Web Service 开发指南

V 2.3

作 者

ProdigyWit (QQ：3562720)
Along (QQ：36224205)

版权说明

文档版本更新说明

版本号	发布时间	说明
V2.3	2009-03-06	加入 2.6、2.7 节——Client.NoSOAPAction 处理和服务端带头信息进行响应
V2.2	2009-02-16	加入 3.6 节——SOAP 头验证和通过 WSDL 规范进行服务端代码生成,Axis 读取头信息
V2.1	2008-12-16	加入 XFire 的使用 WSDL 生成客户端。
V2.0	2008-08-01	加入 XFire 和 CXF 使用，做为《Web Service 开发指南》发布。
V1.0	2008-07-15	做为《AXIS 开发指南》发布。

本文档参考并引用了以下著作

著作名	著作日期	作者
Axis Webservice 教程	未知	未知

目录

1.WebService 简介

1.1 WebService 介绍

WebService 让一个程序可以透明地调用互联网的程序，不用管具体的实现细节。只要WebService公开了服务接口，远程客户端就可以调用服务。WebService是基于http 协议的组件服务，WebService 是分散式应用程序的发展趋势。

1.2 WebService 的开源实现

WebService 更多是一种标准，而不是一种具体的技术。不同的平台，不同的语言大都提供
WebService 的开发实现。在JAVA领域，WebService的框架很多，例如：Axis1&2、XFire、CXF……。其中一个成熟实现是AXIS。AXIS应用比较广泛，而且资料相对也比较多。

2.AXIS 篇

2.1AXIS 使用

2.1 .1AXIS 的介绍

Axis（**A**pache e**X**tensible **I**nteraction **S**ystem）是一款开源的 WebService 运行引擎，它是 SOAP 协议的一个实现，其本身来源于 Apache 的另一个项目 Apache SOAP。Axis 分为 1.x 系列和 2 系列，两个系列体系结构和使用上有较大的区别，相对而言，Axis1.x 更加稳定，文档也比较齐全，因此本文内容以 Axis 1.x 系列最新版本 1.4 为基础。

2.1.2 AXIS 的下载

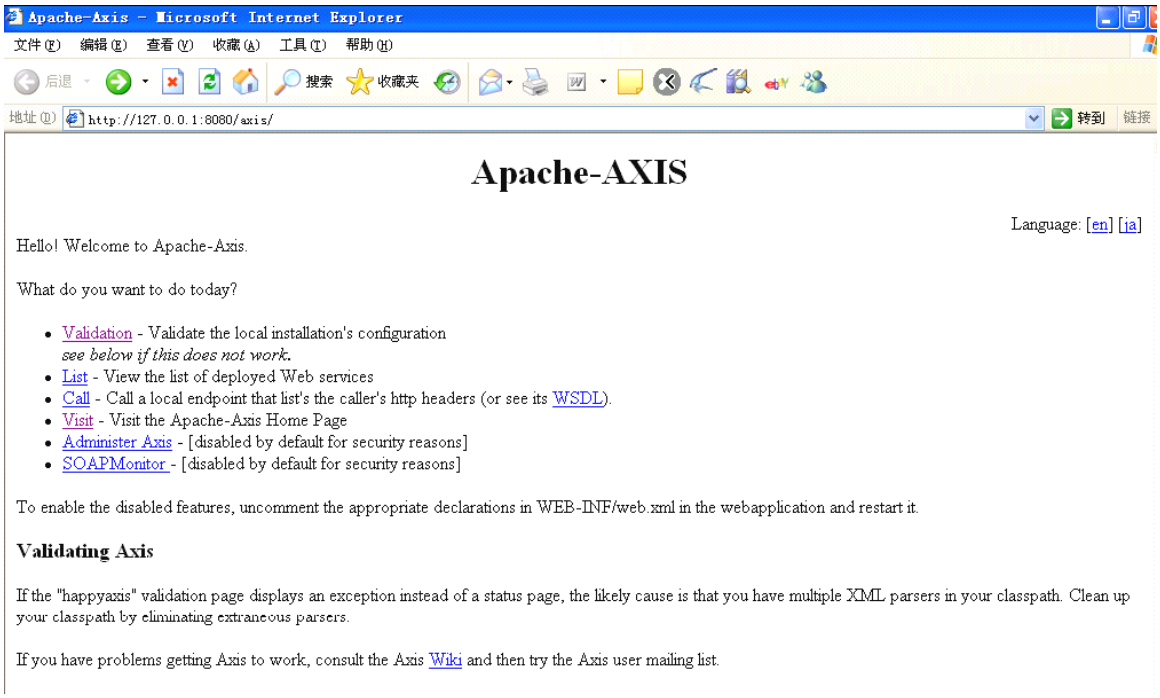
登陆http://www.apache.org/dyn/closer.cgi/ws/axis/1_4 站点，Axis 的版本是1.4，本文的例子都是基于该版本完成的。下载axis-bin-1_4.zip 文件，解压改文件，文件结构如下：

Docs：存放Axis 的说明文档。

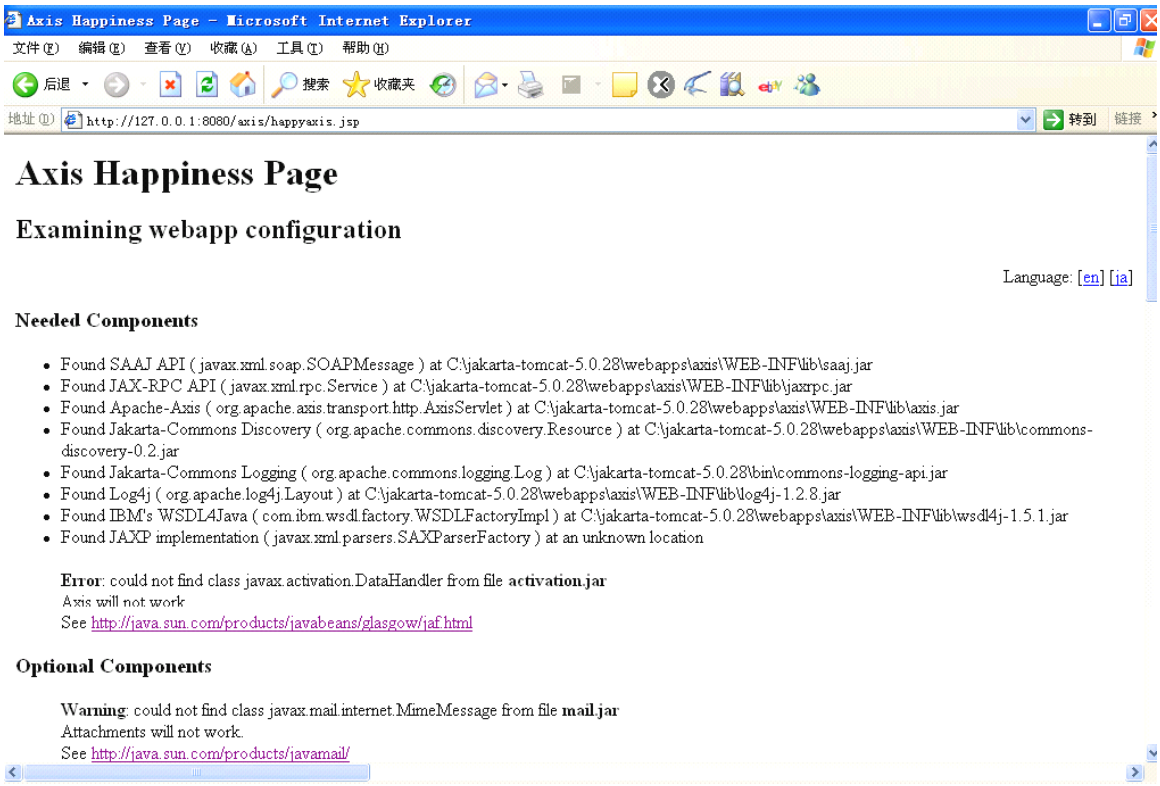
Lib：存放Axis 的二进制发布包。
Samples：存放利用Axis 发布的WebService 的示例代码。
Webapps：存放安装Axis 的基础应用。
Xmls：存放相关说明文档。

2.1.3 AXIS 的安装

安装Axis，首先必须保证系统已经正确安装web 服务器，本人使用的Tomcat5.0.28。也可使用其他应用服务器，例如：weblogic。Webapps下的Axis路径全部复制到Tomcat的Webapps下。然后打开浏览器：http://127.0.0.1:8080/axis。
如果出现如图所示界面则表示已经安装成功了。



然后点击上面页面的”Validation”连接，如图：

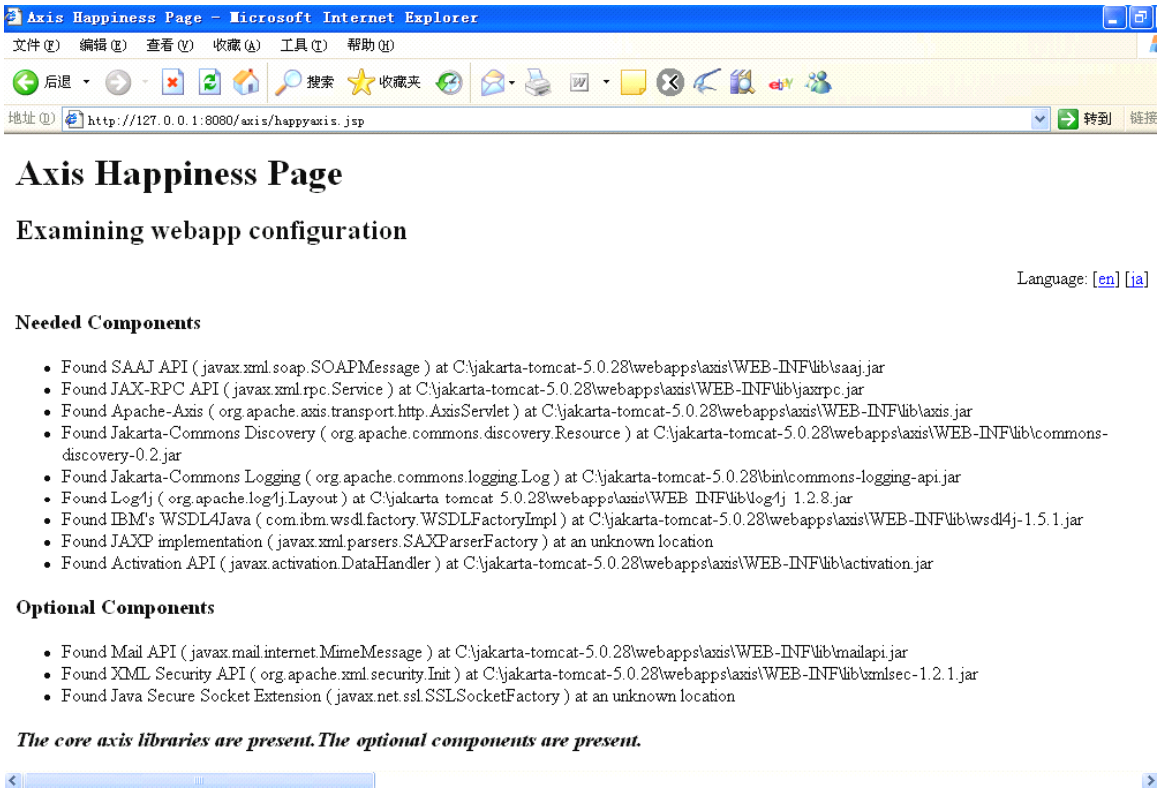


该页显示Axis 的必需组件还缺少一个没有安装，两个可选组件也没有安装，单击上面的提示：

必需组件：
http://java.sun.com/products/javabeans/glasgow/jaf.html

可选组件：
http://java.sun.com/products/javamail/
http://xml.apache.org/security/

分别下载：jaf-1_1-fr.zip，javamail-1_4.zip，xml-security-bin-1_2_1.zip包，然后把jaf-1_1-fr.zip里的Activation.jar，javamail-1_4.zip里的mail.jar，xml-security-bin-1_2_1.zip里的xmlsec-1.2.1.jar，xalan.jar都放到你的axis应用的WEB-INF/lib下然后在校验Axis，看到页面提示全部安装成功。



2.1.4 开发自己的 WebService

AXIS 提供了两种发布方式：

- 即时发布（Instant Deployment）
- 定制发布（ Custom Deployment）

即时发布提供了一种非常简单的WebService的发布方式，但是其中限制太多，因此在实际的开发中定制发布才是首选。这里也将会以定制发布为重点来介绍。

2.1.4.1 即时发布——JWS(Java Web Service)

即时发布提供了一种非常简单发布方式，发布者只要有Java源代码（也就是.java文件），然后把其后缀名改成jws（也就是Java Web Service的缩写）拷贝到%TOMCAT_HOME%\webapps\axis目录下即完成了所有的发布工作。AXIS的编译引擎会处理接下来的所有事情。下面是一段示例代码：

服务端：

Java代码：

```
/**
 * JWS方式WebService服务类
```

```
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-10
 */
public class HelloWorldJWS {

    public String test(String a, Integer b) {
        String result = "a=" + a + ", b=" + b;
        System.out.println("Received: " + result);

        return "Server Response OK, you send: " + result;
    }
}
```

把 HelloWorldJWS.java 文件改成 HelloWorldJWS.jws 然后拷贝到 %TOMCAT_HOME%\webapps\axis 目录下，启动 Tomcat 之后访问 http://localhost:8080/axis/HelloWorldJWS.jws如果能看到Click to see the WSDL这个超链接，就说明已经发布成功了。点击进去就可以看到这个WebService的WSDL描述文件。Server端的发布已经完成了，接下来就是编写Client端测试代码了。

客户端：

Java代码：

```
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 ClientJWS {

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            String url =
"http://127.0.0.1:8080/myaxis/HelloWorldJWS.jws";
            Service serv = new Service();
```

```
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);

            call.setOperationName(new QName(url, "test"));
            String result = (String) call.invoke(new
Object[]{"Quahilong", 100});

            System.out.println("result = " + result);
        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

测试代码很简单，如果熟悉java反射机制的朋友不用两分钟就能看明白。运行后客户端控制台出现以下运行结果。果然很简单吧，不过在这简单背后却是以牺牲灵活性为代价的。

运行结果：
客户端：

```
result = Server Response OK, you send: a=Quahilong, b=100
```

注意：假如你现在手里只有.class 或者一个 jar 包，jws 就不再能满足你的需求了，最要命的就是即时发布不支持带包的类，这点 AXIS 的用户手册上写的也很明白。

2.1.4.2 定制发布——WSDD

比起即时发布，定制发布更加烦琐也更复杂，但是换来的却是更大的灵活性，因此在实际项目中定制发布还是不二的选择。定制发布需要你自己编写一个WSDD（Web Service Deployment Descriptor）文件，其实就是一个XML描述文件，稍后会做出介绍。废话不多说，我们来看代码：

服务端：
Java代码：

```
package webservice.axis.wsdd;

public class HelloWorldWSDD {

    private int requestCount = 0;

    public String hello(String name) {
        requestCount++;
```

```
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + name);

        return "Hello " + name;
    }

    public Float add(Float a, float b) {
        requestCount++;
        String result = "a=" + a + ", b=" + b;
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + result);

        return a + b;
    }
}
```

一个带包的很简单的类，在 eclipse 下编译后按照包名拷到%TOMCAT_HOME%\webapps\axis\WEB-INF\classes目录下。
以这个类为例，拷贝完之后这个 HelloWorldWSDD.class 的路径就是%TOMCAT_HOME%\webapps\axis\WEBINF\classes\webservice\myaxis\wsdd。PS：如果嫌这样太麻烦，可以另外建一个Java Web工程用myeclipse的发布工具发布到Tomcat之后，整体一次性拷贝到WebSericve的工程中。接下来就需要编写发布文件deploy.wsdd。到%TOMCAT_HOME%\webapps\axis\WEB-INF目录下建立这个文件并在其中添加如下内容：

XML代码：

```
<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="webservice.axis.wsdd.HelloWorldWSDD"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="scope" value="request"/>
    </service>
</deployment>
```

简单的介绍下各个节点的含义，“HelloWorldWSDD”当然是这个WebService的名字，后面紧跟的java：RPC指的是服务类型。这里一共有有4种类型，分别是：RPC,Document,Wrapped和Message。有兴趣可以看下org.apache.axis.providers这个包和子包下面的类的API文档。之后的parameter节点第一个当然是指出具体的类，第二个从字面上也很好理解：允许调用的方法。这里的配置告诉引擎可以调用所有的public方法，当然你也可以自己指定。

编写完配置发布文件之后，cmd打开windows的控制台，进入：
%TOMCAT_HOME%\webapps\axis\WEB-INF 目录下键入如下命令：
java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient deploy.wsdd

之后控制台返回Processing file deploy.wsdd 和Done processing 这两段话即说明发布成功。(此时会在同级目录生成一个server-config.wsdd文件）在这里的AdminClient是AXIS提供的一个客户端管理工具。至于java.ext.dirs的含义可以去了解一下classloader和JVM类装载机制方面的知识，在这里就不多解释。还有一点要注意的是在发布的时候Tomcat服务必须处于启动状态，否则就会抛出一堆无法连接的异常信息。发布成功之后你可以通过访问http://localhost:8080/axis/servlet/AxisServlet 来查看你所有的定制发布的服务。

客户端代码：
Java代码：

```
package webservice.axis.wsdd;
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 ClientWSDD {

    public static void main(String[] args) {
        try {
            String url =
"http://127.0.0.1:8080/myaxis/services/HelloWorldWSDD";
            Service serv = new Service();
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);

            call.setOperationName(new QName(url, "hello"));
            String result = (String) call.invoke(new
Object[]{"Quhailong"});
            System.out.println("result = " + result);

            call.setOperationName(new QName(url, "add"));
            Float returnValue = (Float) call.invoke(new Object[]{new
Float(3.2), new Float(2.8)});
            System.out.println("returnValue = " + returnValue);

        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

运行测试代码，输出以下结果，说明发布成功。仔细观察下发现其实除了那个url之外，即时发布和定制发布的客户端调用代码基本上都是一样的。定制发布的URL可以在WSDL文件里找到。其实定制发布还有一些高级特性，这个就留到下一篇再说吧。

运行结果：

```
result = Hello Quhailong
returnValue = 6.0
```

2.1.4.3 取消发布一个 WebService

刚才讲了怎么发布一个WebService，但是如何取消没有却没有提。其实取消一个已经发布的WebService也是非常简单的，我们就刚才的HelloWorld来做例子吧。发布WebService的时候我们有一个deploy.wsdd文件，当然在取消发布的时候就会有一个undeploy.wsdd文件。这个文件的内容也很简单，xml的代码如下。

XML代码:

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

编写完这个xml文件之后，把它同样copy 到%TOMCAT_HOM\webapps\axis\WEB-INF目录下，然后CMD 打开控制台，在控制台输入一个我们很熟悉的命令：

```
java -Djava.ext.dirs=lib org.apache.axis.client.AdminClient undeploy.wsdd
```

运行之后得到如下结果说明取消发布成功。

运行结果：

客户端：

```
Processing file undeploy.wsdd
<Admin>Done processing</Admin>
```

2.2.WSDD 高级特性

2.2.1 WSDD 的高级特性

说完取消发布之后就来说一下AXIS 的一些高级特性，AXIS 在编写deploy.wsdd 这个文件时，每个<service>节点下面会有这样一个子节点。

XML代码：

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

这个节点配置着你的service object 也就是你WebService服务的那个object ，在后面的value 里可以有三个选项request、session、application。熟悉Jsp、Servlet、或者EJB里的SessionBean的朋友应该能很快能明白这个三个配置选项的含义。

- **requst**：这个选项会让AXIS为每一个SOAP的请求产生一个服务对象，可以想像如果这个webservice的对象足够复杂，而且SOAP的请求过多，这个选项是非常耗费服务器性能的。
- **session**：如果选择了session，程序就会给每个调用这个WebService的客户端创造一个服务对象。
- **application**：这个选项最彪悍，程序只会在内存里new出来一个服务对象，然后为所有WebService客户端服务。很显然这个选项不能储存客户端的一些个性化数据。所以在功能性上很多时候不能满足要求。

2.2.2 高级特性 Handler

接下来说一下Axis 的Handler 和Chain 机制，Handler 和Chain 是Axis 引擎提供的一个很强大的工具。假如现在客户有这样一个需求：需要记录某一个WebService被调用的次数。这个时候如果在service object 里去实现这个功能，不仅麻烦而且侵入了原有的程序，也会增加原有程序的不稳定性。有了Handler我们就能很容易的解决这个问题。我们先来编写handler
的代码。

服务端：
Java代码：

```
package webservice.axis.wsddhandler;

import org.apache.axis.AxisFault;
import org.apache.axis.MessageContext;
import org.apache.axis.handlers.BasicHandler;

/**
 * WebService的Handle类，可以在WebService每个方法被调用之前或者之后做一些事情。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-10
```

```
 */
public class HelloWorldHandler extends BasicHandler {

    private static final long serialVersionUID = 3562695953982109022L;

    private static long COUNT = 0L;

    private int requestCount = 0;
    /*
     * (non-Javadoc)
     *
     * @see
org.apache.axis.Handler#invoke(org.apache.axis.MessageContext)
     */
    public void invoke(MessageContext arg0) throws AxisFault {
        requestCount++;
        COUNT++;
        String status = (String) this.getOption("status");
        System.out.println("HelloWorldHandler's status is: " + status
+ ", COUNT=" + COUNT + ", HandlerRequestCount=" + requestCount);
    }
}
```

BasicHandler是一个抽象类，Axis提供了很多Handler的具体实现，BasicHandler只是其中最简单的一个。要实现一个自己的handler首先要从继承BasicHandler这个类开始并实现其中的invoke(MessageContext arg)这个方法。MessageContext可以看成是一个Axis的上下文，里面存储的是一些Axis 和WebService的基本信息。想了解的朋友可以看一下Axis的API。编写完Handler代码之后我们连编写发布文件。

XML代码：

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <handler name="HelloWorldHandler"
type="java:webservice.axis.wsddhandler.HelloWorldHandler">
        <parameter name="status" value="success"/>
    </handler>
    <service name="HelloWorldWSDDHandler" provider="java:RPC">
        <requestFlow>
            <handler type="HelloWorldHandler"/>
        </requestFlow>
        <parameter name="className"
value="webservice.axis.wsdd.HelloWorldWSDD"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="scope" value="request"/>
```

```
        <responseFlow>
            <handler type="HelloWorldHandler"/>
        </responseFlow>
    </service>
</deployment>
```

发布代码中有这样的一句，细心的朋友一定会发现：

```
<parameter name="status" value="success"/>
```

看完这句代码再对比一下Handler的实现代码中的一句，相信大多数人都能明白了。

Java代码：

```
String status = (String) this.getOption("status");
```

Handle r通过getOption(String)这个方法拿到了配置文件中我配置的属性值。而我们上述所做的所有工作对于原来的WebSerivce来说都是透明的，不会对侵入原有的程序当中。一个Handler可以被多个service所使用，通过<requestFlow>这个标签来引用到某一个service中，这里还要多提一句：既然是一个requestFlow，当然可以加多个Handler。接下来编写测试代码运行。

客户端：

Java代码：

```
package webservice.axis.wsddhandler;
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 ClientWSDDHandler {

    public static void main(String[] args) {
        try {
            String url =
"http://127.0.0.1:8080/myaxis/services/HelloWorldWSDDHandler";
            Service serv = new Service();
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);

            call.setOperationName(new QName(url, "hello"));
            String result = (String) call.invoke(new
Object[]{"Quhailong"});
```

```
            System.out.println("result = " + result);

            call.setOperationName(new QName(url, "add"));
            Float returnValue = (Float) call.invoke(new Object[]{new
Float(3.2), new Float(2.8)});
            System.out.println("returnValue = " + returnValue);

        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

在本地应用服务器上会打出如下语句说明测试成功，而且handler是配置在requestFlow标签中所以这段代码会在service代码之前先执行。如果要在service之后执行，应该配置在<responseFlow>标签中。

运行结果：
客户端：

```
result = Hello Quhailong
returnValue = 6.0
```

服务端：

```
HelloWorldHandler's status is: success, COUNT=1, requestCount=1
requestCount=1
Received: Quhailong
HelloWorldHandler's status is: success, COUNT=2, requestCount=2
HelloWorldHandler's status is: success, COUNT=3, requestCount=3
requestCount=1
Received: a=3.2, b=2.8
HelloWorldHandler's status is: success, COUNT=4, requestCount=4
```

2.2.4 高级特性 Chain

介绍完了Handler再来介绍Chain。从Chain的字面意思就能猜到他实现的一连串Handler的功能。假如某个service 需要不止一个Handler，或者要根据Client 的情况来选择需要那些Handler。特别是后一个需求，我们无法用一个或者几个Handler来解决，这个时候我们就需要<Chain>来实现了。我们先再编写一个Handler，加上之前的那个Handler我们来组成一条锁链。

服务端：

Java代码：

```
package webservice.axis.wsddchain;

import org.apache.axis.AxisFault;
import org.apache.axis.MessageContext;
import org.apache.axis.handlers.BasicHandler;

/**
 * WebService的Handle类，可以在WebService每个方法被调用之前或者之後做一些事情。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-10
 */
public class HelloWorldHandler2 extends BasicHandler {

    private static final long serialVersionUID = 772997192033702477L;

    private static long COUNT = 0L;

    private int requestCount = 0;
    /*
     * (non-Javadoc)
     *
     * @see
org.apache.axis.Handler#invoke(org.apache.axis.MessageContext)
     */
    public void invoke(MessageContext arg0) throws AxisFault {
        requestCount++;
        COUNT++;
        String status = (String) this.getOption("status");
        System.out.println("This is an other handler.
HelloWorldHandler2's status is: " + status + ", COUNT=" + COUNT + ",
HandlerRequestCount=" + requestCount);
    }
}
```

之后我们编写 Chain 的代码.

服务端：

Java 代码：

```
package webservice.axis.wsddchain;
```

```
import org.apache.axis.SimpleChain;

import webservice.axis.wsddhandler.HelloWorldHandler;

/**
 * WebService的Handle链，可以在WebService每个方法被调用之前或之後执行多个
Handler。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-10
 */
public class HelloWorldChain extends SimpleChain {

    private static final long serialVersionUID = -5103267089035176l7L;

    public HelloWorldChain() {
        HelloWorldHandler handler1 = new HelloWorldHandler();
        HelloWorldHandler2 handler2 = new HelloWorldHandler2();

        this.addHandler(handler1);
        this.addHandler(handler2);
    }
}
```

在Chain 的构造函数中，把我要的两个Handler用addHandler()方法加载进去。之后我们来编写发布文件。<chain>和<handler>元素有些许不同在这里有必要多句嘴：<chain>元素中的子元素只允许是<handler>或者<chain>。后者也就是允许在“锁链”里再嵌套“锁链”，在这里就拿嵌套<handler>来举例，他同样有两种方式来实现。

第一种是直接包含<handler>:

```
<chain name="HelloWorldChain">
    <handler type="java:webservice.axis.wsddchain.HelloWorldChain"/>
</chain>
```

第二种是引用别的<handler>:

```
<handler  name="myHandler"
type="java:webservice.axis.wsddchain.HelloWorldChain"/>
<chain name="HelloWorldChain">
    <handler type="myHandler"/>
</chain>
```

因为我们这里的HelloWorldChain并没有由BasicHandler来实现，而是由继承SimpleChain这个类来实现，严格意义上讲，SimpleChain也可以算是一个Handler，因为SimpleChain也是从
BasicHandler继承而来，他同样实现了invoke()这个方法。下面回归正题，来看我们的发布代码。

XML代码:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <chain name="HelloWorldChain">
        <handler
type="java:webservice.axis.wsddchain.HelloWorldChain"/>
    </chain>
    <service name="HelloWorldWSDDChain" provider="java:RPC">
        <requestFlow>
            <chain type="HelloWorldChain"/>
        </requestFlow>
        <parameter name="className"
value="webservice.axis.wsdd.HelloWorldWSDD"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="scope" value="request"/>
    </service>
</deployment>
```

从新发布WebService之后，运行我们的测试代码。

客户端:
Java代码:

```
package webservice.axis.wsddchain;
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 ClientWSDDChain {
    public static void main(String[] args) {
        try {
            String url =
"http://127.0.0.1:8080/myaxis/services/HelloWorldWSDDChain";
            Service serv = new Service();
```

```
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);

            call.setOperationName(new QName(url, "hello"));
            String result = (String) call.invoke(new
Object[]{"Quhailong"});
            System.out.println("result = " + result);

            call.setOperationName(new QName(url, "add"));
            Float returnValue = (Float) call.invoke(new Object[]{new
Float(3.2), new Float(2.8)});
            System.out.println("returnValue = " + returnValue);

        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

结果如下：

运行结果：
客户端：

```
result = Hello Quhailong
returnValue = 6.0
```

服务端：

```
HelloWorldHandler's status is: null, COUNT=7, requestCount=3
This is an other handler. HelloWorldHandler2's status is: null, COUNT=3,
requestCount=3
requestCount=1
Received: Quhailong
HelloWorldHandler's status is: null, COUNT=8, requestCount=4
This is an other handler. HelloWorldHandler2's status is: null, COUNT=4,
requestCount=4
requestCount=1
Received: a=3.2, b=2.8
```

2.2.5 传递复杂对象

2.2.5.1 List、Map、数组和自定义对象

在上面介绍Axis的文章里，我们做了一个简单的WebService，我们client side传递了String和int类型的数据给service object。Service 处理之后返回处理结果给Client。对于大多数需求，那个demo显然已经足够应付了。但是如果client端需要传输一个对象给server，那么那个demo 就显得力不从心了。Axis中提供了远程传输对象的方法，通过那些方法我们同样可以随心的传递自己的对象。

先看下面的 JAVABEAN
这个对象是服务端的对象：
Java 代码：

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * 服务端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;
```

```
/** 邮编 */
private String postalCode;

private String[] array;

private List<Integer> list;

private boolean isExist;

public Address() {
list = new ArrayList<Integer>();
list.add(1);
list.add(2);
list.add(3);
}

public Integer getIdentifier() {
    return identifier;
}

public void setIdentifier(Integer identifier) {
    this.identifier = identifier;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}
```

```
public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}

public void setPostalCode(String postalCode) {
    this.postalCode = postalCode;
}

public String[] getArray() {
    return array;
}

public void setArray(String[] array) {
    this.array = array;
}

public boolean isExist() {
    return isExist;
}

public void setExist(boolean isExist) {
    this.isExist = isExist;
}

public List<Integer> getList() {
    return list;
}

public void setList(List<Integer> list) {
    this.list = list;
}
```

```
}
```

注意：所有要传递的对象都要是可序化的。

服务端的业务：
Java 代码：

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import webservice.axis.wsddselfobj.servermodel.Address;

/**
 * 提供List的WebService业务
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class AddressManager {

    public List<Address> getAddressList() {
        List<Address> returnList = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnList.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
```

```
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        returnList.add(address);
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});

        return returnList;
    }

    public List<Address> setAddressList(List<Address> list) {
        return list;
    }

    public Map<Integer, Address> getAddressMap() {
        Map<Integer, Address> returnMap = new HashMap<Integer,
Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnMap.put(address.getIdentifier(), address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        returnMap.put(address.getIdentifier(), address);

        return returnMap;
    }
```

```
    public Map<Integer, Address> setAddressMap(Map<Integer, Address>
map) {
        return map;
    }
}
```

下面的客户端和客户端的 JAVABEAN

客户端 JAVABEAN：
Java 代码：

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * 客户端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;

    /** 邮编 */
    private String postalCode;
```

```
private String[] array;

private List<Integer> list;

private boolean isExist;

public Address() {
list = new ArrayList<Integer>();
list.add(1);
list.add(2);
list.add(3);
}

public Integer getIdentifier() {
    return identifier;
}

public void setIdentifier(Integer identifier) {
    this.identifier = identifier;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}
```

```
    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public String getPostalCode() {
        return postalCode;
    }

    public void setPostalCode(String postalCode) {
        this.postalCode = postalCode;
    }

    public String[] getArray() {
        return array;
    }

    public void setArray(String[] array) {
        this.array = array;
    }

    public boolean isExist() {
        return isExist;
    }

    public void setExist(boolean isExist) {
        this.isExist = isExist;
    }

    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

}
```

客户端：

Java 代码：

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
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 webservice.axis.wsddselfobj.clientmodel.Address;

/**
 * 获得WebService的返回List/Map对象的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
@SuppressWarnings("unchecked")
public class ClientAddress {

    public static void getServerList() {
        String url =
"http://127.0.0.1:8080/axis/services/AddressManager";
        Service service = new Service();

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");

            call.registerTypeMapping(Address.class, qn,
                    new
org.apache.axis.encoding.ser.BeanSerializerFactory(
                            Address.class, qn),
```

```
                    new
org.apache.axis.encoding.ser.BeanDeserializerFactory(
                            Address.class, qn));
            call.setTargetEndpointAddress(new java.net.URL(url));
            call.setOperationName(new QName("AddressManager",
"getAddressList"));

            call.setReturnClass(ArrayList.class);
            Object[] sss = null;

            List<Address> list = (ArrayList) call.invoke(sss);

            System.out.println("List size: " + list.size());

            for (Iterator<Address> iter = list.iterator();
iter.hasNext();) {

                Address address = iter.next();
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void setServerList() {
        String url =
"http://127.0.0.1:8080/axis/services/AddressManager";
        Service service = new Service();
```

```
        List<Address> list = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});
        list.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        list.add(address);

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");

            call.registerTypeMapping(Address.class, qn,
                    new
org.apache.axis.encoding.ser.BeanSerializerFactory(
                            Address.class, qn),
                    new
org.apache.axis.encoding.ser.BeanDeserializerFactory(
                            Address.class, qn));
            call.setTargetEndpointAddress(new java.net.URL(url));
            call.setOperationName(new QName("AddressManager",
"setAddressList"));

            call.setReturnClass(ArrayList.class);
            call.addParameter("list", XMLType.XSD_ANYTYPE,
ParameterMode.IN);
            list = (ArrayList) call.invoke(new Object[]{list});
```

```
            System.out.println("List size: " + list.size());

            for (Iterator<Address> iter = list.iterator();
iter.hasNext();) {

                address = iter.next();
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void getServerMap() {
        String url =
"http://127.0.0.1:8080/axis/services/AddressManager";
        Service service = new Service();

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");

            call.registerTypeMapping(Address.class, qn,
                    new
org.apache.axis.encoding.ser.BeanSerializerFactory(
                            Address.class, qn),
                    new
org.apache.axis.encoding.ser.BeanDeserializerFactory(
```

```
                        Address.class, qn));
            call.setTargetEndpointAddress(new java.net.URL(url));
            Object[] sss = null;

            call.setOperationName(new QName("AddressManager",
"getAddressMap"));
            call.setReturnClass(HashMap.class);

            Map<Integer, Address> map = (Map) call.invoke(sss);

            System.out.println("Map size: " + map.size());

            for (Iterator<Integer> iter = map.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                Address address = map.get(key);
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void setServerMap() {
        String url =
"http://127.0.0.1:8080/axis/services/AddressManager";
        Service service = new Service();

        Map<Integer, Address> map = new HashMap<Integer, Address>();
```

```
        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});
        map.put(address.getIdentifier(), address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        map.put(address.getIdentifier(), address);

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");

            call.registerTypeMapping(Address.class, qn,
                    new
org.apache.axis.encoding.ser.BeanSerializerFactory(
                            Address.class, qn),
                    new
org.apache.axis.encoding.ser.BeanDeserializerFactory(
                            Address.class, qn));
            call.setTargetEndpointAddress(new java.net.URL(url));

            call.setOperationName(new QName("AddressManager",
"setAddressMap"));
            call.setReturnClass(HashMap.class);

            call.addParameter("list", XMLType.XSD_ANYTYPE,
ParameterMode.IN);
            map = (Map) call.invoke(new Object[]{map});
```

```
            System.out.println("Map size: " + map.size());

            for (Iterator<Integer> iter = map.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                address = map.get(key);
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        getServerList();
        getServerMap();
        setServerList();
        setServerMap();
    }
}
```

以上是客户端和服务端的代码，现在主要介绍一下关于 WSDD，代码如下：

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="AddressManager" provider="java:RPC">
        <parameter name="className"
value="webservice.axis.wsddselfobj.AddressManager"/>
    <parameter name="allowedMethods" value="*"/>
```

```
    <!-- qname里的Address要和程序里的new QName("urn:AddressManager",
"Address")里的后面的值相同 -->
    <beanMapping qname="myNSD:Address"
xmlns:myNSD="urn:AddressManager"
languageSpecificType="java:webservice.axis.wsddselfobj.servermodel.Ad
dress"/>
    </service>
</deployment>
```

唯一不同的就是多了<beabMapping>这个节点。QName表示XML 规范中定义的限定名称，他由名称空间URI、本地部分和前缀组成。除了本地部分其他都不是必须的,另外QName是不可变的。xmlns 后面的myNS是必须的。具体根据前面所指定的qname 来决定。之后可以随意命名一个namespace。最后的languageSpecificType 指定的是你传递的对象类型。第一个属性的本地部分和第二个节点你自定义的命名空间会组成一个新的QName，并将你要传输的对象mapping上去。

注意：在上面编写客户端的时候要注意几个问题：

- 如果你有返回值的话一定要设置你的返回值类型。
- 如果服务器上的方法有参数一定要指定你的参数类型。
- 一定要拷贝 wsdl4j-1.5.1.jar 否则会报错，找不到你的转换类型。

运行结果：
客户端：

```
List size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
Map size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
List size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
Map size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
```

2.2.5.2 带内部类的自定义对象

经过上面的了解我们对大部分的 JAVABEAN 对象有了了解，像对返回类型 List、Map。List 里有 JAVABEAN 等这些都有了相应的解决方案。下面介绍一下特殊的情况，就是对内部类的使用废话少说来点代码一般就知道了：

服务端的 JAVABEAN：
Java 代码：

```
package webservice.axis.wsddselfobj.servermodel;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * 服务端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;
```

```
    /** 邮编 */
    private String postalCode;

    private String[] array;

    private List<Integer> list;

    private boolean isExist;

    private InnerClass innC;

    public static class InnerClass implements Serializable {

    private static final long serialVersionUID = -
2330738090948448510L;

        private String innerName;

        public String getInnerName() {
            return innerName;
        }

        public void setInnerName(String innerName) {
            this.innerName = innerName;
        }

    }

    public Address() {
    list = new ArrayList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);
    innC = new InnerClass();
    innC.setInnerName("My Inner name");
    }

    public InnerClass getInnC() {
        return innC;
    }

    public void setInnC(InnerClass innC) {
        this.innC = innC;
    }
```

```
public Integer getIdentifier() {
    return identifier;
}

public void setIdentifier(Integer identifier) {
    this.identifier = identifier;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}
```

```
    public void setPostalCode(String postalCode) {
        this.postalCode = postalCode;
    }

    public String[] getArray() {
        return array;
    }

    public void setArray(String[] array) {
        this.array = array;
    }

    public boolean isExist() {
        return isExist;
    }

    public void setExist(boolean isExist) {
        this.isExist = isExist;
    }

    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

}
```

这个类需要值得注意的是 2 点：

- 内部类必须是可序化的。
- 内部类必须是静态的，否则不能被序列化。

WebService 业务：
Java 代码

```
package webservice.axis.wsddselfobj;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
import webservice.axis.wsddselfobj.servermodel.Address;

/**
 * 提供复杂对象的WebService业务
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class AddressManager {

    public List<Address> getAddressList() {
        List<Address> returnList = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnList.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        returnList.add(address);
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});

        return returnList;
    }

    public List<Address> setAddressList(List<Address> list) {
```

```
        return list;
    }

    public Map<Integer, Address> getAddressMap() {
        Map<Integer, Address> returnMap = new HashMap<Integer,
Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnMap.put(address.getIdentifier(), address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        returnMap.put(address.getIdentifier(), address);

        return returnMap;
    }

    public Map<Integer, Address> setAddressMap(Map<Integer, Address>
map) {
        return map;
    }
}
```

上面的业务写法就不在说任何说明了。

客户端的 JAVABEAN
Java 代码：

```
package webservice.axis.wsddselfobj.clientmodel;
```

```java
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * 客户端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;

    /** 邮编 */
    private String postalCode;

    private String[] array;

    private List<Integer> list;

    private boolean isExist;

    private InnerClass innC;

    public static class InnerClass implements Serializable {
```

```
    private static final long serialVersionUID = -
2330738090948448510L;

        private String innerName;

        public String getInnerName() {
            return innerName;
        }

        public void setInnerName(String innerName) {
            this.innerName = innerName;
        }

    }

    public Address() {
    list = new ArrayList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);
    innC = new InnerClass();
    innC.setInnerName("My Inner name");
    }

    public InnerClass getInnC() {
        return innC;
    }

    public void setInnC(InnerClass innC) {
        this.innC = innC;
    }

    public Integer getIdentifier() {
        return identifier;
    }

    public void setIdentifier(Integer identifier) {
        this.identifier = identifier;
    }

    public String getAddress() {
        return address;
    }
```

```
public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}

public void setPostalCode(String postalCode) {
    this.postalCode = postalCode;
}

public String[] getArray() {
    return array;
}

public void setArray(String[] array) {
    this.array = array;
}
```

```
    public boolean isExist() {
        return isExist;
    }

    public void setExist(boolean isExist) {
        this.isExist = isExist;
    }

    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

}
```

客户端调用：
Java 代码：

```
package webservice.axis.wsddselfobj;

import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
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 webservice.axis.wsddselfobj.clientmodel.Address;

/**
```

```
 * 获得WebService的返回List/Map等负责对象的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
@SuppressWarnings("unchecked")
public class ClientAddress {

    public static void getServerList() {
        String url =
"http://127.0.0.1:8080/myaxis/services/AddressManager";
        Service service = new Service();

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");
            QName qn1 = new QName("urn:AddressInnerClass",
"myNSD:InnerClass");

            call.registerTypeMapping(Address.class, qn,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
            call.registerTypeMapping(Address.InnerClass.class, qn1,
BeanSerializerFactory.class, BeanDeserializerFactory.class);

            call.setTargetEndpointAddress(new java.net.URL(url));
            call.setOperationName(new QName("AddressManager",
"getAddressList"));

            call.setReturnClass(ArrayList.class);
            Object[] sss = null;

            List<Address> list = (ArrayList) call.invoke(sss);

            System.out.println("List size: " + list.size());

            for (Iterator<Address> iter = list.iterator();
iter.hasNext();) {

                Address address = iter.next();
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
```

```
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist()
                        + " innerClass.name: " +
address.getInnC().getInnerName());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void setServerList() {
        String url =
"http://127.0.0.1:8080/myaxis/services/AddressManager";
        Service service = new Service();

        List<Address> list = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});
        list.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
```

```
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        list.add(address);

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");
            QName qn1 = new QName("urn:AddressInnerClass",
"myNSD:InnerClass");

            call.registerTypeMapping(Address.class, qn,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
            call.registerTypeMapping(Address.InnerClass.class, qn1,
BeanSerializerFactory.class, BeanDeserializerFactory.class);

            call.setTargetEndpointAddress(new java.net.URL(url));
            call.setOperationName(new QName("AddressManager",
"setAddressList"));

            call.setReturnClass(ArrayList.class);
            call.addParameter("list", XMLType.XSD_ANYTYPE,
ParameterMode.IN);
            list = (ArrayList) call.invoke(new Object[]{list});

            System.out.println("List size: " + list.size());

            for (Iterator<Address> iter = list.iterator();
iter.hasNext();) {

                address = iter.next();
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist()
                        + " innerClass.name: " +
address.getInnC().getInnerName());
```

```
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void getServerMap() {
        String url =
"http://127.0.0.1:8080/myaxis/services/AddressManager";
        Service service = new Service();

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");
            QName qn1 = new QName("urn:AddressInnerClass",
"myNSD:InnerClass");

            call.registerTypeMapping(Address.class, qn,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
            call.registerTypeMapping(Address.InnerClass.class, qn1,
BeanSerializerFactory.class, BeanDeserializerFactory.class);

            call.setTargetEndpointAddress(new java.net.URL(url));
            Object[] sss = null;

            call.setOperationName(new QName("AddressManager",
"getAddressMap"));
            call.setReturnClass(HashMap.class);

            Map<Integer, Address> map = (Map) call.invoke(sss);

            System.out.println("Map size: " + map.size());

            for (Iterator<Integer> iter = map.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                Address address = map.get(key);
```

```
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
                        + " isExist: " + address.isExist()
                        + " innerClass.name: " +
address.getInnC().getInnerName());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void setServerMap() {
        String url =
"http://127.0.0.1:8080/myaxis/services/AddressManager";
        Service service = new Service();

        Map<Integer, Address> map = new HashMap<Integer, Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});
        map.put(address.getIdentifier(), address);

        address = new Address();
        address.setIdentifier(2);
```

```
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        map.put(address.getIdentifier(), address);

        try {
            Call call = (Call) service.createCall();
            QName qn = new QName("urn:AddressManager", "Address");
            QName qn1 = new QName("urn:AddressInnerClass",
"myNSD:InnerClass");

            call.registerTypeMapping(Address.class, qn,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
            call.registerTypeMapping(Address.InnerClass.class, qn1,
BeanSerializerFactory.class, BeanDeserializerFactory.class);

            call.setTargetEndpointAddress(new java.net.URL(url));

            call.setOperationName(new QName("AddressManager",
"setAddressMap"));
            call.setReturnClass(HashMap.class);

            call.addParameter("list", XMLType.XSD_ANYTYPE,
ParameterMode.IN);
            map = (Map) call.invoke(new Object[]{map});

            System.out.println("Map size: " + map.size());

            for (Iterator<Integer> iter = map.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                address = map.get(key);
                System.out.println("id号: " + address.getIdentifier()
                        + " address: " + address.getAddress()
                        + " city: " + address.getCity()
                        + " country: " + address.getCountry()
                        + " postalCode: " + address.getPostalCode()
                        + " province: " + address.getProvince()
                        + " array: " + address.getArray()[0]
                        + " list: " + address.getList()
```

```
                    + " isExist: " + address.isExist()
                    + " innerClass.name: " +
address.getInnC().getInnerName());
            }
        }
        catch (ServiceException e) {
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        getServerList();

        getServerMap();

        setServerList();

        setServerMap();
    }
}
```

这个地方需要啰嗦几句：

关于 Address 这个类由于他里面有一个内部类，但是这个内部类里并不是 JAVA 的基本类型，所以在你客户端调用的时候你需要注册一下，如：

```
QName qn = new QName("urn:AddressManager", "Address");
QName qn1 = new QName("urn:AddressInnerClass", "myNSD:InnerClass");

call.registerTypeMapping(Address.class, qn,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
call.registerTypeMapping(Address.InnerClass.class, qn1,
BeanSerializerFactory.class, BeanDeserializerFactory.class);
```

大家在这里看见的写法和上面的写法有点区别，其实这是一个简易写法，所表达的意思是相同的，都是表示这个类可以序列化和反序列化。

最后就是关于 WSDD 的部署代码如下：

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="AddressManager" provider="java:RPC">
        <parameter name="className"
value="net.along.webservice.axis.wsddselfobj.AddressManager"/>
    <parameter name="allowedMethods" value="*"/>
    <!-- qname里的Address要和程序里的new QName("urn:AddressManager",
"Address")里的后面的值相同 -->
    <beanMapping qname="myNSD:Address"
xmlns:myNSD="urn:AddressManager"
languageSpecificType="java:net.along.webservice.axis.wsddselfobj.serv
ermodel.Address"/>
    <beanMapping qname="myNSDP:InnerClass"
xmlns:myNSDP="urn:AddressInnerClass"
languageSpecificType="java:webservice.axis.wsddselfobj.servermodel.Ad
dress$InnerClass"/>
    <parameter name="scope" value="request"/>
    </service>
</deployment>
```

这里可以看书 Address 里的内部类也需要做映射。

运行结果：
客户端：

```
List size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
innerClass.name: My Inner name
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
innerClass.name: My Inner name
Map size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
innerClass.name: My Inner name
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
innerClass.name: My Inner name
List size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
innerClass.name: My Inner name
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
```

```
province: Beijing array: A list: [1, 2, 3] isExist: true
innerClass.name: My Inner name
Map size: 2
id号: 1 address: Haidian city: BeiJing country: China postalCode: 100081
province: Beijing array: 1 list: [1, 2, 3] isExist: false
innerClass.name: My Inner name
id号: 2 address: Chaoyang city: BeiJing country: China postalCode: 100081
province: Beijing array: A list: [1, 2, 3] isExist: true
innerClass.name: My Inner name
```

2.2.5.3 RMI 方式远程调用

到目前为止所有例子都是先提供一个类然后在给出对应的实现，下面介绍一种类似RMI的实现例子给大家参考，来看代码：

提供服务的接口：
Java 代码：

```
package webservice.axis.wsdd;

/**
 * 本地实现一个Remote接口，其中包含远程WS的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程WS对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-14
 */
public interface IHelloWorldWSDD extends java.rmi.Remote {

    public String hello(String name);

    public Float add(Float a, float b);
}
```

WebService 服务端：
Java 代码：

```
package webservice.axis.wsdd;

public class HelloWorldWSDD {
```

```
    private int requestCount = 0;

    public String hello(String name) {
        requestCount++;
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + name);

        return "Hello " + name;
    }

    public Float add(Float a, float b) {
        requestCount++;
        String result = "a=" + a + ", b=" + b;
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + result);

        return a + b;
    }
}
```

通过上面的代码可以看出服务端和之前我们写的代码是一样的，没有任何变化。客户端接口提供方法和参数完全和服务端提供的一样，这样你可以根据接口的方法来操作服务端的方法（服务端不需要实现这个接口，这个接口对服务端是透明的）。这样可以给你带来更方便的开发，当你要提供 WebService 服务的时候你可以不用修改你的服务端代码直接以接口形式提供就可以了。

客户端的实现：

Java 代码：

```
package webservice.axis.wsdd;
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 ClientWSDD {

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

```
"http://127.0.0.1:8080/myaxis/services/HelloWorldWSDD";
            Service serv = new Service();
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);

            call.setOperationName(new QName(url, "hello"));
            //String result = (String) call.invoke(new
Object[]{"Quhailong"});
            //System.out.println("result = " + result);
            IHelloWorldWSDD remoteRef = (IHelloWorldWSDD)
serv.getPort(url, IHelloWorldWSDD.class);
            String result = remoteRef.hello("Quhailong");
            System.out.println("result = " + result);
            result = remoteRef.hello("Quhailong");
            System.out.println("result = " + result);

            call.setOperationName(new QName(url, "add"));
            Float returnValue = (Float) call.invoke(new Object[]{new
Float(3.2), new Float(2.8)});
            System.out.println("returnValue = " + returnValue);

        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

WSDD 代码：

```
<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="webservice.axis.wsdd.HelloWorldWSDD"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="scope" value="request"/>
    </service>
</deployment>
```

在这里顺便提一下如果要是 HelloWorldWSDD 只想暴露 2 个方法而不是全部，可以使用 `<parameter name="allowedMethods" value="方法 1,方法 2"/>`这样来达到效果(其中方法 1&2 是用英文的逗号隔开的)。

2.2.6 抛出异常

上一篇介绍了如果在Server 和Client端传递一个自己的对象。有些人也许会问传递异常行不行？答案是可以。只不过传递异常的配置要稍微复杂一些。下面我用代码来说明下：

服务端的异常类：
Java代码：

```
package webservice.axis.wsddexception;

import java.rmi.RemoteException;

/**
 * 服务器端的异常
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class ServerException extends RemoteException {

    private static final long serialVersionUID = -
8703656417727568771L;

    private String errMSG = "";

    public ServerException() {
        System.out.println("Server Exception!");
    }

    public void printErrorDescription() {
        System.out.println(errMSG);
    }

    public String getErrMSG() {
        return errMSG;
    }

    public void setErrMSG(String errMSG) {
        this.errMSG = errMSG;
    }
}
```

提供的WebService代码：
Java代码

```
package webservice.axis.wsddexception;

/**
 * 产生异常的方法测试
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class ExceptionCreater {

    public void getException() throws ServerException {
        ServerException se = new ServerException();
        se.setErrMSG("Server Side Self Exception!");

        throw se;
    }
}
```

客户端的异常：
Java代码：

```
package webservice.axis.wsddexception;

import java.rmi.RemoteException;

/**
 * 客户器端的异常
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class ClientException extends RemoteException {

    private static final long serialVersionUID = -
8703656417727568771L;
```

```
    private String errMSG = "Client Side Self Exception!";

    public ClientException() {
        System.out.println("Client Exception!");
    }

    public void printErrorDescription() {
        System.out.println(errMSG);
    }

    public String getErrMSG() {
        return errMSG;
    }

    public void setErrMSG(String errMSG) {
        this.errMSG = errMSG;
    }
}
```

客户端代码

Java 代码：

```
package webservice.axis.wsddexception;

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;

/**
 * 获得WebService的返回异常的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class ClientExceptionReceiver {

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

```
            String url =
"http://127.0.0.1:8080/myaxis/services/ExceptionCreater";
            Service serv = new Service();
            Call call = (Call) serv.createCall();

            QName qn = new QName("urn:CustomerFault",
"ServerException");

            call.registerTypeMapping(ClientException.class, qn,
                  new
org.apache.axis.encoding.ser.BeanSerializerFactory(
                        ClientException.class, qn),
                  new
org.apache.axis.encoding.ser.BeanDeserializerFactory(
                        ClientException.class, qn));

            call.setTargetEndpointAddress(url);
            Object[] param = null;

            call.setOperationName(new QName(url, "getException"));
            System.out.println("Will Catch Exception");
            String result = (String) call.invoke(param);
            System.out.println("result = " + result);

        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            System.out.println("Catch Exception");
            e.printErrorDescription();
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }
}
```

WSDD 的文件描述:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="ExceptionCreater" provider="java:RPC">
        <parameter name="className"
value="webservice.axis.wsddexception.ExceptionCreater"/>
```

```
    <parameter name="allowedMethods" value="*"/>
    <parameter name="scope" value="Session"/>

    <operation name="getException"
               qname="operNS:getException"
               xmlns:OperNS="getException">
        <fault name="ServerExceptionFault"
            qname="fut:fault"
            xmlns:fut="ServerExceptionFault"
            type="tns:ServerException"
            xmlns:tns="ServerException"/>
    </operation>

    <typeMapping qname="myNs:ServerException"
        xmlns:myNs="urn:CustomerFault"
        type="java:webservice.axis.wsddexception.ServerException"

    serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

    deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactor
y"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </service>
</deployment>
```

首先不同的是多了个命名空间也就是namespace节点，等会测试代码中会看到用途。除了namespace之外还有operation这个节点和里面的fault子节点。先来介绍operation这个节点的属性。

- **name**：操作名称或者方法名称，这个值会和你server发布的相关方法名匹配，所以要和方法名相同。
- **qname**：针对这个operation的限定名。
- **xmlns**：针对这个qname 的命名空间也就是namespace。
- **Fault**：节点代表要这个方法要抛出的异常。
- **name**：随便起的名字。
- **type="tns:ServerException"**：:这里的一定要写出你要抛出的异常的类型。

运行结果：
客户端：

```
Will Catch Exception
Client Exception!
Catch Exception
Server Side Self Exception!
webservice.axis.wsddexception.ClientException
    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.getGeneralPurpos
e(BeanDeserializerFactory.java:89)
    at
org.apache.axis.encoding.ser.BaseDeserializerFactory.getDeserializerA
s(BaseDeserializerFactory.java:89)
    at
org.apache.axis.encoding.DeserializationContext.getDeserializer(Deser
ializationContext.java:464)
    at
org.apache.axis.encoding.DeserializationContext.getDeserializerForTyp
e(DeserializationContext.java:547)
    at
org.apache.axis.message.SOAPFaultDetailsBuilder.onStartChild(SOAPFaul
tDetailsBuilder.java:157)
    at
org.apache.axis.encoding.DeserializationContext.startElement(Deserial
izationContext.java:1035)
    at
com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.startEle
ment(Unknown Source)
    at
com.sun.org.apache.xerces.internal.impl.XMLNSDocumentScannerImpl.scan
StartElement(Unknown Source)
    at
com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImp
```

```
l$FragmentContentDriver.next(Unknown Source)
    at
com.sun.org.apache.xerces.internal.impl.XMLDocumentScannerImpl.next(U
nknown Source)
    at
com.sun.org.apache.xerces.internal.impl.XMLNSDocumentScannerImpl.next
(Unknown Source)
    at
com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImp
l.scanDocument(Unknown Source)
    at
com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(U
nknown Source)
    at
com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(U
nknown Source)
    at
com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown
Source)
    at
com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.parse(Un
known Source)
    at
com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser.p
arse(Unknown Source)
    at javax.xml.parsers.SAXParser.parse(Unknown Source)
    at
org.apache.axis.encoding.DeserializationContext.parse(Deserialization
Context.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(MustUnders
tandChecker.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
webservice.axis.wsddexception.ClientExceptionReceiver.main(ClientExce
ptionReceiver.java:54)
```

服务端：
Server Exception!

2.2.7 传递文件

服务端
Java 代码：

```
package webservice.axis.wsddfiletranspot;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

import javax.activation.DataHandler;

/**
 * 接收文件的服务端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class FileReceiverServer {

    public String receive(DataHandler handler, String fileName) {
        File file = new File(fileName);

        if (handler == null || fileName == null || fileName.equals(""))
 {
            return "failure";
        }

        InputStream input = null;
        FileOutputStream fos = null;

        try {
            input = handler.getInputStream();
            fos = new FileOutputStream(file);

            byte[] buffer = new byte[1024];
```

```
            while(input.read(buffer) != -1) {
                fos.write(buffer);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (input != null) {
                try {
                    input.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return "Success file saved on server. At: " +
file.getAbsolutePath();
    }
}
```

在这个服务端代码里我们主要介绍一个类就是 DataHandler 其实这个类是一个专门的传送器，通过他可以把文件进行序列化。这个方法从 DataHandler 得到一个输入流，从这个流里读出数据然后写到一个新文件里。这些都是一些基本的 Java I/O 操作。

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="FileReceiverServer" provider="java:RPC">
        <parameter name="className"
value="webservice.axis.wsddfiletranspot.FileReceiverServer"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="scope" value="session"/>

    <operation name="receive"
              qname="operNS:receive"
              xmlns:OperNS="receive"
              returnType="rtns:string"
```

```
                xmlns:rtns="http://www.w3.org/2001/XMLSchema">
        <parameter name="handler" type="tns:string"
            xmlns:tns="http://www.w3.org/2001/XMLSchema" />
        <parameter name="fileName" type="myns:DataHandler"
            xmlns:tns="http://www.w3.org/2001/XMLSchema" />
    </operation>

    <typeMapping qname="myNs:DataHandler"
        xmlns:myNs="DataHandler"
        languageSpecificType="java:javax.activation.DataHandler"

    serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializer
Factory"

    deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserial
izerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </service>
</deployment>
```

在这个地方有必要说一下就是大家在以前看见的地方都是用 beanMapping 来配置的，其实用 typeMapping 也是一样的，这里就不多说了。我主要说一下就是关于 org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory 和 org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory

其实这两个类也是很简单的，因为我们要序列化不同的对象所以我们要用到不同的工厂类，同样当你要序列化文件的时候也要用到文件的工厂类。

最后来让我们看看客户端的代码实现：

Java 代码：

```
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 javax.xml.rpc.encoding.XMLType;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory;
import org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory;
```

```
/**
 * 传输文件的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class FileSenderClient {

    public static void main(String[] args) {

        try {
            String fileName = "readme.txt";
            DataHandler dh = new DataHandler(new
FileDataSource(FileSenderClient.class.getResource(fileName).getPath())
);
            String url =
"http://127.0.0.1:8080/myaxis/services/FileReceiverServer";

            Service serv = new Service();
            Call call = (Call) serv.createCall();
            call.setTargetEndpointAddress(url);
            call.setOperationName(new QName(url, "receive"));
            QName qn = new QName("DataHandler", "myNs:DataHandler");
            call.registerTypeMapping(DataHandler.class, qn,
JAFDataHandlerSerializerFactory.class,
JAFDataHandlerDeserializerFactory.class);
            call.addParameter("s1", qn, ParameterMode.IN);
            call.addParameter("s2", XMLType.XSD_STRING,
ParameterMode.IN);
            call.setReturnClass(String.class);
            String returnStr = (String)call.invoke(new Object[]{dh,
"server.txt"});

            System.out.println("Server response: " + returnStr);
        } catch (ServiceException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

然后进行测试。

输出结果：
客户端：

```
Server response: Success file saved on server. At: D:\Tomcat\bin\server.txt
```

上面的列子都是经过测试而且完全跑通的，我们希望大家能够自己动手完成这些列子提高自己的水平。在下一章我们将会简单介绍一下关于 AXIS 的调试工具和相关的命令使用。

2.3.AXIS 的常用的命令和调试工具的使用

2.3.1 AXIS 的常用命令：

部署你的 WSDD 的命令是到 AXIS 的 web-inf 下然后运行 java org.apache.axis.client.AdminClient 你的部署文件。例如：

java org.apache.axis.client.AdminClient e:\deploy.wsdd

上面的部署只是简单的部署，其实关于 AdminClient 有很多参数命令，感兴趣的朋友可以自己看看只要在 AdminClient -help 就可以看到相关的参数命令。在这里我们主要讲解 2 个重要的和比较常用的，-l 和-p 一般有的时候我们不是使用 8080 这个端口，但是当你部署的时候你没有指定端口这样 AXIS 会默认使用 8080 所以就会导致部署失败，遇见这种情况就要使用参数-p 来解决。

例如：你的 TOMCAT 使用的是 8089 端口，当你部署的时候要如下代码：
java org.apache.axis.client.AdminClient e:\deploy.wsdd -p8089 注意：-p 后面直接加端口。

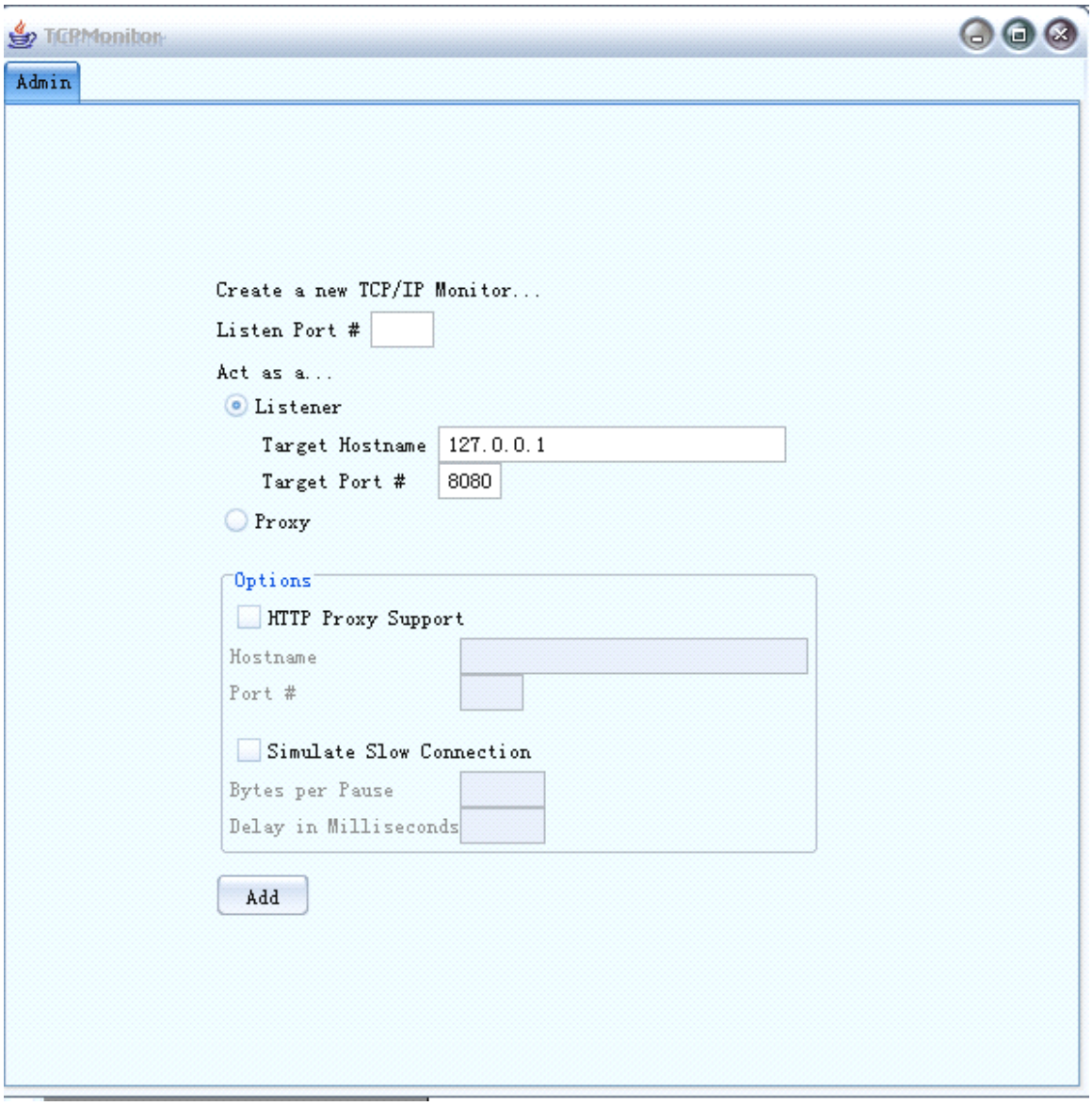
还有一种就是关于-l 的使用，由于 AXIS 默认部署的时候使用的虚拟路径是 axis，但是有的人不喜欢这个虚拟路径往往在 TOMCAT 里面就修改 server.xml 文件，创建自己的服务这个时候当你部署的时候你就需要使用-l 命令了，例如你的虚拟路径是 myaxis 那么你在部署的时候就要如下代码：

```
java org.apache.axis.client.AdminClient e:\deploy.wsdd
-lhttp://127.0.0.1:8080/myaxis/services/AdminService
```

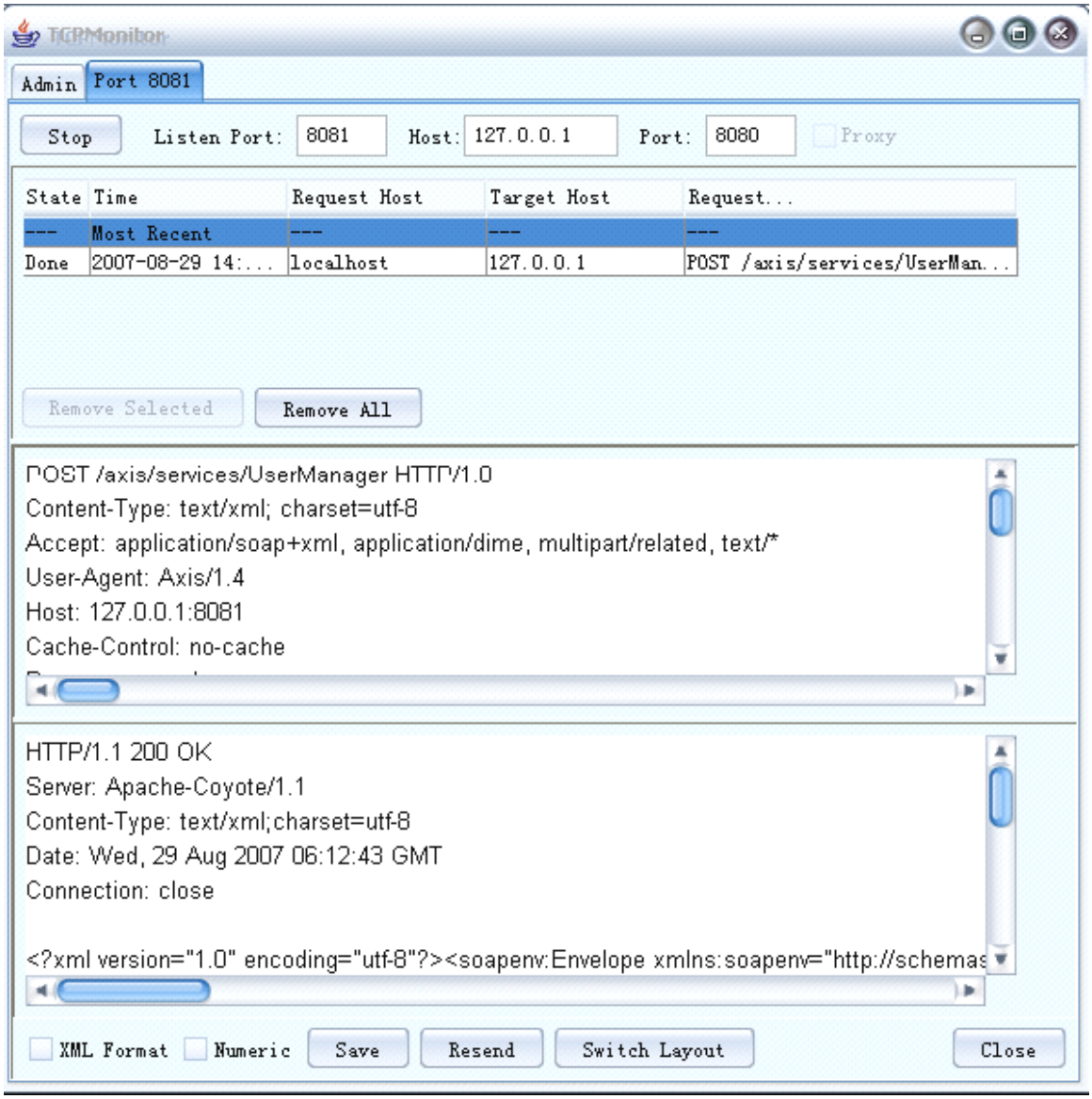
2.3.2 调试工具的使用

关于调试工具的使用在这里我简单的说明一下，代码如下：

运行java org.apache.axis.utils.tcpmon就会出现调试工具如图：



这样你就可以填写本地的地址和端口，还有你要监听的端口（提示：这里说下就是关于监听的用法，在监听了写上你要啊监控的端口号例如：8081 那么你在提供给客户端的URL的时候就要写上这个端口号，而不是服务端的端口号。例如：127.0.0.1：8081 因为这个是你的请求发送到这个工具上然后这个工具又发送到你真正的服务端上。）
然后点Add这样你就可以看到一些传递的详细信息了，如图：



2.4.Axis 通过 WSDL 生成服务端代码

一般情况下我们使用 Web 服务都是通过服务端提供 WSDL 信息来生成客户端的调用框架，但也不尽然，比如说通讯双方先定义好协议，也就是说最新确定下来的是 WSDL，就比如说移动的飞信平台，我们从移动获取到的 wsdl 的文件，现在必须依照 wsdl 文件来生成服务端的实现框架，也就是说我们是作为 web 服务的服务端来给飞信平台调用。做法如下：

java -Djava.ext.dirs={axis-lib} org.apache.axis.wsdl.WSDL2Java --server-side dsmp.wsdl

通过上面的命令帮你生成了服务端代码：

然后在代码里以 BindingImpl 结尾是 Web 服务的方法实现，你剩下的所有代码都可以在里面进行添加和修改。

2.5 Axis1.4 读取头信息

在 Axis 里读取头的信息是通过 Handler 来读取的本身它提供很多这样类似拦截的类有日志, JWS，SOAPMonitor，JAXRPC 等在这里我们只介绍 JAXRPCHandler 使用。
假设你的头信息如下：

```
<soap:Header>
    <msgName>SubscribeService</msgName>
    <transactionID>090204095129</transactionID>
    <version>1.0.0</version>
    <sendAddress>
        <provType>000</provType>
        <platType>210001</platType>
    </sendAddress>
    <destAddress>
        <provType>000</provType>
        <platType>100000</platType>
    </destAddress>
    <originalAddress>
        <provType>000</provType>
        <platType>210001</platType>
    </originalAddress>
    <timeStamp>20090204095129</timeStamp>
</soap:Header>
```

以上是定义好的头信息 假设客户端已经把这个头信息已经写入了，现在我们要用 AXIS 来读取这个头信息，来看代码实现：

```
/*
 * 文件名： TestSOAPMonitorService.java
 *
 * 创建日期： 2009-2-13
 *
 * Copyright(C) 2009, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package com.chinamobile.www.vgop.serviceorder.v1_0;

import java.util.Iterator;
import java.util.List;

import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;

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;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2009-2-13
 */
public class TestSOAPMonitorService extends JAXRPCHandler {

    private static final long serialVersionUID = 4754889674831614631L;

    @SuppressWarnings("unchecked")
    public void invoke(MessageContext msgContext) {
        try {
            // 获取头的对象
            SOAPHeader header =
msgContext.getCurrentMessage().getSOAPHeader();

            // 获取头的子节点
            for (Iterator it = header.getChildElements(); it.hasNext();) {
                SOAPHeaderElement element = (SOAPHeaderElement) it.next();

                //获取每个节点的子节点
                List list = element.getChildren();
                for (int i = 0; i < list.size(); i++) {
                    NodeImpl elementNode = (NodeImpl) list.get(i);
                    //如果没有子节点 那就直接获取值
                    if (!elementNode.hasChildNodes()) {
                        Text text = (Text) elementNode;
                        System.out.println("节点名字: " + element.getName()
                                + " 节点值: " + text.getValue());
                    }
                    //如果有子节点就获取子节点的信息
                    else {
                        MessageElement messageElement = (MessageElement)
elementNode;
```

```
                        List childList = messageElement.getChildren();
                        for (int j = 0; j < childList.size(); j++) {
                            NodeImpl childElementNode = (NodeImpl) childList
                                    .get(j);
                            if (!childElementNode.hasChildNodes()) {
                                Text childText = (Text) childElementNode;
                                System.out.println("父节点名字：" +
element.getName()
                                        + " 节点名字：" +
messageElement.getName()
                                        + " 节点值：" + childText.getValue());
                            }
                        }
                    }
                }
            }
        }
        catch (SOAPException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

以上是代码实现，还需要在你的部署文件里进行相应的配置如下：

```
      <requestFlow>
          <handler name="Handler1"
          type=" TestSOAPMonitorService.java ">
          </handler>
       </requestFlow>
```

这个配置要放到你的 service 这个节点里面，由于部分代码涉及到公司机密所以在这里我就不给贴出来的这 2 个地方都写好了 就可以部署你的 deploy.wsdd 了。然后运行你的客户端这个是否服务端就会打印一下结果：

```
节点名字：msgName 节点值：SubscribeService
节点名字：transactionID 节点值：090204095129
节点名字：version 节点值：1.0.0
父节点名字：sendAddress 节点名字：provType 节点值：000
父节点名字：sendAddress 节点名字：platType 节点值：210001
父节点名字：destAddress 节点名字：provType 节点值：000
父节点名字：destAddress 节点名字：platType 节点值：100000
父节点名字：originalAddress 节点名字：provType 节点值：000
```

父节点名字：originalAddress 节点名字：platType 节点值：210001
节点名字：timeStamp 节点值：20090204095129

2.6 .NET 访问 AXIS 的出现 Client.NoSOAPAction 的解决方案

这个问题是由于在.NET 访问的时候缺少了 SOAPAction，但是由于 AXIS 里又对 SOAPAction 做了校验所有会出现上面的问题，解决办法就是自己继承 AxisServlet 这个类然后进行重写里面的 getSoapAction 方法。

```
    /**
     * Extract the SOAPAction header. if SOAPAction is null then
we'll we be
     * forced to scan the body for it. if SOAPAction is "" then
use the URL
     *
     * @param req
     *            incoming request
     * @return the action
     * @throws AxisFault
     */
    private String getSoapAction(HttpServletRequest req) throws
AxisFault {
        return req.getContextPath();
    }
```

这样就可以了。然后在配置XML的时候使用你自己实现的类。例如我们这里是MyAxisServlet在写应用的时候就要写

```
<servlet>
        <servlet-name>AxisServlet</servlet-name>
        <display-name>Apache-Axis Servlet</display-name>
        <servlet-class>
            com.chinamobile.MyAxisServlet
        </servlet-class>
</servlet>
```

2.7 AXIS 服务端返回 SOAP Header 给客户端

一般的时候都是客户端想服务端传送头信息，然后服务端在处理请求后在把头返回给客户端。我在项目里就涉及到了这方面的应用，我记录下来和大家分享一下。
假设你已经写好了客户端并且头的信息是：

```
<soapenv:Header>
        <msgName>syncOrderRelationReq</msgName>
        <transactionID>13637</transactionID>
```

```
        <version>1.0.0</version>
        <sendAddress>
            <provType>000</provType>
            <platType>100000</platType>
        </sendAddress>
        <destAddress>
            <provType>000</provType>
            <platType>210001</platType>
        </destAddress>
        <originalAddress>
            <provType>000</provType>
            <platType>106000</platType>
        </originalAddress>
        <timeStamp>20090217203710</timeStamp>
 </soapenv:Header>
```

现在我们用 AXIS 来做响应，然后在返回给客户端的时候也是带有这个头信息的。
请看下面的代码：

```
/*
 * 文件名： TestSOAPMonitorService.java
 *
 * 创建日期： 2009-2-13
 *
 * Copyright(C) 2009, by xiaozhi.
 *
 * 原始作者: <a
href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package com.chinamobile.www.vgop.serviceorder.v1_0;

import java.util.Iterator;
import org.apache.axis.message.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import org.apache.axis.Message;
import org.apache.axis.MessageContext;
import org.apache.axis.handlers.BasicHandler;
import org.apache.axis.message.SOAPBody;
import org.apache.axis.message.SOAPHeaderElement;

/**
 *
 *
 * @author <a
href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
```

```
 *
 * @version $Revision$
 *
 * @since 2009-2-13
 */
public class TestSOAPMonitorService extends BasicHandler {

    private static final long serialVersionUID =
4754889674831614631L;

    private Message imsg, omsg;

    private SOAPEnvelope ienvelope, oenvelope;

    private SOAPHeader iheader, oheader;

    private SOAPBody ibody, obody;

    public void invoke(MessageContext msgContext) {
        try {

            Message message = msgContext.getCurrentMessage();

            SOAPEnvelope env = message.getSOAPEnvelope();

            //读取请求信息和获取请求头的对象
            imsg = msgContext.getRequestMessage();
            SOAPEnvelope ienvelope = imsg.getSOAPEnvelope();
            iheader = (SOAPHeader) ienvelope.getHeader();

            //响应信息和响应的头对象
            omsg = msgContext.getResponseMessage();
            SOAPEnvelope oenvelope = omsg.getSOAPEnvelope();
            oheader = (SOAPHeader) oenvelope.getHeader();

            for(Iterator iter = iheader.examineAllHeaderElements()
 ;iter.hasNext();){
                SOAPHeaderElement hel = (SOAPHeaderElement)
iter.next();
                //将获取请求的头方法哦响应信息的头里
                oheader.addChildElement(hel) ;
                String headerName = hel.getNodeName();
            }
```

```
            msgContext.setResponseMessage(omsg);


            System.out.println("header===" + env.getHeader());

        }
        catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

通过以上代码相信不用做过多介绍大家就可以理解了 ，但是代码写完了还要在部署描述符里进行响应的配置，

```
        <responseFlow>
          <handler name="Handler1"

type="java:com.chinamobile.www.vgop.serviceorder.v1_0.TestSOAPMonitorService">
          </handler>
       </responseFlow>
```

这样打开你的 TCPMON 就可以监控到服务端返回给客户端的信息里也有 header 消息了。

3. XFie 篇

3.1 XFire 的简介

XFire 是下一代的 java SOAP 框架。XFire 提供了非常方便的 API，使用这些 API 可以开发面向服务(SOA)的程序。它支持各种标准，性能优良（基于低内存的 STAX 模型）。

- 支持多个重要的 Web Service 标准，包括 SOAP、WSDL、WS-I Basic Profile、WSAddressing、WS-Security 等。
- 高性能的 SOAP 栈。
- 可选的绑定(binding)方式，如 POJO、XMLBeansJAXB1.1、JAXB2.0、Castor 和 JiBX 等。
- 支持 JSR181 API。
- 多种传输方式，如 HTTP、JMS、XMPP、In－JVM 等。
- 灵活的接口。
- 支持多个容器，如 Spring、Pico、Plexus、Loom。
- 支持 JBI，参看 servicemix 项目(http://servicemix.org)。
- 客户端和服务器代码生成。

在开发之前对用到的软件版本说明一下，因为不同的版本有些区别。我们这里所用的是

目前最新的版本。xfire-1.2.6。在这里说明一下：由于 JDK 版本的不同加载包的时候也会有所不同，我们是基于 1.5 以上做的所以加载包如下：activation-1.1.jar，commons-codec-1.3.jar，commons-httpclient-3.0.jar，commons-logging-1.0.4.jar，jaxb-api-2.0.jar，jaxb-impl-2.0.1.jar，jaxen-1.1-beta-9.jar，jdom-1.0.jar，jsr173_api-1.0.jar，mail-1.4.jar，spring.jar（这里的是 2.0 以上的包），stax-api-1.0.1.jar，wsdl4j-1.6.1.jar，wss4j-1.5.1.jar，wstx-asl-3.2.6.jar，xbean-2.2.0.jar，xbean-spring-2.8.jar，xfire-all-1.2.6.jar，xfire-jsr181-api-1.0-M1.jar，XmlSchema-1.1.jar，xmlsec-1.3.0.jar。

在你的 Web.xml 里的配置如下：

```
  <servlet>
    <servlet-name>XFireServlet</servlet-name>
    <servlet-
class>org.codehaus.xfire.transport.http.XFireConfigurableServlet</ser
vlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>

    <servlet-mapping>
        <servlet-name>XFireServlet</servlet-name>
        <url-pattern>/servlet/XFireServlet/*</url-pattern>
    </servlet-mapping>

  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
```

做完以上工作我们就可以开始我的 XFire 之旅了。

3.2 简单的应用

废话少说直接看代码:

WebService 的接口：

```
/*
 * 文件名： IHelloWorld.java
 *
 * 创建日期： 2008-7-14
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
```

```
 */
package webservice.xfire.simple;

/**
 * 本地实现一个Remote接口，其中包含远程WS的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程WS对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-14
 */
public interface IHelloWorld extends java.rmi.Remote {

    public String hello(String name);

    public Float add(Float a, float b);
}
```

服务端：

Java 代码：

```
/*
 * 文件名： HelloWorld.java
 *
 * 创建日期： 2008-7-23
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.simple;

/**
 * 简单的Service实现
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-23
```

```
 */
public class HelloWorld {

    private int requestCount = 0;

    public String hello(String name) {
        requestCount++;
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + name);

        return "Hello " + name;
    }

    public Float add(Float a, float b) {
        requestCount++;
        String result = "a=" + a + ", b=" + b;
        System.out.println("requestCount=" + requestCount);
        System.out.println("Received: " + result);

        return a + b;
    }
}
```

客户端调用：

Java 代码：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-23
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.simple;

import java.net.MalformedURLException;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

/**
```

```
 * XFire客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-23
 */
public class Client {

    public static void main(String[] args) {
        String url =
"http://127.0.0.1:8080/myxfire/services/HelloWorldSimple";

        Service serviceModel = new
ObjectServiceFactory().create(IHelloWorld.class);
        try {
            IHelloWorld service = (IHelloWorld) new
XFireProxyFactory().create(serviceModel, url);

            String result = service.hello(System.getProperty("user.name"));
            System.out.println("result = " + result);

            Float returnValue = service.add(new Float(3.2), new Float(2.8));
            System.out.println("returnValue = " + returnValue);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }

    }

}
```

以上是对应的代码实现，现在开始编写我们的部署文件：首先在你的 src 下创建 META-INF/xfire 文件在下面创建 services.xml。现在介绍一下里面的配置方式，

```
<!-- START SNIPPET: services -->
<beans>
    <service xmlns="http://xfire.codehaus.org/config/1.0">
        <name>HelloWorldSimple</name>

<namespace>http://simple.xfire.webservice/HelloWorld</namespace>

<serviceClass>webservice.xfire.simple.HelloWorld</serviceClass>
```

```
<implementationClass>webservice.xfire.simple.HelloWorld</implementationClass>
        <scope>request</scope>
    </service>
</beans>
<!-- END SNIPPET: services -->
```

在这里说明一下上面的配置文件的意思：首先强调一点就是关于 xmlns=http://xfire.codehaus.org/config/1.0 的写法，由于上面的是和 Spring2.0 做的结合这样就要把上面的这句话加到你的 service 里面。如果是和 1.X 版本结合那么就要把这句话加到你的 beans 后面。

service：包含你要生命的服务。
name：表示你的服务名字。
namesace：表示你的命名空间。
serviceClass： 提供服务的类一般是接口，也可以是具体的实现类。
implementationClass：对应接口的实现。
scope：存活的范围。

以上工作都做好了现在可以进行你的测试了，发布你的服务然后运行客户端程序如果输出：
result = Hello Administrator
returnValue = 6.0
表示你已经成功了。

3.3 传递复杂对象

3.3.1 List、Map、数组和自定义对象

3.3.1.1 在对象里包含的 List，Map，数组

自定义的 JAVA 对象：

```
/*
 * 文件名： Address.java
 *
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
```

```
 */
package webservice.xfire.selfobj.server.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 服务端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;

    /** 邮编 */
    private String postalCode;

    private String[] array;

    private List<Integer> list;

    private Map<Integer, InnerClass> map;
```

```
private boolean isExist;

private InnerClass innC;

public static class InnerClass implements Serializable {

 private static final long serialVersionUID = -2330738090948448510L;

 private String innerName = "static InnerClass";

 public InnerClass() {};

 public InnerClass(String innerName) {
         super();
         this.innerName = innerName;
     }

     public String getInnerName() {
         return innerName;
     }

     public void setInnerName(String innerName) {
         this.innerName = innerName;
     }

}

public Address() {
 list = new ArrayList<Integer>();
 list.add(1);
 list.add(2);
 list.add(3);

 map = new HashMap<Integer, InnerClass>();
 map.put(1, new InnerClass("A"));
 map.put(2, new InnerClass("B"));
 map.put(3, new InnerClass("C"));

 innC = new InnerClass();
 innC.setInnerName("服务端: Address.InnerClass");
}

@Override
```

```
public String toString() {
    String returnStr = super.toString()
        + "id号: " + getIdentifier()
        + " address: " + getAddress()
        + " city: " + getCity()
        + " country: " + getCountry()
        + " postalCode: " + getPostalCode()
        + " province: " + getProvince()
        + " array: " + getArray()[0]
        + " list: " + getList()
        + " map: " + getMap()
        + " isExist: " + isExist()
        + " innerClass.name: " + getInnC().getInnerName();
    return returnStr;
}

public InnerClass getInnC() {
    return innC;
}

public void setInnC(InnerClass innC) {
    this.innC = innC;
}

public Integer getIdentifier() {
    return identifier;
}

public void setIdentifier(Integer identifier) {
    this.identifier = identifier;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}
```

```
public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}

public void setPostalCode(String postalCode) {
    this.postalCode = postalCode;
}

public String[] getArray() {
    return array;
}

public void setArray(String[] array) {
    this.array = array;
}

public boolean isExist() {
    return isExist;
}

public void setExist(boolean isExist) {
    this.isExist = isExist;
}
```

```
    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

    public Map<Integer, InnerClass> getMap() {
        return map;
    }

    public void setMap(Map<Integer, InnerClass> map) {
        this.map = map;
    }
}
```

从上面的对象里可以看到我们自定义的对象里包括了：List，Map 和内部类。

服务端代码：

```
/*
 * 文件名： AddressManager.java
 *
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfobj.server;

import java.util.ArrayList;
import java.util.List;

import webservice.xfire.selfobj.server.model.Address;


/**
 * 提供复杂对象的WebService业务
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
```

```
 *
 * @since 2008-7-11
 */
public class AddressManager {

    private int requestCount = 0;

    public List<Address> getAddressList() {
        requestCount++;
        System.out.println("requestCount=" + requestCount);

        List<Address> returnList = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnList.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        returnList.add(address);
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});

        return returnList;
    }

    public List<Address> setAddressList(List<Address> list) {
        requestCount++;
        System.out.println("requestCount=" + requestCount);
        return list;
    }
```

```
}
```

客户端接口：
Java 代码：

```
/*
 * 文件名： IClientAddressManager.java
 *
 * 创建日期： 2008-7-17
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfobj.client;

import java.util.List;

import webservice.xfire.selfobj.server.model.Address;

/**
 * 本地实现一个Remote接口，其中包含远程WS的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程WS对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-17
 */
public interface IClientAddressManager {

    public List<Address> getAddressList();

    public List<Address> setAddressList(List<Address> list);
}
```

客户端：
Java 代码：

```
/*
 * 文件名： Client.java
```

```
 *
 * 创建日期： 2008-7-23
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfobj.client;

import java.net.MalformedURLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

import webservice.xfire.selfobj.server.model.Address;

/**
 * XFire客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-23
 */
public class Client {

    private static String url =
"http://127.0.0.1:8089/myxfire/services/AddressManager";

    public static void getServerList() {
        Service serviceModel = new
ObjectServiceFactory().create(IClientAddressManager.class);
        try {
            IClientAddressManager service = (IClientAddressManager) new
XFireProxyFactory().create(serviceModel, url);

            List<Address> list = (ArrayList<Address>)
service.getAddressList();
```

```
            System.out.println("List size: " + list.size());

            for (Iterator<Address> iter = list.iterator(); iter.hasNext();) {

                Address address = iter.next();
                System.out.println(address);
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    public static void setServerList() {
        Service serviceModel = new
ObjectServiceFactory().create(IClientAddressManager.class);
        try {
            IClientAddressManager service = (IClientAddressManager) new
XFireProxyFactory().create(serviceModel, url);

            List<Address> list = new ArrayList<Address>();

            Address address = new Address();
            address.setIdentifier(1);
            address.setAddress("Haidian");
            address.setCity("BeiJing");
            address.setCountry("China");
            address.setPostalCode("100081");
            address.setProvince("Beijing");
            address.setExist(false);
            address.setArray(new String[]{"1","2","3"});
            list.add(address);

            address = new Address();
            address.setIdentifier(2);
            address.setAddress("Chaoyang");
            address.setCity("BeiJing");
            address.setCountry("China");
            address.setPostalCode("100081");
            address.setProvince("Beijing");
            address.setExist(true);
            address.setArray(new String[]{"A","B","C"});
            list.add(address);
```

```
            List<Address> returnList = service.setAddressList(list);
            System.out.println("List size: " + returnList.size());

            for (Iterator<Address> iter = returnList.iterator();
iter.hasNext();) {

                address = iter.next();
                System.out.println(address);
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        getServerList();

        setServerList();
    }
}
```

以上都做完了就可以运行你的客户端代码运行结果是：

```
List size: 2
webservice.xfire.selfobj.server.model.Address@b61fd1id号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map: {} isExist: false innerClass.name: 服务
端: Address.InnerClass
webservice.xfire.selfobj.server.model.Address@e2dae9id号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map: {} isExist: true
innerClass.name: 服务端: Address.InnerClass
List size: 2
webservice.xfire.selfobj.server.model.Address@f99ff5id号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map: {} isExist: false innerClass.name: 服务
端: Address.InnerClass
webservice.xfire.selfobj.server.model.Address@74c3aaid号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map: {} isExist: true
innerClass.name: 服务端: Address.InnerClass
```

在这里我们说下要注意的地方，就是关于这个自定义对象的问题，自定义对象这个包名客户端与服务端一定要保证包名和类名一样也可以使用同一个要不会找不到这块经过实验已经获得证实，如果读者有更好的建议请与我们联系谢谢。

3.3.1.2 传递 Map

刚才的 Map 传递是在你定义好的对象里面传递的，现在要传递的是直接服务端的方法返回一个 Map。来看代码实现：

接口代码：
Java 代码：

```
/*
 * 文件名： IAddressManagerMap.java
 *
 * 创建日期： 2008-7-24
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfmap;

import java.util.Map;

import webservice.xfire.selfobj.server.model.Address;

/**
 * 实现MAP的传递，本地与服务器必须用同一个接口
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
public interface IAddressManagerMap {
    public Map<Integer, Address> getAddressMap();

    public Map<Integer, Address> setAddressMap(Map<Integer, Address> map);
}
```

对应的接口实现服务端：
Java 代码：

```
/*
```

```
 * 文件名： AddressManagerMap.java
 *
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfmap.server;

import java.util.HashMap;
import java.util.Map;

import webservice.xfire.selfmap.IAddressManagerMap;
import webservice.xfire.selfobj.server.model.Address;

/**
 * 传递Map的WebService业务
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
public class AddressManagerMap implements IAddressManagerMap{

    private int requestCount = 0;

    public Map<Integer, Address> getAddressMap() {
        requestCount++;
        System.out.println("requestCount=" + requestCount);

        Map<Integer, Address> returnMap = new HashMap<Integer, Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
```

```
        address.setArray(new String[]{"1","2","3"});

        returnMap.put(address.getIdentifier(), address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        returnMap.put(address.getIdentifier(), address);

        return returnMap;
    }

    public Map<Integer, Address> setAddressMap(Map<Integer, Address> map) {
        requestCount++;
        System.out.println("requestCount=" + requestCount);
        return map;
    }
}
```

除了以上的做法还有做一个事情就是绑定 aegis，这个在这说下 aegis 的绑定有个规则就是必须和你服务端的接口叫一样的名字并且放在同一目录下，例如我们的是 IAddressManagerMap 接口那么我们的 aegis 就应该叫做 IAddressManagerMap. aegis.xml 并且要和 IAddressManagerMap 放在同一目录下。命名规范就是：接口名字. aegis.xml
现在给出 IAddressManagerMap. aegis.xml 的代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings>
    <mapping>
        <method name="getAddressMap">
            <return-type
componentType="webservice.xfire.selfobj.server.model.Address" />
        </method>
        <method name="setAddressMap">
            <parameter index="0"
componentType="webservice.xfire.selfobj.server.model.Address" />
            <return-type
componentType="webservice.xfire.selfobj.server.model.Address" />
        </method>
```

```
    </mapping>
</mappings>
```

这里简单介绍一下：
method：表示你要执行的方法，name 是对应的方法名字。
return-type：表示你要返回的类型里面有个 componentType 这个就是表示你返回的类型。
parameter：表示方法所要的参数 index 是参数索引表示第几个参数从 0 开始计算。
componentType 也是表示你参数传递的类型。

还要说一点就是大家认为我返回的是 Map 类型你为什么不写而写 Map 里面的类型呢，其实在你返回数据和设置数据的时候他检查的是你的 Map 里的类型是什么类型你只要告诉他里面的真实类型就行了。

来看下面的客户端：
Java 代码：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-23
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.selfmap.client;

import java.net.MalformedURLException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

import webservice.xfire.selfmap.IAddressManagerMap;
import webservice.xfire.selfobj.server.model.Address;

/**
 * XFire客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
```

```
 *
 * @version $Revision$
 *
 * @since 2008-7-23
 */
public class Client {

    private static final String url =
"http://127.0.0.1:8089/myxfire/services/AddressManagerMap";

    private static final String namespace =
"http://selfmap.xfire.webservice/IAddressManagerMap";

    public static void getAddressMap() {
        Service serviceModel = new
ObjectServiceFactory().create(IAddressManagerMap.class, null, namespace,
null);
        try {
            IAddressManagerMap service = (IAddressManagerMap) new
XFireProxyFactory().create(serviceModel, url);

            Map<Integer, Address> map = (Map<Integer, Address>)
service.getAddressMap();

            System.out.println("Map size: " + map.size());

            for (Iterator<Integer> iter = map.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                Address address = map.get(key);
                System.out.println(address);
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    public static void setAddressMap() {
        Service serviceModel = new
ObjectServiceFactory().create(IAddressManagerMap.class, null, namespace,
null);
        try {
            IAddressManagerMap service = (IAddressManagerMap) new
XFireProxyFactory().create(serviceModel, url);
```

```
            Map<Integer, Address> map = new HashMap<Integer, Address>();

            Address address = new Address();
            address.setIdentifier(1);
            address.setAddress("Haidian");
            address.setCity("BeiJing");
            address.setCountry("China");
            address.setPostalCode("100081");
            address.setProvince("Beijing");
            address.setExist(false);
            address.setArray(new String[]{"1","2","3"});
            map.put(address.getIdentifier(), address);

            address = new Address();
            address.setIdentifier(2);
            address.setAddress("Chaoyang");
            address.setCity("BeiJing");
            address.setCountry("China");
            address.setPostalCode("100081");
            address.setProvince("Beijing");
            address.setExist(true);
            address.setArray(new String[]{"A","B","C"});
            map.put(address.getIdentifier(), address);

            Map<Integer, Address> mapReturn = (Map<Integer, Address>)
service.setAddressMap(map);

            System.out.println("Map size: " + mapReturn.size());

            for (Iterator<Integer> iter = mapReturn.keySet().iterator();
iter.hasNext();) {
                Integer key = iter.next();
                address = mapReturn.get(key);
                System.out.println(address);
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

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

```
            setAddressMap();
        }
}
```

这里大家可能又有些疑问了，观察仔细的朋友可能发现了一点问题，就是为什么上面的客户端实现的代码和前几个实现客户端的代码有些不相同呢，在这里我说明一下为什么，由于以前的实现我们没有准确的根据命名空间去查找，给出源码来看看：

```
    public org.codehaus.xfire.service.Service create(Class clazz)
    {
        return create(clazz, (Map)null);
    }

    public org.codehaus.xfire.service.Service create(Class clazz, Map
properties)
    {
        return create(clazz, (String)null, (String)null, properties);
    }
```

所以在你即时绑定了 aegis 对应的返回类型也不会被找到。运行结果：
客户端：

```
Map size: 2
webservice.xfire.selfobj.server.model.Address@1d80e6did号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map:
{2=webservice.xfire.selfobj.server.model.Address$InnerClass@19b5393,
1=webservice.xfire.selfobj.server.model.Address$InnerClass@8bdcd2,
3=webservice.xfire.selfobj.server.model.Address$InnerClass@4e79f1}
isExist: true innerClass.name: 服务端: Address.InnerClass
webservice.xfire.selfobj.server.model.Address@27e353id号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map:
{2=webservice.xfire.selfobj.server.model.Address$InnerClass@bd928a,
1=webservice.xfire.selfobj.server.model.Address$InnerClass@1dfc547,
3=webservice.xfire.selfobj.server.model.Address$InnerClass@10f6d3}
isExist: false innerClass.name: 服务端: Address.InnerClass
Map size: 2
webservice.xfire.selfobj.server.model.Address@a31e1bid号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map:
{2=webservice.xfire.selfobj.server.model.Address$InnerClass@10da5eb,
1=webservice.xfire.selfobj.server.model.Address$InnerClass@1081d2e,
3=webservice.xfire.selfobj.server.model.Address$InnerClass@1b3f829}
isExist: true innerClass.name: 服务端: Address.InnerClass
```

```
webservice.xfire.selfobj.server.model.Address@698403id号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map:
{2=webservice.xfire.selfobj.server.model.Address$InnerClass@15a0305,
1=webservice.xfire.selfobj.server.model.Address$InnerClass@7c4c51,
3=webservice.xfire.selfobj.server.model.Address$InnerClass@765a16}
isExist: false innerClass.name: 服务端: Address.InnerClass
```

如果不按照我上面的写那么你会得到一个 NULL 值，在网上我也看到很多人在问在这里我们已经给予解决，如果那块说的不对请您加我们的 QQ 来指点一下再次谢过。

3.3.2 异常处理

直接看代码：
异常描述类：
Java 代码：

```
/*
 * 文件名: HelloWordExceptionDetail.java
 *
 * 创建日期: 2008-7-24
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.exception;

import java.io.Serializable;

/**
 * 描述异常的数据类 一定要有默认构造函数
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
public class HelloWordExceptionDetail implements Serializable {

    private static final long serialVersionUID = 1312467612597288019L;
```

```
    private String message;

    public HelloWordExceptionDetail() {
    }

    public HelloWordExceptionDetail(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

这个类是异常描述类这里主要说一下就是他必须有构造函数，别的没有什么了。

异常类：

Java 代码:

```
/*
 * 文件名： HelloWordException.java
 *
 * 创建日期： 2008-7-24
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.exception;

import org.codehaus.xfire.fault.FaultInfoException;

/**
 * 异常类
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
```

```
public class HelloWordException extends FaultInfoException {

    private static final long serialVersionUID = 8373919408375406129L;

    private HelloWordExceptionDetail hwd;

    public HelloWordException(String message, HelloWordExceptionDetail hwd) {

        super(message);
        this.hwd = hwd;
    }

    public HelloWordExceptionDetail getFaultInfo() {
        return hwd;
    }

}
```

这个里主要继承了 FaultInfoException 类，它是处理异常基本类。

服务接口类：

Java 代码：

```
/*
 * 文件名： IHelloWorldService.java
 *
 * 创建日期： 2008-7-24
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.exception;

/**
 * 服务和客户端必须保证包名一样，提供对应的服务接口
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
public interface IHelloWorldService {
```

```
    public String getException(String message) throws HelloWordException;
}
```

服务对应接口的实现类：

```
/*
 * 文件名： HelloWorldServiceImpl.java
 *
 * 创建日期： 2008-7-24
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.exception.server;

import webservice.xfire.exception.HelloWordException;
import webservice.xfire.exception.HelloWordExceptionDetail;
import webservice.xfire.exception.IHelloWorldService;

/**
 * 对应的服务实现
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
public class HelloWorldServiceImpl implements IHelloWorldService {

    /*
     *
     * (non-Javadoc)
     *
     * @see
webservice.xfire.exception.IHelloWorldService#getException(java.lang.String)
     */
    public String getException(String message) throws HelloWordException {

        if (message.equals("")) {
            throw new HelloWordException("参数不能为空 ", new
HelloWordExceptionDetail("异常测试成功"));
        }
```

```
        System.out.println(message);

        return "没有异常";
    }

}
```

客户端：

Java 代码：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-24
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.exception.client;

import java.net.MalformedURLException;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

import webservice.xfire.exception.HelloWordException;
import webservice.xfire.exception.IHelloWorldService;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-24
 */
public class Client {

    private static final String url =
"http://127.0.0.1:8089/myxfire/services/HelloWordException";
```

```
    private static final String namespace =
"http://exception.xfire.webservice/HelloWordException";

    /**
     * @param args
     */
    public static void main(String[] args) {

        Service serviceModel = new
ObjectServiceFactory().create(IHelloWorldService.class, null, namespace,
null);

        try {
            IHelloWorldService client = (IHelloWorldService) new
XFireProxyFactory().create(serviceModel, url);

            String success = client.getException("成功") ;
            System.out.println(success) ;

            client.getException("") ;
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (HelloWordException e) {
            System.out.println(e.getFaultInfo().getMessage());
            //e.printStackTrace();
        }
    }
}
```

这个没有什么好说的了最重要的就是记住描述类要有构造函数，我个人认为写类要按照规范写这样可以避免无谓的错误。

3.3.3 Handler 处理

服务端接口：

Java 代码：

```
/*
 * 文件名： IHelloWorldService.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
```

```
 *
 */
package webservice.xfire.handlers;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public interface IHelloWorldService {

    public String getStr(String message) ;
}
```

对应的实现类：

Java 代码：

```
/*
 * 文件名： HelloWorldService.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.server;

import webservice.xfire.handlers.IHelloWorldService;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class HelloWorldService implements IHelloWorldService {
```

```
    /*
     *
     * (non-Javadoc)
     *
     * @see
webservice.xfire.handlers.IHelloWorldService#getStr(java.lang.String)
     */
    public String getStr(String message) {
        System.out.println("Receive message: " + message);
        return message;
    }

}
```

服务端前置的 Handle 类：

Java 代码：

```
/*
 * 文件名： HelloWordHandler.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.server;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 服务端Handler类
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class ServerHelloWordInHandler extends AbstractHandler {

    /*
```

```
     *
     * (non-Javadoc)
     *
     * @see
org.codehaus.xfire.handler.Handler#invoke(org.codehaus.xfire.MessageContext)
     */
    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用服务端之前做的一件事");

        ctx.setProperty("myparam", "ServerHelloWordInHandler传递的参数") ;

        System.out.println("ServerHelloWordInHandler Exit.");
    }

}
```

服务端后置 Handler 类：

Java 代码：

```
/*
 * 文件名： HelloWordHandler.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.server;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 服务端Handler类
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class ServerHelloWordOutHandler extends AbstractHandler {
```

```
    /*
     *
     * (non-Javadoc)
     *
     * @see
org.codehaus.xfire.handler.Handler#invoke(org.codehaus.xfire.MessageContext)
     */
    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用服务端之后做的一件事");

        System.out.println(ctx.getProperty("myparam"));

        System.out.println("ServerHelloWordOutHandler Exit.");
    }

}
```

客户端前置 Handler 类：

Java 代码：

```
/*
 * 文件名： AddHeaderHandler.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.client;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 写入版本号的Handler类
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class ClientHeaderOutHandler extends AbstractHandler {
```

```
    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("客户端之前做的事");

        ctx.setProperty("myparam", "ClientHeaderOutHandler传递的参数") ;

        System.out.println("ClientHeaderOutHandler Exit.");
    }

}
```

客户端后置 Handler 类：

Java 代码：

```
/*
 * 文件名： AddHeaderHandler.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.client;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 写入版本号的Handler类
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class ClientHeaderInHandler extends AbstractHandler {

    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("客户端之后做的事");

        System.out.println(ctx.getProperty("myparam"));
```

```
        System.out.println("ClientHeaderInHandler Exit.");
    }

}
```

客户端类：

Java 代码：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.handlers.client;

import java.net.MalformedURLException;

import org.codehaus.xfire.client.Client;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;
import webservice.xfire.handlers.IHelloWorldService;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class ClientHandlers {

    private static final String url =
"http://127.0.0.1:8089/myxfire/services/HelloWordHandler";

    private static final String namespace =
"http://handlers.xfire.webservice/HelloWordHandler";
```

```
    /**
     * @param args
     */
    public static void main(String[] args) {

        Service serviceModel = new
ObjectServiceFactory().create(IHelloWorldService.class, null, namespace,
null);

        try {
            IHelloWorldService clientService = (IHelloWorldService) new
XFireProxyFactory().create(serviceModel, url);

            Client client = Client.getInstance(clientService);

            client.addInHandler(new ClientHeaderInHandler());

            client.addOutHandler(new ClientHeaderOutHandler());

            System.out.println(clientService.getStr("测试成功"));
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

}
```

来看对应的部署文件 services.xml：

```
<beans>
    <service xmlns="http://xfire.codehaus.org/config/1.0">
        <name>HelloWordHandler</name>

    <namespace>http://handlers.xfire.webservice/HelloWordHandler</nam
espace>

    <serviceClass>webservice.xfire.handlers.IHelloWorldService</servi
ceClass>

    <implementationClass>webservice.xfire.handlers.server.HelloWorldS
ervice</implementationClass>
        <scope>application</scope>
        <inHandlers>
            <handler
handlerClass="webservice.xfire.handlers.server.ServerHelloWordInHandl
```

```
er" />
        </inHandlers>
        <outHandlers>
            <handler
handlerClass="webservice.xfire.handlers.server.ServerHelloWordOutHand
ler" />
        </outHandlers>
    </service>
</beans>
```

这里我们部署的是在服务端前置和后置的类如果你这么部署了他会在调用方法之前和之后做前置和后置类。

再看输出：

在客户端的输出：

```
客户端之前做的事
ClientHeaderOutHandler Exit.
客户端之后做的事
ClientHeaderOutHandler传递的参数
ClientHeaderInHandler Exit.
测试成功
```

在服务端的输出：

```
调用服务端之前做的一件事
ServerHelloWordInHandler Exit.
Receive message: 测试成功
调用服务端之后做的一件事
ServerHelloWordInHandler传递的参数
ServerHelloWordOutHandler Exit.
```

3.3.4 文件上传处理

服务接口：

Java 代码：

```
/*
 * 文件名： IFileTransport.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.filetransport;
```

```
import javax.activation.DataHandler;

/**
 * 传送文件的接口，本地与服务器必须用同一个接口
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public interface IFileTransport {

    public DataHandler getFile();

    public String uploadFile(DataHandler handler);
}
```

对应的接口实现类：

Java 代码：

```
/*
 * 文件名： FileTransportImpl.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.filetransport.server;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;

import webservice.xfire.filetransport.IFileTransport;
```

```
/**
 * 服务端传送文件的实现类，服务的提供者。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class FileTransportImpl implements IFileTransport {

    /*
     * (non-Javadoc)
     *
     * @see webservice.xfire.filetransport.IFileTransport#getFile()
     */
    public DataHandler getFile() {
        System.out.println("getFile is called");

        String fileName = "server.txt";
        DataHandler dh = new DataHandler(new FileDataSource(
                FileTransportImpl.class.getResource(fileName).getPath()));

        return dh;
    }

    /*
     * (non-Javadoc)
     *
     * @see
webservice.xfire.filetransport.IFileTransport#uploadFile(javax.activation.Da
taHandler)
     */
    public String uploadFile(DataHandler handler) {
        System.out.println("uploadFile is called");

        File file = new File("fromClient.txt");

        if (handler == null) {
            return "failure";
        }

        InputStreamReader input = null;
```

```
        OutputStreamWriter fos = null;
        BufferedReader br = null;

        try {
            input = new InputStreamReader(handler.getInputStream(), "UTF-8");
            fos = new OutputStreamWriter(new FileOutputStream(file), "UTF-8");

            br = new BufferedReader(input);
            String tmpStr = null;
            while ((tmpStr = br.readLine()) != null) {
                fos.write(tmpStr);
                fos.write("\r\n");
            }

        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (input != null) {
                try {
                    input.close();
                    br.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (fos != null) {
                try {
                    fos.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return "Success file saved on server. At: " + file.getAbsolutePath();
    }

}
```

客户端：

Java 代码：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-25
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.xfire.filetransport.client;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.lang.reflect.Proxy;
import java.net.MalformedURLException;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;

import org.codehaus.xfire.client.XFireProxy;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;
import org.codehaus.xfire.soap.SoapConstants;

import webservice.xfire.filetransport.IFileTransport;

/**
 * 测试文件传输的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-25
 */
public class Client {
```

```
    private static final String url =
"http://127.0.0.1:8089/myxfire/services/FileTransport";

    private static final String namespace =
"http://filetransport.xfire.webservice/FileTransport";

    public static void getFileFromServer() {
        InputStreamReader input = null;
        OutputStreamWriter fos = null;
        BufferedReader br = null;
        File file = null;

        Service serviceModel = new ObjectServiceFactory().create(
                IFileTransport.class, null, namespace, null);

        try {
            IFileTransport service = (IFileTransport) new XFireProxyFactory()
                    .create(serviceModel, url);

            org.codehaus.xfire.client.Client client = ((XFireProxy) Proxy
                    .getInvocationHandler(service)).getClient();
            // client.setProperty(HttpTransport.CHUNKING_ENABLED, "true");
            client.setProperty(SoapConstants.MTOM_ENABLED, "true");

            DataHandler handler = service.getFile();

            file = new File("fromServer.txt");

            if (handler == null) {
                System.out.println("failure");
            }

            input = new InputStreamReader(handler.getInputStream(), "UTF-8");
            fos = new OutputStreamWriter(new FileOutputStream(file), "UTF-8");

            br = new BufferedReader(input);
            String tmpStr = null;
            while ((tmpStr = br.readLine()) != null) {
                fos.write(tmpStr);
                fos.write("\r\n");
            }

        }
```

```
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (input != null) {
                try {
                    input.close();
                    br.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (fos != null) {
                try {
                    fos.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        System.out.println("Success file saved on clinet. At: "
                + file.getAbsolutePath());
    }

    public static void sendFileToServer() {
        String fileName = "client.txt";

        try {
            Service serviceModel = new ObjectServiceFactory().create(
                    IFileTransport.class, null, namespace, null);

            IFileTransport service = (IFileTransport) new XFireProxyFactory()
                    .create(serviceModel, url);

            org.codehaus.xfire.client.Client client = ((XFireProxy) Proxy
                    .getInvocationHandler(service)).getClient();
            // client.setProperty(HttpTransport.CHUNKING_ENABLED, "true");
```

```
            client.setProperty(SoapConstants.MTOM_ENABLED, "true");

            DataHandler handler = new DataHandler(new FileDataSource(
                    Client.class.getResource(fileName).getPath()));

            String returnString = service.uploadFile(handler);

            System.out.println(returnString);
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }

    }

    public static void main(String[] args) {
        getFileFromServer();

        sendFileToServer();
    }

}
```

在这里注意一点就是编码格式：如果你不对格式做处理上传后的文件你用 UE 打开或者 EditPlus 打开会看见增加了不知道是什么的乱码。

对应的服务部署 services.xml:

```
<beans>

    <service xmlns="http://xfire.codehaus.org/config/1.0">
        <name>FileTransport</name>

<namespace>http://filetransport.xfire.webservice/FileTransport</names
pace>

<serviceClass>webservice.xfire.filetransport.IFileTransport</serviceC
lass>

<implementationClass>webservice.xfire.filetransport.server.FileTransp
ortImpl</implementationClass>
        <scope>request</scope>
        <!-- 上传文件必须要配置这个property -->
        <properties>
            <property key="mtom-enabled">true</property>
        </properties>
```

```
    </service>
</beans>
```

3.4 XFire 与 Spring 集成

在和 Spring 集成的时候主要有需要配置 Web.xml, applicationContext.xml, xfire-servlet.xml 这 3 个文件，剩下的代码都是上面提供的一样。

Web.xml 配置代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-
value>classpath:applicationContext.xml,classpath:org/codehaus/xfire/spring/xfire.xml</param-value>
    </context-param>

    <!--Spring ApplicationContext 载入 -->
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <listener>
        <listener-
class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>XFireServlet</servlet-name>
        <servlet-
class>org.codehaus.xfire.spring.XFireSpringServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>XFireServlet</servlet-name>
```

```
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>

</web-app>
```

applicationContext.xml 配置代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
          http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
          http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
          http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd"
       default-autowire="byName" default-lazy-init="true">


    <import resource="classpath:org/codehaus/xfire/spring/xfire.xml"
/>

    <import resource="xfire-servlet.xml"/>

</beans>
```

xfire-servlet.xml 代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>

    <!-- 简单的调用 -->
    <bean id="HelloWorldSimple"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name">
        <value>HelloWorldSimple</value>
    </property>
    <property name="namespace">
        <value>http://simple.xfire.webservice/HelloWorld</value>
```

```
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.simple.HelloWorld</value>
    </property>
    <property name="serviceBean">
        <ref bean="helloWorldSimple"/>
    </property>
    </bean>

    <!-- 复杂对象调用 -->
    <bean id="AddressManager"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name">
        <value>AddressManager</value>
    </property>
    <property name="namespace">

    <value>http://server.selfobj.xfire.webservice/AddressManager</val
ue>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.selfobj.server.AddressManager</value>
    </property>
    <property name="serviceBean">
        <ref bean="addressManager"/>
    </property>
    </bean>

    <!-- 返回值是MAP的调用 -->
    <bean id="AddressManagerMap"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="namespace">

    <value>http://selfmap.xfire.webservice/IAddressManagerMap</value>
    </property>
    <property name="name">
        <value>AddressManagerMap</value>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.selfmap.IAddressManagerMap</value>
    </property>
    <property name="serviceBean">
        <ref bean="addressMap"/>
    </property>
```

```
    </bean>


    <!-- handler -->
    <bean id="HelloWordHandler"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="namespace">

    <value>http://handlers.xfire.webservice/HelloWordHandler</value>
    </property>
    <property name="name">
        <value>HelloWordHandler</value>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.handlers.IHelloWorldService</value>
    </property>
    <property name="serviceBean">
        <ref bean="helloWorldService"/>
    </property>
    <property name="inHandlers">
        <list>
            <ref bean="serverHelloWordInHandler"/>
        </list>
    </property>
    <property name="outHandlers">
        <list>
            <ref bean="serverHelloWordOutHandler"/>
        </list>
    </property>
    </bean>

    <!-- 异常 -->
    <bean id="HelloWordException"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="namespace">

    <value>http://exception.xfire.webservice/HelloWordException</valu
e>
    </property>
    <property name="name">
        <value>HelloWordException</value>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.exception.IHelloWorldService</value>
```

```
    </property>
    <property name="serviceBean">
        <ref bean="helloWorldServiceImpl"/>
    </property>
    </bean>

    <!-- 文件传递 -->
    <bean id="FileTransport"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name">
        <value>FileTransport</value>
    </property>
    <property name="namespace">

    <value>http://filetransport.xfire.webservice/FileTransport</value
>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.filetransport.IFileTransport</value>
    </property>
    <property name="serviceBean">
        <ref bean="fileTransportImpl"/>
    </property>
    <property name="properties">
        <map>
            <entry key="mtom-enabled">
                <value>true</value>
            </entry>
        </map>
    </property>
    </bean>

    <bean id="helloWorldSimple"
class="webservice.xfire.simple.HelloWorld"/>

    <bean id="addressManager"
class="webservice.xfire.selfobj.server.AddressManager"/>

    <bean id="addressMap"
class="webservice.xfire.selfmap.server.AddressManagerMap"/>

    <bean id="helloWorldService"
class="webservice.xfire.handlers.server.HelloWorldService"/>
```

```
    <bean id="serverHelloWordInHandler"
class="webservice.xfire.handlers.server.ServerHelloWordInHandler" />

    <bean id="serverHelloWordOutHandler"
class="webservice.xfire.handlers.server.ServerHelloWordOutHandler" />

    <bean id="helloWorldServiceImpl"
class="webservice.xfire.exception.server.HelloWorldServiceImpl"/>

    <bean id="fileTransportImpl"
class="webservice.xfire.filetransport.server.FileTransportImpl"/>
</beans>
```

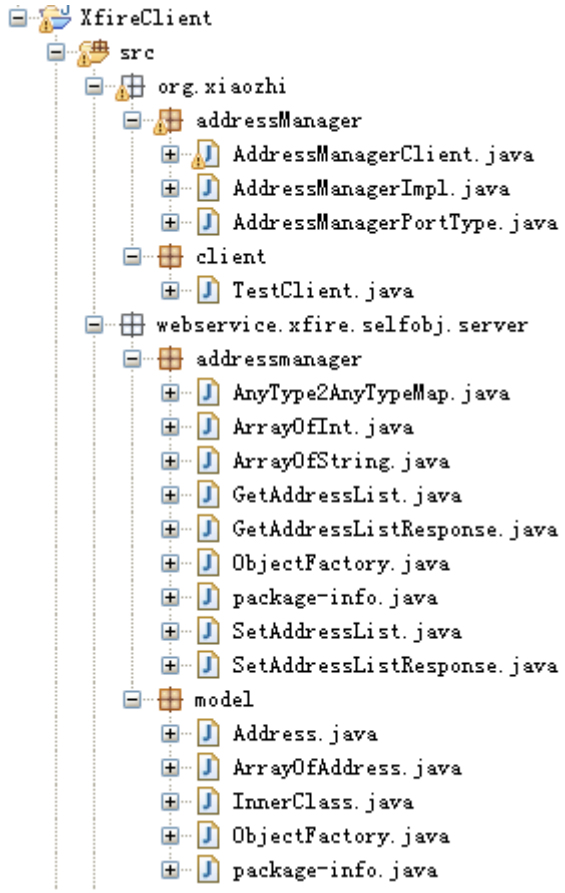
在 xfire-servlet.xml 里面的代码都是 4.2 和 4.3 你写的代码只不过在这是和 Spring 做了结合，测试结果也应该和上面的一样就对了。

3.5 使用 WSDL 生成客户端

在很多项目中往往用户只给你提供了 WSDL 并不给你提供任何数据，所以需要你自己通过 Xfire 本身提供的 API 来生成客户端代码。
一般在开发过程中使用 ANT 进行客户端生成，在这里就以它为例。
Ant 生成客户端代码：

```
    <taskdef classpathref="devlib.classpath"  name="wsgen"
classname="org.codehaus.xfire.gen.WsGenTask" />
    <target name="wsgen" description="generate client">
          <wsgen outputDirectory="./src/"
wsdl="http://127.0.0.1:8080/myfire/services/AddressManager?wsdl"
binding="jaxb" package="org.xiaozhi.addressManager" overwrite="true"
/>
</target>
```

在这里说明2个地方：
第一classpathref这个里面的devlib.classpath是你所有的包（*.jar）
第二 binding 有下两种不同的方式：
(1)jaxb
(2)xmlbeans
这里注意：如果使用xmlbeans你需要先生成wsdl的XMLBeans。所以在这里我们使用jaxb
这里使用List的那个例子为例：
通过上面的ant你会生成如下结构：



这就是用Ant帮你生成的 但是里面的TestClient.java是由我自己编写的客户端。

先贴出代码在做相关解释 如下：

```
/*
 * 文件名： TestClient.java
 *
 * 创建日期： 2008-12-12
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package org.xiaozhi.client;

import java.util.List;

import org.xiaozhi.addressManager.AddressManagerClient;
import org.xiaozhi.addressManager.AddressManagerPortType;
```

```
import webservice.xfire.selfobj.server.model.Address;
import webservice.xfire.selfobj.server.model.ArrayOfAddress;
import webservice.xfire.selfobj.server.model.ObjectFactory;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-12-12
 */
public class TestClient {

    /**
     * @param args
     */
    public static void main(String[] args) {

        getAddressList() ;

        setAddressList() ;
    }

    public static void getAddressList(){

        AddressManagerClient client = new AddressManagerClient();

        AddressManagerPortType service =
client.getAddressManagerHttpPort();

        ArrayOfAddress address = service.getAddressList() ;

        List<Address> listAddress = address.getAddress() ;

        for(int i = 0 ; i < listAddress.size() ; i++){

            Address addressVO = listAddress.get(i) ;

            System.out.println(addressVO.getAddress().getValue()) ;
        }
    }
```

```
    public static void setAddressList(){

        AddressManagerClient client = new AddressManagerClient();

        AddressManagerPortType service =
client.getAddressManagerHttpPort();

        //自动生成的注册类如果创建类或者创建类里属性的时候要使用他来创建
        ObjectFactory factory = new ObjectFactory() ;


        Address addressVO = factory.createAddress() ;

        addressVO.setAddress(factory.createAddressAddress("天上人间")) ;

        addressVO.setCity(factory.createAddressCity("天才堂")) ;

        ArrayOfAddress address = new ArrayOfAddress() ;

        address.getAddress().add(addressVO) ;

        ArrayOfAddress list = service.setAddressList(address) ;

        List<Address> addressList = list.getAddress() ;

        for(int i = 0 ; i < addressList.size() ; i++){

            Address addressObj = addressList.get(i) ;

            System.out.print(addressObj.getAddress().getValue()) ;
        }
    }
}
```

这里说明一下：

AddressManagerClient ：是客户端

AddressManagerPortType：是对应接口

ObjectFactory：自动生成的注册类如果创建类或者创建类里属性的时候要使用他来创建

JAXBElement：这个就是你通过jaxb方式生成的，主要用于读取XML的Element。

以上就是通过WSDL来生成的客户端类。在这里在说明一下：一般考虑到安全别人会给你一个WSDL你需要修改一下WSDL里面的

```
<wsdlsoap:address
    location="http://localhost8080/serviceagent/remoting/ServiceDataFacade" />
```

一般人家是为了本地做测试，所以在你使用的时候要改成他的外网地址才行然后在生成。

3.6. SOAP 头进行验证

WebService 的安全策略有很多在这里我们这介绍用 soa 的头验证的方式。
XFire 是通过 Handler 进行报文的发送和接受的。现在我们来简单的看一个代码实例：
服务端代码：

```
/*
 * 文件名： ServerHeader.java
 *
 * 创建日期： 2008-12-29
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package webservice.xfire.soaheader.server;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.XFireRuntimeException;
import org.codehaus.xfire.handler.AbstractHandler;
import org.jdom.Element;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-12-29
 */
public class ServerHeader extends AbstractHandler {
    /*
     *
     * (non-Javadoc)
     *
     * @see
     *
org.codehaus.xfire.handler.Handler#invoke(org.codehaus.xfire.MessageC
```

```
ontext
     * )
     */
    public void invoke(MessageContext message) throws Exception {
        Element header = message.getInMessage().getHeader();

        if (header == null) {
            throw new XFireRuntimeException("Missing SOAP header");
        }

        Element token = header.getChild("AuthenticationToken");

        String username = token.getChild("username").getValue();

        System.out.println("用户名===" + username);

        System.out.println("header"+header.toString()) ;

    }

}
```

在 XFire 里通过继承 AbstractHandler 可以想头里写东西也可以获取头的信息。以上是获取头的信息。其实这个 AbstractHandler 就想一个拦截器一样就是在你调用服务端方法的时候做了一个验证的事情。除了以上代码的编写你还要声明一下请看代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <!-- 简单的调用 -->
    <bean id="HelloWorldSimple"
class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name">
        <value>HelloWorldSimple</value>
    </property>
    <property name="namespace">
        <value>http://simple.xfire.webservice/HelloWorld</value>
    </property>
    <property name="serviceClass">
        <value>webservice.xfire.simple.HelloWorld</value>
    </property>
    <property name="serviceBean">
        <ref bean="helloWorldSimple"/>
    </property>
    <property name="inHandlers">
```

```
            <list>
                <ref bean="serverHeader" />
            </list>
        </property>
    </bean>
    <bean id="helloWorldSimple"
class="webservice.xfire.simple.HelloWorld"/>
    <bean id="serverHeader"
class="webservice.xfire.soaheader.server.ServerHeader"/>
</beans>
```

以上是和 Spring 结合使用的。这样就完成了头的配置和编写。

这样你每次调用服务里的方法时就要写头的信息也就是客户端要做的代码如下：

```
/*
 * 文件名： TestHellword.java
 *
 * 创建日期： 2008-12-30
 *
 * Copyright(C) 2008, by xiaozhi.
 *
 * 原始作者: <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 */
package org.xiaozhi.client;

import java.lang.reflect.Proxy;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.client.Client;
import org.codehaus.xfire.client.XFireProxy;
import org.codehaus.xfire.handler.AbstractHandler;
import org.jdom.Element;
import org.xiaozhi.helloWorld.HelloWorldSimpleClient;
import org.xiaozhi.helloWorld.HelloWorldSimplePortType;

/**
 *
 *
 * @author <a href="mailto:xiaozhi19820323@hotmail.com">xiaozhi</a>
 *
 * @version $Revision$
 *
 * @since 2008-12-30
 */
```

```
public class TestHellword extends AbstractHandler {

    public TestHellword() {

    }

    /**
     * @param args
     */
    public static void main(String[] args) {

        HelloWorldSimpleClient client = new HelloWorldSimpleClient();

        HelloWorldSimplePortType service =
        client.getHelloWorldSimpleHttpPort();

        XFireProxy proxy = (XFireProxy)
Proxy.getInvocationHandler(service);

        Client c = proxy.getClient();

        c.addOutHandler(new TestHellword());

        service.hello("xiaozhi");
    }

    /*
     *
     * (non-Javadoc)
     *
     * @see
     *
org.codehaus.xfire.handler.Handler#invoke(org.codehaus.xfire.MessageC
ontext
     * )
     */
    public void invoke(MessageContext message) throws Exception {

        Element el = new Element("Header");

        Element token = new Element("AuthenticationToken");

        Element username= new Element("username") ;
```

```
        username.addContent("hellworld") ;

        username.setText("xiaozhi123") ;

        token.addContent(username) ;

        el.addContent(token);

        System.out.println(el.getChild("AuthenticationToken"));

        message.getCurrentMessage().setHeader(el);
    }

}
```

这样就编写完事了，在这里不做过多解释里相信读者通过代码的展示可以明白的。

4.CXF 篇

4.1 CXF 简介

4.1.1 CXF 的由来

Apache CXF 项目是由 Objectweb Celtix 和 Codehaus XFire 合并成立的。Objectweb Celtix 是由 IONA 公司赞助、于 2005 年成立的开源 Java ESB 产品，XFire 则是业界知名的 SOAP 堆栈。合并后的 Apache CXF 融合该两个开源项目的功能精华，提供了实现 SOA 所需要的核心 ESB 功能框架，包括 SOA 服务创建，服务路由，及一系列企业级 QoS 功能。此次发布代表了 Apache CXF 开发人员及社区用户一年的努力结果，并标志 Apache CXF 软件的进一步成熟，成为实现 SOA 的优秀技术解决方案之一。2.1 版本的 CXF，已经是一个正式的 Apache 顶级项目。

4.1.2 CXF 的功能

CXF 提供了一套创建 SOA 服务的基础设施框架，用户由此可以按照自己喜欢的编程模式，利用 Apache CXF 提供的简单易用工具（包括 Maven 插件），创建适合 SOA 环境的任何 WEB 服务，包括 SOAP/HTTP 服务及 REST/HTTP 服务。Apache CXF 可扩展的插拔式架构不但支持 XML 消息格式和 HTTP 通信协议，而且还支持基于其他通信协议如 IIOP 和非 XML 消息格式如 CORBA CDL 或 JSON。

主要功能列表如下：

- 支持 JAX-WS 2.1，部署 JAX-WS 已经更新至 JAX-WS 2.1 规范。
- JAX-RS 0.6 REST 的初期部署基于服务框架。
- Javascript 客户端生成和支持，SOAP 基于于端点的可以有 Javascript 带着 JS URL 自动创建。
- CORBA 的约束力来自 Yoko，JAX-WS 客户端/服务器能使 IIOP 与 CORBA 进程进行交互。
- 支持 Java to WSDL、WSDL to Java、XSD to WSDL、WSDL to XML、WSDL to SOAP、WSDL to Service。
- 支持 XmlBeans 运行库，允许数据模型使用 XmlBeans。
- 支持 SOAP1.1&1.2、WSDL1&2、WS-Addressing、WS-Policy、WS-RM、WS-Security 和 WS-I BasicProfile。
- 支持 JAX-WS、JAX-WSA、JSR-181、SAAJ。
- Apache CXF 提供方便的 Spring 整合方法，可以通过注解、Spring 标签式配置来暴露 WebServices 和消费 WebServices。

4.2 CXF 开发

4.2.1 开发环境

在这里简单说下注意事项：当你使用的是 JDK1.5 的时候你就必须要有 jaxws-api-2.0.jar 这个包的支持.如果使用的是 JDK1.6 就不用使用这个包了。因为 1.6 里已经有了相关的实现 。

在 JDK1.5 的开发环境下我们建议你使用 CXF2.0 开发版本比较好。

4.2.2 简单的 CXF 应用

CXF 内置了 Jetty 应用服务器，当然也支持配置在别的 AppServer 上。我们不需要任何配置，就可以让 CXF 使用 Jetty 来进行发布。

首先定义一个服务端的类：

WebService 服务的提供者：
Java 代码：

```
/**
 * WebService服务端的实现类
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
@WebService
public class HelloWorld {
    public String sayHi(@WebParam(name="text") String text) {
        return "Hello " + text;
    }
}
```

服务端：
Java 代码：

```
/**
 * 简单的WebService服务端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
public class ServerSimple {

    protected ServerSimple() throws Exception {
        System.out.println("Starting Server");
        HelloWorld helloworldImpl = new HelloWorld();
        String address = "http://localhost:9000/helloWorld";
        Endpoint.publish(address, helloworldImpl);
    }

    public static void main(String[] args) throws Exception {
        new ServerSimple();
        System.out.println("Server ready...");
        Thread.sleep(60 * 1000);//60秒后退出。
        System.out.println("Server exiting");
        System.exit(0);
```

```
    }

}
```

通过 CXF 的 Endpoint 类提供的静态方法，可以很容易的发布 WebService。Jetty 默认的是 9000 端口。运行服务端，服务进程会等待 Client 的请求 60 秒。

客户端映射服务端服务的接口代码：
Java 代码：

```
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 * 客户端映射WS服务提供的业务方法的接口，方法名、参数类型与WS服务相同即可。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
@WebService
public interface IHelloWorld {
    public String sayHi(@WebParam(name="text") String text);
}
```

客户端：
Java代码：

```
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;

/**
 * 调用WebService的客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
public final class Client {

    private Client() {
    }
```

```
    /**
     * @param args
     */
    public static void main(String[] args) {
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.setServiceClass(IHelloWorld.class);
        factory.setAddress("http://localhost:9000/helloWorld");

        IHelloWorld client = (IHelloWorld)factory.create();
        System.out.println("Invoke sayHi()....");

System.out.println(client.sayHi(System.getProperty("user.name")));
        System.exit(0);
    }
}
```

客户端也没有什么复杂的地方，可以通过 JaxWsProxyFactoryBean 获得虚拟远程服务的业务对象的引用，像调用本地方法一样直接调用其方法。

运行结果：
服务端：

```
Starting Server
2008-7-17 10:20:52
org.apache.cxf.service.factory.ReflectionServiceFactoryBean
buildServiceFromClass
信息: Creating Service
{http://server.simple.cxf.webservice/}HelloWorldService from class
webservice.cxf.simple.server.HelloWorld
2008-7-17 10:20:53 org.apache.cxf.endpoint.ServerImpl initDestination
信息: Setting the server's publish address to be
http://localhost:9000/helloWorld
2008-07-17 10:20:53.390::INFO:  Logging to STDERR via
org.mortbay.log.StdErrLog
2008-07-17 10:20:53.421::INFO:  jetty-6.1.9
2008-07-17 10:20:53.702::INFO:  Started
SelectChannelConnector@0.0.0.0:9000
Server ready...
```

客户端：

```
2008-7-17 10:23:35
org.apache.cxf.service.factory.ReflectionServiceFactoryBean
buildServiceFromClass
信息: Creating Service
{http://client.simple.cxf.webservice/}IHelloWorldService from class
```

```
webservice.cxf.simple.client.IHelloWorld
Invoke sayHi()....
Hello QHL
```

4.2.3 CXF 对请求的拦截处理

CXF 可以像 AXIS 的 Handler 一样，在每个请求或响应的处理之前或者之后做一些处理，这些处理是通过 Interceptor 实现的。它及提供了很好的扩展性，让用户可以在服务被调用或响应前后做很多事情，极大的降低了代码的耦合度，使服务提供者专注的提供服务，而不用考虑其他的事情。我们来看看 Interceptor 是如何处理的。

服务端：
Java 代码：

```
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsServerFactoryBean;

import webservice.cxf.simple.server.HelloWorld;

/**
 * 带日志的WebService服务端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
public class Server {

    protected Server() throws Exception {
        HelloWorld helloworldImpl = new HelloWorld();
        JaxWsServerFactoryBean svrFactory = new
JaxWsServerFactoryBean();
        svrFactory.setServiceClass(HelloWorld.class);
        svrFactory.setAddress("http://localhost:9000/helloWorld");
        svrFactory.setServiceBean(helloworldImpl);
        svrFactory.getInInterceptors().add(new LoggingInInterceptor());
        svrFactory.getOutInterceptors().add(new
LoggingOutInterceptor());
        svrFactory.create();
    }
```

```
    public static void main(String[] args) throws Exception {
        new Server();
        System.out.println("Server ready...");
        Thread.sleep(60 * 1000);//60秒后退出。
        System.out.println("Server exiting");
        System.exit(0);
    }
}
```

和上一个例子没有太大的差别，这里只是多了两句：

```
svrFactory.getInInterceptors().add(new LoggingInInterceptor());
svrFactory.getOutInterceptors().add(new LoggingOutInterceptor());
```

第一句表示在接收客户端发送请求的时候做日志处理，第二句标识在服务端响应客户端请求的时候做日志处理。

同样客户端也可以做日志处理：

客户端：
Java 代码：

```
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;

import webservice.cxf.simple.client.IHelloWorld;

/**
 * 带输入输出拦截的WebService客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
public final class Client {

    private Client() {
    }

    public static void main(String[] args) {
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.getInInterceptors().add(new LoggingInInterceptor());
```

```
        factory.getOutInterceptors().add(new LoggingOutInterceptor());
        factory.setServiceClass(IHelloWorld.class);
        factory.setAddress("http://localhost:9000/helloWorld");

        IHelloWorld client = (IHelloWorld)factory.create();
        System.out.println("Invoke sayHi()....");

System.out.println(client.sayHi(System.getProperty("user.name")));
        System.exit(0);
    }
}
```

这里也是增加了 2 句代码，我们就不多说了，和之前讲的是一样的。

运行结果如下：
服务端：

```
2008-7-17 10:46:05
org.apache.cxf.service.factory.ReflectionServiceFactoryBean
buildServiceFromClass
信息: Creating Service
{http://server.simple.cxf.webservice/}HelloWorldService from class
webservice.cxf.simple.server.HelloWorld
2008-7-17 10:46:05 org.apache.cxf.endpoint.ServerImpl initDestination
信息: Setting the server's publish address to be
http://localhost:9000/helloWorld
2008-07-17 10:46:05.903::INFO:  Logging to STDERR via
org.mortbay.log.StdErrLog
2008-07-17 10:46:05.919::INFO:  jetty-6.1.9
2008-07-17 10:46:05.012::INFO:  Started
SelectChannelConnector@0.0.0.0:9000
Server ready...
2008-7-17 10:46:11 org.apache.cxf.interceptor.LoggingInInterceptor
logging
信息: Inbound Message
----------------------------
Encoding: UTF-8
Headers: {content-type=[text/xml; charset=UTF-8], connection=[keep-
alive], transfer-encoding=[chunked], Host=[localhost:9000],
SOAPAction=[""], User-Agent=[Java/1.6.0_06], Accept=[*], Pragma=[no-
cache], Cache-Control=[no-cache]}
Messages:
Message:

Payload: <soap:Envelope
```

```
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns
1:sayHi
xmlns:ns1="http://client.simple.cxf.webservice/"><text>QHL</text></ns
1:sayHi></soap:Body></soap:Envelope>
--------------------------------------
2008-7-17 10:46:11
org.apache.cxf.interceptor.LoggingOutInterceptor$LoggingCallback
onClose
信息: Outbound Message
---------------------------
Encoding: UTF-8
Headers: {SOAPAction=[""]}
Messages:
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns
1:sayHiResponse
xmlns:ns1="http://server.simple.cxf.webservice/"><return>Hello
QHL</return></ns1:sayHiResponse></soap:Body></soap:Envelope>
--------------------------------------
```

客户端：

```
2008-7-17 10:46:10
org.apache.cxf.service.factory.ReflectionServiceFactoryBean
buildServiceFromClass
信息: Creating Service
{http://client.simple.cxf.webservice/}IHelloWorldService from class
webservice.cxf.simple.client.IHelloWorld
Invoke sayHi()....
2008-7-17 10:46:11
org.apache.cxf.interceptor.LoggingOutInterceptor$LoggingCallback
onClose
信息: Outbound Message
---------------------------
Encoding: UTF-8
Headers: {SOAPAction=[""], Accept=[*]}
Messages:
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns
1:sayHi
xmlns:ns1="http://client.simple.cxf.webservice/"><text>QHL</text></ns
1:sayHi></soap:Body></soap:Envelope>
--------------------------------------
2008-7-17 10:46:11 org.apache.cxf.interceptor.LoggingInInterceptor
logging
```

```
信息: Inbound Message
----------------------------
Encoding: UTF-8
Headers: {content-type=[text/xml; charset=utf-8], Content-
Length=[223], SOAPAction=[""], Server=[Jetty(6.1.9)]}
Messages:
Message:

Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns
1:sayHiResponse
xmlns:ns1="http://server.simple.cxf.webservice/"><return>Hello
QHL</return></ns1:sayHiResponse></soap:Body></soap:Envelope>
--------------------------------------
Hello QHL
```

这里要讲两句的是 CXF 的所有拦截器都要实现 org.apache.cxf.interceptor.Interceptor 接口，或者继承自 CXF 提供的抽象类。一般我们只要继承自 AbstractPhaseInterceptor 这个抽象类就行了。

当然，像 AXIS 可以处理多个 Handler 一样，CXF 也可以有多个 Interceptor 在请求或响应之前、之后做处理，下面我们自己实现一个 Interceptor 看看效果。

自定义拦截器：
Java 代码：

```
/**
 * 自定义的Interceptor拦截器
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-17
 */
public class MyInterceptor extends AbstractPhaseInterceptor<Message>
{

    public MyInterceptor(String phase) {
        super(phase);
    }

    public MyInterceptor() {
        super(Phase.RECEIVE);
```

```
    }

    /*
     * (non-Javadoc)
     *
     * @see
org.apache.cxf.interceptor.Interceptor#handleFault(org.apache.cxf.mes
sage.Message)
     */
    public void handleFault(Message arg0) {
        System.out.println("Error");
    }

    /*
     * (non-Javadoc)
     *
     * @see
org.apache.cxf.interceptor.Interceptor#handleMessage(org.apache.cxf.m
essage.Message)
     */
    public void handleMessage(Message arg0) throws Fault {
        System.out.println("Hello, This is my Interceptor.");
    }
}
```

客户端代码：

Java 代码：

```
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
import org.apache.cxf.phase.Phase;

import webservice.cxf.interceptor.MyInterceptor;
import webservice.cxf.simple.client.IHelloWorld;

/**
 * 带输入输出拦截的WebService客户端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
```

```
public final class Client {

    private Client() {
    }

    public static void main(String[] args) {
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.getInInterceptors().add(new LoggingInInterceptor());
        factory.getInInterceptors().add(new MyInterceptor());
        factory.getOutInterceptors().add(new
MyInterceptor(Phase.SEND));
        factory.getOutInterceptors().add(new LoggingOutInterceptor());
        factory.setServiceClass(IHelloWorld.class);
        factory.setAddress("http://localhost:9000/helloWorld");

        IHelloWorld client = (IHelloWorld)factory.create();
        System.out.println("Invoke sayHi()....");

System.out.println(client.sayHi(System.getProperty("user.name")));
        System.exit(0);
    }
}
```

服务端代码：

Java 代码：

```
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsServerFactoryBean;
import org.apache.cxf.phase.Phase;

import webservice.cxf.interceptor.MyInterceptor;
import webservice.cxf.simple.server.HelloWorld;

/**
 * 带日志的WebService服务端
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-16
 */
public class Server {
```

```
    protected Server() throws Exception {
        HelloWorld helloworldImpl = new HelloWorld();
        JaxWsServerFactoryBean svrFactory = new
JaxWsServerFactoryBean();
        svrFactory.setServiceClass(HelloWorld.class);
        svrFactory.setAddress("http://localhost:9000/helloWorld");
        svrFactory.setServiceBean(helloworldImpl);
        svrFactory.getInInterceptors().add(new MyInterceptor());
        svrFactory.getInInterceptors().add(new LoggingInInterceptor());
        svrFactory.getOutInterceptors().add(new
MyInterceptor(Phase.SEND));
        svrFactory.getOutInterceptors().add(new
LoggingOutInterceptor());
        svrFactory.create();
    }

    public static void main(String[] args) throws Exception {
        new Server();
        System.out.println("Server ready...");
        Thread.sleep(60 * 1000);//60秒后退出。
        System.out.println("Server exiting");
        System.exit(0);
    }

}
```

这里有几点需要注意：

- 我们的拦截器是不针对服务端还是客户端的。就是说一个实现可以用在两个地方。
- 继承 AbstractPhaseInterceptor 至少要实现一个带参数的构造方法，我们实现了一个带 String 类型参数的构造方法和一个默认无参构造方法。在无参构造方法中，有一句：super(Phase.RECEIVE)；其中 Phase.RECEIVE 表示这个实例处理的 InInterceptors 过程，即便改实例被设置为 OutInterceptors，也不会起任何作用。同理 Phase.SEND 是处理 OutInterceptors 过程的，即便改实例被设置为 InInterceptors，也不会起任何作用。

4.2.4 CXF 和 Spring 集成开发

下面介绍和目前比较流行的框架 Spring 的结合使用。

在集成之前先说下要涉及到的包，在CXF 和 Spring 结合需要有 4 个包的支持就是 spring-beans.jar，spring-context.jar，spring-core.jar，spring-web.jar。这里提示一下这 4 个包最好用 Spring 里的别用 CXF 的否则会出现版本冲突。

4.2.4.1 List、Map、数组和自定义对象

服务端的对象：
Java 代码：

```
/*
 * 文件名： Address.java
 *
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.cxf.selfobj.server.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
 * 服务端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
@XmlType(name="ServerAddress")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;
```

```
    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;

    /** 邮编 */
    private String postalCode;

    private String[] array;

    private List<Integer> list;

    private Map<Integer, InnerClass> map;

    private boolean isExist;

    private InnerClass innC;

    @XmlType(name="ServerInnerClass")
    @XmlAccessorType(XmlAccessType.FIELD)
    public static class InnerClass implements Serializable {

        private static final long serialVersionUID = -2330738090948448510L;

        private String innerName = "static InnerClass";

        public InnerClass() {};

        public InnerClass(String innerName) {
            super();
            this.innerName = innerName;
        }

        public String getInnerName() {
            return innerName;
        }
```

```
    public void setInnerName(String innerName) {
        this.innerName = innerName;
    }
}

public Address() {
    list = new ArrayList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);

    map = new HashMap<Integer, InnerClass>();
    map.put(1, new InnerClass("A"));
    map.put(2, new InnerClass("B"));
    map.put(3, new InnerClass("C"));

    innC = new InnerClass();
    innC.setInnerName("服务端: Address.InnerClass");
}

@Override
public String toString() {
    String returnStr = super.toString()
        + "id号: " + getIdentifier()
        + " address: " + getAddress()
        + " city: " + getCity()
        + " country: " + getCountry()
        + " postalCode: " + getPostalCode()
        + " province: " + getProvince()
        + " array: " + getArray()[0]
        + " list: " + getList()
        + " map: " + getMap()
        + " isExist: " + isExist()
        + " innerClass.name: " + getInnC().getInnerName();
    return returnStr;
}

public InnerClass getInnC() {
    return innC;
}

public void setInnC(InnerClass innC) {
    this.innC = innC;
}
```

```
public Integer getIdentifier() {
    return identifier;
}

public void setIdentifier(Integer identifier) {
    this.identifier = identifier;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}
```

```
    public void setPostalCode(String postalCode) {
        this.postalCode = postalCode;
    }

    public String[] getArray() {
        return array;
    }

    public void setArray(String[] array) {
        this.array = array;
    }

    public boolean isExist() {
        return isExist;
    }

    public void setExist(boolean isExist) {
        this.isExist = isExist;
    }

    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

    public Map<Integer, InnerClass> getMap() {
        return map;
    }

    public void setMap(Map<Integer, InnerClass> map) {
        this.map = map;
    }

}
```

服务端的方法：

Java 代码：

```
/*
 * 文件名： AddressManager.java
 *
```

```
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.cxf.selfobj.server;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.jws.WebService;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;

import webservice.cxf.selfobj.server.model.Address;


/**
 * 提供复杂对象的WebService业务
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
@WebService
public class AddressManager {

    private int requestCount = 0;

    public List<Address> getAddressList() {
        requestCount++;
        System.out.println("requestCount=" + requestCount);

        List<Address> returnList = new ArrayList<Address>();

        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
```

```
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});

        returnList.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        returnList.add(address);
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});

        return returnList;
    }

    public List<Address> setAddressList(List<Address> list) {
        requestCount++;
        System.out.println("requestCount=" + requestCount);
        return list;
    }

}
```

开始配置 Spring 的文件 applicationContext.xml ：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
         http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
         http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
         http://cxf.apache.org/jaxws
```

```
http://cxf.apache.org/schemas/jaxws.xsd"
      default-autowire="byName" default-lazy-init="true">

    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"
/>
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

    <bean id="address"
class="webservice.cxf.selfobj.server.AddressManager" />

    <bean id="inInter"
class="org.apache.cxf.interceptor.LoggingInInterceptor"/>

    <bean id="outInter"
class="org.apache.cxf.interceptor.LoggingOutInterceptor"/>

    <jaxws:server id="addressserver"
serviceClass="webservice.cxf.selfobj.server.AddressManager"
address="/Address">
       <jaxws:serviceBean>
          <ref bean="address"/> <!-- 要暴露的 bean 的引用 -->
        </jaxws:serviceBean>
        <jaxws:inInterceptors>
           <ref bean="inInter"/>
           <ref bean="outInter"/>
        </jaxws:inInterceptors>
    </jaxws:server>
</beans>
```

在这里主要要注意红色的地方。

配置相应的 Web.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="2.4"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <!-- Spring ApplicationContext配置文件的路径  可使用通配符，多个路径用  ,
号分隔。此参数用于后面的Spring-Context loader -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
```

```
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>

    <!--Spring ApplicationContext 载入 -->
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener
-class>
    </listener>

    <listener>
        <listener-
class>org.springframework.web.util.IntrospectorCleanupListener</liste
ner-class>
    </listener>

    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>
            org.apache.cxf.transport.servlet.CXFServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

客户端对象：

Java 代码：

```
/*
 * 文件名： Address.java
 *
 * 创建日期： 2008-7-11
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.cxf.selfobj.client.model;
```

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
 * 客户端的自定义类型
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-11
 */
@XmlType(name="ClientAddress")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address implements Serializable{

    private static final long serialVersionUID = 5182870829593983607L;

    private Integer identifier;

    /** 地址 */
    private String address;

    /** 城市 */
    private String city;

    /** 省份 */
    private String province;

    /** 国家 */
    private String country;

    /** 邮编 */
    private String postalCode;

    private String[] array;
```

```
    private List<Integer> list;

    private Map<Integer, InnerClass> map;

    private boolean isExist;

    private InnerClass innC;

    @XmlType(name="ClientInnerClass")
    @XmlAccessorType(XmlAccessType.FIELD)
    public static class InnerClass implements Serializable {

     private static final long serialVersionUID = -2330738090948448510L;

     public InnerClass() {};

     public InnerClass(String innerName) {
            super();
            this.innerName = innerName;
        }

     private String innerName = "static InnerClass";

        public String getInnerName() {
            return innerName;
        }

        public void setInnerName(String innerName) {
            this.innerName = innerName;
        }

    }

    public Address() {
     list = new ArrayList<Integer>();
     list.add(1);
     list.add(2);
     list.add(3);

     map = new HashMap<Integer, InnerClass>();
     map.put(1, new InnerClass("A"));
     map.put(2, new InnerClass("B"));
     map.put(3, new InnerClass("C"));
```

```
        innC = new InnerClass();
        innC.setInnerName("客户端: Address.InnerClass");
    }

    @Override
    public String toString() {
        String returnStr = super.toString()
                + "id号: " + getIdentifier()
                + " address: " + getAddress()
                + " city: " + getCity()
                + " country: " + getCountry()
                + " postalCode: " + getPostalCode()
                + " province: " + getProvince()
                + " array: " + getArray()[0]
                + " list: " + getList()
                + " map: " + getMap()
                + " isExist: " + isExist()
                + " innerClass.name: " + getInnC().getInnerName();
        return returnStr;
    }

    public InnerClass getInnC() {
        return innC;
    }

    public void setInnC(InnerClass innC) {
        this.innC = innC;
    }

    public Integer getIdentifier() {
        return identifier;
    }

    public void setIdentifier(Integer identifier) {
        this.identifier = identifier;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
```

```
public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getProvince() {
    return province;
}

public void setProvince(String province) {
    this.province = province;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getPostalCode() {
    return postalCode;
}

public void setPostalCode(String postalCode) {
    this.postalCode = postalCode;
}

public String[] getArray() {
    return array;
}

public void setArray(String[] array) {
    this.array = array;
}

public boolean isExist() {
    return isExist;
}
```

```
    public void setExist(boolean isExist) {
        this.isExist = isExist;
    }

    public List<Integer> getList() {
        return list;
    }

    public void setList(List<Integer> list) {
        this.list = list;
    }

    public Map<Integer,  InnerClass> getMap() {
        return map;
    }

    public void setMap(Map<Integer,  InnerClass> map) {
        this.map = map;
    }

}
```

客户端接口：

Java 代码：

```
/*
 * 文件名： IClientAddressManager.java
 *
 * 创建日期： 2008-7-17
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.cxf.selfobj.client;

import java.util.List;
import java.util.Map;

import javax.jws.WebService;

import webservice.cxf.selfobj.client.model.Address;
```

```
/**
 * 本地实现一个Remote接口，其中包含远程WS的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程WS对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-17
 */
@WebService
public interface IClientAddressManager {

    public List<Address> getAddressList();

    public List<Address> setAddressList(List<Address> list);
}
```

客户端 JAVA 类：

```
/*
 * 文件名： Client.java
 *
 * 创建日期： 2008-7-17
 *
 * Copyright(C) 2008, by Along.
 *
 * 原始作者: <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 */
package webservice.cxf.selfobj.client;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;

import webservice.cxf.selfobj.client.model.Address;

/**
 * WebService客户端
```

```
 *
 * @author <a href="mailto:HL_Qu@hotmail.com">Along</a>
 *
 * @version $Revision$
 *
 * @since 2008-7-17
 */
public class Client {
    public static void getServerList() {
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.getInInterceptors().add(new LoggingInInterceptor());
        factory.getOutInterceptors().add(new LoggingOutInterceptor());
        factory.setServiceClass(IClientAddressManager.class);
        factory.setAddress("http://127.0.0.1:8089/mycxf/Address");

        IClientAddressManager client =
(IClientAddressManager)factory.create();
        List<Address> list = (ArrayList<Address>) client.getAddressList();

        System.out.println("getServerList begin----------------") ;

        System.out.println("List size: " + list.size());

        for (Iterator<Address> iter = list.iterator(); iter.hasNext();) {

            Address address = iter.next();
            System.out.println(address);
        }

        System.out.println("getServerList end----------------") ;
    }

    public static void setServerList() {
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.getInInterceptors().add(new LoggingInInterceptor());
        factory.getOutInterceptors().add(new LoggingOutInterceptor());
        factory.setServiceClass(IClientAddressManager.class);
        factory.setAddress("http://127.0.0.1:8089/mycxf/Address");

        IClientAddressManager client =
(IClientAddressManager)factory.create();

       List<Address> list = new ArrayList<Address>();
```

```
        Address address = new Address();
        address.setIdentifier(1);
        address.setAddress("Haidian");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(false);
        address.setArray(new String[]{"1","2","3"});
        list.add(address);

        address = new Address();
        address.setIdentifier(2);
        address.setAddress("Chaoyang");
        address.setCity("BeiJing");
        address.setCountry("China");
        address.setPostalCode("100081");
        address.setProvince("Beijing");
        address.setExist(true);
        address.setArray(new String[]{"A","B","C"});
        list.add(address);

        List<Address> returnList = client.setAddressList(list);
        System.out.println("List size: " + returnList.size());

        for (Iterator<Address> iter = returnList.iterator(); iter.hasNext();)
{

            address = iter.next();
            System.out.println(address);
        }
    }



    public static void main(String[] args) {
        getServerList();

        setServerList();

    }
}
```

输出结果：

客户端：

```
getServerList begin---------------
List size: 2
webservice.cxf.selfobj.client.model.Address@d480eaid号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map:
{2=webservice.cxf.selfobj.client.model.Address$InnerClass@15b8520,
1=webservice.cxf.selfobj.client.model.Address$InnerClass@18105e8,
3=webservice.cxf.selfobj.client.model.Address$InnerClass@1aacd5f}
isExist: false innerClass.name: 服务端: Address.InnerClass
webservice.cxf.selfobj.client.model.Address@913dc1id号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map:
{2=webservice.cxf.selfobj.client.model.Address$InnerClass@16d8196,
1=webservice.cxf.selfobj.client.model.Address$InnerClass@56b93a,
3=webservice.cxf.selfobj.client.model.Address$InnerClass@19abd2b}
isExist: true innerClass.name: 服务端: Address.InnerClass
getServerList end---------------
List size: 2
webservice.cxf.selfobj.client.model.Address@edbe39id号: 1 address:
Haidian city: BeiJing country: China postalCode: 100081 province: Beijing
array: 1 list: [1, 2, 3] map:
{2=webservice.cxf.selfobj.client.model.Address$InnerClass@639bf1,
1=webservice.cxf.selfobj.client.model.Address$InnerClass@1931579,
3=webservice.cxf.selfobj.client.model.Address$InnerClass@166bfd8}
isExist: false innerClass.name: 客户端: Address.InnerClass
webservice.cxf.selfobj.client.model.Address@bd09e8id号: 2 address:
Chaoyang city: BeiJing country: China postalCode: 100081 province:
Beijing array: A list: [1, 2, 3] map:
{2=webservice.cxf.selfobj.client.model.Address$InnerClass@d58ce2,
1=webservice.cxf.selfobj.client.model.Address$InnerClass@14627a,
3=webservice.cxf.selfobj.client.model.Address$InnerClass@4d41e2}
isExist: true innerClass.name: 客户端: Address.InnerClass
```

表示你已经成功，但是在这里说下关于直接通过方法返回 MAP 的时候不行，我们也试验过 CXF 自带的 DEMO 他上面是说使用对 XmlAdapter 的集成来实现但是没有成功。希望有研究的朋友能够加入本群或者联系我和 Along 我们一起讨论下。

5. 后话

以上就是我们对 AXIS1.4、XFire、 CXF 的全部理解。写这篇文档的目的是分享我们的知识，希望能给大家提供方便。以上代码是全部经过我们测试成功的，如果有那些不对的地方希望朋友们能够指出，也可以直接加 QQ 群：3961326 来一起探讨关于 WebService 方面的

问题。

发现什么错误或者愿意一起丰富书中内容的朋友，可以加作者的 QQ 来一起为中国开发力量做贡献。