## Webservice:

## 跨语言 跨平台的通信技术。

## Webservice的俩种类型:

1. 以soap(简单对象访问协议xml)协议风格
2. restful(json)风格

## CXF的webservice搭建:

服务端: 开发接口和实现类 并且发布服务

接口和实现:

@WebService

public interface HelloWorld {

public String sayHello(String name,int age);

}

public class HelloWorldImpl implements HelloWorld {

@Override

public String sayHello(String name, int age) {

System.out.println("客户端调用我了...");

return "你好:"+name+",年龄:"+age;

}

}

服务端发布启动:

public class PublishServer {

public static void main(String[] args) {

**JaxWsServerFactoryBean** jaxbean=new JaxWsServerFactoryBean();

//访问的地址

String address="http://127.0.0.1:0806/Hello";

jaxbean.setAddress(address);

//暴露的接口(实现类)

jaxbean.setServiceClass(HelloWorldImpl.class);

//发布服务

Server server=jaxbean.create();

server.start();

System.out.println("服务启动成功");

}

}

客户端调用:(测试是**在一个工程里**)

public static void main(String[] args) {

// 使用cxf代理对象工厂

**JaxWsProxyFactoryBean** proxy=new JaxWsProxyFactoryBean();

//设置访问的地址

proxy.setAddress("http://127.0.0.1:0806/Hello");

//设置要访问的接口(不是实现类)

proxy.setServiceClass(HelloWorld.class);

//得到接口对象

HelloWorld helloWorld=(HelloWorld) proxy.create();

//调用方法

System.out.println(helloWorld.sayHello("宋付双", 24));

}

**注意:**

1. 接口必须用@WebService标识
2. Soap协议:

Envelope元素:必有标记为soap消息,Header元素:可选头部信息

Body元素:必有调用响应信息,Fault元素:可选错误提示

Tcp/ip监听:

Local monitoring port：可无端访问的端口号;hostname：服务主机地址,port服务的端口号

Wsdl文件解析:

1. 配置cxf的jar的bin环境变量wsdl2java
2. 在cmd中执行Wsdl2java+服务地址?wsdl，
3. 根据生成的代码调用服务端,在另一个项目中调用

public static void main(String[] args) {

//根据生成的代码调用服务端获取对应的接口

HelloWorld hello=new HelloWorldImplService().getHelloWorldImplPort();

//调用接口的方法

System.out.println(hello.sayHello("从客户端过来的", 4));

}

列子:(天气预报)

## RESTFUL风格:

接口:

/\*\*

\* GET请求

\* @param id

\* @return

\*/

public Customer getCustomerById(String id);

/\*\*

\* POST请求

\* @param id

\* @return

\*/

public String addCustomer(Customer customer);

实现类:

@Path("/crm")

public class CustomerServiceImpl implements CustomerService {

/\*\*

\* 1、@Path：标注请求url

2、@GET、@POST、@PUT、@DELETE：标注操作的方法

3、@Produce：指定返回结果的数据类型，xml或者json等。

4.@Consumes("application/x-www-form-urlencoded")限制传过来参数的类型为名值对

5.在实体类中用@FormParam("age")标注如果SpringMVC就不用了

\*/

@Override

@GET

//@Produces("application/xml")

@Produces("application/json")

@Path("/getCustomerById/{id}")

public Customer getCustomerById(@PathParam("id") String id) {

Customer customer=new Customer(23, "宋付双", id);

return customer;

}

@Override

@POST

@Consumes("application/x-www-form-urlencoded")

@Path("/addCustomer")

public String addCustomer(@BeanParam Customer customer) {

System.out.println(customer);

return "OK";

}

}

发布：

public class Punlish {

public static void main(String[] args) {

JAXRSServerFactoryBean jrsfb=new JAXRSServerFactoryBean();

//发布的地址

jrsfb.setAddress("http://192.168.0.103:9999/CustomerServer");

//暴露的接口

jrsfb.setResourceClasses(CustomerServiceImpl.class);

//发布

jrsfb.create().start();

System.out.println("接口启动成功");

}

}

客户端调用(另一个电脑调用):

使用***httpclient***调用:

GET请求:

//1.CloseableHttpClient与HttpClient类似前者可控关闭

//创建HttpClient对象。

CloseableHttpClient httpClient=HttpClientBuilder.create().build();

//设置参数

String id="2";

//2. 创建请求方法的实例，并指定请求URL。如果需要发送GET请求，创建HttpGet对象；如果需要发送POST请求，创建HttpPost对象。

//开发时地址放到配置文件中

HttpGet httpGet=new HttpGet("http://192.168.0.103:9999/CustomerServer/crm/getCustomerById/"+id);

//4. 调用HttpClient对象的execute(HttpUriRequest request)发送请求，该方法返回一个HttpResponse。

HttpResponse response=httpClient.execute(httpGet);

//5.调用HttpResponse的getEntity()方法就是响应结果

//entity是io流对象

HttpEntity entity=response.getEntity();

//对得到后的结果处理判断状态信息HttpStatus.SC\_OK就是200

if(response.getStatusLine().getStatusCode()==HttpStatus.SC\_OK){

System.out.println(EntityUtils.toString(entity,"UTF-8"));

}

else{

System.out.println(response.getStatusLine().toString());

}

//6.关闭

EntityUtils.consume(entity);

httpClient.close();

}

POST请求:

//1.CloseableHttpClient与HttpClient类似前者可控关闭

//创建HttpClient对象。

CloseableHttpClient httpClient=HttpClientBuilder.create().build();

//2. 创建请求方法的实例，并指定请求URL。如果需要发送GET请求，创建HttpGet对象；如果需要发送POST请求，创建HttpPost对象。

//开发时地址放到配置文件中

HttpPost httpPost=new HttpPost("http://192.168.0.103:9999/CustomerServer/crm/addCustomer");

//设置参数

List<NameValuePair> list = new ArrayList<NameValuePair>();

list.add(new BasicNameValuePair("age", "27"));

list.add(new BasicNameValuePair("id", "6"));

list.add(new BasicNameValuePair("name", "何小雨"));

//解决乱码

HttpEntity params=new UrlEncodedFormEntity(list,Consts.UTF\_8);

httpPost.setEntity(params);

//4. 调用HttpClient对象的execute(HttpUriRequest request)发送请求，该方法返回一个HttpResponse。

HttpResponse response=httpClient.execute(httpPost);

//5.调用HttpResponse的getEntity()方法就是响应结果

//entity是io流对象

HttpEntity entity=response.getEntity();

//对得到后的结果处理判断状态信息HttpStatus.SC\_OK就是200

if(response.getStatusLine().getStatusCode()==HttpStatus.SC\_OK){

System.out.println(EntityUtils.toString(entity,"UTF-8"));

}

else{

System.out.println(response.getStatusLine().toString());

}

//6.关闭

EntityUtils.consume(entity);

httpClient.close();

}

注意参数的设置

## Spring整合:

Web.xml配置

<!--webservice专用-->

<servlet>

<display-name>SpringWebService</display-name>

<servlet-name>SpringWebService</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>SpringWebService</servlet-name>

<url-pattern>/services/\*</url-pattern>

</servlet-mapping>

Spring配置文件:

<?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"

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

xsi:schemaLocation="http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd

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

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

<!--xmlns:jaxws="http://cxf.apache.org/jaxws"对应soap就是wsdl-->

<!--xmlns:jaxrs="http://cxf.apache.org/jaxrs"对应restful风格-->

<!--soap风格-->

<jaxws:endpoint id="wsdlWebService" implementor="com.cn.project.webservice.impl.HelloWorldImpl" address="/wsdlWebServiceHello">

</jaxws:endpoint>

<!--restful风格-->

<jaxrs:server id="CustomerWebService" address="/CustService">

<jaxrs:serviceBeans>

<bean class="com.cn.project.service.impl.CustomerServiceImpl"></bean>

</jaxrs:serviceBeans>

</jaxrs:server>

</beans>

## 第二种:Cxf(包括jax-ws/rs):

接口:

@WebService

public interface CrmService {

public String getInfoById(int id);

public int getMessageByName(String name);

}

实现类:

@WebService(endpointInterface="com.cn.project.service.CrmService"

,serviceName="CrmWS" )

/\*

\* endpointInterface:可写可不写 是发布服务的接口

\* serviceName:可写可不写 名字随便起

\*/

public class CrmServiceImpl implements CrmService {

@Override

public String getInfoById(int id) {

System.out.println("B系统调用方法1...");

String str=null;

if(id>1){

str="你的号码为"+id;

}

return str;

}

@Override

public int getMessageByName(String name) {

System.out.println("B系统调用方法2...");

int a=0;

if(name.equals("song")&&name.equals("fu")){

a=1;

}

return a;

}

}

发布:

/\*\*

\* 一下俩种方式都对

\* @author Administrator

\*

\*/

public class publish {

public static void main(String[] args) {

String address="http://127.0.0.1:0806/Hello";

/\* JaxWsServerFactoryBean jaxbean=new JaxWsServerFactoryBean();

//访问的地址

jaxbean.setAddress(address);

//暴露的接口(实现类)

jaxbean.setServiceClass(CrmServiceImpl.class);

//发布服务

Server server=jaxbean.create();

server.start(); \*/

CrmService crm=new CrmServiceImpl();

Endpoint.publish(address,crm);

System.out.println("服务启动成功");

}

}

客户端调用:

1. 生成代码,找到那个类继承了Service类。
2. 调用getXXXXPort()无参数的方法.
3. 根据接口调用相应的方法.

WSDL文件解析:

targetNamespace:服务端package反向

Types:Schema文档

成对的message:

Operation：一个对应一个的方法

权限控制(带有用户名和密码)

一般用户名和密码在XML中的header标签中。

拦截器:看代码吧,很重要

## CXF与Spring整合: