# 命名风格

1. 代码命名不能以下划线或者美元符号开头或者结尾。

2. 代码命名不能以中文拼音或者中文拼音与英文混合方式。

3. 类名使用UpperCamCamelCase风格，但DO、PO、DTO、VO、BO等除外。

4. 方法名、参数名、变量名统一使用lowerCamelCase，必须遵守驼峰命名。

5. 常量名全部大写，单词间用下划线隔开。

6. 抽象类必须以Abstract或者Base开头，异常类必须以Exception结尾，测试类以测试的类的名称开头Test结尾。

7. 类型与中括号紧挨相连标示数组。

8. POJO类中布尔类型变量不要加is前缀。

9. 包名统一小写，点分隔符有且有一个自然语义单词。

10. 避免在父子类和不同代码块中采用相同变量名。

11. 避免不规范的缩写命名。

12. 在对元素命名时用完整单词组合表达其意。

13. 常量和变量命名时，表示类型放在词尾，如：idList、TERMINATED\_TREAD\_COUNT。

14. 接口、类、方法、模块使用设计模式，命名时要体现具体模式。

15. 接口类中的方法和属性不要加任何修饰符，并加上有效的javadoc。

16. 接口和实现类的命名规则：

1、对于service和dao类，实现类必须用Impl结尾；

2、如果是形容能力的接口名称，取对应的形容词为接口名 AbstractTranslator实现 Translatable接口。

17. 枚举类名加Enum后缀，枚举成员名称全大写，单词间用下划线隔开。

18. 各层命名规范：

A) Service/DAO层命名规约

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list做前缀，如：listObjects。

3.获取统计值的方法用count做前缀。

4.插入方法用save/insert做前缀。

5.删除方法用delete/remove做前缀。

6.修改方法用update做前缀。

B）领域模型命名规范

1.数据对象：xxxDO, xxx为数据库表名。

2.数据传输对象：xxxDTO,xxx为业务模型相关名称。

3.展示对象：xxxVO，xxx一般为网页名称。

4.POJO是对DO、DTO、VO、BO的统称，禁止xxxPOJO。

# 常量定义

1. 代码中禁止出现魔法值。

2. 在Long类型中赋值，数值后使用大写L。

3. 不要在一个常量类中维护所有常量，要根据功能分开维护。

4. 常量的复用层次：

1.跨应用：放在二方库中，通常在constant目录下。

2.应用内：放在一方库中，通常在constant目录下。

3.子工程内：放在当前子工程constant目录下。

4.包内共享常量：当前包下单独的constant目录下。

5.类内共享常量：直接在类内部private static final定义。

6. 如果变量值只在固定的范围内变化，用enum类型定义。

# 代码格式

1. 如果大括号代码为空直接'{}',大括号内有代码则：左大括号左侧不换行，右侧换行；右大括号右侧换行，左侧如果不跟else等代码换行，否则不换行。
2. 小括号和字符之间不能有空格，括号内字符和运算符之间有空格 如：if (a == b)。
3. if、for、while、do、switch与括号之间必须有空格。
4. 任何二目、三目运算符前后必须有空格。
5. 采用4个空格，禁止使用tab。
6. 注释的双斜线和内容要有空格。
7. 强制类型转换时，右括号与强制转换值之间不用空格。
8. 单行字符不超过120个，超过要换行。
9. 方法在定义和传参时，必须要加空格。
10. IDE的text file encoding 设置为UTF-8；IDE中 文件的换行符使用Unix格式。
11. 单个方法尽量不超过80行。
12. 不同逻辑、不同语义、不同业务之间的代码插入一个空行分隔符。

# OOP规范

1. 不用一个类型的对象引用来访问静态方法和静态属性，直接类名访问即可。

2. 所有覆写方法，必须加@Override注解。

3. 相同业务含义，相同参数类型才能使用java可变参数。

4. 外部依赖或者二方库依赖的接口，不能修改方法签名。接口过时必须用@Deprecated 注解，并说明新接口或者新服务是什么。

5. 不能使用过时的类或者方法。

6. Object的equals方法容易抛出空指针，应使用常量或者确定值的对象来调用equals。

7. 所有整型包装类之间的值比较都用equals 方法比较。

8. 浮点数之间的等值判断，基本类型不能用==，包装类不能用equals。

解决方案：(1) 指定一个误差范围，两个浮点数的差值在此范围之内，则认为是相等的。

(2) 使用BigDecimal来定义值，再进行浮点数的运算操作。

9. 定义DO类时，属性类型要数据库字段类型相匹配。

10. 防止精度丢失，禁止使用BigDecimal(double)方式将double对象转换成BigDecimal。建议使用BigDecimal的valueOf方法。

11. 基本类型和包装类型的使用标准

1.所有POJO的属性必须用包装类型。

2.RPC方法的参数和返回值必须使用包装类型。

3.所有局部变量使用基本变量。

12. 所有POJO 不要对其属性设置默认值。

13. 序列化类新增时不要修改其serialVersionUID字段。

14. 构造方法里禁止加任何业务处理逻辑，有要加在init()。

15. POJO类必须要写toString方法。

16. 禁止在POJO类中对属性xxx 同时存在isXxx()和getXxx()。

17. 使用索引访问用String的split方法得到数组时，需要对最后一个分隔符有无内容做检查。

18. 一个类有多个构造方法或者多个同名方法，要按照顺序来。

19. 类中的方法顺序 ：共有方法-> 私有方法 -> get/set。

20. setter方法中参数名称和成员变量名称一致，不要在getter和setter方法中加业务逻辑。

21. 循环体内用StringBuilder的append方法进行扩展。

22. final可以修饰类，方法，变量。

23. 慎用Object的clone方法。

24. 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。

# 集合处理

1. hashCode和equals 的处理遵循以下规则：

1）只要覆写equals ，就必须要覆写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存储的对象必须覆写这两个方法。

3）如果自定义对象作为Map的键，那么必须覆写hashCode和equals。

2. ArrayList的subList结果不能强转ArrayList。

3. 使用map的keySet()、values()、entrySet()方法返回对象后不可以对其进行添加元素的操作。

4. Collections类返回的对象，如：emptyList()/singletonList()等都是immutablelist不可对其进行添加或者删除元素的操作。

5. 在subList场景中，高度注意对原集合元素的增加或删除，均会导致子列表的遍历、增加、删除产生ConcurrentModificationException 异常。

6. 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致、长度为0的空数组。

7. 在使用Collection接口任何实现类的addAll()方法时，一定要对输入的集合做NEP判断。

8. 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。

9. 泛型通配符<? extends T>来接收返回的数据，此写法的泛型集合不能使用add方法，而<? super T>不能使用get方法，作为接口调用赋值时易出错。

10. 在无泛型限制定义的集合赋值给泛型限制的集合时，在使用集合元素时，需要进行instanceof判断，避免抛出ClassCastException异常。

11. 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。

12. 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 会抛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。

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

14. 集合初始化时，指定集合初始值大小。

15. 使用entrySet 遍历Map 类集合KV，而不是keySet 方式进行遍历。

16. 高度注意Map类集合K/V能不能存储null值的情况。

17. 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。

18. 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。

# 并发处理

1. 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。
2. 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。
3. 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。
4. 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。
5. 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。
6. 必须回收自定义的ThreadLocal变量，尤其在线程池场景下，线程经常会被复用，如果不清理自定义的 ThreadLocal变量，可能会影响后续业务逻辑和造成内存泄露等问题。尽量在代理中使用try-finally块进行回收。
7. 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。
8. 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。
9. 在使用阻塞等待获取锁的方式中，必须在try代码块之外，并且在加锁方法与try代码块之间没有任何可能抛出异常的方法调用，避免加锁成功后，在finally中无法解锁。
10. 在使用尝试机制来获取锁的方式中，进入业务代码块之前，必须先判断当前线程是否持有锁。锁的释放规则与锁的阻塞等待方式相同。
11. 并发修改同一记录时，避免更新丢失，需要加锁。要么在应用层加锁，要么在缓存加锁，要么在数据库层使用乐观锁，使用version作为更新依据。
12. .多线程并行处理定时任务时，Timer运行多个TimeTask时，只要其中之一没有捕获抛出的异常，其它任务便会自动终止运行，如果在处理定时任务时使用ScheduledExecutorService则没有这个问题。
13. 资金相关的金融敏感信息，使用悲观锁策略。
14. .使用CountDownLatch进行异步转同步操作，每个线程退出前必须调用countDown方法，线程执行代码注意catch异常，确保countDown方法被执行到，避免主线程无法执行至await方法，直到超时才返回结果。
15. 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。
16. 在并发场景下，通过双重检查锁（double-checked locking）实现延迟初始化的优化问题隐患(可参考 The "Double-Checked Locking is Broken" Declaration)，推荐解决方案中较为简单一种（适用于JDK5及以上版本），将目标属性声明为 volatile型。
17. volatile解决多线程内存不可见问题。对于一写多读，是可以解决变量同步问题，但是如果多写，同样无法解决线程安全问题。
18. 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。
19. ThreadLocal对象使用static修饰，ThreadLocal无法解决共享对象的更新问题。

# 控制语句

1.在一个switch块内，每个case要么通过continue/break/return等来终止，要么注释说明程序将继续执行到哪一个case为止；在一个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。

2.当switch括号内的变量类型为String并且此变量为外部参数时，必须先进行null判断。

3.在if/else/for/while/do语句中必须使用大括号。

4.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。

5.表达异常的分支时，少用if-else方式。

6.除常用方法（如getXxx/isXxx）等外，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。

7.不要在其它表达式（尤其是条件表达式）中，插入赋值语句。

8.循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、获取数据库连接，进行不必要的try-catch操作（这个try-catch是否可以移至循环体外）。

9.避免采用取反逻辑运算符。

10.接口入参保护，这种场景常见的是用作批量操作的接口。

11.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。

12.下列情形，不需要进行参数校验：

1）极有可能被循环调用的方法。但在方法说明里必须注明外部参数检查要求。

2）底层调用频度比较高的方法。毕竟是像纯净水过滤的最后一道，参数错误不太可能到底层才会暴露问题。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。

# 注释规范

1. 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。
2. 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。
3. 所有的类都必须添加创建者和创建日期。
4. 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。
5. 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。
6. 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英文原文即可。
7. 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。
8. 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。
9. 对于注释的要求：第一、能够准确反映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。
10. 好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。
11. 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。

# 异常日志

## 异常处理

1.java类库中定义的可以通过预检查方式规避的RuntimeException异常不应该通过catch方式处理。如NullPointException、IndexOutOfBoundsException。

2. 异常不要用作流程控制、条件控制。

3. catch是要分清是稳定代码和非稳定代码，对于非稳定代码catch尽可能的按照异常类型分类。

4. 捕获异常一定要做处理，如果不想处理就抛给上层调用者。

5. 有try块放在事务中，catch异常后如果需要回滚事务，一定要注意手动回滚事务。

6. finally块中必须对资源对象、流对象进行关闭，有异常也要catch。

7. 不要在finally块中使用return。

8. 捕获的异常要和抛的异常匹配或者捕获的异常是抛异常的父类。

9. 在调用RPC、二方包、或动态生成类的相关方法时，捕获异常一定要用Throwable类拦截。

10. 方法的返回值可以是null，但是必须要说明什么情况返回null。

11. 防止NEP：

1. 返回类型是基本类型 ，return包装类型的对象。

2. 数据库查询的结果可能是null。

3. 集合里的元素即时isNotEmpty，取出来的元素也可能是null。

4. 远程调用返回对象时，必须要进行判空处理。

5. 对于Session中的数据要进行判空处理。

6. 级联调用有可能产生空指针。

12. 定义时区分unchecked / checked 异常，避免直接抛出new RuntimeException()，更不允许抛出Exception或者Throwable，应使用有业务含义的自定义异常。推荐业界已定义过的自定义异常，如：DAOException / ServiceException等。

13.对于公司外的http/api开放接口必须使用“错误码”；而应用内部推荐异常抛出；跨应用间RPC调用优先考虑使用Result方式，封装isSuccess()方法、“错误码”、“错误简短信息”。

14. 避免出现重复的代码（Don't Repeat Yourself），即DRY原则。

## 日志规范

1. 应用中不可直接使用日志系统（Log4j、Logback）中的API，而应依赖使用日志框架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。
2. 所有日志文件至少保存15天，因为有些异常具备以“周”为频次发生的特点。网络运行状态、安全相关信息、系统监测、管理后台操作、用户敏感操作需要留存相关的网络日志不少于6个月。
3. 应用中的扩展日志（如打点、临时监控、访问日志等）命名方式：appName\_logType\_logNamelog。logType:日志类型，如stats/monitor/access等；logName:日志描述。这种命名的好处：通过文件名就可知道日志文件属于什么应用，什么类型，什么目的，也有利于归类查找。
4. 在日志输出时，字符串变量之间的拼接使用占位符的方式。
5. 对于trace/debug/info级别的日志输出，必须进行日志级别的开关判断。
6. 避免重复打印日志，浪费磁盘空间，务必在log4jxml中设置additivity=false。
7. 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。
8. 谨慎地记录日志。生产环境禁止输出debug日志；有选择地输出info日志；如果使用warn来记录刚上线时的业务行为信息，一定要注意日志输出量的问题，避免把服务器磁盘撑爆，并记得及时删除这些观察日志。
9. 可以使用warn日志级别来记录用户输入参数错误的情况，避免用户投诉时，无所适从。如非必要，请不要在此场景打出error级别，避免频繁报警。
10. 尽量用英文来描述日志错误信息，如果日志中的错误信息用英文描述不清楚的话使用中文描述即可，否则容易产生歧义。【强制】国际化团队或海外部署的服务器由于字符集问题，使用全英文来注释和描述日志错误信息。

# 单元测试

1. 好的单元测试必须遵守AIR原则。
2. 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。
3. 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。
4. 单元测试是可以重复执行的，不能受到外界环境的影响。
5. 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。
6. 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。
7. 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。
8. 单元测试的基本目标：语句覆盖率达到70%；核心模块的语句覆盖率和分支覆盖率都要达到100%。
9. 编写单元测试代码遵守BCDE原则，以保证被测试模块的交付质量。
10. 对于数据库相关的查询，更新，删除等操作，不能假设数据库里的数据是存在的，或者直接操作数据库把数据插入进去，请使用程序插入或者导入数据的方式来准备数据。
11. 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。
12. 对于不可测的代码在适当的时机做必要的重构，使代码变得可测，避免为了达到测试要求而书写不规范测试代码。
13. 在设计评审阶段，开发人员需要和测试人员一起确定单元测试范围，单元测试最好覆盖所有测试用例。
14. 单元测试作为一种质量保障手段，在项目提测前完成单元测试，不建议项目发布后补充单元测试用例。
15. 为了更方便地进行单元测试，业务代码应避免以下情况：

1.构造方法中做的事情过多。

2. 存在过多的全局变量和静态方法。

3.存在过多的外部依赖。

4.存在过多的条件语句。

# 安全规范

1. 隶属于用户个人的页面或者功能必须进行权限控制校验。
2. 用户敏感数据禁止直接展示，必须对展示数据进行脱敏。
3. 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。
4. 用户请求传入的任何参数必须做有效性验证。
5. 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。
6. 表单、AJAX提交必须执行CSRF安全验证。
7. 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。
8. 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。

# 其他

1. 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。
2. velocity调用POJO类的属性时，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。
3. 后台输送给页面的变量必须加$!{var}——中间的感叹号。
4. 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。
5. 获取当前毫秒数SystemcurrentTimeMillis(); 而不是new Date().getTime();。
6. 日期格式化时，传入pattern中表示年份统一使用小写的y。
7. 不要在视图模板中加入任何复杂的逻辑。
8. 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。
9. 及时清理不再使用的代码段或配置信息。