## 第一章 入门前准备

### 1.学员自我介绍

### 2.分组

### 3.环境要求

### 4.个人设备要求

### 5.课程内容&书本问题

### 6.教程《SpringBoot教程》

* .SpringBoot教程与项目开发的关系
* 为什么要用SpringBoot作为项目开发的基础，该框架简单、快速入门、主流技术、学习资料丰富

## 第二章 项目开的组织和管理

### 1.项目开发的组织和步骤

### 2.项目开发的成本和考虑因素

### 3.软件企业的资质

### 4.项目开发人员的选择

### 5.对于编程语言工具简单的即使最好的

### 6.软件开发人员自我成长和不断再学习

## 第三章 项目开发所需的组件

### 1.后端开发框架

Servlet、Stuts、JSF、Seam、Spring、SpringMVC、SpringBoot、SpringCloud

Java中的辅助框架：

缓存：Spring Cache、Redis(Go语言编写的)

调度：Quartz

安全访问及认证：Spring Security、Shiro

ORM数据访问及映射：JDBC、Hibernate、JPA、Mybatis、JTA、EJB

事务：Fescar （阿里）

拦截：Spring AOP、AspectJ

日志：Logback、Log4J

全文检索系统：Elastic

报表系统：帆软报表

图表系统：HighChart、百度EChart

通讯组件：Kafka、RabbitMQ、RocketMQ、ActiveMQ、Netty

单元测试：Junit4、Junit5

服务测试工具：Postman、Apifox（国产）

分布式服务框架：Dubbo、Nacos

### 2.前端框架

JavaScript/jQuery/JSP/freemaker/Thymeleaf/VueJS

AnglurJS/ReactJS

渲染框架：BootStrap/LayUI/EasyUI/Ace

### 3.项目管理工具

Maven/Gradle

### 4.Web中间件

Tomcat/Jetty/GlassFish/JBOSS/金蝶Apusic/东方通

### 5.云平台虚拟化部署

Docker/kubernetes(简称K8S)

## 第四章 需求管理&分析

### 掌上医院APP的需求及设计思路

公司开发了一款App，实现了可以使用手机挂号的功能，之前只能到医院挂专家号或打电话预约，但打电话经常会占线，拨不通，而且不知道医院有哪些专家，这一天究竟上不上班，使用者使用起来困难重重。但手机App解决了这个问题，解决的完美吗?需求分析人员仔细考察使用了这款App后发现并不完美，因为这款App的思路是：

我得病了-->点开App，我要去医院看病-->我家附近有家3甲医院（新区医院）-->我得了重感冒，我应该去看门诊内科-->听说有个汪大夫是专家,能不能挂他的号，当天不行，就换明天-->上App找到新区医院-->找到门诊内科-->找到汪大夫-->挂他的专家门诊号

这里面的问题在哪？

1. 自己给自己判断是什么病
2. 熟悉地形
3. 熟悉当地医院
4. 熟悉专家

假设你是个刚来到这个城市的旅行者，你怎么办呢？假设你身体一直都健康，从来不去医院，最近突然生病了，扛不住了，才去找医院，两眼一抹黑，你怎么办呢？找同事，找盆友，问来问去，一人一个说法，信息很多，但都不太清晰，你怎么办呢？

那么App该怎么改呢？

我得病了-->点开App，我要去医院看病-->勾选我的症状-->系统判断我大致得了什么病或哪方面有问题(专家系统)-->推荐我附近10公里范围的数家医院而且能看此类病的，有三甲医院，有社区医院，有近有远-->根据自己的情况选择一家医院（例如，为了看病便宜，选择社区医院；为了赶时间，选择距离最近的医院；为了治疗效果，选择三甲医院）-->系统推荐3天内有开门诊的此类病专家（可以查看该专家介绍），同时根据病友评分推荐的此类病专家（当然是在这三天内有开门诊的此类病专家）-->选择专家-->-->挂他的专家门诊号

### 2.自动刷牙机器人的需求分析

牙齿究竟有几个面呢？刷干净牙需要刷几个面？两个面，六个面？

刷牙时从上往下，还是从下往上，还是从左至右，还是从右至左？

刷牙的步骤：

固定身体-->固定头-->打开口腔-->清水漱口-->牙刷上牙膏-->开始刷牙-->清水漱口-->判断漱清口腔了吗，没有重复清水漱口，否则下一步-->擦干嘴-->松开头部固定-->松开身体固定

### 3.电梯运行控制

电梯的运行控制实际上是个典型的计算机控制系统，从现代软件控制思路上分析，是个比较复杂的对象系统。

* 电梯对象

电梯这个对象功能：

上行方法：电梯执行上行，异常会触发报警方法

下行方法：电梯执行下行，异常会触发报警方法

暂停方法：电梯执行暂停，异常会触发报警方法

停止方法：电梯执行停止，异常会触发报警方法

关门方法：电梯执行关门，异常会触发报警方法

开门方法：电梯执行开门，异常会触发报警方法

开灯方法：电梯执行开灯，异常会触发报警方法

关灯方法：电梯执行关灯

报警方法：电梯执行报警

称重方法：电梯执行称重，超标会触发报警方法

开通风方法：电梯会打开通风，异常会触发报警方法

关通风方法：电梯会关闭通风

判断楼层号方法：电梯会判断到了哪一层

电梯的状态属性：

Status:0停止1:暂停2:下行3:上行

FloorLevel：num 楼层号

* 客户请求对象

客户请求对象功能：

请求上行方法：申请上行，但不知层数，电梯外按键

请求下行方法：申请下行，但不知层数，电梯外按键

请求楼层方法：申请上行/下行，确定层数，电梯内按键

取消楼层方法：取消上行/下行，确定层数，电梯内按键

请求开门方法：申请开门，电梯内按键

请求关门方法：申请关门，电梯内按键

客户请求状态属性：

请求启用/取消

* 队列对象

队列对象功能：队列对象中存储排队的请求对象

新增队列对象方法

完成队列对象方法

取消队列对象方法

查看队列对象方法

属性：

队列对象的排序规则：是否是按时间顺序？不是，是按行进方向和最近顺序。

## 第五章 开发环境的配置

### 1.java版本

java1.8或称为java8，安装好后要进行java虚拟机环境测试。

C:/>java -version

### 2.idea安装

* 注意注册问题，先选30天试用。
* 修改字符集utf-8
* 设置IDEA主题

### 3.maven安装配置

* maven3.3.9解压至D:/dev2022下
* 创建自己的本地仓库D:/dev2022/repository
* 修改Maven配置文件
  + <localRepository>D:\dev2022\repository</localRepository>
  + 增加国内阿里云镜像

<mirrors>

<mirror>

<id>aliyunmaven</id>

<mirrorOf>\*</mirrorOf>

<name>阿里云公共仓库</name>

<url>https://maven.aliyun.com/repository/public</url>

</mirror>

<mirror>

<id>aliyunmaven</id>

<mirrorOf>\*</mirrorOf>

<name>阿里云谷歌仓库</name>

<url>https://maven.aliyun.com/repository/google</url>

</mirror>

<mirror>

<id>aliyunmaven</id>

<mirrorOf>\*</mirrorOf>

<name>阿里云阿帕奇仓库</name>

<url>https://maven.aliyun.com/repository/apache-snapshots</url>

</mirror>

<mirror>

<id>aliyunmaven</id>

<mirrorOf>\*</mirrorOf>

<name>阿里云spring仓库</name>

<url>https://maven.aliyun.com/repository/spring</url>

</mirror>

<mirror>

<id>aliyunmaven</id>

<mirrorOf>\*</mirrorOf>

<name>阿里云spring插件仓库</name>

<url>https://maven.aliyun.com/repository/spring-plugin</url>

</mirror>

</mirrors>

### 4.idea设置

* 新建项目目录设置为D:\dev2022\code，工程名为code
* 设置maven为D:/dev2022/maven-3.3.9
* 配置文件为其下的settings.xml
* 本地仓为D:/dev2022/repository

### 5.MySQL数据库安装

* 下载MySQL 必须是8.X社区版
  + 数据库用户：root/root
* 注意DataFiles一般不要放在C盘
* 安装时注意勾选验证方式兼容MySQL5.X
* Navicat数据库第三方工具安装设置
  + 配置连接
  + 创建数据库springboot
  + 导入数据库脚本
  + 培训数据库使用基本操作

### 6.谷歌浏览器安装

### 7.文本对比工具的使用

## 第六章 欢迎进入SpringBoot世界

### 1.新建java项目springboot

### 2.在springboot项目中新建demo模块

### 3.选择Spring initializer,选择Spring Web

* Name：项目名或模块名，如果是project，相当于Eclipse的workspace
* Location:存储路径
* Group ID：组织机构名，通常用域名形式表示，如edu.nxgs.springboot
* Artifact:工艺品，这里通常等同于项目名或模块名
* Package name:包名

### 4.在com.fd.springboot下新建包：controller

### 5.在controller下新建AdminController.java文件

**package** com.fd.springboot.controller;  
  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.ui.Model;  
**import** org.springframework.web.bind.annotation.\*;  
 *//@RequestMapping(value = "/admin")*@Controller  
**public class** WelcomeController {  
 @RequestMapping(value = **"/admin/sayhello"**, method = RequestMethod.***GET***) *//路径,如果不写method，即包含了GET、POST两种方法* @ResponseBody *//返回的是JSON格式字符串* **public** String sayHello() {  
 **return "欢迎你来到SpringBoot世界！"**;  
 }

}

### 6.编译运行

### 7.页面访问

（1）在浏览器中访问<http://127.0.0.1:8080/hello>,查看结果

（2）在浏览器中访问http://localhost:8080/hello,查看结果

（3）在浏览器中访问http://本机机器名:8080/hello,查看结果

### 8.修改端口

（1）在application.properties中增加

 server.port=8090

重新运行，查看结果

（2）在application.ymal或application.yml文件中增加

 server:  
  port: 8090

注意：节名和值之间用冒号+空格分隔，不能省略空格

### 9.上下文路径的根

实际上在项目配置文件中隐含设置了上下文路径的根

 server.servlet.context-path=/

假设改为：

 server.servlet.context-path=/demo1

那么在程序运行后，浏览器访问就必须输入：

<http://127.0.0.1:8080/demo1/hello>

注意:

* 上下文路径非常重要
* 在路径前加“/”和路径后加“/”要特别注意，有时会出现“//”，如果不加，又有可能会出现“”，两种情形都不是我们想要的，虽然在IDEA调试运行时没有问题，但正是部署就会发现有些路径找不到了，尤其是RequestMapping有双层的时候，内外层，系统会自动加“/”,稳妥的办法是用拦截器拦截路径，把“//”变成“/”。

### 10.本章注解

1. @Controller控制器
2. @ResponseBody返回JSON类型
3. RequestMapping设置路径
4. GetMapping设置请求路径
5. PostMapping设置提交路径

## 第七章 读取项目配置参数

### 1.使用@Value读取系统配置参数

在application.properties中增加：

 company.name=银川方达电子系统工程有限公司  
 company.websit=http://www.5starsoft.com.cn

在AdminController.java 中增加：

@GetMapping(**"/getpa1"**)  
@ResponseBody  
**public** String getParams1(){  
 System.***out***.println(**"公司名称1："**+**companyName**);  
 System.***out***.println(**"公司网站1："**+**companyWebSite**);  
 **return "公司名称1:"**+**companyName**+**",公司网站1:"**+**companyWebSite**;  
}

​

注意：@Value是从项目配置文件中读取参数的注解

​

### 2.使用@Component读取系统配置参数

*//@Component(value="Company")*

*//@Component("Company")*

@Component *//Bean是由Web容器(tomcat)在系统启动初始化时自动创建,而且一直存在，直到程序关闭*@ConfigurationProperties(prefix = **"company"**)  
**public class** Company {  
 **private** String **name**;  
 **private** String **website**;  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getWebsite() {  
 **return website**;  
 }  
  
 **public void** setWebsite(String website) {  
 **this**.**website** = website;  
 }  
}

AdminController.java改成：

 @GetMapping(**"/getpa2"**)  
@ResponseBody  
**public** String getParams2(){  
 System.***out***.println(**"公司名称2："**+**company**.getName());  
 System.***out***.println(**"公司网站2："**+**company**.getWebsite());  
 **return "公司名称2:"**+**company**.getName()+**",公司网站2:"**+**company**.getWebsite();  
}

### 3.注解@Autowired自动装配

我们编写spring 框架的代码时候。一直遵循是这样一个规则：所有在spring中注入的bean 都建议定义成私有的域变量。并且要配套写上 get 和 set方法。在本例中使用了company.getName()和company.getWebsit()，但这里并没有声明get/set方法，使用@Autowired自动装配会自动帮我们引入事先声明的get/set方法。

### 4.项目&模块的导入导出

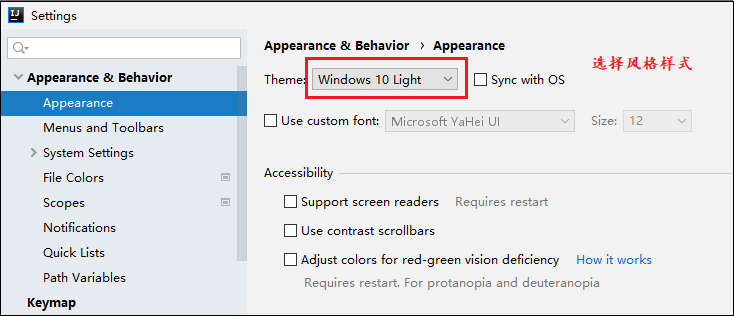
（1）项目和模块的关系

（2）项目或模块的导入导出

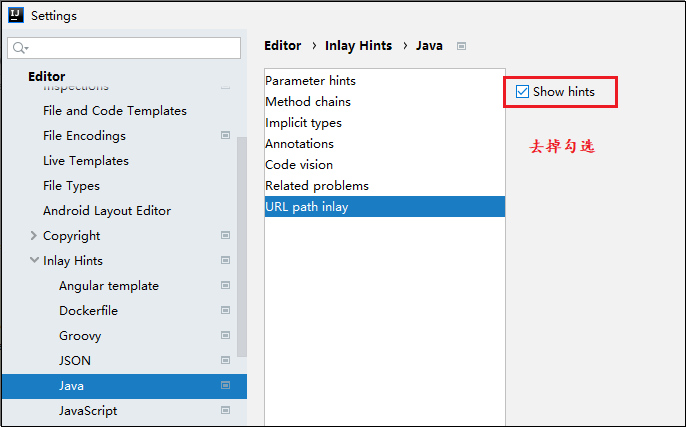
（3）调试跟踪

* sout -->System.out.println(字符串或变量)；
* Debug跟踪，增加断点
* 写入日志：log.info("查询学生信息："+student.name);

### 5.IDEA背景设置



### 6.关闭IDEA编码时路径提示



### 7.上下文路径和端口

context就是“容器”，放的就是应用程序的所有资源，要用时候就访问它，所以context里面的东西，在同一个应用程序里面是全局的。 context的路径就是上下文路径，这个路径也只是个相对路径，因为真正部署以后项目的路径是：部署路径+context路径，上下文路径在Springboot项目中是在application.properties中声明。

端口是项目在编辑运行期间的端口，并不是真正部署时的端口。也是在application.properties中声明。

**server.port=8080**  
**server.servlet.context-path=/aa**

在控制器方法中：

*//上下文路径获取*@GetMapping(**"/admin/getct"**)  
**public** @ResponseBody String getContextPath(HttpServletRequest request){  
 String context = request.getContextPath();  
 String requestURI = request.getRequestURI();  
 **return "上下文路径:"**+context+**",访问请求:"**+requestURI;  
}

### 8.页面调试

（1）打印到控制台“sout”--”System.out.println”

（2）Debug模式

（3）日志模式

### 9.多配置文件

（1）为什么会有多配置文件？

（2）多配置文件之间的主从关系、命名规则

application-dev.properties文件：

**server.port**=**8080  
server.servlet.context-path**=**/aa***#关闭thymeleaf缓存，默认为true,打开状态***spring.thymeleaf.cache**=**false***#thymeleaf\*\*模板前缀，此时为默认值***spring.thymeleaf.prefix**=**classpath:/templates/***#thymeleaf模板后缀，此时为默认值***spring.thymeleaf.suffix**=**.html***#自定义参数***company.name**=**银川方达电子系统工程有限公司  
company.websit**=**http://www.5starsoft.com.cn**

application-product.properties文件

**server.port**=**8081  
server.servlet.context-path**=**/aa***#关闭thymeleaf缓存，默认为true,打开状态***spring.thymeleaf.cache**=**false***#thymeleaf\*\*模板前缀，此时为默认值***spring.thymeleaf.prefix**=**classpath:/templates/***#thymeleaf模板后缀，此时为默认值***spring.thymeleaf.suffix**=**.html***#自定义参数***company.name**=**银川方达电子系统工程有限公司  
company.websit**=**http://www.5starsoft.com.cn**

application-test.properties文件

**server.port**=**8082  
server.servlet.context-path**=**/aa***#关闭thymeleaf缓存，默认为true,打开状态***spring.thymeleaf.cache**=**false***#thymeleaf\*\*模板前缀，此时为默认值***spring.thymeleaf.prefix**=**classpath:/templates/***#thymeleaf模板后缀，此时为默认值***spring.thymeleaf.suffix**=**.html***#自定义参数***company.name**=**银川方达电子系统工程有限公司  
company.websit**=**http://www.5starsoft.com.cn**

application.properties文件

*#激活的配置文件***spring.profiles.active**=**dev**

### 11.Get/Set方法的省略

（1）在pom.xml文件中添加依赖

**<dependency>  
    <groupId>org.projectlombok</groupId>  
    <artifactId>lombok</artifactId>  
  </dependency>**

（2）在类文件中增加注解

@AllArgsConstructor //包含所有字段的构造方法

@NoArgsConstructor //无参构造方法

@Data //增加Get/Set方法

（3）project和module之间的关系

### 12.本章注解

（1）@Value简单类型属性注入

（2）@Autowire自动装配引用类型属性自动注入

（3）@Component组件

（4）@RestController相当于@Controller+@ResponseBody

## 第八章 集成Thymeleaf

### 1.Thymeleaf介绍

**Thymeleaf**（百香果）是用来开发Web和独立环境项目的服务器端的Java模版引擎。

Spring官方支持的服务的渲染模板中，并不包含**jsp**。而是**Thymeleaf**和**Freemarker**等，而**Thymeleaf**与SpringMVC的视图技术，及SpringBoot的自动化配置集成非常完美，几乎没有任何成本，你只用关注**Thymeleaf**的语法即可。

**Thymeleaf**特点：

* 动静结合：**Thymeleaf** 在有网络和无网络的环境下皆可运行，即它可以让美工在浏览器查看页面的静态效果，也可以让程序员在服务器查看带数据的动态页面效果。这是由于它支持 html 原型，然后在 html 标签里增加额外的属性来达到模板+数据的展示方式。浏览器解释 html 时会忽略未定义的标签属性，所以 **thymeleaf** 的模板可以静态地运行；当有数据返回到页面时，**Thymeleaf** 标签会动态地替换掉静态内容，使页面动态显示。
* 开箱即用：它提供标准和**spring**标准两种方言，可以直接套用模板实现**JSTL**、 **OGNL**表达式效果，避免每天套模板、改**JSTL**、改标签的困扰。同时开发人员也可以扩展和创建自定义的方言。
* 多方言支持：**Thymeleaf** 提供**spring**标准方言和一个与 **SpringMVC** 完美集成的可选模块，可以快速的实现表单绑定、属性编辑器、国际化等功能。
* 与**SpringBoot**完美整合，**SpringBoot**提供了**Thymeleaf**的默认配置，并且为**Thymeleaf**设置了视图解析器，我们可以像以前操作**jsp**一样来操作**Thymeleaf**。代码几乎没有任何区别，就是在模板语法上有区别。

前端框架介绍：

1. JavaScript
2. jQuery
3. JSP
4. freemaker
5. thymeleaf
6. ReactJS渐进式框架，以NodeJS为基础
7. VUEJS渐进式框架，以NodeJS为基础
8. AngularJS渐进式框架，以NodeJS为基础，AngularJS诞生于2009年，由Misko Hevery 等人创建，后为Google所收购。是一款优秀的前端JS框架，已经被用于Google的多款产品当中。AngularJS有着诸多特性，最为核心的是：MVVM、模块化、[自动化双向数据绑定](https://www.angularjs.net.cn/tutorial/10.html)、语义化标签、[依赖注入](https://www.angularjs.net.cn/tutorial/17.html)等等。

### 2.pom.xml项目配置文件

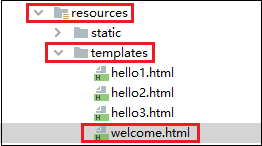
**<!-- 引入Thymeleaf包起步依赖 -->  
 <dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>**  
 ​

### 3.application.properties属性配置文件

*#关闭thymeleaf缓存，默认为true* **spring.thymeleaf.cache=false***#thymeleaf\*\*模板前缀，此时为默认值* **spring.thymeleaf.prefix=classpath:/templates/***#thymeleaf模板后缀，此时为默认值* **spring.thymeleaf.suffix=.html**

### 4.第一个静态页面

（1）页面存放的位置



（2）html标签

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

（3）link标签

**<link th:href="@{/css/btn.css}" rel="stylesheet" />**

（4）script标签

**<script th:src="@{/js/jquery.min.js}"></script>  
 <script th:inline="javascript">...</script>**

（5）img标签

**<img th:src="@{/images/user2-160x160.jpg}" />**

（6）h1普通文本标签

**<h1 th:text="${message}">世界欢迎你</h1>**

1. 变量传值"${参数名}"

welcome.html页面文件：

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 *<!--此句解决上下文路径不是/的情形，页面标签跳转、引用外部文件找不到文件路径的问题 -->  
 <!--http://127.0.0.1:8090/aa -->* <**base th:href="${#request.getContextPath()}+'/'"**>  
  
 <**title**>Title</**title**>  
</**head**>  
<**body**>  
 <**h1 style="color**: **red"**>欢迎你进入SpringBoot</**h1**><**hr**/>  
 <**h3 style="color**: **blue" th:text="${who}"**>姓名</**h3**>  
 <**h3 style="color**: **blue" th:text="${who1}"**>姓名</**h3**>  
 <**h3 style="color**: **blue" th:text="${who2}"**>姓名</**h3**>  
 <**h3 style="color**: **blue"** >[[${who}]]</**h3**> *<!--行内表达式写法 -->* <**audio th:src="@{/audio/沙漠骆驼.mp3}" controls**></**audio**><**br**/>  
 <**img th:src="@{/images/fj1.jpg}" alt="#"**/><**br**/>  
 *<!--控制其方法的访问路径 -->* <**a th:href="@{/admin/hel1}"**>跳转Hello1页</**a**><**br**/>  
 *<!--控制其方法的访问路径 -->* <**a th:href="@{/admin/hel2}"**>跳转Hello2页</**a**><**br**/>  
 *<!--控制其方法的访问路径 -->* <**a th:href="@{/admin/hel3}"**>跳转Hello3页</**a**><**br**/>  
</**body**>  
</**html**>

控制器方法：

@Controller  
**public class** WelcomeController {  
 @RequestMapping(value = **"/admin/sayhello"**, method = RequestMethod.***GET***) *//路径,如果不写method，即包含了GET、POST两种方法* @ResponseBody *//返回的是JSON格式字符串* **public** String sayHello() {  
 **return "欢迎你来到SpringBoot世界！"**;  
 }

}

### 运行测试

在浏览器中输入

http://127.0.0.1:8080/aa/admin/sayhello



注：aa为上下文路径

### 6.页面间跳转

（1）跳转标签

<**a th:href="@{/admin/hel1}"**>跳转Hello1页</**a**><**br**/>

## 第九章 路径访问传参和前后台传参

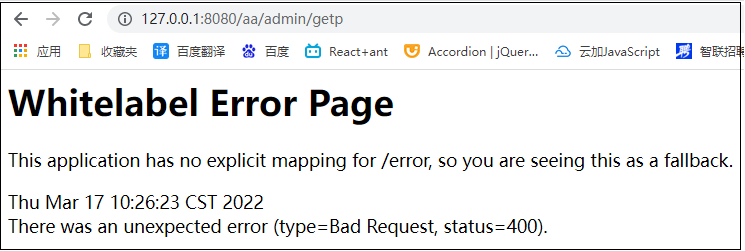
有时候我们需要在访问路径中传参，在访问路径中传参有两种形式，一种是传统的？方式，还有一种是RestFul方式

### 1.传统路径访问传参

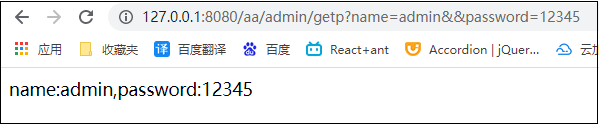
对于有强制路径请求参数的访问，路径中不加请求参数，结果如下：

对于请求路径中有强制传参要求的访问，如果不给参数，访问就会报错，情况如下：

页面会报400错误



当给出了正确的参数以后：



在控制器方法中对请求参数的声明完全体写法：

getPathParam1(@RequestParam(**"name"**) String name,@RequestParam(value=**"password"**,defaultValue=**"000000"**,required=**false**)

获取路径参数的三种方式：

（1）@RequestParam这是Spring的一个注解，可以自动绑定路径参数

@GetMapping(**"/admin/getp1"**)  
**public** @ResponseBody String getPathParam1(@RequestParam(**"name"**) String name,@RequestParam(value=**"password"**,defaultValue=**"000000"**,required=**false**) String password){  
 System.***out***.println(**"name:"**+name+**",password:"**+password);  
 **return "name:"**+name+**",password:"**+password;

（2）采用Java自动绑定方式

@GetMapping(**"/admin/getp2"**)  
**public** @ResponseBody String getPathParam2(String name,String password){ *//自动绑定机制* System.***out***.println(**"name:"**+name+**",password:"**+password);  
 **return "name:"**+name+**",password:"**+password;  
}

1. 采用Servlet的request对象的getParameter方法查询路径参数的值

了解Servlet九大对象

1、out 来源于[Java](http://lib.csdn.net/base/17).io.Writer类，它用于发送输出流到客户端。  
2、request 来源于javax.[servlet](https://so.csdn.net/so/search?q=servlet&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_42734859/article/details/_blank).http.HttpServletRequest接口。它被关联到每一个HTTP请求。  
3、response 来源于javax.servlet.http.HttpServletResponse。response对象用于把取得的数据返回到客户端。  
4、pageContext 提供访问JSP页面的命名空间。它也提供用来访问其他的JSP隐含对象。  
5、session 来源于javax.servlet.http.HttpSession。它用于存储客户端请求的信息，因此它是有状态交互式的。  
6、application 来源于javax.servlet.ServletContext。  
7、config 来源于ServletConfig，它包含了当前JSP/Servlet所在的WEB应用的配置信息。  
8、page 来源于当前被访问JSP页面的实例化。它实际使用的是JSP转换成的Servlet。  
9、exception 用于捕获JSP抛出的异常。它只有在JSP页面属性isErrorPage=true时才可用。

首先需要判定路径参数是在哪个内存对象中，Session?Request?

@GetMapping(**"/admin/getp3"**)  
**public** @ResponseBody String getPathParam3(HttpServletRequest request){ *//自动绑定机制  
 //String name = (String)request.getSession().getAttribute("name");* String name = request.getParameter(**"name"**);  
 *//String[] name1= request.getParameterValues("name");* System.***out***.println(**"name:"**+name);  
  
 **return "name:"**+name;  
}

通过测试证明，传统的路径访问参数，参数对象和值存在于Request对象中。

### 2.RestFul风格路径访问传参

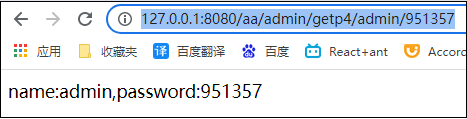
在控制器方法中声明RestFul风格的路径参数如下：

@GetMapping(**"/admin/getp4/{name}/{password}"**)  
**public** String getParam4(@PathVariable(**"name"**) String name,@PathVariable(**"password"**) String password){  
 System.***out***.println(**"name:"**+name+**",password:"**+password);  
 **return "name:"**+name+**",password:"**+password;  
}

注：@PathVariable注解是Spring3.0增加的注解

在浏览器中输入：

http://127.0.0.1:8080/aa/admin/getp4/admin/951357



### Model、ModelMap、ModelAndView对象和@ModelAttribute

前三者都可以作为前后台传参的对象，不过三者都是单向的，要实现双向的传参，需要使用@ModelAttribute

1. Model对象

Model是一个接口，它的实现类为ExtendedModelMap，继承ModelMap类

public class ExtendedModelMap extends ModelMap implements Model

* 请求转发

model对象中存储的数据，相当于存储到了request对象中

我们在jsp/freemaker/thymeleaf中直接按照request对象作用域取值的方式来获取数据即可。

* 重定向

在重定向中，会将第一次请求中model对象的数据作为第二次请求的请求数据携带，第一次请求的model对象销毁。只能携带基本类型的数据。

@ApiOperation(value = **"欢迎"**, notes = **"访问请求http://localhost:8080/aa/admin/welc"**)  
@GetMapping({**"/admin/welc"**})  
**public** String welcomePage(Model msg) {  
 msg.addAttribute(**"who"**, **"周琦"**);  
 msg.addAttribute(**"who1"**, **"王强"**);  
 msg.addAttribute(**"who2"**, **"李雯"**);  
 **return "welcome"**; *//字符串 -- 页面对象*}

1. ModelMap对象

ModelMap继承LinkedHashMap，spring框架自动创建实例并作为controller的入参，用户无需自己创建。

public class ModelMap extends LinkedHashMap

@ApiOperation(value = **"欢迎"**, notes = **"访问请求http://localhost:8080/aa/admin/welc"**)  
@GetMapping({**"/admin/welc"**})  
**public** String welcomePage(ModelMap msg) {  
 msg.addAttribute(**"who"**, **"周琦"**);  
 msg.addAttribute(**"who1"**, **"王强"**);  
 msg.addAttribute(**"who2"**, **"李雯"**);  
 **return "welcome"**; *//字符串 -- 页面对象*}

无论是Mode还是ModelMap底层都是使用BindingAwareModelMap类，所以两者基本没什么区别；

1. ModelAndView对象

而是对于ModelAndView顾名思义，ModelAndView指模型和视图的集合，既包含模型 又包含视图；ModelAndView的实例是开发者自己手动创建的，这也是和ModelMap主要不同点之一；ModelAndView其实就是两个作用，一个是指定返回页面，另一个是在返回页面的同时添加属性；

@GetMapping(value= **"/admin/welc3"**)  
**public** ModelAndView welcome3(@RequestParam String name) {  
 System.***out***.println(**"welcome "**+name);  
  
 *//构建ModelAndView实例，并设置跳转地址* ModelAndView view = **new** ModelAndView(**"welcome3"**);  
 *//将数据放置到ModelAndView对象view中,第二个参数可以是任何java类型* view.addObject(**"who"**,name);  
 *//返回ModelAndView对象view* **return** view;  
}

1. @ModelAttribute

下面代码展示了从后台向前台页面传送数据，可以把@ModelAttribute当Model对象或ModelMap对象使用。

@GetMapping(**"/user/userInsert"**)  
**public** String userInsertBlank(@ModelAttribute(**"user"**) User user){  
 **return "views/userinsert"**;  
}

上面代码与下面代码完全等价

@GetMapping(**"/user/userInsert"**)  
**public** String userNew(Model m){  
 User user = **new** User();  
 m.addAttribute(**"user"**,user);  
 **return "views/userinsert"**;  
}

这是前台向后台传送数据，使用@ModelAttribute来接收数据。

@PostMapping(**"/user/userSave"**)  
**public** @ResponseBody String userPost(@ModelAttribute(**"user"**) User user){  
 user.setUpdateTime(**new** Date());  
 user.setUpdateBy(**"admin"**);  
 System.***out***.println(**"user"**+user.toString());  
 ***log***.info(**"保存："**+user.toString());  
 **userRepository**.save(user);  
 **return "ok"**;  
}

## 第十章 简单Java对象

### 1.简单Java对象（POJO）

（1）声明属性

（2）get/set方法

（3）空构造方法

（4）带参的构造方法

（5）toString()方法

**package** com.fd.springboot.pojo;  
  
**import** java.util.Date;  
  
**public class** Student {  
 **private** Integer **studentId**;  
 **private** String **studentName**;  
 **private** Date **birthday**;  
 **private** String **gender**;  
 **private** String **email**;  
  
 **public** Integer getStudentId() {  
 **return studentId**;  
 }  
  
 **public void** setStudentId(Integer studentId) {  
 **this**.**studentId** = studentId;  
 }  
  
 **public** String getStudentName() {  
 **return studentName**;  
 }  
  
 **public void** setStudentName(String studentName) {  
 **this**.**studentName** = studentName;  
 }  
  
 **public** Date getBirthday() {  
 **return birthday**;  
 }  
  
 **public void** setBirthday(Date birthday) {  
 **this**.**birthday** = birthday;  
 }  
  
 **public** String getGender() {  
 **return gender**;  
 }  
  
 **public void** setGender(String gender) {  
 **this**.**gender** = gender;  
 }  
  
 **public** String getEmail() {  
 **return email**;  
 }  
  
 **public void** setEmail(String email) {  
 **this**.**email** = email;  
 }  
  
 **public** Student() { *//空构造函数* **super**();  
 }  
  
 **public** Student(Integer studentId, String studentName, Date birthday, String gender, String email) {*//带参的构造函数* **this**.**studentId** = studentId;  
 **this**.**studentName** = studentName;  
 **this**.**birthday** = birthday;  
 **this**.**gender** = gender;  
 **this**.**email** = email;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Student{"** +  
 **"studentId="** + **studentId** +  
 **", studentName='"** + **studentName** + **'\''** +  
 **", birthday="** + **birthday** +  
 **", gender='"** + **gender** + **'\''** +  
 **", email='"** + **email** + **'\''** +  
 **'}'**;  
 }  
}

### 测试简单Java类

Spring Boot 2.2.0 版本开始引入 JUnit 5 作为单元测试默认库

[JUnit 5官方文档](https://junit.org/junit5/docs/current/user-guide/" \l "writing-tests-annotations" \t "https://www.cnblogs.com/birdy-silhouette/p/_blank)

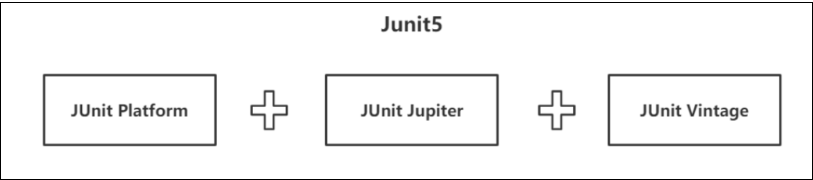
作为最新版本的JUnit框架，JUnit5与之前版本的JUnit框架有很大的不同。由三个不同子项目的几个不同模块组成。

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

JUnit Platform: Junit Platform是在JVM上启动测试框架的基础，不仅支持Junit自制的测试引擎，其他测试引擎也都可以接入。

JUnit Jupiter: JUnit Jupiter提供了JUnit5的新的编程模型，是JUnit5新特性的核心。内部包含了一个测试引擎，用于在Junit Platform上运行。

JUnit Vintage: 由于JUint已经发展多年，为了照顾老的项目，JUnit Vintage提供了兼容JUnit4.x，JUnit3.x的测试引擎。



注意：

* JUnit 5’s Vintage已经从spring-boot-starter-test从移除。如果需要继续兼容Junit4需要自行引入Vintage依赖：
* @Test测试方法注解

@SpringBootTest  
**class** Demo1ApplicationTests {  
  
 @Test  
 **void** contextLoads() {  
 }  
  
 *//空构造方法* @Test  
 **void** test1() {  
 Student student = **new** Student();  
 student.setStudentId(1);  
 student.setStudentName(**"周琦"**);  
 student.setGender(**"男"**);  
 student.setBirthday(**new** Date());  
 student.setEmail(**"zq67@sina.com"**);  
 System.***out***.println(**"student:id="**+student.getStudentId()+**",name="**+student.getStudentName()+  
 **",gender="**+student.getGender());  
 System.***out***.println(student.toString());  
 }  
  
 *//带参的构造方法* @Test  
 **void** test2() **throws** ParseException { *//使用SimpleDateFormat需要注意要抛出异常* SimpleDateFormat sdf = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 Date bdate = sdf.parse(**"2015-08-28"**);  
  
 *//使用SimpleDateFormat转换字符串为日期* Student student = **new** Student(2,**"张鹏"**,bdate,**"男"**,**"zhangpeng@sina.com"**);  
 System.***out***.println(student.toString());  
 }  
  
 *//带参的构造方法* @Test  
 **void** test3(){   
 *//使用自定义字符串转日期方法赋值* Student student = **new** Student(2,**"张鹏"**,*getDate*(**"1998-12-26"**),**"男"**,**"zhangpeng@sina.com"**);  
 System.***out***.println(student.toString());  
 }

*//自定义字符串转日期方法*

**public** Date getDate(String date){  
 DateFormat format1 = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 **try** {  
 **return** format1.parse(date);  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }

}

注：点击测试类方法名前的绿色小箭头，运行测试，可以看到结果会被输出到控制台中。

### 3.控制器方法

@GetMapping(**"/getstu"**)  
**public** String dispStudentInfo(Model model){  
  
 *//使用自定义字符串转日期方法赋值* Student student = **new** Student(2,**"张鹏"**,*getDate*(**"1998-12-26"**),**"男"**,**"zhangpeng@sina.com"**);  
 System.***out***.println(student.toString());  
  
 model.addAttribute(**"student"**,student);  
  
 **return "student"**;  
}

### 4.测试页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
  
 *<!--此句解决上下文路径不是/的情形，页面标签跳转、引用外部文件找不到文件路径的问题 -->  
 <!--http://127.0.0.1:8090/aa -->* <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>Title</**title**>  
</**head**>  
<**body**>  
<**div align="center"**>  
 <**div style="width**:500**px**;**text-align**: **left"**>  
 <**h1 style="text-align**: **center"**>学生信息</**h1**>  
 <**hr**/>  
 学生ID:<**label th:text="${student.studentId}"**></**label**><**br**/>  
 学生姓名:<**label th:text="${student.studentName}"**></**label**><**br**/>  
 学生性别:<**label th:text="${student.gender}"**></**label**><**br**/>  
 出生日期:<**label th:text="${#dates.format(student.birthday,'yyyy-MM-dd')}"**></**label**><**br**/>  
 出生日期:<**label style="color**:**blue" th:text="${#calendars.format(student.birthday,'yyyy-MM-dd')}"**></**label**><**br**/>  
 电子邮件:<**label th:text="${student.email}"**></**label**><**br**/>  
 </**div**>  
</**div**>  
</**body**>  
<**footer**>  
  
</**footer**>  
</**html**>

### 5.运行结果



### 6.端口设置

**server.port**=**8080**

### 7.上下文路径

**server.servlet.context-path**=**/aa**

### 8.日期格式显示处理

出生日期：<**label th:text="${#dates.format(user.birthday,'yyyy-MM-dd')}"**></**label**><**br**/>

### 9.@Component和@Controller的机制

由容器在系统启动时自动创建Bean

## 第十一章 thymeleaf 的th用法

### 1.th:action 表单提交的地址

<form action="subscribe.html" th:action="@{/subscribe}">

### 2.th:each 属性赋值

<tr th:each="user,userStat:${users}">

<**tr th:each="s:${listStudent}"**>

### 3.th:field 常用于表单字段绑定

<input type="text" value="" th:field="\*{username}"></input>

### 4.th:href 链接地址

<a th:href="@{/login}" th:unless=${session.user != null}>Login</a> />

### 5.th:id 替换

id <input th:id="'xxx' + ${collect.id}"/>

### 6.th:if 判断条件

<a th:if="${userId == collect.userId}" >

### 7.th:include 布局标签,替换内容到引入的文件

<head th:include="layout :: htmlhead" th:with="title='xx'"></head> />

### 8.th:fragment 布局标签,定义代码片段,其它引用

<div th:fragment="alert">

### 9.th:object 替换对象

<div th:object="${session.user}">

### 10.th:src 图片类地址引入

[![](@{/img/logo.png})](mailto:![](@{/img/logo.png}))

### 11.th:replace 布局标签,替换整个标签到引入的文件

<div th:replace="fragments/header :: title"></div>

### 12.th:text 文本替换

<p th:text="${collect.description}">description

### 13.th:value 属性赋值

<input th:value="${user.name}" />

### 14.th:inline 定义js脚本可以使用变量

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

### 15.th:remove 删除某个属性

<tr th:remove="all">

### 16.th:style 设置样式

th:style="'display:' + @{(${sitrue} ? 'none' : 'inline-block')} + ''"

### 17.th:onclick 点击事件

th:οnclick="'getCollect()'"

## 第十二章 thymeleaf内置组件方法介绍

### 1.日期组件#dates

${#dates.format(date,’yyyy-MM-dd’)} 显示年月日

${#dates.day(date)} 显示天

${#dates.month(date)} 显示月

${#dates.year(date)} 显示年

根据year,month,day创建日期(java.util.Date)对象，比如：

${#dates.create(year,month,day)} 例如：${#dates.create('2008','08','08')}

${#dates.create(year,month,day,hour,minute)}

${#dates.create(year,month,day,hour,minute,second)}

${#dates.create(year,month,day,hour,minute,second,millisecond)}

${#dates.createToday()}

### 2.日历组件#calendars

${#calendars.format(date,’yyyy-MM-dd’)} 显示年月日

${#calendars.createToday()}创建今天

注：日历组件的方法基本与日期组件相同

### 3.数值组件#numbers

//设置最小整数位数

${#numbers.formatInteger(num,3)}

//设置最小整数位数和千位分隔符:

'POINT'、'COMMA'、'WHITESPACE'、'NONE' 或 'DEFAULT'(根据本地化)

${#numbers.formatInteger(num,3,'POINT')}

//设置最小整数数字和(精确的)十进制数字。 \* 也同样适用于数组、列表或集合

${#numbers.formatDecimal(num,3,2)}

//创建一个从x到y的整数序列(数组)

${#numbers.sequence(from,to)}

${#numbers.sequence(from,to,step)}

### 4.字符组件#strings

//检查String是否为空(或null)。在检查之前执行trim()操作也同样适用于数组、列表或集合

${#strings.isEmpty(name)}

${#strings.arrayIsEmpty(nameArr)}

${#strings.listIsEmpty(nameList)}

${#strings.setIsEmpty(nameSet)}

//对字符串执行“isEmpty()”检查, 如果为false则返回它, 如果为true则默认为另一个指定的字符串。也同样适用于数组、列表或集合

${#strings.defaultString(text,default)}

${#strings.arrayDefaultString(textArr,default)}

${#strings.listDefaultString(textList,default)}

${#strings.setDefaultString(textSet,default)}

//检查字符串中是否包含片段，比如

${#strings.containsIgnoreCase(user.name,'kang')}

//也同样适用于数组、列表或集合

${#strings.contains(name,'ez')} // 也可以是 array\*、list\* 或 set\*

${#strings.containsIgnoreCase(name,'ez')} // 也可以是 array\*、list\* 或 set\*

//检查字符串是否以片段开始或结束,也同样适用于数组、列表或集合

${#strings.startsWith(name,'Don')} // 也可以是 array\*、list\* 或 set\*

${#strings.endsWith(name,endingFragment)} // 也可以是 array\*、list\* 或 set\*

//子串相关操作,也同样适用于数组、列表或集合

${#strings.indexOf(name,frag)} // 也可以是 array\*、list\* 或 set\*

${#strings.substring(name,3,5)} // 也可以是 array\*、list\* 或 set\*

${#strings.substringAfter(name,prefix)} // 也可以是 array\*、list\* 或 set\*

${#strings.substringBefore(name,suffix)} // 也可以是 array\*、list\* 或 set\*

${#strings.replace(name,'las','ler')} // 也可以是 array\*、list\* 或 set\*

//附加和前置,也同样适用于数组、列表或集合

${#strings.prepend(str,prefix)} // 也可以是 array\*、list\* 或 set\*

${#strings.append(str,suffix)} // 也可以是 array\*、list\* 或 set\*

//大小写转换,也同样适用于数组、列表或集合

${#strings.toUpperCase(name)} // 也可以是 array\*、list\* 或 set\*

${#strings.toLowerCase(name)} // 也可以是 array\*、list\* 或 set\*

//拆分和拼接

${#strings.arrayJoin(namesArray,',')}

${#strings.listJoin(namesList,',')}

${#strings.setJoin(namesSet,',')}

${#strings.arraySplit(namesStr,',')} // 返回String []

${#strings.listSplit(namesStr,',')} // 返回List<String>

${#strings.setSplit(namesStr,',')} // 返回Set<String>

//Trim 也同样适用于数组、列表或集合

${#strings.trim(str)} // 也可以是 array\*、list\* 或 set\*

//计算长度 \* 也同样适用于数组、列表或集合

${#strings.length(str)} // 也可以是 array\*、list\* 或 set\*

//缩写文本, 使其最大大小为n。如果文本较大, 它将被剪辑并在末尾附加“...” \* 也同样适用于数组、列表或集合

${#strings.abbreviate(str,10)} // 也可以是 array\*、list\* 或 set\*

//将第一个字符转换为大写(反之亦然)

${#strings.capitalize(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.unCapitalize(str)} // 也可以是 array\*、list\* 或 set\*

//将每个单词的第一个字符转换为大写

${#strings.capitalizeWords(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.capitalizeWords(str,delimiters)} // 也可以是 array\*、list\* 或 set\*

//转义字符串

${#strings.escapeXml(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.escapeJava(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.escapeJavaScript(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.unescapeJava(str)} // 也可以是 array\*、list\* 或 set\*

${#strings.unescapeJavaScript(str)} // 也可以是 array\*、list\* 或 set\*

//空安全比较和连接

${#strings.equals(first, second)}

${#strings.equalsIgnoreCase(first, second)}

${#strings.concat(values...)}

${#strings.concatReplaceNulls(nullValue, values...)}

随机数

${#strings.randomAlphanumeric(count)}

### 5.对象组件#objects

//当obj不为空时，返回obj，否则返回default默认值,其同样适用于数组、列表或集合

${#objects.nullSafe(obj,default)}

${#objects.arrayNullSafe(objArray,default)}

${#objects.listNullSafe(objList,default)}

${#objects.setNullSafe(objSet,default)}

### 6.逻辑组件#bools

//评估条件, 类似于 th:if 标签,也同样适用于数组、列表或集合

${#bools.isTrue(obj)}

${#bools.arrayIsTrue(objArray)}

${#bools.listIsTrue(objList)}

${#bools.setIsTrue(objSet)}

//用否定来评估条件,也同样适用于数组、列表或集合

${#bools.isFalse(cond)}

${#bools.arrayIsFalse(condArray)}

${#bools.listIsFalse(condList)}

${#bools.setIsFalse(condSet)}

### 7.数组组件#arrays

//转换为数组, 试图推断数组组件类。注意, 如果结果数组为空, 或者目标对象的元素不是全部相同的类, 则,此方法将返回Object []

${#arrays.toArray(object)}

//转换为指定组件类的数组。

${#arrays.toStringArray(object)}

${#arrays.toIntegerArray(object)}

${#arrays.toLongArray(object)}

${#arrays.toDoubleArray(object)}

${#arrays.toFloatArray(object)}

${#arrays.toBooleanArray(object)}

//计算数组长度

${#arrays.length(array)}

//检查数组是否为空

${#arrays.isEmpty(array)}

//检查数组中是否包含元素或元素集合

${#arrays.contains(array, element)}

${#arrays.containsAll(array, elements)}

### 列表组件#lists

//转化为 list

${#lists.toList(object)}

//计算大小

${#lists.size(list)}

//列表是否为空

${#lists.isEmpty(list)}

//检查list中是否包含元素或元素集合

${#lists.contains(list, element)}

${#lists.containsAll(list, elements)}

//排序给定列表的副本。列表的成员必须实现comparable, 或者必须定义comparator。

${#lists.sort(list)}

${#lists.sort(list, comparator)}

### 集合组件#sets

//转化为 to set

${#sets.toSet(object)}

//计算大小

${#sets.size(set)}

//检查set是否为empty

${#sets.isEmpty(set)}

//检查set中是否包含元素或元素集合

${#sets.contains(set, element)}

${#sets.containsAll(set, elements)}

### Map组件#maps

//计算大小

${#maps.size(map)}

//检查map是否为空

${#maps.isEmpty(map)}

//检查map中是否包含key/s或value/s

${#maps.containsKey(map, key)}

${#maps.containsAllKeys(map, keys)}

${#maps.containsValue(map, value)}

${#maps.containsAllValues(map, value)}

### 聚合组件#aggregates

//计算求和。如果数组（array） 或 集合（collection）是空（null）的话返回null

${#aggregates.sum(array)}

${#aggregates.sum(collection)}

//计算平均值。如果数组（array） 或 集合（collection）是空（null）的话返回null

${#aggregates.avg(array)}

${#aggregates.avg(collection)}

### 消息组件#messages

//获得外部化消息。可以接收一个键，一个键加上参数，或者一个数组/列表/一组键（在这种情况下，它将返回一个数组/列表/一组外部化消息）。如果未找到消息，则返回默认消息（如'??msgKey??'被返回）。

${#messages.msg('msgKey')}

${#messages.msg('msgKey', param1)}

${#messages.msg('msgKey', param1, param2)}

${#messages.msg('msgKey', param1, param2, param3)}

${#messages.msgWithParams('msgKey', new Object[] {param1, param2, param3, param4})}

${#messages.arrayMsg(messageKeyArray)}

${#messages.listMsg(messageKeyList)}

${#messages.setMsg(messageKeySet)}

//获得外部化消息或 null. 返回Null代替默认值

${#messages.msgOrNull('msgKey')}

${#messages.msgOrNull('msgKey', param1)}

${#messages.msgOrNull('msgKey', param1, param2)}

${#messages.msgOrNull('msgKey', param1, param2, param3)}

${#messages.msgOrNullWithParams('msgKey', new Object[] {param1, param2, param3, param4})}

${#messages.arrayMsgOrNull(messageKeyArray)}

${#messages.listMsgOrNull(messageKeyList)}

${#messages.setMsgOrNull(messageKeySet)}

### Id组件#ids

用于处理 id 可能重复（例如，由于迭代的结果）的属性的实用方法。可以使用ids对象的seq方法生成指定名称的递增id。

//一般使用在th:id属性中, 用于将计数器附加到id属性值，在迭代过程中，它始终是唯一的.

${#ids.seq('someId')}

//一般使用在label标签的th:for属性中, 以便这些标签可以参考由Ids 的#ids.seq(...)产生的值

${#ids.next('someId')}

${#ids.prev('someId')}

<**td class="text-center" th:text="${#ids.seq('')}"**></**td**>

## 第十三章 对象列表显示

### 1.控制器方法

@GetMapping(**"/getstus"**)  
**public** String listStudentInfo(Model model){  
  
 List<Student> listStudent= **new** ArrayList<>();  
 listStudent.add(**new** Student(1,**"周琦"**,*getDate*(**"1998-12-28"**),**"男"**,**"zq67@sina.com"**));  
 listStudent.add(**new** Student(2,**"张鹏"**,*getDate*(**"1998-12-26"**),**"男"**,**"zhangpeng@sina.com"**));  
 listStudent.add(**new** Student(3,**"张书豪"**,*getDate*(**"1998-12-26"**),**"男"**,**"zhangsh@sina.com"**));  
 listStudent.add(**new** Student(4,**"杨旭升"**,*getDate*(**"1998-12-26"**),**"男"**,**"yangxs@sina.com"**));  
 listStudent.add(**new** Student(5,**"刘湛"**,*getDate*(**"1998-12-26"**),**"男"**,**"liuz@sina.com"**));  
 listStudent.add(**new** Student(5,**"马丽荣"**,*getDate*(**"1998-12-26"**),**"男"**,**"malr@sina.com"**));  
  
 model.addAttribute(**"listStudent"**,listStudent);  
  
 **return "liststudent"**;  
}

### 2.测试页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>用户列表</**title**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.css}"**/>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
</**head**>  
<**body**>  
  
 <**div style="width**:900**px**;**margin**:100**"**>  
 <**table class="table table-striped table-bordered table-hover table-set"**>  
 <**thead**>  
 <**tr**>  
 <**td class="center col-md-1"**>用户名称</**td**>  
 <**td class="center col-md-2"**>姓名</**td**>  
 <**td class="center col-md-1"**>性别</**td**>  
 <**td class="center col-md-3"**>出生日期</**td**>  
 <**td class="center col-md-3"**>是否已婚</**td**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr th:each="u:${listUser}"**>  
 <**td class="center"**><**span th:text="${u.name}"**></**td**>  
 <**td class="center"**><**span th:text="${u.firstName+u.lastName}"**></**span**></**td**>  
 <**td class="center"**><**span th:text="${u.gender}"**></**span**></**td**>  
 <**td class="left"**><**span th:text="${#calendars.format(u.birthday,'yyyy-MM-dd')}"**></**span**></**td**>  
 <**td class="center"**><**span th:text="${u.married}"**></**span**></**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
</**div**>  
</**body**>  
</**html**>

### 3.管理页面的按钮

上述页面中，我们想在最后一列增加操作列，此列放置“修改、删除”按钮，在需要修改当前行时点击这些按钮，可以进行数据操作。

### 4.引入行序号列

**使用th:text="${#ids.seq('')}"**生成每页显示数据行的序号，**#ids**组件是thymeleaf的内置组件

### 5.引入CSS渲染框架bootstrap和ace

在页面文件的<head></head>中加载js脚本和css样式文件：

<**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.css}"** />  
<**link rel="stylesheet" type="text/css" th:href="@{/font-awesome/css/font-awesome.min.css}"** />  
<**link rel="stylesheet" type="text/css" th:href="@{/css/ace.min.css}"** />  
<**script th:src="@{/js/jquery.min.js}"**></**script**>  
<**script th:src="@{/js/bootstrap.min.js}"**></**script**>

1. 按钮

<**button type="button" class="btn1 btn-info btn-sm btn-corner" title="修改"**>修改</**button**>  
<**button type="button" class="btn1 btn-danger btn-sm btn-corner" title="删除"**>删除</**button**>

（2）使用<a>标签+ace样式模拟按钮

<**a class="btn btn-info btn-sm"** >  
 <**i class="fa fa-pencil smaller-30"**>修改</**i**>  
</**a**>  
<**a class="btn btn-danger btn-sm"** >  
 <**i class="fa fa-times smaller-30"**>删除</**i**>  
</**a**>

### 6.修改后的页面代码

liststudent.html：

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 *<!--此句解决上下文路径不是/的情形，页面标签跳转、引用外部文件找不到文件路径的问题 -->  
 <!--http://127.0.0.1:8090/aa -->* <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>学生信息管理</**title**>  
  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.css}"** />  
 <**link rel="stylesheet" type="text/css" th:href="@{/font-awesome/css/font-awesome.min.css}"** />  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/ace.min.css}"** />  
 <**script th:src="@{/js/jquery.min.js}"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
  
</**head**>  
<**body**>  
<**div align="center"** >  
 <**table class="col-md-7 table table-sm table-striped table-bordered table-hover table-condensed"** >  
 <**thead**>  
 <**tr class="text-center"**>  
 <**th class="col-md-1"**>NO</**th**>  
 <**th class="col-md-1"**>ID</**th**>  
 <**th class="col-md-1"**>姓名</**th**>  
 <**th class="col-md-1"**>性别</**th**>  
 <**th class="col-md-2"**>出生日期</**th**>  
 <**th class="col-md-2"**>电子邮件</**th**>  
 <**th class="col-md-2"**>操作</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr th:each="s:${listStudent}"**>  
 <**td class="text-center" th:text="${#ids.seq('')}"**></**td**>  
 <**td class="text-center" th:text="${s.studentId}"**>1</**td**>  
 <**td th:text="${s.studentName}"**>李四</**td**>  
 <**td class="text-center" th:text="${s.gender}"**>男</**td**>  
 <**td class="text-center" th:text="${#dates.format(s.birthday,'yyyy-MM-dd')}"**>2000-11-12</**td**>  
 <**td th:text="${s.email}"**>lisi@sina.com</**td**>  
 <**td class="text-center"**>  
 <**button type="button" class="btn1 btn-info btn-sm btn-corner" title="修改"**>修改</**button**>  
 <**button type="button" class="btn1 btn-danger btn-sm btn-corner" title="删除"**>删除</**button**>  
  
 <**a class="btn btn-info btn-sm"** >  
 <**i class="fa fa-pencil smaller-30"**>修改</**i**>  
 </**a**>  
 <**a class="btn btn-danger btn-sm"** >  
 <**i class="fa fa-times smaller-30"**>删除</**i**>  
 </**a**>  
 </**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
</**div**>  
</**body**>  
</**html**>

注：<**td class="text-center" th:text="${#ids.seq('')}"**></**td**>

其中${#ids.seq('')}是用来生成每页显示数据行的序号

### 7.运行访问

http://127.0.0.1:8080/aa/getstus



### 字根图标的用法

1. 访问网站http://www.fontawesome.com.cn/
2. 下载字根图标文件，放置到项目的static目录下
3. 在页面引用font-awesome.min.css

<**link rel="stylesheet" type="text/css"**

**th:href="@{/font-awesome/css/font-awesome.min.css}"** />

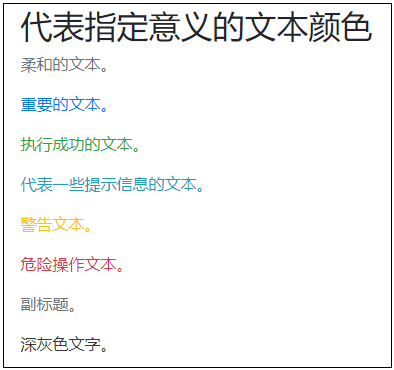
1. 选择字根图标，点开，拷贝字根图标引用类，粘贴到标签的类引用中



## BootStrap渲染框架

### 1.文本颜色

<**div class="container"**>  
 <**h2**>代表指定意义的文本颜色</**h2**>  
 <**p class="text-muted"**>柔和的文本。</**p**>  
 <**p class="text-primary"**>重要的文本。</**p**>  
 <**p class="text-success"**>执行成功的文本。</**p**>  
 <**p class="text-info"**>代表一些提示信息的文本。</**p**>  
 <**p class="text-warning"**>警告文本。</**p**>  
 <**p class="text-danger"**>危险操作文本。</**p**>  
 <**p class="text-secondary"**>副标题。</**p**>  
 <**p class="text-dark"**>深灰色文字。</**p**>  
 <**p class="text-light"**>浅灰色文本（白色背景上看不清楚）。</**p**>  
 <**p class="text-white"**>白色文本（白色背景上看不清楚）。</**p**>  
</**div**>



### 2.网格系统

（1）视口（viewport）

Bootstrap 提供了一套响应式、移动设备优先的流式网格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列。

响应式网格系统随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | | | | 4 | | | | 4 | | | |
| 4 | | | | 8 | | | | | | | |
| 6 | | | | | | 6 | | | | | |
| 12 | | | | | | | | | | | |

（2）表格宽度设置

单元格设置宽度：col-xs-1、col-xs-2、col-xs-3、col-xs-4...col-xs-12

col-md-1、col-md-2、col-md-3、col-md-4...col-md-12

整体表格设置宽度:(单元格1宽度+单元格2宽度+单元格3宽度+.。。+单元格n宽度

### 3.按钮

<!-- 标准的按钮 -->

<button type="button" class="btn btn-default">默认按钮</button>

<!-- 提供额外的视觉效果，标识一组按钮中的原始动作 -->

<button type="button" class="btn btn-primary">原始按钮</button>

<!-- 表示一个成功的或积极的动作 -->

<button type="button" class="btn btn-success">成功按钮</button>

<!-- 信息警告消息的上下文按钮 -->

<button type="button" class="btn btn-info">信息按钮</button>

<!-- 表示应谨慎采取的动作 -->

<button type="button" class="btn btn-warning">警告按钮</button>

<!-- 表示一个危险的或潜在的负面动作 -->

<button type="button" class="btn btn-danger">危险按钮</button>

<!-- 并不强调是一个按钮，看起来像一个链接，但同时保持按钮的行为 -->

<button type="button" class="btn btn-link">链接按钮</button>

按钮如下图：



### 4.按钮组

<**div class="btn-group"**>  
 <**button type="button" class="btn btn-default"**>按钮 1</**button**>  
 <**button type="button" class="btn btn-primary"**>按钮 2</**button**>  
 <**button type="button" class="btn btn-info"**>按钮 3</**button**>  
 <**button type="button" class="btn btn-warning"**>按钮 4</**button**>  
</**div**>

### 5.图片

<**img th:src="@{/images/photos/user5.png}" class="rounded" alt="Cinque Terre"**>  
<**img th:src="@{/images/photos/user5.png}" class="rounded-circle" alt="Cinque Terre"**>  
<**img th:src="@{/images/photos/user5.png}" class="img-thumbnail" alt="Cinque Terre"**>

### 6.表格的table类

|  |  |
| --- | --- |
| **类** | **描述** |
| .table | 为任意 <table> 添加基本样式 (只有横向分隔线) |
| .table-striped | 在 <tbody> 内添加斑马线形式的条纹 ( IE8 不支持) |
| .table-bordered | 为所有表格的单元格添加边框 |
| .table-hover | 在 <tbody> 内的任一行启用鼠标悬停状态 |
| .table-condensed | 让表格更加紧凑 |
| .table-dark | 可以为表格添加黑色背景 |
| .table-borderless | 可以设置一个无边框的表格 |
| .table-primary | 蓝色: 指定这是一个重要的操作 |
| .table-success | 绿色: 指定这是一个允许执行的操作 |
| .table-danger | 红色: 指定这是可以危险的操作 |
| .table-info | 浅蓝色: 表示内容已变更 |
| .table-warning | 橘色: 表示需要注意的操作 |
| .table-active | 灰色: 用于鼠标悬停效果 |
| .table-secondary | 浅灰色，可以是表格行的背景 |
| .table-light | 深灰色，可以是表格行的背景 |

### 7.表单

<form id="aa">

<div class="form-group">

<label for="name">名称</label>

<input type="text" class="form-control" id="name"

placeholder="请输入名称">

</div>

<div class="form-group">

<label for="inputfile">文件输入</label>

<input type="file" id="inputfile">

<p class="help-block">这里是块级帮助文本的实例。</p>

</div>

<div class="checkbox">

<label>

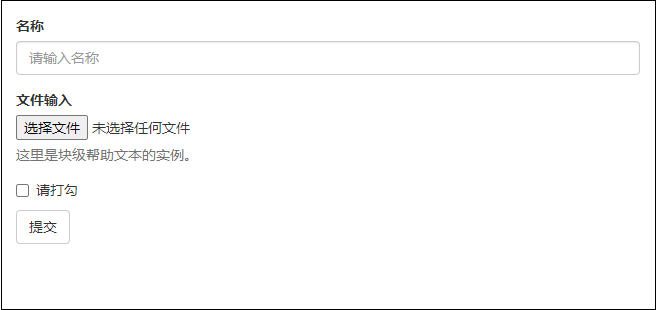
<input type="checkbox"> 请打勾

</label>

</div>

<button type="submit" class="btn btn-default">提交</button>

</form>

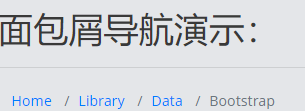


表单组件示例：

<**div class="col-md-5" align="center"**>  
<**div class="form-group"**>  
 <**label for="usr"**>用户名:</**label**>  
 <**input type="text" class="form-control" id="usr"**>  
</**div**><**br**/>  
<**div class="form-group"**>  
 <**label for="pwd"**>密码:</**label**>  
 <**input type="password" class="form-control" id="pwd"**>  
</**div**><**br**/>  
<**div class="form-group"**>  
 <**label for="comment"**>评论:</**label**>  
 <**textarea class="form-control" rows="5" id="comment"**></**textarea**>  
</**div**><**br**/>  
<**div class="form-check"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value=""**>Option 1  
 </**label**>  
</**div**><**br**/>  
<**div class="form-check"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value=""**>Option 2  
 </**label**>  
</**div**>  
<**div class="form-check disabled"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value="" disabled**>Option 3  
 </**label**>  
</**div**><**br**/>  
<**div class="form-check form-check-inline"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value=""**>Option 1  
 </**label**>  
</**div**>  
<**div class="form-check form-check-inline"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value=""**>Option 2  
 </**label**>  
</**div**>  
<**div class="form-check form-check-inline disabled"**>  
 <**label class="form-check-label"**>  
 <**input type="checkbox" class="form-check-input" value="" disabled**>Option 3  
 </**label**>  
</**div**><**br**/>  
<**div class="radio"**>  
 <**label**><**input type="radio" name="optradio"**>Option 1</**label**>  
</**div**>  
<**div class="radio"**>  
 <**label**><**input type="radio" name="optradio"**>Option 2</**label**>  
</**div**>  
<**div class="radio disabled"**>  
 <**label**><**input type="radio" name="optradio" disabled**>Option 3</**label**>  
</**div**><**br**/>  
<**label class="radio-inline"**><**input type="radio" name="optradio"**>Option 1</**label**>  
<**label class="radio-inline"**><**input type="radio" name="optradio"**>Option 2</**label**>  
<**label class="radio-inline"**><**input type="radio" name="optradio" disabled**>Option 3</**label**><**br**/>  
<**div class="form-group"**>  
 <**label for="sel1"**>下拉菜单:</**label**>  
 <**select class="form-control" id="sel1"**>  
 <**option**>1</**option**>  
 <**option**>2</**option**>  
 <**option**>3</**option**>  
 <**option**>4</**option**>  
 </**select**>  
</**div**><**br**/>  
</**div**>

### 8.面包屑导航

<**nav class="breadcrumb"**>  
 <**a class="breadcrumb-item" href="#"**>Home</**a**>  
 <**a class="breadcrumb-item" href="#"**>Library</**a**>  
 <**a class="breadcrumb-item" href="#"**>Data</**a**>  
 <**span class="breadcrumb-item active"**>Bootstrap</**span**>  
</**nav**>



### 9.分页

<**div class="container"**>  
 <**h2**>当前页页码状态</**h2**>  
 <**p**>当前页可以使用 .active 类来高亮显示：</**p**>  
 <**ul class="pagination"**>  
 <**li class="page-item"**><**a class="page-link" href="#"**>Previous</**a**></**li**>  
 <**li class="page-item"**><**a class="page-link" href="#"**>1</**a**></**li**>  
 <**li class="page-item active"**><**a class="page-link" href="#"**>2</**a**></**li**>  
 <**li class="page-item"**><**a class="page-link" href="#"**>3</**a**></**li**>  
 <**li class="page-item"**><**a class="page-link" href="#"**>Next</**a**></**li**>  
 </**ul**>  
</**div**>



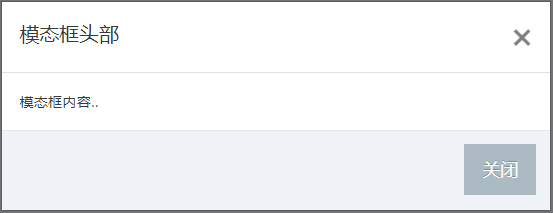
### 10.加载

<**div class="container"**>  
 <**h2**>闪烁的加载效果</**h2**>  
 <**p**>使用 .spinner-grow 类来设置闪烁的加载效果:</**p**>  
  
 <**div class="spinner-grow text-muted"**></**div**>  
 <**div class="spinner-grow text-primary"**></**div**>  
 <**div class="spinner-grow text-success"**></**div**>  
 <**div class="spinner-grow text-info"**></**div**>  
 <**div class="spinner-grow text-warning"**></**div**>  
 <**div class="spinner-grow text-danger"**></**div**>  
 <**div class="spinner-grow text-secondary"**></**div**>  
 <**div class="spinner-grow text-dark"**></**div**>  
 <**div class="spinner-grow text-light"**></**div**>  
</**div**>



### 11.模态窗口

*<!-- 模态框 -->*<**div class="modal fade" id="myModal"**>  
 <**div class="modal-dialog"**>  
 <**div class="modal-content"**>  
  
 *<!-- 模态框头部 -->* <**div class="modal-header"**>  
 <**h4 class="modal-title"**>模态框头部</**h4**>  
 <**button type="button" class="close" data-dismiss="modal"**>**&times;**</**button**>  
 </**div**>  
  
 *<!-- 模态框主体 -->* <**div class="modal-body"**>  
 模态框内容..  
 </**div**>  
  
 *<!-- 模态框底部 -->* <**div class="modal-footer"**>  
 <**button type="button" class="btn btn-secondary" data-dismiss="modal"**>关闭</**button**>  
 </**div**>  
  
 </**div**>  
 </**div**>  
</**div**>



### 12.提示窗口

<**div**>  
 <**h1**>提示窗演示：</**h1**><**hr**/>  
 <**a href="#" data-toggle="tooltip" data-placement="top" title="我是提示内容!"**>鼠标移动到我这</**a**>  
 <**a href="#" data-toggle="tooltip" data-placement="bottom" title="我是提示内容!"**>鼠标移动到我这</**a**>  
 <**a href="#" data-toggle="tooltip" data-placement="left" title="我是提示内容!"**>鼠标移动到我这</**a**>  
 <**a href="#" data-toggle="tooltip" data-placement="right" title="我是提示内容!"**>鼠标移动到我这</**a**>  
</**div**>

### 13.Bootstrap辅助类

（1）文本

以下不同的类展示了不同的文本颜色。如果文本是个链接鼠标移动到文本上会变暗：

|  |  |
| --- | --- |
| **类** | **描述** |
| .text-muted | "text-muted" 类的文本样式 |
| .text-primary | "text-primary" 类的文本样式 |
| .text-success | "text-success" 类的文本样式 |
| .text-info | "text-info" 类的文本样式 |
| .text-warning | "text-warning" 类的文本样式 |
| .text-danger | "text-danger" 类的文本样式 |

（2）背景

以下不同的类展示了不同的背景颜色。 如果文本是个链接鼠标移动到文本上会变暗：

|  |  |
| --- | --- |
| **类** | **描述** |
| .bg-primary | 表格单元格使用了 "bg-primary" 类 |
| .bg-success | 表格单元格使用了 "bg-success" 类 |
| .bg-info | 表格单元格使用了 "bg-info" 类 |
| .bg-warning | 表格单元格使用了 "bg-warning" 类 |
| .bg-danger | 表格单元格使用了 "bg-danger" 类 |

（3）其他

|  |  |
| --- | --- |
| **类** | **描述** |
| .pull-left | 元素浮动到左边 |
| .pull-right | 元素浮动到右边 |
| .center-block | 设置元素为 display:block 并居中显示 |
| .clearfix | 清除浮动 |
| .show | 强制元素显示 |
| .hidden | 强制元素隐藏 |
| .sr-only | 除了屏幕阅读器外，其他设备上隐藏元素 |
| .sr-only-focusable | 与 .sr-only 类结合使用，在元素获取焦点时显示(如：键盘操作的用户) |
| .text-hide | 将页面元素所包含的文本内容替换为背景图 |
| .close | 显示关闭按钮 |
| .caret | 显示下拉式功能 |

## 集成Swagger3

在前后端分离的项目开发中，撰写维护接口文档是个重要但有非常繁琐的工作。与手动编写接口文档不同，swagger是一个自动生成接口文档的工具，在需求不断变更的环境下，随着程序的编译会自动生成接口文档。

### pom文件中引入依赖

*<!--支持swagger3-->*<**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-boot-starter</**artifactId**>  
 <**version**>3.0.0</**version**>  
</**dependency**>  
*<!--支持swagger3 UI-->*<**dependency**>  
 <**groupId**>com.github.xiaoymin</**groupId**>  
 <**artifactId**>swagger-bootstrap-ui</**artifactId**>  
 <**version**>1.9.6</**version**>  
</**dependency**>

swagger-bootstrap-ui是一个swagger3的中文界面插件

注：要降低版本springboot2.6.4--->springboot2.5.2

### 2.使用账号和密码访问API

（1）在pom.xml中添加依赖,目的是为了访问接口文档时需要用户密码验证

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

（2）修改配置application-dev.properties

*# 配置用户名***spring.security.user.name**=**admin***#密码***spring.security.user.password**=**123456**

（3）创建一个配置类ActuatorWebSecurityConfigurationAdapter

@Configuration  
@EnableWebSecurity  
**public class** ActuatorWebSecurityConfigurationAdapter **extends** WebSecurityConfigurerAdapter {  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http  
 .authorizeRequests()  
 *//普通的接口不需要校验* .antMatchers(**"/user/\*\*"**).permitAll()  
 *// swagger页面需要添加登录校验* .antMatchers(**"/swagger-ui/index.html"**).authenticated()  
 .and()  
 .formLogin();  
 }  
}

### 3.创建一个配置类SwaggerUiWebMvcConfigurer

注：这是Swagger3在SpringBoot2.6.0以上版本必须的

**public class** SwaggerUiWebMvcConfigurer **implements** WebMvcConfigurer {  
 **private final** String **baseUrl**;  
  
 **public** SwaggerUiWebMvcConfigurer(String baseUrl) {  
 **this**.**baseUrl** = baseUrl;  
 }  
  
 @Override  
 **public void** addResourceHandlers(ResourceHandlerRegistry registry) {  
 String baseUrl = StringUtils.*trimTrailingCharacter*(**this**.**baseUrl**, **'/'**);  
 registry.  
 addResourceHandler(baseUrl + **"/swagger-ui/\*\*"**)  
 .addResourceLocations(**"classpath:/META-INF/resources/webjars/springfox-swagger-ui/"**)  
 .resourceChain(**false**);  
 }  
  
 @Override  
 **public void** addViewControllers(ViewControllerRegistry registry) {  
 registry.addViewController(**baseUrl** + **"/swagger-ui/"**)  
 .setViewName(**"forward:"** + **baseUrl** + **"/swagger-ui/index.html"**);  
 }  
}

### 4.创建一个配置类SwaggerConfig

@Configuration  
@EnableOpenApi  
**public class** SwaggerConfig {  
 @Bean  
 **public** Docket createRestApi() {  
 **return new** Docket(DocumentationType.***OAS\_30***)  
 .apiInfo(apiInfo())  
 .select()  
 .apis(RequestHandlerSelectors.*withMethodAnnotation*(ApiOperation.**class**))  
 .paths(PathSelectors.*any*())  
 .build();  
 }  
  
 **private** ApiInfo apiInfo() {  
 **return new** ApiInfoBuilder()  
 .title(**"Swagger3接口文档"**)  
 .description(**"Springboot 业务层"**)  
 .contact(**new** Contact(**"ZhouQi"**, **"http://www.5starsoft.com.cn"**, **"zq67@sina.com"**))  
 .version(**"1.0"**)  
 .build();  
 }  
}

### 5.Swagger3注解

* @Api()用于类名
* @ApiOperation()用于方法名
* @ApiParam()用于参数说明
* @ApiModel()用于实体类
* @ApiModelProperty用于实体类属性

以下红色字体为Swagger3注解使用例子

@Controller  
@Api(tags = **"欢迎信息管理"**)  
**public class** WelcomeController {  
 @ApiOperation(value = **"打招呼"**, notes = **"访问请求http://localhost:8080/aa/admin/sayhello"**)  
 @RequestMapping(value = **"/admin/sayhello"**, method = RequestMethod.***GET***) *//路径,如果不写method，即包含了GET、POST两种方法* @ResponseBody *//返回的是JSON格式字符串* **public** String sayHello() {  
 **return "欢迎你来到SpringBoot世界！"**;  
 }  
  
 *//路径别名* @ApiOperation(value = **"打招呼1"**, notes = **"访问请求http://localhost:8080/aa/admin/sayhello1"**)  
 @GetMapping({**"/admin/sayhello1"**, **"/admin/say"**, **"/admin/"**}) *//路径,如果不写method，即包含了GET、POST两种方法* @ResponseBody *//返回的是JSON格式字符串* **public** String sayHello1() {  
 **return "欢迎你来到SpringBoot世界1！"**;  
 }  
  
 *//路径别名* @ApiOperation(value = **"打招呼2"**, notes = **"访问请求http://localhost:8080/aa/admin/say2"**)  
 @GetMapping({**"/admin/sayhello2"**, **"/admin/say2"**, **"/admin/"**}) *//路径,如果不写method，包含了GET方法  
 //返回的是JSON格式字符串* **public** @ResponseBody  
 String sayHello2() {  
 **return "欢迎你来到SpringBoot世界12"**;  
 }  
  
 @ApiOperation(value = **"欢迎"**, notes = **"访问请求http://localhost:8080/aa/admin/welc"**)  
 @GetMapping({**"/admin/welc"**})  
 **public** String welcomePage(Model msg) {  
 msg.addAttribute(**"who"**, **"周琦"**);  
 msg.addAttribute(**"who1"**, **"王强"**);  
 msg.addAttribute(**"who2"**, **"李雯"**);  
 **return "welcome"**; *//字符串 -- 页面对象* }  
  
 @ApiOperation(value = **"欢迎b"**, notes = **"访问请求http://localhost:8080/aa/admin/welcb"**)  
 @GetMapping({**"/admin/welcb"**})  
 **public** String welcomeBootStrapPage(Model msg) {  
 msg.addAttribute(**"who"**, **"周琦"**);  
 msg.addAttribute(**"who1"**, **"王强"**);  
 msg.addAttribute(**"who2"**, **"李雯"**);  
 **return "welcome-bootstrap"**; *//字符串 -- 页面对象* }  
  
 @ApiOperation(value = **"bootstrap测试"**, notes = **"访问请求http://localhost:8080/aa/admin/bt"**)  
 @GetMapping({**"/admin/bt"**})  
 **public** String testBootStrapPage() {  
 **return "bootstraptest"**; *//字符串 -- 页面对象* }  
  
 @GetMapping({**"/admin/hel1"**})  
 **public** String hello1Page() {  
 **return "hello1"**;  
 }  
  
 @GetMapping({**"/admin/hel2"**})  
 **public** String hello2Page() {  
 **return "hello2"**;  
 }  
  
 @GetMapping({**"/admin/hel3"**})  
 **public** String hello3Page() {  
 **return "hello3"**;  
 }  
  
 @ApiOperation(value = **"获得路径参数1"**, notes = **"访问请求http://localhost:8080/aa/admin/getp1?name=xxx&&password=yyy"**)  
 @ApiImplicitParams({  
 @ApiImplicitParam(name = **"name"**,value = **"姓名"**,required = **true**,paramType = **"path"**,dataType = **"String"**,dataTypeClass = String.**class**),  
 @ApiImplicitParam(name = **"password"**,value = **"口令"**,required = **true**,paramType = **"path"**,dataType = **"String"**,dataTypeClass = String.**class**)  
 })  
 @GetMapping(**"/admin/getp1"**)  
 **public** @ResponseBody String getPathParam1(@RequestParam(**"name"**) String name,@RequestParam(value=**"password"**,defaultValue=**"000000"**,required=**false**) String password){  
 System.***out***.println(**"name:"**+name+**",password:"**+password);  
 **return "name:"**+name+**",password:"**+password;  
 }  
  
 @ApiOperation(value = **"获得路径参数2"**, notes = **"访问请求http://localhost:8080/aa/admin/getp2?name=xxx&&password=yyy"**)  
 @ApiImplicitParams({  
 @ApiImplicitParam(name = **"name"**,value = **"姓名"**,required = **true**,paramType = **"path"**,dataType = **"String"**,dataTypeClass = String.**class**),  
 @ApiImplicitParam(name = **"password"**,value = **"口令"**,required = **true**,paramType = **"path"**,dataType = **"String"**,dataTypeClass = String.**class**)  
 })  
 @GetMapping(**"/admin/getp2"**)  
 **public** @ResponseBody String getPathParam2(String name,String password){ *//自动绑定机制* System.***out***.println(**"name:"**+name+**",password:"**+password);  
 **return "name:"**+name+**",password:"**+password;  
 }  
  
  
 @ApiOperation(value = **"获得路径参数3"**, notes = **"访问请求http://localhost:8080/aa/admin/getp3?name=xxx&&password=yyy"**)  
 @ApiImplicitParams({  
 @ApiImplicitParam(name = **"name"**,value = **"姓名"**,required = **true**,paramType = **"path"**,dataType = **"String"**,dataTypeClass = String.**class**),  
 })  
 @GetMapping(**"/admin/getp3"**)  
 **public** @ResponseBody String getPathParam3(HttpServletRequest request){ *//自动绑定机制  
 //String name = (String)request.getSession().getAttribute("name");* String name = request.getParameter(**"name"**);  
 *//String[] name1= request.getParameterValues("name");* System.***out***.println(**"name:"**+name);  
  
 **return "name:"**+name;  
 }  
  
 @ApiOperation(value = **"获得上下文路径"**, notes = **"访问请求http://localhost:8080/aa/admin/getct"**)  
 *//上下文路径获取* @GetMapping(**"/admin/getct"**)  
 **public** @ResponseBody String getContextPath(HttpServletRequest request){  
 String context = request.getContextPath();  
 String requestURI = request.getRequestURI();  
 **return "上下文路径:"**+context+**",访问请求:"**+requestURI;  
 }  
}

@ApiImplicitParam注解中参数类型说明：  
1. name ：参数名。   
2. value ： 参数的具体意义，作用。   
3. required ： 参数是否必填。   
4. dataType ：参数的数据类型。   
5. paramType ：查询参数类型，这里有几种形式：

* path 以地址的形式提交数据
* query 直接跟参数完成自动映射赋值
* body 以流的形式提交 仅支持POST
* header 参数在request headers 里边提交
* form 以form表单的形式提交 仅支持POST

1. dataTypeClass = String.**class //参数值是字符串**

dataTypeClass = Integer.**class //参数值是整数**

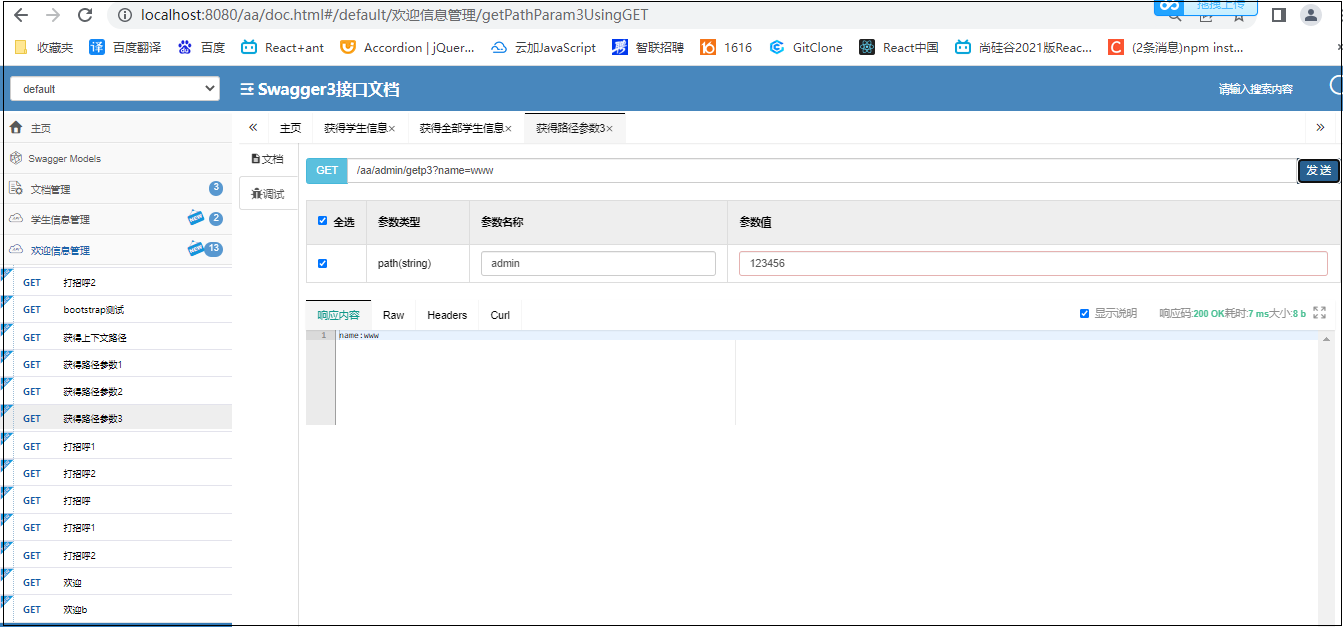
### 编译运行

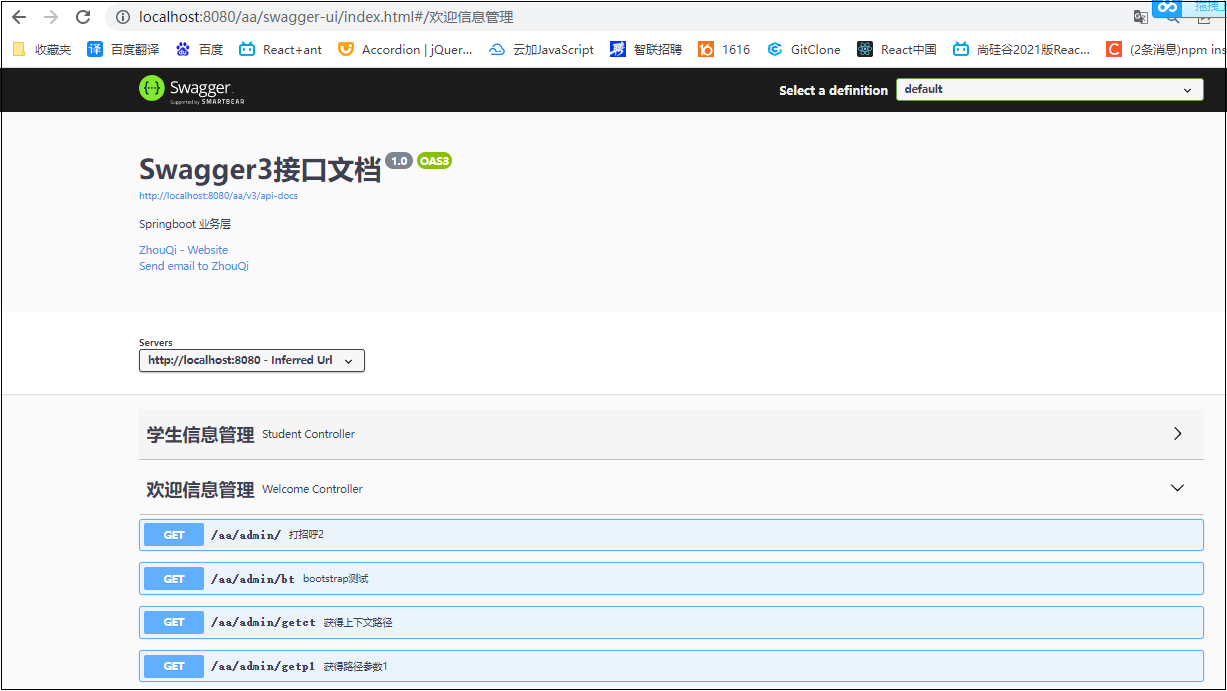
由于SpringBoot2.6.0以后出现于Swagger3的不兼容，在编译时先降SpringBoot版本2.5.x，编译后再改回来，重编译一次，就能正常访问接口文档了。

访问地址：

<http://localhost:8080/aa/doc.html>

http://localhost:8080/aa/swagger-ui/index.html





## 第十六章 模板合成

### 前端页面经常有些东西每个页面都有，每次都重复，而且页面风格、布局要求统一化，一旦修改，每个页面都要改一遍，工作量实在太大，有没有什么好办法能解决这个问题呢？当然有，那就是使用模板合成技术，这也是一个成熟的前端框架必须具有的技术。

### 1.复制项目

（1）拷贝目录demo1,将目录改名为demo2

（2）将pom.xml打开，将demo1改名为demo2,2处

（3）点开src\main\java\com\fd\springboot，将Demo1Application改为Demo2Application，文件里头还有两处

（4）点开src\test\java\com\fd\springboot，Demo1ApplicationTests改为

Demo2ApplicationTests，点开里头还有一处

（5）将项目配置文件中端口号改成新的端口

（6）加载项目（整体code项目是最简单的java项目，不能是spring项目，否则加载模块会出错）

### 1.主控页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
  
</**head**>  
<**body**>  
 <**div th:include="/test/header::header"**></**div**>  
  
 <**h1**>html传统写法</**h1**><**hr**/>  
 <**a href="/index1"**>第一页</**a**><**br**/>  
 <**a href="/index2"**>第二页</**a**><**br**/>  
  
 <**h1**>thymeleaf标准写法</**h1**><**hr**/>  
 <**a th:href="@{/index1}"**>第一页</**a**><**br**/>  
 <**a th:href="@{/index2}"**>第二页</**a**><**br**/>  
 <**h1 th:text="${message}"**>世界欢迎你</**h1**>  
 <**audio th:src="@{/audio/沙漠骆驼.mp3}" controls**></**audio**><**br**/>  
 <**img th:src="@{/image/fj1.jpg}"** />  
  
  
 <**div th:include="/test/ footer::footer"**></**div**>  
</**body**>  
</**html**>

### 2.部件模板

头部部件：

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>页头</**title**>  
  
  
</**head**>  
<**body**>  
 <**div th:fragment="header"** >  
 <**link rel="stylesheet" th:href="@{/css/mycss.css}"**>  
 <**h1 class="m1"**>信息技术学院</**h1**>  
 </**div**>  
  
</**body**>  
</**html**>

脚部部件：

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
</**head**>  
<**body**>  
<**div th:fragment="footer"** >  
 <**link rel="stylesheet" th:href="@{/css/mycss.css}"**>  
 <**h1 class="m1"**>宁夏工商学院 **&copy;** 版权所有</**h1**>  
</**div**>  
</**body**>  
</**html**>

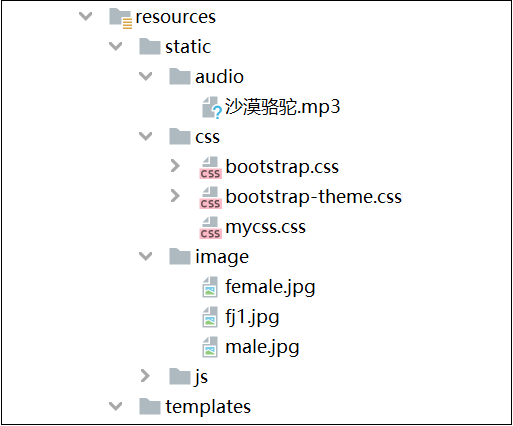
### 3.合并部件

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>页头</**title**>  
  
  
</**head**>  
<**body**>  
 <**div th:fragment="header"** >  
 <**link rel="stylesheet" th:href="@{/css/mycss.css}"**>  
 <**h1 class="m1"**>信息技术学院</**h1**>  
 </**div**>  
  
 <**div th:fragment="footer"** >  
 <**link rel="stylesheet" th:href="@{/css/mycss.css}"**>  
 <**h1 class="m1"**>宁夏工商学院 **&copy;** 版权所有</**h1**>  
 </**div**>  
</**body**>  
</**html**>

### 4.背景图片显示和播放音乐文件

<**audio th:src="@{/audio/沙漠骆驼.mp3}" controls**></**audio**><**br**/>  
<**img th:src="@{/image/fj1.jpg}"** />

### 5.静态资源存放位置

****

## 第十七章 集成lombok

集成lombok的目的是为了省略get、set等固定方法的书写，Lombok本质上就是一个实现了“JSR 269 API”的程序。

### 1.引入依赖

*<!--集成lombok-->*<**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
</**dependency**>

注：

@AllArgsConstructor *//包含所有字段的构造方法*@NoArgsConstructor *//无参构造方法*@Data *//get/set方法*@ToString *//生成tostring方法*

### 2.POJO/Entity类

@AllArgsConstructor *//包含所有字段的构造方法*@NoArgsConstructor *//无参构造方法*@Data *//get/set方法*@ToString *//生成tostring方法***public class** Teacher extends BaseEntity {  
 **private** Integer **teacher\_id**;  
 **private** String **teacher\_name**;  
 **private** String **gender**;  
 **private** Date **birthday**;  
 **private** Boolean **isMarried**;  
}

### 3.继承BaseEntity类

@AllArgsConstructor *//包含所有字段的构造方法*@NoArgsConstructor *//无参构造方法*@Data *//get/set方法*@ToString *//生成tostring方法*  
**public class** BaseEntity **implements** Serializable {  
 @DateTimeFormat(pattern=**"yyyy-MM-dd HH:mm:ss"**)  
 @TableField(value=**"create\_time"**) *//使用数据库时间  
 private* Date **createTime**=**new** Date();  
  
 @DateTimeFormat(pattern=**"yyyy-MM-dd HH:mm:ss"**)  
 @TableField(value=**"update\_time"**,update=**"now()"**) *//使用数据库时间  
 private* Date **updateTime**=**new** Date();  
  
 *private* String **createBy**=**"admin"**;  
  
 *private* String **updateBy**=**"admin"**;  
}

这个类是对公共字段的处理的抽取

### 4.注解

* @NoArgsConstructor： 生成一个无参数的构造方法
* @AllArgsContructor： 会生成一个包含所有变量
* @RequiredArgsConstructor： 会生成一个包含常量，和标识了NotNull的变量的构造方法。生成的构造方法是私有的private。
* @Data ： 注在类上，提供类的get、set、equals、hashCode、canEqual、toString方法
* @Setter ： 注在属性上，提供 set 方法
* @Getter ： 注在属性上，提供 get 方法
* @EqualsAndHashCode ： 注在类上，提供对应的 equals 和 hashCode 方法
* @Log4j/@Slf4j ： 注在类上，提供对应的 Logger 对象，变量名为 log

## 第十八章 集成日志

当我们在构建SpringBoot项目时，引入了一层依赖:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<version>2.1.1.RELEASE</version>

<scope>compile</scope>

</dependency>

这里面就会有一个日志启动器：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

<version>2.1.1.RELEASE</version>

<scope>compile</scope>

</dependency>

可以看到，springboot底层默认是使用SLF4J+Logback 的方式记录日志，并且把其他的日志框架也替换成了SLF4J。当我们引入其他框架的时候，只需要把这个框架的默认日志依赖移除掉即可，SpringBoot能自动适配所有的日志。

SpringBoot默认的日志输出如下所示：

2014-03-05 10:57:51.112 INFO 45469 --- [   main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/7.0.52

2014-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]  : Initializing Spring embedded WebApplicationContext

2014-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1] o.s.web.context.ContextLoader   : Root WebApplicationContext: initialization completed in 1358 ms

2014-03-05 10:57:51.698 INFO 45469 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet' to [/]

2014-03-05 10:57:51.702 INFO 45469 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/\*]

输出以下项目：

* 日期和时间：毫秒精度并且容易排序
* 日志级别：ERROR, WARN, INFO, DEBUG, TRACE
* 进程ID
* --- 分隔符来区分实际日志的开始
* 线程名：括在方括号中（可能会被截断）
* 日志记录器名：这通常是源类名（通常缩写）
* 日志信息

注：Logback中没有FATAL(致命)级别，它被映射到ERROR级别

SpringBoot集成日志有两种方式，一种是直接在项目配置文件中设置，还有一种是使用专门的logback配置文件。

### 在项目配置文件中设置

*#设置绝对输出路径文件  
#logging.file.name=D:/log/my.log  
#设置相对输出路径文件***logging.file.name**=**log/my.log  
logging.file.path**=**log  
logging.level.root**=**info  
logging.level.com.fd.springboot**=**debug***# 检查终端是否支持ANSI，是的话就采用彩色输出***spring.output.ansi.enabled**=*detect*

### 2.独立的logback配置文件

logback-spring.xml文件：(logback-spring是默认的配置文件名)

*<?*xml version="1.0" encoding="UTF-8"*?>  
<!-- 日志级别从低到高分为TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为WARN，则低于WARN的信息都不会输出 -->  
<!-- scan:当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true -->  
<!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。 -->  
<!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。 -->*<configuration scan="true" scanPeriod="10 seconds">  
  
 *<!--<include resource="org/springframework/boot/logging/logback/base.xml" />-->* <contextName>logback</contextName>  
 *<!-- name的值是变量的名称，value的值时变量定义的值。通过定义的值会被插入到logger上下文中。定义变量后，可以使“${}”来使用变量。 -->* **<property name="log.path" value="D:/log" />**  
  
 *<!-- 彩色日志 -->  
 <!-- 彩色日志依赖的渲染类 -->* <conversionRule conversionWord="clr" converterClass="org.springframework.boot.logging.logback.ColorConverter" />  
 <conversionRule conversionWord="wex" converterClass="org.springframework.boot.logging.logback.WhitespaceThrowableProxyConverter" />  
 <conversionRule conversionWord="wEx" converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrowableProxyConverter" />  
 *<!-- 彩色日志格式 -->* <property name="CONSOLE\_LOG\_PATTERN" value="${CONSOLE\_LOG\_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(${LOG\_LEVEL\_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG\_EXCEPTION\_CONVERSION\_WORD:-%wEx}}"/>  
  
 *<!--输出到控制台-->* <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">  
 *<!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级别的日志信息-->* <filter class="ch.qos.logback.classic.filter.ThresholdFilter">  
 <level>DEBUG</level>  
 </filter>  
 <encoder>  
 <Pattern>${CONSOLE\_LOG\_PATTERN}</Pattern>  
 *<!-- 设置字符集 -->* <charset>UTF-8</charset>  
 </encoder>  
 </appender>  
  
 *<!--输出到文件-->  
 <!-- 时间滚动输出 level为 DEBUG 日志 -->* <appender name="DEBUG\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 *<!-- 正在记录的日志文件的路径及文件名 -->* <file>${log.path}/log\_debug.log</file>  
 *<!--日志文件输出格式-->* <encoder>  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 <charset>UTF-8</charset> *<!-- 设置字符集 -->* </encoder>  
 *<!-- 日志记录器的滚动策略，按日期，按大小记录 -->* <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 *<!-- 日志归档 -->* <fileNamePattern>${log.path}/debug/log-debug-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>100MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 *<!--日志文件保留天数-->* <maxHistory>15</maxHistory>  
 </rollingPolicy>  
 *<!-- 此日志文件只记录debug级别的 -->* <filter class="ch.qos.logback.classic.filter.LevelFilter">  
 <level>DEBUG</level>  
 <onMatch>ACCEPT</onMatch>  
 <onMismatch>DENY</onMismatch>  
 </filter>  
 </appender>  
  
 *<!-- 时间滚动输出 level为 INFO 日志 -->* <appender name="INFO\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 *<!-- 正在记录的日志文件的路径及文件名 -->* <file>${log.path}/log\_info.log</file>  
 *<!--日志文件输出格式-->* <encoder>  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 <charset>UTF-8</charset>  
 </encoder>  
 *<!-- 日志记录器的滚动策略，按日期，按大小记录 -->* <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 *<!-- 每天日志归档路径以及格式 -->* <fileNamePattern>${log.path}/info/log-info-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>100MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 *<!--日志文件保留天数-->* <maxHistory>15</maxHistory>  
 </rollingPolicy>  
 *<!-- 此日志文件只记录info级别的 -->* <filter class="ch.qos.logback.classic.filter.LevelFilter">  
 <level>INFO</level>  
 <onMatch>ACCEPT</onMatch>  
 <onMismatch>DENY</onMismatch>  
 </filter>  
 </appender>  
  
 *<!-- 时间滚动输出 level为 WARN 日志 -->* <appender name="WARN\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 *<!-- 正在记录的日志文件的路径及文件名 -->* <file>${log.path}/log\_warn.log</file>  
 *<!--日志文件输出格式-->* <encoder>  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 <charset>UTF-8</charset> *<!-- 此处设置字符集 -->* </encoder>  
 *<!-- 日志记录器的滚动策略，按日期，按大小记录 -->* <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <fileNamePattern>${log.path}/warn/log-warn-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>100MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 *<!--日志文件保留天数-->* <maxHistory>15</maxHistory>  
 </rollingPolicy>  
 *<!-- 此日志文件只记录warn级别的 -->* <filter class="ch.qos.logback.classic.filter.LevelFilter">  
 <level>WARN</level>  
 <onMatch>ACCEPT</onMatch>  
 <onMismatch>DENY</onMismatch>  
 </filter>  
 </appender>  
  
  
 *<!-- 时间滚动输出 level为 ERROR 日志 -->* <appender name="ERROR\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 *<!-- 正在记录的日志文件的路径及文件名 -->* <file>${log.path}/log\_error.log</file>  
 *<!--日志文件输出格式-->* <encoder>  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 <charset>UTF-8</charset> *<!-- 此处设置字符集 -->* </encoder>  
 *<!-- 日志记录器的滚动策略，按日期，按大小记录 -->* <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <fileNamePattern>${log.path}/error/log-error-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>100MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 *<!--日志文件保留天数-->* <maxHistory>15</maxHistory>  
 </rollingPolicy>  
 *<!-- 此日志文件只记录ERROR级别的 -->* <filter class="ch.qos.logback.classic.filter.LevelFilter">  
 <level>ERROR</level>  
 <onMatch>ACCEPT</onMatch>  
 <onMismatch>DENY</onMismatch>  
 </filter>  
 </appender>  
  
 *<!--  
 <logger>用来设置某一个包或者具体的某一个类的日志打印级别、  
 以及指定<appender>。<logger>仅有一个name属性，  
 一个可选的level和一个可选的addtivity属性。  
 name:用来指定受此logger约束的某一个包或者具体的某一个类。  
 level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，  
 还有一个特俗值INHERITED或者同义词NULL，代表强制执行上级的级别。  
 如果未设置此属性，那么当前logger将会继承上级的级别。  
 addtivity:是否向上级logger传递打印信息。默认是true。  
 -->  
 <!--<logger name="org.springframework.web" level="info"/>-->  
 <!--<logger name="org.springframework.scheduling.annotation.ScheduledAnnotationBeanPostProcessor" level="INFO"/>-->  
 <!--  
 使用mybatis的时候，sql语句是debug下才会打印，而这里我们只配置了info，所以想要查看sql语句的话，有以下两种操作：  
 第一种把<root level="info">改成<root level="DEBUG">这样就会打印sql，不过这样日志那边会出现很多其他消息  
 第二种就是单独给dao下目录配置debug模式，代码如下，这样配置sql语句会打印，其他还是正常info级别：  
 -->  
  
 <!--  
 root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性  
 level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，  
 不能设置为INHERITED或者同义词NULL。默认是DEBUG  
 可以包含零个或多个元素，标识这个appender将会添加到这个logger。  
 -->  
  
 <!--特殊包日志设定-->* **<logger name="com.fd.springboot" level="DEBUG"/>**  
  
 <root level="INFO">  
 <appender-ref ref="CONSOLE" />  
 <appender-ref ref="DEBUG\_FILE" />  
 <appender-ref ref="INFO\_FILE" />  
 <appender-ref ref="WARN\_FILE" />  
 <appender-ref ref="ERROR\_FILE" />  
 </root>  
  
 *<!--生产环境:输出到文件-->  
 <!--<springProfile name="pro">-->  
 <!--<root level="info">-->  
 <!--<appender-ref ref="CONSOLE" />-->  
 <!--<appender-ref ref="DEBUG\_FILE" />-->  
 <!--<appender-ref ref="INFO\_FILE" />-->  
 <!--<appender-ref ref="ERROR\_FILE" />-->  
 <!--<appender-ref ref="WARN\_FILE" />-->  
 <!--</root>-->  
 <!--</springProfile>-->*</configuration>

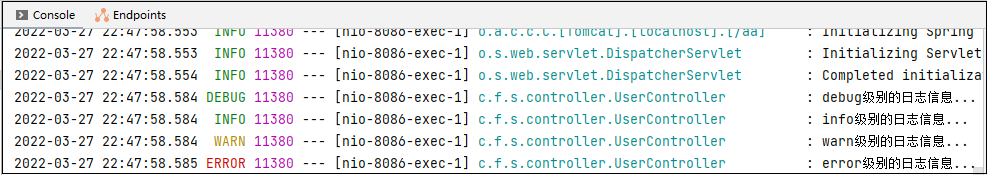
### 3.控制器文件

StudentController.java文件：

@Controller  
@Slf4j  
**public class** UserController {

@GetMapping(**"/user/getUsers"**)  
@ResponseBody  
**public** String getAllUser(){  
 ***log***.debug(**"debug级别的日志信息..."**);  
 ***log***.info(**"info级别的日志信息..."**);  
 ***log***.warn(**"warn级别的日志信息..."**);  
 ***log***.error(**"error级别的日志信息..."**);  
 List<User> users = **userRepository**.findAll();  
 **return** JSONObject.*toJSONString*(users);  
}

### 4.控制台输出



### 5.日志输出

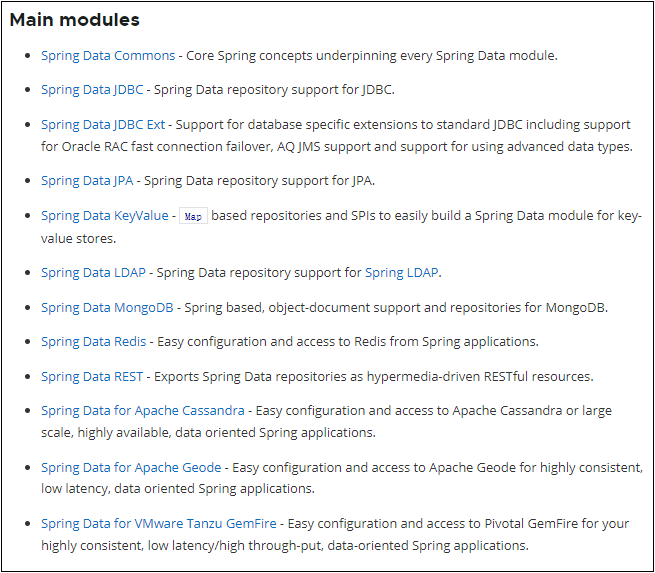


## 集成JPA

### 1.JPA介绍

Spring Data Jpa是属于Spring Data的一个子项目，Spring data项目是一款集成了很多数据操作的项目，其下的子项目有如下图所示：  
官方也有提供了英文版的文档，具体可以参考[Spring data官方文档](https://spring.io/projects/spring-data" \t "https://www.cnblogs.com/mzq123/p/_blank)

JPA是一个规范，并非具体的实现框架，而Spring Data JPA就是依赖于Hibernate JPA的实现。



目前项目（Springboot2.5.2）中所用到的Hibernate包版本为：

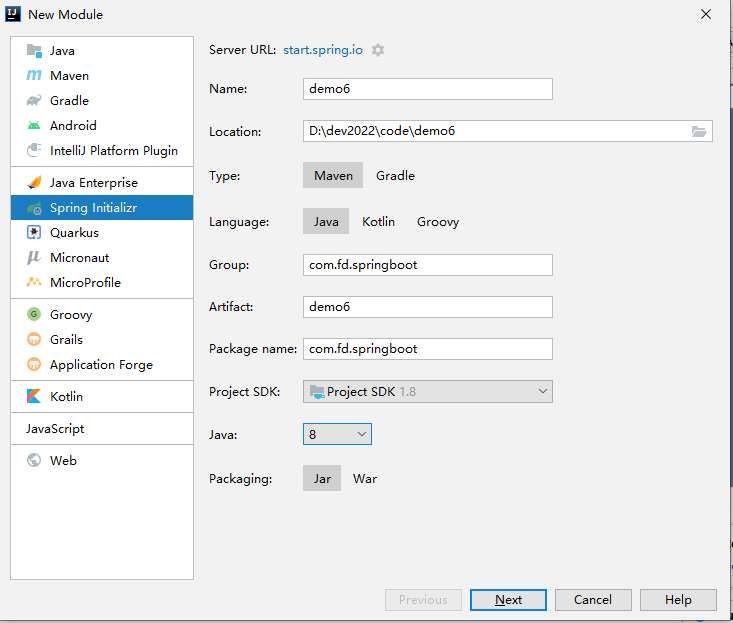


用到的jpa包版本为：

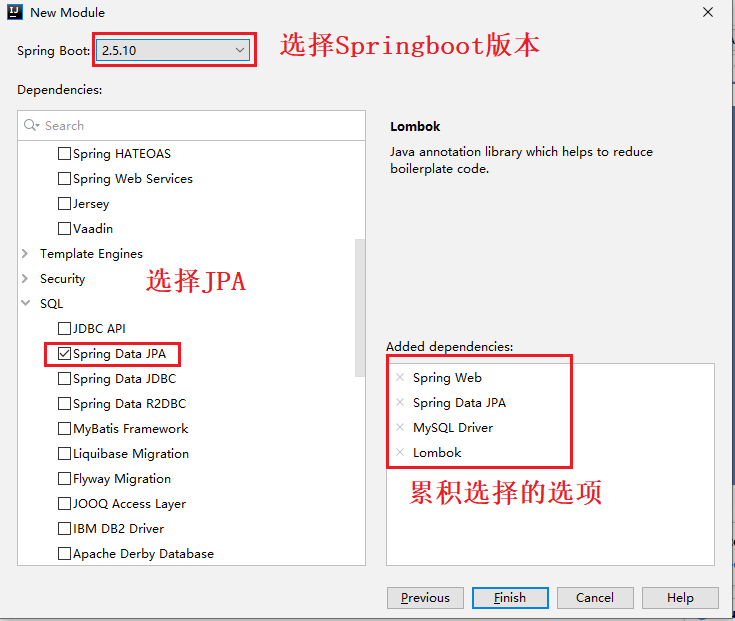


### 创建Springboot项目

在创建时可以将Spring Web、JPA、MySQL、Lombok等一起选上，也可以创建项目后修改pom.xml文件。



点击“Next”后可以勾选组件



### pom依赖文件

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.5.2</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
 <**groupId**>com.fd.springboot</**groupId**>  
 <**artifactId**>demo8</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>demo8</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 </**properties**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**optional**>true</**optional**>  
 </**dependency**>  
 *<!--支持swagger3-->* <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-boot-starter</**artifactId**>  
 <**version**>3.0.0</**version**>  
 </**dependency**>  
 *<!--支持swagger3 UI-->* <**dependency**>  
 <**groupId**>com.github.xiaoymin</**groupId**>  
 <**artifactId**>swagger-bootstrap-ui</**artifactId**>  
 <**version**>1.9.6</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 </**dependencies**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>

<**version**>2.5.2</**version**>  
 <**configuration**>  
 <**excludes**>  
 <**exclude**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 </**exclude**>  
 </**excludes**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
</**project**>

注：集成JPA，前提必须首先集成数据库的驱动，我们选择的是MySQL的链接驱动

### 修改项目配置文件

在项目配置文件中增加两项内容MySQL数据库的连接配置和JPA的配置

*#MySQL数据库配置参数,com.mysql.cj.jdbc.Driver为MySQL8.x驱动***spring.datasource.driver-class-name**=**com.mysql.cj.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://localhost:3306/fd2022?useUnicode=true&characterEncoding=utf-8&useSSL=true&serverTimezone=Asia/Shanghai  
spring.datasource.username**=**root  
spring.datasource.password**=**root**

*#更新或者创建数据表结构***spring.jpa.hibernate.ddl-auto**: **update***#数据库字段与java对象类属性的映射规则，hibernate5由implicit-strategy和physical-strategy两个来负责***spring.jpa.hibernate.naming.implicit-strategy**=**org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl  
spring.jpa.hibernate.naming.physical-strategy**=**org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl***#控制台显示SQL***spring.jpa.show-sql**=**true***#控制台格式化SQL***spring.jpa.properties.hibernate.format\_sql**=**true**

注：

* Hibernate5之前

命名策略采用naming-strategy配置项

**spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.DefaultNamingStrategy**

（1）配置值为org.hibernate.cfg.DefaultNamingStrategy时：

hibernate默认配置，采用直接映射的方式，不会做过多的处理，当然前提是没有使用@Table和@Column注解，如果有则以注解内容为准。

（2）配置值为org.hibernate.cfg.ImprovedNamingStrategy时：

表名，字段为小写，当有大写字母的时候会添加下划线分隔符号，如：user\_id。

* Hibernate5之后

采用implicit-strategy和physical-strategy两个配置项分别控制命名策略

**spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl**

**spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl**

implicit-strategy和physical-strategy的区别

（1）implicit-strategy负责模型对象层次的处理，将对象模型处理为逻辑名称。physical-strategy负责映射成真实的数据名称的处理，将上述的逻辑名称处理为物理名称。

（2）当没有使用@Table和@Column注解时，implicit-strategy配置项才会被使用，当对象模型中已经指定时，implicit-strategy并不会起作用。physical-strategy一定会被应用，与对象模型中是否显式地指定列名或者已经被隐式决定无关。

* implicit-strategy逻辑名称命名策略

有五个配置值：

org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl

org.hibernate.boot.model.naming.ImplicitNamingStrategyComponentPathImpl

org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl

org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImpl

org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy

默认为ImplicitNamingStrategyJpaCompliantImpl，后四者均继承自它。

* physical-strategy物理名称命名策略

有两个配置值：

#直接映射，不会做过多的处理

org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

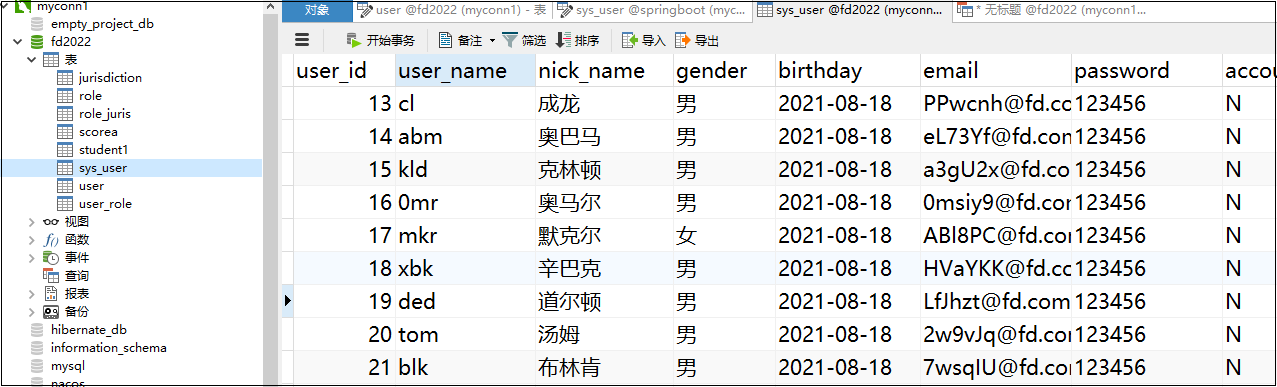
#表名，字段为小写，当有大写字母的时候会添加下划线分隔符号

org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy

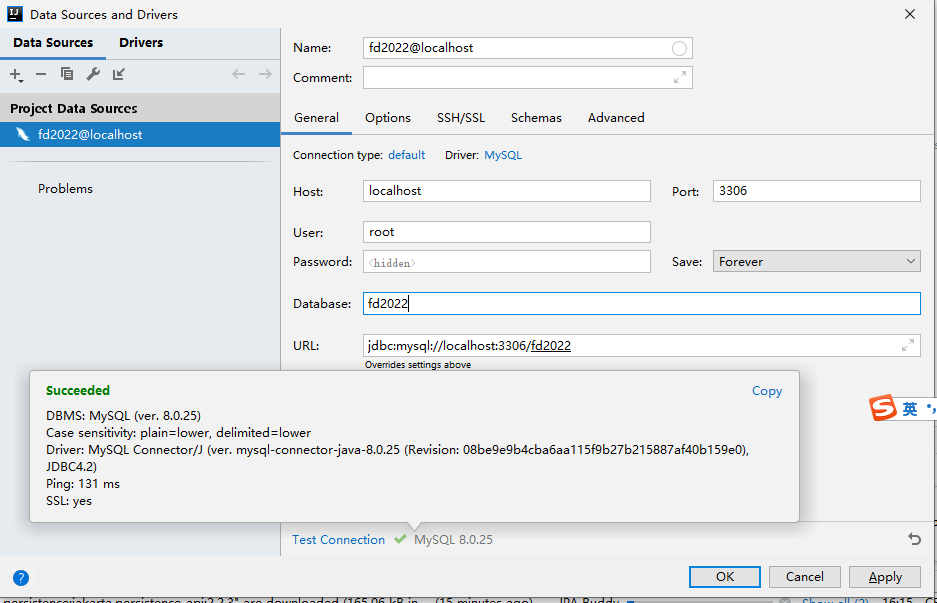
默认为SpringPhysicalNamingStrategy

### 5.在MySQL的fd2022中导入表和数据

在使用JPA之前，需要准备一张数据表sys\_user,将老师提供的sys\_user.sql导入数据库中。



### 6.在IDEA中配置数据库环境

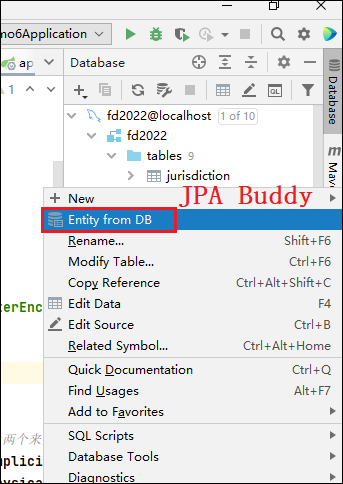


### 创建实体类

在com.fd.springboot下创建实体类包entity，在包中创建实体类User.java

@AllArgsConstructor *//包含所有字段的构造方法*@NoArgsConstructor *//无参构造方法*@Data *//get/set方法*@ToString *//生成tostring方法*@Entity  
@Table(name = **"sys\_user"**)  
**public class** User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"user\_id"**, nullable = **false**)  
 **private** Integer **userid**;  
  
 @Column(name = **"user\_name"**)  
 **private** String **username**;  
  
 @Column(name = **"nick\_name"**)  
 **private** String **nickname**;  
  
 @Column(name = **"gender"**)  
 **private** String **gender** = **"男"**;  
  
 @Column(name = **"birthday"**)  
 **private** Date **birthday** = **new** Date();  
  
 @Column(name = **"email"**)  
 **private** String **email**;  
  
 @Column(name = **"password"**)  
 **private** String **password**;  
  
 @Column(name = **"account\_locked"**)  
 **private** String **accountlocked**;  
  
 @Column(name = **"create\_time"**)  
 **private** Date **createtime** = **new** Date();  
  
 @Column(name = **"create\_by"**)  
 **private** String **createby**;  
  
 @Column(name = **"update\_time"**)  
 **private** Date **updatetime** = **new** Date();  
  
 @Column(name = **"update\_by"**)  
 **private** String **updateby**;  
}

可以安装JPA Buddy插件帮助我们完成实体类的构造，JPA Buddy需要在Marketplace中下载安装。



### 创建Repository（DAO）接口

在com.fd.springboot下创建实体类包repository，在Hibernate和Mybatis中这个接口类包会被直接命名为dao

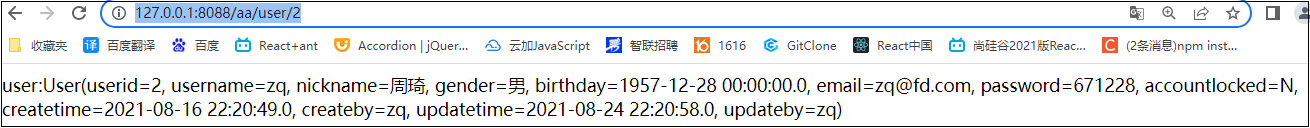
**public interface** UserRepository **extends** JpaRepository<User,Integer> {  
  
}

### 创建控制器类

@Controller  
**public class** UserController {  
  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 @GetMapping(**"/user/{userId}"**)  
 **public** @ResponseBody String getUser(@PathVariable(**"userId"**) Integer userId){  
 *//User user = userRepository.getOne(userId);* User user = **userRepository**.getById(userId);  
 System.***out***.println(**"user:"**+user.toString());  
 **return "user:"**+user.toString();  
 }  
}

### 运行并访问

<http://127.0.0.1:8088/aa/user/2>



### 使用Junit进行测试访问

@SpringBootTest  
**class** Demo8ApplicationTests {  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 @Test  
 **void** contextLoads() {  
 PageRequest pageReques=PageRequest.*of*(0,10); *//获取第1页的10条记录* Page<User> page=**userRepository**.findAll(pageReques);  
 List<User> users=page.getContent();  
 **for**(User t:users)  
 System.***out***.println(t);  
 }  
  
}

运行后控制台输出如下：



### 自定义查询--原生SQL

修改User的repository接口：

**public interface** UserRepository **extends** JpaRepository<User,Integer> {  
 //根据字符串模糊查找姓名

@Query(value = **"select \* from sys\_user u where u.user\_name like %?1%"**,nativeQuery=**true**)  
 **public** List<User> findByName(String userName);

//根据整数参数返回记录条数集合  
 @Query(value = **"select \* from sys\_user order by RAND() limit ?1"**,nativeQuery=**true**)  
 **public** List<User> listRandom(Integer number);  
}

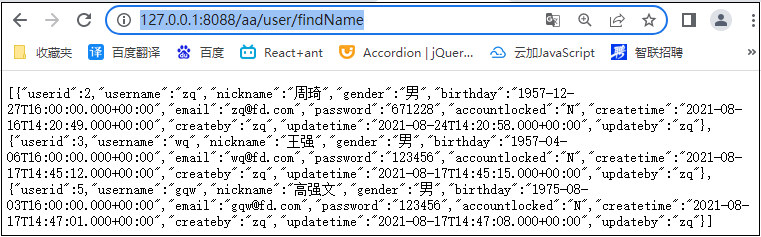
在上面代码中定义了两个自定义接口，使用的是自定义SQL语句

在控制器中增加路径访问：

@ResponseBody *// 返回josn*@GetMapping(**"/user/findName"**)  
**public** List<User> findName() {  
 **return userRepository**.findByName(**"q"**); *// 调用模糊查询的方法*}

### 运行并访问

http://127.0.0.1:8088/aa/user/findName



### 自定义查询-HQL

*//自定义HQL查询*@Query(**"from User u where u.userName = ?1"**)  
*//@Query("select u from User u where u.userName = ?1")*User findByUserName(String userName);

### 14.包含事务的自定义查询-HQL

（1）什么是事务？

我们在开发企业应用时，对于业务人员的一个操作实际是对数据读写的多步操作的结合。由于数据操作在顺序执行的过程中，任何一步操作都有可能发生异常，异常会导致后续操作无法完成，此时由于业务逻辑并未正确的完成，之前成功操作数据的并不可靠，需要在这种情况下进行回退。

事务的作用就是为了保证用户的每一个操作都是可靠的，事务中的每一步操作都必须成功执行，只要有发生异常就回退到事务开始未进行操作的状态。

事务管理是Spring框架中最为常用的功能之一，我们在使用SpringBoot开发应用时，大部分情况下也都需要使用事务。

（2）Springboot中的事务注解**@Transactional**

*//如果是更新或者删除操作，方法上面要加@Modifying,  
// 默认开启的事务只是可读的，更新操作加入@Modifying就会关闭可读*@Modifying  
@Transactional  
@Query(**"update User set password = ?1 where id = ?2"**)  
**void** updatePassword( String password,Integer id);

### 15.Service层

（1）Service层接口类

**public interface** UserService {  
 **public** User findByUserName(String UserName);  
 **public void** updatePassword( String password,Integer id);  
}

1. Service层接口实现类

@Service  
**public class** UserServiceImpl **implements** UserService {  
 @Autowired  
 UserRepository **userRepository**;  
  
 @Override  
 **public** User findByUserName(String UserName) {  
 **return userRepository**.findByUserName(UserName);  
 }  
  
 @Override  
 @Transactional  
 **public void** updatePassword(String password, Integer id) {  
 **userRepository**.updatePassword(password, id);  
*// int i = 1/0;* }  
}

### 实体类的校验

下面语句中的nullable = false, length = 60为实体类定义的相对于数据库定义的校验规则

@Column(name = "user\_name", nullable = false, length = 60)

类似的规则还有很多，下面是Java这端的校验规则：

@Email(message = "邮箱格式不正确")

@Size(min = 0, max = 50, message = "邮箱长度不能超过50个字符")

@Size(min = 0, max = 11, message = "手机号码长度不能超过11个字符")

@NotBlank(message = "登录账号不能为空")

@Size(min = 0, max = 30, message = "登录账号长度不能超过30个字符")

日期的规则：

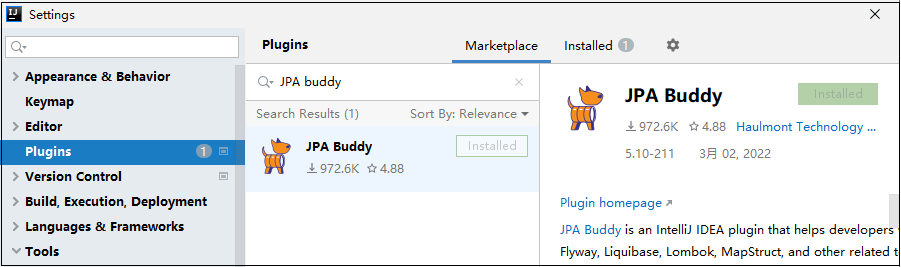
@DateTimeFormat(pattern="yyyy-MM-dd HH:mm:ss") //入参的格式化

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss") //出参的格式化

private Date date;

### JPA Buddy插件的安装和使用

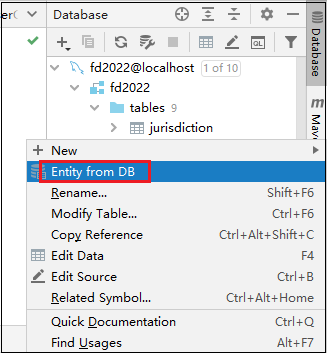
（1）点开Settings菜单：



如图示，点击安装，已安装完毕的会显示“installed”

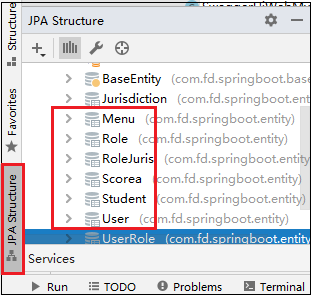
（2）创建实体类

在IDEA的右侧找到“Database”配置数据源，在“tables”上点击右键，显示如下，会多出一个“Entity from DB”的菜单，点击此菜单可以创建对应表的实体类。

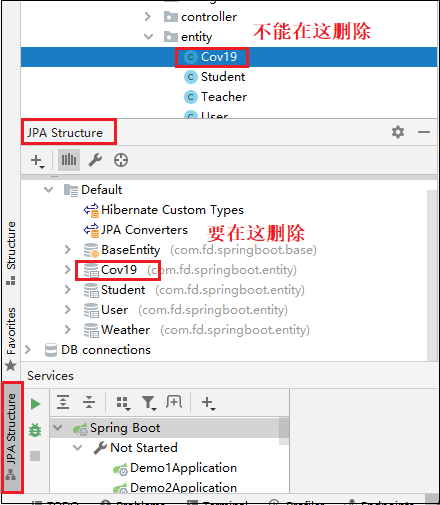


1. 管理已创建的而实体类

点击IDEA窗体左下角的“JPA Structure”，可以看到已创建的实体类，如果实体类想重新生成，需要从这里删除，再重新创建，否则只会创建变化的字段；如果数据库表的字段定义没变化，一个字段都不会显示，无法重新创建；



实体类生成后，因为一些原因需要重建，有两种情况，一种是增加修改了个别字段定义，可以使用JPA Buddy提供的增量重建，默认就是采用增量重建，如果没有修改，会发现无法重建实体类，因为JPA Buddy认为你没必要重建，因为没啥可改的。还有一种方式是强制删除创建的实体类，然后重建，原理上可行，但实际上执行时发现不行，重建不了，实际上是可以的，因为你不能直接去实体类目录下删除实体类，你需要使用JPA Buddy来删除，怎么做呢？看下图。



### 注解

本章的注解较多，总结如下：

@Entity 实体类注解，为容器Bean，为JDK注解

@Table 与数据库表的映射注解，为JDK注解

@Id 与数据库表的主键字段映射的注解，为JDK注解

@@GeneratedValue 主键生成策略注解，为JDK注解

@Column 与数据库表的字段映射注解，为JDK注解

@Repository DAO层注解，为容器Bean，Spring注解

@Service 业务层（服务层）注解，为容器Bean，Spring注解

@Transactional 事务注解，为JDK注解

@Modifying 可编辑注解，Spring JPA注解

@Query 查询注解，JPA构造SQL或HQL语句，Spring JPA注解

## 第二十章 测试工具的使用

### 1.[Postman安装与使用](https://www.cnblogs.com/fnng/p/9136434.html)

Postman一款非常流行的API调试工具。其实，开发人员用的更多。因为测试人员做接口测试会有更多选择，例如Jmeter、soapUI等。不过，对于开发过程中去调试接口，Postman确实足够的简单方便，而且功能强大。

官方网站：[https://www.getpostman.com/](https://www.getpostman.com/" \t "https://www.cnblogs.com/fnng/p/_blank)

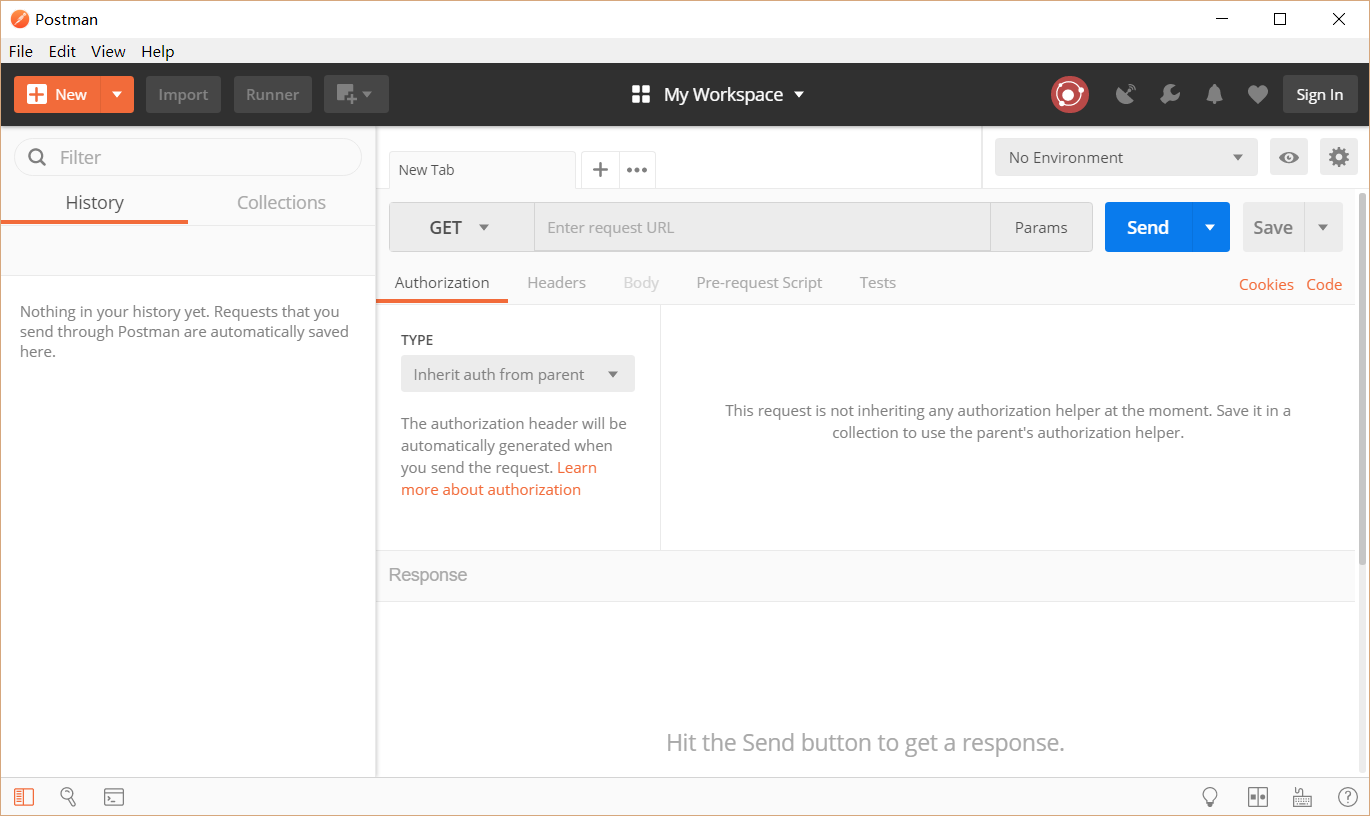
安装

（1）Postman最早是作用chrome浏览器插件存在的，所以，你可以到chrome商店搜索下载安装，因为重所周知的原因，所以，大家都会找别人共享的postman插件文件来安装。由于2018年初Chrome停止对Chrome应用程序的支持。

（2）Postman提供了独立的安装包，不再依赖于Chrome浏览器了。同时支持MAC、Windows和Linux，推荐你使用这种方式安装。[https://www.getpostman.com/apps](https://www.getpostman.com/apps" \t "https://www.cnblogs.com/fnng/p/_blank)

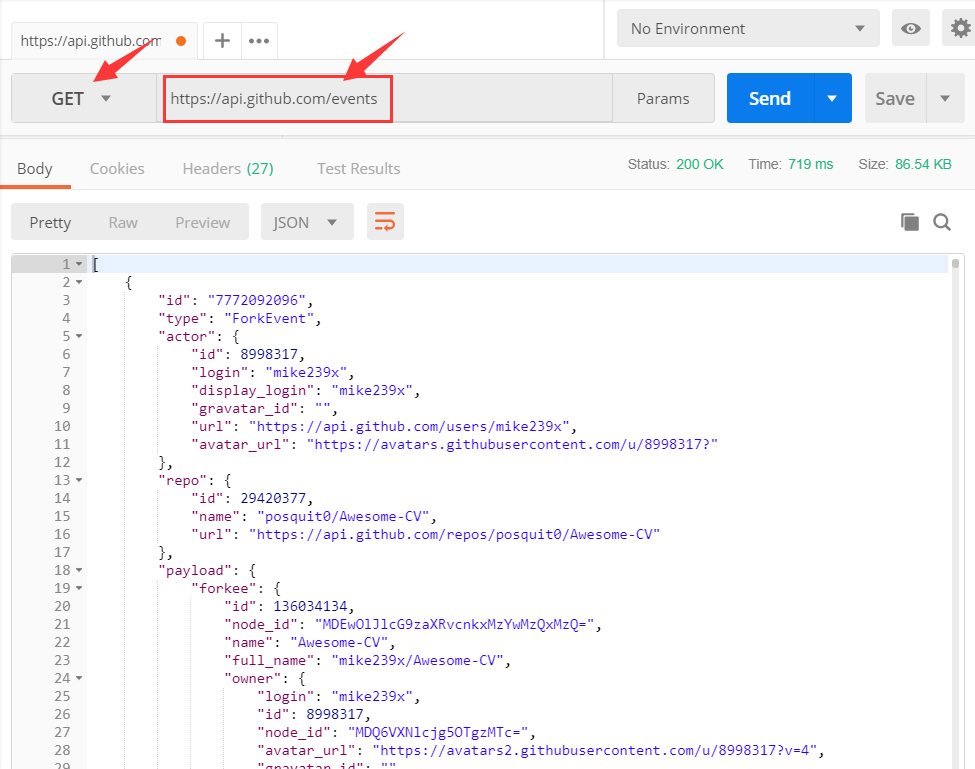
使用

Postman主界面：



（1）简单的Get请求

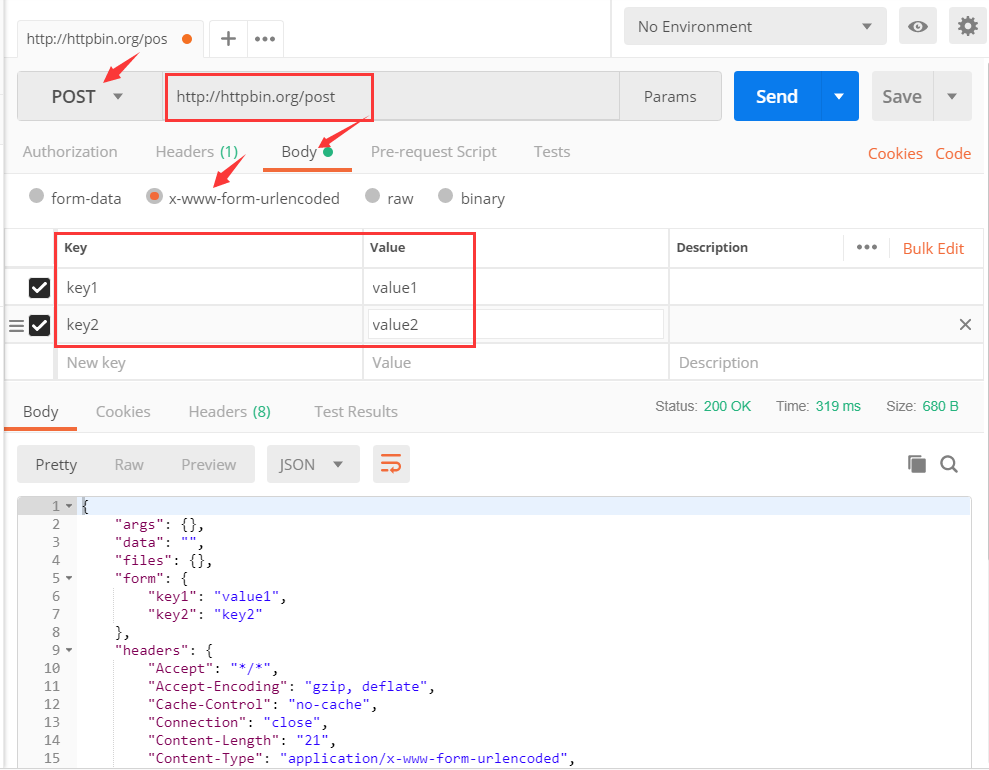
参考：[http://www.python-requests.org/en/master/user/quickstart/](http://www.python-requests.org/en/master/user/quickstart/" \t "https://www.cnblogs.com/fnng/p/_blank)



GET：HTTP的常用请求方法。  
"[https://api.github.com/events](https://api.github.com/events" \t "https://www.cnblogs.com/fnng/p/_blank)"：请求的URL  
点击蓝色“Send”按钮，获取返回值。  
注： GET请求的参数在url后面拼接，如："[https://api.github.com/events?id=1&name=user](https://api.github.com/events?id=1&name=user" \t "https://www.cnblogs.com/fnng/p/_blank)"

（2）简单的POST请求

参考：[http://www.python-requests.org/en/master/user/quickstart/](http://www.python-requests.org/en/master/user/quickstart/" \t "https://www.cnblogs.com/fnng/p/_blank)

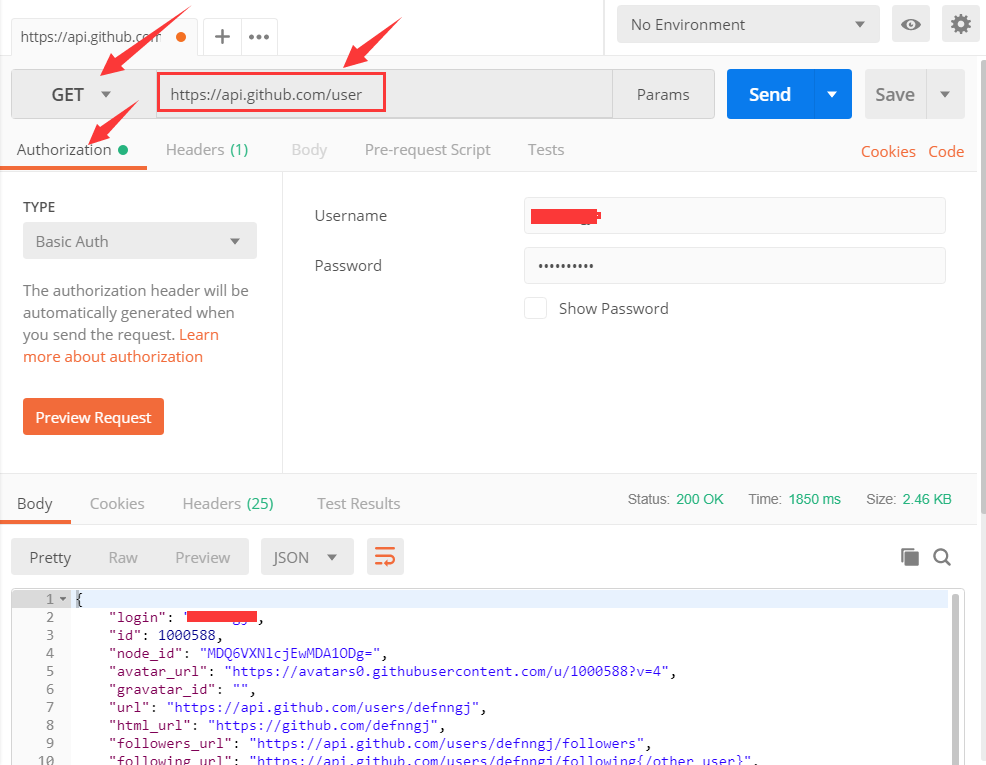


POST：HTTP的常用请求方法。  
"[http://httpbin.org/post](http://httpbin.org/post" \t "https://www.cnblogs.com/fnng/p/_blank)"：请求的URL。  
Body：设置POST请求的参数。

* form-data： HTTP请求中的multipart/form-data，它会将表单的数据处理为一条消息，以标签为单元，用分隔符分开。
* x-wwww-form-urlencode：HTTP请求中的application/x-www-from-urlencoded，会将表单内的数据转换为键值对。
* raw：可以发送任意格式的接口数据，可以text、json、xml、html等。
* binary：HTTP请求中的相Content-Type:application/octet-stream，只可以发送二进制数据。通常用于文件的上传。

（3）认证接口

创建一个接口调用  
参考：[http://www.python-requests.org/en/master/](http://www.python-requests.org/en/master/" \t "https://www.cnblogs.com/fnng/p/_blank)



* Authorization：用于需要认证的接口。
* Basic Auth：最基本的一种认证类型，还有OAuth 1.0/2.0、Digest Auth等认证类型。
* Username/Password：这是针对Basic Auth类型的认证的用户名/密码，并非我们认为的系统登录的用户名密码

### 2.[Apifox安装与使用](https://www.cnblogs.com/fnng/p/9136434.html)

Apifox为国产的项目管理、API测试管理工具

https://www.apifox.cn/

## 第二十一章 过滤器和拦截器

在开发时我们通常不会注意路径中的双斜杠问题，但在实际部署时这个问题就很严重，会导致访问失败，为此我们可以配置过滤器解决这个问题。

### 1.要求Spring Security安全放行

要求Spring Security放行**uriFormatFilter**过滤器。

PermitAllSecurityConfig.java文件：

**package** com.fd.springboot.config;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.web.servlet.FilterRegistrationBean;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.security.config.annotation.SecurityConfigurerAdapter;  
**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;  
**import** org.springframework.security.web.DefaultSecurityFilterChain;  
**import** org.springframework.stereotype.Component;  
  
**import** javax.servlet.Filter;  
  
  
@Component(**"permitAllSecurityConfig"**)  
**public class** PermitAllSecurityConfig **extends** SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {  
  
 @Autowired  
 **private** Filter **uriFormatFilter**;  
  
 @Bean  
 **public** FilterRegistrationBean setFilter() {  
 FilterRegistrationBean filterBean = **new** FilterRegistrationBean();  
 filterBean.setFilter(**uriFormatFilter**);  
 filterBean.setName(**"uriFormatFilter"**);  
 filterBean.addUrlPatterns(**"/\*"**);  
 filterBean.setOrder(-10000);  
 **return** filterBean;  
 }  
}

### 2.自定义的过滤器

UriFormatFilter.java文件：

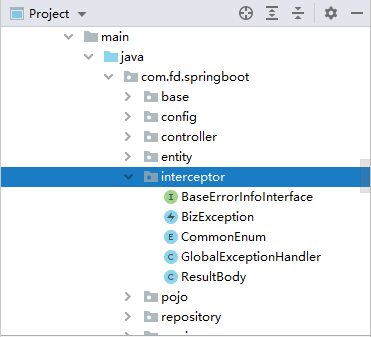
**package** com.fd.springboot.config;  
  
**import** org.springframework.stereotype.Component;  
**import** org.springframework.web.filter.OncePerRequestFilter;  
  
**import** javax.servlet.FilterChain;  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletRequestWrapper;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
  
@Component(**"uriFormatFilter"**)  
**public class** UriFormatFilter **extends** OncePerRequestFilter {  
 @Override  
 **protected void** doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, FilterChain filterChain) **throws** ServletException, IOException {  
  
 String uri = httpServletRequest.getRequestURI();  
 String newUri = uri.replace(**"//"**,**"/"**);  
 httpServletRequest = **new** HttpServletRequestWrapper(httpServletRequest){  
 @Override  
 **public** String getRequestURI() {  
 **return** newUri;  
 }  
 };  
  
 filterChain.doFilter(httpServletRequest, httpServletResponse);  
 }  
  
}

### 3.修改控制器访问方法

*//双斜杠路径处理实例*@GetMapping(**"/user//getUsers"**)  
@ResponseBody  
**public** String getAllUser(){  
 ***log***.debug(**"debug级别的日志信息..."**);  
 ***log***.info(**"info级别的日志信息..."**);  
 ***log***.warn(**"warn级别的日志信息..."**);  
 ***log***.error(**"error级别的日志信息..."**);  
 List<User> users = **userRepository**.findAll();  
 **return** JSONObject.*toJSONString*(users);  
}

### 4.运行并访问

### 自定义拦截器



在com.fd.springboot包下创建interceptor包，在包中分别创建五个文件，代码如下：

1. BaseErrorInfoInterface.java接口

**package** com.fd.springboot.interceptor;  
  
**public interface** BaseErrorInfoInterface {  
 */\*\* 错误码\*/* String getResultCode();  
  
 */\*\* 错误描述\*/* String getResultMsg();  
}

1. BizException.java业务异常类

**package** com.fd.springboot.interceptor;  
  
**public class** BizException **extends** RuntimeException {  
 **private static final long *serialVersionUID*** = 1L;  
 */\*\*  
 \* 错误码  
 \*/* **protected** String **errorCode**;  
 */\*\*  
 \* 错误信息  
 \*/* **protected** String **errorMsg**;  
  
 **public** BizException() {  
 **super**();  
 }  
  
 **public** BizException(BaseErrorInfoInterface errorInfoInterface) {  
 **super**(errorInfoInterface.getResultCode());  
 **this**.**errorCode** = errorInfoInterface.getResultCode();  
 **this**.**errorMsg** = errorInfoInterface.getResultMsg();  
 }  
  
 **public** BizException(BaseErrorInfoInterface errorInfoInterface, Throwable cause) {  
 **super**(errorInfoInterface.getResultCode(), cause);  
 **this**.**errorCode** = errorInfoInterface.getResultCode();  
 **this**.**errorMsg** = errorInfoInterface.getResultMsg();  
 }  
  
 **public** BizException(String errorMsg) {  
 **super**(errorMsg);  
 **this**.**errorMsg** = errorMsg;  
 }  
  
 **public** BizException(String errorCode, String errorMsg) {  
 **super**(errorCode);  
 **this**.**errorCode** = errorCode;  
 **this**.**errorMsg** = errorMsg;  
 }  
  
 **public** BizException(String errorCode, String errorMsg, Throwable cause) {  
 **super**(errorCode, cause);  
 **this**.**errorCode** = errorCode;  
 **this**.**errorMsg** = errorMsg;  
 }  
  
 **public** String getErrorCode() {  
 **return errorCode**;  
 }  
  
 **public void** setErrorCode(String errorCode) {  
 **this**.**errorCode** = errorCode;  
 }  
  
 **public** String getErrorMsg() {  
 **return errorMsg**;  
 }  
  
 **public void** setErrorMsg(String errorMsg) {  
 **this**.**errorMsg** = errorMsg;  
 }  
  
 **public** String getMessage() {  
 **return errorMsg**;  
 }  
  
 @Override  
 **public** Throwable fillInStackTrace() {  
 **return this**;  
 }  
}

1. CommonEnumerate.java异常枚举类

**package** com.fd.springboot.interceptor;  
  
**public enum** CommonEnum **implements** BaseErrorInfoInterface{  
 *// 数据操作错误定义* ***SUCCESS***(**"200"**, **"成功!"**),  
 ***BODY\_NOT\_MATCH***(**"400"**,**"请求的数据格式不符!"**),  
 ***SIGNATURE\_NOT\_MATCH***(**"401"**,**"请求的数字签名不匹配!"**),  
 ***NOT\_FOUND***(**"404"**, **"未找到该资源!"**),  
 ***INTERNAL\_SERVER\_ERROR***(**"500"**, **"服务器内部错误!"**),  
 ***SERVER\_BUSY***(**"503"**,**"服务器正忙，请稍后再试!"**)  
 ;  
  
 */\*\* 错误码 \*/* **private** String **resultCode**;  
  
 */\*\* 错误描述 \*/* **private** String **resultMsg**;  
  
 CommonEnum(String resultCode, String resultMsg) {  
 **this**.**resultCode** = resultCode;  
 **this**.**resultMsg** = resultMsg;  
 }  
  
 @Override  
 **public** String getResultCode() {  
 **return resultCode**;  
 }  
  
 @Override  
 **public** String getResultMsg() {  
 **return resultMsg**;  
 }  
}

1. GlobalExceptionHandler.java全局异常处理类

**package** com.fd.springboot.interceptor;  
  
**import** org.springframework.web.bind.annotation.ControllerAdvice;  
**import** org.springframework.web.bind.annotation.ExceptionHandler;  
**import** org.springframework.web.bind.annotation.ResponseBody;  
  
**import** javax.servlet.http.HttpServletRequest;  
  
@ControllerAdvice  
**public class** GlobalExceptionHandler {  
 *//private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);  
 /\*\*  
 \* 处理自定义的业务异常  
 \** ***@param req*** *\** ***@param e*** *\** ***@return*** *\*/* @ExceptionHandler(value = BizException.**class**)  
 @ResponseBody  
 **public** ResultBody bizExceptionHandler(HttpServletRequest req, BizException e){  
 *//logger.error("发生业务异常！原因是：{}",e.getErrorMsg());* **return** ResultBody.*error*(e.getErrorCode(),e.getErrorMsg());  
 }  
  
 */\*\*  
 \* 处理空指针的异常  
 \** ***@param req*** *\** ***@param e*** *\** ***@return*** *\*/* @ExceptionHandler(value =NullPointerException.**class**)  
 @ResponseBody  
 **public** ResultBody exceptionHandler(HttpServletRequest req, NullPointerException e){  
 *//logger.error("发生空指针异常！原因是:",e);* **return** ResultBody.*error*(CommonEnum.***BODY\_NOT\_MATCH***);  
 }  
  
 */\*\*  
 \* 处理其他异常  
 \** ***@param req*** *\** ***@param e*** *\** ***@return*** *\*/* @ExceptionHandler(value =Exception.**class**)  
 @ResponseBody  
 **public** ResultBody exceptionHandler(HttpServletRequest req, Exception e){  
 *//logger.error("未知异常！原因是:",e);* **return** ResultBody.*error*(CommonEnum.***INTERNAL\_SERVER\_ERROR***);  
 }  
}

（5）ResultBody.java异常信息包装类

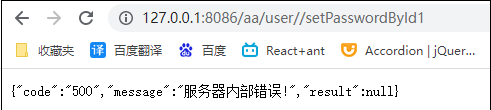
**package** com.fd.springboot.interceptor;  
  
**import** com.alibaba.fastjson.JSONObject;  
  
**public class** ResultBody {**private** String **code**; //*响应代码* **private** String **message**;//*响应消息* **private** Object **result**;//*响应结果*  
  
 **public** ResultBody() {  
 }  
  
 **public** ResultBody(BaseErrorInfoInterface errorInfo) {  
 **this**.**code** = errorInfo.getResultCode();  
 **this**.**message** = errorInfo.getResultMsg();  
 }  
  
 **public** String getCode() {  
 **return code**;  
 }  
  
 **public void** setCode(String code) {  
 **this**.**code** = code;  
 }  
  
 **public** String getMessage() {  
 **return message**;  
 }  
  
 **public void** setMessage(String message) {  
 **this**.**message** = message;  
 }  
  
 **public** Object getResult() {  
 **return result**;  
 }  
  
 **public void** setResult(Object result) {  
 **this**.**result** = result;  
 }  
  
 *//成功，null* **public static** ResultBody success() {  
 **return** *success*(**null**);  
 }  
  
 *//成功，****@param data*****public static** ResultBody success(Object data) {  
 ResultBody rb = **new** ResultBody();  
 rb.setCode(CommonEnum.***SUCCESS***.getResultCode());  
 rb.setMessage(CommonEnum.***SUCCESS***.getResultMsg());  
 rb.setResult(data);  
 **return** rb;  
 }  
  
 *//失败* **public static** ResultBody error(BaseErrorInfoInterface errorInfo) {  
 ResultBody rb = **new** ResultBody();  
 rb.setCode(errorInfo.getResultCode());  
 rb.setMessage(errorInfo.getResultMsg());  
 rb.setResult(**null**);  
 **return** rb;  
 }  
  
 *//失败* **public static** ResultBody error(String code, String message) {  
 ResultBody rb = **new** ResultBody();  
 rb.setCode(code);  
 rb.setMessage(message);  
 rb.setResult(**null**);  
 **return** rb;  
 }  
  
 *//失败* **public static** ResultBody error( String message) {  
 ResultBody rb = **new** ResultBody();  
 rb.setCode(**"-1"**);  
 rb.setMessage(message);  
 rb.setResult(**null**);  
 **return** rb;  
 }  
  
 @Override  
 **public** String toString() {  
 **return** JSONObject.*toJSONString*(**this**);  
 }  
  
}

### 人为制造Bug

修改myUpdatePassword的业务处理方法，制造一个算数异常

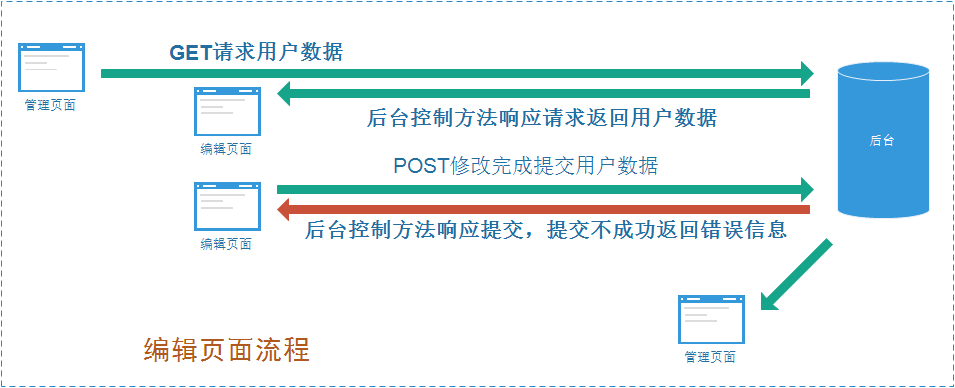
@Service  
**public class** UserServiceImpl **implements** UserService {  
 @Autowired *//@Resource* UserRepository **userRepository**;  
  
 @Override  
 @Transactional *//1.开启事务，一堆复杂的数据库操作 2.对重要的操作* **public void** myUpdatePassword(String password, Integer id) {  
 **userRepository**.updatePassword(password,id);  
 **int** i=1/0; *//用来测试事务,如果没有拦截处理，系统会奔溃* }  
  
 @Override  
 **public** List<User> findByName(String userName) {  
 **return userRepository**.findByName(userName);  
 }  
}

### 运行并访问



## 第二十二章 用户新增和编辑页面处理

编辑页面通常需要两个请求路径，一个是请求输入编辑的页面；一个是提交后的处理，处理后一般由两种情况，处理成功会跳转到管理页面，处理不成功会返回编辑页面，并提示错误信息；



### 1.两个控制器方法

（1）GET请求方法

@GetMapping(**"/user/userEdit/{Id}"**)  
 **public** String userLoad(@PathVariable(**"Id"**) Integer Id, Model m){  
 User user = **userRepository**.getById(Id);  
 System.***out***.println(**"查询:"**+user.toString());  
 ***log***.info(**"查询:"**+user.toString());  
 m.addAttribute(**"user"**,user);  
 **return "views/useredit"**;  
 }

（2）POST提交方法

@PostMapping(**"/user/userSave"**)  
*// @ResponseBody* **public** String userPost(@ModelAttribute(value = **"user"**) User user){  
 user.setCreateTime(**new** Date());  
 user.setUpdateTime(**new** Date());  
 user.setCreateBy(**"admin"**);  
 user.setUpdateBy(**"admin"**);  
 System.***out***.println(**"保存:"**+user.toString());  
 ***log***.info(**"保存:"**+user.toString());  
  
 **return "views/userlist"**;  
 }

### 2.编辑页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
  
 <**title**>用户编辑</**title**>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"**/>  
  
 <**script th:src="@{/js/jquery.min.js}"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
  
 <**style**>  
 [**readonly**=**"readonly"**] {  
 **pointer-events**: **none**;  
 }  
 </**style**>  
</**head**>  
<**body class="bg-light"**>  
<**div class="container"**>  
 <**h2 class="text-center col-4"**>用户编辑</**h2**>  
 <**form th:action="@{user/userSave}" th:object="${user}" method="post"**>  
 <**div class="form-inline"** >  
*<!-- <div class="form-inline" hidden>-->* <**span class="input-group-text col-1"**>用户ID</**span**>  
 <**input type="text" class="form-control col-3" id="id" th:field="${user.id}" readonly** />  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户名称</**span**>  
 <**input type="text" class="form-control col-3" placeholder="输入用户姓名" id="userName"  
 th:field="${user.userName}" autofocus** />  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户呢称</**span**>  
 <**input type="text" class="form-control col-3" placeholder="输入用户姓名" id="nickName"  
 th:field="${user.nickName}"** />  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户性别</**span**>  
 <**select class="form-control col-3" id="gender" placeholder="选择性别"  
 th:field="${user.gender}" th:value="${user.gender}"**>  
 <**option th:selected="${user.gender}=='男'" value="男"**>男</**option**>  
 <**option th:selected="${user.gender}=='女'" value="女"**>女</**option**>  
 </**select**>  
*<!-- <input name="gender" type="radio" th:selected="${user.gender}=='男'" th:field="${user.gender}" value="男">男</input>&nbsp;&nbsp;-->  
<!-- <input name="gender" type="radio" th:selected="${user.gender}=='女'" th:field="${user.gender}" value="女">女</input>-->* </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>出生日期</**span**>  
 <**input type="date" class="form-control col-3" placeholder="选择日期" id="birthday"  
 th:field="${user.birthday}" th:value="${#dates.format(user.birthday,'yyyy-MM-dd')}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>电子邮件</**span**>  
 <**input type="email" class="form-control col-3" placeholder="输入电子邮件" id="email"  
 th:field="${user.email}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户密码</**span**>  
 <**input type="password" class="form-control col-3" placeholder="输入用户密码" id="password"  
 th:field="${user.password}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>账户锁定</**span**>  
 <**span class="form-control col-3"**>  
 <**input name="accountlocked" type="radio" th:selected="${user.accountLocked}+'es'=='Yes'" th:field="${user.accountLocked}" value="Y"**>Yes</**input**>**&nbsp;&nbsp;** <**input name="accountlocked" type="radio" th:selected="${user.accountLocked}+'o'=='No'" th:field="${user.accountLocked}" value="N"**>No</**input**>  
*<!-- <input type="text" class="form-control col-3" placeholder="锁定账户：Y,不锁定：N" id="accountlocked"-->  
<!-- th:field="${user.accountLocked}" />-->* </**span**>  
 </**div**>  
 <**button class="btn btn-info btn-block col-4" type="submit"**>提交</**button**>  
 </**form**>  
</**div**>  
</**body**>  
</**html**>

一个字段数据在前台展示的时候，可以以多种形式出现，可以用下拉列表框，电报钮，检查盒，文本编辑框等等，上述代码中备注是的部分就是在测试不同的展现形式。

### 3.运行访问



### 4.JPA插件自动生成的实体类问题

***// @Temporal(TemporalType.DATE)*** *//这是Hibernate的注解，意思是转换成本地中文格式日期，由JPA Buddy自动生成，但此格式会造成前台返回的包装对象无法自动识别为Java对象，原因是日期格式不对，类型转换时的代码还是英文格式日期，这个错误很隐蔽，后台不会报错，但拦截器可以拦截到500的内部错误。注释掉此注解，问题解决* @DateTimeFormat(pattern=**"yyyy-MM-dd"**) *//入参的格式化* @JsonFormat(pattern = **"yyyy-MM-dd"**) *//出参的格式化* @Column(name = **"birthday"**, nullable = **false**)  
 **private** Date **birthday**;

### 5.数据校验

（1）数据校验有两个注解，这两个注解功能基本相同，都需要配合实体类JSR303规则注解完成后台数据校验功能，在SpringBoot2.3.X以上版本，需要单独引入JSR303依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-validation</**artifactId**>  
</**dependency**>

* @Valid :这是JDK官方的注解
* @Validated :这是Spring的注解,提供了一个分组功能，可以在入参验证时，根据不同的分组采用不同的验证机制。

上述两个注解都可以配合 BindingResult对象，给前台返回校验错误信息。BindingResult对象返回的信息较多，需要做处理，否则前台都接收显示，会影响界面美观。

因为加了@Valid校验，因此JPA buddy自动生成的实体类的创建时间、修改时间、创建者、修改者必须给初值，否则校验会出错，所以只能在实体类定义处给初值了。

@Table(name = **"sys\_user"**)  
@Entity  
@Getter  
@Setter  
@RequiredArgsConstructor  
@ToString  
**public class** User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"user\_id"**, nullable = **false**)  
 **private** Integer **id**;  
  
 @Length(min=1, max=60, message = **"长度必须在1-60个字符之间"**)*//JSR303*  
 @NotEmpty(message = **"用户名不能为空"**) *//JSR303* @Column(name = **"user\_name"**, nullable = **false**, length = 60)  
 **private** String **userName**=**""**;  
  
 @Column(name = **"nick\_name"**, length = 60)  
 **private** String **nickName**;  
  
 @Column(name = **"gender"**, nullable = **false**, length = 1)  
 **private** String **gender**=**"男"**;  
  
 @NotNull(message = **"生日不能为空"**)*//JSR303  
// @Temporal(TemporalType.DATE)* @DateTimeFormat(pattern=**"yyyy-MM-dd"**) *//入参的格式化* @JsonFormat(pattern = **"yyyy-MM-dd"**) *//出参的格式化* @Column(name = **"birthday"**, nullable = **false**)  
 **private** Date **birthday**=**new** Date();  
  
 @Email(message = **"邮箱格式不正确"**)*//JSR303* @Size(min = 0, max = 50, message = **"邮箱长度不能超过50个字符"**)*//JSR303* @Column(name = **"email"**)  
 **private** String **email**;  
  
 @Length(min=6, max=40, message = **"密码长度必须大于6个字符"**)  
 @Column(name = **"password"**, length = 40)  
 **private** String **password**;  
  
 @NotNull(message = **"账户锁定不能为空"**)*//JSR303* @Column(name = **"account\_locked"**, nullable = **false**, length = 1)  
 **private** String **accountLocked**=**"N"**;  
  
 @NotNull(message = **"创建时间不能为空"**)*//JSR303* @DateTimeFormat(pattern=**"yyyy-MM-dd HH:mm:ss"**) *//入参的格式化* @JsonFormat(pattern = **"yyyy-MM-dd HH:mm:ss"**) *//出参的格式化  
// @Temporal(TemporalType.DATE)* @Column(name = **"create\_time"**, nullable = **false**)  
 **private** Date **createTime** = **new** Date();  
  
 @NotNull(message = **"创建者不能为空"**)*//JSR303*  
 @Column(name = **"create\_by"**, nullable = **false**, length = 60)  
 **private** String **createBy** = **"admin"**;  
  
 @NotNull(message = **"修改时间不能为空"**)*//JSR303* @DateTimeFormat(pattern=**"yyyy-MM-dd HH:mm:ss"**) *//入参的格式化* @JsonFormat(pattern = **"yyyy-MM-dd HH:mm:ss"**) *//出参的格式化  
// @Temporal(TemporalType.DATE)* @Column(name = **"update\_time"**, nullable = **false**)  
 **private** Date **updateTime** = **new** Date();  
  
 @NotNull(message = **"修改者不能为空"**)*//JSR303* @Column(name = **"update\_by"**, nullable = **false**, length = 60)  
 **private** String **updateBy** = **"admin"**;  
  
}

POST提交方法中校验并判断是否有校验错误，并使用Model对象将错误信息返回前台。

@PostMapping(**"/user/userSave"**)  
*// @ResponseBody* **public** String userPost(@Valid @ModelAttribute(**"user"**) User user, BindingResult br, Model m){  
 **if** (br.hasErrors()) {  
 List<ObjectError> allErrors = br.getAllErrors();  
 List<String> errorList = **new** ArrayList<>();  
 **for** (ObjectError allError : allErrors) {  
 errorList.add(allError.getDefaultMessage());  
 *//System.out.println("错误msg："+allError.getDefaultMessage());* }  
 *//System.out.println("错误"+listError.toString());* m.addAttribute(**"errorList"**, errorList.toString());  
 *//System.out.println("\*\*\*\*\*\*\*\*userId:"+user.getId());* **if** (user.getId() != **null**){  
 **return "views/useredit"**;}  
 **else**{  
 **return "views/userinsert"**;  
 }  
 }  
  
 user.setCreateBy(**"admin"**);  
 user.setUpdateBy(**"admin"**);  
 System.***out***.println(**"保存:"**+user.toString());  
 ***log***.info(**"保存:"**+user.toString());  
 **userRepository**.save(user);  
 **return "views/userlist"**;  
 }

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**title**>用户编辑</**title**>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"**/>  
  
 <**script th:src="@{/js/jquery.min.js}"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
*<!-- <script th:src="@{/js/popper.min.js}"></script>-->* <**style**>  
 [**readonly**=**"readonly"**] {  
 **pointer-events**: **none**;  
 }  
 **body**{  
 **background-color**: **#FFFFFF**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div class="container"**>  
 <**h2 class="text-center col-4"**>用户编辑</**h2**>  
 <**form th:action="@{/user/userSave}" th:object="${user}" method="post"**>  
 <**div class="form-inline"** >  
 <**span class="input-group-text col-1"**>用户ID</**span**>  
 <**input type="text" class="form-control col-3" id="userid" readonly  
 th:field="${user.id}"**>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户名称</**span**>  
 <**input type="text" class="form-control col-3" placeholder="输入用户姓名" id="username" autofocus  
 th:field="${user.userName}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户呢称</**span**>  
 <**input type="text" class="form-control col-3" placeholder="输入用户姓名" id="nickname"  
 th:field="${user.nickName}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户性别</**span**>  
 <**select class="form-control col-3" id="gender" placeholder="选择性别"  
 th:field="${user.gender}" th:value="${user.gender}"**>  
 <**option th:selected="${user.gender}=='男'" value="男"**>男</**option**>  
 <**option th:selected="${user.gender}=='女'" value="女"**>女</**option**>  
 </**select**>  
*<!-- <input name="gender" type="radio" th:selected="${user.gender}=='男'" th:field="${user.gender}" value="男">男</input>&nbsp;&nbsp;-->  
<!-- <input name="gender" type="radio" th:selected="${user.gender}=='女'" th:field="${user.gender}" value="女">女</input>-->* </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>出生日期</**span**>  
 <**input type="date" class="form-control col-3" placeholder="选择日期" id="birthday"  
 th:field="${user.birthday}" th:value="${#dates.format(user.birthday,'yyyy-MM-dd')}"**/>  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>电子邮件</**span**>  
 <**input type="email" class="form-control col-3" placeholder="输入电子邮件" id="email"  
 th:field="${user.email}"** />  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>用户密码</**span**>  
 <**input type="password" class="form-control col-3" placeholder="输入用户密码" id="pwd"  
 th:field="${user.password}"** />  
 </**div**>  
 <**div class="form-inline"**>  
 <**span class="input-group-text col-1"**>账户锁定</**span**>  
*<!-- <input name="accountlocked" type="radio" th:selected="${user.accountLocked}+'es'=='Yes'" th:field="${user.accountLocked}" value="Y">Yes</input>&nbsp;&nbsp;-->  
<!-- <input name="accountlocked" type="radio" th:selected="${user.accountLocked}+'o'=='No'" th:field="${user.accountLocked}" value="N">No</input>-->* <**select class="form-control col-3" id="accountlocked" placeholder="选择账户是否锁定"  
 th:field="${user.accountLocked}" th:value="${user.accountLocked}"**>  
 <**option th:selected="${user.accountLocked}=='Y'" value="Y"**>Y</**option**>  
 <**option th:selected="${user.accountLocked}=='N'" value="N"**>N</**option**>  
 </**select**>  
 </**div**>  
 <**button class="col-4 btn btn-info btn-block" type="submit"**>提交</**button**>  
 </**form**>  
 <**div name="errormessage" class="col-4" th:text="${errorList}" style="color**: **red"**/>  
</**div**>  
</**body**>  
</**html**>

用户新增页面与用户编辑页面的代码基本相同，因此不再重复。

### 6.可用的 JSR303注解

空检查

@Null(message=) 验证对象是否为null

@NotNull(message=) 验证对象是否不为null, 无法查检长度为0的字符串

@NotBlank(message=) 检查约束字符串是不是Null还有被Trim的长度是否大于0,只对字符串,且会去掉前后空格.

@NotEmpty(message=) 检查约束元素是否为NULL或者是EMPTY.

Booelan检查

@AssertTrue(message=) 验证 Boolean 对象是否为 true

@AssertFalse(message=) 验证 Boolean 对象是否为 false

长度检查

@Size(min=, max=,message=) 验证对象（Array,Collection,Map,String）长度是否在给定的范围之内

@Length(min=, max=,message=) 验证字符串长度是否在给定的范围之内

日期检查

@Past 验证 Date 和 Calendar 对象是否在当前时间之前

@Future 验证 Date 和 Calendar 对象是否在当前时间之后

@Pattern 验证 String 对象是否符合正则表达式的规则

数值检查，建议使用在Stirng,Integer类型，不建议使用在int类型上，因为表单值为“”时无法转换为int，但可以转换为Stirng为"",Integer为null

@Min 验证 Number 和 String 对象是否大等于指定的值

@Max 验证 Number 和 String 对象是否小等于指定的值

@DecimalMax 被标注的值必须不大于约束中指定的最大值. 这个约束的参数是一个通过BigDecimal定义的最大值的字符串表示.小数存在精度

@DecimalMin 被标注的值必须不小于约束中指定的最小值. 这个约束的参数是一个通过BigDecimal定义的最小值的字符串表示.小数存在精度

@Digits 验证 Number 和 String 的构成是否合法

@Digits(integer=,fraction=,message=) 验证字符串是否是符合指定格式的数字，interger指定整数精度，fraction指定小数精度。

@Range(min=, max=,message=) 验证值是否在给定范围内

@Range(min=10000,max=50000,message="range.bean.wage")

private BigDecimal wage;

@Valid 递归的对关联对象进行校验, 如果关联对象是个集合或者数组,那么对其中的元素进行递归校验,如果是一个map,则对其中的值部分进行校验.(是否进行递归验证)

@CreditCardNumber信用卡验证

@Email 验证是否是邮件地址，如果为null,不进行验证，算通过验证。

@ScriptAssert(lang= ,script=, alias=,message=)

@URL(protocol=,host=, port=,regexp=, flags=,message=)

### 7.运行并访问



直接返回BindingResult类型的校验错误信息，内容太多，看起来太乱，因此我们必须对返回的校验错误信息经行处理，处理的代码如下：

List<ObjectError> allErrors = br.getAllErrors();  
 List<String> errorList = **new** ArrayList<>();  
 **for** (ObjectError allError : allErrors) {  
 errorList.add(allError.getDefaultMessage());}m.addAttribute(**"errorList"**, errorList.toString());

处理后页面显示如下,看起来简洁很多。



## 第二十三章 JPA动态查询

在日常工作中，Spring Data [JPA](https://so.csdn.net/so/search?q=JPA&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/liuyunyihao/article/details/_blank)的使用给我们带来了极大的方便，但是实际业务中很多场景需要支持动态查询。比如前端查询功能提供了很多查询条件，用户可以根据一部分条件进行查询，那么后端就需要支持可配置的查询服务。Spring Data JPA是支持动态查询的，需要我们的repo继承JpaSpecificationExecutor接口，使用的时候传入相应参数即可。

### 1.修改接口

之前我们在声明实体类的DAO接口时是这样写的：

@Repository *//DAO层+DTO***public interface** UserRepository **extends** JpaRepository<User,Integer> {

我们现在增加一个接口：

@Repository *//DAO层+DTO***public interface** UserRepository **extends** JpaRepository<User,Integer> , JpaSpecificationExecutor {

### 2.在Service层声明动态查询接口

List<User> selectUser(User user);

### 3.在Service层的impl包中实现动态查询接口

*//动态查询例子* **public** List<User> selectUser(User user) {  
 Specification<User> query = **new** Specification<User>() {  
 @Override  
 **public** Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {  
 List<Predicate> predicates = **new** ArrayList<>();  
 **if**(StringUtils.*isNotEmpty*(user.getId())){  
 predicates.add(cb.equal(root.get(**"id"**),user.getId()));  
 }  
 **if**(StringUtils.*isNotEmpty*(user.getGender())){  
 predicates.add(cb.equal(root.get(**"gender"**),user.getGender()));  
 }  
 **if**(StringUtils.*isNotEmpty*(user.getUserName())){  
 predicates.add(cb.like(root.get(**"userName"**),**"%"**+user.getUserName()+**"%"**));  
 }  
 **if**(StringUtils.*isNotEmpty*(user.getNickName())){  
 predicates.add(cb.like(root.get(**"nickName"**),**"%"**+user.getNickName()+**"%"**));  
 }  
*// if(StringUtils.isNotEmpty(user.getBirthday())){//显示成中文格式，这里就不兼容  
// predicates.add(cb.like(root.get("birthday"), (Expression<String>) user.getBirthday()));  
// }* **if**(StringUtils.*isNotEmpty*(user.getEmail())){  
 predicates.add(cb.like(root.get(**"email"**),**"%"**+user.getEmail()+**"%"**));  
 }  
 **if**(StringUtils.*isNotEmpty*(user.getAccountLocked())){  
 predicates.add(cb.equal(root.get(**"accountLocked"**),user.getAccountLocked()));  
 }  
 **return** cb.and(predicates.toArray(**new** Predicate[predicates.size()]));*//and取交集，or取合集* }  
 };  
 **return userRepository**.findAll(query);  
 }

### 4.测试动态查询接口

测试时发现报错，经查阅资料，是因为有多个启动类，下面指定启动类问题解决。

@SpringBootTest**(classes = Demo5Application.class)**  
**class** Demo5ApplicationTests {  
 @Autowired  
 UserService **userService**;  
  
 @Test  
 **void** contextLoads() {  
  
 }  
 @Test  
 **void** test1(){  
 User user = **new** User();  
 *//user.setUserName("z");* user.setAccountLocked(**"Y"**);  
 List<User> us = **userService**.selectUser(user);  
 System.***out***.println(us.toString());  
 System.***out***.println(Arrays.*toString*(us.toArray()));  
 }  
  
}

## 第二十四章 用户管理页面

### 管理页面

UserController.java中管理页面查询数据的控制器方法：

@GetMapping(**"/user/getUsers"**)  
**public** String usersQuery(Model m){  
 *//查询时排序  
 //Sort sort = new Sort(Sort.Direction.DESC,"id");//旧版* Sort sort = Sort.*by*(Sort.Direction.***DESC***,**"id"**); *//新版* List<User> listUser = **userRepository**.findAll(sort);  
 m.addAttribute(**"listUser"**,listUser);  
 **return "views/userlist"**;  
}

Listuser.html页面代码：

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
  
 <**title**>用户管理</**title**>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"**/>  
  
 <**script th:src="@{/js/jquery.min.js}"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
</**head**>  
<**body class="bg-light"**>  
<**div class="container"**>  
  
 <**div align="center"** >  
 <**h2 class="text-center"**>用户管理</**h2**>  
 <**table class="col-md-12 table table-sm table-striped table-bordered table-hover table-condensed"** >  
 <**thead class="table-dark"**>  
 <**tr class="text-center"**>  
 <**th class="col-md-1"**>NO</**th**>  
 <**th class="col-md-1"**>ID</**th**>  
 <**th class="col-md-1"**>用户名称</**th**>  
 <**th class="col-md-1"**>用户呢称</**th**>  
 <**th class="col-md-1"**>性别</**th**>  
 <**th class="col-md-2"**>出生日期</**th**>  
 <**th class="col-md-2"**>电子邮件</**th**>  
 <**th class="col-md-1"**>锁定</**th**>  
 <**th class="col-md-2"**>操作</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr th:each="u:${listUser}"** >  
 <**td class="text-center" th:text="${#ids.seq('')}"**></**td**>  
 <**td class="text-center" th:text="${u.id}"**>1</**td**>  
 <**td th:text="${u.userName}"**>lisi</**td**>  
 <**td th:text="${u.nickName}"**>李四</**td**>  
 <**td class="text-center" th:text="${u.gender}"**>男</**td**>  
 <**td class="text-center" th:text="${#dates.format(u.birthday,'yyyy-MM-dd')}"**>2000-11-12</**td**>  
 <**td th:text="${u.email}"**>lisi@sina.com</**td**>  
 <**td class="text-center" th:text="${u.accountLocked}"**>N</**td**>  
 <**td class="text-center"**>  
 <**button type="button" class="btn1 btn-info btn-sm btn-corner" title="修改"**>修改</**button**>  
 <**button type="button" class="btn1 btn-danger btn-sm btn-corner" title="删除"**>删除</**button**>  
  
*<!-- <a class="btn btn-info btn-sm" >-->  
<!-- <i class="fa fa-bath smaller-30">修改</i>-->  
<!-- </a>-->  
<!-- <a class="btn btn-danger btn-sm" >-->  
<!-- <i class="fa fa-cog smaller-30">删除</i>-->  
<!-- </a>-->* </**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
 </**div**>  
</**div**>  
</**body**>  
</**html**>

### 2.带分页的管理页面

（1）分页原理

* 逻辑分页

即将所有符合条件的数据一次性的打包送到前台，无论是否能全部显示，还是用户采用滚动条，内存过滤，实际数据全部在客户端内存中，这种方式的客户端负担很重，而且大数量的访问客户会拖垮后台服务器，堵死网络。

* 物理分页

这种情况是根据系统设定页面记录大小参数，每次只传页面记录大小参数的数据至前台，对后台服务器、网络、前台客户端负担都很轻，因此这是非常好的设计。

1. 分页实现

* 继承**JpaSpecificationExecutor** 接口

在UserRepository.java类中增加对**JpaSpecificationExecutor** 接口的继承

@Repository *//DAO层+DTO***public interface** UserRepository **extends** JpaRepository<User,Integer> **, JpaSpecificationExecutor** {

* 控制器方法的修改

*/\*\*  
 \** ***@Author*** *zhouqi  
 \** ***@Description*** *//带分页的用户管理  
 \** ***@Date*** *22:44 2022/4/4  
 \** ***@param pageNo*** *\** ***@param m*** *\** ***@return*** *[pageNo, m]  
 \*\*/* @RequestMapping(value = **"/user/page"**)  
 **public** String getUsersByPage(@RequestParam(value=**"pageNo"**,defaultValue = **"1"**,required=**false**) Integer pageNo,Model m){  
 Sort sort = Sort.*by*(Sort.Direction.***DESC***,**"id"**); *//通过用户id进行排序，orderdate 是OrderBean 中的变量，不是数据库中的字段（\*）* Pageable pageable = PageRequest.*of*(pageNo-1, Consts.*pageSize*,sort); *//Consts为系统静态参数类  
 //Pageable pageable = new PageRequest(pageNo-1,Consts.pageSize,sort); //旧版用法* Page<User> pageData = **userRepository**.findAll(pageable);  
  
 System.***out***.println(**"总页数:"**+pageData.getTotalPages());  
 System.***out***.println(**"总记录数:"**+pageData.getTotalElements());  
 System.***out***.println(**"当前页:"**+pageData.getNumber());  
*// for (User u:pageData.getContent()){  
// System.out.println(u.toString());  
// }* m.addAttribute(**"pageData"**,pageData);  
 **return "views/userlistpage"**;  
 }

* 前端页面userlist.html的修改

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**title**>用户管理</**title**>

<**base th:href="${#request.getContextPath()}+'/'"**>

<**script th:src="@{/js/jquery.min.js}" type="text/javascript"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}" type="text/javascript"**></**script**>  
  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/ace.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/font-awesome/css/font-awesome.min.css}"** />  
  
*<!-- <script th:src="@{/js/popper.min.js}"></script>-->*

<**style**>  
 [**readonly**=**"readonly"**] {  
 **pointer-events**: **none**;  
 }  
 **body**{  
 **background-color**: **#FFFFFF**;  
 }  
</**style**>

<**body**>  
<**div class="container"**>  
  
 <**div align="center"** >  
 <**h2 class="text-center"**>用户管理</**h2**>  
 <**span style="float**:**right"**>  
 <**a th:href="@{/user/userInsert}" class="btn btn-info btn-sm"** >  
 <**i class="fa fa-pencil smaller-30 text-dark"**>新增</**i**>  
 </**a**>  
 <**a id="delBtnBatch" class="btn btn-warning btn-sm" style="color**:**#FFFFFF**;**"**>  
 <**i class="fa fa-times smaller-30 text-dark" onclick="***delRecs*()**"**>批量删除</**i**>  
 </**a**>  
 <**a th:href="@{/admin/main}" class="btn btn-info btn-sm"** >  
 <**i class="fa fa-university text-dark"**>返回</**i**>  
 </**a**>  
 </**span**>  
 <**form id="userForm" th:action="@{/user/userBatchDelete}" method="post"**>  
 <**table class="col-sm-12 table table-sm table-striped table-bordered table-hover table-condensed"** >  
 <**thead class="table-dark"**>  
 <**tr class="text-center"**>  
 <**th class="col-sm-1"**>选择<**input type="checkbox" value="反选" id="revSel"**/></**th**>  
 <**th class="col-sm-1"**>NO</**th**>  
 <**th class="col-sm-1"**>ID</**th**>  
 <**th class="col-sm-1"**>用户名称</**th**>  
 <**th class="col-sm-1"**>用户呢称</**th**>  
 <**th class="col-sm-1"**>性别</**th**>  
 <**th class="col-sm-1"**>出生日期</**th**>  
 <**th class="col-sm-2"**>电子邮件</**th**>  
 <**th class="col-sm-1"**>账户锁定</**th**>  
 <**th class="col-sm-2"**>操作</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr th:each="u:${pageData}"**>  
 <**td class="text-center"**><**input type="checkbox" name="recsel" th:checked="false" th:value="${u.id}"**/></**td**>  
 <**td class="text-center" th:text="${#ids.seq('')}"**></**td**>  
 <**td class="text-center" th:text="${u.id}"**>1</**td**>  
 <**td th:text="${u.userName}"**>lisi</**td**>  
 <**td th:text="${u.nickName}"**>李四</**td**>  
 <**td class="text-center" th:text="${u.gender}"**>男</**td**>  
 <**td class="text-center" th:text="${#dates.format(u.birthday,'yyyy-MM-dd')}"**>2000-11-12</**td**>  
 <**td th:text="${u.email}"**>lisi@sina.com</**td**>  
 <**td class="text-center" th:text="${u.accountLocked}"**>N</**td**>  
 <**td class="text-center"**>  
*<!-- <button type="button" class="btn btn-info btn-sm btn-corner" title="修改">修改</button>-->* <**a class="btn btn-info btn-sm" th:href="@{/user/userEdit/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}"**>  
 <**i class="fa fa-bath smaller-30 text-dark"** >修改</**i**>  
 </**a**>  
*<!-- <a id="delBtna" class="btn btn-warning btn-sm" th:href="@{/user/userDelete/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}">-->  
<!-- <i class="fa fa-cog smaller-30 text-dark">删除</i>-->  
<!-- </a>-->* <**a class="btn btn-warning btn-sm" th:onclick="delcfm([[@{/user/userDelete/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}]])"**>  
 <**i class="fa fa-times smaller-30 text-dark"**>  
 删除  
 </**i**>  
 </**a**>  
*<!-- <button type="button" class="btn btn-danger btn-sm btn-corner" title="删除" th:onclick="delcfm([[@{/user/userDelete/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}]])">删除</button>-->* </**td**>  
 </**tr**>  
 <**tr**>  
 <**td colspan="12"** >  
 <**div**>  
 <**p** >当前<**span th:text="${pageData.getNumber()} + 1"**></**span**>页,总<**span th:text="${pageData.totalPages}"**></**span**>页,  
 共<**span th:text="${pageData.totalElements}"**></**span**>条记录  
 <**span style="float**:**right**;**margin-right**:20**px"**>  
 <**a th:href="@{/user/page}"**>首页</**a**>**&nbsp;&nbsp;&nbsp;&nbsp;** <**a th:href="@{/user/page(pageNo = ${pageData.hasPrevious()} ? ${pageData.getNumber() } : 1)}"**>上一页</**a**>**&nbsp;&nbsp;&nbsp;&nbsp;** <**a th:href="@{/user/page(pageNo = ${pageData.hasNext()} ? ${pageData.getNumber()} + 2 : ${pageData.totalPages})}"**>下一页</**a**>**&nbsp;&nbsp;&nbsp;&nbsp;** <**a th:href="@{/user/page(pageNo = ${pageData.totalPages})}"**>尾页</**a**>**&nbsp;&nbsp;&nbsp;&nbsp;** </**span**>  
 </**p**>  
 </**div**>  
 </**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
 </**form**>  
 </**div**>  
</**div**>  
*<!-- 信息删除确认 -->*<**div class="modal fade" id="delcfmModel"**>  
 <**div class="modal-dialog"**>  
 <**div class="modal-content message\_align"**>  
 <**div class="modal-header"**>  
 <**button type="button" class="close" data-dismiss="modal" aria-label="Close"**><**span aria-hidden="true"**>×</**span**></**button**>  
 <**h4 class="modal-title"** >提示信息</**h4**>  
 </**div**>  
 <**div class="modal-body"**>  
 <**p**>您确认要删除吗？</**p**>  
 </**div**>  
 <**div class="modal-footer"**>  
 <**input type="hidden" id="url"**/>  
 <**button type="button" class="btn btn-default" data-dismiss="modal"**>取消</**button**>  
 <**a onclick="***urlSubmit*()**" class="btn btn-success" data-dismiss="modal"**>确定</**a**>  
 </**div**>  
 </**div**>*<!-- /.modal-content -->* </**div**>*<!-- /.modal-dialog -->*</**div**>*<!-- /.modal -->*<**script th:inline="javascript"** >  
 **function** *urlSubmit*(){  
 **var** url=**$**.trim(**$**(**"#url"**).val());*//获取会话中的隐藏属性URL* ***window***.**location**.**href**=url;  
 }  
  
 **function** *delcfm*(url) {  
 **$**(**'#url'**).val(url);*//给会话中的隐藏属性URL赋值* **$**(**'#delcfmModel'**).**modal**();  
 }  
  
 *//批量删除按钮的表单提交方法* **function** *delRecs*(){  
 *//alert("ok");  
 //document.getElementById("userForm").submit();* **$**(**"#userForm"**).submit();  
 }  
  
 *//$(function() { //$(document).ready(function(){  
 //删除的确认方法* **$**(**"#delBtna"**).click(**function**(event){  
 **return** *confirm*(**"您确定要删除此用户吗？"**);  
 });  
  
 *// document.getElementById("delBtna").onclick(){  
 //  
 // }  
  
 //删除的确认方法* **$**(**"#delBtnBatch"**).click(**function**(event){  
 **return** *confirm*(**"您确定要删除这些用户吗？"**);  
 });  
  
 *//批量选择反选方法  
 //alert("ok");* **$**(**"#revSel"**,**this**).click(**function**(){  
 **var** allChecked = **this**.**checked**;  
 **$**(**":checkbox"**).each(**function**(){  
 **this**.**checked** = allChecked;  
 })  
 })  
 *// })*</**script**>  
</**body**>  
</**html**>

### 修改和新增

*/\*\*  
 \** ***@Author*** *zhouqi  
 \** ***@Description*** *//本方法用于对用户的新增和修改后的保存  
 \** ***@Date*** *2022/4/5 12:31  
 \** ***@param user*** *\** ***@param br*** *\** ***@param m*** *\** ***@param ra*** *\** ***@return*** *[user, br, m, ra]  
\*\*/*@PostMapping(**"/user/userSave"**) *//@Validated***public** String userPost(@Valid @ModelAttribute(**"user"**) User user, BindingResult br,Model m,RedirectAttributes ra){  
 **if** (br.hasErrors()){  
 *//m.addAttribute("errorList",br.getAllErrors()) ;* List<ObjectError> allErrors = br.getAllErrors();  
 List<String> errorList = **new** ArrayList<>();  
 **for** (ObjectError allError : allErrors) {  
 errorList.add(allError.getDefaultMessage());  
 System.***out***.println(**"错误msg："**+allError.getDefaultMessage());  
 }  
 m.addAttribute(**"errorList"**,errorList.toString()) ;  
 **if** (user.getId() != **null**){ **return "views/useredit"** ;}  
 **else**{  
 **return "views/userinsert"** ;  
 }  
 }  
  
 user.setUpdateTime(**new** Date());  
 user.setUpdateBy(**"admin"**);  
 System.***out***.println(**"user"**+user.toString());  
 ***log***.info(**"保存："**+user.toString());  
 **userRepository**.save(user);  
  
 ra.addAttribute(**"pageNo"**,1); //必须初始化页面  
 **return "views/userlistpage"**;  
}

### 4.RedirectAttributes类

RedirectAttributes 是Spring mvc 3.1版本之后出来的一个功能，专门用于重定向之后还能带参数跳转的的工具类。

有两种带参的方式：

第一种：

redirectAttributes.addAttributie(“prama”,value); 这种方法相当于在重定向链接地址追加传递的参数，例如:

redirectAttributes.addAttributie("prama1",value1);

redirectAttributes.addAttributie("prama2",value2);

return:"redirect：/path/list";

以上重定向的方法等同于 ：

return:"redirect：/path/list？prama1=value1&prama2=value2 " ，注意这种方法直接将传递的参数暴露在链接地址上，非常的不安全，慎用。

第二种：

redirectAttributes.addFlashAttributie(“prama”,value); 这种方法是隐藏了参数，链接地址上不直接暴露，但是能且只能在重定向的 “页面” 获取prama参数值。其原理就是放到session中，session在跳到页面后马上移除对象。如果是重定向一个controller中是获取不到该prama属性值的。除非在controller中用(@RequestPrama(value = “prama”)String prama)注解，采用传参的方式。页面获值例如：

redirectAttributes.addFlashAttributie("prama1",value1);

redirectAttributes.addFlashAttributie("prama2",value2);

return:"redirect：/path/list" ;

### 5.删除

（1）页面删除的确认

*//删除的确认方法***$**(**"#delBtna"**).click(**function**(event){  
 **return** *confirm*(**"你确定要删除此用户吗？"**);  
});

（2）控制器中的删除方法

*/\*\*  
 \** ***@Author*** *zhouqi  
 \** ***@Description*** *//删除用户记录  
 \** ***@Date*** *2022/4/4 23:00  
 \** ***@param id*** *\** ***@param pageNo*** *\** ***@param ra*** *专用的往定向跳转的页面传输局的类  
 \** ***@return*** *[id, pageNo, ra]  
\*\*/*@RequestMapping(**"/user/userDelete/{id}"**)  
**public** String deleteUser(@PathVariable(**"id"**) Integer id,@RequestParam(value=**"pageNo"**,defaultValue = **"1"**,required=**false**) Integer pageNo,  
 RedirectAttributes ra){  
 System.***out***.println(**"id:"**+id+**",pageNo:"**+pageNo);  
 *//删除用户* **userRepository**.deleteById(id);  
 ra.addAttribute(**"pageNo"**,pageNo);  
 *//return "redirect:/user/page?pn="+pageNo;* **return "redirect:/user/page"**;  
}

### 批量删除

批量删删的步骤：

（1）在<table>表格中增加checkbox选择列

<**td class="text-center"**><**input type="checkbox" name="recsel" th:checked="false" th:value="${u.id}"**/></**td**>

注意：在此标签中我们埋伏了user.id属性在标签中，这个很重要，另外**name="recsel"**，这个值也很关键。

（2）在<table>表格外增加<form>表单

<**form id="userForm" th:action="@{/user/userBatchDelete}" method="post"**>

...

<**/form**>

注意：上述表格本身没有提交按钮，提交动作是我们外部按钮触发的。

（3）在表单之外增加“批量删除”按钮

<**a class="btn btn-danger btn-sm" style="color**:**#FFFFFF**;**"**>  
 <**i class="fa fa-times smaller-30" onclick="***delRecs*()**"**>批量删除</**i**>  
</**a**>

（4）在页面的<script>...</script>中增加delRecs方法

**function** *delRecs*(){  
 *//alert("ok");  
 //document.getElementById("userForm").submit();* **$**(**"#userForm"**).submit();  
}

（5）在控制器方法中增加“**/user/userBatchDelete**”路径方法

*/\*\*  
 \** ***@Author*** *zhouqi  
 \** ***@Description*** *//批量删除  
 \** ***@Date*** *2022/4/5 13:19  
 \** ***@param request*** *\** ***@param pageNo*** *\** ***@param ra*** *\** ***@return*** *java.lang.String  
\*\*/*@RequestMapping(value = **"/user/userBatchDelete"**, method = RequestMethod.***POST***)  
**public** String delBatchForm(HttpServletRequest request, @RequestParam(value=**"pageNo"**,defaultValue = **"1"**) Integer pageNo,  
 RedirectAttributes ra) {  
 String[] sels = request.getParameterValues(**"recsel"**);  
 **if** (sels != **null**) {  
 *//批量删除用户数据* **for**(String s:sels){  
 Integer uid = Integer.*parseInt*(s);  
 *//System.out.println("uid:"+uid);* **userRepository**.deleteById(uid);  
 }  
 }**else**{  
 System.***out***.println(**"没有选择数据"**);  
 }  
  
 ra.addAttribute(**"pageNo"**,pageNo);  
 **return "redirect:/user/page"**;  
}

1. 增加批量选择的反选方法

*//批量选择反选方法***$**(**function**() {  
 *//alert("ok");* **$**(**"#revSel"**,**this**).click(**function**(){  
 **var** allChecked = **this**.**checked**;  
 **$**(**":checkbox"**).each(**function**(){  
 **this**.**checked** = allChecked;  
 })  
 })  
})

注意：[JQuery](https://so.csdn.net/so/search?q=JQuery&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_41042740/article/details/_blank) 的代码我们通常会包裹在一个$(function(){})函数中，jquery 的$(function(){})也就是$(document).ready(function(){})的简写

### 7.删除前确认

对删除数据这样的危险操作一定要增加确认环节。在本教程中我们采用JavaScript的默认确认方法来询问，用户如果觉得弹出的确认窗口简单、不美观，可以下载jquery-confirm.min.js专业确认包。但这个包中的jquery与BootStrap包使用的jquery版本不一致，所以我们倾向于使用BootStrap或layUI的模态确认窗口，只是稍显复杂。

* 采用JavaScript确认：

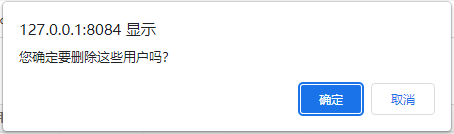
（1）标签声明id

<**a id="delBtna" class="btn btn-warning btn-sm" th:href="@{/user/userDelete/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}"**>  
 <**i class="fa fa-cog smaller-30 text-dark"**>删除</**i**>  
</**a**>

<**a id="delBtnBatch" class="btn btn-warning btn-sm" style="color**:**#FFFFFF**;**"**>  
 <**i class="fa fa-times smaller-30 text-dark" onclick="***delRecs*()**"**>批量删除</**i**>  
</**a**>

（2）在本地端脚本中增加确认方法

<**script type="text/javascript"** >  
 *//删除的确认方法* **$**(**"#delBtna"**).click(**function**(event){  
 **return** *confirm*(**"您确定要删除此用户吗？"**);  
 });  
  
 *//删除的确认方法* **$**(**"#delBtnBatch"**).click(**function**(event){  
 **return** *confirm*(**"您确定要删除这些用户吗？"**);  
 });



* 采用BootStrap确认：

（1）弹出的模态窗口

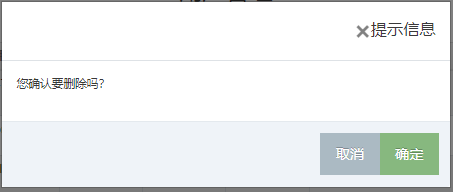
*<!-- 信息删除确认 -->*<**div class="modal fade" id="delcfmModel"**>  
 <**div class="modal-dialog"**>  
 <**div class="modal-content message\_align"**>  
 <**div class="modal-header"**>  
 <**button type="button" class="close" data-dismiss="modal" aria-label="Close"**><**span aria-hidden="true"**>×</**span**></**button**>  
 <**h4 class="modal-title"**>提示信息</**h4**>  
 </**div**>  
 <**div class="modal-body"**>  
 <**p**>您确认要删除吗？</**p**>  
 </**div**>  
 <**div class="modal-footer"**>  
 <**input type="hidden" id="url"**/>  
 <**button type="button" class="btn btn-default" data-dismiss="modal"**>取消</**button**>  
 <**a onclick="***urlSubmit*()**" class="btn btn-success" data-dismiss="modal"**>确定</**a**>  
 </**div**>  
 </**div**>*<!-- /.modal-content -->* </**div**>*<!-- /.modal-dialog -->*</**div**>*<!-- /.modal -->*

（2）删除确认方法和确认后提交方法

**function** *urlSubmit*(){  
 **var** url=**$**.trim(**$**(**"#url"**).val());*//获取会话中的隐藏属性URL* ***window***.**location**.**href**=url;  
}  
  
**function** *delcfm*(url) {  
 **$**(**'#url'**).val(url);*//给会话中的隐藏属性URL赋值* **$**(**'#delcfmModel'**).**modal**();  
}

1. 标签上调用BootStrap删除确认的模态窗口

<**a class="btn btn-warning btn-sm" th:onclick="delcfm([[@{/user/userDelete/{id}(id=${u.id},pageNo=${pageData.getNumber()} + 1)}]])"**>  
 <**i class="fa fa-cog smaller-30 text-dark"**>删除</**i**>  
</**a**>



### 直接访问页面还是强制跳转

* 编辑和新增保存后
* 删除和批量删除后

在以上两种情况下要求采用强制跳转，因为系统始终要保持一个入口，所有对一个页面的访问只能有一个访问路径，这也是后续增加权限访问的要求，同时也防止直接跳转页面，页面没初始化。

@PostMapping(**"/user/userSave"**) *//@Validated***public** String userPost(@Valid @ModelAttribute(**"user"**) User user, BindingResult br,Model m,@RequestParam(value=**"pageNo"**,defaultValue = **"1"**) Integer pageNo,RedirectAttributes ra){

...

ra.addAttribute(**"pageNo"**,pageNo);//对跳转的页面或路径传参  
**return "redirect:/user/page"**; *//"views/userlistpage";直接访问页面是错误的，因为没有初始化数据*

### 9.运行并访问

http://127.0.0.1:8084/aa/user/page



## 第二十五章 集成登录页面

### 1.控制器方法

@GetMapping(**"/admin/login"**)  
 **public** String login(@ModelAttribute(**"user"**) User user, @RequestParam(value = **"message"**,defaultValue = **""**,required = **false**) String message,Model m){ *//采用请求参数收取post提交后的验证错误信息，初次访问是空串*  
  
 **if** (!**""**.equals(message)){  
 m.addAttribute(**"message"**,message);  
 }  
 **return "login"**;  
 }  
  
 @PostMapping(**"/admin/loginCheck"**)  
 **public** String loginVerf(@ModelAttribute(**"user"**) User user, RedirectAttributes ra){User u = **userRepository**.findByUserNameAndPassword(user.getUserName(),user.getPassword());  
 **if** (u != **null**){  
 **return "redirect:/user/page"**;  
 }  
*// User u = new User(); //利用动态查询在Service层封装用户名和口令验证方法  
// List<User> users = userService.getByUsernameAndPassword(user.getUserName(),user.getPassword());  
// if (users.size() > 0 ){  
// return "redirect:/user/page";  
// }  
 //强制跳转时采用RedirectAttributes类传递验证错误信息* ra.addAttribute(**"message"**,**"账户或密码错误"**);

**return "redirect:/admin/login"**;  
 }

注意：

（1）我们有一个规定，每个页面只能有一个路径请求，其它页面需要访问这个页面，都需要通过该页面的专用路径去访问，不能越俎代庖。这就需要用到强制跳转。

（2）强制跳转会打断数据链，因此强制跳转如果要传递信息、变量、实例对象需要使用RedirectAttributes类。

### 登录页面

（1）最基本的登录页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>登录</**title**>  
</**head**>  
<**body**>  
 <**h1**>用户登录</**h1**><**hr**/>  
 <**form th:action="@{/user/loginCheck}" th:object="${user}" method="post"**>  
 用户姓名：<**input name="userName" th:field="${user.username}" autofocus maxlength="60"**></**input**><**br**/>  
 用户口令：<**input name="password" th:field="${user.password}" th:value="${user.password}" type="password"**></**input**><**br**/>  
 <**input type="submit" value="提交"**/>  
*<!-- <button class="btn btn-info btn-block sm-input" style="margin-left: 30px" type="submit">提交</button>-->* </**form**>  
 <**div name="message" th:text="${message}" style="color**: **red"**/>  
</**body**>  
</**html**>

（2）使用AdminEx的Css类进行修饰后的登录页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**title**>登录</**title**>  
 <**link th:href="@{/css/style.css}" rel="stylesheet"**>  
 <**link th:href="@{/css/style-responsive.css}" rel="stylesheet"**>  
 <**link th:href="@{/font-awesome/css/font-awesome.min.css}" rel="stylesheet"** />  
 </**head**>  
<**body class="login-body"**>  
  
<**div class="container"**>  
 <**form class="form-signin" th:action="@{/admin/loginCheck}" th:object="${user}" method="post"**>  
 <**div class="form-signin-heading text-center"**>  
 <**h1 class="sign-title"**>用户登录</**h1**>  
 <**img th:src="@{/images/login-logo.png}" alt=""**/>  
 </**div**>  
 <**div class="login-wrap"**>  
 <**label class="form-text"**>用户名称：</**label**><**input class="form-control" placeholder="User ID" autofocus type="text" name="username" th:field="${user.userName}"** /><**br**/>  
 <**label class="form-text"**>用户密码：</**label**><**input class="form-control" placeholder="Password" type="password" name="password" th:field="${user.password}"** /><**br**/>  
  
 <**button class="btn btn-login btn-block" style="width**:236**px**;**height**:45**px" type="submit"**>  
 <**i class="fa fa-check"**></**i**>  
 </**button**>  
  
 <**div class="registration"**>  
 还没注册?  
 <**a class="" href="#"**>  
 注册  
 </**a**>  
 </**div**>  
 <**label class="checkbox"**>  
 <**input type="checkbox" value="remember-me"**> 记住我  
 <**span class="pull-right"**>  
 <**a data-toggle="modal" href="#myModal"**> 忘记口令?</**a**>  
 </**span**>  
 </**label**>  
 </**div**>  
 <**div name="message" th:text="${message}" style="color**: **red"**/>  
 </**form**>  
  
</**div**>  
<**script th:src="@{/js/jquery.min.js}"**></**script**>  
<**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
<**script th:src="@{/js/modernizr.min.js}"**></**script**>  
</**body**>  
</**html**>

### 3.登录信息保存Session中

我们需要将登录验证检查后正确的登录信息保存到Session中，在需要验证的页面查看是否存在登录信息，如果不存在，要求访问者回到登录页面。为此我们修改控制层登录验证的路径方法，加入登录信息保存的语句，为了保存到Session中，我们又引入了HttpServletRequest类。

@PostMapping(**"/admin/loginCheck"**)  
 **public** String loginVerf(@ModelAttribute(**"user"**) User user, RedirectAttributes ra, HttpServletRequest request){  
  
*// if ("admin".equals(user.getUserName()) && "123456".equals(user.getPassword()))  
// {  
// return "welcome";  
// }* User lUser = **userRepository**.findByUserNameAndPassword(user.getUserName(),user.getPassword());  
 **if** (lUser != **null**){  
 *//把正确的登录信息放入Session中* request.getSession().setAttribute(**"loginUser"**, lUser);  
 **return "redirect:/user/page"**;  
 }  
*// User u = new User();  
// List<User> users = userService.getByUsernameAndPassword(user.getUserName(),user.getPassword());  
  
// if (users.size() > 0 ){  
// return "redirect:/user/page";  
// }* ra.addAttribute(**"message"**,**"账户或密码错误"**);  
 **return "redirect:/admin/login"**;  
 }

### 取出Session中的登录信息

还记得我们新增用户和修改用户时，数据最终保存前，用户属性里有一个属性“updateby”，该属性需要填入当前登录用户的用户名，如果用户是正确登录的，那么Session中一定有登录信息，我们正好取出来，赋给“updateby”属性。

下面的代码是控制层用户保存的路径方法，修改后如下：

@PostMapping(**"/user/userSave"**) *//@Validated***public** String userPost(@Valid @ModelAttribute(**"user"**) User user, BindingResult br,Model m,  
 @RequestParam(value=**"pageNo"**,defaultValue = **"1"**,required=**false**) Integer pageNo,  
 RedirectAttributes ra,HttpServletRequest request){  
 **if** (br.hasErrors()){  
 *//m.addAttribute("errorList",br.getAllErrors()) ;* List<ObjectError> allErrors = br.getAllErrors();  
 List<String> errorList = **new** ArrayList<>();  
 **for** (ObjectError allError : allErrors) {  
 errorList.add(allError.getDefaultMessage());  
 System.***out***.println(**"错误msg："**+allError.getDefaultMessage());  
 }  
 m.addAttribute(**"errorList"**,errorList.toString()) ;  
 **if** (user.getId() != **null**){ **return "views/useredit"** ;}  
 **else**{  
 **return "views/userinsert"** ;  
 }  
 }  
 *//从Session中获取用户信息* User lUser= (User)request.getSession().getAttribute(**"loginUser"**);  
 *//判断用户是否登录* **if** (lUser != **null**){  
 **if** (user.getId() == **null**) *//新增用户* {  
 user.setCreateTime(**new** Date());  
 user.setCreateBy(lUser.getUserName());  
 }  
 user.setUpdateBy(lUser.getUserName());  
 }**else**{//非法登录，没有登录信息  
 *//应强制跳转登录页面，但一个个改太麻烦，我们使用拦截器解决此问题* }  
 user.setUpdateTime(**new** Date());  
 System.***out***.println(**"user"**+user.toString());  
 ***log***.info(**"保存："**+user.toString());  
 **userRepository**.save(user);  
  
 ra.addAttribute(**"pageNo"**,pageNo);  
 *//return "views/userlistpage";* **return "redirect:/user/page"**;  
}

### 5.主页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/ace.min.css}"**/>  
 <**link rel="stylesheet" type="text/css" th:href="@{/css/font-awesome.css}"** />  
  
  
 <**script th:src="@{/js/jquery.min.js}"**></**script**>  
 <**script th:src="@{/js/bootstrap.min.js}"**></**script**>  
</**head**>  
<**body**>  
*<!--<div class="container">-->  
<!--<div align="center" >-->  
<!-- <h2 class="text-center">主窗口</h2>-->  
<!-- <div>-->  
<!-- <a th:href="@{/chart/example1}" class="btn btn-info btn-md" >-->  
<!-- <i class="fa fa-pencil text-dark">chart1</i>-->  
<!-- </a>&nbsp;&nbsp;-->  
<!-- <a th:href="@{/chart/example2}" class="btn btn-info btn-md" >-->  
<!-- <i class="fa fa-pencil text-dark">chart2</i>-->  
<!-- </a>&nbsp;&nbsp;-->  
<!-- <a th:href="@{/chart/example3}" class="btn btn-info btn-md" >-->  
<!-- <i class="fa fa-pencil text-dark">chart3</i>-->  
<!-- </a>-->  
<!-- </div><br/>-->  
<!-- <div>-->  
<!-- <a th:href="@{/user/page}" class="btn btn-warning btn-md" >-->  
<!-- <i class="fa fa-times text-dark" >用户管理</i>-->  
<!-- </a>&nbsp;&nbsp;-->  
<!-- <a th:href="@{/admin/logout}" class="btn btn-warning btn-md" >-->  
<!-- <i class="fa fa-times text-dark" >注销</i>-->  
<!-- </a>-->  
<!-- </div>-->  
<!--</div>-->  
<!--</div>-->*<**nav class="navbar navbar-expand-md bg-dark navbar-dark"**>  
 <**a class="navbar-brand" href="#"**>Navbar</**a**>  
 <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#collapsibleNavbar"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="collapsibleNavbar"**>  
 <**ul class="navbar-nav"**>  
 <**li class="nav-item"**>  
 <**a th:href="@{/user/page}" class="btn btn-warning btn-md"** >  
 <**i class="fa fa-times text-dark"** >用户管理</**i**>  
 </**a**>  
 </**li**>  
 <**li class="nav-item"**>  
 <**a th:href="@{/chart/example1}" class="btn btn-info btn-md"** >  
 <**i class="fa fa-pencil text-dark"**>chart1</**i**>  
 </**a**>  
 </**li**>  
 <**li class="nav-item"**>  
 <**a th:href="@{/chart/example2}" class="btn btn-info btn-md"** >  
 <**i class="fa fa-pencil text-dark"**>chart2</**i**>  
 </**a**>  
 </**li**>  
 <**li class="nav-item"**>  
 <**a th:href="@{/chart/example3}" class="btn btn-info btn-md"** >  
 <**i class="fa fa-pencil text-dark"**>chart3</**i**>  
 </**a**>  
 </**li**>  
 <**li class="nav-item"**>  
 <**a th:href="@{/admin/logout}" class="btn btn-warning btn-md"** >  
 <**i class="fa fa-times text-dark"** >注销</**i**>  
 </**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
</**nav**>  
<**br**>  
  
<**div class="container"**>  
 <**h3**>折叠导航栏</**h3**>  
 <**p**>通常，小屏幕上我们都会折叠导航栏，通过点击来显示导航选项。</**p**>  
 <**p**>提示: 如果你删除 .navbar-expand-md 类，导航链接会一直隐藏，且切换按钮会一直显示。</**p**>  
</**div**>  
  
</**body**>  
</**html**>

主页面运行效果如下：



### 6.路径拦截

如果在每个页面都增加Session中登录对象的判断，根据查验结果选择放行或强制返回登录首页，代码重复，工作量大，而且一旦发生变化，修改量也巨大，这当然不是我们想看到的，我们可以使用AOP（切面编程）或拦截器实现这个要求，一次全部搞定，绝无遗漏。

登录拦截器：

**public class** LoginInterceptor **implements** HandlerInterceptor {  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  
 *//编写拦截的业务规则  
 //从Session中获取用户信息* User lUser= (User)request.getSession().getAttribute(**"loginUser"**);  
 *//判断用户是否登录* **if** (**null**==lUser){  
 *//未登录* response.sendRedirect(request.getContextPath()+**"/"**);  
 **return false**;  
 }  
 **return true**;  
 *//return HandlerInterceptor.super.preHandle(request, response, handler);* }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {  
 HandlerInterceptor.**super**.postHandle(request, response, handler, modelAndView);  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {  
 HandlerInterceptor.**super**.afterCompletion(request, response, handler, ex);  
 }  
}

拦截器配置文件：

@Configuration *//定义此类为配置类，相当于SpringMVC中的配置XML文件***public class** InterceptorConfig **implements** WebMvcConfigurer {  
 *//mvc:interceptors* @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 String[] addPathPatterns={  
 **"/user/\*\*"**,**"/chart/\*\*"** };  
 String[] excludePathPatterns={  
  
 };  
  
 *//mvc:interceptor* registry.addInterceptor(**new** LoginInterceptor())  
 .addPathPatterns(addPathPatterns)  
 .excludePathPatterns(excludePathPatterns);  
 }  
  
}

我们在练习项目中配置了另一套安全机制，当时是为了Swegger的接口发布页面的访问，为了保证路径拦截器的作用，必须修改配置，放行页面的访问。实际上，系统有一套安全机制即可，下面的例子只是为了演示路径拦截的作用。

Spring Security的拦截配置修改如下：

@Configuration  
@EnableWebSecurity  
**public class** ActuatorWebSecurityConfigurationAdapter **extends** WebSecurityConfigurerAdapter {  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http  
 .authorizeRequests()  
 *//普通的接口不需要校验* .antMatchers(**"/admin/\*\*"**).permitAll()  
 .antMatchers(**"/user/\*\*"**).permitAll()  
 .antMatchers(**"/chart/\*\*"**).permitAll()  
 *// swagger页面需要添加登录校验* .antMatchers(**"/swagger-ui/index.html"**).authenticated()  
 .and()  
 .formLogin();  
 }  
}

## 第二十六章 传输JSON数据至前端页面

### 1.传输字符串至浏览器

传输JSON数据至浏览器比较简单，使用@ResponseBody注解即可。

### 2.传输数据集至前端页面

如何传输JSON数据至页面呢？

这似乎与上面是一样的，实际上完全不一样，上面浏览器自己解析，下面需要后台对数据进行处理传值到浏览器的页面上。

采用JSON方式传送集合类型数据至前端需要第三方toJSON方法，目前这类第三方工具类很多，甚至可以在网上找到很多开源的源代码，比较著名的是阿里巴巴的fastJSON,引入该类只需要在pom.xml中增加依赖：

<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.62</**version**>  
</**dependency**>

控制器处理方式：

@RequestMapping({**"/index"**, **"/"**})  
**public** String toIndexPage(Model model) {  
 SideVO sv = **new** SideVO();  
 sv.setParentId(**"0"**); *// 先找父节点为0的数据，也可能有多个* List<SideVO> md = **menuService**.createSideMenu(sv);  
 String mdJSON = JSONArray.*toJSONString*(md, SerializerFeature.***WriteMapNullValue***);  
 model.addAttribute(**"mdJSON"**,mdJSON);  
 **return "index"**;  
}

页面处理方式：

<**script th:inline="javascript"**>  
 **var *menuData***=[[${mdJSON}]];*//取得菜单数据 "$(menuData)"* **$**(**"#menu1"**).sidemenu({  
 **data**:***menuData***,});  
  
</**script**>

在图表系统中，这个方法使用的特别多。

## 第二十七章 Echart图表1

### 引入图标组件Echart

Echart是百度贡献给Apache基金会的开源项目，现在的官方网站为：

https://echarts.apache.org/zh/index.html

开发者可以在该网站学习和自由下载Echart的相关文件。

在Thymeleaf页面只要引入：

<**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>

### 静态图表

（1）example1.html

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>  
</**head**>  
<**body**>  
<**div id="main" style="width**: 600**px**;**height**:400**px**;**"**></**div**>  
<**script type="text/javascript"**>  
 **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**),**'vintage'**);  
 **option** = {  
 **xAxis**: {  
 **type**: **'category'**,  
 **data**: [**'Matcha Latte'**, **'Milk Tea'**, **'Cheese Cocoa'**, **'Walnut Brownie'**]  
 },  
 **yAxis**: {},  
 **series**: [  
 {  
 **type**: **'bar'**,  
 **name**: **'2015'**,  
 **data**: [89.3, 92.1, 94.4, 85.4]  
 },  
 {  
 **type**: **'bar'**,  
 **name**: **'2016'**,  
 **data**: [95.8, 89.4, 91.2, 76.9]  
 },  
 {  
 **type**: **'bar'**,  
 **name**: **'2017'**,  
 **data**: [97.7, 83.1, 92.5, 78.1]  
 }  
 ]  
 };  
 ***myChart***.setOption(**option**);  
</**script**>  
</**body**>  
</**html**>

控制层路径方法：

@GetMapping(**"/chart/example1"**)  
**public** String toExample1(){  
 **return "chart/example1"**;  
}  
  
@GetMapping(**"/chart/example2"**)  
**public** String toExample2(){  
 **return "chart/example2"**;  
}

显示结果如下：



（2）example2.html

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>  
</**head**>  
<**body**>  
<**div id="main" style="width**: 600**px**;**height**:400**px**;**"**></**div**>  
<**script type="text/javascript"**>  
 **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**),**'vintage'**);  
 **option** = {  
 **tooltip**: {  
 **trigger**: **'axis'**,  
 **axisPointer**: { **type**: **'cross'** }  
 },  
 **legend**: {},  
 **xAxis**: [  
 {  
 **type**: **'category'**,  
 **axisTick**: {  
 **alignWithLabel**: **true** },  
 **data**: [  
 **'1月'**,  
 **'2月'**,  
 **'3月'**,  
 **'4月'**,  
 **'5月'**,  
 **'6月'**,  
 **'7月'**,  
 **'8月'**,  
 **'9月'**,  
 **'10月'**,  
 **'11月'**,  
 **'12月'** ]  
 }  
 ],  
 **yAxis**: [  
 {  
 **type**: **'value'**,  
 **name**: **'降水量'**,  
 **min**: 0,  
 **max**: 250,  
 **position**: **'right'**,  
 **axisLabel**: {  
 **formatter**: **'{value} ml'** }  
 },  
 {  
 **type**: **'value'**,  
 **name**: **'温度'**,  
 **min**: 0,  
 **max**: 25,  
 **position**: **'left'**,  
 **axisLabel**: {  
 **formatter**: **'{value} °C'** }  
 }  
 ],  
 **series**: [  
 {  
 **name**: **'降水量'**,  
 **type**: **'bar'**,  
 **yAxisIndex**: 0,  
 **data**: [6, 32, 70, 86, 68.7, 100.7, 125.6, 112.2, 78.7, 48.8, 36.0, 19.3]  
 },  
 {  
 **name**: **'温度'**,  
 **type**: **'line'**,  
 **smooth**: **true**,  
 **yAxisIndex**: 1,  
 **data**: [  
 6.0,  
 10.2,  
 10.3,  
 11.5,  
 10.3,  
 13.2,  
 14.3,  
 16.4,  
 18.0,  
 16.5,  
 12.0,  
 5.2  
 ]  
 }  
 ]  
 };  
 ***myChart***.setOption(**option**);  
</**script**>  
</**body**>  
</**html**>



### 后台传输数据图表

将上面第二个例子改造一下，数据改由后台传送过来。

（1）example3.html

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>  
</**head**>  
<**body**>  
<**div id="main" style="width**: 600**px**;**height**:400**px**;**"**></**div**>  
<**script th:inline="javascript"**>  
 *//var data1 = JSON.parse([[${data1}]]);* **var *data1*** = [[${data1}]];  
 ***console***.log(***data1***);  
 **var *data2*** = [[${data2}]];  
 *//var data2 = JSON.parse([[${data2}]]);* ***console***.log(***data2***);  
 **var *data3*** = [[${data3}]];  
 *//alert(data2);* **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**),**'vintage'**);  
 **option** = {  
 **tooltip**: {  
 **trigger**: **'axis'**,  
 **axisPointer**: { **type**: **'cross'** }  
 },  
 **legend**: {},  
 **xAxis**: [  
 {  
 **type**: **'category'**,  
 **axisTick**: {  
 **alignWithLabel**: **true** },  
 **data**: ***data1*** }  
 ],  
 **yAxis**: [  
 {  
 **type**: **'value'**,  
 **name**: **'降水量'**,  
 **min**: 0,  
 **max**: 250,  
 **position**: **'right'**,  
 **axisLabel**: {  
 **formatter**: **'{value} ml'** }  
 },  
 {  
 **type**: **'value'**,  
 **name**: **'温度'**,  
 **min**: 0,  
 **max**: 25,  
 **position**: **'left'**,  
 **axisLabel**: {  
 **formatter**: **'{value} °C'** }  
 }  
 ],  
 **series**: [  
 {  
 **name**: **'降水量'**,  
 **type**: **'bar'**,  
 **yAxisIndex**: 0,  
 **data**: ***data2*** },  
 {  
 **name**: **'温度'**,  
 **type**: **'line'**,  
 **smooth**: **true**,  
 **yAxisIndex**: 1,  
 **data**: ***data3*** }  
 ]  
 };  
 ***myChart***.setOption(**option**);  
</**script**>  
</**body**>  
</**html**>

注意：包含thymeleaf变量的脚本必须使用**th:inline="javascript"**

<**script th:inline="javascript"**>

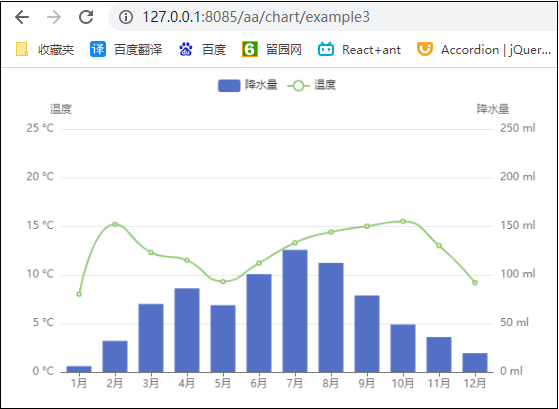
...

<**script**>

(2)控制层路径方法

@GetMapping(**"/chart/example3"**)  
 **public** String toExample3(Model m){  
*// ["1月","2月","3月","4月","5月","6月","7月","8月","9月","10月","11月","12月"]  
// [6, 32, 70, 86, 68.7, 100.7, 125.6, 112.2, 78.7, 48.8, 36.0, 19.3]  
// [6.0,10.2,10.3,11.5,10.3,13.2,14.3,16.4,18.0,16.5,12.0,5.2]* List<String> data1 = **new** ArrayList<>();  
*// data1.add("1M");  
// data1.add("2M");* data1.addAll(Arrays.*asList*(**new** String[]{**"1月"**, **"2月"**, **"3月"**, **"4月"**, **"5月"**, **"6月"**, **"7月"**, **"8月"**, **"9月"**, **"10月"**, **"11月"**, **"12月"**}));  
 List<Float> data2 = **new** ArrayList<Float>();  
*// data2.add(6.0f);  
// data2.add(32.0f);* data2.addAll(Arrays.*asList*(**new** Float[]{6.0f, 32.0f, 70.0f, 86.0f, 68.7f, 100.7f, 125.6f, 112.2f, 78.7f, 48.8f, 36.0f, 19.3f}));  
 List<Float> data3 = **new** ArrayList<Float>();  
 data3.addAll(Arrays.*asList*(**new** Float[]{6.0f,10.2f,10.3f,11.5f,10.3f,13.2f,14.3f,16.4f,18.0f,16.5f,12.0f,5.2f}));  
 m.addAttribute(**"data1"**,JSONArray.*toJSON*(data1));  
 m.addAttribute(**"data2"**,JSONArray.*toJSON*(data2));  
 m.addAttribute(**"data3"**,JSONArray.*toJSON*(data3));  
 **return "chart/example3"**;  
 }

（3）运行访问：



### 从数据库查询数据图表

（1）创建数据库表weather

CREATE TABLE `weather` (

`id` int NOT NULL AUTO\_INCREMENT,

`month` varchar(4) NOT NULL,

`water` decimal(10,2) DEFAULT NULL,

`temperature` decimal(10,2) DEFAULT NULL,

PRIMARY KEY (`id`)

) ;

INSERT INTO `weather` VALUES ('1', '1月', '6.00', '6.00');

INSERT INTO `weather` VALUES ('2', '2月', '32.00', '10.20');

INSERT INTO `weather` VALUES ('3', '3月', '70.00', '10.30');

INSERT INTO `weather` VALUES ('4', '4月', '60.00', '12.00');

INSERT INTO `weather` VALUES ('5', '5月', '55.00', '14.00');

INSERT INTO `weather` VALUES ('6', '6月', '50.00', '16.00');

INSERT INTO `weather` VALUES ('7', '7月', '75.00', '20.00');

INSERT INTO `weather` VALUES ('8', '8月', '85.00', '22.00');

INSERT INTO `weather` VALUES ('9', '9月', '65.00', '15.00');

INSERT INTO `weather` VALUES ('10', '10月', '48.00', '13.00');

INSERT INTO `weather` VALUES ('11', '11月', '36.00', '11.00');

INSERT INTO `weather` VALUES ('12', '12月', '28.00', '8.00');

（2）生成实体类Weather.java

@Table(name = **"weather"**)  
@Entity  
@Getter  
@Setter  
@RequiredArgsConstructor  
@ToString  
**public class** Weather {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**, nullable = **false**)  
 **private** Integer **id**;  
  
 @Column(name = **"month"**, nullable = **false**, length = 4)  
 **private** String **month**;  
  
 @Column(name = **"water"**, precision = 10, scale = 2)  
 **private** BigDecimal **water**;  
  
 @Column(name = **"temperature"**, precision = 10, scale = 2)  
 **private** BigDecimal **temperature**;  
  
}

（3）生成DAO层接口WeatherRepository.java

@Repository  
**public interface** WeatherRepository **extends** JpaRepository<Weather,Integer>, JpaSpecificationExecutor {  
}

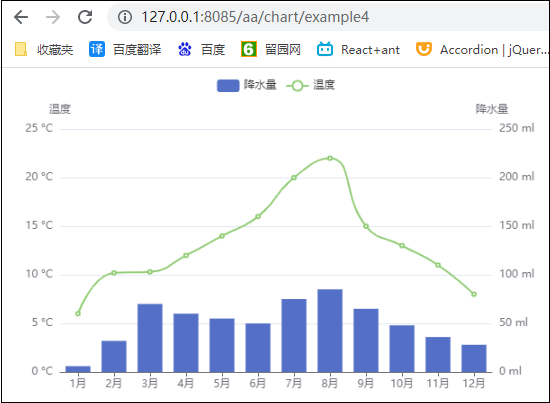
1. 生成控制层路径方法

@GetMapping(**"/chart/example4"**)  
**public** String toExample4(Model m){  
 List<Weather> listWeather = **weatherRepository**.findAll();  
 m.addAttribute(**"data"**,JSONArray.*toJSON*(listWeather));  
 **return "chart/example4"**;  
}

1. 创建页面example4.html

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>  
 <**script th:src="@{/js/jquery.min.js}" type="text/javascript"**></**script**>  
</**head**>  
<**body**>  
<**div id="main" style="width**: 600**px**;**height**:400**px**;**"**></**div**>  
<**script th:inline="javascript"**>  
 *// var data = JSON.parse([[${data}]]);  
 // alert(data);* **var *data*** = [[${data}]]; *//"${}"标签写法\* **var *data1*** = **new *Array***();  
 **var *data2*** = **new *Array***();  
 **var *data3*** = **new *Array***();  
  
 **$**.each(***data***, **function**(index, obj){  
 ***data1***.push(obj.**month**);  
 ***data2***.push(obj.water);  
 ***data3***.push(obj.temperature);  
 } );  
  
  
 ***console***.log(***data***);  
 ***console***.log(***data1***);  
 ***console***.log(***data2***);  
 ***console***.log(***data3***);  
 *//alert(data);  
 // var data2 = [[${data2}]];  
 // //var data2 = JSON.parse([[${data2}]]);  
 // console.log(data2);  
 // var data3 = [[${data3}]];  
 //alert(data2);* **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**),**'vintage'**);  
 **option** = {  
 **tooltip**: {  
 **trigger**: **'axis'**,  
 **axisPointer**: { **type**: **'cross'** }  
 },  
 **legend**: {},  
 **xAxis**: [  
 {  
 **type**: **'category'**,  
 **axisTick**: {  
 **alignWithLabel**: **true** },  
 **data**: ***data1*** }  
 ],  
 **yAxis**: [  
 {  
 **type**: **'value'**,  
 **name**: **'降水量'**,  
 **min**: 0,  
 **max**: 250,  
 **position**: **'right'**,  
 **axisLabel**: {  
 **formatter**: **'{value} ml'** }  
 },  
 {  
 **type**: **'value'**,  
 **name**: **'温度'**,  
 **min**: 0,  
 **max**: 25,  
 **position**: **'left'**,  
 **axisLabel**: {  
 **formatter**: **'{value} °C'** }  
 }  
 ],  
 **series**: [  
 {  
 **name**: **'降水量'**,  
 **type**: **'bar'**,  
 **yAxisIndex**: 0,  
 **data**: ***data2*** },  
 {  
 **name**: **'温度'**,  
 **type**: **'line'**,  
 **smooth**: **true**,  
 **yAxisIndex**: 1,  
 **data**: ***data3*** }  
 ]  
 };  
 ***myChart***.setOption(**option**);  
</**script**>  
</**body**>  
</**html**>

（6）运行结果如下



## 第二十八章 Echart图表2

本章以2019年疫情数据为基础，构造一个比较复杂的图表的开发。

### 实体类

Cov19.java

@Table(name = **"cov19"**)  
@Entity  
@Getter  
@Setter  
@RequiredArgsConstructor  
@ToString  
**public class** Cov19 {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**, nullable = **false**)  
 **private** Integer **id**;  
  
 @Column(name = **"area"**, nullable = **false**, length = 20)  
 **private** String **area**;  
  
 @Column(name = **"country"**, nullable = **false**, length = 50)  
 **private** String **country**;  
  
 @Column(name = **"patients"**, nullable = **false**)  
 **private** Integer **patients**;  
  
 @Column(name = **"death"**, nullable = **false**)  
 **private** Integer **death**;  
  
 @Column(name = **"death\_rate"**, nullable = **false**)  
 **private** Float **deathRate**;  
  
 @Column(name = **"create\_time"**)  
 **private** Instant **createTime**;  
  
 @Column(name = **"create\_by"**, length = 40)  
 **private** String **createBy**;  
  
 @Column(name = **"update\_time"**)  
 **private** Instant **updateTime**;  
  
 @Column(name = **"update\_by"**, length = 40)  
 **private** String **updateBy**;  
  
}

注意：上述代码是JPA Buddy自动生成的。

### 2.DAO层

仅仅是查询数据，那么只需要定义一个简单接口即可。如果是复杂查询，才需要定制开发专用DAO层接口或业务层接口及实现。

Cov19Repository.java

**import** com.fd.springboot.entity.Cov19;  
**import** org.springframework.data.jpa.repository.JpaRepository;  
**import** org.springframework.data.jpa.repository.JpaSpecificationExecutor;  
**public interface** Cov19Repository **extends** JpaRepository<Cov19,Integer>, JpaSpecificationExecutor {  
}

### 3.Controller层

此控制器路径方法包含两个，一个是开始访问时请求数据，一个是Ajax方式请求数据，请注意二者的区别。

@Autowired  
Cov19Repository **cov19Repository**;

//起始访问页面

@GetMapping(**"/chart/cov19"**)  
**public** String toCov19(Model m){  
 List<Cov19> listCov19 = **cov19Repository**.findAll();  
 m.addAttribute(**"data"**,JSONArray.*toJSON*(listCov19));  
 System.***out***.println(JSONArray.*toJSONString*(listCov19));  
 **return "chart/cov19"**;  
}  
  
//动态刷新时Ajax请求页面

@RequestMapping(value = **"/chart/getCov19"**)  
@ResponseBody  
**public** InterfaceResult<Object> Cov19DataAsyn(){  
 InterfaceResult<Object> interfaceResult = **new** InterfaceResult<>();  
 List<Cov19> listCov19 = **cov19Repository**.findAll();  
 interfaceResult.setResult(JSONArray.*toJSON*(listCov19));  
 **return** interfaceResult;  
}

### 4.Ajax提交

使用jquery的Ajax方法可以实现页面的局部刷新，突破表单提交的使用局限性

$.ajax({

url: Path + "/final\_file",

type: "GET",

cache: false,

data: {

serialNo: serialNo,

orgId: orgId

},

dataType: "json", //表示返回值类型，不必须

success: function (data) {

、、、

、、、

、、、

}

});

返回结果data，要求其中包含数据信息还要包含一个成功标志（success）

### 5.Ajax请求专用返回类

因为Ajax请求回调函数的数据格式要求有内容（数据）和成功标志（success），因此必须定义一个此类格式要求的类

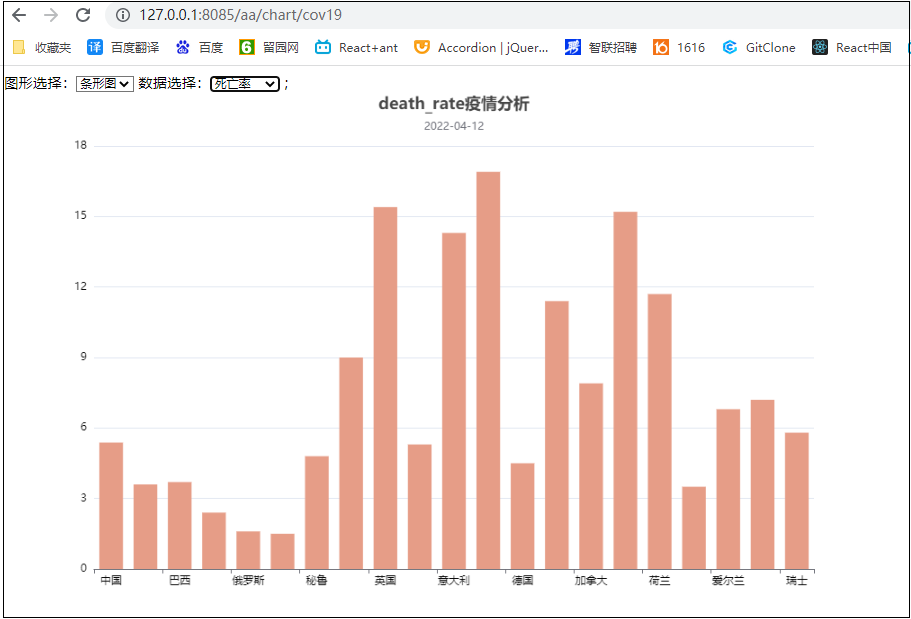
**public class** InterfaceResult<T> **implements** Serializable {  
  
 **private static final long *serialVersionUID*** = -155155513603840660L;  
  
 */\*\*  
 \* 业务结果  
 \*/* **private** T **result**;  
   
 */\*\*  
 \* 业务请求结果  
 \* {0：成功；1：失败；}  
 \*/* **private** String **code** = **"0"**;  
   
 */\*\*  
 \* 业务返回信息  
 \*/* **private** String **msg** = **""**;  
   
 **public** T getResult() {  
 **return result**;  
 }  
  
 **public void** setResult(T result) {  
 **this**.**result** = result;  
 }  
  
 **public** String getCode() {  
 **return code**;  
 }  
  
 **public void** setCode(String code) {  
 **this**.**code** = code;  
 }  
  
 **public** String getMsg() {  
 **return msg**;  
 }  
  
 **public void** setMsg(String msg) {  
 **this**.**msg** = msg;  
 }  
  
 */\*\*  
 \* 业务执行过程发生异常时设置错误信息  
 \** ***@param msg*** *返回给客户端的错误描述  
 \*/* **public void** setError(String msg) {  
 **this**.**code** = **"1"**;  
 **this**.**msg** = msg;  
 }  
 */\*\*  
 \* 业务执行过程发生异常时设置错误信息  
 \** ***@param code*** *返回给客户端的错误代码  
 \** ***@param msg*** *返回给客户端的错误描述  
 \*/* **public void** setError(String code, String msg) {  
 **this**.**code** = code;  
 **this**.**msg** = msg;  
 }   
}

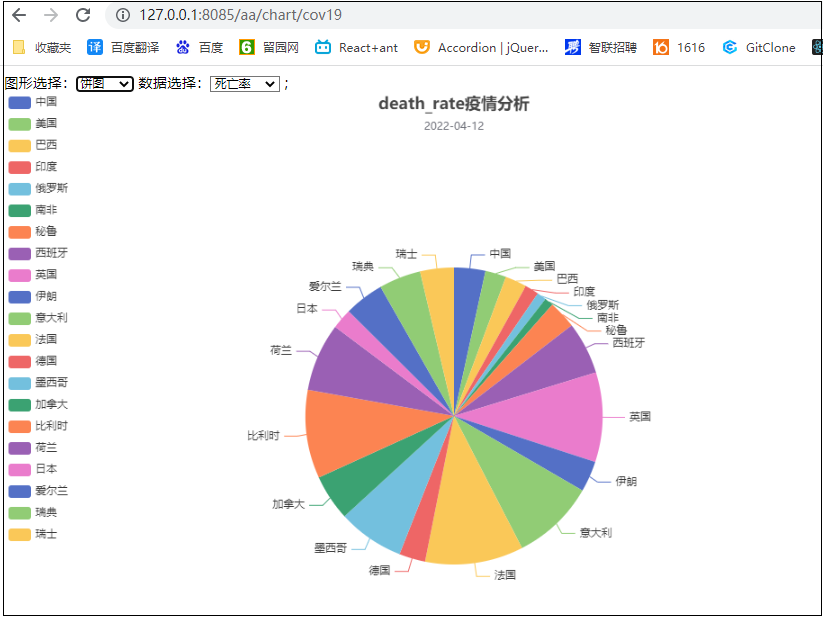
### 6.页面

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Cov2019</**title**>  
 <**base th:href="${#request.getContextPath()}+'/'"**>  
 <**script th:src="@{/js/echarts.min.js}" type="text/javascript"**></**script**>  
 <**script th:src="@{/js/jquery.min.js}" type="text/javascript"**></**script**>  
</**head**>  
<**body**>  
<**div**>  
 <**span**>  
 图形选择：<**select id="icontype" class="col-md-2" onchange="***asynQuery*()**"**>  
 <**option value=""**>请选择</**option**>  
 <**option value="line"**>直线图</**option**>  
 <**option value="bar"**>条形图</**option**>  
 <**option value="pie"**>饼图</**option**>  
 </**select**>  
 </**span**>  
 <**span**>  
 数据选择：<**select id="datatype" class="col-md-2" onchange="***asynQuery*()**"**>  
 <**option value=""**>请选择</**option**>  
 <**option value="patients"**>病人</**option**>  
 <**option value="death"**>死亡人数</**option**>  
 <**option value="death\_rate"**>死亡率</**option**>  
 </**select**>  
 </**span**>  
;  
<**div id="main" style="width**: 100%;**height**:600**px**;**"**></**div**>  
<**script th:inline="javascript"**>  
 *// 初始化开始 ---------------------------------------------* **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**),**'vintage'**);  
 *// var dom = document.getElementById("container");  
 // var myChart = echarts.init(dom);  
 Query*(); *//同步加载  
 asynQuery*(); *//异步加载  
 //初始化结束 ---------------------------------------------  
 //计时器开始----------------------------------------------  
 setInterval*(**function** (){  
 *asynQuery*(); *//异步加载* **var** date=**new *Date***();  
 ***console***.log(**"异步刷新:"**+date.getHours()+**":"**+date.getMinutes()+**":"**+date.getSeconds());  
 },60000);*//计时器自动每隔10秒刷新  
 //计时器结束 ----------------------------------------------  
 //同步查询* **function** *Query*(){  
 **var** ct=**$**(**"#icontype"**).val(); *//icontype* **var** dt=**$**(**"#datatype"**).val(); *//datasort* **var** data = [[${data}]]; *//"${}"标签写法  
 LoadChart*(ct,dt,data); *//加载图形* }  
 *//异步查询* **function** *asynQuery*(){  
 **var** ct=**$**(**"#icontype"**).val(); *//icontype* **var** dt=**$**(**"#datatype"**).val(); *//datasort  
 //var ct = document.getElementById('ct').value;  
 //var dt = document.getElementById('dt').value;* **$**.ajax({  
 **type** : **'GET'**,  
 **url** : [[${#request.getContextPath()}]]+**'/chart/getCov19'**,  
 **data** : {},  
 success: **function**(data){ *//回调函数* **var** dataList = data.**result**;*//$.parseJSON(data.result);  
 //alert(dataList);  
 LoadChart*(ct,dt,dataList); *//加载图形* } *//success* }) *//ajax* };  
  
 *//加载图形* **function** *LoadChart*(ct,dt,data) {  
 **var** titletext = dt+**"疫情分析"**;  
 **var** subtext = *formatDate*(**new *Date***());  
  
 **var** data = [[${data}]]; *//"${}"标签写法* **var** country = **new *Array***();  
 **var** pdata = **new *Array***();  
 **var** ydata = **new *Array***();  
 **$**.each(data, **function**(index, obj){  
 **var** item = **new *Object***();  
 item.**name** = obj.**country**;  
 country.push(obj.**country**);  
 **switch** (dt){  
 **case "patients"**:  
 item.**value** = obj.patients;  
 ydata.push(obj.patients);  
 **break**;  
 **case "death"**:  
 item.**value** = obj.death;  
 ydata.push(obj.death);  
 **break**;  
 **case "death\_rate"**:  
 item.**value** = obj.deathRate;  
 ydata.push(obj.deathRate);  
 **break**;  
 }  
 pdata.push(item);  
 } );  
  
 **var** option1 = {  
 **color**: [  
 **'#da4537'**,  
 **'#e9bb32'**,  
 **'#a9e367'**,  
 **'#37dab9'**,  
 **'#3250e9'**,  
 **'#be67e3'** ],  
 **title** : {  
 **text** : titletext,  
 **subtext** :subtext,  
 **left** : **'center'** },  
 **textStyle** : {  
 **color** : **'Black'** },  
 **backgroundColor** : **'White'**,  
 **tooltip** : {  
 **trigger** : **'axis'**,  
 **axisPointer** : {  
 **type** : **'cross'** }  
 },  
 **xAxis** : {  
 **type** : **'category'**,  
 **data** : country  
 },  
 **yAxis** : {  
 **type** : **'value'** },  
 **series** : [ {  
 **name** : **'值'**,  
 **data** : ydata,  
 **color** : [ **'#e69d87'**, **'#8dc1a9'**, **'#ea7e53'**, **'#eedd78'**,  
 **'#73a373'**, **'#73b9bc'**, **'#7289ab'**, **'#91ca8c'**,  
 **'#f49f42'** ],  
 **type** : ct  
 } ]  
 };  
 **var** option2 = { *//ct="pie"* **title** : {  
 **text** : titletext,  
 **subtext** : subtext,  
 **left** : **'center'** },  
 **tooltip** : {  
 **trigger** : **'item'**,  
 **formatter** : **'{a} <br/>{b} : {c} ({d}%)'** },  
 **legend** : {  
 **orient** : **'vertical'**,  
 **left** : **'left'**,  
 **data** : country  
 },  
 **series** : [ {  
 **name** : **'数据来源'**,  
 **type** : ct,  
 **radius** : **'55%'**,  
 **center** : [ **'50%'**, **'60%'** ],  
 **data** :pdata,  
 **emphasis** : {  
 **itemStyle** : {  
 **shadowBlur** : 10,  
 **shadowOffsetX** : 0,  
 **shadowColor** : **'rgba(0, 0, 0, 0.5)'** }  
 }  
 } ]  
 };  
 **if** (ct != **"pie"**) {  
 *//line和bar* ***myChart***.setOption(option1, **true**);  
 } **else** {  
 *//pie* ***myChart***.setOption(option2, **true**);  
 }  
 }  
  
 *//日期格式化方法* **function** *formatDate*(date) {  
 **var** myyear = date.getFullYear();  
 **var** mymonth = date.getMonth() + 1;  
 **var** myweekday = date.getDate();  
  
 **if** (mymonth < 10) {  
 mymonth = **"0"** + mymonth;  
 }  
 **if** (myweekday < 10) {  
 myweekday = **"0"** + myweekday;  
 }  
 **return** (myyear + **"-"** + mymonth + **"-"** + myweekday);  
 }  
</**script**>  
</**body**>  
</**html**>

### 7.运行并访问

http://127.0.0.1:8085/aa/chart/cov19





### 8.在IDEA开发中的注意事项

不要直接在IDEA编辑环境中的项目模块之间拷贝复用代码，尽量过渡以下，先考到系统目录下，再拷贝到其他引用项目中，否则会造成模块之间的引用，造成模块代码的关系混乱，甚至会造成莫名其妙的问题，无法编译。

## 第二十九章 集成Mybatis-准备工作

Mybatis介绍

MyBatis 本是apache的一个**[开源项目](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90%E9%A1%B9%E7%9B%AE/3406069)**iBatis, 2010年这个**[项目](https://baike.baidu.com/item/%E9%A1%B9%E7%9B%AE/477803)**由apache software foundation 迁移到了**[google code](https://baike.baidu.com/item/google%20code/2346604)**，并且改名为MyBatis 。2013年11月迁移到**[Github](https://baike.baidu.com/item/Github/10145341)**。

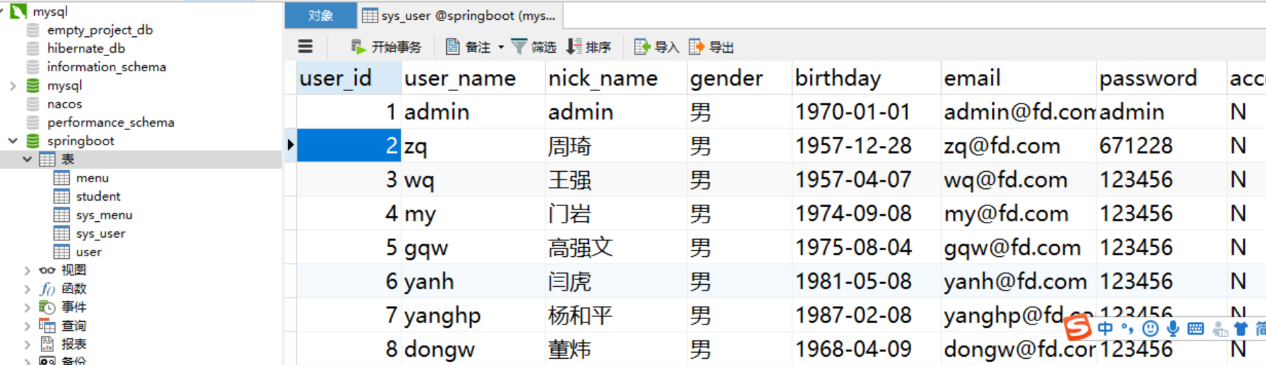
iBATIS一词来源于“internet”和“abatis”的组合，是一个基于Java的**[持久层](https://baike.baidu.com/item/%E6%8C%81%E4%B9%85%E5%B1%82/3584971)**框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAOs）

当前，最新版本是MyBatis 3.5.7 ，其发布时间是2021年4月21日。

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Ordinary Java Object,普通的 Java对象)映射成数据库中的记录。

### 1.数据准备

打开Navicat软件，双击Mysql链接名（自己配置的），双击springboot数据库，导入E:\dev\安装包\sys\_user.sql和sys\_menu两个文件，查看数据是否导入。



### 2.集成Mybatis依赖

 <dependency>  
    <groupId>org.mybatis.spring.boot</groupId>  
    <artifactId>mybatis-spring-boot-starter</artifactId>  
    <version>2.1.4</version>  
 </dependency>

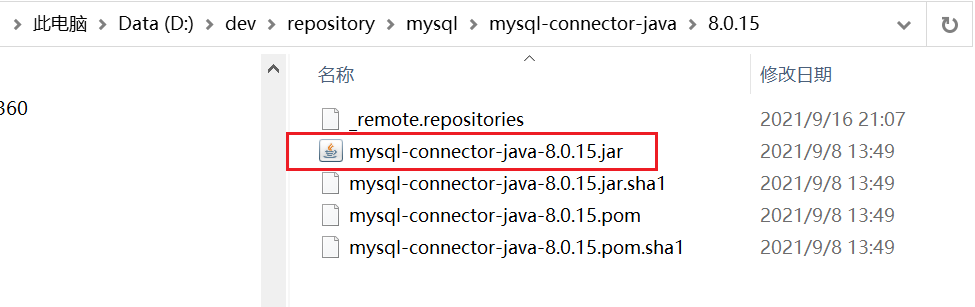
### 3.集成MySQL依赖

 <dependency>  
    <groupId>mysql</groupId>  
    <artifactId>mysql-connector-java</artifactId>  
    <version>8.0.15</version>  
 </dependency>

MySQL的版本必须是5.7.X以上版本，低版本的不能用8.0.15这个版本依赖

这个依赖会下载项目所用的JDBC驱动，会放到哪呢？

会放到我们的本地仓库中E:\dev\repository



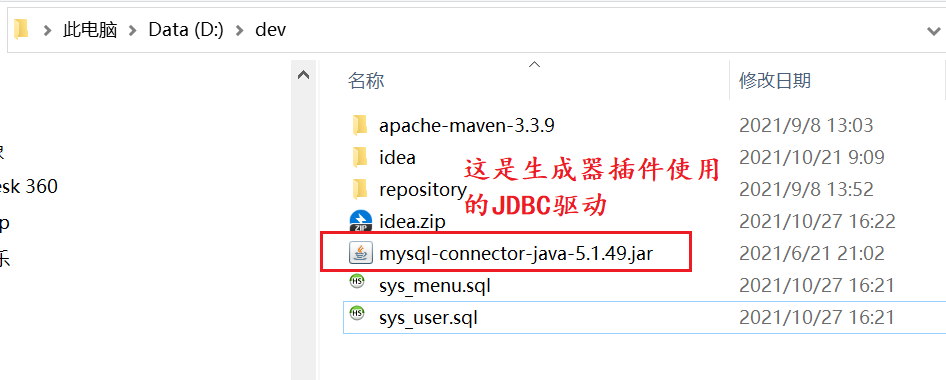
注：老师的笔记本没有E盘，所以是D盘

### 4.集成MyBatis代码生成器插件

 <!--mybatis代码自动生成插件 -->  
 <plugin>  
    <groupId>org.mybatis.generator</groupId>  
    <artifactId>mybatis-generator-maven-plugin</artifactId>  
    <version>1.4.0</version>  
    <configuration>  
        <configurationFile>GeneratorMapper.xml</configurationFile>  
        <verbose>true</verbose>  
        <overwrite>true</overwrite>  
    </configuration>  
 </plugin>

### 5.拷贝MySQL的驱动至E:\dev下

生成器插件本身也是一个java数据库项目，它的版本较低，使用的是MySQL5.1.X的版本，因此jdbc驱动的版本必须是低版本



注：老师的笔记本没有E盘，所以是D盘

### 6.在项目配置文件中增加配置信息

**spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/springboot?useUnicode=true&characterEncoding=utf-8&useSSL=true&serverTimezone=Asia/Shanghai  
spring.datasource.username=root  
spring.datasource.password=root**

注意：低版本的驱动名字和高版本不一样，低版本的名字是：

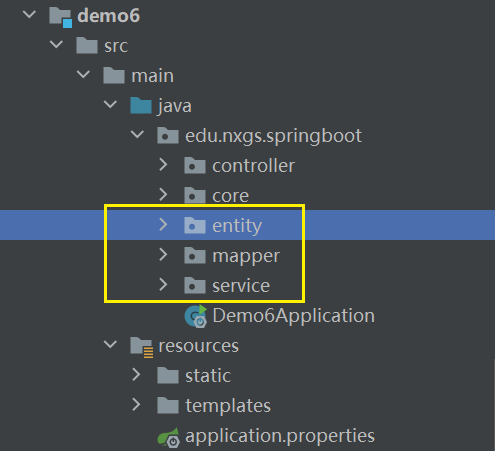
**com.mysql.jdbc.Driver**

### 7.拷贝GeneratorMapper.xml至项目/模块目录下

 <?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE generatorConfiguration  
 PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
 "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  
<generatorConfiguration>  
 <!--指向连接数据库的 JDBC 驱动包所在位置，指定到你本机的完整路径-->  
 <classPathEntry location="**D:\dev\mysql-connector-java-5.1.49.jar**"/>  
 <!--配置table表信息内容体，targetRuntime 指定采用MyBatis3的版本-->  
 <context id="tables" targetRuntime="MyBatis3">  
 <commentGenerator>  
 <property name="suppressAllComments" value="true"/>  
 </commentGenerator>  
 <!--配置数据库连接信息-->  
 <jdbcConnection driverClass="**com.mysql.jdbc.Driver**"  
 connectionURL="jdbc:mysql://localhost:3306/**springboot**"  
 userId="**root**"  
 password="**root**">  
 </jdbcConnection>  
 <javaModelGenerator targetPackage="**edu.nxgs.springboot.entity**"  
 targetProject="src/main/java">  
 <property name="enableSubPackages" value="false"/>  
 <property name="trimStrings" value="false"/>  
 </javaModelGenerator>  
 <sqlMapGenerator targetPackage="**edu.nxgs.springboot.mapper**"  
 targetProject="src/main/java">  
 <property name="enableSubPackages" value="false"/>  
 </sqlMapGenerator>  
 <!--生成Mybatis的Mapper接口类文件，targetPackage指定Mapper接口类的名-->  
 <javaClientGenerator type="XMLMAPPER"  
 targetPackage="**edu.nxgs.springboot.mapper**"  
 targetProject="src/main/java">  
 <property name="enableSubPackages" value="false"/>  
 </javaClientGenerator>  
  
 <!--数据库表名及对应的Java模型类名-->  
 <table tableName="**sys\_user**" domainObjectName="**User**"  
 enableCountByExample="false"  
 enableUpdateByExample="false"  
 enableDeleteByExample="false"  
 enableSelectByExample="false"  
 selectByExampleQueryId="false"/>  
 </context>  
</generatorConfiguration>

这个文件中我们要改很多内容，下面的节将逐步说明。

### 8.新建entity、mapper、service三个包



### 9.修改GeneratorMapper.xml文件

（1）GeneratorMapper.xml中的包名

fd.springboot.mybatis.entity改为：edu.nxgs.springboot.entity

fd.springboot.mybatis.mapper改为：edu.nxgs.springboot.mapper

（2）GeneratorMapper.xml中增加数据库表名及对应的Java模型类名

数据库表名:sys\_user 🡸🡺 user

（3）jdbc连接驱动名：**com.mysql.jdbc.Driver**

（4）数据库名：**springboot**

（5）数据库用户/口令：**root/root**

**注意：学校的计算机上的MySQL的用户口令是：root/123456**

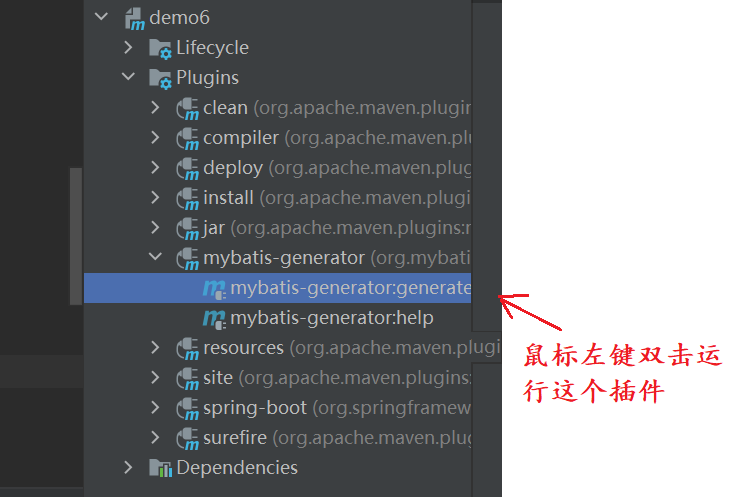
（6）生成器所需要的的JDBC驱动所在的硬盘目录地址：

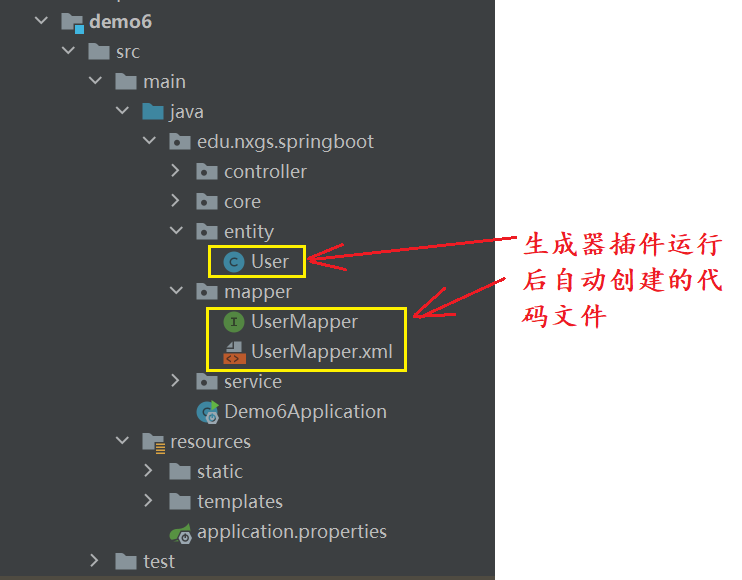
**D:\dev\mysql-connector-java-5.1.49.jar**

### 10.在pom.xml中指定资源文件位置

        <!--手动指定资源目录位置-->  
        <resources>  
            <resource>  
                <directory>src/main/java</directory>  
                <includes>  
                    <include>\*\*/\*.xml</include>  
                </includes>  
            </resource>  
            <resource>  
                <directory>src/main/resources</directory>  
                <includes>  
                    <include>\*\*/\*.\*</include>  
                </includes>  
            </resource>  
        </resources>

### 11.运行mybatis-generator插件





注意：如果要重新生成代码文件，必须把上一次自动生成的代码文件手工删除，才能再次生成，否则会出错。

### 12.在生成的UserMapper.java中增加@mapper注解

package edu.nxgs.springboot.mapper;  
  
import edu.nxgs.springboot.entity.User;  
import org.apache.ibatis.annotations.Mapper;  
  
**@Mapper**  
public interface UserMapper {  
 int deleteByPrimaryKey(Integer userId);  
  
 int insert(User record);  
  
 int insertSelective(User record);  
  
 User selectByPrimaryKey(Integer userId);  
  
 int updateByPrimaryKeySelective(User record);  
  
 int updateByPrimaryKey(User record);  
}

在项目的主程序文件上增加mapper的扫描

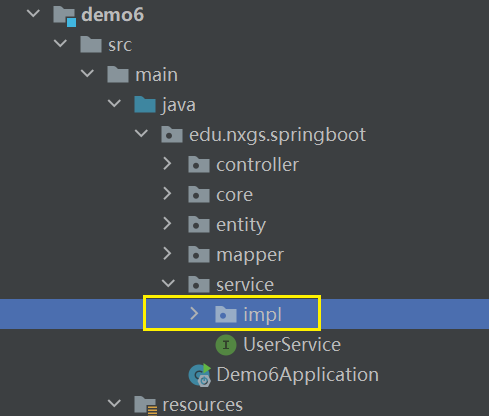
package edu.nxgs.springboot;  
  
import org.mybatis.spring.annotation.MapperScan;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
**@MapperScan("edu.nxgs.springboot.mapper")**  
public class Demo6Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Demo6Application.class, args);  
 }  
  
}

### 13.增加Service层

新建接口类：UserService.java

package edu.nxgs.springboot.service;  
  
import edu.nxgs.springboot.entity.User;  
  
**public interface UserService {  
 public User queryUserById(Integer id);  
}**

在service包下创建子包impl



新建实现类：UserServiceImpl.java

import edu.nxgs.springboot.entity.User;  
import edu.nxgs.springboot.mapper.UserMapper;  
import edu.nxgs.springboot.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
**@Service //此注解的作用是将服务层自动创建为容器Bean  
public class UserServieImpl implements UserService {  
 @Autowired  
 private UserMapper userMapper;  
  
 @Override  
 public User queryUserById(Integer id) {  
 return userMapper.selectByPrimaryKey(id);  
 }  
}**

### 14.Mybatis单元测试

 package edu.nxgs.springboot;  
 ​  
 import edu.nxgs.springboot.entity.User;  
 import edu.nxgs.springboot.mapper.UserMapper;  
 import org.junit.jupiter.api.\*;  
 import org.springframework.beans.factory.annotation.Autowired;  
 import org.springframework.boot.test.context.SpringBootTest;  
 ​  
**​** @TestInstance(TestInstance.Lifecycle.PER\_METHOD)  
 @SpringBootTest  
 class Demo6ApplicationTests { **@Autowired  
    private UserMapper userMapper;**

**@Autowired**

**private UserService userService;**

**@Test  
    void contextLoads() {  
    }  
 ​  
    @Test  
    void testUser1(){    
        User user = userMapper.selectByPrimaryKey(1);  
        System.out.println("Mapper层接口---用户姓名："**

**+user.getUserName());  
    }**

**@Test  
void test2(){  
 User user = userService.queryUserById(3);  
 System.out.println("Service层接口---测试用户姓名:"**

**+user.getUserName());**

**}**

**}**

### 15.控制器方法测试

 package edu.nxgs.springboot.controller;  
 ​  
 import edu.nxgs.springboot.entity.User;  
 import edu.nxgs.springboot.mapper.UserMapper;  
 import org.springframework.beans.factory.annotation.Autowired;  
 import org.springframework.stereotype.Controller;  
 import org.springframework.web.bind.annotation.GetMapping;  
 import org.springframework.web.bind.annotation.ResponseBody;  
 ​  
**@Controller  
 public class UserController {  
    @Autowired  
    private UserMapper userMapper;  
 ​  
    @ResponseBody  
    @GetMapping(value="/user/queryUser")  
    public String getUser(){  
        User user = userMapper.selectByPrimaryKey(2);  
        return "用户："+user.getUserName()+",姓名："+user.getNickName();  
    }  
 }**

## 第三十章 集成Mybatis-改进提高

### 1.修改实体类

User.java实体类

**public class** User {  
 **private** Integer **userId**;  
  
 **private** String **userName=""**;  
  
 **private** String **nickName**;  
  
 **private** String **gender="男"**;  
  
 **@DateTimeFormat(pattern="yyyy-MM-dd")**  
 **private** Date **birthday**=**new** Date();  
  
 **private** String **email**;  
  
 **private** String **password**;  
  
 **private** String **accountLocked="N"**;  
  
 **@DateTimeFormat(pattern="yyyy-MM-dd HH:mm:ss")** **private** Date **createTime=new Date()**;  
  
 **@DateTimeFormat(pattern="yyyy-MM-dd HH:mm:ss")** **private** Date **updateTime=new Date();**  
  
 **private** String **createBy="admin"**;  
  
 **private** String **updateBy="admin"**;  
  
 **public** Integer getUserId() {  
 **return userId**;  
 }  
  
 **public void** setUserId(Integer userId) {  
 **this**.**userId** = userId;  
 }  
  
 **public** String getUserName() {  
 **return userName**;  
 }  
  
 **public void** setUserName(String userName) {  
 **this**.**userName** = userName;  
 }  
  
 **public** String getNickName() {  
 **return nickName**;  
 }  
  
 **public void** setNickName(String nickName) {  
 **this**.**nickName** = nickName;  
 }  
  
 **public** String getGender() {  
 **return gender**;  
 }  
  
 **public void** setGender(String gender) {  
 **this**.**gender** = gender;  
 }  
  
 **public** Date getBirthday() {  
 **return birthday**;  
 }  
  
 **public void** setBirthday(Date birthday) {  
 **this**.**birthday** = birthday;  
 }  
  
 **public** String getEmail() {  
 **return email**;  
 }  
  
 **public void** setEmail(String email) {  
 **this**.**email** = email;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
  
 **public** String getAccountLocked() {  
 **return accountLocked**;  
 }  
  
 **public void** setAccountLocked(String accountLocked) {  
 **this**.**accountLocked** = accountLocked;  
 }  
  
 **public** Date getCreateTime() {  
 **return createTime**;  
 }  
  
 **public void** setCreateTime(Date createTime) {  
 **this**.**createTime** = createTime;  
 }  
  
 **public** Date getUpdateTime() {  
 **return updateTime**;  
 }  
  
 **public void** setUpdateTime(Date updateTime) {  
 **this**.**updateTime** = updateTime;  
 }  
  
 **public** String getCreateBy() {  
 **return createBy**;  
 }  
  
 **public void** setCreateBy(String createBy) {  
 **this**.**createBy** = createBy;  
 }  
  
 **public** String getUpdateBy() {  
 **return updateBy**;  
 }  
  
 **public void** setUpdateBy(String updateBy) {  
 **this**.**updateBy** = updateBy;  
 }  
  
 **@Override  
 public String toString() {** //调试程序时需要 **return "User{" +  
 "userId=" + userId +  
 ", userName='" + userName + '\'' +  
 ", nickName='" + nickName + '\'' +  
 ", gender='" + gender + '\'' +  
 ", birthday=" + birthday +  
 ", email='" + email + '\'' +  
 ", password='" + password + '\'' +  
 ", accountLocked='" + accountLocked + '\'' +  
 ", createTime=" + createTime +  
 ", updateTime=" + updateTime +  
 ", createBy='" + createBy + '\'' +  
 ", updateBy='" + updateBy + '\'' +  
 '}';  
 }**  
  
 **public User() {** //控制器方法中@ModelAttribute和new User()调用需要 **super();  
 }**  
}

MybatisGenerator插件自动生成的实体类代码并不能解决我们开发中所有的实体类代码的问题，还是需要我们补充代码，上述代码中红色部分就是我们实际开发中需要添加的代码，包括：

1. 缺省值，这个缺省值要与数据库标的定义相配合，即数据库表的字段不能为Null的，就需要给初始值，这个设置也可在数据库端完成。
2. 类型格式规范，采用注解方式。例如日期和时间格式。要注意的是彻底解决日期、时间格式规范，涉及到三个部分：

a.ORM关系映射（实体类定义时加日期、时间格式注解）；

@DateTimeFormat(pattern=**"yyyy-MM-dd"**)

@DateTimeFormat(pattern=**"yyyy-MM-dd HH:mm:ss"**)

b.Java部分；例如给日期型对象属性赋值

user.setBirthday(StringUtils.*getDate*(**"1967-12-28"**));

StringUtils.*getDate参考如下：*

**public class** StringUtils {  
 **public static** Date getDate(String date){  
 DateFormat format1 = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 **try** {  
 **return** format1.parse(date);  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
}

c.前端显示部分：

<**td class="left"**><**span th:text="${#calendars.format(u.birthday,'yyyy-MM-dd')}"**></**span**></**td**>

### 2.自定义Mapper层方法

**（1）自定义Mapper层接口**

@Mapper  
**public interface** UserMapper {  
 **int** deleteByPrimaryKey(Integer userId);  
  
 **int** insert(User record);  
  
 **int** insertSelective(User record);  
  
 User selectByPrimaryKey(Integer userId);  
  
 **int** updateByPrimaryKeySelective(User record);  
  
 **int** updateByPrimaryKey(User record);  
  
 ***//查出所有用户记录* List<User> getUsers();  
  
 *//检查用户名和口令* List<User> loginUserCheck(String uStr,String pStr);**  
}

**（2）自定义Mapper层接口实现**

**要想实现Mapper层自定义，必须要看懂Mapper结构，还要熟悉SQL语言的构造。**

1. **resultMap**
2. **Base\_Column\_List**
3. **parameterType="java.lang.String"和jdbcType=VARCHAR**

**获得所有用户数据**

**<select id="getUsers" resultMap="BaseResultMap">**

**select**

**<include refid="Base\_Column\_List" />**

**from sys\_user**

**</select>**

**检查用户口令**

**<select id="loginUserCheck" parameterType="java.lang.String" resultMap="BaseResultMap">**

**select**

**<include refid="Base\_Column\_List" />**

**from sys\_user**

**where user\_name =#{uStr,jdbcType=VARCHAR}**

**and password = #{pStr,jdbcType=VARCHAR}**

**</select>**

### 3.Service服务层方法

UserService.java接口

**public interface** UserService {  
 **public** User queryUserById(Integer Id);  
 *//查询所有用户* **public** List<User> queryUsers();  
 *//用户登录检查* **public** List<User> loginUserQuery(String uStr,String pStr);  
 *//保存用户数据* **public void** saveUser(User user);  
 *//删除用户数据* **public void** delUser(Integer Id);  
 *//新增用户* **public void** insUser(User user);  
}

UserServiceImpl.java接口实现

@Service  
**public class** UserServiceImpl **implements** UserService {  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 @Override  
 **public** User queryUserById(Integer Id) {  
  
 **return userMapper**.selectByPrimaryKey(Id);  
 }  
  
 @Override  
 **public** List<User> queryUsers() {  
 **return userMapper**.getUsers();  
 }  
  
 @Override  
 **public** List<User> loginUserQuery(String uStr, String pStr) {  
 **return userMapper**.loginUserCheck(uStr,pStr);  
 }  
  
 @Override  
 **public void** saveUser(User user) {  
 **userMapper**.updateByPrimaryKeySelective(user);  
 }  
  
 @Override  
 **public void** delUser(Integer Id) {  
 **userMapper**.deleteByPrimaryKey(Id);  
 }  
  
 @Override  
 **public void** insUser(User user) {  
 **userMapper**.insertSelective(user);  
 }  
  
}

### 4.Controller控制器方法

UserController.java

@Controller  
**public class** UserController {  
 @Autowired  
 **private** UserService **userService**;  
  
 @RequestMapping(value=**"user/login"**,method = RequestMethod.***GET***)  
 **public** String requestLogin(@ModelAttribute(value = **"user"**) User user){  
  
 **return "/test/login"** ;  
 }  
  
 @RequestMapping(value=**"user/login"**,method = RequestMethod.***POST***)  
 **public** String responseLogin(@ModelAttribute(value = **"user"**) User user, Model model ){  
 String message = **""**;  
 String us=user.getUserName();  
 String ps=user.getPassword();  
 *//System.out.println("Name:"+us);  
 //System.out.println("password:"+ps);  
  
 //检查用户口令是否正确* List<User> listUsers = **userService**.loginUserQuery(us,ps);  
  
 **if** (listUsers.size() > 0){  
 List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/mainpage"**;  
 }**else**{  
 model.addAttribute(**"message"**,**"用户口令不正确！"**);  
 **return "/test/login"**;  
 }  
 }  
  
 @RequestMapping(value = **"user/userInsert"**, method = RequestMethod.***GET***) *//user/userAdd //user/userDel* **public** String insertNullUser(Model model) {  
 User user = **new** User();  
 model.addAttribute(**"user"**,user);  
 **return "/test/userinsert"**;  
 }  
  
 @RequestMapping(value = **"user/userInsert"**, method = RequestMethod.***POST***) *//user/userAdd //user/userDel* **public** String insertSaveUser(@ModelAttribute(value = **"user"**) User user,Model model) {  
 *//保存用户数据  
  
 //System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 //System.out.println("用户信息:"+user.toString());* **userService**.insUser(user);  
  
 *//查询所有用户数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"**;  
 }  
  
 @RequestMapping(value = **"user/userEdit/{userId}"**, method = RequestMethod.***GET***) *//user/userAdd //user/userDel* **public** String editUser(@PathVariable(**"userId"**) Integer userId,Model model) {  
 *//前台要传一个user\_Id值进来,路径参数  
 //根据user\_Id去数据库查找完整的user* User user = **userService**.queryUserById(userId);  
 *//System.out.println("用户信息："+user.toString());* model.addAttribute(**"user"**, user);  
 **return "/test/useredit"**;  
 }  
  
 @RequestMapping(value = **"user/userEdit"**, method = RequestMethod.***POST***) *//user/userAdd //user/userDel* **public** String saveUser(@ModelAttribute(value = **"user"**) User user, Model model) {  
 *//保存用户数据* **userService**.saveUser(user);  
  
 *//查询所有用户数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"**;  
 }  
  
 @RequestMapping(value = **"user/userDel/{userId}"**, method = RequestMethod.***GET***)  
 **public** String delForm(@PathVariable(**"userId"**) Integer userId, Model model) {  
 *//删除用户数据* **userService**.delUser(userId);  
 *//返回主页面，刷新数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"**;  
 }  
  
 @RequestMapping(value = **"user/userBatchDel"**, method = RequestMethod.***POST***)  
 **public** String delBatchForm(HttpServletRequest request, Model model) {  
 String[] sels = request.getParameterValues(**"recsel"**);  
 **if** (sels != **null**) {  
 *//批量删除用户数据* **for**(String s:sels){  
 Integer uid = Integer.*parseInt*(s);  
 *//System.out.println("uid:"+uid);* **userService**.delUser(uid);  
 }  
 }**else**{  
 System.***out***.println(**"没有选择数据"**);  
 }  
  
 *//返回主页面，刷新数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"**;  
 }  
  
 @RequestMapping(value=**"user/userList"**,method = RequestMethod.***GET***)  
 **public** String userAdmin(Model model){  
 *//返回主页面，刷新数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"** ;  
 }  
  
  
 @RequestMapping(value=**"menu/menuList"**,method = RequestMethod.***GET***)  
 **public** String menuAdmin(Model model){  
 *//返回主页面，刷新数据* List<User> listUser = **userService**.queryUsers();  
 model.addAttribute(**"listUser"**,listUser);  
 **return "/test/userlist"** ;  
 }  
  
 @RequestMapping(value=**"main"**,method = RequestMethod.***GET***)  
 **public** String mainEnter(){  
 **return "/test/mainpage"**;  
 }  
}

### 5.修改项目的配置文件

*#关闭thymeleaf缓存，默认为true***spring.thymeleaf.cache**=**false***#thymeleaf\*\*模板前缀，此时为默认值，****去掉*/templates/后面的斜杠****spring.thymeleaf.prefix**=**classpath:/templates***#thymeleaf模板后缀，此时为默认值***spring.thymeleaf.suffix**=**.html  
  
server.port**=**8095  
server.servlet.context-path**=**/aa  
  
company.name**=**信息工程学院  
company.address**=**银川市西夏区宁夏工商学院***#如果是MySQL5.7以下版本com.mysql.jdbc.Driver***spring.datasource.driver-class-name**=**com.mysql.cj.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://localhost:3306/springboot?useUnicode=true&characterEncoding=utf-8&useSSL=true&serverTimezone=Asia/Shanghai  
spring.datasource.username**=**root  
spring.datasource.password**=**root**

去掉/templates/后面的斜杠是编译打包发布时的要求，在IDE中运行没问题，但程序打包发布后发现，访问不了页面，出现：

Error resolving template [/test/userList], template might not exist or might not be accessible by any of the configured Template Resolvers

经查证发现是**spring.thymeleaf.prefix**=**classpath:/templates/**后面的斜杠引起的，这是引起的错误原因之一。

### 6.修改控制器方法和前端页面路径请求

修改控制器方法和前端页面路径请求将路径前的”/”去除，这是上面的错误原因之二。

### 7.打成jar包部署

（1）修改pom.xml中打包部分

<**build**>  
 <**finalName**>infoadmin1</**finalName**>  
 *<!--手动指定资源目录位置-->* <**resources**>  
 <**resource**>  
 <**directory**>src/main/java</**directory**>  
 <**includes**>  
 <**include**>\*\*/\*.xml</**include**>  
 </**includes**>  
 </**resource**>  
 <**resource**>  
 <**directory**>src/main/resources</**directory**>  
 <**includes**>  
 <**include**>\*\*/\*.\*</**include**>  
 </**includes**>  
 </**resource**>  
 </**resources**>  
  
 <**plugins**>  
 *<!--mybatis代码自动生成插件 -->* <**plugin**>  
 <**groupId**>org.mybatis.generator</**groupId**>  
 <**artifactId**>mybatis-generator-maven-plugin</**artifactId**>  
 <**version**>1.4.0</**version**>  
 <**configuration**>  
 <**configurationFile**>GeneratorMapper.xml</**configurationFile**>  
 <**verbose**>true</**verbose**>  
 <**overwrite**>true</**overwrite**>  
 </**configuration**>  
 </**plugin**>  
  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 <**version**>2.5.4</**version**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

说明：

<**finalName**>infoadmin1</**finalName**>定义打包后的文件名称

<**resources**>定义打包哪些东西到打包文件中，通常是:

\*.java文件编译后的\*.class文件

\*.jar第三方依赖

\*.xml,\*.properties,\*.yml,\*.yaml

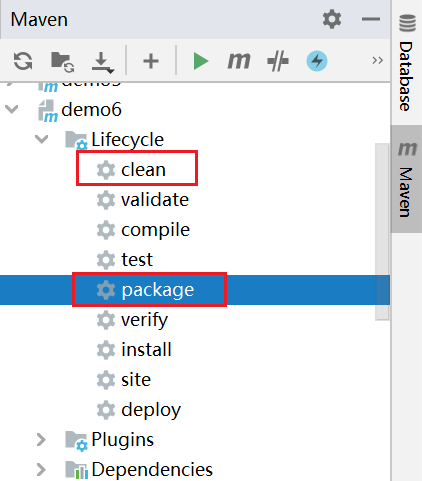
所有的静态资源（图片，文本，音乐文件，视频文件，js,css,font等；所有的页面模板）

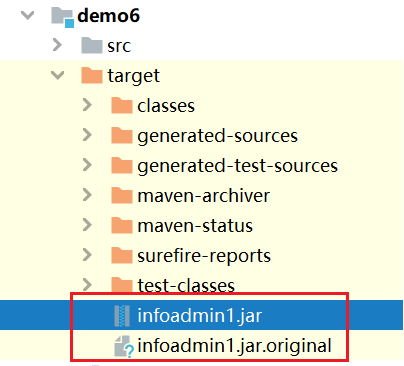
<**plugins**>插件

（2）运行打包

a.在窗体右边“Maven”🡪“Lifecycle”,双击运行“clean”,清除所有旧的编译结果。直接删除“target”目录也可。

b.双击运行“package”

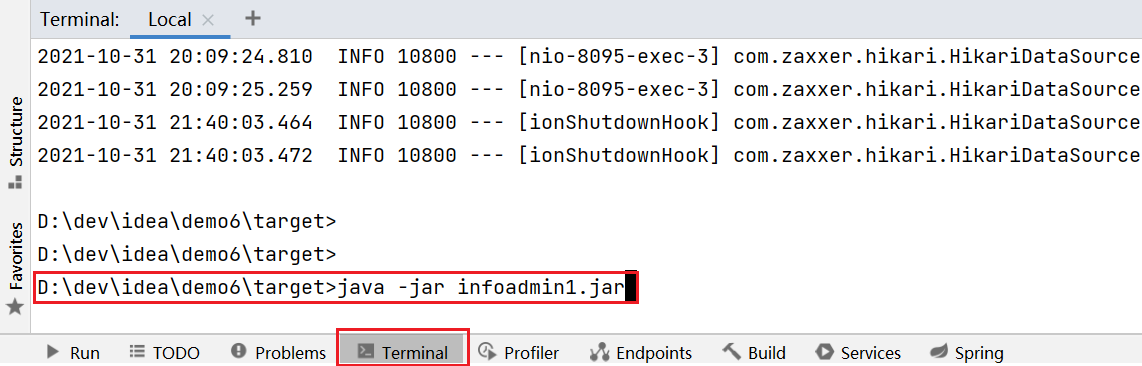




说明：

infoadmin1.jar完整打包

infoadmin1.jar.original不完整打包，如果其他人要调用你的项目，使用这个包，此包不包含依赖（依赖使用的jar包）



（3）运行后即可正常访问，访问方式与IDE编辑器编译调试时相同。

### 8.打成war包部署

（1）修改pom.xml中打包部分

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.5.4</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
 **<packaging>war</packaging>**  
 <**groupId**>edu.nxgs.springboot</**groupId**>

……

说明：

在pom.xml文件中增加<packaging>war</packaging>

（2）修改应用主程序入口类

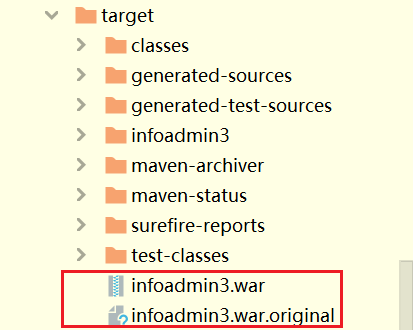
@SpringBootApplication  
@MapperScan(**"edu.nxgs.springboot.mapper"**) *//扫描mapper包下所有带@Mapper注解的类，将它做成Bean***public class** Demo8Application **extends SpringBootServletInitializer**{  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(Demo8Application.**class**, args);  
 }  
 **@Override  
 protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {  
 return builder.sources(Demo8Application.class);  
 }**  
  
}

说明：

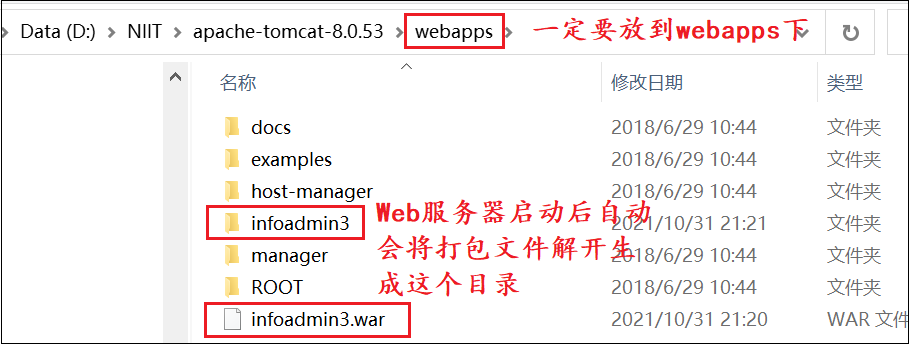
发布成war包需要主程序类继承SpringBootServletInitializer接口类

(3)运行打包

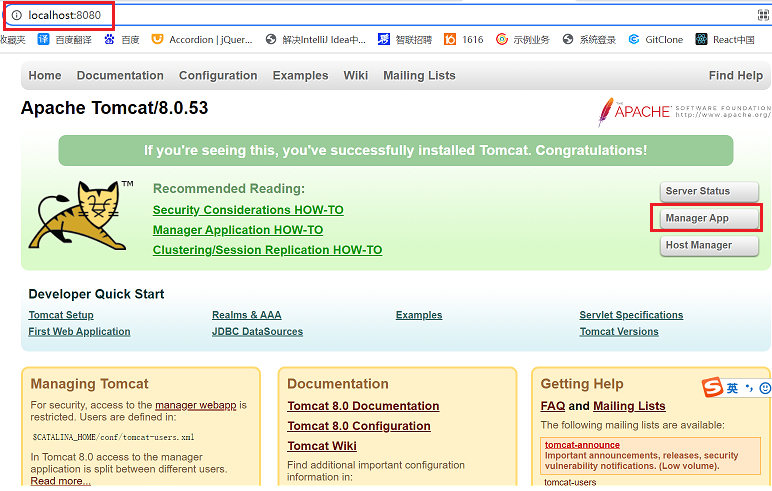
打包过程与打jar包相似



(4)将编译后的打包文件拷贝至Web服务器下



(5)查看Web服务器应用部署情况





说明：

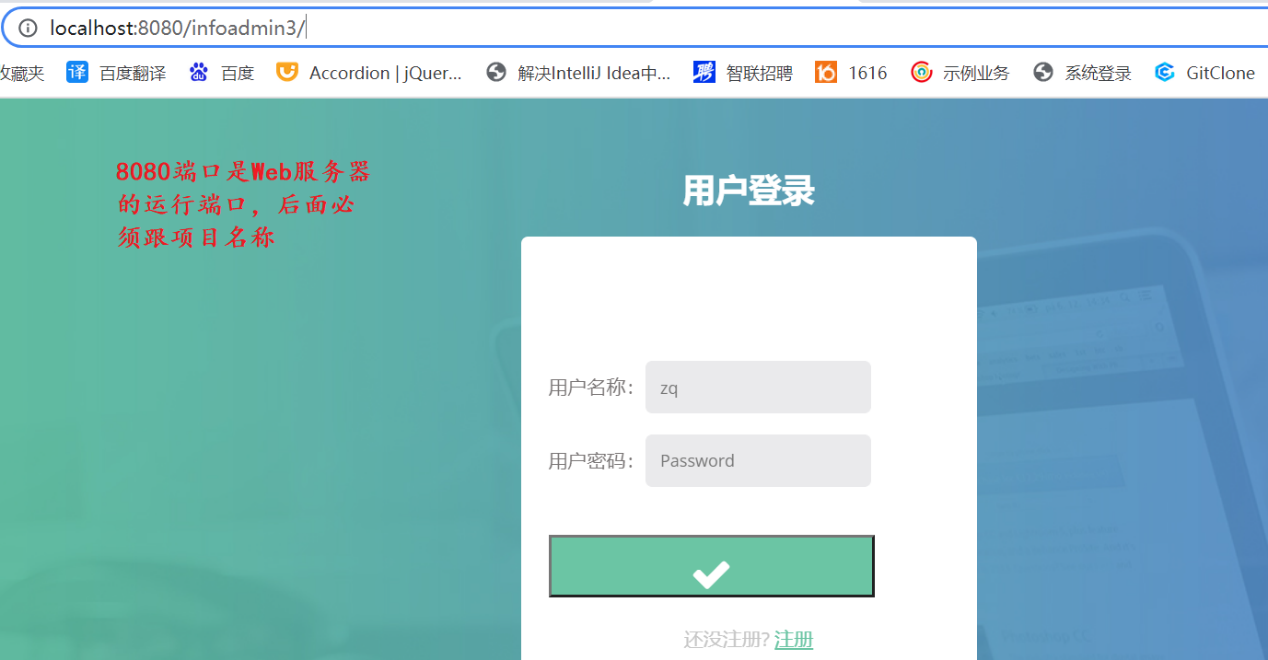
要查看项目，必须要拥有管理权限，用户名/密码和角色在Web服务器的/conf/tomcat-users.xml文件中设置

**<role rolename="manager-gui"/>**

<user password="admin" roles="admin-gui,standard,manager-script,manager-gui" username="admin"/>

(6)运行

与IDE编译调试时不同，application.properties中的端口和上下文路径失效，访问时必须加项目名称。



8080端口是在Web服务器的conf/server.xml文件中定义的。

<Connector URIEncoding="UTF-8" connectionTimeout="20000" port="**8080**" protocol="HTTP/1.1" redirectPort="8443"/>