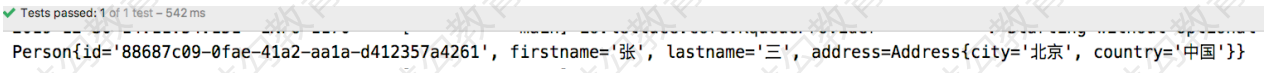
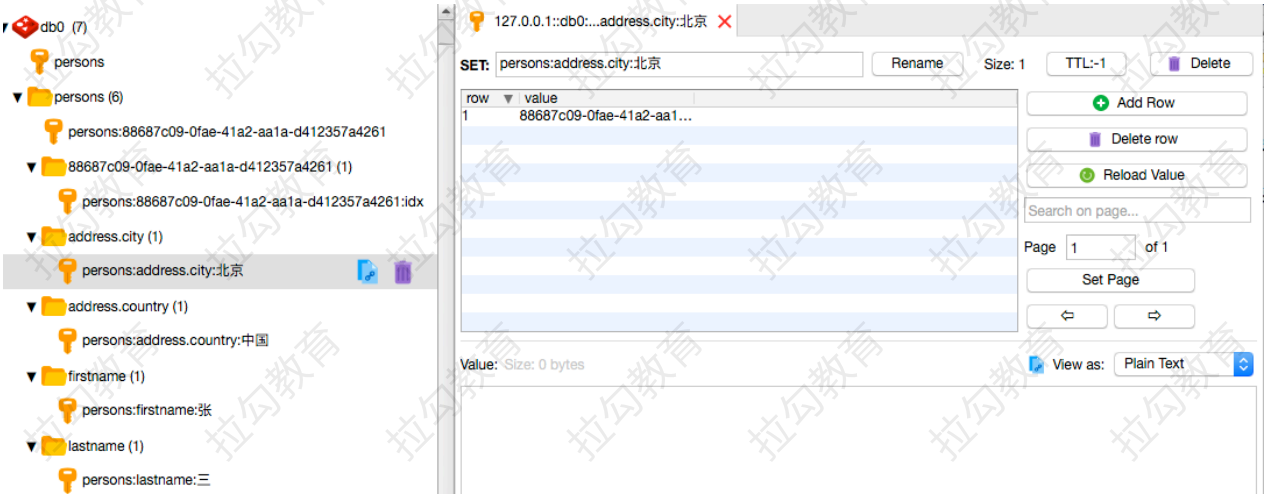
```
        address.setCity("北京");
        address.setCountry("中国");
        person.setAddress(address);

        // 向Redis数据库添加数据
        Person save = repository.save(person);

    }

    @Test
    public void selectPerson() {
        List<Person> list = (List<Person>) repository.findByAddress_City("北京");
        for (Person person : list) {
            System.out.println(person);
        }
    }
```

整合测试：



为了验证savePerson()方法的执行效果，还可以打开之前连接的Redis客户端可视化管理工具查看数据，效果如图 (可能需要Reload刷新)



执行savePerson()方法添加的数据在Redis数据库中存储成功。另外，在数据库列表左侧还生成了一些类似address.city、firstname、lastname等二级索引，这些二级索引是前面创建Person类时在对应属性上添加@Indexed注解而生成的。同时，由于在Redis数据库中生成了对应属性的二级索引，所以可以通过二级索引来查询具体的数据信息，例如repository.findByAddress_City("北京")通过address.city索引查询索引值为“北京”的数据信息。如果没有设置对应属性的二级索引，那么通过属性索引查询数据结果将会为空

4. SpringBoot视图技术

4.1 支持的视图技术

前端模板引擎技术的出现，使前端开发人员无需关注后端业务的具体实现，只关注自己页面的呈现效果即可，并且解决了前端代码错综复杂的问题、实现了前后端分离开发。Spring Boot框架对很多常用的模板引擎技术（如：FreeMarker、Thymeleaf、Mustache等）提供了整合支持

Spring Boot不太支持常用的JSP模板，并且没有提供对应的整合配置，这是因为使用嵌入式Servlet容器的Spring Boot应用程序对于JSP模板存在一些限制：

- Spring Boot默认使用嵌入式Servlet容器以JAR包方式进行项目打包部署，这种JAR包方式不支持JSP模板。
- 如果使用Undertow嵌入式容器部署Spring Boot项目，也不支持JSP模板。
- Spring Boot默认提供了一个处理请求路径“/error”的统一错误处理器，返回具体的异常信息。使用JSP模板时，无法对默认的错误处理器进行覆盖，只能根据Spring Boot要求在指定位置定制错误页面。

上面对Spring Boot支持的模板引擎进行了介绍，并指出了整合JSP模板的一些限制。接下来，对其中常用的Thymeleaf模板引擎进行介绍，并完成与Spring Boot框架的整合实现

4.2 Thymeleaf

Thymeleaf是一种现代的基于服务器端的Java模板引擎技术，也是一个优秀的面向Java的XML、XHTML、HTML5页面模板，它具有丰富的标签语言、函数和表达式，在使用Spring Boot框架进行页面设计时，一般会选择Thymeleaf模板

4.2.1 Thymeleaf语法

常用标签

在HTML页面上使用Thymeleaf标签，Thymeleaf 标签能够动态地替换掉静态内容，使页面动态展示。

为了大家更直观的认识Thymeleaf，下面展示一个在HTML文件中嵌入了Thymeleaf的页面文件，示例代码如下：

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" media="all"
          href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />
    <title>Title</title>
</head>
<body>
    <p th:text="${hello}">欢迎进入Thymeleaf的学习</p>
</body>
</html>
```

上述代码中，“xmlns:th="http://www.thymeleaf.org"“ 用于引入Thymeleaf模板引擎标签，使用关键字“th”标注标签是Thymeleaf模板提供的标签，其中，“th:href”用于引入外联样式文件，“th:text”用于动态显示标签文本内容。

除此之外，Thymeleaf模板提供了很多标签，接下来，通过一张表罗列Thymeleaf的常用标签

th：标签	说明
th:insert	布局标签，替换内容到引入的文件
th:replace	页面片段包含（类似JSP中的include标签）
th:each	元素遍历（类似JSP中的c:forEach标签）
th:if	条件判断，如果为真
th:unless	条件判断，如果为假
th:switch	条件判断，进行选择性匹配
th:case	条件判断，进行选择性匹配
th:value	属性值修改，指定标签属性值
th:href	用于设定链接地址
th:src	用于设定链接地址
th:text	用于指定标签显示的文本内容

标准表达式

Thymeleaf模板引擎提供了多种标准表达式语法，在正式学习之前，先通过一张表来展示其主要语法及说明

说明	表达式语法
变量表达式	${...}
选择变量表达式	*{...}
消息表达式	#{...}
链接URL表达式	@{...}
片段表达式	~{...}

1. 变量表达式 ${...}

变量表达式${...}主要用于获取上下文中的变量值，示例代码如下：

```
<p th:text="${title}">这是标题</p>
```

示例使用了Thymeleaf模板的变量表达式${...}用来动态获取P标签中的内容，如果当前程序没有启动或者当前上下文中不存在title变量，该片段会显示标签默认值“这是标题”；如果当前上下文中存在title变量并且程序已经启动，当前P标签中的默认文本内容将会被title变量的值所替换，从而达到模板引擎页面数据动态替换的效果

同时，Thymeleaf为变量所在域提供了一些内置对象，具体如下所示

```
# ctx：上下文对象
# vars：上下文变量
# locale：上下文区域设置
# request：（仅限Web Context）HttpServletRequest对象
# response：（仅限Web Context）HttpServletResponse对象
# session：（仅限Web Context）HttpSession对象
# servletContext：（仅限Web Context）ServletContext对象
```

结合上述内置对象的说明，假设要在Thymeleaf模板引擎页面中动态获取当前国家信息，可以使用#locale内置对象，示例代码如下

```
The locale country is: <span th:text="${#locale.country}">US</span>.
```

上述代码中，使用th:text="${#locale.country}"动态获取当前用户所在国家信息，其中标签内默认内容为US（美国），程序启动后通过浏览器查看当前页面时，Thymeleaf会通过浏览器语言设置来识别当前用户所在国家信息，从而实现动态替换

2. 选择变量表达式 *{...}

选择变量表达式和变量表达式用法类似，一般用于从被选定对象而不是上下文中获取属性值，如果没有选定对象，则和变量表达式一样，示例代码如下

```
<div th:object="${book}">
  <p>titile: <span th:text="*{title}">标题</span>.</p>
</div>
```

*{title} 选择变量表达式获取当前指定对象book的title属性值。

3. 消息表达式 #{...}

消息表达式#{...}主要用于Thymeleaf模板页面国际化内容的动态替换和展示，使用消息表达式#{...}进行国际化设置时，还需要提供一些国际化配置文件。关于消息表达式的使用，后续会详细说明

4. 链接表达式 @{...}

链接表达式@{...}一般用于页面跳转或者资源的引入，在Web开发中占据着非常重要的地位，并且使用也非常频繁，示例代码如下：

```
<a  th:href="@{http://localhost:8080/order/details(orderId=${o.id})}">view</a>
<a  th:href="@{/order/details(orderId=${o.id})}">view</a>
```

上述代码中，链接表达式@{...}分别编写了绝对链接地址和相对链接地址。在有参表达式中，需要按照@{路径(参数名称=参数值，参数名称=参数值...)}的形式编写，同时该参数的值可以使用变量表达式来传递动态参数值

5. 片段表达式 ~{...}

片段表达式~{...}用来标记一个片段模板，并根据需要移动或传递给其他模板。其中，最常见的用法是使用th:insert或th:replace属性插入片段，示例代码如下：

```
<div th:insert="~{thymeleafDemo::title}"></div>
```

上述代码中，使用th:insert属性将title片段模板引用到该

标签中。thymeleafDemo为模板名称，Thymeleaf会自动查找“/resources/templates/”目录下的thymeleafDemo模板，title为片段名称

4.2.2 基本使用

（1） Thymeleaf模板基本配置

首先 在Spring Boot项目中使用Thymeleaf模板，首先必须保证引入Thymeleaf依赖，示例代码如下：

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
 </dependency>
```

其次，在全局配置文件中配置Thymeleaf模板的一些参数。一般Web项目都会使用下列配置，示例代码如：

```
spring.thymeleaf.cache = true        #启用模板缓存
spring.thymeleaf.encoding = UTF-8    #模板编码
spring.thymeleaf.mode = HTML5    #应用于模板的模板模式
spring.thymeleaf.prefix = classpath:/templates/   #指定模板页面存放路径
spring.thymeleaf.suffix = .html    #指定模板页面名称的后缀
```

上述配置中，spring.thymeleaf.cache表示是否开启Thymeleaf模板缓存，默认为true，在开发过程中通常会关闭缓存，保证项目调试过程中数据能够及时响应；spring.thymeleaf.prefix指定了Thymeleaf模板页面的存放路径，默认为classpath:/templates/；spring.thymeleaf.suffix指定了Thymeleaf模板页面的名称后缀，默认为.html

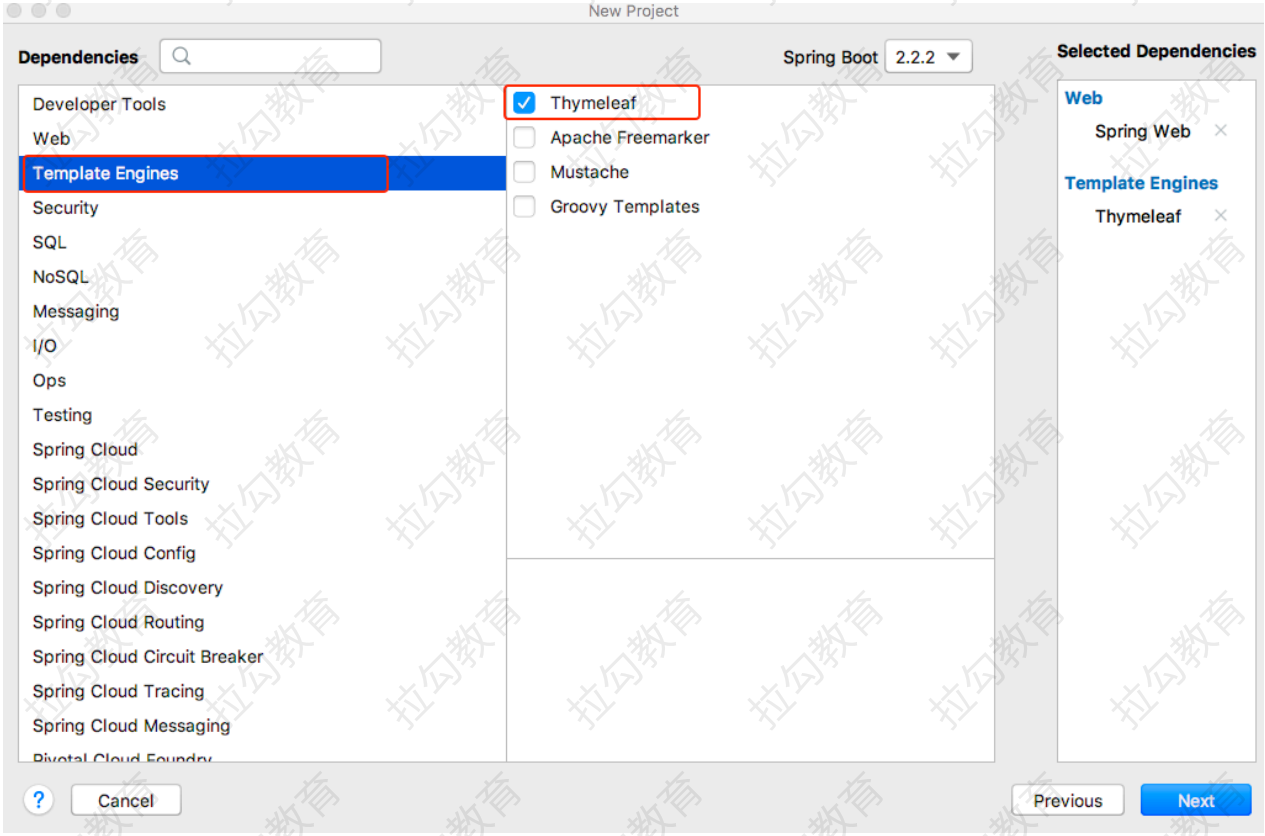
（2）静态资源的访问

开发Web应用时，难免需要使用静态资源。Spring boot默认设置了静态资源的访问路径。

使用Spring Initializr方式创建的Spring Boot项目，默认生成了一个resources目录，在resources目录中新建public、resources、static三个子目录下，Spring boot默认会挨个从public、resources、static里面查找静态资源

4.2.3 完成数据的页面展示

1. 创建Spring Boot项目，引入Thymeleaf依赖



2. 编写配置文件

打开application.properties全局配置文件，在该文件中对Thymeleaf模板页面的数据缓存进行设置

```
  # thymeleaf页面缓存设置（默认为true），开发中方便调试应设置为false，上线稳定后应保持默认true
  spring.thymeleaf.cache=false
```

使用“spring.thymeleaf.cache=false”将Thymeleaf默认开启的缓存设置为了false，用来关闭模板页面缓存

3. 创建web控制类

在项目中创建名为com.lagou.controller的包，并在该包下创建一个用于前端模板页面动态数据替换效果测试的访问实体类LoginController

```
@Controller
public class LoginController {

    /**
         * 获取并封装当前年份跳转到登录页login.html
         */

    @RequestMapping("/toLoginPage")
    public String toLoginPage(Model model){
        model.addAttribute("currentYear",
Calendar.getInstance().get(Calendar.YEAR));
        return "login";
    }
```

toLoginPage()方法用于向登录页面login.html跳转，同时携带了当前年份信息currentYear。

4. 创建模板页面并引入静态资源文件

在“classpath:/templates/”目录下引入一个用户登录的模板页面login.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,shrink-
to-fit=no">
    <title>用户登录界面</title>
    <link th:href="@{/login/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{/login/css/signin.css}" rel="stylesheet">
</head>
<body class="text-center">
<!--  用户登录form表单 -->
<form class="form-signin">
    <img class="mb-4" th:src="@{/login/img/login.jpg}" width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal">请登录</h1>
    <input type="text" class="form-control"
           th:placeholder="用户名" required="" autofocus="">
    <input type="password" class="form-control"
           th:placeholder="密码" required="">
    <div class="checkbox mb-3">
        <label>
            <input type="checkbox" value="remember-me"> 记住我
        </label>
    </div>
    <button class="btn btn-lg btn-primary btn-block" type="submit" >登
录</button>
    <p class="mt-5 mb-3 text-muted">© <span
th:text="${currentYear}">2019</span>-<span
th:text="${currentYear}+1">2020</span></p>
```

```
</form>
</body>
</html>
```

通过“xmlns:th="http://www.thymeleaf.org""引入了Thymeleaf模板标签；

使用“th:href”和“th:src”分别引入了两个外联的样式文件和一个图片；

使用“th:text”引入了后台动态传递过来的当前年份currentYear

5．效果测试

可以看出，登录页面login.html显示正常，在文件4-3中使用“th:*”相关属性引入的静态文件生效，并且在页面底部动态显示了当前日期2019-2020，而不是文件中的静态数字2019-2020。这进一步说明了Spring Boot与Thymeleaf整合成功，完成了静态资源的引入和动态数据的显示

4.2.4 配置国际化页面

1．编写多语言国际化配置文件

在项目的类路径resources下创建名称为i18n的文件夹，并在该文件夹中根据需要编写对应的多语言国际化文件login.properties、login_zh_CN.properties和login_en_US.properties文件

login.properties

```
login.tip=请登录
login.username=用户名
login.password=密码
login.rememberme=记住我
login.button=登录
```

login_zh_CN.properties

```
login.tip=请登录
login.username=用户名
login.password=密码
login.rememberme=记住我
login.button=登录
```

login_en_US.properties

```
login.tip=Please sign in
login.username=Username
login.password=Password
login.rememberme=Remember me
login.button=Login
```

login.properties为自定义默认语言配置文件，login_zh_CN.properties为自定义中文国际化文件，login_en_US.properties为自定义英文国际化文件

需要说明的是，Spring Boot默认识别的语言配置文件为类路径resources下的messages.properties；其他语言国际化文件的名称必须严格按照“文件前缀名*语言代码*国家代码.properties”的形式命名

本示例中，在项目类路径resources下自定义了一个i18n包用于统一配置管理多语言配置文件，并将项目默认语言配置文件名自定义为login.properties，因此，后续还必须在项目全局配置文件中进行国际化文件基础名配置，才能引用自定义国际化文件

2. 编写配置文件

打开项目的application.properties全局配置文件，在该文件中添加国际化文件基础名设置，内容如文件

```
# 配置国际化文件基础名
spring.messages.basename=i18n.login
```

spring.messages.basename=i18n.login”设置了自定义国际化文件的基础名。其中，i18n表示国际化文件相对项目类路径resources的位置，login表示多语言文件的前缀名。如果开发者完全按照Spring Boot默认识别机制，在项目类路径resources下编写messages.properties等国际化文件，可以省略国际化文件基础名的配置

3. 定制区域信息解析器

在完成上一步中多语言国际化文件的编写和配置后，就可以正式在前端页面中结合Thymeleaf模板相关属性进行国际化语言设置和展示了，不过这种实现方式默认是使用请求头中的语言信息（浏览器语言信息）自动进行语言切换的，有些项目还会提供手动语言切换的功能，这就需要定制区域解析器了

在项目中创建名为com.lagou.config的包，并在该包下创建一个用于定制国际化功能区域信息解析器的自定义配置类MyLocalResovel

```
package com.lagou.config;
```

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.lang.Nullable;
import org.springframework.util.StringUtils;
import org.springframework.web.servlet.LocaleResolver;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Locale;


@Configuration
public class MyLocaleResovel implements LocaleResolver {
    // 自定义区域解析方式
    @Override
    public Locale resolveLocale(HttpServletRequest httpServletRequest) {
        // 获取页面手动切换传递的语言参数l
        String l = httpServletRequest.getParameter("l");
        // 获取请求头自动传递的语言参数Accept-Language
        String header = httpServletRequest.getHeader("Accept-Language");
        Locale locale=null;
        // 如果手动切换参数不为空，就根据手动参数进行语言切换，否则默认根据请求头信息切换
        if(!StringUtils.isEmpty(l)){
            String[] split = l.split("_");
            locale=new Locale(split[0],split[1]);
        }else {
            // Accept-Language: en-US,en;q=0.9
          ,zh-CN;q=0.8,zh;q=0.7
            String[] splits = header.split(",");
            String[] split = splits[0].split("-");
            locale=new Locale(split[0],split[1]);
        }
        return locale;
    }
    @Override
    public void setLocale(HttpServletRequest httpServletRequest, @Nullable
            HttpServletResponse httpServletResponse, @Nullable Locale locale)
{
    }
    // 将自定义的MyLocalResovel类重新注册为一个类型LocaleResolver的Bean组件
    @Bean
    public LocaleResolver localeResolver(){
        return new MyLocalResovel();
    }
}
```

MyLocalResolver自定义区域解析器配置类实现了LocaleResolver接口，并重写了其中的resolveLocale()方法进行自定义语言解析，最后使用@Bean注解将当前配置类注册成Spring容器中的一个类型为LocaleResolver的Bean组件，这样就可以覆盖默认的LocaleResolver组件。其中，在resolveLocale()方法中，根据不同需求（手动切换语言信息、浏览器请求头自动切换语言信息）分别获取了请求参数l和请求头参数Accept-Language，然后在请求参数l不为空的情况下就以l参数携带的语言为标准进行语言切换，否则就定制通过请求头信息进行自动切换。

需要注意的是，在请求参数l的语言手动切换组装时，使用的是下划线“_”进行的切割，这是由多语言配置文件的格式决定的（例如login_zh_CN.properties）；而在请求头参数Accept-Language的语言自动切换组装时，使用的是短横线“-”进行的切割，这是由浏览器发送的请求头信息样式决定的（例如Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7）

4. 页面国际化使用

打开项目templates模板文件夹中的用户登录页面login.html，结合Thymeleaf模板引擎实现国际化功能

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,shrink-
to-fit=no">
    <title>用户登录界面</title>
    <link th:href="@{/login/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{/login/css/signin.css}" rel="stylesheet">
</head>
<body class="text-center">
<!--  用户登录form表单 -->
<form class="form-signin">
    <img class="mb-4" th:src="@{/login/img/login.jpg}" width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal" th:text="#{login.tip}">请登录</h1>
    <input type="text" class="form-control"
           th:placeholder="#{login.username}" required="" autofocus="">
    <input type="password" class="form-control"
           th:placeholder="#{login.password}" required="">
    <div class="checkbox mb-3">
        <label>
            <input type="checkbox" value="remember-me"> [[#
{login.rememberme}]]
        </label>
    </div>
    <button class="btn btn-lg btn-primary btn-block" type="submit" th:text="#
{login.button}">登录</button>
    <p class="mt-5 mb-3 text-muted">© <span
th:text="${currentYear}">2018</span>-<span
th:text="${currentYear}+1">2019</span></p>
    <a class="btn btn-sm" th:href="@{/toLoginPage(l='zh_CN')}">中文</a>
    <a class="btn btn-sm" th:href="@{/toLoginPage(l='en_US')}">English</a>
```

```
</form>
</body>
</html>
```

使用Thymeleaf模板的#{}消息表达式设置了国际化展示的部分信息。在对记住我rememberme国际化设置时，需要国际化设置的rememberme在 标签外部，所以这里使用了行内表达式[[#{login.rememberme}]]动态获取国际化文件中的login.rememberme信息。另外，在表单尾部还提供了中文、English手动切换语言的功能链接，在单击链接时会分别携带国家语言参数向“/”路径请求跳转，通过后台定制的区域解析器进行手动语言切换

5. 整合效果测试