

## Spring 整合 CXF, 发布 RSETful 风格 WebService

这篇文章是承接之前 CXF 整合 Spring 的这个项目示例的延伸, 所以有很大一部分都是一样的。关于发布 CXF WebServer 和 Spring 整合 CXF 这里就不再多赘述了。如果你对 Spring 整合 CXF WebService 不了解, 具体你可以参看这两篇文章:

<http://www.cnblogs.com/hoojo/archive/2011/03/30/1999563.html>

<http://www.cnblogs.com/hoojo/archive/2012/07/13/2590593.html>

如果你不了解 restful 风格的 WebService, 你可以参考:

<http://www.oracle.com/technetwork/articles/javase/index-137171.html>

SpringMVC 对 RESTful 的支持:

<http://www.cnblogs.com/hoojo/archive/2011/06/10/2077422.html>

使用 Jersey 框架, 搭建 RESTful WebService (这个也比较简单)

<http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/>

官方文档: <http://jersey.java.net/nonav/documentation/latest/user-guide.html#d4e8>

其中, 比较常用的 RESTful 框架就有 Jersey、Spring REST、CXF RESTful, 这些都可以很好的整合 Spring 框架, 发布也相当的简单。且简单、易用、易上手, 文档也比较丰富。

开发环境:

System: Windows

JavaEE Server: tomcat6

JavaSDK: jdk6+

IDE: eclipse、MyEclipse 6.6

开发依赖库:

JDK6、JavaEE5、CXF-2.3.3、Spring 3.0.4

Email: [hoojo\\_@126.com](mailto:hoojo_@126.com)

Blog: [http://blog.csdn.net/IBM\\_hoojo](http://blog.csdn.net/IBM_hoojo)

<http://hoojo.cnblogs.com/>

<http://hoojo.blogjava.net>

下面我们就接着 <http://www.cnblogs.com/hoojo/archive/2011/03/30/1999563.html>

这篇文章, 开始我们 CXF RESTful WebService 的旅程, enjoy~! ^\_\*

## 准备工作

首先, 你需要添加相关的 jar 包

其中, jsr331-api-1.1.1.jar 是必须的, 利用 CXF 发布 REST 服务得用到它, 在 cxf 的 lib 库中可以找到这个 jar。

下载地址: <http://www.apache.org/dyn/closer.cgi?path=/cxf/2.3.11/apache-cxf-2.3.11.zip>

其它的 jar 包都是非必须的!

JavaEntity

```

package com.hoo.entity;

import java.util.Map;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * <b>function:</b> MapBean 封装 Map 集合元素
 * @author hoojo
 * @createDate 2012-7-20 下午 01:22:31
 * @file MapBean.java
 * @package com.hoo.entity
 * @project CXFWebService
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0
 */
@XmlRootElement
public class MapBean {
    private Map<String, User> map;
    //@XmlElement(type = User.class)
    public Map<String, User> getMap() {
        return map;
    }
    public void setMap(Map<String, User> map) {
        this.map = map;
    }
}

```

```

package com.hoo.entity;

import java.util.HashMap;
import java.util.List;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * <b>function:</b> Users Entity
 * @author hoojo
 * @createDate 2011-3-18 上午 09:27:31
 * @file Users.java
 * @package com.hoo.entity
 * @project CXFWebService
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0

```

```

*/
@XmlRootElement(name = "UserInfos")
public class Users {
    private List<User> users;
    private User[] userArr;
    private HashMap<String, User> maps;
    // getter/setter
}

package com.hoo.entity;

import java.io.Serializable;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * <b>function:</b>User Entity
 * @author hoojo
 * @createDate Dec 16, 2010 10:20:02 PM
 * @file User.java
 * @package com.hoo.entity
 * @project AxisWebService
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0
 */
@XmlRootElement(name = "UserInfo")
public class User implements Serializable {
    private static final long serialVersionUID = 677484458789332877L;
    private int id;
    private String name;
    private String email;
    private String address;
    //getter/setter
    @Override
    public String toString() {
        return this.id + "#" + this.name + "#" + this.email + "#" +
            this.address;
    }
}

```

一、定义你的 **WebService** 的接口 **RESTSample.java**, 代码如下

```

package com.hoo.service;

import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;

import com.hoo.entity.MapBean;
import com.hoo.entity.User;
import com.hoo.entity.Users;

```

```

/*

```

注释 (Annotation)：在 javax.ws.rs.\* 中定义，是 JAX-RS (JSR 311) 规范的一部分。

@Path：定义资源基 URI。由上下文根和主机名组成，资源标识符类似于

http://localhost:8080/RESTful/rest/hello。

@GET：这意味着以下方法可以响应 HTTP GET 方法。

@Produces：以纯文本方式定义响应内容 MIME 类型。

@Context：使用该注释注入上下文对象，比如 Request、Response、UriInfo、ServletContext 等。

@Path("{contact}")：这是 @Path 注释，与根路径 “/contacts” 结合形成子资源的 URI。

@PathParam("contact")：该注释将参数注入方法参数的路径，在本例中就是联系人 id。

其他可用的注释有 @FormParam、@QueryParam 等。

@Produces：响应支持多个 MIME 类型。在本例和上一个示例中，APPLICATION/XML 将是默认的 MIME 类型。

```

*/

```

```

/**

```

```

* <b>function:</b> CXF RESTful 风格 WebService

```

```

* @author hoojo

```

```

* @createDate 2012-7-20 下午 01:23:04

```

```

* @file RESTSampleSource.java

```

```

* @package com.hoo.service

```

```

* @project CXFWebService

```

```

* @blog http://blog.csdn.net/IBM_hoojo

```

```

* @email hoojo_@126.com

```

```

* @version 1.0

```

```

*/

```

```

@Path(value = "/sample")

```

```

public interface RESTSample {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String doGet();
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/request/{param}")
    public String doRequest(@PathParam("param") String param,
        @Context HttpServletRequest servletRequest, @Context
        HttpServletResponse servletResponse);
    @GET
    @Path("/bean/{id}")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public User getBean(@PathParam("id") int id);
    @GET
    @Path("/list")
    @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
    public Users getList();
    @GET
    @Path("/map")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public MapBean getMap();
    /*
    @Consumes: 声明该方法使用 HTML FORM。
    @FormParam: 注入该方法的 HTML 属性确定的表单输入。
    @Response.created(uri).build(): 构建新的 URI 用于新创建的联系人
    (/contacts/{id}) 并设置响应代码 (201/created)。
    您可以使用 http://localhost:8080/Jersey/rest/contacts/<id> 访问新联系人
    */
    @POST
    @Path("/postData")
    public User postData(User user) throws IOException;
    @PUT
    @Path("/putData/{id}")
    @Consumes(MediaType.APPLICATION_XML)
    public User putData(@PathParam("id") int id, User user);
    @DELETE
    @Path("/removeData/{id}")
    public void deleteData(@PathParam("id") int id);
}

```

二、**RESTSample** 接口的实现，这里我们只是简单的实现下，并不是涉及实际的具体业务

```
package com.hoo.service;
```

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.UriInfo;
import com.hoo.entity.MapBean;
import com.hoo.entity.User;
import com.hoo.entity.Users;

```

```

/*

```

注释 (Annotation)：在 javax.ws.rs.\* 中定义，是 JAX-RS (JSR 311) 规范的一部分。

@Path：定义资源基 URI。由上下文根和主机名组成，资源标识符类似于

http://localhost:8080/RESTful/rest/hello。

@GET：这意味着以下方法可以响应 HTTP GET 方法。

@Produces：以纯文本方式定义响应内容 MIME 类型。

@Context：使用该注释注入上下文对象，比如 Request、Response、UriInfo、ServletContext 等。

@Path("{contact}")：这是 @Path 注释，与根路径 “/contacts” 结合形成子资源的 URI。

@PathParam("contact")：该注释将参数注入方法参数的路径，在本例中就是联系人 id。其他可用的注释有 @FormParam、@QueryParam 等。

@Produces：响应支持多个 MIME 类型。在本例和上一个示例中，APPLICATION/XML 将是默认的 MIME 类型。

```

*/

```

```

/**

```

```

* <b>function:</b> CXF RESTful 风格 WebService

```

```

* @author hoojo

```

```

* @createDate 2012-7-20 下午 01:23:04

```

```

* @file RESTSampleSource.java

```

```

* @package com.hoo.service
* @project CXFWebService
* @blog http://blog.csdn.net/IBM_hoojo
* @email hoojo_@126.com
* @version 1.0
*/
@Path(value = "/sample")
public class RESTSampleSource implements RESTSample {
    @Context
    private UriInfo uriInfo;
    @Context
    private Request request;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String doGet() {
        return "this is get rest request";
    }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/request/{param}")
    public String doRequest(@PathParam("param") String param,
        @Context HttpServletRequest servletRequest, @Context
        HttpServletResponse servletResponse) {
        System.out.println(servletRequest);
        System.out.println(servletResponse);
        System.out.println(servletRequest.getParameter("param"));
        System.out.println(servletRequest.getContentType());
        System.out.println(servletResponse.getCharacterEncoding());
        System.out.println(servletResponse.getContentType());
        return "success";
    }

    @GET
    @Path("/bean/{id}")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public User getBean(@PathParam("id") int id) {
        System.out.println("####getBean####");
        System.out.println("id:" + id);
        System.out.println("Method:" + request.getMethod());
        System.out.println("uri:" + uriInfo.getPath());
        System.out.println(uriInfo.getPathParameters());
        User user = new User();
        user.setId(id);
        user.setName("Jojo");
    }
}

```

```

return user;
}
@GET
@Path("/list")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public Users getList() {
    System.out.println("####getList####");
    System.out.println("Method:" + request.getMethod());
    System.out.println("uri:" + uriInfo.getPath());
    System.out.println(uriInfo.getPathParameters());
    List<User> list = new ArrayList<User>();
    User user = null;
    for (int i = 0; i < 4; i++) {
        user = new User();
        user.setId(i);
        user.setName("JojO-" + i);
        list.add(user);
    }
    Users users = new Users();
    users.setUsers(list);
    return users;
}
@GET
@Path("/map")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public MapBean getMap() {
    System.out.println("####getMap####");
    System.out.println("Method:" + request.getMethod());
    System.out.println("uri:" + uriInfo.getPath());
    System.out.println(uriInfo.getPathParameters());
    Map<String, User> map = new HashMap<String, User>();
    User user = null;
    for (int i = 0; i < 4; i++) {
        user = new User();
        user.setId(i);
        user.setName("JojO-" + i);
        map.put("key-" + i, user);
    }
    MapBean bean = new MapBean();
    bean.setMap(map);
    return bean;
}
/*

```

@Consumes: 声明该方法使用 HTML FORM。



`@FormParam`: 注入该方法的 HTML 属性确定的表单输入。

`@Response.created(uri).build()`: 构建新的 URI 用于新创建的联系人 (`/contacts/{id}`) 并设置响应代码 (`201/created`)。

您可以使用 `http://localhost:8080/Jersey/rest/contacts/<id>` 访问新联系人  
\*/

`@POST`

`@Path("/postData")`

`@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })`

`public User postData(User user) throws IOException {`

`System.out.println(user);`

`user.setName("jojo##12321321");`

`return user;`

`}`

`@PUT`

`@Path("/putData/{id}")`

`@Produces({ MediaType.APPLICATION_XML })`

`public User putData(@PathParam("id") int id, User user) {`

`System.out.println("#####putData#####");`

`System.out.println(user);`

`user.setId(id);`

`user.setAddress("hoojo#gz");`

`user.setEmail("hoojo_@126.com");`

`user.setName("hoojo");`

`System.out.println(user);`

`return user;`

`}`

`@DELETE`

`@Path("/removeData/{id}")`

`public void deleteData(@PathParam("id") int id) {`

`System.out.println("#####deleteData#####" + id);`

`}`

`}`

三、配置我们的 **WebService**, 修改 `applicationContext-server.xml`。这里主要是添加 **jaxrs** 标签的支持, 修改头部文件如下:

`<?xml version="1.0" encoding="UTF-8"?>`

`<beans xmlns="http://www.springframework.org/schema/beans"`

`xmlns:context="http://www.springframework.org/schema/context"`

`xmlns:jaxws="http://cxf.apache.org/jaxws"`

`xmlns:jaxrs="http://cxf.apache.org/jaxrs"`

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

`xsi:schemaLocation="http://www.springframework.org/schema/beans`

`http://www.springframework.org/schema/beans/spring-beans-3.0.xsd`

`http://www.springframework.org/schema/context`

`http://www.springframework.org/schema/context/spring-context-3.0.xsd`

```
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd
http://cxf.apache.org/jaxrs
http://cxf.apache.org/schemas/jaxrs.xsd">
```

特别注意上面加粗带下划线的部分，这是新增加的配置。我们发布 restful WebService 需要用到它。

然后在配置文件中添加如下配置

```
<import resource="classpath:META-INF/cxf/cxf.xml"/>
<import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"/>
<import resource="classpath:META-INF/cxf/cxf-servlet.xml"/>
<bean id="restSample" class="com.hoo.service.RESTSampleSource"/>
<!-- 这里的地址很重要，客户端需要通过这个地址来访问 WebService -->
<jaxrs:server id="restServiceContainer" address="/rest">
<jaxrs:serviceBeans>
<ref bean="restSample" />
</jaxrs:serviceBeans>
<jaxrs:extensionMappings>
<entry key="json" value="application/json" />
<entry key="xml" value="application/xml" />
</jaxrs:extensionMappings>
<jaxrs:languageMappings>
<entry key="en" value="en-gb"/>
</jaxrs:languageMappings>
</jaxrs:server>
```

这样服务器端就完成了 CXF RESTful WebService 的发布，启动你的 tomcat。然后在浏览器中服务地址：<http://localhost:8000/CXFWebService/>（其实这里请求的是 CXFServlet，你可以看看上一篇 Spring 整合 CXF 文章的 web.xml 的配置）  
你就可以看到我们这里刚刚发布的 RESTSample rest 的 WebService

你也可以看看里面的 xml，也就是 WebService 的 wsdl 文件内容。我们找一个 GET 方式的 WebService 的方法，在浏览器中调用一下试试

<http://localhost:8000/CXFWebService/rest/sample/bean/123>

这个 url 对应到下面这个方法

```
@GET
@Path("/bean/{id}")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public User getBean(@PathParam("id") int id)
```

结果如下

一篇 xml 文档内容。

#### 四、编写客户端代码，调用 **RESTful Webservice**

```
package com.hoo.client;

import java.io.IOException;
import javax.ws.rs.core.MediaType;
import org.apache.cxf.jaxrs.client.WebClient;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.hoo.entity.MapBean;
import com.hoo.entity.User;
import com.hoo.entity.Users;
import com.hoo.service.RESTSample;

/**
 * <b>function:</b> RESTful 风格 Webservice
 * @author hoojo
 * @createDate 2012-7-20 下午 03:31:03
 * @file RSETServiceClient.java
 * @package com.hoo.client
 * @project CXFWebService
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0
 */
public class RSETServiceClient {

    private static WebClient client;

    @Before
    public void init() {
        // 手动创建 webClient 对象，注意这里的地址是发布的那个/rest 地址
        //String url = "http://localhost:8000/CXFWebService/rest/";
        //client = WebClient.create(url);

        // 从 Spring Ioc 容器中拿 webClient 对象
        ApplicationContext ctx = new
            ClassPathXmlApplicationContext("applicationContext-client.xml");
        client = ctx.getBean("webClient", WebClient.class);
    }

    @After
    public void destory() {
```

```

    }
    @Test
    public void testGet() {
        System.out.println(client.path("sample").accept(MediaType.TEXT_PLAIN)
            .get(String.class));
    }
    @Test
    public void testRequest() {
        System.out.println(client.path("sample/request/234234").accept(MediaType.TEXT_PLAIN)
            .get(String.class));
    }
    @Test
    public void testBean() {
        User user = client.path("sample/bean/{id}",
            25).accept(MediaType.APPLICATION_XML).get(User.class);
        System.out.println(user);
    }
    @Test
    public void testList() {
        System.out.println(client.path("sample/list").accept(MediaType.APPLICATION_XML)
            .get(Users.class).getUsers());
    }
    @Test
    public void testMap() {
        System.out.println(client.path("sample/map").accept(MediaType.APPLICATION_XML)
            .get(MapBean.class).getMap());
    }
    @Test
    public void testDeleteData() {
        client.path("sample/removeData/23").delete();
    }
    @Test
    public void testPostData() {
        User user = new User();
        user.setId(21432134);
        user.setAddress("hoojo#gz");
        user.setEmail("hoojo_@126.com");
        user.setName("hoojo");
        System.out.println(client.path("sample/postData").accept(MediaType.APPLICATION_XML)
            .post(user, User.class));
    }
    @Test
    public void testPutData() {
        User user = new User();

```

```

user.setId(21432134);
System.out.println(client.path("sample/putData/1").accept(MediaType.APPLICATION_XML).put(user).getEntity());
}
}

```

如果你喜欢用 **Spring** 的方式，还需要在 `applicationContext-client.xml` 中增加如下配置

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">
<bean id="webClient" class="org.apache.cxf.jaxrs.client.WebClient"
factory-method="create">
<constructor-arg type="java.lang.String"
value="http://localhost:8000/CXFWebService/rest/" />
</bean>
</beans>

```

这种是利用 **WebClient** 对象来调用 **WebService**，还有一种方法也可以调用 **WebService**，代码如下：

```

// 手动创建
//RETSample sample =
JAXRSClientFactory.create("http://localhost:8000/CXFWebService/rest",
RETSample.class);

```

// 从 Spring Ioc 容器中拿 webClient 对象

```

ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext-client.xml");
RETSample sample = ctx.getBean("restSampleBean", RETSample.class);

```

```

System.out.println(sample);

```

```

System.out.println(sample.doGet());
//System.out.println(sample.doRequest("haha", null, null));
System.out.println(sample.getBean(22));
System.out.println(sample.getList());
System.out.println(sample.getMap().getMap());
User user = new User();
user.setId(21432134);

```

```

user.setAddress("hoojo#gz");
user.setEmail("hoojo_@126.com");
user.setName("hoojo");
System.out.println(sample.postData(user));
System.out.println(sample.putData(111, user));
sample.deleteData(2);

```

这种方式相对比 **WebClient** 要简单，直接使用接口中的方法即可。同样如果你要整合到 **Spring** 可以在 **applicationContext-client.xml** 中增加配置如下：

```

<bean id="restSampleBean"
class="org.apache.cxf.jaxrs.client.JAXRSClientFactory" factory-
method="create">
<constructor-arg type="java.lang.String"
value="http://localhost:8000/CXFWebService/rest/" />
<constructor-arg type="java.lang.Class"
value="com.hoo.service.RESTSample" />
</bean>

```

执行以上方法可以看到控制台打印结果如下：

```

client console
org.apache.cxf.jaxrs.client.ClientProxyImpl@1cf7491
this is get rest request
22#Jojo#null#null
com.hoo.entity.Users@16eb6bc
{key-0=0#Jojo-0#null#null, key-1=1#Jojo-1#null#null, key-2=2#Jojo-
2#null#null, key-3=3#Jojo-3#null#null}

```