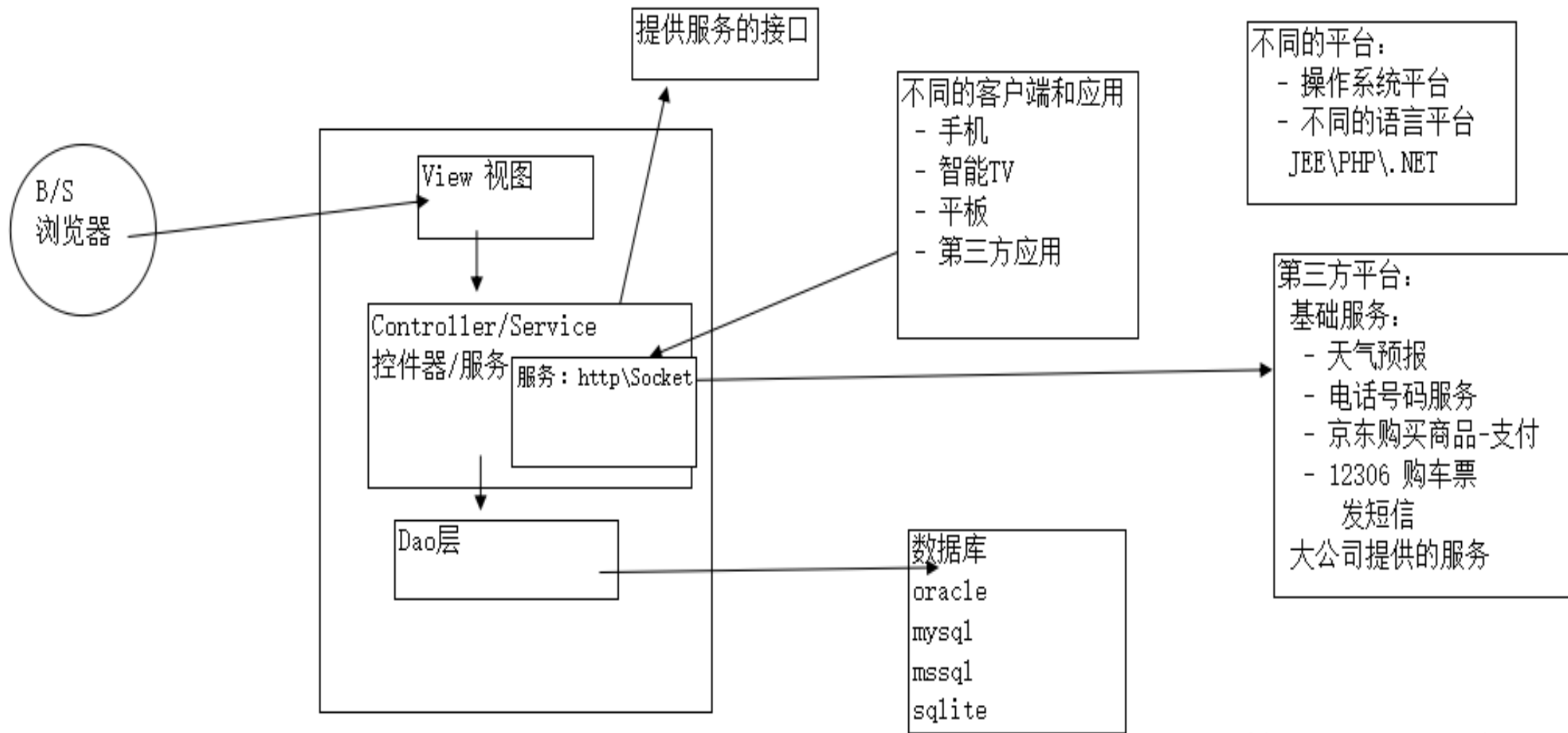


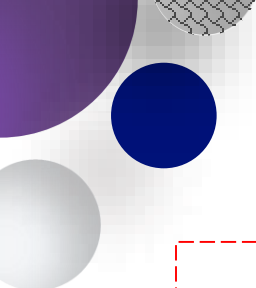


06

Web Service

6.1 Web Service (Why?)





6.1 Web Service

- Web服务(Web Service)提供了一个在不同的应用和平台之间的交互操作标准。
- 这个交互操作通过一系列基于XML的开放标准实现, 包括WSDL、SOAP和UDDI等。这些标准提供了一系列通用方法来定义、发布和使用Web Service。

6.1 Web Service

Web Service的基本层次结构

服务发布协议:	UDDI
服务描述协议:	WSDL
服务操作协议:	SOAP
统一数据格式:	XML
基础连接:	Internet

UDDI : **U**niversal **D**escription **D**iscovery and **I**ntegration

WSDL: **W**eb **S**ervice **D**escription **L**anguage

SOAP : **S**imple **O**bject **A**ccess **P**rotocol

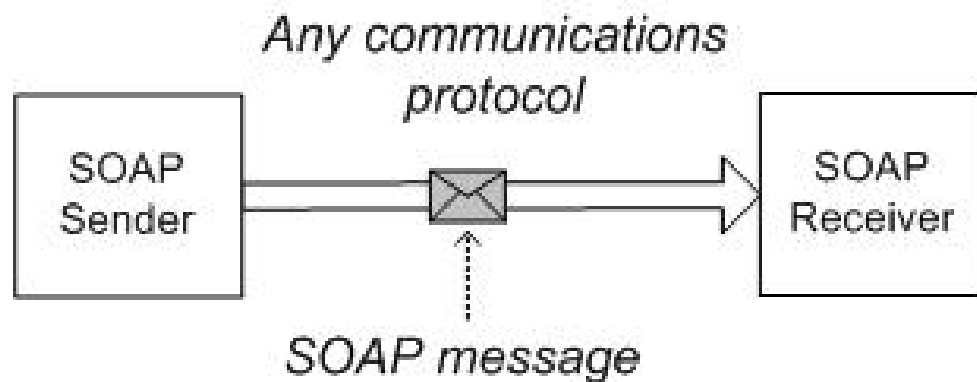
Simple, Open, Broad Industry Support

简单、开放、工业界广泛支持

6.1 Web Service

SOAP是什么？

- SOAP 是一种轻量级协议，用于在分散型、分布式环境中交换结构化信息。SOAP 利用 XML 技术定义一种可扩展的消息处理框架，它提供了一种可通过多种底层协议进行交换的消息结构。这种框架的设计思想是要独立于任何一种特定的编程模型和其他特定实现的语义。
- SOAP的概念最初来自于 Microsoft and Userland software，它已经演化了好几代；当前最新的规范是SOAP 2.0。由W3C组织制定。





6.1 Web Service

SOAP

- SOAP被广泛地认为是新一代跨平台和跨语言的分布式计算机应用的基础框架。
- SOAP 1.1只支持HTTP POST方式向终端提交请求。
- SOAP 1.2支持HTTP POST和GET两种方式。

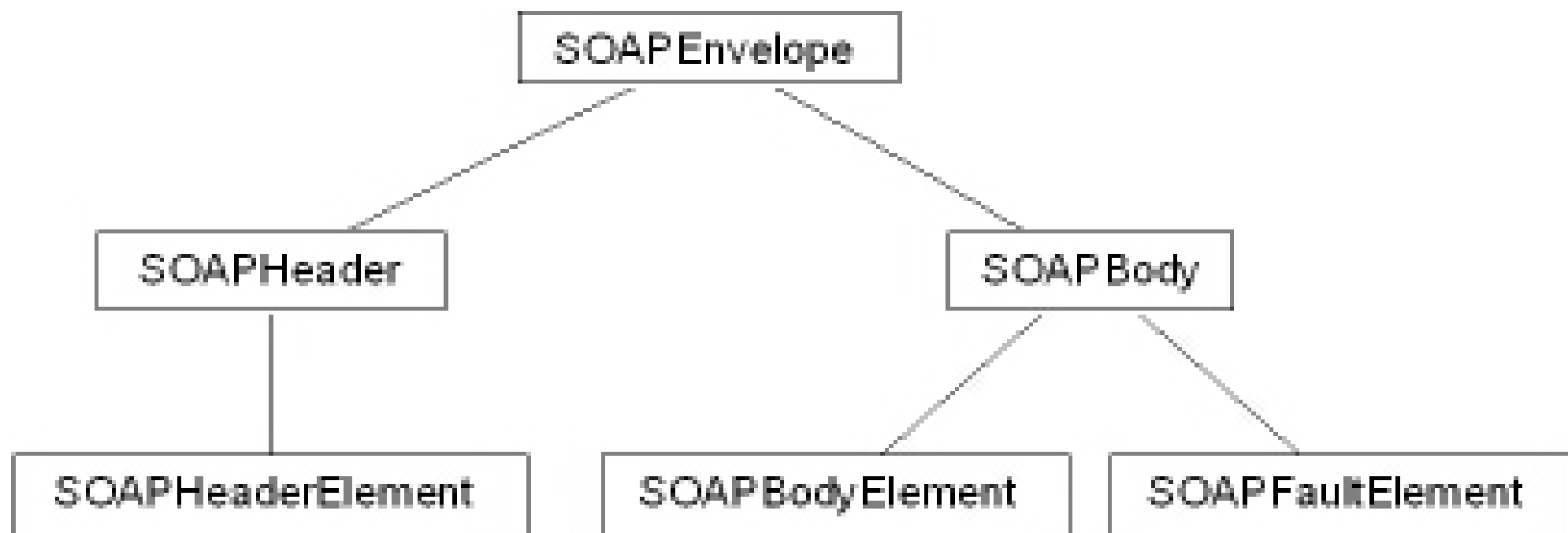
6.1 Web Service

四个主要组成部分

- SOAP是一个基于XML的轻量级规范，其主要使用在分布式系统中，由下面几个部分组成：
 - SOAP封装结构定义了一个整体框架用来表示消息中包含什么内容，谁来处理这些内容以及这些内容是可选的或是必需的。
 - SOAP编码规则定义了用以交换应用程序定义的数据类型的实例的一系列机制。
 - SOAP RPC表示定义了一个用来表示远程过程调用和应答的协定。
 - 虽然这三个部分都作为SOAP的一部分一起描述，但它们在功能上是相交的。特别的，封装和编码规则是在不同的名域中定义的。规范定义了SOAP封装、SOAP编码规则和SOAP-RPC协定之外，这个规范还定义了SOAP和其他协议的绑定，描述了在有或没有HTTP扩展框架的情况下，SOAP消息如何包含在消息中被传送。

6.1 Web Service

SOAP消息结构



6.1 Web Service

SOAP消息处理框架

- SOAP 规范的核心部分就是消息处理框架。SOAP 消息处理框架定义了一整套 XML 元素，用以“封装”任意 XML 消息以便在系统之间传输。
- 该框架包括以下核心 XML 元素：Envelope、Header、Body 和 Fault，所有这些都来自 SOAP 1.1 中的 <http://schemas.xmlsoap.org/soap/envelope/> 命名空间。以下代码中提供了 SOAP 1.1 的完整 XML 架构定义，以供在阅读下文时参考。

SOAP 1.1 XML 架构定义: [SOAP.xml](#)

6.1 Web Service

SOAP Envelope 的结构

```
<soap:Envelope
```

```
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <soap:Header> <!-- optional -->
```

```
    <!-- header blocks go here... -->
```

```
  </soap:Header>
```

```
  <soap:Body>
```

```
    <!-- payload or Fault element goes here... -->
```

```
  </soap:Body>
```

```
</soap:Envelope>
```

6.1 Web Service

SOAP Envelope 的结构

所有的SOAP消息都使用XML形式编码,一个SOAP应用程序产生的消息中,所有由SOAP定义的元素和属性中必须包括正确的域名。SOAP应用程序必须能够处理它接收到的消息中的SOAP域名,并且它可以处理没有SOAP域名的SOAP消息,就象它们有正确的域名一样。SOAP定义了两个域名:

SOAP封装的域名标志符是"http://schemas.xmlsoap.org/soap/envelope/"

SOAP的编码规则的域名标志符是
"http://schemas.xmlsoap.org/soap/encoding/"

6.1 Web Service

封装版本模型

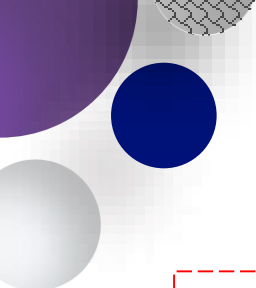
- SOAP没有定义常规的基于主版本号和辅版本号的版本形式。
- SOAP消息必须有一个封装元素与域名 "http://schemas.xmlsoap.org/soap/envelope/" 关联。
- 如果SOAP应用程序接收到的SOAP消息中的SOAP封装元素与其他的域名关联，则视为版本错误，应用程序必须丢弃这个消息。
- 如果消息是通过HTTP之类的请求/应答协议收到的，应用程序必须回答一个SOAP VersionMismatch 错误信息。

6.1 Web Service

Envelope元素

Envelope元素始终是 SOAP 消息的根元素。这就便于应用程序识别“SOAP 消息”——只要检查一下根元素的名称即可。通过检查 Envelope 元素的命名空间，应用程序也可确定所使用的 SOAP 版本。

Envelope元素包含一个可选的 **Header** 元素，后跟一个必要的 **Body** 元素。**Body** 元素代表了该消息的有效内容。它是一种通用容器，因为它可包含来自任何命名空间的任意数量的元素。这就是试图发送数据的最终目的地。



6.1 Web Service

讨论:

在银行帐户之间转帐的请求信息: request.xml

相应的响应信息: response.xml

6.1 Web Service

Fault元素

该消息处理框架还定义了一个名为**Fault** 的元素，用于在发生错误时在 Body 元素中表示错误。这是不可缺少的，因为如果没有一种标准的错误表示方法，每个应用程序将不得不自己创建，从而使通用基础结构不可能区分成功和失败。以下示例 SOAP 消息中包含了一个 Fault 元素，指明在处理该请求时发生了 “Insufficient Funds（资金不足）” 错误：fault.xml

6.1 Web Service

Fault元素

- Fault 元素必须包含一个 **faultcode**，后跟一个 **faultstring** 元素。**faultcode** 元素使用一种符合命名空间的名称对错误进行分类，而 **faultstring** 元素提供一种对错误可读的解释（类似于 HTTP 的工作方式）。表 2 简要地说明了 SOAP 1.1 所定义的各种错误码（所有这些代码都包含在 **<http://schemas.xmlsoap.org/soap/envelope/>** 命名空间中）。
- Fault 元素也可能包含一个 **detail** 元素，以便提供该错误的细节，这样可以帮助客户端诊断问题，特别是在 Client 和 Server 错误码的情况下。

6.1 Web Service

SOAP 1.1 错误码

- VersionMismatch: 处理方发现 SOAP **Envelope** 元素的命名空间是无效的
- MustUnderstand: 处理方没有理解或服从 SOAP Header 元素的某个直接子元素, 而该子元素包含一个值为 “1” 的 SOAP **mustUnderstand** 属性。
- Client: 表明消息的格式错误或者不包含适当的信息, 因而不能成功。这通常表明, 如果不对该消息做出更改, 就不应该重发该消息。
- Server: 表明该消息未能得到处理的原因与消息的内容并没有直接关系, 而是跟该消息的处理有关。例如, 处理过程可能包括与某个上游处理器的通信, 但该处理器没有响应。如果在稍后重发, 该消息可能会成功。

6.1 Web Service

Soap Header

- 大多数现有的协议都区分控制信息（例如，标头）和消息有效负载。在这方面，SOAP 也不例外。SOAP Header 和 Body 元素在易于处理的 XML 世界中也进行同样的区分。除了易用性之外，可扩展 Envelope 的关键优势在于它可用于任何通讯协议。
- 与 Body 元素类似，Header 元素是控制信息的通用容器。其中可包含来自任何命名空间（除 SOAP 命名空间之外）的任意数量的元素，以帮助控制对操作的访问：header.xml
- 可以利用一个名为 **mustUnderstand** 的全局 SOAP 属性对标头块进行标注，以指明接收方在处理该消息之前是否需要理解标头：mustunderstand.xml.

6.1 Web Service

Soap Body

- SOAP体元素 (Body) 提供了一个简单的机制, 使消息的最终接收者能交换必要的信息。使用体元素的典型情况包括配置RPC请求和错误报告。体元素编码为SOAP封装元素的直接子元素。
- 如果已经有一个头元素, 那么体元素必须紧跟在头元素之后, 否则它必须是SOAP封装元素的第一个直接子元素。体元素的所有直接子元素称作体条目, 每个体条目在SOAP体元素中编码为一个独立的元素。条目的编码规则如下:
- 一个条目由它的元素全名 (包括域名URI和局部名) 确定。SOAP体元素的直接子元素可能是域名限制的。

6.1 Web Service

协议绑定

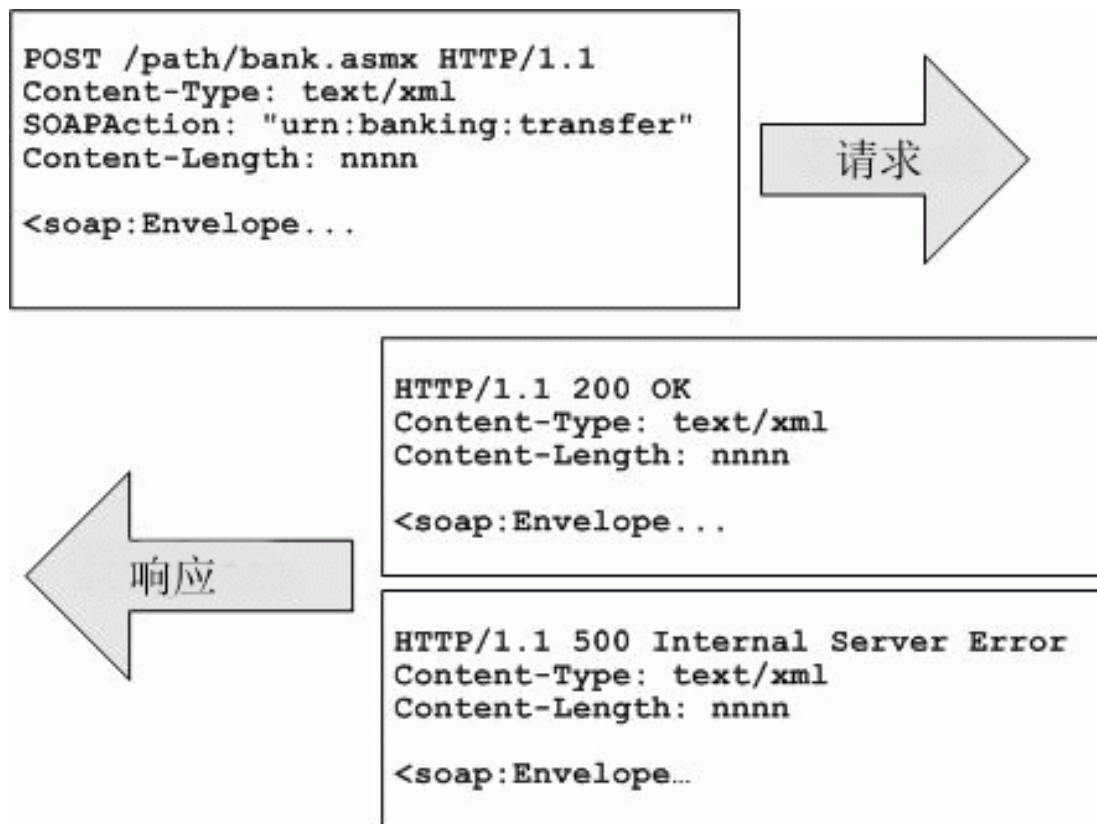
- SOAP可以和很多传输协议进行绑定:
 - SOAP over HTTP/HTTPS GET/POST
 - SOAP over JMS
 - SOAP over SMTP
 - SOAP over RPC

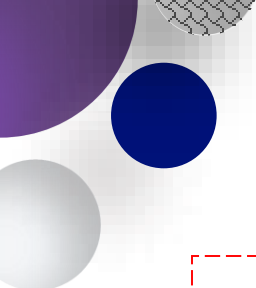
一种具体的协议绑定准确地定义了应该如何利用给定的协议来传输SOAP消息。 换言之，它详细定义了 SOAP 如何适用于另一协议的范围，该协议很可能具有自己的消息处理框架以及多种标头。协议绑定实际所定义的内容很大程度上取决于该协议的功能和选项。例如，针对 HTTP 的协议绑定应很大程度不同于针对 JMS 或针对 SMTP 的协议绑定。

6.1 Web Service

HTTP 绑定

HTTP 协议绑定定义了在使用 SOAP 的规则。SOAP 请求/响应自然地映射到 HTTP 请求/协议模型。





6.1 Web Service

SOAP RPC绑定

要利用 SOAP 进行方法调用，基础结构需要以下信息：

- 终结点位置 (URI)
- 方法名称
- 参数名称/值

6.1 Web Service

SOAP RPC绑定

RPC调用: double add(ref double x, double y)

Request对象:

```
struct add {  
    double x;  
    double y;  
}
```

```
<add>  
  <x>33</x>  
  <y>44</y>  
</add>
```

Response对象:

```
struct addResponse {  
    double result;  
}
```

```
<addResponse>  
  <result>77</result>  
</addResponse>
```

6.1 Web Service

SOAP RPC

- SOAP文档内容

```
<soap:envelope>
```

```
  <soap:body>
```

```
    <myMethod>
```

```
      <x>5</x>
```

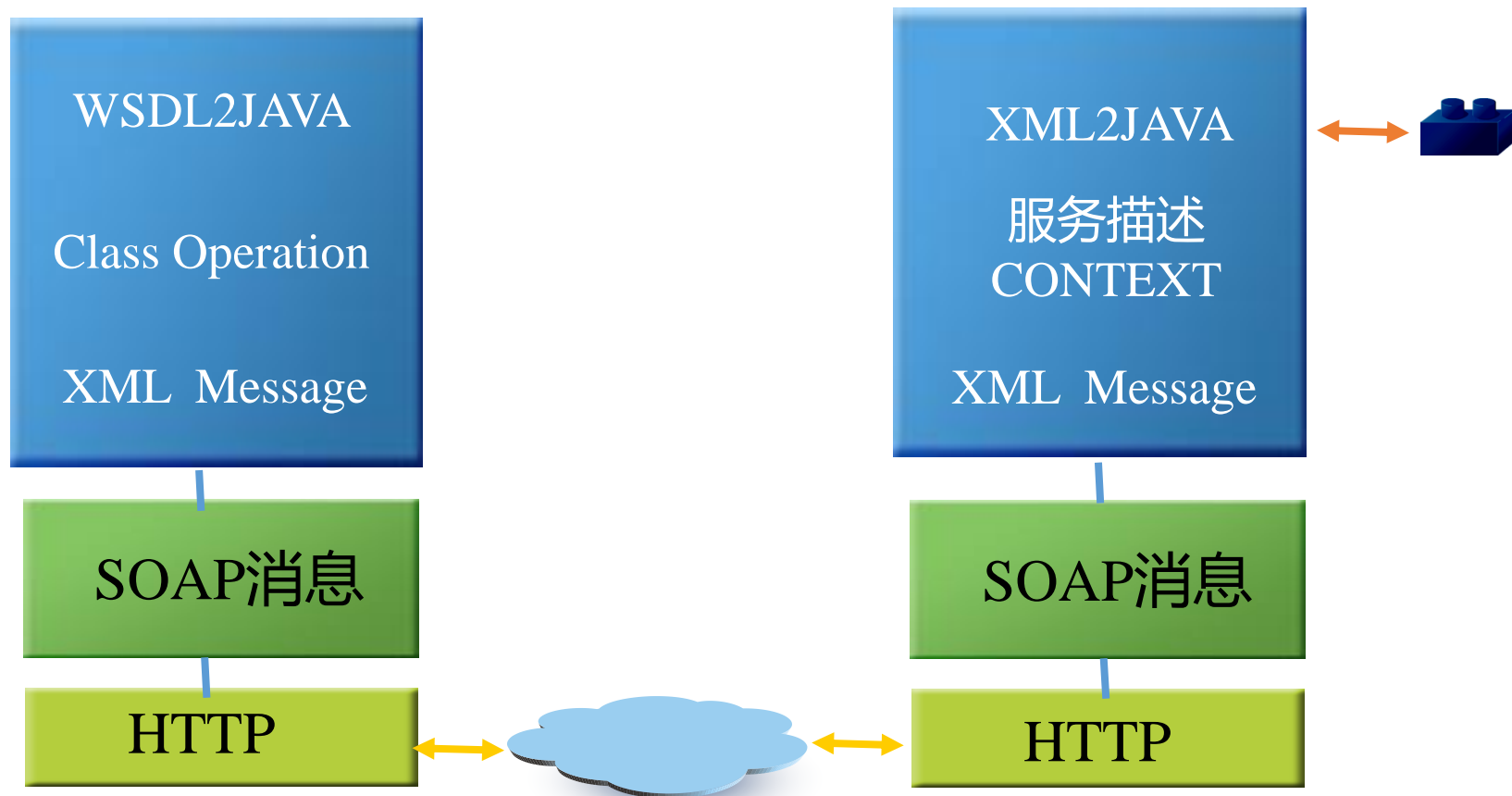
```
    </myMethod>
```

```
  </soap:body>
```

```
</soap:envelope>
```


6.1 Web Service

服务调用





6.1 Web Service

WSDL描述web服务的三个基本属性:

- 服务做些什么?
 - 服务所提供的操作(方法);
- 如何访问服务?
 - 数据格式以及访问服务操作的必要协议;
- 服务位于何处?
 - 由特定协议决定的网络地址, 如URL。

6.1 Web Service

WSDL是什么?

- 服务主要通过六个元素进行定义
 - **types**, 定义了交换信息的数据格式。
 - **message**, 传输消息的抽象定义。一个消息含有多个逻辑部分, 每一部分和一些类型相关联。
 - **portType**, 一些抽象操作的集合。每个操作关联一个输入消息和一个输出消息。
 - **binding**, 针对操作和portType中使用的消息指定实际的协议和数据格式规范。
 - **port**, 指定一个绑定的地址, 这样定义一个通信的终端。
 - **service**, 一些port构成的集合

6.1 Web Service

服务接口定义和服务实现定义

- **服务接口**组成了服务描述中的可重用部分，
 - type元素、message和portType。
 - types元素中描述消息中复杂数据类型的使用。
 - message元素指定XML 数据类型组成消息的各个部分。message元素用于定义操作的输入和输出参数。
 - portType元素中定义了Web服务的操作。操作定义了输入和输出数据流中可以出现的XML消息。

6.1 Web Service

服务接口定义和服务实现定义

- **服务实现**定义是一个描述给定服务提供者如何实现特定服务接口的WSDL文档。
 - binding和services。
 - binding 元素描述特定服务接口的协议、数据格式、安全性和其它属性。
 - service元素。服务元素包含一组port元素。端口将端点与来自服务接口定义的binding 元素关联起来。

6.1 Web Service

WSDL

- WSDL是一种XML应用,它将Web Services描述定义为一组服务访问端点, 客户端可以通过这些服务访问端点对包含面向文档信息或面向过程调用的服务进行访问。
- WSDL首先对访问的操作和访问时使用的请求 / 响应消息进行抽象描述, 然后将其绑定到具体的传输协议和消息格式上, 以最终定义具体部署的服务访问端点。
- 在具体使用中, 可以使用任意的消息格式和网络协议。
- 在WSDL规范中, 定义了如何使用SOAP消息格式、HTTP GET / POST消息格式来完成Web Services交互的规范。

6.1 Web Service

WSDL文档框架

```
<wsdl:definitions name="nmtoken" targetNamespace="uri">
    .....
    <wsdl:types> .....</wsdl:types>
    <wsdl:message name="nmtoken">*
        .....</wsdl:message>
    <wsdl:portType name="nmtoken">*
        .....</wsdl:portType>
    <wsdl:binding name="nmtoken" type="qname">*
        .....</wsdl:binding>
    <wsdl:service name="nmtoken">*.....</wsdl:service>
</wsdl:definitions>
```

6.1 Web Service

类型

- types元素包含了交换消息的数据类型定义。为了实现最大的互操作性（interoperability）和平台中立性（neutrality），WSDL选用XML Schema DataTypes，简称XSD作为标准类型系统，并将它作为固有类型系统。

```
<definitions ....>
```

```
  <types>
```

```
    <xsd:schema .... />*
```

```
  </types>
```

```
</definitions>
```


6.1 Web Service

```
<types>
  <schema...>
    <element name="PO" type="tns:POType"/>
    <complexType name="POType">
      <element name="id" type="string"/>
      <element name="name" type="string"/>
      <element name="items">
        <complexType>
          <element name="item"
            type="tns:Item" minOccurs="0"
            maxOccurs="unbounded"/>
        </complexType>
      </element>
```

```
</complexType>
  <complexType name="Item">
    <element name="quantity" type="int"/>
    <element name="product" type="string"/>
  </complexType>
  <element name="Customer"
    type="tns:CustomerType"/>
    <complexType name="CustomerType">
      <element name="name" type="string"/>
    </complexType>
  </schema>
</types>
```

6.1 Web Service

消息

- 消息由若干个逻辑部件（part）构成。每个部件使用一个消息类型属性与某个系统的类型相关联。

- 消息定义语法如下：

```
<definitions .... >
```

```
  <message name="nmtoken"> *
```

```
    <part name="nmtoken"
```

```
      element="qname"? type="qname"?/> *
```

```
  </message>
```

```
</definitions>
```

- 消息属性指定了消息的名称。如果消息具有多个逻辑单位，则需要使用多个part元素。

6.1 Web Service

消息示例

```
<message name="PO">
```

```
  <part name="po" element="tns:PO"/>
```

```
  <part name="customer" element="tns:Customer"/>
```

```
</message>
```

```
<message name="P1">
```

```
  <part name="address" type="XSD:string"/>
```

```
</message>
```

```
<message name="P2">
```

```
  <part name="composite" type="tns:Composite"/>
```

```
</message>
```

6.1 Web Service

端口类型定义

- 端口类型是一个由抽象操作和抽象消息构成的有名称的集合。

```
<wsdl:definitions .... >
```

```
  <wsdl:portType name="nmtoken"> *
```

```
    <wsdl:operation name="nmtoken">
```

```
      <wsdl:input name="nmtoken"? message="qname"/>
```

```
      <wsdl:output name="nmtoken"? message="qname"/>
```

```
      <wsdl:fault name="nmtoken" message="qname"/>*
```

```
    </wsdl:operation>
```

```
  </wsdl:portType >
```

```
</wsdl:definitions>
```

- 端口类型定义的name属性表示端口类型名称，操作定义的name属性表示操作名称。

6.1 Web Service

操作

- WSDL支持4种消息交换方式，来访问服务端点。
 - 单向（One-way）：服务访问端点接收消息；
 - 请求响应（Request-response）：服务访问端点接收请求消息，然后发送响应消息；
 - 要求应答（Solicit-response）：服务访问端点发送要求消息，然后接收应答消息；
 - 通知（Notification）：服务访问端点发送通知消息。
- 操作中引用到的消息通过message属性指定。

6.1 Web Service

单向操作

- 单向操作语法:

```
<wsdl:definitions .... >
```

```
  <wsdl:portType .... > *
```

```
    <wsdl:operation name="nmtoken">
```

```
      <wsdl:input name="nmtoken"? message="qname"/>
```

```
    </wsdl:operation>
```

```
  </wsdl:portType >
```

```
</wsdl:definitions>
```

- input元素指定用于单向操作的抽象消息格式。

6.1 Web Service

请求响应操作

- 请求响应操作语法

```
<wsdl:definitions .... >
```

```
  <wsdl:portType .... > *
```

```
    <wsdl:operation name="nmtoken"
```

```
      parameterOrder="nmtokens">
```

```
        <wsdl:input name="nmtoken"? message="qname"/>
```

```
        <wsdl:output name="nmtoken"? message="qname"/>
```

```
        <wsdl:fault name="nmtoken" message="qname"/>*
```

```
      </wsdl:operation>
```

```
    </wsdl:portType >
```

```
</wsdl:definitions>
```

6.1 Web Service

要求应答操作

- 要求应答操作语法

```
<wsdl:definitions .... >
```

```
  <wsdl:portType .... > *
```

```
    <wsdl:operation name="nmtoken"
```

```
      parameterOrder="nmtokens">
```

```
        <wsdl:output name="nmtoken"? message="qname"/>
```

```
        <wsdl:input name="nmtoken"? message="qname"/>
```

```
        <wsdl:fault name="nmtoken" message="qname"/>*
```

```
      </wsdl:operation>
```

```
    </wsdl:portType >
```

```
</wsdl:definitions>
```


6.1 Web Service

通知操作

- 通知操作语法

```
<wsdl:definitions .... >
```

```
  <wsdl:portType .... > *
```

```
    <wsdl:operation name="nmtoken">
```

```
      <wsdl:output name="nmtoken"?

```

```
        message="qname"/>

```

```
    </wsdl:operation>

```

```
  </wsdl:portType >

```

```
</wsdl:definitions>
```