**Webservice第一天**

# 认识webservice

**什么是服务**？

Java应用程序

View

Control/Service

DAO/Model

DB

简单的网络应用使用单一语言写成，它的

唯一外部程序就是它所依赖的数据库。

大家想想是不是这样呢？

1. 现在的应用程序变得越来越复杂，甚至只靠单一的应用程序无法完成全部的工作。更别说只使用一种语言了。

2）大家在写应用程序查询数据库时，并没有考虑过为什么可以将查询结果返回给上层 的应用程序，甚至认为，这就是数据库应该做的，其实不然，这是数据库通过TCP/IP 协议与另一个应用程序进行交流的结果，而上层是什么样的应用程序，是用什么语言，数据库本身并不知道，它只知道接收到了一份协议，这就是SQL92查询标准协议。

3） 既然数据库可以依据某些标准对外部其他应用程序提供服务、而且不关心对方使用 什么语言，那我们为什么就不能实现跨平台、跨语言的服务呢？只要我们用Java写 的代码，可以被任意的语言所调用，我们就实现了跨平台，跨语言的服务！

**复杂的网络应用**：

Java应用程序

View

Control-对外公布服务层

DAO/Model

DB

手机终端

PDA

C#.NET应用

更为复杂的应用除了自己内部

调用之外有可能还会对外部

程序提供服务，更或是调用其他

应用。

C#应用

**WebService定义**: 顾名思义就是基于Web的服务。它使用Web(HTTP)方式，接收和响应外部系统的某种请求。从而实现远程调用。

**Webservice理解**：我们可以调用互联网上查询天气信息Web服务，然后将它嵌入到我们的程序(C/S或B/S程序)当中来，当用户从我们的网点看到天气信息时，他会认为我们为他提供了很多的信息服务，但其实我们什么也没有做，**只是简单了调用了一下服务器上的一段代码而已**。WebSerice可以将你的服务(一段代码)发布到互联网上让别人去调用,也可以调用别人机器上发布的WebService,就像使用自己的代码一样.。

# Webservice调用

## 2.1学习webservice调用的预备知识

名词1：**XML**. Extensible Markup Language －扩展性标记语言

XML，用于传输格式化的数据，是Web服务的基础。

namespace-命名空间。

xmlns=“http://itcast.cn” 使用默认命名空间。

xmlns:itcast=“http://itcast.cn”使用指定名称的命名空间。

名词2：**WSDL** – WebService Description Language – Web服务描述语言。

通过XML形式说明服务在什么地方－地址。

通过XML形式说明服务提供什么样的方法 – 如何调用。

名词3：**SOAP**-Simple Object Access Protocol(简单对象访问协议)

SOAP作为一个基于XML语言的协议用于有网上传输数据。

SOAP = 在HTTP的基础上+XML数据。

SOAP是基于HTTP的。

SOAP的组成如下：

Envelope – 必须的部分。以XML的根元素出现。

Headers – 可选的。

Body – 必须的。在body部分，包含要执行的服务器的方法。和发送到服务器的数据。

## 2.2 webservice服务网址

Webservice服务网站：[http://www.webxml.com.cn](http://www.webxml.com.cn/)

## WSDL解析

Wsdl文档从下往上读

**Types** - 数据类型定义的容器，它使用某种类型系统(一般地使用XML Schema中的类型系统)。（入参和出参的数据类型）   
 **Message** - 通信消息的数据结构的抽象类型化定义。使用Types所定义的类型来定义整个消息的数据结构（入参和出参）。   
 **Operation** - 对服务中所支持的操作的抽象描述，一般单个Operation描述了一个访问入口的请求/响应消息对（方法）。   
 **PortType** - 对于某个访问入口点类型所支持的操作的抽象集合，这些操作可以由一个或多个服务访问点来支持（服务类）。   
**Binding** - 特定服务访问点与具体服务类的绑定（不看内容，看关系）。   
**Port** - 定义为webservice单个服务访问点。   
  
 **Service**- 相关服务访问点的集合。

service服务访问点的集合

port服务访问点

binding

portType(提供服务的类)

operation(服务的方法)

message(输入输出的消息)

types(输入输出消息的约束)

## 2.4生成客户端代码

**1. wsimport**是jdk自带的,可以根据wsdl文档生成客户端调用代码的工具.当然,无论2. 服务器端的WebService是用什么语言写的,都将在客户端生成Java代码.服务器端用什么写的并不重要.

3. 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文件和class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。

示例：

C:/> wsimport –s . <http://192.168.0.100/one?wsdl>

4.注意：-s不能分开，-s后面有个小点，用于指定源代码生成的目录。点即当前目录。

如果使用了-s参数则会在目录下生成两份代码，一份为.class代码。一份为.java代码。

.class代码，可以经过打包以后使用。.java代码可以直接Copy到我们的项目中运行。

调用webservice步骤

1. 打开WSDL文档
2. 从下往上读WSDL文档，先找到Services（服务访问点集合），根据Services里面binding属性找到binding元素，再根据binding元素的type属性找到绑定的portType（服务类）
3. 根据WSDL的地址生成客户端代码wsimport -s . -p com.rl.trans d:/wsCode/EnglishChinese.wsdl
4. 把客户端代码拷贝到项目中
5. 创建服务访问点集合对象
6. 根据服务访问点获得服务类
7. 调用服务类的方法

## 2.5消息体

1) SOAP1.1请求消息体

POST /WebServices/MobileCodeWS.asmx HTTP/1.1

Host: webservice.webxml.com.cn

Content-Type: text/xml; charset=utf-8

Content-Length: length

SOAPAction: "http://WebXml.com.cn/getMobileCodeInfo"

<?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>string</mobileCode>

<userID>string</userID>

</getMobileCodeInfo>

</soap:Body>

</soap:Envelope>

1. SOAP1.1相应消息体

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: length

<?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>

<getMobileCodeInfoResponse xmlns="http://WebXml.com.cn/">

<getMobileCodeInfoResult>string</getMobileCodeInfoResult>

</getMobileCodeInfoResponse>

</soap:Body>

</soap:Envelope>

# jdk发布webservice服务

注意：用Jdk1.6.0\_21以后的版本发布一个WebService服务.

说明： 在JDK1.6中JAX-WS规范定义了如何发布一个webService服务。

JAX-WS是指Java Api for XML – WebService.

与Web服务相关的类，都位于javax.xml.ws.\*包中。

主要类有：

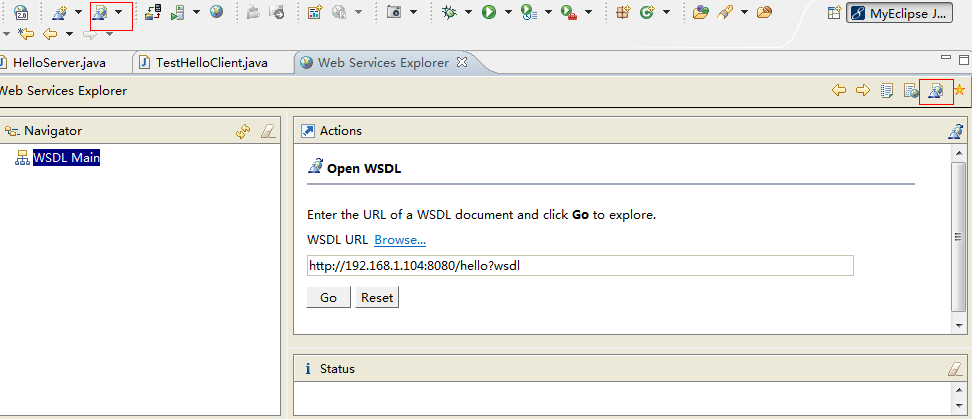
1. @WebService － 它是一个注解，用在类上指定将此类发布成一个webservice服务.
2. Endpoint – 此类为端点服务类，它的方法publish用于将一个已经添加了@WebService注解对象绑定到一个地址的端口上。Endpoint是jdk提供的一个专门用于发布服务的类，它的publish方法接收两个参数，一个是本地的服务地址，二是提供服务的类。它位于javax.xml.ws.\*包中。

static **Endpoint.publish**(String address, Object implementor) 在给定地址处针对指定的实现者对象创建并发布端点。stop方法用于停止服务。

**其他注意事项：**

1. 给类添加上@WebService注解后，类中所有的非静态方法都将会对外公布。不支持静态方法，final方法。
2. 如果希望某个方法(非static,非final)不对外公开，可以在方法上添加@WebMethod(exclude=true)，阻止对外公开。
3. 如果一个类上，被添加了@WebService注解，则必须此类至少有一个可以公开的方法，否则将会启动失败。
4. 服务类中不能没有方法
5. @WebMethod(exclude=**true**)屏蔽方法

使用myeclipse查看消息体



# 其他调用webservice的方式

1. 使用ajax调用

**var** xhr;

**function** invoke(){

**if**(window.ActiveXObject){

xhr = **new** ActiveXObject("Microsoft.XMLHTTP");

}**else**{

xhr = **new** XMLHttpRequest();

}

//指定请求地址

**var** url = "http://127.0.0.1:7777/hello?wsdl";

//定义请求类型和地址和异步

xhr.open("POST", url, **true**);

//设置Content-Type

xhr.setRequestHeader("Content-Type", "text/xml;charset=UTF-8");

//指定回调方法

xhr.onreadystatechange = back;

**var** textVal = document.getElementById("mytext").value;

//组装消息体的数据

**var** data = '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:q0="http://server.hm.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">'

+'<soapenv:Body>'

+'<q0:sayHello>'

+'<arg0>'+textVal+'</arg0>'

+'</q0:sayHello>'

+'</soapenv:Body>'

+'</soapenv:Envelope>';

xhr.send(data);

}

**function** back(){

**if**(xhr.readyState == 4){

**if**(xhr.status == 200){

**var** doc = xhr.responseXML;

alert(doc);

alert(xhr.responseText);

**var** tag = doc.getElementsByTagName("return")[0];

alert(tag)

}

}

}

1. 通过URLConnection调用

//创建url地址

URL url = **new** URL("http://192.168.1.104:8080/hello");

//打开连接

URLConnection conn = url.openConnection();

//转换成HttpURL

HttpURLConnection httpConn = (HttpURLConnection) conn;

//打开输入输出的开关

httpConn.setDoInput(**true**);

httpConn.setDoOutput(**true**);

//设置请求方式

httpConn.setRequestMethod("POST");

//设置请求的头信息

httpConn.setRequestProperty("Content-type", "text/xml;charset=UTF-8");

//拼接请求消息

String data = "<soapenv:Envelope xmlns:soapenv=" +

"\"http://schemas.xmlsoap.org/soap/envelope/\" " +

"xmlns:q0=\"http://server.rl.com/\" " +

"xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" " +

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

+"<soapenv:Body>"

+"<q0:sayHello>"

+"<arg0>renliang</arg0> "

+"</q0:sayHello>"

+"</soapenv:Body>"

+"</soapenv:Envelope>";

//获得输出流

OutputStream out = httpConn.getOutputStream();

//发送数据

out.write(data.getBytes());

//判断请求成功

**if**(httpConn.getResponseCode() == 200){

//获得输入流

InputStream in = httpConn.getInputStream();

//使用输入流的缓冲区

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in));

StringBuffer sb = **new** StringBuffer();

String line = **null**;

//读取输入流

**while**((line = reader.readLine()) != **null**){

sb.append(line);

}

//创建sax的读取器

SAXReader saxReader = **new** SAXReader();

//创建文档对象

Document doc = saxReader.read(**new** StringReader(sb.toString()));

//获得请求响应return元素

List<Element> eles = doc.selectNodes("//return");

**for**(Element ele : eles){

System.*out*.println(ele.getText());

}

# 复杂消息请求

@WebService

**public** **class** TestComplexServer {

List<Person> pList = **new** ArrayList<Person>();

**public** **void** addPerson(Person person){

pList.add(person);

}

**public** List<Person> getPersonList(){

**return** pList;

}

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

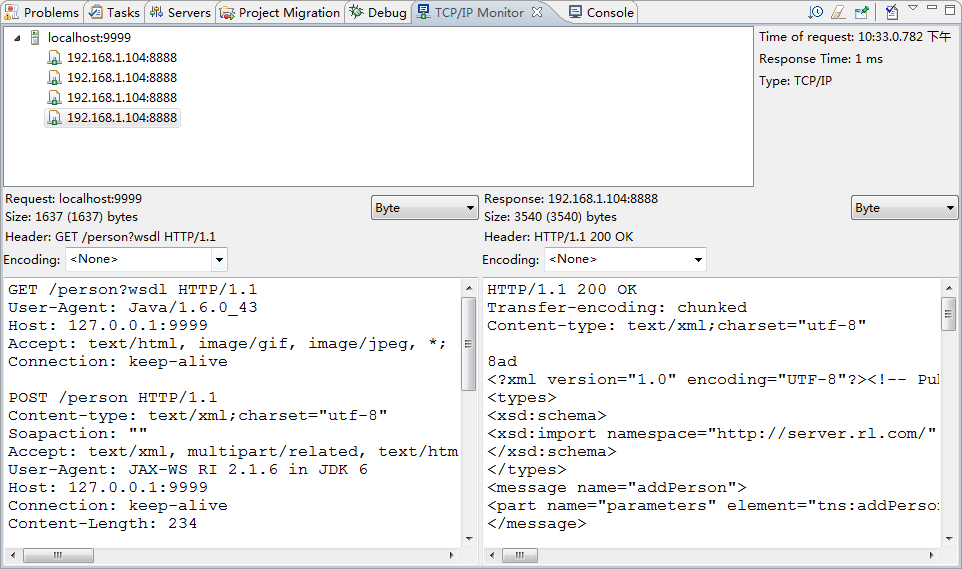
Endpoint.*publish*("http://192.168.1.104:8888/person", **new** TestComplexServer());

}

}

# 监听器

用于查看请求和相应的消息



# wsdl文档元素名称修改

自动生成的文档的名字有时不规范，可以手动进行修改。

@WebService(

portName="myHelloService",修改端口名字

serviceName="HelloServices",修改服务访问点集合名字

name="HelloService",修改服务类的名字

targetNamespace="hello.rl.com" 修改命名空间名字

)

@WebResult(name="sirHello")修改返回值的元素的父标签名字

@WebParam(name="sir")修改传入参数的元素的父标签名字