

## 第3天: SpringMVC

### 一、目标

1. 实现SpringMVC文件上传
2. 掌握SpringMVC统一异常处理
3. 了解SpringMVC的拦截器

### 二、文件上传

#### 2.1 传统方式

##### 2.1.1 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>
```

##### 2.1.2 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>
```



```
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>
22 </dependency>
23 <!-- Json转换由jackson完成 -->
24 <dependency>
25     <groupId>com.fasterxml.jackson.core</groupId>
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>
```

### 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");
```



```
33         file.mkdir();
34     }
35     // 2. 创建组件工具
36     DiskFileItemFactory factory = new DiskFileItemFactory();
37     ServletFileUpload upload = new ServletFileUpload(factory);
38     // 3. 解析请求数据
39     List<FileItem> items = upload.parseRequest(request);
40     for (FileItem item : items) {
41         // 判断是否是普通表单字段
42         //
43         if (item.isFormField()) {
44             System.out.println("普通字段: " + item.getString("UTF-8"));
45         } else {
46             // 保存文件
47             String id = UUID.randomUUID().toString(); // 保证文件名不重复
48             item.write(new File(path + "/" + id + item.getName()));
49         }
50     }
51     return "success";
52 }
53 }
```

## 2.2 SpringMVC

### 2.2.1 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>
```

### 2.2.2 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 <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>
22   </dependency>
23   <!-- Json转换由jackson完成 -->
24   <dependency>
25     <groupId>com.fasterxml.jackson.core</groupId>
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         https://www.springframework.org/schema/context/spring-context.xsd
10        http://www.springframework.org/schema/mvc
11        https://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"
```



```
23     </bean>
24 </beans>
```

### 2.2.3 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")
6  public String hello(HttpServletRequest request, String name, MultipartFile
img) throws Exception {
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 }
```

## 三、统一异常处理

### 3.1 概念

- 在项目开发过程中, 全局的捕捉异常进行统一的处理

#### 3.1.1 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。

#### 3.1.2 处理

方案一:

- 在Controller层使用try catch捕捉异常

```
1  @Controller
2  public class UserController {
```



```
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     }  
17 }
```

#### 方案二:

- 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

## 3.2 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 }
```



```
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;
4 import org.springframework.web.bind.annotation.ExceptionHandler;
5 import org.springframework.web.servlet.HandlerExceptionResolver;
6 import org.springframework.web.servlet.ModelAndView;
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                                     httpServletResponse,
34                                     Object o, Exception e) {
35         ModelAndView mv = new ModelAndView();
36         mv.setViewName("error");
37         mv.addObject("msg", e.getMessage());
38         return mv;
39     }
40 }
```

## 四、SpringMVC拦截器

### 4.1.1 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）。

### 4.1.2 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。

### 4.1.3 拦截器与过滤器

| 组件名称             | 组件来源                         | 作用范围                  |
|------------------|------------------------------|-----------------------|
| 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 |

## 4.2 案例

### 4.2.1 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方法调用钱执行的方法 【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      * Controller方法调用成功后执行的方法 【后置通知】
29      * 需要preHandle方法返回true
30      * 并且Controller方法没有异常才执行
31      */
32      @Override
33      public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
34          System.out.println("3. 后执行postHandle方法..");
35      }
36
37      /**
38      * Controller方法执行完成后执行的方法 【最终通知】
39      * 需要preHandle方法返回true
40      * 不论Controller方法是否异常都会执行
41      */
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  }
```

#### 4.2.2 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"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
14          <property name="prefix" value="/WEB-INF/pages/" />
15          <property name="suffix" value=".jsp" />
16      </bean>
17
18      <!--
19          <mvc:interceptors>: 配置拦截器链(多个)
20              <mvc:interceptor>: 定义拦截器
21                  <mvc:mapping path="/**"/>: 拦截的资源路径-@RequestMapping
22              <bean>: 拦截器定义
```



```
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>
```

#### 4.2.3 执行流程

