网上下载，分享给爱好技术的朋友，共同努力！！

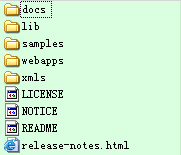
**Axis1.x**

**一、搭建简单的axis web服务**

1、在官方网站下载axis的工程（这个等下就有用的）和源码、jar包等，下载地址是：

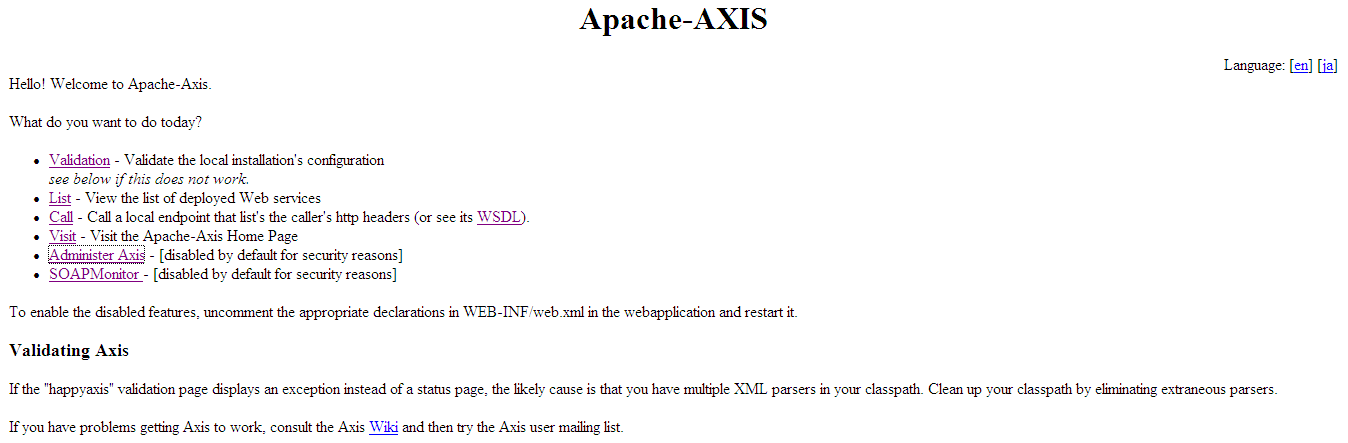
[http://labs.renren.com/apache-mirror//ws/axis/1\_4/](http://labs.renren.com/apache-mirror/ws/axis/1_4/)

2、解压下载的工程或源码（两个中任意一个都可以），解压axis-bin-1.4可以看到大致目录是这样的：



docs是文档、lib是jar包、sample是示例、xmls是当前工程所需的xml、webapps是当前工程的webroot目录；

我们打开webapps目录就可以看到一个axis的文件夹，这个文件夹里面有WEB-INF文件夹和一些页面，将axis复制到你的tomcat的webapps目录下。然后启动tomcat服务，访问<http://localhost:8080/axis/>，看到下面的解码就说明部署成功了：



以后我们将和这个工程不离不弃，它将在我们的axis1.x的webService中发挥很大的作用！

3、创建我们自己的web工程，这里我新建的AxisWebService；创建好工程后，将刚才解压的axis-bin中的lib的jar包copy到当前工程的lib中；

axis-ant.jar

axis.jar

commons-discovery-0.2.jar

commons-logging-1.0.4.jar

jaxrpc.jar

log4j-1.2.8.jar

saaj.jar

wsdl4j-1.5.1.jar

activation-1.1.jar

mail-1.4.jar

创建webService类文件，代码如下：

|  |
| --- |
| **package** com.hoo.service;  /\*\*  \* <b>function:</b>jws的axis WebService  \* **@author** hoojo  \* **@createDate** Dec 15, 2010 17:03:49 PM  \* **@file** HelloWorldService.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloWorldService {  **public** String sayHello(String name, **int** age) {  **return** name + " say : hello world! [axis] my age is " + age;  }  } |

4、复制HelloWorldService.java到我们刚才复制的axis文件夹下即可；也就是tomcat下的webapps下的axis下即可；注意：还有重要的一般就是要将这个java文件中的包名去掉，并且将这个文件重命名为HelloWorldService.jws；如果带包名的话，请求后编译的class将会在包路径下，这样我们在全球当前jws的时候就会出现找不到class，详细的你可以到发布在tomcat下的工程看看WEB-INF目录下的jwsClass就一目了然了。

上面的工作完成后，启动tomcat服务器，访问[http://localhost:8080/axis/](http://localhost:8080/axis/HelloWorldService.jws)[HelloWorldService.jws](http://localhost:8080/axis/HelloWorldService.jws)

你会看到：

There is a Web Service here

[Click to see the WSDL](http://localhost:8080/axis/HelloWorldService.jws?wsdl)

如果你和我看到的是一样的，就证明你已经成功的部署了一个axis1.x的webService。然后我们点击下就可以看到wsdl的xml文件了，内容如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:definitions targetNamespace="**http://localhost:8080/axis/HelloWorldService.jws**" xmlns:apachesoap="**http://xml.apache.org/xml-soap**" xmlns:impl="**http://localhost:8080/axis/HelloWorldService.jws**" xmlns:intf="**http://localhost:8080/axis/HelloWorldService.jws**" xmlns:soapenc="**http://schemas.xmlsoap.org/soap/encoding/**" xmlns:wsdl="**http://schemas.xmlsoap.org/wsdl/**" xmlns:wsdlsoap="**http://schemas.xmlsoap.org/wsdl/soap/**" xmlns:xsd="**http://www.w3.org/2001/XMLSchema**">  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <!--  WSDL created by Apache Axis version: 1.4  Built on Apr 22, 2006 (06:55:48 PDT)    -->  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:message name="**sayHelloResponse**">    <wsdl:part name="**sayHelloReturn**" type="**xsd:string**" />    </wsdl:message>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:message name="**sayHelloRequest**">    <wsdl:part name="**name**" type="**xsd:string**" />    <wsdl:part name="**age**" type="**xsd:int**" />    </wsdl:message>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:portType name="**HelloWorldService**">  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:operation name="**sayHello**" parameterOrder="**name age**">    <wsdl:input message="**impl:sayHelloRequest**" name="**sayHelloRequest**" />    <wsdl:output message="**impl:sayHelloResponse**" name="**sayHelloResponse**" />    </wsdl:operation>    </wsdl:portType>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:binding name="**HelloWorldServiceSoapBinding**" type="**impl:HelloWorldService**">    <wsdlsoap:binding style="**rpc**" transport="**http://schemas.xmlsoap.org/soap/http**" />  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:operation name="**sayHello**">    <wsdlsoap:operation soapAction="" />  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:input name="**sayHelloRequest**">    <wsdlsoap:body encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**" namespace="**http://DefaultNamespace**" use="**encoded**" />    </wsdl:input>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:output name="**sayHelloResponse**">    <wsdlsoap:body encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**" namespace="**http://localhost:8080/axis/HelloWorldService.jws**" use="**encoded**" />    </wsdl:output>    </wsdl:operation>    </wsdl:binding>  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:service name="**HelloWorldServiceService**">  [**-**](http://localhost:8080/axis/HelloWorldService.jws?wsdl) <wsdl:port binding="**impl:HelloWorldServiceSoapBinding**" name="**HelloWorldService**">    <wsdlsoap:address location="**http://localhost:8080/axis/HelloWorldService.jws**" />    </wsdl:port>    </wsdl:service>    </wsdl:definitions> |

分析下wsdl的xml文件内容：

targetNamespace=[**http://localhost:8080/axis/HelloWorldService.jws**](http://localhost:8080/axis/HelloWorldService.jws)

是我们部署的webservice命名空间，也就是我们访问的webService路径。

<wsdl:message name="**sayHelloResponse**">

<wsdl:part name="**sayHelloReturn**" type="**xsd:string**" />

</wsdl:message>

是返回值的信息，**sayHelloResponse**代表响应，即返回值，type是返回值的类型

<wsdl:message name="**sayHelloRequest**">

<wsdl:part name="**name**" type="**xsd:string**" />

<wsdl:part name="**age**" type="**xsd:int**" />

</wsdl:message>

请求方法参数信息，**sayHelloRequest**即请求，part是参数parameter，type是参数的类型

<wsdl:portType name="**HelloWorldService**">

<wsdl:operation name="**sayHello**" parameterOrder="**name age**">

  <wsdl:input message="**impl:sayHelloRequest**" name="**sayHelloRequest**" />

  <wsdl:output message="**impl:sayHelloResponse**" name="**sayHelloResponse**" />

  </wsdl:operation>

  </wsdl:portType>

portType的name是当前webService的名称，operation是一个操作，即可以调用的方法。name就是方法名称了，parameterOrder是参数，input输入即传入参数，output输出即返回的值；

<wsdl:service name="**HelloWorldServiceService**">

<wsdl:port binding="**impl:HelloWorldServiceSoapBinding**" name="**HelloWorldService**">

  <wsdlsoap:address location="**http://localhost:8080/axis/HelloWorldService.jws**" />

  </wsdl:port>

  </wsdl:service>

webService的名称和绑定的信息，以及访问的url地址。

5、下面编写客户端代码

代码如下：

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **public** **class** HelloWorldClient {  /\*\*  \* <b>function:</b>jws axis WebService客户端  \* **@author** hoojo  \* **@createDate** 2010-12-15 下午05:10:28  \* **@param** args  \* **@throws** ServiceException  \* **@throws** RemoteException  \*/  **public** **static** **void** main(String[] args) **throws** ServiceException, RemoteException {  //webService访问地址  //String url = "http://localhost:8080/axis/HelloWorldService.jws";  String url = "http://127.0.0.1:8080/axis/HelloWorldService.jws";    //创建服务  Service service = **new** Service();  //创建调用句柄  Call call = (Call) service.createCall();  //设置请求地址  call.setTargetEndpointAddress(url);  /\*\*  \* 设置调用的方法和方法的命名空间；  \* 因为这里是手动发布到webroot目录下的，所以命名空间和请求地址一致  \* 当然null也可以，因为本身它就没有设置命名空间，一般方法的命名空间是  \* 包名倒写组成，如com.hoo.service,ns=http://service.hoo.com  \*/  call.setOperationName(**new** QName(**null**, "sayHello"));  /\*\*  \* 用call调用sayHello方法，设置请求的参数，返回的就是返回值了  \*/  String result = (String) call.invoke(**new** Object[] { "jack", 99 });  System.*out*.println(result);  }  } |

分析上面的代码

url是根据xml文件中的wsdlsoap:address location的信息得到的，命名空间和方法名称是根据

<wsdl:operation name="sayHello">

<wsdlsoap:operation soapAction="" />

- <wsdl:input name="sayHelloRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://DefaultNamespace" use="encoded" />

</wsdl:input>

- <wsdl:output name="sayHelloResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/axis/HelloWorldJWS.jws" use="encoded" />

</wsdl:output>

的信息得到的，而请求参数和返回值的详细信息是在

<wsdl:message name="sayHelloRequest">

<wsdl:part name="name" type="xsd:string" />

<wsdl:part name="age" type="xsd:int" />

</wsdl:message>

- <wsdl:message name="sayHelloResponse">

<wsdl:part name="sayHelloReturn" type="xsd:string" />

</wsdl:message>

- <wsdl:portType name="HelloWorldJWS">

- <wsdl:operation name="sayHello" parameterOrder="name age">

<wsdl:input message="impl:sayHelloRequest" name="sayHelloRequest" />

<wsdl:output message="impl:sayHelloResponse" name="sayHelloResponse" />

</wsdl:operation>

</wsdl:portType>

里可以很详细的看到。

至于代码的call.invoke是java中反射机制，不懂的建议看看jdk文档java.lang.reflect包下的内容。

运行上面的代码就可以看到控制台输出：

jack say : hello world! [axis] my age is 99

好了，axis的就完成了，下面我们不用官方的axis的工程，我们写一个自己的AxisWebService工程，然后发布的tomcat的webapps中看看。

6、刚才copy了lib下的jar包，现在要copy下web.xml中的内容，去掉里面的AdminServlet这个配置，其他的都可保留。

然后像刚才一样，将HelloWorldService.java复制到webroot目录下，去掉包名，并且修改后缀为HelloWorldService.jws即可。（如果有兴趣可以看看，发布在tomcat目录下的当前工程的web-inf目录，看看里面是否多了些东西）最后发布当前web工程，访问<http://localhost:8080/AxisWebService/HelloWorldService.jws>，如果看到和刚才一样的界面，证明你快成功了。点击链接看到wsdl的xml就成功了。

好了，还没有完。看看web.xml中的配置，你大概就知道为什么了。

web.xml中最主要的文件就是org.apache.axis.transport.http.AxisServlet，它就是webService的中央控制器；即配置jws的后缀也在web.xml中定义的，在看看还有services/\*，这就表明上面的访问路径也可以是这样的：<http://localhost:8080/AxisWebService/services/HelloWorldService>

当然如果要这样写就需要用wsdd的发布方式，详细请看下文！

参考文档：<http://ws.apache.org/axis/java/user-guide.html>

**二、用wsdd（Web Services Deployment Descriptor）方式发布WebService**

wsdd方法比jws方法要稍微复杂些，但一定程度上比jws发布的方法要灵活。

1、首先在AxisWebService工程创建一个java类，里面写简单的2个方法getName、getAge，代码如下：

|  |
| --- |
| **package** com.hoo.service;  /\*\*  \* <b>function:</b>wsdd发布模式的axis WebService  \* **@author** hoojo  \* **@createDate** Dec 15, 2010 17:24:35 PM  \* **@file** HelloWorldWSDD.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloWorldWSDD {    **public** String getName(String name) {  **return** "your name : " + name;  }    **public** **int** getAge(**int** age) {  **return** age + 10;  }  } |

2、如果用wsdd方法首先需要定制我们的wsdd xml文件，这里命名为deploy.wsdd，创建在当前web工程的WEB-INF目录下，代码如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <deployment xmlns="http://xml.apache.org/axis/wsdd/"  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  <service name="HelloWorldWSDD" provider="java:RPC">  <parameter name="className" value="com.hoo.service.HelloWorldWSDD" />  <!-- \* 代表所有的方法都暴露 -->  <parameter name="allowedMethods" value="\*" />  <parameter name="scope" value="request" />  </service>  </deployment> |

service标签代表一个WebService服务，HelloWorldWSDD就是当前WebService的名称；provider是java的WebService类型，分别有： RPC、Document、Wrapped、Message、EJB、RMI；有兴趣的可以看看org.apache.axis.providers.java包下面的WebService的实现类或是文档；

parameter的参数className代表当前WebService的class类路径；

allowedMethods代表暴露的方法，那些方法在客户端可以调用；

<parameter name="scope" value="request" />

这个是当前WebService的作用域，它有3个值，分别是：request、session、application。

request代表为每个WebService SOAP的请求都产生一个服务对象，和Spring的scope很像，在服务请求频繁的话会消耗很多资源。

session 是给每个调用当前WebService的客户端创建一个服务对象

application 是个当前所有的请求创建一个服务对象

3、写完配置后，就需要用axis提供的AdminClient工具类帮我们发布WebService，直到生成server-config.wsdd，步骤如下：运行cmd命令，然后进入当前工程发布的目录，即%tomcat\_home%/webapps/project/WEB-INF>

我的是：C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>

然后输入命令：java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient deploy.wsdd

这里的deploy.wsdd是我们刚才定制的wsdd文件，java当然是jvm的命令，-Djava.ext.dirs=lib设置当前命令的依赖包，AdminClient是axis提供的工具类，这个类本来是可以在官方的工程中admin可以直接运行的（这里不可以，下载下来的少了AdminServlet，有兴趣的可以研究下，就是前面说的官方的示例）；

如果运行命令后，看到：

Processing file deploy.wsdd

<Admin>Done processing</Admin>

就代表快成功了，why？快！ 看看deploy.wsdd同级目录有没有生成server-config.wsdd如果有这个文件就成功了，没有就失败了。如果失败了还有解决办法，首先你得启动tomcat，将我们的工程发布出去。然后在重复上面的命令，不行就换命令行代码如下：

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deploy.wsdd

-lhttp://localhost:8080/AxisWebService/services/AdminService是因为你的端口可能被axis占用了，我们将指定AdminService来完成转换，运行上面命令就没有问题了。

4、在WebBrowser的地址栏输入：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

你就可以看到如下效果：

## And now... Some Services

* HelloWorldWSDD [*(wsdl)*](http://localhost:8080/AxisWebService/services/HelloWorldWSDD?wsdl)
  + getName
  + getAge
* AdminService [*(wsdl)*](http://localhost:8080/AxisWebService/services/AdminService?wsdl)
  + AdminService
* Version [*(wsdl)*](http://localhost:8080/AxisWebService/services/Version?wsdl)
  + getVersion

刚才在deploy.wsdd中指定的WebService就在上面出现了，还有暴露出来的方法。点击wsdl就可以看到刚才一样熟悉的wsdl的xml文档。

5、下面我们编写客户端代码调用上面的WebService，代码块和上面的jws几乎一样，只是参数，方法名称不同。

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  /\*\*  \* <b>function:</b>wsdd 模式的axis WebService客户端  \* **@author** hoojo  \* **@createDate** Dec 15, 2010 17:30:48 PM  \* **@file** HelloWorldWSDDClient.java  \* **@package** com.hoo.client  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloWorldWSDDClient {  **public** **static** **void** main(String[] args) **throws** ServiceException, RemoteException {  //webService访问地址  String url = "http://localhost:8080/AxisWebService/services/HelloWorldWSDD";  //创建服务  Service service = **new** Service();  //创建调用句柄  Call call = (Call) service.createCall();  //设置请求地址  call.setTargetEndpointAddress(url);  /\*\*  \* 设置调用的方法和方法的命名空间；  \* 当然null也可以，因为本身它就没有设置命名空间，一般方法的命名空间是  \* 包名倒写组成，如com.hoo.service,ns=http://service.hoo.com  \*/  call.setOperationName(**new** QName("http://service.hoo.com", "getName"));    /\*\*  \* 用call调用getName方法，设置请求的参数，返回的就是返回值了  \*/  String result = (String) call.invoke(**new** Object[] { "jack" });  System.*out*.println(result);    call.setOperationName(**new** QName("http://service.hoo.com", "getAge"));    /\*\*  \* 用call调用getAge方法，设置请求的参数，返回的就是返回值了  \*/  **int** resultAge = Integer.*parseInt*(call.invoke(**new** Object[] { 89 }).toString());  //服务器端+10  System.*out*.println(resultAge);  }  } |

运行后结果如下：

your name : jack

99

call.setOperationName(new QName("http://service.hoo.com", "getName"));

这里设置命名空间，也可以用null代替，但是不能随意设置，有时候会出现异常。

**三、取消发布的WebService（删除一个发布成功的Web服务）**

首先，取消发布也需要定制wsdd文件，undeploy.wsdd具体格式如下：

|  |
| --- |
| <undeployment xmlns="http://xml.apache.org/axis/wsdd/">  <service name="HelloWorldWSDD"/>  </undeployment> |

同样，像上面发布WebService一样，用命令完成。

命令如下：

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService undeploy.wsdd

键入命令后，回车看到：

Processing file undeploy.wsdd

<Admin>Done processing</Admin>

就代表你取消服务成功了，那样别人就不能访问你的服务了。再在浏览器地址栏输入：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

看看是不是刚才发布的服务被成功取消了！

**四、handler的作用及特性**

Axis中有一个basicHandler的抽象类，这个类可以供我们继承。可以完成很多复杂的功能，如：统计当前服务被调用的次数、验证服务调用信息等待

1、下面我们继承这个类，完成统计服务调用次数的功能；

RequestHandler代码如下：

|  |
| --- |
| **package** com.hoo.service.handler;  **import** org.apache.axis.AxisFault;  **import** org.apache.axis.MessageContext;  **import** org.apache.axis.handlers.BasicHandler;  /\*\*  \* <b>function:</b>继承抽象类BasicHandler完成服务的调用统计功能  \* **@author** hoojo  \* **@createDate** Dec 15, 2010 11:00:20 PM  \* **@file** RequestHandler.java  \* **@package** com.hoo.service.handler  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** RequestHandler **extends** BasicHandler {  **private** **static** **final** **long** *serialVersionUID* = 2128841024713159133L;  **private** **static** **long** *COUNT* = 0L;  **private** **static** **int** *REQUEST\_COUNT* = 0;  **public** **void** invoke(MessageContext msgContext) **throws** AxisFault {  /\*\*  \* MessageContext相当于当前Services的一个上下文，  \* 里面存储的是当前服务和调用的一些信息  \*/  *COUNT* ++;  *REQUEST\_COUNT* ++;  String state = (String) **this**.getOption("state");  System.*out*.println("state: " + state + ", count: " + *COUNT* + ", requestCount: " + *REQUEST\_COUNT*);  }  } |

BasicHandler是一个抽象类，继承这个类需要实现invoke签名方法，上面的代码很简单，就是一个累加统计，MessageContext是当前WebService的一个上下文，可以得到当前服务的公共信息和请求信息。这里的this.getOption("state");是和wsdd中的parameter对应的。

2、编写wsdd文件，加入RequestHandler配置信息，wsdd配置如下：

deployHandler.wsdd:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <deployment xmlns="http://xml.apache.org/axis/wsdd/"  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  <handler name="RequestHandler" type="java:com.hoo.service.handler.RequestHandler">  <parameter name="state" value="success" />  </handler>  <service name="HelloWorldWSDDHandler" provider="java:RPC">  <requestFlow>  <handler type="RequestHandler"/>  </requestFlow>  <parameter name="className" value="com.hoo.service.HelloWorldWSDD" />  <!-- \* 代表所有的方法都暴露 -->  <parameter name="allowedMethods" value="\*" />  <parameter name="scope" value="request" />  <responseFlow>  <handler type="RequestHandler"/>  </responseFlow>  </service>  </deployment> |

handler元素配置的就是在上面编写的RequestHandler，type就是classpath；里面的parameter和上面RequestHandler中的this.getOption(“state”)的对应。

service元素中的requestFlow handler是请求的时候调用的handler、responsHandler是响应的时候调用的handler。

3、发布当前WebService，和前面的一样运行命令行：

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployHandler.wsdd

看到下面的就表示执行成功

Processing file undeploy.wsdd

<Admin>Done processing</Admin>

然后在浏览器中请求：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

就可以看到发布的服务，下面编写客户端代码

4、客户端和上面HelloWorldWSDDClient一样，这里就不写了。

调用后可以看到服务器端的控制台输出：

state: success, count: 5, requestCount: 5

state: success, count: 6, requestCount: 6

state: success, count: 7, requestCount: 7

state: success, count: 8, requestCount: 8

why? 为什么是4次？首先请求的时候执行一次handler，响应执行一次。然后调用2个方法，那就响应了2次，所以是4次。可以注销一个方法看看是不是少了2次，state是配置文件中的消息。

**五、Chain链的作用**

Chain是一个链，是Handler的链。也就是说一次请求可能要执行多个Handler。

1、下面我们就编写一个Handler，让它用chain的方法执行。

|  |
| --- |
| **package** com.hoo.service.chain;  **import** org.apache.axis.SimpleChain;  **import** com.hoo.service.handler.RequestHandler;  **import** com.hoo.service.handler.ResponseHandler;  /\*\*  \* <b>function:</b>用chain实现调用多个handler  \* **@author** hoojo  \* **@createDate** Dec 15, 2010 11:25:27 PM  \* **@file** HandlerChain.java  \* **@package** com.hoo.service.chain  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HandlerChain **extends** SimpleChain {  **private** **static** **final** **long** *serialVersionUID* = 1226195692055911394L;    **public** HandlerChain() {  RequestHandler request = **new** RequestHandler();  ResponseHandler response = **new** ResponseHandler();  **this**.addHandler(request);  **this**.addHandler(response);  }  } |

上面的HandlerChain继承了SimpleChain，在构造器中创建了RequestHandler、ResponseHandler，然后用addHandler添加到chain的handler中。特别说明下，这里的chain其实也是一个handler，为什么这样说？我们继承的SimpleChain就继承了BsaicHandler这个抽象类，所以chain也是一个handler。有人想既然是handler那它的配置也应该和handler差不多，没错chain的配置可以引用handler的配置。

2、配置chain的wsdd文件

先看配置方式1：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <deployment xmlns=*"http://xml.apache.org/axis/wsdd/"*  xmlns:java=*"http://xml.apache.org/axis/wsdd/providers/java"*>  <chain name=*"HandlerChain"*>  <handler type=*"java:com.hoo.service.chain.HandlerChain"*/>  </chain>    <service name=*"HelloWorldWSDDHandlerChain"* provider=*"java:RPC"*>  <requestFlow>  <chain type=*"HandlerChain"*/>  </requestFlow>  <parameter name=*"className"* value=*"com.hoo.service.HelloWorldWSDD"* />  <parameter name=*"allowedMethods"* value=*"\*"* />  <parameter name=*"scope"* value=*"request"* />  </service>  </deployment> |

在chain元素中配置了一个handler标签，这个标签的type就是classpath类路径，然后在requestFlow标签中配置chain元素标签，用type引用上面的chain即可。

配置方式2：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <deployment xmlns=*"http://xml.apache.org/axis/wsdd/"*  xmlns:java=*"http://xml.apache.org/axis/wsdd/providers/java"*>    <handler name=*"hchain"* type=*"java:com.hoo.service.chain.HandlerChain"*/>    <chain name=*"handlerChain"*>  <handler type=*"hchain"*/>  </chain>    <service name=*"HelloWorldWSDDHandlerChain2"* provider=*"java:RPC"*>  <parameter name=*"className"* value=*"com.hoo.service.HelloWorldWSDD"* />  <parameter name=*"allowedMethods"* value=*"\*"* />  <parameter name=*"scope"* value=*"request"* />  <responseFlow>  <chain type=*"handlerChain"*/>  </responseFlow>  </service>  </deployment> |

上面先配置了handler，刚才说过chain也是一种特别的handler。所以用handler标签配置chain，然后在chain标签中用handler的type引用hchain，最好在responseFlow标签中用chain引用上面配置好的chain。

注意：chain标签中只能配置handler、chain这2种元素，chain里面可以嵌套配置chain；

3、发布wsdd配置过chain的WebService

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployChain.wsdd

在浏览器中请求：<http://localhost:8080/AxisWebService/servlet/AxisServlet>

就可以看到发布的服务了。

4、编写客户端代码，请求刚才发布的WebService。客户端的代码和上面的HelloWorldWSDDClient几乎引用，只是请求的服务换成刚才我们发布的那个服务路径就可以了。运行后服务器端控制台效果如下：

state: null, count: 1, requestCount: 1

state: null, count: 1, RESPONSE\_COUNT: 1

state: null, count: 2, requestCount: 2

state: null, count: 2, RESPONSE\_COUNT: 2

**六、传递、返回复杂类型的对象**

上面的服务传递的是简单数据类型，返回的也是简单数据类型。下面我们用JavaBean 、List、Map、Array等返回值或参数完成WebService。

1、首先看看JavaBean模式的参数和返回值的WebService，JavaBean代码如下：

|  |
| --- |
| **package** com.hoo.entity;  **import** java.io.Serializable;  /\*\*  \* <b>function:</b>User Entity  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 10:20:02 PM  \* **@file** User.java  \* **@package** com.hoo.entity  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** User **implements** Serializable {  **private** **static** **final** **long** *serialVersionUID* = 677484458789332877L;  **private** **int** id;  **private** String name;  **private** String email;  **private** String address;  //Constractor  **//getter、setter**  @Override  **public** String toString() {  **return** **this**.id + "#" + **this**.name + "#" + **this**.email + "#" + **this**.address;  }  } |

没有上面特别就是一个普通的Java对象，里面提供一些属性及getter、setter方法和toString方法（在显示数据的时候有用），值得注意的是在远程调用的Java对象都需要实现Serializable接口，这样在网络中传递对象的时候就不会出现重复的JavaBean。用过或是了解EJB、RMI的朋友应该知道这点。 序列化后的对象可以用io进行序列化写入到文件中，也可以反序列化解析到对象中的数据。

2、好，回正题。下面编写WebService的服务器端代码。

|  |
| --- |
| **package** com.hoo.service;  **import** java.util.ArrayList;  **import** java.util.Date;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** com.hoo.entity.User;  /\*\*  \* <b>function:</b>axis WebService传递复杂类型数据  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 10:21:06 PM  \* **@file** ManagerUser.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @SuppressWarnings("deprecation")  **public** **class** ManagerUser {    **public** User getUserByName(String name) {  User user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName(name);  user.setAddress("china");  user.setEmail(name + "@hoo.com");  **return** user;  }    **public** **void** setUser(User user) {  System.*out*.println(user);  }    **public** List<User> getUsers(**int** i) {  List<User> users = **new** ArrayList<User>();  **for** (**int** j = 0; j <= i; j++) {  User user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName("jack#" + j);  user.setAddress("china");  user.setEmail("jack" + j + "@hoo.com");  users.add(user);  }  **return** users;  }    **public** **void** setUserMap(Map<String, User> maps) {  System.*out*.println(maps);  }    **public** User[] getUserArray(**int** i) {  User[] users = **new** User[i];  **for** (**int** j = 0; j < i; j++) {  User user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName("jack#" + j);  user.setAddress("china");  user.setEmail("jack" + j + "@hoo.com");  users[j] = user;  }  **return** users;  }    **public** **void** setUserArray(User[] users) {  **for** (User u : users) {  System.*out*.println(u);  }  }    **public** Map<String, User> getUserMap() {  Map<String, User> users = **new** HashMap<String, User>();  User user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName("jack#");  user.setAddress("china");  user.setEmail("jack@hoo.com");  users.put("A", user);    user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName("tom");  user.setAddress("china");  user.setEmail("tom@hoo.com");  users.put("B", user);  **return** users;  }    **public** **void** setUsers(List<User> users) {  **for** (User u : users) {  System.*out*.println(u);  }  }  } |

3、编写wsdd文本发布当前WebService，wsdd文件代码如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <deployment xmlns=*"http://xml.apache.org/axis/wsdd/"*  xmlns:java=*"http://xml.apache.org/axis/wsdd/providers/java"*>  <service name=*"ManagerUser"* provider=*"java:RPC"*>  <parameter name=*"className"* value=*"com.hoo.service.ManagerUser"* />  <parameter name=*"allowedMethods"* value=*"\*"* />  <parameter name=*"scope"* value=*"request"* />  <!-- 这里的urn:User和new QName("urn:User", "User")这里是对应的 -->  <beanMapping qname=*"myNSD:User"* xmlns:myNSD=*"urn:User"* languageSpecificType=*"java:com.hoo.entity.User"*/>  </service>  </deployment> |

和上面的wsdd文件不同的是，这里多了beanMapping的配置。Qname是xml中规范限定名称，由命名空间url、本地名称、前缀组成，本地名称是必需的且qname是不可变的。后面的myNSD（namespace）是必需的，是通过前面的qname指定的，myNSD后面的urn：User就和程序中的QName对应的第一个参数，第二个参数对应配置文件中的qname的myNSD冒号后面的值。languageSpecificType是javaBean的类路径。

4、发布WebService，还是允许命令行：

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployComplex.wsdd

发布后，在浏览器中请求：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

就可以看到发布的WebService和暴露的方法了。

5、客户端代码的编写，这次客户端和以前有些不一样。因为这里设置了参数和返回值都是复杂类型，而不是简单的类型了。代码如下：

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** java.util.Date;  **import** java.util.List;  **import** java.util.Map;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ParameterMode;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **import** org.apache.axis.encoding.XMLType;  **import** org.apache.axis.encoding.ser.BeanDeserializerFactory;  **import** org.apache.axis.encoding.ser.BeanSerializerFactory;  **import** com.hoo.entity.User;  /\*\*  \* <b>function:</b>axis WebService传递复杂对象，客户端  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 10:32:57 PM  \* **@file** UserServiceClient.java  \* **@package** com.hoo.client  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @SuppressWarnings("unchecked")  **public** **class** UserServiceClient {    **private** **static** **final** String *url* = "http://localhost:8080/AxisWebService/services/ManagerUser";  **private** **static** Service *service* = **new** Service();    **public** **static** User getUser(String name) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  QName qn = **new** QName("urn:User", "User");  //call.registerTypeMapping(User.class, qn, new BeanSerializerFactory(User.class, qn), new BeanDeserializerFactory(User.class, qn));  call.registerTypeMapping(User.**class**, qn, BeanSerializerFactory.**class**, BeanDeserializerFactory.**class**);  call.setOperationName("getUserByName");  /\*  \* 这里客户端和服务器端共用一个User，在实际开发中  \* 客户端和服务器端在不同的机器上，所以User对象可能不一样  \* 需要我们根据WebService的wsdl分析对象的属性  \*/  call.addParameter("name", XMLType.*XSD\_STRING*, ParameterMode.*IN*);  call.setReturnClass(User.**class**);  User user = (User) call.invoke(**new** Object[] { "jackson" });  **return** user;  }    **public** **static** **void** setUser(User user) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("setUser");  //设置参数类型  call.addParameter("user", qn, ParameterMode.*IN*);  call.invoke(**new** Object[] { user });  }    **public** **static** List<User> getUsers(**int** i) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("getUsers");  call.addParameter("i", XMLType.*XSD\_INT*, ParameterMode.*IN*);  call.setReturnClass(List.**class**);  List<User> users = (List<User>) call.invoke(**new** Object[] { i });  **return** users;  }    **public** **static** **void** setUsers(List<User> users) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("setUsers");  call.addParameter("users", XMLType.*XSD\_ANYTYPE*, ParameterMode.*IN*);  call.invoke(**new** Object[] { users });  }    **public** **static** **void** setUserMap(Map<String, User> maps) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("setUserMap");  call.addParameter("maps", XMLType.*XSD\_ANYTYPE*, ParameterMode.*IN*);  call.invoke(**new** Object[] { maps });  }    **public** **static** Map<String, User> getUserMap() **throws** RemoteException, ServiceException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("getUserMap");  //call.addParameter("null", XMLType.XSD\_ANYTYPE, ParameterMode.IN);  call.setReturnClass(Map.**class**);  Map<String, User> maps = (Map<String, User>) call.invoke(**new** Object[]{});  **return** maps;  }    **public** **static** User[] getUserArray(**int** i) **throws** ServiceException, RemoteException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("getUserArray");  call.addParameter("i", XMLType.*XSD\_INT*, ParameterMode.*IN*);  call.setReturnClass(User[].**class**);  User[] users = (User[]) call.invoke(**new** Object[] { i });  **return** users;  }    **public** **static** **void** setUserArray(User[] users) **throws** RemoteException, ServiceException {  Call call = (Call) *service*.createCall();  call.setTargetEndpointAddress(*url*);  //这里的QName的ns和wsdd文件中的对应  QName qn = **new** QName("urn:User", "User");  //这里是将对象序列化和反序列化的配置  call.registerTypeMapping(User.**class**, qn, **new** BeanSerializerFactory(User.**class**, qn), **new** BeanDeserializerFactory(User.**class**, qn));  call.setOperationName("setUserArray");  call.addParameter("users", XMLType.*XSD\_ANYTYPE*, ParameterMode.*IN*);  call.invoke(**new** Object[] { users });  }    @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) **throws** RemoteException, ServiceException {  User user = **new** User();  user.setId(**new** Date().~~getSeconds~~());  user.setName("tom");  user.setAddress("china");  user.setEmail("tom@hoo.com");  System.*out*.println("============getUser=============");  System.*out*.println(*getUser*("jack"));  System.*out*.println("============setUser=============");  *setUser*(user);//看服务器端控制台  System.*out*.println("============getUsers List=============");  List<User> users = *getUsers*(3);  **for** (User u : users) {  System.*out*.println(u);  }  System.*out*.println("============setUsers List=============");  *setUsers*(users);  System.*out*.println("============getUserMap=============");  Map<String, User> map = *getUserMap*();  System.*out*.println(map);  System.*out*.println("============setUserMap=============");  *setUserMap*(map);  System.*out*.println("============getUserArray=============");  User[] arr = *getUserArray*(3);  System.*out*.println("============setUserArray=============");  *setUserArray*(arr);  }  } |

需要说明的是，这里传递的User和服务器端的User是同一个对象，在实际开发中应该不是同一个对象的。如java调用.net的WebService，传递的对象应该不是同一个。需要我们自己根据wsdl文件中的描述创建JavaBean。注意的是对象一个要序列化。

new QName("urn:User", "User")这里的QName的urn:User和wsdd文件中的对应，参数User也是和wsdd文件中的qname=*"myNSD:User"*对应的

这里是将对象序列化和反序列化的配置

call.registerTypeMapping(User.class, qn, new BeanSerializerFactory(User.class, qn), new BeanDeserializerFactory(User.class, qn));

User是Mapping的对象，也就是更加wsdl中参数描述创建的Java对象

call.addParameter("i", XMLType.XSD\_INT, ParameterMode.IN);设置参数信息，ParameterMode.IN代表传递的参数，XMLType.XSD\_INT,代表参数的类型，前面的i是参数名称

call.setReturnClass(User[].class);设置返回值的类型

特别说明：

1. 如果你调用的方法有返回值，一定要设置返回值的类型。call.setReturnClass
2. 如果你调用的方法有参数，一定要设置参数的类型call.addParameter
3. 记得添加wsdl4j.jar，序列化转换的时候需要用到，否则会出现找不到类型异常

运行后，效果如下：

客户端：

============getUser=============

41#jackson#jackson@hoo.com#china

============setUser=============

- No returnType was specified to the Call object! You must call setReturnType() if you have called addParameter().

============getUsers List=============

41#jack#0#jack0@hoo.com#china

41#jack#1#jack1@hoo.com#china

41#jack#2#jack2@hoo.com#china

41#jack#3#jack3@hoo.com#china

============setUsers List=============

- No returnType was specified to the Call object! You must call setReturnType() if you have called addParameter().

============getUserMap=============

{A=41#jack##jack@hoo.com#china, B=41#tom#tom@hoo.com#china}

============setUserMap=============

- No returnType was specified to the Call object! You must call setReturnType() if you have called addParameter().

============getUserArray=============

============setUserArray=============

- No returnType was specified to the Call object! You must call setReturnType() if you have called addParameter().

服务器端：

37#tom#tom@hoo.com#china

37#jack#0#jack0@hoo.com#china

37#jack#1#jack1@hoo.com#china

37#jack#2#jack2@hoo.com#china

37#jack#3#jack3@hoo.com#china

{A=37#jack##jack@hoo.com#china, B=37#tom#tom@hoo.com#china}

37#jack#0#jack0@hoo.com#china

37#jack#1#jack1@hoo.com#china

[37#jack#2#jack2@hoo.com#china](mailto:37#jack)

Map、Array、List、JavaBean都可以顺利转换，不管是参数还是返回值。怎么样，很简单吧！

**传递内部类**

如果你传递的JavaBean带有内部类，那么你的内部类需要是静态化的，且一样要序列化。为Account对象添加一个属性Birthday，Birthday是一个内部类，

|  |
| --- |
| **package** com.hoo.entity;  **import** java.io.Serializable;  /\*\*  \* <b>function:</b>Account Entity  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 10:18:20 PM  \* **@file** Account.java  \* **@package** com.hoo.entity  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** Account {  **private** **int** id;  **private** String name;  **private** String email;  **private** String address;  **private** Birthday birthday;    **public** Birthday getBrithday() {  **return** birthday;  }  **public** **void** setBrithday(Birthday birthday) {  **this**.birthday = birthday;  }    **//getter、setter**    @Override  **public** String toString() {  **return** **this**.id + "#" + **this**.name + "#" + **this**.email + "#" + **this**.address + "#" + **this**.birthday.getBirthday();  }    /\*\*  \* axis中传的的内部类一定要是静态的，且要序列化  \*/  **public** **static** **class** Birthday **implements** Serializable {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** String bitthday;  // **getter、setter**  }  } |

同样我们需要在客户端调用的时候，注册类型序列化和反序列化。

QName qn = **new** QName("urn:Account", "entity.Account");

call.registerTypeMapping(Account.**class**, qn, **new** BeanSerializerFactory(Account.**class**, qn), **new** BeanDeserializerFactory(Account.**class**, qn));

QName qn2 = **new** QName("urn:Birthday", "Account.Birthday");

call.registerTypeMapping(Birthday.**class**, qn2, **new** BeanSerializerFactory(Birthday.**class**, qn2), **new** BeanDeserializerFactory(Birthday.**class**, qn2));

而在wsdd文件中和刚才几乎一样

<beanMapping qname="myNSD:entity.Account" xmlns:myNSD="urn:Account" languageSpecificType="java:com.hoo.entity.Account"/>

<beanMapping qname="myNSD:Account.Birthday" xmlns:myNSD="urn:Birthday" languageSpecificType="java:com.hoo.entity.Account$Birthday"/>*com.hoo.* com.hoo.entity.Account$Birthday是内部类的表示方式

**七、axis调用RMI模式WebService**

和先前写的web service几乎一样，不同的是这里写的WebService要实现接口，而我们实现的接口要继承rmi的Remote接口才行。

1、RMI接口，很简单的2个方法，继承Remote接口

|  |
| --- |
| **package** com.hoo.rmi;  **import** java.rmi.Remote;  /\*\*  \* <b>function:</b>用axis实现RMI调用  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 16:22:25 PM  \* **@file** HelloRMI.java  \* **@package** com.hoo.rmi  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** HelloRMI **extends** Remote {  **public** String sayHello(String name);    **public** **int** getAge(**int** age);  } |

2、实现上面的接口

|  |
| --- |
| **package** com.hoo.rmi.impl;  **import** com.hoo.rmi.HelloRMI;  /\*\*  \* <b>function:</b>实现RMI服务器接口  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 16:23:15 PM  \* **@file** HelloRMIImpl.java  \* **@package** com.hoo.rmi.impl  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloRMIImpl **implements** HelloRMI {  **public** **int** getAge(**int** age) {  **return** age + 10;  }  **public** String sayHello(String name) {  **return** name + " say: hello rmi WebService !";  }  } |

3、定制wsdd发布服务

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <deployment xmlns=*"http://xml.apache.org/axis/wsdd/"*  xmlns:java=*"http://xml.apache.org/axis/wsdd/providers/java"*>  <service name=*"HelloRMI"* provider=*"java:RPC"*>  <parameter name=*"className"* value=*"com.hoo.rmi.impl.HelloRMIImpl"* />  <parameter name=*"allowedMethods"* value=*"getAge,sayHello"* />  <parameter name=*"scope"* value=*"request"* />  </service>  </deployment> |

4、命令行发布服务

java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployRMI.wsdd

在浏览器中输入：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

即可查看到刚才发布的服务

5、编写客户端调用代码

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **import** com.hoo.rmi.HelloRMI;  /\*\*  \* <b>function:</b>axis WebService的RMI调用模式  \* **@author** hoojo  \* **@createDate** Dec 16, 2010 16:28:09 PM  \* **@file** HelloRMIClient.java  \* **@package** com.hoo.client  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloRMIClient {  **public** **static** **void** main(String[] args) **throws** ServiceException, RemoteException {  String url = "http://localhost:8080/AxisWebService/services/HelloRMI";  //axis call调用  Service service = **new** Service();  Call call = (Call) service.createCall();  call.setTargetEndpointAddress(url);  call.setOperationName(**new** QName(url, "sayHello"));  String result = (String) call.invoke(**new** Object[] { "jack" });  System.*out*.println(result);    //远程接口模式，只管接口不管实现，这就是面向接口的好处  HelloRMI rmi = (HelloRMI) service.getPort(url, HelloRMI.**class**);  System.*out*.println(rmi.getAge(22));  }  } |

HelloRMI rmi = (HelloRMI) service.getPort(url, HelloRMI.**class**);

用rmi接口调用的还是比较简单明了，接口中的方法都一目了然。我们也不用操心接口里面的方法是怎么样实现的，更具有面向接口编程、更加透明化。

**八、WebService处理异常信息**

在数据传递和远程调用过程中，有时候调用者传递错误的数据和是调用的方法错误。那我们怎样处理这个错误？方法有很多，这里介绍下用异常处理远程调用的错误方法。

1、编写服务器端的异常类，在远处调用中（WebService）异常类继承RemoteException，记得要序列化下，代码如下：

|  |
| --- |
| **package** com.hoo.exception;  **import** java.rmi.RemoteException;  /\*\*  \* <b>function:</b>定制服务器端异常信息  \* **@author** hoojo  \* **@createDate** Dec 17, 2010 00:00:52 AM  \* **@file** RemoteServerException.java  \* **@package** com.hoo.exception  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** RemoteServerException **extends** RemoteException {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** String message;    **public** String getMessage() {  **return** message;  }  **public** **void** setMessage(String message) {  **this**.message = message;  }  **public** RemoteServerException() {  System.*out*.println("Remote Servier Exception ");  }    **public** **void** showMessage() {  System.*out*.println(**this**.message);  }  } |

2、编写客户端的异常信息类，和前面远程传递对象一样。因为我们并不知道服务器端的异常信息类代码，但是通过wsdl的xml文件的描述我们可以知道远程异常类的基本信息：方法、方法参数、返回值等信息，下面是客户端的异常信息类代码：

|  |
| --- |
| **package** com.hoo.exception;  **import** java.rmi.RemoteException;  /\*\*  \* <b>function:</b>本地客户端异常信息  \* **@author** hoojo  \* **@createDate** Dec 17, 2010 00:05:30 AM  \* **@file** LocalClientException.java  \* **@package** com.hoo.exception  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** LocalClientException **extends** RemoteException {  **private** **static** **final** **long** *serialVersionUID* = 3;  **private** String message;  **public** String getMessage() {  **return** message;  }  **public** **void** setMessage(String message) {  **this**.message = message;  }  **public** LocalClientException() {  System.*out*.println("Local Client Exception ");  }    **public** **void** showMessage() {  System.*out*.println(**this**.message);  }  } |

内容几乎一样，就是类名称不同。

3、编写触发异常的WebService服务器端代码

|  |
| --- |
| **package** com.hoo.service;  **import** com.hoo.exception.RemoteServerException;  /\*\*  \* <b>function:</b>发出异常信息  \* **@author** hoojo  \* **@createDate** Dec 17, 2010 00:08:07 AM  \* **@file** ThrowException.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ThrowException {    **public** **void** doException() **throws** RemoteServerException {  RemoteServerException rse = **new** RemoteServerException();  rse.setMessage("服务器端出现异常");  **throw** rse;  }  } |

4、定制wsdd文件，发布当前WebService。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <deployment xmlns="http://xml.apache.org/axis/wsdd/"  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  <service name="ThrowException" provider="java:RPC">  <parameter name="className" value="com.hoo.service.ThrowException" />  <parameter name="allowedMethods" value="\*" />  <parameter name="scope" value="Session" />  <operation name="doException" qname="operNS:doException" xmlns:operNS="doException">  <fault name="RemoteServerExceptionFault" qname="fut:fault" xmlns:fut="RemoteServerExceptionFault" type="tns:RemoteServerException" xmlns:tns="RemoteServerException"/>  </operation>  <typeMapping qname="myNSD:Exception" xmlns:myNSD="ns:CustomException" languageSpecificType="java:com.hoo.exception.RemoteServerException" serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"  deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>  </service>  </deployment> |

特别说明，这里多了3个xml标签元素。operation、fault、typeMapping，下面将依次介绍。

operation看看这个标签中的内容，就和我们最先看到的wsdl的xml文件中的方法标签很像。没错这里它指定的就是一个方法，operation就指定触发异常信息的方法。和你在服务器端触发异常的WebService的方法对应，qname是限定名称，xmlns是限定名称的命名空间。

fault的name一个名称，可以随便取。type就是你服务器端抛出异常的类型，这里服务器端抛出的RemoteServerException，所以类型就是RemoteServerException。如果你服务器端抛出的是NullPointException这里的type就是NullPointException。

typeMapping这个也很关键，和前面的beanMapping有几分相似特别是前面一段qname、xmlns、以及languageSpecificType都是一样的，用法也一样。分别和客户端的new QName("ns:CustomException", "Exception");中的对应；其中serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"就是序列化的工厂类，deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"这个不说你也知道是反序列化的工厂类，encodingStyle这个是编码样式。

5、用定制好的wsdd文件发布我们的WebService，依旧是命令行：

C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployException.wsdd

发布完成后，在浏览器中输入：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

即可查看刚才发布的服务信息，但是你点击wsdl链接的时候并不能看到xml的内容，那是因为服务器端抛出了异常导致的。这个不是错误是正常的，不过你可以在web.xml中配置异常的信息页面。

6、编写客户端代码，这里的代码和传递对象的WebService也很类似

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **import** org.apache.axis.encoding.ser.BeanDeserializerFactory;  **import** org.apache.axis.encoding.ser.BeanSerializerFactory;  **import** com.hoo.exception.LocalClientException;  **import** com.hoo.exception.RemoteServerException;  **public** **class** TryExceptionClient {  /\*\*  \* <b>function:</b>捕捉服务器端异常信息的WebService的客户端  \* **@author** hoojo  \* **@createDate** Dec 17, 2010 00:12:56 AM  \* **@param** args  \* **@throws** ServiceException  \* **@throws** RemoteException  \*/  **public** **static** **void** main(String[] args) {  String url = "http://localhost:8080/AxisWebService/services/ThrowException";  Service service = **new** Service();  **try** {  Call call = (Call) service.createCall();    /\*\*  \* 注册异常类信息和序列化类  \* ns:CustomException 和 wsdd 配置文件中的typeMapping中的xmlns:myNSD="ns:CustomException"的对应  \* Exception 和 wsdd 配置文件中的typeMapping中的qname="myNSD:Exception"的Exception对应  \*/  QName qn = **new** QName("ns:CustomException", "Exception");  /\*\*  \* 这里配置的LocalClientException，会将服务器端的RemoteServerException转换成本地的异常信息LocalClientException  \*/  call.registerTypeMapping(LocalClientException.**class**, qn,  **new** BeanSerializerFactory(LocalClientException.**class**,qn),  **new** BeanDeserializerFactory(LocalClientException.**class**, qn));  call.setOperationName(**new** QName(url, "doException"));    call.setTargetEndpointAddress(url);  call.invoke(**new** Object[]{});  } **catch** (RemoteServerException e) {  e.showMessage();  System.*out*.println("RemoteServerException:" + e.getMessage());  e.printStackTrace();  } **catch** (LocalClientException e) {  e.showMessage();  System.*out*.println("LocalClientException:" + e.getMessage());  e.printStackTrace();  } **catch** (RemoteException e) {  System.*out*.println("RemoteException:" + e.getMessage());  e.printStackTrace();  } **catch** (ServiceException e) {  System.*out*.println("ServiceException:" + e.getMessage());  e.printStackTrace();  }  }  } |

看上面的registerTypeMapping和wsdd文件中的typeMapping有些相同之处吧，下面运行下上面的代码，结果如下：

|  |
| --- |
| Local Client Exception  服务器端出现异常  LocalClientException:服务器端出现异常  com.hoo.exception.LocalClientException: 服务器端出现异常  at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)  at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)  at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)  at java.lang.reflect.Constructor.newInstance(Unknown Source)  at java.lang.Class.newInstance0(Unknown Source)  at java.lang.Class.newInstance(Unknown Source)  at org.apache.axis.encoding.ser.BeanDeserializer.<init>(BeanDeserializer.java:104)  at org.apache.axis.encoding.ser.BeanDeserializerFactory.getGeneralPurpose(BeanDeserializerFactory.java:89)  at org.apache.axis.encoding.ser.BaseDeserializerFactory.getDeserializerAs(BaseDeserializerFactory.java:89)  at org.apache.axis.encoding.DeserializationContext.getDeserializer(DeserializationContext.java:464)  at org.apache.axis.encoding.DeserializationContext.getDeserializerForType(DeserializationContext.java:547)  at org.apache.axis.message.SOAPFaultDetailsBuilder.onStartChild(SOAPFaultDetailsBuilder.java:157)  at org.apache.axis.encoding.DeserializationContext.startElement(DeserializationContext.java:1035)  at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.startElement(Unknown Source)  at com.sun.org.apache.xerces.internal.impl.XMLNSDocumentScannerImpl.scanStartElement(Unknown Source)  at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl$FragmentContentDispatcher.dispatch(Unknown Source)  at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanDocument(Unknown Source)  at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)  at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)  at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown Source)  at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.parse(Unknown Source)  at javax.xml.parsers.SAXParser.parse(Unknown Source)  at org.apache.axis.encoding.DeserializationContext.parse(DeserializationContext.java:227)  at org.apache.axis.SOAPPart.getAsSOAPEnvelope(SOAPPart.java:696)  at org.apache.axis.Message.getSOAPEnvelope(Message.java:435)  at org.apache.axis.handlers.soap.MustUnderstandChecker.invoke(MustUnderstandChecker.java:62)  at org.apache.axis.client.AxisClient.invoke(AxisClient.java:206)  at org.apache.axis.client.Call.invokeEngine(Call.java:2784)  at org.apache.axis.client.Call.invoke(Call.java:2767)  at org.apache.axis.client.Call.invoke(Call.java:2443)  at org.apache.axis.client.Call.invoke(Call.java:2366)  at org.apache.axis.client.Call.invoke(Call.java:1812)  at com.hoo.client.TryExceptionClient.main(TryExceptionClient.java:44) |

服务器端控制台会输出：Remote Servier Exception

这是在创建异常的时候，在构造函数中输出的。

**九、WebService中文件传输**

WebService处理传递普通的信息，还可以传输文件，下面介绍WebService是怎么完成文件传输的。

1. 首先编写服务器端上传文件的WebService方法

|  |
| --- |
| **package** com.hoo.service;  **import** java.io.File;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.InputStream;  **import** javax.activation.DataHandler;  /\*\*  \* <b>function:</b>Axis WebService完成文件上传服务器端  \* **@author** hoojo  \* **@createDate** Dec 18, 2010 1:16:16 PM  \* **@file** UploadFileService.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** UploadFileService {    /\*\*  \* <b>function:</b>传递文件  \* **@author** hoojo  \* **@createDate** Dec 18, 2010 1:27:58 PM  \* **@param** handler DataHandler这个参数必须  \* **@param** fileName 文件名称  \* **@return** upload Info  \*/  **public** String upload(DataHandler handler, String fileName) {  **if** (fileName != **null** && !"".equals(fileName)) {  File file = **new** File(fileName);  **if** (handler != **null**) {  InputStream is = **null**;  FileOutputStream fos = **null**;  **try** {  is = handler.getInputStream();  fos = **new** FileOutputStream(file);  **byte**[] buff = **new** **byte**[1024 \* 8];  **int** len = 0;  **while** ((len = is.read(buff)) > 0) {  fos.write(buff, 0, len);  }  } **catch**(FileNotFoundException e) {  **return** "fileNotFound";  } **catch** (Exception e) {  **return** "upload File failure";  } **finally** {  **try** {  **if** (fos != **null**) {  fos.flush();  fos.close();  }  **if** (is != **null**) {  is.close();  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  **return** "file absolute path:" + file.getAbsolutePath();  } **else** {  **return** "handler is null";  }  } **else** {  **return** "fileName is null";  }  }  } |

上传方法和我们以前在Web中上传唯一不同的就是参数一DataHandler，可以将这类看成文件传输器，他可以把文件序列化。然后通过DataHandler可以得到一个输入流InputStream，通过这个流可以读到文件的内容。其他的操作和普通上传类似。

1. 定制wsdd发布文件上传的WebService服务

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <deployment xmlns="http://xml.apache.org/axis/wsdd/"  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  <service name="UploadFile" provider="java:RPC">  <parameter name="className" value="com.hoo.service.UploadFileService" />  <parameter name="allowedMethods" value="\*" />  <parameter name="scope" value="Session" />  <!-- 和服务器端上传文件的方法签名对应，参数也对应 -->  <operation name="upload" qname="operNS:upload" xmlns:operNS="upload" returnType="rns:string"  xmlns:rns="http://www.w3.org/2001/XMLSchema">  <parameter name="handler" type="ns:DataHandler" xmlns:ns="http://www.w3.org/2001/XMLSchema"/>  <parameter name="fileName" type="ns:string" xmlns:ns="http://www.w3.org/2001/XMLSchema"/>  </operation>  <typeMapping qname="hns:DataHandler" xmlns:hns="ns:FileUploadHandler"  languageSpecificType="java:javax.activation.DataHandler"  serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory" deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>  </service>  </deployment> |

上面才xml节点元素在前面都见过了，说明下operation中的参数，注意要指定参数类型，特别是DataHandler的类型，然后就是typeMapping的serializer、deserializer的序列化和反序列化工厂类的配置。

1. 用dos命令发布当前WebService

C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployUpload.wsdd

发布完成后，可以通过这个地址查看uploadFile这个service了

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

1. 编写客户端代码

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** javax.activation.DataHandler;  **import** javax.activation.FileDataSource;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ParameterMode;  **import** javax.xml.rpc.ServiceException;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **import** org.apache.axis.encoding.XMLType;  **import** org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory;  **import** org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory;  /\*\*  \* <b>function:</b>上传文件WebService客户端  \*  \* **@author** hoojo  \* **@createDate** Dec 18, 2010 1:38:14 PM  \* **@file** UploadFileClient.java  \* **@package** com.hoo.client  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** UploadFileClient {  **public** **static** **void** main(String[] args) **throws** ServiceException, RemoteException {  String url = "http://localhost:8080/AxisWebService/services/UploadFile";  String fileName = "readMe.txt";  String path = System.*getProperty*("user.dir") + "\\WebRoot\\" + fileName;  System.*out*.println(path);    //这样就相当于构造了一个带文件路径的File了  DataHandler handler = **new** DataHandler(**new** FileDataSource(path));    Service service = **new** Service();  Call call = (Call) service.createCall();  call.setTargetEndpointAddress(url);    /\*\*  \* 注册异常类信息和序列化类 ns:FileUploadHandler 和 wsdd 配置文件中的typeMapping中的xmlns:hns="ns:FileUploadHandler" 的对应 DataHandler  \* 和 wsdd 配置文件中的typeMapping中的qname="hns:DataHandler"的DataHandler对应  \*/  QName qn = **new** QName("ns:FileUploadHandler", "DataHandler");  call.registerTypeMapping(DataHandler.**class**, qn,  JAFDataHandlerSerializerFactory.**class**,  JAFDataHandlerDeserializerFactory.**class**);  call.setOperationName(**new** QName(url, "upload"));    //设置方法形参，注意的是参数1的type的DataHandler类型的，和上面的qn的类型是一样的  call.addParameter("handler", qn, ParameterMode.*IN*);  call.addParameter("fileName", XMLType.*XSD\_STRING*, ParameterMode.*IN*);  //设置返回值类型，下面2种方法都可以  call.setReturnClass(String.**class**);  //call.setReturnType(XMLType.XSD\_STRING);    String result = (String) call.invoke(**new** Object[] { handler, "remote\_server\_readMe.txt" });  System.*out*.println(result);  }  } |

至此，文件传输就完成了。怎么样，还不错吧！

如果你用myEclipse进行开发的话，运行时可能会出现以下的错误:

Exception in thread "main" java.lang.NoClassDefFoundError: com/sun/mail/util/LineInputStream

原因是jar包版本不统一,解决方法如下:

删除Java EE 5 Libraries/javaee.jar/mail里的包有东西.

具体方法如下:

用rar打开X:/Program Files/MyEclipse 6.0/myeclipse/eclipse/plugins/com.genuitec.eclipse.j2eedt.core\_6.0.1.zmyeclipse601200710/data/libraryset/EE\_5/javaee.jar,然后删除mail,一切就ok了.

**十、Axis WebService常用命令和调试工具**

上面我们在发布wsdd服务的时候都会在dos窗口写上一段命令，下面还介绍下其他的几个命令：

1. **AdminClient命令**

java org.apache.axis.client.AdminClient c:/deploy.wsdd

这样就可以用AdminClient发布一个WebService了，如果你对AdminClient的参数信息不是很了解输入：java org.apache.axis.client.AdminClient就可以查看帮助了。

|  |
| --- |
| Usage: AdminClient [Options] [list | <deployment-descriptor-files>]  Processes a set of administration commands.  The following Options are available:  -l<url> sets the AxisServlet URL  -h<hostName sets the AxisServlet host  -p<portNumber> sets the AxisServlet port  -s<servletPath> sets the path to the AxisServlet  -f<fileName> specifies that a simple file protocol should be used  -u<username> sets the username  -w<password> sets the password  -d sets the debug flag (for instance, -ddd would set it to  3)  -t<name> sets the transport chain touse  Commands:  list will list the currently deployed services  quit will send a quit message to SimpleAxisServer  passwd value changes the admin password  Deployment Descriptor files:  <deployment-descriptor-files> deploys or undeploys Axis components and  web services described in these files  If -l or -h -p -s are not set, the AdminClient will invoke  http://localhost:8080/axis/servlet/AxisServlet |

帮助信息如上所示，如果你有下载axis的源码，可以看看AdminClient的源码就知道是怎么回事了。如果上述命令找不到AdminClient，请设置dirs

java org.apache.axis.client.AdminClient c:/deploy.wsdd –p8081

上面的-p8081设置端口信息，默认AdminClient会使用8080端口。当你的端口换成其他的可以用-p设置端口

java org.apache.axis.client.AdminClient

-lhttp://localhost:8080/axiswebProject/services/AdminService deploy.wsdd

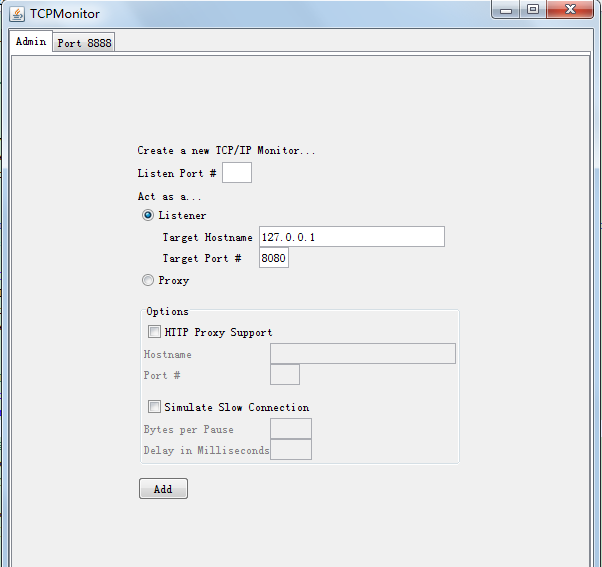
通过-l指定你的AdminService的url路径完成发布wsdd，这个很有效

1. **调试工具，监听WebService的通讯信息**

在dos窗口输入：

C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>java -Djava.ext.dirs=lib org.apache.axis.utils.tcpmon

org.apache.axis.utils.tcpmon是一个端口监听的工具类，它可以监听到指定host、port的WebService通信信息



在listen port填写你要监听的端口即可，是监听的端口不实服务器的端口。也就是你请求服务器的端口。

1. **查看发布的WebService服务信息**

通过在WebBrowser地址栏输入：

[http://localhost:8080/[Project]/servlet/AxisServlet](http://localhost:8080/%5bProject%5d/servlet/AxisServlet)

可以查看你发表过的WebService，Project是你的axis的web工程

**十一、用工具生成代码**

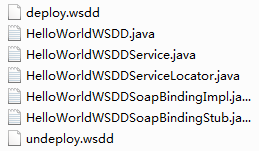
1. 用Axis提供的WSDL2java工具类生成代码，命令如下：

C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>java -Djava.ext.dirs=lib org.apache.axis.wsdl.WSDL2Java -s <http://localhost:8080/AxisWebService/services/HelloWorldWSDD?wsdl> –o c:\

-s 是设置wsdl的路径url和是当前物理硬盘上的wsdl文件（可以将页面上的wsdl的xml文件另存为一份试试，也是可以的）

-o 是设置转换的文件的输出目录

上面的命令会生产服务器端的实现代码和客户端的调用代码。代码文件如下：



deploy、undeploy是发布、卸载WebService的wsdd文件

HelloWorldWSDD是服务器端代码接口文件，继承了Remote接口，定义了getAge、sayHello方法

HelloWorldWSDDService是获得当前Service实例的接口

HelloWorldWSDDServiceLocator实现了HelloWorldWSDDService接口，客户端在调用当前WebService时候有用

HelloWorldWSDDSoapBindingImpl当前WebService接口HelloWorldWSDD的实现，需要自己实现方法的业务

HelloWorldWSDDSoapBindingStub看结尾的Stub就知道，这个可以完成客户端的实现，用它就可以调用到WebService的方法

下面用WSDL2Java生成的代码调用WebService，注意你还得用dos命令将上面的服务通过deploy.wsdd发布出去。下面是调用代码：

|  |
| --- |
| **package** localhost.AxisWebService.services.HelloWorldWSDD.clientimpl;  **import** java.net.MalformedURLException;  **import** java.net.URL;  **import** java.rmi.RemoteException;  **import** javax.xml.rpc.ServiceException;  **import** localhost.AxisWebService.services.HelloWorldWSDD.HelloWorldWSDDServiceLocator;  **import** localhost.AxisWebService.services.HelloWorldWSDD.HelloWorldWSDDSoapBindingStub;  **public** **class** Client {  **public** **static** **void** main(String[] args) **throws** ServiceException, RemoteException, MalformedURLException {  String target = "http://localhost:8080/AxisWebService/services/HelloWorldWSDD";  HelloWorldWSDDServiceLocator service = **new** HelloWorldWSDDServiceLocator();  HelloWorldWSDDSoapBindingStub stub = **new** HelloWorldWSDDSoapBindingStub(**new** URL(target), service);    System.*out*.println(stub.getAge(22));  System.*out*.println(stub.getName("张三"));  }  } |

用自动生成的代码完成客户端调用比较简单，如果你的WebService的实现版本是JAX-WS2.0就可以用MyEclipse的WebService插件工具帮助我们生产代码，如：axis2就可以，下次会介绍Axis2.x的用法。

**十二、用Axis操作 Header头部信息**

Axis中操作Header的信息是通过Handler来完成的，在读取Header的信息的时候，可以完成Header的操作的Handler有JWSHandler、SOAPMonitorHandler、JAXRPCHandler。JWSHandler是完成jws发布模式的WebService的Header的读取操作，

SOAPMonitorHandler是完成SOAPMonitor模式的，JAXRPCHandler是java rpc模式的。

下面将会用JAXRPCHandler读取客户端的Header信息，然后用BasicHandler将客户端请求的Header写入客户端的响应的Header中。

1. 首先编写一个方法，这个方法什么也不做。就是在这个方法请求和响应的时候分别完成读取Header的信息和写入响应的Header内容。代码如下：

|  |
| --- |
| **package** com.hoo.service;  /\*\*  \* <b>function:</b>读取客户端头部Header信息的WebService  \* **@author** hoojo  \* **@createDate** Dec 19, 2010 11:33:25 AM  \* **@file** ReadHeaderService.java  \* **@package** com.hoo.service  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** OperaterHeaderService {  /\*\*  \* <b>function:</b>提供一个空方法，在调用这个方法的时候  \* 触发ReadHeaderHandler读取Header的信息  \* **@author** hoojo  \* **@createDate** Dec 19, 2010 11:33:54 AM  \*/  **public** **void** readHeader() {  }  } |

1. 客户端的Header中的内容是下面这样的xml内容

|  |
| --- |
| <ns1:Header soapenv:mustUnderstand="0" xmlns:ns1="soap">  <ServiceName>ReadHeaderService</ServiceName>  <CreateDate>2010-12-19T06:22:59.931Z</CreateDate>  <ns2:Version xmlns:ns2="service">1.0</ns2:Version>  <author>  <name>axis</name>  <age>14</age>  </author>  <copyRight>  <year>2010</year>  <number>123123123</number>  </copyRight>  <Tel>555666442</Tel>  </ns1:Header> |

下面将用客户端代码将上面的xml元素添加的Header中，代码如下：

|  |
| --- |
| /\*\* 添加请求的header \*/  //父节点，根节点  SOAPHeaderElement el = **new** SOAPHeaderElement(**new** QName("soap", "Header"));  //文本节点  MessageElement msgEl = **new** MessageElement(**new** QName("ServiceName"), "ReadHeaderService");  el.addChild(msgEl);    //文本节点  el.addChild(**new** MessageElement(**new** QName("CreateDate"), **new** Date()));  el.addChild(**new** MessageElement(**new** QName("service", "Version"), "1.0"));  //子元素  msgEl = **new** MessageElement(**new** QName("author"));  msgEl.addChild(**new** MessageElement(**new** QName("name"), "axis"));  msgEl.addChild(**new** MessageElement(**new** QName("age"), "14"));  el.addChild(msgEl);  //子元素  msgEl = **new** MessageElement(**new** QName("copyRight"));  msgEl.addChild(**new** MessageElement(**new** QName("year"), "2010"));  msgEl.addChild(**new** MessageElement(**new** QName("number"), "123123123"));  el.addChild(msgEl);  el.addChild(**new** MessageElement(**new** QName("Tel"), "555666442"));  System.*out*.println(el.toString());  //设置请求的header  call.addHeader(el); |

上面的就完成了xml中的内容，并且添加到请求的Header中。

1. 根据上面的Header的内容，我们现在编写一个Handler用来读取客户端请求信息中的Header的内容，这个Handler需要继承JAXRPCHandler。代码如下：

|  |
| --- |
| **package** com.hoo.service.handler;  **import** java.util.Iterator;  **import** java.util.List;  **import** javax.xml.soap.SOAPHeader;  **import** org.apache.axis.AxisFault;  **import** org.apache.axis.MessageContext;  **import** org.apache.axis.handlers.JAXRPCHandler;  **import** org.apache.axis.message.MessageElement;  **import** org.apache.axis.message.NodeImpl;  **import** org.apache.axis.message.SOAPHeaderElement;  **import** org.apache.axis.message.Text;  /\*\*  \* <b>function:</b>读取Header信息的Handler  \* **@author** hoojo  \* **@createDate** Dec 19, 2010 11:34:56 AM  \* **@file** ReadHeaderHandler.java  \* **@package** com.hoo.service.handler  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ReadHeaderHandler **extends** JAXRPCHandler {  **private** **static** **final** **long** *serialVersionUID* = -546314372478633248L;  @SuppressWarnings("unchecked")  @Override  **public** **void** invoke(MessageContext msgContext) **throws** AxisFault {  **try** {  //头对象  SOAPHeader header = msgContext.getCurrentMessage().getSOAPHeader();  Iterator<SOAPHeaderElement> iter = header.getChildElements();  **while** (iter.hasNext()) {  //子元素  SOAPHeaderElement el = iter.next();  //继续子元素  List<NodeImpl> list = el.getChildren();  **for** (NodeImpl node : list) {  **if** (node.hasChildNodes()) {  MessageElement msg = (MessageElement) node;  **if** (msg.hasChildNodes()) {  List<NodeImpl> msgNode = msg.getChildren();  **for** (NodeImpl n : msgNode) {  **if** (!n.hasChildNodes()) {  Text text = (Text) n;  System.*out*.println("父节点：" + el.getName() + ", 节点：" + msg.getName() + ", 值：" + text.getValue());  } **else** {  MessageElement child = (MessageElement) n;  **if** (child.hasChildNodes()) {  List<NodeImpl> childNode = child.getChildren();  **for** (NodeImpl n2 : childNode) {  **if** (!n2.hasChildNodes()) {  Text text = (Text) n2;  System.*out*.print("父父节点：" + el.getName() + ", 父节点：" + msg.getName());  System.*out*.println(", 节点：" + child.getName() + "##" + child.getNodeName()  + ", 值：" + text.getValue() + "##" + child.getValue());  }  }  }  }  }  }  }  }  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

上面读取Header内容的代码有点小长，不过比较简单。就是用循环遍历Header的内容，Header的内容是一个xml的文档。所以用循环一步步向里面遍历，不过感觉写个递归会更简单。

1. 下面将编写一个Handler来向客户端的Response中写入Header内容，写入的Header的内容就是客户端请求的Header内容。代码如下：

|  |
| --- |
| **package** com.hoo.service.handler;  **import** java.util.Iterator;  **import** javax.xml.soap.SOAPException;  **import** org.apache.axis.AxisFault;  **import** org.apache.axis.Message;  **import** org.apache.axis.MessageContext;  **import** org.apache.axis.handlers.BasicHandler;  **import** org.apache.axis.message.SOAPEnvelope;  **import** org.apache.axis.message.SOAPHeader;  **import** org.apache.axis.message.SOAPHeaderElement;  /\*\*  \* <b>function:</b>用handler向客户端写入Response 的Header信息  \* **@author** hoojo  \* **@createDate** Dec 19, 2010 2:08:35 PM  \* **@file** WriteHeaderHandler.java  \* **@package** com.hoo.service.handler  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** WriteHeaderHandler **extends** BasicHandler {  **private** **static** **final** **long** *serialVersionUID* = 1L;  @SuppressWarnings("unchecked")  **public** **void** invoke(MessageContext msgContext) **throws** AxisFault {  Message msg = msgContext.getCurrentMessage();  SOAPEnvelope se = msg.getSOAPEnvelope();    //request Header的部分头信息  Message reqMsg = msgContext.getRequestMessage();  SOAPEnvelope reqSe = reqMsg.getSOAPEnvelope();  SOAPHeader reqHeader = **null**;  **try** {  reqHeader = (SOAPHeader) reqSe.getHeader();  } **catch** (SOAPException e) {  e.printStackTrace();  }    //response Header部分头信息  Message respMsg = msgContext.getResponseMessage();  SOAPEnvelope respSe = respMsg.getSOAPEnvelope();  SOAPHeader respHeader = **null**;  **try** {  respHeader = (SOAPHeader) respSe.getHeader();  } **catch** (SOAPException e) {  e.printStackTrace();  }    Iterator<SOAPHeaderElement> iter = reqHeader.examineAllHeaderElements();  **while** (iter.hasNext()) {  SOAPHeaderElement she = iter.next();  **try** {  //添加请求信息到响应的Header中  respHeader.addChildElement(she);  } **catch** (SOAPException e) {  e.printStackTrace();  }  System.*out*.println(she.getNodeName());  }  //设置响应信息  msgContext.setResponseMessage(respMsg);  **try** {  System.*out*.println("header Info:" + se.getHeader());  } **catch** (SOAPException e) {  e.printStackTrace();  }  }  } |

上面的代码是取出respHeader = (SOAPHeader) respSe.getHeader();

然后循环遍历Header的元素写入到respHeader.addChildElement(she);

1. 好了，上面已经写好了WebService的方法，以及请求时读取Header内容的Handler和在响应时写入Header内容的Handler。下面开始写wsdd发布当前WebService。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <deployment xmlns="http://xml.apache.org/axis/wsdd/"  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  <handler name="ReadHeaderHandler" type="java:com.hoo.service.handler.ReadHeaderHandler"/>  <handler name="WriteHeaderHandler" type="java:com.hoo.service.handler.WriteHeaderHandler"/>    <service name="OperaterHeader" provider="java:RPC">  <requestFlow>  <handler type="ReadHeaderHandler"/>  </requestFlow>  <parameter name="className" value="com.hoo.service.OperaterHeaderService" />  <parameter name="allowedMethods" value="\*" />  <parameter name="scope" value="request" />  <responseFlow>  <handler type="WriteHeaderHandler"/>  </responseFlow>  </service>  </deployment> |

还是用doc命令发布当前WebService，命令如下：

C:\SoftWare\tomcat-5.0.28\tomcat-5.0.28\webapps\AxisWebService\WEB-INF>java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient -lhttp://localhost:8080/AxisWebService/services/AdminService deployHeader.wsdd

发布完成后，你会看的

Processing file deployHeader.wsdd

<Admin>Done processing</Admin>

的字样就表示发布成功了，然后在浏览器地址栏输入：

<http://localhost:8080/AxisWebService/servlet/AxisServlet>

你就可以看到OperaterHandler这个WebService了

1. OperaterHandler这个WebService成功发布，现在就是要编写客户端代码调用这个WebService，客户端代码比较简单，完整代码如下：

|  |
| --- |
| **package** com.hoo.client;  **import** java.rmi.RemoteException;  **import** java.util.Date;  **import** javax.xml.namespace.QName;  **import** javax.xml.rpc.ServiceException;  **import** javax.xml.soap.SOAPException;  **import** org.apache.axis.Message;  **import** org.apache.axis.client.Call;  **import** org.apache.axis.client.Service;  **import** org.apache.axis.message.MessageElement;  **import** org.apache.axis.message.SOAPEnvelope;  **import** org.apache.axis.message.SOAPHeaderElement;  /\*\*  \* <b>function:</b>Header操作WebService客户端  \* **@author** hoojo  \* **@createDate** Dec 19, 2010 2:18:51 PM  \* **@file** OperaterHeaderClient.java  \* **@package** com.hoo.client  \* **@project** AxisWebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** OperaterHeaderClient {  **public** **static** **void** main(String[] args) **throws** SOAPException, ServiceException, RemoteException {  String url = "http://localhost:8080/AxisWebService/services/OperaterHeader";  //创建服务  Service service = **new** Service();  //创建调用句柄  Call call = (Call) service.createCall();  //设置请求地址  call.setTargetEndpointAddress(url);  call.setOperationName(**new** QName("readHeader"));    /\*\* 添加请求的header \*/  //父节点，根节点  SOAPHeaderElement el = **new** SOAPHeaderElement(**new** QName("soap", "Header"));  //文本节点  MessageElement msgEl = **new** MessageElement(**new** QName("ServiceName"), "ReadHeaderService");  el.addChild(msgEl);    //文本节点  el.addChild(**new** MessageElement(**new** QName("CreateDate"), **new** Date()));  el.addChild(**new** MessageElement(**new** QName("service", "Version"), "1.0"));  //子元素  msgEl = **new** MessageElement(**new** QName("author"));  msgEl.addChild(**new** MessageElement(**new** QName("name"), "axis"));  msgEl.addChild(**new** MessageElement(**new** QName("age"), "14"));  el.addChild(msgEl);  //子元素  msgEl = **new** MessageElement(**new** QName("copyRight"));  msgEl.addChild(**new** MessageElement(**new** QName("year"), "2010"));  msgEl.addChild(**new** MessageElement(**new** QName("number"), "123123123"));  el.addChild(msgEl);  el.addChild(**new** MessageElement(**new** QName("Tel"), "555666442"));    System.*out*.println(el.toString());  //设置请求的header  call.addHeader(el);    /\*\*  \* 注意：在请求readHeader方法的时候，也就是invoke执行的时候。会调用ReadHeaderHandler这个handler  \* 在请求完成后，准备返回值的时候或是说函数在运行完成了后。就会触发WriteHeaderHandler  \*/  call.invoke(**new** Object[] {});    /\*\* 响应的header \*/  Message msg = call.getResponseMessage();  SOAPEnvelope se = msg.getSOAPEnvelope();  System.*out*.println("responseMessage:" + se.getHeader());  }  } |

call.getResponseMessage();可以得到服务器响应的Header的信息

*运行后你可以看到客户端控制台会输出：*

|  |
| --- |
| <ns1:Header soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0" xmlns:ns1="soap" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><ServiceName xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">ReadHeaderService</ServiceName><CreateDate xsi:type="xsd:dateTime" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">2010-12-19T06:22:59.931Z</CreateDate><ns2:Version xsi:type="xsd:string" xmlns:ns2="service" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">1.0</ns2:Version><author><name xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">axis</name><age xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">14</age></author><copyRight><year xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">2010</year><number xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">123123123</number></copyRight><Tel xsi:type="xsd:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">555666442</Tel></ns1:Header>  responseMessage:<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><ns1:Header soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0" xmlns:ns1="soap"><ServiceName xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">ReadHeaderService</ServiceName><CreateDate xsi:type="xsd:dateTime" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">2010-12-19T06:22:59.931Z</CreateDate><ns2:Version xsi:type="soapenc:string" xmlns:ns2="service" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">1.0</ns2:Version><author><name xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">axis</name><age xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">14</age></author><copyRight><year xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">2010</year><number xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">123123123</number></copyRight><Tel xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">555666442</Tel></ns1:Header></soapenv:Header> |

responseMessage前面的内容是客户端将会写入到Header中的xml内容，而responseMessage后面的内容就是响应的Header的内容，这段内容是WriteHeaderHandler写入到ResponseMessage中的。

*再看看服务器端控制台的内容：*

|  |
| --- |
| 父节点：Header, 节点：ServiceName, 值：ReadHeaderService  父节点：Header, 节点：CreateDate, 值：2010-12-19T06:22:59.931Z  父节点：Header, 节点：Version, 值：1.0  父父节点：Header, 父节点：author, 节点：name##name, 值：axis##axis  父父节点：Header, 父节点：author, 节点：age##age, 值：14##14  父父节点：Header, 父节点：copyRight, 节点：year##year, 值：2010##2010  父父节点：Header, 父节点：copyRight, 节点：number##number, 值：123123123##123123123  父节点：Header, 节点：Tel, 值：555666442  ns1:Header  header Info:<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><ns1:Header soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0" xmlns:ns1="soap"><ServiceName xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">ReadHeaderService</ServiceName><CreateDate xsi:type="xsd:dateTime" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">2010-12-19T06:22:59.931Z</CreateDate><ns2:Version xsi:type="soapenc:string" xmlns:ns2="service" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">1.0</ns2:Version><author><name xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">axis</name><age xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">14</age></author><copyRight><year xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">2010</year><number xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">123123123</number></copyRight><Tel xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">555666442</Tel></ns1:Header></soapenv:Header> |

上面的文本内容是ReadHeaderHandler读取客户端的Header的内容和我们定制的xml的内容是一致的。而后面的xml能入是WriteHeaderHandler中输出的内容。

**十三、.Net调用axis WebService出现Client.NoSOAPAction的问题**

关于这个问题网上有很多解决方法，关键是AxisServlet在getSoapAction这个方法有点问题。解决方法就是集成这个AxisServlet然后重写getSoapAction方法。代码片段：

|  |
| --- |
| @Override  **private** String getSoapAction(HttpServletRequest request) **throws** AxisFault {  **return** request.getContextPath();  } |

然后在web.xml中配置你继承的AxisServlet，就不再是原来的AxisServlet了。

**十四、后记**

至此，axis的WebService的技术指南，基本就这么多内容。由于时间的仓促和个人的原因等诸多方面的因素，难免使得这篇文档存在些错误。但这篇文档上的代码都是经过运行测试通过的。

要感谢广大IT朋友的无私奉献，使得这篇文档的出现。本人在学习axis的时候参阅了大量来自互联网的技术文章，以及官方的文档。

参考文档：<http://ws.apache.org/axis/java/user-guide.html>

如果你发现这篇文档有什么问题，或是对这篇文档有更好的内容补充你可以发邮件到我的邮箱或是用QQ取得联系，也欢迎去我的博客留言，也欢迎大家一起交流探讨。

Email：[hoojo\_@126.com](mailto:hoojo_@126.com)

Blog：<http://blog.csdn.net/IBM_hoojo>

<http://hoojo.cnblogs.com/>

QQ:506637315（加入请注明身份）