请求处理

1、请求参数处理

数组接收参数

List集合接收参数

1、SpringMVC对请求参数的处理

在之前的servlet中我们可以通过request.getParameter()来获取请求中的参数，但是在我们编写的SpringMVC的应用程序中，在具体请求的方法中并不包含request参数，那么我们应该如何获取请求中的参数呢?

需要使用以下几个注解：

@RequestParam：获取请求的参数

@RequestHeader：获取请求头信息

@CookieValue：获取cookie中的值

@RequestParam的基本使用

```
1 package cn.tulingxueyuan.controller;
2
```

```
3 import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class RequestController {

    /**
     * 如何获取SpringMVC中请求中的信息
     * 默认情况下，可以直接在方法的参数中填写跟请求一样的名称，此时会默认接受参数
     *      如果有值，直接赋值，如果没有，那么直接给空值
     *
     * @RequestParam:获取请求中的参数值,使用此注解之后，参数的名称不需要跟请求的名称一致，但是必
     *      public String request(@RequestParam("user") String username){
     *
     *      此注解还包含三个参数：
     *      value:表示要获取的参数值
     *      required：表示此参数是否必须，默认是true，如果不写参数那么会报错，如果值为false，那么
     *      defaultValue:如果在使用的时候没有传递参数，那么定义默认值即可
     *
     *
     * @param username
     * @return
     */
    @RequestMapping("/request")
    public String request(@RequestParam(value = "user",required = false,defaultValue = "he
        System.out.println(username);
        return "success";
    }
}
```

@RequestHeader的基本使用：

```
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import sun.management.resources.agent;
```

```
8
9  @Controller
10 public class RequestController {
11
12     /**
13      * 如果需要获取请求头信息该如何处理呢？
14      * 可以使用@RequestHeader注解，
15      *      public String header(@RequestHeader("User-Agent") String agent){
16      *      相当于 request.getHeader("User-Agent")
17      *
18      *      如果要获取请求头中没有的信息，那么此时会报错，同样，此注解中也包含三个参数,跟@Request
19      *          value
20      *          required
21      *          defalutValue
22      * @param agent
23      * @return
24      */
25     @RequestMapping("/header")
26     public String header(@RequestHeader("User-Agent") String agent){
27         System.out.println(agent);
28         return "success";
29     }
30 }
```

@CookieValue的基本使用

```
1  package cn.tulingxueyuan.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.CookieValue;
5  import org.springframework.web.bind.annotation.RequestHeader;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestParam;
8  import sun.management.resources.agent;
9
10 @Controller
11 public class RequestController {
12
13     /**
```

```
     * 如果需要获取cookie信息该如何处理呢？
     * 可以使用@CookieValue注解，
     *      public String cookie(@CookieValue("JSESSIONID") String id){
     *      相当于
     *      Cookie[] cookies = request.getCookies();
     *      for(Cookie cookie : cookies){
     *          cookie.getValue();
     *      }
     *      如果要获取cookie中没有的信息，那么此时会报错，同样，此注解中也包含三个参数,跟@Request
     *          value
     *          required
     *          defalutValue
     * @param id
     * @return
     */
    @RequestMapping("/cookie")
    public String cookie(@CookieValue("JSESSIONID") String id){
        System.out.println(id);
        return "success";
    }
}
```

复杂数据类型处理

JavaBean数据绑定

User.java

```
package cn.tulingxueyuan.bean;

import java.util.Date;

public class User {
    private Integer id;
    private String name;
    private Integer age;
    private Date date;
    private Address address;

    public Integer getId() {
        return id;
    }
```

```
15
16      public void setId(Integer id) {
17          this.id = id;
18      }
19
20      public String getName() {
21          return name;
22      }
23
24      public void setName(String name) {
25          this.name = name;
26      }
27
28      public Integer getAge() {
29          return age;
30      }
31
32      public void setAge(Integer age) {
33          this.age = age;
34      }
35
36      public Date getDate() {
37          return date;
38      }
39
40      public void setDate(Date date) {
41          this.date = date;
42      }
43
44      public Address getAddress() {
45          return address;
46      }
47
48      public void setAddress(Address address) {
49          this.address = address;
50      }
51
52      @Override
53      public String toString() {
54          return "User{" +
```

```
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", date=" + date +
                ", address=" + address +
                '}';
    }
}
```

Address.java

```
package cn.tulingxueyuan.bean;

public class Address {
    private String province;
    private String city;
    private String town;

    public String getProvince() {
        return province;
    }

    public void setProvince(String province) {
        this.province = province;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getTown() {
        return town;
    }

    public void setTown(String town) {
        this.town = town;
```

```
    }

    @Override
    public String toString() {
        return "Address{" +
                "province='" + province + '\'' +
                ", city='" + city + '\'' +
                ", town='" + town + '\'' +
                '}';
    }
}
```

login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<form action="addUser" method="post">
  编号:<input type="text" name="id"/><br>
  姓名:<input type="text" name="name"/><br>
  年龄:<input type="text" name="age"/><br>
  日期:<input type="text" name="date"/><br>
  省份:<input type="text" name="address.province"/><br>
  城市:<input type="text" name="address.city"/><br>
  区域:<input type="text" name="address.town"/><br>
    <input type="submit" value="submit"/><br>
</form>
</body>
</html>
```

UserController.java

```
package cn.tulingxueyuan.controller;

import cn.tulingxueyuan.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
```

```java

@Controller
public class UserController {

    @RequestMapping("/addUser")
    public String addUser(User user){
        System.out.println(user);
        return "success";
    }
}
```

数组绑定
集合绑定

2、乱码问题的解决

我们在表单或者发送请求的时候，经常会遇到中文乱码的问题，那么如何解决乱码问题呢?
GET请求：在server.xml文件中，添加URIEncoding="UTF-8"
POST请求：编写过滤器进行实现

```xml
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-clas
        <!--解决post请求乱码-->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <!--解决响应乱码-->
        <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>
```

注意：如果配置了多个过滤器，那么字符编码过滤器一定要在最前面，否则失效。

3、SpringMVC对原生API的支持

```
package cn.tulingxueyuan.controller;

import cn.tulingxueyuan.bean.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.BufferedReader;
import java.io.PrintWriter;

@Controller
public class UserController {

    @RequestMapping("/addUser")
    public String addUser(User user){
        System.out.println(user);
        return "success";
    }

    /**
     * SpringMVC也可以在参数上使用原生的Servlet API
     *
     * HttpSession
     * HttpServletRequest
     * HttpServletResponse
     *
     * java.security.Principal 安全协议相关
     * Locale：国际化相关的区域信息对象
     * InputStream:
     *      ServletInputStream inputStream = request.getInputStream();
     * OutputStream:
     *      ServletOutputStream outputStream = response.getOutputStream();
     * Reader:
     *      BufferedReader reader = request.getReader();
```

```
     * Writer:
     *     PrintWriter writer = response.getWriter();
     * @param session
     * @param request
     * @param response
     * @return
     */
    @RequestMapping("api")
    public String api(HttpSession session, HttpServletRequest request, HttpServletResponse
        request.setAttribute("requestParam","request");
        session.setAttribute("sessionParam","session");
        return "success";
    }
}
```

2、请求映射处理

1、@RequestMapping

@RequestMapping用来匹配客户端发送的请求，可以在方法上使用，也可以在类上使用。

方法：表示用来匹配要处理的请求

类上：表示为当前类的所有方法的请求地址添加一个前置路径，访问的时候必须要添加此路径

```
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/tulingxueyuan")
public class HelloController{

    /*
    * @RequestMapping就是用来标识此方法用来处理什么请求，其中的/可以取消
    * 取消后默认也是从当前项目的根目录开始查找，一般在编写的时候看个人习惯
    * 同时，@RequestMapping也可以用来加在类上，
    * */
    @RequestMapping("/hello")
    public String hello(Model model){
```

```
        model.addAttribute("msg","hello,SpringMVC");
        return "hello";
    }
}
```

注意：在整个项目的不同方法上不能包含相同的@RequestMapping值

除此以外，@RequestMapping注解还可以添加很多额外的属性值，用来精确匹配请求

```
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/tulingxueyuan")
public class HelloController{

    /*
    * @RequestMapping就是用来标识此方法用来处理什么请求，其中的/可以取消
    * 取消后默认也是从当前项目的根目录开始查找，一般在编写的时候看个人习惯
    * 同时，@RequestMapping也可以用来加在类上，
    * */
    @RequestMapping("/hello")
    public String hello(Model model){
        model.addAttribute("msg","hello,SpringMVC");
        return "hello";
    }

    /**
     * Request的其他属性值
     *  value:要匹配的请求
     *  method:限制发送请求的方式：  POST GET
     *  params:表示请求要接受的参数,如果定义了这个属性，那么发送的时候必须要添加参数
     *         params有几种匹配规则
     *          1、直接写参数的名称，param1,param2
     *              params = {"username"}
     *          2、表示请求不能包含的参数，！param1
     *              params = {"!username"}
```

```
33          *          3、表示请求中需要要包含的参数但是可以限制值 param1=values  param1!=value
34          *              params = {"username=123","age"}
35          *              params = {"username!=123","age"}
36          *  headers:填写请求头信息
37          *          chrome: User-Agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
38          *          firefox:User-Agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Ge
39          *
40          *  consumers:只接受内容类型是哪种的请求，相当于指定Content-Type
41          *  produces:返回的内容类型 Content-Type: text/html;charset=utf-8
42          *
43          * @return
44          */
45         @RequestMapping(value = "/hello2",method = RequestMethod.POST)
46         public String hello2(){
47             return "hello";
48         }
49
50         @RequestMapping(value = "/hello3",params = {"username!=123","age"})
51         public String hello3(String username){
52             System.out.println(username);
53             return "hello";
54         }
55
56         @RequestMapping(value = "/hello4",headers = {"User-Agent=Mozilla/5.0 (Windows NT 10.0
57         public String hello4(){
58             return "hello";
59         }
60     }
```

@RequestMapping还包含了很多复杂的匹配功能，提供了通配符的支持：

```
1
2       /**
3        * @Request包含三种模糊匹配的方式，分别是：
4        *  ？：能替代任意一个字符
5        *  *: 能替代任意多个字符和一层路径
6        *  **: 能代替多层路径
7        * @return
8        */
```

```
    @RequestMapping(value = "/**/h*llo?")
    public String hello5(){
        System.out.println("hello5");
        return "hello";
    }
}
```

2、@PathVariable

如果需要在请求路径中的参数像作为参数应该怎么使用呢？可以使用@PathVariable注解，此注解就是提供了对占位符URL的支持，就是将URL中占位符参数绑定到控制器处理方法的参数中。

```

/***
 * @Author 徐庶   QQ:1092002729
 * @Slogan 致敬大师，致敬未来的你
 *
 * @PathVariable 用在参数上面的
 *  专门用来获取URL目录级别的参数
 *   比如 http://localhost:8080/springmvc/path/user/123/xushu
 *    要获得123   @RequestMapping("/user/{id}") : @PathVariable("id") Integer id
 *
 *    如果是单个参数接收必须要使用@PathVariable来声明对应的参数占位符名字
 *    如果是javaBean可以省略@PathVariable，要保证占位符的名字和javaBean的属性名字一样
 *
 */
@Controller
@RequestMapping("/path")
public class PathvariableController {

    /**
     * 获取用户实体   传入id
     * @return
     */
    @RequestMapping("/user/{id}/{username}")
    public String path01(@PathVariable("id") Integer id,@PathVariable("username") String
        System.out.println(id);
        System.out.println(name);

        return "/index.jsp";
```

```
28      }
29
30      @RequestMapping("/user02/{id}/{name}")
31      public String path02(User user){
32          System.out.println(user);
33          return "/index.jsp";
34      }
35  }
```

3、REST

客户端映射到服务器资源的一种架构设计

URL

restful

一种优雅的URL风格:

万维网 http协议 http://www.tulingxueyuan.cn

混乱：每一个都有一套自己的命名风格

根据id查询一个用户

user/getuser.do?id=1

user.do?action=getUser&id=1

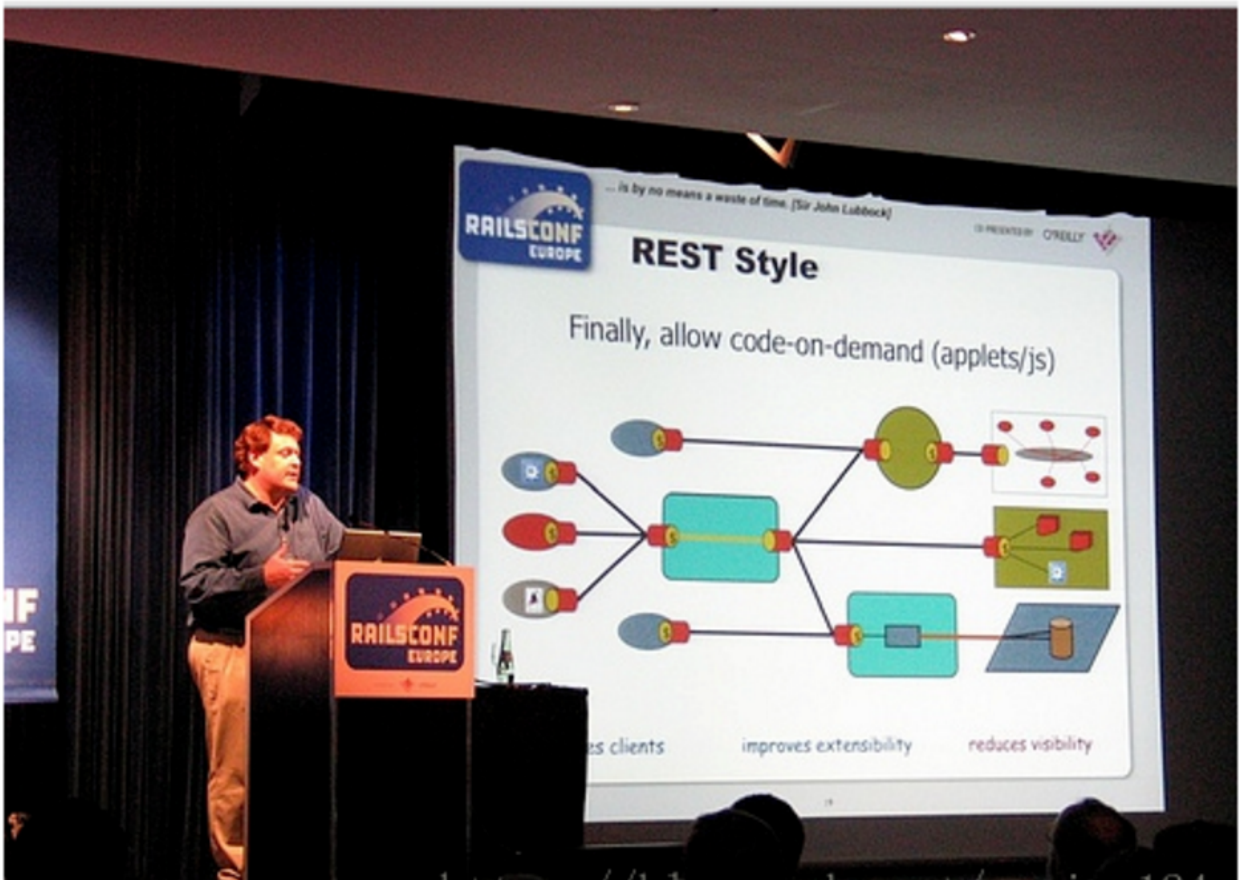
user.do?action=a&id=xx

user/chaxunyonghu?id=1

user/CXYH?id=1

user/tulingxueyuan_cxyh?id=1

user/a.do?action=b?c=xx



REST即表述性状态传递（英文：Representational State Transfer，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种软件架构风格。它是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。

一种相较与之前URL所产生一种更优雅的URL风格

URL CRUD

如果是原来的架构风格，需要发送四个请求，分别是?

查询用户：http://localhost:8080/app/user.do?action=getUser&id=xxx GET

增加用户: http://localhost:8080/app/user_add.do POST

修改用户: http://localhost:8080/app/xiugaiuser.do POST

删除用户: http://localhost:8080/app/delete.do?id=1 GET/POST

按照此方式发送请求的时候比较麻烦，需要定义多种请求，而在HTTP协议中，有不同的发送请求的方式，分别是GET、POST、PUT、DELETE等，我们如果能让不同的请求方式表示不同的请求类型就可以简化我们的查询,改成名词：

面向资源

看URL就知道要什么，， 看http method就知道干什么

查询用户: http://localhost:8080/xxx/user/1 GET --查询

查询多个用户: http://localhost:8080/xxx/users GET

新增用户: http://localhost:8080/xxx/user POST ---新增

修改用户: http://localhost:8080/xxx/user/1 PUT --修改

删除用户:http://localhost:8080/xxx/user/1 DELETE --删除

一切看起来都非常美好，但是大家需要注意了，我们在发送请求的时候只能发送post或者get，没有办法发送put和delete请求，那么应该如何处理呢？下面开始进入代码环节：

RestController.java

```java
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Controller
public class RestController {

    @RequestMapping(value = "/user",method = RequestMethod.POST)
    public String add(){
        System.out.println("添加");
        return "success";
    }

    @RequestMapping(value = "/user/{id}",method = RequestMethod.DELETE)
    public String delete(@PathVariable("id") Integer id){
        System.out.println("删除: "+id);
        return "success";
    }

    @RequestMapping(value = "/user/{id}",method = RequestMethod.PUT)
    public String update(@PathVariable("id") Integer id){
        System.out.println("更新: "+id);
        return "success";
    }

    @RequestMapping(value = "/user/{id}",method = RequestMethod.GET)
    public String query(@PathVariable("id") Integer id){
        System.out.println("查询: "+id);
```

```
        return "success";
    }
}
```

web.xml

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

rest.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/user" method="post">
        <input type="submit" value="增加">
    </form>
    <form action="/user/1" method="post">
        <input name="_method" value="delete" type="hidden">
        <input type="submit" value="删除">
    </form>
    <form action="/user/1" method="post">
        <input name="_method" value="put" type="hidden">
        <input type="submit" value="修改">
    </form>
    <a href="/user/1">查询</a><br/>
</body>
</html>
```

success.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isErrorPage="true" %>
<html>
<head>
   <title>Title</title>
</head>
<body>
666
</body>
</html>
```

解决：405错误的自定义过滤器

4种解决方式：

```
*          1.用tomcat7
*          2.不用转发，用重定向
*          3. 将jsp的page指定 isErrorPage属性改成true(不建议)
*          4. 自定义一个过滤器，将request.method改回POST
```


```java
public class GetMethodConvertingFilter implements Filter {

    @Override
    public void init(FilterConfig config) throws ServletException {
        // do nothing
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain ch
        throws IOException, ServletException {

        chain.doFilter(wrapRequest((HttpServletRequest) request), response);

    }
```

```
14
15      @Override
16      public void destroy() {
17          // do nothing
18      }
19
20      private static HttpServletRequestWrapper wrapRequest(HttpServletRequest request) {
21          return new HttpServletRequestWrapper(request) {
22              @Override
23              public String getMethod() {
24                  return "POST";
25              }
26          };
27      }
28  }
```

web.xml

```
1  <!--为了解决转发到jsp出现：HTTP Status 405 - JSPs only permit GET POST or HEAD
2        但是一般修改和删除都会进行重定向，所以这个过滤器先不用，万一有遇到修改或删除还要转发就使用
3     <filter>
4      <filter-name>backToPostHttpMethodFilter</filter-name>
5      <filter-class>cn.tulingxueyuan.filter.BackToPostHttpMethodFilter</filter-class>
6    </filter>
7    <filter-mapping>
8      <filter-name>backToPostHttpMethodFilter</filter-name>
9      <url-pattern>/*</url-pattern>
10     <dispatcher>FORWARD</dispatcher>
11   </filter-mapping -->
```

4、静态资源的访问

当页面中包含静态资源的时候我们能够正确的获取到吗?

hello.jsp

```
1  <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2  <%
3     pageContext.setAttribute("ctx",request.getContextPath());
4  %>
```

```
<html>
<head>
    <title>Title</title>
</head>
<body>
hello springmvc
<img src="${ctx}/images/timg.jpg">
</body>
</html>
```

此时大家发现我们请求的图片根本访问不到，根据查看发现路径是没有问题的，那么为什么会找不到静态资源呢?

```
12-Mar-2020 23:37:27.660 警告 [http-nio-8080-exec-9] org.springframework.web.servlet.DispatcherServlet
 .noHandlerFound No mapping for GET /springmvc_viewResolver_war_exploded/images/timg.jpg
```

大家发现此时是找不到对应的mapping映射的，此时是因为DispatcherServlet会拦截所有的请求，而此时我们没有对应图片的请求处理方法，所以此时报错了，想要解决的话非常简单，只需要添加一个配置即可

```
<!--
此配置表示 我们自己配置的请求由controller来处理，但是不能请求的处理交由tomcat来处理
静态资源可以访问，但是动态请求无法访问
-->
<mvc:default-servlet-handler/>
```

但是加上此配置之后，大家又发现此时除了静态资源无法访问之外，我们正常的请求也无法获取了，因此还需要再添加另外的配置:

```
<!--保证静态资源和动态请求都能够访问-->
<mvc:annotation-driven></mvc:annotation-driven>
```