

ImportNew

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java小组](#)
- [工具资源](#)

- 导航条 -

Web Service 那点事儿 (2) —— 使用 CXF 开发 SOAP 服务

2017/05/28 | 分类: [基础技术](#) | [0 条评论](#) | 标签: [CXF](#), [soap](#), [Web Service](#)

分享到:

0 原文出处: [黄勇](#)

选框架犹如选媳妇，选来选去，最后我还是选了“丑媳妇（CXF）”，为什么是它？因为 CXF 是 Apache 旗下的一款非常优秀的 WS 开源框架，具备轻量级的特性，而且能无缝整合到 Spring 中。

其实 CXF 是两个开源框架的整合，它们分别是：Celtix 与 XFire，前者是一款 ESB 框架，后者是一款 WS 框架。话说早在 2007 年 5 月，当 XFire 发展到了它的鼎盛时期（最终版本是 1.2.6），突然对业界宣布了一个令人震惊的消息：“XFire is now CXF”，随后 CXF 2.0 诞生了，直到 2014 年 5 月，CXF 3.0 降临了。真是 7 年磨一剑啊！CXF 终于长大了，相信在不久的将来，一定会取代 Java 界 WS 龙头老大 Axis 的江湖地位，貌似 Axis 自从 2012 年 4 月以后就没有升级了，这是要告别 Java 界的节奏吗？还是后面有更大的动作？

如何使用 CXF 开发基于 SOAP 的 WS 呢?

这就是我今天要与您分享的内容，重点是在 Web 容器中发布与调用 WS，这样也更加贴近我们实际工作的场景。

在 CXF 这个主角正是登台之前，我想先请出今天的配角 Oracle JAX-WS RI，简称：RI（日），全称：Reference Implementation，它是 Java 官方提供的 JAX-WS 规范的具体实现。

先让 RI 来跑跑龙套，先来看看如何使用 RI 发布 WS 吧！

1. 使用 RI 发布 WS

第一步：整合 Tomcat 与 RI

这一步稍微有一点点繁琐，不过也很容易做到。首先您需要通过以下地址，下载一份 RI 的程序包：

```
1 | https://jax-ws.java.net/2.2.8/
```

下载完毕后，只需解压即可，假设解压到 D:/Tool/jaxws-ri 目录下。随后需要对 Tomcat 的 config/catalina.properties 文件进行配置：

```
1 | common.loader=${catalina.base}/lib,${catalina.base}/lib/*.jar,${catalina.home}/lib,${catalina.home}/lib/*.jar,D:/Tool/jaxws-ri/lib/*.jar
```

注意：以上配置中的最后一部分，其实就是在 Tomcat 中添加一系列关于 RI 的 jar 包。

看起来并不复杂哦，只是对现有的 Tomcat 有所改造而已，当然，您将这些 jar 包全部放入自己应用的 WEB-INF/lib 目录中也是可行的。

第二步：编写 WS 接口及其实现

接口部分：



```
1 | <!-- lang: java -->
2 | package demo.ws.soap_jaxws;
3 |
4 | import javax.jws.WebService;
5 |
6 | @WebService
7 | public interface HelloService {
8 |
9 |     String say(String name);
10 | }
```



实现部分：

```

1  <!-- lang: java -->
2  package demo.ws.soap_jaxws;
3
4  import javax.jws.WebService;
5
6  @WebService(
7      serviceName = "HelloService",
8      portName = "HelloServicePort",
9      endpointInterface = "demo.ws.soap_jaxws.HelloService"
10 )
11 public class HelloServiceImpl implements HelloService {
12
13     public String say(String name) {
14         return "hello " + name;
15     }
16 }

```

注意：接口与实现类上都标注 `javax.jws.WebService` 注解，可在实现类的注解中添加一些关于 WS 的相关信息，例如：`serviceName`、`portName` 等，当然这是可选的，为了让生成的 WSDL 的可读性更加强而已。

第三步：在 WEB-INF 下添加 sun-jaxws.xml 文件

就是在这个 `sun-jaxws.xml` 文件里配置需要发布的 WS，其内容如下：

```

1  <!-- lang: xml -->
2  <?xml version="1.0" encoding="UTF-8"?>
3  <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
4
5      <endpoint name="HelloService"
6          implementation="demo.ws.soap_jaxws.HelloServiceImpl"
7          url-pattern="/ws/soap/hello"/>
8
9  </endpoints>

```



这里仅发布一个 endpoint，并配置三个属性：WS 的名称、实现类、URL  式。正是通过这个“URL 模式”来访问 WSDL 的，马上您就可以看到。

第四步：部署应用并启动 Tomcat

当 Tomcat 启动成功后，会在控制台上看到如下信息：

```

1  2014-7-2 13:39:31 com.sun.xml.ws.transport.http.servlet.WSServletDelegate <init>
2  信息: WSSERVLET14: JAX-WS servlet 正在初始化
3  2014-7-2 13:39:31 com.sun.xml.ws.transport.http.servlet.WSServletContextListener contextInitialized

```

4 | 信息: WSSERVLET12: JAX-WS 上下文监听程序正在初始化

哎呦，不错哦！还是中文的。

随后，立马打开您的浏览器，输入以下地址：

1 | `http://localhost:8080/ws/soap/hello`

如果不出意外的话，您现在应该可以看到如下界面了：

Web 服务

端点	信息
服务名: { http://soap_jaxws.ws.demo/ }HelloService 端口名: { http://soap_jaxws.ws.demo/ }HelloServicePort	地址: http://localhost:8080/ws/soap/hello WSDL: http://localhost:8080/ws/soap/hello?wsdl 实现类: demo.ws.soap_jaxws.HelloServiceImpl

看起来这应该是一个 WS 控制台，方便我们查看发布了哪些 WS，可以点击上面的 WSDL 链接可查看具体信息。

看起来 RI 确实挺好的！不仅仅有一个控制台，而且还能与 Tomcat 无缝整合。但 RI 似乎与 Spring 的整合能力并不是太强，也许是因为 Oracle 是 EJB 拥护者吧。

那么，CXF 也具备 RI 这样的特性吗？并且能够与 Spring 很好地集成吗？

CXF 不仅可以将 WS 发布在任何的 Web 容器中，而且还提供了一个便于测试的 Web 环境，实际上它内置了一个 Jetty。

我们先看看如何启动 Jetty 发布 WS，再来演示如何在 Spring 容器中整合 CXF。

2. 使用 CXF 内置的 Jetty 发布 WS

第一步：配置 Maven 依赖

如果您是一位 Maven 用户，那么下面这段配置相信一定不会陌生：

```
1 <!-- lang: xml -->
2 <?xml version="1.0" encoding="UTF-8"?>
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6
7 http://maven.apache.org/xsd/maven-4.0.0.xsd">
8
9     <modelVersion>4.0.0</modelVersion>
10
11     <groupId>demo.ws</groupId>
12     <artifactId>soap_cxf</artifactId>
13     <version>1.0-SNAPSHOT</version>
14
15     <properties>
16         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17         <cxf.version>3.0.0</cxf.version>
18     </properties>
19
20     <dependencies>
21         <!-- CXF -->
22         <dependency>
23             <groupId>org.apache.cxf</groupId>
24             <artifactId>cxf-rt-frontend-jaxws</artifactId>
25             <version>${cxf.version}</version>
26         </dependency>
27         <dependency>
28             <groupId>org.apache.cxf</groupId>
29             <artifactId>cxf-rt-transports-http-jetty</artifactId>
30             <version>${cxf.version}</version>
31         </dependency>
32     </dependencies>
33
34 </project>
```

如果您目前还没有使用 Maven，那么就需要从以下地址下载 CXF 的相关 jar 包，并将其放入应用中。

```
1 | http://cxf.apache.org/download.html
```

第二步：写一个 WS 接口及其实现



接口部分：

```
1 <!-- lang: java -->
2 package demo.ws.soap_cxf;
3
4 import javax.jws.WebService;
5
6 @WebService
7 public interface HelloService {
8
```

```
9      String say(String name);
10 }
```

实现部分：

```
1 <!-- lang: java -->
2 package demo.ws.soap_cxf;
3
4 import javax.ws.WebService;
5
6 @WebService
7 public class HelloServiceImpl implements HelloService {
8
9     public String say(String name) {
10         return "hello " + name;
11     }
12 }
```

这里简化了实现类上的 WebService 注解的配置，让 CXF 自动为我们取默认值即可。

第三步：写一个 JaxWsServer 类来发布 WS

```
1 <!-- lang: java -->
2 package demo.ws.soap_cxf;
3
4 import org.apache.cxf.jaxws.JaxWsServerFactoryBean;
5
6 public class JaxWsServer {
7
8     public static void main(String[] args) {
9         JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();
10        factory.setAddress("http://localhost:8080/ws/soap/hello");
11        factory.setServiceClass(HelloService.class);
12        factory.setServiceBean(new HelloServiceImpl());
13        factory.create();
14        System.out.println("soap ws is published");
15    }
16 }
```



发布 WS 除了以上这种基于 JAX-WS 的方式以外，CXF 还提供了另一种选择，名为 simple 方式。

通过 simple 方式发布 WS 的代码如下：

```
1 <!-- lang: java -->
2 package demo.ws.soap_cxf;
3
4 import org.apache.cxf.frontend.ServerFactoryBean;
5
```

```
6 public class SimpleServer {
7
8     public static void main(String[] args) {
9         ServerFactoryBean factory = new ServerFactoryBean();
10        factory.setAddress("http://localhost:8080/ws/soap/hello");
11        factory.setServiceClass(HelloService.class);
12        factory.setServiceBean(new HelloServiceImpl());
13        factory.create();
14        System.out.println("soap ws is published");
15    }
16 }
```

注意：以 simple 方式发布的 WS，不能通过 JAX-WS 方式来调用，只能通过 simple 方式的客户端来调用，下文会展示 simple 方式的客户端代码。

第四步：运行 JaxWsServer 类

当 JaxWsServer 启动后，在控制台中会看到打印出来的一句提示。随后，在浏览器中输入以下 WSDL 地址：

```
1 | http://localhost:8080/ws/soap/hello?wsdl
```

注意：通过 CXF 内置的 Jetty 发布的 WS，仅能查看 WSDL，却没有像 RI 那样的 WS 控制台。

可见，这种方式非常容易测试与调试，大大节省了我们的开发效率，但这种方式并不适合于生产环境，我们还是需要依靠于 Tomcat 与 Spring。

那么，CXF 在实战中是如何集成在 Spring 容器中的呢？见证奇迹的时候到了！

3. 在 Web 容器中使用 Spring + CXF 发布 WS

Tomcat + Spring + CXF，这个场景应该更加接近我们的实际工作情况，开发过程也是非常自然。



第一步：配置 Maven 依赖



```
1 <!-- lang: xml -->
2 <?xml version="1.0" encoding="UTF-8"?>
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6
7         http://maven.apache.org/xsd/maven-4.0.0.xsd">
8
9         <modelVersion>4.0.0</modelVersion>
```

```
10
11 <groupId>demo.ws</groupId>
12 <artifactId>soap_spring_cxf</artifactId>
13 <version>1.0-SNAPSHOT</version>
14 <packaging>war</packaging>
15
16 <properties>
17   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
18   <spring.version>4.0.5.RELEASE</spring.version>
19   <cxf.version>3.0.0</cxf.version>
20 </properties>
21
22 <dependencies>
23   <!-- Spring -->
24   <dependency>
25     <groupId>org.springframework</groupId>
26     <artifactId>spring-context</artifactId>
27     <version>${spring.version}</version>
28   </dependency>
29   <dependency>
30     <groupId>org.springframework</groupId>
31     <artifactId>spring-web</artifactId>
32     <version>${spring.version}</version>
33   </dependency>
34   <!-- CXF -->
35   <dependency>
36     <groupId>org.apache.cxf</groupId>
37     <artifactId>cxf-rt-frontend-jaxws</artifactId>
38     <version>${cxf.version}</version>
39   </dependency>
40   <dependency>
41     <groupId>org.apache.cxf</groupId>
42     <artifactId>cxf-rt-transport-http</artifactId>
43     <version>${cxf.version}</version>
44   </dependency>
45 </dependencies>
46
47 </project>
```

第二步：写一个 WS 接口及其实现

接口部分：

```
1 <!-- lang: java -->
2 package demo.ws.soap_spring_cxf;
3
4 import javax.jws.WebService;
5
6 @WebService
7 public interface HelloService {
8
9     String say(String name);
```




```
10 | }
```

实现部分：

```
1 | <!-- lang: java -->
2 | package demo.ws.soap_spring_cxf;
3 |
4 | import javax.jws.WebService;
5 | import org.springframework.stereotype.Component;
6 |
7 | @WebService
8 | @Component
9 | public class HelloServiceImpl implements HelloService {
10 |
11 |     public String say(String name) {
12 |         return "hello " + name;
13 |     }
14 | }
```

需要在实现类上添加 Spring 的 `org.springframework.stereotype.Component` 注解，这样才能被 Spring IOC 容器扫描到，认为它是一个 Spring Bean，可以根据 Bean ID（这里是 `helloServiceImpl`）来获取 Bean 实例。

第三步：配置 web.xml

```
1 | <!-- lang: xml -->
2 | <?xml version="1.0" encoding="UTF-8"?>
3 | <web-app xmlns="http://java.sun.com/xml/ns/javaee"
4 |         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5 |         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6 |
7 | http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
8 |
9 |         version="3.0">
10 |
11 |     <!-- Spring -->
12 |     <context-param>
13 |         <param-name>contextConfigLocation</param-name>
14 |         <param-value>classpath:spring.xml</param-value>
15 |     </context-param>
16 |     <listener>
17 |         <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
18 |     </listener>
19 |
20 |     <!-- CXF -->
21 |     <servlet>
22 |         <servlet-name>cxfr</servlet-name>
23 |         <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
24 |     </servlet>
25 |     <servlet-mapping>
26 |         <servlet-name>cxfr</servlet-name>
```



```
27         <url-pattern>/ws/*</url-pattern>
28     </servlet-mapping>
29
30 </web-app>
```

所有带有 /ws 前缀的请求，将会交给被 CXFServlet 进行处理，也就是处理 WS 请求了。目前主要使用了 Spring IOC 的特性，利用了 ContextLoaderListener 加载 Spring 配置文件，即这里定义的 spring.xml 文件。

第四步：配置 Spring

配置 spring.xml：

```
1  <!-- lang: xml -->
2  <?xml version="1.0" encoding="UTF-8"?>
3  <beans xmlns="http://www.springframework.org/schema/beans"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xmlns:context="http://www.springframework.org/schema/context"
6         xsi:schemaLocation="http://www.springframework.org/schema/beans
7
8     http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
9
10
11     http://www.springframework.org/schema/context
12
13
14     http://www.springframework.org/schema/context/spring-context-4.0.xsd">
15
16     <context:component-scan base-package="demo.ws"/>
17
18     <import resource="spring-cxf.xml"/>
19
20 </beans>
```

以上配置做了两件事情：

1. 定义 IOC 容器扫描路径，即这里定义的 demo.ws，在这个包下面（包括所有子包）凡是带有 Component 的类都会扫描到 Spring IOC 容器中。
2. 引入 spring-cxf.xml 文件，用于编写 CXF 相关配置。将配置文件分离，是一种很好的开发方式。

第五步：配置 CXF

配置 spring-cxf.xml：

```
1  <!-- lang: xml -->
```

```
2 <?xml version="1.0" encoding="UTF-8"?>
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:jaxws="http://cxf.apache.org/jaxws"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7
8 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
9
10
11 http://cxf.apache.org/jaxws
12
13
14 http://cxf.apache.org/schemas/jaxws.xsd">
15
16     <jaxws:server id="helloService" address="/soap/hello">
17         <jaxws:serviceBean>
18             <ref bean="helloServiceImpl"/>
19         </jaxws:serviceBean>
20     </jaxws:server>
21
22 </beans>
```

通过 CXF 提供的 Spring 命名空间，即 `jaxws:server`，来发布 WS。其中，最重要的是 `address` 属性，以及通过 `jaxws:serviceBean` 配置的 Spring Bean。

可见，在 Spring 中集成 CXF 比想象的更加简单，此外，还有一种更简单的配置方法，那就是使用 CXF 提供的 `endpoint` 方式，配置如下：

```
1 <!-- lang: xml -->
2 <?xml version="1.0" encoding="UTF-8"?>
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:jaxws="http://cxf.apache.org/jaxws"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7
8 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
9
10
11 http://cxf.apache.org/jaxws
12
13
14 http://cxf.apache.org/schemas/jaxws.xsd">
15
16     <jaxws:endpoint id="helloService" implementor="#helloServiceImpl" address="/soap/hello"/>
17
18 </beans>
```

使用 `jaxws:endpoint` 可以简化 WS 发布的配置，与 `jaxws:server` 相比，确实是一种进步。

注意：这里的 `implementor` 属性值是 `#helloServiceImpl`，这是 CXF 特有的简写方式，并非是 Spring 的规范，意思是通过 Spring 的 Bean ID 获取 Bean 实例。

同样，也可以在 Spring 中使用 `simple` 方式来发布 WS，配置如下：

```
1 <!-- lang: xml -->
2 <?xml version="1.0" encoding="UTF-8"?>
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:simple="http://cxf.apache.org/simple"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7
8 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
9
10
11 http://cxf.apache.org/simple
12
13
14 http://cxf.apache.org/schemas/simple.xsd">
15
16     <simple:server id="helloService" serviceClass="#helloService" address="/soap/hello">
17         <simple:serviceBean>
18             <ref bean="#helloServiceImpl"/>
19         </simple:serviceBean>
20     </simple:server>
21
22 </beans>
```

可见，`simple:server` 与 `jaxws:server` 的配置方式类似，都需要配置一个 `serviceBean`。

比较以上这三种方式，我个人更加喜欢第二种，也就是 `endpoint` 方式，因为它够简单！

至于为什么 CXF 要提供如此之多的 WS 发布方式？我个人认为，CXF 为了满足广大开发者的喜好，也是为了向前兼容，所以这些方案全部保留下来了。



第六步：启动 Tomcat



将应用部署到 Tomcat 中，在浏览器中输入以下地址可进入 CXF 控制台：

```
1 http://localhost:8080/ws
```

Available SOAP services:

HelloService <ul style="list-style-type: none">say	Endpoint address: http://localhost:8080/ws/soap/hello WSDL : http://soap.spring_cxf.ws.demo/HelloServiceImplService Target namespace: http://soap.spring_cxf.ws.demo/
---	---

Available RESTful services:

通过以上过程，可以看出 CXF 完全具备 RI 的易用性，并且与 Spring 有很好的可集成性，而且配置也非常简单。

同样通过这个地址可以查看 WSDL：

```
1 | http://localhost:8080/ws/soap/hello?wsdl
```

注意：紧接在 /ws 前缀后面的 /soap/hello，其实是在 address="/soap/hello" 中配置的。

现在已经成功地通过 CXF 对外发布了 WS，下面要做的事情就是用 WS 客户端来调用这些 endpoint 了。

您可以不再使用 JDK 内置的 WS 客户端，也不必通过 WSDL 打客户端 jar 包，因为 CXF 已经为您提供了多种 WS 客户端解决方案，根据您的口味自行选择吧！

4. 关于 CXF 提供的 WS 客户端

方案一：静态代理客户端

```
1 | <!-- lang: java -->
2 | package demo.ws.soap_cxf;
3 |
4 | import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
5 |
6 | public class JaxWsClient {
7 |
8 |     public static void main(String[] args) {
9 |         JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
10 |         factory.setAddress("http://localhost:8080/ws/soap/hello");
11 |         factory.setServiceClass(HelloService.class);
12 |
13 |         HelloService helloService = factory.create(HelloService.class);
14 |         String result = helloService.say("world");
```



```
15         System.out.println(result);
16     }
17 }
```

这种方案需要自行通过 WSDL 打客户端 jar 包，通过静态代理的方式来调用 WS。这种做法最为原始，下面的方案更有特色。

方案二：动态代理客户端

```
1  <!-- lang: java -->
2  package demo.ws.soap_cxf;
3
4  import org.apache.cxf.endpoint.Client;
5  import org.apache.cxf.jaxws.endpoint.dynamic.JaxWsDynamicClientFactory;
6
7  public class JaxWsDynamicClient {
8
9      public static void main(String[] args) {
10         JaxWsDynamicClientFactory factory = JaxWsDynamicClientFactory.newInstance();
11         Client client = factory.createClient("http://localhost:8080/ws/soap/hello?wsdl");
12
13         try {
14             Object[] results = client.invoke("say", "world");
15             System.out.println(results[0]);
16         } catch (Exception e) {
17             e.printStackTrace();
18         }
19     }
20 }
```

这种方案无需通过 WSDL 打客户端 jar 包，底层实际上通过 JDK 的动态代理特性完成的，CXF 实际上做了一个简单的封装。与 JDK 动态客户端不一样的是，此时无需使用 HelloService 接口，可以说是货真价实的 WS 动态客户端。

方案三：通用动态代理客户端

```
1  <!-- lang: java -->
2  package demo.ws.soap_cxf;
3
4  import org.apache.cxf.endpoint.Client;
5  import org.apache.cxf.endpoint.dynamic.DynamicClientFactory;
6
7  public class DynamicClient {
8
9      public static void main(String[] args) {
10         DynamicClientFactory factory = DynamicClientFactory.newInstance();
11         Client client = factory.createClient("http://localhost:8080/ws/soap/hello?wsdl");
12
13         try {
14             Object[] results = client.invoke("say", "world");
```



```

15         System.out.println(results[0]);
16     } catch (Exception e) {
17         e.printStackTrace();
18     }
19 }
20 }

```

这种方案与“方案三”类似，但不同的是，它不仅用于调用 JAX-WS 方式发布的 WS，也用于使用 simple 方式发布的 WS，更加智能了。

方案四：基于 CXF simple 方式的客户端

```

1  <!-- lang: java -->
2  package demo.ws.soap_cxf;
3
4  import org.apache.cxf.frontend.ClientProxyFactoryBean;
5
6  public class SimpleClient {
7
8      public static void main(String[] args) {
9          ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
10         factory.setAddress("http://localhost:8080/ws/soap/hello");
11         factory.setServiceClass(HelloService.class);
12         HelloService helloService = factory.create(HelloService.class);
13         String result = helloService.say("world");
14         System.out.println(result);
15     }
16 }

```

这种方式仅用于调用 simple 方式发布的 WS，不能调用 JAX-WS 方式发布的 WS，这是需要注意的。

方案五：基于 Spring 的客户端

方法一：使用 JaxWsProxyFactoryBean

```

1  <!-- lang: xml -->
2  <?xml version="1.0" encoding="UTF-8"?>
3  <beans xmlns="http://www.springframework.org/schema/beans"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://www.springframework.org/schema/beans
6
7  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">
8
9         <bean id="factoryBean" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
10             <property name="serviceClass" value="demo.ws.soap_spring_cxf.HelloService"/>
11             <property name="address" value="http://localhost:8080/ws/soap/hello"/>
12         </bean>
13
14         <bean id="helloService" factory-bean="factoryBean" factory-method="create"/>

```



```
15  
16 </beans>
```

方法二：使用 jaxws:client (推荐)

```
1 <!-- lang: xml -->  
2 <?xml version="1.0" encoding="UTF-8"?>  
3 <beans xmlns="http://www.springframework.org/schema/beans"  
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
5       xmlns:jaxws="http://cxf.apache.org/jaxws"  
6       xsi:schemaLocation="http://www.springframework.org/schema/beans  
7  
8       http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  
9  
10      http://cxf.apache.org/jaxws  
11  
12      http://cxf.apache.org/schemas/jaxws.xsd">  
13  
14     <jaxws:client id="helloService"  
15                  serviceClass="demo.ws.soap_spring_cxf.HelloService"  
16                  address="http://localhost:8080/ws/soap/hello"/>  
17  
18  
19  
20 </beans>
```

客户端代码：

```
1 <!-- lang: java -->  
2 package demo.ws.soap_spring_cxf;  
3  
4 import org.springframework.context.ApplicationContext;  
5 import org.springframework.context.support.ClassPathXmlApplicationContext;  
6  
7 public class Client {  
8  
9     public static void main(String[] args) {  
10         ApplicationContext context = new ClassPathXmlApplicationContext("spring-client.xml");  
11  
12         HelloService helloService = context.getBean("helloService", HelloService.class);  
13         String result = helloService.say("world");  
14         System.out.println(result);  
15     }  
16 }
```

谈不上那种方案更加优秀，建议根据您的实际场景选择最为合适的方案。

5. 总结

通过阅读本文，相信您已经大致了解了 CXF 的基本用法。可独立使用，也可与 Spring 集成；可面向 API 来编程，也可使用 Spring 配置；发布 WS 的方式有多种，调用 WS 的方式同样也有多种。

尤其是 Spring + CXF 这对搭档，让发布 WS 更加简单，只需以下四个步骤：

1. 配置 web.xml
2. 编写 WS 接口及其实现
3. 配置 CXF 的 endpoint
4. 启动 Web 容器

当然，目前您看到的都是 WS 的基础特性，下期我将带您走进 WS 的高级话题 —— 基于 WS 的 Security 解决方案，业界称为 WS-Security 规范，使用它可确保您的 SOAP 服务更加安全。

0



相关文章

- [Web Service 那点事儿 \(4\) —— 使用 CXF 开发 REST 服务](#)
- [Web Service 那点事儿 \(3\) —— SOAP 及其安全控制](#)
- [SOAP webserivce 和 RESTful webservice 对比及区别](#)
- [Web Service监控教程：如何识别不良部署](#)
- [Web Service入门](#)
- [使用Java创建RESTful Web Service](#)
- [Java Web Services面试问题集锦](#)
- [跟我学Spring3 \(11.1\)：SSH集成开发积分商城之概述](#)
- [Spring4新特性 \(4\)：集成Bean Validation 1.1\(JSR-349\)到SpringMVC](#)
- [线程及同步的性能 — 线程池/ ThreadPoolExecutors/ ForkJoinPool](#)



发表评论

Comment form

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*

请填写评论内容

(*) 表示必填项

提交评论



还没有评论。

[« 深入 JVM 分析 spring-boot 应用 hibernate-validator NoClassDefFoundError](#)

[HashMap 和 Hashtable 到底哪不同? »](#)

Search for:



- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

0 [图解 CMS 垃圾回收机制，你值得...](#)

1 [2018 年 Java 平台发布计划之新...](#)

2 [Java 异常进阶](#)

3 [通向架构师的道路（第一天）之 Apache ...](#)

4 [G1 垃圾收集器之对象分配过程](#)

5 [面试必问的 volatile，你了解多少？](#)



6 [通向架构师的道路（第二天）之 apache tom...](#)









7 [通向架构师的道路（第三天）之 apach...](#)

8 [通向架构师的道路（第四天）之 Tomc...](#)





最新评论

- 
Re: [成小胖学习 ActiveMQ · ...](#)
像是在讲故事，过程很不错 yang
- 
Re: [记一次 Spring Maven 打包...](#)
不好意思 第一次评论需要审核 怕有爬虫机器人制造垃圾评论 唐小娟
- 
Re: [MySQL 死锁与日志二三事](#)
补充：update 走的是二级索引，表中有自增主键； jianhaiqing
- 
Re: [MySQL 死锁与日志二三事](#)
update 假如走索引的话，索引到的数据很多，会一下锁所有行么？ 实际过程来看是不会的，过程是怎么... jianhaiqing
- 
Re: [MySQL 死锁与日志二三事](#) 
脚本很有用，感谢您的分享 jianhaiqing
- 
Re: [Java String 对 null 对象...](#)
确实null是个问题，分开处理是最直接的。如果我们写对象去封装null->Null 这样更像... 沙漠的模样
- 



Re: [JVM类加载的那些事](#)

常量在编译期间会进行替换, 输出Consts.A是会输出100的, 但是类确实没有加载, 请作者更正 Mr.Z



Re: [MySQL 死锁与日志二三事](#)

case1 中, 假如能给出show create table 和 update sql 语句以及前... jianhaiqing



关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的, 这是一个很特别的时刻 :)

ImportNew 由两个 Java 关键字 import 和 new 组成, 意指: Java 开发者学习新知识的网站。import 可认为是学习和吸收, new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email: ImportNew.com@gmail.com

新浪微博: [@ImportNew](#)



推荐微信号



反馈建议: ImportNew.com@gmail.com

广告与商务合作QQ: 2302462408

推荐关注

[小组](#) — 好的话题、有启发的回复、值得信赖的圈子

[头条](#) — 写了文章? 看干货? 去头条!

[相亲](#) — 为IT单身男女服务的征婚传播平台

[资源](#) — 优秀的工具资源导航

[翻译](#) — 活跃 & 专业的翻译小组

[博客](#) — 国内外的精选博客文章

[设计](#) — UI,网页, 交互和用户体验

[前端](#) — JavaScript, HTML5, CSS

[安卓](#) — 专注Android技术分享

[iOS](#) — 专注iOS技术分享

[Java](#) — 专注Java技术分享

[Python](#) — 专注Python技术分享

© 2018 ImportNew

