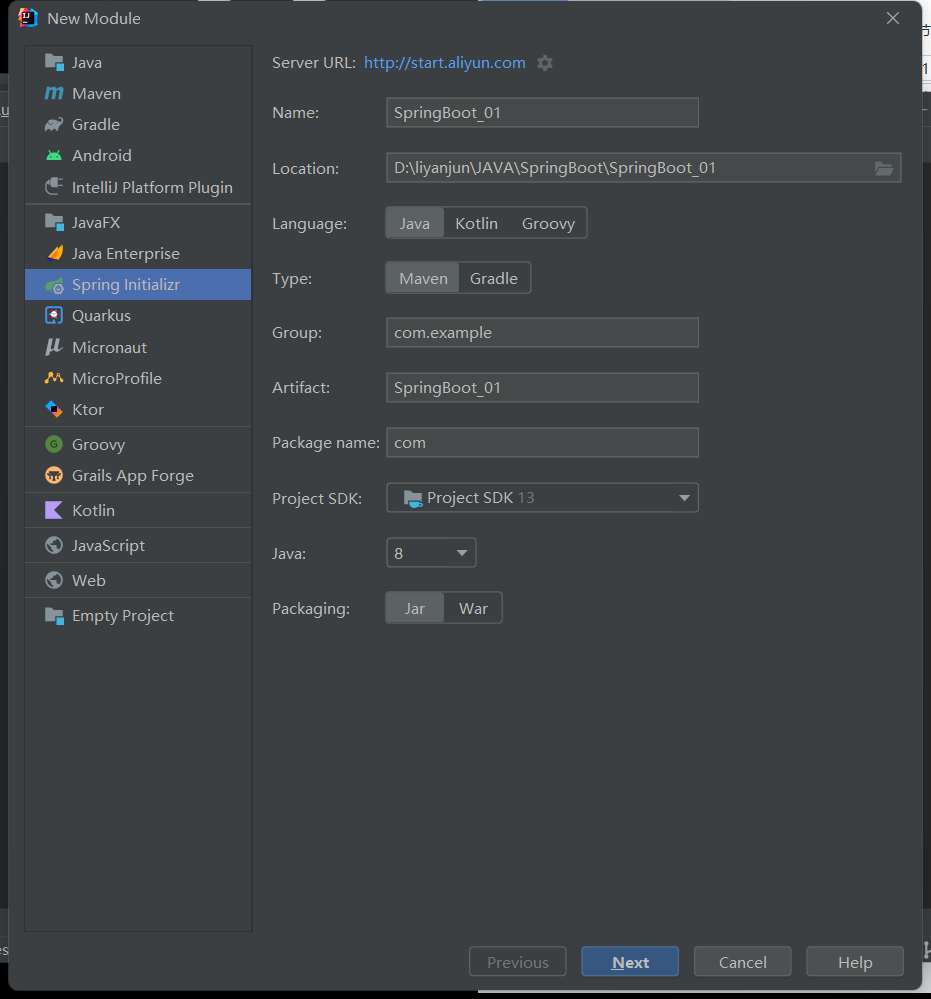
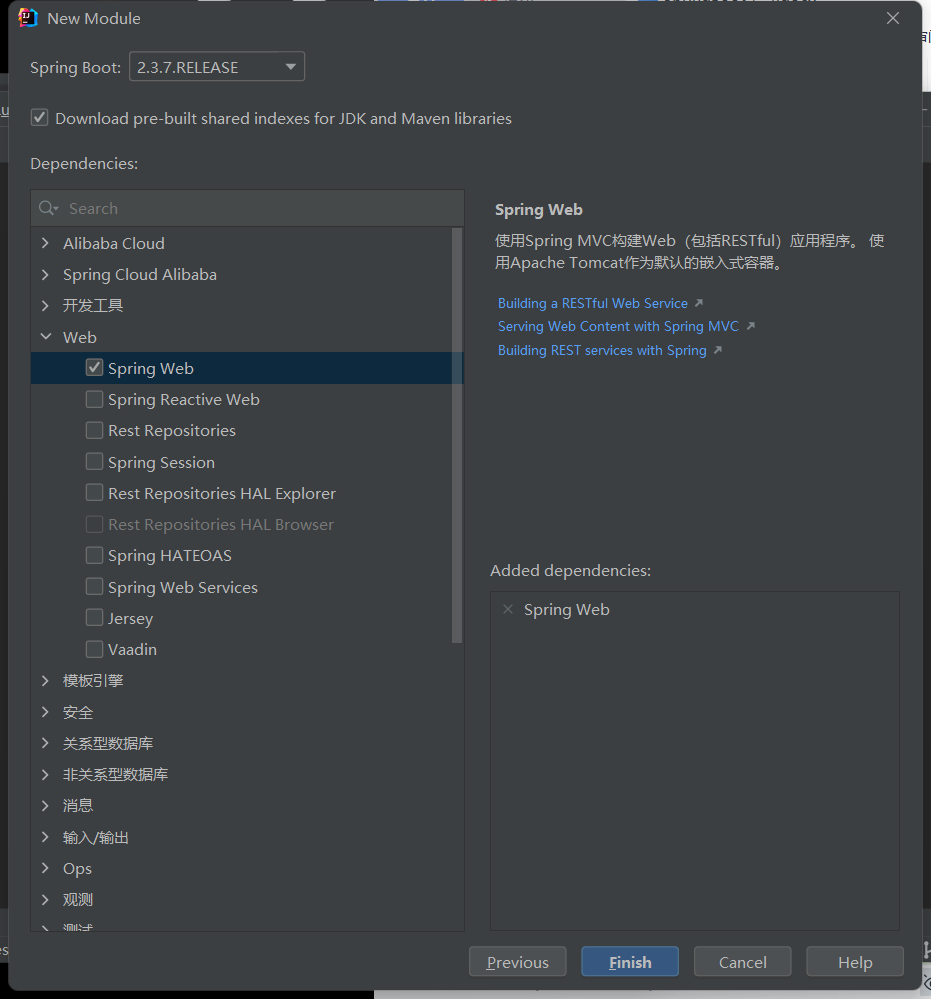
# SpringBoot的开发步骤

目的：用来简化Spring应用的初始搭建和开发过程

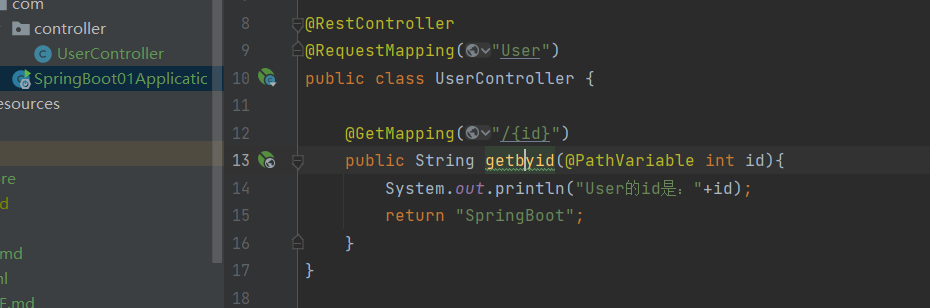
1. 建立SpringBoot工程



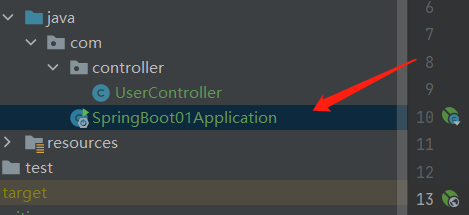
serverURL:http://start.aliyun.com



1. 写出controller



1. 启动Application类



1. 访问http://localhost:8080/控制器路径

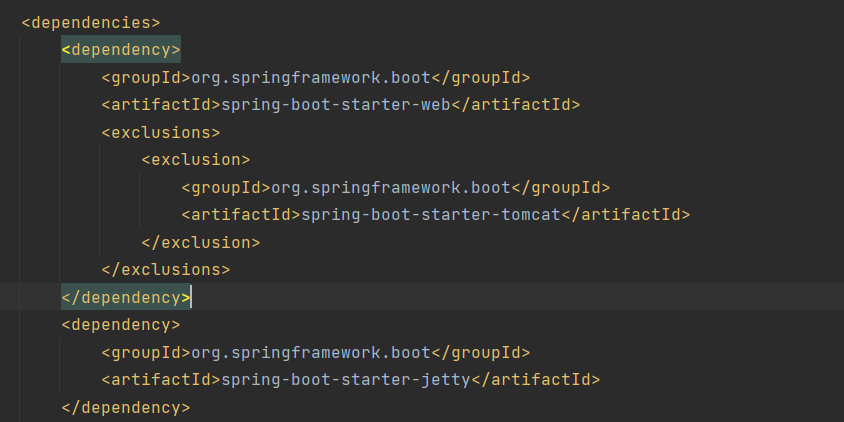
## 注意事项

1. 选择jdk时，要和tomcat位数一致，选择64位的。
2. controller要在启动类的同级别或者下一级别。

# SpringBoot配置信息修改服务器修改

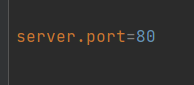
## **服务器修改**

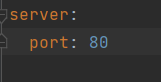
更改启动服务器为jetty



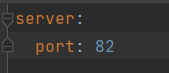
## 基础配置

### 配置档格式

第一种，在properties中写 

第二种，在yml档中写

冒号和数字中间有空格

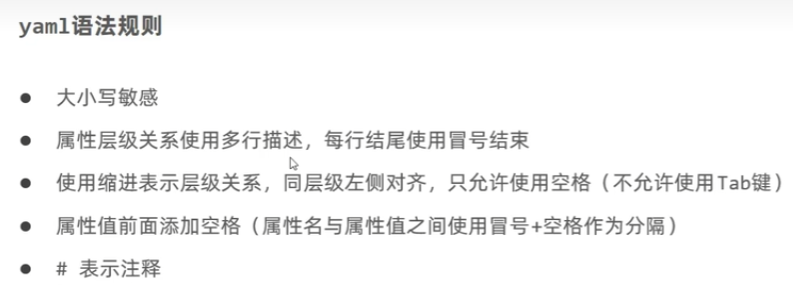
第三种，在yaml档中写

推荐使用yml结尾的

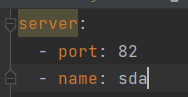
三个都存在的时候，优先级 properties > yml > yaml

### yaml



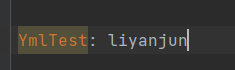


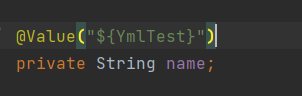
数组正前面使用-

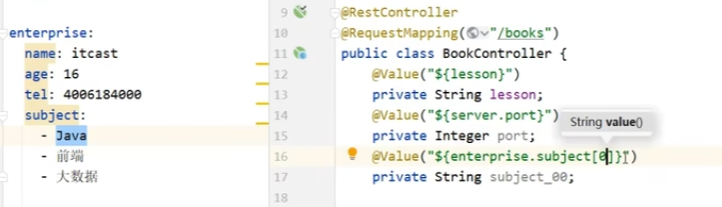


#### 读取yml档

1.使用@Value注解



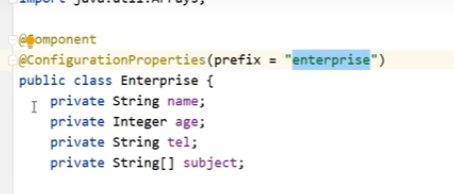




2.用Environment类来加载

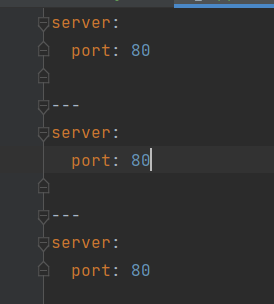


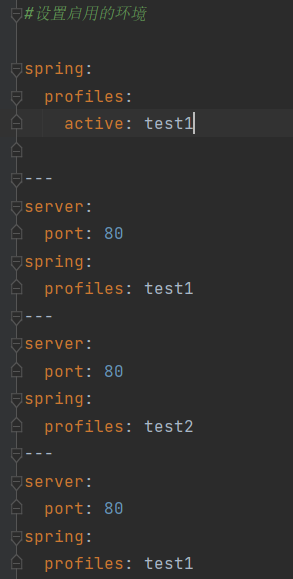
1. 使用bean对象来读属性



### 多环境启动、

设置多环境，每个环境之间使用---隔开



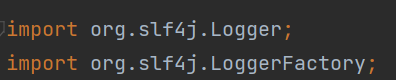


****

## 日志

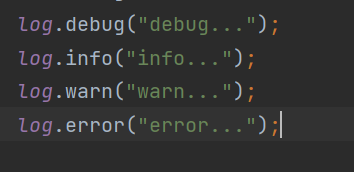
在controller里面记录日志

创建记录日志的对象，使用slf4j





打印日志



讲日志记录到文件信息

log：

file：

name: server.log



## 业务层设置事务回滚

## 删除操作的技巧

不要删除数据库的数据，可以在建立表的时候添加一个active的字段，删除操作只改变字段的值，默认true。

# 基于Springboot完成SSM整合

## 整合JUnit

测试需要和源目录同层

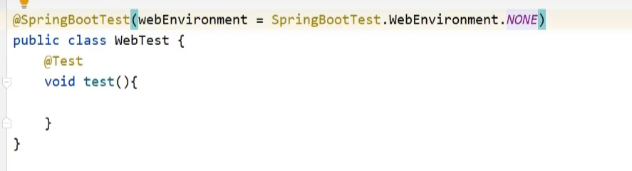
使用@SpringBootTest



或者可以指定



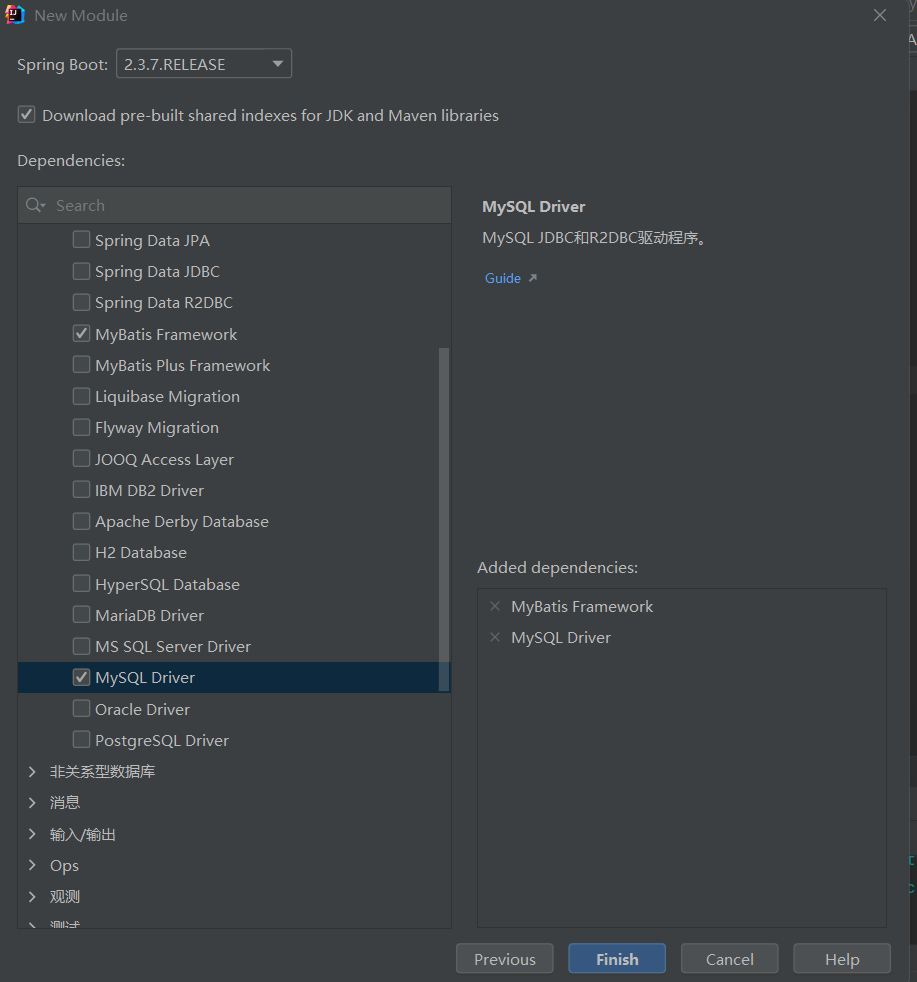
测试环境提供web环境



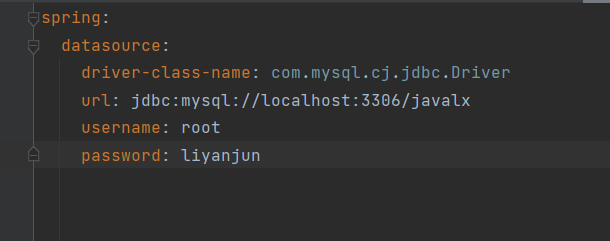


## 整合Mybatis

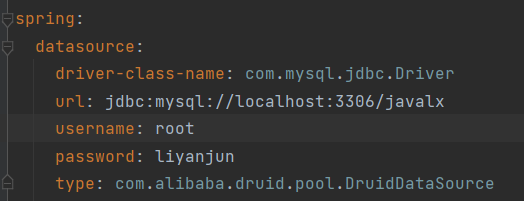
1.在开始加载Myabtis和相关数据库的依赖



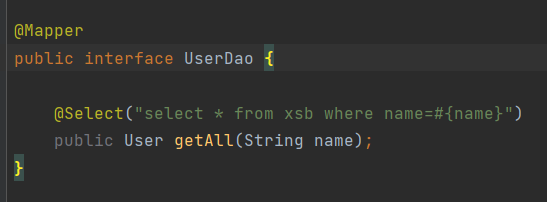
1. 在配置类中写datasource



通过type属性来确定数据源



1. 写出数据层，记得加上@Mapper注解



1. 测试一下

# MybatisPus

基于Mybatis框架基础上开发的增强工具，用于简化开发，提高效率。

不能和mybatis同时使用。

1. 导入坐标

<dependency>

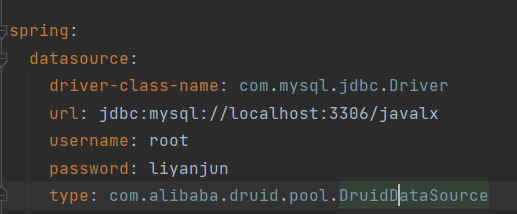
<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

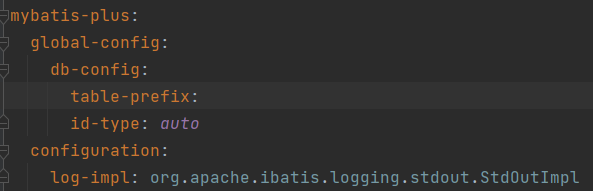
<version>3.1.0</version>

</dependency>

1. 在配置档配置数据库连接池和数据源

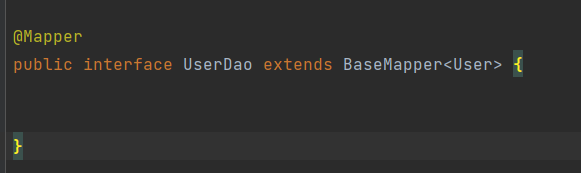


mybatisplus的配置



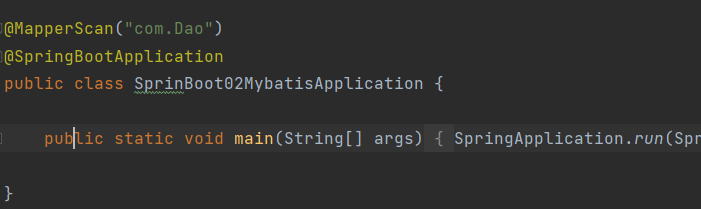
1. 创建实体类，表名和实体类名需要保持一致
2. 写数据层内容界面

继承一个BaseMapper类，泛型为实体类

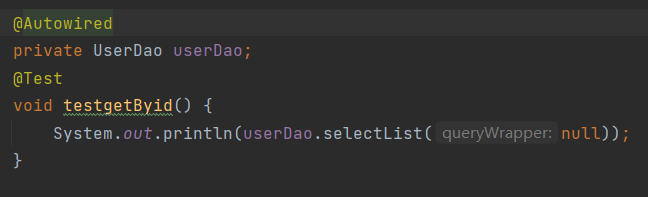


1. 在主启动类中添加扫描mapper的注解

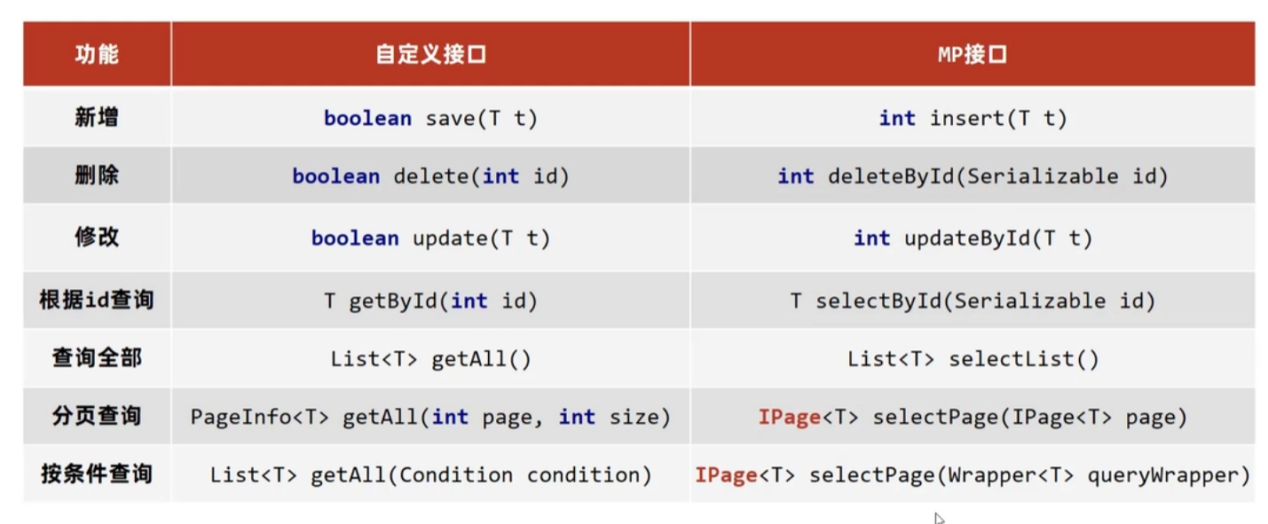
@MapperScan(“com.\*.mapper”)



6.使用



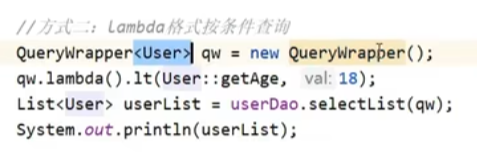
基础的sql语句都含有



## 条件查询

小于

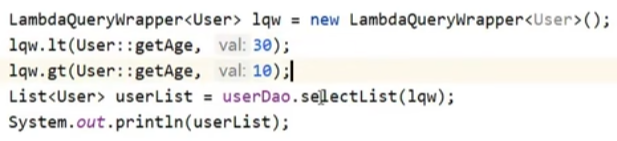






多条件

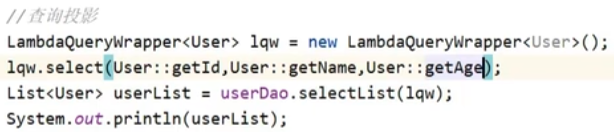
and



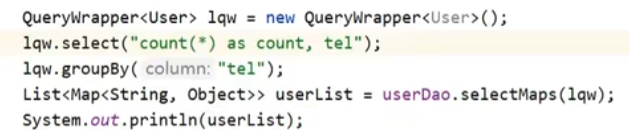
or

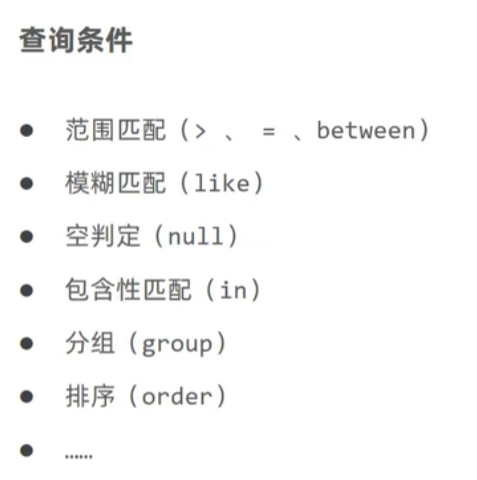


查询投影



查询，分组统计

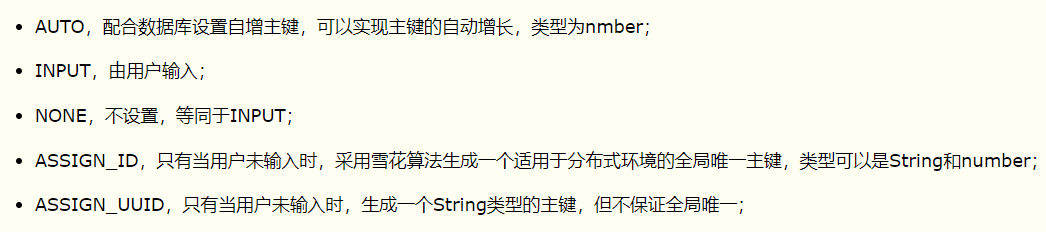




## 增删改







## 逻辑删除

删除操作不删除数据库中的数据

做一个标记，例如新增一个新的列，这个列标记为1表示删除，表示为0代表没有被删除

在配置档加一个全局配置



然后只需要在实体类中添加就ok



## 乐观锁

多个用户同时使用一个进程，会带来并发问题。

2000个请求以下使用乐观锁

乐观锁的实际原理



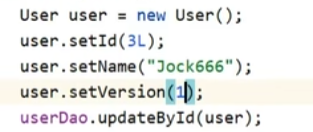
1. 在数据库表结构中加上表字段Version，表明谁在操作表数据
2. 在实体类中添加属性
3. 写一个拦截器

帮助我们修改sql语句

使用mp的拦截器

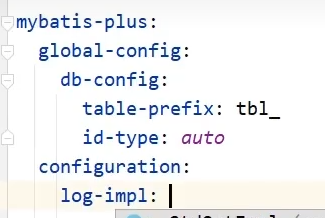


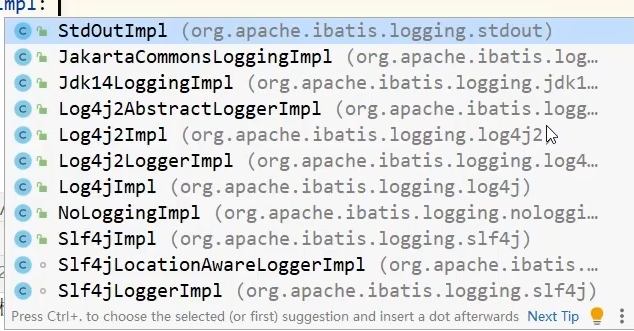
1. 在实现的时候，需要设置Version



## mp的调式日志

在配置文件yml中添加





# SpringBoot的页面开发

页面开发可以放到前端服务器（还不会）。

使用单体服务器（前后端代码在一起）

单体服务器中，页面放在：resources\static 中

如果程序出现一些莫名其妙的故障，去maven中clean一下。

# 热部署

项目不用重启，而看到项目的修改

使用工具devtools



<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

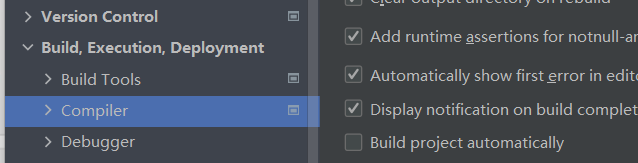
</dependency>

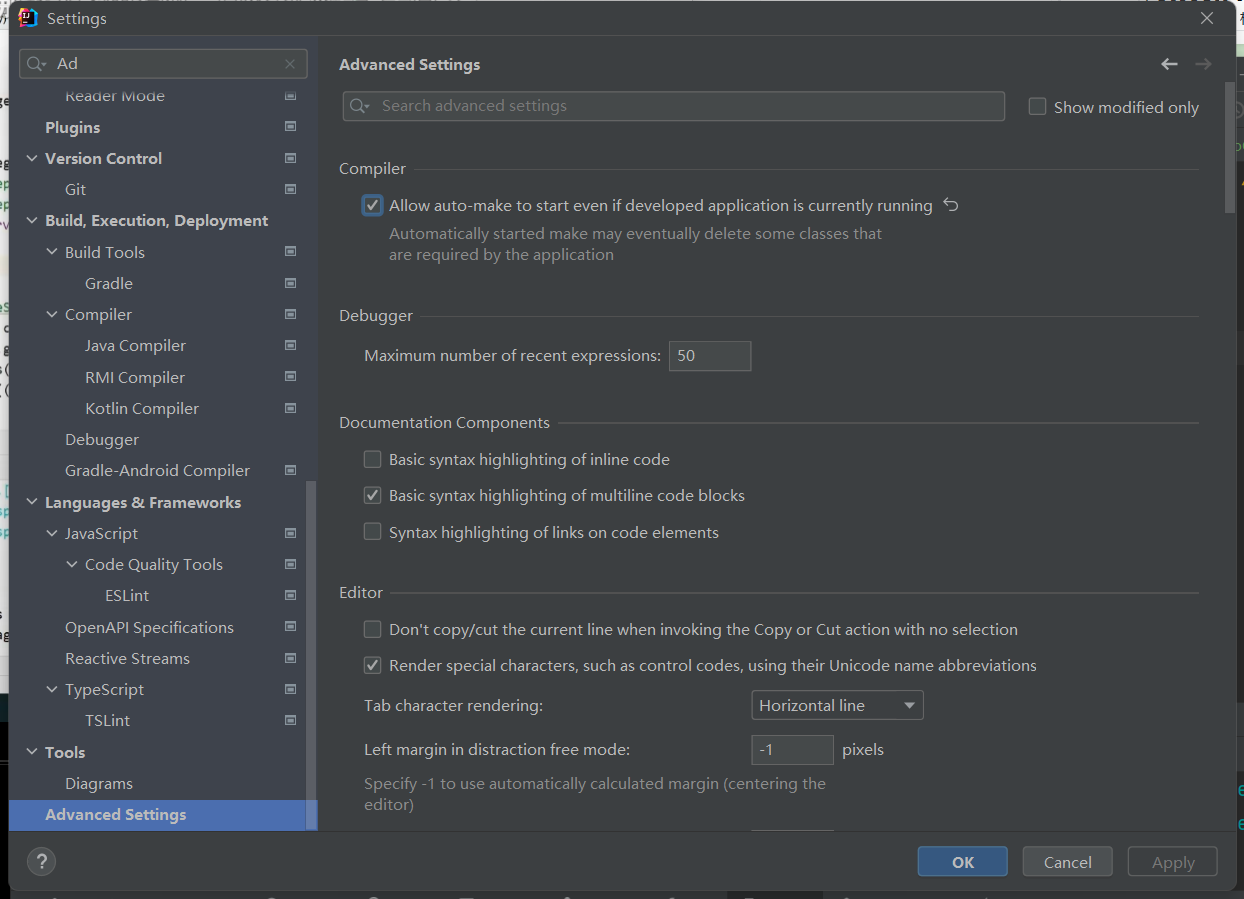
手动启动热部署：

然后修改后，build一下项目（ctrl+F9），就可以了

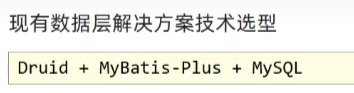
自动启动热部署：

可以在idea中，自动构建项目





# 数据层解决方案

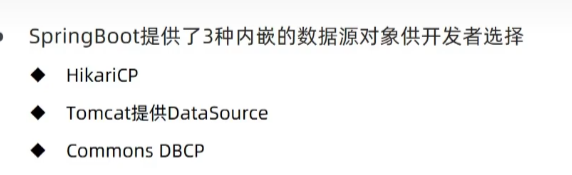


数据源：Druid

持久化技术：Mybatis

数据库： MySQL

## springboot内置的数据源：



默认使用：HikarCP，是最快的轻量级的数据源。

## spring默认持久化技术

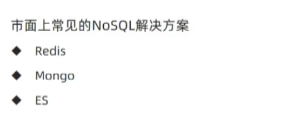
jdbcTemplate

## spring默认数据库

H2，HSQL, Derby数据库

这三个数据库都可以在内存中启动。

## 非关系形数据库



重点学习Redis

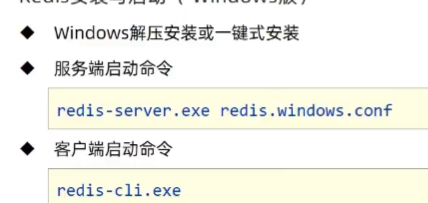
三个技术的简单介绍：

## Redis

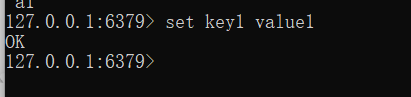
Redis是一款Key-value存储结构的内存级NoSQL数据库

支持多种数据存储格式，集群和持久化。

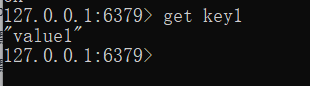
### Redis的启动



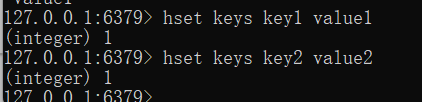
简单的key-vlaue，在redis客户端上可以设置key和value



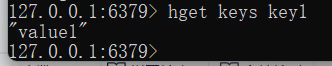
通过get命令查看



一个key中有多个key，每个key对应value，原理是hash，使用hset

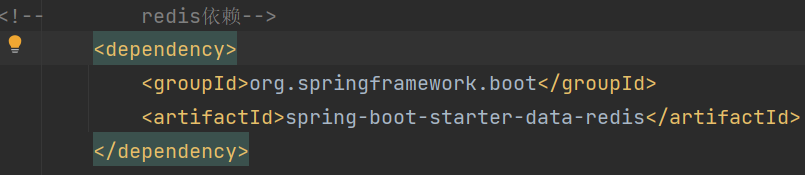


使用hget来获取值

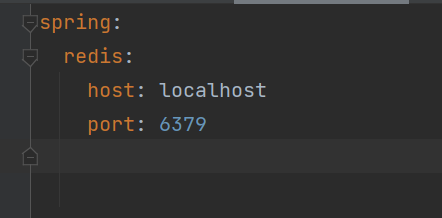


### springboot整合redis

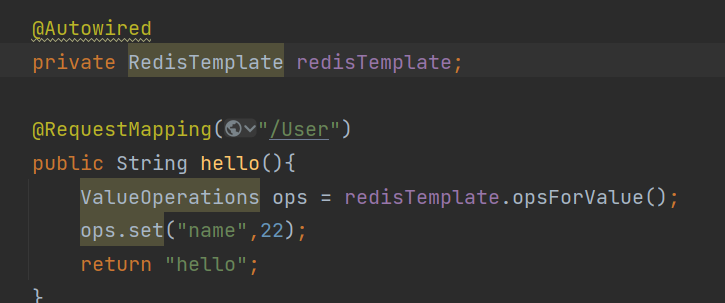
#### 引入redis的依赖



#### 配置redis



#### 使用



使用RedisTemplate就可以操作redis了

第一步要给redis设定操作的对象，opsForValue为简单的key-value结构。然后通过set，get方法，操作数据。

操作redis客户端需要使用



之后操作是一致的。

## Mongodb

是一个开源的，高性能的无模式的文档型数据库。

·临时存储和永久存储相结合，并且修改频率很高的数据，适合存储在数据库和Mongodb

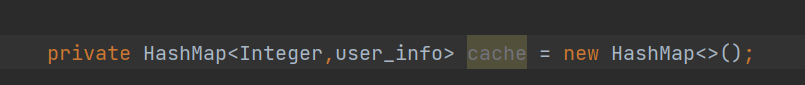
·临时存储，修改频率飞快，适合Mongodb

## ES

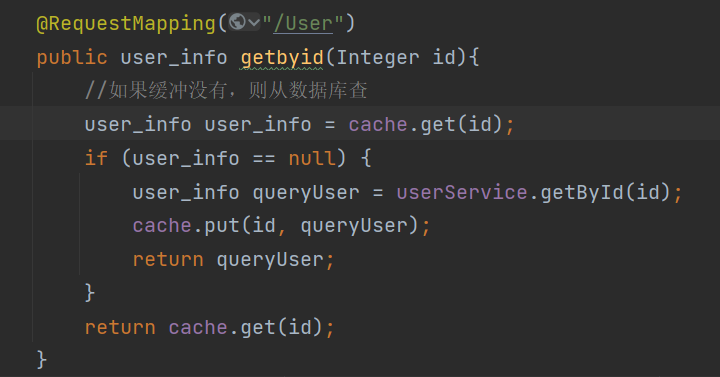
ES是一个分布式全文搜索引擎

## 缓存

存储在内存的数据临时存储介质。



利用主键id为key，id的信息user\_info为value来设定cache。

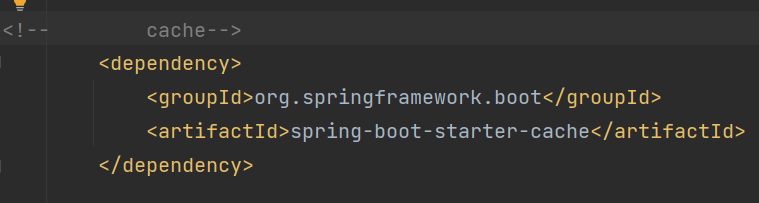


每次查找都将数据放入缓冲，缓冲有就无须在到数据库中查询。

## spring的缓存

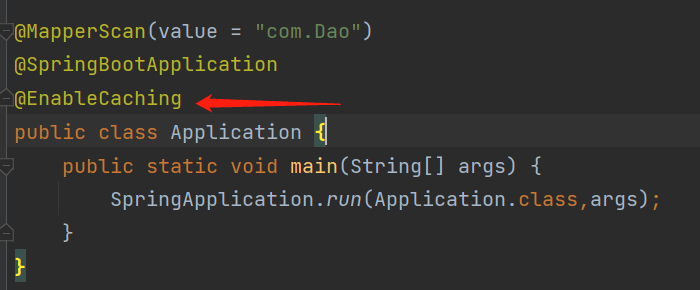
### 导入依赖

spring-boot-starter-cache



### 启动缓存

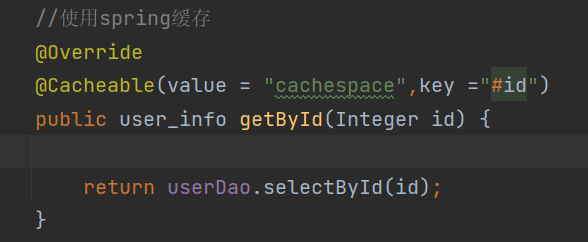
在启动类上添加@EnableCaching注解开启缓存



### 使用缓存

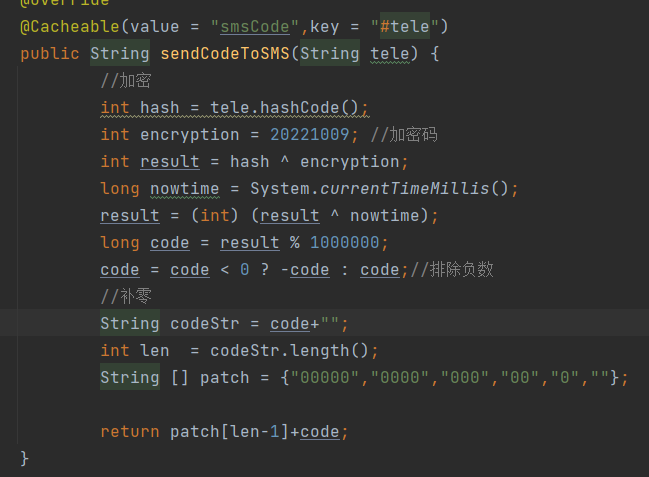
在需要缓冲的方法上使用@Cacheable

其中的value为缓存的空间，key为缓冲查找的值，#id可以匹配到下面的id值。



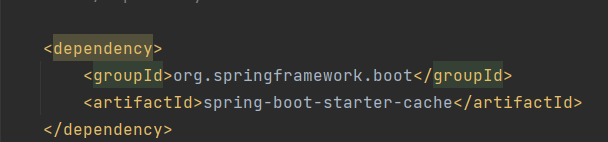
### ⭐手机验证案例

根据手机号制作验证码



## Ehcache缓冲

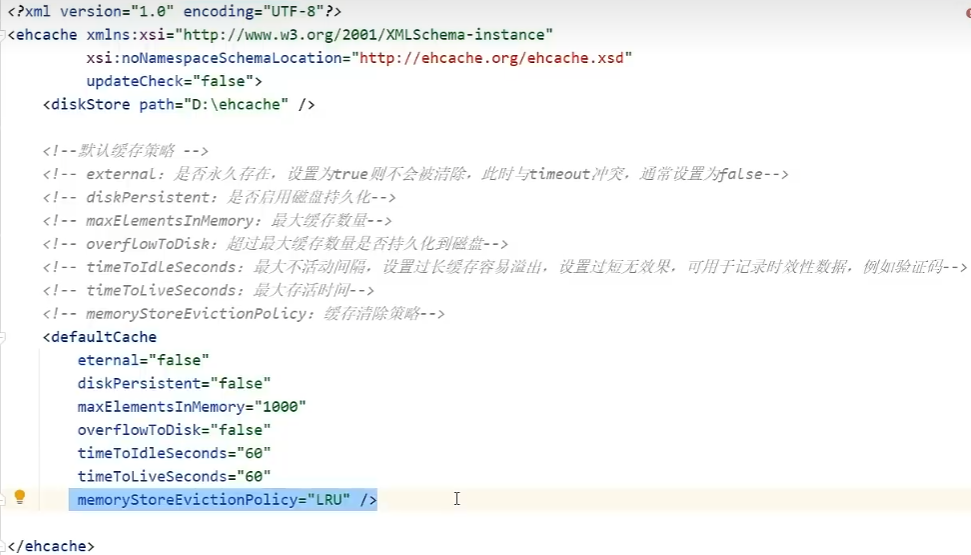
导入坐标：



在配置文件中添加cache的type为ehcache

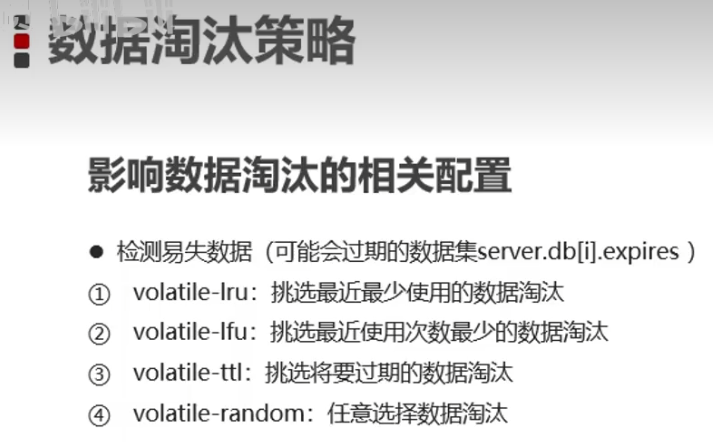


添加ehcache.xml配置文件，因为ehcache不是spring的原生组件，所以使用配置文件加载



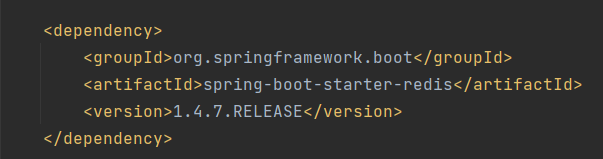
添加name属性来匹配cacheable的value。

### 数据淘汰策略

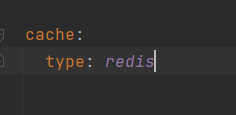
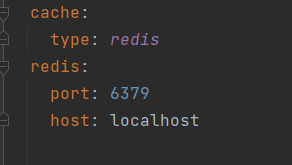


## redis缓冲

引入坐标



更改配置文件



启动redis客户端和服务器（win上）

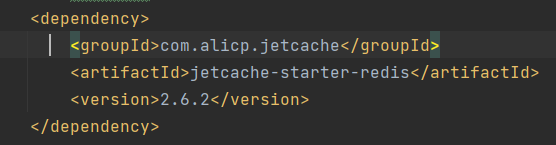
其他的缓存配置不用动

## jetcache框架

jetcache对springcache进行了封装，对原有功能基础上实现了多级缓存，缓存统计，自动刷新，异步调用，数据报表等功能。



导入依赖

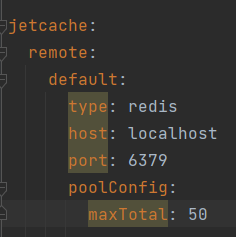


### jetcache本地缓存

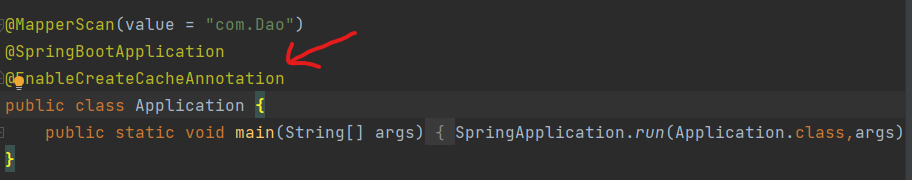
更改配置

### jetcache远程缓存

更改配置

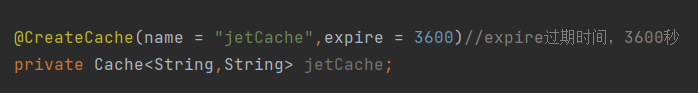


poolConfig一定要配置。

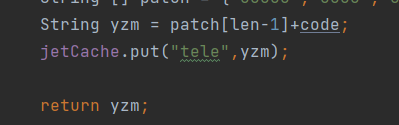


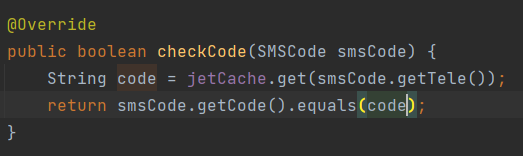
在application启动类中添加EnableCreateCacheAnnotation注解来启动jetcache。只有2.6.2可以使用了，在更高版本的jetcache中，此注解已经被废弃

在需要使用缓存的实现类中，打开jetcache的缓存



并在其中使用





# 定时任务

多线程的Timer定时器

定时框架技术，Quzrtz。

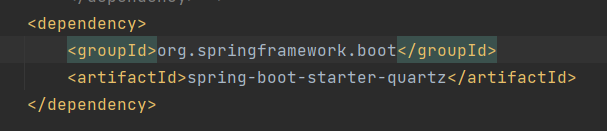
Job：定义具体执行的工作

JobDetail：用于描述定时工作的相关信息

Trigger：用于描述触发工作的规则。通常使用cron表达定义调度规则

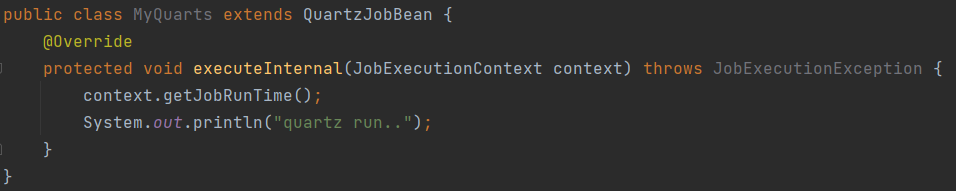
scheduler：描述了工作明细和触发器的1对应关系

导入坐标

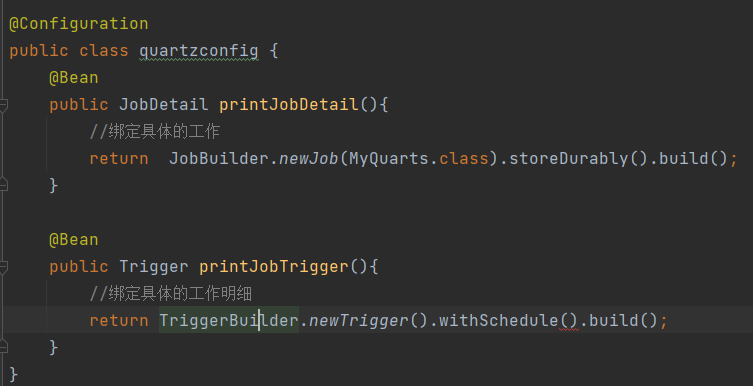


创建一个quartz类，继承QuartJobBean

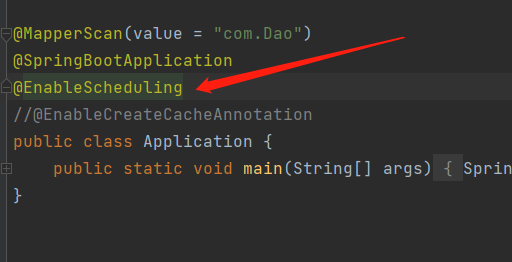
在executeInternal方法，写我们执行的任务。



创建quartz的配类



spring定时框架技术task



在启动类上开启@EnableScheduling

在需要定时任务的上面添加@scheduled(cron =”0/1 \* \* \* \* ?”)

每一秒执行一次，任意分钟，任意小时，任意天，任意月，不限星期



可以在spring中配置Task



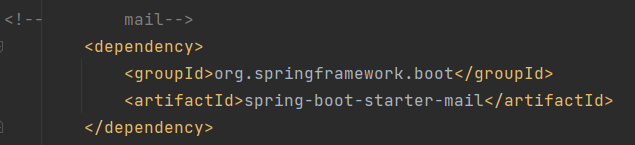
# ⭐发送简单邮件

SMTP，简单邮件传输协议，用于发送电子邮件的传输协议。

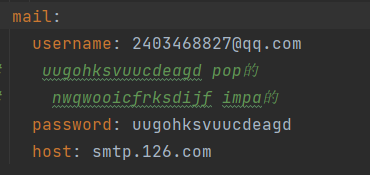
POP，用于接收电子邮件的标准协议。

IMAP，互联网消息协议，是POP的替代协议。（又邮件操作的双向同步）

导入坐标

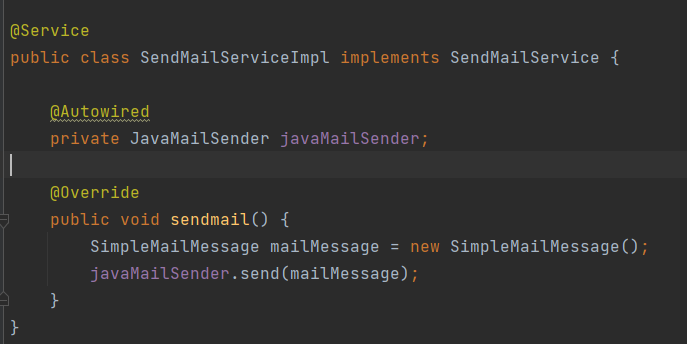


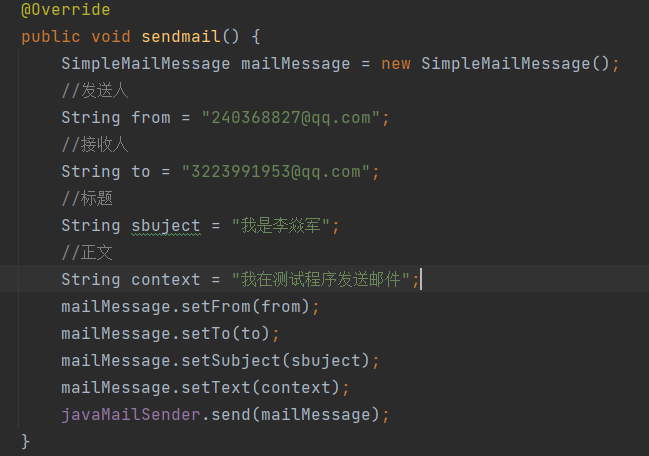
配置邮件



密码要从邮件账户去申请

使用JavaMailSender来发送邮件





需要发送人，接收人，标题，和正文

之后就可以发送邮件了。



使用复杂邮件



加一个true，解析html

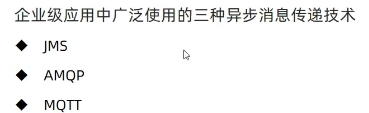
# 消息处理

消息发送方（生产者）

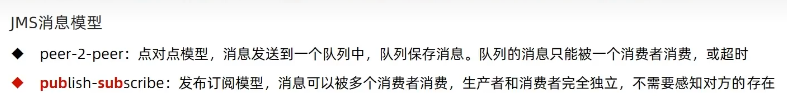
消息接收方（消费者）

消息分为同步消息和异步消息

消息被保存到MQ（消息队列）中，然后依次发送给子业务。



JMS，一个规范，等同JDBC规范，提供对应APi接口





AMQP，一种协议（高级消息队列协议，消息代理规范）规范了网络交换的数据格式，兼容JMS。

优点：具有跨平台，服务器，生产者，消费者可以使用不同的语言来实现。





MQTT，物联网主要成分，消息队列遥测传输。

Kafka，一种高吞吐的分布式发布订阅消息系统，提供实时消息功能。

ActiveMQ

RabbitMQ

RocketMQ

Kafka

# 监控

检测服务状态，运行指标（内存，虚拟机，线程，请求等）

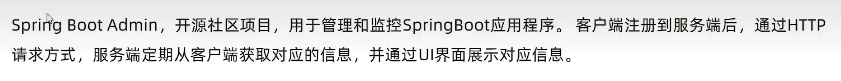
监控日志，管理服务（例如下线等）

监控实施的方式：

显示监控信息的服务器：用于获取服务信息，并显示对应的信息

运行的服务：启动时主动上报，告知监控服务自己需要收到监控

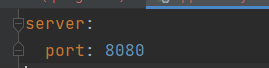
SpringBootAdmin



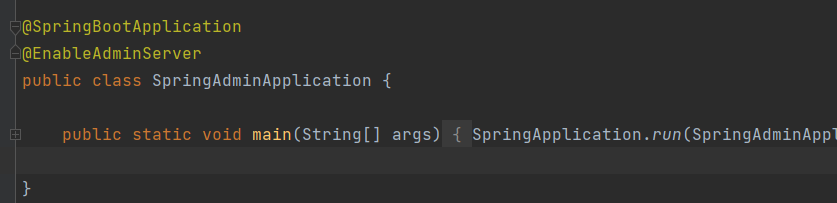
添加依赖



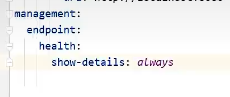
配置端口



在启动类上添加注解



客户端需要配置才能被服务端所监控



### 