**Android移动开发编码规范**

目录

[一、屏幕适配](#一、屏幕适配)

[3](#一、屏幕适配)

[图标尺寸3](#图标尺寸)

[度量单位3](#度量单位)

[单位规范4](#单位规范)

[二、代码布局4](#二、代码布局)

[基本格式5](#基本格式)

[源码格式7](#源码格式)

[三、注释8](#三、注释)

[块注释8](#块注释)

[单行注释8](#单行注释)

[尾端注释9](#尾端注释)

[行末注释9](#行末注释)

[四、命名规则10](#四、命名规则)

[缩写10](#缩写)

[命名方法12](#命名方法)

[包命名13](#包名命名)

[工具文件类命名14](#源文件的包命名)

[变量命名14](#变量命名)

[布尔型变量命名15](#布尔型变量)

[资源文件命名15](#资源文件命名)

[资源文件ID命名16](#资源ID命名)

[class源文件命名16](#类命名)

[接口命名16](#接口命名)

[方法命名16](#方法命名)

[五、类和接口16](#五、类和接口)

[类17](#类)

[接口17](#接口)

[六、异常处理18](#异常处理)

[错误的例子18](#异常错误的例子)

[正确的例子20](#异常正确的例子)

[七、版本控件20](#版本控制)

何时应该更新提交代码[21](#何时应该提交更新代码)

[目录22](#目录结构)

[提交代码22](#何时应该提交更新代码)

[冲突处理22](#冲突处理)

**前言**

编写本文档的初衷是为了进一步地规范Android编程风格，提高源文件的可读性、可靠性和可重用性，确保开发成员或开发团队之间工作可以顺利交接，以此保证花更少的时间浪费在已编写的源码上。以便继续维护与改进以前的工作，提高软件源程序的质量和可维护性，减少软件维护成本。

本规范的内容包括：屏幕适配、代码布局、注释、命名规则、表达式与语句、类、接口、异常处理、版本控制。

本规范分成规则性和建议性两种实行方式：对于规则性规范，要求所有开发成员严格执行；对于建议性规范，开发人员可以根据实际情况选择执行。

自本规范实施之日起，以后新编写的和修改的代码均应按本规范执行。

若文档过时，应当及时更新，进行补救。

1. 屏幕适配

屏幕适配是Android开发无法避免的难题，对普通程序员来说是个随时都会爆炸的炸弹，处理方式多种多样，现提出以下几点建议。

1. 图标尺寸

按实际生产要求，应当使用一套主流尺寸大小的图标，再使用比主流尺寸小并且是目前市场占有率最大的屏幕尺寸。如果有其他机型未能适配，再做特殊处理。

例子:目前主流是1080P的，目前小一点的手机屏幕使用最多的是720P的，那么。

使用一套1080P的图标，以及一套720P的图标。 按实际效果，若有未适配的图标，再按实际处理（代码控制或者替换为指定要求的图标）

1. 度量单位

DPI：常称为DP，解释为屏幕密度，运行在手机上时，会转换为PX进行显示。

SP：字体单位。（在某些特殊情况下为了适应更多屏幕分辨率，允许使用DPI，例子：在不同手机上字体大小差距明显（情况：三星定制的系统与国产定制系统））。

1. 单位规范

在布局xml文件中及类中使用到的度量单位，应当统一书写在dimes.xml资源文件中，便于重用、管理及维护。

注：尽量不使用代码控制控件、图片及字体的样式。

小结：屏幕适配多种多样，基本适配方式就这几种，主要精力应当就在处理特殊情况上。

1. 代码布局

代码布局的目的是显示出程序良好的逻辑结构，提高程序的准确性、连续性、可读性、可维护性。更重要的是，统一的代码布局和编程风格，有助于提高整个项目的开发质量，提高开发效率，降低开发成本。同时，对于普通程序员来说，养成良好的编程习惯有助于提高自己的编程水平，提高编程效率。因此，统一的、良好的程序布局和编程风格不仅仅是个人主观美学上的或是形式上的问题，而且涉及到产品质量，涉及到个人编程能力的提高，必须要引起重视。

如此复杂的布局是不是应该有插件呢，没错AS和EC都提供了code style（格式化代码的规范，如何使用将会在别一篇文档中提到）来帮助我们，并且不需要安装第三方插件，使用它能够方便快速的得到简洁的风格。

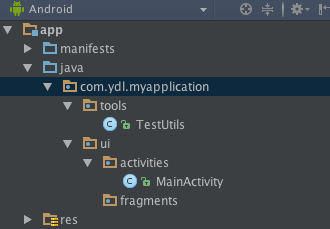
AS快捷键 ctril＋alt+L Mac平台command+alt+L

EC快捷键 ctril+shift+F

1. 项目架构

Android项目一般比较小，采用Androidd原来的设计模式，每个类按功能存放在各自包中。

例子：



1. 资源文件

按官方推荐的方式分类存放，

例如：

drawble:存放编写好的与控件相关xml文件，例子：控件的点击效果selector，控件的圆形背景shape等

mipmap:存放所有的图标

注：必须按此方式存放，否则将会造成无法正常使用资源。

1. 基本格式
2. 源代码文件（.java）的布局顺序是：包、import语句、类。

例子

Package com.yidingliu.dev.包名

import android.support.v7.app.AppCompatActivity;

public class ClassName {

}

1. if,else,else if,for,while,do,switch,foreach等条件语句，执行语句空一行紧跟其后（这样书写能被大家所公认，易于阅读）

例子：

if ( arg1 < arg2 ) {

arg1 = arg2 ;

}

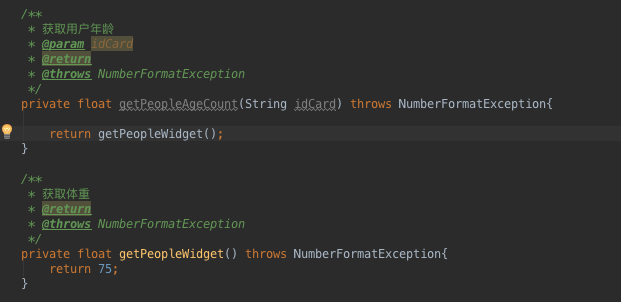
1. 关系密切的变量及常量应当放在一起，并且不相关的应当换行。（这样书写易于阅读和查找，代码整洁了不少）

例子：



1. 关系密切的方法应当放在一起，并且调用者应当处于被调用者上方。

例子：



说明：这两个方法同为操作用户，属于关系密切，放在一起；而且两者存在调用关系，所以被调用者（getPeopleWidget()）放在下面。

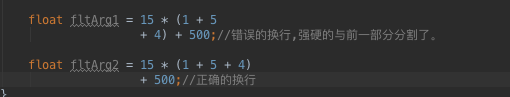
1. 代码审查

采用CheckStyle进入自动审查，使用方法及介绍将会在代码审查说明及使用（随本文档一起存放）中讲解()。

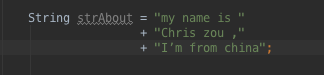
1. 源码格式
2. 源文件应当统一为指定编码：UTF-8
3. 源文件应当尽量避免超过500行（Android源码之所以难以阅读不仅仅是因为注释是英文与新技术，更有很大部分是因为代码行数太多了）（基类与实用方法除外）
4. 一行代码如果过长会难以阅读，应当在适当的地方进行换行。

换行规则：当前换行位置与前一行不是一个整体，此时允许换行。

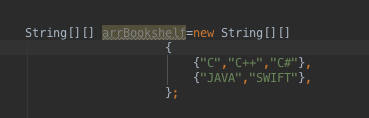
例子：



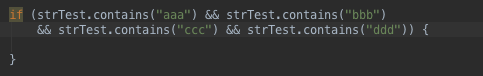
1. 拼接字符串



1. 数组



1. 条件语句



说明：其他类似对齐格式总是，均按此风格书写

注：代码风格基本都可以通过导入code style来进行格式化。（若有未格式化的请手动修改）

小结：注意没关联的变量换行，有关联的方法放在一起，再借助code style，将很轻松的编写出统一风格的高质量且易于阅读的源码。

1. 注释

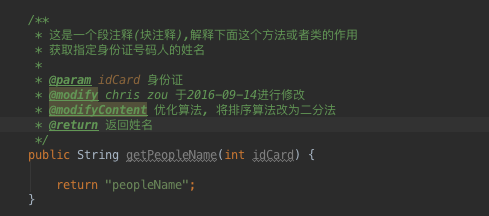
注释是每一个开发必备的素质，是帮助别人快速理解的桥梁，以及避免时间久远之后，忘记的风险。应当做到每个类每个方法及每个变量都应加上注释，通俗地说也就是：人在，注释在。

注释一共分了四种注释方式，而种类却只有两种，段注释与行注释

1. 块注释

块注释被置于每个文件或者方法之前。它们也可以被用于其他地方，比如方法的内部。

在功能和方法内部的块注释应该和他们所描述的代码具有一样的缩进格式。

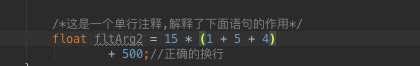


注：方法名上方/\*\* \*/块中为注释

1. 单行注释

单行注释可以显示在一行内，并与其后的代码具有一样的格式缩进层级。如果一个注释不能在一行内写完，就应该使用块注释。单行注释之前应当存在一个空格。

例子：



1. 尾端注释

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。

/Users/xiongweidong/Library/Containers/com.tencent.qq/Data/Library/Application Support/QQ/Users/894919296/QQ/Temp.db/773D10FB-02AA-49D8-8D77-013202B86050.png

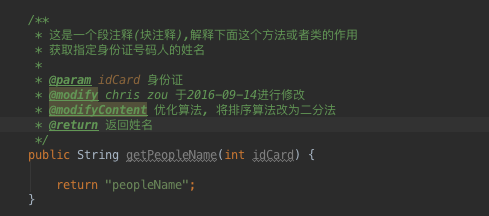
1. 行末注释

效果与尾注释类似，但是此种方法使用更方便，且能快速注释掉暂时不用的代码。

/Users/xiongweidong/Library/Containers/com.tencent.qq/Data/Library/Application Support/QQ/Users/894919296/QQ/Temp.db/5F89B0A7-A90B-4E5E-B4E5-ADE8DBCA3C93.png

重点：段注释用于方法时应当加上：方法作用、参数作用、返回值描述、修改人、修改日期、修改内容。

例子：



注：所有与业务逻辑相关且复杂不易理解的地方（包括变量、常量、语句等），都应当加上注释

1. 命名规则
2. 缩写

在实际开发过程中会发现，一些常用单词使用频繁，允许使用缩写。应使用大家所公认的缩写，应对某个框架系统使用的专用缩写进行统一的说明

若因业务要求，需要进行缩写，则应使用统一的缩略规则，Ex：number缩写为num，那么在整个系统中都应该采用该缩写

使用缩写时，应当按专业术语的要求大小写。Ex：System.IO，而不是System.Io.

例子：以下单词的缩写能够被大家所认可

|  |  |  |  |
| --- | --- | --- | --- |
| Icon | ic | Boolean | bln |
| Color | cl | **Short** | shr |
| BackGround | bg | **Double** | dbl |
| Activity | aty | **Shape** | sp |
| Selector | sl | **Single** | sng |
| String | str | **Object** | obj |
| Array | arr | **Temp** | tmp |
| Statistic | stat | **Increment** | inc |
| Argument | arg | **Buffer** | buf |
| Clear | clr | **Clock** | clk |
| Parameter | param | **Previous** | prev |
| Priority | prio | **Read** | rd |
| Ready | rdy | **Register** | reg |
| Schedule | sched | **Semaphore** | sem |
| Stack | stk | **Synchronize** | sync |
| Timer | tmr | **Trigger** | trig |
| Write | wr | **Text** | txt |
| Compare | cmp | **Configuration** | cfg |
| Display | disp | **enable** | en |
| Error | err | **Function** | fnct |
| Hexadecimal | hex | **Initialize** | init |
| Mailbox | mbox | **Manager** | mgr |
| Maximum | max | **Minimum** | min |
| Multiplex | mux | **Application** | app |
| Context | ctx | **Delay** | dly |
| Device | dev | **Disable** | dis |
| Flag | flg | **Message** | msg |
| Float | flt | **Object** | obj |

1. 命名方法
   1. （驼峰（Camel）命名法

驼峰命名法：又称小驼峰命名法，除了首字母以外，其他单词首字母全部大写。

Ex:

Private Context mContext;

Private String bookTitleStr

* 1. 帕斯卡（pascal）命名法

帕斯卡命名法：又称为大驼峰命名法，所有单词的首字母大写。

Ex:

public class IndexActivity

public class TestActivity

* 1. 匈牙利命名法

匈牙利命名法：在pascal命名法前加上小写的序列。

Ex:

Private String strBookTitle;

Private TextView tvBookTitle;

* 1. 下划线命名法

下划线命名法：所有单词之间用下划线分割开来。

Ex:

Private String book\_title;

Private int book\_total;

1. 根目录包名 命名

单词应当全部为小写，以”com.yidingliu.dev+项目名称“开头，之后再跟上包名名称。

项目名称：由项目经理拟定，统一使用，禁止私自内定

Ex:

com.yidingliu.dev.项目名+包名

1. 存放类的包名命名

单词应当全部为小写，以存放的内容的属性的复数单词来规定。

Ex:

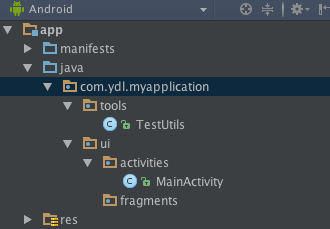
存放工具的命名：tools

特殊情况：

页面分为activity与fragment，他们同属UI，那么除了以上的命名方式以外，应当再给这两种类再放在同一个包里面。

Ex:

<ui>-[[activities]、[fragments]]



1. 通用工具类说明及命名

说明：每个工具类都应加上详细注释[注释格式](#三、注释)。工具类分为通用工具类和针对某个的工具类（简称工具类）。

通用工具类：适用所有类，不针对某个类而特定使用的。

工 具 类：针对某个类而专门提供的工具。

命名：所有工具类应当加上后缀名Utils加以修饰。

注：通用工具类禁止使用单模式，应当使用公共静态方法puhlic static （针对某个类工具类除外）。

1. 变量 采用[匈牙利命名法](#匈牙利命名法)

EX:

Private String strBookTitle;

Private TextView tvBookTitle;

1. 布尔型变量

布尔型变量只有两种状态 是与否，应当在前面加上is来代表是与否的状态。

Ex:

Private Boolean isVisible;

Private Boolean isShow;

1. 资源文件（mipmap,drawable,layout）只能采用下划线命名法

Ex:

图标：

ic\_back\_sl //选中的图标状态

ic\_back\_unsl //未选中的图标状态

bg\_index\_main //首页的主背景

drawable : sl\_back

layout:应当用类型前缀进行标识

activity\_main、 app\_content(app：主页面)、app\_title

1. 资源文件ID（color,string,dimen,menu,style,anim） 驼峰命名法

所有资源文件内容有相联的命名应当加上前缀加以区分

例子：程序主题的字体大小: TitleSizeBold、TitleSizeSmall，同为控件文字大小的应当加上text前缀进行区分

textTitleSizeBold、textTitleSizeSmall

例子：

color：white

string：appName

dimen : acticonSize

menu : index

anim : pageOut

注：style单独处理，应当在根主题名后加上点号再加上风格名

例子：标题的style=根主题（AppTheme）+“.”+“TitleStyle”

AppTheme.TitleStyle

小结：更多实际命名，请查看通用库命名方式。

1. Class源文件 采用pascal命名法

每个类应如果与其他类有关联，应当使用一致的命名。

Ex:首页的activity与model

Activity命名为：public class IndexActivity

Model命名为：public class IndexBean

说明：用index关键将两个类关联起来，使用查找。

1. 方法 命名

方法是用来做动作的，所以命名应当见名之义,用词准确，并且在应当遵从：动词+名词+宾语的规则，且命名应当使用驼峰命名法。

Ex: 设置一本书的标题应该为：动词set+名词book+宾语title

Private void setBookTitle()；

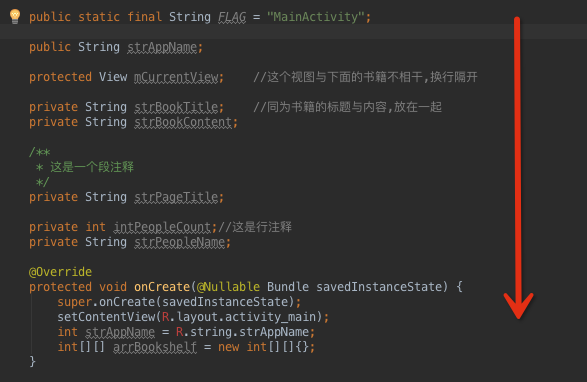
Ex:显示页面的标题应当为：动词show+名词page+宾语title

Private void showPageTitle();

1. 类和接口
2. 类
3. 结构（具体详情请查看通用库源码结构）

类的书写应当按照从常量、变量、方法的顺序从上往下书写。

例子：



说明：顺序可分解为public final常量->public公开变量->private 私有变量->方法

1. 每个类都需要加上CopyRight(只需要导入一个xml文件将自动生成)
2. 语法

请查看[二、代码布局](#二、代码布局)

1. 接口

接口分为普通接口与监听。

对于监听：在观察者模式中，顾名思义，控件的回调，监听控件的状态；这类接口命名时应当加上后缀Listener

例子：控件的OnClickListener，OnTouchListener等监听控件的回调接口

对于普通接口：除了监听，其他都属于普通接口；这类接口命名时应当加上后缀Callback

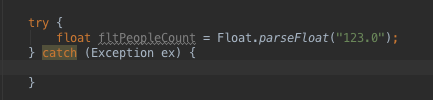
例子：线程的回调接口。

1. 异常处理

异常捕捉是程序中非常重要的一部分，这部分做得好，对于个程序的稳定性、可维护性、质量都会有极大的提升，以下将对异常捕捉进行说明：

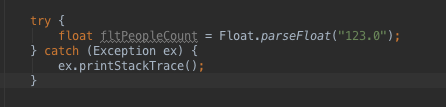
错误的例子：

1. try,catch不能使用空的处理。



说明：有时候疏忽了，忘了进行异常输出，随时留下了一颗定时炸弹。爆炸了还找不到原因

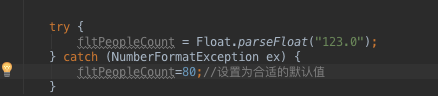
1. 不能使用广泛的异常捕捉（别人的开源代码除外）。



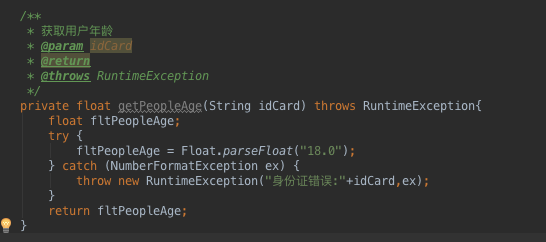
说明：这样写挺诱人的，方便，但是未使用明确的异常限定，出错了，不太容易排查。

以下的例子是可以被接受的：

1. 在错误的地方，替换为合适的值

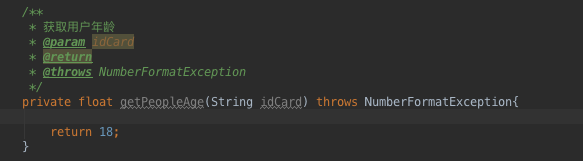


1. 抛出新异常，并附上一条新的消息提示



说明：此种方式需要明确的知道会出哪种错误。

1. 将异常处理抛给调用者



1. 版本控制（GIT）

版本控制的作用在每一个项目中，其作用不言而喻，没有版本控制就不能团队协作开发，而且出现重大事故时无法及时的还原。所以本小节讲述了如何正确的提交代码：

1. 何时应该提交更新代码
2. 更新

每天早上上班，开始准备工作时，应当及时更新，避免产生冲突或者重复工作（别人写了，你又写一遍）

1. 提交

每天下班之前，应当及时的提交代码，详情见下文[如何提交代码](#如何提交代码)

1. 目录结构

~~所有工程源码应当放在”Code->Mobile->Android”目录中。~~

1. 如何提交代码（重点）

为了产生不必要的麻烦，提交代码之前应当做的工作是，先更新！先更新！先更新！（重要的事情说三遍），而且应当键入注释，说明本此提交代码的作者、描述。

注释规则为：作者+”|”+描述

例子：chris|新增图片工具类

不需要提交的代码：

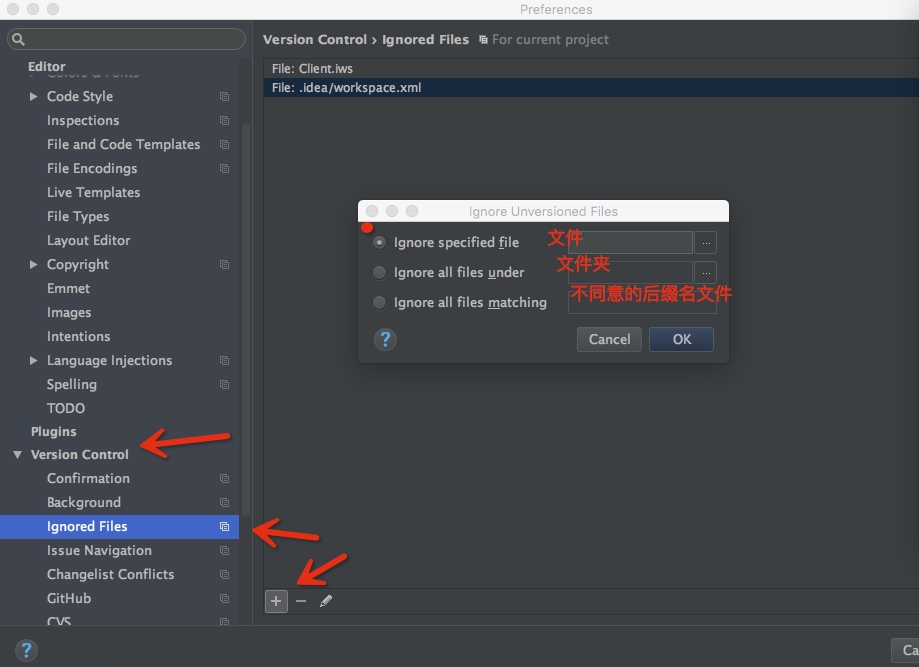
1、build文件夹内的所有文件、

2、.idea、

3、.iml后缀名的代码

4、所有的Test文件夹

自动过渡指定文件及文件夹方法：



注：提交完成之后，应当提醒其他及时更新代码

1. 冲突处理

在多个人同时操作同一个源文件时，将会产生冲突。应当联系同样操作了该文件的作者，协同处理。

注：禁止只同意自己的源码

[再看一遍](#_top)