Java 语言编程规范

1 排版规范

1.1 强制要求

1. 代码缩进必须使用 tab，且设置 tab 为 4 个空格。

2. 分界符（如大括号‘{’和‘}’）左括号不用起新一行，右括号新起一行，同时与引用它们的语句左对齐。在函数体的开始、类和接口的定义、以及 if、for、do、while、switch、case 语句中的程序都要采用如上的缩进方式。

3. if/for/while/switch/do 等保留字与左右括号之间都必须加空格。

4. 不允许把多个短语句写在一行中，即一行只写一条语句。

5. 相对独立的程序块之间、变量说明之后必须加空行。

6. 在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如：. ），后不应加空格。

   说明：采用这种松散方式编写代码的目的是使代码更加清晰。

7. 单行字符数限制不超过 120 个，超出需要换行。

8. 方法参数在定义和传入时，多个参数逗号后边必须加空格。

1.2 推荐使用

1. 类属性和类方法不要交叉放置，不同存取范围的属性或者方法也尽量不要交叉放置。

2. 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。

3. 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。

2 注释规范

2.1 强制要求

1. 一般情况下，源程序有效注释量必须在 30%以上。

   说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。可以用注释统计工具来统计。

2. 所有的类都必须添加创建者信息。

3. 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，提高代码可读性。

4. 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，还必须指明该方法做什么事情，实现什么功能。

5. 方法内部单行注释，使用//注释。注释要和上方代码使用空行分隔，方法内多行注释使用/* xxx */注释，注意注释要和内容同样进行缩进。

6. 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。

7. 对变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

   说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

8. 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

2.2 推荐使用

1. 注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用中文，除非能用非常流利准确的英文表达。

   说明：注释语言不统一，影响程序易读性和外观排版，出于对维护人员的考虑，建议使用中文。

2. 通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。

   说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

3. 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。2）永久不用。前者如果没有注释信息，难以知晓其注释动机，后者建议删除。

4. 顺序实现流程的说明使用 1、2、3、4 在每个实现步骤部分的代码前面进行注释。

   示例：如下是对设置属性的流程注释

   ```
   //1、 判断输入参数是否有效。
   。。。。。
   // 2、设置本地变量。
   ```

。。。。。。

5. 一些复杂的代码需要说明。

6. 避免在一行代码或表达式的中间插入注释。

7. 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，

   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，但是目前还未实现的功能。这是一个 javadoc 标签。

   2) 错误（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。

3 命名规范

3.1 强制要求

1. 所有编程相关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。

   反例： _name / __name / $Object / name_ / name$ / Object$

2. 类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外：（领域模型的相关命名）DO / DTO / VO / DAO 等。

   正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

   反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

3. 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. 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。枚举类中的变量同此命名规则，因为 枚举类是特殊的常量类，并且构造方法被默认强制是私有。

   正例： MAX_STOCK_COUNT 反例： MAX_COUNT

5. 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。

6. 中括号是数组的一部分 String[] args。

   例如：数组应定义为 String[] args

7. POJO 类中的任何布尔类型的变量 ,都不要加 is ,否则部分框架解析会引起序列化错误。

   例如：定义为基本数据类型 boolean isSuccess;的属性，他的方法也是 isSuccess()，rpc 框架在反向解析式，“以为”对应的属性名是 success，导致属性获取不到，进而抛出异常。

8. 包名统一使用小写，点分隔符之间有且仅有一个自然予以的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以用复数的形式。

9. 杜绝完全不规范的缩写，避免看到缩写后不知道它们的意思

10. 不允许出现任何未经定义的常量直接出现在代码中

    反例： String key=“id#hgdk” +tradeId；cache.put(key，value)

11. 接口和实现类命名的规范：对于 service 和 DAO 类，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。

    例如：HelloServiceImpl 是 HelloService 接口的实现。

12. long 或者 Long 类型赋值时，必须要用大些的 L，如果小写的 l 容易以数字 1 混淆

3.2 推荐使用

1. 各层的命名规则建议如下：

   （1）获取单个对象的方法用 get 作为前缀；

   （2）获取多个对象的方法用 list 作为前缀

   （3）获取统计值的方法用 count 作为前缀

   （4）插入的方法用 save 或者 insert 作为前缀

   （5）插入方法用 save 或 insert

   （6）删除方法用 remove 或 delete

   （7）修改的方法用 update 作为前缀

2. 如果使用了设计模式，建议在类名中体现出具体的模式

   例如： public class OrderFactory 使用了工厂模式。

3. 含有集合意义的属性命名，尽量包含其复数的意义。

   示例：customers, orderItems

4. 常用组件类的命名以组件名加上组件类型名结尾。

   示例：

   Application 类型的，命名以 App 结尾——MainApp

   Frame 类型的，命名以 Frame 结尾——TopoFrame

   Panel 类型的，建议命名以 Panel 结尾——CreateCircuitPanel

   Bean 类型的，建议命名以 Bean 结尾——DataAccessBean

   EJB 类型的，建议命名以 EJB 结尾——DBProxyEJB

   Applet 类型的，建议命名以 Applet 结尾——PictureShowApple

5. 如果函数名超过 15 个字母，可采用以去掉元音字母的方法或者以行业内约定俗成的缩写方式缩写函数名。

   示例：*getCustomerInformation() 改为 getCustomerInfo()*

4 编程规范

4.1 强制要求

1. 明确类的功能，精确（而不是近似）地实现类的设计。一个类仅实现一组相近的功能。

   说明：划分类的时候，应该尽量把逻辑处理、数据和显示分离，实现类功能的单一性。

2. 明确方法功能，精确实现方法设计。一个函数仅完成一件功能，即使简单功能也应该编写方法实现。

   说明：虽然为仅用一两行就可完成的功能去编方法好象没有必要，但用方法可使功能明确化，增加程序可读性，亦可方便维护、测试。

3. 明确规定对接口方法参数的合法性检查应由方法的调用者负责还是由接口方法本身负责

4. 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，无谓增加编译器的解析成本，直接使用类名访问。

5. 所有的覆写方法，必须加@Override 注解，不要覆盖父类的静态方法和私有方法，不要覆盖父类的属性。

6. 不要使用两级以上的内部类，不要把内部类定义成私有类。

7. 不要定义不会被用到的局部变量、类私有属性、类私有方法和方法参数。

8. 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：objet.equals(“test”);

9. 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。

10. 构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。

11. 实体类必须写 toString 方法。使用工具类 source> generate toString 时，如果继承了另一个类，注意在前面加一下 super.toString。

    说明：在方法执行抛出异常时，可以直接调用 toString()方法打印其属性值，便于

排查问题。

12. 循环集合不要在 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。

13. 获取单例对象要线程安全。在单例对象里面做操作也要保证线程安全。

14. 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。

15. 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，就不需要用锁，能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。

16. 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，否则可能会进行死锁

    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 顺序。

17. 并发修改同一记录时，避免更新丢失，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。

18. 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。

19. 在一个 switch 块内，每个 case 要么通过 break/return 来终止。

20. switch 语句中的 case 关键字要和后面的常量保持一个空格，switch 语句中不要定义 case 之外的无用标签。

21. 在 if/else/for/while/do 语句中必须使用大括号，即使只有一行代码，避免使用下面形式：if（condition）statements；

22. 不要在 if 语句中使用等号 = 进行赋值操作。

23. 不要使用空的 for 、if 、while 语句。

24. 减小单个方法的复杂度，使用的 if, while, for, switch 语句要在 10 个以内。

25. 不要硬编码 ‘\n’ 和 ‘\r’ 作为换行符号。

26. 注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。

    说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。

27. 在运算中不要减小数据的精度。

28. 数组声明的时候使用 int[] index ，而不要使用 int index[] 。

    说明：使用 int index[] 格式使程序的可读性较差。

29. 使用 System.arraycopy() ，不使用循环来复制数组。

30. 获取当前毫秒数：System.currentTimeMillis(); 而不是 new Date().getTime()。

31. 后台输送给页面的变量必须加$!{var}——中间的感叹号。

    说明： 如果 var = null 或者不存在，那么${var}会直接显示在页面上。

32. 不要在 finalize() 方法中删除监听器（Listeners）。

33. 在 finalize() 方法中的 finally 中调用 super.finalize() 方法。

34. Math.random() 这个方法返回是 double 类型，注意取值范围 0≤x<1（可以取到 0 值，注意除 0 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。

4.2 推荐使用

1. 当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起，便于阅读。

2. 类中方法定义顺序依次是：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。

3. 不要重复调用一个方法获取对象，使用局部变量重用对象。

4. 如果多段代码重复做同一件事情，那么在方法的划分上可能存在问题。

   说明：若此段代码各语句之间有实质性关联并且是完成同一件功能的，那么可考虑把此段代码构造成一个新的方法。

5. 源程序中关系较为紧密的代码应尽可能相邻。

   说明：便于程序阅读和查找。

6. 不要使用难懂的技巧性很高的语句，除非很有必要时。

   说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。

7. 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) 若是 static 成员变量，必须考虑是否为 final。
   6) 类成员方法只供类内部调用，必须是 private。
   7) 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。过宽泛的访问范围，不利于模块解耦。

8. final 可以提高程序响应效率，声明成 final 的情况：

   1) 不需要重新赋值的变量，包括类属性、局部变量。
   2) 对象参数前加 final，表示不允许修改引用的指向。
   3) 类方法确定不允许被重写。

9. 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。

10. 集合中的数据如果不使用了应该及时释放，尤其是可重复使用的集合。

    说明：由于集合保存了对象的句柄，虚拟机的垃圾收集器就不会回收。

11. 使用 entrySet 遍历 Map 集合的 key、value，而不是 keySet 方法遍历。

12. 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：稳定性指集合每次遍历的元素次序是一定的。有序性是指遍历结果按照某种比较规则依次排序。

13. 利用 Set 元素唯一的特性，可以快速对另一个集合进行去重操作，避免使用 List 的 contains 方法进行遍历去重操作。

14. 注意 HashMap 的扩容死链，导致 CPU 飙升的问题。

15. 线程中需要实现 run() 方法。

16. 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 方法可以执行，避免主线程无法执行至 countDown 方法，直到超时才返回结果。

    说明：子线程抛出异常堆栈，不能在主线程 try-catch 到。

17. 避免 Random 实例被多线程使用，虽然共享该实例是线程安全的，但是会竞争同一 seed 导致性能下降。

18. 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有的，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（在线程内定义的）都可以操作这个变量。

19. 尽量少用 else， if-else 的方式可以改写成 if 语句。

20. 不要在条件判断中执行复杂的语句，以提高可读性。

21. 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，获取数据库的连接，镜像不必要的 try-catch 操作（可以考虑移至循环体外）。

22. 循环体内，字符串的联接方式，使用 StringBuilder 的 append 方法进行扩展。

23. 对于“明确停止使用的代码和配置”，如方法、变量、类、配置文件、动态配置属性坚决从程序中清理出去，避免造成过多垃圾。

24. 不要对浮点数进行比较运算，尤其是不要进行 ==, !=运算，减少 >, < 运算。

25. 复杂度：建议的最大规模：

    继承层次　　5 层

    类的行数　　1000 行（包含{}）

    类的属性　　10 个

类的方法　　20 个

类友好方法　10 个

类私有方法　15 个

类保护方法　10 个

类公有方法　10 个

类调用方法　20 个

方法参数　　5 个

return 语句　1 个

方法行数　　30 行

方法代码　　20 行

注释比率　　30%~50%

5 异常处理

5.1 强制要求

1. 不要捕获 Java 类库中定义继承自 RuntimeException 的运行时异常类，如 IndexOutOfBoundsException / NullPointerException，这类异常由程序员预检查来规避，保证程序健壮性。

   正例：if(obj != null){...}

   反例：try{　obj.method()　} catch(NullPointerException e) {...}

2. 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。

3. 对大段代码进行 try-catch 这是不负责任的表现。catch 时需要分清稳定代码和非稳定性代码，稳定代码就是无论如何都不会出错的代码，对于非稳定性代码需要区分异常的类型，再做对应的异常处理。

4. 捕获异常与抛异常，必须是完全匹配，或者捕获异常是抛异常的父类。

5. 捕获异常是为了处理它，不要捕获之后不处理就抛弃，如果不想处理将她抛给调用者，最外层的业务使用者，必须处理异常，将它转换成用户可以理解的内容。

6. 自己抛出的异常必须要填写详细的描述信息。便于问题定位。

   示例：throw new　IOException("Writing data error! Data: " +

   data.toString());

7. 有 try 块放置在事务中，catch 异常后，如果需要事务回滚，注意一定要手动回滚事务。

8. finally 一定要对资源对象、流对象进行关闭，有异常也要 try-cach。

5.2 推荐使用

1. 在捕获异常的时候，不使用 Exception, RuntimeException, Throwable，尽可能使用它们的子类。

2. 在程序中使用异常处理还是使用错误返回码处理，根据是否有利于程序结构来确定，并且异常和错误码不应该混合使用，推荐使用异常。

3. 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。

4. 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。

        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，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。