

# Web开发

## 1 要解决的问题

- 导入静态资源
- 首页

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Feb 22 17:00:07 CST 2020

There was an unexpected error (type=Not Found, status=404).

No message available

- jsp, 我们使用模板引擎 (Thymeleaf)
- 装配扩展SpringMVC
- 拦截器
- 国际化 (I18N)

我们一个个解决。

## 2 静态资源

我们项目中有许多的静态资源, 比如, css, js等文件, 但是SpringBoot对于静态资源放置的位置, 是有规定的。

静态资源映射规则: SpringBoot中, SpringMVC的web配置都在 WebMvcAutoConfiguration 这个配置里面, 我们可以去看看 WebMvcAutoConfigurationAdapter 中有很多配置方法;

点进WebMvcAutoConfiguration 的源码:

```
1  @Override
2  public void addResourceHandlers(ResourceHandlerRegistry registry) {
3      if (!this.resourceProperties.isAddMappings()) {
4          logger.debug("Default resource handling disabled");
5          return;
6      }
7      Duration cachePeriod = this.resourceProperties.getCache().getPeriod();
8      CacheControl cacheControl =
9      this.resourceProperties.getCache().getCachecontrol().toHttpCacheControl();
10     if (!registry.hasMappingForPattern("/webjars/**")) {
11         customizeResourceHandlerRegistration(registry.addResourceHandler("/webjars
12         /**")
13         .addResourceLocations("classpath:/META-INF/resources/webjars/"))
14         .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
15     }
16     String staticPathPattern = this.mvcProperties.getStaticPathPattern();
```

```

15     if (!registry.hasMappingForPattern(staticPathPattern)) {
16
17         customizeResourceHandlerRegistration(registry.addHandler(staticPathPattern)
18
19         .addResourceLocations(getResourceLocations(this.resourceProperties.getStaticLocations()))
20
21         .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
22     }
23 }

```

## 2.1 第一种方法: webjars

源码的第9行到第13行:

```

1  if (!registry.hasMappingForPattern("/webjars/**")) {
2
3      customizeResourceHandlerRegistration(registry.addHandler("/webjars/**")
4
5      .addResourceLocations("classpath:/META-INF/resources/webjars/")
6
7      .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
8  }

```

所有的 /webjars/\*\*, 都需要去 classpath:/META-INF/resources/webjars/ 找对应的资源。

什么是webjars: webjars本质就是以jar包的方式引入我们的静态资源, 我们以前要导入一个静态资源文件, 直接导入即可。使用SpringBoot需要使用webjars

webjars官网: <https://www.webjars.org/>

比如使用jQuery:

| Name    | Versions | Build Tool: SBT / Play 2 Maven Ivy Grape Gradle Buildr Leiningen   |
|---------|----------|--|
| angular | 1.7.9 +  | <pre> &lt;dependency&gt;   &lt;groupId&gt;org.webjars.bower&lt;/groupId&gt;   &lt;artifactId&gt;angular&lt;/artifactId&gt;   &lt;version&gt;1.7.9&lt;/version&gt; &lt;/dependency&gt; </pre> |
| jquery  | 3.4.1 +  | <pre> &lt;dependency&gt;   &lt;groupId&gt;org.webjars&lt;/groupId&gt;   &lt;artifactId&gt;jquery&lt;/artifactId&gt;   &lt;version&gt;3.4.1&lt;/version&gt; &lt;/dependency&gt; </pre>        |
| jquery  | 3.4.1 +  | <pre> &lt;dependency&gt;   &lt;groupId&gt;org.webjars.bower&lt;/groupId&gt;   &lt;artifactId&gt;jquery&lt;/artifactId&gt; </pre>   |

导入依赖:

```

1 <dependency>
2   <groupId>org.webjars</groupId>
3   <artifactId>jquery</artifactId>
4   <version>3.4.1</version>
5 </dependency>

```



我们去访问一下：

<http://localhost:8080/webjars/jquery/3.4.1/jquery.js>

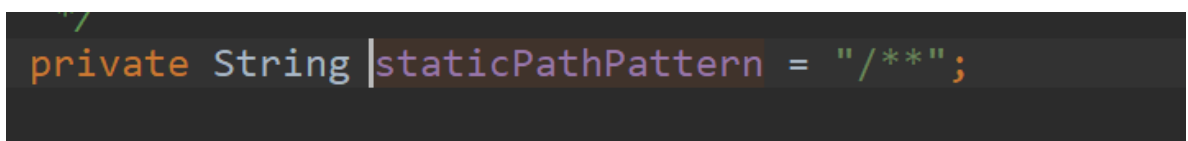


## 2.2 第二种方法【推荐】

源码的第14行到第19行：

```
1 String staticPathPattern = this.mvcProperties.getStaticPathPattern();
2 if (!registry.hasMappingForPattern(staticPathPattern)) {
3
4     customizeResourceHandlerRegistration(registry.addHandler(staticPathPattern)
5
6     .addResourceLocations(getResourceLocations(this.resourceProperties.getStaticLocations()))
7
8     .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
9 }
```

点进去看staticPathPattern的源码：



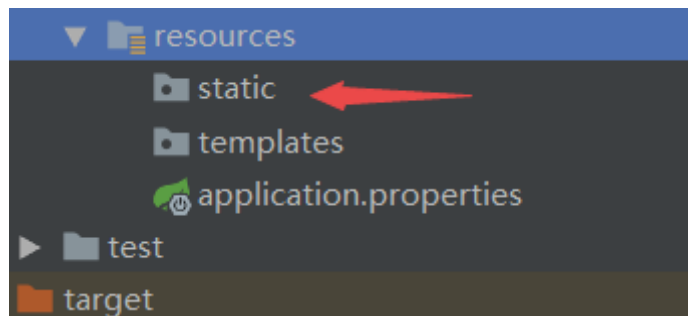
"/\*\*"代表在当前目录下都识别！

再点进ResourceProperties的源码：

```
1 private static final String[] CLASSPATH_RESOURCE_LOCATIONS = {  
2     "classpath:/META-INF/resources/",  
    "classpath:/resources/", "classpath:/static/", "classpath:/public/" };
```

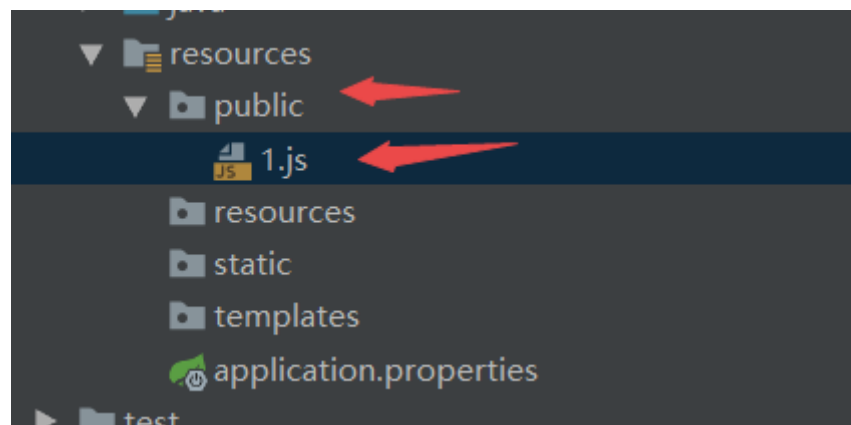
这里我们看到了四个路径以及上面的/\*\*一共五个路径：

- /\*\*
- classpath:/META-INF/resources/ 【这个就是webjars】
- classpath:/resources/ 优先级别最高
- classpath:/static/ 【我们工程中已有】 优先级别第二

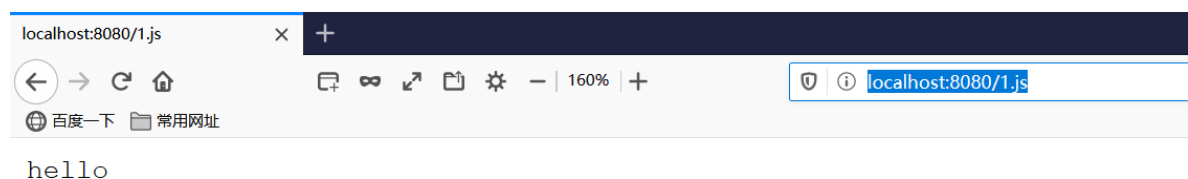


- classpath:/public/ 优先级别第三

测试一下：



<http://localhost:8080/1.js>



小结：

- 使用这种方法，在路径下用/直接去访问上述路径文件夹下的文件。【不用再写上一层的文件名，例如public】
- 当文件名一样的时候，根据**优先级别**进行访问！

## 2.3 总结

可以使用如下几种方法处理静态资源文件：

- webjars 【路径：localhost:端口号/webjars/目录结构】
- resour下的public、static、/\*\*、resources 【路径：localhost:端口号/】

### 3 首页定制问题

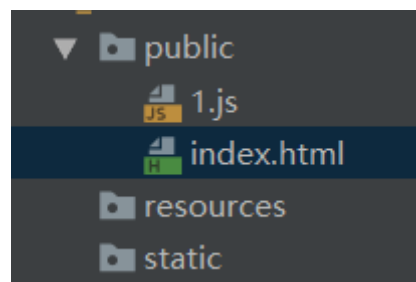
还是去查看源码，所有相关的默认配置都在类WebMvcAutoConfiguration中。

```
1  @Bean
2      public welcomePageHandlerMapping
welcomePageHandlerMapping(ApplicationContext applicationContext,
3                          FormattingConversionService mvcConversionService,
ResourceUrlProvider mvcResourceUrlProvider) {
4      welcomePageHandlerMapping welcomePageHandlerMapping = new
welcomePageHandlerMapping(
5          new TemplateAvailabilityProviders(applicationContext),
applicationContext, getWelcomePage(),
6          this.mvcProperties.getStaticPathPattern());
7
welcomePageHandlerMapping.setInterceptors(getInterceptors(mvcConversionServ
ice, mvcResourceUrlProvider));
8      return welcomePageHandlerMapping;
9  }
10
11     private Optional<Resource> getWelcomePage() {
12         String[] locations =
getResourceLocations(this.resourceProperties.getStaticLocations());
13         return
Arrays.stream(locations).map(this::getIndexHtml).filter(this::isReadable).f
indFirst();
14     }
15
16     private Resource getIndexHtml(String location) {
17         return this.resourceLoader.getResource(location +
"index.html");
18     }
```

第17行: return this.resourceLoader.getResource(location + "index.html");

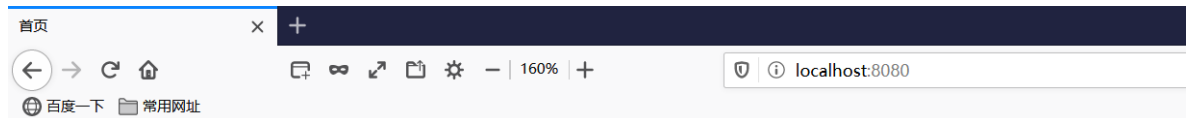
这里的location就是我们上一个话题中的静态资源的位置:

这三个文件夹中: public、resources、static



index.html:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>首页</title>
6 </head>
7 <body>
8 <h1>首页</h1>
9 </body>
10 </html>
```



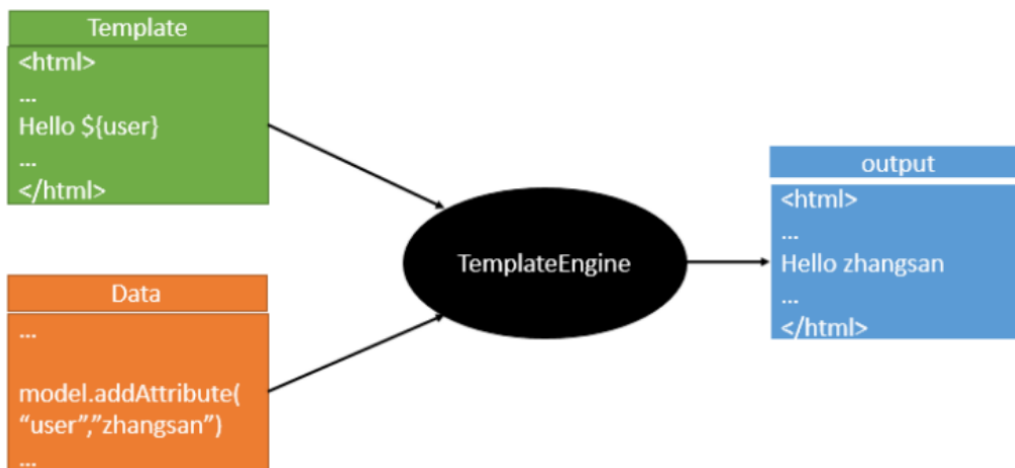
# 首页

## 4 模板引擎——Thymeleaf

### 4.1 介绍

SpringBoot是以jar的方式，不是war，我们用的还是嵌入式的Tomcat，所以呢现在默认是不支持jsp的。因此我们使用模板引擎。

思想：模板引擎的作用就是我们来写一个页面模板，比如有些值呢，是动态的，我们写一些表达式。而这些值，从哪来呢，我们来组装一些数据，我们把这些数据找到。然后把这个模板和这个数据交给我们模板引擎，模板引擎按照我们这个数据帮你把这表达式解析、填充到我们指定的位置，然后把这个数据最终生成一个我们想要的内容给我们写出去，这就是我们这个模板引擎。



### 4.2 使用

- 导入依赖：

```

1 <dependency>
2   <groupId>org.thymeleaf</groupId>
3   <artifactId>thymeleaf-spring5</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>org.thymeleaf.extras</groupId>
7   <artifactId>thymeleaf-extras-java8time</artifactId>
8 </dependency>

```

- 头部依赖

```
1 xmlns:th="http://www.thymeleaf.org"
```

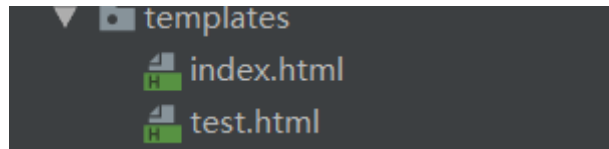
看一下配置源码： (ThymeleafProperties)

```

1 public static final String DEFAULT_PREFIX = "classpath:/templates/"; //前缀
2
3 public static final String DEFAULT_SUFFIX = ".html"; //后缀

```

- 将页面放在templates中



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <h1>Test页面</h1>
9 </body>
10 </html>

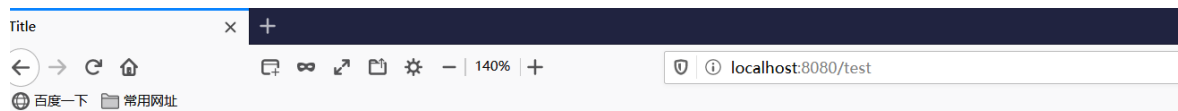
```

```

1 /**
2  * 在templates目录下的所有页面，只能通过Controller跳转
3  * 这个需要模板引擎的支持：Thymeleaf
4  */
5 @Controller
6 public class IndexController {
7   @RequestMapping("/test")
8   public String index(){
9     return "test";
10  }
11 }

```

<http://localhost:8080/test>



# Test页面

## 4.3 语法学习

官方文档: <https://www.thymeleaf.org/>

## 4.4 所有页面的静态资源都需要用Thymeleaf接管。

## 4.5 松校验和热部署

- 添加依赖

```
1      <nekohtml.version>1.9.20</nekohtml.version>
2      <xml-apis.version>1.4.01</xml-apis.version>
3      <batik-ext.version>1.9.1</batik-ext.version>
4      <!--Thymeleaf支持包
5           可以设置松校验和热部署
6      -->
7      <dependency>
8          <groupId>net.sourceforge.nekohtml</groupId>
9          <artifactId>nekohtml</artifactId>
10         <version>${nekohtml.version}</version>
11     </dependency>
12     <dependency>
13         <groupId>xml-apis</groupId>
14         <artifactId>xml-apis</artifactId>
15         <version>${xml-apis.version}</version>
16     </dependency>
17     <dependency>
18         <groupId>org.apache.xmlgraphics</groupId>
19         <artifactId>batik-ext</artifactId>
20         <version>${batik-ext.version}</version>
21     </dependency>
```

- application.yml配置

```
1  spring:
2    thymeleaf:
3      cache: false #关闭Thymeleaf的缓存【为了热部署】
4      mode: LEGACYHTML5 #松校验，以HTML5为准
```

- 热部署

修改完html页面后，可直接在项目中按组合键【ctrl + shift + f9】，之后刷新页面即可。

## 5 MVC配置原理

参考官方文档: <https://docs.spring.io/spring-boot/docs/2.1.6.RELEASE/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-auto-configuration>



## 5.1 自定义

例如想扩展ViewResolver

```
1 package com.kuang.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.View;
5 import org.springframework.web.servlet.ViewResolver;
6 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8 import java.util.Locale;
9
10 /**
11  * 用这个类扩展SpringMVC
12  * 如果想diy一些定制化的功能，只要写这个组件，然后将它交给SpringBoot，SpringBoot就会
13  * 帮我们自动装配
14  */
15 @Configuration
16 public class MyMVConfig implements WebMvcConfigurer {
17
18     public MyViewResolver myViewResolver() {
19         return new MyViewResolver();
20     }
21
22     public static class MyViewResolver implements ViewResolver {
23
24         @Override
25         public View resolveViewName(String viewName, Locale locale) throws
26             Exception {
27             return null;
28         }
29     }
30 }
```

但是我们一般不会这么做，下面的方法才是我们常用的。

## 5.2 官方推荐

举个例子：改变视图跳转

```
1 package com.kuang.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
5 import
6     org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
7 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
8
9 @Configuration
10 // @EnableWebMvc //千万不可以加这个注解，这个注解就是导入了一个类
11 // DelegatingWebMvcConfiguration==》 extends WebMvcConfigurationSupport
12 // 从容器中获取所有的webmvc config
13 public class MyMVConfig implements WebMvcConfigurer {
14     /**
```

```

13      * 视图跳转
14      *
15      * 如果我们要扩展SpringMVC，官方建议我们这样子做
16      * @param registry
17      */
18      @Override
19      public void addViewControllers(ViewControllerRegistry registry) {
20          //当遇到 /kuang 请求的时候，跳转到test页面
21          registry.addViewController("/kuang").setViewName("test");
22      }
23  }

```

有几个注意点：

1. @Configuration 标识这是一个配置类，必须标上
2. @EnableWebMvc 这个绝对不可以用（用了代表我们全面接管SpringMVC，不推荐！）

官网文档也说了：

If you want to keep Spring Boot MVC features and you want to add additional MVC configuration (interceptors, formatters, view controllers, and other features), you can add your own @Configuration class of type WebMvcConfigurer but without @EnableWebMvc. If you wish to provide custom instances of RequestMappingHandlerMapping, RequestMappingHandlerAdapter, or ExceptionHandlerExceptionResolver, you can declare a WebMvcRegistrationsAdapter instance to provide such components.

具体原因看一下源码吧：

```

@Import(DelegatingWebMvcConfiguration.class)
public @interface EnableWebMvc {
}

@Configuration(proxyBeanMethods = false)
public class DelegatingWebMvcConfiguration extends WebMvcConfigurationSupport {

    private final WebMvcConfigurerComposite configurers = new WebMvcConfigurerCompo

    @Autowired(required = false)
    public void setConfigurers(List<WebMvcConfigurer> configurers) {

    }

    @ConditionalOnWebApplication(type = Type.SERVLET)
    @ConditionalOnClass({ Servlet.class, DispatcherServlet.class, WebMvcConfigurer.class })
    @ConditionalOnMissingBean(WebMvcConfigurationSupport.class)
    @AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE + 10)
    @AutoConfigureAfter({ DispatcherServletAutoConfiguration.class, TaskExecutionAutoConfigura
        ValidationAutoConfiguration.class })
    public class WebMvcAutoConfiguration {

        public static final String DEFAULT_PREFIX = "";

        public static final String DEFAULT_SUFFIX = "";
    }
}

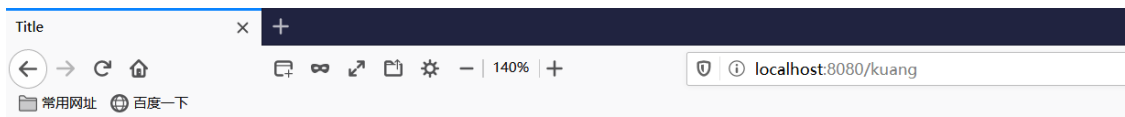
```

我们可以看到，这个注解的父类继承了WebMvcConfigurationSupport，而我们的mvc自动配置类WebMvcAutoConfiguration的注解也说了：

@ConditionalOnMissingBean(WebMvcConfigurationSupport.class) =====》告诉我们如果有使用了WebMvcConfigurationSupport这个类，那么整个WebMvcAutoConfiguration都将失效！

- 运行测试一下吧

<http://localhost:8080/kuang>



# Test页面

发现我们进行kuang的请求时，跳转到了test页面。

## 6 页面国际化 (I18N)

### 6.1 什么是国际化：

比如Dubbo官网，点中文就切换中文，点英文就切换英文。



### 6.2 使用

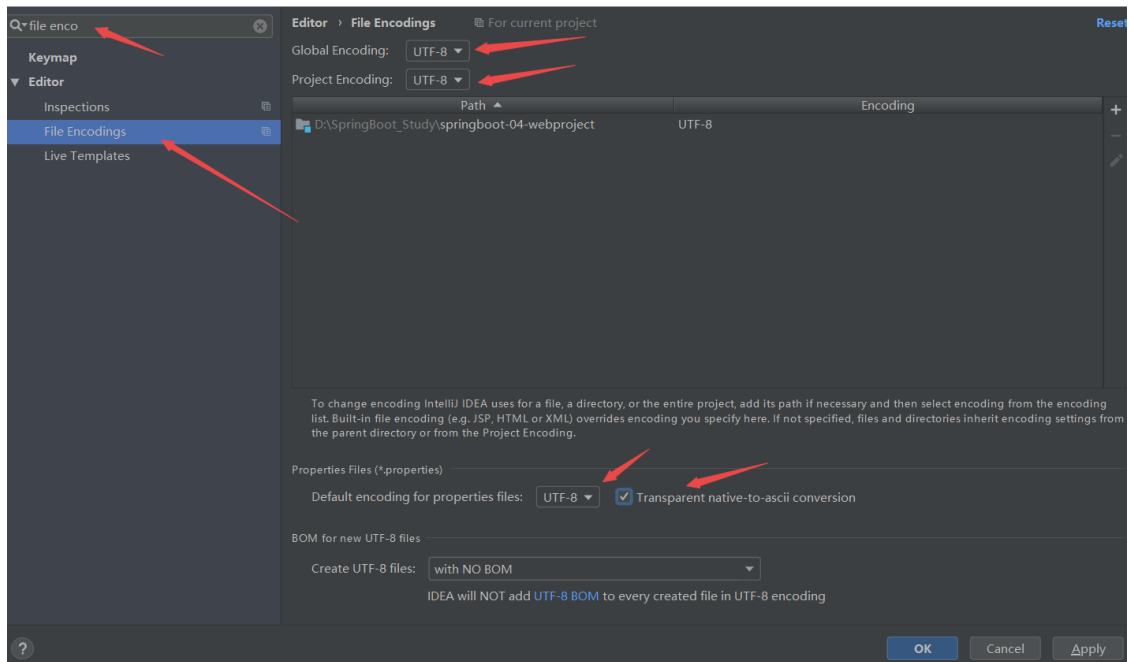
我们要配置的页面：

A screenshot of the Dubbo login page. It features a large blue square logo with a white letter 'B' at the top. Below the logo, the text 'Please sign in' is displayed. Underneath, there are two input fields: 'Username' and 'Password'. Below these fields is a checkbox labeled 'Remember me'. At the bottom, there is a large blue button with the text 'Sign in'.

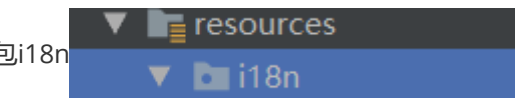
© 2017-2018

中文 English

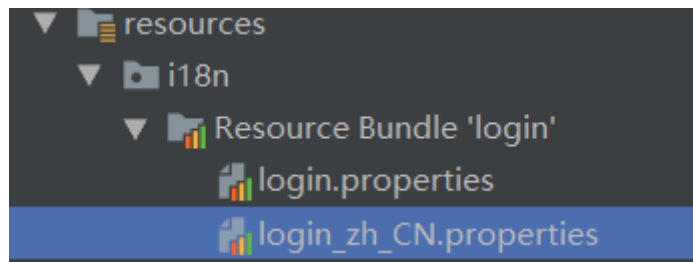
- IDEA设置保证写的东西格式



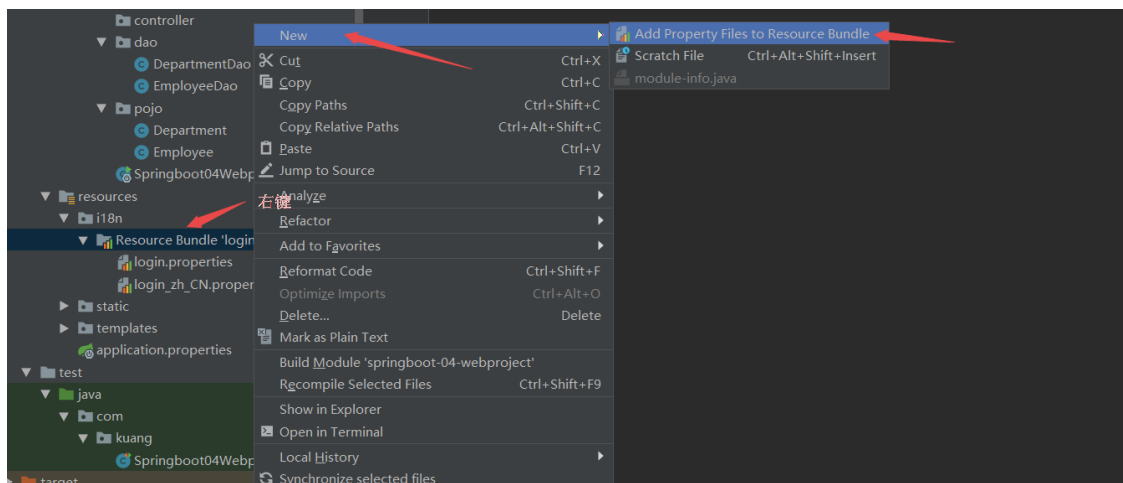
- 在resources下创建一个包i18n
- 在i18n包下创建两个properties文件：
  - login.properties
  - login\_zh\_CN.properties (代表中文)

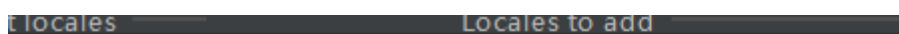
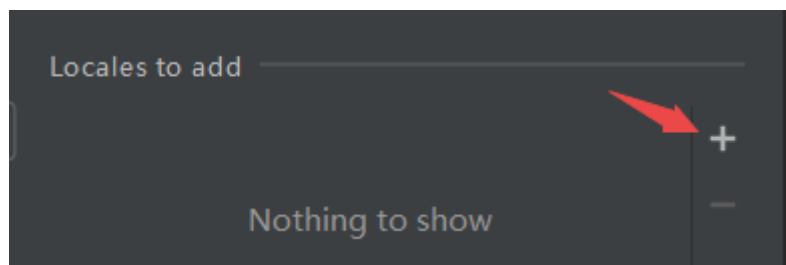
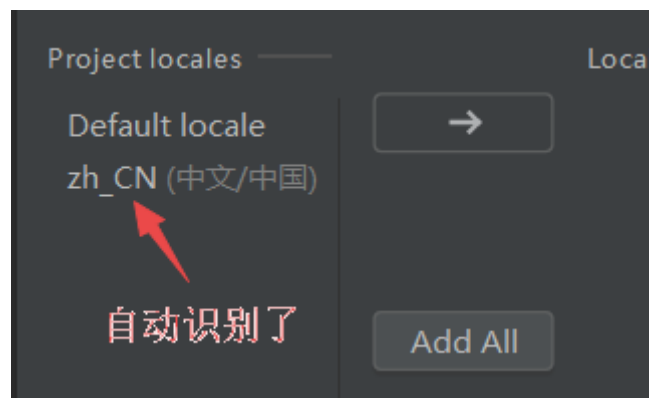



IDEA会帮我们自动合并：

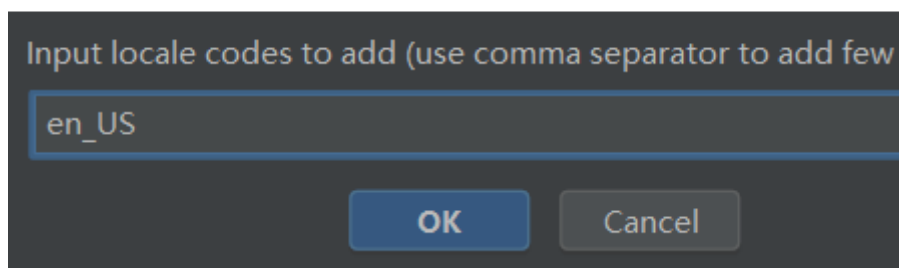


- 再创建一个英文的
- 神奇的事情发生了：

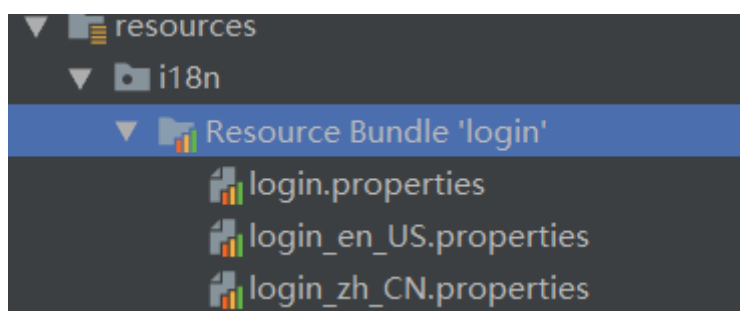
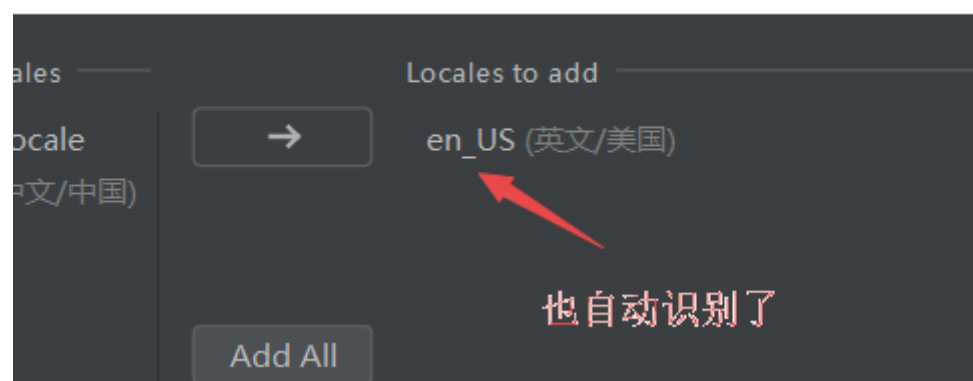




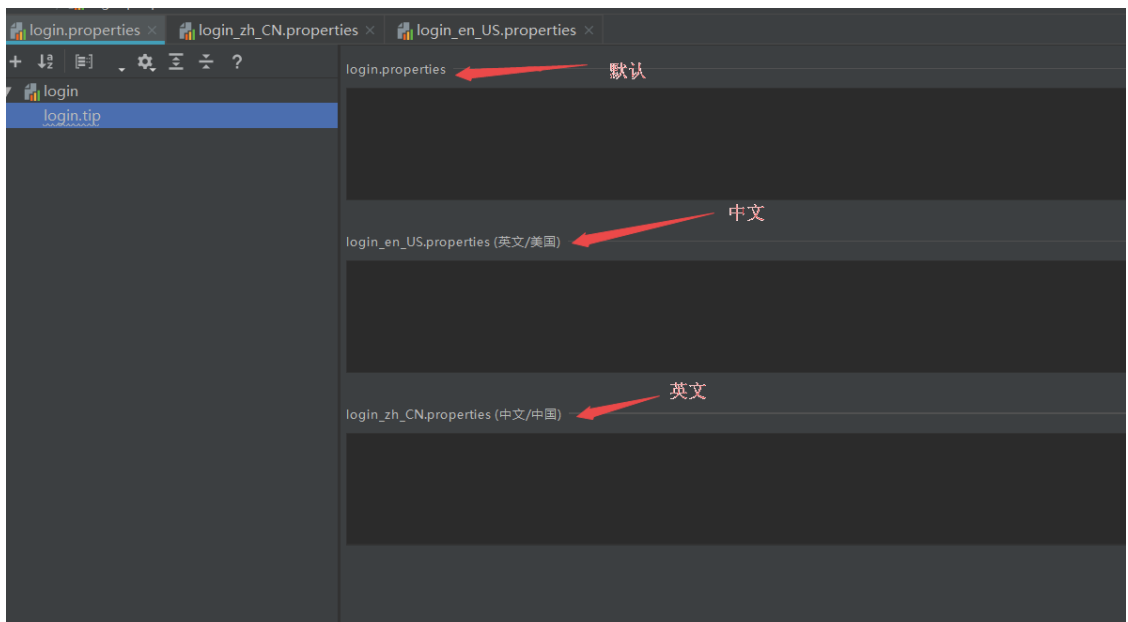
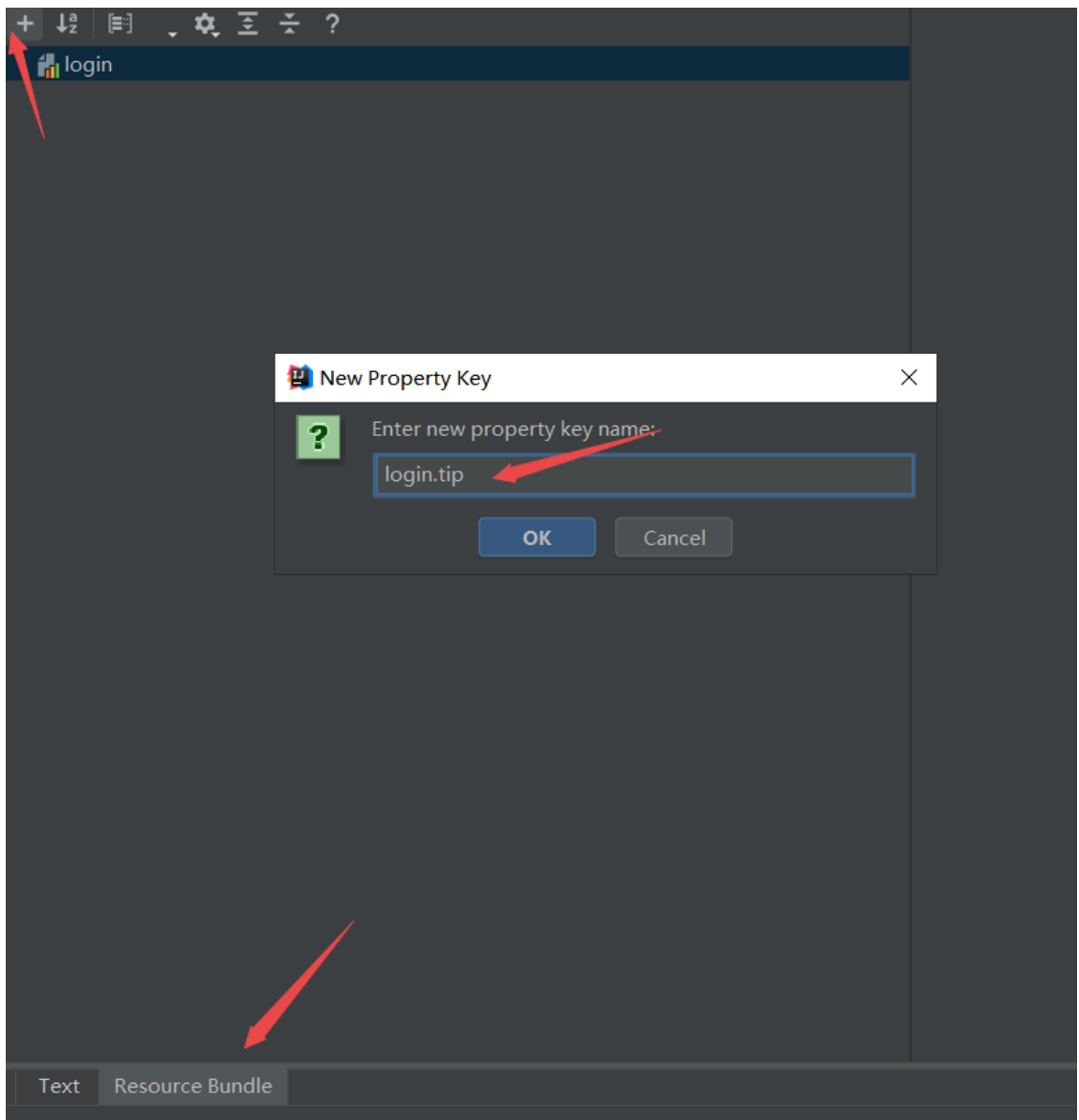
 Add Locales

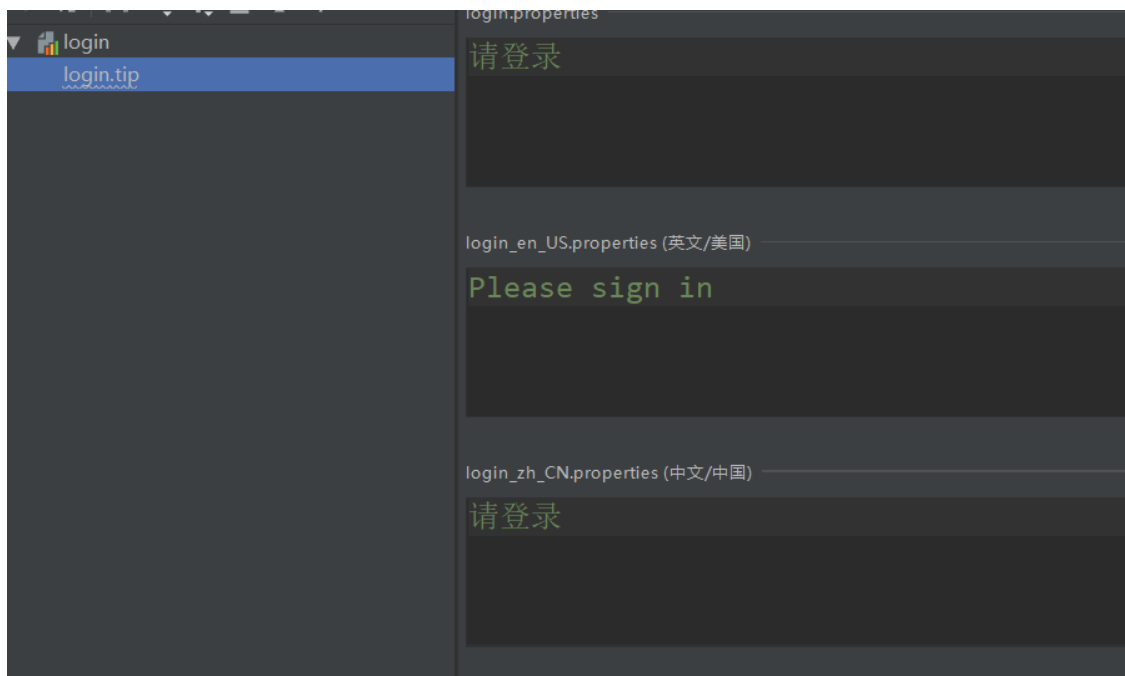


Locales to Resource Bundle login



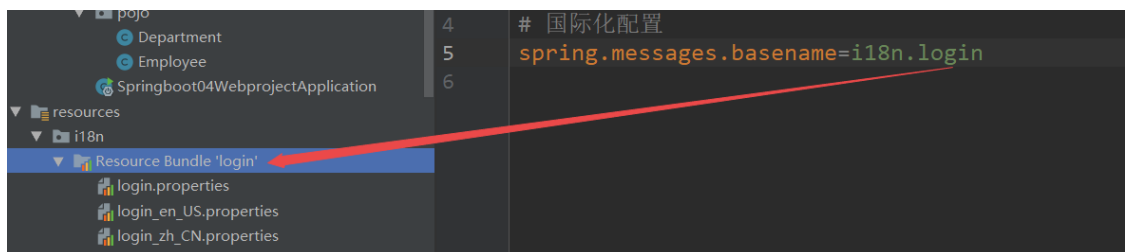
- 配置文件编写





剩下的如上进行配置即可。

- 键可以随便起（就是上面的login.tip）
- 在application.properties绑定



- 页面修改

模板引擎语法：

- Variable Expressions: `${...}`
- Selection Variable Expressions: `*{...}`
- Message Expressions: `#{...}`
- Link URL Expressions: `@{...}`
- Fragment Expressions: `~{...}`

## • Literals

消息用： `#{}`

例如：

```
1 | <h1 class="h3 mb-3 font-weight-normal">Please sign in</h1>
```

改为：

```
1 | <h1 class="h3 mb-3 font-weight-normal" th:text="#{login.tip}">Please sign
  | in</h1>
```

`#{}` 里面是键

看一下初步效果：



请登录

---

注意一种情况：

如果值是在外面的，比如：

```
1 | <input type="checkbox" value="remember-me"> Remember me
```

就不可以用：

```
1 | <input type="checkbox" value="remember-me" th:text="#{login.remember}">
   | Remember me
```

应该用：

```
1 | <input type="checkbox" value="remember-me"> [[#{login.remember}]]
```

`[[ ]]`是固定的，用于识别Thymeleaf，这个后面会很常用。里面放的是Thymeleaf取出的值。

所以要根据情况在合适的位置进行取值！





## 请登录

☐ 记住我

登录

© 2017-2018

中文 English

- 写一个解析器，便于进行切换语言

```
1 package com.kuang.config;
2
3 import org.springframework.web.servlet.LocaleResolver;
4 import org.thymeleaf.util.StringUtils;
5
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import java.util.Locale;
9
10 /**
11  * 自己的国际化解析器
12  */
13 public class MyLocaleResolver implements LocaleResolver {
14     /**
15      * 解析请求
16      * th:href="@{/index.html(lan='zh_CN')}"
17      * th:href="@{/index.html(lan='en_US')}"
18      *
19      * @param request
20      * @return
21      */
22     @Override
23     public Locale resolveLocale(HttpServletRequest request) {
24         //获取请求中的语言参数
```

```

25         String language = request.getParameter("lan");
26
27         Locale locale = Locale.getDefault();//如果没有就使用默认的
28
29         if (!StringUtils.isEmpty(language)) { //language不为空==》请求的链
//携带了国际化的参数
30             //获得两部分数据，国家 & 地区
31             String[] s = language.split("_");
32
33             //设置国家地区
34             Locale myLocale = new Locale(s[0], s[1]);
35
36             return myLocale;
37         }
38
39         return locale;
40     }
41
42     @Override
43     public void setLocale(HttpServletRequest request,
44                           HttpServletResponse response, Locale locale) {
45     }
46 }

```

页面处理:

```

<a class="btn btn-sm" th:href="@{/index.html(lan='zh_CN')}">中文</a>
<a class="btn btn-sm" th:href="@{/index.html(lan='en_US')}">English</a>

```

- 将写的组件注册到Spring容器中

```

1  package com.kuang.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.web.servlet.LocaleResolver;
6  import
org.springframework.web.servlet.config.annotation.ViewControllerRegistr
y;
7  import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
8
9  @Configuration
10 public class MyMVCConfig implements WebMvcConfigurer {
11     /**
12      * 自定义的国际化
13      *
14      * @return
15      */
16     @Bean
17     public LocaleResolver localeResolver() {
18         return new MyLocaleResolver();
19     }
20 }

```

- 测试

点击中文:



请登录

用户名

密码

☐ 记住我

登录

© 2017-2018

[中文](#) [English](#)

点击English:



Please sign in

|          |
|----------|
| Username |
| Password |

☐ Remember me

Sign in

© 2017-2018

中文 English

## 6.3 注意点

- 需要配置i18n文件
- 如果需要在项目中进行按钮自动切换语言，我们需要自定义一个组件（实现接口LocaleResolver，重写方法）
- 记得将写的组件注册到Spring容器中（@Bean）

## 7 拦截器

### 7.1 分析问题

登录的Controller:

```
1 @RequestMapping("/user/login")
2 public String login(@RequestParam("username") String username,
3                     @RequestParam("password") String password,
4                     Model model) {
5     //具体的业务
6     if (!StringUtils.isEmpty(username) && "123456".equals(password)) {
```

```

7         return "redirect:/main.html";//登录成功，重定向跳转到主页
8     } else {
9         //告诉用户，登录失败
10        model.addAttribute("msg", "用户名或者密码错误");
11        return "index";
12    }
13
14 }

```

自定义的视图解析器：

```

1  /**
2   * 自定义的视图解析器
3   *
4   * @param registry
5   */
6  @Override
7  public void addViewControllers(ViewControllerRegistry registry) {
8      registry.addViewController("/main.html").setViewName("dashboard");
9  }

```

但我们输入用户名不为空，密码为“123456”时登录成功，重定向到main.html请求。这个请求在视图解析器中会解析，然后跳转到我们的主页dashboard。

但是这样子做会产生一个问题：即使没有登录，我们也可以通过直接发送main.html请求直接进入首页，显然不符合规范，于是我们在进入主页前添加拦截器！

## 7.2 解决问题

- 自己写一个拦截器（也是组件）：【继承接口HandlerInterceptor，重写preHandle方法即可】

记得在用户登录成功那里用session存一下信息，方便判断是否登录。

```

public class LoginController {
    @RequestMapping("/user/login")
    public String login(@RequestParam("username") String username,
                       @RequestParam("password") String password,
                       Model model,
                       HttpSession session) {
        //具体的业务
        if (!StringUtils.isEmpty(username) && "123456".equals(password)) {
            //存入Session
            session.setAttribute(s: "loginInfo", username);
            return "redirect:/main.html";//登录成功，重定向跳转到主页
        } else {
            //告诉用户，登录失败
            model.addAttribute(attributeName: "msg", attributeValue: "用户名或者密码错误");
            return "index";
        }
    }
}

```

```

1 package com.kuang.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.HandlerInterceptor;
5
6 import javax.servlet.http.HttpServletRequest;

```

```

7  import javax.servlet.http.HttpServletResponse;
8
9  @Configuration
10 public class LoginHandlerInterceptor implements HandlerInterceptor {
11     /**
12      * return true 放行
13      * return false 不放行
14      * <p>
15      * 所以我们要分析什么情况下放行，什么情况下不放行
16      *
17      * @param request
18      * @param response
19      * @param handler
20      * @return
21      * @throws Exception
22      */
23     @Override
24     public boolean preHandle(HttpServletRequest request,
25                             HttpServletResponse response, Object handler) throws Exception {
26         //登录成功，应该有用户的session===》放行
27         Object loginInfo = request.getSession().getAttribute("loginInfo");
28
29         if (null == loginInfo) { //没有登录
30             request.setAttribute("msg", "没有权限，请先登录");
31             request.getRequestDispatcher("/index.html").forward(request,
32 response);
33             return false;
34         } else {
35             return true;
36         }
37     }
38 }

```

- 注册组件到WebMvcConfigurer (和之前的视图解析器一个类中)

```

1  package com.kuang.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.web.servlet.LocaleResolver;
6  import
7  org.springframework.web.servlet.config.annotation.InterceptorRegistry;
8  import
9  org.springframework.web.servlet.config.annotation.ViewControllerRegistr
10 y;
11 import
12 org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
13
14 @Configuration
15 public class MyMVConfig implements WebMvcConfigurer {
16     /**
17      * 自定义拦截器
18      * <p>
19      * addPathPatterns ===》拦截哪些请求    /**代表拦截所有
20      * excludePathPatterns ===》放行哪些请求

```

```

17      * <p>
18      * 注意，这里写的都是请求！
19      *
20      * @param registry
21      */
22      @Override
23      public void addInterceptors(InterceptorRegistry registry) {
24          registry.addInterceptor(new LoginHandlerInterceptor())
25              .addPathPatterns("/**")
26              .excludePathPatterns("/index.html", "/", "/user/login",
27                                  "/css/*", "/img/*", "/js/*");
28      }
29  }

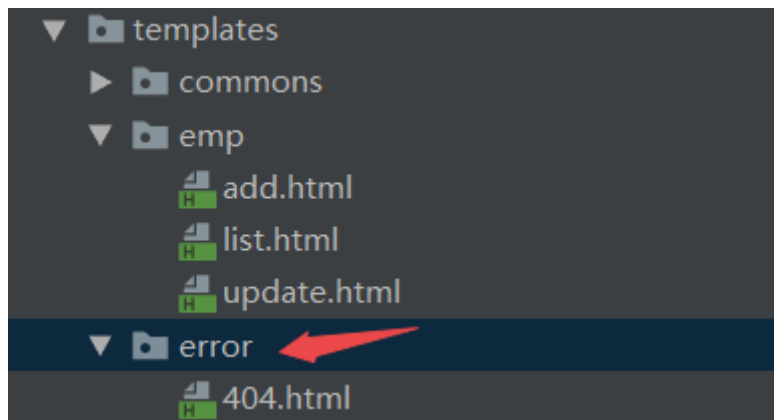
```

addPathPatterns：是代表拦截哪些请求

excludePathPatterns：不拦截哪些请求（放行）===》静态资源记得放行

## 8 404页面处理

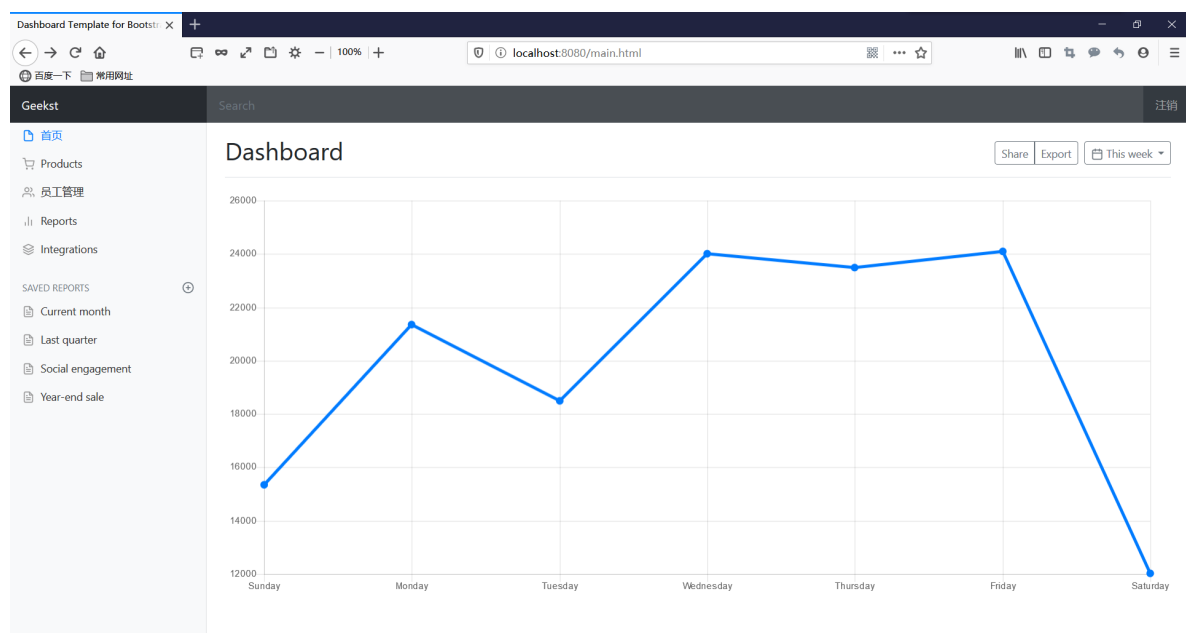
在templates的目录下创建一个“error”目录，注意名字规范，要一致！



error放置遇到404时跳转的页面。

测试一下：

登录成功后：



输入一个不存在的请求或者网页：

<http://localhost:8080/main.html#deddsd>

自动跳转到我们准备的404界面：

