

# WebService

---

## WebService

回顾：

今天任务

教学目标

### 第一章 什么是WebService?

#### 1.1 什么是远程调用技术

#### 1.2 WebService的原理图

### 第二章 WebService入门程序

#### 2.1 需求

#### 2.2 环境

#### 2.3 实现

##### 2.3.1 服务端

开发步骤：

##### 2.3.2 客户端

注意：

开发步骤：

##### 2.3.3 WebService的优缺点

a.优点：

b.缺点：

### 第三章 WebService的应用场景

#### 3.1 软件集成与复用

#### 3.2 适用场景

#### 3.3 不适用场景

### 第四章 WebService的三要素

#### 4.1 SOAP:简单对象访问协议

##### 4.1.1 定义

##### 4.2.1 协议格式

##### 4.2.2 TCP/IP Monitor

##### 4.2.3 SOAP1.1

##### 4.2.4 SOAP1.2

##### 4.2.5 SOAP1.1和SOAP1.2区别

#### 4.2 WSDL:Web服务描述语言

##### 4.2.1 定义

##### 4.2.2 文档结构

##### 4.2.3 阅读方式从下往上

#### 4.3 UDDI:目录服务

### 第五章 WebService客户端调用方式

#### 5.1 生成客户端调用方式

##### 5.1.1 wsimport 命令介绍

##### 5.1.2 调用公网手机号归属地查询服务

##### 5.1.3 公网天气服务端查询

##### 5.1.4 特点

#### 5.2 客户端编程调用方式

#### 5.3 HttpURLConnection调用方式

- 第一步 创建服务地址
- 第二步 打开一个通向服务地址的连接
- 第三步 设置参数
- 第四步 组织SOAP数据，发送请求
- 第五步 接收服务端响应，打印
- 第六章 深入开发：用注解修改WSDL内容
  - 6.1 JAX-WS注解
    - 6.1.1 注解说明
    - 6.1.2 注解示例
    - 6.1.3 使用注解注意的地方
- 课前默写
- 作业
- 面试题

## 回顾：

1. Spring整合MyBatis框架的配置详解
2. Spring整合MyBatis框架的具体操作

## 今天任务

1. WebService简介
2. WebService基础应用
3. WebService的基本要素
4. WebService的客户端调用方式
5. 使用注解修改WSDL内容

## 教学目标

1. 掌握WebService的基本概念
2. 掌握WebService基础应用
3. 掌握WebService的基本要素
4. 掌握WebService的客户端调用方式
5. 掌握如何使用注解修改WSDL内容

## 第一章 什么是WebService?

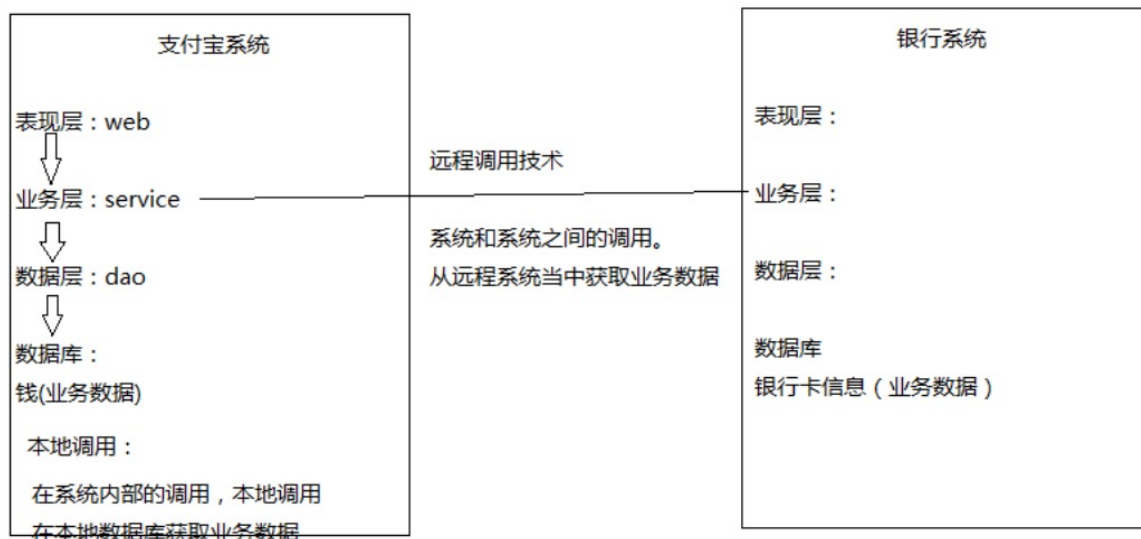
web Service也叫XML Web Service，Web服务。可使用开放的xml标准来描述、发布、发现、协调和配置这些应用程序。用于开发分布式的互操作的应用程序。

是一种跨编程语言、跨操作系统、跨网络的远程服务器调用技术。

☑️Webservice使用http传输SOAP协议的数据的一种远程调用技术

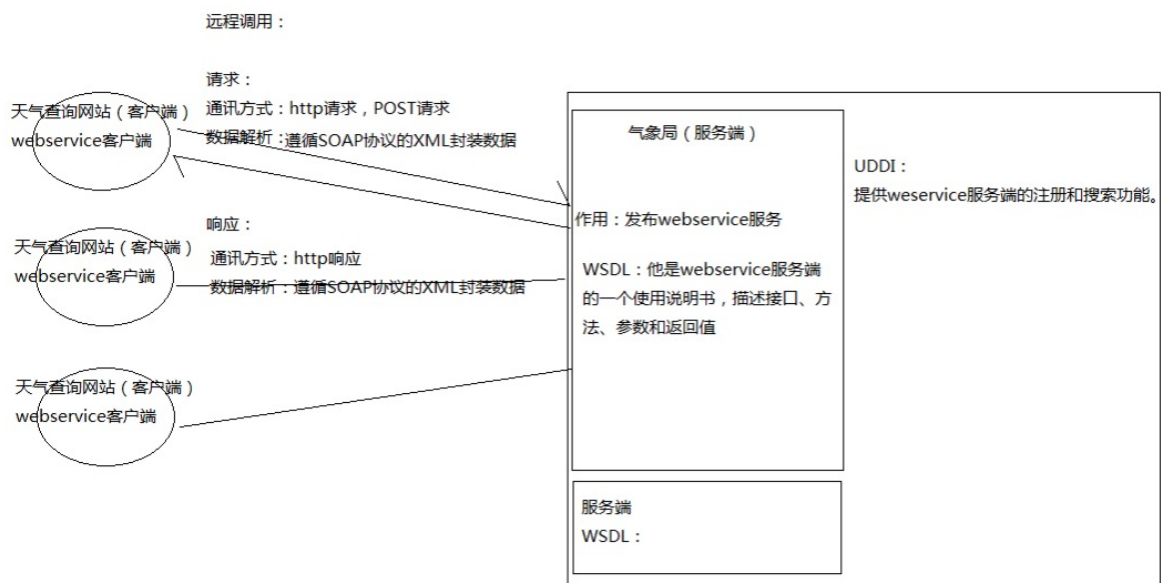
## 1.1 什么是远程调用技术

远程调用数据定义：是系统和系统之间的调用。



## 1.2 WebService的原理图

1. WebService是使用Http发送SOAP协议的数据的一种远程调用技术
2. WebService要开发服务端
3. WebService要开发客户端
4. WebService客户端开发需要阅读服务端的使用说明书 (WSDL)



## 第二章 WebService入门程序

### WebService开发规范

1. JAX-WS: java api for xml web service
2. JAXM: java api from xml message主要定义了包含了发送和接收消息所需的API

### 3. JAX-RS:java api from xml RESTFUL

JAVA 针对REST(Representation State [Transfer](#))风格制定的一套Web 服务规范

以下演示JAX-WS案例:

## 2.1 需求

- 服务端: 发布一个天气查询服务, 接收客户端城市名, 返回天气数据给客户端
- 客户端: 发送城市名称给服务端, 接收服务端的返回天气数据, 打印

## 2.2 环境

JDK:1.8

Eclipse:2017

## 2.3 实现

### 2.3.1 服务端

开发步骤:

第一步:创建SEI (Service Endpoint Interface) 接口, 本质上就是Java接口

```
package com.sky.webservice;  
  
public interface WeatherInterface {  
    public String queryWeather(String cityName);  
}
```

第二步:创建SEI实现类, 在实现类上加入@WebService

//@WebService表示该类是一个服务类, 需要发布其中的public的方法

@WebService

```
public class WeatherInterfaceImpl implements WeatherInterface {  
    @Override  
    public String queryWeather(String cityName) {  
        // TODO Auto-generated method stub  
        System.out.println("from client...."+cityName);  
        String weather="晴";  
        return weather;  
    }  
}
```

第三步:发布服务, Endpoint发布服务, publish方法, 两个参数: 1.服务地址; 2.服务实现类

```
public class WeatherServer {  
    public static void main(String[] args) {  
        /**  
         * Endpoint发布服务  
         * 参数1: 服务地址  
         * 参数2: 实现类  
         */  
        Endpoint.publish("http://127.0.0.1:8088/weather", new  
WeatherInterfaceImpl());  
    }  
}
```

第四步: 测试服务是否发布成功, 通过阅读使用说明书, 确定客户端调用的接口、方法、参数和返回值存在, 证明服务发布成功。

WSDL地址: 服务地址+"?wsdl"

WSDL阅读方式: 从下往上

```
<service name="WeatherInterfaceImplService">
  <port name="WeatherInterfaceImplPort" binding="tns:WeatherInterfaceImplPortBinding">
    <soap:address location="http://127.0.0.1:8088/weather"/>
  </port>
</service>

<binding name="WeatherInterfaceImplPortBinding" type="tns:WeatherInterfaceImpl">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="queryWeather">
    <soap:operation soapAction="">
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

<portType name="WeatherInterfaceImpl">
  <operation name="queryWeather">
    <input message="tns:queryWeather" wsam:Action="http://webservice.sky.com/WeatherInterfaceImpl/queryWeatherRequest"/>
    <output message="tns:queryWeatherResponse" wsam:Action="http://webservice.sky.com/WeatherInterfaceImpl/queryWeatherResponse"/>
  </operation>
</portType>
```

### 2.3.2 客户端

注意:

1. 要先发布服务, 服务运行的状态下生成代码
2. -s后面有个小点, 用于指定源代码生成的目录。点即当前目录。
3. 运行客户端必须开启远程服务

开发步骤:

第一步: wsimport命令生成客户端代码(通过cmd进入到当前项目的src路径下, wsimport.exe命令是在jdk的bin目录下)

wsimport -s . http://127.0.0.1:8088/weather?wsdl

第二步: 根据使用说明书, 使用客户端代码调用服务端

2.1 创建服务视图, 视图是从service标签的name属性获取

2.2 获取服务实现类, 实现类从portType的name属性获取

2.3 获取查询方法, 从portType的operation标签获取

```
public class WeatherClient {
    public static void main(String[] args) {
        //1.创建服务视图
        WeatherInterfaceImplService weatherInterfaceImplService =
            new WeatherInterfaceImplService();
        //2.获取服务实现类
        WeatherInterfaceImpl weatherInterfaceImpl =
            weatherInterfaceImplService.getPort(WeatherInterfaceImpl.class);
        //3.调用查询方法, 打印
        String result = weatherInterfaceImpl.queryWeather("北京");
        System.out.println(result);
    }
}
```

### 2.3.3 WebService的优缺点

#### a.优点:

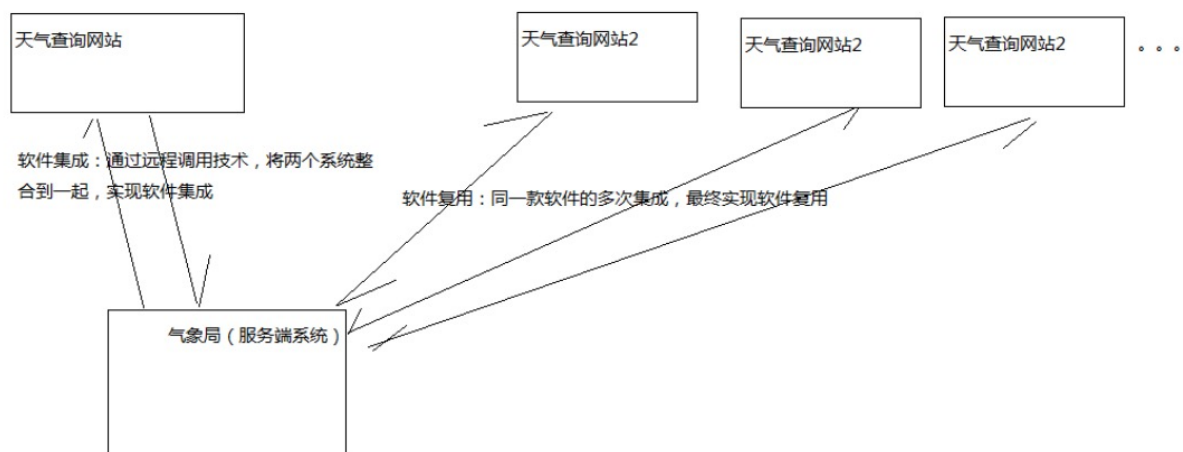
- 1.发送方式采用http的post发送，http的默认端口是80，防火墙默认不拦截80，所以跨防火墙。
- 2.采用XML格式封装数据，XML是跨平台的，所以webservice也是跨平台的。
- 3.webservice支持面向对象

#### b.缺点:

❑采用XML格式封装数据，所以在传输过程中，要传输额外的标签，随着SOAP协议的不断完善，标签越来越大，导致webservice性能下降

## 第三章 WebService的应用场景

### 3.1 软件集成与复用



### 3.2 适用场景

- 发布一个服务（对内/对外），不考虑客户端类型，不考虑性能，建议使用webservice
- 服务端已经确定使用webservice，客户端不能选择，必须使用webservice

### 3.3 不适用场景

- 考虑性能时不建议使用webservice
- 同构程序下不建议使用webservice，比如java 用RMI，不需要翻译成XML的数据

## 第四章 WebService的三要素

### 4.1 SOAP:简单对象访问协议

#### 4.1.1 定义

SOAP:(Simple Object Access Protocol)简单对象访问协议。是XML Web Service 的通信协议.当用户通过UDDI找到 你的WSDL描述文档后, 他通过可以SOAP调用你建立的Web服务中的一个或多个操作。

SOAP是XML文档形式的调用方法的规范, 它可以支持不同的底层接口, 像HTTP(S)或者SMTP.

☐SOAP=http+xml

#### 4.2.1 协议格式

##### *http请求*

```
POST:/Weather http/1.1
Host:127.0.0.1:8088
Content-Type:text/html;charset=utf-8
***** (其他请求头)

cityName=北京&method=queryWeather
```

##### *soap请求*

```
POST:/Weather http/1.1
Host:127.0.0.1:8088
Content-Type:text/xml;charset=utf-8
***** (其他请求头)

<envelope>
  <body>
    <cityName>北京</cityName>
    <method>queryWeather</method>
  </body>
</envelope>
```

1. 必须有Envelope元素, 此元素将整个XML文档标识为一条SOAP消息
2. 可选的Header元素, 包含头部信息
3. 必须有Body元素, 包含所有的调用和响应信息
4. 可选的Fault元素, 提供☐有关在处理此消息所发生错误的信息

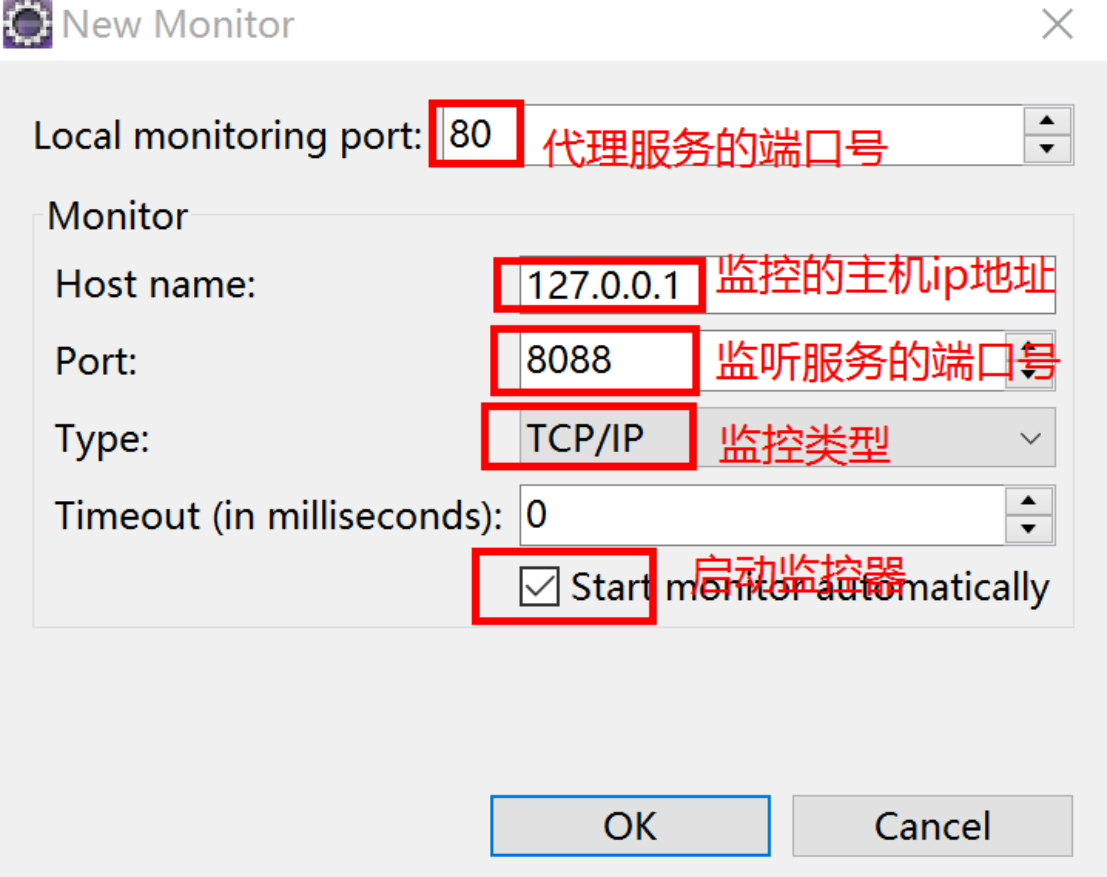
#### 4.2.2 TCP/IP Monitor

- 代理原理

客户端不直接去访问服务端的内容, 而是通过代理服务去访问服务端的内容

- 配置

1. 配置显示监控页面: Eclipse菜单栏中Window—Show View—Debug—TCP/IP Monitor
2. 配置监控器: Eclipse菜单栏中Window—Preferences— Run/Debug—TCP/IP Monitor—Add



The image shows a 'New Monitor' dialog box with the following fields and annotations:

- Local monitoring port:** 80 (Annotated: 代理服务的端口号)
- Monitor section:**
  - Host name:** 127.0.0.1 (Annotated: 监控的主机ip地址)
  - Port:** 8088 (Annotated: 监听服务的端口号)
  - Type:** TCP/IP (Annotated: 监控类型)
  - Timeout (in milliseconds):** 0
  - ☒ Start monitor automatically (Annotated: 启动监控器)
- Buttons:** OK, Cancel

- 测试

在浏览器中输入代理服务地址，能正常访问，代表代理服务器设置成功

#### 4.2.3 SOAP1.1

使用代理服务端口号和地址进行测试

- 请求

```
POST /weather HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction:
"http://webservice.sky.com/WeatherInterfaceImpl/queryWeatherRequest"
User-Agent: JAX-WS RI 2.2.9-b130926.1035 svn-
revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e
Host: 127.0.0.1
Connection: keep-alive
Content-Length: 211

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:queryWeather xmlns:ns2="http://webservice.sky.com/">
      <arg0>北京</arg0>
    </ns2:queryWeather>
  </S:Body>
```



```
</S:Envelope>
```

- 响应

```
HTTP/1.1 200 OK
Date: Tue, 31 Oct 2017 07:45:09 GMT
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8

e4
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body><ns2:queryWeatherResponse
xmlns:ns2="http://webservice.sky.com/">
    <return>晴</return>
  </ns2:queryWeatherResponse>
</S:Body>
</S:Envelope>

0
```

#### 4.2.4 SOAP1.2

- 发布SOAP1.2服务端,在实现类上加入如下注解:  
解:@BindingType(SOAPBinding.SOAP12HTTP\_BINDING)

备注：在某些版本上不支持SOAP1.2服务端发布，直接发布异常。

需要在服务端引入第三方JAR（jaxws-ri-2.2.8\jaxws-ri\lib下所有jar包）

- 请求

```
POST /weather HTTP/1.1
Accept: application/soap+xml, multipart/related
Content-Type: application/soap+xml; charset=utf-8;action="http://webservice.sky.com/WeatherInterfaceImpl/queryWeatherRequest"
User-Agent: JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e
Host: 127.0.0.1
Connection: keep-alive
Content-Length: 209

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns2:queryWeather xmlns:ns2="http://webservice.sky.com/">
      <arg0>北京</arg0>
    </ns2:queryWeather>
  </S:Body>
</S:Envelope>
```

- 响应

```
HTTP/1.1 200 OK
Date: Tue, 31 Oct 2017 08:07:17 GMT
Transfer-encoding: chunked
Content-type: application/soap+xml; charset=utf-8

e2
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns2:queryWeatherResponse xmlns:ns2="http://webservice.sky.com/">
      <return>晴</return>
    </ns2:queryWeatherResponse>
  </S:Body>
</S:Envelope>

0
```

#### 4.2.5 SOAP1.1和SOAP1.2区别

##### ☐相同点：

- ☐请求发送方式相同：都是使用POST
- ☐协议内容相同：都有Envelope和Body标签

##### ☐不同点：

- ☐数据格式不同：content-type不同
  - ☐SOAP1.1：text/xml; charset=utf-8
  - ☐SOAP1.2：application/soap+xml; charset=utf-8
- ☐命名空间不同：
  - ☐SOAP1.1：http://schemas.xmlsoap.org/soap/envelope/
  - ☐SOAP1.2：http://www.w3.org/2003/05/soap-envelope

## 4.2 WSDL:Web服务描述语言

### 4.2.1 定义

WSDL:Web Services Description Language.

是基于 XML 的用于描述Web Service及其函数、参数和返回值。通俗理解Wsd1是webservice的使用说明书.大多数情况下由软件自动生成和使用。

一般用于Web Service的发布的服务的说明文档。

### 4.2.2 文档结构

```

<definitions name="WeatherInterfaceImplService" targetNamespace="http://webservice.sky
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://webservice.sky.com
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsp1_2="http://
xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsu="http://docs.oasis-open.org/w
+ <types>
+ <message name="queryWeather">
+ <message name="queryWeatherResponse">
+ <portType name="WeatherInterfaceImpl">
+ <binding name="WeatherInterfaceImplPortBinding" type="tns:WeatherInterfaceImpl">
+ <service name="WeatherInterfaceImplService">

```

<service>: 服务视图, webservice的服务结点, 它包括了服务端点

<binding>: 为每个服务端点定义消息格式和协议细节

<portType>: 服务端口, 描述 web service可被执行的操作方法, 以及相关的消息, 通过binding指向portType

<message>: 定义一个操作(方法)的数据参数(可多个参数)

<types>: 定义 web service 使用的全部数据类型

### 4.2.3 阅读方式从下往上

```

<types>
- <xsd:schema>
  <xsd:import schemaLocation="http://127.0.0.1:8088/weather?xsd=1" namespace="http://webservice.sky.com/" />
</xsd:schema>
</types>
<message name="queryWeather">
  <part name="parameters" element="tns:queryWeather" />
</message>
<message name="queryWeatherResponse">
  <part name="parameters" element="tns:queryWeatherResponse" />
</message>
<portType name="WeatherInterfaceImpl">
- <operation name="queryWeather">
  <input message="tns:queryWeather" wsam:Action="http://webservice.sky.com/WeatherInterfaceImpl/queryWea
  <output message="tns:queryWeatherResponse" wsam:Action="http://webservice.sky.com/WeatherInterfaceImpl
</operation>
</portType>
<binding name="WeatherInterfaceImplPortBinding" type="tns:WeatherInterfaceImpl">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="queryWeather">
  <soap:operation soapAction="" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<service name="WeatherInterfaceImplService">
- <port name="WeatherInterfaceImplPort" binding="tns:WeatherInterfaceImplPortBinding">
  <soap:address location="http://127.0.0.1:8088/weather" />
</port>

```

阅读方式从下往上，标注了步骤：

- 第一步: 指向 `<port name="WeatherInterfaceImplPort" binding="tns:WeatherInterfaceImplPortBinding">`
- 第二步: 指向 `<binding name="WeatherInterfaceImplPortBinding" type="tns:WeatherInterfaceImpl">`
- 第三步: 指向 `<portType name="WeatherInterfaceImpl">`
- 第四步: 指向 `<operation name="queryWeather">`

### 4.3 UDDI:目录服务

UDDI: Universal Description, Discovery and Integration. 可译为"通用描述, 发现与集成服务"。 服务目录检索

企业可以使用它对 Web services 进行注册和搜索。

企业将自己提供的Web Service注册在UDDI, 也可以使用别的企业在UDDI注册的web service服务, 从而达到资源共享。UDDI旨在将全球的webservice资源进行共享, 促进全球经济合作。

UDDI现状:

目前大部分企业使用webservice并不是必须使用UDDI, 因为用户通过WSDL知道了web service的地址, 可以直接通过WSDL调用webservice。

## 第五章 WebService客户端调用方式

公网服务地址:

[http://www.webxml.com.cn/zh\\_cn/index.aspx](http://www.webxml.com.cn/zh_cn/index.aspx)

### 5.1 生成客户端调用方式

#### 5.1.1 wsimport 命令介绍

1. wsimport就是jdk提供的的一个工具, 作用就是根据WSDL地址生成客户端代码。
2. 位置JAVA\_HOME/bin目录下
3. wsimport常用的参数:
  - s 生成java文件的
  - d 生成class文件的, 默认参数
  - p 指定包名的, 如果不加该参数, 默认包名就是wsdl文档中的命名空间的倒序

#### 5.1.2 调用公网手机号归属地查询服务

第一步: wsimport生成客户端代码

```
wsimport -p com.sky.mobile -s .
```

```
http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl
```

第二步: 阅读使用说明书, 使用生成客户端代码调用服务端

```
package com.sky.mobile;

public class MobileClient {
    public static void main(String[] args) {
        /// 1.创建服务视图
        MobileCodeWS mobileCodeWS = new MobileCodeWS();
        /// 2.获取服务实现类
        MobileCodeWSSoap mobileCodeWSSoap =
mobileCodeWS.getPort(MobileCodeWSSoap.class);
        /// 3.调用查询方法
        String result = mobileCodeWSSoap.getMobileCodeInfo("18600161153", "");
        System.out.println(result);
    }
}
```

### 5.1.3 公网天气服务端查询

```
import java.util.List;
import com.sky.weather.ArrayOfString;
import com.sky.weather.WeatherWS;
import com.sky.weather.WeatherWSSoap;
public class WeatherClient {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        /// 1.创建服务视图
        WeatherWS weatherWS = new WeatherWS();
        // 2.获取服务实现类
        WeatherWSSoap weatherWSSoap = weatherWS.getPort(WeatherWSSoap.class);
        // 3.调用查询方法
        ArrayOfString arrayOfString = weatherWSSoap.getWeather("北京", "");
        List<String> list = arrayOfString.getString();
        for (String string : list) {
            System.out.println(string);
        }
    }
}
```

### 5.1.4 特点

该种方式使用简单，但一些关键的元素在代码生成时写死到生成代码中，不方便维护，所以仅用于测试。

## 5.2 客户端编程调用方式

```
package com.sky.service;
import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;
import java.util.List;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import com.sky.mobile.MobileCodewSSoap;
import com.sky.weather.ArrayOfString;
import com.sky.weather.WeatherWS;
import com.sky.weather.WeatherWSSoap;
public class ServiceClient {
    public static void main(String[] args) throws MalformedURLException {
        /// 1.创建服务视图
        /**
         * 参数1: wsdl地址
         * 参数2: 服务名称
         */
    }
}
```

```
URL url=new URL("http://ws.webxml.com.cn/WebServices/MobileCodeWS.aspx?wsdl");  
/**  
 * 参数1: 命名空间地址  
 *      targetNamespace="http://WebXml.com.cn/"  
 * 参数2: 服务视图名称 ,service的名称值  
 */  
QName qName =new QName("http://WebXml.com.cn/", "MobileCodeWS");  
Service service = Service.create(url, qName);  
// 2.获取服务实现类  
MobileCodeWSSoap mobileCodeWSSoap =  
service.getPort(MobileCodeWSSoap.class);  
// 3.调用查询方法  
String result = mobileCodeWSSoap.getMobileCodeInfo("18888888888", "");  
System.out.println(result);  
}  
}
```

该种方式可以自定义关键元素，方便以后维护，是一种标准的开发方式

## 5.3 HttpURLConnection调用方式

第一步 创建服务地址

第二步 打开一个通向服务地址的连接

第三步 设置参数

设置POST，POST必须大写，如果不大写，报异常

第四步 组织SOAP数据，发送请求

第五步 接收服务端响应，打印

示例：

```
import java.io.BufferedReader;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.Iterator;  
import org.dom4j.Document;  
import org.dom4j.DocumentException;  
import org.dom4j.DocumentHelper;  
import org.dom4j.Element;  
import org.dom4j.Node;  
  
public class HttpClient {  
    public static void main(String[] args) throws Exception {
```

```
// 1.创建服务地址, 不是WSDL地址
URL url = new
URL("http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx");
// 2.打开一个通向服务地址的连接
URLConnection connection = (URLConnection) url.openConnection();
// 3.设置参数
// 3.1发送方式设置: POST必须大写
connection.setRequestMethod("POST");
// 3.2 设置数据格式: content-type
connection.setRequestProperty("content-type", "text/xml;charset=utf-8");
// 3.3设置输入输出, 因为默认新创建的connection没有读写权限
connection.setDoOutput(true);
connection.setDoOutput(true);
// 4. 组织SOAP数据, 发送请求
String aopxml = getXML("18600161153");
OutputStream outputStream = connection.getOutputStream();
outputStream.write(aopxml.getBytes());
// 5.接收服务端响应, 打印
if (200 == connection.getResponseCode()) {
    InputStream is = connection.getInputStream();
    InputStreamReader isr = new InputStreamReader(is);
    BufferedReader br = new BufferedReader(isr);

    StringBuilder sb = new StringBuilder();
    String temp = null;
    while (null != (temp = br.readLine())) {
        sb.append(temp);
    }
    Document document = DocumentHelper.parseText(sb.toString());
    treeWalk(document);
    is.close();
    isr.close();
    br.close();
}
outputStream.close();
}

public static void treeWalk(Document document) {
    treeWalk( document.getRootElement() );
}

public static void treeWalk(Element element) {
    for ( int i = 0, size = element.nodeCount(); i < size; i++ ) {
        Node node = element.node(i);
        if ( node instanceof Element ) {
            treeWalk( (Element) node );
        }
        else {
            // do something....
        }
    }
}
```

```

        if (node.getNodeTypeName().equals("Text")) {
            System.out.println(node.getText());
        }
    }
}

private static String getXML(String string) {
    // TODO Auto-generated method stub
    String xml = "<?xml version='1.0' encoding='utf-8'>\r\n" +
        "<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-  
instance' xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>\r\n" +
        "    <soap:Body>\r\n" +
        "        <getMobileCodeInfo xmlns='http://WebXml.com.cn/'>\r\n" +
        "            <mobileCode>"+string+"</mobileCode>\r\n" +
        "            <userID></userID>\r\n" +
        "        </getMobileCodeInfo>\r\n" +
        "    </soap:Body>\r\n" +
        "</soap:Envelope>";

    return xml;
}
}

```

## 第六章 深入开发：用注解修改WSDL内容

### 6.1 JAX-WS注解

#### 6.1.1 注解说明

WebService的注解都位于javax.jws包下：

@WebService-定义服务，在public class上边

targetNamespace: 指定命名空间

name: portType的名称

portName: port的名称

serviceName: 服务名称

@WebMethod-定义方法，在公开方法上边

operationName: 方法名

exclude: 设置为true表示此方法不是webservice方法，反之则表示

webservice方法

@WebResult-定义返回值，在方法返回值前边

name: 返回结果值的名称

@WebParam-定义参数，在方法参数前边

name: 指定参数的名称

作用：

通过注解，可以更加形象的描述Web服务。对自动生成的wsdl文档进行修改，为使用者提供一个更加清晰的wsdl文档。当修改了WebService注解之后，会影响客户端生成的代码。调用的方法名和参数名也发生了变化



### 6.1.2 注解示例

```
package com.sky.webservice;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

//@WebService表示该类是一个服务类，需要发布其中的public的方法
@WebService(targetNamespace = "http://www.sky.com/", name = "wangwujuan", portName = "skyPort", serviceName = "wangwujuan")
public class WeatherInterfaceImpl implements WeatherInterface {

    @WebMethod(operationName = "query", exclude = false)
    @Override
    public @WebResult(name = "result") String queryWeather(@WebParam(name = "city") String cityName) {
        // TODO Auto-generated method stub
        System.out.println("from client...." + cityName);
        String weather = "晴";
        return weather;
    }
}

package com.sky.webservice;
import javax.xml.ws.Endpoint;
public class WeatherServer {
    public static void main(String[] args) {
        /**
         * Endpoint发布服务
         * 参数1: 服务地址
         * 参数2: 实现类
         */
        Endpoint.publish("http://127.0.0.1:8088/weather", new WeatherInterfaceImpl());
    }
}
```

### 6.1.3 使用注解注意的地方

@WebMethod对所有非静态的公共方法对外暴露为服务。

对于静态方法或非public方法是不可使用@WebMethod注解的。

对public方法可以使用@WebMethod(exclude=true)定义为非对外暴露的服务。

## 课前默写

1. Spring整合MyBatis框架的配置
2. Spring整合MyBatis框架的操作

## 作业

1. 开发一个天气服务
2. 开发一个短信验证码服务(模拟)

## 面试题

1. 什么是WebService, WebService有什么作用
  2. 什么是SOAP、WSDL
  3. WebService一般在什么时候使用, 你在项目中什么地方应用到了
-