

SpringMVC框架第二天

第一章：响应数据和结果视图

1. 返回值分类

1. 返回字符串

1. Controller方法返回字符串可以指定逻辑视图的名称，根据视图解析器为物理视图的地址。

```
@RequestMapping(value="/hello")
public String sayHello() {
    System.out.println("Hello SpringMVC!!");
    // 跳转到XX页面
    return "success";
}
```

2. 具体的应用场景

```
@Controller
@RequestMapping("/user")
public class UserController {

    /**
     * 请求参数的绑定
     */
    @RequestMapping(value="/initUpdate")
    public String initUpdate(Model model) {
        // 模拟从数据库中查询的数据
        User user = new User();
        user.setUsername("张三");
        user.setPassword("123");
        user.setMoney(100d);
        user.setBirthday(new Date());
        model.addAttribute("user", user);
        return "update";
    }

}

<h3>修改用户</h3>
${ requestScope }
<form action="user/update" method="post">
    姓名: <input type="text" name="username" value="${ user.username }"><br>
    密码: <input type="text" name="password" value="${ user.password }"><br>
    金额: <input type="text" name="money" value="${ user.money }"><br>
    <input type="submit" value="提交">
</form>
```

2. 返回值是void

1. 如果控制器的方法返回值编写成void，执行程序报404的异常，默认查找JSP页面没有找到。
 1. 默认会跳转到@RequestMapping(value="/initUpdate") initUpdate的页面。
2. 可以使用请求转发或者重定向跳转到指定的页面

```
@RequestMapping(value="/initAdd")
public void initAdd(HttpServletRequest request, HttpServletResponse response) throws
Exception {
    System.out.println("请求转发或者重定向");
    // 请求转发
    // request.getRequestDispatcher("/WEB-INF/pages/add.jsp").forward(request,
response);
    // 重定向
    // response.sendRedirect(request.getContextPath()+"/add2.jsp");

    response.setCharacterEncoding("UTF-8");
    response.setContentType("text/html;charset=UTF-8");

    // 直接响应数据
    response.getWriter().print("你好");
    return;
}
```

3. 返回值是ModelAndView对象

1. ModelAndView对象是Spring提供的一个对象，可以用来调整具体的JSP视图
2. 具体的代码如下

```
/**
 * 返回ModelAndView对象
 * 可以传入视图的名称（即跳转的页面），还可以传入对象。
 * @return
 * @throws Exception
 */
@RequestMapping(value="/findAll")
public ModelAndView findAll() throws Exception {
    ModelAndView mv = new ModelAndView();
    // 跳转到list.jsp的页面
    mv.setViewName("list");

    // 模拟从数据库中查询所有的用户信息
    List<User> users = new ArrayList<>();
    User user1 = new User();
    user1.setUsername("张三");
    user1.setPassword("123");

    User user2 = new User();
    user2.setUsername("赵四");

    user2.setPassword("456");
}
```

```

        users.add(user1);
        users.add(user2);
        // 添加对象
        mv.addObject("users", users);

        return mv;
    }

    <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>

    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    </head>
    <body>

        <h3>查询所有的数据</h3>
        <c:forEach items="${ users }" var="user">
            ${ user.username }
        </c:forEach>

    </body>
    </html>

```

2. SpringMVC框架提供的转发和重定向

1. forward请求转发

1. controller方法返回String类型，想进行请求转发也可以编写成

```

/**
 * 使用forward关键字进行请求转发
 * "forward:转发的JSP路径", 不走视图解析器了，所以需要编写完整的路径
 * @return
 * @throws Exception
 */
@RequestMapping("/delete")
public String delete() throws Exception {
    System.out.println("delete方法执行了...");
    // return "forward:/WEB-INF/pages/success.jsp";
    return "forward:/user/findAll";
}

```

2. redirect重定向

1. controller方法返回String类型，想进行重定向也可以编写成

```
/**
 * 重定向
 * @return
 * @throws Exception
 */
@RequestMapping("/count")
public String count() throws Exception {
    System.out.println("count方法执行了...");
    return "redirect:/add.jsp";
    // return "redirect:/user/findAll";
}
```

3. ResponseBody响应json数据

1. DispatcherServlet会拦截到所有的资源，导致一个问题就是静态资源（img、css、js）也会被拦截到，从而不能被使用。解决问题就是需要配置静态资源不进行拦截，在springmvc.xml配置文件添加如下配置

1. [mvc:resources](#)标签配置不过滤

1. location元素表示webapp目录下的包下的所有文件
2. mapping元素表示以/static开头的所有请求路径，如/static/a 或者/static/a/b

```
<!-- 设置静态资源不过滤 -->
<mvc:resources location="/css/" mapping="/css/**"/> <!-- 样式 -->
<mvc:resources location="/images/" mapping="/images/**"/> <!-- 图片 -->
<mvc:resources location="/js/" mapping="/js/**"/> <!-- javascript -->
```

2. 使用@RequestBody获取请求体数据

```
// 页面加载
// 页面加载
$(function(){
    // 绑定点击事件
    $("#btn").click(function(){
        $.ajax({
            url:"user/testJson",
            contentType:"application/json;charset=UTF-8",
            data: '{"addressName":"aa","addressNum":100}',
            dataType:"json",
            type:"post",
            success:function(data){
                alert(data);
                alert(data.addressName);
            }
        });
    });
});

/**
```

```

    * 获取请求体的数据
    * @param body
    */
@RequestMapping("/testJson")
public void testJson(@RequestBody String body) {
    System.out.println(body);
}

```

3. 使用@RequestBody注解把json的字符串转换成JavaBean的对象

```

// 页面加载
// 页面加载
$(function(){
    // 绑定点击事件
    $("#btn").click(function(){
        $.ajax({
            url:"user/testJson",
            contentType:"application/json;charset=UTF-8",
            data:'{"addressName":"aa","addressNum":100}',
            dataType:"json",
            type:"post",
            success:function(data){
                alert(data);
                alert(data.addressName);
            }
        });
    });
});

/**
 * 获取请求体的数据
 * @param body
 */
@RequestMapping("/testJson")
public void testJson(@RequestBody Address address) {
    System.out.println(address);
}

```

4. 使用@ResponseBody注解把JavaBean对象转换成json字符串，直接响应

1. 要求方法需要返回JavaBean的对象

```

// 页面加载
$(function(){
    // 绑定点击事件
    $("#btn").click(function(){
        $.ajax({
            url:"user/testJson",
            contentType:"application/json;charset=UTF-8",

            data:'{"addressName":"哈哈","addressNum":100}',

```

```

        dataType:"json",
        type:"post",
        success:function(data){
            alert(data);
            alert(data.addressName);
        }
    });
});
});

@RequestMapping("/testJson")
public @ResponseBody Address testJson(@RequestBody Address address) {
    System.out.println(address);
    address.setAddressName("上海");
    return address;
}

```

5. json字符串和JavaBean对象互相转换的过程中，需要使用jackson的jar包

```

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.9.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.9.0</version>
</dependency>

```

第二章：SpringMVC实现文件上传

1. 文件上传的回顾

1. 导入文件上传的jar包

```

<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>

```

2. 编写文件上传的JSP页面

```

<h3>文件上传</h3>

<form action="user/fileupload" method="post" enctype="multipart/form-data">
    选择文件: <input type="file" name="upload"/><br/>
    <input type="submit" value="上传文件"/>
</form>

```

3. 编写文件上传的Controller控制器

```

/**
 * 文件上传
 * @throws Exception
 */
@RequestMapping(value="/fileupload")
public String fileupload(HttpServletRequest request) throws Exception {
    // 先获取到要上传的文件目录
    String path = request.getSession().getServletContext().getRealPath("/uploads");
    // 创建File对象，一会向该路径下上传文件
    File file = new File(path);
    // 判断路径是否存在，如果不存在，创建该路径
    if(!file.exists()) {
        file.mkdirs();
    }
    // 创建磁盘文件项工厂
    DiskFileItemFactory factory = new DiskFileItemFactory();
    ServletFileUpload fileUpload = new ServletFileUpload(factory);
    // 解析request对象
    List<FileItem> list = fileUpload.parseRequest(request);
    // 遍历
    for (FileItem fileItem : list) {
        // 判断文件项是普通字段，还是上传的文件
        if(fileItem.isFormField()) {

        }else {
            // 上传文件项

```

```

        // 获取到上传文件的名称
        String filename = fileItem.getName();
        // 上传文件
        fileItem.write(new File(file, filename));
        // 删除临时文件
        fileItem.delete();
    }
}

return "success";
}

```

2. SpringMVC传统方式文件上传

1. SpringMVC框架提供了MultipartFile对象，该对象表示上传的文件，要求变量名称必须和表单file标签的name属性名称相同。

2. 代码如下

```

/**
 * SpringMVC方式的文件上传
 *
 * @param request
 * @return
 * @throws Exception
 */
@RequestMapping(value="/fileupload2")
public String fileupload2(HttpServletRequest request,MultipartFile upload) throws
Exception {
    System.out.println("SpringMVC方式的文件上传...");
    // 先获取到要上传的文件目录
    String path = request.getSession().getServletContext().getRealPath("/uploads");
    // 创建File对象，一会向该路径下上传文件
    File file = new File(path);
    // 判断路径是否存在，如果不存在，创建该路径
    if(!file.exists()) {
        file.mkdirs();
    }
    // 获取到上传文件的名称
    String filename = upload.getOriginalFilename();
    String uuid = UUID.randomUUID().toString().replaceAll("-", "").toUpperCase();
    // 把文件的名称唯一化
    filename = uuid+"_"+filename;
    // 上传文件
    upload.transferTo(new File(file,filename));
    return "success";
}

```

3. 配置文件解析器对象


```
<!-- 配置文件解析器对象，要求id名称必须是multipartResolver -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="10485760"/>
</bean>
```

3. SpringMVC跨服务器方式文件上传

1. 搭建图片服务器

1. 根据文档配置tomcat9的服务器，现在是2个服务器
2. 导入资料中day02_springmvc5_02image项目，作为图片服务器使用

2. 实现SpringMVC跨服务器方式文件上传

1. 导入开发需要的jar包

```
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-core</artifactId>
    <version>1.18.1</version>
</dependency>
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
    <version>1.18.1</version>
</dependency>
```

2. 编写文件上传的JSP页面

```
<h3>跨服务器的文件上传</h3>

<form action="user/fileupload3" method="post" enctype="multipart/form-data">
    选择文件: <input type="file" name="upload"/><br/>
    <input type="submit" value="上传文件"/>
</form>
```

3. 编写控制器

```
/**
 * SpringMVC跨服务器方式的文件上传
 *
 * @param request
 * @return
 * @throws Exception
 */

@RequestMapping(value="/fileupload3")
```

```

public String fileupload3(MultipartFile upload) throws Exception {
    System.out.println("SpringMVC跨服务器方式的文件上传...");

    // 定义图片服务器的请求路径
    String path = "http://localhost:9090/day02_springmvc5_02image/uploads/";

    // 获取到上传文件的名称
    String filename = upload.getOriginalFilename();
    String uuid = UUID.randomUUID().toString().replaceAll("-", "").toUpperCase();
    // 把文件的名称唯一化
    filename = uuid+"_"+filename;
    // 向图片服务器上传文件

    // 创建客户端对象
    Client client = Client.create();
    // 连接图片服务器
    WebResource webResource = client.resource(path+filename);
    // 上传文件
    webResource.put(upload.getBytes());
    return "success";
}

```

第三章：SpringMVC的异常处理

1. 异常处理思路

1. Controller调用service，service调用dao，异常都是向上抛出的，最终有DispatcherServlet找异常处理器进行异常的处理。

2. SpringMVC的异常处理

1. 自定义异常类

```

package cn.itcast.exception;

public class SysException extends Exception{

    private static final long serialVersionUID = 4055945147128016300L;

    // 异常提示信息
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public SysException(String message) {
        this.message = message;
    }
}

```

```
}
```

2. 自定义异常处理器

```
package cn.itcast.exception;

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

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

/**
 * 异常处理器
 * @author rt
 */
public class SysExceptionHandler implements HandlerExceptionResolver{

    /**
     * 跳转到具体的错误页面的方法
     */
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse
response, Object handler,
        Exception ex) {
        ex.printStackTrace();
        SysException e = null;
        // 获取到异常对象
        if(ex instanceof SysException) {
            e = (SysException) ex;
        }else {
            e = new SysException("请联系管理员");
        }
        ModelAndView mv = new ModelAndView();
        // 存入错误的提示信息
        mv.addObject("message", e.getMessage());
        // 跳转的Jsp页面
        mv.setViewName("error");
        return mv;
    }

}
```

3. 配置异常处理器

```
<!-- 配置异常处理器 -->
<bean id="sysExceptionHandler" class="cn.itcast.exception.SysExceptionHandler"/>
```

第四章：SpringMVC框架中的拦截器

1. 拦截器的概述

1. SpringMVC框架中的拦截器用于对处理器进行预处理和后处理的技术。
2. 可以定义拦截器链，连接器链就是将拦截器按着一定的顺序结成一条链，在访问被拦截的方法时，拦截器链中的拦截器会按着定义的顺序执行。
3. 拦截器和过滤器的功能比较类似，有区别
 1. 过滤器是Servlet规范的一部分，任何框架都可以使用过滤器技术。
 2. 拦截器是SpringMVC框架独有的。
 3. 过滤器配置了/*，可以拦截任何资源。
 4. 拦截器只会对控制器中的方法进行拦截。
4. 拦截器也是AOP思想的一种实现方式
5. 想要自定义拦截器，需要实现HandlerInterceptor接口。

2. 自定义拦截器步骤

1. 创建类，实现HandlerInterceptor接口，重写需要的方法

```
package cn.itcast.demo1;

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

import org.springframework.web.servlet.HandlerInterceptor;

/**
 * 自定义拦截器1
 * @author rt
 */
public class MyInterceptor1 implements HandlerInterceptor{

    /**
     * controller方法执行前，进行拦截的方法
     * return true放行
     * return false拦截
     * 可以使用转发或者重定向直接跳转到指定的页面。
     */
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler)
        throws Exception {
        System.out.println("拦截器执行了...");
        return true;
    }

}
```

2. 在springmvc.xml中配置拦截器类

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 哪些方法进行拦截 -->
        <mvc:mapping path="/user/*"/>
        <!-- 哪些方法不进行拦截 -->
        <mvc:exclude-mapping path=""/>
        -->
        <!-- 注册拦截器对象 -->
        <bean class="cn.itcast.demo1.MyInterceptor1"/>
    </mvc:interceptor>
</mvc:interceptors>
```

3. HandlerInterceptor接口中的方法

1. preHandle方法是controller方法执行前拦截的方法
 1. 可以使用request或者response跳转到指定的页面
 2. return true放行，执行下一个拦截器，如果没有拦截器，执行controller中的方法。
 3. return false不放行，不会执行controller中的方法。
2. postHandle是controller方法执行后执行的方法，在JSP视图执行前。
 1. 可以使用request或者response跳转到指定的页面
 2. 如果指定了跳转的页面，那么controller方法跳转的页面将不会显示。
3. postHandle方法是在JSP执行后执行
 1. request或者response不能再跳转页面了

3. 配置多个拦截器

1. 再编写一个拦截器的类
2. 配置2个拦截器

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 哪些方法进行拦截 -->
        <mvc:mapping path="/user/*"/>
        <!-- 哪些方法不进行拦截 -->
        <mvc:exclude-mapping path=""/>
        -->
```

```
<!-- 注册拦截器对象 -->
<bean class="cn.itcast.demo1.MyInterceptor1"/>
</mvc:interceptor>

<mvc:interceptor>
  <!-- 哪些方法进行拦截 -->
  <mvc:mapping path="/**"/>
  <!-- 注册拦截器对象 -->
  <bean class="cn.itcast.demo1.MyInterceptor2"/>
</mvc:interceptor>
</mvc:interceptors>
```