se invoca una tarea de usuario, se invoca un servicio utilizando un enlace síncrono o se utilizan actividades dirigidas por temporizador.

  Generalmente se navega de forma asíncrona por las ramas paralelas del proceso, esto es, las actividades de las ramas paralelas se ejecutan de forma simultánea. Según el tipo y el valor de transacción de la actividad, puede ejecutarse una actividad en su propia transacción.

Roles necesarios para las acciones en instancias de proceso

Acceder a la interfaz BusinessFlowManager no garantiza que el llamante pueda realizar todas las acciones de un proceso. El llamante debe iniciar la sesión en la aplicación cliente con un rol que tenga autorización para realizar la acción.

En la tabla siguiente se muestran las acciones en una instancia de proceso que un rol específico puede realizar.

Acción	Rol principal del llamante		
	Lector	**Iniciador**	**Administrador**
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspender			x
transferWorkItem			x

Roles necesarios para acciones en actividades de procesos de empresa

Acceder a la interfaz BusinessFlowManager no garantiza que el llamante pueda realizar todas las acciones de una actividad. El llamante debe iniciar la sesión en la aplicación cliente con un rol que tenga autorización para realizar la acción.

En la tabla siguiente se muestran las acciones en una instancia de actividad que un rol específico puede realizar.

Acción	Rol principal del llamante				
	Lector	**Editor**	**Propietario potencial**	**Propietario**	**Administrador**
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x
getCustomProperties	x	x	x	x	x
getCustomProperty	x	x	x	x	x
getCustomPropertyNames	x	x	x	x	x
getFaultMessage	x	x	x	x	x
getFaultNames	x	x	x	x	x
getInputMessage	x	x	x	x	x
getOutputMessage	x	x	x	x	x
getVariable	x	x	x	x	x
getVariableNames	x	x	x	x	x
getInputVariableNames	x	x	x	x	x
getOutputVariableNames	x	x	x	x	x
getWorkItems	x	x	x	x	x
setCustomProperty		x		x	x
setFaultMessage		x		x	x
setOutputMessage		x		x	x
setVariable					x
transferWorkItem				x Sólo para propietarios potenciales o administra-dores	x

Gestión del ciclo de vida de un proceso de empresa

Una instancia de proceso pasa a existir cuando se invoca un método de la API de Business Process Choreographer que puede iniciar un proceso. La navegación de la instancia de proceso continúa hasta que todas sus actividades están en un estado final. Se pueden llevar a cabo varias acciones en la instancia de proceso para gestionar su ciclo de vida.

Acerca de esta tarea

Se proporcionan ejemplos que muestran cómo puede desarrollar aplicaciones para las siguientes acciones típicas de ciclo de vida en los procesos.

Inicio de procesos de empresa:

La manera en que se inicia un proceso de empresa depende de si el proceso es un microflujo o un proceso de larga ejecución. El servicio que inicia el proceso también es importante para la manera en que se inicia un proceso; el proceso puede tener un servicio inicial exclusivo o varios servicios iniciales.

Acerca de esta tarea

Se proporcionan ejemplos que muestran cómo puede desarrollar aplicaciones de casos típicos para iniciar microflujos y procesos de larga ejecución.

Ejecución de un microflujo que contiene un servicio de arranque exclusivo:

Se puede iniciar un microflujo mediante una actividad de recepción o de obtención. El servicio de arranque es exclusivo si el microflujo se inicia con una actividad de recepción o cuando la actividad de obtención únicamente tiene una definición onMessage.

Acerca de esta tarea

Si el microflujo implementa una operación de petición y respuesta, es decir, el proceso contiene una respuesta, puede utilizar el método call para ejecutar el proceso pasando el nombre de plantilla de proceso como parámetro en la llamada.

Si el microflujo es una operación unidireccional, utilice el método sendMessage para ejecutar el proceso. Este método no está cubierto en este ejemplo.

Procedimiento

1. Opcional: Liste las plantillas de proceso para encontrar el nombre del proceso que desea ejecutar.

   Este paso es opcional si ya sabe el nombre del proceso.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
         PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas clasificadas que se pueden iniciar mediante el método call.

2. Iniciar el proceso con un mensaje de entrada del tipo adecuado.

   Cuando cree el mensaje, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje.

   ```
   ProcessTemplateData template = processTemplates[0];
   //crear un mensaje sólo para la actividad de recepción inicial
   ClientObjectWrapper input = process.createMessage
                              (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //establecer las series del mensaje, por ejemplo, un nombre de cliente
   ```

```
  myMessage.setString("CustomerName", "Smith");
}

//ejecutar el proceso
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");}
```

Esta acción crea una instancia de la plantilla de proceso, CustomerTemplate, y pasa algunos datos al cliente. La operación sólo devuelve resultados cuando el proceso se ha completado. Se devuelve el resultado del proceso, OrderNo, al proceso que efectúa la llamada.

Ejecución de un microflujo que contiene un servicio de arranque no exclusivo:

Se puede iniciar un microflujo mediante una actividad de recepción o de obtención. El servicio de arranque no es exclusivo si el microflujo se inicia con una actividad de obtención que tenga varias definiciones onMessage.

Acerca de esta tarea

Si el microflujo implementa una operación de petición y respuesta, es decir, el proceso contiene una respuesta, puede utilizar el método call para ejecutar el proceso pasando el ID del servicio inicial en la llamada.

Si el microflujo es una operación unidireccional, utilice el método sendMessage para ejecutar el proceso. Este método no está cubierto en este ejemplo.

Procedimiento

1. Opcional: Liste las plantillas de proceso para encontrar el nombre del proceso que desea ejecutar.

   Este paso es opcional si ya sabe el nombre del proceso.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas clasificadas que se pueden iniciar como microflujos.

2. Determinar el servicio de arranque al que se va a llamar.

   Este ejemplo utiliza la primera plantilla que se encuentra.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
           process.getStartActivities(template.getID());
   ```

3. Iniciar el proceso con un mensaje de entrada del tipo adecuado.

   Cuando cree el mensaje, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje.

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //crear un mensaje para el servicio que se va a llamar
   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();
  //establecer las series del mensaje, por ejemplo, un nombre de cliente
  myMessage.setString("CustomerName", "Smith");
}
//ejecutar el proceso
ClientObjectWrapper output = process.call(activity.getServiceTemplateID(),
                                          activity.getActivityTemplateID(),
                                          input);
//comprobar la salida del proceso, por ejemplo, un número de pedido
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");}
```

Esta acción crea una instancia de la plantilla de proceso, CustomerTemplate, y pasa algunos datos al cliente. La operación sólo devuelve resultados cuando el proceso se ha completado. Se devuelve el resultado del proceso, OrderNo, al proceso que efectúa la llamada.

Inicio de un proceso de larga ejecución que contiene un servicio de arranque exclusivo:

Si el servicio de arranque es exclusivo, puede utilizar el método initiate y pasar el nombre de la plantilla de proceso como un parámetro. Este es el caso, cuando el proceso de larga ejecución se inicia con una sola actividad de recepción o de obtención y cuando la actividad de obtención individual solamente tiene una definición onMessage.

Procedimiento

1. Opcional: Listar las plantillas de proceso para encontrar el nombre del proceso que desea iniciar.

   Este paso es opcional si ya sabe el nombre del proceso.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas clasificadas que se pueden iniciar mediante el método initiate.

2. Iniciar el proceso con un mensaje de entrada del tipo adecuado.

   Cuando cree el mensaje, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje. Si especifica un nombre de instancia de proceso, no debe comenzar con el carácter de subrayado. Si no se ha especificado un nombre de instancia de proceso, se utiliza como nombre el ID de instancia de proceso (PIID) con formato de serie.

   ```
   ProcessTemplateData template = processTemplates[0];
   //crear un mensaje sólo para la actividad de recepción inicial
   ClientObjectWrapper input = process.createMessage
                               (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //establecer las series del mensaje, por ejemplo, un nombre de cliente
   ```

```
  myMessage.setString("CustomerName", "Smith");
}
//iniciar el proceso
PIID piid = process.initiate(template.getName(), "CustomerOrder", input);
```

Esta acción crea una instancia, CustomerOrder, y pasa algunos datos al cliente. Cuando se inicia el proceso, la operación devuelve el ID de objeto de la nueva instancia de proceso al proceso que efectúa la llamada.

Se establece el iniciador de la instancia del proceso en el proceso que efectúa la llamada de la petición. Esta persona recibe un elemento de trabajo para la instancia de proceso. Se determinan los administradores de procesos, los lectores y los editores de la instancia de proceso y reciben elementos de trabajo para dicha instancia de proceso. Se determinan las instancias de las actividades que siguen. Se inician automáticamente o, si son actividades de tareas de usuario, de recepción o de obtención, se crean elementos de trabajo para los posibles propietarios.

Inicio de un proceso de larga ejecución que contiene un servicio de arranque que no es exclusivo:

Un proceso de larga ejecución se puede iniciar mediante varias actividades de recepción o de obtención iniciales. Puede utilizar el método initiate para iniciar el proceso. Si el servicio de inicio no es exclusivo, por ejemplo, si el proceso se inicia con varias actividades de recepción o de obtención, o una actividad de obtencion que tiene varias definiciones onMessage, debe identificar el servicio al que se ha de llamar.

Procedimiento

1. Opcional: Listar las plantillas de proceso para encontrar el nombre del proceso que desea iniciar.

   Este paso es opcional si ya sabe el nombre del proceso.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas clasificadas que se pueden iniciar como procesos de larga ejecución.

2. Determinar el servicio de arranque al que se va a llamar.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
           process.getStartActivities(template.getID());
   ```

3. Iniciar el proceso con un mensaje de entrada del tipo adecuado.

   Cuando cree el mensaje, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje. Si especifica un nombre de instancia de proceso, no debe comenzar con el carácter de subrayado. Si no se ha especificado un nombre de instancia de proceso, se utiliza como nombre el ID de instancia de proceso (PIID) con formato de serie.

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //crear un mensaje para el servicio que se va a llamar
   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();
  //establecer las series del mensaje, por ejemplo, un nombre de cliente
  myMessage.setString("CustomerName", "Smith");
}
//iniciar el proceso
PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                            activity.getActivityTemplateID(),
                            input);
```

Esta acción crea una instancia y pasa algunos datos al cliente. Cuando se inicia el proceso, la operación devuelve el ID de objeto de la nueva instancia de proceso al proceso que efectúa la llamada.

Se establece el iniciador de la instancia del proceso en el proceso que efectúa la llamada de la petición y recibe un elemento de trabajo para la instancia del proceso. Se determinan los administradores de procesos, los lectores y los editores de la instancia de proceso y reciben elementos de trabajo para dicha instancia de proceso. Se determinan las instancias de las actividades que siguen. Se inician automáticamente o, si son actividades de tareas de usuario, de recepción o de obtención, se crean elementos de trabajo para los posibles propietarios.

Suspensión y reanudación de un proceso de empresa:

Puede suspender instancias de proceso de nivel superior de larga ejecución mientras se están ejecutando y reanudarlas de nuevo para completarlas.

Antes de empezar

El llamante debe ser un administrador de la instancia de proceso o un administrador de procesos de empresa. Para suspender una instancia de proceso, debe estar en el estado de ejecución o anómalo.

Acerca de esta tarea

Por ejemplo, quizá desee suspender una instancia de proceso, de manera que pueda configurar el acceso a un sistema de programa de fondo que se utiliza posteriormente en el proceso. Cuando se cumplan los prerrequisitos del proceso, puede reanudar la instancia de proceso. También es posible que desee suspender un proceso para solucionar un problema que está haciendo que la instancia de proceso falle y luego volver a reanudarlo cuando se soluciona el problema.

Procedimiento

1. Obtenga el proceso en ejecución, CustomerOrder, que desea suspender.

   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```

2. Suspenda la instancia de proceso.

   ```
   PIID piid = processInstance.getID();
   process.suspend( piid );
   ```

   Esta acción suspende la instancia de proceso de nivel superior especificada. La instancia de proceso se pone en estado suspendido. Los subprocesos con el atributo autonomy establecido en el valor `child` también se suspenden si están en el estado de en ejecución, anómalo, terminando o compensándose. También se suspenden las tareas en línea asociadas a esta instancia de proceso, pero no se suspenden las tareas autónomas asociadas a esta instancia de proceso.

En este estado, las actividades que se inician pueden finalizase pero no se activan actividades nuevas, por ejemplo, se puede completar una actividad de tareas de usuario en estado reclamado.

3. Reanude la instancia de proceso.

   ```
   process.resume( piid );
   ```

   Esta acción pone la instancia de proceso y sus subprocesos en los estados que tenían antes de suspenderse.

Reinicio de un proceso de empresa:

Puede reiniciar una instancia de proceso que esté en estado finalizado, terminado, anómalo o compensado.

Antes de empezar

El llamante debe ser un administrador de la instancia de proceso o un administrador de procesos de empresa.

Acerca de esta tarea

Reiniciar una instancia de proceso es similar a iniciar una instancia de proceso por primera vez. Sin embargo, cuando se reinicia una instancia de proceso, se conoce el ID de instancia de proceso y el mensaje de entrada de la instancia queda disponible.

Si el proceso tiene más de una actividad de recepción o de obtención (también conocida como una actividad de recepción y elección) que pueda crear la instancia de proceso, todos los mensajes que pertenecen a estas actividades se utilizan para reiniciar la instancia de proceso. Si cualquiera de estas actividades implementa una operación de petición-respuesta, la respuesta se vuelve a enviar cuando se navegue por la actividad de respuesta asociada.

Procedimiento

1. Obtenga el proceso que desea reiniciar.

   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```

2. Reinicie la instancia de proceso.

   ```
   PIID piid = processInstance.getID();
   process.restart( piid );
   ```

   Esta acción reinicia la instancia de proceso especificada.

Terminar una instancia de proceso:

A veces, es necesario que un usuario que tenga autorización de administrador de procesos, termine una instancia de proceso de nivel superior de la que se sabe que está en estado irrecuperable. Dado que una instancia de proceso termina inmediatamente, sin esperar a subprocesos o actividades pendientes, debe terminar una instancia de proceso sólo en situaciones excepcionales.

Procedimiento

1. Recupere la instancia de proceso que se ha de finalizar.

   ```
   ProcessInstanceData processInstance =
          process.getProcessInstance("CustomerOrder");
   ```

2. Finalizar la instancia de proceso.

Si termina una instancia de proceso, puede terminar la instancia de proceso con o sin compensación.

Para finalizar la instancia de proceso sin compensación:

```
PIID piid = processInstance.getID();
process.forceTerminate(piid, CompensationBehaviour.INVOKE_COMPENSATION);
```

Para finalizar la instancia de proceso sin compensación:

```
PIID piid = processInstance.getID();
process.forceTerminate(piid);
```

Si termina la instancia de proceso con compensación, la compensación del proceso se ejecuta como si se hubiera producido un error en el ámbito de nivel superior. Si termina la instancia de proceso sin compensación, la instancia de proceso termina inmediatamente sin esperar a que finalicen normalmente las actividades, tareas a realizar o tareas de invocación en línea.

Las aplicaciones que inicia el proceso y las tareas autónomas que están relacionadas con el proceso no se terminan mediante la petición de forzar terminación. Si se han de terminar estas aplicaciones, debe añadir sentencias a la aplicación del proceso que termina explícitamente las aplicaciones iniciadas por el proceso.

Supresión de instancias de proceso:

Las instancias de proceso completadas se suprimen automáticamente de la base de datos de Business Process Choreographer si está establecida la propiedad correspondiente para la plantilla de proceso del modelo de proceso. Quizá desee conservar instancias de proceso en la base de datos, por ejemplo, para consultar datos de instancias de proceso que no se graban en las anotaciones cronológicas de auditoría. No obstante, los datos de la instancia de proceso almacenados no sólo afectan el espacio de disco y el rendimiento sino que también impiden que se creen instancias de proceso utilicen los mismos valores del conjunto de correlación. Por lo tanto, regularmente debe eliminar los datos de la instancia de proceso de la base de datos.

Acerca de esta tarea

Para suprimir una instancia de proceso, necesita derechos de administrador de procesos y la instancia de proceso debe ser una instancia de proceso de nivel superior.

En el ejemplo siguiente se muestra cómo suprimir todas las instancias de proceso finalizadas.

Procedimiento

1. Listar las instancias de proceso que han finalizado.

   ```
   QueryResultSet result =
        process.query("DISTINCT PROCESS_INSTANCE.PIID",
                      "PROCESS_INSTANCE.STATE = PROCESS_INSTANCE.STATE.STATE_FINISHED",
   (String)null, (Integer)null, (TimeZone)null);
   ```

   Esta acción devuelve un conjunto de resultados de consulta que lista las instancias de procesos finalizadas.

2. Suprima las instancias de proceso que hayan finalizado.

   ```
   while (result.next() )
   {
    PIID piid = (PIID) result.getOID(1);
    process.delete(piid);
   }
   ```

Esta acción suprime la instancia de proceso seleccionada y sus tareas en línea de la base de datos.

Proceso de actividades de tareas de usuario

Las actividades de tareas de usuario en procesos de empresa se asignan a distintas personas de la organización mediante elementos de trabajo. Cuando se inicia un proceso, se crean elementos de trabajo para los propietarios potenciales.

Acerca de esta tarea

Cuando se activa una actividad de tarea de usuario, se crean una instancia de actividad y una tarea a realizar asociada. El manejo de la actividad de tarea de usuario y la gestión de elementos de trabajo se delega al Gestor de tareas de usuario. Cualquier cambio de estado de la instancia de actividad se refleja en la instancia de tarea y viceversa.

Un propietario potencial reclama la actividad. Esta persona se encarga de proporcionar la información relevante y completar la actividad.

Procedimiento

1. Enumere las actividades que pertenecen a una persona que ha iniciado la sesión y que están preparadas para utilizarse:

```
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);
```

   Esta acción devuelve un conjunto de resultados de consulta que contiene las actividades con las que puede trabajar la persona que ha iniciado la sesión.

2. Reclame la actividad en la que se va a trabajar:

```
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();
    // leer los valores
    ...
  }
}
```

   Cuando se reclama la actividad, se devuelve el mensaje de entrada de la actividad.

3. Cuando haya acabado el trabajo en la actividad, finalice la actividad. La actividad puede completarse satisfactoriamente o con un mensaje de error. Si la actividad se realiza satisfactoriamente se pasa un mensaje de salida. Si no se realiza satisfactoriamente la actividad, se pone en el estado con anomalía o detenida y se pasa un mensaje de error. Deberá crear los mensajes adecuados para estas acciones. Cuando cree el mensaje, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje.
   a. Para completar la actividad correctamente, cree un mensaje de salida.

```
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();
  //establecer las partes del mensaje, por ejemplo, un número de pedido
  myMessage.setInt("OrderNo", 4711);
}

//completar la actividad
process.complete(aiid, output);
```

Esta acción establece un mensaje de salida que contiene el número de pedido.

b. Para completar la actividad cuando se produce un error, cree un mensaje de error.

```
//recuperar los errores diseñados para la actividad de tarea de usuario
List faultNames = process.getFaultNames(aiid);

//crear un mensaje del tipo adecuado
ClientObjectWrapper myFault =
      process.createMessage(aiid, faultNames.get(0));

// establecer las partes del mensaje de error, por ejemplo, un número
// de error
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //establecer las partes del mensaje, por ejemplo, un nombre de cliente
  myMessage.setInt("error",1304);
}

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

Esta acción establece la actividad en el estado con anomalía o detenida. Si el parámetro **continueOnError** de la actividad del modelo de proceso se establece en true, se pone la actividad en el estado con anomalía y continúa la navegación. Si el parámetro **continueOnError** se establece en false y el error no se captura en el ámbito que lo rodea, la actividad se pasa al estado detenido. En este estado se puede reparar la actividad utilizando force complete o force retry.

Proceso del flujo de trabajo de un solo usuario

Algunos flujos de trabajo sólo los realiza un usuario, por ejemplo pedir libros de una librería en línea. Este tipo de flujo de trabajo no tiene rutas paralelas. La API completeAndClaimSuccessor admite el proceso de este tipo de flujo de trabajo.

Acerca de esta tarea

En una librería en línea, el comprador completa una secuencia de acciones para pedir un libro. Esta secuencia de acciones se puede implementar como una serie de actividades de tareas de usuario (tareas a realizar). Si el comprador decide pedir varios libros esto es equivalente a reclamar la siguiente actividad de tarea de usuario. También se conoce este tipo de flujo de trabajo como *flujo de página* porque se asocian las definiciones de interfaz de usuario a las actividades para controlar el flujo de los diálogos de la interfaz de usuario.

La API completeAndClaimSuccessor completa una actividad de tarea de usuario y reclama la siguiente de la misma instancia de proceso para la persona que ha iniciado la sesión. Devuelve información sobre la siguiente actividad reclamada, incluido el mensaje de entrada sobre el que se va a actuar. Dado que la actividad siguiente está disponible dentro de la misma transacción de la actividad que se ha completado, el comportamiento transaccional de todas las actividades de tareas de usuario del modelo de proceso se debe establecer en `participates`.

Compare este ejemplo que utiliza la API de Business Flow Manager y la API de Human Task Manager.

Procedimiento

1. Reclame la primera actividad de la secuencia de actividades.

```
//
//Consultar la lista de actividades que el usuario conectado puede reclamar
//
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);
...
//
//Reclamar la primera actividad
//
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();
    // leer los valores
    ...
  }
}
```

   Cuando se reclama la actividad, se devuelve el mensaje de entrada de la actividad.

2. Cuando finalice el trabajo de la actividad, complete la actividad y relame la siguiente actividad.

   Para completar esta actividad, se pasa un mensaje de salida. Cuando cree el mensaje de salida, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje.

```
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();
  //establecer las partes del mensaje, por ejemplo, un número de pedido
  myMessage.setInt("OrderNo", 4711);
}
```

```
//complete la actividad y reclame la siguiente
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

Esta acción establece un mensaje de salida que contiene el número de pedido y reclama la siguiente actividad de la secuencia. Si se establece `AutoClaim` para las actividades de sucesor y hay varias vías de acceso que se pueden seguir, se reclaman todas las actividades de sucesor y se devuelve una actividad aleatoria como la actividad siguiente. Si no hay más actividades de sucesor que se puedan asignar a este usuario, se devuelve `Null`.

Si el proceso contiene vías de acceso paralelas que se pueden seguir y estas vías de acceso contienen actividades de tareas de usuario para las que el usuario conectado es un propietario potencial de más de una de estas actividades, se reclama automáticamente una actividad aleatoria y se devuelve como la actividad siguiente.

3. Trabaje en la siguiente actividad.

```
String name = successor.getActivityName();

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // leer los valores
  ...
}

aiid = successor.getAIID();
```

4. Continúe con el paso 2 para completar la actividad.

Tareas relacionadas

"Proceso del flujo de trabajo de una sola persona que incluye tareas de usuario" en la página 106
Algunos flujos de trabajo sólo los realiza un usuario, por ejemplo pedir libros de una librería en línea. En este ejemplo se muestra cómo implementar la secuencia de acciones para pedir el libro como una serie de actividades de tareas de usuario (tareas a realizar). Se utiliza Business Flow Manager y las API de Human Task Manager para procesar el flujo de trabajo.

Envío de un mensaje a una actividad en espera

Puede utilizar las actividades de mensajes de entrada (actividades de recepción, onMessage en actividades de captación, onEvent en manejadores de sucesos) para sincronizar un proceso en ejecución con sucesos del "mundo exterior". Por ejemplo, un suceso de este tipo puede ser cuando se recibe un correo electrónico de un cliente como respuesta a una petición de información.

Acerca de esta tarea

Para enviar el mensaje a la actividad puede utilizar las tareas que la originan.

Procedimiento

1. Liste las plantillas de servicios de actividades que están a la espera de un mensaje del usuario conectado en una instancia de proceso con un ID de instancia de proceso específico.

```
ActivityServiceTemplateData[] services = process.getWaitingActivities(piid);
```

2. Envíe un mensaje al primer servicio en espera.

Se presupone que el primer servicio es el que desea servir. El llamante debe ser el iniciador potencial de la actividad que recibe el mensaje o un administrador de la instancia de proceso.

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

// crear un mensaje para el servicio que se va a llamar
  ClientObjectWrapper message =
        process.createMessage(vtid,atid,inputMessageTypeName);
  DataObject myMessage = null;
  if ( message.getObject()!=
        null && message.getObject() instanceof DataObject )
  {
    myMessage = (DataObject)message.getObject();
    // establecer las series del mensaje, por ejemplo, se va a realizar
    // un pedido de chocolate
    myMessage.setString("Order", "chocolate");
  }

  // enviar el mensaje a la actividad que está en espera
  process.sendMessage(vtid, atid, message);
}
```

Esta acción envía el mensaje especificado al servicio de actividades en espera y se pasarán algunos datos del pedido.

También puede especificar el ID de instancia de proceso para asegurarse de que se envía el mensaje a la instancia de proceso especificada. Si no se especifica el ID de instancia de proceso, se envía el mensaje al servicio de actividades y a la instancia de proceso que identifican los valores de correlación del mensaje. Si se especifica el ID de instancia de proceso, se comprueba la instancia de proceso que se ha encontrado utilizando los valores de correlación para asegurarse de que tiene el ID de instancia de proceso especificado.

Manejo de sucesos

Un proceso de empresa completo y cada uno de sus ámbitos puede asociarse con manejadores de sucesos que se invocan si se produce el suceso asociado. Los manejadores de sucesos son similares para recibir o seleccionar actividades en lo referente a que un proceso puede proporcionar operaciones de servicios Web mediante manejadores de sucesos.

Acerca de esta tarea

Puede invocar un manejador de sucesos cualquier número de veces mientras se ejecute el ámbito correspondiente. Además, varias instancias de un manejador de sucesos pueden activarse de forma simultánea.

El siguiente fragmento de código muestra cómo obtener los manejadores de sucesos activos para una instancia de proceso determinada y cómo enviar un mensaje de entrada.

Procedimiento

1. Determine los datos del ID de instancia de proceso y liste los manejadores de sucesos activos para el proceso.

```
ProcessInstanceData processInstance =
       process.getProcessInstance( "CustomerOrder2711");
EventHandlerTemplateData[] events = process.getActiveEventHandlers(
                                      processInstance.getID() );
```

2. Envíe el mensaje de entrada.

Este ejemplo utiliza el primer manejador de sucesos que se encuentra.

```
EventHandlerTemplateData event = null;
if ( events.length > 0 )
{
    event = events[0];

    // crear un mensaje para el servicio que se va a llamar
    ClientObjectWrapper input = process.createMessage(
    event.getID(), event.getInputMessageTypeName());

    if (input.getObject() != null && input.getObject() instanceof DataObject )
    {
       DataObject inputMessage = (DataObject)input.getObject();
       // establecer contenido del mensaje, por ejemplo, un nombre de
       // cliente y número de pedido
       inputMessage.setString("CustomerName", "Smith");
       inputMessage.setString("OrderNo", "2711");

        // enviar el mensaje
       process.sendMessage( event.getProcessTemplateName(),
                            event.getPortTypeNamespace(),
                            event.getPortTypeName(),
                            event.getOperationName(),
     input );
     }
 }
```

Esta acción envía el mensaje especificado al manejador de sucesos activo para el proceso.

Análisis de los resultados de un proceso

Un proceso puede exponer las operaciones de servicios Web que están diseñadas como operaciones unidireccionales o de petición y respuestas de tipo WSDL (Web Services Description Language). Los resultados de procesos de larga duración con interfaces unidireccionales no pueden recuperarse mediante el método getOutputMessage porque el proceso no tiene salida. En cambio, sin embargo, puede consultar el contenido de las variables.

Acerca de esta tarea

Los resultados del proceso sólo se almacenan en la base de datos si la plantilla de proceso de la que se ha derivado la instancia de proceso no especifica que el mensaje de salida se ha de suprimir automáticamente.

Procedimiento

Analizar los resultados del proceso, por ejemplo, comprobar el número de pedido.

```
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");  }
}
```

Reparación de actividades

Un proceso de larga ejecución puede contener actividades que también son de larga ejecución. Estas actividades pueden encontrar errores no descubiertos y pasar al estado detenido. Una actividad que está en estado de en ejecución también podría parecer que no está respondiendo. En los dos casos, un administrador de procesos puede actuar sobre la actividad de distintos modos de manera que pueda continuar la navegación del proceso.

Acerca de esta tarea

La API de Business Process Choreographer proporciona los métodos forceRetry y forceComplete para reparar las actividades. Se proporcionan ejemplos que muestran cómo se pueden añadir acciones de reparación de actividades a las aplicaciones del usuario.

Forzar la finalización de una actividad: Acerca de esta tarea

A veces, las actividades de procesos de larga ejecución pueden encontrar errores. Si estos errores no son captados por el manejador de errores en el ámbito circundante y si la plantilla de la actividad asociada especifica que la actividad se detenga cuando se produzca un error, la actividad se pasará al estado de detenida para que se pueda reparar. En este estado, puede forzar la finalización de la actividad.

También puede forzar la finalización de actividades en estado de ejecución si, por ejemplo, una actividad no responde.

Existen requisitos adicionales para ciertos tipos de actividades.

Actividades de las tareas de usuario
: Puede pasar parámetros a la llamada force-complete, como el mensaje que debería haberse enviado o el error que debería haberse producido.

Actividades de script
: No puede pasar parámetros en la llamada force-complete. No obstante, debe establecer las variables que se han de reparar.

Invocar actividades
: También puede forzar actividades de invocación completas que llaman a un servicio asíncrono que no sea un subproceso si estas actividades están en estado de ejecución. Puede que desee hacer esto, por ejemplo, si se llama al servicio asíncrono y éste no responde.

Procedimiento

1. Listar las actividades detenidas en estado detenido.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Esta acción devuelve las actividades detenidas para la instancia de proceso CustomerOrder.

2. Completar la actividad, por ejemplo, una actividad de tarea de usuario detenida.

   En este ejemplo, se pasa un mensaje de salida.

```
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();
     //establecer las partes del mensaje, por ejemplo, un número de pedido
     myMessage.setInt("OrderNo", 4711);
   }

   boolean continueOnError = true;
   process.forceComplete(aiid, output, continueOnError);
}
```

Esta acción completa la actividad. Si se produce un error, el parámetro **continueOnError** determina la acción que se va a llevar a cabo si se proporciona un error con la petición forceComplete.

En el ejemplo, el valor de **continueOnError** es true. Este valor significa que si se proporciona un error, la actividad se coloca en estado de error. El error se propaga a los ámbitos que circundan la actividad hasta que se maneja o hasta que se alcanza el ámbito del proceso. A continuación, el proceso se pone en estado de ejecución errónea hasta que finalmente pasa a estado erróneo.

Reintento de la ejecución de una actividad detenida: Acerca de esta tarea

Si durante una actividad de un proceso de larga ejecución se produce un error no capturado en el ámbito que la circunda y si la plantilla de la actividad asociada especifica que la actividad se detenga cuando se produzca un error, la actividad se pasa al estado de detenida para que se pueda reparar. Puede intentar la ejecución de la actividad otra vez.

Puede establecer las variables que utiliza la actividad. A excepción de las actividades de script, también puede pasar parámetros en la llamada force-retry como, por ejemplo, el mensaje que esperaba la actividad.

Procedimiento

1. Listar las actividades detenidas.

   ```
   QueryResultSet result =
         process.query("DISTINCT ACTIVITY.AIID",
                       "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                        PROCESS_INSTANCE.NAME='CustomerOrder'",
                        (String)null, (Integer)null, (TimeZone)null);
   ```

   Esta acción devuelve las actividades detenidas para la instancia de proceso CustomerOrder.

2. Reintentar la ejecución de la actividad, por ejemplo, una actividad de tareas de usuario detenida.

   ```
   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();
         // establecer las series del mensaje, por ejemplo, se va a realizar un
         // pedido de chocolate
   ```

```
        myMessage.setString("OrderNo", "chocolate");
      }

     boolean continueOnError = true;
     process.forceRetry(aiid, input, continueOnError);
}
```

Esta acción reintenta la actividad. Si se produce un error, el parámetro **continueOnError** determina la acción que se realizará si se produce un error durante el proceso de la petición forceRetry.

En el ejemplo, el valor de **continueOnError** es true. Esto significa que si se produce un error durante el proceso de la petición forceRetry, la actividad se pasa a estado de ejecución errónea. El error se propaga a los ámbitos que circundan la actividad hasta que se maneja o hasta que se alcanza el ámbito del proceso. A continuación, el proceso se pone en estado de ejecución errónea y se ejecuta un manejador de errores en el nivel de proceso antes de que el estado de proceso termine en el estado erróneo.

Interfaz BusinessFlowManagerService

La interfaz BusinessFlowManagerService expone funciones de proceso de empresa que una aplicación cliente puede llamar.

Los métodos que la interfaz BusinessFlowManagerService puede llamar dependen del estado del proceso o la actividad y la autorización de la persona que utilice la aplicación que contiene el método. Los métodos principales para manejar objetos de proceso de empresa se listan aquí. Para obtener más información sobre estos y otros métodos que están disponibles en la interfaz BusinessFlowManagerService, consulte el Javadoc que se encuentra en el paquete com.ibm.bpe.api.

Plantillas de proceso

Una plantilla de proceso es un modelo de proceso versionado, desplegado e instalado que contiene la especificación de un proceso de empresa. Se puede crear la instancia e iniciarse emitiendo las peticiones correspondiente, por ejemplo, sendMessage(). La ejecución de la instancia de proceso la dirige automáticamente el servidor.

Tabla 24. Métodos API para plantillas de proceso

Método	Descripción
getProcessTemplate	Recupera la plantilla de proceso especificada.
queryProcessTemplates	Recupera plantillas de proceso que se almacenan en la base de datos.

Instancias de proceso

Los siguientes métodos API están relacionados con el inicio de instancias de proceso.

Tabla 25. Los métodos API están relacionados con el inicio de instancias de proceso

Método	Descripción
call	Crea y ejecuta un microflujo.

Tabla 25. Los métodos API están relacionados con el inicio de instancias de proceso (continuación)

Método	Descripción
callWithReplyContext	Crea y ejecuta un microflujo con un servicio de arranque exclusivo o un proceso de larga ejecución con un servicio de arranque exclusivo a partir de la plantilla de proceso especificada. La llamada espera de forma asíncrona al resultado.
callWithUISettings	Crea y ejecuta un microflujo y devuelve el mensaje de salida y los valores de la interfaz de usuario (UI) de cliente.
initiate	Crea una instancia de proceso e inicia el proceso de la instancia de proceso. Utilice este método para procesos de larga ejecución. También puede utilizar este método para microflujos que desea activar y omitir.
sendMessage	Envía el mensaje especificado al servicio de actividad y la instancia de proceso especificados. Si no existe una instancia de proceso con los mismos valores de conjunto de correlaciones, se creará. El proceso puede tener servicios de arranque exclusivos y no exclusivos.
getStartActivities	Devuelve información sobre las actividades que pueden iniciar una instancia de proceso a partir de la plantilla de proceso especificada.
getActivityServiceTemplate	Recupera la plantilla de servicio de actividad especificada.

Tabla 26. Métodos API para controlar el ciclo de vida de las instancias de proceso

Método	Descripción
suspender	Suspende la ejecución de una instancia de proceso de nivel superior y larga ejecución que está en el estado de ejecución o anómalo.
resume	Reanuda la ejecución de una instancia de proceso de nivel superior y larga ejecución que está en el estado suspendido.
restart	Reinicia una instancia de proceso de nivel superior y larga ejecución en el estado finalizado, anómalo o terminado.
forceTerminate	Termina la instancia de proceso de nivel superior especificada, sus subprocesos con autonomía de hijo y sus actividades de ejecución, reclamadas o en espera.
delete	Suprime la instancia de proceso de nivel superior especificada y sus subprocesos con autonomía de hijo.

Tabla 26. Métodos API para controlar el ciclo de vida de las instancias de proceso (continuación)

Método	Descripción
query	Recupera las propiedades de la base de datos que cumplen los criterios de búsqueda.

Actividades

Para las actividades de invocación, puede especificar en el modelo de proceso que estas actividades continúan en situaciones de error. Si el distintivo continueOnError se establece en false y se produce un error no manejado, la actividad se coloca en estado detenido. A continuación, un administrador de proceso puede reparar la actividad. El distintivo continueOnError y las funciones de reparación asociadas pueden, por ejemplo, utilizarse en un proceso de larga ejecución en el que ocasionalmente falla una actividad de invocación pero el esfuerzo necesario para modelar la compensación y la gestión de errores es demasiado elevado.

Los métodos siguientes están disponibles para trabajar con actividades y repararlas.

Tabla 27. Métodos API para controlar el ciclo de vida de las instancias de actividad

Método	Descripción
claim	Reclama una instancia de actividad preparada para que un usuario trabaje en la actividad.
cancelClaim	Cancela la reclamación de la instancia de actividad.
complete	Completa la instancia de actividad.
completeAndClaimSuccessor	Completa la instancia de actividad y reclama la siguiente en la misma instancia de proceso para la persona que ha iniciado la sesión.
forceComplete	Fuerza la finalización de una instancia de actividad que está en el estado de ejecución o detenido.
forceRetry	Fuerza la repetición de una instancia de actividad que está en el estado de ejecución o detenido.
query	Recupera las propiedades de la base de datos que cumplen los criterios de búsqueda.

Variables y propiedades personalizadas

La interfaz proporciona un método get y set para recuperar y establecer valores para variables. También puede asociar las propiedades con nombre con, y recuperar propiedades con nombre de, las instancias de proceso y actividad. Los nombres y valores de propiedad personalizados deben ser del tipo java.lang.String.

Tabla 28. Métodos API para variables y propiedades personalizadas

Método	Descripción
getVariable	Recupera la variable especificada.
setVariable	Establece la variable especificada.
getCustomProperty	Recupera la propiedad personalizada indicada de la instancia de actividad o proceso especificada.
getCustomProperties	Recupera las propiedades personalizadas de la actividad especificada o instancia de proceso.
getCustomPropertyNames	Recupera los nombres de las propiedades personalizadas de la instancia de actividad o proceso especificada.
setCustomProperty	Almacena valores específicos personalizados para la instancia de actividad o proceso especificada.

Desarrollo de aplicaciones para tareas de usuario

Una tarea consiste en los medios con los que los componentes invocan a usuarios como servicios o con los que los usuarios invocan servicios. Se proporcionan ejemplos de aplicaciones típicas para tareas de usuario.

Acerca de esta tarea

Para obtener más información sobre la API del Gestor de tareas de usuario, consulte el Javadoc en el paquete com.ibm.task.api.

Inicio de una tarea de invocación que invoca una interfaz síncrona

Una tarea de invocación está asociada con un componente SCA (Java Service Component Architecture). Cuando se inicia la tarea, invoca el componente SCA. Inicie una tarea de invocación de forma síncrona sólo si el componente SCA asociado puede invocarse de forma síncrona.

Acerca de esta tarea

Por ejemplo, este tipo de componente SCA puede implementarse como un microflujo o como una simple clase Java.

Este escenario crea una instancia de una plantilla de tarea y pasa algunos datos de cliente. La tarea permanece en estado de ejecución hasta que se devuelve la operación bidireccional. El resultado de la tarea, OrderNo, se devuelve al llamante.

Procedimiento

1. Opcional: Liste las plantillas de tarea para encontrar el nombre de la tarea de invocación que desea ejecutar.

   Este paso es opcional si ya conoce el nombre de la tarea.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
 "TASK_TEMPL.NAME",
  new Integer(50),
  (TimeZone)null);
```

El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas originadoras clasificadas.

2. Cree un mensaje de entrada del tipo adecuado.

```
TaskTemplate template = taskTemplates[0];

// crear un mensaje para la tarea seleccionada
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //establecer las partes del mensaje, por ejemplo, un nombre de cliente
  myMessage.setString("CustomerName", "Torres");
}
```

3. Cree la tarea y ejecútela de forma síncrona.

   Para que una tarea se ejecute de forma síncrona, debe ser una operación bidireccional. El ejemplo utiliza el método createAndCallTask para crear y ejecutar la tarea.

```
ClientObjectWrapper output = task.createAndCallTask( template.getName(),
                                                     template.getNamespace(),
                                                     input);
```

4. Analice el resultado de la tarea.

```
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");}
```

Inicio de una tarea de invocación que invoca una interfaz asíncrona

Una tarea de invocación está asociada con un componente SCA (Java Service Component Architecture). Cuando se inicia la tarea, invoca el componente SCA. Inicie una tarea de invocación de forma asíncrona sólo si el componente SCA asociado puede invocarse de forma asíncrona.

Acerca de esta tarea

Por ejemplo, este tipo de componente SCA puede implementarse como un proceso de larga ejecución o una operación de una dirección.

Este escenario crea una instancia de una plantilla de tarea y pasa algunos datos de cliente.

Procedimiento

1. Opcional: Liste las plantillas de tarea para encontrar el nombre de la tarea de invocación que desea ejecutar.

   Este paso es opcional si ya conoce el nombre de la tarea.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
  ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
   "TASK_TEMPL.NAME",
    new Integer(50),
    (TimeZone)null);
```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas originadoras clasificadas.

2. Cree un mensaje de entrada del tipo adecuado.

```
TaskTemplate template = taskTemplates[0];

// crear un mensaje para la tarea seleccionada
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //establecer las partes del mensaje, por ejemplo, un nombre de cliente
  myMessage.setString("CustomerName", "Torres");
}
```

3. Cree la tarea y ejecútela de forma asíncrona.

   El ejemplo utiliza el método createAndStartTask para crear y ejecutar la tarea.

```
task.createAndStartTask( template.getName(),
                         template.getNamespace(),
                         input,
                         (ReplyHandlerWrapper)null);
```

Creación e inicio de una instancia de tarea

En este caso de ejemplo se muestra cómo crear una instancia de una plantilla de tarea que define una tarea de colaboración (también conocida como *tarea de usuario* en la API) e iniciar la instancia de tarea.

Procedimiento

1. Opcional: Liste las plantillas de tarea para encontrar el nombre de la tarea de colaboración que desea ejecutar.

   Este paso es opcional si ya conoce el nombre de la tarea.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_HUMAN",
 "TASK_TEMPL.NAME",
  new Integer(50),
  (TimeZone)null);
```

   El resultado se clasifica por nombre. La consulta devuelve una matriz que contiene las 50 primeras plantillas de tareas de clasificadas.

2. Cree un mensaje de entrada del tipo adecuado.

```
TaskTemplate template = taskTemplates[0];

// crear un mensaje para la tarea seleccionada
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //establecer las partes del mensaje, por ejemplo, un nombre de cliente
  myMessage.setString("CustomerName", "Torres");
}
```

3. Cree e inicie la tarea de colaboración; en este ejemplo no se especifica un manejador de respuestas.

   En el ejemplo se utiliza el método createAndStartTask para crear e iniciar la tarea.

```
TKIID tkiid = task.createAndStartTask( template.getName(),
                                       template.getNamespace(),
                                       input,
                                       (ReplyHandlerWrapper)null);
```

   Se crearán elementos de trabajo de los usuarios a los que les interesa la instancia de tarea. Por ejemplo, un propietario potencial puede reclamar la nueva instancia de tarea.

4. Reclame la instancia de tarea.

```
ClientObjectWrapper input2 = task.claim(tkiid);
DataObject taskInput = null ;
if ( input2.getObject()!= null && input2.getObject() instanceof DataObject )
{
  taskInput = (DataObject)input2.getObject();
  // leer los valores
  ...
}
```

Cuando se reclama la instancia de tarea, se devuelve el mensaje de entrada de la tarea.

Proceso de tareas a realizar o de colaboración

Las tareas a realizar (también conocidas como *tareas participativas* en la API) o tareas de colaboración (también conocidas como *tareas de usuario* en la API) se asignan a varias personas de la organización mediante elementos de trabajo. Las tareas a realizar y sus elementos de trabajo asociados se crean, por ejemplo, cuando un proceso navega hacia una actividad de tareas de usuario.

Acerca de esta tarea

Uno de los propietarios potenciales reclama la tarea asociada con el elemento de trabajo. Esta persona es responsable de proporcionar la información relevante y completar la tarea.

Procedimiento

1. Liste las tareas pertenecientes a una persona que ha iniciado la sesión y que están preparadas para trabajar con ellas.

```
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);
```

   Esta acción devuelve un conjunto de resultados de consulta que contiene las tareas con las que puede trabajar la persona que ha iniciado la sesión.

2. Reclame la tarea en la que se va a trabajar.

```
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();
    // leer los valores
    ...
  }
}
```

   Cuando se reclama la tarea, se devuelve el mensaje de entrada de la tarea.

3. Cuando haya acabado el trabajo en la tarea, complete la tarea.

   La tarea se puede completar satisfactoriamente o con un mensaje de error. Si la tarea se realiza satisfactoriamente se pasa un mensaje de salida. Si la tarea no se realiza satisfactoriamente se pasa un mensaje de error. Deberá crear los mensajes adecuados para estas acciones.

   a. Para completar la tarea correctamente, cree un mensaje de salida.

```
ClientObjectWrapper output =
      task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //establecer las partes del mensaje, por ejemplo, un número de pedido
  myMessage.setInt("OrderNo", 4711);
}

//completar la tarea
task.complete(tkiid, output);
```

Esta acción establece un mensaje de salida que contiene el número de pedido. La tarea se coloca en el estado de finalizada.

b. Para completar la tarea cuando se produce un error, cree un mensaje de error.

```
//recuperar los errores diseñados para la tarea
List faultNames = task.getFaultNames(tkiid);

//crear un mensaje del tipo adecuado
ClientObjectWrapper myFault =
      task.createFaultMessage(tkiid, (String)faultNames.get(0));

// establecer las partes del mensaje de error, por ejemplo, un número
// de error
DataObject myMessage = null ;
if ( myFault.getObject()!=
               null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //establecer las partes del mensaje, por ejemplo, un nombre de cliente
  myMessage.setInt("error",1304);
}

task.complete(tkiid, (String)faultNames.get(0), myFault);
```

Esta acción establece un mensaje de error que contiene el código de error. La tarea se coloca en el estado de finalizada.

Suspensión y reanudación de instancias de tarea

Puede suspender las instancias de tareas de colaboración (también conocidas como *tareas de usuario* en la API) o instancias de tareas a realizar (también conocidas como *tareas participativas* en la API).

Antes de empezar

La instancia de tarea puede estar en el estado de preparada o reclamada. Se puede escalar. El llamante debe ser el propietario, el originador o el administrador de la instancia de tarea.

Acerca de esta tarea

Puede suspender una instancia de tarea cuando está en ejecución. Podría querer hacerlo, por ejemplo, de modo que pueda recabar información que es necesaria para completar la tarea. Cuando esté disponible la información, puede reanudar la instancia de tarea.

Procedimiento

1. Obtenga una lista de tareas reclamadas por el usuario conectado.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                   "TASK.STATE = TASK.STATE.STATE_CLAIMED",
                                   (String)null,
                                   (Integer)null,
                                   (TimeZone)null);
```

Esta acción devuelve un conjunto de resultados de consulta que contiene una lista de las tareas reclamadas por el usuario conectado.

2. Suspenda la instancia de tarea.

```
if (result.size() > 0)
{
   result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.suspend(tkiid);
}
```

Esta acción suspende la instancia de tarea especificada. Se pondrá la instancia de tarea en estado suspendido.

3. Reanude la instancia de proceso.

```
task.resume( tkiid );
```

Esta acción pone la instancia de tarea en el estado que tenía antes de que se suspendiera.

Análisis de los resultados de una tarea

Una tarea a realizar (también conocida como *tarea participativa* en la API) o una tarea de colaboración (también conocida como *tarea de usuario* en la API) se ejecuta asíncronamente. Si se especifica un manejador de respuestas cuando se inicia la tarea, se devuelve automáticamente el mensaje de salida cuando se completa la tarea. Si no se especifica un manejador de respuestas, el mensaje debe recuperarse explícitamente.

Acerca de esta tarea

Los resultados de la tarea sólo se almacenan en la base de datos si la plantilla de tarea de la que se ha derivado la instancia de tarea no especifica la supresión automática de las instancias de tarea derivadas.

Procedimiento

Analice los resultados de la tarea.

El ejemplo muestra cómo comprobar el número de pedido de una tarea completada satisfactoriamente.

```
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");  }
}
```

Terminación de una instancia de tarea

A veces, es necesario que un usuario que tenga derechos de administrador de procesos, termine una instancia de tarea de la que se sabe que está en estado irrecuperable. Dado que la instancia de tarea se finaliza de forma inmediata, debe finalizar una instancia de tarea solamente en situaciones excepcionales.

Procedimiento

1. Recuperar la instancia de tarea que se ha de terminar.

   ```
   Task taskInstance = task.getTask(tkiid);
   ```

2. Terminar la instancia de tarea.

   ```
   TKIID tkiid = taskInstance.getID();
   task.terminate(tkiid);
   ```

   La instancia de tarea termina inmediatamente sin esperar a otras actividades pendientes.

Supresión de instancias de tarea

Las instancias de tareas sólo se suprimen automáticamente cuando finalizan si se ha especificado así en la plantilla de tarea asociada de donde se derivan dichas instancias. En este ejemplo se muestra cómo suprimir todas las instancias de tarea que han finalizado y no se han suprimido automáticamente.

Procedimiento

1. Liste las instancias de tareas que han finalizado.

   ```
   QueryResultSet result =
        task.query("DISTINCT TASK.TKIID",
                   "TASK.STATE = TASK.STATE.STATE_FINISHED",
                    (String)null, (Integer)null, (TimeZone)null);
   ```

   Esta acción devuelve un conjunto de resultados de consulta que lista las instancias de tarea finalizadas.

2. Suprima las instancias de tarea que hayan finalizado.

   ```
   while (result.next() )
   {
    TKIID tkiid = (TKIID) result.getOID(1);
    task.delete(tkiid);
   }
   ```

Liberación de una tarea reclamada

Cuando un propietario potencial reclama una tarea, esta persona se encarga de completar la tarea. No obstante, a veces la tarea reclamada debe liberarse, de modo que otro propietario potencial pueda reclamarla.

Acerca de esta tarea

A veces, es necesario que un usuario que tenga derechos de administrador libere una tarea reclamada. Esto puede suceder, por ejemplo, cuando deba completarse una tarea pero el propietario de la tarea esté ausente. El propietario de la tarea también puede liberar una tarea reclamada.

Procedimiento

1. Liste las tareas reclamadas que son propiedad de una persona específica, por ejemplo, Smith.

```
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_CLAIMED AND
                 TASK.OWNER = 'Smith'",
                 (String)null, (Integer)null, (TimeZone)null);
```

Esta acción devuelve un conjunto de resultados de consulta que lista las tareas que ha reclamado la persona especificada, Smith.

2. Libere la tarea reclamada.

```
if (result.size() > 0)
{
   result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.cancelClaim(tkiid, true);
}
```

Esta acción devuelve la tarea a estado preparado para que uno de los demás propietarios potenciales pueda reclamarla. Se conservarán los datos de salida o de error establecidos por el propietario inicial.

Gestión de elementos de trabajo

Durante la vida de una instancia de actividad o una instancia de tarea, el conjunto de personas asociadas con el objeto puede cambiar, por ejemplo, porque una persona está de vacaciones, se han contratado otras personas o la carga de trabajo tiene que distribuirse de forma diferente. Para que puedan realizarse estos cambios, puede desarrollar aplicaciones para crear, suprimir o transferir elementos de trabajo.

Acerca de esta tarea

Un elemento de trabajo representa la asignación de un objeto a un usuario o a un grupo de usuarios para un motivo en particular. El objeto es habitualmente una instancia de actividad de tareas de usuario, una instancia de proceso o una instancia de tarea. Los motivos se derivan del rol que el usuario tenga para el objeto. Un objeto puede tener varios elementos de trabajo, porque un usuario puede tener distintos roles en asociación con el objeto, y se crea un elemento de trabajo para cada uno de estos roles. Por ejemplo, una instancia de tarea a realizar puede tener un elemento de trabajo de administrador, lector, editor y propietario al mismo tiempo.

Las acciones que se pueden realizar para gestionar elementos de trabajo dependen del rol que tiene el usuario, por ejemplo, un administrador puede crear, suprimir y transferir elementos de trabajo, pero el propietario de tareas sólo puede transferir elementos de trabajo.

- Crear un elemento de trabajo.

```
// consultar la instancia de tarea para la que se ha de especificar
// un administrador adicional
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
   result.first();
  // crear el elemento de trabajo
  task.createWorkItem((TKIID)(result.getOID(1)),
                        WorkItem.REASON_ADMINISTRATOR,"Smith");
}
```

Esta acción crea un elemento de trabajo para el usuario Smith que tiene el rol de administrador.

- Eliminar un elemento de trabajo.

```
// consultar la instancia de tarea de la que se suprimirá un elemento de trabajo
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
   result.first();
  // eliminar el elemento de trabajo
  task.deleteWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_READER,"Smith");
}
```

  Esta acción suprime el elemento de trabajo para el usuario Smith que tiene el rol de lector.

- Transferir un elemento de trabajo.

```
// consultar la tarea que se ha de volver a planificar
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();
  // transferir el elemento de trabajo del usuario Miller al usuario Smith,
  // para que Smith pueda trabajar en la tarea
  task.transferWorkItem((TKIID)(result.getOID(1)),
                           WorkItem.REASON_POTENTIAL_OWNER,"Miller","Smith");
}
```

  Esta acción transfiere el elemento de trabajo al usuario Smith para que pueda trabajar en el mismo.

Creación de plantillas de tarea e instancias de tarea durante la ejecución

Habitualmente debe utilizarse una herramienta de modelado como, por ejemplo, WebSphere Integration Developer para construir plantillas de tarea. A continuación, instale las plantillas de tarea en WebSphere Process Server y cree instancias a partir de estas plantillas, por ejemplo, mediante Business Process Choreographer Explorer. Sin embargo, también puede crear plantillas o instancias de tareas de usuario o participativas durante la ejecución.

Acerca de esta tarea

Tal vez desee hacerlo, por ejemplo, cuando la definición de tarea no esté disponible cuando se despliegue la aplicación, las tareas que formen parte de un flujo de trabajo no se conozcan todavía, o se necesite una tarea para cubrir colaboraciones ad-hoc entre un grupo de personas.

Puede modelar ad-hoc tareas a realizar o de colaboración mediante la creación de instancias de la clase com.ibm.task.api.TaskModel y su utilización para crear una plantilla de tarea reutilizable, o bien para crear directamente una instancia de tarea de una sola ejecución. Para crear una instancia de la clase TaskModel, hay disponible un conjunto de métodos de fábrica en la clase de fábrica com.ibm.task.api.ClientTaskFactory. El modelado de tareas de usuario en tiempo de ejecución está basado en EMF (Eclipse Modeling Framework).

Procedimiento

1. Cree un org.eclipse.emf.ecore.resource.ResourceSet con el método de fábrica createResourceSet.
2. Opcional: Si tiene previsto utilizar tipos de mensaje complejos, puede definirlos con el org.eclipse.xsd.XSDFactory que puede obtener con el método de fábrica getXSDFactory(), o bien importar directamente un esquema XML con el método de fábrica loadXSDSchema.

   Para que los tipos complejos estén disponibles en WebSphere Process Server, despliéguelos como parte de una aplicación de empresa.
3. Cree o importe una definición WSDL (Web Services Definition Language) del tipo javax.wsdl.Definition.

   Puede crear una nueva definición de WSDL utilizando el método createWSDLDefinition. A continuación, puede añadirle un tipo de puerto y una operación. También puede importar directamente una definición WSDL existente mediante el método de fábrica loadWSDLDefinition.
4. Cree la definición de tarea mediante el método de fábrica createTTask.

   Si desea añadir o manipular más elementos de tarea compleja, utilice la clase com.ibm.wbit.tel.TaskFactory que puede recuperar mediante el método de fábrica getTaskFactory.
5. Cree el modelo de tarea mediante el método de fábrica createTaskModel y páselo al paquete de recursos creado en el paso 1, que agrega otros artefactos que se hayan creado mientras tanto.
6. Opcional: Valide el modelo mediante el método validate de TaskModel.

Resultado

Utilice uno de los métodos create de la API del EJB del Gestor de tareas de usuario que tenga un parámetro **TaskModel** para crear una plantilla de tarea reutilizable, o bien una instancia de una sola ejecución.

Creación de tareas de tiempo de ejecución que utilizan tipos Java simples:

Este ejemplo crea una tarea de tiempo de ejecución que sólo utiliza tipos Java complejos en su interfaz, por ejemplo, un objeto Serie.

Acerca de esta tarea

El ejemplo sólo se ejecuta en el contexto de la aplicación de empresa que llama, para la que se cargan los recursos.

Procedimiento

1. Acceda a ClientTaskFactory y cree un conjunto de recursos que contenga las definiciones del nuevo modelo de tarea.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```
2. Cree la definición WSDL y añada las descripciones de las operaciones.

   ```
   // crear la interfaz WSDL
   Definition definition = factory.createWSDLDefinition
            ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

   // crear un tipo de puerto
   PortType portType = factory.createPortType( definition, "doItPT" );

   // crear una operación; los mensajes de entrada y salida son de tipo Serie:
   ```

```
// no se especifica un mensaje de anomalía
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. Cree el modelo EMF de la nueva tarea de usuario.

   Si crea una instancia de tarea, no se necesita una fecha de válido-desde (UTCDate).

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   Este paso inicializa las propiedades del modelo de tarea con valores por omisión.

4. Modifique las propiedades del modelo de tarea de usuario.

```
// utilizar los métodos del paquete com.ibm.wbit.tel, por ejemplo,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// recuperar la fábrica de tareas para crear o modificar elementos de
// tarea compuestos
TaskFactory taskFactory = factory.getTaskFactory();

// especificar valores de escalada
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// crear escalationReceiver y añadir verbo
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// crear escalada y añadir receptor de escalada
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Cree un modelo de tarea que contenga todas las definiciones de recurso

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. Valide el modelo de tarea y corrija los problemas de validación que se encuentren.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. Cree la instancia o plantilla de tarea de tiempo de ejecución.

   Utilice la interfaz HumanTaskManagerService para crear la instancia de tarea o la plantilla de tarea. Dado que la aplicación sólo utiliza tipos Java simples, no es necesario que especifique un nombre de aplicación.

   - El snippet siguiente crea una instancia de tarea:

     ```
     atask.createTask( taskModel, (String)null, "HTM" );
     ```

   - El snippet siguiente crea una plantilla de tarea:

     ```
     task.createTaskTemplate( taskModel, (String)null );
     ```

Resultado

Si se ha creado una instancia de tarea de tiempo de ejecución, ahora puede iniciarse. Si se ha creado una plantilla de tarea de tiempo de ejecución, ahora puede crear instancias de tarea a partir de la plantilla.

Creación de tareas de tiempo de ejecución que utilizan tipos complejos:

Este ejemplo crea una tarea de tiempo de ejecución que utiliza tipos complejos en su interfaz. Los tipos complejos ya están definidos, es decir, el sistema de archivos local en el cliente tiene archivos XSD que contienen la descripción de los tipos complejos.

Acerca de esta tarea

El ejemplo sólo se ejecuta en el contexto de la aplicación de empresa que llama, para la que se cargan los recursos.

Procedimiento

1. Acceda a ClientTaskFactory y cree un conjunto de recursos que contenga las definiciones del nuevo modelo de tarea.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. Añada las definiciones XSD de los tipos complejos al conjunto de recursos, de manera que estén disponibles cuando defina las operaciones.

   Los archivos están ubicados en una posición relativa a la del lugar donde se ejecuta el código.

   ```
   factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
   factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
   ```

3. Cree la definición WSDL y añada las descripciones de las operaciones.

   ```
   // crear la interfaz WSDL
   Definition definition = factory.createWSDLDefinition
            ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

   // crear un tipo de puerto
   PortType portType = factory.createPortType( definition, "doItPT" );

   // crear una operación; el mensaje de entrada es un InputBO y
   // el mensaje de salida es un OutputBO;
   // no se especifica un mensaje de anomalía
   Operation operation = factory.createOperation
           ( definition, portType, "doIt",
             new QName( "http://Input", "InputBO" ),
             new QName( "http://Output", "OutputBO" ),
             (Map)null );
   ```

4. Cree el modelo EMF de la nueva tarea de usuario.

   Si crea una instancia de tarea, no se necesita una fecha de válido-desde (UTCDate).

   ```
   TTask humanTask = factory.createTTask( resourceSet,
                                          TTaskKinds.HTASK_LITERAL,
                                          "TestTask",
                                          new UTCDate( "2005-01-01T00:00:00" ),
                                          "http://www.ibm.com/task/test/",
                                          portType,
                                          operation );
   ```

   Este paso inicializa las propiedades del modelo de tarea con valores por omisión.

5. Modifique las propiedades del modelo de tarea de usuario.

   ```
   // utilizar los métodos del paquete com.ibm.wbit.tel, por ejemplo,
   humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

   // recuperar la fábrica de tareas para crear o modificar elementos de
   // tarea compuestos
   TaskFactory taskFactory = factory.getTaskFactory();
   ```

```
// especificar valores de escalada
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// crear escalationReceiver y añadir verbo
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// crear escalada y añadir receptor de escalada
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. Cree un modelo de tarea que contenga todas las definiciones de recurso

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

7. Valide el modelo de tarea y corrija los problemas de validación que se encuentren.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

8. Cree la instancia o plantilla de tarea de tiempo de ejecución.

   Utilice la interfaz HumanTaskManagerService para crear la instancia de tarea o la plantilla de tarea. Debe proporcionar un nombre de aplicación que contenga las definiciones de tipo de datos para que se pueda acceder a ellas. La aplicación debe contener también una tarea o proceso ficticio para que Business Process Choreographer cargue la aplicación.

   - El snippet siguiente crea una instancia de tarea:

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - El snippet siguiente crea una plantilla de tarea:

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

Resultado

Si se ha creado una instancia de tarea de tiempo de ejecución, ahora puede iniciarse. Si se ha creado una plantilla de tarea de tiempo de ejecución, ahora puede crear instancias de tarea a partir de la plantilla.

Creación de tareas de tiempo de ejecución que utilizan una interfaz existente:

Este ejemplo crea una tarea de tiempo de ejecución que utiliza una interfaz que ya está definida, es decir, el sistema de archivos local en el cliente tiene un archivo que contiene la descripción de la interfaz.

Acerca de esta tarea

El ejemplo sólo se ejecuta en el contexto de la aplicación de empresa que llama, para la que se cargan los recursos.

Procedimiento

1. Acceda a ClientTaskFactory y cree un conjunto de recursos que contenga las definiciones del nuevo modelo de tarea.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. Acceda a la definición de WSDL y a las descripciones de las operaciones.

   La descripción de la interfaz está ubicada en una posición relativa a la del lugar donde se ejecuta el código.

```
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. Cree el modelo EMF de la nueva tarea de usuario.

   Si crea una instancia de tarea, no se necesita una fecha de válido-desde (UTCDate).

   ```
   TTask humanTask = factory.createTTask( resourceSet,
                                          TTaskKinds.HTASK_LITERAL,
                                          "TestTask",
                                          new UTCDate( "2005-01-01T00:00:00" ),
                                          "http://www.ibm.com/task/test/",
                                          portType,
                                          operation );
   ```

   Este paso inicializa las propiedades del modelo de tarea con valores por omisión.

4. Modifique las propiedades del modelo de tarea de usuario.

   ```
   // utilizar los métodos del paquete com.ibm.wbit.tel, por ejemplo,
   humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

   // recuperar la fábrica de tareas para crear o modificar elementos de
   // tarea compuestos
   TaskFactory taskFactory = factory.getTaskFactory();

   // especificar valores de escalada
   TVerb verb = taskFactory.createTVerb();
   verb.setName("John");

   // crear escalationReceiver y añadir verbo
   TEscalationReceiver escalationReceiver =
                       taskFactory.createTEscalationReceiver();
   escalationReceiver.setVerb(verb);

   // crear escalada y añadir receptor de escalada
   TEscalation escalation = taskFactory.createTEscalation();
   escalation.setEscalationReceiver(escalationReceiver);
   ```

5. Cree un modelo de tarea que contenga todas las definiciones de recurso

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

6. Valide el modelo de tarea y corrija los problemas de validación que se encuentren.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

7. Cree la instancia o plantilla de tarea de tiempo de ejecución.

   Utilice la interfaz HumanTaskManagerService para crear la instancia de tarea o la plantilla de tarea. Debe proporcionar un nombre de aplicación que contenga las definiciones de tipo de datos para que se pueda acceder a ellas. La aplicación debe contener también una tarea o proceso ficticio para que Business Process Choreographer cargue la aplicación.

   - El snippet siguiente crea una instancia de tarea:

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - El snippet siguiente crea una plantilla de tarea:

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

Resultado

Si se ha creado una instancia de tarea de tiempo de ejecución, ahora puede iniciarse. Si se ha creado una plantilla de tarea de tiempo de ejecución, ahora puede crear instancias de tarea a partir de la plantilla.

Creación de tareas de tiempo de ejecución que utilizan una interfaz desde la aplicación que llama:

Este ejemplo crea una tarea de tiempo de ejecución que utiliza una interfaz que forma parte de la aplicación que llama. Por ejemplo, la tarea de ejecución se crea en un snippet Java de un proceso de empresa y utiliza una interfaz de la aplicación de proceso.

Acerca de esta tarea

El ejemplo sólo se ejecuta en el contexto de la aplicación de empresa que llama, para la que se cargan los recursos.

Procedimiento

1. Acceda a ClientTaskFactory y cree un conjunto de recursos que contenga las definiciones del nuevo modelo de tarea.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();

// especificar el cargador de clase de contexto para que se encuentren los
// siguientes recursos
ResourceSet resourceSet = factory.createResourceSet
              ( Thread.currentThread().getContextClassLoader() );
```

2. Acceda a la definición de WSDL y a las descripciones de las operaciones.

   Especifique la vía de acceso del archivo JAR de contenedor.

```
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. Cree el modelo EMF de la nueva tarea de usuario.

   Si crea una instancia de tarea, no se necesita una fecha de válido-desde (UTCDate).

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   Este paso inicializa las propiedades del modelo de tarea con valores por omisión.

4. Modifique las propiedades del modelo de tarea de usuario.

```
// utilizar los métodos del paquete com.ibm.wbit.tel, por ejemplo,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// recuperar la fábrica de tareas para crear o modificar elementos de
// tarea compuestos
TaskFactory taskFactory = factory.getTaskFactory();

// especificar valores de escalada
TVerb verb = taskFactory.createTVerb();
verb.setName("John");
```

```
// crear escalationReceiver y añadir verbo
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// crear escalada y añadir receptor de escalada
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Cree un modelo de tarea que contenga todas las definiciones de recurso

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

6. Valide el modelo de tarea y corrija los problemas de validación que se encuentren.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

7. Cree la instancia o plantilla de tarea de tiempo de ejecución.

   Utilice la interfaz HumanTaskManagerService para crear la instancia de tarea o la plantilla de tarea. Debe proporcionar un nombre de aplicación que contenga las definiciones de tipo de datos para que se pueda acceder a ellas.

   - El snippet siguiente crea una instancia de tarea:

     ```
     task.createTask( taskModel, "WorkflowApplication", "HTM" );
     ```

   - El snippet siguiente crea una plantilla de tarea:

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

Resultado

Si se ha creado una instancia de tarea de tiempo de ejecución, ahora puede iniciarse. Si se ha creado una plantilla de tarea de tiempo de ejecución, ahora puede crear instancias de tarea a partir de la plantilla.

Interfaz HumanTaskManagerService

La interfaz HumanTaskManagerService expone funciones relativas a tareas que un cliente local o remoto puede llamar.

Los métodos que pueden llamarse dependen del estado de la tarea y la autorización de la persona que utiliza la aplicación que contiene el método. Los métodos principales para manejar objetos de tarea se listan aquí. Para obtener más información sobre estos y otros métodos que están disponibles en la interfaz HumanTaskManagerService, consulte el Javadoc que se encuentra en el paquete com.ibm.task.api.

Plantillas de tarea

Para trabajar con plantillas de tareas dispone de los métodos siguientes.

Tabla 29. Métodos API para plantillas de tareas.

Método	Descripción
getTaskTemplate	Recupera la plantilla de tarea especificada.
createAndCallTask	Crea y ejecuta una instancia de tarea a partir de la plantilla de tarea especificada y espere el resultado de manera síncrona.
createAndStartTask	Crea e inicia una instancia de tarea a partir de la plantilla de tarea especificada.
createTask	Crea una instancia de tarea a partir de la plantilla de tarea especificada.

Tabla 29. Métodos API para plantillas de tareas. (continuación)

Método	Descripción
createInputMessage	Crea un mensaje de entrada para la plantilla de tarea especificada. Por ejemplo, cree un mensaje que pueda utilizarse para iniciar una tarea.
queryTaskTemplates	Recupera plantillas de tarea que se almacenan en la base de datos.

Instancias de tareas

Para trabajar con instancias de tarea se dispone de los métodos siguientes.

Tabla 30. Métodos API para instancias de tareas.

Método	Descripción
getTask	Recupera una instancia de tarea; la instancia de tarea puede estar en cualquier estado.
callTask	Inicia una tarea de invocación de forma síncrona.
startTask	Inicia una tarea que ya se ha creado.
suspender	Suspende la tarea a realizar o de colaboración.
resume	Reanuda la tarea a realizar o de colaboración.
terminate	Termina la instancia de tarea especificada. Si se termina una tarea de invocación, esta acción no tiene ningún impacto en el servicio invocado.
delete	Suprime la instancia de tarea especificada.
claim	Reclama la tarea para el proceso.
actualización	Actualiza la instancia de tarea.
complete	Completa la instancia de tarea.
cancelClaim	Libera una instancia de tarea reclamada de manera que pueda trabajar con ella otro propietario poetencial.
createWorkItem	Crea un elemento de trabajo para la instancia de tarea.
transferWorkItem	Transfiere el elemento de trabajo a un propietario especificado.
deleteWorkItem	Suprime el elemento de trabajo.

Escaladas

Para trabajar con escaladas se dispone de los métodos siguientes.

Tabla 31. Métodos API para trabajar con escaladas

Método	Descripción
getEscalation	Recupera la instancia de escalada especificada.

Propiedades personalizadas

Todas las tareas, plantillas de tareas y escaladas tienen sus propiedades personalizadas. La interfaz proporciona un método get y un método set para recuperar y establecer los valores para las propiedades personalizadas. También puede asociar las propiedades con nombre con, y recuperar propiedades con nombre de instancias de tarea. Los nombres y valores de propiedad personalizados deben ser del tipo java.lang.String. Los métodos siguientes son válidos para tareas, plantillas de tareas y escaladas.

Tabla 32. Métodos API para variables y propiedades personalizadas

Método	Descripción
getCustomProperty	Recupera la propiedad personalizada indicada de la instancia de tarea especificada.
getCustomProperties	Recupera las propiedades personalizadas para la instancia de tarea especificada.
getCustomPropertyNames	Recupera los nombres de las propiedades personalizadas de la instancia de tarea.
setCustomProperty	Almacena valores específicos personalizados para la instancia de tarea especificada.

Acciones permitidas para tareas:

Las acciones que pueden realizarse en una tarea dependen de si se trata de una tarea a realizar, una tarea de colaboración, una tarea de invocación o una tarea administrativa.

No puede utilizar todas las acciones proporcionadas por la interfaz HumanTaskManager para todas las clases de tareas. La tabla siguiente muestra las acciones que pueden llevarse a cabo en cada clase de tarea.

	Clase de tarea			
Acción	**Tarea a realizar**	**Tarea de colaboración**	**Tarea de invocación**	**Tarea administrativa**
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

Acción	Clase de tarea			
	Tarea a realizar	**Tarea de colaboración**	**Tarea de invocación**	**Tarea administrativa**
getFaultNames	X	X[1]		
getFaultMessage	X	X[1]	X	
getInputMessage	X	X[1]	X	
getOutputMessage	X	X[1]	X	
getUsersInRole	X	X[1]	X	X
getTask	X	X[1]	X	X
getUISettings	X	X[1]	X	X
resume	X	X[1]		
setCustomProperty	X	X[1]	X	X
setFaultMessage	X	X[1]		
setOutputMessage	X	X[1]		
startTask	X[1]	X[1]	X	X
startTaskAsSubtask[6]	X	X[1]		
startTaskAsSubtask[7]		X[3]	X[3]	
suspend	X	X[1]		
suspendWithCancelClaim	X	X[1]		
terminate	X[1]	X[1]	X[1]	
transferWorkItem	X	X[1]	X	X
update	X	X[1]	X	X

Notas:

1. Para tareas autónomas, tareas ad-hoc y plantillas de tareas únicamente.
2. Para tareas autónomas, tareas en línea de procesos de empresa y tareas de tiempo de ejecución únicamente
3. Sólo para tareas autónomas y tareas ad-hoc
4. Los tipos de tareas que pueden tener tareas de continuación
5. Los tipos de tareas que se pueden utilizar como tareas de continuación
6. Los tipos de tareas que pueden tener subtareas
7. Los tipos de tareas que se pueden utilizar como subtareas

Desarrollo de aplicaciones para procesos de empresa y tareas de usuario

Las personas implicadas en la mayoría de los escenarios de procesos de empresa. Por ejemplo, un proceso de empresa requiere interacción con personas cuando se inicia o administra el proceso, o bien cuando se efectúan actividades de tareas de usuario. Para dar soporte a estos escenarios, debe utilizar la API de Business Flow Manager y la API del Gestor de tareas de usuario.

Acerca de esta tarea

Para implicar personas en escenarios de procesos de empresa, puede incluir los siguientes tipos de tareas en el proceso de empresa:

- Una tarea de invocación en línea (también conocida como *tarea de origen* en la API).

Puede proporcionar una tarea de invocación para cada actividad de recepción, para cada elemento onMessage de una actividad de selección y para cada elemento onEvent de un manejador de sucesos. A continuación, esta tarea controla quién está autorizado para iniciar un proceso o comunicarse con una instancia de proceso en ejecución.

- Una tarea de administración.

  Puede proporcionar una tarea de administración para especificar quién está autorizado para administrar el proceso o para realizar operaciones administrativas sobre las actividades del proceso que contengan errores.
- Una tarea a realizar (también conocida como *tarea participativa* en la API).

  Una tarea a realizar implementa una actividad de tarea de usuario. Este tipo de actividad permite implicar a personas en el proceso.

Las actividades de tareas de usuario del proceso de empresa representan las tareas a realizar que la gente lleva a cabo en el escenario de proceso de empresa. Puede utilizar la API de Business Flow Manager y la API del Gestor de tareas de usuario para realizar estos escenarios.

- El proceso de empresa es el contenedor para todas las actividades que pertenecen al proceso, incluidas las actividades de tareas de usuario que se representan mediante tareas a realizar. Cuando se crea una instancia de proceso, se le asigna un ID de objeto (PIID) exclusivo.
- Cuando se activa una actividad de tarea de usuario durante la ejecución de la instancia de proceso, se crea una instancia de actividad que se identifica mediante su ID de objeto (AIID) exclusivo. Al mismo tiempo, también se crea una instancia de tarea a realizar en línea que se identifica mediante su ID de objeto (TKIID). La relación de la actividad de tarea de usuario con la instancia de tarea se lleva a cabo mediante los ID de objeto:
  - El ID de tarea a realizar de la instancia de actividad se establece en el TKIID de la tarea a realizar asociada.
  - El ID de contexto de contenedor de la instancia de tarea se establece en el PIID de la instancia de proceso que contiene la instancia de actividad asociada.
  - El ID de contexto padre de la instancia de tarea se establece en el AIID de la instancia de actividad asociada.
- La instancia de proceso gestiona los ciclos de vida de todas las instancias de tareas a realizar en línea. Cuando se suprime la instancia de proceso, también se suprimen las instancias de tarea. En otras palabras, todas las tareas que tengan el ID de contexto de contenedor establecido en el PIID de la instancia de proceso se suprimen automáticamente.

Determinación de las plantillas de proceso o actividades que pueden iniciarse

Un proceso de empresa puede iniciarse invocando los métodos call, initiate, o sendMessage de la API de Business Flow Manager. Si el proceso solo tiene una actividad de inicio, puede utilizar la signatura de método que necesita un nombre de plantilla de proceso como parámetro. Si el proceso tiene más de una actividad de inicio, debe identificarla explícitamente.

Acerca de esta tarea

Cuando se modela un proceso de empresa, el modelador puede decidir que solo un subconjunto de usuarios pueda crear una instancia de proceso a partir de la plantilla de proceso. Esto se lleva a cabo asociando una tarea de invocación en línea a una actividad de inicio del proceso y especificando restricciones de

autorización en esa tarea. Solo las personas que sean administradores o iniciadores potenciales de la tarea pueden crear una instancia de la tarea y, por consiguiente, una instancia de la plantilla de proceso.

Si una tarea de invocación en línea no está asociada con la actividad de inicio o si no se especifican las restricciones de autorización para la tarea, todos podrán crear una instancia de proceso mediante la actividad de inicio.

Un proceso puede tener más de una actividad de inicio, cada una de ellas con diferentes consultas de personas sobre administradores o iniciadores potenciales. Esto significa que un usuario puede estar autorizado para iniciar un proceso con la actividad A, pero no con la actividad B.

Procedimiento

1. Utilice la API de Business Flow Manager para crear una lista de las versiones actuales de plantillas de proceso que están en el estado de iniciado.

   Consejo: El método queryProcessTemplates sólo excluye las plantillas de proceso que forman parte de aplicaciones que todavía no se han iniciado. Por tanto, si utiliza este método sin filtrar los resultados, el método devuelve todas las versiones de las plantillas de proceso independientemente del estado en que se encuentren.

   ```
   // indicación de fecha y hora actuales en formato UTC, convertido
   // en aaaa-mm-ddThh:mm:ss
   String now = (new UTCDate()).toXsdString();
   String whereClause = "PROCESS_TEMPLATE.STATE =
                         PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                         PROCESS_TEMPLATE.VALID_FROM =
                         (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                          WHERE NAME=PROCESS_TEMPLATE.NAME AND
                          VALID_FROM <= TS('" + now + "'))";

   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
                                           ( whereClause,
                                             "PROCESS_TEMPLATE.NAME",
                                             (Integer)null, (TimeZone)null);
   ```

   Los resultados se clasifican por nombre de plantilla de proceso.

2. Cree la lista de plantillas de proceso y la lista de actividades de inicio para las que el usuario está autorizado.

   La lista de plantillas de proceso contiene las que tienen una sola actividad de inicio. Estas actividades no están protegidas o el usuario que ha iniciado la sesión puede iniciarlas. También es posible reunir las plantillas de proceso que se puedan iniciar por parte de al menos una de las actividades de inicio.

   Consejo: Un administrador de proceso también puede iniciar una instancia de proceso. Para obtener una lista completa de plantillas, también debe leer la plantilla de tarea de administración que esté asociada con la plantilla de proceso y comprobar si el usuario que ha iniciado la sesión es un administrador.

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

3. Determine las actividades de inicio para cada plantilla de proceso.

   ```
   for( int i=0; i<processTemplates.length; i++ )
   {
     ProcessTemplateData template = processTemplates[i];
     ActivityServiceTemplateData[] startActivities =
                   process.getStartActivities(template.getID());
   ```

4. Para cada actividad de inicio, recupere el ID de la plantilla de tarea de invocación en línea asociada.

```
for( int j=0; j<startActivities.length; j++ )
{
  ActivityServiceTemplateData activity = startActivities[j];
  TKTID tktid = activity.getTaskTemplateID();
```

   a. Si una plantilla de tarea de invocación no existe, la plantilla de proceso no está protegida por esta actividad de inicio.

      En este caso, todas las personas pueden crear una instancia de proceso utilizando esta actividad de inicio.

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

   b. Si existe una plantilla de tarea de invocación, utilice la API del Gestor de tareas de usuario para comprobar la autorización para el usuario que ha iniciado la sesión.

      En el ejemplo, el usuario que ha iniciado la sesión es Torres. El usuario que ha iniciado la sesión debe ser un iniciador potencial de la tarea de invocación o un administrador.

```
if ( tktid != null )
    {
      isAuthorized =
        task.isUserInRole
            (tkid, "Torres", WorkItem.REASON_POTENTIAL_STARTER) ||
        task.isUserInRole(tktid, "Torres", WorkItem.REASON_ADMINISTRATOR);

      if ( isAuthorized )
      {
        authorizedActivityServiceTemplates.add(activity);
      }
    }
```

      Si el usuario tiene el rol especificado o si los criterios de asignación de personas para el rol no se han especificado, el método isUserInRole devuelve el valor `true`.

5. Compruebe si el proceso puede iniciarse utilizando solo el nombre de plantilla de proceso.

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

6. Termine los bucles.

```
  } // fin del bucle para cada plantilla de servicio de actividad
}   // fin del bucle para cada plantilla de proceso
```

Proceso del flujo de trabajo de una sola persona que incluye tareas de usuario

Algunos flujos de trabajo sólo los realiza un usuario, por ejemplo pedir libros de una librería en línea. En este ejemplo se muestra cómo implementar la secuencia de acciones para pedir el libro como una serie de actividades de tareas de usuario (tareas a realizar). Se utiliza Business Flow Manager y las API de Human Task Manager para procesar el flujo de trabajo.

Acerca de esta tarea

En una librería en línea, el comprador completa una secuencia de acciones para pedir un libro. Esta secuencia de acciones se puede implementar como una serie de actividades de tareas de usuario (tareas a realizar). Si el comprador decide pedir varios libros esto es equivalente a reclamar la siguiente actividad de tarea de usuario. Business Flow Manager mantiene la información sobre la secuencia de tareas, mientras que Human Task Manager mantiene las tareas en sí.

Compare este ejemplo con el que utiliza solamente la API de Business Flow Manager.

Procedimiento

1. Utilice la API de Business Flow Manager para obtener la instancia de proceso en la que desea trabajar.

   En este ejemplo, una instancia del proceso CustomerOrder.

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

2. Utilice la API de Human Task Manager para consultar las tareas a realizar preparadas (kind participating, tipo participación) que son parte de la instancia de proceso especificada.

   Utilice el ID de contexto de contención de la tarea para especificar la instancia de proceso que la contiene. Para un flujo de trabajo de una sola persona, la consulta devuelve la tarea a realizar asociada a la primera actividad de tarea de usuario en la secuencia de actividades de tareas de usuario.

   ```
   //
    //Consultar la lista de tareas a realizar que el usuario conectado puede reclamar
    // para la instancia de proceso especificada
   //
   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. Reclame la tarea a realizar que se devuelve.

   ```
   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();
       // leer los valores
       ...
     }
   }
   ```

   Cuando se reclama la tarea, se devuelve el mensaje de entrada de la tarea.

4. Determine la actividad de tarea de usuario asociada a la tarea a realizar.

   Puede utilizar uno de estos métodos para correlacionar las actividades con sus tareas.

   - El método task.getActivityID:

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

   - El ID de contexto padre es parte del objeto de tarea:

```
AIID aiid = null;
Task taskInstance = task.getTask(tkiid);
OID oid = taskInstance.getParentContextID();
if ( oid != null and oid instanceof AIID )
{
  aiid = (AIID)oid;
}
```

5. Cuando haya finalizado el trabajo de la tarea, utilice la API de Business Flow Manager para completar la tarea y su actividad de tarea de usuario asociada, y reclame la próxima actividad de tarea de usuario de la instancia de proceso.

   Para completar la actividad de tarea de usuario, se pasa un mensaje de salida. Cuando cree el mensaje de salida, debe especificar el nombre del tipo de mensaje, para que contenga la definición del mensaje.

```
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();
  //establecer las partes del mensaje, por ejemplo, un número de pedido
  myMessage.setInt("OrderNo", 4711);
}

//completar la actividad de tarea de usuario y su tarea a realizar asociada
// y reclamar la siguiente actividad de tarea de usuario
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   Esta acción establece un mensaje de salida que contiene el número de pedido y reclama la siguiente actividad de la secuencia. Si se establece `AutoClaim` para las actividades de sucesor y hay varias vías de acceso que se pueden seguir, se reclaman todas las actividades de sucesor y se devuelve una actividad aleatoria como la actividad siguiente. Si no hay más actividades de sucesor que se puedan asignar a este usuario, se devuelve `Null`.

   Si el proceso contiene vías de acceso paralelas que se pueden seguir y estas vías de acceso contienen actividades de tareas de usuario para las que el usuario conectado es un propietario potencial de más de una de estas actividades, se reclama automáticamente una actividad aleatoria y se devuelve como la actividad siguiente.

6. Trabaje en la siguiente tarea de usuario.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // leer los valores
  ...
}

aiid = successor.getAIID();
```

7. Continúe con el paso 5 para completar la actividad de tarea de usuario y para recuperar la siguiente actividad de tarea de usuario.

Tareas relacionadas

"Proceso del flujo de trabajo de un solo usuario" en la página 75
Algunos flujos de trabajo sólo los realiza un usuario, por ejemplo pedir libros de una librería en línea. Este tipo de flujo de trabajo no tiene rutas paralelas. La API completeAndClaimSuccessor admite el proceso de este tipo de flujo de trabajo.

Manejo de excepciones y errores

Es posible que un proceso BPEL encuentre un error en puntos diferentes del proceso.

Acerca de esta tarea

Los errores BPEL (Business Process Execution Language) se originan a partir de:

- Invocaciones de servicios Web, errores WSDL (Web Services Description Language)
- Actividades de generación
- Los errores estándar BPEL que Business Process Choreographer reconoce

Existen mecanismos para gestionar estos errores. Utilice uno de estos mecanismos para gestionar los errores generados por una instancia de proceso:

- Ceda el control a los manejadores de errores correspondientes
- Compense el trabajo anterior del proceso
- Detenga el proceso y permita que alguien repare la situación (force-retry, force-complete)

Un proceso BPEL también devuelve errores al que invoca una operación proporcionada por el proceso. Puede diseñar el error en el proceso como una actividad de respuesta con un nombre de error y datos de error. Estos errores se devuelven al que invoca la API como excepciones comprobadas.

Si un proceso BPEL no maneja un error BPEL o si se produce una excepción de la API, se devuelve una excepción de tiempo de ejecución al que invoca a la API. Un ejemplo de una excepción de la API es cuando no existe el modelo de proceso del que se va a crear una instancia.

En las tareas siguientes se describe cómo manejar los errores y excepciones

Manejo de excepciones de la API

Acerca de esta tarea

Si un método de la interfaz BusinessFlowManagerService o HumanTaskManagerService no se completa correctamente, se genera una excepción que indica la causa del error. Puede manejar esta excepción específicamente para proporcionar alguna directriz al llamante

No obstante, en la práctica general se maneja únicamente un subconjunto de las excepciones específicamente y para el resto de las excepciones potencias se proporcionan directrices generales. Todas las excepciones específicas se heredan de una excepción ProcessException o TaskException genérica. El *método recomendado* es capturar las excepciones genéricas que finalizan con la sentencia `catch(ProcessException)` o `catch(TaskException)`. Esta sentencia le ayuda a asegurarse la compatibilidad con versiones posteriores de su programa de aplicación ya que tiene en cuenta todas las demás excepciones que se pueden producir.

Comprobación del error establecido para una actividad

Procedimiento

1. Liste las actividades de tarea que están en estado erróneo o detenido.

```
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);
```

Esta acción devuelve un conjunto de resultados de la consulta que contiene las actividades erróneas o detenidas.

2. Leer el nombre del error.

```
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();
  }
}
```

Esto devuelve el nombre del error. También puede analizar la excepción no manejada para una actividad detenida en lugar de recuperar el nombre del error.

Comprobar si se ha producido un error para una actividad de invocación detenida

Acerca de esta tarea

Si una actividad causa la aparición de un error, el tipo de error determina las acciones que puede emprender para reparar la actividad.

Procedimiento

1. Listar las actividades de tareas de usuario que están en estado detenido.

```
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);
```

Esta acción devuelve un conjunto de resultados de consulta que contiene actividades de invocación detenidas.

2. Leer el nombre del error.

```
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();
  }
}
```

Desarrollo de aplicaciones cliente de la API de servicio Web

Puede desarrollar aplicaciones cliente que acceden a las aplicaciones de proceso de empresa y de tareas de usuario mediante las API de servicios Web.

Acerca de esta tarea

Las aplicaciones cliente se pueden desarrollar en cualquier entorno de cliente de servicio Web, incluidos los servicios Web Java y Microsoft .NET.

Introducción: servicios Web

Los servicios Web son aplicaciones de empresa Web que utilizan estándares abiertos basados en XML y protocolos de transporte para intercambiar los datos con aplicaciones cliente. Los servicios Web permiten utilizar un modelo de programación neutro de lenguaje y entorno.

Los servicios Web utilizan las siguientes tecnologías principales:

- XML (Extensible Markup Language). XML soluciona el problema de independencia de los datos. Puede utilizarlo para describir los datos y también para correlacionar esos datos dentro y fuera de cualquier aplicación o lenguaje de programación
- WSDL (Web Services Description Language). Puede utilizar este lenguaje basado en XML para crear una descripción de una aplicación subyacente. Se trata de la descripción que convierte una aplicación en un servicio Web, actuando de interfaz entre la aplicación subyacente y otras aplicaciones habilitadas para la Web.
- SOAP (Protocolo de acceso a objetos simple). SOAP es el protocolo de comunicaciones principal de la Web y la mayoría de los servicios Web utilizan este protocolo para comunicarse unos con otros.

Componentes de servicio Web y secuencia de control

Varios componentes en el cliente y en el servidor participan en la secuencia de control que representa una petición y respuesta de un servicio Web.

A continuación se muestra una secuencia habitual de control.

1. En el cliente:
   a. Una aplicación cliente (proporcionada por el usuario) emite una petición de un servicio Web.
   b. Un cliente proxy (que también proporciona el usuario pero que se puede generar automáticamente utilizando programas de utilidad en el cliente) incluye la petición de servicio en un sobre de petición SOAP.
   c. La infraestructura de desarrollo en el cliente reenvía la petición a un URL definido como el punto final del servicio Web.
2. La red transmite la petición al punto final del servicio Web utilizando HTTP o HTTPS.
3. En el servidor:
   a. La API de servicios Web genérica recibe y decodifica la petición.
   b. La petición se maneja directamente mediante el componente Business Flow Manager o Human Task Manager o se dirige al proceso de empresa o tarea de usuario especificado.
   c. Los datos devueltos se incluyen en un sobre de respuesta SOAP.

4. La red transmite la respuesta al entorno del extremo del cliente utilizando HTTP o HTTPS.
5. De nuevo en el cliente:
   a. La infraestructura de desarrollo en el cliente desenvuelve el sobre de respuesta SOAP.
   b. El cliente proxy extrae los datos de la respuesta SOAP y los transfiere a la aplicación cliente.
   c. La aplicación cliente procesa los datos devueltos como corresponda.

Visión general de las API de servicios Web

Las API de los servicios Web le permiten desarrollar aplicaciones de cliente que utilizan servicios Web para acceder a los procesos de empresa y a las tareas de usuario en el entorno de Business Process Choreographer.

La API de servicios Web de Business Process Choreographer proporcionan dos interfaces de servicios Web diferentes (tipos de puerto WSDL):

- La API de Business Flow Manager. Permite que las aplicaciones cliente interactúen con los microflujos y procesos de larga ejecución, por ejemplo:
  - Crear plantillas de proceso e instancias de proceso
  - Reclamar procesos existentes
  - Consultar procesos por su ID

  Consulte "Desarrollo de aplicaciones para procesos de empresa" en la página 64 para obtener una lista completa de las acciones posibles.
- La API de Human Task Manager. Permite a las aplicaciones cliente:
  - Crear e iniciar tareas
  - Reclamar tareas existentes
  - Completar tareas
  - Consultar tareas por su ID
  - Consultar una colección de tareas.

  Consulte "Desarrollo de aplicaciones para tareas de usuario" en la página 85 para obtener una lista completa de las acciones posibles.

Las aplicaciones de cliente pueden utilizar cualquiera o las dos interfaces de servicios Web.

Ejemplo

La siguiente es una posible descripción de una aplicación de cliente que accede a la API de servicios Web de Human Task Manager para procesar una tarea de usuario participante:

1. La aplicación cliente emite una llamada de servicio Web query a WebSphere Process Server que solicita una lista de tareas participantes en las que trabajará un usuario.
2. La lista de tareas de participación se devuelve en un sobre de respuesta SOAP/HTTP.
3. La aplicación cliente que emite una llamada de servicio Web a claim para reclamar una de las tareas de participación.
4. WebSphere Process Server devuelve el mensaje de entrada de la tarea.
5. La aplicación cliente emite una llamada de servicio Web a complete para completar la tarea con un mensaje de salida o de error.

Requisitos para los procesos de empresa y las tareas de usuario

Los procesos de empresa y las tareas de usuario desarrolladas con WebSphere Integration Developer para que se ejecuten en Business Process Choreographer deben ajustarse a normas específicas para que se pueda acceder a los mismos desde las API de servicios Web.

Los requisitos son:

1. Las interfaces de los procesos de empresa y las tareas de usuario se deben definir con el estilo ″documento/literal con envoltura″ definido en la API Java para la especificación RPC basada en XML, JAX-RPC 1.1. Este es el estilo por omisión para todos los procesos de empresa y tareas de usuario desarrollados con WID.
2. Los mensajes de error que exponen los procesos de empresa y las tareas de usuario para las operaciones de servicios Web deben constar de una parte de mensaje WSDL individual definida con el elemento de esquema XML. Por ejemplo:

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

   Información relacionada

   Página de descargas de la API Java para RPC basado en XML, JAX-RPC

   ¿Qué estilo de WSDL debe utilizar?

Desarrollo de aplicaciones cliente

El proceso de desarrollo de aplicaciones cliente consta de varios pasos.

Procedimiento

1. Decida qué API de servicios Web necesita utilizar su aplicación de cliente: la API de Business Flow Manager, la API de Human Task Manager o ambas.
2. Exporte los archivos necesarios del entorno WebSphere Process Server. Alternativamente, puede copiar los archivos desde el CD del cliente de WebSphere Process Server.
3. En el entorno de desarrollo de aplicaciones cliente seleccionado, genere un *cliente proxy* utilizando los artefactos exportados.
4. Opcional: Generación de las clases *helper*. Las clases helper son necesarias si la aplicación de cliente interactúa directamente con procesos o tareas concretos en el servidor WebSphere. No obstante, no son necesarios si la aplicación cliente sólo va a realizar tareas genéricas como la emisión de consultas.
5. Desarrolle el código de la aplicación cliente.
6. Añada los mecanismos de seguridad necesarios para la aplicación cliente.

Copia de artefactos

Se deben copiar varios artefactos desde el entorno WebSphere para ayudar a crear las aplicaciones de cliente.

Hay dos modos de obtener estos artefactos:

- Publicar y exportarlas desde el entorno de WebSphere Process Server.
- Copiar los archivos desde el CD del cliente de WebSphere Process Server.

Publicación y exportación de artefactos desde el entorno de servidor

Para poder desarrollar aplicaciones de cliente para acceder a las API de servicios Web, debe publicar y exportar varios artefactos desde el entorno de servidor de WebSphere.

Acerca de esta tarea

Los artefactos que se han de exportar son:

- Los archivos WSDL (Web Service Definition Language) que describen los tipos de puerto y las operaciones que componen las API de servicios Web.
- Los archivos XSD (XML Schema Definition) que contienen definiciones de los tipos de datos a los que hacen referencias los servicios y métodos de los archivos WSDL.
- Archivos WSDL y XSD que describen objetos de empresa. Objetos de empresa que describen procesos de empresa concretos o tareas de usuario que se ejecutan en el servidor de WebSphere. Estos archivos adicionales sólo son necesarios si la aplicación de cliente tiene que interactuar directamente con procesos de empresa o tareas de usuario concretas mediante las API de servicios Web. No son necesarios si la aplicación de cliente sólo va a realizar tareas genéricas como, por ejemplo, emitir consultas.

Una vez publicados estos artefactos, tiene que copiarlos en el entorno de programación de cliente, en el que se utilizan para generar un cliente proxy y clases helper.

Especificación de la dirección de punto final de servicio Web:

La dirección de punto final de servicio Web es el URL que la aplicación cliente debe especificar para acceder a las API de servicios Web. La dirección de punto final se graba al archivo WSDL que exporta para generar un cliente proxy para la aplicación cliente.

Acerca de esta tarea

La dirección de punto final de servicio Web que se va a utilizar depende de la configuración del servidor WebSphere:

- Caso de ejemplo 1. Un solo servidor WebSphere. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y el número de puerto del servidor, por ejemplo, **host1:9080**.
- Caso de ejemplo 2. Un clúster de WebSphere formado por varios servidores. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y puerto del servidor que aloja las API de servicios Web, por ejemplo, **host2:9081**.
- Caso de ejemplo 3. Se utiliza un servidor Web de programa frontal. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y puerto del servidor Web, por ejemplo, **host:80**.

Por omisión, la dirección de punto final de servicio Web adopta la forma de *protocolo*://*sistema_ppal*:*puerto*/*raíz_contexto*/*vía_acceso_fija*. Donde:

- *protocolo*. El protocolo de comunicaciones que se va a utilizar entre la aplicación cliente y el servidor WebSphere. El protocolo por omisión es HTTP. Puede optar por utilizar en su lugar un protocolo HTTPS (HTTP en SSL) más seguro. Se recomienda utilizar HTTPS.

- *sistema_ppal:puerto*. El nombre de sistema principal y número de puerto utilizados para acceder a la máquina que aloja las API de servicios Web. Estos valores varían en función de la configuración del servidor WebSphere, por ejemplo, si la aplicación cliente va a acceder directamente a la aplicación o mediante un programa frontal de servidor Web.
- *raíz_contexto*. Puede seleccionar el valor de raíz de contexto que desee. El valor elegido debe ser, no obstante, único dentro de cada célula de WebSphere. El valor por omisión utiliza un sufijo "sufijo_nodo/cluster" que elimina el riesgo de conflictos de denominación.
- La *vía_acceso_fija* es `/sca/com/ibm/bpe/api/BFMWS` (para la API de Business Flow Manager) o `/sca/com/ibm/task/api/HTMWS` (para la API de Human Task Manager) y no se puede modificar.

La dirección de punto final de servicio Web se especifica inicialmente al configurar el contenedor de procesos de empresa o el contenedor de tareas de usuario:

Procedimiento

1. Inicie la sesión en la consola administrativa con el ID de usuario con derechos de administrador.
2. Seleccione **Aplicaciones → Módulos SCA**.

   Nota: También puede seleccionar **Aplicaciones → Aplicaciones de empresa** para mostrar una lista de todas las aplicaciones de empresa disponibles.
3. Seleccione **BPEContainer** (para el contenedor de procesos de empresa) o **TaskContainer** (para el contenedor de tareas de usuario) de la lista de módulos SCA o aplicaciones.
4. Seleccione **Proporcionar información de URL de punto final HTTP** de la lista de **Propiedades adicionales**.
5. Seleccione uno de los prefijos por omisión de la lista o introduzca un prefijo personalizado. Utilice un prefijo de la lista de prefijos por omisión si las aplicaciones cliente se van a conectar directamente con el servidor de aplicaciones que aloja la API de servicios Web. De lo contrario, especifique un prefijo personalizado.
6. Pulse **Aplicar** para copiar el prefijo seleccionado al módulo SCA.
7. Pulse **Aceptar**. Se guardará la información de URL en el espacio de trabajo.

Resultado

Puede consultar el valor actual en la consola administrativa (por ejemplo, para el contenedor de procesos de empresa: **Aplicaciones de empresa → BPEContainer → Ver descriptor de despliegue**).

En el archivo WSDL exportado, el atributo `location` del elemento `soap:address` contiene la dirección de punto final de servicios Web especificada. Por ejemplo:

```
<wsdl:service name="BFMWSService">
  <wsdl:port name="BFMWSPort" binding="this:BFMWSBinding">
   <soap:address location=
         "https://myserver:9080/WebServicesAPIs/sca/com/ibm/bpe/api/BFMWS"/>
```

Conceptos relacionados

"Adición de seguridad (Java Web services)" en la página 127
Debe proteger las comunicaciones de los servicios Web implementando los mecanismos de seguridad en la aplicación cliente.

Tareas relacionadas

"Adición de la seguridad (.NET)" en la página 136
Puede proteger las comunicaciones de los servicios Web integrando mecanismos de seguridad en la aplicación cliente.

Publicación de archivos WSDL:

Los archivos WSDL (Web Service Definition Language) contienen una descripción detallada de todas las operaciones disponibles con las API de servicios Web. Se dispone de varios archivos WSDL para las API de servicio Web de Business Flow Manager y Human Task Manager. Primero debe publicar estos archivos WSDL, luego copiarlos del entorno WebSphere al entorno de desarrollo, donde se utilizan para generar clientes proxy.

Antes de empezar

Antes de publicar los archivos WSDL, asegúrese de especificar la dirección de punto final de los servicios Web correcta. Se trata del URL que la aplicación cliente utiliza para acceder a las API de servicios Web.

Acerca de esta tarea

Sólo tiene que publicar una vez los archivos WSDL.

Nota: Si tiene el CD del cliente de WebSphere Process Server, puede copiar directamente los archivos al entorno de programación de cliente desde el mismo.

Publicación del WSDL de proceso de empresa:

Utilice la consola administrativa para publicar el archivo WSDL

Procedimiento

1. Inicie la sesión en la consola administrativa con el ID de usuario con derechos de administrador.
2. Seleccione **Aplicaciones → Módulos SCA**

   Nota: También puede seleccionar **Aplicaciones → Aplicaciones de empresa** para mostrar una lista de todas las aplicaciones de empresa disponibles.
3. Seleccione la aplicación **BPEContainer** de la lista de módulos SCA o aplicaciones.
4. Seleccione **Publicar archivos WSDL** de la lista de **Propiedades adicionales**
5. Pulse el archivo zip de la lista.
6. En la ventana Descarga de archivos que se muestra, pulse **Guardar**.
7. Vaya a la carpeta local y pulse **Guardar**.

Resultado

El archivo zip exportado tiene el nombre BPEContainer_WSDLFiles.zip. El archivo zip contiene un archivo WSDL que describe los servicios Web y cualquier referencia de archivos XSD contenidas en el archivo WSDL.

Publicación del WSDL de tareas de usuario:

Utilice la consola administrativa para publicar el archivo WSDL

Procedimiento

1. Inicie la sesión en la consola administrativa con el ID de usuario con derechos de administrador.
2. Seleccione **Aplicaciones → Módulos SCA**

   Nota: También puede seleccionar **Aplicaciones → Aplicaciones de empresa** para mostrar una lista de todas las aplicaciones de empresa disponibles.
3. Seleccione la aplicación **TaskContainer** de la lista de módulos SCA o aplicaciones.
4. Seleccione **Publicar archivos WSDL** de la lista de **Propiedades adicionales**
5. Pulse el archivo zip de la lista.
6. En la ventana Descarga de archivos que se muestra, pulse **Guardar**.
7. Vaya a la carpeta local y pulse **Guardar**.

Resultado

El archivo zip exportado tiene el nombre TaskContainer_WSDLFiles.zip. El archivo zip contiene un archivo WSDL que describe los servicios Web y cualquier referencia de archivos XSD contenidas en el archivo WSDL.

Exportación de objetos de empresa:

Los procesos de empresa y las tareas de usuario tienen interfaces bien definidas que les permiten acceder externamente como servicios Web. Si estas interfaces hacen referencia a los objetos de empresa, tendrá que exportar las definiciones de interfaz y los objetos de empresa al entorno de programación cliente.

Acerca de esta tarea

Este procedimiento debe repetirse para cada objeto de empresa con el que la aplicación cliente tenga que interactuar.

En WebSphere Process Server, los objetos de empresa definen el formato de los mensajes de petición, respuesta y error que interactúan con los procesos de empresa o las tareas de usuario. Estos mensajes también pueden contener definiciones de tipos de datos complejos.

Por ejemplo, para crear e iniciar una tarea de usuario, se deben transferir los siguientes elementos de información a la interfaz de tareas:

- El nombre de la plantilla de la tarea
- El espacio de nombres de la plantilla de tarea.
- Un mensaje de entrada, que contiene los datos de empresa con formato.
- Una envoltura de respuesta para devolver el mensaje de respuesta.
- Un mensaje de error para devolver los errores y las excepciones.

Estos elementos se encapsulan dentro de un solo objeto de empresa. Todas las operaciones de la interfaz de servicio Web se crean como una operación "document/literal con envoltorio". Los parámetros de entrada y salida de estas operaciones se encapsulan en documentos de envoltorio. Otros objetos de empresa definen los formatos de respuesta y error correspondientes.

Para crear e iniciar el proceso de empresa o la tarea de usuario mediante un servicio Web, estos objetos de envoltorio deben estar disponibles para la aplicación de cliente en el extremo del cliente.

Para conseguir esto, se exportan los objetos de empresa del entorno WebSphere como archivos WSDL (Web Service Definition Language) y XSD (XML Schema Definition), se importan las definiciones de tipo de datos al entorno de programación cliente, luego se convierten a clases de ayuda para que las utilice la aplicación cliente.

Procedimiento

1. Inicie el espacio de trabajo de WebSphere Integration Developer si aún no está en ejecución.
2. Seleccione el módulo de biblioteca que contiene los objetos de empresa que se van a exportar. Un módulo de biblioteca es un archivo comprimido que contiene los objetos de empresa necesarios.
3. Exporte el módulo de biblioteca.
4. Copie los archivos exportados al entorno de desarrollo de aplicaciones cliente.

Ejemplo

Supongamos que un proceso de empresa expone la siguiente operación de servicio Web:

```
<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>
```

con los mensajes WSDL definidos como:

```
<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>
```

Los elementos *concretos* definidos por el cliente `types:updateCustomer`, `types:updateCustomerResponse` y `types:updateCustomerFault` deben pasarse a las API de servicios Web y recibirse utilizando los parámetros <xsd:any> en todas las operaciones *genéricas* (call, sendMessage, etc.) realizadas por la aplicación cliente. Estos elementos definidos por el cliente se crean, serializan y deserializan en la aplicación cliente con las clases de ayuda que se generan utilizando los archivos XSD exportados.

Tareas relacionadas

"Creación de clases de ayuda para procesos BPEL (.NET)" en la página 133
Determinadas operaciones de la API de servicios Web requieren que las aplicaciones cliente utilicen los elementos de envoltorio de estilo

"document/literal". Las aplicaciones cliente requieren las clases de ayuda para que les ayuden a generar los elementos de envoltorio necesarios.

Utilización de archivos en el CD del cliente

Como alternativa a la exportación de artefactos desde el entorno de servidor WebSphere, puede copiar los archivos necesarios para generar una aplicación de cliente del CD del cliente de WebSphere Process Server.

En este caso, debe modificar manualmente la dirección de punto final de servicios Web por omisión de la API de Business Flow Manager o de la API de Human Task Manager.

Si la aplicación de cliente es para acceder a las dos API, debe editar la dirección del punto final por omisión para las dos API.

Copia de los archivos desde el CD de cliente:

Los archivos necesarios para acceder a las API de servicios Web están disponibles en el CD del cliente de WebSphere Process Server.

Procedimiento

1. Acceda al CD del cliente y vaya al directorio ProcessChoreographer\client.
2. Copie los archivos necesarios en el entorno de desarrollo de aplicaciones de cliente.

   Para la API de Business Flow Manager, copie:

   BFMWS.wsdl
   Describe los servicios Web disponibles en la API de servicios Web de Business Flow Manager. Este archivo contiene la dirección de punto final.

   BFMIF.wsdl
   Describe los parámetros y el tipo de datos de cada servicio Web en la API de servicios Web de Business Flow Manager.

   BFMIF.xsd
   Describe los tipos de datos utilizados en la API de servicios Web de Business Flow Manager.

   BPCGEN.xsd
   Contiene los tipos de datos que son comunes entre las API de servicio Web de Business Flow Manager y Human Task Manager.

   Para la API de Human Task Manager, copie:

   HTMWS.wsdl
   Describe los servicios Web disponibles en la API de servicios Web de Human Task Manager. Este archivo contiene la dirección de punto final.

   HTMIF.wsdl
   Describe los parámetros y el tipo de datos de cada servicio Web de la API de servicios Web de Human Task Manager.

   HTMIF.xsd
   Describe los tipos de datos utilizados en la API de servicios Web de Human Task Manager.

   BPCGEN.xsd
   Contiene los tipos de datos que son comunes entre las API de servicio Web de Business Flow Manager y Human Task Manager.

Nota: El archivo BPCGen.xsd es común a las dos API.

Después de copiar los archivos, debe modificar manualmente la dirección de punto final de la API de servicios Web en los archivos BFMWS.wsdl o HTMWS.wsdl por la dirección en la que WebSphere Application Server aloja las API de servicios Web.

Cambio manual de la dirección de punto final de servicio Web:

Si copia los archivos del CD de cliente, debe cambiar la dirección de punto final de servicio Web especificada en los archivos WSDL por la del servidor que aloja las API de servicios Web.

Acerca de esta tarea

Puede utilizar la consola administrativa para establecer la dirección de punto final de servicio Web antes de exportar los archivos WSDL. No obstante, si copia los archivos WSDL del CD de cliente de WebSphere, debe modificar manualmente la dirección de punto final de servicio Web por omisión.

La dirección de punto final de servicio Web que se va a utilizar depende de la configuración del servidor WebSphere:

- Caso de ejemplo 1. Hay un solo servidor WebSphere. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y el número de puerto del servidor, por ejemplo, **host1:9080**.
- Caso de ejemplo 2. Un clúster de WebSphere formado por varios servidores. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y puerto del servidor que aloja las API de servicios Web, por ejemplo, **host2:9081**.
- Caso de ejemplo 3. Se utiliza un servidor Web de programa frontal. La dirección de punto final de WebSphere que se va a especificar es el nombre de sistema principal y puerto del servidor Web, por ejemplo, **host:80**.

Modificación del punto final de la API de Business Flow Manager:

Si copia los archivos de la API de Business Flow Manager del CD del cliente de WebSphere Process Server, debe editar manualmente la dirección de punto final por omisión.

Procedimiento

1. Vaya al directorio que contiene los archivos que ha copiado del CD del cliente.
2. Abra el archivo BFMWS.wsdl en un editor de texto o en un editor XML.
3. Localice el elemento `soap:address` (situado junto a la parte inferior del archivo).
4. Modifique el valor del atributo `location` con el URL HTTP del servidor en el que se ejecuta la API de servicio Web. Para hacerlo:
   a. Opcionalmente, sustituya `http` por `https` de modo que utilice el protocolo HTTPS más seguro.
   b. Sustituya *localhost* por el nombre de sistema principal o dirección IP de la dirección de punto final de servidor de la API de servicios Web.
   c. Sustituya *9080* por el número de puerto del servidor de aplicaciones.
   d. Sustituya *BPEContainer_N1_server1* por la raíz de contexto de la aplicación que ejecuta la API de servicios Web. La raíz de contexto por omisión consta de:

- *BPEContainer*. El nombre de la aplicación.
- *N1*. El nombre de nodo.
- *server1*. El nombre de servidor.

e. No modifique la parte fija del URL (`/sca/com/ibm/bpe/api/BFMWS`) .

Por ejemplo, si la aplicación se ejecuta en el servidor **s1.n1.ibm.com** y el servidor acepta peticiones SOAP/HTTP en el puerto **9080**, modifique el elemento `soap:address` del modo siguiente:

```
<soap:address location="http://si.n1.ibm.com:9080/
                  BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

Conceptos relacionados

"Adición de seguridad (Java Web services)" en la página 127
Debe proteger las comunicaciones de los servicios Web implementando los mecanismos de seguridad en la aplicación cliente.

Tareas relacionadas

"Adición de la seguridad (.NET)" en la página 136
Puede proteger las comunicaciones de los servicios Web integrando mecanismos de seguridad en la aplicación cliente.

Modificación del punto final de la API de Human Task Manager:

Si copia los archivos de la API de Human Task Manager del CD del cliente de WebSphere Process Server, debe editar manualmente la dirección de punto final por omisión.

Procedimiento

1. Vaya al directorio que contiene los archivos que ha copiado del CD del cliente.
2. Abra el archivo HTMWS.wsdl en un editor de texto o en un editor XML.
3. Localice el elemento `soap:address` (situado junto a la parte inferior del archivo).
4. Modifique el valor del atributo `location` por la dirección de punto final correcta. Para hacerlo:
   a. Opcionalmente, sustituya `http` por `https` de modo que utilice el protocolo HTTPS más seguro.
   b. Sustituya *localhost* por el nombre de sistema principal o dirección IP de la dirección de punto final de servidor de la API de servicios Web.
   c. Sustituya *9080* por el número de puerto del servidor de aplicaciones.
   d. Sustituya *HTMContainer_N1_server1* por la raíz de contexto de la aplicación que ejecuta la API de servicios Web. La raíz de contexto por omisión consta de:
      - *HTMContainer*. El nombre de la aplicación.
      - *N1*. El nombre de nodo.
      - *server1*. El nombre de servidor.
   e. No modifique la parte fija del URL (`/sca/com/ibm/task/api/HTMWS`) .

   Por ejemplo, si la aplicación se ejecuta en el servidor **s1.n1.ibm.com** y el servidor acepta peticiones SOAP/HTTPS en el puerto **9081**, modifique el elemento `soap:address` del modo siguiente:

```
<soap:address location="https://si.n1.ibm.com:9081/
                  HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

Conceptos relacionados

"Adición de seguridad (Java Web services)" en la página 127
Debe proteger las comunicaciones de los servicios Web implementando los mecanismos de seguridad en la aplicación cliente.

Tareas relacionadas

"Adición de la seguridad (.NET)" en la página 136
Puede proteger las comunicaciones de los servicios Web integrando mecanismos de seguridad en la aplicación cliente.

Desarrollo de aplicaciones cliente en el entorno de servicios Web Java

Puede utilizar cualquier entorno de desarrollo Java compatible con los servicios Web Java para desarrollar aplicaciones cliente para las API de servicios Web.

Generación de un cliente de proxy (Servicios Web Java)

Las aplicaciones cliente del servicio Web Java utilizan un *cliente proxy* para interactuar con las API de servicios Web.

Acerca de esta tarea

Un cliente proxy para los servicios Web Java contiene un número de clases de beans Java que las aplicaciones de cliente invocan para ejecutar las peticiones de servicios Web. El cliente proxy maneja el ensamblaje de los parámetros de servicio en mensajes SOAP, envía los mensajes SOAP al servicio Web a través de HTTP, recibe las respuestas del servicio Web y pasa los datos devueltos a la aplicación de cliente.

Por lo tanto, básicamente un cliente proxy permite que una aplicación invoque un servicio Web como si fuera una función local.

Nota: Sólo tiene que generar una vez el cliente proxy. Todas las aplicaciones de cliente que acceden a la misma API de servicios Web pueden utilizar entonces el mismo cliente proxy.

En el entorno de servicios IBM Web, hay dos modos de generar un cliente proxy:

- Utilizando entornos de desarrollo integrado deRational Application Developer oWebSphere Integration Developer.
- Utilización de la herramienta de línea de mandatos WSDL2Java.

Otros entornos de desarrollo de servicios Web Java generalmente incluyen la herramienta WSDL2Java o los recursos de generación de aplicaciones de cliente.

Utilización de Rational Application Developer para generar un cliente proxy:

El entorno de desarrollo integrado de Rational Application Developer le permite generar un cliente proxy para la aplicación de cliente.

Antes de empezar

Antes de generar un cliente proxy, debe haber exportado anteriormente los archivos WSDL que describen las interfaces de servicios Web de procesos de empresa o de tareas de usuarios desde el entorno de WebSphere (o desde el CD de cliente de WebSphere Process Server) y copiarlos en el entorno de programación del cliente.

Procedimiento

1. Añada el archivo WSDL adecuado al proyecto:
   - Para procesos de empresa:
     a. Descomprima el archivo exportado BPEContainer_*nombre-nodo_nombre-servidor*_WSDLFiles.zip en un directorio temporal.
     b. Importe el subdirectorio META-INF desde el directorio descomprimido BPEContainer_*nombre-nodo_nombre-servidor*.ear/b.jar.
   - Para tareas de usuario:
     a. Descomprima el archivo exportado TaskContainer_*nombre-nodo_nombre-servidor*_WSDLFiles.zip en un directorio temporal.
     b. Importe el subdirectorio META-INF desde el directorio descomprimido TaskContainer_*nombre-nodo_nombre-servidor*.ear/h.jar.

   Se creará una nueva estructura de directorio wsdl y subdirectorio en el proyecto.
2. Modifique las propiedades del asistente de servicio Web:
   a. En Rational Application Developer, seleccione **Preferencias** → **Servicios Web** → **Generación de código** → **Unidad ejecutable IBM WebSphere**
   b. Seleccione la opción **Generar Java a partir de WSDL mediante el estilo sin acomodación**.

   Nota: Si no puede seleccionar la opción **Servicios Web** en el menú **Preferencias**, primero debe habilitar las posibilidades requeridas del modo siguiente: **Ventana** → **Preferencias** → **Entorno de trabajo** → **Posibilidades**. Pulse **Desarrollador de servicios Web** y **Aceptar**. A continuación, vuelva a abrir la ventana Preferencias y cambie la opción **Generación de código**.
3. Seleccione el archivo BFMWS.WSDL o HTMWS.WSDL ubicados en el directorio wsdl de reciente creación.
4. Pulse el botón derecho del ratón y seleccione **Servicios Web** → **Generar cliente**.

   Antes de seguir con los pasos restantes, compruebe que el servidor se haya iniciado.
5. En la ventana Servicios Web, pulse **Siguiente** para aceptar los valores predeterminados.
6. En la ventana Selección de servicios Web, pulse **Siguiente** para aceptar los valores predeterminados.
7. En la ventana Configuración del entorno de cliente:
   a. Pulse **Editar** y cambie la opción Tiempo de ejecución de servicio Web en IBM WebSphere
   b. Cambie la opción Versión de J2EE en 1.4.
   c. Pulse **Aceptar**.
   d. Pulse **Siguiente**.
8. Este paso sólo es necesario si debe generar un cliente de servicios Web que incluya las API de servicios Web de procesos de empresa y de tareas de usuario, ya que hay métodos duplicados en ambos archivos WSDL.
   a. En la ventana Proxy de servicio Web, seleccione Definir correlación personalizada para espacio de nombres con paquete y, a continuación, pulse **Aceptar**.
   b. En la ventana de correlación de cliente de servicio Web de espacio de nombres con paquete, añada los siguientes espacios de nombres y paquete:

Para BFMWS.wsdl:

Espacio de nombres	Paquete
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

For HTMWS.wsdl:

Espacio de nombres	Paquete
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 se le solicita que confirme la sobrescritura, pulse **Sí a todo**.

9. Pulse **Finalizar**.

Resultado

Se generará y añadirá un cliente proxy, formado por un número de clases Java, proxy, locator y helper al proyecto. También se actualizará el descriptor de despliegue.

Utilización de WSDL2Java para generar clientes proxy:

WSDL2Java es una herramienta de línea de mandatos que genera clientes proxy. Los clientes proxy facilitan la programación de aplicaciones cliente.

Antes de empezar

Antes de generar un cliente proxy, debe haber exportado anteriormente los archivos WSDL que describen las API de servicios Web de procesos de empresa o de tareas de usuarios desde el entorno de WebSphere (o el CD del cliente de WebSphere Process Server) y los ha copiado en el entorno de programación.

Acerca de esta tarea

Procedimiento

1. Utilice la herramienta WSDL2Java para generar un cliente proxy: Escriba:
   wsdl2java *opciones vía_acceso_archivo_WSDL*
   Donde:
   - *opciones* incluyen:

-noWrappedOperations (-w)
Inhabilita la detección de operaciones envueltas. Se generan beans Java de mensajes de petición y respuesta.

Nota: Este no es el valor por omisión.

-role (-r)
Especifique el valor **client** para generar los archivos y archivos de enlace del desarrollo en el cliente.

-container (-c)
El contenedor en el cliente que se va a utilizar. Los argumentos válidos incluyen:

client Un contenedor de cliente

ejb Contenedor de EJB (Enterprise JavaBeans).

none Ningún contenedor

web Un contenedor Web

-output (-o)
La carpeta en la que se van a almacenar los archivos generados.

Para obtener una lista completa de parámetros WSDL2Java, utilice el conmutador de línea de mandatos **-help** o la ayuda en línea de la herramienta WSDL2Java de WID/RAD.

- *vía_acceso_archivo_WSDL* es la vía de acceso y nombre de archivo del archivo WSDL que ha exportado desde el entorno de WebSphere o que ha copiado del CD de cliente.

En el ejemplo siguiente se genera un cliente proxy para la API de servicios Web de actividades de tareas de usuario:

```
call wsdl2java.bat -r client -c client -noWrappedOperations
                   -output c:\ws\proxyClient  c:\ws\bin\HTMWS.wsdl
```

2. Incluya los archivos de clase generados en el proyecto.

Creación de clases de ayuda para procesos BPEL (servicios Web Java)

Los objetos de empresa a los que se hace referencia en peticiones de API en concreto (por ejemplo, sendMessage o call) requieren que las aplicaciones cliente utilicen elementos de estilo "document/literal con envoltorio". Las aplicaciones cliente requieren las clases de ayuda para que les ayuden a generar los elementos de envoltorio necesarios.

Antes de empezar

Para crear clases helper, debe haber exportado el archivo WSDL de la API de servicios Web del entorno WebSphere Process Server.

Acerca de esta tarea

Las operaciones call() y sendMessage() de las API de servicios Web permiten la interacción con los procesos BPEL en WebSphere Process Server. El mensaje de entrada de la operación call() espera a que se proporcione la envoltura document/literal del mensaje de entrada del proceso.

Hay varias técnicas posibles para generar clases helper para un proceso BPEL o tarea de usuario, incluidos:

1. Utilice el objeto SoapElement.

   En el entorno Rational Application Developer disponible en WebSphere Integration Developer, el motor de servicio Web da soporte a JAX-RPC 1.1. En JAX-RPC 1.1, el objeto SoapElement amplía un elemento DOM (Document Object Model), por lo tanto, se puede utilizar la API DOM para crear, leer, cargar y guardar mensajes SOAP.

   Por ejemplo, presuponga que el archivo WSDL contiene el siguiente mensaje de entrada para un proceso de flujo de trabajo o tarea de usuario:

   ```
   <xsd:element name="operation1">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="input1" nillable="true" type="xsd:string"/>
             </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
   ```

   El archivo WSDL se crea al desarrollar un módulo de proceso o de tarea de usuario.

   Para crear el mensaje SOAP correspondiente en la aplicación de cliente utilizando la 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);
   ```

   El ejemplo siguiente muestra cómo se crean los parámetros de entrada para la operación sendMessage en la aplicación de cliente:

   ```
   SendMessage inWsend = new SendMessage();
   inWsend.setProcessTemplateName(processtemplatename);
   inWsend.setPortType(porttype);
   inWsend.setOperation(operationname);
   inWsend.set_any(soapmessage);
   ```

2. Utilice la característica de enlace de datos personalizado de WebSphere.

   Esta técnica se describe en los siguientes artículos de developerWorks:

   - Cómo se selecciona una tecnología de correlación personalizada para servicios Web
   - Desarrollo de servicios Web con los SDO de EMF para un esquema XML complejo

   Interoperatividad con patrones y estrategias para los servicios Web basados en documentos

   Soporte de servicios Web para esquema o WSDL que contienen tipos de esquemas XML JAX-RPC 1.0/1.1

Creación de aplicaciones cliente (servicios Web Java)

Las aplicaciones cliente envían las peticiones a las API de servicios Web y reciben las respuestas. Utilizando un cliente proxy para gestionar las comunicaciones y las clases de ayuda para dar formato a los tipos de datos complejos, una aplicación cliente puede invocar los métodos de servicio Web como si fueran funciones locales.

Antes de empezar

Antes de empezar a crear una aplicación cliente, genere el cliente proxy y las clases de ayuda necesarias.

Acerca de esta tarea

Puede desarrollar aplicaciones cliente utilizando una herramienta de desarrollo compatible con los servicios Web, por ejemplo, RAD (IBM Rational Application Developer). Puede generar cualquier tipo de aplicación de servicios Web para llamar a las API de servicios Web.

Procedimiento

1. Cree un nuevo proyecto de aplicación cliente.
2. Genere el cliente proxy y añada las clases de ayuda Java al proyecto.
3. Codifique la aplicación cliente.
4. Genere el proyecto.
5. Ejecute la aplicación cliente.

El ejemplo siguiente muestra cómo se utiliza la API de servicio Web de Business Flow Manager.

```
// crear el proxy
       BFMIFProxy proxy = new BFMIFProxy();
 // preparar los datos de entrada para la operación
      GetProcessTemplate iW = new GetProcessTemplate();
      iW.setIdentifier(nombre_plantilla_proceso);

 // invocar la operación
      GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

 // operación de proceso de la operación
      ProcessTemplateType ptd = oW.getProcessTemplate();
      System.out.println("getName= " + ptd.getName());
      System.out.println("getPtid= " + ptd.getPtid());
```

Tareas relacionadas

"Generación de un cliente de proxy (Servicios Web Java)" en la página 122
Las aplicaciones cliente del servicio Web Java utilizan un *cliente proxy* para interactuar con las API de servicios Web.

"Creación de clases de ayuda para procesos BPEL (servicios Web Java)" en la página 125
Los objetos de empresa a los que se hace referencia en peticiones de API en concreto (por ejemplo, sendMessage o call) requieren que las aplicaciones cliente utilicen elementos de estilo ″document/literal con envoltorio″. Las aplicaciones cliente requieren las clases de ayuda para que les ayuden a generar los elementos de envoltorio necesarios.

Adición de seguridad (Java Web services)

Debe proteger las comunicaciones de los servicios Web implementando los mecanismos de seguridad en la aplicación cliente.

WebSphere Application Server da soporte actualmente a los siguientes mecanismos de seguridad para las API de servicios Web:

- El símbolo de nombre de usuario
- LTPA (Lightweight Third Party Authentication)

Conceptos relacionados

Roles de autorización para procesos de empresa
Un rol es un conjunto de personas que comparten el mismo nivel de autorización. Las acciones que puede realizar en los procesos de empresa dependen de su rol de autorización. Este rol puede ser un rol de J2EE o un rol basado en instancia.

Información relacionada

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

Implementación del símbolo de nombre de usuario:

El mecanismo de seguridad del símbolo de nombre de usuario proporciona los credenciales de nombre de usuario y contraseña.

Acerca de esta tarea

Con el mecanismo de seguridad de símbolo de nombre de usuario, puede optar por implementar varios *manejadores de devolución de llamada*. Dependiendo de su elección.

- Se le solicitará que proporcione un nombre de usuario y una contraseña cada vez que ejecute la aplicación de cliente.
- El nombre de usuario y la contraseña se graban en el descriptor de despliegue.

En cualquiera de los casos, el nombre de usuario y la contraseña proporcionado deben coincidir con los de un rol autorizado en el contenedor de procesos de empresa o contenedor de tareas de usuario correspondiente.

El nombre de usuario y la contraseña quedan encapsulados en el sobre del mensaje de la petición y, por lo tanto, aparecen ″claramente″ en la cabecera del mensaje SOAP. Por lo tanto, se le recomienda encarecidamente que configure la aplicación de cliente para que utilice el protocolo de comunicaciones HTTPS (HTTP en SSL). A continuación, se cifran todas las comunicaciones. Puede seleccionar el protocolo de comunicaciones HTTPS cuando especifique la dirección del URL del punto final de la API de servicio Web.

Para definir un símbolo de nombre de usuario:

Procedimiento

1. Cree un símbolo de seguridad:
   a. Abra el **editor de despliegue** del módulo.
   b. Pulse la pestaña correspondiente a la **extensión WS**.
   c. En las **referencias de servicio**, se pueden listar las siguientes referencias de servicio Web:
      - service/BFMWSService para procesos de empresa
      - service/HTMWSService para tareas de usuario

      Las referencias que se listen depende de si se ha añadido BFMWS.wsdl (para procesos de empresa), HTMWS.wsdl (para tareas de usuario), o ambas al generar el cliente proxy.
   d. Para ambas referencias de servicio:
      1) Seleccione una de las **referencias de servicio**.
      2) Expanda la sección **Configuración del generador de peticiones**.
      3) Expanda la subsección **Símbolo de seguridad**.
      4) Pulse **Añadir**. Se abrirá la ventana Símbolo de seguridad.
      5) En el campo **Nombre**, escriba un nombre para el nuevo símbolo de seguridad: **UserNameTokenBFM** o **UserNameTokenHTM** .

6) En la lista desplegable **Tipo de símbolo**, seleccione **Username** (el campo **Nombre local** se rellena automáticamente con un valor predeterminado).
7) Deje el campo **URI** en blanco. No es necesario ningún valor de URI para un símbolo de nombre de usuario.
8) Pulse **Aceptar**.

2. Cree un generador de símbolos:
   a. Abra el **editor de despliegue** del módulo.
   b. Pulse la pestaña correspondiente al **enlace WS**.
   c. En las **referencias de servicio**, se listan las mismas referencias de servicio Web que en el paso anterior:
      - service/BFMWSService para procesos de empresa
      - service/HTMWSService para tareas de usuario
   d. Para ambas referencias de servicio:
      1) Seleccione una de las **referencias de servicio**.
      2) Expanda la sección **Configuración de enlace del generador de peticiones de seguridad**.
      3) Expanda la subsección **Generador de símbolos**.
      4) Pulse **Añadir**. Se abrirá la ventana Generador de símbolos.
      5) En el campo **Nombre**, escriba un nombre para el nuevo generador de símbolos, como ″UserNameTokenGeneratorBFM″ o ″UserNameTokenGeneratorHTM″.
      6) En el campo **Clase de generador de símbolos**, asegúrese de que se seleccione la siguiente clase de generador de símbolos: **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**.
      7) En la lista desplegable **Símbolo de seguridad**, seleccione el símbolo de seguridad adecuado creado anteriormente.
      8) Marque el recuadro de selección **Utilizar tipo de valor**.
      9) En el campo **Tipo de valor**, seleccione **Símbolo Username** (el campo **Nombre local** se rellena automáticamente para reflejar lo que haya elegido en **Símbolo Username**.)
      10) En el campo **Manejador de devolución de llamada** escriba ″com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler″ (que solicita el nombre de usuario y la contraseña cuando ejecuta la aplicación de cliente) o ″com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler″.
      11) Si selecciona **NonPromptCallbackHandler**, debe especificar un nombre de usuario y una contraseña válidas en el campo correspondiente del descriptor de despliegue.
      12) Pulse **Aceptar**.

Tareas relacionadas

“Especificación de la dirección de punto final de servicio Web” en la página 114
La dirección de punto final de servicio Web es el URL que la aplicación cliente debe especificar para acceder a las API de servicios Web. La dirección de punto final se graba al archivo WSDL que exporta para generar un cliente proxy para la aplicación cliente.

Información relacionada

IBM WebSphere Developer Technical Journal: Web services security with WebSphere Application Server V6

Implementación del mecanismo de seguridad LTPA:

El mecanismo de seguridad LTPA (Lightweight Third Party Authentication) se puede utilizar cuando se ejecuta la aplicación de cliente en un contexto de seguridad establecido anteriormente.

Acerca de esta tarea

El mecanismo de seguridad LTPA sólo está disponible si la aplicación de cliente se ejecuta en un entorno protegido en el que se ha establecido ya un contexto de seguridad. Por ejemplo, si la aplicación de cliente se ejecuta en un contenedor EJB (Enterprise JavaBeans), el cliente EJB debe iniciar la sesión para poder invocar la aplicación de cliente. A continuación, se establece un contexto de seguridad. Si la aplicación de cliente EJB invoca un servicio Web, el manejador de devoluciones de llamada LTPA recupera el símbolo LTPA del contexto de seguridad y lo añade al mensaje de petición SOAP. En el extremo del servidor, el símbolo LTPA lo maneja el mecanismo LTPA.

Para implementar el mecanismo de seguridad LTPA:

Procedimiento

1. En el entorno Rational Application Developer disponible en WebSphere Integration Developer, seleccione **Enlace WS → Configuración de enlace del generador de peticiones de seguridad → Generador de símbolos**.
2. Cree un símbolo de seguridad:
   a. Abra el **editor de despliegue** del módulo.
   b. Pulse la pestaña correspondiente a la **extensión WS**.
   c. En las **referencias de servicio**, se pueden listar las siguientes **referencias de servicio Web**:
      - service/BFMWSService para procesos de empresa
      - service/HTMWSService para tareas de usuario

      Las referencias que se listen depende de si se ha añadido BFMWS.wsdl (para procesos de empresa), HTMWS.wsdl (para tareas de usuario), o ambas al generar el cliente proxy.
   d. Para ambas referencias de servicio:
      1) Seleccione una de las **referencias de servicio**.
      2) Expanda la sección **Configuración del generador de peticiones**.
      3) Expanda la subsección **Símbolo de seguridad**.
      4) Pulse **Añadir**. Se abrirá la ventana Símbolo de seguridad.
      5) En el campo **Nombre**, escriba un nombre para el nuevo símbolo de seguridad: **LTPATokenBFM** o **LTPATokenHTM**.
      6) En la lista desplegable **Tipo de símbolo**, seleccione **LTPAToken** (los campos **URI** y **Nombre local** se rellenan automáticamente con valores predeterminados).
      7) Pulse **Aceptar**.
3. Cree un generador de símbolos:
   a. Abra el **editor de despliegue** del módulo.
   b. Pulse la pestaña correspondiente al **enlace WS**.
   c. En las **referencias de servicio**, se listan las mismas referencias de servicio Web que en el paso anterior:
      - service/BFMWSService para procesos de empresa

- service/HTMWSService para tareas de usuario

d. Para ambas referencias de servicio:
   1) Seleccione una de las **referencias de servicio**.
   2) Expanda la sección **Configuración de enlace del generador de peticiones de seguridad**.
   3) Expanda la subsección **Generador de símbolos**.
   4) Pulse **Añadir**. Se abrirá la ventana Generador de símbolos.
   5) En el campo **Nombre**, escriba un nombre para el nuevo generador de símbolos, como ″LTPATokenGeneratorBFM″ o ″LTPATokenGeneratorHTM″.
   6) En el campo **Clase de generador de símbolos**, asegúrese de que se seleccione la siguiente clase de generador de símbolos: **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**.
   7) En la lista desplegable **Símbolo de seguridad**, seleccione el símbolo de seguridad adecuado creado anteriormente.
   8) Marque el recuadro de selección **Utilizar tipo de valor**.
   9) En el campo **Tipo de valor**, seleccione **LTPAToken** (los campos **URI** y **Nombre local** se rellena automáticamente para reflejar lo que haya elegido en **Símbolo LTPA**).
   10) En el campo **Manejador de devolución de llamada**, escriba ″com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler″.
   11) Pulse **Aceptar**.

Resultado

Durante la ejecución, **LTPATokenCallbackHandler** recupera el símbolo LTPA del contexto de seguridad existente y lo añade al mensaje de petición SOAP.

Adición de soporte de transacciones (servicios Web Java)

Las aplicaciones del cliente de servicios Web Java se pueden configurar de modo que el proceso de las peticiones del extremo del servidor participen en la transacción del cliente pasando un contexto de la aplicación de cliente como parte de la petición de servicio. Este soporte de transacciones atómicas se define en la especificación WS-AT (Web Services-Atomic Transaction).

Acerca de esta tarea

WebSphere Application Server ejecuta cada petición de la API de servicios Web como una transacción atómica. Las aplicaciones de cliente se pueden configurar para utilizar el soporte de transacciones de uno de los modos siguientes:

- Participar en la transacción. El proceso de peticiones del extremo del servidor se efectúa dentro del contexto de la transacción de la aplicación de cliente. Luego, si el servidor se encuentra con un problema cuando se está ejecutando la petición de la API de servicios Web y se retrotrae, la petición de la aplicación cliente también se retrotrae.
- No utilice el soporte de transacciones. WebSphere Application Server continúa creando una transacción nueva en la que se ejecuta la petición pero no se efectúa el proceso de peticiones en el extremo del servidor con el contexto de transacción de aplicaciones de cliente.

Desarrollo de aplicaciones cliente en el entorno .NET

Microsoft .NET ofrece un entorno de desarrollo completo en el que conectar aplicaciones mendiante servicios Web.

Generación de un cliente proxy (.NET)

Las aplicaciones cliente .NET utilizan un *cliente proxy* para interactuar con las API de servicio Web. Un cliente proxy protege las aplicaciones cliente de la complejidad del protocolo de mensajería de servicio Web.

Antes de empezar

Para crear un cliente proxy, primero debe exportar varios archivos WSDL del entorno WebSphere y copiarlos al entorno de programación cliente.

Nota: Si tiene el CD del cliente de WebSphere Process Server, puede copiar los archivos desde allí.

Acerca de esta tarea

Un cliente proxy se compone de un conjunto de clases de bean de C#. Cada clase contiene todos los métodos y objetos expuestos por un solo servicio Web. Los métodos de servicio manejan el ensamblado de los parámetros en mensajes SOAP completos, envían los mensajes SOAP al servicio Web en HTTP, reciben las respuestas del servicio Web y gestionan los datos devueltos.

Nota: Sólo tiene que generar una vez el cliente proxy. Todas las aplicaciones cliente que acceden a las API de servicios Web pueden utilizar entonces el mismo cliente proxy.

Procedimiento

1. Utilice el mandato WSDL para generar un cliente proxy: Escriba:

   wsdl *opciones vía_acceso_archivo_WSDL*

   Donde:

   - *opciones* incluyen:

     /language
     Permite especificar el lenguaje utilizado para crear la clase proxy. Toma por omisión C#. También puede especificar **VB** (Visual Basic), **JS** (JScript) o **VJS** (Visual J#) de argumento de lenguaje.

     /output
     El nombre del archivo de salida, con el sufijo adecuado. Por ejemplo, proxy.cs

     /protocol
     El protocolo implementado en la clase proxy. Toma por omisión el valor **SOAP**.

     Si desea una lista completa de los parámetros **WSDL.exe**, utilice el conmutador de la línea de mandatos **/?** o bien consulte la ayuda en línea de la herramienta WSDL en Visual Studio.

   - *vía_acceso_archivo_WSDL* es la vía de acceso y nombre de archivo del archivo WSDL que ha exportado desde el entorno de WebSphere o que ha copiado del CD de cliente.

   En el ejemplo siguiente se genera un cliente proxy para la API de servicios Web Human Task Manager:

```
wsdl /language:cs /output:proxyclient.cs c:\ws\bin\HTMWS.wsdl
```

2. Compile el cliente proxy como un archivo DLL (Dynamic Link Library).

Creación de clases de ayuda para procesos BPEL (.NET)

Determinadas operaciones de la API de servicios Web requieren que las aplicaciones cliente utilicen los elementos de envoltorio de estilo ″document/literal″. Las aplicaciones cliente requieren las clases de ayuda para que les ayuden a generar los elementos de envoltorio necesarios.

Antes de empezar

Para crear clases helper, debe haber exportado el archivo WSDL de la API de servicios Web del entorno WebSphere Process Server.

Acerca de esta tarea

Las operaciones call() y sendMessage() de las API de servicios Web provocan que los procesos BPEL se inicien en WebSphere Process Server. El mensaje de entrada de la operación call() espera a que se proporcione la envoltura document/literal del mensaje de entrada del proceso BPEL. Para generar los beans y las clases necesarios para el proceso BPEL, copie el elemento `<wsdl:types>` en un archivo XSD nuevo, a continuación, utilice la herramienta xsd.exe para generar clases helper.

Procedimiento

1. Si todavía no lo ha hecho, exporte el archivo WSDL de la interfaz de proceso BPEL desde WebSphere Integration Developer.
2. Abra el archivo WSDL en un editor de texto o un editor XML.
3. Copie el contenido de todos los elementos hijo del elemento `<wsdl:types>` y péguelo a un nuevo archivo XSD esqueleto.
4. Ejecute la herramienta `xsd.exe` en el archivo XSD:

   call xsd.exe file.xsd /classes /o

   Donde:

   file.xsd
   Archivo de definición de esquema XML que se va a convertir.

   /classes (/c)
   Genere las clases de ayuda que corresponden al contenido del archivo o archivos XSD especificados.

   /output (/o)
   Especifique el directorio de salida de los archivos generados. Si se omite este directorio, el valor por omisión es el directorio actual.

   Por ejemplo:

   call xsd.exe ProcessCustomer.xsd /classes /output:c:\temp
5. Añada el archivo de clase que se genera a la aplicación cliente. Si utiliza Visual Studio, por ejemplo, puede hacer esto utilizando la opción de menú **Proyecto → Agregar elemento existente**.

Si el archivo ProcessCustomer.wsdl contiene lo siguiente:

```
<?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>
```

El archivo XSD producido contendrá:

```
<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>
```

Información relacionada

Documentación de Microsoft para la herramienta de definición de esquemas XML (XSD.EXE)

Creación de aplicaciones cliente (.NET)

Las aplicaciones cliente envían las peticiones a las API de servicios Web y reciben las respuestas. Utilizando un cliente proxy para gestionar las comunicaciones y las clases de ayuda para dar formato a los tipos de datos complejos, una aplicación cliente puede invocar los métodos de servicio Web como si fueran funciones locales.

Antes de empezar

Antes de empezar a crear una aplicación cliente, genere el cliente proxy y las clases de ayuda necesarias.

Acerca de esta tarea

Puede desarrollar las aplicaciones de cliente .NET utilizando cualquier herramienta de desarrollo compatible con .NET, por ejemplo, Visual Studio .NET. Puede crear cualquier tipo de aplicación .NET para llamar a las API de servicio Web genéricas.

Procedimiento

1. Cree un nuevo proyecto de aplicación cliente. Por ejemplo, cree una **aplicación Windows WinFX** en Visual Studio.
2. En las opciones del proyecto, añada una referencia al archivo DLL (Dynamic Link Library) del cliente proxy. Añada todas las clases helper que contienen definiciones de objetos de empresa al proyecto. En Visual Studio, por ejemplo, puede hacerlo utilizando la opción **Proyecto → Agregar elemento existente**.
3. Cree un objeto cliente proxy. Por ejemplo:

```
HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService service =
    new HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService();
```

4. Declare los tipos de datos de objetos de empresa utilizados en los mensajes que se van a enviar al servicio Web o recibir. Por ejemplo:

```
HTMClient.HTMReference.TKIID id = new HTMClient.HTMReference.TKIID();

ClipBG bg = new ClipBG();
Clip clip = new Clip();
```

5. Llame a las funciones de servicio Web específicas y especifique los parámetros necesarios. Por ejemplo, para crear e iniciar una tarea de usuario:

```
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. Las tareas y procesos remotos se identifican con los ID permanentes (`id` en el ejemplo del paso anterior). Por ejemplo, para reclamar una tarea de usuario creada previamente:

```
HTMClient.HTMReference.claimTask claim = new HTMClient.HTMReference.claimTask();
claim.inputTask = id;
```

Tareas relacionadas

"Generación de un cliente proxy (.NET)" en la página 132
Las aplicaciones cliente .NET utilizan un *cliente proxy* para interactuar con las

API de servicio Web. Un cliente proxy protege las aplicaciones cliente de la complejidad del protocolo de mensajería de servicio Web.

"Creación de clases de ayuda para procesos BPEL (.NET)" en la página 133
Determinadas operaciones de la API de servicios Web requieren que las aplicaciones cliente utilicen los elementos de envoltorio de estilo "document/literal". Las aplicaciones cliente requieren las clases de ayuda para que les ayuden a generar los elementos de envoltorio necesarios.

Adición de la seguridad (.NET)

Puede proteger las comunicaciones de los servicios Web integrando mecanismos de seguridad en la aplicación cliente.

Acerca de esta tarea

Estos mecanismos de seguridad pueden incluir el símbolo de userName (nombre de usuario y contraseña) o los símbolos binarios personalizados y los símbolos de seguridad basados en XML.

Procedimiento

1. Baje e instale WSE (Web Services Enhancements) 2.0 SP3 para Microsoft .NET. Este producto está disponible en:

   http://www.microsoft.com/downloads/details.aspx?familyid=1ba1f631-c3e7-420a-bc1e-ef18bab66122&displaylang=en
2. Modifique el código cliente de proxy generado como se detalla a continuación.

   Cambie:

```
public class Export1_MyMicroflowHttpService : System.Web.Services.Protocols.SoapHttpClientProtocol {
```

   Por:

```
public class Export1_MyMicroflowHttpService : Microsoft.Web.Services2.WebServicesClientProtocol {
```

   Nota: Estas modificaciones se pierden si vuelve a generar el cliente proxy ejecutando la herramienta WSDL.exe.
3. Modifique el código de aplicación cliente añadiendo las siguientes líneas al principio del archivo:

```
using System.Web.Services.Protocols;
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Security.Tokens;
...
```

4. Añada código para implementar el mecanismo de seguridad deseado. Por ejemplo, el código siguiente añade una protección de nombre de usuario y contraseña:

```
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);
```

Consulta sobre los objetos de procesos de empresa y relativos a tareas

Puede utilizar las API de servicios Web para consultar los objetos de procesos de empresa y relativos a tareas de la base de datos de Business Process Choreographer con el fin de recuperar propiedades específicas de estos objetos.

Acerca de esta tarea

La base de datos de Business Process Choreographer almacena datos de plantilla (modelo) y de instancia (tiempo de ejecución) para gestionar procesos de empresa y tareas.

Mediante las API de servicios Web, las aplicaciones cliente pueden emitir consultas para recuperar información de la base de datos sobre los procesos y tareas de empresa.

Las aplicaciones cliente pueden emitir una consulta específica para recuperar una propiedad concreta de un objeto. Habitualmente, se pueden guardar las consultas que utiliza. Luego la aplicación cliente puede recuperar y utilizar estas consultas almacenadas.

Consultas sobre los objetos de procesos de empresa y relativos a tareas

Utilice la interfaz de consulta de las API de servicios Web para obtener información sobre los procesos de empresa y tareas.

Las aplicaciones de cliente utilizan una sintaxis de tipo SQL para consultar la base de datos.

Ejemplos de servicios Web Java

```
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);
```

La información recuperada de la base de datos se devuelve mediante las API de servicios Web como un *conjunto de resultados de consulta*.

Por ejemplo:

```
QueryResultSetType queryResultSet = queryResponse1.queryResultSet;
if (queryResultSet != null) {
   Console.WriteLine("--> QueryResultSetType");
   Console.WriteLine(" . size= " + queryResultSet.size);
   Console.WriteLine(" . numberColumns= " + queryResultSet.numberColumns);
   string indent = " . ";

   // -- la información de la columna de consulta
   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>");
   }

   // - the query result values
   string[][] result = queryResultSet.result;
   if (result !=null) {
      Console.WriteLine();
      Console.WriteLine("= . result size= " + result.Length);
      for (int i = 0; i &lt; result.Length; i++) {
        Console.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 función query devuelve objetos según la autorización del llamante. El conjunto de resultados de consulta sólo contiene las propiedades de aquellos objetos que el llamante está autorizado a ver.

Se proporcionan vistas de bases de datos predefinidas para que se consulten las propiedades de objeto. Para plantillas de proceso, la función de consulta tiene la sintaxis siguiente:

```
ProcessTemplateData[] queryProcessTemplates
                       (java.lang.String whereClause,
                        java.lang.String orderByClause,
                        java.lang.Integer threshold,
                        java.util.TimeZone timezone);
```

Para plantillas de tarea, la función de consulta tiene la sintaxis siguiente:

```
TaskTemplate[] queryTaskTemplates
                (java.lang.String whereClause,
                 java.lang.String orderByClause,
                 java.lang.Integer threshold,
                 java.util.TimeZone timezone);
```

Para los demás objetos de proceso de empresa, la función de consulta tiene la sintaxis siguiente:

```
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);
```

La interfaz de consulta también contiene un método queryAll. Puede utilizar este método para recuperar todos los datos relevantes acerca de un objeto como, por

ejemplo, a fines de supervisión. El llamante del método queryAll debe tener uno de los siguientes roles de J2EE (Java 2 Platform, Enterprise Edition): BPESystemAdministrator, BPESystemMonitor, TaskSystemAdministrator o TaskSystemMonitor. No se aplica la comprobación de autorización utilizando el elemento de trabajo correspondiente del objeto.

Ejemplo para .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);
      }
```

Parámetros de consulta:

Todas las consultas deben especificar un número de cláusulas SQL y parámetros.

Una consulta se compone de:
- Cláusula select
- Cláusula where
- Cláusula order-by
- Parámetro skip-tuples
- Parámetro threshold
- Parámetro time-zone

Vistas predefinidas para las consultas sobre los objetos de proceso de empresa y tarea de usuario

Se proporcionan vistas de bases de datos predefinidas para objetos de proceso de empresa y de tarea de usuario.

Utilice estas vistas cuando consulte datos de referencia para dichos objetos. Cuando utilice estas vistas, no es necesario que añada predicados de unión para columnas de vista de forma explícita, estas construcciones se añaden automáticamente. Puede utilizar la función de consulta de las API de servicios Web para consultar estos datos.

Gestión de consultas almacenadas

Las consultas almacenadas proporcionan un modo de guardar las consultas que se ejecutan con frecuencia. La consulta almacenada puede ser una consulta que está disponible para todos los usuarios (consulta pública) o una consulta que pertenece a un usuario específico (consulta privada).

Acerca de esta tarea

Una consulta almacenada es una consulta que se almacena en la base de datos y se identifica con un nombre. Una consulta almacenada privada y pública puede tener el mismo nombre; las consultas privadas almacenadas de distintos propietarios también pueden tener el mismo nombre.

Puede tener consultas almacenadas para objetos de proceso de empresa, objetos de tarea, o una combinación de estos dos tipos de objetos.

Gestión de consultas almacenadas públicas
Las consultas almacenadas públicas las crea el administrador del sistema. Estas consultas están disponibles para todos los usuarios.

Gestión de consultas almacenadas privadas de otros usuarios
Cualquier usuario puede crear consultas privadas. Estas consultas sólo están disponibles para el propietario de la consulta y el administrador del sistema.

Cómo trabajar con las consultas almacenadas privadas
Si usted no es el administrador del sistema, puede crear, ejecutar y suprimir sus propias consultas almacenadas privadas. También puede utilizar las consultas almacenadas públicas que el administrador del sistema ha creado.

Desarrollo de aplicaciones de cliente JMS

Puede desarrollar aplicaciones cliente que acceden a las aplicaciones de proceso de empresa mediante la API de JMS (Java Messaging Service).

Acerca de esta tarea

Introducción a JMS

WebSphere Process Server Versión 6.1 da soporte a la mensajería asíncrona, basada en la interfaz de programación JMS (Java Messaging Service), como método de comunicación.

JMS proporciona una manera común para que los clientes Java (aplicaciones de cliente o aplicaciones J2EE) creen, envíen, reciban y lean peticiones como mensajes JMS.

JMS es una interfaz asíncrona basada en mensajes que:

- Utiliza la **mensajería de punto a punto o de publicación/suscripción**. Las infraestructuras basadas en mensajes pueden pasar información a otras aplicaciones sin que estas la soliciten explícitamente. La misma información se puede entregar a muchos suscriptores en paralelo. La interfaz JMS de Business Process Choreographer sólo da soporte a la mensajería de punto a punto.
- Ofrece **independencia de ritmo**. Las infraestructuras JMS funcionan asíncronamente, pero también pueden simular una modalidad síncrona de petición/respuesta. Esto permite que los sistemas de origen y de destino funcionen simultáneamente sin tener que esperarse entre sí. Esta capacidad es

extremadamente útil para Business Process Choreographer, ya que proporciona la capacidad de interactuar asíncronamente con procesos de empresa de larga duración.

- Da soporte a las **transacciones**. Las transacciones permiten que las aplicaciones de cliente gestionen grupos de mensajes enviados o recibidos como si se tratara una sola unidad atómica. Las transacciones JMS se ejecutan en la transacción del servidor. Para la interfaz JMS de Business Process Choreographer, normalmente se envía y recibe un único mensaje para cada transacción.
- **Garantiza la entrega de la información**. Las infraestructuras JMS pueden gestionar mensajes en modalidad transaccional y asegurar la entrega de los mismos (aunque sin garantía alguna de puntualidad en la entrega). Para Business Process Choreographer, esta posibilidad fiable para la entrega de mensajes es particularmente importante porque trata con procesos de empresa.
- Asegura la **interoperatividad entre infraestructuras heterogéneas**. Las aplicaciones de origen y de destino pueden funcionar en entornos heterogéneos sin tener que gestionar problemas de comunicación ni de ejecución relacionados con sus respectivas infraestructuras.
- **Efectúa intercambios más fluídos**. La conmutación a la modalidad de mensajes permite un intercambio de información más precisa.

Requisitos para los procesos de empresa

Los procesos de empresa desarrollados con WebSphere Integration Developer para que se ejecuten en Business Process Choreographer deben ajustarse a normas específicas para que se pueda acceder a los mismos desde la API de JMS.

Los requisitos son:

1. Las interfaces de los procesos de empresa se deben definir con el estilo ″documento/literal con envoltura″ definido en la API Java para la especificación RPC basada en XML, JAX-RPC 1.1. Este es el estilo por omisión para todos los procesos de empresa y tareas de usuario desarrollados con WebSphere Integration Developer.
2. Los mensajes de error que exponen los procesos de empresa y las tareas de usuario para las operaciones de servicios Web deben constar de una parte de mensaje WSDL individual definida con el elemento de esquema XML. Por ejemplo:

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

   Información relacionada

   Página de descargas de la API Java para RPC basado en XML, JAX-RPC

   ¿Qué estilo de WSDL debe utilizar?

Acceso a la interfaz JMS

Para enviar y recibir mensajes a través de la interfaz JMS, una aplicación primero debe crear una conexión a BPC.cellname.Bus, crear una sesión y, a continuación, generar productores y consumidores de mensajes.

Acerca de esta tarea

El servidor de procesos acepta mensajes JMS (Java Message Service) que sigan el paradigma de punto a punto. Una aplicación que envíe o reciba mensajes JMS debe efectuar las acciones siguientes.

En el ejemplo siguiente se da por supuesto que el cliente JMS se ejecuta en un entorno gestionado (EJB, cliente de aplicaciones o contenedor de cliente Web). Si desea ejecutar el cliente JMS en un entorno J2SE, consulte el documento ″IBM Client for JMS on J2SE with IBM WebSphere Application Server″ en http://www-1.ibm.com/support/docview.wss?uid=swg24012804.

Procedimiento

1. Cree una conexión con BPC.*cellname*.Bus. No existe una fábrica de conexiones preconfigurada para las peticiones de una aplicación de cliente: una aplicación de cliente puede utilizar ReplyConnectionFactory de la API de JMS o crear su propia fábrica de conexiones, en cuyo caso puede utilizar la búsqueda JNDI (Java Naming and Directory Interface) para recuperar la fábrica de conexiones. El nombre de búsqueda JNDI debe ser el mismo que el nombre especificado al configurar la cola de peticiones externas de Business Process Choreographer. En el ejemplo siguiente se da por supuesto que la aplicación de cliente crea su propia fábrica de conexiones llamada ″jms/clientCF″.

   ```
   // Obtener el contexto inicial por omisión de JNDI.
   Context initialContext = new InitialContext();

   // Buscar la fábrica de conexiones.
   // Crear una fábrica de conexiones que se conecte al bus BPC.
   // Darle un nombre, por ejemplo, "jms/clientCF".
   // Configurar también un alias de autenticación adecuado.
   ConnectionFactory connectionFactory =
               (ConnectionFactory)initialcontext.lookup("jms/clientCF");

   // Crear la conexión.
   Connection connection = connectionFactory.createConnection();
   ```

2. Cree una sesión para que puedan crearse los productores y consumidores de mensajen.

   ```
   // Crear una sesión de transacción mediante autoreconocimiento.
   Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
   ```

3. Cree un productor de mensajes para enviar mensajes. El nombre de búsqueda JNDI debe ser el mismo que el nombre especificado al configurar la cola de peticiones externas de Business Process Choreographer.

   ```
   // Buscar el destino de la cola de entrada de Business Process Choreographer
   // a la que enviar mensajes.
   Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");

   // Crear un productor de mensajes.
   MessageProducer producer = session.createProducer(sendQueue);
   ```

4. Cree un consumidor de mensajes para recibir respuestas. El nombre de búsqueda JNDI del destino de respuesta puede especificar un destino definido por el usuario, pero también puede especificar el destino de respuesta predeterminado (definido por Business Process Choreographer) `jms/BFMJMSReplyQueue`. En ambos casos, el destino de respuesta debe encontrarse en `BPC.<cellname>.Bus`.

   ```
   // Buscar el destino de la cola de respuesta.
   Queue  replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

   // Crear un consumidor de mensajes.
   MessageConsumer consumer = session.createConsumer(replyQueue);
   ```

5. Envíe un mensaje.

   ```
   // Iniciar la conexión.
   connection.start();

   // Crear un mensaje (consulte las descripciones de tareas para ver ejemplos)
   // y enviarlo.
   ```

```
// Este método está definido en otro lugar...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Establecer cabecera JMS obligatoria.
// targetFunctionName es el nombre de operación de la API de JMS
// (por ejemplo, getProcessTemplate, sendMessage)
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Establecer la cola de respuesta; esto es obligatorio si replyQueue
// no es la cola predeterminada (como sucede en este ejemplo).
requestMessage.setJMSReplyTo(replyQueue);

// Enviar el mensaje.
producer.send(requestMessage);

// Obtener el ID de mensaje.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. Reciba la respuesta.

```
// Recibir el mensaje de respuesta y analizar la respuesta.
TextMessage  replyMessage = (TextMessage) consumer.receive();

// Obtener la carga útil.
String payload = replyMessage.getText();

session.commit();
```

7. Cierre la conexión y libere los recursos.

```
 // Limpieza final; liberar los recursos.
session.close();
connection.close();
```

Nota: No es necesario cerrar la conexión después de cada transacción. Una vez se haya iniciado una conexión, se puede intercambiar cualquier número de mensajes de petición y respuesta antes de cerrar la conexión. En el ejemplo se muestra un caso simple con una sola llamada en un único método de empresa.

Estructura de un mensaje JMS de Business Process Choreographer

La cabecera y cuerpo de los mensajes JMS deben tener una estructura predefinida.

Un mensaje JMS (Java Message Service) consiste en:

- Una cabecera de mensaje para la identificación de mensajes y la información de direccionamiento.
- El cuerpo (carga útil) del mensaje que almacena el contenido.

Business Process Choreographer sólo da soporte a formatos de mensajes de texto.

Cabecera del mensaje

JMS permite que los clientes accedan a un número de campos de cabecera del mensaje.

Un cliente JMS de Business Process Choreographer puede establecer los siguientes campos de cabecera:

- **JMSReplyTo**

El destino al que se envía una respuesta a la petición. Si este campo no se especifica en el mensaje de petición, la respuesta se envía al destino de respuesta predeterminado de la interfaz de exportación (una exportación es una representación de interfaz de cliente de un componente de proceso de empresa). Este destino se puede obtener mediante la utilización de `initialContext.lookup("jms/BFMJMSReplyQueue");`

- **TargetFunctionName**

  El nombre de la operación WSDL, por ejemplo, "queryProcessTemplates". Este campo siempre debe establecerse. Tenga en cuenta que TargetFunctionName especifica la operación de la interfaz de mensajes JMS genérica que se describe aquí. Esta operación no debe confundirse con otras proporcionadas por procesos o tareas determinadas que se pueden invocar indirectamente, por ejemplo, mediante operaciones **call** o **sendMessage**.

Un cliente de Business Process Choreographer también puede acceder a los siguientes campos de cabecera:

- **JMSMessageID**

  Identifica un mensaje de manera exclusiva. El proveedor JMS lo establece al enviar el mensaje. Si el cliente establece el JMSMessageID antes de enviar el mensaje, el proveedor JMS lo sobrescribe. Si se necesita el ID del mensaje para finalidades de autenticación, el cliente puede recuperar el JMSMessageID antes de enviar el mensaje.
- **JMSCorrelationID**

  Enlaza mensajes. No establezca este campo. Un mensaje de respuesta de Business Process Choreographer contiene el JMSMessageID del mensaje de respuesta.

Cada mensaje de respuesta contiene los siguientes campos de cabecera JMS:

- **IsBusinessException**

  "False" para mensajes de salida WSDL o "true" para mensajes de error WSDL.

No se devuelven excepciones ServiceRuntimeExceptions para aplicaciones de cliente asíncronas. Cuando se produce una excepción grave durante el proceso de un mensaje de petición JMS, tiene como resultado un error en tiempo de ejecución que provoca la retrotracción de la transacción que procesa este mensaje de petición. A continuación, el mensaje de petición JMS se vuelve a entregar. Si la anomalía se produce al principio, durante el proceso del mensaje como parte de la exportación de SCA (por ejemplo, al deserializar el mensaje), se efectúan reintentos hasta el número máximo de entregas con error especificado por el destino de recepción de la exportación de SCA. Después de alcanzarse el número máximo de entregas con error, el mensaje de petición se añade al destino de excepción del sistema del bus de Business Process Choreographer. No obstante, si el error se produce durante el proceso real de la petición efectuada por el componente SCA de Business Flow Manager, la infraestructura de gestión de sucesos con error de WebSphere Process Server gestionará el mensaje de petición con error, es decir, puede terminar en la base de datos de gestión de sucesos con error si los reintentos no resuelven la situación de excepción.

Cuerpo del mensaje

El cuerpo de mensaje JMS es una serie que contiene un documento XML que representa el elemento de envoltorio document/literal de la operación.

Un ejemplo simple de un cuerpo de mensaje de petición válido es:

```
<?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>
```

Autorización para representaciones JMS

Para autorizar la utilización de la interfaz JMS, debe habilitar los valores de seguridad en WebSphere Application Server.

Cuando el contenedor de proceso de empresa está instalado, el rol **JMSAPIUser** debe correlacionarse con un ID de usuario. Este ID de usuario se utiliza para emitir todas las peticiones de la API de JMS. Por ejemplo, si **JMSAPIUser** se correlaciona con ″Usuario A″, todas las peticiones de la API de JMS aparecen en el motor de procesos para que se origine desde ″Usuario A″.

Se deben asignar las autorizaciones siguientes al rol **JMSAPIUser**:

Solicitud	Autorización necesaria
forceTerminate	Administrador de procesos
sendEvent	Propietario de actividad potencial o administrador de procesos

Nota: Para el resto de peticiones, no se necesitan autorizaciones especiales.

Se otorga una autorización especial a una persona con el rol de administrador de procesos de empresa. Un administrador de procesos de empresa es un rol especial; es diferente del administrador de procesos o de una instancia de proceso. Un administrador de procesos de empresa tiene todos los privilegios.

No puede suprimir el ID de usuario del iniciador del proceso desde el registro de usuarios mientras existe la instancia de proceso. Si lo hace, la navegación de este proceso no podrá continuar. Recibirá la excepción siguiente en el archivo de anotaciones cronológicas del sistema:

```
El ID no es exclusivo para:
<ID de usuario>
```

Visión general de la API de JMS

La interfaz de mensajes JMS (desde ahora llamada la ″API de JMS″) permite desarrollar aplicaciones de cliente que acceden asíncronamente a los procesos de empresa en ejecución en el entorno de Business Process Choreographer.

La API de JMS permite que las aplicaciones de cliente interactúen asíncronamente con los microflujos y procesos de larga ejecución.

La API de JMS expone la misma interfaz que la API de servicios Web, con las excepciones siguientes:

- Con la API de servicios Web, la operación `call` solo puede utilizarse para invocar microflujos. Sin embargo, con la API de JMS, la operación `call` puede utilizarse para invocar microflujos y procesos de larga duración.
- Las operaciones siguientes no se exponen a través de la API de JMS:
  - La operación `callAsync` (junto con sus operaciones de devolución de llamada asociadas).

– Las operaciones `completeAndClaimSuccessor` y `getParticipatingTask`.

Ejemplo - ejecución de un proceso de larga duración

Para que una aplicación de cliente genérica funcione con procesos de larga duración, la secuencia de pasos a realizar es:

1. Configure el entorno JMS, tal como se describe en "Acceso a la interfaz JMS" en la página 141.
2. Obtenga una lista de definiciones de procesos instalados:
   - Envíe `queryProcessTemplates`.
   - Esto devuelve una lista de objetos **ProcessTemplate**.
3. Obtenga una lista de actividades de inicio (de recepción o selección con `createInstance="yes"`):
   - Envíe `getStartActivities`.
   - Esto devuelve una lista de objetos **InboundOperationTemplate**.
4. Cree un mensaje de entrada. Este es específico de entorno y puede que necesite utilizar artefactos predesplegados específicos de proceso.
5. Cree una instancia de proceso:
   - Emita un `sendMessage`.

   Con la API de JMS, también puede utilizar la operación `call` para interactuar con operaciones de petición-respuesta de larga duración proporcionadas por un proceso de empresa. Esta operación devuelve el resultado o error de la operación en el destino de respuesta especificado, incluso después de un largo período de tiempo. Por tanto, si utiliza la operación `call`, no es necesario utilizar las operaciones `query` y `getOutputMessage` para obtener el mensaje de salida o error del proceso.
6. Opcionalmente, obtenga mensajes de salida desde instancias de proceso repitiendo los pasos siguientes:
   - Emita `query` para obtener el estado de finalizado de la instancia de proceso.
   - Emita `getOutputMessage` .
7. Opcionalmente, trabaje con operaciones adicionales expuestas por el proceso:
   - `getWaitingActivities` o `getActiveEventHandlers` para obtener una lista de objetos **InboundOperationTemplate**.
   - Cree mensajes de entrada.
   - Envíe mensajes con `sendMessage`.
8. Opcionalmente, obtenga y establezca propiedades personalizadas definidas en el proceso o actividades contenidas con `getCustomProperties` y `setCustomProperties`.
9. Opcionalmente, deje de trabajar con una instancia de proceso:
   - Envíe `delete` y `terminate` para dejar de trabajar con el proceso de larga duración.

Desarrollo de aplicaciones JMS

Las aplicaciones de cliente JMS tienen que estar desarrolladas en Java mediante el entorno Java 2 Enterprise Edition (J2EE).

Acerca de esta tarea

Las aplicaciones de cliente JMS intercambian mensajes de petición y respuesta con la API de JMS. Para crear un mensaje de petición, la aplicación de cliente

cumplimenta un cuerpo de mensaje TextMessage JMS con un elemento XML que representa la envoltura document/literal de la operación correspondiente.

Copia de artefactos

Se pueden copiar varios artefactos desde el entorno WebSphere para ayudar a crear las aplicaciones de cliente JMS.

La utilización de estos artefactos sólo es obligatoria si se utiliza BOXMLSerializer para crear el cuerpo de mensaje JMS.

Hay dos modos de obtener estos artefactos:

- Publicar y exportarlas desde el entorno de WebSphere Process Server.

  Para WebSphere Process Server 6.1, todos los artefactos de cliente se encuentran en el directorio *raíz_instalación*\ProcessChoreographer\client. Para la API de JMS, estos artefactos son:

  BFMIF.wsdl

  BFMIF.xsd

  BPCGen.xsd
- Copiar los archivos desde el CD del cliente de WebSphere Process Server.

Publicación de artefactos desde el entorno de servidor:

Para ayudar a desarrollar aplicaciones de cliente que accedan a la API de JMS, puede publicar varios artefactos desde el entorno de servidor de WebSphere.

Acerca de esta tarea

Para WebSphere Process Server 6.1, todos los artefactos de cliente se encuentran en el directorio *inicio_was*\ProcessChoreographer\client. Para la API de JMS, estos artefactos son:

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

Una vez publicados estos artefactos, cópielos en el entorno de programación de cliente.

Copia de los archivos desde el CD de cliente:

Los archivos necesarios para acceder a las API de JMS están disponibles en el CD del cliente de WebSphere Process Server.

Procedimiento

1. Acceda al CD del cliente y vaya al directorio `ProcessChoreographer\client`.
2. Copie los archivos necesarios en el entorno de desarrollo de aplicaciones de cliente.

   Para WebSphere Process Server 6.1, todos los artefactos de cliente se encuentran en el directorio \ProcessChoreographer\client. Para la API de JMS, estos artefactos son:

   BFMIF.wsdl

   BFMIF.xsd

   BPCGen.xsd

Comprobación del mensaje de respuesta para excepciones empresarial

Las aplicaciones de cliente JMS tienen que comprobar si existen excepciones empresariales en la cabecera de mensaje de todos los mensajes de respuesta.

Acerca de esta tarea

Una aplicación de cliente JMS tiene que comprobar, en primer lugar, la propiedad **IsBusinessException** en la cabecera del mensaje de respuesta.

Por ejemplo:

```
// recibir mensaje de respuesta
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject().receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse es un error de empresa
  // las api pueden terminar con processFaultMsg
  // la api de llamada también con businessFaultMsg
 }
else {
  // strResponse es el mensaje de salida
}
```

Desarrollo de aplicaciones Web para procesos de empresa y tareas de usuario, utilizando componentes JSF

Business Process Choreographer proporciona varios componentes JSF (JavaServer Faces). Puede ampliar e integrar estos componentes para añadir funcionalidad de procesos de empresa y de tareas de usuario a las aplicaciones Web.

Acerca de esta tarea

Puede utilizar WebSphere Integration Developer para construir la aplicación Web.

Procedimiento

1. Cree un proyecto dinámico y cambie las propiedades de las Características de proyecto Web para incluir los componentes base de JSF.

   Para obtener más información sobre cómo crear un proyecto Web, vaya al centro de información de WebSphere Integration Developer.
2. Añada los archivos JAR (Java Archive) de Business Process Choreographer Explorer de prerrequisito.

   Añada los archivos siguientes al directorio WEB-INF/lib del proyecto:

   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar
   - bpcjsfcomponents.jar

   Si va a desplegar la aplicación Web en un servidor remoto, añada también los archivos siguientes. Estos archivos se necesitan para el acceso remoto a las API de Business Process Choreographer.

   - bpe137650.jar
   - task137650.jar

   En WebSphere Process Server, todos estos archivos se encuentran en el directorio siguiente:

- En los sistemas Windows: *install_root*\ProcessChoreographer\client
- En los sistemas UNIX, Linux y i5/OS: *raíz_instalación*/ProcessChoreographer/client

3. Añada las referencias de EJB que necesita al descriptor de despliegue de aplicaciones Web, el archivo 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. Añada los componentes JSF de Business Process Choreographer Explorer a la aplicación JSF.
   a. Añada a los archivos JSP (JavaServer Pages) las referencias de bibliotecas de códigos que necesita para las aplicaciones. Habitualmente, se necesitan las bibliotecas de códigos JSF y HTML, y la biblioteca de códigos que necesitan los componentes de 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. Añada un código `<f:view>` al cuerpo de la página JSP y un código `<h:form>` al código `<f:view>`.
   c. Añada los componentes de JSF a los archivos JSP.

      En función de la aplicación, añada a los archivos JSP los componentes List, Details, CommandBar o Message. Puede añadir varias instancias de cada componente.
   d. Configure los beans gestionados en el archivo de configuración de JSF.

      Por omisión, el archivo de configuración es el archivo faces-config.xml. Este archivo está en el directorio WEB-INF de la aplicación Web.

      En función del componente que añada al archivo JSP, también tiene que añadir las referencias a la consulta y otros objetos de reiniciador al archivo de configuración de JSF. Para garantizar un correcto manejo de los errores, también es necesario definir un bean de error y un destino de navegación para la página de error en el archivo de configuración 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>
La página general de error.
</description>
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

En situaciones de error que desencadenan la página de errores, se establece la excepción en el bean de error.

e. Implemente el código personalizado que tiene que dar soporte a los componentes JSF.

5. Despliegue la aplicación.

   Si va a desplegar la aplicación en un entorno de Network Deployment, cambie los nombres JNDI (Java Naming and Directory Interface) del recurso de destino por valores donde las API de Business Flow Manager y del Gestor de tareas de usuario puedan encontrarse en la célula.

   - Si los contenedores de procesos de empresa están configurados en otro servidor de la misma célula gestionada, los nombres tienen la estructura siguiente:

     célula/nodos/*nombre_nodo*/servidores/*nombre_servidor*
     /com/ibm/bpe/api/BusinessManagerHome
     célula/nodos/*nombre_nodo*/servidores/*nombre_servidor*
     /com/ibm/task/api/HumanTaskManagerHome

   - Si los contenedores de procesos de empresa están configurados en un clúster de la misma célula, los nombres tienen la estructura siguiente:

     célula/clústeres/*nombre_clúster*/com/ibm/bpe/api/BusinessFlowManagerHome
     célula/clústeres/*nombre_clúster*/com/ibm/task/api/HumanTaskManagerHome

   Correlacione las referencias de EJB a nombres JNDI o añada manualmente las referencias al archivo ibm-web-bnd.xmi.

   La tabla siguiente lista los enlaces de referencia y sus correlaciones por omisión.

Tabla 33. Correlación de los enlaces de referencia a nombres JNDI

Enlace de referencia	Nombre JNDI	Comentarios
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean de sesión remota
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean de sesión local
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean de sesión remota
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean de sesión local

Resultado

La aplicación Web desplegada contiene la funcionalidad proporcionada por los componentes de Business Process Choreographer Explorer.

Qué hacer a continuación

Si va a utilizar JSP personalizadas para los mensajes de proceso y de tarea, debe correlacionar los módulos Web utilizados para desplegar las JSP en los mismos servidores con los que esté correlacionado el cliente JSF personalizado.

Conceptos relacionados

"Manejo de errores en componentes JSF" en la página 152
Los componentes JSF (JavaServer Faces) utilizan un bean gestionado definido previamente, BPCError, para el manejo de errores. En situaciones de error que desencadenan la página de errores, se establece la excepción en el bean de error.

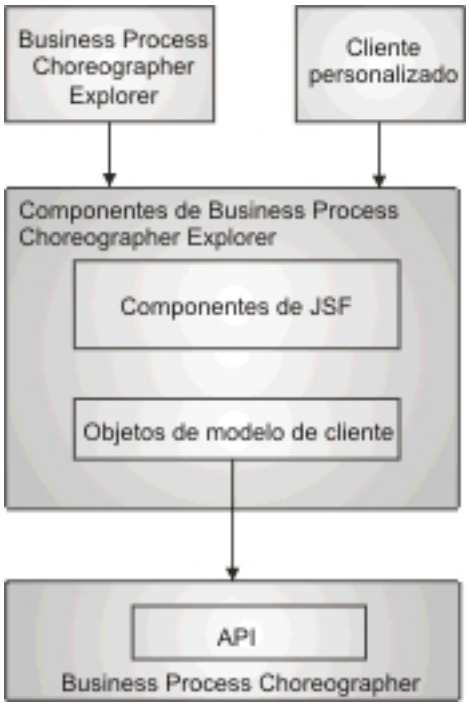
Tareas relacionadas

"Acceso a la interfaz remota del bean de sesión" en la página 22
Una aplicación de cliente EJB accede a la interfaz remota del bean de sesión mediante la interfaz inicial remota del bean.

Componentes de Business Process Choreographer Explorer

Los componentes de Business Process Choreographer Explorer son un conjunto de elementos configurables y reutilizables que están basados en la tecnología JSF (JavaServer Faces). Puede incorporar estos elementos en aplicaciones Web. Las aplicaciones Web pueden acceder a las aplicaciones instaladas de procesos de empresa y tareas de usuario.

Los componentes constan de un conjunto de componentes JSF y un conjunto de objetos de modelo de cliente. La relación de los componentes con Business Process Choreographer, Business Process Choreographer Explorer y otros clientes personalizados se muestran en la figura siguiente.



Componentes de JSF

Los componentes de Business Process Choreographer Explorer incluyen los siguientes componentes de JSF. Debe incorporar estos componentes JSF en los archivos JSP (JavaServer Pages) cuando cree aplicaciones Web para trabajar con procesos de empresa y tareas de usuario.

- Componente List

  El componente List muestra una lista de objetos de aplicación en una tabla, por ejemplo, tareas, actividades, instancias de proceso, plantillas de proceso, elementos de trabajo o escaladas. Este componente tiene un manejador de lista asociado.
- Componente Details

  El componente Details muestra las propiedades de las tareas, elementos de trabajo, actividades, instancias de proceso y plantillas de proceso. Este componente tiene un manejador de detalles asociado.
- Componente CommandBar

  El componente CommandBar muestra una barra con botones. Estos botones representan mandatos que operan en la vista de detalles del objeto o en los objetos seleccionados en una lista. Un manejador de listas o un manejador de detalles proporciona estos objetos.
- Componente Message

  El componente Message muestra un mensaje que puede contener un SDO (Service Data Object) o un tipo simple.

Objetos de modelo de cliente

Los objetos de modelo de cliente se utilizan con los componentes JSF. Los objetos implementan algunas de las interfaces de la API de Business Process Choreographer subyacente y acomodan el objeto original. Los objetos de modelo de cliente proporcionan soporte de idioma nacional para las etiquetas y convertidores para algunas propiedades.

Manejo de errores en componentes JSF

Los componentes JSF (JavaServer Faces) utilizan un bean gestionado definido previamente, BPCError, para el manejo de errores. En situaciones de error que desencadenan la página de errores, se establece la excepción en el bean de error.

Este bean implementa la interfaz com.ibm.bpc.clientcore.util.ErrorBean. La página de errores se muestra en estas situaciones:

- Si se produce un error durante la ejecución de una consulta que se define para un manejador de listas y el método execute de un mandato genera el error como un error de ClientException
- Si el método execute de un mandato genera un error de ClientException y este error no es un error de ErrorsInCommandException ni tampoco implementa la interfaz CommandBarMessage
- Si se muestra una mensaje de error en el componente y sigue el hiperenlace del mensaje

Está disponible una implementación por omisión de la interfaz com.ibm.bpc.clientcore.util.ErrorBeanImpl.

La interfaz se define como se detalla a continuación:

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * Esta llamada al método setter permite que se pase un entorno local
     * y la excepción. Esto permite que los métodos
     * getExceptionMessage devuelvan series adaptadas
```

```
*
     */
    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();

    /*
     * Este método devuelve el mensaje de excepción
     * concatenado recursivamente con los mensajes de todas
     * las excepciones anidadas.
     */
    public String getAllExceptionMessages();

    /*
     * Este método devuelve la pila de excepciones
     * concatenada recursivamente con las pilas de todas
     * las excepciones anidadas.
     */
    public String getAllExceptionStacks();
}
```

Conceptos relacionados

"Manejo de errores del componente List" en la página 158
Cuando se utiliza el componente List para visualizar listas en la aplicación JSF, puede aprovechar las funciones de manejo de errores proporcionadas por la clase com.ibm.bpe.jsf.handler.BPCListHandler.

Convertidores y etiquetas por omisión para objetos de modelo de cliente

Los objetos de modelo de cliente implementan las interfaces correspondientes de la API de Business Process Choreographer.

Los componentes List y Details funcionan en cualquier bean. Puede visualizar todas las propiedades de un bean. Sin embargo, si desea establecer los convertidores y las etiquetas que se utilizan para las propiedades de un bean, debe utilizar el código `column` para el componente List, o bien el código `property` para el componente Details. En lugar de establecer los convertidores y las etiquetas, puede definir convertidores y etiquetas por omisión para las propiedades definiendo los siguientes métodos estáticos. Puede definir los siguientes métodos estáticos:

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

En la tabla siguiente se muestran los objetos de modelo de cliente que implementan las clases de API de Business Flow Manager y del Gestor de tareas de usuario correspondientes y proporcionan etiquetas y convertidores por omisión para sus propiedades. Esta acomodación de las interfaces proporciona etiquetas sensibles al entorno local y convertidores de un conjunto de propiedades. La tabla siguiente muestra la correlación de las interfaces de Business Process Choreographer con los objetos de modelo de cliente correspondientes.

Tabla 34. Cómo las interfaces de Business Process Choreographer se correlacionan con objetos de modelo de cliente

Interfaz de Business Process Choreographer	Clase de objeto de modelo de cliente
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

Adición del componente List a una aplicación JSF

Utilice el componente List de Business Process Choreographer Explorer para visualizar una lista de objetos de modelo de cliente, por ejemplo, instancias de proceso de empresa o instancias de tarea.

Procedimiento

1. Añada el componente List al archivo JSP (JavaServer Pages).

   Añada el código `bpe:list` al código `h:form`. El código `bpe:list` debe incluir un atributo model. Añada los códigos `bpe:column` al código `bpe:list` para añadir las propiedades de los objetos que han de aparecer en cada una de las filas de la lista.

   El siguiente ejemplo muestra cómo añadir un componente List para visualizar instancias de tarea.

   ```
   <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>
   ```

   El atributo model hace referencia a un bean gestionado, TaskPool. El bean gestionado proporciona la lista de objetos Java que la lista reitera y luego visualiza en filas individuales.

2. Configure el bean gestionado al que se hace referencia en el código `bpe:list`.

   Para el componente List, este bean gestionado debe ser una instancia de la clase com.ibm.bpe.jsf.handler.BPCListHandler.

   El siguiente ejemplo muestra cómo añadir el bean gestionado TaskPool al archivo de configuración.

   ```
   <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>
```

En el ejemplo se muestra que TaskPool tiene dos propiedades configurables: query y type. El valor de la propiedad query hace referencia a otro bean gestionado, TaskPoolQuery. El valor de la propiedad type especifica la clase de bean, cuyas propiedades se muestran en las columnas de la lista visualizada. La instancia de consulta asociada también puede tener un tipo de propiedad. Si se especifica un tipo de propiedad, debe ser el mismo que el tipo especificado para el manejador de listas.

Puede añadir cualquier tipo de lógica de consulta a la aplicación JSF siempre que el resultado de la consulta pueda representarse como una lista de beans de tipo fuerte. Por ejemplo, TaskPoolQuery se implementa mediante una lista de objetos com.ibm.task.clientmodel.bean.TaskInstanceBean.

3. Añada el código personalizado del bean gestionado al que hace referencia el manejador de listas.

   El siguiente ejemplo muestra cómo añadir código personalizado al bean gestionado TaskPool.

```
public class TaskPoolQuery implements Query {

  public List execute throws ClientException {

    // Examinar el archivo faces-config para un bean gestionado "htmConnection".
    //
    FacesContext ctx = FacesContext.getCurrentInstance();
    Application  app = ctx.getApplication();
    ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
    htmConnection = (HTMConnection) htmVb.getValue(ctx);
    HumanTaskManagerService taskService =
         htmConnection.getHumanTaskManagerService();

    // A continuación, llamar al método de consulta actual en el servicio de
    // Gestor de tareas de usuario.
    //
    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 );

// Transforma cada fila de QueryResultSet en beans de instancia de tarea.
 for (int i = 0; i < entries; i++) {
   result.next();
   array.add( new TaskInstanceBean( result, connection ));
 }
 return array ;
  }
}
```

El bean TaskPoolQuery consulta las propiedades de los objetos Java. Este bean debe implementar la interfaz com.ibm.bpc.clientcore.Query. Cuando el manejador de listas renueva su contenido, llama al método execute de la consulta. La llamada devuelve una lista de objetos Java. El método getType debe devolver el nombre de clase de los objetos Java devueltos.

Resultado

La aplicación JSF contiene ahora una página JavaServer que muestra las propiedades de la lista de objetos solicitada, por ejemplo, el estado, clase, propietario y originador de las instancias de tarea que están disponibles para el usuario.

Conceptos relacionados

"Información de huso horario específica del usuario" en la página 158
Los componentes JSF (JavaServer Faces) proporcionan un programa de utilidad para gestionar información de huso horario específica del usuario en el componente List.

Referencia relacionada

"Componente List: definiciones de código" en la página 159
El componente List de Business Process Choreographer Explorer muestra una lista de objetos en una tabla, por ejemplo, tareas, actividades, instancias de proceso, plantillas de proceso, elementos de trabajo y escaladas.

Cómo se procesan las listas

Todas las instancias del componente List tienen asociada una instancia de la clase com.ibm.bpe.jsf.handler.BPCListHandler.

Este manejador de listas realiza un seguimiento de los elementos seleccionados y proporciona una mecanismo de notificación para asociar las entradas de lista a las páginas de detalles para los distintos tipos de elementos. El manejador de la lista se enlaza con el componente List mediante el atributo **model** del código `bpe:list`.

El mecanismo de notificación del manejador de listas se implementa utilizando la interfaz com.ibm.bpe.jsf.handler.ItemListener. Puede registrar las implementaciones de esta interfaz en el archivo de configuración de la aplicación JSF (JavaServer Faces).

Se desencadena la notificación cuando se pulsa un enlace de la lista. Los enlaces se representan para todas las columnas para los que se ha establecido el atributo

action. El valor del atributo **action** es un destino de navegación JSF o un método de acción JSF que devuelve un destino de navegación JSF.

La clase BPCListHandler también proporciona un método refreshList. Puede utilizar este método en enlaces de método JSF para implementar un control de interfaz de usuario para ejecutar de nuevo la consulta.

Implementaciones de consulta

Puede utilizar el manejador de listas para mostrar todos los tipos de objetos y sus propiedades. El contenido de la lista que se muestra depende de la lista de objetos que la implementación de la interfaz com.ibm.bpc.clientcore.Query que está configurada para el manejador de listas devuelve. Puede establecer la consulta mediante programación utilizando el método setQuery de la clase BPCListHandler o puede configurarla en los archivos de configuración JSF de la aplicación.

Puede ejecutar las consultas sólo con las API de Business Process Choreographer, pero también con cualquier otra fuente de información que sea accesible desde la aplicación, por ejemplo, un sistema de gestión de contenido o una base de datos. El único requisito es que el resultado de la consulta se devuelva como una `java.util.List` de objetos del método execute.

El tipo de los objetos devueltos debe garantizar que están disponibles los métodos getter adecuados para todas las propiedades que se muestran en las columnas de la lista para la que se define la consulta. Para asegurarse de que el tipo del objeto que se devuelve se ajusta a las definiciones de lista, puede establecer el valor de la propiedad type en la instancia de BPCListHandler que se define en el archivo de configuración de Faces con el nombre de clase plenamente cualificado de los objetos devueltos. Puede devolver este nombre en la llamada a getType de la implementación de consulta. Durante la ejecución, el manejador de listas comprueba que los tipos de objetos son conforme a las definiciones.

Para correlacionar los mensajes de error con entradas específicas de una lista, los objetos devueltos por la consulta deben implementar un método con la signatura `public Object getID()`.

Convertidores y etiquetas por omisión

Los elementos devueltos por una consulta deben ser beans y sus clases deben coincidir con la clase especificada como el tipo en la definición de la clase BPCListHandler o de la interfaz com.ibm.bpc.clientcore.Query. Además, el componente List comprueba si la clase del elemento o una superclase implementa los métodos siguientes:

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

Si estos métodos se definen para los beans, el componente List utiliza la etiqueta como etiqueta predeterminada para la lista y SimpleConverter como convertidor predeterminado para la propiedad. Puede sobrescribir estos valores con los atributos **label** y **converterID** del código `bpe:list`. Si desea obtener más información, consulte el Javadoc para la interfaz SimpleConverter y la clase ColumnTag.

Información de huso horario específica del usuario

Los componentes JSF (JavaServer Faces) proporcionan un programa de utilidad para gestionar información de huso horario específica del usuario en el componente List.

La clase BPCListHandler utiliza la interfaz com.ibm.bpc.clientcore.util.User para obtener información sobre el huso horario y el entorno local de cada usuario. El componente List espera que la implementación de la interfaz se configure con **user** como el nombre de bean gestionado en el archivo de configuración JSF (JavaServer Faces). Si faltara esta entrada en el archivo de configuración, se devolvería el huso horario en el que se ejecuta WebSphere Process Server.

La interfaz com.ibm.bpc.clientcore.util.User se define de la siguiente manera:

```
public interface User {

    /**
     * El entorno local utilizado por el cliente del usuario.
     * @return El entorno local.
     */
    public Locale getLocale();
   /**
    * El huso horario utilizado por el cliente del usuario.
    * @return El huso horario.
    */
    public TimeZone getTimeZone();

   /**
    * El nombre del usuario.
    * @return el nombre del usuario.
    */
    public String getName();
}
```

Manejo de errores del componente List

Cuando se utiliza el componente List para visualizar listas en la aplicación JSF, puede aprovechar las funciones de manejo de errores proporcionadas por la clase com.ibm.bpe.jsf.handler.BPCListHandler.

Errores que se producen cuando se ejecutan consultas o se ejecutan mandatos

Si se produce un error durante la ejecución de una consulta, la clase BPCListHandler distingue entre errores que se han producido por derechos de acceso insuficientes y otras excepciones. Para obtener errores por derechos de acceso insuficientes, el parámetro **rootCause** de la excepción ClientException que el método execute de la consulta genera debe ser una excepción com.ibm.bpe.api.EngineNotAuthorizedException o com.ibm.task.api.NotAuthorizedException. El componente List muestra el mensaje de error en lugar del resultado de la consulta.

Si el error no se ha producido por derechos de acceso insuficientes, la clase BPCListHandler pasa el objeto de excepción a la implementación de la interfaz com.ibm.bpc.clientcore.util.ErrorBean que se define mediante la clave BPCError en el archivo de configuración de la aplicación JSF. Cuando se establece la excepción, se llama al destino de navegación de errores.

Errores que se producen al trabajar con elementos que se muestran en la lista

La clase BPCListHandler implementa la interfaz com.ibm.bpe.jsf.handler.ErrorHandler. Puede proporcionar información sobre estos errores con el parámetro map de tipo java.util.Map del método setErrors. Esta correlación contiene identificadores como claves y las excepciones como valores. Los identificadores deben ser los valores devueltos por el método getID del objeto que ha producido el error. Si se establece la correlación y cualquiera de los ID coincide con cualquiera de los elementos mostrados en la lista, el manejador de listas añade automáticamente una columna que contiene el mensaje de error a la lista.

Para impedir mensajes de error obsoletos en la lista, restablezca la correlación de errores. En las siguientes situaciones, se restablece automáticamente la correlación:

- Se llama a la clase BPCListHandler del método refreshList.
- Se establece una nueva consulta en la clase BPCListHandler.
- Se utiliza el componente CommandBar para desencadenar acciones sobre los elementos de la lista. El componente CommandBar utiliza este mecanismo como uno de los métodos de manejo de errores.

Conceptos relacionados

"Manejo de errores en componentes JSF" en la página 152
Los componentes JSF (JavaServer Faces) utilizan un bean gestionado definido previamente, BPCError, para el manejo de errores. En situaciones de error que desencadenan la página de errores, se establece la excepción en el bean de error.

Componente List: definiciones de código

El componente List de Business Process Choreographer Explorer muestra una lista de objetos en una tabla, por ejemplo, tareas, actividades, instancias de proceso, plantillas de proceso, elementos de trabajo y escaladas.

El componente List consta de los códigos de componente JSF: `bpe:list` y `bpe:column`. El código `bpe:column` es un subelemento del código `bpe:list`.

Clase de componente

com.ibm.bpe.jsf.component.ListComponent

Sintaxis de ejemplo

```
<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>
```

Atributos de código

El cuerpo del código `bpe:list` sólo puede contener códigos `bpe:column`. Cuando se representa la tabla, el componente List itera por la lista de objetos de aplicación y representa todas las columnas para cada objeto.

Tabla 35. Atributos de bpe:list

Atributo	Necesario	Descripción
buttonStyleClass	no	Clase de estilo de hoja de estilos en cascada (CSS) para representar los botones del área de pie de página.
cellStyleClass	no	Clase de estilo CSS para representar celdas de tabla individuales.
checkbox	no	Determina si se representa el recuadro de selección para seleccionar varios elementos. El atributo tiene el valor `true` o `false`. Si el valor se establece en `true`, se representa la columna del recuadro de selección.
headerStyleClass	no	Clase de estilo CSS para representar la cabecera de tabla.
model	sí	Enlace de valor para un bean gestionado de la clase com.ibm.bpe.jsf.handler.BPCListHandler.
rows	no	Número de filas que se muestran en una página. Si el número de elementos sobrepasa el número de filas, se muestran botones de paginación al final de la tabla. Las expresiones de valores no están soportadas para este atributo.
rowClasses	no	Clase de estilo CSS para representar las filas de la tabla.
selectAll	no	Si este atributo se establece en `true`, todos los elementos de la lista se seleccionan por omisión.
styleClass	no	Clase de estilo CSS para representar la tabla general que contiene títulos, filas y botones de paginación.

Tabla 36. Atributos de bpe:column

Atributo	Necesario	Descripción
action	no	Si se especifica el atributo, se representa un enlace en la columna. Un método de acción JavaServer Faces o el destino de navegación Faces se desencadena cuando se pulsa este enlace. Un método de acción JavaServer Faces tiene la signatura siguiente: String method().
converterID	no	El ID de convertidor Faces que se utiliza para convertir el valor de propiedad. Si no se establece este atributo, se utiliza cualquier ID de convertidor Faces que se proporcione por el modelo para esta propiedad.

Tabla 36. Atributos de bpe:column (continuación)

Atributo	Necesario	Descripción
label	no	Un literal o expresión de enlace de valor que se utiliza como etiqueta para la cabecera de la columna o la celda de la fila de cabecera de la tabla. Si no se establece este atributo, se utiliza cualquier etiqueta que se proporcione por el modelo para esta propiedad.
name	sí	Nombre de la propiedad que se visualiza en esta columna.

Adición del componente Details a una aplicación JSF

Utilice el componente Details de Business Process Choreographer Explorer para visualizar las propiedades de tareas, elementos de trabajo, actividades, instancias de proceso y plantillas de proceso.

Procedimiento

1. Añada el componente Details al archivo JSP (JavaServer Pages).

   Añada el código `bpe:details` al código `<h:form>`. El código `bpe:details` debe contener un atributo **model**. Puede añadir propiedades al componente Details con el código `bpe:property`.

   En el ejemplo siguiente se muestra cómo añadir un componente Details para visualizar algunas de las propiedades de una instancia de tarea.

   ```
   <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>
   ```

   El atributo **model** hace referencia a un bean gestionado, TaskInstanceDetails. El bean proporciona las propiedades del objeto Java.

2. Configure el bean gestionado al que se hace referencia en el código `bpe:details`.

   Para el componente Details, este bean gestionado debe ser una instancia de la clase com.ibm.bpe.jsf.handler.BPCDetailsHandler. Esta clase de manejador reinicia un objeto Java y expone sus propiedades públicas al componente de detalles.

   En el ejemplo siguiente se muestra cómo añadir el bean gestionado TaskInstanceDetails al archivo de configuración.

   ```
   <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>
```

En el ejemplo se muestra que el bean TaskInstanceDetails tiene una propiedad `type` configurable. El valor de la propiedad type especifica la clase de bean (com.ibm.task.clientmodel.bean.TaskInstanceBean), cuyas propiedades se muestran en las filas de los detalles visualizados. La clase de bean puede ser cualquier clase de JavaBeans. Si el bean proporciona por omisión etiquetas de propiedades y convertidor, el convertidor y la etiqueta se utilizan para la representación, de igual manera que para el componente List.

Resultado

La aplicación JSF contiene ahora una página JavaServer que muestra los detalles del objeto especificado, por ejemplo, los detalles de una instancia de tarea.

Referencia relacionada

"Componente Details: definiciones de código"
El componente Details de Business Process Choreographer Explorer muestra las propiedades de tareas, elementos de trabajo, actividades, instancias de proceso y plantillas de proceso.

Componente Details: definiciones de código

El componente Details de Business Process Choreographer Explorer muestra las propiedades de tareas, elementos de trabajo, actividades, instancias de proceso y plantillas de proceso.

El componente Details consta de los códigos de componente JSF: `bpe:details` y `bpe:property`. El código `bpe:property` es un subelemento del código `bpe:details`.

Clase de componente

com.ibm.bpe.jsf.component.DetailsComponent

Sintaxis de ejemplo

```
<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>
```

Atributos de código

Utilice códigos `bpe:property` para especificar tanto el subconjunto de atributos que se muestran como el orden en que se muestran estos atributos. Si el código de detalles no contiene códigos de atributo, representa todos los atributos disponibles del objeto modelo.

Tabla 37. Atributos de bpe:details

Atributo	Necesario	Descripción
columnClasses	no	Lista de clases de estilo de la hoja de estilo en cascada (CSS), separada por comas, para representar columnas.

Tabla 37. Atributos de bpe:details (continuación)

Atributo	Necesario	Descripción
id	no	El ID de JavaServer Faces del componente.
model	sí	Enlace de valor para un bean gestionado de la clase com.ibm.bpe.jsf.handler.BPCDetailsHandler.
rowClasses	no	Lista de clases de estilo CSS, separados por comas, para representar filas.
styleClass	no	Clase CSS que se utiliza para representar el elemento HTML.

Tabla 38. Atributos de bpe:property

Atributo	Necesario	Descripción
converterID	no	ID utilizado para registrar el convertidor en el archivo de configuración de JSF (JavaServer Faces).
label	no	Etiqueta de la propiedad. Si no se establece este atributo, la clase de modelo de cliente proporciona una etiqueta por omisión.
name	sí	Nombre de la propiedad que va a visualizarse. Este nombre debe corresponder a una propiedad con nombre tal como se define en la clase de modelo de cliente correspondiente.

Adición del componente CommandBar a una aplicación JSF

Utilice el componente CommandBar de Business Process Choreographer Explorer para visualizar una barra con botones. Estos botones representan mandatos que operan en la vista de detalles de un objeto o los objetos seleccionados en una lista.

Acerca de esta tarea

Cuando el usuario pulsa un botón en la interfaz de usuario, se ejecuta el mandato correspondiente en los objetos seleccionados. Puede añadir y ampliar el componente CommandBar en la aplicación JSF.

Procedimiento

1. Añada el componente CommandBar al archivo JSP (JavaServer Pages).

   Añada el código `bpe:commandbar` al código `<h:form>`. El código `bpe:commandbar` debe contener un atributo model.

   El ejemplo siguiente muestra cómo añadir un componente CommandBar que proporciona mandatos refresh y claim para una lista de instancias de tareas.

```
<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>
```

El atributo **model** hace referencia a un bean gestionado. Este bean debe implementar la interfaz ItemProvider y proporcionar los objetos Java seleccionados. El componente CommandBar suele utilizarse con el componente List o el componente Details en el mismo archivo JSP. Generalmente, el modelo especificado en el código es el mismo que el especificado en el componente List o en el componente Details de la misma página. Así pues, para un componente List, por ejemplo, el mandato actúa sobre los elementos seleccionados de la lista.

En este ejemplo, el atributo **model** hace referencia al bean gestionado TaskInstanceList. Este bean proporciona los objetos seleccionados en la lista de instancias de tareas. El bean debe implementar la interfaz ItemProvider. Las clases BPCListHandler y BPCDetailsHandler implementan esta interfaz.

2. Opcional: Configure el bean gestionado al que se hace referencia en el código `bpe:commandbar`.

   Si el atributo **model** de CommandBar hace referencia a un bean gestionado que ya está configurado, por ejemplo, para un manejador de lista o de detalles, no se necesita ninguna configuración adicional. Si cambia la configuración de uno de estos manejadores o utiliza un bean gestionado distinto, añada un bean gestionado que implemente la interfaz ItemProvider al archivo de configuración JSF.

3. Añada el código que implementa los mandatos personalizados en la aplicación JSF.

   El siguiente fragmento de código muestra cómo escribir una clase de mandato que implemente la interfaz de mandatos. Se hace referencia a esta clase de mandato (MyClaimCommand) mediante el código `bpe:command` en el archivo JSP.

```
public class MyClaimCommand implements Command {

  public String execute(List selectedObjects) throws ClientException {
     if( selectedObjects != null && selectedObjects.size() > 0 ) {
        try {
         // Determinar HumanTaskManagerService de un bean HTMConnection.
         // Configurar el bean en faces-config.xml para facilitar el acceso
         // en la aplicación 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 ) {
             ;   // Error al iterar o reclamar una instancia de tarea.
                  // Omitir para una mejor comprensión del ejemplo.
           }
         }
```

```
        }
        catch( Exception e ) {
          ;  // Error de configuración o comunicación.
             // Ignorar para una mejor comprensión del ejemplo
        }
      }
      return null;
   }

  // Default implementations
  public boolean isMultiSelectEnabled() { return false; }
  public boolean[] isApplicable(List itemsOnList) {return null; }
  public void setContext(Object targetModel) {; // Not used here }
}
```

El mandato se procesa de la manera siguiente:

a. Se invoca un mandato cuando un usuario pulsa el botón correspondiente de la barra de mandatos. El componente CommandBar recupera los elementos seleccionados del proveedor de elementos especificado en el atributo **model** y pasa la lista de objetos seleccionados al método execute de la instancia de CommandBar.
b. El atributo **commandClass** hace referencia a una implementación de mandatos personalizada que implementa la interfaz de mandatos. Esto significa que el mandato debe implementar el método public String execute(List selectedObjects) throws ClientException. El mandato devuelve un resultado que se utiliza para determinar la siguiente norma de navegación para la aplicación JSF.
c. Después de completar el mandato, el componente CommandBar evalúa el atributo **action**. El atributo **action** puede ser una serie estática o un enlace de método a un método de acción JSF con la signatura `public String Method()`. Utilice el atributo **action** para alterar temporalmente el resultado de una clase de mandato o para especificar explícitamente un resultado para las normas de navegación. No se procesará el atributo **action** si el mandato genera una excepción distinta de ErrorsInCommandException.
d. Si el atributo **commandClass** no tiene especificada una clase de mandatos, se llama inmediatamente a la acción. Por ejemplo, para el mandato de renovación, se llama a la expresión de valor JSF `#{TaskInstanceList.refreshList}` en lugar de llamar a un mandato.

Resultado

La aplicación JSF contiene ahora una página JavaServer que implementa una barra de mandatos personalizada.

Referencia relacionada

"Componente CommandBar: definiciones de código" en la página 166
El componente CommandBar de Business Process Choreographer Explorer muestra una barra con botones. Estos botones operan en el objeto en una vista de detalles o en los objetos seleccionados en una lista.

Proceso de los mandatos

Utilice el componente CommandBar para añadir los botones de acción a la aplicación. El componente crea los botones para las acciones de la interfaz de usuario y gestiona los sucesos que se crean al pulsar un botón.

Estos botones desencadenan funciones que actúan sobre los objetos que la interfaz com.ibm.bpe.jsf.handler.ItemProvider devuelve, como la clase BPCListHandler o la

clase BPCDetailsHandler. El componente CommandBar utiliza el proveedor de elementos definido mediante el valor del atributo **model** del código `bpe:commandbar`.

Cuando se pulsa un botón de la sección de barra de mandatos de la interfaz de usuario de la aplicación, el componente CommandBar gestiona el suceso asociado del siguiente modo:

1. El componente CommandBar identifica la implementación de la interfaz com.ibm.bpc.clientcore.Command que se especifica para el botón que ha generado el suceso.
2. Si el modelo asociado al componente CommandBar implementa la interfaz com.ibm.bpe.jsf.handler.ErrorHandler, se invoca el método clearErrorMap para eliminar los mensajes de error de sucesos anteriores.
3. Se llama al método getSelectedItems de la interfaz ItemProvider. La lista de elementos que se devuelve se pasa al método execute del mandato y se invoca el mandato.
4. El componente CommandBar determina el destino de navegación JSF (JavaServer Faces). Si no se especifica un atributo **action** en el código `bpe:commandbar`, el valor de retorno del método execute especifica el destino de navegación. Si el atributo **action** se establece en un enlace de método JSF, la serie que devuelve el método se interpreta como el destino de navegación. El atributo **action** también puede especificar un destino de navegación explícito.

Componente CommandBar: definiciones de código

El componente CommandBar de Business Process Choreographer Explorer muestra una barra con botones. Estos botones operan en el objeto en una vista de detalles o en los objetos seleccionados en una lista.

El componente CommandBar consta de los códigos de componente JSF: `bpe:commandbar` y `bpe:command`. El código `bpe:command` es un subelemento del código `bpe:commandbar`.

Clase de componente

com.ibm.bpe.jsf.component.CommandBarComponent

Sintaxis de ejemplo

```
<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>
```

Atributos de código

Tabla 39. Atributos de bpe:commandbar

Atributo	Necesario	Descripción
buttonStyleClass	no	Clase de estilo de hoja de estilos en cascada (CSS) que se utiliza para representar los botones de la barra de mandatos.
id	no	ID de JavaServer Faces del componente.
model	sí	Una expresión de enlace de valor a un bean gestionado que implementa la interfaz ItemProvider. Este bean gestionado suele ser la clase com.ibm.bpe.jsf.handler.BPCListHandler o com.ibm.bpe.jsf.handler.BPCDetailsHandler que utiliza el componente List o Details del mismo archivo JSP (JavaServer Pages) que el componente CommandBar.
styleClass	no	Clase de estilo CSS que se utiliza para representar la barra de mandatos.

Tabla 40. Atributos de bpe:command

Atributo	Necesario	Descripción
action	no	Un método de acción JavaServer Faces o el destino de navegación Faces que el botón del mandato va a desencadenar. El destino de navegación que devuelve la acción sobrescribe todas las otras normas de navegación. Se llama a la acción si no se genera una excepción o si el mandato genera una excepción ErrorsInCommandException.
commandClass	no	El nombre de la clase de mandato. El componente CommandBar crea una instancia de la clase y se ejecuta si se selecciona el botón de mandato.
commandID	sí	ID del mandato.
context	no	Un objeto que proporciona contexto para mandatos especificados mediante el atributo **commandClass**. El objeto de contexto se recupera cuando se accede a la barra de mandatos por primera vez.
immediate	no	Especifica cuando se desencadena el mandato. Si el valor de este atributo es `true`, el mandato se desencadena antes de procesar la entrada de la página. El valor predeterminado es `false`.
label	sí	Etiqueta del botón que se representa en la barra de mandatos.
rendered	no	Determina si un botón se ha representado. El valor del atributo puede ser un valor booleano o una expresión de valor.
styleClass	no	eClase de estilo CSS que se utiliza para representar el botón. Este estilo altera temporalmente el estilo de botón definido para la barra de mandatos.

Adición del componente Message a una aplicación JSF

Utilice el componente Message de Business Process Choreographer Explorer para representar objetos de datos y tipos primitivos en aplicaciones JSF (JavaServer Faces).

Acerca de esta tarea

Si el tipo de mensaje es un tipo primitivo, se representan una etiqueta y un campo de entrada. Si el tipo de mensaje es un objeto de datos, el componente atraviesa el objeto y representa los elementos en el objeto.

Procedimiento

1. Añada el componente Message al archivo JSP (JavaServer Pages).

   Añada el código `bpe:form` al código `<h:form>`. El código `bpe:form` debe incluir un atributo model.

   En el ejemplo siguiente se muestra cómo añadir un componente 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>
   ```

   El atributo **model** del componente Message hace referencia a un objeto com.ibm.bpc.clientcore.MessageWrapper. Este objeto de envoltorio envuelve un objeto SDO (Service Data Object) o un tipo primitivo Java, por ejemplo, `int` o `boolean`. En el ejemplo, el mensaje lo suministra una propiedad del bean gestionado MyHandler.

2. Configure el bean gestionado al que se hace referencia en el código `bpe:form`.

   El siguiente ejemplo muestra cómo añadir el bean gestionado MyHandler al archivo de configuración.

   ```
   <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. Añada el código personalizado a la aplicación JSF.

   El siguiente ejemplo muestra cómo implementar mensajes de entrada y salida.

   ```
   public class MyHandler implements ItemListener {

     private TaskInstanceBean taskBean;
     private MessageWrapper inputMessage, outputMessage

     /* Método de receptor, por ejemplo, cuando se ha seleccionado una instancia de
      * tarea en un manejador de listas.
      * Asegúrese de que el manejador se registre en faces-config.xml o
      * manualmente.
      */
   ```

```
  public void itemChanged(Object item) {
    if( item instanceof TaskInstanceBean ) {
        taskBean = (TaskInstanceBean) item ;
    }
  }

  /* Obtener el envoltorio de mensajes de entrada
   */
  public MessageWrapper getInputMessage() {
    try{
        inputMessage = taskBean.getInputMessageWrapper() ;
    }
    catch( Exception e ) {
        ;       //...pasar por alto errores para simplicidad
    }
    return inputMessage;
  }

  /* Obtener el envoltorio de mensajes de salida
   */
  public MessageWrapper getOutputMessage() {
    // Recuperar el mensaje del bean. Si no hay ningún mensaje, cree uno
    // si la tarea ha sido reclamada por el usuario. Asegúrese de que sólo
    // los propietarios potenciales o los propietarios pueden manejar el
    // mensaje de salida.
    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 ) {
        ;       //...pasar por alto errores para simplicidad
    }
    return outputMessage
  }
}
```

El bean gestionado MyHandler implementa la interfaz com.ibm.jsf.handler.ItemListener de manera que pueda registrarse como un receptor de elementos de manejadores de lista. Cuando el usuario pulsa un elemento de la lista, se envía una notificación al bean MyHandler sobre el elemento seleccionado en su método itemChanged(Object item). El manejador comprueba el tipo de elemento y, a continuación, almacena una referencia al objeto TaskInstanceBean asociado. Para utilizar esta interfaz, añada una entrada en la lista itemListener del manejador de lista adecuado en el archivo faces-config.xml.

El bean MyHandler proporciona los métodos getInputMessage y getOutputMessage. Ambos métodos devuelven un objeto MessageWrapper. Los métodos delegan las llamadas al bean de instancia de tarea al que se hace referencia. Si el bean de instancia de tarea devuelve un valor nulo, por ejemplo, porque no se haya establecido un mensaje, el manejador crea y almacena un mensaje nuevo y vacío. El componente Message muestra los mensajes proporcionados por el bean MyHandler.

Resultado

La aplicación JSF contiene ahora una página JavaServer que puede representar objetos de datos y tipos primitivos.

Referencia relacionada

"Componente Message: definiciones de código"
El componente Message de Business Process Choreographer Explorer representa objetos `commonj.sdo.DataObject` y tipos primitivos como, por ejemplo, enteros y series, de una aplicación JSF (JavaServer Faces).

Componente Message: definiciones de código

El componente Message de Business Process Choreographer Explorer representa objetos `commonj.sdo.DataObject` y tipos primitivos como, por ejemplo, enteros y series, de una aplicación JSF (JavaServer Faces).

El componente Message consta del código de componente JSF: `bpe:form`.

Clase de componente

com.ibm.bpe.jsf.component.MessageComponent

Sintaxis de ejemplo

```
<bpe:form model="#{TaskInstanceDetailsBean.inputMessageWrapper}"
          simplification="true" readOnly="true"
          styleClass4table="messageData"
          styleClass4output="messageDataOutput">
</bpe:form>
```

Atributos de código

Tabla 41. Atributos de bpe:form

Atributo	Necesario	Descripción
id	no	El ID de JavaServer Faces del componente.
model	sí	Expresión de enlace de valor que hace referencia a un objeto `commonj.sdo.DataObject` o un objeto `com.ibm.bpc.clientcore.MessageWrapper`.
readOnly	no	Si este atributo se establece en `true`, se representa un formato de sólo lectura. Por omisión, este atributo se establece en `false`.
simplification	no	Si se establece este atributo en `true`, las propiedades que contienen tipos simples y cuya cardinalidad es cero o uno no se muestran. Por omisión, este atributo se establece en `true`.
style4validinput	no	Estilo de hoja de estilos en cascada (CSS) para la entrada de representación que es válida.
style4invalidinput	no	Estilo de CSS para representar la entrada que no es válida.
styleClass4invalidInput	no	Nombre de clase de estilo CSS para representar la entrada que no es válida.
styleClass4output	no	Nombre de clase del estilo CSS para representar los elementos de salida.

Tabla 41. Atributos de bpe:form (continuación)

Atributo	Necesario	Descripción
styleClass4table	no	Nombre de clase del estilo de tabla CSS para representar las tablas representadas por el componente de mensajes.
styleClass4validInput	no	Nombre de clase de estilo CSS para representar la entrada que es válida.

Desarrollo de páginas JSP para mensajes de tareas y de procesos

La interfaz de Business Process Choreographer Explorer proporciona formularios de entrada y salida por omisión para visualizar y entrar datos de empresa. Puede utilizar páginas JSP para proporcionar formularios de entrada y salida personalizados.

Acerca de esta tarea

Para incluir páginas JSP (JavaServer Pages) definidos por el usuario en el cliente Web, debe especificarlos cuando modele una tarea de usuario en WebSphere Integration Developer. Por ejemplo, puede proporcionar páginas JSP para una tarea específica y sus mensajes de entrada y salida, y para un rol de usuario específico o para todos los roles de usuario. Durante la ejecución, las páginas JSP definidos por el usuario se incluyen en la interfaz de usuario para visualizar datos de salida y recopilar datos de entrada.

Los formularios personalizados no son páginas Web autocontenidas: son fragmentos de código HTML que Business Process Choreographer Explorer incorpora en un formulario HTML, por ejemplo, fragmentos para todas las etiquetas y campos de entrada de un mensaje.

Cuando se pulsa un botón en la página que contiene los formularios personalizados, la entrada se envía y se valida en Business Process Choreographer Explorer. La validación se basa en el tipo de las propiedades proporcionadas y el entorno local que se utiliza en el navegador. Si no puede validarse la entrada, se mostrará la misma página de nuevo y se proporcionará información sobre los errores de validación en el atributo de petición messageValidationErrors. La información se proporciona como una correlación de la expresión de vía de acceso XML (XPath) de las propiedades que no son válidas con las excepciones de validación que se han producido.

Para añadir formularios personalizados a Business Process Choreographer Explorer, complete los siguientes pasos utilizando WebSphere Integration Developer.

Procedimiento

1. Cree los formularios personalizados.

   Las páginas JSP definidos por el usuario para los formularios de entrada y salida utilizados en la interfaz Web necesitan acceder a los datos de mensaje. Utilice fragmentos de Java en una JSP o el lenguaje de ejecución JSP para acceder a los datos de mensaje. Los datos de los formularios están disponibles a través del contexto de petición.
2. Asigne las páginas JSP a una tarea.

Abra la tarea de usuario en el editor de tareas de usuario. En los valores de cliente, especifique la ubicación de las páginas JSP definidas por el usuario y el rol al que se aplica el formulario personalizado, por ejemplo, el de administrador. Los valores de cliente de Business Process Choreographer Explorer se almacenan en la plantilla de tareas. Durante la ejecución, estos valores se recuperan con la plantilla de tarea.

3. Empaquete las páginas JSP definidas por el usuario en un archivador Web (archivo WAR).

   Puede incluir el archivo WAR en el archivador de empresa con el módulo que contiene las tareas o desplegar el archivo WAR por separado. Si las JSP se despliegan independientemente, las JSP deben estar disponibles en el servidor donde se despliega Business Process Choreographer Explorer o el cliente personalizado.

   Si va a utilizar JSP personalizadas para los mensajes de proceso y de tarea, debe correlacionar los módulos Web utilizados para desplegar las JSP en los mismos servidores con los que esté correlacionado el cliente JSF personalizado.

Resultado

Los formularios personalizados se representan en Business Process Choreographer durante la ejecución.

Fragmentos JSP definidos por el usuario

Los fragmentos JSP (JavaServer Pages) definidos por el usuario se han incorporado en un código de formulario HTML. Durante la ejecución, Business Process Choreographer Explorer incluye estos fragmentos en la página representada.

El fragmento JSP definido por el usuario para el mensaje de entrada se incorpora antes del fragmento JSP para el mensaje de salida.

```
<html....>
   ...
   <form...>
     JSP de entrada (visualiza el mensaje de entrada de tarea)

     JSP de salida (visualiza el mensaje de salida de tarea)

   </form>
   ...
</html>
```

Dado que los fragmentos JSP definidos por el usuario se incorporan en un código de formulario HTML, puede añadir elementos de entrada. El nombre del elemento de entrada debe coincidir con la expresión XPath (XML Path Language) del elemento de datos. Es importante utilizar como prefijo el nombre del elemento de entrada con el valor de prefijo que se proporciona:

```
<input id="address"
       type="text"
       name="${prefix}/selectPromotionalGiftResponse/address"
       value="${messageMap['/selectPromotionalGiftResponse/address"]}
       size="60"
       align="left" />
```

El valor de prefijo se proporciona como atributo de petición. El atributo asegura que el nombre de entrada será exclusivo en el formulario que lo incluye. El prefijo lo genera Business Process Choreographer Explorer y no debe modificarse:

```
String prefix = (String)request.getAttribute("prefix");
```

Sólo se establece el elemento de prefijo si el mensaje puede editarse en el contexto dado. Los datos de salida pueden visualizarse de distintas maneras, en función del estado de la tarea de usuario. Por ejemplo, si la tarea está en estado de reclamado, los datos de salida pueden modificarse. Sin embargo, si la tarea está en estado de finalizado, los datos sólo pueden visualizarse. En el fragmento JSP, puede probar si el elemento de prefijo existe y presentar el mensaje de acuerdo a ello. La siguiente sentencia JSTL muestra cómo puede probar si se ha establecido el elemento de prefijo.

```
...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
< c:choose>
  <c:when test="${not empty prefix}">
    <!--Modalidad de lectura/grabación-->>
  </c:when>
  <c:otherwise>
    <!--Modalidad de sólo lectura-->>
  </c:otherwise>
</c:choose>
```

Creación de los plug-in para personalizar las funciones de las tareas de usuario

Business Process Choreographer proporciona una infraestructura de gestión de sucesos para los sucesos que se producen durante el proceso de las tareas de usuario. También se proporcionan puntos de plug-in para que pueda adaptar las funciones a sus necesidades. Puede utilizar las interfaces del proveedor de servicio (SPI) para crear los plug-in personalizados para manejar sucesos y procesar las consultas de personal.

Acerca de esta tarea

Puede crear plug-in para sucesos de API de tareas de usuario y sucesos de notificación de escalada. También puede crear un plug-in que procesa los resultados devueltos de una resolución de personas. Por ejemplo, en periodos de hora punta quizá desee añadir usuarios a la lista de resultados para ayudar a equilibrar la carga de trabajo.

Puede registrar los plug-ins en niveles diferentes para todas las tareas a nivel global, para las tareas de un componente de aplicación, para todas las tareas asociadas a una plantilla de tarea o para una instancia de tarea individual.

Creación de manejadores de sucesos de API

Se produce un suceso de la API cuando un método de la API manipula una tarea de usuario. Utilice la interfaz SPI (Service Provider Interface) del plug-in de manejador de sucesos de API para gestionar los sucesos de tarea enviados por la API o los sucesos internos que tienen sucesos de API equivalentes.

Acerca de esta tarea

Complete los pasos siguientes para crear un manejador de sucesos de API.

Procedimiento

1. Grabe una clase que implementa la interfaz APIEventHandlerPlugin2 o amplía la clase de implementación APIEventHandler. Esta clase puede invocar los métodos de otras clases.

- Si utiliza la interfaz APIEventHandlerPlugin2, debe implementar todos los métodos de la interfaz APIEventHandlerPlugin2 y la interfaz APIEventHandlerPlugin.
- Si amplía la clase de implementación de SPI, sobrescriba los métodos necesarios.

Esta clase se ejecuta en el contexto de la aplicación J2EE (Java 2 Enterprise Edition) EJB (Enterprise JavaBeans). Asegúrese de que esta clase y sus clases helper siguen la especificación EJB.

Consejo: Si desea llamar a la interfaz HumanTaskManagerService desde esta clase, no llame a un método que actualiza la tarea que ha producido el suceso. Esta acción produciría un punto muerto en la base de datos.

2. Ensamble la clase de plug-in y sus clases de ayuda en un archivo JAR.

   Si varias aplicaciones J2EE utilizan las clases helper, puede empaquetar estas clases en un archivo JAR individual que puede registrar como una biblioteca compartida.

3. Cree un archivo de configuración de proveedor de servicio para el plug-in del directorio META-INF/services/ del archivo JAR.

   El archivo de configuración proporciona el mecanismo para identificar y cargar el plug-in. Este archivo se ajusta a la especificación de la interfaz del proveedor de servicio de Java 2.

   a. Cree un archivo con el nombre `com.ibm.task.spi.`*`nombre_plug-in`*`APIEventHandlerPlugin`, donde *nombre_plug-in* es el nombre del plug-in.

      Por ejemplo, si el plug-in se denomina `Customer` e implementa la interfaz com.ibm.task.spi.APIEventHandlerPlugin, el nombre del archivo de configuración será `com.ibm.task.spi.`**`Customer`**`APIEventHandlerPlugin`.

   b. En la primera línea del archivo que ni es una línea de comentario ni una línea en blanco, especifique el nombre completo de la clase de plug-in que ha creado en el paso 1.

      Por ejemplo, si la clase de plug-in se denomina `MyAPIEventHandler` y está en el paquete `com.customer.plugins`, entonces la primera línea del archivo de configuración debe contener la entrada siguiente: `com.customer.plugins.MyAPIEventHandler`.

Resultado

Tiene un archivo JAR instalable que contiene un plug-in que maneja sucesos de la API y un archivo de configuración de proveedor de servicio que se puede utilizar para cargar el plug-in.

Consejo: Sólo tiene una propiedad eventHandlerName disponible para registrar los manejadores de sucesos de la API y los manejadores de sucesos de notificación. Si desea utilizar un manejador de sucesos de la API y un manejador de sucesos de notificación, las implementaciones del plug-in deben tener el mismo nombre, por ejemplo `Customer`, que el nombre del manejador de sucesos de la implementación de SPI.

Puede implementar ambos plug-ins utilizando una sola clase o dos clases diferentes. En ambos casos, debe crear dos archivos en el directorio META-INF/services/ del archivo JAR, por ejemplo, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin y com.ibm.task.spi.CustomerAPIEventHandlerPlugin.

Empaquete la implementación de plug-in y las clases helper en un solo archivo JAR.

Qué hacer a continuación

Ahora tiene que instalar y registrar el plug-in de modo que esté disponible para el contenedor de tareas de usuario durante la ejecución. Puede registrar los manejadores de sucesos de la API con una instancia de tarea, una plantilla de tarea o un componente de aplicación.

Manejadores de sucesos de API

Los sucesos de la API se producen cuando se modifica una tarea de usuario o cuando cambia su estado. Para gestionar estos sucesos de API, se invoca directamente el manejador de sucesos antes de modificarse la tarea (método anterior al suceso) y justo antes de que la llamada a la API devuelva el control al sistema (método posterior al suceso).

Si el método de pre-suceso genera un excepción ApplicationVetoException, no se realiza la acción de la API, se devuelve la excepción al proceso que invoca la API y se retrotrae la transacción asociada al suceso. Si el método de pre-suceso lo desencadena un suceso interno y se genera un excepción ApplicationVetoException, no se ejecuta el suceso interno como, por ejemplo, una reclamación automática, pero no se devuelve una excepción a la aplicación cliente. En este caso, se graba un mensaje informativo al archivo SystemOut.log. Si el método de la API genera una excepción durante el proceso, se captura la excepción y se pasa al método de post-suceso. Se vuelve a pasar la excepción al proceso que efectúa la llamada después de que se devuelve el método de post-suceso.

Se aplican estas reglas a los métodos anteriores al suceso:

- Los métodos anteriores al suceso reciben los parámetros del método de API o suceso interno asociado.
- Los métodos de pre-suceso pueden generar una excepción ApplicationVetoException para impedir que continúe el proceso.

Se aplican estas reglas a los métodos posteriores al suceso:

- Los métodos de post-suceso reciben los parámetros que se han proporcionado a la llamada a la API y el valor de retorno. Si la implementación del método de la API genera una excepción, el método de post-suceso también recibe la excepción.
- Los métodos de post-suceso no pueden modificar valores de retorno.
- Los métodos de post-suceso no pueden generar excepciones. Las excepciones de tiempo de ejecución se registran cronológicamente pero se ignoran.

Para implementar los manejadores de sucesos de API, puede utilizar la interfaz APIEventHandlerPlugin2, que amplía la interfaz APIEventHandlerPlugin o amplía la clase de implementación de la SPI com.ibm.task.spi.APIEventHandler. Si el manejador de sucesos se hereda de la clase de implementación por omisión, siempre implementa la versión más reciente de SPI. Si realiza la actualización a una versión más reciente de Business Process Choreographer, serán necesarios pocas modificaciones si desea explotar nuevos métodos SPI.

Si dispone de un manejador de sucesos de notificación y de un manejador de sucesos de API, estos dos manejadores deben tener el mismo nombre porque sólo puede registrar un nombre de manejador de sucesos.

Creación de manejadores de sucesos de notificación

Se producen los sucesos de notificación cuando se escalan las tareas de usuario. Business Process Choreographer proporciona funciones para manejar la escalada como, por ejemplo, la creación de elementos de trabajo de la escalada o el envío de correos electrónicos. Puede crear manejadores de sucesos de notificación para personalizar el modo en que se maneja la escalada.

Acerca de esta tarea

Para implementar los manejadores de sucesos de notificación, puede utilizar la interfaz NotificationEventHandlerPlugin o ampliar la clase de implementación de SPI (Service Provider Interface) com.ibm.task.spi.NotificationEventHandler por omisión.

Complete los pasos siguientes para crear un manejador de sucesos de notificación.

Procedimiento

1. Grabe una clase que implementa la interfaz NotificationEventHandlerPlugin o amplía la clase de implementación NotificationEventHandler. Esta clase puede invocar los métodos de otras clases.

   Si utiliza la interfaz NotificationEventHandlerPlugin, debe implementar todos los métodos de interfaz. Si amplía la clase de implementación de SPI, sobrescriba los métodos necesarios.

   Esta clase se ejecuta en el contexto de la aplicación J2EE (Java 2 Enterprise Edition) EJB (Enterprise JavaBeans). Asegúrese de que esta clase y sus clases helper siguen la especificación EJB.

   El plug-in se invoca con la autorización del rol EscalationUser. Este rol se define cuando se configura el contenedor de tareas de usuario.

   Consejo: Si desea llamar a la interfaz HumanTaskManagerService desde esta clase, no llame a un método que actualiza la tarea o la escalada que ha producido el suceso. Esta acción produciría un punto muerto en la base de datos.
2. Ensamble la clase de plug-in y sus clases de ayuda en un archivo JAR.

   Si varias aplicaciones J2EE utilizan las clases helper, puede empaquetar estas clases en un archivo JAR individual que puede registrar como una biblioteca compartida.
3. Cree un archivo de configuración de proveedor de servicio para el plug-in del directorio META-INF/services/ del archivo JAR.

   El archivo de configuración proporciona el mecanismo para identificar y cargar el plug-in.Este archivo se ajusta a la especificación de la interfaz del proveedor de servicio de Java 2.

   a. Cree un archivo con el nombre `com.ibm.task.spi.`*`nombre_plug-in`*`NotificationEventHandlerPlugin`, donde *nombre_plug-in* es el nombre del plug-in.

      Por ejemplo, si el plug-in se denomina `HelpDeskRequest` (nombre del manejador de sucesos) e implementa la interfaz com.ibm.task.spi.NotificationEventHandlerPlugin, el nombre del archivo de configuración será `com.ibm.task.spi.`**`HelpDeskRequest`**`NotificationEventHandlerPlugin`.

   b. En la primera línea del archivo que ni es una línea de comentario ni una línea en blanco, especifique el nombre completo de la clase de plug-in que ha creado en el paso 1.

Por ejemplo, si la clase de plug-in se denomina `MyEventHandler` y se encuentra en el paquete `com.customer.plugins`, entonces la primera línea del archivo de configuración debe contener la siguiente entrada: `com.customer.plugins.MyEventHandler`.

Resultado

Tiene un archivo JAR instalable que contiene un plug-in que maneja sucesos de notificación y un archivo de configuración de proveedor de servicio que se puede utilizar para cargar el plug-in. Puede registrar los manejadores de sucesos de la API con una instancia de tarea, una plantilla de tarea o un componente de aplicación.

Consejo: Sólo tiene una propiedad eventHandlerName disponible para registrar los manejadores de sucesos de la API y los manejadores de sucesos de notificación. Si desea utilizar un manejador de sucesos de la API y un manejador de sucesos de notificación, las implementaciones del plug-in deben tener el mismo nombre, por ejemplo `Customer`, que el nombre del manejador de sucesos de la implementación de SPI.

Puede implementar ambos plug-ins utilizando una sola clase o dos clases diferentes. En ambos casos, debe crear dos archivos en el directorio META-INF/services/ del archivo JAR, por ejemplo, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin y com.ibm.task.spi.CustomerAPIEventHandlerPlugin.

Empaquete la implementación de plug-in y las clases helper en un solo archivo JAR.

Qué hacer a continuación

Ahora tiene que instalar y registrar el plug-in de modo que esté disponible para el contenedor de tareas de usuario durante la ejecución. Puede registrar los manejadores de sucesos de la notification con una instancia de tarea, una plantilla de tarea o un componente de aplicación.

Creación de plug-in para el proceso posterior a los resultados de consulta de personas

La resolución de personal devuelve una lista de los usuarios que están asignados a un rol determinado, por ejemplo, el propietario potencial de una tarea. Puede crear un plug-in para cambiar los resultados de las consultas de personas devueltos por la resolución de personas. Por ejemplo, para mejorar el equilibrio de la carga de trabajo, podría disponer de un plug-in que elimina del resultado de consulta los usuarios que ya tienen una alta carga de trabajo.

Acerca de esta tarea

Puede tener sólo un plug-in de proceso posterior; esto significa que el plug-in debe gestionar los resultados de personas de todas las tareas. El plug-in puede añadir o eliminar usuarios o cambiar la información de usuario o grupo. También puede cambiar el tipo de resultado, por ejemplo, de una lista de usuarios por un grupo o por todos.

Dado que se ejecuta el plug-in después de que finaliza la resolución de personas, ya se han aplicado las reglas que tenga para mantener la confidencialidad o la

seguridad. El plug-in recibe información sobre los usuarios que se han eliminado durante la resolución de personas (en la clave de correlación HTM_REMOVED_USERS). Debe asegurarse de que el plug-in utiliza esta información de contexto para conservar cualquier confidencialidad o las normas de seguridad que puede tener.

Para implementar el proceso posterior a los resultados de consulta de personas, puede utilizar la interfaz StaffQueryResultPostProcessorPlugin. La interfaz dispone de métodos para modificar los resultados de consulta para tareas, escaladas, plantillas de tarea y componentes de la aplicación.

Complete los pasos siguientes para crear un plug-in para el proceso posterior de los resultados de consulta de personas.

Procedimiento

1. Grabe una clase que implementa la interfaz StaffQueryResultPostProcessorPlugin.

   Debe implementar todos los métodos de interfaz. Esta clase puede invocar los métodos de otras clases.

   Esta clase se ejecuta en el contexto de la aplicación J2EE (Java 2 Enterprise Edition) EJB (Enterprise JavaBeans). Asegúrese de que esta clase y sus clases helper siguen la especificación EJB.

   Consejo: Si desea llamar a la interfaz HumanTaskManagerService desde esta clase, no llame a un método que actualiza la tarea que ha producido el suceso. Esta acción produciría un punto muerto en la base de datos.

   En el ejemplo siguiente se muestra cómo podría cambiar el rol de editor de una tarea denominada 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. Ensamble la clase de plug-in y sus clases de ayuda en un archivo JAR.

   Si varias aplicaciones J2EE utilizan las clases helper, puede empaquetar estas clases en un archivo JAR individual que puede registrar como una biblioteca compartida.

3. Cree un archivo de configuración de proveedor de servicio para el plug-in del directorio META-INF/services/ del archivo JAR.

El archivo de configuración proporciona el mecanismo para identificar y cargar el plug-in.Este archivo se ajusta a la especificación de la interfaz del proveedor de servicio de Java 2.

a. Cree un archivo con el nombre `com.ibm.task.spi.`*`nombre_plug-in`*`StaffQueryResultPostProcessorPlugin`, donde *nombre_plug-in* es el nombre del plug-in.

   Por ejemplo, si el plug-in se denomina `MyHandler` e implementa la interfaz com.ibm.task.spi.StaffQueryResultPostProcessorPlugin, el nombre del archivo de configuración será `com.ibm.task.spi.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`.

b. En la primera línea del archivo que ni es una línea de comentario ni una línea en blanco, especifique el nombre completo de la clase de plug-in que ha creado en el paso 1.

   Por ejemplo, si la clase de plug-in se denomina `StaffPostProcessor` y se encuentra en el paquete `com.customer.plugins`, entonces la primera línea del archivo de configuración debe contener la siguiente entrada: `com.customer.plugins.StaffPostProcessor`. Tiene un archivo JAR instalable que contiene un plug-in que realiza un proceso posterior de los resultados de la consulta de personas y un archivo de configuración de proveedor de servicio que se puede utilizar para cargar el plug-in.

4. Instale el plug-in.

   Puede tener sólo un plug-in de proceso posterior para los resultados de consulta de personas. Debe instalar el plug-in como una biblioteca compartida.

5. Registre el plug-in.

   a. En la consola administrativa, vaya a la página Propiedades personalizadas del Gestor de tareas de usuario (**Servidores de aplicaciones** → *nombre_servidor* → **Contenedor de tarea de usuario** → **Propiedades personalizadas**).

   b. Añada una propiedad personalizada con el nombre **Staff.PostProcessorPlugin** y un valor del nombre que ha asignado al plug-in, en este ejemplo `MyHandler`.

Instalación de los plug-ins

Para utilizar un plug-in, debe instalar el plug-in de modo que el contenedor de tareas pueda acceder al mismo.

Acerca de esta tarea

La manera en que puede instalar el plug-in depende de que el plug-in lo vaya a utilizar una sola aplicación J2EE (Java2 Enterprise Edition) o varias.

Complete uno de los pasos siguientes para instalar un plug-in.

- Instale un plug-in para que sólo lo utilice una aplicación J2EE.

  Añada el archivo JAR del plug-in al archivo EAR de la aplicación. En el editor de descriptores de despliegue de WebSphere Integration Developer, instale el archivo JAR del plug-in como un archivo JAR de programa de utilidad de proyecto para la aplicación J2EE del módulo EJB (Enterprise JavaBeans) principal.

- Instale un plug-in para que lo utilicen varias aplicaciones J2EE.

  Coloque el archivo JAR en una biblioteca compartida de WebSphere Application Server y asocie la biblioteca a las aplicaciones que tienen que acceder al plug-in. Para que el archivo JAR esté disponible en un entorno de despliegue de red,

distribuya el archivo JAR en cada servidor manualmente y luego instale la biblioteca compartida una vez para cada célula.

Qué hacer a continuación

Ahora puede registrar el plug-in.

Registro de plug-in

Puede registrar sus plug-ins en niveles diferentes de la jerarquía de artefactos del contenedor de tareas. Por ejemplo, para todas las tareas de un nivel global, para las tareas de un componente de aplicación, para todas las tareas asociadas a una plantilla de tarea o para una instancia de tarea individual.

Acerca de esta tarea

Cuando registre varios plug-ins, se da soporte a los ámbitos. Esto significa que un plug-in que se registra en un nivel inferior de la jerarquía del artefacto del contenedor de tareas como, por ejemplo, una instancia de tarea, se utiliza en lugar del plug-in registrado en un nivel superior como, por ejemplo, una plantilla de tarea o un componente de aplicación. El ámbito se da soporte en todos los niveles de la jerarquía. El contenedor de tarea utiliza el plug-in registrado en el nivel más bajo de la jerarquía.

Puede registrar un plug-in de uno de los modos siguientes.

- Registre el plug-in en el modelo de tarea.

  En el editor de tareas de WebSphere Integration Developer en la página Detalles del área de propiedades para la tarea, especifique el nombre del manejador de sucesos en el campo **Nombre del manejador de sucesos**.
- Registre el plug-in para las tareas ad-hoc o las plantillas de tarea que crea durante la ejecución.

  Utilice el método setEventHandlerName de la clase TTask para registrar el nombre del manejador de sucesos.
- Cambie el manejador de sucesos registrado para una instancia de tarea durante la ejecución.

  Utilice el método update(Task task) para utilizar un manejador de tareas diferente para una instancia de tarea durante la ejecución. El usuario que realice la llamada debe tener autorización de administrador para actualizar esta propiedad.
- Registre el plug-in a nivel global.

  En la consola de administración, en la página Propiedades personalizadas del contenedor de tareas de usuario, defina una propiedad personalizada para el plug-in. El valor de la propiedad personalizada es el nombre del plug-in.

Parte 2. Despliegue de aplicaciones

Capítulo 3. Visión general de la preparación e instalación de módulos

La instalación de módulos (también conocida como despliegue) activa los módulos en un entorno de prueba o de producción. Esta visión general describe brevemente los entornos de prueba y de producción y algunos de los pasos implicados en la instalación de módulos.

Nota: El proceso de instalación de aplicaciones en un entorno de producción es similar al proceso descrito en "Desarrollo y despliegue de aplicaciones" en el centro de información de WebSphere Application Server Network Deployment, versión 6. Si no está familiarizado con estos temas, revíselos antes.

Antes de instalar un módulo en un entorno de producción, verifique siempre los cambios en un servidor de prueba. Para instalar módulos en un entorno de prueba, utilice WebSphere Integration Developer (vea el centro de información de WebSphere Integration Developer. Para instalar módulos en un entorno de producción, utilice WebSphere Process Server.

Este tema describe los conceptos y tareas necesarios para preparar e instalar módulos en un entorno de producción. Otros temas describen los archivos que alojan los objetos que el módulo utiliza y le ayudan a mover el módulo desde el entorno de prueba al entorno de producción. Es importante entender estos archivos y lo que contienen para que pueda estar seguro de que ha instalado correctamente los módulos.

Visión general de bibliotecas y archivos JAR

Los módulos suelen utilizar artefactos que están ubicados en bibliotecas. Los artefactos y bibliotecas están contenidos en archivos Java (JAR) que se identifican al desplegar un módulo.

Al desarrollar un módulo, puede que haya identificado ciertos recursos o componentes que diversos elementos del módulo podrían utilizar. Estos recursos o componentes pueden ser objetos que se crearon al desarrollar el módulo u objetos ya existentes que residen en una biblioteca que ya está desplegada en el servidor. Este tema describe las bibliotecas y los archivos que se necesitarán al instalar una aplicación.

¿Qué es una biblioteca?

Una biblioteca contiene objetos y recursos utilizados por varios módulos de WebSphere Integration Developer. Los artefactos pueden estar en archivos JAR, archivos de recursos (RAR) o archivos de servicio Web (WAR). Algunos de estos artefactos son los siguientes:

- Descriptores de interfaces o servicios Web (archivos con la extensión .wsdl)
- Definiciones de esquema XML de objetos de empresa (archivos con la extensión .xsd)
- Correlaciones de objetos de empresa (archivos con la extensión .map)
- Definiciones de relación y de rol (archivos con la extensión .rel y .rol)

Cuando un módulo necesita un artefacto, el servidor localiza el artefacto a partir de la vía de acceso de clase EAR y carga el artefacto en la memoria, si no está cargado ya. A partir de ese momento, cualquier solicitud del artefacto utilizará esa copia hasta que sea sustituida. Figura 5 muestra cómo una aplicación contiene componentes y bibliotecas relacionadas.

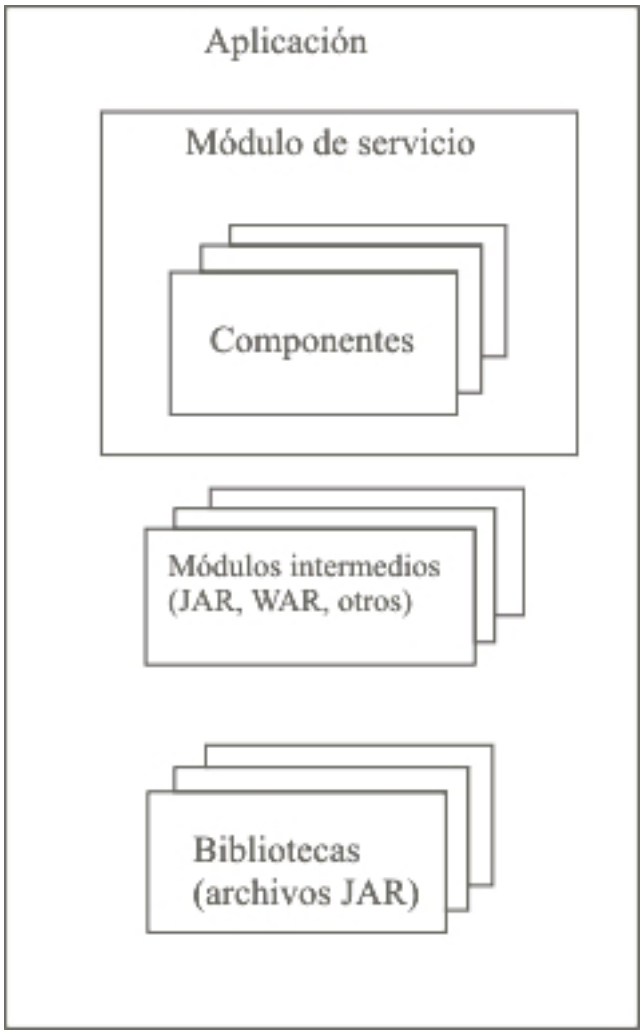


Figura 5. Relación entre módulo, componente y biblioteca

¿Qué son los archivos JAR, RAR y WAR?

Hay varios archivos que pueden contener componentes de un módulo. Estos archivos están completamente descritos en la especificación Java Platform Enterprise Edition. Pueden encontrarse detalles sobre los archivos JAR en la >especificación JAR.

En WebSphere Process Server, un archivo JAR también contiene una aplicación, que es la versión ensamblada del módulo con todas las referencias e interfaces de soporte a otros componentes utilizados por el módulo. Para instalar la aplicación por completo, necesita este archivo JAR, otras bibliotecas como archivos JAR, archivos WAR (archivadores de servicios Web) y RAR (archivadores de recursos), archivos JAR EJB (Enterprise Java Beans) de bibliotecas intermedias o cualquier otro archivador, y crear un archivo EAR instalable mediante el mandato serviceDeploy.

Convenios de denominación para módulos intermedios.

En la biblioteca, hay requisitos para los nombres de los módulos intermedios. Estos nombres son exclusivos para un módulo específico. Indique cualquier otro módulo necesario para desplegar la aplicación, de manera que no se produzcan conflictos con los nombres de los módulos intermedios. Para un módulo denominado *myService*, los nombres de los módulos intermedios son:

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

Nota: El mandato serviceDeploy sólo crea el módulo intermedio Web *myService* si el servicio incluye un servicio de tipo de puerto WSDL.

Consideraciones al utilizar bibliotecas

El uso de bibliotecas proporciona coherencia de objetos de empresa y coherencia de proceso entre módulos, porque cada módulo llamante tiene su propia copia de un componente específico. Para evitar incoherencias y anomalías, es importante asegurarse de que los cambios en componentes y objetos de empresa utilizados por módulos llamantes se coordinen con todos los módulos llamantes. Actualice los módulos llamantes mediante las acciones siguientes:

1. Copiar el módulo y la copia más reciente de las bibliotecas en el servidor de producción
2. Reconstruir el archivo EAR instalable mediante el mandato serviceDeploy
3. Detener la aplicación en ejecución que contiene el módulo llamante y volver a instalarlo
4. Reiniciar la aplicación que contiene el módulo llamante

Referencia relacionada

Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Visión general del archivo EAR

Un archivo EAR es un elemento crítico en el despliegue de una aplicación de servicio en un servidor de producción.

Un archivo EAR (de archivo de empresa) es un archivo comprimido que contiene las bibliotecas, beans de empresa y archivos JAR que la aplicación requiere para el despliegue.

Se crea un archivo JAR al exportar los módulos de aplicación de WebSphere Integration Developer. Utilice este archivo JAR y otras bibliotecas u objetos de artefactos como entrada en el proceso de instalación. El mandato serviceDeploy crea un archivo EAR a partir de los archivos de entrada que contienen las descripciones de componente y código Java que abarcan la aplicación.

Referencia relacionada

Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Preparación para desplegar en un servidor

Después de desarrollar y probar un módulo, debe exportar el módulo desde un sistema de prueba y traerlo a un entorno de producción para su despliegue. Para instalar una aplicación, también debe conocer las vías de acceso necesarias al exportar el módulo y las bibliotecas que el módulo necesite.

Antes de empezar

Antes de iniciar esta tarea, debe haber desarrollado y probado los módulos en un servidor de prueba y haber resuelto los problemas y cuestiones de rendimiento.

Acerca de esta tarea

Esta tarea verifica que todas las piezas necesarias de una aplicación estén disponibles y empaquetadas en los archivos correctos para llevarlas al servidor de producción.

Nota: También puede exportar un archivo EAR (Enterprise ARchive) desde WebSphere Integration Developer e instalarlo directamente en WebSphere Process Server.

Importante: Si los servicios de un componente utilizan una base de datos, instale la aplicación en un servidor conectado directamente a la base de datos.

Procedimiento

1. Localice la carpeta que contiene los componentes correspondientes al módulo que va a desplegar.

   La carpeta de componentes debe denominarse *nombre-módulo* con un archivo denominado *módulo*.module, el módulo base.
2. Verifique que todos los componentes contenidos en el módulo están en subcarpetas de componente debajo de la carpeta de módulo.

   Para su facilidad de uso, indique la subcarpeta similar a *módulo/componente*.
3. Verifique que todos los archivos que abarcan cada componente están contenidos en la subcarpeta de componentes correspondiente y tiene un nombre similar a *nombre-archivo-componente*.component.

   Los archivos de componente contienen las definiciones para cada componente individual en el módulo.
4. Verifique que todos los componentes y artefactos están en las subcarpetas del componente que los necesita.

   En este paso se asegura que las referencias a artefactos que un componente necesita estén disponibles. Los nombres de componentes no deben estar en conflicto con los nombres que el mandato serviceDeploy utiliza para los módulos intermedios. Consulte el apartado Convenios de denominación para módulos intermedios
5. Verifique que un archivo de referencias, *módulo*.references, existe en la carpeta de módulo del paso 1.

   El archivo de referencias define las referencias y las interfaces en el módulo.

6. Verifique que un archivo de cables, *módulo*.wires, existe en la carpeta de componentes.

   El archivo de cables completa las conexiones entre las referencias y las interfaces en el módulo.

7. Verifique que un archivo de manifiesto, *módulo*.manifest, existe en la carpeta de componentes.

   El manifiesto lista el módulo y todos los componentes que abarcan el módulo. También contiene una sentencia classpath de manera que el mandato serviceDeploy pueda localizar los demás módulos que el módulo necesita.

8. Cree un archivo comprimido o JAR del módulo como entrada al mandato serviceDeploy que utilizará para preparar el módulo para la instalación en el servidor de producción.

Ejemplo de estructura de carpetas para el módulo MyValue antes del despliegue

El ejemplo siguiente ilustra la estructura de directorios del módulo MyValueModule, que se compone de los componentes MyValue, CustomerInfo y 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
```

Instale el módulo en los sistemas de producción como se describe en el apartado Instalación de un módulo en un servidor de producciónInstalación de un módulo en un servidor de producción.

Referencia relacionada

Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Consideraciones sobre la instalación de aplicaciones de servicio en clústeres

La instalación de una aplicación de servicio en un clúster pone requisitos adicionales en el usuario. Es importante que tenga en cuenta estas consideraciones al instalar aplicaciones de servicio en un clúster.

Los clústeres pueden proporcionar muchas ventajas al entorno de proceso, ya que proporcionan economías de escala que le ayudarán a equilibrar la carga de trabajo de peticiones entre los servidores y proporcionan un nivel de disponibilidad a los clientes de las aplicaciones. Tenga en cuenta lo siguiente antes de instalar una aplicación que contenga servicios en un clúster:

- ¿Los usuarios de la aplicación necesitarán la potencia de proceso y la disponibilidad que proporciona la agrupación en clúster?

  Si la respuesta es afirmativa, la solución correcta es la agrupación en clúster. La agrupación en clúster aumentará la disponibilidad y la capacidad de las aplicaciones.
- ¿Está el clúster preparado correctamente para las aplicaciones de servicio?

  Debe configurar el clúster correctamente antes de instalar e iniciar la primera aplicación que contiene un servicio. Una anomalía al configurar correctamente el clúster impide que las aplicaciones procesen peticiones de forma correcta.
- ¿Tiene el clúster una copia de seguridad?

  También debe instalar la aplicación en el clúster de copia de seguridad.

Capítulo 4. Instalación de un módulo en un servidor de producción

Este tema describe los pasos implicados en la selección de una aplicación de un servidor de prueba y su despliegue en un entorno de producción.

Antes de empezar

Antes de desplegar una aplicación de servicio en un servidor de producción, ensamble y pruebe la aplicación en un servidor de prueba. Después de la prueba, exporte los archivos pertinentes tal como se describe en *Preparación para el despliegue en un servidor* en el PDF Desarrollo y despliegue de módulos y traslade los archivos al sistema de producción para desplegarlos. Consulte los centros de información correspondientes a WebSphere Integration Developer y WebSphere Application Server Network Deployment para obtener más información.

Procedimiento

1. Copie el módulo y otros archivos en el servidor de producción.

   Los módulos y recursos (archivos EAR, JAR, RAR y WAR) que la aplicación necesita se trasladan al entorno de producción del usuario.
2. Ejecute el mandato serviceDeploy para crear un archivo EAR instalable.

   Este paso define el módulo en el servidor como preparación para instalar la aplicación en producción.
   a. Localice el archivo JAR que contiene el módulo que va a desplegar.
   b. Emita el mandato utilizando el archivo JAR del paso anterior como entrada.
3. Instale el archivo EAR a partir del paso 2. Cómo se instalan las aplicaciones depende de si se instala la aplicación en un servidor autónomo o en un servidor de una célula.

   Nota: Puede utilizar la consola administrativa o un script para instalar la aplicación. Para ver información adicional, consulte el centro de información de WebSphere Application Server.
4. Guarde la configuración. El módulo se instala ahora como una aplicación.
5. Inicie la aplicación.

Resultado

La aplicación está ahora activa y el trabajo debe fluir a través del módulo.

Qué hacer a continuación

Supervise la aplicación para asegurarse de que el servidor procesa las peticiones correctamente.

Referencia relacionada

Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Creación de un archivo EAR instalable utilizando serviceDeploy

Para instalar una aplicación en el entorno de producción, tome los archivos copiados en el servidor de producción y cree un archivo EAR instalable.

Antes de empezar

Antes de iniciar esta tarea, debe tener un archivo JAR que contenga el módulo y los servicios que va a desplegar en el servidor. Consulte Preparación para desplegar en un servidor para obtener más información.

Acerca de esta tarea

El mandato serviceDeploy toma un archivo JAR y otros archivos EAR, JAR, RAR, WAR y ZIP dependientes que pueda haber y construye un archivo EAR que puede instalarse en un servidor.

Procedimiento

1. Localice el archivo JAR que contiene el módulo que va a desplegar.
2. Emita el mandato utilizando el archivo JAR del paso anterior como entrada.
   Este paso crea un archivo EAR.

   Nota: Siga estos pasos en una consola administrativa.
3. Seleccione el archivo EAR que se va a instalar en la consola administrativa del servidor.
4. Pulse **Guardar** para instalar el archivo EAR.

Referencia relacionada

Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Despliegue de aplicaciones utilizando tareas Ant de Apache

Este tema describe cómo utilizar tareas Ant de Apache para automatizar el despliegue de aplicaciones en WebSphere Process Server. Mediante las tareas Ant de Apache, puede definir el despliegue de varias aplicaciones y hacer que se ejecuten en un servidor en modalidad desatendida.

Antes de empezar

En esta tarea se da por supuesto lo siguiente:
- Las aplicaciones que se despliegan ya se han desarrollado y probado.
- Las aplicaciones van a instalarse en el mismo servidor o servidores.
- El usuario ya conoce las tareas Ant de Apache.
- El usuario entiende el proceso de despliegue.

La información sobre el desarrollo y prueba de aplicaciones se encuentra en el centro de información de WebSphere Integration Developer.

La parte de consulta del centro de información de WebSphere Application Server Network Deployment contiene una sección sobre interfaces de programación de

aplicaciones. Las tareas Ant de Apache se describen en el paquete com.ibm.websphere.ant.tasks. A los fines de este tema, las tareas de interés son ServiceDeploy e InstallApplication.

Acerca de esta tarea

Si tiene que instalar varias aplicaciones simultáneamente, desarrolle una tarea Ant de Apache antes de realizar el despliegue. A continuación, la tarea Ant de Apache puede desplegar e instalar las aplicaciones en los servidores sin que el usuario tenga que implicarse en el proceso.

Procedimiento

1. Identifique las aplicaciones que va a desplegar.
2. Cree un archivo JAR para cada aplicación.
3. Copie los archivos JAR en los servidores de destino.
4. Cree una tarea Ant de Apache para ejecutar el mandato ServiceDeploy para crear el archivo EAR para cada servidor.
5. Cree una tarea Ant de Apache para ejecutar el mandato InstallApplication para cada archivo EAR del paso 4 en los servidores pertinentes.
6. Ejecute la tarea ServiceDeploy de Ant de Apache para crear el archivo EAR para las aplicaciones.
7. Ejecute la tarea InstallApplication de Ant de Apache a fin de instalar los archivos EAR desde el paso 6.

Resultado

Las aplicaciones se despliegan correctamente en los servidores de destino.

Ejemplo de despliegue desatendido de una aplicación

Este ejemplo muestra una tarea Ant de Apache contenida en un archivo 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>
```

Esta sentencia muestra cómo invocar la tarea Ant de Apache.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

Consejo: Es posible desplegar varias aplicaciones en modalidad desatendida si se añaden sentencias de proyecto adicionales en el archivo.

Qué hacer a continuación

Utilice la consola administrativa para verificar que las aplicaciones recién instaladas se inician y procesan el flujo de trabajo correctamente.

Referencia relacionada


Mandato serviceDeploy

Utilice el mandato serviceDeploy para empaquetar módulos compatibles con SCA (Service Component Architecture) como aplicaciones Java que se pueden instalar en un servidor. El mandato es útil al realizar instalaciones de procesos por lotes a través de wsadmin.

Capítulo 5. Instalación de aplicaciones de procesos de empresa y tareas de usuario

Puede distribuir módulos de SCA (Service Component Architecture) que contengan procesos de empresa y/o tareas de usuario para destinos de despliegue. Un destino de despliegue puede ser un servidor o un clúster.

Antes de empezar

Verifique que Business Flow Manager y/o Human Task Manager esté instalado y configurado para cada servidor o clúster de aplicaciones donde desee instalar la aplicación.

Acerca de esta tarea

Puede instalar aplicaciones de procesos de empresa y tareas desde la consola administrativa, desde la línea de mandatos o, por ejemplo, ejecutando un script administrativo.

Resultado

Después de instalar un proceso de empresa o una aplicación de tarea de usuario, todas las plantillas de proceso de empresa y de tarea de usuario se colocan en el estado de inicio. Puede crear instancias de proceso e instancias de tareas a partir de estas plantillas.

Qué hacer a continuación

Antes de que pueda crear instancias de de proceso o de tarea, debe iniciar la aplicación.

Conceptos relacionados

"Despliegue de los procesos de empresa y las tareas de usuario" en la página 194
Cuando WebSphere Integration Developer o el despliegue de servicio genera el código de despliegue para el proceso o tarea, cada componente de proceso o componente de tarea se correlaciona con un enterprise bean de sesión. Todo el código de despliegue se empaqueta en el archivo EAR (de aplicación de empresa). Además, para cada proceso, una clase Java que representa código Java en este proceso se genera y se incorpora en el archivo EAR durante la instalación de la aplicación de empresa. Cada versión nueva de un modelo que se vaya a desplegar debe estar empaquetada en una nueva aplicación de empresa.

"Cómo las aplicaciones de procesos de empresa y de tareas de usuario se instalan en un entorno de Network Deployment" en la página 194
Cuando las plantillas de proceso o las plantillas de tarea de usuario se instalan en un entorno de Network Deployment, la instalación de la aplicación efectúa automáticamente las acciones siguientes.

Cómo las aplicaciones de procesos de empresa y de tareas de usuario se instalan en un entorno de Network Deployment

Cuando las plantillas de proceso o las plantillas de tarea de usuario se instalan en un entorno de Network Deployment, la instalación de la aplicación efectúa automáticamente las acciones siguientes.

La aplicación se instala asíncronamente por etapas. Cada etapa debe completarse satisfactoriamente para que la etapa siguiente pueda empezar.

1. La instalación de la aplicación se inicia en el gestor de despliegue.

   Durante esta etapa, las plantillas de proceso de empresa o las plantillas de tarea de usuario se configuran en el repositorio de configuración de WebSphere. La aplicación también se valida. Si se producen errores, estos aparecen en el archivo System.out, en el archivo System.err o como entradas FFDC en el gestor de despliegue.
2. La instalación de la aplicación continúa en el agente de nodo.

   Durante esta etapa, la instalación de la aplicación se desencadena en una instancia del servidor de aplicaciones. Esta instancia del servidor de aplicaciones es el destino de despliegue (o forma parte del mismo). Si el destino de despliegue es un clúster con diversos miembros, la instancia de servidor se elige arbitrariamente entre los miembros de este clúster. Si se producen errores durante esta etapa, estos aparecen en el archivo SystemOut.log, en el archivo SystemErr.log o como entradas FFDC en el agente de nodo.
3. La aplicación se ejecuta en la instancia de servidor.

   Durante esta etapa, las plantillas de proceso o las plantillas de usuario se despliegan en la base de datos de Business Process Choreographer en el destino de despliegue.Si se producen errores, estos aparecen en el archivo System.out, en el archivo SystemErr.log o como entradas FFDC en esta instancia de servidor.

   Tareas relacionadas

   Capítulo 5, “Instalación de aplicaciones de procesos de empresa y tareas de usuario”, en la página 193
   Puede distribuir módulos de SCA (Service Component Architecture) que contengan procesos de empresa y/o tareas de usuario para destinos de despliegue. Un destino de despliegue puede ser un servidor o un clúster.

Despliegue de los procesos de empresa y las tareas de usuario

Cuando WebSphere Integration Developer o el despliegue de servicio genera el código de despliegue para el proceso o tarea, cada componente de proceso o componente de tarea se correlaciona con un enterprise bean de sesión. Todo el código de despliegue se empaqueta en el archivo EAR (de aplicación de empresa). Además, para cada proceso, una clase Java que representa código Java en este proceso se genera y se incorpora en el archivo EAR durante la instalación de la aplicación de empresa. Cada versión nueva de un modelo que se vaya a desplegar debe estar empaquetada en una nueva aplicación de empresa.

Cuando instale una aplicación de empresa que contenga procesos de empresa o tareas de usuario, estos se almacenan como plantillas de proceso o plantillas de tarea de usuario, según corresponda, en la base de datos de Business Process Choreographer. Por omisión, las plantillas recién instaladas están en el estado de

iniciado. Sin embargo, la aplicación de empresa recién instalada está en el estado de detenido. Todas las aplicaciones de empresa se pueden iniciar y detener individualmente.

Puede desplegar muchas versiones diferentes de una plantilla de proceso o de una plantilla de tarea, cada una en una aplicación de empresa diferente. Cuando se instala una nueva aplicación de empresa, la versión de la plantilla que se instala viene determinada de la manera siguiente:

- Si el nombre de la plantilla y el espacio de nombres de destino todavía no existen, se instala una plantilla nueva.
- Si el nombre de la plantilla y el espacio de nombres de destino son los mismos que los de una plantilla existente, pero la fecha de válido-desde es diferente, se instala una versión nueva de una plantilla existente.

Nota: El nombre de la plantilla se deriva del nombre del componente y no del proceso de empresa ni de la tarea de usuario.

Si no se especifica una fecha de válido-desde, la fecha se determinará de la manera siguiente:

- Si utiliza WebSphere Integration Developer, la fecha de válido-desde es la fecha en la se modeló la tarea de usuario o el proceso de empresa.
- Si utiliza el despliegue de servicios, la fecha de válido-desde es la fecha en la que se ejecutó el mandato serviceDeploy. Solo las tareas de colaboración obtienen la fecha en la que se instaló la aplicación como fecha de válido-desde.

Tareas relacionadas

Capítulo 5, “Instalación de aplicaciones de procesos de empresa y tareas de usuario”, en la página 193
Puede distribuir módulos de SCA (Service Component Architecture) que contengan procesos de empresa y/o tareas de usuario para destinos de despliegue. Un destino de despliegue puede ser un servidor o un clúster.

Instalación interactiva de aplicaciones de procesos de empresa y tareas de usuario

Puede instalar interactivamente un aplicación durante la ejecución mediante la herramienta wsadmin y el script installInteractive. Puede utilizar este script para cambiar los valores que no se pueden modificar si utiliza la consola administrativa para instalar la aplicación.

Acerca de esta tarea

Efectúe los pasos siguientes para instalar las aplicaciones de proceso de empresa de forma interactiva.

Procedimiento

1. Inicie la herramienta wsadmin.

   En el directorio *raíz_perfil*/bin, escriba `wsadmin`.
2. Instale la aplicación.

   En el indicador de línea de mandatos wsadmin, escriba el mandato siguiente:

   ```
   $AdminApp installInteractive application.ear
   ```

donde *application.ear* es el nombre cualificado del archivo EAR (Enterprise Archive) que contiene la aplicación de proceso. A través de una serie de tareas verá indicadores donde podrá cambiar los valores de la aplicación.

3. Guarde los cambios de configuración.

   En el indicador de línea de mandatos wsadmin, escriba el mandato siguiente:

   ```
   $AdminConfig save
   ```

   Debe guardar los cambios para transferir las actualizaciones al depósito de configuración maestro. Si un proceso de script finaliza y no ha guardado los cambios, se descartan los cambios.

Configuración de los orígenes de datos y de las referencias del conjunto de las aplicaciones de procesos

Es posible que tenga configurar las aplicaciones de proceso que ejecutan sentencias SQL para la infraestructura de base de datos específica. Estas sentencias SQL pueden proceder de las actividades del servicio de información o pueden ser sentencias que se ejecutan durante la instalación del proceso o el arranque de la instancia.

Acerca de esta tarea

Cuando instala la aplicación, puede especificar los siguientes tipos de orígenes de datos:

- Orígenes de datos que ejecutan sentencias SQL durante la instalación del proceso
- Orígenes de datos que ejecutan sentencias SQL durante el arranque de una instancia de proceso
- Orígenes de datos que ejecutan actividades de snippets SQL

El origen de datos necesario para ejecutar una actividad de snippet SQL se define en una variable BPEL de tipo tDataSource. Los nombres de esquema y tabla de base de datos son necesarios para cualquier actividad de snippet SQL definida en las variables BPEL de tipo tSetReference. Puede configurar los valores iniciales de estas dos variables.

Puede utilizar la herramienta wsadmin para especificar los orígenes de datos.

Procedimiento

1. Instale la aplicación de proceso interactivamente utilizando la herramienta wsadmin.
2. Recorra las tareas hasta que encuentre las tareas para actualizar los orígenes de datos y las referencias del conjunto.

   Configure estos valores para su entorno. El ejemplo siguiente muestra los valores que puede modificar para cada una de estas tareas.
3. Guarde los cambios.

Ejemplo: Actualización de los orígenes de datos y de las referencias del conjunto, mediante la herramienta wsadmin

En la tarea **Actualización de orígenes de datos**, puede cambiar los valores de los orígenes de datos para los valores de variables iniciales y las sentencias que se utilizan durante la instancia del proceso o durante el inicio del proceso. En la tarea **Actualizando referencias del conjunto**, puede configurar los valores relacionados con el esquema de base de datos y los nombres de las tablas.

```
Tarea[24]: Actualización de orígenes de datos

//Cambiar los valores de los orígenes de datos para los valores de variables
//iniciales durante el inicio del proceso

Nombre de proceso:  Test
// Nombre de la plantilla de proceso
Inicio del proceso o tiempo de instalación:  Inicio del proceso
// Indica si se evalúa el valor especificado
//durante el inicio del proceso o la instalación del proceso
Sentencia o variable:  Variable
// Indica que se ha de modificar una variable de origen de datos
Nombre de origen de datos:  MyDataSource
// Nombre de la variable
Nombre JNDI:[jdbc/sample]:jdbc/newName
// Establece el nombre JNDI en jdbc/newName
Tarea[25]: Actualizando referencias del conjunto

// Cambio de los valores de referencia del conjunto que se utilizan como valores
// iniciales para variables BPEL

Nombre de proceso:  Test
// Nombre de la plantilla de proceso
Variable:  SetRef
// El nombre de la variable BPEL
Nombre JNDI:[jdbc/sample]:jdbc/newName
// Establece el nombre JNDI del origen de datos de la referencia del
// conjunto en jdbc/newName
Nombre de esquema: [IISAMPLE]
// El nombre del esquema de base de datos
Prefijo de esquema:  []:
// El prefijo del nombre del esquema.
// Este valor sólo se aplica si se genera el nombre de esquema.
Nombre de tabla: [SETREFTAB]: NEWTABLE
// Establece el nombre de la tabla de base de datos en NEWTABLE
Prefijo de tabla:  []:// El prefijo del nombre de tabla.
// Este valor sólo se aplica si se genera el nombre de prefijo.
```

Desinstalación de aplicaciones de procesos de empresa y tareas de usuario utilizando la consola administrativa.

Puede utilizar la consola administrativa para desinstalar aplicaciones que contienen procesos de empresa o tareas de usuario.

Antes de empezar

Para desinstalar una aplicación que contenga procesos de empresa o tareas de usuario, deben cumplirse los siguiente requisitos previos:

- Si la aplicación se ha instalado en un servidor autónomo, el servidor debe estar en ejecución y tener acceso a la base de datos de Business Process Choreographer.
- Si la aplicación se ha instalado en un clúster, el gestor de despliegue y al menos un miembro del clúster deben estar en ejecución. El miembro del clúster tiene acceso a la base de datos de Business Process Choreographer.
- Si la aplicación se ha instalado en un servidor gestionado, el gestor de despliegue y este servidor deben estar en ejecución. El servidor debe tener acceso a la base de datos de Business Process Choreographer.
- Todas las plantillas de proceso de empresa y tarea de usuario pertenecientes a la aplicación deben en estado detenido.

- No hay presentes instancias de plantillas de proceso de empresa ni de tarea de usuario en ningún estado.
-

Para entornos de servidor autónomo que se utilizan como entornos de desarrollo y de prueba de unidades, el servidor se puede configurar para ejecutarse en modalidad de desarrollo. Esta configuración no necesita que se detengan las plantillas ni que haya instancias. No obstante, esta configuración no es válida para entornos de producción.

Acerca de esta tarea

Para desinstalar una aplicación de empresa que contenga procesos de empresa o tareas de usuario, siga estos pasos:

Procedimiento

1. Detenga todas las plantillas de proceso y de tarea de la aplicación.

   Esta acción impide la creación de instancias de proceso y de tarea.

   a. Pulse **Aplicaciones** → **Módulos SCA** en el panel de navegación de la consola administrativa.
   b. Seleccione el módulo que contenga las plantillas que desee detener.
   c. En Propiedades adicionales, pulse **Procesos de empresa** o **Tareas de usuario**, o ambos, según corresponda.
   d. Seleccione todas las plantillas de proceso y de tarea pulsando el recuadro de selección adecuado.
   e. Pulse **Detener**.

   Repita este paso para todos los módulos EJB que contengan plantillas de proceso de empresa o de tarea de usuario.
2. Verifique que están en ejecución: la base de datos, al menos un servidor de aplicaciones para cada clúster y el servidor autónomo donde se despliega la aplicación.

   En un entorno de Network Deployment, el servidor de despliegue, todos los servidores de aplicaciones autónomos gestionados y, como mínimo, un servidor de aplicaciones deben estar en ejecución para cada clúster donde esté instalada la aplicación.
3. Verifique que la aplicación no tenga instancias de proceso de empresa ni de tarea de usuario.

   Si es necesario, un administrador puede utilizar Business Process Choreographer Explorer para suprimir instancias o tareas de proceso.
4. Detenga y desinstale la aplicación:

   a. Pulse **Aplicaciones** → **Aplicaciones de empresa** en el panel de navegación de la consola administrativa.
   b. Seleccione la aplicación que desee desinstalar y pulse **Detener**.

      Este paso produce un error si todavía existe alguna instancia de proceso o de tarea en la aplicación.
   c. Seleccione de nuevo la aplicación que desea desinstalar y pulse **Deinstalar**.
   d. Pulse **Guardar** para guardar los cambios.

Resultado

Se desinstalará la aplicación.

Desinstalación de aplicaciones de procesos de empresa y tareas de usuario utilizando mandatos administrativos

Los mandatos administrativos proporcionan una alternativa a la consola administrativa para desinstalar aplicaciones que contienen procesos de empresa o tareas de usuario.

Antes de empezar

Para desinstalar una aplicación que contenga procesos de empresa o tareas de usuario, deben cumplirse los siguiente requisitos previos:

- Si la aplicación se ha instalado en un servidor autónomo, el servidor debe estar en ejecución y tener acceso a la base de datos de Business Process Choreographer.
- Si la aplicación se ha instalado en un clúster, el gestor de despliegue y al menos un miembro del clúster deben estar en ejecución. El miembro del clúster tiene acceso a la base de datos de Business Process Choreographer.
- Si la aplicación se ha instalado en un servidor gestionado, el gestor de despliegue y este servidor deben estar en ejecución. El servidor debe tener acceso a la base de datos de Business Process Choreographer.
- Todas las plantillas de proceso de empresa y tarea de usuario pertenecientes a la aplicación deben en estado detenido.
- No hay presentes instancias de plantillas de proceso de empresa ni de tarea de usuario en ningún estado.
-

Para entornos de servidor autónomo que se utilizan como entornos de desarrollo y de prueba de unidades, el servidor se puede configurar para ejecutarse en modalidad de desarrollo. Esta configuración no necesita que se detengan las plantillas ni que haya instancias. No obstante, esta configuración no es válida para entornos de producción.

Además, si está habilitada la seguridad global, verifique que el ID de usuario tiene autorización de operador.

Asegúrese de que el proceso servidor con el que se conecta el cliente de administración esté en ejecución. Para asegurarse de que el cliente administrativo conecte automáticamente con el proceso servidor, no utilice la opción `-conntype NONE` de opción de mandato.

Acerca de esta tarea

En los pasos siguientes se describe cómo utilizar el script bpcTemplates.jacl para desinstalar aplicaciones que contienen plantillas de procesos de empresa o plantillas de tareas de usuario. Por ejemplo, debe detener una plantilla para poder desinstalar la aplicación a la que pertenece. Puede utilizar el script bpcTemplates.jacl para detener y desinstalar plantillas en un paso.

Antes de desinstalar las aplicaciones, puede suprimir todas las instancias de proceso o de tarea asociadas a las plantillas de las aplicaciones, por ejemplo, utilizando Business Process Choreographer Explorer. También puede utilizar la opción**-force** con el script bpcTemplates.jacl para suprimir cualquier instancia asociada a las plantillas, detener las plantillas y desinstalarlas en un paso.

PRECAUCIÓN:
Dado que la opción -force suprime todos los datos de la instancia de proceso y la instancia de tareas, debe utilizar esta opción con precaución.

Procedimiento

1. Vaya al directorio de ejemplos de Business Process Choreographer.

   En las plataformas Windows, introduzca:

   ```
   cd raíz_instalación\ProcessChoreographer\admin
   ```

   En las plataformas Linux, UNIX e i5/OS, introduzca:

   ```
   cd raíz_instalación/ProcessChoreographer/admin
   ```

2. Detenga las plantillas y desinstale la aplicación correspondiente.

   En las plataformas Windows, introduzca:

   ```
   raíz_instalación\bin\wsadmin -f bpcTemplates.jacl
                                [-user nombre_usuario]
                                [-password contraseña de usuario]
                                -uninstall nombre_aplicación
                                [-force]
   ```

   En las plataformas Linux, UNIX e i5/OS, introduzca:

   ```
   raíz_instalación/bin/wsadmin -f bpcTemplates.jacl
                                [-user nombre_usuario]
                                [-password contraseña de usuario]
                                -uninstall nombre_aplicación
                                [-force]
   ```

   Donde:

   nombre_usuario
   Si está habilitada la seguridad global, proporcione el ID de usuario para la autenticación.

   contraseña_usuario
   Si está habilitada la seguridad global, proporcione la contraseña de usuario para la autenticación.

   nombre_aplicación
   Si está habilitada la seguridad global, proporcione la contraseña de usuario para la autenticación.

Resultado

Se desinstalará la aplicación.

Capítulo 6. Instalación de adaptadores

Los adaptadores permiten que la aplicación se comunique con otros componentes del sistema de información de empresa.

El proceso que se utiliza para instalar adaptadores se describe en Configuración y utilización de adaptadores en el centro de información de WebSphere Integration Developer.

Capítulo 7. Instalación de aplicaciones EIS

Un módulo de aplicación EIS puede desplegarse en una plataforma J2EE. El despliegue genera una aplicación, comprimida como un archivo EAR desplegado en el servidor. Se crean todos los artefactos y recursos J2EE, se configura la aplicación y está preparada para ejecutarse.

Acerca de esta tarea

El despliegue en la plataforma J2EE crea los artefactos y recursos J2EE siguientes:

Tabla 42. Correlación de enlaces con artefactos J2EE

Enlace en el módulo SCA	Artefactos J2EE generados	Recursos J2EE creados
Importación de EIS	Referencias de recursos generadas en el módulo EJB de sesión.	ConnectionFactory
Exportación de EIS	Bean controlado por mensajes, generado o desplegado, en función de la interfaz de receptor que el adaptador de recursos admita.	ActivationSpec
Importación de JMS	Se despliega el bean controlado por mensajes (MDB) proporcionado por el motor de ejecución, se generan las referencias a recursos en el módulo EJB de sesión. Observe que el MDB sólo se crea si la importación tiene un destino de recepción.	• ConnectionFactory • ActivationSpec • Destinations
Exportación de JMS	Se despliega el bean controlado por mensajes proporcionado por el motor de ejecución, se generan las referencias a recursos en el módulo EJB de sesión.	• ActivationSpec • ConnectionFactory • Destinations

Cuando la importación o exportación define un recurso como una ConnectionFactory, se genera la referencia al recurso en el descriptor de despliegue del módulo EJB de sesión sin estado. Además, se genera el enlace adecuado en el archivo de enlaces de EJB. El nombre, al que se enlaza la referencia al recurso, es el valor del atributo de destino, si hay uno presente, o el nombre de búsqueda JNDI dado al recurso, basándose en el nombre de módulo y el nombre de importación.

Al despliegue, la implementación localiza el bean de sesión del módulo y lo utiliza para buscar los recursos.

Durante el despliegue de la aplicación en el servidor, la tarea de instalación EIS comprobará si existe el recurso de elemento al que está enlazado. Si no existe y el archivo SCDL especifica al menos una propiedad, la tarea de instalación de EIS creará y configurará el recurso. Si no existe el recurso, no se llevará a cabo ninguna acción, se asume que se creará ese recurso antes de la ejecución de la aplicación.

Cuando se despliega la importación de JMS con un destino de recepción, se despliega un Bean controlado por mensajes (MDB). Éste escucha las respuestas a peticiones que se han enviado. Se asocia al MDB (escucha) el destino enviado con la petición en el campo de cabecera JMSreplyTo del mensaje JMS. Cuando llega el mensaje de respuesta, el MDB utiliza su ID de correlación para recuperar la información de devolución de llamada almacenada en el destino de devolución de llamada e invoca al objeto de devolución de llamada.

La tarea de instalación crea la ConnectionFactory y tres destinos de la información del archivo de importación. Además, crea la ActivationSpec para permitir que el MDB de ejecución escuche las respuestas en el destino de recepción. Las propiedades de la ActivationSpec se derivan de las propiedades Destination/ConnectionFactory. Si el proveedor de JMS es un adaptador de recursos SIBus, se crearán los destinos de SIBus correspondientes al destino de JMS.

Cuando se despliega la exportación de JMS, se despliega un Bean controlado por mensajes (MDB) (no el mismo MDB que el desplegado para la importación de JMS). Éste escucha las peticiones de entrada en el destino de recepción y envía las peticiones para que las procese el SCA. La tarea de instalación crea el conjunto de recursos similar al de la importación de JMS, una ActivationSpec, ConnectionFactory utilizado para enviar una respuesta a dos destinos. Todas las propiedades de estos recursos se especifican en el archivo de exportación. Si el proveedor de JMS es un adaptador de recursos SIBus, se crearán los destinos de SIBus correspondientes al destino de JMS.

Despliegue de módulos de aplicaciones EIS en la plataforma J2SE

Se puede desplegar el módulo EIS en una plataforma J2SE, no obstante, sólo se admitirá la importación de EIS.

Antes de empezar

Antes de comenzar esta tarea, es preciso crear un módulo de aplicación EIS con un enlace de importación JMS en el entorno de WebSphere Integration Development.

Acerca de esta tarea

Se debería facilitar un enlace de importación de JMS a un módulo de aplicación EIS si se desea acceder a sistemas EIS de forma asíncrona mediante el uso de colas de mensajes.

El despliegue en la plataforma J2SE es la única instancia donde se puede ejecutar la implementación del enlace en la modalidad no gestionado. El enlace JMS requiere soporte asíncrono y JNDI, ninguno de los cuáles se proporciona mediante la Service Component Architecture base o J2SE. J2EE Connector Architecture no admite la comunicación entrante no gestionada, eliminando con ello la exportación de EIS.

Cuando se despliega el módulo de aplicación EIS con la importación de EIS en J2SE, además de las dependencias de módulo, se tiene que especificar como dependencia el WebSphere Adapter que utiliza la importación, en el manifiesto o de cualquier otra forma admitida por SCA.

Despliegue de módulos de aplicaciones EIS en la plataforma J2EE

El despliegue de módulos EIS en la plataforma J2EE genera una aplicación, comprimida como un archivo EAR desplegado en el servidor. Se crean todos los artefactos y recursos J2EE, se configura la aplicación y está preparada para ejecutarse.

Antes de empezar

Antes de comenzar esta tarea, es preciso crear un módulo EIS con un enlace de importación JMS en el entorno de WebSphere Integration Development.

Acerca de esta tarea

El despliegue en la plataforma J2EE crea los artefactos y recursos J2EE siguientes:

Tabla 43. Correlación de enlaces con artefactos J2EE

Enlace en el módulo SCA	Artefactos J2EE generados	Recursos J2EE creados
Importación de EIS	Referencias de recursos generadas en el módulo EJB de sesión.	ConnectionFactory
Exportación de EIS	Bean controlado por mensajes, generado o desplegado, en función de la interfaz de receptor que el adaptador de recursos admita.	ActivationSpec
Importación de JMS	Se despliega el bean controlado por mensajes (MDB) proporcionado por el motor de ejecución, se generan las referencias a recursos en el módulo EJB de sesión. Observe que el MDB sólo se crea si la importación tiene un destino de recepción.	• ConnectionFactory • ActivationSpec • Destinations
Exportación de JMS	Se despliega el bean controlado por mensajes proporcionado por el motor de ejecución, se generan las referencias a recursos en el módulo EJB de sesión.	• ActivationSpec • ConnectionFactory • Destinations

Cuando la importación o exportación define un recurso como una ConnectionFactory, se genera la referencia al recurso en el descriptor de despliegue del módulo EJB de sesión sin estado. Además, se genera el enlace adecuado en el archivo de enlaces de EJB. El nombre, al que se enlaza la referencia al recurso, es el valor del atributo de destino, si hay uno presente, o el nombre de búsqueda JNDI dado al recurso, basándose en el nombre de módulo y el nombre de importación.

Al despliegue, la implementación localiza el bean de sesión del módulo y lo utiliza para buscar los recursos.

Durante el despliegue de la aplicación en el servidor, la tarea de instalación EIS comprobará si existe el recurso de elemento al que está enlazado. Si no existe y el archivo SCDL especifica al menos una propiedad, la tarea de instalación de EIS

creará y configurará el recurso. Si no existe el recurso, no se llevará a cabo ninguna acción, se asume que se creará ese recurso antes de la ejecución de la aplicación.

Cuando se despliega la importación de JMS con un destino de recepción, se despliega un Bean controlado por mensajes (MDB). Éste escucha las respuestas a peticiones que se han enviado. Se asocia al MDB (escucha) el destino enviado con la petición en el campo de cabecera JMSreplyTo del mensaje JMS. Cuando llega el mensaje de respuesta, el MDB utiliza su ID de correlación para recuperar la información de devolución de llamada almacenada en el destino de devolución de llamada e invoca al objeto de devolución de llamada.

La tarea de instalación crea la ConnectionFactory y tres destinos de la información del archivo de importación. Además, crea la ActivationSpec para permitir que el MDB de ejecución escuche las respuestas en el destino de recepción. Las propiedades de la ActivationSpec se derivan de las propiedades Destination/ConnectionFactory. Si el proveedor de JMS es un adaptador de recursos SIBus, se crearán los destinos de SIBus correspondientes al destino de JMS.

Cuando se despliega la exportación de JMS, se despliega un Bean controlado por mensajes (MDB) (no el mismo MDB que el desplegado para la importación de JMS). Éste escucha las peticiones de entrada en el destino de recepción y envía las peticiones para que las procese el SCA. La tarea de instalación crea el conjunto de recursos similar al de la importación de JMS, una ActivationSpec, ConnectionFactory utilizado para enviar una respuesta a dos destinos. Todas las propiedades de estos recursos se especifican en el archivo de exportación. Si el proveedor de JMS es un adaptador de recursos SIBus, se crearán los destinos de SIBus correspondientes al destino de JMS.

Capítulo 8. Resolución de problemas de un despliegue anómalo

Este tema describe los pasos que deben realizarse para determinar la causa de un problema al desplegar una aplicación. También presenta algunas soluciones posibles.

Antes de empezar

Este tema da por supuestas las afirmaciones siguientes:
- El usuario tiene una comprensión básica de cómo depurar un módulo.
- El registro cronológico y el rastreo es activo mientras se despliegue el módulo.

Acerca de esta tarea

La tarea de resolución de problemas de un despliegue se inicia después de recibir la notificación de un error. Hay varios síntomas de un despliegue anómalo que tiene que inspeccionarse antes de emprender una acción.

Procedimiento

1. Determine si la instalación de aplicación ha sido anómala.

   Examine si hay mensajes que especifiquen la causa de la anomalía en el archivo SystemOut.log. Algunos de los motivos de que es posible que no se instale una aplicación son los siguientes:
   - Intenta instalar una aplicación en varios servidores en la misma célula de Network Deployment.
   - Una aplicación tiene el mismo nombre que un módulo existente en la célula de Network Deployment en la que se instala la aplicación.
   - Intenta desplegar módulos J2EE en un archivo EAR en servidores de destino diferentes.

   Importante: Si se ha producido un error en la instalación y la aplicación contiene servicios, debe eliminar los destinos de SIBus o las especificaciones de activación J2C creados antes de la anomalía antes de intentar volver a instalar la aplicación. El modo más sencillo de eliminar estos artefactos es pulsar **Guardar > Descartar todo** después de la anomalía. Si guarda los cambios sin querer, debe eliminar manualmente los destinos de SIBus y las especificaciones de activación J2C (consulte Supresión de los destinos de SIBus y Supresión de las especificaciones de activación J2C de la sección Administración).
2. Si la aplicación se ha instalado correctamente, examínela para determinar si se ha iniciado de manera satisfactoria.

   Si la aplicación no se ha iniciado satisfactoriamente, la anomalía se ha produjo cuando el servidor intentó iniciar los recursos para la aplicación.
   a. Examine si hay mensajes que le orienten sobre cómo continuar en el archivo SystemOut.log.
   b. Determine si los recursos que necesita la aplicación están disponibles y/o se han iniciado satisfactoriamente.

      Los recursos no iniciados impiden que se ejecute una aplicación. Esta acción protege la información perdida. Los motivos de que no se inicie un recurso son, entre otros:

- Los enlaces se especifican incorrectamente
- Los recursos no se configuran correctamente
- Los recursos no se incluyen en el archivo RAR (de archivo de recursos)
- Los recursos Web no incluidos en el archivo WAR (de archivo de servicios)

c. Determine si faltan componentes.

El motivo de que falte un componente es un archivo EAR (de archivo de empresa) construido incorrectamente. Asegúrese de que todos los componentes que necesita el módulo se encuentren en las carpetas correctas del sistema de prueba en el que ha compilado el archivo JAR (Java Archive). "Preparación para desplegar en un servidor" contiene información adicional.

3. Examine la aplicación para ver si hay información que fluya a través de ella.

Incluso una aplicación en ejecución puede dejar de procesar información. Las razones de ello son similares a las mencionadas en el paso 2b en la página 207.

a. Determine si la aplicación utiliza servicios incluidos en otra aplicación. Asegúrese de que la otra aplicación esté instalada y se haya iniciado satisfactoriamente.

b. Determine si los enlaces de importación y exportación de los dispositivos contenidos en otras aplicaciones que la aplicación con anomalía utiliza están configurados correctamente. Utilice la consola administrativa para examinar y corregir los enlaces.

4. Corrija el problema y reinicie la aplicación.

Supresión de las especificaciones de activación J2C

El sistema construye especificaciones de aplicación J2C al instalar una aplicación que contenga servicios. Hay ocasiones en que debe suprimir estas especificaciones antes de volver a instalar la aplicación.

Antes de empezar

Si suprime la especificación a causa de una instalación de una aplicación con anomalías, asegúrese de que el módulo del nombre JNDI (Java Naming and Directory Interface) coincida con el nombre del módulo que no se ha podido instalar. La segunda parte del nombre JNDI es el nombre del módulo que ha implementado el destino. Por ejemplo, en sca/SimpleBOCrsmA/ActivationSpec, **SimpleBOCrsmA** es el nombre de módulo.

Rol de seguridad necesario para esta tarea: Cuando está habilitada la seguridad y la autorización según el rol, debe haber iniciado la sesión como administrador o configurador para realizar esta tarea.

Acerca de esta tarea

Suprima especificaciones de activación de J2C cuando guarde de forma inadvertida una configuración después de instalar una aplicación que contenga servicios y no requiera las especificaciones.

Procedimiento

1. Localice la especificación de la activación que desea suprimir.

Las especificaciones están contenidas en el panel del adaptador de recursos. Para ir a este panel, pulse **Recursos > Adaptadores de recursos**.

a. Localice el **Adaptador de recursos SPI de componente de mensajería de plataforma.**

   Para localizar este adaptador, debe estar en el ámbito **nodo** de un servidor autónomo o en el ámbito **servidor** de un entorno de despliegue.

2. Visualice las especificaciones de activación de J2C asociadas con el Adaptador de recursos SPI de componente de mensajería de plataforma.

   Pulse en el nombre de adaptador de recursos y el panel siguiente muestra las especificaciones asociadas.

3. Suprima todas las especificaciones con un **Nombre JNDI** que coincidan con el nombre de módulo que va a suprimir.
   a. Pulse el recuadro de selección situado junto a las especificaciones adecuadas.
   b. Pulse **Suprimir**.

Resultado

El sistema elimina las especificaciones seleccionadas de la pantalla.

Qué hacer a continuación

Guarde los cambios.

Supresión de los destinos de SIBus

Los destinos de SIBus son las conexiones que ponen los servicios a disposición de las aplicaciones. Hay ocasiones en que tendrá que eliminar destinos.

Antes de empezar

Si suprime el destino a causa de la instalación de una aplicación con anomalías, asegúrese de que el módulo del nombre de destino coincida con el nombre del módulo que no se ha podido instalar. La segunda parte del destino es el nombre del módulo que ha implementado el destino. Por ejemplo, en sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer, **SimpleBOCrsmA** es el nombre de módulo.

Rol de seguridad necesario para esta tarea: Cuando está habilitada la seguridad y la autorización según el rol, debe haber iniciado la sesión como administrador o configurador para realizar esta tarea.

Acerca de esta tarea

Suprima destinos de SIBus cuando guarde una configuración sin darse cuenta después de instalar una aplicación que contenga servicios o ya no necesite los destinos.

Nota: Esta tarea sólo suprime el destino del bus de sistema SCA. También debe eliminar las entradas del bus de aplicación antes de volver a instalar una aplicación que contenga servicios (consulte Supresión de las especificaciones de activación de J2C en la sección Administración de este centro de información).

Procedimiento

1. Inicie la sesión en la consola administrativa.
2. Visualice los destinos en el bus del sistema SCA.

Desplácese hasta el panel pulsando **Integración de servicios > Buses**

3. Seleccione los destinos de bus del sistema SCA.

   En la pantalla, pulse en **SCA.SYSTEM.*nombrecélula*.Bus**, donde *nombrecélula* es el nombre de la célula que contiene el módulo con los destinos que va a suprimir.

4. Suprima los destinos que contienen un nombre de módulo que coincida con el módulo que va a eliminar.

   a. Pulse en el recuadro de selección situado junto a los destinos pertinentes.

   b. Pulse **Suprimir**.

Resultado

El panel sólo muestra los destinos restantes.

Qué hacer a continuación

Suprima las especificaciones de activación J2C relacionadas con el módulo que ha creado estos destinos.

Parte 3. Apéndices

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en Estados Unidos.

Puede que IBM no proporcione los productos, servicios o funciones tratados en este documento en otros países. Consulte al representante de IBM de su localidad para obtener información acerca de los productos y servicios que están actualmente disponibles en su localidad. Cualquier referencia a un producto, programa o servicio de IBM no pretende indicar ni implica que sólo se pueda utilizar este producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM

IBM puede tener patentes o aplicaciones pendientes de patente que conciernan al tema descrito en este documento. El suministro de este documento no le otorga ninguna licencia sobre estas patentes. Puede enviar preguntas acerca de licencias por escrito a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias relativas a la información de doble byte (DBCS), póngase en contacto con el Departamento de propiedad intelectual de IBM de su país o envíe sus consultas, por escrito, a:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones contradigan la legislación local:INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN ″TAL CUAL″ SIN GARANTÍA DE NINGÚN TIPO, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUYENDO, PERO SIN LIMITARSE A, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN, COMERCIALIZACIÓN O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas legislaciones no contemplan la exclusión de garantías, explícitas o implícitas en algunas transacciones, por lo que puede haber usuarios a los que no les afecte dicha declaración.

Esta publicación puede contener imprecisiones técnicas o errores tipográficos. La información que ofrece está sometida a modificaciones periódicas, las cuales se van incorporando en ediciones posteriores. IBM puede reservarse el derecho de realizar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento sin previo aviso.

Cualquier referencia en esta información a sitios Web que no son de IBM se proporciona solamente para su comodidad y no equivale de ninguna manera a una

aprobación de esos sitios Web. Los materiales de esos sitios Web no forman parte de los materiales de este producto de IBM y la utilización de esos sitios Web se realiza bajo el propio riesgo del usuario.

IBM puede utilizar o distribuir cualquier información que el usuario le proporcione de la manera que considere adecuada sin incurrir en ninguna obligación con el usuario.

Los usuarios autorizados de este programa que deseen tener información sobre el mismo con el propósito de posibilitar: (i) el intercambio de información entre programas creados independientemente y otros programas (incluyendo éste) y (ii) la utilización mutua de la información que se ha intercambiado, deben ponerse en contacto con:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
EE.UU.

Esta información puede estar disponible, bajo las condiciones y los términos adecuados, incluyendo en algunos casos, el pago de una cuota.

El programa bajo licencia que se describe en este documento y todo el material bajo licencia que se encuentra disponible para el programa se proporcionan de acuerdo con los términos del Acuerdo del Cliente de IBM, el Acuerdo Internacional de Licencia de Programas o cualquier acuerdo equivalente entre IBM y el Cliente.

Cualquier información de rendimiento contenida aquí fue determinada en un entorno controlado. Por tanto, los resultados obtenidos en otros entornos operativos pueden variar de forma significativa. Pueden haberse realizado algunas mediciones en sistemas en nivel de desarrollo y no existen garantías de que estas mediciones sean las mismas en sistemas disponibles para todos los usuarios. Además, algunas mediciones pueden haberse calculado mediante extrapolaciones. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los suministradores de estos productos, sus anuncios publicados u otras fuentes disponibles para el público. IBM no ha probado estos productos y no puede confirmar la precisión del rendimiento, compatibilidad y otras afirmaciones relacionadas con productos que no son de IBM. Las preguntas acerca de las posibilidades de productos que no son de IBM deben dirigirse a los suministradores de estos productos.

Todas las declaraciones referentes a acciones e intenciones futuras de IBM pueden cambiar o ser retiradas sin previo aviso y solamente representan objetivos.

Esta información contiene ejemplos de datos e informes utilizados en operaciones cotidianas de negocios. Para ilustrarlos de la manera más completa posible, los ejemplos incluyen nombres de personas, compañías, marcas y productos. Todos estos nombres son ficticios y cualquier parecido con nombres y direcciones utilizadas por una empresa de negocios real es mera coincidencia.

LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente, que ilustran cómo se realiza la programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo en cualquier formato sin que tenga que pagar a IBM, a fin de desarrollar, utilizar, comercializar o distribuir programas de aplicación adaptados a la interfaz de programación de aplicaciones para la plataforma operativa para la que se han escrito los programas de ejemplo. Estos ejemplos no se han probado a fondo en todas las condiciones. Por consiguiente, IBM no puede garantizar ni implicar la fiabilidad, la capacidad de servicio o el funcionamiento de estos programas.

Todas las copias o fragmentos de estos programas de ejemplo o cualquier trabajo derivado, deben incluir un aviso de copyright como se muestra a continuación: (c) (nombre de la empresa) (año). Algunos fragmentos de este código se derivan de IBM Corp. Sample Programs. (c) Copyright IBM Corp. _especifique el año o los años_. Reservados todos los derechos.

Si ve esta información en copia software, es posible que no aparezcan las fotografías y las ilustraciones en color.

Información de interfaz de programación

La información de interfaz de programación, si se proporciona, está pensada para ayudarle a crear software de aplicación utilizando este programa.

Las interfaces de programación de uso general permiten escribir software de aplicación que obtienen los servicios de las herramientas de este programa.

Sin embargo, este información puede contener también información de diagnosis, modificación y ajuste. La información de diagnosis, modificación y ajuste se proporciona para ayudarle a depurar el software de aplicación.

Aviso: no utilice esta información de diagnosis, modificación y ajuste como interfaz de programación porque está sujeta a cambios.

Marcas registradas y marcas de servicio

IBM, el logotipo de IBM, developerWorks, WebSphere y z/OS son marcas registradas de International Business Machines Corporation en los Estados Unidos y/o en otros países.

Adobe es una marca registrada de Adobe Systems Incorporated en los Estados Unidos y/o en otros países.

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.

Otros nombres de compañías, productos o servicios pueden ser marcas registradas o de servicio de terceros.

Este producto incluye software desarrollado por Eclipse Project (http://www.eclipse.org).

IBM WebSphere Process Server for Multiplatforms, Versión 6.1.0