

07 SpringMVC-JstlView &国际化 & 自定义视图

SpringMVC

一：资源视图的描述

1、InternalResourceViewResolver

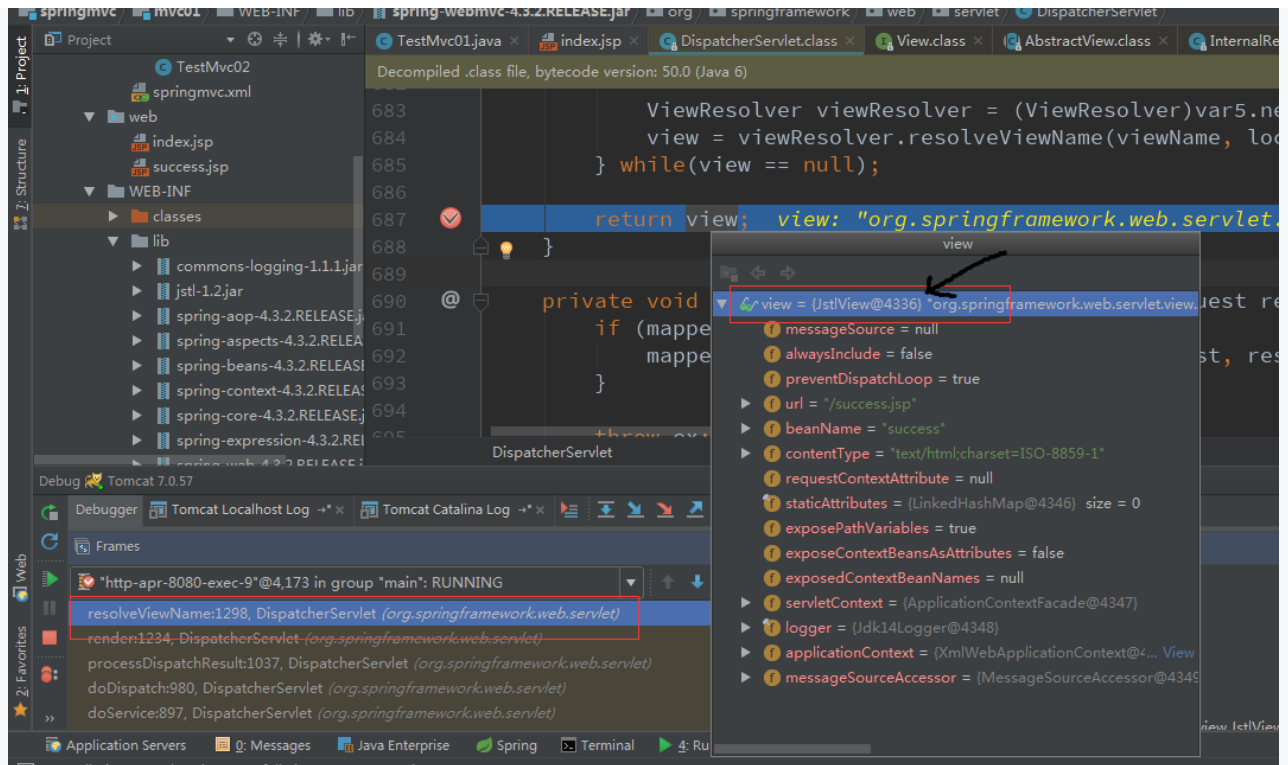
- 1、是将JSP或者其它的资源封装成一个视图，InternalResourceView是默认使用的视图解析类

2、JstlView

- 1、如jsp中使用了，JSTL国际化标签的功能，则需要使用JstlView视图的功能

二：JstlView视图(国际化)

- 1、在使用该视图(国际化)
- 2、需要添加jar包standard.jar和jstl-1.2.jar
- 3、进行DUG测试，会发现在DispatcherServlet中的resolveViewName方法，获取的View是JstlView了



4、默认情况下，SpringMVC 根据 Accept-Language 参数判断客户端的本地化类型。

5、当接受到请求时，SpringMVC 会在上下文中查找一个本地化解析器（LocalResolver），找到后使用它获取请求所对应的本地化类型信息

6、SpringMVC 还允许装配一个动态更改本地化类型的拦截器，这样通过指定一个请求参数就可以控制单个请求的本地化类型

1、进行配置国际化

1、在SRC目录创建一个文
i18n.properties，i18n_en_US.properties，i18n_zh_CN.properties文件

1. //i18n.properties, i18n_en_US.properties的文件信息
2. i18n.username=Username
3. i18n.password=Password
4. //中文
5. i18n.username=用户

```
6. i18n.password=密码
```

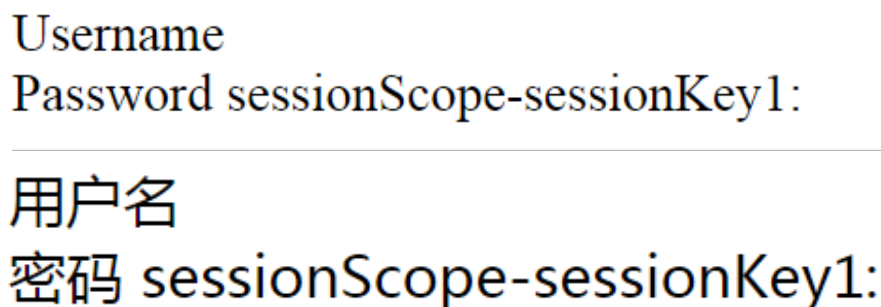
2、页面进行测试

```
1. //导入Jstl的标签
2. <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3. <html>
4.     <head>
5.         <title>${Title$}</title>
6.     </head>
7.     <body>
8.         <fmt:message key="i18n.username"></fmt:message>
9.         <BR/>
10.        <fmt:message key="i18n.password"></fmt:message>
```

3、springMVC.xml文件的配置

```
1. <!--配置国际化的文件-->
2. <bean id="messageSource"
3.     class="org.springframework.context.support.ResourceBundleMessageSource"
4.     >
5.         <property name="basename" value="i18n"></property>
6.     </bean>
```

4、运行结果(需要改了进行修改浏览器的语言)



Username
Password sessionScope-sessionKey1:

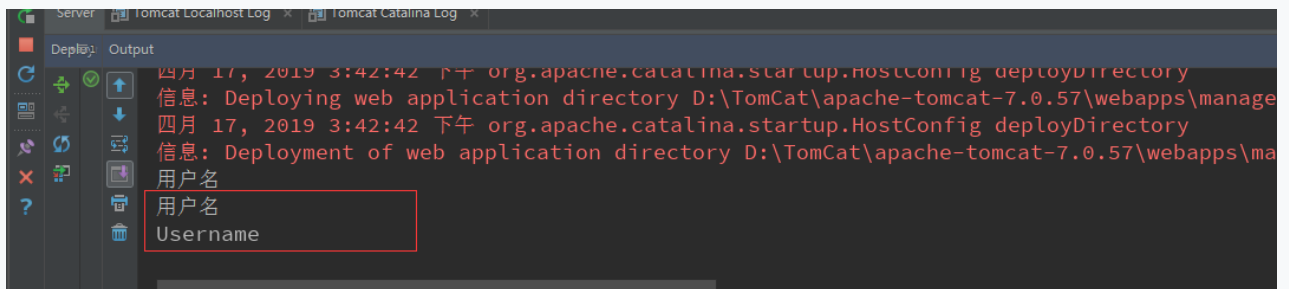
用户名
密码 sessionScope-sessionKey1:

2、ResourceBundleMessageSource获取国际化的信息

1、创建一个方法,并注入ResourceBundleMessageSource

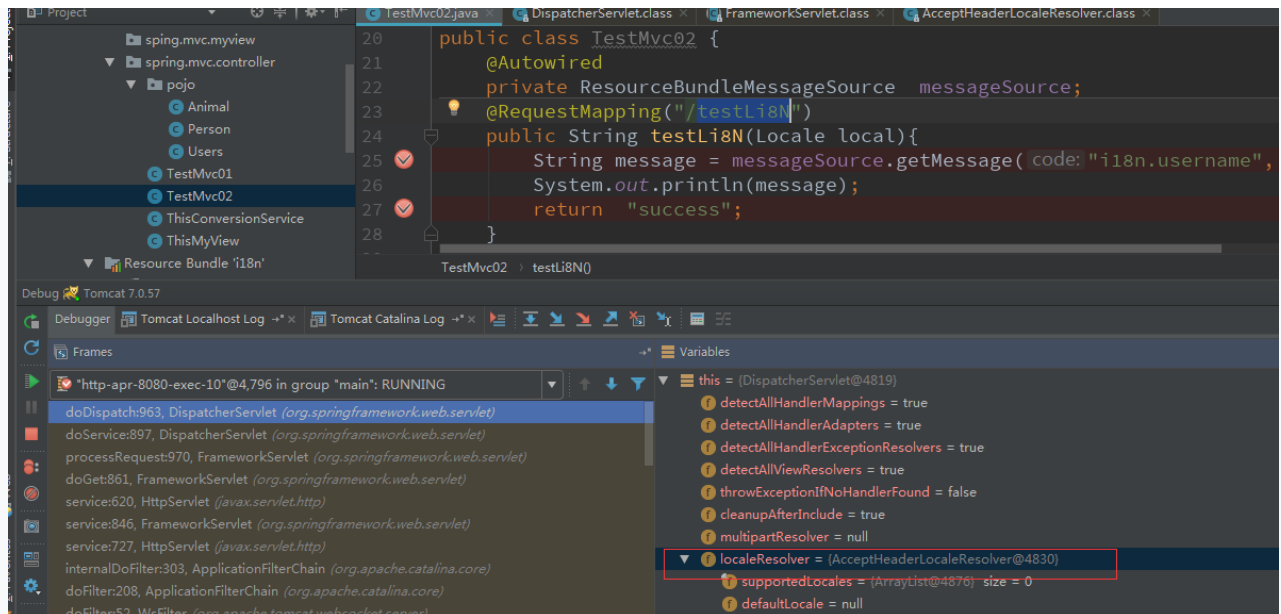
```
1. @Controller
2. public class TestMvc02 {
3.     @Autowired
4.     private ResourceBundleMessageSource messageSource;
5.     @RequestMapping("/testLi8N")
6.     public String testLi8N(Locale local) {
7.         String message = messageSource.getMessage("i18n.username", null
8.         , local);
9.         System.out.println(message);
10.        return "success";
11.    }
```

2、进行测试访问，该地址并修改浏览器访问语言



1、源码ResourceBundleMessageSource

1、源码查询在ResourceBundleMessageSource对象调用getMessage(...)处进行DUG断点测试



2、查询AcceptHeaderLocaleResolver类

```

1.  public class AcceptHeaderLocaleResolver implements LocaleResolver {
2.      private final List<Locale> supportedLocales = new ArrayList(4);
3.      //代码省略...
4.
5.      public Locale resolveLocale(HttpServletRequest request) {
6.          Locale defaultLocale = this.getDefaultLocale();
7.          if (defaultLocale != null && request.getHeader("Accept-Language") == null) {
8.              return defaultLocale;
9.          } else {
10.             /**
11.              直接是从request域中获取Locale进行解析
12.              使用超链接进行操作国际化，直接把request域中的gLocale
13.              */
14.             Locale locale = request.getLocale();
15.             if (!this.isSupportedLocale(locale)) {
16.                 locale = this.findSupportedLocale(request, locale);
17.             }
18.
19.             return locale;
20.         }
21.     }
22.     //代码省略...

```

3、使用A标签进行转换国际化

1、本地化解析器和本地化拦截器

2、AcceptHeaderLocaleResolver：

-----根据 HTTP 请求头的Accept-Language 参数确定本地化类型，如果没有显式定义本地化解析器，SpringMVC 使用该解析器

3、CookieLocaleResolver：

-----根据指定的 Cookie 值确定本地化类型

4、SessionLocaleResolver

-----根据 Session 中特定的属性确定本地化类型

5、LocaleChangeInterceptor：

-----从请求参数中获取本次请求对应的本地化类型

1、配置

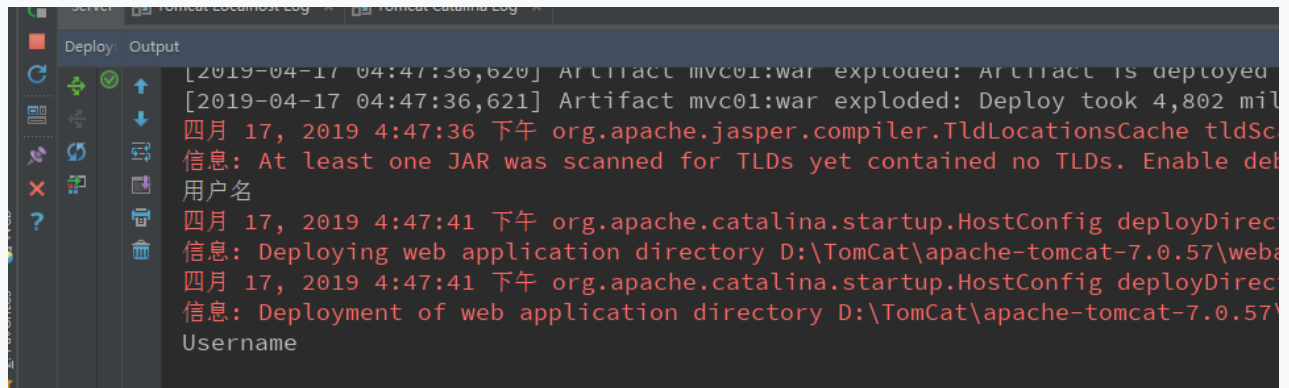
1、配置Spring.xml文件

```
1.      <!--配置SessionLocalResolver-->
2.          <bean id="localeResolver"
3.
4.      class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
5.          </bean>
6.      <!--配置localChanceInterceptor-->
7.          <mvc:interceptors>
8.              <bean
9.      class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
10.          </bean>
11.          </mvc:interceptors>
```

2、页面代码

```
1.      testLi8N:<a href="testLi8N" >testLi8N</a>
2.      <br>
3.      中文:<a href="testLi8N?locale=zh_CN" >testLi8N</a>
4.      <br>
5.      英文<a href="testLi8N?locale=en_US" >testLi8N</a>
6.      <br>
```

3、请求



2、源码解析

1、在DispatcherServlet类中找到doDispatch方法

```
1.     protected void doDispatch(HttpServletRequest request,....){
2.         //代码省略.....
3.         //获取视图
4.         mv = ha.handle(processedRequest, response, mappedHandler.getHandler()
5.         );
6.         //代码省略.....
7.         //进行渲染视图，点击查询...
8.         this.processDispatchResult(processedRequest, response, mappedHandler
9.         , mv,....
10.        //代码省略.....
11.    }
```

2、this.processDispatchResult(

```
1.     private void processDispatchResult(HttpServletRequest ....){
2.         //代码省略.....
3.         if (mv != null && !mv.wasCleared()) {
4.             //点击render可以看到，里面进行解析了locale
5.             this.render(mv, request, response);
6.             if (errorView) {
7.                 //代码省略.....
8.             }
```

3、render

```
1.     protected void render(ModelAndView mv, HttpServletRequest request, Http
2.         ServletResponse response) throws Exception {
3.         Locale locale = this.localeResolver.resolveLocale(request);
4.         //代码省略.....
```

1-1、LocaleChangeInterceptor类

1、查询LocaleChangeInterceptor 源码

```
1.     public class LocaleChangeInterceptor extends HandlerInterceptorAdapter
2.     {
3.         //代码省略.....
4.         //该方法是doDispatch方法调用 mv =ha.handle....方法之前调用
5.         public boolean preHandle(HttpServletRequest request, .....){
6.         /**
7.         获取locale与解析locale
8.         */
9.         String newLocale = request.getParameter(this.getParamName());
10.         if (newLocale != null && this.checkHttpMethod(request.getMethod
11.             ())) {
12.             LocaleResolver localeResolver = RequestContextUtils.getLoca
13.                 leResolver(request);
14.             if (localeResolver == null) {
15.                 throw new IllegalStateException("No LocaleResolver foun
16.                 d: not in a DispatcherServlet request?");
17.             }
18.             try {
19.                 /**
20.                 点击 localeResolver 源码查询
21.                 localeResolver中就包含了locale
22.                 结果抽象方法，进行查询实现类
23.                 */
24.                 localeResolver.setLocale(request, response, this.parseL
25.                     ocaleValue(newLocale));
26.             } catch (IllegalArgumentException var7) {
27.             }
28.         }
29.         //代码省略.....
30.         =====
31.         =====
```



```

27.  =====
28.     protected void doDispatch(HttpServletRequest request,...){
29.         //代码省略.....
30.         //调用的preHandle
31.         if (!mappedHandler.applyPreHandle(processedRequest, response)) {
32.             return;
33.         }
34.         mv = ha.handle(processedRequest, response,mappedHandler.getHandler());
35.     }

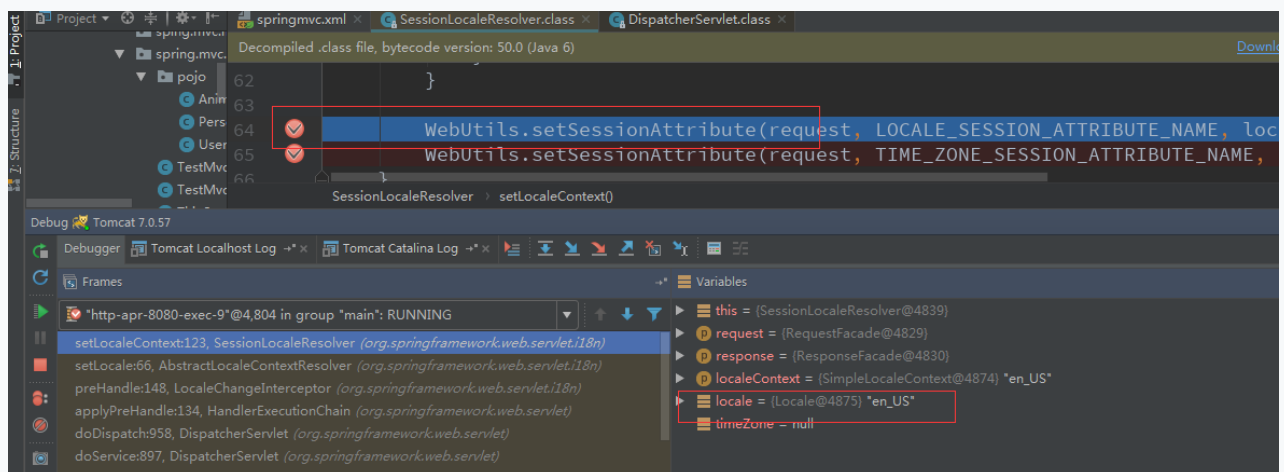
```

2、setLocale是实现类，在进行点击setLocaleContext的实现类SessionLocaleResolver

```

1.     public void setLocaleContext(HttpServletRequest request,
2.         HttpServletResponse response, LocaleContext localeContext) {
3.         //代码省略.....
4.         //设置到Session中，可进行DUG进行测试
5.         WebUtils.setSessionAttribute(request,
6.             LOCALE_SESSION_ATTRIBUTE_NAME, locale);
7.         WebUtils.setSessionAttribute(request,
8.             TIME_ZONE_SESSION_ATTRIBUTE_NAME, timeZone);
9.     }

```



三：mvc:view-controller 标签

1、在springMVC.xml中使用mvc:view-controller 这无需通过Controller进行跳转如

```
1.      <mvc:view-controller path="/success" view-name="success">
      </mvc:view-controller>
```

- 2、访问：<http://localhost:8080/success>；可以直接跳转到对应的页面，
- 3、但是其他的controller是不能进行访问的，那怎么处理？则加mvc:annotation-driven标签就可，访问了

```
1.      <!--在开发中，通常配置mvc:annotation-driven标签-->
2.      <mvc:annotation-driven></mvc:annotation-driven>
```

四：自定义视图

- 1、在src的目录下创建一个src\com\sping\mvc\controller\ThisMyView.java

```
1.      @Component
2.      public class ThisMyView implements View {
3.          @Override
4.          public String getContentType() {
5.              return "text/html;charset=UTF-8";
6.          }
7.          @Override
8.          public void render(Map<String, ?> map, HttpServletRequest httpServlet
9.              etRequest, HttpServletResponse httpServletResponse) throws Exception {
10.              httpServletResponse.getWriter().print("Hello "+new Date());
11.          }
```

- 2、SpringMVC.xml文件配置视图解析器

```
1.      <!--配置视图BeanNameViewResolver解析器-->
2.      <bean
3.          class="org.springframework.web.servlet.view.BeanNameViewResolver">
4.          <!--order属性定义,视图的解析器的优先级, order值越小, 优先越高;
5.              默认的: InternalResourceViewResolver的Order值在父类中。
6.          -->
7.          <property name="order" value="100"></property>
```

```
7.         </bean>
```

3、Controller层

```
1.         @RequestMapping("/testThisMyView")
2.         public String testThisMyView() {
3.             System.out.println("testThisMyView:");
4.             //调用用Serivce层
5.             //返回的视图名称是自定义的视图类名的小写
6.             return "thisMyView";
7.         }
8.     =====请求的页面=====
9.     <HR>
10.    <a href="testThisMyView">testThisMyView</a>
11.    <HR>
```