# SpringBoot整合

**《Spring Boot 2.x》**

**SpringDataJpa 与ThymeLeaf**

## 课程主要内容

* SpringDataJpa进阶使用
* SpringDataJpa自定义查询
* 整合Servlet、Filter、Listener
* 文件上传
* Thymeleaf常用标签

## 整合Servlet

### 注解方式

#### 启动类上添加注解

@SpringBootApplication

@ServletComponentScan

**public** **class** Springboot011Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Springboot011Application.**class**, args);

}

}

#### Servlet类

@WebServlet(name = "myServlet",urlPatterns = "/srv",loadOnStartup = 1)

**public** **class** MyServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

System.***out***.println("111");

**super**.doGet(req, resp);

}

}

### 编码方式

#### 启动类中添加

@Bean

**public** ServletRegistrationBean<MyServlet2> getServletRegistrationBean(){

ServletRegistrationBean<MyServlet2> bean = **new** ServletRegistrationBean<>(**new** MyServlet2(), "/s2");

bean.setLoadOnStartup(1);

**return** bean;

}

#### Servlet

无需注解

## 整合Filter

**implements** Filter

过滤器

同Servlet

## Listener

监听器

MyListener **implements** ServletContextListener

同上

# 静态资源访问

src/main/resources/static

src/main/webapp

# SpringBoot文件上传

## 指定大小

spring.http.multipart.maxFileSize=200MB

spring.http.multipart.maxRequestSize=200MB

spring.servlet.multipart.max-request-size = 200MB

spring.servlet.multipart.max-file-size = 200MB

## 表单

<form action="fileUploadController" method="post" enctype="multipart/form-data">

上传文件：<input type="file" name="filename"/><br/>

<input type="submit"/>

</form>

## Controller

@RequestMapping("/fileUploadController")

**public** String fileUpload(MultipartFile filename) **throws** Exception{

System.***out***.println(filename.getOriginalFilename());

filename.transferTo(**new** File("e:/"+filename.getOriginalFilename()));

**return** "ok";

}

# 表单数据接收

@GetMapping(value = "/hello/{id}")

public String hello(@PathVariable("id") Integer id){

return "ID:" + id;

}

@PostMapping(value = "/user-2")

public User saveUser2(@RequestBody User user) {

return user;

}

@PostMapping(value = "/post")

public String post(@RequestParam(name = "name") String name,

@RequestParam(name = "age") Integer age) {

String content = String.format("name = %s,age = %d", name, age);

return content;

}

# SpringData Jpa进阶使用

## 显示SQL

spring.jpa.show-sql=true

## 自定义查询

### 自定义接口

* And --- 等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；
* Or --- 等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；
* Between --- 等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；
* LessThan --- 等价于 SQL 中的 "<"，比如 findBySalaryLessThan(int max)；
* GreaterThan --- 等价于 SQL 中的">"，比如 findBySalaryGreaterThan(int min)；
* IsNull --- 等价于 SQL 中的 "is null"，比如 findByUsernameIsNull()；
* IsNotNull --- 等价于 SQL 中的 "is not null"，比如 findByUsernameIsNotNull()；
* NotNull --- 与 IsNotNull 等价；
* Like --- 等价于 SQL 中的 "like"，比如 findByUsernameLike(String user)；
* NotLike --- 等价于 SQL 中的 "not like"，比如 findByUsernameNotLike(String user)；
* OrderBy --- 等价于 SQL 中的 "order by"，比如 findByUsernameOrderBySalaryAsc(String user)；
* Not --- 等价于 SQL 中的 "！ ="，比如 findByUsernameNot(String user)；
* In --- 等价于 SQL 中的 "in"，比如 findByUsernameIn(Collection<String> userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；
* NotIn --- 等价于 SQL 中的 "not in"，比如 findByUsernameNotIn(Collection<String> userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

### 自定义SQL@Query

#### 占位符

public interface UserDao extends Repository<AccountInfo, Long> {

@Query("select a from AccountInfo a where a.accountId = ?1")

public AccountInfo findByAccountId(Long accountId);

   @Query("select a from AccountInfo a where a.balance > ?1")

public Page<AccountInfo> findByBalanceGreaterThan(

Integer balance,Pageable pageable);

}

#### 参数名

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

@Query("from AccountInfo a where a.accountId = :id")

public AccountInfo findByAccountId(@Param("id")Long accountId);

  @Query("from AccountInfo a where a.balance > :balance")

  public Page<AccountInfo> findByBalanceGreaterThan(

@Param("balance")Integer balance,Pageable pageable);

}

#### 更新

@Modifying

@Query("update AccountInfo a set a.salary = ?1 where a.salary < ?2")

public int increaseSalary(int after, int before);

### 直接使用Native SQL

public interface UserRepository extends JpaRepository<User, Long> {

@Query(value = "SELECT \* FROM USERS WHERE EMAIL\_ADDRESS = ?1", nativeQuery = true)

User findByEmailAddress(String emailAddress);

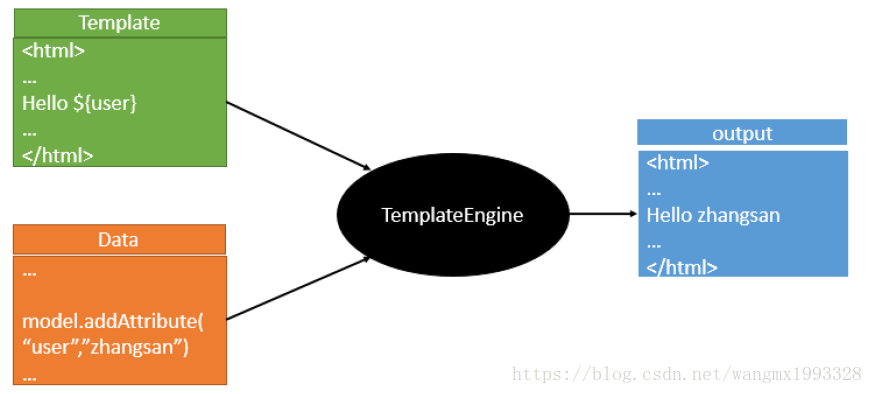
}

# Thymeleaf

## 编辑插件

安装地址：

<html xmlns:th=*"http://www.thymeleaf.org"*>



## URL地址

在应用上下文内

1）@{userList} 相对当前路径结果为：http://localhost/thymeleaf/user/userList

2）@{./userList} 相对当前路径结果为：http://localhost/thymeleaf/user/userList

3）@{../tiger/home} 相对当前路径结果为：http://localhost/thymeleaf/tiger/home

4）@{/tiger/home} 相对应用根目录结果为：http://localhost/thymeleaf/tiger/home

5）@{https://www.baidu.com/} 绝对路径结果为：https://www.baidu.com

6）<link type="text/css" rel="stylesheet" th:href="@{/css/home.css}">，@ 以 "/" 开头相对应用根目录，否则是相对当

## Href

<body>

<a th:href="@{userList(id=9527)}">1、@{userList(id=9527)}</a>

<a th:href="@{userList(id=9527,name=华安)}">2、@{userList(id=9527,name=华安)}</a>

<a th:href="@{userList(id=9527,name=${userName})}">3、@{userList(id=9527,name=${userName})}</a>

</body>

## 文本

<!--空格属于特殊字符，必须使用单引号包含整个字符串-->

<p class=*"css1 css2"* th:class=*"'css1 css2'"*>样式</p>

<!--下面如果没用单引号 th:text="Big China"，则页面直接报错-->

<p th:text=*"'Big China'"*>中国</p>

<!--后台使用：model.addAttribute("info", "Love you 中国"); 传值有空格也是没有问题的-->

<p th:text=*"${userName}"*>userName</p>

<!--后台传值字符串有空格是可以的，可以使用 + 进行字符串连接-->

<p th:text=*"'small smile'+',very good.' + ${userName}"*>浅浅的微笑</p>

## 数字计算

<p th:text=*"80"*>8</p>

<!--计算结果为 16 在进行替换-->

<p th:text=*"8+8"*>8 + 8</p>

<!--前面 8+8 计算结果为 16，然后字符串拼接上 Love，后面的 9+9也会被当做字符串拼接-->

<p th:text=*"8+8+' Love '+9+9"*>8 + 8+' Love '+9+9</p>

<!--前面 8+8 计算结果为 16，后面的 9+9因为有括号,所以也会计算结果，最后拼接 Love 字符串-->

<p th:text=*"8+8+' Love '+(9+9)"*>8 + 8+' Love '+(9+9)</p>

<!--后台传了一个：model.addAttribute("age", 35);取得结果后在进行计算-->

<p th:text=*"100-${age}"*></p>

## Boolean判断

<p th:text=*"true"*>布尔</p>

<!--true、false 是布尔值，and 是布尔运行符，and(与)，or(或)，not(非)、!(非)-->

<p th:text=*"true and false"*>true and true</p>

<!--后台使用 model.addAttribute("isMarry", true); 传了值-->

<!--th:if 表达式为 true，则显示标签内容，否则不显示-->

<p th:if=*"${isMarry}"*>已结婚</p>

<!--后台传值：model.addAttribute("age", 35);-->

<!--比较运算符：&gt;，&lt;，&gt; =，&lt;=（gt，lt，ge，le）-->

<p th:if=*"${age}&gt;18"*>已成年</p>

<p th:if=*"${age}&lt;18"*>未成年</p>

## 运算

<p th:text=*"15 \* 4"*>值为 60 </p>

<p th:text=*"15 \* 4-100/10"*>值为 50 </p>

<p th:text=*"100 % 8"*>值为 4</p>

## 比较

<p th:if=*"5>3"*>5 大于 3</p>

<p th:if=*"5 &gt;4"*>5 大于 4</p>

<p th:if=*"10>=8 and 7 !=8"*>10大于等于8，且 7 不等于 8 </p>

<p th:if=*"!${isMarry}"*>!false</p>

<p th:if=*"not(${isMarry})"*>not(false)</p>

## 三元运算符

<p th:text=*"7&gt;5?'7大':'5大'"*>三元运算符</p>

<!--后台控制器输出了：model.addAttribute("age", 35);-->

<!--因为 ${xx}取值时，如果值为null，则默认整个标签不再显示-->

<p th:text=*"${age}!=null?${age}:'age等于 null'"*></p>

<!--这里使用嵌套判断，嵌套的部分要使用括号-->

<p th:text=*"${age}!=null?(${age}>=18?'成年':'未成年'):'age等于 null'"*></p>

<!--变量 age2 后台并没有输出，所以 age2 不存在，此时 age2 ==null-->

<p th:text=*"${age2}!=null?${age2}:'age2等于 null'"*></p>

<!--后台输出了：model.addAttribute("isMarry", true);-->

<!--A>B?X:Y，这里的 Y 部分是省略的，此时如果 A>B 返回 false，则整个三元运算结果为 null-->

<p th:class=*"${isMarry}?'css2':'css3'"*>已婚</p>

## TH标签

### th:utext转义

map .addAttribute("china", "<b>Chian</b>,USA,UK");

<p th:text=*"${china}"*>默认转义</p>

<!--th:utext 不会结果进行转义-->

<p th:utext=*"${china}"*>不会转义</p>

### th:attr 设置属性

HTML5 所有的属性，都可以使用 th:\* 的形式进行设置值

<a href="http://baidu.com" th:attr="title='百度'">百度</a>

<!--设置 title、href 多个属性-->

<a href="" th:attr="title='前往百度',href='http://baidu.com'">前往百度</a>

<!--设置 href 属性-->

<a href="userList.html" th:attr="href=@{/user/userHome}">用户首页</a>

<!--设置 id 属性，data-schoolName 属性 Html 本身是没有的，但允许用户自定义 -->

<a href="#" th:attr="id='9527',data-target='user'">归海一刀</a>

<p th:abc=*"9527"*>th:abc="9527"</p>

<!--输出：<p abc123="华安">th:abc123="华安"</p>-->

<p th:abc123=*"华安"*>th:abc123="华安"</p>

### Checked selected

<input type=*"checkbox"* name=*"option1"* checked/><span>是否已婚1？</span>

<input type=*"checkbox"* name=*"option2"* checked=*"checked"*/><span>是否已婚2？</span>

---<br>

<!--后台控制器传递了一个：model.addAttribute("isMarry", true);-->

<!--option3、option4 会选中；option5 不会选中-->

<input type=*"checkbox"* name=*"option3"* th:checked=*"${isMarry}"*/><span>是否已婚3？</span>

<input type=*"radio"* name=*"option4"* th:checked=*"${isMarry}"*/><span>是否本科？</span>

<input type=*"radio"* name=*"option5"* th:checked=*"!${isMarry}"*/><span>是否应届生？</span>

---------------------

<select>

<option>a</option>

<option th:selected=*"${isMarry}"*>已婚</option>

<option th:selected=*"${!isMarry}"*>未婚</option>

</select>

<input type=*"text"* th:autofocus=*"false"*>

<input type=*"text"* th:autofocus=*"true"*>

<input type=*"text"* th:autofocus=*"false"*>

## 日期格式化

    <span th:text=*"${#dates.format(date, 'yyyy-MM-dd HH:mm')}"*></span>

## 循环

JSTL 有一个 <c:foreach>，同理 Thymeleaf 也有一个 th:each。作用都是一样的，都是用于遍历数组、List、Set、Map 等数据。

### 在Select上循环

<option th:each="city : ${list}" th:text="${city.name}" th:selected="${cityName} eq ${city.name}">横岗</option>

### 状态变量loopStatus

如果不指定 为变量+Stat

* index: 当前迭代对象的index（从0开始计算）
* count: 当前迭代对象的index(从1开始计算)
* size: 被迭代对象的大小 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开始计算）
* 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

<tr th:each="city,status : ${list}" th:style="${status.odd}?'background-color:#c2c2c2'">

<!-- EL JSTL-->

<td th:text = "${status.count}"></td>

<td th:text = "${city.id}"></td>

<td th:text = "${city.name}"></td>

</tr>

## If/else

<p th:if=*"${isMarry}"*>已婚1</p>

<p th:unless=*"${isMarry}"*>未婚</p>

## Switch/case 多条件判断

<div th:switch=*"1"*>

<p th:case=*"0"*>管理员</p>

<p th:case=*"1"*>操作员</p>

<p th:case=*"\*"*>未知用户</p>

</div>

<!--数字类型：当没有 case 匹配时，取默认值，当有多个匹配，只取第一个-->

<div th:switch=*"-1"*>

<p th:case=*"0"*>管理员</p>

<p th:case=*"\*"*>操作员</p>

<p th:case=*"\*"*>未知用户</p>

</div>

<!--布尔类型，多个case满足时，只取第一个-->

<div th:switch=*"${isMarry}"*>

<p th:case=*"true"*>已婚</p>

<p th:case=*"true"*>已成年</p>

<p th:case=*"false"*>未婚</p>

</div>

<!--字符串类型-->

<div th:switch=*"'For China'"*>

<p th:case=*"'For USA'"*>美国</p>

<p th:case=*"'For UK'"*>英国</p>

<p th:case=*"'For China'"*>中国</p>

<p th:case=*"\*"*>未知国籍</p>

</div>

## 内联表达式

[[...]] 等价于 th:text（结果将被 HTML 转义），[(...)] 等价于 th:utext（结果不会执⾏HTML转义）

<p>[[${china}]]</p>

<p>[(${china})]</p>

<p>[[Lo1ve]]</p>

<p>[['I Love You Baby']]</p>

<p>[(9527)]</p>

th:inline =“none” 来禁⽤内联。

### 内联 JavaScript

<script type="text/javascript" th:inline="javascript">

**var** info = [[${info}]];

**var** age = [[${age}]];

**var** id = [[${id}]];

**var** name = [[${name}]];

console.log(id, name, age, info);

</script>

### 前后端分离

<script type="text/javascript" th:inline="javascript">

/\*\*

\* Thymeleaf 将自动忽略掉注释之后 和 分号之前的所有内容,如下为 "前端测试"

\*/

**var** info = /\*[[${info}]]\*/ "前端测试";

console.log(info);

</script>

## 对象属性

### URL

<p>${param.size()}=[[${param.size()}]]</p>

<!--/\*判断请求参数是否为空\*/-->

<p>${param.isEmpty()}=[[${param.isEmpty()}]]</p>

<!--获取某个参数值，不存在时为null-->

<p>${param.u\_id}=[[${param.u\_id}]]</p>

### Session

<p>${session.size()}=[[${session.size()}]]</p>

<!--/\*判断请求参数是否为空\*/-->

<p>${session.isEmpty()}=[[${session.isEmpty()}]]</p>

<!--获取某个参数值，不存在时为null-->

<p>${session.user.id}=[[${session.user.id}]]</p>

## 完整文档

https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#base-objects