从这篇文章开始我们暂停一下对android源码的分析，开始讲一下android产品研发中一些常用的技术，技巧，方法，实践等姿势。这里需要强调的是我们所讲解的这些东西可能对产品开发中比较常用的，因为对于项目开发中，可能更多的强调管理，进度方法的东西，对工程化的东西比较强调，而我们这里更多的是对产品技术方面的归纳总结。

而本文中选择将开发规范作为这个系列的第一篇文章，就是个人感觉产品研发过程中，开发规范真的很重要，很重要，非常重要（重要的事情说三遍），一个好的开发规范可以让团队中的人对他人的代码更熟悉，新人也可以更好的了解产品的业务逻辑。开发规范并不是一个死的一成不变的，每个团队可能都有自己的开发规范，只要是适合团队的开发规范就是最好的开发规范。

所以本文中所讲解的开发规范只能是抛砖引玉，有可取的地方可以借鉴，引用，不能照搬全抄不假思索，毕竟不同的团队有不同的实际情况。最好的方式就是可以根据本文的开发规范总结出自身团队比较适合的规范流程。

好吧，废话不多说了，下面我们就介绍一下我在实践中总结的android开发规范。

---

1 前言

1.1 为什么需要开发规范

```
编码规范对于程序员而言尤为重要，有以下几个原因：
* 一个软件的生命周期中，80%的花费在于维护
* 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
* 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
* 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 开发规范的作用

```
* 减少维护花费
* 提高可读性
* 加快工作交接
* 减少名字增生
* 降低缺陷引入的机会
```

2 命名规范

2.1 常量命名规范

2.1.1 类型

常量命名规范

2.1.2 说明

常量用于保存需要常驻内存中并且经常使用变化不多的数据，定义常量的名称的时候需要遵循望文知意的原则；

2.1.3 规则

- 1. 全部为大写字母；
- 2. 中间以“_”连接；
- 3. 望文知意原则；

2.1.4 备注

代码中涉及到直接使用某个字符串或者其他基本类型的值时，建议定义成常量，避免多处直接使用同样的值作为参数。

2.1.5 举例

- 如：定义一个常量表示最小屏幕宽度的常量，则可以定义一个int类型的常量，该常量可以命名为：“MIN_SCREEN_WIDTH“；
- 其他举例：
- 例如：static final int MIN_SCREEN_WIDTH = 4；(√)
- 例如：static final int min_screen_width = 4;（×）
- 例如：static final int minScreenWidth = 4; （×）
- 例如：static final int WIDTH = 4； （×）
- 例如：static final int width = 4； （×）
- 例如：static final int wd = 4； （×）

2.2 变量命名规范

2.2.1 类型

变量命名规范

2.2.2 说明

变量用于保存系统中的临时数据，变量命名时遵循望文知意，简单明了，驼峰标示等原则。

2.2.3 规则

1. 首字母大写；
2. java驼峰命名；
3. 望文知意原则；
4. 推荐引用类型变量添加前缀“m”；
5. 如果是View组件变量，则组件名称为xml文件中定义的ID名称去掉下划线，下划线后一位大写；

2.2.4 备注

无

2.2.5 举例

- 如：定义一个表示最小屏幕宽度的变量，则可以定义一个int型的临时变量为：mMinScreenWidth；
- 例如：static final int mMinScreenWidth = 4；（√）
- 例如：static final int minWidth = 4；（×）
- 例如：static final int screenWidth = 4；（×）
- 例如：static final int width = 4；（×）
- 例如：static final int min = 4；（×）
- 例如：static final int msw = 4；（×）

2.3 方法命名规范

2.3.1 类型

方法命名规范

2.3.2 说明

方法名的命名应该遵循简单明了的原则；

2.3.3 规则

1. 首字母小写；
2. java驼峰命名；
3. 简单明了原则；
4. 初始化View方法init*（每个init做一件事）

2.3.4 备注

- 同时在方法的实现上，尽量不要在一个方法中出现太多实现代码，如一个方法有几百行的实现逻辑，推荐在逻辑复杂时，按功能点拆分出多个方法，便于阅读。
- 另外，出现功能一样的实现逻辑，尽量抽取公用方法，避免将实现逻辑复制到多个用到的地方。

2.3.5 举例

- 如：定义一个获取屏幕宽度的方法，依照上述原则，则可以定义为一个静态方法：public static int getScreenWidth();
- 例如：public static int getScreenWidth();(√)
- 例如：public static int getscreenwidth();（×）
- 例如：public static int getScreenwidth();（×）
- 例如：public static int getWidth();（×）
- 例如：public static int getScreen();（×）
- 例如：public static int getSW();（×）

2.4 类命名规范

2.4.1 类型

类命名规范

2.4.2 说明

类名主要表示一个类的作用，需要简明扼要，望文知意，并且首字母大写。

2.4.3 规则

1. 首字母大写；
2. java驼峰命名；
3. 望文知意原则；
4. 能够说明类的功能和主要作用（注释的作用）；
5. Acitivity类以Acitivity结尾；
6. Fragment类以Fragment结尾；
7. Service类以Service结尾；
8. BroadcastReceiver类以Receiver结尾；
9. ContentProvider类以Provider结尾；
10. Application类以Application结尾；
11. 自定义View类以Custom**View结尾；
12. 自定义Adapter类以Adapter结尾；
13. adapter中的ViewHolder以Holder结尾；
14. 实体Bean以Model结尾；

2.4.4 备注

无

2.4.5 举例

- 如：定义一个获取屏幕信息的工具类，则可以定义为public class ScreenUtils；
- 例如：public class ScreenUtils; (√)
- 例如：public class Screenutils; （×）
- 例如：public class Screen; （×）
- 例如：public class screenutils; （×）
- 例如：public class screen; （×）
- 例如：public class su； （x）

2.5 接口命名规范

2.5.1 类型

接口命名规范

2.5.2 说明

接口命名需要简单明了，长度不宜过长；

2.5.3 规则

1. 首字母大写（第二个字母也是大写）；
2. java驼峰命名；
3. 望文知意原则；
4. 建议在名称前面追加“I”；

2.5.4 备注

- I**Listener
- I**CallBack
- I**；

2.5.5 举例

- 如：定义一个activity的方法接口，实现接口中的某些方法：public interface IFunctionListener；
- 例如：public interface IFunctionListener；(√)
- 例如：public interface BaseActivity; （×）
- 例如：public interface Baseactivityinter; （×）
- 例如：public interface BaseInter; （×）
- 例如：public interface ActivityInter； （×）

2.6 包名规范

2.6.1 类型

包名规范

2.6.2 说明

用于分类管理类文件；

2.6.3 规则

1. 所有字母小写；
2. 简单明了，层级很深，没有拼接的包名；
3. 望文知意；
4. 按功能划分包名，如“我的”
5. 工具类可以划分为一个工具类的包名，utils，里面可以添加包名层级；
6. 系统类的可以划分为一个系统类的包，system，里面可以添加包名层级；
7. 组件类的可以划分为一个组件类的包，***，里面添加adapter的包名，自定义view包名；
8. Service类的可以划分为一个服务类的包，service，里面可以添加包名层级；
9. 数据库相关类可以划分为一个数据库类，db，里面可以添加数据库相关类，Bean类，数据库服务类等；
10. 广播类的可以划分为广播类的包，receiver，可以放一些广播相关的类；
11. 网络类相关的可以划分为，network，放一些网络相关的类；
12. Fragment类存放在fragment包下；
13. Activity类存放在Activity包下；

2.6.4 备注

无

2.6.5 举例

无

2.7 目录名称规范

2.7.1 类型

目录名称规范

2.7.2 说明

主要是一些jar包，so文件的配置目录名称；

2.7.3 规则

1. 全部为小写字母；
2. 简单明了；
3. 望文知意；
4. 驼峰表示；

2.7.4 备注

无

2.7.5 举例

- 后期增加目录的可能性不多，现列举出系统中存在的目录结构：
- lib：第三方jar的保存路径；
- jniLibs：jni引用的so文件的目录；

2.8 布局文件名称规范

2.8.1 类型

布局文件名称规范

2.8.2 说明

主要包含资源文件的命名问题；

2.8.3 规则

1. 全部为小写字母；
2. 中间以"_"连接；
3. 望文知意原则；
4. 布局文件的开头问类名；
5. 列表项的xml布局文件名称：类名_item.xml；
6. activity类的xml文件名称：类名_activity.xml；
7. fragment类的xml文件名称：类名_fragment.xml；
8. 自定义View的xml文件的名称：类名_父类名.xml；

2.8.4 备注

无

2.8.5 举例

如：如定义H5Activity的xml文件名称，则可以定义为h5.xml；尽量不使用大写字母等。

2.9 drawable文件名称规范

2.9.1 类型

drawable文件名称命名规范

2.9.2 说明

主要包含资源文件的命名问题；

2.9.3 规则

1. 全部为小写字母；
2. 中间以"_"连接；
3. 望文知意原则；
4. 布局文件的开头问类名；
5. 11_22_33_44，44：selector，shape（大概五六个，暂时不定义其他的）； 33：src、bg、color（可扩展，可为空）； 22：状态名称或者为空；11：业务名称

2.9.4 备注

无

2.9.5 举例

无

2.10 资源ID规范

2.10.1 类型

资源ID命名规范

2.10.2 说明

各种资源ID的定义问题；

2.10.3 规则

1. 全部为小写字母；
2. 中间以"_"连接；
3. 望文知意原则；

2.10.4 备注

可以考虑按照组件的名称的缩写作为前缀，（同一个xml文件中ID名称不能重复）如：组件简写（大写字母缩写）_业务名称 TextView的组件:tv_pay_money Button的组件：btn_pay_money EditText的组件：et_user_name LinerLayout组件：ll_container

2.10.5 举例

如：比如一个textview组件，可点击用于支付的按钮，则可以把ID定义为： tv_pay_money；

3 注释规范

3.1 类注释

在类、接口定义之前当对其进行注释，包括类、接口的目的、作用、功能、继承于何种父类，实现的接口、实现的算法、使用方法、示例程序等。

```
/**
 * author:作者
 * time:时间
 * desc:描述
 */
```

3.2 方法注释

方法注释的模板：

```
/**
 * desc:描述
 * @param 参数名 参数描述
 * @param 参数名2 参数描述
 * @return 返回值类型说明
 * @throws Exception 异常说明
 */
```

3.3 类成员变量和常量注释

成员变量和常量需要使用如下注释的形式，注释位于变量的上侧；

```
/**
```

* **/

3.4 内部逻辑注释

```
内部逻辑注释模板：
//支付成功
if (response.getRet() == 0) {
   Toast.makeText(H5Activity.this, "支付成功", Toast.LENGTH_LONG).show();
   goToNext(response);
}
//支付失败
else if (response.getRet() == -1) {
  Toast.makeText(H5Activity.this, "支付失败", Toast.LENGTH_LONG).show();
  //刷新当前页面
  reflush(currentUrl);
}
```

4 代码顺序

4.1 代码顺序

```
在一个典型的Activity中代码的顺序如下：
/**
* author:sh
* desc:该class的作用
* time:yyyy-MM-dd
**/
public class ClassName {
    //（1） 成员变量集合
    //（2） 回调方法集合
    若该类为activity，则：onCreate、**、onDestory；
    若该类为Fragment、则：onCreateView、**、onDestory；
    //（3） 其他方法集合
}
```

5 代码风格

5.1 大括号换行

```
左大括号不换行，右大括号换行；
class MyClass {
    int func() {
        if (something) {
            // ...
        } else if (somethingElse) {
            // ...
        } else {
            // ...
        }
    }
}
```

5.2 小括号空格

```
if (condition) {
    body();
}                          // 推荐
```

5.3 缩进

- 4 个空格作为缩进排版的一个单位，不使用制表符 tab。
- 8 个空格作为换行后的缩进，包括函数调用和赋值。
- Instrument i =

```
someLongexpression_r(that, NotFit, on, one, line);    // 推荐
```

5.4 每一行的长度

- 尽量避免一行的长度超过 100 个字符。
- 例外：如果注释行包含了超过 100 个字符的命令示例或者 url 文字，为了便于剪切和复制，其长度可以超过 100 个字符。
- 例外：import 行可以超过限制，因为很少有人会去阅读它。这也简化了编程工具的写入操作。

5.5 每次声明一个变量

- 推荐一行一个声明，因为这样以利于写注释；
  int level; // indentation level
- int size;   // size of table

5.6 if-else语句

```
if-else语句应该具有如下格式：
if (condition) {
    statements;
}
```

```
if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else{
    statements;
}
注意：if语句总是用”{“和”}“括起来，避免使用如下容易引起错误的格式：
if (condition)  // 避免
    statement;
```

5.7 for语句

```
一个for语句应该具有如下格式：
for (initialization; condition; update) {
    statements;
}
当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。
若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句）使用单独的语句。
```

5.8 while语句

```
一个while语句应该具有如下格式：
while (condition) {
    statements;
}
```

5.9 do-while语句

```
do {
    statements;
} while (condition);
```

5.10 switch语句

```
一个switch语句应该具有如下格式：
switch (condition) {
    case ABC:
        statements;
```

```
        /* falls through */
    case DEF:
        statements;
        break;

    case XYZ:
        statements;
        break;

    default:
        statements;
        break;
}
每当一个case顺着往下执行时（因为没有break语句），通常应在break语句的位置添加注释。
```

6 异常规范

6.1 异常名称

定义异常的时候，异常的后缀名称以Exception结尾，及**Exception；

6.2 异常描述

尽量英文描述，简单明了；

6.3 异常格式

```
一个try-catch语句应该具有如下格式：
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}

try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

7 其他规范

7.1 源文件的函数小于2K

一般来说源文件的行数不能大于2K行，过多的话可以考虑拆分功能，拆分函数等；

7.2 使用TODO注释

- 对那些临时性的、短期的、够棒但不完美的代码，请使用 TODO 注释。
- TODO 注释应该包含全部大写的 TODO，后跟一个冒号：
- // TODO: Remove this code after the UrlTable2 has been checked in.
- // TODO: Change this to use a flag instead of a constant.
  如果 TODO 注释是“将来要做某事”的格式）。

7.3 使用自定义LOG

在系统中需要打印LOG的时候，尽量使用自定义的LOG，自定义的LOG在开发环境的时候会打印日志，正式环境的时候不会打印日志。

7.4 使用自定义TAG

在系统打印LOG的时候，使用TAG尽量使用tab，同意的TAG标志。

另外对产品研发技术，技巧，实践方面感兴趣的同学可以参考我的：
android产品研发（一）-->实用开发规范