

第2天: SpringMVC

一、目标

1. 掌握json数据交互
2. 了解RESTful风格支持
3. 实现SpringMVC文件上传
4. 掌握SpringMVC统一异常处理
5. 了解SpringMVC的拦截器

一、json数据交互

4.1 注解说明

4.1.1 @RequestBody

- 位置: 参数
- 作用: 将请求的json数据转换成对象

4.1.2 @ResponseBody

- 位置: 方法
- 作用: 将返回的对象转换成json数据

4.2 案例演示

- 添加依赖: pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <parent>
5     <artifactId>mvc2</artifactId>
6     <groupId>com.itheima</groupId>
7     <version>1.0-SNAPSHOT</version>
8   </parent>
9   <modelVersion>4.0.0</modelVersion>
10  <artifactId>mvc-day02-controller</artifactId>
11  <packaging>war</packaging>
12  <dependencies>
13    <dependency>
14      <groupId>org.springframework</groupId>
15      <artifactId>spring-webmvc</artifactId>
16      <version>5.1.7.RELEASE</version>
17    </dependency>
18    <dependency>
19      <groupId>javax.servlet</groupId>
20      <artifactId>servlet-api</artifactId>
21      <version>2.5</version>
```



```
24      <!-- Json转换由jackson完成 -->
25      <dependency>
26          <groupId>com.fasterxml.jackson.core</groupId>
27          <artifactId>jackson-databind</artifactId>
28          <version>2.9.9</version>
29      </dependency>
30  </dependencies>
31 </project>
```

- com.itheima.controller.JsonController

```
1  package com.itheima.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestBody;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.ResponseBody;
7
8  /**
9   * Json格式数据交互案例代码
10   * @RestController: 等于@Controller+@ResponseBody
11   */
12 @Controller
13 public class JsonController {
14
15     /**
16      * @RequestBody: 将请求的json数据解析成对象（默认由jackson提供支持）
17      * @ResponseBody: 将返回的对象转换成json数据（默认由jackson提供支持）
18      *
19      */
20     @ResponseBody
21     @RequestMapping("hello")
22     public Object hello(@RequestBody Order order) {
23         System.out.println(order);
24         return order;
25     }
26 }
27
28
```

- index.jsp

```
1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <html>
3  <head>
4      <title>测试页面</title>
5  </head>
6  <body>
7      <h3>提交Json数据请求</h3>
8      <form id="jsonForm">
9          <input name="id" value="1"/>
10         <input name="name" value="2"/>
11         <button type="button" onclick="submitJson()">提交</button>
12     </form>
```

```
15     </div>
16 </body>
17 <script>
18     function submitJson(){
19         // 1. 创建异步请求对象
20         let req = new XMLHttpRequest();
21         // 2. 打开请求
22         req.open("POST", "/hello.do", true);
23         // 3. 设置请求的数据类型(必须)
24         req.setRequestHeader("Content-Type", "application/json;charset=UTF-
25 8");
26         // 4. 异步读取响应
27         req.onreadystatechange = function(){
28             // 2.1 判断响应状态
29             if(req.readyState==4 || req.status==200){
30                 let data = req.responseText;
31                 document.getElementById("jsonRes").innerHTML=data;
32             }
33         };
34         // 5. 封装数据
35         let es =
36         document.getElementById("jsonForm").getElementsByTagName("input");
37         let json = {};
38         for (let i=0; i<=es.length; i++){
39             if(es[i]!=undefined){
40                 let name = es[i].name;
41                 let value = es[i].value;
42                 json[name] = value;
43             }
44         }
45         // 6. 发送数据
46         let body = JSON.stringify(json);
47         req.send(body);
48     }
49 </script>
50 </html>
```

二、RESTful支持

5.1 RESTful介绍

5.1.1 基本介绍

REST全称是Representational State Transfer，中文意思是表述（编者注：通常译为表征）性状态转移。它首次出现在2000年Roy Fielding的博士论文中，Roy Fielding是HTTP规范的主要编写者之一。他在论文中提到：“我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。REST指的是一组架构约束条件和原则。”如果一个架构符合REST的约束条件和原则，我们就称它为RESTful架构。

响很深，但是理论上REST架构风格并不是绑定在HTTP上，只不过目前HTTP是唯一与REST相关的实例。所以我们这里描述的REST也是通过HTTP实现的REST。

5.1.2 RESTful优点

- 结构清晰
- 符合标准
- 易于理解
- 扩展方便
- 地址简洁
- 应用广泛

5.1.3 RESTful示例

| 请求地址(URI) | 请求方法 | 请求含义 |
|-----------|--------|----------|
| /user/1 | GET | 获取编号1的用户 |
| /user/1 | PUT | 修改编号1的用户 |
| /user/1 | DELETE | 删除编号1的用户 |
| /user | POST | 添新增用户 |

5.2 URL风格对比

| | 增 | 删 | 查 | 改 |
|---------|-----------|-------------------|----------------|-------------------|
| 传统 | /user/add | /user/delete?id=1 | /user/get?id=1 | /user/update?id=1 |
| RESTful | /user | /user/1 | /user/1 | /user/1 |

5.3 案例演示

5.3.1 特点一：方法区分

浏览器form表单只支持GET与POST请求，而DELETE、PUT等方法并不支持，spring3.0添加了一个过滤器，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，该过滤器为HiddenHttpMethodFilter。

HiddenHttpMethodFilter

- web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6   version="2.5">
7
8
9   <!-- 1. 使用统一的Servlet处理请求 -->
```



```
12     <url-pattern>*.do</url-pattern>
13 </servlet-mapping>
14
15 <!-- 2. 定义前端控制器统一处理请求 -->
16 <servlet>
17     <servlet-name>DispatcherServlet</servlet-name>
18     <servlet-
19 class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
20     <init-param>
21         <param-name>contextConfigLocation</param-name>
22         <param-value>classpath:springMVC.xml</param-value>
23     </init-param>
24     <load-on-startup>1</load-on-startup>
25 </servlet>
26
27 <!-- 1. 配置需要过滤器的资源 -->
28 <filter-mapping>
29     <filter-name>hiddenHttpMethodFilter</filter-name>
30     <url-pattern>/*</url-pattern>
31 </filter-mapping>
32
33 <!-- 2. 使用filter实现对PUT,DELETE的支持 -->
34 <filter>
35     <filter-name>hiddenHttpMethodFilter</filter-name>
36     <filter-
37 class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
38 </filter>
39 </web-app>
```

- index.jsp

```
1 <h3>提交PUT,DELETE请求</h3>
2 <!-- 使用HiddenHttpMethodFilter过滤器实现PUT,DELETE,POST,GET..支持 --%>
3 <!-- form标签属性method必须设置为post --%>
4 <form action="/hello.do" method="post">
5     <!-- 表示提交方法的方式: --%>
6     <!-- 1. 设置隐藏域name=_method 2. 设置值为真实的提交方法PUT/DELETE --%>
7     <input type="hidden" name="_method" value="PUT"/>
8     <input name="id" value="1"/>
9     <input name="name" value="2"/>
10    <input type="submit"/>
11 </form>
```

- com.itheima.controller.RestController

```
1 package com.itheima.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
```



```
11  */
12  @Controller
13  public class RestController {
14
15
16      /**
17       * 提交PUT方法
18       * 注意1: 需要使用HiddenHttpMethodFilter过滤请求
19       * 注意2: 只支持返回json类型的数据
20       */
21      @RequestMapping(value = "hello", method = RequestMethod.PUT)
22      @ResponseBody
23      public String hello(Integer id, String name){
24          System.out.println(id);
25          System.out.println(name);
26          return "success";
27      }
28
29      /**
30       * 提交DELETE方法
31       * 注意1: 需要使用HiddenHttpMethodFilter过滤请求
32       * 注意2: 只支持返回json类型的数据
33       */
34      @RequestMapping(value = "hello", method = RequestMethod.DELETE)
35      @ResponseBody
36      public String hello2(Integer id, String name){
37          System.out.println(id);
38          System.out.println(name);
39          return "success";
40      }
41  }
```

5.3.2 特点二: 地址传参

5.3.2 @PathVariable

- com.itheima.controller.RestController

```
1  /**
2   * @PathVariable: 绑定URL地址中的参数
3   * value: 参数名称
4   * required: 默认true,必传;
5   *
6   * 请求示例: http://localhost:8080/hello/1/Jason
7   */
8  @RequestMapping(value = "hello/{id}/{name}")
9  @ResponseBody
10 public String hello3(@PathVariable Integer id, @PathVariable("name") String
    name){
11     System.out.println(id);
12     System.out.println(name);
13     return "success";
14 }
```

6.1 传统方式

6.1.1 页面要求

1. 需要设置method为post
2. 需要设置enctype为multipart/form-data
3. 需要设置文件

- index.jsp

```
1 <h3>文件上传</h3>
2 <form action="/upload.do"
3     method="post" <!-- 1. 需要设置method为post提交 -->
4     enctype="multipart/form-data" <!-- 2. 需要设置enctype为multipart/form-
data -->>
5     <input name="name" type="text" value="Jason" />
6     <input name="file" type="file" /> <!-- 3. 需要设置文件 -->
7     <input type="submit"/>
8 </form>
```

6.1.2 后端代码

添加依赖: pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6     http://maven.apache.org/xsd/maven-4.0.0.xsd">
7     <parent>
8         <artifactId>mvc2</artifactId>
9         <groupId>com.itheima</groupId>
10        <version>1.0-SNAPSHOT</version>
11    </parent>
12    <modelVersion>4.0.0</modelVersion>
13    <artifactId>mvc-day02-controller</artifactId>
14    <packaging>war</packaging>
15    <dependencies>
16        <dependency>
17            <groupId>org.springframework</groupId>
18            <artifactId>spring-webmvc</artifactId>
19            <version>5.1.7.RELEASE</version>
20        </dependency>
21        <dependency>
22            <groupId>javax.servlet</groupId>
23            <artifactId>servlet-api</artifactId>
24            <version>2.5</version>
25        </dependency>
26        <!-- Json转换由jackson完成 -->
27        <dependency>
28            <groupId>com.fasterxml.jackson.core</groupId>
29            <artifactId>jackson-databind</artifactId>
30            <version>2.9.9</version>
31        </dependency>
```



```
31     <groupId>commons-fileupload</groupId>
32     <artifactId>commons-fileupload</artifactId>
33     <version>1.4</version>
34 </dependency>
35 </dependencies>
36 </project>
```

处理请求: UploadController

```
1  package com.itheima.controller;
2
3  import org.apache.commons.fileupload.FileItem;
4  import org.apache.commons.fileupload.disk.DiskFileItemFactory;
5  import org.apache.commons.fileupload.servlet.ServletFileUpload;
6  import org.springframework.stereotype.Controller;
7  import org.springframework.web.bind.annotation.RequestMapping;
8
9  import javax.servlet.http.HttpServletRequest;
10 import java.io.File;
11 import java.util.List;
12 import java.util.UUID;
13
14 /**
15  * 文件上传案例代码.
16  *
17  * @author : Jason.lee
18  * @version : 1.0
19  */
20 @Controller
21 public class UploadController {
22
23     /**
24      * 传统文件上传
25      * 需要使用commons-fileupload组件的原生API解析请求并处理文件存储
26      */
27     @RequestMapping("upload")
28     public String hello(HttpServletRequest request) throws Exception {
29         // 1. 指定上传目录
30         String path =
31 request.getSession().getServletContext().getRealPath("/upload");
32         File file = new File(path);
33         if (!file.exists()) {
34             file.mkdir();
35         }
36         // 2. 创建组件工具
37         DiskFileItemFactory factory = new DiskFileItemFactory();
38         ServletFileUpload upload = new ServletFileUpload(factory);
39         // 3. 解析请求数据
40         List<FileItem> items = upload.parseRequest(request);
41         for (FileItem item : items) {
42             // 判断是否是普通表单字段
43             //
44             if (item.isFormField()) {
45                 System.out.println("普通字段: " + item.getString("UTF-8"));
46             }
47         }
48     }
49 }
```



```
48         item.write(new File(path + "/" + id + item.getName()));
49     }
50 }
51 return "success";
52 }
53 }
```

6.2 SpringMVC

6.2.1 页面

- index.jsp

```
1 <h3>文件上传</h3>
2 <form action="/upload.do"
3     method="post" <!-- 1. 需要设置method为post提交 -->
4     enctype="multipart/form-data" <!-- 2. 需要设置enctype为multipart/form-
data -->
5     >
6     <input name="name" type="text" value="Jason" />
7     <!-- 文件名name需要严格与接收参数名称一致 -->
8     <input name="img" type="file" /> <!-- 3. 需要设置文件 -->
9     <input type="submit"/>
10 </form>
```

6.2.2 配置

- pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6     http://maven.apache.org/xsd/maven-4.0.0.xsd">
7     <parent>
8         <artifactId>mvc2</artifactId>
9         <groupId>com.itheima</groupId>
10        <version>1.0-SNAPSHOT</version>
11    </parent>
12    <modelVersion>4.0.0</modelVersion>
13    <artifactId>mvc-day02-controller</artifactId>
14    <packaging>war</packaging>
15    <dependencies>
16        <dependency>
17            <groupId>org.springframework</groupId>
18            <artifactId>spring-webmvc</artifactId>
19            <version>5.1.7.RELEASE</version>
20        </dependency>
21        <dependency>
22            <groupId>javax.servlet</groupId>
23            <artifactId>servlet-api</artifactId>
24            <version>2.5</version>
25        </dependency>
26    </dependencies>
```



```

26     <artifactId>jackson-databind</artifactId>
27     <version>2.9.9</version>
28 </dependency>
29 <!-- 文件上传的组件支持 -->
30 <dependency>
31     <groupId>commons-fileupload</groupId>
32     <artifactId>commons-fileupload</artifactId>
33     <version>1.4</version>
34 </dependency>
35 </dependencies>
36 </project>

```

- springMVC.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mvn="http://www.springframework.org/schema/mvc"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7       http://www.springframework.org/schema/beans/spring-beans.xsd
8       http://www.springframework.org/schema/context
9       http://www.springframework.org/schema/context/spring-context.xsd
10      http://www.springframework.org/schema/mvc
11      http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13     <!-- 1. 开启注解扫描 -->
14     <context:component-scan base-package="com.itheima.controller"/>
15     <!-- 2. 开启注解驱动（配置了处理器映射器，处理器适配器等） -->
16     <mvn:annotation-driven/>
17     <!-- 3. 配置视图解析器 -->
18     <bean
19         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
20         <property name="prefix" value="/pages/" />
21         <property name="suffix" value=".jsp" />
22     </bean>
23     <!-- 4. 配置文件解析器 -->
24     <!-- id: 必须固定为 multipartResolver -->
25     <bean id="multipartResolver"
26         class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
27         <!-- maxUploadSize: 设置上传的最大值: 10M(10*1024KB*1024B) -->
28         <property name="maxUploadSize" value="10485760" />
29     </bean>
30 </beans>

```

6.2.3 代码

- com.itheima.controller.UploadController

```

1 /**
2  * SpringMVC文件上传
3  * 基于commons-fileupload组件处理的请求
4  */
5 @RequestMapping("upload")

```



```
7      // 1. 指定上传目录
8      String path =
request.getSession().getServletContext().getRealPath("/upload");
9      File file = new File(path);
10     if (!file.exists()) {
11         file.mkdir();
12     }
13     // 2. 处理上传数据
14     System.out.println(name);
15     String id = UUID.randomUUID().toString(); // 保证文件名不重复
16     String filename = img.getOriginalFilename(); // 获取文件名的API不是getName
17     // 保存文件的API是transferTo (转移到指定位置)
18     img.transferTo(new File(path + "/" + id + filename));
19     return "success";
20 }
```

四、统一异常处理

2.1 概念

- 在项目开发过程中, 全局的捕捉异常进行统一的处理

2.1.1 背景

在三层架构中, 如果Dao持久层有异常可以抛出到调用方(Service), Service有异常可以抛出到Controller, 但是Controller有异常, 不建议抛出到客户端(用户), 因为用户看不懂异常, 无法进行处理会导致用户体验非常差。

2.1.2 处理

方案一:

- 在Controller层使用try catch捕捉异常

```
1  @Controller
2  public class UserController {
3
4      @Autowired
5      UserService userService;
6
7      @RequestMapping("list")
8      public String list(){
9          try{
10             userService.findAll();
11         }catch(Exception e){
12             // 异常处理
13             return "error";
14         }
15         return "success";
16     }
```

- 在servlet过滤器(Filter)调用代码上捕捉异常

```
1 public class ExceptionFilter implements Filter {
2     public void doFilter(ServletRequest request, ServletResponse response,
3         FilterChain chain){
4         try{
5             chain.doFilter(request, response);
6         }catch(Exception e){
7             // 异常处理
8         }
9     }
10 }
```

方案三:

- 使用SpringMVC提供的异常处理器支持

2.2 案例演示

- com.itheima.ex.UserController

```
1 package com.itheima.ex;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 /**
8  * 模拟查询所有用户异常.
9  *
10  * @author : Jason.lee
11  * @version : 1.0
12  */
13 @Controller
14 public class UserController {
15
16
17     @RequestMapping("list")
18     public String list(Model model){
19         model.addAttribute("msg", "用户列表");
20         // 希望抛出异常进行友好提示: 返回error.jsp页面
21         int i = 1/0;
22         return "success";
23     }
24 }
```

- com.itheima.ex.CustomExceptionHandler

```
1 package com.itheima.ex;
2
3 import org.springframework.stereotype.Component;
```

```
7
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 /**
12  * 自定义异常处理器.
13  *
14  * @author : Jason.lee
15  * @version : 1.0
16  */
17 @Component
18 public class CustomExceptionHandler implements HandlerExceptionResolver {
19
20
21     /**
22      * 发现异常将调用该方法处理
23      * @param httpRequest 请求
24      * @param httpResponse 响应
25      * @param o 处理的方法
26      * @param e 异常
27      * @return 返回视图
28      */
29     @Override
30     public ModelAndView resolveException(HttpServletRequest
31 httpRequest,
32                                     HttpServletResponse
33                                     Object o, Exception e) {
34         ModelAndView mv = new ModelAndView();
35         mv.setViewName("error");
36         mv.addObject("msg", e.getMessage());
37         return mv;
38     }
39 }
```

五、SpringMVC拦截器

3.1 概念

3.1.1 什么是拦截器

springmvc框架中的拦截器，相当于web阶段学习的过滤器（filter），可以实现前置增强和后置增强功能。在springmvc框架中，拦截器可以对处理器方法执行预处理（前置增强），和执行后处理（后置增强）。

3.1.2 拦截器的作用

- Spring MVC 的处理器拦截器类似于 Servlet 开发中的过滤器 Filter
- 用于对处理器进行**预处理** 和 **后处理**。
- 用户可以自己定义一些拦截器来实现特定的功能。

3.1.3 拦截器与过滤器

| | | |
|----------------------|------------------------------|-----------------------|
| 过滤器(Filter) | WEB项目均可使用(源于Servlet规范) | 可以在web.xml中配置/*过滤所有资源 |
| 拦截器 (Interceptor) | SpringMVC项目中(SpringMVC特有的组件) | 只拦截Controller的方法对应的资源 |

3.2 案例

3.2.1 编写拦截器

- com.itheima.interceptor.CustomHandlerInterceptor

```
1 package com.itheima.interceptor;
2
3 import org.springframework.web.servlet.HandlerInterceptor;
4 import org.springframework.web.servlet.ModelAndView;
5
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 /**
10  * 自定义拦截器.
11  *
12  * @author : Jason.lee
13  * @version : 1.0
14  */
15 public class CustomHandlerInterceptor implements HandlerInterceptor {
16
17     /**
18      * Controller方法调用前执行的方法 【前置通知】
19      * @return 是否执行处理器方法
20      */
21     @Override
22     public boolean preHandle(HttpServletRequest request,
23                             HttpServletResponse response, Object handler) throws Exception {
24         System.out.println("1. 先执行preHandle方法..");
25         return false;
26     }
27
28     /**
29      * Controller方法调用成功后执行的方法 【后置通知】
30      * 需要preHandle方法返回true
31      * 并且Controller方法没有异常才执行
32      */
33     @Override
34     public void postHandle(HttpServletRequest request, HttpServletResponse
35                             response, Object handler, ModelAndView modelAndView) throws Exception {
36         System.out.println("3. 后执行postHandle方法..");
37     }
38
39     /**
40      * Controller方法执行完成后执行的方法 【最终通知】
41      * 需要postHandle方法返回true
42      */
43 }
```



```
42     @Override
43     public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex) throws
    Exception {
44         System.out.println("4. 最终执行afterCompletion方法..");
45     }
46 }
```

3.2.2 配置拦截器

- springMVC.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xmlns:mvc="http://www.springframework.org/schema/mvc"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    https://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    https://www.springframework.org/schema/mvc/spring-mvc.xsd">
7
8
9      <context:component-scan base-package="com.itheima.ex"/>
10
11     <mvc:annotation-driven/>
12
13     <bean id="viewResolver"
14         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
15         <property name="prefix" value="/WEB-INF/pages/" />
16         <property name="suffix" value=".jsp" />
17     </bean>
18
19     <!--
20         <mvc:interceptors>: 配置拦截器链(多个)
21         <mvc:interceptor>: 定义拦截器
22         <mvc:mapping path="/**"/>: 拦截的资源路径-@RequestMapping
23         <bean>: 拦截器定义
24     -->
25     <mvc:interceptors>
26         <mvc:interceptor>
27             <!-- path: 拦截的资源路径 -->
28             <mvc:mapping path="/**"/>
29             <!-- class: 拦截器类路径 -->
30             <bean
31                 class="com.itheima.interceptor.CustomHandlerInterceptor"/>
32             </mvc:interceptor>
33         </mvc:interceptors>
34     </beans>
```

3.2.3 执行流程

