编程人生

JAVA webservice之CXF

博客分类：
- j2ee相关

昨天我们一起学习了一下xfire，今天我们来看一下CXF，为什么学完那个接着学这个呢。因为CXF是在xfire的基础上实现

的，所以我们学习它会比较简单点，毕竟我们昨天刚看过了xfire的实现方法。废话少说，直接来例子。

1）首先呢，还是包的问题，在http://cxf.apache.org/download.html这里可以下到最新版的CXF，当然，我用的是最新版的。接下来还是那句废话，建WEB项目，放入JAR包。而JAR包我们就不选择了，一堆全部放入。

我们会看到它包含了spring的JAR包，后面当我们需要把CXF作为WEB项目部署时，就需要用到spring的配置文件，这个后面再讲。

还是接口类和实现类：

Java代码

```
@WebService
public interface IReaderService {
        public Reader getReader(@WebParam(name="name") String name,@WebParam(name="password") String password);
        public List<Reader> getReaders();
}
```

Java代码

```
@WebService(endpointInterface="com.cxf.servlet.IReaderService",serviceName="readerService")
public class ReaderService implements IReaderService{
        public Reader getReader(@WebParam(name="name") String name,@WebParam(name="password") String password) {
                return new Reader(name,password);
        }

        public List<Reader> getReaders(){
                List<Reader> readerList = new ArrayList<Reader>();
                readerList.add(new Reader("shun1","123"));
                readerList.add(new Reader("shun2","123"));
                return readerList;
        }
}
```

这两个类除了加入注解外，其他均和昨天讲的webservice的一样。这里就不多讲了，对注解的解释，大家可以看看JAVAEE的文档。不过按意思应该很容易理解的。

接下来就是JAVABEAN，还是那个Reader类：

Java代码

```
public class Reader{
        private static final long serialVersionUID = 1L;
        private String name;
        private String password;

        public Reader(){}
        public Reader(String name,String password) {
                this.name = name;
                this.password = password;
        }
        //Get/Set方法省略
        public String toString(){
                return "Name:"+name+",Password:"+password;
        }

}
```

上面的已经写完了。

2）我们要用做WEB项目吗？不急，先不用，CXF自带了一个轻量的容器服务，相当于spring自己提供了IOC容器一样。我们可以先用它来测试一下我们部署成功没。

直接来一个测试类：

Java代码

```
1. public static void main(String[] args) {
2.                 System.out.println("Server is starting...");
3.                 ReaderService readerService = new ReaderService();
4.                 Endpoint.publish("http://localhost:8080/readerService",readerService);
5.                 System.out.println("Server is started...");
6.         }
```

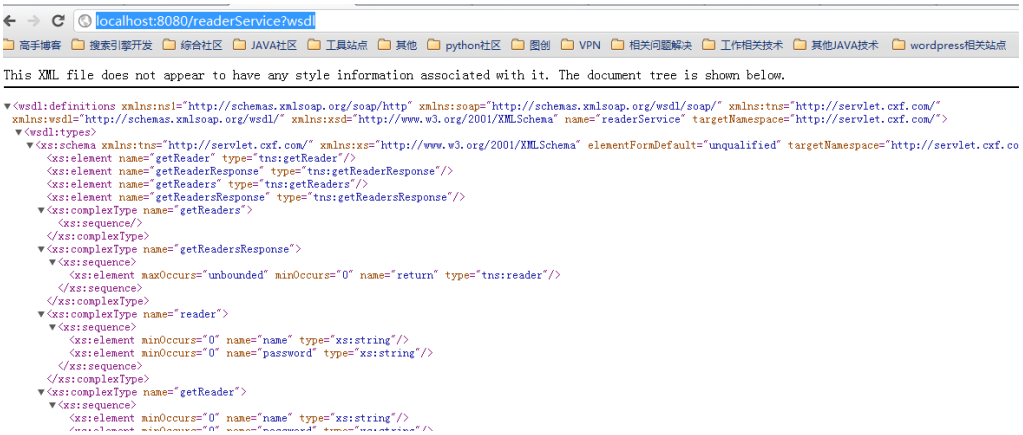
简单得不得了吧。直接publish地址，然后指定接口或类就OK了。我这里用的是类，但尽量用接口，毕竟面向接口编程才是真正的面对对象思想。

我们启动看看结果：

```
Server is starting...
2011-11-23 22:10:17 org.apache.cxf.service.factory.ReflectionServiceFactoryBean buildServiceFromClass
信息: Creating Service {http://servlet.cxf.com/}readerService from class com.cxf.servlet.IReaderService
2011-11-23 22:10:19 org.apache.cxf.endpoint.ServerImpl initDestination
信息: Setting the server's publish address to be http://localhost:8080/readerService
2011-11-23 22:10:19 org.eclipse.jetty.server.Server doStart
信息: jetty-7.5.3.v20111011
2011-11-23 22:10:19 org.eclipse.jetty.server.AbstractConnector doStart
信息: Started SelectChannelConnector@localhost:8080 STARTING
2011-11-23 22:10:19 org.eclipse.jetty.server.handler.ContextHandler startContext
信息: started o.e.j.s.h.ContextHandler{,null}
Server is started...
```

我们看到启动已经完成，接着启动浏览器看看是否成功了。

直接在浏览器输入http://localhost:8080/readerService?wsdl，我们可以看到：



它生成了我们所需要的wsdl文件，说明我们部署成功了。

3）部署成功后，我们就是要调用啦，它的调用也相当简单，跟xfire类似，取得接口，然后就可以跟本地类一样调用方法了。

Java代码

```
1. public static void main(String[] args) {
2.                 JaxWsProxyFactoryBean factoryBean = new JaxWsProxyFactoryBean();
3.                 factoryBean.setServiceClass(IReaderService.class);
4.                 factoryBean.setAddress("http://localhost:8080/readerService");
5.
6.                 IReaderService readerService = (IReaderService)factoryBean.create();
7.                 Reader reader = readerService.getReader("shun","123");
8.                 System.out.println("Reader:"+reader);
9.         }
```

这里很简单，也是取得一个工厂类，然后直接设接口和地址再create就可以得取相应的接口了，这里跟xfire一样，也是需要调用端先定义好接口原型，否则这些调用将无从说起。

我们运行得到结果：

```
<terminated> ReaderClient (3) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011-11-23 下午10:10:29)
2011-11-23 22:10:30 org.apache.cxf.service.factory.ReflectionServiceFactoryBean buildServiceFromClass
信息: Creating Service {http://servlet.cxf.com/}IReaderServiceService from class com.cxf.servlet.IReaderService
Reader:Name:shun,Password:123
```

没问题，跟我们预想的结果一致。

4）但很多情况下，我们并不希望我们的webservice和我们的应用分开两个服务器，而希望他们在同一个容器，tomcat或JBOSS或其他的，这样我们就必须通过WEB来部署我们前面完成的webservice。

注意，我们这里需要用到spring定义文件。

首先看看web.xml：

Java代码

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.          xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
```

```
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID"  version="3.0">

        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>WEB-INF/beans.xml</param-value>
        </context-param>

        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>

        <servlet>
                <servlet-name>CXFServlet</servlet-name>
                <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>CXFServlet</servlet-name>
                <url-pattern>/webservice/*</url-pattern>
        </servlet-mapping>
</web-app>
```

这里很简单，只是指定了spring的监听器和相应的配置文件路径，并且指定了CXF的拦截方式。

接下来看看beans.xml:

Java代码

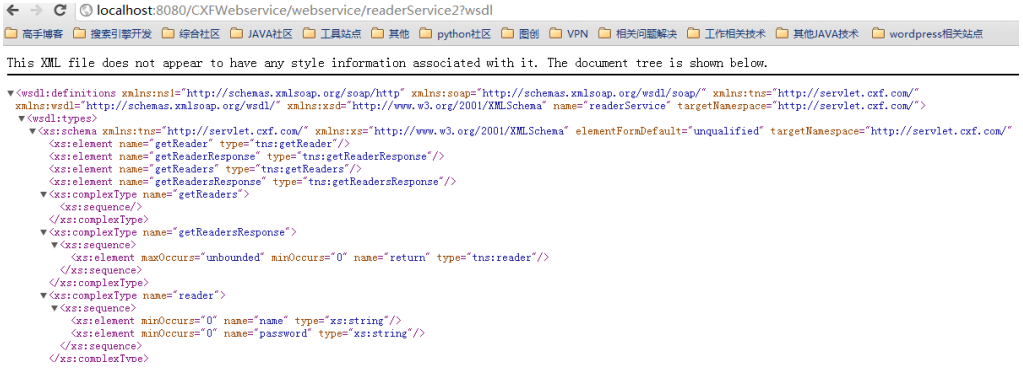
```
<?xml  version="1.0"  encoding="UTF-8"?>
<beans  xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jaxws="http://cxf.apache.org/jaxws"
        xsi:schemaLocation="
                http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://cxf.apache.org/jaxws
                http://cxf.apache.org/schemas/jaxws.xsd">
        <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"  />

        <jaxws:endpoint  id="readerServicce2"
                implementor="com.cxf.servlet.ReaderService"  address="/readerService2"  />
</beans>
```

这里很简单，只是通过jaxws:endpoint定义了一个webservice，implementor是webservice的处理类，而address是它的访问路径，跟我们前面写的readerService类似。

这时我们可以把它部署到tomcat中，通过http://localhost:8080/CXFWebservice/webservice/readerService2?wsdl可以直接访问。

有些朋友会问，为什么这次访问的URL跟前面的不一样呢。其实前面的访问地址是我们自己定义的，而这里的webservice地址是我们在配置文件中配置好的，并且是通过web项目来部署的，这里就需要用项目名称，而且我们在CXFServlet那里配置了url-pattern是webservice，所以最后的URL就跟上面一致了。

我们可以看到效果：



这证明我们部署成功了。

可以再次用前面的测试类测试一下，注意，需要把address修改成我们发布后的URL。

CXF相比xfire又更简洁了一些，虽然它增加了一些注解，但这些无伤大雅，它只是把以前的services.xml中的信息集中到类中，反而更方便维护，但这还是见仁见智的，有些人就喜欢配置文件，而有些人就不喜欢。另外CXF的调用方式更加简洁，比起xfire它的代码量更小了，是一个较大的进步。

有些朋友在搭建的过程中出现了一些问题，免去一个个回复了，这里放出代码，有需要的朋友可以下载看看。

lib目录下的所有包均没有放入，把cxf的所有包放入即可。

注：所用IDE为idea，文件结构跟eclipse不通用，如果需要在eclipse下使用的，可以直接复制代码和文件到eclipse新建的项目即可。