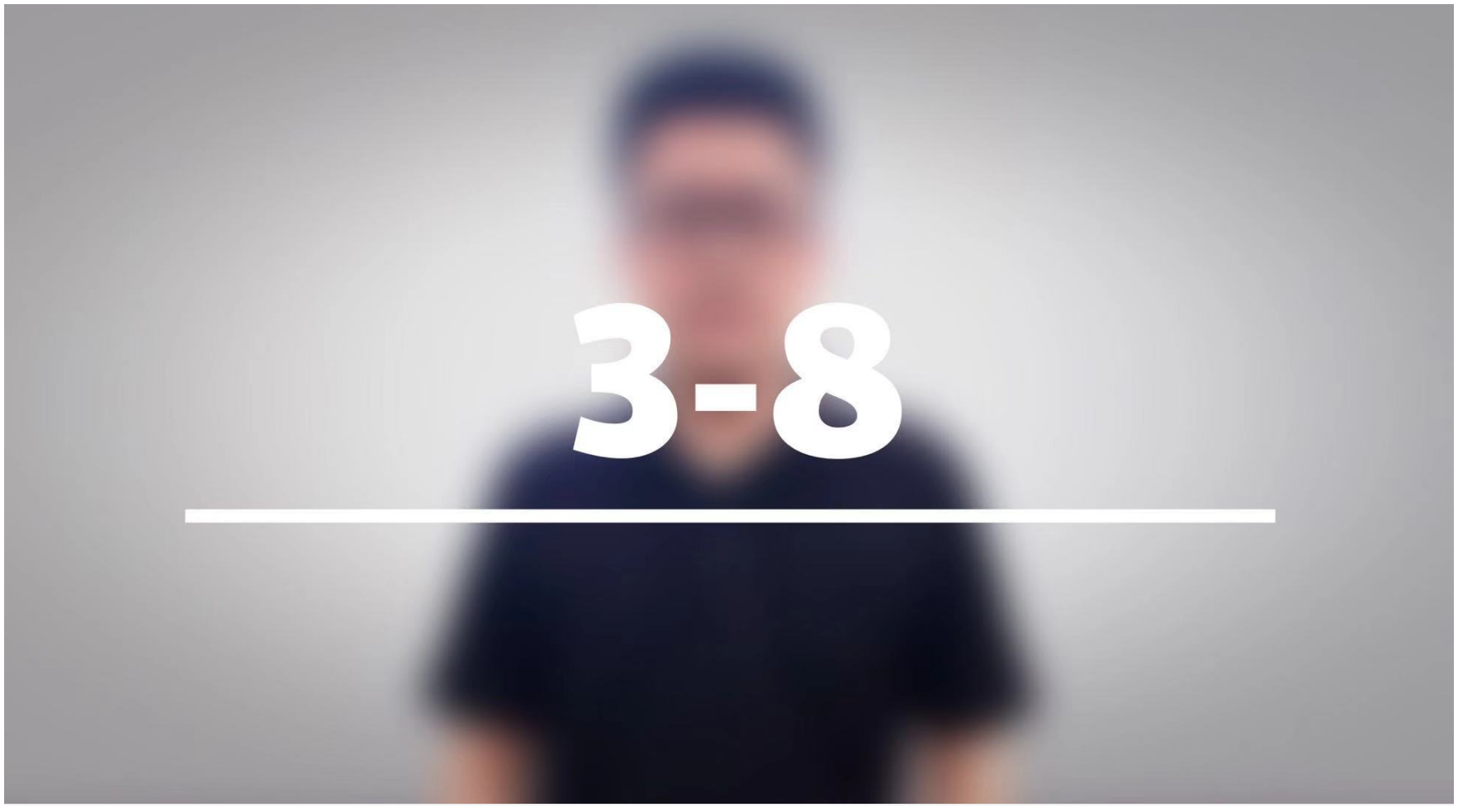


第 3-8 课：Spring Data JPA 和 Thymeleaf 综合实践

在前面课程中，我们学习了 Spring Boot Web 开发、JPA 数据库操作、Thymeleaf 和页面交互技术，这节课综合这些内容做一个用户管理功能，包括展示用户列表（分页）、添加用户、修改用户和删除用户。有人说程序员的一生都是在增、删、改、查，这句话不一定全对，但也有一定的道理，相比于这句话，我更认同的是这句：程序员的技术学习都是从增、删、改、查开始的。

这节课将介绍如何使用 JPA 和 Thymeleaf 做一个用户管理功能。

配置信息

添加依赖

pom 包里面添加 JPA 和 Thymeleaf 的相关包引用。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-Thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-Jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

配置文件

在 application.properties 中添加配置：

```
spring.datasource.url=jdbc:mysql://localhost:3306/test?serverTime-
zone=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.My-
SQL5InnoDBDialect
spring.jpa.show-sql= true

spring.thymeleaf.cache=false
```

其中，spring.Thymeleaf.cache=false 是关闭 Thymeleaf 的缓存，不然在开发过程中修改页面不会立刻生效需要重启，生产可配置为 true。

在项目 resources 目录下会有两个文件夹：static 目录用于放置网站的静态内容如 css、js、图片；templates 目录用于放置项目使用的页面模板。

启动类

启动类需要添加 Servlet 的支持：

```
@SpringBootApplication
public class JpaThymeleafApplication extends SpringBootServletIni-
tializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplication-
Builder application) {
        return application.sources(JpaThymeleafApplication.class);
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(JpaThymeleafApplication.class, args);
    }
}
```

添加 SpringBootServletInitializer 是为了支持将项目打包成独立的 war 在 Tomcat 中运行的情况。

数据库层

实体类映射数据库表：

```
@Entity
public class User {
    @Id
    @GeneratedValue
    private long id;
    @Column(nullable = false, unique = true)
    private String userName;
    @Column(nullable = false)
    private String passWord;
    @Column(nullable = false)
    private int age;
    @Column(nullable = false)
    private Date regTime;
    //省略getter settet方法
}
```

继承 JpaRepository 类会自动实现很多内置的方法，包括增、删、改、查，也可以根据方法名来自动生成相关 SQL。

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("select u from User u")
    Page<User> findList(Pageable pageable);
    User findById(long id);
    User findByUserName(String userName);
    void deleteById(Long id);
}
```

Repository 内编写我们需要的 SQL 和分页查询。

实现一个添加功能

在处理前端业务的时候一般是使用 param 结尾的参数来处理，在项目下新建 param 包，在 param 包下创建 UserParam 类接收添加用户的请求参数。另外，需要对接收的参数做校验，按照前面课程的内容，引入 hibernate-validator 做校验。

```
public class UserParam {
    private long id;
    @NotEmpty(message="姓名不能为空")
    private String userName;
    @NotEmpty(message="密码不能为空")
    @Length(min=6,message="密码长度不能小于6位")
    private String passWord;
    @Max(value = 100, message = "年龄不能大于100岁")
    @Min(value= 18 ,message= "必须年满18岁！" )
    private int age;
    //省略getter settet方法
}
```

Controller 负责接收请求，首先判断参数是否正确，如果有错误直接返回页面，将错误信息展示给用户，再判断用户是否存在，如果用户已经存在同样返回页面给出提示。验证通过后，将 UserParam 属性复制到 User 并添加用户注册时间，最后将用户信息保存到数据库中。

```
@RequestMapping("/add")
public String add(@Valid UserParam userParam,BindingResult re-
sult, Model model) {
    String errorMsg="";
    // 参数校验
    if(result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg=errorMsg + error.getCode() + "-" + error.get-
DefaultMessage() +";";
        }
        model.addAttribute("errorMsg",errorMsg);
        return "user/userAdd";
    }
    //判断是否重复添加
    User u= userRepository.findByUserName(userParam.getUserName());
    if(u!=null){
        model.addAttribute("errorMsg","用户已存在!");
        return "user/userAdd";
    }
    User user=new User();
    BeanUtils.copyProperties(userParam,user);
    user.setRegTime(new Date());
    //保存
    userRepository.save(user);
    return "redirect:/list";
}
```

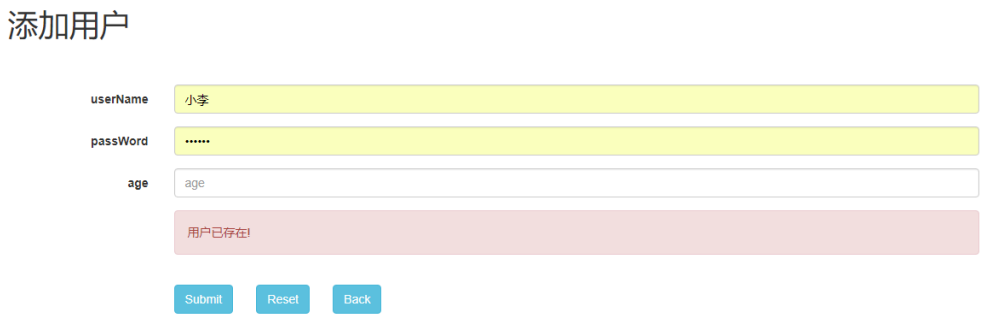
- model 对象主要用于传递控制方法处理数据到结果页面;
- return "redirect:/list"; 代表添加成功后直接跳转到用户列表页面。

添加用户部分页面（userAdd.html）

前端页面引入了 Bootstrap 前端框架，以下表单按照 Bootstrap 的格式进行设计。

```
<form class="form-horizontal"   th:action="@{/add}"  method="post">
    <!-- 表单内容-->
    <div class="form-group">
        <label for="userName" class="col-sm-2 control-label">user-
Name</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="userName"
  id="userName" placeholder="userName"/>
        </div>
    </div>
    <div class="form-group">
        <label for="password" class="col-sm-2 control-label" >pass-
Word</label>
        <div class="col-sm-10">
            <input type="password" class="form-control" name="pass-
Word" id="passWord" placeholder="passWord"/>
        </div>
    </div>
    ....
    <!-- 错误信息展示区-->
    <div class="form-group">
        <label  class="col-sm-2 control-label"></label>
        <div class="col-sm-10">
            <div th:if="${errorMsg != null}"  class="alert alert-da
nger" role="alert" th:text="${errorMsg}">
            </div>
        </div>
    </div>
    <!-- 按钮区-->
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <input type="submit" value="Submit" class="btn btn-info
" />
                 
            <input type="reset" value="Reset" class="btn btn-info"
/>
                 
            <a href="/toAdd" th:href="@{/list}" class="btn btn-info
">Back</a>
        </div>
    </div>
</form>
```

效果图:



用户列表

参考前面课程，JPA 依赖 Pageable 为用户列表页做分页，默认每页展示 6 个用户，并且按照用户注册的倒序来排列，具体信息如下：

```
@RequestMapping("/list")
public String list(Model model,@RequestParam(value = "page", de-
faultValue = "0") Integer page,
                   @RequestParam(value = "size", defaultValue = "6"
) Integer size) {
    Sort sort = new Sort(Sort.Direction.DESC, "id");
    Pageable pageable = PageRequest.of(page, size, sort);
    Page<User> users=userRepository.findList(pageable);
    model.addAttribute("users", users);
    return "user/list";
}
```

- @RequestParam 常用来处理简单类型的绑定，注解有三个属性：value、re - quired 和 defaultValue；value 用来指定要传入值的 ID 名称，required 用来指示参数是否必须绑定，defaultValue 可以设置参数的默认值。

前端页抽取一个公共的分页信息——page.html，页面部分信息如下：

```
<div th:if="${(users.totalPages le 10) and (users.total-
Pages gt 0)}" th:remove="tag">
    <div th:each="pg : ${#numbers.sequence(0, users.total-
Pages - 1)}" th:remove="tag">
            <span th:if="${pg eq users.getNumber()}" th:remove="tag
">
                <li class="active"><span class="current_-
page line_height" th:text="${pg+1}">${pageNumber}</span></li>
            </span>
        <span th:unless="${pg eq users.getNumber()}" th:remove="tag
">
                <li><a href="#" th:href="@{${pageUrl}(page=${pg})}"
 th:text="${pg+1}"></a></li>
            </span>
    </div>
</div>

<li th:if="${users.hasNext()}"><a href="#" th:href="@{${page-
Url}(page=${users.number+1})}">下一页</a></li>
<li><a href="#" th:href="${users.totalPages le 0 ? page-
Url+'page=0':pageUrl+'&amp;page='+(users.totalPages-1)}">尾页</a></
li>
<li><span th:utext="'共'+${users.totalPages}+'页 / '+${users.to-
talElements}+' 条'"></span></li>
```

page.html 页面的作用是显示主页的页码，包括首页、末页、第几页，共几页这类信息，需要根据页码的数据进行动态调整。页面中使用了 Thymeleaf 大量语法：th:if 判断、th:each 循环、th:href 链接等，分页信息主要从后端传递的 Page 对象获取。

然后在 list.html 页面中引入 page.html 页面分页信息。

```
<h1>用户列表</h1>
<br/><br/>
<div class="with:80%">
    <table class="table table-hover">
        <thead>
         <!-- 表头信息-->
        <tr>
            <th>#</th>
            <th>User Name</th>
            <th>Password</th>
            <th>Age</th>
            <th>Reg Time</th>
            <th>Edit</th>
            <th>Delete</th>
        </tr>
        </thead>
        <tbody>
        <!-- 表循环展示用户信息-->
        <tr  th:each="user : ${users}">
            <th scope="row" th:text="${user.id}">1</th>
            <td th:text="${user.userName}">neo</td>
            <td th:text="${user.passWord}">Otto</td>
            <td th:text="${user.age}">6</td>
            <td th:text="${#dates.format(user.reg-
Time, 'yyyy/MMM/dd HH:mm:ss')}"></td>
            <td><a th:href="@{/toEdit(id=${user.id})}">edit</a></td
>
            <td><a th:href="@{/delete(id=${user.id})}"  onclick="re
turn confirm('确认是否删除此用户? ')"  >delete</a></td>
        </tr>
        </tbody>
    </table>
    <!-- 引入分页内容-->
    <div th:include="page :: pager" th:remove="tag"></div>
</div>
<div class="form-group">
    <div class="col-sm-2 control-label">
        <a href="/toAdd" th:href="@{/toAdd}" class="btn btn-info">a
dd</a>
    </div>
</div>
```

`<tr th:each="user : ${users}">` 这里会从 Controler 层 model set 的对象去获取相关的内容，th:each 表示会循环遍历对象内容。

效果图如下：

修改功能

点击修改功能的时候，需要带上用户的 ID 信息：

```
<td><a th:href="@{/toEdit(id=${user.id})}">edit</a></td>
```

后端根据用户 ID 获取用户信息，并放入 Model 中。

```
@RequestMapping("/toEdit")
public String toEdit(Model model,Long id) {
    User user=userRepository.findById(id);
    model.addAttribute("user", user);
    return "user/userEdit";
}
```

修改页面展示用户信息，以下为 userEdit.html 页面部分内容：

```
<form class="form-horizontal"   th:action="@{/edit}" th:object="${u
ser}"  method="post">
    <!--隐藏用户 ID-->
    <input type="hidden" name="id" th:value="*{id}" />
    <div class="form-group">
        <label for="userName" class="col-sm-2 control-label">user-
Name</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="userName"
  id="userName" th:value="*{userName}" placeholder="userName"/>
        </div>
    </div>
    <div class="form-group">
        <label for="password" class="col-sm-2 control-label" >pass-
Word</label>
        <div class="col-sm-10">
            <input type="password" class="form-control" name="pass-
Word" id="passWord"  th:value="*{passWord}" placeholder="passWord"/
>
        </div>
    </div>

    <!--错误信息-->
    <div class="form-group">
        <label  class="col-sm-2 control-label"></label>
        <div class="col-sm-10">
            <div th:if="${errorMsg != null}"  class="alert alert-da
nger" role="alert" th:text="${errorMsg}">

            </div>
        </div>
    </div>

    <!--按钮区-->
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <input type="submit" value="Submit" class="btn btn-info
" />
                 
            <a  th:href="@{/list}" class="btn btn-info">Back</a>
        </div>

    </div>
</form>
```

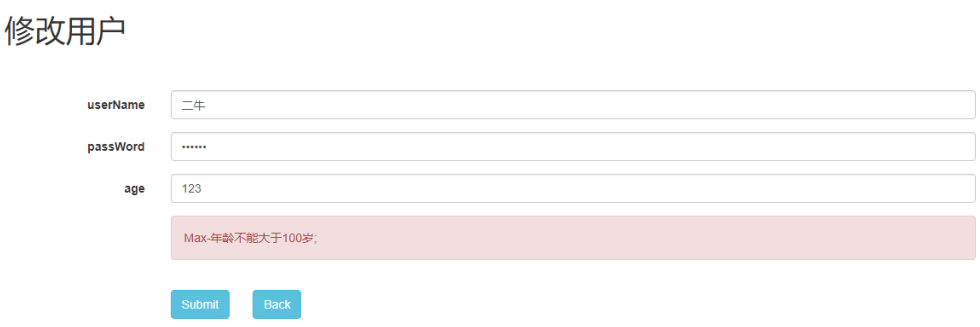
修改完成后提交到后台:

```
@RequestMapping("/edit")
public String edit(@Valid UserParam userParam, BindingResult re-
sult,Model model) {
    String errorMsg="";
    //参数校验
    if(result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg=errorMsg + error.getCode() + "-" + error.get-
DefaultMessage() +";";
        }
        model.addAttribute("errorMsg",errorMsg);
        model.addAttribute("user", userParam);
        return "user/userEdit";
    }

    //复制属性保持修改后数据
    User user=new User();
    BeanUtils.copyProperties(userParam,user);
    user.setRegTime(new Date());
    userRepository.save(user);
    return "redirect:/list";
}
```

后台同样需要进行参数验证，无误后修改对应的用户信息。

效果图:

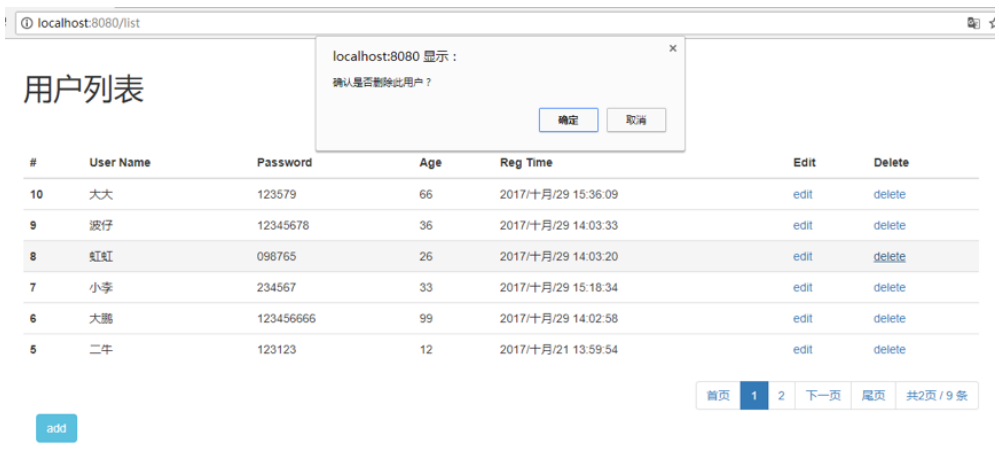


删除功能

单击删除按钮的时候需要用户再次确认，确认后才能删除。

```
<td><a th:href="@{/delete(id=${user.id})}"  onclick="return con-
firm('确认是否删除此用户? ')"  >delete</a></td>
```

效果如下:



后端根据用户 ID 进行删除即可。

```
@RequestMapping("/delete")
public String delete(Long id) {
    userRepository.delete(id);
    return "redirect:/list";
}
```

删除完成之后，再跳转到用户列表页。

总结

用户管理功能包含了用户的增加、修改、删除、展示等功能，也是我们日常开发中最常用的四个功能。在实现用户管理功能的过程中使用了 JPA 的增加、修改、删除、查询、分页查询功能；使用了 Thymeleaf 展示用户信息，在 list 页面引入分页模板，使用了 Thymeleaf 内嵌的 dates 对日期进行了格式化；经过今天的学习较全面演练了前期的学习内容。

> 点击这里下载源码 (https://github.com/ityouknow/spring-boot-lean - ing/tree/gitbook_column2.0)。