第5章 Spring Boot实现Web MVC

广东财经大学信息学院
罗 东 俊 博士
ZSUJONE@126.COM

（内部资料，请勿外传）

目的和要求

- 熟悉Thymeleaf模板引擎基本语法。
- 掌握Spring Boot整合Thymeleaf模板引擎的使用。
- 掌握Spring Boot整合Spring MVC的使用。
- 掌握拦截器的作用和使用方法。
- 掌握Spring Boot中MVC功能的定制。

主要内容

- 5.1 MVC设计概述
- 5.2 使用视图技术Thymeleaf
- 5.3 使用控制器
- 5.4 使用拦截器
- 5.5 自定义Web MVC配置

5.1 MVC设计概述

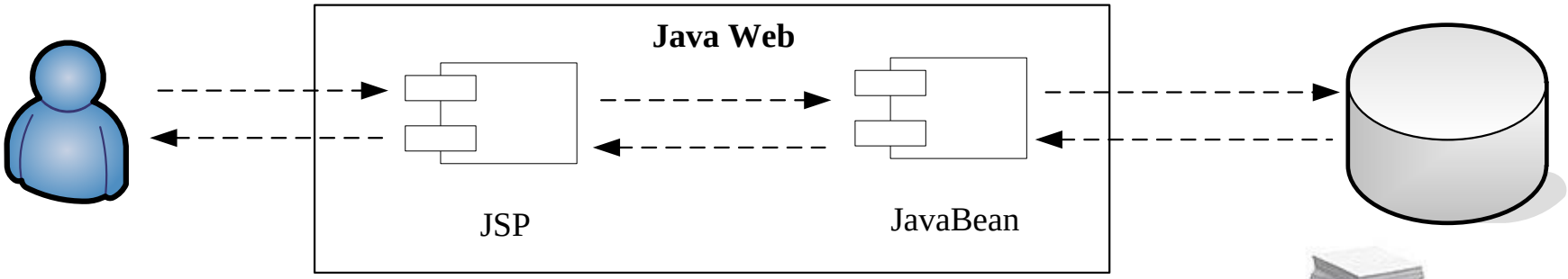
- 5.1.1 Java Web框架的演变
- 5.1.2 Spring MVC框架
- 5.1.3 Spring MVC的整合支持

5.1.1 Java Web框架的演变

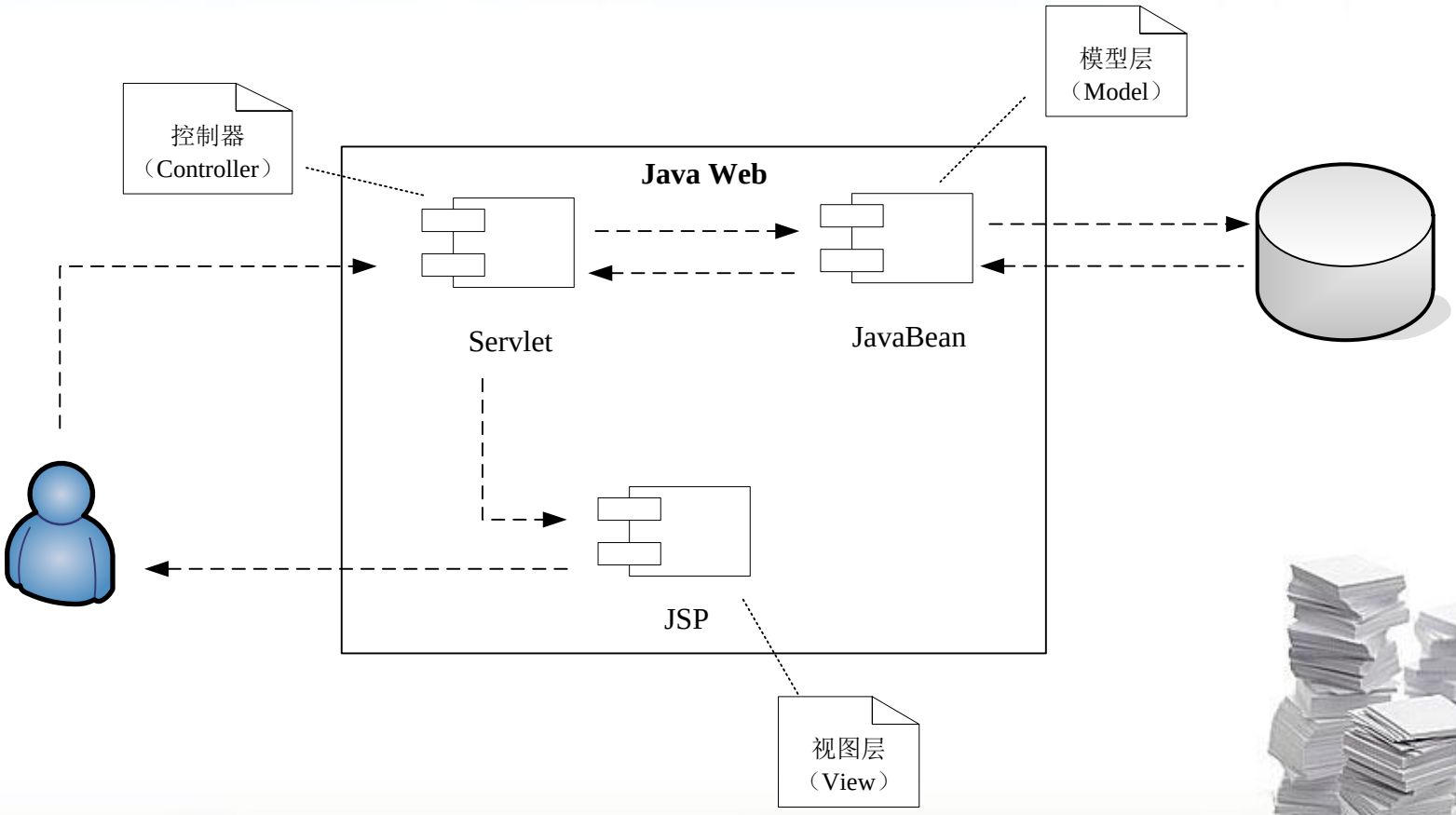
- **MVC框架是一种设计理念，它不仅存在于Java Web应用中，而且还广泛存在于各类语言和开发中，比如前端、PHP、.Net等。**
- **其根本目的在于解耦各个模块。**

1.原始的Java Web

- 非MVC架构，JSP和JavaBean之间严重耦合，Java和HTML也耦合。

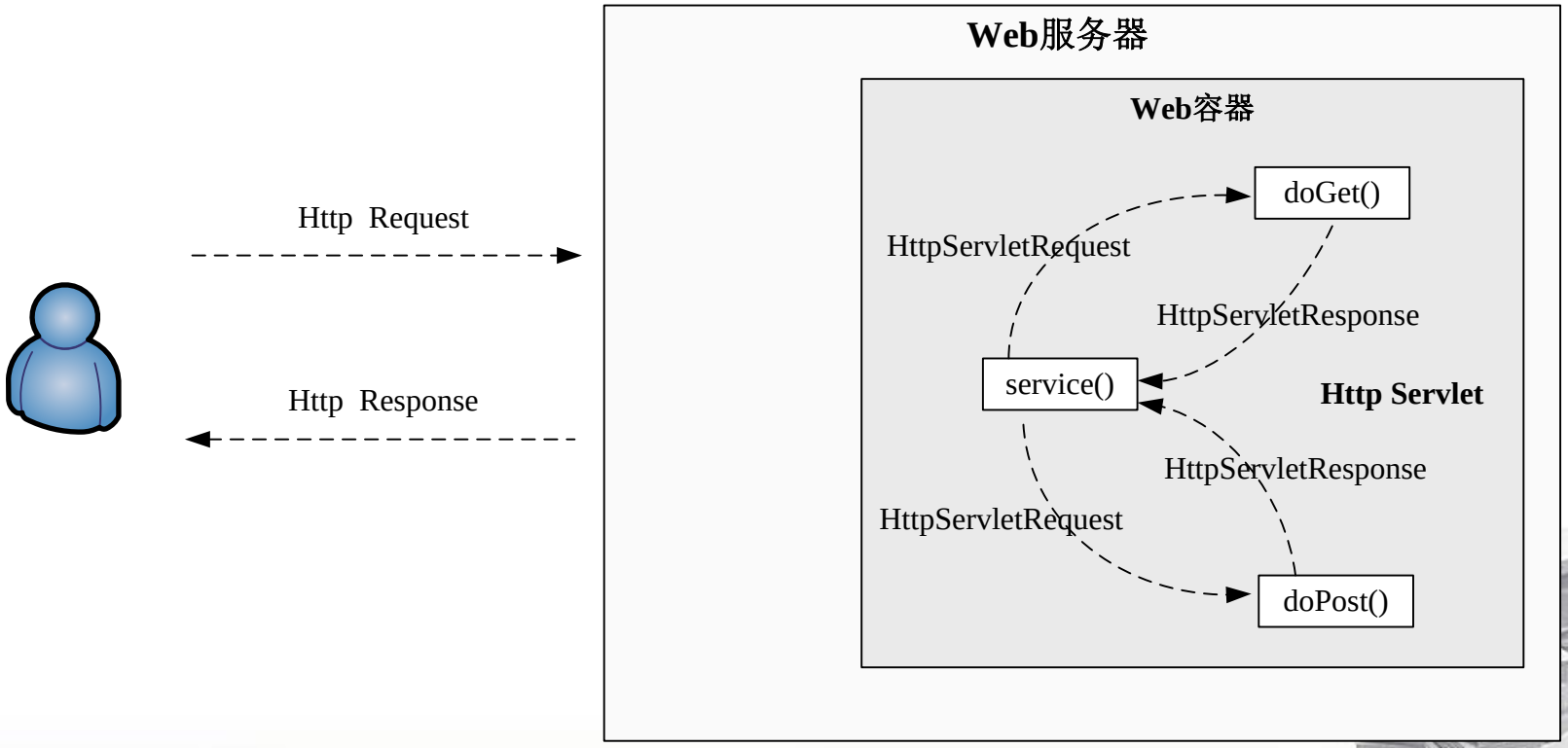


2.早期的Web MVC模型

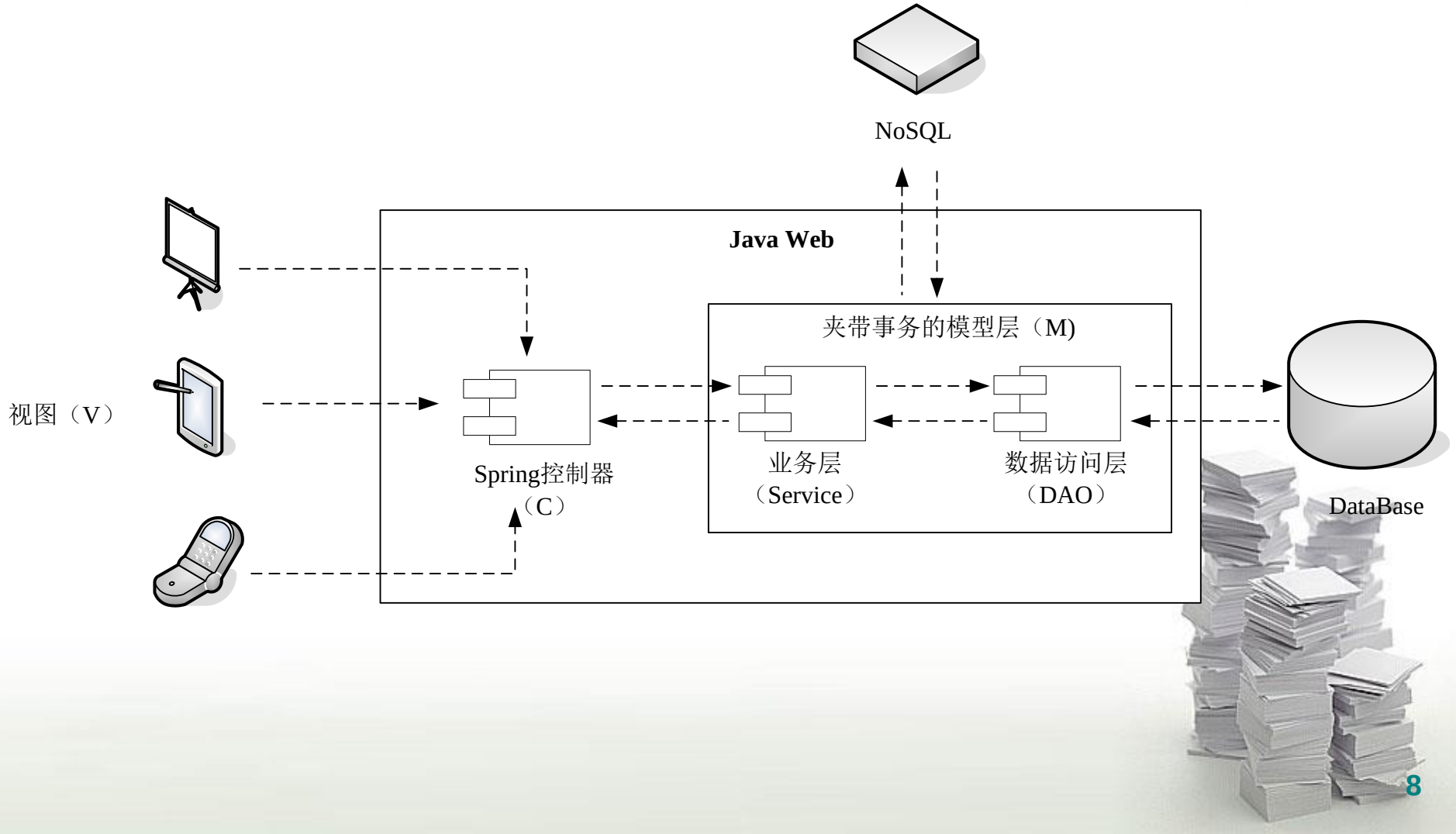


仍存在一定耦合，Java依赖于页面编程

Servlet工作原理



3.Spring Web MVC模型



4. Spring WebFlux模型

- Spring WebFlux是从Spring Framework 5.0开始引入的异步响应式Web框架。
- WebFlux的工作流程：
  - ✧主线程收到请求→立刻返回数据与函数的组合（Mono或Flux，不是结果）→开启一个新Work线程去做实际的数据准备工作，进行真正的业务操作→Work线程完成工作→返回给用户真实数据（结果）
- WebFlux可以在资源有限情况下提高系统的吞吐量和伸缩性（不是提高性能），可以处理更多的请求（不是业务）。
- Spring WebFlux和Spring Web MVC可以混合使用。

5.1.2 Spring MVC框架

- 1.Spring MVC简介
- 2.Spring MVC工作流程

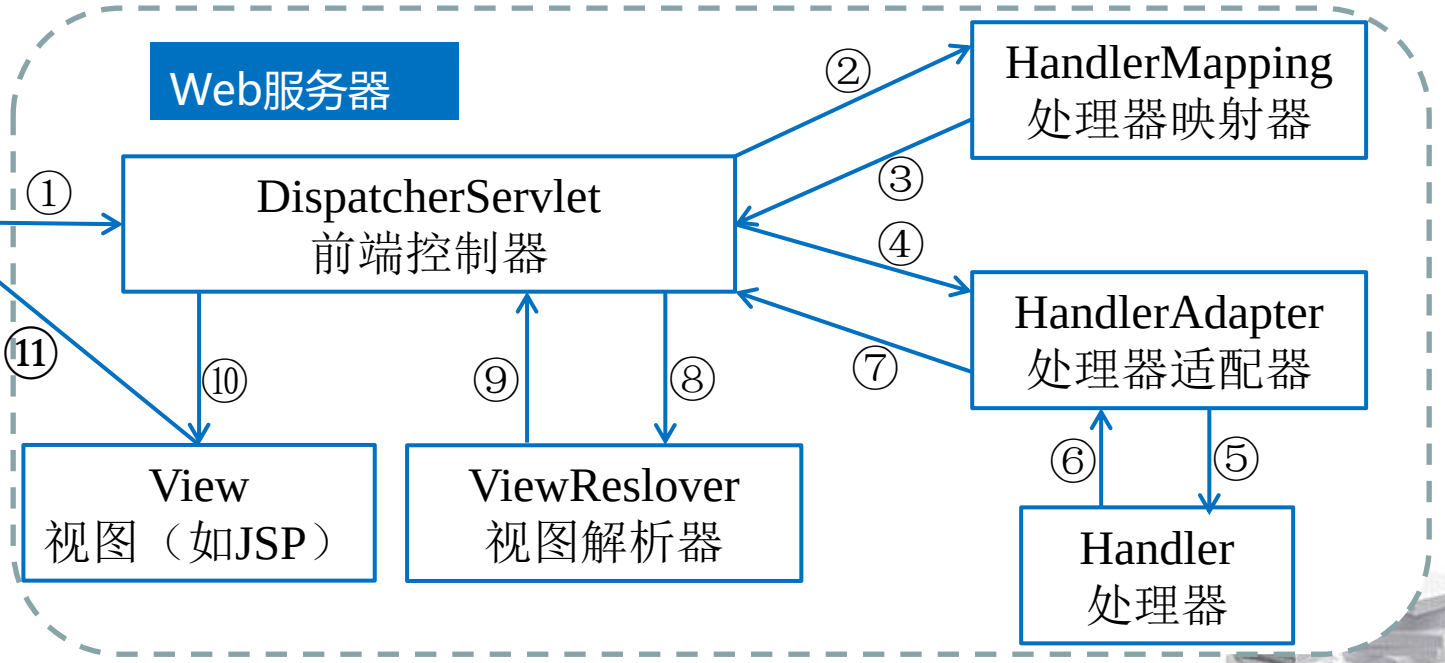
1.Spring MVC简介

- Spring MVC（ Spring Web MVC ）是Spring提供的一个实现了Web MVC设计模式的轻量级Web框架。
- 它与Struts2框架一样，都属于MVC框架，但其使用和性能等方面比Struts2更加优异。

Spring MVC特点

- **Spring MVC具有以下特点：**
  - ✧ 是Spring框架的一部分，可以方便的利用Spring所提供的其他功能。
  - ✧ 灵活性强，易于与其他框架集成。
  - ✧ 提供了一个前端控制器DispatcherServlet，使开发人员无需额外开发控制器对象。
  - ✧ 可自动绑定用户输入，并能正确的转换数据类型。
  - ✧ 内置了常见的校验器，可以校验用户输入。如果校验不能通过，那么就会重定向到输入表单。
  - ✧ 支持国际化。可以根据用户区域显示多国语言。
  - ✧ 支持多种视图技术。它支持Thymeleaf、JSP、Velocity和FreeMarker等视图技术。
  - ✧ 使用基于XML的配置文件，在编辑后，不需要重新编译应用程序。

2.Spring MVC工作流程



Spring MVC工作流程

- ①用户通过浏览器向服务器发送请求，请求会被Spring MVC的前端控制器DispatcherServlet所拦截;
- ②DispatcherServlet拦截到请求后，会调用HandlerMapping处理器映射器;
- ③处理器映射器根据请求URL找到具体的处理器，生成处理器对象及处理器拦截器（如果有则生成）一并返回给DispatcherServlet;
- ④DispatcherServlet会通过返回信息选择合适的HandlerAdapter（处理器适配器）;

Spring MVC工作流程

- ⑤HandlerAdapter会调用并执行Handler（处理器），这里的处理器指的就是程序中编写的Controller类，也被称之为后端控制器;
- ⑥Controller执行完成后，会返回一个ModelAndView对象，该对象中会包含视图名或包含模型和视图名;
- ⑦HandlerAdapter将ModelAndView对象返回给DispatcherServlet;
- ⑧DispatcherServlet会根据ModelAndView对象选择一个合适的ViewReslover（视图解析器）;

Spring MVC工作流程

- ⑨ViewReslover解析后，会向DispatcherServlet中返回具体的View（视图）；
- ⑩DispatcherServlet对View进行渲染（即将模型数据填充至视图中）；
- ⑪视图渲染结果会返回给客户端浏览器显示。

5.1.3 Spring MVC的整合支持

- 1.Spring MVC自动配置介绍
- 2.Spring MVC整合案例

1.Spring MVC自动配置介绍

- 在Spring Boot项目中，一旦引入了Web依赖启动器spring-boot-starter-web，那么Spring Boot整合Spring MVC框架默认实现的一些XxxAutoConfiguration自动配置类就会自动生效，几乎可以在无任何额外配置的情况下进行Web开发。

主要功能特性

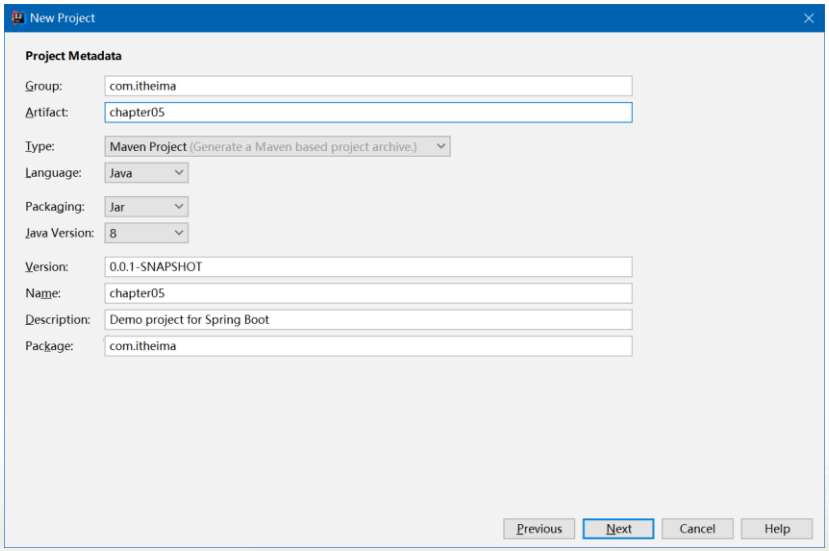
- 1）内置了两个视图解析器：ContentNegotiatingViewResolver和BeanNameViewResolver；
- 2）支持静态资源以及WebJars；
- 3）自动注册了转换器和格式化器；
- 4）支持Http消息转换器；
- 5）自动注册了消息代码解析器；
- 6）支持静态项目首页index.html；
- 7）支持定制应用图标favicon.ico；
- 8）自动初始化Web数据绑定器ConfigurableWebBindingInitializer。

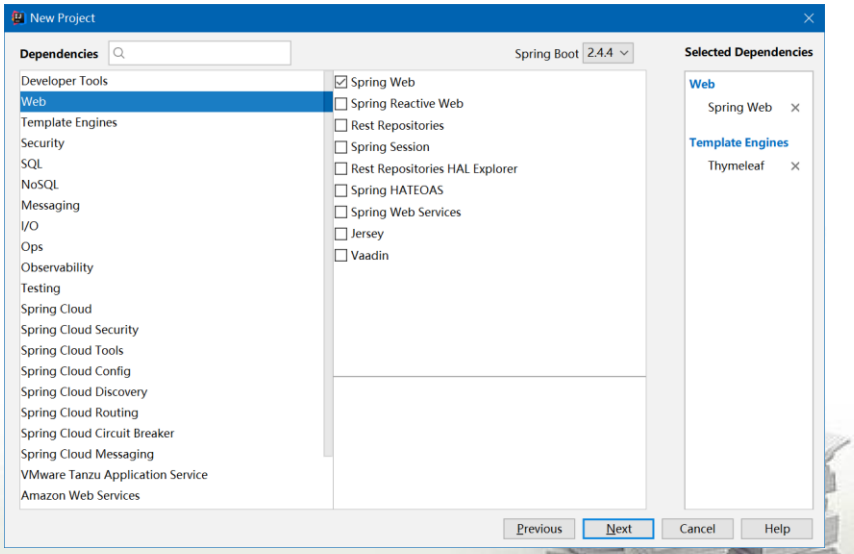
2. Spring MVC整合案例

- 本案例展示Spring Boot整合Spring MVC并利用Thymeleaf视图技术进行Web开发的步骤，实现Web静态资源的引入和动态数据的显示。
- 整合步骤：
  - ✧①创建Spring Boot项目
  - ✧②编写全局配置文件
  - ✧③创建Web控制类
  - ✧④创建模板页面
  - ✧⑤效果测试

①创建Spring Boot项目

- 使用Spring Initializr方式创建一个Spring Boot项目chapter05，在Dependencies依赖选择中选择Web模块中的Spring Web依赖和Template Engines模块中的Thymeleaf依赖





如果进行响应式WebFlux开发，选择Web模块中的Spring Reactive Web依赖。

②编写全局配置文件

- 为了开发中方便调试，打开全局配置文件，将**Thymeleaf**页面的数据缓存设置为**false**（默认为**true**），上线稳定后应保持默认**true**

```
spring.thymeleaf.cache=false
```

③创建Web控制类

- 在chapter05项目中新建一个com.itheima.controller包，并在包中新建一个Web控制类LoginController，实现向前端模板页面动态数据传递。

LoginController类

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import java.util.Calendar;
@Controller
public class LoginController {
    //获取并封装当前年份跳转到登录页login.html
    @GetMapping("/toLoginPage")
    public String toLoginPage(Model model){
        model.addAttribute("currentYear", Calendar.getInstance().get(Calendar.YEAR));
        return "login";
    }
  @ModelAttribute("institute")
    public String getInstitute(){
        return "广东财经大学";
    }
}
```

属性值只在此次请求login.html时有效

请求处理方法的另两种形式

■形式一：

```
@GetMapping("/toLoginPage")
public String toLoginPage(HttpServletRequest  request){
    request.setAttribute("currentYear", Calendar.getInstance().get(Calendar.YEAR));
    return "login";
}
```

属性值只在此次请求login.html时有效

■形式二：

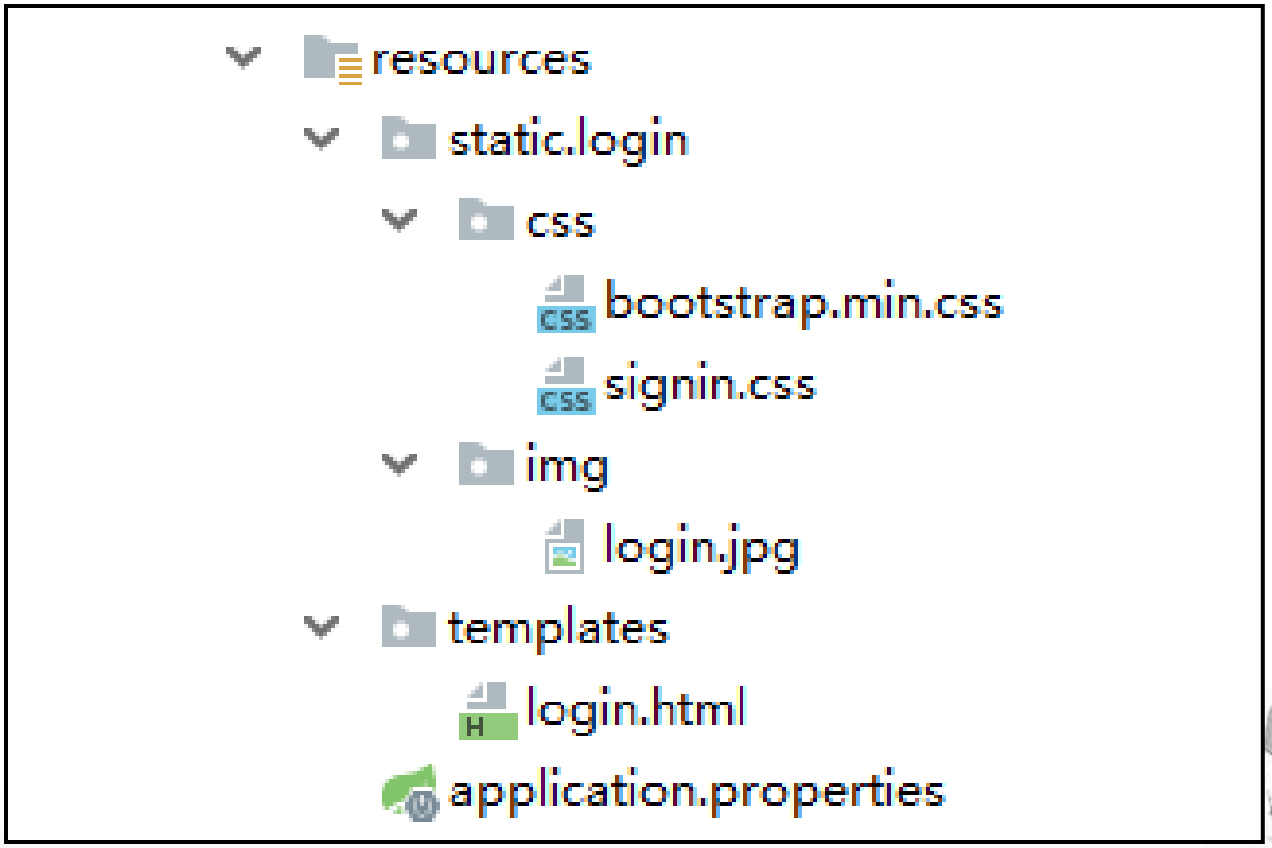
```
@GetMapping("/toLoginPage")
public String toLoginPage(HttpSession  sess){
    sess.setAttribute("currentYear", Calendar.getInstance().get(Calendar.YEAR));
    return "login";
}
```

属性值在整个会话都有效

④创建模板页面

- 在chapter05项目resources的templates目录下，创建一个用户登录的模板页面login.html，在其中引入后台传递过来的动态数据，同时引入CSS样式文件、图片文件等静态资源文件。

静态资源文件项目结构图



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">
```

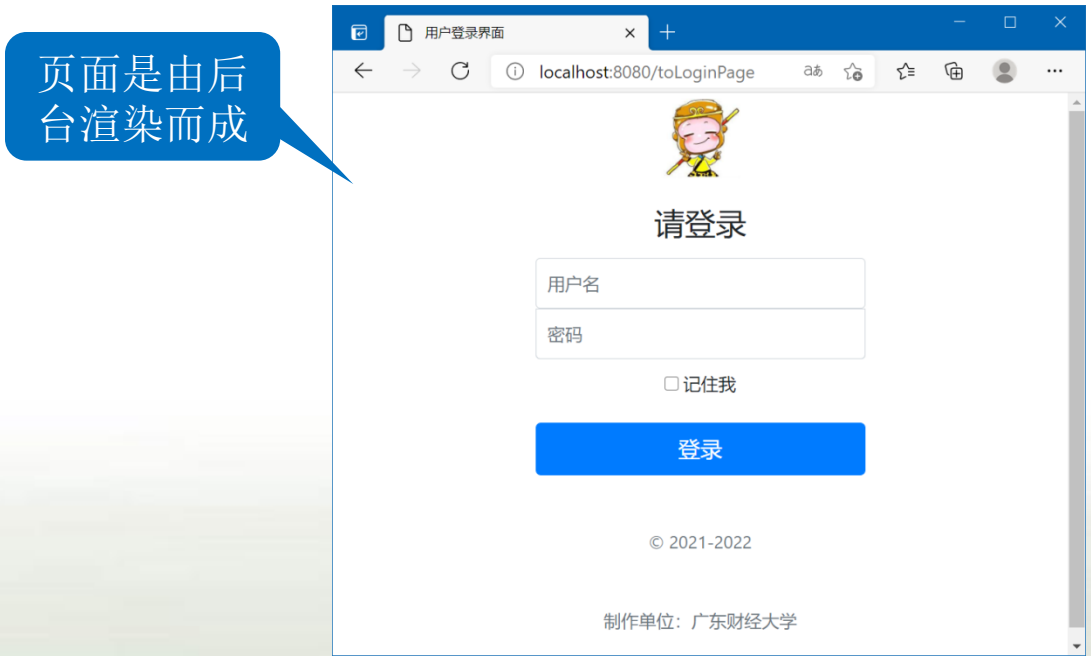
模板页面login.html

```
<!-- 用户登录form表单 -->
<form class="form-signin" th:action="@{/login}" method="post">
    <img class="mb-4" th:src="@{/login/img/login.jpg}" width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal" th:text="请登录">欢迎登录</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" th:text="登录">登录</button>
    <p class="mt-5 mb-3 text-muted">© <span>[[${currentYear}]]</span>-<span th:text="${currentYear}+1">2019</span></p>
    <p class="mt-5 mb-3 text-muted">制作单位：<span th:text="${institute}">清华大学</span></p>
</form>
</body>
</html>
```

或者基于session传递时使用
${session.currentYear}

⑤效果测试

- 启动项目进行测试，在浏览器上访问**http://localhost:8080/toLoginPage**，在页面底部动态显示了当前日期**2021-2022**和制作单位“广东财经大学”。



5.2 使用视图技术Thymeleaf

- 5.2.1 认识Thymeleaf
- 5.2.2 基础语法
- 5.2.3 数据分页
- 5.2.4 页面国际化

5.2.1 认识Thymeleaf

- Thymeleaf是一个Java类库，是一个xml/xhtml/html5的模板引擎（解析器），能够处理HTML、XML、JavaScript以及CSS，可以作为MVC Web应用的View层显示数据。
- Thymeleaf通过属性进行模板渲染，不需要引入不能被浏览器识别的新的标签，页面直接作为HTML文件，可以降低前后端人员的沟通成本。

引入依赖

```
<!-- Thymeleaf模板引擎启动器 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
 </dependency>
```

```
<!-- thymeleaf模板引擎整合security控制页面安全访问依赖 -->
 <dependency>
   <groupId>org.thymeleaf.extras</groupId>
   <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

Thymeleaf的配置

- 在全局配置文件application.properties中，可以配置Thymeleaf如下：

```
spring.thymeleaf.cache = false                  #为了便于测试，在开发时需要关闭缓存
spring.thymeleaf.encoding = UTF-8               #页面编码格式
spring.thymeleaf.mode = HTML5                   #语法检查模式
spring.thymeleaf.content-type = text/html       #页面文档类型
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
```

Spring Boot默认的页面映射路径（即模板文件存放的位置）为“classpath:/templates/*.html”，静态文件路径为“classpath:/static/”，其中可以存放CSS等模板共用的静态文件。

5.2.2 基础语法

- 1.引入命名空间
- 2.常用th属性和sec属性
- 3.标准表达式
- 4.运算符
- 5.条件判断
- 6.循环遍历
- 7.公共片段
- 8.内置对象
- 9.错误提示

1.引入命名空间

- 要使用**Thymeleaf**，需要先加入依赖，然后在模板文件中引入命名空间。

```
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">
……
</html>
```

2.常用th属性和sec属性

th属性和sec属性	作用
th:insert	页面片段包含（类似JSP中的include标签）
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，条件成立时显示th所在标签里的内容
th:unless	条件判断，条件不成立时显示th所在标签里的内容
th:switch	条件判断，进行选择性匹配
th:case	th:switch分支的条件判断
th:object	用于绑定对象
th:with	用于定义局部变量
th:onclick	用于指定点击事件
th:action	定义后台控制器路径
th:field	常用于表单参数绑定，通常与th:object一起使用
th:id	用于声明 的id

常用th属性和sec属性

th属性和sec属性	作用
th:attr	设置标签属性，多个属性可以用逗号分隔
th:attrprepend	通用属性修改，将计算结果追加前缀到现有属性值
th:attrappend	通用属性修改，将计算结果追加后缀到现有属性值
th:value	用于指定标签属性值
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
th:utext	用于指定标签显示的文本内容（支持html标签转义）
th:fragment	声明片段
th:remove	移除片段
th:style	用于指定标签的style
th:class	用于指定样式类
sec:authentication	用于安全认证
sec:authorize	用于安全认证

示例代码1

- 假如在国际化资源文件**messages_en_US.properties**中有以下属性和值：**test.myText=Test International Message**，那么在页面中可以使用如下两种方式获得属性值：
  - ✧** **
    - 不识别HTML标签，即输出**Test International Message**
  - ✧** **
    - 识别HTML标签，即输出加粗的“**Test International Message**”

示例代码2——首页面index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>影视直播厅</title>
</head>
<body>
<h1 align="center">欢迎进入电影网站首页</h1>
<!-匿名->
<div sec:authorize="isAnonymous()">
        <h2 align="center">游客您好，如果想查看电影<a th:href="@{/login}">请登录</a>或<a
th:href="@{/register}">请注册</a></h2>
</div>
```

示例代码2——首页面index.html

```
<!-已登录->
<div sec:authorize="isAuthenticated()">
            <h2 align="center">登录名：<span sec:authentication="name" style="color: #007bff"></span></h2>
   <h2 align="center">Principal：<span sec:authentication="principal" style="color:darkkhaki"></span></h2>
   <h2 align="center">权限：<span sec:authentication="principal.authorities" style="color:darkkhaki"></span></h2>
   <h2 align="center">Username：<span sec:authentication="principal.username" style="color: #007bff"></span></h2>
   <form action="/show" method="post">
      <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
      <input th:type="submit" th:value="显示" />
   </form>
   <form th:action="@{/logout}" method="post">
      <input th:type="submit" th:value="注销" />
   </form>
</div>
<hr>
```

示例代码2——首页面index.html

```
<div sec:authorize="hasRole('common')">
        <h3>普通电影</h3>
        <ul>
                <li><a th:href="@{/detail/common/1}">飞驰人生</a></li>
                <li><a th:href="@{/detail/common/2}">夏洛特烦恼</a></li>
        </ul>
</div>
<div sec:authorize="hasAuthority('ROLE_vip')">
        <h3>VIP专享</h3>
        <ul>
                <li><a th:href="@{/detail/vip/1}">速度与激情</a></li>
                <li><a th:href="@{/detail/vip/2}">猩球崛起</a></li>
        </ul>
</div>
</body>
</html>
```

3.标准表达式

- 1）变量表达式：${...}
  - 用于获取容器上下文环境中的变量值，示例代码如下：
    -
- 2）消息表达式：#{...}
  - 用于Thymeleaf模板页面国际化内容的动态替换和展示，示例代码如下：
    -
- 3）链接表达式：@{...}
  - 用于页面跳转或者资源的引入，表达式结构如下：
    - 无参：@{/xxx}
    - 有参：@{/xxx(k1=v1,k2=v2)}，对应url结构：xxx?k1=v1&k2=v2
  - 示例代码如下：
    -

标准表达式

- 4）片段表达式：~{...}
  - 用来将声明的片段移动或传递到模板，示例代码如下：
    -
- 5）选择变量表达式：*{...}
  - 用于从被选定对象（th:object属性绑定的对象）获取属性值，示例代码如下（firstName、lastName、nationality为user对象的属性）：
    -
      name: <span th: text="*{firstName}">
      surname: <span th: text="*{lastName}">
      nationality: <span th: text="*{nationality}">

    - <form th:action="@{/login}" th:object="${user}">
      <input type="text" value="" th:field="*{username}"></input>
      <input type="text" value="" th:field="*{role}"></input>
      </form>

4. 运算符

- 在Thymeleaf模板的表达式中可以使用+、-、*、/、%等各种算术运算符，也可以使用>（gt）、<（lt）、<=（le）、>=（ge）、==（eq）、!=（ne）等各种逻辑运算符。示例代码如下：
  - ✧<tr th:class="(${row}== 'even')? 'even' : 'odd'">...</tr>
  - ✧

5.条件判断

- 1）if和unless
  - 标签只有在th:if条件成立时才显示，th:unless与th:if相反，只有条件不成立时，才显示标签内容。示例代码如下：
    - 成功
    - 成功
- 2）switch语句
  - Thymeleaf模板也支持多路选择switch语句结构，默认属性default可用“*”表示。示例代码如下：
    -
      User is an administrator
      User is a teacher
      User is a student

6.循环遍历

- **Thymeleaf模板使用属性 th:each="obj,iterStat:${objList}" 进行迭代循环，迭代对象可以是java.util.List、java.util.Map、数组等。**

示例代码

```
<!-- 循环取出数组数据，th:each所在div标签重复出现-->
<div class="col-md-4 col-sm-6" th:each="book:${books}">
        <a href="">
            <img th:src="'images/' + ${book.picture}" alt="图书封面" style="height: 180px; width: 40%;"/>
        </a>
        <div class="caption">
            <h4 th:text="${book.bname}"></h4>
            <p th:text="${book.author}"></p>
            <p th:text="${book.isbn}"></p>
            <p th:text="${book.price}"></p>
            <p th:text="${book.publishing}"></p>
        </div>
</div>
```

循环状态的使用

- 在th:each属性中可以使用循环状态变量，该变量有如下属性：
  - ✧index：当前迭代对象的index（从0开始计数）。
  - ✧count：当前迭代对象的index（从1开始计数）。
  - ✧size：迭代对象的大小。
  - ✧current：当前迭代变量。
  - ✧even/odd：布尔值，当前循环是否是偶数/奇数（从0开始计数）。
  - ✧first：布尔值，当前循环是否是第一个。
  - ✧last：布尔值，当前循环是否是最后一个。

示例代码

```
<!-- 循环取出数组数据， th:each所在div标签重复出现-->
<div class="col-md-4 col-sm-6" th:each="book,bookStat:${books}">
        <a href="">
            <img th:src="'images/' + ${book.picture}" alt="图书封面" style="height: 180px; width: 40%;"/>
        </a>
        <div class="caption">
            <!--循环状态bookStat-->
            <h3 th:text="${bookStat.count}"></h3>
            <h4 th:text="${book.bname}"></h4>
            <p th:text="${book.author}"></p>
            <p th:text="${book.isbn}"></p>
            <p th:text="${book.price}"></p>
            <p th:text="${book.publishing}"></p>
        </div>
</div>
```

7.公共片段

- **Thymeleaf通过在模板文件（ 如：templatename.html）中用th:fragment声明一个公共片段，即：th:fragment="fragmentname"**

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<div class=“header”  th:fragment=“header(变量名)">
    公共 header，[[${变量名}]]
</div>
<div class="footer"  th:fragment="footer">
    公共 footer
</div>
</body>
</html>
```

片段表达式

- 片段表达式支持两种语法结构：
  - ~{templatename::fragmentname}（推荐）
    - templatename：模版名，Thymeleaf会根据模版名解析完整路径：/resources/templates/templatename.html，要注意文件的路径。
    - fragmentname：片段名
  - ~{templatename::#id}
    - id：HTML的id选择器，使用时要在前面加上#号，不支持class选择器。

调用公共片段

- 调用公共片段可用以下th属性实现：
  - ✧th:insert：将公共片段整个插入到使用了th:insert的HTML标签中。示例代码如下：
    -
  - ✧th:replace：将公共片段整个替换使用了th:replace的HTML标签中。示例代码如下：
    -
  - ✧th:include：将公共片段包含的内容插入到使用了th:include的HTML标签中。示例代码如下：
    -

调用效果比较

th:insert

```
<div>
    <div class="footer">
        公共 footer
    </div>
</div>
```

th:replace

```
<div class="footer">
    公共 footer
</div>
```

th:include

```
<div>
    公共 footer
</div>
```

8.内置对象

- 和JSP一样，Thymeleaf也提供了一系列内置对象，可以通过“#”直接访问。
- 常见的内置对象：

内置对象	说明
#ctx	上下文对象
#locale	通过该对象可获取上下文的国际区域
#request	HttpServletRequest对象
#response	HttpServletResponse对象
#session	HttpSession对象
#servletContext	ServletContext对象
#vars	通过该对象可获取上下文变量
#messages	通过该对象可获取指定的消息文本
#fields	为表单和表单元素提供工具方法

内置对象

内置对象	说明
#dates	为 java.util.Date对象提供工具方法
#calendars	为java.util.Calendar对象提供工具方法
#numbers	为数值型对象提供工具方法
#strings	为String对象提供工具方法
#objects	为Object对象提供常用的工具方法
#bools	为Boolean对象提供常用的工具方法
#arrays	为对象型数组提供常用的工具方法
#lists	为List对象提供常用的工具方法
#sets	为Set对象提供常用的工具方法
#maps	为Map对象提供常用的工具方法
#aggregates	为对象型数组提供元素统计方法
#uris	用于在Thymeleaf标准表达式中执行URI / URL操作
#conversions	用于在模板的任意位置执行转换服务
#ids	为可能需要循环的ID属性提供常用的工具方法

示例代码

- 访问内置对象：
  - [[${#request.locale}]]、[[${#locale.country}]] 、
    [[${#vars.currentYear}]]、
    [[${#messages.msg('msgKey')}]]、
    [[${#dates.format(nowDate,'yyyy/MM/dd')}]]、
    [[${#dates.create(2021,4,2)}]]、
    [[${#calendars.format(nowCalendar,'yyyy/MM/dd')}]]、
    [[${#numbers.formatInteger(myBigDecimal,整数位数)}]]、
    [[${#numbers.sequence(0, 9)}]]、
    [[${#objects.nullSafe(myBigDecimal,默认值)}]]、
    [[${#bools.isFalse(myBoolean)}]]、
    [[${#arrays.contains(myIntegerArray,5)}]]、
    [[${#lists.sort(myList)}]]、
    [[${#sets.contains(mySet,'e1')}]]、
    [[${#maps.containsKey(myMap,'key1')}]]、
    [[${#aggregates.sum(myIntegerArray)}]]、
    [[${#aggregates.avg(myIntegerArray)}]]、
    [[${#fields.hasErrors('username')}]]

示例代码

- 访问spring容器中的对象：
  - [[${@userService.getUsername()}]]
- 访问Web上下文中的对象：
  - [[${#ctx.session.currentYear}]] 或 [[${#vars.session.currentYear}]] 或 [[${session.currentYear}]]
  - [[${#ctx.application.size()}]] 或 [[${application.size()}]]
  - [[${#ctx.param.size()}]] 或 [[${param.size()}]]

9.错误提示

- Thymeleaf提供了以下属性和方法来实现错误信息反馈：
  - th:field：用于表单参数绑定
  - th:errors：用于显示错误信息
  - th:errorclass：用于定义错误出现后的CSS样式
  - #fields.hasErrors(“*”)：判断是否有错误
  - #fields.errors(“*”)：获取错误信息
- 须结合后台Spring Boot内置的验证器（Validator）一起使用。

示例代码

```
<form class="form-signin" th:action="@{/checkUser}" th:object="${user}" method="post">
    <h1 class="h3 mb-3 font-weight-normal" th:text="请注册">欢迎注册</h1>
    <input type="text" th:field="*{username}" class="form-control" th:errorclass="warn“ th:placeholder="用户名"
required="" autofocus="">
    <span  class="warn" th:if="${#fields.hasErrors('username')}" th:errors="*{username}">用户名错误</span>
    <input type="password"  th:field="*{password}" class="form-control" th:placeholder="密码"  required="">
    <span  class="warn" th:if="${#fields.hasErrors('password')}" th:errors="*{password}">密码错误</span>
    <input type="password" th:field="*{matchingpwd}" class="form-control" th:placeholder="确认密码"  required="">
    <span  class="warn" th:if="${#fields.hasErrors('${user}')}" th:errors="${user}">确认密码错误</span>
    <input type="text" th:field="*{phone}" class="form-control" th:errorclass="warn"  th:placeholder="手机号码"
required="">
    <span  class="warn" th:if="${#fields.hasErrors('phone')}" th:errors="*{phone}">手机号码错误</span>
    <ul th:if="${#fields.hasErrors('*')}">
        <li th:each="err:${#fields.errors('*')}" th:text="${err}">输入错误</li>
    </ul>
    <button class="btn btn-lg btn-primary btn-block" type="submit" th:text="注册">注册</button>
</form>
```

5.2.3 数据分页

- 在MVC开发过程中，分页是常用的功能，Thymeleaf可以处理由控制器传入的Page<T>参数。
- 可选的分页工具：
  - com.baomidou.mybatisplus.extension.plugins.pagination.Page：须与MyBatis-Plus结合使用
  - spring-boot-starter-data-redis或-mongodb或-jpa内置分页工具：可单独使用

数据分页案例

- 本案例基于Spring Boot内置分页工具，以Thymeleaf为视图技术实现数据分页。
- 搭建步骤：
  - ✧①引入依赖启动器
  - ✧②创建持久化类
  - ✧③创建业务层类
  - ✧④创建Web控制类
  - ✧⑤创建分页页面
  - ✧⑥效果测试

①引入启动器依赖

- 在chapter05项目的pom.xml中引入Spring Data Redis依赖启动器，提供对数据集分页的支持。

```
<!-- Spring Data Redis依赖启动器 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

②创建持久化类

- 在**chapter05**项目中新建一个**com.itheima.domain**包，并在包中新建一个实体类**User**。

```
public class User {
    private Integer id;
    private String username;
    private String password;
    private String matchingpwd;
    private String phone;
    //省略属性的getXX()和setXX()方法
}
```

③创建业务层类

- 在chapter05项目中新建一个com.itheima.service包，并在包中新建一个业务类UserService，实现对User数据集的分页和排序处理。

UserService类

```
import com.itheima.domain.User;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
@Service
public class UserService {
    private int getTotalElements(){
        return 23;
    }
    // pageNum当前页号，pageSize每页显示的记录数
    public Page<User> getUserPage(int pageNum, int pageSize) {
        Sort.Order order=new Sort.Order(Sort.Direction.DESC, "id");
        Pageable pageable = PageRequest.of(pageNum, pageSize, Sort.by(order));
        int totalElements=getTotalElements();          //获取数据集总记录数
        List<User> userList=new ArrayList<>();
        int fromIndex = pageNum*pageSize;
        int toIndex = (pageNum+1)*pageSize;
        if (toIndex>totalElements) toIndex = totalElements;
```

UserService类

```
    for (int i=fromIndex;i<toIndex;i++){            //生成当前页记录
        User user = new User();
        user.setId(i);
        user.setUsername("UserName"+i);
        user.setPhone("Phone"+i);
        userList.add(user);
    }
    userList.sort(new Comparator<User>() {      //对当前页记录排序
        @Override
        public int compare(User o1, User o2) {
            if(pageable.getSort().getOrderFor("id").isAscending())
                return o1.getId().compareTo(o2.getId());
            else
                return o2.getId().compareTo(o1.getId());
        }
    });
    return new PageImpl<>(userList, pageable, totalElements);       //当前页分页配置
  }
}
```

④创建Web控制类

- 在chapter05项目的com.itheima.controller包中新建一个Web控制类PagingController，实现向前端模板页面传递page对象。

PagingController类

```
import com.itheima.domain.User;
import com.itheima.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
@Controller
public class PagingController {
    @Autowired
    private UserService userService;
    //跳转到分页模板页pages.html
    @GetMapping("/toPaging")
    public String toPaging(Model model,
                           @RequestParam(value = "pageNum", defaultValue = "0") int pageNum,
                           @RequestParam(value = "pageSize", defaultValue = "5") int pageSize){
        Page<User> users=userService.getUserPage(pageNum,pageSize);
        model.addAttribute("users",users);
        return "pages";
    }
}
```

⑤创建分页页面

- 在chapter05项目resources的templates目录下，创建一个分页模板页面pages.html，在其中接收page对象并处理。
- page对象相关属性：
  - ✧totalElements：数据集总记录数
  - ✧totalPages：数据集总页数
  - ✧number：当前页页号（从0开始）
  - ✧first：当前页是否首页
  - ✧last：当前页是否尾页

模板页面pages.html

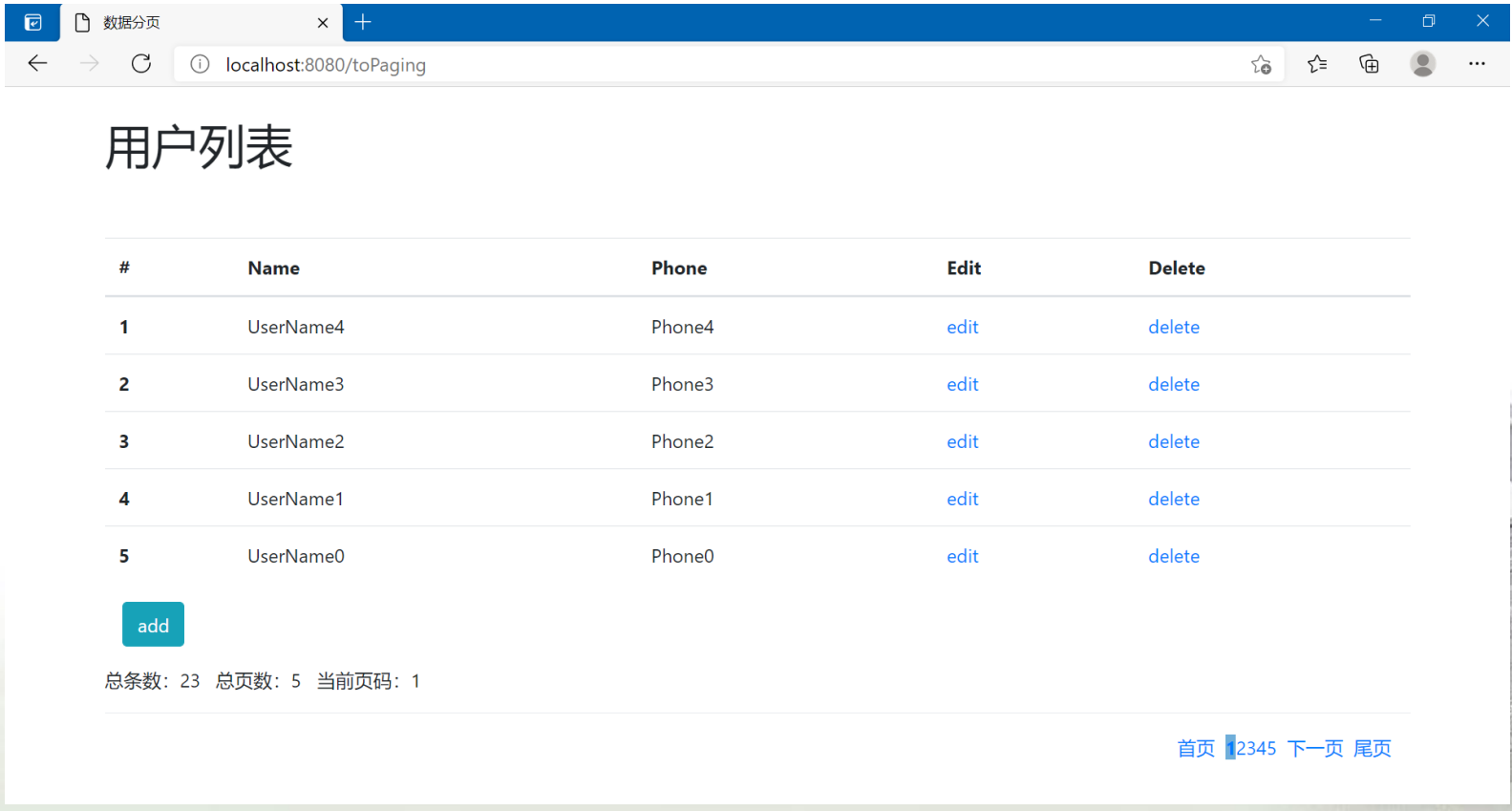
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>数据分页</title>
    <link rel="stylesheet"
th:href="@{/login/css/bootstrap.min.css}">
</head>
<body class="container">
<br/>
<h1>用户列表</h1>
<br/><br/>
<div class="with:80%">
    <table class="table table-hover">
        <thead>
        <tr>
            <th>#</th>
            <th>Name</th>
            <th>Phone</th>
            <th>Edit</th>
            <th>Delete</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="user : ${users}">
            <th scope="row" th:text="${userStat.index + 1}">1</th>
            <td th:text="${user.username}"></td>
            <td th:text="${user.phone}"></td>
            <td><a onclick="return confirm('确定修改吗？')" th:href="@{/toEdit(id=${user.id})}">edit</a></td>
            <td><a onclick="return confirm('确定删除吗？')" th:href="@{/delete(id=${user.id})}">delete</a></td>
        </tr>
        </tbody>
    </table>
</div>
<div class="form-group">
    <div class="col-sm-2 control-label">
        <a th:href="@{/toAdd}" class="btn btn-info">add</a>
    </div>
</div>
<div class="form-group">
    总条数：<span th:text="${users.totalElements}"></span>  
    总页数：<span th:text="${users.totalPages}"></span>  
    当前页码：<span th:text="${users.number}+1"></span>
</div>
```

模板页面pages.html

```
<div class="modal-footer no-margin-top">
   <ul class="pagination pull-right no-margin">
      <li>
         <a th:href="@{/toPaging}">首页</a>  
      </li>
      <li th:if="${not users.first}">
         <a th:href="@{/toPaging(pageNum=${users.number - 1})}" th:text="上一页"></a>  
      </li>
      <!-- 中间页 -->
      <li th:each="n:${#numbers.sequence(0, users.totalPages - 1)}">
         <a th:href="@{/toPaging(pageNum=${n})}" th:text="${n + 1}"  th:if="${n ne users.number}"></a>
         <a th:href="@{/toPaging(pageNum=${n})}" th:text="${n + 1}“  th:if="${n eq users.number}"
            th:style="'font-weight:bold;background: #6faed9;'"></a>
      </li>  
      <li th:if="${not users.last}">
         <a th:href="@{/toPaging(pageNum=${users.number + 1})}" th:text="下一页"></a>  
      </li>
      <li>
         <a th:href="@{/toPaging(pageNum=${users.totalPages - 1})}">尾页</a>
      </li>
   </ul>
</div>
</body>
</html>
```

⑥效果测试

- 启动项目进行测试，在浏览器上访问 **http://localhost:8080/toPaging**



5.2.4 页面国际化

- 通过设置多语言属性配置文件和定制区域化解析器，应用Themeleaf视图技术很容易实现页面的国际化。
- 实现步骤：
  - ✧①编写多语言属性配置文件
  - ✧②定制区域化解析器
  - ✧③创建Web控制类
  - ✧④创建国际化页面
  - ✧⑤效果测试

①编写多语言属性配置文件

- 在**chapter05**项目**resources**下创建一个用于统一管理多语言配置文件的名称为**i18n**的文件夹，在该文件夹中根据需要编写对应多语言的属性配置文件，并在全局配置文件**application.properties**中添加多语言属性配置文件基础名。
  - ✧多语言属性配置文件严格按以下格式命名：
    - 基础名_语言代码_国家代码**.properties**
    - 项目默认的语言属性配置文件为：基础名**.properties**

多语言属性配置文件

```
#配置多语言属性配置文件基础名
spring.messages.basename=i18n.login
```

login.properties、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
```

②定制区域化解析器

- 在chapter05项目中新建一个com.itheima.config包，并在包中新建一个自定义配置类LocaleConfig，用于对区域化解析器和拦截器的定制，应用程序会根据前端选择的语言（反映在请求头中的Accept-Language信息上）自动进行语言切换。

LocaleConfig配置类

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;
import org.springframework.web.servlet.i18n.SessionLocaleResolver;
import java.util.Locale;
@Configuration
public class LocaleConfig implements WebMvcConfigurer {
    @Bean
    // 根据用户本次会话过程中的语义设定语言区域（如用户进入首页时选择的语言种类）
    public LocaleResolver localeResolver() {
        SessionLocaleResolver slr = new SessionLocaleResolver();
        slr.setDefaultLocale(Locale.US);  //设置默认语言区域
        return slr;
    }
```

LocaleConfig配置类

```
    @Bean
    // 使用SessionLocaleResolver存储语言区域时，须定制LocaleChangeInterceptor拦截器
    public LocaleChangeInterceptor localeChangeInterceptor() {
        LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
        lci.setParamName("loc");    //设置选择语言的参数名
        return lci;
    }
    @Override
    // 注册拦截器，拦截器会在业务处理器（即Web控制器中URL映射处理方法）执行前执行
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(localeChangeInterceptor());
    }
}
```

③创建Web控制类

- 在chapter05项目的com.itheima.controller包中修改Web控制类LoginController，实现向前端模板页面及其公共片段动态数据传递。

LoginController类

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import java.util.Calendar;
@Controller
public class LoginController {
    //跳转到登录页login.html
    @GetMapping("/toLoginPage")
    public String toLoginPage(){
        return "login";
    }
    @ModelAttribute("currentYear")
    public int getCurrentYear(){
        return Calendar.getInstance().get(Calendar.YEAR);
    }
    @ModelAttribute("institute")
    public String getInstitute(){
        return "广东财经大学";
    }
}
```

目标页的公共变量

目标页的公共变量

④创建国际化页面

- 在chapter05项目resources的templates目录下，修改login.html视图页面，在页面中使用消息表达式#{...}获得国际化信息，并使用公共片段设计footer。

国际化页面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" th:action="@{/toPaging}" method="GET">
    <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}">欢迎登录</h1>
    <input type="text" class="form-control“ th:placeholder="#{login.username}" required="" autofocus="">
    <input type="password" class="form-control" th:placeholder="#{login.password}" required="">
```

国际化页面login.html

```
    <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>
    <div>
        <a class="btn mt-5 btn-sm" th:href="@{/toLoginPage(loc='zh_CN')}">中文</a>
        <a class="btn mt-5 btn-sm" th:href="@{/toLoginPage(loc='en_US')}">English</a>
    </div>
    <div th:replace="~{footer::footer(${currentYear},${institute})}"></div>
</form>
</body>
</html>
```

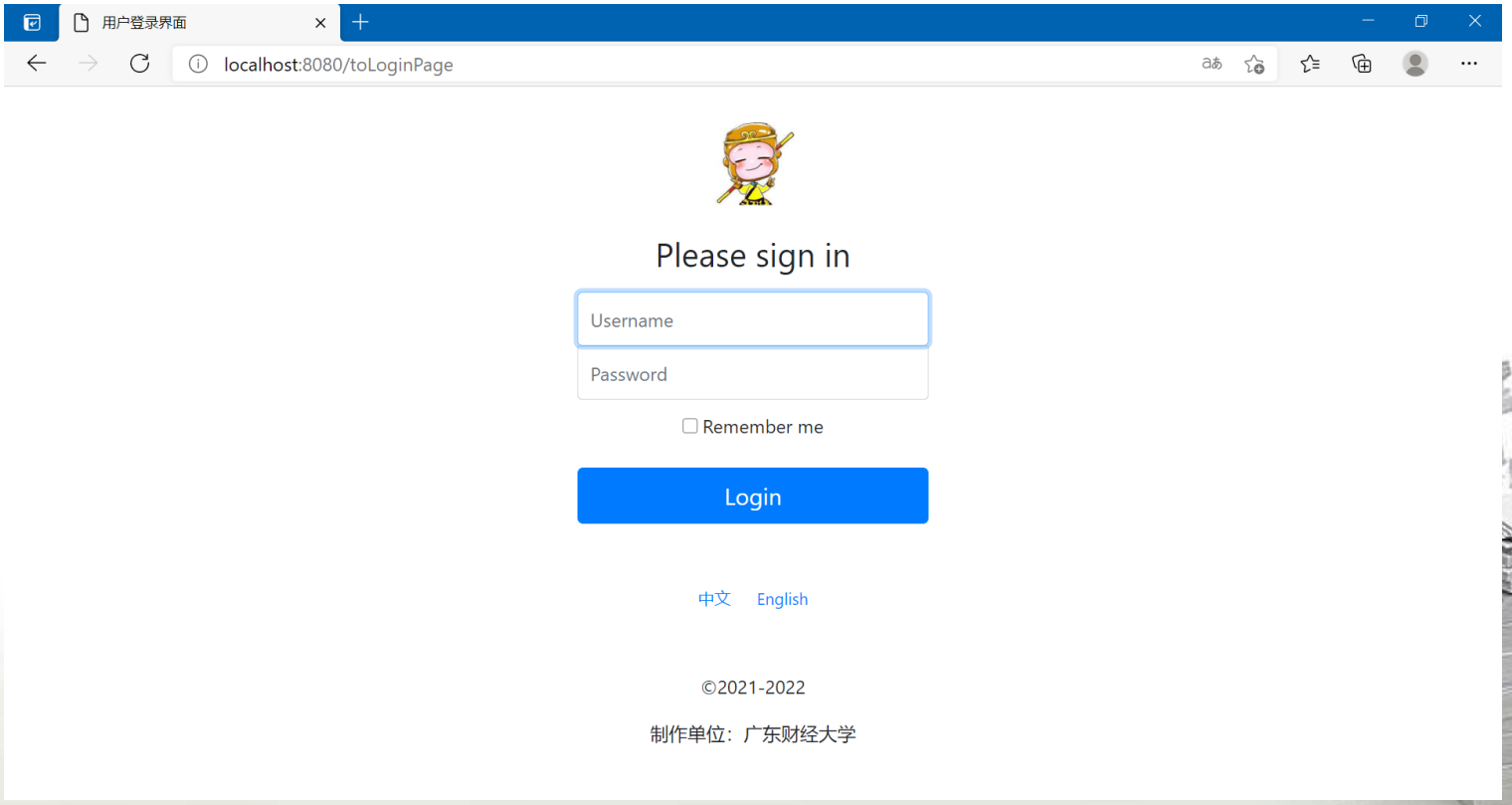
公共片段footer.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">


<div class="footer mt-5 mb-3"  th:fragment="footer(currentYear,institute)">
    <p>©<span>[[${currentYear}]]</span>-<span th:text="${currentYear}+1">2019</span></p>
    <p>制作单位：<span th:text="${institute}">清华大学</span></p>
</div>
</body>
</html>
```

⑤效果测试

- 启动项目进行测试，在浏览器上访问 **http://localhost:8080/toLoginPage**




聰明等于勤奋
勞力就是幸福