Spring MVC第二部分

回顾：

1. SpringMVC核心理论
2. SpringMVC基础配置
3. Spring常用注解
4. Spring的其他常用操作

今天任务

1. Controller中向页面传值的几种方式
2. Controller中使用Servlet API
3. RESTful风格编码
4. SpringMVC中处理静态资源
5. SpringMVC中的核心组件
6. SpringMVC中处理JSON
7. SpringMVC中异常处理
8. SpringMVC中拦截器
9. SpringMVC中的文件上传下载

教学目标

1. 掌握Controller中向页面传值的几种方式
2. 掌握Controller中使用Servlet API
3. 掌握RESTful风格编码
4. 掌握SpringMVC中处理静态资源
5. 掌握SpringMVC中的核心组件
6. 掌握SpringMVC中处理JSON
7. 掌握SpringMVC中异常处理
8. 掌握SpringMVC中拦截器
9. 掌握SpringMVC中的文件上传下载

一. controller向页面传递数据

实现方案三种:

第一种: ModelAndView

第二种: Map

第三种: Model

第一种:

java代码:

```
@RequestMapping("/edit")
public ModelAndView edit(){
    User user = new User();
    user.setUsername("用户名!");
    user.setAge(11);
```

```
    user.setPassword("123");

    Address address = new Address();
    address.setProvince("辽宁");
    address.setCity("葫芦岛");

    user.setAddress(address);

    ModelAndView modelAndView = new ModelAndView();
    //viewname 指定要查找的页面
    modelAndView.setViewName("user/form");
    //添加modelAndView添加携带的user数据
    modelAndView.addObject("user", user);

    return modelAndView;
}
```

页面显示代码:

/WEB-INF/user/form.jsp

直接使用el表达式!

```
<form action="/spring-mvc-22/user/add" method="post">
      <label for="username">用户名:<input type="text" value="${user.username }"
id="username" name="username" /></label><br/>
      <label for="password">密码:<input type="password" id="password"
name="password" /></label><br/>
      <label for="age">年龄:<input type="text" value="${user.age }" name="age" />
</label><br/>
      <label for="address.province">省份:<input type="text"
value="${user.address.province }" name="address.province" /></label><br/>
      <label for="address.city">城市:<input type="text" value="${user.address.city
}" name="address.city" /></label><br/>
      <label for="address.sp.pname">特产名称:<input type="text"
name="address.sp.pname" /></label><br/>
      <label for="address.sp.price">特产价格:<input type="text"
name="address.sp.price" /></label><br/>
      <input type="submit"  value="提交" />
</form>
```

第二种

Java代码:

```
@RequestMapping("/map")
public String retMap(Map<String,Object> map){

     map.put("name","map return");

     return "show";
}
```

方法中添加 map参数, 往map中放置数据,就可以发送到jsp页面!

显示代码如上!

第三种

将map替换成model即可!

java代码:

```
@RequestMapping("/model")
public String retModel(Model model){
   model.addAttribute("name","model return!");
   return "show";
}
```

总结: 使用以上三种情况可以将数据返回给页面,页面使用EL表达式即可获取!但是要注意!数据放入的是requestScope域!其他域获取不到!

验证代码:

```
${name}
${requestScope.name}
${sessionScope.name}

结果显示值!前两个显示!sessionScope不显示!
model return! model return!
```

如果需要将在sessionScope赋值一份!可以利用@SessionAttributes属性!

```
@SessionAttributes(names = "name")  //names可以选择多个值,但是必须是已有的命名!
@Controller
@RequestMapping("/user")
public class UserController {
```

二. 控制器方法中使用原生API

如果我们需要使用Servlet内部常用类:

- HttpServletRequest
- HttpServletResponse
- HttpSession 等

直接在Controller层的方法中传入对应参数即可!不分顺序!

注意:如果使用maven项目 需要导入 jsp jstl servlet api

```
 <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
 </dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.2.1</version>
    <scope>provided</scope>
</dependency>
```

Java代码示例:

```
  @RequestMapping("/slt")
  public  String getSlt(HttpServletRequest request){
    //TODO 如果需要引入原生api类!直接在参数引入即可!
    return "show";
  }
```

三. RESTful风格编码

1. RESTful介绍

REST:即Representational State Transfer , (资源)表现层状态转化,是目前最流行的一种互联网软件架构。它结构清晰、符合标注、易于理解、方便扩展，所以越来越多的网站采用!

具体说，就是HTTP协议里面,四个表示操作方式的动词:

GET POST PUT DELETE

它们分别代表着四种基本操作:

- GET用来获取资源
- POST用来创建新资源
- PUT用来更新资源
- DELETE用来删除资源

示例:

- /order/1 HTTP GET :得到id = 1 的 order
- /order/1 HTTP DELETE: 删除 id=1 的order
- /order/1 HTTP PUT : 更新id = 1的 order
- /order HTTP POST : 新增 order

2. Spring中实现RESTful风格

HiddenHttpMethodFilter:浏览器form表单只支持GET和POST,不支持DELETE和PUT请求,Spring添加了一个过滤器,可以将这些请求转换为标准的http方法,支持GET,POST,DELETE,PUT请求!

3.具体实现

3.1 web.xml添加HiddenHttpMethodFilter配置

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

3.2 实现查,改,删 框架!

```
@Controller
@RequestMapping("/order")
public class OrderController
{
    @RequestMapping(value = "/list",method = RequestMethod.GET)
    public  String list(){
        //获取用户列表
        return "order/list";
```

```
    }

    @RequestMapping(value = "/{id}",method = RequestMethod.DELETE)
    public  String delete(@PathVariable("id") Integer id){
        //执行删除操作
        System.out.println("id = " + id);
        return "redirect:/order/list";
    }


    @RequestMapping(value = "/{id}",method = RequestMethod.PUT)
    public  String update(@PathVariable("id") Integer id){
        //执行更新操作
        System.out.println("id = " + id);
        return "redirect:/order/list";
    }

}
```

Jsp代码:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
     <h1>获取用户列表</h1>
     <a href="javascript:void(0)" onclick="deleteById()">删除</a>
     <form action="/order/1" method="post" id="deleteForm">
           <input type="hidden" name="_method" value="DELETE" />
     </form>


     <a href="javascript:void(0)" onclick="updateById()">修改</a>
     <form action="/order/1" method="post" id="updateForm">

         <input type="hidden" name="_method" value="PUT" />

     </form>

     <script>
          function updateById() {
              var form = document.getElementById("updateForm");
              form.submit();
          }
```

```
        function deleteById() {
           //TODO 删除
            var form = document.getElementById("deleteForm");
            form.submit();
        }

    </script>

</body>
</html>
```

需要注意: 由于doFilterInternal方法只对method为post的表单进行过滤，所以在页面中必须如下设置:

```
  <form action="..." method="post">
        <input type="hidden" name="_method" value="put" />
  </form>
```

代表post请求,但是HiddenHttpMethodFilter将把本次请求转化成标准的put请求方式!
name="_method"固定写法!

测试: 查看方法可以调通即可!

3.3 使用RESTful风格 CRUD操作!

1. 添加Order Pojo类!

```
public class Order {

    private Integer id;
    private String  code;
    private Double  money;

    public Order() {
    }

    public Order(Integer id, String code, Double money) {
        this.id = id;
        this.code = code;
        this.money = money;
    }
```

```
    //setter getter toString
}
```

1. 添加OrderDao

```
@Repository
public class OrderDao {

    private static Map<Integer,Order> orders = null;
    //虚拟数据
    static {
        orders = new HashMap<Integer, Order>();
        orders.put(1,new Order(1,"订单1",20.5));
        orders.put(2,new Order(2,"订单2",22.5));
        orders.put(3,new Order(3,"订单3",24.5));
        orders.put(4,new Order(4,"订单4",25.5));
        orders.put(5,new Order(5,"订单5",26.5));
    }

    public static  Integer initId = 6; //记录id

    /**
     * 保存/更新 order
     * @param order
     */
    public  void save(Order order){
        if (order.getId() == null)
        {
            order.setId(initId++);
        }
        orders.put(order.getId(),order);
    }

    /**
     * 返回所有数据
     * @return
     */
    public Collection<Order> getAll(){
        return  orders.values();
    }

    /**
     * 获取数据
     * @param id
     * @return
     */
    public Order get(Integer id){
```

```
        return  orders.get(id);
    }

    /**
     * 删除
     * @param id
     */
    public  void delete(Integer id)
    {
        orders.remove(id);
    }
}
```

1. 修改list.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
    <title>Title</title>
</head>
<body>
     <h1>获取用户列表</h1>
     <form action="/order/1" method="post" id="deleteForm">
         <input type="hidden" name="_method" value="DELETE" />
     </form>

     <form action="/order/1" method="post" id="updateForm">
         <input type="hidden" name="_method" value="PUT" />
     </form>

     <table align="center"  width="60%" border="1px" bordercolor="#FB5832"
CELLSPACING="0">
         <tr>
           <td>id</td>
           <td>订单</td>
           <td>金额</td>
           <td>操作</td>
         </tr>

         <c:forEach items="${orderList }" var="order">
             <tr>
                 <td>${order.id}</td>
                 <td>${order.code}</td>
                 <td>${order.money}</td>
                 <td>
```

```
                    <a href="javascript:void(0)"
onclick="deleteById(${order.id})">删除</a>
                    <a href="javascript:void(0)"
onclick="updateById(${order.id})">修改</a>
                </td>
            </tr>
        </c:forEach>
    </table>

    <center>
        <h3>添加订单</h3>
        <form action="/order/save" method="post">
              <label for="code">订单<input id="code" type="text" name="code" />
</label>
              <label for="money">总金额<input id="money" type="text" name="money"
/></label>
              <button>添加订单</button>
        </form>
    </center>

    <script>

         function updateById(id) {
             //更新订单
             //注意 要动态修改form的action
             var form = document.getElementById("updateForm");
             form.action = "/order/"+id;
             form.submit();
         }

        function deleteById(id) {
           //TODO 删除
            var form = document.getElementById("deleteForm");
            form.action = "/order/"+id;
            form.submit();
        }

    </script>

</body>
</html>
```

1. 完善Controller

   因为练习省略了业务层!直接在Controller层进行调用

```
@Controller
@RequestMapping("/order")
public class OrderController
{
    @Autowired
    private OrderDao orderDao;

    @RequestMapping(value = "/list",method = RequestMethod.GET)
    public  String list(Model model){
        //获取用户列表
        Collection<Order> all = orderDao.getAll();
        model.addAttribute("orderList",all);
        return "order/list";
    }

    @RequestMapping(value = "/{id}",method = RequestMethod.DELETE)
    public  String delete(@PathVariable("id") Integer id){
        //执行删除操作
        System.out.println("id = " + id);
        orderDao.delete(id);
        return "redirect:/order/list";
    }

    @RequestMapping(value = "/{id}",method = RequestMethod.PUT)
    public  String update(@PathVariable("id") Integer id){
        //执行更新操作
        System.out.println("id = " + id);
        orderDao.get(id).setCode("修改过后的订单");
        return "redirect:/order/list";
    }

    @RequestMapping(value = "/save",method = RequestMethod.POST)
    public  String save(Order order){
        //执行更新操作
        orderDao.save(order);
        return "redirect:/order/list";
    }

}
```

四. 处理静态资源

> 需要注意一种,DispatcherServlet拦截资源设置成了 / 避免了死循环,但是 / 不拦截jsp资源,但是它会拦截其他静态资源,例如 html , js , 图片等等, 那么我们在使用jsp内部添加 静态资源就无法成功,所以,我们需要单独处理下静态资源!

处理方案: 在springmvc的配置文件中添加mvc命名空间下的标签!

1. 修改Spring MVC对应配置文件,添加mvc命名空间和约束

```
<?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
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
">
```

2. 添加处理标签

```
  <mvc:annotation-driven />  <!--注解驱动-->
  <mvc:resources mapping="/static/**" location="/static/" ></mvc:resources>
```

配置解释: 将静态资源(图片,css,js,html)放入了webApp/static文件夹下! 直接访问DispatcherServlet会拦截出现404问题!

location元素表示webapp目录下的static包下的所有文件;

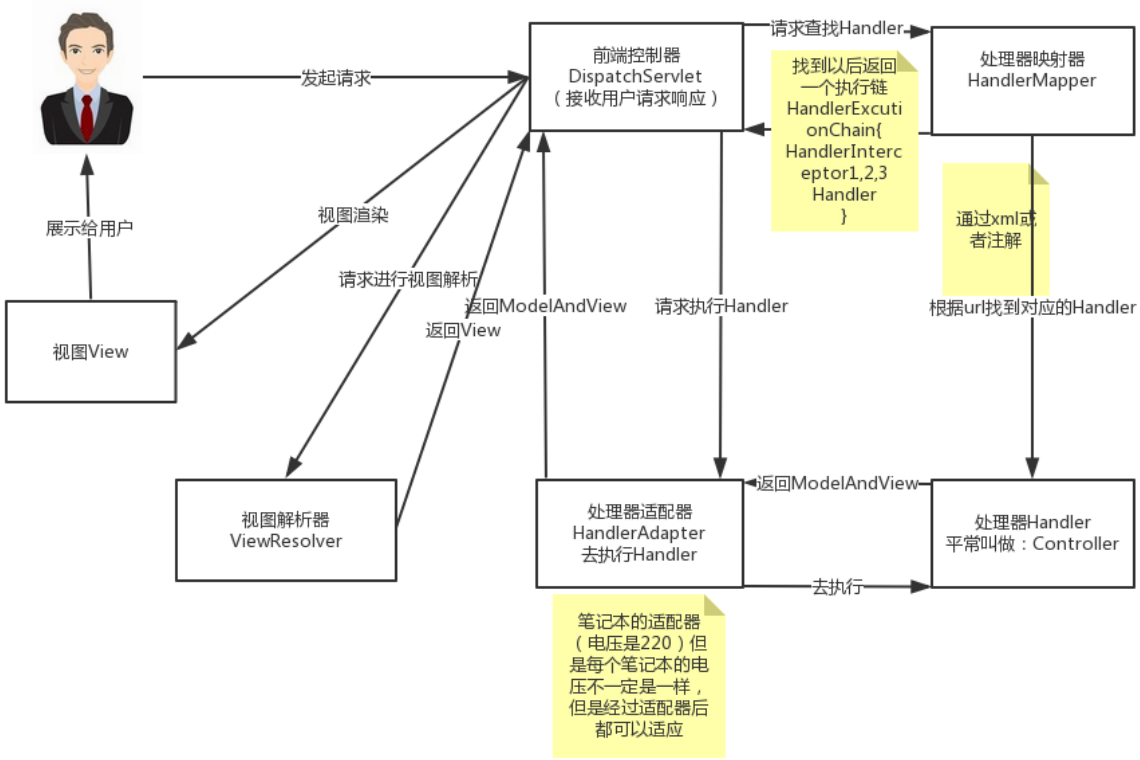
mapping元素表示将 location对应文件加下的资源对应到 /static路径下!

该配置的作用是：DispatcherServlet不会拦截以/static开头的所有请求路径，并当作静态资源

交由Servlet处理

五. Spring MVC的组件

1. SpringMVC工作原理



2. 组件回顾

- DispatcherServlet：作为前端控制器，整个流程控制的中心，控制其它组件执行，统一调度，降低组件之间的耦合性，提高每个组件的扩展性。
- HandlerMapping：通过扩展处理器映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。
- HandlAdapter：通过扩展处理器适配器，支持更多类型的处理器,调用处理器传递参数等工作!
- ViewResolver：通过扩展视图解析器，支持更多类型的视图解析，例如：jsp、freemarker、pdf、excel等。

注意:但是我们之前编写代码只添加了DispatcherServlet和ViewResolver,但是没有涉及到HandlerMapping和HandlerAdapter!因为DispatcherServlet默认已经配置了HandlerMapping和HandlerAdapter!所以,我们没有自己定义HandlerMapping和HandlerAdapter!当然我们也可以进行自定义,本章节主要讲解自定义ViewResolver,HandlerMapping和HandlerAdapter。

查看DispatcherServlet的默认配置

文件位置: org/springframework/web/servlet/DispatcherServlet.properties

```
org.springframework.web.servlet.LocaleResolver=org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver

org.springframework.web.servlet.ThemeResolver=org.springframework.web.servlet.theme.FixedThemeResolver
//默认的HandlerMapping
```

```
org.springframework.web.servlet.HandlerMapping=org.springframework.web.servlet.han
dler.BeanNameUrlHandlerMapping,\
    org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping
//默认HandlerAdapter
org.springframework.web.servlet.HandlerAdapter=org.springframework.web.servlet.mvc
.HttpRequestHandlerAdapter,\
    org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter,\
    org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

org.springframework.web.servlet.HandlerExceptionResolver=org.springframework.web.s
ervlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver,\

org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver,\
    org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver

org.springframework.web.servlet.RequestToViewNameTranslator=org.springframework.we
b.servlet.view.DefaultRequestToViewNameTranslator

//默认视图解析器
org.springframework.web.servlet.ViewResolver=org.springframework.web.servlet.view.
InternalResourceViewResolver

org.springframework.web.servlet.FlashMapManager=org.springframework.we
```

3. 配置HandlerAdapter 和 HandlerMapping

> 配置 RequestMappingHandlerAdapter 和 RequestMappingHandlerMpping
>
> 文件上传,返回JSON需要依赖以上适配器和映射器!

配置

```
<!-- 处理器映射器-->
<bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandler
Mapping" />
<!-- 处理器适配器-->
<bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandler
Adapter" />

<mvc:annotation-driven /> <!--可以替换以上两种! -->
```

解释: <mvc:annotaion-driveb /> 默认对应的处理类是 org.springframework.web.servlet.config.AnnotationDrivenBeanDefinitionParser ;

```
@Override
    public BeanDefinition parse(Element element, ParserContext parserContext) {
        ....
        //加载: RequestMappingHandlerMapping
        RootBeanDefinition handlerMappingDef = new
RootBeanDefinition(RequestMappingHandlerMapping.class);

        //加载:RequestMappingHandlerAdapter
        RootBeanDefinition handlerAdapterDef = new
RootBeanDefinition(RequestMappingHandlerAdapter.class);
        handlerAdapterDef.setSource(source);
        handlerAdapterDef.setRole(BeanDefinition.ROLE_INFRASTRUCTURE);
        ...
```

六.返回JSON数据

访问控制器返回Json类型数据!

1. 导入对应的JSON包!

支持:

jackson : jackson-databind/jackson-annotations/jack-core

gson: gson

注意: jackson需要三个jar包!如果使用maven引入jackson-databind会连带引入 core和 annotations。

如果非maven项目,必须加入三个jar包!

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.8.9</version>
</dependency>
```

或者:

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.2.4</version>
</dependency>
```

2. 实现代码

主要使用@ResponseBody

```
@ResponseBody
@RequestMapping("/json")
public List<User> getJson(){
  //TODO 返回JSON数据,返回List或者对象都可以!
  //ResponseBody都可以转成JSON类型
    User user = new User();
    user.setName("测试1");
    user.setAge("12");
    user.setBirthday(new Date());

    List<User> list = new ArrayList<User>();
    list.add(user);
    list.add(user);

    return  list;
}
```

3. 回顾Spring MVC返回值类型

- String
  - 情况1: 查找到指定的视图

    return "user/show";
  - 情况2: 转发或者重定向

    return "redirect: path";

    return "forword:path";
- ModelAndView

  返回数据视图模型对象!

  ModelAndView mv = new ModelAndView();

  mv.setViewName("查找视图路径");

  mv.addObject("key","object type data");

- Object

  配合@ResponseBody返回Json数据类型!

- void

  可以返回其他mimetype类型的数据!通常将方法定义成void!

  配合方法传参得到Response对象,通过Response.getWriter().writer("数据");

七.Spring MVC异常处理

1. Spring MVC异常处理介绍

Spring MVC通过HandlerExceptionResolver处理程序的异常,包括处理映射,数据绑定及处理器执行时发生异常。HandlerExceptionResolver仅有一个接口方法:

ModelAndView resolveException(HttpServletRequest reqeust,HttpServletResponse response,Object handler,Exception ex);

当发生异常时,Spring MVC将调用 resolveException()方法,并转到ModelAndView对应视图中,作为一个异常报告页面,反馈给用户!

HandlerExceptionResolver拥有4个实现类:

- DefaultHandlerExceptionResolver
- SimpleMappingExceptionResolver
- AnnotationMethodHandlerExceptionResolver
- ResponseStatusExceptionResolver

2. 异常处理方案

2.1 DefaultHandlerExceptionResolver

Spring MVC默认装配了DefaultHandlerExceptionResolver,它会将Spring MVC框架的异常转换为相应的相应状态码!

异常和相应状态码对应表

异常类型	响应状态码
ConversionNotSupportedException	500(Web服务器内部错误)
HttpMediaTypeNotAcceptableException	406(无和请求accept匹配的MIME类型)
HttpMediaTypeNotSupportedException	415(不支持MIME类型)
HttpMessageNotReadableException	400
HttpMessageNotWritableException	500
HttpRequestMethodNotSupportedException	405
MissingServletRequestParameterException	400

在web.xml响应状态码配置一个对应页面

```
<error-page>
   <error>404</error>
   <location>/404.html</location>
</error-page>
```

注意: 静态资源注意会被DispatcherServlet拦截!

2.2 AnnotationMethodHandlerExceptionResolver

Spring MVC 默认注册了 AnnotationMethodHandlerExceptionResolver,它允许通过@ExceptionHandler注解指定处理特定异常的方法!

```
 @ExceptionHandler
 public  String  handleException(RuntimeException re, HttpServletRequest request)
 {
    return "forward:/user/error";
 }
```

通过@ExceptionHandler指定了当前类的一个错误处理方法!如果当前类中出现异常,会触发错误处理方法!

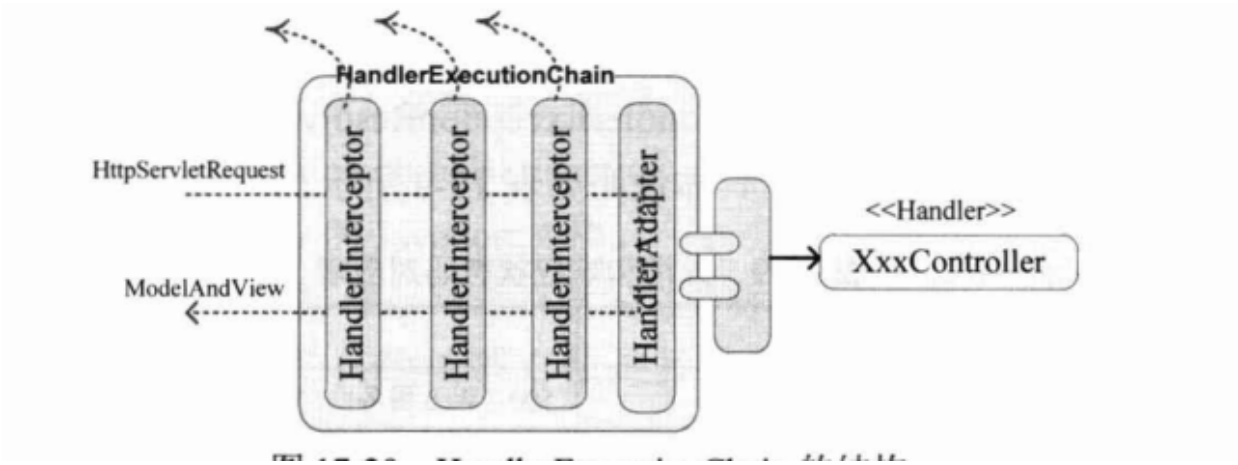
但是@ExceptionHandler的异常处理方法只能对同一处理类中的其他处理方法进行异常响应处理!!

2.3 全局异常处理

```
@ControllerAdvice
public class MyExecptionHandler {

    @ExceptionHandler(Exception.class)
    public ModelAndView handleException(Exception ex)
    {
        System.out.println("全局异常:ex = " + ex);
        ModelAndView modelAndView = new ModelAndView();

        modelAndView.setViewName("error");
        modelAndView.addObject("exception",ex);
        return  modelAndView;
    }

}
```

此处可以捕捉全局异常,但是不要忘了在spring配置的时候扫描该类!

八. Spring MVC拦截器实现



Spring MVC 的拦截器类似于Servlet中的拦截器!需要先定义一个类实现HandlerInterceptor接口!

添加未实现的方法,在springmvc配置中配置!具体实现步骤如下:

1. 创建拦截器类

```
public class MyInterceptors implements HandlerInterceptor {

    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        System.out.println("调用之前触发此方法!"+handler);
        /**
         * return false  代表拦截,不调用controller方法直接返回
         *        true   不拦截
         */
        return false;
    }

    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        /**
         * 调用之后调用此方法
         */
        System.out.println("handler已经被调用完毕!"+handler);
    }

    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        /**
         * 响应已经被渲染后,执行该方法.
         */
```

```
        System.out.println("响应已经进行了渲染");
    }
}
```

- boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler):在请求到达Handler之前,先执行这个前置处理方法.当该方法返回false时,请求直接返回,不会传递到链中的下一个拦截器,更不会传递到链尾的Handler,只有返回true时,请求才会向链中的下一个处理节点传递!
- void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView); 在相应已经被渲染后,执行该方法.
- void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex); 在响应已经被渲染后,执行该方法!

2. SpringMVC 配置文件中配置拦截器

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/user/*"/>
        <bean class="com.itqf.spring.interceptors.MyInterceptors" />
    </mvc:interceptor>
</mvc:interceptors>
```

九. Spring MVC处理文件上传

> Spring MVC为文件上传提供了直接支持,这种支持是通过即插即用的MultipartResolver实现.Spring使用Jakarta Commons FileUpload技术实现了一个MultipartResolver实现类:CommonsMultipartResolver。
>
> 在SpringMVC上下文中默认没有装配MultipartResolver,因此默认情况下不能处理文件上传工作。如果想使用Spring的文件上传功能,则需要先在上下文中配置MultipartResolver。

1. 引入jar包!

commons-fileupload.jar

commons-io.jar

maven项目pom.xml

```
<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

2. 配置MultipartResolver

配置

```
 <!--multipartResolver配置-->
 <bean id="multipartResolver"
       class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
       p:defaultEncoding="UTF-8"
       p:maxUploadSize="5242880"
       p:uploadTempDir="file:/d:/temp"
 />
```

3. 编写控制器和文件上传表单

- 编写文件上传表单 upload.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
  <center>
      <form method="post" enctype="multipart/form-data" action="/user/upload">
            <label for="name">文件名称<input type="text" id="name" name="name" />
</label>
            <input type="file" name="file" />
            <button>提交</button>
      </form>
  </center>
</body>
</html>
```

- 编写控制器代码

```
 @RequestMapping("/toUpload")
```

```
 public String toUpload(){
    //跳转到上传页面
    return  "user/upload";
 }

 @RequestMapping("/upload")
 public  String saveFile(@RequestParam("name") String name ,
@RequestParam("file")MultipartFile file) throws IOException {
    //接收表单提交的数据,包含文件
    System.out.println("name = " + name);
    if (!file.isEmpty())
     {
       file.transferTo(new File("G:/temp/"+file.getOriginalFilename()));
     }
   return "success";
 }
```

SpringMVC会将上传文件绑定到MultipartFile对象上. MultipartFile提供了获取长传文件内容,文件名等方法,通过transferTo()方法还可将文件存储到磁盘中,具体方法如下:

方法名称	方法解释
byte [] getBytes()	获取文件数据
String getContentType()	获取文件MIMETYPE类型,如image/jpeg,text/plain等
InputStream getInputStream()	获取文件输入流
String getName()	获取表单中文件组件的名称 name值!
String getOriginalFilename()	获取文件上传的原名
long getSize()	获取文件的字节大小,单位为byte
boolean isEmpty()	是否有长传的文件
void transferTo(File dest)	可以将上传的文件保存到指定的文件中

课前默写

1. SpringMVC基础配置
2. 使用SpringMVC框架进行基础的页面操作

作业

1. 扩展作业的作业内容
2. 添加登录功能，并使用拦截器实现未登录拦截
3. 在员工表里添加员工照片（photo）字段，并实现其上传下载功能
4. 完成异常的处理

面试题

1. 简述Controller中向页面传值的几种方式
2. 简述SpringMVC中RESTful的应用
3. 简述SpringMVC中处理静态资源的方案
4. 简述SpringMVC中的HandlerMapping和HandlerAdapter的作用
5. 简述SpringMVC中JSON的处理方式
6. 简述SpringMVC中异常处理
7. 简述SpringMVC中拦截器的使用
8. 简述SpringMVC中如何进行文件上传下载