* [1 前言](https://github.com/Blankj/AndroidStandardDevelop" \l "1-%E5%89%8D%E8%A8%80)
* [2 AS规范](https://github.com/Blankj/AndroidStandardDevelop#2-as%E8%A7%84%E8%8C%83)
* [3 命名规范](https://github.com/Blankj/AndroidStandardDevelop#3-%E5%91%BD%E5%90%8D%E8%A7%84%E8%8C%83)
* [4 资源文件规范](https://github.com/Blankj/AndroidStandardDevelop#4-%E8%B5%84%E6%BA%90%E6%96%87%E4%BB%B6%E8%A7%84%E8%8C%83)
* [5 版本统一规范](https://github.com/Blankj/AndroidStandardDevelop#5-%E7%89%88%E6%9C%AC%E7%BB%9F%E4%B8%80%E8%A7%84%E8%8C%83)
* [6 第三方库规范](https://github.com/Blankj/AndroidStandardDevelop#6-%E7%AC%AC%E4%B8%89%E6%96%B9%E5%BA%93%E8%A7%84%E8%8C%83)
* [7 注释规范](https://github.com/Blankj/AndroidStandardDevelop#7-%E6%B3%A8%E9%87%8A%E8%A7%84%E8%8C%83)
* [8 测试规范](https://github.com/Blankj/AndroidStandardDevelop#8-%E6%B5%8B%E8%AF%95%E8%A7%84%E8%8C%83)
* [9 RN规范](https://github.com/Blankj/AndroidStandardDevelop#9-RN%E8%A7%84%E8%8C%83)
* [10 其他的一些规范](https://github.com/Blankj/AndroidStandardDevelop#10-%E5%85%B6%E4%BB%96%E7%9A%84%E4%B8%80%E4%BA%9B%E8%A7%84%E8%8C%83)

**1 前言**

为了利于项目维护以及规范开发，促进成员之间Code Review的效率，故提出以下开发规范，如有更好建议，欢迎到GitHub提issue，原文地址： [**安卓开发规范(updating)**](https://github.com/Blankj/AndroidStandardDevelop)

**2 AS规范**

工欲善其事，必先利其器。

1. 尽量使用最新版的IDE进行开发；
2. 编码格式统一为**UTF-8**；
3. 编辑完.java, .xml等文件后一定要**格式化**（基本格式方面使用 AS 默认模板即可）；
4. 删除多余的import，减少警告出现，可利用AS的Optimize Imports(Settings → Keymap → Optimize Imports)快捷键；
5. AS常用开发插件可以参考这里～**[AS常用开发插件](http://www.jianshu.com/p/c76b0d8a642d)**

**3 命名规范**

代码中的命名严禁使用拼音与英文混合的方式,更不允许直接使用中文的方式。正确的英文拼写和语法可以让阅读者易于理解,避免歧义。

注意：即使纯拼音命名方式也要避免采用。但alibaba、taobao、youku、hangzhou等国际通用的名称,可视同英文。

**3.1 包名**

包名全部小写，连续的单词只是简单地连接起来，不使用下划线。

采用反域名命名规则，全部使用小写字母。一级包名是顶级域名，通常为com,edu,gov,net,org等，二级包名为公司名，三级包名根据应用进行命名，四级包名为模块名或层级名。

| **包名** | **此包中包含** |
| --- | --- |
| com.xx.应用名称缩写.activity | 用户界面中所有的Activity类 |
| com.xx.应用名称缩写.fragment | 界面中所有的Fragment类 |
| com.xx.应用名称缩写.base | 基础共享的类 |
| com.xx.应用名称缩写.adapter | 页面用到的Adapter类 (适配器的类) |
| com.xx.应用名称缩写.view | 自定义的View类 |
| com.xx.应用名称缩写.util | 此包中包含：公共工具方法类（util模块名） |
| com.xx.应用名称缩写.bean | 下面可分：vo、po、dto 此包中包含：JavaBean类 |
| com.xx.应用名称缩写.model | 此包中包含：模型类 |
| com.xx.应用名称缩写.db | 数据库操作类 |
| com.xx.应用名称缩写.view (或者 com.xx.应用名称缩写.widget ) | 自定义的View类等 |
| com.xx.应用名称缩写.service | Service服务 |
| com.xx.应用名称缩写.receiver | BroadcastReceiver服务 |
| com.xx.应用名称缩写.config | 所有的配置相关的类 |

注意：如果项目采用MVP，所有M、V、P抽取出来的接口都放置在相应模块的i包下，所有的实现都放置在相应模块的impl下

**3.2 类名**

类名都以UpperCamelCase风格编写。

类名通常是名词或名词短语，接口名称有时可能是形容词或形容词短语。现在还没有特定的规则或行之有效的约定来命名注解类型。

名词，采用大驼峰命名法，尽量避免缩写，除非该缩写是众所周知的， 比如HTML, URL，如果类名称中包含单词缩写，则单词缩写的每个字母均应大写。

| **类** | **描述** | **例如** |
| --- | --- | --- |
| Activity 类 | Activity为后缀标识 | 欢迎页面类WelcomeActivity |
| Adapter类 | Adapter 为后缀标识 | 新闻详情适配器 NewDetailAdapter |
| 解析类 | Parser为后缀标识 | 首页解析类HomePosterParser |
| 工具方法类 | Utils或Manager为后缀标识（与系统或第三方的Utils区分）或功能+Utils | 线程池管理类：ThreadPoolManager日志工具类：LogUtils（Logger也可）打印工具类：PrinterUtils |
| 数据库类 | 以DBHelper后缀标识 | 新闻数据库：NewDBHelper |
| Service类 | 以Service为后缀标识 | 时间服务TimeService |
| BroadcastReceiver类 | 以Receiver为后缀标识 | 推送接收JPushReceiver |
| ContentProvider类 | 以Provider为后缀标识 | ShareProvider |
| 自定义的共享基础类 | 以Base开头 | BaseActivity,BaseFragment |

测试类的命名以它要测试的类的名称开始，以Test结束。例如：HashTest或HashIntegrationTest。

接口（interface）：命名规则与类一样采用大驼峰命名法，多以able或ible结尾，如 interface Runnable、interface Accessible。

注意：如果项目采用MVP，所有Model、View、Presenter的接口都以I为前缀，不加后缀，其他的接口采用上述命名规则。

**3.3 方法名**

方法名都以lowerCamelCase风格编写。

方法名通常是动词或动词短语。

| **方法** | **说明** |
| --- | --- |
| initXX() | 初始化相关方法,使用init为前缀标识，如初始化布局initView() |
| isXX() checkXX() | 方法返回值为boolean型的请使用is或check为前缀标识 |
| getXX() | 返回某个值的方法，使用get为前缀标识 |
| setXX() | 设置某个属性值 |
| handleXX()/processXX() | 对数据进行处理的方法 |
| displayXX()/showXX() | 弹出提示框和提示信息，使用display/show为前缀标识 |
| updateXX() | 更新数据 |
| saveXX() | 保存数据 |
| resetXX() | 重置数据 |
| clearXX() | 清除数据 |
| removeXX() | 移除数据或者视图等，如removeView(); |
| drawXX() | 绘制数据或效果相关的，使用draw前缀标识 |

**3.4 常量名**

常量名命名模式为CONSTANT\_CASE，全部字母大写，用下划线分隔单词。那，到底什么算是一个常量？

每个常量都是一个静态final字段，但不是所有静态final字段都是常量。在决定一个字段是否是一个常量时，考虑它是否真的感觉像是一个常量。例如，如果任何一个该实例的观测状态是可变的，则它几乎肯定不会是一个常量。只是永远不打算改变对象一般是不够的，它要真的一直不变才能将它示为常量。

// Constants

static final int NUMBER = 5;

static final ImmutableListNAMES = ImmutableList.of("Ed", "Ann");

static final Joiner COMMA\_JOINER = Joiner.on(','); // because Joiner is immutable

static final SomeMutableType[] EMPTY\_ARRAY = {};

enum SomeEnum { ENUM\_CONSTANT }

// Not constants

static String nonFinal = "non-final";

final String nonStatic = "non-static";

static final SetmutableCollection = new HashSet();

static final ImmutableSetmutableElements = ImmutableSet.of(mutable);

static final Logger logger = Logger.getLogger(MyClass.getName());

static final String[] nonEmptyArray = {"these", "can", "change"};

**3.5 非常量字段名**

非常量字段名以lowerCamelCase风格的基础上改造为如下风格：基本结构为scopeVariableNameType。

**scope：范围**

非公有，非静态字段命名以m开头。

静态字段命名以s开头。

公有非静态字段命名以p开头。

公有静态字段（全局变量）命名以g开头。

例子：

public class MyClass {

int mPackagePrivate;

private int mPrivate;

protected int mProtected;

private static MyClass sSingleton;

public int pField;

public static int gField;

}

使用1字符前缀来表示作用范围，1个字符的前缀必须小写，前缀后面是由表意性强的一个单词或多个单词组成的名字，而且每个单词的首写字母大写，其它字母小写，这样保证了对变量名能够进行正确的断句。

**Type：类型**

考虑到Android中使用很多UI控件，为避免控件和普通成员变量混淆以及更好达意，所有用来表示控件的成员变量统一加上控件缩写作为后缀（文末附有缩写表）。

对于普通变量一般不添加类型后缀，如果统一添加类型后缀，请参考文末的缩写表。

用统一的量词通过在结尾处放置一个量词，就可创建更加统一的变量，它们更容易理解，也更容易搜索。

注意：如果项目中使用ButterKnife，则不添加m前缀，以lowerCamelCase风格命名。

例如，请使用mCustomerStrFirst和mCustomerStrLast，而不要使用mFirstCustomerStr和mLastCustomerStr。

| **量词列表** | **量词后缀说明** |
| --- | --- |
| First | 一组变量中的第一个 |
| Last | 一组变量中的最后一个 |
| Next | 一组变量中的下一个变量 |
| Prev | 一组变量中的上一个 |
| Cur | 一组变量中的当前变量 |

说明：

集合添加如下后缀：List、Map、Set 数组添加如下后缀：Arr

注意：所有的VO（值对象）统一采用标准的lowerCamelCase风格编写，所有的DTO（数据传输对象）就按照接口文档中定义的字段名编写。

**3.6 参数名**

参数名以lowerCamelCase风格编写。 参数应该避免用单个字符命名。

**3.7 局部变量名**

局部变量名以lowerCamelCase风格编写，比起其它类型的名称，局部变量名可以有更为宽松的缩写。

虽然缩写更宽松，但还是要避免用单字符进行命名，除了临时变量和循环变量。

即使局部变量是final和不可改变的，也不应该把它示为常量，自然也不能用常量的规则去命名它。

**3.8 临时变量**

临时变量通常被取名为i、j、k、m和n，它们一般用于整型；c、d、e，它们一般用于字符型。 如：for (int i = 0; i < len ; i++)。

**3.9 类型变量名**

类型变量可用以下两种风格之一进行命名：

1. 单个的大写字母，后面可以跟一个数字(如：E, T, X, T2)。
2. 以类命名方式(参考[3.2 类名](https://github.com/Blankj/AndroidStandardDevelop" \l "32-%E7%B1%BB%E5%90%8D))，后面加个大写的T(如：RequestT, FooBarT)。

更多还可参考～[**阿里巴巴Java开发手册**](https://102.alibaba.com/newsInfo.htm?newsId=6)

**4 资源文件规范**

**4.1 资源布局文件（XML文件（layout布局文件））**

全部小写，采用下划线命名法

**4.1.1 contentView命名**

必须以全部单词小写，单词间以下划线分割，使用名词或名词词组。

所有Activity或Fragment的contentView必须与其类名对应，对应规则为：将所有字母都转为小写，将类型和功能调换（也就是后缀变前缀）。

例如：activity\_main.xml

**4.1.2 Dialog命名**

规则：dialog\_描述.xml

例如：dialog\_hint.xml

**4.1.3 PopupWindow命名**

规则：ppw\_描述.xml

例如：ppw\_info.xml

**4.1.4 列表项命名**

规则：item\_描述.xml

例如：item\_city.xml

**4.1.5 包含项命名**

规则：模块\_(位置)描述.xml

例如：activity\_main\_head.xml、activity\_main\_bottom.xml

注意：通用的包含项命名采用：项目名称缩写\_描述.xml

例如：xxxx\_title.xml

**4.2 资源文件（图片drawable文件夹下）**

全部小写，采用下划线命名法，加前缀区分

命名模式：可加后缀 \_small 表示小图， \_big 表示大图，逻辑名称可由多个单词加下划线组成，采用以下规则：

* 用途\_模块名\_逻辑名称
* 用途\_模块名\_颜色
* 用途\_逻辑名称
* 用途\_颜色

说明：用途也指控件类型（具体见附录[UI控件缩写表](https://github.com/Blankj/AndroidStandardDevelop" \l "ui%E6%8E%A7%E4%BB%B6%E7%BC%A9%E5%86%99%E8%A1%A8)）

例如：

| **名称** | **说明** |
| --- | --- |
| btn\_main\_home.png | 按键 |
| divider\_maket\_white.png | 分割线 |
| ic\_edit.png | 图标 |
| bg\_main.png | 背景 |
| btn\_red.png | 红色按键 |
| btn\_red\_big.png | 红色大按键 |
| ic\_head\_small.png | 小头像 |
| bg\_input.png | 输入框背景 |
| divider\_white.png | 白色分割线 |

如果有多种形态，如按钮选择器：btn\_xx.xml(selector)

| **名称** | **说明** |
| --- | --- |
| btn\_xx | 按钮图片使用btn\_整体效果（selector） |
| btn\_xx\_normal | 按钮图片使用btn\_正常情况效果 |
| btn\_xx\_pressed | 按钮图片使用btn\_点击时候效果 |
| btn\_xx\_focused | state\_focused聚焦效果 |
| btn\_xx\_disabled | state\_enabled (false)不可用效果 |
| btn\_xx\_checked | state\_checked选中效果 |
| btn\_xx\_selected | state\_selected选中效果 |
| btn\_xx\_hovered | state\_hovered悬停效果 |
| btn\_xx\_checkable | state\_checkable可选效果 |
| btn\_xx\_activated | state\_activated激活的 |
| btn\_xx\_windowfocused | state\_window\_focused |
| bg\_head | 背景图片使用bg\_功能\_说明 |
| def\_search\_cell | 默认图片使用def\_功能\_说明 |
| ic\_more\_help | 图标图片使用ic\_功能\_说明 |
| seg\_list\_line | 具有分隔特征的图片使用seg\_功能\_说明 |
| sel\_ok | 选择图标使用sel\_功能\_说明 |

注意：使用AndroidStudio的插件SelectorChapek可以快速生成selector，前提是命名要规范。

**4.3 动画文件（anim文件夹下）**

全部小写，采用下划线命名法，加前缀区分。

具体动画采用以下规则：模块名\_逻辑名称。

例如：refresh\_progress.xml、market\_cart\_add.xml、market\_cart\_remove.xml。

普通的tween动画采用如下表格中的命名方式：动画类型\_方向

| **名称** | **说明** |
| --- | --- |
| fade\_in | 淡入 |
| fade\_out | 淡出 |
| push\_down\_in | 从下方推入 |
| push\_down\_out | 从下方推出 |
| push\_left | 推向左方 |
| slide\_in\_from\_top | 从头部滑动进入 |
| zoom\_enter | 变形进入 |
| slide\_in | 滑动进入 |
| shrink\_to\_middle | 中间缩小 |

**4.4 values中name命名**

**4.4.1 colors.xml**

colors的name命名使用下划线命名法，在你的colors.xml文件中应该只是映射颜色的名称一个ARGB值，而没有其它的。不要使用它为不同的按钮来定义ARGB值。

**不要这样做**

<resources>

<color name="button\_foreground">#FFFFFF</color>

<color name="button\_background">#2A91BD</color>

<color name="comment\_background\_inactive">#5F5F5F</color>

<color name="comment\_background\_active">#939393</color>

<color name="comment\_foreground">#FFFFFF</color>

<color name="comment\_foreground\_important">#FF9D2F</color>

...

<color name="comment\_shadow">#323232</color>

使用这种格式，你会非常容易的开始重复定义ARGB值，这使如果需要改变基本色变的很复杂。同时，这些定义是跟一些环境关联起来的，如button或者comment, 应该放到一个按钮风格中，而不是在color.xml文件中。

**相反，这样做**

<resources>

<!-- grayscale -->

<color name="white" >#FFFFFF</color>

<color name="gray\_light">#DBDBDB</color>

<color name="gray" >#939393</color>

<color name="gray\_dark" >#5F5F5F</color>

<color name="black" >#323232</color>

<!-- basic colors -->

<color name="green">#27D34D</color>

<color name="blue">#2A91BD</color>

<color name="orange">#FF9D2F</color>

<color name="red">#FF432F</color>

</resources>

向应用设计者那里要这个调色板，名称不需要跟"green"、"blue"等等相同。"brand\_primary"、"brand\_secondary"、"brand\_negative"这样的名字也是完全可以接受的。 像这样规范的颜色很容易修改或重构，会使应用一共使用了多少种不同的颜色变得非常清晰。 通常一个具有审美价值的UI来说，减少使用颜色的种类是非常重要的。

**4.4.2 dimens.xml**

像对待colors.xml一样对待dimens.xml文件 与定义颜色调色板一样，你同时也应该定义一个空隙间隔和字体大小的“调色板”。 一个好的例子，如下所示：

<resources>

<!-- font sizes -->

<dimen name="font\_larger">22sp</dimen>

<dimen name="font\_large">18sp</dimen>

<dimen name="font\_normal">15sp</dimen>

<dimen name="font\_small">12sp</dimen>

<!-- typical spacing between two views -->

<dimen name="spacing\_huge">40dp</dimen>

<dimen name="spacing\_large">24dp</dimen>

<dimen name="spacing\_normal">14dp</dimen>

<dimen name="spacing\_small">10dp</dimen>

<dimen name="spacing\_tiny">4dp</dimen>

<!-- typical sizes of views -->

<dimen name="button\_height\_tall">60dp</dimen>

<dimen name="button\_height\_normal">40dp</dimen>

<dimen name="button\_height\_short">32dp</dimen>

</resources>

布局时在写margins和paddings时，你应该使用spacing\_xxxx尺寸格式来布局，而不是像对待string字符串一样直接写值。 这样写会非常有感觉，会使组织和改变风格或布局是非常容易。

**4.4.3 strings.xml**

strings的name命名使用下划线命名法，采用以下规则：模块名+逻辑名称

| **名称** | **说明** |
| --- | --- |
| main\_menu\_about | 主菜单按键文字 |
| friend\_title | 好友模块标题栏 |
| friend\_dialog\_del | 好友删除提示 |
| login\_check\_email | 登录验证 |
| dialog\_title | 弹出框标题 |
| button\_ok | 确认键 |
| loading | 加载文字 |

**4.4.4 styles.xml**

几乎每个项目都需要适当的使用style文件，因为对于一个视图来说有一个重复的外观是很常见的，将所有的外观细节属性（colors、padding、font）放在style文件中。 在应用中对于大多数文本内容，最起码你应该有一个通用的style文件，例如：

<style name="ContentText">

<item name="android:textSize">@dimen/font\_normal</item>

<item name="android:textColor">@color/basic\_black</item>

</style>

应用到TextView中:

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/price"

style="@style/ContentText"

/>

你或许需要为按钮控件做同样的事情，不要停止在那里。将一组相关的和重复android:\*\*\*\*的属性放到一个通用的style中。

**将一个大的style文件分割成多个文件**， 你可以有多个styles.xml 文件。Android SDK支持其它文件，styles这个文件名称并没有作用，起作用的是在文件 里xml的<style>标签。因此你可以有多个style文件styles.xml、style\_home.xml、style\_item\_details.xml、styles\_forms.xml。 不同于资源文件路径需要为系统构建起的有意义，在res/values目录下的文件可以任意命名。

**4.5 layout中的id命名**

命名模式为：view缩写\_模块名\_逻辑名，比如btn\_main\_search 使用AndroidStudio的插件ButterKnife Zelezny，生成注解非常方便，原生的话可以使用Android Code Generator插件。

如果想对资源文件进行分包可以参考我这篇文章～[**Android Studio下对资源进行分包**](http://www.jianshu.com/p/8e893581b9c7)

**5 版本统一规范**

Android开发存在着众多版本的不同，比如compileSdkVersion、minSdkVersion、targetSdkVersion以及项目中依赖第三方库的版本，不同的module及不同的开发人员都有不同的版本，所以需要一个统一版本规范的文件。

具体可以参考我写的这篇博文～[**Android开发之版本统一规范**](http://www.jianshu.com/p/db6ef4cfa5d1)

**6 第三方库规范**

别再闭门造车了，用用最新最火的技术吧，安利一波～[**Android 流行框架查速表**](http://www.ctolib.com/cheatsheets-Android-ch.html)，顺便带上自己的干货～[**Android开发人员不得不收集的代码**](https://github.com/Blankj/AndroidUtilCode)

希望Team能用时下较新的技术，对开源库的选取，一般都需要选择比较稳定的版本，作者在维护的项目，要考虑作者对issue的解决，以及开发者的知名度等各方面。选取之后，一定的封装是必要的。

个人推荐Team可使用如下优秀轮子：

* [**Retrofit**](https://github.com/square/retrofit)
* [**RxAndroid**](https://github.com/ReactiveX/RxAndroid)
* [**OkHttp**](https://github.com/square/okhttp)
* [**Glide**](https://github.com/bumptech/glide) / [**Fresco**](https://github.com/facebook/fresco)
* [**Gson**](https://github.com/google/gson) / [**Fastjson**](https://github.com/alibaba/fastjson)
* [**EventBus**](https://github.com/greenrobot/EventBus) / [**AndroidEventBus**](https://github.com/bboyfeiyu/AndroidEventBus)
* [**GreenDao**](https://github.com/greenrobot/greenDAO)
* [**Dagger2**](https://github.com/google/dagger)(选用)
* [**Tinker**](https://github.com/Tencent/tinker)(选用)

**7 注释规范**

为了减少他人阅读你代码的痛苦值，请在关键地方做好注释。

**7.1 类注释**

每个类完成后应该有作者姓名和联系方式的注释，对自己的代码负责。

/\*\*

\* <pre>

\* author : Blankj

\* e-mail : xxx@xx

\* time : 2017/03/07

\* desc : xxxx描述

\* version: 1.0

\* </pre>

\*/

public class WelcomeActivity {

...

}

具体可以在AS中自己配制，Settings → Editor → File and Code Templates → Includes → File Header，输入

/\*\*

\* <pre>

\* author : ${USER}

\* e-mail : xxx@xx

\* time : ${YEAR}/${MONTH}/${DAY}

\* desc :

\* version: 1.0

\* </pre>

\*/

这样便可在每次新建类的时候自动加上该头注释。

**7.2 方法注释**

每一个成员方法（包括自定义成员方法、覆盖方法、属性方法）的方法头都必须做方法头注释，在方法前一行输入/\*\* + 回车或者设置Fix doc comment(Settings → Keymap → Fix doc comment)快捷键，AS便会帮你生成模板，我们只需要补全参数即可，如下所示。

/\*\*

\* bitmap转byteArr

\*

\* @param bitmap bitmap对象

\* @param format 格式

\* @return 字节数组

\*/

public static byte[] bitmap2Bytes(Bitmap bitmap, CompressFormat format) {

if (bitmap == null) return null;

ByteArrayOutputStream baos = new ByteArrayOutputStream();

bitmap.compress(format, 100, baos);

return baos.toByteArray();

}

**7.3 块注释**

块注释与其周围的代码在同一缩进级别。它们可以是/\* ... \*/风格，也可以是// ...风格(**//后最好带一个空格**）。对于多行的/\* ... \*/注释，后续行必须从\*开始， 并且与前一行的\*对齐。以下示例注释都是OK的。

/\*

\* This is // And so /\* Or you can

\* okay. // is this. \* even do this. \*/

\*/

注释不要封闭在由星号或其它字符绘制的框架里。

Tip：在写多行注释时，如果你希望在必要时能重新换行(即注释像段落风格一样)，那么使用/\* ... \*/。

**8 测试规范**

业务开发完成之后，开发人员做单元测试，单元测试完成之后，保证单元测试全部通过同时单元测试代码覆盖率达到一定程度（这个需要开发和测试约定，理论上越高越好），开发提测。

// TODO...

**9 RN规范**

// TODO...

**10 其他的一些规范**

1. 合理布局，有效运用<merge>、<ViewStub>、<include>标签；
2. Activity和Fragment里面有许多重复的操作以及操作步骤，所以我们都需要提供一个BaseActivity和BaseFragment，让所有的Activity和Fragment都继承这个基类。
3. 方法基本上都按照调用的先后顺序在各自区块中排列；
4. 相关功能作为小区块放在一起（或者封装掉）；
5. 当一个类有多个构造函数，或是多个同名方法，这些函数/方法应该按顺序出现在一起，中间不要放进其它函数/方法；
6. 数据提供统一的入口。无论是在 MVP、MVC 还是 MVVM 中，提供一个统一的数据入口，都可以让代码变得更加易于维护。比如可使用一个DataManager，把 http、preference、eventpost、database 都放在DataManger里面进行操作，我们只需要与DataManger打交道；
7. 多用组合, 少用继承；
8. 提取方法, 去除重复代码。对于必要的工具类抽取也很重要，这在以后的项目中是可以重用的。
9. 可引入 Dagger2 减少模块之间的耦合性。Dagger2 是一个依赖注入框架，使用代码自动生成创建依赖关系需要的代码。减少很多模板化的代码，更易于测试，降低耦合，创建可复用可互换的模块；
10. 项目引入RxJava + RxAndroid这些响应式编程，可以极大的减少逻辑代码；
11. 通过引入事件总线，如：EventBus、AndroidEventBus、RxBus，它允许我们在DataLayer中发送事件，以便ViewLayer中的多个组件都能够订阅到这些事件，减少回调；
12. 尽可能使用局部变量；
13. 及时关闭流；
14. 尽量减少对变量的重复计算；

如下面的操作：

for (int i = 0; i < list.size(); i++) {

...

}

建议替换为：

for (int i = 0, int length = list.size(); i < length; i++) {

...

}

1. 尽量采用懒加载的策略，即在需要的时候才创建；

例如：

String str = "aaa";

if (i == 1) {

list.add(str);

}

建议替换为：

if (i == 1) {

String str = "aaa";

list.add(str);

}

1. 不要在循环中使用try…catch…，应该把其放在最外层；
2. 使用带缓冲的输入输出流进行IO操作；
3. 尽量使用HashMap、ArrayList、StringBuilder，除非线程安全需要，否则不推荐使用Hashtable、Vector、StringBuffer，后三者由于使用同步机制而导致了性能开销；
4. 尽量在合适的场合使用单例；

使用单例可以减轻加载的负担、缩短加载的时间、提高加载的效率，但并不是所有地方都适用于单例，简单来说，单例主要适用于以下三个方面：

（1）控制资源的使用，通过线程同步来控制资源的并发访问

（2）控制实例的产生，以达到节约资源的目的

（3）控制数据的共享，在不建立直接关联的条件下，让多个不相关的进程或线程之间实现通信

1. 把一个基本数据类型转为字符串，基本数据类型.toString()是最快的方式、String.valueOf(数据)次之、数据 + ""最慢；
2. 使用AS自带的Lint来优化代码结构（什么，你不会？右键module、目录或者文件，选择Analyze → Inspect Code）；
3. 最后不要忘了内存泄漏的检测；

**附录**

**UI控件缩写表**

| **名称** | **缩写** |
| --- | --- |
| TextView | txt |
| EditText | edit |
| Button | btn |
| ImageButton | ibtn |
| ImageView | img |
| ListView | lv |
| RadioGroup | rgroup |
| RadioButton | rbtn |
| ProgressBar | rbar |
| SeekBar | seek |
| CheckBox | cb |
| Spinner | spinner |
| TableLayout | table |
| TableRow | row |
| LinearLayout | ll |
| RelativeLayout | rl |
| ScrollView | scroll |
| SearchView | search |
| TabHost | thost |
| TabWidget | twidget |

**常见的英文单词缩写表**

| **名称** | **缩写** |
| --- | --- |
| icon | ic （主要用在app的图标） |
| color | cl（主要用于颜色值） |
| divider | di（主要用于分隔线，不仅包括Listview中的divider，还包括普通布局中的线） |
| selector | sl（主要用于某一view多种状态，不仅包括Listview中的selector，还包括按钮的selector） |
| average | avg |
| background | bg（主要用于布局和子布局的背景） |
| buffer | buf |
| control | ctrl |
| delete | del |
| document | doc |
| error | err |
| escape | esc |
| increment | inc |
| infomation | info |
| initial | init |
| image | img |
| Internationalization | I18N |
| length | len |
| library | lib |
| message | msg |
| password | pwd |
| position | pos |
| server | srv |
| string | str |
| temp | tmp |
| window | wnd(win) |

程序中使用单词缩写原则：不要用缩写，除非该缩写是约定俗成的。