Axis的简单准备

Axis2.x WebService开发指南

Axis的入门实例

Axis复杂对象类型的WebService

Axis的辅助工具发布、调用WebService

Axis WebService会话Session的管理

Axis用控制台Dos命令发布WebService

Axis跨多个WebService管理Session

Axis用Spring的JavaBean发布WebService

Axis异步调用WebService

Axis 的Module模块

Axis使用SoapMonitar监视WebService的请求和响应信息

Version1.0 2011-03-15

hoojo

授之以鱼，不如授之以渔！希望能给广大IT朋友带来一些帮助

**IT达人**

**自由IT职业**

**blog.csdn.net/IBM\_hoojo http://hoojo.cnblogs.com/**

**506637315**

**2011-3-15**

**Axis2.x WebService**

上次介绍了axis1.x的用法，这次继续上次的，将叙述axis2的用法。

Axis1.x在线博文：<http://www.cnblogs.com/hoojo/archive/2010/12/20/1911349.html>

1. **准备工作**

1、开发准备

首先需要下载axis2的相关jar包，到axis的官方网站即可获得开发的依赖包。

下载地址：<http://axis.apache.org/axis2/java/core/download.cgi>

现在最高的版本是1.5.4的

然后你需要下载官方提供的axis的eclipse插件工具，可以帮助我们打包（aar）及其生产客户端调用代码。

下载页面：<http://axis.apache.org/axis2/java/core/tools/index.html>

[Service Archive Wizard - Eclipse Plug-in](http://axis.apache.org/axis2/java/core/tools/eclipse/servicearchiver-plugin.html)

[Code Generator Wizard - Eclipse Plug-in](http://axis.apache.org/axis2/java/core/tools/eclipse/wsdl2java-plugin.html)

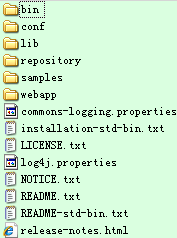
2、安装eclipse插件

如果你的eclipse插件安装成功后，会看到如下效果：



3、分析axis2-bin文件目录结构

下载下来的axis2的依赖库，其中有一个axis2-1.5.3-bin.zip的库文件，目录结构如下：



bin文件夹是axis2的常用工具，其中有将wsdl文件转换成客户端调用的wsdl2java工具及将java转换成wsdl文件的工具

conf是axis2的配置文件

lib运行所要的依赖库

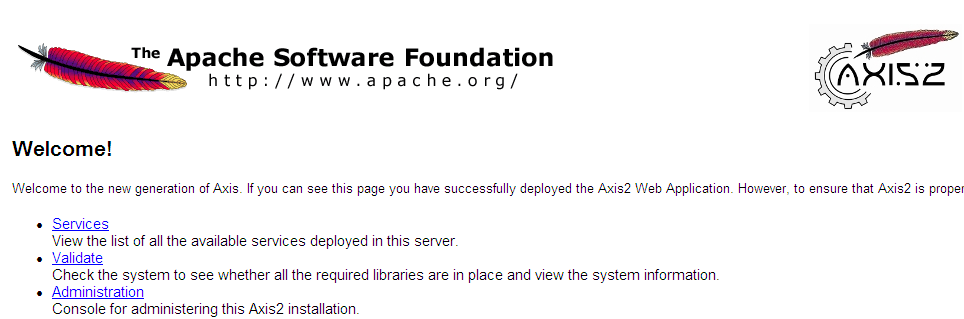
repository是发布过的axis服务和文件

sample是示例

webapp是web文件和jsp页面等

4、我们需要将下载下来的axis2-1.5.3-war.zip中的axis2.war这个文件放在我们的tomcat目录下，启动tomcat就会把war文件转成一个可以跑起来的axis2的项目。

Tomcat启动后，在浏览器中输入：<http://localhost:8080/axis2/>你可以看到



1. **Axis2的简单WebService示例**

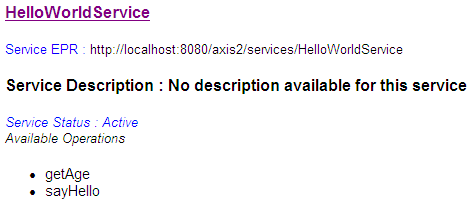
1、编写一个简单的WebService的服务器端代码，代码如下：

|  |
| --- |
| **import** java.util.Random;  /\*\*  \* <b>function:</b> WebService HelloWorld服务示例  \* **@author** hoojo  \* **@createDate** 2011-1-5 下午03:35:06  \* **@file** HelloWorldService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloWorldService {  **public** String sayHello(String name) {  **return** name + " say: hello [axis2]";  }    **public** **int** getAge(**int** i) {  **return** i + **new** Random().nextInt(100);  }  } |

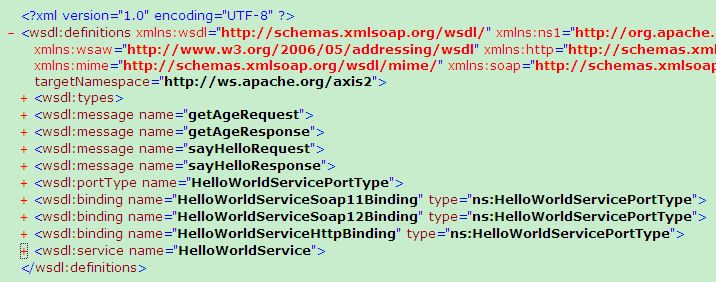
注意，上面的HelloWorldService是没有package的。Copy这个类的class文件，放到tomcat目录下的webapps的axis2的WEB-INF目录的pojo文件夹下。如果没有pojo这个目录就手动创建一个一个文件夹。然后在浏览器输入：<http://localhost:8080/axis2/>

点击Services的链接就可以看到我们手动发布的HelloWorldService了，或者是浏览器地址栏输入：<http://localhost:8080/axis2/services/listServices>

你就可以看到你刚才粘贴过去的这个WebService了。



点击链接就可以看到wsdl文件的内容了。内容很多，如果你看过axis1.x的介绍就知道wsdl文件的大致结构了。



下面讲解下为什么要将class放在pojo文件夹下。首先我们看看[tomcat\_home]/webapps/axis2/WEB-INF/conf/axis2.xml

该文件中有这样一行代码：

<deployer extension=".class" directory="pojo" class="org.apache.axis2.deployment.POJODeployer"/>

.class的后缀文件放在目录pojo目录下。

2、测试这个WebService的方法

复制上面的HelloWorldService的链接地址，然后带上方法名称和参数值即可测试调用是否成功。如下：

http://localhost:8080/axis2/services/HelloWorldService/sayHello?name=jack

<http://localhost:8080/axis2/services/HelloWorldService>这个是WebService的地址

/sayHello是方法名称，？name=jack是参数名称和值

在浏览器中输入上面的地址后，可以看到如下效果：



可以看到返回值，和方法名称。ns:sayHelloResponse是方法名称，所有的方法名称后都会带上Response，后面的ns当然是当前方法所在的类的包名了，这里没有package就是默认的axis2的域名。

同样，getAge方法，也是一样的调用方法。

<http://localhost:8080/axis2/services/HelloWorldService/getAge?i=22>

结果如下：



3、下面我们看下客户端调用代码的编写，代码如下：

|  |
| --- |
| **package** com.hoo.service;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>HelloWorldService 客户端调用代码  \* **@author** hoojo  \* **@createDate** 2011-1-7 下午03:55:05  \* **@file** HelloWorldClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** HelloWorldClient {  **public** **static** **void** main(String[] args) **throws** AxisFault {  //RPCServiceClient是RPC方式调用  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  //设置调用WebService的URL  String address = "http://localhost:8080/axis2/services/HelloWorldService";  EndpointReference epf = **new** EndpointReference(address);  options.setTo(epf);    /\*\*  \* 设置将调用的方法，http://ws.apache.org/axis2是方法  \* 默认（没有package）命名空间，如果有包名  \* 就是http://service.hoo.com 包名倒过来即可  \* sayHello就是方法名称了  \*/  QName qname = **new** QName("http://ws.apache.org/axis2", "sayHello");  //指定调用的方法和传递参数数据，及设置返回值的类型  Object[] result = client.invokeBlocking(qname, **new** Object[] { "jack" }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);    qname = **new** QName("http://ws.apache.org/axis2", "getAge");  result = client.invokeBlocking(qname, **new** Object[] { **new** Integer(22) }, **new** Class[] { **int**.**class** });  System.*out*.println(result[0]);  }  } |

如果你了解或是[看过axis1.x的WebService](http://www.cnblogs.com/hoojo/archive/2010/12/20/1911349.html)，这里的客户端调用代码比较简单，有部分关键注释，这里就不多加赘述。值得注意的是axis2的WebService客户端调用的返回对象是一个Object的数组，这点和axis1.x有很大不同。我们一般拿数组的第一个值，转换强转即可。

1. **复杂对象类型的WebService**

1、这次我们编写复杂点的WebService方法，返回的数据是我们定义属性带getter、setter方法JavaBean，一维数组、二维数组等。

看代码：

|  |
| --- |
| **import** java.io.File;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.util.Random;  **import** data.User;  /\*\*  \* <b>function:</b>复杂类型数据的WebService：字节数组、返回一维int数组、二维String数组及自定义JavaBean对象等  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午03:34:52  \* **@file** ComplexTypeService.java  \* **@package**  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ComplexTypeService {    **public** String upload4Byte(**byte**[] b, **int** len) {  String path = "";  FileOutputStream fos = **null**;  **try** {  String dir = System.*getProperty*("user.dir");  File file = **new** File(dir + "/" + **new** Random().nextInt(100) + ".jsp");  fos = **new** FileOutputStream(file);  fos.write(b, 0, len);  path = file.getAbsolutePath();  System.*out*.println("File path: " + file.getAbsolutePath());  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  fos.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** path;  }    **public** **int**[] getArray(**int** i) {  **int**[] arr = **new** **int**[i];  **for** (**int** j = 0; j < i; j++) {  arr[j] = **new** Random().nextInt(1000);  }  **return** arr;  }    **public** String[][] getTwoArray() {  **return** **new** String[][] { { "中国", "北京" }, { "日本", "东京" }, { "中国", "上海", "南京" } };  }    **public** User getUser() {  User user = **new** User();  user.setAddress("china");  user.setEmail("jack@223.com");  user.setName("jack");  user.setId(22);  **return** user;  }  } |

上面的WebService服务分别是传递字节完成数据上传，返回一维int数组和二维字符串数组，以及返回User JavaBean对象。

下面看看User Bean代码：

|  |
| --- |
| **package** data;  **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;    //getter/setter  @Override  **public** String toString() {  **return** **this**.id + "#" + **this**.name + "#" + **this**.email + "#" + **this**.address;  }  } |

值得注意的是这个User对象的package是data，如果是其它的package，你就需要在tomcat目录下的webapps中的axis2的WEB-INF目录下创建一个data目录，和你的User对象的目录保持一致。否则你的WebService将会出现ClassNotFontException异常。然后重启你的tomcat，虽然axis2支持热部署。

2、编写调用上面WebService的客户端代码，代码如下：

|  |
| --- |
| **package** com.hoo.service;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  **import** com.hoo.entity.User;  /\*\*  \* <b>function:</b>复杂类型数据WebService客户端调用代码  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午03:36:38  \* **@file** ComplexTypeServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ComplexTypeServiceClient {  **public** **static** **void** main(String[] args) **throws** IOException {  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  String address = "http://localhost:8080/axis2/services/ComplexTypeService";  EndpointReference epr = **new** EndpointReference(address);  options.setTo(epr);    QName qname = **new** QName("http://ws.apache.org/axis2", "upload4Byte");  String path = System.*getProperty*("user.dir");  File file = **new** File(path + "/WebRoot/index.jsp");  FileInputStream fis = **new** FileInputStream(file);  **int** len = (**int**) file.length();  **byte**[] b = **new** **byte**[len];  **int** read = fis.read(b);  //System.out.println(read + "#" + len + "#" + new String(b));  fis.close();  Object[] result = client.invokeBlocking(qname, **new** Object[] { b, len }, **new** Class[] { String.**class** });  System.*out*.println("upload:" + result[0]);    qname = **new** QName("http://ws.apache.org/axis2", "getArray");  result = client.invokeBlocking(qname, **new** Object[] { 3 }, **new** Class[] { **int**[].**class** });  **int**[] arr = (**int**[]) result[0];  **for** (Integer i : arr) {  System.*out*.println("int[] :" + i);  }    qname = **new** QName("http://ws.apache.org/axis2", "getTwoArray");  result = client.invokeBlocking(qname, **new** Object[] {}, **new** Class[] { String[][].**class** });  String[][] arrStr = (String[][]) result[0];  **for** (String[] s : arrStr) {  **for** (String str : s) {  System.*out*.println("String[][]: " + str);  }  }    qname = **new** QName("http://ws.apache.org/axis2", "getUser");  result = client.invokeBlocking(qname, **new** Object[] {}, **new** Class[] { User.**class** });  User user = (User) result[0];  System.*out*.println("User: " + user);  }  } |

上面的代码运行后的结果是：

|  |
| --- |
| upload:D:\tomcat-6.0.28\bin\28.jsp  int[] :548  int[] :201  int[] :759  String[][]: 中国  String[][]: 北京  String[][]: 日本  String[][]: 东京  String[][]: 中国  String[][]: 上海  String[][]: 南京  User: 22#jack#jack@223.com#china |

1. **用axis2的辅助工具发布、调用WebService**

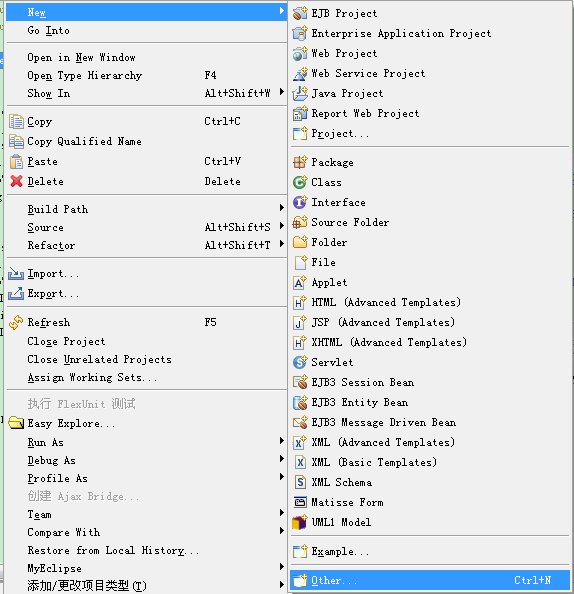
这次我们编辑一个返回User对象、List<User>、Map<String, User>、User[]的形式，并且用axis2的工具完成发布WebService。不再复制class到axis2的工程目录下。

下面看看服务器端WebService代码：

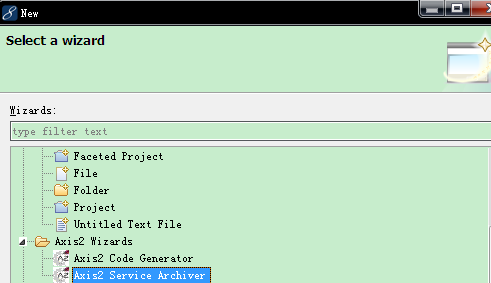
|  |
| --- |
| **package** com.hoo.service;  **import** java.util.ArrayList;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Random;  **import** com.hoo.entity.User;  /\*\*  \* <b>function:</b>传递User对象类型的List、Map、User、Array等数据方法的WebService  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午03:50:06  \* **@file** ManagerUserService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ManagerUserService {    /\*\*  \* <b>function:</b>传递User对象，返回User对象  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午03:54:36  \* **@param** user  \* **@return** User  \*/  **public** User editUser(User user) {  user.setId(**new** Random().nextInt(100));  **return** user;  }    /\*\*  \* <b>function:</b>返回List&lt;User&gt;的数据类型  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午03:57:19  \* **@param** j  \* **@return** List&lt;User&gt;  \*/  **public** List<User> getUsers(**int** j) {  List<User> users = **new** ArrayList<User>();  **for** (**int** i = 0; i < j; i++) {  User user = **new** User();  user.setAddress("china");  user.setEmail("hoojo\_@126.com");  user.setName("hoojo");  user.setId(22 + i);  users.add(user);  }  **return** users;  }    /\*\*  \* <b>function:</b>返回Map&lt;String, User&gt;类型的数据  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午04:03:41  \* **@param** j  \* **@return** Map&lt;String, User&gt;  \*/  **public** Map<String, User> getUser4Map(**int** j) {  Map<String, User> users = **new** HashMap<String, User>();  **for** (**int** i = 0; i < j; i++) {  User user = **new** User();  user.setAddress("china");  user.setEmail("amy@223.com");  user.setName("amy");  user.setId(22 + i);  users.put("user#"+ i, user);  }  **return** users;  }    /\*\*  \* <b>function:</b>返回User的数组数据  \* **@author** hoojo  \* **@createDate** 2011-1-13 下午04:05:23  \* **@param** j  \* **@return** User[]  \*/  **public** User[] getUser4Array(**int** j) {  User[] users = **new** User[j];  **for** (**int** i = 0; i < j; i++) {  User user = **new** User();  user.setAddress("china");  user.setEmail("tom@223.com");  user.setName("tom");  user.setId(22 + i);  users[i] = user;  }  **return** users;  }  } |

**A、下面用axis2的工具生成aar文件，然后完成发布WebService。**

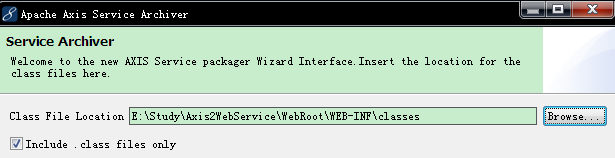
1. 点击工作空间WorkSpace，右键New选择Other



1. 选择Axis2 Service Archiver，点击Next



1. 然后点击Browser选择你当前的工程的classes目录

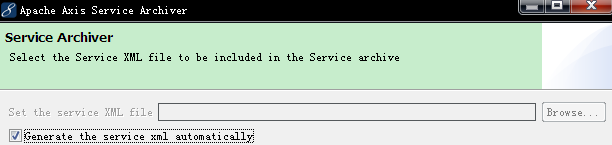


点击Next

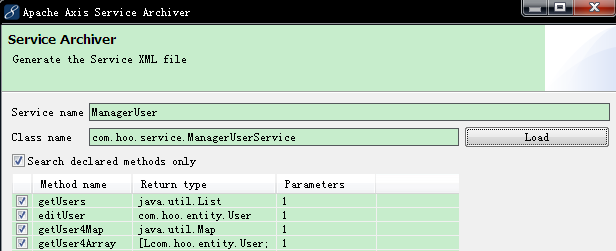
1. 没有WSDL，选择第一项，点击Next，然后再Next一次



1. 不用设置service.xml，直接Next

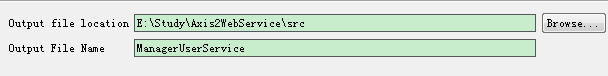


1. 最关键一步，填写WebService的Name，设置class路径点击Load你就可以看到当前class的方法



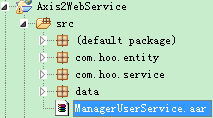
勾中的方法表示将会被暴露到客户端可以调用的方法，然后点击Next

1. 设置aar文件的名称和保持的目录

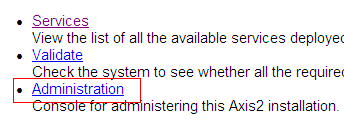


点击Finish你可以看到提示successfully的提示。

1. 刷新当前工程可以看到ManangerUserService.aar文件了



1. 下面我们在浏览器中访问这个地址：<http://localhost:8080/axis2/>



点击Admin这个链接，可以看到要输入用户名和密码，用户名和密码在

Tomcat\_Home\webapps\axis2\WEB-INF\conf下的axis2.xml中可以找到

<parameter name="userName">admin</parameter>

<parameter name="password">axis2</parameter>

1. 输入密码进入管理页面后可以看到



下面介绍下上面常用的功能

Upload Service当然是上传aar的文件，完成发布WebService的工具了

Available Service是查看当前发布的Service详细信息

Available Service Groups是Service组

Global Chains是可以用的全局的Chain

Operation Specific Chains是某个具体操作的chain

Deactivate Service 是取消某个Service

Activate Service 是将取消发布的Service再发布出去

Edit Parameters是修改WebService类的参数

1. 我们这里用Upload Service这个工具，然后选择我们刚才用axis2的工具生成aar文件。



其实这一步我们可以直接将ManagerUserService.aar文件copy到

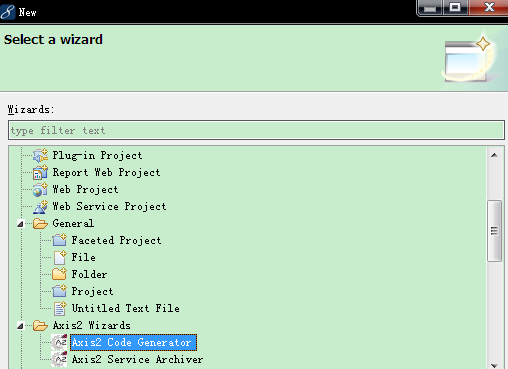
tomcat-home\webapps\axis2\WEB-INF\services这个目录下。

1. 然后点击Available Service就可以看到ManagerUserService这个Service了

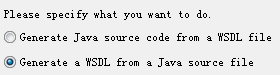


**B、现在我们要用axis2的工具生成wsdl文件**

1、点击WorkSpace，右键New选择Other。然后选择客户端代码生成工具

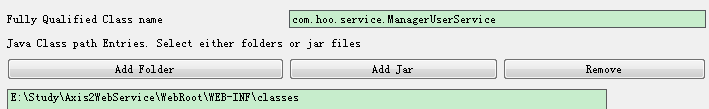


2、点击Next，你可以看到让你选择wsdl。选择第一个，然后点击Next



上面是根据Java class代码生成wsdl文件

3、填写你的WebService的classpath，然后选择你当前class所在工程的class目录

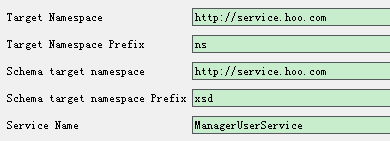


然后点击Add Folder选择你的工程的classes 或 bin目录，也就是你填写的class所在的目录，然后点击Test 测试是否正确。

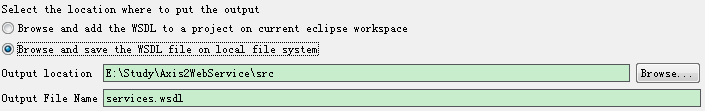


点击Next继续下一步

4、可以看到即将发布的WebService名称和命名空间



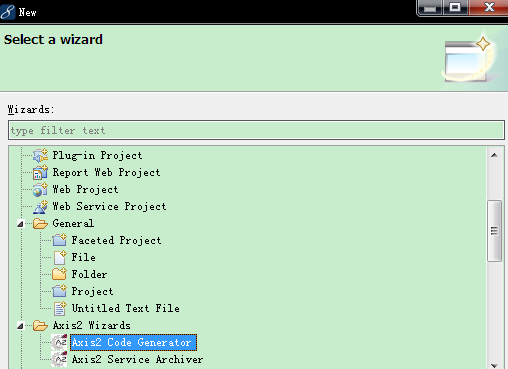
5、继续Next，选择wsdl文件保存目录



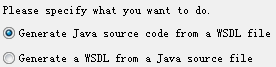
这样services.wsdl文件就生成了。

**C、用axis2的工具生成客户端调用的代码**

1、右键workspace，点击New选择Other，然后选择aixs2的生成代码工具



2、选择第一项，根据wsdl文件生成Java代码，然后Next



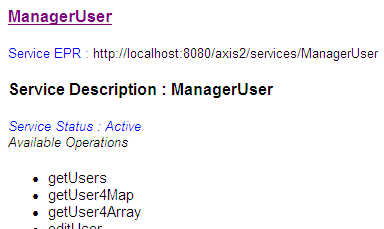
3、然后选择刚才上面生成好的wsdl文件



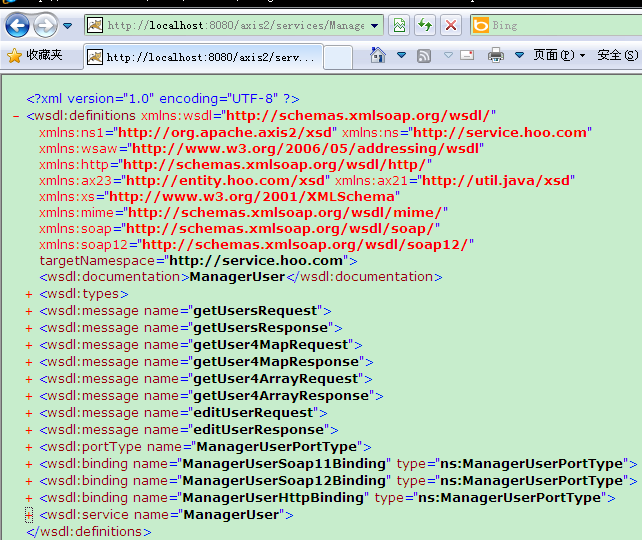
如果你有向tomcat中发布过你的WebService，那么你可以通过WebBrowser访问

<http://localhost:8080/axis2/>

点击Services链接，就可以看到你的WebService。点击你的WebService



样你也可以看到wsdl内容



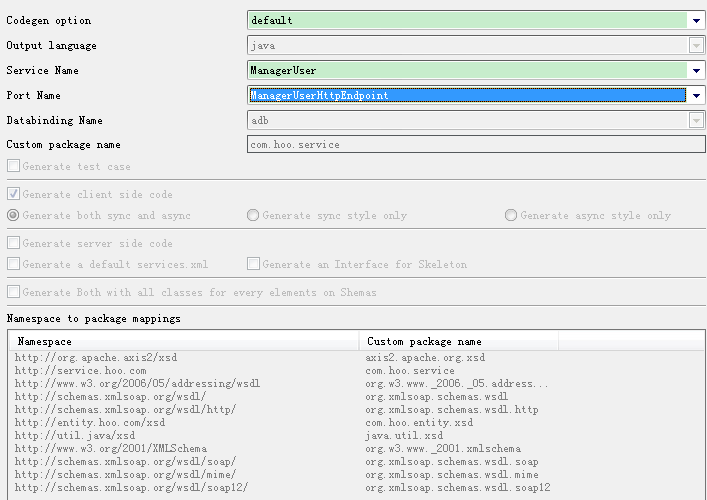
这样你也可以在上面的WSDL file location中填写：



<http://localhost:8080/axis2/services/ManagerUser?wsdl>

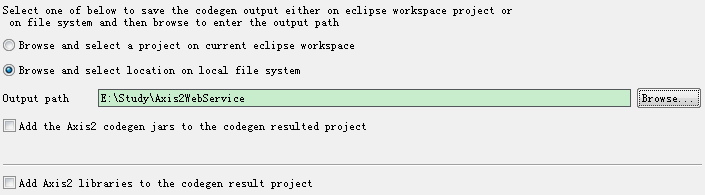
这个地址也是可以的

4、点击Next，就可以看到你要生成指定方法的代码的选项了



注意的是在选择PortName的时候，不同的选项将会生产不同的调用代码。其调用方式也略有不同。但你可以看生成的源代码进行详细的了解。

5、点击Next，选择代码输出保存的目录，然后点击Finish



刷新目录后发现多了2个文件



1. **编写客户端代码**

|  |
| --- |
| **package** com.hoo.service;  **import** java.rmi.RemoteException;  **import** com.hoo.service.ManagerUserStub.EditUserResponse;  /\*\*  \* <b>function:</b>ManagerUserService客户端调用代码  \* **@author** hoojo  \* **@createDate** 2011-1-14 下午03:17:31  \* **@file** ManagerUserServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** ManagerUserServiceClient {  **public** **static** **void** main(String[] args) **throws** RemoteException {  String target = "http://localhost:8080/axis2/services/ManagerUser";  ManagerUserStub stub = **new** ManagerUserStub(target);    /\*\*  \* 调用User Bean模式：  \* EditUser、User是内部静态类，axis2会帮我们进行参数封装  \*/  ManagerUserStub.EditUser editUser = **new** ManagerUserStub.EditUser();  ManagerUserStub.User u = **new** ManagerUserStub.User();  u.setAddress("china");  u.setEmail("axis2@axis2.com");  u.setName("axis2");  u.setId(222);  editUser.setUser(u);  //返回值也被封装  EditUserResponse eur = stub.editUser(editUser);  ManagerUserStub.User returnUser = eur.get\_return();  //returnUser.getId()在服务器端动态重置过  System.*out*.println(returnUser.getId() + "#" + returnUser.getName() + "#" + returnUser.getEmail() + "#" + returnUser.getAddress());    /\*\*  \* User数组模式  \*/  ManagerUserStub.GetUser4Array userArray = **new** ManagerUserStub.GetUser4Array();  userArray.setJ(3);  ManagerUserStub.GetUser4ArrayResponse userArrResponse = stub.getUser4Array(userArray);  ManagerUserStub.User[] userArr = userArrResponse.get\_return();  **for** (ManagerUserStub.User user : userArr) {  System.*out*.println(user.getId() + "#" + user.getName() + "#" + user.getEmail() + "#" + user.getAddress());  }      /\*\*  \* Map User模式，不支持Map、List  \*/  /\*ManagerUserStub.GetUser4Map userMap = new ManagerUserStub.GetUser4Map();  userMap.setJ(3);  ManagerUserStub.GetUser4MapResponse userMapResponse = stub.getUser4Map(userMap);  ManagerUserStub.Map map = userMapResponse.get\_return();  System.out.println(map);\*/    /\*\*  \* GetUsers是ManagerUserStub的内部类，axis2的代码生成工具会帮我们把参数更具wsdl文件的内容进行封装  \*/  /\*ManagerUserStub.GetUsers getUsers = new ManagerUserStub.GetUsers();  getUsers.setJ(3);//服务器的参数int j  ManagerUserStub.GetUsersResponse usersResponse = stub.getUsers(getUsers);  System.out.println(usersResponse.get\_return());\*/  }  } |

运行后发现不支持Map、List，不能读取数据类型。Wsdl文件中是anyType，可能需要自己用配置描述返回类型。【有待跟进、解决】

1. **下面用MyEclipse的WebService工具生成代码**
2. **WebService会话Session的管理**

1、新建Session的WebService测试代码，代码很简单。就是记录用户登录的状态信息，放在MessageContext的ServiceContext中。代码如下：

|  |
| --- |
| **package** com.hoo.service;  **import** org.apache.axis2.context.MessageContext;  **import** org.apache.axis2.context.ServiceContext;  /\*\*  \* <b>function:</b>WebService Session会话消息  \* **@author** hoojo  \* **@createDate** 2011-3-8 下午04:11:27  \* **@file** LoginService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** LoginService {  **public** **boolean** login(String userName, String password) {  MessageContext context = MessageContext.*getCurrentMessageContext*();  ServiceContext ctx = context.getServiceContext();  **if** ("admin".equals(userName) && "123456".equals(password)) {  ctx.setProperty("userName", userName);  ctx.setProperty("password", password);  ctx.setProperty("msg", "登陆成功");  **return** **true**;  }  ctx.setProperty("msg", "登陆失败");  **return** **false**;  }    **public** String getLoginMessage() {  MessageContext context = MessageContext.*getCurrentMessageContext*();  ServiceContext ctx = context.getServiceContext();  **return** ctx.getProperty("userName") + "#" + ctx.getProperty("msg");  }  } |

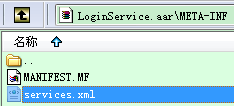
这里保存Session信息的主要是通过MessageContext这个上下文来获取ServiceContext的上下文，然后通过setProperty来保存session的信息，通过getProperty来获取session信息。

Session的作用可以在使用WebService登录的时候，保存用户的登录状态或是会话消息。但是使用session需要设置发布Service的时候，设置WebService的作用域。作用域默认是request，它还有另外三个值，分别是：application、soapsession、transportsession；我们可以选择使用transportsession和application分别实现同一个WebService类和跨WebService类的会话管理。

2、使用axis2的工具生成aar文件，并发布LoginService服务。



关于这里发布LoginService的步骤就不一一介绍，上面已经有说过了。当你用这个步骤发布WebService的时候，你可以打开压缩文件的方式aar文件（其实不一定要用aar文件，只是官方推荐的aar，jar文件也是可以的），在META-INF目录下有一个services.xml文件，看到它的<service>这个标签，没有设置scope属性。



scope默认的是request，我们得修改services.xml的内容。

修改后services.xml内容如下：

|  |
| --- |
| <service name=*"LoginService"* scope=*"transportsession"*>  <description>  Please Type your service description here  </description>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  </messageReceivers>  <parameter name=*"ServiceClass"*>com.hoo.service.LoginService</parameter>  </service> |

这个地方的scope是关键，如果这里不设置的话session是无法存放信息。

**3、用控制台Dos命令发布WebService**

这个是手工打包，也就是我们经常用的jar命令

首先我们得需要一个services.xml文件，文件内容如上面的。

<service name=*"LoginService"* scope=*"transportsession"*>

name是当前发布WebService的名称，scope是会话作用域。保存在Session中就要用这个transportsession这个作用域。

<parameter name=*"ServiceClass"*>com.hoo.service.LoginService</parameter>

这个是当前Service的类路径，包名+类名

<messageReceivers>

<messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*

class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />

<messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*

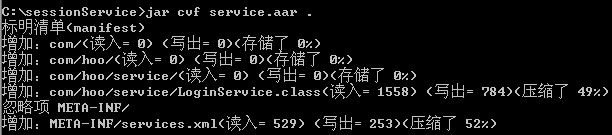
class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"*/>

</messageReceivers>

这个应该是参数和返回值信息的解析类，in-only是输入也就是参数，in-out是输出也就是返回值。

1. 首先，我们在C盘下新建一个sessionService的目录，然后将上面的services.xml文件中的ServiceClass的类的class文件copy到这个目录下，注意要带上package目录，然后在sessionService目录下新建一个META-INF将上面的services.xml文件放到这个目录下
2. 然后运行cmd进入控制台，进入sessionService这个目录，键入命令如下：

jar cvf service.aar



service.aar是打包后的文件名称，.是指代当前目录

如果你不懂jar命令，你可以输入jar会有命令语法的提示



进入sessionService这个目录中，你可以看到service.aar文件就有了。然后将这个文件copy到axis2的WEB-INF目录下的services目录中，也就是

[tomcat-home]\webapps\axis2\WEB-INF\services

然后可以通过：<http://localhost:8080/axis2/services/listServices>

就可以查看刚才发布的服务了。

**这样就完成手工发布打包发布WebService了**

1. 编写WebService客户端的请求代码

|  |
| --- |
| **package** com.hoo.service;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>LoginService Session客户端代码  \* **@author** hoojo  \* **@createDate** 2011-3-9 上午11:21:59  \* **@file** LoginServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** LoginServiceClient {  **public** **static** **void** main(String[] args) **throws** AxisFault {  String target = "http://localhost:8080/axis2/services/LoginService";  //target = "http://localhost:8080/axis2/services/MyLoginService";  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  options.setManageSession(**true**);    EndpointReference epr = **new** EndpointReference(target);  options.setTo(epr);    QName qname = **new** QName("http://service.hoo.com", "login");  //指定调用的方法和传递参数数据，及设置返回值的类型  Object[] result = client.invokeBlocking(qname, **new** Object[] { "admin", "123456" }, **new** Class[] { **boolean**.**class** });  System.*out*.println(result[0]);    qname = **new** QName("http://service.hoo.com", "getLoginMessage");  result = client.invokeBlocking(qname, **new** Object[] { **null** }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);  }  } |

执行后，结果如下：

true

admin#登陆成功

上面要说明的是options.setManageSession(**true**);是开启对Session管理的支持

1. **跨多个WebService管理Session**

当多个WebService的时候，我们要管理它的Session。这个时候我们得依靠ServiceGroupContext保存session信息；然后在发布WebService的时候，services.xml文件的的service表情的scope就不再说request或是transportsession了，而是application；最后同样要开启对session的管理，即options.setManageSession(**true**);

1. 首先多个WebService的session管理的代码如下：

|  |
| --- |
| **package** com.hoo.service;  **import** org.apache.axis2.context.MessageContext;  **import** org.apache.axis2.context.ServiceGroupContext;  /\*\*  \* <b>function:</b>管理多个会话Session信息  \* **@author** hoojo  \* **@createDate** 2011-3-9 下午05:11:07  \* **@file** LoginSessionService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** LoginSessionService {  **public** **boolean** login(String userName, String password) {  MessageContext context = MessageContext.*getCurrentMessageContext*();  ServiceGroupContext ctx = context.getServiceGroupContext();  **if** ("admin".equals(userName) && "123456".equals(password)) {  ctx.setProperty("userName", userName);  ctx.setProperty("password", password);  ctx.setProperty("msg", "登陆成功");  **return** **true**;  }  ctx.setProperty("msg", "登陆失败");  **return** **false**;  }    **public** String getLoginMessage() {  MessageContext context = MessageContext.*getCurrentMessageContext*();  ServiceGroupContext ctx = context.getServiceGroupContext();  **return** ctx.getProperty("userName") + "#" + ctx.getProperty("msg");  }  } |

和上面的Session一样的操作，只不过是用ServiceGroupContext上下文来存取session信息

另外还需要用一个Service来查询session的信息，SearchService的代码如下：

|  |
| --- |
| **package** com.hoo.service;  **import** org.apache.axis2.context.MessageContext;  **import** org.apache.axis2.context.ServiceGroupContext;  /\*\*  \* <b>function:</b>查找多服务Session会话中的消息  \* **@author** hoojo  \* **@createDate** 2011-3-9 下午05:22:39  \* **@file** SearchSessionServcie.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** SearchSessionServcie {  **public** String findSessionMessage(String key) {  MessageContext mc = MessageContext.*getCurrentMessageContext*();  ServiceGroupContext ctx = mc.getServiceGroupContext();  **if** (ctx.getProperty(key) != **null**) {  **return** "找到的数据<" + key + ", " + ctx.getProperty(key) + ">";  } **else** {  **return** "没有找到<" + key + ">的数据";  }  }  } |

1. 编写services.xml来发布这2个服务，还以前不一样的。这一次是用一个services.xml文件配置2个service，同时发布2个服务。Xml代码如下：

|  |
| --- |
| <serviceGroup>  <service name=*"LoginSessionService"* scope=*"application"*>  <description>  Web Service Session例子  </description>  <parameter name=*"ServiceClass"*>  com.hoo.service.LoginSessionService  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  </messageReceivers>  </service>  <service name=*"SearchSessionService"* scope=*"application"*>  <description>  Web Service Search Session例子  </description>  <parameter name=*"ServiceClass"*>  com.hoo.service.SearchSessionServcie  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  </messageReceivers>  </service>  </serviceGroup> |

1. 发布完成后，可以通过<http://localhost:8080/axis2/services/listServices>查看发布的WebService服务，编写客户端的测试代码，code如下：

|  |
| --- |
| **package** com.hoo.service;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>多会话Session管理，WebService客户端请求代码  \* **@author** hoojo  \* **@createDate** 2011-3-9 下午05:17:15  \* **@file** LoginSessionServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** LoginSessionServiceClient {  **public** **static** **void** main(String[] args) **throws** AxisFault {  String target = "http://localhost:8080/axis2/services/LoginSessionService";  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  options.setManageSession(**true**);    EndpointReference epr = **new** EndpointReference(target);  options.setTo(epr);    QName qname = **new** QName("http://service.hoo.com", "login");  //指定调用的方法和传递参数数据，及设置返回值的类型  Object[] result = client.invokeBlocking(qname, **new** Object[] { "admin", "123456" }, **new** Class[] { **boolean**.**class** });  System.*out*.println(result[0]);    qname = **new** QName("http://service.hoo.com", "getLoginMessage");  result = client.invokeBlocking(qname, **new** Object[] { **null** }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);    target = "http://localhost:8080/axis2/services/SearchSessionService";  epr = **new** EndpointReference(target);  options.setTo(epr);    qname = **new** QName("http://service.hoo.com", "findSessionMessage");  result = client.invokeBlocking(qname, **new** Object[] { "userName" }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);  qname = **new** QName("http://service.hoo.com", "findSessionMessage");  result = client.invokeBlocking(qname, **new** Object[] { "msg" }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);    qname = **new** QName("http://service.hoo.com", "findSessionMessage");  result = client.invokeBlocking(qname, **new** Object[] { "password" }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);  }  } |

运行后结果如下：

true

admin#登陆成功

找到的数据<userName, admin>

找到的数据<msg, 登陆成功>

找到的数据<password, 123456>

1. 如果将services.xml文件<service name=*"SearchSessionService"* scope=*"application"*>的内容改成scope=transportsession，看看什么情况。是不是找不到session中的内容。
2. **用Spring创建的JavaBean发布成WebService**

1、首先，看看这个简单的类代码：

|  |
| --- |
| **package** com.hoo.service;  /\*\*  \* <b>function:</b>Spring的装载Bean的Service  \* **@author** hoojo  \* **@createDate** 2011-3-9 下午06:30:26  \* **@file** SpringService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** SpringService {  **private** String name;  **private** **int** number;    **public** **void** setName(String name) {  **this**.name = name;  }  **public** **void** setNumber(**int** number) {  **this**.number = number;  }  **public** String getMessage() {  **return** **this**.number + "#" + **this**.name;  }  } |

将这个类编译后的class，放到[tomcat-home]\webapps\axis2\WEB-INF\classes中，注意带上package类路径的目录。

2、在<Tomcat安装目录>\webapps\axis2\WEB-INF\web.xml文件中加入下面的内容：

|  |
| --- |
| <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:applicationContext.xml</param-value>  </context-param> |

这段配置会Spring的朋友肯定知道，这个是加载spring容器和设置spring的配置文件的。

3、在[tomcat-home]\webapps\axis2\WEB-INF\classes中新建一个applicationContext.xml文件，文件中的内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>  <bean id=*"SpringBeanService"* class=*"com.hoo.service.SpringService"*>  <property name=*"name"* value=*"hoojo"*/>  <property name=*"number"* value=*"1"*/>  </bean>  </beans> |

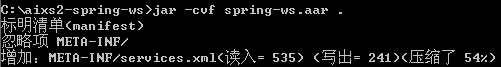
上面配置了SpringService，利用Spring的容器帮我们设置name、number这2个属性的值。

1. 在tomcat-home\webapps\axis2\WEB-INF\lib目录中有一个axis2-spring-1.5.3.jar文件，该文件用于将被装配JavaBean的发布成WebService。同样这里我们采用手动打包发布的模式，在C盘建立一个axis2-spring-ws的目录，然后在此目录中新建一个META-INF的目录，目录中添加一个services.xml文件，内容如下：

|  |
| --- |
| <service name=*"springService"*>  <description>  Spring aware  </description>  <parameter name=*"ServiceObjectSupplier"*>  org.apache.axis2.extensions.spring.receivers.SpringServletContextObjectSupplier  </parameter>  <!-- 这里的值和Spring的applicationContext中配置的bean的id对应,就是通过在spring的容器获取注入的bean -->  <parameter name=*"SpringBeanName"*>  SpringBeanService  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  </messageReceivers>  </service> |

在文件中需要通过ServiceObjectSupplier参数指定SpringServletContextObjectSupplier类来获得Spring的ApplicationContext对象；

然后用jar命令创建aar文件

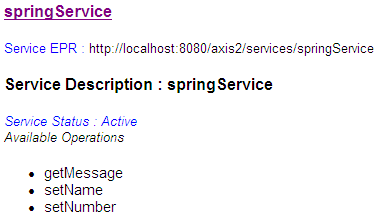


然后将生产的aar文件，复制到[tomcat\_home]\webapps\axis2\WEB-INF\services目录中

1. 然后重启tomcat，访问：

<http://localhost:8080/axis2/services/listServices>

就可以看到刚才发布的springService了



1. 编写客户端测试代码

|  |
| --- |
| **package** com.hoo.service;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>SpringService客户端代码  \* **@author** hoojo  \* **@createDate** 2011-3-14 上午14:21:59  \* **@file** LoginServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** SpringServiceClient {  **public** **static** **void** main(String[] args) **throws** AxisFault {  String target = "http://localhost:8080/axis2/services/springService";  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  options.setManageSession(**true**);    EndpointReference epr = **new** EndpointReference(target);  options.setTo(epr);    QName qname = **new** QName("http://service.hoo.com", "getMessage");  Object[] result = client.invokeBlocking(qname, **new** Object[] { **null** }, **new** Class[] { String.**class** });  System.*out*.println(result[0]);  }  } |

运行后，看到bean注入的值有输出。

|  |
| --- |
| 1#hoojo |

1. **异步调用WebService**

异步，说到异步需要首先将以下同步。同步就是代码按照顺序执行，当前面的代码的请求没有正常返回结果的情况下，后面的代码是不能运行。而异步正好和这点不同，异步是代码运行后，不管当前的请求是否返回结果，后面的代码都会继续运行。

关于异步在此就不再赘述了，有兴趣的可以去网上查查这方面的资料。

1. 编写服务器端的代码。

|  |
| --- |
| **package** com.hoo.service;  /\*\*  \* <b>function:</b>异步WebService服务器端代码  \* **@author** hoojo  \* **@createDate** 2011-3-14 上午08:16:59  \* **@file** AsynchronousService.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** AsynchronousService {  **public** String execute() {  System.*out*.println("正在执行此代码……");  //延迟5秒后，返回结果  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  **return** "完成";  }  } |

1. services.xml文件，创建aar文件，然后复制aar文件到[tomcat-home]\webapps\axis2\WEB-INF\services目录下

services.xml

|  |
| --- |
| <service name=*"AsyncService"*>  <description>  AsyncService  </description>  <parameter name=*"ServiceClass"*>  com.hoo.service.AsynchronousService  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  </messageReceivers>  </service> |

1. 编写客户端测试代码

|  |
| --- |
| **package** com.hoo.service;  **import** java.io.IOException;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.client.async.AxisCallback;  **import** org.apache.axis2.context.MessageContext;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>异步WebService客户端代码  \* **@author** hoojo  \* **@createDate** 2011-3-14 上午09:00:03  \* **@file** AsynchronousServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** AsynchronousServiceClient {  **public** **static** **void** main(String[] args) **throws** IOException {  String target = "http://localhost:8080/axis2/services/AsyncService?wsdl";  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  options.setManageSession(**true**);    EndpointReference epr = **new** EndpointReference(target);  options.setTo(epr);    QName qname = **new** QName("http://service.hoo.com", "execute");  //指定调用的方法和传递参数数据，及设置返回值的类型  client.invokeNonBlocking(qname, **new** Object[] {}, **new** AxisCallback() {    **public** **void** onMessage(MessageContext ctx) {  System.*out*.println(ctx.getEnvelope());  System.*out*.println("Message:" + ctx.getEnvelope().getFirstElement().getFirstElement().getFirstElement().getText());  }    **public** **void** onFault(MessageContext ctx) {  }    **public** **void** onError(Exception ex) {  }    **public** **void** onComplete() {  }  });  System.*out*.println("异步WebService");  //阻止程序退出  System.*in*.read();  }  } |

上面是异步调用WebService的代码，调用的方法是client.invokeNonBlocking，这个方法有三个参数，参数一是执行的方法签名，参数二是执行该方法的参数，参数三是异步回调，这里隐式实现AxiaCallback接口

注意的是运行程序的时候要用Debug方式运行。

1. **编写Module模块**

Axis可以通过Module模块进行扩展，用户可以编写定制自己的Module模块。编写一个Module的模块至少需要实现两个接口，分别是Handler和Module接口。开发axis2的Module模块需要如下步骤：

1. 实现Module接口的实现类，这个类要完成基本的初始化、销毁等操作
2. 实现Handler接口的实现类，这个类主要是完成业务处理
3. 在META-INF目录下，创建module.xml配置文件
4. 在axis2.xml中增加配置module的模块
5. 在services.xml中增加module的模块配置
6. 最后发表axis2的module模块，需要用jar命令将工程打包成mar，然后将mar文件发布到[tomcat\_home]/webapps/axis2/WEB-INF/modules目录下；

首先编写一个简单的WebService，代码如下：

|  |
| --- |
| **package** com.hoo.module;  **import** java.util.Random;  /\*\*  \* <b>function:</b>编写一个简单的WebService服务器端代码  \* **@author** hoojo  \* **@createDate** 2011-3-15 上午06:19:15  \* **@file** SimpleWebService.java  \* **@package** com.hoo.module  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** SimpleWebService {  **public** **int** randInt() {  **return** **new** Random().nextInt(100);  }  } |

编写Module接口的实现类，代码如下：

|  |
| --- |
| **package** com.hoo.module;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.context.ConfigurationContext;  **import** org.apache.axis2.description.AxisDescription;  **import** org.apache.axis2.description.AxisModule;  **import** org.apache.axis2.modules.Module;  **import** org.apache.neethi.Assertion;  **import** org.apache.neethi.Policy;  /\*\*  \* <b>function:</b>axis2模块，实现Module接口实现类  \* **@author** hoojo  \* **@createDate** 2011-3-15 上午03:22:24  \* **@file** CustomModule.java  \* **@package** com.hoo.module  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** CustomModule **implements** Module {  **public** **void** applyPolicy(Policy policy, AxisDescription axisDescription) **throws** AxisFault {  System.*out*.println("###############applyPolicy##############");  }  **public** **boolean** canSupportAssertion(Assertion assertion) {  System.*out*.println("##############canSupportAssertion##############");  **return** **true**;  }  **public** **void** engageNotify(AxisDescription axisDescription) **throws** AxisFault {  System.*out*.println("##############engageNotify#############");  }  **public** **void** init(ConfigurationContext ctx, AxisModule module) **throws** AxisFault {  System.*out*.println("##############init##############");  }  **public** **void** shutdown(ConfigurationContext ctx) **throws** AxisFault {  System.*out*.println("##############shutdown#############");  }  } |

编写实现Handler接口的实现类，代码如下：

|  |
| --- |
| **package** com.hoo.module;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.context.MessageContext;  **import** org.apache.axis2.description.HandlerDescription;  **import** org.apache.axis2.description.Parameter;  **import** org.apache.axis2.engine.Handler;  /\*\*  \* <b>function:</b>axis2模块Handler接口实现  \* **@author** hoojo  \* **@createDate** 2011-3-15 上午03:37:43  \* **@file** CustomHandler.java  \* **@package** com.hoo.module  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** CustomHandler **implements** Handler {    **private** String name;    **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getName() {  **return** **this**.name;  }    **public** **void** cleanup() {  System.*out*.println("###########cleanup###########");  }  **public** **void** flowComplete(MessageContext ctx) {  System.*out*.println("###########flowComplete###########");  System.*out*.println(ctx.getEnvelope().toString());  }  **public** HandlerDescription getHandlerDesc() {  System.*out*.println("###########getHandlerDesc###########");  **return** **null**;  }  **public** Parameter getParameter(String name) {  System.*out*.println("###########getParameter###########");  **return** **null**;  }  **public** **void** init(HandlerDescription handlerDescription) {  System.*out*.println("###########init###########");  }  **public** InvocationResponse invoke(MessageContext ctx) **throws** AxisFault {  System.*out*.println(ctx.getEnvelope().toString());  **return** InvocationResponse.*CONTINUE*;  }  } |

编写module.xml文件

|  |
| --- |
| <module name=*"customModule"* class=*"com.hoo.module.CustomModule"*>  <InFlow>  <handler name=*"InFlowLogHandler"* class=*"com.hoo.module.CustomHandler"*>  <order phase=*"customPhase"*/>  </handler>  </InFlow>  <OutFlow>  <handler name=*"OutFlowLogHandler"* class=*"com.hoo.module.CustomHandler"*>  <order phase=*"customPhase"*/>  </handler>  </OutFlow>    <InFaultFlow>  <handler name=*"FaultInFlowLogHandler"* class=*"com.hoo.module.CustomHandler"*>  <order phase=*"customPhase"*/>  </handler>  </InFaultFlow>    <OutFaultFlow>  <handler name=*"FaultOutFlowLogHandler"* class=*"com.hoo.module.CustomHandler"*>  <order phase=*"customPhase"*/>  </handler>  </OutFaultFlow>  </module> |

编写services.xml文件

|  |
| --- |
| <service name=*"CustomModuleService"*>  <description>使用CustomModule SimpleWebService模块</description>  <!-- 引用customModule模块 -->  <module ref=*"customModule"*/>  <parameter name=*"ServiceClass"*>  com.hoo.module.SimpleWebService  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  </messageReceivers>  </service> |

在[tomcat\_home]\webapps\axis2\WEB-INF\conf中axis2.xml文件中加入内容，在所有的<phaseOrder>标签中加入 <phase name="customPhase"/>

打包发布module，在c盘建立CustomModuleService，然后将CustomModule.class和CustomHandler.class以及类路径目录复制到该目录。然后将module.xml文件放到META-INF（没有新建）目录。

运行jar命令：jar cvf custom-module.mar .

将生成的custom-module.mar文件粘贴到[tomcat\_home] \webapps\axis2\WEB-INF\modules目录中

发布WebService，建立目录simpleWebService，将SimpleWebService.xml和类路径复制到该目录下，将services.xml复制到META-INF目录。

运行jar命令：jar cvf simple-service.aar .

将生成的simple-service.aar文件复制到[tomcat\_home] \webapps\axis2\WEB-INF\services目录下

然后重启tomcat服务。

客户端访问WebService代码

|  |
| --- |
| **package** com.hoo.service;  **import** javax.xml.namespace.QName;  **import** org.apache.axis2.AxisFault;  **import** org.apache.axis2.addressing.EndpointReference;  **import** org.apache.axis2.client.Options;  **import** org.apache.axis2.rpc.client.RPCServiceClient;  /\*\*  \* <b>function:</b>访问SimpleWebService  \* **@author** hoojo  \* **@createDate** 2011-3-15 上午07:26:08  \* **@file** SimpleWebServiceClient.java  \* **@package** com.hoo.service  \* **@project** Axis2WebService  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** SimpleWebServiceClient {  **public** **static** **void** main(String[] args) **throws** AxisFault {  String target = "http://localhost:8080/axis2/services/CustomModuleService";  RPCServiceClient client = **new** RPCServiceClient();  Options options = client.getOptions();  options.setManageSession(**true**);    EndpointReference epr = **new** EndpointReference(target);  options.setTo(epr);    QName qname = **new** QName("http://module.hoo.com", "randInt");  Object[] result = client.invokeBlocking(qname, **new** Object[] { **null** }, **new** Class[] { **int**.**class** });  System.*out*.println(result[0]);  }  } |

1. **使用SoapMonitar监视WebService的请求和响应信息**

使用soapmonitar可以监视到请求和响应的WebService的基本信息，可以讲响应在控制台的信息显示在applet控件上。

步骤如下：

1. 部署相关的applet和servlet

Axis2有自带的monitor模块，这里就不需要单独安装。在[tomcat\_home]\webapps\axis2\WEB-INF\lib目录中，找到axis2-soapmonitor-servlet-1.5.3.jar这个文件，将这个文件解压后，将里面的servlet的class文件放到[tomcat\_home]\webapps\axis2\WEB-INF\classes目录下。将applet的class放到[tomcat\_home]\webapps\axis2这个目录下即可。

1. 配置Servlet，这里配置的Servlet就包含上一步复制的servlet-class

在web.xml中增加如下配置

|  |
| --- |
| <servlet>  <servlet-name>SOAPMonitorService</servlet-name>  <servlet-class>org.apache.axis2.soapmonitor.servlet.SOAPMonitorService</servlet-class>  <init-param>  <param-name>SOAPMonitorPort</param-name>  <param-value>5001</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>SOAPMonitorService</servlet-name>  <url-pattern>/SOAPMonitor</url-pattern>  </servlet-mapping> |

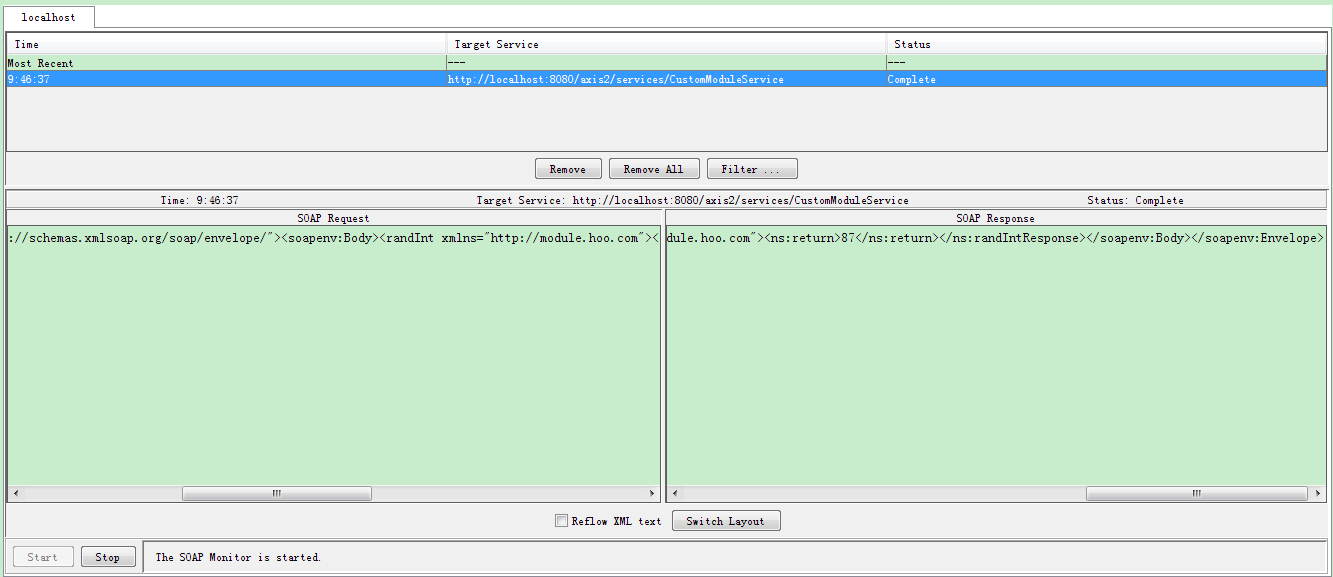
1. 在要监视的WebService的services.xml文件中关联soapmonitor，services.xml配置文件如下：

|  |
| --- |
| <service name=*"CustomModuleService"*>  <description>使用soapmonitor SimpleWebService模块</description>  <!-- 引用soapmonitor模块 -->  <module ref=*"soapmonitor"*/>  <parameter name=*"ServiceClass"*>  com.hoo.module.SimpleWebService  </parameter>  <messageReceivers>  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"*  class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"* />  <messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"*  class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />  </messageReceivers>  </service> |

将上次的WebService关联soapmonitor这个module，然后重新打包aar发布。

1. 请求<http://localhost:8080/axis2/SOAPMonitor>

可以看到Java的Applet，界面如下：



然后运行请求执行CustomModuleService这个service的时候，你就可一看到上面出现xml的内容。上面的内容是和控制台的同步输出的。