WebSphere® Process Server for Multiplatforms

版本 6.1.0

开发和部署模块

WebSphere® Process Server for Multiplatforms

版本 6.1.0

开发和部署模块

注意

在使用本资料之前，请务必阅读文档末尾的声明一节中的一般信息。

2008 年 2 月 1 日

此版本适用于 WebSphere Process Server for Multiplatforms V6.1.0（产品编号为 5724-L01）及所有后续发行版和修订版，直到在新版本中另有声明为止。

要向我们发送您对本文档的意见，请向 ctscrcf@cn.ibm.com 发送电子邮件消息。我们期待您的宝贵意见。

当您发送信息给 IBM 后，即授予 IBM 非专有权，IBM 可以它认为合适的任何方式使用或分发此信息，而无须对您承担任何责任。

目录

图

表

第 1 部分 开发应用程序

第 1 章 模块开发概述

模块是 WebSphere® Process Server 应用程序的基本部署单元。模块包含应用程序使用的一个或多个组件库和登台模块。组件可以引用其他服务组件。开发模块时，需要确保应用程序所需的组件、登台模块和库（模块引用的工件集合）都在生产服务器上。

开发将部署到 WebSphere Process Server 的模块时，使用的主要工具是 WebSphere Integration Developer。虽然可以在其他环境中开发模块，但使用 WebSphere Integration Developer 最好。

WebSphere Process Server 支持两类服务模块：业务服务模块和调解模块。业务服务模块实现了流程的逻辑。调解模块将服务调用转换为目标所能理解的格式、将该请求传递至目标并将结果返回给发起者，从而实现了应用程序之间的通信。

下列各节描述了如何实现和更新 WebSphere Process Server 上的模块。

组件概要

组件是用于封装可复用业务逻辑的基本构建块。服务组件与接口、引用和实现相关联。接口定义服务组件与调用组件之间的约定。通过 WebSphere Process Server，服务模块可以导出服务组件以供其他模块使用，也可以导入要使用的服务组件。为了调用服务组件，调用模块对该服务组件的接口进行引用。对接口的引用是通过配置从调用模块到相关接口的引用解析的。

要开发模块，必须执行下列操作:

1. 定义模块中组件的接口。
2. 定义、修改或处理服务组件所使用的业务对象。
3. 通过服务组件的接口来定义或修改该服务组件。

   注: 服务组件是通过它的接口定义的。
4. （可选）导出或导入服务组件。
5. 创建 EAR 文件，用来安装采用组件的模块。使用 WebSphere Integration Developer 中的“导出 EAR”功能或者使用 serviceDeploy 命令来创建 EAR 文件，该文件可用于安装使用了服务组件的服务模块。

开发类型

WebSphere Process Server 提供了组件编程模型来方便您进行面向服务的编程。要使用这种模型，提供者导出服务组件的接口，以便使用者能够导入那些接口，并像本地组件一样使用该服务组件。开发者使用强类型接口或动态类型接口来实现或调用该服务组件。本信息中心的“参考”部分对接口及其方法作了描述。

在将服务模块安装到服务器后，可以使用管理控制台来更改从应用程序中进行的引用的目标组件。新目标必须接受相同的业务对象类型，并且必须执行从应用程序中进行的引用所请求的操作。

服务组件开发注意事项

在开发服务组件时，您应该询问自己下列问题:

- 此服务组件将被导出并由另一个模块使用吗？

  如果是这样，请确保为该组件定义的接口可以被另一模块使用。
- 该服务组件的运行时间相对较长吗？

  如果是这样，请考虑实现该服务组件的异步接口。
- 使该服务组件分散化是否有益？

  如果是这样，请考虑让部署在服务器集群上的服务模块包含该服务组件的副本，以便从并行处理中受益。
- 应用程序是否既需要一阶段落实资源也需要两阶段落实资源？

  如果是这样，请确保对该应用程序启用最后参与者支持。

  注: 如果使用 WebSphere Integration Developer 来创建应用程序，或者使用 serviceDeploy 命令来创建可安装的 EAR 文件，这些工具就会自动地对该应用程序启用此支持。请参阅 WebSphere Application Server Network Deployment 信息中心中的“在同一个事务中使用单阶段落实和两阶段落实资源”主题。

开发服务模块

服务组件必须包含在服务模块中。向其他模块提供服务的关键在于开发服务模块以包含服务组件。

开始之前

本任务假定需求分析表明实现服务组件以供其他模块使用能产生有益的结果。

关于此任务

在分析需求后，您可能会确定提供并使用服务组件是高效的信息处理方法。如果您确定可复用的服务组件将使环境受益，请创建服务模块以包含服务组件。

过程

1. 标识其他模块可以使用的服务组件。

   标识服务组件后，就可以继续开发服务组件。
2. 标识应用程序中哪些服务组件可以使用其他服务模块中的服务组件。

   标识服务组件及其目标组件后，就可以继续调用组件。
3. 通过连线使客户机组件与目标组件相连。

开发服务组件

开发服务组件以便为服务器中的多个应用程序提供可复用的逻辑。

开始之前

本任务假定您已开发并标识了可用于多个模块的处理方法。

关于此任务

一个服务组件可以由多个模块使用。如果导出服务组件，那么其他模块就可通过接口引用该服务组件。本任务描述了如何构建服务组件以供其他模块使用。

注: 一个服务组件可以包含多个接口。

过程

1. 定义数据对象以便在调用者与该服务组件之间移动数据。

   数据对象及其类型是调用者与服务组件之间的接口的组成部分。
2. 定义调用者在引用该服务组件时要使用的接口。

   此接口定义指定了服务组件并列示了该服务组件中的任何可用方法。
3. 开发用于定义实现的类。
   - 如果该组件的运行时间较长（或者是异步组件），那么转到步骤 4。
   - 如果该组件的运行时间不长（或者是同步组件），那么转到步骤 5。
4. 开发异步实现。

   要点: 不能将异步组件接口的 joinsTransaction 属性设置为 `true`。
   a. 定义代表该同步服务组件的接口。
   b. 定义服务组件的实现。
   c. 转到步骤 6。
5. 开发同步实现。
   a. 定义代表该同步服务组件的接口。
   b. 定义服务组件的实现。
6. 将组件接口和实现保存到扩展名为 .java 的文件中。
7. 将该服务模块以及必需的资源打包到 JAR 文件。

   请参阅本信息中心中的“将模块部署到生产服务器”以获取步骤 7 到 9 的描述。
8. 运行 serviceDeploy 命令以创建包含该应用程序的可安装 EAR 文件。
9. 在服务器节点上安装该应用程序。
10. 可选: 如果调用另一服务模块中的服务组件，那么请配置调用者与相应服务组件之间的连线。

    本信息中心的“管理”部分描述了连线的配置方法。

组件开发示例

本示例说明实现了单个方法 CustomerInfo 的同步服务组件。第一部分定义该服务组件的接口，该接口实现了名为 getCustomerInfo 的方法。

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

以下代码块实现了该服务组件。

```
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;
 }
}
```

本示例开发异步服务组件。第一部分的代码定义该服务组件的接口，该接口实现了名为 getQuote 的方法。

```
public interface StockQuote {

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

以下代码段是与 StockQuote 相关联的类的实现。

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

以下代码段实现了异步接口 StockQuoteAsync。

```
public interface StockQuoteAsync {

 // deferred response
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);

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

以下代码段是 StockQuoteCallback 接口，它定义了 onGetQuoteResponse 方法。

```
public interface StockQuoteCallback {

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

下一步做什么?

调用该服务。

调用组件

包含模块的组件可以使用 WebSphere Process Server 集群中任何节点上的组件。

开始之前

在调用组件之前，请确保已将包含该组件的模块安装到 WebSphere Process Server 上。

关于此任务

组件可以通过使用 WebSphere Process Server 集群中任何服务组件的名称并传递该组件所需的数据类型来使用那些组件。在此环境中调用组件包括查找所需组件并接着创建对该组件的引用。

注：模块中的组件可以调用同一模块中的其他组件，这称为模块内调用。外部调用（模块间调用）是通过在提供组件中导出接口并在调用组件中导入该接口实现的。

要点：如果调用的组件不在运行调用模块的服务器上，那么必须对服务器进行其他配置。需要执行的配置工作取决于是以异步方式还是以同步方式调用该组件。相关任务对本例中的应用程序服务器配置方式作了描述。

过程

1. 确定调用模块所需的组件。

   记录该组件中的接口名以及该接口所需的数据类型。
2. 定义数据对象。

   虽然输入或返回的内容可以是 Java™ 类，但最好使用服务数据对象。
3. 查找该组件。
   a. 使用 ServiceManager 类来获取对调用模块的引用。
   b. 使用 locateService() 方法来查找该组件。

      根据组件的不同，该接口可以是 Web 服务描述符语言（WSDL）端口类型接口或 Java 接口。
4. 以同步方式或异步方式调用该组件。

   可以通过 Java 接口来调用该组件，也可以使用 invoke() 方法来以动态方式调用该组件。
5. 处理返回的结果。

   该组件可能会生成异常，因此客户机必须能够处理这种可能性。

组件调用示例

以下示例创建 ServiceManager 类。

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

以下示例使用 ServiceManager 类来从包含组件引用的文件中获取组件列表。

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

以下代码查找实现了 StockQuote Java 接口的组件。

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

以下代码查找实现了 Java 或 WSDL 端口类型接口的组件。调用模块使用 Service 接口与该组件进行交互。

提示: 如果该组件实现了 Java 接口，那么可以通过该接口或 invoke() 方法来调用该组件。

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

以下示例说明了调用另一个组件的 MyValue 代码。

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

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

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

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

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

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

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

下一步做什么?

配置调用模块引用与组件接口之间的连线。

以动态方式调用组件

当模块调用包含 Web 服务描述符语言（WSDL）端口类型接口的组件时，该模块必须使用 invoke() 方法来以动态方式调用该组件。

开始之前

本任务假定调用组件以动态方式调用另一个组件。

关于此任务

对于 WSDL 端口类型接口，调用组件必须使用 invoke() 方法来调用该组件。调用模块也可以通过这种方法来调用包含 Java 接口的组件。

过程

1. 确定包含所需组件的模块。
2. 确定该组件所需的数组。

   输入数组可以具有下列三种类型的某一种:
   - 大写的 Java 基本类型或此类型的数组
   - 普通的 Java 类或者这些类的数组
   - 服务数据对象（SDO）
3. 定义一个数组以包含该组件返回的响应。

   响应数组的类型可以与输入数组相同。
4. 使用 invoke() 方法来调用所需的组件，并将该数组对象传递给该组件。
5. 处理结果。

动态调用组件的示例

在以下示例中，一个模块使用 invoke() 方法来调用使用了大写 Java 基本数据类型的组件。

```
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);
```

以下示例使用 invoke 方法并将 WSDL 端口类型接口用作目标。

```
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 encapsulates all the parameters of methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

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

有关将模块与目标隔离的概述

在开发模块时，您将标识可以由多个模块使用的服务。以此方式使用服务可以最大程度地缩短开发周期并降低开发成本。如果一个服务由许多模块使用，那么应该将调用模块与目标隔离，这样，即使将目标升级，切换到新服务的行为对于调用模块来说也

是透明的。本主题将简单调用模型和隔离调用模型进行对比，并提供了一个示例来说明隔离的用途。特定的示例中描述的方法，并不是将模块与目标隔离的唯一方法。

简单调用模型

在开发模块时，通过将服务导入到模块中并进行调用，可以使用其他模块中的服务。导入的服务将连接到由 WebSphere Integration Developer 中的另一模块导出的服务或者通过在管理控制台中绑定该服务导出的服务。简单调用模型对此模型进行了说明。

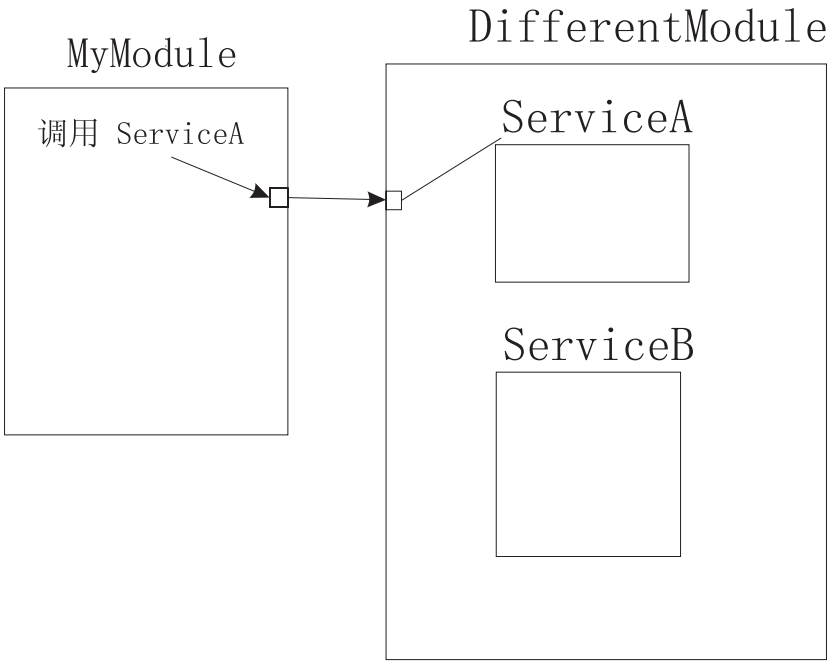


图 1. 简单调用模型

隔离调用模型

要在不停止调用模块的情况下更改调用目标，请将调用模块与调用目标隔离。这样，由于更改的是下游目标而不是模块本身，所以在更改目标期间，模块可以继续执行处理。应用程序隔离示例说明了隔离使您能够在不影响调用模块状态的情况下更改目标。

应用程序隔离示例

在使用简单调用模型时，多个调用同一服务的模块就像是多个调用单一服务的应用程序。MODA、MODB 和 MODC 全都调用 CalculateFinalCost。

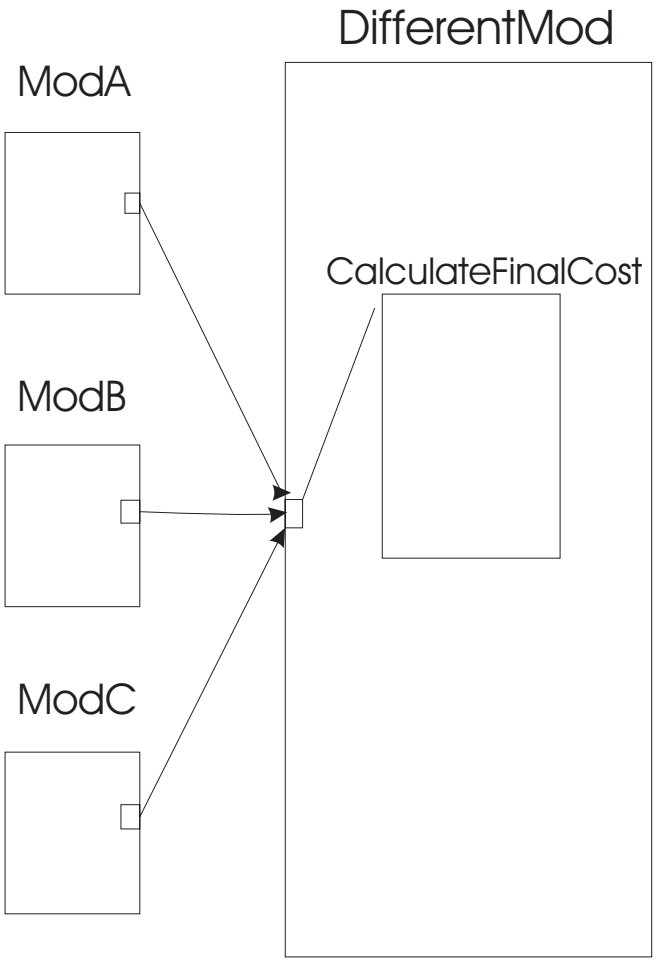


图 2. 多个调用单一服务的应用程序

CalculateFinalCost 提供的服务需要进行更新，以使新成本反映在所有使用该服务的模块中。开发小组构建并测试新服务 UpdatedCalculateFinal 以合并更改。您已准备好在生产环境中使用新服务。在未进行隔离时，必须将所有调用了 CalculateFinalCost 的模块更新为调用 UpdateCalculateFinal。进行隔离后，只需要更改用于将缓冲模块连接到目标的绑定。

注: 如果以此方式更改服务，就可以继续向其他需要使用原始服务的模块提供该服务。

进行隔离后，就在应用程序与目标之间创建了缓冲模块（请参阅调用了 UpdateCalculateFinal 的隔离调用模型）。

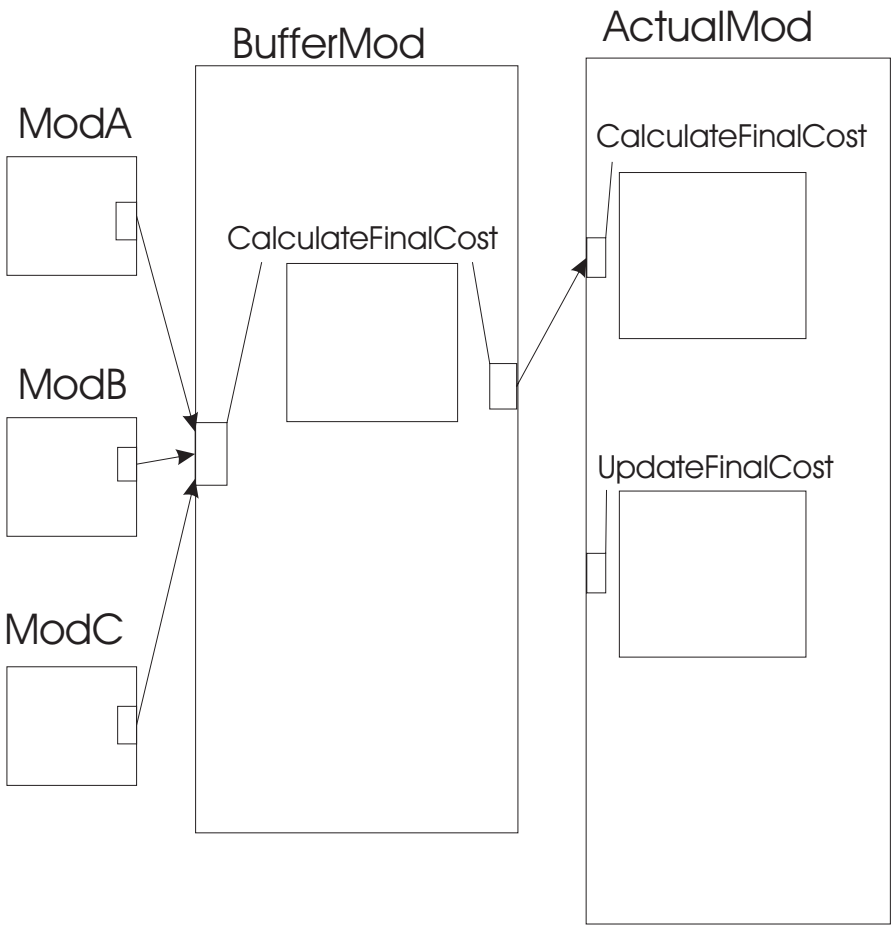


图 3. 调用了 UpdateCalculateFinal 的隔离调用模型

对于此模型，调用模块不更改，而只需要更改从缓冲模块导入到目标的绑定（请参阅调用了 UpdatedCalculateFinal 的隔离调用模型）。

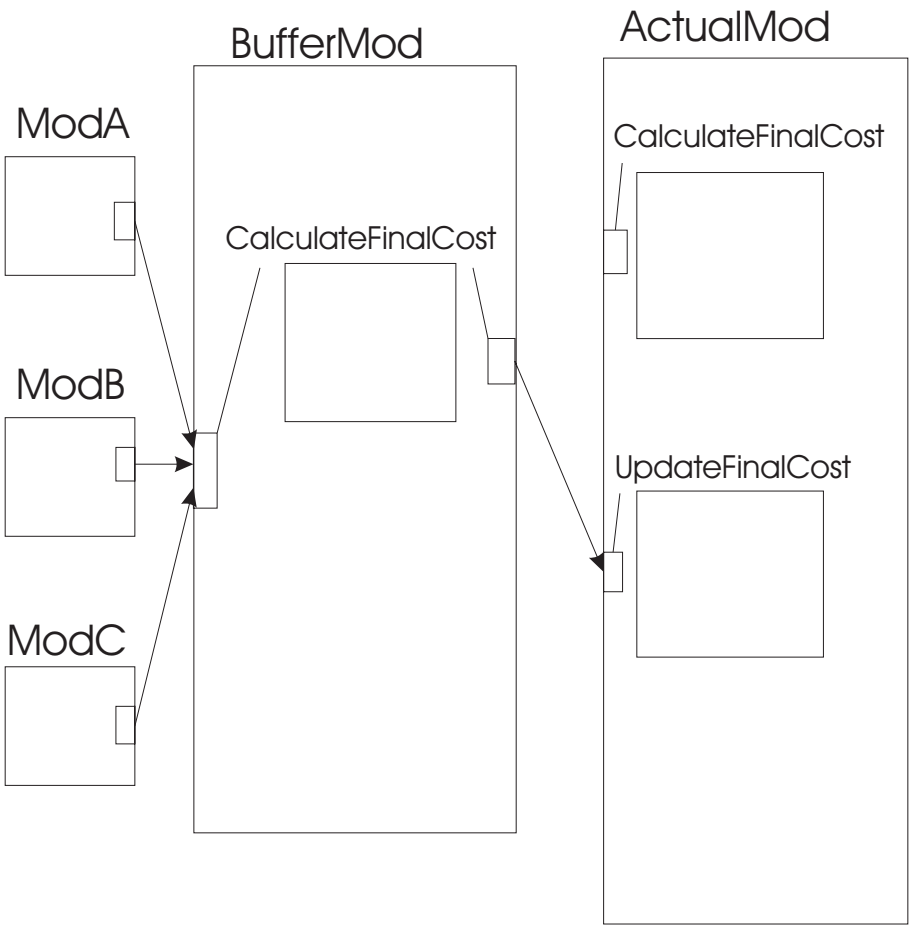


图 4. 调用了 UpdatedCalculateFinal 的隔离调用模型

如果缓冲模块以同步方式调用目标，在重新启动缓冲模块（无论是调解模块还是业务服务模块）时，返回到原始应用程序的结果都来自新目标。如果缓冲模块以异步方式调用目标，那么下次进行调用时，返回到原始应用程序的结果将来自新目标。

相关任务

更改目标

更改引用目标后，由于无需重新编译和重新安装应用程序，所以使应用程序能够灵活地利用组件中的提升。

HTTP 绑定

HTTP 绑定设计用于提供与 HTTP 的服务组件体系结构（SCA）连接。这允许现有或新开发的 HTTP 应用程序参与面向服务的体系结构（SOA）环境。

此外，SCA 运行时环境的网络可以在现有 HTTP 基础结构上进行通信。

HTTP 绑定展示了多种 HTTP 功能:

- 消息采用一种保留 HTTP 格式和消息头信息的方式向中间组件呈示。这为 HTTP 应用程序员、用户和管理员提供了更为熟悉的视图。

- 现有数据绑定框架已为 HTTP 协定进行扩展，并提供 SCA 消息和 HTTP 消息头和主体之间的映射。
- 导入和导出可配置为支持一系列常见 HTTP 功能。
- 当安装包含 HTTP 导入或导出的 SCA 模块时，运行时环境会自动适当地配置为允许与 HTTP 连接。

有关创建 HTTP 导入和导出的详细指示信息可在 信息中心中找到: **WebSphere Integration Developer > 开发集成应用程序 > HTTP 数据绑定**。

相关任务

显示 HTTP 绑定

在部署应用程序后，您可能想要检查 HTTP 绑定，以确保它们是正确的。

更改 HTTP 导出绑定

管理控制台允许您更改 HTTP 导出绑定的配置，而无需更改原始源代码并重新部署应用程序。

更改 HTTP 导入绑定

管理控制台允许您更改 HTTP 导入绑定的配置，而无需更改原始源代码并重新部署应用程序。

覆盖生成的服务组件体系结构实现

有时，系统创建的介于 Java 代码与服务数据对象（SDO）之间的转换可能不符合您的需要。使用此过程将缺省服务组件体系结构（SCA）类实现替换为自己的服务组件体系结构（SCA）类实现。

开始之前

确保您已使用 WebSphere Integration Developer 或 genMapper 命令来生成 Java 至 Web Service 定义语言（WSDL）类型转换。

关于此任务

通过将生成的代码替换为符合需要的代码，您可覆盖将 Java 类型映射至 WSDL 类型的已生成组件。如果已定义自己的 Java 类，那么考虑使用您自己的映射。使用此过程来进行更改。

过程

1. 查找生成的组件。 该组件命名为 *java_class*Mapper.component。
2. 使用文本编辑器编辑组件。
3. 注释掉生成的代码并提供自己的方法。

   切勿更改包含组件实现的文件名。

这是要替换的已生成组件的示例:

```
 private DataObject javatodata_setAccount_output(Object myAccount) {

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


 return SDOJavaObjectMediator.java2Data(myAccount);


}
```

将组件和其他文件复制到所包含模块驻留的目录，再连接 WebSphere Integration Developer 中的组件或使用 serviceDeploy 命令生成企业归档（EAR）文件。

相关概念

第 16 页的『用于 Java 至服务数据对象转换的运行时规则』
要正确地覆盖生成的代码或确定与 Java 至服务数据对象（SDO）转换相关的可能的运行时异常，了解所涉及的规则十分重要。大部分转换都是直接了当的，但也存在一些复杂的情况，运行时在转换生成的代码时会提供最佳的可能性。

相关参考

Java 至 XML 转换

系统使用预定义的规则生成基于 Java 类型的 XML。

genMapper 命令

使用 genMapper 命令来生成将服务组件体系结构（SCA）引用连接到 Java 接口的组件。

覆盖服务数据对象至 Java 转换

有时，系统创建的介于服务数据对象（SDO）和 Java 类型对象之间的转换可能不符合您的需要。使用此过程将缺省实现替换为自己的实现。

开始之前

确保您已使用 WebSphere Integration Developer 或 genMapper 命令来生成 WSDL 至 Java 类型转换。

关于此任务

通过将生成的代码替换为符合需要的代码，您可覆盖将 WSDL 类型映射至 Java 类型的已生成组件。如果已定义自己的 Java 类，那么考虑使用您自己的映射。使用此过程来进行更改。

过程

1. 查找生成的组件。 该组件命名为 *java_class*Mapper.component。
2. 使用文本编辑器编辑组件。
3. 注释掉生成的代码并提供自己的方法。

切勿更改包含组件实现的文件名。

这是要替换的已生成组件的示例:

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

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

将组件和其他文件复制到所包含模块驻留的目录，再连接 WebSphere Integration Developer 中的组件或使用 serviceDeploy 命令生成企业归档（EAR）文件。

相关概念

『用于 Java 至服务数据对象转换的运行时规则』

要正确地覆盖生成的代码或确定与 Java 至服务数据对象（SDO）转换相关的可能的运行时异常，了解所涉及的规则十分重要。大部分转换都是直接了当的，但也存在一些复杂的情况，运行时在转换生成的代码时会提供最佳的可能性。

相关参考

Java 至 XML 转换

系统使用预定义的规则生成基于 Java 类型的 XML。

genMapper 命令

使用 genMapper 命令来生成将服务组件体系结构（SCA）引用连接到 Java 接口的组件。

用于 Java 至服务数据对象转换的运行时规则

要正确地覆盖生成的代码或确定与 Java 至服务数据对象（SDO）转换相关的可能的运行时异常，了解所涉及的规则十分重要。大部分转换都是直接了当的，但也存在一些复杂的情况，运行时在转换生成的代码时会提供最佳的可能性。

基本类型和类

运行时执行服务数据对象与基本 Java 类型和类之间的直接转换。基本类型和类包括:

- Char 或 java.lang.Character
- 布尔值
- Java.lang.Boolean
- Byte 或 java.lang.Byte
- Short 或 java.lang.Short
- Int 或 java.lang.Integer
- Long 或 java.lang.Long
- Float 或 java.lang.Float

- Double 或 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[]

用户定义的 Java 类和数组

当从 Java 类或数组转换为 SDO 时，运行时会创建数据对象，它具有通过转换 Java 类型的程序包名生成的 URI 和等同于 Java 类名称的类型。例如，Java 类 com.ibm.xsd.Customer 已转换为 SDO 和 URI http://xsd.ibm.com（具有 Customer 类型）。然后，运行时会检查 Java 类成员的内容，并在 SDO 中将值指派给属性。

当从 SDO 转换至 Java 类型，运行时通过转换 URI 来生成程序包名，并且类型名等同于 SDO 的类型。例如，类型为 Customer 和 URI 为 http://xsd.ibm.com 的数据对象生成 Java 程序包 com.ibm.xsd.Customer 的实例。然后，运行时从 SDO 的属性中抽取值，并且将那些属性指派给 Java 类的实例中的字段。

当 Java 类是用户定义的接口时，您必须覆盖生成的代码并提供运行时可实例化的具体类。如果运行时无法创建具体类，那么将发生异常。

Java.lang.Object

当 Java 类型是 java.lang.Object 时，生成的类型是 xsd:anyType。模块可以使用任何 SDO 来调用此接口。如果运行时可以找到具体类，那么运行时会尝试采用与对用户定义的 Java 类和数组所采用的相同方法来实例化具体类。否则，运行时将 SDO 传递至 Java 接口。

即使该方法返回 java.lang.Object 类型，运行时也会在该方法返回具体类型时转换为 SDO。运行时使用与它类似的转换，将用户定义的 Java 类和数组转换为 SDO，如下一段如述。

当从 Java 类或数组转换为 SDO 时，运行时会创建数据对象，它具有通过转换 Java 类型的程序包名生成的 URI 和等同于 Java 类名称的类型。例如，Java 类 com.ibm.xsd.Customer 已转换为 SDO 和 URI http://xsd.ibm.com（具有 Customer 类型）。然后，运行时会检查 Java 类成员的内容，并在 SDO 中将值指派给属性。

在任一情况下，如果运行时无法完成转换，那么会发生异常。

Java.util 容器类

当转换至具体的 Java 容器类（如向量、散列映射、散列设置和类似事物）时，运行时会实例化正确的容器类。运行时使用类似于对用户定义的 Java 类和数组所用的方法来填充容器类。如果运行时无法找到具体的 Java 类，那么运行时会使用 SDO 来填充容器类。

当将具体的 Java 容器类转换为 SDO 时，运行时使用“Java 至 XML 转换”中所示的已生成模式。

Java.util 接口

对于 java.util 程序包中的某些容器接口，运行时实例化下列具体类:

表 1. WSDL 类型至 Java 类转换

接口	缺省具体类
集合	散列设置
映象	散列映射
列表	数组列表
设置	散列设置

相关任务

第 14 页的『覆盖生成的服务组件体系结构实现』
有时，系统创建的介于 Java 代码与服务数据对象（SDO）之间的转换可能不符合您的需要。使用此过程将缺省服务组件体系结构（SCA）类实现替换为自己的服务组件体系结构（SCA）类实现。

第 15 页的『覆盖服务数据对象至 Java 转换』
有时，系统创建的介于服务数据对象（SDO）和 Java 类型对象之间的转换可能不符合您的需要。使用此过程将缺省实现替换为自己的实现。

相关参考

Java 至 XML 转换

系统使用预定义的规则生成基于 Java 类型的 XML。

genMapper 命令

使用 genMapper 命令来生成将服务组件体系结构（SCA）引用连接到 Java 接口的组件。

第 2 章 为业务流程和任务开发客户机应用程序

您可以使用建模工具来构建和部署业务流程与任务。这些流程和任务在运行时期间进行交互作用。例如，在启动流程时或声明和完成任务时，都会发生流程和任务交互作用。您可以使用业务流程编排器资源管理器与流程和任务进行交互，也可以使用业务流程编排器 API 为这些交互开发定制的客户机。

关于此任务

这些客户机可以是 Enterprise JavaBeans™（EJB）客户机、Web Service 客户机或 Web 客户机，它们利用业务流程编排器资源管理器 JavaServer Faces（JSF）组件。业务流程编排器为您提供用于 Web Service 的 Enterprise JavaBeans（EJB）API 和接口，以便开发这些客户机。包括另一个 EJB 应用程序在内的任何 Java 应用程序都可以访问该 EJB API。可以从 Java 环境或 Microsoft® .Net 环境访问 Web Service 的接口。

为业务流程和人员任务开发 EJB 客户机应用程序

EJB API 提供一组普通方法，用于开发 EJB 客户机应用程序，以便处理安装在 WebSphere Process Server 上的业务流程和人员任务。

关于此任务

使用这些 Enterprise JavaBeans（EJB）API，您可以创建客户机应用程序来执行下列操作:

- 管理从开始执行到完成时将其删除的流程和任务的整个生命周期
- 修复活动和流程
- 管理和分发工作组成员之间的工作负载

EJB API 作为两个无状态会话企业 Bean 提供:

- BusinessFlowManagerService 接口为业务流程应用程序提供方法
- HumanTaskManagerService 接口为基于任务的应用程序提供方法

有关 EJB API 的更多信息，请参阅 com.ibm.bpe.api 包和 com.ibm.task.api 包中的 Javadoc。

下列步骤概述了开发 EJB 客户机应用程序所需执行的操作。

过程

1. 确定应用程序要提供的功能。
2. 确定将要使用的会话 Bean。

   根据应用程序的实现方案，可以使用其中一个会话 Bean，也可以同时使用这两个会话 Bean。
3. 确定应用程序的用户所需的权限。

应用程序的用户必须分配有适当的授权角色，以便调用应用程序中包括的方法和查看这些方法返回的对象和对象属性。当创建适当会话 Bean 的实例时，WebSphere Application Server 将使上下文与该实例相关联。上下文包含有关调用者的主体标识、组成员资格列表和角色的信息。此信息用来检查调用者对每个调用的权限。

Javadoc 包含每个方法的权限信息。

4. 确定应用程序的呈示方式。

   可以采用本地方式或远程方式调用 EJB API。

5. 开发应用程序。
   a. 访问 EJB API。
   b. 使用 EJB API 与流程或任务进行交互。
      - 查询数据。
      - 处理数据。

访问 EJB API

Enterprise JavaBeans（EJB）API 作为两个无状态会话企业 Bean 提供。业务流程应用程序和任务应用程序通过该 Bean 的 home 接口来访问适当的会话企业 Bean。

关于此任务

BusinessFlowManagerService 接口为业务流程应用程序提供方法，而 HumanTaskManagerService 接口为基于任务的应用程序提供方法。该应用程序可以是任何 Java 应用程序，其中包括另一个 Enterprise JavaBeans（EJB）应用程序。

访问会话 Bean 的远程接口

EJB 客户机应用程序通过 Bean 的远程 home 接口来访问会话 Bean 的远程接口。

关于此任务

会话 Bean 可以是 BusinessFlowManager 会话 Bean（用于流程应用程序）或 HumanTaskManager 会话 Bean（用于任务应用程序）。

过程

1. 在应用程序部署描述符中添加对会话 Bean 的远程接口的引用。 请添加对下列其中一个文件的引用:
   - application-client.xml 文件（用于 Java 2 Platform, Enterprise Edition（J2EE）客户机应用程序）
   - web.xml 文件（用于 Web 应用程序）
   - ejb-jar.xml 文件（用于 Enterprise JavaBeans（EJB）应用程序）

   以下示例显示了流程应用程序中对远程 home 接口的引用:

```
<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>
```

   以下示例显示了任务应用程序中对远程 home 接口的引用:

```
<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>
```

如果使用 WebSphere Integration Developer 来在部署描述符中添加 EJB 引用，就会在部署该应用程序时自动为 EJB 引用创建绑定。有关添加 EJB 引用的更多信息，请参阅 WebSphere Integration Developer 文档。

2. 将生成的存根与应用程序打包到一起。

   如果该应用程序与 BPEContainer 应用程序或 TaskContainer 应用程序不在同一 Java 虚拟机（JVM）中运行，那么完成下列操作:

   a. 对于流程应用程序来说，请将 *<install_root>*/ProcessChoreographer/client/bpe137650.jar 文件与应用程序的企业归档（EAR）文件打包到一起。

   b. 对于任务应用程序来说，请将 *<install_root>*/ProcessChoreographer/client/task137650.jar 文件与应用程序的 EAR 文件打包到一起。

   c. 将应用程序模块清单文件中的 **Classpath** 参数设置为包括该 JAR 文件。

      该应用程序模块可以是 J2EE 应用程序、Web 应用程序或 EJB 应用程序。

   d. 如果在业务流程或人员任务中使用复杂数据类型，并且客户机未在 EJB 应用程序或 Web 应用程序中运行，那么将相应 XSD 或 WSDL 文件与应用程序的 EAR 文件打包到一起。

3. 通过 Java 命名和目录接口（JNDI）找到会话 Bean 的远程 home 接口。

   以下示例显示了流程应用程序中的此步骤:

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the BusinessFlowManager bean
Object result =
       initialContext.lookup("java:comp/env/ejb/BusinessFlowManagerHome");

// Convert the lookup result to the proper type
   BusinessFlowManagerHome processHome =
           (BusinessFlowManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,BusinessFlowManagerHome.class);
```

   会话 Bean 的远程 home 接口包含 EJB 对象的 create 方法。该方法返回该会话 Bean 的远程接口。

4. 访问会话 Bean 的远程接口。

   以下示例显示了流程应用程序中的此步骤:

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

   对会话 Bean 的访问权并不保证调用者可以执行该 Bean 提供的所有操作；还必须另外授权该调用者执行这些操作。当创建会话 Bean 的实例时，将使上下文与会话 Bean 的实例相关联。上下文包含调用者的主体标识和组成员资格列表，并指示调用者是否具有其中一个业务流程编排器 J2EE 角色。即使在未设置全局安全性时，该上下文也可用来检查调用者对每个调用的权限。如果未设置全局安全性，那么调用者的主体标识具有值 UNAUTHENTICATED。

5. 调用服务接口公布的企业函数。

   以下示例显示了流程应用程序中的此步骤:

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

   应用程序中执行的调用作为事务运行，而事务是以下列其中一种方式建立和结束的:

   - 由 WebSphere Application Server 自动建立和结束（部署描述符指定了 TX_REQUIRED）。
   - 由应用程序显式地建立和结束。可以将多个应用程序调用捆绑成一个事务:

     ```
     // Obtain user transaction interface
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("jta/usertransaction");

     // Begin a transaction
     transaction.begin();

     // Applications calls ...

     // On successful return, commit the transaction
     transaction.commit();
     ```

   提示: 要防止数据库锁定冲突，请避免并行运行与下列内容相似的语句:

   ```
   // Obtain user transaction interface
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup("jta/usertransaction");

   transaction.begin();

   //read the activity instance
   process.getActivityInstance(aiid);
   //claim the activity instance
   process.claim(aiid);

   transaction.commit();
   ```

   getActivityInstance 方法和其他读操作设置读锁定。在此示例中，对活动实例的读锁定更级为对活动实例的更新锁定。当并行运行这些事务时，这会导致数据库死锁。

示例

以下是在任务应用程序中执行步骤 3 到 5 的示例。

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the HumanTaskManager bean
Object result =
         initialContext.lookup("java:comp/env/ejb/HumanTaskManagerHome");

// Convert the lookup result to the proper type
HumanTaskManagerHome taskHome =
           (HumanTaskManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,HumanTaskManagerHome.class);

...
//Access the remote interface of the session bean.
HumanTaskManager task = taskHome.create();
```

```
...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

访问会话 Bean 的本地接口

EJB 客户机应用程序通过会话 Bean 的本地 home 接口来访问该 Bean 的本地接口。

关于此任务

会话 Bean 可以是 BusinessFlowManager 会话 Bean（用于流程应用程序）或 HumanTaskManager 会话 Bean（用于人员任务应用程序）。

过程

1. 在应用程序部署描述符中添加对会话 Bean 的本地接口的引用。 请添加对下列其中一个文件的引用:
   - application-client.xml 文件（用于 Java 2 Platform, Enterprise Edition（J2EE）客户机应用程序）
   - web.xml 文件（用于 Web 应用程序）
   - ejb-jar.xml 文件（用于 Enterprise JavaBeans（EJB）应用程序）

   以下示例显示了流程应用程序中对本地 home 接口的引用:

   ```
   <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>
   ```

   以下示例显示了任务应用程序中对本地 home 接口的引用:

   ```
   <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>
   ```

   如果使用 WebSphere Integration Developer 来在部署描述符中添加 EJB 引用，就会在部署该应用程序时自动为 EJB 引用创建绑定。有关添加 EJB 引用的更多信息，请参阅 WebSphere Integration Developer 文档。

2. 通过 Java 命名和目录接口（JNDI）找到会话 Bean 的本地 home 接口。

   以下示例显示了流程应用程序中的此步骤:

   ```
   // Obtain the default initial JNDI context
   InitialContext initialContext = new InitialContext();

   // Lookup the local home interface of the BusinessFlowManager bean

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

   会话 Bean 的本地 home 接口包含 EJB 对象的 create 方法。该方法返回该会话 Bean 的本地接口。

3. 访问会话 Bean 的本地接口。

以下示例显示了流程应用程序中的此步骤:

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

对会话 Bean 的访问权并不保证调用者可以执行该 Bean 提供的所有操作；还必须另外授权该调用者执行这些操作。当创建会话 Bean 的实例时，将使上下文与会话 Bean 的实例相关联。上下文包含调用者的主体标识和组成员资格列表，并指示调用者是否具有其中一个业务流程编排器 J2EE 角色。即使在未设置全局安全性时，该上下文也可用来检查调用者对每个调用的权限。如果未设置全局安全性，那么调用者的主体标识具有值 UNAUTHENTICATED。

4. 调用服务接口公布的企业函数。

以下示例显示了流程应用程序中的此步骤:

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

应用程序中执行的调用作为事务运行，而事务是以下列其中一种方式建立和结束的:

- 由 WebSphere Application Server 自动建立和结束（部署描述符指定了 TX_REQUIRED）。
- 由应用程序显式地建立和结束。可以将多个应用程序调用捆绑成一个事务:

```
// Obtain user transaction interface
UserTransaction transaction=
       (UserTransaction)initialContext.lookup("jta/usertransaction");

// Begin a transaction
transaction.begin();

// Applications calls ...

// On successful return, commit the transaction
transaction.commit();
```

提示: 要防止数据库死锁，请避免并行运行与下列内容相似的语句:

```
// Obtain user transaction interface
UserTransaction transaction=
       (UserTransaction)initialContext.lookup("jta/usertransaction");

transaction.begin();

//read the activity instance
process.getActivityInstance(aiid);
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

getActivityInstance 方法和其他读操作设置读锁定。在此示例中，对活动实例的读锁定更级为对活动实例的更新锁定。当并行运行这些事务时，这会导致数据库死锁。

示例

以下是在任务应用程序中执行步骤 2 到 4 的示例。

```
//Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the local home interface of the HumanTaskManager bean
LocalHumanTaskManagerHome taskHome =
       (LocalHumanTaskManagerHome)initialContext.lookup
```

```
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Access the local interface of the session bean
LocalHumanTaskManager task = taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

查询业务流程以及与任务相关的对象

客户机应用程序处理业务流程和与任务相关的对象。可以在数据库中查询业务流程以及与任务相关的对象，以检索这些对象的特定属性。

关于此任务

在配置业务流程编排器期间，将使一个关系数据库与业务流程容器和任务容器相关联。此数据库存储所有用于管理业务流程和任务的模板（模型）和实例（运行时）数据。您使用类似于 SQL 的语法来查询此数据。

可以执行一次性的查询来检索某个对象的特定属性。也可以将常用的查询保存下来并将这些存储查询包括在应用程序中。

对业务流程以及与任务相关的对象执行的查询

使用服务 API 的 query 方法或 queryAll 方法来检索已存储的有关业务流程和任务的信息。

query 方法可以由所有用户调用，并且它会返回存在工作项的对象的属性。queryAll 方法只能由具有下列其中一个 J2EE 角色的用户调用: BPESystemAdministrator、TaskSystemAdministrator、BPESystemMonitor 或 TaskSystemMonitor。此方法会返回存储在数据库中的所有对象的属性。

所有 API 查询均映射到 SQL 查询。所生成的 SQL 查询的格式取决于下列方面:

- 查询是否由具有其中一个 J2EE 角色的人员调用。
- 已查询的对象。您可以使用预定义的数据库视图来查询对象属性。
- 插入 from 子句、连接条件和特定于用户的条件以获得访问控制。

您可以在查询中包括定制属性和变量属性。如果在查询中包括多个定制属性或变量属性，那么这将导致对相应数据库表进行自连接。视数据库系统而定，这些 query() 调用可能具有执行含义。

您还可以使用 createStoredQuery 方法来将查询存储在业务流程编排器数据库中。当定义存储查询时，可提供该查询条件。当存储查询运行时，将动态地应用该条件，也就是说，在运行时期间汇编数据。如果存储查询包含一些参数，那么在查询运行时也将解析这些参数。

有关业务流程编排器 API 的更多信息，请参阅 com.ibm.bpe.api 包（对于与流程相关的方法）和 com.ibm.task.api 包（对于与任务相关的方法）中的 Javadoc。

API query 方法的语法:

业务流程编排器 API 查询的语法与 SQL 查询的语法相似。查询可以包含 select 子句、where 子句、order-by 子句、skip-tuples 参数、阈值参数和时区参数。

查询的语法取决于对象类型。下表显示每个不同对象类型的语法。

表 2.

对象	语法
流程模板	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
任务模板	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
业务流程和与任务相关的数据	`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);`

Select 子句:

查询函数中的 Select 子句标识查询所要返回的对象属性。

Select 子句描述查询结果。它指定了一组名称，这些名称标识了要返回的对象属性（结果列）。Select 子句的语法与 SQL SELECT 子句的语法相似；它使用逗号来分隔子句的各个部分。此子句的每个部分都必须指定其中一个预定义视图中的列。这些列必须完全由视图名称和列名指定。QueryResultSet 对象中返回的列的顺序与 Select 子句中指定列的顺序相同。

Select 子句不支持 SQL 聚集函数，例如 AVG()、SUM()、MIN() 或 MAX()。

要选择多个“名称/值”对的属性（例如，可以查询的定制属性和变量属性），请对视图名添加一位数的计数器。此计数器可以采用 1 至 9 之间的值。

Select 子句示例

- ″WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON″

  获取相关对象的对象类型以及工作项的分配原因。
- ″DISTINCT WORK_ITEM.OBJECT_ID″

  获取调用者具有其工作项的相应对象的所有标识（不带重复项）。
- ″ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON″

  获取调用者具有其工作项的相应活动的名称及其分配原因。
- ″ACTIVITY.STATE, PROCESS_INSTANCE.STARTER″

  获取活动的状态及其相关流程实例的启动者。
- ″DISTINCT TASK.TKIID, TASK.NAME″

获取调用者具有其工作项的相应任务的所有标识和名称（不带重复项）。

- ″TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE″

  获取 Where 子句中进一步指定的定制属性的值。

- ″QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE

  获取可查询的变量的属性值。在 Where 子句中进一步指定这些部分。

- ″COUNT(DISTINCT TASK.TKIID)″

  计算满足 Where 子句的唯一任务的工作项数。

Where 子句:

查询函数中的 Where 子句描述要对查询范围应用的过滤器条件。

where 子句的语法与 SQL WHERE 子句的语法相似。您不需要显式地对 API Where 子句添加 SQL FROM 子句或 JOIN 谓词，这些构造在查询运行时会自动添加。如果您不想应用过滤器条件，那么必须将 Where 子句指定为 `null`。

Where 子句的语法支持:

- 关键字: AND, OR, NOT
- 比较运算符: =、<=、<、<>、>、>= 和 LIKE

  LIKE 操作支持为所查询的数据库定义的通配符。

- 集合操作: IN

下列规则也适用:

- 将对象标识常量指定为 `ID('string-rep-of-oid')`。
- 将二进制常量指定为 `BIN('UTF-8 string')`。
- 使用符号常量来代替整数枚举。例如，指定 `ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY` 来代替活动状态表达式 `ACTIVITY.STATE=2`。
- 如果比较语句中的属性值包含单引号（'），请成倍地指定单引号，例如 `"TASK_CPROP.STRING_VALUE='d''automatisation'"`。
- 要引用多个“名称/值”对的属性（例如定制属性），请对视图名添加由 1 个数字组成的后缀。例如: `"TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'"`
- 将时间戳记常量指定为 `TS('yyyy-mm-ddThh:mm:ss')`。要引用当前日期，请指定 `CURRENT_DATE` 作为时间戳记。

  在时间戳记中必须至少指定日期值或时间值:

  – 如果仅指定日期，那么时间值将设置为零。
  – 如果仅指定时间，那么日期将设置为当前日期。
  – 如果指定了日期，那么年份必须包含 4 位数；月份值和天值是可选的。缺少的月份值和天值将设置为 01。例如，`TS('2003')` 与 `TS('2003-01-01T00:00:00')` 相同。
  – 如果指定了时间，那么这些值是使用 24 小时制表达的。例如，如果当前日期是 2003 年 1 月 1 日，那么 `TS('T16:04')` 或 `TS('16:04')` 与 `TS('2003-01-01T16:04:00')` 相同。

Where 子句示例

- 将对象标识与现有标识作比较

  ```
  "WORK_ITEM.WIID = ID('_WI:800c00ed.df8d7e7c.fefff80.38')"
  ```

  此类 Where 子句通常是使用上一个调用返回的现有对象标识动态创建的。如果此对象标识存储在 *wiid1* 变量中，那么可以将此子句构造成:

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

- 使用时间戳记

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

- 使用符号常量

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

- 使用布尔值 true 和 false

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

- 使用定制属性

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

Order-by 子句:

查询函数中的 Order-By 子句指定查询结果集的排序条件。

您可以在视图中指定列的列表（结果按视图排序）。这些列必须由视图和列的名称完全限定。它是用来指定处于 select 子句中的列的最佳做法。

order-by 子句的语法与 SQL order-by 子句的语法相似；使用逗号来分隔子句的每个部分。您还可以指定 ASC 来按升序对列进行排序，以及指定 DESC 来按降序对列进行排序。如果您不想对查询结果集进行排序，那么必须将 Order-by 子句指定为 null。

将在服务器上应用排序条件，即，使用服务器的语言环境来进行排序。如果您指定多个列，那么查询结果集将首先按第一个列的值进行排序，然后按第二个列的值进行排序，依此类推。您无法像使用 SQL 查询那样在 order-by 子句中按位置来指定列。

Order-by 子句示例

- ″PROCESS_TEMPLATE.NAME″

  按流程模板名的字母顺序对查询结果进行排序。

- ″PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC″

  按创建日期对查询结果进行排序，并且，对于特定日期，按流程实例名的字母顺序逆序对结果进行排序。

- ″ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE″

  依次按活动所有者、活动模板名和活动状态对查询结果进行排序。

Skiptuples 参数:

Skip-tuples 参数指定从查询结果集的起始位置算起的这样一些查询结果集元组的数目:将要被忽略且不返回至查询结果集中的调用者。

将此参数与 Threshold 参数配合使用，可以在客户机应用程序中实现分页，例如，检索前 20 个项，然后再检索随后的 20 个项，依此类推。

如果将此参数设置为 null，并且未设置 Threshold 参数，那么将返回所有符合条件的元组。

Skiptuples 参数示例

- new Integer(5)

  指定不返回前 5 个符合条件的元组。

Threshold 参数:

查询函数中的 Threshold 参数对查询结果集中从服务器返回到客户机的对象数进行限制。

因为生产方案中的查询结果集可以包含数千个甚至数百万个项，所以始终指定阈值是最佳做法。Threshold 参数非常有用，例如，在一次只应该显示少量项目的图形用户界面中，就是这种情况。如果适当地设置 Threshold 参数，就可以提高数据库查询的运行速度，并且可以减少需要从服务器传递到客户机的数据量。

如果将此参数设置为 null，并且未设置 Skiptuples 参数，那么将返回所有符合条件的对象。

Threshold 参数示例

- new Integer(50)

  指定返回 50 个符合条件的元组。

Timezone 参数:

查询函数中的 Timezone 参数定义该查询中的时间戳记常量的时区。

在启动查询的客户机与处理该查询的服务器之间，时区可能会不同。请使用 Timezone 参数来指定 Where 子句中使用的时间戳记常量的时区，例如，使用该参数来指定本地时间。查询结果集中返回的日期将采用该查询中指定的时区。

如果此参数设置为 null，那么将假定时间戳记常量是全球标准时间（UTC）。

Timezone 参数示例

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

  返回 2005 年 1 月 1 日本地时间 17:40 后启动的活动的对象标识。

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

  返回 2005 年 1 月 1 日 UTC 时间 17:40 后启动的活动的对象标识。例如，此时间比东部标准时间早 6 小时。

存储查询中的参数:

存储查询是指存储在数据库中并由名称标识的查询。当运行该查询时，将动态地汇编标准元组。为了使存储查询可重复使用，您可以使用在运行时期间解析的查询定义中的参数。

例如，您已定义定制属性来存储客户名称。可以定义查询来返回与特定客户 ACME Co. 相关联的任务。为了查询此信息，查询中的 where 子句可能与下列示例相似:

```
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.'";
```

为了使此查询可重复使用，以便还可以搜索客户 BCME Ltd，您可以对定制属性的值使用这些参数。如果将参数添加至任务查询，那么该查询可能与下列示例相似:

```
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'";
```

在运行时期间，将解析已传递至 query 方法的参数列表中的 `@param1` 参数。下列规则适用于查询中的参数用法:

- 只能在 where 子句中使用参数。
- 参数是字符串。
- 在运行时期间使用字符串来替换参数。如果需要特殊字符，那么必须在 where 子句中指定这些字符，或在运行时期间将其作为参数的一部分传入。
- 参数名称由与整数并置的字符串 `@param` 组成。最小数字是 1，它指向在运行时期间已传递至查询 API 的参数列表中的第一个项。
- 参数可以在 where 子句中多次使用，所有出现的该参数全部由同一个值替换。

相关任务

第 52 页的『管理存储查询』
存储查询提供了保存经常运行的查询的方法。存储查询可以是可供所有用户使用的查询（公用查询），也可以是属于特定用户的查询（专用查询）。

查询结果:

查询结果集包含查询结果。

结果集的元素是满足调用者提供的 Where 子句且调用者有权查看的对象的属性。您可以使用 API next 方法来以相对方式读取元素，也可以使用 first 和 last 方法来以绝对方式读取元素。由于查询结果集的隐式游标最初定位在第一个元素之前，所以在读取元素前，必须调用 first 或 next 方法。可以使用 size 方法来确定集合中的元素数目。

查询结果集的元素由工作项的所选属性及其相关被引用对象（例如活动实例和流程实例）组成。QueryResultSet 元素的第一个属性（列）指定查询请求中 Select 子句指定的第一个属性的值。QueryResultSet 元素的第二个属性（列）指定查询请求中 Select 子句指定的第二个属性的值，依此类推。

可以通过调用与属性类型兼容的方法以及通过指定适当的列索引来检索属性值。列索引号从 1 开始。

属性类型	方法
字符串	getString
OID	getOID
时间戳记	getTimestamp getString getTimestampAsLong
整型	getInteger getShort getLong getString getBoolean
布尔值	getBoolean getShort getInteger getLong getString
byte[]	getBinary

示例:

运行以下查询:

```
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);
```

返回的查询结果集包含 4 列:

- 第 1 列是时间戳记
- 第 2 列是字符串
- 第 3 列是对象标识
- 第 4 列是整数

可以使用下列方法来检索属性值:

```
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);
}
```

可以使用结果集的显示名，例如，在打印的表中，将显示名用作标题。这些名称是视图的列名，或者是查询中的 AS 子句定义的名称。可以使用以下方法来检索本示例中的显示名:

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

特定于用户的访问条件:

当从 API 查询生成 SQL SELECT 语句时，将添加特定于用户的访问条件。这些条件保证只有那些满足调用者指定条件且调用者对其拥有权限的对象才会返回给调用者。

被添加的访问条件取决于用户是否是系统管理员。

非系统管理员的用户调用的查询

生成的 SQL WHERE 子句将 API where 子句与特定于用户的访问控制条件相结合。查询仅检索用户有权访问的对象，即用户对其拥有工作项的那些对象。工作项表示将用户或用户组分配给业务对象的授权角色，例如任务或流程。例如，如果用户 John Smith 是给定任务的潜在所有者成员，那么就会存在表示此关系的工作项对象。

例如，如果非系统管理员的用户查询任务，那么在未启用组工作项的情况下，下列访问条件会被添加到 WHERE 子句:

```
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 )
```

因此，如果 John Smith 想要获取他作为潜在所有者的任务的列表，那么 API where 子句可能会如下所示:

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

此 API where 子句导致 SQL 语句中的下列访问条件:

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

这也意味着: 如果 John Smith 想要查看他作为流程读者或流程管理员以及对其没有工作项的活动和任务，那么必须将来自 PROCESS_INSTANCE 视图的属性添加到查询的 select、where 或 order-by 子句，例如 PROCESS_INSTANCE.PIID。

如果组工作项已启用，那么会将其他访问条件添加到 WHERE 子句，从而允许用户访问该组对其拥有访问权的对象。

系统管理员调用的查询

系统管理员可以调用 query 方法来检查具有相关联的工作项的对象。在此情况下，与 WORK_ITEM 视图的连接将会被添加到已生成的 SQL 查询，但 WORK_ITEM.OWNER_ID 没有访问控制条件。

在此情况下，任务的 SQL 查询包含下列内容:

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

queryAll queries

此类查询只能由系统管理员或系统监视员调用。将不会添加访问控制条件，也不会添加与 WORK_ITEM 视图的连接。此类查询将返回所有对象的全部数据。

query 和 queryAll 方法的示例:

这些示例说明处理查询时生成的各种典型的 API 查询和关联的 SQL 语句的语法。

示例: 查询处于就绪状态的任务:

此示例说明如何使用 query 方法来检索已登录用户可处理的任务。

John Smith 想要获取已分配给他的任务的列表。为了使用户能够处理任务，该任务必须处于就绪状态。已登录用户还必须拥有该任务的潜在所有者工作项。下列代码段说明此查询的 query 方法调用:

```
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 )
```

当生成 SQL SELECT 语句时，将执行下列操作:

- 将访问控制的条件添加到 where 子句。此示例假定未启用组工作项。
- 常量（如 TASK.STATE.STATE_READY）被其数值替换。
- 添加 FROM 子句和连接条件。

下列代码段说明已从 API 查询生成的 SQL 语句:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND    WI.REASON = 1
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

要限制 API 查询以用于特定流程的任务（例如 sampleProcess），查询将如下所示:

```
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 )
```

示例: 查询处于已声明状态的任务:

此示例说明如何使用 query 方法来检索已登录用户声明的任务。

用户 John Smith 想要搜索他已声明的任务以及仍处于已声明状态的任务。指定“claimed by John Smith”的条件是 TASK.OWNER = 'JohnSmith'。下列代码段说明了该查询的 query 方法调用:

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

下列代码段说明已从 API 查询生成的 SQL 语句:

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

```
   AND    TA.STATE = 8
   TA.OWNER = 'JohnSmith'
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

当声明任务时，将会为任务所有者创建工作项。因此，形成 John Smith 的已声明任务的查询的另一方法是将下列条件添加到查询，而不是使用 TASK.OWNER = 'JohnSmith':

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

然后，该查询与下列代码段相似:

```
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 )
```

当生成 SQL SELECT 语句时，将执行下列操作:

- 将访问控制的条件添加到 where 子句。此示例假定未启用组工作项。
- 常量（如 TASK.STATE.STATE_READY）被其数值替换。
- 添加 FROM 子句和连接条件。

下列代码段说明已从 API 查询生成的 SQL 语句:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   AND    WI.REASON = 4
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

John 准备要去度假，因此他的团队领导 Anne Grant 想要检查他的当前工作负载。Anne 拥有系统管理员权限。她调用的查询与 John 调用的查询相同。然而，由于 Anne 是管理员，所以生成的 SQL 语句不同。下列代码段说明已生成的 SQL 语句:

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

由于 Anne 是管理员，所以访问控制条件未添加到 WHERE 子句。

示例: 查询升级:

此示例说明如何使用 query 方法来为已登录用户检索升级。

当升级任务时，将创建升级接收者工作项。用户 Mary Jones 想要查看已升级给她的任务列表。下列代码段说明了该查询的 query 方法调用:

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

当生成 SQL SELECT 语句时，将执行下列操作:

- 将访问控制的条件添加到 where 子句。此示例假定未启用组工作项。
- 常量（如 TASK.STATE.STATE_READY）被其数值替换。
- 添加 FROM 子句和连接条件。

下列代码段说明已从 API 查询生成的 SQL 语句:

```
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 )
```

示例: 使用 queryAll 方法:

此示例说明如何使用 queryAll 方法来检查属于流程模板的所有活动。

queryAll 方法仅可供具有系统管理员或系统监视员权限的用户使用。下列代码段说明了该查询的 queryAll 方法调用, 它检索属于流程模板 sampleProcess 的所有活动:

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

下列代码段说明已从 API 查询生成的 SQL 查询:

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

由于该调用由管理员调用, 所以访问控制条件未添加到生成的 SQL 语句中。与 WORK_ITEM 视图的连接也未添加。这意味着查询检索流程模板的所有活动, 包括没有工作项的那些活动。

示例: 在查询中包含查询属性:

此示例说明如何使用 query 方法来检索属于业务流程的任务。该流程已为其定义了您想要将它包含在搜索中的查询属性。

例如, 您想要搜索属于业务流程的所有处于就绪状态的人员任务。该流程具有值 CID_12345 和名称空间的查询属性 **customerID**。下列代码段说明了该查询的 query 方法调用:

```
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 );
```

如果您现在想要将第二个查询属性添加到查询（例如, 具有给定名称空间的 **Priority**）, 那么该查询的 query 方法调用如下所示:

```
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 );
```

如果将多个查询属性添加到该查询，那么必须为所添加的每个属性编号，如代码段中所示。然而，查询定制属性会影响性能；性能将随着查询中的定制属性数量增加而下降。

示例：在查询中包含定制属性:

此示例说明如何使用 query 方法来检索具有定制属性的任务。

例如，您想要搜索处于就绪状态的具有值为 CID_12345 的定制属性 **customerID** 的所有人员任务。下列代码段说明了该查询的 query 方法调用:

```
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 );
```

如果您现在想要检索任务及其定制属性，那么该查询的 query 方法调用如下所示:

```
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 );
```

下列代码段显示从此 API 查询生成的 SQL 语句:

```
SELECT DISTINCT TA.TKIID , TACP.NAME , TACP.STRING_VALUE
   FROM  TASK TA LEFT JOIN TASK_CPROP TACP ON (TA.TKIID = TACP.TKIID),
         WORK_ITEM WI
   WHERE WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND  (WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID IS NULL AND WI.EVERYBODY = 1 )
```

此 SQL 语句包含 TASK 视图与 TASK_CPROP 视图之间的外连接。这意味着: 即使满足 WHERE 子句的任务没有任何定制属性，也仍会被检索。

对业务流程对象和人员任务对象执行的查询的预定义视图:

为业务流程对象和人员任务对象提供了预定义的数据库视图。在查询这些对象的引用数据时，可使用这些视图。

使用这些预定义视图时，不需要为视图列显式地添加 JOIN 谓词，这些构造是自动添加的。您可以使用服务 API 的通用查询函数（BusinessFlowManagerService 或 HumanTaskManagerService）来查询此数据。也可以使用 HumanTaskManagerDelegate API 的相应方法或者由您的 ExecutableQuery 接口实现提供的预定义查询。

注: 视图可能包含未描述的列。这些列仅供内部使用。

ACTIVITY 视图:

这个预定义数据库视图用于查询活动。

表 3. ACTIVITY 视图中的列

列名	类型	注释
PIID	标识	流程实例标识。
AIID	标识	活动实例标识。
PTID	标识	流程模板标识。
ATID	标识	活动模板标识。
KIND	整型	活动的类型。可能的值有: 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	时间戳记	活动的完成时间。
ACTIVATED	时间戳记	活动的激活时间。
FIRST_ACTIVATED	时间戳记	活动的首次激活时间。
STARTED	时间戳记	活动的启动时间。
STATE	整型	活动的状态。可能的值有: 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)

表 3. ACTIVITY 视图中的列 (续)

列名	类型	注释
OWNER	字符串	所有者的主体标识。
DESCRIPTION	字符串	如果活动模板描述包含占位符，那么此列包含解析占位符后的活动实例描述。
TEMPLATE_NAME	字符串	相关活动模板的名称。
TEMPLATE_DESCR	字符串	相关活动模板的描述。
BUSINESS_RELEVANCE	布尔值	指定活动是否与业务相关。可能的值有: **TRUE** 该活动与业务相关。您可以在业务流程编排器资源管理器中查看活动状态。 **FALSE** 该活动与业务无关。
EXPIRES	时间戳记	活动到期日期和时间。如果该活动已到期，那么它是发生此事件时的日期和时间。

ACTIVITY_ATTRIBUTE 视图:

这个预定义数据库视图用于查询活动的定制属性。

表 4. ACTIVITY_ATTRIBUTE 视图中的列

列名	类型	注释
AIID	标识	定制属性所属活动实例的标识。
NAME	字符串	定制属性的名称。
VALUE	字符串	定制属性的值。

ACTIVITY_SERVICE 视图:

这个预定义数据库视图用于查询活动服务。

表 5. ACTIVITY_SERVICE 视图中的列

列名	类型	注释
EIID	标识	事件实例的标识。
AIID	标识	正在等待事件的活动实例的标识。
PIID	标识	事件所属流程实例的标识。
VTID	标识	描述了事件的服务模板的标识。
PORT_TYPE	字符串	端口类型的名称。
NAME_SPACE_URI	字符串	名称空间的 URI。
OPERATION	字符串	服务的操作名。

APPLICATION_COMP 视图:

这个预定义数据库视图用于查询任务的应用程序组件标识和缺省设置。

表 6. APPLICATION_COMP 视图中的列

列名	类型	注释
ACOID	字符串	应用程序组件的标识。
BUSINESS_ RELEVANCE	布尔值	与任务业务相关的组件缺省策略。任务模板或任务中的定义可以覆盖此值。此属性影响对审计跟踪进行的日志记录。可能的值有: **TRUE** 任务与业务相关并且进行审计。 **FALSE** 任务与业务无关并且不进行审计。
NAME	字符串	应用程序组件的名称。
SUPPORT_ AUTOCLAIM	布尔值	组件的缺省自动声明策略。如果此属性设置为 TRUE，那么当单个用户是潜在所有者时，可以自动声明该任务。任务模板或任务中的定义可以覆盖此值。
SUPPORT_CLAIM_ SUSP	布尔值	用于确定是否可以声明已暂挂任务的组件缺省设置。如果此属性设置为 TRUE，那么可以声明已暂挂的任务。任务模板或任务中的定义可以覆盖此值。
SUPPORT_ DELEGATION	布尔值	组件的缺省任务委托策略。如果此属性设置为 TRUE，那么可以修改对任务指定的工作项。这表示可以创建、删除或传递工作项。
SUPPORT_ FOLLOW_ON	布尔值	组件的缺省后续任务策略。如果此属性设置为 TRUE，那么可以为任务创建后续任务。任务模板或任务中的定义可以覆盖此值。
SUPPORT_ SUB_TASK	布尔值	组件的缺省子任务策略。如果此属性设置为 TRUE，那么可以为任务创建子任务。任务模板或任务中的定义可以覆盖此值。

ESCALATION 视图:

这个预定义数据库视图用于查询升级数据。

表 7. ESCALATION 视图中的列

列名	类型	注释
ESIID	字符串	升级实例的标识。
ACTION	整型	升级触发的操作。可能的值有: **ACTION_CREATE_WORK_ITEM (1)** 为每个升级接收者创建工作项。 **ACTION_SEND_EMAIL (2)** 向每个升级接收者发送电子邮件。 **ACTION_CREATE_EVENT (3)** 创建并发布事件。

表 7. ESCALATION 视图中的列 (续)

列名	类型	注释
ACTIVATION_STATE	整型	如果相应任务进入下列其中一种状态，那么创建升级实例: **ACTIVATION_STATE_READY (2)** 指定人员任务或参与任务已经为声明作好准备。 **ACTIVATION_STATE_RUNNING (3)** 指定发端任务已启动并且正在运行。 **ACTIVATION_STATE_CLAIMED (8)** 指定已声明了任务。 **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** 指定该任务正在等待子任务完成。
ACTIVATION_TIME	时间戳记	升级的激活时间。
AT_LEAST_ EXP_STATE	整型	升级所需的任务状态。如果发生超时，那么会将任务状态与此属性的值进行比较。可能的值有: **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** 指定已声明了任务。 **AT_LEAST_EXPECTED_STATE_ENDED (20)** 指定任务处于最终状态（已完成、已失败、已终止或已到期）。 **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** 指定该任务的所有子任务都已完成。
ESTID	字符串	相应升级模板的标识。
FIRST_ESIID	字符串	链中第一个升级的标识。
INCREASE_PRIORITY	整型	指示如何提高任务优先级。可能的值有: **INCREASE_PRIORITY_NO (1)** 不提高任务优先级。 **INCREASE_PRIORITY_ONCE (2)** 每次将任务优先级提高一级。 **INCREASE_PRIORITY_REPEATED (3)** 每当升级重复时，将任务优先级提高一级。
NAME	字符串	升级的名称。
STATE	整型	升级的状态。可能的值有: STATE_INACTIVE (1) STATE_WAITING (2) STATE_ESCALATED (3) STATE_SUPERFLUOUS (4)
TKIID	字符串	升级所属任务实例的标识。

ESCALATION_CPROP 视图:

这个预定义数据库视图用于查询升级的定制属性。

表 8. ESCALATION_CPROP 视图中的列

列名	类型	注释
ESIID	字符串	升级标识。
NAME	字符串	属性的名称。
DATA_TYPE	字符串	非字符串定制属性的类的类型。
STRING_VALUE	字符串	字符串类型的定制属性值。

ESCALATION_DESC 视图:

这个预定义数据库视图用于查询升级的多语言描述性数据。

表 9. ESCALATION_DESC 视图中的列

列名	类型	注释
ESIID	字符串	升级标识。
LOCALE	字符串	与描述或显示名相关联的语言环境名。
DESCRIPTION	字符串	任务模板的描述。
DISPLAY_NAME	字符串	升级的描述性名称。

ESC_TEMPL 视图:

这个预定义数据库视图用于查询升级模板的数据。

表 10. ESC_TEMPL 视图中的列

列名	类型	注释
ESTID	字符串	升级模板的标识。
ACTION	整型	升级触发的操作。可能的值有: **ACTION_CREATE_WORK_ITEM (1)** 为每个升级接收者创建工作项。 **ACTION_SEND_EMAIL (2)** 向每个升级接收者发送电子邮件。 **ACTION_CREATE_EVENT (3)** 创建并发布事件。
ACTIVATION_STATE	整型	如果相应任务进入下列其中一种状态，那么创建升级实例: **ACTIVATION_STATE_READY (2)** 指定人员任务或参与任务已经为声明作好准备。 **ACTIVATION_STATE_RUNNING (3)** 指定发端任务已启动并且正在运行。 **ACTIVATION_STATE_CLAIMED (8)** 指定已声明了任务。 **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** 指定该任务正在等待子任务完成。

表 10. ESC_TEMPL 视图中的列 (续)

列名	类型	注释
AT_LEAST_EXP_STATE	整型	升级所需的任务状态。如果发生超时，那么会将任务状态与此属性的值进行比较。可能的值有: **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** 指定已声明了任务。 **AT_LEAST_EXPECTED_STATE_ENDED (20)** 指定任务处于最终状态（已完成、已失败、已终止或已到期）。 **AT_LEAST_EXPECTED_STATE_SUBTASKS_COMPLETED (21)** 指定该任务的所有子任务都已完成。
CONTAINMENT_CTX_ID	字符串	如果升级模板属于内联任务模板，那么包含上下文是流程模板。如果升级模板上下文属于独立任务模板，那么包含上下文是任务模板。
FIRST_ESTID	字符串	升级模板链中第一个升级模板的标识。
INCREASE_PRIORITY	整型	指示如何提高任务优先级。可能的值有: **INCREASE_PRIORITY_NO (1)** 不提高任务优先级。 **INCREASE_PRIORITY_ONCE (2)** 每次将任务优先级提高一级。 **INCREASE_PRIORITY_REPEATED (3)** 每当升级重复时，将任务优先级提高一级。
NAME	字符串	升级模板的名称。
PREVIOUS_ESTID	字符串	升级模板链中上一个升级模板的标识。
TKTID	字符串	升级模板所属的任务模板标识。

ESC_TEMPL_CPROP 视图:

这个预定义数据库视图用于查询升级模板的定制属性。

表 11. ESC_TEMPL_CPROP 视图中的列

列名	类型	注释
ESTID	字符串	升级模板的标识。
NAME	字符串	属性的名称。
TKTID	字符串	升级模板所属的任务模板标识。
DATA_TYPE	字符串	非字符串定制属性的类的类型。
VALUE	字符串	字符串类型的定制属性值。

ESC_TEMPL_DESC 视图:

这个预定义数据库视图用于查询升级模板的多语言描述性数据。

表 12. ESC_TEMPL_DESC 视图中的列

列名	类型	注释
ESTID	字符串	升级模板的标识。
LOCALE	字符串	与描述或显示名相关联的语言环境名。
TKTID	字符串	升级模板所属的任务模板标识。
DESCRIPTION	字符串	任务模板的描述。
DISPLAY_NAME	字符串	升级的描述性名称。

PROCESS_ATTRIBUTE 视图:

这个预定义数据库视图用于查询流程的定制属性。

表 13. PROCESS_ATTRIBUTE 视图中的列

列名	类型	注释
PIID	标识	定制属性所属流程实例的标识。
NAME	字符串	定制属性的名称。
VALUE	字符串	定制属性的值。

PROCESS_INSTANCE 视图:

这个预定义数据库视图用于查询流程实例。

表 14. PROCESS_INSTANCE 视图中的列

列名	类型	注释
PTID	标识	流程模板标识。
PIID	标识	流程实例标识。
NAME	字符串	流程实例的名称。
STATE	整型	流程实例的状态。可能的值有: 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	时间戳记	流程实例的创建时间。
STARTED	时间戳记	流程实例的启动时间。
COMPLETED	时间戳记	流程实例的完成时间。
PARENT_PIID	标识	父流程实例的标识。
PARENT_NAME	字符串	父流程实例的名称。

表 14. PROCESS_INSTANCE 视图中的列 (续)

列名	类型	注释
TOP_LEVEL_PIID	标识	顶级流程实例的流程实例标识。如果没有顶级流程实例，那么这将是当前流程实例的流程实例标识。
TOP_LEVEL_NAME	字符串	顶级流程实例的名称。如果没有顶级流程实例，这将是当前流程实例的名称。
STARTER	字符串	流程实例的启动者的主体标识。
DESCRIPTION	字符串	如果流程模板描述包含占位符，那么此列包含解析占位符后的流程实例描述。
TEMPLATE_NAME	字符串	相关流程模板的名称。
TEMPLATE_DESCR	字符串	相关流程模板的描述。
RESUMES	时间戳记	自动继续执行流程实例的时间。

PROCESS_TEMPLATE 视图:

这个预定义数据库视图用于查询流程模板。

表 15. PROCESS_TEMPLATE 视图中的列

列名	类型	注释
PTID	标识	流程模板标识。
NAME	字符串	流程模板的名称。
VALID_FROM	时间戳记	流程模板能够被实例化的开始时间。
TARGET_NAMESPACE	字符串	流程模板的目标名称空间。
APPLICATION_NAME	字符串	流程模板所属企业应用程序的名称。
VERSION	字符串	用户定义的版本。
CREATED	时间戳记	在数据库中创建流程模板的时间。
STATE	整型	指定流程模板是否可用于创建流程实例。可能的值有: STATE_STARTED (1) STATE_STOPPED (2)
EXECUTION_MODE	整型	指定流程模板所派生的流程实例的运行方式。可能的值有: EXECUTION_MODE_MICROFLOW (1) EXECUTION_MODE_LONG_RUNNING (2)
DESCRIPTION	字符串	流程模板的描述。
COMP_SPHERE	整型	指定流程模板中微流动实例的补偿行为；即，是连接现有的补偿范围还是创建补偿范围。 可能的值有: COMP_SPHERE_REQUIRED (2) COMP_SPHERE_SUPPORTS (4)
DISPLAY_NAME	字符串	流程的描述性名称。

QUERY_PROPERTY 视图:

这个预定义数据库视图用于查询流程级别变量。

表 16. QUERY_PROPERTY 视图中的列

列名	类型	注释
PIID	标识	流程实例标识。
VARIABLE_NAME	字符串	流程级别变量的名称。
NAME	字符串	查询属性的名称。
NAMESPACE	字符串	查询属性的名称空间。
GENERIC_VALUE	字符串	未映射至以下其中一个已定义类型的属性类型的字符串表示: STRING_VALUE、 NUMBER_VALUE、 DECIMAL_VALUE 或 TIMESTAMP_VALUE。
STRING_VALUE	字符串	如果属性类型已映射至字符串类型，那么这是该字符串的值。
NUMBER_VALUE	整型	如果属性类型已映射至整数类型，那么这是该整数的值。
DECIMAL_VALUE	十进制	如果属性类型已映射至浮点类型，那么这是该十进制的值。
TIMESTAMP_VALUE	时间戳记	如果属性类型已映射至时间戳记类型，那么这是时间戳记的值。

TASK 视图:

这个预定义数据库视图用于查询任务对象。

表 17. TASK 视图中的列

列名	类型	注释
TKIID	标识	任务实例的标识。
ACTIVATED	时间戳记	任务的激活时间。
APPLIC_ DEFAULTS_ID	标识	指定了任务缺省值的应用程序组件的标识。
APPLIC_NAME	字符串	任务所属企业应用程序的名称。
BUSINESS_ REL-EVANCE	布尔值	指定任务是否与业务相关。此属性影响对审计跟踪进行的日志记录。可能的值有: **TRUE** 任务与业务相关并且进行审计。 **FALSE** 任务与业务无关并且不进行审计。
COMPLETED	时间戳记	任务的完成时间。
CONTAINMENT_ CTX_ID	标识	任务的包含上下文。此属性确定任务的生命周期。删除任务的包含上下文时，也将删除该任务。

表 17. TASK 视图中的列 (续)

列名	类型	注释
CTX_ AUTHORIZATION	整型	允许任务所有者访问任务上下文。可能的值有: **AUTH_NONE** 不需要相关上下文对象的权限。 **AUTH_READER** 要对相关上下文对象执行操作，需要阅读者权限（例如，读取流程实例的属性）。
DUE	时间戳记	任务的到期时间。
EXPIRES	时间戳记	任务的到期日期。
FIRST_ACTIVATED	时间戳记	任务的第一次激活时间。
FOLLOW_ON_TKIID	标识	后续任务的实例的标识。
HIERARCHY_ POSITION	整型	可能的值有: **HIERARCHY_POSITION_TOP_TASK (0)** 任务层次结构中的顶级任务。 **HIERARCHY_POSITION_SUB_TASK (1)** 该任务是任务层次结构中的子任务。 **HIERARCHY_POSITION_FOLLOW_ON_TASK (2)** 该任务是任务层次结构中的后续任务。
IS_AD_HOC	布尔值	指示是在运行时期间动态地创建此任务，还是通过任务模板创建此任务。
IS_ESCALATED	布尔值	指示任务的升级是否已发生。
IS_INLINE	布尔值	指示任务是否是业务流程中的内联任务。
IS_WAIT_FOR_ SUB_TK	布尔值	指示父任务是否在等待子任务到达结束状态。
KIND	整型	任务的类型。可能的值有: **KIND_HUMAN (101)** 声明该任务是由人员创建和处理的*协作任务*。 **KIND_WPC_STAFF_ACTIVITY (102)** 声明该任务是这样一个人员任务，它是 WebSphere Business Integration Server Foundation V5 业务流程的人员活动。 **KIND_ORIGINATING (103)** 声明该任务是支持人机交互的*调用任务*，它使人员能够创建和启动服务。 **KIND_PARTICIPATING (105)** 声明该任务是支持人机交互的*待执行任务*，它使人员能够实现服务。 **KIND_ADMINISTRATIVE (106)** 声明该任务是管理任务。
LAST_MODIFIED	时间戳记	任务的上次修改时间。
LAST_STATE_ CHANGE	时间戳记	任务的状态的上次修改时间。

表 17. TASK 视图中的列 (续)

列名	类型	注释
NAME	字符串	任务的名称。
NAME_SPACE	字符串	用来对任务进行分类的名称空间。
ORIGINATOR	字符串	任务发起者的主体标识。
OWNER	字符串	任务所有者的主体标识。
PARENT_ CONTEXT_ID	字符串	任务的父上下文。此属性提供调用应用程序组件中相应上下文的键。父上下文是由创建任务的应用程序组件设置的。
PRIORITY	整型	任务的优先级。
RESUMES	时间戳记	自动继续执行任务的时间。
STARTED	时间戳记	任务的启动时间（STATE_RUNNING 和 STATE_CLAIMED）。
STARTER	字符串	任务启动者的主体标识。
STATE	整型	任务的状态。可能的值有: **STATE_READY (2)** 指示已准备好任务以进行声明。 **STATE_RUNNING (3)** 指示任务已启动并正在运行。 **STATE_FINISHED (5)** 指示任务已成功完成。 **STATE_FAILED (6)** 指示任务未成功完成。 **STATE_TERMINATED (7)** 指示任务已由于外部或内部请求而终止。 **STATE_CLAIMED (8)** 指示任务已被声明。 **STATE_EXPIRED (12)** 指示任务已由于超出指定的持续时间而结束。 **STATE_FORWARDED (101)** 指示任务已随后续任务一起完成。
SUPPORT_ AUTOCLAIM	布尔值	指示任务在被指定到单一用户时是否将自动地被声明。
SUPPORT_CLAIM_ SUSP	布尔值	指示任务在暂挂后是否能够被声明。
SUPPORT_ DELEGA-TION	布尔值	指示任务是否支持通过创建、删除或传递工作项来进行工作委托。
SUPPORT_ FOLLOW_ON	布尔值	指示此任务是否支持创建后续任务。
SUPPORT_SUB_TASK	布尔值	指示此任务是否支持创建子任务。
SUSPENDED	布尔值	指示任务是否已暂挂。
TKTID	标识	任务模板标识。
TOP_TKIID	标识	如果这是子任务，那么它为顶级父任务实例标识。

表 17. TASK 视图中的列 (续)

列名	类型	注释
TYPE	字符串	用来对任务进行分类的类型。

TASK_CPROP 视图:

这个预定义数据库视图用于查询任务对象的定制属性。

表 18. TASK_CPROP 视图中的列

列名	类型	注释
TKIID	字符串	任务实例标识。
NAME	字符串	属性的名称。
DATA_TYPE	字符串	非字符串定制属性的类的类型。
STRING_VALUE	字符串	字符串类型的定制属性值。

TASK_DESC 视图:

这个预定义数据库视图用于查询任务对象的多语言描述性数据。

表 19. TASK_DESC 视图中的列

列名	类型	注释
TKIID	字符串	任务实例标识。
LOCALE	字符串	与描述或显示名相关联的语言环境名。
DESCRIPTION	字符串	任务的描述。
DISPLAY_NAME	字符串	任务的描述性名称。

TASK_TEMPL 视图:

这个预定义数据库视图存放用来将任务实例化的数据。

表 20. TASK_TEMPL 视图中的列

列名	类型	注释
TKTID	字符串	任务模板标识。
VALID_FROM	时间戳记	任务模板变为可用于实例化的时间。
APPLIC_ DEFAULTS_ID	字符串	指定了任务模板缺省值的应用程序组件的标识。
APPLIC_NAME	字符串	任务模板所属企业应用程序的名称。
BUSINESS_ REL-EVANCE	布尔值	指定任务模板是否与业务相关。此属性影响对审计跟踪进行的日志记录。可能的值有: **TRUE** 任务与业务相关并且进行审计。 **FALSE** 任务与业务无关并且不进行审计。
CONTAINMENT_ CTX_ID	标识	任务模板的包含上下文。此属性确定任务模板的生命周期。删除包含上下文时，也将删除该任务模板。

表 20. TASK_TEMPL 视图中的列 (续)

列名	类型	注释
CTX_ AUTHORIZA-TION	整型	允许任务所有者访问任务上下文。可能的值有: **AUTH_NONE** 不需要相关上下文对象的权限。 **AUTH_READER** 要对相关上下文对象执行操作，需要阅读者权限（例如，读取流程实例的属性）。
DEFINITION_NAME	字符串	任务执行语言（TEL）文件中任务模板定义的名称。
DEFINITION_NS	字符串	TEL 文件中任务模板定义的名称空间。
IS_AD_HOC	布尔值	指示是在运行时期间动态创建此任务模板，还是在将该任务作为 EAR 文件的一部分部署时才创建此任务模板。
IS_INLINE	布尔值	指示任务模板是否是作为业务流程中的任务建模的。
KIND	整型	任务模板所派生的任务的类型。可能的值有: **KIND_HUMAN (101)** 声明该任务是由人员创建和处理的*协作任务*。 **KIND_ORIGINATING (103)** 声明该任务是支持人机交互的*调用任务*，它使人员能够创建和启动服务。 **KIND_PARTICIPATING (105)** 声明该任务是支持人机交互的*待执行任务*，它使人员能够实现服务。 **KIND_ADMINISTRATIVE (106)** 声明该任务是管理任务。
NAME	字符串	任务模板的名称。
NAMESPACE	字符串	用来对任务模板进行分类的名称空间。
PRIORITY	整型	任务模板的优先级。
STATE	整型	任务模板的状态。可能的值有: **STATE_STARTED (1)** 指定任务模板可用于创建任务实例。 **STATE_STOPPED (2)** 指定任务模板已停止。在此状态下，无法根据任务模板来创建任务实例。
SUPPORT_ AUTOCLAIM	布尔值	指示任务模板所派生的任务在被指定给单一用户时是否能够被自动声明。
SUPPORT_CLAIM_ SUSP	布尔值	指示任务模板所派生的任务在暂挂后是否能够被声明。
SUPPORT_ DELEGA-TION	布尔值	指示任务模板所派生的任务是否支持通过创建、删除或传递工作项进行工作委托。
SUPPORT_ FOLLOW_ON	布尔值	指示任务模板是否支持创建后续任务。
SUPPORT_SUB_TASK	布尔值	指示任务模板是否支持创建子任务。

表 20. TASK_TEMPL 视图中的列 (续)

列名	类型	注释
TYPE	字符串	用来对任务模板进行分类的类型。

TASK_TEMPL_CPROP 视图:

这个预定义数据库视图用于查询任务模板的定制属性。

表 21. TASK_TEMPL_CPROP 视图中的列

列名	类型	注释
TKTID	字符串	任务模板标识。
NAME	字符串	属性的名称。
DATA_TYPE	字符串	非字符串定制属性的类的类型。
STRING_VALUE	字符串	字符串类型的定制属性值。

TASK_TEMPL_DESC 视图:

这个预定义数据库视图用于查询任务模板对象的多语言描述性数据。

表 22. TASK_TEMPL_DESC 视图中的列

列名	类型	注释
TKTID	字符串	任务模板标识。
LOCALE	字符串	与描述或显示名相关联的语言环境名。
DESCRIPTION	字符串	任务模板的描述。
DISPLAY_NAME	字符串	任务模板的描述性名称。

WORK_ITEM 视图:

这个预定义数据库视图用于查询流程、任务和升级的工作项和授权数据。

表 23. WORK_ITEM 视图中的列

列名	类型	注释
WIID	标识	工作项标识。
OWNER_ID	字符串	所有者的主体标识。
GROUP_NAME	字符串	相关联的组工作列表的名称。
EVERYBODY	布尔值	指定是否每个人都拥有此工作项。

表 23. WORK_ITEM 视图中的列 (续)

列名	类型	注释
OBJECT_TYPE	整型	相关联的对象的类型。可能的值有: **OBJECT_TYPE_ACTIVITY (1)** 指定此工作项是为活动创建的。 **OBJECT_TYPE_PROCESS_INSTANCE (3)** 指定此工作项是为流程实例创建的。 **OBJECT_TYPE_TASK_INSTANCE (5)** 指定此工作项是为任务创建的。 **OBJECT_TYPE_TASK_TEMPLATE (6)** 指定此工作项是为任务模板创建的。 **OBJECT_TYPE_ESCALATION_ INSTANCE (7)** 指定此工作项是为升级实例创建的。 **OBJECT_TYPE_APPLICATION_ COMPONENT (9)** 指定此工作项是为应用程序组件创建的。
OBJECT_ID	标识	相关联的对象（例如相关联的流程或任务）的标识。
ASSOC_OBJECT_TYPE	整型	ASSOC_OID 属性所引用的对象的类型（例如任务、流程或外部对象）。使用 OBJECT_TYPE 属性的值。
ASSOC_OID	标识	具有相关联的工作项的对象的标识。例如，活动实例（为其创建了此工作项）所在流程实例的标识（PIID）。
REASON	整型	此工作项的分配原因。可能的值有: 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	时间戳记	工作项的创建日期和时间。

在查询中使用变量来过滤数据

查询结果返回与查询条件匹配的对象。您可能需要采用变量值来过滤这些结果。

关于此任务

您可以定义在运行时期间由流程在流程模型中使用的变量。可以为这些变量声明可查询的部分。

例如，John Smith 调用他的保险公司服务编号，以了解其受损汽车的保险索赔的进度。索赔管理员使用客户标识来查找该索赔。

过程

1. 可选: 列示流程中可查询的变量的属性。

   使用流程模板标识来识别该流程。如果知道可查询的变量，那么可以跳过此步骤。

   ```
   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. 列示具有与过滤器条件匹配的变量的流程实例。

   对于此流程，客户标识已建模为可查询的变量 customerClaim 的一部分。因此，您可以使用该客户标识来查找索赔。

   ```
   QueryResultSet result = process.query
       ("PROCESS_INSTANCE.NAME, QUERY_PROPERTY.STRING_VALUE",
        "QUERY_PROPERTY.VARIABLE_NAME = 'customerClaim' AND " +
        "QUERY_PROPERTY.NAME = 'customerID' AND " +
        "QUERY_PROPERTY.STRING_VALUE like 'Smith%'",
         (String)null, (Integer)null,
         (Integer)null, (TimeZone)null );
   ```

   此操作返回一个查询结果集，该结果集包含起始标识为 Smith 的客户的流程实例名和客户标识值。

管理存储查询

存储查询提供了保存经常运行的查询的方法。存储查询可以是可供所有用户使用的查询（公用查询），也可以是属于特定用户的查询（专用查询）。

关于此任务

存储查询是指存储在数据库中并由名称标识的查询。专用存储查询和公用存储查询可以具有相同的名称；来自不同拥有者的专用存储查询也可以具有相同的名称。

您可以使用业务流程对象或任务对象的存储查询，也可以同时使用这两种对象类型的存储查询。

相关概念

第 29 页的『存储查询中的参数』

存储查询是指存储在数据库中并由名称标识的查询。当运行该查询时，将动态地汇编标准元组。为了使存储查询可重复使用，您可以使用在运行时期间解析的查询定义中的参数。

管理公用存储查询:

公用存储查询由系统管理员创建。这些查询可供所有用户使用。

关于此任务

作为系统管理员，您可以创建、查看和删除公用存储查询。如果在 API 调用中未指定用户标识，那么假定该存储查询是公用存储查询。

过程

1. 创建公用存储查询。

   例如，下列代码段为流程实例创建存储查询，并使用名称 CustomerOrdersStartingWithA 来保存该查询。

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

   存储查询的结果是所有以字母 A 开头的流程实例名及其相关流程实例标识（PIID）的已排序列表。

2. 运行由存储查询定义的查询。

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

   此操作返回满足条件的对象。在本例中，此操作返回所有以 A 开头的客户订单。

3. 列示可用的公用存储查询的名称。

   下列代码段说明如何只将已返回查询的列表限制为公用查询。

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

4. 可选: 检查特定存储查询定义的查询。

   专用存储查询可以与公用存储查询同名。如果这些名称相同，那么将返回专用存储查询。下列代码段说明如何仅返回带指定名称的公用查询。如果要为基于任务的项目运行此查询，那么指定 StoredQuery 作为已返回的对象类型，而不是指定 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. 删除公用存储查询。

   下列代码段说明如何删除步骤 1 中创建的存储查询。

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

为其他用户管理专用存储查询:

任何用户都可以创建专用查询。这些查询只有查询的所有者和系统管理员可以使用。

关于此任务

作为系统管理员，您可以管理属于特定用户的专用存储查询。

过程

1. 为用户标识 Smith 创建专用存储查询。

例如，下列代码段为流程实例创建存储查询，并使用名称 CustomerOrdersStartingWithA 为该用户标识 Smith 保存该查询。

```
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);
```

存储查询的结果是所有以字母 A 开头的流程实例名及其相关流程实例标识（PIID）的已排序列表。

2. 运行由存储查询定义的查询。

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

此操作返回满足条件的对象。在本例中，此操作返回所有以 A 开头的客户订单。

3. 获取属于特定用户的专用查询的名称列表。

例如，下列代码段说明如何获取属于用户 Smith 的专用查询的列表。

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

4. 查看特定查询的详细信息。

下列代码段说明如何查看用户 Smith 拥有的 CustomerOrdersStartingWithA 查询的详细信息。

```
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. 删除专用存储查询。

下列代码段说明如何删除用户 Smith 拥有的专用查询。

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

处理专用存储查询:

如果您不是系统管理员，那么可以创建、运行和删除自己的专用存储查询。还可以使用系统管理员创建的公用存储查询。

过程

1. 创建专用存储查询。

例如，下列代码段为流程实例创建存储查询，并使用特定的名称来保存该查询。如果未指定用户标识，那么假定该存储查询是已登录用户的专用存储查询。

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

此查询返回所有以字母 A 开头的流程实例名及其相关流程实例标识（PIID）的已排序列表。

2. 运行由存储查询定义的查询。

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

此操作返回满足条件的对象。在本例中，此操作返回所有以 A 开头的客户订单。

3. 获取已登录用户可以访问的存储查询的名称列表。

下列代码段说明如何获取用户可以访问的公用存储查询和专用存储查询。

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

4. 查看特定查询的详细信息。

下列代码段说明如何查看用户 Smith 拥有的 CustomerOrdersStartingWithA 查询的详细信息。

```
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. 删除专用存储查询。

下列代码段说明如何删除专用存储查询。

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

为业务流程开发应用程序

业务流程是一组与业务相关的活动，您按特定顺序调用这些活动以实现业务目标。所提供的示例说明了如何为流程的典型操作开发应用程序。

关于此任务

业务流程可以是微流动，也可以是长期运行的流程:

- 微流动是可同步执行的运行时间较短的业务流程。在一段非常短的时间过后，会将结果返回给调用者。
- 可中断的长期运行的流程是作为一系列链接到一起的活动执行。使用流程中的某些构造在流程流中导致中断，例如，调用人员任务、使用同步绑定来调用服务或使用计时器驱动的活动。

  通常会以异步方式浏览流程的并行分支，即，并发执行并行分支中的活动。根据活动的类型和事务设置的不同，活动可以在它自己的事务中运行。

对流程实例执行的操作所需的角色

对 BusinessFlowManager 接口的访问权并不保证调用者可以对流程执行所有操作。该调用者必须使用有权执行该操作的角色来登录该客户机应用程序。

下表说明了特定角色能够对流程实例执行的操作。

操作	调用者的主体角色		
	阅读者	启动者	管理员
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspend			x
transferWorkItem			x

对业务流程活动执行的操作所需的角色

对 BusinessFlowManager 接口的访问权并不保证调用者可以对活动执行所有操作。该调用者必须使用有权执行该操作的角色来登录该客户机应用程序。

下表说明了特定角色能够对活动实例执行的操作。

操作	调用者的主体角色				
	阅读者	编辑者	潜在所有者	所有者	管理员
cancelClaim				x	x
claim			x		x

操作	调用者的主体角色				
	阅读者	编辑者	潜在所有者	所有者	管理员
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 只能传递给潜在所有者或管理员	x

管理业务流程的生命周期

当调用可以启动流程的业务流程编排器 API 方法时，流程实例就存在了。流程实例的导航将继续进行到它的所有活动都处于结束状态为止。您可以对该流程实例执行各种操作来管理其生命周期。

关于此任务

所提供的示例说明了如何为流程的下列典型生命周期操作开发应用程序。

启动业务流程:

业务流程的启动方式取决于该流程是微流动还是长期运行的流程。启动流程的服务对于流程的启动方式来说也很重要；流程可以有唯一的启动服务，也可以有若干个启动服务。

关于此任务

所提供的示例说明了如何为微流动和长期运行的流程的典型启动方案开发应用程序。

运行包含唯一启动服务的微流动:

微流动可以由 receive 活动或 pick 活动启动。如果微流动由 receive 活动启动，或者 pick 活动只有一个 onMessage 定义，那么启动服务是唯一的。

关于此任务

如果微流动实现了“请求/响应”操作（即，该流程包含应答），那么可使用 call 方法来运行该流程并传递流程模板名作为调用参数。

如果微流动是单向操作，请使用 sendMessage 方法来运行该流程。本示例未阐述此方法。

过程

1. 可选: 列示流程模板以查找所要运行的流程的名称。

   如果您已知道该流程的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询将返回一个数组，该数组包含 call 方法可以启动的前 50 个已排序模板。

2. 使用适当类型的输入消息来启动流程。

   创建消息时，必须指定其消息类型名，以便包含消息定义。

   ```
   ProcessTemplateData template = processTemplates[0];
   //create a message for the single starting receive activity
   ClientObjectWrapper input = process.createMessage
                               (template.getID(),
                                template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }

   //run the process
   ClientObjectWrapper output = process.call(template.getName(), input);
   DataObject myOutput = null;
   if ( output.getObject() != null && output.getObject() instanceof DataObject )
   {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
   }
   ```

此操作创建流程模板 CustomerTemplate 的实例并传递一些客户数据。仅当该流程完成后，此操作才会返回。而且，该流程的结果（OrderNo）将返回给调用者。

运行包含非唯一启动服务的微流动:

微流动可以由 receive 活动或 pick 活动启动。如果微流动由带有多个 onMessage 定义的 pick 活动启动，那么启动服务不是唯一的。

关于此任务

如果微流动实现了“请求/响应”操作（即，该流程包含应答），那么可使用 call 方法来运行该流程并在进行调用时传递启动服务的标识。

如果微流动是单向操作，那么使用 sendMessage 方法来运行该流程。本示例未阐述此方法。

过程

1. 可选: 列示流程模板以查找所要运行的流程的名称。

   如果您已知道该流程的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询将返回一个数组，该数组包含前 50 个可以作为微流动启动的已排序模板。

2. 确定要调用的启动服务。

   本示例使用找到的第一个模板。

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

3. 使用适当类型的输入消息来启动流程。

   创建消息时，必须指定其消息类型名，以便包含消息定义。

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input =
         process.createMessage(activity.getServiceTemplateID(),
                               activity.getActivityTemplateID(),
                               activity.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //run the process
   ClientObjectWrapper output = process.call(activity.getServiceTemplateID(),
                                             activity.getActivityTemplateID(),
                                             input);
   //check the output of the process, for example, an order number
   DataObject myOutput = null;
   ```

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

此操作创建流程模板 CustomerTemplate 的实例并传递一些客户数据。仅当该流程完成后，此操作才会返回。而且，该流程的结果（OrderNo）将返回给调用者。

启动包含唯一启动服务的长期运行的流程:

如果启动服务是唯一的，那么可使用 initiate 方法并传递流程模板名作为参数。当使用单个 receive 或 pick 活动来启动长期运行的流程时，以及当单个 pick 活动只有一个 onMessage 定义时，启动服务就是唯一的。

过程

1. 可选: 列示流程模板以查找所要启动的流程的名称。

   如果您已知道该流程的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询将返回一个数组，该数组包含 initiate 方法可以启动的前 50 个已排序模板。

2. 使用适当类型的输入消息来启动流程。

   创建消息时，必须指定其消息类型名，以便包含消息定义。如果指定流程实例名，那么该名称不能以下划线开头。如果未指定流程实例名，那么将使用字符串格式的流程实例标识（PIID）作为名称。

   ```
   ProcessTemplateData template = processTemplates[0];
   //create a message for the single starting receive activity
   ClientObjectWrapper input = process.createMessage
                               (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //start the process
   PIID piid = process.initiate(template.getName(), "CustomerOrder", input);
   ```

   此操作创建实例 CustomerOrder 并传递一些客户数据。当该流程启动时，此操作会将新流程实例的对象标识返回给调用者。

   流程实例的启动者将被设置为该请求的调用者。此人员将接收到该流程实例的工作项。在操作中将确定该流程实例的流程管理员、阅读者和编辑者，他们将接收该流程实例的工作项。将确定后续活动实例。这些后续活动实例将自动启动，或者，如果它们是人员任务、receive 活动或 pick 活动，那么就会为潜在所有者创建工作项。

启动包含非唯一启动服务的长期运行的流程:

可以通过多个启动 receive 活动或 pick 活动来启动长期运行的流程。可以使用 initiate 方法来启动流程。如果启动服务不是唯一的（例如，流程是通过多个 receive 活动或 pick 活动启动的，或者由具有多个 onMessage 定义的单个 pick 活动启动），那么必须标识要调用的服务。

过程

1. 可选: 列示流程模板以查找所要启动的流程的名称。

   如果您已知道该流程的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询将返回一个数组，该数组包含前 50 个可以作为长期运行的流程启动的已排序模板。

2. 确定要调用的启动服务。

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

3. 使用适当类型的输入消息来启动流程。

   创建消息时，必须指定其消息类型名，以便包含消息定义。如果指定流程实例名，那么该名称不能以下划线开头。如果未指定流程实例名，那么将使用字符串格式的流程实例标识（PIID）作为名称。

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input = process.createMessage
                               (activity.getServiceTemplateID(),
                                activity.getActivityTemplateID(),
                                activity.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //start the process
   PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                                   activity.getActivityTemplateID(),
                                   input);
   ```

   此操作创建实例并传递一些客户数据。当该流程启动时，此操作会将新流程实例的对象标识返回给调用者。

   流程实例的启动者将被设置为该请求的调用者，它将接收该流程实例的工作项。在操作中将确定该流程实例的流程管理员、阅读者和编辑者，他们将接收该流程实例的工作项。将确定后续活动实例。这些后续活动实例将自动启动，或者，如果它们是人员任务、receive 活动或 pick 活动，那么就会为潜在所有者创建工作项。

暂挂和恢复业务流程:

可以将处于“正在运行”状态的顶级长期运行的流程实例暂挂，然后再次恢复以完成该流程实例。

开始之前

调用者必须是流程实例管理员或业务流程管理员。要暂挂流程实例，它必须处于“正在运行”状态或“将要失败”状态。

关于此任务

例如，您可能需要暂挂流程实例，以便可以配置对该流程稍后要使用的后端系统的访问。满足该流程的先决条件后，可以恢复该流程实例。还可以暂挂流程以解决正在导致流程实例发生故障的问题，然后在问题解决之后再次恢复该流程。

过程

1. 获取处于“正在运行”状态并且要暂挂的流程 CustomerOrder。

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

2. 暂挂该流程实例。

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

   此操作暂挂指定的顶级流程实例。该流程实例将进入“已暂挂”状态。将 autonomy 属性设置为 child 的子流程如果处于“正在运行”、“将要失败”、“正在终止”或“正在补偿”状态，则它们也将被暂挂。还可以暂挂与此流程实例相关联的内联任务；但不能暂挂与此流程实例相关联的独立任务。

   在此状态下，仍然可以完成已启动的活动，但不会激活新的活动，例如，可以完成处于已声明状态的人员任务活动。

3. 恢复该流程实例。

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

   此操作使该流程实例及其子流程进入它们被暂挂前所处的状态。

重新启动业务流程:

可以重新启动处于“已完成”、“已终止”、“已失败”或“已补偿”状态的流程实例。

开始之前

调用者必须是流程实例管理员或业务流程管理员。

关于此任务

重新启动流程实例与第一次启动流程实例类似。但是，重新启动流程实例时，流程实例标识是已知的，并且已有该实例的输入消息。

如果该流程有多个能够创建流程实例的 receive 活动或 pick 活动（也称为 receive choice 活动），那么将使用所有属于这些活动的消息来重新启动该流程实例。如果任何这些活动实现了“请求/响应”操作，那么将在对相关应答活动进行导航时再次发送响应。

过程

1. 获取要重新启动的流程。

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

2. 重新启动该流程实例。

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

此操作将重新启动指定的流程实例。

终止流程实例:

有时，具有流程管理员权限的用户需要终止已知处于不可恢复状态的顶级流程实例。由于流程实例将立即终止，而不会等待任何未完成的子流程或活动完成，所以只有在异常情况下才应该终止流程实例。

过程

1. 检索要终止的流程实例。

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

2. 终止该流程实例。

   如果要终止流程实例，可以在进行补偿或不进行补偿的情况下终止该流程实例。

   要在进行补偿的情况下终止该流程实例:

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

   要在不进行补偿的情况下终止该流程实例:

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

   如果在进行补偿的情况下终止流程实例，那么会运行流程的补偿，就好像在顶级范围上发生了故障一样。如果在未进行补偿的情况下终止流程实例，那么该流程实例将立即终止，而不会等待活动、待执行任务或内联调用任务正常完成。

   强制终止请求不会终止流程和与流程相关的独立任务所启动的应用程序。如果要终止这些应用程序，那么必须在流程应用程序中添加语句以显式地终止该流程启动的应用程序。

删除流程实例:

如果在流程模型中为流程模板设置了相应的属性，就会自动地从业务流程编排器数据库中删除已完成的流程实例。例如，您可能需要将流程实例保留在数据库中，以便查询未写入审计日志的流程实例的数据。但是，已存储的流程实例数据并不仅影响磁盘空间和性能，而且还可防止创建那些使用同一个相关集合值的流程实例。因此，应该定期地删除数据库中的流程实例数据。

关于此任务

要删除流程实例，您需要具有流程管理员权限，并且该流程实例必须是顶级流程实例。

以下示例说明如何删除所有已完成的流程实例。

过程

1. 列示已完成的流程实例。

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

   此操作将返回一个查询结果集，该结果集列示了已完成的流程实例。

2. 删除已完成的流程实例。

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

   此操作将从数据库中删除所选的流程实例及其内联任务。

处理人员任务活动

业务流程中的人员任务活动通过工作项被指定到企业中的各个人员。当流程启动时，就会为潜在所有者创建工作项。

关于此任务

激活人员任务活动后，将创建活动实例和相关的待执行任务。人员任务活动的处理和工作项管理将托付给人员任务管理器。该活动实例的任何状态更改都会在任务实例中反映出来，反之亦然。

潜在所有者声明了该活动。此人员负责提供相关信息并完成该活动。

过程

1. 列示准备要处理的已登录人员负责的活动:

```
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);
```

   此操作返回一个查询结果集，该结果集包含已登录人员能够处理的活动。

2. 声明要处理的活动:

```
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();
```

```
    // read the values
    ...
  }
}
```

声明活动时，将返回该活动的输入消息。

3. 当活动处理完毕，将完成该活动。 该活动可能成功完成，也可能以报告故障消息而告终。如果该活动成功，那么将传递输出消息。如果该活动不成功，那么将使该活动处于“已失败”或“已停止”状态，并且将传递故障消息。必须为这些操作创建适当的消息。创建消息时，必须指定消息类型名，以便包含消息定义。

   a. 要成功地完成活动，请创建输出消息。

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the activity
process.complete(aiid, output);
```

   此操作将设置包含订单号的输出消息。

   b. 要在发生故障时完成活动，请创建故障消息。

```
//retrieve the faults modeled for the human task activity
List faultNames = process.getFaultNames(aiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      process.createMessage(aiid, faultNames.get(0));

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

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

   此操作设置处于“已失败”或“已停止”状态的活动。如果流程模型中该活动的 **continueOnError** 参数设置为 true，就会使该活动处于“已失败”状态，并且导航将继续进行。如果将 **continueOnError** 参数设置为 false，并且在周围的作用域中未捕获故障，那么该活动就会进入“已停止”状态。在此状态下，可以通过强制完成或强制重试来修复该活动。

处理单人工作流

某些工作流仅由一个人执行，例如，从在线书店订购书籍。此类工作流没有并行路径。completeAndClaimSuccessor API 支持此类工作流的处理。

关于此任务

买方于在线书店中完成一系列操作来订购书籍。此系列的操作可以作为一系列人员任务活动（待执行任务）来实现。如果买方决定订购多本书，那么这相当于声明下一个人员任务活动。由于用户界面定义与那些控制用户界面中对话框的流的活动相关联，所以此类工作流也称为*页顺序*。

completeAndClaimSuccessor API 完成人员任务活动，并在同一个流程实例中为已登录人员声明下一个人员任务活动。它返回有关下一个已声明活动的信息，其中包括有关要处理的输入消息的信息。由于已使下一个活动在所完成活动的同一个事务中可用，所以在流程模型中必须将所有人员任务活动的事务行为设置为 participates。

将此示例与使用业务流程管理器 API 和人员任务管理器 API 的示例相比较。

过程

1. 声明一系列活动中的第一个活动。

```
//
//Query the list of activities that can be claimed by the logged-on user
//
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);
...
//
//Claim the first activity
//
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();
    // read the values
    ...
  }
}
```

   声明活动时，将返回该活动的输入消息。

2. 当处理完活动后，完成该活动并声明下一个活动。

   要完成该活动，将传递输出消息。创建输出消息时，必须指定消息类型名，以便包含消息定义。

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}
```

```
//complete the activity and claim the next one
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

此操作将设置包含订单号的输出消息并声明该序列中的下一个活动。如果已为后续活动设置 AutoClaim，且存在多个可以使用的路径，那么会声明所有后续活动，并且返回随机活动以作为下一个活动。如果没有其他后续活动可分配给此用户，那么将返回 Null。

如果该流程包含可使用的并行路径，并且这些路径包含人员任务活动，已登录用户是多个此类活动的潜在所有者，那么将会自动声明随机活动并将其作为下一个活动返回。

3. 处理下一个活动。

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

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

4. 继续执行步骤 2 以完成该活动。

相关任务

第 93 页的『处理包含人员任务的单人工作流程』

某些工作流仅由一个人执行，例如，从在线书店订购书籍。此示例说明如何将订购书籍的操作顺序作为一系列人员任务活动（待执行任务）实现。业务流程管理器和人员任务管理器 API 均用于处理工作流程。

向处于“正在等待”状态的活动发送消息

您可以使用入站消息活动（receive 活动、pick 活动中的 onMessage 以及事件处理程序中的 onEvent），以便使运行中的流程与“外部世界”的事件同步。例如，接收客户发送的电子邮件以响应信息请求就是这样的事件。

关于此任务

您可以使用发端任务来将消息发送至该活动。

过程

1. 在具有特定流程实例标识的流程实例中列示正在等待已登录用户的消息的活动服务模板。

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

2. 将消息发送至第一个等待服务。

假定第一个服务是您要服务的对象。调用者必须是接收消息的活动的潜在启动者，或者是流程实例的管理员。

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

```
// create a message for the service to be called
  ClientObjectWrapper message =
        process.createMessage(vtid,atid,inputMessageTypeName);
  DataObject myMessage = null;
  if ( message.getObject()!= null && message.getObject() instanceof DataObject )
  {
    myMessage = (DataObject)message.getObject();
    //set the strings in the message, for example, chocolate is to be ordered
    myMessage.setString("Order", "chocolate");
  }

  // send the message to the waiting activity
  process.sendMessage(vtid, atid, message);
}
```

此操作将指定的消息发送至处于“正在等待”状态的活动服务并传递一些订单数据。

您还可以指定流程实例标识以确保将该消息发送至指定的流程实例。如果未指定流程实例标识，就会将该消息发送至活动服务和由该消息中的关联值标识的流程实例。如果指定了流程实例标识，就会对使用关联值找到的流程实例进行检查，以确保它具有指定的流程实例标识。

处理事件

整个业务流程及其每个作用域都可以有相关联的事件处理程序，当相关联的事件发生时，就会调用那些事件处理程序。就流程可以使用事件处理程序来提供 Web Service 操作而言，事件处理程序与 receive 活动或 pick 活动类似。

关于此任务

只要相应的作用域处于运行状态，就可以多次调用事件处理程序。此外，可以同时激活事件处理程序的多个实例。

以下代码段说明如何获取给定流程实例的活动事件处理程序以及如何发送输入消息。

过程

1. 确定流程实例标识的数据并列示该流程的活动事件处理程序。

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

2. 发送输入消息。

   本示例使用找到的第一个事件处理程序。

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

    // create a message for the service to be called
    ClientObjectWrapper input = process.createMessage(
    event.getID(), event.getInputMessageTypeName());

    if (input.getObject() != null && input.getObject() instanceof DataObject )
    {
       DataObject inputMessage = (DataObject)input.getObject();
       // set content of the message, for example, a customer name, order number
       inputMessage.setString("CustomerName", "Smith");
       inputMessage.setString("OrderNo", "2711");
```

```
          // send the message
          process.sendMessage( event.getProcessTemplateName(),
                               event.getPortTypeNamespace(),
                               event.getPortTypeName(),
                               event.getOperationName(),
        input );
        }
    }
```

此操作将指定的消息发送至流程的活动事件处理程序。

分析流程结果

流程可以显示已建模为 Web 服务描述语言（WSDL）单向操作或“请求/响应”操作的 Web Service 操作。因为具有单向接口的长期运行的流程没有输出，所以使用 getOutputMessage 方法无法检索该流程的结果。然而，您可以查询变量的内容。

关于此任务

仅当用于派生流程实例的流程模板未指定自动删除所派生的流程实例时，流程结果才会存储在数据库中。

过程

分析流程结果，例如，检查订单号。

```
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");
  }
}
```

修复活动

长期运行的流程可以包含运行时间同样较长的活动。这些活动可能会遇到未捕获的错误并进入“已停止”状态。处于运行状态的活动也可能会表现为未响应。在这两种情况下，流程管理员都可以通过多种方法对该活动执行操作，以使流程的导航能够继续进行。

关于此任务

业务流程编排器 API 提供了 forceRetry 和 forceComplete 方法来修复活动。所提供的示例说明了如何在应用程序中添加活动修复操作。

强制完成活动: **关于此任务**

长期运行的流程中的活动有时会遇到故障。如果外层作用域中的故障处理程序未捕获这些故障，并且相关联的活动模板指定该活动在出错时应该停止，该活动就会进入“已停止”状态以便进行修复。在此状态下，可以强制完成该活动。

例如，对于处于“正在运行”状态的活动来说，如果该活动未响应，那么也可以强制使其完成。

某些类型的活动有一些其他要求。

人员任务活动

可以在强制完成调用中传递参数，例如应该已发送的消息或者应该已发生的故障。

脚本活动

不能在强制完成调用中传递参数。但是，必须设置需要修复的变量。

调用活动

如果调用活动调用了不是子流程的异步服务，并且这些活动处于“正在运行”状态，那么也可以强制完成这些活动。例如，如果该异步服务被调用并且未响应，您可能会希望强制完成调用活动。

过程

1. 列示处于“已停止”状态的已停止活动。

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

   此操作将返回 CustomerOrder 流程实例的已停止活动。

2. 完成活动，例如，完成已停止的人员任务活动。

   在本示例中，传递了输出消息。

```
if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);
   ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper output =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( output.getObject()!= null && output.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)output.getObject();
     //set the parts in your message, for example, an order number
     myMessage.setInt("OrderNo", 4711);
   }

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

   此操作将完成该活动。如果发生错误，那么 **continueOnError** 参数将确定在 forceComplete 请求时出现故障的时候要执行的操作。

在本示例中，**continueOnError** 值是 true。此值表示如果出现故障，那么该活动会进入“已失败”状态。此故障将传播到该活动的外层作用域，直到它被处理或者到达流程作用域为止。如果该故障到达流程作用域，该流程就会进入“将要失败”状态并最终进入“已失败”状态。

重新尝试执行已停止的活动: 关于此任务

如果长期运行的流程中的活动在外层作用域中遇到未捕获的故障，并且相关联的活动模板指定该活动在出错时应该停止，那么该活动会进入“已停止”状态以便进行修复。您可以重新尝试执行该活动。

您可以设置该活动使用的变量。还可以在强制重试调用中传递参数，例如该活动所需的消息，但这不适用于脚本活动。

过程

1. 列示已停止的活动。

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

   此操作将返回 CustomerOrder 流程实例的已停止活动。

2. 重新尝试执行该活动，例如，执行已停止的人员任务活动。

```
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();
      //set the strings in your message, for example, chocolate is to be ordered
      myMessage.setString("OrderNo", "chocolate");
    }

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

   此操作将重试该活动。**continueOnError** 参数确定了 forceRetry 请求处理期间出错时执行的操作。

   在本示例中，**continueOnError** 值是 true。此值表示如果 forceRetry 请求处理期间出错，该活动就会进入“已失败”状态。此故障将传播到该活动的外层作用域，直到它被处理或者到达流程作用域为止。然后，流程就会进入“将要失败”状态，并且在该流程状态最终进入“已失败”状态之前，在流程级别运行故障处理程序。

BusinessFlowManagerService 接口

BusinessFlowManagerService 接口公布可以由客户机应用程序调用的业务流程函数。

可以由 BusinessFlowManagerService 接口调用的方法取决于流程状态或活动状态，并取决于包含该方法的应用程序的用户所具有的权限。下面列示了业务流程对象的主要处

理方法。有关这些方法以及 BusinessFlowManagerService 接口中提供的其他方法的更多信息，请参阅 com.ibm.bpe.api 包中的 Javadoc。

流程模板

流程模板是已版本化、已部署并且已安装的流程模型，它包含业务流程的规范。可以通过发出诸如 sendMessage() 之类适当的请求来将模板实例化并将它启动。流程实例的执行是由服务器自动驱动的。

表 24. 流程模板的 API 方法

方法	描述
getProcessTemplate	检索指定的流程模板。
queryProcessTemplates	检索数据库中存储的流程模板。

流程实例

下列 API 方法与启动流程实例相关。

表 25. API 方法与启动流程实例相关

方法	描述
call	创建并运行微流动。
callWithReplyContext	根据指定的流程模板创建并运行具有唯一启动服务的微流动或者具有唯一启动服务的长期运行的流程。该调用将以异步方式等待结果。
callWithUISettings	创建并运行微流动，然后返回输出消息和客户机用户界面（UI）设置。
initiate	创建流程实例并启动该流程实例的处理。可对长期运行的流程使用此方法。也可以对要激发但忘记激发的微流动使用此方法。
sendMessage	将指定的消息发送至指定的活动服务和流程实例。如果具有同一个相关集合值的流程实例不存在，那么将创建该实例。该流程可以具有唯一的启动服务，也可以具有不唯一的启动服务。
getStartActivities	返回有关可以根据所指定流程模板启动流程实例的活动的信息。
getActivityServiceTemplate	检索指定的活动服务模板。

表 26. 用于控制流程实例生命周期的 API 方法

方法	描述
suspend	暂停执行处于“正在运行”或“将要失败”状态的顶级长期运行的流程实例。
resume	继续执行处于“已暂挂”状态的顶级长期运行的流程实例。
restart	重新启动处于“已完成”、“已失败”或“已终止”状态的顶级长期运行的流程实例。

表 26. 用于控制流程实例生命周期的 API 方法 (续)

方法	描述
forceTerminate	终止指定的顶级流程实例、其 autonomy 属性值为 child 的子流程及其处于“正在运行”、“已声明”或“正在等待”状态的活动。
delete	删除指定的顶级流程实例及其 autonomy 属性值为 child 的子流程。
query	从数据库中检索与搜索条件匹配的属性。

活动

对于调用活动来说，可以在流程模型中指定这些活动在出错时将继续执行。如果将 continueOnError 标志设置为 false，并且发生了未处理的错误，那么该活动就会进入“已停止”状态。然后，流程管理员就可以修复该活动。例如，在调用活动偶尔失败但补偿建模和故障处理成本过高的长期运行的流程中，可以使用 continueOnError 标志和相关联的修复函数。

下列方法可用于处理和修复活动。

表 27. 用于控制活动实例生命周期的 API 方法

方法	描述
claim	声明已就绪的活动实例，以便用户处理该活动。
cancelClaim	取消声明活动实例。
complete	完成活动实例。
completeAndClaimSuccessor	完成活动实例，并在同一个流程实例中为已登录人员声明下一个活动实例。
forceComplete	强制完成处于“正在运行”或“已停止”状态的活动实例。
forceRetry	强制重复处于“正在运行”或“已停止”状态的活动实例。
query	从数据库中检索与搜索条件匹配的属性。

变量和定制属性

接口提供了用于检索和设置变量值的 get 和 set 方法。还可以使指定的属性与流程实例和活动实例相关联，并可以从这些实例中检索指定的属性。定制属性名和属性值的类型必须是 java.lang.String。

表 28. 变量和定制属性的 API 方法

方法	描述
getVariable	检索指定的变量。
setVariable	设置指定的变量。
getCustomProperty	从指定的活动实例或流程实例中检索指定的定制属性。
getCustomProperties	从指定的活动实例或流程实例中检索定制属性。

表 28. 变量和定制属性的 API 方法 (续)

方法	描述
getCustomPropertyNames	从指定的活动实例或流程实例中检索定制属性的名称。
setCustomProperty	为指定的活动实例或流程实例存储特定定制属性的值。

为人员任务开发应用程序

任务是组件以服务形式调用人员或者人员调用服务时采用的方法。本书提供了人员任务的典型应用程序示例。

关于此任务

有关人员任务管理器 API 的更多信息，请参阅 com.ibm.task.api 包中的 Javadoc。

启动用于调用同步接口的调用任务

调用任务与服务组件体系结构（SCA）组件相关联。当任务启动后，它会调用 SCA 组件。仅在您可以同步调用关联的 SCA 组件时，才会同步启动调用任务。

关于此任务

例如，此类 SCA 组件可以作为微流动或简单 Java 类实现。

本方案创建任务模板的实例并传递一些客户数据。该任务在双向操作返回前将一直处于“正在运行”状态。会将该任务的结果（OrderNo）返回给调用者。

过程

1. 可选: 列示任务模板以查找所要运行的调用任务的名称。

   如果您已知道该任务的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询返回一个数组，该数组包含前 50 个已排序的发起模板。

2. 创建适当类型的输入消息。

   ```
   TaskTemplate template = taskTemplates[0];

   // create a message for the selected task
   ClientObjectWrapper input = task.createInputMessage( template.getID());
   DataObject myMessage = null ;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the parts in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   ```

3. 创建该任务并以同步方式运行它。

要以同步方式运行任务，它必须是双向操作。本示例使用 createAndCallTask 方法来创建和运行任务。

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

4. 分析任务结果。

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

启动用于调用异步接口的调用任务

调用任务与服务组件体系结构（SCA）组件相关联。当任务启动后，它会调用 SCA 组件。仅在您可以异步调用关联的 SCA 组件时，才会异步启动调用任务。

关于此任务

例如，此类 SCA 组件可以作为长期运行的流程或单向操作实现。

本方案创建任务模板的实例并传递一些客户数据。

过程

1. 可选: 列示任务模板以查找所要运行的调用任务的名称。

   如果您已知道该任务的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询返回一个数组，该数组包含前 50 个已排序的发起模板。

2. 创建适当类型的输入消息。

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. 创建该任务并以异步方式运行它。

   本示例使用 createAndStartTask 方法来创建和运行任务。

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

创建和启动任务实例

本方案说明如何创建定义了协作任务（在 API 中也称为*人员任务*）的任务模板实例以及如何启动该任务实例。

过程

1. 可选: 列示任务模板以查找所要运行的协作任务的名称。

   如果您已知道该任务的名称，那么此步骤是可选的。

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

   结果将按名称排序。此查询返回一个数组，该数组包含前 50 个已排序的任务模板。

2. 创建适当类型的输入消息。

   ```
   TaskTemplate template = taskTemplates[0];

   // create a message for the selected task
   ClientObjectWrapper input = task.createInputMessage( template.getID());
   DataObject myMessage = null ;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the parts in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   ```

3. 创建并启动该协作任务；在本示例中未指定应答处理程序。

   本示例使用 createAndStartTask 方法来创建和启动任务。

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

   为该任务实例的相关人员创建了工作项。例如，潜在所有者可以声明新的任务实例。

4. 声明任务实例。

   ```
   ClientObjectWrapper input2 = task.claim(tkiid);
   DataObject taskInput = null ;
   if ( input2.getObject()!= null && input2.getObject() instanceof DataObject )
   {
     taskInput = (DataObject)input2.getObject();
     // read the values
     ...
   }
   ```

   声明任务实例时，将返回该任务的输入消息。

处理待执行任务或协作任务

待执行任务（在 API 中也称为*参与任务*）或协作任务（在 API 中也称为*人员任务*）通过工作项被指定到企业中的各个人员。例如，待执行任务及其相关工作项是在流程进行到人员任务活动时创建的。

关于此任务

其中一个潜在所有者声明与该工作项相关联的任务。此人员负责提供相关信息并完成该任务。

过程

1. 列示准备要处理的已登录人员负责的任务。

```
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);
```

   此操作返回一个查询结果集，该结果集包含已登录人员能够处理的任务。

2. 声明要处理的任务。

```
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();
    // read the values
    ...
  }
}
```

   声明任务时，将返回该任务的输入消息。

3. 当任务处理完毕，将完成该任务。

   该任务可能成功完成，也可能以报告故障消息而告终。如果该任务成功，那么将传递输出消息。如果该任务不成功，那么将传递故障消息。必须为这些操作创建适当的消息。

   a. 要成功地完成任务，请创建输出消息。

```
ClientObjectWrapper output =
      task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the task
task.complete(tkiid, output);
```

      此操作将设置包含订单号的输出消息。该任务将进入“已完成”状态。

   b. 要在发生故障时完成任务，请创建故障消息。

```
//retrieve the faults modeled for the task
List faultNames = task.getFaultNames(tkiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      task.createFaultMessage(tkiid, (String)faultNames.get(0));
```

```
// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

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

此操作设置包含错误代码的故障消息。该任务将进入“已失败”状态。

暂挂和恢复任务实例

您可以暂挂协作任务实例（在 API 中也称为*人员任务*）或待执行任务实例（在 API 中也称为*参与任务*）。

开始之前

任务实例可以处于“就绪”或“已声明”状态。可以将其升级。调用者必须是该任务实例的所有者、发起者或管理员。

关于此任务

可以将处于“正在运行”状态的任务实例暂挂。例如，通过执行此操作，可以收集完成该任务所需的信息。获得该信息后，就可以恢复该任务实例。

过程

1. 获取已登录用户声明的任务列表。

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

   此操作返回一个查询结果集，该结果集包含已登录用户声明的任务列表。

2. 暂挂任务实例。

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

   此操作暂挂指定的任务实例。该任务实例将进入“已暂挂”状态。

3. 恢复该流程实例。

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

   此操作将使该任务实例进入它被暂挂前所处的状态。

分析任务结果

待执行任务（在 API 中也称为*参与*任务）或协作任务（在 API 中也称为*人员任务*）以异步方式运行。如果任务启动时指定了应答处理程序，该任务完成时就会自动返回输出消息。如果未指定应答处理程序，就必须显式地检索该消息。

关于此任务

仅当用于派生任务实例的任务模板未指定自动删除所派生的任务实例时，任务结果才会存储在数据库中。

过程

分析任务结果。

本示例说明如何检查成功完成的任务的订单号。

```
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");
  }
}
```

终止任务实例

有时，具有管理员权限的用户需要终止已知处于不可恢复状态的任务实例。因为将会立即终止任务实例，所以您应该仅在意外情况下才终止任务实例。

过程

1. 检索要终止的任务实例。

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

2. 终止该任务实例。

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

   该任务实例将立即终止，而不会等待任何未完成的任务完成。

删除任务实例

仅当在用于派生任务实例的相关任务模板中指定应该自动删除任务实例时，才会在任务实例完成时自动删除它们。此示例说明如何删除所有已完成并且未被自动删除的任务实例。

过程

1. 列示已完成的任务实例。

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

   此操作将返回一个查询结果集，该结果集列示了已完成的任务实例。

2. 删除已完成的任务实例。

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

释放已声明的任务

当潜在所有者声明某个任务时，此人员将负责完成该任务。但是，有时必须释放已声明的任务，以便另一个潜在所有者可以声明该任务。

关于此任务

有时，具有管理员权限的用户必须释放已声明的任务。例如，当必须完成某个任务，但该任务的所有者不存在时，就可能会发生这种情况。已声明的任务也可以由该任务的所有者释放。

过程

1. 列示特定人员（例如 Smith）拥有的已声明任务。

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

   此操作返回一个查询结果集，该结果集列示了由指定人员（Smith）声明的任务。

2. 释放已声明的任务。

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

   此操作使该任务回到“就绪”状态，以便可以被其他某个潜在所有者声明。将保留由原始所有者设置的任何输出或故障数据。

管理工作项

在活动实例或任务实例的生存期内，与该对象相关联的人员集合会有所变化，例如，变化原因可能包括人员度假、聘用了新人员或者需要以另一方式分配工作负载。为了适应这些变化，您可以开发应用程序来创建、删除或传递工作项。

关于此任务

工作项表示由于特定原因而将一个对象分配给一个用户或一组用户。该对象通常是人员任务活动实例、流程实例或任务实例。原因是从用户对对象具有的角色派生的。由于一个用户可以具有不同的角色与对象相关联，并且将为这些角色中的每一个创建工作项，所以一个对象可以有多个工作项。例如，待执行任务实例可以同时具有管理员、读者、编辑和所有者工作项。

管理工作项时可以执行的操作取决于用户的角色，例如，管理员能够创建、删除和传递工作项，但任务所有者只能传递工作项。

- 创建工作项。

```
// query the task instance for which an additional
// administrator is to be specified
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // create the work item
  task.createWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_ADMINISTRATOR,"Smith");
}
```

此操作为具有管理员角色的用户 Smith 创建工作项。

- 删除工作项。

```
// query the task instance for which a work item is to be deleted
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // delete the work item
  task.deleteWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_READER,"Smith");
}
```

此操作为具有阅读者角色的用户 Smith 删除工作项。

- 传递工作项。

```
// query the task that is to be rescheduled
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.NAME='CustomerOrder' AND
                 TASK.STATE=TASK.STATE.STATE_READY AND
                 WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER AND
                 WORK_ITEM.OWNER_ID='Miller'",
                 (String)null, (Integer)null, (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // transfer the work item from user Miller to user Smith
  // so that Smith can work on the task
  task.transferWorkItem((TKIID)(result.getOID(1)),
                           WorkItem.REASON_POTENTIAL_OWNER,"Miller","Smith");
}
```

此操作将工作项传递给用户 Smith 以使其能够处理该工作项。

在运行时期间创建任务模板和任务实例

您通常使用诸如 WebSphere Integration Developer 之类的建模工具来构建任务模板。然后，将任务模板安装到 WebSphere Process Server 中并根据这些模板来创建实例；例如，使用业务流程编排器资源管理器来创建实例。但是，也可以在运行时期间创建人员任务实例、参与任务实例或模板。

关于此任务

例如，在下列情况下，您可能需要这样做：部署应用程序时没有任务定义可用；工作流程中包含的任务不是已知的；要让某个任务处理一组人员之间的某些临时协作。

您可以通过创建 com.ibm.task.api.TaskModel 类的实例和使用它们来创建可重新使用的任务模板或直接创建运行一次的任务实例，从而对特殊待执行或协作任务建模。com.ibm.task.api.ClientTaskFactory 工厂类中提供了一组工厂方法，以创建 TaskModel 类的实例。在运行时期间为人员任务建模将基于 Eclipse 建模框架（EMF）。

过程

1. 使用 createResourceSet 工厂方法来创建 org.eclipse.emf.ecore.resource.ResourceSet。
2. 可选： 如果想要使用复杂的消息类型，那么可以使用您通过工厂方法 getXSDFactory() 获得的 org.eclipse.xsd.XSDFactory 来定义它们，或使用 loadXSDSchema 工厂方法直接导入现有 XML 模式。

   要使复杂类型可用于 WebSphere Process Server，请将它们作为企业应用程序的一部分部署。
3. 创建或导入类型为 javax.wsdl.Definition 的 Web Service 定义语言（WSDL）定义。

   您可以使用 createWSDLDefinition 方法来创建新的 WSDL 定义。然后，可以为它添加端口类型和操作。此外，还可以使用 loadWSDLDefinition 工厂方法来导入现有的 WSDL 定义。
4. 使用 createTTask 工厂方法来创建任务定义。

   如果想要添加或操作更多复杂的任务元素，那么可以使用 com.ibm.wbit.tel.TaskFactory 类，您可以使用 getTaskFactory 工厂方法来检索该类。
5. 使用 createTaskModel 工厂方法来创建任务模型，然后将在步骤 1 创建的且聚集您同时创建的其他所有工件的资源束传递给该任务模型。
6. 可选： 使用 TaskModel validate 方法来验证模型。

结果

使用其中一个人员任务管理器 EJB API create 方法，这些方法具有 **TaskModel** 参数，用于创建可重复使用的任务模板或运行一次的任务实例。

创建使用简单 Java 类型的运行时任务:

本示例创建一个运行时任务，该任务的接口只使用简单 Java 类型（例如 String 对象）。

关于此任务

本示例仅在调用企业应用程序已装入资源的情况下运行。

过程

1. 访问 ClientTaskFactory 并创建资源集以包含新任务模型的定义。

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. 创建 WSDL 定义并添加操作描述。

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

// create a port type
PortType portType = factory.createPortType( definition, "doItPT" );

// create an operation; the input and output messages are of type String:
```

```
// a fault message is not specified
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. 创建新人员任务的 EMF 模型。

   如果创建任务实例，那么不需要“生效日”（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 );
```

   此步骤使用缺省值来初始化任务模型的属性。

4. 修改人员任务模型的属性。

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 创建包含所有资源定义的任务模型。

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. 验证任务模型并更正找到的验证问题。

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. 创建运行时任务实例或模板。

   使用 HumanTaskManagerService 接口来创建任务实例或任务模板。由于此应用程序仅使用简单 Java 类型，所以不需要指定应用程序名。

   - 以下代码段创建任务实例:

     ```
     atask.createTask( taskModel, (String)null, "HTM" );
     ```

   - 以下代码段创建任务模板:

     ```
     task.createTaskTemplate( taskModel, (String)null );
     ```

结果

如果创建了运行时任务实例，那么现在可以将它启动。如果创建了运行时任务模板，那么现在可以根据该模板创建任务实例了。

创建使用复杂类型的运行时任务:

本示例创建一个运行时任务，该任务的接口使用复杂类型。复杂类型是已定义的类型，即，在客户机的本地文件系统中，有一些 XSD 文件包含复杂类型的描述。

关于此任务

本示例仅在调用企业应用程序已装入资源的情况下运行。

过程

1. 访问 ClientTaskFactory 并创建资源集以包含新任务模型的定义。

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. 将复杂类型的 XSD 定义添加到该资源集中，以便定义操作时能够使用这些定义。

   文件位置相对于代码的执行位置。

```
factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
```

3. 创建 WSDL 定义并添加操作描述。

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

// create a port type
PortType portType = factory.createPortType( definition, "doItPT" );

// create an operation; the input message is an InputBO and
// the output message an OutputBO;
// a fault message is not specified
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://Input", "InputBO" ),
          new QName( "http://Output", "OutputBO" ),
          (Map)null );
```

4. 创建新人员任务的 EMF 模型。

   如果创建任务实例，那么不需要“生效日”（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 );
```

   此步骤使用缺省值来初始化任务模型的属性。

5. 修改人员任务模型的属性。

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
```

```
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. 创建包含所有资源定义的任务模型。

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

7. 验证任务模型并更正找到的验证问题。

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

8. 创建运行时任务实例或模板。

   使用 HumanTaskManagerService 接口来创建任务实例或任务模板。您必须提供包含数据类型定义的应用程序的名称，这样才能访问那些数据类型定义。该应用程序还必须包含一个哑任务或哑流程，这样业务流程编排器才能装入该应用程序。

   - 以下代码段创建任务实例:

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - 以下代码段创建任务模板:

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

结果

如果创建了运行时任务实例，那么现在可以将它启动。如果创建了运行时任务模板，那么现在可以根据该模板创建任务实例了。

创建使用现有接口的运行时任务:

本示例创建一个运行时任务。该任务使用已定义的接口，即，在客户机的本地文件系统中，有一个文件包含该接口的描述。

关于此任务

本示例仅在调用企业应用程序已装入资源的情况下运行。

过程

1. 访问 ClientTaskFactory 并创建资源集以包含新任务模型的定义。

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. 访问 WSDL 定义和操作描述。

   接口描述的位置相对于代码的执行位置。

   ```
   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. 创建新人员任务的 EMF 模型。

   如果创建任务实例，那么不需要“生效日”（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 );
```

此步骤使用缺省值来初始化任务模型的属性。

4. 修改人员任务模型的属性。

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 创建包含所有资源定义的任务模型。

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. 验证任务模型并更正找到的验证问题。

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. 创建运行时任务实例或模板。

   使用 HumanTaskManagerService 接口来创建任务实例或任务模板。您必须提供包含数据类型定义的应用程序的名称，这样才能访问那些数据类型定义。该应用程序还必须包含一个哑任务或哑流程，这样业务流程编排器才能装入该应用程序。

   - 以下代码段创建任务实例:

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - 以下代码段创建任务模板:

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

结果

如果创建了运行时任务实例，那么现在可以将它启动。如果创建了运行时任务模板，那么现在可以根据该模板创建任务实例了。

创建使用调用应用程序中的接口的运行时任务:

本示例创建一个运行时任务，该任务使用调用应用程序中的接口。例如，该运行时任务是在业务流程的 Java 片段中创建的，并且使用流程应用程序中的接口。

关于此任务

本示例仅在调用企业应用程序已装入资源的情况下运行。

过程

1. 访问 ClientTaskFactory 并创建资源集以包含新任务模型的定义。

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();

// specify the context class loader so that following resources are found
ResourceSet resourceSet = factory.createResourceSet
             ( Thread.currentThread().getContextClassLoader() );
```

2. 访问 WSDL 定义和操作描述。

   指定外层包 JAR 文件中的路径。

```
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. 创建新人员任务的 EMF 模型。

   如果创建任务实例，那么不需要“生效日”（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 );
```

   此步骤使用缺省值来初始化任务模型的属性。

4. 修改人员任务模型的属性。

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 创建包含所有资源定义的任务模型。

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. 验证任务模型并更正找到的验证问题。

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. 创建运行时任务实例或模板。

   使用 HumanTaskManagerService 接口来创建任务实例或任务模板。您必须提供包含数据类型定义的应用程序的名称，这样才能访问那些数据类型定义。

   - 以下代码段创建任务实例:

```
task.createTask( taskModel, "WorkflowApplication", "HTM" );
```

- 以下代码段创建任务模板:

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

结果

如果创建了运行时任务实例，那么现在可以将它启动。如果创建了运行时任务模板，那么现在可以根据该模板创建任务实例了。

HumanTaskManagerService 接口

HumanTaskManagerService 接口公布与任务相关的函数，这些函数可以由本地或远程客户机调用。

可以调用的方法取决于任务状态，并取决于包含该方法的应用程序的用户所具有的权限。下面列示了任务对象的主要处理方法。有关这些方法以及 HumanTaskManagerService 接口中提供的其他方法的更多信息，请参阅 com.ibm.task.api 包中的 Javadoc。

任务模板

下列方法可用于处理任务模板。

表 29. 任务模板的 API 方法

方法	描述
getTaskTemplate	检索指定的任务模板。
createAndCallTask	根据指定的任务模板来创建并运行任务实例，然后以同步方式等待结果。
createAndStartTask	根据指定的任务模板来创建并启动任务实例。
createTask	根据指定的任务模板来创建任务实例。
createInputMessage	为指定的任务模板创建输入消息。例如，创建可用来启动任务的消息。
queryTaskTemplates	检索数据库中存储的任务模板。

任务实例

下列方法可用于处理任务实例。

表 30. 任务实例的 API 方法

方法	描述
getTask	检索任务实例；该任务实例可以处于任何状态。
callTask	以同步方式启动调用任务。
startTask	启动已创建的任务。
suspend	暂挂协作或待执行任务。
resume	恢复协作或待执行任务。
terminate	终止指定的任务实例。如果调用任务已终止，那么此操作不会影响所调用的服务。
delete	删除指定的任务实例。
claim	声明任务，以便进行处理。
update	更新任务实例。

表 30. 任务实例的 API 方法 (续)

方法	描述
complete	完成任务实例。
cancelClaim	释放已声明的任务实例，以使其可供另一潜在所有者处理。
createWorkItem	为任务实例创建工作项。
transferWorkItem	将工作项传递给指定的所有者。
deleteWorkItem	删除工作项。

升级

下列方法可用于处理升级。

表 31. 用于处理升级的 API 方法

方法	描述
getEscalation	检索指定的升级实例。

定制属性

任务、任务模板和升级都可以具有定制属性。该接口提供了用于检索和设置定制属性值的 get 和 set 方法。您还可以使指定的属性与任务实例相关联，并可以从这些实例中检索指定的属性。定制属性名和属性值的类型必须是 java.lang.String。下列方法对任务、任务模板和升级有效。

表 32. 变量和定制属性的 API 方法

方法	描述
getCustomProperty	从指定的任务实例中检索指定的定制属性。
getCustomProperties	从指定的任务实例中检索定制属性。
getCustomPropertyNames	为任务实例检索定制属性的名称。
setCustomProperty	为指定的任务实例存储特定于定制属性的值。

允许对任务执行的操作:

可以对任务执行的操作取决于该任务是待执行任务、协作任务、调用任务还是管理任务。

不能对所有类型的任务使用由 HumanTaskManager 接口提供的全部操作。下表描述了可以对每一类任务执行的操作。

	任务类型			
操作	待执行任务	协作任务	调用任务	管理任务
callTask			X	
cancelClaim	X	X[1]		
claim	X	X[1]		
complete	X	X[1]		X
completeWithFollowOnTask[4]	X	X[1]		

操作	任务类型			
	待执行任务	协作任务	调用任务	管理任务
completeWithFollowOnTask[5]		X[3]	X[3]	
createFaultMessage	X	X	X	X
createInputMessage	X	X	X	X
createOutputMessage	X	X	X	X
createWorkItem	X	X[1]	X	X
delete	X[1]	X[1]	X	X[1]
deleteWorkItem	X	X[1]	X	X
getCustomProperty	X	X[1]	X	X
getDocumentation	X	X[1]	X	X
getFaultNames	X	X[1]		
getFaultMessage	X	X[1]	X	
getInputMessage	X	X[1]	X	
getOutputMessage	X	X[1]	X	
getUsersInRole	X	X[1]	X	X
getTask	X	X[1]	X	X
getUISettings	X	X[1]	X	X
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

注:

1. 仅适用于独立任务、特别任务和任务模板
2. 仅适用于独立任务、业务流程中的内联任务和特别任务
3. 仅适用于独立任务和特别任务
4. 可以具有后续任务的任务类型
5. 可以用作后续任务的任务类型
6. 可以具有子任务的任务类型
7. 可以用作子任务的任务类型

为业务流程和人员任务开发应用程序

大多数业务流程方案均涉及到人员。例如，业务流程在启动或管理流程时或执行人员任务活动时均需要人员交互。要支持这些方案，您需要使用业务流程管理器 API 和人员任务管理器 API。

关于此任务

要将人员卷入业务流程方案中，您可以在业务流程中包括下列任务种类:

- 内联调用程序（在 API 中也称为*发端任务*）。

  您可以为每个 receive 活动、pick 活动的每个 onMessage 元素和每个处理程序的每个 onEvent 元素提供调用任务。然后，此任务会控制谁有权启动流程或与正在运行的流程实例通信。
- 管理任务。

  您可以提供管理任务来指定谁有权管理流程或对失败的流程活动执行管理操作。
- 待执行任务（在 API 中也称为*参与任务*）。

  待执行任务实现人员任务活动。此类活动允许您在流程中调用人员。

业务流程中的人员任务活动表示人员在业务流程方案中执行的待执行任务。您可以使用业务流程管理器 API 和人员任务管理器 API 来实现这些方案:

- 业务流程是属于流程的所有活动的容器，这些活动包括由待执行任务表示的人员任务活动。当创建流程实例时，将分配唯一的对象标识（PIID）。
- 在执行流程实例期间激活人员任务活动时，将创建活动实例，它由其唯一的对象标识（AIID）标识。同时也创建由对象标识（TKIID）标识的内联待执行任务实例. 人员任务活动与任务实例的关系可通过使用对象标识来实现:
  - 活动实例的待执行任务标识已设置为关联的待执行任务的 TKIID。
  - 任务实例的包含上下文标识被设置为包含关联活动实例的流程实例的 PIID。
  - 任务实例的父上下文标识已设置为关联的活动实例的 AIID。
- 所有内联待执行任务实例的使用寿命由流程实例管理。删除流程实例时，任务实例也会被删除。换而言之，将自动删除将包含上下文标识设置为流程实例的 PIID 的所有任务。

确定可启动的流程模板或活动

可以通过调用业务流程管理 API 的 call、initiate 或 sendMessage 方法来启动业务流程。如果流程只有一个起始活动，那么您可以使用需要流程模板名称作为参数的方法特征符。如果流程具有多个起始活动，那么您必须明确标识起始活动。

关于此任务

当对业务流程建模时，建模器可以决定只有一部分用户可以根据流程模板创建流程实例。这是通过将内联调用任务与流程的起始活动相关联以及对该任务指定授权限制来实现的。只有任务的潜在启动者或管理员才能创建该任务实例，流程模板的实例同样也是如此。

如果内联调用任务未与起始活动相关联，或者没有为任务指定授权限制，那么每个人都可以使用起始活动创建流程实例。

流程可以具有多个起始活动，对于起始者或管理员，每个活动具有不同的人员查询。这意味着用户有权使用活动 A（而不是使用活动 B）来启动流程。

过程

1. 使用业务流程管理器 API 来创建处于已启动状态的当前版本的流程模板的列表。

   提示: queryProcessTemplates 方法仅排除那些作为尚未启动的应用程序的一部分的流程模板。因此，如果您使用此方法而不过滤结果，那么该方法会返回所有版本的流程模板，而不管它们处于何种状态。

   ```
   // current timestamp in UTC format, converted to yyyy-mm-ddThh:mm:ss
   String now = (new UTCDate()).toXsdString();
   String whereClause = "PROCESS_TEMPLATE.STATE =
                         PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                         PROCESS_TEMPLATE.VALID_FROM =
                         (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                          WHERE NAME=PROCESS_TEMPLATE.NAME AND
                          VALID_FROM <= TS('" + now + "'))";

   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
                                           ( whereClause,
                                             "PROCESS_TEMPLATE.NAME",
                                             (Integer)null, (TimeZone)null);
   ```

   结果将按流程模板名称排序。

2. 创建用户对其拥有权限的流程模板和起始活动的列表。

   流程模板列表包含具有单一起始活动的那些流程模板。这些活动未被保护，或者允许已登录用户启动它们。此外，您可能想要收集可由其中至少一个起始活动启动的流程模板。

   提示: 流程管理员也可以启动流程实例。要获取完整的模板列表，还需要读取与流程模板关联的管理任务模板，并且检查已登录用户是否为管理员。

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

3. 为每个流程模板确定起始活动。

   ```
   for( int i=0; i<processTemplates.length; i++ )
   {
     ProcessTemplateData template = processTemplates[i];
     ActivityServiceTemplateData[] startActivities =
                    process.getStartActivities(template.getID());
   ```

4. 对于每个起始活动，检查关联的内联调用任务模板的标识。

   ```
   for( int j=0; j<startActivities.length; j++ )
   {
     ActivityServiceTemplateData activity = startActivities[j];
     TKTID tktid = activity.getTaskTemplateID();
   ```

   a. 如果调用任务模板不存在，那么流程模板不受此起始活动保护。

      在此情况下，每个人可以使用此起始活动来创建流程实例。

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

b. 如果调用任务模板存在，那么使用人员任务管理器 API 来检查已登录用户的权限。

在本示例中，已登录用户是 Smith。已登录用户必须是调用任务的潜在启动者或管理员。

```
if ( tktid != null )
    {
      isAuthorized =
        task.isUserInRole
            (tkid, "Smith", WorkItem.REASON_POTENTIAL_STARTER) ||
        task.isUserInRole(tktid, "Smith", WorkItem.REASON_ADMINISTRATOR);

      if ( isAuthorized )
      {
        authorizedActivityServiceTemplates.add(activity);
      }
    }
```

如果用户具有指定的角色，或角色的人员分配条件未指定，那么 isUserInRole 方法会返回值 `true`。

5. 检查是否仅使用流程模板名称就可以启动流程。

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

6. 结束循环。

```
  } // end of loop for each activity service template
}   // end of loop for each process template
```

处理包含人员任务的单人工作流程

某些工作流仅由一个人执行，例如，从在线书店订购书籍。此示例说明如何将订购书籍的操作顺序作为一系列人员任务活动（待执行任务）实现。业务流程管理器和人员任务管理器 API 均用于处理工作流程。

关于此任务

买方于在线书店中完成一系列操作来订购书籍。此系列的操作可以作为一系列人员任务活动（待执行任务）来实现。如果买方决定订购多本书，那么这相当于声明下一个人员任务活动。有关任务顺序的信息由业务流程管理器维护，而任务本身由人员任务管理器维护。

将此示例与仅使用业务流程管理器 API 的示例相比较。

过程

1. 使用业务流程管理器 API 来获得要处理的流程实例。

在本示例中，CustomerOrder 流程的实例。

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

2. 使用人员任务管理器 API 来查询作为指定流程实例一部分的已就绪的待执行任务（要参与的种类）。

使用任务的包含上下文标识来指定包含流程实例。对于单人工作流程，查询将返回与人员任务活动顺序中第一个人员任务活动相关联的待执行任务。

```
//
 // Query the list of to-do tasks that can be claimed by the logged-on user
 // for the specified process instance
//
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.CONTAINMENT_CTX_ID = ID('" + piid + "') AND
                 TASK.STATE = TASK.STATE.STATE_READY AND
                 TASK.KIND = TASK.KIND.KIND_PARTICIPATING AND
                 WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                (String)null, (Integer)null, (TimeZone)null);
```

3. 声明已返回的待执行任务。

```
if (result.size() > 0)
{
 result.first();
 TKIID tkiid = (TKIID) result.getOID(1);
ClientObjectWrapper input = task.claim(tkiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
    taskInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

声明任务时，将返回该任务的输入消息。

4. 确定与待执行任务相关联的人员任务活动。

您可以使用下列其中一个方法来使活动与其任务相关联。

- task.getActivityID 方法:

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

- 作为任务对象一部分的父上下文标识:

```
AIID aiid = null;
Task taskInstance = task.getTask(tkiid);
OID oid = taskInstance.getParentContextID();
if ( oid != null and oid instanceof AIID )
{
  aiid = (AIID)oid;
}
```

5. 当处理完任务后，请使用业务流程管理器 API 来完成任务及其相关联的人员任务活动，然后声明流程实例中的下一个人员任务活动。

要完成人员任务活动，将传递输出消息。创建输出消息时，必须指定消息类型名，以便包含消息定义。

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}
```

```
//complete the human task activity and its associated to-do task,
// and claim the next human task activity
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

此操作将设置包含订单号的输出消息，并且声明该顺序中的下一个人员任务活动。如果已为后续活动设置 AutoClaim，且存在多个可以使用的路径，那么会声明所有后续活动，并且返回随机活动以作为下一个活动。如果没有其他后续活动可分配给此用户，那么将返回 Null。

如果该流程包含可使用的并行路径，并且这些路径包含人员任务活动，已登录用户是多个此类活动的潜在所有者，那么将会自动声明随机活动并将其作为下一个活动返回。

6. 处理下一个人员任务活动。

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

7. 继续执行步骤 5 以完成人员任务活动并检索下一个人员任务活动。

相关任务

第 65 页的『处理单人工作流』

某些工作流仅由一个人执行，例如，从在线书店订购书籍。此类工作流没有并行路径。completeAndClaimSuccessor API 支持此类工作流的处理。

处理异常和故障

BPEL 流程可能会在流程的不同环节中遇到故障。

关于此任务

业务流程执行语言（BPEL）故障源自:

- Web Service 调用（Web 服务描述语言（WSDL）故障）
- 抛出活动
- 业务流程编排器确认的 BPEL 一般故障

可使用下列机制来处理这些故障。使用下列其中一种机制来处理流程实例生成的故障:

- 将控制权传递至相应的故障处理程序
- 对该流程中的先前工作进行补偿
- 停止该流程并允许某人纠正该情况（强制重试或强制完成）

BPEL 流程还可以将故障返回至该流程提供的操作的调用者。您可以在流程中将故障作为具有故障名称和故障数据的应答活动建模。这些故障作为已校验的异常返回至 API 调用者。

如果 BPEL 流程未处理 BPEL 故障或如果发生 API 异常，那么会将运行时异常返回至 API 调用者。例如，当用于创建实例的流程模型不存在时抛出 API 异常。

下列任务对故障和异常的处理方式作了描述。

处理 API 异常

关于此任务

如果 BusinessFlowManagerService 接口或 HumanTaskManagerService 接口中的方法未成功完成，就会抛出指示了错误原因的异常。您可以明确地处理此异常以便向调用者提供指导。

但是，通常的做法是仅明确地处理一小部分异常并提供其他潜在异常的一般指导。所有特定异常都继承通用的 ProcessException 或 TaskException。*最好的做法*是使用终态的 `catch(ProcessException)` 或 `catch(TaskException)` 语句来捕获通用异常。由于此语句会考虑所有其他可能发生的异常，所以它有助于确保应用程序向上兼容。

检查为活动设置了哪个故障

过程

1. 列示处于“已失败”或“已停止”状态的任务活动。

```
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);
```

   此操作返回一个查询结果集，该结果集包含已失败或已停止的活动。

2. 读取故障的名称。

```
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();
  }
}
```

   这段代码返回故障名。您也可以对已停止的活动的未处理异常进行分析来代替检索故障名。

检查已停止的调用活动所发生的故障

关于此任务

如果某个活动导致故障发生，那么可通过故障类型确定可用来修复该活动的操作。

过程

1. 列示处于“已停止”状态的人员任务活动。

```
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);
```

此操作返回一个查询结果集，该结果集包含已停止的调用活动。

2. 读取故障的名称。

```
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();
  }
}
```

开发 Web Service API 客户机应用程序

您可以开发通过 Web Service API 访问业务流程应用程序和人员任务应用程序的客户机应用程序。

关于此任务

可以在任何 Web Service 客户机环境中开发客户机应用程序，这些环境包括 Java Web Service 和 Microsoft .NET。

简介: Web Service

Web Service 是基于 Web 的企业应用程序，它们使用基于 XML 的开放式标准和传输协议与客户机应用程序交换数据。Web Service 允许您使用语言和环境均中性的编程模型。

Web Service 使用下列核心技术:

- XML（可扩展标记语言）。XML 解决了数据独立性的问题。您可使用它来描述数据，并且还可将该数据映射到应用程序或编程语言中，以及从中映射出来
- WSDL（Web 服务描述语言）。您使用此基于 XML 的语言来创建底层应用程序的描述。正是此描述通过在底层应用程序和其他支持 Web 的应用程序之间充当接口，将应用程序转换为 Web Service。
- SOAP（简单对象访问协议）。SOAP 是 Web 的核心通信协议，大多数 Web Service 都使用此协议进行相互通信。

Web Service 组件和控制顺序

许多客户端组件和服务器端组件都参与控制序列来表示 Web Service 请求和响应。

典型的控制顺序如下所示。

1. 在客户端上:
   a. 用户提供的客户机应用程序发出 Web Service 请求。

b. 代理客户机（也是由用户提供，但可以使用客户端实用程序自动生成）将服务请求打包在 SOAP 请求包络中。

c. 客户端开发基础结构将该请求转发至定义为 Web Service 的端点的 URL。

2. 网络使用 HTTP 或 HTTPS 来将该请求传送至 Web Service 端点。

3. 在服务器端上:

a. 普通的 Web Service API 接收该请求，并且对请求进行解码。

b. 可以通过普通的业务流程管理器或人员任务管理器组件直接处理该请求，也可以将该请求转发至指定的业务流程或人员任务。

c. 返回的数据已打包在 SOAP 响应包络中。

4. 网络使用 HTTP 或 HTTPS 来将该响应传送至客户端环境。

5. 返回到客户端上:

a. 客户端开发基础结构打开 SOAP 响应包络。

b. 代理客户机从 SOAP 响应中抽取数据，并将其传递至客户机应用程序。

c. 客户机应用程序在必要时可处理返回的数据。

Web Service API 的概述

Web Service API 允许您开发客户机应用程序，这些应用程序使用 Web Service 来访问在业务流程编排器环境中运行的业务流程和人员任务。

业务流程编排器 Web Service API 提供两个单独的 Web Service 接口（WSDL 端口类型）:

- 业务流程管理器 API。允许客户机应用程序与微流动和长期运行的流程进行交互，例如:
  - 创建流程模板和流程实例
  - 声明现有的流程
  - 按标识来查询流程

  请参阅第 55 页的『为业务流程开发应用程序』以获得可能操作的完整列表。
- 人员任务管理器 API。允许客户机应用程序执行下列操作:
  - 创建和启动任务
  - 声明现有的任务
  - 完成任务
  - 按标识来查询任务
  - 查询任务集合。

  请参阅第 74 页的『为人员任务开发应用程序』以获得可能操作的完整列表。

客户机应用程序可以使用一个或两个上述 Web Service 接口。

示例

以下大致概述了访问人员任务管理器 Web Service API 来处理参与人员任务的客户机应用程序:

1. 客户机应用程序向 WebSphere Process Server 发出 query Web Service 调用，请求用户要处理的参与任务的列表。

2. 在 SOAP/HTTP 响应包络中返回参与任务的列表。
3. 然后，客户机应用程序发出 claim Web Service 调用，以声明其中一个参与任务。
4. WebSphere Process Server 返回该任务的输入消息。
5. 客户机应用程序发出 complete Web Service 调用，以完成带输出消息或故障消息的任务。

对业务流程和人员任务的要求

使用 WebSphere Integration Developer 开发以在业务流程编排器上运行的业务流程和人员任务必须符合特定的规则，以便可以通过 Web Service API 进行访问。

这些要求是:

1. 必须使用在 Java API for XML-based RPC（JAX-RPC 1.1）规范中定义的“document/literal 包装”样式来定义业务流程和人员任务的接口。这是使用 WID 开发的所有业务流程和人员任务的缺省样式。
2. 业务流程和人员任务所显示的 Web Service 操作的故障消息必须包含使用 XML 模式元素定义的单个 WSDL 消息部分。例如:

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

   相关信息

   Java API for XML based RPC（JAX-RPC）下载页面

   我应该使用哪种 WSDL 样式？

开发客户机应用程序

客户机应用程序开发流程由许多步骤组成。

过程

1. 确定客户机应用程序需要使用的 Web Service API: 业务流程管理器 API 和/或人员任务管理器 API。
2. 从 WebSphere Process Server 环境中导出必需的文件。 此外，还可以从 WebSphere Process Server 客户机 CD 中复制文件。
3. 在所选客户机应用程序开发环境中，使用导出的工件来生成*代理客户机*。
4. 可选: 生成*辅助控件类*。 如果客户机应用程序直接与 WebSphere 服务器上的具体流程或任务进行交互，那么需要辅助控件类。但是，如果客户机应用程序仅准备执行诸如发出查询之类的普通任务，那么这些辅助控件类不是必需的。
5. 为客户机应用程序开发代码。
6. 将所有必需的安全性机制添加至客户机应用程序。

复制工件

必须从 WebSphere 环境中复制许多工件，以帮助创建客户机应用程序。

有两种方法来获得这些工件:

- 从 WebSphere Process Server 环境中发布和导出它们。
- 从 WebSphere Process Server 客户机 CD 中复制文件。

从服务器环境中发布和导出工件

在可以开发客户机应用程序来访问 Web Service API 之前，您必须从 WebSphere 服务器环境中发布和导出许多工件。

关于此任务

要导出的工件是:

- 描述组成 Web Service API 的端口类型和操作的 Web Service 定义语言（WSDL）文件。
- 包含 WSDL 文件中服务和方法所引用的数据类型定义的 XML 模式定义（XSD）文件。
- 其他描述业务对象的 WSDL 文件和 XSD 文件。业务对象描述了在 WebSphere 服务器上运行的具体业务流程或人员任务。如果客户机应用程序需要通过 Web Service API 直接与具体业务流程或人员任务进行交互，那么仅需要这些其他的文件。如果客户机应用程序仅准备执行诸如发出查询之类的普通任务，那么这些其他的文件不是必需的。

在发布这些工件之后，您需要将它们复制至客户机编程环境，工件在这个环境中用于生成代理客户机和辅助控件类。

指定 Web Service 端点地址:

该 Web Service 端点地址是客户机应用程序为访问 Web Service API 而必须指定的 URL。该端点地址写入 WSDL 文件中，您可以将该文件导出以便为客户机应用程序生成代理客户机。

关于此任务

要使用的 Web Service 端点地址取决于 WebSphere 服务器配置:

- 方案 1. 一台 WebSphere 服务器。要指定的 WebSphere 端点地址是该服务器的主机名和端口号，例如 **host1:9080**。
- 方案 2. 一个由若干服务器组成的 WebSphere 集群。要指定的 WebSphere 端点地址是主管 Web Service API 的服务器的主机名和端口号，例如 **host2:9081**。
- 方案 3. 将 Web 服务器用作前端。要指定的 WebSphere 端点地址是该 Web 服务器的主机名和端口号，例如 **host:80**。

缺省情况下，该 Web Service 端点地址采用以下格式: *protocol*://*host*:*port*/*context_root*/*fixed_path*。其中:

- *protocol*。在客户机应用程序和 WebSphere 服务器之间使用的通信协议。缺省协议是 HTTP。但您可以选择使用更为安全的 HTTPS（HTTP over SSL）协议。我们建议您使用 HTTPS。
- *host:port*。用于访问主管 Web Service API 的机器的主机名和端口号。这些值因 WebSphere 服务器配置的不同而异；例如，您的客户机应用程序是直接访问该应用程序还是通过 Web 服务器前端进行访问。
- *context_root*。可以为上下文根自由选择任何值。但是，所选的值在每个 WebSphere 单元中必须是唯一的。该缺省值使用“node_server/cluster”后缀来消除发生命名冲突的风险。

- *fixed_path* 是 /sca/com/ibm/bpe/api/BFMWS（适用于业务流程管理器 API）或 /sca/com/ibm/task/api/HTMWS（适用于人员任务管理器 API），并且不能被修改。

当配置业务流程容器或人员任务容器时，一开始就指定该 Web Service 端点地址:

过程

1. 使用具有管理员权限的用户标识登录管理控制台。
2. 选择**应用程序 → SCA 模块**。

   注: 也可以选择**应用程序 → 企业应用程序**，以显示所有可用企业应用程序的列表。
3. 从 SCA 模块或应用程序的列表中选择 **BPEContainer**（适用于业务流程容器）或 **TaskContainer**（适用于人员任务容器）。
4. 从**其他属性**的列表中选择**提供 HTTP 端点 URL 信息**。
5. 从该列表中选择其中一个缺省前缀，或输入定制前缀。 如果客户机应用程序要直接连接至主管 Web Service API 的应用程序服务器，那么使用缺省前缀列表中的前缀。否则，请指定定制前缀。
6. 单击**应用**以将所选前缀复制至 SCA 模块。
7. 单击**确定**。 将该 URL 信息保存至工作空间。

结果

可以在管理控制台中查看当前值（例如，对于业务流程容器: **企业应用程序 → BPEContainer → 查看部署描述符**）。

在导出的 WSDL 文件中，soap:address 元素的 location 属性包含指定的 Web Service 端点地址。例如:

```
<wsdl:service name="BFMWSService">
  <wsdl:port name="BFMWSPort" binding="this:BFMWSBinding">
   <soap:address location=
         "https://myserver:9080/WebServicesAPIs/sca/com/ibm/bpe/api/BFMWS"/>
```

相关概念

第 112 页的『添加安全性（Java Web Service）』
您必须通过在客户机应用程序中实现安全性机制来保护 Web Service 通信安全。

相关任务

第 120 页的『添加安全性（.NET）』
您可以通过将安全性机制集成到客户机应用程序中来保护 Web Service 通信安全。

发布 WSDL 文件:

Web Service 定义语言（WSDL）文件包含随 Web Service API 提供的所有操作的详细描述。业务流程管理器和人员任务管理器 Web Service API 具有单独的 WSDL 文件。您必须首先发布这些 WSDL 文件，然后将它们从 WebSphere 环境复制至使用这些文件来生成代理客户机的开发环境。

开始之前

在发布 WSDL 文件之前，确保指定正确的 Web Service 端点地址。这是您的客户机应用程序用于访问 Web Service API 的 URL。

关于此任务

您只需要发布 WSDL 文件一次。

注: 如果具有 WebSphere Process Server 客户机 CD，那么可以将文件从该 CD 中直接复制至客户机编程环境。

发布业务流程 WSDL:

使用管理控制台来发布 WSDL 文件。

过程

1. 使用具有管理员权限的用户标识登录管理控制台。
2. 选择**应用程序 → SCA 模块**

   注: 也可以选择**应用程序 → 企业应用程序**，以显示所有可用企业应用程序的列表。
3. 从 SCA 模块或应用程序的列表中选择 **BPEContainer** 应用程序。
4. 从**其他属性**的列表中选择**发布 WSDL 文件**
5. 单击列表中的 zip 文件。
6. 在出现的文件下载窗口中，单击**保存**。
7. 浏览至本地文件夹，然后单击**保存**。

结果

导出的 zip 文件被命名为 BPEContainer_WSDLFiles.zip。该 zip 文件包含描述 Web Service 的 WSDL 文件以及在 WSDL 文件中引用的任何 XSD 文件。

发布人员任务 WSDL:

使用管理控制台来发布 WSDL 文件。

过程

1. 使用具有管理员权限的用户标识登录管理控制台。
2. 选择**应用程序 → SCA 模块**

   注: 也可以选择**应用程序 → 企业应用程序**，以显示所有可用企业应用程序的列表。
3. 从 SCA 模块或应用程序的列表中选择 **TaskContainer** 应用程序。
4. 从**其他属性**的列表中选择**发布 WSDL 文件**
5. 单击列表中的 zip 文件。
6. 在出现的文件下载窗口中，单击**保存**。
7. 浏览至本地文件夹，然后单击**保存**。

结果

导出的 zip 文件被命名为 TaskContainer_WSDLFiles.zip。该 zip 文件包含描述 Web Service 的 WSDL 文件以及在 WSDL 文件中引用的任何 XSD 文件。

导出业务对象:

业务流程和人员任务具有严格定义的接口，可允许将它们作为 Web Service 从外部进行访问。如果这些接口引用业务对象，那么需要将接口定义和业务对象导出至客户机编程环境。

关于此任务

必须对客户机应用程序需要与之交互的每个业务对象重复此过程。

在 WebSphere Process Server 中，业务对象定义了与业务流程或人员任务交互的请求、响应和故障消息的格式。这些消息还可以包含复杂数据类型的定义。

例如，要创建和启动人员任务，必须将下列信息项传递至任务接口:

- 任务模板名称
- 任务模板名称空间
- 包含已格式化业务数据的输入消息
- 用于返回响应消息的响应包装器
- 用于返回故障和异常的故障消息

这些项封装在单个业务对象中。Web Service 接口的所有操作均建模为“document/literal 包装”操作。这些操作的输入和输出参数均封装在包装器文档中。其他业务对象定义相应的响应和故障消息格式。

为了通过 Web Service 创建和启动业务流程或人员任务，必须使这些包装器对象可供客户端的客户机应用程序使用。

通过将业务对象从 WebSphere 环境中作为 Web Service 定义语言（WSDL）和 XML 模式定义（XSD）文件导出，将该数据类型定义导入客户机编程环境，然后将它们转换为辅助控件类以供客户机应用程序使用，就可以实现上述目标。

过程

1. 如果尚未运行 WebSphere Integration Developer 工作空间，那么将它启动。
2. 选择包含要导出的业务对象的库模块。 库模块是经过压缩的文件，它包含必需的业务对象。
3. 导出该库模块。
4. 将导出的文件复制至客户机应用程序开发环境。

示例

假定业务流程显示下列 Web Service 操作:

```
<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>
```

并且将 WSDL 消息定义为:

```
<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>
```

必须在客户机应用程序执行的所有*通用*操作（call 和 sendMessage 等）中使用 <xsd:any> 参数，来将*具体的*客户定义的元素 “types:updateCustomer”、“types:updateCustomerResponse” 和 “types:updateCustomerFault”传递至 Web Service API 以及从 Web Service API 接收这些元素。通过那些使用已导出 XSD 文件所生成的辅助控件类，在客户机应用程序端创建这些客户定义的元素，以及对它们进行序列化和反序列化。

相关任务

第 117 页的『为 BPEL 流程创建辅助控件类（.NET）』

某些 Web Service API 操作要求客户机应用程序使用“document/literal”样式包装的元素。客户机应用程序需要辅助控件类来帮助它们生成必需的包装器元素。

使用客户机 CD 上的文件

作为从 WebSphere 服务器环境中导出工件的另一个方法，您可以从 WebSphere Process Server 客户机 CD 中复制用于生成客户机应用程序所必需的文件。

在此情况下，您必须手动修改业务流程管理器 API 或人员任务管理器 API 的缺省 Web Service 端点地址。

如果客户机应用程序要访问两个 API，那么您必须为这两个 API 编辑缺省端点地址。

从客户机 CD 中复制文件:

WebSphere Process Server 客户机 CD 中提供有访问 Web Service API 所必需的文件。

过程

1. 访问该客户机 CD，然后浏览至 ProcessChoreographer\client 目录。
2. 将必需的文件复制至客户机应用程序开发环境。

对于业务流程管理器 API，请复制:

BFMWS.wsdl
描述业务流程管理器 Web Service API 中可用的 Web Service。此文件包含端点地址。

BFMIF.wsdl
描述业务流程管理器 Web Service API 中每个 Web Service 的参数和数据类型。

BFMIF.xsd
描述在业务流程管理器 Web Service API 中使用的数据类型。

BPCGEN.xsd
包含在业务流程管理器 Web Service API 和人员任务管理器 Web Service API 之间公用的数据类型。

对于人员任务管理器 API，请复制:

HTMWS.wsdl
描述人员任务管理器 Web Service API 中可用的 Web Service。此文件包含端点地址。

HTMIF.wsdl
描述人员任务管理器 Web Service API 中每个 Web Service 的参数和数据类型。

HTMIF.xsd
描述在人员任务管理器 Web Service API 中使用的数据类型。

BPCGEN.xsd
包含在业务流程管理器 Web Service API 和人员任务管理器 Web Service API 之间公用的数据类型。

注: 该 BPCGen.xsd 文件是两个 API 的公共文件。

在复制文件之后，必须将 BFMWS.wsdl 或 HTMWS.wsdl 文件包含的 Web Service API 端点地址手动更改为主管该 Web Service API 的 WebSphere Application Server 的端点地址。

手动更改 Web Service 端点地址:

如果从客户机 CD 中复制文件，那么必须将 WSDL 文件中指定的缺省 Web Service 端点地址更改为主管该 Web Service API 的服务器的端点地址。

关于此任务

在导出 WSDL 文件之前，您可以使用管理控制台来设置 Web Service 端点地址。但是，如果从 WebSphere Process Server 客户机 CD 中复制该 WSDL 文件，那么必须手动修改缺省 Web Service 端点地址。

要使用的 Web Service 端点地址取决于 WebSphere 服务器配置:

- 方案 1. 一台 WebSphere 服务器。要指定的 WebSphere 端点地址是该服务器的主机名和端口号，例如 **host1:9080**。
- 方案 2. 一个由若干服务器组成的 WebSphere 集群。要指定的 WebSphere 端点地址是主管 Web Service API 的服务器的主机名和端口号，例如 **host2:9081**。
- 方案 3. 将 Web 服务器用作前端。要指定的 WebSphere 端点地址是该 Web 服务器的主机名和端口号，例如 **host:80**。

更改业务流程管理器 API 端点:

如果从 WebSphere Process Server 客户机 CD 中复制业务流程管理器 API 文件，那么必须手动编辑缺省的端点地址。

过程

1. 浏览至包含从该客户机 CD 中复制的文件的目录。
2. 在文本编辑器或 XML 编辑器中打开 BFMWS.wsdl 文件。
3. 在接近文件底部的位置找到 `soap:address` 元素。
4. 使用 Web Service API 在其上运行的服务器的 HTTP URL 来修改 `location` 属性的值。 为此，请执行下列操作:

a. （可选）将 http 替换为 https 以使用更为安全的 HTTPS 协议。

b. 将 *localhost* 替换为 Web Service API 服务器端点地址的主机名或 IP 地址。

c. 将 *9080* 替换为应用程序服务器的端口号。

d. 将 *BPEContainer_N1_server1* 替换为运行 Web Service API 的应用程序的上下文根。 缺省上下文根包含:

- *BPEContainer*。应用程序名。
- *N1*。节点名。
- *server1*。服务器名。

e. 切勿修改 URL 的固定部分（/sca/com/ibm/bpe/api/BFMWS）。

例如，如果应用程序在服务器 **s1.n1.ibm.com** 上运行，并且该服务器接受端口 **9080** 上的 SOAP/HTTP 请求，那么将 soap:address 元素修改为:

```
<soap:address location="http://si.n1.ibm.com:9080/
                  BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

相关概念

第 112 页的『添加安全性（Java Web Service）』
您必须通过在客户机应用程序中实现安全性机制来保护 Web Service 通信安全。

相关任务

第 120 页的『添加安全性（.NET）』
您可以通过将安全性机制集成到客户机应用程序中来保护 Web Service 通信安全。

更改人员任务管理器 API 端点:

如果从 WebSphere Process Server 客户机 CD 中复制人员任务管理器 API 文件，那么必须手动编辑缺省的端点地址。

过程

1. 浏览至包含从该客户机 CD 中复制的文件的目录。
2. 在文本编辑器或 XML 编辑器中打开 HTMWS.wsdl 文件。
3. 在接近文件底部的位置找到 soap:address 元素。
4. 使用正确的端点地址来修改 location 属性的值。 为此，请执行下列操作:
   a. （可选）将 http 替换为 https 以使用更为安全的 HTTPS 协议。
   b. 将 *localhost* 替换为 Web Service API 服务器的端点地址的主机名或 IP 地址。
   c. 将 *9080* 替换为应用程序服务器的端口号。
   d. 将 *HTMContainer_N1_server1* 替换为运行 Web Service API 的应用程序的上下文根。 缺省上下文根包含:
      - *HTMContainer*。应用程序名。
      - *N1*。节点名。
      - *server1*。服务器名。
   e. 切勿修改 URL 的固定部分（/sca/com/ibm/task/api/HTMWS）。

   例如，如果应用程序在服务器 **s1.n1.ibm.com** 上运行，并且该服务器接受端口 **9081** 上的 SOAP/HTTPS 请求，那么将 soap:address 元素修改为:

```
<soap:address location="https://si.n1.ibm.com:9081/
                  HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

相关概念

第 112 页的『添加安全性（Java Web Service）』
您必须通过在客户机应用程序中实现安全性机制来保护 Web Service 通信安全。

相关任务

第 120 页的『添加安全性（.NET）』
您可以通过将安全性机制集成到客户机应用程序中来保护 Web Service 通信安全。

在 Java Web Service 环境中开发客户机应用程序

您可以使用与 Java Web Service 兼容的任何基于 Java 的开发环境，以便为 Web Service API 开发客户机应用程序。

生成代理客户机（Java Web Service）

Java Web Service 客户机应用程序使用*代理客户机*与 Web Service API 进行交互。

关于此任务

Java Web Service 的代理客户机包含许多 Java Bean 类，客户机应用程序可以调用这些类来执行 Web Service 请求。该代理客户机将服务参数组装至 SOAP 消息，通过 HTTP 将 SOAP 消息发送至 Web Service，接收来自 Web Service 的响应，以及将返回的数据传递至客户机应用程序。

因此，代理客户机主要是允许客户机应用程序调用 Web Service，就像它是本地函数一样。

注: 您只需要生成代理客户机一次。然后，所有访问相同 Web Service API 的客户机应用程序都可以使用同一个代理客户机。

在 IBM® Web Service 环境中，有两种方法来生成代理客户机:

- 使用 Rational® Application Developer 或 WebSphere Integration Developer 集成开发环境。
- 使用 WSDL2Java 命令行工具。

其他 Java Web Service 开发环境通常包括 WSDL2Java 工具或享有专利权的客户机应用程序生成设施。

使用 Rational Application Developer 来生成代理客户机:

Rational Application Developer 集成开发环境允许您为客户机应用程序生成代理客户机。

开始之前

在生成代理客户机之前，必须事先从 WebSphere 环境（或 WebSphere Process Server 客户机 CD）中导出用于描述业务流程或人员任务 Web Service 接口的 WSDL 文件，然后将它们复制至客户机编程环境。

过程

1. 将适当的 WSDL 文件添加至项目中:
   - 对于业务流程:
     a. 将导出的文件 BPEContainer_*nodename_servername*_WSDLFiles.zip 解压缩到临时目录。

b. 从已解压缩的目录 BPEContainer_*nodename_servername*.ear/b.jar 中导入子目录 META-INF。

- 对于人员任务:
  a. 将导出的文件 TaskContainer_*nodename_servername*_WSDLFiles.zip 解压缩到临时目录。
  b. 从已解压缩的目录 TaskContainer_*nodename_servername*.ear/h.jar 导入子目录 META-INF。

将在项目中创建新目录 wsdl 和子目录结构。

2. 修改 Web Service 向导属性:
   a. 在 Rational Application Developer 中，选择**首选项 → Web Service → 代码生成 → IBM WebSphere 运行时**。
   b. 选择**使用不换行样式从 WSDL 生成 Java** 选项。

   注: 如果无法选择**首选项**菜单中的 **Web Service** 选项，那么必须先启用所需的功能，如下所示: **窗口 → 首选项 → 工作台 → 功能**。依次单击 **Web Service 开发者**和**确定**。然后，重新打开首选项窗口并更改**代码生成**选项。
3. 选择位于新创建的 wsdl 目录中的 BFMWS.WSDL 或 HTMWS.WSDL 文件。
4. 右键单击并选择 **Web Service → 生成客户机**。

   在继续余下步骤之前，请确保服务器已启动。
5. 在 Web Service 窗口上，单击**下一步**以接受所有缺省值。
6. 在 Web Service 选择窗口上，单击**下一步**以接受所有缺省值。
7. 在客户机环境配置窗口上:
   a. 单击**编辑**并将 Web Service 运行时选项更改为 IBM WebSphere
   b. 将 J2EE 版本选项更改为 1.4。
   c. 单击**确定**。
   d. 单击**下一步**。
8. 此步骤仅在您需要生成包含业务流程和人员任务 Web Service API 的 Web Service 客户机时才有必要，原因是两个 WSDL 文件中存在重复的方法。
   a. 在“Web Service 代理”窗口中，选择定义定制映射以供名称空间打包，然后单击**确定**。
   b. 在“要打包映射的 Web Service 客户机名称空间”窗口中，添加下列名称空间和软件包:

      对于 BFMWS.wsdl:

名称空间	软件包
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

对于 HTMWS.wsdl:

名称空间	软件包
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

如果系统要求您确认覆盖，那么单击**全部为是**。

9. 单击**完成**。

结果

将生成由多个代理、定位器和辅助控件 Java 类组成的代理客户机，然后将其添加至项目中。此外，还会更新部署描述符。

使用 WSDL2Java 来生成代理客户机:

WSDL2Java 是可生成代理客户机的命令行工具。代理客户机使编写客户机应用程序更为容易。

开始之前

在生成代理客户机之前，必须事先从 WebSphere 环境（或 WebSphere Process Server 客户机 CD）中导出用于描述业务流程或人员任务 Web Service API 的 WSDL 文件，然后将它们复制至客户机编程环境。

关于此任务

过程

1. 使用 WSDL2Java 工具来生成代理客户机: 请输入:

   wsdl2java *options WSDLfilepath*

   其中:

   - *options* 包括:

     -noWrappedOperations (-w)
     禁止对已打包的操作进行检测。生成用于请求和响应消息的 Java bean。

     注: 这不是缺省值。

     -role (-r)
     指定值 **client** 以便为客户端开发生成文件和绑定文件。

     -container (-c)
     要使用的客户端容器。有效的自变量包括:

     client 客户机容器

     ejb Enterprise JavaBeans（EJB）容器。

     none 无容器

web Web 容器

-output (-o)
要存储已生成的文件的文件夹。

要获得 WSDL2Java 参数的完整列表，请使用 **-help** 命令行开关，或参阅联机帮助以了解 WID/RAD 中的 WSDL2Java 工具。

- *WSDLfilepath* 是您已从 WebSphere 环境中导出或从客户机 CD 中复制的 WSDL 文件的路径和文件名。

下列示例为人员任务活动 Web Service API 生成代理客户机:

```
call wsdl2java.bat -r client -c client -noWrappedOperations
                   -output c:\ws\proxyClient  c:\ws\bin\HTMWS.wsdl
```

2. 包括在项目中生成的类文件。

为 BPEL 流程创建辅助控件类（Java Web Service）

具体 API 请求（例如，sendMessage 或 call）中引用的业务对象需要客户机应用程序使用“document/literal 包装”样式元素。客户机应用程序需要辅助控件类来帮助它们生成必需的包装器元素。

开始之前

要创建辅助控件类，您必须从 WebSphere Process Server 环境中导出 Web Service API 的 WSDL 文件。

关于此任务

Web Service API 的 call() 和 sendMessage() 操作允许与 WebSphere Process Server 上的 BPEL 流程进行交互。call() 操作的输入消息期望提供该流程输入消息的 document/literal 包装器。

现在有多种可能的技术为 BPEL 流程或人员任务生成辅助控件类，其中包括:

1. 使用 SoapElement 对象。

   在 WebSphere Integration Developer 中提供的 Rational Application Developer 环境中，Web Service 引擎支持 JAX-RPC 1.1。在 JAX-RPC 1.1 中，SoapElement 对象扩展文档对象模型（DOM）元素，以便可以使用 DOM API 来创建、阅读、装入和保存 SOAP 消息。

   例如，假定 WSDL 文件包含工作流过程或人员任务的下列输入消息:

```
<xsd:element name="operation1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input1" nillable="true" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

   在开发流程或人员任务模块时创建 WSDL 文件。

   要使用 DOM API 在客户机应用程序中创建相应的 SOAP 消息:

```
SOAPFactory soapfactoryinstance = SOAPFactory.newInstance();
SOAPElement soapmessage = soapfactoryinstance.createElement
                          ("operation1", namespaceprefix, interfaceURI);
SOAPElement inputelement = soapfactoryinstance.createElement("input1");
inputelement.addTextNode( message value);
soapmessage.addChildElement(outputelement);
```

下列示例说明如何在客户机应用程序中为 sendMessage 操作创建输入参数:

```
SendMessage inWsend = new SendMessage();
inWsend.setProcessTemplateName(processtemplatename);
inWsend.setPortType(porttype);
inWsend.setOperation(operationname);
inWsend.set_any(soapmessage);
```

2. 使用 WebSphere 定制数据绑定功能部件。

   下列 developerWorks 文章描述了此技术:

   - 如何为 Web Service 选择定制映射技术
   - 使用 EMF SDO 为复杂的 XML 模式开发 Web Service

    与基于文档的 Web Service 的模式和策略的互操作性

    对包含可选 JAX-RPC 1.0/1.1 XML 模式类型的模式/WSDL 的 Web Service 支持

创建客户机应用程序（Java Web Service）

客户机应用程序将请求发送至 Web Service API 并从 Web Service API 接收响应。通过使用代理客户机管理通信和辅助控件类以对复杂数据类型进行格式化，客户机应用程序可以调用 Web Service 方法，其方式跟这些方法是本地函数一样。

开始之前

在开始创建客户机应用程序之前，生成代理客户机和所有必需的辅助控件类。

关于此任务

您可以使用诸如 IBM Rational Application Developer（RAD）之类与 Web Service 兼容的开发工具来开发客户机应用程序。可以构建任何类型的 Web Service 应用程序来调用 Web Service API。

过程

1. 创建新的客户机应用程序项目。
2. 生成代理客户机，然后将该 Java 辅助控件类添加至项目中。
3. 编写客户机应用程序代码。
4. 构建项目。
5. 运行客户机应用程序。

下列示例说明如何使用业务流程管理器 Web Service API。

```
// create the proxy
       BFMIFProxy proxy = new BFMIFProxy();
 // prepare the input data for the operation
      GetProcessTemplate iW = new GetProcessTemplate();
      iW.setIdentifier(your_process_template_name);
```

```
// invoke the operation
     GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

// process output of the operation
     ProcessTemplateType ptd = oW.getProcessTemplate();
     System.out.println("getName= " + ptd.getName());
     System.out.println("getPtid= " + ptd.getPtid());
```

相关任务

第 107 页的『生成代理客户机（Java Web Service）』
Java Web Service 客户机应用程序使用*代理客户机*与 Web Service API 进行交互。

第 110 页的『为 BPEL 流程创建辅助控件类（Java Web Service）』
具体 API 请求（例如，sendMessage 或 call）中引用的业务对象需要客户机应用程序使用“document/literal 包装”样式元素。客户机应用程序需要辅助控件类来帮助它们生成必需的包装器元素。

添加安全性（Java Web Service）

您必须通过在客户机应用程序中实现安全性机制来保护 Web Service 通信安全。

WebSphere Application Server 当前支持下列 Web Service API 的安全性机制:

- 用户名令牌
- 轻量级第三方认证（LTPA）

相关概念

业务流程的授权角色
角色是权限级别相同的一组人员。可以对业务流程执行的操作取决于您的授权角色。此角色可以是 J2EE 角色，也可以是基于实例的角色。

相关信息

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

实现用户名令牌:

用户名令牌安全性机制提供用户名和密码凭证。

关于此任务

凭借用户名令牌安全性机制，可以选择实现各种*回调处理程序*。具体情况视您所作的选择而定:

- 每次运行客户机应用程序时，系统都会提示您提供用户名和密码。
- 用户名和密码已写入部署描述符。

在这些情况下，提供的用户名和密码必须与相应业务流程容器或人员任务容器中的已授权角色的用户名和密码相匹配。

用户名和密码封装在请求消息包络中，并且就这样“以明文方式”显示在 SOAP 消息头。因此，强烈建议您将客户机应用程序配置为使用 HTTPS（HTTP over SSL）通信协议。然后，就可以对所有通信进行加密。当指定 Web Service API 的端点 URL 地址时，您可以选择 HTTPS 通信协议。

要定义用户名令牌:

过程

1. 创建安全性令牌:
   a. 打开模块的**部署编辑器**
   b. 单击 **WS 扩展**选项卡。
   c. 在**服务引用**下面，可能会列示以下 Web Service 引用:
      - service/BFMWSService（对于业务流程）
      - service/HTMWSService（对于人员任务）

      列示的内容取决于在生成代理客户机时是否添加 BFMWS.wsdl（对于业务流程）和/或 HTMWS.wsdl（对于人员任务）。
   d. 对于这两个服务引用:
      1) 选择其中一个**服务引用**。
      2) 展开**请求生成器配置**节。
      3) 展开**安全性令牌**子节。
      4) 单击**添加**。这将打开安全性令牌窗口。
      5) 在**名称**字段中，输入新安全性令牌的名称: **UserNameTokenBFM** 或 **UserNameTokenHTM**。
      6) 在**标记类型**下拉列表中，选择 **Username**。（自动使用缺省值来填充**局部名**字段。）
      7) 将 **URI** 字段留为空白。用户名令牌不需要 URI 值。
      8) 单击**确定**。
2. 创建令牌生成器:
   a. 打开模块的**部署编辑器**
   b. 单击 **WS 绑定**选项卡
   c. 在**服务引用**下面，相同的 Web Service 引用列示在上一步:
      - service/BFMWSService（对于业务流程）
      - service/HTMWSService（对于人员任务）
   d. 对于这两个服务引用:
      1) 选择其中一个**服务引用**。
      2) 展开**安全请求生成器绑定配置**节。
      3) 展开**令牌生成器**子节。
      4) 单击**添加**。这将打开“令牌生成器”窗口。
      5) 在**名称**字段中，输入新令牌生成器的名称，例如“UserNameTokenGeneratorBFM”或“UserNameTokenGeneratorHTM”。
      6) 在**令牌生成器类**字段中，确保已选择下列令牌生成器类: **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**。
      7) 在**安全性令牌**下拉列表中，选择先前创建的相应安全性令牌。
      8) 选择**使用值类型**复选框。
      9) 在**值类型**字段中，选择**用户名令牌**。（将自动填充**局部名**字段，以反映您的**用户名令牌**选择。）

10) 在**回调处理程序**字段中，输入“com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler”（运行客户机应用程序时，它提示您输入用户名和密码）或“com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler”。

11) 如果选择 **NonPromptCallbackHandler**，那么必须在部署描述符的相应字段中指定有效的用户名和密码。

12) 单击**确定**。

相关任务

第 100 页的『指定 Web Service 端点地址』

该 Web Service 端点地址是客户机应用程序为访问 Web Service API 而必须指定的 URL。该端点地址写入 WSDL 文件中，您可以将该文件导出以便为客户机应用程序生成代理客户机。

相关信息

IBM WebSphere Developer 技术日志: Web Service 安全性与 WebSphere Application Server V6

实现 LTPA 安全性机制:

当客户机应用程序在先前建立的安全上下文中运行时，可以使用轻量级第三方认证（LTPA）安全性机制。

关于此任务

仅当客户机应用程序在已建立安全上下文的安全环境中运行时，LTPA 安全性机制才可用。例如，如果客户机应用程序在 Enterprise JavaBeans（EJB）容器中运行，那么在可以调用客户机应用程序之前必须登录该 EJB 客户机。然后，建立安全上下文。如果该 EJB 客户机应用程序接着调用 Web Service，那么 LTPA 回调处理程序会检索安全上下文中的 LTPA 令牌，并将其添加至 SOAP 请求消息。在服务器端上，该 LTPA 令牌由 LTPA 机制处理。

要实现 LTPA 安全性机制，请执行下列操作:

过程

1. 在 WebSphere Integration Developer 中提供的 Rational Application Developer 环境中，选择 **WS 绑定 → 安全性请求生成器绑定配置 → 令牌生成器**。
2. 创建安全性令牌:
   a. 打开模块的**部署编辑器**
   b. 单击 **WS 扩展**选项卡。
   c. 在**服务引用**下面，可能会列示以下 **Web Service 引用**:
      - service/BFMWSService（对于业务流程）
      - service/HTMWSService（对于人员任务）

      列示的内容取决于在生成代理客户机时是否添加 BFMWS.wsdl（对于业务流程）和/或 HTMWS.wsdl（对于人员任务）。
   d. 对于这两个服务引用:
      1) 选择其中一个**服务引用**。

2) 展开**请求生成器配置**节。
3) 展开**安全性令牌**子节。
4) 单击**添加**。这将打开安全性令牌窗口。
5) 在**名称**字段中，输入新安全性令牌的名称：**LTPATokenBFM** 或 **LTPATokenHTM**。
6) 在**标记类型**下拉列表中，选择 **LTPAToken**。（将自动使用缺省值来填充 **URI** 和**局部名**字段。）
7) 单击**确定**。

3. 创建令牌生成器:
   a. 打开模块的**部署编辑器**
   b. 单击 **WS 绑定**选项卡
   c. 在**服务引用**下面，相同的 Web Service 引用列示在上一步:
      - service/BFMWSService（对于业务流程）
      - service/HTMWSService（对于人员任务）
   d. 对于这两个服务引用:
      1) 选择其中一个**服务引用**。
      2) 展开**安全请求生成器绑定配置**节。
      3) 展开**令牌生成器**子节。
      4) 单击**添加**。这将打开“令牌生成器”窗口。
      5) 在**名称**字段中，输入新令牌生成器的名称，例如“LTPATokenGeneratorBFM”或“LTPATokenGeneratorHTM”。
      6) 在**令牌生成器类**字段中，确保已选择下列令牌生成器类：**com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**。
      7) 在**安全性令牌**下拉列表中，选择先前创建的相应安全性令牌。
      8) 选择**使用值类型**复选框。
      9) 在**值类型**字段中，选择 **LTPAToken**。（将自动填充 **URI** 和**局部名**字段，以反映您的 **LTPA 令牌**选择。）
      10) 在**回调处理程序**字段中，输入“com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler”。
      11) 单击**确定**。

结果

在运行时期间，**LTPATokenCallbackHandler** 会检索现有安全上下文中的 LTPA 令牌，然后将其添加至 SOAP 请求消息。

添加事务支持（Java Web Service）

通过将客户机应用程序上下文作为服务请求的一部分来传递，可以将 Java Web Service 客户机应用程序配置为允许服务器端请求处理参与客户机事务。Web Service 原子事务（WS-AT）规范中定义了此原子事务支持。

关于此任务

WebSphere Application Server 将每个 Web Service API 请求作为单独的原子事务来运行。客户机应用程序可以配置为采用下列其中一种方式来使用事务支持:

- 参与事务。在客户机应用程序事务上下文内执行服务器端请求处理。然后，如果服务器在 Web Service API 请求运行时遇到问题并回滚，那么也会回滚客户机应用程序的请求。
- 不使用事务支持。WebSphere Application Server 仍会创建可在其中运行该请求的新事务，但不在客户机应用程序事务上下文内执行服务器端请求处理。

在 .NET 环境中开发客户机应用程序

Microsoft .NET 提供了功能强大的开发环境，您可以在此开发环境中通过 Web Service 来连接应用程序。

生成代理客户机（.NET）

.NET 客户机应用程序使用*代理客户机*与 Web Service API 进行交互。代理客户机使客户机应用程序摆脱 Web Service 消息传递协议带来的复杂性。

开始之前

要创建代理客户机，您必须首先从 WebSphere 环境中导出许多 WSDL 文件，然后将它们复制至客户机编程环境。

注: 如果具有 WebSphere Process Server 客户机 CD，那么可以从该 CD 中复制文件。

关于此任务

代理客户机由一组 C# bean 类组成。每个类包含单个 Web Service 所显示的所有方法和对象。这些服务方法将参数组装至完整的 SOAP 消息，通过 HTTP 将 SOAP 消息发送至 Web Service，接收来自 Web Service 的响应，以及处理所有返回的数据。

注: 您只需要生成代理客户机一次。然后，所有访问 Web Service API 的客户机应用程序都可以使用同一个代理客户机。

过程

1. 使用 WSDL 命令来生成代理客户机: 请输入:

   wsdl *options WSDLfilepath*

   其中:

   - *options* 包括:

     /language
     允许您指定用于创建代理类的语言。缺省值是 C#。您也可以指定 **VB**（Visual Basic）、**JS**（JScript）或 **VJS**（Visual J#）作为语言自变量。

     /output
     具有相应后缀的输出文件的名称。例如，proxy.cs

     /protocol
     在代理类中实现的协议。**SOAP** 是缺省设置。

   要获得 **WSDL.exe** 参数的完整列表，请使用 **/?** 命令行开关，或参阅联机帮助以了解 Visual Studio 中的 WSDL 工具。

- *WSDLfilepath* 是您已从 WebSphere 环境中导出或从客户机 CD 中复制的 WSDL 文件的路径和文件名。

下列示例为人员任务管理器 Web Service API 生成代理客户机:

```
wsdl /language:cs /output:proxyclient.cs c:\ws\bin\HTMWS.wsdl
```

2. 将代理客户机编译为动态链接库（DLL）文件。

为 BPEL 流程创建辅助控件类（.NET）

某些 Web Service API 操作要求客户机应用程序使用“document/literal”样式包装的元素。客户机应用程序需要辅助控件类来帮助它们生成必需的包装器元素。

开始之前

要创建辅助控件类，您必须从 WebSphere Process Server 环境中导出 Web Service API 的 WSDL 文件。

关于此任务

Web Service API 的 call() 和 sendMessage() 操作导致在 WebSphere Process Server 中启动 BPEL 流程。call() 操作的输入消息期望提供 BPEL 流程输入消息的 document/literal 包装器。要为 BPEL 流程生成必需的 Bean 和类，请将 `<wsdl:types>` 元素复制到新的 XSD 文件，然后使用 xsd.exe 工具来生成辅助控件类。

过程

1. 如果尚未执行此操作，那么从 WebSphere Integration Developer 中导出 BPEL 流程接口的 WSDL 文件。
2. 在文本编辑器或 XML 编辑器中打开 WSDL 文件。
3. 复制 `<wsdl:types>` 元素的所有子元素内容，然后将其粘贴到新的框架 XSD 文件。
4. 对 XSD 文件运行 `xsd.exe` 工具:

   call xsd.exe file.xsd /classes /o

   其中:

   file.xsd
   要转换的 XML 模式定义文件。

   /classes (/c)
   生成与所指定 XSD 文件的内容相对应的辅助控件类。

   /output (/o)
   为生成的文件指定输出目录。如果省略此目录，那么缺省值为当前目录。

   例如:

   call xsd.exe ProcessCustomer.xsd /classes /output:c:\temp
5. 添加已生成至客户机应用程序的类文件。 例如，如果在使用 Visual Studio，那么可以使用**项目 → 添加现有项**菜单选项来执行此操作。

如果 ProcessCustomer.wsdl 文件包含下列内容:

```
<?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>
```

生成的 XSD 文件包含:

```
<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>
```

相关信息

XML 模式定义工具（XSD.EXE）的 Microsoft 文档

创建客户机应用程序（.NET）

客户机应用程序将请求发送至 Web Service API 并从 Web Service API 接收响应。通过使用代理客户机管理通信和辅助控件类以对复杂数据类型进行格式化，客户机应用程序可以调用 Web Service 方法，其方式跟这些方法是本地函数一样。

开始之前

在开始创建客户机应用程序之前，生成代理客户机和所有必需的辅助控件类。

关于此任务

您可以使用诸如 Visual Studio .NET 之类与 .NET 兼容的开发工具来开发 .NET 客户机应用程序。可以构建任何类型的 .NET 应用程序来调用普通的 Web Service API。

过程

1. 创建新的客户机应用程序项目。 例如，在 Visual Studio 中创建 **WinFX Windows® 应用程序**。
2. 在项目选项中，将引用添加至代理客户机的动态链接库（DLL）文件。 将所有包含业务对象定义的辅助控件类添加至您的项目。例如，在 Visual Studio 中，可以使用**项目 → 添加现有项**选项来执行此操作。
3. 创建代理客户机对象。 例如:

```
HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService service =
    new HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService();
```

4. 声明在要发送至 Web Service 或从 Web Service 接收的消息中使用的任何业务对象数据类型。 例如:

```
HTMClient.HTMReference.TKIID id = new HTMClient.HTMReference.TKIID();

ClipBG bg = new ClipBG();
Clip clip = new Clip();
```

5. 调用特定 Web Service 函数，并且指定所有必需参数。 例如，要创建并启动人员任务:

```
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. 使用持久标识（上一步骤的示例中的 id）来标识远程流程和任务。 例如，要声明先前创建的人员任务:

```
HTMClient.HTMReference.claimTask claim = new HTMClient.HTMReference.claimTask();
claim.inputTask = id;
```

相关任务

第 116 页的『生成代理客户机（.NET）』
.NET 客户机应用程序使用*代理客户机*与 Web Service API 进行交互。代理客户机使客户机应用程序摆脱 Web Service 消息传递协议带来的复杂性。

第 117 页的『为 BPEL 流程创建辅助控件类（.NET）』
某些 Web Service API 操作要求客户机应用程序使用“document/literal”样式包装的元素。客户机应用程序需要辅助控件类来帮助它们生成必需的包装器元素。

添加安全性（.NET）

您可以通过将安全性机制集成到客户机应用程序中来保护 Web Service 通信安全。

关于此任务

这些安全性机制可包括用户名令牌（用户名和密码）或定制的二进制和基于 XML 的安全性令牌。

过程

1. 下载并安装 Web Services Enhancements（WSE）2.0 SP3 for Microsoft .NET。 可以从以下网址获得此产品:

   http://www.microsoft.com/downloads/details.aspx?familyid=1ba1f631-c3e7- 420a-bc1e-ef18bab66122&displaylang=en
2. 修改生成的代理客户机代码，如下所示。

   将以下内容:

```
public class Export1_MyMicroflowHttpService : System.Web.Services.Protocols.SoapHttpClientProtocol {
```

   更改为:

```
public class Export1_MyMicroflowHttpService : Microsoft.Web.Services2.WebServicesClientProtocol {
```

   注: 如果通过运行 WSDL.exe 工具来生成代理客户机，那么会丢失这些修改。
3. 通过在文件顶部添加下列各行代码来修改客户机应用程序代码:

```
using System.Web.Services.Protocols;
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Security.Tokens;
...
```

4. 添加代码以实现所需的安全性机制。 例如，下列代码将添加用户名和密码保护:

```
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);
```

查询业务流程以及与任务相关的对象

可以使用 Web Service API 在业务流程编排器数据库中查询业务流程和与任务相关的对象，以检索这些对象的特定属性。

关于此任务

该业务流程编排器数据库存储用于管理业务流程和任务的模板（模型）与实例（运行时）数据。

通过 Web Service API，客户机应用程序可以发出查询，以便从数据库中检索有关业务流程和任务的信息。

客户机应用程序可以发出一次性的查询来检索对象的特定属性。您可以保存经常使用的查询。然后，客户机应用程序就可以检索和使用这些存储查询。

对业务流程以及与任务相关的对象执行的查询

使用 Web Service API 的查询接口来获得有关业务流程和任务的信息。

客户机应用程序使用类似于 SQL 的语法来查询数据库。

Java Web Service 的示例

```
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);
```

从数据库中检索的信息作为*查询结果集*通过 Web Service API 返回。

例如:

```
QueryResultSetType queryResultSet = queryResponse1.queryResultSet;
if (queryResultSet != null) {
   Console.WriteLine("--> QueryResultSetType");
   Console.WriteLine(" . size= " + queryResultSet.size);
   Console.WriteLine(" . numberColumns= " + queryResultSet.numberColumns);
   string indent = " . ";

   // -- the query column info
   QueryColumnInfoType[] queryColumnInfo = queryResultSet.QueryColumnInfo;
   if (queryColumnInfo.Length > 0) {
      Console.WriteLine();
      Console.WriteLine("= . QueryColumnInfoType size= " + queryColumnInfo.Length);
      Console.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>");
}
```

此查询函数根据调用者的权限返回对象。查询结果集仅包含调用者有权查看的那些对象的属性。

您可以使用预定义的数据库视图来查询对象属性。对于流程模板来说，查询函数的语法如下所示:

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

对于任务模板来说，查询函数的语法如下所示:

```
TaskTemplate[] queryTaskTemplates
               (java.lang.String whereClause,
                java.lang.String orderByClause,
                java.lang.Integer threshold,
                java.util.TimeZone timezone);
```

对于其他业务流程以及与任务相关的对象来说，查询函数的语法如下所示:

```
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);
```

查询接口还包含 queryAll 方法。例如，您可以为监视目的而使用此方法来检索某个对象的所有相关数据。queryAll 方法的调用者必须具有下列其中一个 Java 2 Platform, Enterprise Edition（J2EE）角色: BPESystemAdministrator、BPESystemMonitor、TaskSystemAdministrator 或 TaskSystemMonitor。使用对象的相应工作项进行授权检查在此处不适用。

.NET 的示例

```
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);
}
```

查询参数:

每个查询都必须指定许多类似于 SQL 的子句和参数。

查询包含以下内容:

- Select 子句
- Where 子句
- Order-by 子句
- Skiptuples 参数
- Threshold 参数
- Timezone 参数

对业务流程对象和人员任务对象执行的查询的预定义视图

为业务流程对象和人员任务对象提供了预定义的数据库视图。

在查询这些对象的引用数据时，可使用这些视图。使用这些视图时，不需要显式地添加视图列的 JOIN 谓词，这些构造是自动添加的。您可以使用 Web Service API 的查询函数来查询此数据。

管理存储查询

存储查询提供了保存经常运行的查询的方法。存储查询可以是可供所有用户使用的查询（公用查询），也可以是属于特定用户的查询（专用查询）。

关于此任务

存储查询是指存储在数据库中并由名称标识的查询。专用存储查询和公用存储查询可以具有相同的名称；来自不同拥有者的专用存储查询也可以具有相同的名称。

您可以使用业务流程对象或任务对象的存储查询，也可以同时使用这两种对象类型的存储查询。

管理公用存储查询
公用存储查询由系统管理员创建。 这些查询可供所有用户使用。

为其他用户管理专用存储查询
任何用户都可以创建专用查询。这些查询只有查询的所有者和系统管理员可以使用。

处理专用存储查询
如果您不是系统管理员，那么可以创建、运行和删除自己的专用存储查询。还可以使用系统管理员创建的公用存储查询。

开发 JMS 客户机应用程序

您可以开发通过 Java 消息传递服务（JMS）API 来访问业务流程应用程序的客户机应用程序。

关于此任务

JMS 简介

WebSphere Process Server V6.1 支持基于 Java 消息传递服务（JMS）编程接口的异步消息传递作为通信方法。

JMS 为 Java 客户机（客户机应用程序或 J2EE 应用程序）提供了将请求作为 JMS 消息进行创建、发送、接收和读取的常用方法。

JMS 是基于异步消息的接口:

- 使用**点到点或发布/预订消息传递**。无需显式请求，基于消息的框架即可将信息推送到其他应用程序。相同信息可并行传递给多个订户。业务流程编排器的 JMS 接口仅支持点到点消息传递。
- 提供**节奏独立性**。JMS 框架以异步方式工作，但也能够模拟同步请求/响应方式。这允许源和目标系统同时工作，而不必等待对方。对于业务流程编排器来说，此功能特别有用，原因是它使您能够以异步方式与长期运行的业务流程进行交互。
- 支持**事务**。事务使客户机应用程序能够处理已发送的消息组或将它们作为单个原子单元来接收。JMS 事务在服务器的事务内运行。对于业务流程编排器的 JMS 接口，您通常为每个事务发送和接收单个消息。
- **保证信息传递**。JMS 框架可以采用事务方式管理消息和确保消息传递（虽然没有保证及时传递）。对于业务流程编排器，此可靠的消息传递功能特别重要，这是因为它在处理业务流程。
- 确保**不同种类的框架之间的互操作性**。源和目标应用程序可以在不同种类的环境中工作，而不必处理与其各自框架相关的通信和执行问题。
- **使交换更为流畅**。切换到消息方式允许您交换更高细颗粒度的信息。

业务流程的要求

使用 WebSphere Integration Developer 开发以在业务流程编排器上运行的业务流程必须符合特定的规则，以便通过 JMS API 进行访问。

这些要求是:

1. 必须使用在 Java API for XML-based RPC（JAX-RPC 1.1）规范中定义的“document/literal 包装”样式来定义业务流程的接口。这是使用 WebSphere Integration Developer 开发的所有业务流程和人员任务的缺省样式。
2. 业务流程和人员任务所显示的 Web Service 操作的故障消息必须包含使用 XML 模式元素定义的单个 WSDL 消息部分。例如:

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

   相关信息

   Java API for XML based RPC（JAX-RPC）下载页面

   我应该使用哪种 WSDL 样式？

访问 JMS 接口

要通过 JMS 接口发送和接收消息，应用程序必须先创建与 BPC.cellname.Bus 的连接，创建会话，然后生成消息生产者和使用者。

关于此任务

流程服务器接受遵循点到点范例的 Java 消息服务（JMS）消息。发送或接收 JMS 消息的应用程序必须执行下列操作。

下列示例假定已在受管环境（EJB、应用程序客户机或 Web 客户机容器）中执行 JMS 客户机。如果要在 J2SE 环境中执行 JMS 客户机，那么请参阅位于 http://www-1.ibm.com/support/docview.wss?uid=swg24012804 的“IBM Client for JMS on J2SE with IBM WebSphere Application Server”。

过程

1. 创建与 BPC.*cellname*.Bus 的连接。 客户机应用程序的请求中不存在预先配置的连接工厂: 客户机应用程序可以使用 JMS API 的 ReplyConnectionFactory 或创建其自己的连接工厂，在此情况下，它可以使用 Java 命名和目录接口（JNDI）查询来检索连接工厂。JNDI 查询名必须与配置业务流程编排器的外部请求队列时指定的名称相同。以下示例假定客户机应用程序创建它自己的连接工厂“jms/clientCF”。

   ```
   // Obtain the default initial JNDI context.
   Context initialContext = new InitialContext();

   // Look up the connection factory.
   // Create a connection factory that connects to the BPC bus.
   // Call it, for example, "jms/clientCF".
   // Also configure an appropriate authentication alias.
   ConnectionFactory connectionFactory =
               (ConnectionFactory)initialcontext.lookup("jms/clientCF");

   // Create the connection.
   Connection connection = connectionFactory.createConnection();
   ```

2. 创建会话以便可以创建消息生产者和使用者。

   ```
   // Create a transaction session using auto-acknowledgement.
   Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
   ```

3. 创建消息生产者以发送消息。 JNDI 查询名必须与配置业务流程编排器的外部请求队列时指定的名称相同。

   ```
   // Look up the destination of the Business Process Choreographer input queue to
   // send messages to.
   Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");
   ```

```
// Create a message producer.
MessageProducer producer = session.createProducer(sendQueue);
```

4. 创建消息使用者以接收应答。 应答目标的 JNDI 查询名可以指定用户定义的目标，但也可以指定业务流程编排器定义的缺省应答目标 jms/BFMJMSReplyQueue。在这两种情况下，应答目标都必须在 BPC.<cellname>.Bus 上。

```
// Look up the destination of the reply queue.
Queue  replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

// Create a message consumer.
MessageConsumer consumer = session.createConsumer(replyQueue);
```

5. 发送消息。

```
// Start the connection.
connection.start();

// Create a message - see the task descriptions for examples - and send it.
// This method is defined elsewhere ...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Set mandatory JMS header.
// targetFunctionName is the operation name of JMS API
// (for example, getProcessTemplate, sendMessage)
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Set the reply queue; this is mandatory if the replyQueue
// is not the default queue (as it is in this example).
requestMessage.setJMSReplyTo(replyQueue);

// Send the message.
producer.send(requestMessage);

// Get the message ID.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. 接收应答。

```
// Receive the reply message and analyse the reply.
TextMessage  replyMessage = (TextMessage) consumer.receive();

// Get the payload.
String payload = replyMessage.getText();

session.commit();
```

7. 关闭连接并释放资源。

```
 // Final housekeeping; free the resources.
session.close();
connection.close();
```

注: 不必在每个事务完成后关闭连接。一旦建立连接，可在关闭该连接前交换任意数目的请求消息和响应消息。该示例通过单个业务方法中的简单调用来说明一个简单的案例。

业务流程编排器 JMS 消息的结构

每个 JMS 消息的消息头和主体均必须具有预定义的结构。

Java 消息服务（JMS）消息由以下部分组成:

- 消息头，用于消息标识和传递信息。
- 消息主体（有效内容），它拥有内容。

业务流程编排器仅支持文本消息格式。

消息头

JMS 允许客户机访问多个消息头字段。

下列头字段可以由业务流程编排器 JMS 客户机设置:

- **JMSReplyTo**

  将应答发送到请求的目标。如果未在请求消息中指定此字段，那么会将应答发送到导出接口的缺省应答目标（导出是业务流程组件的客户机接口呈示）。可使用 `initialContext.lookup("jms/BFMJMSReplyQueue");` 获取此目标

- **TargetFunctionName**

  WSDL 操作的名称，例如“queryProcessTemplates”。您必须始终设置此字段。请注意，TargetFunctionName 指定此处描述的通用 JMS 消息接口的操作。这不应该与可间接调用（例如，使用 **call** 或 **sendMessage** 操作调用）的具体流程或任务所提供的操作相混淆。

业务流程编排器客户机还可以访问下列头字段:

- **JMSMessageID**

  唯一地标识消息。它在发送消息时由 JMS 提供程序设置。如果在发送消息之前客户机设置了 JMSMessageID，那么它会被 JMS 提供程序覆盖。如果认证需要消息标识，那么客户机可以在发送消息后检索 JMSMessageID。

- **JMSCorrelationID**

  链接消息。切勿设置此字段。业务流程编排器应答消息包含请求消息的 JMSMessageID。

每个响应消息包含下列 JMS 头字段:

- **IsBusinessException**

  对于 WSDL 输出消息则为“False”，对于 WSDL 故障消息则为“true”。

ServiceRuntimeExceptions 未返回到异步客户机应用程序。当处理 JMS 请求消息期间发生了重大异常时，它会导致运行时故障，从而造成正在处理此请求消息的事务回滚。然后，将会再次传递 JMS 请求消息。在将消息作为 SCA 导出一部分进行处理（例如，将消息反序列化）期间，如果早期发生了故障，那么将会进行重试，尝试次数最多为 SCA 导出的接收目标所指定的已失败传递的最大数目。在达到已失败传递的最大数目后，请求消息将被添加到业务流程编排器总线的系统异常目标。然而，如果在业务流程管理器的 SCA 组件实际处理请求期间发生故障，那么已失败的请求消息由 WebSphere Process Server 的已失败事件管理基础结构处理，即，如果重试并没有解决异常情况，那么它可能会结束在已失败事件管理数据库中。

消息体

JMS 消息体是包含用于表示操作的 document/literal 包装器元素的 XML 文档的字符串。

有效请求消息体的简单示例是:

```
<?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>
```

JMS 呈示授权

要授权使用 JMS 接口，必须在 WebSphere Application Server 中启用安全设置。

当安装业务流程容器后，必须将角色 **JMSAPIUser** 映射到某个用户标识。此用户标识用于发出所有 JMS API 请求。例如，如果 **JMSAPIUser** 已映射到“用户 A”，那么所有 JMS API 请求对于流程引擎来说就像是从“用户 A”发出一样。

必须为 **JMSAPIUser** 角色分配下列权限:

请求	所需权限
forceTerminate	流程管理员
sendEvent	潜在活动所有者或流程管理员

注: 对于其他所有请求，不需要特权。

对具有业务流程管理员角色的人员授予特权。业务流程管理员是特殊角色；它与流程实例的流程管理员不同。业务流程管理员具有所有特权。

当流程实例存在时，您不能从用户注册表中删除流程启动者的用户标识。如果删除了该用户标识，那么此流程的导航就无法继续进行。并且系统日志文件将包含以下异常:

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

JMS API 的概述

JMS 消息接口（此后称为“JMS API”） 允许您开发客户机应用程序，这些应用程序可以采用异步方式访问在业务流程编排器环境中运行的业务流程。

JMS API 允许客户机应用程序采用异步方式与微流动和长期运行的流程进行交互。

JMS API 展示与 Web Service API 相同的接口，但存在下列例外情况:

- 对于 Web Service API，`call` 操作只能用于调用微流动。然而，借助于 JMS API，`call` 操作可用于调用微流动和长期运行的流程。
- 下列操作未通过 JMS API 展示:
  - `callAsync` 操作（与其关联的回调操作一起）。
  - `completeAndClaimSuccessor` 和 `getParticipatingTask` 操作

示例 - 执行长期运行的流程

对于要处理长期运行的流程的普通客户机应用程序，这些步骤的顺序是:

1. 设置 JMS 环境，如第 125 页的『访问 JMS 接口』中所述。
2. 获取已安装流程定义的列表:
   - 发送 `queryProcessTemplates`

- 这将返回 **ProcessTemplate** 对象的列表。

3. 获取启动活动的列表（具有 createInstance="yes" 的 receive 或 pick）:
   - 发送 getStartActivities。
   - 这将返回 **InboundOperationTemplate** 对象的列表。
4. 创建输入消息。这是特定于环境的，并且可能需要使用预先部署的特定于流程的工件。
5. 创建流程实例:
   - 发出 sendMessage。

   使用 JMS API，您还可以使用 call 操作来与业务流程提供的长期运行的请求/ 响应操作进行交互。此操作向指定应答目标返回操作结果或故障，即使在很长时间后也是如此。因此，如果您使用 call 操作，那么不需要使用 query 和 getOutputMessage 操作来获取流程输出或故障消息。
6. （可选）通过重复下列步骤从流程实例中获取输出消息:
   - 发出 query 以获取流程实例的已完成状态。
   - 发出 getOutputMessage。
7. （可选）使用由流程展示的其他操作:
   - getWaitingActivities 或 getActiveEventHandlers，可获取 **InboundOperationTemplate** 对象的列表。
   - 创建输入消息
   - 使用 sendMessage 来发送消息
8. （可选）使用 getCustomProperties 与 setCustomProperties 来获取和设置在流程或已包含的活动上定义的定制属性。
9. （可选）处理完流程实例:
   - 发送 delete 和 terminate，以便处理完长期运行的流程。

开发 JMS 应用程序

必须采用 Java 语言在 Java 2 Enterprise Edition（J2EE）环境中开发 JMS 客户机应用程序。

关于此任务

JMS 客户机应用程序通过 JMS API 交换请求和响应消息。客户机应用程序使用表示相应操作的 document/literal 包装器的 XML 元素来填充 JMS TextMessage 消息体，以创建请求消息。

复制工件

可以从 WebSphere 环境复制许多工件以帮助创建 JMS 客户机应用程序。

仅在使用 BOXMLSerializer 来创建 JMS 消息体时，才必须使用这些工件。

有两种方法来获得这些工件:

- 从 WebSphere Process Server 环境中发布和导出它们。

  对于 WebSphere Process Server 6.1，将在 *install_root*\ProcessChoreographer\client 目录中找到所有客户机工件。对于 JMS API，这些工件是:

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

- 从 WebSphere Process Server 客户机 CD 中复制文件。

从服务器环境中发布工件:

要帮助开发用于访问 JMS API 的客户机应用程序，您可以从 WebSphere 服务器环境发布许多工件。

关于此任务

对于 WebSphere Process Server 6.1，您将在 *was_home*\ProcessChoreographer\client 目录中找到所有客户机工件。对于 JMS API，这些工件是:

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

在发布这些工件之后，将它们复制到客户机编程环境。

从客户机 CD 中复制文件:

WebSphere Process Server 客户机 CD 中提供有访问 JMS API 所必需的文件。

过程

1. 访问该客户机 CD，然后浏览至 `ProcessChoreographer\client` 目录。
2. 将必需的文件复制至客户机应用程序开发环境

   对于 WebSphere Process Server 6.1，将在 \ProcessChoreographer\client 目录中找到所有客户机工件。对于 JMS API，这些工件是:

   BFMIF.wsdl

   BFMIF.xsd

   BPCGen.xsd

检查业务异常的响应消息

JMS 客户机应用程序必须检查所有业务异常的响应消息的消息头。

关于此任务

JMS 客户机应用程序必须先检查响应消息的消息头中的 **IsBusinessException** 属性。

例如:

```
// receive response message
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject().receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse is a bussiness fault
  // any api can end w/a processFaultMsg
  // the call api also w/a businessFaultMsg
```

```
 }
else {
  // strResponse is the output message
}
```

使用 JSF 组件为业务流程和人员任务开发 Web 应用程序

业务流程编排器提供了若干个 JavaServer Faces（JSF）组件。您可以扩展和集成这些组件，以便在 Web 应用程序中添加业务流程和人员任务功能。

关于此任务

可以使用 WebSphere Integration Developer 来构建 Web 应用程序。

过程

1. 创建一个动态项目，然后更改“Web 项目功能部件”属性以包括 JSF 基本组件。

   有关创建 Web 项目的更多信息，请访问 WebSphere Integration Developer 的信息中心。

2. 添加必备的业务流程编排器资源管理器 Java 归档（JAR 文件）。

   将下列文件添加到项目的 WEB-INF/lib 目录中:

   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar
   - bpcjsfcomponents.jar

   如果要在远程服务器上部署 Web 应用程序，那么也添加下列文件。远程访问业务流程编排器 API 时需要这些文件。

   - bpe137650.jar
   - task137650.jar

   在 WebSphere Process Server 中，所有这些文件均在下列目录中:

   - 在 Windows 系统上: *install_root*\ProcessChoreographer\client
   - 在 UNIX®、Linux® 和 i5/OS® 系统上: *install_root*/ProcessChoreographer/client

3. 在 Web 应用程序部署描述符 web.xml 文件中添加所需的 EJB 引用。

```
<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. 在 JSF 应用程序中添加业务流程编排器资源管理器 JSF 组件。
   a. 在 JavaServer Pages（JSP）文件中添加应用程序所需的标记库引用。 通常，需要 JSF 和 HTML 标记库以及 JSF 组件所需的标记库。
      - `<%@ 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. 在 JSP 页面的主体中添加 `<f:view>` 标记，并在 `<f:view>` 标记中添加 `<h:form>` 标记。
   c. 在 JSP 文件中添加 JSF 组件。

      根据应用程序的不同，在 JSP 文件中添加 List 组件、Details 组件、CommandBar 组件或 Message 组件。对于每个组件，可以添加多个实例。
   d. 在 JSF 配置文件中对受管 Bean 进行配置。

      缺省情况下，配置文件是 faces-config.xml 文件。此文件在 Web 应用程序的 WEB-INF 目录中。

      根据在 JSP 文件中添加的组件的不同，还需要在 JSF 配置文件中添加对查询和其他包装器对象的引用。要确保正确地处理错误，您还需要在 JSF 配置文件中为错误页定义错误 Bean 和导航目标。

```
<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>
The general error page.
</description>
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

      在触发了错误页的错误情况下，将对错误 Bean 设置此异常。
   e. 实现支持 JSF 组件所需的定制代码。
5. 部署应用程序。

如果要在 Network Deployment 环境中部署应用程序，那么将目标资源 Java 命名和目录接口（JNDI）名称更改为可在单元中找到业务流程管理器 API 和人员任务管理器 API 的值。

- 如果已在同一受管单元中的另一服务器上配置业务流程容器，那么该名称具有下列结构:

  cell/*nodes*/*nodename*/*servers*/*servername*/com/ibm/bpe/api/BusinessManagerHome
  cell/*nodes*/*nodename*/*servers*/*servername*/com/ibm/task/api/HumanTaskManagerHome

- 如果已在同一单元中的集群上配置业务流程容器，那么该名称具有下列结构:

  cell/*clusters*/*clustername*/com/ibm/bpe/api/BusinessFlowManagerHome
  cell/*clusters*/*clustername*/com/ibm/task/api/HumanTaskManagerHome

将 EJB 引用映射到 JNDI 名称，或者在 ibm-web-bnd.xmi 文件中手动添加引用。

下表列示了引用绑定及其缺省映射。

表 33. 从引用绑定到 JNDI 名称的映射

引用绑定	JNDI 名称	注释
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	远程会话 Bean
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	本地会话 Bean
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	远程会话 Bean
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	本地会话 Bean

结果

部署的 Web 应用程序将包含业务流程编排器资源管理器组件提供的功能。

下一步做什么?

如果您要对流程和任务消息使用定制 JSP，那么必须将用于部署 JSP 的 Web 模块映射到定制 JSF 客户机映射到的同一服务器上。

相关概念

第 135 页的『JSF 组件中的错误处理』
JavaServer Faces（JSF）组件使用预定义的受管 bean BPCError 来处理错误。在触发了错误页的错误情况下，将对错误 Bean 设置此异常。

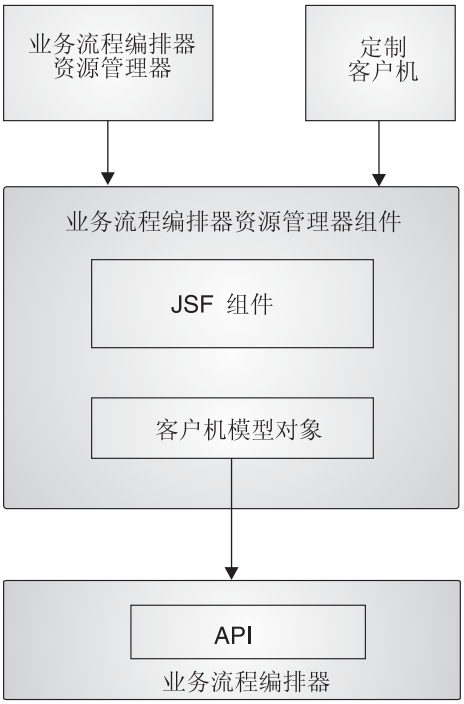
相关任务

第 20 页的『访问会话 Bean 的远程接口』
EJB 客户机应用程序通过 Bean 的远程 home 接口来访问会话 Bean 的远程接口。

业务流程编排器资源管理器组件

业务流程编排器资源管理器组件是一组可配置的可复用元素，它们基于 JavaServer Faces（JSF）技术。可以在 Web 应用程序中嵌入这些元素。然后，Web 应用程序就可以访问已安装的业务流程和人员任务应用程序。

这些组件由一组 JSF 组件和一组客户机模型对象组成。下图显示了这些组件与业务流程编排器、业务流程编排器资源管理器以及其他定制客户机的关系。



JSF 组件

业务流程编排器资源管理器组件包括下列 JSF 组件。在构建要与业务流程和人员任务配合工作的 Web 应用程序时，您将这些 JSF 组件嵌入到 JavaServer Pages（JSP）文件中。

- List 组件

  List 组件在表中显示一组应用程序对象，例如，任务、活动、流程实例、流程模板、工作项或升级。此组件有相关联的列表处理程序。

- Details 组件

  Details 组件显示任务、工作项、活动、流程实例和流程模板的属性。此组件有相关联的详细信息处理程序。

- CommandBar 组件

  CommandBar 组件显示带有按钮的栏。这些按钮代表作用于详细视图中的对象或列表中的所选对象的命令。这些对象是由列表处理程序或详细信息处理程序提供的。

- Message 组件

  Message 组件显示可以包含服务数据对象（SDO）或简单类型的消息。

客户机模型对象

客户机模型对象与 JSF 组件配合使用。这些对象实现了底层业务流程编排器 API 的一些接口并包装了原始对象。客户机模型对象提供了对某些属性的标签和转换器的本地语言支持。

JSF 组件中的错误处理

JavaServer Faces（JSF）组件使用预定义的受管 bean BPCError 来处理错误。在触发了错误页的错误情况下，将对错误 Bean 设置此异常。

此 Bean 实现了 com.ibm.bpc.clientcore.util.ErrorBean 接口。在下列情况下将显示错误页:

- 在执行为列表处理程序定义的查询期间发生了错误，并且该错误是由命令的 execute 方法作为 ClientException 错误生成的
- 如果命令的 execute 方法生成了 ClientException 错误，并且此错误既不是 ErrorsInCommandException 错误，也未实现 CommandBarMessage 接口
- 如果在该组件中显示了错误消息，并且您转到了该消息的超链接

com.ibm.bpc.clientcore.util.ErrorBeanImpl 接口提供了缺省实现。

此接口的定义如下所示:

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * This setter method call allows a locale and
     * the exception to be passed. This allows the
     * getExceptionMessage methods to return localized Strings
     *
     */
    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();

    /*
     * This method returns the exception message
     * concatenated recursively with the messages of all
     * the nested exceptions.
     */
    public String getAllExceptionMessages();

    /*
     * This method is returns the exception stack
     * concatenated recursively with the stacks of all
     * the nested exceptions.
     */
    public String getAllExceptionStacks();
}
```

相关概念

第 140 页的『List 组件中的错误处理』

使用 List 组件以便在 JSF 应用程序中显示列表时，可以利用 com.ibm.bpe.jsf.handler.BPCListHandler 类提供的错误处理功能。

客户机模型对象的缺省转换器和标签

客户机模型对象实现了业务流程编排器 API 的相应接口。

List 组件和 Details 组件对任何 Bean 均有影响。您可以显示 Bean 的所有属性。然而，如果要设置将用于 Bean 的属性的转换器和标签，那么必须对 List 组件使用 column 标记，或者对 Details 组件使用 property 标记。您可以通过定义下列静态方法来为属性定义缺省转换器和标签，而无需设置转换器和标签。您可以定义下列静态方法:

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

下表显示客户机模型对象，这些对象实现相应业务流程管理器和人员任务管理器 API 类，并且为其属性提供缺省标签和转换器。接口的这种包装为一组属性提供了与语言环境相关的标签和转换器。下表说明了从业务流程编排器接口到相应客户机模型对象的映射。

表 34. 从业务流程编排器接口到客户机模型对象的映射

业务流程编排器接口	客户机模型对象类
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

将 List 组件添加至 JSF 应用程序

使用业务流程编排器资源管理器的 List 组件来显示一组客户机模型对象，例如，业务流程实例或任务实例。

过程

1. 在 JavaServer Pages（JSP）文件中添加 List 组件。

   在 h:form 标记中添加 bpe:list 标记。bpe:list 标记必须包含 model 属性。在 bpe:list 标记中添加 bpe:column 标记，以添加列表每一行所要显示的对象的属性。

   下列示例说明如何添加 List 组件以显示任务实例。

   ```
   <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>
   ```

   model 属性引用了名为 TaskPool 的受管 Bean。该受管 Bean 提供了一组 Java 对象，列表将对那些对象执行迭代，然后在各行中显示那些对象。

2. 配置 bpe:list 标记中引用的受管 Bean。

对于 List 组件，这个受管 Bean 必须是 com.ibm.bpe.jsf.handler.BPCListHandler 类的实例。

以下示例说明如何在配置文件中添加 TaskPool 受管 Bean。

```
<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>
```

此示例显示 TaskPool 有两个可配置的属性：query 和 type。查询属性的值引用了另一个受管 Bean，即 TaskPoolQuery。type 属性值指定了 Bean 类，该类的属性将显示在您所见的列表的列中。相关联的查询实例也可以带有 type 属性。如果指定了 type 属性，那么它必须与您对列表处理程序指定的 type 相同。

只要查询结果可以表示为强类型 Bean 的列表，您可以将任何类型的查询逻辑添加至 JSF 应用程序。例如，使用 com.ibm.task.clientmodel.bean.TaskInstanceBean 对象的列表来实现 TaskPoolQuery。

3. 添加列表处理程序所引用的受管 Bean 的定制代码。

以下示例说明如何添加 TaskPool 受管 Bean 的定制代码。

```
public class TaskPoolQuery implements Query {

  public List execute throws ClientException {

    // Examine the faces-config file for a managed bean "htmConnection".
    //
    FacesContext ctx = FacesContext.getCurrentInstance();
    Application  app = ctx.getApplication();
    ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
    htmConnection = (HTMConnection) htmVb.getValue(ctx);
    HumanTaskManagerService taskService =
         htmConnection.getHumanTaskManagerService();
```

```
    // Then call the actual query method on the Human Task Manager service.
    //
    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 );

// Transforms each row in the QueryResultSet to a task instance beans.
 for (int i = 0; i < entries; i++) {
   result.next();
   array.add( new TaskInstanceBean( result, connection ));
 }
 return array ;
  }
}
```

TaskPoolQuery Bean 查询 Java 对象的属性。此 Bean 必须实现 com.ibm.bpc.clientcore.Query 接口。当列表处理程序刷新其内容时，它将调用该查询的 execute 方法。此调用将返回 Java 对象列表。getType 方法必须返回那些已返回的 Java 对象的类名。

结果

现在，JSF 应用程序包含一个 JavaServer 页面，该页面显示所请求的对象列表的属性，例如可供您使用的任务实例的状态、类型、所有者和发起者。

相关概念

第 139 页的『特定于用户的时区信息』
JavaServer Faces（JSF）组件提供了实用程序来处理 List 组件中特定于用户的时区信息。

相关参考

第 141 页的『List 组件: 标记定义』
业务流程编排器资源管理器的 List 组件在表中显示一组对象，例如，任务、活动、流程实例、流程模板、工作项和升级。

列表的处理方式

List 组件的每个实例都与 com.ibm.bpe.jsf.handler.BPCListHandler 类的一个实例相关联。

此列表处理程序跟踪相关列表中选定的项，并提供了通知机制使列表条目与不同类型项的详细信息页相关联。此列表处理程序通过 `bpe:list` 标记的 **model** 属性绑定至 List 组件。

列表处理程序的通知机制是使用 com.ibm.bpe.jsf.handler.ItemListener 接口实现的。您可以在 JavaServer Faces（JSF）应用程序的配置文件中注册此接口的实现。

当单击列表中的链接时，将会触发通知。将为对其设置 **action** 属性的所有列呈示链接。**action** 属性的值是 JSF 导航目标或返回 JSF 导航目标的 JSF 操作方法。

BPCListHandler 类还提供了 refreshList 方法。可以在 JSF 方法绑定中使用此方法来实现用于再次运行查询的用户界面控件。

查询实现

可以使用列表处理程序来显示所有类型的对象及其属性。显示的列表内容取决于为该列表处理程序配置的 com.ibm.bpc.clientcore.Query 接口实现过程所返回的对象列表。可以使用 BPCListHandler 类的 setQuery 方法以编程方式设置查询，也可以在应用程序的 JSF 配置文件中配置查询。

不仅可以对业务流程编排器 API 运行查询，而且也可以对应用程序能够访问的任何其他信息来源（例如，内容管理系统或数据库）运行查询。唯一的要求是 execute 方法将查询结果作为对象的 `java.util.List` 返回。

返回的对象类型必须保证有适当的 getter 方法可用于处理那些定义了该查询的列表列中显示的所有属性。要确保返回的对象类型适合于列表定义，可以将 Faces 配置文件中定义的 BPCListHandler 实例的 type 属性值设置为所返回对象的标准类名。可以在查询实现的 getType 调用中返回此名称。在运行时期间，列表处理程序将检查对象类型是否与定义一致。

要将错误消息映射到列表中的特定条目，查询返回的对象必须实现具有以下特征符的方法：`public Object getID()`。

缺省转换器和标签

查询返回的项必须是 bean，并且其类必须与在 BPCListHandler 类或 com.ibm.bpc.clientcore.Query 接口的定义中指定为类型的类匹配。此外，List 组件会检查项类或超类是否实现下列方法：

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

如果已为 bean 定义这些方法，那么 List 组件使用标签作为列表的缺省标签，并且使用 SimpleConverter 作为属性的缺省转换器。您可以使用 `bpe:list` 标记的 **label** 和 **converterID** 属性来覆盖这些设置。有关更多信息，请参阅 SimpleConverter 接口和 ColumnTag 类的 Javadoc。

特定于用户的时区信息

JavaServer Faces（JSF）组件提供了实用程序来处理 List 组件中特定于用户的时区信息。

BPCListHandler 类使用 com.ibm.bpc.clientcore.util.User 接口来获取有关每个用户的时区信息和语言环境信息。List 组件期望将此接口的实现中的 **user** 配置成 JavaServer Faces（JSF）配置文件中的受管 Bean 名称。如果配置文件未包含此条目，那么将返回 WebSphere Process Server 使用的时区。

com.ibm.bpc.clientcore.util.User 接口的定义如下所示：

```
public interface User {

    /**
     * The locale used by the client of the user.
     * @return Locale.
     */
    public Locale getLocale();
   /**
    * The time zone used by the client of the user.
    * @return TimeZone.
    */
    public TimeZone getTimeZone();

   /**
    * The name of the user.
    * @return name of the user.
    */
    public String getName();
}
```

List 组件中的错误处理

使用 List 组件以便在 JSF 应用程序中显示列表时，可以利用 com.ibm.bpe.jsf.handler.BPCListHandler 类提供的错误处理功能。

运行查询或执行命令时出错

如果执行查询期间发生了错误，那么 BPCListHandler 类会对由于访问权不足而导致的错误与其他异常进行区分。要捕获由于访问权不足而导致的错误，查询的 execute 方法所抛出的 ClientException 的 **rootCause** 参数必须是 com.ibm.bpe.api.EngineNotAuthorizedException 或 com.ibm.task.api.NotAuthorizedException 异常。List 组件将显示错误消息，而不是显示查询结果。

如果该错误不是由于访问权不足而导致的，那么 BPCListHandler 类会将该异常对象传递给 JSF 应用程序配置文件中 BPCError 键定义的 com.ibm.bpc.clientcore.util.ErrorBean 接口实现。设置该异常时，将调用错误导航目标。

处理列表中显示的项时出错

BPCListHandler 类实现了 com.ibm.bpe.jsf.handler.ErrorHandler 接口。您可以在 setErrors 方法中使用类型为 java.util.Map 的映射参数来提供有关这些错误的信息。此映射包含标识（作为键）和异常（作为值）。这些标识必须是引起错误的对象的 getID 方法所返回的值。如果设置了该映射，并且任何标识与该列表中显示的任何项相匹配，那么列表处理程序会在该列表中自动添加包含该错误消息的列。

为了避免该列表包含过期的错误消息，请重置错误映射。在下列情况下，该映射将自动重置:

- 调用了 BPCListHandler 类的 refreshList 方法。
- 对 BPCListHandler 类设置了新查询。
- 使用 CommandBar 组件对列表项触发了操作。CommandBar 组件使用此机制作为其中一种错误处理方法。

相关概念

第 135 页的『JSF 组件中的错误处理』

JavaServer Faces（JSF）组件使用预定义的受管 bean BPCError 来处理错误。在触发了错误页的错误情况下，将对错误 Bean 设置此异常。

List 组件: 标记定义

业务流程编排器资源管理器的 List 组件在表中显示一组对象，例如，任务、活动、流程实例、流程模板、工作项和升级。

List 组件由 JSF 组件标记组成: `bpe:list` 和 `bpe:column`。`bpe:column` 标记是 `bpe:list` 标记的子元素。

组件类

com.ibm.bpe.jsf.component.ListComponent

语法示例

```
<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>
```

标记属性

`bpe:list` 标记的主体只能包含 `bpe:column` 标记。呈示该表时，List 组件将对应用程序对象列表进行迭代，并为每个对象呈示所有列。

表 35. bpe:list 属性

属性	必需	描述
buttonStyleClass	否	用于在表脚区域呈示按钮的级联样式表（CSS）样式类。
cellStyleClass	否	用于呈示各个表单元格的 CSS 样式类。
checkbox	否	确定是否呈示用于选择多个项的复选框。此属性的值为 `true` 或 `false`。如果该值已设置为 `true`，那么将呈示复选框列。
headerStyleClass	否	用于呈示表头的 CSS 样式类。
model	是	com.ibm.bpe.jsf.handler.BPCListHandler 类的受管 Bean 的值绑定。
rows	否	每页显示的行数。如果项数超出行数，那么将在表尾显示分页按钮。此属性不支持值表达式。
rowClasses	否	用于呈示表行的 CSS 样式类。
selectAll	否	如果此属性设置为 `true`，那么缺省情况下会选择列表中的所有项。
styleClass	否	用于呈示整个表（包含标题、行和分页按钮）的 CSS 样式类。

表 36. bpe:column 属性

属性	必需	描述
action	否	如果指定了此属性，那么将在列中呈示链接。单击此链接后，将触发 JavaServer Faces 操作方法或 Faces 导航目标。JavaServer Faces 操作方法具有下列特征符：String method()。
converterID	否	用于转换属性值的 Faces 转换器标识。如果未设置此属性，那么将使用由模型为此属性提供的任何 Faces 转换器标识。
label	否	列头或表头行单元格用作标签的字面值或值绑定表达式。如果未设置此属性，那么将使用由模型为此属性提供的任何标签。
name	是	此列中显示的属性的名称。

将 Details 组件添加至 JSF 应用程序

使用业务流程编排器资源管理器的 Details 组件来显示任务、工作项、活动、流程实例和流程模板的属性。

过程

1. 在 JavaServer Pages（JSP）文件中添加 Details 组件。

   在 `<h:form>` 标记中添加 `bpe:details` 标记。`bpe:details` 标记必须包含 **model** 属性。可以使用 `bpe:property` 标记来对 Details 组件添加属性。

   下列示例说明如何添加 Details 组件以显示任务实例的某些属性。

   ```
   <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>
   ```

   model 属性引用了名为 TaskInstanceDetails 的受管 Bean。此 Bean 提供了 Java 对象的属性。

2. 配置 `bpe:details` 标记中引用的受管 Bean。

   对于 Details 组件，这个受管 Bean 必须是 com.ibm.bpe.jsf.handler.BPCDetailsHandler 类的实例。此处理程序类包装 Java 对象并向“Details”组件公布其公用属性。

   下列示例说明如何在配置文件中添加 TaskInstanceDetails 受管 Bean。

```
<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>
```

该示例显示 TaskInstanceDetails Bean 具有可配置的 type 属性。该 type 属性的值指定了 Bean 类（com.ibm.task.clientmodel.bean.TaskInstanceBean），该类的属性将显示在所示详细信息行中。该 bean 类可以是任何 JavaBeans 类。如果该 bean 提供缺省转换器和属性标签，那么会使用该转换器和标签来呈示，所采用方式与用于 List 组件的方式相同。

结果

现在，JSF 应用程序包含一个 JavaServer 页面，该页面将显示所指定的对象的详细信息，例如任务实例的详细信息。

相关参考

『Details 组件: 标记定义』
业务流程编排器资源管理器的 Details 组件显示任务、工作项、活动、流程实例和流程模板的属性。

Details 组件: 标记定义

业务流程编排器资源管理器的 Details 组件显示任务、工作项、活动、流程实例和流程模板的属性。

Details 组件由 JSF 组件标记组成: `bpe:details` 和 `bpe:property`。`bpe:property` 标记是 `bpe:details` 标记的子元素。

组件类

com.ibm.bpe.jsf.component.DetailsComponent

语法示例

```
<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>
```

标记属性

使用 `bpe:property` 标记来指定要显示的属性子集以及这些属性的显示顺序。如果 details 标记未包含任何属性标记，那么它将呈示模型对象的所有可用属性。

表 37. bpe:details 属性

属性	必需	描述
columnClasses	否	用于呈示列的一组由逗号分隔的级联样式表（CSS）样式类。
id	否	组件的 JavaServer Faces 标识。
model	是	com.ibm.bpe.jsf.handler.BPCDetailsHandler 类的受管 Bean 的值绑定。
rowClasses	否	用于呈示行的一组由逗号分隔的 CSS 样式类。
styleClass	否	用于呈示 HTML 元素的 CSS 类。

表 38. bpe:property 属性

属性	必需	描述
converterID	否	用来在 JavaServer Faces（JSF）配置文件中注册转换器的标识。
label	否	属性的标签。如果未设置此属性，那么缺省标签由客户机模型类提供。
name	是	要显示的属性的名称。此名称必须与相应客户机模型类中定义的 named 属性相对应。

将 CommandBar 组件添加至 JSF 应用程序

使用业务流程编排器资源管理器的 CommandBar 组件来显示带有按钮的栏。这些按钮代表作用于对象详细视图或列表中的所选对象的命令。

关于此任务

当用户单击用户界面中的按钮时，将对所选对象运行相应的命令。您可以在 JSF 应用程序中添加和扩展 CommandBar 组件。

过程

1. 在 JavaServer Pages（JSP）文件中添加 CommandBar 组件。

   在 `<h:form>` 标记中添加 `bpe:commandbar` 标记。`bpe:commandbar` 标记必须包含 model 属性。

   下列示例说明如何添加为任务实例列表提供刷新和声明命令的 CommandBar 组件。

```
<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>
```

model 属性引用了受管 Bean。这个 Bean 必须实现 ItemProvider 接口并提供所选的 Java 对象。CommandBar 组件通常与同一个 JSP 文件中的 List 组件或 Details 组件配合使用。一般来说，该标记中指定的 model 与同一页面中的 List 组件或 Details 组件中指定的 model 相同。因此，对于 List 组件，命令将对该列表中的所选项执行操作。

在本示例中，**model** 属性引用了名为 TaskInstanceList 的受管 Bean。这个 Bean 在任务实例列表中提供所选的对象。该 Bean 必须实现 ItemProvider 接口。此接口是由 BPCListHandler 类和 BPCDetailsHandler 类实现的。

2. 可选: 配置 bpe:commandbar 标记中引用的受管 Bean。

   如果 CommandBar 组件的 **model** 属性引用了已配置的受管 Bean，例如，列表或详细信息处理程序的受管 Bean，那么不需要执行进一步的配置工作。如果更改了这些处理程序中任何一个的配置，或者使用了另一个受管 Bean，那么必须在 JSF 配置文件中添加实现了 ItemProvider 接口的受管 Bean。

3. 在 JSF 应用程序中添加实现定制命令的代码。

   以下代码段说明如何编写实现 Command 接口的命令类。JSP 文件中的 bpe:command 标记引用了此命令类（MyClaimCommand）。

```
public class MyClaimCommand implements Command {

  public String execute(List selectedObjects) throws ClientException {
     if( selectedObjects != null && selectedObjects.size() > 0 ) {
    try {
         // Determine HumanTaskManagerService from an HTMConnection bean.
         // Configure the bean in the faces-config.xml for easy access
         // in the JSF application.
         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 while iterating or claiming task instance.
                  // Ignore for better understanding of the sample.
           }
         }
       }
       catch( Exception e ) {
         ;  // Configuration or communication error.
           // Ignore for better understanding of the sample
       }
     }
     return null;
  }

 // Default implementations
```

```
 public boolean isMultiSelectEnabled() { return false; }
 public boolean[] isApplicable(List itemsOnList) {return null; }
 public void setContext(Object targetModel) {; // Not used here }
}
```

命令是按以下方式处理的:

a. 当用户单击命令栏中的相应按钮时，就调用了命令。CommandBar 组件从 **model** 属性中指定的项提供者检索所选项，并将所选对象列表传递给 commandClass 实例的 execute 方法。

b. **commandClass** 属性引用实现了 Command 接口的定制命令实现。这意味着该命令必须实现 public String execute(List selectedObjects) throws ClientException 方法。此命令返回的结果将用于确定 JSF 应用程序的下一个导航规则。

c. 该命令完成后，CommandBar 组件将对 **action** 属性进行求值。**action** 属性可以是静态字符串，也可以是具有 `public String Method()` 特征符的 JSF 操作方法的方法绑定。请使用 **action** 属性来覆盖命令类的结果或者显式地指定导航规则的结果。如果该命令生成除 ErrorsInCommandException 以外的异常，那么不会处理 **action** 属性。

d. 如果未对 **commandClass** 属性指定命令类，那么将立即调用该操作。例如，对于此示例中的刷新命令，将调用 JSF 值表达式 `#{TaskInstanceList.refreshList}`，而不是调用命令。

结果

现在，JSF 应用程序包含一个 JavaServer 页面，该页面实现了定制的命令栏。

相关参考

第 147 页的『CommandBar 组件: 标记定义』
业务流程编排器资源管理器的 CommandBar 组件显示带有按钮的栏。这些按钮作用于详细视图中的对象或列表中的所选对象。

命令的处理方式

使用 CommandBar 组件将操作按钮添加至应用程序中。此组件为操作在用户界面中创建按钮并处理单击按钮时创建的事件。

这些按钮触发的功能对 com.ibm.bpe.jsf.handler.ItemProvider 接口返回的对象（例如，BPCListHandler 类或 BPCDetailsHandler 类）执行操作。CommandBar 组件使用 `bpe:commandbar` 标记中 **model** 属性值定义的项提供者。

当单击应用程序用户界面上命令栏部分中的按钮时，CommandBar 组件将按以下方式处理相关联的事件。

1. CommandBar 组件确定 com.ibm.bpc.clientcore.Command 接口的实现，该接口是为生成该事件的按钮指定的。
2. 如果与 CommandBar 组件相关联的模型实现 com.ibm.bpe.jsf.handler.ErrorHandler 接口，那么调用 clearErrorMap 方法以从先前事件中除去错误消息。
3. 调用 ItemProvider 接口的 getSelectedItems 方法。将返回的项列表传递给该命令的 execute 方法并调用该命令。
4. CommandBar 组件确定 JavaServer Faces（JSF）导航目标。如果在 `bpe:commandbar` 标记中未指定 **action** 属性，那么 execute 方法的返回值会指定导航目标。如果

action 属性设置为 JSF 方法绑定，那么将该方法返回的字符串解释为导航目标。该 **action** 属性也可以指定显式的导航目标。

CommandBar 组件: 标记定义

业务流程编排器资源管理器的 CommandBar 组件显示带有按钮的栏。这些按钮作用于详细视图中的对象或列表中的所选对象。

CommandBar 组件由 JSF 组件标记组成: `bpe:commandbar` 和 `bpe:command`。`bpe:command` 标记是 `bpe:commandbar` 标记的子元素。

组件类

com.ibm.bpe.jsf.component.CommandBarComponent

语法示例

```
<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>
```

标记属性

表 39. bpe:commandbar 属性

属性	必需	描述
buttonStyleClass	否	用于呈示命令栏中的按钮的级联样式表（CSS）样式类。
id	否	组件的 JavaServer Faces 标识。
model	是	实现 ItemProvider 接口的受管 Bean 的值绑定表达式。这个受管 Bean 通常是 CommandBar 组件所在 JavaServer Pages（JSP）文件中的 List 组件或 Details 组件所使用的 com.ibm.bpe.jsf.handler.BPCListHandler 类或 com.ibm.bpe.jsf.handler.BPCDetailsHandler 类。
styleClass	否	用于呈示命令栏的 CSS 样式类。

表 40. bpe:command 属性

属性	必需	描述
action	否	将由命令按钮触发的 JavaServer Faces 操作方法或 Faces 导航目标。由操作返回的导航目标覆盖其他所有导航规则。在未抛出异常或命令抛出 ErrorsInCommandException 异常时，将调用该操作。

表 40. bpe:command 属性 (续)

属性	必需	描述
commandClass	否	命令类的名称。如果选择命令按钮，那么将由 CommandBar 组件创建并运行类的实例。
commandID	是	命令的标识。
context	否	为使用 **commandClass** 属性指定的命令提供上下文的对象。当第一次访问命令时，将会检索上下文对象。
immediate	否	指定何时触发命令。如果此属性值为 `true`，那么将在处理页面输入之前触发命令。缺省值是 `false`。
label	是	在命令栏中呈示的按钮标签。
rendered	否	确定是否呈示按钮。该属性值可以是 Boolean 值或值表达式。
styleClass	否	用于呈示按钮的 CSS 样式类。此样式将覆盖为命令栏定义的按钮样式。

将 Message 组件添加至 JSF 应用程序

使用业务流程编排器资源管理器的 Message 组件在 JavaServer Faces（JSF）应用程序中呈示数据对象和基本类型。

关于此任务

如果消息类型是基本类型，那么将呈示标签和输入字段。如果消息类型是数据对象，那么该组件就会对该对象进行遍历并呈示该对象中的元素。

过程

1. 在 JavaServer Pages（JSP）文件中添加 Message 组件。

   在 `<h:form>` 标记中添加 `bpe:form` 标记。 `bpe:form` 标记必须包含 model 属性。

   下列示例说明如何添加 Message 组件。

```
<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>
```

   Message 组件的 **model** 属性引用了 com.ibm.bpc.clientcore.MessageWrapper 对象。此包装器对象包装了服务数据对象（SDO）或 Java 基本类型（例如 `int` 或 `boolean`）。在本示例中，消息是由名为 MyHandler 的受管 Bean 的属性提供的。

2. 配置 `bpe:form` 标记中引用的受管 Bean。

   以下示例说明如何在配置文件中添加 MyHandler 受管 Bean。

```
<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. 在 JSF 应用程序中添加定制代码。

   以下示例说明如何实现输入消息和输出消息。

```
public class MyHandler implements ItemListener {

  private TaskInstanceBean taskBean;
  private MessageWrapper inputMessage, outputMessage

  /* Listener method, e.g. when a task instance was selected in a list handler.
   * Ensure that the handler is registered in the faces-config.xml or manually.
   */
  public void itemChanged(Object item) {
    if( item instanceof TaskInstanceBean ) {
        taskBean = (TaskInstanceBean) item ;
    }
  }

  /* Get the input message wrapper
   */
  public MessageWrapper getInputMessage() {
    try{
        inputMessage = taskBean.getInputMessageWrapper() ;
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return inputMessage;
  }

  /* Get the output message wrapper
   */
  public MessageWrapper getOutputMessage() {
    // Retrieve the message from the bean. If there is no message, create
    // one if the task has been claimed by the user. Ensure that only
    // potential owners or owners can manipulate the output message.
    try{
        outputMessage = taskBean.getOutputMessageWrapper();
        if( outputMessage == null
         && taskBean.getState() == TaskInstanceBean.STATE_CLAIMED ) {
           HumanTaskManagerService htm = getHumanTaskManagerService();
           outputMessage = new MessageWrapperImpl();
           outputMessage.setMessage(
              htm.createOutputMessage( taskBean.getID() ).getObject()
           );
         }
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return outputMessage
  }
}
```

MyHandler 受管 Bean 实现了 com.ibm.jsf.handler.ItemListener 接口，因此，它可以将其本身注册为列表处理程序的项侦听器。当用户单击列表中的项时，MyHandler Bean 在它的 itemChanged(Object item) 方法中将接收到有关所选项的通知。处理程序将检查项类型，然后存储对相关 TaskInstanceBean 对象的引用。要使用此接口，请在 faces-config.xml 文件中添加相应列表处理程序中的 itemListener 列表的条目。

MyHandler Bean 提供了 getInputMessage 和 getOutputMessage 方法。这两个方法都返回 MessageWrapper 对象。这些方法把调用委托给所引用的任务实例 Bean 进行。例如，如果由于未设置消息而导致任务实例 Bean 返回空，该处理程序就会创建并存储新的空消息。Message 组件将显示 MyHandler Bean 提供的消息。

结果

现在，JSF 应用程序包含一个 JavaServer 页面，该页面可以呈示数据对象和基本类型。

相关参考

『Message 组件: 标记定义』
业务流程编排器资源管理器的 Message 组件在 JavaServer Faces（JSF）应用程序中呈示诸如整数和字符串之类的 commonj.sdo.DataObject 对象与基本类型。

Message 组件: 标记定义

业务流程编排器资源管理器的 Message 组件在 JavaServer Faces（JSF）应用程序中呈示诸如整数和字符串之类的 commonj.sdo.DataObject 对象与基本类型。

Message 组件由 JSF 组件标记组成: bpe:form。

组件类

com.ibm.bpe.jsf.component.MessageComponent

语法示例

```
<bpe:form model="#{TaskInstanceDetailsBean.inputMessageWrapper}"
          simplification="true" readOnly="true"
          styleClass4table="messageData"
          styleClass4output="messageDataOutput">
</bpe:form>
```

标记属性

表 41. bpe:form 属性

属性	必需	描述
id	否	组件的 JavaServer Faces 标识。
model	是	引用了 commonj.sdo.DataObject 对象或 com.ibm.bpc.clientcore.MessageWrapper 对象的值绑定表达式。
readOnly	否	如果此属性设置为 true，那么将呈示只读表单。缺省情况下，此属性设置为 false。
simplification	否	如果此属性设置 true，那么将显示包含简单类型且基数为 0 或 1 的属性。缺省情况下，此属性设置为 true。
style4validinput	否	用于呈示有效输入的级联样式表（CSS）样式。

表 41. bpe:form 属性 (续)

属性	必需	描述
style4invalidinput	否	用于呈示无效输入的 CSS 样式。
styleClass4invalidInput	否	用于呈示无效输入的 CSS 样式类名。
styleClass4output	否	用于呈示输出元素的 CSS 样式类名。
styleClass4table	否	用于呈示消息组件所呈示的表的 CSS 表样式类名。
styleClass4validInput	否	用于呈示有效输入的 CSS 样式类名。

为任务和流程消息开发 JSP 页面

业务流程编排器资源管理器接口提供缺省输入与输出格式来显示和输入业务数据。您可以使用 JSP 页面来提供定制输入和输出格式。

关于此任务

要在 Web 客户机中包括用户定义的 JavaServer 页面（JSP），当在 WebSphere Integration Developer 中对人员任务建模时，您必须指定这些页面。例如，您可以为特定任务及其输入和输出消息以及特定用户角色或所有用户角色提供 JSP 页面。在运行时期间，用户界面中包含了用户定义的 JSP 页面，以显示输出数据和收集输入数据。

定制格式不是自包含 Web 页面；它们是业务流程编排器资源管理器采用 HTML 格式嵌入的 HTML 片段，例如，消息的所有标签和输入字段的片段。

当单击包含定制格式的页面上的按钮时，将提交该输入并在业务流程编排器资源管理器中对其进行验证。该验证基于所提供的属性类型和浏览器中使用的语言环境。如果无法验证输入，那么将再次显示相同的页面，并且在 messageValidationErrors 请求属性中提供有关验证错误的信息。该信息作为映射来提供，它将无效属性的 XML 路径表达式（XPath）映射到已发生的验证异常。

要将定制格式添加到业务流程编排器资源管理器，使用 WebSphere Integration Developer 来完成下列步骤。

过程

1. 创建定制格式。

   在 Web 接口中使用的输入和输出格式的用户定义的 JSP 页面需要访问消息数据。使用采用 JSP 或 JSP 执行语言的 Java 片段来访问消息数据。该格式的数据可通过请求上下文获得。

2. 将 JSP 页面分配给任务。

   在人员任务编辑器中打开人员任务。在客户机设置中，指定用户定义的 JSP 页面的位置和定制格式适用的角色（例如管理员）。业务流程编排器资源管理器的客户机设置存储在任务模板中。在运行时期间，将使用任务模板检索这些设置。

3. 将用户定义的 JSP 页面打包在 Web 归档文件（WAR 文件）中。

您可以将 WAR 文件纳入带有包含任务的模块的企业归档中，或者单独部署 WAR 文件。如果已单独部署 JSP，那么使 JSP 在部署了业务流程编排器资源管理器或定制客户机的服务器上可用。

如果您要对流程和任务消息使用定制 JSP，那么必须将用于部署 JSP 的 Web 模块映射到定制 JSF 客户机映射到的同一服务器上。

结果

定制格式在运行时会呈示在业务流程编排器资源管理器中。

用户定义的 JSP 片段

用户定义的 JavaServer 页（JSP）片段已嵌入在 HTML 格式标记中。在运行时期间，业务流程编排器资源管理器将这些片段包含在呈示的页面中。

输入消息的用户定义的 JSP 片段已嵌入在输出消息的 JSP 片段之前。

```
<html....>
   ...
   <form...>
     Input JSP (display task input message)

     Output JSP (display task output message)

   </form>
   ...
</html>
```

由于用户定义的 JSP 片段已嵌入在 HTML 格式标记中，所以您可以添加输入元素。输入元素的名称必须与数据元素的 XML 路径语言（XPath）表达式匹配。对于使用提供的前缀值来为输入元素名称添加前缀，这是很重要的:

```
<input id="address"
       type="text"
       name="${prefix}/selectPromotionalGiftResponse/address"
       value="${messageMap['/selectPromotionalGiftResponse/address"]}
       size="60"
       align="left" />
```

前缀值作为请求属性提供。该属性确保输入名称在封闭格式中是唯一的。前缀由业务流程编排器资源管理器生成，您不应该对其进行更改:

```
String prefix = (String)request.getAttribute("prefix");
```

仅当可以在给定上下文中编辑消息时，才能设置前缀元素。视人员任务的状态而定，可采用不同方法显示输出数据。例如，如果任务处于已声明状态，那么您可以修改输出数据。然而，如果任务处于已完成状态，那么您只能显示数据。在 JSP 片段中，可以测试前缀元素是否存在以及相应地呈示消息。下列 JSTL 语句说明您测试前缀元素是否已设置的可能方法。

```
...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<c:choose>
  <c:when test="${not empty prefix}">
    <!--Read/write mode-->
  </c:when>
```

```
  <c:otherwise>
    <!--Read-only mode-->
  </c:otherwise>
</c:choose>
```

创建插件以定制人员任务功能

业务流程编排器为处理人员任务期间发生的事件提供事件处理基础结构。同时还提供了插件点，以便您可以使该功能适应您的需要。可以使用服务提供者接口（SPI）来创建用于处理事件和人员查询的定制插件。

关于此任务

可以为人员任务 API 事件和升级通知事件创建插件。还可以创建用于处理从人员分析返回的结果的插件。例如，在峰值期间，可以向结果列表添加用户，以便帮助平衡工作负载。

您可以为全局级别的所有任务、应用程序组件中的任务、与任务模板相关联的所有任务或单个任务实例在不同级别上注册插件。

创建 API 事件处理程序

在 API 方法处理人员任务时将发生 API 事件。使用 API 事件处理程序插件服务提供者接口（SPI）来创建插件，以处理 API 所发送的任务事件或具有同等 API 事件的内部事件。

关于此任务

完成下列步骤以创建 API 事件处理程序。

过程

1. 编写可实现 APIEventHandlerPlugin2 接口或扩展 APIEventHandler 实现类的类。 此类可以调用其他类的方法。
   - 如果使用 APIEventHandlerPlugin2 接口，那么必须实现 APIEventHandlerPlugin2 接口和 APIEventHandlerPlugin 接口的所有方法。
   - 如果扩展 SPI 实现类，那么覆盖所需的方法。

   此类在 Java 2 Enterprise Edition（J2EE）Enterprise JavaBeans（EJB）应用程序的上下文中运行。确保此类及其辅助控件类遵循 EJB 规范。

   提示: 如果要从此类调用 HumanTaskManagerService 接口，请勿调用更新已生成事件的任务的方法。此操作会导致数据库死锁。
2. 将插件类及其辅助控件类打包至 JAR 文件。

   如果有多个 J2EE 应用程序使用辅助控件类，那么可以将这些类打包在已注册为共享库的单独 JAR 文件中。
3. 在 JAR 文件的 META-INF/services/ 目录中为插件创建服务提供者配置文件。

   配置文件提供用于识别和装入插件的机制。此文件符合 Java 2 服务提供者接口规范。

a. 创建名为 com.ibm.task.spi.*plug-in_name*APIEventHandlerPlugin 的文件，其中 *plug-in_name* 是插件的名称。

   例如，如果插件名为 Customer，并且它实现 com.ibm.task.spi.APIEventHandlerPlugin 接口，那么配置文件的名称是 com.ibm.task.spi.**Customer**APIEventHandlerPlugin。

b. 在该文件中，第一个既非注释也非空白的行应指定在步骤 1 中创建的插件类的标准名称。

   例如，如果插件类名为 MyAPIEventHandler，并且它位于 com.customer.plugins 包中，那么该配置文件的第一行必须包含下列条目：com.customer.plugins.MyAPIEventHandler。

结果

您具有包含用于处理 API 事件的插件的可安装 JAR 文件，以及可用于装入该插件的服务提供者配置文件。

提示: 仅具有一个 eventHandlerName 属性可用于注册 API 事件处理程序和通知事件处理程序。如果要同时使用 API 事件处理程序和通知事件处理程序，那么这两个插件实现必须同名，例如，将 Customer 用作 SPI 实现的事件处理程序名。

您可以使用一个类或两个单独的类来实现两个插件。在两种情况下，您需要在 JAR 文件的 META-INF/services/ 目录中创建两个文件，例如，com.ibm.task.spi.CustomerNotificationEventHandlerPlugin 和 com.ibm.task.spi.CustomerAPIEventHandlerPlugin。

将插件实现和辅助控件类打包在单个 JAR 文件中。

下一步做什么?

现在，您需要安装和注册插件，以便在运行时可供人员任务容器使用。可以对 API 事件处理程序注册任务实例、任务模板或应用程序组件。

API 事件处理程序

在修改人员任务或人员任务状态变化时，将发生 API 事件。要处理这些 API 事件，在修改任务之前（事件前方法）和 API 调用返回之前（事件后方法），可直接调用事件处理程序。

如果事件前方法抛出 ApplicationVetoException 异常，那么不执行 API 操作，该异常返回至 API 调用者，并且与事件相关联的事务被回滚。如果内部事件已触发该事件前方法，并且抛出了 ApplicationVetoException 异常，那么将不执行诸如自动声明之类的内部事件，但异常不返回至客户机应用程序。在此情况下，会将参考消息写入 SystemOut.log 文件。如果 API 方法在处理期间抛出异常，那么会捕获该异常并将其传递至事件后方法。在事件后方法返回之后，会再次将该异常传递至调用者。

下列规则适用于事件前方法:

- 事件前方法接收关联 API 方法或内部事件的参数。
- 事件前方法可以抛出 ApplicationVetoException 异常以阻止继续进行处理。

下列规则适用于事件后方法:

- 事件后方法接收已提供给 API 调用的参数以及返回值。如果 API 方法实现抛出了异常，那么事件后方法也会接收到异常。
- 事件后方法无法修改返回值。
- 事件后方法无法抛出异常；运行时异常将被记录，但会被忽略。

要实现 API 事件处理程序，可以使用将扩展 APIEventHandlerPlugin 接口的 APIEventHandlerPlugin2 接口，或者扩展缺省的 com.ibm.task.spi.APIEventHandler SPI 实现类。如果事件处理程序继承自缺省实现类，那么会始终实现最新版本的 SPI。如果升级至更新版本的业务流程编排器，那么在希望利用新的 SPI 方法时，只需进行较少的更改。

如果具有通知事件处理程序和 API 事件处理程序，那么这两个处理程序必须具有相同的名称，其原因是您只能注册一个事件处理程序名称。

创建通知事件处理程序

当升级人员任务时，将生成通知事件。业务流程编排器提供用于处理升级的功能，例如，创建升级工作项或发送电子邮件。您可以创建通知事件处理程序，以定制处理升级的方式。

关于此任务

要实现通知事件处理程序，可以使用 NotificationEventHandlerPlugin 接口，也可以扩展缺省的 com.ibm.task.spi.NotificationEventHandler 服务提供者接口（SPI）实现类。

完成下列步骤以创建通知事件处理程序。

过程

1. 编写用于实现 NotificationEventHandlerPlugin 接口或扩展 NotificationEventHandler 实现类的类。 此类可以调用其他类的方法。

   如果使用 NotificationEventHandlerPlugin 接口，那么必须实现所有接口方法。如果扩展 SPI 实现类，那么覆盖所需的方法。

   此类在 Java 2 Enterprise Edition（J2EE）Enterprise JavaBeans（EJB）应用程序的上下文中运行。确保此类及其辅助控件类遵循 EJB 规范。

   使用 EscalationUser 角色的权限来调用该插件。在配置人员任务容器时，将会定义此角色。

   提示: 如果要从此类调用 HumanTaskManagerService 接口，请勿调用更新已生成事件的任务或升级的方法。此操作会导致数据库死锁。
2. 将插件类及其辅助控件类打包至 JAR 文件。

   如果有多个 J2EE 应用程序使用辅助控件类，那么可以将这些类打包在已注册为共享库的单独 JAR 文件中。
3. 在 JAR 文件的 META-INF/services/ 目录中为插件创建服务提供者配置文件。

   配置文件提供用于识别和装入插件的机制。此文件符合 Java 2 服务提供者接口规范。

a. 创建名为 com.ibm.task.spi.*plug-in_name*NotificationEventHandlerPlugin 的文件，其中 *plug-in_name* 是插件的名称。

   例如，如果插件名为 HelpDeskRequest（事件处理程序名称），并且它实现 com.ibm.task.spi.NotificationEventHandlerPlugin 接口，那么配置文件的名称是 com.ibm.task.spi.**HelpDeskRequest**NotificationEventHandlerPlugin。

b. 在该文件中，第一个既非注释也非空白的行应指定在步骤 1 中创建的插件类的标准名称。

   例如，如果插件类名为 MyEventHandler，并且它位于 com.customer.plugins 包中，那么该配置文件的第一行必须包含下列条目：com.customer.plugins.MyEventHandler。

结果

您具有包含用于处理通知事件的插件的可安装 JAR 文件，以及可用于装入该插件的服务提供者配置文件。可以对 API 事件处理程序注册任务实例、任务模板或应用程序组件。

提示: 仅具有一个 eventHandlerName 属性可用于注册 API 事件处理程序和通知事件处理程序。如果要同时使用 API 事件处理程序和通知事件处理程序，那么这两个插件实现必须同名，例如，将 Customer 用作 SPI 实现的事件处理程序名。

您可以使用一个类或两个单独的类来实现两个插件。在两种情况下，您需要在 JAR 文件的 META-INF/services/ 目录中创建两个文件，例如，com.ibm.task.spi.CustomerNotificationEventHandlerPlugin 和 com.ibm.task.spi.CustomerAPIEventHandlerPlugin。

将插件实现和辅助控件类打包在单个 JAR 文件中。

下一步做什么?

现在，您需要安装和注册插件，以便在运行时可供人员任务容器使用。可以对通知事件处理程序注册任务实例、任务模板或应用程序组件。

创建插件以对人员查询结果进行后处理

人员分析返回一组已分配特定角色的用户，例如，任务的潜在所有者。您可以创建插件以更改人员分析返回的人员查询结果。例如，为了改进工作负载平衡，可以具有从工作负载较高的查询结果中除去用户的插件。

关于此任务

仅可以具有一个后处理插件；这意味着该插件必须处理所有任务的人员查询结果。插件可以添加或除去用户，也可以更改用户或组信息。它还可以更改结果类型，例如，从用户列表更改为组或每个人。

因为插件在人员分析完成之后才会运行，所以已经应用您用于保持机密性或安全性的所有规则。该插件在 HTM_REMOVED_USERS 映射键中接收到有关人员分析期间除去的用户的信息。您必须确保插件使用此上下文信息来保持可能具有的机密性或安全性规则。

要实现对人员查询结果进行后处理，可使用 StaffQueryResultPostProcessorPlugin 接口。该接口具有修改任务查询结果、升级、任务模板和应用程序组件的方法。

完成下列步骤以创建插件对人员查询结果进行后处理。

过程

1. 编写可实现 StaffQueryResultPostProcessorPlugin 接口的类。

   您必须实现所有接口方法。此类可以调用其他类的方法。

   此类在 Java 2 Enterprise Edition（J2EE）Enterprise JavaBeans（EJB）应用程序的上下文中运行。确保此类及其辅助控件类遵循 EJB 规范。

   提示: 如果要从此类调用 HumanTaskManagerService 接口，请勿调用更新已生成事件的任务的方法。此操作会导致数据库死锁。

   下列示例显示如何更改名为 SpecialTask 的任务的编辑器角色。

```
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. 将插件类及其辅助控件类打包至 JAR 文件。

   如果有多个 J2EE 应用程序使用辅助控件类，那么可以将这些类打包在已注册为共享库的单独 JAR 文件中。

3. 在 JAR 文件的 META-INF/services/ 目录中为插件创建服务提供者配置文件。

   配置文件提供用于识别和装入插件的机制。此文件符合 Java 2 服务提供者接口规范。

   a. 创建名为 com.ibm.task.spi.*plug-in_name*StaffQueryResultPostProcessorPlugin 的文件，其中 *plug-in_name* 是插件的名称。

      例如，如果插件名为 MyHandler，并且它实现 com.ibm.task.spi.StaffQueryResultPostProcessorPlugin 接口，那么配置文件的名称是 com.ibm.task.spi.**MyHandler**StaffQueryResultPostProcessorPlugin。

b. 在该文件中，第一个既非注释也非空白的行应指定在步骤 1 中创建的插件类的标准名称。

   例如，如果插件类名为 StaffPostProcessor，并且它位于 com.customer.plugins 包中，那么该配置文件的第一行必须包含下列条目：com.customer.plugins.StaffPostProcessor。 您具有包含用于对人员查询结果进行后处理的插件的可安装 JAR 文件，以及可用于装入该插件的服务提供者配置文件。

4. 安装插件。

   仅可以具有一个后处理插件来处理人员查询结果。您必须将该插件作为共享库进行安装。

5. 注册插件。
   a. 在管理控制台中，转至人员任务管理器的定制属性页面（**应用程序服务器** → ***server_name*** → **人员任务容器** → **定制属性**）。
   b. 添加名称为 **Staff.PostProcessorPlugin** 的定制属性。将该定制属性的值设置为在此示例的步骤 3a 中授予插件的名称 MyHandler。

安装插件

要使用插件，您必须安装该插件，以便任务容器可以对其进行访问。

关于此任务

安装插件的方式取决于该插件是仅供一个 Java 2 Enterprise Edition（J2EE）应用程序使用，还是供多个应用程序使用。

完成下列其中一个步骤以安装插件。

- 安装插件以供一个 J2EE 应用程序使用。

  将插件 JAR 文件添加至应用程序 EAR 文件。在 WebSphere Integration Developer 的部署描述符编辑器中，安装您的插件 JAR 文件，以作为主 Enterprise JavaBeans（EJB）模块的 J2EE 应用程序的项目实用程序 JAR 文件。

- 安装插件以供多个 J2EE 应用程序使用。

  将该 JAR 文件置于 WebSphere Application Server 共享库中，然后将该库与需要访问该插件的应用程序相关联。要使 JAR 文件在 Network Deployment 环境中可用，请在每个服务器上手动分发该 JAR 文件，然后为每个单元安装该共享库一次。

下一步做什么?

现在，您可以注册该插件。

注册插件

可以在任务容器工件层次结构的不同级别上注册插件。例如，为全局级别的所有任务、为应用程序组件的任务、为任务模板相关联的所有任务或为单个任务实例注册插件。

关于此任务

当注册多个插件时，支持作用域限定。这意味着可以使用在诸如任务实例之类任务容器工件层次结构的较低级别上注册的插件，而不使用在诸如任务模板或应用程序组件之类较高级别上注册的插件。所有层次结构级别均支持作用域限定。该任务容器使用在层次结构的最低级别上注册的插件。

您可以采用下列其中一种方法来注册插件。

- 在任务模型中注册插件。

  在 WebSphere Integration Developer 的任务编辑器中，在该任务属性区域的“详细信息”页面上的**事件处理程序名称**字段中指定事件处理程序的名称。
- 为您在运行时期间创建的特别任务或任务模板注册插件。

  使用 TTask 类的 setEventHandlerName 方法来注册该事件处理程序的名称。
- 在运行时期间为任务实例更改已注册的事件处理程序。

  使用 update(Task task) 方法以在运行时期间对任务实例使用不同的事件处理程序。调用者必须具有任务管理员权限来更新此属性。
- 在全局级别上注册插件。

  在管理控制台上的人员任务容器的定制属性页面中，为插件定义定制属性。定制属性的值是插件名称。

第 2 部分 部署应用程序

第 3 章 有关准备和安装模块的概述

安装模块（亦称“部署”）是指在测试环境或生产环境中激活那些模块。本概述对测试和生产环境以及安装模块时需要执行的一些步骤作了简要描述。

注: 在生产环境中安装应用程序的过程与 WebSphere Application Server Network Deployment V6 信息中心中的“开发和部署应用程序”中描述的过程类似。如果您不熟悉那些主题，请首先阅读主题。

在生产环境中安装模块之前，始终应该在测试环境中验证更改。要在测试环境中安装模块，请使用 WebSphere Integration Developer（请参阅 WebSphere Integration Developer 信息中心以了解更多信息）。要在生产环境中安装模块，请使用 WebSphere Process Server。

本主题描述了在准备模块以及在生产环境中安装该模块时需要了解的概念和需要执行的任务。其他主题描述了模块使用的对象所在的文件，这些主题能够帮助您将模块从测试环境移至生产环境。为了确保正确地安装模块，了解这些文件及其内容是十分重要的。

库和 JAR 文件概述

模块通常使用库中的工件。工件和库包含在您部署模块时标识的 Java 归档（JAR）文件中。

在开发模块时，可标识出模块各个部分要使用的某些资源或组件。这些资源或组件可以是您开发该模块时创建的对象，也可以是服务器上已部署的库中的现有对象。本主题描述安装应用程序时需要使用的库和文件。

什么是库?

库包含由 WebSphere Integration Developer 中的多个模块使用的对象或资源。工件可以包含在 JAR、资源归档（RAR）或 Web service 归档（WAR）文件中。这些工件包括:

- 接口或 Web 服务描述符（扩展名为 .wsdl 的文件）
- 业务对象 XML 模式定义（扩展名为 .xsd 的文件）
- 业务对象映射（扩展名为 .map 的文件）
- 关系和角色定义（扩展名为 .rel 和 .rol 的文件）

当模块需要工件时，服务器就会在 EAR 类路径中查找该工件，如果尚未装入该工件，就会将其装入内存。从那以后，直到该工件被替换之前，任何对该工件的请求都将使用该副本。第 164 页的图 5 说明了应用程序如何包含组件和相关的库。

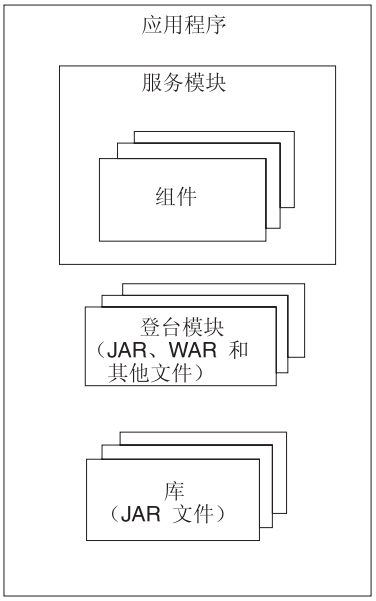


图 5. 模块、组件和库之间的关系

什么是 JAR、RAR 和 WAR 文件?

模块的组件可以包含在许多文件中。Java Platform, Enterprise Edition 规范对这些文件作了详细描述。在 JAR 规范中可以找到有关 JAR 文件的详细信息。

在 WebSphere Process Server 中，JAR 文件还包含应用程序。应用程序是模块的组装版本，它包含该模块使用的任何其他服务组件的支持引用和接口。要全面地安装应用程序，您需要这个 JAR 文件、任何其他归档或库，例如 JAR 文件、Web service 归档（WAR）文件、资源归档（RAR）文件、登台库（Enterprise Java Bean - EJB）JAR 文件，并使用 serviceDeploy 命令来创建可安装的 EAR 文件。

登台模块的命名约定

在库中，对登台模块的名称有一些要求。这些名称对于特定模块来说是唯一的。请对部署应用程序时所需的任何其他模块进行命名，以避免与登台模块名发生冲突。对于名为 *myService* 的模块来说，登台模块名是:

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

注: 如果服务包含 WSDL 端口类型服务，那么 serviceDeploy 命令将仅创建 *myService* Web 登台模块。

使用库时的注意事项

由于每个调用模块都有自己的特定组件副本，正在使用的库在模块之间提供了业务对象一致性以及处理的一致性。为了防止出现不一致和故障情况，确保在所有调用模块之间协调对调用模块所使用的组件和业务对象进行的更改是很重要的。通过执行下列操作，更新调用模块:

1. 将该模块以及库的最新副本复制到生产服务器
2. 使用 serviceDeploy 命令来重建可安装的 EAR 文件
3. 停止运行中的包含调用模块的应用程序，然后重新安装它
4. 重新启动包含调用模块的应用程序

相关参考


serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

EAR 文件概述

在将服务应用程序部署到生产服务器时，EAR 文件十分关键。

企业归档（EAR）文件是一个压缩文件，它包含部署应用程序时所需的库、企业 Bean 和 JAR 文件。

JAR 文件是您从 WebSphere Integration Developer 中导出应用程序模块时创建的。这个 JAR 文件以及任何其他工件库或对象是安装过程的输入。serviceDeploy 命令根据输入文件创建 EAR 文件，那些输入文件包含组件描述以及组成应用程序的 Java 代码。

相关参考


serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

准备部署到服务器

在开发和测试模块之后，必须从测试系统中导出该模块，然后将其导入生产环境以便进行部署。要安装应用程序，您还需要了解导出模块时需要使用的路径以及模块所需的库。

开始之前

在开始执行本任务之前，您应该已在测试服务器上开发和测试模块并解决了各种问题（包括性能问题）。

关于此任务

本任务验证应用程序的所有必需内容是否都可用以及是否都已打包到要传送至生产服务器的正确文件中。

注: 也可以从 WebSphere Integration Developer 中导出企业归档（EAR）文件并将该文件直接安装到 WebSphere Process Server 中。

要点: 如果某个组件中的服务使用数据库，那么请将该应用程序安装到直接连接至数据库的服务器上。

过程

1. 找到要部署的模块的组件所在的文件夹。

   组件文件夹应该名为 *module-name*，此文件夹包含名为 *module*.module 的文件（基本模块）。
2. 验证该模块中包含的所有组件是否都在模块文件夹下面的组件子文件夹中。

   为了便于使用，请将子文件夹命名为类似于 *module*/*component*。
3. 验证组成每个组件的所有文件是否都包含在适当的组件子文件夹中并且名称类似于 *component-file-name*.component。

   组件文件包含模块中每个组件的定义。
4. 验证所有其他组件和工件是否都在需要它们的组件的子文件夹中。

   在此步骤中，您要确保任何对组件所需的工件的引用都可用。组件名不能与 serviceDeploy 命令使用的登台模块名冲突。请参阅登台模块的命名约定。
5. 验证引用文件 *module*.references 是否在步骤 1 使用的模块文件夹中。

   引用文件定义模块中的引用和接口。
6. 验证连线文件 *module*.wires 是否在组件文件夹中。

   连线文件完成模块中引用与接口之间的连接。
7. 验证清单文件 *module*.manifest 是否在组件文件夹中。

   清单列示模块以及所有组成该模块的组件。它还包含类路径语句，该语句使 serviceDeploy 命令可以找到该模块所需的任何其他模块。
8. 创建该模块的压缩文件或 JAR 文件以作为 serviceDeploy 命令的输入，您将使用该命令来准备该模块以便将其安装到生产服务器上。

MyValue 模块在部署前的文件夹结构示例

以下示例说明了模块 MyValueModule 的目录结构，该模块由组件 MyValue、CustomerInfo 和 StockQuote 组成。

```
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
```

将该模块安装到生产系统上，如在生产服务器上安装模块在生产服务器上安装模块所述。

相关参考

 serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

在集群中安装服务应用程序时的注意事项

如果在集群中安装服务应用程序，就会有另外一些要求。在集群中安装任何服务应用程序时，牢记这些注意事项极为重要。

集群提供了可伸缩性来帮助您在服务器之间平衡请求工作负载，并且在一定程度上为应用程序的客户机提供了可用性，从而使处理环境受益匪浅。在集群中安装包含服务的应用程序之前，请考虑下列问题:

- 应用程序用户是否需要集群所提供的处理能力和可用性？

  如果是，那么集群就是正确的解决方案。集群将提高应用程序的可用性和容量。
- 是否为服务应用程序正确地准备了集群？

  在安装和启动第一个包含服务的应用程序之前，必须正确地配置集群。如果未正确地配置集群，应用程序就无法正确地处理请求。
- 集群是否有备份？

  如果是，那么还必须在备份集群中安装应用程序。

第 4 章 在生产服务器上安装模块

本主题描述从测试服务器上获取应用程序并将其部署到生产环境时需要执行的步骤。

开始之前

在将服务应用程序部署到生产服务器之前，请在测试服务器上组装并测试该应用程序。进行测试后，请导出相关文件（如《开发和部署模块》PDF 中的『准备部署到服务器』所述），然后将那些文件传送到生产系统以便进行部署。请参阅 WebSphere Integration Developer 和 WebSphere Application Server Network Deployment 的信息中心以了解更多信息。

过程

1. 将该模块和其他文件复制到生产服务器。

   将应用程序所需的模块和资源（EAR、JAR、RAR 和 WAR 文件）移至生产环境。
2. 运行 serviceDeploy 命令以创建可安装的 EAR 文件。

   此步骤向服务器定义该模块，从而准备将该应用程序安装到生产环境中。
   a. 找到包含所要部署的模块的 JAR 文件。
   b. 使用上一步骤中找到的 JAR 文件作为输入，发出命令。
3. 安装步骤 2 所生成的 EAR 文件。 应用程序的安装方式取决于是在独立服务器上还是在单元中的服务器上安装该应用程序。

   注: 可以使用管理控制台或脚本来安装应用程序。请参阅 WebSphere Application Server 信息中心以了解其他信息。
4. 保存配置。 现在，已将该模块安装成应用程序。
5. 启动该应用程序。

结果

现在，该应用程序已处于活动状态，可使用该模块接受并处理工作了。

下一步做什么?

监视该应用程序以确保服务器正确地处理请求。

相关参考


serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

使用 serviceDeploy 来创建可安装的 EAR 文件

要在生产环境中安装应用程序，请使用已复制到生产服务器的文件来创建可安装的 EAR 文件。

开始之前

在开始执行本任务前，您必须有包含要部署到服务器的模块和服务的 JAR 文件。有关更多信息，请参阅准备部署到服务器。

关于此任务

serviceDeploy 命令将使用 JAR 文件以及任何其他从属 EAR、JAR、RAR、WAR 和 ZIP 文件并构建可以安装到服务器上的 EAR 文件。

过程

1. 找到包含所要部署的模块的 JAR 文件。
2. 使用上一步骤中找到的 JAR 文件作为输入，发出命令。

   此步骤将创建一个 EAR 文件。

   注: 在管理控制台中执行下列步骤。
3. 在服务器的管理控制台中选择要安装的 EAR 文件。
4. 单击**保存**以安装该 EAR 文件。

相关参考

serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

使用 Apache Ant 任务来部署应用程序

本主题描述如何使用 Apache™ Ant 任务来自动地将应用程序部署到 WebSphere Process Server。通过使用 Apache Ant 任务，您可以定义多个应用程序的部署方式并以无人照管方式在服务器上运行这些应用程序。

开始之前

本任务假定:
- 要部署的应用程序已开发并测试完毕。
- 这些应用程序要安装在相同的服务器上。
- 您了解一些有关 Apache Ant 任务的知识。
- 您了解部署过程。

WebSphere Integration Developer 信息中心提供了有关开发和测试应用程序的信息。

WebSphere Application Server Network Deployment 信息中心的“参考”部分包含有关应用程序编程接口的章节。com.ibm.websphere.ant.tasks 包对 Apache Ant 任务作了描述。在本主题中，您将使用 ServiceDeploy 和 InstallApplication 任务。

关于此任务

如果需要同时安装多个应用程序，请在部署前开发 Apache Ant 任务。Apache Ant 任务能够在服务器上部署和安装应用程序，而不要求您参与该过程。

过程

1. 确定要部署的应用程序。
2. 为每个应用程序创建一个 JAR 文件。
3. 将 JAR 文件复制到目标服务器。
4. 创建一个 Apache Ant 任务以运行 ServiceDeploy 命令，从而为每个服务器创建 EAR 文件。
5. 创建一个 Apache Ant 任务，以便在合适的服务器上对步骤 4 创建的每个 EAR 文件运行 InstallApplication 命令。
6. 运行 ServiceDeploy Apache Ant 任务，以便为应用程序创建 EAR 文件。
7. 运行 InstallApplication Apache Ant 任务，以便安装步骤 6 创建的 EAR 文件。

结果

应用程序已正确地部署在目标服务器上。

以无人照管方式部署应用程序的示例

本示例介绍了 myBuildScript.xml 文件中包含的 Apache Ant 任务。

```
<?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>
```

以下语句说明如何调用这个 Apache Ant 任务。

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

提示: 通过在该文件中添加其他 project 语句，就能够以无人照管方式部署多个应用程序。

下一步做什么?

使用管理控制台来验证新安装的应用程序能否正确地启动以及能否正确地处理工作流程。

相关参考

serviceDeploy 命令

使用 serviceDeploy 命令将符合程序包服务组件体系结构（SCA）标准的模块封装为可安装在服务器上的 Java 应用程序。当通过 wsadmin 执行批量安装时，该命令很有用。

第 5 章 安装业务流程和人员任务应用程序

可以将包含业务流程和/或人员任务的服务组件体系结构（SCA）模块分布到部署目标。部署目标可以是服务器或集群。

开始之前

验证是否已经为每个要安装应用程序的应用程序服务器或集群安装并配置了业务流程管理器和/或人员任务管理器。

关于此任务

例如，可以从管理控制台、从命令行或通过运行管理脚本来安装业务流程和人员任务应用程序。

结果

在安装业务流程或人员任务应用程序后，所有业务流程模板和人员任务模板都已进入启动状态。您可以通过这些模板来创建流程实例和任务实例。

下一步做什么?

在可以创建流程实例或任务实例之前，必须启动该应用程序。

相关概念

第 174 页的『部署业务流程和人员任务』
当 WebSphere Integration Developer 或服务部署为流程或任务生成部署代码时，每个流程组件或任务组件均被映射到一个会话企业 bean。所有部署代码都将打包到企业应用程序（EAR）文件中。此外，对于每个流程，在安装企业应用程序期间，将生成在此流程中表示 Java 代码的 Java 类并将其嵌入在 EAR 文件中。必须将所要部署的模型的每个新版本都打包到新的企业应用程序中。

『如何将业务流程和人员任务应用程序安装在 Network Deployment 环境中』
当流程模板或人员任务模板安装在 Network Deployment 环境时，应用程序安装将自动执行下列操作。

如何将业务流程和人员任务应用程序安装在 Network Deployment 环境中

当流程模板或人员任务模板安装在 Network Deployment 环境时，应用程序安装将自动执行下列操作。

各个阶段以异步方式安装应用程序。每个阶段均必须成功完成，然后才会开始下一阶段。

1. 应用程序安装在 Deployment Manager 上开始进行。

   在此阶段期间，业务流程模板和人员任务模板已在 WebSphere 配置存储库中配置。此外，还将验证应用程序。如果发生错误，那么将在 System.out 文件和 System.err 文件中报告这些错误，或在 Deployment Manager 上将其作为 FFDC 条目报告。
2. 应用程序安装在 Node Agent 上继续进行。

在此阶段期间，将触发一个应用程序服务器实例上的应用程序安装。此应用程序服务器实例是部署目标的一部分或者就是部署目标。如果部署目标是带有多个集群成员的集群，那么将从此集群的集群成员中任意选择服务器实例。如果在此阶段期间发生错误，那么将在 SystemOut.log 文件和 SystemErr.log 文件中报告这些错误，或在 Node Agent 上将其作为 FFDC 条目报告。

3. 应用程序在服务器实例上运行。

   在此阶段期间，流程模板和人员模板部署到部署目标上的业务流程编排器数据库。如果发生错误，那么将在 System.out 文件和 SystemErr.log 文件中报告这些错误，或者在此服务器实例上将其作为 FFDC 条目报告。

相关任务

第 173 页的第 5 章，『安装业务流程和人员任务应用程序』
可以将包含业务流程和/或人员任务的服务组件体系结构（SCA）模块分布到部署目标。部署目标可以是服务器或集群。

部署业务流程和人员任务

当 WebSphere Integration Developer 或服务部署为流程或任务生成部署代码时，每个流程组件或任务组件均被映射到一个会话企业 bean。所有部署代码都将打包到企业应用程序（EAR）文件中。此外，对于每个流程，在安装企业应用程序期间，将生成在此流程中表示 Java 代码的 Java 类并将其嵌入在 EAR 文件中。必须将所要部署的模型的每个新版本都打包到新的企业应用程序中。

当安装包含业务流程或人员任务的企业应用程序时，它们作为业务流程模板或人员任务模板相应地存储在业务流程编排器数据库中。缺省情况下，新安装的模板将处于“已启动”状态。但是，新安装的企业应用程序将处于“已停止”状态。您可以逐个地启动和停止每个已安装的企业应用程序。

您可以部署许多不同版本的流程模板或任务模板，将每个版本部署在不同的企业应用程序中。当安装新企业应用程序时，将如下所示确定已安装的模板的版本:

- 如果模板名称和目标名称空间尚未存在，那么将安装新模板
- 如果模板名称和目标名称空间与现有模板的名称及目标名称空间相同，但生效日不同，那么将安装现有模板的新版本

注: 模板名称从组件的名称衍生，而不是从业务流程或人员任务衍生。

如果未指定生效日，那么日期是按如下方式确定的:

- 如果使用 WebSphere Integration Developer，那么生效日是人员任务或业务流程的建模日期。
- 如果使用服务部署，那么生效日是运行 serviceDeploy 命令的日期。只有协作任务将安装应用程序的日期作为生效日。

相关任务

第 173 页的第 5 章，『安装业务流程和人员任务应用程序』
可以将包含业务流程和/或人员任务的服务组件体系结构（SCA）模块分布到部署目标。部署目标可以是服务器或集群。

以交互方式安装业务流程和人员任务应用程序

您可以使用 wsadmin 工具和 installInteractive 脚本在运行时期间以交互方式安装应用程序。可以使用此脚本来更改在使用管理控制台来安装应用程序时无法更改的设置。

关于此任务

以交互方式执行下列步骤以安装业务流程应用程序。

过程

1. 启动 wsadmin 工具。

   在 *profile_root*/bin 目录中，输入 `wsadmin`。
2. 安装该应用程序。

   在 wsadmin 命令行提示符处输入下列命令:

   ```
   $AdminApp installInteractive application.ear
   ```

   其中 *application.ear* 是包含流程应用程序的企业归档文件的标准名称。 在执行一系列任务过程中会在可更改应用程序值的地方对您进行提示。
3. 保存配置更改。

   在 wsadmin 命令行提示符处输入下列命令:

   ```
   $AdminConfig save
   ```

   您必须保存更改以将更新传送至主配置库。如果脚本流程结束，并且尚未保存更改，那么会放弃这些更改。

配置流程应用程序数据源和集合引用设置

您可能需要为特定数据库基础结构配置运行 SQL 语句的流程应用程序。这些 SQL 语句可以来自信息服务活动，也可以是您在流程安装或实例启动期间运行的语句。

关于此任务

当安装该应用程序时，可以指定下列类型的数据源:

- 要在流程安装期间运行 SQL 语句的数据源
- 要在启动流程实例期间运行 SQL 语句的数据源
- 要运行 SQL 片段活动的数据源

运行 SQL 片段活动所需的数据源是在类型 tDataSource 的 BPEL 变量中定义的。SQL 片段活动所需的数据库模式和表名是在类型 tSetReference 的 BPEL 变量中定义的。您可以配置这两个变量的初始值。

可以使用 wsadmin 工具来指定数据源。

过程

1. 使用 wsadmin 工具以交互方式安装流程应用程序。
2. 逐步完成任务，直至到达更新数据源和集合引用的任务。

   为您的环境配置这些设置。下列示例显示可为每个任务更改的设置。

3. 保存更改。

示例: 使用 wsadmin 工具更新数据源和集合引用

在**更新数据源**任务中，可以为安装流程或启动流程期间使用的初始变量值和语句更改数据源值。在**更新集合引用**任务中，可以配置与数据库模式和表名相关的设置。

```
Task [24]: Updating data sources

//Change data source values for initial variable values at process start

Process name:  Test
// Name of the process template
Process start or installation time:  Process start
// Indicates whether the specified value is evaluated
//at process startup or process installation
Statement or variable:  Variable
// Indicates that a data source variable is to be changed
Data source name:  MyDataSource
// Name of the variable
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name to jdbc/newName
Task [25]: Updating set references

// Change set reference values that are used as initial values for BPEL variables

Process name:  Test
// Name of the process template
Variable:  SetRef
// The BPEL variable name
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name of the data source of the set reference to jdbc/newName
Schema name: [IISAMPLE]
// The name of the database schema
Schema prefix: []:
// The schema name prefix.
// This setting applies only if the schema name is generated.
Table name: [SETREFTAB]: NEWTABLE
// Sets the name of the database table to NEWTABLE
Table prefix: []:
// The table name prefix.
// This setting applies only if the prefix name is generated.
```

使用管理控制台来卸载业务流程和人员任务应用程序

您可以使用管理控制台来卸载包含业务流程或人员任务的应用程序。

开始之前

要卸载包含业务流程或人员任务的应用程序，必须符合下列先决条件:

- 如果已在独立服务器上安装应用程序，那么服务器必须在运行，并且可访问业务流程编排器数据库。
- 如果已在集群上安装应用程序，那么 Deployment Manager 和至少一个集群成员必须在运行。集群成员可访问业务流程编排器数据库。
- 如果已在受管服务器上安装应用程序，那么 Deployment Manager 和此服务器必须在运行。服务器必须可访问业务流程编排器数据库。
- 属于应用程序的所有业务流程模板和人员任务模板必须处于已停止状态。
- 不存在任何状态的业务流程或人员任务模板的实例。

-

对于用作开发和单元测试环境的独立服务器环境，可配置服务器以便在开发方式下运行。此配置不要求停止模板，也不要求没有实例存在。然而，此配置对生产环境无效。

关于此任务

要卸载包含业务流程或人员任务的企业应用程序，请执行下列操作:

过程

1. 停止该应用程序中的所有流程和任务模板。

   此操作的目的是防止创建流程和任务实例。

   a. 单击管理控制台导航窗格中的**应用程序 → SCA 模块**。
   b. 选择包含您要停止的模板的模块。
   c. 在“其他属性”下面，根据情况单击**业务流程**和/或**人员任务**。
   d. 通过单击相应的复选框，选择所有流程和任务模板。
   e. 单击**停止**。

   对所有包含业务流程模板或人员任务模板的 EJB 模块重复此步骤。
2. 验证数据库、每个集群的至少一个应用程序服务器以及部署了该应用程序的独立服务器是否正在运行。

   在 Network Deployment 环境中，对于已安装应用程序的每个集群，Deployment Manager、所有受管独立应用程序服务器和至少一个应用程序服务器必须在运行。
3. 验证应用程序以确保没有业务流程实例或人员任务实例。

   在必要的时候，管理员可以使用业务流程编排器资源管理器来删除任何流程或任务实例。
4. 停止并卸载该应用程序:
   a. 在管理控制台导航窗格中，单击**应用程序 → 企业应用程序**。
   b. 选择要卸载的应用程序，然后单击**停止**。

      如果该应用程序仍包含任何流程实例或任务实例，那么此步骤将失败。
   c. 再次选择要卸载的应用程序，然后单击**卸载**。
   d. 单击**保存**以保存更改。

结果

该应用程序被卸载。

使用管理命令来卸载业务流程和人员任务应用程序

可使用管理命令来代替管理控制台卸载包含业务流程或人员任务的应用程序。

开始之前

要卸载包含业务流程或人员任务的应用程序，必须符合下列先决条件:

- 如果已在独立服务器上安装应用程序，那么服务器必须在运行，并且可访问业务流程编排器数据库。
- 如果已在集群上安装应用程序，那么 Deployment Manager 和至少一个集群成员必须在运行。集群成员可访问业务流程编排器数据库。
- 如果已在受管服务器上安装应用程序，那么 Deployment Manager 和此服务器必须在运行。服务器必须可访问业务流程编排器数据库。
- 属于应用程序的所有业务流程模板和人员任务模板必须处于已停止状态。
- 不存在任何状态的业务流程或人员任务模板的实例。
-

对于用作开发和单元测试环境的独立服务器环境，可配置服务器以便在开发方式下运行。此配置不要求停止模板，也不要求没有实例存在。然而，此配置对生产环境无效。

此外，如果启用了全局安全性，那么验证您的用户标识是否具有操作员权限。

确保管理客户机所连接的服务器流程正在运行。要确保管理客户机自动连接到该服务器流程，请不要使用 `-conntype NONE` 命令选项。

关于此任务

下列步骤描述如何使用 bpcTemplates.jacl 脚本来卸载包含业务流程模板或人员任务模板的应用程序。在可以卸载模板所属的应用程序之前，必须先停止该模板。可以使用 bpcTemplates.jacl 脚本来通过一个步骤停止和卸载模板。

在卸载应用程序之前，可以删除应用程序中与模板相关联的流程实例或任务实例，例如，使用业务流程编排器资源管理器来删除这些实例。您还可以将 **-force** 选项与 bpcTemplates.jacl 脚本配合使用，以便通过一个步骤就删除所有与模板相关联的实例、停止模板以及卸载模板。

注意:
因为 -force 选项会删除所有流程实例和任务实例数据，所以使用此选项时应小心谨慎。

过程

1. 切换到业务流程编排器样本目录。

   在 Windows 平台上，输入:

   ```
   cd install_root\ProcessChoreographer\admin
   ```

   在 Linux、UNIX 和 i5/OS 平台上，输入:

   ```
   cd install_root/ProcessChoreographer/admin
   ```

2. 停止模板并卸载相应的应用程序。

   在 Windows 平台上，输入:

   ```
   install_root\bin\wsadmin -f bpcTemplates.jacl
                               [-user user_name]
                               [-password user password]
                               -uninstall application_name
                               [-force]
   ```

   在 Linux、UNIX 和 i5/OS 平台上，输入:

```
install_root/bin/wsadmin -f bpcTemplates.jacl
                           [-user user_name]
                           [-password user password]
                           -uninstall application_name
                           [-force]
```

其中:

user_name
如果启用了全局安全性，那么提供用户标识以便进行认证。

user_password
如果启用了全局安全性，那么提供用户密码以便进行认证。

application_name
如果启用了全局安全性，那么提供用户密码以便进行认证。

结果

该应用程序被卸载。

第 6 章 安装适配器

适配器允许您的应用程序与企业信息系统的其他组件进行通信。

WebSphere Integration Developer 信息中心中的配置和使用适配器描述了安装适配器所采用的步骤。

第 7 章 安装 EIS 应用程序

EIS 应用程序模块可部署到 J2EE 平台。执行该部署将会产生打包为部署到服务器上的 EAR 文件的应用程序。在部署过程中将创建所有 J2EE 工件和资源并对该应用程序进行配置，部署完成后，该应用程序已能够运行。

关于此任务

在部署到 J2EE 平台时，将创建下列 J2EE 工件和资源:

表 42. 从绑定到 J2EE 工件的映射

SCA 模块中的绑定	生成的 J2EE 工件	创建的 J2EE 资源
EIS 导入	对模块会话 EJB 生成的资源引用。	连接工厂
EIS 导出	根据资源适配器支持的侦听器接口生成的或部署的消息驱动的 Bean。	激活规范
JMS 导入	部署运行时提供的消息驱动的 Bean（MDB），对模块会话 EJB 生成资源引用。注意，如果导入有接收目标，那么将仅创建 MDB。	• 连接工厂 • 激活规范 • 目标
JMS 导出	部署运行时提供的消息驱动的 Bean，对模块会话 EJB 生成资源引用。	• 激活规范 • 连接工厂 • 目标

如果导入或导出定义了资源（例如连接工厂），就会在模块无状态会话 EJB 的部署描述符中生成资源引用。并且，还将在 EJB 绑定文件中生成适当的绑定。如果指定了 target 属性，那么与资源引用绑定的名称是该属性的值，否则，该名称是根据模块名和导入名对资源指定的缺省 JNDI 查询名称。

部署完成后，实现将查找模块会话 Bean 并使用它来查找资源。

在将应用程序部署到服务器期间，EIS 安装任务将检查与其绑定的元素资源是否存在。如果该资源不存在，并且 SCDL 文件至少指定了一个属性，EIS 安装任务就会创建并配置该资源。如果该资源不存在，那么不执行任何操作，并假定在执行该应用程序前将创建该资源。

使用接收目标部署 JMS 导入时，将部署消息驱动的 Bean（MDB）。它将侦听已发送的请求的应答。该 MDB 与随 JMS 消息的 JMSreplyTo 头字段中的请求一起发送的目标相关联（侦听该目标）。应答消息到达时，MDB 将使用其相关标识来检索回调目标中存储的回调信息，然后调用回调对象。

安装任务根据导入文件中的信息创建连接工厂和三个目标。并且，它将创建激活规范以使运行时 MDB 能够侦听接收目标上的应答。激活规范的属性是从目标/连接工厂属性派生的。如果 JMS 提供程序是 SIBus 资源适配器，那么将创建与 JMS 目标相对应的 SIBus 目标。

部署 JMS 导出时，将部署消息驱动的 Bean（MDB）。注意，这不是为 JMS 导入部署的那个 MDB。它侦听接收目标上的入局请求，然后将那些请求分派给 SCA 处理。安装任务将创建如下一组资源，这些资源与部署 JMS 导入时创建的资源类似：激活规范、连接工厂和两个目标。这些资源的所有属性都是在导出文件中指定的。如果 JMS 提供程序是 SIBus 资源适配器，那么将创建与 JMS 目标相对应的 SIBus 目标。

将 EIS 应用程序模块部署到 J2SE 平台

可以将 EIS 模块部署到 J2SE 平台，但是，只支持 EIS 导入。

开始之前

在开始执行本任务前，您需要在 WebSphere 集成开发环境中创建包含 JMS 导入绑定的 EIS 应用程序模块。

关于此任务

当要使用消息队列来以异步方式访问 EIS 系统时，EIS 应用程序模块将包含 JMS 导入绑定。

部署到 J2SE 平台是唯一一种支持以非受管方式执行绑定实现的情况。JMS 绑定需要异步和 JNDI 支持，但基本服务组件体系结构和 J2SE 都没有提供这两种支持。J2EE 连接器体系结构不支持非受管入站通信，因此消除了 EIS 导出。

在将包含 EIS 导入的 EIS 应用程序模块部署到 J2SE 时，除了模块依赖项以外，还必须在清单中或者以 SCA 支持的任何其他方式将该导入使用的 WebSphere Adapter 指定为依赖项。

将 EIS 应用程序模块部署到 J2EE 平台

将 EIS 模块部署到 J2EE 平台时，将在该服务器上部署一个打包成 EAR 文件的应用程序。在部署过程中将创建所有 J2EE 工件和资源并对该应用程序进行配置，部署完成后，该应用程序已能够运行。

开始之前

在开始执行本任务前，您需要在 WebSphere 集成开发环境中创建包含 JMS 导入绑定的 EIS 模块。

关于此任务

在部署到 J2EE 平台时，将创建下列 J2EE 工件和资源：

表 43. 从绑定到 J2EE 工件的映射

SCA 模块中的绑定	生成的 J2EE 工件	创建的 J2EE 资源
EIS 导入	对模块会话 EJB 生成的资源引用。	连接工厂
EIS 导出	根据资源适配器支持的侦听器接口生成的或部署的消息驱动的 Bean。	激活规范

表 43. 从绑定到 J2EE 工件的映射 (续)

SCA 模块中的绑定	生成的 J2EE 工件	创建的 J2EE 资源
JMS 导入	部署运行时提供的消息驱动的 Bean（MDB），对模块会话 EJB 生成资源引用。注意，如果导入有接收目标，那么将仅创建 MDB。	• 连接工厂 • 激活规范 • 目标
JMS 导出	部署运行时提供的消息驱动的 Bean，对模块会话 EJB 生成资源引用。	• 激活规范 • 连接工厂 • 目标

如果导入或导出定义了资源（例如连接工厂），就会在模块无状态会话 EJB 的部署描述符中生成资源引用。并且，还将在 EJB 绑定文件中生成适当的绑定。如果指定了 target 属性，那么与资源引用绑定的名称是该属性的值，否则，该名称是根据模块名和导入名对资源指定的缺省 JNDI 查询名称。

部署完成后，实现将查找模块会话 Bean 并使用它来查找资源。

在将应用程序部署到服务器期间，EIS 安装任务将检查与其绑定的元素资源是否存在。如果该资源不存在，并且 SCDL 文件至少指定了一个属性，EIS 安装任务就会创建并配置该资源。如果该资源不存在，那么不执行任何操作，并假定在执行该应用程序前将创建该资源。

使用接收目标部署 JMS 导入时，将部署消息驱动的 Bean（MDB）。它将侦听已发送的请求的应答。该 MDB 与随 JMS 消息的 JMSreplyTo 头字段中的请求一起发送的目标相关联（侦听该目标）。应答消息到达时，MDB 将使用其相关标识来检索回调目标中存储的回调信息，然后调用回调对象。

安装任务根据导入文件中的信息创建连接工厂和三个目标。并且，它将创建激活规范以使运行时 MDB 能够侦听接收目标上的应答。激活规范的属性是从目标/连接工厂属性派生的。如果 JMS 提供程序是 SIBus 资源适配器，那么将创建与 JMS 目标相对应的 SIBus 目标。

部署 JMS 导出时，将部署消息驱动的 Bean（MDB）。注意，这不是为 JMS 导入部署的那个 MDB。它侦听接收目标上的入局请求，然后将那些请求分派给 SCA 处理。安装任务将创建如下一组资源，这些资源与部署 JMS 导入时创建的资源类似：激活规范、连接工厂和两个目标。这些资源的所有属性都是在导出文件中指定的。如果 JMS 提供程序是 SIBus 资源适配器，那么将创建与 JMS 目标相对应的 SIBus 目标。

第 8 章 对失败的部署进行故障诊断

本主题描述部署应用程序时用于确定问题原因的步骤。本主题还提供了一些可能有效的解决方案。

开始之前

本主题假设:
- 您基本了解模块的调试方式。
- 部署模块时，日志记录和跟踪处于活动状态。

关于此任务

在接收到错误通知后，部署故障诊断任务就开始了。在执行操作前，必须检查失败的部署的各种症状。

过程

1. 确定应用程序的安装是否已失败。

   检查 SystemOut.log 文件以获取指定了故障原因的消息。下面是其中一些导致无法安装应用程序的原因:
   - 您尝试在同一 Network Deployment 单元中的多个服务器上安装应用程序。
   - 应用程序与正在安装该应用程序的 Network Deployment 单元上的现有模块同名。
   - 您尝试将 EAR 文件中的 J2EE 模块部署到不同的目标服务器上。

   要点: 如果安装已失败，并且应用程序包含服务，那么在尝试重新安装该应用程序前，必须除去发生故障前创建的任何 SIBus 目标或 J2C 激活规范。除去这些工件的最简单方法是在发生故障后单击**保存 > 全部废弃**。如果无意中保存了更改，那么必须手动除去 SIBus 目标和 J2C 激活规范（请参阅“管理”部分中的“删除 SIBus 目标”和“删除 J2C 激活规范”）。
2. 如果正确地安装了应用程序，那么检查它是否已成功启动。

   如果该应用程序未成功启动，那么表明服务器尝试启动该应用程序的资源时发生了故障。
   a. 检查 SystemOut.log 文件以获取指示了如何继续进行处理的消息。
   b. 确定应用程序所需的资源是否可用和/或是否已成功启动。

      未启动的资源导致应用程序无法运行。这可防止信息丢失。导致资源无法启动的原因包括:
      - 未正确地指定绑定
      - 未正确地配置资源
      - 资源未包含在资源归档（RAR）文件中
      - Web 资源未包含在 Web service 归档（WAR）文件中

   c. 确定是否遗漏了任何组件。

遗漏组件的原因是未正确地构建企业归档（EAR）文件。确保模块所需的所有组件都在构建 Java 归档（JAR）文件的测试系统上的正确文件夹中。『准备部署到服务器』包含其他信息。

3. 检查该应用程序，了解是否有信息流经该应用程序。

   即使处于运行状态的应用程序也可能无法处理信息。此故障的原因与第 187 页的 2b步骤中提到的原因类似。

   a. 确定应用程序是否使用了另一应用程序中包含的任何服务。 确保其他应用程序已安装并成功启动。
   b. 确定是否已正确地配置了由失败应用程序使用的其他应用程序中包含的设备的导入和导出绑定。 使用管理控制台来检查并更正这些绑定。

4. 更正问题，然后重新启动该应用程序。

删除 J2C 激活规范

在安装包含服务的应用程序时，系统将构建 J2C 应用程序规范。在某些情况下，重新安装该应用程序前必须删除这些规范。

开始之前

如果由于安装应用程序失败而需要删除规范，请确保 Java 命名和目录接口（JNDI）名称中的模块与未能安装的模块名称匹配。JNDI 名称的第二部分是实现了该目标的模块名称。例如，在 sca/SimpleBOCrsmA/ActivationSpec 中，**SimpleBOCrsmA** 是模块名。

此任务所需的安全角色：当启用安全和基于角色的授权时，您必须以管理员或配置员的身份登录来执行此任务。

关于此任务

在安装包含服务并且不需要激活规范的应用程序后，如果无意中保存了配置，那么可删除 J2C 激活规范。

过程

1. 查找要删除的激活规范。

   激活规范包含在资源适配器面板中。通过单击**资源 > 资源适配器**，转到此面板。

   a. 查找**平台消息传递组件 SPI 资源适配器**。

      要查找此适配器，您必须在**节点**作用域中（对于独立服务器）或**服务器**作用域中（对于部署环境）。

2. 显示与平台消息传递组件 SPI 资源适配器相关联的 J2C 激活规范。

   单击资源适配器名，下一个面板将显示相关联的规范。

3. 删除 **JNDI 名称**与所要删除的模块名相匹配的所有规范。

   a. 单击相应规范旁边的复选框。
   b. 单击**删除**。

结果

系统将不再显示选择的规范。

下一步做什么?

保存更改。

删除 SIBus 目标

SIBus 目标是向应用程序提供服务的连接。在某些情况下，您必须除去这些目标。

开始之前

如果由于安装应用程序失败而需要删除目标，请确保目标名中的模块与未能安装的模块名称匹配。目标名的第二部分是实现了该目标的模块名称。例如，在 sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer 中，**SimpleBOCrsmA** 是模块名。

此任务所需的安全角色: 当启用安全和基于角色的授权时，您必须以管理员或配置员的身份登录来执行此任务。

关于此任务

在安装包含服务的应用程序或不再需要 SIBus 目标后，如果无意中保存了配置，那么可删除 SIBus 目标。

注: 本任务仅从 SCA 系统总线中删除目标。在重新安装包含服务的应用程序之前，还必须从应用程序总线中除去条目（请参阅本信息中心“管理”部分中的“删除 J2C 激活规范”）。

过程

1. 登录管理控制台。
2. 显示 SCA 系统总线上的目标。

   通过单击**服务集成 > 总线**，转到该面板。
3. 选择 SCA 系统总线目标。

   在屏幕上，单击 **SCA.SYSTEM.*cellname*.Bus**，其中 *cellname* 是带有所要删除的目标的模块所在单元的名称。
4. 删除模块名与所要删除的模块相匹配的目标。
   a. 单击相关目标旁边的复选框。
   b. 单击**删除**。

结果

此面板将仅显示余下的目标。

下一步做什么?

删除创建了这些目标的模块的相关 J2C 激活规范。

第 3 部分 附属资料

声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档中所描述内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation North Castle Drive Armonk, NY 10504-1785 U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

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

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含日常商业运作所使用的数据和报表的示例。为了尽可能完整地说明这些示例，这些示例中包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明: ©（贵公司的名称）（年）。此部分代码根据 IBM 公司的样本程序衍生。© Copyright IBM Corp. _输入年份_。All rights reserved.

如果您正以软拷贝格式查看本信息，图片和彩色图例可能无法显示。

编程接口信息

如果提供了编程接口信息，则该信息旨在帮助您使用本程序来创建应用软件。

通用编程接口允许您编写获取此程序工具的服务的应用软件。

然而，本信息还可能包含诊断、修改和调整信息。这些诊断、修改和调整信息用于帮助您调试应用软件。

警告: 不要将此诊断、修改和调整信息用作编程接口，因为它是会更改的。

商标和服务标记

IBM、IBM 徽标、developerWorks、WebSphere 和 z/OS 是 International Business Machines Corporation 在美国和/或其他国家或地区的注册商标。

Adobe 是 Adobe Systems Incorporated 在美国和/或其他国家或地区的注册商标。

Java 和所有基于 Java 的商标是 Sun Microsystems, Inc. 在美国和/或其他国家或地区的商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

此产品包括由 Eclipse Project（http://www.eclipse.org）开发的软件。

IBM WebSphere Process Server for Multiplatforms V6.1.0

中国印刷