注:学习时候使用版本

NoteDate:2016-3-2

Idea:2016.1

CXF:3.1.6

Spring:4.2.6

JDk:1.7.0\_79

Tomcat:7.0.68

# (零)问题汇总:

1. CXF与Spring 整合的时候,初始化Spring上下文的时候遇到命名空间异常,然后查看官网的index页面的Use of Schemas and Namespaces页面查看所有命名空间.
2. CXF自带的wsdl2java方法解析.net发布的WebService的时候回遇到解析错误.具体异常如下: undefined element declaration 's:schema'.因为

遇到这种问题,可以将远端.net发布的WebService的wsdl文档保存到本地然后用<s:any minOccurs="2" maxOccurs="2"/>代替<s:element ref="s:schema" /><s:any />,然后再对修改后的本地wsdl文档进行解析.

# (一)CXF

由Celtix+Xfire演变而来,自身采用工厂设计模式,有ESB总线架构,依靠Spring完成模块的集成,实现SOA方式。能够灵活地构建和部署WebService,它支持多种传输协议,如SOAP1.1/1.2、XML+HTTP、Restful、JMS、CORBA(Common Object Request Architecture/早起语言使用的ws)

# (二)CXF配置

为了方便使用CXF的重要脚本功能.配置环境变量,把bin配置到path变量上.把lib下的cxf-manifest.jar配置到ClassPath上,ClassPath上配置的jar包等于配置了CXF工具使用的编译好的类。(CXF的wsdl2java工具与jdk的wsimport工具一样都可以生成本地客户端代码.不同的是wsdl2java可以生成soap1.1/1.2)

#### 通过wsimport生成客户端代码

通过jdk自带的wsimport生成客户端代码。进入$JAVA\_HOME/bin下，新建bin和src两个文件夹，执行以下命令：

**wsimport -d ./bin -s ./src -keep http://localhost:8080/services/layout?wsdl**

其中几个参数有以下几个，

-d:生成客户端执行类的class文件存放目录，

-s:生成客户端执行类的源文件存放目录，

-p:定义生成类的包名

 -verbose:显示生成过程

# (三)jar包说明

将jar包引入到项目中.由于CXF是ESB总线架构的所有自带的jar包比较多.然后CXF项目的lib目录下有个对所有包进行说明的文档WHICH\_JARS.可以针对不同的项目选择不同的不同的jar包

# (四)使用CXF发布WebService

CXF发布WebService与JaxWS [不同.CXF是要将@WebService注解添加到接口上.并且可以设置BindingType设置发布WebService](mailto:不同.CXF是要将@WebService注解添加到接口上.并且可以设置BindingType设置发布WebService)的协议类型.

发布步骤:

1. 编写服务的接口类并且通过注解来完善服务的描述. 例:

@WebService(targetNamespace = "http://service.learningPath.cn/",//包的地址结构会根据命名空间的命名倒写如生成客户端的时候的包的结构为cn.learningpath.service  
 name = "helloWorlName",//着就是暴露出去的服务的porttype的名称,也就是binding的类型名称  
 portName = "helloWorldPortName",  
 serviceName = "helloWorldServcieName")  
@BindingType(SOAPBinding.*SOAP12HTTP\_BINDING*)//指定发布服务的协议类型  
public interface HelloWordServiceInterface {  
 //接口中的方法的参数@WebParam注解是必须的,因为java接口不存储参数名称,如果不配置名字将会被命名为arg0  
 @WebResult(name = "sayHelloMessage") String sayHello(@WebParam(name = "name") String name);  
 @WebResult(name = "languageRaningMessage") String languageRaning(@WebParam(name = "ranking") int ranking);  
}

1. 编写实现类
2. 发布,CXF提供两个工厂类用于发布WebService,ServerFactoryBean和JaxWsServerFactoryBean两个类,其中ServerFactoryBean类不支持注解和拦截器.JaxWsServerFactoryBean类是ServerFactoryBean的子类,对ServerFactoryBean进行了功能扩展,运用JaxWsServerFactoryBean就必须在发布服务的接口上添加@WebService注解,如果不添加注解无法把服务暴露出去.并且暴露出去的WSDL文件更加规范,而且JaxWsServerFactoryBean类可以通过拦截器来拦截请求和返回信息.例:

HelloWordService ws = new HelloWordService();  
JaxWsServerFactoryBean bean = new JaxWsServerFactoryBean();  
bean.setAddress("http://127.0.0.1:9999/hello");  
bean.setServiceClass(HelloWordServiceInterface.class);//接口类  
bean.setServiceBean(ws);//服务实例  
bean.getInInterceptors().add(new LoggingInInterceptor()); //添加输入拦截器  
bean.getOutInterceptors().add(new LoggingOutInterceptor());//添加输出拦截器

bean.create();//默认启动的是jetty容器

# (五)使用CXF调用远端WebService.

CXf提供两个调用远端WebService的工厂类, JaxWsProxyFactoryBean与JaxWsDynamicClientFactory.其中JaxWsProxyFactory需要生成本地代理. JaxWsDynamicClientFactory可以不生成本地代理,只需要设置Webservice的wsdl发布地址.然后指定方法以及传入的参数就可以.

1.使用wsdl2java工具来解析远端wsdl文件使其生成本地代理

2.使用JaxWsProxyFactoryBean工厂类,并且设置好Address和ServiceClass来生成本地代理,就可调用远端服务的方法

JaxWsProxyFactoryBean jaxWsProxyFactoryBean = new JaxWsProxyFactoryBean();  
//address为远端服务wsdl发布地址.  
jaxWsProxyFactoryBean.setAddress("http://localhost:9843/WastePaperManagement/QualityControl/QCReportManagement/v001");  
//serviceClass为本地代理的接口类  
jaxWsProxyFactoryBean.setServiceClass(QCReportManagement.class);

1. 使用JaxWsDynamicClientFactory工厂类

JaxWsDynamicClientFactory clientFactory = JaxWsDynamicClientFactory.newInstance();  
Client client=clientFactory.createClient("http://localhost:8080/learningPath/webService/HelloWorldService?wsdl");  
Object[] result = client.invoke("sayHello","东莞");  
for(int i=0; i<result.length;i++) {  
 System.out.println(result[i]);  
}

# (六)CXF整合Spring发布WebService

1.在web.xml上启动Spring上下文

<listener>  
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>  
<context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:applicationContext.xml</param-value>  
</context-param>

1. 在web.xml启动CXF-servlet

<servlet>  
 <servlet-name>cxf</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>  
 <load-on-startup>1</load-on-startup>  
</servlet>  
<servlet-mapping>  
 <servlet-name>cxf</servlet-name>  
 <url-pattern>/webService/\*</url-pattern>  
</servlet-mapping>

1. 配置applicationContext.xml文件

<!--Spring与CXF整合，发布WebService  
 WebService发布地址=tomcat地址+CXF在Web.xml中servlet的路径+jaxws:server中的address-->  
<jaxws:server address="/HelloWorldService" serviceClass="study.CXF.HelloWorldServiceInterface">  
 <jaxws:serviceBean>  
 <ref bean="HelloWorldService"/>  
 </jaxws:serviceBean>  
 <jaxws:inInterceptors>  
 <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"></bean>  
 </jaxws:inInterceptors>  
 <jaxws:outInterceptors>  
 <bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"></bean>  
 </jaxws:outInterceptors>  
</jaxws:server>  
<!--import deployed WebService from public network-->  
<jaxws:client id="helloWorldClient" address="http://localhost:8080/learningPath/webService/HelloWorldService?wsdl"  
 serviceClass ="cn.learningpath.service.HelloWorldName">  
</jaxws:client>

1. 将项目部署到Tomcat上,访问localhost:8080/项目名/CXF-servlet拦截名来获取本项目发布的所有webService

# (七)Restful风格个人理解

注:RestFul个人理解: REST 是一种软件架构模式，只是一种风格，rest服务采用HTTP 做传输协议，REST 对于HTTP 的利用分为以下两种：资源定位和资源操作。

1.资源定位: 更加准确去定位一个互联网资源。使用url定位一个互联网资源。例如:查询一个学生信息url

不使用rest的url：http://ip:port/queryUser.action?userType=XXX&studentid=001&XXXX

使用 rest的url：<http://ip:port/user/student/001>

使用rest方式表达url更加简单地表达出这个url是用来查询学生信息的

2.资源操作: 利用HTTP 的GET、POST、PUT、DELETE 四种操作来表示数据库操作的SELECT、UPDATE、INSERT、DELETE 操作。

# (八)CXF发布Restful服务接口

注:发布接口的地址为,服务发布地址+接口Path+方法接口Path

编写返回实体类 使用下面注解将类转成XML.如果不添加转换注解的话,接口能够正常发布,但是返回信息报错.说返回信息为空,或者找不到信息类型例如:No message body writer has been found for class Exception

/\*使用@XmlRootElement注解讲POJO转成xml

Name中的名称将会成为Xml返回信息中的一个根节点\*/

@XmlRootElement(name = "User")

2.编写服务接口类

@WebService  
@Path("/user")  
public interface UserService {  
 @GET//HTTP Get Method  
 @Path("/query/{id}")//transfer id by url  
 @Produces(MediaType.*APPLICATION\_XML*)//set MediaType to XML  
 public User queryUser(@PathParam("id")long id);  
  
 @GET  
 @Path("/querylist/{type}")  
 @Produces({"application/json;charset=utf-8"})//set Nediatype to json  
 public List<User> queryUserList(@PathParam("type")String type);  
}

3.编写接口实现类

4.通过JAXRSServerFactoryBean工厂类对服务进行发布

public static void main(String[] args) {  
 JAXRSServerFactoryBean jaxrsServerFactoryBean = new JAXRSServerFactoryBean();  
 jaxrsServerFactoryBean.setAddress("http://127.0.0.1:12345/rest");  
 jaxrsServerFactoryBean.setServiceBean(new UserServiceImpl());  
 jaxrsServerFactoryBean.setServiceClass(UserServiceImpl.class);  
 jaxrsServerFactoryBean.create();  
}

# (九)CXF整合Spring发布Restful服务接口

<!--Spring与CXF整合,发布rest服务接口-->  
<!--引入自己编写好的rest服务接口的实现类-->  
<bean id="UserService" class="study.CXF.restful.UserServiceImpl"/>  
<!--发布Restful服务接口-->  
<jaxrs:server address="/rest">  
 <jaxrs:serviceBeans>  
 <ref bean="UserService"/>  
 </jaxrs:serviceBeans>  
</jaxrs:server>

# (十)CXF整合SpringBoot