Neuedu

SpringMVC框架技术

版本：3.6.0

第4章 数据校验、异常处理和拦截器

东软睿道教育信息技术有限公司

本章教学目标

- 了解异常处理思路；
- 理解数据校验概述；
- 掌握数据校验步骤、分组校验；
- 掌握数据回显；
- 掌握自定义异常；
- 掌握拦截器的定义和配置；

本章教学内容

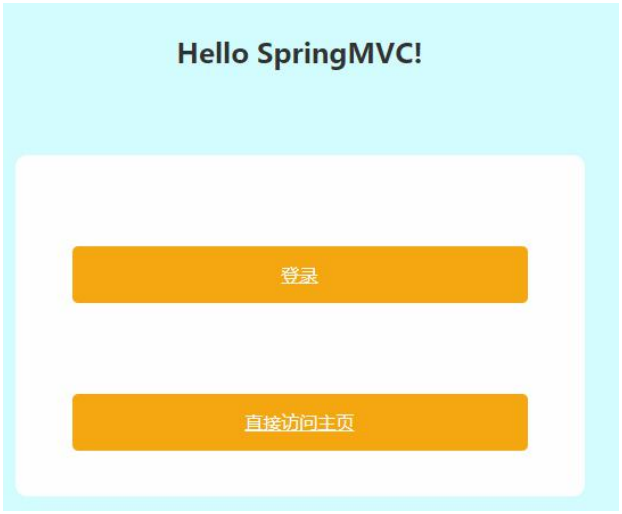
Neuedu

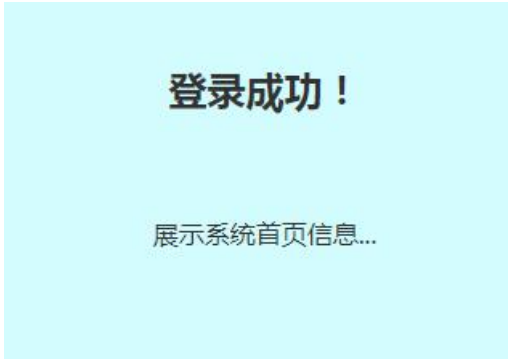
节	知识点	掌握程度	难易程度	教学形式	对应在线微课
数据校验	数据校验概述	理解		线上	数据校验概述
	数据校验步骤	掌握		线上	数据校验步骤
	分组校验	了解	难	线上	分组校验
	数据回显	掌握		线上	数据回显
异常处理	异常处理思路	了解		线上	异常处理思路
	自定义异常	掌握		线上	自定义异常
拦截器	定义拦截器	理解		线下	定义拦截器
	配置拦截器	掌握		线下	配置拦截器
本章实战项目任务实现	实战项目任务实现	掌握	难	线下	实战项目任务实现

本章实战项目任务

- 通过本章内容的学习，完成《跨境电商系统》用户首页访问拦截
  - 登录成功后可以访问用户首页（main.jsp）
  - 用户如果没有登录就访问首页，显示登录页面
  - 用户登录时，对用户名、密码添加校验规则进行校验，校验不通过给出提示

- 运行效果如图：





CONTENTS

目录

数据校验概述

Neuedu

- 项目中，通常使用较多是前端的校验，比如页面中js校验。对于安全要求较高的建议在服务端进行校验。
- 服务端校验：
  - 控制层conroller：校验页面请求的参数的合法性。在服务端控制层conroller校验，不区分客户端类型（浏览器、手机客户端、远程调用）
  - 业务层service（使用较多）：主要校验关键业务参数，仅限于service接口中使用的参数。
  - 持久层dao：一般是不校验的。

数据校验概述

Neuedu

- springmvc使用hibernate的校验框架validation
- 校验思路：
  - 页面提交请求的参数，请求到controller方法中，使用validation进行校验。
  - 如果校验出错，将错误信息展示到页面。

数据校验

- 示例：
  - 使用springmvc完成商品信息修改
  - 添加校验（校验商品名称长度，生产日期的非空校验）
    - 如果校验出错，在商品修改页面显示错误信息。

- 全部代码参见：ch04_springmvc01工程

数据校验步骤

- springmvc数据校验步骤
  - pom.xml导入校验jar文件
  - 配置校验器
  - 校验器注入到处理器适配器中
  - 添加校验规则
  - 错误信息文件
  - 捕获错误信息
  - 显示错误信息

数据校验步骤

Neuedu

- pom.xml配置导入校验jar文件
  - hibernate的校验框架validation所需要jar包

```
52        <dependency>
53            <groupId>org.hibernate</groupId>
54            <artifactId>hibernate-validator</artifactId>
55            <version>4.3.0.Final</version>
56        </dependency>
57        <dependency>
58            <groupId>javax.validation</groupId>
59            <artifactId>validation-api</artifactId>
60            <version>1.0.0.GA</version>
61        </dependency>
62        <dependency>
63            <groupId>org.jboss.logging</groupId>
64            <artifactId>jboss-logging</artifactId>
65            <version>3.1.0.CR2</version>
66        </dependency>
```

数据校验步骤

- 配置校验器
  - 在springmvc配置文件中配置

```
    <!-- 校验器 -->
    <bean id="validator"
        class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
        <!-- hibernate校验器-->
        <property name="providerClass" value="org.hibernate.validator.HibernateValidator" />
        <!-- 指定校验使用的资源文件，在文件中配置校验错误信息，如果不指定则默认使用classpath下的ValidationMessages.pr
        <property name="validationMessageSource" ref="messageSource" />
    </bean>
<!-- 校验错误信息配置文件 -->
    <bean id="messageSource"
        class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <!-- 资源文件名-->
        <property name="basenames">
         <list>
            <value>classpath:CustomValidationMessages</value>
         </list>
        </property>
        <!-- 资源文件编码格式 -->
        <property name="fileEncodings" value="utf-8" />
        <!-- 对资源文件内容缓存时间，单位秒 -->
        <property name="cacheSeconds" value="120" />
    </bean>
```

数据校验步骤

- 校验器注入到处理器适配器中
  - 在springmvc配置文件中配置

```
<mvc:annotation-driven conversion-service="conversionService" validator="validator">
</mvc:annotation-driven>
```

数据校验步骤

Neuedu

- pojo中添加校验规则
  - 在ItemsCustom.java中添加

```
public class Items {

    private Integer id;
    @Size(min=1,max=20,message="{items.name.length.error}")
    private String name;

    private Float price;

    private String pic;
    @NotNull(message="{items.createtime.isNULL}")
    private Date createtime;
```

  - 全部代码参见：ch04_springmvc01工程

数据校验步骤

- 错误信息文件
  - 在CustomValidationMessages.properties配置校验错误信息
  - 把配置文件放到类路径中

```
#添加校验错误提交信息
items.name.length.error=请输入1到30个字符的商品名称
items.createtime.isNUll=请输入 商品的生产日期
```

  - 全部代码参见：ch04_springmvc01工程

数据校验步骤

Neuedu

- 捕获错误信息
  - 在controller方法中捕获
  - 添加@Validated表示在对items参数绑定时进行校验，校验信息写入BindingResult中，在要校验的pojo后边添加BingdingResult， 一个BindingResult对应一个pojo，且BingdingResult放在pojo的后边。

```
@RequestMapping("/editItemsSubmit")
// 在需要校验的pojo前边添加@Validated，在需要校验的pojo后边添加BindingResult
// bindingResult接收校验出错信息
// 注意：@Validated和BindingResult bindingResult是配对出现，并且形参顺序是固定的（一前一后）。
public String editItemsSubmit(Model model,Integer id,
        @Validated ItemsCustom itemsCustom, BindingResult bindingResult) throws Exception {
```

数据校验步骤

Neuedu

```
if(bindingResult.hasErrors()){
    List<ObjectError> allErrors = bindingResult.getAllErrors();
    for(ObjectError objectError:allErrors) {
        System.out.println(objectError.getDefaultMessage());

    }

    // 将错误信息传到页面
    model.addAttribute("allErrors", allErrors);

    // 出错重新到商品修改页面
    return "items/editItems";
```

- 全部代码参见：ch04_springmvc01工程

数据校验步骤

- 显示错误信息
  - 在jsp页面中将错误信息展示

```
<c:if test="${allErrors !=null }">
    <c:forEach items="${allErrors }" var="error">
        ${error.defaultMessage } <br>
    </c:forEach>
</c:if>
```

  - 全部代码参见：ch04_springmvc01工程

分组校验

Neuedu

- 在pojo中定义校验规则，而pojo是被多个 controller所共用，当不同的controller方法对同一个pojo进行校验，但是每个controller方法需要不同的校验。
- 解决方法：
  - 定义多个校验分组（其实是一个java接口），分组中定义有哪些规则
  - 每个controller方法使用不同的校验分组

分组校验

- 定义校验分组
  - 分组就是一个标识，这里定义一个接口

```
public interface ValidGroup1 {
    //接口中不需要定义任何方法，仅是对不同的校验规则进行分组
    //此分组只校验商品名称长度

}
```

  - 全部代码参见：ch04_springmvc02工程

分组校验

- 在校验规则中添加分组
  - 指定分组ValidGroup1，表示此@Size校验只适用ValidGroup1校验

```
private Integer id;
@Size(min=1,max=20,message="{items.name.length.error}",groups={ValidGroup1.class})
private String name;
```

  - 全部代码参见：ch04_springmvc02工程

分组校验

Neuedu

- 在controller方法中使用指定分组的校验
  - 在@Validated中添加value={ValidGroup1.class}表示商品修改使用了ValidGroup1分组校验规则
  - 可以指定多个分组，中间用逗号分隔
    - @Validated(value={ValidGroup1.class, ValidGroup2.class })

```
@RequestMapping("/editItemsSubmit")
// 在需要校验的pojo前边添加@Validated，在需要校验的pojo后边添加BindingResult
// bindingResult接收校验出错信息
// 注意：@Validated和BindingResult bindingResult是配对出现，并且形参顺序是固定的（一前一后）。
public String editItemsSubmit(Model model,Integer id,
        @Validated(value={ValidGroup1.class,ValidGroup2.class}) ItemsCustom itemsCustom, BindingResult bindingResult) throws Exception {
```

- 全部代码参见：ch04_springmvc02工程

校验注解

Neuedu

限制	说明
@Null	限制只能为null
@NotNull	限制必须不为null
@AssertFalse	限制必须为false
@AssertTrue	限制必须为true
@DecimalMax(value)	限制必须为一个不大于指定值的数字
@DecimalMin(value)	限制必须为一个不小于指定值的数字
@Digits(integer,fraction)	限制必须为一个小数，且整数部分的位数不能超过integer，小数部分的位数不能超过fraction
@Future	限制必须是一个将来的日期

校验注解

Neuedu

@Max(value)	限制必须为一个不大于指定值的数字
@Min(value)	限制必须为一个不小于指定值的数字
@Past	限制必须是一个过去的日期
@Pattern(value)	限制必须符合指定的正则表达式
@Size(max,min)	限制字符长度必须在min到max之间
@Past	验证注解的元素值（日期类型）比当前时间早
@NotEmpty	验证注解的元素值不为null且不为空（字符串长度不为0、集合大小不为0）
@NotBlank	验证注解的元素值不为空（不为null、去除首位空格后长度为0），不同于@NotEmpty，@NotBlank只应用于字符串且在比较时会去除字符串的空格
@Email	验证注解的元素值是Email，也可以通过正则表达式和flag指定自定义的email格式

数据回显

- 数据回显
  - 表单提交失败需要再回到表单页面重新填写，原来提交的数据需要重新在页面上显示。
  - 简单数据类型
  - pojo类型

数据回显

- 简单数据类型
  - 对于简单数据类型，如：Integer、String、Float等，使用model.addAttribute( )方法，将传入的参数再放到request域实现显示。

```
@RequestMapping(value="/editItems",method={RequestMethod.GET})
public String editItems(Model model,Integer id)throws Exception{

    //传入的 id 重新放到 request 域
    model.addAttribute("id", id);
```

数据回显

Neuedu

- pojo数据类型
  - springmvc默认支持pojo数据回显
  - springmvc自动将形参中的pojo重新放回request域中，request的key为pojo的类名（首字母小写）

```
@RequestMapping("/editItemSubmit")
public String editItemSubmit(Integer id,ItemsCustom itemsCustom)throws Exception{
```

  - springmvc自动将itemsCustom放回request，相当于调用下边的代码：
    model.addAttribute("itemsCustom", itemsCustom);

  - 全部代码参见：ch04_springmvc01工程

数据回显

Neuedu

- pojo数据类型
  - 如果key不是pojo的类名(首字母小写)，可以使用@ModelAttribute完成数据回显。
- @ModelAttribute作用如下：
  - 如果key不是pojo的类名(首字母小写)，可以使用@ModelAttribute完成数据回显。

```
// 商品修改提交
@RequestMapping("/editItemSubmit")
public String editItemSubmit(Model model,@ModelAttribute("item")
        ItemsCustom itemsCustom)
```

```
<td>商品名称</td>
    <td><input type="text" name="name" value="${item.name }"/></td>
</tr>
<tr>
    <td>商品价格</td>
    <td><input type="text" name="price" value="${item.price }"/></td>
</tr>
```

  - 全部代码参见：ch04_springmvc02工程

数据回显

Neuedu

- pojo数据类型
  - 使用最简单方法使用model.addAttribute( ), 可以不用@ModelAttribute

```
//可以直接使用model将提交pojo回显到页面
model.addAttribute("items",itemsCustom);
```

  - 全部代码参见：ch04_springmvc02工程

课堂练习（40分钟）

- 1、简述数据校验步骤
- 2、简述不同类型数据的回显方法
- 3、理解并独立完成教学中的案例

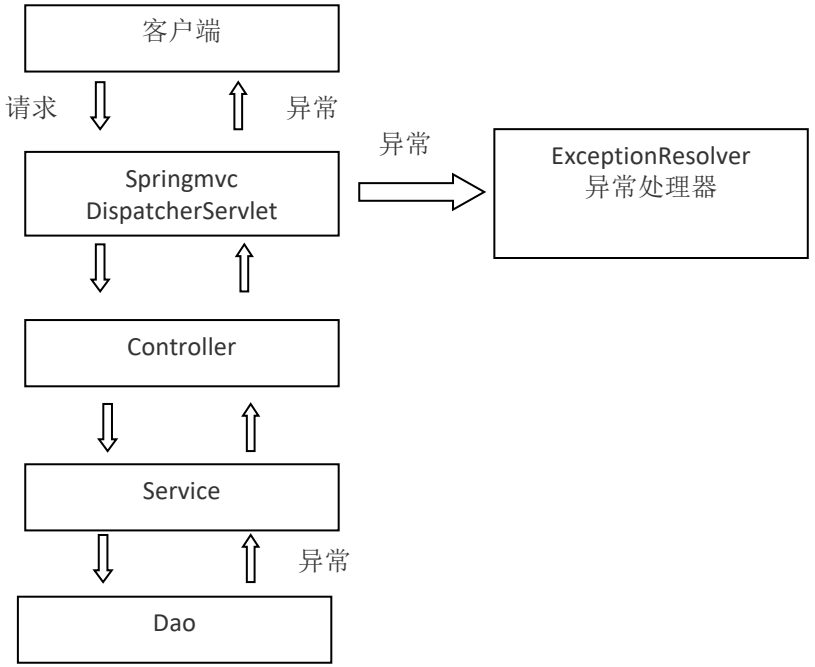
CONTENTS 目录

异常处理思路

- 系统中异常包括两类
  - 预期异常（非运行时异常）和运行时异常RuntimeException
  - 前者通过捕获异常从而获取异常信息，后者主要通过规范代码开发、测试等手段减少运行时异常的发生。
- springmvc在处理请求过程中出现异常信息交由异常处理器进行处理，自定义异常处理器可以实现一个系统的异常处理逻辑。
  - 系统的dao、service、controller出现异常，都通过throws Exception向上抛出，最后由springmvc前端控制器交由异常处理器进行异常处理，如下图：

异常处理思路

- 异常处理思路
  - springmvc提供全局异常处理器（一个系统只有一个异常处理器）进行统一异常处理。



自定义异常

Neuedu

- 示例：
  - 使用springmvc完成商品信息修改
    - 如果id不存在，报异常

- 全部代码参见：ch04_springmvc03工程

自定义异常

Neuedu

- Springmvc 自定义异常处理步骤
  - 定义自定义异常类
  - 定义异常处理器
  - 配置异常处理器
  - 编写异常信息文件
  - 异常类应用

- 全部代码参见：ch04_springmvc03工程

自定义异常

Neuedu

- 自定义异常类
  - 为了区别不同的异常通常根据异常类型自定义异常类
  - 创建一个自定义系统异常，如果controller、service、dao抛出此类异常，说明是系统预期处理的异常信息
  - 自定义异常类，要继承Exception

```
public class CustomException extends Exception {
    public String message;

    public CustomException(String message) {
        super(message);
        this.message = message;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
```

自定义异常

Neuedu

- 异常处理器
  - 实现HandlerExceptionResolver接口
  - 解析出异常类型
  - 如果该 异常类型是系统 自定义的异常，直接取出异常信息，在错误页面展示
  - 如果该 异常类型不是系统 自定义的异常，构造一个自定义的异常类型（信息为“未知错误”）

```
public class CustomExceptionResolver implements HandlerExceptionResolver {

    @Override
    public ModelAndView resolveException(HttpServletRequest requtst,
            HttpServletResponse response, Object handler, Exception ex) {
        // TODO Auto-generated method stub
        CustomException customException = null;
        if(ex instanceof CustomException) {
            customException = (CustomException)ex;
        } else {
            customException = new CustomException("未知错误");
        }
```

自定义异常

- 配置异常处理器
  - 在springmvc.xml中配置

```
<!-- 全局异常处理器，因为实现了接口HandlerExceptionResolver -->
<bean class="com.neuedu.exception.CustomExceptionResolver">
</bean>
```

自定义异常

Neuedu

- 异常页面
  - 显示异常错误信息

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>错误提示</title>
</head>
<body>
 ${message }
</body>
```

自定义异常

- 异常类应用
  - 在controller、service、dao中任意一处需要手动抛出异常。
  - 如果与业务功能相关的异常，建议在service中抛出异常。
  - 与业务功能没有关系的异常，建议在controller中抛出。

课堂练习（20分钟）

- 1、简述异常处理步骤
- 2、理解并独立完成教学中的案例

CONTENTS 目录

拦截器

- Spring Web 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。

定义拦截器

Neuedu

- 定义拦截器
  - 实现HandlerInterceptor接口
  - 接口中提供三个方法

```
//进入 Handler方法之前执行
//比如身份认证，如果认证不通过表示当前用户没有登陆，需要此方法拦截不再向下执行
public boolean preHandle(HttpServletRequest request,

//进入Handler方法之后，返回modelAndView之前执行
//应用场景从modelAndView出发：将公用的模型数据(比如菜单导航)在这里传到视图，也可以在这里统一
public void postHandle(HttpServletRequest request,

//执行Handler完成执行此方法
//应用场景：统一异常处理，统一日志处理
public void afterCompletion(HttpServletRequest request,
```

  - 全部代码参见：ch04_springmvc04工程

配置拦截器

Neuedu

- 配置拦截器
  - springmvc拦截器针对HandlerMapping进行拦截设置
  - 如果在某个HandlerMapping中配置拦截，经过该 HandlerMapping映射成功的handler最终使用该拦截器。
  - 在springmvc.xml中配置

```
<bean
 class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
 <property name="interceptors">
      <list>
           <ref bean="handlerInterceptor1"/>
           <ref bean="handlerInterceptor2"/>
      </list>
 </property>
</bean>
 <bean id="handlerInterceptor1" class="springmvc.intercapter.HandlerInterceptor1"/>
 <bean id="handlerInterceptor2" class="springmvc.intercapter.HandlerInterceptor2"/>
```

  - 全部代码参见：ch04_springmvc04工程

配置拦截器

Neuedu

- 配置全局拦截器
  - springmvc配置全局的拦截器，springmvc框架将配置的全局的拦截器注入到每个HandlerMapping中。
  - 在springmvc.xml中配置

```
<!-- 拦截器 -->
<!--多个拦截器,顺序执行 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- /**表示所有url包括子url路径 -->
        <mvc:mapping path="/**"/>
        <bean class="com.ttc.ssm.interceptor.HandlerInterceptor1"></bean>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="com.ttc.ssm.interceptor.HandlerInterceptor2"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

  - 全部代码参见：ch04_springmvc04工程

课堂练习（3分钟）

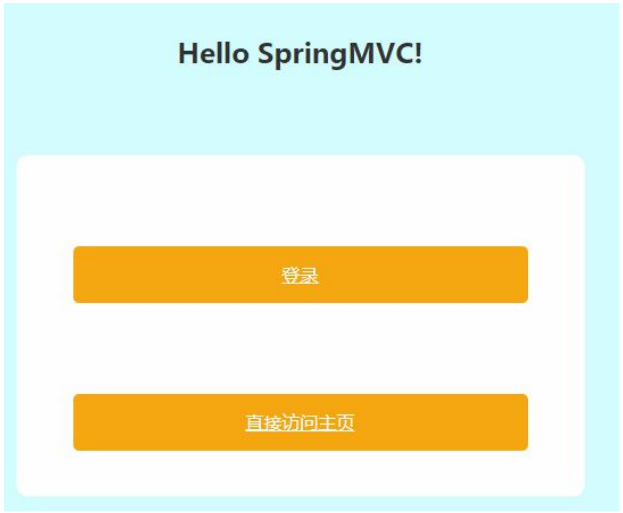
- 1、拦截器的作用
- 2、简述拦截器的定义和配置

CONTENTS 目录

本章实战项目任务实现

- 通过本章内容的学习，完成《跨境电商系统》用户首页访问拦截
  - 登录成功后可以访问用户首页（main.jsp）
  - 用户如果没有登录就访问首页，显示登录页面
  - 用户登录时，对用户名、密码添加校验规则进行校验，校验不通过给出提示

- 运行效果如图：



本章重点总结

- 了解异常处理思路；
- 理解数据校验概述；
- 掌握数据校验步骤、分组校验；
- 掌握数据回显；
- 掌握自定义异常；
- 掌握拦截器的定义和配置；

课后作业【必做任务】

- 1、完成《跨境电商系统》用户登录页面的数据验证
  - 如果用户名为空，提示“请输入1到10个字符的用户名称”
  - 使用非分组校验

- 运行效果如图：

课后作业【选做任务】

- 1、完成《跨境电商系统》用户登录页面的数据验证
  - 如果用户名为空，抛出异常，回到登录页面，提示“请输入1到10个字符的用户名称”，同时在控制台输出异常信息
  - 页面信息使用分组校验

- 运行效果如图：

课后作业【线上任务】

- 线上任务
  - 安排学员线上学习任务（安排学员到睿道实训平台进行复习和预习的任务，主要是进行微课的学习）