Spring MVC-REST

简介

我们一般请求的url是如下： http://...../xxx.action?id=001&type=aaa 而REST的url风格是什么样子呢？一般它类似于： http://..../xxx/001 所以REST有个很明显的特点：使url变得简洁，将参数通过url传到服务端。

什么是REST?

REST：即 Representational State Transfer。（资源）表现层状态转化。

是目前 最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便， 所以正得到越来越多网站的采用 • 资源（Resources）：网络上的一个实体，或者说是网络上的一个具体信息。它 可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的存在。 可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的 URI 。要 获取这个资源，访问它的URI就可以，因此 URI 即为每一个资源的独一无二的识 别符。 **• 表现层（Representation）**：把资源具体呈现出来的形式，叫做它的表现层 （Representation）。比如，文本可以用 txt 格式表现，也可以用 HTML 格 式、XML 格式、JSON 格式表现，甚至可以采用二进制格式。 **• 状态转化（State Transfer）**：每发出一个请求，就代表了客户端和服务器的一 次交互过程。HTTP协议，是一个无状态协议，即所有的状态都保存在服务器 端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生“ 状态转化”（State Transfer）。而这种转化是建立在表现层之上的，所以就是 “ 表现层状态转化”。具体说，就是 HTTP 协议里面，四个表示操作方式的动 词：GET、POST、PUT、DELETE。

它们分别对应四种基本操作：GET 用来获 取资源，POST 用来新建资源，PUT 用来更新资源，DELETE 用来删除资源。

是目前最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便， 所以正得到越来越多网站的采用。

什么是REST FUL?

一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

• 需求示例：

/某路径/1 HTTP GET ： 得到 id = 1 的 user **/某路径/1** HTTP DELETE：删除 id = 1的 user **/某路径** HTTP PUT ： 更新查询id = 1的 user，进行PUT请求更新 **/某路径** HTTP POST ： 新增 user

http://localhost:8080/pro/user/1 GET

http://localhost:8080/pro/user/1 DELETE

http://localhost:8080/pro/user/1 PUT

http://localhost:8080/pro/user/ POST

@PathVariable 注解：映射URL绑定

注解：映射 URL 绑定的占位符

带占位符的 URL 是 Spring3.0 新增的功能，该功能在 SpringMVC 向 REST 目标挺进发展过程中具有里程碑的 意义

通过 @PathVariable 可以将 URL 中占位符参数绑定到控 制器处理方法的入参中：URL 中的 {xxx} 占位符可以通过 @PathVariable("xxx") 绑定到操作方法的入参中。

```
@RequestMapping("/delete/{id}")
public String delete(@PathVariable("id") Integer id){
    UserDao.delete(id);
    return "redirect:/user/list.action";
}
```

REST FUL风格开发示例

浏览器 form 表单只支持 GET 与 POST 请求，而DELETE、PUT 等 method 并不支 持，

Spring3.0 添加了一个过滤器**HiddenHttpMethodFilter**，可以将这些请求转换 为标准的 http 方法，使得支持 GET、POST、PUT 与 DELETE 请求。

RestFul风格web.xml配置

```
<filter>
    <filter-name>HiddenHttpMethod</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HiddenHttpMethod</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

SpringMVC给我们提供了这样一个Filter：HiddenHttpMethodFilter可以将post请求转化为put和delete请求！其工作原理是在发送请求时拦截请求，获取一个名为*“method”代表的是请求方式的参数，如果检测到请求方式是post，然后再检测表单是否有代表* _method的隐藏域的参数，然后再将其转化为put或者delete

Controller层中编写

```

```

View页面中

查询功能 /某路径/1

HTTP GET： 得到 id = 1 的 user

```

```

删除功能

/某路径/1 HTTP DELETE：删除 id = 1的 user

```
<a href="javascript:del(${u.id})">删除</a>

<form action="user" method="post" name="delForm">
    <input type="hidden" name="_method" value="DELETE">
</form>

<script type="text/javascript">
    function del(id){
        alert(id);
        document.delForm.action="user/"+id;
        document.delForm.submit();
    }
</script>
```

修改功能

HTTP PUT： 更新查询id = 1的 user，进行PUT请求更新

```
@RequestMapping(value="update/{id}",method=RequestMethod.GET) // user/update/id
public String update(@PathVariable("id")int id,Map<String,Object> map){
    User user=userService.selectById(id);
    map.put("user", user);
    return "/update.jsp";
}
```

修改页面JSP：

```
<form action="user/" method="post">
    <input name="_method" value="PUT">
    <tr>
        .........
        <td>
            <input type="submit" value="修改">
        </td>
    </tr>
</form>
```

进行PUT请求更新 user/

```
@RequestMapping(value="/",method=RequestMethod.PUT)
public String updateTo(Map<String,Object> map,User u){
    System.out.println("传递的: "+u);
    return "redirect:/user/list";
}
```

添加功能

HTTP POST： 新增 order

处理静态资源

问题:

优雅的 REST 风格的资源URL 不希望带 .html 或 .do 等后缀

若将 DispatcherServlet 请求映射配置为 /，则 Spring MVC 将捕获 WEB 容器的所有请求，包括静态资源的请求，SpringMVC 会将他 们当成一个普通请求处理，因找不到对应处理器将导致错误。

解决

可以在 SpringMVC 的配置文件中配置 <mvc:default-servlet- handler/> 的方式解决静态资源的问题:

– mvc:default-servlet-handler/ 将在 SpringMVC 上下文中定义一个 DefaultServletHttpRequestHandler，它会对进入 DispatcherServlet 的 请求进行筛查，如果发现是没有经过映射的请求，就将该请求交由 WEB 应用服务器默认的 Servlet 处理，如果不是静态资源的请求，才由 DispatcherServlet 继续处理

– 一般 WEB 应用服务器默认的 Servlet 的名称都是 default。若所使用的 WEB 服务器的默认 Servlet 名称不是 default，则需要通过 default- servlet-name 属性显式指定

注意:

mvc:annotation-driven/ 是告知Spring，启用注解驱动。然后Spring会自动为我们注册上面说到多个Bean到工厂中，来处理我们的请求。

主要有两个:

RequestMappingHandlerMapping RequestMappingHandlerAdapter 第一个是HandlerMapping的实现类，它会处理@RequestMapping 注解，并将其注册到请求映射表中。 第二个是HandlerAdapter的实现类，它是处理请求的适配器，就是确定调用哪个类的哪个方法，并且构造方法参数，返回值。 --支持使用 **@RequestBody 和 @ResponseBody 注解**

最佳化实践

为了便于配置文件管理。在很多配置中一般都会进行分开配置，这种配置就像各施其职一样，显得特别清晰。

spring_core.xml

```
<?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:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:task="http://www.springframework.org/schema/task"
    xmlns:redis="http://www.springframework.org/schema/redis"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task-4.3.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
        http://www.springframework.org/schema/redis
http://www.springframework.org/schema/redis/spring-redis-1.0.xsd
        http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd">


    <!-- Spring IOC容器 扫描 base-package   url-pattren=**/*.class
     注解: @Repository @Service @Controller  @Compont
     com.qfjy.project.weixin.servlet

     作用: SpringIOC容器扫描 创建对象。
     约定大于配置。

  -->

    <context:component-scan   base-package="com.qfjy"
        use-default-filters="true">
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>

    </context:component-scan>
    <!-- AOP配置    -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>


    <context:property-placeholder location="classpath:jdbc.properties" />
```

```
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
        init-method="init" destroy-method="close">
        <property name="driverClassName" value="${jdbc_driver}" />
        <property name="url" value="${jdbc_url}" />
        <property name="username" value="${jdbc_user}" />
        <property name="password" value="${jdbc_password}" />
        <!-- 配置初始化大小、最小、最大 -->
        <property name="initialSize" value="1" />
        <property name="minIdle" value="1" />
        <property name="maxActive" value="10" />

        <!-- 配置获取连接等待超时的时间 -->
        <property name="maxWait" value="10000" />

        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />

        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name="minEvictableIdleTimeMillis" value="300000" />

        <property name="testWhileIdle" value="true" />

        <!-- 这里建议配置为TRUE，防止取到的连接不可用 -->
        <property name="testOnBorrow" value="true" />
        <property name="testOnReturn" value="false" />

        <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->
        <property name="poolPreparedStatements" value="true" />
        <property name="maxPoolPreparedStatementPerConnectionSize"
            value="20" />
        <!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->
        <property name="defaultAutoCommit" value="true" />
        <!-- 验证连接有效与否的SQL，不同的数据配置不同 -->
        <property name="validationQuery" value="select 1 " />

        <!-- 配置监控统计拦截的filters -->
        <property name="filters" value="stat" />

    </bean>

    <!--2 Spring JDBC模版 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

</beans>
```

spring_mvc.xml

```
    <!-- base-package:用于扫描的包 resource-pattern:扫描的目录，默认值 **/*.calss 默认是级联，（当前包、和所有子包.）
        use-default-filters 默认为true @Component, @Repository, @Service, @Controller
        作用：Spring容器来管理这些对象（对象的创建） -->
    <context:component-scan base-package="com.qfjy.web.controller"
    use-default-filters="false" >

    <context:include-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>

    </context:component-scan>

    <mvc:default-servlet-handler />
    <mvc:annotation-driven />
```

web.xml

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

完整的web.xml

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

<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
```

```
</filter-mapping>

<servlet>
    <servlet-name>SpringMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring_mvc.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>SpringMVC</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- 浏览器 form 表单只支持 GET 与 POST 请求，而DELETE、PUT 等 method 并不支 持，需要让其配置支持REST
风格 -->
<filter>
    <filter-name>HiddenHttpMethod</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HiddenHttpMethod</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>


<!-- druid 监控 -->
<servlet>
    <servlet-name>DruidStatView</servlet-name>
    <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DruidStatView</servlet-name>
    <url-pattern>/druid/*</url-pattern>
</servlet-mapping>
```