Android 开发规范

Java 部分

命名规范

1. 代码中的命名不能以**下划线或美元符号**开始，也不能以**下划线或美元符号**结束。

   name / $Object / name / name$ /Object$

2. 代码中的命名禁止使用拼音与英文混合的方式，不允许直接使用中文；可以使用纯拼音，建议使用纯英文的命名方式。

3. 类名使用 UpperCamelCase 风格，必须遵从驼峰形式。

   UserInfo / UserInfoActivity / UserInfoFragment

4. 方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

   userInfo / getUserInfo() / userId

5. 类的命名尽量根据类的实际用途来进行命名

   UserInfoActivity / LoginActivity / HomeFragment

常量定义

1. 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要在意名字长。

   MAX_USER_COUNT

2. 常量定义

   - 不允许出现任何魔法值(即未经定义的常量)直接出现在代码中；
   - 不建议使用一个常量类来维护所有的常量，应该按常量功能进行归类，分开维护。

格式规范

1. 随时使用 Android Studio 中的代码格式化，保证代码的排版。

2. 代码过长必须换行显示，以 Android Studio 中的基线为准。其中，
    - 方法调用的点符号与下文一起换行；
    - 在多个参数超长，逗号后进行换行；
    - 在括号前不宜换行。
3. Object 的 equals 方法容易抛空指针异常，应使用常量或确定有值的对象来调用 equals。

```
”test ”.equals(object);
```

4. 在数据实体类中，由于后台返回的数据存在 **null**，所以使用 GsonFormat 生成的实体类，需要对 getter 方法做判断；

```
public String getPages() {
     return pages ==null:0?pages;
 }
```

5. 在 if / else / for / while / do 语句中必须使用大括号，即使只有一行代码，避免使用下面的形式：

```
if (condition) statements;
```

6. 在一个 switch 块内，每个 case 要么通过 break / return 等来终止,要么注释说明程序将继续执行到哪一个 case 为止;在一个 switch 块内,都必须包含一个 default 语句并且放在最后,即使它什么代码也没有。
7. 推荐尽量少用 else, if - else 的方式可以改写成：

```
if(condition){
        ...
        return obj;
}
// 接着写 else 的业务逻辑代码;
```

    如果非得使用 if()…else if()…else… 方式表达逻辑,请勿超过 3 层,超过请使用状态设计模式。逻辑上超过 3 层的 if-else 代码可以使用 **卫语句**，或者 **状态模式** 来实现。
8. 当 if 语句内的逻辑判断相当复杂的时候，建议将逻辑判断定义成一个变量再来使用。

```
    //伪代码如下
    boolean existed = (file.open(fileName, "w") != null) && (...) ||
(...);
    if (existed) {
        ...
    }
```

注释规范

1. 所有的类注明创建者、创建日期、类的信息。

```
    /**
    * @author T.K
    * @time 2018/4/10 20:05
    * @description 地址管理
    */
```

2. 类、类属性、类方法的注释必须使用 Javadoc 规范，使用/*内容*格式，不得使用 // xxx 方式。

3. 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/ /注释，注意与代码对齐。

4. 代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。

5. 注释掉的代码尽量要配合说明，而不是简单的注释掉。代码被注释掉有两种可能性：

    - 后续会恢复此段代码逻辑。
    - 永久不用。

    前者如果没有备注信息，难以知晓注释动机。后者建议直接删掉(代码仓库保存了历史代码)。

Android 部分

Android 资源文件命名与使用

1. layout 文件的命名方式。

    - Activity 的 layout 以 module_activity 开头；
    - Fragment 的 layout 以 module_fragment 开4头；
    - Dialog 的 layout 以 module_dialog 开头；
    - include 的 layout 以 module_include 开头；
    - RecyclerView 的 item layout 以 module_recycle_item 开头；

2. drawable 资源名称以小写单词+下划线的方式命名，根据分辨率不同存放在不同的 drawable 目录下，如果介意包大小建议只使用一套，系统去进行缩放。采用规则如下：

   > 模块名_业务功能描述_控件描述_控件状态限定词。 如：module_login_btn_pressed,module_tabs_icon_home_normal

3. anim 资源名称以小写单词+下划线的方式命名，采用以下规则：

   > 模块名_逻辑名称_[方向|序号]

   Tween 动画（使用简单图像变换的动画，例如缩放、平移）资源：尽可能以通用的动画名称命名:

   > 如 module_fade_in , module_fade_out , module_push_down_in (动画+方向)。

   Frame 动画（按帧顺序播放图像的动画）资源：尽可能以模块+功能命名+序号:

   > 如module_loading_grey_001。

4. color 资源使用 #AARRGGBB 格式，写入 colors.xml 文件中，命名格式采用以下规则：

   > xxx_颜色，如
   >
   > ```
   > <color name="white_color">#ffffffff</color>
   > ```

5. dimen 资源以小写单词+下划线方式命名，写入 dimens.xml 文件中，采用以下规则：

   > value_dimens，如
   >
   > ```
   > <dimen name="1_dimens">1dp</dimen>
   > ```

6. style 资源采用“父 style 名称.当前 style 名称”方式命名，写入 styles.xml 文件中，首字母大写。

7. string资源文件或者文本用到字符需要全部写入 strings.xml 文件中，字符串以小写单词+下划线的方式命名，采用以下规则：

   > 模块名_逻辑名称。 moudule_login_tips,module_homepage_notice_desc

8. id 资源原则上以驼峰法命名，View 组件的资源 id 建议以 View 的缩写作为前缀。常用缩写表如下：

控 件	缩 写
LinearLayout	ll
RelativeLayout	rl
ConstraintLayout	cl
ScollView	sv
TextView	tv
Button	btn
ImageView	cb
RecyclerView	rv
CheckBox	cb
RadioButton	rb
EditText	et

其它控件的缩写推荐使用小写字母并用下划线进行分割，例如：ProgressBar 对应的缩写为 progress_bar；DatePicker 对应的缩写为 date_picker。

9. 图片根据其分辨率，放在不同屏幕密度的 drawable 目录下管理，否则可能在低密度设备上导致内存占用增加，又可能在高密度设备上导致图片显示不够清晰。 为了支持多种屏幕尺寸和密度，Android 提供了多种通用屏幕密度来适配。常用的如下。

密度	值	文件大小
ldpi	120dpi	
mdpi	160dpi	48 x 48
hdpi	240dpi	72 x 72
xhdpi	320dpi	96 x 96
xxhdpi	480dpi	144 x 144
xxxhdpi	640dpi	

UI 与布局

1. Android 项目中，字体大小一律使用 dp 表示。
2. 布局中不得不使用 ViewGroup 多重嵌套时，不要使用 LinearLayout 嵌套，改用 RelativeLayout，可以有效降低嵌套数。
3. 灵活使用布局，推荐 include、 merge、ViewStub 来优化布局，尽可能多的减少 UI 布局层级，推荐使用 FrameLayout，LinearLayout、RelativeLayout 次之。
4. 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。
5. 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。即使需要连续弹出 Toast，也应避免直接调用 Toast#makeText。
6. 禁止在非 UI 线程进行 View 相关操作。
7. 禁止在设计布局时多次为子 View 和父 View 设置同样背景进而造成页面过度绘制，推荐将不需要显示的布局进行及时隐藏。
8. 不能在 Activity 没有完全显示时显示 PopupWindow 和 Dialog。
9. 在 Activity 中显示对话框或弹出浮层时，尽量使用 DialogFragment，而非 Dialog/AlertDialog，这样便于随Activity生命周期管理对话框/弹出浮层的生命周期。

```
    public void showPromptDialog(String text) {
        DialogFragment promptDialog = new DialogFragment() {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup
container, BundlesavedInstanceState) {
            getDialog().requestWindowFeature(Window.FEATURE_NO_TITLE);
            View view = inflater.inflate(R.layout.fragment_prompt,
container);
            return view;
        }
    };
    promptDialog.show(getFragmentManager(), text);
    }
```

进程、线程与消息通信

1. 不要通过 Intent 在 Android 基础组件之间传递大数据（binder transaction缓存为 1MB），可能导致 OOM。
2. 线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

3. 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。

4. 新建线程时，定义能识别自己业务的线程名称，便于性能优化和问题排查。

文件与数据库

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

2. 当使用外部存储时，必须检查外部存储的可用性。

```
// 读/写检查
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
// 只读检查
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

3. 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。

4. 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()。

> SharedPreference 相关修改使用 apply 方法进行提交会先写入内存，然后异步写入磁盘， commit 方法是直接写入磁盘。如果频繁操作的话 apply 的性能会优于 commit，apply 会将最后修改内容写入磁盘。但是如果希望立刻获取存储操作的结果，并据此做相应的其他操作，应当使用 commit。

5. 数据库 Cursor 必须确保使用完后关闭，以免内存泄漏。

应用安全

1. 将 android:allowbackup 属性必须设置为 false，阻止应用数据被导出。
2. 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。
3. 不要把敏感信息打印到 log 中。
4. 确保应用发布版本的 android:debuggable 属性设置为 false。
5. 本地加密秘钥不能硬编码在代码中，更不能使用 SharedPreferences 等本地持久化机制存储。应选择 Android 自身的秘钥库（KeyStore）机制或者其他安全性更高的安全解决方案保存。

插件

1. 建议统一使用 ButterKnife，来减少 findViewById 的使用。
2. 从后台请求回来的 Json 数据来生成数据实体类时统一使用 GsonFormat，除自己定义的外。

版本控制

1. 代码完成后需要及时提交到 svn 上。
2. 每天下班前必需要往 svn 提交代码。
3. 在修改小组其他成员的代码时，需要及时沟通，避免提交时出现冲突。
4. 新建新项目时，由组长完成新建工作，并完成忽略文件配置，提交到 svn 上后，通知组员去 checkout 项目。组员 checkout 好项目后，导入项目完成后也需要完成一次忽略文件配置操作。需要添加的忽略文件如下：
    - idea文件夹
    - .gradle文件夹
    - 所有的build文件夹
        - Mask:build 表示忽略所有build文件夹，包括所有Module的build文件夹；
    - 所有的.iml文件
        - Mask:*.iml 表示忽略所有iml格式的文件。
    - local.properties文件。

    注意一点，配置忽略文件必须在Share到SVN之前进行，如果在Commit后配置，貌似就不起作用了。

5. 提交代码需要写好注释，格式如下：

```
日期：xxxx/xx/xx
提交人：xxx
提交内容：xxxxx
```