Android编码规范

五成胜算队

1. 前言

1.1 编写背景

现代软件产业经过几十年的发展，一个软件由一个人单枪匹马完成，已经很少见了，软件都是在相互合作中完成的。合作的最小单位是两个人，两个工程师在一起，做的最多的事情就是“看代码”，每个人都能看“别人的代码”，并发表意见。

编码规范对于程序员而言尤为重要，有以下几个原因：

（1）一个软件的生命周期中，80%的花费在于维护。

（2）几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。

（3）编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码。

（4）如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品。

1.2 编辑规范

本文档遵循编辑格式：任务分工及文档编辑规范

1.3 仅供参考

基本格式方面使用 Android Studio 默认模板（使用格式化快捷键处理后基本符合）。

2. 源文件基础

2.1 文件名

源文件以其最顶层的类名来命名，大小写敏感，文件扩展名为.kt。

2.2 文件编码

源文件编码格式为：UTF-8。

3. 源文件结构

一个源文件包含（按顺序地）：

（1）许可证或版权信息(如有需要)。

（2）package语句。

（3）import语句。

（4）一个顶级类（只有一个）以上每个部分之间用一个空行隔开。

3.1 package语句

package 语句不换行，即 package 语句始终写在一行当中。

3.2 import语句

import 语句不换行，即 import 语句始终写在一行当中。
import 语句不使用通配符，即不出现形如 `import java.util.*;` 的语句。

3.3 类声明

3.3.1 只有一个顶级类

类声明每个顶级类都在一个与它同名的源文件中。

3.3.2 类成员顺序

类成员按照某种逻辑排序，维护者应该要能解释这种排序逻辑。
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。

4. 命名约定

4.1 通用规则

标识符只能使用ASCII字母和数字，因此每个有效的标识符名称都能匹配正则表达式。

4.2 标识符类型的规则

4.2.1 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。一级包名为com，二级包名为xx（可以是公司或则个人的随便），三级包名根据应用进行命名，四级包名为模块名或层级名。
例如： `com.hymobile.nloc.activities`

4.2.2 类名和接口名

类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。采用大驼峰命名法，尽量避免缩写。
接口一般要使用 able、ible、er 等后缀。

4.2.3 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/check为前缀标识
getXX()	返回某个值的方法，使用get为前缀标识
handleXX()	对数据进行处理的方法，尽量使用handle为前缀标识
displayXX()/showXX()	弹出提示框和提示信息，使用display/show为前缀标识
saveXX()	与保存数据相关的，使用save为前缀标识
resetXX()	对数据重组的，使用reset前缀标识
clearXX()	清除数据相关的
removeXXX()	清除数据相关的
drawXXX()	绘制数据或效果相关的，使用draw前缀标识

4.2.4 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。
例如： `private val REQUEST_CODE_PICK_IMAGE = 1023`

4.2.5 变量名

方法名都以 lowerCamelCase 风格编写。
变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即能够指出其用途。尽量避免单个字符的变量名。

基本类型

例如： `private var lastClickTime = 0L`

其他数据类型

例如： `private var linearLayout: LinearLayout?= null`

4.2.6 layout名

全部单词小写，单词间以下划线分割，并且尽可能使用名词或名词词组，采用 **类型名_模块名_功能名称** 的命名格式。

例如： `activity_article_share.xml`

4.2.7 id名

全部单词小写，单词间以下划线分割，并且尽可能使用名词或名词词组，采用 **类型名_模块名_功能名称** 的命名格式。

例如： `android:id="@+id/btn_edit_camera"`

4.2.8 drawable名

全部单词小写，单词间以下划线分割，并且尽可能使用名词或名词词组，采用 **icon_类型名_模块名_功能名称** 的命名格式。

例如： `icon_action_home_sort.png`

5. 注释

5.1 文件注释

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明。

```
/**
 * 文件名
 * 包含类名列表
 * 版本信息，版本号
 * 创建日期
 * 版权声明
 */
```

5.2 类注释

每一个类都要包含如下格式的注释，以说明当前类的功能等。

```
/**
 * 类名
 * @author 作者
 * 实现的主要功能。
 * 创建日期
 * 修改者，修改日期，修改内容。
 */
```

5.3 方法注释

每一个方法都要包含如下格式的注释，以说明当前方法的功能等。

```
/**
 * 方法的一句话概述
 * <p>方法详述（简单方法可不必详述）</p>
 * @param s 说明参数含义
 * @return 说明返回值含义
 * 修改者，修改日期，修改内容。
 */
```

5.4 其他注释

使用双斜杠 // 进行注释。

5.5 XML注释

如果当前 layout 或资源需要被多处调用，或为公共使用的layout（若list_item），则需要在xml写明注释。要求注释清晰易懂。

6. 代码风格

6.1 大括号

6.1.1 非空块

对于非空块和块状结构，大括号遵循 Kernighan 和 Ritchie 风格（Egyptian brackets）：
（1）左大括号前不换行。
（2）左大括号后换行。
（3）右大括号前换行
（4）如果右大括号是一个语句、函数体或类的终止，则右大括号后换行; 否则不换行。
例如：

```
override fun onResume() {
    super.onResume()
    if (isFirstIn) {
        isFirstIn = false
        setOnScrollChangeListener(editor, onScrollChangeListener)
    }
}
```

6.1.2 空块

一个空的块状结构里什么也不包含，大括号可以简洁地写成{}，不需要换行。
例外：如果它是一个多块语句的一部分（例如：try/catch/finally），即使大括号内没内容，右大括号也要换行。

6.2 缩进

不允许使用Tab进行缩进，使用空格进行缩进，推荐缩进为4空格。

6.3 行宽

采用一行100个字符的列限制，除了下述例外，任何一行如果超过这个字符数限制，必须自动换行。
不可能满足列限制的行（例如，Javadoc中的一个长URL，或是一个长的JSNI方法参考）。

6.4 空行

通常在 **变量声明区域之后** 要用空行分隔，**常量声明区域之后** 要有空行分隔，**方法声明之前** 要有空行分隔。
下列情况应该总是使用空行：

1. 一个源文件的两个片段(section)之间。
2. 类声明和接口声明之间。
3. 两个方法之间。
4. 方法内的局部变量和方法的第一条语句之间。
5. 一个方法内的两个逻辑段之间，用以提高可读性。

6.5 分行

不允许把多条语句放在一行，不允许把多个变量定义在一行上。

7. 部分kotlin书写要求

7.1 判断语句

对于单条判断语句可使用 `if` 语句，多条判断语句尽可能使用 `when` 语句。
例如：

```
if (type == TYPE_CACHE) {
    // to do something
} else if (type == TYPE_DOWNLOAD) {
    // to do something
}
```

改写：

```
when (type) {
    TYPE_CACHE -> {
        // to do something
    }
    TYPE_DOWNLOAD -> {
        // to do something
    }
}
```

7.2 lambdas表达式

Lambdas 表达式在减少源文件中代码的总行数的同时，也支持函数式编程。
例如：

```
button.setOnClickListener { view ->
    startDetailActivity()
}
```

7.3 全局常量

Kotlin 允许开发者定义能在整个应用程序的所有地方都能够访问的常量。通常情况下，常量应该尽可能减小它们的作用域，但当需要有全局作用域的常量时，这是一种很好的方法，你不需要实现一个常量类：

```
package com.savvyapps.example

import android.support.annotation.StringDef

// Note that this is not a class, or an object
const val PRESENTATION_MODE_PRESENTING = "presenting"
const val PRESENTATION_MODE_EDITING = "editing"
```

这些全局常量可以在工程中任何地方访问：

```
import com.savvyapps.example.PRESENTATION_MODE_EDITING

val currentPresentationMode = PRESENTATION_MODE_EDITING
```

需要记住的是，为了减小代码复杂性，常量应该尽可能的缩小它的作用域。如果有一个常量值只和 user 类相关，那么应该将这个常量定义在 user 类的 companion 对象中，而不是通过全局常量的方式。

8. 参考资料

1. 《构建之法》（第三版）第四章节
2. kotlin官方文档之编码规范
3. Google Java Style（英文版）
4. Google Java编程风格指南（中文版）
5. Kotlin 在 Android 开发中的 16 个建议