阅读目录

回到顶部

1). 类名，接口名：
以大写开头，如果一个类的类名由多个单词组成，所有单词的首字母必须大写，单词尽量写全称，不要简写，除非约定俗成的名字，例如：URL，RTMP，RTSP 这些广泛使用的专有名词，可以全部大写，也可以首字母大写。
例如 HttpRequest，CourseActivity

2). 局部变量，类的成员变量，类的成员函数，函数参数：
以小写字母开头其他的单词首字母大写，变量名不建议使用下划线分隔单词，建议使用驼峰命名法，Android的系统类都采用此方法。
例如 toString() onCreateView(Bundle savedInstanceState)

3). 静态常量：全部大写，单词之间使用下划线分开，常量单词全部大写，所以单词之间使用下划线分隔。
例如 WHAT_EMPTY_CONTENT

4). 控件变量的命名，控件的ID命名：
建议：xml布局文件中的控件的id的命名与*.java的代码文件中的控件对象的命名一致。

```
1
2
3
4
5
6
7
```

```
class MyActivity extends Activity{
TextView txtUserName;
...
protected void onCreate(Bundle savedInstanceState) {
```

```
txtUserName = (TextView) findViewById(R.id.txtUserName);
}
}
```

5). 常用控件以及类对象命名的规范说明(红色部分为建议的前缀或者后缀)：

类名	变量名	类名	变量名
TextView	txtDescription	ProgressBar	progressDescripti
Button	btnDescription	SeekBar	seekBarDescriptic
ImageButton	imgBtnDescription	VideoView	vvDescription
ImageView	imgDescription	Spinner	spinDescription
RadioButton	rbDescription	WebView	webViewDescript
EditText	editDescription	ListView	listViewDescriptic
ScrollView	scrollDescription	GridView	gridDescription
Handler	descriptionHandler	RatingBar	ratingBarDescript
PullToRefreshListView	pullRefreshViewDescription	Adapter	descriptionAdapt
Fragment	descriptionFragment	Activity	descriptionActivit
List	descriptionList	Map<>	mapDescription
SlidingMenu	slidMenuDescription	ViewPager	viewPagerDescrip
CheckBox	chBoxDescription	View	viewDescription
RadioGroup	rgDescription	ExpandableListView	expDescription
FrameLayout	frameLayDescription	SharedPreferences	spDescription
LinearLayout	lineLayDescription	RelativeLayout	relativeLayDescrip
startActivityForResult(requestCode)	REQUEST_CODE_DESCRIPTION	msg.what	WHAT_DESCRIPT

6). 资源命名：

layout资源文件的命名(全部小写，下划线分隔)：

activity的资源文件：activity_description1_description2.xml

fragment的资源文件：fragment_description1_description2.xml

listview列表项的资源文件：list_item_description1_description2.xml

可复用(被include)的组件资源文件: control_description1_description2.xml
drawable资源： controlName_description1_description2_selector.xml
controlName表示该资源要用在什么类型的控件上面，例如如果是按钮的图片切换则
应该这么定义 button_bg_sendmessage_selector.xml
selector表示该资源的形式，例如还有shape等
图片资源的名字：同上
颜色值的命名： color_description 以color为前缀，全部小写，下划线分隔。description既可以是该颜色值使用的功能描述，也可以是该颜色值的英文描述，也可以是具体的颜色值，例如：
#ffffff
#cccccc
#dddddd
因为grey可能有很多等级，有时候需要不同等级的灰色，没有那么多英文名可以区分，所以名字中可以直接使用颜色值
#4c4c4c 根据功能定义description，表示该颜色用于按钮被按下

注：不允许出现毫无意义的命名，例如textview1,textview2

回到顶部

代码中不允许出现直接硬编码的字面常量，如果是控件上面显示的文本，必须放在strings.xml资源文件中。 如果是代码中用到常量字符串，必须定义成 public static final String类型的常量值，在代码中使用该定义的常量值。这样做的好处是以后需要修改该常量值，只需要修改一个地方。如果是硬编码在代码中则要修改所有使用它的地方，而且拷贝容易出错。在Activity之间传递参数的时候，intent.putExtra 的key值也要命名规范，并且统一定义为静态常量，不能直接硬编码在代码中，否则想要修改的时候很麻烦。某一个Activity在被启动的时候需要接受参数，那么这些参数的key定义就应该放在该Activity中。
回到顶部

Android中调用服务端的接口一般返回的是json数据，在解析json的时候，无论是使用原始的手工解析方式，还是使用javabean的解析方式，解析出来的结果在使用的时候必须都进行判空处理。不允许因为服务端的json出问题，导致app在解析json的时候出现崩溃。
回到顶部

所有类的成员变量一定要赋初始值，不允许只定义，不赋值。
回到顶部

函数返回的时候，如果返回的int类型的数据并不是真实的实用的数据值(例如表示高度，宽度，大小等值)，仅仅表示函数执行成功、失败、异常的状态值，并且这些值是有限的几个值，必须要将这些值使用静态常量描述，或者使用枚举，例如：
int GetJsonString()
该函数返回-1表示获取解析json数据异常，返回0表示成功，返回1表示网络连接异常，返回2表示json内容中的数据部分为空。那么在函数内部的代码里不要直接使用这些字面值，这些字面值对于程序员来说是毫无意义的，代码可阅读性很差，建议做成下面的模式：
public static final int RESULT_PARSE_JSON_EXCEPTION = -1;

public static final int RESULT_SUCCESS = 0;
public static final int RESULT_NETWORK_EXCEPTION = 1;
public static final int RESULT_NO_DATA = 2;
使用这些符号常量值代替字面值的好处是，符号常量值是由大写的英文单词组成，是有意义的，可以帮助程序员更好的理解函数返回值的意义，而且符号常量值对应的具体的赋值在后期是很方便修改的。
回到顶部

如果一个Activity可能在多个地方被打开，或者一个Fragment可能在多个地方被用到。那么在设计该Activity和Fragment的时候一定要考虑低耦合，对外提供统一的参数接口，启动Activity的过程封装在该Activity类的静态成员方法里面，类似如下：

```
class MyActivity extends Activity{
...
public static void startActivity(Context context,Params param){
Intent intent = new Intent(context, MyActivity.class);

intent.putExtra("param", param);
startActivity(intent);
}
public static void startActivityForResult(Context context,Params param){
Intent intent = new Intent(context, MyActivity.class);

intent.putExtra("param", param);
startActivityForResult(intent,REQUEST_CODE);
```

```
}
}
```

参数的传递最好是封装在一个Model实体类中，避免使用Map这种方式进行参数传递。建议该实体类实现为对应的Activity的静态可序列化的内部类。

回到顶部

AndroidStudio中的项目的包结构应该根据工程各个部分的功能来组织。

回到顶部

每一个Activity里面几乎都会定义一个Handler内部类，但是很多Activity里面的Handler都使用了重复的消息类型，这里面是有冗余代码的，所以应该把这些Activity都使用到的Handler类的消息部分，提取成一个公用的Handler类。然后在各个Activity里面使用继承的方式，来提供该Activity特有的Handler消息类型的Handler类实现。

另外Handler发送消息应该使用Handler类的成员函数，不应该直接使用`handler.obtainMessage(xxx).sendToTarget();`这种原始的发送消息的方式，这样不利于降低耦合，这种细节应该隐藏在Handler类的里面。Handler的消息类型应该定义为Handler类里面的静态常量，而该常量不应是public的，对外部不可见。也就是说使用handler对象发送消息的细节不应该暴露给外部。

回到顶部

封装ListView的数据更新，在handlerMessage中更新数据，避免出现 java.lang.IllegalStateException 问题

回到顶部

Activity与Fragment的数据传递采用interface的方式，这样可以降低耦合，有利于Fragment的复用：

113

Try using interfaces.

Any fragment that should pass data back to its containing activity should declare an interface to handle and pass the data. Then make sure your containing activity implements those interfaces. For example:

In your fragment, declare the interface...

```
public interface OnDataPass {
    public void onDataPass(String data);
}
```

Then, connect the containing class' implementation of the interface to the fragment in the onAttach method, like so:

```
OnDataPass dataPasser;

@Override
public void onAttach(Activity a) {
    super.onAttach(a);
    dataPasser = (OnDataPass) a;
}
```

Within your fragment, when you need to handle the passing of data, just call it on the dataPasser object:

```
public void passData(String data) {
    dataPasser.onDataPass(data);
}
```

Finally, in your containing activity **which implements** OnDataPass...

```
@Override
public void onDataPass(String data) {
    Log.d("LOG","hello " + data);
}
```

回到顶部

一般在Activity中我们通过网络请求服务端的接口获得数据，这个过程一般是在一个线程中做的，获取到数据之后，再通过Activity中的handler发送消息来通知Activity更新数据。该负责获取数据的线程类，我们一般都实现为一个Activity的内部类，该类可以直接访问Activity的成员变量，例如handler，数据列表对象等。但是这样不利于该数据获取线程的复用。如果另一个Activity里面也需要获取相同的数据，那么这个功能是不能复用的，所以这个负责数据请求的线程类，不应该与具体的Handler和Activity联系过于紧密。应该定义为一个静态类，handler应该作为参数传递进来，而不是直接访问外部类的成员变量。

回到顶部

Log功能应该封装成为自动将当前所在类的类名变成log输出的TAG参数，发布的app最好是能循环写日志文件到系统存储中，并且日志文件应该使用反复覆盖的方式重复利用。下面仅仅是一个不完善的例子：

```
1
2
3
4
5
6
7
public class MyLog {
public static final String TAG = "myapp ";
public static void v(Object o,String message)
```

```
{
Log.v(TAG+o.getClass().getSimpleName(),message);
}
}
```

使用

```
MyLog.v(this,"hello log");
```

打印结果

```
V/myapp MainActivity: hello log
```

回到顶部

使用自动化版本管理，自动生成版本号，使应用程序的版本与版本库上保持一致。使用hg替换工程目录下的app目录下的build.gradle文件即可，如果manifest里面也有版本号的设置，AndroidStudio还是以build.gradle为准。不应该在每次发布的时候，在AndroidStudio的工程设置里面手工修改版本号。

回到顶部

应该为app添加全局异常捕获，app中总会有一些我们未捕获的异常，一旦用户使用过程中遇到这样的异常，程序就会崩溃，我们应该检测该类未捕获的异常信息，程序崩溃的时候通过写文件日志，或者发送邮件的方式获得异常信息，以便解决bug。