# webservice之标签详解 - longwei000的博客 - CSDN博客

## Web Service概述

Web Service的定义
W3C组织对其的定义如下，它是一个软件系统，为了支持跨网络的机器间相互操作交互而设计。Web Service服务通常被定义为一组模块化的API，它们可以通过网络进行调用，来执行远程系统的请求服务。

这里我们从一个程序员的视角来观察web service。在传统的程序编码中，存在这各种的函数方法调用。通常，我们知道一个程序模块M中的方法A，向其发出调用请求，并传入A方法需要的参数P，方法A执行完毕后，返回处理结果R。这种函数或方法调用通常发生在同一台机器上的同一程序语言环境下。现在的我们需要一种能够在不同计算机间的不同语言编写的应用程序系统中，通过网络通讯实现函数和方法调用的能力，而Web service正是应这种需求而诞生的。

最普遍的一种说法就是，Web Service = SOAP + HTTP + WSDL。其中，SOAP Simple Object Access Protocol）协议是web service的主体，它通过HTTP或者SMTP等应用层协议进行通讯，自身使用XML文件来描述程序的函数方法和参数信息，从而完成不同主机的异构系统间的计算服务处理。这里的WSDL（Web Services Description Language）web 服务描述语言也是一个XML文档，它通过HTTP向公众发布，公告客户端程序关于某个具体的 Web service服务的URL信息、方法的命名，参数，返回值等。
下面，我们先来熟悉一下SOAP协议，看看它是如何描述程序中的函数方法、参数及结果对象的。

## SOAP协议简介

什么是SOAP
SOAP 指简单对象访问协议，它是一种基于XML的消息通讯格式，用于网络上，不同平台，不同语言的应用程序间的通讯。可自定义，易于扩展。一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素:
• Envelope 元素，标识XML 文档一条 SOAP 消息
• Header 元素，包含头部信息的XML标签
• Body 元素，包含所有的调用和响应的主体信息的标签
• Fault 元素，错误信息标签。

以上的元素都在 SOAP的命名空间http://www.w3.org/2001/12/soap-envelope中声明;
SOAP的语法规则
• SOAP 消息必须用 XML 来编码
• SOAP 消息必须使用 SOAP Envelope 命名空间
• SOAP 消息必须使用 SOAP Encoding 命名空间
• SOAP 消息不能包含 DTD 引用
• SOAP 消息不能包含 XML 处理指令

SOAP 消息的基本结构
Java代码

```
17. "1.0"?>
18. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
19. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
20.
21.   ...
22.   ...
23.
24.
25.   ...
26.   ...
27.
28.     ...
29.     ...
30.
31.
32.
```

SOAP Envelope 元素

Envelope 元素是 SOAP 消息的根元素。它指明 XML 文档是一个SOAP 消息。它的属性 xmlns:soap的值必须是 http://www.w3.org/2001/12/soap-envelope。

 encodingStyle 属性，语法：soap:encodingStyle="URI"

encodingStyle 属性用于定义文档中使用的数据类型。此属性可出现在任何 SOAP 元素中，并会被应用到元素的内容及元素的所有子元素上。

Java代码

```
8. "1.0"?>
9. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
10. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
11.   ...
12.   Message information goes here
13.   ...
14.
```

SOAP Header 元素

- actor 属性，语法soap:actor="URI"

通过沿着消息路径经过不同的端点，SOAP 消息可从某个发送者传播到某个接收者。并非 SOAP 消息的所有部分均打算传送到 SOAP 消息的最终端点，不过，另一个方面，也许打算传送给消息路径上的一个或多个端点。SOAP 的 actor 属性可被用于将 Header 元素寻址到一个特定的端点。

- mustUnderstand 属性 ，语法soap:mustUnderstand="0|1"

SOAP 的 mustUnderstand 属性可用于标识标题项对于要对其进行处理的接收者来说是强制的还是可选的。假如您向 Header 元素的某个子元素添加了 "mustUnderstand="1"，则要求处理此头部的接收者必须认可此元素。

[java]view plaincopy

```
12. "1.0"?>
13. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
14. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
15.
16. xmlns:m="http://www.jsoso.net/transaction/"
17. soap:mustUnderstand="1"
18. soap:actor="http://www.w3schools.com/appml/“ >234
19.
20. ...
21. ...
22.
```

SOAP Body 元素

必需的 SOAP Body 元素可包含打算传送到消息最终端点的实际 SOAP 消息。Body元素中既可以包含SOAP定义的命名空间中的元素，如Fault，也可以是用户的应用程序自定义的元素。以下是一个用户定义的请求:

[java]view plaincopy

```
10. "1.0"?>
11. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
12. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
13.
14.    "http://www.jsoso.net/prices">
15.      Apples
16.
17.
```

```
18.
```

上面的例子请求苹果的价格。请注意，上面的 m:GetPrice 和 Item 元素是应用程序专用的元素。它们并不是 SOAP 标准的一部分。而对应的 SOAP 响应应该类似这样：

Java代码

```
10. "1.0"?>
11. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
12. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
13.
14.    "http://www.jsoso.net/prices">
15.       1.90
16.
17.
18.
```

SOAP Fault 元素

Fault 元素表示 SOAP的错误消息。它必须是 Body 元素的子元素，且在一条 SOAP 消息中，Fault 元素只能出现一次。Fault 元素拥有下列子元素：

常用的SOAP Fault Codes

HTTP协议中的SOAP 实例

下面的例子中，一个 GetStockPrice 请求被发送到了服务器。此请求有一个 StockName 参数，而在响应中则会返回一个 Price 参数。此功能的命名空间被定义在此地址中： "http://www.jsoso.net/stock"

- SOAP 请求：(注意HTTP的Head属性)

Java代码

```
14. POST /InStock HTTP/1.1
15. Host: www.jsoso.net
16. Content-Type: application/soap+xml; charset=utf-8
17. Content-Length: XXX
18. "1.0"?>
19. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
20. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
21.    "http://www.jsoso.net/stock">
22.
23.       IBM
24.
25.
26.
```

- SOAP 响应：(注意HTTP的Head属性)

Java代码

```
13. HTTP/1.1200 OK
14. Content-Type: application/soap+xml; charset=utf-8
15. Content-Length: XXX
16. "1.0"?>
17. xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
18. soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
19.    "http://www.jsoso.net/stock">
20.
```

```
21.     34.5
22.
23.
24.
```

HTTP协议中的SOAP RPC工作流程

## WSDL简介

介绍过了SOAP，让我们关注Web Service中另外一个重要的组成WSDL。
WSDL的主要文档元素

WSDL文档可以分为两部分。顶部分由抽象定义组成，而底部分则由具体描述组成。抽象部分以独立于平台和语言的方式定义SOAP消息，它们并不包含任何随机器或语言而变的元素。这就定义了一系列服务，截然不同的应用都可以实现。具体部分，如数据的序列化则归入底部分，因为它包含具体的定义。在上述的文档元素中，、、属于抽象定义层，、属于具体定义层。所有的抽象可以是单独存在于别的文件中，也可以从主文档中导入。

WSDL文档的结构实例解析
下面我们将通过一个实际的WSDL文档例子来详细说明各标签的作用及关系。
Java代码

```
78. "1.0" encoding="UTF-8"?>
79. xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
80. xmlns:tns="http://www.jsoso.com/wstest"
81. xmlns:xsd="http://www.w3.org/2001/XMLSchema"
82. xmlns="http://schemas.xmlsoap.org/wsdl/"
83. targetNamespace="http://www.jsoso.com/wstest"
84. name="Example">
85.
86.
87.   import
88.    namespace="http://www.jsoso.com/wstest"
89.    schemaLocation="http://localhost:8080/hello?xsd=1">import>
90.
91.
92. "toSayHello">
93.   "userName" type="xsd:string">
94.
95. "toSayHelloResponse">
96.   "returnWord" type="xsd:string">
97.
98. "sayHello">
99.   "person" type="tns:person">
100.   "arg1" type="xsd:string">
101.
102. "sayHelloResponse">
103.   "personList" type="tns:personArray">
104.
105. "HelloException">
106.   "fault" element="tns:HelloException">
107.
108. "Example">
109.   "toSayHello" parameterOrder="userName">
110.     "tns:toSayHello">
111.     "tns:toSayHelloResponse">
112.
113.   "sayHello" parameterOrder="person arg1">
114.     "tns:sayHello">
115.     "tns:sayHelloResponse">
116.     "tns:HelloException" name="HelloException">
117.
```

```
118.
119. "ExamplePortBinding" type="tns:Example">
120.
121.    transport="http://schemas.xmlsoap.org/soap/http"
122.    style="rpc">
123.   "toSayHello">
124.    "sayHello">
125.
126.      "literal"
127.        namespace="http://www.jsoso.com/wstest">
128.
129.
130.      "literal"
131.         namespace="http://www.jsoso.com/wstest">
132.
133.
134.   "sayHello">
135.    "sayHello">
136.
137.      "literal"
138.        namespace="http://www.jsoso.com/wstest">
139.
140.
141.      "literal"
142.        namespace="http://www.jsoso.com/wstest">
143.
144.     "HelloException">
145.      "HelloException" use="literal">
146.
147.
148.
149. "Example">
150.   "ExamplePort" binding="tns:ExamplePortBinding">
151.     "http://localhost:8080/hello">
152.
153.
154.
```

由于上面的事例XML较长，我们将其逐段分解讲解

WSDL文档的根元素：＜definitions＞

Java代码

```
11. xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
12. xmlns:tns="http://www.jsoso.com/wstest"
13. xmlns:xsd="http://www.w3.org/2001/XMLSchema"
14. xmlns="http://schemas.xmlsoap.org/wsdl/"
15. targetNamespace="http://www.jsoso.com/wstest"
16. name="Example">
17. ……
18. ……
19.
```

＜definitions＞定义了文档中用到的各个xml元素的namespace缩写，也界定了本文档自己的targetNamespace="http://www.jsoso.com/wstest"，这意味着其它的XML要引用当前XML中的元素时，要声明这个namespace。注意xmlns:tns="http://www.jsoso.com/wstest"这个声明，它标示了使用tns这个前缀指向自身的命名空间。

引用

WSDL文档数据类型定义元素：

Java代码

```
8.
9.
10. import
11.  namespace="http://www.jsoso.com/wstest"
12.  schemaLocation="http://localhost:8080/hello?xsd=1">import>
13.
14.
```

标签定义了当前的WSDL文档用到的数据类型。要说明的是，为了最大程度的平台中立性，WSDL 使用 XML Schema 语法来定义数据类型。这些数据类型用来定义web service方法的参数和返回指。对于通用的原生数据类型如：integer , boolean , char , float等，在W3C的标准文档http://www.w3.org/2001/XMLSchema中已经做了定义。这里我们要引入的schema定义 schemaLocation="http://localhost:8080/hello?xsd=1"是我们自定义的Java对象类型。

WSDL文档消息体定义元素：< message >

Java代码

```
17. "toSayHello">
18.  "userName" type="xsd:string">
19.
20. "toSayHelloResponse">
21.  "returnWord" type="xsd:string">
22.
23. "sayHello">
24.  "person" type="tns:person">
25.  "arg1" type="xsd:string">
26.
27. "sayHelloResponse">
28.  "personList" type="tns:personArray">
29.
30. "HelloException">
31.  "fault" element="tns:HelloException">
32.
```

< message > 元素定义了web service函数的参数。 < message > 元素中的每个 < part > 子元素都和某个参数相符。输入参数在 < message > 元素中定义，与输出参数相隔离，输出参数有自己的 < message > 元素。兼作输入、输出的参数在输入输出的 < message > 元素中有它们相应的 < part > 元素。输出 < message > 元素以"Response"结尾，对Java而言方法得返回值就对应一个输出的 < message > 。每个 < part > 元素都有名字和类型属性，就像函数的参数有参数名和参数类型。

在上面的文档中有两个输入参数、两个输出参数和一个错误参数（对应Java中的Exception）。

 输入参数 < message > 的name属性分别命名为toSayHello，sayHello。
toSayHello对应输入参数userName，参数类型为xsd:string，在Java语言中就是String；
sayHello对应两个输入参数person和arg1，类型为tns:person和xsd:string。这里tns:person类型就是引用了< types >标签中的类型定义。

 输出参数 < message > 的name属性分别命名为toSayHelloResponse和sayHelloResponse。
这个名称和输入参数的 < message > 标签name属性对应，在其后面加上Response尾缀。
toSayHelloResponse对应的返回值是returnWord，参数类型为xsd:string；
sayHelloResponse对应的返回值是personList，参数类型为tns:personArray（自定义类型）；

 错误参数 < message > 的name属性为HelloException。
它的子标签element而不是type来定义类型。

以上的message标签的name属性通常使用web service函数方法名作为参照，错误参数标签则使用异常类名为参照。标签中的参数名称，即part子元素的name属性是可自定义的（下一章节详细说明）。message标签的参数类型将引用types标签的定义。

WSDL文档函数体定义元素：< portType >

[java]view plaincopy

```
12. "Example">
13.  "toSayHello" parameterOrder="userName">
14.   "tns:toSayHello">
15.   "tns:toSayHelloResponse">
16.
17.  "sayHello" parameterOrder="person arg1">
18.   "tns:sayHello">
19.   "tns:sayHelloResponse">
20.   "tns:HelloException" name="HelloException">
21.
22.
```

元素是最重要的 WSDL 元素。它可描述一个 web service、可被执行的操作，以及相关的消息。portType的name属性对应Java中的一个服务类的类名。 元素使用其子元素< operation>描述一个web service的服务方法。

在元素中，name属性表示服务方法名，parameterOrder属性表示方法的参数顺序，使用空格符分割多个参数，
如：“parameterOrder="person arg1”。元素的子标签表示输入参数说明，它引用 < message > 标签中的输入参数。表示输出参数说明，它引用 < message > 标签中的输出参数。标签在Java方法中的特别用来表示异常（其它语言有对应的错误处理机制），它引用 < message > 标签中的错误参数。

WSDL绑定实现定义元素：< binding >
[java]view plaincopy

```
31. "ExamplePortBinding" type="tns:Example">
32.
33.   transport="http://schemas.xmlsoap.org/soap/http"
34.   style="rpc">
35.  "toSayHello">
36.   "sayHello">
37.
38.    "literal"
39.     namespace="http://www.jsoso.com/wstest">
40.
41.
42.    "literal"
43.     namespace="http://www.jsoso.com/wstest">
44.
45.
46.  "sayHello">
47.   "sayHello">
48.
49.    "literal"
50.     namespace="http://www.jsoso.com/wstest">
51.
52.
53.    "literal"
54.     namespace="http://www.jsoso.com/wstest">
55.
56.   "HelloException">
57.    "HelloException" use="literal">
58.
59.
60.
```

标签是完整描述协议、序列化和编码的地方，和标签处理抽象的数据内容，而标签是处理数据传输的物理实现。
标签把前三部分的抽象定义具体化。

首先标签使用的transport和style属性定义了Web Service的通讯协议HTTP和SOAP的请求风格RPC。其次子标签将portType中定义的operation同SOAP的请求绑定，定义了操作名称soapAction，输出输入参数和异常的编码方式及命名空间。

WSDL服务地址绑定元素：< service >

[java]view plaincopy

```
6. "Example">
7.   "ExamplePort" binding="tns:ExamplePortBinding">
8.    "http://localhost:8080/hello">
9.
10.
```

service是一套<port>元素。在一一对应形式下，每个<port>元素都和一个location关联。如果同一个<binding>有多个<port>元素与之关联，可以使用额外的URL地址作为替换。

一个WSDL文档中可以有多个<service>元素，而且多个<service>元素十分有用，其中之一就是可以根据目标URL来组织端口。在一个WSDL文档中，<service>的name属性用来区分不同的service。在同一个service中，不同端口，使用端口的"name"属性区分。

这一章节，我们简单的描述了WSDL对SOAP协议的支持，以及在Web Service中的作用。在接下来的章节中，我们将学习如何使用Java6.0的Annotation标签来定义和生成对应的WSDL。

## JavaSE6.0下的Web Service

从JavaSE6.0开始，Java引入了对Web Service的原生支持。我们只需要简单的使用Java的Annotation标签即可将标准的Java方法发布成Web Service。（PS：Java Annotation资料请参考JDK5.0 Annotation学习笔记(一)）

但不是所有的Java类都可以发布成Web Service。Java类若要成为一个实现了Web Service的bean，它需要遵循下边这些原则：

- 这个类必须是public类
- 这些类不能是final的或者abstract
- 这个类必须有一个公共的默认构造函数
- 这个类绝对不能有finalize()方法

下面我们将通过一个具体的Java Web Service代码例子，配合上述的WSDL文件，讲述如何编写JavaSE6.0的原生Web Service应用。

完整的Java Web Service类代码

[java]view plaincopy

```
45. package org.jsoso.jws.server;
46. import java.util.ArrayList;
47. import javax.jws.WebMethod;
48. import javax.jws.WebParam;
49. import javax.jws.WebResult;
50. import javax.jws.WebService;
51. import javax.jws.WebParam.Mode;
52. import javax.jws.soap.SOAPBinding;
53. /
54. * 提供WebService服务的类
55. */
56. @WebService(name="Example", targetNamespace="http://www.jsoso.com/wstest", serviceName="Example")
57. @SOAPBinding(style=SOAPBinding.Style.RPC)
58. publicclass Example {
59. private ArrayList persons = new ArrayList();;
60. /**
61.     *
62.     * 返回一个字符串
63.     * @param userName
64.     * @return
65.     */
66. @WebMethod(operationName="toSayHello",action="sayHello",exclude=false)
67. @WebResult(name="returnWord")//自定义该方法返回值在WSDL中相关的描述
68. public String sayHello(@WebParam(name="userName")String userName) {
69. return"Hello:" + userName;
70.     }
```

```
71. /**
72.     * web services 方法的返回值与参数的类型不能为接口
73.     * @param person
74.     * @return
75.     * @throws HelloException
76.     */
77. @WebMethod(operationName="sayHello", action="sayHello")
78. @WebResult(partName="personList")
79. public Person[] sayHello(@WebParam(partName="person", mode=Mode.IN)Person person,
80.             String userName) throws HelloException {
81. if (person == null || person.getName() == null) {
82. thrownew HelloException("说 hello出错，对像为空。。");
83.         }
84.         System.out.println(person.getName() + " 对 " + userName + " 说：Hello，我今年" + person.getAge() + "岁");
85.         persons.add(person);
86. return persons.toArray(new Person[0]);
87.     }
88. }
```

Annotation 1@WebService(name="Example", targetNamespace="http://www.jsoso.com/wstest", serviceName="Example")
@WebService标签主要将类暴露为WebService，其中targetNamespace属性定义了自己的命名空间，serviceName则定义了< definitions >标签和标签的name属性。

Annotation 2：@SOAPBinding(style=SOAPBinding.Style.RPC)
@SOAPBinding标签定义了WSDL文档中SOAP的消息协议，其中style属性对应SOAP的文档类型，可选的有RPC和DOCUMENT

Annotation 3：@WebMethod(operationName="toSayHello",action="sayHello",exclude=false)
@WebMethod定义Web Service运作的方法
属性action 对应操作的活动，如
属性operationName匹配的wsdl:operation 的名称，如
属性exclude 用于阻止将某一继承方法公开为web服务，默认为false

Annotation 4：@WebResult(name="returnWord")
@ WebResult定义方法返回值得名称，如

Annotation 5：@WebParam(partName="person", mode=Mode.IN
@WebParam定义方法的参数名称，如，其中mode属性表示参数的流向，可选值有IN / OUT / INOUT

这里要着重说明的是，上述Web Service类的sayHello方法中，带有HelloException这个异常声明，造成该服务类不能直接发布成Web Service。需要使用wsgen工具为其生存异常Bean。关于wsgen工具的使用，请参考wsgen与wsimport命令说明

发布一个的Java Web Service
在完成了上述的Web Service Annotation注释后，我们使用wsgen工具为其进行服务资源文件的构造（这里主要是生成一个名为org.jsoso.jws.server.jaxws.HelloExceptionBean的异常bean类），最后使用以下的类发布Web 服务：
[java]view plaincopy

```
35. package org.jsoso.jws.server;
36. import java.util.LinkedList;
37. import java.util.List;
38. import javax.xml.ws.Binding;
39. import javax.xml.ws.Endpoint;
40. import javax.xml.ws.handler.Handler;
41. /**
42. * @author zsy 启动web services服务
43. */
44. publicclass StartServer {
45. /**
46.     * @param args
47.     */
```

```
48. publicstaticvoid main(String[] args) {
49. /*
50.      * 生成Example 服务实例
51.      */
52.     Example serverBean = new Example();
53. /*
54.      * 发布Web Service到http://localhost:8080/hello地址
55.      */
56.     Endpoint endpoint =
57.       Endpoint.publish("http://localhost:8080/hello", serverBean);
58.     Binding binding = endpoint.getBinding();
59. /*
60.      * 设置一个SOAP协议处理栈
61.      * 这里就简单得打印SOAP的消息文本
62.      */
63.     List handlerChain = new LinkedList();
64.     handlerChain.add(new TraceHandler());
65.     binding.setHandlerChain(handlerChain);
66.     System.out.println("服务已启动 http://localhost:8080/hello");
67.   }
68. }
```

在控制台运行这个类，就可以使用URL： http://localhost:8080/hello?wsdl 浏览到上文所描述的WSDL的全文了。这说明您的第一个Web Service应用发布成功!

构建Web Service客户端
使用JavaSE6.0构建Web Service的客户端是一件相当简单的事。这里我们要使用到JDK中的另一个命令行工具wsimport。在控制台下输入以下命令：
引用
wsimport -d ./bin -s ./src -p org.jsoso.jws.client.ref http://localhost:8080/hello?wsdl

即可在包org.jsoso.jws.client.ref中生成客户端的存根及框架文件。其中我们要使用的类只有两个：服务类Example_Service和本地接口Example。编写如下客户端，即可调用Web Service服务：
[java]view plaincopy

```
35. package org.jsoso.jws.client;
36. import org.jsoso.jws.client.ref.*;
37. publicclass RunClient {
38. /**
39.    * @param args
40.    */
41. publicstaticvoid main(String[] args) {
42. //初始化服务框架类
43.     Example_Service service = new Example_Service();
44. //或者本地服务借口的实例
45.     Example server = (Example) service.getExamplePort();
46. try {
47. //调用web service的toSayHello方法
48.       System.out.println("输入toSayHello的返回值——" + server.toSayHello("阿土"));
49.        Person person = new Person();
50.        person.setName("阿土");
51.        person.setAge(25);
52. //调用web service的sayHello方法
53.        server.sayHello(person, "机器人");
54.        person = new Person();
55.        person.setName("aten");
56.        person.setAge(30);
57. //调用web service的sayHello方法
58.        PersonArray list = server.sayHello(person, "机器人");
59. //输出返回值
60.        System.out.println("/n以下输入sayHello的返回值——");
61. for (Person p : list.getItem()) {
```

```
62.             System.out.println(p.getName() + ":" + p.getAge());
63.         }
64.     } catch (HelloException_Exception e) {
65.         e.printStackTrace();
66.     }
67. }
68. }
```

届此，本次Web Service的学习暂告一个段落。Java Web Service是一个相当庞大的知识体系，其中涉及的相关技术较多，这里无法一一道来，我们将会在今后的开发和使用中，同大家做进一步深入的探讨和学习。

## 附录：wsgen与wsimport命令说明

wsgen
wsgen是在JDK的bin目录下的一个exe文件（Windows版），该命令的主要功能是用来生成合适的JAX-WS。它读取Web Service的终端类文件，同时生成所有用于发布Web Service所依赖的源代码文件和经过编译过的二进制类文件。这里要特别说明的是，通常在Web Service Bean中用到的异常类会另外生成一个描述Bean，如果Web Service Bean中的方法有申明抛出异常，这一步是必需的，否则服务器无法绑定该对像。此外，wsgen还能辅助生成WSDL和相关的xsd文件。wsgen从资源文件生成一个完整的操作列表并验证web service是否合法，可以完整发布。
命令参数说明：

- □ -cp 定义classpath
- □ -r 生成 bean的wsdl文件的存放目录
- □ -s 生成发布Web Service的源代码文件的存放目录（如果方法有抛出异常，则会生成该异常的描述类源文件）
- □ -d 生成发布Web Service的编译过的二进制类文件的存放目录（该异常的描述类的class文件）

命令范例：wsgen -cp ./bin -r ./wsdl -s ./src -d ./bin -wsdl org.jsoso.jws.server.Example

wsimport
wsimport也是在JDK的bin目录下的一个exe文件（Windows版），主要功能是根据服务端发布的wsdl文件生成客户端存根及框架，负责与Web Service 服务器通信，并在将其封装成实例，客户端可以直接使用，就像使用本地实例一样。对Java而言，wsimport帮助程序员生存调用web service所需要的客户端类文件.java和.class。要提醒指出的是，wsimport可以用于非Java的服务器端，如：服务器端也许是C#编写的web service，通过wsimport则生成Java的客户端实现。
命令参数说明：

- □ -d 生成客户端执行类的class文件的存放目录
- □ -s 生成客户端执行类的源文件的存放目录
- □ -p 定义生成类的包名