面向用户体验与接口编程

/**

* Created by Lynch on 2017/08

* 源码版本控制

* Java 编码

* (xml) UI 编码

*todo one more thing。。。不放过任何（技术）问题，自己解决要限定一个时间，超时就抛给团队，并做好问题记录以汇总交流

* 参考：阿里巴巴 Java 开发手册

*/

/* **(xml) UI 编码**：

【】完全遵循设计图上的尺寸，比例，颜色

【】避免所有硬编码资源(Java 代码尽量使用常量)，占位符尽量官方且统一

【】善用 tools(xmlns:tools="http://schemas.android.com/tools")；<Space>，<merge>，<include>，<ViewStub> 等

【】动态检测：TextView 系的 text 加长，visibility，ImageView 的 scaleType， 预测总高度超屏否等

【】资源命名不能按单位命名：反例（ <dimen name="sp_10">10sp</dimen> ）

*/

/*源码版本控制：

update -》 resolve -》 add -》 commit -》 update -》 resolve -》

* 频率：酌情日更或周更

* 提交条件：格式化，去冗余（如：Android Studio Lint 工具）； 过测试

* 必要时建立多个本地备份（暂不与主项目同步），比如实验或后置功能

* */

/* Java 编码：

【】IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式，compileSdkVersion ，targetSdkVersion 尽量用最新

【】尽量不使用 deprecated 的类或方法或变量。

【】第三方库使用与管理：再封装； 版本固定. 不要增加功能相同的库.

【】所有的覆盖方法必须加@Override 注解， 方法过时必须加@Deprecated 注解；

【】类，方法，变量，参数名尽量能使顾名思义（不嫌长）

【】long 或 Long 型赋值时，必须使用大写 L，小写容易跟数字 1 混淆，造成误解：

Long a = 2L(正) Long a = 2l(误)

【】switch 块内必须包含一个 default 语句并且放在最后； if(){}else{} 大括号必写。

【】方法体内先验参；

【】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。

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。

【】序列化类新增属性时，请不要修改 serialVersionUID 字段值，避免反序列化异常；若完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。

【*注释*】要能够准确反应设计思想和代码逻辑；要能够描述业务含义。

0. 【】所有的抽象方法（包括接口中的方法）必须要用 Javadoc 注释、不仅是返回值、参数、异常说明，还必须指出该方法做了什么，也可以加上设计初衷（为什么）。对子类的实

现要求，或者调用注意事项，请一并说明。所有的类都必须添加创建者信息。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。

1. 【】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。反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。

2. 【】代码修改的同时，注释也要进行相应的修改。

3. 【】被注释掉的代码要配合说明：代码被注释掉有两种可能性：1）后续可能会恢复此段代码逻辑。2）永久不用。因为前者如果没有备注信息，难以知晓注释动机；后者建议直接删掉（代码仓库保存了历史代码）。

4. 【】特殊注释标记，请注明标记人与标记时间。注意经常清理此类标记：

1（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。只能应用于类，接口和方法（因为它是一个 Javadoc 标签）。

2（FIXME）：（标记人，标记时间，[预计处理时间]）

标记某代码是需要被及时纠正的。

【*日志*】（release/debug)

1.【】异常信息应该包括：案发现场信息和异常堆栈信息，如果不处理，那么往上抛。例：logger.error(各类参数或者对象 toString + "_" + e.getMessage(), e);

2.【】可以使用 warn 日志级别来记录用户输入参数错误的情况。注意日志输出的级别，error 级别只记录系统逻辑出错、异常等重要的错误信息。

3.【】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。记录日志时请思考：这些日志真的有人看吗？看到这条日志你能做什么？能不能给问题排查带来好处？

【*异常*】（ release/debug)

0. 【】不要捕获 Java 类库中定义的继承自 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){...}

1. 【】异常尽量不要用来做流程控制，条件控制，注意下层有 throw 才轮得到 try-catch。

2. 【】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，并将其转化为用户可以理解的内容。

3. 【】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。

4. 【】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：JDK7 可以使用 try-with-resources 方式。

5. 【】不要在 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。

6. 【】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，就邪恶了。

7. 【】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。

说明：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，运行时异常等场景返回 null。eg：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，一律要求判空。

5） 对于 Session 中获取的数据，建议 NPE 检查。

6） 级联调用 obj.getA().getB().getC() 易产生 NPE。

8. 【】在代码中使用“抛异常”还是“返回错误码”，直接面向用户的接口必须“说人话”。

9. 【】定义时区分 unchecked / checked 异常，避免直接使用 RuntimeException 抛出，更不允许抛出 Exception 或者 Throwable ，应使用有业务含义的自定义异常。

【*集合*】

0. 【】ArrayList 的 subList 方法返回值不是 ArrayList ，否则 ClassCastException;

1. 【】在 subList 场景中，高度注意对原集合元素个数的修改，会导致子列表的遍历、增加、删除均产生 ConcurrentModificationException 异常。

2. 【】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()。

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。

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);
```

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。

3. 【】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 。

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[] { "a", "b" };
List list = Arrays.asList(str);
```

第一种情况：list.add("c"); 运行时异常；

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

4. 【】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。

说明：苹果装箱后返回一个<? extends Fruits>对象，此对象就不能往里加任何水果，包括苹果。

5. 【】建议不要在 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。

反例：List<String> a = new ArrayList<String>();

```
a.add("1");
a.add("2");
for (String temp : a) {
if("1".equals(temp)){
a.remove(temp);}
}
```

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

正例：Iterator<String> it = a.iterator();

```
while(it.hasNext()){
```

```
String temp = it.next();
if(删除元素的条件){ it.remove();}
}
```

6. 【】在 JDK7 版本以上，Comparator 要满足自反性，传递性，对称性，不然 Arrays.sort ，Collections.sort 会报 IllegalArgumentException 。

说明：1） 自反性：x，y 的比较结果和 y，x 的比较结果相反。

2）传递性：x>y,y>z,则 x>z。

3）对称性：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;
}}
```

7. 【】集合初始化时，尽量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 loaderfactor ）默认为 0.75，如果暂时无法确定初始值大小，请设置为 16。

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。

8. 【】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。

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

集合类	Key	Value	Super	说明
Hashtable	不允许 null	不允许 null	Dictionary	线程安全
ConcurrentHashMap	不允许 null	不允许 null	AbstractMap	分段锁
TreeMap	不允许 null	允许 null	AbstractMap	线程不安全
HashMap	允许 null	允许 null	AbstractMap	线程不安全

10. 【】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。有序性是指遍历的结果是按某种比较规则依次排列的。如：ArrayList 是 order/unsort ；HashMap 是 unorder/unsort ；TreeSet 是 order/sort 。

11. 【】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。

【*线程*】

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。

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，不建议在应用中自行显式创建线程。线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式。

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。

4. 【】不建议用 Timer，因为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。

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 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 Simpledateformatter.

【】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或用 ary[ary.length-1]，否则会有抛 IndexOutOfBoundsException 的风险。

说明：String[] ary = "a,b,c,,".split(",");// 预期长度大于 3，结果是 3

【】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);

【】数学运算(精度)：

建议先乘后除;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。

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

说明：更加精确的纳秒级用 System.nanoTime() 。在 JDK8 中，针对统计时间等场景，推荐使用 Instant 类。

【】权限：（非(23-)）/运行时(23+ 26-; 26+): 痛啊，定制过权限系统的 ROM：

【】四大组件等

充分协同生命周期回调处理数据与交互

Activity
Fragment

Service(IntentService)

ContentProvider

Broadcast() ：进程内消息尽量使用 LocalBroadcast

【】注意内存泄露：因为被引用者与引用者生命周期不统一。eg:Activity 被静态量强引用，Activity 非静态内部类长期强引用 Activity ，非静态 Handler 实例处理消息延迟到 Activity 被 destroy 后等

【】ANR：因为 UI 线程超时响应

【】可以只给 arm64-v8a 。armeabi-v7a 与 arm64-v8a 都兼容 armeabi，而市面上的 x86 机器，也基本上都可以使用 intel 的 libhounini 项目直接在 x86 机器上运行仅含 armeabi 的动态库代码。如果不是为了性能，只需要提供 armeabi ，armeabi 不支持硬件浮点，所以微信还自带的部分的 v7a 版本库，用于动态加载。但是，如果你只要提供一个可执行的二进制文件，则必须要提供所有的版本。某些 vivo 的机器上，虽然是 64 位的机器，但是只认 32 位的动态库（即使给了 64 位的库）

*/

/* **其他**

【*安全*】

1. 【】Manifest.xml 里 android:allowBackup="false"　;

2. 【】四大组件的 exported 属性；

*/