1. **什么是RESTful Web Services**  
       RESTful web service是创建来能在web更好的运行的web service。REST是一种架构类型，指定了如统一的接口等应用于web service的约束。REST提供了如性能、可扩展性和可变性等特性，使得service能够更好的在web上工作。在REST框架中，数据和功能被认为是资源，是通过URI来访问的，通常是web链接。资源是通过使用一组简单的、定义良好的操作来生效。REST的架构方式限定了客户/服务器架构，是设计来使用无状态的通信协议的，通常是HTTP。在REST框架类型中，客户端和服务器使用标准的接口和协议交换资源的representation。  
       下面的原则使得RESTful的应用程序简单、轻量并快捷：  
   **通过URI确定资源**：一个RESTful的web service会公开一组资源，这组资源确定了和客户端互动的目标。资源是通过URI来确定的，URI为service和资源提供了全球的地址空间。更多的信息见第239页的“@Path注解和URI路径模版”。  
       **统一的接口**：资源是通过固定的操作PUT，GET，POST和DELETE来创建、读取、修改和删除的。PUT会创建一个新的资源，DELETE会删除一个资源。GET会获取资源的当前状态。POST将资源的状态转变成新的值。更多信息见底241页的“对HTTP资源的响应”。  
       **自描述的消息**：资源和它们的表现是解耦的，因此可以以不同的格式来访问，如HTML，XML，纯文本，PDF，JPEG，JSON等。关于资源的元数据是有效的并且用来完成控制缓存、检测传输错误、商讨合适的表现格式、执行身份验证和访问控制等。更多信息见第241页的“对HTTP资源的响应”和第243页的“使用实体提供者来映射HTTP response和request的实体段”。  
       **使用超链接的无状态的互动**：和资源的互动都是无状态的，也就是说，request消息是自包含的。无状态的互动是基于显示状态转换的概念的。如URI重写、cookies和隐藏的表单字段等技术是为了交换状态而存在的。状态可以被嵌套在response消息中来指向互动的将来有效的状态。更多信息见“使用实体提供者来映射HTTP response和request的实体段”和JAX-RS Overview文档中的“Building URIs”。  
   2.**创建RESTful的Web根资源类**  
       根资源类（Root resource classes）是由@Path注解标记的POJO或者至少有一个方法有@Path注解或请求方法指示符注解（@GET，@PUT，@POST或@DELETE）的POJO。  
       资源方法（resource method）是资源类中含有请求方法指示符注解的方法。  
       本节中将会解释怎样使用JAX-RS来注解Java类来创建RESTful的web service。  
   **使用JAX-RS来开发RESTful web service**  
       JAX-RS是一个Java编程语言接口，被设计用来简化使用REST架构的应用程序的开发。  
       JAX-RS API使用Java编程语言的注解来简化RESTful web service的开发。开发人员使用JAX-RS的注解修饰Java编程语言的类文件来定义资源和能够应用在资源上的行为。JAX-RS的注解是运行时的注解，因此运行时的映射会为资源生成辅助类和其他的辅助文件。包含JAX-RS资源类的Java EE应用程序中资源是被配置好的，辅助类和辅助文件是生成的，资源通过被发布到Java EE服务器上来公开给客户端。  
       下表列出了JAX-RS定义的一些Java注解以及怎样使用它们的简要的描述。更进一步的JAX-RS的API见[http://download.oracle.com/javaee/6/api](http://download.oracle.com/javaee/6/api" \t "http://blog.csdn.net/xiong9999/article/details/_blank)。

|  |  |
| --- | --- |
| 注解 | 描述 |
| @Path | @Path注解的值是一个相对的URI路径，这个路径指定了该[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/xiong9999/article/details/_blank)类的位置，例如/helloworld。  在这个URI中可以包含变量，例如可以获取用户的姓名然后作为参数传入URI中：/helloworld/{username}。 |
| @GET | @GET注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPGET请求。资源的行为由资源回应的HTTP方法决定。 |
| @POST | @POST注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPPOST请求。资源的行为由资源回应的HTTP方法决定。 |
| @PUT | @PUT注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPPUT请求。资源的行为由资源回应的HTTP方法决定。 |
| @DELETE | @DELETE注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPDELETE请求。  资源的行为由资源回应的HTTP方法决定。 |
| @HEAD | @HEAD注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPHEAD请求。资源的行为由资源回应的HTTP方法决定。 |
| @PathParam | @PathParam注解是可以抽取并用在资源类中的一类参数。URIpath参数是从请求的URI中抽取的，  而且参数的名称和@Path注解中定义的变量名对应。 |
| @QueryParam | @QueryParam注解是可以抽取并在资源类中使用的一类参数。Query参数是从请求URI的查询参数中抽取的。 |
| @Consumes | @Consumes注解是用来指定资源能够接受的客户发送的MIME媒体类型。 |
| @Produces | @Produces注解用来指定资源能够生成并发送给客户端的MIME媒体类型，例如“text/plain”. |
| @Provider | @Provider注解用在任何对JAX-RS运行时（如MessageBodyReader和MessageBodyWriter）有意义的事物上。  对HTTP请求，MessageBodyReader用来将HTTP请求实体段映射为方法参数。在响应的时候，  返回的值使用MessageBodyWriter来映射成HTTP响应实体段。如果应用程序需要提供其他的元数据，如HTTP头或不同的状态代码，方法可以返回一个打包了实体的Response，该Response可以使用Response.ResponseBuilder创建。 |

1. 添加所需的jar包（我这里用的是maven）

|  |
| --- |
| <!-- cxf-rs -->  <!-- https://mvnrepository.com/artifact/org.apache.cxf/cxf-rt-frontend-jaxrs -->  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-frontend-jaxrs</artifactId>  <version>3.1.6</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.cxf/cxf-rt-rs-client -->  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-rs-client</artifactId>  <version>3.1.6</version>  </dependency>  <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.jaxrs/jackson-jaxrs-json-provider -->  <dependency>  <groupId>com.fasterxml.jackson.jaxrs</groupId>  <artifactId>jackson-jaxrs-json-provider</artifactId>  <version>2.8.9</version>  </dependency> |

1. 在web.xml中添加cxf的servlet

|  |
| --- |
| <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>/rsWebService/\*</url-pattern>  </servlet-mapping> |

1. 创建服务发布类

|  |
| --- |
| **package** tt.rs.service;  **import** javax.ws.rs.GET;  **import** javax.ws.rs.Path;  **import** javax.ws.rs.Produces;  **import** javax.ws.rs.QueryParam;  /\*\*  \* 这里的@Path注解的值是一个相对的URI路径，这个路径指定了该Java类的位置，  \* 例如/helloworld。在这个URI中可以包含变量，例如可以获取用户的姓名然后作为参数传入URI中：/helloworld/{username}。  \*  \* **@Produces注解用来指定资源能够生成并发送给客户端的MIME媒体类型**，例如“text/plain”.  \* **@author** user  \*  \*/  @Path(value = "/say")  @Produces("text/plain;charset=UTF-8")  //@Produces({"application/json"})  **public** **interface** RestFulService {  /\*\*  \* 这里的@Path注解的值是一个相对的URI路径，这个路径指定了该Java类的位置，  \* 例如/helloworld。在这个URI中可以包含变量，例如可以获取用户的姓名然后作为参数传入URI中：/helloworld/{username}。  \*  \* **@GET**:@GET注解是请求方法指示符，这个指示符注解的Java方法会处理HTTPGET请求。资源的行为由资源回应的HTTP方法决定。  \* 四种媒体类型：  \* 1.text/plain:文本类型  \* 2.text/html:html类型  \* 3.application/json:json类型  \* 4.application/xml:xml类型  \*  \* **@QueryParam**:@QueryParam注解是可以抽取并在资源类中使用的一类参数。Query参数是从请求URI的查询参数中抽取的。  \*  \* **@param** name  \* **@return**  \*/  @Path(value = "/hello")  @GET  String sayHi(@QueryParam(value = "name") String name);  } |

注意：在使用@Produces指定媒体类型时，建议在后面加上;charset=UTF-8以防中文乱码

1. 实现类

|  |
| --- |
| **package** tt.rs.service.impl;  **import** tt.rs.service.RestFulService;  **public** **class** RestFulServiceImpl **implements** RestFulService {  @Override  **public** String sayHi(String name) {  String str = name + ",你好,我想对你说Hello World!";  **return** str;  }  } |

1. applicationContent-cxf.xml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:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd*  *http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd"*>  <bean id=*"helloWorldRest"* class=*"tt.rs.service.impl.RestFulServiceImpl"*/>  <jaxrs:server id=*"userWebCXFService"* address=*"/rs"*>  <jaxrs:serviceBeans>  <ref bean=*"helloWorldRest"*/>  </jaxrs:serviceBeans>    <jaxrs:providers>  <bean class=*"com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider"*/>  </jaxrs:providers>  </jaxrs:server>  </beans> |

1. 有这样一个业务场景，有两个系统，一个为学生管理系统，和学生信息查询系统，假设学生通过在学生信息查询系统的登录名，进行自己信息的查询
2. httpurlconnection的调用
3. 调用get方法

|  |
| --- |
| **package** tt.rs.test;  **import** java.io.ByteArrayOutputStream;  **import** java.io.InputStream;  **import** java.net.HttpURLConnection;  **import** java.net.URL;  **public** **class** ClientTest {  **public** **static** **void** main(String[] args) {  **try** {  URL url = **new** URL("http://localhost:8080/say/hello?name=zhangsan");    //打开连接  HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();  System.***out***.println(urlConnection.getResponseCode());    **if**(200 == urlConnection.getResponseCode()){  //得到输入流  InputStream is = urlConnection.getInputStream();  ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  **byte**[] buffer = **new** **byte**[1024];  **int** len = 0;  **while**(-1 != (len = is.read(buffer))){  baos.write(buffer,0,len);  baos.flush();  }  System.***out***.println(baos.toString("utf-8"));  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

1. 调用post方法

|  |
| --- |
| **package** tt.rs.test;  **import** java.io.BufferedInputStream;  **import** java.io.ByteArrayOutputStream;  **import** java.io.PrintWriter;  **import** java.net.HttpURLConnection;  **import** java.net.URL;  **public** **class** ClientPostTest {  **public** **static** **void** main(String[] args) {  URL url = **null**;    **try** {  url = **new** URL("http://localhost:8080/select/getName");  HttpURLConnection httpUrlConnection = (HttpURLConnection) url.openConnection();  httpUrlConnection.setRequestMethod("POST"); //提交方式  // httpUrlConnection.setConnectTimeout(10000);//连接超时 单位毫秒  // httpUrlConnection.setReadTimeout(2000);//读取超时 单位毫秒  // 发送POST请求必须设置如下两行  httpUrlConnection.setDoOutput(**true**);  httpUrlConnection.setDoInput(**true**);    // 获取URLConnection对象对应的输出流  PrintWriter printWriter = **new** PrintWriter(httpUrlConnection.getOutputStream());  // 发送请求参数  String post = "{\"name\":\"zhangsan\"}";  printWriter.write(post);//post的参数 xx=xx&yy=yy  // flush输出流的缓冲  printWriter.flush();  //开始获取数据  BufferedInputStream bis = **new** BufferedInputStream(httpUrlConnection.getInputStream());  ByteArrayOutputStream bos = **new** ByteArrayOutputStream();  **int** len;  **byte**[] arr = **new** **byte**[1024];  **while**((len=bis.read(arr))!= -1){  bos.write(arr,0,len);  bos.flush();  }  bos.close();    System.***out***.println(bos.toString("utf-8"));      } **catch** (Exception e) {  e.printStackTrace();  }      }  } |