# 编写规范问题文档

1.书写格式规范要求较少，位置:第5页java书写格式规范中。

建议:

添加

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();

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.oop规约不够全面，第15页oop规约中

建议:

添加

7.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。

8. 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。

9.定义 DO/DTO/VO 等 POJO 类时，不要设定任何属性默认值。

10.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。

11.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。

1. POJO 类必须写 toString 方法。使用 IDE 中的工具：source> generate toString 时，如果继承了另一个 POJO 类，注意在前面加一下 super.toString。
2. 禁止在 POJO 类中，同时存在对应属性 xxx 的 isXxx()和 getXxx()方法。

3.文档中缺少对异常处理的规范

建议:

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 的方式来处理，比如：NullPointerException，IndexOutOfBoundsException 等等。

说明：无法通过预检查的异常除外，比如，在解析字符串形式的数字时，不得不通过 catch

NumberFormatException 来实现。

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。

说明：异常设计的初衷是解决程序运行中的各种意外情况，且异常的处理效率比条件判断方式

要低很多。

3. 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。

4. 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。

5. 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。

6. finally 块必须对资源对象、流对象进行关闭，有异常也要做 try-catch。

说明：如果 JDK7 及以上，可以使用 try-with-resources 方式。

7. 不要在 finally 块中使用 return。

说明：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。

8. 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。

说明：如果预期对方抛的是绣球，实际接到的是铅球，就会产生意外情况。

4.文档缺少集合处理规约

建议

补充集合处理规约

1. 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） 如果自定义对象作为 Map 的键，那么必须重写 hashCode 和 equals。

说明：String 重写了 hashCode 和 equals 方法，所以我们可以非常愉快地使用 String 对象

作为 key 来使用。

2. ArrayList的subList结果不可强转成ArrayList，否则会抛出ClassCastException

异常，即 java.util.RandomAccessSubList cannot be cast to java.util.ArrayList。

说明：subList 返回的是 ArrayList 的内部类 SubList，并不是 ArrayList 而是 ArrayList

的一个视图，对于 SubList 子列表的所有操作最终会反映到原列表上。

3. 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。阿里巴巴 Java 开发手册

10/38

4. 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 list = new ArrayList(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。

5. 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。

6. 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，而不能使用 get 方法，作为接口调用赋值时易出错。

说明：扩展说一下 PECS(Producer Extends Consumer Super)原则：第一、频繁往外读取内

容的，适合用extends T>。第二、经常往里插入的，适合用。

7. 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 list = new ArrayList<>();

list.add("1");

list.add("2");

Iterator 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);

}

}

8. 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() {

@Override

public int compare(Student o1, Student o2) {

return o1.getId() > o2.getId() ? 1 : -1;

}

};

9. 【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 users = new ArrayList(10);

5.文档缺少并发处理规约

建议:

补充并发处理规约

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。

说明：资源驱动类、工具类、单例工厂类都需要注意。

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。

正例：

public class TimerTaskThread extends Thread {

public TimerTaskThread() {

super.setName("TimerTaskThread");

...

}

}

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。

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。阿里巴巴 Java 开发手册

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。

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. 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 df = new ThreadLocal() {

@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. 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. 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. 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。

说明：如果每次访问冲突概率小于 20%，推荐使用乐观锁，否则使用悲观锁。乐观锁的重试次

数不得小于 3 次。

9. 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，使用 ScheduledExecutorService 则没有这个问题。

10. 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。

说明：注意，子线程抛出异常堆栈，不能在主线程 try-catch 到。

6.Java语句规范不是很全面，第11页 5.2复合语句

建议

补充

1.在一个 switch 块内，每个 case 要么通过 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，即使空代码。

2. 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。

反例：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。