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

```
<?xml version="1.0" encoding="UTF-8"?>

<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">
  <parent>
    <artifactId>mvc2</artifactId>
    <groupId>com.itheima</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>mvc-day02-controller</artifactId>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.1.7.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
```

```
    <!-- Json转换由jackson完成 -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.9.9</version>
    </dependency>
  </dependencies>
</project>
```

- com.itheima.controller.JsonController

```
package com.itheima.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * Json格式数据交互案例代码
 * @RestController: 等于@Controller+@ResponseBody
 */
@Controller
public class JsonController {



    /**
     * @RequestBody: 将请求的json数据解析成对象 (默认由jackson提供支持)
     * @ResponseBody: 将返回的对象转换成json数据 (默认由jackson提供支持)
     *
     */
    @ResponseBody
    @RequestMapping("hello")
    public Object hello(@RequestBody Order order) {
        System.out.println(order);
        return order;
    }
}

```

- index.jsp

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

```
    </div>
</body>
<script>
    function submitJson(){
        // 1. 创建异步请求对象
        let req = new XMLHttpRequest();
        // 2. 打开请求
        req.open("POST", "/hello.do", true);
        // 3. 设置请求的数据类型(必须)
        req.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
        // 4. 异步读取响应
        req.onreadystatechange = function(){
            // 2.1 判断响应状态
            if(req.readyState==4 || req.status==200){
                let data = req.responseText;
                document.getElementById("jsonRes").innerHTML=data;
            }
        };
        // 5. 封装数据
        let es = document.getElementById("jsonForm").getElementsByTagName("input");
        let json = {};
        for (let i=0; i<=es.length; i++){
            if(es[i]!=undefined){
                let name = es[i].name;
                let value = es[i].value;
                json[name] = value;
            }
        }
        // 6. 发送数据
        let body = JSON.stringify(json);
        req.send(body);
    }
</script>
</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method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

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">


    <!-- 1. 使用统一的Servlet处理请求 -->
```

```
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <!-- 2. 定义前端控制器统一处理请求 -->
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springMVC.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>


    <!-- 1. 配置需要过滤器的资源 -->
    <filter-mapping>
        <filter-name>hiddenHttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- 2. 使用filter实现对PUT,DELETE的支持 -->
    <filter>
        <filter-name>hiddenHttpMethodFilter</filter-name>
        <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
    </filter>
</web-app>
```

- index.jsp

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

- com.itheima.controller.RestController

```
package com.itheima.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.bind.annotation.ResponseBody;

```

```
 */
@Controller
public class RestController {


    /**
     * 提交PUT方法
     *  注意1: 需要使用HiddenHttpMethodFilter过滤请求
     *  注意2: 只支持返回json类型的数据
     */
    @RequestMapping(value = "hello", method = RequestMethod.PUT)
    @ResponseBody
    public String hello(Integer id, String name){
        System.out.println(id);
        System.out.println(name);
        return "success";
    }

    /**
     * 提交DELETE方法
     *  注意1: 需要使用HiddenHttpMethodFilter过滤请求
     *  注意2: 只支持返回json类型的数据
     */
    @RequestMapping(value = "hello", method = RequestMethod.DELETE)
    @ResponseBody
    public String hello2(Integer id, String name){
        System.out.println(id);
        System.out.println(name);
        return "success";
    }
}
```

5.3.2 特点二: 地址传参

5.3.2 @PathVariable

- com.itheima.controller.RestController

```
/**
     * @PathVariable: 绑定URL地址中的参数
     *  value: 参数名称
     *  required: 默认true,必传;
     *
     *  请求示例: http://localhost:8080/hello/1/Jason
     */
@RequestMapping(value = "hello/{id}/{name}")
@ResponseBody
public String hello3(@PathVariable Integer id, @PathVariable("name") String
name){
    System.out.println(id);
    System.out.println(name);
    return "success";
}
```

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

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

6.1.2 后端代码

添加依赖: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<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">
  <parent>
    <artifactId>mvc2</artifactId>
    <groupId>com.itheima</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>mvc-day02-controller</artifactId>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.1.7.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
    <!-- Json转换由jackson完成 -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.9.9</version>
    </dependency>
```

```
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.4</version>
    </dependency>
  </dependencies>
</project>
```

处理请求: UploadController

```
package com.itheima.controller;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.util.List;
import java.util.UUID;

/**
 * 文件上传案例代码.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Controller
public class UploadController {

    /**
     * 传统文件上传
     *  需要使用commons-fileupload组件的原生API解析请求并处理文件存储
     */
    @RequestMapping("upload")
    public String hello(HttpServletRequest request) throws Exception {
        // 1. 指定上传目录
        String path =
request.getSession().getServletContext().getRealPath("/upload");
        File file = new File(path);
        if (!file.exists()) {
            file.mkdir();
        }
        // 2. 创建组件工具
        DiskFileItemFactory factory = new DiskFileItemFactory();
        ServletFileUpload upload = new ServletFileUpload(factory);
        // 3. 解析请求数据
        List<FileItem> items = upload.parseRequest(request);
        for (FileItem item : items) {
            // 判断是否是普通表单字段
            //
            if (item.isFormField()) {
                System.out.println("普通字段: " + item.getString("UTF-8"));
            } else {
```

```
                item.write(new File(path + "/" + id + item.getName()));
            }
        }
        return "success";
    }
}
```

6.2 SpringMVC

6.2.1 页面

- index.jsp

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

6.2.2 配置

- pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<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">
  <parent>
    <artifactId>mvc2</artifactId>
    <groupId>com.itheima</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>mvc-day02-controller</artifactId>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.1.7.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
```

```
      <artifactId>jackson-databind</artifactId>
      <version>2.9.9</version>
    </dependency>
    <!-- 文件上传的组件支持 -->
    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.4</version>
    </dependency>
  </dependencies>
</project>
```

- springMVC.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:mvn="http://www.springframework.org/schema/mvc"
       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">

    <!-- 1. 开启注解扫描 -->
    <context:component-scan base-package="com.itheima.controller"/>
    <!-- 2. 开启注解驱动 (配置了处理器映射器, 处理器适配器等) -->
    <mvn:annotation-driven/>
    <!-- 3. 配置视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
    <!-- 4. 配置文件解析器-->
    <!-- id: 必须固定为 multipartResolver -->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <!-- maxUploadSize: 设置上传的最大值: 10M(10M*1024KB*1024B) -->
        <property name="maxUploadSize" value="10485760"/>
    </bean>
</beans>
```

6.2.3 代码

- com.itheima.controller.UploadController

```
/**
 * SpringMVC文件上传
 *  基于commons-fileupload组件处理的请求
 */
@RequestMapping("upload")
```

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

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

```
@Controller
public class UserController {

    @Autowired
    UserService userService;

    @RequestMapping("list")
    public String list(){
        try{
            userService.findAll();
        }catch(Exception e){
            // 异常处理
            return "error";
        }
        return "success";
    }
```

- 在servlet过滤器(Filter)调用代码上捕捉异常

```
public class ExceptionFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain){
        try{
            chain.doFilter(request, response);
        }catch(Exception e){
            // 异常处理
        }
    }
}
```

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

```
package com.itheima.ex;

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

/**
 * 模拟查询所有用户异常.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Controller
public class UserController {


    @RequestMapping("list")
    public String list(Model model){
        model.addAttribute("msg", "用户列表");
        // 希望抛出异常进行友好提示: 返回error.jsp页面
        int i = 1/0;
        return "success";
    }
}
```

- com.itheima.ex.CustomExceptionHandler

```
package com.itheima.ex;

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 httpServletRequest 请求
24      * @param httpServletResponse 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
   httpServletRequest,
31                                          HttpServletResponse
   httpServletResponse,
32                                          Object o, Exception e) {
33         ModelAndView mv = new ModelAndView();
34         mv.setViewName("error");
35         mv.addObject("msg", e.getMessage());
36         return mv;
37     }
38 }
```

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

```
package com.itheima.interceptor;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 自定义拦截器.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
public class CustomHandlerInterceptor implements HandlerInterceptor {

    /**
     * Controller方法调用钱执行的方法 【前置通知】
     * @return 是否执行处理器方法
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
        System.out.println("1. 先执行preHandle方法..");
        return false;
    }

    /**
     * Controller方法调用成功后执行的方法 【后置通知】
     *  需要preHandle方法返回true
     *  并且Controller方法没有异常才执行
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("3. 后执行postHandle方法..");
    }

    /**
     * Controller方法执行完成后执行的方法 【最终通知】
     *  需要preHandle方法返回true
```

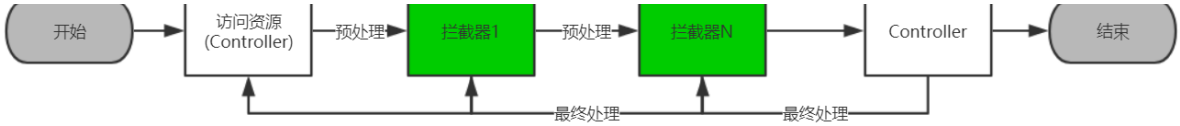
```
    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) throws
Exception {
        System.out.println("4. 最终执行afterCompletion方法..");
    }
}
```

3.2.2 配置拦截器

- springMVC.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"
       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">


    <context:component-scan base-package="com.itheima.ex"/>

    <mvc:annotation-driven/>

    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <!--
        <mvc:interceptors>: 配置拦截器链(多个)
            <mvc:interceptor>: 定义拦截器
                <mvc:mapping path="/**"/>: 拦截的资源路径-@RequestMapping
                <bean: 拦截器定义

    -->
    <mvc:interceptors>
        <mvc:interceptor>
            <!-- path: 拦截的资源路径 -->
            <mvc:mapping path="/**"/>
            <!-- class: 拦截器类路径 -->
            <bean
class="com.itheima.interceptor.CustomHandlerInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>
</beans>
```

3.2.3 执行流程


开始
访问资源
(Controller)
预处理
拦截器1
预处理
拦截器N
Controller
结束
最终处理
最终处理