1. 个人介绍

计算机科班出身，双非院校，搞了15年码农，大厂小厂单干都玩过，目前一家外资企业做独立架构师(外派菊厂架构，主要就是带小朋友上路，哈哈哈)，结合我多年的实际经验和小朋友们的愉快沟通，我决定出一个新手入门快速干项目的教程，当然大厂都有自己的架子，不可能来了让小盆友开发架子，做基础，这套视频的核心时让大家认识到大厂的开发基础脚手架都有哪些，咋么使用其实需要举一反三的自己慢慢领悟

1. Spring boot 与spring mvc 的区别

2.1 Spring boot核心思想:约定大于配置，功能模块化，可插拔，拿来即用

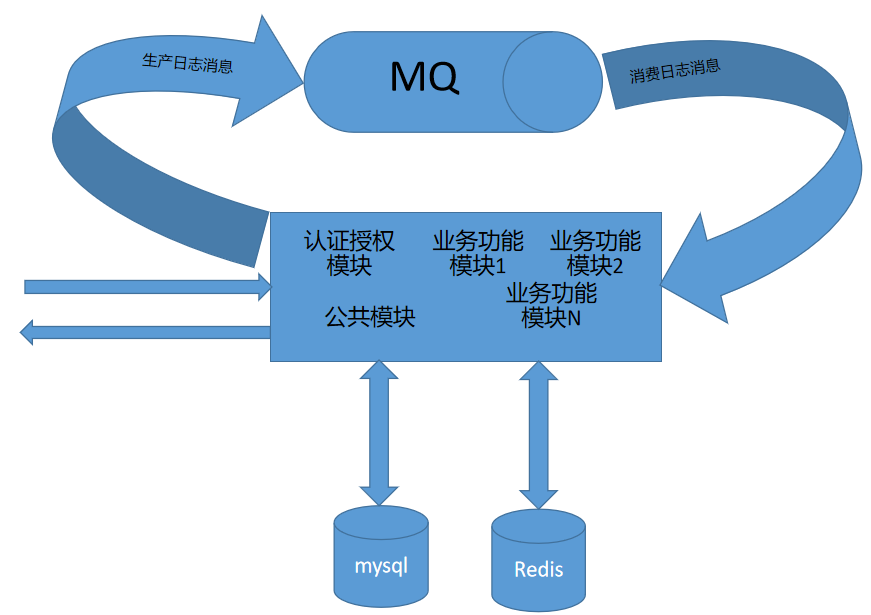
Spring mvc 能干的事spring boot 也能干(前提是要加入相关模块)，两者不能直接划等号

Spring mvc相当于是个功能全家桶的代称相当于中国里面包含了很多省份

Spring boot 只是个框子里面是空的，用到什么在再召唤，召唤方式就是POM依赖引入对应模块坐标，

同时依赖模块全部封装成相对独立的组件，通过约定的注解即可在项目中引入

下面我们将结合实际案例从J2EE开发标准四件套(认证与权限管理，mysql,MyBaties pus，redis,MQ)给大家安项目运作的方式逐一讲解



后续课程会按企业项目开发的模式应用搭建好的基础脚手架common模块开发业务功能模块

1. Spring boot 环境搭建

3.1 简单的login登录请求

创建父工程，父类工程中声明使用springboot 2.50，一般情况时是为了集中版本管理，防止子module单独引用jar包引起版本紊乱

<parent>

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

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

<version>2.5.0</version>

</parent>

<!-- 打包插件 -->

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

创建子模块，模块名称为common,继承父工程

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

@RestController

public class LoginControl {

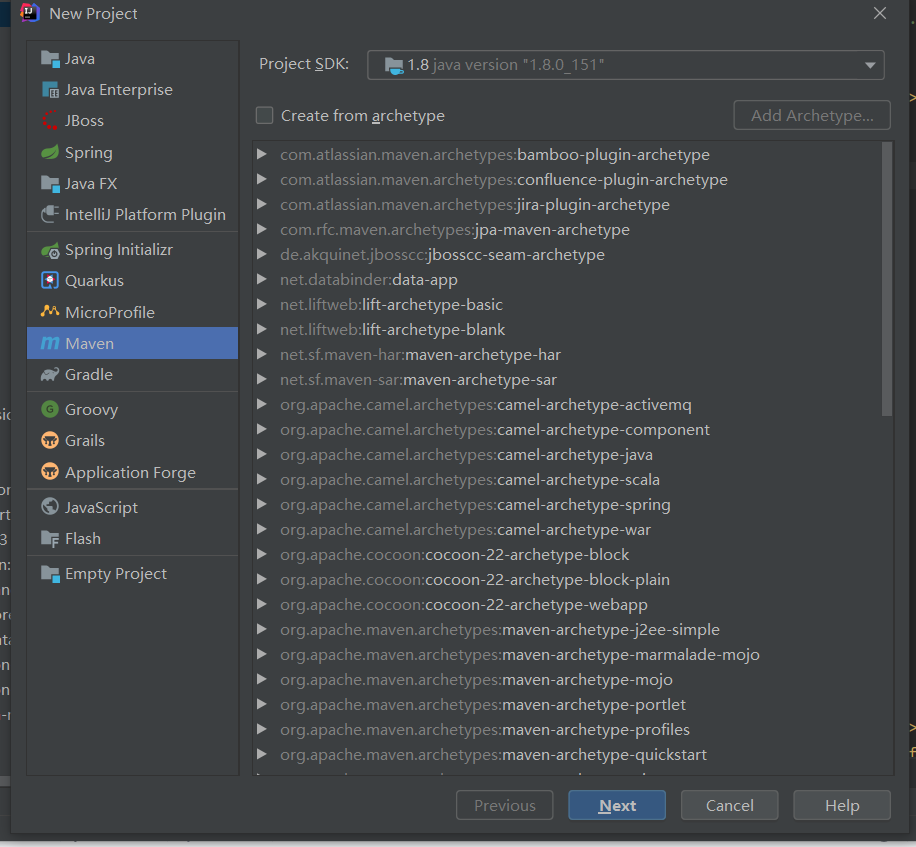
@GetMapping("/admin/login")

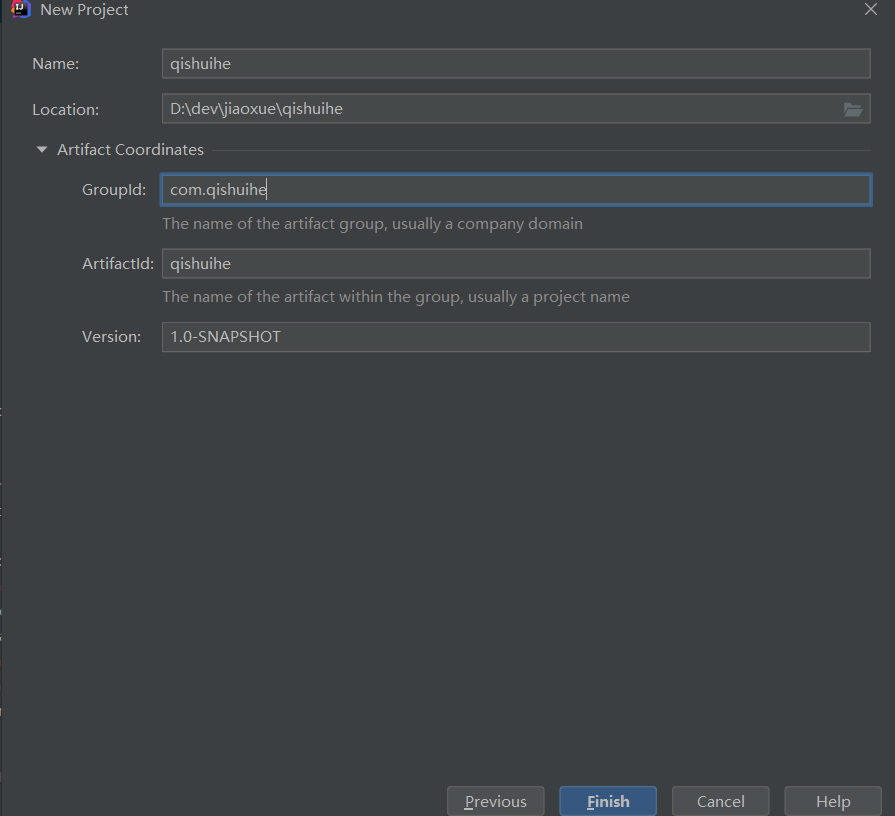
String login(@RequestParam String name,@RequestParam String password){

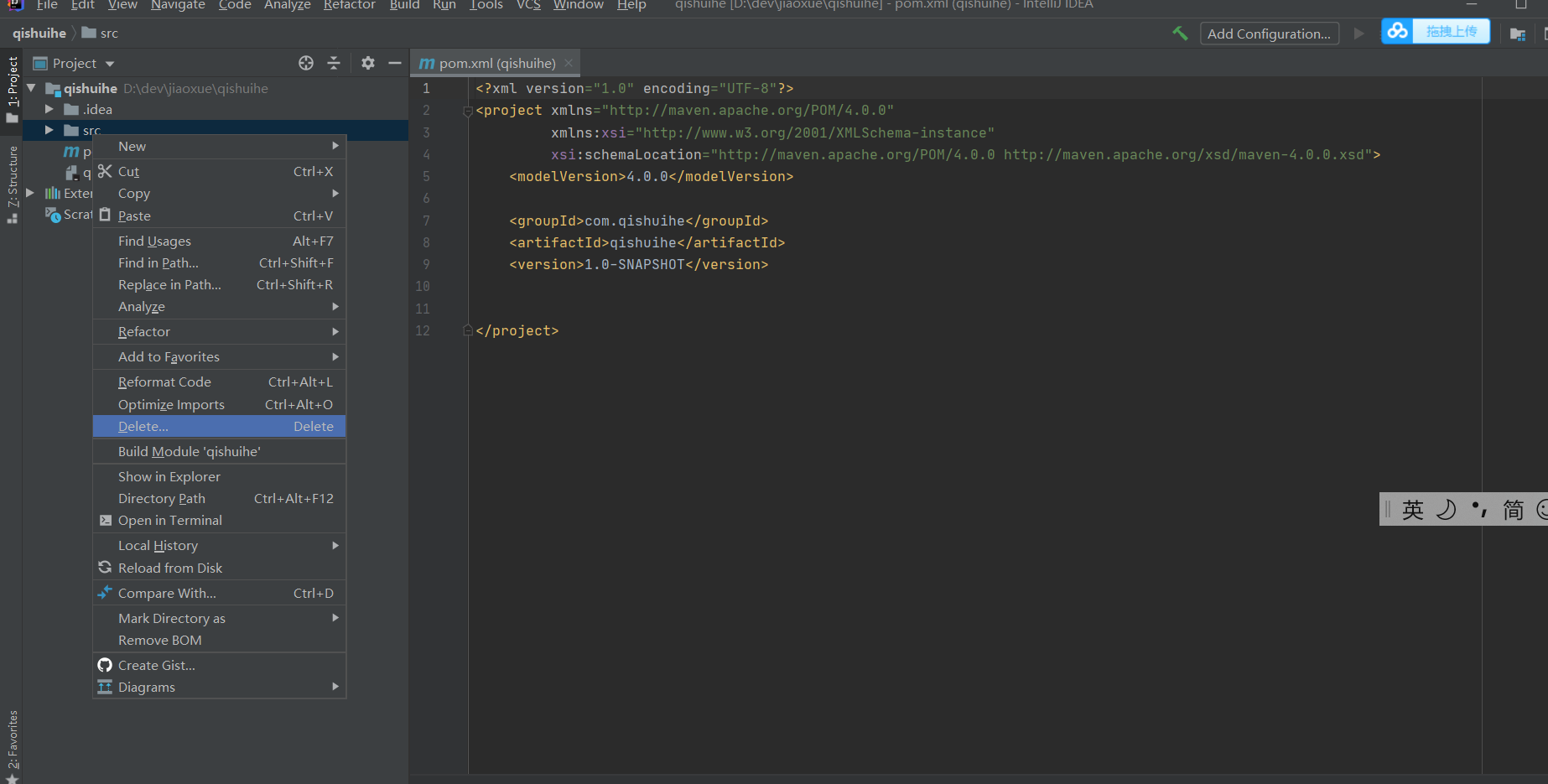
return name+password;

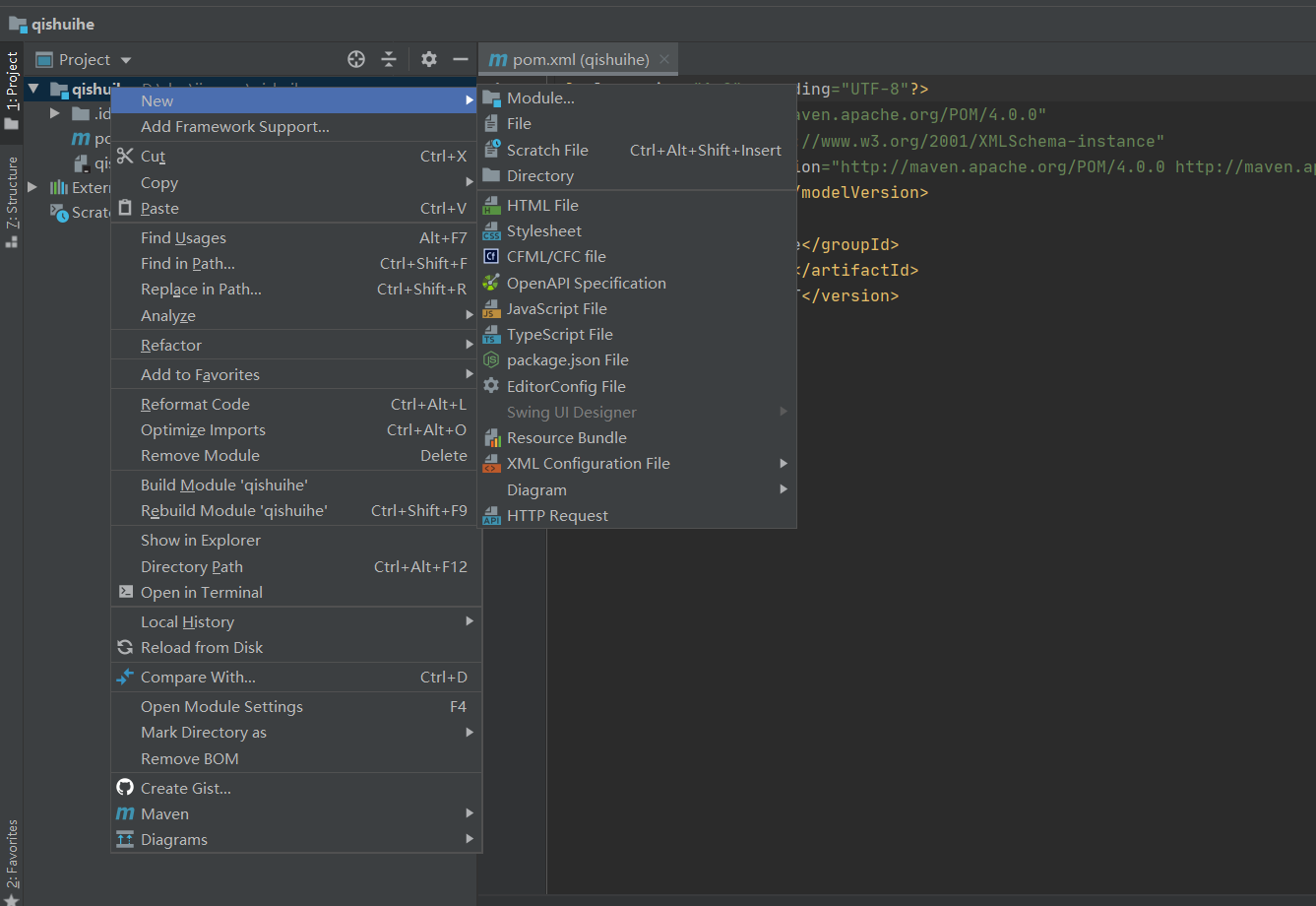
}

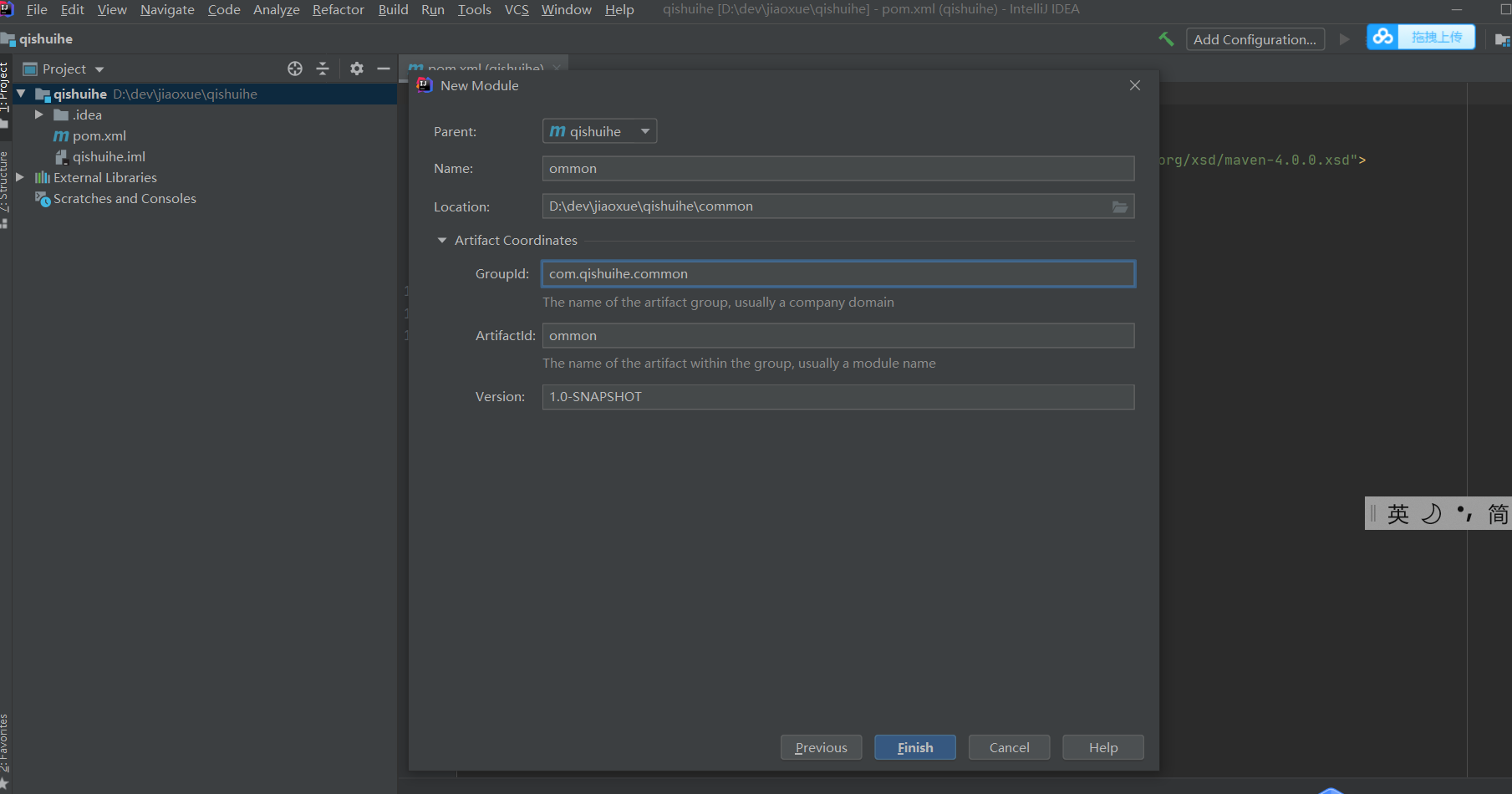
}

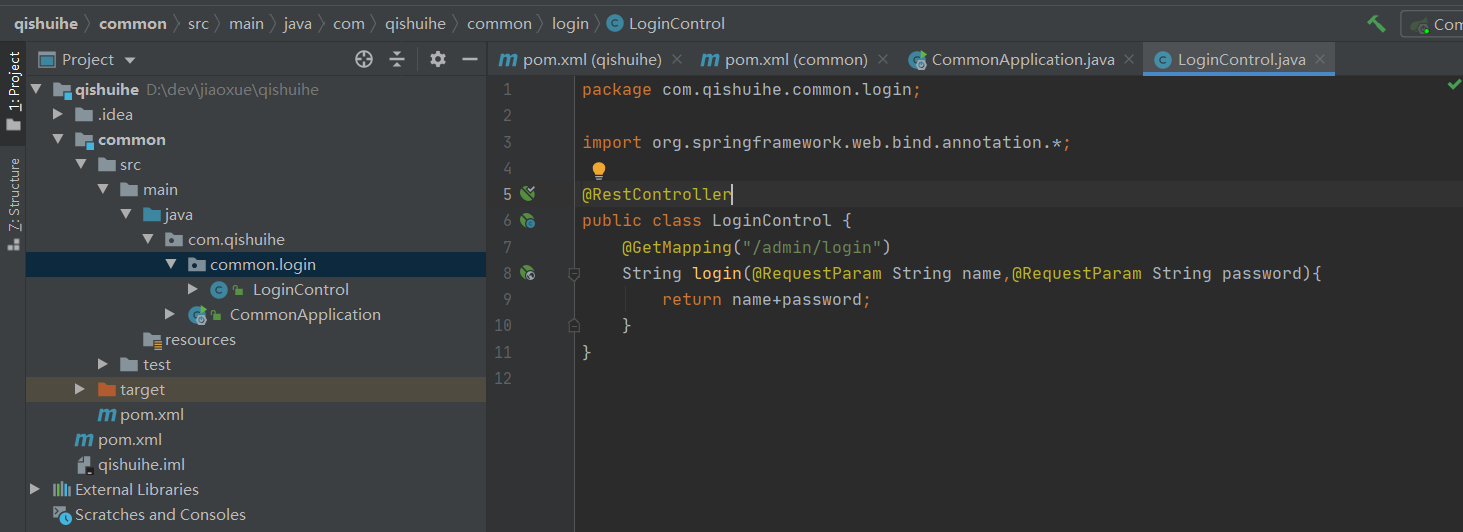












1. Spring boot 与swager集成

在现在企业开发过程中，前后端都是分离的，因此后端开发的接口需要有接口清单及接口参数展示的窗口，以便开发人员和测试人员了解使用

4.1 引入依赖

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.7.0</version>

</dependency>

4.2 接口类及方法上加对应的swagger注解

@Api("登录")

@RestController("/admin")

@Slf4j

public class LoginControl {

@PostMapping("/admin/login")

@ApiOperation(value="login",notes="登录")

@LogAnnotation

LoginResultDto login(@RequestBody User user){

log.debug("登录");

return loginServiceImpl.login(user);

}

4.3 启动类加上swagger注解，触发swagger集成

@EnableSwagger2

注入swagger实例对象bean

@Bean

public Docket webApiConfig(){

return new Docket(DocumentationType.SWAGGER\_2).groupName("webApi").apiInfo(webApiInfo()).select().paths(Predicates.not(PathSelectors.regex("/error\*"))).build().globalOperationParameters(getParameterList());

}

private ApiInfo webApiInfo(){

return new ApiInfoBuilder().title("漆水河API中心").description("spring boot 从零开始").version("1.0").build();

}

private List<Parameter> getParameterList(){

ParameterBuilder clientIdTickt = new ParameterBuilder();

List<Parameter> parameters = new ArrayList<Parameter>();

clientIdTickt.name(CommonConstant.TOKEN\_STR).description("请求令牌").modelRef(new ModelRef("String"))

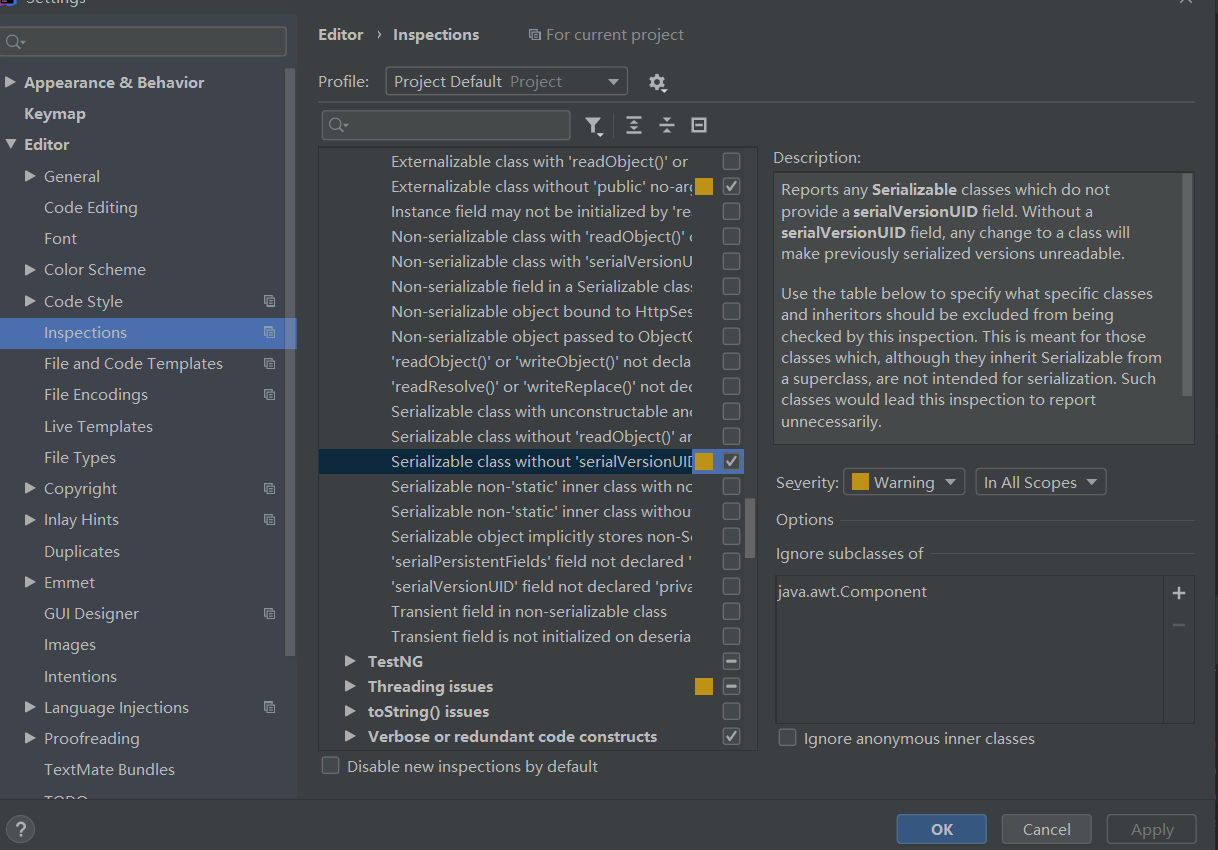
.parameterType("header").required(false).build();

parameters.add(clientIdTickt.build());

return parameters;

}

增加自动序列化



1. Spring boot 与spring security 集成

5.1 概念

Spring Security 是通过加入自定义过滤器的方式，对访问请求进行认证和鉴权

核心过滤器就是认证过滤器和鉴权过滤器

业务接口

核心组件:

...

...

FilterSecurityInterceptor

ExceptionTranslationFilter

UsernamePasswordAuthenticationFilter

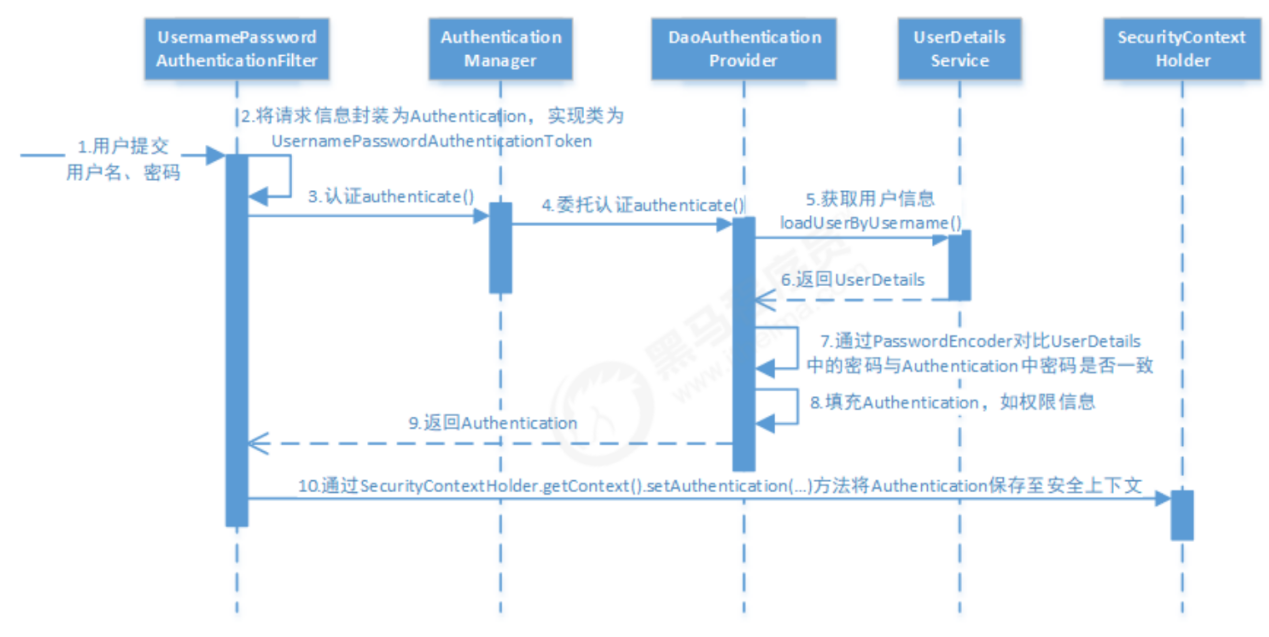
UsernamePasswordAuthenticationFilter 登录认证

(RememberMeAuthenticationFilter,BasicAuthenticationFilter,AnonymousAuthenticationFilter)

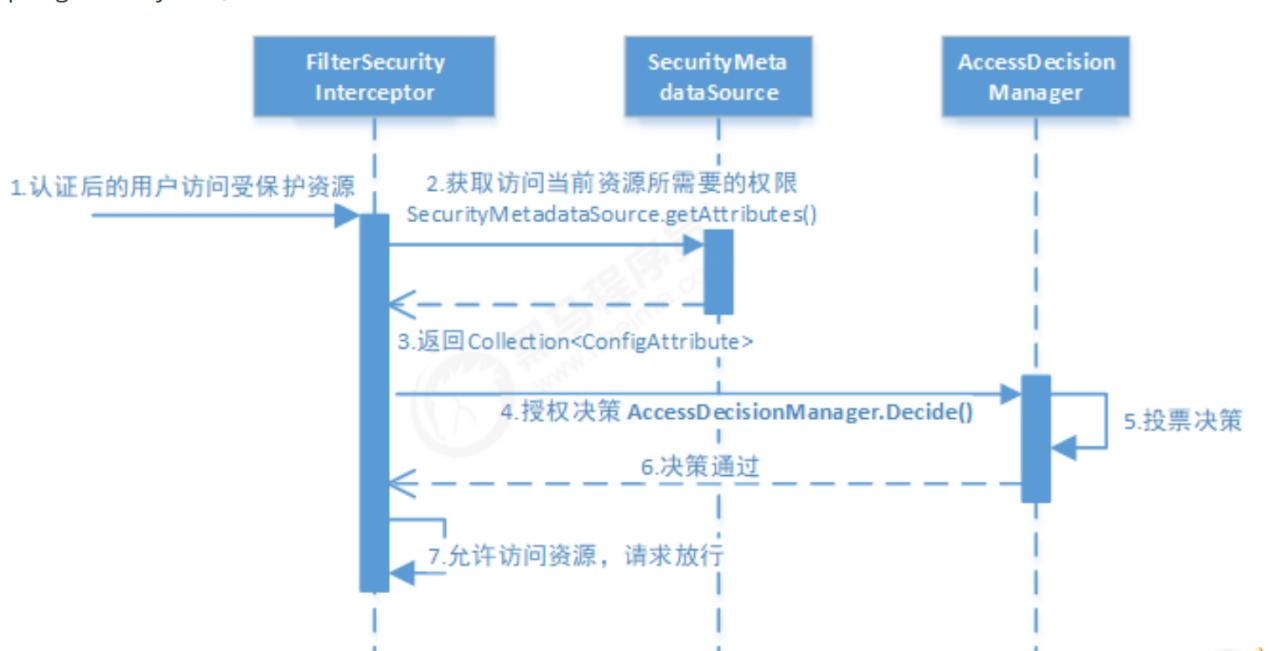
ExceptionTranslationFilter 异常处理

FilterSecurityInterceptor 鉴权拦截处理

认证的完成流程



鉴权投票过程



5.2 引入依赖

Spring 用户认证与权限

<dependency>

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

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

<scope>compile</scope>

</dependency>

JWT加解密工具类

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

Spring boot日志模块，方便我们调试程序

<dependency>

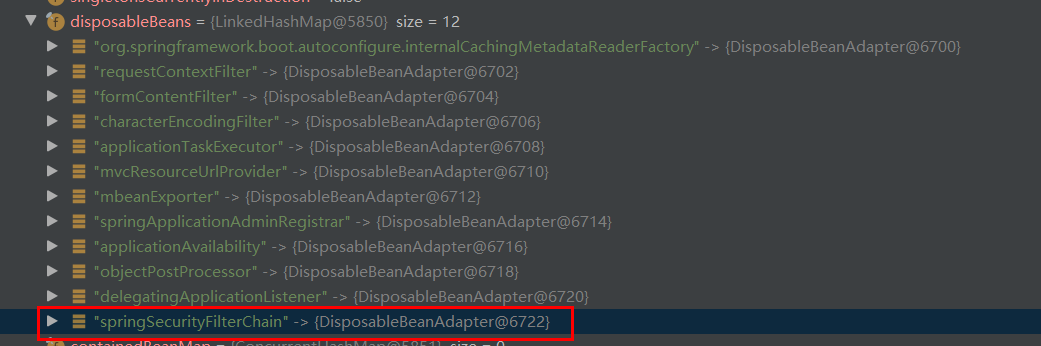
<groupId>org.springframework.boot</groupId>

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

</dependency>

5.2 配置spring security权限

安全过滤器已加入请求的链条



http.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and().authorizeRequests()

.antMatchers("/swagger\*\*","/v2/api-docs","/webjars/\*\*","/swagger-resources/\*\*","/favicon.ico","/admin/login").anonymous()

声明加密方式

@Bean

public PasswordEncoder getPasswordEncoder(){

return new BCryptPasswordEncoder();

}

认证管理器

@Bean

public AuthenticationManager getAuthenticationManager() throws Exception {

return super.authenticationManager();

}

重载用户自定义的认证鉴权信息

5.3 实现UserDetailsService接口

构造一个用户密码和用户访问权限

5.4 集成JWT,客户端token生成

创建JWTUtils，生成token并解析token，

5.5 配置基于token免登录过滤器JWTSecurityFilter

过滤继承OncePerRequestFilter，核心就是做token过滤解析，如果获取到token就解析token，token有效就把认证和授权过的信息获取到初始化到当前上下文，下一个认证过滤器就直接不再认证，如果过期下一个过滤器就需要重新认证

同时需要把JWT过滤器配置再用户名密码认证过滤器之前

https://spring.io/guides/topicals/spring-security-architecture/

1. **Spring boot 与mybatis plus+mysql 集成**

6.1 概念

Mysql8 ，schema，user，privileges

InnoDB是MySQL默认的存储引擎,utf8mb4,数据类型--auto\_increment。

mysql默认使用innodb，如果要用事务和外键就使用innodb，如果这张表只用来查询，可以用myisam。如果更新删除增加频繁就使用innodb

https://dev.mysql.com/downloads/workbench/

mybatis plus是对mybatis的功能增强开源项目，可以帮我自动生成基础层的代码，帮助我们尽可能少的写SQL代码，支持很多基础功能扩展，大家可以参考如下地址

https://baomidou.com/pages/779a6e/#%E4%BD%BF%E7%94%A8

<https://baomidou.com/>

我在这个项目中暂时只给大家用到了实体按照指定属性自动填充功能，后续或扩展删除功能，租户功能等特性

建用户，建仓库，给用户授权

建表

CREATE TABLE `sys\_user\_t` (

`id` bigint NOT NULL COMMENT '系统生成的唯一标识',

`username` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '用户姓名',

`password` varchar(200) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '登录密码',

`sex` int DEFAULT '0' COMMENT '0:男,1女',

`work\_id` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '工号，作为登录的唯一标识',

`email` varchar(45) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '邮件地址',

`create\_by` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '创建者',

`create\_date` datetime DEFAULT NULL,

`update\_by` varchar(45) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '最后更新人',

`update\_date` datetime DEFAULT NULL,

`tenant` varchar(45) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin DEFAULT NULL COMMENT '租户编码',

`enable` int DEFAULT NULL COMMENT '是否删除，0有效，1删除',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_bin COMMENT='系统用户表';

初始化一条数据

INSERT INTO `sys\_user\_t` VALUES (10001,'qishuihe',NULL,1,'10001','10001@qishuihe.com','10001','2022-08-15 22:15:00','10001','2022-08-15 22:16:00','10001',0);

6.2引入模块依赖

<dependency>

<groupId>com.baomidou</groupId>

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

<version>3.4.3</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

通过代码生成器生成对应的领域曾(第一次使用建议自己手动敲完整代码，这样有助于自己对mybatis plus代码继承关系的理解)

1. 实体对象与数据库映射
2. 继承mybatis的SQL自定义mapper模块
3. 通过扩展重构service泛型对象，继承对领域对象的增删改查

数据库链接配置

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://localhost:3306/qishuihe?useUnicode=true&useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai

username: qishuihe

password: qishuihe@123456

自定义SQL查询

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.qishuihe.common.domain.user.service.mapper.UserMapper">

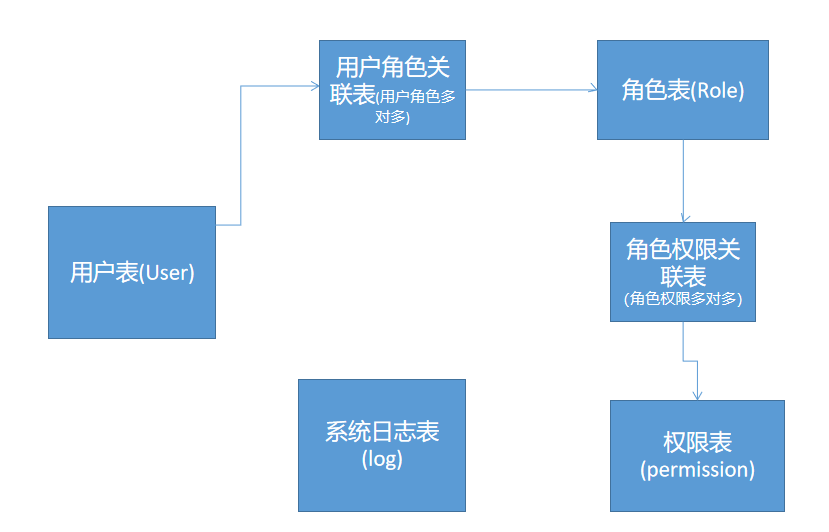
<select id="findUserByName" resultType="com.qishuihe.common.domain.user.entity.UserEntity">

select \* from sys\_user\_t where username = #{username}

</select>

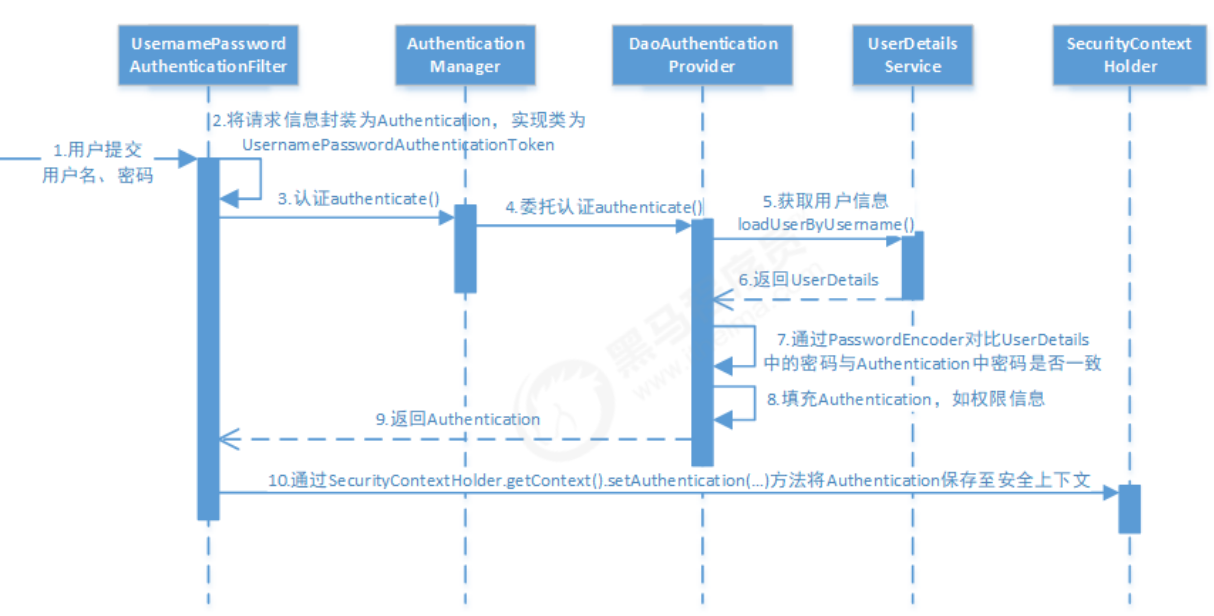
</mapper>

6.3 认证权限，日志记录数据库设计



6.4 在UserDetailsService接口实现类中重构loadUserByUsername方法

将用户的认证信息和权限信息装载进LoginUser



1. Spring boot 与redis集成

7.1 概念

Redis就是一种缓存数据库，查询性能非常好，但对事物的完整性支持比较弱,用redis的缓存特性来减轻数据库查询的压力，让数据库腾出双手来处理事务完整性，具体优缺点自己问度娘

7.2 Redis服务端安装

下载地址:https://github.com/tporadowski/redis/releases

具体安装参考:https://www.runoob.com/redis/redis-install.html

我这里使用的是windows版本

redis 下载安装好以后如下修改可以在关闭服务时能够及时存储数据  
1. 修改配置 ： redis.windows.conf 文件  
找到 appendonly no 改为 appendonly yes 允许redis中的数据罗盘  
找到 appendfsync 设置为 appendfsync everysec

执行redis-server --service-install redis.windows-service.conf --loglevel verbose

使redis服务在widows环境中成为幕后的守护进程运行，我们关闭当前启动窗口不会闭关redis server服务

用客户端连接服务端

进入目录

CD D:\qishuihe\Redis-x64-5.0.14.1

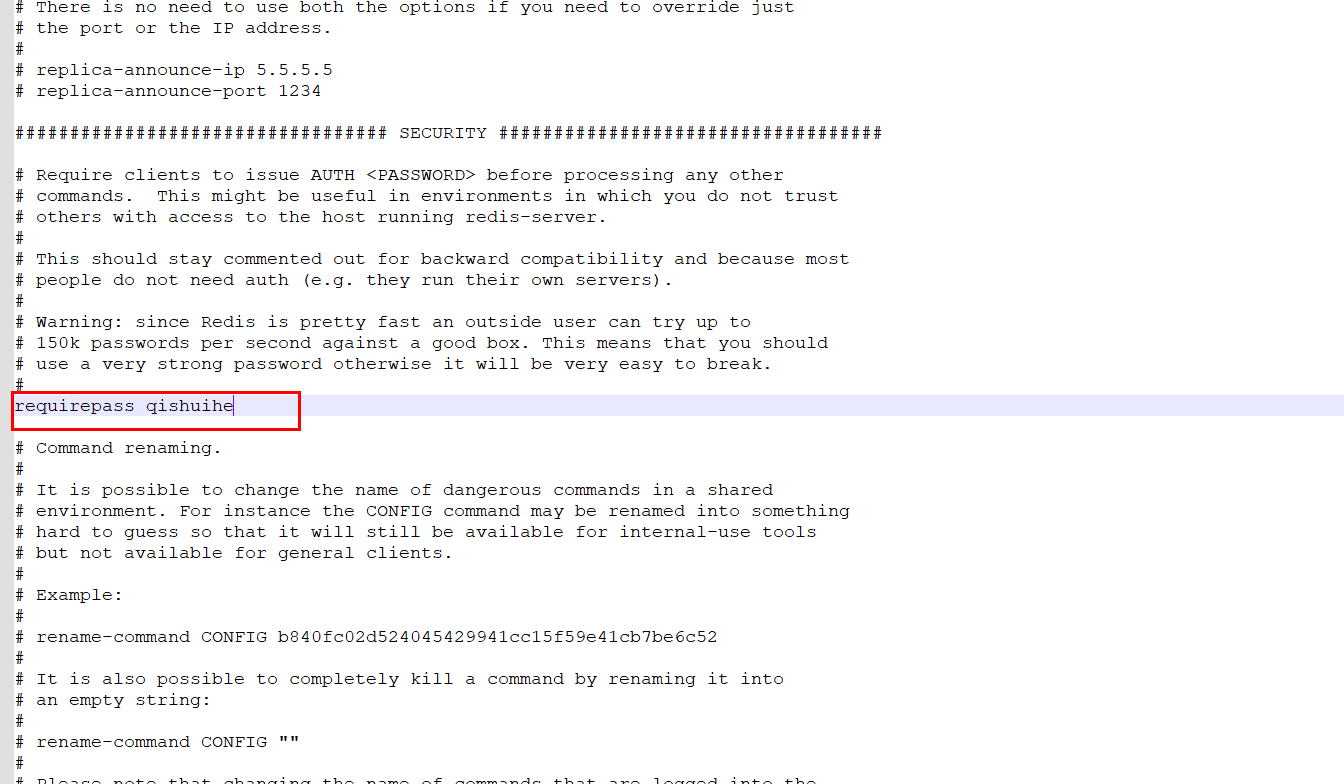
客户端连接密令设置密码

redis-cli -h 127.0.0.1 -p 6379

命令设置密码

CONFIG set requirepass "qishuihe"

配置文件设置密码



7.3 引入redis模块

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

</dependency>

6.4 构造redis配置参数

创建连接池配置RedisPoolPropertiesConfig，使用Jedis初始化连接池RedisPoolConfig，

1. Spring boot 与MQ集成

8.1 概念

MQ是为了降低请求链路的非必要等待时间，比如一个请求了来了我们需要记录日志(谁的请求，请求是干什么，什么时候，这些记录信息对业务方是非必要的事情，对系统事后安全审计又是必要的信息，我们就可以采用异步解耦的方式处理)，这是MQ就显得特别友好，既降低请求链路的非必要时间损耗又能提升大规模日志排队记录的性能。

8.2模块引入

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

</dependency>

8.3 安装服务端kafka

cd D:\qishuihe\kafka\_2.12-3.2.0\bin\windows

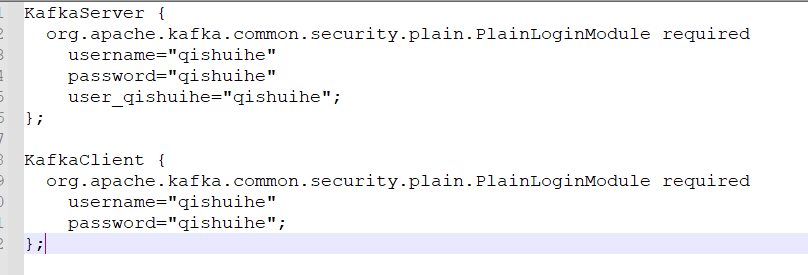
命令行切到kafka配置的windows目录下，启动自带的zookeeper

zookeeper-server-start.bat ../../config/zookeeper.properties

配置kafka-server-start.bat，将用户名密码配置文件加载到起动参数中



Config目录下创建kafka\_server\_jaas.conf文件



编辑Config目录下server.properties增加如下配置

listeners=PLAINTEXT://127.0.0.1:9092

log.dirs=/app/kafka/logs

num.partitions=3

zookeeper.connect=192.168.6.61:2181

##########

listeners=SASL\_PLAINTEXT://0.0.0.0:9092

advertised.listeners=SASL\_PLAINTEXT://127.0.0.1:9092

security.inter.broker.protocol=SASL\_PLAINTEXT

sasl.enabled.mechanisms=PLAIN

sasl.mechanism.inter.broker.protocol=PLAIN

#authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer

authorizer.class.name=kafka.security.authorizer.AclAuthorizer

#修改验证机制

allow.everyone.if.no.acl.found=true

启动kafka

cd D:\qishuihe\kafka\_2.12-3.2.0\bin\windows

kafka-server-start.bat ../../config/server.properties

8.4 构造MQ配置参数

kafka:

bootstrap-servers: 127.0.0.1:9092

template:

default-topic: qishuihe

producer:

key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

consumer:

group-id: qishuiheGroup

enable-auto-commit: true

auto-commit-interval: 1000

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

properties:

security:

protocol: SASL\_PLAINTEXT

sasl:

mechanism: PLAIN

jaas:

config: 'org.apache.kafka.common.security.scram.ScramLoginModule required username="qishuihe" password="qishuihe";'

1. Spring boot 与使用AOP

9.1 概念

AOP为Aspect Oriented Programming 面向切面编程，一般我们都称为拦截器，用于处理基于特定方法的拦截处理，分为，事前，事后，环绕拦截处理。简单理解为横向拦截

Spring 自己定义的框架拦截器HandlerInterceptor

与过滤器，监听器的区别，spring 框架定义的拦截器

我们前面介绍的spring security 就主要用的是过滤器(filter),通过过滤器链条层层对请求过滤拦截处理，可以简单理解为纵向拦截处理

监听器就是通过不断的心跳发送消息，获取指定点位的请求，属于精准定位定时交互，简单理解就是点对点精准交互

所有拦截器，过滤器，监听器是在不同维度上对请求的业务做特定逻辑处理

