CHAPTER

6

第 6 章　BPEL 编程思想详解

随着 Web Servcie 技术日益成熟和流行，许多企业的很多部门相应地都创建了 Web Service 服务。如何在不改变这些 Web Service 正常运行的情况下，将这些 Web Service 集成起来创造出新的业务模型、业务流程就成为一个比较突出的业务需求。

要解决这个问题，一定要有一种新的流程语言能够将 Web Service 给串起来，这种新的业务流程语言就是 BPEL。

BPEL 需要在 BPEL 的运行环境（BPEL 引擎）下运行。一个 BPEL 流程一般会创建一个 BPEL 流程实例，BPEL 就是定义这个 BPEL 流程如何和合作伙伴（外部的 Web Service）进行交互的。

BPEL 将通过合作伙伴连接来实现服务的调用。BPEL 的合作伙伴连接只定义所要调用的接口，一个抽象的 WSDL 接口，这个接口在 BPEL 运行时再绑定到真正的服务提供者上面。这样使接口和实现之间达到了一种松散耦合的效果，如果说 Web Service 实现了接口的可重用性，那么 BPEL 实现的是流程的可重用性。因为 BPEL 实现了抽象的 WSDL 接口的集成，所以它也属于 SOA 的解决方案之一。

本章主要内容：

- 结合 Java 程序的实例来说明 BPEL 的基本思想；
- 通过房屋贷款的实例详细说明 BPEL 的创建过程。
- 通过实例阐明 BPEL 的各种过程组件的概念和用法；
- 通过实例阐明 BPEL 的各种活动的概念和用法。

6.1　BPEL的基本思想

许多开发人员觉得 BPEL 很神秘，不知道到底是什么意思。主要是因为它是根据很抽象的基于 WSDL 的 Web Service 再定义一些抽象执行的流程。

其实 BPEL 一点都不神秘，是一个很简单的东西。

首先谈一下 BPEL 和 WSDL 的区别，WSDL 只是定义接口参数，不会定义如何实现接口，而 BPEL 不仅有自己的接口定义（也是一个 WSDL 文件，包括输入参数、方法操作名、返回参数），BPEL 会定义如何调用其他服务的接口来实现自己的接口。简单地说，BPEL 通过流程编程将各种接口组合在一起，其目的在于提供一个“集成了各种接口”的接口。

首先 BPEL 会有一个起点和终点。

- 它的起点就是“receive”，也就是接收它自己接口的输入参数。
- 它的终点就是“reply”，也就是得到它自己接口的返回参数。

整个 BPEL 就是定义如何通过它的接口输入参数，调用其他外部服务的接口，得到其接口的返回参数。

它最重要的是两个定义：

- 一个是赋值命令（Assign/Copy），通过赋值命令将某一变量值赋给所要调用的接口的输入参数。
- 另一个是调用命令（Invoke），通过 Invoke 命令来调用外部服务。

另外一个比较重要的就是条件命令（Case Condition），根据变量的不同来定义各种条件，然后根据各种条件来调用不同的服务。这些变量可以来自于输入参数，也可以来自于调用外部服务的结果。作为 BPEL 运行环境的一个重要功能就是能够保存并查询到这些变量。

6.1.1 用 Java 实例模拟 BPEL 的创建过程

这里用一个简单的实例来进一步说明 BPEL 的基本思想和核心本质，并用 Java 实例来模拟 BPEL 的创建过程。

首先假定已经有了两个 Web Service，一个为“加法服务”，专门处理两个数相加，名为 AddService；另外一个为“减法服务”，专门处理两个数相减，名为 SubtractService。

现在有一个新的需求，需要将上面两个 Web Service 集成起来，也就是需要创建一个新的服务，称为“运算服务”，名为 CaculatorService，它有一个运算类型的参数，当运算类型为“加法”时，调用加法服务，当运算类型为“减法”时，调用减法服务。

看到这里，读者可能会想，直接写一个 Java 程序，调用者两个 Web Service 不就可以了吗？笔者的回答是：直接用 Java 编程当然可以，事实上现在大家就是这么做的。但是 Java 本质上是一种具体的程序语言，只能运行于 JVM 的 Java 运行环境，不是一种抽象性的通用的标准语言，而 BPEL 是一种标准化的执行语言，如何能够真正运行 BPEL 不是它所关心的事情，它还需要各个厂商去开发自己的 BPEL 运行环境，尽管各个厂商所开发的 BPEL 的运行环境可能会不一样，但是同一个 BPEL 所开发的程序，应该可以运行于这些不同的 BPEL 运行环境，得到的结果应该是一样的。就像 Web Service 的 WSDL 文件一样，它只管如何定义服务、服务接口、服务操作、服务参数等，如何具体实现 Web Service 不是 WSDL 所要关心的事情。

下面继续前面的实例，两个 Service 都会有自己的 WSDL 定义。下面用实际的 Java 语言来类似地描述一下，以便于理解。

下面的 AddService .java 相当于 AddService.wsdl（用 Java 的 Interface 可能更贴切一点，用 Java 的 class 可以说明得更详细一点）。

```
Public class AddService{
    Public double add(double addParameter1, double addParameter2) {
        Double addResposne;
            addResposne = addParameter1 ＋addParameter2;
                // WSDL 不会定义具体如何实现，
```

```
                    //此处只是说明如何实现操作
                Return addResponse;
            }
        }
```

下面的 SubstractService .java 相当 SubstractService.wsdl。

```
Public class SubstractService{
    Public double substract(double substractParameter1, double substractParameter2) {
        Double substractResposne = substractParameter1 － substractParameter2;
        Return substractResposne;
    }
}
```

下面用 Java 模拟 BPEL 的创建过程。

首先需要创建 BPEL 的接口，下面用 CaculatorBPELInterface.java 来说明，它将有 3 个参数，其中 paramter1 和 paramter2 是需要运算的两个数，第 3 个参数 processType 表示运算的类型。

```
Public Interface CaculatorBPELInterface{

    Public double caculatorProcess(double parameter1, double parameter2, String processType) ;
}
```

下面将创建一个 Java 实现类，说明如何对应于 BPEL 的创建过程（许多语句（如一些变量定义和赋值定义）对 Java 来说是不必要的；为了模拟 BPEL 的创建过程，让 Java 开发人员更好地把握 BPEL 的创建过程，相应地加入了这些程序语句）。

```
Public class CaculatorBPELImple implements CaculatorBPELInterface {

    Public double caculatorProcess (double parameter1, double parameter2, String processType) {

//步骤 1：定义所要调用的外部类（相当于定义 BPEL 里面 partnerLink）
        AddService addServer = new AddService();
        SubstractService substractSevice = new SubstractService();

/*步骤 2：定义输入和输出变量（相当于定义 BPEL 里面变量 variable）
定义的变量如下：
● BPEL 接口的输入和输出变量
● 所要调用的外部类的接口方法的输入和输出变量 */
```

```
//定义 BPEL 接口的输入变量
Double caculatorProcessParameter1Request;
Double caculatorProcessParameter2Request;
Double caculatorProcessTypeRequest;

//定义 BPEL 接口的输出变量
Double caculatorProcessResponse;
```

```
//定义加法服务的输入变量
Double addParameter1Request;
Double addParameter2Request;

//定义加法服务的输出变量
Double addResponse;

//定义减法服务的输入变量
Double substractParameter1Request;
Double substractParameter2Request;

//定义减法服务的输出变量
Double substractParameter1Response;

//***将请求参数赋值给 BPEL 接口的输入变量 （相当于 BPEL 的 receive）
caculatorProcessParameter1Request = parameter1;
caculatorProcessParameter2Request= parameter2;
Double caculatorProcessTypeRequest= processType;

//步骤 3：定义条件，并调用外部接口
If (caculatorProcessTypeRequest.equals.(“add”’)
// （相当于 BPEL 的 switch/condition/case）
{
      //**下面将调用加法服务
      //将接口请求变量传给加法服务的请求变量 （相当于 BPEL 的 assign/copy）
 addParameter1Request = caculatorProcessParameter1Request;
 addParameter2Request = caculatorProcessParameter2Request;

 //调用 addService 的接口 （相当于 BPEL 的 Invoke）
 addResponse = addServer.add(addParameter1Request, addParameter2Request);

 //将 addResponse 赋值给 BPEL 接口的输出变量 （相当于 BPEL 的 assign/copy）
caculatorProcessResponse = addResponse;
} else        //（相当于 BPEL 的  condition/otherwise）
{
 //将接口请求变量传给减法服务的请求变量 （相当于 BPEL 的 assign/copy）
 substractParameter1Request = caculatorProcessParameter1Request;
 substractParameter2Request = caculatorProcessParameter2Request;

 //调用 substractService 的接口 （相当于 BPEL 的 invoke）
 substractResponse =
     substractServer.substract(substractParameter1Request, substractParameter2Request);

 //将 substractResponse 赋值给 BPEL 接口的输出变量（相当于 BPEL 的 assign/copy）
 caculatorProcessResponse = substractResponse;
```

```
            }
            Return caculatorProcessResponse;   //相当于 BPEL 的 reply
        }
    }
```

6.1.2 用实例概述 BPEL 的创建过程

上面已经用 Java 模拟了 BPEL 的创建过程。下面将基于同样的实例，用描述性的 WSDL 和 BPEL 来实现其创建过程，使读者对 BPEL 能有一个整体性的掌握。

假设已经有了前面的加法服务和减法服务的 WSDL 文件，加法服务为 AddService.wsdl，减法服务为 Substract.wsdl，它们的主要内容如下所示：

```
AddService.wsdl
    |——getRequest （请求消息）
            |——addParameter1（double）
            |——addParameter2（double）
    |——getResponse（返回消息）
            |——addResponse（double）
    |——addServcie（portType 接口）
            |——add（operation 接口操作）
    |——AddService（service 服务名称）


substractService.wsdl
    |——getRequest（请求消息）
            |——subtractParameter1（double）
            |——subtractParameter2（double）
    |——getResponse（返回消息）
            |——substractResponse（double）
    |——substractService（portType 接口）
            |——substract（operation 接口操作）
    |——SubstractService（服务名称）
```

下面需要为 BPEL 创建一个服务接口，caculator.wsdl 如下：

```
caculatorService.wsdl
    |——getRequest（请求消息）
            |——parameter1（double）
            |——parameter2（double）
            |——processType （String）
    |——getResponse（返回消息）
            |——caculatorProcessResponse（double）
    |——caculatorService（portType 接口）
|——caculatorPorcess（operation 接口操作）
    |——CaculatorService（service 服务名称）
```

下面介绍创建 BPEL 的基本过程，即 caculatorServiceProcess.bpel（下面只是说明 BPEL 的创建过程，没有完全按照 BPEL 的语法，具体的 BPEL 语法和 BPEL 编程实例后面会详

细介绍）。

1）**创建变量**<bpel:variables>

所定义的变量包括：

- BPEL 接口的输入和输出变量。
- 所要调用的外部服务 partnerLink 的接口操作的输入和输出变量。

```
<bpel:variables>

 <!—BPEL 流程的请求变量和返回变量 -->
 variable   name="request-bpel"   messageType 采用 caculatorService/getRequest
 variable   name="response-bpel"   messageType 采用 caculatorService/getResponse

 <!—加法服务的请求变量和返回变量 -->
 variable   name="request-add"   messageType 采用 addService/getRequest
 variable   name="response-add"   messageType 采用 addService/getResponse
 <!—减法服务请求变量和返回变量 -->
 variable   name="request-substract"   messageType 采用 substractService/getRequest
 variable   name="response-substract"   messageType 采用 substractService/getResponse
</bpel:variables>
```

BPEL 定义变量的方式与 Java 是不一样的，主要因为 BPEL 所调用的 WSDL 是 XML 语言上面的所定义的变量并不是对应一个具体的值，它实际上对应的是一个数据结构。

如 caculatorService/getRequest 对应的是 caculatorService.wsdl 下面的

```
getRequest
            |——parameter1（double）
            |——parameter2  (double)
            |——processType （String）
```

也就是说 getRequest 下面的所有参数都包含进去了。

2）**创建接收**<bpel:receive>

```
<bpel:receive name="request" partnerLink=" CaculatorService "
        portType=" caculatorService " operation=" caculatorPorcess "
        variable=" request-bpel " >
```

Receive 是整个 BPEL 的起点，所定义的变量 request-bpel 从服务请求中得到赋值，整个后面的业务过程将以这个请求变量作为触发点。

3）**创建条件**<bpel:switch/case>

程序将根据不同的条件调用不同的服务，所以先要设立各种条件：

```
<bpel:switch>
  <bpel:case
  condition="getVariableData('request-bpel','payload','getRequest/processType'>=   'add' >
            … 调用加法服务
</bpel:case>
```

上面的程序表示在 request-bpel 的变量中取出路径为 getRequest/ processType 所对应的变量值，如果满足这个值为“add”的条件时，可以在里面加入程序，完成相应的任务。

```
<bpel:otherwise>
  …… 调用减法服务
<bpel:otherwise>
```

如果上面的所有条件都不满足时，可以在里面加入程序，完成默认的任务。

4）**给所要调用的服务的请求变量赋值**<bpel:assign/copy>

为了调用外部服务，先要给外部服务赋值：

```
<bpel:assign>
        <bpel:copy>
        <bpel:from variable="request-bpel" part="payload" query="/getRequest/parameter1" />
        <bpel:to variable="request-add"    part="payload" query="/getRequest /addParameter1" />
        </bpel:copy>
</bpel:assign>
```

上面表示将 BPEL 过程所收到的初始变量赋给加法服务的请求变量。

5）**调用外部服务**<bpel:invoke>

```
<bpel:invoke name="addService" partnerLink="AddService"
                               portType="addService" operation="add"
                               inputVariable=" request-add"
                               outputVariable="response-add" />
```

上面将会调用加法服务 AddSerivce，其中 request-add 为输入变量。

6）**将服务的结果赋给 BPEL 的返回变量**<bpel:assign/copy>

在完成了外部服务的调用之后，就可以将外部服务的输出结果赋给 BPEL 流程的返回变量。文法同步骤 4。

7）**调用**<bpel:reply>

调用 BPEL 的 reply 命令，将 BPEL 流程的返回变量返回给服务请求者。

```
<bpel:reply name="response" partnerLink="CaculatorService"
            portType=" caculatorService" operation="caculatorPorcess"
            variable=" response-bpel" />
```

这里比较一下步骤 2 的<bpel:receive>和步骤 6 的<bpel:reply>，可以看到它们的 partnerLink 的名字、portType 的名字、operation 的名字都是一样的。它们是调用的同一个服务下面的同一个接口操作，只是两个命令的变量不一样。<bpel:receive>是收到请求消息的变量值，<bpel:reply>是将响应变量的结果返回给服务请求者。

事实上，BPEL 运行环境执行<bpel:receive>后，就在等待<bpel:reply>的返回结果。

上面通过实例介绍了 BPEL 的基本创建过程，有了这些基本的 BPEL 整体创建思路后，就可以为更好地理解一些具体的语法打下了基础。

6.2 房屋贷款BPEL实例详解

因为 BPEL 本质上只是一个程序语言，要让开发人员知道程序语言的用法，详细介绍程序语言用法，最后是结合实际的典型的实例。所以这里先提出一个房屋贷款实例的实例

来，然后在后面的章节再详细介绍用法。本节的 BPEL 房屋贷款实例需要结合 7.9 节的 ServiceMix 来运行，7.9 节将提供外部服务让 BPEL 来集成，同时 ServiceMix 也将提供 BPEL 的运行环境。

6.2.1 房屋贷款案例介绍

IT 技术本质上都是为业务服务的，只有真正理解了业务，才能真正理解 IT。下面将详细介绍房屋贷款业务的来龙去脉。

1．案例的由来

目前我国的房价正在以飞快的速度上涨，这与“炒房者”一人拥有多套房子不无关系，其实大部分“炒房者”大都是用银行贷款的钱在“炒房”，这与我国目前的银行贷款政策不无关系。目前我国的银行房屋贷款政策对购买一套房和多套房的首付和贷款利率是一样的，这样银行政策是将“炒房者”和“真正的购房者”置于同等的地位，为“炒房者”提供了机会，使他们能够“以小搏大”，以少量的首付贷款得到房子，然后倒手得到巨大的差价。

银行应该根据“购房者”目前的拥有的房屋数量提供不同的首付比例和贷款利率。房屋数量为 0 的客户应该得到最小的首付比例和最低的贷款利率；然后根据客户的房屋数量依次相应提高首付比例和贷款利率，这样应该能够起到平抑房价的作用。下面的实例假定银行政策已经这么执行了。

2．业务现状

目前有 5 个独立的 Web Service，它们相互之间互不知道，也不能互相调用。

- HouseLoanAgency：专门负责根据客户姓名查询客户的已有的房屋数量。
- Bank0：专门负责处理已有房屋数量为 0 的客户的房屋贷款需求，有专门针对房屋数量为 0 的贷款首付和贷款利率。
- Bank1：专门负责处理已有房屋数量为 1 的客户的房屋贷款需求，有专门针对房屋数量为 1 的贷款首付和贷款利率。
- Bank2：专门负责处理已有房屋数量为 2 的客户的房屋贷款需求，有专门针对房屋数量为 2 的贷款首付和贷款利率。
- Bank3：专门负责处理已有房屋数量大于 2 的客户的房屋贷款需求，有专门针对房屋数量大于 2 的贷款首付和贷款利率。

3．业务需求

在完全不影响上面的 5 个 Web Service 正常运行的情况下，包括不修改上面 5 个 Web Service 的任何代码的情况下，将上面的 5 个 Web Service 集成起来。这个新的系统只需要输入用户姓名，这个系统会首先自动查出客户的房屋数量，然后自动转到相应的银行进行房屋贷款服务，客户最后可以得到对应于他的目前已有房屋数量的贷款首付和贷款利率。

4．实现流程

图 6-1 显示了用 BPEL 实现的业务流程，具体过程如下。

（1）BPEL 将首先以客户姓名 name 作为输入变量，调用 HouseLoanAgency 的 Web Service，得到客户的目前拥有的房屋数量。

（2）如果客户房屋数量为 0，BPEL 将调用 Bank0 的服务；如果客户房屋数量为 1，BPEL 将调用 Bank1 的服务；如果客户房屋数量为 2，BPEL 将调用 Bank2 的服务；如果客户房屋数量大于 2，BPEL 将调用 Bank3 的服务。

（3）BPEL 将从 Bank 返回的首付比率和贷款利率返回给服务请求者。

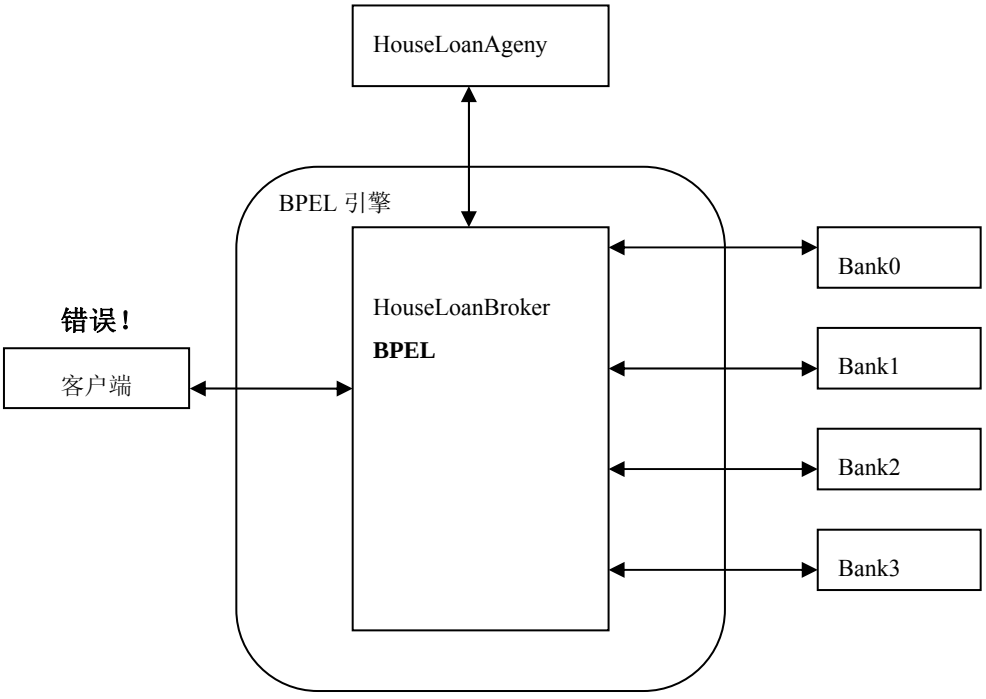

图 6-1　通过 BPEL 实现房屋贷款业务

6.2.2　定义 BPEL 流程的接口 WSDL

定义 BPEL 流程的接口 WSDL 主要有两个目的：

（1）让客户端知道如何使用相应的组织数据来调用这个 BPEL 服务。例如下面的接口 portType 名称为“HouseLoanBroker”；其下的输入 input 为“tns:getLoanQuoteRequest”，其中“tns”为命名空间“urn:sample:soa:houseloanbroker”；getLoanQuoteRequest 下面含有元素“name”；总的结构如下：

```
HouseLoanBroker（portType）
                    |——tns:getLoanQuoteRequest （input）
                                        |——name （String）
```

这样客户端可以创建响应的 SOAP 请求消息：

```
<getLoanQuoteRequest xmlns='urn:sample:soa:houseloanbroker'>
    <name>Zhang San</name>
</getLoanQuoteRequest>
```

（2）BPEL 引擎可以根据请求消息创建相应的 BPEL 实例，执行 BPEL 流程。例如：

BPEL 在收到上面的服务请求后，根据命名空间“urn:sample:soa:houseloanbroker”和请求消息“getLoanQuoteRequest”，就知道创建和执行该 BPEL 流程 houseloanbroker。

例程 6-1 显示了 BPEL 流程的接口 WSDL 文件。

例程 6-1　houseloanbroker.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:sample:soa:houseloanbroker"
  xmlns:tns="urn:sample:soa:houseloanbroker"
  xmlns:typens="urn:sample:soa:houseloanbroker:types"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <import namespace="urn:sample:soa:houseloanagency" location="houseloanagency.wsdl" />

  <import namespace="urn:sample:soa:bank" location="bank.wsdl" />

  <!-- type defs -->
  <types>
        <xsd:schema
              targetNamespace="urn:sample:soa:houseloanbroker:types"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema">

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

              <xsd:complexType name="getLoanQuoteResponse">
                    <xsd:sequence>
                          <xsd:element name="rate" type="xsd:double" />
                          <xsd:element name="firstpaidratio" type="xsd:double" />
                    </xsd:sequence>
              </xsd:complexType>

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

        </xsd:schema>
  </types>
```

```
<message name="getLoanQuoteRequest">
        <part name="payload" type="typens:getLoanQuoteRequest" />
</message>

<message name="getLoanQuoteResponse">
        <part name="payload" type="typens:getLoanQuoteResponse" />
</message>

<message name="unknownNAMEFault">
        <part name="payload" type="typens:unknownNAMEFault" />
</message>

<portType name="HouseLoanBroker">
        <operation name="getLoanQuote">
                <input message="tns:getLoanQuoteRequest" />
                <output message="tns:getLoanQuoteResponse" />
                <fault name="UnknownNAME" message="tns:unknownNAMEFault" />
        </operation>
</portType>


<plnk:partnerLinkType name="HouseLoanBrokerPL">
        <plnk:role name="HouseLoanBrokerService" portType="tns:HouseLoanBroker" />
</plnk:partnerLinkType>

<binding name="HouseLoanBroker" type="tns:HouseLoanBroker">
        <operation name="request"></operation>
</binding>

<service name="HouseLoanBrokerService">
        <port name="houseloanbroker" binding="tns:HouseLoanBroker" />
</service>
</definitions>
```

6.2.3 外部服务合作伙伴的 WSDL

下面是 BPEL 需要调用的外部服务 houseloanagency 的 WSDL 文件，这个外部服务接收顾客姓名，输出顾客已有的房屋数量。通过这个 WSDL，BPEL 可以知道所要调用的外部服务的接口、接口操作、输入消息和输出消息、异常，这都是 BPEL 流程在调用外部服务时需要知道的，对于 houseloanagency.wsdl，这些参数如下。

- 接口：HouseLoanAgency；
- 接口操作：getHouseNumber；
- 输入消息为下面的树状结构：

```
getHouseNumberRequest
        |——name （String）
```

- 输出消息为下面的树状结构：

```
getHouseNumberResponse
        |——housenumber （int）
```

- 异常：unknownNAMEFault。

此外，BPEL 需要引入该 WSDL，并定义该服务为合作伙伴，所以还需要在此加入合作伙伴的定义。

```
<plnk:partnerLinkType name="HouseLoanAgencyPL">
    <plnk:role name="HouseLoanAgencyService" portType="tns:HouseLoanAgency" />
</plnk:partnerLinkType>
```

上面 portType="tns:HouseLoanAgency"指该合作伙伴所对应的 WSDL 的接口。BPEL 的一个合作伙伴，对应于 WSDL 的一个接口。如果 WSDL 有多个接口，则需要定义多个合作伙伴。

例程 6-2 显示了 houseloanagency.wsdl 的内容。

例程 6-2 houseloanagency.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:sample:soa:houseloanagency"
  xmlns:tns="urn:sample:soa:houseloanagency"
  xmlns:typens="urn:sample:soa:houseloanagency:types"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <types>
      <xsd:schema
          targetNamespace="urn:sample:soa:houseloanagency:types"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">

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

          <xsd:complexType name="getHouseNumberResponse">
              <xsd:sequence>
                  <xsd:element name="housenumber" type="xsd:int" />
              </xsd:sequence>
          </xsd:complexType>

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

```
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:schema>
        </types>

        <message name="getHouseNumberRequest">
            <part name="payload" type="typens:getHouseNumberRequest" />
        </message>

        <message name="getHouseNumberResponse">
            <part name="payload" type="typens:getHouseNumberResponse" />
        </message>

        <message name="unknownNAMEFault">
            <part name="payload" type="typens:unknownSSNFault" />
        </message>

        <portType name="HouseLoanAgency">
            <operation name="getHouseNumber">
                <input message="tns:getHouseNumberRequest" />
                <output message="tns:getHouseNumberResponse" />
                <fault name="UnknownNAME" message="tns:unknownNAMEFault"/>
            </operation>
        </portType>

        <plnk:partnerLinkType name="HouseLoanAgencyPL">
            <plnk:role name="HouseLoanAgencyService" portType="tns:HouseLoanAgency" />
        </plnk:partnerLinkType>
    </definitions>
```

下面是所要调用的 4 个 Bank 的服务，它们的服务接口一样，只是输出结果的首付比例和贷款比例不同，所以采用同一个 WSDL 文件即可，但是在 BPEL 中需要定义 4 个合作伙伴，这 4 个合作伙伴所指向的服务地址是不同的，Bank 的 WSDL 内容如下。

- 接口：Bank；
- 接口操作：getLoanQuote；
- 输入消息为如下树状结构；

```
getLoanQuoteRequest
    |——housenumber（int）
```

- 输出消息为如下的树状结构：

```
getHouseNumberResponse
    |——rate （double）
    |——firstpaidratio （double）
```

此外，BPEL 需要引入该 WSDL，并定义该服务为合作伙伴，所以还需要在此加入合作伙伴定义：

```
    <plnk:partnerLinkType name="BankPL">
        <plnk:role name="BankService" portType="tns:Bank" />
    </plnk:partnerLinkType>
```

例程 6-3 显示了 bank.wsdl 的内容。

例程 6-3　bank.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:sample:soa:bank"
  xmlns:tns="urn:sample:soa:bank"
  xmlns:typens="urn:sample:soa:bank:types"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <types>
      <xsd:schema
          targetNamespace="urn:sample:soa:bank:types"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">

          <xsd:complexType name="getLoanQuoteRequest">
              <xsd:sequence>
                  <xsd:element name="housenumber" type="xsd:int" />
              </xsd:sequence>
          </xsd:complexType>

          <xsd:complexType name="getLoanQuoteResponse">
              <xsd:sequence>
                  <xsd:element name="rate" type="xsd:double" />
                  <xsd:element name="firstpaidratio" type="xsd:double" />
              </xsd:sequence>
          </xsd:complexType>

      </xsd:schema>
  </types>

  <message name="getLoanQuoteRequest">
      <part name="payload" type="typens:getLoanQuoteRequest" />
  </message>

  <message name="getLoanQuoteResponse">
      <part name="payload" type="typens:getLoanQuoteResponse" />
  </message>

  <portType name="Bank">
      <operation name="getLoanQuote">
```

```
            <input message="tns:getLoanQuoteRequest" />
            <output message="tns:getLoanQuoteResponse" />
        </operation>
    </portType>

    <plnk:partnerLinkType name="BankPL">
        <plnk:role name="BankService" portType="tns:Bank" />
    </plnk:partnerLinkType>
</definitions>
```

6.2.4 定义合作伙伴的链接

因为本例需要调用 5 个外部服务 Web Servcie，每个外部服务有一个接口（portType），所以总共需要定义 5 个合作伙伴，它们的分别为 HouseLoanBroker 、Bank1、Bank2、Bank3、Bank4。

这里需要说明的是，尽管 4 个 Bank 的名称不一样，但是它们的 partnerRole 名称和 partnerLinkType 名称是一样的。因为它们都用同样的 WSDL 接口。

下面是合作伙伴的具体内容：

```
<bpel:process name="houseloanbrokerProcess"
  targetNamespace="urn:sample:soa:houseloanbroker"
  xmlns:tns="urn:sample:soa:houseloanbroker"
  xmlns:ca="urn:sample:soa:houseloanagency"
  xmlns:bk="urn:sample:soa:bank"
  xmlns:svc="urn:sample:soa:service" suppressJoinFailure="yes"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sm="http://servicemix.apache.org/schemas/bpe/1.0"
  xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/business-process/
http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/" location="houseloanbroker.wsdl"
                    namespace="urn:sample:soa:houseloanbroker"/>

  <bpel:partnerLinks>
      <bpel:partnerLink myRole="HouseLoanBrokerService" name="HouseLoanBroker"
partnerLinkType="tns:HouseLoanBrokerPL"/>
      <bpel:partnerLink partnerRole="HouseLoanAgencyService" name="HouseLoanAgency"
partnerLinkType="ca:HouseLoanAgencyPL"/>
      <bpel:partnerLink partnerRole="BankService" name=
          "Bank0" partnerLinkType="bk:BankPL"/>
      <bpel:partnerLink partnerRole="BankService" name=
          "Bank1" partnerLinkType="bk:BankPL"/>
      <bpel:partnerLink partnerRole="BankService" name=
          "Bank2" partnerLinkType="bk:BankPL"/>
      <bpel:partnerLink partnerRole="BankService" name=
```

```
                "Bank3" partnerLinkType="bk:BankPL"/>
        </bpel:partnerLinks>
```

6.2.5 声明变量

对应于上面定义的每一个合作伙伴都应该定义一个输入和输出变量，以便调用<invoke>、<receive>、<reply>时用到。因为 Bank 的接口一样，所以对 4 个 Bank 服务只定义一个输入变量；因为有可能要对 Bank 服务的结果进行聚集和比较，所以对每个 Bank 合作伙伴都定义一个输出变量。

本例中所定义的变量如下。

- Request：对应于 BPEL 接口的输入变量；
- Response：对应于 BPEL 接口的输出变量；
- ca-housenumber-request：合作伙伴 HouseLoanAgency 的输入变量；
- ca-housenumber-response：合作伙伴 HouseLoanAgnecy 的输出变量；
- bk-loanquote-request：Bank 的输入变量（对应于所有 Bank）；
- bk-loanquote-response-0：Bank0 的输出变量；
- bk-loanquote-response-1：Bank1 的输出变量；
- bk-loanquote-response-1：Bank2 的输出变量；
- bk-loanquote-response-1：Bank3 的输出变量。

本例所定义的变量程序如下：

```
<bpel:variables>
  <bpel:variable name="request" messageType="tns:getLoanQuoteRequest" />
  <bpel:variable name="response" messageType="tns:getLoanQuoteResponse" />
  <bpel:variable name="ca-housenumber-request" messageType="ca:getHouseNumberRequest" />
  <bpel:variable name="ca-housenumber-response" messageType="ca:getHouseNumberResponse" />
  <bpel:variable name="bk-loanquote-request" messageType="bk:getLoanQuoteRequest" />
  <bpel:variable name="bk-loanquote-response-0" messageType="bk:getLoanQuoteResponse" />
  <bpel:variable name="bk-loanquote-response-1" messageType="bk:getLoanQuoteResponse" />
  <bpel:variable name="bk-loanquote-response-2" messageType="bk:getLoanQuoteResponse" />
  <bpel:variable name="bk-loanquote-response-3" messageType="bk:getLoanQuoteResponse" />
  <bpel:variable name="unknownNAME" messageType="tns:unknownNAMEFault" />
</bpel:variables>
```

6.2.6 声明异常处理

本例中将会定义一个异常处理，它会截获 HouseLoanAgency 所抛出的“unknownNAME”。它表示 HouseLoanAgency 房屋数量查询机构没有该顾客姓名的记录，该顾客姓名不在数据库中。

```
<bpel:faultHandlers>
        <bpel:catch faultName="ca:UnkownNAME">
                <bpel:sequence>
                        <bpel:assign>
                                <bpel:copy>
```

```
                    <bpel:from variable="request" part="payload"
                            query="/tns:getLoanQuoteRequest/tns:name" />
                    <bpel:to variable="unknownNAME" part="payload"
                            query="/tns:unknownNAMEFault/tns:name" />
                </bpel:copy>
            </bpel:assign>
            <bpel:reply name="response" partnerLink="HouseLoanBrokerResponse"
                portType="tns:HouseLoanBroker" operation="getLoanQuote"
                variable="unknownNAME" faultName="tns:unknownNAME" >
            </bpel:reply>
        </bpel:sequence>
    </bpel:catch>
</bpel:faultHandlers>
```

6.2.7 开发 BPEL 流程

下面介绍 BPEL 的流程。

<receive>是整个 BPEL 的起点，它将接收来自客户端的请求消息，程序如下：

```
<bpel:sequence>
    <bpel:receive name="request" partnerLink="HouseLoanBroker"
        portType="tns:HouseLoanBroker" operation="getLoanQuote" variable="request"
        createInstance="yes">
    </bpel:receive>
```

下面将 BPEL 接口收到的输入变量“request”中的“name”参数通过<bpel:assign>赋值给 ca-housenumber-request 变量的“name”参数（这两个变量都是上面刚刚定义的）：

```
<bpel:flow>
  <bpel:sequence>
    <bpel:assign>
        <bpel:copy>
            <bpel:from variable="request" part="payload"
                    query="/tns:getLoanQuoteRequest/tns:name" />
            <bpel:to variable="ca-housenumber-request" part="payload"
                    query="/ca:getHouseNumberRequest/ca:name" />
        </bpel:copy>
    </bpel:assign>
```

下面通过合作伙伴的名称 HouseLoanAgency 调用其接口的接口操作“getHouseNumber”，并且以上面刚刚得到赋值的 ca-housenumber-request 作为输入变量 inputVariable：

```
            <bpel:invoke name="service" partnerLink="HouseLoanAgency"
                portType="ca:HouseLoanAgency" operation="getHouseNumber"
                inputVariable="ca-housenumber-request"
                outputVariable="ca-housenumber-response" />
        </bpel:sequence>
    </bpel:flow>
```

下面将<invoke>得到的房屋数量输出变量赋值给上面定义的 Bank 服务的输入变量：

```
<bpel:assign>
  <bpel:copy>
        <bpel:from variable="ca-housenumber-response" part="payload"
                                        query="/ca:getHouseNumberResponse/ca:housenumber" />
        <bpel:to variable="bk-loanquote-request" part="payload"
                                        query="/bk:getLoanQuoteRequest/bk:housenumber"/>
  </bpel:copy>
</bpel:assign>
```

下面是 BPEL 的分支，即房屋数量为 0 时的一个分支：

```
<bpel:switch>
  <bpel:case condition="getVariableData('bk-loanquote-request', 'payload',
                                        '/bk:getLoanQuoteRequest/bk:housenumber') = 0 ">
        <bpel:sequence>
            <bpel:flow>
```

下面表示房屋数量为 0 时，将会调用 Bank0 的服务：

```
<bpel:invoke name="bank0" partnerLink="Bank0"
        portType="bk:Bank" operation="getLoanQuote"
        inputVariable="bk-loanquote-request"
        outputVariable="bk-loanquote-response-0"
        sm:endpoint="urn:logicblaze:soa:bank:Bank0:bank" />
</bpel:flow>
```

将 Bank0 输出变量中的 rate 参数赋值给 BPEL 接口输出变量 response 的 rate 参数：

```
<bpel:assign>
  <bpel:copy>
        <bpel:from expression="getVariableData
          ('bk-loanquote-response-0',
                        'payload', '/bk:getLoanQuoteResponse/
                        bk:rate')" />
        <bpel:to variable="response" part="payload"
                        query="/tns:getLoanQuoteResponse/
                        tns:rate" />
  </bpel:copy>
```

将 Bank0 输出变量中的 firstpaidratio 赋值给 BPEL 输出变量的 firstpaidratio：

```
<bpel:copy>
  <bpel:from expression="getVariableData
    ('bk-loanquote-response-0',
                'payload', '/bk:getLoanQuoteResponse/
                bk:firstpaidratio')" />
  <bpel:to variable="response" part="payload"
                query="/tns:getLoanQuoteResponse/
                tns:firstpaidratio" />
</bpel:copy>
```

```
            </bpel:assign>
        </bpel:sequence>
    </bpel:case>
```

下面省略房屋数量为 1 的 BPEL 分支情形代码，因为业务一样、程序一样，将会调用 Bank1 的服务，将 Bank 输出变量中的 rate 和 firstpaidratio 参数赋值给 BPEL 接口输出变量 response 的 rate 和 firstpaidratio 参数：

```
    <bpel:case condition="getVariableData('bk-loanquote-request', 'payload',
                          '/bk:getLoanQuoteRequest/
                          bk:housenumber') = 1 " >
      <bpel:sequence>
        ……
      </bpel:sequence>
    </bpel:case>
```

下面省略房屋数量为 2 的 BPEL 分支情形，过程同上。

```
    <bpel:case condition="getVariableData('bk-loanquote-request', 'payload',
                          '/bk:getLoanQuoteRequest/
                          bk:housenumber') = 2 ">
      <bpel:sequence>
            ……
      </bpel:sequence>
    </bpel:case>
```

下面是用<otherwise>来实现 BPEL 条件的最后一个分支，房屋数量大于 2 时的一个分支。

```
    <bpel:otherwise>
      <bpel:sequence>
          <bpel:flow>
```

在默认情形下将会调用 Bank3 的服务：

```
        <bpel:invoke name="bank3" partnerLink="Bank3"
            portType="bk:Bank" operation="getLoanQuote"
            inputVariable="bk-loanquote-request"
            outputVariable="bk-loanquote-response-3"
            sm:endpoint="urn:logicblaze:soa:bank:Bank3:bank" />
    </bpel:flow>
```

将 Bank3 输出变量中的 rate 参数赋值给 BPEL 接口输出变量 response 的 rate 参数：

```
    <bpel:assign>
      <bpel:copy>
          <bpel:from expression="getVariableData
            ('bk-loanquote-response-3',
                          'payload', '/bk:getLoanQuoteResponse/
                          bk:rate')" />
          <bpel:to variable="response" part="payload"
```

```
                                query="/tns:getLoanQuoteResponse/
                                tns:rate" />
        </bpel:copy>
```

将 Bank3 输出变量中的 firstpaidratio 赋值给 BPEL 输出变量 response 的 firstpaidratio：

```
                        <bpel:copy>
                            <bpel:from expression="getVariableData
                                ('bk-loanquote-response-3',
                                'payload', '/bk:getLoanQuoteResponse/
                                bk:firstpaidratio')" />
                            <bpel:to variable="response" part="payload"
                                query="/tns:getLoanQuoteResponse/
                                tns:firstpaidratio" />
                        </bpel:copy>
                    </bpel:assign>
                </bpel:sequence>
            </bpel:otherwise>
        </bpel:switch>
```

<reply>是 BPEL 的结束，将变量结果返回非服务请求者，程序如下：

```
            <bpel:reply name="response" partnerLink="HouseLoanBroker"
                portType="tns:HouseLoanBroker" operation="getLoanQuote"
                variable="response" />


        </bpel:sequence>
    </bpel:process>
```

6.3 BPEL过程组件

BPEL 流程的本质就是通过 BPEL 的活动（Activity）将 BPEL 的过程组件“串起来”。所谓“串起来”就是通过 BPEL 的活动让 BPEL 的过程组件产生一种动态的交互。比如说将一个变量赋值给另外一个变量，就是一种变量之间的动态的交互。

BPEL 流程中经常用到的一些组件如下：

- 合作伙伴连接（Partner Links）；
- 变量 Varaibles；
- 相关集 Correlation Sets；
- 错误处理 Fault handles；
- 补偿处理（Compensation Handlers）等。

6.3.1 合作伙伴连接（Partner Links）

合作伙伴连接（Partner Links）是指在 BPEL 中的服务提供者，它主要分为两种，一种是 BPEL 流程所要调用的外部服务；另一种是指 BPEL 自己所要提供的服务。

定义合作伙伴连接（Partner Links）实际上包括定义两件事情：

- 在 BPEL 中定义<partnerLink>
- 在对应的 WSDL 中定义<partnerLinkType>

BPEL 首先需要定义合作伙伴的<partnerLink>，然后需要在所要引入的 WSDL 中定义所对应的 WSDL 接口 PortType。

1．在 BPEL 中定义<partnerLink>

如上面的实例在 BPEL 文件中定义 Bank0 这么一个 partnerLink：

```
<bpel:partnerLink partnerRole="BankService" name="Bank0"
    partnerLinkType="bk:BankPL"/>
```

如果是外部服务，需要采用 partnerRole 来定义合作伙伴类型，如上例中：

```
<bpel:partnerLink partnerRole="BankService" name="Bank0"
    partnerLinkType="bk:BankPL"/>
```

如果是 BPEL 自己的接口，则通过 myRole 来定义了一个合作伙伴，它表示这个服务接口是 BPEL 自己提供服务的，如前面实例：

```
<bpel:partnerLink myRole="HouseLoanBrokerService" name="HouseLoanBroker"
                  partnerLinkType="tns:HouseLoanBrokerPL"/>
```

2．在对应的 WSDL 中定义< partnerLinkType >

为了使合作伙伴能够对应相应的 WSDL 的接口，需要在对应的 bank.wsdl 里面通过<partnerLinkType>定义它所对应的 WSDL 的 portType：

```
<plnk:partnerLinkType name="BankPL">
    <plnk:role name="BankService" portType="tns:Bank" />
</plnk:partnerLinkType>
```

上面就将 BPEL 的 partnerLinkTyp（BankPL）和 WSDL 的 portType（Bank）对应上了。

BPEL 是通过合作伙伴的名字来调用<invoke>外部服务的，BPEL 引擎也是通过合作伙伴的名字来和真正的服务提供者进行绑定的，如前面的实例将通过“Bank0”来调用外部的 Bank0 服务。

```
<bpel:invoke name="bank0" partnerLink="Bank0"
             portType="bk:Bank" operation="getLoanQuote"
             inputVariable="bk-loanquote-request"
             outputVariable="bk-loanquote-response-0"
             sm:endpoint="urn:sample:soa:bank:Bank0:bank" />
```

6.3.2 变量 Variables

BPEL 里面的变量 Varaibles 类似于 Java 的变量的概念，是一个保存数据的地方。但是 BPEL 和 Java 语言不太一样，Java 语言的参数可以直接调用，比方说，Java 程序的其他函数接口可以直接调用上面的 double parameter1。但是作为 BPEL 语言是不行的。因为 BPEL 需要集成 WSDL 定义的 Web Service。同一个参数名字在不同的 WSDL 中参数是不一样的，因为 WSDL 是一种 XML 结构，其本质是一种树状的数据结构，直接赋值是赋不进去的，所以需要先定义变量。一般来说对每一个合作伙伴，不管有多少输入参数和输出参数，定义两个变量即可，一个请求（request）变量，一个响应变量。

```
<bpel:variables>
        <bpel:variable name="request" messageType="tns:getLoanQuoteRequest" />
</bpel:variables>
```

上面房屋贷款中的变量实例，可以通过 messageType 将 WSDL 文件中的变量类型取过来，如上面的变量“request”实际上对应 houseloanbroker.wsdl 所定义的变量，结构如下：

```
getLoanQuoteRequest
                |——name（String）
```

这样，通过 messageType 可以直接定义变量的数据结构。

6.3.3 相关集（Correlation Sets）

相关集（Correlation Sets）用一组特定的数据，来关联和标定一个 BPEL 过程实例。因为本质上 BPEL 是一个分布式的消息系统，不像分布式的对象系统，可以通过 Instance ID 来标识实例。BPEL 是通过交换消息里面的一些属性值来标识的。而相关集可以不需要 Instance ID 就可以识别 BPEL 过程实例，对许多需要保持会话状态的 BPEL 业务需求来说，是非常有用的。

下面用一个日常生活中的实例进一步说明这个问题。

假设一个小学一个班级有 30 个学生。小学校长想知道每个学生的数学、语文和英语的成绩。小学校长准备了 30 个空白信封，每个信封里面有一张白纸让相应的老师填入成绩。

1．没有相关集的情形

小学校长直接将 30 个空白信封先交给了数学老师，数学老师在信封里的白纸上填好数学成绩交回给校长。

小学校长接着将这 30 个信封交给语文老师，希望在原有数学成绩的基础上再填上语文成绩。语文老师没法完成此工作，因为信封上没有标志，语文老师无法知道哪个学生对应哪个信封。

2．有相关集的情形

小学校长首先将 30 个空白信封上标上学生的姓名和学生证号码，交给数学老师，数学老师就可以根据信封上的标志分别填上每个学生相应的数学成绩，然后交给语文老师；语文老师就可以根据信封上的标志分别填上每个学生相应的语文成绩，然后交给英语老师；英语老师就可以根据信封上的标志分别填上每个学生相应的英语成绩。这样每个学生的各科成绩就都填进去了。

上面实例中的信封相当于 BPEL 的一个实例，信封上的学生的姓名和学生证号码相当于一个相关集。

在 BPEL 中设置相关集比较复杂，下面以房屋贷款的实例来说明在 BPEL 中设置相关集的过程思路，下面以顾客姓名“name”数据建立相关集。

首先需要建立“相关属性”如下：

```
<definitions name="properties"
targetNamespace="http://example.com/houseloanCorrelation.wsdl"
xmlns:tns="http://example.com/ houseloanCorrelation.wsdl"
```

```
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<!-- define correlation properties -->
<bpws:property name="name" type="xsd:string"/>
</definitions>
```

接下来需要在 WSDL 中定义“相关属性”。因为同一个属性名，比如“name”，它可能是请求变量的参数，也可能是响应变量的参数，还有路径的关系，因为变量在 SOAP 消息中是树状的数据结构。所以这里需要用<bpws:propertyAlias>标签从请求消息中得到“name”值。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace=" http://example.com/messageCorrelation.wsdl "
 xmlns:tns=" http://example.com/messageCorrelation.wsdl "
xmlns:cor="http://example.com/houseloanCorrelation.wsdl"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 ……
 xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
 <types>
        <xsd:schema>
                <xsd:complexType name="getLoanQuoteRequest">
                        <xsd:sequence>
                                <xsd:element name="name" type="xsd:string" />
                        </xsd:sequence>
                </xsd:complexType>
……
        </xsd:schema>
 </types>

 <message name="getLoanQuoteRequest">
        <part name="payload" type="typens:getLoanQuoteRequest" />
 </message>

 <message name="getLoanQuoteResponse">
        <part name="payload" type="typens:getLoanQuoteResponse" />
 </message>

  ……
<bpws:propertyAlias propertyName="cor:name"
messageType="tns: getLoanQuoteRequest " part=" payload "
query="/ getLoanQuoteRequest /name"/>

  ……
</definitions>
```

接下来在 BPEL 中定义相关集：

```
<correlationSets    xmlns:cor="http://example.com/houseloanCorrelation.wsdl">
```

```
<correlationSet name="HouseLoanCor"    properties="cor:name "/>
</correlationSets>
```

进而在 BPEL 的< receive>活动中调用相关集：

```
<bpel:receive name="request" partnerLink="HouseLoanBroker"
        portType="tns:HouseLoanBroker" operation="getLoanQuote" variable="request">
        <correlations>
        <correlation set=" HouseLoanCor " initiate="yes">
        </correlations>
</bpel:receive>
```

这样就将房屋贷款的流程实例和相关集关联起来了。

6.3.4 错误处理（Fault Handlers）

BPEL 程序在调用合作伙伴的过程中，或者合作伙伴的服务有可能会抛出异常，或者在 BPEL 流程内部调用中也会抛出异常（如将字符串 String 类型的变量赋值给整数类型 int 的变量）。BPEL 流程有一套机制可以捕获异常，并可以定义捕获异常后的下一步行动。如本例中定义了一个“unknownNAME”（系统中无此顾客姓名的异常），BPEL 流程在捕获此异常后，通过<reply>返回给服务请求者。

需要说明的是，这个“unknownNAME”必须事先在 BPEL 的接口 WSDL 中定义，本例中的定义如下：

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

而且合作伙伴返回的异常必须符合上面的格式，才能捕获这个异常，否则是不会捕获这类异常的。合作伙伴返回的异常实例如下：

```
<InvalidNAME xmlns="urn:sample:soa:houseloanagency">
<name>Zhang San1</name>
</InvalidNAME>
```

下面是 BPEL 异常处理的写法：

```
<bpel:faultHandlers>
        <bpel:catch faultName="ca:UnkownNAME">
                <bpel:sequence>
                        <bpel:assign>
                                <bpel:copy>
                                        <bpel:from variable="request" part="payload"
                                                query="/tns:getLoanQuoteRequest/tns:name" />
                                        <bpel:to variable="unknownNAME" part="payload"
                                                query="/tns:unknownNAMEFault/tns:name" />
                                </bpel:copy>
                        </bpel:assign>
                        <bpel:reply name="response" partnerLink="HouseLoanBrokerResponse"
```

```
                                portType="tns:HouseLoanBroker" operation="getLoanQuote"
                                variable="unknownNAME" faultName="tns:unknownNAME" >
                            </bpel:reply>
                        </bpel:sequence>
                    </bpel:catch>
                </bpel:faultHandlers>
```

6.3.5 补偿处理（Compensation Handlers）

BPEL 流程需要处理一系列的业务流程活动，业务流程往往有这样的业务需求，如果某一个活动出现异常后，希望它前面的活动也能够取消。

以上面的房屋贷款的实例来说，假如房屋贷款的 BPEL 流程已经调用了外部银行 Bank 的贷款服务，得到了相应的首付和贷款利率，银行那边也有了相应的记录；但是 BPEL 流程将 Bank 的响应结果返回给客户时，出现了异常。这个时候 BPEL 流程就需要通知 Bank 取消相应的银行贷款记录。这就需要用到 BPEL 的补偿处理机制。

继续以房屋贷款为例来说明这个问题。

补偿处理机制是在一定的范围内进行的，因为要对银行贷款服务进行补偿，先用<scope>定义一个范围。此外，需要银行不仅有前面的“getLoanQuote”的贷款功能的接口操作，同时 Bank 也需要提供一个取消房屋贷款的接口操作，这里假设已经有了这么一个接口操作，名为“canelLoanQuote”。

定义补偿的具体过程是先定义一个<scope>，此处为"bank-loan-scope"；再定义<compensationHandler>，并在里面定义相应的活动，如这里的活动为<invoke>取消贷款的操作（Operation）="canelLoanQuote"，程序如下所示：

```
<scope name="bank-loan-scope">
    <compensationHandler>
        <bpel:invoke name="bank0" partnerLink="Bank0"
          portType="bk:Bank" operation="canelLoanQuote"
          inputVariable="bk-loanquote-request"
          outputVariable="bk-loanquote-response-0"
          sm:endpoint="urn:sample:soa:bank:Bank0:bank" >
</bpel:invoke>
    </compensationHandler>

  <bpel:invoke name="bank0" partnerLink="Bank0"
          portType="bk:Bank" operation="getLoanQuote"
          inputVariable="bk-loanquote-request"
          outputVariable="bk-loanquote-response-0"
          sm:endpoint="urn:sample:soa:bank:Bank0:bank" >
</bpel:invoke>
</scope>
```

此外需要在 BPEL 的接口中定义 WSDL 返回消息给客户时的异常，假设为“ReplyFault”，当 BPEL 流程截获这个异常时，将会调用所表明范围内的补偿机制，补偿机制会进一步调用银行的取消贷款的操作，下面是 BPEL 的异常处理程序片段。

```
<bpel:faultHandlers>
    <bpel:catch faultName="ca: ReplyFault ">
        <bpel:sequence>
            <compensate scope="bank-loan-scope"/>
        </bpel:sequence>
    </bpel:catch>
</bpel:faultHandlers>
```

6.4 BPEL活动（Activity）

BPEL 活动（Activity）是指 BPEL 流程中一条语句或者一个步骤的执行。BPEL 常用的一些基本活动如下：

- <assign>/<invoke>（赋值/调用）；
- <condition>/<otherwise>（条件/否则）；
- <sequence>/<flow>（顺序/并行）；
- <link>/<source>/<target>（链接/源/目标）；
- <pick>/< onMessage>/ <onAlarm>（选择/监听/闹钟）。

下面将予以详细的介绍。

6.4.1 <receive> /<reply>（接收/恢复）

接收<receive>是整个 BPEL 的起点，一旦 BPEL 引擎从客户端接收到请求消息，它将会启动一个 BPEL 的流程。

<receive>的属性如下。

- name：定义 receive 的名称，此处为“request”。
- partnerLink：对应于 BPEL 流程定义的 partnerLink 的名字，实例如下。

```
<bpel:partnerLink myRole="HouseLoanBrokerService" name="HouseLoanBroker"
                  partnerLinkType="tns:HouseLoanBrokerPL"/>
```

- portType：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口 portType。
- operation：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口操作（Operation）。
- variable：variable="request"表示客户端的请求消息将会被赋值到所定义的变量“request”中。
- createInstance：createInstance="yes"表示 BPEL 将会创建一个新的实例

下面是房屋贷款的 receive 实例：

```
<bpel:receive name="request" partnerLink="HouseLoanBroker"
  portType="tns:HouseLoanBroker" operation="getLoanQuote" variable="request"
  createInstance="yes">
</bpel:receive>
```

恢复<reply>是整个 BPEL 的终点，BPEL 流程将会把响应结果返回给服务请求者。
< reply >的属性如下：

- name：定义 receive 的名称，此处为“response”。
- partnerLink：对应于 BPEL 流程定义的 partnerLink 的名字；

```
<bpel:partnerLink myRole="HouseLoanBrokerService"
    name="HouseLoanBroker" partnerLinkType="tns:HouseLoanBrokerPL"/>
```

- portType：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口 portType。
- operation：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口操作（Operation）。
- variable：variable=" response "表示 BPEL 流程会将结果变量“response”返回给服务请求者。

下面是房屋贷款的 reply 的实例：

```
<bpel:reply name="response" partnerLink="HouseLoanBroker"
        portType="tns:HouseLoanBroker" operation="getLoanQuote"
        variable="response" />
```

6.4.2 <assign> /<invoke>（赋值/调用）

<assign>赋值和<invoke>调用经常是一前一后的两个任务。这是因为在调用合作伙伴的服务时，需要先给合作伙伴的接口的输入变量赋值，然后再调用服务。

下面是 assign 的写法实例，它主要是要定义两个元素，一个是<from>，一个<to>；其过程就是要将<from>变量下的某个参数复制到<to>变量下的某个参数中。值得注意的是，两个参数的类型必须一致。此外，不能将整个变量复制给另一变量，只能复制变量下面的参数。

```
<bpel:assign>
  <bpel:copy>
        <bpel:from variable="ca-housenumber-response" part="payload"
                                        query="/ca:getHouseNumberResponse/ca:housenumber" />
        <bpel:to variable="bk-loanquote-request" part="payload"
                                        query="/bk:getLoanQuoteRequest/bk:housenumber"/>
  </bpel:copy>
</bpel:assign>
```

<invoke>调用是 BPEL 中最重要的活动，其他所有的活动都是为了辅助整个活动，它将会调用合作伙伴的服务，得到返回结果。

<invoke>的属性如下：

- name：定义 invoke 的名称，此处为“bank0”。
- partnerLink：对应于 BPEL 流程定义的 partnerLink 的名字，实例如下：

```
<bpel:partnerLink partnerRole="BankService" name="Bank0"
                        partnerLinkType="bk:BankPL"/>
```

- portType：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口 portType。
- operation：对应于合作伙伴链接中 partnerLinkType 所定义的 WSDL 的接口操作（Operation）。
- inputVariable：表示在调用合作伙伴服务之前，事先定义并且已经赋值的输入变量。
- outputVariable：存储合作伙伴服务调用结果的输出变量。

下面是房屋贷款的 invoke 的实例：

```
<bpel:invoke name="bank0" partnerLink="Bank0"
    portType="bk:Bank" operation="getLoanQuote"
    inputVariable="bk-loanquote-request"
    outputVariable="bk-loanquote-response-0"
    sm:endpoint="urn:logicblaze:soa:bank:Bank0:bank" />
```

6.4.3 <condition> /<otherwise>（条件/否则）

<condition> /<otherwise>（条件/否则）是 BPEL 中的条件语句。因为 BPEL 是流程管理的程序语言，因此它需要根据不同的条件来调用不同的合作伙伴的服务。<condition> /<otherwise>一般会一起使用，<otherwise>表示上述条件都不满足时的下一步活动。

<condition>通常会调用 getVariableData()来得到变量中某一路径下的参数值，再和常量或者其他变量的参数值进行比较。

下面是房屋贷款中<condition> /<otherwise>的实例。

首先是房屋数量为 0 时的情形的一个分支，调用 Bank0 的服务。

```
<bpel:switch>
<bpel:case condition="getVariableData('bk-loanquote-request', 'payload',
                                      '/bk:getLoanQuoteRequest/bk:housenumber') = 0 ">
        <bpel:invoke name="bank0" partnerLink="Bank0"
                                      portType="bk:Bank" operation="getLoanQuote"
                                      inputVariable="bk-loanquote-request"
                                      outputVariable="bk-loanquote-response-0"
                                      sm:endpoint="urn:logicblaze:soa:bank:Bank0:bank" />
 </bpel:case>
```

最后用<otherwise>来实现 BPEL 条件的最后一个分支，房屋数量大于 2 时的一个分支。

```
<bpel:otherwise>
        <bpel:invoke name="bank3" partnerLink="Bank3"
                portType="bk:Bank" operation="getLoanQuote"
                inputVariable="bk-loanquote-request"
                outputVariable="bk-loanquote-response-3"
                sm:endpoint="urn:logicblaze:soa:bank:Bank3:bank" />
</bpel:otherwise>
</bpel:switch>
```

6.4.4 <sequence> /<flow>（顺序/并行）

<sequence>表示两个活动是按顺序一前一后地进行，前面的活动不完成，后面的活动不开始。<flow>表示两个或者多个活动可以并行。

下面是<sequence>的实例，表示必须<receive>完成之后，<reply>才能开始，程序如下：

```
<bpel:sequence>
    <bpel:receive name="request" partnerLink="HouseLoanBroker"
        portType="tns:HouseLoanBroker" operation="getLoanQuote" variable="request"
        createInstance="yes">
    </bpel:receive>

    <bpel:reply name="response" partnerLink="HouseLoanBroker"
        portType="tns:HouseLoanBroker" operation="getLoanQuote"
        variable="response" />
</bpel:sequence>
```

下面是<flow>的实例，如果按照下面的写法，表示可以同时调用 Bank0 和 Bank1 的服务，程序如下：

```
<bpel:flow>
    <bpel:invoke name="bank0" partnerLink="Bank0"
                portType="bk:Bank" operation="getLoanQuote"
                inputVariable="bk-loanquote-request"
                outputVariable="bk-loanquote-response-0"
                sm:endpoint="urn:sample:soa:bank:Bank0:bank" />

    <bpel:invoke name="bank1" partnerLink="Bank1"
                portType="bk:Bank" operation="getLoanQuote"
                inputVariable="bk-loanquote-request"
                outputVariable="bk-loanquote-response-1"
                sm:endpoint="urn:sample:soa:bank:Bank1:bank" />
</bpel:flow>
```

6.4.5 <link> /<source>/<target>(链接/源/目标)

对于比较复杂的并行的流程，通过链接的定义使得流程能从一个活动跳转到另一个活动。每个<link>链接都会定义一个<source>源和一个<target>目标；如果链接<source>源的转换条件满足的话，将会跳到链接<target>源所在的活动；可以通过 transitionCondition 来给<source>加上转换条件，如果不加条件，那么认为是 true。

下面将房屋贷款的实例做一点改造来说明这个用法。首先定义两个链接“invoke-bank0”和“invoke-bank1”，它们的<source>源都放在<bpel:assign>的活动中。并通过 transitionCondition 分别给这两个链的源接加上跳转的条件。如果 invoke-bank0 的源的条件 transitionCondition 满足的话（即房屋数量为 0），它将会跳转到 invoke-bank0 的目标所在的活动，即 invoke Bank0 的活动；同理，如果 invoke-bank1 的源的 transitionCondition 转换条件（即房屋数量为 1）满足的话，它将会跳转到 invoke-bank1 的目标所在的活动，即 invoke

Bank1 的活动。

```
<bpel:flow>

    <links>
      <link name="invoke-bank0"/>
      <link name="invoke-bank1"/>
    </links>

  <bpel:assign>
        <bpel:copy>
                <bpel:from variable="ca-housenumber-response" part="payload"
                        query="/ca:getHouseNumberResponse/ca:housenumber" />
                <bpel:to variable="bk-loanquote-request" part="payload"
                        query="/bk:getLoanQuoteRequest/bk:housenumber"/>
        </bpel:copy>

        <source linkName="invoke-bank0"
                transitionCondition="getVariableData('bk-loanquote-request', 'payload',
                        '/bk:getLoanQuoteRequest/bk:housenumber') = 0 "/>

         <source linkName="invoke-bank1"
                transitionCondition="getVariableData('bk-loanquote-request', 'payload',
                        '/bk:getLoanQuoteRequest/bk:housenumber') = 1 "/>
</bpel:assign>

<bpel:invoke name="bank0" partnerLink="Bank0"
   portType="bk:Bank" operation="getLoanQuote"
   inputVariable="bk-loanquote-request"
   outputVariable="bk-loanquote-response-0"
   sm:endpoint="urn:sample:soa:bank:Bank0:bank" >

   <target linkName="invoke-bank0"/>

</bpel:invoke>

<bpel:invoke name="bank1" partnerLink="Bank1"
     portType="bk:Bank" operation="getLoanQuote"
     inputVariable="bk-loanquote-request"
     outputVariable="bk-loanquote-response-1"
     sm:endpoint="urn:sample:soa:bank:Bank1:bank" >

     <target linkName="invoke-bank1"/>
  </bpel:invoke>
</flow>
```

6.4.6 <pick> /< onMessage>/ <onAlarm>（选择/监听/闹钟）

前面所定义的<receive>是整个 BPEL 流程的一个起点，BPEL 引擎处于一种等待接收客户消息的状态。

BPEL 也可以提供另外一种功能，它可以把接收请求消息作为一种活动来处理，请求消息这个活动只是在某段时间有效，过了这段时间，这个请求消息活动就停止了。

这种情形在现实生活中很多。我们经常可以看到“报名从什么时间开始，截止到什么时间”等。BPEL 是通过<pick>、<onMessage>、<onAlarm>.来实现此功能的。

下面是修改的房屋贷款的实例。

<onAlarm for="'P15DT10H' ">先将报警闹钟设为 15 天 10 小时。在这个时间之内，<onMessage>都会接收房屋贷款的请求消息，调用 Bank 服务，将结果返回给客户。一旦过了此时间，就不会再接收房屋贷款的请求消息了。

```
<bpel:pick createInstance="no" name="house-loan">
        <bpel:onMessage name="request"
                        partnerLink="HouseLoanBroker"
                        portType="tns:HouseLoanBroker"
                        operation="getLoanQuote"
                        variable="request">
            ……
            <bpel:invoke name="bank0" partnerLink="Bank0"
                         portType="bk:Bank" operation="getLoanQuote"
                         inputVariable="bk-loanquote-request"
                         outputVariable="bk-loanquote-response-0"
                         sm:endpoint="urn:sample:soa:bank:Bank0:bank" />
            ……
            <bpel:reply name="response" partnerLink="HouseLoanBroker"
                        portType="tns:HouseLoanBroker" operation="getLoanQuote"
                        variable="response" />
        </bpel:onMessage>
        <onAlarm for="'P15DT10H' ">
            <empty/>
        </onAlarm>
</bpel:pick>
```

6.5 小结

本章首先介绍了 BPEL 的基本思想，它是一种新的流程执行语言，能够将 Web Service 给串起来。它通过 WSDL 给出的 portType 接口、接口操作、输入变量、输出变量来实现对 Web Service 的操作。它通过合作伙伴来指向外部服务，合作伙伴并不是真正的 Web Service，里面没有绑定信息，合作伙伴的绑定需要 BPEL 运行环境来实现，因为 BPEL 实现了抽象的 WSDL 接口的集成，所以它也属于 SOA 的解决方案之一。

接下来通过房屋贷款的实例详细介绍了 BPEL 的实现过程，包括定义 BPEL 的接口、

定义合作伙伴的连接，声明变量，声明异常处理，开发 BPEL 流程等。

接下来详细介绍了 BPEL 过程组件的概念和写法，如合作伙伴连接、变量、相关集、错误处理、补偿处理等。合作伙伴连接主要分两种，一种属于外部服务，另一种是 BPEL 自己提供的服务；各种组件之间值的传递是通过变量来完成的，BPEL 有保存变量值的功能。相关集主要通过一组属性值来标识一个 BPEL 实例。错误处理是 BPEL 收到异常和错误后如何处理。补偿处理主要是发生异常后，BPEL 可以取消以前的操作，则需要外部服务提供取消服务的功能。

最后介绍了 BPEL 各种活动。<receive>/<reply>（接收/恢复）一般是 BPEL 的流程一个起点和终点；<assign> /<invoke>（赋值/调用）一般会一起使用，如在外部服务之前，需要先给外部服务的输入变量赋值；<condition> /<otherwise>（条件/否则）是 BPEL 的条件语句；<sequence> /<flow>（顺序/并行）用来定义流程的执行方式，顺序执行或者并行；<link> /<source>/<target>（链接/源/目标）一般用于比较复杂的并行的流程。<pick> /< onMessage>/ <onAlarm>（选择/监听/闹钟）可以用来监听消息，并且设置定时机制。