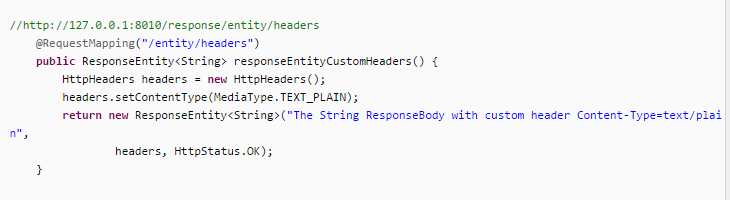
SPRING MVC3的@ResponseBody使Controller直接返回数据，而不是直接指向具体的视图；

MessageConverter和produces（如produces="text/plain;charset=UTF-8"）可以返回各种格式的数据（XML，json，RSS，TEXT，字节流等),本章只介绍最简单的使用；

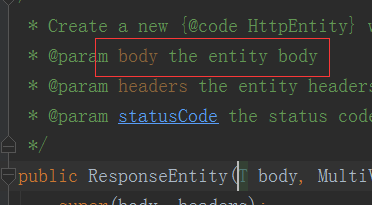
见代码：

@ResponseBody可以直接返回结果数据，也可以定义返回的HttpHeaders和HttpStatus。

而ResponseEntity***可以定义返回的HttpHeaders和HttpStatus，见文章最后的两张图，不是注解***







# i18n问题

<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  
 <property name="basename" value="i18n/message"/>  
 </bean>

<mvc:interceptors>   
 *<!-- 国际化操作拦截器 如果采用基于（请求/Session/Cookie）则必需配置 -->* <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" /> (用来判断参数locale是否存在，若存在，调用定义的LocaleResolver.setLocale(HttpReq,Res,Locale)方法)  
</mvc:interceptors>   
*<!-- 基于Session -->*<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />(实现了 setLocale(HttpReq,Res,Locale))

是通过设置保存在session中的attribute-SessionLocaleResolver.*LOCALE\_SESSION\_ATTRIBUTE\_NAME*

的value-Locale来实现的

然后定义一个controller，执行

request.getSession().setAttribute(SessionLocaleResolver.*LOCALE\_SESSION\_ATTRIBUTE\_NAME*,locale);

这样就在session里注入了新的Locale

因为在MessageSource中，获得message的方法

String getMessage(String code, Object[] args, String defaultMessage, Locale locale);

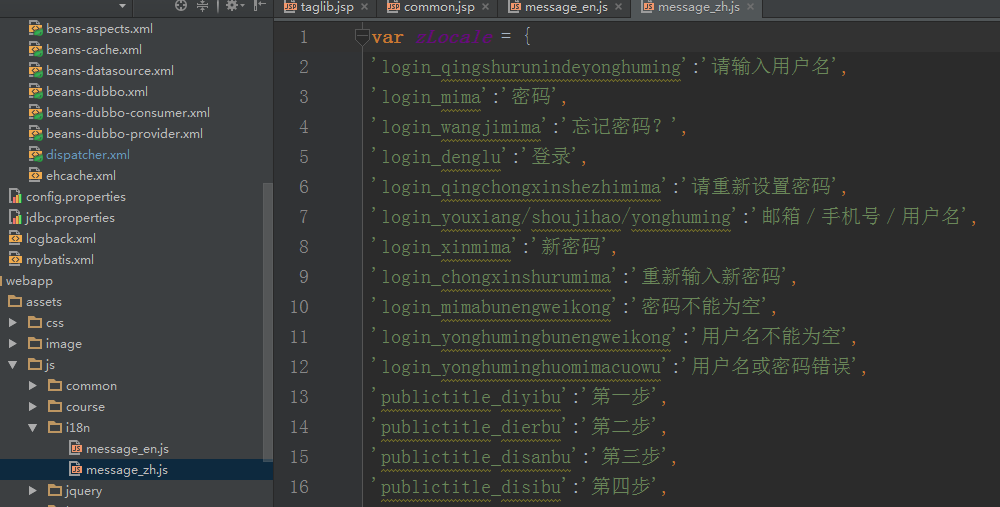
String getMessage(String code, Object[] args, Locale locale) throws NoSuchMessageException;

所以现在只需要一个code就ok。

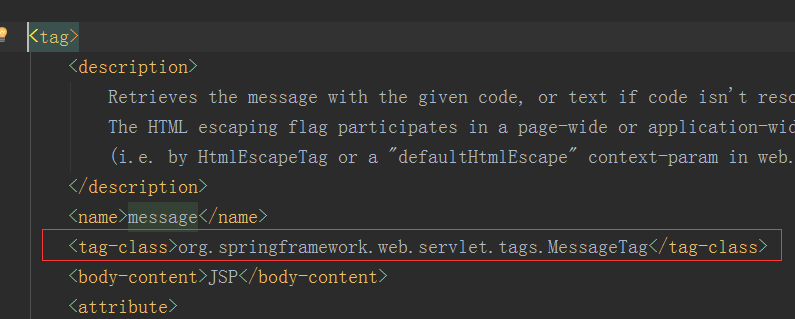
使用方法：<**spring:message code**="firststep\_pianhua"/>

相应的资源也可以区分

<script type="text/javascript" src="**${**http\_treenity**}**/assets/js/i18n/message\_**${**z\_locale\_code**}**.js"></script>

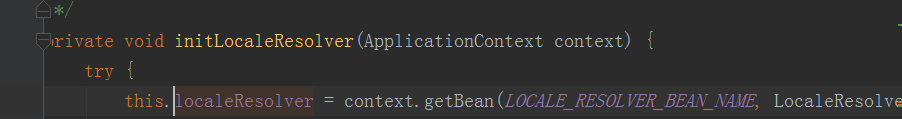


标签定义

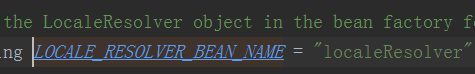


总的流程：

DispatchServlet在启动时

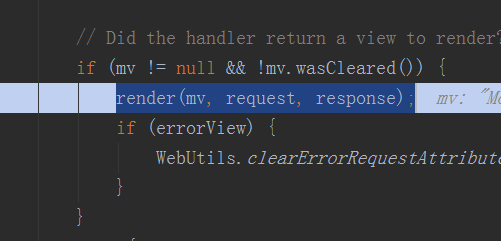


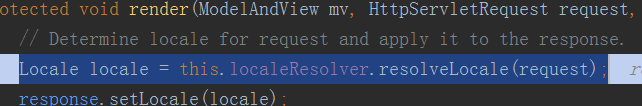
会寻找



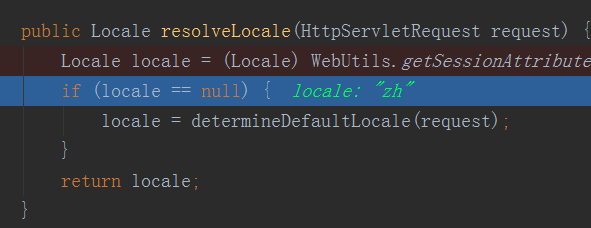
并引用该对象

在请求的render时期内，



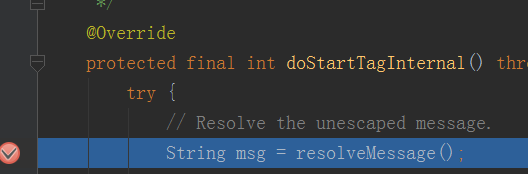


最终调用我们定义了的<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />的



方法返回当前Locale

然后执行标签体body class：



最终调用

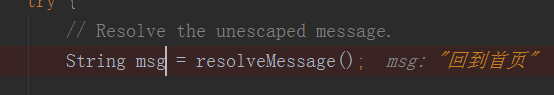
// We have no fallback text to consider.  
return messageSource.getMessage(  
 resolvedCode, argumentsArray, getRequestContext().getLocale());

其中的messageSource我们也定义了

<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  
 <property name="basename" value="i18n/message"/>  
 </bean>

提供了code文件基址

执行完之后



msg就有值了

ModelAndView重定向：

|  |  |
| --- | --- |
| 1  2  3  4 | public ModelAndView getPage404MV() {      ModelAndView mv = new ModelAndView("redirect:/404.htm");      return mv;  } |

要使用forward重定向就只需把redirect换成forward即可，特别的ModelAndView默认使用forward重定向方式。

forward是服务器内部重定向，客户端并不知道服务器把你当前请求重定向到哪里去了，地址栏的url与你之前访问的url保持不变。redirect则是客户端重定向，是服务器将你当前请求返回，然后给个状态标示给你，告诉你应该去重新请求另外一个url，具体表现就是地址栏的url变成了新的url。

Spring MVC处理异常有3种方式：   
（1）使用Spring MVC提供的简单异常处理器SimpleMappingExceptionResolver；   
（2）实现Spring的异常处理接口HandlerExceptionResolver 自定义自己的异常处理器；   
（3）使用@ExceptionHandler注解实现异常处理；

全局异常处理

*<!-- 全局异常处理 -->*<bean  
 class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  
 <property name="defaultStatusCode" value="500" />  
 <property name="defaultErrorView" value="error/500" />  
</bean>

1. **<bean** class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"**>**
2. <!-- 定义默认的异常处理页面，当该异常类型的注册时使用 -->
3. **<property** name="defaultErrorView" value="error"**></property>**
4. <!-- 定义异常处理页面用来获取异常信息的变量名，默认名为exception -->
5. **<property** name="exceptionAttribute" value="ex"**></property>**
6. <!-- 定义需要特殊处理的异常，用类名或完全路径名作为key，异常也页名作为值 -->
7. **<property** name="exceptionMappings"**>**
8. **<props>**
9. **<prop** key="cn.basttg.core.exception.BusinessException"**>**error-business**</prop>**
10. **<prop** key="cn.basttg.core.exception.ParameterException"**>**error-parameter**</prop>**
12. <!-- 这里还可以继续扩展对不同异常类型的处理 -->
13. **</props>**
14. **</property>**
15. **</bean>**

**3.3.2 实现HandlerExceptionResolver 接口自定义异常处理器**   
1、增加HandlerExceptionResolver 接口的实现类MyExceptionHandler，代码如下：

**Java代码  收藏代码**

1. **public** **class** MyExceptionHandler **implements** HandlerExceptionResolver {
3. **public** ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,
4. Exception ex) {
5. Map<String, Object> model = **new** HashMap<String, Object>();
6. model.put("ex", ex);
8. // 根据不同错误转向不同页面
9. **if**(ex **instanceof** BusinessException) {
10. **return** **new** ModelAndView("error-business", model);
11. }**else** **if**(ex **instanceof** ParameterException) {
12. **return** **new** ModelAndView("error-parameter", model);
13. } **else** {
14. **return** **new** ModelAndView("error", model);
15. }
16. }
17. }

2、在Spring的配置文件applicationContext.xml中增加以下内容：

**Xml代码  收藏代码**

1. **<bean** id="exceptionHandler" class="cn.basttg.core.exception.MyExceptionHandler"**/>**

**3.3.3 使用@ExceptionHandler注解实现异常处理**   
1、增加BaseController类，并在类中使用@ExceptionHandler注解声明异常处理，代码如下：

**Java代码  收藏代码**

1. **public** **class** BaseController {
2. /\*\* 基于@ExceptionHandler异常处理 \*/
3. @ExceptionHandler
4. **public** String exp(HttpServletRequest request, Exception ex) {
6. request.setAttribute("ex", ex);
8. // 根据不同错误转向不同页面
9. **if**(ex **instanceof** BusinessException) {
10. **return** "error-business";
11. }**else** **if**(ex **instanceof** ParameterException) {
12. **return** "error-parameter";
13. } **else** {
14. **return** "error";
15. }
16. }
17. }

2、修改代码，使所有需要异常处理的Controller都继承该类，如下所示，修改后的TestController类继承于BaseController：

**Java代码  收藏代码**

1. **public** **class** TestController **extends** BaseController

**3.4 未捕获异常的处理**   
对于Unchecked Exception而言，由于代码不强制捕获，往往被忽略，如果运行期产生了Unchecked Exception，而代码中又没有进行相应的捕获和处理，则我们可能不得不面对尴尬的404、500……等服务器内部错误提示页面。   
我们需要一个全面而有效的异常处理机制。目前大多数服务器也都支持在Web.xml中通过<error-page>(Websphere/Weblogic)或者<error-code>(Tomcat)节点配置特定异常情况的显示页面。修改web.xml文件，增加以下内容：

1. <!-- 出错页面定义 -->
2. **<error-page>**
3. **<exception-type>**java.lang.Throwable**</exception-type>**
4. **<location>**/500.jsp**</location>**
5. **</error-page>**
6. **<error-page>**
7. **<error-code>**500**</error-code>**
8. **<location>**/500.jsp**</location>**
9. **</error-page>**
10. **<error-page>**
11. **<error-code>**404**</error-code>**
12. **<location>**/404.jsp**</location>**
13. **</error-page>**

路径映射到地址

<mvc:resources mapping="/favicon.ico" location="http://assets.zhihuishu.com/icon/favicon.ico" />

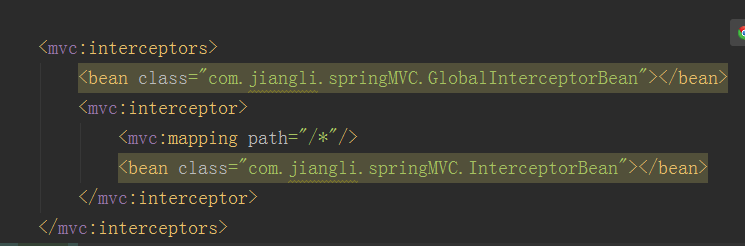
路径映射到view

<mvc:view-controller path="/" view-name="course/home"/>

一般的Controller是将路径映射到Handler

@ResponseBody  
@RequestMapping(value="")  
private ModelAndView index(@RequestParam(value="cccId",required= false)long courseId) {  
 ModelAndView mav = new ModelAndView();  
 mav.addObject("cccId", courseId);  
 mav.setViewName("/course/fifthStep");  
 return mav;  
}

拦截器



执行顺序：

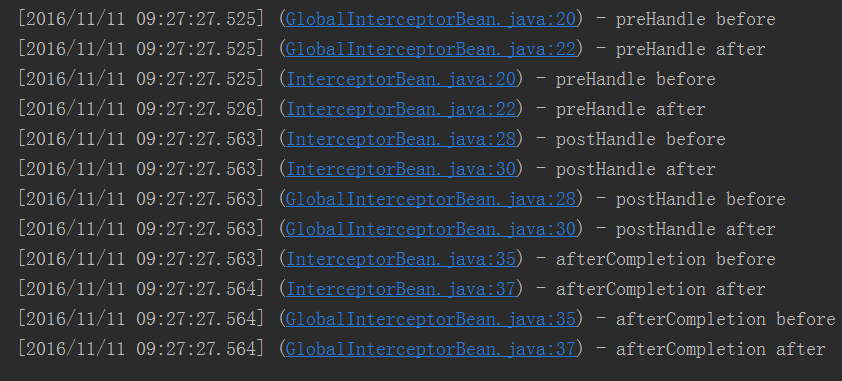
preHandle

postHandle

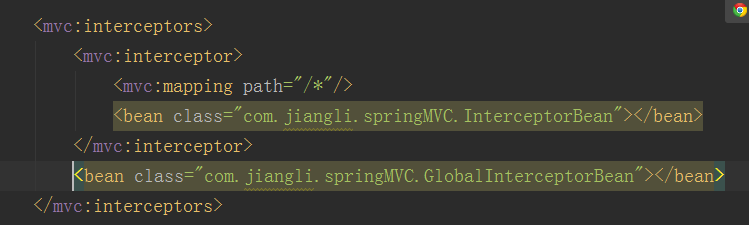
afterCompletion

并没有afterConcurrentHandlingStarted

详细结果：



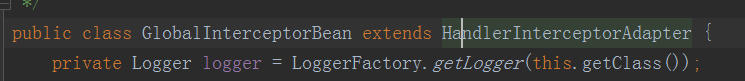
将定义修改成如下，以测试全局和局部的执行顺序是否受定义影响



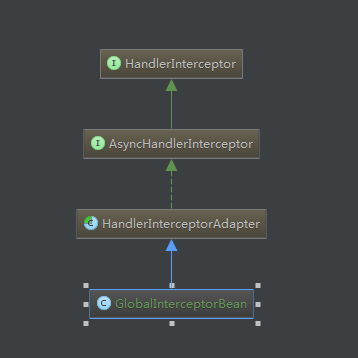
结果

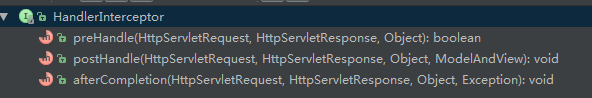


说明全局和局部拦截器的执行顺序的确受定义的先后顺序影响

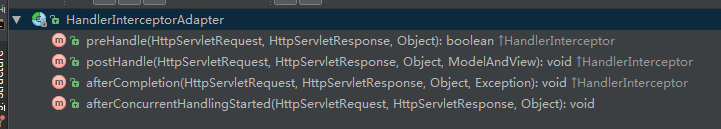


其中父类不过是实现了接口的空方法，子类按需重写即可









org.springframework.web.util。abstract

WebUtils

