# 5WebService简介

Web Service也叫XML Web Service WebService是一种可以接收从Internet或者Intranet上的其它系统中传递过来的请求，轻量级的独立的通讯技术。是通过SOAP在Web上提供的软件服务，使用WSDL文件进行说明，并通过UDDI进行注册。WebService是一种跨编程语言和跨操作系统平台的远程调用技术。

还可以从多个角度来理解WebService，从表面看，WebService就是一个应用程序向外界暴露出一个能通过Web进行调用的API，也就是说能用编程的方法通过Web来调用这个应用程序。我们把调用这个WebService的应用程序叫做客户端，而把提供这个WebService的应用程序叫做服务端。从深层次看，WebService是建立可互操作的分布式应用程序的新平台，是一个平台，是一套标准。它定义了应用程序如何在Web上实现互操作性，你可以用任何你喜欢的语言，在任何你喜欢的平台上写Web service ，只要我们可以通过Web service标准对这些服务进行查询和访问。

# WebService三要素

SOAP、WSDL、UDDI（UniversalDescriptionDiscovery andIntegration）三者构成了WebService的三要素。下面详细阐述这三大技术：

**SOAP**

WebService通过HTTP协议发送请求和接收结果时，发送的请求内容和结果内容都采用XML格式封装，并增加了一些特定的HTTP消息头，以说明HTTP消息的内容格式，这些特定的HTTP消息头和XML内容格式就是SOAP协议。SOAP提供了标准的RPC(远程调用技术)方法来调用Web Service。

SOAP协议组成：

SOAP协议 = HTTP协议 + XML数据格式

SOAP协议定义了SOAP消息的格式，SOAP协议是基于HTTP协议的，SOAP也是基于XML和XSD的，XML是SOAP的数据编码方式。

**WSDL**

好比我们去商店买东西，首先要知道商店里有什么东西可买，然后再来购买，商家的做法就是张贴广告海报。 WebService也一样，WebService客户端要调用一个WebService服务，首先要有知道这个服务的地址在哪，以及这个服务里有什么方法可以调用，所以，WebService务器端首先要通过一个WSDL文件来说明自己家里有啥服务可以对外调用，服务是什么（服务中有哪些方法，方法接受的参数是什么，返回值是什么），服务的网络地址用哪个url地址表示，服务通过什么方式来调用。

**WSDL(**Web Services Description Language)就是这样一个基于XML的语言，用于描述Web Service及其函数、参数和返回值。它是WebService客户端和服务器端都能理解的标准格式。因为是基于XML的，所以WSDL既是机器可阅读的，又是人可阅读的，这将是一个很大的好处。一些最新的开发工具既能根据你的Web service生成WSDL文档，又能导入WSDL文档，生成调用相应WebService的代理类代码。

**UDDI**

uddi是一个跨产业，跨平台的开放性架构，可以帮助 Web 服务提供商在互联网上发布 Web 服务的信息。UDDI 是一种目录服务，企业可以通过 UDDI 来注册和搜索 Web 服务。简单来说，UDDI 就是一个目录，只不过在这个目录中存放的是一些关于 Web 服务的信息而已

# 为什么要使用WebService

1. 跨平台调用
2. 跨语言调用
3. 远程调用

# 发布WebService。

## 3.1服务端

### 3.1.1定义接口

|  |
| --- |
| /\*\*  \* 定义一个有关天气的服务  \* **@author** admin  \*  \*/  **public** **interface** IWeatherService {  **public** String queryWeather(String cityName);  } |

### 3.1.2定义实现类

|  |
| --- |
| @WebService // 用该注解修改表示当前类是一个服务类，只发布public修饰的方式  **public** **class** WeatherServiceImpl **implements** IWeatherService {  @Override  **public** String queryWeather(String cityName) {  System.***out***.println("WeatherServiceImpl.queryWeather() " + cityName);  **return** "大晴天";  }  } |

@WebMethod(exclude=true)方法上面添该注解表示不发布这个方法

### 3.1.3发布服务

|  |
| --- |
| **public** **class** WeatherService {  **public** **static** **void** main(String[] args) {  /\*\*  \* 参数1：服务地址  \* 参数2：服务类  \*/  Endpoint.*publish*("http://localhost:8080/weatherService", **new** WeatherServiceImpl());  System.***out***.println("服务发布成功");  }  } |

### 3.1.4 wsdl结构

|  |  |
| --- | --- |
| <service> | 服务视图，WebService的服务端点 |
| <binding> | Web Services的通信协议，还描述Web Services的方法、输入、输出。 |
| <portType> | 描述了WebService可执行的操作，通过binding指向portType |
| <message> | 描述了服务中发布的方法，包括参数，返回值等。 |
| <types> | 定义了WebService中使用的数据类型 |

## 3.2客户端

### 3.2.1生成客户端代码

|  |
| --- |
| 通过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)  第一种方式【远程地址】 wsimport -s . <http://127.0.0.1:8088/weather?wsdl>  第二种方式【xml文件】 wsimport -s . 文件路径(文件路径后面不需要加?wsdl) |

### 3.2.2 客户端调用第一种方式

|  |
| --- |
| // 1.创建服务视图(视图是从service标签的name属性获取)  WeatherServiceImplService weatherServiceImplService = **new** WeatherServiceImplService();    // 2.获取服务实现类(实现类从portType的name属性获取)  WeatherServiceImpl serviceImpl = weatherServiceImplService.getPort(WeatherServiceImpl.**class**);    // 3.调用方法(从portType的operation标签获取 )  String result = serviceImpl.queryWeather("深圳");    // 4.输出服务端返回的结果  System.***out***.println(result); |

### 3.2.3 客户端调用第二种方式(常用)

|  |
| --- |
| // 1.设置服务端地址  URL url = **new** URL("http://localhost:8080/weatherService?wsdl");    // 2.设置服务名称和命名空间  // 参数1:wsdl的命名空间(targetNamespace)  // 参数2:是服务视图的名称(service的name值)  QName qName = **new** QName("http://impl.service.qf.com/", "WeatherServiceImplService");    // 3.创建服务视图  Service service = Service.*create*(url, qName);    // 3.得到服务视图的实现类  WeatherServiceImpl weatherServiceImpl = service.getPort(WeatherServiceImpl.**class**);    // 4.调用方法  String result = weatherServiceImpl.queryWeather("青海");    // 5.输出服务端返回结果  System.***out***.println(result); |

### 3.2.4 客户端调用第三种方式

|  |
| --- |
| // 1.创建服务地址，不是WSDL地址  URL url = **new** URL("http://localhost:8080/weatherService");  // 2.打开一个通向服务地址的连接  HttpURLConnection connection = (HttpURLConnection) url.openConnection();  // 3.设置参数,  connection.setRequestMethod("POST"); // POST必须大写,否则抛出异常  // 这里是text/xml不是text/html  connection.setRequestProperty("content-Type", "text/xml;charset=utf-8");    // 4.设置输入输出，默认是false没有读写的权限  connection.setDoOutput(**true**);  connection.setDoInput(**true**);    // 5.组织SOAP数据，发送请求  String soapXml = getXml("青海");  connection.getOutputStream().write(soapXml.getBytes());  **if** (connection.getResponseCode() == 200) {  InputStream ips = connection.getInputStream();  Scanner scanner = **new** Scanner(ips);  StringBuffer buffer = **new** StringBuffer();  **while**(scanner.hasNextLine()){  buffer.append(scanner.nextLine());  }  scanner.close();  String result = parseXmlToString(buffer);  System.***out***.println("result:"+result);  }  **private** String parseXmlToString(StringBuffer buffer) {  **try** {  Document document = DocumentHelper.*parseText*(buffer.toString());  // "//"从任意位置的节点上选择名称为 item 的节点。  Node node = document.selectSingleNode("//return");  **return** node.getText();  } **catch** (DocumentException e) {  e.printStackTrace();  }  **return** **null**;  }  **private** String getXml(String string) {  String xml = "<?xml version=\"1.0\" ?>" + "<S:Envelope xmlns:S=\"http://schemas.xmlsoap.org/soap/envelope/\">"  + "<S:Body>" + "<ns2:queryWeather xmlns:ns2=\"http://impl.service.qf.com/\">" + "<arg0>" + string  + "</arg0>" + "</ns2:queryWeather>" + "</S:Body>" + "</S:Envelope>\"";  **return** xml;  } |

SelectNodes的用法

|  |
| --- |
| SelectNodes("item")  从当前节点的儿子节点中选择名称为 item 的节点。  SelectNodes("/item")  从根节点的儿子节点中选择名称为 item 的节点。  SelectNodes("//item")  从任意位置的节点上选择名称为 item 的节点。要重点突出这个任意位置，它不受当前节点的影响，也就是说假如当前节点是在第 100 层（有点夸张），也可以选择第一层的名称为 item 的节点。  SelectNodes(".")  选择当前节点。  SelectNodes("..")  选择当前节点的父节点。  SelectNodes("//item[@name]")  在 SelectNodes("//item") 的基础上，增加了一个限制，就是要求拥有 name 属性。  SelectNodes("//item[@name='111']")  在 SelectNodes("//item[@name]") 的基础上，增加了一个限制，就是要求 name 属性值为 111。注意语法中有引号；如果没有引号，则表示是数字类型，对于数字类型可以使用大于号、小于号等，比如：SelectNodes("//item[@v>333]")。  SelectNodes("//item[1]")  选择第一个 item，注意是第一个，不是第二个。  还有这里是指亲兄弟间的第一个 item，也就是说：父级若有三个 item，则选择第一个；若父级第二个 item 的有两个名称同样为 item 的儿子，则第一个儿子会被选择出来；若父级第三个 item 的也有两个名称同样为 item 的儿子，则第一个儿子也会被选择出来……  SelectNodes("//item[last()-1]")  倒数第二个节点，同样是指亲兄弟间的倒数第二个。  SelectNodes("//item[position()<=2]")  位置为第一和第二的节点（第一个节点的 position() 为 1），同样是指亲兄弟间的位置。  SelectNodes("//@name")  SelectNodes("/root/item/@name") 取 item 的 name 属性  选择 name 属性，注意这下选择的是属性，而不是节点了。用 Value 属性获取属性集合的属性值。  SelectNodes("/root/item")  根节点 root 下的 item 儿子节点。  SelectNodes("/root//item")  根节点 root 下的不管是儿子、孙子、重孙子……，只要是名称为 item 的统统选取出来。 |

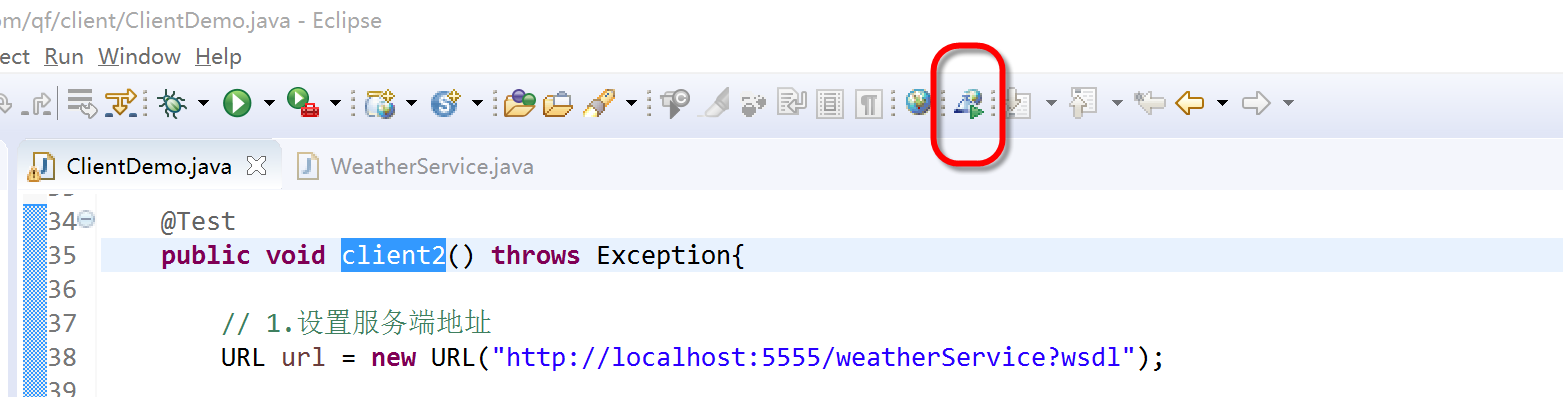
# 案例

地址：http://www.webxml.com.cn/zh\_cn/web\_services.aspx

5.1天气预报

5.2手机归属地

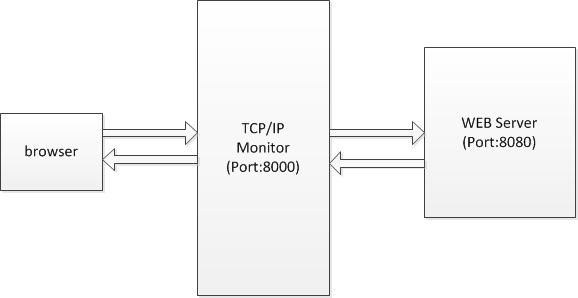
# 6.Eclipse提供的WebService客户端



# 7.监听工具TCP/IP Monitor

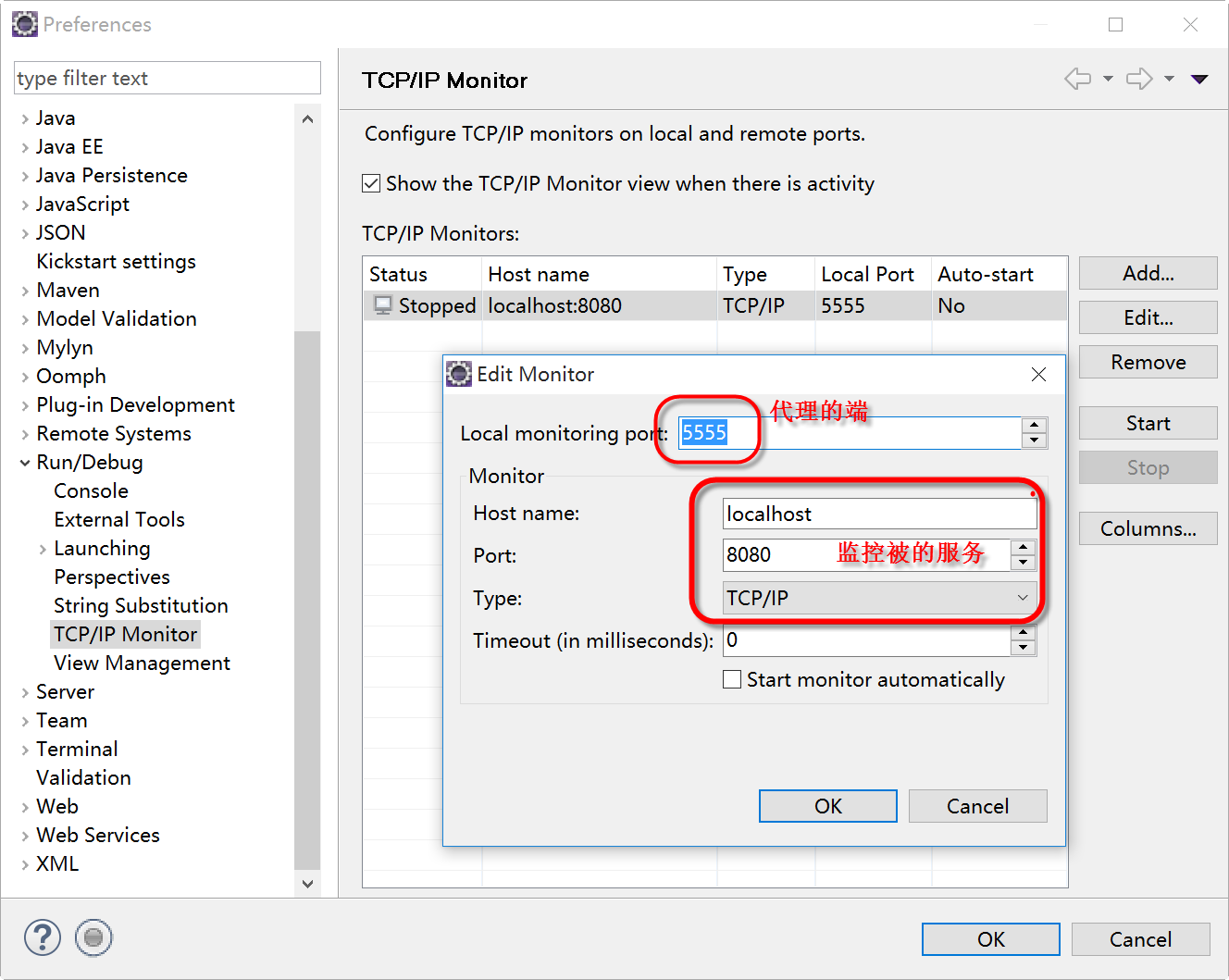
Eclipse中提供了一个很好的端口监听的工具TCP/IP Monitor，通过这个工具，我们可以监听目标端口输入输出的数据。

事实上，这个TCP/IP Monitor相当于一个“ 拦截器”，所有我们向目标服务器发送的数据包和服务器返回的数据包都要经过这个拦截器（也叫作代理服务器），进而拦截到数据包。

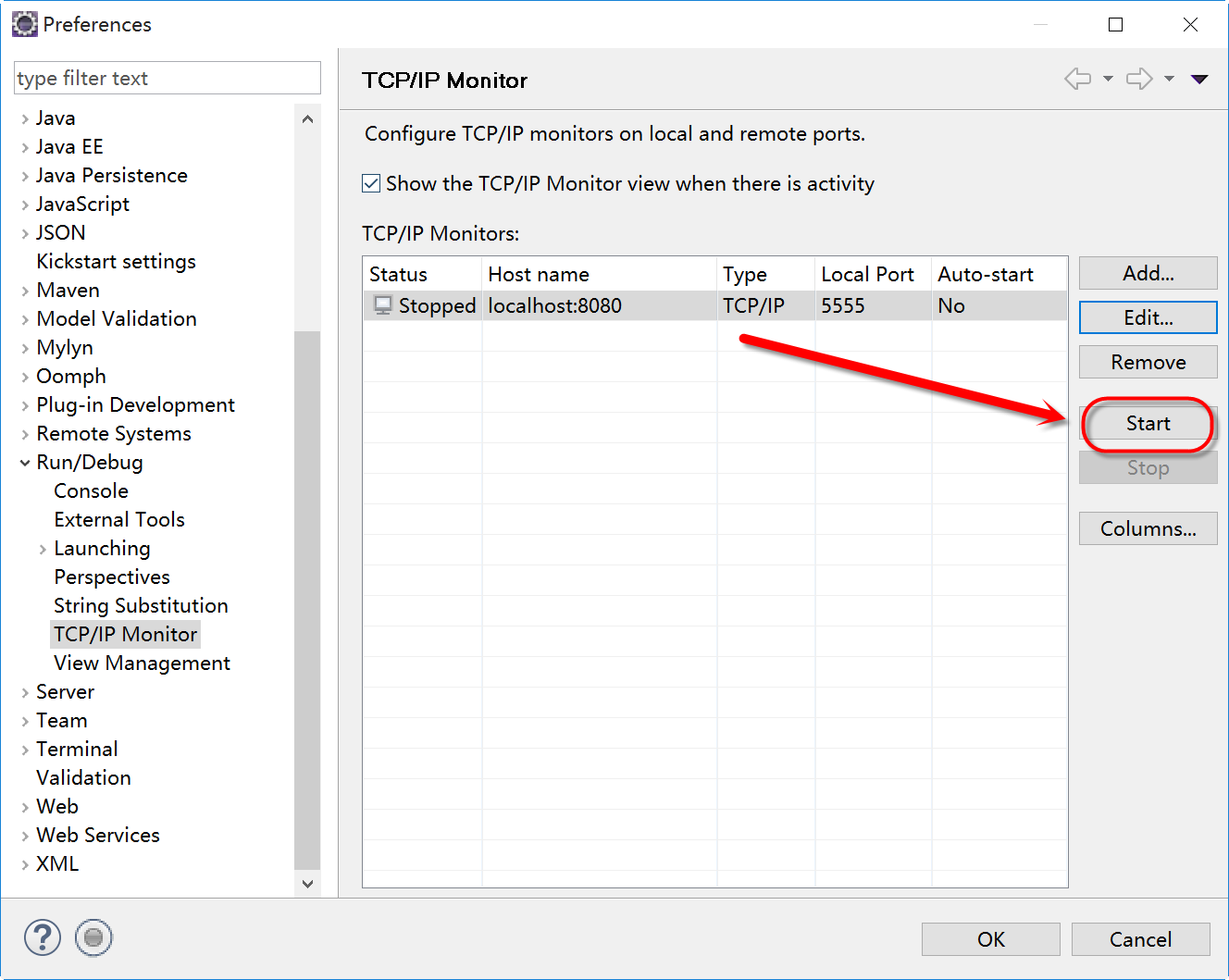


## 7.1配置TCP/IP Monitor

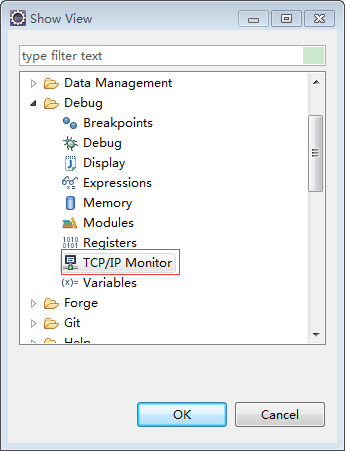
我们点击eclipse菜单栏的windows->Preferences->Run/Debug->TCP/IP Mointor,然后，我们点击Add按钮，添加一个监听端口。



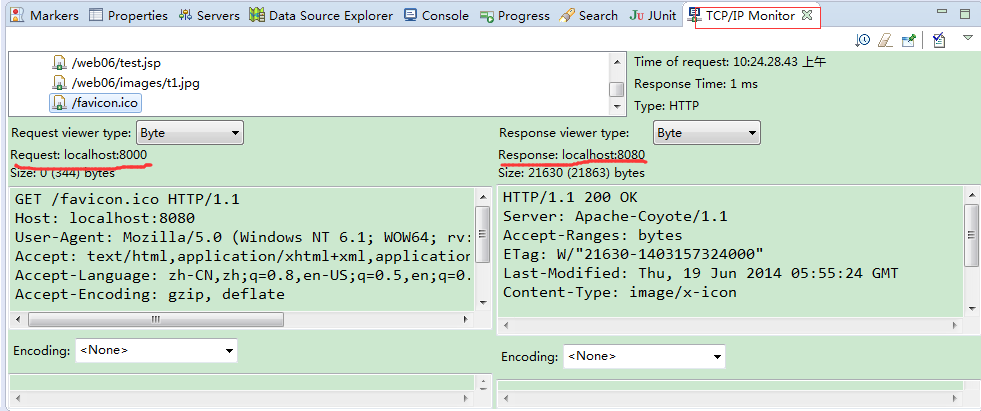
如上图所示，添加一个8000端口用以监听8080。确定后，我们要单击start启动TCP/IP Mointor。



接着，我们要为TCP/IP Mointor添加一个窗口,点击Window->Show View ->Other,选择Debug->TCP/IP Mointor。



访问后，我们可以看到TCP/IP Mointor中的监听数据：



## 7.2跟踪SOAP的请求和响应内容

### 7.2.1请求

|  |
| --- |
| POST /weatherService HTTP/1.1  Accept: text/xml, multipart/related  Content-Type: text/xml; charset=utf-8  SOAPAction: "http://impl.service.qf.com/WeatherServiceImpl/queryWeatherRequest"  User-Agent: JAX-WS RI 2.2.4-b01  Host: localhost:5555  Connection: keep-alive  Content-Length: 212  <?xml version="1.0" ?>  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  <S:Body>  <ns2:queryWeather xmlns:ns2="http://impl.service.qf.com/">  <arg0>青海</arg0>  </ns2:queryWeather>  </S:Body>  </S:Envelope> |

### 2.2.2响应

|  |
| --- |
| HTTP/1.1 200 OK  Transfer-encoding: chunked  Content-type: text/xml; charset=utf-8  Date: Wed, 13 Jun 2018 05:06:34 GMT  5e  <?xml version="1.0" ?>  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  <S:Body>  8d  <ns2:queryWeatherResponse xmlns:ns2="http://impl.service.qf.com/">  <return>大晴天</return>  </ns2:queryWeatherResponse>  </S:Body>  </S:Envelope>  0 |

# 8.WebService优缺点

## 7.1优点

1)采用XML格式封装数据，XML是跨平台的,所以webservice也是跨平台的。

2)支持面向对象

3)通过SOAP协议实现异地调用

## 7.2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，标签越来越大，导致webservice性能下降

# 9.WebService使用场景

## 7.1使用场景

1)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

2)服务端已经确定使用webservice，客户端不能选择，必须使用webservice

## 7.2不适用场景

1) 考虑性能时不建议使用webservice

2) 同构程序(一个公司中系统之间的接口)下不建议使用webservice，比如java用RMI(远程方法调用)，不需要翻译成XML的数据

# 10.通过注解规范发布接口

## 10.1相关注解

1. @WebService
2. @WebMethod
3. @WebResult
4. @WebParam

## 10.2 说明

注解的作用对webservice的接口规范化，如果修改webservice的接口内容，比如namespace、portType，必须要重新生成客户端调用代码。

# 11案例一

要求：发布查询一个省市县查询的服务，可以根据省的id查询所有的市，也可以更具市的id查询所有的县，而且数据的传输和响应都必须是xml格式的，要求实现分页

## 第一步：创建接口

|  |
| --- |
| **public** **interface** IAreaService {  **public** String getAreaListByParentId(String xmlStr);  } |

## 第二步：接口的实现类

|  |
| --- |
| @WebService  **public** **class** AreaServiceImpl **implements** IAreaService {  **private** IAreaDao areaDao = **new** AreaDaoImpl();  @Override  **public** String getAreaListByParentId(String xmlStr) {  Map<String, Object> map = parseXmlToMap(xmlStr);  Integer id = Integer.*parseInt*(map.get("id").toString());  Integer startIndex = Integer.*parseInt*(map.get("startIndex").toString());  Integer size = Integer.*parseInt*(map.get("size").toString());  List<BaseArea> list = areaDao.getAreaListByParentId(id, startIndex, size);  **return** parseListToxmlString(list);  }  **private** Map<String, Object> parseXmlToMap(String xmlStr) {  Map<String, Object> map = **new** HashMap<String,Object>();  **try** {  Document document = DocumentHelper.*parseText*(xmlStr);  Element root = document.getRootElement();  Iterator<Element> areas = root.elementIterator();  **while** (areas.hasNext()) {  Element element = (Element) areas.next();  map.put(element.getName(), element.getTextTrim());  }  } **catch** (DocumentException e) {  e.printStackTrace();  }  **return** map;  }  **private** String parseListToxmlString(List<BaseArea> list) {  Document document = DocumentHelper.*createDocument*();  Element root = document.addElement("areas");  **for** (BaseArea baseArea : list) {  Element area = root.addElement("area");  area.addElement("id").setText(baseArea.getAreaId());  area.addElement("name").setText(baseArea.getAreaName());  }  **return** document.asXML();  }  } |

## 第三步：Dao层

|  |
| --- |
| **public** **interface** IAreaDao {  **public** List<BaseArea> getAreaListByParentId(Integer id,Integer startIndex,Integer size);  } |

## 第四步：Dao层的实现类

|  |
| --- |
| **public** **class** AreaDaoImpl **implements** IAreaDao {  **private** **static** String *userName* = "root";  **private** **static** String *password* = "root";  **private** **static** String *driver* = "com.mysql.jdbc.Driver";  **private** **static** String *url* = "jdbc:mysql://localhost:3306/1709\_OA";  @Override  **public** List<BaseArea> getAreaListByParentId(Integer id, Integer startIndex, Integer size) {  Connection connection = **null**;  PreparedStatement prst = **null**;  ResultSet resultSet = **null**;  List<BaseArea> list = **new** ArrayList<BaseArea>();  **try** {  Class.*forName*(*driver*);  connection = DriverManager.*getConnection*(*url*, *userName*, *password*);  String sql = "select \* from t\_base\_area where area\_parent\_id = ? limit ?,?";  prst = connection.prepareStatement(sql);  prst.setInt(1, id);  prst.setInt(2, startIndex);  prst.setInt(3, size);  resultSet = prst.executeQuery();  **while** (resultSet.next()) {  BaseArea baseArea = **new** BaseArea();  **int** areaId = resultSet.getInt("area\_id");  String name = resultSet.getString("area\_name");  baseArea.setAreaId(areaId + "");  baseArea.setAreaName(name);  list.add(baseArea);  }  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (resultSet != **null**) {  **try** {  resultSet.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **if** (prst != **null**) {  **try** {  prst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **if** (connection != **null**) {  **try** {  connection.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  **return** list;  } |

## 第五步：发布服务

|  |
| --- |
| **public** **class** AreaService {  **public** **static** **void** main(String[] args) {  Endpoint.*publish*("http://localhost:8080/areaService/", **new** AreaServiceImpl());  }  } |

## 第六步：客户端调用

|  |
| --- |
| **public** **static** **void** main(String[] args) {  URL wsdlDocumentLocation;  **try** {  // wsdl路径  wsdlDocumentLocation = **new** URL("http://localhost:8080/areaService/?wsdl");  //  // 第一个参数是wsdl的命名空间  // 第二个参数是服务视图的名称  QName serviceName = **new** QName("http://impl.service.qf.com/", "AreaServiceImplService");  // 1.创建服务视图  Service service = Service.*create*(wsdlDocumentLocation, serviceName);  // 2.得到服务视图的portType  AreaServiceImpl areaServiceImpl = service.getPort(AreaServiceImpl.**class**);  String xml = "<areas>"+  "<id>1</id>"+  "<startIndex>0</startIndex>"+  "<size>6</size> "+  "</areas>";  String areaListByParentId = areaServiceImpl.getAreaListByParentId(xml);  System.***out***.println(areaListByParentId);  } **catch** (MalformedURLException e) {  e.printStackTrace();  }  } |

# 12 案例二

发布一个自定义POJO服务

# 13.cxf

## 11.1简介

Apache CXF = Celtix + XFire，开始叫 Apache CeltiXfire，后来更名为 Apache CXF。

Apache CXF 是一个开源的Web Service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" \t "_blank)/[HTTP](http://baike.baidu.com/view/9472.htm)、[REST](http://baike.baidu.com/view/1077487.htm" \t "_blank)ful HTTP。CXF 大大简化了 Web Service 的创建，可以天然地和 Spring 进行无缝集成，CXF支持数据格式有XML，JSON

## 11.2环境搭建

1.将下载的文件进行解压,配置环境变量，新建CXF\_HOME,输入下载文件库的路径，

示例：C:\software\apache-cxf-3.2.1\apache-cxf-2.7.11

2.在系统环境变量的Path追加%CXF\_HOME%\bin

3.测试，在cmd下加入wsdl2java –

注意使用cxf3.2.1版本时jdk必须要用1.8以上的。

## 11.3 利用CXF 发布/调用服务

### 11.3.1服务端

接口

|  |
| --- |
| @WebService // cxf中@WebService注解加到接口中  **public** **interface** IWeatherService {  **public** String queryWeacher(String city);  } |

实现类

|  |
| --- |
| **public** **class** WeatherServiceImpl **implements** IWeatherService {  **public** String queryWeacher(String city) {  **if** ("深圳".equals(city)) {  **return** "大晴天";  } **else** **if** ("西宁".equals(city)) {  **return** "多云";  }  **return** "暂无数据";  }  } |

发布服务

|  |
| --- |
| // 1.使用jaws发布WebService  JaxWsServerFactoryBean jaxWsServerFactoryBean = **new** JaxWsServerFactoryBean();    // 2.设置服务的地址  jaxWsServerFactoryBean.setAddress("http://localhost:8080/WeatherService");    // 3.设置服务接口  jaxWsServerFactoryBean.setServiceClass(IWeatherService.**class**);    //4.设置服务实现类  jaxWsServerFactoryBean.setServiceBean(**new** WeatherServiceImpl());    // 5.发布  jaxWsServerFactoryBean.create();  System.***out***.println("发布成功"); |

### 11.3.2客户端

#### 11.3.2.1利用CXF生成客户端代码

wsdl2java命令是CXF提供的生成客户端的工具，他和wsimport类似，可以根据WSDL生成客户端代码。

wsdl2java常用参数：

-d，指定输出目录

-p，指定包名，如果不指定该参数，默认包名是WSDL的命名空间的倒序

|  |
| --- |
|  |

#### 11.3.2.2客户端调用

如果使用cxf调用服务端，需要加入cxf的jar包。

|  |
| --- |
| // 1.利用JaxWsProxyFactoryBean调用服务  JaxWsProxyFactoryBean jaxWsProxyFactoryBean = **new** JaxWsProxyFactoryBean();  // 2.设置服务接口  jaxWsProxyFactoryBean.setServiceClass(IWeatherService.**class**);    // 3.设置服务地址  jaxWsProxyFactoryBean.setAddress("http://localhost:8080/WeatherService");    // 4.获取服务接口实例  IWeatherService weatherService = (IWeatherService)jaxWsProxyFactoryBean.create();    // 5.调用方法  String result = weatherService.queryWeacher("深圳");  System.***out***.println(result); |

原始的客户端代码也可以调用CXF发布的WebService服务。

Cxf发布webservice：

JaxWsServerFactoryBean：发布webservice服务

JaxWsProxyFactoryBean：调用webservice服务端

# 14.cxf整合Spring

Cxf框架本身依赖spring，从官方下载cxf包中有spring的jar包。

上边使用JaxWsServerFactoryBean和JaxWsProxyFactoryBean，改为spring配置方式:

发布服务：使用spring和cxf整合的标签<jaxws:server >

客户端调用服务：使用spring和cxf整合的标签<jaxws:client>

上边的<jaxws:server>和<jaxws:client>相当于spring容器的bean，发布服务和客户端调用交给spring容器管理。

## 12.1服务端

第一步：创建web工程，引入cxf的jar包

第二步：创建SEI接口\*\*。(SEI(WebService EndPoint Interface))是web service的终端接口，就是WebService服务器端用来处理请求的接口)

第三步：创建SEI接口实现类

第四步：配置Spring配置文件 applicationContext.xml\*\*

|  |
| --- |
| <beans xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:cxf=*"http://cxf.apache.org/cxf"*  xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"*  xmlns:jaxws=*"http://cxf.apache.org/jaxws"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.3.xsd*  *http://cxf.apache.org/cxf*  *http://cxf.apache.org/schemas/core.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"*>  <!-- 服务实现类 -->  <bean id=*"weatherServiceImpl"* class=*"com.qf.service.impl.WeatherServiceImpl"*/>  <!--  利用jaxws:server发布服务端  address:服务地址  serviceClass:服务接口的全类名  -->  <jaxws:server address=*"/WeatherService"* serviceClass=*"com.qf.service.IWeatherService"*>  <jaxws:serviceBean>  <!-- 引入服务的实现类 -->  <ref bean=*"weatherServiceImpl"*/>  </jaxws:serviceBean>  </jaxws:server>  </beans> |

第五步：配置web.xml\*\*

|  |
| --- |
| <!-- 配置Spring环境 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!--  配置CXF的Servlet用于解析cxf发布webservice。  -->  <servlet>  <servlet-name>CXF</servlet-name>  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>CXF</servlet-name>  <url-pattern>/ws/\*</url-pattern>  </servlet-mapping> |

第六步：启动tomcat,部署web工程到tomcat\*\*

第七步：测试服务是否发布成功\*\*

-WSDL地址规则：http://localhost:端口号/项目名称/servlet拦截路径/服务名称?wsdl

|  |
| --- |
|  |

## 12.2客户端

第一步：创建web工程作为客户端，引入cxf的jar包

第二步：生成客户端代码

示例：wsdl2java -p com.ws.cxf.client －d . <http://localhost/CXFSpring/ws/weather?wsdl>

第三步：配置spring的配置文件

|  |
| --- |
| <!--  使用jaxws:client调用服务端  serviceClass:设置服务接口(客户端生成的服务接口)  address:服务地址  -->  <jaxws:client id=*"weatherClient"* serviceClass=*"com.qf.springcxf.IWeatherService"*  address=*"http://localhost:8080/cxf\_webservice01/ws/WeatherService?wsdl"*></jaxws:client> |

第四步：初始化spring上下文，获取接口实现类，调用查询方法

|  |
| --- |
| **public** **class** CxfSpringClientDemo {  **private** ApplicationContext ctx = **null**;    @Before  **public** **void** init(){  // 初始化容器  ctx = **new** ClassPathXmlApplicationContext("applicationContext.xml");  }    @Test  **public** **void** cxfClientDemo(){    // 获取服务接口实例  IWeatherService uService = ctx.getBean("weatherClient", IWeatherService.**class**);    System.***out***.println(uService.queryWeacher("深圳"));  }  } |

# 15. CXF发布restFul服务

## 12.1 什么是restFul

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 的利用分为以下两种：资源定位和资源操作。

* 资源定位

更加准确去定位一个互联网资源。使用url定位一个互联网资源。

* 资源操作

利用HTTP 的GET、POST、PUT、DELETE 四种操作来表示数据库操作的SELECT、UPDATE、INSERT、DELETE 操作。

## 12.2 cxf发布restFul风格的webservice

### 1.创建学生类，需要加入@ XmlRootElement

|  |
| --- |
| @XmlRootElement(name="student") //@XmlRootElement可以实现对象和XML数据之间的转换  **public** **class** Student {  **private** Integer id;    **private** String name;    **private** Integer age;    **private** Date brithday;  } |

### 2.创建SEI接口

|  |
| --- |
| @WebService  @Path("/student")  **public** **interface** IStudentService {  @GET // 指定请求方式，如果服务端发布的时候指定的是GET，那么客户端访问时必须使用GET  @Path("queryStudentById/{id}")  @Produces(value=MediaType.***APPLICATION\_XML***) // /指定服务端返回数据类型  **public** Student queryStudentById(@PathParam("id")Integer id);    @GET  @Path("findStudentList")  // @Produces({MediaType.APPLICATION\_XML,MediaType.APPLICATION\_JSON)  @Produces({MediaType.***APPLICATION\_XML***,"application/json;charset=utf-8"})  // 后面追加?\_type=xml返回xml  // 后面追加?\_type=json返回json  **public** List<Student> findStudentList();  } |

### 3.创建SEI接口实现类

|  |
| --- |
| **public** **class** StudentServiceImpl **implements** IStudentService{  @Override  **public** Student queryStudentById(Integer id) {  System.***out***.println("StudentServiceImpl.queryStudentById():"+id);  Student student = **new** Student();  student.setId(id);  student.setName("张三");  student.setAge(22);  student.setBrithday(**new** Date());  **return** student;  }  @Override  **public** List<Student> findStudentList() {  List<Student> students = **new** ArrayList<Student>();  **for**(**int** i=0;i<10;i++){  Student student = **new** Student();  student.setId(i+1);  student.setName("张\_"+i);  student.setAge(i+10);  student.setBrithday(**new** Date());  students.add(student);  }  **return** students;  }  } |

### 4.发布服务编程方式

|  |
| --- |
| // 1.使用JAXRSS发布WebService  JAXRSServerFactoryBean jaxrsServerFactoryBean = **new** JAXRSServerFactoryBean();  // 2.设置服务的地址  jaxrsServerFactoryBean.setAddress("http://localhost:8080/studentServie");  // 3.设置服务实现类  jaxrsServerFactoryBean.setServiceBean(**new** StudentServiceImpl());  // 4.设置资源对象,如果有多个资源类，可以以“,”隔开。  jaxrsServerFactoryBean.setResourceClasses(Student.**class**);  // 5.发布  jaxrsServerFactoryBean.create();  System.***out***.println("发布成功"); |

编程式的发布后地址是xxxxx?\_wadl

## 12.3 客户端

客户端还是使用之前的最原始的方式调用

|  |
| --- |
| // 1.创建服务地址，不是WSDL地址  URL url = **new** URL("http://localhost:8080/studentServie/student/findStudentList?\_type=json");  // 2.打开一个通向服务地址的连接  HttpURLConnection connection = (HttpURLConnection) url.openConnection();  // 3.设置请求请求方式,  connection.setRequestMethod("GET");  // 4.设置输入输出，默认是false没有读写的权限  connection.setDoOutput(**true**);  connection.setDoInput(**true**);  **if** (connection.getResponseCode() == 200) {  InputStream ips = connection.getInputStream();  Scanner scanner = **new** Scanner(ips);  StringBuffer buffer = **new** StringBuffer();  **while** (scanner.hasNextLine()) {  buffer.append(scanner.nextLine());  }  scanner.close();  System.***out***.println(buffer);  } |

## 12.4 cxf配置的方式发布rest

之前的实体类，接口，实现类都保持不变，只需要在spring配置中发布即可。

修改Spring的配置文件如下：

|  |
| --- |
| <!-- 配置的方式发布Rest服务 -->  <!-- 服务实现类 -->  <bean id=*"studentServiceImpl"* class=*"com.qf.service.impl.StudentServiceImpl"*/>    <!--  address:服务地址  -->  <jaxrs:server address=*"/studentService"* >  <jaxrs:serviceBeans>  <ref bean=*"studentServiceImpl"*/>  </jaxrs:serviceBeans>  </jaxrs:server> |

# 17.综合案例

集成公网手机号归属地查询服务  
对外发布自己的手机号归属地查询服务  
提供查询界面

第一步：创建web项目（引入jar包）

第二步：生成公网客户端代码

|  |
| --- |
| wsimport -extension -s . -p com.qf.mobile http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl |

## 第三步：创建SEI接口

|  |
| --- |
| @WebService  **public** **interface** IMobileService {  **public** String queryMobile(String code);  } |

## 第四步：创建SEI实现类

|  |
| --- |
| **public** **class** MobileServiceImpl **implements** IMobileService {  // 需要注入公网号码查询服务  **private** MobileCodeWSSoap mobileCodeWSSoap;    @Override  **public** String queryMobile(String code) {  **return** getMobileCodeWSSoap().getMobileCodeInfo(code, "");  }  /\*\*  \* **@return** the mobileCodeWSSoap  \*/  **public** MobileCodeWSSoap getMobileCodeWSSoap() {  **return** mobileCodeWSSoap;  }  /\*\*  \* **@param** mobileCodeWSSoap the mobileCodeWSSoap to set  \*/  **public** **void** setMobileCodeWSSoap(MobileCodeWSSoap mobileCodeWSSoap) {  **this**.mobileCodeWSSoap = mobileCodeWSSoap;  }  } |

## 第五步：创建queryMobile.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <base href=*"*<%=request.getContextPath() + "/"%>*"*>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <form action=*"MobilerController/queryMobile"* method=*"post"*>  输入手机号：<input type=*"text"* name=*"mobile"*><br>  <input type=*"submit"* value=*"查询"*>  </form>  ${msg}  </body>  </html> |

## 第六步：创建MobileController

|  |
| --- |
| @Controller  @RequestMapping(value = "MobilerController/")  **public** **class** MobilerController {  @Autowired  **private** IMobileService mobildeService;  @RequestMapping(value = "/queryMobile")  **public** String queryMobile(String mobile, ModelMap map) {  System.***out***.println("MobilerController.queryMobile()" + mobile);  **if** (mobile != **null** && !"".equals(mobile)) {  map.put("msg", mobildeService.queryMobile(mobile));  }  **return** "index";  }  } |

## 第七步：配置spring配置文件，applicationContext.xml

|  |
| --- |
| <!--  公网手机查询客户端  serviceClass:prtType的名字  address:portType对应的地址  -->  <jaxws:client id=*"mobileClinet"* serviceClass=*"com.qf.mobile.MobileCodeWSSoap"*  address=*"http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx"*>  </jaxws:client>  <!-- 手机号查询的服务类 -->  <bean id=*"mobileServiceImpl"* class=*"com.qf.service.impl.MobileServiceImpl"*>  <property name=*"mobileCodeWSSoap"* ref=*"mobileClinet"*></property>  </bean>      <!--  发布手机号服务  serviceClass:手机号查询的接口  address:访问服务地址  -->  <jaxws:server address=*"/mobile"* serviceClass=*"com.qf.service.IMobileService"*>  <jaxws:serviceBean>  <ref bean=*"mobileServiceImpl"*/>  </jaxws:serviceBean>  </jaxws:server> |

## 第八步：配置web.xml

|  |
| --- |
| <!-- 配置Spring环境 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- SpringMV核心的组件 -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    <!--配置CXF的Servlet -->  <servlet>  <servlet-name>CXF</servlet-name>  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>CXF</servlet-name>  <url-pattern>/ws/\*</url-pattern>  </servlet-mapping> |