[**JAX-WS创建和处理SOAP消息**](http://blog.csdn.net/wanghuan203/article/details/9219565)

标签： [SOAP](http://www.csdn.net/tag/SOAP)[JAX-WS](http://www.csdn.net/tag/JAX-WS)

2013-07-01 21:22 6411人阅读 [评论](http://blog.csdn.net/wanghuan203/article/details/9219565#comments)(5) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/wanghuan203/article/details/9219565#report)

本文章已收录于：

[[](http://lib.csdn.net/base/17?source=blogtop) **Java EE知识库**](http://lib.csdn.net/base/17?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

SOA（2） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

     JAX-WS规范是一组XML web services的JAVA API，在 JAX-WS中，一个远程调用可以转换为一个基于XML的协议例如SOAP，在使用JAX-WS过程中，开发者不需要编写任何生成和处理SOAP消息的代码。JAX-WS的运行时实现会将这些API的调用转换成为对应的SOAP消息。

 JAX-WS 也提供了一组针对底层消息进行操作的API调用，你可以通过Dispatch 直接使用SOAP消息或XML消息发送请求或者使用Provider处理SOAP或XML消息。通过web service所提供的互操作环境，我们可以用JAX-WS轻松实现JAVA平台与其他编程环境（.net等）的互操作。

     现在WebService的主要成熟的框架有axis、cxf等，都很好的支持了SOAP协议，而且CXF也提供了对 JAX-WS 全面的支持。可以算是JAX-WS进行封装的框架。

     这篇文章就对处理web services的JAVA基础API JAX-WS进行介绍。

     首先看如何发布web services服务。

     1、编写接口，使用@WebService注解

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">**package** webservice.wh;
3. **import** javax.jws.WebParam;
4. **import** javax.jws.WebResult;
5. **import** javax.jws.WebService;
7. @WebService
8. **public** **interface** IMyService {
9. @WebResult(name="addResult")
10. //对返回值和参数进行名字定义，否则默认为arg0，arg1...
11. **public** **int** add(@WebParam(name="firstA")**int** a ,@WebParam(name="SecondB")**int** b);
12. //与上面方法进行对比
13. **public** **int** minus(**int** a,**int** b);
14. }
15. </span>

    2、编写编写实现类，标明@WebService注解

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">**package** webservice.wh;
3. **import** javax.jws.WebResult;
4. **import** javax.jws.WebService;
6. @WebService(endpointInterface="webservice.wh.IMyService")
7. **public** **class** MyServiceImpl **implements** IMyService {
9. @Override
10. @WebResult(name="addResult")
11. **public** **int** add(**int** a, **int** b) {
12. System.out.print(a+"+"+b+"="+(a+b));
13. **return** a+b;
14. }
16. @Override
17. **public** **int** minus(**int** a, **int** b) {
18. System.out.print(a+"-"+b+"="+(a-b));
19. **return** a-b;
20. }
22. }
23. </span>

     3、编写服务类，根据url和是实现类启动服务

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">**package** webservice.wh;
2. **import** javax.xml.ws.Endpoint;
3. **public** **class** MyService {
4. **public** **static**  **void** main(String[] args){
5. String address="http://localhost:6666/ns";
6. Endpoint.publish(address, **new** MyServiceImpl());
7. }
8. }</span>

     4、运行3中代码，发布服务，在浏览器可查看



     5、使用jax-ws封装好的服务进行客户端调用

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">**package** webservice.wh;
3. **import** java.net.MalformedURLException;
4. **import** java.net.URL;
6. **import** javax.xml.namespace.QName;
7. **import** javax.xml.ws.Service;
9. **public** **class** TestClient {
10. **public** **static** **void** main(String[] args) {
11. **try** {
12. URL url = **new** URL("http://localhost:7777/ns?wsdl");
13. QName sname = **new** QName("http://wh.webservice/", "MyServiceImplService");
14. Service service = Service.create(url,sname);
15. IMyService ms = service.getPort(IMyService.**class**);
16. System.out.println(ms.add(12,33));
17. } **catch** (MalformedURLException e) {
18. e.printStackTrace();
19. }
20. }
21. }
22. </span>

    这样的客户端调用直接使用封装好的service进行操作，在调用过程中将我们的代码转换为SOAP“信件”与服务进行交互，我们不需要编写任何生成和处理SOAP消息的代码。很是方便。不过既然是JAVA 底层API，它也提供了底层的SOAP操作API，我们来用易用，了解一下SOAP是怎样生成的。

     6、创建SOAP消息

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">**public** **class** TestSoap {
3. **private** String ns = "http://service.soap.org/";
4. **private** String wsdlUrl = "http://localhost:6666/ms?wsdl";
6. /\*\*
7. \* 创建SOAP消息
8. \* @author WHuan
9. \*/
10. @Test
12. **public** **void** createSoap() {
13. **try** {
14. // 创建消息工厂
15. MessageFactory factory = MessageFactory.newInstance();
16. // 根据消息工厂创建SoapMessage
17. SOAPMessage message = factory.createMessage();
18. // 创建SOAPPart
19. SOAPPart part = message.getSOAPPart();
20. // 获取SOAPEnvelope
21. SOAPEnvelope envelope = part.getEnvelope();
22. // 通过SoapEnvelope可以获取到相应的Body和Header等信息
23. SOAPBody body = envelope.getBody();
24. // 根据Qname创建相应的节点,Qname是一个带有命名空间的节点
25. QName qname = **new** QName("http://java.wh/webservice/", "add", "ns");
26. // <ns:add xmlns="http://java.wh/webservice/"
27. SOAPBodyElement ele = body.addBodyElement(qname);
28. ele.addChildElement("a").setValue("11");
29. ele.addChildElement("b").setValue("22");
30. // 打印消息信息
31. message.writeTo(System.out);
32. } **catch** (Exception e) {
33. e.printStackTrace();
34. }
35. }
36. }</span>

     代码比较容易理解，首先通过消息工厂MessageFactory创建SOAPMessage，然后创建SOAPPart，我的理解相当于创建了一个邮箱，然后创建信件SOAPEnvelope，之后是SOAP的BODY和Header部分，其中Header部分可以省略。不过现在的API可以省略SOAPPart，直接创建BODY和Header，鉴于Part是经典方式，所以还是按照传统来了。

     这里涉及到SOAP消息的格式，很形象的可以理解为一封信件，鉴于是基础知识，这里便不再赘述，不清楚的可以先补充一下SOAP格式，理解了SOAP的格式之后，上面那段代码是很容易理解的。

     这样进行运行后，我们便可以得到一个完整的SOAP消息：



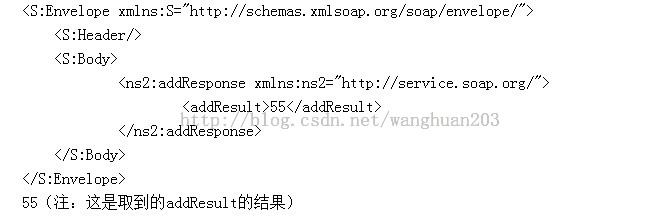
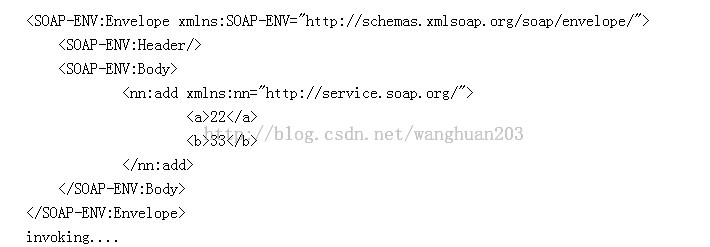
     得到这样的SOAP消息后，怎么与我们发布的WSDL进行交互呢？

     7、SOAP消息传递和处理，使用Dispatch

**[java]** [view plain](http://blog.csdn.net/wanghuan203/article/details/9219565) [copy](http://blog.csdn.net/wanghuan203/article/details/9219565)

1. <span style="font-family:SimSun;font-size:14px;">   /\*\*
2. \* 创建SOAP消息,并提交得到返回值
3. \* @author WHuan
4. \*/
5. @Test
6. **public** **void** soapDispatch() {
7. **try** {
8. // 创建服务service
9. URL url = **new** URL(wsdlUrl);
10. QName sname= **new** QName(ns,"MyServiceImplService");
11. Service service =Service.create(url,sname);
13. //创建DIspatch
14. Dispatch<SOAPMessage> dispatch=service.createDispatch(**new** QName(ns,"MyServiceImplPort"),
15. SOAPMessage.**class**, Service.Mode.MESSAGE);
17. //创建SOAPMessage
18. SOAPMessage msg=MessageFactory.newInstance().createMessage();
19. SOAPEnvelope envelope =msg.getSOAPPart().getEnvelope();
20. SOAPBody body=envelope.getBody();
22. //创建QName来指定消息中传递数据
23. QName ename=**new** QName(ns,"add","nn");
24. //<nn:add xmlns="xx"/>
25. SOAPBodyElement ele=body.addBodyElement(ename);
26. ele.addChildElement("firstA").setValue("22");
27. ele.addChildElement("SecondB").setValue("33");
28. msg.writeTo(System.out);
29. System.out.println("\n invoking....");
31. //通过Dispatch传递消息,会返回相应消息
32. SOAPMessage response = dispatch.invoke(msg);
33. response.writeTo(System.out);
34. System.out.println();
36. //将相应的消息转换为doc对象
37. Document doc= response.getSOAPPart().getEnvelope().getBody().extractContentAsDocument();
38. String str=doc.getElementsByTagName("addResult").item(0).getTextContent();
39. System.out.println(str);
41. } **catch** (Exception e) {
42. e.printStackTrace();
43. }</span>

    这样直接创建了SOAP消息后使用dispatch便可以进行传递，通过extractConentAsDocument方法得到Document类型的返回值，从返回值的SOAP消息中取得我们想要的标签信息。输出结果如下：



    本文主要从使用JAVA自带API出发，介绍一些稍微底层些的SOAP消息的创建和传递处理。理解了底层，也对web Service的框架使用多一份熟悉，也当给大家复习下SOAP消息吧。

**JAX-WS简介：**

JAX\_RPC(Java API for XML-Based RPC)允许Java应用程序可以通过已知的描述信息调用一个基于Java的Web服务，描述信息与Web服务的WSDL描述相一致

JAX-RPC2.0更名为JAX-WS2.0(Java API for XML-Based Web Services)

JAX-WS中，一个远程调用可以转换为一个基于XML的协议，如SOAP。开发者在使用JAX-WS的过程中，不需要编写任何生成、处理SOAP消息的代码，JAX-WS在运行时自动将API的调用转换为相应的SOAP消息

在服务器端，用户只需要通过Java语言定义远程调用所需实现的接口，并提供相应实现，通过调用JAX-WS的服务发布接口即可将其发布为WebService接口

在客户端，用户可以通过JAX-WS的API创建一个代理来实现对于远程服务器端的调用

**JAX-WS服务端：**

JAX-WS服务端采用注释描述WebService，不再依赖WebService描述文件

使用JDK1.6\_45（JDK1.5中不包含所需类）

**[java]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. **package** com.sean.server;
3. **import** javax.jws.WebMethod;
4. **import** javax.jws.WebParam;
5. **import** javax.jws.WebService;
7. @WebService
8. **public** **interface** Plus {
10. @WebMethod
11. **public** **int** plus(@WebParam(name="x") **int** x,@WebParam(name="y") **int** y);
12. }

**[java]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. **package** com.sean.server;
3. **import** javax.jws.WebMethod;
4. **import** javax.jws.WebParam;
5. **import** javax.jws.WebService;
7. @WebService
8. **public** **class** PlusImpl **implements** Plus {
10. @WebMethod
11. **public** **int** plus(@WebParam(name="x") **int** x, @WebParam(name="y") **int** y) {
12. **return** x + y;
13. }
14. }

**[java]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. **package** com.sean.server;
3. **import** javax.xml.ws.Endpoint;
5. **public** **class** Server {
6. **public** **static** **void** main(String[] args) {
7. PlusImpl plus = **new** PlusImpl();
8. String addr = "http://127.0.0.1:8888/Plus";
9. Endpoint.publish(addr, plus);
10. }
11. }

程序启动后，访问http://127.0.0.1:8888/Plus?wsdl即可查看自动生成的WSDL文件

**[html]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
3. JAX-WS RI 2.1.6 in JDK 6. --**>**
4. <!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
5. JAX-WS RI 2.1.6 in JDK 6. --**>**
6. **<definitions** name="PlusImplService" targetNamespace="http://server.sean.com/"
7. xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8. xmlns:tns="http://server.sean.com/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"**>**
9. **<types>**
10. **<xsd:schema>**
11. **<xsd:import** schemaLocation="http://127.0.0.1:8888/Plus?xsd=1"
12. namespace="http://server.sean.com/" **/>**
13. **</xsd:schema>**
14. **</types>**
15. **<message** name="plus"**>**
16. **<part** name="parameters" element="tns:plus" **/>**
17. **</message>**
18. **<message** name="plusResponse"**>**
19. **<part** name="parameters" element="tns:plusResponse" **/>**
20. **</message>**
21. **<portType** name="PlusImpl"**>**
22. **<operation** name="plus"**>**
23. **<input** message="tns:plus" **/>**
24. **<output** message="tns:plusResponse" **/>**
25. **</operation>**
26. **</portType>**
27. **<binding** name="PlusImplPortBinding" type="tns:PlusImpl"**>**
28. **<soap:binding** style="document"
29. transport="http://schemas.xmlsoap.org/soap/http" **/>**
30. **<operation** name="plus"**>**
31. **<soap:operation** soapAction="" **/>**
32. **<input>**
33. **<soap:body** use="literal" **/>**
34. **</input>**
35. **<output>**
36. **<soap:body** use="literal" **/>**
37. **</output>**
38. **</operation>**
39. **</binding>**
40. **<service** name="PlusImplService"**>**
41. **<port** name="PlusImplPort" binding="tns:PlusImplPortBinding"**>**
42. **<soap:address** location="http://127.0.0.1:8888/Plus" **/>**
43. **</port>**
44. **</service>**
45. **</definitions>**

使用 SoapUI5.0.0尝试用上面的WSDL创建WebService服务端报错（org.apache.xmlbeans.XmlException:error:does not close tag.）

使用SoapUI4.5.2则一切正常，只能归咎于不同版本的SoapUI对文件格式校验不同

**JAX-WS客户端：**

**[java]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. **package** com.sean.client;
3. **import** java.net.URL;
5. **import** javax.xml.namespace.QName;
6. **import** javax.xml.ws.Service;
8. **import** com.sean.server.Plus;
10. **public** **class** Client {
11. **public** **static** **void** main(String[] args) **throws** Exception {
12. QName serviceName = **new** QName("http://server.sean.com/", "PlusImplService");
13. QName portName = **new** QName("http://server.sean.com/", "PlusImplPort");
15. String addr = "http://127.0.0.1:8888/Plus?wsdl";
16. URL url = **new** URL(addr);
18. Service service = Service.create(url, serviceName);
19. Plus plus = service.getPort(portName,Plus.**class**);
20. //Plus plus = service.getPort(Plus.class);
22. **int** result = plus.plus(1, 2);
23. System.out.println("result:" + result);
24. }
25. }

使用Plus plus = service.getPort(Plus.class)方法时，客户端调用时报错：

**[plain]** [view plain](http://blog.csdn.net/a19881029/article/details/32101373) [copy](http://blog.csdn.net/a19881029/article/details/32101373)

1. Exception in thread "main" javax.xml.ws.WebServiceException: Undefined port type: {http://server.sean.com/}Plus

解决方式一：

如客户端示例，将Plus plus = service.getPort(Plus.class)修改为Plus plus = service.getPort(portName,Plus.class)

解决方式二：

修改PlusImpl类的@WebService标记修改为@WebService(endpointInterface="com.sean.server.Plus")

参考：<http://stackoverflow.com/questions/13417454/javax-xml-ws-webserviceexception-undefined-port-type-java-struts-soap-wsdl>