### [簡單的Jax-WS WebService實現](http://haohaoxuexi.iteye.com/blog/1985995)

**博客分類：**

* [cxf](http://haohaoxuexi.iteye.com/category/299032)

[cxf](http://www.iteye.com/blogs/tag/cxf)[spring](http://www.iteye.com/blogs/tag/spring)[Maven](http://www.iteye.com/blogs/tag/Maven)[java2ws](http://www.iteye.com/blogs/tag/java2ws)[cxf-codegen-plugin](http://www.iteye.com/blogs/tag/cxf-codegen-plugin)

本文章已收錄於： [[](http://lib.csdn.net/base/17) Java EE](http://lib.csdn.net/base/17)

**目錄**

1       定義Service

2       服務端發佈Service

2.1      排除WebService中的某個方法

3       客戶端訪問Service

4       java2ws工具的使用

4.1      命令行使用java2ws

4.2      Maven使用java2ws

5       基於[**spring**](http://lib.csdn.net/base/17)的Jax-ws WebService

5.1      Service定義

5.2      服務端發佈Service

5.3      客戶端獲取Service

6       WSDL生成[**Java**](http://lib.csdn.net/base/17)代碼

6.1      wsdl2java工具

6.2      cxf-codegen-plugin

       Jax-ws是WebService的一種規範。

# 1       定義Service

       Jax-ws的WebService定義是通過註解進行的，我們必須在其WebService類的介面上使用@WebService註解進行標記。

**Java代碼  [收藏代碼](javascript:void())**

1. @WebService
2. **public** **interface** HelloWorld {
4. **public** String sayHi(String who);
6. }

       如上，我們把HelloWorld定義為一個WebService，其對應有一個sayHi操作。其對應的實現類如下：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** HelloWorldImpl **implements** HelloWorld {
3. @Override
4. **public** String sayHi(String who) {
5. **return**"Hi, " + who;
6. }
8. }

# 2       服務端發佈Service

       對於Jax-ws WebService而言，發佈Service有兩種方式。

第一種：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** Server {
3. **private** **final** **static** String ADDRESS = "http://localhost:8080/test/jaxws/services/HelloWorld";
5. **public** **static** **void** main(String args[]) {
6. HelloWorld hw = **new** HelloWorldImpl();
7. Endpoint.publish(ADDRESS, hw);
8. }
10. }

第二種：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** Server {
3. **private** **final** **static** String ADDRESS = "http://localhost:8080/test/jaxws/services/HelloWorld";
5. **public** **static** **void** main(String args[]) {
6. HelloWorld hw = **new** HelloWorldImpl();
7. JaxWsServerFactoryBean jwsFactory = **new** JaxWsServerFactoryBean();
8. jwsFactory.setAddress(ADDRESS);   //指定WebService的發布地址
9. jwsFactory.setServiceClass(HelloWorld.**class**);//WebService對應的類型
10. jwsFactory.setServiceBean(hw);//WebService對應的實現對象
11. jwsFactory.create();
12. }
14. }

       發佈之後我們就可以通過發佈位址?wsdl查看WebService的定義了（WSDL是Web Service Description Language的簡稱，即網絡服務描述語言）。通過在瀏覽器輸入<http://localhost:8080/test/jaxws/services/HelloWorld?wsdl>我們可以看到如下內容：

**Xml代碼  [收藏代碼](javascript:void())**

1. This XML file does not appear to have any style information associated
2. with it. The document tree is shown below.
3. **<wsdl:definitions** xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4. xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://jaxws.sample.cxftest.tiantian.com/"
5. xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
6. name="HelloWorldImplService" targetNamespace="http://jaxws.sample.cxftest.tiantian.com/"**>**
7. **<wsdl:types>**
8. **<xs:schema** xmlns:xs="http://www.w3.org/2001/XMLSchema"
9. xmlns:tns="http://jaxws.sample.cxftest.tiantian.com/"
10. elementFormDefault="unqualified" targetNamespace="http://jaxws.sample.cxftest.tiantian.com/"
11. version="1.0"**>**
12. **<xs:element** name="sayHi" type="tns:sayHi" **/>**
13. **<xs:element** name="sayHiResponse" type="tns:sayHiResponse" **/>**
14. **<xs:complexType** name="sayHi"**>**
15. **<xs:sequence>**
16. **<xs:element** minOccurs="0" name="arg0" type="xs:string" **/>**
17. **</xs:sequence>**
18. **</xs:complexType>**
19. **<xs:complexType** name="sayHiResponse"**>**
20. **<xs:sequence>**
21. **<xs:element** minOccurs="0" name="return" type="xs:string" **/>**
22. **</xs:sequence>**
23. **</xs:complexType>**
24. **</xs:schema>**
25. **</wsdl:types>**
26. **<wsdl:message** name="sayHiResponse"**>**
27. **<wsdl:part** element="tns:sayHiResponse" name="parameters"**></wsdl:part>**
28. **</wsdl:message>**
29. **<wsdl:message** name="sayHi"**>**
30. **<wsdl:part** element="tns:sayHi" name="parameters"**></wsdl:part>**
31. **</wsdl:message>**
32. **<wsdl:portType** name="HelloWorld"**>**
33. **<wsdl:operation** name="sayHi"**>**
34. **<wsdl:input** message="tns:sayHi" name="sayHi"**></wsdl:input>**
35. **<wsdl:output** message="tns:sayHiResponse" name="sayHiResponse"**></wsdl:output>**
36. **</wsdl:operation>**
37. **</wsdl:portType>**
38. **<wsdl:binding** name="HelloWorldImplServiceSoapBinding"
39. type="tns:HelloWorld"**>**
40. **<soap:binding** style="document"
41. transport="http://schemas.xmlsoap.org/soap/http" **/>**
42. **<wsdl:operation** name="sayHi"**>**
43. **<soap:operation** soapAction="" style="document" **/>**
44. **<wsdl:input** name="sayHi"**>**
45. **<soap:body** use="literal" **/>**
46. **</wsdl:input>**
47. **<wsdl:output** name="sayHiResponse"**>**
48. **<soap:body** use="literal" **/>**
49. **</wsdl:output>**
50. **</wsdl:operation>**
51. **</wsdl:binding>**
52. **<wsdl:service** name="HelloWorldImplService"**>**
53. **<wsdl:port** binding="tns:HelloWorldImplServiceSoapBinding"
54. name="HelloWorldImplPort"**>**
55. **<soap:address** location="http://localhost:8080/test/jaxws/services/HelloWorld" **/>**
56. **</wsdl:port>**
57. **</wsdl:service>**
58. **</wsdl:definitions>**

       我們可以看到在上面我們的sayHi操作的參數who變成了arg0，這是因為介面在被編譯為class檔的時候不能保存參數名，有時候這會影響可讀性。如果需要參數名顯示的可讀性強一些的話，我們可以使用@WebParam來指定，如：

**Java代碼  [收藏代碼](javascript:void())**

1. @WebService(serviceName="!@#$%^", name="123456")
2. **public** **interface** HelloWorld {
4. **public** String sayHi(@WebParam(name="who") String who);
6. }

@WebService的serviceName可以用來指定service的名稱，默認情況下如果Service是通過Endpoint.publish()方法發佈的則serviceName為實現類的簡單名稱+Service（如HelloWorldImplService），如果是通過JaxWsServerFactoryBean的create方法發佈的則為介面的簡單名稱+Service（如HelloWorldService）。name屬性可以用來指定service對應的portName，默認情況下如果Service是通過Endpoint.publish()方法發佈的則portName為實現類的簡單名稱+Port（如HelloWorldImplPort），如果是通過JaxWsServerFactoryBean的create方法發佈的則為介面的簡單名稱+Port（如HelloWorldPort）。

       除了@WebService、@WebParam之外，Jax-ws還為我們提供了一系列的註解，如@WebMethod、@OneWay。

**@WebMethod**是用來標註在WebService的方法即操作上的，通過它我們可以指定某一個操作的操作名、使用SOAP綁定時的Action名稱，以及該方法是否允許發佈為WebService。@WebMethod有三個屬性：operationName、action和exclude。@WebMethod還是挺有用的，當你發佈的WebService中有方法不希望WebService訪問的時候就可以通過@WebMethod來指定將其排除在外。下面我們來介紹一下使用WebMethod排除WebService中的一個方法。

**@Oneway**是標註在Service介面的操作方法上的。使用@Oneway標註的方法表示它不需要等待服務端的返回，也不需要預留資源來處理服務端的返回。

## 2.1     排除WebService中的某個方法

       首先我們往我們的HelloWorld介面中新增一個方法sayHello。

**Java代碼  [收藏代碼](javascript:void())**

1. @WebService
2. **public** **interface** HelloWorld {
4. **public** String sayHi(@WebParam(name="who") String who);
6. **public** String sayHello(@WebParam(name="who") String who);
8. }

       對於有介面的WebService而言，@WebMethod的exclude定義必須指定在其實現類對應的需要排除的方法上，而其他屬性則必須在介面方法上定義。所以我們的實現類定義如下：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** HelloWorldImpl **implements** HelloWorld {
3. @Override
4. **public** String sayHi(String who) {
5. **return**"Hi, " + who;
6. }
8. @Override
9. @WebMethod(exclude=**true**)
10. **public** String sayHello(String who) {
11. **return**"Hello, " + who;
12. }
14. }

       這樣，在服務發佈時就會把sayHello方法排除在外。有興趣的朋友可以通過查看指定exclude前後的wsdl檔驗證一下sayHello方法是否在exclude為true時被排除了。客戶端在訪問sayHello方法的時候也是不會成功的。

3       客戶端訪問Service

類似於WebService簡單實現裡面的ClientProxyFactoryBean，在使用Jax-ws時我們可以通過JaxWsProxyFactoryBean來訪問服務，如：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** Client {
3. **private** **final** **static** String ADDRESS = "http://localhost:8080/test/jaxws/services/HelloWorld";
5. **public** **static** **void** main(String args[]) {
6. JaxWsProxyFactoryBean jwpFactory = **new** JaxWsProxyFactoryBean();
7. jwpFactory.setAddress(ADDRESS);
8. jwpFactory.setServiceClass(HelloWorld.**class**);
9. HelloWorld hw = (HelloWorld)jwpFactory.create();
10. String response = hw.sayHi("world");
11. System.out.println(response);
12. }
14. }

除了上面的方式之外，我們還可以這樣獲取Service：

**Java代碼  [收藏代碼](javascript:void())**

1. //第一個參數為服務發佈的targetNameSpace，可以通過查看對應的wsdl檔獲得，默認是發佈Service所在包的包名倒過來的形式；第二個參數是serviceName
2. **private** **final** **static** QName SERVICE\_NAME = **new** QName("http://jaxws.sample.cxftest.tiantian.com/", "HelloWorldService");
3. //第一個參數是服務發佈的targetNameSpace，第二個參數是portName
4. **private** **final** **static** QName PORT\_NAME = **new** QName("http://jaxws.sample.cxftest.tiantian.com/", "HelloWorldPort");
5. //服務發佈的地址
6. **private** **final** **static** String ADDRESS = "http://localhost:8080/test/jaxws/services/HelloWorld";
8. **public** **static** **void** main(String args[]) {
9. Service service = Service.create(SERVICE\_NAME);
10. //根據portName、服務發佈地址、數據綁定類型創建一個Port。
11. service.addPort(PORT\_NAME, SOAPBinding.SOAP11HTTP\_BINDING, ADDRESS);//默認是SOAP1.1Binding
12. //獲取服務
13. HelloWorld hw = service.getPort(HelloWorld.**class**);
14. String response = hw.sayHi("world");
15. System.out.println(response);
16. }

       在上面的代碼中我們只需要有一個Service實例，就能通過它來獲取真正的WebService，所以，我們如果把上面的代碼改成如下形式也是可以的。

**Java代碼  [收藏代碼](javascript:void())**

1. //第一個參數為服務發佈的targetNameSpace，可以通過查看對應的wsdl檔獲得，默認是發佈Service所在包的包名倒過來的形式；第二個參數是serviceName
2. **private** **final** **static** QName SERVICE\_NAME = **new** QName("http://jaxws.sample.cxftest.tiantian.com/", "HelloWorldService");
3. //第一個參數是服務發佈的targetNameSpace，第二個參數是portName
4. **private** **final** **static** QName PORT\_NAME = **new** QName("http://jaxws.sample.cxftest.tiantian.com/", "HelloWorldPort");
5. //服務發佈的地址
6. **private** **final** **static** String ADDRESS = "http://localhost:8080/test/jaxws/services/HelloWorld";
8. **public** **static** **void** main(String args[]) {
9. Service service = Service.create(**null**);
10. //根據portName、服務發佈地址、數據綁定類型創建一個Port。
11. service.addPort(PORT\_NAME, SOAPBinding.SOAP11HTTP\_BINDING, ADDRESS);//默認是SOAP1.1Binding
12. //獲取服務
13. HelloWorld hw = service.getPort(HelloWorld.**class**);
14. String response = hw.sayHi("world");
15. System.out.println(response);
16. }

上面這種通過Service來獲取WebService的方法是不適用前面介紹的簡單WebService實現的，即不適用獲取通過ServerFactoryBean發佈的WebService。

# 4       java2ws工具的使用

       Cxf為我們提供了一個java2ws工具。通過該工具我們可以根據SEI（Service Endpoint Implementation的縮寫）類及其相關的類來生成WebService的服務端、客戶端和wsdl文件等。SEI需要是使用了@WebService標註的Service介面或類。

## 4.1     命令行使用java2ws

       在Cxf根目錄下的bin目錄下有一個java2ws工具，我們可以在命令列下使用該指令生成相應代碼。Java2ws指令後面可以接很多參數，主要有：

|  |  |
| --- | --- |
| **參數名** | **作用** |
| -?|-h|-help | 這三個參數都可輸出java2ws指令的幫助資訊 |
| -wsdl | 生成wsdl檔 |
| -o fileName | 指定生成的wsdl檔的名稱 |
| -d resourceDir | 指定資原始檔案存放的目錄，生成的wsdl文件也會放在該目錄下 |
| -server | 指定生成服務端代碼 |
| -client | 指定生成客戶端代碼 |
| -wrapperbean | 指定生成wrapper和fault bean |
| -cp classpath | 指定java2ws搜尋SEI和相關類型類的路徑 |
| -t targetNameSpace | 指定targetNameSpace |
| -servicename serviceName | 指定serviceName |
| -verbose | 在生成代碼的時候顯示註釋資訊 |
| -quiet | 在生成代碼的時候不顯示註釋信息 |
| -s sourceDir | 指定生成的java代碼存放的目錄 |
| -classdir classdir | 指定生成的java代碼編譯後的class類存放的目錄 |
| className | 指定SEI類的名稱 |

       在使用java2ws指令時className是必須給定的，其他的參數都可選。java2ws將會到classpath路徑下或使用-cp參數指定的classpath路徑下尋找SEI類及其相關類型。

       示例：

       java2ws –server com.tiantian.cxftest.sample.jaxws.HelloWorld（生成服務端代碼）

       java2ws –wsdl –cp . com.tiantian.cxftest.sample.jaxws.HelloWorld（在當前目錄下尋找HelloWorld類生成wsdl檔）

## 4.2     Maven使用java2ws

       Cxf為java2ws工具提供了一個Maven的外掛程式，叫cxf-java2ws-plugin。所以當我們的項目是使用Maven的時候我們可以在項目的pom.xml檔中加入Cxf提供的cxf-java2ws-plugin，這樣當我們執行Maven的某個操作的時候就會觸發java2ws指令生成對應的代碼。如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<plugin>**
2. **<groupId>**org.apache.cxf**</groupId>**
3. **<artifactId>**cxf-java2ws-plugin**</artifactId>**
4. **<version>**${cxf.version}**</version>**
5. **<executions>**
6. **<execution>**
7. **<id>**process-classes**</id>**
8. **<phase>**process-classes**</phase>**
9. **<configuration>**
10. **<className>**com.tiantian.cxftest.sample.jaxws.HelloWorld**</className>**
11. **<genWsdl>**true**</genWsdl>**
12. **<verbose>**true**</verbose>**
13. **</configuration>**
14. **<goals>**
15. **<goal>**java2ws**</goal>**
16. **</goals>**
17. **</execution>**
18. **</executions>**
19. **</plugin>**

       對於java2ws指令的參數可以通過configuration元素下的元素來指定。可以配置的參數如下：

|  |  |
| --- | --- |
| **參數名** | **作用** |
| className | 指定SEI類的名稱 |
| classpath | 搜選SEI類等相關類的類路徑，默認情況下會到當前項目的類路徑下尋找 |
| outputFile | 生成的wsdl檔存放的位置，默認是${project.build.directory}/generated/wsdl/${className}.wsdl |
| genWsdl | 默認為true。是否生成wsdl檔。 |
| genServer | 默認false。是否生成Server端代碼。 |
| genClient | 默認為false。是否生成Client端代碼。 |
| genWrapperbean | 默認為false。是否生成wrapper bean |
| frontend | 表明使用的frontend。支持jaxws和simple，默認是jaxws。 |
| databinding | 指定數據綁定。默認frontend為jaxws時用jaxb，simple用aegis |
| serviceName | 指定serviceName |
| soap12 | 指定生成的wsdl檔包含soap1.2綁定 |
| targetNameSpace | 指定用來生成wsdl檔時的target namespace |
| verbose | 生成代碼的過程中顯示註釋信息 |
| quiet | 在生成代碼的時候不顯示註釋信息 |
| attachWsdl | 當為true時，在對當前項目進行Maven的install操作時會把生成的wsdl檔以當前項目的groupId、artifactId和version，以type為wsdl一起安裝到Maven的repository中去。默認為true |
| address | 指定port的address |

       使用cxf-java2ws-plugin外掛程式生成的Java代碼默認會放在項目的根目錄下。

5       基於Spring的Jax-ws WebService

## 5.1     Service定義

       Service定義跟之前的定義是一樣的。

## 5.2     服務端發佈Service

       首先在web.xml文件中定義一個CXFServlet，用於發佈和攔截WebService請求。

**Xml代碼  [收藏代碼](javascript:void())**

1. <!-- Jax-ws實現 -->
2. **<servlet>**
3. **<display-name>**jaxws-cxf**</display-name>**
4. **<servlet-name>**jaxws-cxf**</servlet-name>**
5. **<servlet-class>**org.apache.cxf.transport.servlet.CXFServlet**</servlet-class>**
6. **<load-on-startup>**1**</load-on-startup>**
7. **<init-param>**
8. **<param-name>**config-location**</param-name>**
9. **<param-value>**WEB-INF/jaxws-cxf-servlet.xml**</param-value>**
10. **</init-param>**
11. **</servlet>**
13. **<servlet-mapping>**
14. **<servlet-name>**jaxws-cxf**</servlet-name>**
15. **<url-pattern>**/jaxws/services/\***</url-pattern>**
16. **</servlet-mapping>**

       接下來在我們的WebService設定檔裡面定義我們的WebService發佈，即CXFServlet指定的jaxws-cxf-servlet.xml文件（默認是cxf-servlet.xml文件）。這裡我們定義如下：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<beans** xmlns="http://www.springframework.org/schema/beans"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="http://cxf.apache.org/jaxws"
3. xsi:schemaLocation="
4. http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
5. http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd"**>**
7. <!-- 相當於使用Endpoint.publish()進行服務發佈 -->
8. **<jaxws:endpoint** address="/HelloWorld" implementorClass="com.tiantian.cxftest.sample.jaxws.HelloWorldImpl"**/>**
10. <!-- 相當於使用JaxWsServerFactoryBean進行服務發佈 -->
11. **<jaxws:server** address="/HelloWorld2" serviceClass="com.tiantian.cxftest.sample.jaxws.HelloWorldImpl"**/>**
13. <!-- JaxWsServerFactoryBean使用外部bean作為服務進行發佈 -->
14. **<jaxws:server** address="/HelloWorld3" serviceBean="#hw"**/>**
15. <!-- 普通bean對象 -->
16. **<bean** id="hw" class="com.tiantian.cxftest.sample.jaxws.HelloWorldImpl"**/>**
18. <!-- JaxWsServerFactoryBean使用內部bean作為服務進行發佈 -->
19. **<jaxws:server** address="/HelloWorld4"**>**
20. **<jaxws:serviceBean>**
21. **<bean** class="com.tiantian.cxftest.sample.jaxws.HelloWorldImpl"**/>**
22. **</jaxws:serviceBean>**
23. **</jaxws:server>**
25. **</beans>**

## 5.3     客戶端獲取Service

       客戶端可以直接從Spring的bean設定檔中把WebService配置為一個個普通的Spring bean對象進行使用。這主要有兩種方式，第一種是使用Jaxws命名空間，第二種是使用JaxWsProxyFactoryBean。

**第一種**：

       在定義之前需要往bean設定檔中引入Jax-ws的命名空間。這裡我們在classpath下定義一個jaxws-cxf-client.xml文件，其內容如下所示：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<beans** xmlns="http://www.springframework.org/schema/beans"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="http://cxf.apache.org/jaxws"
3. xsi:schemaLocation="
4. http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
5. http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd"**>**
7. **<jaxws:client** id="hw"
8. address="http://localhost:8080/test/jaxws/services/HelloWorld"
9. serviceClass="com.tiantian.cxftest.sample.jaxws.HelloWorld" **/>**
11. **<jaxws:client** id="hw2"
12. address="http://localhost:8080/test/jaxws/services/HelloWorld2"
13. serviceClass="com.tiantian.cxftest.sample.jaxws.HelloWorld"**/>**
15. **<jaxws:client** id="hw3"
16. address="http://localhost:8080/test/jaxws/services/HelloWorld3"
17. serviceClass="com.tiantian.cxftest.sample.jaxws.HelloWorld"**/>**
19. **<jaxws:client** id="hw4"
20. address="http://localhost:8080/test/jaxws/services/HelloWorld4"
21. serviceClass="com.tiantian.cxftest.sample.jaxws.HelloWorld"**/>**
23. **</beans>**

**第二種**：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<bean** id="factoryBean" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean"**>**
2. **<property** name="address" value="http://localhost:8080/test/jaxws/services/HelloWorld"**/>**
3. **<property** name="serviceClass" value="com.tiantian.cxftest.sample.jaxws.HelloWorld"**/>**
4. **</bean>**
6. **<bean** id="helloWorld" factory-bean="factoryBean" factory-method="create"**/>**

       之後我們就可以把這些定義好的WebService當做一個普通的bean對象來使用了，如：

**Java代碼  [收藏代碼](javascript:void())**

1. **public** **class** SpringClient {
3. **public** **static** **void** main(String args[]) {
4. ApplicationContext context = **new** ClassPathXmlApplicationContext("jaxws-cxf-client.xml");
5. accessService(context, "hw");
6. accessService(context, "hw2");
7. accessService(context, "hw3");
8. accessService(context, "hw4");
9. accessService(context, "helloWorld");
10. }
12. **private** **static** **void** accessService(ApplicationContext context, String beanName) {
13. HelloWorld hw = context.getBean(beanName, HelloWorld.**class**);
14. System.out.println(hw.sayHi("world"));
15. }
17. }

6       WSDL生成Java代碼

       如果我們已經有了wsdl定義檔的話，我們就可以通過wsdl檔生成對應的Java代碼，包括WebService的Service類定義、服務端代碼、客戶端代碼等。這裡主要介紹兩種WSDL生成Java代碼的方式。

## 6.1     wsdl2java工具

       wsdl2java是CXF自帶的工具，在CXF根目錄下的bin目錄下。我們可以在命令行使用這一工具。wsdl2java命令後可以接很多參數，主要有：

|  |  |
| --- | --- |
| 參數 | 作用 |
| -?|-h|-help | 這三個參數都可以查看wsdl2java指令的幫助資訊，可以查看wsdl2java可以帶哪些參數 |
| -p packageName | 指定生成的java類使用的包名稱，如未指定將根據wsdl檔裡面的targetNameSpace來決定。 |
| -d dir | 生成的Java文件的存放在dir目錄 |
| -compile | 編譯生成的Java檔 |
| -classdir dir | 指定編譯生成的檔存放在dir目錄下 |
| -server | 生成WebService的服務端發佈代碼 |
| -client | 生成WebService的客戶端訪問代碼 |
| -impl | 生成Service類的實現類代碼，即上面的HelloWorldImpl |
| -all | 生成所有可以生成的代碼 |
| wsdlurl | Wsdl檔的訪問路徑，可以是本地file，也可以是web上的請求。 |

       如：wsdl2java –all http://localhost:8080/test/services/HelloWorld?wsdl

       wsdl2java在生成Service實現類的時候只會生成基本的代碼結構，至於裡面的操作的具體邏輯還需要我們自己來實現。

## 6.2     cxf-codegen-plugin

       cxf-codegen-plugin是cxf針對於maven的一個外掛程式。當我們的項目是基於Maven的項目時，我們可以在pom.xml文件中引入cxf-codegen-plugin，如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<plugin>**
2. **<groupId>**org.apache.cxf**</groupId>**
3. **<artifactId>**cxf-codegen-plugin**</artifactId>**
4. **<version>**2.7.6**</version>**
5. **<executions>**
6. **<execution>**
7. **<id>**generate-sources**</id>**
8. **<phase>**generate-sources**</phase>**
9. **<goals>**
10. **<goal>**wsdl2java**</goal>**
11. **</goals>**
12. **<configuration>**
13. **<sourceRoot>**${project.build.directory}/generated-sources/cxf**</sourceRoot>**
14. **<wsdlOptions>**
15. **<wsdlOption>**
16. **<wsdl>**${basedir}/src/main/resources/HelloWorld.wsdl**</wsdl>**
17. **</wsdlOption>**
18. **</wsdlOptions>**
19. **</configuration>**
20. **</execution>**
21. **</executions>**
22. **</plugin>**

在上面例子中，當我們執行mvn generate-sources時就會運行wsdl2java這個指令。其配置資訊是通過configuration元素來指定的。通過sourceRoot可以指定生成的Java代碼的存放位置，上面我們指定了生成代碼的存放位置為target目錄下的generated-sources/cxf目錄，該目錄也是在沒有指定sourceRoot時，cxf-codegen-plugin存放生成的Java代碼的默認位置。每個wsdlOption元素對應於一個用來生成代碼的WSDL定義。wsdl元素用於指定wsdl檔的存放位置。wsdl2java指令的其他參數可以通過wsdlOption元素下的extraargs元素來指定。如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<plugin>**
2. **<groupId>**org.apache.cxf**</groupId>**
3. **<artifactId>**cxf-codegen-plugin**</artifactId>**
4. **<version>**2.7.6**</version>**
5. **<executions>**
6. **<execution>**
7. **<id>**generate-sources**</id>**
8. **<phase>**generate-sources**</phase>**
9. **<goals>**
10. **<goal>**wsdl2java**</goal>**
11. **</goals>**
12. **<configuration>**
13. **<wsdlOptions>**
14. **<wsdlOption>**
15. **<wsdl>**${basedir}/src/main/resources/HelloWorld.wsdl**</wsdl>**
16. **<extraargs>**
17. **<extraarg>**-impl**</extraarg>**
18. **<extraarg>**-server**</extraarg>**
19. **</extraargs>**
20. **</wsdlOption>**
21. **</wsdlOptions>**
22. **</configuration>**
23. **</execution>**
24. **</executions>**
25. **</plugin>**

       每一個extraarg元素代表一個參數，如果使用extraarg元素指定wsdl2java參數後面還帶有參數值時，參數值也用一個extraarg來表示。如我們需要指定生成代碼的包名時，我們的cxf-codegen-plugin應該這樣聲明：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<plugin>**
2. **<groupId>**org.apache.cxf**</groupId>**
3. **<artifactId>**cxf-codegen-plugin**</artifactId>**
4. **<version>**2.7.6**</version>**
5. **<executions>**
6. **<execution>**
7. **<id>**generate-sources**</id>**
8. **<phase>**generate-sources**</phase>**
9. **<goals>**
10. **<goal>**wsdl2java**</goal>**
11. **</goals>**
12. **<configuration>**
13. **<wsdlOptions>**
14. **<wsdlOption>**
15. **<wsdl>**${basedir}/src/main/resources/HelloWorld.wsdl**</wsdl>**
16. **<extraargs>**
17. **<extraarg>**-impl**</extraarg>**
18. **<extraarg>**-server**</extraarg>**
19. **<extraarg>**-p**</extraarg>**<!-- 指定包名，後面緊跟著的extraarg為對應的值 -->
20. **<extraarg>**com.tiantian**</extraarg>**
21. **</extraargs>**
22. **</wsdlOption>**
23. **</wsdlOptions>**
24. **</configuration>**
25. **</execution>**
26. **</executions>**
27. **</plugin>**

**使用serviceName元素指定要生成Java代碼的service**

       通過serviceName元素我們可以指定要針對當前wsdl檔裡面的哪個service定義來生成檔。如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<configuration>**
2. **<wsdlOptions>**
3. **<wsdlOption>**
4. **<wsdl>**${basedir}/src/main/resources/HelloWorld.wsdl**</wsdl>**
5. <!-- 指定要生成代碼的service名稱 -->
6. **<serviceName>**HelloWorldImplService**</serviceName>**
7. **</wsdlOption>**
8. **</wsdlOptions>**
9. **</configuration>**

**使用wsdlRoot來指定wsdl檔存放的目錄**

       我們知道在使用cxf-codegen-plugin的時候一個wsdlOption元素代表要生成的一個wsdl定義，當我們有多個wsdl檔時就需要定義多個wsdlOption，如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<configuration>**
2. **<wsdlOptions>**
3. **<wsdlOption>**
4. **<wsdl>**${basedir}/src/main/resources/wsdl/HelloWorld.wsdl**</wsdl>**
5. **</wsdlOption>**
6. **<wsdlOption>**
7. **<wsdl>**${basedir}/src/main/resources/wsdl/HelloWorld2.wsdl**</wsdl>**
8. **</wsdlOption>**
9. **</wsdlOptions>**
10. **</configuration>**

       當我們的wsdl檔少的時候這樣做還可以，但是當wsdl檔比較多的時候這樣做就有點麻煩了。這個時候我們可以通過wsdlRoot元素來指定我們的wsdl文件存放的目錄，cxf-codegen-plugin在掃瞄的時候只會掃瞄wsdlRoot指定的目錄，而不會掃瞄其子目錄。**光指定了wsdlRoot還不能起作用，wsdlRoot必須和includes元素或者excludes元素配合使用**。includes元素表示要包含哪些檔，而excludes元素表示要排除哪些檔。所以當我們要使用wsdl2java指定對wsdlRoot下的所有檔生成java代碼時，我們可以這樣做：

**Java代碼  [收藏代碼](javascript:void())**

1. <configuration>
2. <wsdlRoot>${basedir}/src/main/resources/wsdl</wsdlRoot>
3. <includes>
4. <include>\*.wsdl</include>
5. </includes>
6. </configuration>

       或者這樣做：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<configuration>**
2. **<wsdlRoot>**${basedir}/src/main/resources/wsdl**</wsdlRoot>**
3. **<excludes>**
4. **<exclude>**null**</exclude>**
5. **</excludes>**
6. **</configuration>**

**使用defaultOptions來指定公共的選項**

       當我們使用wsdlRoot元素來指定wsdl檔存放目錄時，我們就不能再指定wsdl2java指令執行時的可選參數了。這個時候我們可以通過defaultOptions元素來指定。defaultOptions元素下指定的選項對所有的wsdlOption元素都會產生作用。如：

**Xml代碼  [收藏代碼](javascript:void())**

1. **<configuration>**
2. **<defaultOptions>**
3. **<extraargs>**
4. **<extraarg>**-all**</extraarg>**
5. **</extraargs>**
6. **</defaultOptions>**
7. **<wsdlRoot>**${basedir}/src/main/resources/wsdl**</wsdlRoot>**
8. **<includes>**
9. **<include>**helloWorld.wsdl**</include>**
10. **</includes>**
11. **</configuration>**