X您好，我叫刘金鑫，老家是山西临汾的，15年大学毕业，毕业之后就一直从事着咋们这个java行业，也做了一些项目，包括电商，OA，挂号，还有近期的保险。

公司里面

个险在职单督导关系建立：



首先，业务概述：就是为了合理的分配保单，建立对应的督导关系。业务流程：1保单分配，也就是把保险单分配给满足条件的专员，先选择机构，选择类型和日期，接着分配给建立过督导关系的续费专员 2督导关系查询，根据不同的条件，查出机构的各个层级选中相关查询记录，录入、复制粘贴续期专业编码或双击选择出专员编码，点击督导关系建立，将个险层级和续期专员建立起督导关系。3选择查询出的记录，若该记录的机构下的个险层级（区部组）已经建立督导关系，点击督导关系删除，删除已经建立的督导关系。4如果营业区已经和专员A建立督导关系，则默认营业区下面的部、组都和专员A建立了督导关系。5如果营业区和营业区下的营业部或营业组对应的专员不一致，则营业区的专员覆盖掉营业部和营业组的专 员，显示营业区对应的专员。6如果营业部对应专员A，则该营业部之下的营业组和专员A建立督导关系。7如果营业部和营业部下的营业组对应的专员不一致，则营业部的专员会覆盖掉营业组的专员。8个险的业务层级为区部组，组是最小单位，各个小组建立的督导关系，才能覆盖所有的个险在职保单。9如果专员A已经和营业区/建立督导关系，若选择该营业区下的部或组，和专业B建立督导关系，也就是上下级对应不一致，起了冲突，系统需要提示：某某区或某某部已经和专员A建立督导关系，不支持某某部或某某组在和B建立督导关系。保单分配到已建立督导关系的续期专员。如果营业区甲这个层级和专员A建立督导关系，则所辖的部和组都建立了督导关系，选择营业部或营业组督导关系查询的时候，显示对应的专员为A。录入管理机构、督导关系状态、层级等查询条件，点击【督导关系查询】，查询出机构相应的个险业务层级，专员编码列下拉出管理机构下的专员编码，点击【督导关系建立】时，需要校验该列是否为空、是否有效、是否和原服务人员在同一中支机构，如果为空，提示“专员编码不能为空！”，如果专员编码不为在职或二次入司，则提示 “专员编码+不是有效的续期专员”，如果专员编码和原服务人员编码不在一个中支公司，则提示“要分配的续期专员与原服务人员不在同一中心支公司！”，选中一条或多条查询结果点击【督导关系建立】，将该个险业务层级和续期专员建立起督导关系，如果该个险业务组已建立督导关系，分配给新的专员后，原督导关系结束，支持多选和全选。如果全选时部分个险业务组专员信息为空，代表此次该业务组不参与分配，保存时跳过该记录。这些条件其实就是这个模块的一些逻辑，就是非常繁琐，并且需要理解层级和督导还有专员还有在职单这些关键点之间的关系，而且在写代码的时候，这些条件都满足成功后，会提示：机构下的个险业务区或部或组和续期专员的督导关系建立；

银行网点服务关系建立：

1录入管理机构等查询条件，点击分配，将满足条件的保单分配给建立过服务关系的续费专员。2录入管理机构、服务状态、应收年月等查询条件，点击查询，查询出机构下应收年月保单的银行网点信息，银行网点为抽档成功，也就是调取成功的保单所在的银行网点，选中一条或者多条或者全部记录，录入续费专员编码，点击建立关系，将银行网点和续期专员建立起督导关系。未录入专员的编码不处理。3前期建立过服务关系的网点，若在应收年月还有该网点的应收保单，则默认该网点为以前建立服务关系的专员（专员在职）；打个比方：网点甲，在2018年3月对应专员A，如果在2018年5月份该网点有应收，且专员A还在职，在服务关系查询时，默认已经建立服务，对应专员A。4专员已经离职，默认已经建立的服务（银保网点）/督导（个险在职保单）关系为空。**备注：不管个险还是银保的在职保单，保单已经分配给续期专员（督导位置处），销售人员若离职，督导的专员还在职的话，自动实现在职单变为孤儿保单的分配，现服务人员直接变更为原来的督导专员，实现在职单变为孤儿保单的自动分配。**

保单导入导出：@RequestMapping("exp")

public void exp(HttpServletResponse response) throws IOException{

//实例化

T\_user uu = new T\_user();

//规定导入(Excel从0格开始,到XXX结束)

uu.setStartPos(0);

uu.setEndPos(9999);

//查询完list结果

List<T\_user> list = us.ulist(uu);

//创建Excel表

HSSFWorkbook wb = new HSSFWorkbook();

//根据Excel创建页面

HSSFSheet sheet = wb.createSheet("物资管理");

//规定行数,从0行开始

HSSFRow createRow = sheet.createRow(0);

//第0行的第一个数据规定为"编号",之后依次往后推,直到这行结束

createRow.createCell(0).setCellValue("编号");

createRow.createCell(1).setCellValue("姓名");

createRow.createCell(2).setCellValue("年龄");

createRow.createCell(3).setCellValue("部门");

//用for循环将代码(list集合)里的数据依次传入Excel单元格

for (int i = 0; i< list.size(); i++) {

//由于我们表(Excel)中,只能由对象去赋值,并且是根据下标,所以我们将其转换,根据下标(i)进行转换

T\_user user = (T\_user)list.get(i);

//这个也是行数,只不过0行已经被占用,也就是我们的信息描述(例:姓名)占用了第0行,所以我们从i+1开始填充数据

HSSFRow row = sheet.createRow(i+1);

//这个是编号

row.createCell(0).setCellValue(i+1);

//这个是信息

row.createCell(1).setCellValue(user.getId());

row.createCell(2).setCellValue(user.getUname());

row.createCell(3).setCellValue(user.getUage());

row.createCell(4).setCellValue(user.getDept().getDname());

//我们循环完一条数据,就另起一行再次添加,所以i+1

sheet.autoSizeColumn(i+1,true);

}

//io可以理解为字节流,而字节流里面的file是自己定义的路径,根据自己电脑来设置

FileOutputStream io = new FileOutputStream(new File("D:\\物资管理.xls"));

wb.write(io);

io.close();

}

}

导出用的是xls方式，用的是手动导出，并不是网页下载，但是在这个过程中，有两个问题，第一个问题是效率过慢，第二个问题就是在大量数据的时候会内存溢出，内存溢出有两个方面，第一种是查询数据量过大，造成内存溢出，这个可以用分批查询来解决，第二种是下载时转化数据流的时候造成内存溢出，这个可以用sxssfworkbook来解决

首先注册并且登录到我们的阿里大于的平台，选择短信服务，添加属于自己的签名，申请完事之后等待咋们的审批，在提交的时候能选择是个人还是企业，然后申请模板，还有一个属于自己的keyid。这些准备流程弄完之后，下载api，导入依赖，调用接口，下发短信，必填的信息有手机号，签名，模板，如何读取配置的？用Environment中的get.property获取，用setproperty(defaultConnectionTimeout)和setproperty(.defaultReadTimeout)设置验证码过期时间。之后用rabbitMQ的convertAndSendhandler方法存入mq，handler和listener去执行与监听

验证码如何生成的呢？先设置一个max数值，然后再设置一个min，random生成的随机数用max作为上限，如果小于最小值，就让生成的数+最小值，就是规定的长度了。

如何保证用户刷新或者多次点击反复发送呢？

Bcrypt加密，首先要导入springsecurity依赖，我们目 前只是需要用到BCrypt密码加密的部分，所以我们要添加一个配置类，配置为所有地址 都可以匿名访问。配置类先有@EnableWebSecurity注解，此类继承WebSecurityConfigurerAdapter。.authorizeRequests() 请求权限                .antMatchers("/\*\*").permitAll()    路径             .anyRequest().authenticated() 认证过的任何请求                .and().csrf().disable(); 跨域请求他默认是全部攻击自己的，然后拦截所有跨域请求。实现密码加密是注入BcryptPasswordEncoder这个类，这个类里面用两个方法，一个是encode加密，一个是mathches密码对比，加密是在注册的时候用的，对比是在登陆时候用的。

然后在登陆时候，选择的是基于JWT的Token认证机制实现，认证流程：1. 客户端使用用户名跟密码请求登录 2. 服务端收到请求，去验证用户名与密码 3. 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端 4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里 5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token 6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向 客户端返回请求的数据。

创建token：导入jjwt依赖，设置签发时间，设置密钥，每次的token都是不一样的，因为载荷当中存在了时间，

Token拦截授权：public void addInterceptors(InterceptorRegistry registry) {      registry.addInterceptor(jwtFilter).          addPathPatterns("/\*\*").                  excludePathPatterns("/\*\*/login")。登陆的

登陆流程：先验证身份，查询用户名，根据用户信息去查询密码，之后再通过加密的密码和查询的密码做对比，如果通过了，那么就生成token，

那为什么用token不用cookie？

Token机制相对于Cookie机制又有什么好处呢？ 支持跨域访问: Cookie是不允许垮域访问的，这一点对Token机制是不存在的，前提 是传输的用户认证信息通过HTTP头传输. 无状态(也称：服务端可扩展行):Token机制在服务端不需要存储session信息，因为 Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储 状态信息. 更适用CDN: 可以通过内容分发网络请求你服务端的所有资料（如：javascript， HTML,图片等），而你的服务端只要提供API即可. 去耦: 不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在 你的API被调用的时候，你可以进行Token生成调用即可. 更适用于移动应用: 当你的客户端是一个原生平台（iOS, Android，Windows 8等） 时，Cookie是不被支持的（你需要通过Cookie容器进行处理），这时采用Token认 证机制就会简单得多。 CSRF:因为不再依赖于Cookie，所以你就不需要考虑对CSRF（跨站请求伪造）的防 范。 性能: 一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256 计算 的Token验证和解析要费时得多. 不需要为登录页面做特殊处理: 如果你使用Protractor 做功能测试的时候，不再需要 为登录页面做特殊处理.

配置zuul：首先创建一个服务，然后让他被eureka管理，导入两个依赖，一个eureka一个zuul，接着配置eureka服务，因为砸门要让erka管理，eruka：client：service-url：zone（默认地址），接着配置虚拟网关：zuul：routes（权限）：然后配置哪个服务进行访问，也就是砸门在yml文件里面的服务name，接着配置规则（path），这个规则相当于怎么通过zuul来访问项目，在项目里是用/自己的名字/\*\*来规则的，规则配置完毕之后，要让zuul知道去访问谁，然后配置一个serviceID zuul启动类配置 @enablezuul。

Zuul过滤器配置：继承zuulfilter

阿里大于调用过程。

Zuul虚拟网关介绍。Bcrypt加密算法怎么用的？在哪里用的？token是怎么形成的？为啥用jwt？