# Android笔记(下篇)

[Android笔记(下篇) 1](#_Toc463546160)

[1 优化 2](#_Toc463546161)

[1.1. 代码优化 2](#_Toc463546162)

[1.1.1 基本思路 2](#_Toc463546163)

[1.1.2 声明和定义 2](#_Toc463546164)

[1.1.3 代码的使用 3](#_Toc463546165)

[1.1.4 封装 4](#_Toc463546166)

[1.1.5 抽取 4](#_Toc463546167)

[1.2. 布局优化 4](#_Toc463546168)

[1.2.1 Merge标签 4](#_Toc463546169)

[1.2.2 ViewStub 4](#_Toc463546170)

[1.3. 绘图优化 4](#_Toc463546171)

[2 开源框架及第三方SDK 5](#_Toc463546172)

[2.1. 开源框架 5](#_Toc463546173)

[2.1.1 使用时的常见问题 5](#_Toc463546174)

[2.1.2 MPAndroidChart 5](#_Toc463546175)

[2.1.3 Retrofit、volley、ion 5](#_Toc463546176)

[2.1.4 Glide、Picasso、Fresco 5](#_Toc463546177)

[2.1.5 ImageLoader 5](#_Toc463546178)

[2.1.6 SlidingMenu 5](#_Toc463546179)

[2.1.7 NineOldAnimator 5](#_Toc463546180)

[2.1.8 PhotoView 5](#_Toc463546181)

[2.1.9 Xutils 6](#_Toc463546182)

[2.1.10 StellarMap 6](#_Toc463546183)

[2.1.11 ObservableScrollView 6](#_Toc463546184)

[2.1.12 BGARefreshLayout 6](#_Toc463546185)

[2.1.13 QuickBaseAdapter 6](#_Toc463546186)

[2.2. 第三方SDK 6](#_Toc463546187)

[3 业务相关 7](#_Toc463546188)

[4 调试 8](#_Toc463546189)

[4.1. 友盟 8](#_Toc463546190)

[4.2. Gradle 8](#_Toc463546191)

[4.3. Studio项目导入的相关问题 8](#_Toc463546192)

[4.4. ADB Shell Commands 8](#_Toc463546193)

[4.5. 开发工具bug 8](#_Toc463546194)

[4.6. Eclipse和Studio的差别 8](#_Toc463546195)

[5 冷门内容 10](#_Toc463546196)

[5.1. JNI使用流程 10](#_Toc463546197)

[6 杂项 12](#_Toc463546198)

[7 代码 13](#_Toc463546199)

[7.1. 方法 13](#_Toc463546200)

[7.2. 效果 13](#_Toc463546201)

[7.2.1 ActionBar的overflow 13](#_Toc463546202)

1. 优化
   1. 代码优化
      1. 基本思路
2. 与用户交互的时候，动态实时启动的代码块里，尽量少放代码，因为用户启动时才会执行其中的代码，如果代码执行时间较长，会产生卡顿的感觉。可以提前加载的内容可以在Activity的onCreate()方法中写代码，这样可以减少动态实时启动时的启动时间。但是这样也有个问题，就是将本来的局部变量会被提升为成员变量，对象在内存中存在的时间就会变长。
3. ListView数据的分批加载（实际开发是采用第二种方式）

分批加载可以通过两种方式实现：

1. ListView自身每次只加载数据集合的一部分内容。
2. 服务器将不同批次的数据放到不同的url地址中，这样代码中只要迭代url地址就可以实现分批加载数据。
3. 使用成员变量还是参数将数据传给方法?

ViewPager的setCurrentItem()方法代码如下：

mPopulatePending，mFirstLayout两个状态是整个ViewPager的，所以通过成员变量直接给到方法中。item这种需要ViewPager类外部代码提供的，以参数的形式给到方法。感觉涉及到ViewPager整体运行情况的变量，都被设置成了成员变量，并且直接提供给方法使用。

需要调用类内部成员变量的方法可以像下面的代码，提供给外部一个需要提供参数的方法，然后该方法内部调用类内部的方法，类内部的方法调用成员变量。

|  |
| --- |
| public void setCurrentItem(int item) {  mPopulatePending = false;  setCurrentItemInternal(item, !mFirstLayout, false);  } |

1. 抽取父类的时候，成员变量可以直接被子类使用，但是具体是为子类直接提供成员变量还是提供获取成员变量的方法，需要继续学习。但是抽取父类的时候，应该充分考虑子类将会用到的变量，并在父类中实现原始变量获取，然后再提供给子类。父类中获取原始变量的方式应该是可重用的。
2. 子线程中使用的变量，如果是从子线程外获取的，那么需要注意这个变量的生命周期是否可以坚持到子线程结束。但是这个变量即使用了，只是会造成内存泄漏，不会在子线程运行过程中变成空指针。提供变量的对象还是可以关闭。
3. 运行时异常也应该try，以免APP正常运行时崩掉。
   * 1. 声明和定义
4. viewHolder定义类的时候，数据量大不要写静态，可能出问题。数据量小可以用静态。
5. 常量的声明：

下面这段代码用到了Android对Layout\_width参数常量的定义，之所以 MATCH\_PARENT和 WRAP\_CONTENT声明为负值，看第一行代码，如果这两个状态定义为负值，用户填写精确值的时候就可以直接进行判断，不会被MATCH\_PARENT和 WRAP\_CONTENT两个值干扰。

|  |
| --- |
| if (childDimension >= 0) {  resultSize = childDimension;  resultMode = MeasureSpec.EXACTLY;  } else if (childDimension == LayoutParams.MATCH\_PARENT) {  // Child wants to be our size. So be it.  resultSize = size;  resultMode = MeasureSpec.EXACTLY;  } else if (childDimension == LayoutParams.WRAP\_CONTENT) {  // Child wants to determine its own size. It can't be  // bigger than us.  resultSize = size;  resultMode = MeasureSpec.AT\_MOST; |

* + 1. 代码的使用

1. Inflate的调用：

如果不需要拿到LayoutInflater对象，直接使用View.inflate()。因为直接调用SystemService服务获取LayoutInflater对象并不保证百分之百能够得到。LayoutInflater.from()方法对不能得到LayoutInflater对象的情况做了处理。所以最不安全的做法就是调用SystemService来打气一个view。

如果需要拿到LayoutInflater对象，使用LayoutInflater.from()方法。

三个类的实现见下面：

1. View类

|  |
| --- |
| public static View inflate(Context context, int resource, ViewGroup root) {  LayoutInflater factory = LayoutInflater.from(context);  return factory.inflate(resource, root);  } |

1. LayoutInflater

|  |
| --- |
| public static LayoutInflater from(Context context) {  LayoutInflater LayoutInflater =  (LayoutInflater) context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);  if (LayoutInflater == null) { //保证拿不到的时候报错，而不会造成后面的代码出  现问题。  throw new AssertionError("LayoutInflater not found.");  }  return LayoutInflater;  } |

1. SystemService

|  |
| --- |
| context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE); |

1. Action事件的判断

|  |
| --- |
| 旧的：  final int action = ev.getAction ();  … …  switch (action & MotionEvent.ACTION\_MASK) { //action的数值可能因为干扰使某些位上产生了  数值，这里要过滤一下再使用。  case MotionEvent.ACTION\_DOWN:  新的：  final int actionMasked = ev.getActionMasked(); //直接调用新的Api，就已经过滤完了 |

* + 1. 封装
    2. 抽取
  1. 布局优化
     1. Merge标签

1. Merge标签什么属性都没有，因为布局文件必须有根标签，所以如果用任意一个ViewGroup作为根标签，均会增加View的层级结构。如果上下相邻的两级ViewGroup是相同的类型，那么就可以用Merge标签代替下一级的ViewGroup标签，这样就避免了出现两级ViewGroup同类型重复的问题。
2. 系统只将Merge标签看做一个占位符。
3. Merge标签下的控件可以按照将要添加到的ViewGroup的布局参数进行设置。
   * 1. ViewStub
   1. 绘图优化
   2. OOM
      1. MAT使用
4. Android好像在系统发生OOM的时候不会自动保存堆内存信息，
5. 当OOM发生时，系统会记录下OOM时堆内存当时的状态，然后可以通过MAT分析这时的堆内存状态，来判断为什么产生内存溢出。多数情况下是内存泄露造成的，所以这时如果哪个对象占了很大的内存，那么这个对象就极有可能发生了内存泄露。
6. 通常我们都会采用下面的“三步曲”来分析内存泄露问题：
7. 首先，对问题发生时刻的系统内存状态获取一个整体印象。
8. 第二步，找到最有可能导致内存泄露的元凶，通常也就是消耗内存最多的对象
9. 接下来，进一步去查看这个内存消耗大户的具体情况，看看是否有什么异常的行为。
10. 内存空间中垃圾回收的工作由垃圾回收器 (Garbage Collector,GC) 完成的，它的核心思想是：对虚拟机可用内存空间，即堆空间中的对象进行识别，如果对象正在被引用，那么称其为存活对象，反之，如果对象不再被引用，则为垃圾对象，可以回收其占据的空间，用于再分配。

在垃圾回收机制中有一组元素被称为根元素集合，它们是一组被虚拟机直接引用的对象，比如，正在运行的线程对象，系统调用栈里面的对象以及被 system class loader 所加载的那些对象。堆空间中的每个对象都是由一个根元素为起点被层层调用的。因此，一个对象还被某一个存活的根元素所引用，就会被认为是存活对象，不能被回收，进行内存释放。因此，我们可以通过分析一个对象到根元素的引用路径来分析为什么该对象不能被顺利回收。如果说一个对象已经不被任何程序逻辑所需要但是还存在被根元素引用的情况，我们可以说这里存在内存泄露。

1. 开源框架及第三方SDK
   1. 开源框架
      1. 使用时的常见问题
2. 库文件要跟项目在同一个目录下，跨文件夹关联可能会失败。
3. 库文件中的v4包等文件，如果和自己项目的文件冲突，优先保留库文件的。
   * 1. MPAndroidChart
   1. 图表
      1. Retrofit、volley、ion
   2. 网络架构
      1. Glide、Picasso、Fresco
   3. 图片加载：包括gif动画；大图片预取样，加快显示速度，然后加载整张图片
   4. 视频
      1. ImageLoader

1.图片如果需要四个角显示圆弧，图片控件的宽高要是精确的，不然图片无法正常显示。

* + 1. SlidingMenu

1. 侧滑菜单
2. 单例设计模式
   * 1. NineOldAnimator
3. 使1.0版本~3.0版本系统实现属性动画置
   * 1. PhotoView
4. 继承于ImageView，用于展示图片，并且在展示的同时可以实现缩放的效果。
5. 在ViewPager中使用PhotoView时，注意将ViewPager替换为HackyViewPager，因为手势缩放会使ViewPager的onTouchEvent方法报错，HackyViewPager隐藏了错误，使程序不会崩掉。
   * 1. Xutils
6. 6.0以后由于Android取消了对httpClient的支持，所以不好用了，但能用。
7. Volly，okHttp是5.0, 6.0自带的
8. noHttp修改okHttp的产物
9. 能够实现断点续传功能
   * 1. StellarMap
10. 屏幕會填充很多控件，像宇宙中的星系一樣，飛過來的感覺。但是控件大小等自身的屬性，需要在Adapter中自己進行設置。
    * 1. ObservableScrollView
11. 根据ScrollView的滚动实现一些其他的随动效果。
    * 1. BGARefreshLayout
12. 上拉加载下拉刷新的layout。
    * 1. QuickBaseAdapter

各种BaseAdapter、其他Adapter的重写，简化了重写adapter的代码量。

* 1. 第三方SDK

1. 社会化分享：mob的shareSDK
2. 数据统计：友盟
3. 数据推送：极光推送（xmpp）
4. 短信验证：shareSDK（免费），极光推送
5. APP Splash页面广告推送：极光推送
6. 业务相关
7. SplashActivity页面进行软件版本更新的时候，第一次联网查询的数据应该包含：
   1. 版本号
   2. 版本名称
   3. 新版本下载地址
8. 调试
   1. 友盟
9. 混淆要注意，因为友盟内部采用了反射，如果被混淆了，那么反射就会在调用时报错“no such method”。
   1. Gradle
10. 创建APK的时候如果需要打包的时候指定删除或操作resource的内容，可以在build.gradle中的defaultConfig中使用resConfig()
11. 打包压缩
12. 没用到的类（第三方）
13. 没用到的资源Resource
14. 国际化的文件
15. 兼容包的资源：一般AS打包之后，删除资源文件，然后用签名命令单独打包（不用Studio）
    1. Studio项目导入的相关问题
16. 在导入项目的时候，如果build tools，gradle version这些参数和当前的Studio不同的话，studio会自动进行提示进行下载。如果这时选择不下载，自己修改参数，有可能会在根build.gradle文件中加入Android{}代码，然后在sync now的时候会报错，”can not find method android() for argument... ”。
17. 这个错误其实是module的build.gradle还存在错误，gradle认为project中没有module，所以才会将project当成module。
18. 这个问题只要将各个module的build.gradle文件配置正确之后，就会得到解决。
19. 工具窗口Message，在导入项目的时候Gradle Build Android()
    1. ADB Shell Commands
    2. 开发工具bug
20. Layout.xml文件的根元素设置id之后，findViewById(R.id.\*\*)时，\*\*处就显示不出来根元素下面子组件的id。重启eclipse，或者重新修改layout.xml文件再保存。
    1. Eclipse和Studio的差别

|  |  |  |
| --- | --- | --- |
| 问题 | Eclipse | Studio |
| 屏幕适配 | 删除values-\*\*文件夹 | 不能用 |
| 导jar包 | buildPath手动进行，自动经常出问题 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. 冷门内容
   1. JNI使用流程
2. JNI实际开发时候的任务分工
   1. Java工程师的工作范围：
      1. Java代码中的本地方法定义
      2. 在bear.c或bear.cpp文件中对java中定义的本地方法进行实现（假设方法名叫method，下面用到）。
   2. C工程师的工作范围：
      1. 在bear.c或bear.cpp文件中，编写底层使用的c或c++函数，供java工程师在自己编写的method方法中调用。
   3. bear.c或bear.cpp文件中，既有C工程师的代码，也有java工程师的代码。
3. 实现JNI流程（c和c++）
   1. 两种方式：Android项目修改为JNI项目、自己创建所有相关内容
   2. 创建Android项目
   3. 在java代码中定义本地方法。
   4. 在根目录下创建jni目录。在其中创建.c或.cpp文件，文件中实现java中定义的native函数；创建Android.mk和Application.mk文件。
   5. 在.c文件的目录调用命令行，执行ndk-build命令生成.c或.cpp的.so类库文件。
   6. Java代码中导入类库，即可使用本地方法。
   7. c和c++实现的细微差别：
      1. c++的env变量和c的env变量类型是不同的，前者是一级指针，后者是二级指针，所以使用env的时候，c++：env->\*\*；c: (\*env)->\*\*
      2. c++实现方法前，要先声明方法，引入javah命令获得的.h文件即可
      3. c++中env->\*\*调用方法的时候，不需要传入env这个参数
4. 注意事项：
5. C中的方法名称：Java\_包名\_类名\_方法名。方法名可以调用jdk的javah命令生成
6. Application.mk文件决定生成的类库文件所属的平台类型。
7. Android.mk用来指定类库的名称和需要编译的.c或.cpp文件名
8. 常见错误
9. 找不到类库：findLibrary return null
   1. 没有添加对x86的支持
   2. 加载类库时写错库名
10. 找不到本地方法：native method not found
    1. 忘记加载类库
    2. C函数名写错了
11. Javah命令
    1. 用来获取java本地方法的C函数名
    2. 1.7及以上：在src目录下执行 javah com.itheima.hello2.MainActivity
    3. 1.7以下：在bin/classes下执行javah com.itheima.hello2.MainActivity
12. JNI项目添加本地支持
    1. 自动创建jni文件夹，文件夹下自动生成Android.mk文件和cpp文件
    2. 需要手动指定jni.h头文件的路径
    3. c文件中可以实时编译
    4. 项目部署时，自动编译打包c文件生成动态库
13. C调用java
    1. 反射时，指定要获取的字节码，不能使用包名，要写路径"com/itheima/calljava/MainActivity"
14. Javap命令
    1. 用于获取methodId
    2. 在bin/classes目录下执行javap -s com.itheima.calljava.MainActivity
    3. 执行Javah命令生成的文件中也有methodId
15. 杂项
16. 在安卓系统下，即使两个应用的包名均是com.\*\*.\*\*，也并不是说两个包的com文件夹就在同一个上级文件夹中。所以A应用即使知道B应用中某个类的包名，也不一定能够直接通过包名调用到这个类。
17. 包只是一个路径，文件夹的层级结构。
18. 一个类文件，放在不同的包中，即使代码实现完全相同，对于系统来说也不是同一个类文件。比如A文件中调用com.me.a这个包中的demo类文件，A只认识com.me.a.demo，如果另一个包com.me.b中也有一个demo类文件，A是使用不了的。因为有这种可能，demo在两个包中均需要定义，但是代码是不同的，如果系统检查类文件时不核对包名，就无法区分连个不同的demo文件。系统是否只核对到包名层级，不再核对应用名称层级？
19. OutOfMemory: OOM
20. 每个应用的VM Heap大小由手机的屏幕大小决定。
21. 调用sqlite数据库也是要关闭的。
22. 声明变量会占用栈的内存空间。
23. 堆内存在系统中只有一个，不同进程共用一个堆内存，不同进程中各个对象的地址在堆内存中互相交叉，没有说哪一块堆内存专门给某一个进程使用。各个进程各自的栈内存地址是连续的，预先分配好的。
24. 6.0动态权限申请的目的：

运行时权限提供给用户关于应用所需权限更多的相关上下文和可视性，这也让开发者帮助用户更好的理解：为什么应用需要所请求的权限，授权将有什么样的好处，拒绝将有何种不便。用户可以通过设置中的菜单来撤销应用的权限。

这要申请了，就长期有效。

1. 代码
   1. 方法
2. 切换Activity的进入和退出的动画：overridePendingTransition();
   1. 效果
      1. ActionBar的overflow
3. 见开源中国第二天上午代码