本文介绍Android开发过程中的一些基本常识，大多是一些流程、专业术语和解决问题的方法等。

软件开发流程

一个完整的软件开发流程离不开策划、交互、视觉、软件、测试、维护和运营这七个环节，这七个环节并不是孤立的，它们是开发一款成功产品的前提，但每一项也都可以形成一个学科，是一个独立的岗位，随着敏捷开发的流行，以及来到了体验为王的时代，现代软件开发更多的是注重效率和敏捷，而不是循规蹈矩的遵循这些开发流程，比如软件开发的岗位不再仅仅是个技术岗位，它需要去参与前期的设计和评审、可以在视觉和交互方面提出自己的见解，在开发的过程中需要自测程序尽快解决现存问题，运营和维护的过程中也需要软件的帮助。可见现代软件开发对开发者的综合素质（这并不是facebook所讲的全栈工程师）越来越高，自称为码农或者程序猿显然是不合理的，因为这个过程是脑力劳动和体力脑动并存，称呼自己为工程师显得更为合理。

- **策划：需求收集（通过用户调研、灰度发布、大数据分析、竞品分析、领导拍脑袋等方式获取需求）、需求整理（将需求归类、划分优先级等）、将需求转换成解决方案（输出设计文档）；**
- **交互：从心理学（利用人性的弱点）、人性化（心智）、个性化的角度将解决方案转换成可交互的功能和界面（需要输出交互文档），比如加载等待、消息提示、页面布局、页面内和页面间的交互逻辑、页面切换动画等等，这个过程中一般会使用Axure或者PowerPoint来制作交互文档；**
- **视觉：根据交互图，使用PhotoShop来做视觉效果，在Android上的图片格式大多是png和jpg，对于需要屏幕适配，程序又适合做屏幕适配的地方可以使用九图，格式为*.9.png。**
- **软件：根据视觉和交互效果将需求转化为具体的实现，在实现的过程中可能会因为需求、交互或者视觉的变动导致软件实现的变动，因为策划、交互、视觉这每一个环节都可能会有信息失真的现象，或者是由于市场环境的变化、获取信息不够准确、领导拍脑袋等等情况导致软件始终处于被动状态，所以现在会提倡敏捷开发、结对编程、程序设计、同行评审、单元测试来提高程序的灵活性和稳定性；**
- **测试：软件达到可交互的标准后，需要将可交互的程序提供测试，其中灰度发布（用户测试）、自测（开发自测）、SQA（品质保证）都算是测试；**
- **维护和运营：通过测试程序达到稳定标准后，软件就可以上线了，软件上线后，需要去维护，用户反馈的问题要及时解决、用户有疑问要及时解答；根据后台统计信息、抓住可运营的节日、民族文化需要做运营来提高用户使用产品的粘度，让更多的用户知道、使用产品都是运营应该做的。**

注：

- 可以查看这个答案了解一个APP从创意到上线的具体流程，开发一个APP有多难？
- 可以查看笔戈科技的这篇文章了解一个手机（平板或其它电子产品也差不多）的诞生需要哪些环节，一个手机的诞生过程

提问的智慧

大多数工作都是以结果为导向的，特别是软件开发这个职业，绩效考核、KPI这些都是在考核你工作的成果，所以工作更多地是需要你解决问题的能力，至于学习这个事情，还是在工作之外的时间去做吧。对于提高解决问题能力我有两个建议：

- **学会学习和思考：学习的过程中要广度和深度并存，Android应用开发本身对技术功底的要求不高（因为很多底层的东西都被google、框架、开源代码给封装起来了，多数时候你只需要看ReadMe或者API知道怎么用就可以了），更多地是在你遇到问题的时候知道这个问题能够通过什么方法和方式来解决。书要看，但多逛逛论坛、QQ群、Github、StackOverflow、CSDN博客专栏对自己都是有益的。**
- **学会提问：你身边有很多资源，比如同事、StackOverflow、QQ技术交流群、搜索引擎，当你遇到问题的时候完全可以利用身边的资源来解决遇到的问题，如果一个问题在一个小时之内自己都不能够解决它，我就会通过搜索引擎、Github、QQ技术交流群、同事、StackOverflow（以上排序是按优先级排列的）来解决它。如果你需要好的答案你就需要有好的提问，特别是在QQ群或者论坛，在提问的过程中需要体现出你的思考，能够通过搜索引擎解决的问题坚决不问他人，这是对别人的尊重，在这里推荐几个链接，认真看会对你有莫大的帮助：**

如何用好 Google 等搜索引擎？
程序员应该如何提问？
提问的智慧
Smart Questions

解决bug的方法

为了写这一项我专门在知乎上提过一个问题：
你有哪些解决bug的技巧？

在知道如何快速解决bug之前，你需要知道什么是bug。没有完成策划、交互、视觉要求的功能，这不叫bug，这叫功能缺陷；一个功能完成后不能正常使用也不叫bug，因为它根本还没达到可测试的标准。我认为当你的程序达到可测试标准之后发现的问题才叫bug。综合我自己解决bug的经验和知乎上的回答，总结常见的解决bug的方法有（你想要高效解决bug的前提是你能够快速定位到缺陷所在的位置，所以以下方法多数讲的是如何快速定位问题，至于真正解决bug，需要你自己修改程序才行）：

- **断点调试：**

以Eclipse为例：

1、打断点：

（1）打断点：

打断点

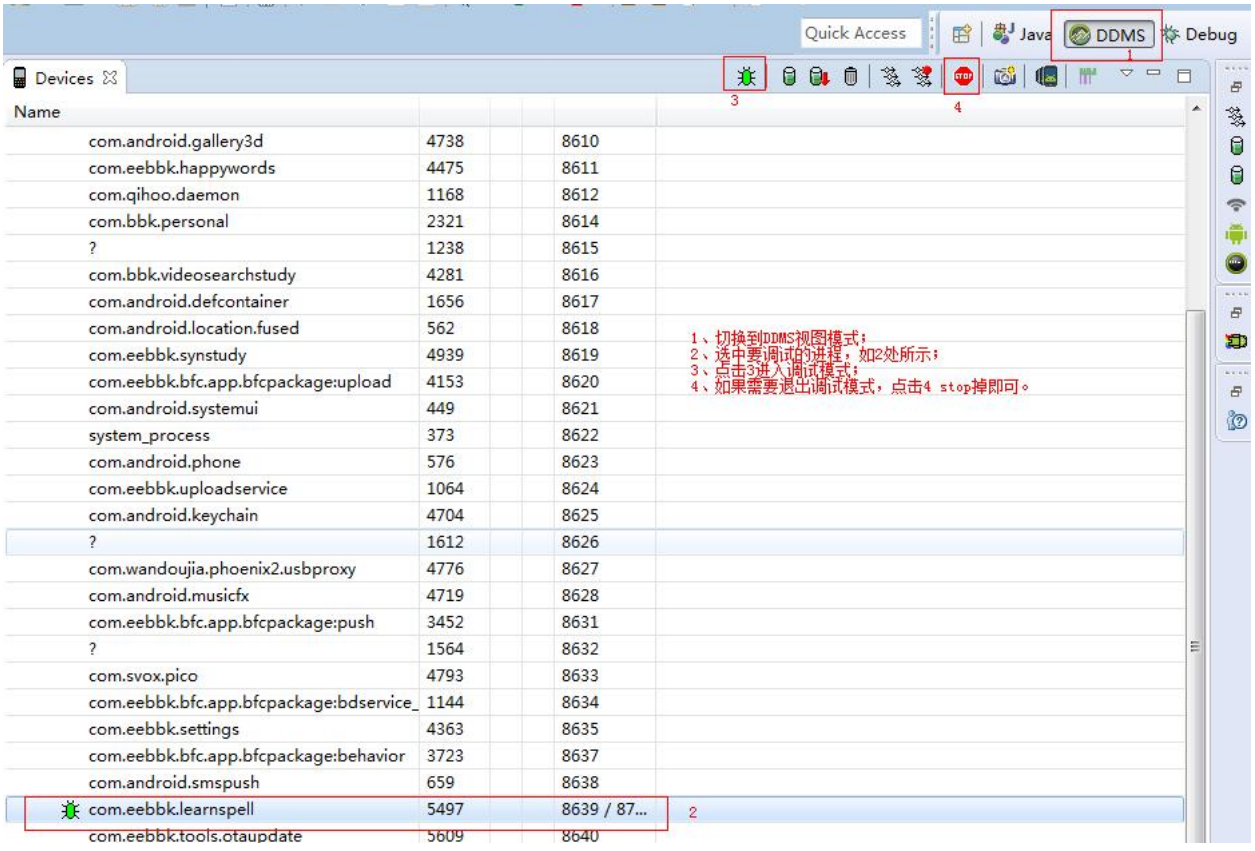
（2）清除断点：

清除断点

2、启动调试模式的两种方式：

（1）通过debug as启动调试程序：右键工程名-->Debug AS -->Android Application -->模拟器或者真机会弹出.....watching for the debugger.....的提示框，不要点击等待其自动消失 --> 此时已经进入调试模式，操作程序到达打断点的地方。

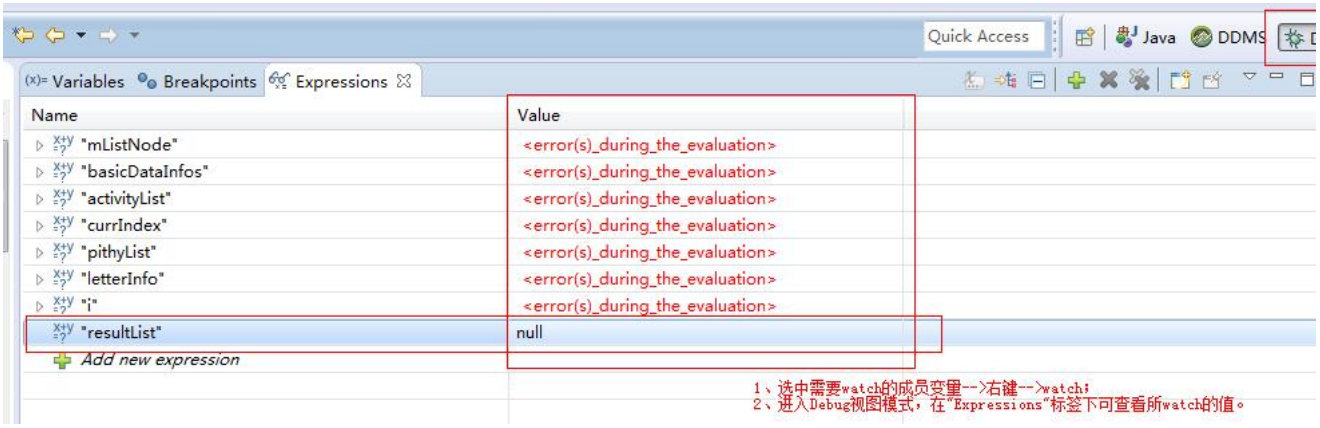
（2）在程序运行过程中，在DDMS视图下选中要调试的程序，启动调试模式：



DDMS视图进入调试模式

3、调试：请自行尝试F5、F6、F7、F8这几个调试的快捷键；

4、watch成员变量：在调试的过程中，比如在执行for、while、do while循环、递归、系统回调等程序时可以通过watch来观察成员变量或者方法返回值的变化情况，watch的方法：



watch

注：更多关于在Eclipse IDE中调试Android程序的知识请参见：Android eclipse中程序调试

- **打印：**

打印调试的方法对于循环、异步加载、递归、JNI等代码段非常有用，特别是在循环中，在循环次数非常大时，通过打断点调试显然是一件费力的事情，这时候打印就显得更“智能”了，我通常会通过下面封装的打印调试类来输出打印信息，这个类可以打印print、log、行号、文件名、StrictMode等信息，当不需要打印信息时，只需要将DEBUG_MODE改为false就可以了：

```
import android.content.Context;  import android.os.StrictMode;  import android.util.Log;  import android.widget.Toast;  /** * 调试打印类 * */publicclassDebugUtils{  privateDebugUtils(){  }  publicstaticvoidprintln(String printInfo){  if(Debug.DEBUG_MODE && null != printInfo){  System.out.println(printInfo);  }  }  publicstaticvoidprint(String printInfo){  if(Debug.DEBUG_MODE && null != printInfo){  System.out.print(printInfo);  }  }  publicstaticvoidprintLogI(String logInfo){  printLogI(TAG, logInfo);  }  publicstaticvoidprintLogI(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag && null != logInfo){  Log.i(tag, logInfo);  }  }  publicstaticvoidprintLogE(String logInfo){  printLogE(TAG, logInfo);  }  publicstaticvoidprintLogE(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag && null != logInfo){  Log.e(tag, logInfo);  }  }  publicstaticvoidprintLogW(String logInfo){  printLogW(TAG, logInfo);  }  publicstaticvoidprintLogW(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag && null != logInfo){  Log.w(tag, logInfo);  }  }  publicstaticvoidprintLogD(String logInfo){  printLogD(TAG, logInfo);  }  publicstaticvoidprintLogD(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag && null != logInfo){  Log.d(tag, logInfo);  }  }  publicstaticvoidprintLogV(String logInfo){  printLogV(TAG, logInfo);  }  publicstaticvoidprintLogV(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag || null != logInfo){  Log.v(tag, logInfo);  }  }  publicstaticvoidprintLogWtf(String logInfo){  printLogWtf(TAG, logInfo);  }  publicstaticvoidprintLogWtf(String tag, String logInfo){  if(Debug.DEBUG_MODE && null != tag && null != logInfo){  Log.wtf(tag, logInfo);  }  }  publicstaticvoidshowToast(Context context, String toastInfo){  if(null != context && null != toastInfo){  Toast.makeText(context, toastInfo, Toast.LENGTH_LONG).show();  }  }  publicstaticvoidshowToast(Context context, String toastInfo, int timeLen){  if(null != context && null != toastInfo && (timeLen > 0)){  Toast.makeText(context, toastInfo, timeLen).show();  }  }  publicstaticvoidprintBaseInfo(){  if(Debug.DEBUG_MODE){  StringBuffer strBuffer = new StringBuffer();  StackTraceElement[] stackTrace = new Throwable().getStackTrace();  strBuffer.append(";  class:").append(stackTrace[1].getClassName())  .append(";  method:").append(stackTrace[1].getMethodName())  .append(";  number:").append(stackTrace[1].getLineNumber())  .append(";  fileName:").append(stackTrace[1].getFileName());  println(strBuffer.toString());  }  }  publicstaticvoidprintFileNameAndLinerNumber(){  if(Debug.DEBUG_MODE){  StringBuffer strBuffer = new StringBuffer();  StackTraceElement[] stackTrace = new Throwable().getStackTrace();  strBuffer.append(";  fileName:").append(stackTrace[1].getFileName())  .append(";  number:").append(stackTrace[1].getLineNumber());  println(strBuffer.toString());  }  }  publicstaticintprintLineNumber(){  if(Debug.DEBUG_MODE){  StringBuffer strBuffer = new StringBuffer();  StackTraceElement[] stackTrace = new Throwable().getStackTrace();  strBuffer.append(";  number:").append(stackTrace[1].getLineNumber());  println(strBuffer.toString());  return stackTrace[1].getLineNumber();  }else{  return0;  }  }  publicstaticvoidprintMethod(){  if(Debug.DEBUG_MODE){  StringBuffer strBuffer = new StringBuffer();  StackTraceElement[] stackTrace = new Throwable().getStackTrace();  strBuffer.append(";  number:").append(stackTrace[1].getMethodName());  println(strBuffer.toString());  }  }  publicstaticvoidprintFileNameAndLinerNumber(String printInfo){  if(null == printInfo || !Debug.DEBUG_MODE){  return;  }  StringBuffer strBuffer = new StringBuffer();  StackTraceElement[] stackTrace = new Throwable().getStackTrace();  strBuffer.append(";  fileName:").append(stackTrace[1].getFileName())  .append(";  number:").append(stackTrace[1].getLineNumber()).append("\n")  .append((null != printInfo) ? printInfo : "");  println(strBuffer.toString());  }  publicstaticvoidshowStrictMode(){  if(DebugUtils.Debug.DEBUG_MODE){  StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()  .detectDiskReads().detectDiskWrites().detectNetwork().penaltyLog().build());  StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()  .detectLeakedSqlLiteObjects().detectLeakedClosableObjects().penaltyLog().penaltyDeath().build());  }  }  publicstaticvoidd(String tag, String msg){  if(DebugUtils.Debug.DEBUG_MODE){  Log.d(tag, msg);  }  }  publicclassDebug{  publicstaticfinalboolean DEBUG_MODE = true;  }  publicstaticfinal String TAG = "Debug";  }
```

- **目视法：**

这适合于code review，但是不太靠谱，因为人的精力毕竟有限，有时候你多敲一个分号，缩进不对都有可能导致程序出现问题，但在代码量较少时是一个高效率的方法。

- **自动化测试：**

Android的自动化测试（分白盒测试和黑盒测试）工具有：monkey、Robotium、Appium、云端测试（比如testin）,具体用法可参见：

android实用测试方法之Monkey与MonkeyRunner

Robotium

Testin

Appium中文教程

- **排除法：**

调试、打印、目视这三种方法适合于可以复现的问题，对于随机问题（实际上不存在随机问题，只是问题不那么容易复现而已），比如在线程、音频播放、AnsynTask、Timer切换或者结束时刚好做了相应地人为操作导致出现灵异现象。这时候可以通过排除法来排查问题，具体的方法是首先大概定位到出现问题的位置，然后将代码一段一段地注释，观察程序现象，逐步缩小出现问题的范围。

版本管理介绍

在较大的软件开发过程中，可能有多个软件工程师同时开发一个项目的情况，比如有负责读取数据、获取网络数据等API封装的，有负责程序架构的，有负责上层界面实现的，为了能够最终编译一个完成的程序出来，需要将代码整合，这个时候最方便的方法就是使用版本管理工具，固定时间上传（比如每天、没改动一个功能等等），这样能够实时保证服务器上的代码是最完整、最新的，也可以避免由于自然灾害、电脑异常导致本地电脑挂掉损失掉代码的问题。

常见的版本管理工具有SVN和Git，我也使用过CVS，关于版本管理工具的介绍参见：

版本控制

版本控制系统的选择之路

git教程

git简易指南

注：对于windows用户来说，建议使用乌龟壳系列的版本控制客户端，使用github的朋友可以使用github for windows客户端：

tortoisegit

tortoisecvs

tortoisesvn

github for windows

编译

通常我们用Eclipse或者Android Studio开发android程序时，只需要运行程序就可以在模拟器或者机器上运行程序了，但为了保证代码的完整性、能够在服务器上编译，需要通过编译工具将代码编译成apk，常见的编译工具有：ant、gradle，但这两种编译工具都是需要通过手动敲命令来完成编译功能（当然你也可以自己写脚本来实现编译自动化），jenkins是一个持续集成的工具,通过它可以代码克隆、编译以及程序加密自动化，其实它也是通过批处理来实现的，ant、gradle和jenkins的具体用法自行谷歌，使用起来很简单，目前android studio和github上很多功能都是通过gradle来编译的。

专业术语介绍

以下解释完全是本人的理解，详细解释可自行谷歌。

- **版本迭代：按照需求优先级，在保证基本功能OK后持续开发和升级，这样能够降低软件开发的风险，并且能够及时解决用户反馈的问题，船小好掉头嘛；**
- **敏捷开发：小步快跑，大概意思就是不要过于注重文档，要注重当面交流，能够在实现时高保真的还原用户的需求场景，并且能够快速地解决用户的需求。**
- **单元测试：白盒测试的一种，对核心方法通过写程序来测试自己的程序，单元测试的目的是让你有意识地降低程序间的耦合，保证每一个方法都是最小单元，但这对于测试程序逻辑是没有帮助，这是我自己的理解。。。**
- **灰度发布：先找一部分用户来使用即将发布的程序（这部分用户可以是随机抽取、制定年龄段、指定地区或者通过某种方式知道他是活跃用户），在测试的过程中给与用户一点好处让用户写用户体验报告、反馈问题等方式来发现程序存在的问题和缺陷；**
- **DA统计：也叫后台统计，通过在程序中埋点的方式，在有网络的情况下将用户的操作行为和数据上传到后台，将每个用户的信息都上传回来就叫大数据，通过建模对这些数据分析就叫大数据分析。**

- **开放平台：比如分享到QQ空间、分享到微信、讯飞语音、友盟的后台统计、天气、地图等等都叫做开放平台，它提供了一些开放的接口给开发者，方便开发者使用它的服务，开放平台多数服务都是免费的，但有时候也可能不稳定，比如用的人少它自然就活不下去了，然后就没有然后了。**
- **同行评审：你的同行和你一起看看你的代码，发现是否有问题；**
- **结对编程：在写代码的过程中，有个人坐在你旁边或者你坐在别人旁边，编写边讨论，降低程序出现逻辑和低级错误的概率。**

Android开发资源

参见我的另一篇文章：Android开发者网址导航

建议

- 尽量阅读官方文档，这才是原汁原味、不失真的开发指导；
- 即使你认为设计程序是浪费时间，你只是喜欢写程序，至少你也得用思维导图理清思路，思维导图对于帮助你理解设计文档、理清思路有很大的帮助；
- 不要用Intent传递大量的数据，这有可能导致ANR或者报异常；
- 在退出页面后，系统不一定会及时执行onDestory方法，如果你在onDestory方法里做关闭文件、释放内存的操作可能出现退出程序又立即进入时，由于需要重新初始化这些信息导致代码重入的异常；
- 在改动JNI后，运行程序之前记得卸载掉已经安装在模拟器或者真机上的该程序，如果直接运行，android不会load最新编译的so，也就不能立即看到修改后的效果；
- 代码至少每天备份一次，或者是完善一个功能就备份一次，不要堆积之后一次性备份，因为在你的代码出问题需要回溯代码时你需要从服务器上重新取代码，同时也可以避免代码不是最新导致最后和其他人合并时不知道改了哪些地方；
- 将打印信息封装成一个方法，用一个标志位控制这个这个方法的方法体是否需要执行，这样在由debug版释放到release版本时，不需要傻傻地一行一行地去掉代码，你只需要改变标志位的值就可以了；
- 对于有返回值的JNI函数，即使你不返回任何值，用NDK编译JNI的时候也不会报错，所以在写JNI代码的时候，一定要仔细检查代码；
- JNI频繁读写文件操作会影响程序的运行性能，可以考虑一次性在内存中申请一块大内存作为缓存空间，用这种空间换时间的方式可以大大提高程序的运行效率；
- 不要指望类的finalize方法去处理需要回收和销毁的工作，因为finalize是系统回调的方法，调用时机不可预见，切记；
- 使用文件流、Cursor时，使用结束后记得一定要关闭，否则可能导致内存泄漏，严重的情况可能引发程序崩溃；
- 优先使用Google搜索引擎(少用百度)，如果不能正常使用Google搜索引擎建议通过代理、VPN、修改hosts文件等方式搭建梯子。这里提供一个免费的谷歌搜索引擎
- 对于不需要使用硬件加速的activity（没有动画效果、视频播放以及各种多媒体文件的操作都可以关掉硬件加速），在AndroidManifest.xml文件中通过“android:hardwareAccelerated="false"”关掉硬件加速可节省应用内存；
- 对于需要横竖屏转换的应用，又不想在横竖屏切换的时候重新跑onCreate方法，可以在AndroidManifest.xml文件中对应的Activity标签下调用“android:configChanges="screenSize|orientation"”；
- 为了减轻应用程序主进程的内存压力，对于耗内存比较多的界面（比如视频播放界面、flash播放界面等），可以在AndroidManifest.xml文件中对应的Activity标签下调用“android:process=".processname"”单开一个进程，但在退出这个界面的时候一定要在该界面的onDestory方法中调用System的kill方法来杀掉该进程；
- 在res/values/arrays.xml文件中定义的单个数组的元素个数不宜过大，过大会导致加载数据时非常慢，有时候你需要使用数组资源时数据有可能还没加载完成；
- 一个Activity中最耗费内存的是activity的背景（多数情况如此，特别是对于分辨率很大的机器，一个界面的背景算下来都需要好几兆内存），所以在程序界面较多时，可以考虑将图片转换成静态的drawable，然后多个activity共用这一张背景图；
- 可以通过为application、activity自定义主题的方式来关掉多点触摸功能，只需要在自定义的主题下添加这两个标签：

```
<itemname="android:windowEnableSplitTouch">falseitem><itemname="android:splitMotionEvents">falseitem>
```

- 很多游戏进入时，播放的片头动画多数是一个视频文件；
- Android单个dex文件的方法数不能超过65536个，android使用多个dex能否避开65536方法数限制?
- 使用模拟器genymotion代替android自带模拟器（它需要虚拟机vituralbox的支持，不过官网已经提供了一个集成虚拟机的安装包了，直接下载下来安装即可），可以大大提高使用模拟器的体验（流畅、快），它也可以以插件的形式集成在Eclipse中，这是视频教程
- 给Application或者activity设置自定义主题时，最好不要设置为全透明，否则在activity按Home键回退到桌面的时候效果很渣；
- 如果你需要取消toast显示的功能，在一个类中你只需要实例化该类一次（也就是说将Toast定义成一个全局的成员变量），这样你就可以调用mToast.cancel()了，我把它写成了一个静态类：

```
publicclassToastUtils {      privateToastUtils(){      }      publicstaticvoidshowToast( Context context, String toast ){          if( null == mToast ){
mToast = Toast.makeText( context, toast, Toast.LENGTH_LONG );          }else{              mToast.setText( toast );          }          mToast.show( );      }
publicstaticvoidcancel(){          if( null != mToast ){              mToast.cancel( );          }      }      publicstatic Toast mToast = null;  }
```

- 你可以定义一个静态类来实现防止按钮被重复点击导致重复执行一段代码的问题：

```
/** * 按钮重复点击 * * */publicclass BtnClickUtils {      privateBtnClickUtils( ){      }                  publicstatic boolean isFastDoubleClick(){          long
time = System.currentTimeMillis();          long timeD = time - mLastClickTime;          if ( 0 < timeD && timeD < 1000) {              returntrue;
}          mLastClickTime = time;          returnfalse;          }      privatestaticlong mLastClickTime = 0;  }
```

- 放在apk的assets或者raw目录下的数据文件最好做加密处理，在需要使用的时候才解密，这样可以避免在apk被他人破解时数据也被破解的问题；
- 最好不要再activity的onCreate方法里面调用popupwindow的show方法，有可能由于activity没有完全初始化导致程序异常（**android.view.WindowManager$BadTokenException: Unable to add window -- token null is not valid**），如果非要在一进activity就显示popupwindow，建议用handler.post、View.postDelay来处理；
- 对于自定义View，在构造方法里面是获取不到视图的宽高的（此时获取长宽都为0），需要在onMeasure方法中或者跑了onMeasure方法后才能够获取到视图的宽高，不过你可以通过在构造方法里面强制测量视图的宽高来实现在构造方法里获取视图的宽高信息，具体见MeasureSpec介绍及使用详解
- 如果你觉得在安装Eclipse后还需要配置android开发环境很麻烦，你可以直接使用ADT Bundle，它是一个懒人套餐，下载下来就可以用了，可以在这里下载。
- 有时间看看阿里技术嘉年华、InfoQ演讲与访谈、Google IO视频，可以学习到一些解决问题、做大项目的经验。
- 当应用中动画比较多，并且动画都是通过图片来切换的时候，可以考虑借用Cocos的精灵表单思想，这样就可以避免图片命名的烦恼。

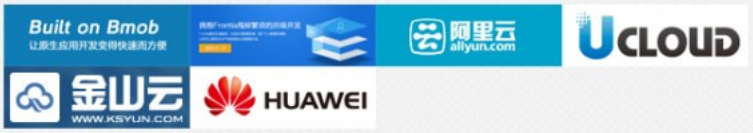
工具推荐

- 代码对比：Beyond compare
- 屏幕取色：ColorPix
- 梯子：红杏
- 思维导图： mindmanager
- 在线工具：在线工具

Android应用开发第三方解决方案

下图为Android应用开发第三方解决方案汇总，有些可以借助第三方平台搞定的就尽量不要自己搞，一是可以节省成本，二是你没人家专业，原文链接：Android应用开发第三方解决方案

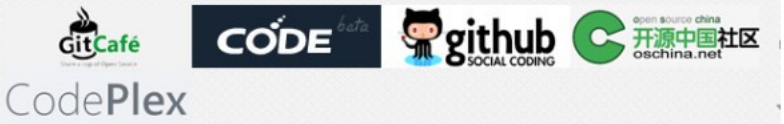
安卓后端存储解决方案：



安卓地图定位解决方案：

安卓应用测试解决方案：

安卓代码托管解决方案：



安卓音视频解决方案:

安卓移动支付解决方案：

安卓社会化问题解决方案：

安卓自动更新解决方案：

安卓文件存储解决方案：

安卓轻开发解决方案：

安卓应用安全解决方案：

安卓图像处理解决方案：

安卓用户反馈解决方案：

安卓第三方登录解决方案：

安卓反编译解决方案：

第三方解决方案