***查询“阿里编程规范 （如：阿里巴巴JAVA开发手册）；华为编程军规”等，对照自己的代码看有哪些不符合规范的地方，修改。***

这里简要分析两种编程规范的要点以及可取之处,代码修改不在此列出.

### 华为编程军规:

https://zhuanlan.zhihu.com/p/158219110

从代码规范和项目架构方面，主要总结了以下几点:

1. 规范的命名：军规要求开发人员使用清晰、准确且一致的命名方式。命名应该具有描述性，并遵循公司的命名约定。
2. 减少复杂性：军规强调尽量避免复杂的代码逻辑和嵌套结构。开发人员应该尽可能简化代码，并注重可读性和可维护性。
3. 注重代码重用：军规鼓励开发人员提高代码重用率。可以通过抽象和封装常用功能，创建可重用的模块和组件。
4. 高内聚低耦合：军规要求模块内部的代码功能高度聚合，模块之间的耦合度要尽可能低。这样可以提高代码的可维护性和可测试性。
5. 异常处理：军规要求开发人员在代码中合理处理异常情况，并进行适当的日志记录。异常处理应该考虑到代码的稳定性和可靠性。
6. 版本控制和代码管理：军规强调使用版本控制工具（如Git）进行代码管理，合理使用分支和标签，确保代码的可追溯性和版本控制。
7. 团队协作和交流：军规鼓励团队成员之间的良好沟通和协作。开发人员应该定期进行代码审查、团队讨论和知识共享，以提高代码质量和团队效率。
8. 安全和隐私保护：军规要求开发人员充分考虑软件安全和用户隐私，遵循相关的法律法规和最佳实践。

对于本项目代码的修改意见:

1. 命名简洁易懂,以下划线连接单词等形式形成简单变量名,检查变量、函数和类名是否具有描述性。
2. 代码简化：查找复杂的逻辑和嵌套结构，提出简化代码的建议，例如拆分长函数为多个独立的小函数，简化条件判断等。
3. 检查代码是否存在重复的逻辑，提出创建可重用模块或组件的建议，以减少冗余代码。
4. 分析模块内部的代码功能是否高度聚合，提出合并或拆分模块的建议，以及减少模块之间的耦合度。
5. 检查异常处理的方式和位置，提出改进异常处理的建议，以增强代码的稳定性和可靠性。
6. 评估项目的版本控制策略和使用，提出合理使用分支和标签的建议，以及增强版本追踪和管理能力。
7. 检查代码注释和文档的准确性和完整性，提出补充和改进注释和文档的建议。
8. 检查代码风格是否一致，例如缩进、空格、括号的使用等。
9. 审查代码中的安全和隐私问题，提出加强安全性和隐私保护的建议，例如数据加密、权限验证等。

### 阿里编程规范：

https://developer.aliyun.com/article/850913

根据网络上资料的学习，将阿里编程规范中的代码的规范总结为了以下几个点：

1. 命名风格

* 代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。
* 严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。
* 代码和注释中避免使用任何语言的种族歧视性词语。
* 类名使用UpperCamelCase风格，但以下情形除外：DO/BO/DTO/VO/AO/PO/UID等
* 方法名、参数名、成员变量、局部变量都统一使用lowerCamelCase风格。
* 常量命名全部大写，单词间用下划线隔开，力求予以表达完整清楚，不要嫌名字长。
* 抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾；测试类命名以它要测试的类的名称开始，以 Test 结尾
* 类型与中括号紧挨相连来表示数组。
* POJO 类中的任何布尔类型的变量，都不要加 is 前缀，否则部分框架解析会引起序列 化错误。
* 包名统一使用小写，点分隔符之间有且仅有个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式
* 避免在子父类的成员变量之间、或者不同代码块的局部变量之间采用完全相同的命名，使可理解性降低
* 杜绝完全不规范的缩写，避免望文不知义。

1. 常量定义

* 不允许使用未经预先定义的常量
* 在 long 或者 Long 赋值时，数值后使用大写字母 L，不能是小写字母 l，小写容易跟数字混淆，造成误解

1. 代码格式

* 如果是大括号内为空，则简洁地写成{}即可，大括号中间无需换行和空格；如果是非空代码块则： 1）左大括号前不换行。 2）左大括号后换行。 3）右大括号前换行 4）右大括号后还有 else 等代码则不换行；表示终止的右大括号后必需换行。
* 左小括号和右边相邻字符之间不出现空格；右小括号和左边相邻字符之间也不出现空 格；而左大括号前需要加空格。
* if/for/while/switch/do 等保留字与括号之间都必需加空格。
* 任何二目、三目运算符的左右两边都需要加一个空格。
* 说明：包括赋值运算符号=、逻辑运算符&&、加减乘除符号等
* 采用 4 个空格缩进，禁止使用 Tab 字符。说明：如果使用 Tab 缩进，必需设置 1 个 Tab 为 4 个空格。IDEA 设置 Tab 为 4 个空格时，请勿勾选 Use tab character；而在 Eclipse 中，必需勾选 insert spaces for tabs
* 注释的双斜线与注释内容之间有且仅有一个空格。
* 在进行类型强制转换时，右括号与强制转换值之间不需要任何空格隔开
* 单行字符数限制不超过 120 个，超出需要换行，换行时遵循如下原则： 1）第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进。2）运算符与下文一起换行。 3）方法调用的点符号与下文一起换行。 4）方法调用中的多个参数需要换行时，在逗号后进行。 5）在括号前不要换行
* 方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。
* 方法参数在定义和传入时，多个参数逗号后必需加空格
* IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式，不要使用 Windows 格式.

1. 注释规约

* 类、类属性、类方法的注释必须使用 Javadoc 规范，使用/\*内容/格式，不得使用 //xxx 方式。
* 所有的抽象方法(包括接口中的方法)必须要用 Javadoc 注释、除了返回值、参数、 异常说明外，还必须指出该方法做什么事情，实现什么功能。对子类的实现要求，或者调用注意事项，需要一并说明。
* 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/\* \*/注释，注意与代码对齐。
* 所有的枚举类型字段必须要有注释，说明每个数据项的用途。
* 代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。
* 注释掉的代码尽量要配合说明，而不是简单的注释掉。说明:代码被注释掉有两种可能性：
* 1）后续会恢复此段代码逻辑。如果没有备注信息，难以知晓注释动机。
* 2）永久不用。建议直接删掉(代码仓库保存了历史代码)。
* 特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描， 经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。
* 1）待办事宜(TODO)：( 标记人，标记时间，[预计处理时间]) 表示需要实现，但目前还未实现的功能。这实际上是一个 Javadoc 的标签，目前的 Javadoc还没有实现，但已经被广泛使用。只能应用于类，接口和方法(因为它是一个 Javadoc 标签)。
* 2）错误，不能工作(FIXME)：(标记人，标记时间，[预计处理时间])在注释中用 FIXME 标记某代码是错误的，而且不能工作，需要及时纠正的情况。

1. OOP规约

* 避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。
* 所有的覆写方法，必须加@Override注解。
* 说明：getObject()与get0bject()的问题。一个是字母的O，一个是数字的0，加@Override 可以准确判断是否覆盖成功。另外，如果在抽象类中对方法签名进行修改，其实现类会马上编译报错。
* 相同参数类型，相同业务含义，才可以使用Java的可变参数，避免使用Object.。
* 说明:可变参数必须放置在参数列表的最后。(建议开发者尽量不用可变参数编程)
* 外部正在调用或者二方库依赖的接口，不允许修改方法签名，避免对接口调用方产生影响。接口过时必须加@Deprecated注解，并清晰地说明采用的新接口或者新服务是什么。
* 不能使用过时的类或方法。
* object的equals方法容易抛空指针异常应使用常量或确定有值的对象来调用equals。
* 说明︰推荐使用JDK7引入的工具类java.util.Objectst#equals(Object a, Object b)
* 所有整型包装类对象之间值的比较，全部使用equals方法比较。
* 任何货币金额，均以最小货币单位且整型类型来进行存储。
* 浮点数之间的等值判断,基本数据类型不能用==来比较,包装数据类型不能用equals来判断。
* 说明:浮点数采用“尾数+阶码”的编码方式，类似于科学计数法的“有效数字+指数”的表示方式。二进制无法精确表示大部分的十进制小数，具体原理参考《码出高效》。
* 如上所示BigDecimal的等值比较应使用compareTo()方法，而不是equals()方法。
* 说明:equals()方法会比较值和精度(1.0与1.00返回结果为false )，而compareTo()则会忽略精度。
* 定义数据对象DO类时，属性类型要与数据库字段类型相匹配。
* 禁止使用构造方法BigDecimal(double)的方式把double值转化为BigDecimal对象。
* 说明:BigDecimal(double)存在精度损失风险，在精确计算或值比较的场景中可能会导致业务逻辑异常。
* 关于基本数据类型与包装数据类型的使用标准如下:
* 1)所有的POJO类属性必须使用包装数据类型。
* 2)RPC方法的返回值和参数必须使用包装数据类型。

1. 日期时间

* 日期格式化时，传入pattern中表示年份统一使用小写的y。
* 说明∶日期格式化时，yyyy表示当天所在的年，而大写的YYYY代表是week in which year ( JDK7之后引入的概念）,意思是当天所在的周属于的年份，一周从周日开始，周六结束，只要本周跨年，返回的YYYY就是下一年。
* 区分M---月份；m---分钟；H---24小时制；12---12小时制
* 获取当前毫秒数：System.currentTimeMillis();
* 不允许在任何程序地方使用java.sql.Date, java.sql.Time, java.sql.Timestamp
* 不要在程序中写死一年为365天，避免在公历闰年时出现日期转换错误或程序逻辑错误

1. 集合处理

* 关于 hashCode 和 equals 的处理，遵循如下规则：
* 1）只要重写 equals ，就必须重写 hashCode 。
* 2）因为 Set 存储的是不重复的对象，依据 hashCode 和 equals 进行判断，所以 Set 存储的对象必须重写这两个方法。
* 3）如果自定义对象做为 Map 的键，那么必须重写 hashCode 和 equals 。
* 不要在 foreach 循环里进行元素的 remove / add 操作。 remove 元素请使用 Iterator方式，如果并发操作，需要对 Iterator 对象加锁。
* 集合初始化时，尽量指定集合初始值大小。
* 说明： ArrayList 尽量使用 ArrayList(int initialCapacity) 初始化。
* 使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。
* 判断集合内部的元素是否为空，使用isEmpty（）方法

1. 控制语句

* 在switch块内，每个case都可以通过continue/break/return来终止，要么注释说明程序将执行到哪个case;在每一个 switch 块内，都必须包含一个 default 语句并且放在最后，即使它什么代码也没有。
* 在 if/else/for/while/do 语句中必须使用大括号，即使只有一行代码。
* 除常用方法(如 getXxx/isXxx)外，不要在条件判断中执行其它复杂的语句，将复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性。
* 方法中需要进行参数校验的场景:
* 1）调用频次低的方法。
* 2）执行时间开销很大的方法，参数校验时间几乎可以忽略不计，但如果因为参数错误导致中间执行回退，或者错误，那得不偿失。
* 3）需要极高稳定性和可用性的方法。
* 4）对外提供的开放接口，不管是RPC/API/HTTP接口。
* 5）敏感权限入口。
* 方法中不需要参数校验的场景:
* 1）极有可能被循环调用的方法，不建议对参数进行校验。但在方法说明里必须注明外部参数检查。
* 2）底层的方法调用频度都比较高，一般不校验。一般 DAO 层与 Service 层都在同一个应用中，部署在同一台服务器中，所以 DAO 的参数校验，可以省略。
* 3）被声明成private只会被自己代码所调用的方法，如果能够确定调用方法的代码传入参数已经做过检查或者肯定不会有问题，此时可以不校验参数。

1. 异常

* 异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。
* catch 时请分清稳定代码和非稳定代码，稳定代码指的是无论如何不会出错的代码。对于非稳定代码的 catch 尽可能进行区分异常类型，再做对应的异常处理。
* 捕获异常是为了处理它，不要捕获了却什么都不处理而抛弃之。最外层的业务使用者，必须处理异常，将其转化为用户可以理解的内容。
* 有 try 块放到了事务代码中，catch 异常后，如果需要回滚事务，一定要注意手动回滚事务。
* 不能在 finally 块中使用 return，finally 块中的 return 返回后方法结束执行，不会再执行 try 块中的 return 语句。
* 方法的返回值可以为 null，不强制返回空集合，或者空对象等，必须添加注释充分说明什么情况下会返回 null 值。调用方需要进行 null 判断防止 NPE 问题。
* 防止 NPE，注意 NPE 产生的场景:
* 1）返回类型为包装数据类型，有可能是null，返回int值时注意判空。
* 2）数据库的查询结果可能为null。
* 3）集合里的元素即使isNotEmpty，取出的数据元素也可能为null。
* 4）远程调用返回对象，一律要求进行NPE判断。
* 5）对于Session中获取的数据，建议NPE检查，避免空指针。
* 6）级联调用obj.getA().getB().getC();一连串调用，易产生NPE。
* 避免出现重复的代码

1. 前后端规约

* 前后端交互的APl，需要明确协议、域名、路径、请求方法、请求内容、状态码、响应体。
* 前后端数据列表相关的接口返回，如果为空，则返回空数组[]或空集合f。
* 服务端发生错误时，返回给前端的响应信息必须包含HTTP状态码，errorCode.errorMessage、用户提示信息四个部分。
* 在前后端交互的JSON格式数据中，所有的key必须为小写字母开始的lowerCamelCase风格，符合英文表达习惯，且表意完整。
* 对于需要使用超大整数的场景，服务端一律使用String字符串类型返回，禁止使用Long类型。
* HTTP请求通过URL传递参数时，不能超过2048字节。

上述的编程规范是比较严格的，我们应该养成良好地代码书写习惯，根据这些比较成熟的代码编写规范来修改完善自己的代码，并且在后面的代码书写时按照这些要求，可以提高我们代码的规范性、可读性，避免造成不必要的错误，也方便后续的修改与他人的阅读。