**WebService是一种跨编程语言和跨操作系统平台的远程调用技术**

SOA ，即Service Oriented Architecture ，SOA的基础是web service

Web service实现跨应用不同语言或相同语言程序进行通信和跨平台进行通信

在Web 服务的体系结构中，涉及到三个角色，

一个是 Web 服务提供者，一个是 Web 服务中介者，还有一个就是 Web 服务请求者，

同时还涉及到三类动作，即发布，查找，绑定，

**Web** **服务提供者：**

可以发布 Web 服务，并且对使用自身服务的请求进行响应，

Web 服务的拥有者，它会等待其他的服务或者是应用程序访问自己。

**Web** **服务请求者：**

也就是 Web 服务功能的使用者，它通过服务注册中心也就是 Web 服务中介者查找到所需要的服务，

再利用 SOAP 消息向 Web 服务提供者发送请求以获得服务。

**Web** **服务中介者：**

也称为服务代理，用来注册已经发布的 Web服务提供者，并对其进行分类，同时提供搜索服务，

简单来说的话，Web 服务中介者的作用就是把一个 Web 服务请求者和合适的 Web 服务提供者联系在一起，

充当一个管理者的角色，一般是通过 UDDI来实现。

**发布：**

通过发布操作，可以使 Web服务提供者向 Web 服务中介者注册自己的功能以及访问的接口。

**发现（查找）：**

使得 Web 服务请求者可以通过 Web 服务中介者来查找到特点的种类的 Web 服务。

**绑定：**

这里就是实现让服务请求者能够使用服务提供者提供的服务了。

**WebServices三种基本元素之 SOAP**

**SOAP** **即 Simple Object AccessProtocol** **也就是简单对象访问协议。**

是一种用于访问 Web 服务的协议。

因为 SOAP 基于XML 和 HTTP ，其通过XML 来实现消息描述，然后再通过 HTTP 实现消息传输。

SOAP 是用于在应用程序之间进行通信的一种通信协议。

**WebServices三种基本元素之 WSDL**

WSDL 即Web Services Description Language也就是 Web 服务描述语言。

**是基于 XML的用于描述 Web 服务以及如何访问 Web 服务的语言。**

**WSDL 描述了 Web服务的三个基本属性：**

（1）服务所提供的操作

（2）如何访问服务

（3）服务位于何处（通过 URL 来确定就 OK 了）

**WebServices三种基本元素之 UDDI**

UDDI 即 Universal Description，Discovery and Integration，也就是通用的描述，发现以及整合。

WSDL 呢，用来描述了访问特定的 Web 服务的一些相关的信息，可以在互联网上，

或者是在企业的不同部门之间，如何来发现我们所需要的 Web 服务呢？

而 Web 服务提供商又如何将自己开发的 Web 服务公布到因特网上，

这就需要使用到 UDDI 了，UDDI的话，是一个跨产业，跨平台的开放性架构，

可以帮助 Web 服务提供商在互联网上发布 Web 服务的信息。

UDDI 呢是一种目录服务，企业可以通过 UDDI 来注册和搜索 Web 服务。

简单来时候话，UDDI 就是一个目录，只不过在这个目录中存放的是一些关于 Web 服务的信息而已。

Soap

一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

* 必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息
* 可选的 Header 元素，包含头部信息
* 必需的 Body 元素，包含所有的调用和响应信息
* 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

这里是一些重要的语法规则：

* SOAP 消息必须用 XML 来编码
* SOAP 消息必须使用 SOAP Envelope 命名空间
* SOAP 消息必须使用 SOAP Encoding 命名空间
* SOAP 消息不能包含 DTD 引用
* SOAP 消息不能包含 XML 处理指令

可选的 SOAP Header 元素可包含有关 SOAP 消息的应用程序专用信息（比如认证、支付等）。如果 Header 元素被提供，则它必须是 Envelope 元素的第一个子元素。

必需的 SOAP Body 元素可包含打算传送到消息最终端点的实际 SOAP 消息。

可选的 SOAP Fault 元素用于指示错误消息。

如果已提供了 Fault 元素，则它必须是 Body 元素的子元素。在一条 SOAP 消息中，Fault 元素只能出现一次。

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>

...

...

</soap:Header>

<soap:Body>

...

...

<soap:Fault>

...

...

</soap:Fault>

</soap:Body>

</soap:Envelope>

### SOAP 请求：

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">

   <m:GetStockPrice>

<m:StockName>IBM</m:StockName>

    </m:GetStockPrice>

</soap:Body>

</soap:Envelope>

### SOAP 响应：

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">

<m:GetStockPriceResponse>

<m:Price>34.5</m:Price>

</m:GetStockPriceResponse>

</soap:Body>

</soap:Envelope>

WSDL

一个 WSDL 文档的主要结构是类似这样的：

<definitions>

<types>

definition of types........

</types>

<message>

definition of a message....

</message>

<portType>

definition of a port.......

</portType>

<binding>

definition of a binding....

</binding>

</definitions>

WSDL 文档可包含其它的元素，比如 extension 元素，以及一个 service 元素，此元素可把若干个 web services 的定义组合在一个单一的 WSDL 文档中。

**WSDL 端口**

**<portType>** 元素是最重要的 WSDL 元素。

它可描述一个 web service、可被执行的操作，以及相关的消息。

可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）。

**WSDL 消息**

**<message>** 元素定义一个操作的数据元素。

每个消息均由一个或多个部件组成。可以把这些部件比作传统编程语言中一个函数调用的参数。

**WSDL types**

**<types>** 元素定义 web service 使用的数据类型。

为了最大程度的平台中立性，WSDL 使用 XML Schema 语法来定义数据类型。

**WSDL Bindings**

**<binding>** 元素为每个端口定义消息格式和协议细节。

**WSDL 实例**

这是某个 WSDL 文档的简化的片段：

<message name="getTermRequest">

<part name="term" type="xs:string"/>

</message>

<message name="getTermResponse">

<part name="value" type="xs:string"/>

</message>

<portType name="glossaryTerms">

<operation name="getTerm">

<input message="getTermRequest"/>

<output message="getTermResponse"/>

</operation>

</portType>

在这个例子中，**<portType>** 元素把 "glossaryTerms" 定义为某个**端口**的名称，把 "getTerm" 定义为某个**操作**的名称。

操作 "getTerm" 拥有一个名为 "getTermRequest" 的**输入消息**，以及一个名为 "getTermResponse" 的**输出消息**。

**<message>** 元素可定义每个消息的**部件**，以及相关联的数据类型。

对比传统的编程，glossaryTerms 是一个函数库，而 "getTerm" 是带有输入参数 "getTermRequest" 和返回参数 getTermResponse 的一个函数。

**操作类型**

请求-响应是最普通的操作类型，不过 WSDL 定义了四种类型：

|  |  |
| --- | --- |
| **类型** | **定义** |
| One-way | 此操作可接受消息，但不会返回响应。 |
| Request-response | 此操作可接受一个请求并会返回一个响应 |
| Solicit-response | 此操作可发送一个请求，并会等待一个响应。 |
| Notification | 此操作可发送一条消息，但不会等待响应。 |

指的是模式，并不是具体标签，是WSDL的交互模式

UUID

假如行业发布了一个用于航班比率检测和预订的 UDDI 标准，航空公司就可以把它们的服务注册到一个 UDDI 目录中。然后旅行社就能够搜索这个 UDDI 目录以找到航空公司预订界面。当此界面被找到后，旅行社就能够立即与此服务进行通信，这样由于它使用了一套定义良好的预订界面。

**谁在支持 UDDI？**

UDDI 是一个跨行业的研究项目，由所有主要的平台和软件提供商驱动，比如：Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, 以及 Sun, 它既是一个市场经营者的团体，也是一个电子商务的领导者。

已有数百家公司参与了这个 UDDI 团体。

Restful

REST是一种架构风格，其核心是面向资源，REST专门针对网络应用设计和开发方式，以降低开发的复杂性，提高系统的可伸缩性。REST提出设计概念和准则为：

1.网络上的所有事物都可以被抽象为资源(resource)

2.每一个资源都有唯一的资源标识(resource identifier)，对资源的操作不会改变这些标识

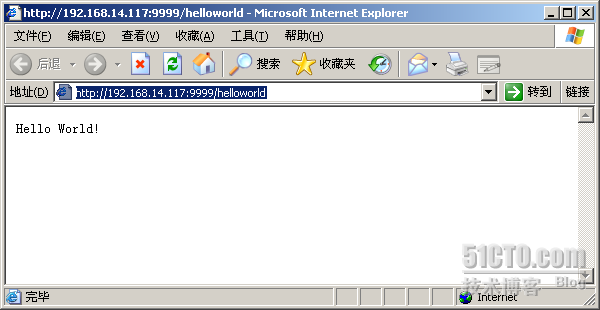
3.所有的操作都是无状态的

REST简化开发，其架构遵循CRUD原则，该原则告诉我们对于资源(包括网络资源)只需要四种行为：创建，获取，更新和删除就可以完成相关的操作和处理。您可以通过统一资源标识符（Universal Resource Identifier，URI）来识别和定位资源，并且针对这些资源而执行的操作是通过 HTTP 规范定义的。其核心操作只有GET,PUT,POST,DELETE。

对于SOAP Webservice和Restful Webservice的选择问题，首先需要理解就是SOAP偏向于面向活动，有严格的规范和标准，包括安全，事务等各个方面的内容，同时SOAP强调操作方法和操作对象的分离，有WSDL文件规范和XSD文件分别对其定义。而REST强调面向资源，只要我们要操作的对象可以抽象为资源即可以使用REST架构风格。

如果从这个意义上讲，是否使用REST就需要考虑资源本身的抽象和识别是否困难，如果本身就是简单的类似增删改查的业务操作，那么抽象资源就比较容易，而对于复杂的业务活动抽象资源并不是一个简单的事情。比如校验用户等级，转账，事务处理等，这些往往并不容易简单的抽象为资源。

/\*\*   
\* RESTful WebService入门   
\* @author leizhimin 2009-11-18 16:42:43   
\*/   
package example;   
import com.sun.jersey.api.container.httpserver.HttpServerFactory;   
import com.sun.net.httpserver.HttpServer;   
import javax.ws.rs.GET;   
import javax.ws.rs.Path;   
import javax.ws.rs.Produces;   
import java.io.IOException;   
  
//指定URI   
@Path("/helloworld")   
public class HelloWorld {   
        //处理HTTP的GET请求   
        @GET   
        // 处理请求反馈的内容格式为"text/plain"   
        @Produces("text/plain")   
        public String getClichedMessage() {   
                return "Hello World!";   
        }   
  
        public static void main(String[] args) throws IOException {   
                //创建RESTful WebService服务   
                HttpServer server = HttpServerFactory.create("http://192.168.14.117:9999/");   
                //启动服务，这会导致新开一个线程   
                server.start();   
                //输出服务的一些提示信息到控制台   
                System.out.println("RESTful WebService服务已经启动");   
                System.out.println("服务访问地址: http://192.168.14.117:9999/helloworld");   
        }   
}



下载完成后解压并找到lib目录，将其中的jar文件添加你的[**Java**](http://lib.csdn.net/base/17)项目中，接下来就可以开始编写你的Web Service程序了。话不多说，直接上代码。

package com.lovo.domain;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "Student")

public class Student {

private Integer id;

private String name;

private String birthday;

public Student() {

}

public Student(Integer id, String name, String birthday) {

this.id = id;

this.name = name;

this.birthday = birthday;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

}

package com.lovo.infrastructure;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.lovo.domain.Student;

public class StudentRepository {

private Map<Integer, Student> map = new HashMap<>();

public StudentRepository() {

map.put(1001, new Student(1001, "骆昊", "1980-11-28"));

map.put(1002, new Student(1002, "王大锤", "1992-2-2"));

map.put(1003, new Student(1003, "张三丰", "1930-3-3"));

}

public void save(Student student) {

if (student != null) {

map.put(student.getId(), student);

}

}

public void delete(Student student) {

if (student != null && map.containsKey(student.getId())) {

map.remove(student.getId());

}

}

public void update(Student student) {

delete(student);

save(student);

}

public Student findById(Integer id) {

return map.get(id);

}

public List<Student> findAll() {

return new ArrayList<Student>(map.values());

}

}

package com.lovo.service;

import java.util.List;

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 com.lovo.domain.Student;

import com.lovo.infrastructure.StudentRepository;

@Path("/service")

@Produces("application/json")

public class StudentService {

private StudentRepository studentRepo = new StudentRepository();

@GET

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

@Consumes("application/json")

public Student getStudent(@PathParam("id") Integer id) {

return studentRepo.findById(id);

}

@GET

@Path("/stu")

@Consumes("application/json")

public List<Student> getAllStudents() {

return studentRepo.findAll();

}

@POST

@Path("/stu")

@Consumes("application/json")

public boolean addStudent(Student student) {

if (getStudent(student.getId()) == null) {

studentRepo.save(student);

return true;

}

return false;

}

@PUT

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

@Consumes("application/json")

public boolean updateStudent(@PathParam("id") Integer id, Student student) {

if (getStudent(id) != null) {

studentRepo.update(student);

return true;

}

return false;

}

@DELETE

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

@Consumes("application/json")

public boolean deleteStudent(@PathParam("id") Integer id) {

Student student = getStudent(id);

if (student != null) {

studentRepo.delete(student);

return true;

}

return false;

}

}

  最后来启动Web Service的服务器并进行测试。

package com.lovo;

import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;

import com.lovo.domain.Student;

import com.lovo.service.StudentService;

public class MyRestServer {

public static void main(String[] args) {

JAXRSServerFactoryBean myRESTfulServer = new JAXRSServerFactoryBean();

myRESTfulServer.setResourceClasses(Student.class);

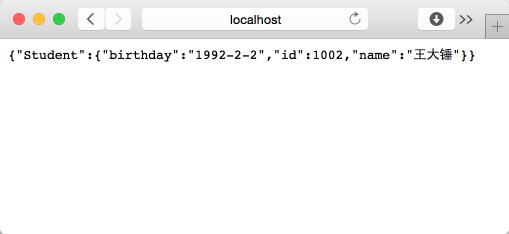
myRESTfulServer.setServiceBean(new StudentService());

myRESTfulServer.setAddress("http://localhost:9999/");

myRESTfulServer.create();

}

}

  在浏览器中分别输入以下两个URI查看结果：   
<http://localhost:9999/service/stu/1002>   


<http://localhost:9999/service/stu>