# 道纲

有物混成，先天地生，寂兮廖兮，独立而不改，周行而不殆，可以为天地母。吾不知其名，强字之曰道，强为之名曰大。大曰逝，逝曰远，远曰反。故道大，天大，地大，人亦大。域中有四大，而人居其一焉。人法地，地法天，天法道，道法自然！

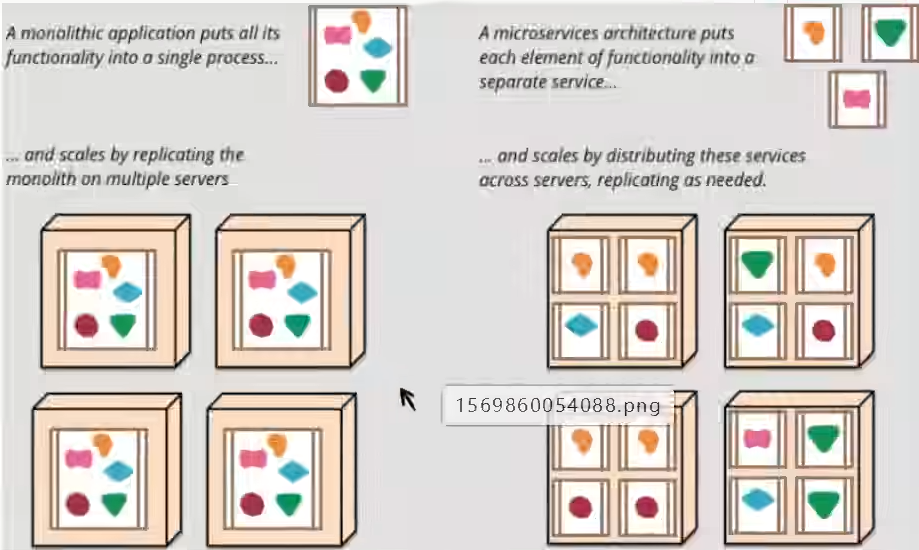
世间万物皆不断变化，技术更是不断更新迭代，而人若不背道而驰，则必须不断学习，此为顺势而为，合道。

而学习的本身除了学会如何使用某种技术，更重要的是如何去学习新东西，如何持续学习，如何超前了解并掌握未来前景。

人世间最大的悲哀不就是浪费了年轻的大好时光吗？

# 微服务架构

微服务是一种架构风格，它要求我们在开发一个应用的时候，这个应用必须构建成一系列小服务的组合，通过http的方式互通。打破传统all in one的方式，把每个功能独立出来，把独立出来的功能元素动态组合，需要的功能元素才拿过来组合。



# 单体应用架构

所谓单体应用架构是指将一个应用中的所有应用服务都封装在一个应用中。无论是什么系统，都把各个功能放在一个war包内。统称all in one

优点：把war复制多分在部署到服务器上即可。

缺点：如果需要修改某一部分内容即使十分微小，也需要停掉整个服务器然后重新打包部署。经典用例：SSM项目，javaweb项目

# SpringBoot

简介：springboot也是一个javaweb的开发框架，它存在的意义就是更简洁，这也是很多新框架存在的意义。既然更简洁，同样的规则，约定大于配置。

几乎所有的技术框架似乎都遵循了一条主线规律：从一个复杂运用场景衍生出一种规范的框架，使用者只需要根据约束规范去配置而不需要手动实现。这些强大的配置功能就成了优点！

后来随着配置过于冗杂，于是有了“约定大于配置”的理念，进而衍生出一些新的解决方案。

这就是java企业级应用-》j2EE-》spring-》springboot的演化过程。

## SpringBoot入门

spring官网<https://start.spring.io/>提供了生成springboot应用程序的功能，idea也集成了该网站

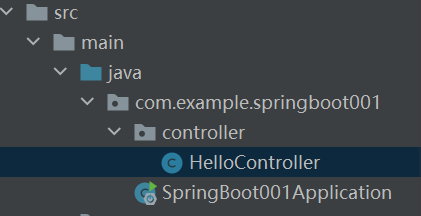
引入spring-boot-stater-web依赖，编写控制层接口，如下：

//springboot功能：自动装配

@RestController  
public class HelloController {  
 //路径：localhost:8080/hello  
 @RequestMapping("/hello")  
 public String hello(){  
 return "hello";  
 }  
}

注意项：

建包要和启动类同级，不能比启动类更高级！



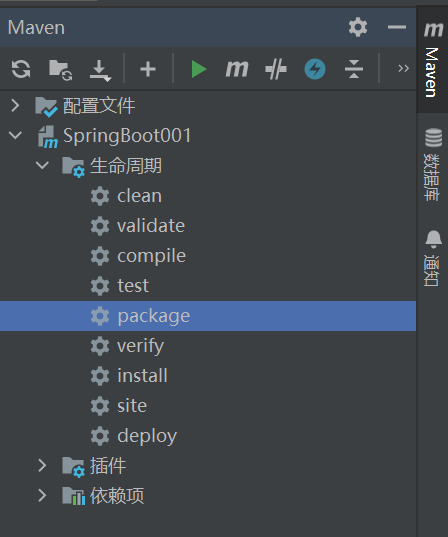
因为springboot是自动装配，如果项目引入了数据源，需要在配置文件中配置数据源，否则会报异常！

项目依赖解析：

1. spring-boot-stater-web用于实现http接口，该依赖包含了springmvc,并且使用tomcat作为默认嵌入式容器。
2. spring-boot-stater 是springboot项目依赖的头
3. build：构建配置部分，默认使用spring-boot-maven-plugin，配合spring-boot-stater-parent就可以把应用打包成JAR之间运行。

项目打包：

打包以后复制项目包到指定的路径，通过java命令行编译（java -jar 路径/包名）



## SpringBoot配置

SpringBoot使用一个全局的配置文件，配置文件名是固定的

* application.properties
  + 语法：key = value
* application.yml
  + 语法：key: 空格value

yml(yaml)配置文件缩进代表层级关系，大括号表示对象，中括号表示数组（-表示元素）

*# 对象*person:  
 name: 孙道名  
*# 对象行内写法，类似于JS*student: {name: xxx,age: 99}  
*# 数组*pets:  
 - cat  
 - dog  
 - pig  
*#数组行内写法*animal: [dog,pig,bird]

配置文件注入对象，springboot自动装配就利用@ConfigurationProperties

//类上添加两个注解，第一个表示此为组件，可以被扫描到

//第二个注解将yaml配置文件中的person对象引入，代表该类有一个实例对象person

//注入person实例只需要注解@Autowired

@Component  
@ConfigurationProperties(prefix = "person")  
public class Person {  
 private String name;  
 private int age;  
}



使用第二个注解需要pom.xml中导入依赖

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
</dependency>

//指定配置文件  
@PropertySource(value = "classpath:yi.properties")  
public class Person {  
 //指定配置文件需要使用@Value结合el表达式取出配置文件中的值  
 @Value("${name}")  
 private String name;

yml配置文件中使用spel表达式：${a:b}值不为空则取a,b为默认值，只有值不存在时使用！

*# 对象*person:  
 name: 孙道名${random.int}  
 age: 99  
 birthday: 1999/07/27  
 map: {A: a,B: b}  
 list: [a,b,c,d,e,f,g]

### JSR303数据校验

pom依赖：

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

在需要校验的实体类上使用注解@Validated

@Validated//数据校验  
public class Person {  
 @Email  
 private String email;  
 @NotNull  
 private String name;  
 @Null

//详细更多注解百度百科  
}

### 多环境配置以及配置文件位置

配置文件application.properties/yaml/yml优先级问题

项目目录下新建config目录，该目录下的application配置文件优先级最高

其次就是项目目录下的application配置文件

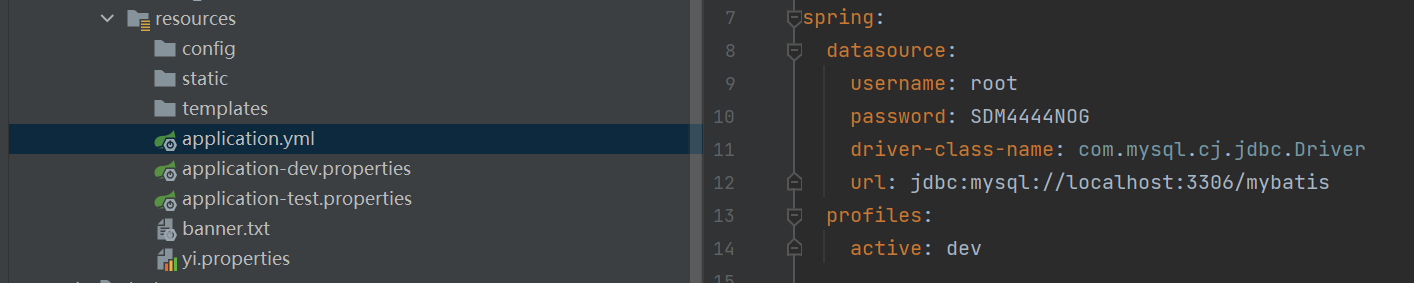
再其次就是resources目录下config目录下的配置文件

最后才是resources目录下的配置文件

即file:/config/ > file:/ > classpath:/config/ > classpath:/

项目默认的配置文件是类路径下的名为application的配置文件，通过spring.file.active=xxx指定配置文件以此来切换开发环境。

示例：指定环境为application-dev环境



*#单文件配置多个环境,默认第一个开发环境！*server:  
 port: 8081  
*#使用---代表新配置文件，profiles起名为application-xxx*---  
server:  
 port: 8082  
spring:  
 profiles: dev  
---  
server:  
 port: 8083  
spring:  
 profiles: test

### 自动装配原理\*\*\*

pom.xml中，

spring-boot-starter-parent依赖于父类工程

spring-boot-dependencies 核心依赖都存放在该配置中

spring-boot-stater：启动器！

spring-boot-stater-web：导入web环境下的所有依赖

springboot会将所有的功能场景都变成一个个启动器，我们需要什么功能只需要在pom.xml中引入该功能对应的启动器依赖：spring-boot-stater-xxx

@SpringBootApplication注解：源码如下，层层深入！

@SpringBootConfiguration//springboot配置  
@EnableAutoConfiguration//自动注入配置  
@ComponentScan//扫描包

springboot的自动化配置是非侵入式的，任意时刻，用户可以自定义配置代替自动化配置的某一个部分。

启动：

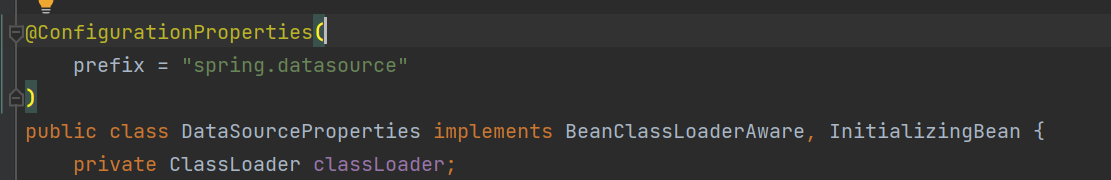
SpringApplication.*run*(SpringBoot001Application.class, args);

SpringApplication主要做了以下四件事情：

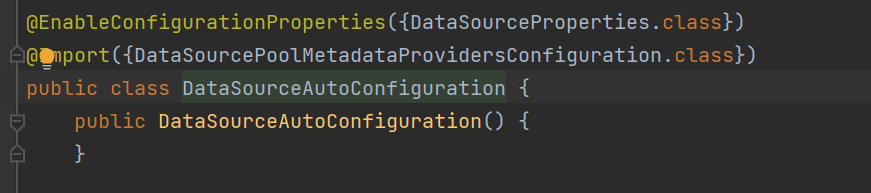
1. 推断类型是普通项目还是web项目
2. 找到并加载所以可用初始化器，设置到initializers属性中
3. 找到所有应用程序监听器，设置到listener属性中
4. 推断并设置main方法的定义类，找到运行的主类。

springboot所有配置文件在启动的时候被扫描并通过@@ConfigurationProperties注解将配置文件信息注入到对应的properties类中完成该类的实例化！

以JDBC举例：这是DataSourceProperties利用注解完成实例化



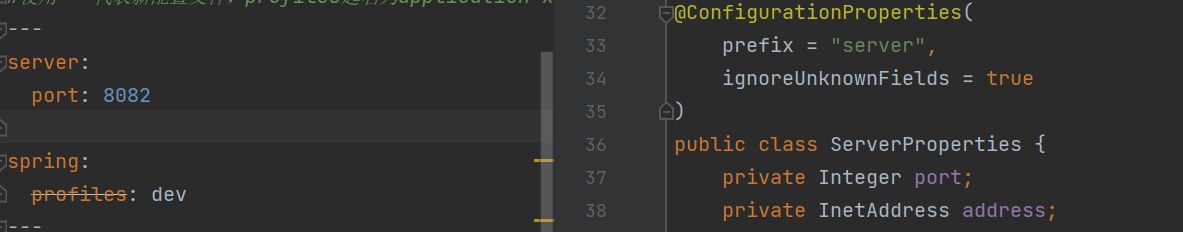
实例化完成以后再将该类加载到自动配置类中



spring.factories所有的自动配置类都在这里，但是不一定生效。要判断条件是否成立，只有导入了对应的start就有对应的启动器，有了启动器，我们自动装配就会生效。然后就配置成功！

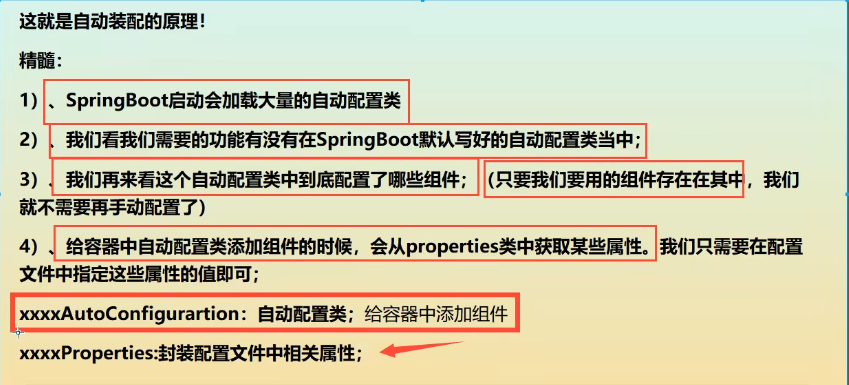
配置文件中能够配置的选项说明都对应着某个xxxproperties类，而该xxxproperties类又对应着某个xxxautoConguration类。

### 配置原理



在右侧源码中引入了server.properties配置文件对应ServerProperties这个类，同理，在yml或者properties配置文件中能配置的都对应着该类中的可写属性。

总结补充：在配置文件中配置debug:true查看哪些配置生效哪些没有生效



## SpringBoot Web开发

springboot到底配置了什么？能不能修改？能修改哪些？能不能扩展？

自动装配：

* xxxxAutoConfiguration：向容器中自动装配组件
* xxxxProperties：自动配置类，装配配置文件中自定义的内容

### 资源和路径问题

#### 静态资源以及映射

查看源码：crtl+n按名字搜索类名，crtl+h查看类的继承关系

源码中WebMvcProperties类中有一个属性如下：

说明静态资源映射地址默认就是 ip:端口号/\*\*\*

private String staticPathPattern = "/\*\*";

源码中WebProperties的内部类Resource属性

private static final String[] *CLASSPATH\_RESOURCE\_LOCATIONS* = new String[]

{"classpath:/META-INF/resources/", "classpath:/resources/", "classpath:/static/", "classpath:/public/"};

所以，默认自定义的静态资源路径右四个。

源码中定义了四个路径存放静态资源

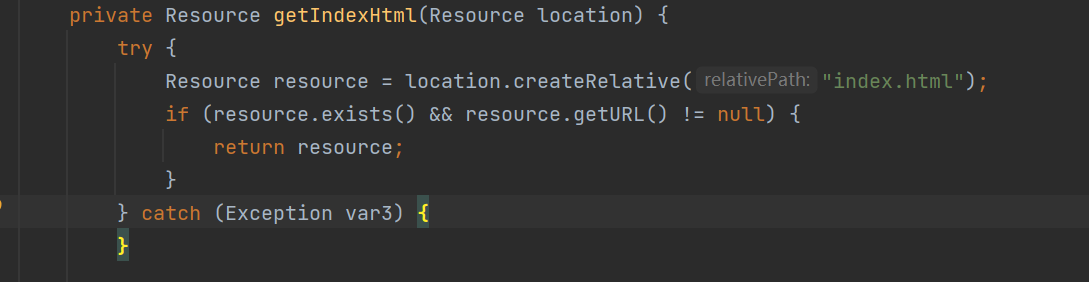
* classpath:META-INF/resources优先级最高
* classpath:resources/ 其次
* classpath:static/其次
* classpath:public/其次
* 也可以自定义静态资源，需要自定义映射路径以及资源位置
* spring:  
   mvc:  
   static-path-pattern: /sdm/\*\* *#映射路径* web:  
   resources:  
   static-locations: classpath:/public/ *#资源位置*

一旦自定义资源路径，默认路径失效！

前端页面中，/代表了项目的根目录(Resources)

#### 默认首页

部分源码如图，得出在默认的资源目录下寻找index.html作为网站首页，如果有多个首页，则根据路径优先级选择首页



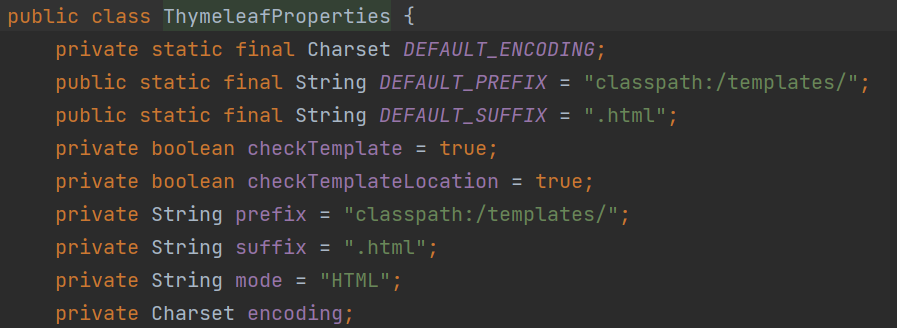
### 模板引擎

SpringBoot默认不支撑JSP，推荐使用模板引擎。模板引擎可以将数据解析道html页面上，本质上和jsp类似，jsp本质也是一个模板引擎。这里使用thymeleaf引擎！

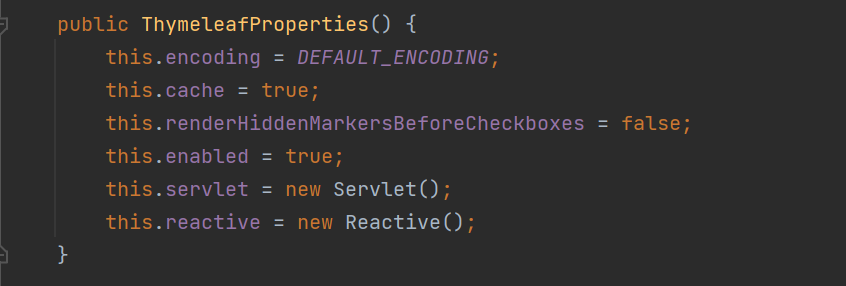
模板引擎的作用就是我们写一个页面默认，对于动态的值，通过表达式取出来。

thymeleafproperties源码

根据默认前后缀，得出这是针对类路径下的templeates文件夹的html页面！



默认构造方法，可以选择在配置文件中配置



themeleaf官网<https://www.thymeleaf.org/>

使用themeleaf需要给html文件添加头

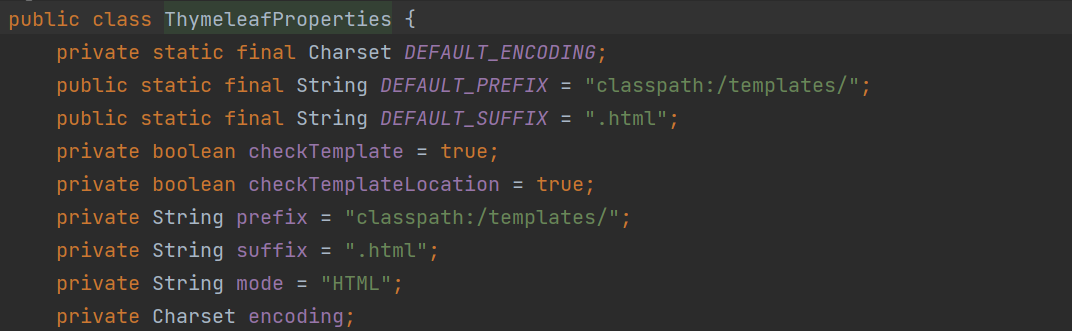
<html xmlns:th="http://www.thymeleaf.org">

theleaf可以接管替换所有的html元素，th:元素

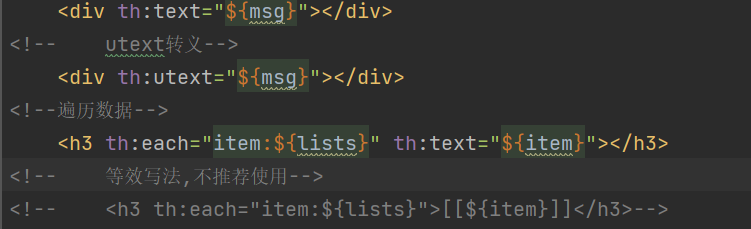
<div th:text="${msg}"></div>//msg在控制器中被添加

### Thymeleaf语法

springboot中提供了thymeleaf引擎的默认配置类，thymeleafproperties

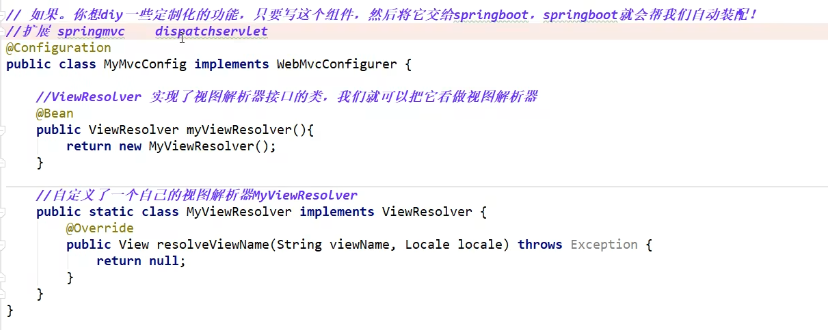


默认解析的路径是类路径下templeates路径下的html文件



### SpringMVC扩展

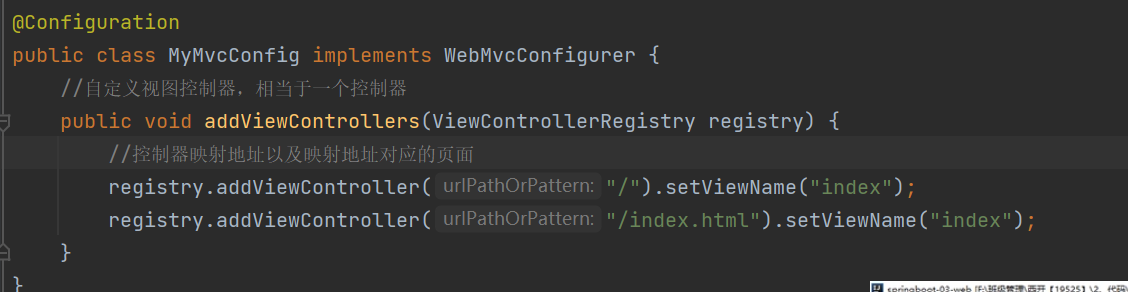
扩展mvc,只需要添加注解@Configuration并且类型是WebMvcConfigurer（实现该类即可）,注意，不能标注@EnableWebMvc,@Bean将自定义视图解析器交给bean容器管理



自动配置的原理都是一样的，通过分析源码得出结论。

springboot在自动化配置时，先看容器中有没有用户配置，如果用户配置了，就使用用户配置的，否则就使用默认的！如果有些组件有多个，比如视图解析器，就会把用户配置的和自己默认的组合使用。

扩展WebMvc时，还可以添加视图控制器和拦截器，等等………



### WebDemo注意点

所有页面都要被thymeleaf结构，页面中对静态资源的连接也需要使用thymeleaf,即使项目路径发生变化，thymeleaf结构的页面依旧能够找到静态资源。

国际化：

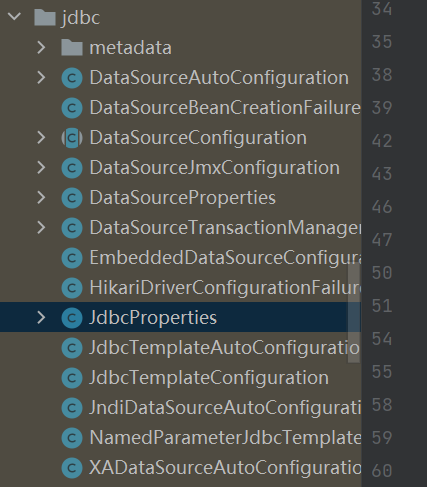
1. 配置i18n文件
2. 自定义组件LocaleResolver
3. 将组件配置到 spring容器中，@Bean

### Spring Data

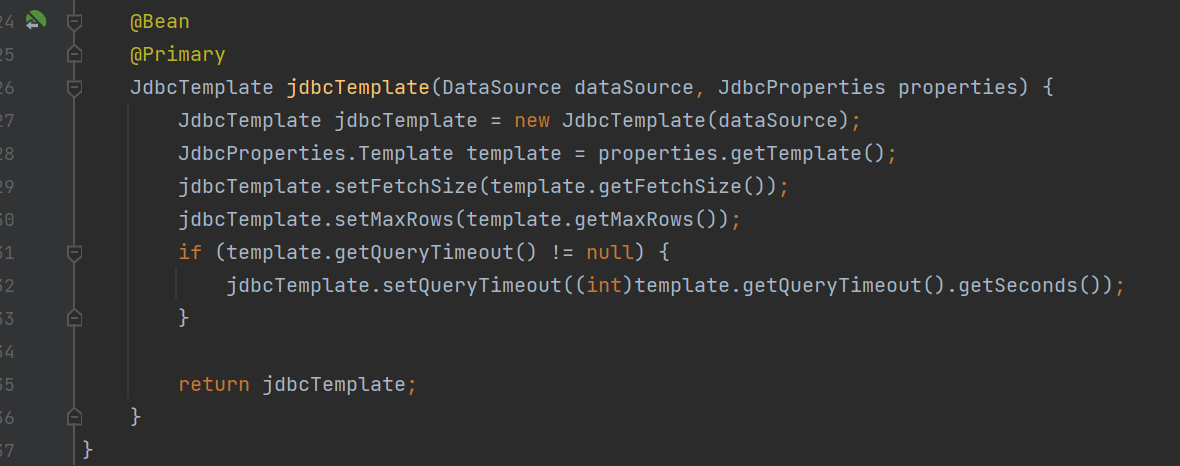
#### 整合JDBC

pom引入依赖JDBC，mysql driver，web依赖！

在spring-boot-autoconfigure找到JDBC目录，分析源码。



jdbcTemplate产生一个JdbcTemplate类型的bean,由spring容器管理！ 该bean可用于执行sql语句。



示例：

@RestController  
public class UserController {  
 @Autowired  
 JdbcTemplate jdbcTemplate;//spring容器中有该实例  
 @RequestMapping("/users")  
 public List<Map<String,Object>> getUsers(){  
 String sql = "select \* from user";  
 List<Map<String,Object>> users = jdbcTemplate.queryForList(sql);  
 return users;

#### Druid数据源

德鲁伊数据源是阿里巴巴开源平台的一个数据库连接池实现，结合了C3P0,DBCP，PROXOOL等数据库池的优点，同时加入了日志监控（log4j）,所以如果使用该日志监控功能，需要引入log4j依赖，示例如下：

@Configuration  
public class DruidConfig {  
 @Bean  
 @ConfigurationProperties(prefix = "spring.datasource")  
 public DataSource druidDataSource(){  
 return new DruidDataSource();  
 }  
 //后台监控功能  
// springboot内置了servlet容器，  
// 所以没有web.xml，替代方法就是将ServletRegistrationBean注入，需要什么bean，就new  
// 这里需要使用德鲁伊数据源的监控功能，也就是StatViewServlet类，所以有如下操作  
 @Bean//把该bean注入容器  
 public ServletRegistrationBean<StatViewServlet>

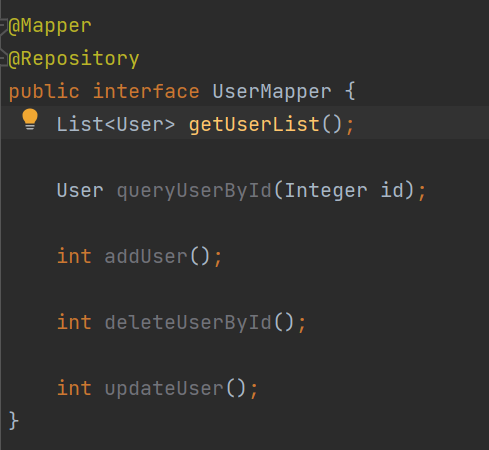
statViewServlet(){  
 ServletRegistrationBean<StatViewServlet> bean  
 =new ServletRegistrationBean<>(new

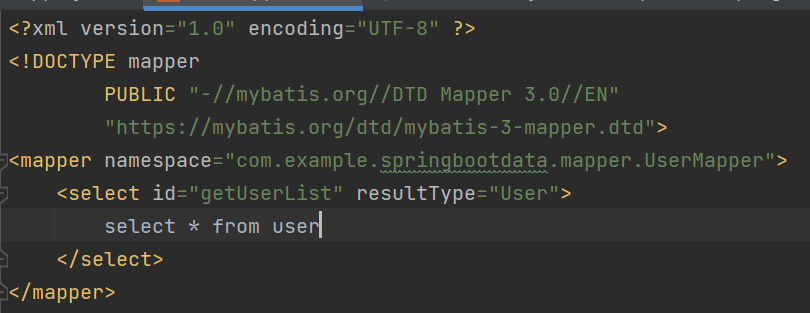
StatViewServlet(),"/druid/\*");  
 //后台人员的登录  
 Map<String, String> initParameters = new HashMap<>();  
 //配置账号密码,key是固定的  
 initParameters.put("loginUsername","admin");  
 initParameters.put("loginPassword","SDM4444NOG");  
 //运行谁可以访问  
 initParameters.put("allow","");  
 //禁止谁访问 initParameters.put("xxx","ip地址");  
 bean.setInitParameters(initParameters);  
 return bean;  
 }  
}

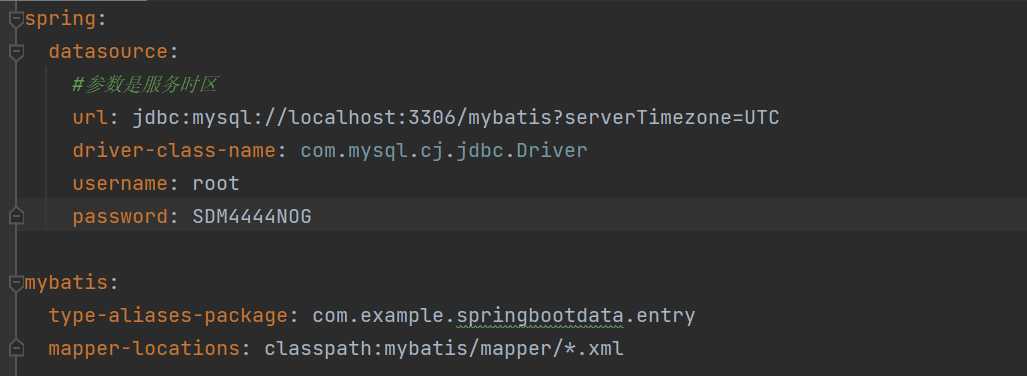
#### 整合mybatis

和spring类似，配置mapper文件以及对应的mapper接口，不同的是mybatis配置在全局配置文件中完成！

mapper接口添加注解@Mapper，@Repository表示这是一个mapper以及存储型bean,控制层自动注入bean即可！







#### 整合Redis

SpringBoot操作数据，spring-data,jpa,jdbc,mongodb,redis!

SpringData也是和SpringBoot齐名的项目

在pom.xml中引入redis的starter依赖

导入配置

连接测试

redis的properties配置如下：



### Web安全框架

#### SpringSecurity

web开发中，安全第一！过滤器，拦截器等

做网站：在设计之初就应考虑安全问题。

安全的框架：shiro,SpringSecurity

功能：认证，授权！

* 功能权限
* 访问权限
* 菜单权限
* 拦截器，过滤器（原生代码）

Spring Security是针对spring项目的安全框架，也是spring boot底层安全模块的技术选型，它可以实现强大的web安全控制。对于安全控制，只需要引入springboot默认的security安全模块，进行少量的配置即可实现强大的安全管理。

记住几个类：

* WebSecurityConfigurerAdapter自定义Security策略
* AuthenticationManagerBuilder自定义认证策略
* EnableWebSecurity开启websecurity模式，@Enablexxx开启某个功能

Spring Security的两个主要目标是认证和授权

官网：<https://spring.io/projects/spring-security>

#### Shiro

### Swagger

前后端分离时代：Vue+SpringBoot

* 后端：后端控制层，服务层，数据访问层
* 前端：前端控制层，视图层

前后端如何交互？==》API

前后端相对独立，松耦合

前后端甚至可以部署在不同服务器上

问题：

* 前后端无法做到即时协商，今早解决

解决方案：

* 制定schema,实时更新API
* 早些年指定word文档
* 前后端分离：
  + 前端测试后端接口：postman工具
  + 后端提供接口，实时更新

Swagger

* 号称世界上最流行的API框架
* RestFul文档在线自动生成工具，API文档与API定义同步更新
* 直接运行，在线测试API接口
* 支持多种语法：java,php等

项目中使用Swagger,需要引入依赖swagger-ui和swagger2

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>3.0.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>3.0.0</version>

</dependency>

SpringBoot集成Swagger

@Configuration  
@EnableSwagger2//开启swagger  
public class SwaggerConfig {  
}

### 任务

#### 异步任务

* 在异步任务方法上添加注解@Async
* 在启动函数上添加注解@EnableAsync

#### 邮件发送

* 引入依赖，spring-boot-starter-mail
* 开启发送邮件账号的SMTP服务（设置-账户）
* 配置邮件发送服务器和发送邮件的账户相关信息

#邮件发送服务器，qq邮箱就是smtp.qq.com  
spring.mail.host=smtp.qq.com  
#服务器端口  
spring.mail.port=587  
#发送邮件的账户  
spring.mail.username=2846456105@qq.com  
#该账户的秘钥  
spring.mail.password=dtvyzkeorbfgdgdi  
spring.mail.default-encoding=UTF-8

* 使用工具类，JavaMailSender类用于发送邮件，简单邮件类SimpleMailMessage,复杂邮件类（可以解析html标签,可以携带附件资源）MimelMessage复杂邮件类通过邮件发送类的方法创建,

MimeMessage mimeMailMessage = javaMailSender.createMimeMessage();



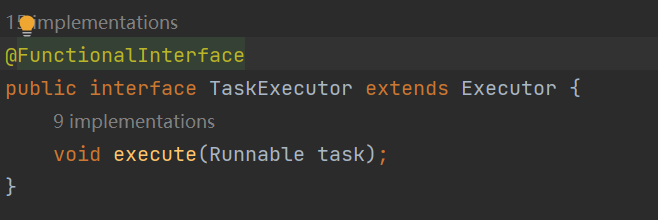
编写测试类，注入MailService（@Component把MailService作为组件注入spring容器中），发送邮件！



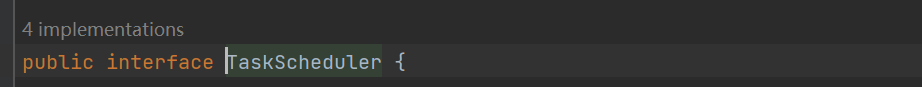
#### 定时任务

定时任务通过两个自带的工具类实现，启动函数上使用注解@EnableScheduling开启定时任务，定时任务不需要被调用，当满足定时条件时，自动执行

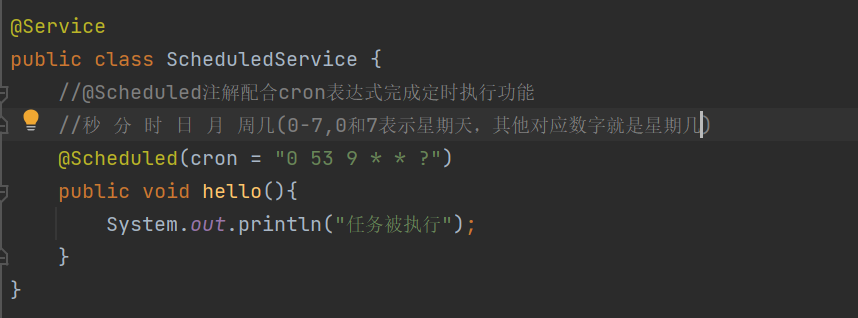
* TaskExecutor：任务执行者



* TaskScheduler：任务调度者



定时使用@Scheduled配合cron表达式使用



##### CRON表达式

表达式是一个字符串：数值之间用空格隔开

秒 分 时 日 月 星期 年（可省略）

通配符：

* \*：每，任意
* -：指定范围
* ，：指定固定的数
* /：增加幅度
* ？：只有日和周能使用，这两个字段存在一定的冲突，所以使用？表示不生效

详情百科

### 分布式系统理论思想

简介：分布式系统是若干独立计算机的集合，这些计算机对于用户来说就像是单个相关的系统。分布式系统是由一组通过网络进行通信，为了完成共同的任务协调工作的计算机节点组成的系统。

需要明确的是只有当单个节点的处理能力无法满足日益增长的计算存储任务，且提升硬件资源得不偿失时，才会考虑分布式系统。 分布式系统的引入回来单机系统所没有的很多问题（网络不可靠产生的种种问题），因此需要仔细斟酌。

#### RPC

Remote Procedure Call指远程过程调用，是一种进程间的通信方式，是一种技术思想而不是规范。它允许程序调用另一个地址空间（另一台计算机）的过程或者函数，而不用程序员显式编码这个远程调用的细节。即不管程序员是调用本地还是远程函数，本质上编写的调用代码基本相同。

RPC的两个核心：通信和序列化

##### Dubbo

Dubbo是一款高性能轻量级的java RPC框架，它主要提供了三大核心功能：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

##### zookeeper

zookeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Hadoop和Hbase的重要组件，它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护，域名服务，分布式同步，组服务等。

# 设计一个网站

步骤如下：

1. 设计前端
2. 设计数据库
3. 前端独立运行
4. 前后端数据对接JSON
5. 联调测试

前端解决：

前端：模板+自己手动修改

框架：组件（自己手动拼接）！BootStrap,Layui,semantic-ui,

* 栅格系统
* 导航栏
* 侧边栏
* 表单

使用框架自己手动拼接一套网站模板

* 首页
* 登录页

后台模板：必须要有一个自己的后台模板,推荐x-admin