



# WebService



# 1 教学计划

时间	内容	目标
第一天	<p>什么是 Webservice</p> <p><b>WSDL 的使用方法</b></p> <p><b>Soap 协议</b></p> <p><b>Jax-ws 编程(重点)</b></p> <p><b>案例 webservice 发送 xml(重点)</b></p>	<p>了解 webservice 的应用场合</p> <p>掌握 wsdl 的使用方法</p> <p>掌握 soap1.1、soap1.2 定义方法</p> <p>掌握 jax-ws 编写 webservice 服务端、客户端</p> <p>掌握 webservice 发送 xml 数据方法</p>
第二天	<p>Jax-ws 编程深入</p> <p>Cxf 编程</p> <p>Cxf+spring 整合(重点)</p> <p>CXF rest 实现(重点)</p> <p>便民查询网站开发</p>	<p>掌握 jax-ws 注解规范 webservice 接口方法</p> <p>掌握 cxf 编写 webservice 服务端、客户端</p> <p>掌握 Cxf 和 spring 整合使用方法</p> <p>掌握 CXF 发布 rest 服务</p>

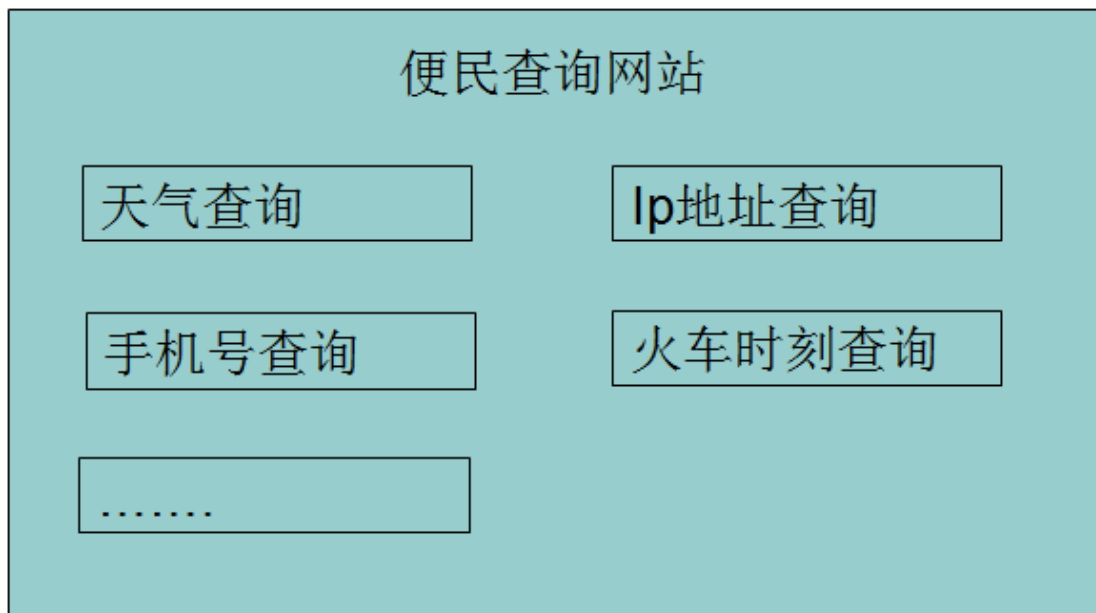
## 2 什么是 Webservice?

### 2.1 【客户服务器模式】

#### 2.1.1 便民查询网站分析

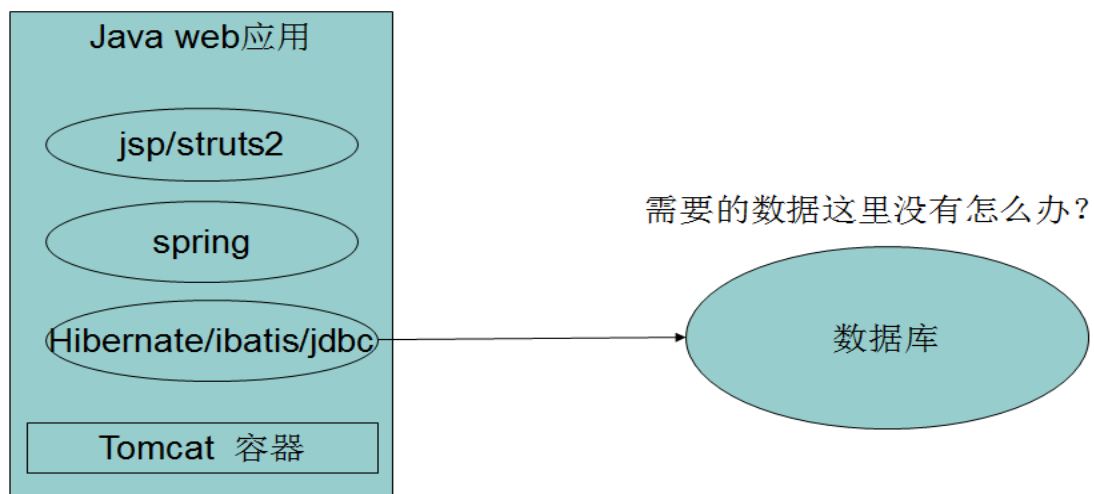
为了方便广大网民日常查询需求，通过便民查询网站可以查询手机号、ip 地址、天气等信息。

如何实现一个便民查询网站？



#### ■ 实现方案 1:

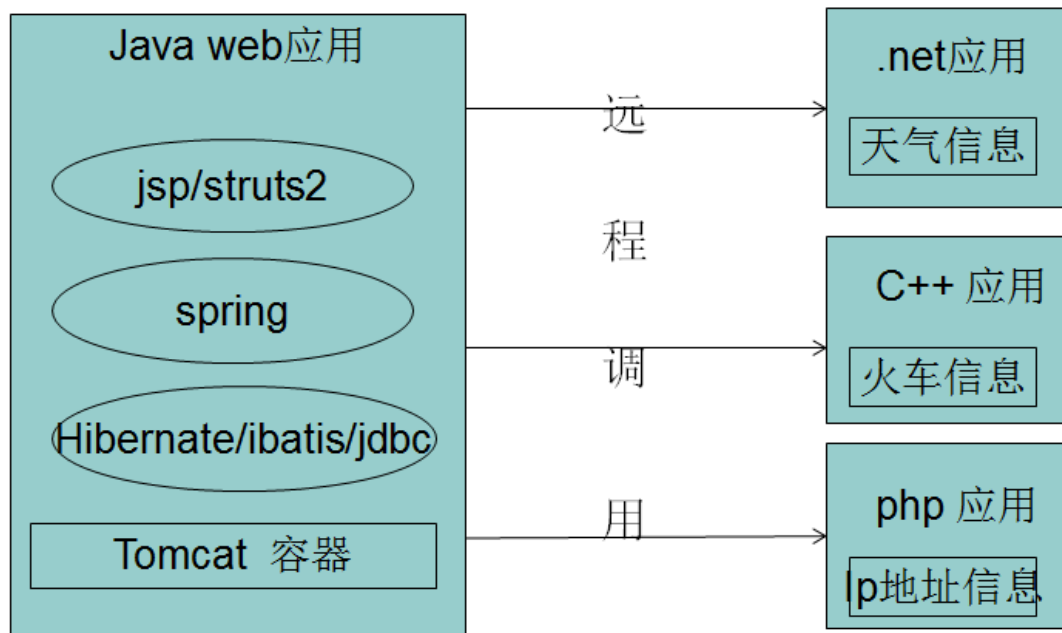
通常我们写的 javaweb 应用从数据库查询数据，如下图：



实时天气信息在我们的系统数据库中是不存在的所以此方案不可行。

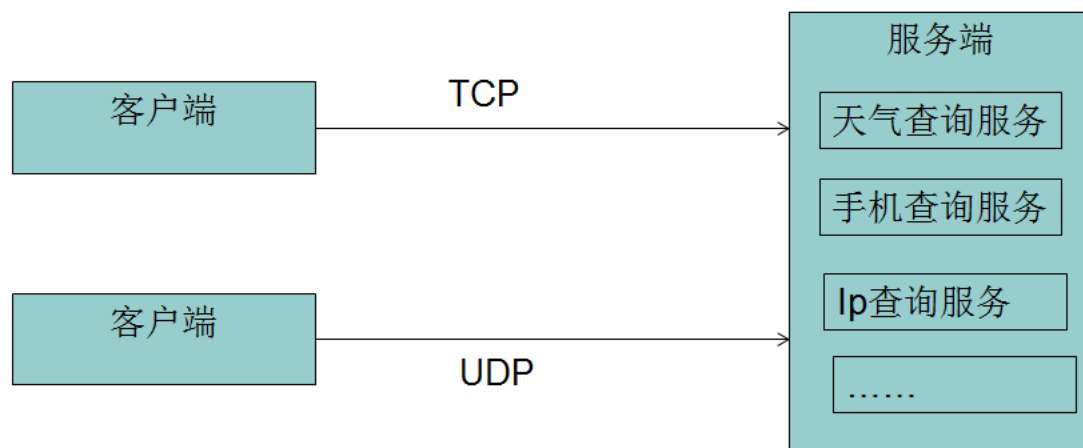
#### ■ 实现方案 2:

既然天气、火车等信息不在本系统的数据库中，可以设想能不能使用一种技术通过网络调用拥有天气、火车等信息的系统去获取这些信息呢？如下图：



### 2.1.2 客户服务器模式

对上图进行抽象，将调用方称为客户端，将被调方称为服务端，客户端通过网络通信协议访问服务端提供的接口，即如下图：



客户端程序和服务端程序可以是部署在同一个计算机上的两个程序，但通常客户端和服务端是部署在不同的计算机上的。

客户端和服务端之间的网络必须连通，客户端和服务端之间使用网络协议进行通信。

服务端监听一个端口，客户端通过服务端监听的端口进行请求，客户端通过 ip+端口去调用服务端，比如：tomcat 启动起来监听 8080 端口即 tomcat 对外提供服务，浏览器即客户端通过 http 协议访问 tomcat 的 8080 端口。

客户端通过网络通信协议访问服务端，网络协议包括 TCP 和 UDP 两大通信协议：

TCP 是一种面向连接的协议，提供可靠的数据传输，一般服务质量要求比较高的情况，使用这个协议。TCP 支持的应用协议主要有：Telnet、FTP、SMTP、HTTP 等；

UDP---用户数据报协议，是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。UDP 支持的应用层协议主要有：NFS(网络文件系统)、SNMP（简单网络管理协议）、DNS（主域名称系统）、TFTP（通用文件传输协议）等。

客户服务器模式早期主要应用于 c/s 应用，web 兴起后主要应用于 b/s 应用，b/s 比 c/s 的好处就在于 b/s 是基于浏览器客户端访问服务端。

## 2.2 【什么是 webservice】

### 2.2.1 什么是 webservice？

- Web service 即 web 服务，它是一种跨编程语言和跨操作系统平台的远程调用技术即跨平台远程调用技术。
- 采用标准 SOAP(Simple Object Access Protocol) 协议传输，soap 属于 w3c 标准。Soap 协议是基于 http 的应用层协议，soap 协议传输是 xml 数据。
- 采用 wsdl 作为描述语言即 webservice 使用说明书，wsdl 属 w3c 标准。
- xml 是 webservice 的跨平台的基础，XML 主要的优点在于它既与平台无关，又与厂商无关。
- XSD，W3C 为 webservice 制定了一套传输数据类型，使用 xml 进行描述，即 XSD(XML Schema Datatypes)，任何编程语言写的 webservice 接口在发送数据时都要转换成 webservice 标准的 XSD 发送。
- 当前非 SOAP 协议的 webService 以轻量为首要目标，比如 http rest 方式也是

webservice 的一种方式，或者直接使用 http 自定义数据协议，比如 http 传输 json 数据，http 传输 xml 数据等。

## 2.2.2 webservice 三要素

### 2.2.2.1 soap

SOAP 即简单对象访问协议(Simple Object Access Protocol) 是一种简单的基于 XML 的协议，它使应用程序通过 HTTP 来交换信息，简单理解为 soap=http+xml。Soap 协议版本主要使用 soap1.1、soap1.2。

SOAP 不是 webservice 的专有协议，其他应用协议也使用 soap 传输数据。例如，SMTP、tr069 等。

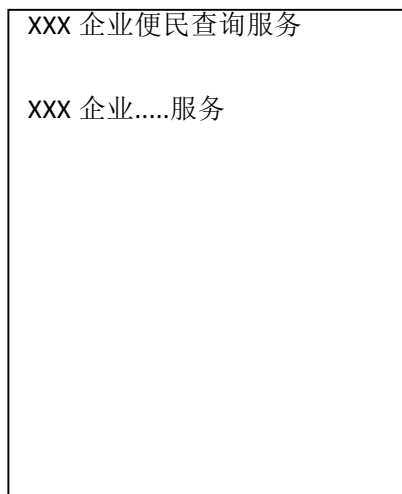
### 2.2.2.2 wsdl

WSDL 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 Wsdl 是 webservice 的使用说明书。

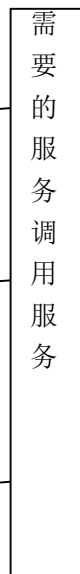
### 2.2.2.3 UDDI

UDDI 是一种目录服务，通过它，企业可注册并搜索 Web services。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 webservcie 资源进行共享，促进全球经济合作。

XXX 企业新商品查询服务



企  
业  
查  
找



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**。

## 2.2.3 webservice 开发规范

JAVA 中共有三种 WebService 规范，分别是 JAX-WS (JAX-RPC)、JAXM&SAAJ、JAX-RS。

下面来分别简要的介绍一下这三个规范。

### 2.2.3.1 JAX-WS 介绍

JAX-WS 的全称为 Java API for XML-Based Webservices，早期的基于 SOAP 的 JAVA 的 Web 服务规范 JAX-RPC (Java API For XML-Remote Procedure Call) 目前已经被 JAX-WS 规范取代。从 java5 开始支持 JAX-WS2.0 版本，Jdk1.6.0\_13 以后的版本支持 2.1 版本，jdk1.7 支持 2.2 版本。

### 2.2.3.2 JAXM&SAAJ

JAXM (JAVA API For XML Message) 主要定义了包含了发送和接收消息所需的 API, SAAJ (SOAP With Attachment API For Java, JSR 67) 是与 JAXM 搭配使用的 API, 为构建 SOAP 包和解析 SOAP 包提供了重要的支持, 支持附件传输等, JAXM&SAAJ 与 JAX-WS 都是基于 SOAP 的 Web 服务, 相比之下 JAXM&SAAJ 暴露了 SOAP 更多的底层细节, 编码比较麻烦, 而 JAX-WS 更加抽象, 隐藏了更多的细节, 更加面向对象, 实现起来你基本上不需要关心 SOAP 的任何细节

### 2.2.3.3 JAX-RS

JAX-RS 是 JAVA 针对 REST(Representation State Transfer)风格制定的一套 Web 服务规范, 由于推出的较晚, 该规范 (JSR 311, 目前 JAX-RS 的版本为 1.0) 并未随 JDK1.6 一起发行。

## 2.3 【Jax-ws 第一个例子】

### 2.3.1 第一步：服务端开发

1. 编写 SEI(Service Endpoint Interface), SEI 在 webservice 中称为 portType, 在 java 中称为接口。

代码如下:

```
/**
 * 天气查询服务接口
 * @author 传智播客 Java学院
 * @version V1.0
 */
public interface WeatherInterface {
    //天气查询
    public String queryWeather(String cityName);
}
```



```
}
```

## 2. 编写 SEI 实现类，此类作为 webservice 提供服务类

代码如下：

```
/**
 * 天气查询服务接口实现类
 * @author 传智播客 Java学院
 * @version V1.0
 */
@WebService
public class WeatherInterfaceImpl implements WeatherInterface {

    @Override
    public String queryWeather(String cityName) {
        System.out.println("from client.."+cityName);
        String result = "晴朗";
        System.out.println("to client..." + result);
        return result;
    }

    public static void main(String[] args) {
        //发送webservice服务
        Endpoint.publish("http://192.168.1.100:1234/weather", new
WeatherInterfaceImpl());
    }
}
```

注意：

SEI 实现类中至少要有有一个非静态的公开方法需要作为 webservice 服务方法。

public class 上边要加上@WebService

## 3. endpoint 发布服务

```
//发送webservice服务
Endpoint.publish("http://192.168.1.100:1234/weather", new
```

```
WeatherInterfaceImpl());
```

### 2.3.2 第二步：查看 wsdl

Webservice 发布成功，通过 wsdl 查看 webservice 发布的正确性

1. 在地址栏输入(注意后面的参数?wsdl)

<http://192.168.1.100:1234/weather?wsdl>

2. Wsdl 不是 webService,只是获取一个用于描述 WebService 的说明文件
3. wsdl- WebServiceDescriptionLanguage,是以 XML 文件形式来描述 WebService 的“说明书”,有了说明书,我们才可以知道如何使用或是调用这个服务.

### 2.3.3 第三步：Wsimport 生成客户端调用类

#### 2.3.3.1 Wsimport 介绍

wsimport 是 jdk 自带的 webservice 客户端工具,可以根据 wsdl 文档生成客户端调用代码(java 代码).当然,无论服务器端的 WebService 是用什么语言写的,都可以生成调用 webservice 的客户端代码,服务端通过客户端代码调用 webservice。

wsimport.exe 位于 JAVA\_HOME\bin 目录下.

常用参数为:

-d<目录> - 将生成.class 文件。默认参数。

-s<目录> - 将生成.java 文件。

-p<生成的新包名> -将生成的类,放于指定的包下。

(wsdlurl) - http://server:port/service?wsdl, 必须的参数。

示例:

```
C:/> wsimport -s . http://127.0.0.1:1234/weather?wsdl
```

注意：-s 不能分开，-s 后面有个小点

### 2.3.3.2 客户端生成注意事项：

1. 可以通过 `java -version` 检查你当前的版本号保存是 `jdk1.6` 以上。`Jdk1.6.0_13` 以后的版本支持 `jaxws2.1`。
2. 在 Eclipse 中创建一个空的 java 工程为 `wsimport`，此工程作为存放客户端代码。
3. cmd 命令行进入此 `wsimport` 工程的 `src` 目录，输入以下命令：

```
\wsgenerator\src>wsimport -s . http://192.168.1.100:1234/weather?wsdl
```

参数说明：-s 是指编译出源代码文件,后面的.(点)指将代码放到当前目录下。

最后面的 `http...` 是指获取 `wsdl` 说明书的地址。

4. 生成完成，刷新 Eclipse 中 `wsimport` 工程，将 `src` 下生成 `.java` 文件代码 Copy 到 `webservice` 客户端工程(见下)中。

### 2.3.4 第四步：客户端编写

代码如下：

```
/**
 * 天气查询客户端
 * @author 传智播客 Java学院
 * @version V1.0
 */
public class WeatherClient {
    public static void main(String[] args) {
```

```
//创建服务视图
WeatherInterfaceImplService weatherInterfaceImplService =new
WeatherInterfaceImplService();
//通过服务视图得到服务端点
WeatherInterfaceImpl weatherInterfaceImpl=
weatherInterfaceImplService.getPort(WeatherInterfaceImpl.class);
//调用webservice服务方法
String result = weatherInterfaceImpl.queryWeather("郑州");
System.out.println(result);
}
}
```

### 2.3.5 webservice 优点

- 1、采用 xml 支持跨平台远程调用。
- 2、基于 http 的 soap 协议，可跨越防火墙。
- 3、支持面向对象开发。
- 4、有利于软件和数据重用，实现松耦合。

### 2.3.6 webservice 缺点

由于 soap 是基于 xml 传输,本身使用 xml 传输会传输一些无关的东西从而效率不高,随着 soap 协议的完善,soap 协议增加了许多内容,这样就导致了使用 soap 协议进行数据传输的效率不高。

## 2.4 【webService 应用场景】

### 2.4.1 宏观

用于软件集成和复用

## 2.4.2 微观

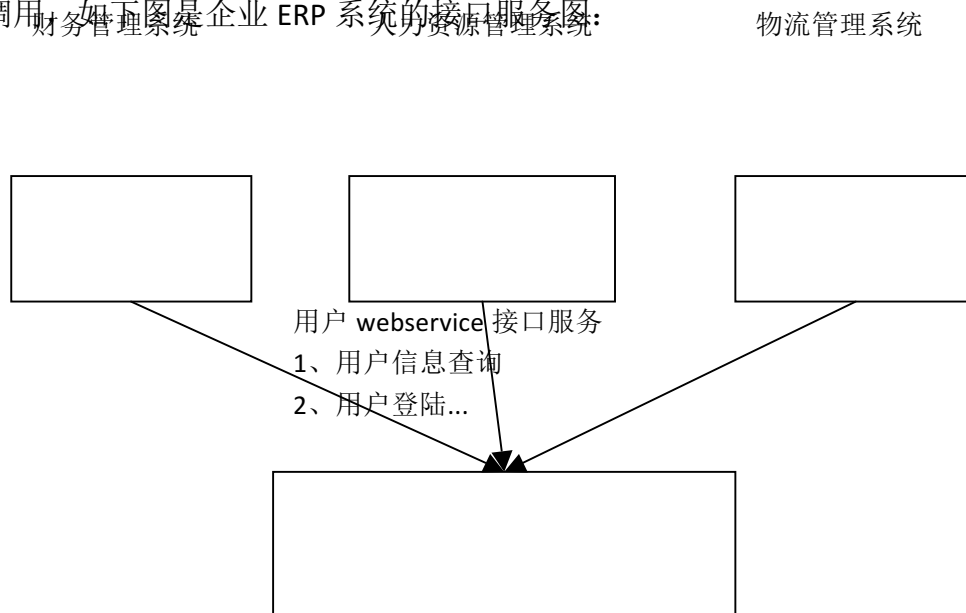
适用：

- 用于接口服务，不考虑客户端类型，不考虑性能，建议使用
  - 用于公开接口服务

面向互联网公开的接口，例如：某公司产品促销介绍、股票信息查询等，因为 webservice 使用的 soap 协议是一个标准协议，其它公司使用标准协议通信，方便系统开发和维护。比如：便民网站的天气查询接口、火车时刻查询接口等。

- 用于内部接口服务

一个大的系统平台是由若干个系统组成，系统与系统之间存在数据访问需求，为了减少系统与系统之间的耦合性可以将接口抽取出来提供单独的接口服务供它系统调用。下图是企业 ERP 系统的接口服务图：



- 服务端已经确定使用 webservice，客户端无法选择，只能使用 webservice

不适用：

- 对性能要求很高的应用，不建议使用 webservice  
比如银行交易系统、股票交易系统等，任何延迟都可能造成无法估量的损失。
- 同构程序之间通信不建议使用 webservice  
比如 Java 的 RMI 同样可以实现远程调用，而且性能比 webservice 好很多。

## 3 【WSDL】

### 3.1 WSDL 是什么？

WSDL 指网络服务描述语言(Web Services Description Language)。

WSDL 是一种使用 XML 编写的文档。这种文档可描述某个 Web service。它可规定服务的位置，以及此服务提供的操作（或方法）。

WSDL 是一种 XML 文档

WSDL 用于描述网络服务

WSDL 也可用于定位网络服务

### 3.2 wsdl 说明书结构

<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> 服务端点，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(可有多个参数)

<types> 定义 web service 使用的全部数据类型

### 3.3 wsdl 说明书阅读方式

从下往上读

先找到服务视图，通过 binding 找到 portType，找到了 portType 就找到了我们要调用的 webservice 方法。

查看 wsdl 说明要从下往上看：

#### 1 服务视图(根)

```
-<service name="WeatherInterfaceImplService">
  -<port name="WeatherInterfaceImplPort" binding="tns:WeatherInterfaceImplPortBinding">
    <soap:address location="http://192.168.1.100:1234/weather"/>
  </port>
</service>

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

#### 2 通过服务视图，binding 到 porttype

```
-<portType name="WeatherInterfaceImpl">
  -<operation name="queryWeather">
    <input message="tns:queryWeather"/>
    <output message="tns:queryWeatherResponse"/>
  </operation>
</portType>
```

#### 3、调用 portType 的方法

## 4 【wsdl 应用--公网天气查询】

### 4.1 wsdl 使用方法

第一步：阅读 wsdl，搞清楚服务视图 service、协议格式 binding 服务端点 porttype

第二步：创建服务视图

第三步：创建服务端点

第四步：通过服务端点调用服务方法

## 4.2 操作步骤

第一步：找到天气查询的 webservice WSDL 文件拷贝到 D 盘根目录

第二步：根据 WSDL 生成客户端调用类，使用 wsimport 生成

Wsimport -s <file:///d:/WeatherWebService.wsdl>

第三步：将生成的代码拷贝至工程中

第四步：编写客户端调用代码

## 4.3 客户端代码如下：

```
/**
 * 互联网天气查询客户端
 * @author 传智播客 Java学院
 * @version V1.0
 */
public class WebWeatherClient {
    public static void main(String[] args) {
        //创建服务视图
        WeatherWebService weatherWebService = new WeatherWebService();
        //通过服务视图得到服务端点
        WeatherWebServiceSoap weatherWebServiceSoap
=weatherWebService.getWeatherWebServiceSoap();
        //调用webservice服务方法
        ArrayOfString arrayOfString =
weatherWebServiceSoap.getWeatherbyCityName("郑州");
        List<String> resultlist = arrayOfString.getString();
        //查询天气查询结果
        for(String result:resultlist){
            System.out.println(result);
        }
    }
}
```



## 4.4 【webservice 客户端编程-使用 service 类】

### 4.4.1 步骤

第一步：创建 URL，指定资源地址即 wsdl 地址

第二步：创建 QName，指定命名空间和视图名称

第三步：创建服务视图对象 service

第四步：从服务视图中得到服务端点即服务接口（这里需要服务接口类型，可使用 wsimport 生成后只留下 porttype）

第五步：通过服务端点调用服务方法

### 4.4.2 天气查询客户端代码

```
/**
 * 使用javax.xml.ws.Service调用webservice服务
 * @author 传智播客 Java学院
 * @version V1.0
 */
public class WeatherClient2 {
    public static void main(String[] args) throws MalformedURLException {
        //定义url，参数为wsdl地址
        URL url = new URL("http://192.168.1.100:1234/weather?wsdl");
        //定义qname，第一个参数是命名空间，第二个参数名称是wsdl里边的服务名
        QName qName = new QName("http://impl.sei.jaxws.ws.itcast.cn/",
            "WeatherInterfaceImplService");
        //创建服务视图
        Service service = Service.create(url, qName);
        //通过服务视图得到服务端点
        WeatherInterfaceImpl weatherInterfaceImpl
            =service.getPort(WeatherInterfaceImpl.class);
        //调用webservice
        System.out.println(weatherInterfaceImpl.queryWeather("郑州"));
    }
}
```

### 4.4.3 公网天气查询客户端代码：

```
/**
 * 使用javax.xml.ws.Service调用公网webservice服务
 * @author 传智播客 Java学院
 * @version V1.0
 */
public class WebWeatherClient2 {
    public static void main(String[] args) throws MalformedURLException {
        //定义url, 参数为wsdl地址
        URL url = new URL("file:/D:/WeatherWebService.wsdl");
        //定义qname, 第一个参数是命名空间, 第二个参数名称是wsdl里边的服务名
        QName qName = new QName("http://WebXml.com.cn/",
"WeatherWebService");
        //创建服务视图
        Service service = Service.create(url, qName);
        //通过服务视图得到服务端点
        WeatherWebServiceSoap weatherWebServiceSoap
=service.getPort(WeatherWebServiceSoap.class);
        //调用webservice
        ArrayOfString arrayOfString =
weatherWebServiceSoap.getWeatherbyCityName("郑州");
        List<String> resultlist = arrayOfString.getString();
        //查询天气查询结果
        for(String result:resultlist){
            System.out.println(result);
        }
    }
}
```

### 4.4.4 公网手机号归属地查询客户端代码：

```
/**
 * 使用service类调用公网手机号归属地查询服务
 */
public class MobileClient_Service {
    public static void main(String[] args) throws
MalformedURLException {
```

```
//wsdl的url
URL url = new
URL("http://webservice.webxml.com.cn/WebServices/MobileCodeWS.
asmx");
//定义servicename
QName serviceName = new QName("http://WebXml.com.cn/",
"MobileCodeWS");

//得到 服务视图
Service service = Service.create(url, serviceName);

//得到portType
MobileCodeWSSoap mobileCodeWSSoap =
service.getPort(MobileCodeWSSoap.class);
//设备portType的方法
String resultString =
mobileCodeWSSoap.getMobileCodeInfo("1863819", "");

//查询数据
System.out.println(resultString);

}
}
```

## 4.5 使用 Service 调用和 Wsimport 代码调用方式区别:

Wsimport 生成代码调用 webservice 无法指定 webservice 的地址，使用生成的服务视图类获取服务端点(portType)实例。

Service 调用 Webservice 可以指定 webservice 的地址，只需要服务端点的接口即可获取服务端点实例。

## 5 Soap

### 5.1 soap 是什么

SOAP 是一种网络通信协议

SOAP 即 Simple Object Access Protocol 简易对象访问协议

SOAP 用于跨平台应用程序之间的通信

SOAP 被设计用来通过因特网(http)进行通信

SOAP = HTTP+XML，其实就是通过 HTTP 发 xml 数据

SOAP 很简单并可扩展支持面向对象

SOAP 允许您跨越防火墙

SOAP 将被作为 W3C 标准来发展

### 5.2 使用 TCP/IP Monitor 监视 Soap 协议

使用 TCP/IP Monitor 可以监视 tcp/ip 协议的报文内容，由于 http 是基于 Tcp 的应用协议，而 webservice 是基于 http 实现，所以通过 tcp/ip monitor 可以监视 webservice 请求及响应的内容。

### 5.3 Soap1.1:

#### 5.3.1 客户端代码:

```
//定义url, 参数为wsdl地址
URL url = new URL("http://127.0.0.1:54321/weather?wsdl");
//定义qname, 第一个参数是命名空间, 第二个参数名称是wsdl里边的服务名
QName qName = new
QName("http://server.jaxws.webservice.itcast.cn/",
"WeatherInterfaceImplService");
```

```
//创建服务视图
Service service = Service.create(url, qName);
//通过服务视图得到服务端点
WeatherInterfaceImpl weatherInterfaceImpl
=service.getPort(WeatherInterfaceImpl.class);
//调用webservice
System.out.println(weatherInterfaceImpl.queryWeather("郑州"));
```

### 5.3.2 请求:

注意红色标注:

POST /weather HTTP/1.1

Accept: text/xml, multipart/related

Content-Type: text/xml; charset=utf-8

SOAPAction: "http://server.jaxws.ws.itcast.cn/WeatherServer/queryWeatherRequest"

User-Agent: JAX-WS RI 2.2.8 svn-revision#13980

Host: 127.0.0.1:4321

Connection: keep-alive

Content-Length: 232

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:queryWeather xmlns:ns2="http://server.jaxws.ws.itcast.cn/">
      <arg0>郑州</arg0>
    </ns2:queryWeather>
  </S:Body>
</S:Envelope>
```

### 5.3.3 响应:

注意红色标注:

HTTP/1.1 200 OK

Transfer-encoding: chunked

Content-type: text/xml; charset=utf-8

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:queryWeatherResponse xmlns:ns2="http://server.jaxws.ws.itcast.cn/">
```

```
<return>天气晴朗</return>
</ns2:queryWeatherResponse>
</S:Body>
</S:Envelope>
```

## 5.4 soap 协议体包含下列元素

**必需有** Envelope 元素，此元素将整个 XML 文档标识为一条 SOAP 消息

可选的 Header 元素，包含头部信息

**必需有** 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

## 5.5 soap 消息基本结构

```
<?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>
```

## 5.6 http 发送 soap 协议测试

webservice 使用 soap 协议传输数据，soap 是基于 http 的应用协议，可以使用 http 发送 soap 协议数据完成 webservice 的请求。

本例子解析响应的 xml 数据使用 dom4j。

```

* 通过http发送soap协议请求webservice
* @author 传智播客 Java学院
* @version V1.0
*/
public class HttpRequestSoap {

    public static void main(String[] args) throws IOException
    {
        //webservice地址
        String webservice_url =
"http://127.0.0.1:1234/weather";
        //发送的soap协议内容
        String soap_xml = soap_xml("郑州");
        System.out.println(soap_xml);
        //创建url
        URL url = new URL(webservice_url);
        //创建http链接对象
        HttpURLConnection httpURLConnection =
(HttpURLConnection)url.openConnection();
        //设置请求方法
        httpURLConnection.setRequestMethod("POST");
        //设置Content-type
        httpURLConnection.setRequestProperty("Content-type",
"text/xml; charset=utf-8");
        //使用http进行输出
        httpURLConnection.setDoOutput(true);
        //使用http进行输入
        httpURLConnection.setDoInput(true);

        //通过输出流发送数据
        OutputStream outputStream =
httpURLConnection.getOutputStream();
        outputStream.write(soap_xml.getBytes());
        outputStream.close();

        //接收服务端响应数据
        InputStream inputStream =
httpURLConnection.getInputStream();

        //使用buffer存在读取的数据
        byte[] buffer = new byte[1024];

        //使用字节输出流存储读取的数据
        ByteArrayOutputStream byteArrayOutputStream = new

```

```

ByteArrayOutputStream();
    while(true){
        int len = inputStream.read(buffer);
        //如果流水读取完则退出循环
        if(len == -1){
            break;
        }
        byteArrayOutputStream.write(buffer,0,len);
    }

    //得到响应数据
    String response_string =
byteArrayOutputStream.toString();

    System.out.println(response_string);

    parseXml(response_string);
}
//soap协议内容
public static String soap_xml(String cityName){
    String soap_xml = "<?xml version=\"1.0\"
encoding=\"utf-8\"?>"
        + "<S:Envelope
xmlns:S=\"http://schemas.xmlsoap.org/soap/envelope/\">"
        + "<S:Body>"
        + "<ns2:queryWeather
xmlns:ns2=\"http://impl.sei.jaxws.ws.itcast.cn/\">"
        + "<arg0>" + cityName + "</arg0>"
        + "</ns2:queryWeather>"
        + "</S:Body>"
        + "</S:Envelope>";

    return soap_xml;
}

//解析响应的xml
public static String parseXml(String xmlString){
    String result = null;

    try {
        Document document =
DocumentHelper.parseText(xmlString);
        //创建xpath解析对象
    
```



```
DefaultXPath defaultXPath = new
DefaultXPath("//ns2:queryWeatherResponse");
//指定命名空间

defaultXPath.setNamespaceURIs(Collections.singletonMap("ns2",
"http:// impl.sei.jaxws.ws.itcast.cn/"));

List<Element> elements=
defaultXPath.selectNodes(document);

Element response = elements.get(0);

List<Element> results = response.selectNodes("return");

System.out.println(results.get(0).getText());

} catch (DocumentException e) {
    e.printStackTrace();
}

return result;
}
}
```

## 5.7 Soap1.2:

### 5.7.1 下载 jaxws-ri-2.2.8

Jaxws 实现 soap1.2 需要加入 jaxws 扩展包，从 sun 下载 jaxws-ri-2.2.8，解压 jaxws-ri-2.2.8 并将 lib 下的 jar 包加载到 java 工程中。

### 5.7.2 添加 BindingType

在 SEI 实现类上添加如下注解

@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP\_BINDING)

### 5.7.3 请求:

注意红色标注:

POST /weather HTTP/1.1

Accept: application/soap+xml, multipart/related

Content-Type: application/soap+xml;  
charset=utf-8;action="http://server.jaxws.ws.itcast.cn/WeatherServer/queryWeather  
Request"

User-Agent: JAX-WS RI 2.2.8 svn-revision#13980

Host: 127.0.0.1:4321

Connection: keep-alive

Content-Length: 230

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
<S:Body>
<ns2:queryWeather xmlns:ns2="http://server.jaxws.ws.itcast.cn/">
<arg0>郑州</arg0>
</ns2:queryWeather>
</S:Body>
</S:Envelope>
```

### 5.7.4 响应:

注意红色标注:

HTTP/1.1 200 OK

Transfer-encoding: chunked

Content-type: application/soap+xml; charset=utf-8

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
<S:Body>
<ns2:queryWeatherResponse xmlns:ns2="http://server.jaxws.ws.itcast.cn/">
<return>天气晴朗</return>
</ns2:queryWeatherResponse>
</S:Body>
</S:Envelope>
```

## 5.8 Soap1.1 与 soap1.2 异同

相同之处：

soap1.1 和 soap1.2 都是使用 post 方法  
都包括 Envelope 和 body

内容类型 context-type 不同：

soap1.1 使用 text/xml  
soap1.2 使用 application/soap+xml

命名空间 Envelope xmlns 不同：

soap1.1 使用 <http://schemas.xmlsoap.org/soap/envelope/>  
soap1.2 使用 <http://www.w3.org/2003/05/soap-envelope>

## 6 jax-ws 开发深入

### 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

endpointInterface：SEI 接口地址，如果一个服务类实现了多个接口，只需要发布一个接口的方法，可通过此注解指定要发布服务的接口。

@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:

```
/**
 * 天气查询服务接口实现类
 * @author 传智播客 Java学院
 * @version V1.0
 */
@WebService(targetNamespace="http://webservice.itcast.cn",
serviceName="weatherService",
portName="weatherServicePort",
name="weatherServiceInterface"
)
public class WeatherInterfaceImpl implements WeatherInterface {

    @WebMethod(operationName="queryWeather")
    public @WebResult(name="weatherResult")String queryWeather(
        @WebParam(name="cityName")String cityName) {
        System.out.println("from client.."+cityName);
        String result = "晴朗";
        System.out.println("to client..." + result);
        return result;
    }

    public static void main(String[] args) {
        //发送webservice服务
        Endpoint.publish("http://192.168.1.100:1234/weather", new
WeatherInterfaceImpl());
    }
}
```

```
}  
  
}
```

### 6.1.3 使用注解注意

@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 方法是不可以使用 @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。

## 6.2 使用复杂数据类型发布 webservice

### 6.2.1 需求

SOAP 协议支持对象格式数据，我们可以将天气查询的结果封装在一个查询结果对象中，字段包括：城市、日期、天气、温度等信息。

服务端编写 webservice 的方法和前边一样，最后使用 endpoint 发布服务。  
wsimport 生成客户端代码。

### 6.2.2 代码：

#### 6.2.2.1 服务端：

```
@WebService(targetNamespace="http://webservice.itcast.cn",  
serviceName="weatherService",  
portName="weatherServicePort",  
name="weatherServiceInterface"  
)  
public class WeatherInterfaceImpl implements WeatherInterface {  
  
    @Override
```

```
public @WebResult(name="weatherResult")List<WeatherModel>
queryWeather(
@WebParam(name="cityName")String cityName) throws Exception {

    //构造三天的天气结果
    Calendar calendar = Calendar.getInstance();
    int day = calendar.get(Calendar.DATE);
    //第一天的天气
    WeatherModel weatherModel_1 = new WeatherModel();
    weatherModel_1.setDate(new Date());
    weatherModel_1.setDetail("晴朗");
    weatherModel_1.setTemperature_max(30);
    weatherModel_1.setTemperature_min(23);
    //第二天的天气
    WeatherModel weatherModel_2 = new WeatherModel();
    calendar.set(Calendar.DATE, day+1);
    weatherModel_2.setDate(calendar.getTime());
    weatherModel_2.setDetail("晴转多云");
    weatherModel_2.setTemperature_max(28);
    weatherModel_2.setTemperature_min(21);
    //第三天的天气
    WeatherModel weatherModel_3 = new WeatherModel();
    calendar.set(Calendar.DATE, day+2);
    weatherModel_3.setDate(calendar.getTime());
    weatherModel_3.setDetail("多云转小雨");
    weatherModel_3.setTemperature_max(25);
    weatherModel_3.setTemperature_min(18);

    List<WeatherModel> list = new ArrayList<WeatherModel>();
    list.add(weatherModel_1);
    list.add(weatherModel_2);
    list.add(weatherModel_3);
    //返回三天的天气
    return list;
}
}
```

#### 6.2.2.2 客户端：

```
public class WeatherClient {
```

```
public static void main(String[] args) throws
MalformedURLException, Exception_Exception {
    //wsdl的url
    URL url = new URL("http://127.0.0.1:12345/weather?wsdl");
    //serviceName
    //第一参数是命名空间地址，第二个参数是service名称
    QName serviceName = new
QName("http://server.jaxws.webservice.itcast.cn/",
"WeatherInterfaceImplService");
    //创建服务视图
    Service service = Service.create(url, serviceName);
    //通过服务视图得到portType

    WeatherInterfaceImpl weatherInterfaceImpl =
service.getPort(WeatherInterfaceImpl.class);
    //调用porttype方法
    List<WeatherModel> list =
weatherInterfaceImpl.queryWeather("北京");

    for(WeatherModel weatherModel:list){
        Date date =
weatherModel.getDate().toGregorianCalendar().getTime();

        System.out.println(new
SimpleDateFormat("yyyy-MM-dd").format(date));
    }
}
```

## 7 CXF 基础

### 7.1 什么是 CXF

Apache CXF = Celtix + Xfire, 开始叫 Apache CeltiXfire, 后来更名为 Apache CXF 了, 以下简称为 CXF。Apache CXF 是一个开源的 web Services 框架, CXF 帮助您构建和开发 web Services, 它支持多种协议, 比如: SOAP1.1,1,2、XML/HTTP、RESTful

HTTP 或者 CORBA。

CORBA（Common Object Request Broker Architecture 公共对象请求代理体系结构，早期语言使用的 WS。C,c++,C#）

Cxf 是基于 SOA 总线结构，依靠 spring 完成模块的集成，实现 SOA 方式。

灵活的部署：可以运行有 Tomcat,Jboss,Jetty(内置),weblogic 上面。

## 7.2 CXF 的安装及配置

从官网下载 2.7.11

### 2.7.11

The 2.7.11 release is our latest release. For a complete list of new features, API changes, etc... please see the [release notes](#) and the [migration guide](#).

Description	File	MD5	SHA1	PGP
Source distribution	<a href="#">apache-cxf-2.7.11-src.tar.gz</a>	<a href="#">apache-cxf-2.7.11-src.tar.gz.md5</a>	<a href="#">apache-cxf-2.7.11-src.tar.gz.sha1</a>	<a href="#">apache-cxf-2.7.11-src.tar.gz.asc</a>
	<a href="#">apache-cxf-2.7.11-src.zip</a>	<a href="#">apache-cxf-2.7.11-src.zip.md5</a>	<a href="#">apache-cxf-2.7.11-src.zip.sha1</a>	<a href="#">apache-cxf-2.7.11-src.zip.asc</a>
Binary distribution	<a href="#">apache-cxf-2.7.11.tar.gz</a>	<a href="#">apache-cxf-2.7.11.tar.gz.md5</a>	<a href="#">apache-cxf-2.7.11.tar.gz.sha1</a>	<a href="#">apache-cxf-2.7.11.tar.gz.asc</a>
	<a href="#">apache-cxf-2.7.11.zip</a>	<a href="#">apache-cxf-2.7.11.zip.md5</a>	<a href="#">apache-cxf-2.7.11.zip.sha1</a>	<a href="#">apache-cxf-2.7.11.zip.asc</a>

## 7.3 环境变量配置：

JAVA\_HOME,

CXF\_HOME=cxf 的目录

Path = %JAVA\_HOME%\bin;%CXF\_HOME%\bin;

CLASSPATH=.;%CXF\_HOME%\lib\cxf-manifest.jar



## 7.4 CXF 例子

### 7.4.1 第一步：创建 java 工程

### 7.4.2 第二步：将 cxf 的 jar 包加入工程

### 7.4.3 第三步：创建服务接口和服务实现类

创建服务接口和服务类的方法同上边章节描述，编写 SEI 及 SEI 的实现。

注意：与 jaxws 编程不同的是将 `@WebService` 注解加在接口上边。

#### 7.4.3.1 服务接口：

使用 cxf 开发 webservice 这里只需要在接口上加 `@webservice` 注解即可，和 jaxws 开发不同。

```
@WebService
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public interface WeatherInterface {
    //根据城市名称查询未来三天的天气
    public List<WeatherModel> queryWeather(String cityName)
    throws Exception;
}
```

#### 7.4.3.2 服务接口实现类：

使用 cxf 开发不用在接口实现类上加 `@webservice` 注解，因为 cxf 发布服务时可以指定接口。

```
public class WeatherInterfaceImpl implements WeatherInterface {

    @Override
    public List<WeatherModel> queryWeather(String cityName)
    throws Exception {
```

//构造三天的天气结果

```
Calendar calendar = Calendar.getInstance();
int day = calendar.get(Calendar.DATE);

WeatherModel weatherModel_1 = new WeatherModel();
weatherModel_1.setDate(new Date());
weatherModel_1.setDetail("晴朗");
weatherModel_1.setTemperature_max(30);
weatherModel_1.setTemperature_min(23);

WeatherModel weatherModel_2 = new WeatherModel();
calendar.set(Calendar.DATE, day+1);
weatherModel_2.setDate(calendar.getTime());
weatherModel_2.setDetail("晴转多云");
weatherModel_2.setTemperature_max(28);
weatherModel_2.setTemperature_min(21);

WeatherModel weatherModel_3 = new WeatherModel();
calendar.set(Calendar.DATE, day+2);
weatherModel_3.setDate(calendar.getTime());
weatherModel_3.setDetail("多云转小雨");
weatherModel_3.setTemperature_max(25);
weatherModel_3.setTemperature_min(18);

List<WeatherModel> list = new ArrayList<WeatherModel>();
list.add(weatherModel_1);
list.add(weatherModel_2);
list.add(weatherModel_3);

return list;
}
}
```

#### 7.4.3.3 发布服务类:

```
package cn.itcast.ws.jaxws.server;

import org.apache.cxf.jaxws.JaxWsServerFactoryBean;

/**
 * CXF发布jaxws服务类
 */
```

```
* @author Thinkpad
*
*/
public class Server {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //创建服务工厂bean
        JaxWsServerFactoryBean jaxWsServerFactoryBean = new
JaxWsServerFactoryBean();
        //指定服务接口

        jaxWsServerFactoryBean.setServiceClass(WeatherServerInterfa
ce.class);

        //指定服务实现对象

        jaxWsServerFactoryBean.setServiceBean(new
WeatherInterfaceImpl());
        //指定webservice的地址

        jaxWsServerFactoryBean.setAddress("http://192.168.1.100:123
4/weather");
        //创建webservice服务
        jaxWsServerFactoryBean.create();
    }
}
```

#### 7.4.4 第四步：根据 wsdl 地址生成客户端代码

我们分别使用 `wsimport` 和 `wsdl2java` 生成客户端代码，都可以正常使用。  
**\*\*wsdl2java 可以生成 soap1.1 和 soap1.2**

#### 7.4.4.1 wsdl2java 生成客户代码

先让我们了解一下 cxf 的 wsdl2java 工具，它的功能就如同 wsimport 一样，可以生成一堆客户端调用的代码。

在命令行执行：

```
wsdl2java -d . http://192.168.1.100:1234/weather?wsdl
```

注意：

生成后 WeatherService 报错：

原因是 cxf 需要 JAX-WS API 2.2 而 jdk6 的 jax-ws 是 2.1 版本，需要 wsdl2java 使用 “-frontend jaxws21 “

即如下：

```
wsdl2java -d . -frontend jaxws21 http://localhost:1234/weather?wsdl
```

#### 7.4.5 第五步：编写客户端：

##### 7.4.5.1 方式 1、使用 javax.xml.ws.Service 调用客户端

```
package cn.itcast.ws.jaxws.client;

import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import cn.itcast.ws.jaxws.server.WeatherServerInterface;

public class Client3 {

    public static void main(String[] args) throws
    MalformedURLException {

        //创建URL,资源定位
        URL url = new
        URL("http://192.168.1.100:1234/weather?wsdl");
```

```
//Qname,确定命名空间地址, 和服务视图名称
QName qName = new QName("http://service.itcast.cn/",
"WeatherInterfaceService");
//创建service
Service service = Service.create(url, qName);
//创建porttype(服务端点)
WeatherInterface weatherInterface =
service.getPort(WeatherInterface.class);

//通过服务端点调用服务方法
weatherServerInterface.queryWather("郑州");
}
}
```

#### 7.4.5.2 方式 2、JaxwsProxyFactoryBean 调用服务端:

```
//创建代码工厂bean
JaxWsProxyFactoryBean jaxWsProxyFactoryBean = new
JaxWsProxyFactoryBean();
//设置接口类型(portType)
jaxWsProxyFactoryBean.setServiceClass(WeatherInterface.class);
//设置webservice地址
jaxWsProxyFactoryBean.setAddress("http://192.168.1.100:1234/weather"
);
//创建portType调用实例
WeatherInterface weatherInterface =(WeatherInterface)
jaxWsProxyFactoryBean.create();
//调用webservice
weatherServerInterface.queryWather("郑州");
```

### 7.4.6 SOAP1.2 生成

在服务接口的上面都添加

```
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
```

如下：

```
@WebService
```

```
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
```

```
public interface WeatherServerInterface
```

```
@WebService(endpointInterface =
```

```
"cn.itcast.ws.jaxws.server.WeatherServerInterface")
```

```
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
```

```
public class WeatherServer implements WeatherServerInterface
```

## 8 Cxf 与 spring 集成

### 8.1.1 第一步建立一个 web 项目

### 8.1.2 第二步填充 CXF JAR 包。

### 8.1.3 第三步创建服务接口及服务类

编写方法同上面章节描述。

### 8.1.3.1 服务接口：

```
@WebService(targetNamespace="http://service.itcast.cn/",
name="WeatherInterface", //porttype的名称
portName="WeatherInterfacePort",
serviceName="WeatherInterfaceService"
)
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public interface WeatherInterface {
    //根据城市名称查询公网天气服务接口
    public List<String> queryWeather(String cityName) throws
Exception;
}
```

### 8.1.3.2 服务实现类：

需求：服务类需要调用公网天气查询客户端。

```
public class WeatherInterfaceImpl implements WeatherInterface {

    //此为公网查询客户端需要通过spring配置并注入
    private WeatherWebServiceSoap weatherWebServiceSoap;

    @Override
    public List<String> queryWeather(String cityName) throws
Exception {

        ArrayOfString arrayOfString =
weatherWebServiceSoap.getWeatherbyCityName("郑州");
        List<String> results = arrayOfString.getString();
        for(String result:results){
            System.out.println(result);
        }

        return results;
    }

    public WeatherWebServiceSoap getWeatherWebServiceSoap() {
        return weatherWebServiceSoap;
    }
}
```

```
}  
  
    public void setWeatherWebServiceSoap(WeatherWebServiceSoap  
weatherWebServiceSoap) {  
        this.weatherWebServiceSoap = weatherWebServiceSoap;  
    }  
  
}
```

### 8.1.3.3 公网天气查询代码生成

将上边章节生成的公网天气查询客户端调用代码考入本工程。  
生成方法在此不再赘述。

### 8.1.4 第四步创建 applicationContext.xml

将 applicationContext.xml 放在 WEB-INF 下

内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:jaxws="http://cxf.apache.org/jaxws"  
    xmlns:jaxrs="http://cxf.apache.org/jaxrs"  
    xmlns:cxf="http://cxf.apache.org/core"  
    xsi:schemaLocation="http://www.springframework.org/schema/b  
eans  
  
http://www.springframework.org/schema/beans/spring-beans.xsd  
http://cxf.apache.org/jaxrs  
http://cxf.apache.org/schemas/jaxrs.xsd  
http://cxf.apache.org/jaxws  
http://cxf.apache.org/schemas/jaxws.xsd  
http://cxf.apache.org/core  
http://cxf.apache.org/schemas/core.xsd">  
<!-- 配置发布webservice服务 -->  
<jaxws:server address="/weather"  
serviceClass="cn.itcast.webservice.jaxws.server.WeatherInterfa  
ce">  
    <jaxws:serviceBean>  
        <ref bean="weatherInterface"/>  

```



```
</jaxws:serviceBean>
</jaxws:server>

<!-- 公网天气查询客户端 -->
<jaxws:client id="weatherClient"
serviceClass="cn.com.webxml.WeatherWebServiceSoap"
address="http://www.webxml.com.cn/WebServices/WeatherWebService.asmx" />

<!-- 本系统对外服务的天气查询service -->
<bean id="weatherInterface"
class="cn.itcast.webservice.jaxws.server.WeatherInterfaceImpl"
>
    <property name="weatherWebServiceSoap" ref="weatherClient" />
</bean>

</beans>
```

### 8.1.5 第五步在 web.xml 配置 Spring 环境

```
<context-param>
    <param-name>contextConfigLocation</param-name>

    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>
org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

### 8.1.6 第六步在 web.xml 配置 CXF 的 servlet

```
<servlet>
    <servlet-name>cx<u>f</u></servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
```

```
<servlet-name>cxfr</servlet-name>  
<url-pattern>/ws/*</url-pattern>  
</servlet-mapping>
```

### 8.1.7 第七步：启动 web 容器

访问：<http://192.168.1.100:8080/.../ws/weather?wsdl>

### 8.1.8 第八步：编写客户端工程

客户端创建同前边章节，此处不再赘述。

## 9 CXF 发布 rest 服务

### 9.1 什么是 rest 服务

REST 是一种软件架构模式，只是一种风格，rest 服务采用 HTTP 做传输协议，REST 对于 HTTP 的利用实现精确的资源定位。

Rest 要求对资源定位更加准确，如下：

非 rest 方式：<http://ip:port/queryUser.action?userType=student&id=001>

Rest 方式：<http://ip:port/user/student/query/001>

Rest 方式表示互联网上的资源更加准确，但是也有缺点，可能目录的层级较多不容易理解。

REST 是一种软件架构理念，现在被移植到 Web 服务上，那么在开发 Web 服务上，偏于面向资源的服务适用于 REST，REST 简单易用，效率高，SOAP 成熟度较高，安全性较好。

注意：REST 不等于 Webservice，JAX-RS 只是将 REST 设计风格应用到 Web 服务开发上。

## 9.2 发布 rest 服务

### 9.2.1 需求：

发布查询学生信息的服务，以 json 和 xml 数据格式返回。

### 9.2.2 pojo

```
@XmlElement(name="student")
public class Student {

    private long id;
    private String name;
    private Date birthday;
```

### 9.2.3 SEI

```
@WebService
@Path("/student")
public interface StudentService {

    @GET
    @Produces(MediaType.APPLICATION_XML)
    @Path("/query/{id}")
    public Student queryStudent(@PathParam("id") long id) throws
Exception;

    @GET
    @Produces({"application/json;charset=utf-8", MediaType.APPLI
CATION_XML})
    @Path("/querylist/{id}")
    public List<Student> queryStudentList(@PathParam("id") long
id) throws Exception;
}
```

上边代码中：

queryStudent 方法以 xml 格式发布

queryStudentList 方法以 json 和 xml 两种格式发布。

## 9.2.4 SEI 实现类

```
public class StudentServiceImpl implements StudentService {

    @Override
    public Student queryStudent(long id) throws Exception {
        Student student = new Student();
        student.setId(1000001);
        student.setName("张三");
        student.setBirthday(new Date());
        return student;
    }

    @Override
    public List<Student> queryStudentList(long id) throws
Exception {
        Student student1 = new Student();
        student1.setId(1000001);
        student1.setName("李四");
        student1.setBirthday(new Date());

        Student student2 = new Student();
        student2.setId(1000001);
        student2.setName("张三");
        student2.setBirthday(new Date());

        List<Student> list = new ArrayList<Student>();
        list.add(student1);
        list.add(student2);
        return list;
    }
}
```

## 9.2.5 程序代码发布

```
public class RestServer {
```

```
public static void main(String[] args) {

    JAXRSServerFactoryBean jaxrsServerFactoryBean = new
JAXRSServerFactoryBean();
    //rest地址

    jaxrsServerFactoryBean.setAddress("http://127.0.0.1:12345/r
est");
    //设置SEI实现类

    jaxrsServerFactoryBean.setResourceClasses(StudentServiceImp
l.class);
    jaxrsServerFactoryBean.create();
}
}
```

## 9.2.6 Spring 配置文件发布

```
<!-- rest服务发布 -->
<jaxrs:server address="/rest">
<jaxrs:serviceBeans>
    <ref bean="studentService"/>
</jaxrs:serviceBeans>

</jaxrs:server>
<!-- 学生查询，rest方式 -->
<bean id="studentService"
class="cn.itcast.ws.cxf.rest.server.StudentServiceImpl">
</bean>
```

## 9.2.7 测试

queryStudent 方法测试:

http://127.0.0.1:8080/工程名/ws/rest/student/query/1

```
-<student>
  <birthday>2014-12-29T21:53:46.179+08:00</birthday>
  <id>100000</id>
  <name>张三</name>
</student>
```

queryStudentList 方法测试:

返回 json

http://127.0.0.1:8080/工程名/ws/rest/student/querylist/1?\_type=json

```
{"student":[{"birthday":"2014-12-29T21:55:13.577+08:00","id":100000,"name":"李四"}, {"birthday":"2014-12-29T21:55
```

返回 xml

http://127.0.0.1:8080/工程名/ws/rest/student/querylist/1?\_type=xml

```
-<students>
  -<student>
    <birthday>2014-12-29T21:57:03.974+08:00</birthday>
    <id>100000</id>
    <name>李四</name>
  </student>
  -<student>
    <birthday>2014-12-29T21:57:03.974+08:00</birthday>
    <id>100000</id>
    <name>张三</name>
  </student>
</students>
```

## 10 便民网站

### 10.1 需求:

使用 springmvc+CXF 实现便民网站，同时对外提供 webservice 服务。

调用公网的 webservice，  
将自己的服务发布成 webservice。

## 10.2 步骤

### 10.2.1 第一步：创建 web 项目工程















### 10.2.2 第二步：导入 jar 包

#### 10.2.2.1 导入 cxf 的全部 jar 包

参考 cxf 下的 lib 目录

#### 10.2.2.2 更换 cxf 中原始 spring 的 jar 包

Cxf2.7.11 默认和 spring3.0.7 整合，这里我们使用 spring3.1.4，将 cxf 中的 spring 开头的 jar 去掉，拷贝 spring 及 springmvc 的所有 jar 包。

-  commons-logging-1.1.1.jar
-  jstl-1.2.jar
-  spring-aop-3.1.4.RELEASE.jar
-  spring-asm-3.1.4.RELEASE.jar
-  spring-batch-infrastructure-2.2.2.RELEASE.jar
-  spring-beans-3.1.4.RELEASE.jar
-  spring-context-3.1.4.RELEASE.jar
-  spring-context-support-3.1.4.RELEASE.jar
-  spring-core-3.1.4.RELEASE.jar
-  spring-expression-3.1.4.RELEASE.jar
-  spring-jdbc-3.1.4.RELEASE.jar
-  spring-retry-1.0.2.RELEASE.jar
-  spring-test-3.1.4.RELEASE.jar
-  spring-tx-3.1.4.RELEASE.jar
-  spring-web-3.1.4.RELEASE.jar
-  spring-webmvc-3.1.4.RELEASE.jar

## 10.2.3 第三步：天气查询服务接口开发

### 10.2.3.1 服务接口

```
@WebService(targetNamespace="http://service.itcast.cn/",
name="WeatherInterface", //porttype的名称
portName="WeatherInterfacePort",
serviceName="WeatherInterfaceService"
)
public interface WeatherService {
    //根据城市名称查询三天的天气
    public List<WeatherModel> queryWeather(String cityName)
    throws Exception;
}
```

### 10.2.3.2 服务实现

```
public class WeatherServiceImpl implements WeatherService {

    @Resource
    private WeatherWebServiceSoap weatherWebServiceSoap;

    @Override
    public List<WeatherModel> queryWeather(String cityName)
    throws Exception {
        //调用dao获取天气信息
        ArrayOfString arrayOfString =
        weatherWebServiceSoap.getWeatherbyCityName(cityName);
        List<String> list = arrayOfString.getString();
        for(String result:list){
            System.out.println(result);
        }

        //解析获取的天气信息
        WeatherModel weatherModel_1 = new WeatherModel();
        weatherModel_1.setDetail(list.get(6)); //天气概况
        weatherModel_1.setImg(list.get(8)); //图标
    }
}
```



```
WeatherModel weatherModel_2 = new WeatherModel();
weatherModel_2.setDetail(list.get(13)); //天气概况
weatherModel_2.setImg(list.get(15)); //图标

WeatherModel weatherModel_3 = new WeatherModel();
weatherModel_3.setDetail(list.get(18)); //天气概况
weatherModel_3.setImg(list.get(20)); //图标
//返回的结果集
List<WeatherModel> list_1 = new ArrayList<WeatherModel>();

list_1.add(weatherModel_1);
list_1.add(weatherModel_2);
list_1.add(weatherModel_3);

return list_1;
}

}
```

### 10.2.3.3 公网查询客户端代码

将上边章节生成的公网天气查询客户端调用代码考入本工程。  
生成方法在此不再赘述。

## 10.2.4 第四步：spring 环境

将上边编写的天气查询服务接口在 spring 环境配置。

Classpath 下添加 applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xmlns:jaxrs="http://cxf.apache.org/jaxrs"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd"
```

```
http://cxf.apache.org/jaxrs
http://cxf.apache.org/schemas/jaxrs.xsd">

<!-- 导入资源文件 -->
<bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="Locations">
        <list>
            <value>classpath:config.properties</value>
        </list>
    </property>
</bean>

<!-- 配置发布webservice服务 -->
<jaxws:server address="/weather"
    serviceClass="cn.itcast.weather.service.WeatherService">
    <jaxws:serviceBean>
        <ref bean="weatherService" />
    </jaxws:serviceBean>
</jaxws:server>

<!-- 天气查询的bean -->
<bean id="weatherService"
class="cn.itcast.weather.service.impl.WeatherServiceImpl">
</bean>

<!-- 公网天气查询客户端 -->
<jaxws:client id="weatherWebServiceSoap"
serviceClass="cn.com.webxml.WeatherWebServiceSoap"
    address="${webservice_url_weather}" />

</beans>
```

### 10.2.5 第五步：系统参数配置文件

```
config.properties
//公网天气查询地址
webservice_url_weather=http://www.webxml.com.cn/WebServices/WeatherWebService.asmx
```

## 10.2.6 第六步：控制层开发

实现用户进入天气查询页面，输入城市查询天气。

控制层调用天气查询服务接口。

```
package cn.itcast.ws.cxf.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.context.ApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

import cn.itcast.ws.cxf.mobile.service.MobileService;

public class MobileServlet extends HttpServlet {

    private MobileService mobileService;

    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException,
        ServletException{
        String code = request.getParameter("code");
        ApplicationContext context =
            WebApplicationContextUtils.getWebApplicationContext(this.getServletContext());
        mobileService = (MobileService)
            context.getBean("mobileService");
        if(null != code && !"".equals(code)){
            String result = mobileService.queryMobile(code);
            request.setAttribute("result", result);
        }

        request.getRequestDispatcher("WEB-INF/jsp/queryMobile.jsp")
            .forward(request, response);
    }
}
```

```

    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException,
        ServletException{
        String code = request.getParameter("code");
        ApplicationContext context =
        WebApplicationContextUtils.getWebApplicationContext(this.getSe
        rvletContext());
        mobileService = (MobileService)
        context.getBean("mobileService");
        if(null != code && !"".equals(code)){
            String result = mobileService.queryMobile(code);
            request.setAttribute("result", result);
        }

        request.getRequestDispatcher("WEB-INF/jsp/queryMobile.jsp")
        .forward(request, response);
    }
}

```

## 10.2.7 第七步：web.xml

Web.xml 中做以下配置：

启动系统加载 spring 环境

springmvc 的 servlet

CXF 的 servlet

```

<!-- spring配置文件 -->
<context-param>
    <param-name>contextConfigLocation</param-name>

    <param-value>/WEB-INF/classes/applicationContext.xml</param-value>
</context-param>
<!-- 启动加载spring -->
<listener>

    <listener-class>org.springframework.web.context.ContextLoaderListene

```

```

r</listener-class>
</listener>
<!-- CXF的servlet -->
<servlet>
    <description>Apache CXF Endpoint</description>
    <display-name>cx</display-name>
    <servlet-name>cx</servlet-name>

    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>cx</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
<!-- mobileServlet配置 -->
<servlet>
    <servlet-name>mobileServlet</servlet-name>
    <servlet-class>
cn.itcast.ws.cxf.servlet.MobileServlet</servlet-class>

</servlet>
<servlet-mapping>
    <servlet-name> mobileServlet</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>

<!-- post提交乱码处理 -->

<filter>
    <filter-name>CharacterEncodingFilter</filter-name>

    <filter-class>org.springframework.web.filter.CharacterEncodingFilter
</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

