

Android 代码规范

为什么需要代码规范？

[便于维护]

一个软件的生命周期中，80%的花费在于维护。

[提高代码质量]

良好的编码习惯和规范的编码规约可以提高程序的鲁棒性。

[团队协作]

风格统一，规范一致的代码有助于团队内成员快速上手，降低交接成本，阅读成本。

[程序员个人能力]

是的，我们常常从一个程序员写的代码本身看出很多东西。

参考链接

[[面向贡献者的 AOSP 代码样式指南](#)]

[[Google Java 编程风格指南](#)]

[[阿里巴巴 Java 开发手册](#)]

[[Android Java 编码规范\(adroplv\)](#)]

[[阿里巴巴 Android 开发手册](#)]

一、命名规范

【使用场景】编程过程中所有需要自定义名称的场合

【目的】规范命名便于本人及团队其他成员理解代码，降低维护成本

【说明】

- 大驼峰 (UpperCamelCase): 每个词首字母大写，其他字母小写

- 小驼峰 (lowerCamelCase): 第一个词全部小写，其他词首字母大写，其他字母小写

【正文】

1.[通用][强制]代码中命名不使用下划线或美元符开始和结尾。***

1.[通用][强制]代码中的命名严禁使用拼音与英文混合的方式。*

1.[通用][强制]代码中的命名拼写错误必须检查，避免拼写错误 (专有名词/通用缩写除外)。

```
public String signattrue = null; // 签名  
private CountdownTimer mCountDownlTimer
```

```
private OnVideoReleaseCreateListener onRelaseListener;
```

```
// 是否支持多点触摸? (SDK>=7.0 屏幕支持多点)  
public static boolean IS_SUPPORT_MULTITOUCH_GESTURER = true;
```

- IDE 会把拼写不正确的单词标波浪线，并提示“typo: In word”。每出现这种情况时，必须检查是否需要修正 (1.添加下划线或者遵循驼峰规则，大写单词的第一个字母 2.修正拼写错误)。

1.[通用][强制]代码中的命名严禁使用不规范的缩写 (望文不知义)。对英文缩写拿不准时使用完整单词，命名长些好过以后难以理解。*

1.[包名][强制]包名统一使用小写，点分隔之前有且仅有一个自然语义单词，不使用下滑线。

- 一级包名为顶级域名: com/edu/gov/net/org

- 二级包名为公司/个人名

- 常见分包: db/base/util/activity/adapter/manager/test/common/model 等等

1.[包名][建议]包名统一使用单数形式

1.[类名][强制] 类名使用大驼峰 (UpperCamelCase) 风格。***

1.[类名][建议] 类名命名时建议根据其功能使用通用的前后缀。

- 示例

抽象类	使用 Abstract 或 Base 开头
异常类	使用 Exception 结尾
测试类	以它要测试的类名开始，以 Test 结尾
解析类	Paser 结尾
自定义共享基础类	以 Base 开头

1.[类名][建议] 使用到设计模式的类命名时，尽量体现出具体的模式，有助于快速理解。

- 示例: XXXProxy, XXXFactory, XXXDelegate

1.[类名][建议] 类名通用缩写不需全部大写。

- 示例: HtmlActivity、XmlFileDecoder、HttpRequestSender。(Android 特有，jdk 中专有缩写全部大写)

1.[方法名][强制] 方法名统一使用小驼峰(lowerCamelCase)风格。***

- 方法名尽量能体现方法功能
- 方法命名也有常用的前缀：init/is/check/handle/clear/remove... [建议]命名时根据方法的实际功能选择常用的前缀命名。

1.[变量][强制]局部变量名统一使用小驼峰(lowerCamelCase)风格。***

1.[变量][强制]类成员变量名统一使用小驼峰(lowerCamelCase)风格，[建议]以小写 m 开头。***

1.[变量][强制]静态字段名统一使用小驼峰(lowerCamelCase)风格，[建议]以小写 s(private)/g(public)开头。***

1.[常量][强制]常量命名统一使用大写字母加下划线格式。***

1.[控件 ID][建议]xml 文件内控件 id 的命名使用小写词加下划线格式，建议使用“view 缩写_描述”的方式命名。

- 常见控件缩写：

LinearLayout	ll
RelativeLayout	rl
FrameLayout	fl
RecyclerView	rv
ListView	lv
GridView	gv
ImageView	igv
TextView	txv
Button	btn

- 示例：btn_title_back、iv_download_manager、fl_content

1.[控件变量名][建议]代码中控件变量的命名参照[]，变量以控件类型的缩写结尾，大块背景控件以 rootView 或 layout 作为变量名称结尾（ll,fl,rl 不太好理解）

- 示例： mNameTxv、mNumberEdt、mPhotoIgv、mRootView、mCtrlLayout、mApplcon

1.[layout 文件名][强制]命名方式：小写字母加下划线。***

- Activity/Fragment/Dialog 以 activity_/fragment_/dialog_ 开头

- ListView/RecyclerView/GridView 的子项以 list_item/recycler_item/grid_item 开头

- 自定义 View 以 view_ 开头

1.[xml drawable 文件名][强制]命名方式：小写字母加下划线。***

- 以 selector_/shape_/layer_list_ 开头

1.[图片资源][强制]命名方式：小写字母加下划线。***

- 图标统一以 icon_ 开头（icon_back.png）

- 背景图以 bg_ 开头（bg_default_banner.png）

- 普通通用图片 pic_ 开头（pic_red_dot.png）

1.[anim 文件名][强制]命名方式：小写字母加下划线。***

- 以动画名称命名: fade_in.xml、slip_out.xml

1.[color/strings/dimens][强制]命名方式: 小写字母加下划线。***

- 使用模块名+描述的方式:

```
<color name="home_navigation_btn_tx_color">#112233</color>
<string name="home_navigation">导航</ string>
<dimens name="home_navigation_btn_height">56dp</ dimens>
```

1.[declare-styleable/style][强制]命名方式: 大驼峰 (UpperCamelCase) 风格。***

- 示例:

```
<style name="BigRatingBar" parent="@android:style/Widget.RatingBar">
    <item name="android:progressDrawable">@drawable/layer_big_ratingbar</item>
    <item name="android:minHeight">27dip</item>
</style>
<declare-styleable name="ExpandableTextView">
    <attr name="maxCollapsedLines" format="integer"/>
    <attr name="animDuration" format="integer"/>
    <attr name="animAlphaStart" format="float"/>
    <attr name="expandDrawable" format="reference"/>
    <attr name="collapseDrawable" format="reference"/>
</declare-styleable>
```

每当不确定某种类型的命名方式时, 参考 Android 源码中相应内容的命名方式。

二、 代码风格

【使用场景】代码编写（无关于功能实现的部分）。

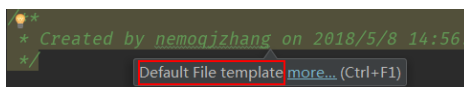
【目 的】统一文件的编写方式，方便团队内成员阅读

【正文】

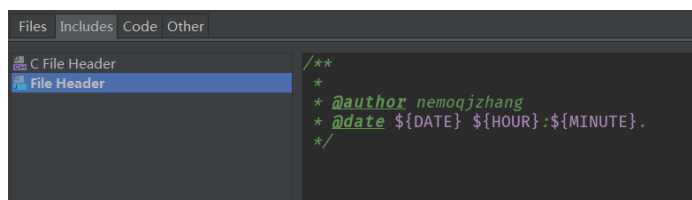
1.[注释][强制]类、类成员、类方法的注释必须使用 Javadoc 规范，即 “`/** 内容 */`” 格式，不使用 “`//内容`” 方式。***

1.[注释][强制]所有的类都必须添加创建者和创建日期。***

- 默认的文件头注释 IDE 会提示格式问题：



建议自定义文件头模板。File->Settings->Editor->File and Code Templates->Includes->File Header 可以设置新建文件的注释模板。

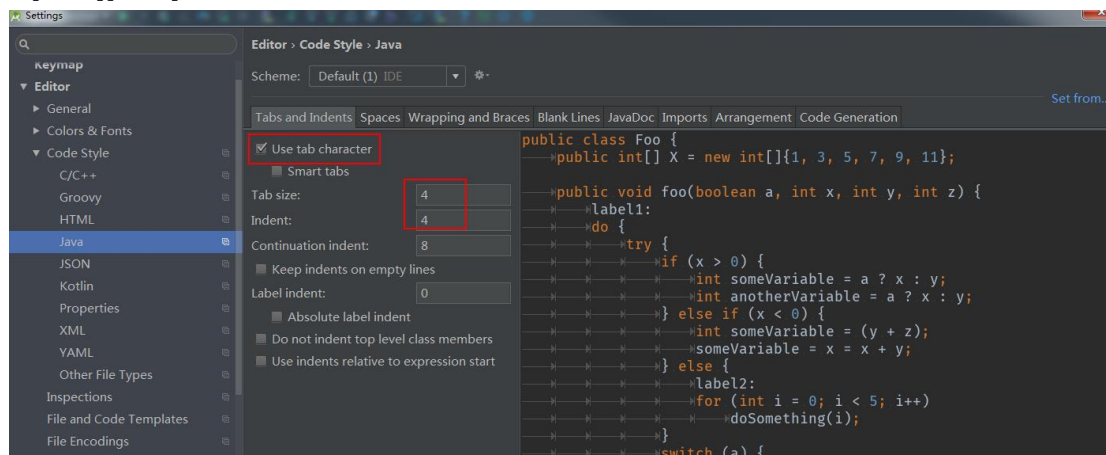


1.[注释][强制]不使用行尾注释。***

1.[注释][建议]注释掉的代码，要么说明保留的原因，要么删除掉（历史代码有版本控制工具可以保存）。

1.[注释][建议]抽象方法、接口中的方法须添加注释说明（参数、返回值、完成的功能等）。

1.[缩进][建议]把 AndroidStudio 设置为使用 tab 缩进，并将 tab 设置为 4 空格大小。



- 关于缩进应该使用 tab 还是 space 的问题。个人觉得：都行，关键是统一。

- 据说使用空格缩进的薪酬普遍更高...: <https://www.zhihu.com/question/19960028>

1.[大括号][建议]大括号内容为空简写为“{ }”

1.[大括号][强制]大括号内容非空时：***

- 左大括号前不换行

- 左大括号后换行

- 右大括号前换行
- 右大括号后还有 `else/while` 等代码时不换行，表示块终止时必须换行

1.[接口][强制]接口内的变量、常量、方法不添加任何修饰符。 **

- 接口中所有的方法都是 `public && abstract`；所有的成员都是 `public && final && static`。接口定义时没必要再写多余的修饰符，保持代码简洁。

```
public interface ActivityAnimationListener {
    /**
     * 进入动画开始
     */
    public abstract void onActivityEnterStart();
    /**
     * 进入动画结束
     */
    public abstract void onActivityEnterEnd();
    /**
     * 返回动画开始
     */
    public abstract void onActivityBackStart();
    /**
     * 返回动画结束
     */
    public abstract void onActivityBackEnd();
}
```

1.[方法摆放][建议]类内方法摆放顺序，从上至下依次按照： `public/protected > private>getter/setter` 的顺序摆放。

1.[方法摆放][强制]类内同名方法放在一起（优先级高于 1.）。 ***

1.[变量摆放][建议]类内变量按照类型（常量、静态变量、成员变量）分块放置，便于阅读。

```
JsBridge VIDEO_DOWNLOAD_STATE_CALLBACK
/**
 * 公共静态常量
 */
public static final int JS_BRIDGE_VERSION = 1;
public static final String JS_BRIDGE_SCHEME = "jsb://";
public Map<String, BaseJsBridgeImpl> getJsBridgeImpls() { return mJsBridgeImpls; }
public final Map<String, BaseJsBridgeImpl> mJsBridgeImpls = new ConcurrentHashMap<>();
// 把无法解释的参数放到一个统一的类里
public final JsBridgeParams mJsParams = new JsBridgeParams();
public static final String STATE_CALLBACK_FUNCTION_NAME = "stateCallback"; //任务下载状态回调
public static final String VIDEO_DOWNLOAD_STATE_CALLBACK = "videoDownloadStateCallback"; //视频下载状态回调
public static final String LOGIN_CALLBACK_FUNCTION_NAME = "loginCallback"; //登录结果回调
public static final String BUTTON_CLICK_CALLBACK_FUNCTION_NAME = "clickCallback"; //按钮点击事件

public static final String READY_CALLBACK_FUNCTION_NAME = "readyCallback"; //接口授权准备完毕回调
public static final String ACTIVITY_STATE_CALLBACK_FUNCTION_NAME = "activityStateCallback"; //
public static final String SHARE_CALLBACK_FUNCTION_NAME = "shareCallback"; //分享结果的回调
public static final String FILE_CHOOSER_CALLBACK_FUNCTION_NAME = "fileChooserCallback"; //文件选择的回调
public static final String APP_INSTALL_UNINSTALL = "appInstallUninstall"; //应用安装卸载回调
public static final String APP_INSTALLED_AND_ACTION_COMPLETE_FUNCTION = "appInstalledAndActionCompleteCallback"; //应用安装

public static final String JS_INTERFACE_READY_NAME = "isInterfaceReady";
public static final String CALL_BATCH_NAME = "callBatch";

public GetDomainCapabilityEngine getDomainCapabilityEngine = new GetDomainCapabilityEngine();

public static final String JS_CONFIG = "js_config"; // 用于存储数据的SharedPreferences
public static final String JS_CONFIG_PREFIX = "js_config.";

public static final String WB_REPORT = "wb_report"; // WebView兼容性上报，用于存储SharedPreferences的key
```

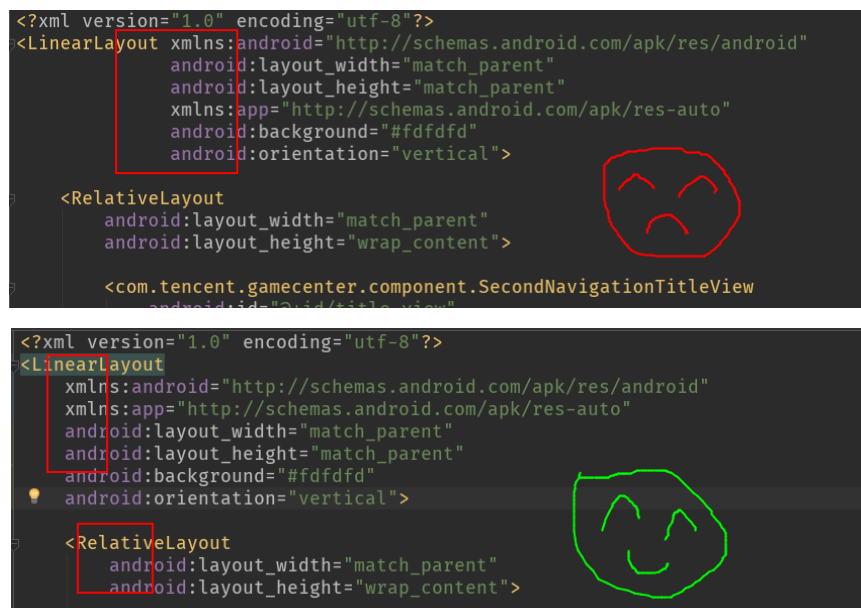
1.[文件编码][强制]工程内文本文件编码方式统一设置为 **UTF-8**。 ***

1.[换行][建议] 每一行文本的长度都应该不超过 100 个字符（超过 IDE 的行宽提示线，就可以考虑换行）

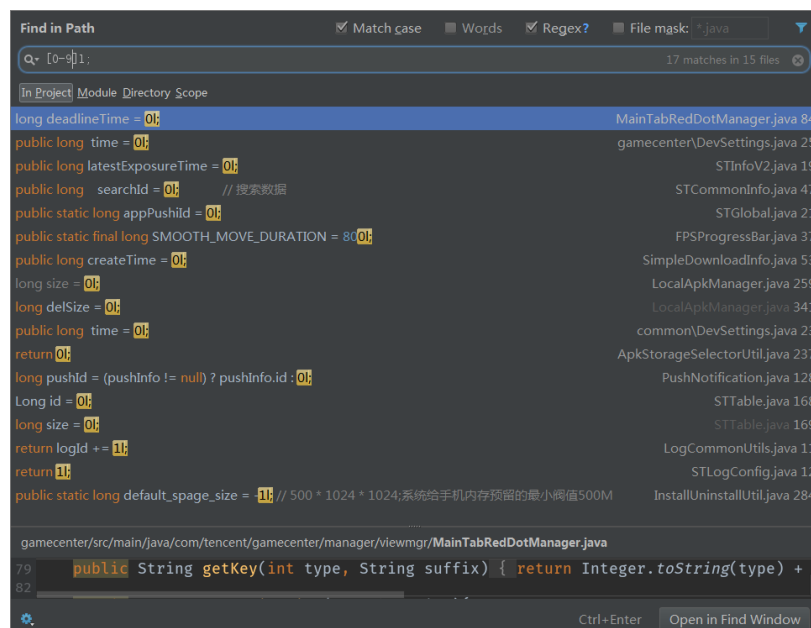
1.[换行][建议]换行时的断开规则

- 在非赋值运算符处换行，建议在该符号前断开(比如+，它将位于下一行)
- 在赋值运算符处换行，建议在该符号后断开(比如=，它与前面的内容留在同一行)
- 逗号、foreach 语句中的分号同赋值运算符
- 大括号换行规则参见[]

1.[换行][建议] 布局文件中控件标签名后统一换行：



1.[Long][强制] long 或者 Long 初始赋值时，使用大写的 L，不能是小写的 l。*



1.[数组类型][强制] 定义数组类型时中括号是类型的一部分，与类名放在一起：String[] args 而非 String args[]。***

1.[日志打印][建议] 类打印日志定义 private final static String TAG，并使用类名（适当结合前缀）的字符串常量命名。

- 正式版本不要打印包含敏感信息的日志
- 普通日志打印使用 i；错误日志使用 e
- 同一功能模块的日志 TAG 使用相同的前缀，方便过滤该模块所有日志。

1.[代码格式化][强制] 新代码提交前: *

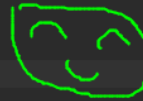
- 格式化代码 (Ctrl+ A && Ctrl+Shift+F) || (Ctrl+ A && Ctrl+Alt+L)
- 清理导包 (Ctrl+ Shift + O)

1.[布局控件][建议] 非 ViewGroup 控件定义时建议使用<TextView />格式, 不建议写成<TextView ></TextView>

```
<com.tencent.gamecenter.component.SecondNavigationView  
    android:id="@+id/title_view"  
    android:layout_width="match_parent"  
    android:layout_height="@dimen/common_title_bar_height">  
</com.tencent.gamecenter.component.SecondNavigationView>
```



```
<com.tencent.gamecenter.component.SecondNavigationView  
    android:id="@+id/title_view"  
    android:layout_width="match_parent"  
    android:layout_height="@dimen/common_title_bar_height"/>
```



三、 编码规约

【使用场景】代码实现。

【目 的】规范代码实现，避免易出错的实现方式，减少程序出错的可能，方便维护。

【Java 篇】


1.[finalize][强制]不要重载 finalize 方法。***

1.[finally][强制]不要在 finally 块中使用 return。

1.[switch][建议] switch 语句最好包含一个 default，即使什么也不做。

1.[复写][强制] 复写父类方法一定要使用@Override 标注。**

- 标明 Override 有助于读者快速了解方法所有信息。



```
public void goBack() { Ref
public void goForward() {
    try{
        ReflectTool.invoke
    }catch(Exception e){
        e.printStackTrace(
        ContentErrorRepo
    }finally{
    }
}
public void reload() {
    try{
```

1.[值比较][强制] 非基本类型的值比较统一使用 equals。**

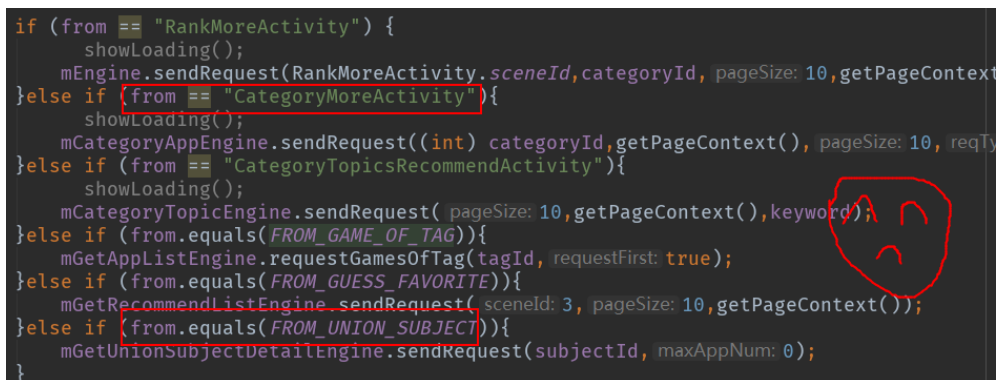
- 你清楚下边这两个比较的结果吗？

```
Integer i_100 = 100;
Integer i_100_copy = 100;
Integer i_200 = 200;
Integer i_200_copy = 200;

Log.i(TAG, msg: "i_100 == i_100_copy is " + (i_100 == i_100_copy));
Log.i(TAG, msg: "i_1200 == i_200_copy is " + (i_200 == i_200_copy));
```

1.[值比较][强制] equals 尽量用确认有值的对象调用。**

- 使用"1".equals(selfLink)而不是 selfLink.equals("1")。



```
if (from == "RankMoreActivity") {
    showLoading();
    mEngine.sendRequest(RankMoreActivity.sceneId, categoryId, pageSize: 10, getPageContext
} else if (from == "CategoryMoreActivity") {
    showLoading();
    mCategoryAppEngine.sendRequest((int) categoryId, getPageContext(), pageSize: 10, reqTy
} else if (from == "CategoryTopicsRecommendActivity") {
    showLoading();
    mCategoryTopicEngine.sendRequest( pageSize: 10, getPageContext(), keyword);
} else if (from.equals(FROM_GAME_OF_TAG)) {
    mGetAppListEngine.requestGamesOfTag(tagId, requestFirst: true);
} else if (from.equals(FROM_GUESS_FAVORITE)) {
    mGetRecommendListEngine.sendRequest( sceneId: 3, pageSize: 10, getPageContext());
} else if (from.equals(FROM_UNION_SUBJECT)) {
    mGetUnionSubjectDetailEngine.sendRequest(subjectId, maxAppNum: 0);
}
```

1.[字符串拼接][强制] 循环体内字符串拼接使用 StringBuilder 的 append（并发场景用 StringBuffer）。*

- 用 javap 查看字节码可以发现，字符串的“+”操作最终会被优化为 StringBuilder 对象的 append，而在循环体内字符串的“+”操作意味着 StringBuilder 对象的反复创建，影响性能。

- “+”和 append 的详细分析参考:<https://segmentfault.com/a/1190000007099818>。

```
public static void main(String[] args) {
    String a = appendByAddTest();
    String b = appendByStringBuilder();
    assert a.equals(b);
}

private static String appendByAddTest(){
    String s = "";
    long start = System.currentTimeMillis();
    for(int i = 0; i < 100000; i++){
        s = s + i%10;
    }
    System.out.println("<appendByAddTest    >time cost:" + (System.currentTimeMillis() - start) + "ms");
    return s;
}

private static String appendByStringBuilder(){
    StringBuilder sb = new StringBuilder();
    long start = System.currentTimeMillis();
    for(int i = 0; i < 100000; i++){
        sb.append(i%10);
    }
    System.out.println("<appendByStringBuilder>time cost:" + (System.currentTimeMillis() - start) + "ms");
    return sb.toString();
}
```

Foo

```
"C:\Program Files\Java\jdk1.8.0_65\bin\java" ...
<appendByAddTest    >time cost:4832ms
<appendByStringBuilder>time cost:3ms
Process finished with exit code 0
```

1.[Map][建议] 使用 entrySet 遍历 Map，而不是 keySet 方式进行遍历

```
HashMap<Integer, String> map = new HashMap<>();
Set<Integer> keySet = map.keySet();
for (int key : keySet) {
    String value = map.get(key);
    Log.i(TAG, msg: "key=" + key + ",value=" + value);
}

Set<Map.Entry<Integer, String>> entrySet = map.entrySet();
for (Map.Entry<Integer, String> entry : entrySet) {
    int key = entry.getKey();
    String value = entry.getValue();
    Log.i(TAG, msg: "key=" + key + ",value=" + value);
}
```

- 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。

1.[Map][建议] 以 Integer 作为 Map 的 key 时，数据量不大的情况下建议使用 SparseArray 代替 HashMap。

- SparseArray 仅能存储以 int 值为 key 的数据。内部把数据按 key 的大小存储，同时对数据的删除和存储机制有优化，使得它内存占用更小，性能更好。

- 详细分析: <https://juejin.im/entry/57c3e8c48ac24700634bd3cf>

1.[线程创建][强制] 线程创建时必须指定有意义的线程名。***

- 方便出问题找到它的主人，便于快速定位。

1.[线程创建][建议] 尽量避免直接创建线程，建议使用线程池操作线程。

- 频繁使用子线程的场景一定要使用线程池。控制线程创建开销，统一管理子线程。

1.[加锁][强制] 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。**

- 线程一获取 ABC 锁之后才能更新数据，线程二加锁顺序也必须是 ABC，否则可能出现死锁。


1.[异常处理][强制] 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。******

- 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。

1.[异常处理][强制] 捕获异常是为了处理它，不要捕获了却什么都不处理。`catch` 尽可能区分异常类型，再做对应的处理。*****

1.[编码习惯][建议] 表达异常的分支时，少用 `if-else`，这种方式可以改写成卫语句，或者根据实际场景选择策略模式、状态模式等优化代码结构。

```
if (condition1) {
    boolean condition2 = random.nextBoolean();
    if (condition2) {
        boolean condition3 = random.nextBoolean();
        if (condition3) {
            boolean condition4 = random.nextBoolean();
            if (condition4) {
                //do my job.
            }
        }
    }
}
```




```
Random random = new Random( seed: 120);
boolean condition1 = random.nextBoolean();
if (!condition1) {
    return;
}
boolean condition2 = random.nextBoolean();
if (!condition2) {
    return;
}

boolean condition3 = random.nextBoolean();
if (!condition3) {
    return;
}

boolean condition4 = random.nextBoolean();
if (!condition4) {
    return;
}

//do my job.
```



1.[编码习惯][建议] 异常不要用来做流程/条件控制。

1.[编码习惯][建议] 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。

1.[编码习惯][建议] 不要使用组合声明，比如 `int a, b`。

1.[编码习惯][强制] 判断语句的条件如果逻辑很复杂，需要适当分组拆分，并注释说明。*****

```
if (info != null && ((TextUtils.isEmpty(info.packageName) || !info.packageName.equals(Global.getApp().getPackageName())) || (info != null && !info.isUiTypeWiseDownload()))) {
    // wifi智能下载的应用不自动安装 (wifi智能下载中的普通应用更新会，因为需求需要它能自动升级到新版本)
    if (Global.ASSISTANT_DEBUG) XLog.d(TAG, msg: "Download succ callback.installApk,state:"+info.downloadState+",ticket:"+info.downloadTicket);
    if (info.autoInstall) {
        if (Global.ASSISTANT_DEBUG) XLog.d(tag: "WiseDownload", msg: "ignore by user, : " + info.name + " " + info.packageName);
        break;
    }
}
```

```
//奇葩的|| != &&
if (currentResourceHolder == null || resourceHolder.bgRes != currentResourceHolder.bgRes && downloadText != null) {
    // 一半会儿理解不了上边if条件的脑回路，做个判空保证程序安全
    if (downloadText != null) {
        downloadText.setBackgroundResource(resourceHolder.bgRes);
    }
}
```

1.[编码习惯][建议] 编写 Comparator 时需处理相等的情况。

- 不处理相等的情况时，使用该 comparator 比较时会出现 $a > b$ 和 $b > a$ 同时成立的情况。

【Android 篇】

1.[组件生命周期][强制] 避免在 UI 线程执行耗时操作。***

1.[组件生命周期][建议] 不要在 Activity#onDestroy()内执行释放资源的工作，例如一些工作线程的销毁和停止，因为 onDestroy()执行的时机可能较晚。可根据实际需要，在 Activity#onPause()/onStop()中结合 isFinishing()的判断来执行。

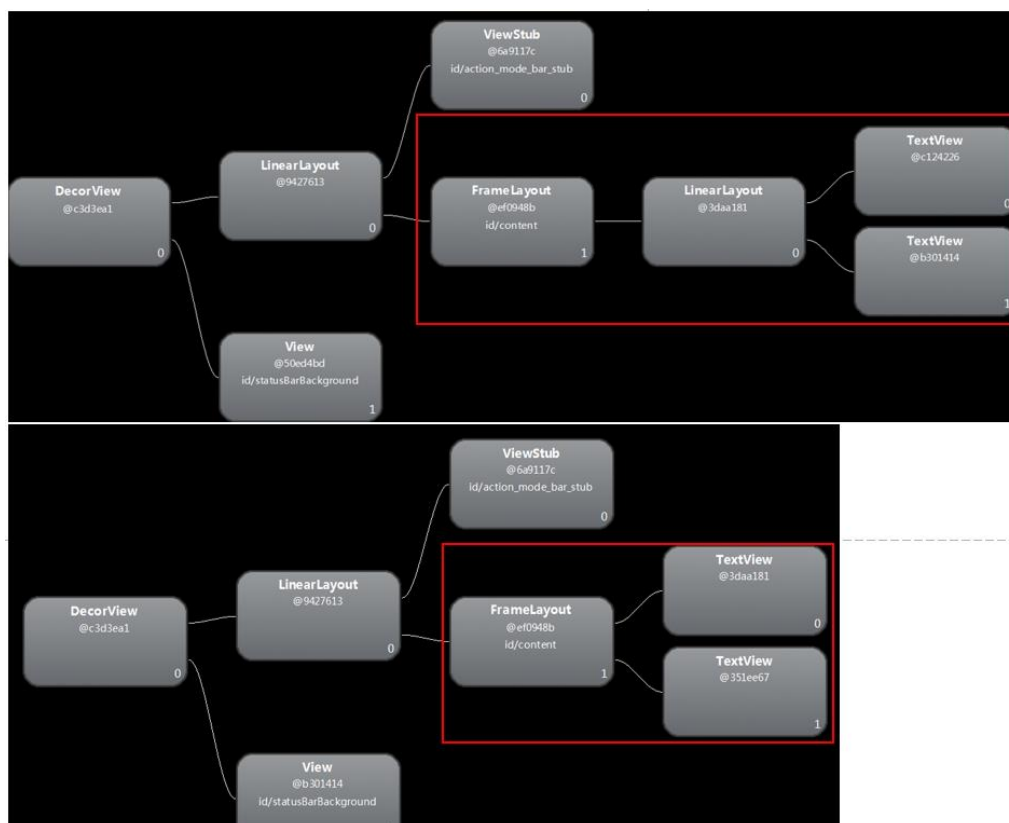
1.[组件生命周期][强制] Activity 或者 Fragment 中动态注册 BroadcastReceiver 时，registerReceiver() 和 unregisterReceiver()要成对出现。***

1.[组件生命周期][强制] Activity 的 onPause 方法中不要进行耗时操作。页面跳转时，当前 Activity 的 onPause 执行完之后才会执行下个 Activity 的 onCreate，过于耗时会影响到跳转时的效率。***

1.[UI 布局][建议] 文本大小使用 dp 单位，view 大小使用 dp 单位。

1.[UI 布局][建议] TextView 在文字大小确定的情况下建议使用 wrap_content 布局，手动设置 TextView 控件精确宽高可能导致文字显示不全。

1.[UI 布局][建议] 页面布局内层级尽可能的少，可以使用 merge/viewstub 优化布局，减少视图树种的节点个数、延迟加载某些非必须控件。



1.[UI 布局][强制] 不使用 ScrollView 包裹 RecyclerView、ListView、GridView、ExpandableListView。***

- 存在各种滑动上的冲突，需要增加滑动处理成本

- ScrollView 为了计算子控件的高度（宽度）需要把列表控件的所有 Item 都绘制出来，会消耗巨大的内存和 cpu，影响性能

1.[UI 布局][建议] 使用 styles 复用样式定义，减少重复代码。

1.[通信][强制] 不要通过 Intent 在 Android 基础组件之间传递大数据（binder transaction 缓存为 1MB）。***

1.[通信][强制] 子线程中不能更新界面，更新界面必须在主线程中进行，网络操作不能在主线程中调用。***

1.[通信][强制] 禁止在多进程之间用 SharedPreferences 共享数据，（MODE_MULTI_PROCESS）模式已过时，并不能保证多进程情况下的数据一致，官方已不推荐。***

1.[文件操作][强制]禁止硬编码文件路径，使用 Android 文件系统 API 访问文件系统。***

1.[文件操作][建议] 应用间共享文件时应使用 FileProvider。

- 背景：为了提高私有目录的安全性，防止应用信息的泄漏，从 Android 7.0 开始，应用私有目录的访问权限被做限制。具体表现为，开发人员不能够再简单地通过 file:// URI 访问其他应用的私有目录文件或者让其他应用访问自己的私有目录文件。

- 表现：targetApi 为 24 及以上使用到相关功能时，出现 FileUriExposedException 异常，导致应用崩溃。

- 常见场景：Call 起安装、调用系统拍照、调用系统裁剪等

适配 FileProvider: <https://blog.csdn.net/lmj623565791/article/details/72859156>

file:///storage/emulated/0/tencent/gamecenter/apk/com.tencent.tmgp.sydmhy2017_10000.apk

1.[文件操作][建议] SharedPreference 提交数据时，如果仅是为了保存这次修改，并没有立即读取操作时，建议使用 apply 而非 commit。（apply 提交内存，commit 写入磁盘）

1.[文件操作][强制] zip 解压时需要对可能的 ../../file 这样的路径做过滤，解压含有 "../../" 的文件路径的文件，可能覆盖已有文件，造成攻击。***

- 当 zip 压缩包中允许存在 "../../" 的字符串，攻击者可以利用多个 "../../" 在解压时改变 zip 文件存放的位置，当文件已经存在时就会进行覆盖，如果覆盖掉的文件是 so、dex 或者 odex 文件，就有可能造成严重的安全问题

- 判断代码：

```
public static boolean isValidEntry(ZipEntry entry) {
    boolean invalid = false;
    if(null != entry) {
        final String name = entry.getName();
        if(null != name && (name.contains("../../") || name.contains("../"))) {
            invalid = true;
        }
    }
    return invalid;
}
```

1.[资源][建议]大分辨率图片统一放置在 xxhdpi 目录下，并给低端机型适配低分辨率图。否则可能出现低端机型加载高分辨率图片导致。

- 为了支持多种屏幕尺寸和密度，Android 为多种屏幕提供不同的资源目录进行适配。为不同屏幕密度提供不同的位图可绘制对象，可用于密度特定资源的配置限定符（在下面详述）包括 ldpi（低）、mdpi（中）、hdpi（高）、xhdpi（超高）、xxhdpi（超超高）和 xxxhdpi（超超超高）。例如，高密度屏幕的位图应使用 drawable-hdpi/。根据当前的设备屏幕尺寸和密度，将会寻找最匹配的资源，如果将高分辨率图片放入低密度目录，将会造成低端机加载过大图片资源，又可能造成 OOM，同时也是资源浪费，没有必要在低端机使用大图。

1.[广播][建议]对于只用于应用（进程）内的广播，优先使用 LocalBroadcastManager 来进行

注册和发送，LocalBroadcastManager 安全性更好，同时拥有更高的运行效率。

1.[数据库][强制]多线程操作写入数据库时，需要使用事务，以免出现同步问题。 ***

```
public void insert(SQLiteDatabase db, String userId, String content) {
    ContentValues cv = new ContentValues();
    cv.put("userId", userId);
    cv.put("content", content);
    db.beginTransaction();
    try {
        db.insert( table: "TABLE_NAME", nullColumnHack: null, cv);
        // 其他操作
        db.setTransactionSuccessful();
    } catch (Exception e) {
        // TODO
    } finally {
        db.endTransaction();
    }
}
```

- 使用事务操作数据库可以保证数据的统一性和完整性，同时也可以提高效率。事务操作说白了就是把对数据库的一组增删改查操作打包，一起写入数据库（如果失败一起回滚）。

- Android 数据库事务简述：<https://www.cnblogs.com/wuyudong/p/5571168.html>

- room 框架自动生成的数据库写入代码均采用了上述事务操作模式。

1.[内存][建议] bitmap 使用结束后，在 2.3.3 及以下需要调用 recycle()函数，在 2.3.3 以上 GC 会自动管理，把 bitmap 的引用置空即可，除非明确不需要再用。

1.[内存][建议] 能用 Application Context 替代的地方，尽量不要使用 Activity、Service 的 Context。

1.[内存][建议]慎用 static 变量引用资源消耗大的示例，比如某个非全局 Context。

1.[安全][建议] 将 android:allowbackup 属性设置为 false，防止 adb backup 导出数据。