第2讲 软件服务开发技术之 WebService

郭 勇

哈尔滨工业大学 计算机科学与技术学院

主要内容

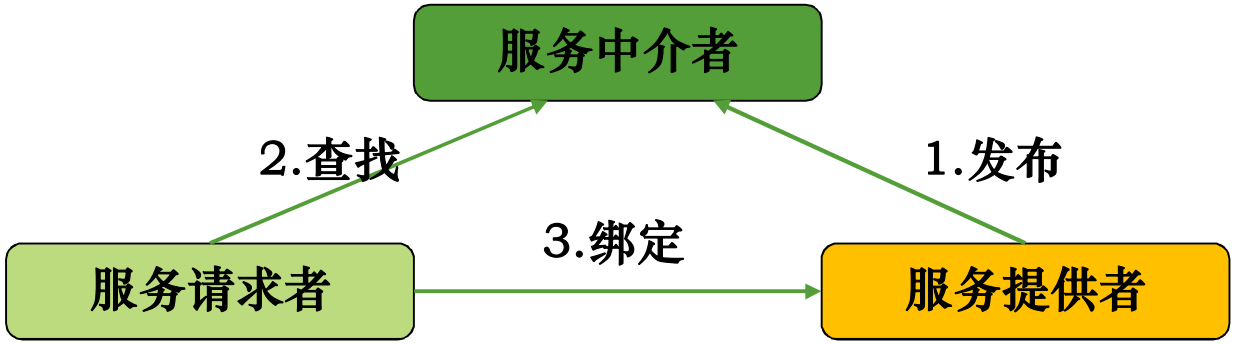
1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

1.1什么是WebService

- 什么是Web Service
  - 是一种跨编程语言和跨操作系统的远程调用技术
  - 是一个平台独立的，低耦合的，基于可编程的Web的应用程序
  - 可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的互操作的应用程序
  - 相关概念
    - stub：为屏蔽客户调用远程主机上的对象，必须提供某种方式来模拟本地对象，这种本地对象称为存根(stub)，存根负责接收本地方法调用，并将它们委派给各自的具体实现对象

1.2 WebServices体系结构

- Web 服务体系结构中
  - 涉及到三个角色:
    - Web 服务提供者、Web 服务中介者、Web 服务请求者
  - 同时还涉及到三类动作：即发布、查找、绑定
  - Web Service 结构

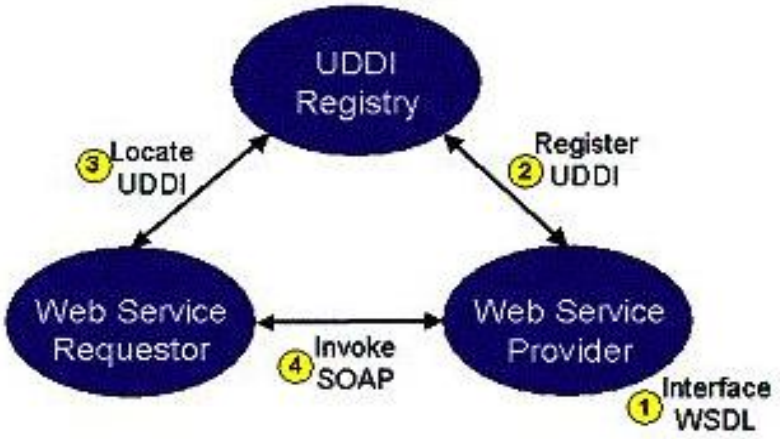


1.2 WebServices体系结构

- 在Web 服务的体系结构中
  - Web 服务提供者
    - 可以发布 Web 服务，并且对使用自身服务的请求进行响应
    - Web 服务的拥有者，它会等待其他服务或者是应用程序访问
  - Web 服务请求者
    - 即 Web 服务功能的使用者
    - 它通过服务注册中心也就是 Web 服务中介者查找到所需要的服务，再利用 SOAP 消息向 Web 服务提供者发送请求以获得服务
  - Web 服务中介者
    - 也称为服务代理，用来注册已经发布的 Web服务提供者，并对其进行分类，同时提供搜索服务
    - 充当一个管理者的角色，一般是通过 UDDI来实现

1.2 WebServices体系结构

- **发布**
  - 通过发布操作，可以使 Web服务提供者向 Web 服务中介者注册自己的功能以及访问的接口
- **发现（查找）**
  - 使得 Web 服务请求者可以通过 Web 服务中介者来查找到特定的种类的 Web 服务
- **绑定**
  - 让服务请求者能够使用服务提供者提供的服务



1.3WebService的三种基本元素

- WebService的三种基本元素
  - SOAP 即 Simple Object Access Protocol 也就是简单对象访问协议，是一种用于访问 Web 服务的协议。
  - WSDL 即Web Services Description Language也就是 Web 服务描述语言。WSDL 描述了 Web服务的三个基本属性:
    - 服务所提供的操作
    - 如何访问服务
    - 服务位于何处(通过 URL 来确定)
  - UDDI 即 Universal Description，Discovery and Integration，也就是通用的描述，发现以及整合。

1.4 WebService栈

- 发现服务层：用来帮助客户端解析远程服务的位置
- 描述服务层：为客户端程序提供与远程服务交互的描述信息
- 消息格式层：保证客户端应用程序和服务器在格式设置上保持一致
- 编码格式层：为客户端和服务器之间提供一个标准的、独立于平台的数据交换编码格式
- 传输协议层：为客户端和服务器之间提供交互的网络通信协议

发现服务	UDDI、DISCO
描述服务	WSDL、XML、Schema
消息格式层	SOAP
编码格式层	XML
传输协议层	HTTP、TCP/IP、SMTP等

1.5WebService开发

WebService开发可以分为服务器端开发和客户端开发两个方面

- 服务端开发
  - 把系统业务功能开发成WebService服务，供远程调用
  - 通常借助一些WebService框架来实现。Java方面的典型WebService框架包括：axis、xfire、cxf 等。Java EE服务器通常也支持发布WebService服务，例如JBoss。
- 客户端开发
  - 调用别人发布的WebService服务
  - 可使用服务提供商的WSDL2Java之类的工具生成静态调用的代理类代码；服务提供商的客户端编程API类；使用SUN公司早期标准的jax-rpc开发包；使用 SUN公司最新标准的jax-ws开发包等。

1.5WebService开发

- WebService 的工作调用原理
  - 客户端
    - 给WebService客户端API传递WSDL文件的URL地址，这些API就会创建出底层的代理类
    - 我们调用这些代理，就可以访问到Webservice服务
    - 代理类把客户端的方法调用变成SOAP格式的请求数据再通过HTTP协议发出去，并把接收到的SOAP 数据变成返回值返回
  - 服务端
    - 各类WebService框架的本质就是一个大大的Servlet，当远程调用客户端给它通过HTTP协议发送过来 SOAP格式的请求数据时，它分析这个数据，就知道要调用哪个Java类的哪个方法，于是去查找或创建这个对象，并调用其方法
    - 再把方法返回的结果包装成SOAP格式的数据，通过HTTP响应消息回给客户端

1.5WebService开发

- 几种流行Webservice框架
  - 开发webservice应用程序中离不开框架的支持
  - 性能是Webservice的关键要素，不同的框架性能上存在较大差异
  - 下面是比较流行的几个框架：Apache Axis1、Apache Axis2、Codehaus XFire、Apache CXF、Apache Wink、Jboss RESTEasy、sun JAX-WS(最简单、方便)、阿里巴巴Dubbo

1.5 WebService开发

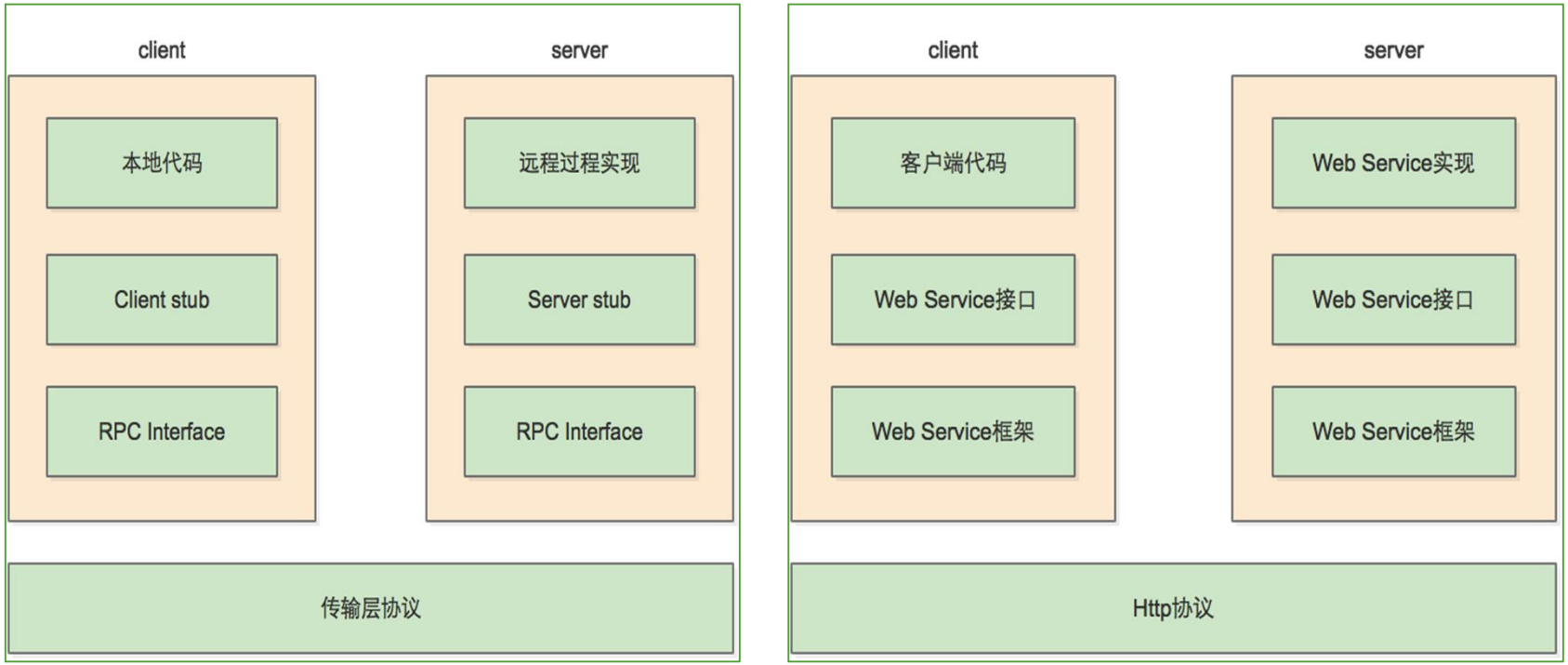
- 框架比较
  - Apache CXF是CodehausXFire的第二代产品。
  - 相比其他框架，CXF具有几个突出的特性：支持JAX-WS、Spring集成、支持RESTful services、支持Apache协议、代码实现简洁。
  - Apache Axis2是Apache Axis1的第二代产品，架构上也非常不错，关键特性：支持多语言（C/C++）、支持各种规范、可插拔模块化设计、支持热部署等。与CXF相比性能也非常优异。

主要内容

1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

2.WebService与RPC

- 两者类似，在组件层次和交互时序上没有差别，只是方框内的字不一样，但是实际上承担的职责却是完全对应的。
- Web Service接口就是RPC中的stub组件，规定了Server能够提供的服务（Web Service），这在Server和Client上是一致的。
- Web Service能更好的跨语言跨平台。由于Web Service规范中的WSDL文件的存在，各平台的Web Service框架，都可以基于WSDL文件，自动生成web service接口。



2.WebService与RPC

- 常见的RPC框架
  - Hessian、Thrift、Hetty、阿里的Dubbo等
- JAVA中共有三种WebService规范
  - JAXM(Java API for XML Messaging)&SAAJ(SOAP with Attachments API for JAVA)
  - JAX-WS（JAX-RPC）
  - JAX-RS
- 支持WebService的框架
  - 支持JAX-WS服务规范的框架有：CXF、Axis、Xfire、Axis2。结合java语言均可可实现JAX-WS
  - 支持JAX-RS服务规范的框架有:
    - CXF——XFire和Celtix的合并
    - Jersey——Sun公司的JAX-RS参考实现
    - RESTEasy——JBoss的JAX-RS项目
    - Restlet——也许是最早的REST框架了，在JAX-RS之前就有了

主要内容

1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

3.WSDL

- 什么是WSDL
  - 网络服务描述语言，Web Services Description Language
  - 基于 XML 的语言，用于描述 Web Services 以及如何对它们进行访问
  - 它包含一系列描述某个 web service 的定义
- WSDL 文档结构

元素	定义
<portType>	web service 执行的操作
<message>	web service 使用的消息
<types>	web service 使用的数据类型
<binding>	web service 使用的通信协议

3.WSDL

- XML
  - XML 指可扩展标记语言（EXtensible Markup Language）
  - XML 是一种标记语言，很类似 HTML
  - XML 的设计宗旨是传输数据，而非显示数据
  - XML 标签没有被预定义。您需要自行定义标签。
  - XML 被设计为具有自我描述性。
  - XML 是 W3C 的推荐标准

```
<note>
   <to>George</to>
   <from>John</from>
   <heading>Reminder</heading>
   <body>Don't forget the meeting!</body>
</note>
```

3.WSDL

❖ 一个 WSDL 文档的主要结构是类似这样的

```
<definitions>
    <types>
      definition of types……..
    </types>
    <message>
      definition of a message….
    </message>
    <portType>
      definition of a port…….
    </portType>
    <binding>
      definition of a binding….
    </binding>
</definitions>
```

■ WSDL 文档可包含其它的元素,比如 extension 元素,以及一个 service 元素,此元素可把若干个 web services 的定义组合在一个单一的 WSDL 文档中。

3.WSDL

- WSDL 端口
  - <portType> 元素是最重要的 WSDL 元素。
  - 它可描述一个 web service可被执行的操作以及相关的消息。
  - 可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）
- WSDL 消息
  - <message> 元素定义一个操作的数据元素。
  - 每个消息均由一个或多个部件组成。可以把这些部件比作传统编程语言中一个函数调用的参数。
- WSDL types
  - <types> 元素定义 web service 使用的数据类型。
  - 为了最大程度的平台中立性,WSDL 使用 XML Schema 语法来定义数据类型。
  - ＜types＞元素包含了Types栏。如果没有需要声明的数据类型，这栏可以缺省。在WSDL范例中，没有应用程序特定的types声明，但我仍然使用了Types栏，只是为了声明Schema namespaces。

3.WSDL

❖ WSDL 实例

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>
<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
```

- 例子中<portType> 元素把 “glossaryTerms” 定义为某个端口的名称，把 “ getTerm ” 定义为某个操作的名称。
- 操作 “getTerm” 拥有一个名为 “getTermRequest” 的输入消息，以及一个名为 “getTermResponse” 的输出消息。
- <message> 元素可定义每个消息的部件，以及相关联的数据类型。
- 对比传统的编程glossaryTerms 是一个函数库，而 “ getTerm ” 是带有输入参数 “ getTermRequest ” 和返回参数 “ getTermResponse ” 的一个函数。

3.WSDL

- **操作(operation)类型**
  - WSDL 定义了四种类型，请求-响应是最普通的操作类型

类型	定义
One-way	服务端接收消息；
Request-response	服务端点接收请求消息，然后发送响应消息；
Solicit-response	服务访问端发送要求消息，然后接收应答消息。
Notification	服务访问端发送通知消息

3.WSDL

❖ One-Way 操作例子

- 例子中端口"glossaryTerms"定义了一个名为"setTerm"的 one-way 操作。
- 这个 “setTerm” 操作可接受消息的输入，这些消息使用一条名为 “newTermValues” 的消息，此消息带有输入参数 “term” 和 “value” 。不过，没有为这个操作定义任何输出。

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

3.WSDL

❖ Request-Response 操作

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

- 例子中端口"glossaryTerms"定义了一个名为"getTerm"的 request-response 操作。
- “getTerm” 操作会请求一个名为 “getTermRequest” 的输入消息，有一个名为“term” 的参数；返回一个名为“getTermResponse” 的输出消息，此消息带有一个名为 "value" 的参数。

3.WSDL

- WSDL 绑定
  - 为porttype中的operation和message指定一个具体的传输协议（SOAP协议）和数据格式
  - use属性描述了消息序列化的方式

```
<message name="getTermRequest">
  <part name="term" type="xs:string" />
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string" />
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest" />
    <output message="getTermResponse" />
  </operation>
</portType>
```

```
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://example.com/getTerm" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

3.WSDL

- binding 元素有两个属性 - name 属性和 type 属性
  - name 属性定义 binding 的名称
  - type 属性指向用于 binding 的端口，在这个例子中是 "glossaryTerms" 端口。
- soap:binding 元素有两个属性 - style 属性和 transport 属性
  - style 属性可取值 “rpc” 或 “document”，在这个例子中我们使用 document。
  - transport 属性定义了要使用的 SOAP 协议，在这个例子中我们使用 HTTP。
- operation 元素定义了每个端口提供的操作符
  - 对于每个操作，相应的 SOAP 行为都需要被定义。
  - 同时必须指定如何对输入和输出进行编码，在这个例子中我们使用了 "literal"。

主要内容

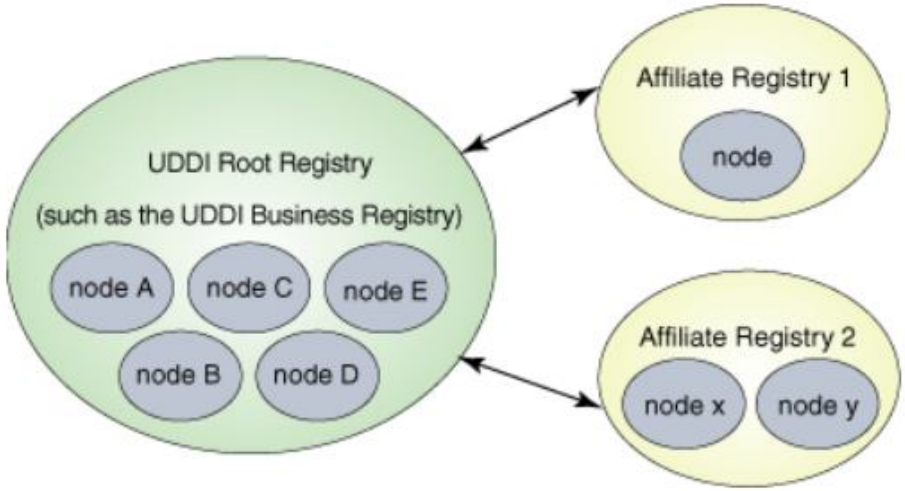
1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

4.UDDI

- 什么是UDDI
  - 英文为 “Universal Description, Discovery and Integration”，可译为“通用描述、发现与集成服务”
  - 是一种用于存储有关 web services 信息的目录
  - UDDI标准定义了企业注册服务的框架，但是它并没有指定任何注册服务实现的细节
  - 是一种由 WSDL 描述的 Web Services 界面的目录
  - 经由 SOAP 进行通信

4. UDDI

- UDDI 如何被使用
  - 行业发布一套某服务的 UDDI 标准和目录
  - 同行业的提供服务的公司就可以把它们要发布的服务注册到这个 UDDI 目录中
  - 然后服务使用者就能够搜索这个 UDDI 目录以找到相关服务。
  - 服务找到后,使用者就能够与此服务进行通信，使用相关服务。



主要内容

1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

5.SOAP

- 为什么使用 SOAP?
  - RPC:
    - 可使用多种协议(包括HTTP以及在TCP的自定义协议)和序列化方式(Json/xml/二进制)，组件之间的耦合度比较高。服务管理的机制相对较弱
    - RPC 会产生兼容性以及安全问题；防火墙和代理服务器通常会阻止此类流量。
  - SOAP
    - SOAP使不同的操作系统上的不同编程语言的应用程序可以互相进行通信
    - SOAP 很简单，基本上是一个用 XML 信封作为有效负载的 HTTP POST
    - SOAP 消息支持 Web 服务体系结构中的发布、查找和绑定操作

5.1 什么是SOAP

- 什么是SOAP
  - 简单对象访问协议 SOAP（Simple Object Access Protocol）
  - 是在松散的、分布的环境中使用XML交换结构化的和类型化的信息的一种简单协议。
  - 用于应用程序之间的通信、是一种用于发送消息的格式
  - 被设计用来通过因特网进行通信
  - 独立于平台、独立于语言、基于 XML、SOAP 很简单并可扩展、SOAP 允许您绕过防火墙、SOAP 属于W3C 标准

```
public class Person{
    String name;
    int age;
    //方法的定义...
}
```

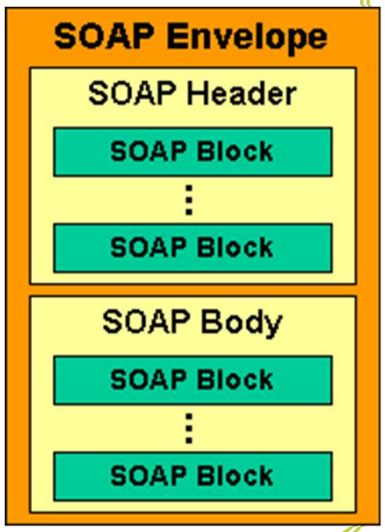
```
<Person>
    <name>zhang3</name>
    <age>20</age>
</Person>
```

5.1 什么是SOAP

- SOAP的设计目标
  - 简单性和可扩展性
  - 这意味着传统的消息系统和分布对象系统的某些性质不是SOAP规范的一部分
- SOAP没有定义任何底层的传输协议
  - 可以使用HTTP、FTP、SMTP或者JMS,甚至是自定义协议来传输SOAP报文
  - 一般使用HTTP协议
- SOAP的调用效率比较低
  - HTTP不是高效率的通信协议
  - XML需要额外的文件解析

5.2 SOAP语法

- SOAP 模块
  - 一条 SOAP 消息就是一个普通的 XML 文档,包含下列元素:
    - 必需的 Envelope 元素,用于把 XML 文档标识为一条 SOAP 消息
    - 可选的 Header 元素,包含头部信息
    - 必需的 Body 元素,包含所有的调用和响应信息
    - 可选的 Fault 元素,提供有关在处理此消息所发生错误的信息
  - 以上的元素均被声明在用于 SOAP 封装的命名空间中:
    - http://www.w3.org/2001/12/soap-envelope
  - 以及针对 SOAP 编码和数据类型的默认命名空间:
    - http://www.w3.org/2001/12/soap-encoding



5.2 SOAP语法

- **重要的语法规则**
  - SOAP 消息必须用 XML 来编码
  - SOAP 消息必须使用 SOAP Envelope 命名空间
  - SOAP 消息必须使用 SOAP Encoding 命名空间
  - SOAP 消息不能包含 DTD（文档类型定义）引用
  - SOAP 消息不能包含 XML 处理指令

  - 例：下面是一个文档定义，这在SOAP中是不允许的。
    - <!DOCTYPE root-element [element-declarations]>

5.2 SOAP语法

- SOAP 消息的基本结构

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
      …
      …
    </soap:Header>

    <soap:Body>
          …
          …
          <soap:Fault>
            …
            …
          </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

5.2 SOAP语法

- SOAP Envelope 元素
  - SOAP 的 Envelope 元素是 SOAP 消息的根元素。
  - 它可把 XML 文档定义为 SOAP 消息。
  - 请注意 xmlns:soap 命名空间的使用。它的值应当始终是:

http://www.w3.org/2001/12/soap-envelope

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  …
  Message information goes here
  …
</soap:Envelope>
```

5.2 SOAP语法

- xmlns:soap 命名空间
  - SOAP 消息必须拥有与命名空间 "http://www.w3.org/2001/12/soap-envelope" 相关联的一个 Envelope 元素。
  - 如果使用了不同的命名空间,应用程序会发生错误,并抛弃此消息
- encodingStyle 属性
  - 语法:soap:encodingStyle="URI"

5.2 SOAP语法

- 实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding" >
...
Message information goes here
...
</soap:Envelope>
```

5.2 SOAP语法

- SOAP Header 元素
  - 可选的
  - 可包含应用程序专用信息（比如认证、支付等）。
  - 如果 Header 元素被提供,则它必须是 Envelope 元素的第一个子元素。头元素的所有直接子元素称作条目。
  - 注:所有 Header 元素的直接子元素必须是合法的命名空间。
  - SOAP为相互通信的程序之间提供了一种很灵活的机制:
    - 在无须预先协定的情况下,以分散但标准的方式扩展消息。
    - 可以在SOAP头中添加条目实现这种扩展,典型的例子有认证,事务管理,支付等等。

5.2 SOAP语法

例:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
      <m:Trans
      xmlns:m="http://www.w3school.com.cn/transaction/"
      soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
...
</soap:Envelope>
```

- 上面的例子包含了一个带有一个 "Trans" 元素的头部,它的值是 234,此元素的 "mustUnderstand" 属性的值是 "1"。
- SOAP 在默认的命名空间中 ("http://www.w3.org/2001/12/soap-envelope") 定义了三个属性:actor、 mustUnderstand 以及 encodingStyle。
- 这些在 SOAP 头部的属性定义了如何对 SOAP 消息进行处理。

5.2 SOAP语法

- actor 属性
  - 一个消息从始点到终点可能经过多个中间结点，
  - 如果一个SOAP消息的某部分要传给消息路径上的一个或多个中间结点,则可用actor 指定。
  - SOAP 的 actor 属性可被用于将 Header 元素寻址到一个特定的端点
  - 语法:soap:actor="URI"

5.2 SOAP语法

- 实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
      <m:Trans
      xmlns:m="http://www.w3school.com.cn/transaction/"
      soap:actor="http://www.w3school.com.cn/appml/ ">
            234
      </m:Trans>
</soap:Header>
……
</soap:Envelope>
```

  - URI " http://www.w3school.com.cn/appml/"指出了第一个处理这个消息的SOAP应用程序需要这个头元素。省略SOAP actor属性表示接收者是SOAP消息的终节点。

5.2 SOAP语法

- mustUnderstand 属性
  - SOAP mustUnderstand全局属性用来指示接受者在处理消息时这个条目是否必须处理
  - 假如在 Header 元素的某个子元素添加了 mustUnderstand="1"
    - 那么条目的接受者必须
      - 或者遵守语义(如以元素的全名传送)并按照语义正确的处理
      - 或者放弃处理消息。
  - 语法:soap:mustUnderstand="0|1"

5.2 SOAP语法

❖ 实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
      <m:Trans
      xmlns:m="http://www.w3school.com.cn/transaction/"
      soap: mustUnderstand ="1">
      234
      </m:Trans>
</soap:Header>
…
…
</soap:Envelope>
```

5.2 SOAP语法

- SOAP Body
  - SOAP Body 元素包含实际的 SOAP 消息。
  - SOAP Body 元素的直接子元素可以是合法的命名空间。
  - SOAP 在默认的命名空间 "http://www.w3.org/2001/12/soap-envelope"中定义了 Body 元素内部的一个元素-- Fault 元素,用于指示错误消息。

5.2 SOAP语法

❖ SOAP Body举例

- 请求苹果价格的消息

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3school.com.cn/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

- 上面的 m:GetPrice 和 Item 元素是应用程序专用的元素。它们并不是 SOAP 标准的一部分。

5.2 SOAP语法

■ 返回苹果价格的消息

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3school.com.cn/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

5.2 SOAP语法

- SOAP Fault 元素
  - 是可选的
  - 用于指示错误消息。
  - 如果提供了 Fault 元素,则它必须是 Body 元素的子元素。在一条 SOAP 消息中,Fault 元素只能出现一次。
  - SOAP 的 Fault 元素拥有下列子元素:

子元素	描述
<faultcode>	供识别故障的代码
<faultstring>	可供人阅读的有关故障的说明
<faultactor>	有关是谁引发故障的信息
<detail>	存入涉及 Body 元素的应用程序专用错误信息

5.2 SOAP语法

- SOAP Fault 代码
  - 下面定义的是可能出现在faultcode中信息

错误	描述
VersionMismatch	SOAP Envelope 元素的无效命名空间
MustUnderstand	Header 元素的一个直接子元素（带有设置为 "1" 的 mustUnderstand 属性）无法被理解
Client	消息被不正确地构成,或包含了不正确的信息。
Server	服务器有问题,因此无法处理进行下去。

<faultcode>SOAP:Client</faultcode>

5.3 SOAP编码

- SOAP编码格式基于一个简单的类型系统
  - 概括了程序语言,数据库和半结构化数据等类型系统的共同特性。
  - 一个类型或者是一个简单的（标量的）类型,或者是由几个部分组合而成的复合类型,其中每个部分都有自己的类型。

5.3 SOAP编码

- XML中的类型编码规则
  - XML允许非常灵活的数据编码方式，但SOAP定义了一个较小的规则集合，下面的术语用来描述编码规则:
  - “value”是一个字符串(string)，一个可量度对象(数字、日期、枚举)的名字、或是数个简单值的组合。所有的值都有明确的类型。。
  - “simple value”是一个不可分的值,它没有名部分, 如特定的字符串,整数,枚举值等等。
  - “compound value”是相关的值的结合,如定单,股票报表,等。
    - 在“compound value”中,每个相关的值都潜在的以名,序数或这两者来区分，这叫作“accessor”(访问器)。

5.3 SOAP编码

- “array”是一个复合值,成员值按照在数组中的位置相互区分。
- “struct”也是一个复合值,成员值之间的唯一区别是accessor名,accessor名互不相同。
- “simple type”是简单值的类,如叫做"string" "integer"的类,还有枚举类等等。
- “compound type”是复合值的类。复合类型的例子有定单类,它们有相同的accessor名（shipTo, totalCost等）,但可能会有不同的值（可能以后被设置为确定的值）。

5.3 SOAP编码

- 简单类型(simple type)
  - SOAP采用了"XML Schema Part 2: Datatypes"规范[11]"Built-in datatypes"节中的所有类型作为简单类型,包括值和取值范围

类型	举例
int	58502
float	314159265358979E+1
negativeInteger	-32768
string	Louis "Satchmo" Armstrong

5.3 SOAP编码

- 在XML Schema规范中声明的数据类型可以直接用在元素schema中,也可以使用从这些类型衍生的新类型

```
<element name="age" type="int"/>
<element name="height" type="float"/>
<element name="displacement" type="negativeInteger"/>
<element name="color">
    <simpleType base="xsd:string">
        <enumeration value="Green"/>
        <enumeration value="Blue"/>
    </simpleType>
</element>
<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
<color>Blue</color>
```

5.3 SOAP编码

- Enumerations
  - "Enumeration"作为一个概念表示不同的名字的集合。
  - 一个枚举就是对应于基类型的不同的值的列表
  - 例如
    - 集合("Green", "Blue", "Brown")是基于字符串类型的枚举
    - ("1", "3", "5")是一个基于整型数的枚举

```
<element name="EyeColor" type="tns:EyeColor" />
<simpleType name="EyeColor base="xsd:string" >
    <enumeration value="Green" />
    <enumeration value="Blue" />
    <enumeration value="Brown" />
</simpleType>
<Person>
   <Name>Henry Ford</Name>
   <Age>32</Age>
   <EyeColor>Brown</EyeColor>
</Person>
```

5.3 SOAP编码

- 复合类型
  - Struct
    - 其成员的存取标识名是相互区别的唯一标志,应彼此各不相同。
  - Array
    - 其成员的顺序位置是相互区别的唯一标志。

5.3 SOAP编码

- 复合值及对值的引用
  - 复合值的成员被编码为存取标识元素。
  - 存取标识由他们的名字来相区别(例如在struct里面),而元素名就是存取标识名。
  - 存取标识名是局部的,作用域是包含他们的类型中,具备一个未修饰的元素名,而其他则有完全修饰名。
  - 下面是一个“Book”结构的例子:

```
<e:Book xmlns:e="http://example.org/2001/06/books" >
   <author>Henry Ford</author>
   <preface>Prefactory text</preface>
   <intro>This is a book.</intro>
</e:Book>
```

5.3 SOAP编码

- 下面则是一个描述该结构的模式(Schema)片段:

```
<xs:element name="Book"
xmlns:xs='http://www.w3.org/2001/XMLSchema' >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="author" type="xs:string" />
      <xs:element name="preface" type="xs:string" />
      <xs:element name="intro" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

5.3 SOAP编码

- 数组
  - 数组被定义为类型为“enc:Array”
  - 数组值的是一个该数组组成元素的一个有序序列。
  - 一个数组值的元素名可以是任意，因为数组值通过位置区分。
  - 下面是一个模式的片段声明了一个数组:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:enc="http://www.w3.org/2001/06/soap-encoding" >
<xs:import namespace="http://www.w3.org/2001/06/soap-encoding" />
  <xs:element name="myFavoriteNumbers" type="enc:Array" />
</xs:schema>
```

5.3 SOAP编码

- 根据上面的模式,给数组赋值

```
<myFavoriteNumbers
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:enc="http://www.w3.org/2001/06/soap-encoding"
enc:arrayType="xs:int[2]" >
    <number>3</number>
    <number>4</number>
</myFavoriteNumbers>
```

主要内容

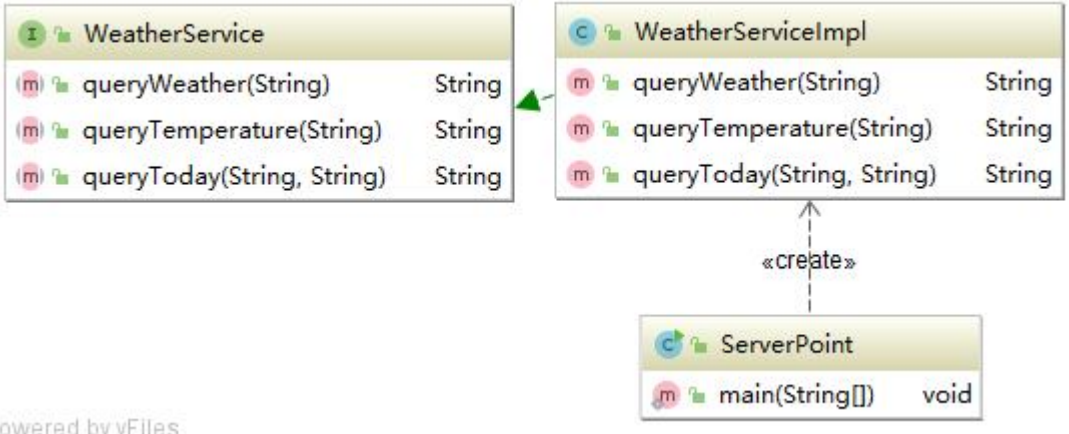
1. 什么是WebService
2. WebService与RPC
3. WSDL
4. UDDI
5. SOAP
6. 举例

6.举例

- 以Java 语言为例
  - webservice程序的创建
    - 可以借助IDE, 比如IntelliJ Idea 或 Eclipse 通过程序向导直接生成webservice 工程
    - 创建一个普通Java工程,然后直接编写webservice程序
  - webservice的发布
    - webservice的发布一般都是使用WSDL文件的来发布的
    - 在WSDL文件里面,包含这个webservice暴露在外面可供使用的接口。

6.1举例-服务程序

- 创建普通Java工程,编写Webservice
  - 共三个类:
    - 一个服务接口类:WeatherService
    - 一个服务接口的实现类:WeatherServiceImpl
    - 一个启动服务类:ServerPoint



6.1举例-服务程序

- 接口类-WeatherService

```
package com.webservice.gyws;

public interface WeatherService {

    //查询天气的方法
    public String queryWeather(String cityName);

    //查询温度
    public String queryTemperature(String cityName);

    public String queryToday(String cityName,String strWhat);

}
```

6.1举例-服务程序

- 接口类的实现-WeatherServiceImpl

```
package com.webservice.gyws;
import javax.jws.WebService;

@WebService
public class WeatherServiceImpl implements WeatherService {
    //查询天气
    public String queryWeather(String cityName) {
        String strResult="Not Found";
         if(cityName.equals("北京"))
        {
            strResult=cityName+":晴";
        }else if(cityName.equals("哈尔滨"))
        {
            strResult=cityName+":大雪";
        }else  {
            strResult=cityName+":晴转多云";
        }
        System.out.println(strResult);
        return strResult;
    }
    … …
}
```

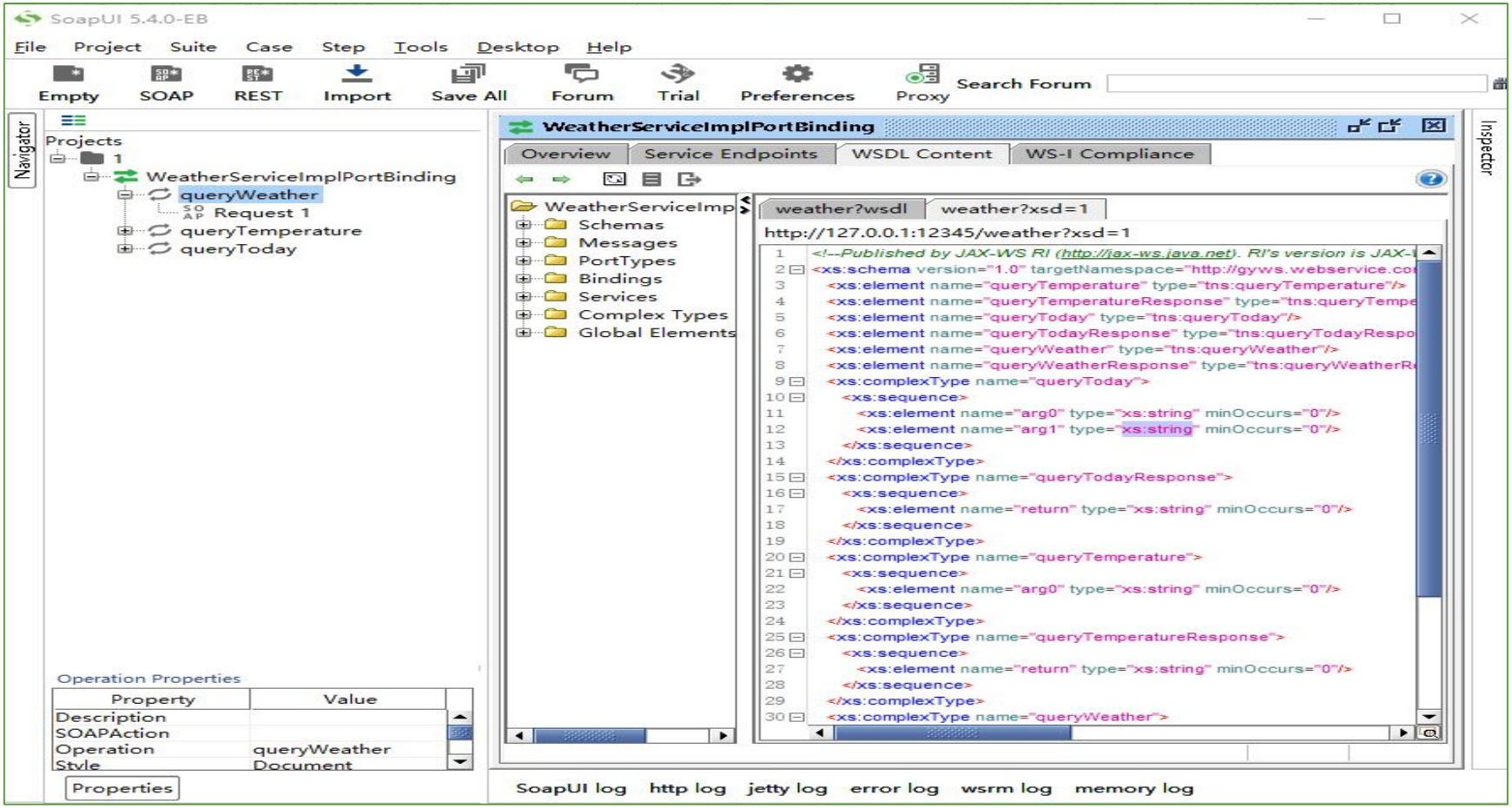
6.1举例-服务程序

- 一个启动服务类:ServerPoint

```
package com.webservice.gyws;
import javax.xml.ws.Endpoint;

public class ServerPoint
{
    public static void main(String[] args) {
        //参数1:服务地址,weather为服务的名称
        //参数2:服务实现类
        String strServicePort="http://127.0.0.1:12345/weather";
        Endpoint.publish(strServicePort, new WeatherServiceImpl());

        System.out.println("Service Port:"+strServicePort);
    }
}
```

6.1举例-服务程序

- **服务程序测试:**
  - 可以使用SoapUI 等工具直接进行测试



6.2 服务的调用

```
package com.webservice.method1;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
import java.rmi.RemoteException;

public class clientMethod1 {
static Call call;
public static void main(String[] args) {
   try {
      String url = "http://127.0.0.1:12345/weather?wsdl";
      Service service = new Service();
      call = (Call) service.createCall();
      call.setTargetEndpointAddress(url);
      String result=queryWeather("北京");
      System.out.println("result is " + result);
      result=queryToday("哈尔滨","Temperature");
      System.out.println("result is " + result);

   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

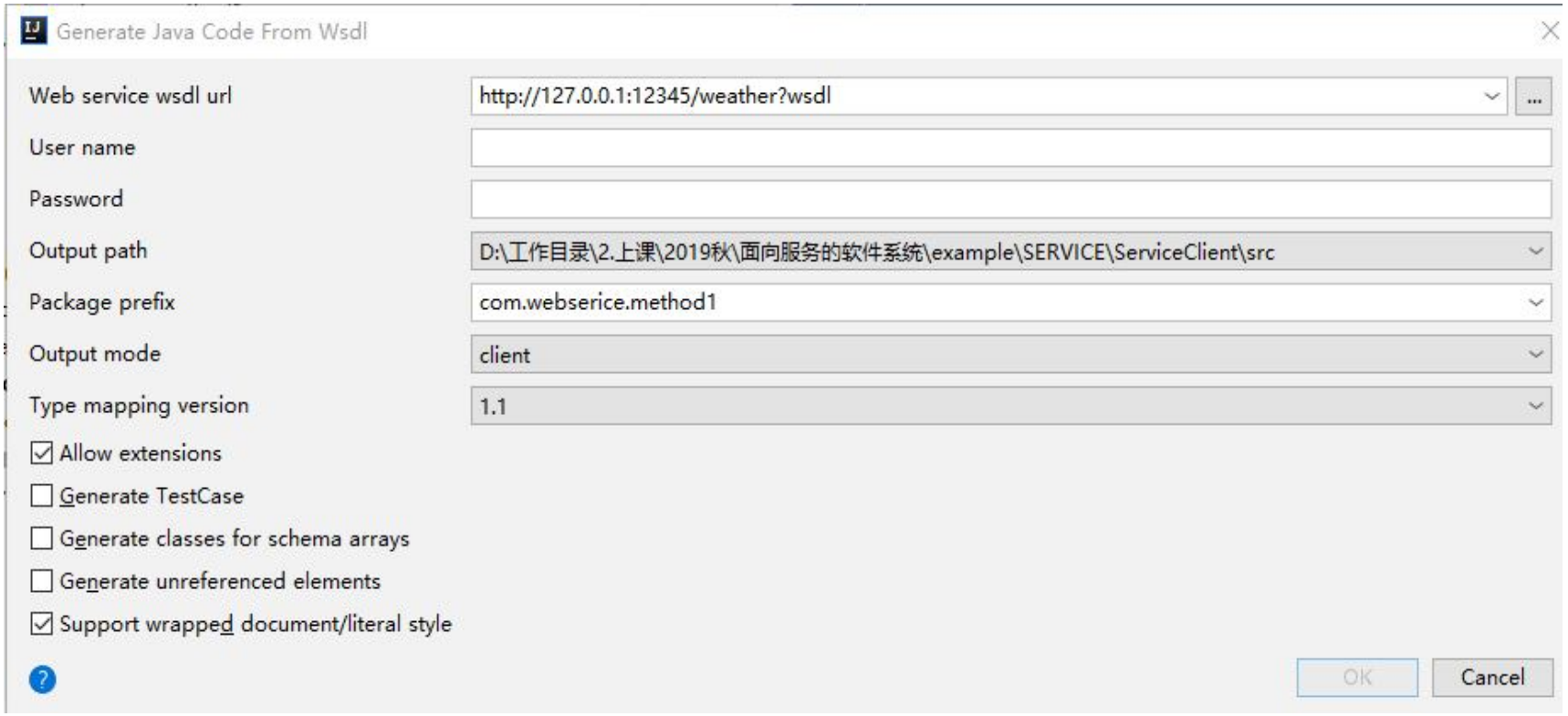
6.2 服务的调用

```
static public String queryWeather(String city) throws RemoteException
 {

      //“http://gyws.webservice.com/”为命名空间,在wsdl 中可以找到
       call.clearOperation();
       QName qName = new QName("http://gyws.webservice.com/", "queryWeather");

       // WSDL里面描述的接口名称
       call.setOperationName(qName);
      // 接口的参数
        call.addParameter("arg0", XMLType.XSD_STRING, ParameterMode.IN);
      // 设置返回类型
       call.setReturnType(XMLType.XSD_STRING);
       String temp = city;
      // 给方法传递参数,并且调用方法
       String result = (String) call.invoke(new Object[ ]{temp});
      return result;

   }
}
```

6.2 服务的调用

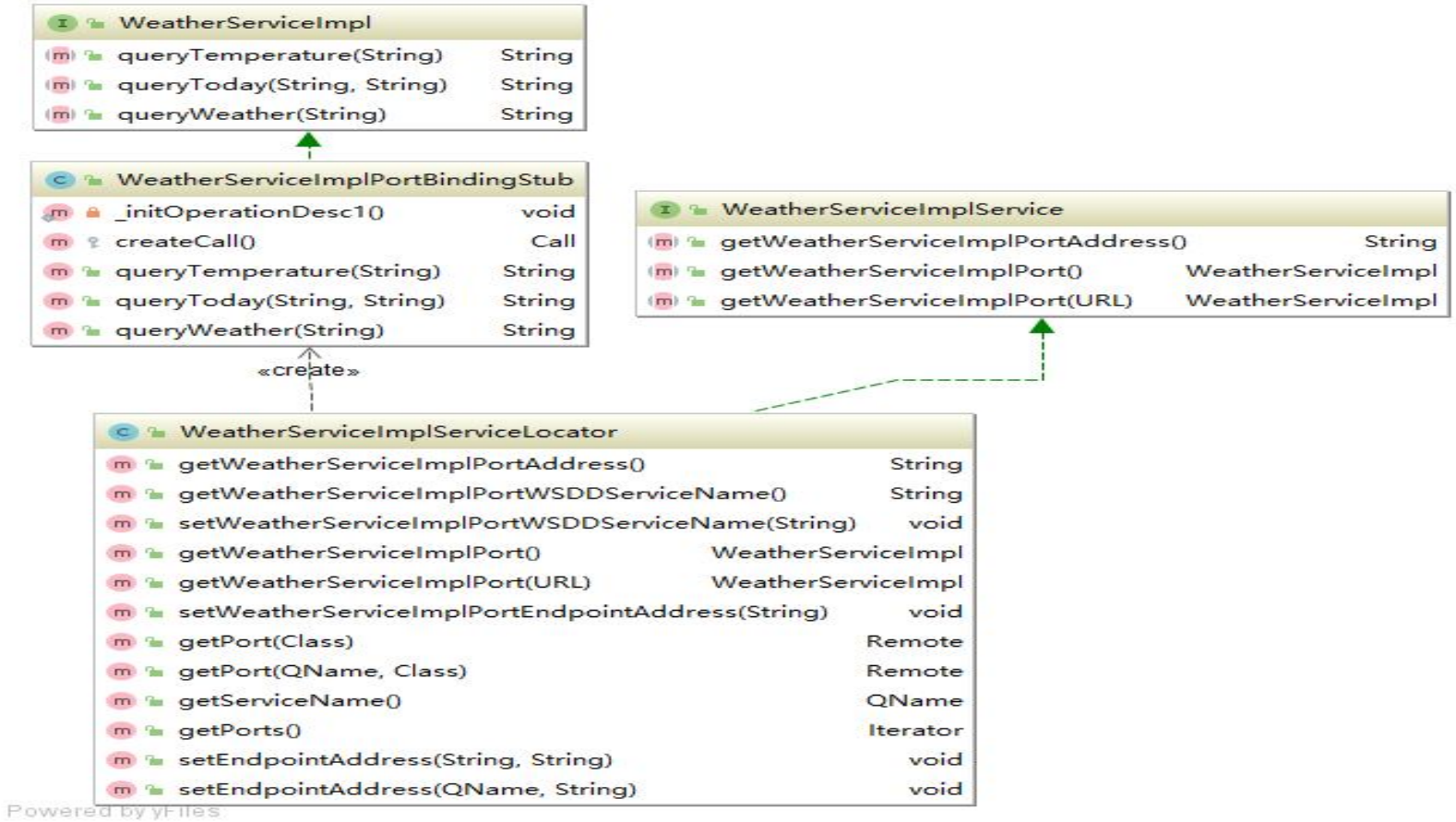
2.用IDE插件或使用wsdl2java把WSDL文件转成本地类

- 首选启动服务
- 在IDEA中选中要使用服务的工程,点击右键,然后选中webservice -» Generate Java code from wsdl.



6.2 服务的调用

- 自动生成如下类
- 可以通过WeatherServiceImplServiceLocatory调用服务



6.2 服务的调用

❖ 示例代码

```
package com.webserice.method1;
import javax.xml.rpc.ServiceException;
import java.rmi.RemoteException;
public class clientMethod1 {
    public static void main(String[] args) {
        WeatherServiceImplServiceLocator wsis= new WeatherServiceImplServiceLocator();
        //获取服务实现类
        WeatherServiceImpl wsi = null;
        try {
            wsi = (WeatherServiceImpl) wsis.getPort(WeatherServiceImpl.class);
        } catch (ServiceException e) {
            e.printStackTrace();
        }
        //调用查询方法
        String weather = null;      String temperature=null;      String strRes=null;
        try {
            weather = wsi.queryWeather("北京");
            temperature= wsi.queryTemperature("哈尔滨");
            strRes=wsi.queryToday("北京","Weather");

        } catch (RemoteException e) {
            e.printStackTrace();
        }
        System.out.println(weather);
        System.out.println(temperature);
        System.out.println(strRes);
    }
}
```

6.2 服务的调用

3.直接SOAP调用远程的webservice

```
public static void main(String[] args)
  {
      final String Url = "http://127.0.0.1:12345/weather?wsdl";
      final String namespace = "http://gyws.webservice.com/";
      final String method = "queryWeather";
      final String paraName = "arg0";
      final String paraValue = "北京";
         try {
             Service service = new Service();
             Call call = (Call) service.createCall();
             //访问路径
             call.setTargetEndpointAddress(Url);
             //访问的方法名称
             call.setOperationName(new QName(namespace, method));
             //接口参数
             call.addParameter("arg0", XMLType.XSD_STRING, ParameterMode.IN);
             //使用SOAP方式请求
             call.setUseSOAPAction(true);
             // 设置请求的路径
             call.setSOAPActionURI(namespace + method);
             //设置返回结果类型
             call.setReturnType(XMLType.XSD_STRING);
             //开始执行,并获取结果
             String obj = (String)call.invoke(new Object[]{paraValue});
             System.out.println(obj);
         } catch (Exception e) {
             e.printStackTrace();
         }
  }
```

谢 谢!