<BPEL 简明开发手册>

版本 <0.1>

目录

1. 前言 3

1.1 关于BPEL简明开发手册 3
1.2 名词解释 3

2. BPEL背景知识 4

3. 与WSDL的关系 4

4. 定义业务流程 5

4.1 BPEL实例教程 5

5. 业务流程的结构 18

5.1 <基元活动 --- BASIC ACTIVITY> 21
5.1.1 <Activity –invoke> 21
5.1.2 <Activity –receive〉 22
5.1.3 <Activity –reply〉 23
5.1.4 <Activity –assign〉 24
5.1.5 <Activity –throw〉 25
5.1.6 <Activity –wait〉 25
5.1.7 <Activity –empty〉 25
5.1.8 <Activity –terminate〉 26
5.1.9 <Activity –compensate〉 26
5.2 <结构活动 --- STRUCTURE ACTIVITY> 27
5.2.1 <Activity –sequence〉 27
5.2.2 <Activity –switch〉 27
5.2.3 <Activity –while〉 28
5.2.4 <Activity –flow〉 28
5.2.5 <Activity –pick〉 29
5.3 <特殊活动 --- SCOPE> 31

6. 合作伙伴链接类型、合作伙伴、服务引用 33

6.1 合作伙伴链接 33
6.2 伙伴链接 34
6.3 服务引用 34

7. 参考手册 36

BPEL 简明开发手册

1. 前言

1.1 关于 BPEL 简明开发手册

本手册是针对 SIKA 用户编写的 BPEL 简明开发手册，对其它开发人员学习 BPEL 也提供了一个相对容易的入门的学习手册。本手册是参考 IBM 官方网站的 bpel v1.1 规范、ORACLE 中国网站及其它网络上有关 bpel 资源，以我们的理解方式整理出来的。我们不对其最终的正确定以负责。不过，参考本手册开发出的 bpel 流程文件适合在 SIKA 系统的执行。

1.2 名词解释

1. BPEL：Business Process Execution Language，Web 服务的业务流程执行语言，是一种使用 Web 服务定义和执行业务流程的语言。
2. WSDL :Web 服务描述语言（Web Services Description Language，WSDL）。
3.

2. BPEL 背景知识

BPEL 规范是由 Microsoft、IBM、BEA 在各自的一套规范基础上共同制定和推广的，BPEL 基于 XML 和 Web 服务构建；它使用一种基于 Web 的语言，该语言支持 web 服务技术系列，包括 SOAP、WSDL、UDDI、Web 服务可靠性消息、Web 服务寻址、Web 服务协调以及 Web 服务事务。

BPEL 代表了两种早期工作流语言---WSFL 和 XLANG 的交汇（都是 Web 服务语言）。WSFL 由 IBM 基于有向图概念设计。XLANG 是一种由 Microsoft 设计的块结构化语言。BPEL 组合了这两种方法，并提供了丰富的词汇来描述业务流程。

BPEL 的第一个版本诞生于 2002 年 8 月。此后，随着许多主要供应商（包括 Oracle）的纷纷加入了，催生了多项修改和改进，并于 2003 年 3 月推出了 1.1 版。2003 年 4 月，BPEL 提交结构化信息标准促进组织（OASIS）以实现标准化，并组建了 Web 服务业务流程执行语言技术委员会（WSBPEL TC）。该委员会努力使 BPEL 在业界获得更广范围的认可。

在企业内部，BPEL 用于标准化企业应用程序集成以及将此集成扩展到先前孤立的系统。在企业之间，BPEL 使与业务合作伙伴的集成变得更容易、更高效。BPEL 激发企业进一步定义它们的业务流程，从而导致业务流程的优化、重新设计以及选择合适的流程，进而实现组织的进一步优化。

BPEL 中描述的业务流程定义并不影响现有系统，因此对升级产生了促进作用。在已经或将要通过 Web 服务公开功能的环境中，BPEL 是一项重要的技术。随着 Web 服务的不断普及，BPEL 的重要性也随之提高。

3. 与 WSDL 的关系

BPEL4WS 依赖于以下基于 XML 的规范：WSDL 1.1、XML Schema 1.0 和 XPath 1.0。在这些规范中，WSDL 对 BPEL4WS 语言的影响最大。BPEL4WS 流程模型位于由 WSDL 1.1 所定义的服务模型之上。位于 BPEL4WS 流程模型核心的是由 WSDL 描述的服务间的对等交互概念；流程及其伙伴都被建模成 WSDL 服务。业务流程定义了怎样协调流程实例与它的伙伴间的交互。在这个意义上，一个 BPEL4WS 流程定义提供和 / 或使用一个或多个 WSDL 服务，还通过 Web 服务接口提供流程实例相对于它的伙伴和资源的行为和交互的描述。也就是说，BPEL4WS 定义了交互中某个角色的业务流程遵守的消息交换协议。

BPEL4WS 业务流程的定义也遵循 WSDL 的分离模型，即把业务流程使用的抽象消息内容与部署信息（消息和 portType 与绑定和地址信息）分开。具体地说，BPEL4WS 流程用抽象 WSDL 接口（portType 和操作）来表示所有的伙伴以及与这些伙伴的交互；它并不引用流程实例使用的实际服务。BPEL4WS 流程是可重用的定义，可以不同的方式在不同的情况下被部署同时在它们之间保持一致的应用程序级别的行为。请注意：BPEL4WS 流程的部署的描述超出了本规范的范围。

4. 定义业务流程

描述业务流程的方式有两种。可执行业务流程模拟业务交互中的参与者的实际行为。在可执行流程中，并不把业务流程分成从外部可看见的（或者说“公共”）部分和内部部分。相对而言，业务协议使用的流程描述指定了涉及协议的每一方的相互可以看见的消息交换行为并隐藏它们的内部行为。涉及业务协议的流程被称为抽象流程。一般来说，抽象流程是不可执行的。它们应被用来耦合 Web 服务接口定义与行为规范，这些行为规范既被用于约束业务角色的实现，也被用于以准确的词汇来定义业务协议中的每一方可以期望的对方行为。BPEL4WS 应被用来定义这两种流程。*两者之间的差异仅限于这两种流程中用于数据处理的不同功能集。*

BPEL 流程指定参与的 Web 服务的确切调用顺序 - 顺序地或并行地。使用 BPEL，您可以表述条件行为。例如，某个 Web 服务的调用可以取决于上次调用的值。还可以构造循环、声明变量、复制和赋予值、定义故障处理程序等。通过组合所有这些构造，您可以以算法的形式定义复杂业务流程。

通常情况下，BPEL 业务流程接收请求。为了满足请求，该流程调用相关的 Web 服务，然后响应原始调用方。由于 BPEL 流程与其它 Web 服务通信，因此它在很大程度上依赖于复合型 Web 服务调用的 Web 服务的 WSDL 描述。

开发一个 BPEL 实例一般所要的步骤：

1. 熟悉相关的 web 服务
2. 为此 BPEL 流程定义 WSDL
3. 定义合作伙伴链接类型状态
4. 进行开发 BPEL 流程开发
   A. 定义合作伙伴链接
   B. 声明变量
   C. 编写逻辑流程定义

在定义业务流程这个小结中，为了使您更清楚地了解和掌握定义的过程，我们会用一个例子加以说明。(暂时我们还是先用 ORACLE BPEL 实例教程的例子)，通过该流程实例，使开发人员了解一个工作流（process）的开发。

4.1 BPEL 实例教程

我们来看一个示例。一个 BPEL 流程由多个步骤组成，每个步骤称作“活动”。BPEL 支持基元活动和结构活动。基元活动表示基本构造，用于如下所示的常见任务：

- 使用 <invoke> 调用其它 Web 服务
- 使用 <receive>（接收请求）等待客户端通过发送消息调用业务流程
- 使用 <reply> 生成同步操作的响应
- 使用 <assign> 操作数据变量

- 使用〈throw〉指示故障和异常
- 使用〈wait〉等待一段时间
- 使用〈terminate〉终止整个流程。

然后，我们可以组合这些基元活动以及其它基元活动，以定义准确指定业务流程步骤的复杂算法。为组合基元活动，BPEL 支持几个结构活动。其中最重要的是：

- 顺序（〈sequence〉），它允许定义一组将按顺序调用的活动。
- 流（〈flow〉），用于定义一组将并行调用的活动
- Case-switch 构造（〈switch〉），用于实现分支
- While（〈while〉），用于定义循环
- 使用〈pick〉能够选择多个替换路径之一。

每个 BPEL 业务还将使用〈partnerLink〉定义合作伙伴链接，使用〈variable〉声明变量。

为了理解 BPEL 是如何描述业务流程的，我们将定义雇员的出差安排简化业务流程：客户端调用此业务流程，指定雇员姓名、目的地、出发日期以及返回日期。此 BPEL 业务流程首先检查雇员出差状态。我们将假设存在一个可用于进行此类检查的 Web 服务。然后，此 BPEL 流程首先检查雇员出差状态。我们假设存在一个可用于进行此类检查的 Web 服务。然后，此 BPEL 流程将检查以下两家航空公司的机票价格：美国航空公司和达美航空公司。我们将再次假设这两家航空公司均提供了可用于进行此类检查的 Web 服务。最后，此 BPEL 流程将选择较低的价格并将出差计划返回给客户端。

然后，我们将构建一个异步 BPEL 流程。我们将假设用于检查雇员出差状态的 Web 服务是同步的。由于可以立即获取此数据并将其返回给调用方，因此这是一个合理的方法。为了获取机票价格，我们使用异步调用。由于确认飞机航班时刻表可能需要稍长的时间，因此这也是一个合理的方法。为了简化示例，我们假设以上两家航空公司均提供了 Web 服务，且这两个 Web 服务完全相同（即提供相同的端口类型和操作）。

在实际情形下，您通常无法选择 Web 服务，而是必须使用您的合作伙伴提供的服务。如果您有幸能够同时设计 Web 服务和 BPEL 流程，则应考虑用哪个接口更好。通常，您将对持续时间较长的操作使用异步服务，而对在相对较短的时间内返回结果的操作使用同步服务。如果使用异步 Web 服务，则 BPEL 流程也通常是异步的。

当您用 BPEL 定义业务流程时，您实际上定义了一个由现有服务组成的新 Web 服务。该新 BPEL 复合 Web 服务的接口使用一组端口类型提供了类似任何其它 Web 服务的操作。要调用 BPEL 描述的业务流程，则必须调用生成的复合 Web 服务。图 3 是我们流程的示意图。

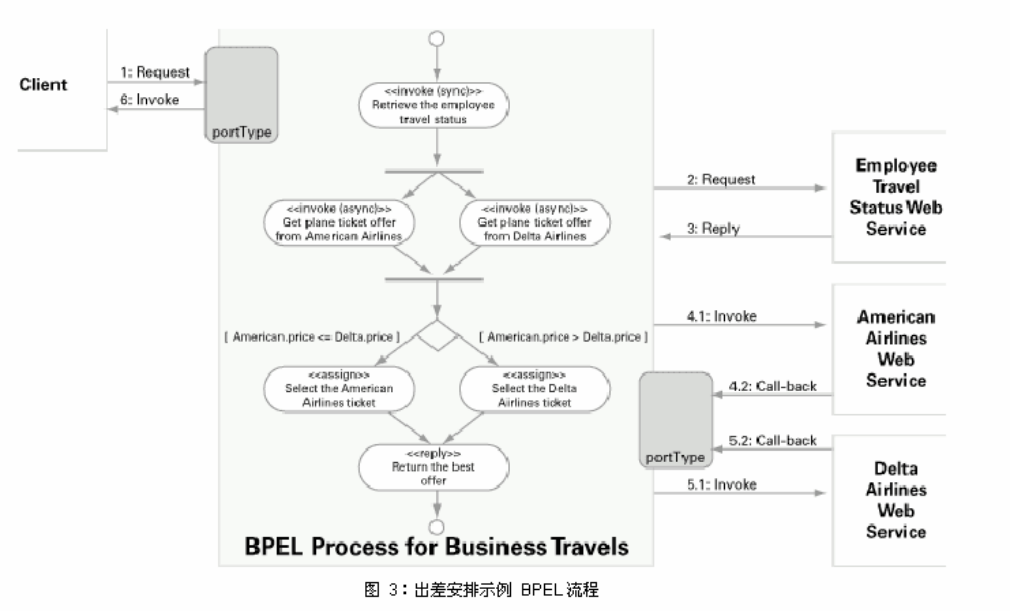

图 3：出差安排示例 BPEL 流程

在开发此示例 BPEL 流程的过程中，您将经历下列步骤：

- 熟悉相关的 Web 服务
- 为此 BPEL 流程定义 WSDL
- 定义合作伙伴链接类型
- 开发此 BPEL 流程：
  - 定义合作伙伴链接
  - 声明变量
  - 编写流程逻辑定义。

第 1 步：列出相关 Web 服务的清单 ————————定义每一个接口，即所谓的 web 服务清单的 PortType

在您开始编写 BPEL 流程定义之前，必须先熟悉从业务流程中调用 Web 服务。这些服务称作*合作伙伴 Web 服务*。

在您开始编写 BPEL 流程定义之前，必须先熟悉 coong 业务流程中调用 Web 服务。这些服务称作为*合作伙伴 Web 服*务。本示例使用雇员出差状态 Web 服务以及美国航空公司和达美航空公司 Web 服务（这两个 Web 服务具有相同的 WSDL 描述）。（同样，本示例中使用的 Web 服务是虚构的。）

雇员出差状态 Web 服务雇员出差状态 Web 服务提供 EmployeeTravelStatusPT 端口类型，通过它可以使用 EmployeeTravelStatus 操作检查雇员出差状态。此操作将返回雇员可以使用的乘机标准（可能为经济舱、商务舱或头等舱）。（见图 4。）

图 4：雇员出差状态 Web 服务

航空公司 Web 服务航空公司 Web 服务是异步的；因此它指定了两个端口类型：第一个端口类型 FlightAvailabilityPT 用于使用 FlightAvailability 操作检查航班可用性。为返回结果，该 Web 服务指定了第二个端口类型 FlightCallbackPT。此端口类型指定 FlightTicketCallback 操作。

尽管航空公司 Web 服务定义了两个端口类型，但它只实现 FlightAvailabilityPT。FlightCallbackPT 则由作为 Web 服务客户端的 BPEL 流程实现。图 5 是此 Web 服务体系结构的示意图：

图 5：航空公司 Web 服务

第 2 步：为 BPEL 流程定义 WSDL

接下来，我们必须将此业务出差 BPEL 公开为 Web 服务。因此，第二步是为它定义 WSDL。此流程将必须从它的客户端接收消息并返回结果。它必须公开 TravelApprovalPT 端口类型，后者将指定一个输入消息。它还必须声明 ClientCallbackPT 端口类型（用于使用回调将结果异步返回给客户端）。图 6 说明了此流程。

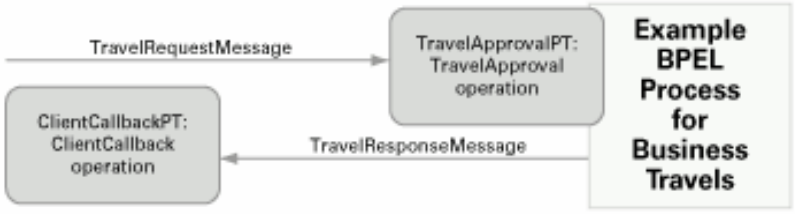

图 6：此 BPEL 流程的 WSDL

第 3 步：定义合作伙伴链接类型(partnerLinkType) ————————定义了由哪些 PortType 组成了一个交互调用过程。需要为每一个交互调用定义一个 partnerLinkType

第三步是定义合作伙伴链接类型。合作伙伴链接类型表示 BPEL 流程与相关方（包括 BPEL 流程调用的 Web 服务以及调用 BPEL 流程的客户端）之间的交互。

本示例包含三个不同的合作伙伴：客户端、雇员出差状态服务和航空公司服务。理想情况下，每个 Web 服务都应在 WSDL 包装合作伙伴链接。（实际情形可能不是这样的。）然后，我们可以用 WSDL 包装合作伙伴 Web 服务（倒入 Web 服务的 WSDL 兵定义合作伙伴链接类型）。或者，我们可以在 BPEL 流程的 WSDL 中定义所有合作伙伴链接。但由于此方法违反了封装原则，因此不建议使用。

对于本示例，我们定义了三个合作伙伴链接类型（每个类型位于 Web 服务的相应 WSDL 中）：

- travelLT：用于描述此 BPEL 流程客户端与此 BPEL 流程本身之间的交互。此交互是异步交互。此合作伙伴链接类型在此 BPEL 流程的 WSDL 中定义。
- employeeLT：用于描述此 BPEL 流程与雇员出差状态 Web 服务之间的交互。此交互是同步交互。此合作伙伴链接类型在雇员 Web 服务的 WSDL 中定义。
- flightLT：描述此 BPEL 流程与航空公司 Web 服务之间的交互。此交互是异步交互，且航空公司 Web 服务对此 BPEL 流程调用一个回调。此合作伙伴链接类型在航空公司 Web 服务的 WSDL 中定义。

每个合作伙伴链接可以拥有一个或两个角色，我们必须为每个角色指定它使用的 portType。对于同步操作，由于操作只是单向调用，因此每个合作伙伴链接类型仅有一个角色。例如，此 BPEL 流程对雇员出差状态 Web 服务调用 EmployeeTravelStatus 操作。由于它是同步操作，因此此 BPEL 流程等待完成并仅在完成操作后取得响应。

对于异步回调操作，我们必须指定两个角色。第一个角色描述客户端操作调用。第二个角色描述回调操作调用。在本示例中，BPEL 流程与航空公司 Web 服务之间存在一个异步关系。

正如我们已经指出的，我们需要三个合作伙伴链接类型：两个链接类型指定两个角色（因为它们是异步的），一个链接类型指定一个角色（因为它是同步的）。

合作伙伴链接类型在特殊命名空间http://schemas.xmlsoap.org/ws/2003/05/partner-link/ 的 WSDL 定义。首先，我们在客户端使用的 BPEL 流程 WSDL 中定义 travelLT 链接类型以调用此 BPEL 流程。所需的第一个角色是出差服务（即，我们的 BPEL 流程）的角色。客户端使用 TravelApprovalPT 端口类型与此 BPEL 服务通信。第二个角色 travelServiceCustomer 描述了此 BPEL 流程将在 ClientCallbackPT 端口类型中对其执行回调的客户端的特征：

```
<plnk:partnerLinkType name="travelLT">
    <plnk:role name="travelService">
        <plnk:portType name="tns:TravelApprovalPT" />
    </plnk:role>
    <plnk:role name="travelServiceCustomer">
        <plnk:portType name="tns:ClientCallbackPT" />
    </plnk:role>
```

```
</plnk:partnerLinkType>
```

第二个链接类型是 employeeLT。它用于描述此 BPEL 流程与雇员出差状态 Web 服务之间的通信，并在此雇员 Web 服务的 WSDL 中定义。此交互是同步交互，因此我们需要一个名为 employeeTravelStatusService 的角色。此 BPEL 流程使用雇员 Web 服务上的 EmployeeTravelStatusPT：

注意：因为此过程序同步的，所以只有一个角色，不同于其它两个通信过程。

```
<plnk:partnerLinkType name="employeeLT">
    <plnk:role name="employeeTravelStatusService">
        <plnk:portType name="tns:EmployeeTravelStatusPT" />
    </plnk:role>
</plnk:partnerLinkType>
```

最后一个合作伙伴链接类型 flightLT 用于描述此 BPEL 流程与航空公司 Web 服务之间的通信。此通信是异步通信。此 BPEL 流程对航空公司 Web 服务调用一个异步操作。此 Web 服务在完成请求后对此 BPEL 流程调用一个回调。因此，我们需要两个角色。第一个角色描述航空公司 Web 服务对于此 BPEL 流程服务的角色，即航空公司服务 (airlineService)。此 BPEL 流程使用 FlightAvailabilityPT 端口类型进行异步调用。第二个角色描述了此 BPEL 流程对于航空公司 Web 服务的角色。对于航空公司 Web 服务而言，此 BPEL 流程是一个航空公司客户，因此角色名称为 airlineCustomer。航空公司 Web 服务使用 FlightCallbackPT 端口类型进行回调。此合作伙伴链接类型在航空公司 Web 服务的 WSDL 中定义：

```
<plnk:partnerLinkType name="flightLT">
    <plnk:role name="airlineService">
        <plnk:portType name="tns:FlightAvailabilityPT" />
    </plnk:role>
    <plnk:role name="airlineCustomer">
        <plnk:portType name="tns:FlightCallbackPT" />
    </plnk:role>
</plnk:partnerLinkType>
```

了解合作伙伴链接类型对于开发 BPEL 流程规范至关重要。有时，它可以帮助生成所有交互的图表。定义合作伙伴链接类型后，我们已经完成了准备阶段，并准备开始编写业务流程定义。

第 4 步：创建业务流程 －－－－－－－－在 bpel 文件中定义业务流程

现在，您就可以开始编写 BPEL 流程了。通常，BPEL 流程等待客户端传入的消息，以启动业务流程的执行。在本示例中，客户端通过发送输入消息TravelRequest 启动此 BPEL 流程。然后，此 BPEL 流程通过发送 EmployeeTravelStatusRequest 消息调用雇员出差状态 Web 服务。由于此调用是同步调用，因此它等待 EmployeeTravelStatusResponse 消息。然后，此 BPEL 流程通过向上述两家航空公司 Web 服务发送 FlightTicketRequest 消息对它们进行并发异步调用。每个航空公司 Web 服务通过发送 TravelReponse 消息进行回调。然后，此 BPEL 流程选择较合适的航空公司并使用 TravelResponse 消息对客户端进行回调。我们首先编写一个空的 BPEL 流程提纲，它展示了每个 BPEL 流程定义文档的基本结构：

```
<process name="BusinessTravelProcess" ... >

<partnerLinks>
<!-- The declaration of partner links -->
</partnerLinks>

<variables>
<!-- The declaration of variables -->
</variables>

<sequence>
<!-- The definition of the BPEL business process main body -->
</sequence>

</process>
```

我们首先添加所需的命名空间。此处，我们必须定义目标命名空间以及用于访问雇员和航空公司 WSDL 以及此 BPEL 流程 WSDL 的命名空间。我们还必须为所有 BPEL 活动标记声明命名空间（此处采用缺省命名空间，以便不必限定每个 BPEL 标记名）。BPEL 活动命名空间必须为 http://schemas.xmlsoap.org/ws/2003/03/business-process/：

```
<process name="BusinessTravelProcess"
targetNamespace="http://packtpub.com/bpel/travel/"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:trv="http://packtpub.com/bpel/travel/"
xmlns:emp="http://packtpub.com/service/employee/"
xmlns:aln="http://packtpub.com/service/airline/" >
...
```

合作伙伴链接(partnerLink) ———在 bpel 文件中定义此 partnerLinks 目的在于导入各个 wsdl 文件中的 partnerLink，并且确定在交互过程忠各自的角色如何。

接下来，我们必须定义*合作伙伴链接*，它们定义与此 BPEL 流程交互的不同方。每个合作伙伴链接都与描述其特性的特定 partnerLinkType 相关。每个合作伙伴链接还最多指定两个属性：

- myRole：表明业务流程本身的角色。
- partnerRole：表明合作伙伴的角色。

注意： myRole 是此 bpel 流程所扮演的角色，而 partnerRole 是其合作伙伴所扮演的角色。

合作伙伴链接仅可以指定一个角色，通常同步请求/响应操作也仅能指定一个角色。对于异步操作，它指定两个角色。在本示例中，我们定义四个角色。第一个合作伙伴链接称作客户端，由 travelLT 合作伙伴链接类型描述其特性。此客户端调用该业务流程。我们需要指定 此 BPEL 流程（myRole 属性）的角色为 travelService。我们必须指定第二个角色：partnerRole。此处，该角色为 travelServiceCustomer，它

描述 BPEL 流程客户端的特性。

第二个合作伙伴链接称作 employeeTravelStatus，由 employeeLT 合作伙伴链接类型描述其特性。它是 BPEL 流程与 Web 服务之间的一个同步请求/响应关系；我们再次仅指定一个角色。此时，该角色为 partnerRole，这是因为我们描述了 Web 服务（它是此 BPEL 流程的合作伙伴）的角色：

最后两个合作伙伴链接对应于航空公司 Web 服务。由于它们使用同一类型的 Web 服务，因此我们基于一个合作伙伴链接类型 flightLT 指定两个合作伙伴链接。此处，由于我们使用异步回调通信，因此需要两个角色。此 BPEL 流程 (myRole) 对于航空公司 Web 服务的角色为 airlineCustomer，而航空公司 (partnerRole) 的角色为 airlineService：

```
<partnerLinks>
<partnerLink name="client" partnerLinkType="trv:travelLT" myRole="travelService"
partnerRole="travelServiceCustomer"/>

<partnerLink name="employeeTravelStatus" partnerLinkType="emp:employeeLT"
partnerRole="employeeTravelStatusService"/>

<partnerLink name="AmericanAirlines" partnerLinkType="aln:flightLT" myRole="airlineCustomer"
partnerRole="airlineService"/>

<partnerLink name="DeltaAirlines" partnerLinkType="aln:flightLT" myRole="airlineCustomer"
partnerRole="airlineService"/>
</partnerLinks>
```

变量

变量 BPEL 流程中的变量用于存储消息以及对这些消息进行重新格式化和转换。您通常需要为发送到合作伙伴以及从合作伙伴收到的每个消息定义一个变量。就我们的流程而言，我们需要七个变量。我们将它们命名为 TravelRequest、EmployeeTravelStatusRequest、EmployeeTravelStatusResponse、FlightDetails、FlightResponseAA、FlightResponseDA 和 TravelResponse。

我们必须为每个变量指定类型。可以使用 WSDL 消息类型、XML 模式简单类型或 XML 模式元素。在我们的示例中，我们对所有变量使用 WSDL 消息类型：

```
<variables>
<!-- input for this process -->
<variable name="TravelRequest" messageType="trv:TravelRequestMessage"/>
<!-- input for the Employee Travel Status web service -->
<variable name="EmployeeTravelStatusRequest"
messageType="emp:EmployeeTravelStatusRequestMessage"/>
<!-- output from the Employee Travel Status web service -->
<variable name="EmployeeTravelStatusResponse"
messageType="emp:EmployeeTravelStatusResponseMessage"/>
<!-- input for American and Delta web services -->
```

```
<variable name="FlightDetails" messageType="aln:FlightTicketRequestMessage"/>
<!-- output from American Airlines -->
<variable name="FlightResponseAA" messageType="aln:TravelResponseMessage"/>
<!-- output from Delta Airlines -->
<variable name="FlightResponseDA" messageType="aln:TravelResponseMessage"/>
<!-- output from BPEL process -->
<variable name="TravelResponse" messageType="aln:TravelResponseMessage"/>
</variables>
```

BPEL 流程主体

BPEL 流程主体流程主体指定调用合作伙伴 Web 服务的顺序。它通常以 <sequence>（用于定义多个将按顺序执行的操作）开始。在顺序中，我们首先指定启动业务流程的输入消息。我们使用 <receive> 构造（它等待匹配消息，在本示例中为 TravelRequest 消息）实现此目的。在 <receive> 构造中，我们不直接指定消息。而是指定合作伙伴链接、端口类型、操作名称以及可选变量（用于保存收到的消息以用于随后的操作）。

我们将消息接收与客户端合作伙伴链接在一起，并等待对端口类型 TravelApprovalPT 调用 TravelApproval 操作。我们将收到的消息存储到 TravelRequest 变量中：

```
<sequence>

<!-- Receive the initial request for business travel from client -->
<receive partnerLink="client" portType="trv:TravelApprovalPT" operation="TravelApproval"
variable="TravelRequest" createInstance="yes" />
...
```

<receive> 等待客户端调用 TravelApproval 操作，并将传入的消息以及有关业务出差的参数存储到 TravelRequest 变量中。此处，此变量名与消息名相同，但并不一定要相同。

接下来，我们需要调用雇员出差状态 Web 服务。但在调用之前，我们必须为此 Web 服务准备输入。查看雇员 Web 服务的 WSDL，可以看到我们必须发送由雇员部分组成的消息。我们可以通过复制客户端发送的消息的雇员部分来构造此消息。编写相应的赋值语句：

```
...
<!-- Prepare the input for the Employee Travel Status Web Service -->
<assign>
    <copy>
        <from variable="TravelRequest" part="employee"/>
        <to variable="EmployeeTravelStatusRequest" part="employee"/>
    </copy>
</assign>
...
```

现在，我们就可以调用雇员出差状态 Web 服务了。为了进行同步调用，我们使用 <invoke> 活动。我们使用 employeeTravelStatus 合作伙伴链接，并对 EmployeeTravelStatusPT 端口类型调用

EmployeeTravelStatus 操作。我们已经在 EmployeeTravelStatusRequest 变量中准备了输入消息。由于它是同步调用，因此该调用等待回应并将其存储在 EmployeeTravelStatusResponse 变量中：...

```
<!-- Synchronously invoke the Employee Travel Status Web Service -->
<invoke partnerLink="employeeTravelStatus" portType="emp:EmployeeTravelStatusPT"
operation="EmployeeTravelStatus"
inputVariable="EmployeeTravelStatusRequest"
outputVariable="EmployeeTravelStatusResponse" />
```

同步调用会有两个变量：inputVariable、outputVariable。

...

下一步是调用上述两个航空公司 Web 服务。同样，我们先准备所需的输入消息（这两个 Web 服务的输入消息相同）。FlightTicketRequest 消息包含两部分：

- flightData：它从客户端消息（TravelRequest）中检索而得。
- travelClass：它从 EmployeeTravelStatusResponse 变量中检索而得。

因此，我们编写一个包含两个 copy 元素的赋值：

```
...
<!-- Prepare the input for AA and DA -->
<assign>
    <copy>
        <from variable="TravelRequest" part="flightData"/>
        <to variable="FlightDetails" part="flightData"/>
    </copy>
    <copy>
        <from variable="EmployeeTravelStatusResponse" part="travelClass"/>
        <to variable="FlightDetails" part="travelClass"/>
    </copy>
</assign>
...
```

输入数据包含需要传递给航空公司 Web 服务的数据。由于格式相同，因此我们可以使用一个简单复制直接传递它。在实际情况下，通常需要执行转换。为此，可以使用具有 <assign> 的 XPath 表达式、使用转换服务（如 XSLT 引擎）或使用由特定 BPEL 服务器提供的转换功能。

现在，我们准备调用这两个航空公司 Web 服务。我们将进行并发的异步调用。为表述并发，BPEL 提供了 <flow> 活动。对每个 Web 服务的调用将包含两个步骤：

- 使用 <invoke> 活动进行异步调用。
- 使用 <receive> 活动等待回调。

我们使用 <sequence> 对这两个活动进行分组。这两个调用只在合作伙伴链接名称上存在差别。我们对一个调用使用 AmericanAirlines，对另一个调用使用 DeltaAirlines。两者均对 FlightAvailabilityPT 端

口类型调用 FlightAvailability 操作，发送 FlightDetails 变量中的消息。

使用 <receive> 活动接收回调。我们再次使用这两个合作伙伴链接名。<receive> 等待对 FlightCallbackPT 端口类型调用 FlightTicketCallback 操作。我们将结果消息分别存储到 FlightResponseAA 和 FlightResponseDA 变量中：

```
...
<!-- Make a concurrent invocation to AA in DA -->
<flow>

<sequence>
<!-- Async invoke of the AA web service and wait for the callback-->

<invoke partnerLink="AmericanAirlines" portType="aln:FlightAvailabilityPT"
operation="FlightAvailability" inputVariable="FlightDetails" />

<receive partnerLink="AmericanAirlines" portType="aln:FlightCallbackPT"
operation="FlightTicketCallback" variable="FlightResponseAA" />

</sequence>

<sequence>
<!-- Async invoke of the DA web service and wait for the callback-->

<invoke partnerLink="DeltaAirlines"  portType="aln:FlightAvailabilityPT"
operation="FlightAvailability" inputVariable="FlightDetails" />

<receive partnerLink="DeltaAirlines"  portType="aln:FlightCallbackPT"
operation="FlightTicketCallback" variable="FlightResponseDA" />

</sequence>

</flow>
...
```

在该流程的这个阶段，我们收到两个机票报价。在下一步中，我们必须选择一个机票报价。为此，我们使用 <switch> 活动。

```
...
<!-- Select the best offer and construct the TravelResponse -->
<switch>
<case condition="bpws:getVariableData('FlightResponseAA',
'confirmationData','/confirmationData/Price')
```

```
<= bpws:getVariableData('FlightResponseDA', 'confirmationData','/confirmationData/Price')">

<!-- Select American Airlines -->
<assign>
    <copy>
        <from variable="FlightResponseAA" />
        <to variable="TravelResponse" />
    </copy>
</assign>
</case>

<otherwise>
<!-- Select Delta Airlines -->
<assign>
    <copy>
        <from variable="FlightResponseDA" />
        <to variable="TravelResponse" />
    </copy>
</assign>
</otherwise>
</switch>
...
```

在 <case> 元素中，我们检查美国航空公司的机票报价 (FlightResponseAA) 是等于还是低于达美航空公司的机票报价 (FlightResponseDA)。为此，我们使用 BPEL 函数 getVariableData 并指定变量名。价格位于 confirmationData 消息的内部，虽然它是唯一的消息部分，但我们仍必须指定它。我们还必须指定查询表达式以找到价格元素。此处，我们采用简单的 XPath 1.0 表达式。

如果美国航空公司的机票报价低于达美航空公司的机票报价，则将 FlightResponseAA 变量复制到 TravelResponse 变量（我们最终将此变量返回给客户端）。否则，我们将复制 FlightResponseDA 变量。

我们已经到达此 BPEL 业务流程的最后一步 — 使用 <invoke> 活动将回调返回给客户端。对于此回调，我们使用客户端合作伙伴链接并对 ClientCallbackPT 端口类型调用 ClientCallback 操作。保存答复消息的变量为 TravelResponse：

```
...
<!-- Make a callback to the client -->
<invoke partnerLink="client" portType="trv:ClientCallbackPT"
operation="ClientCallback" inputVariable="TravelResponse" />
</sequence>

</process>
```

到此，我们已经完成了我们的第一个 BPEL 业务流程规范。您可以看到，BPEL 并不是很复杂，并允许相对

简单和自然的业务流程规范。

通过 Oracle 提供的 bpel 实例，相信大家对如何开发 bpel 流程已有一个比较清楚的概念。在接下来的篇章中，我们打算对 bpel 的各种元素做一个详细的解释，希望结合上述的例子，使您掌握开发 bpel 流程的基础知识。

5. 业务流程的结构

这一节里，我们较详细地介绍 BPEL4WS 的基础知识，以比较详细的方式介绍各个活动（activity）。
我们首先对一些符号进行说明：

“？”：0 个或 1 个；
“*”：0 个或多个；
“+”：1 个或多个。

BPEL 语言定义流程的基本结构是：

```
<process name="ncname" targetNamespace="uri"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         enableInstanceCompensation="yes|no"?
         abstractProcess="yes|no"?
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>?
    <!-- Note: At least one role must be specified. -->
    <partnerLink name="ncname" partnerLinkType="qname"
             myRole="ncname"? partnerRole="ncname"?>+
    </partnerLink>
  </partnerLinks>

  <partners>?
     <partner name="ncname">+
      <partnerLink name="ncname"/>+
   </partner>
```

```
  </partners>

  <variables>?
    <variable name="ncname" messageType="qname"?
                type="qname"? element="qname"?/>+
  </variables>

  <correlationSets>?
    <correlationSet name="ncname" properties="qname-list"/>+
  </correlationSets>

  <faultHandlers>?
    <!-- Note: There must be at least one fault handler or default. -->
    <catch faultName="qname"? faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
  </faultHandlers>

  <compensationHandler>?
    activity
  </compensationHandler>
```

```
<eventHandlers>?
  <!-- Note: There must be at least one onMessage or onAlarm handler. -->
  <onMessage partnerLink="ncname" portType="qname"
             operation="ncname" variable="ncname"?>
    <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
    <correlations>
    activity
  </onMessage>
  <onAlarm for="duration-expr"? until="deadline-expr"?>*
    activity
  </onAlarm>
</eventHandlers>
  activity
</process>
```

最高级别的属性如下：

- queryLanguage 这个属性指定了在赋值、属性定义和其它使用中用于选择节点的 XML 查询语言。这个属性的缺省值是 XPath 1.0，其代表是 XPath 1.0 规范的 URI：http://www.w3.org/TR/1999/REC-xpath-19991116。

- bleInstanceCompensation 这个属性决定流程实例是否可被作为整体由特定于平台的方式来补偿。这个属性的缺省值是 "no"。

- abstractProcess 这个属性指定所定义的流程是抽象的（还是可执行的）。这个属性的缺省值是 "no"。

Activity 可以是 BPEL 定义里的任何一个活动，我们接下来主要是对 activity 进行讨论。
BPEL 流程主体，主要就是由这些 Activity 组织的。
每个活动都有的属性和元素我们称之为标准元属性和标准元素，有如下一些标准属性和标准元属：

A. <标准属性>是每个活动都有的，但是可选的，其标准属性如下：

1. name="ncname"?
   一个属性名称。
2. joinCondition="bool-expr"?
   一个连接条件。属性值是表达式语言中的取值为布尔值的表达式。
3. suppressJoinFailure="yes|no"?
   一个在连结故障发生时指示是否压制它的指示符。缺省值是 no。

B. <标准元属>是每个活动都有的，但是可选的，其标准元素如下：

1. <source linkName="ncname" transitionCondition="bool-expr"?/>*
   可以有零个，一个或多个；
   通过包括一个或多个 <source> 元素，一个活动可以把自己声明为一个或多个链接的*源*。
   transitionCondition 可以省略，缺省值是 true。
2. <target linkName="ncname"/>*
   可以有零个，一个或多个；
   通过包括一个或多个 <target> 元素，一个活动可以把自己声明为一个或多个链接的*目标*。

Activity 分为两个部分，一个是基元活动，另一部分是结构活动，下面分别对两者活动进行详细说明：

5.1 〈基元活动 --- Basic Activity〉

基元活动是与外界进行交互最简单的形式。它们是无序的个别步骤，与服务进行交互、操作、传输数据或者处理异常等。

BPEL4WS 的基元活动包括如下：

1. 流程用于和外界进行交互的基元活动：receive、invoke、reply；
2. 流程用于传输数据的基元活动：assign；
3. 流程中其它的基元活动：
   - 通过 throw 活动发出故障信号；
   - 通过 terminate 活动放弃所有流程实例的执行；
   - 通过 wait 活动使流程等待一段时间或到达某个截止期限后再执行；
   - 通过 empty 活动不执行任何的动作；
   - 通过 compensate 活动做一些补偿动作，通常需要和 scope 联合使用；

5.1.1 〈Activity –invoke〉

语言结构：

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
        inputVariable="ncname"? outputVariable="ncname"?
        standard-attributes>
  standard-elements
  <correlations>?
     <correlation set="ncname" initiate="yes|no"?
                  pattern="in|out|out-in"/>+
  </correlations>
  <catch faultName="qname" faultVariable="ncname"?>*
    activity
  </catch>
  <catchAll>?
```

invoke：主要用于调用其它的 Web 服务（在 wsdl 文件中描述）。

除了活动的标准属性（见 5.A）和标准元素（见 5.B）之外，invoke 活动还包括：

1. 基本属性如下：
   - A. partnerLink="ncname" 描述两个web services之间的接口关系
   - B. portType="QName" 端口类型
   - C. operation="ncname" 调用操作
   - D. inputvariable="ncname" 输入变量
   - E. outputvariable="ncname" 输出变量

2. 相关集

```
<correlations>?
     correlation set="ncname" initiation="yes|no"?
                                  pattern="in|out|out-in"/>+
</correlations>
```

相关集属性组：correlation：单个相关集；set=“ncname”：相关集集合名称；initiation：是否实例化该相关集，值为 yes 时，相关集被实例化，为 no 时，不实例

化。缺省值为 no。
当调用的操作是同步的请求 / 响应时，一个 pattern 属性被用来指出相关性是应用到出站的（请求）消息(pattern=“out”)还是应用到入站的（响应）消息(pattern=“in”)还是应用到两者(pattern=“out-in”)。

3. catch 动作

```
<catch faultName="QName" faultVarible="ncname"? >*
     Activity
</catch>
```

被定义的每个 catch 动作都能拦截某种故障（由全局唯一的故障 QName 和有与该故障关联的数据的变量来定义）。如果没有故障名，那么 catch 将拦截全部有适合类型的故障数据变量的故障。故障变量是可选的，这是因为故障可能没有与之关联的额外数据。
如果只有故障QName，那么 catch动作将拦截全部有适合类型的故障QName的故障。
Activity，catch子句下的活动，这个是唯一的，但是activity里面是可以嵌套的。

4. catchall 动作

```
<catchAll>?
    activity
</catchAll>
```

catchAll 子句可被用来捕获所有未被特定的 catch 处理程序捕获的故障，可有可无。
Activity，catchAll子句下的活动，这个是唯一的，但是activity里面是可以嵌套的。

5. compensationHandler动作

```
<compensationHandler>?
    activity
</compensationHandler>
```

补偿处理程序，在目前的 BPEL4WSv1.1 版本中，补偿处理程序仅仅是补偿活动的包装，即补偿处理程序里的activity就是补偿活动，补偿处理程序（一旦被安装后）可被看作完全独立的操作，业务流程实例的全局状态既不影响它也不受它的影响。它只影响外部的实体。
Activity，compensationHandler子句下的活动，这个是唯一的，但是activity里面是可以嵌套的。

5.1.2 〈Activity –receive〉

语言结构：

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
         variable="ncname"? createInstance="yes|no"?
         standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</receive>
```

receive：主要用于（接收请求）等待客户端通过发送消息调用业务流程。
除了活动的标准属性（见 A）和标准元素（见 B）之外，receive 活动还包含：

1. 基本属性如下：
   A. createInstance 流程实例创建与否，值为yes时，创建流程实例，值为no时不创建，缺省值为no；
   B. partnerLink="ncname" 描述两个web services之间的接口关系；
   C. portType="QName" 端口类型；
   D. operation="ncname" 调用操作；
   E. variable="ncname" 变量；

2. 相关集
   <correlations>?
   <correlation set="ncname" initiation="yes|no"?>+
   </correlations>
   相关集属性组：correlation：单个相关集；set=“ncname”：相关集集合名称；initiation：是否实例化该相关集，值为 yes 时，相关集被实例化，为 no 时，不实例化。缺省值为 no。

业务流程通过 receive 活动和相应的 reply 活动把服务提供给它的伙伴。receive 活动指定了它期望从哪个伙伴那里接收，还指定了它期望伙伴调用的端口类型和操作。receive活动可以用来实例化一个流程，即把 createInstance 属性设置为 "yes"，该属性的缺省值是no。一个业务流程实例绝不可以为相同的伙伴、portType 和操作同时启用两个 receive 活动，如果执行该操作则抛出标准故障 bpws:conflictingReceive。receive 是阻塞操作，其含义是它的执行在流程实例接收到匹配的消息后才完成。

5.1.3 〈Activity –reply〉

语言结构：

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
       variable="ncname"? faultName="qname"?
       standard-attributes>
  standard-elements
  <correlations>?
     <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</reply>
```

reply：主要用于生成同步操作的响应。
除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，reply 活动还包含：

1. 基本属性如下：
   A. partnerLink="ncname" 描述两个web services之间的接口关系；
   B. portType="QName" 端口类型；
   C. operation="ncname" 调用操作；
   D. variable="ncname" 变量；
   E. faultName="QName" QName，暗含的一个相应错误名称；

2. 相关集.
   <correlations>?
   <correlation set="ncname" initiation="yes|no"?>+
   </correlations>

相关集属性组：correlation：单个相关集；set=“ncname”：相关集集合名称；initiation：是否实例化该相关集，值为 yes 时，相关集被实例化，为 no 时，不实例化。缺省值为 no。

reply 活动被用来发送对先前通过 receive 活动被接受的请求的响应。这种响应一般只对同步交互有意义。异步响应的发送方式基本是调用伙伴的服务上的相应的单向操作。在流程中可以定义多个 reply 活动来回答该伙伴的调用；但是，一次只可能有一个匹配的 〈reply〉 成为激活的。适当的 reply 活动的匹配是在运行时进行的，此时流程寻找这样一个活动 — 准备运行并拥有和 〈receive〉 相同的 portType、操作和伙伴。如果出现该规范错误，则抛出标准故障 bpws:conflictingRequest。如果响应正常，则 faultName 属性不会被使用，变量 variable 响应一个正常的消息类型。反之，如果隐含一个错误的响应，则 faultName 属性有用，变量 variable 暗含一个错误的消息类型。

5.1.4 〈Activity –assign〉

语言结构：

```
<assign standard-attributes>
  standard-elements
  <copy>+
    from-spec
    to-spec
  </copy>
</assign>
```

assign：主要用于操作数据变量。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，assign 活动还包含：

1. copy +把数据从一个变量复制到另外一个变量
   - A. from-spec 赋值数据的源，是以下形式中的一种：
     - I. 〈from variable="ncname" part="ncname"? query="queryString"? /〉
     - II. 〈from partnerLink="ncname" endpointReference="*myRole*|*partnerRole*"/〉
     - III. 〈from variable="ncname" property="QName"/〉
     - IV. 〈from expression="general-expr"/〉
     - V. 〈from〉... literal value ...〈/from〉
   - B. to-spec 赋值数据的目的地,是以下形式中的一种：
     - I. 〈to variable="ncname" part="ncname"? query="queryString"?/〉
     - II. 〈to partnerLink="ncname"/〉
     - III. 〈to variable="ncname" property="QName"/〉

在各种形式中，variable属性提供变量名；part属性提供该变量中的一部分的名称；query 的属性值是用于识别源变量部分或目标变量部分中的单个值的查询字符串（在抽象流程中禁止使用 query），BPEL4WS 为用于这些查询的语言提供了可扩展的机制。该语言由 〈process〉 元素的“queryLanguage”属性来指定。partnerLink属性合作伙伴链接名称。endpointReference属性是对角色的引用。expr允许流程对属性和变量进行简单的计算。允许把给出的文字值作为源值赋给目的地。

from 和 to 的匹配，应该注意：

1. from 的关联类型和 to 的关联类型要相同；
2. from 和 to 都是变量时，两个变量必须属于相同的消息类型；
3. from 和 to 中，不允许一个是变量，另外一个不是变量；

如果上述几种方式不匹配，抛出标准故障 bpws:mismatchedAssignmentFailure。

Assign 活动可以把数据从一个变量复制到另一个变量，也可使用表达式来构造和插入新数据。使用表达式的主要动机是为了进行简单的计算（例如递增序列号）以用于描述业务协议行为。表达式对消息选择、属性和文字常数进行运算以产生变量属性或选择新的值。最后，这个活动还可把服务引用复制到伙伴链接，或把伙伴链接复制到服务引用。

throw：主要用于抛出相关的故障和异常。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，throw 活动还包含：

1. faultName ="QName" 每个故障的一个全局唯一的 QName，故障名称；
2. faultVariable ="ncname"? 可选的数据变量，包含有关故障的更多信息；

当业务流程需要显式地发出内部故障信号时可以使用 throw 活动，每个故障需要有一个全局唯一的QName，throw活动必须提供故障这样的名称（faultName），还可以提供可选的数据变量（faultVariable），数据变量包含有关故障的更多信息。故障处理程序可以使用这种数据，以分析和处理该故障，并植入需被发送到其它服务的所有故障消息。

5.1.5 〈Activity –throw〉

语言结构：

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
  standard-elements
</throw>
```


throw：主要用于抛出相关的故障和异常。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，throw 活动还包含：

1. faultName ="QName" 每个故障的一个全局唯一的 QName，故障名称；
2. faultVariable ="ncname"? 可选的数据变量，包含有关故障的更多信息；

当业务流程需要显式地发出内部故障信号时可以使用 throw 活动，每个故障需要有一个全局唯一的QName，throw活动必须提供故障这样的名称（faultName），还可以提供可选的数据变量（faultVariable），数据变量包含有关故障的更多信息。故障处理程序可以使用这种数据，以分析和处理该故障，并植入需被发送到其它服务的所有故障消息。

5.1.6 〈Activity –wait〉

语言结构：

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

wait：主要用于等待一段时间。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，wait 活动还包含：

1. for="duration-expr" ： 等待的时间长度，是一个长度表达式；
2. until="deadline-expr"： 等待时间截止期限；

wait 活动使流程能够等待一段特定的时间间隔；或者一直等到某个截止期限为止。

5.1.7 〈Activity –empty〉

语言结构：

```
<empty standard-attributes>
  standard-elements
</empty>
```

empty：不做任何事情。
empty 活动只有活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）。
empty 活动不会执行任何动作，是个空活动，目的是为了结构的完整性。如果为了捕获一个异常并抑制它，就可以使用这个活动。

5.1.8 〈Activity –terminate〉

语言结构：

```
<terminate standard-attributes>
  standard-elements
</terminate>
```

terminate：主要用于终止整个流程。
terminate 活动只有活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）。
〈terminate〉活动可以用于立即放弃执行着该终止活动的业务流程实例中的所有执行。即所有的当前运行的活动必须尽可能快地被终止而不出现任何故障处理和补偿行为。

5.1.9 〈Activity –compensate〉

语言结构：

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

compensate：主要用于构造被用来在已正常完成执行的内层作用域上调用补偿。
除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，compensate活动还包括：
1. scope="ncname"？：用于compensate活动的scope名称。

〈compensate〉活动被用来在已正常完成执行的内层作用域上调用补偿。只能从故障处理程序或另一个补偿处理程序中调用这个活动。仅当作用域正常完成执行后该作用域的补偿处理程序才可被调用。调用未被安装的补偿处理程序等同于 empty 活动（它是 no-op）—— 这就确保了故障处理程序不必依赖于状态来决定哪个嵌套的作用域已成功地完成。补偿处理程序只能被调用一次，如果一个已被安装的补偿处理程序被调用的次数超过一次，那么就会抛出标准故障 bpws:repeatedCompensation。

5.2 〈结构活动 --- Structure Activity〉

结构化的活动规定了一组活动发生的顺序。他们描述了业务流程是怎样通过把它执行的基本活动组成结构而被创建的，这些结构表达了涉及业务协议的流程实例间的控制形式、数据流程、故障和外部事件的处理以及消息交换的协调。

BPEL4WS 的结构化的活动包括如下：

1. 活动间一般的顺序控制由 sequence、switch 和 while 来提供；
2. 活动间的并发和同步由 flow 来提供；
3. 基于外部事件的不确定的选择由 pick 来提供；

我们可以用一般的方法来递归地使用结构化的活动。需要理解的要点是结构化的活动可以被任意的嵌套和组合。

5.2.1 〈Activity –sequence〉

语言结构：

```
<sequence standard-attributes>
  standard-elements
    activity+
</sequence>
```

sequence：主要用于定义一组将按顺序调用的活动。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，sequence 活动还包括：“ activity + ”。

“Activity+”是嵌套在 sequence 中的一个或一组按先后顺序被执行的活动，当 sequence 中的最后一个活动完成后，该 sequence 活动也就完成了。

5.2.2 〈Activity –switch〉

语言结构：

```
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>
```

switch：主要用于实现分支。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，switch 活动还包括：

1. 〈case condition="bool-expr">+ 一个或一组case组成的分支；
   A. Condition：分支条件，一般用布尔表达式描述(bool-expr)；

B. Activity： 分支下的具体活动；

2. otherwise：Case活动之外的另一个分支，可写可不写，缺省默认存在；
   A. Activity： otherwise分支下的活动；

Switch 活动是以各种不同形式的条件行为组成的结构化活动，即 switch 活动里可以根据不同的行为分成几个活动分支。

Switch活动由 case 元素定义的一个或多个条件分支的有序列表组成（每个case里面有一个布尔表达式bool-expr的条件），最后可跟，也可以不跟一个 otherwise 分支。以 switch 活动中的 case 分支的出现顺序来考虑它们。即条件是 true 的第一个分支被选择，并被作为该 switch 的被执行的活动。如果有条件的分支都未被选择，那么 otherwise 分支将被选择。如果 otherwise 分支未被显式地指定，那么有 empty 活动的 otherwise 分支将被认为存在。当被选中的分支里的活动完成后，switch 活动也就完成了。

5.2.3 〈Activity –while〉

语言结构：

```
<while condition="bool-expr" standard-attributes>
   standard-elements
   activity
</while>
```

while：主要用于定义循环。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，while 活动还包括：

1. Condition：while执行条件，布尔表达式描述(bool-expr)；
2. Activity： 嵌套在 while 里的活动；

While活动支持指定在while里的活动的反复执行，活动重复至给出的while条件的布尔表达式（condition="bool-expr"）值为false。即只要给出的while条件的布尔表达式的值为true，就一直执行嵌套在while里的活动，反之，跳出while活动，while结束。

5.2.4 〈Activity –flow〉

语言结构：

```
<flow standard-attributes>
  standard-elements
  <links>?
    <link name="ncname">+
  </links>

  activity+
</flow>
```

flow：主要用于定义一组将并行调用的活动。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，flow 活动还包括：

1. Links 链接

Name：每个 link 的名称，对应于活动基本属性中 source 和 target 元素中的 linkName；

2. Activity+ 一组并发的活动
   activity+ 是一组嵌套在 flow 中的并行的活动。同时为了定义任意的控制结构，可以在并行的活动中使用链接（link），每个 link 都有一个名称（linkName），flow 活动的所有链接必须在 flow 活动中被分开定义。

Flow活动的标准元素中的source 和 target被用来链接两个活动，即每个链接（有相应的linkName区别不同的链接）是从源（source）活动开始到目标（target）活动结束，其中链接的源必须指定链接名的 source 元素，链接的目标必须指定链接名的 target 元素。source具有transitionCondition属性，该属性是判断链接状态（linkStatus）是true还是false，链接状态是true时，可以立即执行相应的target活动，反之不执行。如果 transitionCondition 属性被省略，那么默认该属性存在且默认其值为true。

5.2.5 〈Activity –pick〉

语言结构

```
<pick createInstance="yes|no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="ncname" portType="qname"
             operation="ncname" variable="ncname"?>+
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
    activity
  </onMessage>
  <onAlarm (for="duration-expr" | until="deadline-expr")>*
    activity
  </onAlarm>
</pick>
```

pick：主要用处是它能够选择多个替换路径之一。
除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，pick 活动还包括：

1. createInstance：流程实例创建与否，是个布尔值，为yes时，创建流程实例；反之不创建，缺省值为no；
2. onMessage+ ：一个或多个 onMessage 分支组成的事件信息。各个 onMessage 是互相排斥的，只有一个分支能够被选择，包含的属性有：
   A. partnerLink="ncname"：描述两个web services之间的接口关系；
   B. portType="QName"： 端口类型；
   C. operation="ncname"： 调用操作；
   D. variable="ncname"： 变量；
   E. <correlations>?
      <correlation set="ncname" initiation="yes|no"?>+
      </correlations> 相关集属性组，correlation：单个相关集，set="ncname"：相

关集集合名称；initiation：是否实例话该相关集，值是布尔值，为 yes 时，相关集被实例化，反之，不实例化，缺省值为 no。

F. Activity：嵌套在 onMessage 里的活动；

3. onAlarm* ：0 个或多个onAlarm组成的警报事件组，当onMessage触发时，onAlarm不执行，当createInstance 值为yes时，onAlarm不能存在。
主要包含的属性如下：
A. for="duration-expr"： 警报的时间长度，是一个长度表达式；
B. until="deadline-expr"：警报时间点；
C. activity：嵌套在onAlarm里的活动；
其中，时间长度和时间点两个只有一个存在，用时间长度表达则不存在时间点表达，反之亦然。

Pick 活动等待一组事件（在onMessage中定义的）中的一个事件的发生，然后执行与发生的事件关联的活动。事件的发生总是相互排斥的（某流程接收到接受消息，或者接收到拒绝消息，但不可能同时接收到这两个消息），即各个onMessage之间是互相排斥的，只有一个分支能够被选择，当 pick 活动接受处理一个事件后，pick 将不再接受其它事件。

当业务流程的实例的创建是由于接收到一组可能的消息中的一个消息而发生时，可以使用 pick 的特殊形式。在这种情况下，pick 本身的 createInstance 属性的值是 yes（这个属性的缺省值是 no）。在这种情况下，pick 中的事件必须都是进站的消息，每一个等同于属性为 "createInstance=yes" 的 receive。对于这种特殊情况，不允许出现警报。pick活动是有警报事件的(onAlarm定义的)，它定义了一组经过多长的时间段或者在某个时刻，onMessage事件不发生时则执行onAlarm警报事件。

5.3 〈特殊活动 --- Scope〉

语言结构

```
<scope variableAccessSerializable="yes|no" standard-attributes>
    standard-elements
    <variables>?
      ... see above under <process> for syntax ...
    </variables>
    <correlationSets>?
      ... see above under <process> for syntax ...
    </correlationSets>
    <faultHandlers>?
      ... see above under <process> for syntax ...
    </faultHandlers>
    <compensationHandler>?
      ... see above under <process> for syntax ...
    </compensationHandler>
    <eventHandlers>?
      ...
    </eventHandlers>
    activity
  </scope>
```

Scope活动为嵌套在其中的活动提供故障处理功能和补偿处理功能。scope 可以提供故障处理程序、补偿处理程序、数据变量和相关集。

除了活动的标准属性（见 3.1.5）和标准元素（见 3.1.6）之外，scope 活动还包含：

1. variableAccessSerializable： 序列化作用域，值为yes或no；
2. variables? ： 作用域内的变量，只在该作用域内有效；
3. correlationSets? ： 相关集集合；
4. faultHandlers? ： 故障处理功能；
   A. catch 活动
      <catch faultName=“QName” faultVariable=“ncname”?>*
          Activity
      </catch>
      catch活动，被定义的每个 catch 活动能拦截某种故障（由全局唯一的故障 QName 和有与该故障关联的数据的变量来定义）。如果没有故障名，那么 catch 将拦截全部有适合类型的故障数据变量的故障。故障变量是可选的，这是因为故障可能没有与之关联的额外数据。如果只有故障QName，那么 catch 将拦截全部有适合类型的故障QName的故障。
      Activity，catch子句下的活动，这个是唯一的，但是activity里面是可以嵌套的。
   B. catchAll 活动
      <catchAll>
          Activity
      </catchAll>
      catchAll 子句可被用来捕获所有未被特定的 catch 处理程序捕获的故障，可有可无。
      Activity，catchAll子句下的活动，这个是唯一的，但是activity里面是可以嵌套的。
5. compensationHandlers? ：补偿处理功能；
   A. activity：执行补偿处理的活动；
6. eventHandlers? ：事件处理功能；

在 eventHandlers 中，onMessage 和 onAlarm 两者之和至少要存在一个。

A. onMessage* ：0 个或多个 onMessage 分支组成的事件信息，各个 onMessage 不互相排斥，包含的属性如下：

一、 partnerLink="ncname"： 描述两个web services之间接口关系；

二、 portType="QName" ： 端口类型；

三、 operation="ncname" ： 调用操作；

四、 variable="ncname" ： 变量；

五、 相关集

```
<correlations>?
        <correlation set="ncname" initiation="yes|no"?>+
</correlations>
```

相关集属性组：correlation：单个相关集；set=“ncname”：相关集集合名称；initiation：是否实例化该相关集，值为 yes 时，相关集被实例化，为 no 时，不实例化。缺省值为 no。

六、 activity 嵌套在 onMessage 里的活动；

B. onAlarm* ：0 个或多个 onAlarm 组成的警报事件组，当 onMessage 触发时，onAlarm 就不再被执行，包含的属性如下：

一、 for="duration-expr" ： 警报的时间长度，是一个长度表达式；

二、 until="deadline-expr"： 警报的时间点；

三、 activity ： 嵌套在onAlarm里的活动；

其中，时间长度和时间点两个只有一个存在，用时间长度表达则不存在时间点表达，反之亦然。

7. Activity： 嵌套在scope内的活动，这个是唯一的，但是activity里面是可以嵌套的。

作用域scope 可以提供故障处理程序、补偿处理程序、事件处理程序和数据变量。整个流程process也可以定义为一个scope。scope里的元素是可选的。每个scope 有一个定义它的正常行为的主要活动。该主要活动可以是一个基元活动，也可以是一个复杂的结构化的活动，其中有任意深度的嵌套的活动。所有的嵌套的活动都共享该 scope。scope活动执行的过程中，如果有异常，则在异常处理程序里进行处理；如果异常处理程序中又抛出异常，则把异常向外层作用域抛出。

通过以上活动(Activity)的学习，对 BPEL 的基础知识应该有了比较详细的了解。BPEL 主要就是由这些 Activity 组成的。

6. 合作伙伴链接类型、合作伙伴、服务引用

BPEL4WS 的很重要（可能也是最重要）的用例是描述跨企业业务交互，在这种交互中，每个企业的业务流程通过 Web 服务接口与其它企业的流程交互。为了在这种环境中真实地模拟业务流程的进行，一个重要的要求是模拟伙伴流程的能力。WSDL 已能在抽象级别和具体级别上描述由伙伴提供的服务的功能。业务流程与伙伴的关系通常是对等的，这需要在服务级别上有双向的相关性。换句话说，伙伴表示由业务流程提供的服务的消费者，同时，对于业务流程来说，伙伴又表示服务的提供者。这种情况的典型例子是基于异步消息传递（而不是基于远程过程调用）的交互。服务链接这个概念被用来直接模拟对等伙伴关系。为了定义与伙伴的关系的形式，服务链接定义了在交互的两个方向上用到的消息和端口类型。然而，实际的伙伴服务可以在流程中被动态的确定。为了表示描述伙伴服务所需的动态数据，BPEL4WS 定义了服务引用这个概念。

在这里有必要强调所用的服务链接概念和服务引用概念是初步的。目前，这些与 Web 服务相关的概念还没有广泛接受的规范，我们期待着在将来出现这些概念的标准定义。为了符合预期的未来标准，BPEL4WS 规范将被相应地更新。

6.1 合作伙伴链接

为了描述两个服务间的关系，服务链接类型定义了关系中每个服务所扮演的“角色”并指定每个角色所提供的 portType。下面举例说明服务链接类型声明的基本语法：

```
<partnerLinkType name="BuyerSellerLink"
        xmlns="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <role name="Buyer">
    <portType name="buy:BuyerPortType"/>
  </role>
  <role name="Seller">
    <portType name="sell:SellerPortType"/>
  </role>
</partnerLinkType>
```

每个角色可以包括任意个 WSDL portType。

在常见的情况中，每个角色的 portType 产生于不同的名称空间。然而在有些情况下可以用来自相同名称空间的 portType 来定义服务链接类型的两个角色。在服务间定义“回调”关系的服务链接类型就会出现后一种情况。

服务链接类型定义可以是独立于任一个服务的 WSDL 文档的单独的构件。服务链接类型定义也可以被放在定义 portType 的 WSDL 文档中，这些 portType 也被用来定义不同的角色。

WSDL 1.1 的扩展机制被用来把 serviceLinkType 定义为新的定义类型并且在所有的情况下都是作为<wsdl:definitions> 元素的直接子元素被放置。这是为了支持 WSDL 目标名称空间规范的再利用，更为重要的是，也是为了支持导入 portType 的导入机制。如果服务链接类型声明链接了两个不同服务的 portType，那么服务链接类型声明可以被放在单独的（有自己的 targetNamespace 的）WSDL 文档中。定义服务链接类型的语法如下：

```
<definitions name="ncname" targetNamespace="uri"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  ...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
    <plnk:role name="ncname">?
      <plnk:portType name="qname"/>
    </plnk:role>
  </plnk:partnerLinkType>
  ...
</definitions>
```

以上定义了服务链接类型，其名称空间是 WSDL 文档元素的“targetNamespace”属性值。正如引用所有顶层 WSDL 定义那样，通过使用 QNames 来引用标识在 <plnk:role> 中的 portType。
请注意，在有些情况下定义包含一个角色（而不是两个角色）的服务链接类型是有意义的。在这种服务链接情形中，一个服务愿意链接任何其它服务而不对其它服务提出任何要求。
您可以在本规范中的各种业务流程示例中找到 partnerLinkType 声明的示例。

6.2 伙伴链接

在 BPEL4WS 中，与业务流程交互的服务被模拟成伙伴。每个伙伴由 partnerLinkType 来描述。同一个 partnerLinkType 可以描述多个伙伴。例如，某个采购流程可以在它的事务中使用多个供应商，但是对于所有的供应商都使用相同的 partnerLinkType。

```
  <partnerLinks>
    <partnerLink name="ncname" partnerLinkType="qname"
             myRole="ncname"? partnerRole="ncname"?>+
    </partnerLink>
  </partnerLinks>
```

每个伙伴被命名，这个名称被用于与这个伙伴的所有服务交互。这一点很重要，例如，为了多个同时产生的同一种请求而使响应与不同的伙伴相关。属性 myRole 指出了业务流程的角色而属性，partnerRole 指出了伙伴的角色。

6.3 服务引用

WSDL 严格地区分 portType 和端口。PortType 使用抽象消息来定义抽象功能。端口提供实际访问信息，包括通信端点和（通过使用扩展元素）其它有关部署的信息（例如用于加密的公钥）。绑定使两者连结在一起。虽然服务的用户必须静态地依赖于由 portType 定义的抽象接口，但是在通常情况下可以动态地发现和使用包括在端口定义中的信息。

服务引用的基本用途是作为一种机制，用于服务的特定于端口的数据的动态通信。服务引用使您能在

BPEL4WS 中动态地为某个服务类型选择提供者和调用它们的操作。BPEL4WS 提供使消息与服务的有状态*实例*相关的一般机制，所以携带中立于实例间的端口信息的服务引用在多数情况下是足够的。然而，一般来说，有必要在服务引用本身中携带额外的实例标识标记。

服务引用的语法结构是：

```
<partner name="SellerShipper"
        xmlns="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <partnerLink name="Seller"/>
  <partnerLink name="Shipper"/>
</partner>
```

服务引用是〈wsdl:service〉元素的限定名，其中的元素要么被直接插入在服务引用中，要么可以假定服务引用的接收方已经知晓该元素。以下是服务引用的最简单的示例：

```
<partners>
  <partner name="ncname">+
    <partnerLink name="ncname"/>+
  </partner>
</partners>
```

如果不能假设服务可通过引用已被知晓，那么它的定义可被直接插入在服务引用中，以动态地表示 WSDL 文档的服务定义部分

本手册只是一个简明的参考手册，如果要研究 BPEL 或相关方面的内容，建议去读 BPEL 规范手册。

7. 参考手册

1. W3C 推荐“XML 规范”
2. W3C 纪要“Simple Object Access Protocol (SOAP) 1.1”
3. W3C 纪要“Web Services Description Language (WSDL) 1.1”
4. 业界倡议“Universal Description, Discovery and Integration
5. XLANG：Web Services for Business Process Design
6. WSFL：Web Services Flow Language 1.0
7. W3C 提议的推荐“XML Schema Part 1: Structures”
8. W3C 提议的推荐“XML Schema Part 2: Datatypes”
9. W3C 推荐“XML Path Language (XPath) Version 1.0”