|  |
| --- |
| 山西华晟众慧信息科技有限公司 |
| Android代码规范 |
|  |

|  |
| --- |
| 王晓涛  2020-3-11 |
| Version:0.1 |

|  |  |  |
| --- | --- | --- |
| 更新记录 | | |
| 作者 | 日期 | 备注 |
| 王晓涛 | 2020/3/11 | 新建文档，版本号 0.1 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

目录

[1. 前言 4](#_Toc34810770)

[2. Android Studio 规范 4](#_Toc34810771)

[3. Kotlin命名 4](#_Toc34810772)

[3.1. 包名 4](#_Toc34810773)

[3.2. 类名 4](#_Toc34810774)

[3.3. 方法名 5](#_Toc34810775)

[3.4. 常量名 5](#_Toc34810776)

[4. Kotlin代码样式 6](#_Toc34810777)

[4.1. 类型推断 6](#_Toc34810778)

[4.2. 分号 7](#_Toc34810779)

[4.3. 声明When 7](#_Toc34810780)

[4.4. 可空类型 7](#_Toc34810781)

[4.5. 语言 8](#_Toc34810782)

[4.6. 大括号 8](#_Toc34810783)

[4.7. 编写简短方法 9](#_Toc34810784)

[4.8. 类成员的顺序 9](#_Toc34810785)

[4.9. 函数参数的排序 10](#_Toc34810786)

[4.10. 字符串常量的命名和值 11](#_Toc34810787)

[4.11. 行长限制 11](#_Toc34810788)

[4.11.1. 换行策略 11](#_Toc34810789)

[4.11.2. 操作符的换行 12](#_Toc34810790)

[4.11.3. 函数链的换行 12](#_Toc34810791)

[4.11.4. 多参数的换行 12](#_Toc34810792)

[4.11.5. RxJava 链式的换行 13](#_Toc34810793)

[5. Android资源文件 13](#_Toc34810794)

[5.1. 动画资源文件（anim/ 和 animator/） 13](#_Toc34810795)

[5.2. 颜色资源文件（color/） 14](#_Toc34810796)

[5.3. 图片资源文件（drawable/ 和 mipmap/） 14](#_Toc34810797)

[5.4. 布局资源文件（layout/） 15](#_Toc34810798)

[5.5. 菜单资源文件（menu/） 16](#_Toc34810799)

[5.6. values 资源文件（values/） 16](#_Toc34810800)

[5.6.1. colors.xml 16](#_Toc34810801)

[5.6.2. dimens.xml 17](#_Toc34810802)

[5.6.3. strings.xml 18](#_Toc34810803)

[5.6.4. styles.xml 19](#_Toc34810804)

[6. Android基本组件 19](#_Toc34810805)

[7. 代码 22](#_Toc34810806)

[8. UI与布局 23](#_Toc34810807)

[9. 安全 24](#_Toc34810808)

[10. 注释 24](#_Toc34810809)

[11. 测试 26](#_Toc34810810)

[12. Git使用 26](#_Toc34810811)

[13. 附录 28](#_Toc34810812)

# 前言

为了有利于项目维护、增强代码可读性、提升 Code Review 效率以及规范团队安卓开发，故提出以下安卓开发规范。

# Android Studio 规范

尽量使用最新的稳定版 Android Studio 进行开发；

编码格式统一为 UTF-8；

# Kotlin命名

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。

## 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，后面就是对包名的划分了，关于包名的划分，推荐采用 PBF（按功能分包 Package By Feature）。

// BAD

com.RayWenderlich.funky\_widget

// GOOD

com.huacheng.huiservers

## 类名

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。

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；或者以 I 为前缀。

|  |  |  |
| --- | --- | --- |
| 类 | 描述 | 示例 |
| Activity 类 | 以Activity 为后缀标识 | SplashActivity |
| Adapter 类 | 以Adapter 为后缀标识或者前缀 | 小区列表适配器  AdapteCommunityList |
| 工具方法类 | 以Util、Tool、Manager 为后缀标识或者前缀 | LogUtil |
| 数据库类 | 以Sql为后缀（项目使用的是greendao） | UserSql |
| Service类 | 以 Service 为后缀标识 | 时间服务 TimeService 用户组件服务IUserService |
| BroadcastReceiver 类 | 以 Receiver 为后缀标识 | NotificationClickReceiver |
| ContentProvider | 以 Provider 为后缀标识 | ShareProvider |
| 自定义的共享基础类 | 以 Base 为前缀 | BaseActivity, BaseFragment |
| presenter | 以presenter为后缀 | updatepresenter |

上面提到工具类以 Util 和 Tool 为后缀，那么 Util 和 Tool 的区别是什么？Util 是无业务逻辑的，Tool 是有业务逻辑的。比如 HttpUtil 只是包含了基本网络请求，而 HttpTool 中包含了项目的一些配置，如在每个请求增加 token 。也可以这么说，HttpUtil 可以跨项目使用，而 HttpTool 只能在该项目中使用。

## 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。

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

Android SDK 中的很多类都用到了键值对函数，比如 SharedPreferences、Bundle、Intent，所以，即便是一个小应用，我们最终也不得不编写大量的字符串常量。

当时用到这些类的时候，我们 必须 将它们的键定义为 const val 字段，并遵循以下指示作为前缀。

|  |  |
| --- | --- |
| 类 | 字段名前缀 |
| SharedPreferences | PREF\_ |
| Bundle | BUNDLE\_ |
| Fragment Arguments | ARGUMENT\_ |
| Intent Extra | EXTRA\_ |
| Intent Action | ACTION\_ |
| save instance state | SAVED\_ |

虽然 Fragment.getArguments() 和 onSaveInstanceState 也是使用的 Bundle ，但为了区分不同的使用场景，所以定义了不同的前缀。

# Kotlin代码样式

## 类型推断

**在可能的情况下，应选择使用类型推断代替显式声明实际的类型。**

// BAD

val something: MyType = MyType()

val meaningOfLife: Int = 42

// GOOD

val something = MyType()

val meaningOfLife = 42

## 分号

应该尽可能避免在Kotlin中使用分号。

// BAD

val horseGiftedByTrojans = true;

if (horseGiftedByTrojans) {

bringHorseIntoWalledCity();

}

// GOOD

val horseGiftedByTrojans = true

if (horseGiftedByTrojans) {

bringHorseIntoWalledCity()

}

## 声明When

如果应该以相同的方式处理它们，请使用逗号分隔。始终包括else情况。

// BAD

when (anInput) {

1 -> doSomethingForCaseOne()

2 -> doSomethingForCaseOneOrTwo()

3 -> doSomethingForCaseThree()

}

// GOOD

when (anInput) {

1, 2 -> doSomethingForCaseOneOrTwo()

3 -> doSomethingForCaseThree()

else -> println("No case satisfied")

}

## 可空类型

避免使用 !! ，减少出现空指针的情况。

// BAD

editText!!.setText("foo")

tvTitle.setText(title!!)

// GOOD

editText?.setText("foo")

tvTitle.setText(title ?: "") or title?.let { tvTitle.setText(it) }

## 语言

使用美式英语 ??拼写。

// BAD

val colourName = "red"

// GOOD

val colorName = "red"

## 大括号

左大括号不单独占一行，与其前面的代码位于同一行：

class MyClass {

fun func(): Int {

if (something) {

// ...

} else if (somethingElse) {

// ...

} else {

// ...

}

}

}

我们需要在条件语句周围添加大括号。例外情况：如果整个条件语句（条件和主体）适合放在同一行，那么您可以（但不是必须）将其全部放在一行上。例如，我们接受以下样式：

if (condition) {

body()

}

同样也接受以下样式：

if (condition) body()

但不接受以下样式：

if (condition)

body() // bad!

## 编写简短方法

在可行的情况下，尽量编写短小精炼的方法。有些情况下较长的方法是恰当的，因此对方法的代码长度没有做出硬性限制。如果某个方法的代码超出 40 行，请考虑是否可以在不破坏程序结构的前提下对其拆解，一个方法最好只做一件事情。

## 类成员的顺序

这并没有唯一的正确解决方案，但如果都使用一致的顺序将会提高代码的可读性，推荐使用如下排序：

常量

字段（public -> protected -> private）

构造函数

重写函数和回调 （在Android 中，应该将生命周期的函数放在前面）

公有函数

私有函数

内部类或接口

class MainActivity : Activity() {

companion object {

private const val TAG = "tag"

}

var updateTime = 0L

protected var mContent = ""

private var mTitle = ""

private lateinit var mTextViewTitle: TextView

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

...

}

fun setTitle(title:String){

mTitle = title

}

private fun setUpView(){

...

}

inner class AnInnerClass{

}

}

如果类继承于 Android 组件（例如 Activity 或 Fragment），那么把重写函数按照他们的生命周期进行排序是一个非常好的习惯，例如，Activity 实现了 onCreate()、onDestroy()、onPause()、onResume()，它的正确排序如下所示：

class MainActivity : Activity() {

// Order matches Activity lifecycle

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

}

override fun onResume() {

super.onResume()

}

override fun onPause() {

super.onPause()

}

override fun onDestroy() {

super.onDestroy()

}

}

## 函数参数的排序

在 Android 开发过程中，Context 在函数参数中是再常见不过的了，我们最好把 Context 作为其第一个参数。

把回调接口或函数应该作为其最后一个参数。

// Context always goes first

fun loadUser(context: Context, userId: Int): User

// Callbacks or funcation always go last

fun loadUserAsync(context: Context, userId: Int, callback: UserCallback)

fun loadUserAsync(context: Context, userId: Int, action: (User) -> Unit)

## 字符串常量的命名和值

Android SDK 中的很多类都用到了键值对函数，比如 SharedPreferences、Bundle、Intent，所以，即便是一个小应用，我们最终也不得不编写大量的字符串常量。

当时用到这些类的时候，我们 必须 将它们的键定义为 static final 字段，并遵循以下指示作为前缀。

类 字段名前缀

SharedPreferences PREF\_

Bundle BUNDLE\_

Fragment Arguments ARGUMENT\_

Intent Extra EXTRA\_

Intent Action ACTION\_

onSaveInstanceState SAVED\_

说明：虽然 Fragment.getArguments() 和 onSaveInstanceState 也是 Bundle ，但因为这是 Bundle 的常用用法，所以特意为此定义一个不同的前缀。

## 行长限制

代码中每一行文本的长度都应该不超过 100 个字符。虽然关于此规则存在很多争论，但最终决定仍是以 100 个字符为上限，如果行长超过了 100（AS 窗口右侧的竖线就是设置的行宽末尾 ），我们通常有两种方法来缩减行长。

提取一个局部变量或方法（最好）。

使用换行符将一行换成多行。

不过存在以下例外情况：

如果备注行包含长度超过 100 个字符的示例命令或文字网址，那么为了便于剪切和粘贴，该行可以超过 100 个字符。

导入语句行可以超出此限制，因为用户很少会看到它们（这也简化了工具编写流程）。

### 换行策略

这没有一个准确的解决方案来决定如何换行，通常不同的解决方案都是有效的，但是有一些规则可以应用于常见的情况。

### 操作符的换行

除赋值操作符之外，我们把换行符放在操作符之前，例如：

int longName = anotherVeryLongVariable + anEvenLongerOne - thisRidiculousLongOne

+ theFinalOne;

赋值操作符的换行我们放在其后，例如：

int longName =

anotherVeryLongVariable + anEvenLongerOne - thisRidiculousLongOne + theFinalOne;

### 函数链的换行

当同一行中调用多个函数时（比如使用构建器时），对每个函数的调用应该在新的一行中，我们把换行符插入在 . 之前。

例如：

Picasso.with(context).load("https://blankj.com/images/avatar.jpg").into(ivAvatar);

1

我们应该使用如下规则：

Picasso.with(context)

.load("https://blankj.com/images/avatar.jpg")

.into(ivAvatar);

### 多参数的换行

当一个方法有很多参数或者参数很长的时候，我们应该在每个 , 后面进行换行。

比如：

loadPicture(context, "https://blankj.com/images/avatar.jpg", ivAvatar, "Avatar of the user", clickListener);

1

我们应该使用如下规则：

loadPicture(context,

"https://blankj.com/images/avatar.jpg",

ivAvatar,

"Avatar of the user",

clickListener);

### RxJava 链式的换行

RxJava 的每个操作符都需要换新行，并且把换行符插入在 . 之前。

例如：

public Observable<Location> syncLocations() {

return mDatabaseHelper.getAllLocations()

.concatMap(new Func1<Location, Observable<? extends Location>>() {

@Override

public Observable<? extends Location> call(Location location) {

return mRetrofitService.getLocation(location.id);

}

})

.retry(new Func2<Integer, Throwable, Boolean>() {

@Override

public Boolean call(Integer numRetries, Throwable throwable) {

return throwable instanceof RetrofitError;

}

});

}

# Android资源文件

资源文件命名为全部小写，采用下划线命名法，资源文件需带module名作为前缀。(注意：现在项目里好多资源文件命名不是很规范)

## 动画资源文件（anim/ 和 animator/）

安卓主要包含属性动画和视图动画，其视图动画包括补间动画和逐帧动画。属性动画文件需要放在 res/animator/ 目录下，视图动画文件需放在 res/anim/ 目录下。

命名规则：module\_逻辑名称。

例如：common\_refresh\_progress.xml

如果是普通的补间动画或者属性动画，可采用：动画类型\_方向 的命名方式。

例如：

名称 说明

module\_fade\_in 淡入

module\_fade\_out 淡出

module\_push\_down\_in 从下方推入

module\_push\_down\_out 从下方推出

module\_push\_left 推向左方

module\_slide\_in\_from\_top 从头部滑动进入

module\_zoom\_enter 变形进入

module\_slide\_in 滑动进入

module\_shrink\_to\_middle 中间缩小

## 颜色资源文件（color/）

命名规则：module\_类型\_逻辑名称。

例如：module\_sel\_btn\_font.xml。

颜色资源也可以放于 res/drawable/ 目录，引用时则用 @drawable 来引用，但不推荐这么做，最好还是把两者分开。

## 图片资源文件（drawable/ 和 mipmap/）

**r**es/drawable/ 目录下放的是位图文件（.png、.9.png、.jpg、.gif）或编译为可绘制对象资源子类型的 XML 文件，而 res/mipmap/ 目录下放的是不同密度的启动图标，所以 res/mipmap/ 只用于存放启动图标，其余图片资源文件都应该放到 res/drawable/ 目录下。

命名规则：module\_类型\_逻辑名称

例如：user\_btn\_logout.png

大分辨率图片（单维度超过 1000）建议统一放在 xxhdpi 目录下管理，否则将导致占用内存成倍数增加 。

如果有多种形态，如按钮选择器：module\_sel\_btn\_xx.xml，采用如下命名：

名称 说明

module\_sel\_btn\_xx 作用在 btn\_xx 上的 selector

module\_btn\_xx\_normal 默认状态效果

module\_btn\_xx\_pressed state\_pressed 点击效果

module\_btn\_xx\_focused state\_focused 聚焦效果

module\_btn\_xx\_disabled state\_enabled 不可用效果

module\_btn\_xx\_checked state\_checked 选中效果

module\_btn\_xx\_selected state\_selected 选中效果

module\_btn\_xx\_hovered state\_hovered 悬停效果

module\_btn\_xx\_checkable state\_checkable 可选效果

module\_btn\_xx\_activated state\_activated 激活效果

module\_btn\_xx\_window\_focused state\_window\_focused 窗口聚焦效果

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

## 布局资源文件（layout/）

按照对应类型增加前缀：

类型 前缀（注意：项目中采用的是前缀如 activity\_home）

Activity module\_activity

Fragment module\_fragment

Dialog module\_dialog

include module\_include

ListView module\_list\_item

RecyclerView module\_recycle\_item

GridView module\_grid\_item

Activity/Fragment/Dialog的类名不需要增加module前缀，直接为每一个module分配一个package，比如：a.b.c.user、a.b.c.launch，所有的代码都放在该package下。

## 菜单资源文件（menu/）

命名规则：module\_逻辑名称

## values 资源文件（values/）

values/ 资源文件下的文件都以 s 结尾，如 attrs.xml、colors.xml、dimens.xml，起作用的不是文件名称，而是 <resources> 标签下的各种标签，比如 <style> 决定样式，<color> 决定颜色，所以，可以把一个大的 xml 文件分割成多个小的文件，比如可以有多个 style 文件，如 styles.xml、styles\_home.xml、styles\_item\_details.xml、styles\_forms.xml。

一个 Module 的资源文件都在 res 目录下，如果一个 Module 涉及到的页面很多，资源文件很多，不方便管理，可以多配置几个资源文件目录，这样我们可以对每个模块的资源都进行具体分类。

**(下面这个方法超屌的)**

方法很简单，配置我们的app文件夹下的 build.gradle 文件，比如：

android {

...

sourceSets {

main {

res.srcDirs('src/main/res', 'src/main/res\_core', 'src/main/res\_sub')

}

}

}

配置完之后，sync project 一下就成功了。

### colors.xml

<color> 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。

例如，不要像下面这样做：

<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，会非常容易重复定义 ARGB 值，而且如果应用要改变基色的话会非常困难。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 colors.xml 文件中。

相反，应该这样做：

<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。

注意：如果某些颜色和主题有关，那就单独写一个 colors\_theme.xml。

### 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\_22">22sp</dimen>

<dimen name="font\_18">18sp</dimen>

<dimen name="font\_15">15sp</dimen>

<dimen name="font\_12">12sp</dimen>

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

<dimen name="spacing\_40">40dp</dimen>

<dimen name="spacing\_24">24dp</dimen>

<dimen name="spacing\_14">14dp</dimen>

<dimen name="spacing\_10">10dp</dimen>

<dimen name="spacing\_4">4dp</dimen>

<!-- typical sizes of views -->

<dimen name="button\_height\_60">60dp</dimen>

<dimen name="button\_height\_40">40dp</dimen>

<dimen name="button\_height\_32">32dp</dimen>

</resources>

布局时在写 margins 和 paddings 时，你应该使用 spacing\_xx 尺寸格式来布局，而不是像对待 string 字符串一样直接写值，像这样规范的尺寸很容易修改或重构，会使应用所有用到的尺寸一目了然。 这样写会非常有感觉，会使组织和改变风格或布局非常容易。

### strings.xml

<string> 的 name 命名使用下划线命名法，采用以下规则：module\_逻辑名称，这样方便同一个界面的所有 string 都放到一起，方便查找。

减少复用字符串，每个组件维护自己的，不要嫌麻烦。因为一些语言同样一个意思在不同的语境下，会用到不同的单词。

使用字符串格式化替代字符串拼接。

### styles.xml

<style> 的 name 命名使用大驼峰命名法，几乎每个项目都需要适当的使用 styles.xml 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 styles.xml 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 styles.xml 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"/>

或许你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:xxxx 的属性放到一个通用的 <style>中。

# Android基本组件

1.Activity 间的数据通信，对于数据量比较大的，避免使用 Intent + Parcelable 的方式传递大数据（binder transaction缓存为 1MB），可能导致 OOM。可以考虑 EventBus 等替代方案，以免造成 TransactionTooLargeException。

2.Activity#onSaveInstanceState()方法不是 Activity 生命周期方法，也不保证一定会被调用。它是用来在 Activity 被意外销毁时保存 UI 状态的，只能用于保存临时性数据，例如 UI 控件的属性等，不能跟数据的持久化存储混为一谈。持久化存储应该在 Activity#onPause()/onStop()中实行。

3.避免在 Service#onStartCommand()/onBind()方法中执行耗时操作，如果确 实有需求，应改用 IntentService 或采用其他异步机制完成。

4.避免在 BroadcastReceiver#onReceive()中执行耗时操作，如果有耗时工作， 应该创建 IntentService 完成，而不应该在 BroadcastReceiver 内创建子线程去做。

5.对于只用于应用内的广播，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。

6.添 加 Fragment 时 ， 确 保 FragmentTransaction#commit() 在 Activity#onPostResume()或者 FragmentActivity#onResumeFragments()内调用。 不要随意使用 FragmentTransaction#commitAllowingStateLoss()来代替，任何 commitAllowingStateLoss()的使用必须经过 code review，确保无负面影响。

7.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。

8.当前 Activity 的 onPause 方法执行结束后才会创建(onCreate)或恢复 (onRestart)别的 Activity，所以在 onPause 方法中不适合做耗时较长的工作，这 会影响到页面之间的跳转效率。

9.Activity 或者 Fragment 中动态注册 BroadCastReceiver 时，registerReceiver() 和 unregisterReceiver()要成对出现。

10.Android的最小兼容版本到17.关于Android不同系统版本的市场占比情况详解

11.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。

12.必须支持界面被系统回收，用户再次打开App的时候能够恢复用户当时的使用状态，分析出具体在哪些情况下需要，哪些情况下不需要。例如弹窗，恢复弹窗上显示的数据，恢复弹窗的按钮的点击事件**。**

13.使用ArrayMap、ArraySet、SparseArray替换HashMap<T, T>、HashMap<int, T> 和 HashSet<T>。

14.自定义控件的自定义属性，必须使用控件名称作为前缀，否则容易造成同名属性。

15.新建线程时，必须通过线程池提供（AsyncTask 或者 ThreadPoolExecutor或者其他形式自定义的线程池），不允许在应用中自行显式创建线程 。

16.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 。

17.不要在非 UI 线程中初始化 ViewStub，否则会返回 null。

18.任何时候不要硬编码文件路径，请使用 Android 文件系统 API 访问。

示例：Android 应用提供内部和外部存储，分别用于存放应用自身数据以及应用产生的用

户数据。可以通过相关 API 接口获取对应的目录，进行文件操作。

android.os.Environment#getExternalStorageDirectory()

android.os.Environment#getExternalStoragePublicDirectory()

android.content.Context#getFilesDir()

android.content.Context#getCacheDir

19.当使用外部存储时，必须检查外部存储的可用性 。

20.SharedPreference 中只能存储简单数据类型（int、 boolean、 String 等），复杂数据类型建议使用文件、数据库等其他方式存储 。

21.SharedPreference 提 交 数 据 时 ， 尽 量 使 用 Editor#apply() ， 而 非Editor#commit()。一般来讲，仅当需要确定提交结果，并据此有后续操作时，才使用 Editor#commit()。

22.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 。

23.执行 SQL 语句时，应使用 SQLiteDatabase#insert()、 update()、 delete()，不要使用SQLiteDatabase#execSQL()，以免 SQL 注入风险。

24.不要通过 Msg 传递大的对象，会导致内存问题。

25.不能在 Activity 没有完全显示时显示 PopupWindow 和 Dialog。

26.不要在 Android 的 Application 对象中缓存数据。基础组件之间的数据共享请使用 Intent 等机制，也可使用 SharedPreferences 等数据持久化机制。

27.使用 Toast 时，建议定义一个全局的 Toast 对象，这样可以避免连续显示Toast 时不能取消上一次 Toast 消息的情况(如果你有连续弹出 Toast 的情况，避免使用 Toast.makeText)。

# 代码

1.使用Gson中的 @SerializedName 将服务器端返回数据字段不符合命名规范的转换为符合规范的命名，分离服务器和客户端字段的硬绑定。

2.多用组合，少用继承。

3.当一个类有多个构造函数，或是多个同名函数，这些函数应该按顺序出现在一起，中间不要放进其它函数。

4.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**。**

5.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。

6.尽可能使用局部变量。

7.及时关闭流。

8.不要在循环中使用 try…catch…，应该把其放在最外层。

9.使用带缓冲的输入输出流进行 IO 操作。

10.尽量在合适的场合使用单例；

11.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。

12.使用 AS 自带的 Lint 来优化代码结构（右键 module、目录或者文件，选择 Analyze -> Inspect Code）。

13.Kotlin 和 Java 是两种语言，在使用Kotlin的时候应该使用Kotlin中的特性，而不是固执的使用Java特性。比如：对于变量的赋值，如果仅仅是赋值，那么应该直接开放变量权限；如果涉及到一系列后续操作，那么使用方法。

# UI与布局

1.使用 start 和 end 替代 left 和 right，使布局能适应 RightRoLeft布局。

2.合理布局，有效运用 <merge>、<ViewStub>、<include> 标签。比如content 是一个 FrameLayout ，所以这里简单布局是不需要外层ViewGroup，直接使用 <merge> 来作为布局文件的根ViewGroup就可以了；

3.所有布局都需要考虑到大屏幕和小屏幕显示的问题，尤其是对大屏幕一个页面就可以展示完，而对于小屏幕不能展示完，需要增加 <ScrollView> 让页面能够滚动。目前一般情况下最小屏幕是 1280 \* 720。

4.字体单位使用 dp 设置字体而不是 sp。

5.对于设计图上按钮大小小于 24dp \* 24dp，应该使用 padding 将按钮的实际大小控制在大于等于 24dp \* 24dp。

6.使用 ConstraintLayout 替换 RelativeLayout。

7.不要过分依赖通用资源，否则会导致修改困难，该拆分的就拆分，不要在乎资源内容是一样的。

8.所有的文本按钮，还是使用 Button，不要使用 TextView ,毕竟系统对 Button 有样式渲染。同理对于图片按钮，应该使用 ImageButton 而不是 ImageView。

9.使用Space控件占据不显示内容的空间。

10.ScrollView 内部嵌套有 ListView 或 RecycleView 等，注意要考虑到ScrollView默认位置不是最顶部的情况。

11.考虑过渡绘制，不要直接在整个布局增加背景颜色，考虑是否在布局的一部分设置背景颜色就可以达到UI效果。

# 安全

1.将 android:allowbackup 属性必须设置为 false，阻止应用数据被导出。说明：android:allowBackup 原本是 Android 提供的 adb 调试功能，如果设置为 true， 可以导出应用数据备份并在任意设备上恢复。这对应用安全性和用户数据隐私构成 极大威胁，所以必须设置为 false，防止数据泄露。

2.在 SDK 支持的情况下，Android 应用必须使用 V2 签名，这将对 APK 文件的修改做更多的保护。

3.所有的 Android 基本组件(Activity、Service、BroadcastReceiver、ContentProvider 等)都不应在没有严格权限控制的情况下，将 android:exported 设 置为 true。

4.确保应用发布版本的 android:debuggable 属性设置为 false。

5.在 Android 4.2(API Level 17)及以上，对安全性要求较高的应用可在 Activity中，对 Activity 所关联的 Window 应用 WindowManager.LayoutParams.FLAG\_SECURE，防止被截屏、录屏。但要注意的是，一个 Activity 关联的 Window 可能不止一个，如果使用了 Dialog / DialogFragment 等控件弹出对话框，它们本身也会创建一个新的 Window，也一样需要保护**。**

# 注释

注释的功能是辅助程序员理解逻辑和业务，不是每一行都需要注释。一个好的命名就可以反应该字段或者函数的意思，没有必要多此一举增加注释。在具体的业务逻辑或者关键代码处，添加注释就可。

**9.1 类注释**

每个类完成后应该有作者姓名的注释，对自己的代码负责。

/\*\*

\* <pre>

\* author : wangxiaotao

\* time : 2020/04/23

\* desc : xxxx 描述

\* version: 1.0

\* </pre>

\*/

class SplashActivity {

...

}

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

/\*\*

\* <pre>

\* author : ${USER} //这里也可以写死，比如说 author : XiaoYangZi

\* 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。

**9.2 函数注释**

有选择性的书写。

/\*\*

\* 一句话功能描述

\* 功能详细描述

\* @param [参数1] [参数1说明]

\* @param [参数2] [参数2说明]

\* @return [返回类型说明]

\* @exception/throws [异常类型] [异常说明]

\* @deprecated [是否废弃]

\*/

**9.3 代码块注释**

写出代码的目的，而不是行为，行为可以通过具体的代码来判断。

# 测试

# Git使用

**12.1 分支**

**master：**

主分支，不轻易改动，新版本上线后，将新版本代码合并到该分支

**develop：(注意：我们的项目目前的分支是new\_branch\_20191101)**

团队人员很少的时候，所有的开发人员都在这个分支进行开发。

团队人员超过一个规模，每个开发人员在自己分支（dev\_xx）上开发，等开发到一定阶段，将自己的代码合并到develop分支上。

**fix\_version：**

修复Bug分支，修复对应版本的bug。

**refactor\_xx：**

重构分支，区别于修改Bug分支，重构不一定会在下一个新版本中上线。

**12.2 提交**

feat： 新功能（feature）

fix： 修复bug

docs： 仅文档修改（documentation）

style： 不影响代码含义的变化（空白，格式化，缺少分号等）

refactor：重构（既不是新增功能，也不是修改bug的变动）

chore： 构建过程、辅助工具、编辑器配置的变动

perf： 改进性能的代码更改

test: 添加缺失测试或更正现有测试

build: 影响构建系统或外部依赖关系的更改（示例范围：gulp，broccoli，npm）

ci: 更改我们的持续集成文件和脚本（示例范围：Travis，Circle，BrowserStack，SauceLabs）

示例：

fix(首页)：修复缓存异常

feat(用户)：新增修改用户头像的功能

**12.3 代码评审**

在两个及两个以上开发人员的项目中，应该进行代码评审，检查代码风格和是否有潜在的BUG。

**12.4其他**

**在Android Studio中设置好代码注释、模版或者其他配置文件后，导出settings.jar 包，可实现团队或换电脑时一键统一配置。**

**https://www.jb51.net/softjc/445324.html**

# 附录

**UI 控件缩写表**

|  |  |
| --- | --- |
| 名称 | 缩写 |
| Button | btn |
| CheckBox | cb |
| EditText | edt |
| GridView | gv |
| ImageButton | ib |
| ImageView | iv |
| LinearLayout | ll |
| ListView | lv |
| ProgressBar | pb |
| RadioButtion | rb |
| RecyclerView | rv |
| RelativeLayout | rl |
| ScrollView | sv |
| TextView | tv |
| SeekBar | sb |
| WebView | wv |

**常见的英文单词缩写表**

**名称 缩写**

**average avg**

**background bg（主要用于布局和子布局的背景）**

**buffer buf**

**control ctrl**

**current cur**

**default def**

**delete del**

**document doc**

**error err**

**escape esc**

**icon ic（主要用在 App 的图标）**

**increment inc**

**information info**

**initial init**

**image img**

**Internationalization I18N**

**length len**

**library lib**

**message msg**

**password pwd**

**position pos**

**previous pre**

**selector sel（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）**

**server srv**

**string str**

**temporary tmp**

**window 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。**