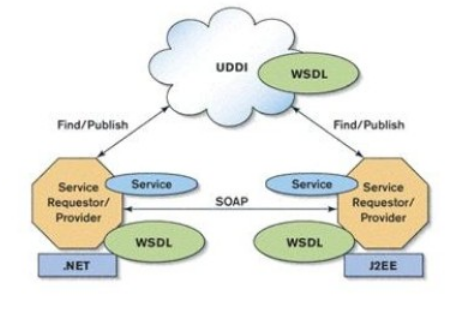
**Webservice**

## 什么是webservice



1 Webservice 即是web服务，它是一种跨越编程语言和跨操作系统平台的远程调用技术即跨平台远程调用技术webservice是一种跨平台，跨语言的规范，用于不同平台，不同语言开发的应用之间的交互。 比如，平台淘宝、京东想获取其他快递公司数据接口，需快递公司开放数据接口。那么 webservice就是出于以上类似需求而定义出来的规范；无需关心对方什么平台上开发以及使用何种语言开发。只关心调用对方发布webservice接口的一些对我们获取有用数据的方法。开发人员一般就是在具体平台开发webservice接口，以及调用webservice接口；每种开发语言都有自己的webservice实现框架。比如Java 就有**Apache Axis1、Apache Axis2、Codehaus XFire、Apache CXF、Apache Wink、Jboss  RESTEasyd等。**

2 采用标准SOAP(Simple Object Access Protocol) 协议传输，soap属于w3c标准。Soap协议是基于http的应用层协议，soap协议传输是xml数据。

3 采用wsdl作为描述语言即webservice使用说明书，wsdl属w3c标准。

4 xml是webservice的跨平台的基础，XML主要的优点在于它既与平台无关，又与厂商无关。

5 XSD，W3C为webservice制定了一套传输数据类型，使用xml进行描述，即XSD（XML Schema Datatypes)，任何编程语言写的webservice接口在发送数据时都要转换成webservice标准的XSD发送。

6 当前非SOAP协议的webService以轻量为首要目标，比如http rest方式也是webservice的一种方式，或者直接使用http自定义数据协议，比如http传输json数据，http传输xml数据等。

**7 webservice优点**

（1）、采用xml支持跨平台远程调用。

（2）、基于http的soap协议，可跨越防火墙。

（3）、支持面向对象开发。

（4）、有利于软件和数据重用，实现松耦合。

**8 webservice缺点**

由于soap是基于xml传输，本身使用xml传输会传输一些无关的东西从而效率不高，随着soap协议的完善，soap协议增加了许多内容，这样就导致了使用soap协议进行数据传输的效率不高。

**9 webService的应用场景**

（1）宏观：用于软件集成和复用

（2）微观：用于接口服务：不考虑客户端类型，不考虑性能，建议使用用于公开接口服务：面向互联网公开的接口，比如火车时刻查询接口等等。用于内部接口服务：一个大的系统是由若干个系统组成的，系统与系统之间存在数据访问需求，为了减少系统与系统之间的耦合性可以将接口抽取出来单独的接口服务供给其他系统调用。

（3）适用：用于接口，不考虑客户端类型，不考虑性能，建议使用服务端已经确定使用webservice，客户端无法选择，只能使用webservice

（4）不适用：对性能要求很高的应用，不建议使用webservice，比如银行交易系统、股票交易系统等，任何延迟都可能造成无法估量的损失。同构程序之间通信不建议使用webservice， 比如Java的RMI同样可以实现远程调用，而且性能比webservice好很多。

## 二、WebService的三要素

**1 SOAP：**

简单对象访问协议（Simple Object Access Protocal ）是一种简单的基于XML的协议，他使应用程序通过HTTP来交换信息，简单理解为soap=http+xml。soap协议版本主要使用soap1.1和soap1.2.SOAP不是webservice的专有协议，其他应用协议也使用soap传输数据。例如，SMTP、tr069等。

**2 wsdl：**

wsDL:是基于XML的用于描述webservice及其函数、参数和返回值。通俗理解为wsdl是webservice的使用说明书。

**3 UUDI：**

是一种目录服务，通过它，企业可注册并搜索 Web services。企业将自己提供的Web Service注册在UDDI，也可以使用别的企业在UDDI注册的web service服务，从而达到资源共享。 UDDI旨在将全球的webservcie资源进行共享，促进全球经济合作。

## 三、webservice开发规范

JAVA 中共有三种WebService 规范，分别是**JAX-WS（JAX-RPC）、JAXM&SAAJ、JAX-RS。**目前使用较多的是jax-ws和jax-rs. jax-rs推出的时间不久，所以一般大公司用的比较多，jax-ws小公司用的多。

**1 JAX-WS：** Java API for XML-BasedWebservices 。早期的基于SOAP 的JAVA 的Web 服务规范JAX-RPC（Java API For XML-Remote Procedure Call）目前已经被JAX-WS 规范取代。

**2 JAX&SAAJ：**JAVA API FOR XML Message。SAAJ（SOAP With Attachment APIFor Java，JSR 67）是与JAXM 搭配使用的API，为构建SOAP 包和解析SOAP 包提供了重要的支持，支持附件传输等，JAXM&SAAJ 与JAX-WS 都是基于SOAP 的Web 服务，相比之下JAXM&SAAJ 暴漏了SOAP更多的底层细节，编码比较麻烦，而JAX-WS 更加抽象，隐藏了更多的细节，更加面向对象，实现起来你基本上不需要关心SOAP 的任何细节

**3 JAX-RS:JAX-RS** 是JAVA 针对REST(Representation State Transfer)风格制定的一套Web 服务规范，由于推出的较晚，该规范（JSR 311，目前JAX-RS 的版本为1.0）并未随JDK1.6 一起发行。

## 四、四种客户端调用方式

**1 客户端服务器模式**

**（1）服务器端开发**

编写SEI（Service Endpoint Interface）,SEI在webService中称为portType，在java中成为接口。

public interface WeatherInterface {

//天气查询

public String queryWeather(String cityName);

}

**（2）编写SEI实现类并发布，此类为webService提供服务类**

@WebService

public class WeatherInterfaceImpl implements WeatherInterface {

@Override

public String queryWeather(String cityName) {

System.out.println("from client.."+cityName);

String result = "晴朗";

System.out.println("to client..."+result);

return result;

}

public static void main(String[] args) {

//发送webservice服务

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

}

}

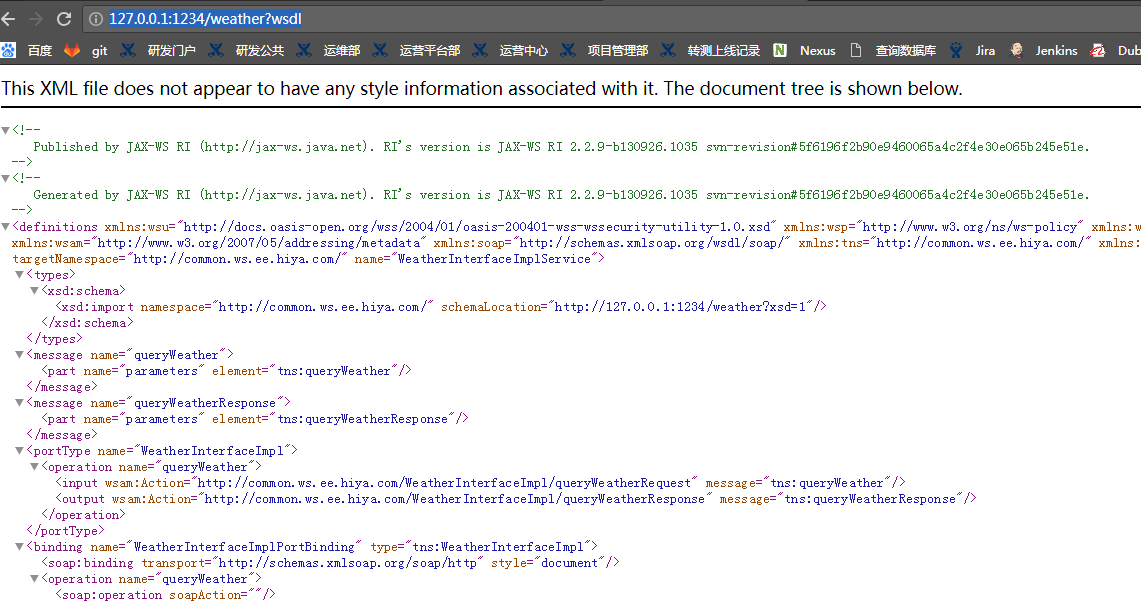
注意：SEI实现类中至少要有一个非静态的公共方法需要作为webservice服务的方法，。public class上边要加上@webservice

**（3）查看wsdl:**

在地址栏输入(注意后面的参数?wsdl)

<http://127.0.0.1:1234/weather?wsdl>

Wsdl不是webService,只是获取一个用于描述WebService的说明文件wsdl- WebServiceDescriptionLanguage,是以XML文件形式来描述WebService的”说明书”,有了说明书,我们才可以知道如何使用或是调用这个服务.



**（4）Wsimport生成客户端调用类**

wsimport是jdk自带的webservice客户端工具,可以根据wsdl文档生成客户端调用代码(java代码).当然,无论服务器端的WebService是用什么语言写的,都可以生成调用webservice的客户端代码，服务端通过客户端代码调用webservice。

**（5）wsimport.exe位于JAVA\_HOME\bin目录下.**

常用参数为:

-d<目录> - 将生成.class文件。默认参数。

-s<目录> - 将生成.java文件。

-p<生成的新包名> -将生成的类，放于指定的包下。

(wsdlurl) - http://server:port/service?wsdl，必须的参数。

**（6）示例：**

C:/> wsimport -keep -s F:\goto\HIYA\4\_HIYA\_JOBS\_SELF\code -verbose http://127.0.0.1:1234/weather?wsdl

**keep -d D:\temp\d -s D:\temp\s -p com.map -verbose <http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl>**

-keep：是否生成java源文件

-d：指定.class文件的输出目录

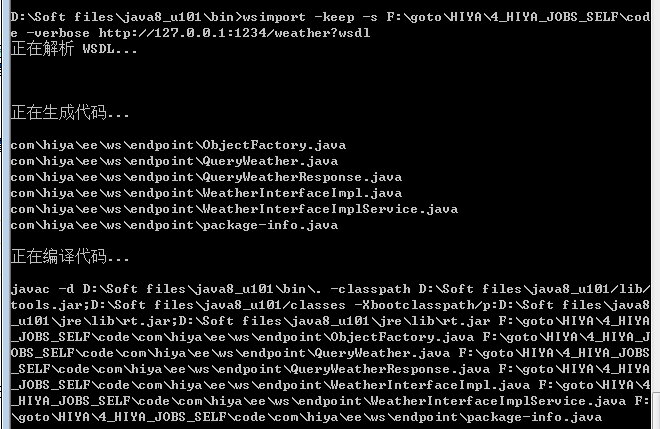
-s：指定.java文件的输出目录

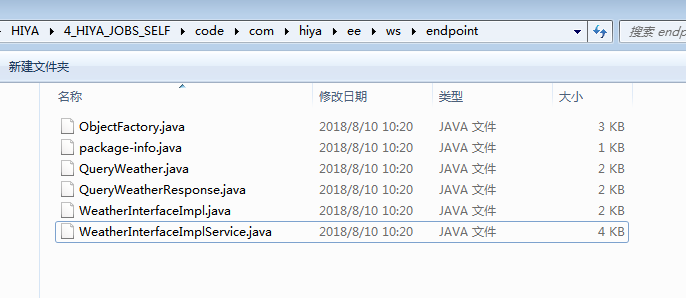
-p：定义生成类的包名，不定义的话有默认包名

-verbose：在控制台显示输出信息

-b：指定jaxws/jaxb绑定文件或额外的schemas

-extension：使用扩展来支持SOAP1.2





**（7）客户端编写（Wsimport实现）：**

public class WeatherClient {

public static void main(String[] args) {

//创建服务视图

WeatherInterfaceImplService weatherInterfaceImplService =new WeatherInterfaceImplService();

//通过服务视图得到服务端点

WeatherInterfaceImpl weatherInterfaceImpl= weatherInterfaceImplService.getPort(WeatherInterfaceImpl.class);

//调用webservice服务方法

String result = weatherInterfaceImpl.queryWeather("郑州");

System.out.println(result);

}

}

**2 service方式**

**（1）客户端编写（service实现）**

/\*\*

\* 使用javax.xml.ws.Service调用webservice服务

\* @author 传智播客 Java学院

\* @version V1.0

\*/

public class WeatherClient2 {

public static void main(String[] args) throws MalformedURLException {

//定义url，参数为wsdl地址

URL url = new URL("http://192.168.1.100:1234/weather?wsdl");

//定义qname，第一个参数是命名空间，第二个参数名称是wsdl里边的服务名

QName qName = new QName("http://impl.sei.jaxws.ws.itcast.cn/", "WeatherInterfaceImplService");

//创建服务视图

Service service = Service.create(url, qName);

//通过服务视图得到服务端点

WeatherInterfaceImpl weatherInterfaceImpl =service.getPort(WeatherInterfaceImpl.class);

//调用webservice

System.out.println(weatherInterfaceImpl.queryWeather("郑州"));

}

}

**（2）使用service调用和wsimport代码调用方式区别：**

Wsimport生成代码调用webservice无法指定webservice的地址，使用生成的服务视图类获取服务端点(postType)实例。Service调用Webservice可以指定webservice的地址，只需要服务端点的接口即可获取服务端点实例。

**（3）soap是什么**

SOAP 是一种网络通信协议

SOAP即Simple Object Access Protocol简易对象访问协议

SOAP 用于跨平台应用程序之间的通信

SOAP 被设计用来通过因特网(http)进行通信

SOAP ＝ HTTP+XML，其实就是通过HTTP发xml数据

SOAP 很简单并可扩展支持面向对象

SOAP 允许您跨越防火墙

SOAP 将被作为 W3C 标准来发展

使用TCP/IP Monitor监视soap协议

使用TCP/IP Monitor可以监视tcp/ip协议的报文内容，由于http是基于Tcp的应用协议，而webservice是基于http实现，所以通过tcp/ipmonitor可以监视webservice请求及响应的内容。

**（4）客户端编码：**

tcp/ip在eclipse的show view下边的other中输入tcp查找选择显示，然后通过下拉三角找到properties从而添加一个monitor

//定义url，参数为wsdl地址

URL url = new URL("http://127.0.0.1:54321/weather?wsdl");

//定义qname，第一个参数是命名空间，第二个参数名称是wsdl里边的服务名

QName qName = new QName("http://server.jaxws.webservice.itcast.cn/", "WeatherInterfaceImplService");

//创建服务视图

Service service = Service.create(url, qName);

//通过服务视图得到服务端点

WeatherInterfaceImpl weatherInterfaceImpl =service.getPort(WeatherInterfaceImpl.class);

//调用webservice

System.out.println(weatherInterfaceImpl.queryWeather("郑州"));

**（5）soap协议体重包含以下元素：**

必需有 Envelope 元素，此元素将整个 XML 文档标识为一条 SOAP 消息

可选的 Header 元素，包含头部信息

必需有Body 元素，包含所有的调用和响应信息

可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

**（6）soap消息基本结构部：**

<?xmlversion="1.0"?>

<soap:Envelopexmlns:soap="http://www.w3.org/2001/12/soap-envelope"soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>

... ...

</soap:Header>

<soap:Body>

... ...

<soap:Fault>

... ...

</soap:Fault>

</soap:Body>

</soap:Envelope>

**3 http发送soap协议方式**

public class HttpRequestSoap {

public static void main(String[] args) throws IOException {

//webservice地址

String webservice\_url = "http://127.0.0.1:1234/weather";

//发送的soap协议内容

String soap\_xml = soap\_xml("郑州");

System.out.println(soap\_xml);

//创建url

URL url = new URL(webservice\_url);

//创建http链接对象

HttpURLConnection httpURLConnection = (HttpURLConnection)url.openConnection();

//设置请求方法

httpURLConnection.setRequestMethod("POST");

//设置Content-type

httpURLConnection.setRequestProperty("Content-type", "text/xml;charset=\"utf-8\"");

//使用http进行输出

httpURLConnection.setDoOutput(true);

//使用http进行输入

httpURLConnection.setDoInput(true);

//通过输出流发送数据

OutputStream outputStream = httpURLConnection.getOutputStream();

outputStream.write(soap\_xml.getBytes());

outputStream.close();

//接收服务端响应数据

InputStream inputStream = httpURLConnection.getInputStream();

//使用buffer存在读取的数据

byte[] buffer = new byte[1024];

//使用字节输出流存储读取的数据

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

while(true){

int len = inputStream.read(buffer);

//如果流水读取完则退出循环

if(len == -1){

break;

}

byteArrayOutputStream.write(buffer,0,len);

}

//得到响应数据

String response\_string = byteArrayOutputStream.toString();

System.out.println(response\_string);

parseXml(response\_string);

}

//soap协议内容

public static String soap\_xml(String cityName){

String soap\_xml = "<?xml version=\"1.0\" encoding=\"utf-8\"?>"

+ "<S:Envelope xmlns:S=\"http://schemas.xmlsoap.org/soap/envelope/\">"

+ "<S:Body>"

+ "<ns2:queryWeather xmlns:ns2=\"http://impl.sei.jaxws.ws.itcast.cn/\">"

+ "<arg0>"+ cityName + "</arg0>"

+ "</ns2:queryWeather>"

+ "</S:Body>"

+ "</S:Envelope>";

return soap\_xml;

}

//解析响应的xml

public static String parseXml(String xmlString){

String result = null;

try {

Document document = DocumentHelper.parseText(xmlString);

//创建xpath解析对象

DefaultXPath defaultXPath = new DefaultXPath("//ns2:queryWeatherResponse");

//指定命名空间

defaultXPath.setNamespaceURIs(Collections.singletonMap("ns2", "http:// impl.sei.jaxws.ws.itcast.cn/"));

List<Element> elements= defaultXPath.selectNodes(document);

Element response = elements.get(0);

List<Element> results = response.selectNodes("return");

System.out.println(results.get(0).getText());

} catch (DocumentException e) {

e.printStackTrace();

}

return result;

}

}

**4 ajax方式**

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script type="text/javascript">

function queryMobile(){

//创建XMLHttpRequest对象

var xhr = new XMLHttpRequest();

//打开连接

xhr.open("post","http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx",true);

//设置数据类型

xhr.setRequestHeader("content-type","text/xml;charset=utf-8");

//设置回调函数

xhr.onreadystatechange=function(){

//判断是否发送成功和判断服务端是否响应成功

if(4 == xhr.readyState && 200 == xhr.status){

alert(xhr.responseText);

}

}

//组织SOAP协议数据

var soapXML = "<?xml version=\"1.0\" encoding=\"utf-8\"?>"

+"<soap:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"

+"<soap:Body>"

+"<getMobileCodeInfo xmlns=\"http://WebXml.com.cn/\">"

+"<mobileCode>"+document.getElementById("phoneNum").value+"</mobileCode>"

+"<userID></userID>"

+"</getMobileCodeInfo>"

+"</soap:Body>"

+"</soap:Envelope>";

alert(soapXML);

//发送数据

xhr.send(soapXML);

}

</script>

</head>

<body>

手机号查询：<input type="text" id="phoneNum"/> <input type="button" value="查询" onclick="javascript:queryMobile();"/>

</body>

</html>

**Cxf**

## 一、CXF简介

CXF （Celtix + XFire）是一个开源的Services框架。**CXF 帮助您利用 Frontend 编程 API 来构建和开发 Services** ，像 JAX-WS 。这些 Services 可以支持多种协议，比如：SOAP、XML/HTTP、RESTful HTTP 或者 CORBA ，并且可以在多种传输协议上运行，比如：HTTP、JMS 或者 JBI，CXF 大大简化了 Services 的创建，同时它继承了 XFire 传统

，一样可以天然地和 Spring 进行无缝集成。

## 二、功能特性

**1 CXF 包含了大量的功能特性，但是主要集中在以下几个方面**

（1）支持 Web Services 标准：CXF 支持多种 Web Services 标准，包含 SOAP、Basic Profile、WS-Addressing、WS-Policy、WS-ReliableMessaging 和 WS-Security。

（2）Frontends：CXF 支持多种“Frontend”编程模型，CXF 实现了 JAX-WS API （遵循 JAX-WS 2.0 TCK 版本），它也包含一个“simple frontend”允许客户端和 EndPoint 的创建，而不需要 Annotation 注解。CXF 既支持 （3）WSDL 优先开发，也支持从 Java 的代码优先开发模式。

（4）容易使用： CXF 设计得更加直观与容易使用。有大量简单的 API 用来快速地构建代码优先的 Services，各种 Maven 的插件也使集成更加容易，支持 JAX-WS API ，支持 Spring 2.0 更加简化的 XML 配置方式，等等。

（5）支持二进制和遗留协议：CXF 的设计是一种可插拨的架构，既可以支持 XML ，也可以支持非 XML 的类型绑定，比如：JSON 和 CORBA。

**2 支持多种标准**

支持 JAX-WS、 JAX-WSA、JSR-181 和 SAAJ；

支持 SOAP 1.1、1.2、WS-I BasicProfile、WS-Security、WS-Addressing、WS-RM 和 WS-Policy；

支持 WSDL 1.1 、2.0；

支持 MTOM；

**3 多种传输方式、Bindings、Data Bindings 和 Format**

Bindings：SOAP、REST/HTTP；

Data Bndings：目前支持 JAXB 2.0、Aegis 两种，默认是 JAXB 2.0。XMLBeans、Castor 和 JiBX 数据绑定方式将在 CXF 2.1 版本中得到支持；

格式（Format）：XML、JSON；

传输方式：HTTP、Servlet、JMS 和 Jabber；

可扩展的 API 允许为 CXF 增加其它的 Bindings，以能够支持其它的消息格式，比如：CSV 和固定记录长度。

**4 灵活部署**

轻量级容器：可在 Tomcat 或基于 Spring 的容器中部署 Services；

集成 JBI：可以在如 ServiceMix, OpenESB or Petals 等等的 JBI 容器中将它部署为一个服务引擎；

集成 SCA：可以部署在如 Tuscany 之类的 SCA 容器中；

集成 J2EE：可以在 J2EE 应用服务器中部署 Services，比如：Geronimo、JOnAS、JBoss、WebSphere Application Server 和 WebLogic Application Server，以及 Jetty 和 Tomcat；立的 Java 客户端／服务器。

**5 支持多种编程语言**

全面支持 JAX-WS 2.0 客户端／服务器编程模型；

支持 JAX-WS 2.0 synchronous、asynchronous 和 one-way API's；

支持 JAX-WS 2.0 Dynamic Invocation Interface (DII) API；

支持 wrapped and non-wrapped 风格；

支持 XML messaging API；

支持 JavaScript 和 ECMAScript 4 XML (E4X) ，客户端与服务端均支持；

通过 Yoko 支持 CORBA；

通过 Tuscany 支持 SCA；

通过 ServiceMix 支持 JBI ；

**6 代码生成**

Java to WSDL；

WSDL to Java；

XSD to WSDL；

WSDL to XML；

WSDL to SOAP；

WSDL to Service；

## 三、代码简单分析

1 通过查看 Server.java 及 Client.java 的代码，来了解一下 CXF 的运作过程。Server.java 中主要的代码片断如下，它利用 ServerFactoryBean 来进行 Web Services 的发布，实例化一个实现类 HelloWorldImpl，设置将要进行发布的地址 address，最后通过 ServerFactoryBean 的 create() 方法就成功地发布了 Web Services，如此简单而已，只有六行代码

1 HelloWorldImpl helloworldImpl = new HelloWorldImpl();

2 ServerFactoryBean svrFactory = new ServerFactoryBean();

3 svrFactory.setServiceClass(HelloWorld.class);

4 svrFactory.setAddress("http://localhost:9000/Hello");

5 svrFactory.setServiceBean(helloworldImpl);

6 svrFactory.create();

2 Client.java 中的主要代码片断如下，通过 ClientProxyFactoryBean 代理工厂类来创建一个服务，绑定到 endPointAddress 地址，就可以 create 并得到服务，并进行服务消费了：

1 ClientProxyFactoryBean factory = new ClientProxyFactoryBean();

2 factory.setServiceClass(HelloWorld.class);

3 factory.setAddress("http://localhost:9000/Hello");

4 HelloWorld client = (HelloWorld)factory.create();

5 System.out.println("Invoke sayHi()....");

6 System.out.println(client.sayHi("user"));

## 四、 CXF 应用开发

**1 接口类创建**

在项目的 src 目录中新建一个 ws.cxf 包，并在里面创建接口类 ISurveyService.java，为了简单示示例起见，我们仅创建一个方法 public String vote(String username,int point); 这里要注意的是我们在接口上用 @WebService 注解标明这是一个即将暴露为 Web Service 的接口，并将里面的方法都暴露出去。完整的接口代码清单如下：

<!--添加cxf支持 -->

2 <dependency>

3 <groupId>org.apache.cxf</groupId>

4 <artifactId>cxf-rt-frontend-jaxws</artifactId>

5 <version>3.1.9</version>

6 </dependency>

7 <dependency>

8 <groupId>org.apache.cxf</groupId>

9 <artifactId>cxf-rt-transports-http-jetty</artifactId>

10 <version>3.1.9</version>

11 </dependency>

12 <dependency>

13 <groupId>org.apache.cxf</groupId>

14 <artifactId>cxf-core</artifactId>

15 <version>3.1.9</version>

16 </dependency>

package ws.cxf;

import javax.jws.WebService;

@WebService

public interface ISurveyService

{

public String vote(String username,int point);

}

接下来，我们根据接口的定义，来实现它。

**2 具体类实现**

针对接口的定义，我们创建一个相应的实现类，并将其定义在 sw.cxf.impl 包中，完整的代码清单如下：

package ws.cxf.impl;

import javax.jws.WebService;

import ws.cxf.ISurveyService;

@WebService

public class SurveyService implements ISurveyService

{

private String excludeName = "Michael";

private int leastPonit = 5;

public String vote(String username,int point)

{

String result = "";

if(excludeName.equals(username))

{

result = " 您不能重复进行投票！";

}

else

{

result = " 谢谢您的投票！";

if(point < leastPonit)

{

result += " 您的投票分数太低！";

}

else

{

result += " 您的投票分数通过审核！";

}

}

return result;

}

public String getExcludeName()

{

return excludeName;

}

public void setExcludeName(String excludeName)

{

this.excludeName = excludeName;

}

public int getLeastPonit()

{

return leastPonit;

}

public void setLeastPonit(int leastPonit)

{

this.leastPonit = leastPonit;

}

}

接口定义与具体的实现就这样简单完成了，接下来就是相关的配置工作了，首先进行 Spring 的 Bean 配置。

**3 Spring 配置**

在 src 目录中创建 beanRefServer.xml 文件，用来定义 Spring 的 Bean 的配置，CXF 支持 Spring 2.0 Schema 标签配置方式，并且提供快捷暴露 Web Services 的标签。

（1）首先，我们需要引入 Spring 与 CXF 的命名空间（namespace），如下：

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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://cxf.apache.org/jaxws

http://cxf.apache.org/schemas/jaxws.xsd">

（2）这样，我们可以使用 Spring 与 CXF 的标签配置了。接着，我们需要引入我们所需要的 CXF 的 Bean 定义文件，如下：

<!-- Import Apache CXF Bean Definition -->

<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"/>

（3）接着定义我们具体实现的 Bean ，这个 Bean 的定义与 Spring 普通的 Bean 定义是一样的：

1 <!-- SurveyService -->

2 <bean id="surveyService" class="ws.cxf.impl.SurveyService">

3 <property name="excludeName" value="Michael"/>

4 <property name="leastPonit" value="10"/>

5 </bean>

（4）最后，将定义的 Bean 暴露出去成为 Web Service 服务，通过 CXF 提供的 Schema 标签配置 <jaxws:server> ，这样定义的配置显得更加简洁与方便，定义如下：

<!-- Expose SurveyWebService -->

<jaxws:server id="surveyWebService"

serviceClass="ws.cxf.ISurveyService"

address="/SurveyWebService">

<jaxws:serviceBean>

<ref bean="surveyService"/> <!-- 要暴露的 bean 的引用 -->

</jaxws:serviceBean>

</jaxws:server>

在配置中，serviceClass 的值是我们的接口类的名称，address 为将要暴露出去的 Web Service 访问地址。比如：/SurveyWebService，那么客户端消费 Web Service 的地址就会成为 http://host:port/WebAPPName/SurveyWebService ，与之相应的 WSDL 地址则为： http://host:port/WebAPPName/SurveyWebService?wsdl 。

**4 Web 应用配置**

（1）由于我们的示例是需要通过 Servlet 容器进行服务暴露，因此需要配置相对应的 web.xml 文件，首先是增加 Spring 的配置文件加载 Listener，如下：

1 <!-- Spring Config Location -->

2 <context-param>

3 <param-name>contextConfigLocation</param-name>

4 <param-value>/WEB-INF/classes/beanRefServer.xml</param-value>

5 </context-param>

6 <!-- Spring ContextLoaderListener -->

7 <listener>

8 <listener-class>

9 org.springframework.web.context.ContextLoaderListener

10 </listener-class>

11 </listener>

（2）接下来配置 CXF Servlet 的定义，以及它的映射，如下：

<!-- Apache CXFServlet -->

<servlet>

<servlet-name>CXFServlet</servlet-name>

<display-name>CXF Servlet</display-name>

<servlet-class>

org.apache.cxf.transport.servlet.CXFServlet

</servlet-class>

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

</servlet>

<!-- CXFServlet Mapping -->

<servlet-mapping>

<servlet-name>CXFServlet</servlet-name>

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

</servlet-mapping>

我们将之映射为 /\* 。这样，服务端的代码与配置就全部完成了，接下来就是将应用程序部署到 Web 容器中去，并验证服务是否正常发布。

**5 消费服务--服务器端写完接下来就可以利用客户端进行消费了**

（1）回到 Eclipse 开发平台，开始编写消费服务相关的代码，首先通过 Spring 与 CXF 的配置来定义 Web Service 的客户端 Bean，在 src 目录下创建 beanRefClient.xml 配置文件，同样，我们也需要引入 Spring 与 CXF 命名空间的声明，并引入 CXF 的 Bean 的定义文件，最后通过与服务端配置相对的 CXF 标签 <jaxws:client> 来定义客户端访问服务的声明，完整的定义内容如下：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:jaxws="http://cxf.apache.org/jaxws"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd

http://cxf.apache.org/jaxws

http://cxf.apache.org/schemas/jaxws.xsd">

<!-- Import Apache CXF Bean Definition -->

<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"/>

<!-- SurveyWebService Client -->

<jaxws:client id="surveyServiceClient"

serviceClass="ws.cxf.ISurveyService"

address="http://localhost:8080/CXF\_Spring\_Survey/SurveyWebService"/>

</beans>

（2） 定义说明：id 为 Spring 定义的 id，用来在程序里进行获取它的标识，serviceClass 仍是为服务端定义的接口类，address 为完整的 Web Service 地址，这个与服务端的定义不一样。

（3） 定义完配置文件，接下来我们编写访问的具体代码，在 test 目录下创建 ws.cxf.client 包，然后创建 SurveyServiceClient.java，完整的代码如下：

1 package ws.cxf.client;

3 import org.springframework.context.ApplicationContext;

4 import org.springframework.context.support.ClassPathXmlApplicationContext;

5 import ws.cxf.ISurveyService;

7 public class SurveyServiceClient

8 {

9 public static void main(String[] args)

10 {

11 // 加载客户端的配置定义

12 ApplicationContext context = new

13 ClassPathXmlApplicationContext("beanRefClient.xml");

14 // 获取定义的 Web Service Bean

15 ISurveyService surveyService =

16 (ISurveyService)context.getBean("surveyServiceClient");

17 // 1、定义调查投票的变量与内容，用来发送给服务

18 String username = "Test";

19 int point = 88;

20 // 调用方法进行服务消费

21 String result = surveyService.vote(username,point);

22 System.out.println("Result:" + result);

23 // 2、传递不一样的调查投票内容

24 username = "Michael";

25 point = 100;

26 // 再次调用方法进行服务消费，得到不一样的结果

27 result = surveyService.vote(username,point);

28 System.out.println("Result:" + result);

29 // 3、第三次传递与调用

30 username = "Jordan";

31 point = 9;

32 result = surveyService.vote(username,point);

33 System.out.println("Result:" + result);

34 }

35 }

**6 生成服务器端接口类**

（1）安装cxf

在cxf官网下载cxf的3.1.8版本，笔者下载的zip版。将文件解压缩并放到一个合适的目录下。在环境变量中加入变量“CXF\_HOME”，值就是安装路径；再在“PATH”变量中加入：

%CXF\_HOME%\bin

这样cxf的命令就可以使用了。

（2）使用“wsdl2java”命令生成接口类

wsdl2java的用法如下：

wsdl2java –p 包名 –d 目录名 wsdl路径

具体的意义如下：

-p：指定其wsdl的命名空间，也就是要生成代码的包名

-d：指定要产生代码所在目录

-client 生成客户端测试web service的代码

-server：生成服务器启动web service的代码

-impl：生成web service的实现代码

-ant：生成build.xml文件

-compile：生成代码后编译

-quient：静默模式，不输出警告与错误信息

-all：生成所有开始端点代码：types,service proxy,service interface, server mainline, client mainline, implementation object, and an Ant build.xml file.

**wsdl2java -p com.yhd.webservice.cxf.client.info -d D:\test http://localhost:8081/hello/person?wsdl**

（3）执行之后就会在-d后面的路径下生成相应的类。将接口类拷贝到客户端项目中

**Axis**

## 一、axis实战例子

**1、先创建一个web项目**，名字叫testAxis2Demo

**2、下载axit2 的war包**，地址是：

http://apache.fayea.com//axis/axis2/java/core/1.6.2/axis2-1.6.2-war.zip

**3、将下载axit2中war包下的WEB-INF/lib/ 下所有的jar包**，copy下来，放在web项目下的lib目录中

**4、在web.xml 中添加如下配置：**

<servlet>

<servlet-name>AxisServlet</servlet-name>

<servlet-class>org.apache.axis2.transport.http.AxisServlet</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>AxisServlet</servlet-name>

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

</servlet-mapping>

**5、在WebContent/ 目录下，创建一个文件夹axit2-web ，然后在该文件下创建一个名叫 listServices.jsp 的文件，**

该jsp中的代码如下：

<%@ page contentType="text/html;charset=UTF-8" language="java"%>

<%@

page

import="org.apache.axis2.Constants,

org.apache.axis2.description.AxisOperation,

org.apache.axis2.description.AxisService,

java.util.Collection,

java.util.HashMap,

java.util.Iterator"%><html>

<head>

<title>List Services</title>

<style>

h2 {

margin: 20 0 5 0;

}

ul {

margin-top: 5;

}

</style>

</head>

<body>

<h1>Available services</h1>

<%

HashMap serviceMap = (HashMap) request.getSession().getAttribute(Constants.SERVICE\_MAP);

Collection servicecol = serviceMap.values();

if(servicecol.size()==0){%>Available services is Empty.<%}

for (Iterator iterator = servicecol.iterator(); iterator.hasNext();) {

AxisService axisService = (AxisService) iterator.next();

Iterator opItr = axisService.getOperations();

String serviceName = axisService.getName();

%>

<h2>

<font color="blue"><a href="<%=serviceName %>?wsdl"

target="\_blank"><%=serviceName%></a></font>

</h2>

<i>Available Operations</i>

<ul>

<%

while (opItr.hasNext()) {

AxisOperation axisOperation = (AxisOperation) opItr.next();

%><li><%=axisOperation.getName().getLocalPart()%></li>

<%

}

%>

</ul>

<%

}

%>

</body>

</html>

**6、这样环境的搭建基本成功**

可以访问一下地址：http://localhost:8080/testAxis2Demo/services/listServices

注意：testAxis2Demo 这个是你项目的路径

如果出现下面信息，就说明搭建成功了

Available services

Available services is Empty.

**7、在WEB-INF 的文件下创建：services/dome/META-INF/services.xml**

目录是没有的，要自己创建

**8、services.xml 中必须的如下配置：**

<?xml version="1.0" encoding="UTF-8"?>

<serviceGroup>

<!-- 接口名 -->

<service name="testWebService">

<!-- 接口描述 -->

<description>测试接口</description>

<!-- 指定调用该接口会进行哪个类中去执行 -->

<parameter name="ServiceClass">com.webservice.control.TestWebservice</parameter>

<!-- 指定执行该类的哪个方法，并设置消息接收和返回的信息 -->

<parameter name="getMessege">

<!-- 设置消息交换模式：In-Out消息交换模式总是存在SOAP请求和应答 -->

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

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

</parameter>

</service>

</serviceGroup>

**9、在src下创建处理接口请求的类,在包com.webservice.control/TestWebservice.java，**

该java代码中如下：

package com.webservice.control;

public class TestWebservice {

public String getMessege(String params){

String info = null;

try{

info = "接口调入成功，传入的参数是："+params;

}catch (Exception e){

e.printStackTrace();

}

return info;

}

}

**10、此时，webservice的demo就成功创建好**了，自己可以用地址：http://localhost:8080/testAxis2Demo/services/testWebService?wsdl

在soapUI中进行调用。以下是该项目的war包，可以直接运行：http://download.csdn.net/detail/u012149894/8404661

## 二、发布web服务

**1、软件环境**

（1）、axis-1\_2 (从apache网站下载最新axis-bin-1\_2.zip解压即可）

（2）、Tomcat5.0

（3）、JDK5.0

**2、相关配置**

（1）、在你的%TOMCAT\_HOME%\common\lib下需要加入三个包 activation.jar、mail.jar、tools.jar

（2）、环境变量设置

AXIS\_HOME 即axis-bin-1\_2.zip解压的目录（我的是在F:\soap\axis-1\_2）

AXIS\_LIB 即 %AXIS\_HOME%\lib

AXISCLASSPATH即 %AXIS\_LIB%\axis.jar;%AXIS\_LIB%\commons-discovery- 0.2.jar;%AXIS\_LIB%\commons-logging-1.0.4.jar;%AXIS\_LIB% \jaxrpc.jar;%AXIS\_LIB%\saaj.jar;%AXIS\_LIB%\log4j-1.2.8.jar;也就是把%AXIS\_LIB%下所用JAR文件都导入

**3、实验一下**

在%AXIS\_HOME%\webapps下找到axis文件夹，将其整个拷贝到%TOMCAT\_HOME%\webapps下，启动

Tomcat,打开浏览器访问http://localhost:8080/axis/，出现以下页面说明你配置成功了。很简单吧：）

**4、发布我们的第一个程序**

第一个程序简单的返回HELLO WORLD！

public class HelloWorld {

public String sayHello()

{

return "HELLO WORLD!";

}

}

**（1）我们的第一种发布方式**

将HelloWorld.java拷贝到%TOMCAT\_HOME%\webapps\axis下，然后将其改名为HelloWorld.jws,这样AXIS就自然将其发布了。现在写个客户端程序访问一下：

TestClient.java

import org.apache.axis.client.Call;

import org.apache.axis.client.Service;

import javax.xml.rpc.ParameterMode;

public class TestClient

{

public static void main(String [] args) throws Exception {

String endpoint = "http://localhost:" +"8080"+ "/axis/HelloWorld.jws";//指明服务所在位置

Service service = new Service(); //创建一个Service实例，注意是必须的！

Call call = (Call) service.createCall();//创建Call实例，也是必须的！

call.setTargetEndpointAddress( new java.net.URL(endpoint) );//为Call设置服务的位置

call.setOperationName( "sayHello" );//注意方法名与HelloWorld.java中一样！！

String res = (String) call.invoke( new Object[] {} );//返回String，没有传入参数

System.out.println( res );

}

}

我的测试是在jbuilder2005中，注意项目中要导入其自带的AXIS包（当然应该把其中JAR文件替换一下），可以看到程序返回了 "HELLO WORLD!"

可以看到在AXIS里发布服务其实是一件很容易的事，这是因为这个服务很简单的原因：）下面我们介绍第二种发布方式，这是常用的。

**（2） 我们的第二种发布方式：**

@ 将HelloWorld.java编译成HelloWorld.class,

放到%TOMCAT\_HOME%\webapps\axis\WEB-INF\classes下

@ 在%TOMCAT\_HOME%\webapps\axis\WEB-INF下新建deploy.wsdd文件，即SOAP服务发布描述文件

deploy.wsdd

<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="" target="\_blank">http://xml.apache.org/axis/wsdd/providers/java">

<service name="HelloWorld" provider="java:RPC">

<parameter name="className" value="HelloWorld"/>

<parameter name="allowedMethods" value="sayHello"/>

</service>

</deployment>

在DOS下转换目录到%TOMCAT\_HOME%\webapps\axis\WEB-INF，命令：

java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient deploy.wsdd

你会发现目录下多了一个server-config.wsdd文件，这就是AXIS的配置文件，以后所有的服务发布描述都会在里面找到。（当然，你可以直接修改它，不用再写deploy.wsdd）然后打开浏览器http://localhost:8080/axis/servlet/AxisServlet，你就会看到你的服务已发布

注意：可以直接写server－config.wsdd,如下：

<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<handler type="java:org.apache.axis.handlers.http.URLMapper" name="URLMapper"/>

<service name="myService" provider="java:RPC">

<parameter name="className" value="com.service.myService"/>

<parameter name="allowedMethods" value="getusername"/>

</service>

<transport name="http">

<requestFlow>

<handler type="URLMapper"/>

</requestFlow>

</transport>

</deployment>

同样用客户端程序访问一下：（注意和上边的差别！！）

HelloClient.java

import org.apache.axis.client.Call;

import org.apache.axis.client.Service;

public class HelloClient

{

public static void main(String [] args) throws Exception {

String endpoint = "http://localhost:" +"8080"+ "/axis/services/HelloWorld";//注意！差别仅仅在这里！！

Service service = new Service();

Call call = (Call) service.createCall();

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

call.setOperationName("sayHello" );

String res = (String) call.invoke( new Object[] {} );

System.out.println( res );

}

}

好了，相信你对AXIS已有了大致的了解。接下来将会涉及到传参数、JAVABEAN对象，及AXIS的安全问题，下次再说吧：）也欢迎和我,一个快乐的JAVA程序员，联系:)ronghao100@hotmail.com

**（3） 第三种方式：**

@ 编写服务端程序server，SayHello.java，编译server.SayHello.java

package server;

public class SayHello

{

public String getName(String name)

{

return "hello "+name;

}

}

@ 编写wsdd文件

deploy.wsdd文件内容如下：

<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<service provider="java:RPC">

<parameter value="server.SayHello.getName"/>

<parameter value="\*"/>

</service>

</deployment>

@ 发布服务：

编辑一个deploy.bat，Axis\_Lib为axis.jar路径。内容如下：

set Axis\_Lib=D:\workspace\test\WEB-INF\lib

set Java\_Cmd=java -Djava.ext.dirs=%Axis\_Lib%

set Axis\_Servlet=http://localhost:8080/test/servlet/AxisServlet

%Java\_Cmd% org.apache.axis.client.AdminClient -l%Axis\_Servlet% deploy.wsdd

执行这个批处理文件，这时候，如果提示成功的话，访问http://localhost:8080/test/services 就会显示服务列表。

@ 生成客户端client stub文件

在浏览器上访问服务器端的服务，可以下载到WSDL文件，通过Axis的相关工具，可以自动从WSDL文件中生成Web Service的客户端代码。

编写一个WSDL2Java.bat文件，其内容如下：

set Axis\_Lib=D:\workspace\test\WEB-INF\lib

set Java\_Cmd=java -Djava.ext.dirs=%Axis\_Lib%

set Output\_Path=D:\workspace\test\src

set Package=server.SayHello

%Java\_Cmd% org.apache.axis.wsdl.WSDL2Java -o%Output\_Path% -p%Package% SayHello.wsdl

执行这个批处理文件就可以生成client stub.

生成的stub client文件列表为：SayHello.java，SayHelloService.java，SayHelloServiceLocator.java，SayHelloSoapBindingStub.java .

@ 编写客户端程序，编译并执行

下面是一段junit测试客户端代码。

import java.net.URL;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

public class TestWSClient extends TestCase {

public TestWSClient(String string) {

super(string);

}

public void SayHelloClient() throws Exception {

SayHelloService service = new SayHelloServiceLocator();

SayHello\_PortType client = service.getSayHello() ;

String retValue = client.getName("clientname");

System.out.println(retValue);

}

public static Test suite() {

TestSuite suite = new TestSuite();

suite.addTest(new TestWSClient("SayHelloClient"));

return suite;

}

}

## 三、发布Web服务使用Handler来增强Web服务的功能

**1 Handler的基本概念**

J2EE Web 服务中的Handler技术特点非常像Servlet技术中的Filter。我们知道，在Servlet中，当一个HTTP到达服务端时，往往要经过多个Filter对请求进行过滤，然后才到达提供服务的Servlet，这些Filter的功能往往是对请求进行统一编码，对用户进行认证，把用户的访问写入系统日志等。相应的，Web服务中的Handler通常也提供一下的功能

**2 使用场景**

（1）对客户端进行认证、授权；

（2）把用户的访问写入系统日志；

（3）对请求的SOAP消息进行加密，解密；

（4）为Web Services对象做缓存。

（5）SOAP消息Handler能够访问代表RPC请求或者响应的SOAP消息。在JAX-RPC技术中，SOAP消息Handler可以部署在服务端，也可以在客户端使用。

**3 典型的SOAP消息Handler处理顺序**

某个在线支付服务需要防止非授权的用户访问或者撰改服务端和客户端传输的信息，从而使用消息摘要（Message Digest）的方法对请求和响应的SOAP消息进行加密。当客户端发送SOAP消⑹保?突Ф说?andler把请求消息中的某些敏感的信息（如信用卡密码）进行加密，然后把加密后的SOAP消息传输到服务端；服务端的SOAP消息Handler截取客户端的请求，把请求的SOAP 消息进行解密，然后把解密后的SOAP消息派发到目标的Web服务端点。 Apache axis是我们当前开发Web服务的较好的选择，使用axisWeb服务开发工具，可以使用Handler来对服务端的请求和响应进行处理。典型的情况下，请求传递如图1所示。

**4 例子**

（1） 在axis环境下，SOAP消息Handler必须实现org.apache.axis.Handler接口（在JAX-RPC 1.0规范中，SOAP消息Handler必须实现javax.xml.rpc.handler.Handler接口），org.apache.axis.Handler接口的部分代码如下： 为了提供开发的方便，在编写Handler时，只要继承org.apache.axis.handlers. BasicHandler即可，BasicHandler是Handler的一个模板，我们看它的部分代码： BasicHandler中的(MessageContext msgContext)方法是Handler实现类必须实现的方法，它通过MessageContext来获得请求或者响应的SOAPMessage对象，然后对SOAPMessage进行处理。

（2）在介绍Handler的开发之前，我们先来看一下目标Web服务的端点实现类的代码，如例程3所示。

例程3 目标Web服务的端点实现类

package com.hellking.webservice;

public class HandleredService

{

//一个简单的Web服务

public String publicMethod(String name)

{

return "Hello!"+name;

}

}

//另一个Web服务端点：

package com.hellking.webservice;

public class OrderService

{

//web服务方法：获得客户端的订单信息，并且对订单信息进行对应的处理，

通常情况是把订单的信息写入数据库，然后可客户端返回确认信息。

public String orderProduct(String name,String address,String item,int quantity,Card card)

{

String cardId=card.getCardId();

String cardType=card.getCardType();

String password=card.getPassword();

String rderInfo="name="+name+",address="+address+",item="+item+",quantity="+quantity+"

,cardId="+cardId+",cardType="+cardType+",password="+password;

System.out.println("这里是客户端发送来的信息:");

System.out.println(orderInfo);

return orderInfo;

}

}

**5 使用Handler为系统做日志**

（1） Handler为系统做日志是一种比较常见而且简单的使用方式。和Servlet中的Filter一样，我们可以使用Handler来把用户的访问写入系统日志。下面我们来看日志Handler的具体代码，如例程4所示。

package com.hellking.webservice;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import java.util.Date;

import org.apache.axis.AxisFault;

import org.apache.axis.Handler;

import org.apache.axis.MessageContext;

import org.apache.axis.handlers.BasicHandler;

public class LogHandler extends BasicHandler {

public void invoke(MessageContext msgContext) throws AxisFault

{

try {

Handler handler = msgContext.getService();

String filename = (String)getOption("filename");

if ((filename == null) || (filename.equals("")))

throw new AxisFault("Server.NoLogFile",

"No log file configured for the LogHandler!",

null, null);

FileOutputStream fos = new FileOutputStream(filename, true);

PrintWriter writer = new PrintWriter(fos);

Integer counter = (Integer)handler.getOption("accesses");

if (counter == null)

counter = new Integer(0);

counter = new Integer(counter.intValue() + 1);

Date date = new Date();

msgContext.getMessage().writeTo(System.out);

String result = "在"+date + ": Web 服务 " +

msgContext.getTargetService() +

" 被调用，现在已经共调用了 " + counter + " 次.";

handler.setOption("accesses", counter);

writer.println(result);

writer.close();

} catch (Exception e) {

throw AxisFault.makeFault(e);

}

}

}

（2） 前面我们说过，Handler实现类必须实现invoke方法，invoke方法是Handler处理其业务的入口点。LogHandler的主要功能是把客户端访问的Web服务的名称和访问时间、访问的次数记录到一个日志文件中。 下面部署这个前面开发的Web服务对像，然后为Web服务指定Handler。编辑Axis\_Home/WEB-INF/ server-config.wsdd文件，在其中加入以下的内容：

<service name="HandleredService" provider="java:RPC">

<parameter name="allowedMethods" value="\*"/>

<parameter name="className" value="com.hellking.webservice.HandleredService"/>

<parameter name="allowedRoles" value="chen"/>

<beanMapping languageSpecificType="java:com.hellking.webservice.Card"

qname="card:card" xmlns:card="card"/>

<requestFlow>

<handler name="logging" type="java:com.hellking.webservice.LogHandler">

<parameter name="filename" value="c:\\MyService.log"/>

</handler>

</requestFlow>

</service>

…

</globalConfiguration>

…

<handler name="logging" type="java:com.hellking.webservice.LogHandler">

<parameter name="filename" value="c:\\MyService.log"/>

</handler>

…

<service name="HandleredService" provider="java:RPC">

…

<requestFlow>

<handler type="logging"/>

…<!--在这里可以指定多个Handler-->

</requestFlow>

</service>

**http://127.0.0.1:8080/handler/services/HandleredService?wsdl&method=publicMethod&name=chen**

注意：这个URL需要根据具体情况改变。

在Sun Jul 06 22:42:03 CST 2003: Web 服务 HandleredService 被调用，现在已经共调用了 1 次.

在Sun Jul 06 22:42:06 CST 2003: Web 服务 HandleredService 被调用，现在已经共调用了 2 次.

在Sun Jul 06 22:42:13 CST 2003: Web 服务 HandleredService 被调用，现在已经共调用了 3 次.

## 四、建立安全的AXIS服务

在前面的文章中，我们实现了最简单的AXIS服务。现在我们一起来讨论一下Web服务的安全问题。根据应用的对安全要求的级别不同，可以采用不同的方式来实现安全性，以下是目前最常用的一些实现方式（从低到高排列）

1、J2EE Web应用默认的访问控制（数据是明文的）；

2、使用axis的Handler进行访问控制（数据是明文的）；

3、使用Servlet过滤器（Filter）进行访问控制（数据是明文的）；

4、使用SSL/HTTPS协议来传输（加密的数据传输协议）；

5、使用WS-Security规范对信息进行加密与身份认证（数据被加密传输）。

我们仅讨论第2、4、5种实现方式。在此之前我们先来了解一下AXIS自带的一个工具SOAPMonitor。打开http://localhost:8080/axis/进入AXIS的主页面，你会看见：

SOAPMonitor-[disabled by default for security reasons] ，默认状态下其是不可用的，现在我们就来激活它。

**1、到目录%TOMCAT\_HOME%\webapps\axis下，你会找到SOAPMonitorApplet.java，在命令行中编译它：**

javac -classpath %AXIS\_HOME%\lib\axis.jar SOAPMonitorApplet.java

编译完之后你会看见目录下多了很多CLASS文件，它们的名字是SOAPMonitorApplet\*.class

**2、在目录%TOMCAT\_HOME%\webapps\axis\WEB-INF下打开server-config.wsdd文件，将下面的两部分代码直接加入其中相应的位置**

第一部分：

<handler name="soapmonitor" type="java:org.apache.axis.handlers.SOAPMonitorHandler">

<parameter name="wsdlURL" value="/axis/SOAPMonitorService-impl.wsdl"/>

<parameter name="namespace" value="" target="\_blank">http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>

<parameter name="serviceName" value="SOAPMonitorService"/>

<parameter name="portName" value="Demo"/>

</handler>

第二部分：

<service name="SOAPMonitorService" provider="java:RPC">

<parameter name="allowedMethods" value="publishMessage"/>

<parameter name="className" value="org.apache.axis.monitor.SOAPMonitorService"/>

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

</service>

**3、选择你要监控的服务**

以上次的HelloWorld服务为例，在server-config.wsdd中你会找到这段代码

<service name="HelloWorld" provider="java:RPC">

<parameter name="allowedMethods" value="sayHello"/>

<parameter name="className" value="HelloWorld"/>

</service>

在这段代码中加入以下的代码：

<requestFlow>

<handler type="soapmonitor"/>

</requestFlow>

<responseFlow>

<handler type="soapmonitor"/>

</responseFlow>

最后的样子是：

<service name="HelloWorld" provider="java:RPC">

<requestFlow>

<handler type="soapmonitor"/>

</requestFlow>

<responseFlow>

<handler type="soapmonitor"/>

</responseFlow>

<parameter name="allowedMethods" value="sayHello"/>

<parameter name="className" value="HelloWorld"/>

</service>

这样HelloWorld服务就被监控了

**4、启动Tomcat,打开http://localhost:8080/axis/SOAPMonitor，你就会看到Applet界面**，在 jbuilder2005中运行我们上次写的客户端程序 TestClient.java。OK！你会在Applet界面看见客户端与服务器端互发的XML内容，注意这里是明文！

## 五、使用SSL/HTTPS协议来传输

Web服务也可以使用SSL作为传输协议。虽然JAX-RPC并没有强制规定是否使用SSL协议，但在tomcat下使用HTTPS协议。

1. 使用JDK自带的工具创建密匙库和信任库。

**1）通过使用以下的命令来创建服务器端的密匙库：**

**keytool -genkey -alias Server -keystore server.keystore -keyalg RSA**

输入keystore密码： changeit

您的名字与姓氏是什么？

[Unknown]： Server

您的组织单位名称是什么？

[Unknown]： ec

您的组织名称是什么？

[Unknown]： ec

您所在的城市或区域名称是什么？

[Unknown]： beijing

您所在的州或省份名称是什么？

[Unknown]： beijing

该单位的两字母国家代码是什么

[Unknown]： CN

CN=Server, OU=ec, O=ec, L=beijing, ST=beijing, C=CN 正确吗？

[否]： y

输入<Server>的主密码（如果和 keystore 密码相同，按回车）：

以上命令执行完成后，将获得一个名为server.keystore的密匙库。

**2)生成客户端的信任库。首先输出RSA证书：**

keytool -export -alias Server -file test\_axis.cer -storepass changeit -keystore server.keystore

然后把RSA证书输入到一个新的信任库文件中。这个信任库被客户端使用，被用来验证服务器端的身份。

keytool -import -file test\_axis.cer -storepass changeit -keystore client.truststore -alias serverkey -noprompt

以上命令执行完成后，将获得一个名为client.truststore的信任库。

**3）同理生成客户端的密匙库client.keystore和服务器端的信任库server.truststore.方便起见给出.bat文件**

gen-cer-store.bat内容如下：

set SERVER\_DN="CN=Server, OU=ec, O=ec, L=BEIJINGC, S=BEIJING, C=CN"

set CLIENT\_DN="CN=Client, OU=ec, O=ec, L=BEIJING, S=BEIJING, C=CN"

set KS\_PASS=-storepass changeit

set KEYINFO=-keyalg RSA

keytool -genkey -alias Server -dname %SERVER\_DN% %KS\_PASS% -keystore server.keystore %KEYINFO% -keypass changeit

keytool -export -alias Server -file test\_axis.cer %KS\_PASS% -keystore server.keystore

keytool -import -file test\_axis.cer %KS\_PASS% -keystore client.truststore -alias serverkey -noprompt

keytool -genkey -alias Client -dname %CLIENT\_DN% %KS\_PASS% -keystore client.keystore %KEYINFO% -keypass changeit

keytool -export -alias Client -file test\_axis.cer %KS\_PASS% -keystore client.keystore

keytool -import -file test\_axis.cer %KS\_PASS% -keystore server.truststore -alias clientkey -noprompt

好的，现在我们就有了四个文件：server.keystore，server.truststore，client.keystore，client.truststore

**2、更改Tomcat的配置文件（server.xml），增加以下部署描述符：（其实里面有，只是被注释掉了）**

<Connector port="8440"

maxThreads="150" minSpareThreads="25" maxSpareThreads="75"

enableLookups="false" disableUploadTimeout="true"

acceptCount="100" scheme="https" secure="true"

clientAuth="true" keystoreFile="f:\server.keystore" keystorePass="changeit"

truststoreFile="f:\server.truststore" truststorePass="changeit"

sslProtocol="TLS" />

**3、把HelloWorld重新部署一次，在server-config.wsdd中修改如下部署代码。（还原了而已）**

<service name="HelloWorld" provider="java:RPC">

<requestFlow>

<handler type="soapmonitor"/>

</requestFlow>

<responseFlow>

<handler type="soapmonitor"/>

</responseFlow>

<parameter name="allowedMethods" value="sayHello"/>

<parameter name="className" value="HelloWorld"/>

</service>

**4、修改客户端程序 TestClient.java（修改的部分已标出）**

public class TestClient

{

public static void main(String [] args) throws Exception {

String endpoint = "https://localhost:" +"8440"+ "/axis/HelloWorld";//注意区别在这里！https！

Service service = new Service();

Call call = (Call) service.createCall();

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

call.setOperationName( "sayHello" );

String res = (String) call.invoke( new Object[] {} );

System.out.println( res );

}

}

**5、最后使用命令来执行客户端程序**

java -cp %AXISCLASSPATH%

-Djavax.net.ssl.keyStore=client.keystore

-Djavax.net.ssl.keyStorePassword=changeit

-Djavax.net.ssl.trustStore=client.truststore

TestClient

## 六、使用WS-Security规范对信息进行加密与身份认证

我们打算用Handler结合WSSecurity实现Web服务安全

WSClientRequestHandler.java位于客户端对客户端发出的XML文档进行加密

WSServerRequestHandler.java位于服务器端对客户端发出的加密后的XML文档进行解密

WSServerResponseHandler.java位于服务器端对服务器端返回的XML文档进行加密

WSClientResponseHandler.java位于客户端对服务器端返回的XML文档进行解密

**1、使用ISNetworks安全提供者，ISNetworks实现了RSA加密、解密算法。**

当然，你也可以使用其它的安全提供者，并且可以使用不同的加密算法。

ISNetworks相关包ISNetworksProvider.jar。拷贝到%TOMCAT\_HOME% \webapps\axis\WEB-INF\lib

**2、Trust Services Integration Kit提供了一个WS-Security实现**。你可以从http://www.xmltrustcenter.org获得相关库文件，分别是ws-security.jar和tsik.jar。ws-security.jar中包含一个WSSecurity类，我们使用它来对XML进行数字签名和验证，加密与解密。同样拷贝到%TOMCAT\_HOME%\webapps\axis\WEB-INF\lib

**3、创建密匙库和信任库。（见上文，一模一样！）**

**4、框架结构**

WSClientHandler.java //基类，包含了一些公用方法

WSClientRequestHandler.java //继承于WSClientHandler.java，调用WSHelper.java对客户端发出的XML文档进行加密

WSClientResponseHandler.java //继承于WSClientHandler.java，调用WSHelper.java对服务器端返回的XML文档进行解密

WSServerHandler.java //基类，包含了一些公用方法

WSServerRequestHandler.java //继承于WSServerHandler.java，调用WSHelper.java对客户端发出的加密后的XML文档进行解密

WSServerResponseHandler.java//继承于WSServerHandler.java，调用WSHelper.java对服务器端返回的XML文档进行加密

WSHelper.java //核心类，对SOAP消息签名、加密、解密、身份验证

MessageConverter.java //帮助类，Document、SOAP消息互相转换

**5、具体分析（在此强烈建议看一下tsik.jar的API）**

WSHelper.java

public class WSHelper {

static String PROVIDER="ISNetworks";//JSSE安全提供者。

//添加JSSE安全提供者，你也可以使用其它安全提供者。只要支持DESede算法。这是程序里动态加载还可以在JDK中静态加载

static

{

java.security.Security.addProvider(new com.isnetworks.provider.jce.ISNetworksProvider());

}

/\*\*

\*对XML文档进行数字签名。

\*/

public static void sign(Document doc, String keystore, String storetype,

String storepass, String alias, String keypass) throws Exception {

FileInputStream fileInputStream = new FileInputStream(keystore);

java.security.KeyStore keyStore = java.security.KeyStore.getInstance(storetype);

keyStore.load(fileInputStream, storepass.toCharArray());

PrivateKey key = (PrivateKey)keyStore.getKey(alias, keypass.toCharArray());

X509Certificate cert = (X509Certificate)keyStore.getCertificate(alias);

SigningKey sk = SigningKeyFactory.makeSigningKey(key);

KeyInfo ki = new KeyInfo();

ki.setCertificate(cert);

WSSecurity wSSecurity = new WSSecurity();//ws-security.jar中包含的WSSecurity类

wSSecurity.sign(doc, sk, ki);//签名。

}

/\*\*

\*对XML文档进行身份验证。

\*/

public static boolean verify(Document doc, String keystore, String storetype,

String storepass) throws Exception {

FileInputStream fileInputStream = new FileInputStream(keystore);

java.security.KeyStore keyStore = java.security.KeyStore.getInstance(storetype);

keyStore.load(fileInputStream, storepass.toCharArray());

TrustVerifier verifier = new X509TrustVerifier(keyStore);

WSSecurity wSSecurity = new WSSecurity();

MessageValidity[] resa = wSSecurity.verify(doc, verifier, null,null);

if (resa.length > 0)

return resa[0].isValid();

return false;

}

/\*\*

\*对XML文档进行加密。必须有JSSE提供者才能加密。

\*/

public static void encrypt(Document doc, String keystore, String storetype,

String storepass, String alias) throws Exception {

try

{

FileInputStream fileInputStream = new FileInputStream(keystore);

java.security.KeyStore keyStore = java.security.KeyStore.getInstance(storetype);

keyStore.load(fileInputStream, storepass.toCharArray());

X509Certificate cert = (X509Certificate)keyStore.getCertificate(alias);

PublicKey pubk = cert.getPublicKey();

KeyGenerator keyGenerator = KeyGenerator.getInstance("DESede",PROVIDER);

keyGenerator.init(168, new SecureRandom());

SecretKey key = keyGenerator.generateKey();

KeyInfo ki = new KeyInfo();

ki.setCertificate(cert);

WSSecurity wSSecurity = new WSSecurity();

//加密。

wSSecurity.encrypt(doc, key, AlgorithmType.TRIPLEDES, pubk, AlgorithmType.RSA1\_5, ki);

}

catch(Exception e)

{

e.printStackTrace();

}

}

/\*\*

\*对文档进行解密。

\*/

public static void decrypt(Document doc, String keystore, String storetype,

String storepass, String alias, String keypass) throws Exception {

FileInputStream fileInputStream = new FileInputStream(keystore);

java.security.KeyStore keyStore = java.security.KeyStore.getInstance(storetype);

keyStore.load(fileInputStream, storepass.toCharArray());

PrivateKey prvk2 = (PrivateKey)keyStore.getKey(alias, keypass.toCharArray());

WSSecurity wSSecurity = new WSSecurity();

//解密。

wSSecurity.decrypt(doc, prvk2, null);

WsUtils.removeEncryptedKey(doc);//从 WS-Security Header中删除 EncryptedKey 元素

}

public static void removeWSSElements(Document doc) throws Exception {

WsUtils.removeWSSElements(doc);// 删除WSS相关的元素。

}

}

WSClientHandler.java

//继承自org.apache.axis.handlers.BasicHandler即AXIS内在的

public class WSClientHandler extends BasicHandler{

protected String keyStoreFile ;

protected String keyStoreType ="JKS";//默认

protected String keyStorePassword ;

protected String keyAlias ;

protected String keyEntryPassword ;

protected String trustStoreFile ;

protected String trustStoreType = "JKS";//默认

protected String trustStorePassword ;

protected String certAlias ;

public void setInitialization(String keyStoreFile,String keyStoreType,String keyStorePassword,

String keyAlias,String keyEntryPassword,String trustStoreFile,

String trustStoreType,String trustStorePassword,String certAlias){

this.keyStoreFile=keyStoreFile;

this.keyStoreType=keyStoreType;

this.keyStorePassword=keyStorePassword;

this.keyAlias=keyAlias;

this.keyEntryPassword=keyEntryPassword;

this.trustStoreFile=trustStoreFile;

this.trustStoreType=trustStoreType;

this.trustStorePassword=trustStorePassword;

this.certAlias=certAlias;

}

public void setInitialization(String keyStoreFile,String keyStorePassword,

String keyAlias,String keyEntryPassword,String trustStoreFile,

String trustStorePassword,String certAlias){

this.keyStoreFile=keyStoreFile;

this.keyStorePassword=keyStorePassword;

this.keyAlias=keyAlias;

this.keyEntryPassword=keyEntryPassword;

this.trustStoreFile=trustStoreFile;

this.trustStorePassword=trustStorePassword;

this.certAlias=certAlias;

}

public void invoke(MessageContext messageContext) throws AxisFault {//在这个方法里对XML文档进行处理

//do nothing now!

}

public void onFault(MessageContext msgContext) {

System.out.println("处理错误，这里忽略！");

}

}

WSClientRequestHandler.java

public class WSClientRequestHandler extends WSClientHandler{

public void invoke(MessageContext messageContext) throws AxisFault {

try {

SOAPMessage soapMessage = messageContext.getMessage();

Document doc = MessageConverter.convertSoapMessageToDocument(soapMessage); //soapMessage转换为Document

WSHelper.sign(doc, keyStoreFile, keyStoreType,keyStorePassword, keyAlias, keyEntryPassword); //数字签名

WSHelper.encrypt(doc, trustStoreFile, trustStoreType, trustStorePassword, certAlias); //加密

soapMessage = MessageConverter.convertDocumentToSOAPMessage(doc);

//处理后的Document再转换回soapMessage

messageContext.setMessage(soapMessage);

} catch (Exception e){

System.err.println("在处理响应时发生以下错误： " + e);

e.printStackTrace(); }

}

}

WSClientResponseHandler.java

public class WSClientResponseHandler extends WSClientHandler{

public void invoke(MessageContext messageContext) throws AxisFault {

try {

SOAPMessage soapMessage = messageContext.getCurrentMessage();

Document doc = MessageConverter.convertSoapMessageToDocument(soapMessage);

WSHelper.decrypt(doc, keyStoreFile, keyStoreType,

keyStorePassword, keyAlias, keyEntryPassword);//解密

WSHelper.verify(doc, trustStoreFile, trustStoreType, trustStorePassword);//验证

WSHelper.removeWSSElements(doc);

soapMessage = MessageConverter.convertDocumentToSOAPMessage(doc);

messageContext.setMessage(soapMessage);

} catch (Exception e){

e.printStackTrace();

System.err.println("在处理响应时发生以下错误： " + e);

}

}

}

WSServerHandler.java

public class WSServerHandler extends BasicHandler{

protected String keyStoreFile ;

protected String keyStoreType ="JKS";//默认

protected String keyStorePassword ;

protected String keyAlias ;

protected String keyEntryPassword ;

protected String trustStoreFile ;

protected String trustStoreType = "JKS";//默认

protected String trustStorePassword ;

protected String certAlias ;

public void invoke(MessageContext messageContext) throws AxisFault {

//do nothing now!

}

public void onFault(MessageContext msgContext) {

System.out.println("处理错误，这里忽略！");

}

public void init() { //初始化，从配置文件server-config.wsdd中读取属性

keyStoreFile = (String)getOption("keyStoreFile");

if(( keyStoreFile== null) )

System.err.println("Please keyStoreFile configured for the Handler!");

trustStoreFile = (String)getOption("trustStoreFile");

if(( trustStoreFile== null) )

System.err.println("Please trustStoreFile configured for the Handler!");

keyStorePassword = (String)getOption("keyStorePassword");

if(( keyStorePassword== null) )

System.err.println("Please keyStorePassword configured for the Handler!");

keyAlias = (String)getOption("keyAlias");

if(( keyAlias== null) )

System.err.println("Please keyAlias configured for the Handler!");

keyEntryPassword = (String)getOption("keyEntryPassword");

if(( keyEntryPassword== null) )

System.err.println("Please keyEntryPassword configured for the Handler!");

trustStorePassword = (String)getOption("trustStorePassword");

if(( trustStorePassword== null) )

System.err.println("Please trustStorePassword configured for the Handler!");

certAlias = (String)getOption("certAlias");

if ((certAlias==null))

System.err.println("Please certAlias configured for the Handler!");

if ((getOption("keyStoreType")) != null)

keyStoreType = (String)getOption("keyStoreType");

if ((getOption("trustStoreType")) != null)

trustStoreType = (String)getOption("trustStoreType");

}

}

WSServerRequestHandler.java

public class WSServerRequestHandler extends WSServerHandler{

public void invoke(MessageContext messageContext) throws AxisFault {

try {

SOAPMessage msg = messageContext.getCurrentMessage();

Document doc = MessageConverter.convertSoapMessageToDocument(msg);

System.out.println("接收的原始消息:");

msg.writeTo(System.out);

WSHelper.decrypt(doc, keyStoreFile, keyStoreType,

keyStorePassword, keyAlias, keyEntryPassword);//解密

WSHelper.verify(doc, trustStoreFile, trustStoreType, trustStorePassword);//验证

WSHelper.removeWSSElements(doc);

msg = MessageConverter.convertDocumentToSOAPMessage(doc);

System.out.println("怀原后的原始消息:");

msg.writeTo(System.out);

messageContext.setMessage(msg);

} catch (Exception e){

e.printStackTrace();

System.err.println("在处理响应时发生以下错误： " + e);

}

}

}

WSServerResponseHandler.java

public class WSServerResponseHandler extends WSServerHandler{

public void invoke(MessageContext messageContext) throws AxisFault {

try {

SOAPMessage soapMessage = messageContext.getMessage();

System.out.println("返回的原始消息:");

soapMessage.writeTo(System.out);

Document doc = MessageConverter.convertSoapMessageToDocument(soapMessage);

WSHelper.sign(doc, keyStoreFile, keyStoreType,

keyStorePassword, keyAlias, keyEntryPassword);//数字签名

WSHelper.encrypt(doc, trustStoreFile, trustStoreType,//加密

trustStorePassword, certAlias);

soapMessage = MessageConverter.convertDocumentToSOAPMessage(doc);

System.out.println("返回的加密后的消息:");

soapMessage.writeTo(System.out);

messageContext.setMessage(soapMessage);

} catch (Exception e){

System.err.println("在处理响应时发生以下错误： " + e);

e.printStackTrace();

}

}

}

**6、应用**

为方便使用，把上述文件打包为ws-axis.jar,

放入%TOMCAT\_HOME%\webapps\axis\WEB-INF\lib

1）把HelloWorld重新部署一次，在server-config.wsdd中修改如下部署代码。

<service name="HelloWorld" provider="java:RPC">

<parameter name="allowedMethods" value="\*"/>

<parameter name="className" value="HelloWorld"/>

<requestFlow>

<handler type="soapmonitor"/>

<handler type="java:com.ronghao.WSAxis.WSServerRequestHandler">

<parameter name="keyStoreFile" value="f:\server.keystore"/>

<parameter name="trustStoreFile" value="f:\server.truststore"/>

<parameter name="keyStorePassword" value="changeit"/>

<parameter name="keyAlias" value="Server"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="certAlias" value="clientkey"/>

</handler>

</requestFlow>

<responseFlow>

<handler type="soapmonitor"/>

<handler type="java:com.ronghao.WSAxis.WSServerResponseHandler">

<parameter name="keyStoreFile" value="f:\server.keystore"/>

<parameter name="trustStoreFile" value="f:\server.truststore"/>

<parameter name="keyStorePassword" value="changeit"/>

<parameter name="keyAlias" value="Server"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="certAlias" value="clientkey"/>

</handler>

</responseFlow>

</service>

2)修改客户端程序 TestClient.java（修改的部分已标出，记着导入ws-axis.jar）

import javax.xml.namespace.QName;

import org.apache.axis.client.Call;

import org.apache.axis.client.Service;

import com.ronghao.WSAxis.\*;

public class WSSClient1

{

public static void main(String [] args)

{

try {

//服务端的url，需要根据情况更改。

String endpointURL = "http://localhost:8080/axis/services/HelloWorld";

Service svc = new Service();

WSClientHandler handler=new WSClientRequestHandler();

//注意新加的HANDLER

handler.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

WSClientHandler handlee=new WSClientResponseHandler();

//注意新加的HANDLER

handlee.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

Call call =(Call)svc.createCall();

call.setClientHandlers(handler,handlee);//添加Handler

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

call.setOperationName(new QName("sayHello"));

String result = (String) call.invoke( new Object [] {});

System.out.println("the result"+result);

} catch (Exception e) {

e.printStackTrace();

}

}

}

运行的时候http://localhost:8080/axis/SOAPMonitor中看到的请求的XML就已加密！总结这里对代码的解释是不够的，很多概念没有提到。建议你最好看tsik.jar和AXIS的API深入了解。另外对ws-axis.jar的加解密实现打算运用apache的wss4j,相关网址http://ws.apache.org/ws-fx/wss4j/。不过这个东西也应该够用了暂时。所有的源文件在附件中附件：soapTest.zip(279K)

## 七、在AXIS服务间传递JavaBean及其安全解决

客户端并没有向服务器端递参数，现在我们来传传JavaBean。当然，也可以传递你自己定义的JAVA类，但那样你必须自己创建专门的XML序列化器和反序列化器；而对JavaBean，AXIS提供了现成的序列化器。

1. **服务器端CLASS类两个Order.class,OrderTest.class**

位于%TOMCAT\_HOME%\webapps\axis\WEB-INF\classes下，这两个类都直接给出源码

Order.java

public class Order {

private String id;

private String name;

public void setId(String id){

this.id=id;

}

public String getId(){

return id;

}

public void setName(String name){

this.name=name;

}

public String getName(){

return name;

}

}

OrderTest.java

public class OrderTest {

public Order returnOrder(Order order){

Order newOrder=new Order();

if(order.getId().equals("1"))

newOrder.setName("ronghao");

else newOrder.setName("haorong");

return newOrder;

}

}

**2、修改服务器端配置文件server-config.wsdd**

在server-config.wsdd中相应位置添加以下代码

<service name="Order" provider="java:RPC">

<parameter name="allowedMethods" value="returnOrder"/>

<parameter name="className" value="OrderTest"/>

<beanMapping languageSpecificType="java:Order" qname="ns1:Order"

xmlns:ns1="urn:BeanService"/>

</service>

可以看到和前面的发布服务代码相比仅多了一行代码

<beanMapping languageSpecificType="java:Order" qname="ns1:Order"

xmlns:ns1="urn:BeanService"/>

languageSpecificType属性指定JavaBean类文件位置,例如：

languageSpecificType="java:com.ronghao.axis.Order"

qname属性指定JavaBean类的名字其他是固定的。

**3 客户端类文件一个OrderClient.class，代码如下（变化的部分加注释）：**

public class OrderClient

{

public static void main(String args[])

throws Exception

{

String endpoint = "http://localhost:8080/axis/services/Order"; //服务所在位置

Order order=new Order(); //JavaBean

order.setId("1");

Service service = new Service();

Call call = (Call)service.createCall();

//注册JavaBean，注意和server-config.wsdd中的配置代码比较

QName qn = new QName("urn:BeanService", "Order");

call.registerTypeMapping(Order.class, qn, new BeanSerializerFactory(Order.class, qn),

new BeanDeserializerFactory(Order.class, qn));

String name="no!";

try

{

call.setTargetEndpointAddress(new URL(endpoint));

//调用的服务器端方法

call.setOperationName(new QName("Order", "returnOrder"));

//设定传入的参数，这里qn即Order.class

call.addParameter("arg1", qn, ParameterMode.IN);

//设定返回的参数是Order.class

call.setReturnType(qn, Order.class);

Order result = (Order)call.invoke(new Object[] {

order

});

if(result != null)

name = result.getName();

}

catch(Exception e)

{

System.err.println(e);

}

System.out.println(name);

}

}

OK！运行一下，就可以看到返回了"ronghao"。

**4 不容许在网络中传递XML是明文，于是需要加密和验证。**

这里我们继续采用上次所讲的框架。（已打包成ws-axis.jar)

（1）修改服务器端配置文件server-config.wsdd（和上一文章一模一样！不再罗嗦）

在server-config.wsdd中相应位置添加以下代码

<requestFlow>

<handler type="soapmonitor"/>

<handler type="java:com.ronghao.WSAxis.WSServerRequestHandler">

<parameter name="keyStoreFile" value="f:\server.keystore"/>

<parameter name="trustStoreFile" value="f:\server.truststore"/>

<parameter name="keyStorePassword" value="changeit"/>

<parameter name="keyAlias" value="Server"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="certAlias" value="clientkey"/>

</handler>

</requestFlow>

<responseFlow>

<handler type="soapmonitor"/>

<handler type="java:com.ronghao.WSAxis.WSServerResponseHandler">

<parameter name="keyStoreFile" value="f:\server.keystore"/>

<parameter name="trustStoreFile" value="f:\server.truststore"/>

<parameter name="keyStorePassword" value="changeit"/>

<parameter name="keyAlias" value="Server"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="certAlias" value="clientkey"/>

</handler>

</responseFlow>

（2）客户端，先在这里要说一下，客户端代码编写困扰了我很长一段时间（整整一天），因为它并不象我想象的那么简单，当然解决起来还是挺简单的：）问题的解决经历了三个阶段

**第一阶段：在这个阶段我想当然的在OrderClient.class中加入了如下代码：**

WSClientHandler handler=new WSClientRequestHandler();//注意新加的HANDLER

handler.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

WSClientHandler handlee=new WSClientResponseHandler();//注意新加的HANDLER

handlee.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

call.setClientHandlers(handler,handlee);//添加Handler

这个方法也是我在上一文章里介绍的，结果抛出以下异常：

faultString: org.xml.sax.SAXException: Deserializing parameter

&apos;newProfileReturn&apos;: could not find deserializer for type

{urn:BeanService Order}SerializableProfile

也就是说不能正常解析XML文件，于是理所当然的郁闷了，觉得代码中肯定漏设了CALL的一个属性，于是查看AXIS的源代码，没有结果！转机出现在下面一行代码，在不断的抛出异常中我修改了代码

将call.setClientHandlers(handler,handlee);改为

call.setClientHandlers(null,null);

结果程序还是抛出同样的异常，于是意识到这可能是AXIS的一个BUG，为证明这一点，我将下面的Handler初始化代码删除

WSClientHandler handler=new WSClientRequestHandler();//注意新加的HANDLER

handler.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

WSClientHandler handlee=new WSClientResponseHandler();//注意新加的HANDLER

handlee.setInitialization("f:/client.keystore","changeit","Client","changeit",

"f:/client.truststore","changeit","serverkey");//初始化

结果还是抛出同样的异常，果然是BUG！得到这个结论后去了apache AXIS主页，在问题列表中见到了完全一样问题的提交，但没有解答（晕！）

最后得到了结论：call的setClientHandlers()方法只有当call处理简单的数据类型，如String,int等等才能正常使用！

（当然，如果你对这个问题有不同的见解，欢迎和我联系。或许我错了，但程序不运行是真的：））

**第二阶段**

开始在google上找问题的解决方法，这也是我的习惯：）。找了一个类似问题的讨论，地址如下：

http://marc.theaimsgroup.com/?l=axis-user&m=111259980822735&w=2

他们的解决方法是Handler继承于javax.xml.rpc.handler.Handler，然后在程序里动态注册而在我的ws-axis.jar里Handler继承于org.apache.axis.handlers.BasicHandler。当然，

javax.xml.rpc.handler.Handler 是org.apache.axis.handlers.BasicHandler的老爸，但在程序里老爸和儿子之间却不能很好的兼容，这也许就是所谓的代沟？？无奈中重新写了Handler,但在运行中却抛出异常，提示message在被invoke的时候已被更改。我靠，Handler的作用就是来更改 message的啊！这是什么世道！

我知道很多程序采用的就是这种方法，但我好象怎么修改都抛出上述异常。

**第三阶段**

既然在程序里动态注册Handler行不通，于是决定写个单独的配置文件来注册Handler。如果这种方法不幸失败就返回第二阶段。好马为什么不吃回头草？？

1、ws-axis.jar中修改WSClientHandler.class,修改后如下，我想你一看就明白为何修改

public class WSClientHandler extends BasicHandler{

protected String keyStoreFile ;

protected String keyStoreType ="JKS";

protected String keyStorePassword ;

protected String keyAlias ;

protected String keyEntryPassword ;

protected String trustStoreFile ;

protected String trustStoreType = "JKS";

protected String trustStorePassword ;

protected String certAlias ;

public void init() {

keyStoreFile = (String)getOption("keyStoreFile");

if(( keyStoreFile== null) )

System.err.println("Please keyStoreFile configured for the Handler!");

trustStoreFile = (String)getOption("trustStoreFile");

if(( trustStoreFile== null) )

System.err.println("Please trustStoreFile configured for the Handler!");

keyStorePassword = (String)getOption("keyStorePassword");

if(( keyStorePassword== null) )

System.err.println("Please keyStorePassword configured for the Handler!");

keyAlias = (String)getOption("keyAlias");

if(( keyAlias== null) )

System.err.println("Please keyAlias configured for the Handler!");

keyEntryPassword = (String)getOption("keyEntryPassword");

if(( keyEntryPassword== null) )

System.err.println("Please keyEntryPassword configured for the Handler!");

trustStorePassword = (String)getOption("trustStorePassword");

if(( trustStorePassword== null) )

System.err.println("Please trustStorePassword configured for the Handler!");

certAlias = (String)getOption("certAlias");

if ((certAlias==null))

System.err.println("Please certAlias configured for the Handler!");

if ((getOption("keyStoreType")) != null)

keyStoreType = (String)getOption("keyStoreType");

if ((getOption("trustStoreType")) != null)

trustStoreType = (String)getOption("trustStoreType");

}

public void invoke(MessageContext messageContext) throws AxisFault {

//do nothing now!

}

public void onFault(MessageContext msgContext) {

System.out.println("处理错误，这里忽略！");

}

}

（3）写客户端的配置代码client-config.wsdd,如下：

<?xml version="1.0" encoding="UTF-8"?>

<deployment name="defaultClientConfig"

xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="" target="\_blank">http://xml.apache.org/axis/wsdd/providers/java">

<transport name="http"

pivot="java:org.apache.axis.transport.http.HTTPSender"/>

<transport name="local"

pivot="java:org.apache.axis.transport.local.LocalSender"/>

<transport name="java"

pivot="java:org.apache.axis.transport.java.JavaSender"/>

<globalConfiguration>

<requestFlow>

<handler type="java:com.ronghao.WSAxis.WSClientRequestHandler">

<parameter name="keyStoreFile" value="D:\Tomcat5.5\webapps\axis\WEB-INF\client.keystore"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="certAlias" value="serverkey"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="trustStoreFile" value="D:\Tomcat5.5\webapps\axis\WEB-INF\client.truststore"/>

<parameter name="keyAlias" value="Client"/>

<parameter name="keyStorePassword" value="changeit"/>

</handler>

</requestFlow>

<responseFlow>

<handler type="java:com.ronghao.WSAxis.WSClientResponseHandler">

<parameter name="keyStoreFile" value="D:\Tomcat5.5\webapps\axis\WEB-INF\client.keystore"/>

<parameter name="keyEntryPassword" value="changeit"/>

<parameter name="certAlias" value="serverkey"/>

<parameter name="trustStorePassword" value="changeit"/>

<parameter name="trustStoreFile" value="D:\Tomcat5.5\webapps\axis\WEB-INF\client.truststore"/>

<parameter name="keyAlias" value="Client"/>

<parameter name="keyStorePassword" value="changeit"/>

</handler>

</responseFlow>

</globalConfiguration>

</deployment>

同样不再解释，不明白可以参考我的上一篇文章

（4）修改OrderClient.class

在OrderClient.class中加入了如下代码：

EngineConfiguration conf =

new FileProvider("F:\\Tomcat\\webapps\\axis\\WEB-INF\\client-config.wsdd");//位置

Service service = new Service(conf);

当然记得导入

import org.apache.axis.EngineConfiguration;

import org.apache.axis.configuration.FileProvider;

运行一下，返回"ronghao"，靠，搞定！

注意：这次我把OrderClient.class的调用放到了一个JSP文件中而不是jbuilder中，因为有client-config.wsdd，所以你必须有完整的WEB程序发布到TOMCAT中，否则会报找不到相应文件。

**Xfire**

**1 基于xfire实现webservice的实例**

先下载xfire的jar包,并导入项目当中

下载地址：http://xfire.codehaus.org/Download

**2 编写实现类**

首先建一个接口把对外开放的方法添加进该接口，实际上就是开放此接口让客户端

package com.test.service;

public class HelloXfireImpl implements HelloXfire {

public String sayHello(String str) {

return str + " this is xfire";

}

}

**3 映射web.xml**

通过配置web.xml来映射相应的路径，在web.xml添加如下代码

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

version="2.5"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<servlet>

<servlet-name>XFireServlet</servlet-name>

<servlet-class>org.codehaus.xfire.transport.http.XFireConfigurableServlet</servlet-class>

<load-on-startup>0</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>XFireServlet</servlet-name>

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

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

**4 services配置文件**

创建services文件，在src目录下创建META-INF目录，在该目录下创建一个xfire目录在该目录下创建services.xml通过配置此文件来公开我们的接口，客户端可以通过此文件的映射可以找到相应的需要公开的接口

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://xfire.codehaus.org/config/1.0">

<service>

<name>HelloXfire</name>

<serviceClass>com.test.service.HelloXfire</serviceClass>

<implementationClass>

com.test.service.HelloXfireImpl

</implementationClass>

<mce:style><!-- wrapped -->

</mce:style>

<style mce\_bogus="1">wrapped</style>

<use>literal</use>

<scope>application</scope>

</service></beans>

**5 获取wsdl文件的地址**

什么是wsdl？

参考: http://blog.csdn.net/sunchaohuang/archive/2008/10/14/3076375.aspx

通过web.xml与services.xml对地址的映射我们可以获得如下地址并进行访问后就可以得到wsdl文件的url了

访问的地址：

http://localhost:9099/HelloXfire/services/HelloXfire?wsdl

访问后会得到一个类似xml格式的内容 该文件就是一个wsdl文件

**6 编写客户端**

客户端获得webservice的url地址后创建对外开放的接口的实体类，然后向相应的方法传递参数，经过服务端相应的处理返回需要的结果

package com.test.client;

import java.net.MalformedURLException;

import org.codehaus.xfire.XFireFactory;

import org.codehaus.xfire.client.XFireProxyFactory;

import org.codehaus.xfire.service.Service;

import org.codehaus.xfire.service.binding.ObjectServiceFactory;

import com.test.service.HelloXfire;

public class HelloXfireClient {

public static void main(String[] args) {

Service srvcModel = new ObjectServiceFactory().create(HelloXfire.class);

XFireProxyFactory factory = new XFireProxyFactory(XFireFactory

.newInstance().getXFire());

String helloWorldURL = "http://localhost:9099/HelloXfire/services/HelloXfire";

try {

HelloXfire srvc = (HelloXfire) factory.create(srvcModel,

helloWorldURL);

String result = srvc.sayHello("hello ");

System.out.print(result);

} catch (MalformedURLException e) {

e.printStackTrace();

}

}

}

**7 不同项目的webservice调用**

一般情况下webservice的调用都是在不同的项目间或者不同的语言中来使用的如果是不同的语言调用webservice应用：客户端如果是其他语言，则通过得到服务端生成的wsdl或者或者wsdl文件的地址做相应的操作得到本语言具体的客户端实现类来调用服务端对外开放的方法如果是不同项目间的调用（java语言为例）在myeclipse下新建一个web service Client选择xfire下一步后选择wsdl的

地址:http://localhost:9099/HelloXfire/services/HelloXfire?wsdl 完成后会自动生成相应的客户端类。

在HelloXfireClient类中调取服务端对外开放的接口方法获得相应数据

具体代码如下

package com.test.service;

import java.net.MalformedURLException;

import java.util.Collection;

import java.util.HashMap;

import javax.xml.namespace.QName;

import org.codehaus.xfire.XFireRuntimeException;

import org.codehaus.xfire.aegis.AegisBindingProvider;

import org.codehaus.xfire.annotations.AnnotationServiceFactory;

import org.codehaus.xfire.annotations.jsr181.Jsr181WebAnnotations;

import org.codehaus.xfire.client.XFireProxyFactory;

import org.codehaus.xfire.jaxb2.JaxbTypeRegistry;

import org.codehaus.xfire.service.Endpoint;

import org.codehaus.xfire.service.Service;

import org.codehaus.xfire.soap.AbstractSoapBinding;

import org.codehaus.xfire.transport.TransportManager;

public class HelloXfireClient {

private static XFireProxyFactory proxyFactory = new XFireProxyFactory();

private HashMap endpoints = new HashMap();

private Service service0;

public HelloXfireClient() {

create0();

Endpoint HelloXfirePortTypeLocalEndpointEP = service0 .addEndpoint(new QName("http://service.test.com", "HelloXfirePortTypeLocalEndpoint"), new QName("http://service.test.com", "HelloXfirePortTypeLocalBinding"), "xfire.local://HelloXfire");

endpoints.put(new QName("http://service.test.com", "HelloXfirePortTypeLocalEndpoint"), HelloXfirePortTypeLocalEndpointEP);

Endpoint HelloXfireHttpPortEP = service0 .addEndpoint(new QName("http://service.test.com", "HelloXfireHttpPort"), new QName("http://service.test.com", "HelloXfireHttpBinding"), "http://localhost:9099/HelloXfire/services/HelloXfire");

endpoints.put(new QName("http://service.test.com", "HelloXfireHttpPort"), HelloXfireHttpPortEP);

}

public Object getEndpoint(Endpoint endpoint) {

try {

return proxyFactory.create((endpoint).getBinding(), (endpoint).getUrl());

} catch (MalformedURLException e) {

throw new XFireRuntimeException("Invalid URL", e);

}

}

public Object getEndpoint(QName name) {

Endpoint endpoint = ((Endpoint) endpoints.get((name)));

if ((endpoint) == null) {

throw new IllegalStateException("No such endpoint!");

}

return getEndpoint((endpoint));

}

public Collection getEndpoints() {

return endpoints.values();

}

private void create0() {

TransportManager tm = (org.codehaus.xfire.XFireFactory.newInstance().getXFire().getTransportManager());

HashMap props = new HashMap();

props.put("annotations.allow.interface", true);

AnnotationServiceFactory asf = new AnnotationServiceFactory(new Jsr181WebAnnotations(), tm, new AegisBindingProvider(new JaxbTypeRegistry()));

asf.setBindingCreationEnabled(false);

service0 = asf.create((com.test.service.HelloXfirePortType.class), props);

{

AbstractSoapBinding soapBinding = asf.createSoap11Binding(service0, new QName("http://service.test.com", "HelloXfirePortTypeLocalBinding"), "urn:xfire:transport:local");

}

{

AbstractSoapBinding soapBinding = asf.createSoap11Binding(service0, new QName("http://service.test.com", "HelloXfireHttpBinding"), "http://schemas.xmlsoap.org/soap/http");

}

}

public HelloXfirePortType getHelloXfirePortTypeLocalEndpoint() {

return ((HelloXfirePortType)(this).getEndpoint(new QName("http://service.test.com", "HelloXfirePortTypeLocalEndpoint")));

}

public HelloXfirePortType getHelloXfirePortTypeLocalEndpoint(String url) {

HelloXfirePortType var = getHelloXfirePortTypeLocalEndpoint();

org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);

return var;

}

public HelloXfirePortType getHelloXfireHttpPort() {

return ((HelloXfirePortType)(this).getEndpoint(new QName("http://service.test.com", "HelloXfireHttpPort")));

}

public HelloXfirePortType getHelloXfireHttpPort(String url) {

HelloXfirePortType var = getHelloXfireHttpPort();

org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);

return var;

}

public static void main(String[] args) {

HelloXfireClient client = new HelloXfireClient();

//create a default service endpoint

HelloXfirePortType service = client.getHelloXfireHttpPort();

String string= service.sayHello("hello");

System.out.println(string);

//TODO: Add custom client code here

//

//service.yourServiceOperationHere();

System.out.println("test client completed");

System.exit(0);

}

}

如果能够成功运行并正确获得调用的方法返回的值说明webservice成功实现了

**8 常见错误及解决办法**

（1）jar包冲突

如果运行后得到以下错误

org.codehaus.xfire.XFireRuntimeException: Could not invoke service.. Nested exception is org.codehaus.xfire.fault.XFireFault: Could not marshall type.

说明jar包有冲突

解决办法

**如果发现以下的包用多个**

**jaxb-api-1.0.jar**

**jaxb-api-2.0.jar**

**jaxb-impl-1.0.5.jar**

**jaxb-impl-2.0.1.jar**

**则删除低版本的**

（2）Could not find valid implementation for: 2.0

如果出现如下错误

java.lang.IllegalStateException: Could not find valid implementation for: 2.0

at org.apache.xbean.spring.context.impl.XBeanHelper.createBeanDefinitionReader(XBeanHelper.java:48)

at org.codehaus.xfire.spring.XFireConfigLoader.getXFireApplicationContext(XFireConfigLoader.java:74)

at org.codehaus.xfire.spring.XFireConfigLoader.loadContext(XFireConfigLoader.java:41)

at org.codehaus.xfire.transport.http.XFireConfigurableServlet.loadConfig(XFireConfigurableServlet.java:86)

问题的原因是因为xfire不支持spring2.0的新特性，需要修改service.xml文件，将beans元素的命名空间放到services元素上，修改后的xml如下：

<beans>

<services xmlns="http://xfire.codehaus.org/config/1.0">

</services>

</beans>

另外，由于我使用了MyEclipse的Spring Capailities和WebService Capailities（使用xfire），由于后者自带了Spring1.2.6的包，造成了和2.0版本的冲突，所以必须作以下工作：

**删除**

**<MyEclise\_Home>/eclipse/plugins/com.genuitec.eclipse.ws.xfire\_5.9.100/data/1.2/lib目录的Spring1.2.6包**

**删除Tomcat/Webapps/<项目目录>/WEB-INF/lib目录下的Spring1.2.6包（好像重新部署不能删除已经存在的包）**

**重新启动MyEclipse**

问题解决。

**CXF,Axis,Xfire对比**

**1 最新版本**

@axis最新的版本为April 22, 2006

@axis2最新的版本为19 - Dec - 2010 1.5.4

@xfire最新的版本为1.2.6 - May 3, 2007 网站已经说了“XFire is now CXF” --|||

@cxf 最新的版本为Feb 28, 2011 - Apache CXF 2.3.3 is released!

**2 XFire VS Axis**

XFire是与Axis2 并列的新一代WebService平台。之所以并称为新一代，因为它：

（1）支持一系列Web Service的新标准--JSR181、WSDL2.0 、JAXB2、WS-Security等；

（2）使用Stax解释XML,性能有了质的提高。XFire采用Woodstox 作Stax实现；

（3）容易上手，可以方便快速地从pojo发布服务；

（4）pring的结合；

（5）灵活的Binding机制，包括默认的Aegis,xmlbeans,jaxb2,castor。

**3 XFire与Axis1性能的比较**

（1）XFire比Axis1.3快2-6倍

（2）XFire的响应时间是Axis1.3的1/2到1/5

**4 XFire与Axis2的比较**

虽然XFire与Axis2都是新一代的WebService平台，但是Axis2的开发者太急于推出1.0版本，所以1.0还不是一个稳定的版本，它的开发者宣称1.1版本即将推出，希望1.1版本会是个稳定的版本。在XFire捐献给apache后有人认为Axis2将会灭亡。其实在很多人眼里，Axis2并不是pojo形式，DanDiephouse证明了XFire比Axis更有市场，我也发现了有很多人开始从Axis转向XFire，包括我也在说服身边的人转向利用XFire进行WebService的开发，很典型的是我可以在几分钟之内教会我的团队实用XFire来发布一个他自己的Web服务。

**5 AXIS VS CXF**

在SOA领域，我们认为Web Service是SOA体系的构建单元（building block）。对于服务开发人员来说，AXIS和CXF一定都不会陌生。这两个产品都是Apache孵化器下面的Web Service开源开发工具。Axis2的最新版本是1.3.CXF现在已经到了2.0版本。这两个框架 都是从已有的开源项目发展起来的。Axis2是从Axis1.x系列发展而来。CXF则是XFire和Celtix项目的结合产品。Axis2是从底层全部重新实现，使用了新的扩展性更好模块架构。 CXF也重新的深化了XFire和Celtix这两个开发工具。新产品的退出导致了几个问题。是不是现有的使用Axis 1.x，XFire和Celix的应用需要迁移的新的版本上。如果一个开发人员确定要迁移它的应用到新的框架上，那么他应该选择哪一个呢？相反的，如果一个开发者决定从头开发一个新的Web Service，他应该使用哪个呢？ 这两个框架哪一个更好一些呢？对于系统迁移来说，也许迁移到新的框架并不难。Axis和CXF都提供了迁移的指导。能够给开发者一些迁移的技巧和经验。但是对于这样迁移，这两个开源项目都没有提供迁移的工具。对于这样的迁移工作，尽管很值得去寻找所有的可行方案。Axis2和CXF都有各自不同的WebService开发方法，每个方法都有相当数量拥护者。

**6 通过一个比较矩阵来比较Axis2和CXF变得有现实的意义。这两个项目都开发不够成熟，但是最主要的区别在以下几个方面：**

（1）CXF支持 WS-Addressing，WS-Policy， WS-RM， WS-Security和WS-I Basic Profile。Axis2不支持WS-Policy，但是承诺在下面的版本支持。

（2） CXF可以很好支持Spring。Axis2不能

（3） AXIS2支持更广泛的数据并对，如XMLBeans，JiBX，JaxMe和JaxBRI和它自定义的数据绑定ADB。注意JaxME和JaxBRI都还是试验性的。CXF只支持JAXB和Aegis。在CXF2.1

（4）Axis2支持多语言-除了Java,他还支持C/C++版本。

**5 从开发者的角度比较**

从开发者的角度，两个框架的特性相当的不同。 Axis2的开发方式类似一个小型的应用服务器，Axis2的开发包要以WAR的形式部署到Servlet容器中，比如Tomcat，通过这些容器可以对工作中的Web Service进行很好的监控和管理。Axis2的Web administrion模块可以让我们动态的配置Axis2.一个新的服务可以上载，激活，使之失效，修改web服务的参数。管理UI也可以管理一个或者多个处于运行状态的服务。这种界面化管理方式的一个弊端是所有在运行时修改的参数没有办法保存，因为在重启动之后，你所做的修改就会全部失效。Axis2允许自己作为独立的应用来发布Web Service，并提供了大量的功能和一个很好的模型，这个模型可以通过它本身的架构（modular architecture）不断添加新的功能。有些开发人员认为这种方式对于他们的需求太过于繁琐。这些开发人员会更喜欢CXF。CXF更注重开发人员的工效（ergonomics）和嵌入能力（embeddability）。大多数配置都可以API来完成，替代了比较繁琐的XML配置文件， Spring的集成性经常的被提及，CXF支持Spring2.0和CXF's API和Spring的配置文件可以非常好的对应。CXF强调代码优先的设计方式（code-first design)，使用了简单的API使得从现有的应用开发服务变得方便。不过你选择Axis2还是CXF，你都可以从开源社区得到大量的帮助。这两个框架都有商业公司提供服务，WSO2提供AXIS2的支持，Iona提供CXF的支持。这两公司都有很活跃的开发者社区。Axis2出现的时间较早，CXF的追赶速度快。如果你需要多语言的支持，你应该选择AXIS2。如果你需要把你的实现侧重JAVA并希望和Spring集成，CXF就是更好的选择，特别是把你的Web Service嵌入其他的程序中。如果你觉得这两个框架的新特性对于你并没有太大的用处，你会觉得Axis1也是不错的选择，你应该继续使用它知道你有充分的理由去更换它。

**6 如何抉择**

（1）如果应用程序需要多语言的支持，Axis2 应当是首选了；

（2）如果应用程序是遵循 Spring 哲学路线的话，Apache CXF 是一种更好的选择，特别对嵌入式的 Web Services 来说；

（3）如果应用程序没有新的特性需要的话，就仍是用原来项目所用的框架，比如 Axis1，XFire，Celtrix或 BEA 等等厂家自己的 Web Services 实现，就别劳民伤财了。

**7 纵向对比**

（1）JWS是Java语言对WebService服务的一种实现，用来开发和发布服务。而从服务本身的角度来看JWS服务是没有语言界限的。但是Java语言为Java开发者提供便捷发布和调用WebService服务的一种途径。

（2）Axis2是Apache下的一个重量级WebService框架，准确说它是一个Web Services / SOAP / WSDL 的引擎，是WebService框架的集大成者，它能不但能制作和发布WebService，而且可以生成Java和其他语言版WebService客户端和服务端代码。这是它的优势所在。但是，这也不可避免的导致了Axis2的复杂性，使用过的开发者都知道，它所依赖的包数量和大小都是很惊人的，打包部署发布都比较麻烦，不能很好的与现有应用整合为一体。但是如果你要开发Java之外别的语言客户端，Axis2提供的丰富工具将是你不二的选择。

（3）XFire是一个高性能的WebService框架，在Java6之前，它的知名度甚至超过了Apache的Axis2，XFire的优点是开发方便，与现有的Web整合很好，可以融为一体，并且开发也很方便。但是对Java之外的语言，没有提供相关的代码工具。XFire后来被Apache收购了，原因是它太优秀了，收购后，随着Java6 JWS的兴起，开源的WebService引擎已经不再被看好，渐渐的都败落了。

（4）CXF是Apache旗下一个重磅的SOA简易框架，它实现了ESB（企业服务总线）。CXF来自于XFire项目，经过改造后形成的，就像目前的Struts2来自WebWork一样。可以看出XFire的命运会和WebWork的命运一样，最终会淡出人们的视线。CXF不但是一个优秀的Web Services / SOAP / WSDL 引擎，也是一个不错的ESB总线，为SOA的实施提供了一种选择方案，当然他不是最好的，它仅仅实现了SOA架构的一部分。

**Webservice/RPC/Rest/Httpclient/Soa对比**

**1 关于架构方面有很多名词：RPC/web service/REST/SOA/SOAP**

@归类，从属关系

@区别与联系

@应用场景及优缺点

**2 先对上面的名词做一个概要介绍**

**（1）RPC ,**远程过程调用 (面向方法)，你可以这么理解，就是在另外一台服务器上有一段代码（函数），你可以通过网络远程调用它。用什么协议（http，tcp，udp…），传输什么数据格式（json，xml，二进制…）你都可以自己定义。很简单的概念, 像调用本地服务(方法)一样调用服务器的服务(方法)。

**（2）SOA ,**面向服务的架构(面向消息)，SOA是一种架构方法（SOA不是Web Service，Web Service是目前最适合实现SOA的技术。）

**（3）REST ,** Representational state transfer远程过程调用 (面向资源)

**（4）web service**顾名思义这是一种提供service的形式，而且只能通过http（web）来提供service(web service三要素：SOAP、WSDL(WebServicesDescriptionLanguage)、UDDI(UniversalDescriptionDiscovery andIntegration))。SOAP,简单对象访问协议,是一种轻量的、简单的、基于XML（标准通用标记语言下的一个子集）的协议，它被设计成在WEB上交换结构化的和固化的信息。SOAP和RPC都是web service的实现方式

**3 RPC与SOA的区别**

总的来讲：RPC是一种进程远程调用的方式，更强调的是异构平台之间进程通信的机制。SOA是一种产品架构的理念，以服务为中心，松耦合，通过定义严谨明确的接口进行通信。有比较完善的服务管理机制。两者并不是一个层面上的概念，可以说RPC是SOA架构的一种实现。

**4 RPC与REST的区别**

如果你想只记住一点,那么就请记住 RPC是以动词为中心的, REST是以名词为中心的, 此处的 动词指的是一些方法, 名词是指资源.你会发现,以动词为中心意味着,当你要需要加入新功能时,你必须要添加更多的动词, 这时候服务器端需要实现 相应的动词(方法), 客户端需要知道这个新的动词并进行调用.而以名词为中心, 假使我请求的是 hostname/friends/, 无论这个URI对应的服务怎么变化,客户端是无需 关注和更新的,而这种变化对客户端也是透明的.至于其它的区别,如对实现语言的依赖, 耦合性等,这些都是上面提到的这个根本区别所衍生的.当你每天使用HTTP冲浪时,你都在使用 REST 与远程的服务器进行亲密接触. 当你使用Gtalk和同事朋友沟通时,你则是在享受着 RPC 的便利.

**5 场景及选择**

比如公司要做一个社交游戏的项目, 让你去负责整个系统的后端架构和通信等, 我们就有必要去仔细地学习和研究下facebook/人人网/新浪等社交网站（游戏）开放的API, 我们知道facebook使用的是REST 的架构, 所以即使facebook本身是PHP开发的,这也不妨碍我们使用python来开发, 还有更多的PHP, Java, .net, Perl等客户端API封装.这样就能灵活的适配多端的服务了。于是在想,倘若facebook的架构使用的不是 REST ,会有这样的灵活性吗? 如果使用的是 RPC 可能不会有这么便利。另外,因为我们的前端使用的是flash, 与后端的python通信采用的是 Django+Flex+AMF , 如果你了解 flash,你会知道AMF是一种二进制的flash数据交互协议, 而 它是基于RPC !所以，我们需要根据当前需求的情况来选择REST或则RPC

**6 REST与SOAP的区别**

国外很多大网站都发布了自己的开发API，很多都提供了SOAP和REST两种Web Service，根据调查部分网站的REST风格的使用情况要高于SOAP。但是由于REST只是一种基于Http协议实现资源操作的思想，因此各个网站的REST实现都自有一套，在后面会讲诉各个大网站的REST API的风格。也正是因为这种各自实现的情况，在性能和可用性上会大大高于SOAP发布的web service，但统一通用方面远远不及SOAP。由于这些大网站的SP往往专注于此网站的API开发，因此通用性要求不高。

SOAP协议对于消息体和消息头都有定义，同时消息头的可扩展性为各种互联网的标准提供了扩展的基础，WS-\*系列就是较为成功的规范。但是也由于SOAP由于各种需求不断扩充其本身协议的内容，导致在SOAP处理方面的性能有所下降。同时在易用性方面以及学习成本上也有所增加。REST被人们的重视，其实很大一方面也是因为其高效以及简洁易用的特性。这种高效一方面源于其面向资源接口设计以及操作抽象简化了开发者的不良设计，同时也最大限度的利用了Http最初的应用协议设计理念。同时，在我看来REST还有一个很吸引开发者的就是能够很好的融合当前Web2.0的很多前端技术来提高开发效率。例如很多大型网站开放的REST风格的API都会有多种返回形式，除了传统的xml作为数据承载，还有（JSON,RSS,ATOM）等形式，这对很多网站前端开发人员来说就能够很好的mashup各种资源信息。

**因此在效率和易用性上来说，REST更胜一筹。**

REST对于资源型服务接口来说很合适，同时特别适合对于效率要求很高，但是对于安全要求不高的场景。而SOAP的成熟性可以给需要提供给多开发语言的，对于安全性要求较高的接口设计带来便利。所以我觉得纯粹说什么设计模式将会占据主导地位没有什么意义，关键还是看应用场景。

**7 rest 是一种设计风格**

**http 接口 按照rest 风格设计 就是 restfull http**

**webservice 接口 按照rest 风格设计 就是 restfull webserivce.**

**用http 做例子：**

**正常我们设计一个http接口 ：**

**url: http://www.xx.com/api/getconentbyname**

**请求参数 ：json 类型**

**{“name”:"123132123"};**

**返回格式：{**

**"status":0 ,**

**"msg":"",**

**"data":{**

**}**

**}**

**这和rest 风格的http 接口 比较来说 ，只能算是实现了 一个接口 ，但是 没有 更加规范的定义 http 接口，**

**这个接口的定义 不易于扩展，比如说客户端想要 xml 格式的数据，我们就没有办法了。**

**因为我们定义的接口不灵活，不规范。 rest 风格就是教我们如何规范的定义接口。**

**一般我们设计接口 就只会使用 post 和 get 方法**

**restfull api 教我们设计接口的时候 一定要 适当的使用 put 和delete 方法。**

**总之 restfull api 就是一个风格，按照这个风格设计出来的接口 会很棒，但是不是必须的，我们如果只是要实现某个功能，完全可以忽略 restfull 风格，按照自己的 风格来设计就可以了。**

restfull是一种风格，不是规范，也不是所谓的封装，他只是将http协议用的更彻底了，我们在普通的开发中，虽然说是基于http的，但是http中很多东西我们都没有用到，比如http的put,delete提交方式，通过http的Accept和Content-Type参数获得不同格式的数据。  
首先说url命名，每个url对应一种资源，也就你要请求的数据资源，通过url中的变参传参，比如说查询一个id为123的成员：http://localhost:8080/member/123，id值可以作为url的一部分。并且有一点非常重要，url中的命名都是名词而非动词（比如http://localhost:8080/getmember/123,这种命名就是不符合风格的，或者叫外行的命名）  
其二，http，提供了get，post,delete,put，可以使用这四种不同的提交方式对应不同的业务操作，比如get就是查询，post是更新或新增，delete删除，put是update  
第三，就是数据格式了，根据TTP请求的头信息中用Accept和Content-Type的类型，返回不同的数据格式，html,xml,json，图片等等。  
第四，通过充分的使用http协议，我们可以忽略交互双方的平台，语言等等，就和websevice一样，我们要的仅仅的数据，符合规范格式的数据。

**8 SOAPwebservice 与Restfull webservice之间的区别**

简单对象访问协议（Simple Object Access Protocol，SOAP）是一种基于 XML 的协议，可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（HTTP），简单邮件传输协议（SMTP），多用途网际邮件扩充协议（MIME），基于“通用”传输协议是 SOAP的一个优点。它还支持从消息系统到远程过程调用（Remote Procedure Call，RPC）等大量的应用程序。SOAP提供了一系列的标准，如WSRM（WS-Reliable Messaging）形式化契约确保可靠性与安全性，确保异步处理与调用；WS-Security、WS-Transactions和WS-Coordination等标准提供了上下文信息与对话状态管理。

相对而言，SOAP协议属于复杂的、重量级的协议，当前随着Web2.0的兴起，表述性状态转移（Representational State Transfer，REST）逐步成为一个流行的架构风格。REST是一种轻量级的Web Service架构风格，其实现和操作比SOAP和XML-RPC更为简洁，可以完全通过HTTP协议实现，还可以利用缓存Cache来提高响应速度，性能、效率和易用性上都优于SOAP协议。REST架构对资源的操作包括获取、创建、修改和删除资源的操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法，这种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。REST架构尤其适用于完全无状态的CRUD（Create、Read、Update、Delete，创建、读取、更新、删除）操作。

基于REST的软件体系结构风格（Software Architecture Style）称之为面向资源体系架构（Resource-oriented Architecture，ROA）。按照REST原则设计的软件、体系结构，通常被称为“REST式的”（RESTful），在本文中以下称之为RESTful Web服务，以便于和基于SOAP的Web服务区别。

服务器端采用J2EE，客户端采用JSP、Flex、JavaFX、AIR等可以直接调用Servlet，其他的实现技术基本上不能直接调用，但是无论是那种客户端，对于基于SOAP的Web服务或者基于RESTful Web服务务都是支持的，如AJAX的 XMLHttpRequest、Flex的HTTPService等。