

# 教学计划

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

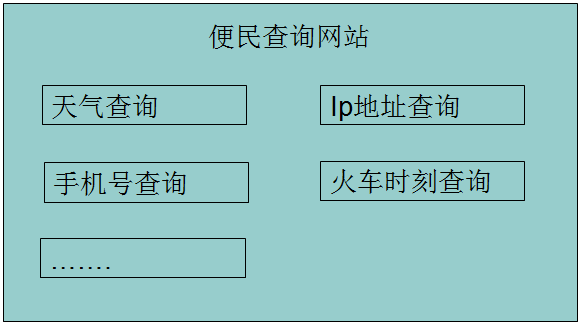
# 什么是Webservice？

## 【客户服务器模式 】

### 便民查询网站分析

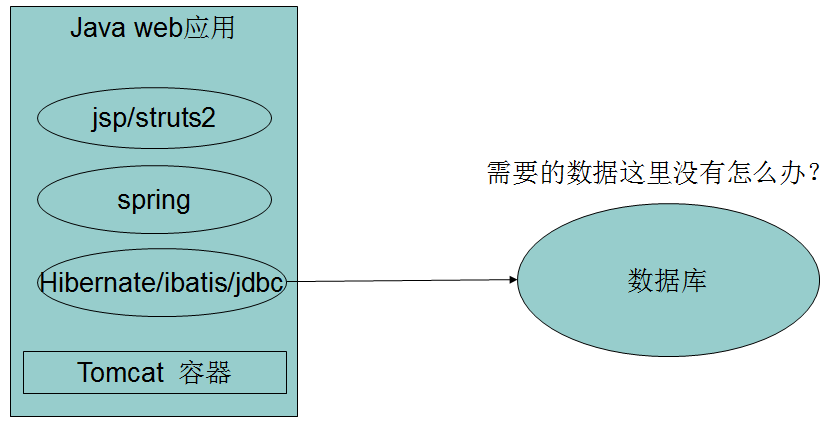
为了方便广大网民日常查询需求，通过便民查询网站可以查询手机号、ip地址、天气等信息。

如何实现一个便民查询网站？



* 实现方案1：

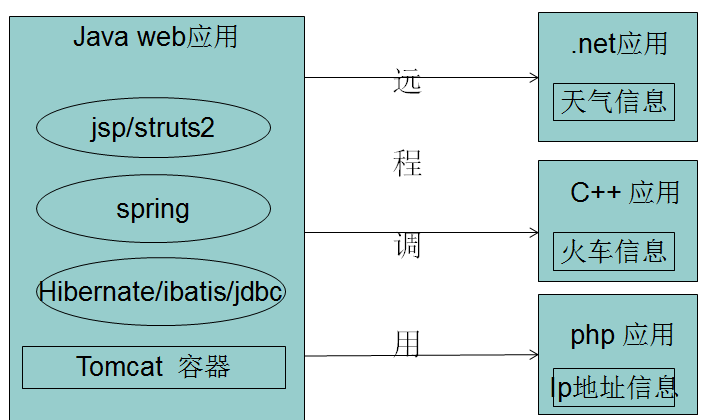
通常我们写的javaweb应用从数据库查询数据，如下图：



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。

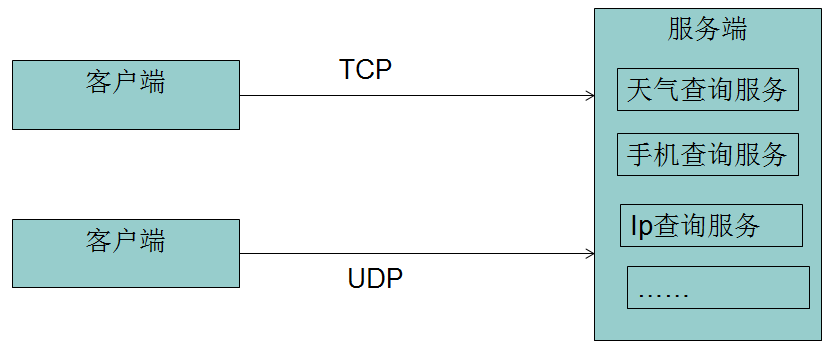
* 实现方案2：

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：



### 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。

## 【什么是webservice】

### 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。

### webService三要素

#### soap

SOAP即简单对象访问协议(Simple Object Access Protocal) 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。

#### 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。

#### 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。

UDDI目录服务

XXX企业新商品查询服务

XXX企业便民查询服务

XXX企业.....服务

企

业

查

找

需

要

的

服

务

调

用

服

务

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。

### 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。

#### 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。

#### 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 暴漏了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

#### 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。

## 【Jax-ws第一个例子】

### 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);

}

1. 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

1. endpoint发布服务

//发送webservice服务

Endpoint.*publish*("http://192.168.1.100:1234/weather", **new** WeatherInterfaceImpl());

### 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>

1. Wsdl不是webService,只是获取一个用于描述WebService的说明文件
2. 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.

### 第三步：Wsimport生成客户端调用类

#### 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

#### 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:



参数说明：-s是指编译出源代码文件,后面的.(点)指將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.

1. 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。

### 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);

}

}

### 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。

### 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。

## 【webService应用场景】

### 宏观

用于软件集成和复用

### 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，如下图是企业ERP系统的接口服务图：

财务管理系统

人力资源管理系统

物流管理系统

用户webservice接口服务

1、用户信息查询

2、用户登陆...

* 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。

# 【Wsdl】

## 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

## 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

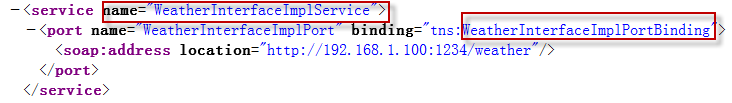
## wsdl 说明书阅读方式

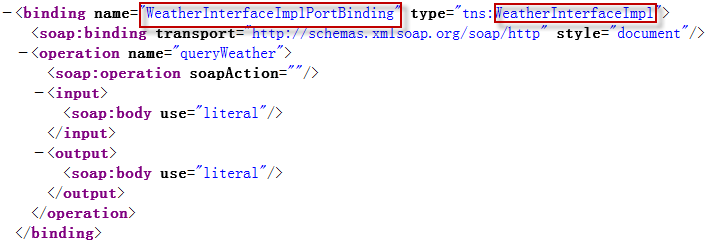
从下往上读

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

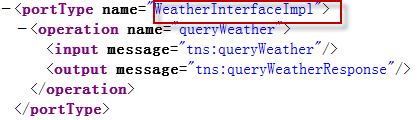
**查看wsdl说明要从下往上看：**

1服务视图(根)





2通过服务视图，bingding到porttype



3、调用portType的方法

# 【wsdl应用--公网天气查询】

## 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

## 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

## 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);

}

}

}

## 【webservice客户端编程-使用service类】

### 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

### 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"));

}

}

### 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);

}

}

}

### 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);

}

}

## 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(postType)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。

# Soap

## 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

## 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。

## Soap1.1：

### 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"));

### 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>

### 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>

## 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

## 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>

## 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;

}

}

## Soap1.2：

### 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。

### 添加BindingType

在SEI实现类上添加如下注解

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

### 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/queryWeatherRequest"

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>

### 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>

## 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>

# jax-ws开发深入

## JAX-WS注解

### 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，可以更加形像的描述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

### 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());

}

}

### 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。

## 使用复杂数据类型发布webservice

### 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。

### 代码：

#### 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;

}

}

#### 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));

}

}

}

# CXF基础

## 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://baike.baidu.com/view/63.htm)/[HTTP](http://baike.baidu.com/view/9472.htm)、[REST](http://baike.baidu.com/view/1077487.htm)ful HTTP 或者 [CORBA](http://baike.baidu.com/view/153815.htm)。

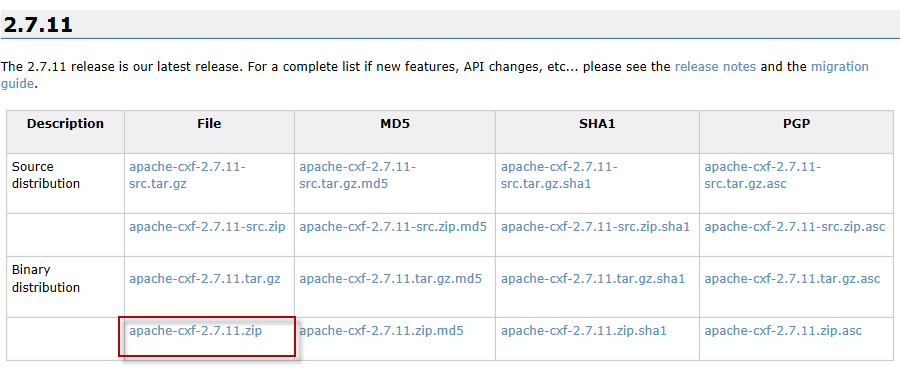
CORBA（Common Object Request Broker Architecture公共对象请求代理体系结构,早期语言使用的WS。C,c++,C#）

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。

## CXF的安装及配置

从官网下载2.7.11



## 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

## CXF例子

### 第一步：创建java工程

### 第二步：将cxf的jar 包加入工程

### 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。**

#### 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;

}

#### 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;

}

}

#### 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(WeatherServerInterface.**class**);

//指定服务实现对象

jaxWsServerFactoryBean.setServiceBean(**new** WeatherInterfaceImpl());

//指定webservice的地址

jaxWsServerFactoryBean.setAddress("http://192.168.1.100:1234/weather");

//创建webservice服务

jaxWsServerFactoryBean.create();

}

}

### 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

#### 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

### 第五步：编写客户端：

#### 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");

}

}

#### 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");

### 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

# Cxf与spring集成

### 第一步建立一个web项目

### 第二步填充CXF JAR包。

### 第三步创建服务接口及服务类

编写方法同上面章节描述。

#### 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;

}

#### 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;

}

}

#### 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。

### 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/beans*

*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.WeatherInterface"*>

<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>

### 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>

### 第六步在web.xml配置CXF的servlet

<servlet>

<servlet-name>cxf</servlet-name> <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>cxf</servlet-name>

<url-pattern>/ws/\*</url-pattern>

</servlet-mapping>

### 第七步：启动web容器

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

### 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。

# CXF 发布rest服务

## 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。

## 发布rest服务

### 需求：

发布查询学生信息的服务，以json和xml数据格式返回。

### pojo

@XmlRootElement(name="student")

**public** **class** Student {

**private** **long** id;

**private** String name;

**private** Date birthday;

### 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.*APPLICATION\_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。

### SEI实现类

**public** **class** StudentServiceImpl **implements** StudentService {

@Override

**public** Student queryStudent(**long** id) **throws** Exception {

Student student = **new** Student();

student.setId(100000l);

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(100000l);

student1.setName("李四");

student1.setBirthday(**new** Date());

Student student2 = **new** Student();

student2.setId(100000l);

student2.setName("张三");

student2.setBirthday(**new** Date());

List<Student> list = **new** ArrayList<Student>();

list.add(student1);

list.add(student2);

**return** list;

}

}

### 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/rest");

//设置SEI实现类

jaxrsServerFactoryBean.setResourceClasses(StudentServiceImpl.**class**);

jaxrsServerFactoryBean.create();

}

}

### 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 -->

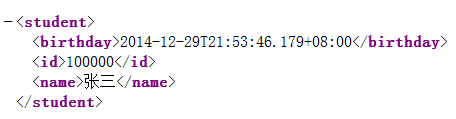
<bean id=*"studentService"* class=*"cn.itcast.ws.cxf.rest.server.StudentServiceImpl"*>

</bean>

### 测试

queryStudent方法测试：

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



queryStudentList方法测试：

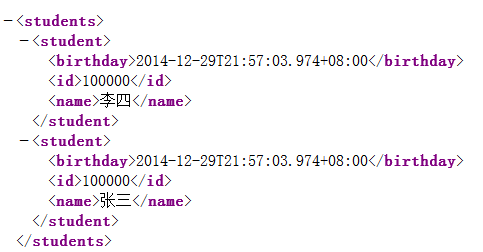
返回json

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



返回xml

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



# 便民网站

## 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。

## 步骤

### 第一步：创建web项目工程

### 第二步：导入jar包

#### 导入cxf的全部jar包

参考cxf下的lib目录

#### 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。



### 第三步：天气查询服务接口开发

#### 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;

}

#### 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\_l = **new** ArrayList<WeatherModel>();

list\_l.add(weatherModel\_1);

list\_l.add(weatherModel\_2);

list\_l.add(weatherModel\_3);

**return** list\_l;

}

}

#### 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。

### 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>

### 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

### 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**.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);  }  } |

### 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.ContextLoaderListener</listener-class>

</listener>

<!-- CXF的servlet -->

<servlet>

<description>Apache CXF Endpoint</description>

<display-name>cxf</display-name>

<servlet-name>cxf</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>cxf</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>