后台 Java 开发规范

后台 Java 开发规范

版本号	制定人	更新日期	备注
1.0.0	Max	**2019.03.01**	增加设计规约（详尽版）
1.0.1	Max	**2019.03.07**	增加 mvc 各层命名方式，模块依赖图，去掉与项目不太相关的规范，工程目录结构图，常量定义，异常常量定义，处理平滑升级，控制层规约
1.0.2	Max	**2019.04.25**	增加 control 及方法的命名规约，以及 redis 的使用规约

目录

目录

1. 引言

1.1. 编写目的

本手册的旨在**码出高效，码出质量**。现代软件架构的复杂性需要协同开发完成，如何高效地协同呢？无规矩不成方圆，无规范难以协同，比如，制订交通法规表面上是要限制行车权，实际上是保障公众的人身安全，试想如果没有限速，没有红绿灯，谁还敢上路行驶。对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。代码的字里行间流淌的是软件系统的血液，质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升系统稳定性，码出质量。

1.2. 背景

《Whispark Java 开发手册》是基于阿里巴巴集团技术团队的集体智慧结晶和经验总结而修订的，经历了阿里巴巴集团技术团队多次大规模一线实战的检验及不断完善，系统化地整理成册，回馈给广大开发

者。现代软件行业的高速发展对开发者的综合素质要求越来越高，因为不仅是编程知识点，其它维度的知识点也会影响到软件的最终交付质量。比如：数据库的表结构和索引设计缺陷可能带来软件上的架构缺陷或性能风险；工程结构混乱导致后续维护艰难；没有鉴权的漏洞代码易被黑客攻击等等。所以本手册以 Java 开发者为中心视角，划分为编程规约、异常日志、单元测试、安全规约、MySQL 数据库、工程结构、设计规约七个维度，再根据内容特征，细分成若干二级子目录。根据约束力强弱及故障敏感性，规约依次分为强制、推荐、参考三大类。对于规约条目的延伸信息中，“说明”对规约做了适当扩展和解释；“正例”提倡什么样的编码和实现方式；“反例”说明需要提防的雷区，以及真实的错误案例。

1.3. 定义

1.4. 参考资料

阿里巴巴开发手册 1.4 版

2. 编程规约

2.1. 命名风格

1. 【强制】类名使用 UpperCamelCase 风格，但以下情形例外：DO / BO / DTO / VO / AO /

   PO / UID 等。

   正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

   反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

2. 【强制】类名定义不要再加额外前缀，service bean name 当前实现类的接口 service 首字母小写为准。如：UserService 的实现类，bean name 为：userService。常量定义需要继承 BaseConstant，VO 定义也需要继承 BaseVO，异常类常量定义需要继承 baseExceptionConstant，code 定义名称以当前模块缩写+0001。新建的工程一般以实际的业务定义来确定名称。如 app-meet，指当前模块是存放 meet 见面服务相关。Service，dao，可根据表名驼峰式命名，controller 名称定义均以实际业务意义定义，

3. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

   正例： localValue / getHttpMessage() / inputUserId

4. 【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。正例：MAX_STOCK_COUNT

   反例：MAX_COUNT

5. 【强制】POJO 类中布尔类型的变量，都不要加 is 前缀，否则部分框架解析会引起序列化错误。反例：定义为基本数据类型 Boolean isDeleted 的属性，它的方法也是 isDeleted()，RPC 框架在反向解析的时候，“误以为”对应的属性名称是 deleted，导致属性获取不到，进而抛出异常。

6. 【强制】包名统一使用小写，点分隔符之间有且仅有一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。

   正例：应用工具类包名为 com.alibaba.ai.util、类名为 MessageUtils（此规则参考 spring

   的框架结构）

7. 【强制】杜绝完全不规范的缩写，避免望文不知义。

AbstractClass“缩写”命名成 AbsClass；condition“缩写”命名成 condi，类随

意缩写严重降低了代码的可阅读性。

8.【强制】各层命名规约：

A) Service/DAO 层方法命名规约

1） 获取单个对象的方法用 get 做前缀。

2） 获取多个对象的方法用 findxxxList 命名。

3） 获取统计值的方法用 getxxxCount 命名。

4） 插入的方法用 save 做前缀。

5） 删除的方法用 delete 做前缀。

6） 修改的方法用 update 做前缀，如果只是更新少数字段，请新起方法去更新。

7） 如果是单个查询条件查询，可以命名如 getXXXByNo findXXListByNo

8） 如果多个条件则注释标明查询条件

B) 领域模型命名规约

1） 数据对象：xxx，xxx 即为数据表名驼峰式无下划线。

2） 数据传输对象：xxxDTO，xxx 为业务领域相关的名称。

3） 展示对象：xxxVO，xxx 一般为网页名称。

4） POJO 是 DO/DTO/BO/VO 的统称，禁止命名成 xxxPOJO。

9.【推荐】为了达到代码自解释的目标，任何自定义编程元素在命名时，使用尽量完整的单词

组合来表达其意。

正例：在 JDK 中，表达原子更新的类名为：AtomicReferenceFieldUpdater。

变量 int a 的随意命名方式。

10. 【推荐】如果模块、接口、类、方法使用了设计模式，在命名时需体现出具体模式。

说明：将设计模式体现在名字中，有利于阅读者快速理解架构设计理念。

正例：public class OrderFactory;

public class LoginProxy;

public class ResourceObserver;

11. 【强制】接口类中的方法和属性不要加任何修饰符号（public 也不要加），保持代码的简洁性，并加上有效的 Javadoc 注释。尽量不要在接口里定义变量，如果一定要定义变量，肯定是

与接口方法相关，并且是整个应用的基础常量。

正例：接口方法签名 void commit();

接口基础常量 String COMPANY = "alibaba"; 反例： 接口方法定义 public abstract void f();

说明：JDK8 中接口允许有默认实现，那么这个 default 方法，是对所有实现类都有价值的默认实现。

12. 接口和实现类的命名有两套规则：

1）【强制】对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别。

正例：CacheServiceImpl 实现 CacheService 接口。

2）【推荐】 如果是形容能力的接口名称，取对应的形容词为接口名（通常是–able 的形式）。

正例：AbstractTranslator 实现 Translatable 接口。

13. 【参考】枚举类名建议带上 Enum 后缀，枚举成员名称需要全大写，单词间用下划线隔开。

说明：枚举其实就是特殊的类，域成员均为常量，且构造方法被默认强制是私有。

正例：枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS / UNKNOWN_REASON。

2.2. 常量定义

1. 【强制】在 long 或者 Long 赋值时，数值后使用大写的 L，不能是小写的 l，小写容易跟数字 1 混淆，造成误解。

说明：Long a = 2l; 写的是数字的 21，还是 Long 型的 2?

2. 【推荐】不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。

说明：大而全的常量类，杂乱无章，使用查找功能才能定位到修改的常量，不利于理解和维护。

正例：缓存相关常量放在类 CacheConsts 下；系统配置相关常量放在类 ConfigConsts 下，异常类放在 ExceptionConsts 下。

3. 【推荐】常量的复用层次有五层：跨模块共享常量、模块内共享常量、包内共享常量、类内共享常量。

1） 跨模块共享常量：放置在 model 模块，通常是 com.whispark.common 中的 constant 目录下。

2） 模块内共享常量：通常是子模块 com.whispark.common 中的 constant 目录下。

易懂变量也要统一定义成应用内共享常量，两位攻城师在两个类中分别定义了表示“是”的变量：

类 A 中：public static final String YES = "yes";

类 B 中：public static final String YES = "y";

A.YES.equals(B.YES)，预期是 true，但实际返回为 false，导致线上问题。

3）包内共享常量：即在当前包下单独的 constant 目录下。

4）类内共享常量：直接在类内部 private static final 定义。

5. 【推荐】如果变量值仅在一个固定范围内变化用 enum 类型来定义。

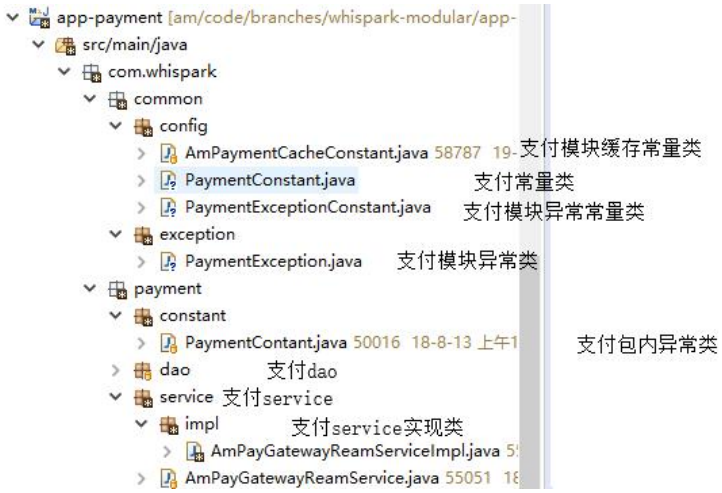
说明：如果存在名称之外的延伸属性应使用 enum 类型，下面正例中的数字就是延伸信息，表示一年中的第几个季节。

正例：

```
public enum SeasonEnum {
    SPRING(1), SUMMER(2), AUTUMN(3), WINTER(4);
    private int seq;
    SeasonEnum(int seq){
        this.seq = seq;
    }
}
```

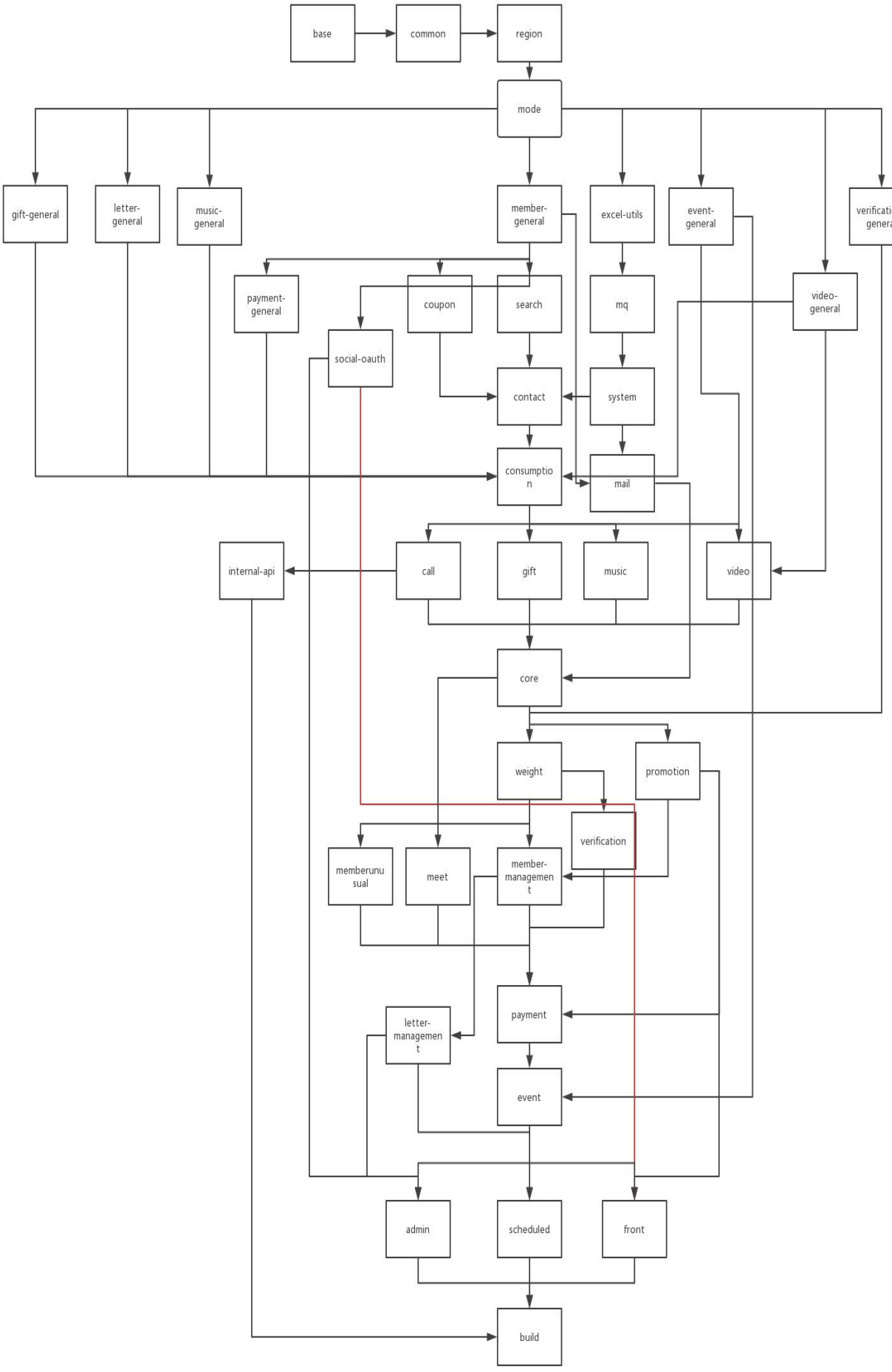
2.3. 新建模块

1.【强制】目录结构：以现在支付模块为例



2.【强制】模块依赖

现在模块之间依赖关系为:


base
common
region
mode
gift-general
letter-general
music-general
member-general
excel-utils
event-general
verification-general
payment-general
coupon
search
mq
video-general
social-oauth
contact
system
consumption
mail
internal-api
call
gift
music
video
core
weight
promotion
verification
memberunusual
meet
member-management
payment
letter-management
event
admin
scheduled
front
build

新建模块要根据上面模块依赖添加基础引用为 model 模块，禁止出现重复依赖，多次依赖问题。

2.4. 代码格式

1. 可以导入 ide 格式模板，每次提交代码格式化。模板路径：svn://192.168.0.150/docs/05 编码开发/eclipse 配置模板

2. 【强制】在 if/else/for/while/do 语句中必须使用大括号，即使只有一行代码，避免使用 下面的形式：if (condition) statements;

3. 【强制】采用 4 个空格缩进，禁止使用 tab 字符。
说明：如果使用 tab 缩进，必须设置 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，必须勾选 insert spaces for tabs。
正例：（涉及 1-5 点）

```
public static void main(String[] args) {

    // 缩进 4 个空格
    String say = "hello";
    // 运算
符的左右
必须有一
个空格
    int flag =
    0;
    // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0
    与右括号不需要空格 if (flag == 0) {
        System.out.println(say);
    }

    // 左大括号前加空格且不换行；左大括号后换行
    if (flag == 1) {
        System.out.println("world");
    // 右大括号前换行，右大括号后有 else，不用换行
    } else
        { System.o
        ut.println("
        ok");

    // 在右大括号后直接结束，则必须换行
    }
}
```

6.【强制】注释的双斜线与注释内容之间有且仅有一个空格。正例：

```
// 这是示例注释，请注意在双斜线之后有一个空格
String ygb = new String();
```

。

7.【推荐】单个方法的总行数不超过 80 行。

说明：包括方法签名、结束右大括号、方法内代码、注释、空行、回车及任何不可见字符的总行数不超过 80 行。

正例：代码逻辑分清红花和绿叶，个性和共性，绿叶逻辑单独出来成为额外方法，使主干代码

更加清晰；共性逻辑抽取成为共性方法，便于复用和维护。

8.【推荐】没有必要增加若干空格来使某一行的字符与上一行对应位置的字符对齐。

正例：

```
int one = 1;
long two = 2L;
float three = 3F;
StringBuffer sb = new StringBuffer();
```

说明：增加 sb 这个变量，如果需要对齐，则给 a、b、c 都要增加几个空格，在变量比较多的

情况下，是非常累赘的事情。

9.【推荐】不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。说明：任何情形，没有必要插入多个空行进行隔开。

2.5. OOP 规约

1.【强制】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。

2.【强制】不能使用过时的类或方法。

说明：java.net.URLDecoder 中的方法 decode(String encodeStr) 这个方法已经过时，应

该使用双参数 decode(String source, String encode)。接口提供方既然明确是过时接口，

那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

3.【强制】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。

正例："test".equals(object);

反例：object.equals("test");

说明：推荐使用 java.util.Objects#equals（JDK7 引入的工具类）

4.【强制】所有的相同类型的包装类对象之间值的比较，全部使用 equals 方法比较。说明：对于 Integer var = ? 在-128 至 127 范围内的赋值，Integer 对象是在 IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用 equals 方法进行判断。

5.关于基本数据类型与包装数据类型的使用标准如下：

1） 【强制】所有的 POJO 类属性必须使用包装数据类型。

2） 【强制】RPC 方法(即：远程调用方法)的返回值和参数必须使用包装数据类型。

3） 【推荐】所有的局部变量使用基本数据类型。

说明：POJO 类属性没有初值是提醒使用者在需要使用时，必须自己显式地进行赋值，任何

NPE 问题，或者入库检查，都由使用者来保证。

正例：数据库的查询结果可能是 null，因为自动拆箱，用基本数据类型接收有 NPE 风险。

反例：比如显示成交总额涨跌情况，即正负 x%，x 为基本数据类型，调用的 RPC 服务，调用不成功时，返回的是默认值，页面显示为 0%，这是不合理的，应该显示成中划线。所以包装

数据类型的 null 值，能够表示额外的信息，如：远程调用失败，异常退出。

6.【强制】定义 DO/DTO/VO 等 POJO 类时，不要设定任何属性默认值。

反例：POJO 类的 gmtCreate 默认值为 new Date()，但是这个属性在数据提取时并没有置入具体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

7.【强制】序列化类新增属性时，请不要修改 serialVersionUID 字段，避免反序列失败；如果完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。

说明：注意 serialVersionUID 不一致会抛出序列化运行时异常。

8.【强制】禁止在 POJO 类中，同时存在对应属性 xxx 的 isXxx()和 getXxx()方法。说明：框架在调用属性 xxx 的提取方法时，并不能确定哪个方法一定是被优先调用到。

9.【推荐】使用索引访问用 String 的 split 方法得到的数组时，需做最后一个分隔符后有无内容的检查，否则会有抛 IndexOutOfBoundsException 的风险。

说明：

```
String str = "a,b,c,,";
String[] ary = str.split(",");
//  预期大于 3，
结果是 3
System.out.println(
ary.length);
```

10.【推荐】当一个类有多个构造方法，或者多个同名方法，按参数数量排序，这些方法应该按顺序放置在一起，便于阅读

11.【推荐】 类内方法定义的顺序依次是：公有方法或保护方法 > 私有方法 > getter/setter

方法。

说明：公有方法是类的调用者和维护者最关心的方法，首屏展示最好；保护方法虽然只是子类关心，也可能是“模板设计模式”下的核心方法；而私有方法外部一般不需要特别关心，是一个

12.【推荐】setter 方法中，参数名称与类成员变量名称一致，this.成员名 = 参数名。在 getter/setter 方法中，不要增加业务逻辑，增加排查问题的难度。

反例：

```
public Integer getData() {
    if (condition) {
        return this.data + 100;
    } else {
        return this.data - 100;
    }
}
```

13.【推荐】循环体内，字符串的连接方式，使用 StringBuilder 的 append 方法进行扩展。说明：下例中，反编译出的字节码文件显示每次循环都会 new 出一个 StringBuilder 对象，然后进行 append 操作，最后通过 toString 方法返回 String 对象，造成内存资源浪费。

反例：

```
String str = "start";
for (int i = 0; i < 100; i++) {
    str = str + "hello";
}
```

14.【推荐】慎用 Object 的 clone 方法来拷贝对象。

说明：对象的 clone 方法默认是浅拷贝，若想实现深拷贝需要重写 clone 方法实现域对象的

深度遍历式拷贝。

15.【推荐】类成员与方法访问控制从严：

1） 如果不允许外部直接通过 new 来创建对象，那么构造方法必须是 private。

2） 工具类不允许有 public 或 default 构造方法。

3） 类非 static 成员变量并且与子类共享，必须是 protected。

4） 类非 static 成员变量并且仅在本类使用，必须是 private。

5） 类 static 成员变量如果仅在本类使用，必须是 private。

6） 若是 static 成员变量，考虑是否为 final。

7） 类成员方法只供类内部调用，必须是 private。

8）类成员方法只对继承类公开，那么限制为 protected。

说明：任何类、方法、参数、变量，严控访问范围。过于宽泛的访问范围，不利于模块解耦。

思考：如果是一个 private 的方法，想删除就删除，可是一个 public 的 service 成员方法或

成员变量，删除一下，不得手心冒点汗吗？变量像自己的小孩，尽量在自己的视线内，变量作用域太大，无限制的到处跑，那么你会担心的。

2.6. 集合处理

1. 【强制】使用集合转数组的方法，必须使用集合的 toArray(T[] array)，传入的是类型完全一样的数组，大小就是 list.size()。

说明：使用 toArray 带参方法，入参分配的数组空间不够大时，toArray 方法内部将重新分配内存空间，并返回新数组地址；如果数组元素个数大于实际所需，下标为[list.size()] 的数组元素将被置为 null，其它数组元素保持原值，因此最好将方法入参数组大小定义与集合元素个数一致。

正例：

```
List<String> list = new ArrayList<String>(2);
list.add("guan");
list.add("bao");
String[] array = new String[list.size()];
array = list.toArray(array);
```

反例：直接使用 toArray 无参方法存在问题，此方法返回值只能是 Object[] 类，若强转其它

类型数组将出现 ClassCastException 错误。

2. 【强制】使用工具类 Arrays.asList()把数组转换成集合时，不能使用其修改集合相关的方法，它的 add/remove/clear 方法会抛出 UnsupportedOperationException 异常。

说明：asList 的返回对象是一个 Arrays 内部类，并没有实现集合的修改方法。Arrays.asList

体现的是适配器模式，只是转换接口，后台的数据仍是数组。

```
String[] str = new String[] { "you", "wu" };

List list = Arrays.asList(str);
```

第一种情况：list.add("yangguanbao"); 运行时异常。

第二种情况：str[0] = "gujin"; 那么 list.get(0)也会随之修改。

3. 【强制】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。

正例：

```
List<String> list = new ArrayList<>();
list.add("1");
list.add("2");
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String item = iterator.next();
    if (删除元素的条件) {
        iterator.remove();
    }
}
```

反例：

```
for (String item : list) {
    if ("1".equals(item)) {
        list.remove(item);
    }
}
```

说明：以上代码的执行结果肯定会出乎大家的意料，那么试一下把“1”换成“2”，会是同样的结果吗？

4. 【强制】 在 JDK7 版本及以上，Comparator 实现类要满足如下三个条件，不然 Arrays.sort，

Collections.sort 会报 IllegalArgumentException 异常。

说明：三个条件如下

1） x，y 的比较结果和 y，x 的比较结果相反。

2） x>y，y>z，则 x>z。

3） x=y，则 x，z 比较结果和 y，z 比较结果相同。

反例：下例中没有处理相等的情况，实际使用中可能会出现异常：

```
        new Comparator<Student>() {
            @Override
    public int compare(Student o1, Student o2) {
            return o1.getId() > o2.getId() ? 1 : -1;
             }
        };
```

5. 【推荐】集合泛型定义时，在 JDK7 及以上，使用 diamond 语法或全省略。

说明：菱形泛型，即 diamond，直接使用<>来指代前边已经指定的类型。

正例：

```
// <> diamond 方式
HashMap<String, String> userCache = new HashMap<>(16);
//  全省略方式
ArrayList<User> users = new ArrayList(10);
```

6. 【推荐】集合初始化时，指定集合初始值大小。

说明：HashMap 使用 HashMap(int initialCapacity) 初始化。

正例：initialCapacity = (需要存储的元素个数 / 负载因子) + 1。注意负载因子（即 loader factor）默认为 0.75，如果暂时无法确定初始值大小，请设置为 16（即默认值）。

反例：HashMap 需要放置 1024 个元素，由于没有设置容量初始大小，随着元素不断增加，容

量 7 次被迫扩大，resize 需要重建 hash 表，严重影响性能。

7. 【推荐】使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。

说明：keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出

key 所对应的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效

率更高。如果是 JDK8，使用 Map.foreach 方法。

正例：values()返回的是 V 值集合，是一个 list 集合对象；keySet()返回的是 K 值集合，是一个 Set 集合对象；entrySet()返回的是 K-V 值组合集合

8. 【推荐】高度注意 Map 类集合 K/V 能不能存储 null 值的情况，如下表格：

集合类	Key	Value	Super	说明

Hashtable			Dictionary	线程安全
ConcurrentHashMap			AbstractMap	锁分段技术（JDK8:CAS）
TreeMap		允许为 null	AbstractMap	线程不安全
HashMap	允许为 null	允许为 null	AbstractMap	线程不安全

由于 HashMap 的干扰，很多人认为 ConcurrentHashMap 是可以置入 null 值，而事实上，

存储 null 值时会抛出 NPE 异常。

9. 【参考】合理利用好集合的有序性(sort)和稳定性(order)，避免集合的无序性(unsort)和不稳定性(unorder)带来的负面影响。

说明：有序性是指遍历的结果是按某种比较规则依次排列的。稳定性指集合每次遍历的元素次序是一定的。如：ArrayList 是 order/unsort；HashMap 是 unorder/unsort；TreeSet 是 order/sort。

10. 【参考】利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的 contains 方法进行遍历、对比、去重操作。

2.7. 并发处理

1.【强制】获取单例对象需要保证线程安全，其中的方法也要保证线程安全。说明：资源驱动类、工具类、单例工厂类都需要注意。

2.【强制】创建线程或线程池时请指定有意义的线程名称，方便出错时回溯。

正例：

```
public class TimerTaskThread extends Thread {
    public TimerTaskThread() {
        super.setName("TimerTaskThread");
        ...
    }
}
```

3.【强制】线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。

说明：使用线程池的好处是减少在创建和销毁线程上所消耗的时间以及系统资源的开销，解决资源不足的问题。如果不使用线程池，有可能造成系统创建大量同类线程而导致消耗完内存或者“过度切换”的问题。

4.【强制】线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样

的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

说明：Executors 返回的线程池对象的弊端如下：

1）FixedThreadPool 和 SingleThreadPool:

允许的请求队列长度为 Integer.MAX_VALUE，可能会堆积大量的请求，从而导致 OOM。

2）CachedThreadPool 和 ScheduledThreadPool:

允许的创建线程数量为 Integer.MAX_VALUE，可能会创建大量的线程，从而导致 OOM。

5.【强制】SimpleDateFormat 是线程不安全的类，一般不要定义为 static 变量，如果定义为

static，必须加锁，或者使用 DateUtils 工具类。

正例：注意线程安全，使用 DateUtils。亦推荐如下处理：

```
private static final ThreadLocal<DateFormat> df = new ThreadLocal<DateFormat>()
    { @Override

    protected DateFormat initialValue() {
        return new SimpleDateFormat("yyyy-MM-dd");
    }
};
```

说明：如果是 JDK8 的应用，可以使用 Instant 代替 Date，LocalDateTime 代替 Calendar，

DateTimeFormatter 代替 SimpleDateFormat，官方给出的解释：simple beautiful strong immutable thread-safe。

6.【强制】高并发时，同步调用应该去考量锁的性能损耗。能用无锁数据结构，就不要用锁；能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。

说明：尽可能使加锁的代码块工作量尽可能的小，避免在锁代码块中调用 RPC 方法。

7.【强制】对多个资源、数据库表、对象同时加锁时，需要保持一致的加锁顺序，否则可能会造成死锁。

说明：线程一需要对表 A、B、C 依次全部加锁后才可以进行更新操作，那么线程二的加锁顺序也必须是 A、B、C，否则可能出现死锁。

8.【推荐】避免 Random 实例被多线程使用，虽然共享该实例是线程安全的，但会因竞争同一 seed 导致的性能下降。

说明：Random 实例包括 java.util.Random 的实例或者 Math.random()的方式。

正例：在 JDK7 之后，可以直接使用 API ThreadLocalRandom，而在 JDK7 之前，需要编码保

证每个线程持有一个实例。

9.【参考】 HashMap 在容量不够进行 resize 时由于高并发可能出现死链，导致 CPU 飙升，在开发过程中可以使用其它数据结构或加锁来规避此风险。

10.【参考】ThreadLocal 无法解决共享对象的更新问题，ThreadLocal 对象建议使用 static

修饰。这个变量是针对一个线程内所有操作共享的，所以设置为静态变量，所有此类实例共享此静态变量 ，也就是说在类第一次被使用时装载，只分配一块存储空间，所有此类的对象(只要是这个线程内定义的)都可以操控这个变量。

2.8. 控制语句

1.【强制】在高并发场景中，避免使用"等于"判断作为中断或退出的条件。

说明：如果并发控制没有处理好，容易产生等值判断被"击穿"的情况，使用大于或小于的区间判断条件来代替。

判断剩余奖品数量等于 0 时，终止发放奖品，但因为并发处理错误导致奖品数量瞬间变成了负数，这样的话，活动无法终止。

2.【推荐】避免采用取反逻辑运算符。

说明：取反逻辑不利于快速理解，并且取反逻辑写法必然存在对应的正向逻辑写法。

正例：使用 `if (x < 628)` 来表达 x 小于 628。

反例：使用 `if (!(x >= 628))` 来表达 x 小于 628。

3.【参考】下列情形，需要进行参数校验：

1） 调用频次低的方法。

2） 执行时间开销很大的方法。此情形中，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。

6）如果参数校验结果影响最终执行结果，以抛异常结束，避免出现安全性问题（max 添加）

2.9. 注释规约

1.【强制】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。

说明：在 IDE 编辑窗口中，Javadoc 方式会提示相关注释，生成 Javadoc 可以正确输出相应注释；在 IDE 中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高阅读效率。

2.【强制】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，如果是查询，则要表明是根据什么条件查询的。（特殊情况则额外标准）

说明：对子类的实现要求，或者调用注意事项，请一并说明。

3.【强制】废弃旧方法时标明新方法的链接，如 @see 类名#方法名

4.【强制】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。

5.【推荐】与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持

6.【推荐】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。

说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，就失去了导航的意义。

7.【参考】谨慎注释掉代码。在上方详细说明，而不是简单地注释掉。如果无用，则删除。

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。2）永久不用。前者如果没有备注信息，难以知晓注释动机。后者直接删掉（代码仓库保存了历史代码）。

8.【参考】对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。

9.【参考】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]，说明）

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）。

说明：当前的功能是什么、逻辑大概是什么样子的、要想怎么去做、当前版本有问题。

2）错误，不能工作（FIXME）:（标记人，标记时间，[预计处理时间]，说明）

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。

说明：当前的功能是什么、逻辑大概是什么样子的、要想怎么去做、当前版本有问题。

2.10. 控制层规约（controller）

1. 【强制】类级别的 mapping 定义要根据当前业务定义，比方当前 controller 为会员信息增删改查，则类级别 mapping 应该定义为

“/member”,最好与 control 命名一致,具体接口上定义则根据具体业务，如：查询单个会员“/get”，查询多个会员：“/list” 最好与方法的命名一致

2. 【强制】接口请求方式定义，查询为 get 方式，更新为 put 方式，新增为 post 方式

3. 【强制】新的接口定义参数，必须以 VO 接收参数，返回也用 VO 返回给前端。严格控制返回参数内容。

4. 【强制】host 端或者 agency 端，新的控制层定义必须要实现 AmAdminBaseApiController 类，旧的使用原类：AmAdminBaseController，国际端新的控制层定义必须要实现 AmFrontBaseApiController，旧的使用原类：AmFrontBaseController，使用父类的方法返回参数，不得自己定义返回格式。

2.11. 平滑升级

- 【强制】对于项目内方法升级或者是接口升级，保留原接口，待版本上线以后，下个版本再删除旧方法或者接口，旧方法要进行@Deprecated 标注，并且标明注释。
  说明：避免发版时候，刚好有用户踩到修改的接口，而原接口不存在导致的报错
- 【强制】接口中如果是参数的修改，请加多方法，保留原接口，如果保证没有其他地方调用则加上@Deprecated 标注，并且标明注释。等到下一个版本再删除掉不用的接口

2.12. 其它

1. 【强制】如果要使用 Redis，必须使用 redis 的封装类 Icache，然后设置过期时间,key 的命名规范，格式：哪个端.模块名.常量名 如 front.gift.giftCode。
2. 【强制】在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。
   说明：不要在方法体内定义：Pattern pattern = Pattern.compile(“规则”);
3. 【强制】Mybatis 映射文件 xml，顺序按照 Columns（查询内容），joins（关联表）,queryCondition（查询条件）,select（查询）,update（更新）,insert（新增）,delete（删除）的顺序，没有的略过。
4. 【强制】velocity 调用 POJO 类的属性时，建议直接使用属性名取值即可，模板引擎会自动按规范调用 POJO 的 getXxx()，如果是 boolean 基本数据类型变量（boolean 命名不需要加 is 前缀），会自动调用 isXxx()方法。

说明：注意如果是 Boolean 包装类对象，优先调用 getXxx()的方法。

5. 【强制】注意 Math.random() 这个方法返回是 double 类型，注意取值的范围 0≤x<1（能够取到零值，注意除零异常），如果想获取整数类型的随机数，不要将 x 放大 10 的若干倍然后取整，直接使用 Random 对象的 nextInt 或者 nextLong 方法。

6. 【强制】获取当前毫秒数 System.currentTimeMillis(); 而不是 new Date().getTime();

   说明：如果想获取更加精确的纳秒级时间值，使用 System.nanoTime()的方式。在 JDK8 中，针对统计时间等场景，推荐使用 Instant 类。

7. 【推荐】任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存。

8. 【推荐】及时清理不再使用的代码段或配置信息。

   说明：对于垃圾代码或过时配置，坚决清理干净，避免程序过度臃肿，代码冗余。

   正例：对于暂时被注释掉，后续可能恢复使用的代码片断，在注释代码上方，统一规定使用

3. 异常日志

3.1. 异常处理

1.【强制】异常的常量定义必须要唯一，整个项目只需要定义一个异常类（BaseException），用不同的常量即可。异常常量定义不用枚举，用普通常量定义即可。

2.【强制】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。

   说明：对大段代码进行 try-catch，使程序无法根据不同的异常做出正确的应激反应，也不利于定位问题，这是一种不负责任的表现。

正例：用户注册的场景中，如果用户输入非法字符，或用户名称已存在，或用户输入密码过于简单，在程序上作出分门别类的判断，并提示给用户。

3.【强制】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，如果不想处理它，请将该异常抛给它的调用者。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。。

4.【推荐】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。

说明：本手册明确防止 NPE 是调用者的责任。即使被调用方法返回空集合或者空对象，对调用
者来说，也并非高枕无忧，必须考虑到远程调用失败、序列化失败、运行时异常等场景返回

null 的情况。

5.【推荐】防止 NPE，是程序员的基本修养，注意 NPE 产生的场景：

1）返回类型为基本数据类型，return 包装数据类型的对象时，自动拆箱有可能产生 NPE。

反例：public int f() { return Integer 对象}， 如果为 null，自动解箱抛 NPE。

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） 远程调用返回对象时，一律要求进行空指针判断，防止 NPE。

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。正例：使用 JDK8 的 Optional 类来防止 NPE 问题。

3.2. 日志规约

1.【强制】关键的业务节点一定要打上日志，如支付系统中，开始支付时保存订单，开始下单都需要打印日志，并且带上唯一的标识，如订单号。在业务系统则尽量第一个参数带上男士 NO.，这样便于以后线上排查问题。避免出现无意义的日志打印，如：logger.info("======save order start=======")

2.【强制】打印日志需使用日志框架 SLF4J 中的 API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。

```
import org.slf4j.Logger;


import org.slf4j.LoggerFactory;
private static final Logger logger = LoggerFactory.getLogger(Abc.class)；
```

3. 【强制】对 trace/debug/info 级别的日志输出，必须使用条件输出形式或者使用占位符的方式。

说明：logger.debug("Processing trade with id: " + id + " and symbol: " + symbol);

如果日志级别是 warn，上述日志不会打印，但是会执行字符串拼接操作，如果 symbol 是对象，会执行 toString()方法，浪费了系统资源，执行了上述操作，最终日志却没有打印。

正例：（条件）建议采用如下方式

```
if (logger.isDebugEnabled()) {
    logger.debug("Processing trade with id: " + id + " and symbol: " + symbol);
}
```

正例：（占位符）

```
logger.debug("Processing trade with id: {} and symbol : {} ", id, symbol);
```

4. 【强制】避免重复打印日志，浪费磁盘空间，务必在 log4j.xml 中设置 additivity=false。

正例：<logger name="com.taobao.dubbo.config" additivity="false">

5. 【强制】异常信息应该包括两类信息：案发现场信息和异常堆栈信息。如果不处理，那么通过关键字 throws 往上抛出。

正例：logger.error(各类参数或者对象 toString() + "_" + e.getMessage(), e);

6. 【强制】使用全英文来注释和描述日志错误信息。

4. 单元测试

1. 【强制】单元测试需要继承 BaseTest，BaseTest 有基于登陆登出的一些方法封装。
2. 【强制】好的单元测试必须遵守 AIR 原则。

说明：单元测试在线上运行时，感觉像空气（AIR）一样并不存在，但在测试质量的保障上，

却是非常关键的。好的单元测试宏观上来说，具有自动化、独立性、可重复执行的特点。

- A：Automatic（自动化）
- I：Independent（独立性）
- R：Repeatable（可重复）

3.【强制】单元测试应该是全自动执行的，并且非交互式的。测试用例通常是被定期执行的，执行过程必须完全自动化才有意义。输出结果需要人工检查的测试不是一个好的单元测试。单元测试中不准使用 System.out 来进行人肉验证，必须使用 assert 来验证。

4.【强制】保持单元测试的独立性。为了保证单元测试稳定可靠且便于维护，单元测试用例之间决不能互相调用，也不能依赖执行的先后次序。

反例：method2 需要依赖 method1 的执行，将执行结果作为 method2 的输入。

5.【强制】单元测试是可以重复执行的，不能受到外界环境的影响。

说明：单元测试通常会被放到持续集成中，每次有代码 check in 时单元测试都会被执行。如果单测对外部环境（网络、服务、中间件等）有依赖，容易导致持续集成机制的不可用。

正例：为了不受外界环境影响，要求设计代码时就把 SUT 的依赖改成注入，在测试时用 spring 这样的 DI 框架注入一个本地（内存）实现或者 Mock 实现。

6.【强制】对于单元测试，要保证测试粒度足够小，有助于精确定位问题。单测粒度至多是类级别，一般是方法级别。

说明：只有测试粒度小才能在出错时尽快定位到出错位置。单测不负责检查跨类或者跨系统的交互逻辑，那是集成测试的领域。

7.【强制】核心业务、核心应用、核心模块的增量代码确保单元测试通过。

说明：新增代码及时补充单元测试，如果新增代码影响了原有单元测试，请及时修正。

8.【强制】单元测试代码必须写在如下工程目录：src/test/java，不允许写在业务代码目录下。

说明：源码构建时会跳过此目录，而单元测试框架默认是扫描此目录。

9.【推荐】和数据库相关的单元测试，可以设定自动回滚机制，不给数据库造成脏数据。或者

对单元测试产生的数据有明确的前后缀标识。

正例：在 RDC 内部单元测试中，使用 RDC_UNIT_TEST_的前缀标识数据。

10.【推荐】单元测试作为一种质量保障手段，不建议项目发布后补充单元测试用例，建议在项目提测前完成单元测试。

5. 安全规约

1. 【强制】隶属于用户个人的页面或者功能必须进行权限控制校验。

   说明：防止没有做水平权限校验就可随意访问、修改、删除别人的数据，比如查看他人的私信内容、修改他人的订单。

2. 【强制】返回参数必须要进行敏感信息的过滤如：会员的邮箱，手机号等敏感信息，只返回必要的参数，严格控制返回参数内容

3. 【强制】用户敏感数据禁止直接展示，必须对展示数据进行脱敏。

   说明：中国大陆个人手机号码显示为:158****9119，隐藏中间 4 位，防止隐私泄露。

4. 【强制】用户输入的 SQL 参数严格使用参数绑定或者 METADATA 字段值限定，防止 SQL 注入，禁止字符串拼接 SQL 访问数据库。

5. 【强制】用户请求传入的任何参数必须做有效性验证。说明：忽略参数校验可能导致：

   - page size 过大导致内存溢出
   - 恶意 order by 导致数据库慢查询

- 任意重定向
- SQL 注入
- 反序列化注入
- 正则输入源串拒绝服务 ReDoS

说明：Java 代码用正则来验证客户端的输入，有些正则写法验证普通用户输入没有问题，

但是如果攻击人员使用的是特殊构造的字符串来验证，有可能导致死循环的结果。

6. 【强制】禁止向 HTML 页面输出未经安全过滤或未正确转义的用户数据。

7. 【强制】表单、AJAX 提交必须执行 CSRF 安全验证。

说明：CSRF(Cross-site request forgery)跨站请求伪造是一类常见编程漏洞。对于存在

CSRF 漏洞的应用/网站，攻击者可以事先构造好 URL，只要受害者用户一访问，后台便在用户不知情的情况下对数据库中用户参数进行相应修改。

8. 【强制】在使用平台资源，譬如短信、邮件、电话、下单、支付，必须实现正确的防重放的机制，如数量限制、疲劳度控制、验证码校验，避免被滥刷而导致资损。

说明：如注册时发送验证码到手机，如果没有限制次数和频率，那么可以利用此功能骚扰到其它用户，并造成短信平台资源浪费。

9. 【推荐】发贴、评论、发送即时消息等用户生成内容的场景必须实现防刷、文本内容违禁词过滤等风控策略。

6. MySQL 数据库

6.1. 建表规约

1. 【强制】表名定义遵循业务场景，主键使用自增长，字段名以驼峰式命名，如创建时间：create_date，表必备字段：id, create_by,update_by,create_date,update_date,remark，del_flag。状态，类型的定义都用 int，或者是 tinyint。创建时间加索引，其他字段根据实际情况添加索引。

2. 【强制】数据库建表自增长起始值，设置为从 10000 开始，主要这样做的目的是为了后续能够人为插入数据的时候可以自己预设一个 10000 以内的 id 值

3. 【强制】表达是与否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint （1 表示是，0 表示否）。

   说明：任何字段如果为非负数，必须是 unsigned。

   注意：POJO 类中的任何布尔类型的变量，都不要加 is 前缀，所以，需要在<resultMap>设置

   从 is_xxx 到 Xxx 的映射关系。数据库表示是与否的值，使用 tinyint 类型，坚持 is_xxx 的命名方式是为了明确其取值含义与取值范围。

   正例：表达逻辑删除的字段名 is_deleted，1 表示删除，0 表示未删除。

4. 【强制】表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。

   正例：aliyun_admin，rdc_config，level3_name

   反例：AliyunAdmin，rdcConfig，level_3_name

5. 【强制】表名不使用复数名词。

说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

6. 【强制】禁用保留字，如 desc、range、match、delayed 等，请参考 MySQL 官方保留字。

7. 【强制】主键索引名为 pk_字段名；唯一索引名为 uk_字段名；普通索引名则为 idx_字段名。说明：pk_ 即 primary key；uk_ 即 unique key；idx_ 即 index 的简称。

8. 【强制】小数类型为 decimal，禁止使用 float 和 double。

   说明：float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

9. 【强制】如果存储的字符串长度几乎相等，使用 char 定长字符串类型。

10. 【强制】varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text，独立出来一张表，用主键来对应，避免影响其它字
    段索引效率。

11. 【推荐】表的命名最好是加上“业务名称_表的作用”。

    正例：alipay_task / force_project / trade_config

12. 【推荐】库名与应用名称尽量一致。

13. 【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

14. 【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

    1）不是频繁修改的字段。

    2）不是 varchar 超长字段，更不能是 text 字段。

    正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存

    储类目名称，避免关联查询。

15. 【推荐】单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。

说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

16. 【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

正例：如下表，其中无符号值可以避免误存负数，且扩大了表示范围。

对象	年龄区间	类型	字节	表示范围
人	150 岁之内	tinyint unsigned	1	无符号值：0 到 255
龟	数百岁	smallint unsigned	2	无符号值：0 到 65535
恐龙化石	数千万年	int unsigned	4	无符号值：0 到约 42.9 亿
太阳	约 50 亿年	bigint unsigned	8	无符号值：0 到约 10 的 19 次方

6.2. 索引规约

1.【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。

说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

2.【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度即可。

说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度会高达 90%以上，可以使用 count(distinct left(列名, 索引长度))/count(*)的区分度来确定。

3.【强制】数据表有添加索引的字段，对该字段插入或更新时建议不要设为 null 值，避免查询时索引失效

4.【强制】列表查询条件使用模糊查询的时候，需要评估后期数据量大不大，如果过大，则需要建议产品修改需求

5.【推荐】如果有 order by 的场景，请注意利用索引的有序性。order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。正例：where a=? and b=? order by c; 索引：a_b_c

反例：索引中有范围查找，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b; 索引 a_b 无法排序。

6.【推荐】利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

正例：先快速定位需要获取的 id 段，然后再关联：

```
SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b.id
```

7.【推荐】 SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts

最好。

说明：

1）consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。

2）ref 指的是使用普通的索引（normal index）。

3）range 对索引进行范围检索。

反例：explain 表的结果，type=index，索引物理文件全扫描，速度非常慢，这个 index 级

别比较 range 还低，与全表扫描是小巫见大巫。

8.【推荐】建组合索引的时候，区分度最高的在最左边。

正例：如果 where a=? and b=? ，如果 a 列的几乎接近于唯一值，那么只需要单建 idx_a 索引即可。

说明：存在非等号和等号混合时，在建索引时，请把等号条件的列前置。如：where c>? and d=? 那么即使 c 的区分度更高，也必须把 d 放在索引的最前列，即索引 idx_d_c。

9.【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

10.【参考】创建索引时避免有如下极端误解：

1）宁滥勿缺。认为一个查询就需要建一个索引。

2）宁缺勿滥。认为索引会消耗空间、严重拖慢更新和新增速度。

3）抵制惟一索引。认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

6.3. SQL 语句

1. 【强制】不要使用 count(列名)或 count(常量)来替代 count(*)，count(*)是 SQL92 定义的

   标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。

   说明：count(*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。

2. 【强制】count(distinct col) 计算该列除 NULL 之外的不重复行数，注意 count(distinct col1, col2) 如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。

3. 【强制】当某一列的值全是 NULL 时，count(col)的返回结果为 0，但 sum(col)的返回结果为

   NULL，因此使用 sum()时需注意 NPE 问题。

   正例：可以使用如下方式来避免 sum 的 NPE 问题：SELECT IF(ISNULL(SUM(g)),0,SUM(g)) FROM table;

4. 【强制】使用 ISNULL()来判断是否为 NULL 值。
   说明：NULL 与任何值的直接比较都为 NULL。

   1） NULL<>NULL 的返回结果是 NULL，而不是 false。

   2） NULL=NULL 的返回结果是 NULL，而不是 true。

   3） NULL<>1 的返回结果是 NULL，而不是 true。

5. 【强制】 在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。

6. 【强制】不得使用外键与级联，一切外键概念必须在应用层解决。

   说明：以学生和成绩的关系为例，学生表中的 student_id 是主键，那么成绩表中的 student_id 则为外键。如果更新学生表中的 student_id，同时触发成绩表中的 student_id 更新，即为级联更新。外键与级联更新适用于单

机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

7. 【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

8. 【强制】数据订正（特别是删除、修改记录操作）时，要先 select，避免出现误删除，确认无误才能执行更新语句。

9. 【推荐】in 操作能避免则避免，若实在避免不了，需要仔细评估 in 后边的集合元素数量，控制在 1000 个之内。

10. 【参考】如果有国际化需要，所有的字符存储与表示，均以 utf-8 编码，注意字符统计函数的区别。

    说明：

    SELECT LENGTH("轻松工作"); 返回为 12

    SELECT CHARACTER_LENGTH("轻松工作"); 返回为 4

    如果需要存储表情，那么选择 utf8mb4 来进行存储，注意它与 utf-8 编码的区别。

11. 【参考】TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE

    无事务且不触发 trigger，有可能造成事故，故不建议在开发代码中使用此语句。

    说明：TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同。

6.4. ORM 映射

1. 【强制】在表查询中，一律不要使用 * 作为查询的字段列表，需要哪些字段必须明确写明。说明：1）增加查询分析器解析成本。2）增减字段容易与 resultMap 配置不一致。3）无用字段增加网络消耗，尤其是 text 类型的字段。

2. 【强制】POJO 类的布尔属性不能加 is，而数据库字段必须加 is_，要求在 resultMap 中进行字段与属性之间的映射。

说明：参见定义 POJO 类以及数据库字段定义规定，在<resultMap>中增加映射，是必须的。在 MyBatis Generator 生成的代码中，需要进行对应的修改。

3. 【强制】不要用 resultClass 当返回参数，即使所有类属性名与数据库字段一一对应，也需要定义；反过来，每一个表也必然有一个 POJO 类与之对应。

   说明：配置映射关系，使字段与 DO 类解耦，方便维护。

4. 【强制】sql.xml 配置参数使用：#{}，#param# 不要使用${} 此种方式容易出现 SQL 注入。

5. 【强制】iBATIS 自带的 queryForList(String statementName,int start,int size)不推荐使用。

   说明：其实现方式是在数据库取到 statementName 对应的 SQL 语句的所有记录，再通过 subList

   取 start,size 的子集合。

   正例：
   ```
   Map<String, Object> map = new HashMap<>();
   map.put("start", start);
   map.put("size", size);
   ```

6. 【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出。

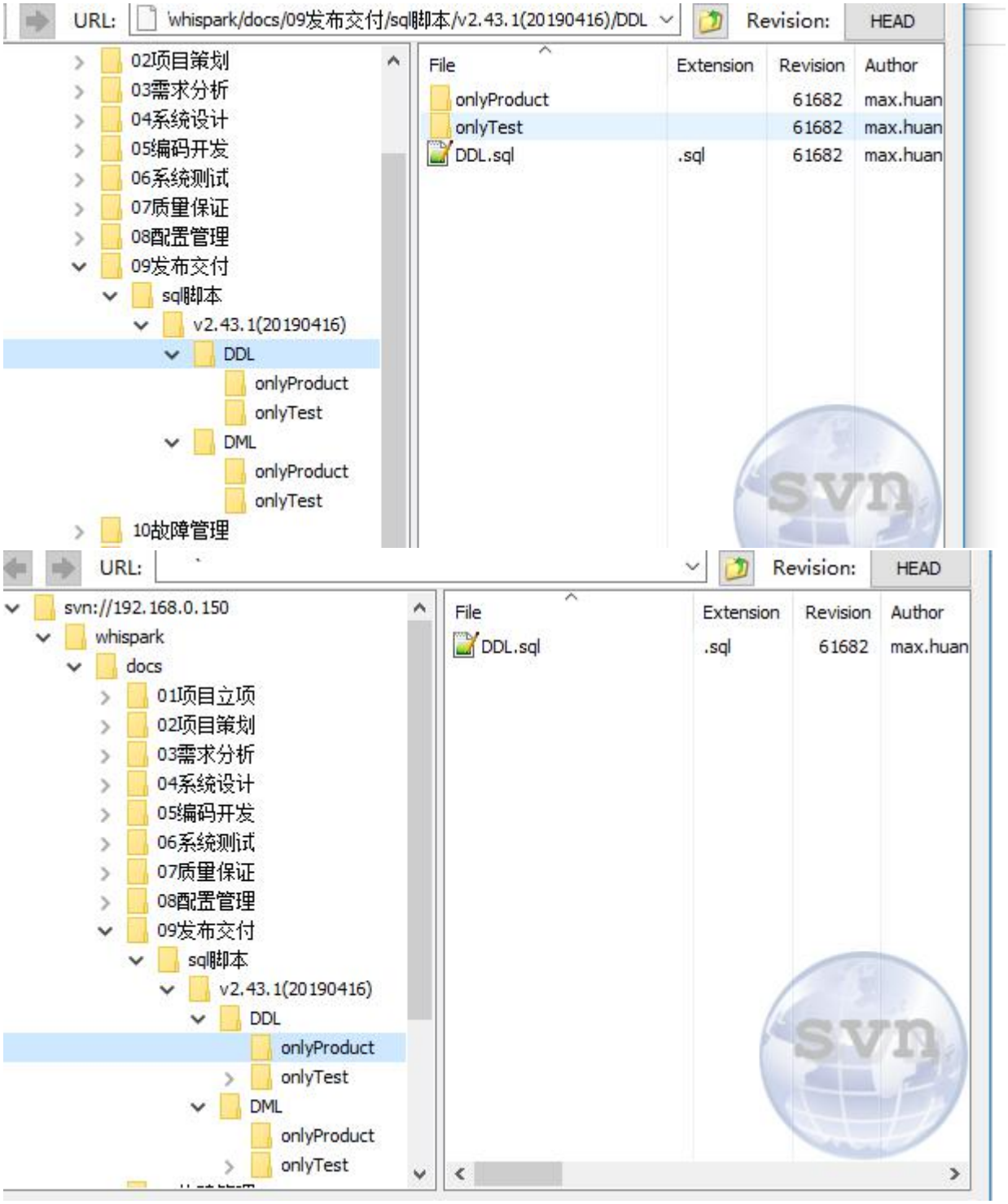
   说明：resultClass="Hashtable"，会置入字段名和属性值，但是值的类型不可控。

7. 【强制】更新数据表记录时，必须同时更新记录对应的 gmt_modified 字段值为当前时间。【推荐】不要写一个大而全的数据更新接口。传入为 POJO 类，不管是不是自己的目标更新字段，都进行 update table set c1=value1,c2=value2,c3=value3; 这是不对的。执行 SQL 时，不要更新无改动的字段，一是易出错；二是效率低；三是增加 binlog 存储。

6.5. Sql 语句文件

由于目前版本 sql 语句执行比较困难。后续打算以版本为节点，执行 sql，分为 dml（数据操纵语言）和 ddl（数据库定义语言）两种。

结构如：



7. 设计规约

1. 【强制】存储方案和底层数据结构的设计获得评审一致通过，并沉淀成为文档。

说明：有缺陷的底层数据结构容易导致系统风险上升，可扩展性下降，重构成本也会因历史数据迁移和系统平滑过渡而陡然增加，所以，存储方案和数据结构需要认真地进行设计和评审，生产环境提交执行后，需要进行 double check。

正例：评审内容包括存储介质选型、表结构设计能否满足技术方案、存取性能和存储空间能否满足业务发展、表或字段之间的辩证关系、字段名称、字段类型、索引等；数据结构变更（如在原有表中新增字段）也需要进行评审通过后上线。

2. 【强制】如果某个业务对象的状态超过 3 个，使用状态图来表达并且明确状态变化的各个触发条件。

   说明：状态图的核心是对象状态，首先明确对象有多少种状态，然后明确两两状态之间是否存在直接转换关系，再明确触发状态转换的条件是什么。

   正例：淘宝订单状态有已下单、待付款、已付款、待发货、已发货、已收货等。比如已下单与已收货这两种状态之间是不可能有直接转换关系的。

3. 【强制】如果系统中模型类超过 5 个，并且存在复杂的依赖关系，使用类图来表达并且明确类之间的关系。

   说明：类图像建筑领域的施工图，如果搭平房，可能不需要，但如果建造蚂蚁 Z 空间大楼，肯定需要详细的施工图。

4. 【推荐】需求分析与系统设计在考虑主干功能的同时，需要充分评估异常流程与业务边界。反例：用户在淘宝付款过程中，银行扣款成功，发送给用户扣款成功短信，但是支付宝入款时由于断网演练产生异常，淘宝订单页面依然显示未付款，导致用户投诉。

5. 【推荐】类在设计与实现时要符合单一原则。

说明：单一原则最易理解却是最难实现的一条规则，随着系统演进，很多时候，忘记了类设计的初衷。

6. 【推荐】谨慎使用继承的方式来进行扩展，优先使用聚合/组合的方式来实现。

说明：不得已使用继承的话，必须符合里氏代换原则，此原则说父类能够出现的地方子类一定能够出现，比如，“把钱交出来”，钱的子类美元、欧元、人民币等都可以出现。

7.【推荐】系统设计时，根据依赖倒置原则，尽量依赖抽象类与接口，有利于扩展与维护。

说明：低层次模块依赖于高层次模块的抽象，方便系统间的解耦。

8.【推荐】系统设计时，注意对扩展开放，对修改闭合。

说明：极端情况下，交付的代码都是不可修改的，同一业务域内的需求变化，通过模块或类的扩展来实现。

9.【推荐】系统设计阶段，共性业务或公共行为抽取出来公共模块、公共配置、公共类、公

共方法等，避免出现重复代码或重复配置的情况。

说明：随着代码的重复次数不断增加，维护成本指数级上升。

10.【参考】系统设计主要目的是明确需求、理顺逻辑、后期维护，次要目的用于指导编码。

说明：避免为了设计而设计，系统设计文档有助于后期的系统维护，所以设计结果需要进行分类归档保存。

11.【参考】设计的本质就是识别和表达系统难点，找到系统的变化点，并隔离变化点。

说明：世间众多设计模式目的是相同的，即隔离系统变化点。

12.【参考】系统架构设计的目的：

- 确定系统边界。确定系统在技术层面上的做与不做。
- 确定系统内模块之间的关系。确定模块之间的依赖关系及模块的宏观输入与输出。
- 确定指导后续设计与演化的原则。使后续的子系统或模块设计在规定的框架内继续演化。
- 确定非功能性需求。非功能性需求是指安全性、可用性、可扩展性等。