

宝贵建议请发送至：wangzhenyang@itcast.cn



黑马程序员
itheima.com 上海

- 编程，始于黑马

Android 面试宝典

Version 3.0

By 阳哥

1. 内容介绍.....	17
2. JavaSE 基础 (★★)	18
一、Java 面向对象思想.....	19
1、面向对象都有哪些特性以及你对这些特性的理解.....	19
2、 如何理解 clone 对象 (2012/2/24)	20
二、Java 中的多态.....	24
1、Java 中实现多态的机制是什么?	24
三、Java 的异常处理.....	24
1、Java 中异常分为哪些种类.....	24
2、 调用下面的方法，得到的返回值是什么.....	25
3、 error 和 exception 的区别 (2017-2-23)	25
4、 java 异常处理机制 (2017-2-23)	26
5、 请写出你最常见的 5 个 RuntimeException (2017-2-23)	26
四、Java 的数据类型.....	27
1、Java 的基本数据类型都有哪些各占几个字节.....	27
2、 String 是基本数据类型吗？可以被继承吗?	27
五、Java 的 IO.....	27
1、Java 中有几种类型的流.....	27
2、 字节流如何转为字符流.....	28
3、 如何将一个 java 对象序列化到文件里.....	28
4、 字节流和字符流的区别 (2017-2-23)	28
六、Java 的集合.....	29

1、HashMap 排序题，上机题。(本人主要靠这道题入职的第一家公司).....	29
2、 集合的安全性问题.....	31
3、 ArrayList 内部用什么实现的？(2015-11-24)	32
4、 并发集合和普通集合如何区别？(2015-11-24)	37
5、 List 的三个子类的特点 (2017-2-23)	38
6、 List 和 map 的区别 (2017-2-23)	39
7、 HashMap 和 Hashtable 有什么区别？(2017-2-23)	39
8、 数组和链表分别比较适合用于什么场景，为什么？(2017-2-23)	39
9、 Java 中 ArrayList 和 LinkedList 区别？(2017-2-23)	41
10、 List a=new ArrayList()和 ArrayList a =new ArrayList()的区别？(2017-2-24)	42
11、 要对集合更新操作时，ArrayList 和 LinkedList 哪个更适合？(2017-2-24).....	42
12、 请用两个队列模拟堆栈结构 (2017-2-24)	46
七、 Java 的多线程.....	47
1、 多线程的两种创建方式.....	47
2、 在 java 中 wait 和 sleep 方法的不同？.....	47
3、 synchronized 和 volatile 关键字的作用.....	47
4、 分析线程并发访问代码解释原因.....	49
5、 什么是线程池，如何使用？.....	50
6、 请叙述一下您对线程池的理解？(2015-11-25)	51
7、 线程池的启动策略？(2015-11-25)	51
8、 如何控制某个方法允许并发访问线程的个数？(2015-11-30)	53
9、 三个线程 a、 b、 c 并发运行， b,c 需要 a 线程的数据怎么实现 (上海 3 期学员提供)	54

10、 同一个类中的 2 个方法都加了同步锁 ,多个线程能同时访问同一个类中的这两个方法吗 ? (2017-2-24)	57
11、 什么情况下导致线程死锁 ,遇到线程死锁该怎么解决 ? (2017-2-24)	59
12、 Java 中多线程间的通信怎么实现?(2017-2-24)	66
3. JavaSE 高级 (★★)	69
一、 Java 中的反射	69
1、 说说你对 Java 中反射的理解	69
二、 Java 中的动态代理	69
1、 写一个 ArrayList 的动态代理类 (笔试题)	69
2、 动静态代理的区别 ,什么场景使用 ? (2015-11-25)	70
三、 Java 中的设计模式&回收机制	70
1、 你所知道的设计模式有哪些	70
2、 单例设计模式	70
3、 工厂设计模式	71
4、 建造者模式 (Builder)	75
5、 适配器设计模式	76
6、 装饰模式 (Decorator)	78
7、 策略模式 (strategy)	79
8、 观察者模式 (Observer)	80
9、 JVM 垃圾回收机制和常见算法	82
10、 谈谈 JVM 的内存结构和内存分配	86
11、 Java 中引用类型都有哪些 ? (重要)	87

12、 heap 和 stack 有什么区别 (2017-2-23)	90
四、 Java 的类加载器 (2015-12-2)	96
1、 Java 的类加载器的种类都有哪些?	96
2、 类什么时候被初始化?	96
3、 Java 类加载体系之 ClassLoader 双亲委托机制 (2017-2-24)	97
五、 Java8 的新特性以及使用 (2016-9-7)	101
4. Android 基础 (★★★)	101
一、 Android 基本常识	101
1、 写 10 个简单的 linux 命令	101
2、 书写出 android 工程的目录结构	102
3、 什么是 ANR 如何避免它?	103
4、 谈谈 Android 的优点和不足之处	103
5、 一条最长的短信息约占多少 byte?	104
6、 sim 卡的 EF 文件有何作用? (不用看, 废弃)	104
7、 如何判断是否有 SD 卡?	107
8、 dvm 的进程和 Linux 的进程, 应用程序的进程是否为同一个概念?	107
9、 Android 程序与 Java 程序的区别?	108
10、 启动应用后, 改变系统语言, 应用的语言会改变么?	108
11、 请介绍下 adb、ddms、aapt 的作用	108
12、 ddms 和 traceview 的区别	109
13、 补充知识: TraceView 的使用	109
14、 Android 中数据存储方式有哪些?	113

15、 DVM 和 JVM 的区别？	114
16、 谈一谈 Android 的安全机制	114
17、 Android 的四大组件都需要在清单文件中注册吗？	114
18、 在 Android 中进程的级别有哪些？	115
19、 sp 频繁操作有什么后果？sp 能存多少数据？（上海 3 期学员提供）	115
20、 描述一下 Android 的系统架构（2017-2-23）	115
21、 解释一下 Android 程序运行时权限与文件系统权限的区别？（2017-2-23）	116
22、 Android6.0 的权限机制（2017-2-23）	117
23、 AndroidManifest.xml 中的 targetSDK 设置有什么作用？（2017-2-24）	120
二、 Activity	123
1、 什么是 Activity？	123
2、 请描述一下 Activity 生命周期	123
3、 Activity 的状态都有哪些？	124
4、 如何保存 Activity 的状态？	124
5、 两个 Activity 之间跳转时必然会执行的是哪几个方法？（重要）	125
6、 横竖屏切换时 Activity 的生命周期	125
7、 如何将一个 Activity 设置成窗口的样式？	126
8、 如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？	126
9、 请描述一下 Activity 的启动模式都有哪些以及各自的特点	127
10、 一个启动模式为 singleTop 的 activity，如果再试图启动会怎样？面试官想问的是 onNewIntent() （2015-11-24）	137
11、 两个 Activity 之间传递数据，除了 intent，广播接收者，content provider 还有啥方式？(2017-2-23)	

.....	137
12、 怎样在两个 Activity 之间传递一张图片 (2017-2-23)	138
13、 如何实现切换主题功能 ? (2017-2-23)	139
14、 Android 中 Activity 是如何启动的 ? (2017-2-24)	142
三、 Service	150
1、 Service 是否在 main thread 中执行, service 里面是否能执行耗时的操作?	150
2、 Activity 怎么和 Service 绑定 , 怎么在 Activity 中启动自己对应的 Service ?	150
3、 请描述一下 Service 的生命周期	150
4、 什么是 IntentService ? 有何优点 ?	152
5、 说说 Activity、Intent、Service 是什么关系	154
6、 Service 和 Activity 在同一个线程吗	154
7、 Service 里面可以弹吐司么 ?	155
8、 如何让一个 Service 成为前置进程 ?	155
9、 Service 的 onStartCommand 方法有几种返回值 ? 各代表什么意思 ?	155
10、 Service 的 onRebind (Intent) 方法在什么情况下会执行 ?	156
11、 Activity 调用 Service 中的方法都有哪些方式 ?	156
12、 Activity 如何给 Service 发送 Message ? (2015.10.18)	159
13、 Service 如何给 Activity 发送 Message ? (2015.10.18)	160
14、 子线程不能代替 service 吗 ? (2017-2-23)	164
四、 BroadCastReceiver	164
1、 请描述一下 BroadcastReceiver	164
2、 在 manifest 和代码中如何注册和使用 BroadcastReceiver	165

3、 BroadcastReceiver 的生命周期.....	165
4、 如何让自己的广播只让指定的 app 接收 (2015.09.02)	166
5、 什么是最终广播接收者？.....	167
6、 广播的优先级对无序广播生效吗？.....	167
7、 动态注册的广播优先级谁高？.....	167
8、 如何判断当前 BroadcastReceiver 接收到的是有序广播还是无序广播？ (2015-10-16)	167
9、 Android 引入广播机制的用意 (2017-2-23)	167
10、 网络状态改变是无序广播还是有序广播，安装了，没启动过，会接受这个广播么？ (2017-2-24)	168
五、 ContentProvider&数据库.....	168
1、 请介绍下 ContentProvider 是如何实现数据共享的？.....	168
2、 为什么要用 ContentProvider？它和 sql 的实现上有什么差别？.....	169
3、 说说 ContentProvider、ContentResolver、ContentObserver 之间的关系.....	169
4、 如何访问 asserts 资源目录下的数据库？.....	169
5、 如何在高并发下进行数据库查询？ (2015-11-25)	170
六、 Android 中的布局.....	170
1、 Android 中常用的布局都有哪些.....	170
2、 谈谈 UI 中，Padding 和 Margin 有什么区别？.....	171
3、 使用权重如何让一个控件的宽度为父控件的 1/3？.....	171
4、 Android 中布局的优化措施都有哪些？.....	171
5、 android:layout_gravity 和 android:gravity 的区别？.....	172
七、 ListView.....	172
1、 ListView 如何提高其效率？.....	172

2、 ViewHolder 为什么要声明为静态类？	173
3、 在 Activity 中使用 Handler 的时候如何去除警告信息？	173
4、 谈谈 ListView 中的 MVC 思想？	173
5、 ListView 使用了哪些设计模式？	174
6、 当 ListView 数据集改变后，如何更新 ListView？	174
7、 ListView 如何实现分页加载	174
8、 ListView 可以显示多种类型的条目吗？	175
9、 ListView 如何定位到指定位置	175
10、 如何在 ScrollView 中如何嵌入 ListView	175
11、 ListView 中如何优化图片	178
12、 ListView 中图片错位的问题是如何产生的	179
13、 scrollView 嵌套 listview 方式除了测量还有什么方法？（2015-11-29）	179
八、 JNI&NDK	181
1、 在 Android 中如何调用 C 语言	181
2、 请介绍一下 NDK	182
3、 JNI 调用常用的两个参数	182
九、 Android 中的网络访问	182
1、 Android 中如何访问网络	182
2、 如何解析服务器传来的 JSON 文件	183
3、 如何解析服务器传来的 XML 格式数据	185
4、 如何从网络上加载一个图片显示到界面	187
5、 如何播放网络视频	187

6、 常见的访问网络 API 都有哪些？	187
十、 Intent	188
1、 Intent 传递数据时，可以传递哪些类型数据？	188
2、 Serializable 和 Parcelable 的区别	188
3、 请描述一下 Intent 和 IntentFilter	189
4、 under what condition could the code sample below crash your application?How would you modify the code to avoid this potential problem?Explain your answer?(重要)	191
5、 what are Activity and Fragment?where and why should you use one over the other?	192
6、 can you use an intent to provide data to a ContentProvider ? if not ,what would be the proper mechanism for doing this?	192
十一、 Fragment	192
1、 Fragment 跟 Activity 之间是如何传值的？	192
2、 描述一下 Fragment 的生命周期	193
3、 Fragment 的 replace 和 add 方法的区别（ 2015.8.30 ）	193
4、 Fragment 如何实现类似 Activity 栈的压栈和出栈效果的？（ 2015.8.30 ）	194
5、 ViewPager+Fragment 的左右滑动，如何实现 Fragment 的懒加载，Viewpager 默认加载几个？ （ 2017-2-24 ）	196
5. Android 高级（ ★★★ ）	198
一、 Android 性能优化	198
1、 如何对 Android 应用进行性能分析	198
2、 什么情况下会导致内存泄露	200
3、 如何避免 OOM 异常	204

4、 Android 中如何捕获未捕获的异常(重要).....	206
5、 Android 性能优化博客.....	207
6、 Android 动态加载机制.....	209
7、 如果加载高清大图片，不用第三方，不压缩，怎么处理防止 OOM (2017-2-24)	212
二、 Android 屏幕适配.....	215
1、 屏幕适配方式都有哪些.....	215
2、 屏幕适配的处理技巧都有哪些.....	221
3、 dp 和 px 之间的关系.....	224
三、 AIDL.....	224
1、 什么是 AIDL 以及如何使用.....	224
四、 自定义控件.....	225
1、 如何自定义一个控件.....	225
2、 请描述一下 View 的绘制流程.....	226
3、 View,SurfaceView,GLSurfaceView 有什么区别？ (2017-2-23)	228
五、 Android 中的事件处理.....	232
1、 Handler 机制.....	232
2、 事件分发机制.....	233
3、 在 Android 中主线程如何给子线程发 Message？ (2015-12-1) 重要.....	235
六、 Android 签名.....	237
1、 简单描述下 Android 数字签名.....	238
2、 使用 Eclipse 如何生成数字签名.....	239
七、 Android 中的动画.....	239

1、 Android 中的动画有哪几类，它们的特点和区别是什么	239
2、 如何修改 Activity 进入和退出动画	239
八、 编写自己的框架	240
1. 编写 ViewUtils 框架 (2016-1-20)	240
2. 编写 AsyncTask 框架 (2016-1-20)	246
3. 编写 ImageLoader 框架 (2016-1-20)	246
九、 其他知识 (集大众智慧)	250
1、 AsyncTask 如何使用	250
2、 都使用过哪些框架、平台	255
3、 Glide 原理 (2015-11-29)	255
4、 Android 四大著名图片处理框架 (2015-11-29)	257
5、 都使用过哪些自定义控件	267
6、 Android 程序员进阶必备网站 (2015-11-29)	267
7、 volley 的原理 (2015-11-29)	273
8、 okhttp 的原理 (2015-11-29)	278
9、 ViewPagerindicator 的原理 (2015-11-29)	279
10、 slidingmenu 的原理 (2015-11-29)	279
11、 RecyclerView 的原理 (2017-2-25)	280
十、 网络协议	281
1、 Http 和 Https 有什么区别？ (2017-2-24)	281
2、 简述 Socket 通讯编程 (2017-2-25)	286
3、 Binder 机制 (2017-2-25)	289

4、 如何保证网络传输数据的安全性 (2017-2-25)	291
5、 自己设计一个 Push 推送服务，需要考虑到那些点 (2017-2-25)	291
6. Android 项目 (★★★)	300
1、 如何让 LinearLayout 自动换行如下图的颜色分类所示。	301
2、 ImageLoader 在项目中的使用	301
3、 Java 和 javaScript 互相调用 (webview 和 js 的互相调用)	302
4、 PopupWindow 弹出层在项目中的使用	304
5、 Notification 在 Android 中的使用	305
6、 带索引的 ListView 在 Android 中的应用	306
7、 随手势滑动而消失 Activity 的使用	306
8、 TouchGallery 在 Android 中的应用	308
9、 TextView 显示富文本	308
10、 CircleImageView 实现圆形图片	309
11、 网易新闻客户端频道管理的实现 (2015-11-25)	310
12、 Android 瀑布流的实现	311
13、 监听键盘事件 (2015-11-29)	314
14、 可以按照字母排序的 ListView (2015-11-29)	317
15、 省市区三级联动	318
16、 购物客户端二级菜单	319
17、 二维码扫描	320
18、 微信图片选择器	320
19、 如何从 html 的一个动作打开 app 并跳转到指定的 Activity(2017-2-25)	321

20、 AAR 库怎么来配置(2017-2-25).....	323
21、 AndroidStudio 项目中 Module Library 和 App 的 Mainifest 编译合并 minSdkVersion 有什么限制关系(2017-2-25).....	325
22、 不借助第三方怎么显示圆形图片(2017-2-25).....	326
7. 项目面试常见问题 (★★★)	326
1. 公司人员构成.....	326
2. 开发周期.....	327
3. 项目中遇到的难题.....	327
4. 项目中最大的收获.....	328
5. 项目是如何上线的.....	328
6. 项目是如何盈利的.....	328
7. 绘制项目架构图.....	328
8. 项目开发流程.....	329
9. 你在项目中的角色.....	329
10. 你负责项目中的哪些模块.....	330
11. 讲讲你负责模块的具体实现.....	330
12. 项目中都用到了哪些第三发框架.....	330
13. 有没有自己写过框架.....	330
14. 业余时间你是如何提高自己 (学习) 的.....	330
15. 有没有自己的技术 blog.....	330
16. 你的职业规划.....	330
17. 为什么离职.....	330

18. 为什么选择我们公司.....	330
19. 说说你们项目的亮点和不足.....	330
20. 你们的项目是如何保持风格一致的.....	330
21. 项目架构是如何搭建的.....	330
22. 屏幕适配是如何解决的.....	331
23. 都看过哪些源码.....	331
24. 项目版本是如何升级的.....	331
25. 用的什么版本控制工具.....	331
26. 你能独立开发吗.....	331
27. App 跟服务器是如何交互的.....	331
28. 需求文档写过吗.....	331
29. 接口文档写过吗.....	331
30. 云服务器都用过哪些.....	332
31. 第三方平台都用过哪些.....	332
8. 面试实战记录 (★★).....	332
挑战公司 No.1：上海创梯信息科技有限公司.....	332
挑战公司 No.2：上海复深蓝信息技术有限公司.....	337
挑战公司 No.3：中阜投融资产管理股份有限公司---中投融.....	347
挑战公司 No.4：上海游竞网络科技有限公司---PLU.....	355
挑战公司 No.5：一号店旗下壹药网.....	363
挑战公司 No.6：聚信租赁.....	373
他人面试心得.....	380

一、 一个五年 Android 开发者百度、阿里、聚美、映客的面试心经	380
二、 近期 Android 面试经历总结	399
9. BAT 面试题	405
1. 腾讯 2016 年面试题	405
1. 已知一棵二叉树，如果先序遍历的节点顺序是：ADCEFGHB，中序遍历是：CDFEGHAB，则后序遍历结果为：(D)	405
2. 下列哪两个数据结构，同时具有较高的查找和删除性能？(CD)	407
3. 下列排序算法中，哪些时间复杂度不会超过 $n \log n$ ？(BC)	408
4. 初始序列为 1 8 6 2 5 4 7 3 一组数采用堆排序，当建堆（小根堆）完毕时，堆所对应的二叉树中序遍历序列为：(A)	409
5. 当 $n = 5$ 时，下列函数的返回值是：(A)	410
6. S 市 A，B 共有两个区，人口比例为 3 : 5，据历史统计 A 区的犯罪率为 0.01%，B 区为 0.015%，现有一起新案件发生在 S 市，那么案件发生在 A 区的可能性有多大？(C)	410
7. Unix 系统中，哪些可以用于进程间的通信？(ABCD)	411
8. 静态变量通常存储在进程哪个区？(C)	412
9. 如何提供查询 Name 字段的性能 (B)	412
10. IP 地址 131.153.12.71 是一个 (B) 类 IP 地址	413
11. 浏览器访问某页面，HTTP 协议返回状态码为 403 时表示：(B)	413
12. 如果某系统 $15 \times 4 = 112$ 成立，则系统采用的是 (A) 进制	414
13. TCP 和 IP 分别对应了 OSI 中的哪几层？(CD)	414
14. 一个栈的入栈序列是 A，B，C，D，E，则栈的不可能的输出序列是？(C)	415
15. 同一进程下的线程可以共享以下？(BD)	416

16. 对于派生类的构造函数，在定义对象时构造函数的执行顺序为？（ D ）	417
17. 递归函数最终会结束，那么这个函数一定？（ B ）	417
18. 编译过程中，语法分析器的任务是（ BCD ）	417
19. 同步机制应该遵循哪些基本准则？（ ABCD ）	418
20. 进程进入等待状态有哪几种方式？（ D ）	418
21. 设计模式中，属于结构型模式的有哪些？（ BC ）	418
10. Android 最新技术	419
1. Android 架构思考(模块化、多进程)	419
2. Android 热修复	419
3. Android 性能优化	420
4. Android 动态加载	420
5. Android AOP	421
6. MVP+Retrofit+RxJava 网络请求框架	421
7. RxJava	422
8. 安居客 Android 项目架构演进	422
9. Android 进程保活	423
10. Android 状态栏	423
11. Android Context 的种类	424
附录：更新记录	425

Android 面试宝典-V3.0

1. 内容介绍

该面试宝典不仅收录了本人亲身面试遇到的问题，还收录了从黑马学子那里收集过来的问题。在以后的工作中本人也会不断的更新和充实该面试宝典，当然也希望大家能够多多奉献比较优质的面试题。

该面试宝典不仅展示了常见的面试问题以及回答技巧，还详细讲解了每一道题所包含的知识点，让黑马学子不仅知其然，更知其所以然。

本人的面试实战记录发布在黑马论坛：<http://bbs.itheima.com/thread-196394-1-1.html>

大家可以访问上面的网址，通过阳哥的实战记录略微感知一下真实面试的情况，从中学习一些面试技巧以便让自己在未来的面试中能够得心应手，顺利拿到自己喜欢的 offer。

注意：该面试宝典仅供参考，本人无法对其所有内容的正确性和准确性做保证。由于本人的知识水平有限加之编写时间仓促因此难免有不少 bug 的存在，希望大家见谅。

如果您有不错的知识或者面试题，您可以发送面试题到 wangzhenyang@itcast.cn，本人将不胜感激。让天下没有难学的知识，希望你我的努力能帮到更多的莘莘学子。

写在面试宝典 V3.0 前面的话：

该面试宝典正在被越来越多的分校学员使用，一方面自己很欣慰，另一方面自己也感觉很愧疚。愧疚的是本人已经有很长一段时间没有更新该宝典了，技术日新月异的当今，稍有懈怠就会落伍，因此该宝典是时候更新了。

世间事，很多都可投机取巧，但技术却必须靠日积月累的努力来提高。从 V3.0 版本开始，本宝典更加注重的是知识的掌握，而不仅仅是对面试题的应付。

V3.0 版本主要添加了如下方面的知识点：

- 1、OKHttp 和 Retrofit 的使用
- 2、Java8 的新特性

- 3、RXJava 和 RXAndroid
- 4、EventBus 的内部原理
- 5、Android6.0 的权限机制
- 6、MVP 和 MVVM 设计思想
- 7、Android 内存泄露检测工具之 MAT 和 LeakCanary 的使用
- 8、Android 的插件式开发
- 9、混合开发之 ReactNative 如何开发 Android 项目
- 10、Crash 追踪之腾讯 Bugly
- 11、第三方登录的接入
- 12、应用加固平台
- 13、Android5.0 之后新控件的使用
- 14、Context 上下文的理解
- 15、Android 的多分包技术
- 16、Android 热修复
- 17、对称和非对称加密
- 18、如何实现长连接以及和短连接的区别
- 19、其他待定

2. JavaSE 基础 (★★)

有人可能会问对于我们学 Android 的同学来讲，面试还会问 Java 基础吗？答案是会的，但是不会太多，因此我给了两颗星的重要程度。一般笔试的时候出现 java 基础题的概率比较大，口头面试的时候比较少，比如自己在面试的时候一些对基础知识比较看重的面试官会深究着 Java 基础去问，比如问你异常的类型以及处理方式，集合的体系结构

等等。

一、Java 面向对象思想

1、面向对象都有哪些特性以及你对这些特性的理解

◆ 继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段。

◆ 封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口。

◆ 多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当 A 系统访问 B 系统提供的服务时，B 系统有多种提供服务的方式，但一切对 A 系统来说都是透明的。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1. 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2. 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

◆ 抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

2、如何理解 clone 对象 (2012/2/24)

2.1 为什么要用 clone?

首先，克隆对象是很有必要的，当一个对象需要被多人操作，但是又不想相互影响，需要保持原对象的状态，这时，就会克隆很多个相同的对象。

2.2 new 一个对象的过程和 clone 一个对象的过程区别

new 操作符的本意是分配内存。程序执行到 new 操作符时，首先去看 new 操作符后面的类型，因为知道了类型，才能知道要分配多大的内存空间。分配完内存之后，再调用构造函数，填充对象的各个域，这一步叫做对象的初始化，构造方法返回后，一个对象创建完毕，可以把他的引用（地址）发布到外部，在外部就可以使用这个引用操纵这个对象。

clone 在第一步是和 new 相似的，都是分配内存，调用 clone 方法时，分配的内存和原对象（即调用 clone 方法的对象）相同，然后再使用原对象中对应的各个域，填充新对象的域，填充完成之后，clone 方法返回，一个新的相同的对象被创建，同样可以把这个新对象的引用发布到外部。

2.3 clone 对象的使用

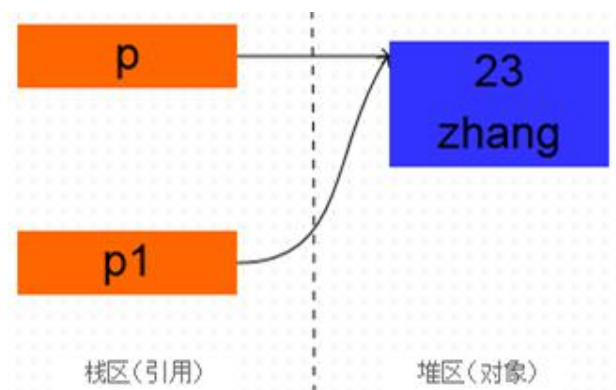
2.3.1 复制对象和复制引用的区别

```
1. Person p = new Person(23, "zhang");
2. Person p1 = p;
3. System.out.println(p);
4. System.out.println(p1);
```

当 Person p1 = p; 执行之后，是创建了一个新的对象吗？首先看打印结果：

```
1. com.itheima.Person@2f9ee1ac
2. com.itheima.Person@2f9ee1ac
```

可以看出，打印的地址值是相同的，既然地址都是相同的，那么肯定是同一个对象。p 和 p1 只是引用而已，他们都指向了一个相同的对象 Person(23, "zhang")。可以把这种现象叫做引用的复制。上面代码执行完成之后，内存中的情景如下图所示：



而下面的代码是真真正正的克隆了一个对象。

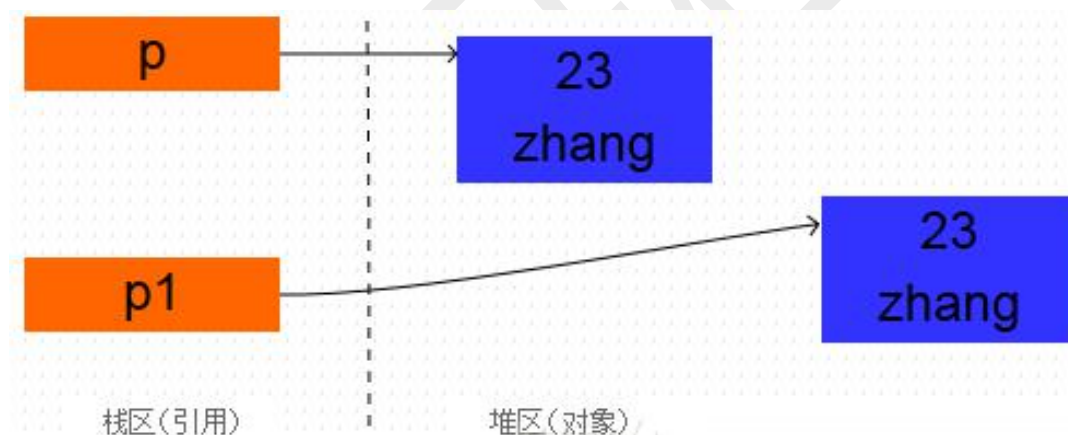
```
3. Person p = new Person(23, "zhang");
4. Person p1 = (Person) p.clone();
5. System.out.println(p);
6. System.out.println(p1);
```

从打印结果可以看出，两个对象的地址是不同的，也就是说创建了新的对象，而不是把原对象的地址赋给了一个

新的引用变量：

```
1. com.itheima.Person@2f9eelac
2. com.itheima.Person@67f1fba0
```

以上代码执行完成后，内存中的情景如下图所示：



2.3.2 深拷贝和浅拷贝

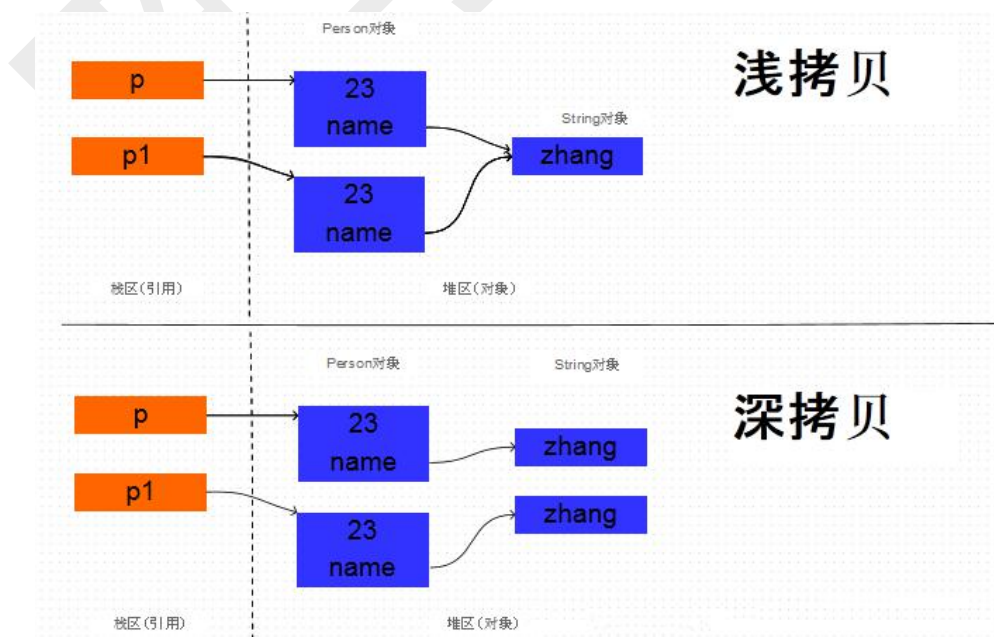
上面的示例代码中，Person 中有两个成员变量，分别是 name 和 age，name 是 String 类型，age 是 int 类型。

代码非常简单，如下所示：

```
1. public class Person implements Cloneable{
```

```
2.     private int age ;
3.     private String name;
4.     public Person(int age, String name) {
5.         this.age = age;
6.         this.name = name;
7.     }
8.     public Person() {}
9.     public int getAge() {
10.        return age;
11.    }
12.    public String getName() {
13.        return name;
14.    }
15.    @Override
16.    protected Object clone() throws CloneNotSupportedException {
17.        return (Person)super.clone();
18.    }
19. }
```

由于 age 是基本数据类型，那么对它的拷贝没有什么疑议，直接将一个 4 字节的整数值拷贝过来就行。但是 name 是 String 类型的，它只是一个引用，指向一个真正的 String 对象，那么对它的拷贝有两种方式：直接将原对象中的 name 的引用值拷贝给新对象的 name 字段，或者是根据原 Person 对象中的 name 指向的字符串对象创建一个新的相同的字符串对象，将这个新字符串对象的引用赋给新拷贝的 Person 对象的 name 字段。这两种拷贝方式分别叫做浅拷贝和深拷贝。深拷贝和浅拷贝的原理如下图所示：



下面通过代码进行验证。如果两个 Person 对象的 name 的地址值相同，说明两个对象的 name 都指向同一个 String 对象，也就是浅拷贝，而如果两个对象的 name 的地址值不同，那么就说明指向不同的 String 对象，也就是在拷贝 Person 对象的时候，同时拷贝了 name 引用的 String 对象，也就是深拷贝。验证代码如下：

```
1. Person p = new Person(23, "zhang");
2. Person p1 = (Person) p.clone();
3. String result = p.getName() == p1.getName()
4.         ? "clone 是浅拷贝的" : "clone 是深拷贝的";
5. System.out.println(result);
```

打印结果为：

6. clone 是浅拷贝的

所以，clone 方法执行的是浅拷贝，在编写程序时要注意这个细节。

如何进行深拷贝：

由上一节的内容可以得出如下结论：如果想要深拷贝一个对象，这个对象必须要实现 Cloneable 接口，实现 clone 方法，并且在 clone 方法内部，把该对象引用的其他对象也要 clone 一份，这就要求这个被引用的对象必须也要实现 Cloneable 接口并且实现 clone 方法。那么，按照上面的结论，实现以下代码 Body 类组合了 Head 类，要想深拷贝 Body 类，必须在 Body 类的 clone 方法中将 Head 类也要拷贝一份。代码如下：

```
1. static class Body implements Cloneable{
2.     public Head head;
3.     public Body() {}
4.     public Body(Head head) {this.head = head;}
5.     @Override
6.     protected Object clone() throws CloneNotSupportedException {
7.         Body newBody = (Body) super.clone();
8.         newBody.head = (Head) head.clone();
9.         return newBody;
10.    }
11. }
12. static class Head implements Cloneable{
13.     public Face face;
14.     public Head() {}
15.     @Override
16.     protected Object clone() throws CloneNotSupportedException {
17.         return super.clone();
18.    } }
```

```
19. public static void main(String[] args) throws CloneNotSupportedException {
20.     Body body = new Body(new Head(new Face()));
21.     Body body1 = (Body) body.clone();
22.     System.out.println("body == body1 : " + (body == body1) );
23.     System.out.println("body.head == body1.head : " + (body.head == body1.head));
24. }
```

打印结果为：

```
1. body == body1 : false
2. body.head == body1.head : false
```

二、Java 中的多态

1、Java 中实现多态的机制是什么？

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

三、Java 的异常处理

1、Java 中异常分为哪些种类

1) 按照异常需要处理的时机分为编译时异常也叫 `CheckedException` 和运行时异常也叫 `RuntimeException`。只有 java 语言提供了 `Checked` 异常，Java 认为 `Checked` 异常都是可以被处理的异常，所以 Java 程序必须显式处理 `Checked` 异常。如果程序没有处理 `Checked` 异常，该程序在编译时就会发生错误无法编译。这体现了 Java 的设计哲学：没有完善错误处理的代码根本没有机会被执行。对 `Checked` 异常处理方法有两种：

- 1 当前方法知道如何处理该异常，则用 `try...catch` 块来处理该异常。
- 2 当前方法不知道如何处理，则在定义该方法是声明抛出该异常。

运行时异常只有当代码在运行时才发行的异常，编译时不需要 try catch。Runtime 如除数是 0 和数组下标越界等，其产生频繁，处理麻烦，若显示申明或者捕获将会对程序的可读性和运行效率影响很大。所以由系统自动检测并将它们交给缺省的异常处理程序。当然如果你有处理要求也可以显示捕获它们。

2、调用下面的方法，得到的返回值是什么

```
1 public int getNum(){
2     try {
3         int a = 1/0;
4         return 1;
5     } catch (Exception e) {
6         return 2;
7     }finally{
8         //int b = 1/0;
9         return 3;
10    }
```

代码在走到第 3 行的时候抛出了一个 `MathException`，这时第四行的代码就不会执行了，代码直接跳转到 catch 语句中，走到第 6 行的时候，异常机制有这么一个原则如果在 catch 中遇到了 return 或者异常等能使该函数终止的话那么用 finally 就必须先执行完 finally 代码块里面的代码然后再返回值。因此代码又跳到第 8 行，可惜第 8 行是一个 return 语句，那么这个时候方法就结束了，因此第 6 行的返回结果就无法被真正返回。如果 finally 仅仅是处理了一个释放资源的操作，那么该道题最终返回的结果就是 2。

因此上面返回值是 3。

3、error 和 exception 的区别 (2017-2-23)

Error 类和 Exception 类的父类都是 Throwable 类，他们的区别是：

Error 类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和预防，遇到这样的错误，建议让程序终止。

Exception 类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢

复运行，而不应该随意终止异常。

Exception 类又分为运行时异常 (Runtime Exception) 和受检查的异常 (Checked Exception)，运行时异常; ArithmeticException, IllegalArgumentException，编译能通过，但是一运行就终止了，程序不会处理运行时异常，出现这类异常，程序会终止。而受检查的异常，要么用 try。。。catch 捕获，要么用 throws 字句声明抛出，交给它的父类处理，否则编译不会通过。

4、java 异常处理机制 (2017-2-23)

Java 对异常进行了分类，不同类型的异常分别用不同的 Java 类表示，所有异常的根类为 java.lang.Throwable，Throwable 下面又派生了两个子类：Error 和 Exception，Error 表示应用程序本身无法克服和恢复的一种严重问题。Exception 表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界 (ArrayIndexOutOfBoundsException)，空指针异常 (NullPointerException)、类转换异常 (ClassCastException)；普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。

java 为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须 try..catch 处理或用 throws 声明继续抛给上层调用方法处理，所以普通异常也称为 checked 异常，而系统异常可以处理也可以不处理，所以，编译器不强制用 try..catch 处理或用 throws 声明，所以系统异常也称为 unchecked 异常。

5、请写出你最常见的 5 个 RuntimeException (2017-2-23)

NullPointerException, IndexOutOfBoundsException, ClassCastException, ClassNotFoundException, IllegalArgumentException, UnsupportedOperationException。

四、Java 的数据类型

1、Java 的基本数据类型都有哪些各占几个字节

Java 有 8 种基本数据类型

byte 1

char 2

short 2

int 4

float 4

double 8

long 8

boolean 1 (boolean 类型比较特别可能只占一个 bit，多个 boolean 可能共同占用一个字节)

2、String 是基本数据类型吗？可以被继承吗？

String 是引用类型，底层用 char 数组实现的。因为 String 是 final 类，在 java 中被 final 修饰的类不能被继承，因此 String 当然不可以被继承。

五、Java 的 IO

1、Java 中有几种类型的流

字节流和字符流。字节流继承于 InputStream 和 OutputStream，字符流继承于 InputStreamReader 和 OutputStreamWriter。

2、字节流如何转为字符流

字节输入流转字符输入流通过 `InputStreamReader` 实现，该类的构造函数可以传入 `InputStream` 对象。

字节输出流转字符输出流通过 `OutputStreamWriter` 实现，该类的构造函数可以传入 `OutputStream` 对象。

3、如何将一个 java 对象序列化到文件里

在 java 中能够被序列化的类必须先实现 `Serializable` 接口，该接口没有任何抽象方法只是起到一个标记作用。

```
//对象输出流
ObjectOutputStream objectOutputStream = new ObjectOutputStream(new
FileOutputStream(new File("D://obj")));
objectOutputStream.writeObject(new User("zhangsan", 100));
objectOutputStream.close();
//对象输入流
ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream(new File("D://obj")));
User user = (User)objectInputStream.readObject();
System.out.println(user);
objectInputStream.close();
```

4、字节流和字符流的区别 (2017-2-23)

要把一片二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为 IO 流，对应的抽象类为 `OutputStream` 和 `InputStream`，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

在应用中，经常要完全是字符的一段文本输出或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流

的包装类。

底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向 IO 设别写入或读取字符串提供了一点点方便。

字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，

其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

六、Java 的集合

1、HashMap 排序题，上机题。（本人主要靠这道题入职的第一家公司）

已知一个 `HashMap<Integer, User>` 集合，`User` 有 `name (String)` 和 `age (int)` 属性。请写一个方法实现对 `HashMap` 的排序功能，该方法接收 `HashMap<Integer, User>` 为形参，返回类型为 `HashMap<Integer, User>`，要求对 `HashMap` 中的 `User` 的 `age` 倒序进行排序。排序时 `key=value` 键值对不得拆散。

Tips：要做出这道题必须对集合的体系结构非常的熟悉。`HashMap` 本身就是不可排序的，但是该道题偏偏让给 `HashMap` 排序，那我们就得想在 API 中有没有这样的 Map 结构是有序的，`LinkedHashMap`，对的，就是他，他是 Map 结构，也是链表结构，有序的，更可喜的是他是 `HashMap` 的子类，我们返回 `LinkedHashMap<Integer, User>` 即可，还符合面向接口（父类编程的思想）。

但凡是对集合的操作，我们应该保持一个原则就是能用 JDK 中的 API 就有 JDK 中的 API，比如排序算法我们不应该去用冒泡或者选择，而是首先想到用 `Collections` 集合工具类。

```
public class HashMapTest {
    public static void main(String[] args) {
        HashMap<Integer, User> users = new HashMap<>();
        users.put(1, new User("张三", 25));
        users.put(3, new User("李四", 22));
        users.put(2, new User("王五", 28));
        System.out.println(users);
        HashMap<Integer, User> sortHashMap = sortHashMap(users);
        System.out.println(sortHashMap);
        /**
         * 控制台输出内容
         * {1=User [name=张三, age=25], 2=User [name=王五, age=28], 3=User [name=李四,
age=22]}
         * {2=User [name=王五, age=28], 1=User [name=张三, age=25], 3=User [name=李四,
age=22]}
         */
    }

    public static HashMap<Integer, User> sortHashMap(HashMap<Integer, User> map) {
        // 首先拿到 map 的键值对集合
        Set<Entry<Integer, User>> entrySet = map.entrySet();

        // 将 set 集合转为 List 集合，为什么，为了使用工具类的排序方法
        List<Entry<Integer, User>> list = new ArrayList<Entry<Integer,
User>>(entrySet);
```

```
// 使用 Collections 集合工具类对 list 进行排序，排序规则使用匿名内部类来实现
Collections.sort(list, new Comparator<Entry<Integer, User>>() {

    @Override
    public int compare(Entry<Integer, User> o1, Entry<Integer, User> o2) {
        //按照要求根据 User 的 age 的倒序进行排
        return o2.getValue().getAge()-o1.getValue().getAge();
    }
});
//创建一个新的有序的 HashMap 子类的集合
LinkedHashMap<Integer, User> linkedHashMap = new LinkedHashMap<Integer,
User>();
//将 List 中的数据存储存储在 LinkedHashMap 中
for(Entry<Integer, User> entry : list){
    linkedHashMap.put(entry.getKey(), entry.getValue());
}
//返回结果
return linkedHashMap;
}
```

2、集合的安全性问题

请问 ArrayList、HashSet、HashMap 是线程安全的吗？如果不是我想要线程安全的集合怎么办？

我们都看过上面那些集合的源码（如果没有那就看看吧），每个方法都没有加锁，显然都是线程不安全的。话又说过来如果他们安全了也就没第二问了。

在集合中 Vector 和 Hashtable 倒是线程安全的。你打开源码会发现其实就是把各自核心方法添加上了 **synchronized** 关键字。

Collections 工具类提供了相关的 API，可以让上面那 3 个不安全的集合变为安全的。

```
// Collections.synchronizedCollection(c)
// Collections.synchronizedList(list)
// Collections.synchronizedMap(m)
// Collections.synchronizedSet(s)
```

上面几个函数都有对应的返回值类型，传入什么类型返回什么类型。打开源码其实实现原理非常简单，就是将集合的核心方法添加上了 **synchronized** 关键字。

3、ArrayList 内部用什么实现的？（2015-11-24）

（回答这样的问题，不要只回答个皮毛，可以再介绍一下 ArrayList 内部是如何实现数组的增加和删除的，因为数组在创建的时候长度是固定的，那么就有个问题我们往 ArrayList 中不断的添加对象，它是如何管理这些数组呢？）

ArrayList 内部是用 Object[]实现的。接下来我们分别分析 ArrayList 的构造、add、remove、clear 方法的实现原理。

一、构造函数

1) 空参构造

```
/**
 * Constructs a new {@code ArrayList} instance with zero initial capacity.
 */
public ArrayList() {
    array = EmptyArray.OBJECT;
}
```

array 是一个 Object[]类型。当我们 new 一个空参构造时系统调用了 EmptyArray.OBJECT 属性，EmptyArray 仅仅是一个系统的类库，该类源码如下：

```
public final class EmptyArray {
    private EmptyArray() {}

    public static final boolean[] BOOLEAN = new boolean[0];
    public static final byte[] BYTE = new byte[0];
    public static final char[] CHAR = new char[0];
    public static final double[] DOUBLE = new double[0];
    public static final int[] INT = new int[0];

    public static final Class<?>[] CLASS = new Class[0];
    public static final Object[] OBJECT = new Object[0];
    public static final String[] STRING = new String[0];
    public static final Throwable[] THROWABLE = new Throwable[0];
    public static final StackTraceElement[] STACK_TRACE_ELEMENT = new StackTraceElement[0];
}
```

也就是说当我们 new 一个空参 ArrayList 的时候，系统内部使用了一个 new Object[0]数组。

2) 带参构造 1

```
/**
 * Constructs a new instance of {@code ArrayList} with the specified
 * initial capacity.
 *
 * @param capacity
 *         the initial capacity of this {@code ArrayList}.
 */
public ArrayList(int capacity) {
    if (capacity < 0) {
        throw new IllegalArgumentException("capacity < 0: " + capacity);
    }
    array = (capacity == 0 ? EmptyArray.OBJECT : new Object[capacity]);
}
```

该构造函数传入一个 int 值，该值作为数组的长度值。如果该值小于 0，则抛出一个运行时异常。如果等于 0，则使用一个空数组，如果大于 0，则创建一个长度为该值的新数组。

3) 带参构造 2

```
/**
 * Constructs a new instance of {@code ArrayList} containing the elements of
 * the specified collection.
 *
 * @param collection
 *         the collection of elements to add.
 */
public ArrayList(Collection<? extends E> collection) {
    if (collection == null) {
        throw new NullPointerException("collection == null");
    }

    Object[] a = collection.toArray();
    if (a.getClass() != Object[].class) {
        Object[] newArray = new Object[a.length];
        System.arraycopy(a, 0, newArray, 0, a.length);
        a = newArray;
    }
    array = a;
    size = a.length;
}
```

如果调用构造函数的时候传入了一个 Collection 的子类，那么先判断该集合是否为 null，为 null 则抛出空指针异常。如果不是则将该集合转换为数组 a，然后将该数组赋值为成员变量 array，将该数组的长度作为成员变量 size。这里面它先判断 a.getClass 是否等于 Object[].class，其实一般都是相等的，我也暂时没想明白为什么多加了这个判断，toArray 方法是 Collection 接口定义的，因此其所有的子类都有这样的方法，list 集合的 toArray 和 Set 集合的 toArray 返回的都是 Object[] 数组。

这里讲些题外话，其实在看 Java 源码的时候，作者的很多意图都很费人心思，我能知道他的目标是啥，但是不知道他为何这样写。比如对于 ArrayList，array 是他的成员变量，但是每次在方法中使用该成员变量的时候作者都会重新在方法中开辟一个局部变量，然后给局部变量赋值为 array，然后再使用，有人可能说这是为了防止并发修改 array，毕竟 array 是成员变量，大家都可以使用因此需要将 array 变为局部变量，然后再使用，这样的说法并不是都成立的，也许有时候就是老外们写代码的一个习惯而已。

二、add 方法

add 方法有两个重载，这里只研究最简单的那个。

```
/**
 * Adds the specified object at the end of this {@code ArrayList}.
 *
 * @param object
 *         the object to add.
 * @return always true
 */
@Override public boolean add(E object) {
    Object[] a = array;
    int s = size;
    if (s == a.length) {
        Object[] newArray = new Object[s +
            (s < (MIN_CAPACITY_INCREMENT / 2) ?
                MIN_CAPACITY_INCREMENT : s >> 1)];
        System.arraycopy(a, 0, newArray, 0, s);
        array = a = newArray;
    }
    a[s] = object;
    size = s + 1;
    modCount++;
}
```

```
return true;
}
```

- 1、首先将成员变量 array 赋值给局部变量 a，将成员变量 size 赋值给局部变量 s。
- 2、判断集合的长度 s 是否等于数组的长度（如果集合的长度已经等于数组的长度了，说明数组已经满了，该重新分配新数组了），重新分配数组的时候需要计算新分配内存的空间大小，如果当前的长度小于 MIN_CAPACITY_INCREMENT/2（这个常量值是 12，除以 2 就是 6，也就是如果当前集合长度小于 6）则分配 12 个长度，如果集合长度大于 6 则分配当前长度 s 的一半长度。这里面用到了三元运算符和位运算， $s >> 1$ ，意思就是将 s 往右移 1 位，相当于 $s=s/2$ ，只不过位运算是效率最高的运算。
- 3、将新添加的 object 对象作为数组的 a[s]个元素。
- 4、修改集合长度 size 为 s+1
- 5、modCount++，该变量是父类中声明的，用于记录集合修改的次数，记录集合修改的次数是为了防止在用迭代器迭代集合时避免并发修改异常，或者说用于判断是否出现并发修改异常的。
- 6、return true，这个返回值意义不大，因为一直返回 true，除非报了一个运行时异常。

三、remove 方法

remove 方法有两个重载，我们只研究 remove (int index) 方法。

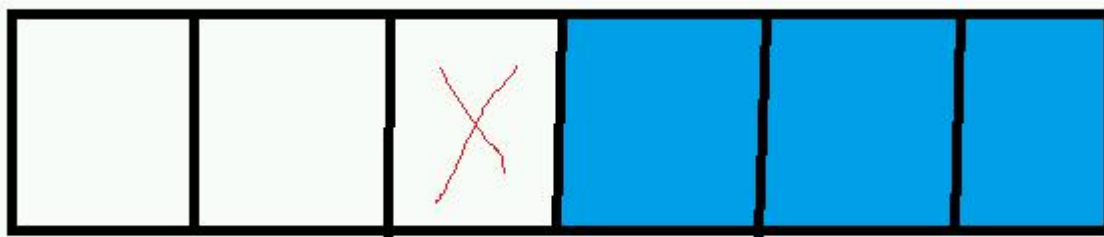
```
/**
 * Removes the object at the specified location from this list.
 *
 * @param index
 *         the index of the object to remove.
 * @return the removed object.
 * @throws IndexOutOfBoundsException
 *         when {@code location < 0 || location >= size()}
 */
@Override public E remove(int index) {
    Object[] a = array;
    int s = size;
    if (index >= s) {
        throwIndexOutOfBoundsException(index, s);
    }
    @SuppressWarnings("unchecked")
```



```
E result = (E) a[index];
System.arraycopy(a, index + 1, a, index, --s - index);
a[s] = null; // Prevent memory leak
size = s;
modCount++;
return result;
}
```

- 1、先将成员变量 array 和 size 赋值给局部变量 a 和 s。
- 2、判断形参 index 是否大于等于集合的长度，如果成了则抛出运行时异常
- 3、获取数组中脚标为 index 的对象 result，该对象作为方法的返回值
- 4、调用 System 的 arraycopy 函数，拷贝原理如下图所示。

```
System.arraycopy(a, index + 1, a, index, --s - index);
```



假设index=2，那么要删除的对象如图打叉部分，我们只需要将该数组后面蓝色区域整体往前移动一位位置即可。上面的代码完成就是这项工作。

5、接下来就是很重要的一个工作，因为删除了一个元素，而且集合整体向前移动了一位，因此需要将集合最后一个元素设置为 null，否则就可能内存泄露。

- 6、重新给成员变量 array 和 size 赋值
- 7、记录修改次数
- 8、返回删除的元素（让用户再看最后一眼）

四、clear 方法

```
/**
 * Removes all elements from this {@code ArrayList}, leaving it empty.
 *
 * @see #isEmpty
 * @see #size
 */
```

```
*/  
@Override public void clear() {  
    if (size != 0) {  
        Arrays.fill(array, 0, size, null);  
        size = 0;  
        modCount++;  
    }  
}
```

如果集合长度不等于 0，则将所有数组的值都设置为 null，然后将成员变量 size 设置为 0 即可，最后让修改记录加 1。

4、并发集合和普通集合如何区别？（2015-11-24）

并发集合常见的有 ConcurrentHashMap、ConcurrentLinkedQueue、ConcurrentLinkedDeque 等。并发集合位于 java.util.concurrent 包下，是 jdk1.5 之后才有的，主要作者是 Doug Lea (<http://baike.baidu.com/view/3141057.htm>) 完成的。

在 java 中有普通集合、同步（线程安全）的集合、并发集合。普通集合通常性能最高，但是不保证多线程的安全性和并发的可靠性。线程安全集合仅仅是给集合添加了 synchronized 同步锁，严重牺牲了性能，而且对并发的效率就更低了，并发集合则通过复杂的策略不仅保证了多线程的安全又提高的并发时的效率。

参考阅读：

ConcurrentHashMap 是线程安全的 HashMap 的实现，默认构造同样有 initialCapacity 和 loadFactor 属性，不过还多了一个 concurrencyLevel 属性，三属性默认值分别为 16、0.75 及 16。其内部使用锁分段技术，维持这锁 Segment 的数组，在 Segment 数组中又存放着 Entry[] 数组，内部 hash 算法将数据较均匀分布在不同锁中。

put 操作：并没有在此方法上加上 synchronized，首先对 key.hashCode 进行 hash 操作，得到 key 的 hash 值。hash 操作的算法和 map 也不同，根据此 hash 值计算并获取其对应的数组中的 Segment 对象（继承自 ReentrantLock），接着调用此 Segment 对象的 put 方法来完成当前操作。

ConcurrentHashMap 基于 concurrencyLevel 划分出了多个 Segment 来对 key-value 进行存储，从而避免每次

put 操作都得锁住整个数组。在默认的情况下，最佳情况下可允许 16 个线程并发无阻塞的操作集合对象，尽可能地减少并发时的阻塞现象。

get(key)

首先对 key.hashCode 进行 hash 操作，基于其值找到对应的 Segment 对象，调用其 get 方法完成当前操作。而 Segment 的 get 操作首先通过 hash 值和对象数组大小减 1 的值进行按位与操作来获取数组上对应位置的 HashEntry。在这个步骤中，可能会因为对象数组大小的改变，以及数组上对应位置的 HashEntry 产生不一致性，那么 ConcurrentHashMap 是如何保证的？

对象数组大小的改变只有在 put 操作时有可能发生，由于 HashEntry 对象数组对应的变量是 volatile 类型的，因此可以保证如 HashEntry 对象数组大小发生改变，读操作可看到最新的对象数组大小。

在获取到了 HashEntry 对象后，怎么能保证它及其 next 属性构成的链表上的对象不会改变呢？这点 ConcurrentHashMap 采用了一个简单的方式，即 HashEntry 对象中的 hash、key、next 属性都是 final 的，这也就意味着没办法插入一个 HashEntry 对象到基于 next 属性构成的链表中间或末尾。这样就可以保证当获取到 HashEntry 对象后，其基于 next 属性构建的链表是不会发生变化的。

ConcurrentHashMap 默认情况下采用将数据分为 16 个段进行存储，并且 16 个段分别持有各自不同的锁 Segment，锁仅用于 put 和 remove 等改变集合对象的操作，基于 volatile 及 HashEntry 链表的不变性实现了读取的不加锁。这些方式使得 ConcurrentHashMap 能够保持极好的并发支持，尤其是对于读远比插入和删除频繁的 Map 而言，而它采用的这些方法也可谓是对于 Java 内存模型、并发机制深刻掌握的体现。

推荐博客地址：<http://m.oschina.net/blog/269037>

5、List 的三个子类的特点 (2017-2-23)

ArrayList 底层结构是数组,底层查询快,增删慢

LinkedList 底层结构是链表型的,增删快,查询慢

voclor 底层结构是数组 线程安全的,增删慢,查询慢

6、List 和 map 的区别 (2017-2-23)

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合，List 中存储的数据是有顺序，并且允许重复；Map 中存储的数据是没有顺序的，其键是不能重复的，它的值是可以有重复的。

7、HashMap 和 HashTable 有什么区别? (2017-2-23)

HashMap 是线程不安全的,HashMap 是一个接口,是 Map 的一个子接口,是将键映射到值得对象,不允许键值重复,允许空键和空值;由于非线程安全,HashMap 的效率要较 HashTable 的效率高一一些.

HashTable 是线程安全的一个集合,不允许 null 值作为一个 key 值或者 Value 值;

HashTable 是 synchronize,多个线程访问时不需要自己为它的方法实现同步,而 HashMap 在被多个线程访问的时候需要自己为它的方法实现同步;

8、数组和链表分别比较适合用于什么场景，为什么？(2017-2-23)

1、数组和链表简介

在计算机中要对给定的数据集进行若干处理，首要任务是把数据集的一部分（当数据量非常大时，可能只能一部分一部分地读取数据到内存中来处理）或全部存储到内存中，然后再对内存中的数据进行各种处理。

例如，对于数据集 $S\{1, 2, 3, 4, 5, 6\}$ ，要求 S 中元素的和，首先要把数据存储到内存中，然后再将内存中的数据相加。

当内存空间中有足够大的连续空间时，可以把数据连续的存放在内存中，各种编程语言中的数组一般都是按这种方式存储的（也可能有例外），如图 1（b）；当内存中只有一些离散的可用空间时，想连续存储数据就非常困难了，这时能想到的一种解决方式是移动内存中的数据，把离散的空间聚集成连续的一块大空间，如图 1（c）所示，这样做

当然也可以，但是这种情况因为可能要移动别人的数据，所以会存在一些困难，移动的过程中也有可能会把一些别人的重要数据给丢失。另外一种，不影响别人的数据存储方式是把数据集中的数据分开离散地存储到这些不连续空间中，如图(d)。这时为了能把数据集中的所有数据联系起来，需要在前一块数据的存储空间中记录下一块数据的地址，这样只要知道第一块内存空间的地址就能环环相扣地把数据集整体联系在一起了。C/C++中用指针实现的链表就是这种存储形式。

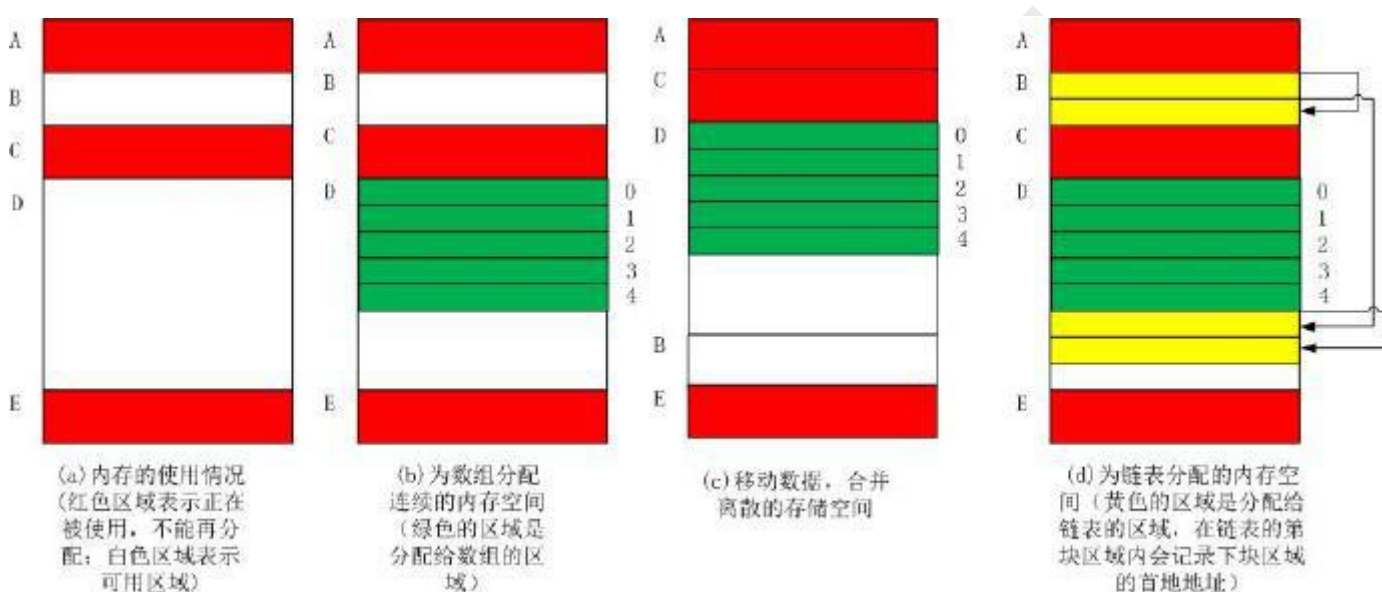


图 内存分配

由上可知，内存中的存储形式可以分为连续存储和离散存储两种。因此，数据的物理存储结构就有连续存储和离散存储两种，它们对应了我们通常所说的数组和链表，

2. 数组和链表的区别

数组是将元素在内存中连续存储的；它的优点：因为数据是连续存储的，内存地址连续，所以在查找数据的时候效率比较高；它的缺点：在存储之前，我们需要申请一块连续的内存空间，并且在编译的时候就必须确定好它的空间的大小。在运行的时候空间的大小是无法随着你的需要进行增加和减少而改变的，当数据两比较大的时候，有可能会出现越界的情况，数据比较小的时候，又有可能会浪费掉内存空间。在改变数据个数时，增加、插入、删除数据效率比较低

链表是动态申请内存空间，不需要像数组需要提前申请好内存的大小，链表只需在用的时候申请就可以，根据需要来动态申请或者删除内存空间，对于数据增加和删除以及插入比数组灵活。还有就是链表中数据在内存中可以在任意的位置，通过应用来关联数据（就是通过存在元素的指针来联系）

3. 链表和数组使用场景

数组应用场景：数据比较少；经常做的运算是按序号访问数据元素；数组更容易实现，任何高级语言都支持；构建的线性表较稳定。

链表应用场景：对线性表的长度或者规模难以估计；频繁做插入删除操作；构建动态性比较强的线性表。

参考网址：

<http://blog.jobbole.com/94423/>

<http://blog.csdn.net/u011277123/article/details/53908387>

9、Java 中 ArrayList 和 LinkedList 区别？（2017-2-23）

ArrayList 和 Vector 使用了数组的实现，可以认为 ArrayList 或者 Vector 封装了对内部数组的操作，比如向数组中添加，删除，插入新的元素或者数据的扩展和重定向。

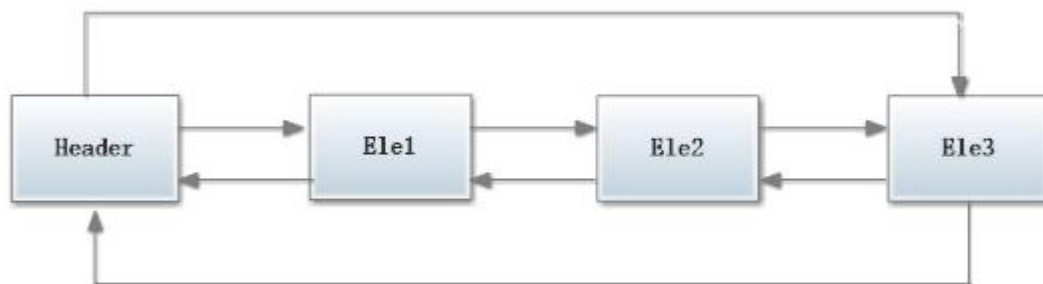
LinkedList 使用了循环双向链表数据结构。与基于数组的 ArrayList 相比，这是两种截然不同的实现技术，这也决定了它们将适用于完全不同的工作场景。

LinkedList 链表由一系列表项连接而成。一个表项总是包含 3 个部分：元素内容，前驱表和后驱表，如图所示：



在下图展示了一个包含 3 个元素的 LinkedList 的各个表项间的连接关系。在 JDK 的实现中，无论 LikedList 是否为空，链表内部都有一个 header 表项，它既表示链表的开始，也表示链表的结尾。表项 header 的后驱表项便是链表

中第一个元素，表项 header 的前驱表项便是链表中最后一个元素。



10、List a=new ArrayList()和 ArrayList a =new ArrayList()的区别？（2017-2-24）

List list = new ArrayList();这句创建了一个 ArrayList 的对象后把上溯到了 List。此时它是一个 List 对象了，有些 ArrayList 有但是 List 没有的属性和方法，它就不能再用了。而 ArrayList list=new ArrayList();创建一对象则保留了 ArrayList 的所有属性。所以需要用到 ArrayList 独有的方法的时候不能用前者。实例代码如下：

```
1.List list = new ArrayList();
2.ArrayList arrayList = new ArrayList();
3.list.trimToSize(); //错误，没有该方法。
4.arrayList.trimToSize(); //ArrayList 里有该方法。
```

11、要对集合更新操作时，ArrayList 和 LinkedList 哪个更适合？(2017-2-24)

- 1.ArrayList 是实现了基于动态数组的数据结构，LinkedList 基于链表的数据结构。
- 2.如果集合数据是对于集合随机访问 get 和 set，ArrayList 绝对优于 LinkedList，因为 LinkedList 要移动指针。
- 3.如果集合数据是对于集合新增和删除操作 add 和 remove，LinkedList 比较占优势，因为 ArrayList 要移动数据。

ArrayList 和 LinkedList 是两个集合类，用于存储一系列的对象引用(references)。例如我们可以用 ArrayList 来存储一系列的 String 或者 Integer。那么 ArrayList 和 LinkedList 在性能上有什么差别呢？什么时候应该用 ArrayList 什么时候又该用 LinkedList 呢？

一. 时间复杂度

首先一点关键的是，ArrayList 的内部实现是基于基础的对象数组的，因此，它使用 get 方法访问列表中的任意一个元素时(random access)，它的速度要比 LinkedList 快。LinkedList 中的 get 方法是按照顺序从列表的一端开始检查，直到另外一端。对 LinkedList 而言，访问列表中的某个指定元素没有更快的方法了。

假设我们有一个很大的列表，它里面的元素已经排好序了，这个列表可能是 ArrayList 类型的也可能是 LinkedList 类型的，现在我们对这个列表来进行二分查找(binary search)，比较列表是 ArrayList 和 LinkedList 时的查询速度，看下面的程序：

```
1. public class TestList{
2.     public static final int N=50000;    //50000 个数
3.     public static List values;          //要查找的集合
4.     //放入 50000 个数给 value;
5.     static{
6.         Integer vals[]=new Integer[N];
7.         Random r=new Random();
8.         for(int i=0,currval=0;i<N;i++){
9.             vals=new Integer(currval);
10.            currval+=r.nextInt(100)+1;
11.        }
12.        values=Arrays.asList(vals);
13.    }
14.    //通过二分查找法查找
15.    static long timeList(List lst){
16.        long start=System.currentTimeMillis();
17.        for(int i=0;i<N;i++){
18.            int index=Collections.binarySearch(lst, values.get(i));
19.            if(index!=i)
20.                System.out.println("***错误***");
21.        }
22.        return System.currentTimeMillis()-start;
23.    }
24.    public static void main(String args[]){
25.        System.out.println("ArrayList 消耗时间："+timeList(new ArrayList(values)));
26.        System.out.println("LinkedList 消耗时间："+timeList(new LinkedList(values)));
27.    }
28. }
```

得到的输出是：

1. ArrayList 消耗时间：15
2. LinkedList 消耗时间：2596

这个结果不是固定的，但是基本上 ArrayList 的时间要明显小于 LinkedList 的时间。因此在这种情况下不宜用 LinkedList。二分查找法使用的随机访问(random access)策略，而 LinkedList 是不支持快速的随机访问的。对一个 LinkedList 做随机访问所消耗的时间与这个 list 的大小是成比例的。而相应的，在 ArrayList 中进行随机访问所消耗的时间是固定的。

这是否表明 ArrayList 总是比 LinkedList 性能要好呢？这并不一定，在某些情况下 LinkedList 的表现要优于 ArrayList，有些算法在 LinkedList 中实现时效率更高。比方说，利用 Collections.reverse 方法对列表进行反转时，其性能就要好些。看这样一个例子，加入我们有一个列表，要对其进行大量的插入和删除操作，在这种情况下 LinkedList 就是一个较好的选择。请看如下一个极端的例子，我们重复的在一个列表的开端插入一个元素：

```
1. import java.util.*;
2. public class ListDemo {
3.     static final int N=50000;
4.     static long timeList(List list){
5.         long start=System.currentTimeMillis();
6.         Object o = new Object();
7.         for(int i=0;i<N;i++)
8.             list.add(0, o);
9.         return System.currentTimeMillis()-start;
10.    }
11.    public static void main(String[] args) {
12.        System.out.println("ArrayList 耗时："+timeList(new ArrayList()));
13.        System.out.println("LinkedList 耗时："+timeList(new LinkedList()));
14.    }
15. }
```

这时我的输出结果是

1. ArrayList 耗时：2463
2. LinkedList 耗时：15

二．空间复杂度

在 LinkedList 中有一个私有的内部类，定义如下：

```
1.private static class Entry {  
2.    Object element;  
3.    Entry next;  
4.    Entry previous;  
5. }
```

每个 Entry 对象 reference 列表 中的一个元素，同时还有在 LinkedList 中它的上一个元素和下一个元素。一个有 1000 个元素的 LinkedList 对象将有 1000 个链接在一起的 Entry 对象，每个对象都对应于列表中的一个元素。这样的话，在一个 LinkedList 结构中将有有一个很大的空间开销，因为它要存储这 1000 个 Entity 对象的相关信息。

ArrayList 使用一个内置的数组来存储元素，这个数组的起始容量是 10。当数组需要增长时，新的容量按如下公式获得：新容量=(旧容量*3)/2+1，也就是说每一次容量大概会增长 50%。这就意味着，如果你有一个包含大量元素的 ArrayList 对象，那么最终将有很大的空间会被浪费掉，这个浪费是由 ArrayList 的工作方式本身造成的。如果没有足够的空间来存放新的元素，数组将不得不被重新进行分配以便能够增加新的元素。对数组进行重新分配，将会导致性能急剧下降。如果我们知道一个 ArrayList 将会有多少个元素，我们可以通过构造方法来指定容量。我们还可以通过 trimToSize 方法在 ArrayList 分配完毕之后去掉浪费掉的空间。

三．总结

ArrayList 和 LinkedList 在性能上各有优缺点，都有各自所适用的地方，总的说来可以描述如下：

1．对 ArrayList 和 LinkedList 而言，在列表末尾增加一个元素所花的开销都是固定的。对 ArrayList 而言，主要是在内部数组中增加一项，指向所添加的元素，偶尔可能会导致对数组重新进行分配；而对 LinkedList 而言，这个开销是统一的，分配一个内部 Entry 对象。

2．在 ArrayList 的中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动；而在 LinkedList 的中间插入或删除一个元素的开销是固定的。

3．LinkedList 不支持高效的随机元素访问。

4．ArrayList 的空间浪费主要体现在在 list 列表的结尾预留一定的容量空间，而 LinkedList 的空间花费则体现在它的每一个元素都需要消耗相当的空间

可以这样说：当操作是在一系列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用

ArrayList 会提供比较好的性能 ;当你的操作是在一系列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用 LinkedList 了。

12、请用两个队列模拟堆栈结构 (2017-2-24)

两个队列模拟一个堆栈，队列是先进先出，而堆栈是先进后出。模拟如下

队列 a 和 b

(1) 入栈：a 队列为空，b 为空。例：则将“ a,b,c,d,e” 需要入栈的元素先放 a 中，a 进栈为“ a,b,c,d,e”

(2) 出栈：a 队列目前的元素为“ a,b,c,d,e” 。将 a 队列依次加入 ArrayList 集合 a 中。以倒序的方法，将 a 中的集合取出，放入 b 队列中，再将 b 队列出列。代码如下：

```
1. public static void main(String[] args) {
2.     Queue<String> queue = new LinkedList<String>(); //a 队列
3.     Queue<String> queue2=new LinkedList<String>(); //b 队列
4.     ArrayList<String> a=new ArrayList<String>(); //arraylist 集合是中间参数
5.     //往 a 队列添加元素
6.     queue.offer("a");
7.     queue.offer("b");
8.     queue.offer("c");
9.     queue.offer("d");
10.    queue.offer("e");
11.    System.out.print("进栈：");
12.    //a 队列依次加入 list 集合之中
13.    for(String q : queue){
14.        a.add(q);
15.        System.out.print(q);
16.    }
17.    //以倒序的方法取出 ( a 队列依次加入 list 集合 ) 之中的值，加入 b 对列
18.    for(int i=a.size()-1;i>=0;i--){
19.        queue2.offer(a.get(i));
20.    }
21.    //打印出栈队列
22.    System.out.println("");
23.    System.out.print("出栈：");
24.    for(String q : queue2){
25.        System.out.print(q);
```

```
26.     }  
27. }
```

打印结果为（遵循栈模式先进后出）：

```
进栈：a b c d e  
出栈：e d c b a
```

七、Java 的多线程

1、多线程的两种创建方式

`java.lang.Thread` 类的实例就是一个线程但是它需要调用 `java.lang.Runnable` 接口来执行，由于线程类本身就是调用的 `Runnable` 接口所以你可以继承 `java.lang.Thread` 类或者直接实现 `Runnable` 接口来重写 `run()` 方法实现线程。

2、在 java 中 wait 和 sleep 方法的不同？

最大的不同是在等待时 `wait` 会释放锁，而 `sleep` 一直持有锁。`wait` 通常被用于线程间交互，`sleep` 通常被用于暂停执行。

3、synchronized 和 volatile 关键字的作用

一旦一个共享变量（类的成员变量、类的静态成员变量）被 `volatile` 修饰之后，那么就具备了两层语义：

1) 保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。

2) 禁止进行指令重排序。

`volatile` 本质是在告诉 jvm 当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；

`synchronized` 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。

1.volatile 仅能使用在变量级别；

synchronized 则可以使用在变量、方法、和类级别的

2.volatile 仅能实现变量的修改可见性，并不能保证原子性；

synchronized 则可以保证变量的修改可见性和原子性

3.volatile 不会造成线程的阻塞；

synchronized 可能会造成线程的阻塞。

4.volatile 标记的变量不会被编译器优化；

synchronized 标记的变量可以被编译器优化

4、分析线程并发访问代码解释原因

```
public class Counter {
    private volatile int count = 0;
    public void inc(){
        try {
            Thread.sleep(3);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        count++;
    }
    @Override
    public String toString() {
        return "[count=" + count + "]";
    }
}

//-----华丽的分割线-----
public class VolatileTest {
    public static void main(String[] args) {
        final Counter counter = new Counter();
        for(int i=0;i<1000;i++){
            new Thread(new Runnable() {
                @Override
                public void run() {
                    counter.inc();
                }
            }).start();
        }
        System.out.println(counter);
    }
}
```

上面的代码执行完后输出的结果确定为 1000 吗？

答案是不一定，或者不等于 1000。这是为什么吗？

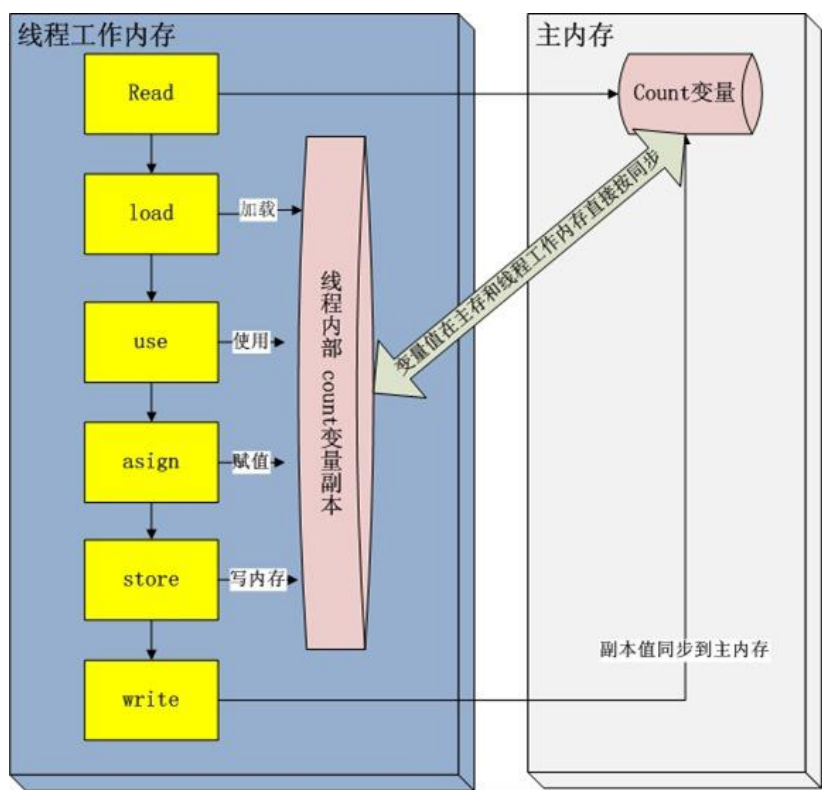
在 java 的内存模型中每一个线程运行时都有一个线程栈，线程栈保存了线程运行时候变量值信息。当线程访问某一个对象时候值的时候，首先通过对象的引用找到对应堆内存的变量的值，然后把堆内存变量的具体值 load 到线程本地内存中，建立一个变量副本，之后线程就不再和对象在堆内存变量值有任何关系，而是直接修改副本变量的值，

在修改完之后的某一个时刻（线程退出之前），自动把线程变量副本的值回写到对象在堆中变量。这样在堆中的对象的值就产生变化了。

也就是说上面主函数中开启了 1000 个子线程，每个线程都有一个变量副本，每个线程修改变量只是临时修改了自己的副本，当线程结束时再将修改的值写入在主内存中，这样就出现了线程安全问题。因此结果就不可能等于 1000 了，一般都会小于 1000。

上面的解释用一张图表示如下：

（图片来自网络，非本人所绘）



5、什么是线程池，如何使用？

线程池就是事先将多个线程对象放到一个容器中，当使用的时候就不用 new 线程而是直接去池中拿线程即可，节省了开辟子线程的时间，提高的代码执行效率。

在 JDK 的 `java.util.concurrent.Executors` 中提供了生成多种线程池的静态方法。

```
ExecutorService newCachedThreadPool = Executors.newCachedThreadPool();
ExecutorService newFixedThreadPool = Executors.newFixedThreadPool(4);
ScheduledExecutorService newScheduledThreadPool =
    Executors.newScheduledThreadPool(4);
ExecutorService newSingleThreadExecutor = Executors.newSingleThreadExecutor();
```

然后调用他们的 `execute` 方法即可。

6、请叙述一下您对线程池的理解？（2015-11-25）

（如果问到了这样的问题，可以展开的说一下线程池如何用、线程池的好处、线程池的启动策略）

合理利用线程池能够带来三个好处。

第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。

第三：提高线程的可管理性。线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，调优和监控。

7、线程池的启动策略？（2015-11-25）

官方对线程池的执行过程描述如下：

```
1.  /*
2.      * Proceed in 3 steps:
3.      *
4.      * 1. If fewer than corePoolSize threads are running, try to
5.      * start a new thread with the given command as its first
6.      * task. The call to addWorker atomically checks runState and
7.      * workerCount, and so prevents false alarms that would add
8.      * threads when it shouldn't, by returning false.
9.      *
10.     * 2. If a task can be successfully queued, then we still need
11.     * to double-check whether we should have added a thread
12.     * (because existing ones died since last checking) or that
13.     * the pool shut down since entry into this method. So we
14.     * recheck state and if necessary roll back the enqueueing if
```



```
15.      * stopped, or start a new thread if there are none.
16.      *
17.      * 3. If we cannot queue task, then we try to add a new
18.      * thread. If it fails, we know we are shut down or saturated
19.      * and so reject the task.
20.      */
```

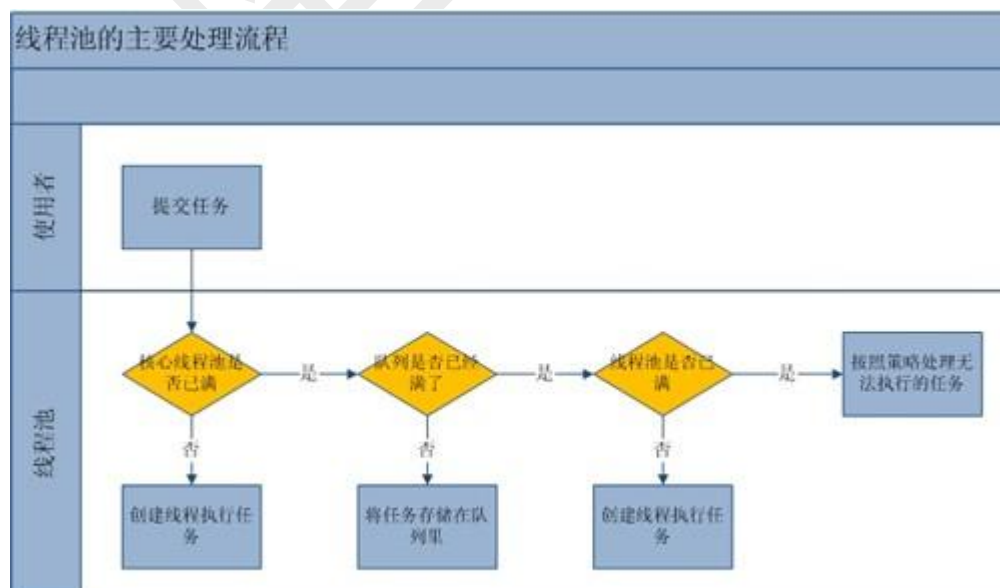
1、线程池刚创建时，里面没有一个线程。任务队列是作为参数传进来的。不过，就算队列里面有任务，线程池也不会马上执行它们。

2、当调用 `execute()` 方法添加一个任务时，线程池会做如下判断：

- 如果正在运行的线程数量小于 `corePoolSize`，那么马上创建线程运行这个任务；
- 如果正在运行的线程数量大于或等于 `corePoolSize`，那么将这个任务放入队列。
- 如果这时候队列满了，而且正在运行的线程数量小于 `maximumPoolSize`，那么还是要创建线程运行这个任务；
- 如果队列满了，而且正在运行的线程数量大于或等于 `maximumPoolSize`，那么线程池会抛出异常，告诉调用者“我不能再接受任务了”。

3、当一个线程完成任务时，它会从队列中取下一个任务来执行。

4、当一个线程无事可做，超过一定的时间（`keepAliveTime`）时，线程池会判断，如果当前运行的线程数大于 `corePoolSize`，那么这个线程就被停掉。所以线程池的所有任务完成后，它最终会收缩到 `corePoolSize` 的大小。



8、如何控制某个方法允许并发访问线程的个数？（2015-11-30）

```
1. package com.yange;
2.
3. import java.util.concurrent.Semaphore;
4. /**
5.  *
6.  * @author wzy 2015-11-30
7.  *
8.  */
9. public class SemaphoreTest {
10. /*
11.  * permits the initial number of permits available. This value may be negative,
12.  * in which case releases must occur before any acquires will be granted.
13.  * fair true if this semaphore will guarantee first-in first-out granting of
14.  * permits under contention, else false
15.  */
16. static Semaphore semaphore = new Semaphore(5,true);
17. public static void main(String[] args) {
18.     for(int i=0;i<100;i++){
19.         new Thread(new Runnable() {
20.
21.             @Override
22.             public void run() {
23.                 test();
24.             }
25.         }).start();
26.     }
27.
28. }
29.
30. public static void test(){
31.     try {
32.         //申请一个请求
33.         semaphore.acquire();
34.     } catch (InterruptedException e1) {
35.         e1.printStackTrace();
36.     }
37.     System.out.println(Thread.currentThread().getName()+"进来了");
38.     try {
39.         Thread.sleep(1000);
40.     } catch (InterruptedException e) {
41.         e.printStackTrace();
42.     }
```

```
43.     System.out.println(Thread.currentThread().getName()+"走了");
44.     //释放一个请求
45.     semaphore.release();
46. }
47. }
```

可以使用 Semaphore 控制，第 16 行的构造函数创建了一个 Semaphore 对象，并且初始化了 5 个信号。这样的效果是控件 test 方法最多只能有 5 个线程并发访问，对于 5 个线程时就排队等待，走一个来一下。第 33 行，请求一个信号（消费一个信号），如果信号被用完了则等待，第 45 行释放一个信号，释放的信号新的线程就可以使用了。

9、三个线程 a、b、c 并发运行，b,c 需要 a 线程的数据怎么实现（上海 3 期学员提供）

根据问题的描述，我将问题用以下代码演示，ThreadA、ThreadB、ThreadC，ThreadA 用于初始化数据 num，只有当 num 初始化完成之后再让 ThreadB 和 ThreadC 获取到初始化后的变量 num。

分析过程如下：

考虑到多线程的不确定性，因此我们不能确保 ThreadA 就一定先于 ThreadB 和 ThreadC 前执行，就算 ThreadA 先执行了，我们也无法保证 ThreadA 什么时候才能将变量 num 给初始化完成。因此我们必须让 ThreadB 和 ThreadC 去等待 ThreadA 完成任何后发出的消息。

现在需要解决两个难题，一是让 ThreadB 和 ThreadC 等待 ThreadA 先执行完，二是 ThreadA 执行完之后给 ThreadB 和 ThreadC 发送消息。

解决上面的难题我能想到的两种方案，一是使用纯 Java API 的 Semaphore 类来控制线程的等待和释放，二是使用 Android 提供的 Handler 消息机制。

```
1. package com.example;
2. /**
3.  * 三个线程 a、b、c 并发运行，b,c 需要 a 线程的数据怎么实现（上海 3 期学员提供）
4.  *
5.  */
6. public class ThreadCommunication {
7.     private static int num;//定义一个变量作为数据
8.
9.     public static void main(String[] args) {
10.
```

```
11.     Thread threadA = new Thread(new Runnable() {
12.
13.         @Override
14.         public void run() {
15.             try {
16.                 //模拟耗时操作之后初始化变量 num
17.                 Thread.sleep(1000);
18.                 num = 1;
19.
20.             } catch (InterruptedException e) {
21.                 e.printStackTrace();
22.             }
23.         }
24.     });
25.     Thread threadB = new Thread(new Runnable() {
26.
27.         @Override
28.         public void run() {
29.             System.out.println(Thread.currentThread().getName()+"获取到 num 的值为："+num);
30.         }
31.     });
32.     Thread threadC = new Thread(new Runnable() {
33.
34.         @Override
35.         public void run() {
36.             System.out.println(Thread.currentThread().getName()+"获取到 num 的值为："+num);
37.         }
38.     });
39.     //同时开启 3 个线程
40.     threadA.start();
41.     threadB.start();
42.     threadC.start();
43.
44. }
45. }
46.
```

解决方案一：

```
1. public class ThreadCommunication {
2.     private static int num;
3.     /**
4.      * 定义一个信号量，该类内部维持了多个线程锁，可以阻塞多个线程，释放多个线程，
5.      * 线程的阻塞和释放是通过 permit 概念来实现的
6.      * 线程通过 semaphore.acquire() 方法获取 permit，如果当前 semaphore 有 permit 则分配给该线程，
```

```
7. 如果没有则阻塞该线程直到 semaphore
8.  * 调用 release ( ) 方法释放 permit。
9.  * 构造函数中参数：permit ( 允许 ) 个数，
10. */
11. private static Semaphore semaphore = new Semaphore(0);
12. public static void main(String[] args) {
13.
14.     Thread threadA = new Thread(new Runnable() {
15.
16.         @Override
17.         public void run() {
18.             try {
19.                 //模拟耗时操作之后初始化变量 num
20.                 Thread.sleep(1000);
21.                 num = 1;
22.                 //初始化完参数后释放两个 permit
23.                 semaphore.release(2);
24.
25.             } catch (InterruptedException e) {
26.                 e.printStackTrace();
27.             }
28.         }
29.     });
30.     Thread threadB = new Thread(new Runnable() {
31.
32.         @Override
33.         public void run() {
34.             try {
35.                 //获取 permit，如果 semaphore 没有可用的 permit 则等待，如果有则消耗一个
36.                 semaphore.acquire();
37.             } catch (InterruptedException e) {
38.                 e.printStackTrace();
39.             }
40.             System.out.println(Thread.currentThread().getName()+"获取到 num 的值为："+num);
41.         }
42.     });
43.     Thread threadC = new Thread(new Runnable() {
44.
45.         @Override
46.         public void run() {
47.             try {
48.                 //获取 permit，如果 semaphore 没有可用的 permit 则等待，如果有则消耗一个
49.                 semaphore.acquire();
50.             } catch (InterruptedException e) {
```

```
51.         e.printStackTrace();
52.     }
53.     System.out.println(Thread.currentThread().getName()+"获取到 num 的值为："+num);
54. }
55. });
56. //同时开启 3 个线程
57. threadA.start();
58. threadB.start();
59. threadC.start();
60.
61. }
62. }
```

10、同一个类中的 2 个方法都加了同步锁，多个线程能同时访问同一个类中的这两个方法吗？（2017-2-24）

这个问题需要考虑到 Lock 与 synchronized 两种实现锁的不同情形。因为这种情况下使用 Lock 和 synchronized 会有截然不同的结果。Lock 可以让等待锁的线程响应中断，Lock 获取锁，之后需要释放锁。如下代码，多个线程不可访问同一个类中的 2 个加了 Lock 锁的方法。

```
63. package com;
64. import java.util.concurrent.locks.Lock;
65. import java.util.concurrent.locks.ReentrantLock;
66. public class qq {
67.
68.     private int count = 0;
69.     private Lock lock = new ReentrantLock();//设置 lock 锁
70.     //方法 1
71.     public Runnable run1 = new Runnable(){
72.         public void run() {
73.             lock.lock(); //加锁
74.             while(count < 1000) {
75.                 try {
76.                     //打印是否执行该方法
77.                     System.out.println(Thread.currentThread().getName() + " run1: "+count++);
78.                 } catch (Exception e) {
79.                     e.printStackTrace();
80.                 }
81.             }
82.         }
83.         lock.unlock();
84.     }
85. }
```

```
84.     }  
85.     //方法2  
86.     public Runnable run2 = new Runnable(){  
87.         public void run() {  
88.             lock.lock();  
89.             while(count < 1000) {  
90.                 try {  
91.                     System.out.println(Thread.currentThread().getName() +  
92.                         " run2: "+count++);  
93.                 } catch (Exception e) {  
94.                     e.printStackTrace();  
95.                 }  
96.             }  
97.             lock.unlock();  
98.         }  
99.  
100.  
101.  
102.     public static void main(String[] args) throws InterruptedException {  
103.         qq t = new qq();    //创建一个对象  
104.         new Thread(t.run1).start(); //获取该对象的方法1  
105.  
106.         new Thread(t.run2).start(); //获取该对象的方法2  
107.     }  
108. }
```

结果是：

```
Thread-0 run1: 0  
Thread-0 run1: 1  
Thread-0 run1: 2  
Thread-0 run1: 3  
Thread-0 run1: 4  
Thread-0 run1: 5  
Thread-0 run1: 6  
.....
```

而 synchronized 却不行，使用 synchronized 时，当我们访问同一个类对象的时候，是同一把锁，所以可以访问

该对象的其他 synchronized 方法。代码如下：

```
1. package com;  
2. import java.util.concurrent.locks.Lock;  
3. import java.util.concurrent.locks.ReentrantLock;  
4. public class qq {  
5.     private int count = 0;  
6.     private Lock lock = new ReentrantLock();
```

```
7.     public Runnable run1 = new Runnable(){
8.         public void run() {
9.             synchronized(this) { //设置关键字 synchronized, 以当前类为锁
10.                while(count < 1000) {
11.                    try {
12.                        //打印是否执行该方法
13.                        System.out.println(Thread.currentThread().getName() + " run1: "+count++);
14.                    } catch (Exception e) {
15.                        e.printStackTrace();
16.                    }
17.                }
18.            }
19.        }
20.     public Runnable run2 = new Runnable(){
21.         public void run() {
22.             synchronized(this) {
23.                while(count < 1000) {
24.                    try {
25.                        System.out.println(Thread.currentThread().getName()
26.                            + " run2: "+count++);
27.                    } catch (Exception e) {
28.                        e.printStackTrace();
29.                    }
30.                }
31.            }
32.        }
33.     public static void main(String[] args) throws InterruptedException {
34.         qq t = new qq(); //创建一个对象
35.         new Thread(t.run1).start(); //获取该方法的方法 1
36.         new Thread(t.run2).start(); //获取该方法的方法 2
37.     }
38. }
```

结果为：

```
Thread-1 run2: 0
Thread-1 run2: 1
Thread-1 run2: 2
Thread-0 run1: 0
Thread-0 run1: 4 Thread-0 run1: 5 Thread-0 run1: 6
.....
```

11、什么情况下导致线程死锁，遇到线程死锁该怎么解决？（2017-2-24）

11.1 死锁的定义：所谓死锁是指多个线程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程

都将无法向前推进。

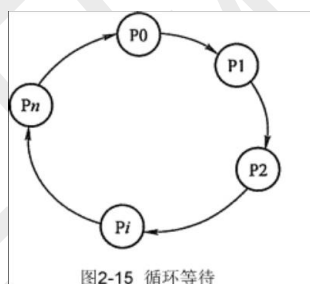
11.2 死锁产生的必要条件：

互斥条件：线程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某资源仅为一个线程所占有。此时若有其他线程请求该资源，则请求线程只能等待。

不剥夺条件：线程所获得的资源在未使用完毕之前，不能被其他线程强行夺走，即只能由获得该资源的线程自己来释放（只能是主动释放）。

请求和保持条件：线程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他线程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

循环等待条件：存在一种线程资源的循环等待链，链中每一个线程已获得的资源同时被链中下一个线程所请求。即存在一个处于等待状态的线程集合 $\{P_1, P_2, \dots, P_n\}$ ，其中 P_i 等待的资源被 P_{i+1} 占有（ $i=0, 1, \dots, n-1$ ）， P_n 等待的资源被 P_0 占有，如图 2-15 所示。



11.3 产生死锁的一个例子

```
1.package itheima.com;
2. /**
3. * 一个简单的死锁类
4. * 当 DeadLock 类的对象 flag==1 时 (td1)，先锁定 o1, 睡眠 500 毫秒
5. * 而 td1 在睡眠的时候另一个 flag==0 的对象 (td2) 线程启动，先锁定 o2, 睡眠 500 毫秒
6. * td1 睡眠结束后需要锁定 o2 才能继续执行，而此时 o2 已被 td2 锁定；
7. * td2 睡眠结束后需要锁定 o1 才能继续执行，而此时 o1 已被 td1 锁定；
8. * td1、td2 相互等待，都需要得到对方锁定的资源才能继续执行，从而死锁。
9. */
10. public class DeadLock implements Runnable {
11.     public int flag = 1;
```

```
12.    //静态对象是类的所有对象共享的
13.    private static Object o1 = new Object(), o2 = new Object();
14.    public void run() {
15.        System.out.println("flag=" + flag);
16.        if (flag == 1) {
17.            synchronized (o1) {
18.                try {
19.                    Thread.sleep(500);
20.                } catch (Exception e) {
21.                    e.printStackTrace();
22.                }
23.                synchronized (o2) {
24.                    System.out.println("1");
25.                }
26.            }
27.        }
28.        if (flag == 0) {
29.            synchronized (o2) {
30.                try {
31.                    Thread.sleep(500);
32.                } catch (Exception e) {
33.                    e.printStackTrace();
34.                }
35.                synchronized (o1) {
36.                    System.out.println("0");
37.                }
38.            }
39.        }
40.    }
41.    public static void main(String[] args) {
42.        DeadLock td1 = new DeadLock();
43.        DeadLock td2 = new DeadLock();
44.        td1.flag = 1;
45.        td2.flag = 0;
46.        //td1,td2 都处于可执行状态，但 JVM 线程调度先执行哪个线程是不确定的。
47.        //td2 的 run() 可能在 td1 的 run() 之前运行
48.        new Thread(td1).start();
49.        new Thread(td2).start();
50.    }
51. }
```

11.4 如何避免死锁

在有些情况下死锁是可以避免的。两种用于避免死锁的技术：

1) 加锁顺序 (线程按照一定的顺序加锁)

```
1.package itheima.com;
2.public class DeadLock {
3.    public int flag = 1;
4.    //静态对象是类的所有对象共享的
5.    private static Object o1 = new Object(), o2 = new Object();
6.    public void money(int flag) {
7.        this.flag=flag;
8.        if( flag ==1){
9.            synchronized (o1) {
10.                try {
11.                    Thread.sleep(500);
12.                } catch (Exception e) {
13.                    e.printStackTrace();
14.                }
15.                synchronized (o2) {
16.                    System.out.println("当前的线程是"+
17.                        Thread.currentThread().getName()+" "+flag 的值"+"1");
18.                }
19.            }
20.        }
21.        if(flag ==0){
22.            synchronized (o2) {
23.                try {
24.                    Thread.sleep(500);
25.                } catch (Exception e) {
26.                    e.printStackTrace();
27.                }
28.                synchronized (o1) {
29.                    System.out.println("当前的线程是"+
30.                        Thread.currentThread().getName()+" "+flag 的值"+"0");
31.                }
32.            }
33.        }
34.    }
35.
36.    public static void main(String[] args) {
37.        final DeadLock td1 = new DeadLock();
38.        final DeadLock td2 = new DeadLock();
39.        td1.flag = 1;
40.        td2.flag = 0;
41.        //td1,td2 都处于可执行状态，但 JVM 线程调度先执行哪个线程是不确定的。
42.        //td2 的 run() 可能在 td1 的 run() 之前运行
```

```
43.         final Thread t1=new Thread(new Runnable() {
44.             public void run() {
45.                 td1.flag = 1;
46.                 td1.money(1);
47.             }
48.         });
49.         t1.start();
50.         Thread t2= new Thread(new Runnable() {
51.             public void run() {
52.                 // TODO Auto-generated method stub
53.                 try {
54.                     //让 t2 等待 t1 执行完
55.                     t1.join();//核心代码，让 t1 执行完后 t2 才会执行
56.                 } catch (InterruptedException e) {
57.                     // TODO Auto-generated catch block
58.                     e.printStackTrace();
59.                 }
60.                 td2.flag = 0;
61.                 td1.money(0);
62.             }
63.         });
64.         t2.start();
65.     }
66. }
```

结果：

当前的线程是 Thread-0 flag 的值 1

当前的线程是 Thread-1 flag 的值 0

2) 加锁时限 (线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁)

```
1. package itheima.com;
2. import java.util.concurrent.TimeUnit;
3. import java.util.concurrent.locks.Lock;
4. import java.util.concurrent.locks.ReentrantLock;
5. public class DeadLock {
6.     public int flag = 1;
7.     //静态对象是类的所有对象共享的
8.     private static Object o1 = new Object(), o2 = new Object();
9.     public void money(int flag) throws InterruptedException {
10.         this.flag=flag;
11.         if( flag ==1){
12.             synchronized (o1) {
13.                 Thread.sleep(500);
14.                 synchronized (o2) {
```

```
15 .          System.out.println("当前的线程是"+
16 .          Thread.currentThread().getName()+" "+"flag 的值"+"1");
17 .      }
18 .  }
19 . }
20 . if(flag ==0){
21 .     synchronized (o2) {
22 .         Thread.sleep(500);
23 .         synchronized (o1) {
24 .             System.out.println("当前的线程是"+
25 .             Thread.currentThread().getName()+" "+"flag 的值"+"0");
26 .         }
27 .     }
28 . }
29 . }
30 .
31 . public static void main(String[] args) {
32 .     final Lock lock = new ReentrantLock();
33 .     final DeadLock td1 = new DeadLock();
34 .     final DeadLock td2 = new DeadLock();
35 .     td1.flag = 1;
36 .     td2.flag = 0;
37 .     //td1,td2 都处于可执行状态，但 JVM 线程调度先执行哪个线程是不确定的。
38 .     //td2 的 run() 可能在 td1 的 run() 之前运行
39 .
40 .     final Thread t1=new Thread(new Runnable(){
41 .         public void run() {
42 .             // TODO Auto-generated method stub
43 .             String tName = Thread.currentThread().getName();
44 .
45 .             td1.flag = 1;
46 .             try {
47 .                 //获取不到锁，就等 5 秒，如果 5 秒后还是获取不到就返回 false
48 .                 if (lock.tryLock(5000, TimeUnit.MILLISECONDS)) {
49 .                     System.out.println(tName + "获取到锁！");
50 .                 } else {
51 .                     System.out.println(tName + "获取不到锁！");
52 .                     return;
53 .                 }
54 .             } catch (Exception e) {
55 .                 e.printStackTrace();
56 .             }
57 .
58 .             try {
```

```
59 .            td1.money(1);
60 .            } catch (Exception e) {
61 .                System.out.println(tName + "出错了!!!");
62 .            } finally {
63 .                System.out.println("当前的线程是"+Thread.currentThread().getName()+"释放锁!!");
64 .                lock.unlock();
65 .            }
66 .        }
67 .    });
68 .    t1.start();
69 .    Thread t2= new Thread(new Runnable(){
70 .        public void run() {
71 .            String tName = Thread.currentThread().getName();
72 .            // TODO Auto-generated method stub
73 .            td1.flag = 1;
74 .            try {
75 .                //获取不到锁，就等 5 秒，如果 5 秒后还是获取不到就返回 false
76 .                if (lock.tryLock(5000, TimeUnit.MILLISECONDS)) {
77 .                    System.out.println(tName + "获取到锁!");
78 .                } else {
79 .                    System.out.println(tName + "获取不到锁!");
80 .                    return;
81 .                }
82 .            } catch (Exception e) {
83 .                e.printStackTrace();
84 .            }
85 .            try {
86 .                td2.money(0);
87 .            } catch (Exception e) {
88 .                System.out.println(tName + "出错了!!!");
89 .            } finally {
90 .                System.out.println("当前的线程是"+Thread.currentThread().getName()+"释放锁!!");
91 .                lock.unlock();
92 .            }
93 .        }
94 .    });
95 .    t2.start();
96 . }
97 . }
```

打印结果：

```
Thread-0 获取到锁！
当前的线程是 Thread-0 flag 的值 1
当前的线程是 Thread-0 释放锁！！
```

```
Thread-1 获取到锁！  
当前的线程是 Thread-1 flag 的值 0  
当前的线程是 Thread-1 释放锁！！
```

12、Java 中多线程间的通信怎么实现?(2017-2-24)

线程通信的方式：

1.共享变量

线程间通信可以通过发送信号，发送信号的一个简单方式是在共享对象的变量里设置信号值。线程 A 在一个同步块里设置 boolean 型成员变量 `hasDataToProcess` 为 `true`，线程 B 也在同步块里读取 `hasDataToProcess` 这个成员变量。这个简单的例子使用了一个持有信号的对象，并提供了 `set` 和 `get` 方法：

```
1.package itheima.com;  
2.public class MySignal{  
3.    //共享的变量  
4.    private boolean hasDataToProcess=false;  
5.    //取值  
6.    public boolean getHasDataToProcess() {  
7.        return hasDataToProcess;  
8.    }  
9.    //存值  
10.   public void setHasDataToProcess(boolean hasDataToProcess) {  
11.       this.hasDataToProcess = hasDataToProcess;  
12.   }  
13.   public static void main(String[] args){  
14.       //同一个对象  
15.       final MySignal my=new MySignal();  
16.       //线程 1 设置 hasDataToProcess 值为 true  
17.       final Thread t1=new Thread(new Runnable(){  
18.           public void run() {  
19.               my.setHasDataToProcess(true);  
20.           }  
21.       });  
22.       t1.start();  
23.       //线程 2 取这个值 hasDataToProcess  
24.       Thread t2=new Thread(new Runnable(){  
25.           public void run() {  
26.               try {
```

```
27.          //等待线程 1 完成然后取值
28.          t1.join();
29.      } catch (InterruptedException e) {
30.          e.printStackTrace();
31.      }
32.      my.getHasDataToProcess();
33.      System.out.println("t1 改变以后的值：" + my.isHasDataToProcess());
34.  }
35.  });
36.  t2.start();
37. }
38. }
```

结果：

t1 改变以后的值：true

2.wait/notify 机制

以资源为例，生产者生产一个资源，通知消费者就消费掉一个资源，生产者继续生产资源，消费者消费资源，以

此循环。代码如下：

```
1. package itheima.com;
2. //资源类
3. class Resource{
4.     private String name;
5.     private int count=1;
6.     private boolean flag=false;
7.     public synchronized void set(String name){
8.         //生产资源
9.         while(flag) {
10.            try{
11.                //线程等待。消费者消费资源
12.                wait();
13.            }catch(Exception e){}
14.        }
15.        this.name=name+"---"+count++;
16.        System.out.println(Thread.currentThread().getName()+"...生产者..."+this.name);
17.        flag=true;
18.        //唤醒等待中的消费者
19.        this.notifyAll();
20.    }
21.    public synchronized void out(){
22.        //消费资源
23.        while(!flag) {
24.            //线程等待，生产者生产资源
```



```
25 .         try{wait();}catch(Exception e){}
26 .     }
27 .     System.out.println(Thread.currentThread().getName()+"...消费者..."+this.name);
28 .     flag=false;
29 .     //唤醒生产者，生产资源
30 .     this.notifyAll();
31 .     }
32 . }
33 . //生产者
34 . class Producer implements Runnable{
35 .     private Resource res;
36 .     Producer(Resource res){
37 .         this.res=res;
38 .     }
39 .     //生产者生产资源
40 .     public void run(){
41 .         while(true){
42 .             res.set("商品");
43 .         }
44 .     }
45 . }
46 . //消费者消费资源
47 . class Consumer implements Runnable{
48 .     private Resource res;
49 .     Consumer(Resource res){
50 .         this.res=res;
51 .     }
52 .     public void run(){
53 .         while(true){
54 .             res.out();
55 .         }
56 .     }
57 . }
58 . public class ProducerConsumerDemo{
59 .     public static void main(String[] args){
60 .         Resource r=new Resource();
61 .         Producer pro=new Producer(r);
62 .         Consumer con=new Consumer(r);
63 .         Thread t1=new Thread(pro);
64 .         Thread t2=new Thread(con);
65 .         t1.start();
66 .         t2.start();
67 .     }
68 . }
```

3. JavaSE 高级 (★★)

一、Java 中的反射

1、说说你对 Java 中反射的理解

Java 中的反射首先是能够获取到 Java 中要反射类的字节码，获取字节码有三种方法，1.Class.forName(className) 2.类名.class 3.this.getClass()。然后将字节码中的方法，变量，构造函数等映射成相应的 Method、Filed、Constructor 等类，这些类提供了丰富的方法可以被我们所使用。

二、Java 中的动态代理

1、写一个 ArrayList 的动态代理类 (笔试题)

```
final List<String> list = new ArrayList<String>();

List<String> proxyInstance = (List<String>)
Proxy.newProxyInstance(list.getClass().getClassLoader(),
list.getClass().getInterfaces(), new InvocationHandler() {

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        return method.invoke(list, args);
    }
});
proxyInstance.add("你好");
System.out.println(list);
```

2、动静代理的区别，什么场景使用？（2015-11-25）

静态代理通常只代理一个类，动态代理是代理一个接口下的多个实现类。

静态代理事先知道要代理的是什么，而动态代理不知道要代理什么东西，只有在运行时才知道。

动态代理是实现 JDK 里的 InvocationHandler 接口的 invoke 方法，但注意的是代理的是接口，也就是你的业务类必须要实现接口，通过 Proxy 里的 newProxyInstance 得到代理对象。

还有一种动态代理 CGLIB，代理的是类，不需要业务类继承接口，通过派生的子类来实现代理。通过在运行时，动态修改字节码达到修改类的目的。

AOP 编程就是基于动态代理实现的，比如著名的 Spring 框架、Hibernate 框架等等都是动态代理的使用例子。

三、Java 中的设计模式&回收机制

1、你所知道的设计模式有哪些

Java 中一般认为有 23 种设计模式，我们不需要所有的都会，但是其中常用的几种设计模式应该去掌握。下面列出了所有的设计模式。需要掌握的设计模式我单独列出来了，当然能掌握的越多越好。

总体来说设计模式分为三大类：

创建型模式，共五种：**工厂方法模式**、**抽象工厂模式**、**单例模式**、**建造者模式**、**原型模式**。

结构型模式，共七种：**适配器模式**、**装饰器模式**、**代理模式**、**外观模式**、**桥接模式**、**组合模式**、**享元模式**。

行为型模式，共十一种：**策略模式**、**模板方法模式**、**观察者模式**、**迭代子模式**、**责任链模式**、**命令模式**、**备忘录模式**、**状态模式**、**访问者模式**、**中介者模式**、**解释器模式**。

2、单例设计模式

最好理解的一种设计模式，分为懒汉式和饿汉式。

◆ 饿汉式：

```
public class Singleton {  
    // 直接创建对象  
    public static Singleton instance = new Singleton();  
  
    // 私有化构造函数  
    private Singleton() {  
    }  
  
    // 返回对象实例  
    public static Singleton getInstance() {  
        return instance;  
    }  
}
```

◆ 懒汉式：

```
public class Singleton {  
    // 声明变量  
    private static volatile Singleton singleton = null;  
  
    // 私有构造函数  
    private Singleton() {  
    }  
  
    // 提供对外方法  
    public static Singleton getInstance() {  
        if (singleton == null) {  
            synchronized (Singleton.class) {  
                if (singleton == null) {  
                    singleton = new Singleton();  
                }  
            }  
        }  
        return singleton;  
    }  
}
```

3、工厂设计模式

工厂模式分为工厂方法模式和抽象工厂模式。

◆ 工厂方法模式

工厂方法模式分为三种：普通工厂模式，就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。

多个工厂方法模式，是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。

静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

◆ 普通工厂模式

```
public interface Sender {
    public void Send();
}

public class MailSender implements Sender {

    @Override
    public void Send() {
        System.out.println("this is mail sender!");
    }
}

public class SmsSender implements Sender {

    @Override
    public void Send() {
        System.out.println("this is sms sender!");
    }
}

public class SendFactory {
    public Sender produce(String type) {
        if ("mail".equals(type)) {
            return new MailSender();
        } else if ("sms".equals(type)) {
            return new SmsSender();
        } else {
            System.out.println("请输入正确的类型!");
            return null;
        }
    }
}
```

◆ 多个工厂方法模式

该模式是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，

而多个工厂方法模式是提供多个工厂方法，分别创建对象。

```
public class SendFactory {
    public Sender produceMail(){
        return new MailSender();
    }

    public Sender produceSms(){
        return new SmsSender();
    }
}

public class FactoryTest {
    public static void main(String[] args) {
        SendFactory factory = new SendFactory();
        Sender sender = factory.produceMail();
        sender.send();
    }
}
```

◆ 静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

```
public class SendFactory {
    public static Sender produceMail(){
        return new MailSender();
    }

    public static Sender produceSms(){
        return new SmsSender();
    }
}

public class FactoryTest {
    public static void main(String[] args) {
        Sender sender = SendFactory.produceMail();
        sender.send();
    }
}
```

◆ 抽象工厂模式

工厂方法模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，如何解决？就用到抽象工厂模式，创建多个工厂类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。

```
public interface Provider {
    public Sender produce();
}

-----

public interface Sender {
    public void send();
}

-----

public class MailSender implements Sender {

    @Override
    public void send() {
        System.out.println("this is mail sender!");
    }
}

-----

public class SmsSender implements Sender {

    @Override
    public void send() {
        System.out.println("this is sms sender!");
    }
}

-----

public class SendSmsFactory implements Provider {

    @Override
    public Sender produce() {
        return new SmsSender();
    }
}
```

```
public class SendMailFactory implements Provider {  
  
    @Override  
    public Sender produce() {  
        return new MailSender();  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Provider provider = new SendMailFactory();  
        Sender sender = provider.produce();  
        sender.send();  
    }  
}
```

4、建造者模式 (Builder)

工厂类模式提供的是创建单个类的模式，而建造者模式则是将各种产品集中起来进行管理，用来创建复合对象，所谓复合对象就是指某个类具有不同的属性，其实建造者模式就是前面抽象工厂模式和最后的 Test 结合起来得到的。

```
public class Builder {  
    private List<Sender> list = new ArrayList<Sender>();  
  
    public void produceMailSender(int count) {  
        for (int i = 0; i < count; i++) {  
            list.add(new MailSender());  
        }  
    }  
  
    public void produceSmsSender(int count) {  
        for (int i = 0; i < count; i++) {  
            list.add(new SmsSender());  
        }  
    }  
}
```



```
public class TestBuilder {  
    public static void main(String[] args) {  
        Builder builder = new Builder();  
        builder.produceMailSender(10);  
    }  
}
```

5、适配器设计模式

适配器模式将某个类的接口转换成客户端期望的另一个接口表示，目的是消除由于接口不匹配所造成的类的兼容性问题。主要分为三类：类的适配器模式、对象的适配器模式、接口的适配器模式。

◆ 类的适配器模式

```
public class Source {  
    public void method1() {  
        System.out.println("this is original method!");  
    }  
}  
-----  
public interface Targetable {  
    /* 与原类中的方法相同 */  
    public void method1();  
    /* 新类的方法 */  
    public void method2();  
}  
public class Adapter extends Source implements Targetable {  
    @Override  
    public void method2() {  
        System.out.println("this is the targetable method!");  
    }  
}  
public class AdapterTest {  
    public static void main(String[] args) {  
        Targetable target = new Adapter();  
        target.method1();  
        target.method2();  
    }  
}
```

◆ 对象的适配器模式

基本思路和类的适配器模式相同，只是将 Adapter 类作修改，这次不继承 Source 类，而是持有 Source 类的实例，以达到解决兼容性的问题。

```
public class Wrapper implements Targetable {
    private Source source;

    public Wrapper(Source source) {
        super();
        this.source = source;
    }

    @Override
    public void method2() {
        System.out.println("this is the targetable method!");
    }

    @Override
    public void method1() {
        source.method1();
    }
}

-----

public class AdapterTest {

    public static void main(String[] args) {
        Source source = new Source();
        Targetable target = new Wrapper(source);
        target.method1();
        target.method2();
    }
}
```

◆ 接口的适配器模式

接口的适配器是这样的：有时我们写的一个接口中有多个抽象方法，当我们写该接口的实现类时，必须实现该接口的所有方法，这明显有时比较浪费，因为并不是所有的方法都是我们需要的，有时只需要某一些，此处为了解决这个问题，我们引入了接口的适配器模式，借助于一个抽象类，该抽象类实现了该接口，实现了所有的方法，而我们不

和原始的接口打交道，只和该抽象类取得联系，所以我们写一个类，继承该抽象类，重写我们需要的方法就行。

6、装饰模式 (Decorator)

顾名思义，装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例。

```
public interface Sourceable {
    public void method();
}

-----

public class Source implements Sourceable {
    @Override
    public void method() {
        System.out.println("the original method!");
    }
}

-----

public class Decorator implements Sourceable {
    private Sourceable source;
    public Decorator(Sourceable source) {
        super();
        this.source = source;
    }

    @Override
    public void method() {
        System.out.println("before decorator!");
        source.method();
        System.out.println("after decorator!");
    }
}

-----

public class DecoratorTest {
    public static void main(String[] args) {
        Sourceable source = new Source();
        Sourceable obj = new Decorator(source);
        obj.method();
    }
}
```

7、策略模式 (strategy)

策略模式定义了一系列算法，并将每个算法封装起来，使他们可以相互替换，且算法的变化不会影响到使用算法的客户。需要设计一个接口，为一系列实现类提供统一的方法，多个实现类实现该接口，设计一个抽象类（可有可无，属于辅助类），提供辅助函数。策略模式的决定权在用户，系统本身提供不同算法的实现，新增或者删除算法，对各种算法做封装。因此，策略模式多用在算法决策系统中，外部用户只需要决定用哪个算法即可。

```
public interface ICalculator {
    public int calculate(String exp);
}

-----

public class Minus extends AbstractCalculator implements ICalculator {

    @Override
    public int calculate(String exp) {
        int arrayInt[] = split(exp, "-");
        return arrayInt[0] - arrayInt[1];
    }
}

-----

public class Plus extends AbstractCalculator implements ICalculator {

    @Override
    public int calculate(String exp) {
        int arrayInt[] = split(exp, "\\+");
        return arrayInt[0] + arrayInt[1];
    }
}

-----

public class AbstractCalculator {
    public int[] split(String exp, String opt) {
        String array[] = exp.split(opt);
        int arrayInt[] = new int[2];
        arrayInt[0] = Integer.parseInt(array[0]);
        arrayInt[1] = Integer.parseInt(array[1]);
        return arrayInt;
    }
}
```

```
public class StrategyTest {
    public static void main(String[] args) {
        String exp = "2+8";
        ICalculator cal = new Plus();
        int result = cal.calculate(exp);
        System.out.println(result);
    }
}
```

8、观察者模式 (Observer)

观察者模式很好理解，类似于邮件订阅和 RSS 订阅，当我们浏览一些博客或 wiki 时，经常会看到 RSS 图标，就这意思是，当你订阅了该文章，如果后续有更新，会及时通知你。其实，简单来讲就一句话：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化！对象之间是一种一对多的关系。

```
public interface Observer {
    public void update();
}

public class Observer1 implements Observer {
    @Override
    public void update() {
        System.out.println("observer1 has received!");
    }
}

public class Observer2 implements Observer {
    @Override
    public void update() {
        System.out.println("observer2 has received!");
    }
}

public interface Subject {
    /*增加观察者*/
    public void add(Observer observer);

    /*删除观察者*/
    public void del(Observer observer);
}
```

```
/*通知所有的观察者*/
public void notifyObservers();

/*自身的操作*/
public void operation();
}

public abstract class AbstractSubject implements Subject {

    private Vector<Observer> vector = new Vector<Observer>();

    @Override
    public void add(Observer observer) {
        vector.add(observer);
    }

    @Override
    public void del(Observer observer) {
        vector.remove(observer);
    }

    @Override
    public void notifyObservers() {
        Enumeration<Observer> enumo = vector.elements();
        while (enumo.hasMoreElements()) {
            enumo.nextElement().update();
        }
    }
}

public class MySubject extends AbstractSubject {

    @Override
    public void operation() {
        System.out.println("update self!");
        notifyObservers();
    }
}

public class ObserverTest {
    public static void main(String[] args) {
        Subject sub = new MySubject();
        sub.add(new Observer1());
    }
}
```

```
        sub.add(new Observer2());  
        sub.operation();  
    }  
}
```

9、JVM 垃圾回收机制和常见算法

理论上讲 Sun 公司只定义了垃圾回收机制规则而不局限于其实现算法，因此不同厂商生产的虚拟机采用的算法也不尽相同。

GC (Garbage Collector) 在回收对象前首先必须发现那些无用的对象，如何去发现定位这些无用的对象？常用的**搜索算法**如下：

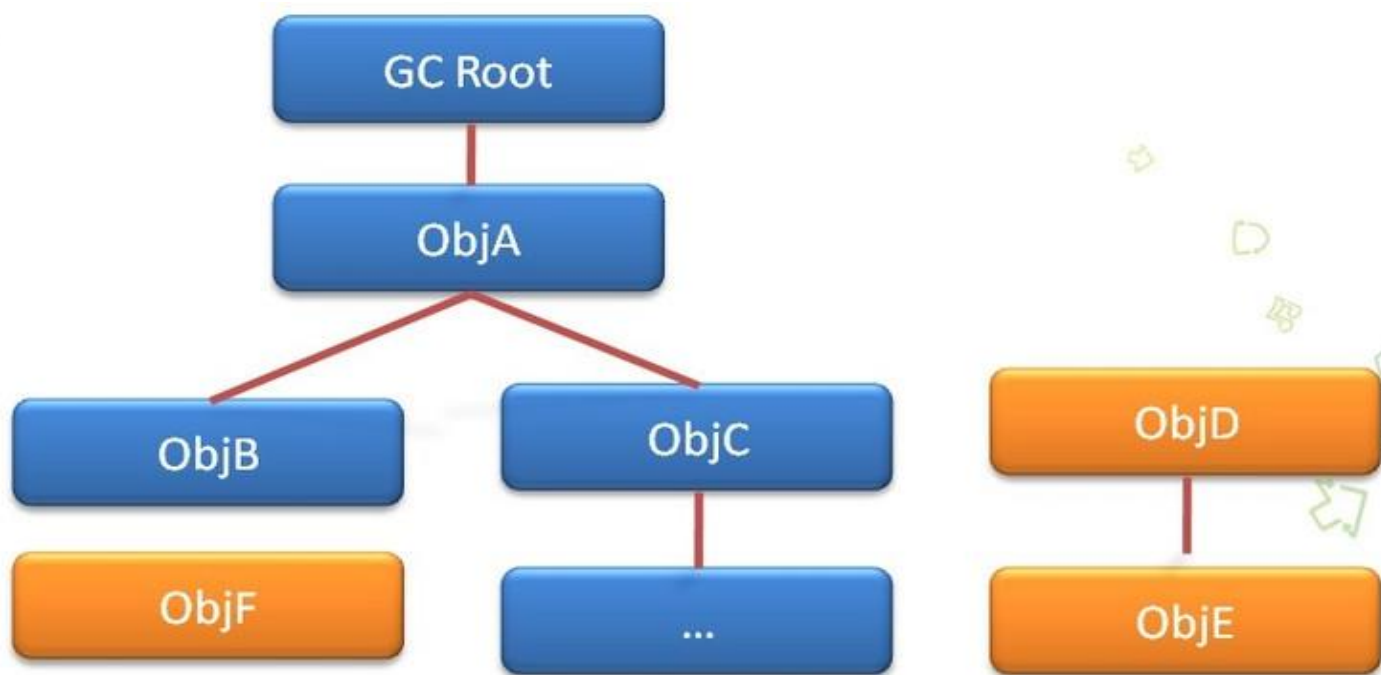
1) 引用计数器算法 (废弃)

引用计数器算法是给每个对象设置一个计数器，当有地方引用这个对象的时候，计数器+1，当引用失效的时候，计数器-1，当计数器为 0 的时候，JVM 就认为对象不再被使用，是“垃圾”了。

引用计数器实现简单，效率高；但是不能解决循环引用问题（A 对象引用 B 对象，B 对象又引用 A 对象，但是 A,B 对象已不被任何其他对象引用），同时每次计数器的增加和减少都带来了额外很多的开销，所以在 JDK1.1 之后，这个算法已经不再使用了。

2) 根搜索算法 (使用)

根搜索算法是通过一些“GC Roots”对象作为起点，从这些节点开始往下搜索，搜索通过的路径成为引用链 (Reference Chain)，当一个对象没有被 GC Roots 的引用链连接的时候，说明这个对象是不可用的。



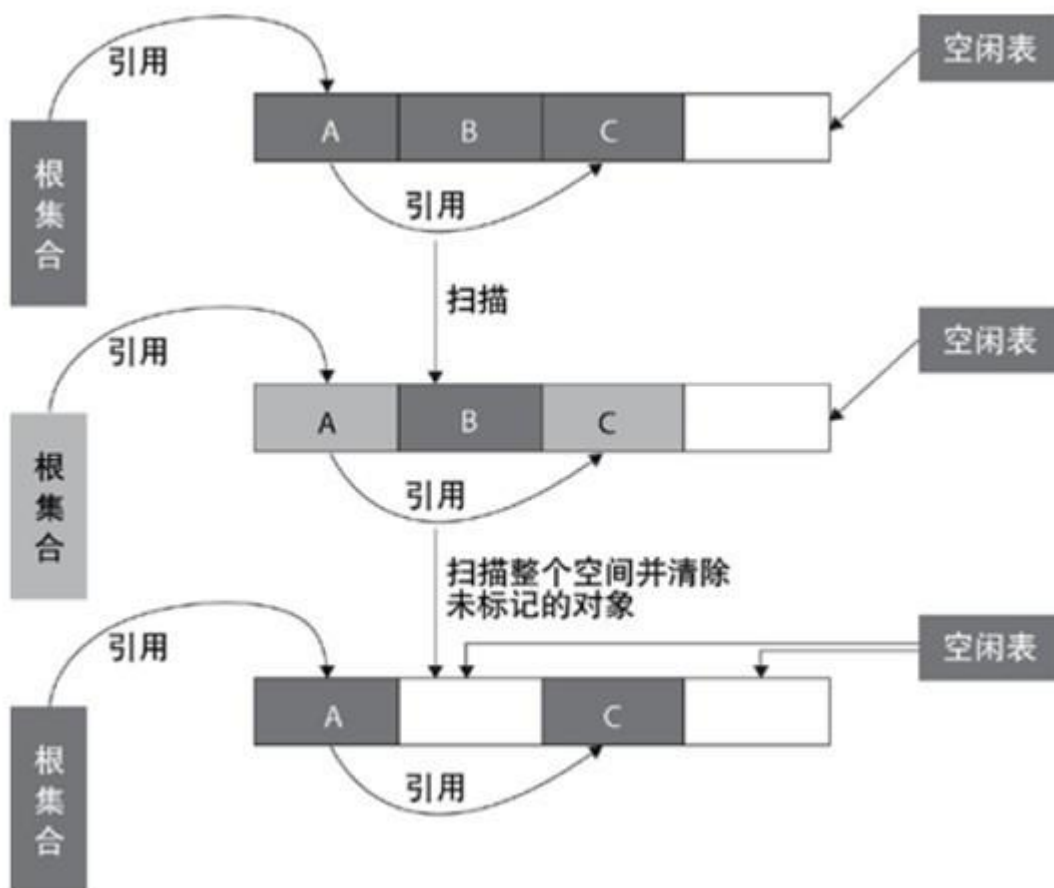
GC Roots 对象包括：

- a) 虚拟机栈（栈帧中的本地变量表）中的引用的对象。
- b) 方法区域中的类静态属性引用的对象。
- c) 方法区域中常量引用的对象。
- d) 本地方法栈中 JNI（Native 方法）的引用的对象。

通过上面的算法搜索到无用对象之后，就是回收过程，**回收算法**如下：

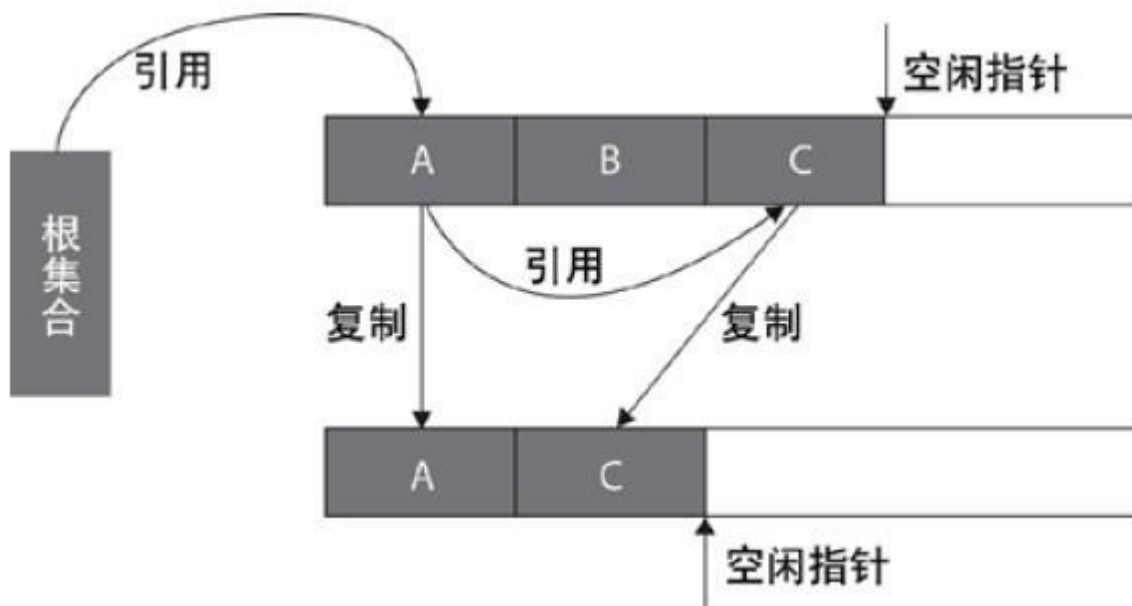
1) 标记—清除算法（Mark-Sweep）（DVM 使用的算法）

标记—清除算法包括两个阶段：“标记”和“清除”。在标记阶段，确定所有要回收的对象，并做标记。清除阶段紧随标记阶段，将标记阶段确定不可用的对象清除。标记—清除算法是基础的收集算法，标记和清除阶段的效率不高，而且清除后会产生大量的不连续空间，这样当程序需要分配大内存对象时，可能无法找到足够的连续空间。



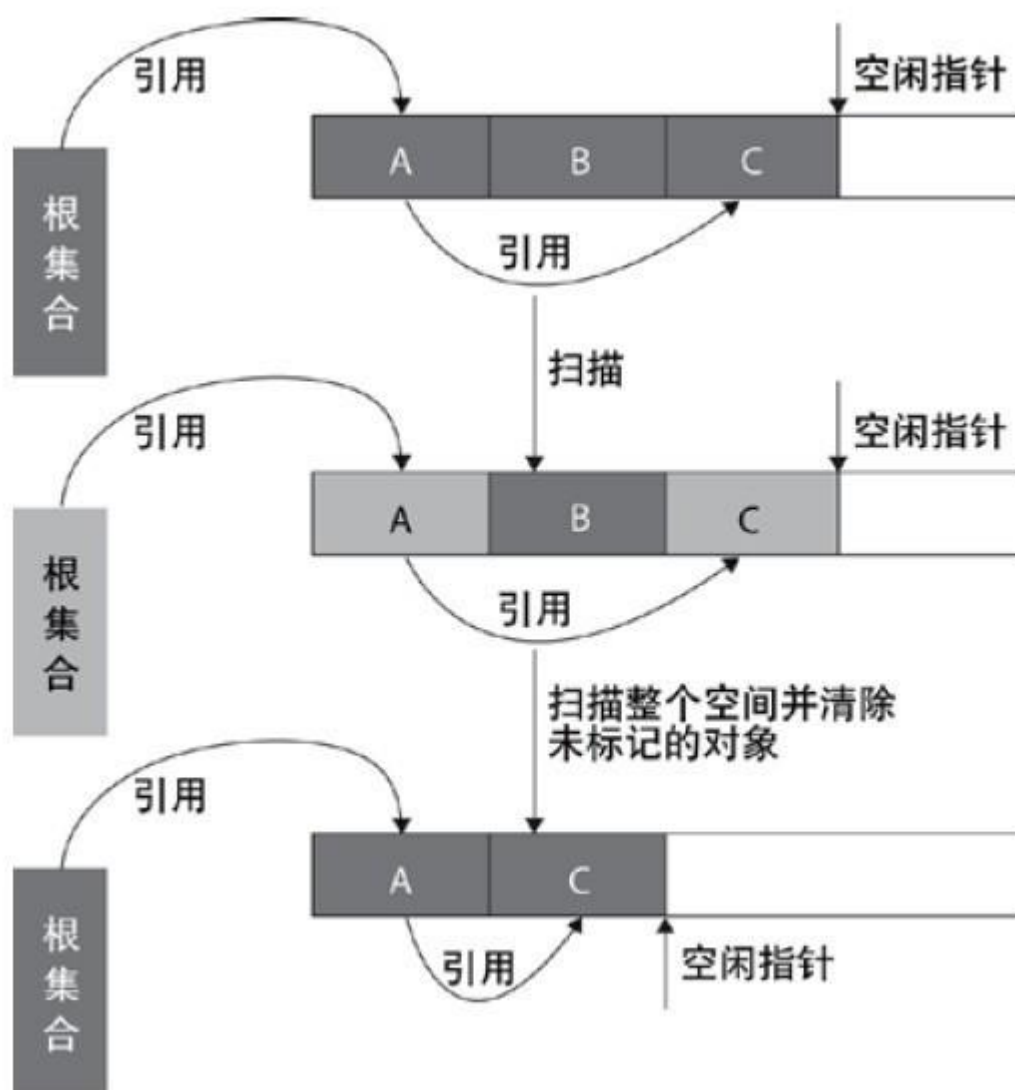
2) 复制算法 (Copying)

复制算法是把内存分成大小相等的两块，每次使用其中一块，当垃圾回收的时候，把存活的对象复制到另一块上，然后把这块内存整个清理掉。复制算法实现简单，运行效率高，但是由于每次只能使用其中的一半，造成内存的利用率不高。现在的 JVM 用复制方法收集新生代，由于新生代中大部分对象（98%）都是朝生夕死的，所以两块内存的比例不是 1:1(大概是 8:1)。



3) 标记—整理算法 (Mark-Compact)

标记—整理算法和标记—清除算法一样，但是标记—整理算法不是把存活对象复制到另一块内存，而是把存活对象往内存的一端移动，然后直接回收边界以外的内存。标记—整理算法提高了内存的利用率，并且它适合在收集对象存活时间较长的老年代。



4) 分代收集 (Generational Collection)

分代收集是根据对象的存活时间把内存分为新生代和老年代，根据各个代对象的存活特点，每个代采用不同的垃圾回收算法。新生代采用复制算法，老年代采用标记—整理算法。垃圾算法的实现涉及大量的程序细节，而且不同的虚拟机平台实现的方法也各不相同。

10、谈谈 JVM 的内存结构和内存分配

a) Java 内存模型

Java 虚拟机将其管辖的内存大致分三个逻辑部分：方法区(Method Area)、Java 栈和 Java 堆。

1、方法区是静态分配的，编译器将变量绑定在某个存储位置上，而且这些绑定不会在运行时改变。

常数池，源代码中的命名常量、String 常量和 static 变量保存在方法区。

2、Java Stack 是一个逻辑概念，特点是后进先出。一个栈的空间可能是连续的，也可能是不连续的。

最典型的 Stack 应用是方法的调用，Java 虚拟机每调用一次方法就创建一个方法帧（frame），退出该方法则对应的 方法帧被弹出(pop)。栈中存储的数据也是运行时确定的。

3、Java 堆分配(heap allocation)意味着以随意的顺序，在运行时进行存储空间分配和收回的内存管理模型。

堆中存储的数据常常是大小、数量和生命期在编译时无法确定的。Java 对象的内存总是在 heap 中分配。

我们每天都在写代码，每天都在使用 JVM 的内存。

b) java 内存分配

1、基础数据类型直接在栈空间分配;

2、方法的形式参数，直接在栈空间分配，当方法调用完成后从栈空间回收;

3、引用数据类型，需要用 new 来创建，既在栈空间分配一个地址空间，又在堆空间分配对象的类变量;

4、方法的引用参数，在栈空间分配一个地址空间，并指向堆空间的对象区，当方法调用完后从栈空间回收;

5、局部变量 new 出来时，在栈空间和堆空间中分配空间，当局部变量生命周期结束后，栈空间立刻被回收，堆空间区域等待 GC 回收;

6、方法调用时传入的实际参数，先在栈空间分配，在方法调用完成后从栈空间释放;

7、字符串常量在 DATA 区域分配，this 在堆空间分配;

8、数组既在栈空间分配数组名称，又在堆空间分配数组实际的大小！

11、Java 中引用类型都有哪些？（重要）

Java 中对象的引用分为四种级别，这四种级别由高到低依次为：强引用、软引用、弱引用和虚引用。

◆ 强引用（StrongReference）

这个就不多说，我们写代码天天在用的就是强引用。如果一个对象被被人拥有强引用，那么垃圾回收器绝不会回收它。当内存空间不足，Java 虚拟机宁愿抛出 `OutOfMemoryError` 错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足问题。

Java 的对象是位于 heap 中的，heap 中对象有强可及对象、软可及对象、弱可及对象、虚可及对象和不可到达对象。应用的强弱顺序是强、软、弱、和虚。对于对象是属于哪种可及的对象，由他的最强的引用决定。如下代码：

```
String abc=new String("abc"); //1
SoftReference<String> softRef=new SoftReference<String>(abc); //2
WeakReference<String> weakRef = new WeakReference<String>(abc); //3
abc=null; //4
softRef.clear();//5
```

第一行在 heap 堆中创建内容为 “abc” 的对象，并建立 abc 到该对象的强引用，该对象是强可及的。

第二行和第三行分别建立对 heap 中对象的软引用和弱引用，此时 heap 中的 abc 对象已经有 3 个引用，显然此时 abc 对象仍是强可及的。

第四行之后 heap 中对象不再是强可及的，变成软可及的。

第五行执行之后变成弱可及的。

◆ 软引用 (`SoftReference`)

如果一个对象只具有软引用，那么如果内存空间足够，垃圾回收器就不会回收它，如果内存空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。

软引用可以和一个引用队列 (`ReferenceQueue`) 联合使用，如果软引用所引用的对象被垃圾回收，Java 虚拟机就会把这个软引用加入到与之关联的引用队列中。

软引用是主要用于内存敏感的高速缓存。在 jvm 报告内存不足之前会清除所有的软引用，这样以来 gc 就有可能收集软可及的对象，可能解决内存吃紧问题，避免内存溢出。什么时候会被收集取决于 gc 的算法和 gc 运行

时可用内存的大小。当 gc 决定要收集软引用时执行以下过程,以上面的 softRef 为例：

- 1 首先将 softRef 的 referent (abc) 设置为 null , 不再引用 heap 中的 new String("abc")对象。
- 2 将 heap 中的 new String("abc")对象设置为可结束的(finalizable)。
- 3 当 heap 中的 new String("abc")对象的 finalize()方法被运行而且该对象占用的内存被释放， softRef

被添加到它的 ReferenceQueue(如果有的话)中。

注意:对 ReferenceQueue 软引用和弱引用可以有可无，但是虚引用必须有。

被 Soft Reference 指到的对象，即使没有任何 Direct Reference，也不会被清除。一直要到 JVM 内存不足且没有 Direct Reference 时才会清除，SoftReference 是用来设计 object-cache 之用的。如此一来 SoftReference 不但可以把对象 cache 起来，也不会造成内存不足的错误（OutOfMemoryError）。

◆ 弱引用（WeakReference）

如果一个对象只具有弱引用，那该类就是可有可无的对象，因为只要该对象被 gc 扫描到了随时都会把它干掉。

弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。

弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java 虚拟机就会把这个弱引用加入到与之关联的引用队列中。

◆ 虚引用（PhantomReference）

"虚引用"顾名思义，就是形同虚设，与其他几种引用都不同，虚引用并不会决定对象的生命周期。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收。虚引用主要用来跟踪对象被垃圾回收的活动。

虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列（ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关

联的引用队列中。程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。程序如果发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。

建立虚引用之后通过 `get` 方法返回结果始终为 `null`，通过源代码你会发现，虚引用通向会把引用的对象写进 `referent`，只是 `get` 方法返回结果为 `null`。先看一下和 `gc` 交互的过程再说一下他的作用。

- 1 不把 `referent` 设置为 `null`，直接把 `heap` 中的 `new String("abc")` 对象设置为可结束的(`finalizable`)。
- 2 与软引用和弱引用不同，先把 `PhantomReference` 对象添加到它的 `ReferenceQueue` 中，然后在释放虚可及的对象。

12、heap 和 stack 有什么区别 (2017-2-23)

从以下几个方面阐述堆 (`heap`) 和栈 (`stack`) 的区别。

1. 申请方式

`stack`: 由系统自动分配。例如，声明在函数中一个局部变量 `int b`；系统自动在栈中为 `b` 开辟空间

`heap`: 需要程序员自己申请，并指明大小，在 `c` 中 `malloc` 函数，对于 `Java` 需要手动 `new Object()` 的形式开辟

2. 申请后系统的响应

`stack`：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

`heap`：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

3. 申请大小的限制

`stack`：栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是

系统预先规定好的，在 WINDOWS 下，栈的大小是 2M（也有的说是 1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示 overflow。因此，能从栈获得的空间较小。

heap：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

4. 申请效率的比较：

stack：由系统自动分配，速度较快。但程序员是无法控制的。

heap：由 new 分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便。

5. heap 和 stack 中的存储内容

stack：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的 C 编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。

当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。

heap：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

6. 数据结构层面的区别

还有就是数据结构方面的堆和栈，这些都是不同的概念。这里的堆实际上指的就是（满足堆性质的）优先队列的一种数据结构，第 1 个元素有最高的优先权；栈实际上就是满足先进后出的性质的数学或数据结构。

虽然堆栈，堆栈的说法是连起来叫，但是他们还是有很大区别的，连着叫只是由于历史的原因。

7. 拓展知识（Java 中堆栈的应用）

1). 栈(stack)与堆(heap)都是 Java 用来在 Ram 中存放数据的地方。与 C++不同，Java 自动管理栈和堆，程序员不能直接地设置栈或堆。

2). 栈的优势是，存取速度比堆要快，仅次于直接位于 CPU 中的寄存器。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。另外，栈数据可以共享，详见第 3 点。堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，Java 的垃圾回收器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。

3). Java 中的数据类型有两种。

一种是基本类型(primitive types)，共有 8 种，即 int, short, long, byte, float, double, boolean, char(注意，并没有 string 的基本类型)。这种类型的定义是通过诸如 `int a = 3; long b = 255L;` 的形式来定义的，称为自动变量(自动变量：只在定义它们的时候才创建，在定义它们的函数返回时系统回收变量所占存储空间。对这些变量存储空间的分配和回收是由系统自动完成的。)。值得注意的是，自动变量存的是字面值，不是类的实例，即不是类的引用，这里并没有类的存在。如 `int a = 3;` 这里的 a 是一个指向 int 类型的引用，指向 3 这个字面值。这些字面值的数据，由于大小可知，生存期可知(这些字面值固定定义在某个程序块里面，程序块退出后，字段值就消失了)，出于追求速度的原因，就存在于栈中。

另外，栈有一个很重要的特殊性，就是存在栈中的数据可以共享。假设我们同时定义

```
int a = 3;
```

```
int b = 3;
```

编译器先处理 `int a = 3;`；首先它会在栈中创建一个变量为 a 的引用，然后查找有没有字面值为 3 的地址，没找到，就开辟一个存放 3 这个字面值的地址，然后将 a 指向 3 的地址。接着处理 `int b = 3;`；在创建完 b 的引用变量后，由于在栈中已经有 3 这个字面值，便将 b 直接指向 3 的地址。这样，就出现了 a 与 b 同时均指向 3 的情况。

特别注意的是，这种字面值的引用与类对象的引用不同。假定两个类对象的引用同时指向一个对象，如果一个对象引用变量修改了这个对象的内部状态，那么另一个对象引用变量也即刻反映出这个变化。相反，通过字面值的引用来修改其值，不会导致另一个指向此字面值的引用的值也跟着改变的情况。如上例，我们定义完 a 与 b 的值后，再令 `a=4;`；那么，b 不会等于 4，还是等于 3。在编译器内部，遇到 `a=4;` 时，它就会重新搜索栈中是否有 4 的字面

值，如果没有，重新开辟地址存放 4 的值；如果已经有了，则直接将 a 指向这个地址。因此 a 值的改变不会影响到 b 的值。

另一种是包装类数据，如 Integer, String, Double 等将相应的基本数据类型包装起来的类。这些类数据全部存在于堆中，Java 用 new() 语句来显示地告诉编译器，在运行时才根据需要动态创建，因此比较灵活，但缺点是要占用更多的时间。

4). 每个 JVM 的线程都有自己的私有的栈空间，随线程创建而创建，java 的 stack 存放的是 frames，java 的 stack 和 c 的不同，只是存放本地变量，返回值和调用方法，不允许直接 push 和 pop frames，因为 frames 可能是有 heap 分配的，所以 java 的 stack 分配的内存不需要是连续的。java 的 heap 是所有线程共享的，堆存放所有 runtime data，里面是所有的对象实例和数组，heap 是 JVM 启动时创建。

5). String 是一个特殊的包装类数据。即可以用 String str = new String("abc"); 的形式来创建，也可以用 String str = "abc"; 的形式来创建(作为对比，在 JDK 5.0 之前，你从未见过 Integer i = 3; 的表达式，因为类与字面值是不能通用的，除了 String。而在 JDK 5.0 中，这种表达式是可以的！因为编译器在后台进行 Integer i = new Integer(3) 的转换)。前者是规范的类的创建过程，即在 Java 中，一切都是对象，而对象是类的实例，全部通过 new() 的形式来创建。那为什么在 String str = "abc"; 中，并没有通过 new() 来创建实例，是不是违反了上述原则？其实没有。

5.1). 关于 String str = "abc" 的内部工作。Java 内部将此语句转化为以下几个步骤：

(1) 先定义一个名为 str 的对 String 类的对象引用变量：String str；

(2) 在栈中查找有没有存放值为 "abc" 的地址，如果没有，则开辟一个存放字面值为 "abc" 的地址，接着创建一个新的 String 类的对象 o，并将 o 的字符串值指向这个地址，而且在栈中这个地址旁边记下这个引用的对象 o。如果已经有了值为 "abc" 的地址，则查找对象 o，并返回 o 的地址。

(3) 将 str 指向对象 o 的地址。

值得注意的是，一般 String 类中字符串值都是直接存值的。但像 String str = "abc"; 这种场合下，其字符

串值却是保存了一个指向存在栈中数据的引用！

为了更好地说明这个问题，我们可以通过以下的几个代码进行验证。

```
String str1 = "abc";

String str2 = "abc";

System.out.println(str1==str2);    //true
```

注意，我们这里并不用 `str1.equals(str2)` 的方式，因为这将比较两个字符串的值是否相等。`==`号，根据JDK的说明，只有在两个引用都指向了同一个对象时才返回真值。而我们在这里要看的是，`str1` 与 `str2` 是否都指向了同一个对象。

结果说明，JVM 创建了两个引用 `str1` 和 `str2`，但只创建了一个对象，而且两个引用都指向了这个对象。

我们再来更进一步，将以上代码改成：

```
String str1 = "abc";

String str2 = "abc";

str1 = "bcd";

System.out.println(str1 + "," + str2);    //bcd, abc

System.out.println(str1==str2);    //false
```

这就是说，赋值的变化导致了类对象引用的变化，`str1` 指向了另外一个新对象！而 `str2` 仍旧指向原来的对象。

上例中，当我们将 `str1` 的值改为"bcd"时，JVM 发现在栈中没有存放该值的地址，便开辟了这个地址，并创建了一个新的对象，其字符串的值指向这个地址。

事实上，`String` 类被设计成为不可改变(`immutable`)的类。如果你要改变其值，可以，但 JVM 在运行时根据新值悄悄创建了一个新对象，然后将这个对象的地址返回给原来类的引用。这个创建过程虽说是完全自动进行的，但它毕竟占用了更多的时间。在对时间要求比较敏感的环境中，会带有一定的不良影响。

再修改原来代码：

```
String str1 = "abc";

String str2 = "abc";

str1 = "bcd";

String str3 = str1;

System.out.println(str3);    //bcd

String str4 = "bcd";

System.out.println(str1 == str4);    //true
```

str3 这个对象的引用直接指向 str1 所指向的对象(注意，str3 并没有创建新对象)。当 str1 改完其值后，再创建一个 String 的引用 str4，并指向因 str1 修改值而创建的新的对象。可以发现，这回 str4 也没有创建新的对象，从而再次实现栈中数据的共享。

我们再接着看以下的代码。

```
String str1 = new String("abc");

String str2 = "abc";

System.out.println(str1==str2);    //false
```

创建了两个引用。创建了两个对象。两个引用分别指向不同的两个对象。

以上两段代码说明，只要是用 new() 来新建对象的，都会在堆中创建，而且其字符串是单独存值的，即使与栈中的数据相同，也不会与栈中的数据共享。

6). 数据类型包装类的值不可修改。不仅仅是 String 类的值不可修改，所有的数据类型包装类都不能更改其内部的值。

7). 结论与建议：

(1)我们在用诸如 String str = "abc"; 的格式定义类时，总是想当然地认为，我们创建了 String 类的对象 str。担心陷阱！对象可能并没有被创建！唯一可以肯定的是，指向 String 类的引用被创建了。至于这个引用到底是否

指向了一个新的对象，必须根据上下文来考虑，除非你通过 `new()` 方法来显要地创建一个新的对象。因此，更为准确的说法是，我们创建了一个指向 `String` 类的对象的引用变量 `str`，这个对象引用变量指向了某个值为 "abc" 的 `String` 类。清醒地认识到这一点对排除程序中难以发现的 bug 是很有帮助的。

(2)使用 `String str = "abc";` 的方式，可以在一定程度上提高程序的运行速度，因为 JVM 会自动根据栈中数据的实际情况来决定是否有必要创建新对象。而对于 `String str = new String("abc");` 的代码，则一概在堆中创建新对象，而不管其字符串值是否相等，是否有必要创建新对象，从而加重了程序的负担。这个思想应该是享元模式的思想，但 JDK 的内部在这里实现是否应用了这个模式，不得而知。

(3)当比较包装类里面的数值是否相等时，用 `equals()` 方法；当测试两个包装类的引用是否指向同一个对象时，用 `==`。

(4)由于 `String` 类的 `immutable` 性质，当 `String` 变量需要经常变换其值时，应该考虑使用 `StringBuffer` 类，以提高程序效率。

如果 java 不能成功分配 heap 的空间，将抛出 `OutOfMemoryError`。

四、Java 的类加载器（2015-12-2）

1、Java 的类加载器的种类都有哪些？

- 1、根类加载器(Bootstrap) --C++写的，看不到源码
- 2、扩展类加载器 (Extension) --加载位置：jre\lib\ext 中
- 3、系统(应用)类加载器(System\App) --加载位置：classpath 中
- 4、自定义加载器(必须继承 `ClassLoader`)

2、类什么时候被初始化？

- 1) 创建类的实例，也就是 `new` 一个对象

- 2) 访问某个类或接口的静态变量，或者对该静态变量赋值
- 3) 调用类的静态方法
- 4) 反射 (`Class.forName("com.lyj.load")`)
- 5) 初始化一个类的子类 (会首先初始化子类的父类)
- 6) JVM 启动时标明的启动类，即文件名和类名相同的那个类

只有这 6 中情况才会导致类的初始化。

类的初始化步骤：

- 1) 如果这个类还没有被加载和链接，那先进行加载和链接
- 2) 假如这个类存在直接父类，并且这个类还没有被初始化 (注意：在一个类加载器中，类只能初始化一次)，那就初始化直接的父类 (不适用于接口)
- 3) 加入类中存在初始化语句 (如 `static` 变量和 `static` 块)，那就依次执行这些初始化语句。

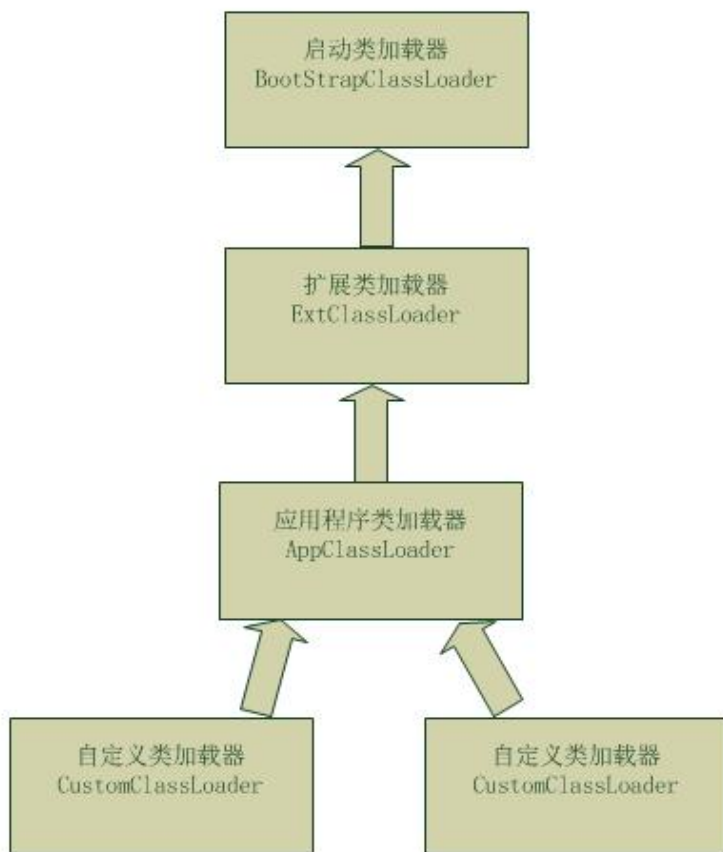
3、Java 类加载体系之 ClassLoader 双亲委托机制 (2017-2-24)

java 是一种类型安全的语言，它有四类称为安全沙箱机制的安全机制来保证语言的安全性，这四类安全沙箱分别是：

- 1) 类加载体系
- 2) `.class` 文件检验器
- 3) 内置于 Java 虚拟机 (及语言) 的安全特性
- 4) 安全管理器及 Java API

主要讲解类的加载体系：

java 程序中的 `.java` 文件编译完会生成 `.class` 文件，而 `.class` 文件就是通过被称为类加载器的 `ClassLoader` 加载的，而 `ClassLoader` 在加载过程中会使用“双亲委派机制”来加载 `.class` 文件，先上图：



BootstrapClassLoader：启动类加载器，该 ClassLoader 是 jvm 在启动时创建的，用于加载 `$JAVA_HOME$/jre/lib` 下面的类库（或者通过参数 `-Xbootclasspath` 指定）。由于启动类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用，所以不能直接通过引用进行操作。

ExtClassLoader 扩展类加载器，该 ClassLoader 是在 `sun.misc.Launcher` 里作为一个内部类 `ExtClassLoader` 定义的（即 `sun.misc.Launcher$ExtClassLoader`），`ExtClassLoader` 会加载 `$JAVA_HOME$/jre/lib/ext` 下的类库（或者通过参数 `-Djava.ext.dirs` 指定）。

AppClassLoader：应用程序类加载器，该 ClassLoader 同样是在 `sun.misc.Launcher` 里作为一个内部类 `AppClassLoader` 定义的（即 `sun.misc.Launcher$AppClassLoader`），`AppClassLoader` 会加载 `java` 环境变量 `CLASSPATH` 所指定的路径下的类库，而 `CLASSPATH` 所指定的路径可以通过 `System.getProperty("java.class.path")` 获取；当然，该变量也可以覆盖，可以使用参数 `-cp`，例如：`java -cp 路径`（可以指定要执行的 class 目录）。

CustomClassLoader：自定义类加载器，该 ClassLoader 是指我们自定义的 ClassLoader，比如 tomcat 的 StandardClassLoader 属于这一类；当然，大部分情况下使用 AppClassLoader 就足够了。

前面谈到了 ClassLoader 的几类加载器，而 ClassLoader 使用双亲委派机制来加载 class 文件的。ClassLoader 的双亲委派机制是这样的（这里先忽略掉自定义类加载器 CustomClassLoader）：

1) 当 AppClassLoader 加载一个 class 时，它首先不会自己去尝试加载这个类，而是把类加载请求委派给父类加载器 ExtClassLoader 去完成。

2) 当 ExtClassLoader 加载一个 class 时，它首先也不会自己去尝试加载这个类，而是把类加载请求委派给 BootstrapClassLoader 去完成。

3) 如果 BootstrapClassLoader 加载失败（例如在 \$JAVA_HOME\$/jre/lib 里未查找到该 class），会使用 ExtClassLoader 来尝试加载；

4) 若 ExtClassLoader 也加载失败，则会使用 AppClassLoader 来加载，如果 AppClassLoader 也加载失败，则会报出异常 ClassNotFoundException。

下面贴下 ClassLoader 的 loadClass(String name, boolean resolve)的源码：

```
1. protected synchronized Class<?> loadClass(String name, boolean resolve)
2. throws ClassNotFoundException {
3.     // 首先找缓存是否有 class
4.     Class c = findLoadedClass(name);
5.     if (c == null) {
6.         //没有判断有没有父类
7.         try {
8.             if (parent != null) {
9.                 //有的话，用父类递归获取 class
10.                 c = parent.loadClass(name, false);
11.             } else {
12.                 //没有父类。通过这个方法加载
13.                 c = findBootstrapClassOrNull(name);
14.             }
15.         } catch (ClassNotFoundException e) {
16.             // ClassNotFoundException thrown if class not found
17.             // from the non-null parent class loader
18.         }
```



```
19.         if (c == null) {
20.             // 如果还是没有找到，调用 findClass(name) 去找这个类
21.             c = findClass(name);
22.         }
23.     }
24.     if (resolve) {
25.         resolveClass(c);
26.     }
27.     return c;
28. }
```

代码很明朗：首先找缓存（findLoadedClass），没有的话就判断有没有 parent，有的话就用 parent 来递归的 loadClass，然而 ExtClassLoader 并没有设置 parent，则会通过 findBootstrapClassOrNull 来加载 class，而 findBootstrapClassOrNull 则会通过 JNI 方法“private native Class findBootstrapClass(String name)”来使用 BootStrapClassLoader 来加载 class。

然后如果 parent 未找到 class，则会调用 findClass 来加载 class，findClass 是一个 protected 的空方法，可以覆盖它以便自定义 class 加载过程。

另外，虽然 ClassLoader 加载类是使用 loadClass 方法，但是鼓励用 ClassLoader 的子类重写 findClass(String)，而不是重写 loadClass，这样就不会覆盖了类加载默认的双亲委派机制。

双亲委派托机制为什么安全

举个例子，ClassLoader 加载的 class 文件来源很多，比如编译器编译生成的 class、或者网络下载的字节码。而一些来源的 class 文件是不可靠的，比如我可以自定义一个 java.lang.Integer 类来覆盖 jdk 中默认的 Integer 类，例如下面这样：

```
1. package java.lang;
2. public class Integer {
3.     public Integer(int value) {
4.         System.exit(0);
5.     }
6. }
```

初始化这个 Integer 的构造器是会退出 JVM，破坏应用程序的正常进行，如果使用双亲委派机制的话该 Integer 类永远不会被调用，以为委托 BootstrapClassLoader 加载后会加载 JDK 中的 Integer 类而不会加载自定义的这个，可以看下下面这测试用例：

```
1. public static void main(String... args) {  
2.     Integer i = new Integer(1);  
3.     System.err.println(i);  
4. }
```

执行时 JVM 并未在 new Integer(1)时退出，说明未使用自定义的 Integer，于是就保证了安全性。

五、Java8 的新特性以及使用（2016-9-7）

整理中，待续...

4. Android 基础 (★★★★)

一、Android 基本常识

1、写 10 个简单的 linux 命令

mkdir 创建文件夹

rmdir 删除文件夹

rm 删除文件

mv 移动文件

cp 拷贝文件

cat 查看文件

tail 查看文件尾部

more 分页查看文件

cd 切换当前目录

ls 列出文件清单

reboot 重启

date 显示日期

cal 显示日历

ps 查看系统进程相当于 windows 的任务管理器

ifconfig 配置网络

2、书写出 android 工程的目录结构

src 源文件

gen 生成的文件 R 文件就在此

android.jar 依赖的 android sdk

assets 资源文件

bin 生成的字节码 apk 在此

libs 依赖 jar 和 so

res 资源文件

drawable

drawable-hdpi

layout

menu

values

AndroidManifest.xml

project.properties

3、什么是 ANR 如何避免它？

在 Android 上，如果你的应用程序有一段时间响应不够灵敏，系统会向用户显示一个对话框，这个对话框称作应用程序无响应（ANR：Application Not Responding）对话框。用户可以选择让程序继续运行，但是，他们在使用你的应用程序时，并不希望每次都要处理这个对话框。因此，在程序里对响应性能的设计很重要，这样，系统不会显示 ANR 给用户。

不同的组件发生 ANR 的时间不一样，主线程（Activity、Service）是 5 秒，BroadcastReceiver 是 10 秒。

解决方案：

将所有耗时操作，比如访问网络，Socket 通信，查询大量 SQL 语句，复杂逻辑计算等都放在子线程中去，然后通过 handler.sendMessage、runOnUiThread、AsyncTask 等方式更新 UI。无论如何都要确保用户界面操作的流畅度。如果耗时操作需要让用户等待，那么可以在界面上显示进度条。

4、谈谈 Android 的优点和不足之处

优点：

- 1、开放性，开源，免费，可定制
- 2、挣脱运营商束缚
- 3、丰富的硬件选择
- 4、不受任何限制的开发商
- 5、无缝结合的 Google 应用

缺点：

- 1、安全问题、隐私问题
- 2、同质化严重
- 3、运营商对 Android 手机仍然有影响
- 4、山寨化严重
- 5、过分依赖开发商，缺乏标准配置

5、一条最长的短信息约占多少 byte?

在国内的三大运营商通常情况下中文 70(包括标点)，英文 160 个。对于国外的其他运行商具体多长需要看运营商类型了。

android 内部是通过如下代码进行判断具体一个短信多少 byte 的。

```
ArrayList<String> android.telephony.SmsManager.divideMessage(String text)
```

6、sim 卡的 EF 文件有何作用？（不用看，废弃）

基本文件 EF(Elementary File)是 SIM 卡文件系统的一部分。

文件	文件标识符	文件缩写	中文名称	文件作用
MF	3F00	根目录	备注：所有非 ETSI GSM 协议中规定的应用文件由各厂家自行定义在根目录下（如：PIN1，PIN2…）	
EF _{ICCID}	2FE2	ICCID	SIM 卡唯一的识别号	包含运营商、卡商、发卡时间、省市代码等信息
DF _{GSM}	7F20	GSM 目录	备注：根据 ETSI GSM 09. 91 的规定 Phase2(或以上)的 SIM 卡中应该有 7F21 并指向 7F20, 用以兼容 Phase1 的手机	
EF _{LP} 语言选择	6F05	LP	语言选择文件	包含一种或多种语言编码
EF _{IMSI}	6F07	IMSI	国际移动用户识别符	包含 SIM 卡所对应的号段，比如 46000 代表 135—139 号段、46002 代表 1340—1348

EF _{KC} 语音加密密钥	6F20	Kc	计算密钥	用于 SIM 卡的加密、解密
EF _{PLMNsel} 网络选择表	6F30	PLMNsel	公共陆地网选择	决定 SIM 卡选择哪种网络，在这里应该选择中国移动的网络
EF _{HPLMN} 归属地网络选择表	6F31	HPLMN	两次搜索 PLMN 的时间间隔	两次搜索中国移动的网络的时间间隔
EF _{ACMmax} 最大计费额	6F37	ACMmax	包含累积呼叫表的最大值	全部的 ACM 数据存在 SIM 卡中，此处取最大值
EF _{SST} SIM 卡服务表	6F38	SST	SIM 卡服务列表	指出 SIM 卡可以提供服务的种类，哪些业务被激活哪些业务没有开通
EF _{ACM} 累加计费计数器	6F39	ACM	累计呼叫列表	当前的呼叫和以前的呼叫的单位总和
EF _{GID1} 分组识别 1	6F3E	GID1	1 级分组识别文件	包含特定的 SIM-ME 组合的标识符，可以识别一组特定的 SIM 卡
EF _{GID2} 分组识别 2	6F3F	GID2	2 级分组识别文件	包含特定的 SIM-ME 组合的标识符，可以识别一组特定的 SIM 卡
EF _{PUCT} 单位价格/货币表	6F41	PUCT	呼叫单位的价格和货币表	PUCT 是与计费通知有关的信息，ME 用这个信息结合 EFACM，以用户选择的货币来计算呼叫费用
EF _{CBMI} 小区广播识别号	6F45	CBMI	小区广播信息标识符	规定了用户希望 MS 采纳的小区广播消息内容的类型
EF _{SPN} 服务提供商	6F46	SPN	服务提供商名称	包含服务提供商的名称和 ME 显示的相应要求
EF _{CBMID}	6F48	CBMID	数据下载的小区广播消息识别符	移动台将收到的 CBMID 传送给 SIM 卡
EF _{SUME}	6F54	SUME	建立菜单单元	建立 SIM 卡中的菜单
EF _{BCCH} 广播信道	6F74	BCCH	广播控制信道	由于 BCCH 的存储，在选择小区时，MS 可以缩小对 BCCH 载波的搜索范围
EF _{ACC} 访问控制级别	6F78	ACC	访问控制级别	SIM 卡有 15 个级别，10 个普通级别，5 个高级级别

EF _{FPLMN} 禁止网络号	6F7B	FPLMN	禁用的 PLMN	禁止选择除中国移动以外的其他运营商，比如中国联通、中国卫通等
EF _{LOCI} 位置信息	6F7E	LOCI	位置信息	存储临时移动用户识别符、位置区信息等内容
EF _{AD} 管理数据	6FAD	AD	管理数据	包括关于不同类型 SIM 卡操作模式的信息。例如：常规模式（PLMN 用户用于 GSM 网络操作），型号认证模式（允许 ME 在无线设备的认证期间的特殊应用）；小区测试模式（在小区商用之前，进行小区测试），制造商特定模式（允许 ME 制造商在维护阶段进行特定的性能自动测试）
EF _{PHASE} 阶段	6FAE	PHASE	阶段标识	标识 SIM 卡所处的阶段信息，比如是普通 SIM 卡还是 STK 卡等
DF _{TELECOM}	7F10	电信目录		
EF _{ADN} 缩位拨号	6F3A	AND	电话簿	用于将电话记录存放在 SIM 卡中
EF _{FDN} 固定拨号	6F3B	FDN	固定拨号	包括固定拨号（FDN）和/或补充业务控制字串（SSC），还包括相关网络/承载能力的识别符和扩展记录的识别符，以及有关的 α 识别符
EF _{SMS} 短消息	6F3C	SMS	短消息	用于将短消息记录存放在 SIM 卡中
EF _{CCP} 能力配置参数	6F3D	CCP	能力配置参数	包括所需要的网络和承载能力的参数，以及当采用一个缩位拨号号码，固定拨号号码，MSISDN、最后拨号号码、服务拨号号码或禁止拨号方式等，建立呼叫时相关的 ME 配置
EF _{MSISDN} 电话号码	6F40	MSISDN	移动基站国际综合业务网号	存放用户的手机号
EF _{SMSP} 短信息参	6F42	SMSP	短消息业务参数	包括短信中心号码等信息

数				
EF _{SMSS} 短信息状态	6F43	SMSS	短消息状态	这个标识是用来控制流量的
EF _{LND} 最后拨号	6F44	LND	最后拨叫号码	存储最后拨叫号码
EF _{Ext1} 扩展文件 1	6F4A	EXT1	扩展文件 1	包括 AND, MSISDN 或 LND 的扩展数据
EF _{Ext2} 扩展文件 2	6F4B	EXT2	扩展文件 2	包含 FDN 的扩展数据

7、如何判断是否有 SD 卡？

通过如下方法：

```
Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)
```

如果返回 true 就是有 sdcard，如果返回 false 则没有。

8、dvm 的进程和 Linux 的进程，应用程序的进程是否为同一个概念？

dvm 指 dalvik 的虚拟机。每一个 Android 应用程序都拥有一个独立的 Dalvik 虚拟机实例,应用程序都在它自己的进程中运行。而每一个 dvm 都是在 Linux 中的一个进程，所以说可以近似认为是同一个概念。

什么是 android DVM: Dalvik 是 Google 公司自己设计用于 Android 平台的 Java 虚拟机,每一个 Dalvik 应用作为一个独立的 Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

Dalvik 和 Java 虚拟机的区别

1：Dalvik 主要是完成对象生命周期管理，堆栈管理，线程管理，安全和异常管理，以及垃圾回收等等重要功能。

2：Dalvik 负责进程隔离和线程管理，每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例，其代码在虚拟机的解释下得以执行。

3：不同于 Java 虚拟机运行 java 字节码，Dalvik 虚拟机运行的是其专有的文件格式 Dex

4: dex 文件格式可以减少整体文件尺寸，提高 I/O 操作的类查找速度。

5: odex 是为了在运行过程中进一步提高性能，对 dex 文件的进一步优化。

6：所有的 Android 应用的线程都对应一个 Linux 线程，虚拟机因而可以更多的依赖操作系统的线程调度和管理机制

7：有一个特殊的虚拟机进程 Zygote，他是虚拟机实例的孵化器。它在系统启动的时候就会产生，它会完成虚拟机的初始化，库的加载，预制类库和初始化的操作。如果系统需要一个新的虚拟机实例，它会迅速复制自身，以最快的数据提供给系统。对于一些只读的系统库，所有虚拟机实例都和 Zygote 共享一块内存区域。

9、Android 程序与 Java 程序的区别？

Android 程序用 android sdk 开发,java 程序用 javasdk 开发.

Android SDK 引用了大部分的 Java SDK，少数部分被 Android SDK 抛弃，比如说界面部分，java.awt swing package 除了 java.awt.font 被引用外，其他都被抛弃，在 Android 平台开发中不能使用。android sdk 添加工具 jar httpclient，pull opengl

10、启动应用后，改变系统语言，应用的语言会改变么？

这个一般是不会的，一般需要重启应用才能改变应用语言。但是对应应用来说如果做了国际化处理则支持如果没有处理那系统语言再更改也是无用的。

11、请介绍下 adb、ddms、aapt 的作用

adb 是 Android Debug Bridge ,Android 调试桥的意思，ddms 是 Dalvik Debug Monitor Service，dalvik 调试监视服务。aapt 即 Android Asset Packaging Tool，在 SDK 的 build-tools 目录下。该工具可以查看，创建，更新 ZIP 格式的文档附件(zip, jar, apk)。也可将资源文件编译成二进制文件，尽管我们没有直接使用过该工具，但是开发工具会使用这个工具打包 apk 文件构成一个 Android 应用程序。

Android 的主要调试工具是 adb(Android debugging bridge)，ddms 是一个在 adb 基础上的一个图形化工具。

adb，它是一个命令行工具。而 ddms 功能与 adb 相同，只是它有一个图形化界面。对不喜欢命令操作方式的人

来说是一个不错的选择。

12、ddms 和 traceview 的区别

ddms 原意是：davik debug monitor service。简单的说 ddms 是一个程序执行查看器，在里面可以看见线程和堆栈等信息，traceView 是程序性能分析器。traceview 是 ddms 中的一部分内容。

13、补充知识：TraceView 的使用

一、TraceView 简介

Traceview 是 Android 平台特有的数据采集和分析工具，它主要用于分析 Android 中应用程序的 hotspot（瓶颈）。Traceview 本身只是一个数据分析工具，而数据的采集则需要使用 Android SDK 中的 Debug 类或者利用 DDMS 工具。二者的用法如下：

开发者在一些关键代码段开始前调用 Android SDK 中 Debug 类的 startMethodTracing 函数，并在关键代码段结束前调用 stopMethodTracing 函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是 Java 线程）的函数执行情况，并将采集数据保存到/mnt/sdcard/下的一个文件中。开发者然后需要利用 SDK 中的 Traceview 工具来分析这些数据。

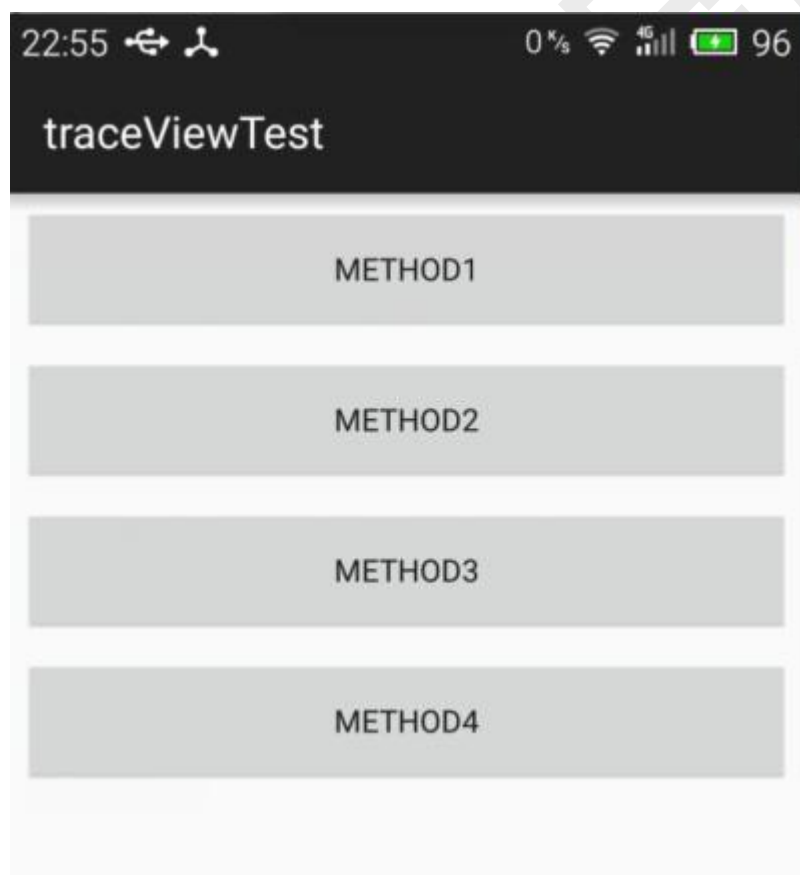
借助 Android SDK 中的 DDMS 工具。DDMS 可采集系统中某个正在运行的进程的函数调用信息。对开发者而言，此方法适用于没有目标应用源代码的情况。DDMS 工具中 Traceview 的使用如下图所示。



点击上图中所示按钮即可以采集目标进程的数据。当停止采集时，DDMS 会自动触发 Traceview 工具来浏览采集数据。

下面，我们通过一个示例程序介绍 Traceview 的使用。

实例程序如下图所示：界面有 4 个按钮，对应四个方法。



点击不同的方法会进行不同的耗时操作。

```
public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void method1(View view) {
        int result = jisuan();
        System.out.println(result);
    }
```

```
private int jisuan() {
    for (int i = 0; i < 10000; i++) {
        System.out.println(i);
    }
    return 1;
}

public void method2(View view) {
    SystemClock.sleep(2000);
}

public void method3(View view) {
    int sum = 0;
    for (int i = 0; i < 1000; i++) {
        sum += i;
    }
    System.out.println("sum=" + sum);
}

public void method4(View view) {
    Toast.makeText(this, "" + new Date(), 0).show();
}
}
```

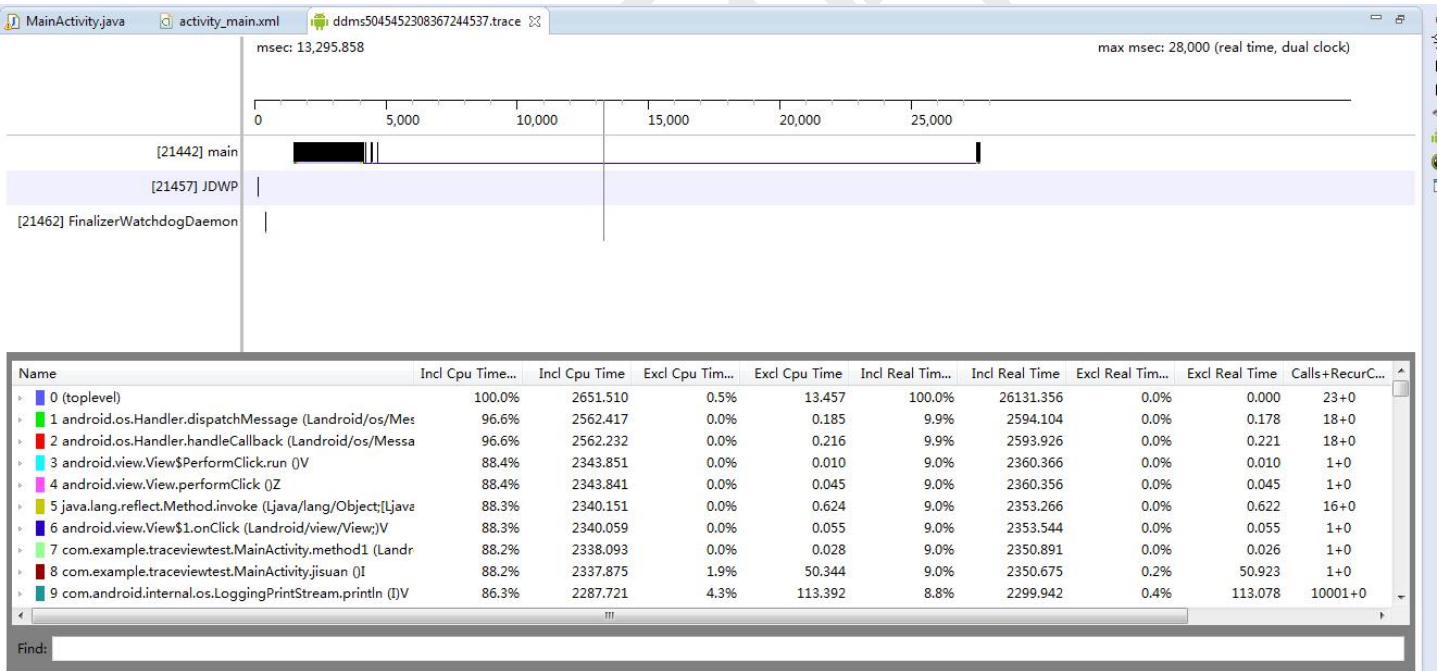
我们分别点击按钮一次，要求找出最耗时的方法。点击前通过 DDMS 启动 Start Method Profiling 按钮。



然后依次点击 4 个按钮，都执行后再次点击上图中红框中按钮，停止收集数据。

接下来我们开始对数据进行分析。

当我们停止收集数据的时候会出现如下分析图表。该图表分为 2 大部分，上面分不同的行，每一行代表一个线程的执行耗时情况。main 线程对应行的内容非常丰富，而其他线程在这段时间内干得工作则要少得多。图表的下半部分是具体的每个方法执行的时间情况。显示方法执行情况的前提是先选中某个线程。



我们主要是分析 main 线程。

上面方法指标参数所代表的意思如下：

列名	描述
Name	该线程运行过程中所调用的函数名

Incl Cpu Time	某函数占用的 CPU 时间，包含内部调用其它函数的 CPU 时间
Excl Cpu Time	某函数占用的 CPU 时间，但不含内部调用其它函数所占用的 CPU 时间
Incl Real Time	某函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间
Excl Real Time	某函数运行的真实时间（以毫秒为单位），不含调用其它函数所占用的真实时间
Call+Recur Calls/Total	某函数被调用次数以及递归调用占总调用次数的百分比
Cpu Time/Call	某函数调用 CPU 时间与调用次数的比。相当于该函数平均执行时间
Real Time/Call	同 CPU Time/Call 类似，只不过统计单位换成了真实时间

我们为了找到最耗时的操作，那么可以通过点击 Incl Cpu Time，让其按照时间的倒序排列。我点击后效果如下图：

Name	Incl Cpu Time...	Incl Cpu ...	Excl Cpu Tim...	Excl Cpu Time	Incl Real Tim...	Incl Real Time	Excl Real Tim...	Excl Real Time	Calls+RecurC...
0 (toplevel)	100.0%	2651.510	0.5%	13.457	100.0%	26131.356	0.0%	0.000	23+0
1 android.os.Handler.dispatchMessage (Landroid/os/Mes	96.6%	2562.417	0.0%	0.185	9.9%	2594.104	0.0%	0.178	18+0
2 android.os.Handler.handleCallback (Landroid/os/Messa	96.6%	2562.232	0.0%	0.216	9.9%	2593.926	0.0%	0.221	18+0
3 android.view.View\$PerformClick.run ()V	88.4%	2343.851	0.0%	0.010	9.0%	2360.366	0.0%	0.010	1+0
4 android.view.View.performClick ()Z	88.4%	2343.841	0.0%	0.045	9.0%	2360.356	0.0%	0.045	1+0
5 java.lang.reflect.Method.invoke (Ljava/lang/Object;[Jjav	88.3%	2340.151	0.0%	0.624	9.0%	2353.266	0.0%	0.622	16+0
6 android.view.View\$1.onClick (Landroid/view/View;)V	88.3%	2340.059	0.0%	0.055	9.0%	2353.544	0.0%	0.055	1+0
7 com.example.traceviewtest.MainActivity.method1 (Landr	88.2%	2338.093	0.0%	0.028	9.0%	2350.891	0.0%	0.026	1+0
8 com.example.traceviewtest.MainActivity.jisuan ()I	88.2%	2337.875	1.9%	50.344	9.0%	2350.675	0.2%	50.923	1+0
9 com.android.internal.os.LoggingPrintStream.println (I)V	86.3%	2287.721	4.3%	113.392	8.8%	2299.942	0.4%	113.078	10001+0
10 com.android.internal.os.LoggingPrintStream.flush (Z)V	52.1%	1382.477	8.9%	234.993	5.3%	1388.300	0.9%	235.907	10001+0
11 java.lang.StringBuilder.append (Ljava/lang/StringBuilc	29.9%	791.852	2.6%	69.036	3.1%	798.386	0.3%	69.077	10001+0
12 java.lang.IntegralToString.appendInt (Ljava/lang/Abstr	27.3%	722.816	2.6%	67.797	2.8%	729.309	0.3%	67.760	10001+0
13 java.lang.IntegralToString.convertInt (Ljava/lang/Abstre	24.7%	655.084	5.5%	146.132	2.5%	659.603	0.6%	147.898	10002+0
14 java.lang.StringBuilder.substring (I)Ljava/lang/String;	13.7%	363.810	2.6%	70.150	1.4%	365.069	0.3%	69.746	10001+0
15 java.lang.ThreadLocal.get ()Ljava/lang/Object;	11.6%	308.482	7.6%	201.578	1.2%	311.870	0.8%	203.362	9984+0

通过分析发现：method1 最耗时，耗时 2338 毫秒。

0 android.view.View\$1.onClick (Landroid/view/View;)V
7 com.example.traceviewtest.MainActivity.method1 (Landr
8 com.example.traceviewtest.MainActivity.jisuan ()I

那么有了上面的信息我们可以进入我们的 method1 方法查看分析我们的代码了。

14、Android 中数据存储方式有哪些？

- a) 文件存储
- b) xml , SharedPreferences
- c) SQLiteDatabase

- d) ContentProvider
- e) 网络

15、DVM 和 JVM 的区别？

- a) dvm 执行的是.dex 文件，而 jvm 执行的是.class。Android 工程编译后的所有.class 字节码会被 dex 工具抽取到一个.dex 文件中。
- b) dvm 是基于寄存器的虚拟机 而 jvm 执行是基于虚拟栈的虚拟机。寄存器存取速度比栈快的多，dvm 可以根据硬件实现最大的优化，比较适合移动设备。
- c) .class 文件存在很多的冗余信息，dex 工具会去除冗余信息，并把所有的.class 文件整合到.dex 文件中。减少了 I/O 操作，提高了类的查找速度。

16、谈一谈 Android 的安全机制

- 1、Android 是基于 Linux 内核的，因此 Linux 对文件权限的控制同样适用于 Android

在 Android 中每个应用都有自己的/data/data/包名 文件夹，该文件夹只能该应用访问，而其他应用则无权访问。

- 2、Android 的权限机制保护了用户的合法权益

如果我们的代码想拨打电话、发送短信、访问通信录、定位、访问 sdcard 等所有可能侵犯用户权益的行为都是必须要在 AndroidManifest.xml 中进行声明的，这样就给了用户一个知情权。

- 3、Android 的代码混淆保护了开发者的劳动成果

17、Android 的四大组件都需要在清单文件中注册吗？

Activity、Service、ContentProvider 如果要使用则必须在 AndroidManifest.xml 中进行注册，而 BroadcastReceiver 则有两种注册方式，静态注册和动态注册。其中静态注册就是指在 AndroidManifest.xml 中进行

注册，而动态注册时通过代码注册。

18、在 Android 中进程的级别有哪些？

- a) Foreground process
- b) Visible process
- c) Service process
- d) Background process
- e) Empty process

19、sp 频繁操作有什么后果？sp 能存多少数据？（上海 3 期学员提供）

Sp 的底层是由 xml 来实现的，操作 sp 的过程就是 xml 的序列化和解析的过程。Xml 是存储在磁盘上的，因此考虑到需要 I/O 速度问题，sp 不适宜频繁操作。同时序列化 xml 就是将内存中的数据写到 xml 文件中，由于 dvm 的内存是很有限的，因此单个 sp 文件不建议太大，具体多大是没有一个具体的要求的，但是我们知道 DVM 堆内存也就是 16M，因此数据大小肯定不能超过这个数字的。其实 sp 设置的目的是为了保存用户的偏好和配置信息的，因此不要保存太多的数据。

20、描述一下 Android 的系统架构（2017-2-23）

1. android 系统架构分从下往上为 linux 内核层、运行库、应用程序框架层、和应用程序层。
2. linuxkernel：负责硬件的驱动程序、网络、电源、系统安全以及内存管理等功能。
3. libraries 和 androidruntime：libraries：即 c/c++ 函数库部分，大多数都是开放源代码的函数库，例如 webkit，该函数库负责 android 网页浏览器的运行，例如标准的 c 函数库 libc、openssl、sqlite 等，当然也包括支持游戏开发 2dsgl 和 3dopengles，在多媒体方面有 mediaframework 框架来支持各种影音和图

形文件的播放与显示,例如 mpeg4、h.264、mp3、aac、amr、jpg 和 png 等众多多媒体文件格式。android 的 runtime 负责解释和执行生成的 dalvik 格式的字节码。

4. applicationframework (应用软件架构) , java 应用程序开发人员主要是使用该层封装好的 api 进行快速开发。
5. applications:该层是 java 的应用程序层, android 内置的 googlemaps、e-mail、即时通信工具、浏览器、mp3 播放器等处于该层, java 开发人员开发的程序也处于该层, 而且和内置的应用程序具有平等的位置, 可以调用内置的应用程序, 也可以替换内置的应用程序。

21、解释一下 Android 程序运行时权限与文件系统权限的区别？ (2017-2-23)

apk 程序是运行在虚拟机上的,对应的是 Android 独特的权限机制, 只有体现到文件系统上时才使用 linux 的权限设置。

- linux 文件系统上的权限如下表示

```
-rwxr-x--x system    system    4156 2010-04-30 16:13 test.apk
```

代表的是相应的用户/用户组及其他人对此文件的访问权限, 与此文件运行起来具有的权限完全不相关。比如上面的例子只能说明 system 用户拥有对此文件的读写执行权限; system 组的用户对此文件拥有读、执行权限; 其他人对此文件只具有执行权限。而 test.apk 运行起来后可以干哪些事情, 跟这个就不相关了。千万不要看 apk 文件系统上属于 system/system 用户及用户组, 或者 root/root 用户及用户组, 就认为 apk 具有 system 或 root 权限

- Android 的权限规则

a.Android 中的 apk 必须签名

b.基于 UserID 的进程级别的安全机制

c.默认 apk 生成的数据对外是不可见的

d.AndroidManifest.xml 中的显式权限声明

从 Android6.0 之后，Android 升级了权限机制，提出了动态权限的概念。

22、Android6.0 的权限机制 (2017-2-23)

1) 新权限思想

Android 的权限系统一直是首要的安全概念，因为这些权限只在安装的时候被询问一次。一旦安装了，app 可以在用户毫不知晓的情况下访问权限内的所有东西，而且一般用户安装的时候很少会去仔细看权限列表，更不会去深入了解这些权限可能带来的相关危害。

但是在 Android 6.0 Marshmallow 版本之后，系统不会在软件安装的时候就赋予该 app 所有其申请的权限，对于一些危险级别的权限，app 需要在运行时一个一个询问用户授予权限。

2) 对旧版本 App 的兼容

那么问题来了，是不是所有以前发布的 app 都会出现问题呢？答案是不会，只有那些 targetSdkVersion 设置为 23 及以上的应用才会出现异常，在使用危险权限的时候系统必须要获得用户的同意才能使用，要不然应用就会崩溃，出现类似下面的错误。

```
java.lang.SecurityException: Permission Denial...
```

所以 targetSdkVersion 如果没有设置为 23 版本或者以上，系统还是会使用旧规则：在安装的时候赋予该 app 所申请的所有权限。所以 app 当然可以和以前一样正常使用了，但是还有一点需要注意的是 6.0 的系统里面，用户可以手动将该 app 的权限关闭，在 App info 里面 Permissions 下边，可以关闭某个权限。如果以前的老应用申请的权限被用户手动关闭了，不会抛出异常，不会崩溃，只不过调用那些被用户禁止权限的 api 接口返回值都为 null 或者 0，所以我们只需要做一下判空操作就可以了，这是需要注意的。

3) 普通权限和危险权限列表

现在对于新版本的权限变更应该有了基本的认识，那么，是不是所有权限都需要去进行特殊处理呢？当然不是，只有那些危险级别的权限才需要，可参考官网。

<http://developer.android.com/training/permissions/requesting.html>

<http://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>

所以仔细去看看自己的 app，对照列表，如果有需要申请其中的一个权限，就需要进行特殊操作。还有一个比较人性的地方就是如果同一组的任何一个权限被授权了，其他权限也自动被授权。例如，一旦 WRITE_EXTERNAL_STORAGE 被授权了，app 也有 READ_EXTERNAL_STORAGE 权限了。

如下表格展示了危险级别的权限清单。

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none">• READ_CALENDAR• WRITE_CALENDAR
CAMERA	<ul style="list-style-type: none">• CAMERA
CONTACTS	<ul style="list-style-type: none">• READ_CONTACTS• WRITE_CONTACTS• GET_ACCOUNTS
LOCATION	<ul style="list-style-type: none">• ACCESS_FINE_LOCATION• ACCESS_COARSE_LOCATION
MICROPHONE	<ul style="list-style-type: none">• RECORD_AUDIO
PHONE	<ul style="list-style-type: none">• READ_PHONE_STATE• CALL_PHONE• READ_CALL_LOG• WRITE_CALL_LOG• ADD_VOICEMAIL• USE_SIP• PROCESS_OUTGOING_CALLS
SENSORS	<ul style="list-style-type: none">• BODY_SENSORS

SMS	<ul style="list-style-type: none">• SEND_SMS• RECEIVE_SMS• READ_SMS• RECEIVE_WAP_PUSH• RECEIVE_MMS
STORAGE	<ul style="list-style-type: none">• READ_EXTERNAL_STORAGE• WRITE_EXTERNAL_STORAGE

4) 支持 Marshmallow 新版本权限机制

在 Android M 的 api 中，我们可以通过 `checkSelfPermission` 检测软件是否有某一项权限，以及使用 `requestPermissions` 去请求一组权限。

```
1. int hasWriteContactsPermission =
2. checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE);
3. if (hasWriteContactsPermission != PackageManager.PERMISSION_GRANTED) {
4.     requestPermissions(new String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE},
5.         CODE_FOR_WRITE_PERMISSION);
6.     return;
7. }
```

以上的代码块展示了检测软件是否有写文件的权限，如果没有写文件的权限，则通过 `requestPermissions` 去向用户发起请求权限的流程。向用户发起请求之后，请求完成，会有相对应的回调方法，通知软件用户是否授予了权限。

通过在 Activity 或者 Fragment 中重写 `onRequestPermissionsResult` 方法。

```
1. @Override
2. public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)
3. {
4.     if (requestCode == CODE_FOR_WRITE_PERMISSION) {
5.         if (permissions[0].equals(Manifest.permission.WRITE_EXTERNAL_STORAGE)
6.             && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
7.             // 用户同意写文件
8.         } else {
9.             // 用户不同意，自行处理即可
10.        }
```

```
10.    }  
11. }
```

5) 使用兼容库

以上的代码在 6.0 版本上使用没有问题，但是在之前就有问题了，最简单粗暴的解决方法可能就是利用 `Build.VERSION.SDK_INT >= 23` 这个判断语句来判断了，方便的是 SDK 23 的 v4 包加入了专门类进行相关的处理：

`ContextCompat.checkSelfPermission()` 被授权函数返回 `PERMISSION_GRANTED`，否则返回 `PERMISSION_DENIED`，在所有版本都是如此。

`ActivityCompat.requestPermissions()`这个方法在 6.0 之前版本调用，`OnRequestPermissionsResultCallback` 直接被调用，带着正确的 `PERMISSION_GRANTED` 或者 `PERMISSION_DENIED`。

`ActivityCompat.shouldShowRequestPermissionRationale()`在 6.0 之前版本调用，永远返回 `false`。

```
1. // 使用兼容库就无需判断系统版本  
2. int hasWriteContactsPermission =  
3. ContextCompat.checkSelfPermission(getApplication(),  
4. Manifest.permission.WRITE_EXTERNAL_STORAGE);  
5. if (hasWriteContactsPermission == PackageManager.PERMISSION_GRANTED) {  
6.  
7. } // 需要弹出 dialog 让用户手动赋予权限  
8. else {  
9.     ActivityCompat.requestPermissions(Activity.this,  
10. new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, CODE_FOR_WRITE_PERMISSION);  
11. }
```

`onRequestPermissionsResult` 函数不变。后两个方法，我们也可以在 `Fragment` 中使用，用 v13 兼容包：`FragmentCompat.requestPermissions()` 和 `FragmentCompat.shouldShowRequestPermissionRationale()` 和 `activity` 效果一样。

23、AndroidManifest.xml 中的 `targetSdk` 设置有什么作用？（2017-2-24）

用于指定 Android 应用中所需要使用的 SDK 的版本，比如我们的应用必须运行于 Android 4.1 以上版本的系统 SDK 之上，那么就需要指定应用支持最小的 SDK 版本数为 16；当然，每个 SDK 版本都会有指定的整数值与之对应，

比如我们最常用的 Android 2.3 的版本数是 11。当然，除了可以指定最低版本之外，标签还可以指定最高版本和目标版本，语法范例如下。

```
< android:targetSdkVersion="integer"  
    android:maxSdkVersion="integer" />
```

附带一些 application 节点的介绍

第一层：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" //命名空间  
    package="com.woody.test" //指定本应用内 java 主程序包的包名  
    android:sharedUserId="string" //数据权限，Android 给每个 APK 分配唯一的 UserID，禁止共享数据  
    android:sharedUserLabel="string resource"//共享的用户名，设置 sharedUserId 才有意义  
    android:versionCode="integer"//设备识别版本  
    android:versionName="string"//这个名称是给用户看的  
    android:installLocation=["auto" | "internalOnly" | "preferExternal"] >  
</manifest>
```

installLocation

安装参数，是 Android 2.2 中的一个新特性，installLocation 有三个值可以选择：internalOnly、auto、preferExternal。选择 preferExternal，系统会优先考虑将 APK 安装到 SD 卡上（当然最终用户可以选择为内部 ROM 存储上，如果 SD 存储已满，也会安装到内部存储上）。

选择 auto，系统将会根据存储空间自己去适应。

选择 internalOnly 是指必须安装到内部才能运行。

第二层：

```
<application android:allowClearUserData=["true" | "false"]//用户是否能选择自行清除数据  
    android:allowTaskReparenting=["true" | "false"]//是否允许 activity 更换从属的任务，比如从短  
    信息任务切换到浏览器任务  
    android:backupAgent="string"  
    android:debuggable=["true" | "false"] //APP 在手机上可以被调试  
    android:description="string resource" //  
    android:enabled=["true" | "false"]  
    android:hasCode=["true" | "false"]  
    android:icon="drawable resource" //APP 的图标  
    android:killAfterRestore=["true" | "false"] //是否复位需要重启  
    android:label="string resource"  
    android:manageSpaceActivity="string" //让应用手动管理应用的数据目录
```

```
android:name="string"    //Application 子类的全名
android:permission="string"
android:persistent=["true" | "false"]
android:process="string" //程序运行的进程名
android:restoreAnyVersion=["true" | "false"]
android:taskAffinity="string" //拥有相同的 affinity 的 Activity 理论上属于相同的 Task，应用程序
                                默认的 affinity 的名字是元素中设定的 package 名

    android:theme="resource or theme" > //定义了一个默认的主题风格给所有的 activity
</application>
```

A、android:backupAgent

这也是 Android2.2 中的一个新特性，设置该 APP 的备份，属性值应该是一个完整的类名，如 com.project.TestCase，此属性并没有默认值，并且类名必须得指定(就是个备份工具，将数据备份到云端的操作)

B、android:description/android:label

此两个属性都是为许可提供的，均为字符串资源，当用户去看许可列表(android:label)或者某个许可的详细信息(android:description)时，这些字符串资源就可以显示给用户。label 应当尽量简短，之需要告知用户该许可是在保护什么功能就行。而 description 可以用于具体描述获取该许可的程序可以做哪些事情，实际上让用户可以知道如果他们同意程序获取该权限的话，该程序可以做什么。我们通常用两句话来描述许可，第一句描述该许可，第二句警告用户如果批准该权限会可能有什么不好的事情发生

C、android:enabled

Android 系统是否能够实例化该应用程序的组件，如果为 true，每个组件的 enabled 属性决定那个组件是否可以被 enabled。如果为 false，它覆盖组件指定的值；所有组件都是 disabled。

D、android:hasCode('true' or 'false')

表示此 APP 是否包含任何的代码，默认为 true，若为 false，则系统在运行组件时，不会去尝试加载任何的 APP 代码一个应用程序自身不会含有任何的代码，除非内置组件类，比如 Activity 类，此类使用了 AliasActivity 类，当然这是个罕见的现象

(在 Android2.3 可以用标准 C 来开发应用程序，可在 androidManifest.xml 中将此属性设置为 false,因为这个 APP 本身已经不含有任何的 JAVA 代码了)

E、android:permission

设置许可名，这个属性若在上定义的话，是一个给应用程序的所有组件设置许可的便捷方式，当然它是被各组件设置的许可名所覆盖的

F、android:restoreAnyVersion

同样也是 android2.2 的一个新特性，用来表明应用是否准备尝试恢复所有的备份，甚至该备份是比当前设备上更要新的版本，默认是 false

二、Activity

1、什么是 Activity?

四大组件之一,通常一个用户交互界面对应一个 activity。activity 是 Context 的子类,同时实现了 window.callback 和 KeyEvent.callback, 可以处理与窗体用户交互的事件。

常见的 Activity 类型有 FragmentActivity, ListActivity, TabActivity 等。

如果界面有共同的特点或者功能的时候,还会自己定义一个 BaseActivity。

2、请描述一下 Activity 生命周期

Activity 从创建到销毁有多种状态，从一种状态到另一种状态时会激发相应的回调方法，这些回调方法包括：

onCreate onStart onResume onPause onStop onDestroy

其实这些方法都是两两对应的，onCreate 创建与 onDestroy 销毁；

onStart 可见与 onStop 不可见；onResume 可编辑（即焦点）与 onPause；

这 6 个方法是相对应的，那么就只剩下一个 onRestart 方法了，这个方法在什么时候调用呢？

答案就是：在 Activity 被 onStop 后，但是没有被 onDestroy，在再次启动此 Activity 时就调用 onRestart（而不再调用 onCreate）方法；

如果被 onDestroy 了，则是调用 onCreate 方法。

3、Activity 的状态都有哪些？

- a) foreground activity
- b) visible activity
- c) background activity
- d) empty process

4、如何保存 Activity 的状态？

Activity 的状态通常情况下系统会自动保存的，只有当我们需要保存额外的数据时才需要使用到这样的功能。

一般来说，调用 onPause()和 onStop()方法后的 activity 实例仍然存在于内存中，activity 的所有信息和状态数据不会消失，当 activity 重新回到前台之后，所有的改变都会得到保留。

但是当系统内存不足时，调用 onPause()和 onStop()方法后的 activity 可能会被系统摧毁，此时内存中就不会存有该 activity 的实例对象了。如果之后这个 activity 重新回到前台，之前所作的改变就会消失。为了避免此种情况的发生，我们可以覆写 onSaveInstanceState()方法。onSaveInstanceState()方法接受一个 Bundle 类型的参数，开发者可以将状态数据存储到这个 Bundle 对象中，这样即使 activity 被系统摧毁，当用户重新启动这个 activity 而调用它的 onCreate()方法时，上述的 Bundle 对象会作为实参传递给 onCreate()方法，开发者可以从 Bundle 对象中取出保存的数据，然后利用这些数据将 activity 恢复到被摧毁之前的状态。

需要注意的是，onSaveInstanceState()方法并不是一定会被调用的，因为有些场景是不需要保存状态数据的。比如用户按下 BACK 键退出 activity 时，用户显然想要关闭这个 activity，此时是没有必要保存数据以供下次恢复的，也

就是 `onSaveInstanceState()` 方法不会被调用。如果调用 `onSaveInstanceState()` 方法，调用将发生在 `onPause()` 或 `onStop()` 方法之前。

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    // TODO Auto-generated method stub
    super.onSaveInstanceState(outState);
}
```

5、两个 Activity 之间跳转时必然会执行的是哪几个方法？（重要）

一般情况下比如说有两个 activity, 分别叫 A,B, 当在 A 里面激活 B 组件的时候, A 会调用 `onPause()` 方法, 然后 B 调用 `onCreate()`, `onStart()`, `onResume()`。

这个时候 B 覆盖了窗体, A 会调用 `onStop()` 方法。如果 B 是个透明的, 或者是对话框的样式, 就不会调用 A 的 `onStop()` 方法。

如下图，打开一个 MainActivity，然后点击 MainActivity 中的按钮跳转到 SecondActivity 的日志。

Text
MainActivity onCreate
MainActivity onStart
MainActivity onResume
MainActivity onPause
SecondActivity onCreate
SecondActivity onStart
SecondActivity onResume
MainActivity onStop

6、横竖屏切换时 Activity 的生命周期

此时的生命周期跟清单文件里的配置有关系。

1、不设置 Activity 的 `android:configChanges` 时，切屏会重新调用各个生命周期

默认首先销毁当前 activity，然后重新加载。

如下图，当横竖屏切换时先执行 onPause/onStop 方法

```
Text
SecondActivity onPause
SecondActivity onStop
SecondActivity onCreate
SecondActivity onStart
SecondActivity onResume
```

2、设置 Activity 的 android:configChanges="orientation|keyboardHidden|screenSize"时，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法。

通常在游戏开发，屏幕的朝向都是写死的。

7、如何将一个 Activity 设置成窗口的样式？

只需要给我们的 Activity 配置如下属性即可。

```
android:theme="@android:style/Theme.Dialog"
```

8、如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

- 1、通常情况用户退出一个 Activity 只需按返回键，我们写代码想退出 activity 直接调用 finish()方法就行。
- 2、记录打开的 Activity：

每打开一个 Activity，就记录下来。在需要退出时，关闭每一个 Activity 即可。

```
//伪代码
List<Activity> lists ;// 在 application 全局的变量里面
lists = new ArrayList<Activity>();
lists.add(this);
for(Activity activity: lists)
{
    activity.finish();
}
lists.remove(this);
```

3、发送特定广播：

在需要结束应用时，发送一个特定的广播，每个 Activity 收到广播后，关闭即可。

//给某个 activity 注册接受接受广播的意图

```
registerReceiver(receiver, filter)
```

//如果过接受到的是 关闭 activity 的广播 就调用 finish()方法 把当前的 activity finish()掉

4、递归退出

在打开新的 Activity 时使用 startActivityForResult，然后自己加标志，在 onActivityResult 中处理，递归关闭。

5、其实 也可以通过 intent 的 flag 来实现 intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)激活一个新的 activity。此时如果该任务栈中已经有该 Activity，那么系统会把这个 Activity 上面的所有 Activity 干掉。其实相当于给 Activity 配置的启动模式为 SingleTop。

9、请描述一下 Activity 的启动模式都有哪些以及各自的特点

启动模式(launchMode)在多个 Activity 跳转的过程中扮演着重要的角色，它可以决定是否生成新的 Activity 实例，是否重用已存在的 Activity 实例，是否和其他 Activity 实例公用一个 task 里。这里简单介绍一下 task 的概念，task 是一个具有栈结构的对象，一个 task 可以管理多个 Activity，启动一个应用，也就创建一个与之对应的 task。

Activity 一共有以下四种 launchMode：

- ◆ 1.standard
- ◆ 2.singleTop
- ◆ 3.singleTask
- ◆ 4.singleInstance

我们可以在 AndroidManifest.xml 配置<activity>的 android:launchMode 属性为以上四种之一即可。

下面我们结合实例——介绍这四种 launchMode：

8.1 standard

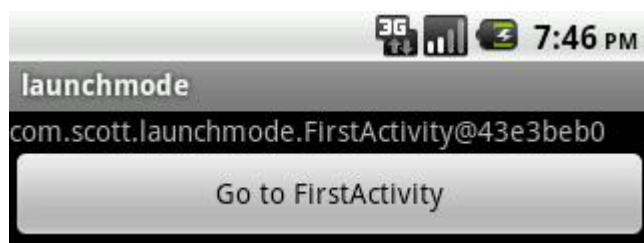
standard 模式是默认的启动模式，不用为<activity>配置 android:launchMode 属性即可，当然也可以指定值为 standard。

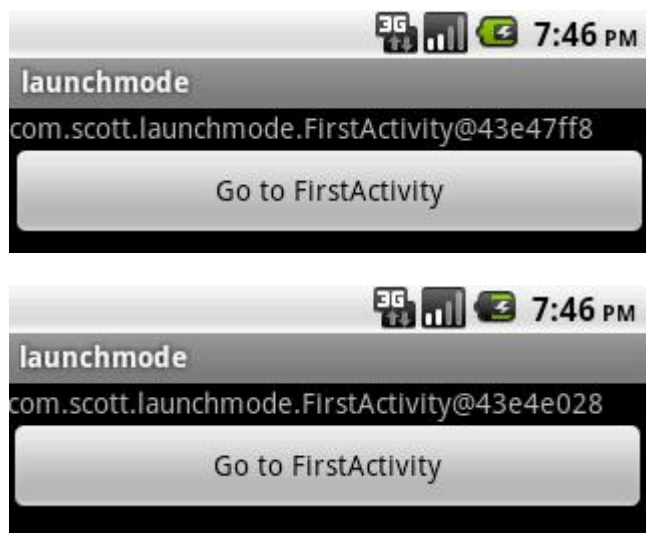
我们将创建一个 Activity，命名为 FirstActivity，来演示一下标准的启动模式。FirstActivity 代码如下：

```
public class FirstActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first);
        TextView textView = (TextView) findViewById(R.id.tv);
        textView.setText(this.toString());
        Button button = (Button) findViewById(R.id.bt);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(FirstActivity.this,
                FirstActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

FirstActivity 界面中的 TextView 用于显示当前 Activity 实例的序列号，Button 用于跳转到下一个 FirstActivity 界面。

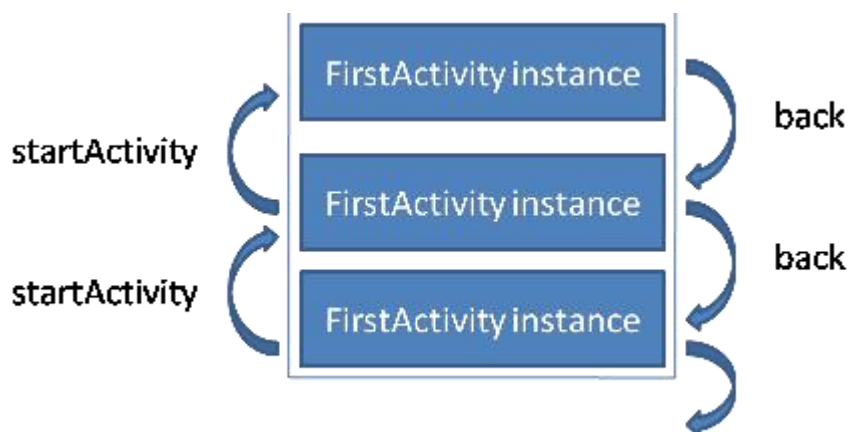
然后我们连续点击几次按钮，将会出现下面的现象：





我们注意到都是 FirstActivity 的实例，但序列号不同，并且我们需要连续按后退键两次，才能回到第一个 FirstActivity。

standard 模式的原理如下图所示：

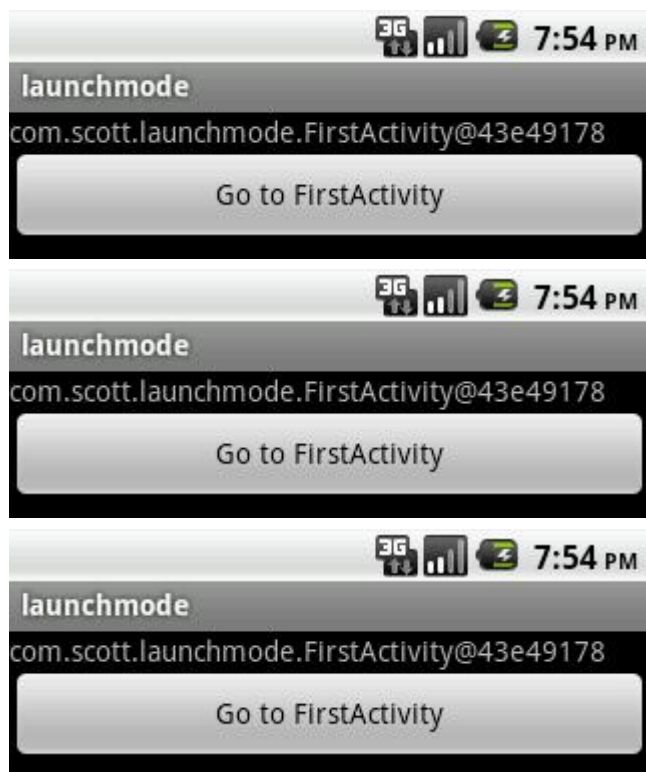


如图所示，每次跳转系统都会在新的 task 中生成一个新的 FirstActivity 实例，并且放于栈结构的顶部，当我们按下后退键时，才能看到原来的 FirstActivity 实例。

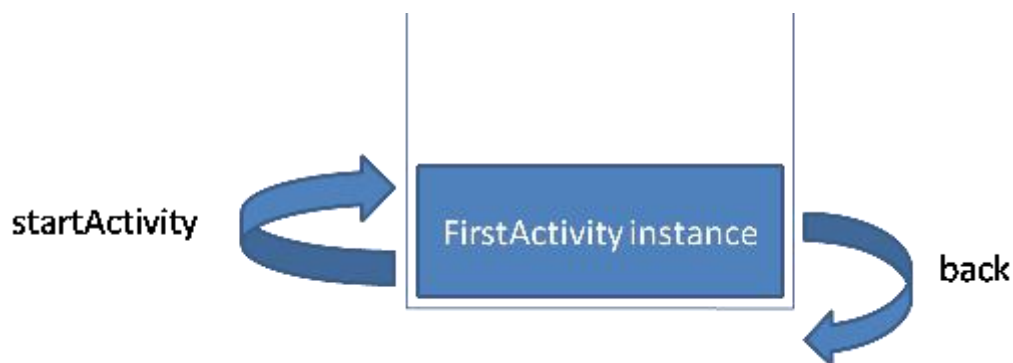
这就是 standard 启动模式，不管有没有已存在的实例，都生成新的实例。

8.2 singleTop

我们在上面的基础上为 <activity> 指定属性 **android:launchMode="singleTop"**，系统就会按照 singleTop 启动模式处理跳转行为。我们重复上面几个动作，将会出现下面的现象：



我们看到这个结果跟 standard 有所不同，三个序列号是相同的，也就是说使用的都是同一个 FirstActivity 实例；如果按一下后退键，程序立即退出，说明当前栈结构中只有一个 Activity 实例。singleTop 模式的原理如下图所示：



正如下图所示，跳转时系统会先在栈结构中寻找是否有一个 FirstActivity 实例正位于栈顶，如果有则不再生成新的，而是直接使用。也许朋友们会有疑问，我只看到栈内只有一个 Activity，如果是多个 Activity 怎么办，如果不是在栈顶会如何？我们接下来再通过一个示例来证实一下大家的疑问。

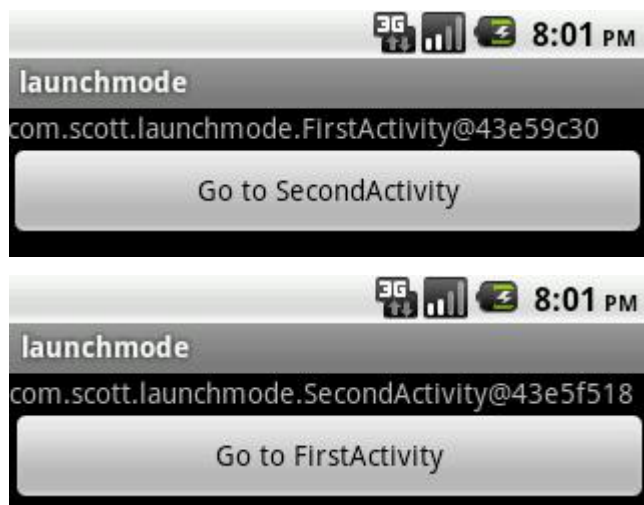
我们再新建一个 Activity 命名为 SecondActivity，如下：

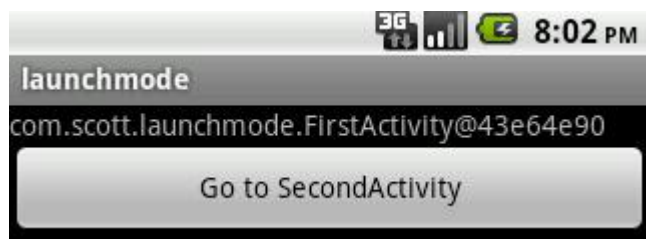
```
public class SecondActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        TextView textView = (TextView) findViewById(R.id.tv);
        textView.setText(this.toString());
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(SecondActivity.this,
                FirstActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

然后将之前的 FirstActivity 跳转代码改为：

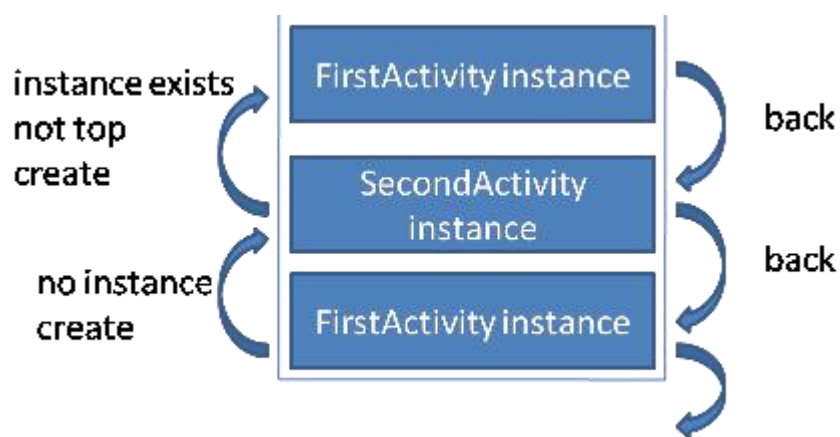
```
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
startActivity(intent);
```

这时候，FirstActivity 会跳转到 SecondActivity，SecondActivity 又会跳转到 FirstActivity。演示结果如下：





我们看到，两个 FirstActivity 的序号是不同的，证明从 SecondActivity 跳转到 FirstActivity 时生成了新的 FirstActivity 实例。原理图如下：

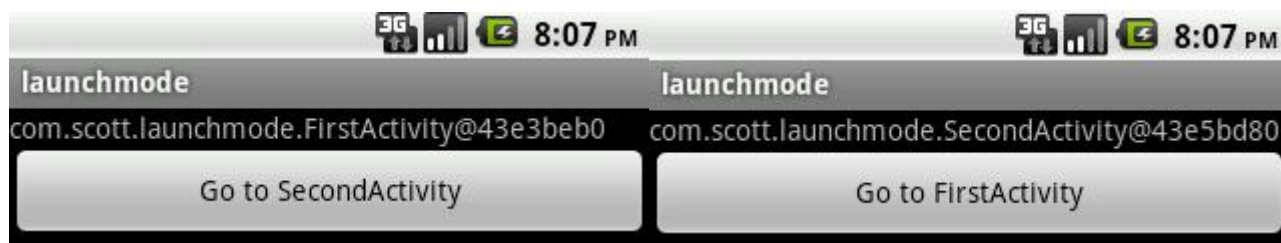


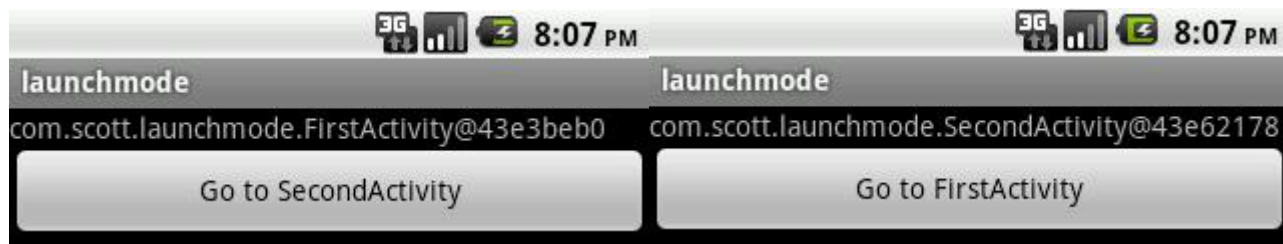
我们看到，当从 SecondActivity 跳转到 FirstActivity 时，系统发现存在有 FirstActivity 实例,但不是位于栈顶，于是重新生成一个实例。

这就是 singleTop 启动模式，如果发现有对应的 Activity 实例正位于栈顶，则重复利用，不再生成新的实例。

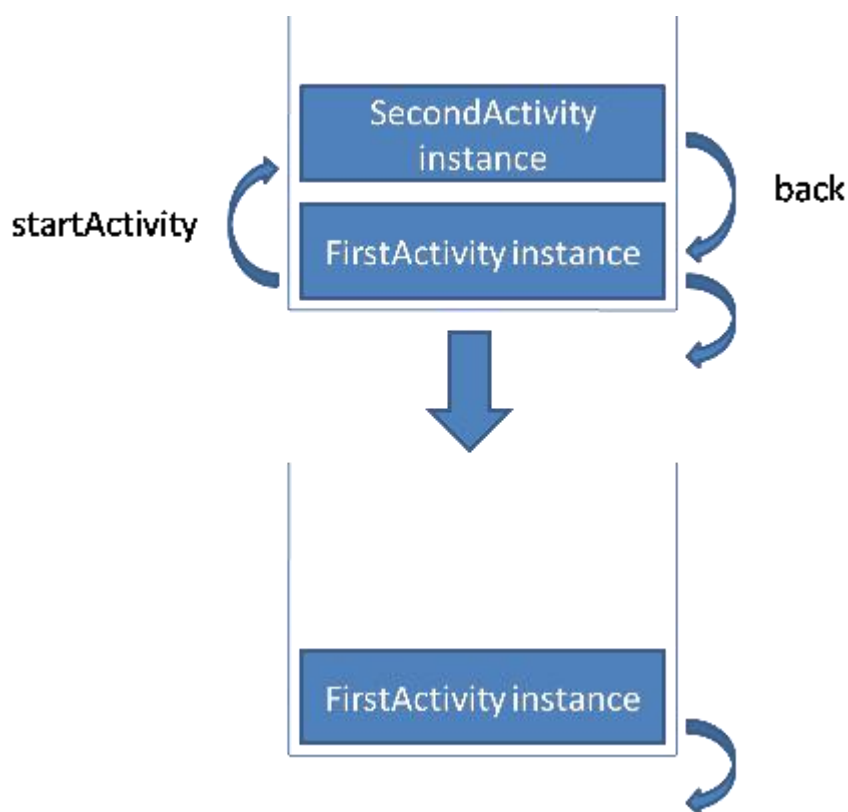
8.3 singleTask

在上面的基础上我们修改 FirstActivity 的属性 **android:launchMode="singleTask"**。演示的结果如下：





我们注意到，在上面的过程中，FirstActivity 的序列号是不变的，SecondActivity 的序列号却不是唯一的，说明从 SecondActivity 跳转到 FirstActivity 时，没有生成新的实例，但是从 FirstActivity 跳转到 SecondActivity 时生成了新的实例。singleTask 模式的原理图如下图所示：



在图中的下半部分是 SecondActivity 跳转到 FirstActivity 后的栈结构变化的结果，我们注意到，SecondActivity 消失了，没错，在这个跳转过程中系统发现有存在的 FirstActivity 实例，于是不再生成新的实例，而是将 FirstActivity 之上的 Activity 实例统统出栈，将 FirstActivity 变为栈顶对象，显示到幕前。也许朋友们有疑问，如果将 SecondActivity 也设置为 singleTask 模式，那么 SecondActivity 实例是不是可以唯一呢？在我们这个示例中是不可能的，因为每次从 SecondActivity 跳转到 FirstActivity 时，SecondActivity 实例都被迫出栈，下次等 FirstActivity 跳转到

SecondActivity 时，找不到存在的 SecondActivity 实例，于是必须生成新的实例。但是如果有 ThirdActivity，让 SecondActivity 和 ThirdActivity 互相跳转，那么 SecondActivity 实例就可以保证唯一。

这就是 singleTask 模式，如果有对应的 Activity 实例，则使此 Activity 实例之上的其他 Activity 实例统统出栈吗，使此 Activity 实例成为栈顶对象，显示到幕前。

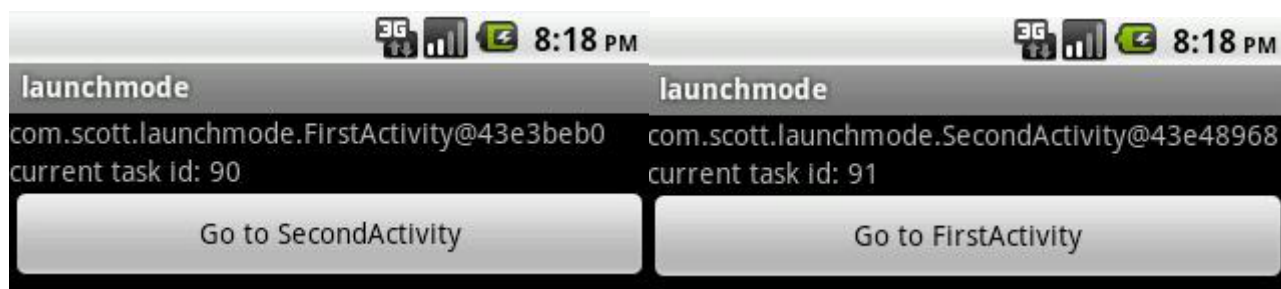
8.4 singleInstance

这种启动模式比较特殊，因为它会启用一个新的栈结构，将 Activity 放置于这个新的栈结构中，并保证不再有其他 Activity 实例进入。

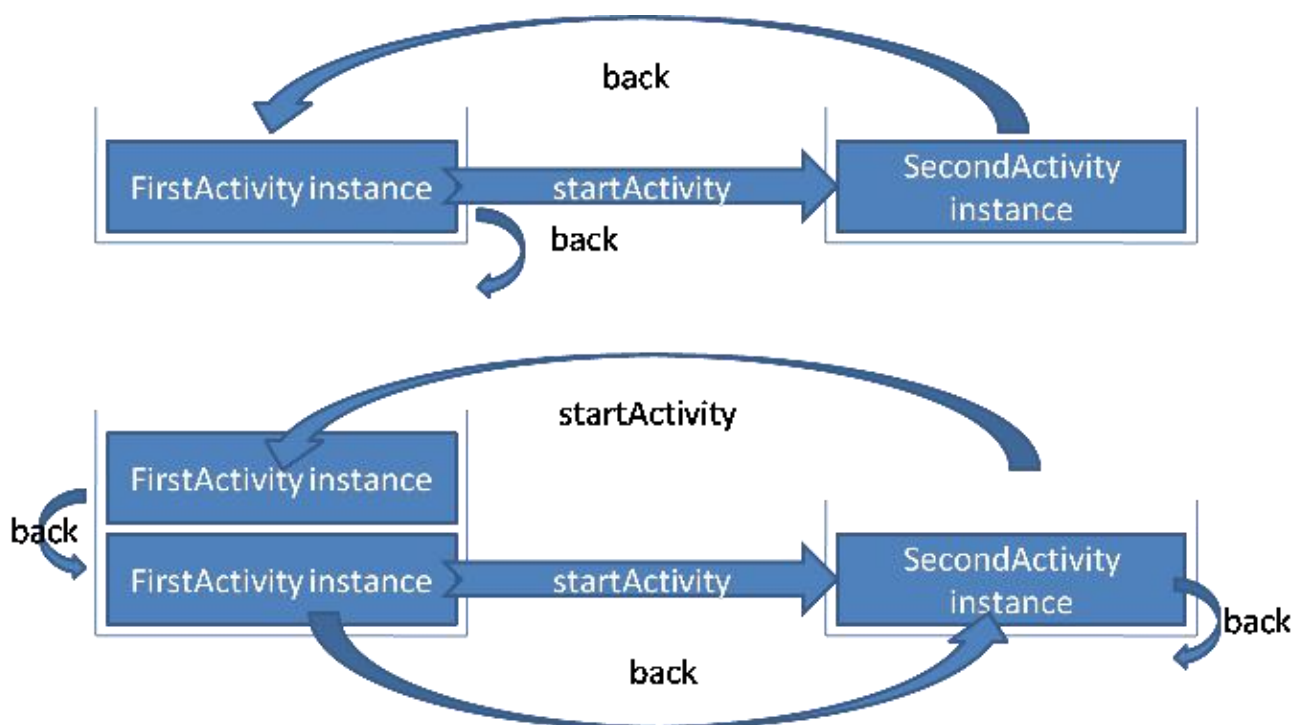
我们修改 FirstActivity 的 launchMode="standard"，SecondActivity 的 launchMode="singleInstance"，由于涉及到了多个栈结构，我们需要在每个 Activity 中显示当前栈结构的 id，所以我们为每个 Activity 添加如下代码：

```
TextView taskIdView = (TextView) findViewById(R.id.taskIdView);  
taskIdView.setText("current task id: " + this.getTaskId());
```

然后再演示一下这个流程：

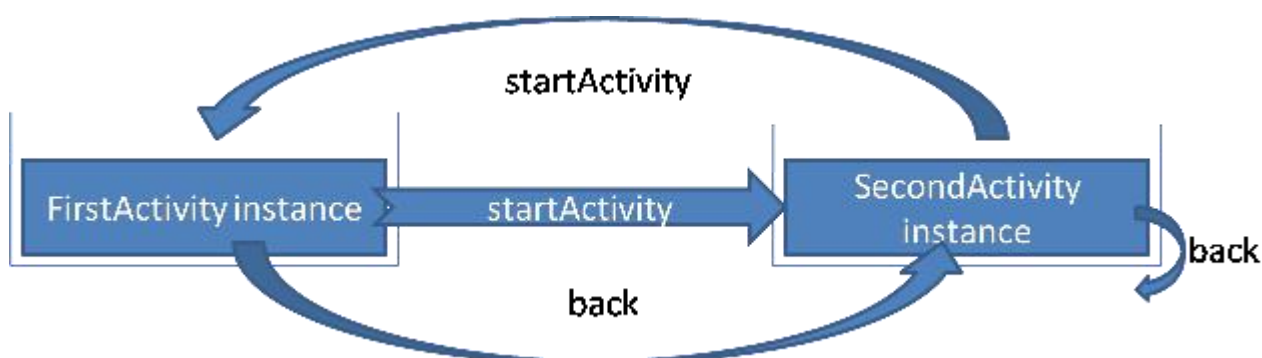


我们发现这两个 Activity 实例分别被放置在不同的栈结构中，关于 singleInstance 的原理图如下



我们看到从 FirstActivity 跳转到 SecondActivity 时，重新启用了一个新的栈结构，来放置 SecondActivity 实例，然后按下后退键，再次回到原始栈结构；图中下半部分显示的在 SecondActivity 中再次跳转到 FirstActivity，这个时候系统会在原始栈结构中生成一个 FirstActivity 实例，然后回退两次，注意，并没有退出，而是回到了 SecondActivity，为什么呢？是因为从 SecondActivity 跳转到 FirstActivity 的时候，我们的起点变成了 SecondActivity 实例所在的栈结构，这样一来，我们需要“回归”到这个栈结构。

如果我们修改 FirstActivity 的 launchMode 值为 singleTop、singleTask、singleInstance 中的任意一个，流程将会如图所示：



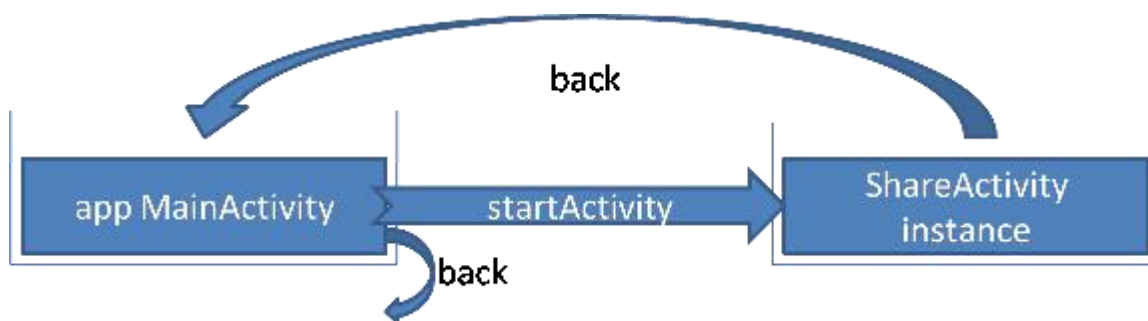
singleInstance 启动模式可能是最复杂的一种模式，为了帮助大家理解，我举一个例子，假如我们有一个 share 应用，其中的 ShareActivity 是入口 Activity，也是可供其他应用调用的 Activity，我们把这个 Activity 的启动模式设置为 singleInstance，然后在其他应用中调用。我们编辑 ShareActivity 的配置：

```
<activity
    android:name=".ShareActivity"
    android:launchMode="singleInstance" >
    <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
    <action android:name="android.intent.action.SINGLE_INSTANCE_SHARE" />
    <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

然后我们在其他应用中这样启动该 Activity：

```
Intent intent = new Intent("android.intent.action.SINGLE_INSTANCE_SHARE");
startActivity(intent);
```

当我们打开 ShareActivity 后再按后退键回到原来界面时，ShareActivity 作为一个独立的个体存在，如果这时我们打开 share 应用，无需创建新的 ShareActivity 实例即可看到结果，因为系统会自动查找，存在则直接利用。大家可以在 ShareActivity 中打印一下 taskId，看看效果。关于这个过程，原理图如下：



10、一个启动模式为 singleTop 的 activity ,如果再试图启动会怎样？ 面试官想问的是 onNewIntent() (2015-11-24)

Activity 有一个 onNewIntent(Intent intent)回调方法，该方法我们几乎很少使用，导致已经将其忽略掉。该方法的官方解释如下：

```
This is called for activities that set launchMode to "singleTop" in their package, or if a client used the Intent.FLAG_ACTIVITY_SINGLE_TOP flag when calling startActivity. In either case, when the activity is re-launched while at the top of the activity stack instead of a new instance of the activity being started, onNewIntent() will be called on the existing instance with the Intent that was used to re-launch it.
```

```
An activity will always be paused before receiving a new intent, so you can count on onResume being called after this method.
```

```
Note that getIntent still returns the original Intent. You can use setIntent to update it to this new Intent.
```

上文大概意思如下：

该方法被启动模式设置为“singleTop”的 Activity 回调，或者当通过设置 Intent.FLAG_ACTIVITY_SINGLE_TOP 的 Intent 启动 Activity 时被回调。在任何情况下，只要当栈顶的 Activity 被重新启动时没有重新创建一个新的 Activity 实例而是依然使用该 Activity 对象，那么 onNewIntent(Intent)方法就会被回调。

当一个 Activity 接收到新 Intent 的时候会处于暂停状态，因此你可以统计到 onResume()方法会被再次执行，当然这个执行是在 onNewIntent 之后的。

注意：如果我们在 Activity 中调用了 getIntent () 方法，那么返回的 Intent 对象还是老的 Intent (也就是第一次启动该 Activity 时的传入的 Intent 对象)，但是如果想让 getIntent () 返回最新的 Intent，那么我们可以通过 setIntent(Intent)方法设置。

11、两个 Activity 之间传递数据，除了 intent，广播接收者，content provider 还有啥方式？(2017-2-23)

1) 利用 static 静态数据，public static 成员变量

2) 利用外部存储的传输，

例如 File 文件存储

SharedPreferences 首选项

Sqlite 数据库

3) 如果不跨进程，可以使用 EventBus

4) 使用剪切板传递数据

a) 设置数据

```
1. ClipboardManager manager = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
2. String name = "给我死过去";
3. manager.setText(name);
4. Intent intent = new Intent(MainActivity.this, SencedActivity.class);
5. startActivity(intent);
```

b) 获取数据

```
6. ClipboardManager manager = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
7. String name = "给我死过去";
8. manager.setText(name);
9. Intent intent = new Intent(MainActivity.this, SencedActivity.class);
10. startActivity(intent);
```

12、怎样在两个 Activity 之间传递一张图片 (2017-2-23)

- a) Intent 可以传递基本数据类型、Uri 和序列化对象
- b) Bitmap 对象实现了 Parcelable 序列化接口，但是不建议放到 Intent 里传递。因为 Pancelable 对象序列化过程是将对象 A 的属性暂存一份到内存里，反序列化时再使用暂存的数据，创建一个属性完全相同的对象 B。
- c) 对于 Bitmap 而言，就是把图片的二进制数据 0 复制一份到内存里构成二进制数据 1，反序列化时根据二进制数据 1 创建 Bitmap 对象，此时会生成二进制数据 2。也就是同一个图片的数据在内存里存放了三份。
- d) 也就是说把 Bitmap 放到 Intent 里会导致巨大的内存损耗，所以在传递图片时应该是传递 URI 地址，新界面根据 URI 生成新图片。同时还可以到图片缓存里使用 URI 查找已有图片，节约内存。

13、如何实现切换主题功能？（2017-2-23）

1、定义自定义属性

Activity 布局文件中使用的属性如果想实现主题切换功能，那么就必须使用自定义的属性

```
<declare-styleable name="MyThemeAttrs">
    <attr name="btn_color" format="color" />
    <attr name="btn_background" format="color" />
    <attr name="text_color" format="color" />
</declare-styleable>
```

2、利用上面的属性定义主题

```
<style name="BlueTheme">
    <item name="btn_color">#00f</item>
    <item name="btn_background">#0ff</item>
    <item name="text_color">#00f</item>
    <item name="android:windowBackground">@color/blue_background</item>
</style>
```

3、在布局文件中设置控件属性值时，引用第 1 步定义的属性

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="?attr/text_color"
    android:text="@string/hello_world" />

<Button
    android:onClick="click"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world"
    android:textColor="?attr/btn_color"
    android:background="?attr/btn_background"/>
```

4、在 Activity 中设置主题

```
setTheme(R.style.DefaultTheme); // 必须放在 setContentView 之前
setContentView(R.layout.activity_main);
```

上面几步只是在 Activity 使用自己的主题，实现动态切换主题的思路是把 setTheme 里面的主题作为一个变量，

当需要切换主题时，改变这个变量的值，然后重新创建当前 Activity。

如：

5、在自定义 Application 中记录当前主题

```
public class MyApp extends Application {  
    private int currentTheme = R.style.DefaultTheme;  
    public void changeTheme(int theme){  
        this.currentTheme = theme;  
    }  
    public void initTheme(Context context){  
        context.setTheme(currentTheme);  
    }  
}
```

6、Activity 的 onCreate 方法首先设置主题

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    myApp = (MyApp) getApplication();  
    myApp.initTheme(this);  
    setContentView(R.layout.activity_main);  
}
```

7、当想要切换主题时，先替换 MyApp 中的当前主题，然后重新创建当前 Activity，新的 Activity 创建后就使用了改变后的主题

```
public void click(View v){  
    myApp.changeTheme(R.style.BlueTheme);  
    recreate(); //该方法是 Activity 中的方法，可以让当前 Activity 重新创建实例  
}
```

上面虽然可以切换 Activity 的主题，但是由于每次都需要杀死当前 Activity 重新创建新的 Activity，因此用户体验并不好。

上面的更改主题方案，属于内置主题，该方案虽然可以很方便地修改界面，并且不需要怎么改代码。但它有一个比较致命的缺陷，就是一旦程序发布后，应用所支持的主题风格就固定了。要想增加更多的效果，只能发布新的版本。

如何实现通过网络下载更换主题呢？下面给大家提供一种思路。

为了解决这种问题，便有了 APK 主题方案。

APK 主题方案的基本思路是：在 Android 中，所有的资源都是基于包的。资源以 id 进行标识，在同一个应用中，

每个资源都有唯一标识。但在不同的应用中，可以有相同的 id。因此，只要获取到了其他应用的 Context 对象，就可以通过它的 getResources 获取到其绑定的资源对象。然后，就可以使用 Resources 的 getXXX 方法获取字符串、颜色、dimension、图片等。

要想获取其他应用的 Context 对象，Android 已经为我们提供好了接口。那就是 android.content.ContextWrapper.createPackageContext(String packageName, int flags)方法。

示例代码如下：

```
package com.itheima.baodian;
import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
public class DemoRemoteThemeActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_theme);
        TextView text = (TextView) findViewById(R.id.remoteText);
        TextView color = (TextView) findViewById(R.id.remoteColor);
        ImageView image = (ImageView) findViewById(R.id.remote_image);
        try {
            String remotePackage = "com.xxx.themepackage";
            Context remoteContext = createPackageContext(remotePackage,
                CONTEXT_IGNORE_SECURITY);
            Resources remoteResources = remoteContext.getResources();
            text.setText(remoteResources.getText(remoteResources.getIdentifier("application_name",
"string", remotePackage)));
            color.setTextColor(remoteResources.getColor(remoteResources.getIdentifier("delete_target_hover_tint",
"color", remotePackage)));
            image.setImageDrawable(remoteResources.getDrawable(remoteResources.getIdentifier("ic_launcher_home",
"drawable", remotePackage)));
        } catch (NameNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

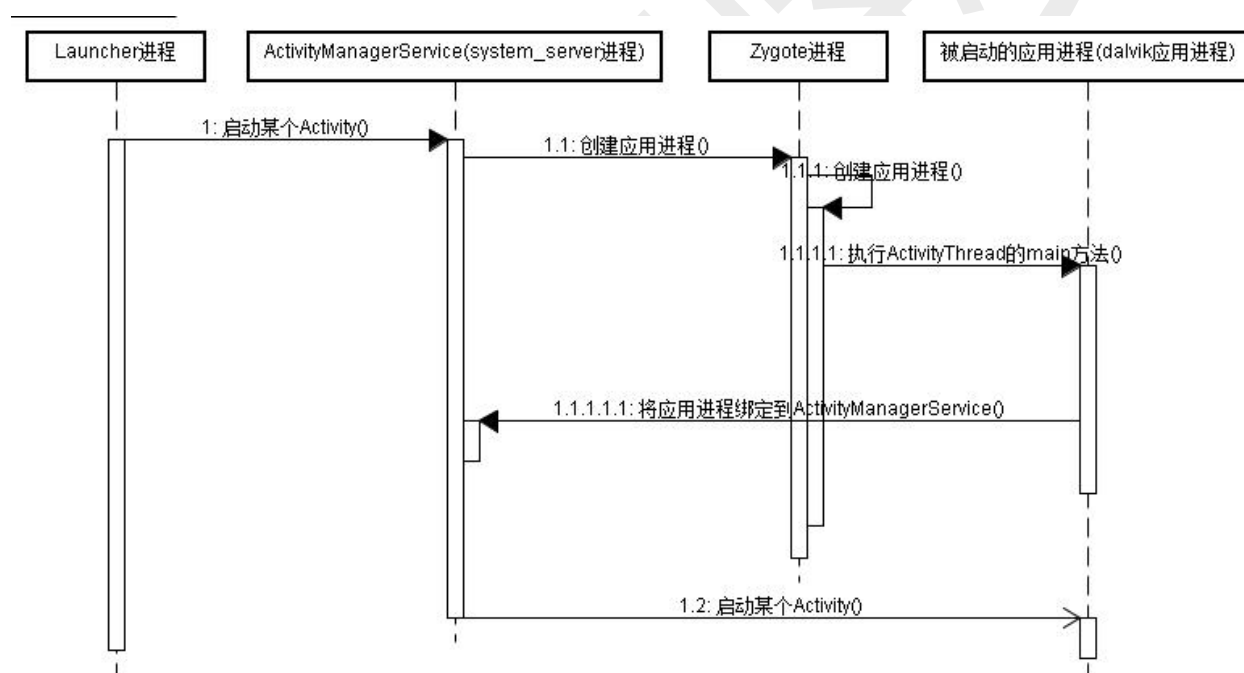
首先，我通过 `createPackageContext` 获取到了包名为 `com.xxx.themepackage` 的应用的上下文。

然后，通过 `Resources` 的 `getIdentifier` 方法获取到相应资源名在该应用中的 `id`。当然，有的人也可以通过使用自身应用的 `id` 的方式，不过这有一个前提，那就是同名资源在主题包应用与当前应用中的 `id` 相同。这貌似可以通过修改编译流程来实现，就像 `framework` 里的 `public.xml` 那样。不过这不在本文的讨论范畴内。

最后，就是通过 `Resources` 的 `getXXX` 方法获取资源了。

14、Android 中 Activity 是如何启动的？（2017-2-24）

Activity 启动时的概要交互流程如下图：



用户在 Launcher 程序里点击应用图标时，会通知 `ActivityManagerService` 启动应用的入口 Activity，`ActivityManagerService` 发现这个应用还未启动，则会通知 `Zygote` 进程孵化出应用进程，然后在这个 `dalvik` 应用进程里执行 `ActivityThread` 的 `main` 方法。在该方法里会先准备好 `Looper` 和消息队列，然后调用 `attach` 方法将应用进程绑定到 `ActivityManagerService`，然后进入 `loop` 循环，不断地读取消息队列里的消息，并分发消息。

```
1. //ActivityThread类 public static void main(String[] args) {
2.     //...
3.     Looper.prepareMainLooper();
4.     ActivityThread thread = new ActivityThread();
5.     thread.attach(false);
6.     if (sMainThreadHandler == null) {
7.         sMainThreadHandler = thread.getHandler();
8.     }
9.     AsyncTask.init();
10.    //...
11.    Looper.loop();
12.    //...}
```

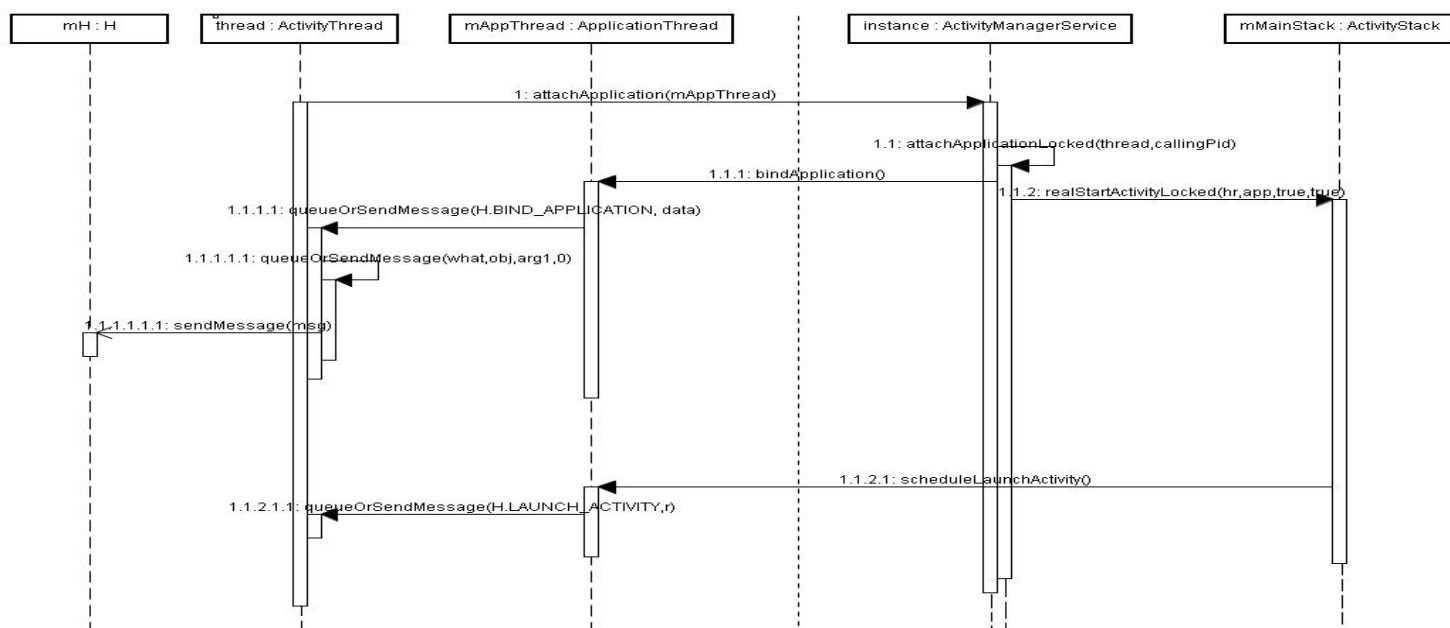
在 ActivityThread 的 main 方法里调用 thread.attach(false);attach 方法的主要代码如下所示:

```
1. //ActivityThread类 private void attach(boolean system) {
2.     sThreadLocal.set(this);
3.     mSystemThread = system;
4.     if (!system) {
5.         //...
6.         IActivityManager mgr = ActivityManagerNative.getDefault();
7.         try {
8.             //调用 ActivityManagerService 的 attachApplication 方法
9.             //将 ApplicationThread 对象绑定至 ActivityManagerService ,
10.            //这样 ActivityManagerService 就可以
11.            //通过 ApplicationThread 代理对象控制应用进程
12.            //调用的是 ActivityManagerService 的 attachApplication
13.            mgr.attachApplication(mAppThread);
14.        } catch (RemoteException ex) {
15.            // Ignore
16.        }
17.    } else {
18.        //...
19.    }
20.    //... }
```

ActivityManagerService 的 attachApplication 方法执行 attachApplicationLocked(thread, callingPid)进行绑定。attachApplicationLocked 方法有两个重要的函数调用 thread.bindApplication 和 mMainStack.realStartActivityLocked。thread.bindApplication 将应用进程的 ApplicationThread 对象绑定到 ActivityManagerService，也就是说获得 ApplicationThread 对象的代理对象。mMainStack.realStartActivityLocked 通知应用进程启动 Activity。

```
1. //ActivityManagerService 类
2. private final boolean attachApplicationLocked(IApplicationThread thread,int pid) {
3.     ProcessRecord app;
4.     //...
5.     app.thread = thread;
6.     //...
7.     try {
8.         //...
9.         //thread 对象其实是 ActivityThread 里 ApplicationThread
10.        //对象在 ActivityManagerService 的代理对象，故此执行
11.        //thread.bindApplication，最终会调用 ApplicationThread 的 bindApplication 方法
12.        //最后会调用 queueOrSendMessage 会往 ActivityThread 的消息队列发送消息，
13.        //消息的用途是 BIND_APPLICATION 这样会在 handler 里处理 BIND_APPLICATION 消息，
14.        //接着调用 handleBindApplication 方法处理绑定消息。
15.        thread.bindApplication(processName, appInfo, providers,
16.                                app.instrumentationClass, profileFile, profileFd, profileAutoStop,
17.                                app.instrumentationArguments, app.instrumentationWatcher, testMode,
18.                                enableOpenGLTrace, isRestrictedBackupMode || !normalMode, app.persistent,
19.                                new Configuration(mConfiguration), app.compat, getCommonServicesLocked(),
20.                                mCoreSettingsObserver.getCoreSettingsLocked());
21.        //...
22.    } catch (Exception e) {
23.        //...
24.    }
25.    //...
26.    ActivityRecord hr = mMainStack.topRunningActivityLocked(null);
27.    if (hr != null && normalMode) {
28.        if (hr.app == null && app.uid == hr.info.applicationInfo.uid
29.            && processName.equals(hr.processName)) {
30.            try {
31.                if (mHeadless) {
32.                    Slog.e(TAG, "Starting activities not supported on headless device: " + hr);
33.                    // realStartActivity 会调用 scheduleLaunchActivity 启动 activity
34.                    //最终会调用 ApplicationThread 的 scheduleLaunchActivity 方法。
35.                    //调用了 queueOrSendMessage 往 ActivityThread 的消息队列发送了消息
36.                    //，消息的用途是启动 Activity
37.                } else if (mMainStack.realStartActivityLocked(hr, app, true, true)) {
38.                    //mMainStack.realStartActivityLocked 真正启动 activity
39.                    didSomething = true;
40.                }
41.            } catch (Exception e) {
42.                //...
43.            }
44.        } else { //... } //... Return true; }
```

综上时序图为：



ActivityThread 的 handler 调用 handleLaunchActivity 处理启动 Activity 的消息，handleLaunchActivity 的主要

代码如下所示:

```

1. //ActivityThread 类
2. private void handleLaunchActivity(ActivityClientRecord r, Intent customIntent) {
3.     //...
4.     Activity a = performLaunchActivity(r, customIntent);
5.     if (a != null) {
6.         //...
7.         handleResumeActivity(r.token, false, r.isForward,
8.             !r.activity.mFinished && !r.startsNotResumed);
9.         //...
10.    } else {
11.        //...
12.    }}
  
```

handleLaunchActivity 方法里有两个重要的函数调用，performLaunchActivity 和 handleResumeActivity,performLaunchActivity 会调用 Activity 的 onCreate,onStart,onRestoreInstanceState 方法,handleResumeActivity 会调用 Activity 的 onResume 方法.代码如下：

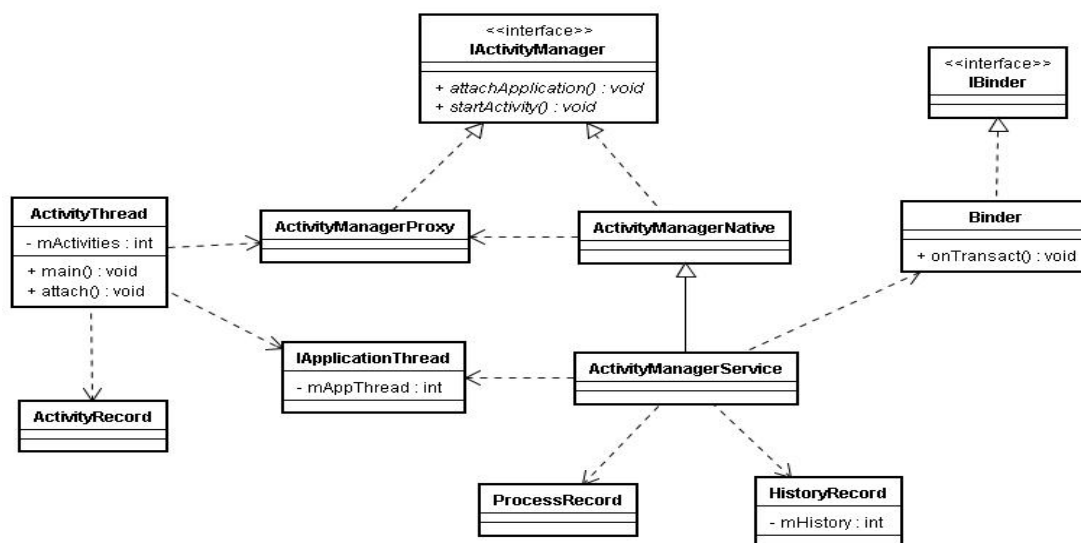
performLaunchActivity 的主要代码如下所示:

```

1.  //ActivityThread类
2.  private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {
3.  //...
4.      //会调用 Activity 的 onCreate 方法
5.      mInstrumentation.callActivityOnCreate(activity, r.state);
6.      //...
7.      //...
8.      //调用 Activity 的 onStart 方法
9.      if (!r.activity.mFinished) {
10.         activity.performStart();
11.         r.stopped = false;
12.     }
13.     if (!r.activity.mFinished) {
14.         if (r.state != null) {
15.             //会调用 Activity 的 onRestoreInstanceState 方法
16.             mInstrumentation.callActivityOnRestoreInstanceState(activity, r.state);
17.         }.... }

```

启动 Activity 主要涉及到的类：



Activity 的管理采用 binder 机制，管理 Activity 的接口是 IActivityManager。ActivityManagerService 实现了 Activity 管理功能，位于 system_server 进程，ActivityManagerProxy 对象是 ActivityManagerService 在普通应用进程的一个代理对象，应用进程通过 ActivityManagerProxy 对象调用 ActivityManagerService 提供的功能。应用进程并不会直接创建 ActivityManagerProxy 对象，而是通过调用 ActivityManagerNative 类的工具方法 getDefault 方法得到 ActivityManagerProxy 对象。所以在应用进程里通常这样启动 Activity。

```
1. ActivityManagerNative.getDefault().startActivity()
```

ActivityManagerService 也需要主动调用应用进程以控制应用进程并完成指定操作。这样

ActivityManagerService 也需要应用进程的一个 Binder 代理对象，而这个代理对象就是 ApplicationThreadProxy 对象。 ActivityManagerService 通过 IApplicationThread 接口管理应用进程， ApplicationThread 类实现了 IApplicationThread 接口，实现了管理应用的操作， ApplicationThread 对象运行在应用进程里。

ApplicationThreadProxy 对象是 ApplicationThread 对象在 ActivityManagerService 线程 (ActivityManagerService 线程运行在 system_server 进程)内的代理对象， ActivityManagerService 通过 ApplicationThreadProxy 对象调用 ApplicationThread 提供的功能，比如让应用进程启动某个 Activity。

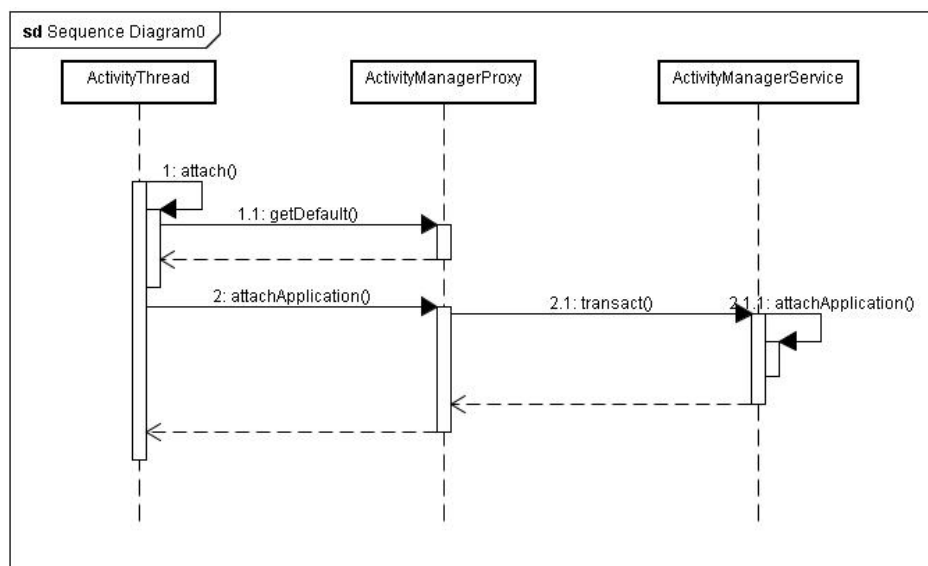
```
1. /**
2. public static void main(String[] args) {
3.     Process.setArgV0("<pre-initialized>");
4.     //创建 looper
5.     Looper.prepareMainLooper();
6.     //会通过 thread.attach(false)来初始化应用程序的运行环境
7.     ActivityThread thread = new ActivityThread();
8.     thread.attach(false);
9. }
```

在建立消息循环之前，会通过 thread.attach(false)来初始化应用程序的运行环境，并建立 activityThread 和 ActivityManagerService 之间的桥 mAppThread， mAppThread 是 IApplicationThread 的一个实例。

```
1. RuntimeInit.setApplicationObject(mAppThread.asBinder());
2. final IActivityManager mgr = ActivityManagerNative.getDefault();
3. try {
4.     mgr.attachApplication(mAppThread);
5. } catch (RemoteException ex) {
6.     throw ex.rethrowFromSystemServer();
7. }
```

注意：每个应用程序对应着一个 ActivityThread 实例，应用程序由 ActivityThread.main 打开消息循环。每个应用程序同时也对应着一个 ApplicationThread 对象。该对象是 activityThread 和 ActivityManagerService 之间的桥梁。在 attach 中还做了一件事情，就是通过代理调用 attachApplication，并利用 binder 的 transact 机制，在

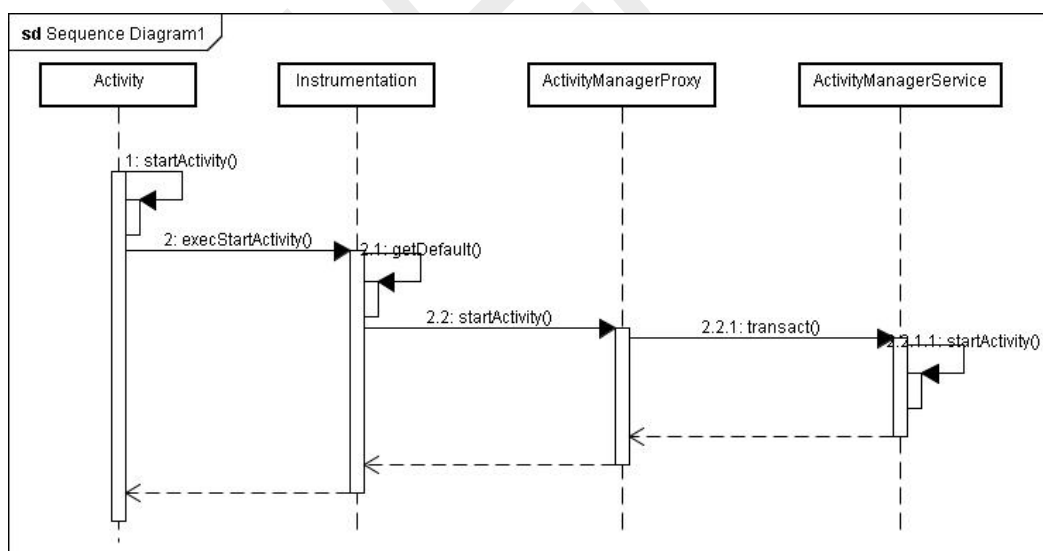
ActivityManagerService 中建立了 ProcessRecord 信息。



ActivityManagerService:

在 ActivityManagerService 中，也有一个用来管理 activity 的地方：mHistory 栈，这个 mHistory 栈里存放的是服务端的 activity 记录 HistoryActivity (class HistoryRecord extends IApplicationToken.Stub)。处于栈顶的就是当前 running 状态的 activity。

我们来看一下 Activity 的 startActivity 方法的请求过程：



从该时序图中可以看出，Activity.startActivity()方法最终是通过代理类和 Binder 机制，在

ActivityManagerService.startActivity 方法中执行的。那么在 ActivityManagerService 的 startActivity 中，主要做了那些事情？我们来看下里面比较重要的代码段：

根据 activity、ProcessRecord 等信息创建 HistoryRecord 实例 r

```
1. HistoryRecord r = new HistoryRecord(  
2.     this, callerApp, callingUid, intent, resolvedType, aInfo,  
3.     mConfiguration, resultRecord, resultWho, requestCode, componentSpecified);
```

把 r 加入到 mHistory 中。

```
4. mHistory.add(addPos, r);
```

activity 被加入到 mHistory 之后，只是说明在服务端可以找到该 activity 记录了，但是在客户端目前还没有该 activity 记录。还需要通过 ProcessRecord 中的 thread (IApplication) 变量，调用它的 scheduleLaunchActivity 方法在 ActivityThread 中创建新的 ActivityRecord 记录（之前我们说过，客户端的 activity 是用 ActivityRecord 记录的，并放在 mActivities 中）。

```
5. app.thread.scheduleLaunchActivity(  
new Intent(r.intent), r, System.identityHashCode(r),  
6.     r.info, r.icicle, results, newIntents, !andResume, isNextTransitionForward());
```

在这个里面主要是根据服务端返回回来的信息创建客户端 activity 记录 ActivityRecord。并通过 Handler 发送消息到消息队列，进入消息循环。在 ActivityThread.handleMessage() 中处理消息。最终在 handleLaunchActivity 方法中把 ActivityRecord 记录加入到 mActivities (mActivities.put(r.token,r)) 中，并启动 activity。

总结：1) 在客户端和服务端分别有一个管理 activity 的地方，服务端是在 mHistory 中，处于 mHistory 栈顶的就是当前处于 running 状态的 activity，客户端是在 mActivities 中。2) 在 startActivity 时，首先会在 ActivityManagerService 中建立 HistoryRecord，并加入到 mHistory 中，然后通过 scheduleLaunchActivity 在客户端创建 ActivityRecord 记录并加入到 mActivities 中。最终在 ActivityThread 发起请求，进入消息循环，完成 activity 的启动和窗口的管理等。

三、Service

1、Service 是否在 main thread 中执行, service 里面是否能执行耗时的操作?

默认情况,如果没有显示的指 service 所运行的进程, Service 和 activity 是运行在当前 app 所在进程的 main thread(UI 主线程)里面。

service 里面不能执行耗时的操作(网络请求,拷贝数据库,大文件)

特殊情况 ,可以在清单文件配置 service 执行所在的进程 ,让 service 在另外的进程中执行

```
<service
    android:name="com.baidu.location.f"
    android:enabled="true"
    android:process=":remote" >
</service>
```

2、Activity 怎么和 Service 绑定，怎么在 Activity 中启动自己对应的 Service ？

Activity 通过 `bindService(Intent service, ServiceConnection conn, int flags)`跟 Service 进行绑定，当绑定成功的时候 Service 会将代理对象通过回调的形式传给 `conn`，这样我们就拿到了 Service 提供的服务代理对象。

在 Activity 中可以通过 `startService` 和 `bindService` 方法启动 Service。一般情况下如果想获取 Service 的服务对象那么肯定需要通过 `bindService ()` 方法，比如音乐播放器，第三方支付等。如果仅仅只是为了开启一个后台任务那么可以使用 `startService ()` 方法。

3、请描述一下 Service 的生命周期

Service 有绑定模式和非绑定模式，以及这两种模式的混合使用方式。不同的使用方法生命周期方法也不同。

非绑定模式 当第一次调用 `startService` 的时候执行的方法依次为 `onCreate()`、`onStartCommand()` (`onStart()`)

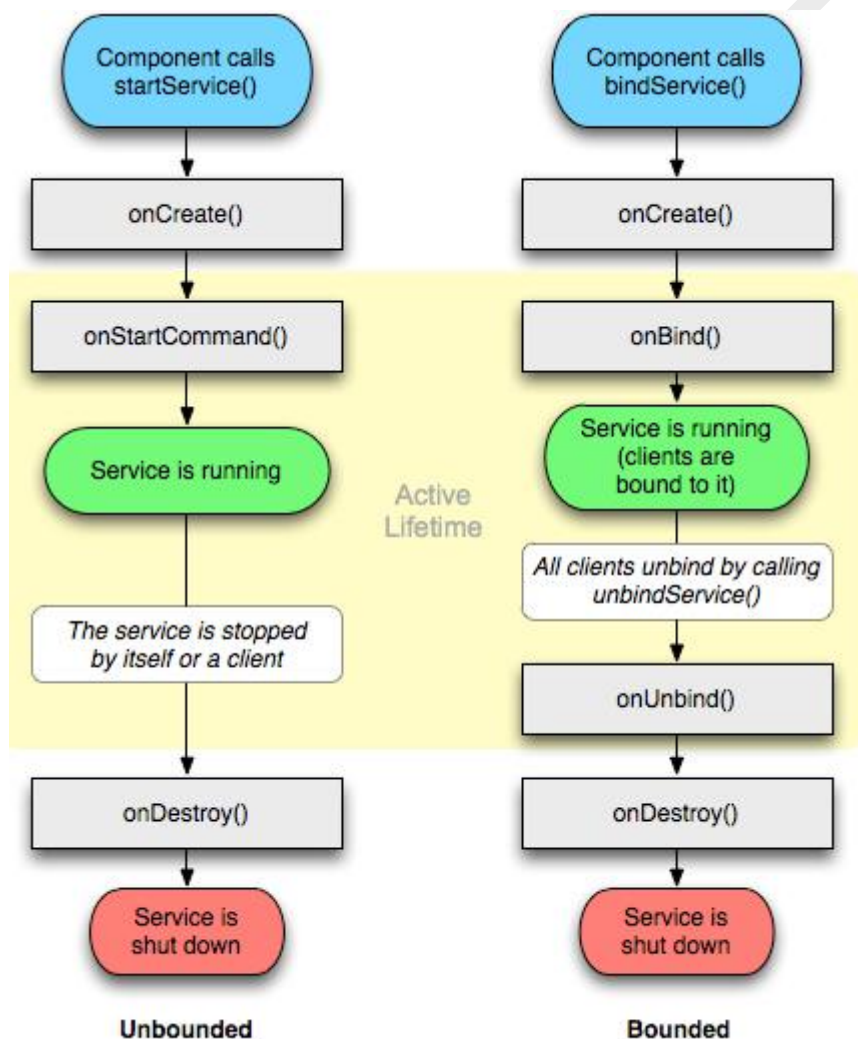
当 Service 关闭的时候调用 `onDestory` 方法。

绑定模式：第一次 `bindService()` 的时候，执行的方法为 `onCreate()`、`onBind()` 解除绑定的时候会执行 `onUnbind()`、`onDestory()`。

上面的两种生命周期是在相对单纯的模式下的情形。我们在开发的过程中还必须注意 Service 实例只会有一个，也就是说如果当前要启动的 Service 已经存在了那么就不会再次创建该 Service 当然也不会调用 `onCreate()` 方法。

一个 Service 可以被多个客户进行绑定，只有所有的绑定对象都执行了 `onBind()` 方法后该 Service 才会销毁，不过如果有一个客户执行了 `onStart()` 方法，那么这个时候如果所有的 bind 客户都执行了 `unBind()` 该 Service 也不会销毁。

Service 的生命周期图如下所示，帮助大家记忆。



4、什么是 IntentService ? 有何优点 ?

我们通常只会使用 Service , 可能 IntentService 对大部分同学来说都是第一次听说。那么看了下面的介绍相信你就不会再陌生了。如果你还是不了解那么在面试的时候你就坦诚说没用过或者不了解等。并不是所有的问题都需要回答上来的。

一、IntentService 简介

IntentService 是 Service 的子类，比普通的 Service 增加了额外的功能。先看 Service 本身存在两个问题：

Service 不会专门启动一条单独的进程，Service 与它所在应用位于同一个进程中；

Service 也不是专门一条新线程，因此不应该在 Service 中直接处理耗时的任务；

二、IntentService 特征

会创建独立的 worker 线程来处理所有的 Intent 请求；

会创建独立的 worker 线程来处理 onHandleIntent()方法实现的代码，无需处理多线程问题；

所有请求处理完成后，IntentService 会自动停止，无需调用 stopSelf()方法停止 Service ；

为 Service 的 onBind()提供默认实现，返回 null ；

为 Service 的 onStartCommand 提供默认实现，将请求 Intent 添加到队列中；

三、使用 IntentService

本人写了一个 IntentService 的使用例子供参考。该例子中一个 MainActivity 一个 MyIntentService，这两个类都是四大组件当然需要在清单文件中注册。这里只给出核心代码：

MainActivity.java:

```
public void click(View view){
    Intent intent = new Intent(this, MyIntentService.class);
    intent.putExtra("start", "MyIntentService");
    startService(intent);
}
```

MyIntentService.java

```
public class MyIntentService extends IntentService {

    private String ex = "";
    private Handler mHandler = new Handler() {

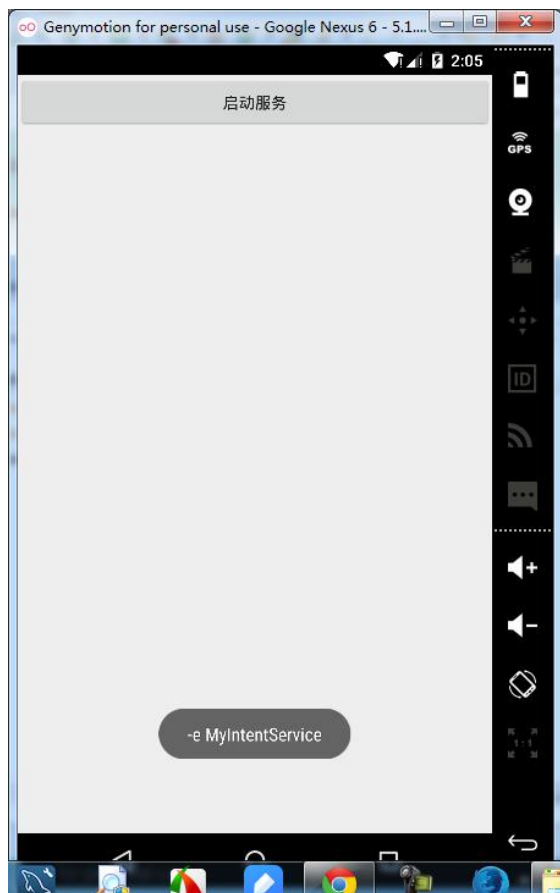
        public void handleMessage(android.os.Message msg) {
            Toast.makeText(MyIntentService.this, "-e " + ex,
                Toast.LENGTH_LONG).show();
        }
    };

    public MyIntentService(){
        super("MyIntentService");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        ex = intent.getStringExtra("start");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        /**
         * 模拟执行耗时任务
         * 该方法是在子线程中执行的，因此需要用到 handler 跟主线程进行通信
         */
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        mHandler.sendEmptyMessage(0);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

运行后效果如下：



5、说说 Activity、Intent、Service 是什么关系

他们都是 Android 开发中使用频率最高的类。其中 Activity 和 Service 都是 Android 四大组件之一。他俩都是 Context 类的子类 ContextWrapper 的子类，因此他俩可以算是兄弟关系吧。不过兄弟俩各有各的本领，Activity 负责用户界面的显示和交互，Service 负责后台任务的处理。Activity 和 Service 之间可以通过 Intent 传递数据，因此可以把 Intent 看作是通信使者。

6、Service 和 Activity 在同一个线程吗

对于同一 app 来说默认情况下是在同一个线程中的，main Thread（UI Thread）。

7、Service 里面可以弹吐司么？

可以的。弹吐司有个条件就是得有一个 Context 上下文，而 Service 本身就是 Context 的子类，因此在 Service 里面弹吐司是完全可以的。比如我们在 Service 中完成下载任务后可以弹一个吐司通知用户。

8、如何让一个 Service 成为前置进程？

在启动该 Service 的时候可以在添加上如下方法：

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Notification notification = new Notification(R.drawable.ic_launcher, "服务正在运行", System.currentTimeMillis());
    Intent notificationIntent = new Intent(this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
    notification.setLatestEventInfo(this, "消息标题", "消息内容", pendingIntent);
    startForeground(1, notification);
    return super.onStartCommand(intent, flags, startId);
}
```

9、Service 的 onStartCommand 方法有几种返回值？各代表什么意思？

有四种返回值，不同值代表的意思如下：

START_STICKY：如果 service 进程被 kill 掉，保留 service 的状态为开始状态，但不保留递送的 intent 对象。随后系统会尝试重新创建 service，由于服务状态为开始状态，所以创建服务后一定会调用 `onStartCommand(Intent,int,int)` 方法。如果在此期间没有任何启动命令被传递到 service 那么参数 Intent 将为 null。

START_NOT_STICKY：“非粘性的”。使用这个返回值时，如果在执行完 `onStartCommand` 后，服务被异常 kill 掉，系统不会自动重启该服务。

START_REDELIVER_INTENT：重传 Intent。使用这个返回值时，如果在执行完 `onStartCommand` 后，服务被异常

常 kill 掉，系统会自动重启该服务，并将 Intent 的值传入。

START_STICKY_COMPATIBILITY：START_STICKY 的兼容版本，但不保证服务被 kill 后一定能重启。

10、Service 的 onRebind (Intent) 方法在什么情况下会执行？

如果在 onUnbind () 方法返回 true 的情况下会执行，否则不执行。

官方解释如下：

Called when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind. This will only be called if the implementation of onUnbind was overridden to return true.

11、Activity 调用 Service 中的方法都有哪些方式？

Activity 调用 Service 中的方法主要是通过绑定服务的模式实现的，绑定服务又分为三种方式。如下所示：

一、Extending the Binder class

通过 Binder 接口的形式实现，当 Activity 绑定 Service 成功的时候 Activity 会在 ServiceConnection 的类的 onServiceConnected () 回调方法中获取到 Service 的 onBind () 方法 return 过来的 Binder 的子类。

二、Using a Messenger

这是官方给出的另外一种沟通方式。原文如下：

If you need your interface to work across different processes, you can create an interface for the service with a Messenger. In this manner, the service defines a Handler that responds to different types of Message objects. This Handler is the basis for a Messenger that can then share an IBinder with the client, allowing the client to send commands to the service using Message objects. Additionally, the client can define a Messenger of its own so the service can send messages back.

This is the simplest way to perform interprocess communication (IPC), because the Messenger queues all requests into a single thread so that you don't have to design your service to be thread-safe.

Here's a summary of how to use a Messenger:

- The service implements a Handler that receives a callback for each call from a client.
- The Handler is used to create a Messenger object (which is a reference to the Handler).
- The Messenger creates an IBinder that the service returns to clients from onBind().

- Clients use the `IBinder` to instantiate the `Messenger` (that references the service's `Handler`), which the client uses to send `Message` objects to the service.
- The service receives each `Message` in its `Handler`—specifically, in the `handleMessage()` method.

In this way, there are no "methods" for the client to call on the service. Instead, the client delivers "messages" (`Message` objects) that the service receives in its `Handler`.

Here's a simple example service that uses a `Messenger` interface:

```
1. public class MessengerService extends Service {
2.     /** Command to the service to display a message */
3.     static final int MSG_SAY_HELLO = 1;
4.
5.     /**
6.      * Handler of incoming messages from clients.
7.      */
8.     class IncomingHandler extends Handler {
9.         @Override
10.        public void handleMessage(Message msg) {
11.            switch (msg.what) {
12.                case MSG_SAY_HELLO:
13.                    Toast.makeText(getApplicationContext(),
14. "hello!", Toast.LENGTH_SHORT).show();
15.                    break;
16.                default:
17.                    super.handleMessage(msg);
18.            }
19.        }
20.    }
21.
22.    /**
23.     * Target we publish for clients to send messages to IncomingHandler.
24.     */
25.    final Messenger mMessenger = new Messenger(new IncomingHandler());
26.
27.    /**
28.     * When binding to the service, we return an interface to our messenger
29.     * for sending messages to the service.
30.     */
31.    @Override
32.    public IBinder onBind(Intent intent) {
33.        Toast.makeText(getApplicationContext(),
34. "binding", Toast.LENGTH_SHORT).show();
35.        return mMessenger.getBinder();
36.    }
37. }
```

Notice that the `handleMessage()` method in the `Handler` is where the service receives the incoming `Message` and decides what to do, based on the `what` member.

All that a client needs to do is create a `Messenger` based on the `IBinder` returned by the service and send a message using `send()`. For example, here's a simple activity that binds to the service and delivers the `MSG_SAY_HELLO` message to the service:

```
1. public class ActivityMessenger extends Activity {
2.     /** Messenger for communicating with the service. */
3.     Messenger mService = null;
4.
5.     /** Flag indicating whether we have called bind on the service. */
6.     boolean mBound;
7.
8.     /**
9.      * Class for interacting with the main interface of the service.
10.     */
11.     private ServiceConnection mConnection = new ServiceConnection() {
12.         public void onServiceConnected(ComponentName className, IBinder service) {
13.             // This is called when the connection with the service has been
14.             // established, giving us the object we can use to
15.             // interact with the service. We are communicating with the
16.             // service using a Messenger, so here we get a client-side
17.             // representation of that from the raw IBinder object.
18.             mService = new Messenger(service);
19.             mBound = true;
20.         }
21.
22.         public void onServiceDisconnected(ComponentName className) {
23.             // This is called when the connection with the service has been
24.             // unexpectedly disconnected -- that is, its process crashed.
25.             mService = null;
26.             mBound = false;
27.         }
28.     };
29.
30.     public void sayHello(View v) {
31.         if (!mBound) return;
32.         // Create and send a message to the service, using a supported 'what' value
33.         Message msg = Message.obtain(null, MessengerService.MSG_SAY_HELLO, 0, 0);
34.         try {
35.             mService.send(msg);
36.         } catch (RemoteException e) {
37.             e.printStackTrace();
38.         }
```

```
39.    }
40.
41.    @Override
42.    protected void onCreate(Bundle savedInstanceState) {
43.        super.onCreate(savedInstanceState);
44.        setContentView(R.layout.main);
45.    }
46.
47.    @Override
48.    protected void onStart() {
49.        super.onStart();
50.        // Bind to the service
51.        bindService(new Intent(this, MessengerService.class), mConnection,
52.            Context.BIND_AUTO_CREATE);
53.    }
54.
55.    @Override
56.    protected void onStop() {
57.        super.onStop();
58.        // Unbind from the service
59.        if (mBound) {
60.            unbindService(mConnection);
61.            mBound = false;
62.        }
63.    }
64. }
```

Notice that this example does not show how the service can respond to the client. If you want the service to respond, then you need to also create a Messenger in the client. Then when the client receives the `onServiceConnected()` callback, it sends a Message to the service that includes the client's Messenger in the `replyTo` parameter of the `send()` method.

三、Using AIDL

aidl 比较适合当客户端和服务端不在同一个应用下的场景。

12、Activity 如何给 Service 发送 Message ? (2015.10.18)

该题和下一题 都是关于 Activity 和 Service 直接护发消息的知识 非常的有意思 但是也非常不好回答。Activity 给 Service 还算稍微简单一点，但是 Service 给 Activity 发送消息就有点难度了，不是常人能所了解的。如果你没有

看过 Android 官方关于 Bound Service 的解释，估计很难应付。

Activity 如何给 Service 发送 Message 见《Activity 调用 Service 中的方法都有哪些方式？》题目中 Activity 跟 Service 绑定的第二种方式：Using a Messenger。

13、Service 如何给 Activity 发送 Message？（2015.10.18）

Service 和 Activity 如果想互发 Message 就必须使用使用 Messenger 机制。

Service 如何给 Activity 发送 Message 见《Activity 调用 Service 中的方法都有哪些方式？》题目中 Activity 跟 Service 绑定的第二种方式：Using a Messenger。

考虑到官方只给出了 Activity 给 Service 发送 Message 的代码，在这里我给出一个 Activity 跟 Service 之间互相发送 Message 通信的示例代码：

1. MainActivity 代码

```
1. package com.example.serviceAndActivity;
2.
3. import android.os.Bundle;
4. import android.os.Handler;
5. import android.os.IBinder;
6. import android.os.Message;
7. import android.os.Messenger;
8. import android.os.RemoteException;
9. import android.app.Activity;
10. import android.app.Service;
11. import android.content.ComponentName;
12. import android.content.Intent;
13. import android.content.ServiceConnection;
14. import android.util.Log;
15. import android.view.View;
16. import android.widget.Toast;
17. /**
18.  * Activity 和 Service 互发 Message
19.  *
20.  * @author wzy 2015-11-25
21.  *
22.  */
23. public class MainActivity extends Activity {
```

```
24. private Messenger messenger;
25. //将该 Handler 发送 Service
26. private Messenger mOutMessenger = new Messenger(new OutgoingHandler());
27.
28. @Override
29. protected void onCreate(Bundle savedInstanceState) {
30.     super.onCreate(savedInstanceState);
31.     setContentView(R.layout.activity_main);
32. }
33. // 绑定服务
34. public void click1(View view) {
35.     Intent intent = new Intent(this, MessengerService.class);
36.     ServiceConnection conn = new MyServiceConnection();
37.     bindService(intent, conn, Service.BIND_AUTO_CREATE);
38. }
39.
40. // 发送消息
41. public void click2(View view) throws RemoteException {
42.     if (messenger == null) {
43.         Toast.makeText(this, "服务不可用！", Toast.LENGTH_SHORT).show();
44.         return;
45.     }
46.     Message message = new Message();
47.     message.obj="长江长江我是黄河";
48.     message.what =0;
49.     messenger.send(message);
50.
51. }
52.
53. class OutgoingHandler extends Handler{
54.     @Override
55.     public void handleMessage(Message msg) {
56.         Log.d("tag", msg.toString());
57.     }
58. }
59.
60. class MyServiceConnection implements ServiceConnection {
61.
62.     @Override
63.     public void onServiceConnected(ComponentName name, IBinder service) {
64.         Toast.makeText(MainActivity.this, "连接成功！", Toast.LENGTH_SHORT).show();
65.         messenger = new Messenger(service);
66.         Message message=new Message();
67.         message.what = 1;
```

```
68. //Activity 绑定 Service 的时候给 Service 发送一个消息，该消息的 obj 属性是一个 Messenger 对象
69.     message.obj = mOutMessenger;
70.     try {
71.         messenger.send(message);
72.     } catch (RemoteException e) {
73.         e.printStackTrace();
74.     }
75. }
76.
77. @Override
78. public void onServiceDisconnected(ComponentName name) {
79.     Toast.makeText(MainActivity.this, "连接已经断开！", Toast.LENGTH_SHORT).show();
80. }
81.
82. }
83.
84. }
```

2. MessengerService 代码

```
1. package com.example.serviceAndActivity;

import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;
/**
 * 该 Service 接收到 Activity 的消息后会在返回一条消息
 *
 * @author wzy 2015-11-25
 *
 */
public class MessengerService extends Service {

    private Messenger messenger = new Messenger(new IncomingHandler());

    private Messenger mActivityMessenger ;

    @Override
    public IBinder onBind(Intent intent) {
        IBinder binder = messenger.getBinder();
    }
}
```

```
        return binder;
    }

    //1.定义一个 Handler 对象，该 Handler 处理 Activity 发送过来的消息
    class IncomingHandler extends Handler{
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case 0:
                    Log.d("tag", msg.toString());
                    if (mActivityMessenger!=null) {
                        Message message= new Message();
                        message.what = 2;
                        message.obj="地瓜地瓜我是土豆";
                        try {
                            mActivityMessenger.send(message);
                        } catch (RemoteException e) {
                            e.printStackTrace();
                        }
                    }
                    break;
                case 1:
                    mActivityMessenger = (Messenger) msg.obj;
                    Log.d("tag", "已经获取到 Activity 发送了的 Messenger 对象");
                    break;
                default:
                    break;
            }
        }
    }
}
```

运行界面以及日志如下图所示：



14、子线程不能代替 service 吗？(2017-2-23)

a) 不能替代

b) 首先要确认一件事，服务作为四大组件之一，是运行在主线程的，可以直接显示吐司，修改 View 等。如果要运行耗时操作，服务需要自己开启子线程。

c) 只作为后台来理解的话，相比于线程，服务具备完善的生命周期，更方便随时释放资源。

d) 服务自己就有上下文(Context)对象，可以确定上下文是正常可用的。线程需要从外部获取上下文对象，在运行时无法保证该对象没有被系统销毁。

四、BroadcastReceiver

1、请描述一下 BroadcastReceiver

BroadcastReceiver 是 Android 四大组件之一，主要用于接收系统或者 app 发送的广播事件。

广播分两种：有序广播和无序广播。

内部通信实现机制：通过 Android 系统的 Binder 机制实现通信。

无序广播：完全异步，逻辑上可以被任何广播接收者接收到。优点是效率较高。缺点是一个接收者不能将处理结果传递给下一个接收者，并无法终止广播 intent 的传播。

有序广播：按照被接收者的优先级顺序，在被接收者中依次传播。比如有三个广播接收者 A, B, C, 优先级是 A > B > C。那这个消息先传给 A, 再传给 B, 最后传给 C。每个接收者有权终止广播，比如 B 终止广播，C 就无法接收到。此外 A 接收到广播后可以对结果对象进行操作，当广播传给 B 时，B 可以从结果对象中取得 A 存入的数据。

在通过 Context.sendOrderedBroadcast(intent, receiverPermission, resultReceiver, scheduler, initialCode, initialData, initialExtras)时我们可以指定 resultReceiver 广播接收者，这个接收者我们可以认为是最终接收者，通常

情况下如果比他优先级更高的接收者如果没有终止广播，那么他的 onReceive 会被执行两次，第一次是正常的按照优先级顺序执行，第二次是作为最终接收者接收。如果比他优先级高的接收者终止了广播，那么他依然能接收到广播。

在我们的项目中经常使用广播接收者接收系统通知，比如开机启动、sd 挂载、低电量、外播电话、锁屏等。

如果我们做的是播放器，那么监听到用户锁屏后我们应该将我们的播放之暂停等。

2、在 manifest 和代码中如何注册和使用 BroadcastReceiver

在清单文件中注册广播接收者称为静态注册，在代码中注册称为动态注册。静态注册的广播接收者只要 app 在系统中运行则一直可以接收到广播消息，动态注册的广播接收者当注册的 Activity 或者 Service 销毁了那么就接收不到广播了。

静态注册：在清单文件中进行如下配置

```
<receiver android:name=".BroadcastReceiver1" >
    <intent-filter>
        <action android:name="android.intent.action.CALL" >
        </action>
    </intent-filter>
</receiver>
```

动态注册：在代码中进行如下注册

```
receiver = new BroadcastReceiver();
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(CALL_ACTION);
context.registerReceiver(receiver, intentFilter);
```

3、BroadcastReceiver 的生命周期

- 广播接收者的生命周期非常短暂的，在接收到广播的时候创建，onReceive()方法结束之后销毁；
- 广播接收者中不要做一些耗时的工作，否则会弹出 Application No Response 错误对话框；
- 最好也不要再在广播接收者中创建子线程做耗时的工作，因为广播接收者被销毁后进程就成为了空进程，很容易

被系统杀掉；

d. 耗时的较长的工作最好放在服务中完成；

4、如何让自己的广播只让指定的 app 接收 (2015.09.02)

1、自己的应用（假设名称为应用 A）在发送广播的时候给自己发送的广播添加自定义权限，假设权限名为：

`com.itheima.android.permission`，然后需要在应用 A 的 `AndroidManifest.xml` 中声明如下权限：

```
<permission android:name="com.itheima.android.permission"
android:protectionLevel="normal"></permission>
<uses-permission android:name="com.itheima.android.permission"/>
```

2、其他应用（假设名称为应用 B）如果想接收该广播，那么就必须知道应用 A 广播使用的权限。然后在应用 B

的清单文件中如下配置：

```
<uses-permission android:name="com.itheima.android.permission"/>
```

或者在应用 `AndroidManifest.xml` 中的 `<receiver>` 标签中进行如下配置：

```
<receiver android:name="com.itheima.android.broadcastReceiver.MyReceiver"
android:permission="com.itheima.android.permission">
    <intent-filter>
        <action android:name="com.itheima.mybroadcast"></action>
    </intent-filter>
</receiver>
```

每个权限通过 `protectionLevel` 来标识保护级别：

normal：低风险权限，只要申请了就可以使用（在 `AndroidManifest.xml` 中添加 `<uses-permission>` 标签），安装时不需要用户确认；

dangerous：高风险权限，安装时需要用户的确认才可使用；

signature：只有当申请权限的应用程序的数字签名与声明此权限的应用程序的数字签名相同时（如果是申请系统权限，则需要与系统签名相同），才能将权限授给它；

signatureOrSystem：签名相同，或者申请权限的应用为系统应用（在 `system image` 中）。

上述四类权限级别同样可用于自定义权限中。如果开发者需要对自己的应用程序（或部分应用）进行访问控制，则可以通过在 `AndroidManifest.xml` 中添加 `<permission>` 标签，将其属性中的 `protectionLevel` 设置为上述四类级别中的某一种来实现。

5、什么是最终广播接收者？

最终广播是我们自己应用发送有序广播时通过 `ContextWrapper.sendOrderedBroadcast()` 方法指定的当前应用下的广播，该广播可能会被执行两次，第一次是作为普通广播按照优先级接收广播，第二次是作为 final receiver 必须接收一次。

6、广播的优先级对无序广播生效吗？

生效的。广播的优先级推荐的范围是： $[-1000, +1000]$ ，但是如果设置的优先级值超过这个范围也是可以的。

7、动态注册的广播优先级谁高？

谁先注册谁优先级高。

8、如何判断当前 BroadcastReceiver 接收到的是有序广播还是无序广播？

(2015-10-16)

在 `BroadcastReceiver` 类中 `onReceive()` 方法中，可以调用 `boolean b = isOrderedBroadcast()`；该方法是 `BroadcastReceiver` 类中提供的方法，用于告诉我们当前的接收到的广播是否为有序广播。

9、Android 引入广播机制的用意 (2017-2-23)

a.从 MVC 的角度考虑(应用程序内) 其实回答这个问题的时候还可以这样问，android 为什么要有那 4 大组件，现在的移动开发模型基本上也是照搬的 web 那一套 MVC 架构，只不过是改了点嫁妆而已。android 的四大组件本质上就是为了实现移动或者说嵌入式设备上的 MVC 架构，它们之间有时候是一种相互依存的关系，有时候又是一种补充关系，引入广播机制可以方便几大组件的信息和数据交互。

b.程序间互通消息(例如在自己的应用程序内监听系统来电)

c.效率上(参考 UDP 的广播协议在局域网的方便性)

d.设计模式上(反转控制的一种应用，类似监听者模式)

10、网络状态改变是无序广播还是有序广播，安装了，没启动过，会接受这个广播么？

(2017-2-24)

无序广播。不会。android 在 3.0 之后，对广播增加了一个标记：Intent.FLAG_EXCLUDE_STOPPED_PACKAGES，这个是为了加强了对“停止”状态 APP 的管理（比如 app 安装后未启动或者被用户强制停止）。广播加上这个 Flag 之后，处于“停止”状态的 APP 是无法收到广播的，系统发出的广播基本都有这个 Flag。因此该类广播我们在使用的时候主要是采用动态注册的方式。

五、ContentProvider&数据库

1、请介绍下 ContentProvider 是如何实现数据共享的？

在 Android 中如果想将自己应用的数据（一般多为数据库中的数据）提供给第三发应用，那么我们只能通过 ContentProvider 来实现了。

ContentProvider 是应用程序之间共享数据的接口。使用的时候首先自定义一个类继承 ContentProvider，然后覆写 query、insert、update、delete 等方法。因为其是四大组件之一因此必须在 AndroidManifest 文件中进行注册。

```
<provider
    android:exported="true"
    android:name="com.itheima.contentProvider.provider.PersonContentProvider"
    android:authorities="com.itheima.person" />
```

第三方可以通过 ContentResolver 来访问该 Provider。

2、为什么要用 ContentProvider ? 它和 sql 的实现上有什么差别 ?

ContentProvider 屏蔽了数据存储的细节,内部实现对用户完全透明,用户只需要关心操作数据的 uri 就可以了 , ContentProvider 可以实现不同 app 之间共享。

Sql 也有增删改查的方法 , 但是 sql 只能查询本应用下的数据库。而 ContentProvider 还可以去增删改查本地文件. xml 文件的读取等。

3、说说 ContentProvider、ContentResolver、ContentObserver 之间的关系

ContentProvider 内容提供者 , 用于对外提供数据

ContentResolver.notifyChange(uri)发出消息

ContentResolver 内容解析者 , 用于获取内容提供者提供的数据

ContentObserver 内容监听器 , 可以监听数据的改变状态

ContentResolver.registerContentObserver()监听消息。

4、如何访问 asserts 资源目录下的数据库 ?

//1. 获取到 assert 目录下的 db 文件

```
AssetManager assetManager = getAssets();
InputStream is = assetManager.open("myuser.db");
//将文件拷贝到
/data/data/com.itheima.android.asserts.sqlite/databases/myuser.db
//如果 databases 目录不存在则创建
File file = new
File("/data/data/com.itheima.android.asserts.sqlite/databases");
if (!file.exists()) {
    file.mkdirs();
}
FileOutputStream fos = new FileOutputStream(new File(file, "myuser.db"));
byte[] buff = new byte[1024*8];
int len=-1;
while((len=is.read(buff))!=-1){
    fos.write(buff, 0, len);
}
```

```
    }  
    fos.close();  
    is.close();  
  
    //访问数据库  
    SQLiteDatabase database = openOrCreateDatabase("myuser.db", MODE_PRIVATE,  
null);  
    String sql = "select c_name from t_user";  
    Cursor cursor = database.rawQuery(sql, null);  
    while(cursor.moveToNext()){  
        String string = cursor.getString(0);  
        Log.d("tag", string);  
    }  
    cursor.close();  
    database.close();  
}
```

5、如何在高并发下进行数据库查询？（2015-11-25）

（这个问题的回答很广泛，可以自由发挥）

比如：不要关联多表查询，减少链接时间，创建索引、将查询到的数据采用缓存策略等等。

六、Android 中的布局

1、Android 中常用的布局都有哪些

FrameLayout

RelativeLayout

LinearLayout

AbsoluteLayout

TableLayout

GriddleLayout(Android 4.0 推出)

2、谈谈 UI 中，Padding 和 Margin 有什么区别？

android:padding 和 android:layout_margin 的区别，其实概念很简单，padding 是站在父 view 的角度描述问题，它规定它里面的内容必须与这个父 view 边界的距离。margin 则是站在自己的角度描述问题，规定自己和其他（上下左右）的 view 之间的距离，如果同一级只有一个 view，那么它的效果基本上就和 padding 一样了。

3、使用权重如何让一个控件的宽度为父控件的 1/3？

可以在水平方向的 LinearLayout 中设置 weightSum 为 3，然后让其子控件的 weight 为 1，那么该子控件就是父控件的 1/3。

4、Android 中布局的优化措施都有哪些？

1、尽可能减少布局的嵌套层级

可以使用 sdk 提供的 hierarchyviewer 工具分析视图树，帮助我们发现没有用到的布局。

2、不用设置不必要的背景，避免过度绘制

比如父控件设置了背景色，子控件完全将父控件给覆盖的情况下，那么父控件就没有必要设置背景。

3、使用 <include> 标签复用相同的布局代码

4、使用 <merge> 标签减少视图层次结构

该标签主要有两种用法：

1) 因为所有的 Activity 视图的根节点都是 FrameLayout，因此如果我们的自定义的布局也是 FragmentLayout 的时候那么可以使用 merge 替换。

2) 当应用 Include 或者 ViewStub 标签从外部导入 xml 结构时，可以将被导入的 xml 用 merge 作为根节点表示，这样当被嵌入父级结构中后可以很好的将它所包含的子集融合到父级结构中，而不会出现冗余的节点。

<merge> 只能作为 xml 布局的根元素。

5、通过<ViewStub>实现 View 的延迟加载

布局如下：

```
<ViewStub
    android:id="@+id/vs"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:inflatedId="@+id/my_view"
    android:layout="@layout/my_layout" />
```

核心代码：

```
public void loadVS(View view){
    ViewStub vs = (ViewStub) findViewById(R.id.vs);
    View inflate = vs.inflate();
    int inflatedId = vs.getInflatedId();
    int id = inflate.getId();
    Toast.makeText(this, "inflatedId="+inflatedId+"***"+id,
        Toast.LENGTH_SHORT).show();
}
```

5、android:layout_gravity 和 android:gravity 的区别？

第一个是让该布局在其父控件中的布局方式，第二个是该布局布置其子对象的布局方式。

七、ListView

1、ListView 如何提高其效率？

- ① 复用 convertView

- ② 自定义静态类 ViewHolder
- ③ 使用分页加载
- ④ 使用 WeakReference 引用 ImageView 对象

2、ViewHolder 为什么要声明为静态类？

非静态内部类拥有外部类对象的强引用，因此为了避免对外部类（外部类很可能是 Activity）对象的引用，那么最好将内部类声明为 static 的。

3、在 Activity 中使用 Handler 的时候如何去除警告信息？

将 Handler 类声明为 static，同时在 Handler 类中使用弱引用去引用 Context 对象。

```
1. static class MyHandler extends Handler {  
2.     private SoftReference<Context> srf;  
3.  
4.     public MyHandler(Context context) {  
5.         srf = new SoftReference<Context>(context);  
6.     }  
7.  
8.     @Override  
9.     public void handleMessage(Message msg) {  
10.         Toast.makeText(srf.get(), msg.toString(), Toast.LENGTH_SHORT).show();  
11.     }  
12. }
```

4、谈谈 ListView 中的 MVC 思想？

M : model

V : view

C : Controller

5、ListView 使用了哪些设计模式？

- 1、适配器
- 2、观察者
- 3、享元设计模式

6、当 ListView 数据集改变后，如何更新 ListView？

使用该 ListView 的 adapter 的 `notifyDataSetChanged()` 方法。该方法会使 ListView 重新绘制。

7、ListView 如何实现分页加载

- ① 设置 ListView 的滚动监听器：`setOnScrollListener(new OnScrollListener{....})`

在监听器中有两个方法：滚动状态发生变化的方法(`onScrollStateChanged`)和 listView 被滚动时调用的方法(`onScroll`)

- ② 在滚动状态发生改变的方法中，有三种状态：

手指按下移动的状态：`SCROLL_STATE_TOUCH_SCROLL`: // 触摸滑动

惯性滚动（滑翔（fling）状态）：`SCROLL_STATE_FLING`: // 滑翔

静止状态：`SCROLL_STATE_IDLE`: // 静止

对不同的状态进行处理：

分批加载数据，只关心静止状态：关心最后一个可见的条目，如果最后一个可见条目就是数据适配器（集合）里的最后一个，此时可加载更多的数据。在每次加载的时候，计算出滚动的数量，当滚动的数量大于等于总数量的时候，可以提示用户无更多数据了。

8、ListView 可以显示多种类型的条目吗？

这个当然可以的，ListView 显示的每个条目都是通过 baseAdapter 的 getView(int position, View convertView, ViewGroup parent)来展示的，理论上我们完全可以让每个条目都是不同类型的 view，除此之外 adapter 还提供了 getViewTypeCount () 和 getItemViewType(int position)两个方法。在 getView 方法中我们可以根据不同的 viewtype 加载不同的布局文件。

9、ListView 如何定位到指定位置

可以通过 ListView 提供的 lv.setSelection(48);方法。

10、如何在 ScrollView 中如何嵌入 ListView

通常情况下我们不会在 ScrollView 中嵌套 ListView，但是如果面试官非让我嵌套的话也是可以的。

在 ScrollView 添加一个 ListView 会导致 listview 控件显示不全，通常只会显示一条，这是因为两个控件的滚动事件冲突导致。所以需要通过 listview 中的 item 数量去计算 listview 的显示高度，从而使其完整展示，如下提供一个方法供大家参考。

如下图，在 ScrollView 中嵌套了 ListView，ListView 展现的是物流状态。

09:21

0.09 K/s  84

物流查询

订单编号 13283790161 已发货

配送公司： 晟邦物流-上海 4006666066

✓	2015-10-10 23:05:11 配送中（客户推迟收货），晟邦物流-上海 4006666066 晟邦物流-上海
✓	2015-10-10 22:34:53 货物已由[上海宽容S10]的小件员[宽容]反馈为[分站 滞留],电话[4006666066],原因:客户更改派送时间(通 用) 晟邦物流-上海
✓	2015-10-10 22:32:56 货物由[上海宽容S10]的派件员[宽容]正在派件..电话 [4006666066] 晟邦物流-上海
✓	2015-10-10 17:07:06 配送中，晟邦物流-上海 4006666066 上海市
	2015-10-10 05:25:20

猜你喜欢



```

        lv = (ListView) findViewById(R.id.lv);
        adapter = new MyAdapter();
        lv.setAdapter(adapter);
        setListViewHeightBasedOnChildren(lv);
    }

    public void setListViewHeightBasedOnChildren(ListView listView) {
        ListAdapter listAdapter = listView.getAdapter();
        if (listAdapter == null) {
            return;
        }
        int totalHeight = 0;
        for (int i = 0; i < listAdapter.getCount(); i++) {
            View listItem = listAdapter.getView(i, null, listView);
            listItem.measure(0, 0);
            totalHeight += listItem.getMeasuredHeight();
        }
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = totalHeight + (listView.getDividerHeight() *
(listAdapter.getCount() - 1));
        params.height += 5; // if without this statement, the listview will be a
                           // little short
        listView.setLayoutParams(params);
    }

```

布局文件片段:

```

1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:layout_width="match_parent"
3.     android:layout_height="match_parent"
4.     android:orientation="vertical" >
5.     <ScrollView
6.         android:id="@+id/sv"
7.         android:layout_width="match_parent"
8.         android:layout_height="0dp"
9.         android:layout_weight="1" >
10.
11.         <LinearLayout
12.             android:layout_width="match_parent"
13.             android:layout_height="match_parent"
14.             android:orientation="vertical" >
15.
16.             <ListView
17.                 android:id="@+id/lv"

```

```
18.         android:layout_width="match_parent"
19.         android:layout_height="wrap_content" >
20.     </ListView>
21. </LinearLayout>
22. </ScrollView>
23.
24. </LinearLayout>
```

注意:如果直接将 ListView 放到 ScrollView 中,那么上面的代码依然是没有效果的.必须将 ListView 放到 LinearLayout 等其他容器中才行.

11、ListView 中如何优化图片

图片的优化策略比较多。

1、处理图片的方式：

如果 ListView 中自定义的 Item 中有涉及到大量图片的，一定要对图片进行细心的处理，因为图片占的内存是 ListView 项中最头疼的，处理图片的方法大致有以下几种：

- ①、不要直接拿路径就去循环 `BitmapFactory.decodeFile`;使用 Options 保存图片大小、不要加载图片到内存去。
- ②、对图片一定要经过边界压缩尤其是比较大的图片，如果你的图片是后台服务器处理好的那就不需要了
- ③、在 ListView 中取图片时也不要直接拿个路径去取图片，而是以 WeakReference (使用 WeakReference 代替强引用。比如可以使用 `WeakReference mContextRef`)、`SoftReference`、`WeakHashMap` 等的来存储图片信息。
- ④、在 `getView` 中做图片转换时，产生的中间变量一定及时释放

2、异步加载图片基本思想：

- 1)、先从内存缓存中获取图片显示 (内存缓冲)
- 2)、获取不到的话从 SD 卡里获取 (SD 卡缓冲)
- 3)、都获取不到的话从网络下载图片并保存到 SD 卡同时加入内存并显示 (视情况看是否要显示)

原理：

优化一：先从内存中加载，没有则开启线程从 SD 卡或网络中获取，这里注意从 SD 卡获取图片是放在子线程里执行的，否则快速滚屏的话会不够流畅。

优化二：于此同时，在 adapter 里有个 busy 变量，表示 listview 是否处于滑动状态，如果是滑动状态则仅从内存中获取图片，没有的话无需再开启线程去外存或网络获取图片。

优化三：ImageLoader 里的线程使用了线程池，从而避免了过多线程频繁创建和销毁，如果每次总是 new 一个线程去执行这是非常不可取的，好一点的用的 AsyncTask 类，其实内部也是用到了线程池。在从网络获取图片时，先是将其保存到 sd 卡，然后再加载到内存，这么做的好处是在加载到内存时可以做个压缩处理，以减少图片所占内存。

12、ListView 中图片错位的问题是如何产生的

图片错位问题的本质源于我们的 listview 使用了缓存 convertView，假设一种场景，一个 listview 一屏显示九个 item，那么在拉出第十个 item 的时候，事实上该 item 是重复使用了第一个 item，也就是说在第一个 item 从网络中下载图片并最终要显示的时候，其实该 item 已经不在当前显示区域内了，此时显示的后果将可能在第十个 item 上输出图像，这就导致了图片错位的问题。所以解决之道在于可见则显示，不可见则不显示。

13、scrollView 嵌套 listview 方式除了测量还有什么方法？（2015-11-29）

1、手动设置 ListView 高度

经过测试发现，在 xml 中直接指定 ListView 的高度，是可以解决这个问题的，但是 ListView 中的数据是可变的，实际高度还需要实际测量。

于是手动代码设置 ListView 高度的方法就诞生了。

```
/**
```

```
 * 动态设置 ListView 的高度
```



```
* @param listView
*/

public static void setListViewHeightBasedOnChildren(ListView listView) {

    if(listView == null) return;

    ListAdapter listAdapter = listView.getAdapter();

    if (listAdapter == null) {

        // pre-condition

        return;

    }

    int totalHeight = 0;

    for (int i = 0; i < listAdapter.getCount(); i++) {

        View listItem = listAdapter.getView(i, null, listView);

        listItem.measure(0, 0);

        totalHeight += listItem.getMeasuredHeight();

    }

    ViewGroup.LayoutParams params = listView.getLayoutParams();

    params.height = totalHeight + (listView.getDividerHeight() * (listAdapter.getCount() - 1));

    listView.setLayoutParams(params);

}
```

2、使用单个 ListView 取代 ScrollView 中所有内容

如果满足头布局 and 脚布局的 UI 设计，直接使用 listview 替代 scrollview

3、使用 LinearLayout 取代 ListView

既然 ListView 不能适应 ScrollView，那就换一个可以适应 ScrollView 的控件，干嘛非要吊死在 ListView 这一棵树上呢？

而 LinearLayout 是最好的选择。但如果我仍想继续使用已经定义好的 Adapter 呢？我们只需要自定义一个类继承自 LinearLayout，为其加上对 BaseAdapter 的适配。

4、自定义可适应 ScrollView 的 ListView

这个方法和上面的方法是异曲同工，方法 3 是自定义了 LinearLayout 以取代 ListView 的功能，但如果我脾气就是倔，就是要用 ListView 怎么办？

那就只好自定义一个类继承自 ListView，通过重写其 onMeasure 方法，达到对 ScrollView 适配的效果。

5、参考博客

<http://bbs.anzhuo.cn/thread-982250-1-1.html>

八、JNI&NDK

1、在 Android 中如何调用 C 语言

当我们的 Java 需要调用 C 语言的时候可以通过 JNI 的方式，Java Native Interface。Android 提供了对 JNI 的支持，因此我们在 Android 中可以使用 JNI 调用 C 语言。在 Android 开发目录的 libs 目录下添加 xxx.so 文件，不过 xxx.so 文件需要放在对应的 CPU 架构名目录下，比如 armeabi，x86 等。

在 Java 代码需要通过 `System.loadLibrary(libName)`；加载 so 文件。同时 C 语言中的方法在 java 中必须以 native 关键字来声明。普通 Java 方法调用这个 native 方法接口，虚拟机内部自动调用 so 文件中对应的方法。

2、请介绍一下 NDK

1.NDK 是一系列工具的集合

NDK 提供了一系列的工具，帮助开发者快速开发 C (或 C++) 的动态库，并能自动将 so 和 java 应用一起打包成 apk。NDK 集成了交叉编译器，并提供了相应的 mk 文件隔离 CPU、平台、ABI 等差异，开发人员只需要简单修改 mk 文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出 so。

2.NDK 提供了一份稳定、功能有限的 API 头文件声明

Google 明确声明该 API 是稳定的，在后续所有版本中都稳定支持当前发布的 API。从该版本的 NDK 中看出，这些 API 支持的功能非常有限，包含有：C 标准库 (libc)、标准数学库 (libm)、压缩库 (libz)、Log 库 (liblog)。

3、JNI 调用常用的两个参数

JNIEnv *env, jobject obj

第一个是指向虚拟机对象的指针，是一个二级指针。里面封装了很多方法和中间变量供我们使用。

第二个代表着调用该方法的 Java 对象的 C 语言表示。

九、Android 中的网络访问

1、Android 中如何访问网络

Android 提供了 org.apache.http.HttpClientConnection 和 java.net.HttpURLConnection 两个连接网络对象。使用哪个都行，具体要看企业领导的要求了。

除此之外一般我比较喜欢使用 xUtils 中的 HttpUtils 功能，该模块底层使用的就是 org.apache.http.client.HttpClient，使用起来非常方便。

2、如何解析服务器传来的 JSON 文件

在 Android 中内置了 JSON 的解析 API，在 org.json 包中包含了如下几个类：JSONArray、JSONObject、JSONStringer、JSONTokener 和一个异常类 JSONException。

1、JSON 解析步骤

1)、读取网络文件数据并转为一个 json 字符串

```
InputStream in = conn.getInputStream();
```

```
String jsonStr = DataUtil.Stream2String(in); //将流转换成字符串的工具类
```

2)、将字符串传入相应的 JSON 构造函数中

①、通过构造函数将 json 字符串转换成 json 对象

```
JSONObject jsonObject = new JSONObject(jsonStr);
```

②、通过构造函数将 json 字符串转换成 json 数组：

```
JSONArray array = new JSONArray(jsonStr);
```

3)、解析出 JSON 中的数据信息：

①、从 json 对象中获取你所需要的键所对应的值

```
JSONObject json=jsonObject.getJSONObject("weatherinfo");
```

```
String city = json.getString("city");
```

```
String temp = json.getString("temp")
```

②、遍历 JSON 数组，获取数组中每一个 json 对象，同时可以获取 json 对象中键对应的值

```
for (int i = 0; i < array.length(); i++) {
```

```
    JSONObject obj = array.getJSONObject(i);
```

```
    String title=obj.getString("title");
```

```
String description=obj.getString("description");  
  
}
```

2、生成 JSON 对象和数组

1) 生成 JSON :

方法 1、创建一个 map，通过构造方法将 map 转换成 json 对象

```
Map<String, Object> map = new HashMap<String, Object>();  
  
map.put("name", "zhangsan");  
  
map.put("age", 24);  
  
JSONObject json = new JSONObject(map);
```

方法 2、创建一个 json 对象，通过 put 方法添加数据

```
JSONObject json=new JSONObject();  
  
json.put("name", "zhangsan");  
  
json.put("age", 24);
```

2) 生成 JSON 数组 :

创建一个 list，通过构造方法将 list 转换成 json 对象

```
Map<String, Object> map1 = new HashMap<String, Object>();  
  
map1.put("name", "zhangsan");  
  
map1.put("age", 24);  
  
Map<String, Object> map2 = new HashMap<String, Object>();  
  
map2.put("name", "lisi");  
  
map2.put("age", 25);  
  
List<Map<String, Object>> list=new ArrayList<Map<String,Object>>();
```

```
list.add(map1);

list.add(map2);

JSONArray array=new JSONArray(list);

System.out.println(array.toString());
```

3、如何解析服务器传来的 XML 格式数据

Android 为我们提供了原生的 XML 解析和生成支持。

1、XML 解析

获取解析器: `Xml.newPullParser()`

设置输入流: `setInput()`

获取当前事件类型: `getEventType()`

解析下一个事件, 获取类型: `next()`

获取标签名: `getName()`

获取属性值: `getAttributeValue()`

获取下一个文本: `nextText()`

获取当前文本: `getText()`

5 种事件类型: `START_DOCUMENT`, `END_DOCUMENT`, `START_TAG`, `END_TAG`, `TEXT`

示例代码：

```
public List<Person> getPersons(InuptStream in){

    XmlPullParser parser=Xml.newPullParser();//获取解析器

    parser.setInput(in,"utf-8");

    for(int type=){    //循环解析
```

```
}
```

```
}
```

2、XML 生成

获取生成工具: `Xml.newSerializer()`

设置输出流: `setOutput()`

开始文档: `startDocument()`

结束文档: `endDocument()`

开始标签: `startTag()`

结束标签: `endTag()`

属性: `attribute()`

文本: `text()`

示例代码：

```
XmlSerializer serial=Xml.newSerializer();//获取 xml 序列化工具
```

```
serial.setOutput(out,"utf-8");
```

```
serial.startDocument("utf-8",true);
```

```
serial.startTag(null,"persons");
```

```
for(Person p:persons){
```

```
    serial.startTag(null,"persons");
```

```
    serial.attribute(null,"id",p.getId().toString());
```

```
    serial.startTag(null,"name");
```

```
    serial.attribute(null,"name",p.getName().toString());
```

```
    serial.endTag(null,"name");
```

```
serial.startTag(null,"age");

serial.attribute(null,"age",p.getAge().toString());

serial.endTag(null,"age");

serial.endTag(null,"persons");

}
```

4、如何从网络上加载一个图片显示到界面

可以通过 `BitmapFactory.decodeStream(inputStream);` 方法将图片转换为 bitmap，然后通过 `imageView.setImageBitmap(bitmap);` 将该图片设置到 `ImageView` 中。这是原生的方法，还可以使用第三方开源的工具来实现，比如使用 `SmartImageView` 作为 `ImageView` 控件，然后直接设置一个 url 地址即可。也可以使用 `xUtils` 中的 `BitmapUtils` 工具。

5、如何播放网络视频

除了使用 `Android` 提供的 `MediaPlayer` 和 `VideoView` 外通常还可以使用第三方开源万能播放器，`VitamioPlayer`。该播放器兼容性好，支持几乎所有主流视频格式。

6、常见的访问网络 API 都有哪些？

`Android` 系统自带的 `URLConnection/HttpClient`。

`java.net.HttpURLConnection;`

`org.apache.http.client.HttpClient;`

`OKhttp;`

<http://blog.csdn.net/lmj623565791/article/details/47911083>

`xUtils :`

<https://github.com/wyouflf/xUtils>

十、Intent

1、Intent 传递数据时，可以传递哪些类型数据？

Intent 可以传递的数据类型非常的丰富，java 的基本数据类型和 String 以及他们的数组形式都可以，除此之外还可以传递实现了 Serializable 和 Parcelable 接口的对象。

2、Serializable 和 Parcelable 的区别

- 1.在使用内存的时候，Parcelable 类比 Serializable 性能高，所以推荐使用 Parcelable 类。
- 2.Serializable 在序列化的时候会产生大量的临时变量，从而引起频繁的 GC。
- 3.Parcelable 不能使用在要将数据存储在磁盘上的情况。尽管 Serializable 效率低点，但在这种情况下，还是建

议你用 Serializable 。

实现：

- 1 Serializable 的实现，只需要继承 Serializable 即可。这只是给对象打了一个标记，系统会自动将其序列化。
- 2 Parcelabel 的实现，需要在类中添加一个静态成员变量 CREATOR，这个变量需要继承 Parcelable.Creator 接

口。

```
public class MyParcelable implements Parcelable {
    private int mData;

    public int describeContents() {
        return 0;
    }

    public void writeToParcel(Parcel out, int flags) {
        out.writeInt(mData);
    }

    public static final Parcelable.Creator<MyParcelable> CREATOR
        = new Parcelable.Creator<MyParcelable>() {
        public MyParcelable createFromParcel(Parcel in) {
            return new MyParcelable(in);
        }

        public MyParcelable[] newArray(int size) {
            return new MyParcelable[size];
        }
    };

    private MyParcelable(Parcel in) {
        mData = in.readInt();
    }
}
```

3、请描述一下 Intent 和 IntentFilter

Android 中通过 Intent 对象来表示一条消息，一个 Intent 对象不仅包含有这个消息的目的地，还可以包含消息的内容，这好比一封 Email，其中不仅应该包含收件地址，还可以包含具体的内容。对于一个 Intent 对象，消息“目的地”是必须的，而内容则是可选项。

通过 Intent 可以实现各种系统组件的调用与激活。

IntentFilter: 可以理解为邮局或者是一个信笺的分拣系统...

这个分拣系统通过 3 个参数来识别

Action: 动作 view

Data: 数据 uri uri

Category：而外的附加信息

Action 匹配

Action 是一个用户定义的字符串，用于描述一个 Android 应用程序组件，一个 IntentFilter 可以包含多个 Action。在 AndroidManifest.xml 的 Activity 定义时可以在其 <intent-filter> 节点指定一个 Action 列表用于标示 Activity 所能接受的“动作”，例如：

```
<intent-filter >

<action android:name="android.intent.action.MAIN" />

<action android:name="cn.itheima.action" />

.....

</intent-filter>
```

如果我们在启动一个 Activity 时使用这样的 Intent 对象：

```
Intent intent =new Intent();

intent.setAction("cn.itheima.action");
```

那么所有的 Action 列表中包含了“cn.itheima”的 Activity 都将会匹配成功。

Android 预定义了一系列的 Action 分别表示特定的系统动作。这些 Action 通过常量的方式定义在 android.content.Intent 中，以“ACTION_”开头。我们可以在 Android 提供的文档中找到它们的详细说明。

URI 数据匹配

一个 Intent 可以通过 URI 携带外部数据给目标组件。在 <intent-filter> 节点中，通过 <data/> 节点匹配外部数据。

mimeType 属性指定携带外部数据的数据类型，scheme 指定协议，host、port、path 指定数据的位置、端口、

和路径。如下：

```
<data android:mimeType="mimeType" android:scheme="scheme"
android:host="host" android:port="port" android:path="path"/>
```

电话的 uri tel: 12345

http://www.baidu.com

自己定义的 uri itcast://cn.itcast/person/10

如果在 Intent Filter 中指定了这些属性，那么只有所有的属性都匹配成功时 URI 数据匹配才会成功。

Category 类别匹配

<intent-filter >节点中可以为组件定义一个 Category 类别列表，当 Intent 中包含这个列表的所有项目时

Category 类别匹配才会成功。

4、under what condition could the code sample below crash your application?How would you modify the code to avoid this potential problem?Explain your answer?(重要)

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT,textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYEP);/"text/plain" MIME type
context.startActivity(sendIntent);
```

answer:

if startActivity(Intent) method is not called from in Activity ,for example,from service or receiver,it will crash your application.

I can add a flag "Intent_ACTIVITY_NEW_TASK" on sendIntent to avoid this error.

5、 what are Activity and Fragment?where and why should you use one over the other?

6、 can you use an intent to provide data to a ContentProvider ? if not ,what would be the proper mechanism for doing this?

SQLiteDatabase is often used to provide data to a ContentProvider.

十一、Fragment

1、Fragment 跟 Activity 之间是如何传值的？

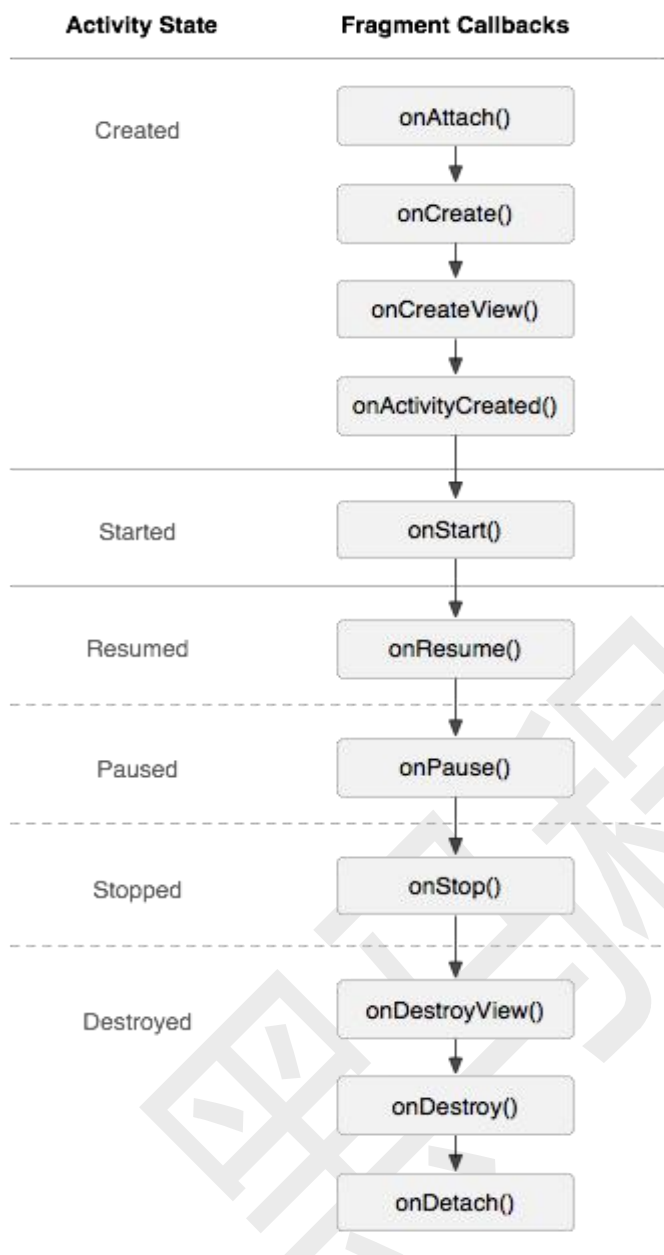
1) 当 Fragment 跟 Activity 绑定之后，在 Fragment 中可以直接通过 getActivity () 方法获取到其绑定的 Activity 对象，这样就可以调用 Activity 的方法了。在 Activity 中可以通过如下方法获取到 Fragment 实例

```
FragmentManager fragmentManager = getFragmentManager();  
Fragment fragment = fragmentManager.findFragmentByTag(tag);  
Fragment fragment = fragmentManager.findFragmentById(id);
```

获取到 Fragment 之后就可以调用 Fragment 的方法。也就实现了通信功能。

2) 另外也可以调用 fragment.setArguments(Bundle)方法，将数据绑定到 Fragment 域中。

2、描述一下 Fragment 的生命周期



3、Fragment 的 replace 和 add 方法的区别 (2015.8.30)

Fragment 本身并没有 replace 和 add 方法，这里的理解应该为使用 FragmentManager 的 replace 和 add 两种方法切换 Fragment 时有什么不同。

我们经常使用的一个架构就是通过 RadioGroup 切换 Fragment，每个 Fragment 就是一个功能模块。

```
        case R.id.rb_1:
            rb_1.setBackgroundColor(Color.RED);
            transaction.show(fragment1);
            transaction.replace(R.id.fl, fragment1, "Fragment1");
            break;
        case R.id.rb_2:
            rb_2.setBackgroundColor(Color.YELLOW);
            transaction.replace(R.id.fl, fragment2, "Fragment2");
            transaction.show(fragment2);
            break;
        case R.id.rb_3:
            rb_3.setBackgroundColor(Color.BLUE);
            transaction.replace(R.id.fl, fragment3, "Fragment3");
            transaction.show(fragment3);
            break;
```

实现这个功能可以通过 replace 和 add 两种方法。

Fragment 的容器一个 FrameLayout，add 的时候是把所有的 Fragment 一层一层的叠加到了 FrameLayout 上了，而 replace 的话首先将该容器中的其他 Fragment 去除掉然后将当前 Fragment 添加到容器中。

一个 Fragment 容器中只能添加一个 Fragment 种类，如果多次添加则会报异常，导致程序终止，而 replace 则无所谓，随便切换。

因为通过 add 的方法添加的 Fragment，每个 Fragment 只能添加一次，因此如果要想达到切换效果需要通过 Fragment 的 hide 和 show 方法结合者使用。将要显示的 show 出来，将其他 hide 起来。这个过程 Fragment 的生命周期没有变化。

通过 replace 切换 Fragment，每次都会执行上一个 Fragment 的 onDestroyView，新 Fragment 的 onCreateView、onStart、onResume 方法。

基于以上不同的特点我们在使用的使用一定要结合着生命周期操作我们的视图和数据。

4、Fragment 如何实现类似 Activity 栈的压栈和出栈效果的？（2015.8.30）

Fragment 的事物管理器内部维持了一个双向链表结构，该结构可以记录我们每次 add 的 Fragment 和 replace

的 Fragment，然后当我们点击 back 按钮的时候会自动帮我们实现退栈操作。

Add this transaction to the back stack. This means that the transaction will be remembered after it is committed, and will reverse its operation when later popped off the stack.

Parameters:

name An optional name for this back stack state, or null.

```
transaction.addToBackStack("name");
```

//实现源码 在 BackStackRecord 中

```
public FragmentTransaction addToBackStack(String name) {  
    if (!mAllowAddToBackStack) {  
        throw new IllegalStateException(  
            "This FragmentTransaction is not allowed to be added to the back  
stack.");  
    }  
    mAddToBackStack = true;  
    mName = name;  
    return this;  
}
```

//上面的源码仅仅做了一个标记

除此之外因为我们要使用 FragmentManger 用的是 FragmentActivity 因此 FragmentActivity 的 onBackPressed 方法必定重新覆写了。打开看一下，发现确实如此。

```
/**  
 * Take care of popping the fragment back stack or finishing the activity  
 * as appropriate.  
 */  
public void onBackPressed() {  
    if (!mFragments.popBackStackImmediate()) {  
        finish();  
    }  
}  
//mFragments 的原型是 FragmentManagerImpl，看看这个方法都干嘛了
```



```
@Override
public boolean popBackStackImmediate() {
    checkStateLoss();
    executePendingTransactions();
    return popBackStackState(mActivity mHandler, null, -1, 0);
}
```

//看看 popBackStackState 方法都干了啥，其实通过名称也能大概了解 只给几个片段吧，代码太多了

```
while (index >= 0) {
    //从后退栈中取出当前记录对象
    BackStackRecord bss = mBackStack.get(index);
    if (name != null && name.equals(bss.getName())) {
        break;
    }
    if (id >= 0 && id == bss.mIndex) {
        break;
    }
    index--;
}
```

5、ViewPager+Fragment 的左右滑动，如何实现 Fragment 的懒加载，Viewpager 默认加载几个？（2017-2-24）

Viewpager 默认加载 3 个。1 个 Activity 里面可能会以 Viewpager(或其他容器)与多个 Fragment 来组合使用，而如果每个 fragment 都需要去加载数据，或从本地加载，或从网络加载，那么在这个 Activity 刚创建的时候就变成需要初始化大量资源。所以我们要进行懒加载。

方法比较多，这里给大家提供两种方案：

方案 1：在 Fragment 里的 setUserVisibleHint，该方法用于告诉系统，这个 Fragment 的 UI 是否是可见的。所以我们只需要继承 Fragment 并重写该方法，即可实现在 Fragment 可见时才进行数据加载操作，即 Fragment 的懒加载。实现代码：

```
1.package fragment;
2.import android.support.v4.app.Fragment;
```

```

3. /**
4.  * Fragment 懒加载
5.  */
6. public abstract class LazyFragment extends Fragment{
7.     protected boolean isVisible;
8.     @Override
9.     //fragment 从不可见到完全可见的时候，会调用该方法
10.    public void setUserVisibleHint(boolean isVisibleToUser) {
11.        super.setUserVisibleHint(isVisibleToUser);
12.        if (getUserVisibleHint()){
13.            isVisible = true;
14.            onVisible();//可见
15.        }else {
16.            isVisible = false;
17.            onInvisible();//不可见
18.        }
19.    }
20.    //懒加载的方法,在这个方法里面我们为 Fragment 的各个组件去添加数据
21.    protected abstract void lazyLoad();
22.    protected void onVisible(){
23.        lazyLoad();
24.    }
25.    protected void onInvisible(){
26.    }
27. }

```

使用：

```

1. public class Fragment extends LazyFragment {
2.     private boolean isPrepared; // 标志位，标志已经初始化完成。
3.     //在这个方法里面去给我们的 Fragment 添加数据
4.     @Override
5.     protected void lazyLoad() {
6.         if (isPrepared && isVisible){
7.             getNewsDate(getActivity(), channelId);
8.             page++;
9.             isPrepared = false;
10.        }
11.    }
12.    @Override
13.    public View onCreateView(LayoutInflater inflater,
14.        ViewGroup container, Bundle savedInstanceState) {
15.        View view = LayoutInflater.from(
16.            getActivity()).inflate(R.layout.fragment, container, false);
17.        isPrepared = true;

```

```
18.         lazyLoad();//这里我们调用以下去加载我们的数据
19.         return view;
20. }
```

方案2:在继承 `FragmentStatePagerAdapter` 重写以下的方法,返回 `PagerAdapter.POSITION_NONE`;

```
/**
 * Called when the host view is attempting to determine if an item's position
 * has changed. Returns {@link #POSITION_UNCHANGED} if the position of the given
 * item has not changed or {@link #POSITION_NONE} if the item is no longer present
 * in the adapter.
 *
 * <p>The default implementation assumes that items will never
 * change position and always returns {@link #POSITION_UNCHANGED}.
 *
 * @param object Object representing an item, previously returned by a call to
 *        {@link #instantiateItem(View, int)}.
 * @return object's new position index from [0, {@link #getCount()}),
 *        {@link #POSITION_UNCHANGED} if the object's position has not changed,
 *        or {@link #POSITION_NONE} if the item is no longer present.
 */

//这个参数为不进行预加载 adapter

@Override
public int getItemPosition(Object object) {
    return PagerAdapter.POSITION_NONE;
}
```

5. Android 高级 (★★★★)

一、Android 性能优化

1、如何对 Android 应用进行性能分析

一款 App 流畅与否安装在自己的真机里，玩几天就能有个大概的感性认识。不过通过专业的分析工具可以使我们的应用。

如果不考虑使用其他第三方性能分析工具的话，我们可以直接使用 ddms 中的工具，其实 ddms 工具已经非常的强大了。ddms 中有 traceview、heap、allocation tracker 等工具都可以帮助我们分析应用的方法执行时间效率和内存使用情况。

◆ [traceview](#) 工具在本文中已经有详细的介绍，因此这里就不再赘述。

◆ heap

heap 工具可以帮助我们检查代码中是否存在会造成内存泄漏的地方。

用 heap 监测应用进程使用内存情况的步骤如下：

1. 启动 eclipse 后，切换到 DDMS 透视图，并确认 Devices 视图、Heap 视图都是打开的；
2. 点击选中想要监测的进程，比如 system_process 进程；
3. 点击选中 Devices 视图界面中最上方一排图标中的“Update Heap”图标；
6. 点击 Heap 视图中的“Cause GC”按钮；
7. 此时在 Heap 视图中就会看到当前选中的进程的内存使用量的详细情况。

说明：

- a) 点击“Cause GC”按钮相当于向虚拟机请求了一次 gc 操作；
- b) 当内存使用信息第一次显示以后，无须再不断的点击“Cause GC”，Heap 视图界面会定时刷新，在对应用的不断的操作过程中就可以看到内存使用的变化；
- c) 内存使用信息的各项参数根据名称即可知道其意思，在此不再赘述。

如何才能知道我们的程序是否有内存泄漏的可能性呢。这里需要注意一个值：Heap 视图中部有一个 Type 叫做 data object，即数据对象，也就是我们的程序中大量存在的类类型的对象。在 data object 一行中有一列是“Total Size”，其值就是当前进程中所有 Java 数据对象的内存总量，一般情况下，这个值的大小决定了是否有内存泄漏。可以这样判断：

- a) 不断的操作当前应用，同时注意观察 data object 的 Total Size 值；

b) 正常情况下 Total Size 值都会稳定在一个有限的范围内，也就是说由于程序中的代码良好，没有造成对象不被垃圾回收的情况，所以说虽然我们不断的操作会不断的生成很多对象，而在虚拟机不断的进行 GC 的过程中，这些对象都被回收了，内存占用量会落到一个稳定的水平；

c) 反之如果代码中存在没有释放对象引用的情况，则 data object 的 Total Size 值在每次 GC 后不会有明显的回落，随着操作次数的增多 Total Size 的值会越来越大，

直到到达一个上限后导致进程被 kill 掉。

d) 此处以 system_process 进程为例，在我的测试环境中 system_process 进程所占用的内存的 data object 的 Total Size 正常情况下会稳定在 2.2~2.8 之间，而当其值超过 3.55 后进程就会被 kill。

总之，使用 DDMS 的 Heap 视图工具可以很方便的确认我们的程序是否存在内存泄漏的可能性。

◆ allocation tracker

运行 DDMS，只需简单的选择应用进程并单击 Allocation tracker 标签，就会打开一个新的窗口，单击 “Start Tracing” 按钮；

然后，让应用运行你想分析的代码。运行完毕后，单击 “Get Allocations” 按钮，一个已分配对象的列表就会出现第一个表格中。

单击第一个表格中的任何一项，在表格二中就会出现导致该内存分配的栈跟踪信息。通过 allocation tracker，不仅知道分配了哪类对象，还可以知道在哪个线程、哪个类、哪个文件的哪一行。

2、什么情况下会导致内存泄露

Android 的虚拟机是基于寄存器的 Dalvik，它的最大堆大小一般是 16M，有的机器为 24M。因此我们所能利用的内存空间是有限的。如果我们的内存占用超过了一定的水平就会出现 OutOfMemory 的错误。

内存溢出的几点原因：

1、资源释放问题

程序代码的问题，长期保持某些资源，如 Context、Cursor、IO 流的引用，资源得不到释放造成内存泄露。

2、对象内存过大问题

保存了多个耗用内存过大的对象（如 Bitmap、XML 文件），造成内存超出限制。

3、static 关键字的使用问题

static 是 Java 中的一个关键字，当用它来修饰成员变量时，那么该变量就属于该类，而不是该类的实例。所以用 static 修饰的变量，它的生命周期是很长的，如果用它来引用一些资源耗费过多的实例（Context 的情况最多），这时就要谨慎对待了。

```
public class ClassName {  
  
    private static Context mContext;  
  
    //省略  
  
}
```

以上的代码是很危险的，如果将 Activity 赋值到 mContext 的话。那么即使该 Activity 已经 onDestroy，但是由于仍有对象保存它的引用，因此该 Activity 依然不会被释放。

我们举 Android 文档中的一个例子。

```
private static Drawable sBackground;
@Override
protected void onCreate(Bundle state) {
    super.onCreate(state);
    TextView label = new TextView(this); //getApplicationContext
    label.setText("Leaks are bad");
    if (sBackground == null) {
        sBackground = getDrawable(R.drawable.large_bitmap);
    }
    label.setBackgroundDrawable(sBackground);
    setContentView(label);
}
```

sBackground 是一个静态的变量 ,但是我们发现 ,我们并没有显式的保存 Context 的引用 ,但是 ,当 Drawable 与 View 连接之后 ,Drawable 就将 View 设置为一个回调 ,由于 View 中是包含 Context 的引用的 ,所以 ,实际上我们依然保存了 Context 的引用。这个引用链如下 :

Drawable->TextView->Context

所以 ,最终该 Context 也没有得到释放 ,发生了内存泄露。

◆ 针对 static 的解决方案

① 应该尽量避免 static 成员变量引用资源耗费过多的实例 ,比如 Context。

② Context 尽量使用 ApplicationContext ,因为 Application 的 Context 的生命周期比较长 ,引用它不会出现内存泄露的问题。

③ 使用 WeakReference 代替强引用。比如可以使用 WeakReference<Context> mContextRef;

4、线程导致内存溢出

线程产生内存泄露的主要原因在于线程生命周期的不可控。我们来考虑下面一段代码。

```
public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        new MyThread().start();
    }
    private class MyThread extends Thread{
    @Override
        public void run() {
            super.run();
            //do something while(true)
        }
    }
}
```

这段代码很平常也很简单，是我们经常使用的形式。我们思考一个问题：假设 MyThread 的 run 函数是一个很费时的操作，当我们开启该线程后，将设备的横屏变为了竖屏，一般情况下当屏幕转换时会重新创建 Activity，按照我们的想法，老的 Activity 应该会被销毁才对，然而事实上并非如此。

由于我们的线程是 Activity 的内部类，所以 MyThread 中保存了 Activity 的一个引用，当 MyThread 的 run 函数没有结束时，MyThread 是不会被销毁的，因此它所引用的老的 Activity 也不会被销毁，因此就出现了内存泄露的问题。

有些人喜欢用 Android 提供的 AsyncTask，但事实上 AsyncTask 的问题更加严重，Thread 只有在 run 函数不结束时才出现这种内存泄露问题，然而 AsyncTask 内部的实现机制是运用了 ThreadPoolExecutor，该类产生的 Thread 对象的生命周期是不确定的，是应用程序无法控制的，因此如果 AsyncTask 作为 Activity 的内部类，就更容易出现内存泄露的问题。

针对这种线程导致的内存泄露问题的解决方案：

第一、将线程的内部类，改为静态内部类（因为非静态内部类拥有外部类对象的强引用，而静态类则不拥有）。

第二、在线程内部采用弱引用保存 Context 引用。

3、如何避免 OOM 异常

想要避免 OOM 异常首先我们要知道什么情况下会导致 OOM 异常。

1、图片过大导致 OOM

Android 中用 bitmap 时很容易内存溢出，比如报如下错误：Java.lang.OutOfMemoryError : bitmap size exceeds VM budget.

解决方法：

方法 1：等比例缩小图片

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inSampleSize = 2;
//Options 只保存图片尺寸大小，不保存图片到内存
BitmapFactory.Options opts = new BitmapFactory.Options();
opts.inSampleSize = 2;
Bitmap bmp = null;
bmp = BitmapFactory.decodeResource(getResources(),
mImageIds[position],opts);
//回收
bmp.recycle();
```

以上代码可以优化内存溢出，但它只是改变图片大小，并不能彻底解决内存溢出。

方法 2：对图片采用软引用，及时地进行 recycle()操作

```
SoftReference<Bitmap> bitmap = new SoftReference<Bitmap>(pBitmap);
if(bitmap != null){
    if(bitmap.get() != null && !bitmap.get().isRecycled()){
        bitmap.get().recycle();
        bitmap = null;
    }
}
```

2、界面切换导致 OOM

有时候我们会发现这样的问题，横竖屏切换 N 次后 OOM 了。

这种问题没有固定的解决方法，但是我们可以从以下几个方面下手分析。

1、看看页面布局当中有没有大的图片，比如背景图之类的。

去除 xml 中相关设置，改在程序中设置背景图（放在 onCreate()方法中）：

```
Drawable drawable = getResources().getDrawable(R.drawable.id);
ImageView imageView = new ImageView(this);
imageView.setBackgroundDrawable(drawable);
```

在 Activity destroy 时注意，drawable.setCallback(**null**)；防止 Activity 得不到及时的释放。

2、跟上面方法相似，直接把 xml 配置文件加载成 view 再放到一个容器里，然后直接调用 this.setContentView(View view);方法，避免 xml 的重复加载。

3、在页面切换时尽可能少地重复使用一些代码

比如：重复调用数据库，反复使用某些对象等等.....

常见的内存使用不当的情况

3、查询数据库没有关闭游标

程序中经常会进行查询数据库的操作，但是经常会有使用完毕 Cursor 后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在长时间大量操作的情况下才会出现内存问题，这样就会给以后的测试和问题排查带来困难和风险。

4、构造 Adapter 时，没有使用缓存的 convertView

在使用 ListView 的时候通常会使用 Adapter，那么我们应该尽可能的使用 convertView。

5、Bitmap 对象不再使用时调用 recycle() 释放内存

有时我们会手工的操作 Bitmap 对象，如果一个 Bitmap 对象比较占内存，当它不再被使用的时候，可以调用 Bitmap.recycle()方法回收此对象的像素所占用的内存，但这不是必须的，视情况而定。

6、其他

Android 应用程序中最典型的需要注意释放资源的情况是在 Activity 的生命周期中，在 onPause()、onStop()、onDestroy()方法中需要适当的释放资源的情况。

4、Android 中如何捕获未捕获的异常(重要)

- 1、自定义一个 Application，比如叫 MyApplication 继承 Application 实现 UncaughtExceptionHandler。
- 2、覆写 UncaughtExceptionHandler 的 onCreate 和 uncaughtException 方法。

```
@Override
public void onCreate() {
    super.onCreate();
    Thread.setDefaultUncaughtExceptionHandler(this);
}

@Override
public void uncaughtException(final Thread thread, final Throwable ex) {
    new Thread(new Runnable() {

        @Override
        public void run() {
            Looper.prepare();
            System.out.println(Thread.currentThread());
            Toast.makeText(getApplicationContext(), "thread="+thread.getId()+"
ex="+ex.toString(), 1).show();
            Looper.loop();
        }
    }).start();
    SystemClock.sleep(3000);
    android.os.Process.killProcess(android.os.Process.myPid());
}
}
```

注意：上面的代码只是简单的将异常打印出来。

在 onCreate 方法中我们给 Thread 类设置默认异常处理 handler，如果这句代码不执行则一切都是白搭。

在 uncaughtException 方法中我们必须新开辟个线程进行我们异常的收集工作，然后将系统给杀死。

3、在 AndroidManifest 中配置该 Application

```
<application
    android:name="com.example.uncatchexception.MyApplication"
```

4、blog 分享

关于异常数据的收集在网上有一篇不错的 blog 可以推荐给大家。

<http://blog.csdn.net/jdsjlzx/article/details/7606423>

5、Android 性能优化博客

给大家推荐一篇 Android 性能优化的 blog，其目录如下：

1.背景

2.应用 UI 性能问题分析

2-1 应用 UI 卡顿原理

2-2 应用 UI 卡顿常见原因

2-3 应用 UI 卡顿分析解决方法

2-3-1 使用 HierarchyViewer 分析 UI 性能

2-3-2 使用 GPU 过度绘制分析 UI 性能

2-3-3 使用 GPU 呈现模式图及 FPS 考核 UI 性能

2-3-4 使用 Lint 进行资源及冗余 UI 布局等优化

2-3-5 使用 Memory 监测及 GC 打印与 Allocation Tracker 进行 UI 卡顿分析

2-3-6 使用 Traceview 和 dmtracedump 进行分析优化

2-3-7 使用 Systrace 进行分析优化

2-3-8 使用 trace.txt 文件进行 ANR 分析优化

2-4 应用 UI 性能分析解决总结

3.应用开发 Memory 内存性能分析优化

3-1 Android 内存管理原理

3-2 Android 内存泄露性能分析

3-2-1 Android 应用内存泄露概念

3-2-2 Android 应用内存泄露察觉手段

3-2-3 Android 应用内存泄露 leakcanary 工具定位分析

3-2-4 Android 应用内存泄露 MAT 工具定位分析

3-2-5 Android 应用开发规避内存泄露建议

3-3 Android 内存溢出 OOM 性能分析

3-3-1 Android 应用内存溢出 OOM 概念

3-3-2 Android 应用内存溢出 OOM 性能分析

3-3-3 Android 应用规避内存溢出 OOM 建议

3-4 Android 内存性能优化总结

4.Android 应用 API 使用及代码逻辑性能分析

4-1 Android 应用 StringStringBuilderStringBuffer 优化建议

4-2 Android 应用 OnTrimMemory 实现性能建议

4-3 Android 应用 HashMap 与 ArrayMap 及 SparseArray 优化建议

4-4 Android 应用 ContentProviderOperation 优化建议

4-5 Android 应用其他逻辑优化建议

5.Android 应用移动设备电池耗电性能分析

5-1 Android 应用耗电量概念

5-2 Android 应用耗电量优化建议

6.Android 应用开发性能优化总结

blog 原文如下：

<http://blog.csdn.net/yanbober/article/details/48394201>

6、Android 动态加载机制

在目前的软硬件环境下，Native App 与 Web App 在用户体验上有着明显的优势，但在实际项目中有些会因为业务的频繁变更而频繁的升级客户端，造成较差的用户体验，而这也恰恰是 Web App 的优势。

Android 应用开发在一般情况下，常规的开发方式和代码架构就能满足我们的普通需求。但是有些特殊问题，常常引发我们进一步的沉思。我们从沉思中产生顿悟，从而产生新的技术形式。

如何开发一个可以自定义控件的 Android 应用？就像 eclipse 一样，可以动态加载插件；如何让 Android 应用执行服务器上的不可预知的代码？如何对 Android 应用加密，而只在执行时自解密，从而防止被破解？.....

熟悉 Java 技术的朋友，可能意识到，我们需要使用类加载器灵活的加载执行的类。这在 Java 里已经算是一项比较成熟的技术了，但是在 Android 中，我们大多数人都还非常陌生。

类加载机制

Dalvik 虚拟机如同其他 Java 虚拟机一样，在运行程序时首先需要将对应的类加载到内存中。而在 Java 标准的虚拟机中，类加载可以从 class 文件中读取，也可以是其他形式的二进制流，因此，我们常常利用这一点，在程序运行时手动加载 Class，从而达到代码动态加载执行的目的。

然而 Dalvik 虚拟机毕竟不算是标准的 Java 虚拟机，因此在类加载机制上，它们有相同的地方，也有不同之处。我们必须区别对待。

例如，在使用标准 Java 虚拟机时，我们经常自定义继承自 ClassLoader 的类加载器。然后通过 defineClass

方法来从一个二进制流中加载 Class。然而，这在 Android 里是行不通的，大家就没必要走弯路了。参看源码我们知道，Android 中 ClassLoader 的 defineClass 方法具体是调用 VMClassLoader 的 defineClass 本地静态方法。而这个本地方法除了抛出一个 “UnsupportedOperationException” 之外，什么都没做，甚至连返回值都为空。

```
1. static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args,JValue* pResult){
2.     Object* loader = (Object*) args[0];
3.     StringObject* nameObj = (StringObject*) args[1];
4.     const u1* data = (const u1*) args[2];
5.     int offset = args[3];
6.     int len = args[4];
7.     Object* pd = (Object*) args[5];
8.     char* name = NULL;
9.     name = dvmCreateCstrFromString(nameObj);
10.    LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n",loader, name, data, offset, len, pd);
11.    dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of
class file");
12.    free(name);
13.    RETURN_VOID();
14. }
```

Dalvik 虚拟机类加载机制

那如果在 Dalvik 虚拟机里 ,ClassLoader 不好使 ,我们如何实现动态加载类呢？Android 为我们从 ClassLoader 派生出了两个类：DexClassLoader 和 PathClassLoader。其中需要特别说明的是 PathClassLoader 中一段被注释掉的代码：

```
1. /* --this doesn't work in current version of Dalvik--
2.     if (data != null) {
3.         System.out.println("--- Found class " + name
4.             + " in zip[" + i + "] '" + mZips[i].getName() + "'");
5.         int dotIndex = name.lastIndexOf('.');
6.         if (dotIndex != -1) {
7.             String packageName = name.substring(0, dotIndex);
8.             synchronized (this) {
9.                 Package packageObj = getPackage(packageName);
10.                 if (packageObj == null) {
11.                     definePackage(packageName, null, null,
12.                         null, null, null, null, null);
13.                 }
14.             }
15.         }
```

```
16.         return defineClass(name, data, 0, data.length);
17.     }
18. */
```

这从另一方面佐证了 `defineClass` 函数在 Dalvik 虚拟机里确实是被阉割了。而在这两个继承自 `ClassLoader` 的类加载器，本质上是重载了 `ClassLoader` 的 `findClass` 方法。在执行 `loadClass` 时，我们可以参看 `ClassLoader` 部分源码：

```
1. protected Class<?> loadClass(String className, boolean resolve) throws ClassNotFoundException {
2.     Class<?> clazz = findLoadedClass(className);
3.     if (clazz == null) {
4.         try {
5.             clazz = parent.loadClass(className, false);
6.         } catch (ClassNotFoundException e) {
7.             // Don't want to see this.
8.         }
9.         if (clazz == null) {
10.            clazz = findClass(className);
11.        }
12.    }
13.    return clazz;
14. }
```

因此 `DexClassLoader` 和 `PathClassLoader` 都属于符合双亲委派模型的类加载器（因为它们没有重载 `loadClass` 方法）。也就是说，它们在加载一个类之前，回去检查自己以及自己以上的类加载器是否已经加载了这个类。如果已经加载过了，就会直接将之返回，而不会重复加载。

`DexClassLoader` 和 `PathClassLoader` 其实都是通过 `DexFile` 这个类来实现类加载的。这里需要顺便提一下的是，Dalvik 虚拟机识别的是 dex 文件，而不是 class 文件。因此，我们供类加载的文件也只能是 dex 文件，或者包含有 dex 文件的 apk 或 jar 文件。

也许有人想到，既然 `DexFile` 可以直接加载类，那么我们为什么还要使用 `ClassLoader` 的子类呢？`DexFile` 在加载类时，具体是调用成员方法 `loadClass` 或者 `loadClassBinaryName`。其中 `loadClassBinaryName` 需要将包含包名的类名中的“.” 转换为“/” 我们看一下 `loadClass` 代码就清楚了：

```
1. public Class loadClass(String name, ClassLoader loader) {
2.     String slashName = name.replace('.', '/');
3.     return loadClassBinaryName(slashName, loader);
}
```



```
4. }
```

在这段代码前有一段注释,截取关键一部分就是说 :If you are not calling this from a class loader, this is most likely not going to do what you want. Use {@link Class#forName(String)} instead. 这就是我们需要使用 ClassLoader 子类的原因。至于它是如何验证是否是在 ClassLoader 中调用此方法的,我没有研究,大家如果有兴趣可以继续深入下去。

有一个细节,可能大家不容易注意到。PathClassLoader 是通过构造函数 new DexFile(path)来产生 DexFile 对象的;而 DexClassLoader 则是通过其静态方法 loadDex (path, outputPath, 0) 得到 DexFile 对象。这两者的区别在于 DexClassLoader 需要提供一个可写的 outputPath 路径,用来释放.apk 包或者.jar 包中的 dex 文件。换个说法来说,就是 PathClassLoader 不能主动从 zip 包中释放出 dex ,因此只支持直接操作 dex 格式文件 ,或者已经安装的 apk(因为已经安装的 apk 在 cache 中存在缓存的 dex 文件)。而 DexClassLoader 可以支持.apk、.jar 和.dex 文件,并且会在指定的 outputPath 路径释放出 dex 文件。

另外,PathClassLoader 在加载类时调用的是 DexFile 的 loadClassBinaryName,而 DexClassLoader 调用的是 loadClass。因此,在使用 PathClassLoader 时类全名需要用"/" 替换 "."

原文地址: <http://blog.csdn.net/jiangwei0910410003/article/details/17679823>

7、如果加载高清大图片,不用第三方,不压缩,怎么处理防止 OOM (2017-2-24)

首先不压缩,按照原图尺寸加载,那么屏幕肯定是不够大的,并且考虑到内存的情况,不可能一次性整图加载到内存中,所以肯定是局部加载,那么就需要用到一个类:BitmapRegionDecoder,主要用于显示图片的某一块矩形区域,再通过手势进行滑动。

关键代码:

MainActivity.java 的使用

```
1. public class MainActivity extends AppCompatActivity {  
2. private LargeImageView mImageView;//自定义控件  
3. @Override  
4. protected void onCreate(Bundle savedInstanceState) {
```

```
5.    super.onCreate(savedInstanceState);
6.    // setContentView(R.layout.activity_main);
7.    setContentView(R.layout.activity_large_image_view);
8.    mImageView = (LargeImageView) findViewById(R.id.id_imageview);
9.    try{
10.        InputStream inputStream = getAssets().open("tangyan.jpg");//假设 assets 目录下有张大图片
11.        mImageView.setInputStream(inputStream);
12.    } catch (IOException e){
13.        e.printStackTrace(); }
14. }
```

Dsfjska

自定义 LargeImageView.java:

```
1. public class LargeImageView extends View {
2.     private BitmapRegionDecoder mDecoder;           //创建类
3.     private int mImageWidth, mImageHeight;         //图片的宽度和高度
4.     private volatile Rect mRect = new Rect();       //绘制的区域
5.     private MoveGestureDetector mDetector;          //手势识别
6.     public LargeImageView(Context context, AttributeSet attrs){
7.         super(context, attrs);
8.         init();//做一些初始化
9.     }
10. //手势滑动识别。获取手指坐标的宽高，进行判断图片移动的距离。
11. public void init(){
12.     mDetector=new MoveGestureDetector(getContext(),
13.         new MoveGestureDetector.SimpleMoveGestureDetector(){
14.             @Override
15.             public boolean onMove(MoveGestureDetector detector){
16.                 int moveX = (int) detector.getMoveX();//获取 x 轴距离
17.                 int moveY = (int) detector.getMoveY();//获取 y 轴距离
18.                 if (mImageWidth > getWidth()){
19.                     mRect.offset(-moveX, 0);
20.                     //是否超出边界
21.                     checkWidth();
22.                     invalidate();
23.                 }
24.                 if (mImageHeight > getHeight()) {
25.                     mRect.offset(0, -moveY);
26. //是否超出边界
27.                     checkHeight();
28.                     invalidate();
29.                 }
30.                 return true;}}); }
31. //把事件给手势
```

```
32.     @Override
33.     public boolean onTouchEvent(MotionEvent event){
34.         mDetector.onToucEvent(event);
35.         return true;
36.     }
37.     @Override
38.     protected void onDraw(Canvas canvas){
39.         //解析图片的中间矩形区域
40.         Bitmap bm = mDecoder.decodeRegion(mRect, options);
41.         canvas.drawBitmap(bm, 0, 0, null);
42.     }
43.     @Override
44.     protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec){
45.         super.onMeasure(widthMeasureSpec, heightMeasureSpec)
46.         int width = getMeasuredWidth();//获取屏幕的宽
47.         int height = getMeasuredHeight();//获取屏幕的高
48.         int imageWidth = mImageWidth;    //获取图片的宽
49.         int imageHeight = mImageHeight;  //获取图片的高
50.         //默认直接显示图片的中心区域，可以自己去调节
51.         mRect.left = imageWidth / 2 - width / 2 ;
52.         mRect.top = imageHeight / 2 - height / 2;
53.         mRect.right = mRect.left + width;
54.         mRect.bottom = mRect.top + height;
55.     }
56.     //获取 option
57.     private static final BitmapFactory.Options options = new BitmapFactory.Options(
58.         static {options.inPreferredConfig = Bitmap.Config.RGB_565;}
59.     //设置图片的方法
60.     public void setInputStream(InputStream is){
61.         try{
62.             //解析流获取图片的宽，高
63.             BitmapFactory.Options tmpOptions = new BitmapFactory.Options(
64.                 tmpOptions.inJustDecodeBounds = true;
65.                 BitmapFactory.decodeStream(is, null, tmpOptions);//解析流
66.                 mImageWidth = tmpOptions.outWidth; //
67.                 mImageHeight = tmpOptions.outHeight;
68.                 //获取实例， 传入两个参数：图片流，boolean值（false 表示 copy 一份当前的图片）
69.                 mDecoder = BitmapRegionDecoder.newInstance(is, false)
70.                 requestLayout();    //重新绘制
71.                 invalidate();    //重新绘制
72.             } catch (IOException e) {
73.                 e.printStackTrace();
74.             } finally{
75.                 try{
```

```
76.         if (is != null) is.close();
77.         } catch (Exception e){           }}}
```

二、Android 屏幕适配

1、屏幕适配方式都有哪些

1.1 适配方式之 dp

名词解释：

◆ 分辨率：eg：480*800,1280*720。表示物理屏幕区域内像素点的总和。(切记：跟屏幕适配没有任何关系)

因为我们既可以把 1280*720 的分辨率做到 4.0 的手机上面。我也可以把 1280*720 的分辨率做到 5.0 英寸的手机上面，如果分辨率相同，手机屏幕越小清晰。

◆ px(pix)：像素，就是屏幕中最小的一个显示单元

◆ dpi (像素密度)：即每英寸屏幕所拥有的像素数，像素密度越大，显示画面细节就越丰富。

计算公式：像素密度 = $\sqrt{(\text{长度像素数}^2 + \text{宽度像素数}^2)}$ / 屏幕尺寸

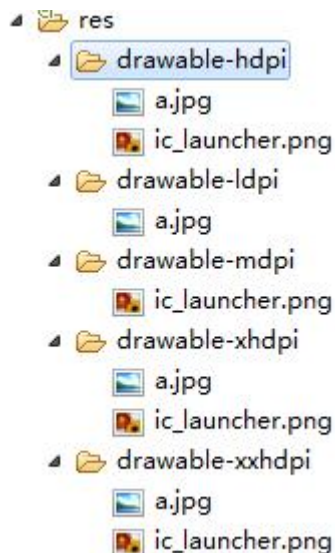
注：屏幕尺寸单位为英寸 例：分辨率为 1280*720 屏幕宽度为 6 英寸 计算所得像素密度约等于 245，屏幕尺寸指屏幕对角线的长度。

在 Android 手机中 dpi 分类：

ldpi	Resources for low-density (<i>ldpi</i>) screens (~120dpi).
mdpi	Resources for medium-density (<i>mdpi</i>) screens (~160dpi). (This is the baseline density.)

hdpi	Resources for high-density (<i>hdpi</i>) screens (~240dpi).
xhdpi	Resources for extra high-density (<i>xhdpi</i>) screens (~320dpi).

在我们的 Android 工程目录中有如下 drawable-*dpi 目录，这些目录是用来适配不同分辨率手机的。



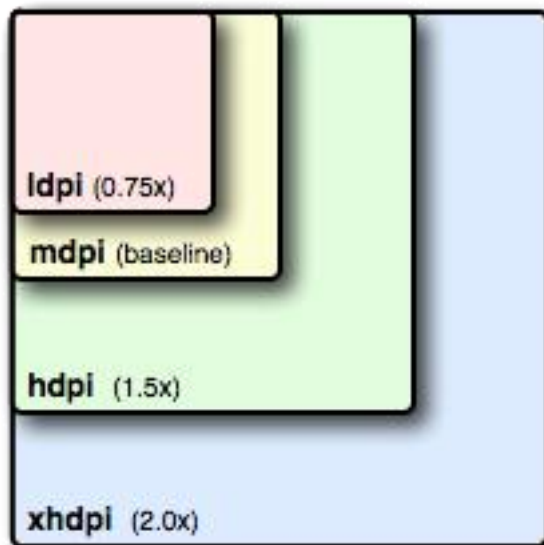
Android 应用在查找图片资源时会根据其分辨率自动从不同的文件目录下查找（这本身就是 Android 系统的适配策略），如果在低分辨的文件目录中比如 drawable-mdpi 中没有图片资源，其他目录中都有，当我们将该应用部署到 mdpi 分辨率的手机上时，那么该应用会查找分辨率较高目录下的资源文件，如果较高分辨率目录下也没有资源则只好找较低目录中的资源了。

常见手机屏幕像素及对应分辨率级别：

- ◆ ldpi 320*240
- ◆ mdpi 480*320
- ◆ hdpi 800*480
- ◆ xhdpi 1280*720
- ◆ xxhdpi 1920*1080

dp 和 px 之间的简单换算关系：

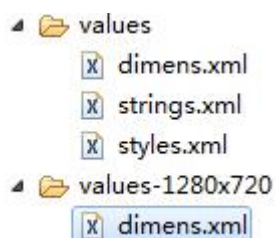
- ◆ ldpi 的手机 1dp=0.75px
- ◆ mdpi 的手机 1dp=1.0px
- ◆ hdpi 的手机 1dp=1.5px
- ◆ xhdpi 的手机 1dp=2.0px
- ◆ xxhdpi 的手机 1dp=3.0px



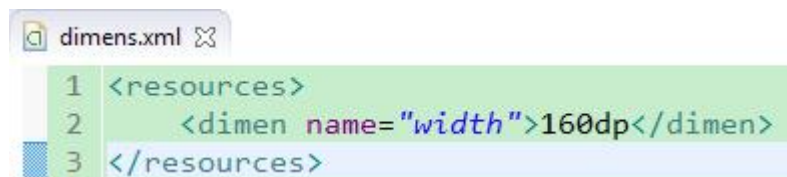
Tips：根据上面的描述我们得出如下结论，对于 mdpi 的手机，我们的布局通过 dp 单位可以达到适配效果。

1.2 适配方式之 `dimens`

跟 `drawable` 目录类似的，在 Android 工程的 `res` 目录下有 `values` 目录，这个是默认的目录，同时为了适配不同尺寸手机我们可以创建一个 `values-1280x720` 的文件夹，同时将 `dimens.xml` 文件拷贝到该目录下。

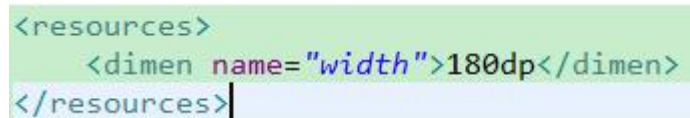


在 `dimens.xml` 中定义一个尺寸，如下图所示。



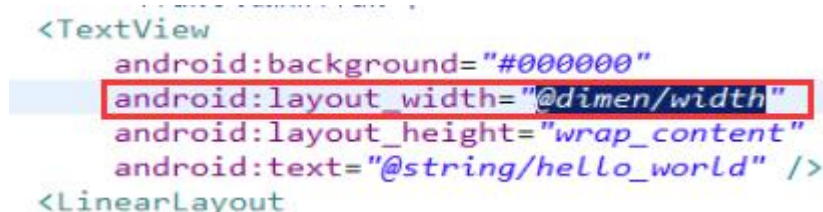
```
1 <resources>
2     <dimen name="width">160dp</dimen>
3 </resources>
```

在 values-1280x720 目录中的 dimens.xml 中定义同样的尺寸名称，但是使用不同的尺寸，如下图所示。



```
<resources>
    <dimen name="width">180dp</dimen>
</resources>
```

当我们在布局文件中使用长或者宽度单位时，比如下图所示，应该使用@dimen/width 来灵活的定义宽度。

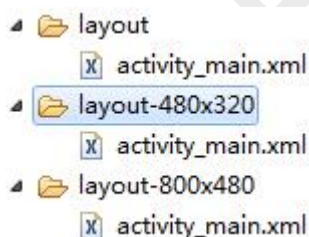


```
<TextView
    android:background="#000000"
    android:layout_width="@dimen/width"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
</LinearLayout>
```

Tips：在 values-1280x720 中，中间的是大写字母 X 的小写形式 x，而不是加减乘除的乘号。如果我们在 values-1280x720 中放置了 dimens 常量，一定记得也将该常量的对应值在 values 目录下的 dimens.xml 中放一份，因为该文件是默认配置，当用户的手机不是 1280*720 的情况下系统应用使用的是默认 values 目录中的 dimens.xml。

1.3 适配方式之 layout

跟 values 一样，在 Android 工程目录中 layout 目录也支持类似 values 目录一样的适配，在 layout 中我们可以针对不同手机的分辨率制定不同的布局，如下图所示。



```
layout
├── activity_main.xml
├── layout-480x320
│   └── activity_main.xml
└── layout-800x480
    └── activity_main.xml
```

1.4 适配方式之 java 代码适配

为了演示用 java 代码控制适配的效果，因此假设有这样的需求，让一个 TextView 控件的宽和高分别为屏幕的宽和高的一半。

我们新创建一个 Android 工程，修改 main_activity.xml，布局文件清单如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
```

```
<!-- 当前控件宽高为屏幕宽度的各 50% -->
<TextView
    android:id="@+id/tv"
    android:background="#000000"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
</RelativeLayout>
```

在 MainActivity.java 类中完成用 java 代码控制 TextView 的布局效果，其代码清单如下：


```
public class MainActivity extends Activity {

    private static final String tag = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //去掉 title
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        //获取 TextView 控件
        TextView tv = (TextView) findViewById(R.id.tv);
        //找到当前控件的父控件(父控件上给当前的子控件去设定一个规则)
        DisplayMetrics metrics = new DisplayMetrics();
        //给当前 metrics 去设置当前屏幕信息(宽(像素)高(像素))
        getWindowManager().getDefaultDisplay().getMetrics(metrics);
        //获取屏幕的高度和宽度
        Constant.srceenHeight = metrics.heightPixels;
        Constant.srceenWidth = metrics.widthPixels;
        //日志输出屏幕的高度和宽度
        Log.i(tag, "Constant.srceenHeight = "+Constant.srceenHeight);
        Log.i(tag, "Constant.srceenWidth = "+Constant.srceenWidth);
        //宽高各 50%
        RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams(
            //数学角度上 四舍五入
            (int)(Constant.srceenWidth*0.5+0.5),
            (int)(Constant.srceenHeight*0.5+0.5));
        //给 tv 控件设置布局参数
        tv.setLayoutParams(layoutParams);
    }
}
```

其中 Constant 类是一个常量类，很简单，只有两个常量用来记录屏幕的宽和高，其代码清单如下：

```
public class Constant {
    public static int srceenHeight;
    public static int srceenWidth;
}
```

1.5 适配方式之 weight 权重适配

在控件中使用属性 `android:layout_weight="1"` 可以起到适配效果，但是该属性的使用有如下规则：

- 1、只能用在线性控件中，比如 LinearLayout。
- 2、竖直方向上使用权重的控件高度必须为 0dp（Google 官方的推荐用法）
- 3、水平方向上使用权重的控件宽度必须为 0dp（Google 官方的推荐用法）

2、屏幕适配的处理技巧都有哪些

手机自适应主要分为两种情况：横屏和竖屏的切换，以及分辨率大小的不同。

2.1 横屏和竖屏的切换

1、Android 应用程序支持横竖屏幕的切换，Android 中每次屏幕的切换都会重启 Activity，所以应该在 Activity 销毁（执行 onPause()方法和 onDestroy()方法）前保存当前活动的状态；在 Activity 再次创建的时候载入配置，那样，进行中的游戏就不会自动重启了！有的程序适合从竖屏切换到横屏，或者反过来，这个时候怎么办呢？可以在配置 Activity 的地方进行如下的配置 android:screenOrientation="portrait"（landscape 是横向，portrait 是纵向）。这样就可以保证是竖屏总是竖屏了。

2、而有的程序是适合横竖屏切换的。如何处理呢？首先要在配置 Activity 的时候进行如下的配置：

android:configChanges="keyboardHidden|orientation"，另外需要重写 Activity 的 onConfigurationChanged 方法。实现方式如下：

```
@Override
public void onConfigurationChanged(Configuration newConfig){
    super.onConfigurationChanged(newConfig);
    if(this.getResources().getConfiguration().orientation==Configuration.ORIENTATION_
    LANDSCAPE){
        //TODO
    }else
    if(this.getResources().getConfiguration().orientation==Configuration.ORIENTATION_
    PORTRAIT){
        //TODO
    }
}
```

2.2 分辨率大小不同

对于分辨率问题，官方给的解决办法是创建不同的 layout 文件夹，这就需要对每种分辨率的手机都要写一个布局文件，虽然看似解决了分辨率的问题，但是如果其中一处或多处有修改了，就要每个布局文件都要做出修改，这样就造成很大的麻烦。那么可以通过以下几种方式解决：

一) 使用 layout_weight

目前最为推荐的 Android 多屏幕自适应解决方案。

该属性的作用是决定控件在其父布局中的显示权重，一般用于线性布局中。其值越小，则对应的 layout_width 或 layout_height 的优先级就越高（一般到 100 作用就不太明显了）；一般横向布局中，决定的是 layout_width 的优先级；纵向布局中，决定的是 layout_height 的优先级。

传统的 layout_weight 使用方法是将当前控件的 layout_width 和 layout_height 都设置成 fill_parent，这样就可以把控件的显示比例完全交给 layout_weight；这样使用的话，就出现了 layout_weight 越小，显示比例越大的情况（即权重越大，显示所占的效果越小）。不过对于 2 个控件还好，如果控件过多，且显示比例也不相同的时候，控制起来就比较麻烦了，毕竟反比不是那么好确定的。于是就有了现在最为流行的 0px 设值法。看似让人难以理解的 layout_height=0px 的写法，结合 layout_weight，却可以使控件成正比例显示，轻松解决了当前 Android 开发最为头疼的碎片化问题之一。

二) 清单文件配置：【不建议使用这种方式，需要对不同的界面写不同的布局】

需要在 AndroidManifest.xml 文件的 <manifest> 元素如下添加子元素

```
<supports-screens android:largeScreens="true"
```

```
android:normalScreens="true"
```

```
android:anyDensity="true"
```

```
android:smallScreens="true"
```

```
android:xlargeScreens="true">
```

</supports-screens>

以上是为我们屏幕设置多分辨率支持（更准确的说是适配大、中、小三种密度）。

Android:anyDensity="true"，这一句对整个屏幕都起着十分重要的作用，值为 true，我们的应用程序当安装在不同密度的手机上时，程序会分别加载 hdpi,mdpi,ldpi 文件夹中的资源。相反，如果值设置为 false，即使我们在 hdpi,mdpi,ldpi，xdpi 文件夹下拥有同一种资源，那么应用也不会自动地去相应文件夹下寻找资源。而是会在大密度和小密度手机上加载中密度 mdpi 文件中的资源。

有时候会根据需要在代码中动态地设置某个值，可以在代码中为这几种密度分别设置偏移量，但是这种方法最好不要使用，最好的方式是在 xml 文件中不同密度的手机进行分别设置。这里地图的偏移量可以在 values-xdpi，values-hdpi,values-mdpi,values-ldpi 四种文件夹中的 dimens.xml 文件进行设置。

三)、其他：

说明：

在不同分辨率的手机模拟器下，控件显示的位置会稍有不同

通过在 layout 中定义的布局设置的参数，使用 dp (dip)，会根据不同的屏幕分辨率进行适配

但是在代码中的各个参数值，都是使用的像素 (px) 为单位的

技巧：

1、尽量使用线性布局，相对布局，如果屏幕放不下了，可以使用 ScrollView (可以上下拖动)

ScrollView 使用的注意：

在不同的屏幕上显示内容不同的情况，其实这个问题我们往往是用滚动视图来解决的，也就是 ScrollView；需要注意的是 ScrollView 中使用 layout_weight 是无效的，既然使用 ScrollView 了，就把它里面的控件的大小都设成固定的吧。

2、指定宽高的时候，采用 dip 的单位，dp 单位动态匹配

3、由于 android 代码中写的单位都是像素，所有需要通过工具类进行转化

4、尽量使用 9-patch 图，可以自动的依据图片上面显示的内容被拉伸和收缩。其中在编辑的时候，灰色区域是被拉伸的，上下两个点控制水平方向的拉伸，左右两点控制垂直方向的拉伸

3、dp 和 px 之间的关系

dp：是 dip 的简写，指密度无关的像素。

指一个抽象意义上的像素，程序用它来定义界面元素。一个与密度无关的，在逻辑尺寸上，与一个位于像素密度为 160dpi 的屏幕上的像素是一致的。要把密度无关像素转换为屏幕像素，可以用这样一个简单的公式：
 $\text{pixels} = \text{dips} * (\text{density} / 160)$ 。举个例子，在 DPI 为 240 的屏幕上，1 个 DIP 等于 1.5 个物理像素。

布局时最好使用 dp 来定义我们程序的界面 因为这样可以保证我们的 UI 在各种分辨率的屏幕上都可以正常显示。

```
/**
 * 根据手机的分辨率从 px(像素) 的单位 转成为 dp
 */
public static int px2dip(Context context, float pxValue) {
    final float scale = context.getResources().getDisplayMetrics().density;
    return (int) (pxValue / scale + 0.5f);
}
/**
 * 根据手机的分辨率从 dip 的单位 转成为 px(像素)
 */
public static int dip2px(Context context, float dpValue) {
    final float scale = context.getResources().getDisplayMetrics().density;
    return (int) (dpValue * scale + 0.5f);
}
```

三、AIDL

1、什么是 AIDL 以及如何使用

①aidl 是 Android interface definition Language 的英文缩写，意思 Android 接口定义语言。

②使用 `aidl` 可以帮助我们发布以及调用远程服务，实现跨进程通信。

③将服务的 `aidl` 放到对应的 `src` 目录，工程的 `gen` 目录会生成相应的接口类

我们通过 `bindService (Intent , ServiceConnect , int)` 方法绑定远程服务，在 `bindService` 中有一个 `ServiceConnec` 接口，我们需要覆写该类的 `onServiceConnected(ComponentName,IBinder)` 方法，这个方法的第二个参数 `IBinder` 对象其实就是已经在 `aidl` 中定义的接口，因此我们可以将 `IBinder` 对象强制转换为 `aidl` 中的接口类。

我们通过 `IBinder` 获取到的对象（也就是 `aidl` 文件生成的接口）其实是系统产生的代理对象，该代理对象既可以跟我们的进程通信，又可以跟远程进程通信，作为一个中间的角色实现了进程间通信。

四、自定义控件

1、如何自定义一个控件

自定义控件可以分为两种自定义组合控件和自定义 `view`。

◆ 自定义组合控件

自定义组合控件就是把多个控件做为一个整体看待、处理。这样的好处不仅可以减轻 `xml` 的代码量，也提高了代码的复用性。

在手机卫士项目中我们第一次接触了自定义组合控件。

1. 声明一个 `View` 对象，继承相对布局，或者线性布局或者其他 `ViewGroup`。
2. 在自定义的 `View` 对象里面重写它的构造方法，在构造方法里面就把布局都初始化完毕。
3. 根据业务需求添加一些 `api` 方法，扩展自定义的组合控件；
4. 希望在布局文件里面可以自定义一些属性。
5. 声明自定义属性的命名空间。

```
xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe"
```

6. 在 `res` 目录下的 `values` 目录下创建 `attrs.xml` 的文件声明我们写的属性。

7. 在布局文件中写自定义的属性。

8. 使用这些定义的属性。自定义 View 对象的构造方法里面有一个带两个参数的构造方法布局文件里面定义的属性都放在 AttributeSet attrs，获取那些定义的属性。

◆ 自定义 view

自定义 View 首先要实现一个继承自 View 的类。添加类的构造方法，通常是三个构造方法，不过从 Android5.0 开始构造方法已经添加到 4 个了。override 父类的方法，如 onDraw，(onMeasure) 等。如果自定义的 View 有自己的属性，需要在 values 下建立 attrs.xml 文件，在其中定义属性，同时代码也要做修改。

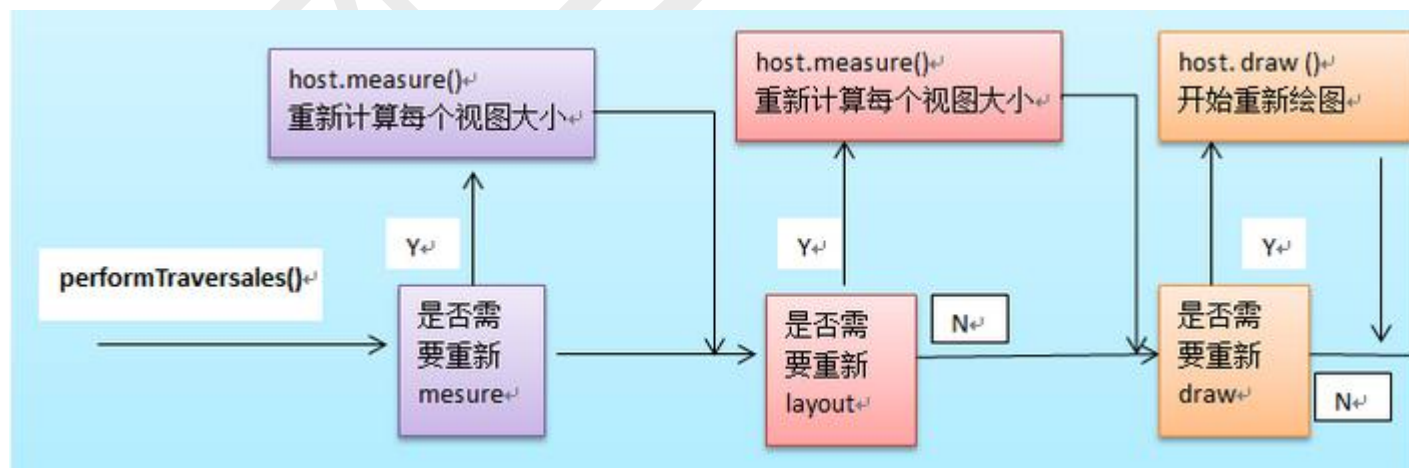
blog 分享：<http://blog.csdn.net/lmj623565791/article/details/24252901>

2、请描述一下 View 的绘制流程

整个 View 树的绘图流程是在 ViewRoot.java 类(该类位于 Android 源码下面：

D:\AndroidSource_GB\AndroidSource_GB\frameworks\base\core\java\android\view)的

performTraversals()函数展开的，该函数做的执行过程可简单概况为根据之前设置的状态，判断是否需要重新计算视图大小(measure)、是否重新需要安置视图的位置(layout)、以及是否需要重绘 (draw)，其框架过程如下：



1、mesarue() 过程

主要作用：为整个 View 树计算实际的大小，即设置实际的高(对应属性:mMeasuredHeight)和宽(对应属性:

mMeasureWidth)，每个 View 的控件的实际宽高都是由父视图和本身视图决定的。

具体的调用链如下：ViewRoot 根对象的属性 mView(其类型一般为 ViewGroup 类型)调用 measure()方法去计算 View 树的大小，回调 View/ViewGroup 对象的 onMeasure()方法，该方法实现的功能如下：

1、设置本 View 视图的最终大小，该功能的实现通过调用 setMeasuredDimension()方法去设置实际的高(对应属性：mMeasuredHeight)和宽(对应属性：mMeasureWidth)。

2、如果该 View 对象是个 ViewGroup 类型，需要重写该 onMeasure()方法，对其子视图进行遍历的 measure()过程。对每个子视图的 measure()过程，是通过调用父类 ViewGroup.java 类里的 measureChildWithMargins()方法去实现，该方法内部只是简单地调用了 View 对象的 measure()方法。

2、layout 布局过程

主要作用：为将整个根据子视图的大小以及布局参数将 View 树放到合适的位置上。

具体的调用链如下：

1、layout 方法会设置该 View 视图位于父视图的坐标轴，即 mLeft，mTop，mLeft，mBottom(调用 setFrame()函数去实现)接下来回调 onLayout()方法(如果该 View 是 ViewGroup 对象，需要实现该方法，对每个子视图进行布局)。

2、如果该 View 是个 ViewGroup 类型，需要遍历每个子视图 childView，调用该子视图的 layout()方法去设置它的坐标值。

3、draw()绘图过程

由 ViewRoot 对象的 performTraversals()方法调用 draw()方法发起绘制该 View 树，值得注意的是每次发起绘图时，并不会重新绘制每个 View 树的视图，而只会重新绘制那些“需要重绘”的视图，View 类内部变量包含了一个标志位 DRAWN，当该视图需要重绘时，就会为该 View 添加该标志位。

调用流程：

- 1、绘制该 View 的背景
- 2、为显示渐变框做一些准备操作(大多数情况下，不需要改渐变框)
- 3、调用 onDraw()方法绘制视图本身(每个 View 都需要重载该方法，ViewGroup 不需要实现该方法)
- 4、调用 dispatchDraw ()方法绘制子视图(如果该 View 类型不为 ViewGroup，即不包含子视图，不需要重载该方法)

值得说明的是，ViewGroup 类已经为我们重写了 dispatchDraw ()的功能实现，应用程序一般不需要重写该方法，但可以重载父类函数实现具体的功能。

参考 blog 分享：<http://blog.csdn.net/qinjuning/article/details/7110211>

3、View,SurfaceView,GLSurfaceView 有什么区别？（2017-2-23）

1) View

View 类是 Android 的一个非常重要的超类，它是 Android 里所有与用户交互的控件的父类，包括 Widget 类的交互 UI 控件(按钮、文本框等)和 ViewGroup 类布局控件。SurfaceView 继承自 View，GLSurfaceView 继承自 SurfaceView。

2) SurfaceView

2.1 SurfaceView 概述

普通的 Android 控件，例如 TextView、Button 和 CheckBox 等，它们都是将自己的 UI 绘制在宿主窗口的绘图表面之上，这意味着它们的 UI 是在应用程序的主线程中进行绘制的。由于应用程序的主线程除了要绘制 UI 之外，还需要及时地响应用户输入，否则的话，系统就会认为应用程序没有响应了，因此就会弹出一个 ANR 对话框出来。对于一些游戏画面，或者摄像头预览、视频播放来说，它们的 UI 都比较复杂，而且要求能够进行高效的绘制，因此，它们的 UI 就不适合在应用程序的主线程中进行绘制。这时候就必须给那些需要复杂而高效 UI 的视图生成一个独立的绘

图表面，以及使用一个独立的线程来绘制这些视图的 UI。

2.2 SurfaceView 的大致实现原理

Android 应用程序窗口是通过 SurfaceFlinger 服务来绘制自己的 UI 的。一般来说，每一个窗口在 SurfaceFlinger 服务中都对应有一个 Layer，用来描述它的绘图表面。对于那些具有 SurfaceView 的窗口来说，每一个 SurfaceView 在 SurfaceFlinger 服务中还对应有一个独立的 Layer 或者 LayerBuffer，用来单独描述它的绘图表面，以区别于它的宿主窗口的绘图表面。

无论是 LayerBuffer，还是 Layer，它们都是以 LayerBase 为基类的，也就是说，SurfaceFlinger 服务把所有的 LayerBuffer 和 Layer 都抽象为 LayerBase，因此就可以用统一的流程来绘制和合成它们的 UI。

我们假设在一个 Activity 窗口的视图结构中，除了有一个 DecorView 顶层视图之外，还有两个 TextView 控件，以及一个 SurfaceView 视图，这样该 Activity 窗口在 SurfaceFlinger 服务中就对应有两个 Layer 或者一个 Layer 的一个 LayerBuffer，如图所示：

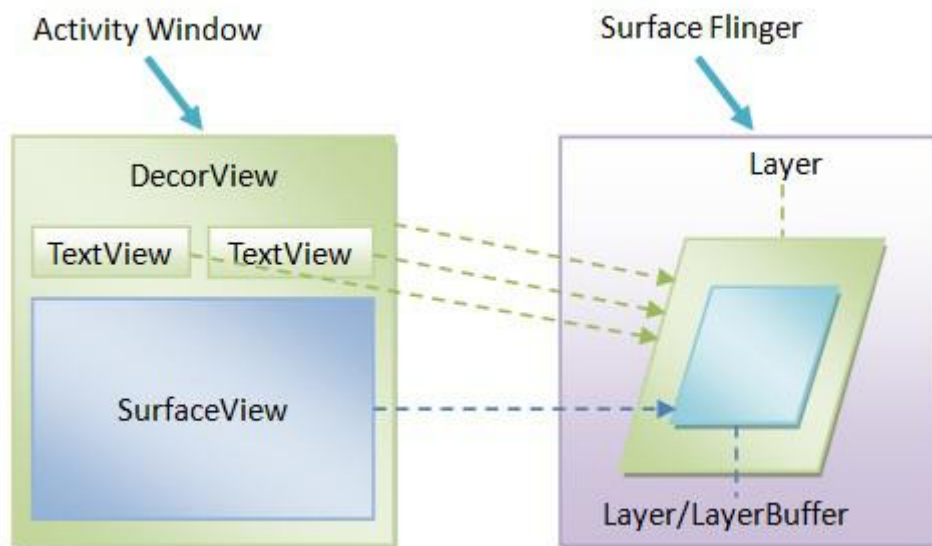


图 SurfaceView 及其宿主 Activity 窗口的绘图表面示意图

在图中，Activity 窗口的顶层视图 DecorView 及其两个 TextView 控件的 UI 都是绘制在 SurfaceFlinger 服务中的同一个 Layer 上面的，而 SurfaceView 的 UI 是绘制在 SurfaceFlinger 服务中的另外一个 Layer 或者 LayerBuffer 上的。

注意，用来描述 SurfaceView 的 Layer 或者 LayerBuffer 的 Z 轴位置是小于用来其宿主 Activity 窗口的 Layer 的 Z 轴位置的，但是前者会在后者的上面挖一个“洞”出来，以便它的 UI 可以对用户可见。实际上，SurfaceView 在其宿主 Activity 窗口上所挖的“洞”只不过是其在宿主 Activity 窗口上设置了一块透明区域。

2.3 SurfaceView 的简单使用

a. 获取到 SurfaceView 对应的 SurfaceHolder，给 SurfaceHolder 添加一个 SurfaceHolder.Callback 对象。

b. 创建渲染线程对象

c. 在子线程中开始在 Surface 上面绘制图形，因为 SurfaceView 没有对我们暴露 Surface，而只是暴露了 Surface 的包装器 SurfaceHolder，所以使用 SurfaceHolder 的 lockCanvas() 获取 Surface 上面指定区域的 Canvas，在该 Canvas 上绘制图形，绘制结束后，使用 SurfaceHolder 的 unlockCanvasAndPost() 方法解锁 Canvas，并且让 UI 线程把 Surface 上面的东西绘制到 View 的 Canvas 上面。示例代码如下：

```
1. public class GameUI extends SurfaceView implements SurfaceHolder.Callback {
2.     private SurfaceHolder holder;
3.     private RenderThread renderThread;
4.     private boolean isDraw = false; // 控制绘制的开关
5.     public GameUI(Context context) {
6.         super(context); holder = this.getHolder();
7.         holder.addCallback(this);
8.         renderThread = new RenderThread();
9.     }
10.    @Override public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
11.    { }
12.    @Override public void surfaceCreated(SurfaceHolder holder) {
13.        isDraw = true;
14.        renderThread.start();
15.    }
16.    @Override public void surfaceDestroyed(SurfaceHolder holder) {
```

```
17.         isDraw = false;
18.     }
19.     /**
20.      * 绘制界面的线程
21.      *
22.      * @author Administrator
23.      *
24.      */
25.     private class RenderThread extends Thread {
26.         @Override public void run() {
27.             // 不停绘制界面
28.             while (isDraw) {
29.                 drawUI();
30.             }
31.             super.run();
32.         }
33.     }
34.     /**
35.      * 界面绘制
36.      */
37.     public void drawUI() {
38.         Canvas canvas = holder.lockCanvas();
39.         try {
40.             drawCanvas(canvas);
41.         } catch (Exception e) {
42.             e.printStackTrace();
43.         } finally {
44.             holder.unlockCanvasAndPost(canvas);
45.         }
46.     }
47.     private void drawCanvas(Canvas canvas) {
48.         // 在 canvas 上绘制需要的图形
49.     }
50. }
```

3) GLSurfaceView

3.1 GLSurfaceView 概述

GLSurfaceView 是一个视图，继承至 SurfaceView，它内嵌的 surface 专门负责 OpenGL 渲染。

GLSurfaceView 提供了下列特性：

- 1> 管理一个 surface，这个 surface 就是一块特殊的内存，能直接排版到 Android 的视图 view 上。
- 2> 管理一个 EGL display，它能让 opengl 把内容渲染到上述的 surface 上。
- 3> 用户自定义渲染器(render)。
- 4> 让渲染器在独立的线程里运作，和 UI 线程分离。
- 5> 支持按需渲染(on-demand)和连续渲染(continuous)。
- 6> 一些可选工具，如调试。

五、Android 中的事件处理

1、Handler 机制

Android 中主线程也叫 UI 线程，那么从名字上我们也知道主线程主要是用来创建、更新 UI 的，而其他耗时操作，比如网络访问，或者文件处理，多媒体处理等都需要在子线程中操作，之所以在子线程中操作是为了保证 UI 的流畅程度，手机显示的刷新频率是 60Hz，也就是一秒钟刷新 60 次，每 16.67 毫秒刷新一次，为了不丢帧，那么主线程处理代码最好不要超过 16 毫秒。当子线程处理完数据后，为了防止 UI 处理逻辑的混乱，Android 只允许主线程修改 UI，那么这时候就需要 Handler 来充当子线程和主线程之间的桥梁了。

我们通常将 Handler 声明在 Activity 中，然后覆写 Handler 中的 handleMessage 方法，当子线程调用 handler.sendMessage()方法后 handleMessage 方法就会在主线程中执行。

这里面除了 Handler、Message 外还有隐藏的 Looper 和 MessageQueue 对象。

在主线程中 Android 默认已经调用了 Looper.preper() 方法，调用该方法的目的是在 Looper 中创建 MessageQueue 成员变量并把 Looper 对象绑定到当前线程中。当调用 Handler 的 sendMessage (对象) 方法的时候就将 Message 对象添加到了 Looper 创建的 MessageQueue 队列中，同时给 Message 指定了 target 对象，其实

这个 target 对象就是 Handler 对象。主线程默认执行了 `Looper.looper()` 方法，该方法从 `Looper` 的成员变量 `MessageQueue` 中取出 `Message`，然后调用 `Message` 的 target 对象的 `handleMessage()` 方法。这样就完成了整个消息机制。

2、事件分发机制

2.1 事件分发中的 `onTouch` 和 `onTouchEvent` 有什么区别，又该如何使用？

这两个方法都是在 `View` 的 `dispatchTouchEvent` 中调用的，`onTouch` 优先于 `onTouchEvent` 执行。如果在 `onTouch` 方法中通过返回 `true` 将事件消费掉，`onTouchEvent` 将不会再执行。

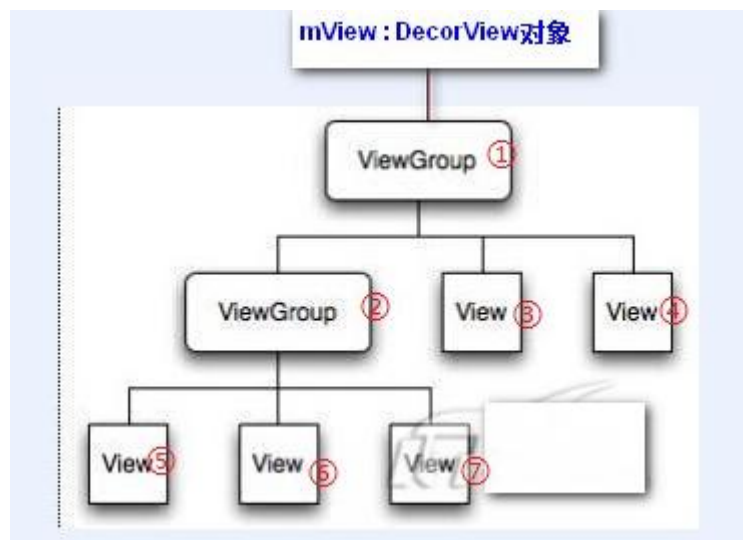
另外需要注意的是，`onTouch` 能够得到执行需要两个前提条件，第一 `mOnTouchListener` 的值不能为空，第二当前点击的控件必须是 `enable` 的。因此如果你有一个控件是非 `enable` 的，那么给它注册 `onTouch` 事件将永远得不到执行。对于这一类控件，如果我们想要监听它的 `touch` 事件，就必须通过在该控件中重写 `onTouchEvent` 方法来实现。

2.2 请描述一下 Android 的事件分发机制

Android 的事件分发机制主要是 `Touch` 事件分发，有两个主角：`ViewGroup` 和 `View`。`Activity` 的 `Touch` 事件事实上是调用它内部的 `ViewGroup` 的 `Touch` 事件，可以直接当成 `ViewGroup` 处理。

`View` 在 `ViewGroup` 内，`ViewGroup` 也可以在其他 `ViewGroup` 内，这时候把内部的 `ViewGroup` 当成 `View` 来分析。

先分析 `ViewGroup` 的处理流程：首先得有个结构模型概念：`ViewGroup` 和 `View` 组成了一棵树形结构，最顶层为 `Activity` 的 `ViewGroup`，下面有若干的 `ViewGroup` 节点，每个节点之下又有若干的 `ViewGroup` 节点或者 `View` 节点，依次类推。如图：



当一个 Touch 事件(触摸事件为例)到达根节点，即 Activity 的 ViewGroup 时，它会依次下发，下发的过程是调用子 View(ViewGroup)的 dispatchTouchEvent 方法实现的。简单来说，就是 ViewGroup 遍历它包含着的子 View，调用每个 View 的 dispatchTouchEvent 方法，而当子 View 为 ViewGroup 时，又会通过调用 ViewGroup 的 dispatchTouchEvent 方法继续调用其内部的 View 的 dispatchTouchEvent 方法。上述例子中的消息下发顺序是这样的：①-②-⑤-⑥-⑦-③-④。dispatchTouchEvent 方法只负责事件的分发，它拥有 boolean 类型的返回值，当返回为 true 时，顺序下发会中断。在上述例子中如果⑤的 dispatchTouchEvent 返回结果为 true，那么⑥-⑦-③-④将都接收不到本次 Touch 事件。

1.Touch 事件分发中只有两个主角:ViewGroup 和 View。ViewGroup 包含 onInterceptTouchEvent、dispatchTouchEvent、onTouchEvent 三个相关事件。View 包含 dispatchTouchEvent、onTouchEvent 两个相关事件。其中 ViewGroup 又继承于 View。

2.ViewGroup 和 View 组成了一个树状结构，根节点为 Activity 内部包含的一个 ViewGroup。

3.触摸事件由 Action_Down、Action_Move、Action_UP 组成，其中一次完整的触摸事件中，Down 和 Up 都只有一个，Move 有若干个，可以为 0 个。

4.当 Activity 接收到 Touch 事件时，将遍历子 View 进行 Down 事件的分发。ViewGroup 的遍历可以看成是递归的。分发的目的是为了找到真正要处理本次完整触摸事件的 View，这个 View 会在 onTouchEvent 结果返回 true。

5.当某个子 View 返回 true 时，会中止 Down 事件的分发，同时在 ViewGroup 中记录该子 View。接下去的 Move

和 Up 事件将由该子 View 直接进行处理。由于子 View 是保存在 ViewGroup 中的，多层 ViewGroup 的节点结构时，上级 ViewGroup 保存的会是真实处理事件的 View 所在的 ViewGroup 对象:如 ViewGroup0-ViewGroup1-TextView 的结构中，TextView 返回了 true，它将被保存在 ViewGroup1 中，而 ViewGroup1 也会返回 true，被保存在 ViewGroup0 中。当 Move 和 UP 事件来时，会先从 ViewGroup0 传递至 ViewGroup1，再由 ViewGroup1 传递至 TextView。

6.当 ViewGroup 中所有子 View 都不捕获 Down 事件时，将触发 ViewGroup 自身的 onTouch 事件。触发的方式是调用 super.dispatchTouchEvent 函数，即父类 View 的 dispatchTouchEvent 方法。在所有子 View 都不处理的情况下，触发 Activity 的 onTouchEvent 方法。

7.onInterceptTouchEvent 有两个作用：1.拦截 Down 事件的分发。2.中止 Up 和 Move 事件向目标 View 传递，使得目标 View 所在的 ViewGroup 捕获 Up 和 Move 事件。

3、在 Android 中主线程如何给子线程发 Message？（2015-12-1）重要

这是一个很好玩的话题通常我们都是 Activity 中，让子线程执行耗时任务，执行完之后给主线程发送消息让主线程更新 UI。其实还有很多应用场景需要让主线程给子线程发送消息，该消息作为任务的载体，比如在 IntentService 中，主线程就给子线程发送了消息，让子线程干活。

给大家写个 Demo 演示主线程给子线程发送消息。

【文件 1-1】MainActivity.java 代码片段

```
1. package com.yange.message2subthread;
2.
3. import android.os.Bundle;
4. import android.os.Handler;
5. import android.os.Looper;
6. import android.os.Message;
7. import android.util.Log;
8. import android.view.View;
9. import android.app.Activity;
10. /**
11. *
```



```
12. * @author wzy 2015-12-1
13. *
14. * 演示主线程给子线程发送 Message
15. *
16. */
17. public class MainActivity extends Activity {
18.     private Handler mMySubHandler;
19.
20.     @Override
21.     protected void onCreate(Bundle savedInstanceState) {
22.         super.onCreate(savedInstanceState);
23.         setContentView(R.layout.activity_main);
24.     }
25.     @Override
26.     protected void onStart() {
27.         super.onStart();
28.         /**
29.          * 当 Activity 启动的时候创建一个子线程, 并启动
30.          */
31.         MyThread thread = new MyThread();
32.         thread.start();
33.     }
34.     /**
35.      * 绑定布局中的点击按钮, 点击后给子线程发送消息
36.      */
37.      * @param view
38.      */
39.     public void click(View view){
40.         Message msg = new Message();
41.         msg.obj = "MainActivity";
42.         mMySubHandler.sendMessage(msg);
43.     }
44.     /**
45.      * 绑定布局文件中的按钮 2, 点击后让子线程退出, 关闭子线程
46.      * 其实这里只需要让子线程的 Looper 对象退出即可, 因为 Looper.loop(); 是线程阻塞的.
47.      * @param view
48.      */
49.     public void click2(View view){
50.         if (myLooper!=null) {
51.             myLooper.quit();
52.         }
53.     }
54.     /**
55.      * 将 Looper 对象声明为成员变量
```

```
56. */
57. private Looper myLooper;
58. @Override
59. protected void onDestroy() {
60.     super.onDestroy();
61.     /**
62.      * 在退出的时候,将子线程释放掉,不然可能会导致内存泄露
63.      */
64.     if (myLooper!=null) {
65.         myLooper.quit();
66.     }
67. }
68. class MyThread extends Thread{
69.
70.     @Override
71.     public void run() {
72.         //1. 创建一个 Looper 对象(内部创建了 MessageQueue,并将 MessageQueue 作
73. 为 Looper 对象的成员,然后将 Looper 对象绑定到 ThreadLocal 中)
74.         Looper.prepare();
75.         /**
76.          * 创建一个 Handler
77.          */
78.         mMySubHandler = new Handler(){
79.             @Override
80.             public void handleMessage(android.os.Message msg) {
81.                 //处理主线程发送的消息
82.                 Log.d("tag", "接收到信息"+msg);
83.             };
84.         };
85.         //2. 获取当前 Looper 对象
86.         myLooper = Looper.myLooper();
87.         //3. 让消息循环起来
88.         Looper.loop();
89.         Log.d("tag", "子线程退出");
90.     }
91. }
92.
93. }
94.
```

六、Android 签名

1、简单描述下 Android 数字签名

在 Android 系统中，所有安装到系统的应用程序都必有一个数字证书，此数字证书用于标识应用程序的作者和在应用程序之间建立信任关系。

Android 系统要求每一个安装进系统的应用程序都是经过数字证书签名的，数字证书的私钥则保存在程序开发者的手中。Android 将数字证书用来标识应用程序的作者和在应用程序之间建立信任关系，不是用来决定最终用户可以安装哪些应用程序。

这个数字证书并不需要权威的数字证书签名机构认证(CA)，它只是用来让应用程序包自我认证的。

同一个开发者的多个程序尽可能使用同一个数字证书，这可以带来以下好处。

(1)有利于程序升级，当新版程序和旧版程序的数字证书相同时，Android 系统才会认为这两个程序是同一个程序的不同版本。如果新版程序和旧版程序的数字证书不相同，则 Android 系统认为他们是不同的程序，并产生冲突，会要求新程序更改包名。

(2)有利于程序的模块化设计和开发。Android 系统允许拥有同一个数字签名的程序运行在一个进程中，Android 程序会将他们视为同一个程序。所以开发者可以将自己的程序分模块开发，而用户只需要在需要的时候下载适当的模块。

在签名时，需要考虑数字证书的有效期：

(1)数字证书的有效期要包含程序的预计生命周期，一旦数字证书失效，持有改数字证书的程序将不能正常升级。

(2)如果多个程序使用同一个数字证书，则该数字证书的有效期要包含所有程序的预计生命周期。

(3)Android Market 强制要求所有应用程序数字证书的有效期要持续到 2033 年 10 月 22 日以后。

Android 数字证书包含以下几个要点：

(1)所有的应用程序都必须有数字证书，Android 系统不会安装一个没有数字证书的应用程序

(2)Android 程序包使用的数字证书可以是自签名的，不需要一个权威的数字证书机构签名认证

(3)如果要正式发布一个 Android ，必须使用一个合适的私钥生成的数字证书来给程序签名，而不能使用 adt 插

件或者 ant 工具生成的调试证书来发布。

(4)数字证书都是有有效期的，Android 只是在应用程序安装的时候才会检查证书的有效期。如果程序已经安装在系统中，即使证书过期也不会影响程序的正常功能。

2、使用 Eclipse 如何生成数字签名

可以通过 Eclipse 导出工程时为当前工程设置签名证书。File -> Export ->Export Android Application ->Create New keystore

七、Android 中的动画

1、Android 中的动画有哪几类，它们的特点和区别是什么

Android 中动画分为两种，一种是 Tween 动画、还有一种是 Frame 动画。

Tween 动画，这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化；

Frame 动画，传统的动画方法，通过顺序的播放排列好的图片来实现，类似电影。

2、如何修改 Activity 进入和退出动画

可以通过两种方式，一是通过定义 Activity 的主题，二是通过覆写 Activity 的 overridePendingTransition 方法。

◆ 通过设置主题样式

在 styles.xml 中编辑如下代码：

```
<style name="AnimationActivity" parent="@android:style/Animation.Activity">
    <item name="android:activityOpenEnterAnimation">@anim/slide_in_left</item>
    <item name="android:activityOpenExitAnimation">@anim/slide_out_left</item>
    <item name="android:activityCloseEnterAnimation">@anim/slide_in_right</item>
    <item name="android:activityCloseExitAnimation">@anim/slide_out_right</item>
</style>
```

添加 themes.xml 文件：

```
<style name="ThemeActivity">
    <item name="android:windowAnimationStyle">@style/AnimationActivity</item>
    <item name="android:windowNoTitle">true</item>
</style>
```

在 AndroidManifest.xml 中给指定的 Activity 指定 theme。

◆ 覆写 overridePendingTransition 方法

```
overridePendingTransition(R.anim.fade, R.anim.hold);
```

八、编写自己的框架

初级程序员使用别人的框架，中级程序员不仅会使用别人的框架还知道内部的实现原理，高级程序员则按需编写自己的框架。添加该模块的目的就是想提交大家的逼格，让大家养成一个动手编写“自主知识产权”框架的意识。

1. 编写 ViewUtils 框架 (2016-1-20)

业界比较出名的基于完全注解方式就可以进行 UI 绑定和事件绑定，无需 findViewById 和 setClickListener 等的 IOC (Inverse Of Control 控制反转，就是将 UI 的初始化和事件绑定的“权利”交给框架来完成) 框架有：

1. Butter Knife <http://jakewharton.github.io/butterknife/>
2. xUtils 中的 ViewUtils 模块 <https://github.com/wyouflf/xUtils>

xUtils 中 ViewUtils 的基本使用方式如下：

```
1. // xUtils 的 view 注解要求必须提供 id，以使代码混淆不受影响。
2. @ViewInject(R.id.textview)
3. TextView textView;
4.
5. //@ViewInject(vale=R.id.textview, parentId=R.id.parentView)
6. //TextView textView;
7.
8. @ResInject(id = R.string.label, type = ResType.String)
9. private String label;
10.
```

```
11. // 取消了之前使用方法名绑定事件的方式，使用 id 绑定不受混淆影响
12. // 支持绑定多个 id @OnClick({R.id.id1, R.id.id2, R.id.id3})
13. // or @OnClick(value={R.id.id1, R.id.id2, R.id.id3}, parentId={R.id.pid1, R.id.pid2, R.id.pid3})
14. // 更多事件支持参见 ViewCommonEventListener 类和包 com.lidroid.xutils.view.annotation.event。
15. @OnClick(R.id.test_button)
16. public void testButtonClick(View v) { // 方法签名必须和接口中的要求一致
17.     ...
18. }
19. ...
20. //在 Activity 中注入：
21. @Override
22. public void onCreate(Bundle savedInstanceState) {
23.     super.onCreate(savedInstanceState);
24.     setContentView(R.layout.main);
25.     ViewUtils.inject(this); //注入 view 和事件
26.     ...
27.     textView.setText("some text...");
28.     ...
29. }
30. //在 Fragment 中注入：
31. @Override
32. public View onCreateView(LayoutInflater inflater, ViewGroup container,
33.     Bundle savedInstanceState) {
34.     View view = inflater.inflate(R.layout.bitmap_fragment, container, false);
35. // 加载 fragment 布局
36.     ViewUtils.inject(this, view); //注入 view 和事件
37.     ...
38. }
39. //在 PreferenceFragment 中注入：
40. public void onActivityCreated(Bundle savedInstanceState) {
41.     super.onActivityCreated(savedInstanceState);
42.     ViewUtils.inject(this, getPreferenceScreen()); //注入 view 和事件
43.     ...
44. }
```

接下来，我们开始编写自己的框架实现 findViewById 和 setOnClickListener 功能。

1. 编写自定义注解类 ViewInject 和 Click;

ViewInject 注解类用于添加在 Filed 上。Click 注解类用于添加到 Method 上。

【文件】ViewInject.java

```
1. package com.example.viewutils;
2.
```

```
3. import java.lang.annotation.ElementType;
4. import java.lang.annotation.Retention;
5. import java.lang.annotation.RetentionPolicy;
6. import java.lang.annotation.Target;
7. /**
8.  * @Retention 用于声明该注解生效的生命周期，有三个枚举值可以选择<br>
9.  * 1. RetentionPolicy.SOURCE 注释只保留在源码上面，编译成 class 的时候自动被编译器抹除
10. * 2. RetentionPolicy.CLASS 注释只保留到字节码上面，VM 加载字节码时自动抹除
11. * 3. RetentionPolicy.RUNTIME 注释永久保留，可以被 VM 加载时加载到内存中
12. * 注意：由于我们的目的是想在 VM 运行时对 Filed 上的该注解进行反射操作，因此 Retention 值必须设置为 RUNTIME
13. *
14. * @Target 用于指定该注解可以声明在哪些成员上面，常见的值有 FIELD 和 Method，
15. 由于我们的当前注解类是想声明在 Filed 上面
16. * 因此这里设置为 ElementType.FIELD。
17. * 注意：如果@Target 值不设置，则默认可以添加到任何元素上，不推荐这么写。
18. *
19. * @interface 是声明注解类的组合关键字。
20. */
21. @Retention(RetentionPolicy.RUNTIME)
22. @Target(ElementType.FIELD)
23. public @interface ViewInject {
24. /**
25.  * 定义 该注解的参数名和参数类型，value 是所有注解类默认的属性名。
26.  */
27. int value();
28. }
29.
```

【文件】Click.java

```
1. package com.example.viewutils;
2.
3. import java.lang.annotation.ElementType;
4. import java.lang.annotation.Retention;
5. import java.lang.annotation.RetentionPolicy;
6. import java.lang.annotation.Target;
7. /**
8.  *
9.  * @author wzy 2016-1-20
10. *
11. * 用于添加到方法上，在方法上指定 Button 的 id，也可以指定多个 id，也就是支持绑定多个 Button，
12. 当点击 Button 时通过反射调用方法。
13. */
14. @Retention(RetentionPolicy.RUNTIME)
15. @Target(ElementType.METHOD)
```

```
16. public @interface Click {
17. /**
18. * 数据类型的 int[]，目的在于支持让一个方法可以同时绑定多个 Button
19. */
20. int[] value();
21. }
```

2. 编写核心方法 YangeViewUtils.inject(Activity);

【文件】YangeViewUtils.java

```
1. package com.example.viewutils;
2.
3. import java.lang.reflect.Field;
4. import java.lang.reflect.Method;
5.
6. import android.app.Activity;
7. import android.view.View;
8. import android.view.View.OnClickListener;
9. /**
10. * 支持 UI 的绑定和方法的 setOnClickListener
11. *
12. * @author wzy 2016-1-20
13. *
14. */
15. public class YangeViewUtils {
16. public static void inject(final Activity activity){
17.     /**
18.     * 初始化控件然后赋值给对应的 Field
19.     */
20.     /**
21.     * 通过字节码获取 activity 类中所有的字段，在获取 Field 的时候一定要使用 getDeclaredFields(),
22.     * 因为只有该方法才能获取到任何权限修饰的 Filed，包括私有的。
23.     */
24.     Field[] declaredFields = activity.getClass().getDeclaredFields();
25.     //一个 Activity 中可能有多个 Field，因此遍历。
26.     for(int i=0;i<declaredFields.length;i++){
27.         Field field = declaredFields[i];
28.         //设置为可访问，暴力反射，就算是私有的也能访问到
29.         field.setAccessible(true);
30.         //获取到字段上面的注解对象
31.         YangeViewInject annotation = field.getAnnotation(YangeViewInject.class);
32.         //一定对 annotation 是否等于 null 进行判断，因为并不是所有 Filed 上都有我们想要的注解
33.         if (annotation!=null) {
34.             //获取注解中的值
```



```
35.         int id = annotation.value();
36.         //获取控件
37.         View view = activity.findViewById(id);
38.         //将该控件设置给 field 对象
39.         try {
40.             /**
41.              * 参数 1：当前 Field 所属的对象，显然这里是 activity
42.              * 参数 2：要给 Field 设置的值
43.              */
44.             field.set(activity, view);
45.         } catch (Exception e) {
46.             e.printStackTrace();
47.         }
48.     }
49. }
50.
51. /**
52.  * 给 Button 设置点击监听事件
53.  */
54. //获取所有的方法（私有方法也可以获取到）
55. Method[] declaredMethods = activity.getClass().getDeclaredMethods();
56. for(int i=0;i<declaredMethods.length;i++){
57.     final Method method = declaredMethods[i];
58.     //暴力反射
59.     method.setAccessible(true);
60.     //获取方法上面的注解
61.     Click annotation = method.getAnnotation(Click.class);
62.     if (annotation==null) {
63.         //如果该方法上没有注解，循环下一个
64.         continue;
65.     }
66.     //获取注解中的数据，因为可以给多个 button 绑定点击事件，因此定义注解类时使用的是 int[]作为数据类型。
67.     int[] ids = annotation.value();
68.     for(int j=0;j<ids.length;j++){
69.         //获取到 button 控件
70.         final View button = activity.findViewById(ids[j]);
71.         //给 button 绑定点击监听
72.         button.setOnClickListener(new OnClickListener() {
73.
74.             @Override
75.             public void onClick(View v) {
76.                 try {
77.                     //反射调用用户指定的方法
78.                     method.invoke(activity, button);
```

```
79.         } catch (Exception e) {
80.             e.printStackTrace();
81.         }
82.     }
83.     });
84. }
85.
86. }
87. }
88.
89. }
90.
```

3. 框架的使用

框架已经编写好，接下来就可以使用了，这里只给出简单的测试代码。

【文件】YangeViewUtils 的使用

```
1. public class MainActivity extends Activity {
2.
3.     @YangeViewInject(R.id.tv1)
4.     private TextView tv1;
5.     @YangeViewInject(R.id.et1)
6.     private EditText et1;
7.     @YangeViewInject(R.id.btn1)
8.     Button btn1;
9.     @YangeViewInject(R.id.btn2)
10.    Button btn2;
11.    @YangeViewInject(R.id.btn3)
12.    Button btn3;
13.
14.    @Override
15.    protected void onCreate(Bundle savedInstanceState) {
16.        super.onCreate(savedInstanceState);
17.        setContentView(R.layout.activity_main);
18.        YangeViewUtils.inject(this);
19.        Log.d("tag", "TextView="+tv1.getText());
20.    }
21.
22.    @Click({R.id.btn1,R.id.btn2,R.id.btn3})
23.    public void submit(View view){
24.        Toast.makeText(this, "您点击的 Button 为"+
25.            ((Button)view).getText(), Toast.LENGTH_SHORT).show();
26.    }
```

```
27.  
28. }
```

2. 编写 AsyncTask 框架 (2016-1-20)

编写自定义 AsyncTask 框架的前提有两个：1. 会使用 AsyncTask 2. 熟悉 AsyncTask 源码。

3. 编写 ImageLoader 框架 (2016-1-20)

自定义 ImageLoader 框架请参考如下视频 (

Android-仿微信图片选择器)，视频总长 3 个小时，认真看完后相信你会有巨大的收获：

<http://www.imooc.com/view/489>

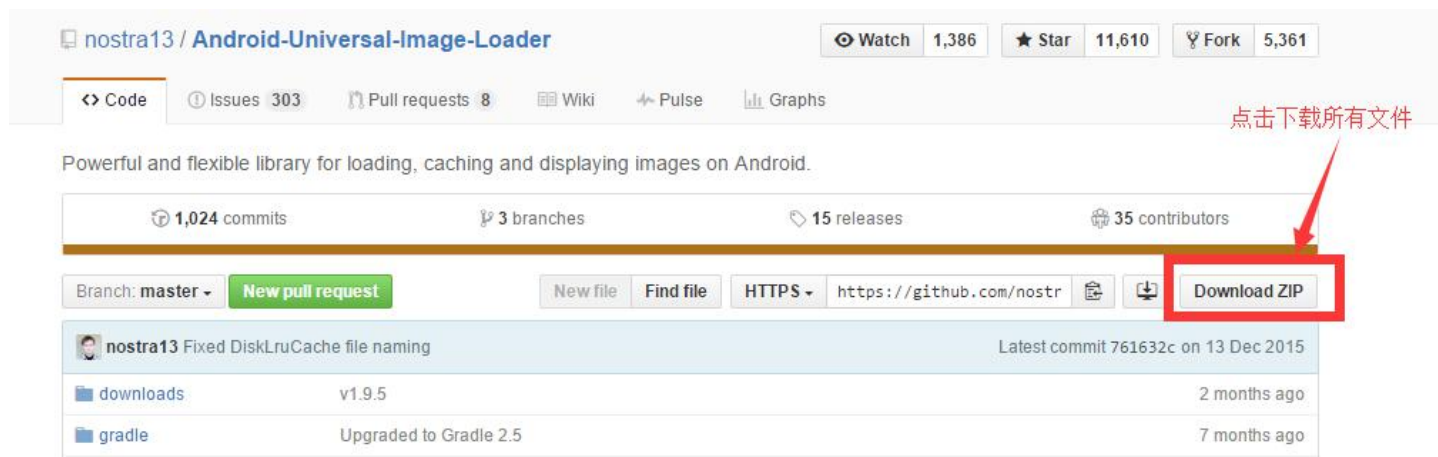
官方 ImageLoader 的使用如下：

官方地址：<https://github.com/nostra13/Android-Universal-Image-Loader>

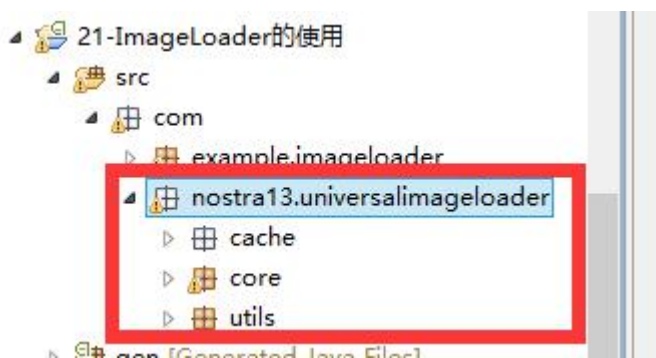
Android-Universal-Image-Loader 是一个开源的 UI 组件程序，该项目的目的是提供一个可异步实现图像加载，缓存和显示的框架。所以如果我们的程序里需要这个功能的话，那么不妨试试它。因为它已经封装好了一些类和方法。我们可以直接拿来用，而不用重复去写。其实，写一个这方面的程序还是比较麻烦的，要考虑多线程，缓存，内存溢出等很多方面。

一、下载源码或者 jar 包

从 ImageLoader 官网地址下载 zip 包。



将 zip 包中的如下文件夹拷贝到自己工程的 src 目录下（这里使用的是源码）：
Android-Universal-Image-Loader-master.zip\Android-Universal-Image-Loader-master\library\src\main\java
。添加到 src 目录后的效果图如下所示：



二、添加权限

因为 ImageLoader 主要目的是从网络下载图片，因此需要 INTERNET 权限，同时 ImageLoader 支持将数据缓存到 sdcard，因此需要添加 WRITE_EXTERNAL_STORAGE 权限。

1. `<uses-permission android:name="android.permission.INTERNET"/>`
2. `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>`

三、对 ImageLoader 进行全局配置

该过程必须在使用 ImageLoader 前配置，通常我们会在我们自定义 Application 的 onCreate() 方法中进行。

1. `ImageLoaderConfiguration config = new ImageLoaderConfiguration`
2. `.Builder(getApplicationContext())`
3. `.memoryCacheExtraOptions(480, 800) // max width, max height,`
4. 即保存的每个缓存文件的最大长宽
5. `.threadPoolSize(3) // 线程池内加载的数量`
6. `.threadPriority(Thread.NORM_PRIORITY - 2)`

```
7.         .denyCacheImageMultipleSizesInMemory()
8.         .memoryCache(new UsingFreqLimitedMemoryCache(2 * 1024 * 1024))
9. // You can pass your own memory cache implementation/你可以通过自己的内存缓存实现
10.        .memoryCacheSize(2 * 1024 * 1024)
11.        .discCacheSize(50 * 1024 * 1024)
12.        .discCacheFileNameGenerator(new Md5FileNameGenerator())
13. //将保存的时候的 URI 名称用 MD5 加密
14.        .tasksProcessingOrder(QueueProcessingType.LIFO)
15.        .discCacheFileCount(100) //缓存的文件数量
16.        .discCache(new UnlimitedDiskCache(
17. new File(Environment.getExternalStorageDirectory()+"/imageloader/")))//自定义缓存路径
18.        .defaultDisplayImageOptions(DisplayImageOptions.createSimple())
19.        .imageDownloader(new BaseImageDownloader(getApplicationContext(),
20. 5 * 1000, 30 * 1000)) // connectTimeout (5 s), readTimeout (30 s)超时时间
21.        .writeDebugLogs() // Remove for release app
22.        .build();//开始构建
23.        // Initialize ImageLoader with configuration.
24.        ImageLoader.getInstance().init(config);// 全局初始化此配置
```

四、在 Activity 中对 ImageLoader 进行局部配置

我们通常是在 Activity 中使用 ImageLoader，当然也可以是 Fragment 中。

(1) 在 Activity 中声明变量。

```
1. protected ImageLoader imageLoader = ImageLoader.getInstance();
2. private DisplayImageOptions options;
```

(2) 初始化 options

```
1. options = new DisplayImageOptions.Builder()
2.     .showImageOnLoading(R.drawable.ic_launcher) //设置图片在下载期间显示的图片
3.     .showImageForEmptyUri(R.drawable.ic_launcher)//设置图片 Uri 为空或是错误的时候显示的图片
4.     .showImageOnFail(R.drawable.ic_launcher)//设置图片加载/解码过程中错误时候显示的图片
5.     .cacheInMemory(true)//设置下载的图片是否缓存在内存中
6.     .cacheOnDisc(true)//设置下载的图片是否缓存在 SD 卡中
7.     .considerExifParams(true) //是否考虑 JPEG 图像 EXIF 参数 (旋转, 翻转)
8.     .imageScaleType(ImageScaleType.EXACTLY_STRETCHED)//设置图片以如何的编码方式显示
9.     .bitmapConfig(Bitmap.Config.RGB_565)//设置图片的解码类型//
10.    .decodingOptions(new android.graphics.BitmapFactory.Options())//设置图片的解码配置
11.    //delayBeforeLoading(int delayInMillis)//int delayInMillis 为你设置的下载前的延迟时间
12.    //设置图片加入缓存前, 对 bitmap 进行设置
13.    //preProcessor(BitmapProcessor preProcessor)
14.    .resetViewBeforeLoading(true)//设置图片在下载前是否重置, 复位
15.    .displayer(new RoundedBitmapDisplayer(20))//是否设置为圆角, 弧度为多少
16.    .displayer(new FadeInBitmapDisplayer(100))//是否图片加载好后渐入的动画时间
```

```
17.         .build();
```

注意：

```
1.  1) .imageScaleType(ImageScaleType imageScaleType) 是设置 图片的缩放方式
2.      缩放类型 imageScaleType:
3.          EXACTLY :图像将完全按比例缩小的目标大小
4.          EXACTLY_STRETCHED:图片会缩放到目标大小完全
5.          IN_SAMPLE_INT:图像将被二次采样的整数倍
6.          IN_SAMPLE_POWER_OF_2:图片将降低 2 倍, 直到下一减少步骤, 使图像更小的目标大小
7.          NONE:图片不会调整
8.  2) .displayer(BitmapDisplayer displayer) 是设置 图片的显示方式
9.      显示方式 displayer:
10.          RoundedBitmapDisplayer (int roundPixels) 设置圆角图片
11.          FakeBitmapDisplayer ( ) 这个类什么都没做
12.          FadeInBitmapDisplayer (int durationMillis) 设置图片渐显的时间
13.          SimpleBitmapDisplayer () 正常显示一张图片
```

(3) 在需要的地方使用 ImageLoader

```
1. imageLoader.displayImage(url, ImageView, options);
```

ImageLoader 有很多方法的重载，比如，如果不需要 options (则默认使用系统默认的配置：

```
1. imageLoader.displayImage(url, ImageView) ;
```

如果想添加图片加载的监听：

```
1. imageLoader.displayImage(imageUrl, imageView, options, new ImageLoadingListener() {
    @Override
    public void onLoadingStarted() {
        //开始加载的时候执行
    }
    @Override
    public void onLoadingFailed(FailReason failReason) {
        //加载失败的时候执行
    }
    @Override
    public void onLoadingComplete(Bitmap loadedImage) {
        //加载成功的时候执行
    }
    @Override
    public void onLoadingCancelled() {
        //加载取消的时候执行
    }
});
```

五、其他

ImageLoader 不仅可以从网络加载图片，也可以从本地加载图片：uri 的编写规范如下所示。

```
String imageUri = "http://site.com/image.png"; // from Web
```

```
String imageUri = "file:///mnt/sdcard/image.png"; // from SD card
```

```
String imageUri = "content://media/external/audio/albumart/13"; // from content provider
```

```
String imageUri = "assets://image.png"; // from assets
```

```
String imageUri = "drawable://" + R.drawable.image; // from drawables (only images, non-9patch)
```

九、其他知识（集大众智慧）

1、AsyncTask 如何使用

AsyncTask 用于处理异步任务，该类是一个抽象的泛型类。类的签名如下：public abstract class AsyncTask<Params, Progress, Result>。

三种泛型类型分别代表“启动任务执行的输入参数”、“后台任务执行的进度”、“后台计算结果的类型”。在特定场合下，并不是所有类型都被使用，如果没有被使用，可以用 java.lang.Void 类型代替。

一个异步任务的执行一般包括以下几个步骤：

1.execute(Params... params)，执行一个异步任务，需要我们在代码中调用此方法，触发异步任务的执行。

2.onPreExecute()，在 execute(Params... params)被调用后立即执行，一般用来在执行后台任务前对 UI 做一些标记。

3.doInBackground(Params... params)，在 onPreExecute()完成后立即执行，用于执行较为费时的操作，此方法将接收输入参数和返回计算结果。在执行过程中可以调用 publishProgress(Progress... values)来更新进度信息。

4.onProgressUpdate(Progress... values)，在调用 publishProgress(Progress... values)时，此方法被执行，直接将进度信息更新到 UI 组件上。

5.onPostExecute(Result result),当后台操作结束时,此方法将会被调用,计算结果将做为参数传递到此方法中,直接将结果显示到 UI 组件上。

在使用的时候,有几点需要格外注意:

- 1.异步任务的实例必须在 UI 线程中创建。
- 2.execute(Params... params)方法必须在 UI 线程中调用。
- 3.不要手动调用 onPreExecute(), doInBackground(Params... params), onProgressUpdate(Progress... values), onPostExecute(Result result)这几个方法。
- 4.不能在 doInBackground(Params... params)中更改 UI 组件的信息。
- 5.一个任务实例只能执行一次,如果执行第二次将会抛出异常。

我写一个简单的例子来演示 AsyncTask 的用法。

MainActivity.java


```
public class MainActivity extends Activity {

    private TextView tv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.tv);
    }
    /**
     * 点击 button 绑定的事件
     */
    public void start(View view){
        AsyncTask<Integer,Integer,String> asyncTask = new MyAsyncTask();
        asyncTask.execute(100);
    }
    class MyAsyncTask extends AsyncTask<Integer, Integer, String>{

        /**
         * 该方法在子线程中运行，因此不能有任何修改 UI 操作
         */
        @Override
        protected String doInBackground(Integer... params) {
            for(int i=0;i<params[0];i++){
                try {
                    //模拟耗时操作
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //发送进度
                publishProgress(i);
            }
            return "任务已经完成";
        }
        /**
         * 任务执行前在 UI 线程中调用
         */
    }
}
```

```
@Override
protected void onPreExecute() {
    Toast.makeText(MainActivity.this, "开始执行任务", 0).show();
    super.onPreExecute();
}
/**
 * 任务执行后在 UI 线程中调用<br>
 * @param result 正是 doInBackground 的返回值
 */
@Override
protected void onPostExecute(String result) {
    Toast.makeText(MainActivity.this, result, 0).show();
    super.onPostExecute(result);
}
/**
 * 在 UI 线程中执行
 * 当 doInBackground 执行 publishProgress 时调用该方法
 */
@Override
protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    tv.setText("当前进度: "+values[0]);
}
}
}
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="显示进度"
        android:textSize="30sp" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_margin="10dp"
        android:onClick="start"
        android:text="开始" />

</RelativeLayout>
```

运行效果图：

点击开始后，界面显示的数值不同的更改。



2、都使用过哪些框架、平台

2.1 EventBus（事件处理）

2.2 xUtils（网络、图片、ORM）

2.3 JPush（推送平台）

2.4 友盟（统计平台）

2.5 有米（优米）（广告平台）

2.6 百度地图

2.7 bmob（服务器平台、短信验证、邮箱验证、第三方支付）

2.8 阿里云 OSS（云存储）

2.9 ShareSDK（分享平台、第三方登录）

2.10 ImageLoader（图片加载器，支持多种缓存，避免 OOM）

3、Glide 原理（2015-11-29）

（Glide 是一个著名的网络访问框架，如果是面试初级程序员，可能只会问你会不会用，要是面试高级程序员可能会问 Glide 内部原理你懂不懂，会不会和懂不懂是不同层次的问题，希望大家从能会不会的层次提高到懂不懂原理的层次）

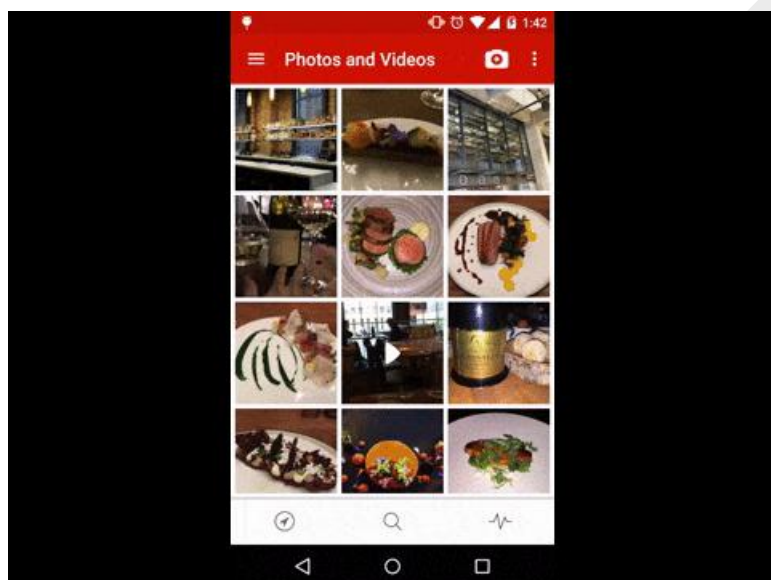
Glide 最简单的使用案例就是从远程服务器或者本地文件系统加载图片，把它们放在磁盘与内存缓存中，然后加载到 view 上。它可以用在全是图片的 app 中，Glide 为包含图片的滚动列表做了尽可能流畅的优化。

对象池

Glide 原理的核心是为 bitmap 维护一个对象池。对象池的主要目的是通过减少大对象的分配以重用来提高性能。

Dalvik 和 ART 虚拟机都没有使用 [compacting garbage collector](#) , compacting garbage collector 是一种模式，这种模式中 GC 会遍历堆，同时把活跃对象移到相邻内存区域，让更大的内存块可以用在后续的分配中。因为安卓没有这种模式，就可能会出现被分配的对象分散在各处，对象之间只有很小的内存可用。如果应用试图分配一个大于邻近的闲置内存块空间的对象，就会导致 OutOfMemoryError，然后崩溃，即使总的空余内存空间大于对象的大小。

使用对象池还可以帮助提高滚动的性能，因为重用 bitmap 意味着更少的对象被创建与回收。垃圾回收会导致“停止一切(Stop The World)”事件，这个事件指的是回收器执行期间，所有线程（包括 UI 线程）都会暂停。这个时候，图像帧无法被渲染同时 UI 可能会停滞，这在滚动期间尤其明显。



4、Android 四大著名图片处理框架 (2015-11-29)

	Imageloader	Picasso	Glide	Fresco
作者	nostra13	Square	Sam sjudd	Facebook
时间	2011/09	2013/02	2012/12	2015/03
Star/Issues Con/PR	10k/1k 35/120	7k/800 58/350	5k/600 22/61	6k/600 17/50
最新版本	1.9.5	2.5.3	4.0	0.7

Universal ImageLoader 是很早开源的图片缓存，在早期被很多应用使用。

Picasso 是 Square 开源的项目，且他的主导者是 JakeWharton，所以广为人知。

Glide 是 Google 员工的开源项目，被一些 Google App 使用，在去年的 Google I/O 上被推荐，不过目前国内资料不多。

Fresco 是 Facebook 在今年上半年开源的图片缓存，主要特点包括：

- (1) 两个内存缓存加上 Native 缓存构成了三级缓存
- (2) 支持流式，可以类似网页上模糊渐进式显示图片
- (3) 对多帧动画图片支持更好，如 Gif、WebP

鉴于 Fresco 还没发布正式的 1.0 版本，同时一直没太多时间熟悉 Fresco 源码，后面对比不包括 Fresco，以后有时间再加入对比。

4.1 基本概念

在正式对比前，先了解几个图片缓存通用的概念：

- (1) RequestManager：请求生成和管理模块

(2) Engine：引擎部分，负责创建任务(获取数据)，并调度执行

(3) GetDataInterface：数据获取接口，负责从各个数据源获取数据。

比如 MemoryCache 从内存缓存获取数据、DiskCache 从本地缓存获取数据，下载器从网络获取数据等。

(4) Displayer：资源(图片)显示器，用于显示或操作资源。

比如 ImageView，这几个图片缓存都不仅仅支持 ImageView，同时支持其他 View 以及虚拟的 Displayer 概念。

(5) Processor 资源(图片)处理器

负责处理资源，比如旋转、压缩、截取等。

以上概念的称呼在不同图片缓存中可能不同，比如 Displayer 在 ImageLoader 中叫做 ImageAware，在 Picasso 和 Glide 中叫做 Target。

4.2 共同优点

(1) 使用简单

都可以通过一句代码可实现图片获取和显示。

(2) 可配置度高，自适应程度高

图片缓存的下载器(重试机制)、解码器、显示器、处理器、内存缓存、本地缓存、线程池、缓存算法等大都可轻松配置。

自适应程度高，根据系统性能初始化缓存配置、系统信息变更后动态调整策略。

比如根据 CPU 核数确定最大并发数，根据可用内存确定内存缓存大小，网络状态变化时调整最大并发数等。

(3) 多级缓存

都至少有两级缓存、提高图片加载速度。

(4) 支持多种数据源

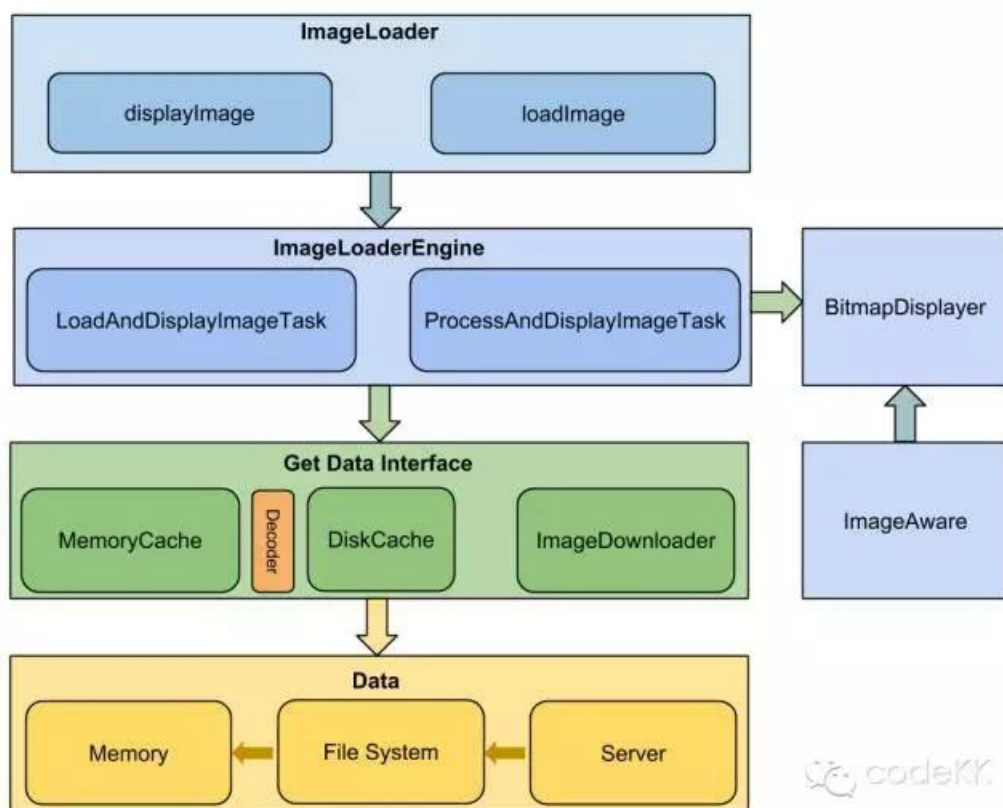
支持多种数据源，网络、本地、资源、Assets 等

(5) 支持多种 Displayer

不仅仅支持 ImageView，同时支持其他 View 以及虚拟的 Displayer 概念。

其他小的共同点包括支持动画、支持 transform 处理、获取 EXIF 信息等。

4.3 ImageLoader 设计及优点



4.3.1 总体设计及流程

上面是 ImageLoader 的总体设计图。整个库分为 ImageLoaderEngine，Cache 及 ImageDownloader，ImageDecoder，BitmapDisplayer，BitmapProcessor 五大模块，其中 Cache 分为 MemoryCache 和 DiskCache 两部分。

简单的讲就是 ImageLoader 收到加载及显示图片的任务，并将它交给 ImageLoaderEngine，ImageLoaderEngine 分发任务到具体线程池去执行，任务通过 Cache 及 ImageDownloader 获取图片，中间可能经过 BitmapProcessor 和 ImageDecoder 处理，最终转换为 Bitmap 交给 BitmapDisplayer 在 ImageAware 中显示。

4.3.2 ImageLoader 优点

(1) 支持下载进度监听

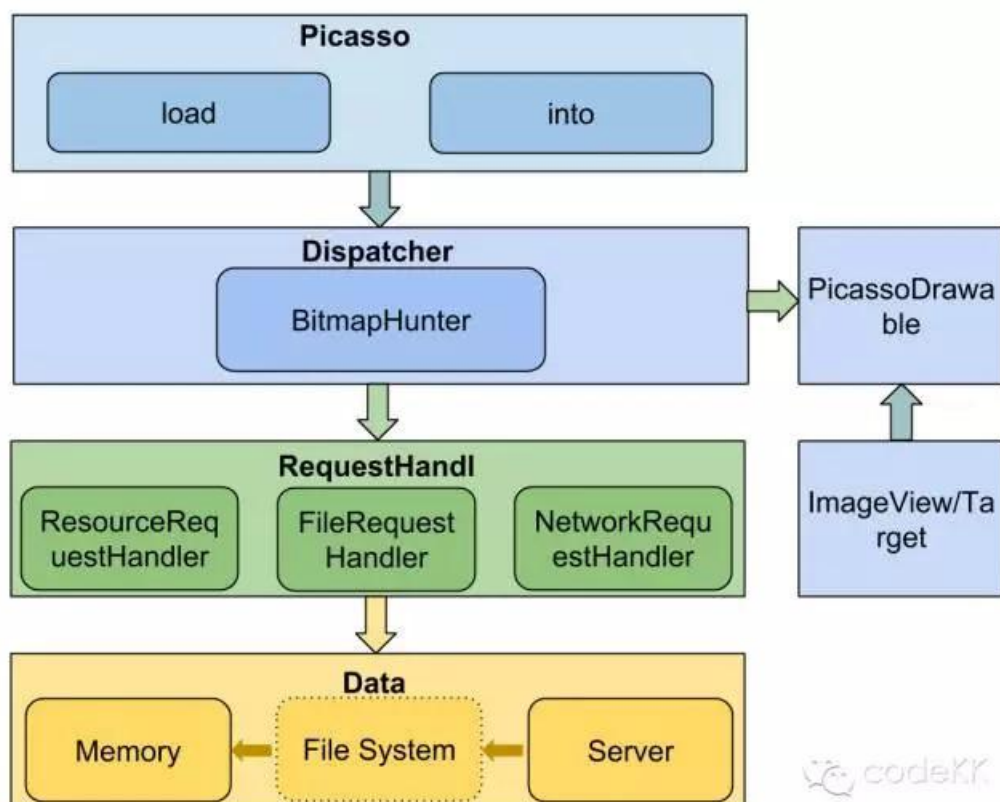
(2) 可以在 View 滚动中暂停图片加载

通过 PauseOnScrollListener 接口可以在 View 滚动中暂停图片加载。

(3) **默认实现多种内存缓存算法** 这几个图片缓存都可以配置缓存算法，不过 ImageLoader 默认实现了较多缓存算法，如 Size 最大先删除、使用最少先删除、最近最少使用、先进先删除、时间最长先删除等。

(4) 支持本地缓存文件名规则定义

4.4、Picasso 设计及优点



4.4.1 总体设计及流程

上面是 Picasso 的总体设计图。整个库分为 Dispatcher，RequestHandler 及 Downloader，PicassoDrawable 等模块。

Dispatcher 负责分发和处理 Action，包括提交、暂停、继续、取消、网络状态变化、重试等等。

简单的讲就是 Picasso 收到加载及显示图片的任务，创建 Request 并将它交给 Dispatcher，Dispatcher 分发任务到具体 RequestHandler，任务通过 MemoryCache 及 Handler(数据获取接口) 获取图片，图片获取成功后通过 PicassoDrawable 显示到 Target 中。

需要注意的是上面 Data 的 File system 部分，Picasso 没有自定义本地缓存的接口，默认使用 http 的本地缓存，API 9 以上使用 okhttp，以下使用 Urlconnection，所以如果需要自定义本地缓存就需要重定义 Downloader。

4.4.2 Picasso 优点

(1) 自带统计监控功能

支持图片缓存使用的监控，包括缓存命中率、已使用内存大小、节省的流量等。

(2) 支持优先级处理

每次任务调度前会选择优先级高的任务，比如 App 页面中 Banner 的优先级高于 Icon 时就很适用。

(3) 支持延迟到图片尺寸计算完成加载

(4) 支持飞行模式、并发线程数根据网络类型而变

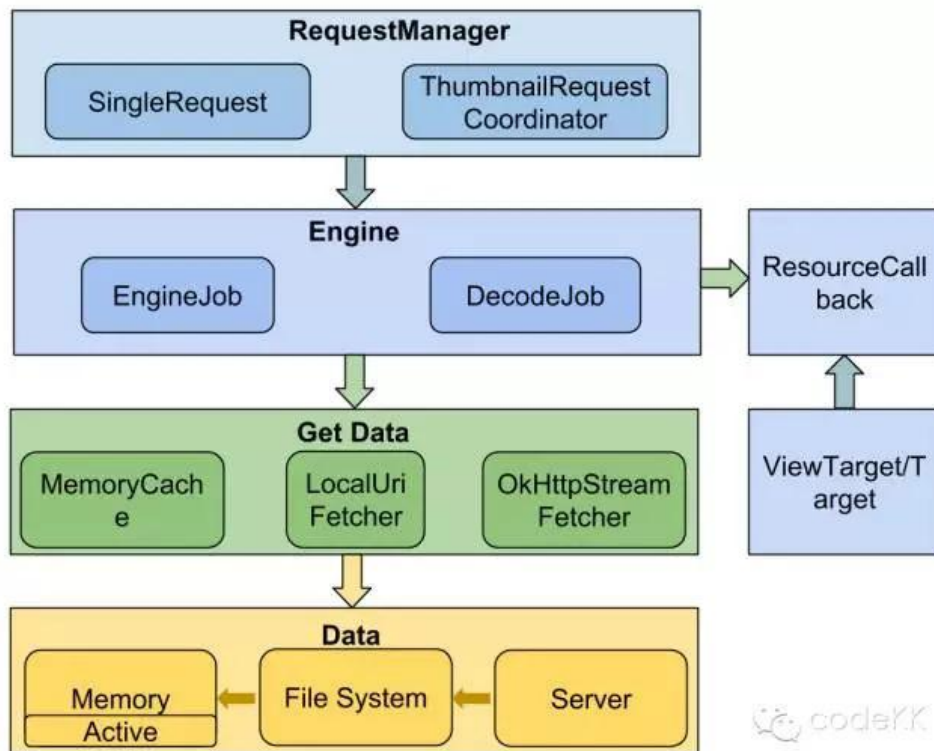
手机切换到飞行模式或网络类型变换时会自动调整线程池最大并发数，比如 wifi 最大并发为 4，4g 为 3，3g 为 2。

这里 Picasso 根据网络类型来决定最大并发数，而不是 CPU 核数。

(5) “无”本地缓存

无”本地缓存，不是说没有本地缓存，而是 Picasso 自己没有实现，交给了 Square 的另外一个网络库 okhttp 去实现，这样的好处是可以通过请求 Response Header 中的 Cache-Control 及 Expired 控制图片的过期时间。

4.5 Glide 设计及优点



4.5.1 总体设计及流程

上面是 Glide 的总体设计图。整个库分为 RequestManager(请求管理器), Engine(数据获取引擎)、Fetcher(数据获取器)、MemoryCache(内存缓存)、DiskLRUCache、Transformation(图片处理)、Encoder(本地缓存存储)、Registry(图片类型及解析器配置)、Target(目标) 等模块。

简单的讲就是 Glide 收到加载及显示资源的任务, 创建 Request 并将它交给 RequestManager, Request 启动 Engine 去数据源获取资源(通过 Fetcher), 获取到后 Transformation 处理后交给 Target。

Glide 依赖于 DiskLRUCache、GifDecoder 等开源库去完成本地缓存和 Gif 图片解码工作。

4.5.2 Glide 优点

(1) 图片缓存->媒体缓存

Glide 不仅是一个图片缓存, 它支持 Gif、WebP、缩略图。甚至是 Video, 所以更该当做一个媒体缓存。

(2) 支持优先级处理

(3) 与 Activity/Fragment 生命周期一致，支持 trimMemory

Glide 对每个 context 都保持一个 RequestManager，通过 FragmentTransaction 保持与 Activity/Fragment 生命周期一致，并且有对应的 trimMemory 接口实现可供调用。

(4) 支持 okhttp、Volley

Glide 默认通过 UrlConnection 获取数据，可以配合 okhttp 或是 Volley 使用。实际 ImageLoader、Picasso 也都支持 okhttp、Volley。

(5) 内存友好

① Glide 的内存缓存有个 active 的设计

从内存缓存中取数据时，不像一般的实现用 get，而是用 remove，再将这个缓存数据放到一个 value 为软引用的 activeResources map 中，并计数引用数，在图片加载完成后进行判断，如果引用计数为空则回收掉。

② 内存缓存更小图片

Glide 以 url、view_width、view_height、屏幕的分辨率等做为联合 key，将处理后的图片缓存在内存缓存中，而不是原始图片以节省大小

③ 与 Activity/Fragment 生命周期一致，支持 trimMemory

④ 图片默认使用默认 RGB_565 而不是 ARGB_888，虽然清晰度差些，但图片更小，也可配置到 ARGB_888。

其他：Glide 可以通过 signature 或不使用本地缓存支持 url 过期

4.5.3 Glide 的使用

Glide 使用起来很简单，而且不需要任何特别的配置就自动包含了 bitmap pooling。

```
1DrawableRequestBuilder requestBuilder = Glide.with(context).load(imageUrl);  
2requestBuilder.into(imageView);
```

这就是加载一张图片的全部要求。就像安卓中的很多地方一样，with() 方法中的 context 到底是哪种类型是不清楚的。有一点很重要需要记住，就是传入的 context 类型影响到 Glide 加载图片的优化程度，Glide 可以监视 activity 的生命周期，在 activity 销毁的时候自动取消等待中的请求。但是如果你使用 Application context，你就失去了这种优化效果。

译者注：其实以上的代码是一种比较规范的写法，我们更熟悉的写法是：

```
Glide.with(context)  
1    .load("http://inthecheesefactory.com/uploads/source/glidedepicasso/cover.j  
2    pg")  
3    .into(ivImg);
```

4.5.4 优化特性

类似的是，如果相关的 item 已经滚出了屏幕的范围，Glide 会自动取消列表中的悬着的图片请求。因为绝大多数开发者都会在 adapter 中利用 view 的回收，Glide 做到这点是通过在 ImageView 上设置一个 tag，在加载另外一张图片之前检查这个 tag，如果存在就取消第一次请求。

Glide 提供了几个让你感觉图片加载速度变快的特性。第一个就是在图片显示在屏幕上之前就预先取出图片。它提供了一个 ListPreloader 类，它被应该事先取出的 item 数目实例化。然后通过 setOnScrollListener(OnScrollListener) 被传递给 ListView。你希望在 ListView 之外也能预先取出图片吗？没问题，使用前面的 builder 对象就可以了，只需调用 builder.downloadOnly()。

4.6 汇总

	Imageloader	Picasso	Glide
作者	nostra13	Square	Sam sjudd
时间	2011/09	2013/02	2012/12
Star/Issues Con/PR	10k/1k 35/120	7k/800 58/350	5k/600 22/61
最新版本	1.9.5	2.5.3	4.0
功能			
代码理解			

三者总体上来说，ImageLoader 的功能以及代理容易理解长度都一般。

Picasso 代码虽然只在一个包下，没有严格的包区分，但代码简单、逻辑清晰，一两个小时就能叫深入的了解完。

Glide 功能强大，但代码量大、流转复杂。在较深掌握的情况下才推荐使用，免得出了问题难以下手解决。

5、都使用过哪些自定义控件

3.1 pull2RefreshListView

3.2 LazyViewPager

3.3 SlidingMenu

3.4 SmoothProgressBar

3.5 自定义组合控件

3.6 ToggleButton

3.7 自定义吐司

.....

6、Android 程序员进阶必备网站 (2015-11-29)

要想成为一名高级程序员，仅仅学会黑马的课程还是远远不够的。还需要我们平常的不断学习、总结、积累。推荐大家几个学习 Android 的网站。

4.1 黑马程序员官网 （编程，始于黑马）

<http://www.itheima.com/>


黑马程序员
www.itheima.com

为莘莘学子改变命运而讲课，为千万学生少走弯路而著书！

[新手指南](#)
[首页](#)
[课程介绍](#)
[视频教程](#)
[在线公开课](#)
[如何报名](#)
[黑马论坛](#)

[> 认识黑马，从这里开始！](#)
[> 我该选择学习哪门课程？](#)
[> 没有基础适不适合学编程？](#)
[> 我要实地考察，怎么走？](#)
[> 校区分布图，我该选择哪一所？](#)
[> 新生报到注意事项？](#)
[> 一部电影，看4个月的学习生活！](#)



Android 6.0 重磅推出

黑马程序员Android 6.0课程新品发布会
与2015年11月Android开发者大会同步

[▶ 抢先观看](#)

[我还有其他问题](#)
 一张贴看完
黑马所有微电影
  一张贴看完
黑马所有薪资
  一句话说出
你对黑马的印象
  登陆报名系统
开启黑马入学流程

4.2 CSDN （中国第一程序员社区）

<http://www.csdn.net/>




让移动连接你我
CSDN 移动客户端上线

[App store 下载](#)
[Android 市场下载](#)

[业界](#)
[云计算](#)
[移动](#)
[研发](#)
[学院](#)
[论坛](#)
[博客](#)
[下载](#)
[问答](#)
[ITeye](#)
[商城](#)
[CODE](#)
[活动](#)
[CTO](#)
[年费会员](#)
[外包](#)

[学院](#)
[线下培训](#)
[Java](#)
[iOS](#)
[C/C++](#)
[游戏开发](#)
[PHP](#)
[Azure](#)
[H3C](#)
[华为云计算](#)
[英特尔软件](#)
[IBM大学](#)
[异构开发](#)
[AWS](#)
[Qualcomm](#)
[腾讯云](#)
[容联](#)
[高校](#)



**芯片巨头为何痴恋开源软件？
英特尔Imad Sousou来解密**

在10月底举办的OpenStack东京峰会期间，记者见到了首次面对中国媒体的英特尔软件与服务事业部副总裁、英特尔开源技术中心总经理Imad Sousou。通过他公开了英特尔的开源战略。

临阵磨枪，血拼季网站优化的最后三板斧

京东王晓雨：Apache Kylin在云海的实践

Java 8 vs. Scala (二)：Stream vs. Collection

ECMAScript标准制定过程展示及ES7新特性披露

CSDN学院移动客户端，打造指尖上的IT课堂

深度剖析C++对象池自动回收技术实现

SDCC2015前端专场札记



豆瓣音乐
应用实战
HTML 5无处不在

快速入口 定制你的个性CSDN内容

推荐频道： BDTC 2015 SDCC 2015 微信开发学习路线 Rust路线图 CSDN学院 极客头条 Swift 智能硬件

推荐服务： C币兑换 博客专栏 精品资源

推荐专区： COSOS开源OS社区 触控科技 长跑节

4.3 51CTO（技术成就梦想）

<http://www.51cto.com/>

51CTO.com 技术成就梦想

社区 ▾ 学院 WOT MDSA 高招

输入您要搜索的内容

新闻 云计算 移动 开发 系统 网络 安全 数据库 服务器 虚拟化 博客 下载 论坛 读书 CIOAge

混合云社区 专题汇总 运维挑战 OpenStack五周年 开源PaaS方案 Python视频教程 下一代防火墙 Java20周年 安全报告 iOS实战开发 微

如何利用AngularJS打造一款简单Web应用 2/5

精选专题 | IBM Bluemix注册试用 | Linux运维高薪就业指导绝密资料 | Swift时代下iOS攻城狮学习路线图

最受管理员欢迎的顶级开源工具大盘点

血拼双十一 莫要购来了商品丢失了信息

今日重点

原创 | WOT2015谢佳标：基于R语言的大数据处理及建模技术

原创 | WOT2015李大学专访：互联网+浪潮下的颠覆与重构

原创 | WOT2015章天锋：用大数据把夜晚还给分拣员

原创 | WOT2015 大数据技术峰会靠什么勾引技术宅

原创 | WOT2015 刘志军：互联网和大数据倒逼消费金融发展

原创 | WOT2015刘明浩：互联网金融安全形势不容乐观

原创 | WOT2015罗奇斌：互联网和基因行业的融合是未来大趋势

原创 | 心理大师于际敬：技术人群心理特征及压力管理

4.4 ITEye (CSDN 旗下网站)

<http://www.iteye.com/>

ITEYE 100offer 互联网人才拍卖 告别盲目投简历 让海量好机会主动来找你 [了解更多](#)

首页 | 资讯 | 精华 | 论坛 | 问答 | 博客 | 专栏 | 群组 | 招聘 | 搜索

高手问答：Java多线程编程实战指南 (设计模式篇)

活动奖品：《Java多线程编程实战指南(设计模式篇)》

活动时间：11月23日~30日

高手问答：Java多线程编程实战指南 1 2 3

热点聚焦

十大算法，让你轻松进阶高手

快速排序算法、归并排序、二分查找算法、BFPRT(线性查找算法)、DFS(深度优先搜索)、BFS(广度优先搜索)、Dijkstra算法、动态规划算法、朴素贝叶斯分 [\[详情\]](#)

人人都该懂点儿TCP

即使你的工作也许不需要对TCP了如指掌，也不需要去了解具体的TCP/IP实例。你也应该懂一些基本的TCP知识，本文会告诉你为什么。 [\[详情\]](#)

致我们终将组件化的 Web

按模块划分“目录结构，把当前模块下的所有逻辑和资源都放一起了，这对于多人独自开发和维护个人模块不是很好吗？当然了，那争论的结果是我乖乖地改掉 [\[详情\]](#)

推荐精华文章

- [编程语言 IDE 对比](#)
- [最全的静态网站生成器\(开源项目\)](#)
- [Java NIO 系列教程](#)
- [Java并发教程 \(Oracle官方资料\)](#)
- [编程精华资源 \(ITeye优秀专栏\)](#)

4.5 eoeAndroid

<http://www.eoeandroid.com/portal.php>



最新消息

完整安卓 Android开发视频教程共10季，迅雷
第1季 - 《老罗Android开发视频教程》-android入门介绍-第1集：android学习路线图介绍

longchamp小羊皮 DuaAV pkaG 27hn
gif显示的问题
3E世界游戏平台专业的游戏运营平台
签到不了了怎么办？
android studio 安装在mac上，能否与手机进
android studio 如何打开eclipse创建的项目
如何让应用窗口全屏显示
Android 之 json数据的解析 (jsonReader)
AppCan 2015年度TOP10开发者团队评选

eoe推荐

8月7日，最新 TOP 10—Android精品源码
Android精品源码【免费下载】，包含日历效果实现、Fit Chart环形计数器等。

PullLoadMoreRecyclerView
android 面试技巧，本人面试多家公司经验获
Android Studio 权威教程
eoe专家答疑索引（汇总）帖，一帖看尽精华
安卓面试题大集【内含javaweb, J2ee，数
小团队长期承接苏州，上海android项目
接android项目，demo，UI,html5壳子
BoreDream模板代码-社交应用-新浪微博
迪恩客服端（医疗项目源码）

赞助商

云·端·创变
2015 AppCan移动云大会
11月27日 中国·北京
20W+年薪, 30+公司, 寻Android大
【推荐】极客校园大使等你来！

4.6 Android 巴士

<http://www.apkbus.com/>



安卓巴士 您好, 欢迎来到安卓巴士! 登录 注册 | 切换到宽版

APKBUS

HUAWEI | 华为云服务

首页 技术博客 开发者服务 开源代码 视频教程 线下活动 应聘招聘 开发工具

官方微博 巴友QQ群 精华专题 官方微信 认证开发

主题 回复 收藏 评论 动态

最新热门 精华区 我的安卓巴士

《Android开发工程师》微专业
够极客，真潮流
限时团购价，学费省到HIGH
Android开发工程师微专业

想成为产品经理?
网易五个亿级产品负责人亲授
限时团购价，学费省到high!

Android shape属性详细整理
有时候，为了满足一些需

【Android高级】Android系统以及Activity启动
一、Android系统启动[/backc

史上最简单的短信接口接入方法
这是我见过的一个简单的短信

Android软件开发之盘点自定义View界面大合
自定义View界面大合集 今天我自己写的一

Android Support Annotations 使用详解
在Android Support Library1

给 Android 开发者的 RxJava 详解
RxJava 到底是什么一个词：异

4.7 开源中国 (有 apk 可下载)

<http://www.oschina.net/>

首页 开源项目 问答 代码 博客 翻译 资讯 专题 城市圈 当前访客身份: 游客 [登录] [加入开源中国]

open source china 开源中国社区 oschina.net 众包 悬赏 作品 整包 我的 码云 Git Team PaaS Sonar 招聘 职位 简历 实习 招聘 在 39395 个开源项目中搜索 搜索

综合资讯 —— 源创会年终盛典 更多» 软件更新资讯 更多»

12月12日 北京 OSC 源创会 —— 国际会议中心
2015年源创会巡回的终点 - 北京，一场献给社区开发者的年度盛典正在酝酿中..... 源创会是开源中国坚持了...

- 众包翻译分享 —— 《Apache Mesos 官方文档》 11/29
- Fetchr —— web 应用通用数据访问层 11/29
- Linux 基金会说，没有 Linux 就没有 3D 电影 11/29
- OSChina 周日乱弹 —— 人生何必复杂 11/29
- Git 项目推荐 | 基于 Servlet 3 的 JFinal 改造版本 11/29
- 博客 | AFNetworking+NgInx+HTTPS 服务器通信 11/29
- Win 10 用户现在可以从桌面查看微软硬件产品 11/29
- VPN 协议漏洞暴露用户真实 IP 11/29
- OSChina 周六乱弹 —— 你敢把电脑借给父母用吗？ 11/28
- JTune : LinkedIn 的 Java CMS 高精度优化 11/28

Seafile 5.0.0 发布，开源文件云存储
Seafile 5.0.0 发布，该版本三大主要更新：Change most icons from png to icon font Most of the serv...

- mzept v2.0，一日开数站 PHP 极速框架 11/29
- Monkey 1.6.6 发布，Web 服务器 11/29
- CoreOS 877.1.0 Alpha 发布，服务器操作系统 11/29
- Birdfont 2.14 发布，字体编辑器 11/29
- GIMP 2.9.2 发布，开源图片处理软件 11/29
- TACTIC 4.4.0.v2 发布下载，生产资产管理系统 11/29
- Elive 2.6.12 (Beta) 发布 11/29
- FreeType 2.6.2 发布，字体工具 11/29
- Akka 2.4.1 发布，Actor 开发模型库 11/29
- Sequelize v3.14.2 发布，Node.js 的 ORM 11/29

七牛融合CDN全面降价，降幅达42%
今天你动弹了吗？热门头条大战！

今天你动弹了吗？ 动弹

小十郎：学了两天；发现打轮都不会；突然发现我的智商有点问题了
1分钟前 (0评) 0

javacc：周日第一次登陆os
12分钟前 (0评) 0

OSC大胖森：说实话，你有多久没吃过麻麻做的饭了？
17分钟前 (1评) iPhone 0

4.8 apk 源码（安卓源码服务专家，有上百个 apk 源码）

<http://www.javaapk.com/>

首页 VIP源码 例子大全11.23更新 基友群 运行不了？ 例如:2048 搜索

今天 2015年11月23日 星期一
您还没有登录，[登录] [注册] [找回密码]

最新发布

Android项目源码仿微信登录注册聊天换肤二维码扫描
JavaApk 2015年11月23日 0 3次 交友

本项目是一个基于安卓的Android仿微信客户端-猫友。是一个csdn上的朋友的原创项目，原帖可以看看这里<http://blog.csdn.net/ericfantastic/article/details/49451249> 实现了微信的登录注册、主界面、聊天会话、通讯录、发现界面、个人设置、添加好友.....

Android项目源码使用百度翻译接口的在线翻译
JavaApk 2015年11月22日 0 2次 翻译

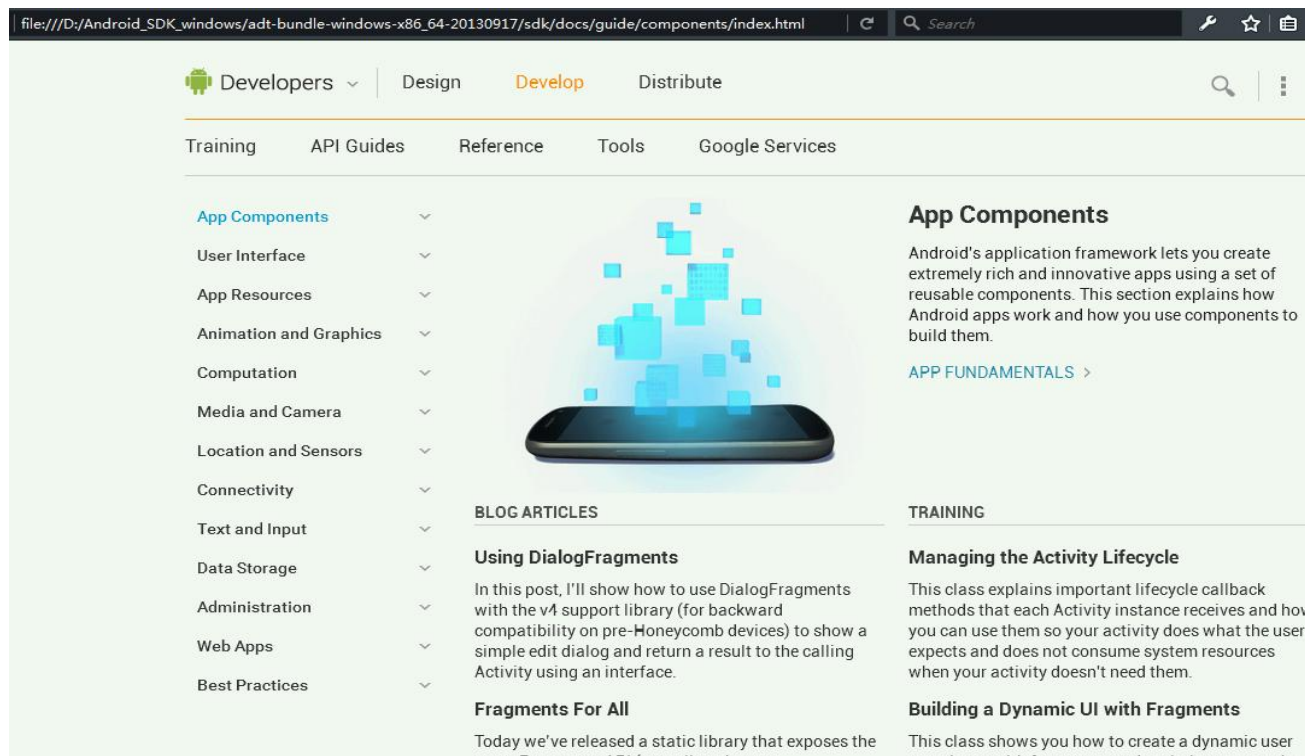
本项目是一个基于安卓的简单在线翻译app应用项目源码，试用了百度翻译的接口，通过直接带入字符串的方式进行翻译，可以翻译单个文字也可以翻译一段话，www.javaapk.com之前也介绍过很多关于翻译的项目源码，需要的可以在www.javaapk.com搜索“翻译”就可以找到下载。.....

例子源码

aidl	arduino	Bmob	EditText
GIF显示	GPS	IOS风格	JSON
ListView	MD5	NFC	ocr
SD卡	Socket	sqlite	TextView
UDP	ViewPager	VPN	webservice
wifi	WIN8风格	xml操作	Zip操作
上传下载	下拉刷新	九宫格锁屏	书籍翻页
产品介绍	仪表盘	传感器	侧滑菜单
其他	内存管理	列表控件	动态布局
动画特效	可穿戴	各种菜单	后台服务
图片剪裁	图片滤镜	图片轮播	地图定位
天气查询	字体使用	字母索引	屏幕截图
应用管理	开发框架	异步图片	引导页面

4.9 Android 官方文档

在 sdk 目录下的 docs 目录中。最好的文档，没有之一。很遗憾的就是无法访问其网站，只能离线查看。



4.10 泡在网上的日子

<http://www.jcodecraeer.com/>



7、volley 的原理 (2015-11-29)

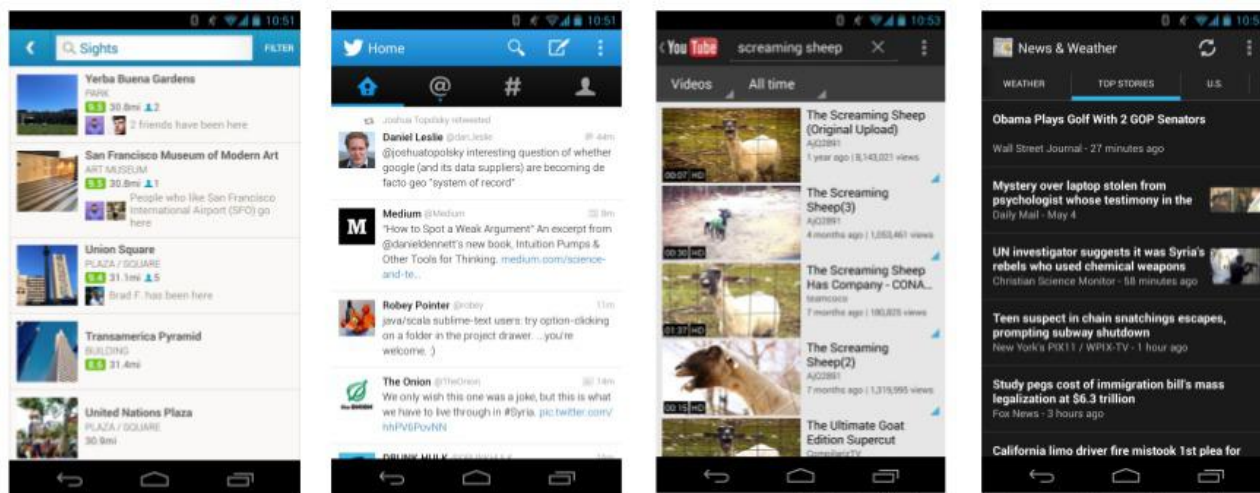
7.1 Volley 简介

我们平时在开发 Android 应用的时候不可避免地都需要用到网络技术，而多数情况下应用程序都会使用 HTTP 协议来发送和接收网络数据。Android 系统中主要提供了两种方式进行 HTTP 通信，HttpURLConnection 和 HttpClient，几乎在任何项目的代码中我们都能看到这两个类的身影，使用率非常高。

不过 HttpURLConnection 和 HttpClient 的用法还是稍微有些复杂的，如果不进行适当封装的话，很容易就会写出不少重复代码。于是乎，一些 Android 网络通信框架也就应运而生，比如说 AsyncHttpClient，它把 HTTP 所有的通信细节全部封装在了内部，我们只需要简单调用几行代码就可以完成通信操作了。再比如 Universal-Image-Loader，它使得在界面上显示网络图片的操作变得极度简单，开发者不用关心如何从网络上获取图片，也不用关心开启线程、回收图片资源等细节，Universal-Image-Loader 已经把一切都做好了。

Android 开发团队也是意识到了有必要将 HTTP 的通信操作再进行简单化，于是在2013年 Google I/O 大会上推出了一个新的网络通信框架——Volley。Volley 可是说是把 AsyncHttpClient 和 Universal-Image-Loader 的优点集于了一身，既可以像 AsyncHttpClient 一样非常简单地进行 HTTP 通信，也可以像 Universal-Image-Loader 一样轻松加载网络上的图片。除了简单易用之外，Volley 在性能方面也进行了大幅度的调整，它的设计目标就是非常适合去进行数据量不大，但通信频繁的网络操作，而对于大数据量的网络操作，比如说下载文件等，Volley 的表现就会非常糟糕。

下图所示的这些应用都是属于数据量不大，但网络通信频繁的，因此非常适合使用 Volley。



http://blog.csdn.net/guolin_blog

7.2 下载 Volley

介绍了这么多理论的东西，下面我们就准备开始进行实战了，首先需要将 Volley 的 jar 包准备好，如果你的电脑上装有 Git，可以使用如下命令下载 Volley 的源码：

下载完成后将它导入到你的 Eclipse 工程里，然后再导出一个 jar 包就可以了。如果你的电脑上没有 Git，那么也可以直接使用我导出好的 jar 包，下载地址是：http://www.kwstu.com/ResourcesView/kwstu_201441183330928。

新建一个 Android 项目，将 volley.jar 文件复制到 libs 目录下，这样准备工作就算是做好了。

7.3 StringRequest 的用法

前面已经说过，Volley 的用法非常简单，那么我们就从最基本的 HTTP 通信开始学习吧，即发起一条 HTTP 请求，然后接收 HTTP 响应。首先需要获取到一个 RequestQueue 对象，可以调用如下方法获取到：

```
RequestQueue mQueue = Volley.newRequestQueue(context);
```

注意这里拿到的 RequestQueue 是一个请求队列对象，它可以缓存所有的 HTTP 请求，然后按照一定的算法并发地发出这些请求。RequestQueue 内部的设计就是非常合适高并发的，因此我们不必为每一次 HTTP 请求都创建一个 RequestQueue 对象，这是非常浪费资源的，基本上在每一个需要和网络交互的 Activity 中创建一个 RequestQueue 对象就足够了。

接下来为了要发出一条 HTTP 请求，我们还需要创建一个 `StringRequest` 对象，如下所示：

```
StringRequest stringRequest = new StringRequest("http://www.baidu.com",  
  
        new Response.Listener<String>() {  
  
            @Override  
  
            public void onResponse(String response) {  
  
                Log.d("TAG", response);  
  
            }  
  
        }, new Response.ErrorListener() {  
  
            @Override  
  
            public void onErrorResponse(VolleyError error) {  
  
                Log.e("TAG", error.getMessage(), error);  
  
            }  
  
        });
```

可以看到，这里 `new` 出了一个 `StringRequest` 对象，`StringRequest` 的构造函数需要传入三个参数，第一个参数就是目标服务器的 URL 地址，第二个参数是服务器响应成功的回调，第三个参数是服务器响应失败的回调。其中，目标服务器地址我们填写的是百度的首页，然后在响应成功的回调里打印 出服务器返回的内容，在响应失败的回调里打印出失败的详细信息。

最后，将这个 `StringRequest` 对象添加到 `RequestQueue` 里面就可以了，如下所示：

```
mQueue.add(stringRequest);
```

另外，由于 `Volley` 是要访问网络的，因此不要忘记在你的 `AndroidManifest.xml` 中添加如下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```

好了，就是这么简单，如果你现在运行一下程序，并发出这样一条 HTTP 请求，就会看到 `LogCat` 中会打印出如

下图所示的数据。

```

Text
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN" "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"><!--STATUS OK--><head><meta http-equiv="Content-Type" content="text/
r: #262626;}form {position: relative;margin: 12px 15px 91px;height: 41px;}img {border: 0}.wordWrap{margin-right:
{background-color: #F5F5F5;border: 1px solid #787878;font-size: 16px;font-weight: bold;height: 41px;letter-spac
0}.lg {margin-top: 30px}a {text-decoration: none;color: #46446d}.a {margin-top: 20px}.d {margin: 58px 0 8px;}.l
ding-left: 18px;border-left: 1px solid #dadada;}</style><title>百度一下,你就知道</title><script>function form_subm
"http://m.baidu.com/static/index/u.png" alt="百度首页"/></div><form action="/from=844b/s" method="get" onsubmit="
nput type="hidden" value="0" name="ts"/><input type="hidden" name="ms" value="1"/></div><input type="submit" va
p;news?idx=30000&itj=311">文库</a><a href="http://m.baidu.com/img?idx=30000&ssid=0&from=844b&bd_
4%2Cta%40utouch_2_4.1__&itj=34&device=mobile">知道</a><a href="http://m.baidu.com/ssid=0/from=844b/bd_pa
40utouch_2_4.1__&itj=31">百科</a></div><div class="a"><a href="http://m.baidu.com/ssid=0/from=844b/bd_page_t
321_1004%2Cta%40utouch_2_4.1__&idx=30000&itj=35&wtj=wi">地图</a><a href="http://tieba.baidu.com/?idx
type=1&uid=0&pu=sz%401321_1004%2Cta%40utouch_2_4.1__&idx=30000&itj=39">hao123</a><a href="http:/
lass="d"><div class="b"><a href="http://duokoo.baidu.com/novel/?fr=home&ssid=0&from=844b&bd_page_ty
http://blog.csdn.net/guolin_blog

```

没错，百度返回给我们的就是这样一长串的 **HTML** 代码，虽然我们看起来会有些吃力，但是浏览器却可以轻松地对这段 **HTML** 代码进行解析，然后将百度的首页展现出来。

这样的话，一个最基本的 HTTP 发送与响应的功能就完成了。你会发现根本还没写几行代码就轻易实现了这个功能，主要就是进行了以下三步操作：

1. 创建一个 `RequestQueue` 对象。
2. 创建一个 `StringRequest` 对象。
3. 将 `StringRequest` 对象添加到 `RequestQueue` 里面。

不过大家都知道，**HTTP** 的请求类型通常有两种，**GET** 和 **POST**，刚才我们使用的明显是一个 **GET** 请求，那么如果想要发出一条 **POST** 请求应该怎么做 呢？**StringRequest** 中还提供了另外一种四个参数的构造函数，其中第一个参数就是指定请求类型的，我们可以使用如下方式进行指定：

```
StringRequest stringRequest = new StringRequest(Method.POST, url, listener, errorListener);
```

可是这只是指定了 **HTTP** 请求方式是 **POST**，那么我们要提交给服务器的参数又该怎么设置呢？很遗憾，**StringRequest** 中并没有提供设置 **POST** 参数的方法，但是当发出 **POST** 请求的时候，**Volley** 会尝试调用 **StringRequest** 的父类——**Request** 中的 **getParams()** 方法 来获取 **POST** 参数，那么解决方法自然也就有了，我们只需要在 **StringRequest** 的匿名类中重写 **getParams()** 方法，在这里设置 **POST** 参数就可以了，代码如下所示：

```
StringRequest stringRequest = new StringRequest(Method.POST, url, listener, errorListener) {
```

```
@Override
```

```
protected Map<String, String> getParams() throws AuthFailureError {
```

```
    Map<String, String> map = new HashMap<String, String>();
```

```
    map.put("params1", "value1");
```

```
    map.put("params2", "value2");
```

```
    return map;
```

```
}
```

```
};
```

你可能会说，每次都这样用起来岂不是很累？连个设置 **POST** 参数的方法都没有。但是不要忘记，**Volley** 是开源的，只要你愿意，你可以自由地在里面添加和修改任何的方法，轻松就能定制出一个属于你自己的 **Volley** 版本。

7.4. JsonRequest 的用法

学完了最基本的 **StringRequest** 的用法，我们再来进阶学习一下 **JsonRequest** 的用法。类似于 **StringRequest**，**JsonRequest** 也是继承自 **Request** 类的，不过由于 **JsonRequest** 是一个抽象类，因此我们无法直接创建它的实例，那么只能从它的子类入手了。**JsonRequest** 有两个直接子类，**JsonObjectRequest** 和 **JSONArrayRequest**，从名字上你应该能就看出它们的区别了吧？一个是用于请求一段 **JSON** 数据的，一个是用于请求一段 **JSON** 数组的。

至于它们的用法也基本上没有什么特殊之处，先 **new** 出一个 **JsonObjectRequest** 对象，如下所示：

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest("http://m.weather.com.cn/data/
```

```
101010100.html", null, new Response.Listener<JSONObject>() {
```

```
    @Override
```

```
    public void onResponse(JSONObject response) {
```

```
        Log.d("TAG", response.toString());
```

```
    }
```

```
}, new Response.ErrorListener() {  
  
    @Override  
  
    public void onErrorResponse(VolleyError error) {  
  
        Log.e("TAG", error.getMessage(), error);  
  
    }  
  
});
```

可以看到，这里我们填写的 URL 地址是 <http://m.weather.com.cn/data/101010100.html>，这是中国天气网提供的一个查询天气信息的接口，响应的数据就是以 JSON 格式返回的，然后我们在 `onResponse()` 方法中将返回的数据打印出来。

最后再将这个 `JsonObjectRequest` 对象添加到 `RequestQueue` 里就可以了，如下所示：

```
mQueue.add(jsonObjectRequest);
```

这样当 HTTP 通信完成之后，服务器响应的天气信息就会回调到 `onResponse()` 方法中，并打印出来。现在运行一下程序，发出这样一条 HTTP 请求，就会看到 LogCat 中会打印出如下图所示的数据。

由此可以看出，服务器返回给我们的数据确实是 JSON 格式的，并且 `onResponse()` 方法中携带的参数也正是一个 `JSONObject` 对象，之后只需要从 `JSONObject` 对象取出我们想要得到的那部分数据就可以了。

你应该发现了吧，`JsonObjectRequest` 的用法和 `StringRequest` 的用法基本上是完全一样的，Volley 的易用之处也在这里体现出来了，会了一种就可以让你举一反三，因此关于 `JsonArrayRequest` 的用法相信已经不需要我再去讲解了吧。

8、okhttp 的原理 (2015-11-29)

主要是通过 Dispatcher 不断从 `RequestQueue` 中取出请求 (Call)，根据是否已缓存调用 Cache 或 Network 这两类数据获取接口之一，从内存缓存或是服务器取得请求的数据。

该引擎有同步和异步请求，同步请求通过 `Call.execute()` 直接返回当前的 `Response`，而异步请求会把当前的请

求 Call.enqueue 添加 (AsyncCall) 到请求队列中，并通过回调 (Callback) 的方式来获取最后结果。

参考博客：<http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0326/2643.html>

9、ViewPagerindicator 的原理 (2015-11-29)

该项目总体设计非常简单，一个 pageIndicator 接口类，具体样式的导航类实现该接口，然后根据具体样式去实现相应的逻辑。IcsLinearLayout : LinearLayout 的扩展，支持了 4.0 以上的 divider 特性。

CirclePageIndicator、LinePageIndicator、UnderlinePageIndicator、TitlePagerIndicator 继承自 View。
TabPageIndicator、IconPageIndicator 继承自 HorizontalScrollView。

CirclePageIndicator、LinePageIndicator、UnderlinePageIndicator 继承自 View 的原因是它们样式相对简单，继承 View，自己去定制一套测量和绘制逻辑更简单，而且免去了 Measure 部分繁琐的步骤，效率更高。

TitlePagerIndicator 相对复杂，Android 系统提供的控件中没有类似的，而且实现底部线条的精准控制也复杂，所以只能继承自 View，实现绘制逻辑，达到理想的 UI 效果。

TabPageIndicator、IconPageIndicator 继承自 HorizontalScrollView 是由于它们各自的 ChildView 较多，而且具有相似性些，继承自 LinearLayout，通过 for 循环一个个 add 上去更简单，而且 HorizontalScrollView 具有水平滑动的功能，当 tab 比较多时，可以左右滑动。

参考博客：<http://blog.csdn.net/ljx19900116/article/details/43482083>

10、slidingmenu 的原理 (2015-11-29)

SlidingMenu 设计思路 三个主要的 ViewGroup, 主 ViewGroup 里面包含两个重叠的 ViewGroup, 盖在上面的就

是显示内容,而下面的就是菜单上面的侧滑以后露出后面的 View

主 ViewGroup 作为一个自定义控件,里面的内容和菜单利用自定义属性设置 SlidingMenu(继承 RelativeLayout),就是这个主 ViewGroup,其中又包含两个,内容 CustomViewAbove 和 菜单 CustomViewBehind(都继承 ViewGroup),SlidingMenu 提供两个自定义属性,供使用者传入两个布局 id,对应主体内容和菜单布局,此外还有其他一些属性供开发者设置菜单效果。

参考博客：<http://www.tuicool.com/articles/eErqY3>

11、RecyclerView 的原理 (2017-2-25)

1. RecyclerView 的功能：GridView 的功能，横向 ListView 的功能，横向 ScrollView 的功能，瀑布流效果，便于添加 Item 增加和移除动画。

原理：RecyclerView 继承自 ViewGroup，在绘制中，RecyclerView 是将 onMeasure(),onLayout(),交给我们 LayoutManager 去处理的，所以设置不同的 LayoutManager 就会达到不同的效果。

2. itemDecorattion 的原理，我们的 itemDecoration 本质上是一个 Drawable，RecyclerView 当他执行到 onDraw 的时候，如果重写了这个方法，相当于在 RecyclerView 上画了一个 Drawable 的东西，然后他的内部，就有一个方法叫 getItemoffsets 这个方法，让每一个 item 往后偏移。

3. Adapter 的原理：是一个适配器，和 ListView 不同的是，ListView 的适配器是直接返回 View。RecyclerView 返回的是 ViewHolder，它在缓存区域缓存的是 holder。

RecyclerView 的缓存原理：三级缓存，第一级是两个 ArrayList（如下源码截图），RecyclerView 中那些仍依赖于 RecyclerView 的，但是被标记移除的 itemView 会添加到 mAttachedScrap 集合中，没有被标记的则会添加到 mCachedViews。第二级是 ViewCacheExtension，称为附加缓存池，第一级缓存找不到了就到从 ViewCacheExtension 找。第三级缓存，是 RecycledViewPool，将 ViewHolder 放入 ArrayList 中，之后会放到，Map 集合，系统根据不同 itemType 来去 ViewHolder。


```
5174         final ArrayList<ViewHolder> mAttachedScrap = new ArrayList<>();
5175         ArrayList<ViewHolder> mChangedScrap = null;
5176
5177         final ArrayList<ViewHolder> mCachedViews = new ArrayList<>();
5187
5188         private ViewCacheExtension mViewCacheExtension;
5189
5190         static final int DEFAULT_CACHE_SIZE = 2;
5191
5192     /**
5193      * RecycledViewPool lets you share Views between multiple RecyclerViews.
5194      *
5195      * <p>
5196      * If you want to recycle views across RecyclerViews, create an instance of RecycledViewPool
5197      * and use {@link RecyclerView#setRecycledViewPool\(RecycledViewPool\)}.
5198      *
5199      * <p>
5200      * RecyclerView automatically creates a pool for itself if you don't provide one.
5201      *
5202     public static class RecycledViewPool {
5203         private static final int DEFAULT_MAX_SCRAP = 5;
```

十、网络协议

1、Http 和 Https 有什么区别？（2017-2-24）

一、HTTP 和 HTTPS 的基本概念

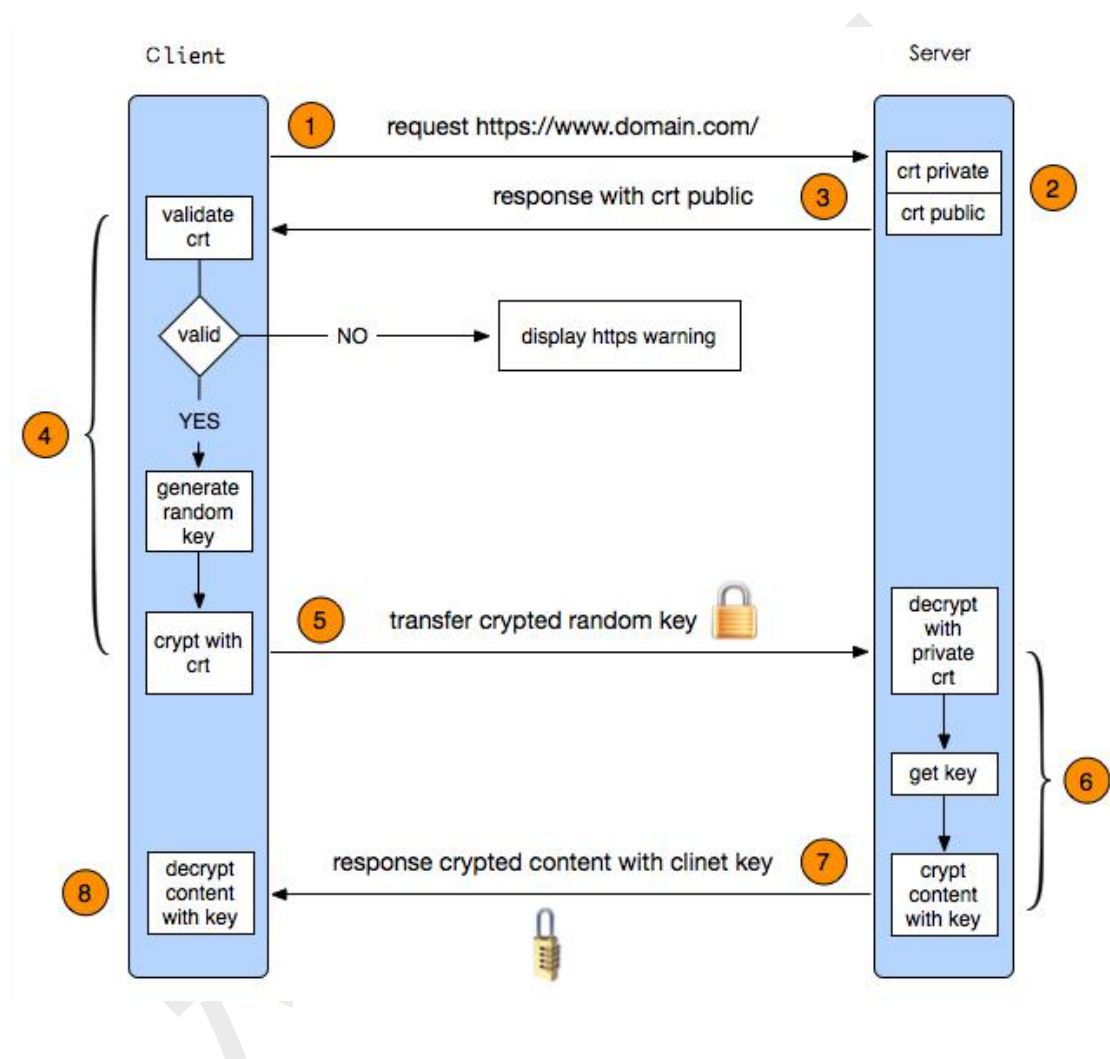
HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务端请求和应答的标准（TCP），用于从 WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

HTTPS：是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。HTTPS 协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

二、HTTP 与 HTTPS 有什么区别？

HTTP 协议传输的数据都是未加密的，也就是明文的，因此使用 HTTP 协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是美国的 NetScape(网景)公司设计了 SSL (Secure Sockets Layer) 协议用于对 HTTP 协议传输的数据进行加密，从而就诞生了 HTTPS。

HTTPS 加密、及验证过程，如下图所示：



简单来说，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全。

HTTPS 和 HTTP 的区别主要如下：

1、https 协议需要到 ca 申请证书(数字证书授权颁发中心，常用正证书申请地址：

<https://freessl.wosign.com/>)，一般免费证书较少，因而需要一定费用。

2、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

三、HTTPS 的工作原理

我们都知道 HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。



1、客户端发起 HTTPS 请求

这个没什么好说的，就是用户在浏览器里输入一个 https 网址，然后连接到 server 的 443 端口。

2、服务端的配置

采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面(startssl 就是个不错的选择，有 1 年的免费服务)。

这套证书其实就是一对公钥和私钥，如果对公钥和私钥不太理解，可以想象成一把钥匙和一个锁头，只是全世界只有你一个人有这把钥匙，你可以把锁头给别人，别人可以用这个锁把重要的东西锁起来，然后发给你，因为只有你一个人有这把钥匙，所以只有你才能看到被这把锁锁起来的东西。

3、传送证书

这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间等等。

4、客户端解析证书

这部分工作是由客户端的 TLS 来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密，就好像上面说的，把随机值用锁头锁起来，这样除非有钥匙，不然看不到被锁住的内容。

5、传送加密信息

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。

6、服务端解密信息

服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密，所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

7、传输加密后的信息

这部分信息是服务端用私钥加密后的信息，可以在客户端被还原。

8、客户端解密信息

客户端用之前生成的私钥解密服务端传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听到了数据，也束手无策。

四、HTTPS 的优点

正是由于 HTTPS 非常的安全，攻击者无法从中找到下手的地方，从站长的角度来说，HTTPS 的优点有以下 2 点：

1、SEO（搜索引擎优化，就是如何让自己的网站排名更前）方面

谷歌曾在 2014 年 8 月份调整搜索引擎算法，并称“比起同等 HTTP 网站，采用 HTTPS 加密的网站在搜索结果中的排名将会更高”。

2、安全性

尽管 HTTPS 并非绝对安全，掌握根证书的机构、掌握加密算法的组织同样可以进行中间人形式的攻击，但 HTTPS 仍是现行架构下最安全的解决方案，主要有以下几个好处：

五、HTTPS 的缺点

虽然说 HTTPS 有很大的优势，但其相对来说，还是有些不足之处的，具体来说，有以下 2 点：

1、SEO 方面

据 ACM CoNEXT 数据显示，使用 HTTPS 协议会使页面的加载时间延长近 50%，增加 10%到 20%的耗电，此外，HTTPS 协议还会影响缓存，增加数据开销和功耗，甚至已有安全措施也会受到影响也会因此而受到影响。而且 HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

2、经济方面

(1)、SSL 证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

(2)、SSL 证书通常需要绑定 IP，不能在同一个 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗（SSL 有扩展可以部分解决这个问题，但是比较麻烦，而且要求浏览器、操作系统支持，Windows XP 就不支持这个扩展，考虑到 XP 的装机量，这个特性几乎没用）。

(3)、HTTPS 连接缓存不如 HTTP 高效，大流量网站如非必要也不会采用，流量成本太高。

(4)、HTTPS 连接服务器端资源占用高很多，支持访客稍多的网站需要投入更大的成本，如果全部采用 HTTPS，基于大部分计算资源闲置的假设的 VPS 的平均成本会上去。

(5)、HTTPS 协议握手阶段比较费时，对网站的相应速度有负面影响，如非必要，没有理由牺牲用户体验。

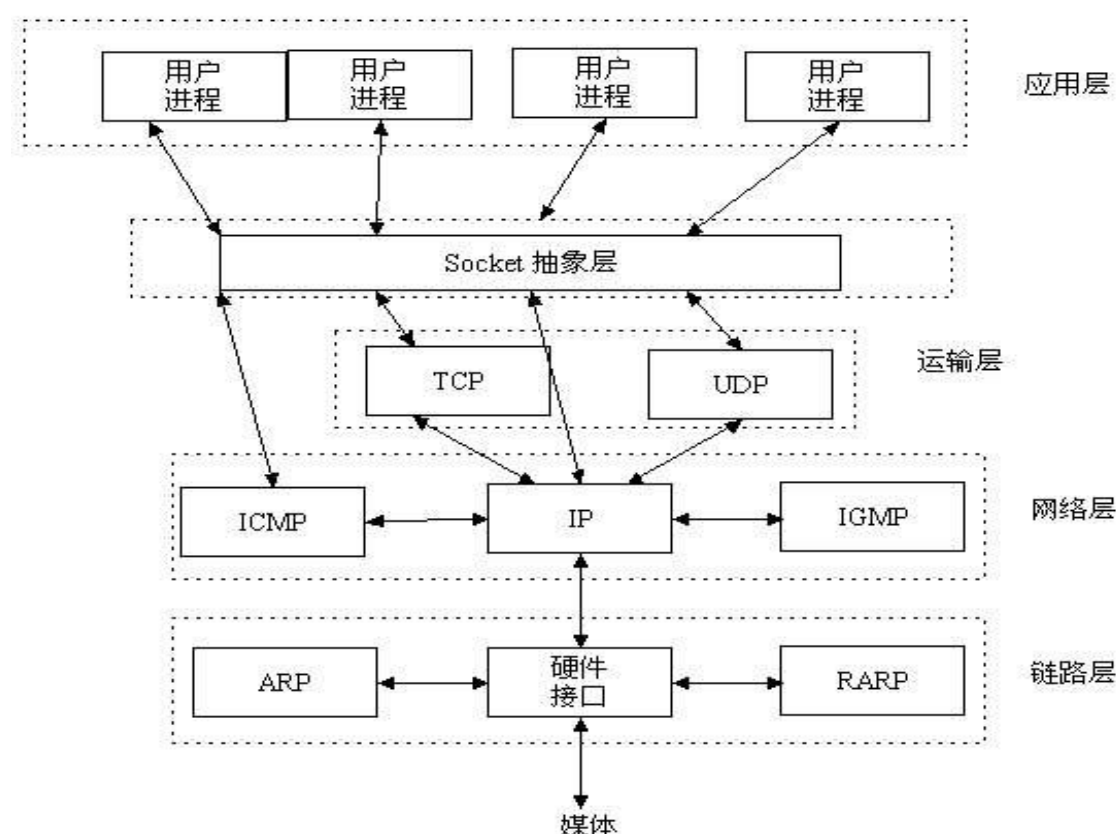
2、简述 Socket 通讯编程（2017-2-25）

了解 Socket 通讯编程先要了解什么是 TCP/IP、UDP？

TCP (Transmission Control Protocol)和 UDP(User Datagram Protocol)协议属于传输层协议。其中 TCP 提供 IP 环境下的数据可靠传输，它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复用。通过面向连接、端到端和可靠的数据包发送。通俗说，它是事先为所发送的数据开辟出连接好的通道，然后再进行数据发送；而 UDP 则不为 IP 提供可靠性、流控或差错恢复功能。一般来说，TCP 对应的是可靠性要求高的应用，而 UDP 对应的则是可靠性要求低、传输经济的应用。

Socket 是什么呢？

Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。如图：



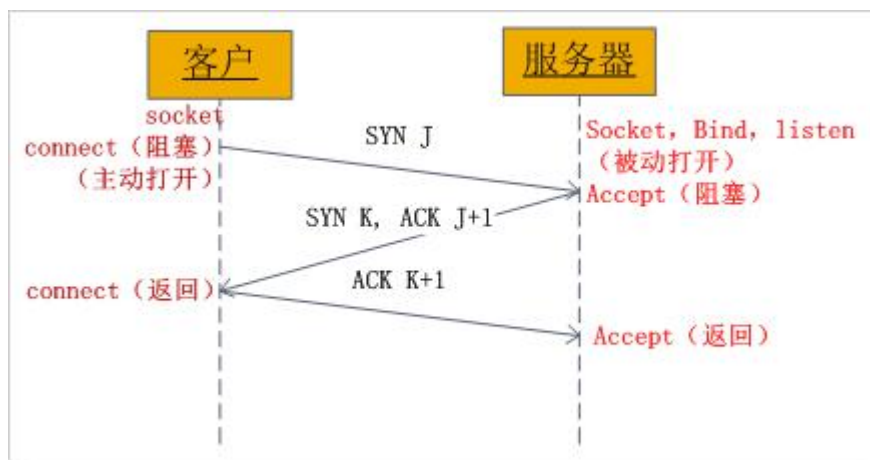
Socket 起源于 Unix，而 Unix/Linux 基本哲学之一就是“一切皆文件”，都可以用“打开 open -> 读写 write/read -> 关闭 close”模式来操作。Socket 就是该模式的一个实现，Socket 即是一种特殊的文件，一些 Socket 函数就是对其进行的操作（读/写 IO、打开、关闭），有相对应的函数可操作。

Socket 中 TCP 的三次握手建立连接详解

我们知道 tcp 建立连接要进行“三次握手”，大致流程如下：

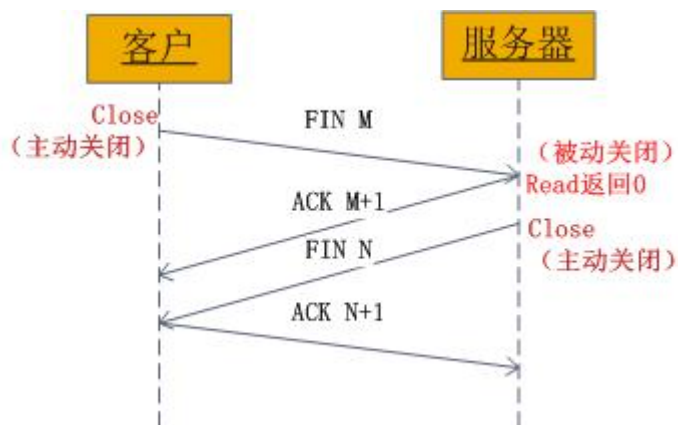
- 1.客户端向服务器发送一个 SYN (synchronous) J
- 2.服务器向客户端响应一个 SYN K，并对 SYN J 进行确认 ACK J+1
- 3.客户端再想服务器发一个确认 ACK K+1

但是这个三次握手发生在 socket 的那几个函数中呢？请看下图：



从图中可以看出，当客户端调用 `connect` 时，触发了连接请求，向服务器发送了 SYN J 包，这时 `connect` 进入阻塞状态；服务器监听到连接请求，即收到 SYN J 包，调用 `accept` 函数接收请求向客户端发送 SYN K，ACK J+1，这时 `accept` 进入阻塞状态；客户端收到服务器的 SYN K，ACK J+1 之后，这时 `connect` 返回，并对 SYN K 进行确认；服务器收到 ACK K+1 时，`accept` 返回，至此三次握手完毕，连接建立。

上面介绍了 Socket 中 TCP 的三次握手建立过程，及其涉及的 Socket 函数。现在我们介绍 Socket 中的四次握手释放连接的过程，请看下图：



1. 某个应用进程首先调用 close 主动关闭连接，这时 TCP 发送一个 FIN M；
2. 另一端接收到 FIN M 之后，执行被动关闭，对这个 FIN 进行确认。它的接收也作为文件结束符传递给应用进程，因为 FIN 的接收意味着应用进程在相应的连接上再也接收不到额外数据；
3. 一段时间之后，接收到文件结束符的应用进程调用 close 关闭它的 socket。这导致它的 TCP 也发送一个 FIN N；
4. 接收到这个 FIN 的源发送端 TCP 对它进行确认。

这样每个方向上都有一个 FIN 和 ACK。

3、Binder 机制 (2017-2-25)

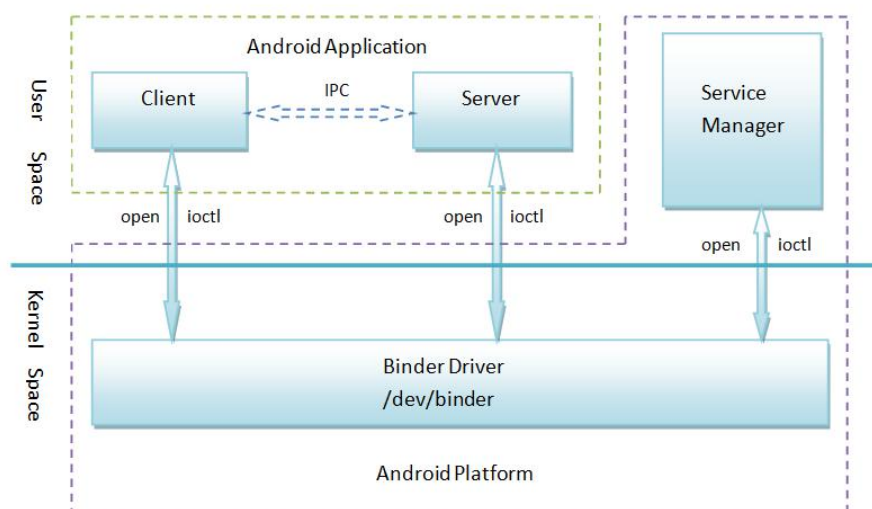
首先 Binder 是 Android 系统进程间通信(IPC)方式之一。

Binder 使用 Client - Server 通信方式。Binder 框架定义了四个角色：Server, Client, ServiceManager 以及 Binder 驱动。其中 Server, Client, ServiceManager 运行于用户空间，Binder 驱动运行于内核空间。Server 创建了 Binder 实体，为其取一个字符形式，可读易记的名字，将这个 Binder 连同名字以数据包的形式通过 Binder 驱动发送给 ServiceManager。通知 ServiceManager 注册一个名字为 XX 的 Binder，它位于 Server 中。驱动

为 Binder 创建位于内核中的 ServiceManager 对实体的引用。再把 Binder 的名字和引用打包给 ServiceManager。ServiceManager 收到数据包后。将名字和引用填在一张表中。Client 可以通过名字获得 binder 的引用。

Binder 就是一种把这四个组件粘合在一起的粘连剂了，其中，核心组件便是 Binder 驱动程序了，ServiceManager 是一个守护进程，用来管理 Server，并向 Client 提供查询 Server 接口的能力，Client 和 Server 之间的进程间通信通过 Binder 驱动程序间接实现。

Android 系统 Binder 机制中的四个组件 Client、Server、Service Manager 和 Binder 驱动程序的关系如下图所示：



1. Client、Server 和 Service Manager 实现在用户空间中，Binder 驱动程序实现在内核空间中
2. Binder 驱动程序和 Service Manager 在 Android 平台中已经实现，开发者只需要在用户空间实现自己的 Client 和 Server
3. Binder 驱动程序提供设备文件/dev/binder 与用户空间交互，Client、Server 和 Service Manager 通过 open 和 ioctl 文件操作函数与 Binder 驱动程序进行通信
4. Client 和 Server 之间的进程间通信通过 Binder 驱动程序间接实现
5. Service Manager 是一个守护进程，用来管理 Server，并向 Client 提供查询 Server 接口的能力

4、如何保证网络传输数据的安全性 (2017-2-25)

通过网络传输数据，需要保证数据的完整性、保密性，以及能够对数据的发送者进行身份验证。这些都需要通过一些加密算法实现。

(1) 对称加密：

加密和解密使用同一个密钥，特点：保证了数据的保密性。局限性：无法解决密钥交换问题。常用的算法有：

DES,3DES,AES;

(2) 公钥加密：

生成一个密钥对（私钥和公钥），加密时用私钥加密，解密时用公钥解密，特点：解决了密钥交换问题。局限性：对大的数据加密速度慢。

(3) 单向加密：

提取数据的特征码，特点：定长输出，不可逆，可检验数据的完整性。局限性：无法保证数据的保密性。常用算法：MD5、SHA1、CRC-32。

三种加密方法各有优缺点，在实际应用中，数据从发送方到达接收方，通常是这样应用的：

- 1)首先对要发送的数据做单向加密，获取数据的特征码；
- 2)对特征码用发送方的私钥进行加密生成 S1；
- 3)然后对 S1 和数据进行对称加密生成 S2；
- 4)最后将 S2 和对称加密的密码使用接收方的公钥进行加密。

5、自己设计一个 Push 推送服务，需要考虑到那些点 (2017-2-25)

12.1 什么是移动 Push 推送

移动 Push 推送是移动互联网最基础的需求之一，用于满足移动互联环境下消息到达 App 客户端。以转转 (58 赶

集团旗下真实个人的闲置交易平台)为例,当买家下单后,我们通过移动 Push 推送消息告诉卖家,当卖家已经发货时,我们通过移动 Push 消息告诉买家,让买卖双方及时掌握二手商品交易的实时订单动态。

12.2 为什么需要移动 Push 推送?

移动互联网络环境下,经常会出现弱网环境,特别是 2G、3G 等网络环境下,网络不够稳定,App 客户端和相应服务器端的长连接已经断开,消息无法触达 App 客户端。而我们业务需要把 Message (转转 App 交易消息等)、Operation (转转 App 运营活动等)、Alert (转转红包未消费提醒等)等消息推送给 App 客户端,从而触发用户看到这些消息,通过点击这些 Push 消息达到相应目标。

12.3 推送原理和方案对比

移动 Push 推送主要有以下三种实现方式。

12.3.1 移动 App 轮询方式 (PULL)

App 客户端定期发起 Push 消息查询请求,来达到消息推送的目的。PULL 方案的优点和缺点都比较明显,整体架构简单但实时性较差,我们可以通过加快查询频率,提高实时性,但这会造成电量、流量消耗过高。

12.3.2 移动 App 基于短信推送方式 (SMS Push)

通过短信发送推送消息,并在客户端置入短信拦截模块,能拦截短信,并解析后转发给 App 应用处理。这个方案实时性好、到达率高,但成本很高。

12.3.3 移动 App 长连接方式 (Push)

移动 Push 推送基于 TCP 长连接实现,消息实时性好,这是目前主流的实现方式,需要维护 App 客户端和服务端的长连接心跳,会带来额外的电量、流量消耗;在架构设计时,需要做些折中,以避免流量和电量的大量消耗。此外 Push 推送技术架构复杂度较高,维护移动 App 客户端的海量长连接请求,并建立与 App 客户端通信的加密通道,整合成内部少量有限的长连接,对通信数据进行压缩与解压,以节省流量。

目前移动 Push 推送技术基本都是结合这 3 个方案进行,但对于不同的移动终端平台,又有各自不同的实现,这里详细介绍 Android 平台上的具体实现方案。

12.4 Android 平台

在 Android 平台上，由于对 service 常驻没有限制，可用的方案就多一些：可以通过 Google 官方 GCM 完成、开源方案（例如 XMPP）、借助第三方（环信、极光、融云、个推等），或者完全自主研发的移动 Push 推送方案。

移动 Push 推送开源方案

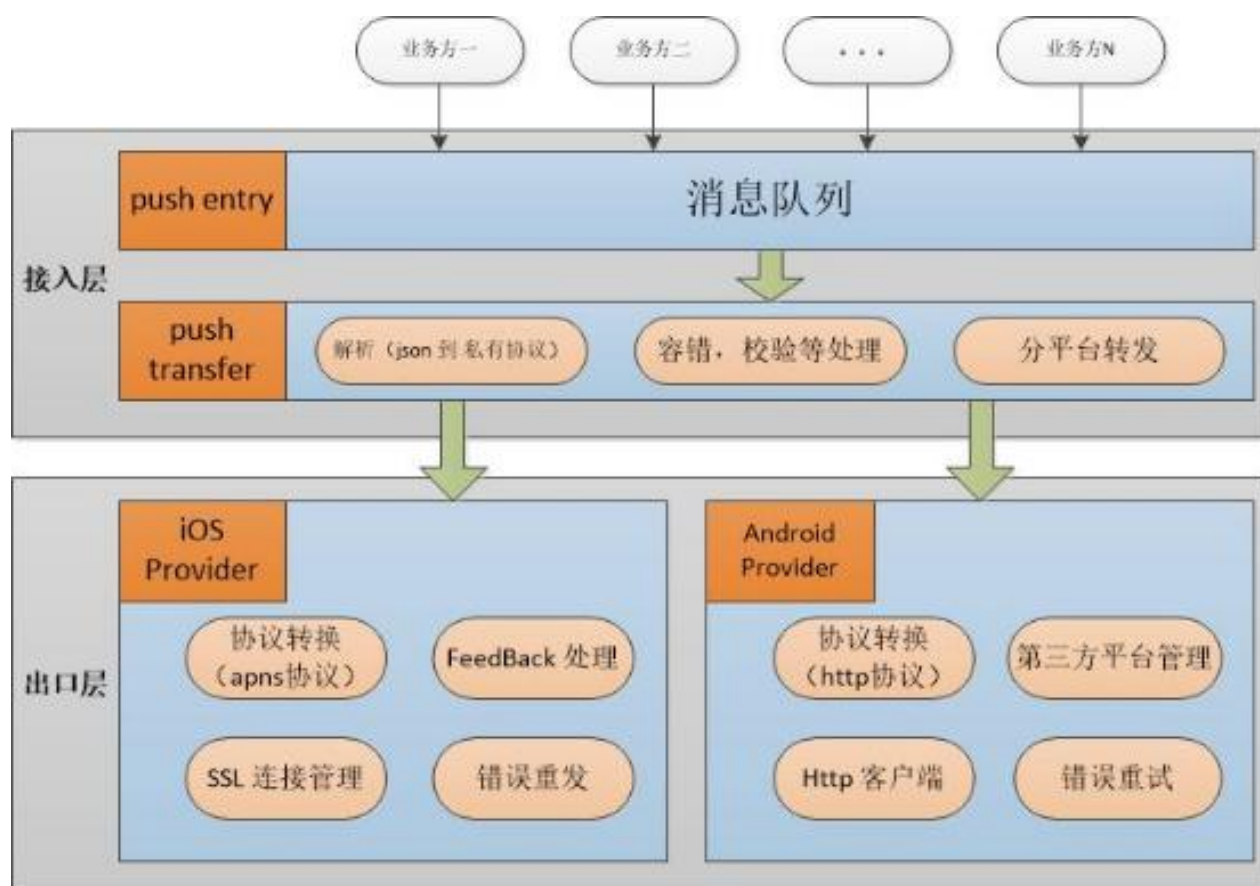
对于开源移动 Push 推送协议，常见的有 XMPP 等，事实上 Google 的 C2DM 底层也是基于 XMPP 协议实现的，我们通过线下测试发现，开源移动 Push 推送方案主要有两个问题：第一，没有 ACK 机制，消息到达没有保证，不可靠；第二，当移动 Push 消息请求量并发增大时，系统开始变得不稳定，甚至出现了模块宕机的情况。因此直接使用移动 Push 推送开源方案，也不是非常可靠。

完全自主研发的移动 Push 推送方案：

需要解决几个难点：第一，移动 Push 推送服务端对移动 App 客户端海量长连接的维护管理。第二，App 客户端常驻 service 稳定性，如何使 Push service 常驻？我们可以借助父子进程互相监控的方式来做到，一旦发现对方进程不在了，会重新建立，继续循环监控。第三，手机内存不足时，系统会杀掉 Push service，甚至有些操作系统比较强势，它会向 iOS 系统一样并不允许第三方 Push service 常驻。第四，移动 Push 推送到达率的提高，除了技术手段外，还有一些 PR 的手段，比如移动 App 客户端 Push service 通过在相应操作系统上添加白名单的方式使其永久常驻。

12.5 58 同城移动 Push 推送方案

58 同城并没有选择从零开始完全自主研发而是采用了基于第三方移动 Push 推送平台和自主研发高性能 Provider 的方案（如下图所示），满足每天百亿量级的吞吐量，并通过动态组合和扩展的方式，结合离线的移动 Push 推送数据分析，不同手机使用不同的推送策略，针对性地优化。在 Android 平台，我们融合多种第三方移动 Push 平台，从而有效提升到达率。



12.6.1 第一阶段（单平台）架构如何设计

背景&需求

58 帮帮，是一款满足 58 用户和商户之间沟通的即时通讯软件，用户间可以互相添加好友、收发消息等。58 帮帮的消息推送基于 App 客户端和服务器的长连接，一旦这条长连接断开，那么 IM 服务端的消息将无法推送给 App 客户端，用户也无法看到这些消息。在 iOS 平台上，58 帮帮 App 切换到后台后，App 与 IM 的长连接断开，消息无法触达，这时候我们需要借助 iOS APNS 机制，IM 消息需要发送给 APNS，APNS 再转发对应的消息到 58 帮帮 App。Android 切换至后台，App 与 IM 的长连接保持，IM 消息可以正常推送，因此在这个阶段我们需要解决的问题是在 iOS 平台上，当 58 帮帮 App 切后台后，IM 在长连接断开后的消息触达需求。

设计目标

基于上述的背景和需求，我们在设计移动 Push 推送第一阶段（单平台）架构时，首先要满足在 iOS 平台上，当 IM 长连接断开后，IM 消息的能够触达到 App 客户端。其次我们的移动 Push 推送协议设计也具备很好的扩展性，在可以预见的未来，Push 推送平台将逐步接入更多的 App，因此我们设计目标 iOSProvider 是一个通用的 iOS 推送服务。不同 App 通过使用不同的移动 Push 推送证书借助同一 iOSProvider 完成移动 Push 消息推送，对于不同 App 的接入，我们采用了配置文件方式动态扩展接入，iOSProvider 根据所配置 App 证书与 APNS 建立并维护多条 TLS 连接。配置文件的格式如下：

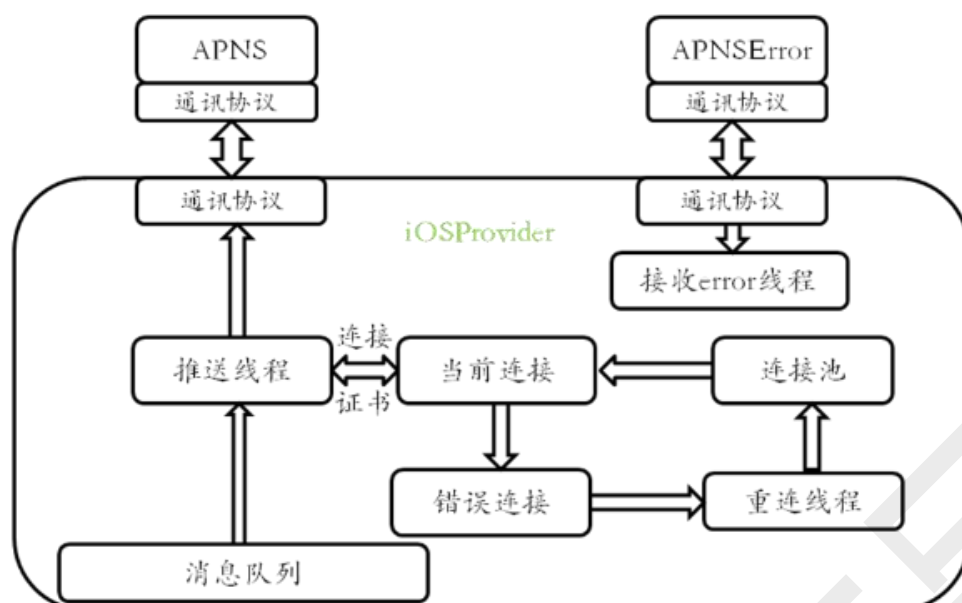
```
第一个域#第二个域#第三个域#第四个域
```

其中，第一个域为推送服务类型 Type，以备扩展，1 为 APNS；第二个域为内部定义的 APPID 号，对应服务的 App；第三个域为 App 的 Apple 证书文件名；第四个域为与 APNS 建立的连接数；

每个 App 接入的配置为一行，举例如下：

```
1.1 #88 #zhuanzhuan.p12 #64
2.1 #66 #58tongcheng.p12 #32
```

除此之外，iOSProvider 需要对每个接入 App 的 APNS 连接池进行管理，动态增删 TLS 连接，具备动态重连机制，并具有单独的反馈接收线程，用于异步接收 APNS 返回无效的 Token，反馈给移动 Push 推送业务方，用于下次移动 Push 消息推送的优化。iOSProvider 根据 Type、APPID 选择对应的 APNS 连接，通过推送线程组装 APNS 包发送到 APNS 服务器，如图 4 所示。



12.6.2 第二阶段（多平台）：架构如何设计优化

目前还不具备 Android 移动 Push 推送的能力.综合目前可选择的方案，我们选择了基于第三方推送平台以及自主研发高性能 AndroidProvider 的方案。

首先重点讲述针对 Android 移动 Push 推送的流程：第一，App 客户端向第三方移动 Push 推送平台注册，获取对应的 App 唯一标示（Token）。第二，App 将 Token 信息发送给 AndroidProvider 并集中存储，以便后续基于 Token 的移动 Push 推送。第三，AndroidProvider 通过 HTTPS 或者 TSL 的方式和第三方移动 Push 推送平台建立连接，并把需要推送的消息发送到第三方移动 Push 推送平台。第四，第三方移动 Push 推送平台收到 AndroidProver 推送的消息后，会把此消息及时推送到 App，从而完成整个推送过程，如图 5 所示。

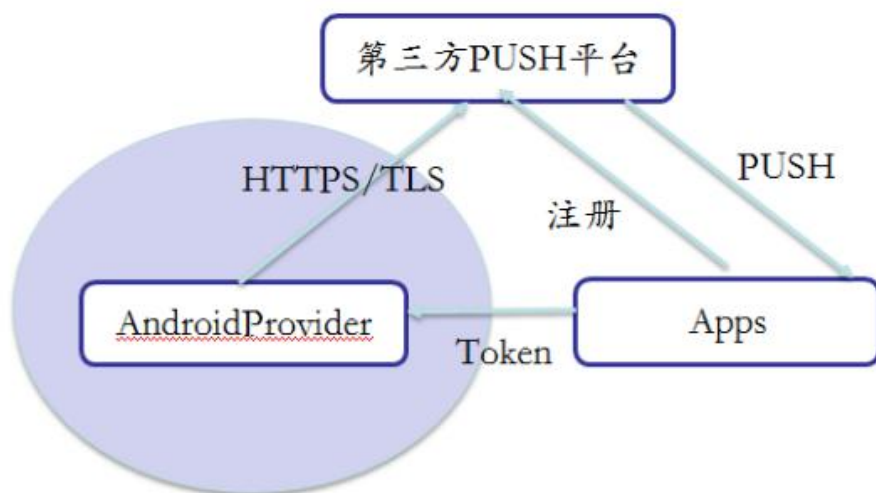


图 5 Android 移动 PUSH 推送流程

AndroidProvider 子系统整体结构分为四个层次，第一层为业务方移动 Push 推送接入，用于众多移动 Push 推送业务方的接入。

第二层为网络交互层，用于接收移动 Push 推送业务方的消息数据以及发送请求处理层的处理数据给业务推动调用。第三层为请求处理层，用于处理网络交互层放入请求队列的数据，组装成第三方移动 Push 推送接口需要的数据，通过 HTTP 或者 HTTPS 的方式调用下游的接口，并等待请求结果的返回，把请求返回的结果放入回应队列。第四层为第三方移动 Push 推送平台，由第三方提供，开放给使用方接口，供调用其功能，如图 6 所示。

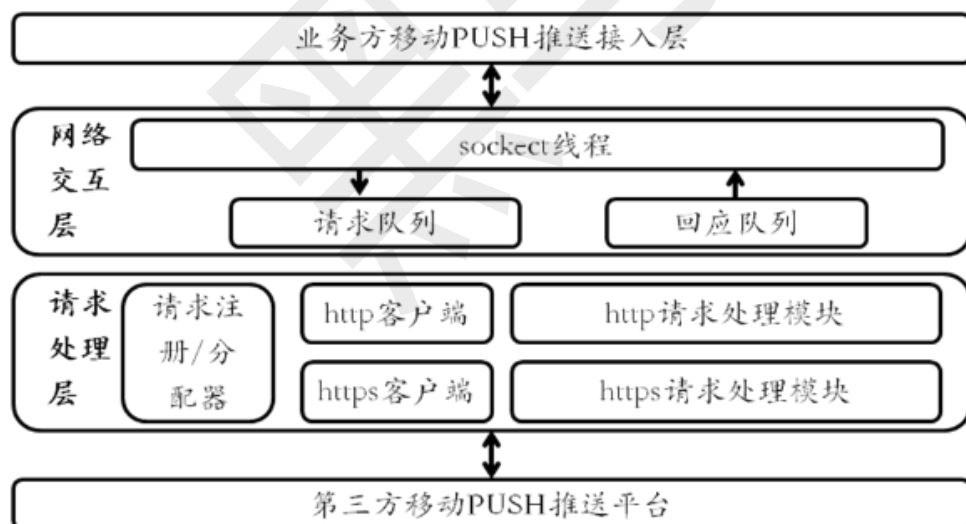


图 6 AndroidProiver 系统架构图

随着越来越多的移动 App 接入，移动 Push 推送需求趋向多样化，同时移动 Push 推送业务逻辑复杂化（多终端、批量发送、业务规则多样），公共策略每个业务方重复开发（深夜防打扰功能、发送频率和发送速率的限制等），造成开发效率低下。为了解决这些问题，我们抽象了公共的逻辑，并进行了统一的封装，对业务调用方透明，这些公共的逻辑包括：通用的策略和通用的控制，如图 7 所示。

图 7 所示。

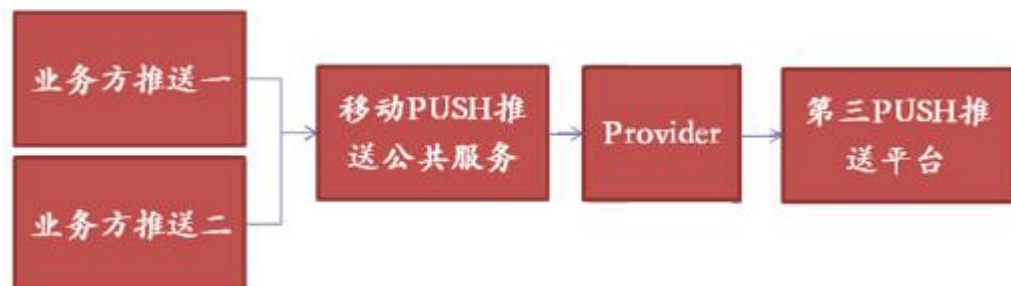


图 7 Android 移动 PUSH 推送演进业务架构

12.6.3 第三阶段架构和协议如何设计和优化

移动 Push 推送平台第三阶段我们如何架构和设计？首先我们满足对下游接入方多种连接的管理（HTTP、HTTPS、TCP、SSL、TSL），具备了多种连接动态伸缩性，从而满足 Provider 层对移动 Push 推送连接的要求。其次平台要具备高并发的特性，通过完全异步的设计和多线程支持，做到了高并发和支持 10 万 QPS 吞吐量。再次我们需要对接入下游的错误进行处理，一旦发现连接被断开等错误后，要能够自动使用新的连接，并且对已经发出还没到达 App 客户端的推送消息进行重发，以保证消息不丢失。第四我们需要对通道进行封装，对外提供统一的友好接入接口，屏蔽底层 iOS 和 Android 接入的差异性。最后在 Android 移动推送方面，我们接入了更多的第三方推送平台，以达到更高的到达率。

基于这些方面的考虑，58 同城移动 Push 推送平台采用了低耦合的分层架构设计（如图 3 所示），分为三层 Push Entry、Push Transfer、Provider（iOSProvider 和 AndroidProvider）。其中 Push Entry 是业务方调用的入口，我们采用异步消息队列的方式，提供了较高的业务方发送的速度，并且具备了消息缓冲的功能，使得高峰期的海量移动 Push 消息推送对整个平台冲击较少，也起到了保护推送系统的作用。Push Transfer 会从 Push Entry 层接收消息进行解析，对推送消息进行合法性检查，如果格式不合法，直接丢弃，同时会进行接收到的推送消息格式转换成内部的消息格式，分平台转发到 iOSProvider 或者 AndroidProvider

上；provider 接收到 Push Transfer 的消息后，会按照下游需要的消息格式（APNS 协议、Android 协议）进行转换，进行消息的下发，在下发的过程中，会进行消息的重发，以确保消息下发到第三方推送平台。

Provider 模块内部如何设计？以 iOSProvider 为例，它分为三个层次：接入逻辑、业务逻辑、APNS 出口。其中接入逻辑主要处理网络交互和请求分发；业务逻辑主要处理线程分裂扩展、并发处理和错误处理；APNS 出口处理向 APNS 的发送逻辑，如图 8。

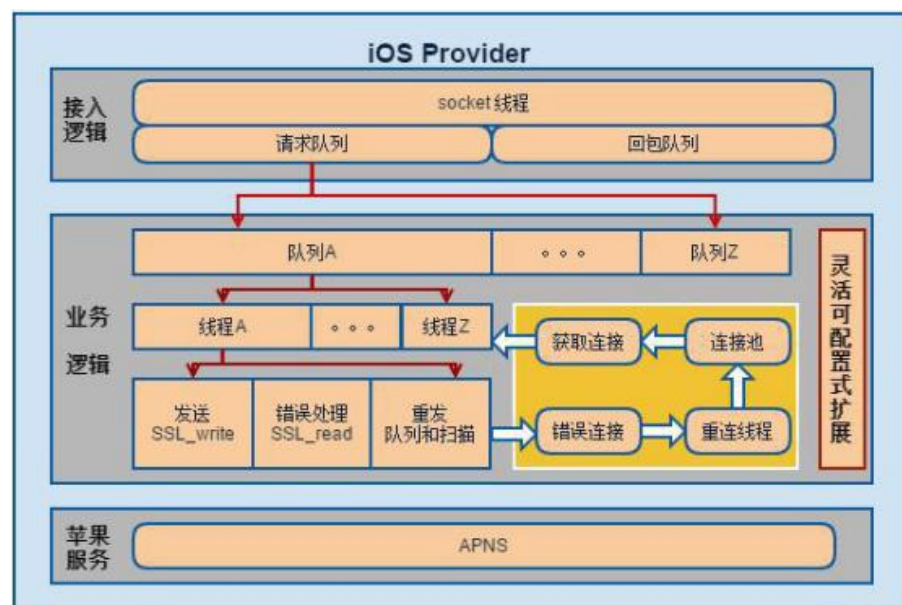


图 8 iOSProvider 模块结构图

对于移动 Push 推送平台来说，追求达到率是我们最核心的指标，没有之一。因此在 Android 方面，我们融合了多个第三方推送平台，通过机型控制，对不同的机型使用不同通道，进一步提升推送到达率。AndroidProvider 层进行消息推送策略的控制，先推送一通道，根据此推送通道 ACK 情况，是否继续推送其他通道。推送多个 Push 通道，会出现推送消息重复到达 App 客户端的情形，此时需要 App 客户端根据推送消息 ID 进行去重，收到的重复推送消息忽略处理。

典型性能问题分析解决以及高性能，高可用，高稳定性如何保证

在移动 Push 推送不断演进的过程中，我们遇到了 AndroidProvider 并发低的问题，仔细分析，是因为我们采用 HTTPS 库，由于库中 HTTPS 的连接实现不是线程安全的，对每个 HTTPS 的请求都加锁串行化处理，以保证线程的安全性。发现问题后，我们通过在增加多进程部署的方式暂时解决，使得我们有足够的时间分析此问题产生的根本原因。经过深入分析，发现原因是我们对

HTTPS 的库掌握不够，导致加锁粒度过大，通过 HTTPS 库提供的更小粒度的锁，我们不仅解决了线程不安全的问题，也提升了 AndroidProvider 的并发度，如图 9 所示。

```
23 static void lock_callback(int mode, int type, const char *file, int line)
24 {
25     if(mode & CRYPTO_LOCK)
26     {
27         pthread_mutex_lock( &(lockArray[type]));
28     }
29     else
30     {
31         pthread_mutex_unlock( &(lockArray[type]));
32     }
33 }
```

图 9 HTTPS 库细粒度锁实现方式

总之，58 同城统一的高性能移动 Push 推送平台通过无状态化设计和冗余部署等方式确保了推送平台的高可用，通过纯异步、动态多线程的支持提供推送平台的高性能，通过质量保证、多种监控机制（进程监控、语义监控、错误日志监控、数据波动监控等），有问题及时发现处理保证了推送平台的高稳定性。

6. Android 项目 (★★★★)

1、如何让 LinearLayout 自动换行如下图的颜色分类所示。



颜色分类:

米黄色 粉红色 浅灰色 黑色

浅蓝色

尺码:

均码

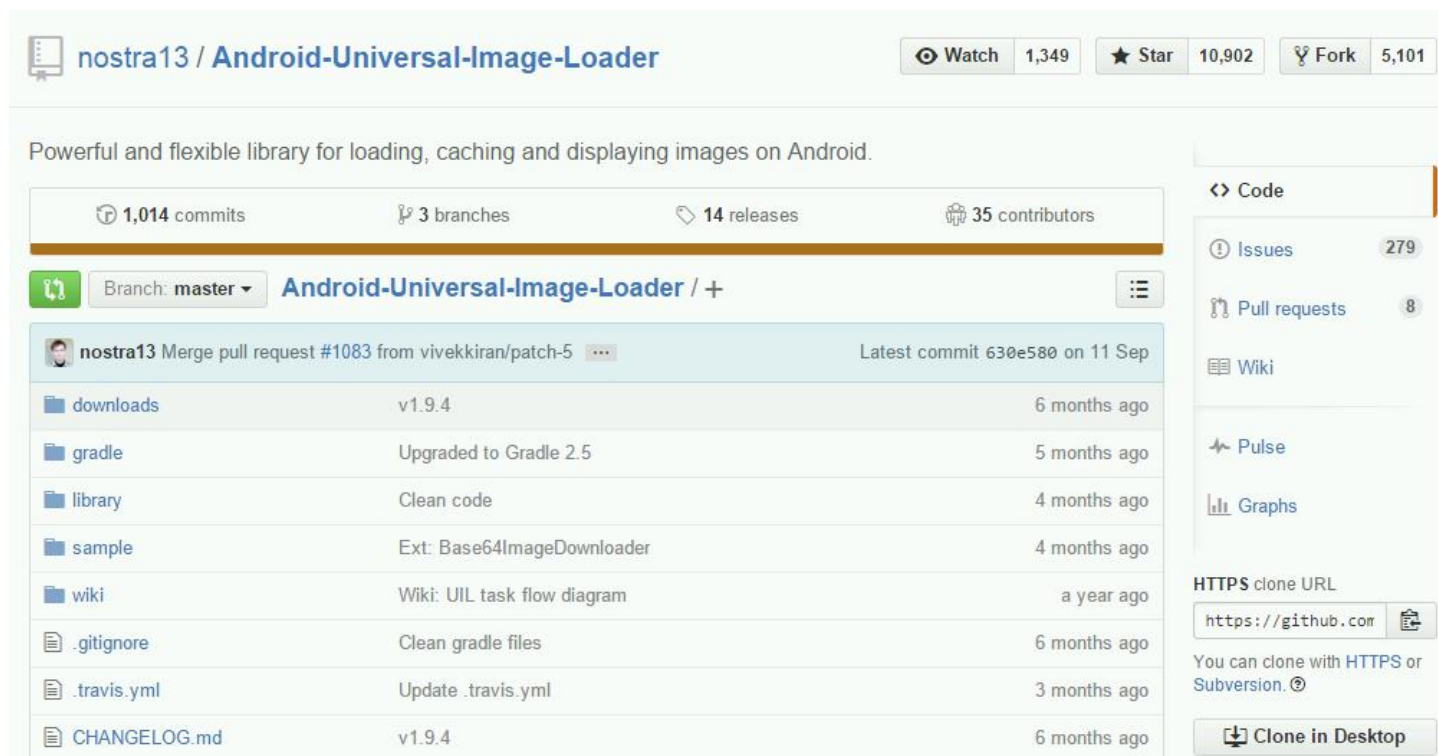
数量:

- 1 +

<http://yanmingming.sinaapp.com/?p=1205#more-1205>

2、ImageLoader 在项目中的使用

几乎所有的 App 都有图片的展示，这些图片或来自网络或来自本地存储，在使用图片的时候，我们不仅要考虑避免内存的 OOM，又要考虑加载的效率。ImageLoader 是一个比较好用的开源框架，代码托管在 github 上，地址如下：<https://github.com/nostra13/Android-Universal-Image-Loader>



nostra13 / **Android-Universal-Image-Loader**

Powerful and flexible library for loading, caching and displaying images on Android.

1,014 commits 3 branches 14 releases 35 contributors

Branch: master **Android-Universal-Image-Loader** / +

nostra13 Merge pull request #1083 from vivekkiran/patch-5 Latest commit 630e580 on 11 Sep

File	Commit Message	Time
downloads	v1.9.4	6 months ago
gradle	Upgraded to Gradle 2.5	5 months ago
library	Clean code	4 months ago
sample	Ext: Base64ImageDownloader	4 months ago
wiki	Wiki: UIL task flow diagram	a year ago
.gitignore	Clean gradle files	6 months ago
.travis.yml	Update .travis.yml	3 months ago
CHANGELOG.md	v1.9.4	6 months ago

HTTPS clone URL
https://github.com

You can clone with HTTPS or Subversion. ®

Clone in Desktop

Android 开源框架 Universal-Image-Loader 完全解析 (一) --- 基本介绍及使用

<http://blog.csdn.net/xiaanming/article/details/26810303>

Android 开源框架 Universal-Image-Loader 完全解析 (二) --- 图片缓存策略详解

<http://blog.csdn.net/xiaanming/article/details/27525741>

Android 开源框架 Universal-Image-Loader 完全解析 (三) --- 源代码解读

<http://blog.csdn.net/xiaanming/article/details/39057201>

3、Java 和 JavaScript 互相调用 (webview 和 js 的互相调用)

目前越来越多的 Android App 都采用了混合编程的架构，也就是 Android 原生控件+html5，当然 html 页面也必须运行在 webview 控件中，我们可以简单吧 webview 看成内置到 app 中的小型浏览器，这里面最核心的问题就是 html 中的 js 代码如何调用 java 代码，比如 js 调用 java 代码，然后打开一个新的 Activity 界面。比如淘宝已经大量采用了 h5 来实现其客户端功能，商品列表是 Android 端控件，点击商品列表进入某个商品详情可能就是 h5 界面，这里

就存在 java 给 js 传递数据的问题，点击 h5 的商品详情界面的立即购买，可能就又从 h5 跳转到了 Activity 界面，这里又牵扯到 js 给 java 传递数值的问题。

看了下面的两篇博客，然后在写一个 demo 加深印象之后，相信大家会对 js 和 java 直接的纠缠搞的更明白。

android webview js 交互 第一节 （java 和 js 交互）

<http://blog.csdn.net/wangtingshuai/article/details/8631835>

android webview js 交互， 响应 webview 中的图片点击事件

<http://blog.csdn.net/wangtingshuai/article/details/8635787>

4、PopupWindow 弹出层在项目中的使用



在绝大多数 app 中都用到了 popupWindow，因此掌握 popupWindow 是 Android 程序员必备技能之一。

上图中 popupWindow 的实现博客如下：

<http://www.linuxidc.com/Linux/2012-05/61519.htm>

<http://www.linuxidc.com/Linux/2012-05/61519p2.htm>

<http://www.linuxidc.com/Linux/2012-05/61519p3.htm>

5、Notification 在 Android 中的使用

几乎所有的 app 都会使用状态的通知，如下图所示。



如果你现在还不知道 Notification 如何使用，那么请看完下面这篇博客，看完之后再建议你写一个自己的 demo，这样到公司工作的时候代码也可以直接拿过来使用。

【Android】状态栏通知 Notification、NotificationManager 详解

<http://blog.csdn.net/feng88724/article/details/6259071>

6、带索引的 ListView 在 Android 中的应用



如上图所示，在很多应用中都有的效果。上图实现如下博客地址：

<http://blog.csdn.net/xiaanming/article/details/12684155>

7、随手势滑动而消失 Activity 的使用

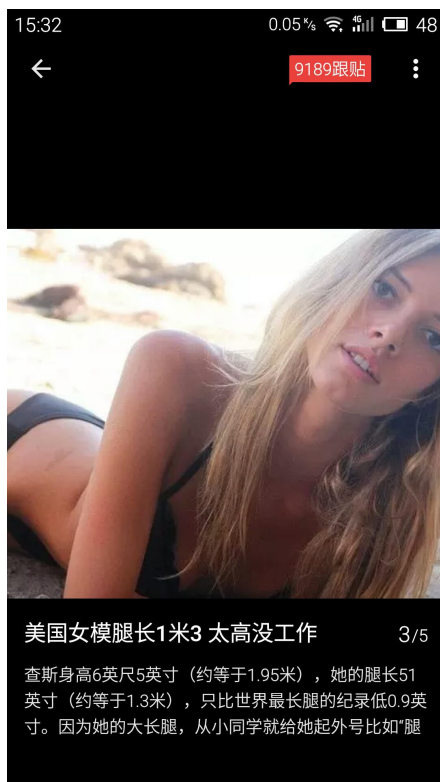
在部分 app 中，Activity 的销毁可以随着手势的滑动而销毁，给人一种很炫酷的感觉，其实现原理如下博客所示：

<http://blog.csdn.net/xiaanming/article/details/20934541>

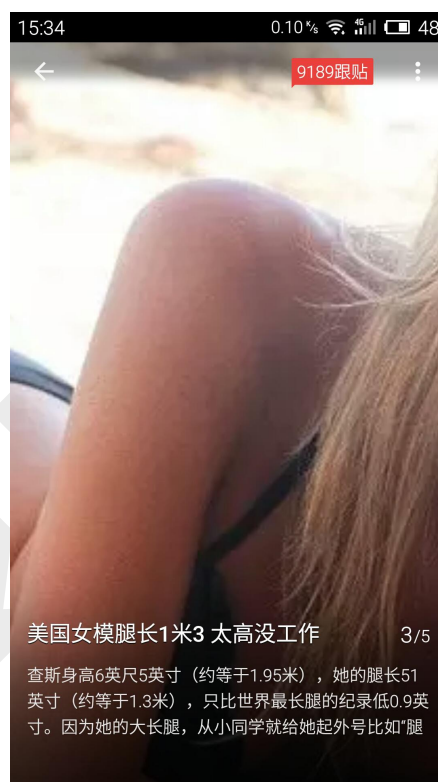


向右滑动消失。

8、TouchGallery 在 Android 中的应用



用两个手向外滑动后的效果



在很多应用中都使用了可以手势识别的画廊，比如网易新闻以及其他所有的新闻客户端。这里使用到最核心的代码托管在 github 上。地址如下：

<https://github.com/Dreddik/AndroidTouchGallery>

9、TextView 显示富文本

部分 app 使用 TextView 显示出来很绚丽的效果，这是利用了 TextView 对 Html 的支持做到的。只不过 TextView 支持的 Html 是有限的。

下面博客实现了如下功能：

实现 TextView 里的文字有不同颜色

TextView 显示 html 文件中的图片

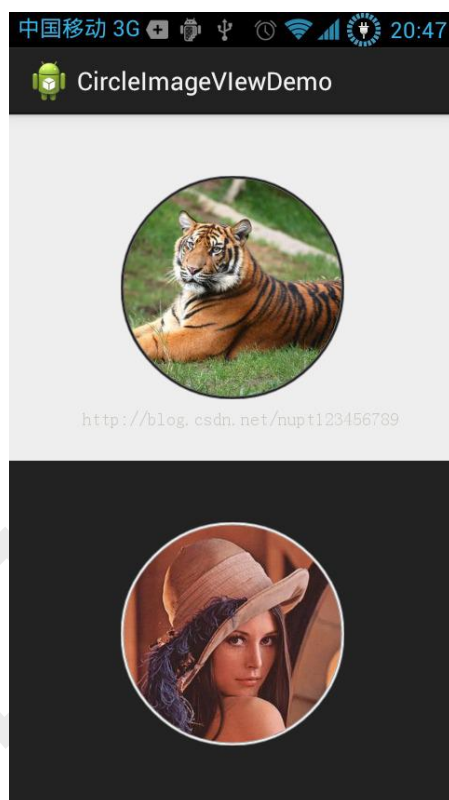
Android---文字中插入表情

android 短信字符转表情显示过程

Android TextView 支持的 HTML 标签

<http://www.cnblogs.com/playing/archive/2011/03/17/1987033.html>

10、CircleImageView 实现圆形图片



如上图所示，很多 app 在用户登录后都会使用圆形图片展示用户的头像。其实现原理如下博客所示：

<http://blog.csdn.net/nupt123456789/article/details/24886451>

11、网易新闻客户端频道管理的实现 (2015-11-25)



如上图所示，很多新闻资讯类客户端都用这种功能。其实现思路：

实现思路：

1. 获取数据库中频道的列表，如果为空，赋予默认列表，并存入数据库，之后通过对应的适配器赋给对应的 GridView
2. 2 个 GridView-- (1.DragGrid 2. OtherGridView)

DragGrid 用于显示我的频道，带有长按拖拽效果

OtherGridView 用于显示更多频道，不带推拽效果

注：由于屏幕大小不一定，外层使用 ScrollView，所以 2 者都要重写计算高度

3. 点击 2 个 GridView 的时候，根据点击的 Item 对应的 position，获取 position 对应的 view，进行创建一层移动

的动画层

起始位置 : 点击的 `position` `getLocationInWindow()` 获取。终点位置 : 另一个 `GridView` 的最后一个 `ITEM` 的 `position + 1` 的位置。

并赋予移动动画，等动画结束后对 2 者对应的频道列表进行数据的 `remove` 和 `add` 操作。

4. 设置点击和拖动的限制条件，如 推荐 这个 `ITEM` 是不允许用户操作的。

5. 拖动的 `DragGrid` 的操作：

(1) 长按获取长按的 `ITEM` 的 `position` -- `dragPosition` 以及对应的 `view`，手指触摸屏幕的时候，调用 `onInterceptTouchEvent` 来获取 `MotionEvent.ACTION_DOWN` 事件，获取对应的数据。由于这里是继承了 `GridView`，所以长按时间可以通过 `setOnItemLongClickListener` 监听来执行，或则你也可以通过计算点击时间来监听是否长按。

(2) 通过 `onTouchEvent(MotionEvent ev)` 来监听手指的移动和抬起动作。当它移动到 其它的 `item` 下面，并且下方的 `item` 对应的 `position` 不等于 `dragPosition`，进行数据交换，并且 2 者之间的所有 `item` 进行移动动画，动画结束后，数据更替刷新界面。

(3) 抬起手后，清除掉拖动时候创建的 `view`，让 `GridView` 中的数据显示。

6. 退出时候，将改变后的频道列表存入数据库。

其实现原理如下：

<http://www.javaapk.com/source/4529.html>

本人网盘下载地址：<http://pan.baidu.com/s/1dD3Bo6t>

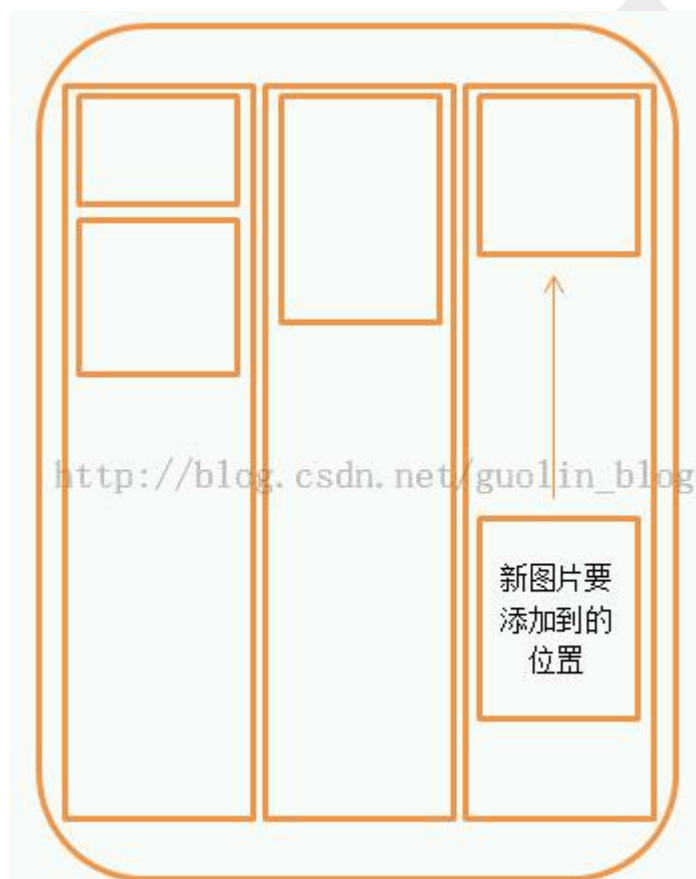
12、Android 瀑布流的实现

如下图所示，不规则也是一种美。下面的瀑布流在很多 `Android` 应用中都有使用。

传统界面的布局方式总是行列分明、坐落有序的，这种布局已是司空见惯，在不知不觉中大家都已经对它产生了审美疲劳。这个时候瀑布流布局的出现，就给人带来了耳目一新的感觉，这种布局虽然看上去貌似毫无规律，但是却有一种说不上来的美感，以至于涌现出了大批的网站和应用纷纷使用这种新颖的布局来设计界面。

实现原理：

瀑布流的布局方式虽然看起来好像排列的很随意，其实它是有很科学的排列规则的。整个界面会根据屏幕的宽度划分成等宽的若干列，由于手机的屏幕不是很大，这里我们就分成三列。每当需要添加一张图片时，会将这张图片的宽度压缩成和列一样宽，再按照同样的压缩比例对图片的高度进行压缩，然后在这三列中找出当前高度最小的一列，将图片添加到这一列中。之后每当需要添加一张新图片时，都去重复上面的操作，就会形成瀑布流格局的照片墙，示意图如下所示。



听我这么说完后，你可能会觉得瀑布流的布局非常简单嘛，只需要使用三个 `LinearLayout` 平分整个屏幕宽度，然后动态地 `addView()` 进去就好了。确实如此，如果只是为了实现功能的话，就是这么简单。可是别忘了，我们是在手

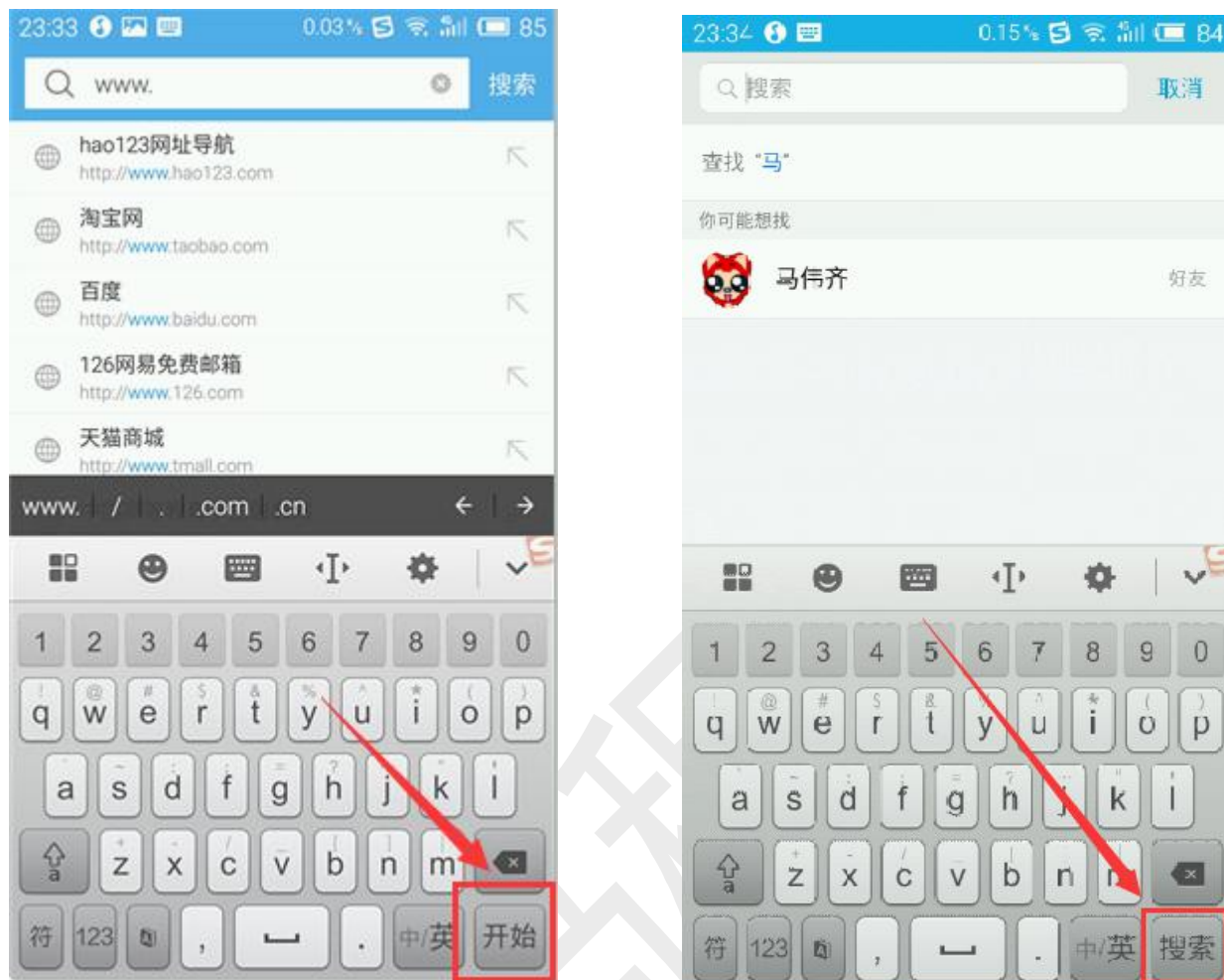
机上进行开发，如果不停地往 LinearLayout 里添加图片，程序很快就会 OOM。因此我们还需要一个合理的方案来对图片资源进行释放，这里仍然是准备使用 LruCache 算法。



其实现博客如下所示：

http://blog.csdn.net/guolin_blog/article/details/10470797

13、监听键盘事件 (2015-11-29)



如上图所示一个是 UC 浏览器，一个是 QQ 的界面，如何动态修改输入法键盘红色框中（右下角）的文字，以及如何监听右下角按钮的点击事件？

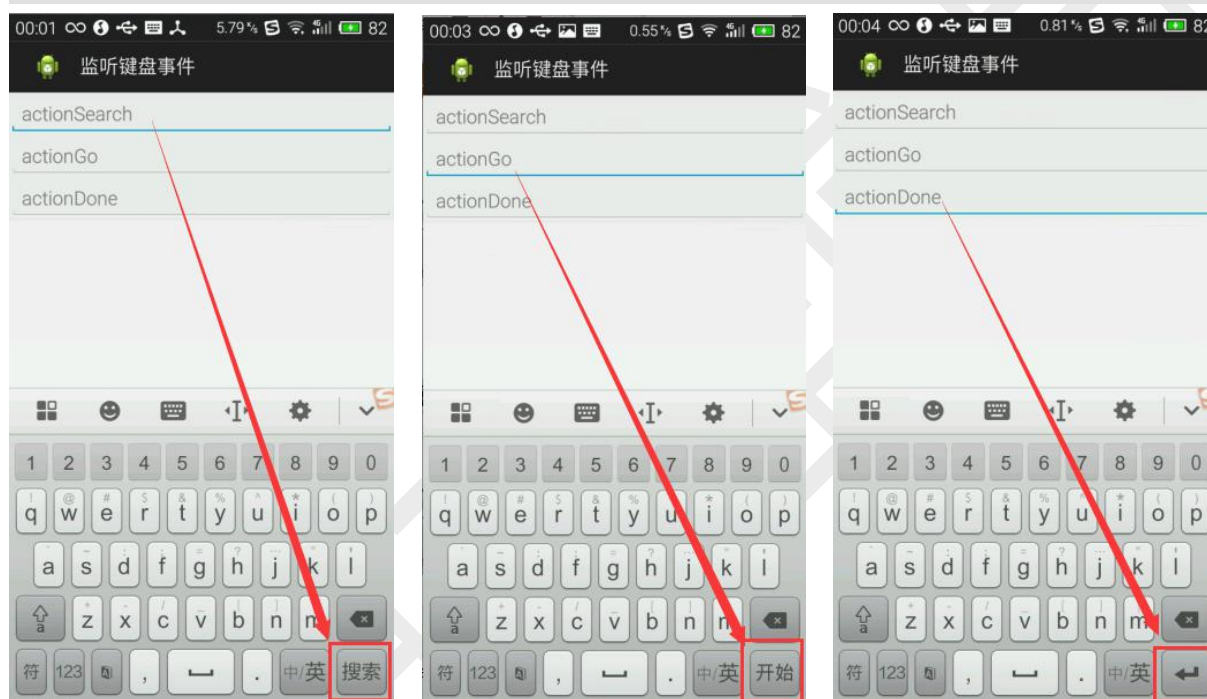
上面其实是两个问题：

1、如果修改右下角的文字

EditText 有 android:imeOptions 属性，只有修改下面的属性即可。比如给出实例代码如下：

```
5. <EditText
6.     android:layout_width="match_parent"
7.     android:layout_height="wrap_content"
8.     android:hint="actionSearch"
9.     android:inputType="text"
10.    android:imeOptions="actionSearch"
11. />
12. </EditText
```

```
13.         android:layout_width="match_parent"
14.         android:layout_height="wrap_content"
15.         android:hint="actionGo"
16.         android:inputType="text"
17.         android:imeOptions="actionGo"
18.     />
19.     <EditText
20.         android:hint="actionDone"
21.         android:inputType="text"
22.         android:layout_width="match_parent"
23.         android:layout_height="wrap_content"
24.         android:imeOptions="actionDone"
25.     />
```



上面三种属性值，对应的效果如上图所示。

但是仅仅是显示不同的样式而已，还没有响应点击事件。接下来就需要通过代码监听红色框中按钮的点击事件。

2、如何监听右下角按钮的点击事件

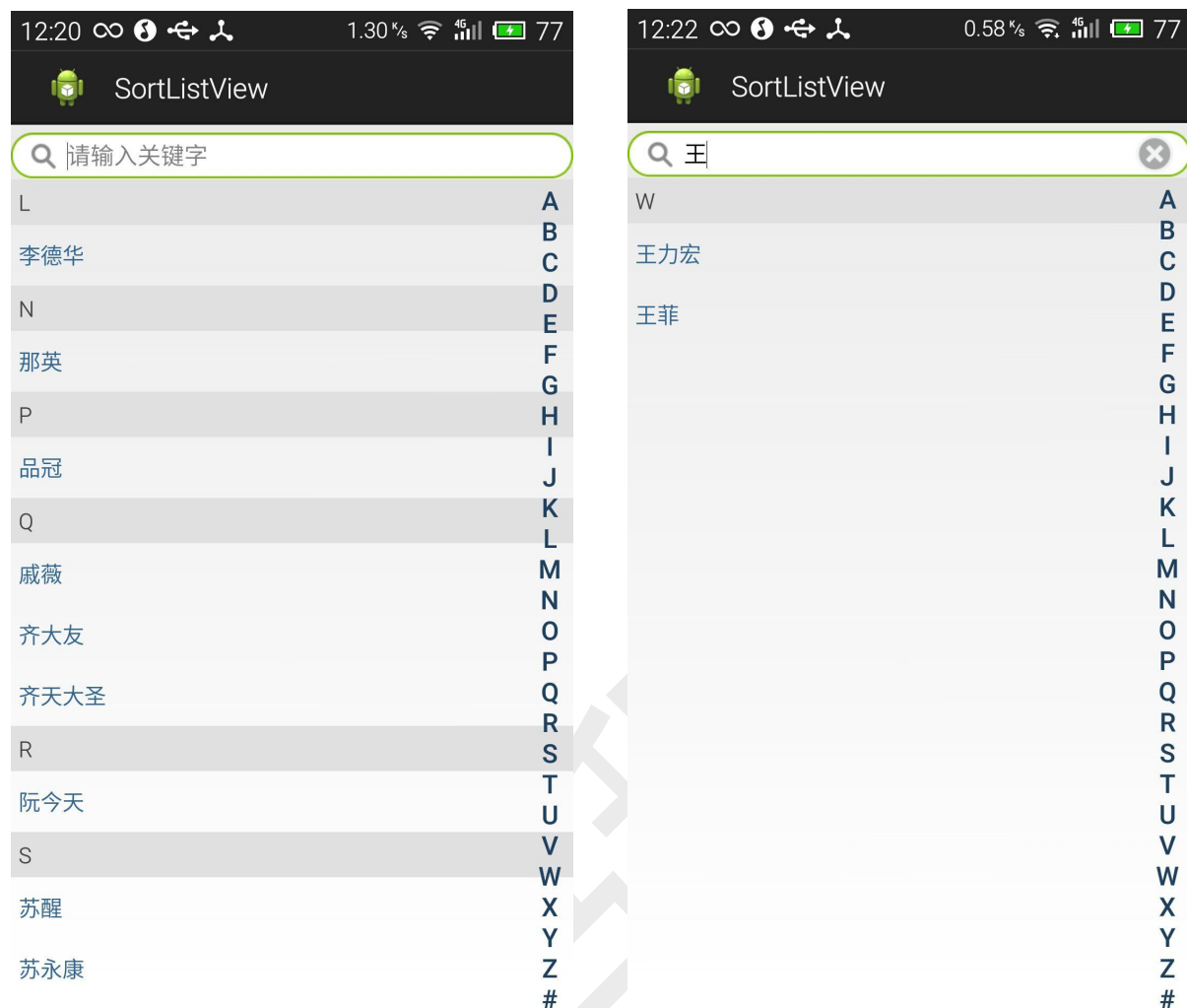
如下代码所示，只需要给 EditText 对象设置一个 `setOnEditorActionListener` 监听器即可。注意：该监听器只能监听上图中右下角红框中的按键动作。

```
1. /**
2.  * 监听输入法键盘的事件
3.  *
4.  * @author wzy 2015-11-28
```



```
5.  *
6.  */
7.  public class MainActivity extends Activity {
8.
9.      private EditText et_search;
10.
11.     @Override
12.     protected void onCreate(Bundle savedInstanceState) {
13.         super.onCreate(savedInstanceState);
14.         setContentView(R.layout.activity_main);
15.         et_search = (EditText) findViewById(R.id.et_search);
16.         //给 EditText 设置按键（注意：只能监听右下角按键）的监听
17.         et_search.setOnEditorActionListener(new OnEditorActionListener() {
18.
19.             @Override
20.             public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
21.                 Toast.makeText(MainActivity.this, "点击按键了", Toast.LENGTH_SHORT).show();
22.                 return true;
23.             }
24.         });
25.     }
26. }
```

14、可以按照字母排序的 ListView (2015-11-29)



如上图所示，很多应用都有的效果。代码比较多，为了方便大家学习，我把其 eclipse 工程上传到我的百度网盘上供大家下载。

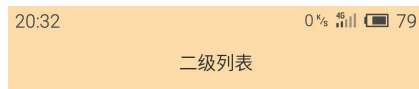
地址：<http://pan.baidu.com/s/1kTEout5>

15、省市区三级联动



地址：<http://pan.baidu.com/s/1eQ39oCq>

16、购物客户端二级菜单



地址：<http://pan.baidu.com/s/1qWaa1NQ>

17、二维码扫描

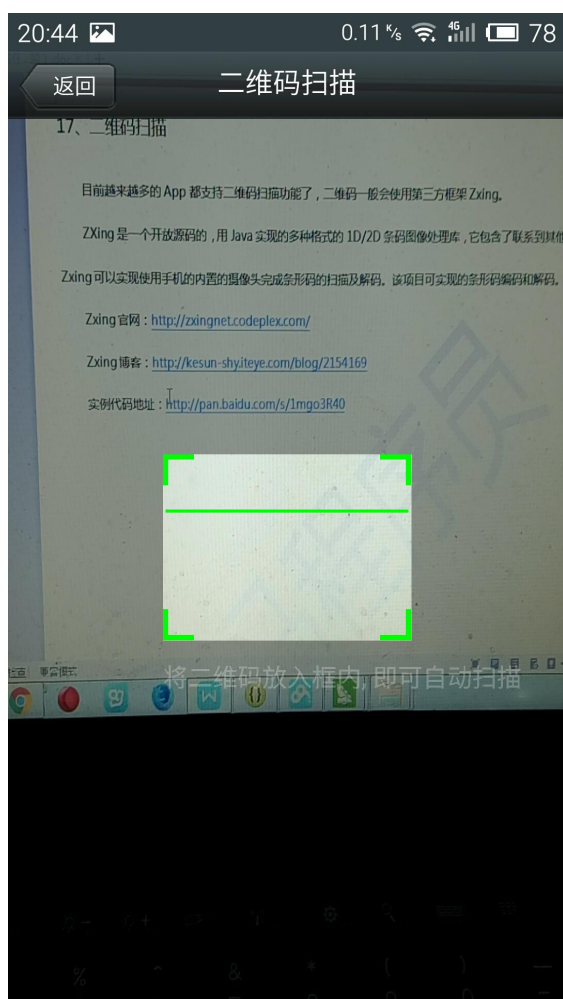
目前越来越多的 App 都支持二维码扫描功能了，二维码一般会使用第三方框架 Zxing。

ZXing 是一个开放源码的，用 Java 实现的多种格式的 1D/2D 条码图像处理库，它包含了联系到其他语言的端口。

Zxing 可以实现使用手机的内置的摄像头完成条形码的扫描及解码。该项目可实现的条形码编码和解码。

Zxing 官网：<http://zxingnet.codeplex.com/>

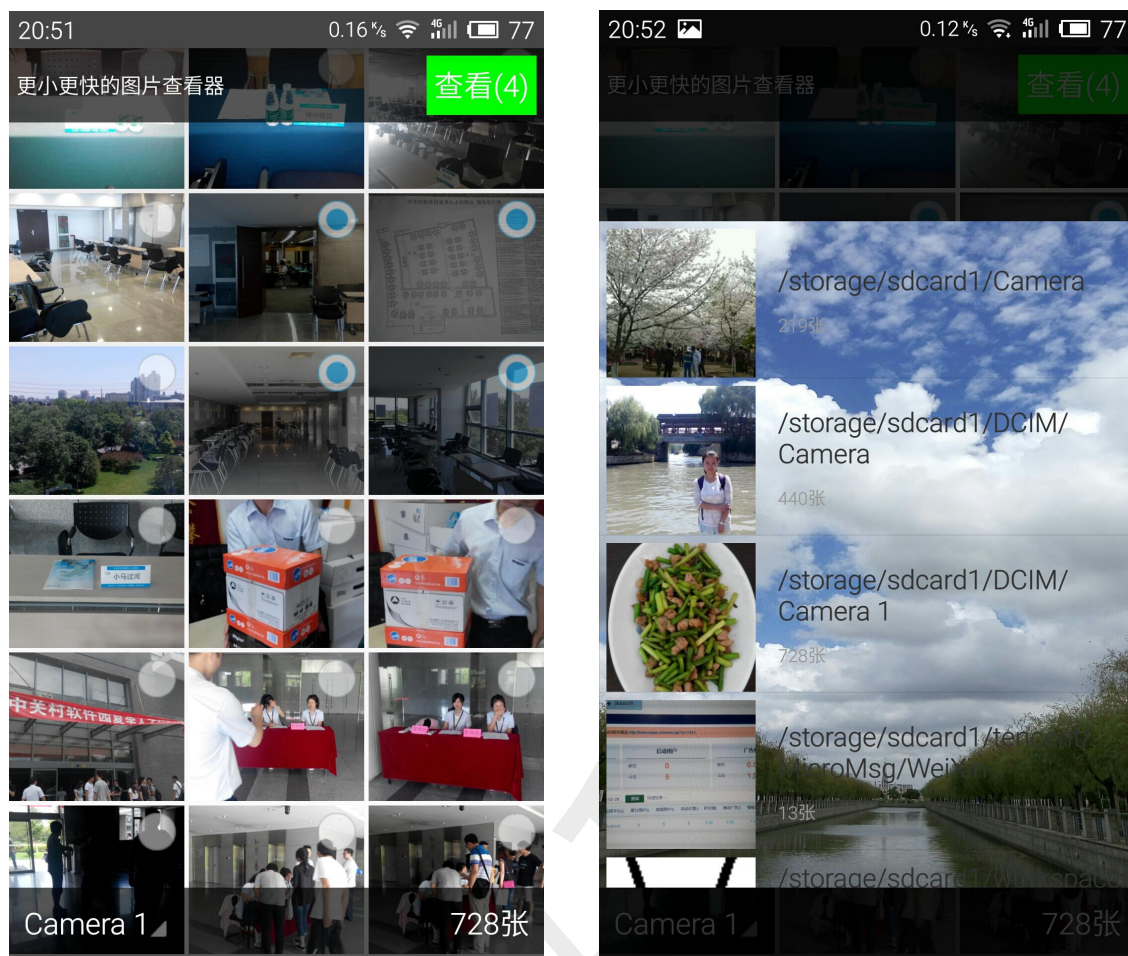
Zxing 博客：<http://kesun-shy.iteye.com/blog/2154169>



实例代码地址：<http://pan.baidu.com/s/1mgo3R40>

18、微信图片选择器

微信图片选择器项目在慕课网上有非常好视频，地址：<http://www.imooc.com/learn/489>。



如上图所示是自己做的仿微信图片选择器，主界面是一个 GridView，展示手机中某个文件夹的图片，点击最下侧后会从底部弹出一个 popwindow，popwindow 里面是一个 ListView 列出所有图片的路径。选中某一个条目后，会回到主界面展示选中文件夹中所有的图片，选择图片后，图片会变暗，然后点击查看会跳转到一个画廊 Activity，该 Activity 支持手势识别，可以放大缩小图片查看。

源码地址：<http://pan.baidu.com/s/1jGvt5aA>

19、如何从 html 的一个动作打开 app 并跳转到指定的 Activity(2017-2-25)

就 Android 平台而言，URI 主要分三个部分：scheme, authority and path。其中 authority 又分为 host 和 port。

格式如下：

scheme://host:port/path

举个实际的例子：

content://com.example.project:200/folder/subfolder/etc

```

\-----/ \-----/ \---/ \-----/
scheme          host          port          path
          \-----/
              authority
  
```

写一个简单的网页

```
1.<a href="znn://aa.bb:80/test?p=12&d=1">test</a> //html 中的代码
```

AndroidManifest.xml

```

<intent-filter> //activity 中 intent-filter 的配置
2. <action android:name="android.intent.action.VIEW"/>
3.     <category android:name="android.intent.category.DEFAULT"/>
4.     <category android:name="android.intent.category.BROWSABLE"/>
5.     <data android:scheme="znn"/> //要和标签中一致
6. </intent-filter>
  
```

Activity 中的代码

```

1.Intent intent = getIntent();
2.String scheme = intent.getScheme();//得到传过来的 scheme
3.Uri uri = intent.getData();
4.System.out.println("scheme:"+scheme);//scheme:znn
5.if (uri != null) {
6.    //获取 host 的值
7.    String host = uri.getHost();
8.    System.out.println("host:"+host); //host:aa.bb
9.    //获取 host
10.   String dataString = intent.getDataString();
11.   System.out.println("dataString:"+dataString); //dataString:znn://aa.bb:80/test?p=12&d=1
12.   //获取参数值 d
13.   String id = uri.getQueryParameter("d");
14.   System.out.println("id:"+id); //id:1
15.   //获取 path 值:
16.   String path = uri.getPath();
17.   System.out.println("path:"+id); //path:test
18.   //获取所有的参数值
19.   String queryString = uri.getQuery();
20.   System.out.println("queryString:"+queryString);//queryString:p=12&d=1
  
```


20、AAR 库怎么来配置(2017-2-25)

1. 什么是 AAR

Android Library 编译成的文件就叫做 AAR (Android Archive) , 该结构跟 Android app 一样 , 它里面可以包含 : 源码 , 资源文件和 Android manifest 文件。

2. 如何生成 AAR

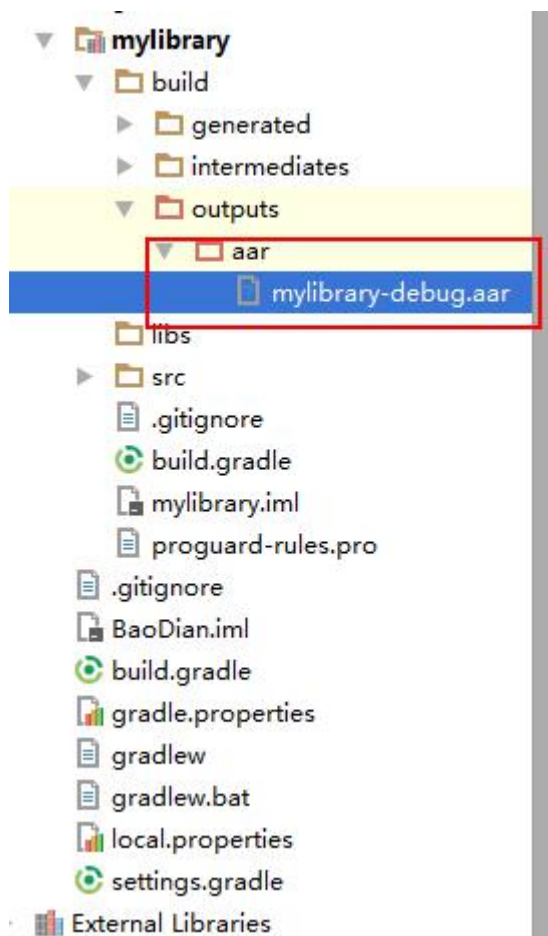
有两种方式都可以生成 AAR , 一种是直接创建 Library Module , 另外一种是将已有的 App Module 修改为 Library Module。

1) 创建一个 Library Module 的流程如下 :

1) 点击 File > New > New Module

2) 在 Create New Module 窗口 , 点击 Android Library, 然后点击 Next.

3) 给 Library 输入名字 , 选择一个最小支持的 SDK 版本 , 然后点击 Finish。当该 Library 编译好以后就能在如下图所示的目录中看到编译好的 aar。不过默认是 debug 版本的 , 需要通过 build 菜单项生成 release 版本。



2) App Module 转变成 Library Module

1) 打开 app module 的 build.gradle 文件，在顶部，你可以看到: `apply plugin: 'com.android.application'` 更改 plugin 的值为 `apply plugin: 'com.android.library'`.

2) 删除掉 `applicationId` 这个配置，library module 是没有 `applicationId`，只有在 manifest 文件中有个 `package` 值表明包目录。

3) 点击 Sync Project with Gradle Files.

注意：我们不应该在 library 中声明的 activity 中添加:

```
1. <intent-filter>
2.     <action android:name="android.intent.action.MAIN"/>
3.     <category android:name="android.intent.category.LAUNCHER"/>
4. </intent-filter>
```

3. 如何使用 AAR

如果预引入 AAR 的 APP 跟 Library 是同一个工程中的，那么我们其实是没有必要多此一举去引用 Library 生成的 AAR，而是应该直接将 Library 作为 APP 的依赖即可，这就是我们最常用引用库文件方式之一。如果我们只有一个 AAR 文件，那么可以采用如下方式将其添加到我们的工程中。

下面就以 mylib.aar 文件为例来详述下方法

- 1、把 aar 文件放在一个文件目录内，比如就放在 libs 目录内
- 2、在 app 的 build.gradle 文件添加如下内容

```
repositories {  
  
    flatDir {  
  
        dirs 'libs' //this way we can find the .aar file in libs folder  
  
    }  
  
}  
  
dependencies {  
  
    compile(name:'mylib', ext:'aar')  
  
}
```

- 3、之后就可以使用上面 AAR 中的一切资源了

21、AndroidStudio 项目中 Module Library 和 App 的 Manifest 编译合并 minSdkVersion 有什么限制关系(2017-2-25)

app module 的 minSdkVersion 必须大于等于 library module。library module 最后被编译成 app module 的一部分，所以 app module 使用的 SDK 版本必须包含 library module 使用的 SDK 版本中的所有 API。如图

```
Execution failed for task ':app:processDebugManifest'.  
> Manifest merger failed : uses-sdk:minSdkVersion 19 cannot be smaller than version 20 declared in library [zhihui Beijing:library_indicator:unspecified]  
F:\AWSP\zhihui Beijing\app\build\intermediates\exploded-aar\zhihui Beijing\library_indicator\unspecified\AndroidManifest.xml  
Suggestion: use tools:overrideLibrary="com.viewpagerindicator" to force usage  
BUILD FAILED  
Total time: 2.73 secs
```

22、不借助第三方怎么显示圆形图片(2017-2-25)

最新的 Android SDK 中，增加了对于圆形、圆角图的支持，引入了 RoundedBitmapDrawable，RoundedBitmapDrawable 的作用是将一个常规图片修剪成圆形或圆角图。RoundedBitmapDrawable 的出现，从此在处理简单圆形、圆角图时候简单多了

```
5. ImageView imageView= (ImageView) findViewById(R.id.imageView);
6.
7.     RoundedBitmapDrawable circleDrawable = RoundedBitmapDrawableFactory.create(
8. getResources(), BitmapFactory.decodeResource(getResources(), R.drawable.art));
9.     circleDrawable.getPaint().setAntiAlias(true);
10.    circleDrawable.setCircular(true);
11.    imageView.setImageDrawable(circleDrawable);
```

7. 项目面试常见问题 (★★★★)

1. 公司人员构成

不同的公司人员构成差别很大，下面给个真实案例说明。

真实案例 1：北京某给政府做即时通信的公司，中等规模。

公司大致分为开发部（有 10 几个人），产品组（2 个产品经理，1 个美工），测试组（3 个人，兼客服）和人事行政部（3 个）。

开发部又以项目分组：密讯组、加密电话组、原网组、rom 组（前四个都是安卓）和网络组。每组设一个项目负责人，APP 组员一般情况下是 2-3 个人维护开发。根据各个项目的开发情况，总监会灵活调配开发人员。

真实案例 2：北京某科技公司，给广电总局和各个电视台做后台系统，总共人员 280 人。

公司分销售部、财务部、事业一部、事业二部、事业三部、基础研发部。每个事业部都有三个组，分别为 Java 开发组，C/C++ 开发组，测试组。每个组大概 5 到 8 个人。

2. 开发周期

以真实项目密讯（即时通讯）为例：

一般 3~5 个月一个大版本（目前 6.0：重大性能优化，加密算法的改变等），原网组几个 APP 也大致这个时间。

平时的维护无特殊情况是一周一个小版本（例如：6.0.1，两周交替，一周 bug 修复，一周小功能添加，重大 bug 修复或者添加重要功能，直接升 6.1.0）。

3. 项目中遇到的难题

以真实项目密讯（即时通讯）为例：

1.对于初接触信息加密的新手来说，即时通讯所涉及的加密方案比较复杂，理清思路是开发前期比较麻烦的一块。

（接着人家要问，展开回答）

2.云文件的模块，服务器方面维护麻烦，客户端优化不足，前期用户体验不好，后来直接丢给阿里云服务器了。

（关于断点续传，第三方平台，文件迁移等展开）

3.长连接不太稳定，一直在优化，现在虽然差强人意也还凑合。（优化的过程可以展开）

4.UI 方面，密讯的聊天界面是精华（具体实现可以展开）

补充：开发工具是 eclipse，用的版本管理工具是 Git，用户服务器是自己的，文件服务器用的是阿里云。

总结：

大家找项目的时候大致可以根据这样的思路去找。项目里用到什么技术，怎么实现，都有哪些重难点，从这几个方面入手进行准备。

面试的时候适当的去引导到你比较擅长的模块，然后进行展开延伸。这里就可以体现自己项目技术的难点，解决方案和自己的收获。

遇到不是很清楚的问题说出自己的理解，完全不懂的问题要坦诚别瞎说，可以跟面试官交流，尽量在面试中多学点东西。

4. 项目中最大的收获

5. 项目是如何上线的

以真实项目密讯（即时通讯）为例：每周三提测，每周四晚经测试组许可，打包发给网络部发布。（因为用户群体办公需要，若新版有测试未发现的 bug，周五我们可以发布紧急修复版）。

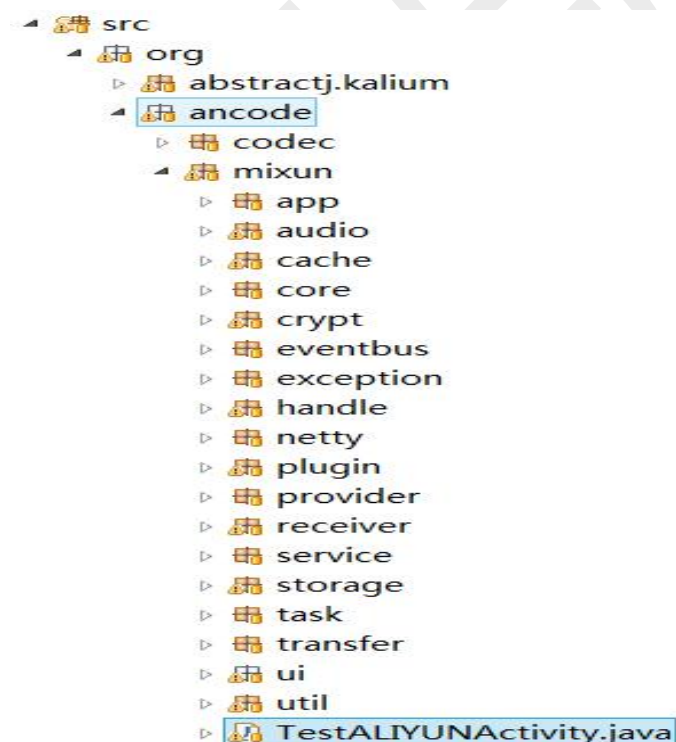
6. 项目是如何盈利的

以真实项目密讯（即时通讯）为例：就我所知，公司的盈利主要有两块：一个是政府立项，具体金额未知，另一个是加密电话组（驻扎奇虎 360 公司，目前该项目在奇酷手机上投入使用），一年是 5 百万，连续三年。另：我司卖给政府的定制手机（刷个 rom，装一些我们的 APP）是一台 1 万，手机原价 2 千。

7. 绘制项目架构图

这里目前只给出公司项目包结构图：

以真实项目密讯（即时通讯）为例：



8. 项目开发流程

以真实项目密讯（即时通讯）为例：

我进公司时，密讯已经迭代到 4.0。现在跟大家说说从 5.0-6.0 这个大版本迭代的整个过程。（底层加密算法的替换，UI 的大幅更改，相当于重做了一遍，有一定的借鉴价值）

1.技术总监开会，负责底层加密模块的同事给我们讲新算法（椭圆加密算法，这整个是项目的技术核心），根据项目的功能模块需求敲定框架。

2.分工：

开发部分：加密底层一人（用 C 语言做加密算法的 so 类库），JNI 一人（对底层库的封装，测试接口，写接口文档），服务器一人（技术总监，erLang 语言写的），APP 两人（负责 APP 的实现，接口功能的具体测试）。

产品部分：产品经理负责设计功能的业务逻辑和交互，美工配合产品出图

测试部分：每开发一模块的功能就进行测试

3.三线并行：

①：（底层实现）加密底层封装 — 接口调试 — 写接口文档

②：服务器开发

③：框架重构 — UI 实现 — 调通接口的功能接入实现 - 提测

9. 你在项目中的角色

以真实项目密讯（即时通讯）为例：

我是密讯组的组员，平时主要负责维护密讯，也被抽调到原网组到新项目部分模块开发。

主要职责：

1.参与信息加密方案的制定

2.负责登录，联系人，云文件等模块的开发和维护

3. 后续新功能的开发和 bug 的修复

10. 你负责项目中的哪些模块
11. 讲讲你负责模块的具体实现
12. 项目中都用到了哪些第三方框架

以真实项目密讯（即时通讯）为例：

- Android Private Libraries
 - oss-android-sdk-1.0.0.jar - D:\项目
 - okio-1.4.0.jar - D:\项目\mixun\libs
 - android-support-v4.jar - D:\项目\
 - okhttp-2.4.0.jar - D:\项目\mixun\li
 - guava-r09.jar - D:\项目\mixun\libs
 - sqlcipher.jar - D:\项目\mixun\libs
 - eventbus-2.4.0.jar - D:\项目\mixu
 - commons-codec.jar - D:\项目\mix
 - netty-3.9.5.Final.jar - D:\项目\mixu
- Android Dependencies
 - mixun5.0-viewpagerindicator.jar
 - mixun6.0-bottomsheet.jar - D:\项目
 - mixun6.0-pullrefresh.jar - D:\项目
 - mixun6.0-switchbutton.jar - D:\项目

13. 有没有自己写过框架
14. 业余时间你是如何提高自己（学习）的
15. 有没有自己的技术 blog
16. 你的职业规划
17. 为什么离职
18. 为什么选择我们公司
19. 说说你们项目的亮点和不足
20. 你们的项目是如何保持风格一致的
21. 项目架构是如何搭建的

22. 屏幕适配是如何解决的
23. 都看过哪些源码
24. 项目版本是如何升级的
25. 用的什么版本控制工具
26. 你能独立开发吗
27. App 跟服务器是如何交互的
28. 需求文档写过吗
29. 接口文档写过吗

以真实项目密讯（即时通讯）为例：这里只给出部分接口文档截图。

二、接口定义

2.1注册及登陆

2.1.1客户端注册新用户第一次上传自己的签名及加密公钥

1. 客户端向服务器递交自己的签名和加密公钥对。

```
post https://www.han2011.com/pubkey/v2/register
"sign.public": Hex(ClientSignKeyPair.Public),
"box.public": Hex(ClientBoxKeyPair.Public),
```

2. 服务器需要对上传上来的数据进行签名后返回给客户端,返回数据包括:

```
BoxPubkeyHex = Hex(ClientEncryptKeyPair.Public),
SignPubkeyHex = Hex(ClientSignKeyPair.Public),
{
  "mixunid": MixunId
  "public.sig": Hex(crypto_sign(BoxPubkeyHex+SignPubkeyHex+MixunId), ServerSignKeyPair.Private)
}
```

3. 客户端收到该数据后，对该签名进行验证，验证过程如下：

```
crypto_sign_open(ClientSignKeyPair.Public+ClientEncryptKeyPair.Public, DeHex(public.sig), Serve
```

4. 验证通过后，向服务器发出确认公钥请求。

```
SignPubkeyHex = HEX(ClientSignKeyPair.Public),
BoxPubkeyHex = HEX(ClientEncryptKeyPair.Public),
post https://www.han2011.com/pubkey/v2/register/confirm/{mixunid}
{
  "public.sig": Hex(crypto_sign(BoxPubkeyHex+SignPubkeyHex, ClientSignKeyPair.Private)),
}
```

5. 服务器收到该数据后，对该签名进行验证，验证通过后，将客户端上传的数据根据用户标识确认保存数据库，验证过程如下

```
crypto_sign_open(ClientSignKeyPair.Public+ClientEncryptKeyPair.Public, DeHex(public.sig), Client
```


30. 云服务器都用过哪些

阿里云。

31. 第三方平台都用过哪些

百度地图、极光推送、Bmob 云数据库、环信即时通信、七牛云存储、友盟统计。

8. 面试实战记录 (★★)

(本章节的内容记录了本人于 2015 年 5 月份在上海的面试的真实情况。在这里分享给大家，希望对大家有帮助。

该部分内容网络连接：<http://bbs.itheima.com/thread-196394-1-1.html>)

挑战公司 No. 1：上海创梯信息科技有限公司

公司地址：上海杨浦区昆明路 1209 号尚凯大厦副楼 504 室



面试时间：5 月 12 日 10:00 AM.

面试结果：顺利砍下 Android 技术总监，15K offer:victory:

面试过程：

10:00 到公司，前台 MM 给了张面试人员登记表，10 分钟搞定表格。

10:30 由于公司 BOSS 正在面试其他人，因此，又等了几分钟。

10:40 在 BOSS 办公室与 BOSS 斗智斗勇，聊了有接近 1 个小时。

疯狂的笔试+面试记录（前方内容“高能” :funk:，请准备高度精神集中前行）：

1. 笔试

由于 BOSS 正在打造公司新业务，刚刚开始组建公司技术团队，公司没有人懂 IT 技术，故本次面试没有笔试题，额。。。

2. 面试过程问答精选：

旁白：进入办公室后，我淡定等待，约莫几分钟后，一位 35 岁左右，看起来非常老道的 BOSS 走了进来。几句寒暄之后，问了我一些关于我的学校、专业、工作经验、接触的 Android 项目等普通流水线式的没有营养问题，轻松搞定！接着，BOSS 终于切入到了重点：

BOSS 问：其实我们想做一个快递业务的 APP，类似于滴滴打车。用户如果想要寄快递，只需要打开 APP，查找附近都有哪些快递员，然后直接跟附近的快递员联系。如此，既能方便用户，又能提高快递员的收入。

我（阳哥）答：我有一个问题想要问一下，咱们公司（注意，一定要说“咱们”，让别人感觉你已经融入他们的公司团队了）不像顺丰，不是典型的物流公司，为什么我们要做快递类的 APP 呢（主动沟通交流，有问题就问，这样面试官才能跟你聊起来！）？

BOSS 答：你知道的那些物流公司现在都是各自为政。例如，你用顺丰快递，你可以下载个顺丰的 APP。但是，你以后可能还要用中通、申通、圆通等等这些物流公司，难道你要下载十几个 APP 吗？而我们要做通用的快递类 APP！

BOSS 问：你知道一个 APP 重要的是什么吗？

我（阳哥）答：用户体验！

BOSS 说：不对，是流量，也就是用户数量（好吧，阳哥作为技术屌丝，关注多的就是 APP 用户体验，所以，不敢去反驳他，先顺着他的意思来）。

BOSS 问：如果让你去做一个类似于滴滴打车的 APP，你认为自己做的出来吗？

我（阳哥）答：像滴滴打车这样的 APP，不仅仅只是客户端的问题。它不是由几个简单的客户端程序员就能做出来的，实际上，它应该是由一个技术团队完成的大型项目。您目前能够看得见的仅仅是用户客户端，看不见的是庞大的后台系统。说详细点就是：滴滴打车这样的项目应该分 3 部分：服务器端，用户客户端，出租车司机客户端。服务端考虑的是数据的存储，业务的调度处理，以及与各个客户端的数据交互。而用户客户端我个人理解主要是一个 UI 的显示和与用户数据交互的功能。比如，用户将自己的当前坐标发送到服务器端（当前坐标可以通过百度地图、高德地图获取），服务器根据用户的坐标查找附近的所有空闲状态的出租车，然后将用户的用车需求推送到出租车司机客户端。出租车司机接收到信息后，如果愿意接这笔单子，就向服务器发送同意信息。服务器就是作为一个中介将用户与出租车司机联系起来。大概逻辑就是这样一个过程，中间的技术细节比较多。总之，一个人做滴滴打车是不太现实的（大家可以看出来，这一段，阳哥是傻瓜式教学，没有说的太复杂，好让 BOSS 能够听明白。

不然，技术说的太深，BOSS 就不知道说什么了，还怎么聊的开心。面试的最终目的就是让对方觉得看你很顺眼，这是关键！）。

（顺便啰嗦一下，黑马的 Android 课程中有部分 JavaWEB 课程。我们做为 APP 开发人员，不管是服务器端还是移动端都应该有所了解。如果仅仅会移动端开发技术而不懂服务端知识，长久看来，确实会很影响 Android 程序员向更高层次发展。）

BOSS 问：那么，如果我让你负责客户端的开发呢？

我（阳哥）答：客户端 Android 开发并不难，比如支付宝是一个很强大的金融系统。但是，你让我做它的 Android 端开发，我感觉我没问题。毕竟，Android 客户端重点是信息的展示和与用户的交互。所以一个人做客户端是没有问题的。但是，目前市场上的企业很少有一个人做一个客户端项目的。市场竞争如此激烈，一个人做耗时太长，可能你还没有做出来，产品已经被淘汰了。因此，应该多招几个 Android 程序员，这样团队就算有人员流失也不会影响整个公司的发展（展示自己对于项目组人数把控的个人看法）。

旁白：几轮 PK 下来，BOSS 基本上对我已经很信服了。

BOSS 说：其实，在面试你之前，我已经招聘到了两个移动开发的程序员，一个是 Android 程序员（6K），一个是 iOS 程序员（8K），你可以看一下他们两个的面试登记信息和他们的简历。

我（阳哥）问：根据这两份简历可以看出来，这两个人的技术有些一般（不是装逼，他们的简历写的确实有点 LOW）。

BOSS 问：是的，我看的出来，你的经验要比他们丰富地多。所以，我准备让你出任我们公司的技术总监，带着他们两个做客户端开发，你认为你有信心做好吗？

我（阳哥）答：这个职位我以前没有做过，所以心里没有多大的底（阳哥认为这样的问题确实比较考验人，你直接回答“可以，没问题”，就会显得你浮夸、不谨慎、办事不牢靠。如果你直接回答“不能”，

又显得你不敢担当，没有上进和挑战的精神。因此回答这样的问题必须两者兼顾，既要谦虚谨慎又要表现出富有挑战精神）。

BOSS 说：我知道你没有做过，但是，一个人挑战任何一样新工作之前都是没有做过的。你有没有想过一旦你能够胜任我们的公司的技术总监，你以后的身价将和现在完全不同，所以，对自己要有一些信心（BOSS 故意画大饼，就说明你有戏了！）。当然，这是一个管理岗位，需要什么样的技术人员你可以招。

我（阳哥）说：好吧，其实我希望挑战一下，但是我不一定能够达到您对我那么高的期望（始终保持谦虚的态度很重要！你的态度决定别人对待你的态度！）。

BOSS 问：你尽力就好，请问你的期望月薪是多少？

我（阳哥）答：15k（第一次面试，我心里也不太清楚上海的市场，15k 一般是黑马班里的大神级同学要的薪资，所以，试水一下）。

BOSS 说：好的，没问题（竟然没有砍价:L，额，土豪公司）。

旁白：随后，阳哥又和 BOSS 聊了一些其他内容，由于 BOSS 马上要开会，就告诉阳哥如果你同接受我们的邀请，那么我希望我们明天下午可以好好聊聊。。。之后，一堆寒暄，不列出来了。虽然，第一家面试就这样结束了，没有技术上太多的 PK。但是，却使阳哥在精神上却得到了巨大的鼓励，面试第二家上海公司有了更大的信心！

面试吐槽：

阳哥的上海处女面就这样丢掉了！遗憾的是 BOSS 不太懂技术，没有碰撞出技术的火花。但是，让人幸运的是他提供给我了一个技术总监的岗位。整个过程可以看到，面试的时候，面试官一方面会明中考察我们的技术。另一方面，在暗中实际上也在考察我们的沟通能力、思维逻辑能力、管理能力等这些软实力。所以，黑白的每一位达人，在黑白拼命码代码的同事，别忘了多和你的小伙伴分享技术，不

要忘记中午的时候抓住每一次的公开演讲机会。有时候，一个人综合实力远大于单纯的技术竞争力！加油吧，骚年们！

挑战公司 No.2：上海复深蓝信息技术有限公司

公司地址：上海市徐汇区漕河泾开发区虹梅路 2007 号 1 号楼



面试时间：5 月 12 日 14:00 PM.

面试结果：顺利砍下 Android 工程师，17K offer+1K 补助 = 18K 月薪:victory:

面试过程：

14:00 到公司，前台 MM 给了一张面试登记表和一张 Android 笔试试题，然后一位和蔼的大叔（技术大拿）将我带公司接待处，让我开始笔试。

14:20 笔试完成，笔试很简单，做完后把试题交给前台，前台帮我联系面试官。

14:25 跟一个年龄大概在 25~30 岁之间的年轻技术官进行了接近 1 个小时的面试（通过交谈阳哥感觉面试官应该是项目经理或者开发组组长）。

15:20 人事面试，人事面试后提前跟我说 CTO 会对我进行一个电话复试。

第二天晚上大概 19:00，公司 CTO 对我进行了技术复试，总共持续了 31 分钟。电话复试题要比初试难的多了，姜还是老的辣。我把复试的对话录音保存下来了，作为以后上海黑马就业指导课程的案例讲解（内容何止“高能”，简直就是高达！）。

疯狂的笔试+面试记录（前方内容“残酷”：funkt，请准备受虐心态前行）：

1. 笔试

笔试时出现了一个小插曲，和大家分享一下。阳哥笔试所在的接待区是一个半开放式的区域，共有 3 张桌子，每张桌子上都有一支签字笔。我先用第一张桌子，发现笔是坏的（郁闷），然后换到第二张桌子，发现笔还是坏的（心想：尼玛，这么霉，顺便汗一下），最后只能换到第三张桌子，你猜怎么滴，对，笔还是坏的（想要骂娘了，FUCK！）。没辙了，翻遍我的背包，自己也没带笔。难道企业是用这种方式考察一个应聘者的吗？这怎么办？

我想去前台借吧，前台肯定有（不要去人事那里借，前台毕竟只是前台，不会管你太多的细节，人事可就不一样了，他们会认为一个面试者来比试不带笔是极不严肃的表现）。热心的前台 MM 很乐意地把自己的笔借给了我（心里一阵暖啊！这家必须拿下！）。

为了方便大家更清晰的看到笔试题目，阳哥手录笔试题和答案，看你能做对多少？！

Q：Android 的四大组件有哪些？

A：Activity、Service、ContentProvider、BroadcastReceiver。

Q：请描述下 Activity 的生命周期？

A：onCreate、onStart、onResume、onPause、onStop、onDestroy、onRestart。

Q：如何将一个 Activity 设置成窗口模式？

A：将 Activity 的样式设置成：android:theme="@android:style/Theme.Dialog

Q：如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

A：调用 Activity 的 finish 方法可以退出当前 Activity。可以自定义一个 Application，在 Application 中声明一个成员变量 ArrayList 用于存放打开的 Activity，当退出时遍历 ArrayList，依次调用 Activity 的 finish 方法。

Q：请介绍下 Android 的五种布局。

A：LinearLayout、RelativeLayout、FrameLayout、RelativeLayout、TableLayout

Q：请介绍下 Android 的数据存储方式。

A：SharedPreferences、XML、SQLite、文件系统

Q：DDMS 和 TraceView 的区别。

A：DDMS 的全称是 Dalvik Debug Monitor Service，是 Android 开发环境中的 Dalvik 虚拟机调试监控服务。TraceView 是程序性能分析器

Q：说说 Activity、Intent、Service 以及他们之间的关系。

A：Activity 负责界面的显示和用户的交互，Intent 封装了数据，可以实现 Activity 之间以及 Activity 和 Service 之间数据的传递。Service 运行在后台进程，一般我们会让给其运行一些后台任务，Activity 通过 StartService (Intent) 或者 BindService (Intent) 可以启动 Service。

Q：请介绍一下 ContentProvider 是如何实现数据共享的。

A：我们可以定义一个类继承 `ContentProvider`，然后覆写该类的 `insert`、`delete`、`update` 等方法，在这些方法里访问数据库等资源。同时我们将 `ContentProvider` 注册在 `AndroidManifest` 文件中，其他应用需要使用的时候只需获取 `ContentResolver`，然后通过 `ContentResolver` 访问即可。

2. ROUND 1：PK 项目经理技术面试问答精选

项目经理问：Activity 都有哪些生命周期？

我（阳哥）答：这个问题其实在笔试题中我已经给出答案了（此时，阳哥表示非常疑惑~）。

项目经理说：我知道，你再说一遍。

旁白：当时阳哥我真没搞明白，为何笔试上的题目他还是问我一遍，而且笔试上的好几道题他都重复问了一遍。现在想想应该是他怀疑我笔试的时候作弊了，比如我可以百度什么的。好吧，我不就是笔试的时候出去跟前台妹子借了一支笔，然后又把笔试题给拍了个照片而已嘛。而且，我那 3G 的联通定制机安装的移动 4G 的卡，只能享受 2G 的网速，我打开个百度都得几分钟。算了，不能和面试官较劲！我忍！

我（阳哥）答：Activity 有以下生命周期回调方法，比如常用的有 `onCreate`、`onStart`、`onResume`、`onPause`、`onStop`、`onDestroy`、`onRestart`。默认情况下，如果我们不给 Activity 设置横竖屏配置信息的话，在横竖屏切换时会将一个 Activity 销毁掉然后重新创建。

项目经理问：Fragment 你用过吗？

我（阳哥）答：这个当然用过呀。现在的应用中基本都有 Fragment 的应用，Fragment 比较小巧灵活。

项目经理问：那 Fragment 跟 Activity 之间是如何实现值传递的？

旁白：其实项目经理在问我会不会 Fragment 的时候，我已经预料到接下来会问我 Fragment 跟 Activity 直接的值传递问题。对于前面的问题，如果我说不会，就不会有下面的问题，但是如果说不会的话，那么项目经理很可能因为这一个问题而把我 PASS 掉，因为 Fragment 是 Android 中一个非常重要的知

识，这个是必须会的（黑马的 Android 基础课程中有一天就是讲 Fragment 的）。如果连这个都不会，那么再好的面试技巧也挽救不了同学们的！

我（阳哥）答：Activity 可以先获取 `FragmentManager` 或者 `SupportFragmentManager`，前者是 v4 包下的，向下兼容因此用的比较多。然后这些 Manager 通过 Fragment 的 tag 或者 id 调用 `findFragmentByTag("tag")`，`findFragmentById("id")` 找到我们需要的 Fragment 对象，然后通过调用 Fragment 对象的方法来进行值的传递。

项目经理问：Android 中都有哪些组件需要在清单文件中注册？

旁白：四大组件是学习 Android 的必会知识点，也是黑马 Android 基础中的重点内容，因此这个问题同学们其实应该是很好回答的！

我（阳哥）答：一般来说四大组件都需要在 `AndroidManifest.xml` 中进行注册，不过其中 Activity、Service、ContentResolver 是必须注册的，而 `BroadcastReceiver` 可以在清单文件中注册，也可以不注册，这也分别叫做静态注册和动态注册。

旁白：在 Android 中一般通过 XML 文件注册的组件，我们叫静态注册，而通过代码注册或者创建的组件我们叫做动态注册。动态注册和静态注册这两个名称听起来很高大山，其实理解起来 so easy 滴！

项目经理问：你自己有做过自定义控件吗？

我 阳哥 答：自定义控件做过，比如我们项目中的 `SlideMenu`，`LazyViewPager`，`Pull2RefreshListView`，`VerticalSeekBar`，`RandomLayout` 等都是自定义控件。

项目经理问：那你说说 View 的绘制过程？

我（阳哥）答：View 绘制是从根节点（Activity 是 `DecorView`）开始，他是一个自上而下的过程。View 的绘制经历三个过程：Measure、Layout、Draw。

旁白：黑马的老版本课程体系中有 2 天的自定义控件，在这两天课程中同学们能够学会 SlideMenu，Pull2RefreshListView，优酷菜单等自定义控件，目前的黑马课程又对自定义控制进行了加强，添加了多种 QQ5.0 新特性内容，当然难度其实也不小。

项目经理问：ListView 你们应该有用过吧？

我（阳哥）答：这个在我们的项目中应用的很多，其实在如今所有流行的 App 中，ListView 都有一个大量的应用，说 ListView 是一个使用率高的控件都不为过。

项目经理问：ListView 的优化你们是怎么做的？

我（阳哥）答：ListView 的优化有多种多样的策略。在我们的项目中主要做了如下优化。1、重用 ConvertView，2、给 ConvertView 绑定 ViewHolder，3、分页加载数据，4、使用缓存。前两个是通用的解决方案，后两个是针对我们业务的个性化解决方案。我们的数据来自服务端，如果服务端有 1000 条数据的话，我们客户端不可能傻瓜式的一次性用 ListView 把这些数据全部加载进来，因此我们就用分页加载数据，每次加载 20 页，当用户请求更多的时候再获取更多数据，网络的访问就算网速再快也多多少少会有一定的延迟，因此我们的网络请求是异步处理的，同时从网络加载来的数据使用了 2 级缓存来处理，第一级是内存级别的缓存，第二级是本地文件的缓存。当 ListView 加载数据的时候首先从内存中找，如果找不到再去本地文件中找，只有都找不到的情况下才去请求网络。

旁白：ListView 的优化是黑马课程一个重要的知识点，因此大家上课的时候这个必须得学会，不然在以后的面试中肯定会栽跟头的。

3. ROUND 2：第二轮 PK CTO（非人类）技术面试问答精选

旁白：第二轮技术复试是在第二天晚上 7 点时开始的，当时，阳哥我刚从外面面试完回到家中。跟我聊的是他们公司的 CTO，通过聊天也能感受到他的技术非同寻常，前几个问题问我时感觉多方有种咄咄逼

人的气势（貌似面试的人太多了，对于被面试人员都是不屑的口气）。不过几轮技术 PK 后，发现原来 CTO 对人类也可以这么温柔和气的（尼玛，技术好才能被人看得起啊！心底话！）。

CTO 问：说说你对泛型的了解？

我（阳哥）答：泛型是 jdk5.0 版本出来的新特性，他的引入主要有两个好处，一是提高了数据类型的安全性，可以将运行时异常提高到编译时期，比如 ArrayList 类就是一个支持泛型的类，这样我们给 ArrayList 声明成什么泛型，那么他只能添加什么类型的数据。第二，也是我个人认为意义远远大于第一个的就是他实现了我们代码的抽取，大大简化了代码的抽取，提高了开发效率。比如我们对数据的操作，如果我们有 Person、Department、Device 三个实体，每个实体都对应数据库中的一张表，每个实体都有增删改查方法，这些方法基本都是通用的，因此我们可以抽取出一个 BaseDao<T>，里面提供 CRUD 方法，这样我们操作谁只需要将我之前提到的三个类作为泛型值传递进去就 OK 了。而数据的安全性，其实程序员本身通过主观意识是完全可以避免的，何况某些情况下，我们还真的想在 ArrayList 中既添加 String 类型的数据又添加 Integer 类型的数据。

CTO 问：你知道 Java 的继承机制吗？

我（阳哥）答：知道呀，这个问题很简单呀！java 是单继承多实现呀。

CTO 问：那你知道 java 为何这样设计吗？

旁白：从上面的问题也可以看出越是资历老的程序猿越喜欢刨根问底，因此如果同学们面试的时候遇到一个年龄稍微大点的程序员，那么一定要提前做好思想准备了，他可能先问你一个看似很简单的问题，然后再追问一个很深的思想或者原理。

我（阳哥）答：为何 Java 这样设计，其实这也是我一直的一个小疑惑。不过我是这样理解的。我只能用反证法，如果一个类继承了类 A 和类 B，A 和 B 都有一个 C 方法，那么当我们用这个子类对象调用 C 方法的时候，jvm 就晕了，因为他不能确定你到底是调用 A 类的 C 方法还是调用了 B 类的 C 方法。而多实现就不会出现这样的问题，假设 A 和 B 都是接口，都有 C 方法，那么问题就能解决了，因为接口

里的方法仅仅是个方法的声明，并没有实现，子类实现了 A 和 B 接口只需要实现一个 C 方法就 OK 了，这样调用子类的 C 方法时，Java 不至于神志不清。从另外一个方面考虑的话应该就是 Java 是严格的面向对象思想的语言，一个孩子只能有一个亲爸爸。

CTO 问：Java 的异常体系你知道吗？

我（阳哥）答：知道呀，顶层是 Throwable 接口，往下分了两大类，一个 RuntimeException 另一个是普通的 Exception。

CTO 问：那你知道这两类异常的区别吗？

我（阳哥）答：当然知道，java 的命名是见名知意的。从名字上我们也知道 RuntimeException 就是运行时异常，在运行的时候才能被 jvm 发现导致程序的终止，而普通 Exception 必须进行 try、catch 处理，或者在方法上用 throws 声明。

CTO 问：那你的期望薪资是多少？

我（阳哥）答：我期望的薪资已经给贵公司人事说过了，是 17k。

CTO 问：你这么年轻，就想要到 17k 呀！

我（阳哥）问：对的，我是还年轻，高中同样是学习 3 年，有的考上了重点大学，有的却只考上了个大专院校，甚至落榜，不同的人学习能力是完全不一样的，甚至可以用天壤之别来形容，因此如果只简单的用时间来衡量一个人的价值显然就是不太合理的，比尔盖茨跟我这样大年龄的时候已经是亿万富翁了，而我还在找 17、18k 薪水的工作（前面的问题已经回答的这么漂亮了，谈薪水的时候一定要表现出绝对的自信！）。

CTO 问：你说的对，不过我还得问你几个问题，你说你们项目中有用到图片吗？

我（阳哥）答：这个当然有呀，我们新闻客户端基本上每条新闻都有图片，只有图文并茂的新闻才会有人看。

CTO 问：那你说你们遇到 OOM 异常吗？

我（阳哥）说：这个前期的时候我们的 APP 确实遇到过这样的问题，不过现在新的版本早就把这些为题给解决了。

CTO 问：那你们是怎么解决的？

我（阳哥）答：OOM 异常是 Android 中经常遇到的一个问题，程序员稍微不注意可能就导致其产生。因为 Android 的每一个应用都是一个 Dalvik 虚拟机，该虚拟机的默认堆内存只有 16M，远远无法跟我们的 PC 机比较，因此和容易导致 OOM (Out Of Memory) 异常的产生。导致这样的异常主要有以下原因：1、加载大图片或数量过多的图片，因为图片是超级耗内存的，2、操作数据库的时候 Cursor 忘记关闭，3、资源未释放，比如 io 流，file 等，4、内存泄露。我们用到的 OOM 主要是加载图片导致的。因为后面的三种原因都是可以通过约束程序员的编码规范来进行预防，或者使用性能分析工具来检查。

CTO 问：好的，那你们的图片是怎么处理的？

旁白：随后这就是 CTO 面试应聘者的一个习惯，他会追着一个知识点往死里问，直到问到系统的底层，或者他能理解的底。

我（阳哥）答：图片的处理主要用两种方式。我们的应用中有两处用到了图片，一个是 ListView 中展示的图片缩略图，这种情况的特点是数量大，但是单个图片内存小，只有几 kb，另外一种是大图片，就是用户通过手机拍摄的图片，然后通过 http 的 post 提交的方式提交到服务器上。然后在客户端将这个图片也展示出来。对于第一种情形，我们是通过三种技术手段来解决问题的，一是图片的缓存策略，二是 ListView 的优化，其实在上面我已经讲过，三是 WeakReference(弱引用)的使用。对于第二种情形，我们主要是首先通过 BitmapFactory.Options 参数获取图片的宽和高，然后再根据我们 ImageView 的宽高对图片进行一个很大比例压缩。

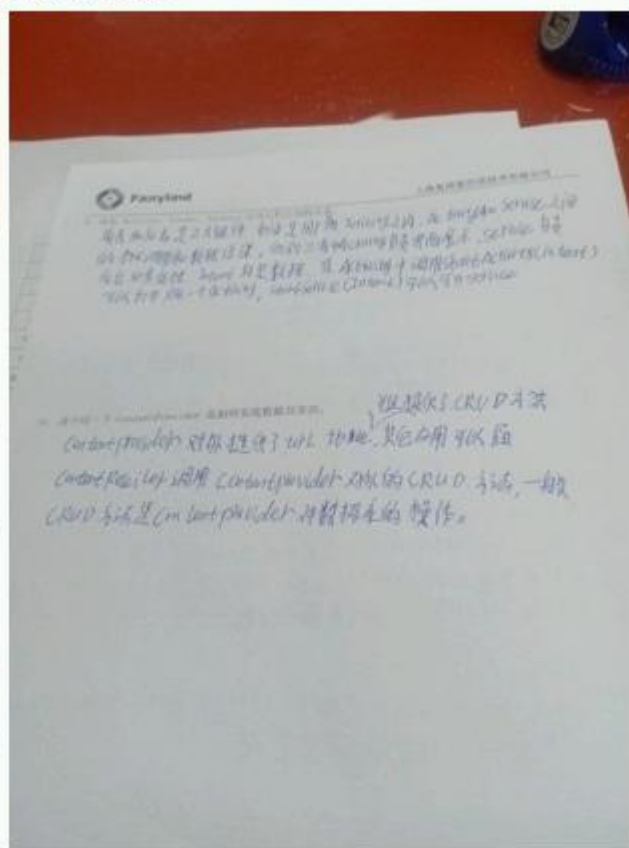
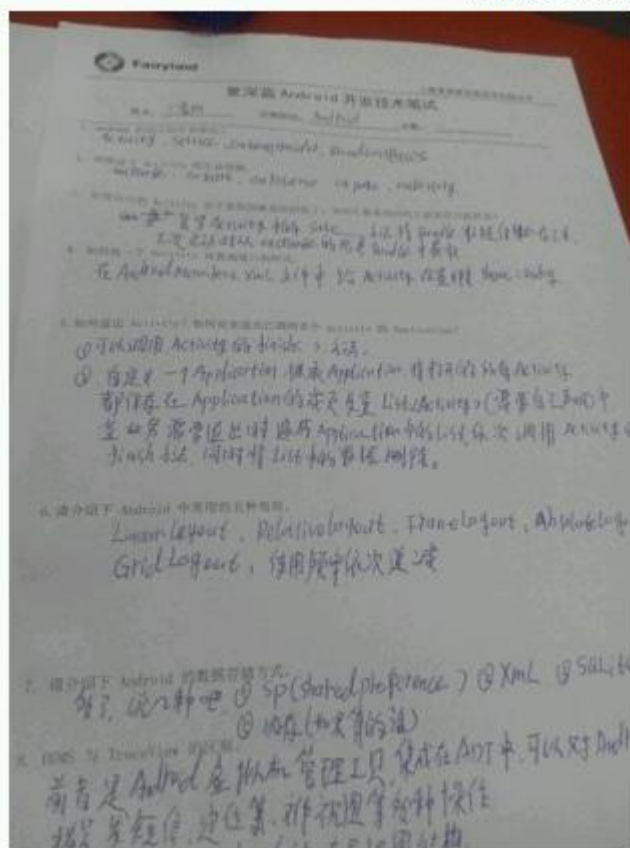
CTO 问：那你说说弱引用是怎么使用的？

我（阳哥）答：WeakRefrence 是一个类，在 ArrayList 中我们把这个类作为对象传递进去，把我们的图片放在 WeakRefrence 里面，这样当 davlik 虚拟机内存不够用的时候，就会把 WeakRefrence 对象回收掉，这样我们在 WeakRefrence 里面保存的数据也被回收了。

面试吐槽：

阳哥在上海的第二面终于遇到懂技术的人，把笔试、技术一面、人事、技术二面一气呵成的通了个关，不拖泥带水，最终人事给了基本薪资 17k+1k 多补助的 offer，这种感觉也许只有你经历过了才能体会吧！相信黑马学子经过四个月的刻苦磨练也能远远超过我的水平。加油吧，骚年们！

阳哥冒死偷拍的笔试题！



阳哥录用通知邮件！

2015年发体盟新员工报到须知（王阳）.docx

尊敬的王阳先生：

您好！

非常感谢您应聘上海发体盟信息技术有限公司的职位！您在面试中的出色表现给我们留下了深刻的印象。本公司拟同意录用您，我们诚挚的邀请您于2015年5月18日上午9:00到公司办理具体录用事宜。但是无论在任何情况下，您必须在收到本通知3天内书面回复本公司（书面回复包括短信、特快专递、通过面试登记表上记载的电子邮箱发出的电子邮件或登记的手机号码发出的短信），且在指定时间内到公司，否则本通知自动无效。本公司录用您的前提是：您提供的入职申请资料合法、真实、完整、有效，且您具备胜任本职位工作的身体健康状况。

请您一并携带以下材料、证件及报告：

1. 身份证复印件、学历（学位）证明复印件及其他资质证书复印件（验原件）（应届毕业生须携带学生证）；
 2. 原公司有效工单、劳动手册或离职证明（应届毕业生可忽略此项）；
 3. 累计就业工龄证明（社会保险累计缴费记录证明，可网上截图(详情见附件一)，应届毕业生可忽略此项）；
 4. 技术资质证书或者其他认证证书（需携带原件或者原件扫描件）；
 5. 公积金账号（应届毕业生可忽略此项）；
- 1) 请提供上家单位全称及个人公积金账号；

挑战公司 No.3：中阜投融资资产管理股份有限公司---中投融

公司地址：上海市静安区威海路 228 号招商局广场南楼 21 楼



面试时间：5 月 12 日 16:30 PM.

面试结果：顺利拿下 Android 工程师，15K offer+1k 住房补贴=16k:victory:

面试过程：

16:30 到公司，因为这一天安排了 3 家面试，这家本来是安排的 16:00 的，但是迟到了半个小时，不过企业招人都比较急，这个可以理解。

16:30~16:40 填写面试登记表，这家没有笔试题，填完后坐在公司前台附件的沙发上等了几分钟，桌子上放了一盘水果糖，诱人，也不敢吃，哈哈。

16:40~17:15 跟一位 Android 程序员进行第一轮 PK。

17:20~17:50 跟项目经理进行第二轮 PK。

17:50~18:00 跟人事谈薪资。

疯狂的笔试+面试记录（前方内容“高能”：funk:，请准备高度精神集中前行）：

1. 笔试

这一家没有笔试题，其实阳哥发现一个规律，一般 Android 开发团队刚刚组建的，或者 Android 开发人员不多的企业基本都没有笔试这一环节，这可能是他们公司的 Android 开发团队各方面还没有形成一套标准的流程原因吧。

2. 第一轮技术面试过程问答精选：

旁白：面试我的哥们儿比较腼腆，也许是他也刚进公司没多久的缘故，好像才 4 个月的样子。他主要问了我都做过哪些 Android 项目，怎么学习的 Android，都做过哪些 Android 控件，然后他又问了一个他们公司目前正在做的项目遇到的一个难题，问我怎么解决。

PS：最后这个问题，阳哥感觉他并不是在考察我的技术，而是以请教我问题的态度在向我咨询了。面试能面到这种程度，基本已经有戏了。

面试官问：介绍下你都做过哪些 Android 项目？

我（阳哥）答：这自己主要做过 3 个项目，新闻类的，应用平台类的，手机管理类的，其中自己参与多也做的成熟的是新闻类的一款 App。

旁白：上面的三种类型的项目，都是黑马 Android 课程体系中的教案，因此介绍项目基本没压力。

面试官问：那你在项目的开发中担任一个什么角色？

旁白：这样的问题其实很多面试官都问我了，感觉被问到的概率有 80%，此种问题显然是想考察我们的担当能力，技术担当和责任担当。在上海黑马 66 期的开班典礼上，我是这样给大家说的，我们 66 期有 70 多位同学，我们班分成 10 个小组，每个小组选出一名组长，组长负责全组的学习，组长不是固定不变的，每周考试一次，每次考试成绩高者为组长。每次考试如果这个组的平均分在班排名低的，组长得无条件接受惩罚。在黑马，没有个人排名，只有团队排名，我们更注重团队的协同能力，而不注重个人的表现。

我（阳哥）答：我在这个项目中属于主要参与者之一吧，或者说是主要负责人，我们 Android 项目不像是 javaweb 项目那样需要大量的程序员，像我们的项目，总共 2 到 3 个程序员 3 个月功夫就能搞定，因此我们几个人也没有绝对的说谁领导谁。不过这样的项目，让我们任何一个人去开发都是很 OK 的，只是时间问题。

面试官问：你们用什么代码管理软件？

我（阳哥）答：SVN。

旁白：在黑马有专门的一节课是讲 SVN 和 Git 的，并且后面的项目也是用 SVN 跟大家进行代码共享的。

面试官问：你学习 Android 的途径都有哪些？

我（阳哥）答：现在是互联网时代了，不像我们大学时代主要通过书本学习。在大学的时候选修的 Java 课程，那时候还是诺基亚时代，图书馆的移动开发书架基本是被诺基亚的塞班占据的。自己大学毕业的时候 Android 已经开始风靡全国了，那时候自己从网络上看到的 Android 开发的视频，然后开始学习的 Android。自己在 Android 的开发中遇到的各种问题一般都是靠百度解决的，说度娘是好老师真不为过，其实也用过 Google，不过被屏蔽了，也懒得翻墙，百度现在做的也不差，百度出来的 Android 技术主要来自如下专业网站，比如：CSDN、51CTO、ITEye、AndroidBus、EOEAndroid 等，对了还有一个国外的没有被屏蔽的网站是 Github，我们项目中的很多控件其实都是从 Github 上学习过来的。

面试官问：那你从 Github 上都用到过什么控件？

我（阳哥）答：Github 上的开源项目非常的多，基本上你想用的东西都有，比如 xUtils、HelloCharts、Clander、SlideMenu、SeatTable、LDrawer、SmoothProgress、Touch Gallery、ViewPagerIndicator。

面试官问：你自己有做过自定义控件吗？

我（阳哥）答：自定义控件做过，比如我们项目中的 SlideMenu、LazyViewPager、Pull2RefreshListView、VerticalSeekBar、RandomLayout 等都是自定义控件。哦，对了，最近我刚给我女朋友还做了一个 HideSlideBar 控件，我可以让你看一看。里面主要由 VerticalSeekBar+ProgressBar+NineOldAnimation 完成的。

旁白：我把我自己做的自定义控件给他演示了一遍，这个是我女朋友项目中需要用的一个需求，她不会做，我帮她写了一个 Demo，没想到正好用上了。

面试官问：哦，不错，你这个动画用的是属性动画吧？

我（阳哥）答：对的，这个属性动画，用了 JakeWharton 大神的开源框架。不过这个属性动画其实自己写也可以，内部原理比较简单，就是给控件设置一个开始位置，一个结束位置，设置一个延时时间，然后不停的更改控件的 layout 位置即可。

旁白：这个东西在黑马的项目中都有讲解，回答不难，在 Android 应用中我们会用到很多第三方的框架，我们不仅仅要会用别人的东西还必须知道别人写这个东西的内部原理。

面试官问：那好，现在我们有个项目遇到一个问题，你看如果让你做你应该怎么去解决？

旁白：这时候，阳哥已经感觉到自己已经 PK 成功了。不过面试官性格也不错，遇到问题善于寻求外界的任何帮助。

我（阳哥）答：可以的，什么需求您讲吧，我听听。

面试官问：我们的项目中有多个 Fragment，Fragment A 跳转到 Fragment B，Fragment B 跳转到 FragmentC，那么这时候我多次按返回键，如何能让 Fragment 跟 Activity 的任务栈一样，依次从 FragmentC 跳转到 FragmentB，再跳转到 FragmentA？

旁白：这个确实是需要比较棘手的问题。其实面试官也知道你没有做过类似需求的开发，事实也是这样。这就比较考验临场发挥能力了。这种问题可以从模仿 Activity 任务栈考虑解决。任务栈是一种数据结构，我们可以自定义这种数据结构，然后管理这个数据结构，但是每个 Fragment 都是独立的，如何把这些独立的 Fragment 关联起来，这都是需要考虑的问题。

我（阳哥）答：这个需求应该很好实现。我们可以这样，首先定义一个 BaseFragment，让 FragmentA、FragmentB、FragmentC 都继承 BaseFragment，第二在 BaseFragment 中定义一个 ArrayList，每打开一个 Fragment，把这个 Fragment 对象添加到 ArrayList 中，这样这个 ArrayList 就可以当做一个栈结构，第三我们需要设置返回键监听，当监听到返回键的时候，查看当前 ArrayList 中倒数第二个 Fragment 有没有 Fragment，如果有则取出该 Fragment 并把 ArrayList 中末尾 Fragment 删除，然后用 FragmentManager 的 Replace 方法，将当前 Fragment 替换成最新 Fragment 即可，如果 ArrayList 中只有一个 Fragment，且监听到了返回键，那就不对 Fragment 做处理，同时也不拦截该事件，这样也不会影响其他 Activity 之间的切换。

旁白：回答了上面的问题后，面试官说他回去试试，然后就去叫他们的领导了。

3. 第二轮技术复试过程问答精选：

旁白：第二轮面试我的是个技术大拿，主要负责公司整个软件架构的设计，这家公司之前只有 web 端，全都是他干的，现在公司向移动端发展，因此最近一直在招 Android 和 iOS 开发人员，他不太懂 Android，不然估计他一个人就能把一个 Android 项目搞定了。他问了我一些关于 http、数据安全

的知识，然后就开始跟我谈人生了，一直到他们快下班，他才把人事叫来跟我谈了谈薪资。

经理问：你做的 Android 的项目跟服务器交互都有哪些接口？

旁白：大家不要被接口这个词给迷惑了。这里的接口不是 java 中的接口类，也不是很高大上的东西，其实换成大白话就是你们的客户端跟服务端是如何实现数据的交互的。

我（阳哥）答：我们的接口有多种形式，第一种是 http 的形式，客户端跟服务器通过 http 协议传输数据，比如我们的新闻列表的请求都是给服务器发送的 get 请求，然后服务器把数据发给我们，我们上传给服务的图片是通过 http 的 post 请求方式完成的。第二种是 socket 完成的，服务端开启 ServerSocket，客户端开启 socket，然后客户端跟服务端建立长连接，这样实现了客户端跟服务端数据的即时通信，通信的协议是我们公司按照 xmpp 开放协议的基础上修改的，其实 xmpp 协议就是一个 xml 格式的数据。第三种是集成了第三方接口，比如分享功能用的是 ShareSDK，消息推送用的是 JPush，内置广告用的是万普世纪。

经理问：你们 http 传输数据的时候安全是怎么保证的？

我（阳哥）答：我们的数据有些是需要安全设置的有些不需要，我们的新闻类数据不需要特殊的添加安全设置，而用户注册，用户登录以及用户隐私数据保存是考虑安全性的。用户的密码等信息肯定不能进行明文传输的，我们将用户的密码在本地进行了 MD5 算法的加密，然后再传输。同时保存在本地的时

候也是加密后的数据。还有需要安全性更高的数据需要通过我们自定义协议通过 Socket 传输。

经理问：那你知道 MD5 的原理吗？

旁白：老程序员就是喜欢刨根问底，如果技术功底不雄厚的话确实这个问题很难应付，因为对于大多数人只需要用一个技术就行了，不会去关注他的内部原理。

我（阳哥）答：MD5 算以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了复杂处理后，输出由四个 32 位分组组成，将这四个 32 位分组合级联后将生成一个 128 位散列值。这个过程是不可逆的，我们也把他叫做数据指纹，但是我们依然对用户的输入进行安全校验，如果是纯数字类型密码，是不允许注册的，因此就算你 MD5 加密了，黑客也可以通过彩虹碰撞的形式进行暴力破解。

旁白：接下来，经理问了我几个问题可能难住我，关键是他也不懂 Android，只能问我 javaSE 和思想上的一些东西，之后就开始跟我聊人生了，说他们公司是国企背景，自己在公司发展多么好，公司未来要做一个什么样的产品，公司弹性工作制，每年至少 14 薪，又问我女朋友在哪，什么时候考虑结婚，在哪买房，老家是哪的，爸妈是干什么工作的，你是如何规划未来的等等。。。。。

4. 人事面试：

这家的人事面试与其说是人事面试还不如说成人事咨询，人事跟我介绍了公司背景，跟我谈了薪资，跟我谈了入职安排，又谈公司发展，公司的福利待遇等等。人事的最后一个问题都是出奇的相似，

人事：我该问你的问题都问完了，你还有什么要问我的吗？

PS：这些问题千万不要一个都不问，这样人事会认为你这个应聘者对我们公司不感兴趣。如果有其他人也去面试，那么你就可以能被 PASS 了。其实我们真的想去这家公司的话，肯定会有一大堆问题的，但是也不要问的太多，问的太多显得你很啰嗦，人事的耐心也是有限的，同时也不要问太低级的问题。对于我们程序员来说，应该最关注如下两大类问题，一是自己将要加入的开发团队的情况，二是公司的薪资待遇以及员工培训发展情况。第一类问题显得我们技术专业，技术屌丝的特性被发挥的淋漓尽致，第二类问题很现实，我们不仅要眼前的工资还要考虑未来的发展。

我（阳哥）答：咱们公司的技术团队目前有多少人，都做哪些项目？

人事：公司目前技术团队还在不停的扩建，目前总共有十几名，不过服务端人比较多。现在主要做 p2p 理财类产品，这也是很火的一个趋势。

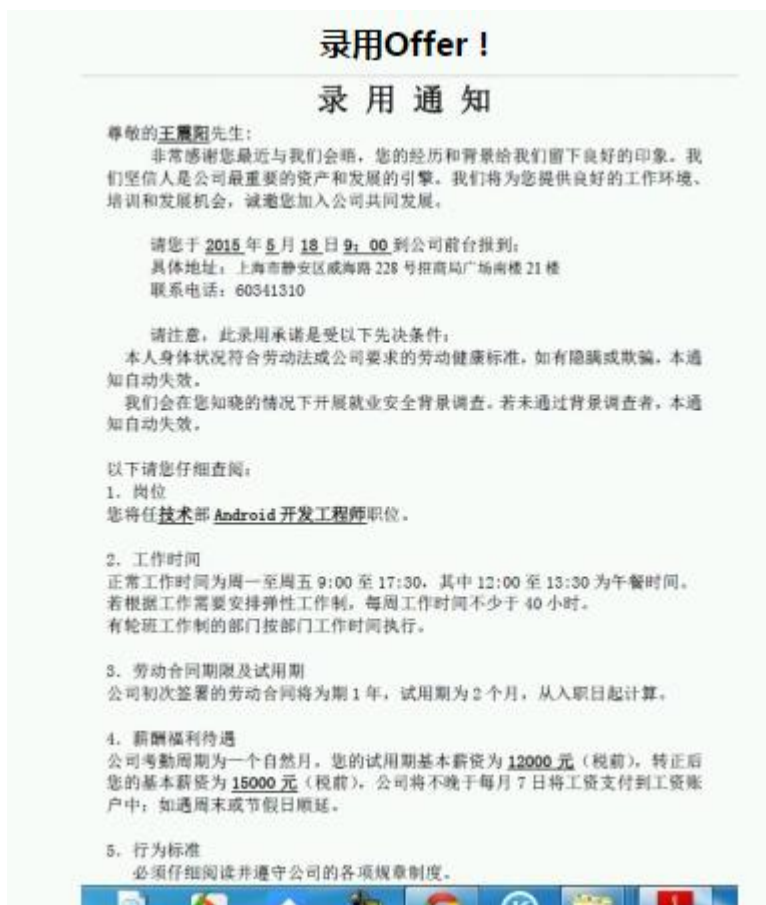
我（阳哥）答：咱们公司给开发人员有定期的培训吗？

人事：有的，公司每年都有拓展培训，拓展培训是针对全体员工的，技术团队的话每周或者每个月可能有技术分享活动，每周一名技术人员进行分享，同时可以获 100 元/次的奖励。

面试吐槽：

阳哥在上海第一天竟然面试了 3 家，上海这么大，每个公司都在不同的区，回到家已经晚上 7 点多了，我的感受只有一个字儿累，躺在床上就起不来了，不过还好今天本来是抱着被面试官“虐”的心态去的，结果没有被“虐”，而且还能旗开得胜。这给了我很多自信，不仅仅是对自己技术的自信，更是对黑马 Android 课程的自信。

最后送大家一句话：没有企业给不了的薪资，只有自己掌握不了的技术。没有自己掌握不了的技术，只有不够努力的自己。



挑战公司 No.4：上海游竞网络科技有限公司---PLU

公司地址：上海市闸北区广中西路 777 弄 99 号江裕大厦 10 层



面试时间：5月13日 14:00 PM.

面试结果：Android 工程师，16K offer+500 元住房补贴+490 元交通和饭补+200 元全勤奖
=17.1k:victory:

面试过程：

16:00 到公司，这家公司在闸北区，比较远，不过办公楼很高大上。

16:00~16:30 填写面试登记表和笔试题。

16:30~17:15 跟面试官进行技术 PK（这家面试流程比较简洁，技术只考察一轮就让人事谈薪资了）。

17:20~17:40 跟人事谈人生谈薪资。

疯狂的笔试+面试记录（前方内容“高能”：funk:，请准备高度精神集中前行）：

1. 笔试

PS : 笔试是在公司前台旁白的桌子上做的。阳哥本来胆子就小，这一下搞的偷拍都没自信了，导致对焦不成功，拍出来的照片很模糊，大家凑合着看吧，我把图片上的试题以文本的形式列出来，如下。

Q : Activity 的生命周期？

PS : 我晕，跟复深蓝的笔试题如此的雷同，难道他们两家公司技术人员互相抄袭的吗？忍住，不笑！

A : onCreate、onStart、onResume、onPause、onStop、onDestroy、onRestart。

Q : Activity 销毁前，如何保存 Activity 的状态？

A : 可以使用 onSaveInstanceState (Bundle) 方法将 Activity 中需要的数据保存起来，当下次重新启动 Activity 的时候在 onCreate (Bundle) 中获取 Bundle 数据。

Q : 请介绍下 Android 中常用的 5 种布局？

A : LinearLayout、RelativeLayout、FrameLayout、RelativeLayout、TableLayout。

Q : 请介绍下 Android 的数据存储方式？

A : SharedPreferences、XML、SQLite、文件系统、内存（如果算的话）。

Q : AIDL 的全称是什么？如何工作？能处理哪些类型的数据？

A :

① Android Interface Definition Language

② AIDL 一般用于远程服务，也就是进程间通信。我们可以分服务端和客户端，服务端声明 AIDL 文件，该文件命名为 xxx.aidl,ADT 会自动将 xxx.aidl 生成代码文件，代码文件提供了 aidl 中接口的实现。客户端如果要使用服务端提供的服务需要将 xxx.aidl 文件放到客户端源代码目录下 然后生成 xxx.java 类，客户端通过 bindService 的形参 ServiceConnection 的 onServiceConnected 获取到 Service 对象，这个对象通过 Stub.asInterface (service) 返回 aidl 的实现类。之后我们就可用调用这个 aidl 的实现类。

③ 基本数据类型都可以，复杂对象也可以，只不过需要实现 Parcelable 接口。

Q：请介绍一下 handler 机制？

A：Android 中 handler 多用于主线程和子线程之间的通信，比如在 Android 中子线程是不允许修改 UI 的，如果修改只能让子线程给主线程通过 handler 发送 message，然后主线程进行修改。Handler 整个机制的实现，还依赖 Looper、Message 两个核心内容。在主线程中 Android 默认给我们创建了 Looper

Q：java 如何调用 c、c++ 语言？

A：java 通过 JNI 调用 C/C++ 代码，在使用的时候首先通过 `System.loadLibrary("xxx")` 将 xxx.so 文件加载到 jvm 中，同时在类中必须对 so 文件中的方法进行生命，格式：`public native void test();`

Q：Android 分几层，分别是什么？

A：四层。Linux Core、Libraries (Android Runtime)、Application Framework、Applications。

Q：final、finally、finalize 的区别？

A：第一个是关键字最终，用 final 修饰的类为最终类，不能被继承，修饰的方法不能被覆写，修饰的变量不能被改变。finally 是异常体系中的关键字，当系统遇到异常是，在进行 trycatch 的时候，finally 代码块里的代码是必须被执行的。finalize 是 Object 类中的方法，当 GC 回收对象时回调的方法。

Q：heap 和 stack 的区别？

A：堆和栈。栈存放对象的引用，堆存放对象实体。堆中的对象是有 jvm 的垃圾回收器负责回收。

2. 第一轮技术面试过程问答精选：

旁白：面试我的是 85 年出生的 Android 组组长（这是后来他们公司人事给我打电话让我入职的时候，跟我说的）。他们公司是做游戏视频直播平台的，因此对 app 的性能要求比较高，他用手机让我看了一段代码。代码（记得不是很清楚了）大概如下：

```
1. public class MainActivity extends Activity {  
2.     @Override
```

```
3.         protected void onCreate(Bundle savedInstanceState) {
4.             super.onCreate(savedInstanceState);
5.             setContentView(R.layout.activity_main);
6.             new Thread(new Runnable() {
7.                 @Override
8.                 public void run() {
9.                     while (true) {
10.                        SystemClock.sleep(1000);
11.                    }
12.                }
13.            }).start();
14.        }
```

面试官问：上面的代码有问题吗？

我（阳哥）答：当然有问题呀，这个 Activity 在启动的时候开启了一个子线程，但是当 Activity 退出的时候该子线程还在运行，并没有停止。子线程运行在进程中的，Activity 退出的时候进程并没有退出，而由前台进程变为后台进程。

面试官问：那应该怎么解决？

我（阳哥）答：我们可以这样，把 while (true) 改成 while(flag)，flag 是一个 boolean 类型的变量，通过改变 flag 的 true 或者 false 来终止子线程的运行，当 Activity 退出时会调用 onDestroy 方法，因此在该方法中我们可以把 flag 设置为 false。

旁白：面试官听了我的答案，从他的表情上来看好像他不是很满意。但是他也没说我说的对不对，而是给我说了他心中的答案，不过他给的答案我当时是虚心接受了，不过到现在我还没明白他说的道理在哪，我写了测试代码也没测试出来啥。

面试官问：这个代码有问题就出在这个 Thread 是一个匿名的，而且没有声明为静态成员变量？

我（阳哥）答：哦，这样子呀，这个我真不知道。

旁白：其实我内心是很想问他为什么呢？但是如果他答不上来就会让他很难堪，如果回答上来了倒没啥问题。从他也不自信的言语中我感觉还是不问他比较好？

面试官问：你对 RTSP 流媒体协议有了解吗？有没有做过手机播放器的应用？

我（阳哥）答：这个知道，自己做过流媒体播放器。我使用过开源的 Vitamio 开源库。

旁白：黑马课程中有一个项目是手机影音。因此这个东西自己是知道的。不懂的人可能以为 RSTP、ffmpeg 这些专业名词都多么的高大上，其实在黑马手机影音中，我们会做一个视频播放器，视频播放器分两种，一种是 Android 自带解码，第二种是万能播放器，所谓万能播放器就是绝大多数流媒体格式都支持。

面试官问：那你知道代码测试怎么测？

旁白：关于测试，阳哥真心不太懂，怎么办？？尽量回答一些自己知道的，同时把测试方面的话题给关闭掉，省得面试官追问我。

我（阳哥）答：您说的是 Monkey Test 吗？

面试官问：恩。。。也算吧？还有其他吗？

我（阳哥）答：我们程序员在写代码的时候都是有规范的，优良的代码规范是规避 bug 重要的一步，同时我们公司每周都有一个代码走读会议，所谓的代码走读，就是开发人员互相看对方的代码，检查代码是否规范以及是否存在 bug。当然也有测试，不过我们的测试都是黑盒测试。

面试官问：哦，代码测试，就是可以通过测试一段代码来分析这段代码有没有问题，性能是否可靠。

我（阳哥）答：对的，Android 自带了 AndroidTest 功能，可以对一个类一个方法进行测试。

面试官问：那好，图片你有处理过吧？

我（阳哥）答：这个肯定有呀，哪个 Android 应用没有图片。比如图片缓存呀，图片缩放呀，图片缓存呀。

旁白：其实自己对图片处理这块儿，还是比较了解的，可惜他也没怎么深问我，没有给我很大的发挥机会。

面试官问：我们现在要做一个视频在线播放的 app，播放的都是游戏直播或者录像视频，iOS 已经做好了，Android 还没有做好，你要是进来的话你就是做视频直播这一块儿。

我（阳哥）答：这个已经做好的能让我看一下吗？

旁白：面试官拿来水果机让我看了一下，主界面就是一个 ListView，每个 ListView 的一行都有大概 4 个视频预览图，点击之后可以播放。

我（阳哥）问：这个切图，设计啥的都做好了吧，剩下的就是编码了其实？

面试官答：是的，后期我们 Android 团队还会不停的夸大，我们的发展重心已经偏向移动端。

旁白：接下来就是跟面试官侃大山了，从天文到地理，从 BAT 到初创型公司，阳哥也不能甘拜下风。

我（阳哥）答：恩，现在是移动互联网时代了，移动端肯定得做好，不管什么样的公司都特别重视移动端。咱们公司现在才开始研发移动端，其实已经稍微有点儿晚了，就得赶紧拼命追赶了。

3. 人事面试：

跟面试官聊完后，技术官对我的评价是，我挺喜欢你，你跟人事好好聊聊吧，哈哈，我肯定会跟人事 MM 好好聊天的。人事问的问题太老套路了，感觉全上海人事问的问题都一样，难道他们都是从同一个地方培训出来的？人事也有培训吗？其实应付人事的问题不难，难的是如何回答人事的问题。答题的方式直接决定了我们在人事那里的印象。我把人事问的几个问题大概列一下。

人事：你为什么离职来上海？

PS：人事问这个问题的主要目的就是看你的动机纯不纯，有没有不良倾向。

我（阳哥）答：主要是两个方面吧，第一个是我的女朋友在这边工作，我年龄也不小了，也到了谈婚论嫁的时候了，我不着急，家里父母亲着急呀，来上海工作的话离女朋友比较近，结婚就比较好办，第二个主要还是。。怎么说呢。。。高大上的说就是为了自己事业更高的发展吧，说的直接点就是上海毕竟

国际化都市，发展机会很多，自己也想来上海工作个 3~5 年挣钱买个房子首付啥的，如果在上海的这几年发展的好可以考虑再公司附近买房，如果自己发展的不好，在上海周边买个房也行。

人事：你女朋友一直在上海，还是刚回来上海？

我（阳哥）答：她上一年来的这边，之前是我大学同学，主要是她家有亲戚帮她找了一份设计院的工作，而我只能靠自己找工作。

人事：你以后打算长期留在上海了吗？

旁白：上面的问题其实是人事在考察你的稳定性，公司可不想招到一个人刚培养出来就走掉了。但是如果你直接肯定的回答：我会一直留在上海，又显得很假。“装”的高境界就是让对方感觉你很不会“装”。

我（阳哥）答：这个有想过，自己其实蛮喜欢上海的，可是计划赶不上变化，来上海的前一年我也从来没想到我会来，因此不能说一辈子都在上海，不过目前这几年是打算在上海好好发展一下。上海的房价、户口政策都是问题，现在不是我选不选择上海，而是上海选择不选择我的问题，呵呵，其实谁都不傻，都想来好地方，关键就看自己有没有本事扎根于此了。

人事：你能承受加班吗？

旁白：其实互联网企业加班是很常见的现象，尤其是像上海这种生活节奏比较快的城市，加班肯定是避免不了的，但是我们也有我们的底线，不能无条件无休止的为公司加班，但是也不能一点儿班都不能加，毕竟有时候公司忙的话，比如新产品上线，新 bug 的解决都是需要加班的，这个时候我们还真的义不容辞的为公司付出，毕竟我们的薪水是公司给的。在公司工作一方面是我们为公司付出，另一方面我也要知道感恩公司。

我（阳哥）答：加班很正常，自己之前的公司也经常加班。不过公司加班一般都不会让员工平白无辜的加班的，我们加班的时候公司会发晚餐补助，打车补助等，如果加班超过半天会计入到我们的存休中，等活不忙的时候我们可以申请休息。因此只要不是高频度的天天加班到凌晨应该没啥问题的。

人事：你的五年规划是啥？

我（阳哥）答：自己还是想往技术方向发展，自己也比较喜欢代码，愿意去研究技术。人往高处走水往低处流，经过自己几年的积累后自己想当一名技术总监或者项目经理类的技术管理类岗位。不过这些都得靠自己的努力以及对机遇的把握情况了。

面试吐槽：

这家企业各个方面的条件真心不错，想拒绝都很难找到合适的理由！我要了 15k 的薪资，人家给了 16k，还有各种诱人的福利。感觉还是游戏类的公司土豪。你以为你要 15k，人家给你 16k 已经很高了吗？请让我把话说完，告诉你啥叫游戏公司。该入职的时候，当然我没有去入职（入职是上午 10:00），上午 10:20 的公司人事给我打电话，问我迟到了还是怎么会儿事儿，我说对不起，感谢贵公司对我的认可，自己已经有了新的选择。把他们给拒绝了。下午 1 点多的时候，我正在吃饭，结果这家公司的技术官又给我打电话来了，在电话里跟我聊了很久，他说如果再给你多几 k 你会考虑过来上班呢？我哑巴了，自己从来就没有享受过这么好的待遇，受宠若惊的阳哥无言以对！自己的内心像打翻了五味瓶一样，这种感觉估计也没有几个人理解。后来还是被我给拒绝了，内心在流泪！

这篇文章阳哥已经写到尾声了，但是想给大家说的话却依然很多很多。

亲爱的黑马同学们：

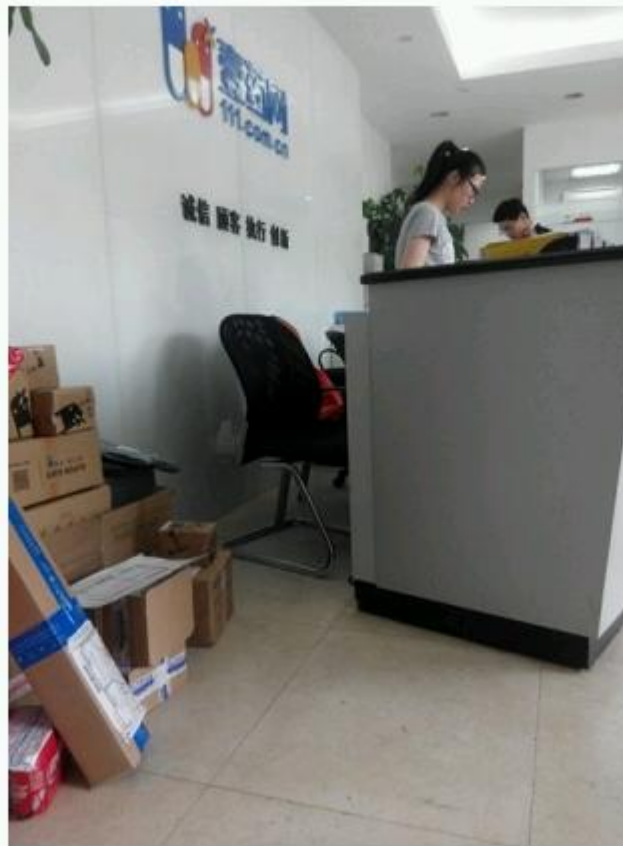
如果你还在撸着代码，看着视频，再苦再难，请你一定要坚持，今天你付出的一点一滴都将会在明天变成千千万万倍的回报。

加油&坚持！

挑战公司 No.5：一号店旗下壹药网

公司地址：上海市浦东新区碧波路 572 弄 115 号 10 幢

(偷拍前台妹子~天热，穿的少其实是正常的，可惜桌子挡住了该看的，哦，不该看的:lol)



面试时间：5月14日 10:00 AM.

面试结果：Android 工程师，15K offer&14个月薪资:victory:

面试过程：

10:10 到公司，这就是传说中的一号店！，公司总共三栋独栋楼，两个在装修，一个在用，因此工位紧张，导致我面试都是在前台所在的大厅里进行的。

10:20~10:50 填写面试登记表和技术官面试。

10:50~11:10 跟研发部 Leader 面试。

11:10~11:30 跟人事谈薪资以及入职事宜。

疯狂的笔试+面试记录（前方内容“高能” :funk:，请准备高度精神集中前行）：

PS : 笔试跟面试是一起的，一个技术官拿着一张正反面都写满题的 A4 纸，从第一题到最后一题，他指一题我回答一题，我回答一题，他指下一题。配合的天意无缝，哈哈~~~

1. 笔试

1) java 基础部分

Q : Java 面向对象有哪些特征？

A : 封装、继承、多态。

Q : `short s1=1;s1=s1+1` 有什么错？

`short s1=1;s1+=1`有什么错？

A : 第一个是有错的，`short` 在内存中占 2 个字节，而整数 1 默认为 `int` 型占 4 个字节，`s1+1` 其实这个时候就向上转型为 `int` 类型了，因此第一行代码必须强转才行。第二个之所以可以是以为这句话翻译过来就是 `s1++`，也就是 `short` 类型的数据自身加增 1，因此不会有问题。

Q : 静态成员类、非静态成员类有什么区别？什么是匿名内部类？

A : 静态成员类相当于外部类的静态成员，是外部类在加载的时候进行初始化，非静态成员类相当于外部类的普通成员，当外部类创建对象的时候才会初始化。匿名内部一般都是在方法里面直接通过 `new ClassName(){};` 形式的类。比如我们 `new Thread (new Runnable(){}).start()`；就用到了匿名内部类。

Q : `abstract class` 和 `interface` 有什么区别？

A : 前者是抽象类，可以有抽象方法，也可以没有。后者是接口，只能有抽象方法。他们都不能创建对象，需要被继承。

Q : `ArrayList` 是不是线程安全的？如果不是，如何是 `ArrayList` 成为线程安全的？

A : 不安全的。可以使用 `Collections.synchronizedList(list)` 将 `list` 变为线程安全的。

Q : 是否可以继承 `String` 类？

A : 不可以，因为 String 类是 final 类。为啥不解释了吧。

Q : 以下两条语句返回值为 true 的有：

A : "yiyaowang"=="yiyaowang";

B: "yiyaowang".equals(new String("yiyaowang"));

A : 第一个返回 true，都是字符串常量，存储在字符串常量池中，且只有一份。第二个返回 true，用 equals 比较的是字符串内容。

Q : 当一个对象被当做参数传递到一个方法后，此方法可以改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？

A : java 中只有值传递，没有引用传递。这里的引用本身就是值，传递的是引用这个值。

Q : 定义类 A 和类 B 如下：

```
1. class A{
2.     int a =1;
3.     double d = 2.0;
4.     void show(){
5.         System.out.println("Class A:a="+a+"\td="+d);
6.     }
7. }
8. class B extends A{
9.     float a = 3.0f;
10.    String d = "Hello World!";
11.    void show(){
12.        super.show();
13.        System.out.println("Class B:a="+a+"\td="+d);
14.    }
15. }
```

(1)若在应用程序的 main 方法中有以下语句：

A a = new A();

a.show();

则输出结果是？

(2)若在应用程序的 main 方法中定义类 B 的对象 b:

```
A b = new B();
```

```
b.show();
```

则输出结果是？

A : 第一个是 ClassA : a=1 d=2.0

第二个是 ClassA : a =1 d=2.0

ClassB : a=3.0 d=Hello World!

Q : heap 和 stack 有什么区别？

A : 堆和栈。栈存放对象的引用，堆存放对象实体。堆中的对象是有 jvm 的垃圾回收器负责回收。

Q : 请描述下 JVM 加载 class 文件的原理机制。

A : JVM 加载 class 是动态性的，也就是当“需要”的时候才会加载，这也是为节约 JVM 内存来考虑的。同时 JVM 的类加载是父类委托机制，这个机制简单来讲，就是“类装载机有载入类的需求时，会先请示其 Parent 使用其搜索路径帮忙载入，如果 Parent 找不到，那么才由自己依照自己的搜索路径搜索类”。

2) Android 基础部分

Q : 如何适配不同屏幕分辨率的机型？

A : 屏幕适配方式就多了去了。Android 系统本身提供了很多适配方法，比如存放图片资源的 drawable 目录根据不同分辨率的手机提供了 drawable-hdpi、drawable-ldpi、drawable-mdpi、drawable-xhdpi 等多中目录。我们只需把适应不同分辨率的多套图片分别放到对应的目录中即可。Android 的 layout、values 目录也提供了类似 drawable 的适配功能。

但是在开发中，不可能针对不同的手机分辨率提供多种图片资源，这太耗费资源了。我们一般在写控件宽高的时候都会用 dp 单位取代 pix 单位。因为 dp 是一个相对单位，pix 是绝对单位，使用 dp 替代 pix 也可以解决很多适配问题。dp 跟 pix 之间可以通过公式进行转换。

Q : View 和 ViewGroup 的关系是什么？View 的绘制过程（主要方法）有哪些？

A : ViewGroup 继承了 View。onMeasure、onLayout、onDraw。

Q : Activity 和 Task 的区别及启动模式有哪些？

A : Activity 运行 Task 中。Activity 有四种启动模式。standard、singleTop、singleTask、singleInstance。
standard:默认的启动模式，多个 Activity 位于同一 Task 中。singleTop，顾名思义就是 Task 栈顶只能有一个相同的 Activity，singleTask 就是一个 Task 中只有一个 Activity，singleInstance 就是一个 Activity 独享一个 Task。

PS : 关于 Activity 的四种启动模式其实还有更详细的说法，我在上面就简单介绍一下了，如果面试官需要问的更详细再往深处介绍就行了。

Q : 如何注册 BroadcastReceiver 和 Service？Service 有什么特征，哪些情况会用到 Service？

A : 都可以通过 AndroidManifest.xml 进行静态注册。不过 BroadcastReceiver 可以在代码中通过 registerReceiver 方法来注册。Service 运行在后台进程中，一般需要在后台一直运行的任务会让 Service 来完成。比如我们的 telephoneService、locationService 等等。

Q : Android 有哪些安全机制？

A : 权限机制。我们的应用只要涉及到了用户的隐私、网络都需要在 AndroidManifest.xml 中进行声明，这样用户在安装的时候可以根据你申请的权限进行判断是否允许应用的某些行为。

Q : Handler 机制的原理，内部是如何实现的？

A : Android 中 handler 多用于主线程和子线程之间的通信，比如在 Android 中子线程是不允许修改 UI 的，如果修改只能让子线程给主线程通过 handler 发送 message，然后主线程进行修改。Handler 整个机制的实现，还依赖 Looper、Message 两个核心内容。在主线程中 Android 默认给我们创建了 Looper,当我们通过 handler.sendMessage()后,该消息被添加到 MessageQueue 中,Looper.looper

中有个 while(true)的循环不停的从消息队列中取消息。取消息的过程是线程阻塞的，这样不至于在没有消息的时候过多的耗费 CPU 资源。

Q：Thread 和 AsyncTask 的区别是什么？

A：AsyncTask 是封装好的线程池，比起 Thread+Handler 的方式，AsyncTask 在操作 UI 线程上更方便，因为 onPreExecute()、onPostExecute()及更新 UI 方法 onProgressUpdate()均运行在主线程中，这样就不用 Handler 发消息处理了；

Q：说说 MVC 模式的原理，在 Android 中的运用。

A：MVC 是 Model、View、Controller 三部分组成的。其中 View 主要由 xml 布局文件，或者用代码编写动态布局来体现。Model 是数据模型，其实类似 javabean，不过这些 JavaBean 封装了对数据库、网络等的操作。Controller 一般由 Activity 负责，它根据用户的输入，控制用户界面数据的显示及更新 model 对象的状态，它通过控制 View 和 Model 跟用户进行交互。

Q：如何加载 ndk 库？如何在 jni 中注册 native 函数，有几种注册方式？

A：通过 System.loadLibrary("xxx")进行加载。其实 native 有几种注册方式，自己当时并不知道，自己只知道一种注册方法，就是首先根据 native 方法名，生成 Java_com_xxx_MethodName(xxx,xxx); 当然这个 c/c++源码文件中需要引入 jni.h，然后把这个 c/c++源码编译成 so 文件。

自己后来百度了一下，网上有人数还有一种注册方式是动态注册，我就把关于动态注册的东西直接拷贝过来：

JNI 允许你提供一个函数映射表，注册给 Java 虚拟机，这样 Jvm 就可以用函数映射表来调用相应的函数，就可以不必通过函数名来查找需要调用的函数了。Java 与 JNI 通过 JNINativeMethod 的结构来建立联系，它在 jni.h 中被定义，其结构内容如下：

```
1. typedef struct {  
2.     const char* name; //Java 中函数的名字  
3.     const char* signature; //用字符串描述的函数的参数和返回值  
4.     void* fnPtr; //指向 C 函数的函数指针
```

```
5. } JNINativeMethod;
```

第一个变量 name 是 Java 中函数的名字。

第二个变量 signature，用字符串是描述了函数的参数和返回值

第三个变量 fnPtr 是函数指针，指向 C 函数。

当 java 通过 System.loadLibrary 加载完 JNI 动态库后，紧接着会查找一个 JNI_OnLoad 的函数，如果有，就调用它，

而动态注册的工作就是在这里完成的。

1)JNI_OnLoad()函数

JNI_OnLoad()函数在 VM 执行 System.loadLibrary(xxx)函数时被调用，它有两个重要的作用：

指定 JNI 版本：告诉 VM 该组件使用那一个 JNI 版本(若未提供 JNI_OnLoad()函数，VM 会默认该使用最老的 JNI 1.1 版)，如果要使用新版本的 JNI，

例如 JNI 1.4 版，则必须由 JNI_OnLoad()函数返回常量 JNI_VERSION_1_4(该常量定义在 jni.h 中) 来告知 VM。

初始化设定，当 VM 执行到 System.loadLibrary()函数时，会立即先呼叫 JNI_OnLoad()方法，因此在该方法中进行各种资源的初始化操作很恰当，

2)RegisterNatives

RegisterNatives 在 AndroidRunTime 里定义

syntax:

```
jint RegisterNatives(jclass clazz, const JNINativeMethod* methods,jint nMethods)
```

Q：App 在什么情况下会出现内存泄露？如何避免这些情况？

A：造成内存泄露的可能性有很多，我说几种吧，1) 资源未及时释放，比如引用的 io 流资源、网络资源、数据库游标 Cursor 等没有释放 2) 注册的监听器、广播等未及时取消 3) 集合对象没有及时清理 4) 不良代码

避免上述问题，主要还看程序员知识掌握的程度和编码经验的多少，但是从技术角度考虑我们需要注意一些细节，比如，重复使用的资源可以考虑使用缓存技术、池技术。使用的任何资源都记得关闭或者异常处理，保证在恶劣的情况下也能使资源得到释放。对于图片的操作要注意缓存的使用，同时要记住对图片对象进行及时的回收。使用 ListView 的时候，尽量让 convertView 得到复用。

3) 逻辑思考

Q: 你让工人为你工作 7 天，给你工人的回报是一根金条，金条评分成 7 段，你必须在每天结束时给他们一段金条，如果只许你两次把金条弄断，如何给你的工人付费？

PS: 因为上面的问题自己回答的都比较溜，所有这个逻辑思考题，面试官直接说我想着你也会就不做吧。哈哈~面试官已经放弃继续考察我了，嗨皮~~~然后他去找老大去了，趁机我赶紧把笔试题给偷拍了下来（是不是阳哥胆子越来越大呀？！），这样才能大家看到真实的笔试题是什么样子滴。

哦，对了，上面的逻辑思考题我面试的时候是免试的，亲爱的黑马童鞋们你们知道答案吗？

2. 第二轮面试过程问答精选：

旁白: 面试我的应该是开发组部门的老大。胖胖的，黑黑的，矮矮的，从开始面试到结束，一直面带微笑。因为之前那个技术官已经面试过我的技术了，因此他也没问我技术，就是跟我瞎聊了一些关于 Android 方面的东西。因此详细内容这里就省略了，请大家直接进入下一关~duang~

3. 人事面试：

人事面试的问题，其实我在 V4 版本中也说过，基本所有的人事问的问题都大同小异，跟这个人事聊的唯一有趣的就是，由于公司正在装修，找不到面试我的办公地点，然后她竟然把我带到公司旁边的凉亭子下面，我们两个并排坐着，就聊起来了，知道的知道我们是在面试，不知道的估计还以为我们在谈恋爱呢。: #

面试吐槽：

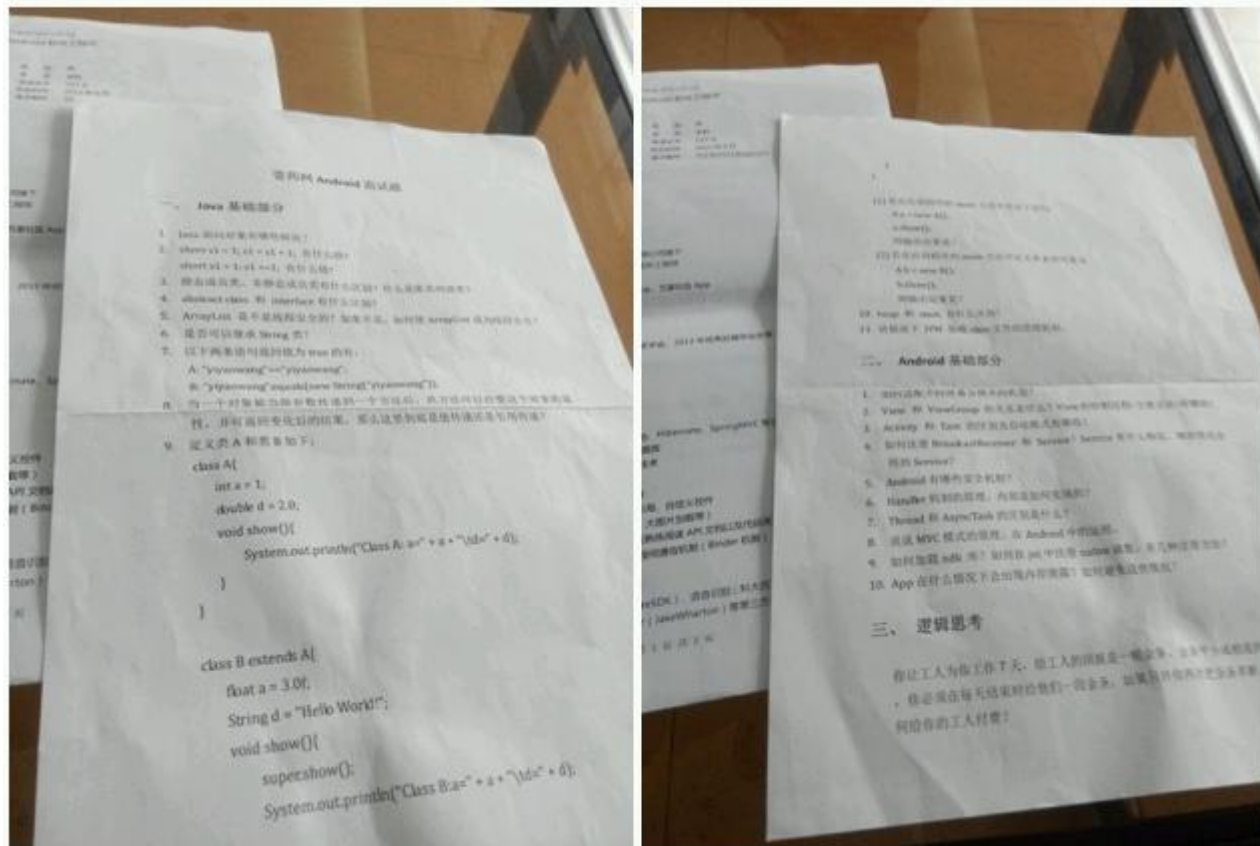
在前台我填写面试登记表的时候写的期望薪资是 15k，他们也没有压低我工资，我知道我要少了。不过，没办法，谁让阳哥不懂行情，在投递简历的时候都是写着期望 15k 呢！想写个 16、17、18 都不好意思了，因此大家以后找工作可别学阳哥这么傻，能多要一定多要。

跟大家分享一个励志语句以结束本篇文章吧：

相信梦想是价值的源泉，相信眼光决定未来的一切，相信成功的信念比成功本身更重要，相信人生有挫折没有失败，相信生命的质量来自决不妥协的信念。



笔试题（在前台眼皮底下偷拍~）



挑战公司 No.6：聚信租赁

公司地址：上海市徐汇区漕溪北路 398 号汇智大厦 28 楼



面试时间：5月14日 14:00 PM.

面试结果：Android 工程师，16K + 入职送 iPhone 6 + 每年出国旅游一次 + 补充住房公积金 + 至少 14

薪:victory:

面试过程：

14:00 到公司前台，领了一大堆资料，然后让我在一间办公室做题。

14:00~14:30 填写面试登记表和性格测试。

14:30~15:20 两个技术官坐在我对面，同时面试我。

15:20~15:30 人事妹子跟我单聊公司薪资福利以及入职事宜。

疯狂的笔试+面试记录（前方内容“高能”：funk:，请准备高度精神集中前行）：

1. 笔试

这家比较奇怪，是给了 6 页的笔试题，打开一看全是性格测试题。性格测试题只拍了一张图片，在该篇文章的结果。估计大多数童鞋都想遇到阳哥这样的狗屎运吧;P~阳哥的性格绝对没问题的，这个我可以保证哈~:P

2. 技术面试

竟然同时来了两个技术官面我，一大一小，一高一矮，并排坐在一起，一对二，好吧，阳哥还是首次遇到 1:2 的阵容，这下可有好戏看了。**PS**：阳哥是个没有见过世面的人，面试的时候上个 31 层高的楼都兴奋的不得了，上去以后，发现上错楼了~糗事说多了都是泪:#，要不是阳哥有着过五关斩六将身经百战的成功经验，估计要吓尿:loveliness:。好吧开始吧，阳哥命中注定必有次劫，看来是躲不过了。

Q：你做一个自我介绍吧？

旁白：其实遇到好几家面试官都让我做自我介绍了，该如何自我介绍阳哥估计都会背了，好玩（**恶心**）的是在万达信息面试，面试了 3 个技术官，每个人都分别让我做了自我介绍，尼玛，他们 3 个就不会沟通一下要问我啥吗，一个问题至于问我 3 遍吗~:funk:阳哥是敢怒不敢言，毕竟在人家的地盘。

PS：自我介绍的内容就不说了，每个人都是独特的，我就跟大家说一下应该如何自我介绍吧。

一个优良的自我介绍会给面试官留下深刻的印象，大部分情况下，所谓的面试好坏其实看的就是你给面试官留下的印象怎么样了，我们用俗语叫感觉。

自我介绍应该分以下几个部分，按照一定的逻辑连贯起来。如果连贯不起来，或者不够熟练一定在台下多背几遍，多讲几遍，但是面试的时候不要说的跟背过似的，高境界就是让面试官感觉你是临场发挥的，却又比背的都好。

1) 个人基本信息（姓名、年龄、老家、居住地等）

2) 自己来自哪里（工作地点），是干什么的（给自己一个清晰的定位，比如：我是一名 Android 开发工程师），担任过什么职务、做过什么样的项目

3) 自己为何来贵公司面试

4) 最后祝愿 (希望能得到贵公司的认可等等, 不用太多, 一两句话就 ok)

Q: 介绍一下你做过的项目吧?

PS: 黑马那么多项目, 随便准备 3 个就 ok 了。

介绍项目大概的思路如下:

1) 这个项目是干什么的 (比如是一个类似网易新闻的地方新闻客户端, 或者类似美团的 o2o, 或者类似豌豆荚的一个应用市场, 或者类似淘宝的购物平台)? 解释就是拿一个市场上耳熟能详的应用跟自己的应用做类比, 省的面试官听的云里雾里的。

2) 自己负责了哪些模块 (功能) 的职责 (比如负责系统的架构, 核心代码的编写, xx 功能模块的开发等等)

3) 自己在这个项目中担当的责任 (比如, 这个项目是自己独立开发的, 这个项目是和另外一个同事一起架构一起开发的, 这个项目是自己负责了几个核心模块)

4) 项目中都用到了哪些技术

5) 从项目中学到了哪些东西 (可以从技术方向和业务两个方向入手)

旁白: 面试官问的很多技术性问题跟之前问的都大同小异, 因此这里只给出有特色且技术含量高的。阳哥正在写面试宝典, 该宝典核心内容针对的还是技术问题, 阳哥会从 javase 基础到 javase 高级, 从 Android 基础到 Android 高级以及到 Android 项目依次展开分析, 其次也会写一些常见的非技术性问题, 敬请期待~

Q: ①在 ListView 的优化中, 我们为何使用 ConvertView? ②为何使用 ViewHolder? ③你认为哪个更能解决问题? ④你认为 view.inflate 和 view.findViewById 哪个更耗时, 为什么? ⑤如果这两个 AP 让你重新写, 你怎么写?

PS :上面的问题，阳哥认为是面试以来遇到很难的一个，也是很有技术含量的一道题。前一半问题还好回答，最后一个问题真的需要发挥想象了。

A :①使用 ConvertView 可以实现对 view 的复用，这样大大节约了每次创建对象的时间，提升了 ListView 的显示效率。②使用 ViewHolder 作为内部类，可以将 view 的子控件封装在 ViewHolder 类中，然后通过 View.setTag(ViewHolder)将 view 和 ViewHolder 进行绑定，这样我们就不用每次都调用 view 的 findViewById(id)方法来查找控件。③使用 ConvertView 解决了一大部分问题，使用 ViewHolder 实现了控件换时间的问题，因为给 View 对象设置一个 Tag 本身就是占用内存的，因此 ViewHolder 的使用还是需要区分不同的应用场景的，没有绝对的好与不好。如果内存足够需要高效则 ViewHolder 建议使用，否则不建议使用。④当然是 view.inflate 耗时，这个函数完成的功能是把 xml 布局文件通过 pullParser 的形式给解析到内存中，需要 io，需要递归子节点。⑤我其实还不太相信我写出来的代码比 Google 官方写的好，如果让我写的话我可能会这样考虑，当用户在使用 view.inflate 的时候将多个 id 作为数组添加到形参中，这样在初始化 view 的使用我就可以给这个 view 直接调用 setTag 方法绑定需要的子控件。不过这个原生方法其实也应该保留共不同的需求使用。

PS :技术面试时间并不长，我回答了几个之后，他们两个大眼瞪小眼，A 看看 B 问：你还有什么问的吗？B 说我没有，你还有吗？A 说我也没了。那行，接下来，他们就让我等人事了。

3. 人事面试：

人事问的问题都差不多，我在上一篇也说过，这里就不说人事的问题了。唯一要给大家爆料的就是人事给我讲的他们公司的福利待遇，可以用土豪不差钱形容:)~

人事说：入职后转正即送 iPhone 6 一部。

我（阳哥）说：我是做 Android 开发的，给我 iPhone 6 干嘛

旁白：你认为阳哥真不想要 iPhone 6 吗？为了显示咱的高度敬业精神，毕竟咱是做 Android 开发的，更需要的是 Android 手机，决不能像 iPhone 低腰（ni dao shi gei wo ya）！

人事说：我们 Android 开发人员也会额外配 Android 机的，iPhone6 可以生活用。

旁白：我去~这就是没见过世面的阳哥，当时的表情。



人事说：转正满一年，每年都有一次出国旅游。

我（阳哥）说：哦，咱们公司还挺不错的嘛！

旁白：不知说啥好了，只能用表情来形容了，阳哥至今没出过国，国外啥个样子嘛，



人事说：我们一般一年更少发 14 薪，都是介于 14~16 薪之间，具体发多少要看个人平时的一个 KPI 考核了。

我（阳哥）说：哦，这样呀，还行吧。



面试吐槽：

在家公司的人事跟我算了基本工资 16k+各种补助 1k=17k+。每年出国旅游，入职送 iPhone，补充住房公积金，好吧，阳哥受宠若惊了。自己回来会百度了一下，终于搞明白这家公司为何这么奢侈！不多说，看图吧。

聚信租赁获13亿银团贷款 IPO仍在排队中

2015-01-08 17:26:42 来源: 网易财经



16

网易财经1月8日讯 今日，聚信国际租赁股份有限公司（以下简称“聚信租赁”）在上海召开发布会，正式与农行上海静安支行、建行上海分行、交行上海分行以及南京银行上海分行共同签订13亿元银团贷款协议。

去年10月17日聚信租赁正式发布招股说明书预披露，目前公司仍处于排队审核，等待上市中。公告称，公司拟赴上海证券交易所上市，拟发行新股10000万股，包括公开发行的新股及公司股东公开发售的股份。发行后总股本为44285.71万股，预计融资额12.5亿元。募集资金在扣除发行费用后，将用于充实发行人净资产，补充自有资金以支持未来业务发展。

据介绍，本次银团贷款项目融资由农行上海静安支行作为牵头行，建行上海分行和交行

跟人事面试的时候，人事说他们在上海总部目前有大概 100 多名员工，阳哥不懂经济，不知道 100 多人的公司能融资 13 个亿是个什么样的概念？这再次验证了阳哥的那句千古流传的话语，只要技术好，公司不差钱！

应聘登记表

应聘部门及职位：_____ 可到职日期：_____

应聘人概况：_____

本人所有资料均为本人信息，我们将为您严格保密，所有资料均为必填。

性别：	出生日期：	民族：
<input type="checkbox"/> 未婚 <input type="checkbox"/> 已婚 <input type="checkbox"/> 其他	身份证号码：	
户口所在地：	籍贯：	E-mail 地址：
家庭住址：	邮编：	
户口地址：	邮编：	
现所在地：		
固定电话：	紧急联系人：	
手机：	紧急联系人电话：	
特长：		
语种和读写程度：		
使用的软件：		

性格测试

本问卷中的所有问题都反映了人们的日常生活，请根据您的实际情况回答。

请仔细阅读问题，并选择一个答案。您的答案将用于评估您的性格特征。

请根据您的实际情况回答，不要有任何顾虑。您的答案将用于评估您的性格特征。

请根据您的实际情况回答，不要有任何顾虑。您的答案将用于评估您的性格特征。

1. 我倾向于从何处得到力量？

(E) 他人。

(I) 我自己的思想。

2. 当我参加一个社交聚会时，我倾向于有更多的力气：

(E) 在夜色很浓时，一旦我开始投入，也许就是最晚离开的那个。

(I) 在夜晚开始的时候，我就疲倦了并且想回家。

3. 下列哪一种听起来比较吸引人？

(E) 与我的情人到有很多人且社交活动频繁的地方。

(I) 待在家中与我的情人做一些特别的事情，例如说观赏一部有趣的录像带并享用带食物。

4. 在约会中，我通常：

(E) 整体来说会健谈的。

(I) 较安静并保留，直到我觉得舒服。

5. 过去，我倾向遇见我大部分情人：

(E) 在宴会中、夜总会、工作上、休闲活动中、会议上、或当朋友介绍我给过私人的方式，例如个人广告、录像约会、或是由亲密的朋友和家人

他人面试心得

一、一个五年 Android 开发者百度、阿里、聚美、映客的面试心经

整理转原文来自：

<http://gdky005.com/2016/07/08/%E4%B8%80%E4%B8%AA%E4%BA%94%E5%B9%B4Android%E5%BC%80%E5%8F%91%E8%80%85%E7%99%BE%E5%BA%A6%E3%80%81%E9%98%BF%E9%87%8C%E3%80%81%E8%81%9A%E7%BE%8E%E3%80%81%E6%98%A0%E5%AE%A2%E7%9A%84%E9%9D%A2%E8%AF%95%E5%BF%83%E7%BB%8F/?winzoom=1>

花絮

先简单说说我最近的面试经历吧。面试的公司很多，其中有让我热血沸腾的经历，也有让我感到失望到无助的经历，我将这些体会都记录下来，细想之后很值得，面了这么多公司，要是最后什么也没有留下来，那就太浪费了。至少对于我来说有些东西在整理总结之后才能得到一个肯定的答案。希望这些能对即将换工作或者打算看看机会的你有一些帮助。

为何离职？

先从我的换工作的动机开始说吧。

公司裁员的时候老大说：『你就留下好好干吧，以后不管公司怎么分股票、期权，肯定少不了你』。我非常信任我的老大，跟着老大一起工作，感觉是一种享受。

但是没想到裁员后，公司内部大动荡，主业务线从客户端 A 业务线转移到另外的 B 业务线上。我主要负责 A 客户端的架构，这下可真闲下来了。B 业务线那边的业务量还是很忙的，没时间配合我做一些架构上的事情。于是我每天就看看资料，补充点能量。

呆了几天后，就后悔当初没有拿 N+1 走，有一种被老大忽悠的感觉。因为公司接下来的操作让我很是不爽，先是晚上打车不能超过 30，然后福利大减，瞬间没有工作的心情了。再过了一两周后公司宣布新一轮融资成功，可惜只融到了 2 千多万美元（按照预期应会更高），然后接着招新人。

我特么无语了，站在公司的角度是没有任何问题的，可以节省开销，也可以容纳新鲜血液。但是我作为一个老员工，心寒，走的员工都拿到了 N+1，我们这些老员工什么也没有得到，反而福利大减，伤人啊！现在即使我想走，什么也得不到，一种莫名的恼火涌上心头（只怪本人经历尚浅，看不清一些大的趋势，还是老鸟们聪明，拿钱走人，然后换一个新工作，好不自在啊）。

不过理智分析一些这样确实有好处，可以给自己留很多的时间来选择更好的公司。就如此刻的我一样，在公司悠闲的上着班，骑驴找马，遇到合适的，可以立刻走。其实细想一下，如果我当时拿了 N+1 走了后，可能会迫切的需要一份合适的工作，然后迅速入职。至于新公司怎么样，还真不敢确定。

已经动了想走的心，意味着再也不可能在这里很安分的待下去了。

面试分级

于是我决定开始投递简历（世界那么大，我想去外面的世界看看）。这次看机会与往常不同，我决定好好准备一番，然后开始投递简历，主要渠道是“X 钩”，辅助渠道是猎头。

这次看机会我将所有公司分为三类：

A 类：BAT 公司，非常靠谱，各项待遇都是很优厚的

B 类：一些知名的互联网公司（基本都在 C 轮以上），基本很靠谱，该有的都少不了

C 类：就是那些正在招聘的公司，没啥名气，虽然钱多但是事也多。靠不靠谱真还不知道，只能碰运气

基础知识不可少

以前我基本都是直接去面试，总以为 Android 工作好几年了，出去面试基本没啥问题，因此带着那份傲娇的自信总是碰壁，尤其遇到很多基础性的问题，一时真不知道怎么回答？还有一些问题之前都记得很准确，但是在面试官问的时候，就一个大写的懵逼表情。

在我出去面试之前，我已经把《大话数据结构》基本看完了（想想我之前的生活，每天早上七点多起床，然后看几页，洗漱完就去公司）。虽然没怎么记住，但是遇到这些相关问题，还是能很容易回答出来的。因为有了以前的教训，而且这次我也是很认真的准备了好久（可以说蓄谋已久啦，我心里其实很明白互联网公司可能存在很多风险，尤其是没有盈利的公司，唯有技术这东西必须牢牢掌握住，才能立于不败之地），因此我准备把 Java 基础巩固下，但是手头没啥合适的书籍和资料。

还好民间有很多厉害的开发者的，他们不以盈利为目的，只为完成某种需求，开发一款 app，然后发布到应用市场，给需要的人。于是我就找到一个“Java 面试训练”的 App，下载量还可以，就安装到手机上，开启刷题模式，应该刷了 10 来天吧（都是在上班，下班时间看一点，虽然时间比较零散，但是这样记得最深刻）。在之后的面试中，基本很少遇见一些奇葩的 java 基础。

ps:

这里不得不提一件事，那就是从 app 崛起的那一刻起，就有很多的中间商，一个小作坊的屋子里有很多电脑或者不知名的设备，屋子里慢慢的数据线，犹如蜘蛛网一样连接着很多设备，做着一些神秘的事情。不用我说你们应该也知道他们做着一些很肮脏的事情，我就不细说干什么了，简单举个例子：这群人的老大看中某个市场上某款游戏非常火爆，或者 app 特别的火，于是通过反编译等技术修改这些 app，然后重新打包上线到一些不是很知名的 app 渠道或者小型应用时长，还有一些论坛，一旦有用户下载，就会在 app 中弹出广告，在游戏中做各种充值操作，甚至在你无意间点到一个按钮就会自动扣除你的话费。这是前几年干的事情，新闻中也披露了很多，这里只能说监管不力。

但是随后各个公司都意识到这样的安全问题于是有了 app 加固的技术，无法修改 app，即便修改了，但是也运行不起来，所以一定要注重安全性问题。

刚踏入架构师之路的经历

这次我给自己的规划是做一个架构师，但是我深知架构师可不是闹着玩的，必须要有很强的一面，因此我在简历里面写的只是“架构师方向”。我在 K 公司做得是架构师方向，因此我觉得有必要朝着这个方向发力，虽然现在不是很厉害，但是坚持一两年后，即使不是非常厉害，但是也距离非常厉害很近（这里使用了《孙子兵法》的一句：“求其上，得其中；求其中，得其下；求其下，必败。”）。

这个想法来源于在 K 公司我第一任 leader 曾经跟我说过的话：『对于新东西，如果你觉得掌握了，但是不应用到项目里面来，是没有什么意义的，时间长了还是会忘记的。』我很庆幸我有一个好老大（我是属于双领导型的，K 公司 A 项目的负责人是我的 leader，但是我的直接汇报对象是 K 公司的副技术总监，下文就成为老大），用他的话来说就是经常踢着我的屁股走。

当我在网上了解到很多实用的新技术时，跟他随意吐露一句话，他就能非常用心的倾听我的想法，并鼓励我将这些东西带入到项目中来。从那以后我就开始看很多新技术，感觉合适的会引进到我们的项目中。从之后的证明中来看，是非常有价值的。

曾经遇到的情况是这样的：当我刚进入 K 公司后，打杂一个多月，就被关到了小黑屋（呜呜呜，好可怕的小黑屋，996 的制度）。然后才开始正常的架构师之路，第一步就是统一开发环境，在我来公司后，我发现公司的 android 同

事用的开发工具种类真是繁多啊，神马 Eclipse、IntelliJ IDEA、Android Studio、Windows、Ubuntu、Mac。刚进公司的时候我曾经用鄙夷的眼神看过那些 Eclipse 的童鞋，真是无力吐槽了。于是我给 老大说：『咱们的开发环境最好统一起来，现在各式各样的工具，弄个东西真费劲。』于是老大二话不说，就在群里跟大家吼，都务必切换到 Android Studio（以下简称 AS），由我来监督并执行。于是我拿着鸡毛当令箭，给大伙把地址什么的都找好，发到群里去，让他们自己下载（后期我们就搭建了 ftp 服务器将这些常用的工具都放在里面，省的再去下载了）。翻墙工具我使用 goagent（不怎么稳定），给其他人分享也太费劲了，因此让他们自己搞定。老大自己有一个 VPS，于是给大伙共享后，环境基本就统一了。

ps:

期间有一个小插曲：

一个年龄比我大的同事在用 Eclipse，在我推广我的 AS 时，他说比较忙，没时间弄。我就急了，因为我刚到公司不久，老大分配给我的任务，推行不下去，这可不行啊，没说几句吵起来了。最后我也知道不能太着急，但是已经吵了，关系肯定不咋样，老大当时开会去了，我知道自己太心虚了，因此主动给老大承认错误，说我和那谁谁吵架了，因为他不用 AS。最后在老大的劝说下，这个人就勉强切换到 AS 了。

其实这个人就是我之后的新 Leader，每每想到这里我就全身发冷汗，Leader 要虐你，你还能有好活路么？还好这个 Leader 人比较好，人也比较大气会处事，不怎么跟我计较。我已经对着佛祖忏悔了 N 多次。

第一天面试

我用“拉钩”开始捡一些不怎么有名的 C 类公司投递，很快就收到了很多的面试邀请。

首次面试——国外输入法

记得当时去的第一家公司是做国外做输入法的，做的还不错。从外面能看见一栋略微有点老的大厦，办公环境很一般。

进去后很巧的是遇见了一个熟人，第一位面试官竟然认识我之前在 X 游的一个同事，然后我们就聊开了，他也没怎么难为我，就问了我几个很简单的问题，例如：handler 的原理，多线程。我按照记忆中的样子说给他听，然后就

第一关就轻松过了。

等了一会，另外一个面试官进来了，问了一长串问题，基本就是 Android 的相关的基础，然后第二个又轻松过了。

等到第三关的时候，一个年龄稍微大的人进来了，很容易能看出，这个人应是该技术团队的负责人，问了一些工作经历后，然后问了一个最让我印象深刻的问题是：『你了解过 Android 上的黑科技么？比如 Android 5.0 之上有一个辅助功能，如果用户开启后，就能像豌豆荚那样自动安装 app，等同于拥有了 root 权限，但是手机重启后，这个就自动关闭了，有没有办法可以自动打开呢？』据他了解，有很多不知名的小 App 都实现了，但是很多大公司都没用。我想好好一会，说可能这些 app 被厂商列入了白名单，因此重启手机后还能自动打开那个辅助功能。我实在想不出如何实现这样的效果。最后他告诉我，其实他们也是分析了好久，才发现，那些小 App，都是开启了一个进程（或者是 service，具体记不清了，有兴趣的童鞋可以试试）来守护，因此能够开启。这么一说，我也瞬间明白了。

但同时我提到这样做可能会导致耗电量增加啊，对方的一句话把我真雷住了。“那能费多少电。。。”，我瞬间无语了。但是他们可能因为某些需求必须如此做，因此要实现这样的功能，相对于电量来说应该也能接受，不至于比什么都玩不了的强，体验也确实提升了很多。不用用户每次去开启那个开关，虽然有点风险，但是相对于 Android 上的风险来说，确实低很多。

等第三轮面试完成后，然后 Hr 小妹妹带我到一个很大的会议室，见到一个很年轻的人，听 Hr 说，这个人应是 CEO 之类的，反正职称很高。他就问了些职业规划，平时有什么兴趣爱好，以后有什么打算，薪资要多少？我说到公司后可以先接触一些业务层面的东西，然后慢慢再走架构路线，之后可以负责主要核心模块。平时就看看书，参加沙龙活动，没事打打游戏。他也简单回答我一些问题。之后就是让我先走，等通知。

傻傻的我还就这样高兴的走了，因为我总体感觉还是很棒的，毕竟连过 4 轮哈。从最后的结果中能明白，其实应该是要的薪资太高了。为什么这么说呢？因为一般情况下，最后一轮就是简单看看你这个人怎么样，技术关肯定没问题，否则前三关就 pass 了。可能对方觉得你要的薪资和你的实力不符合，也可能他们想再对比看看，选择一个更合适的人选。

58 到家

从上一家公司面过后，我就紧接着去第二家公司 58 到家，在大屯路东地铁站附近。到了后刚好 12 点，电话联系后，他们说班车司机都午休去了，要等到 2 点才能过去（58 到家面试需要从地铁站做班车过去，路程还算能接受的）。然后我就吃了点饭，在附近网吧 撸一局，看时间点差不多了，我就去那块坐车了，差不多走了 5 分钟做就到了。

北苑路北美国际商务中心，这块有很多公司，什么珍爱网之类的都在那附近。

第一轮面试我的是一个小伙，问了一些基本的 Android 基础，然后问了一下 android 的绘图原理，我说：onMeasure, onLayout, onDraw。然后他说每一个什么作用？那个 onMeasure 主要做什么的？并举了一个例子：一个自定义的滚动 View A 里面如何放另外一个滚动的 View B？我说把 View B 将 onMeasure 里面的高设置成最大，这样就能解决冲突问题。最后他简单说了一些 onMeasure 里面的几个参数，我对此加深了解了。

第一关也就这样过去了，等到第二关的时候看起来一个挺帅气的男人带着一个很显眼的婚戒跟我说一些项目流程上的东西，因为我在 K 公司这块跟老大接触的比较多，因此一般问题难不住我，轻松就过了。

等到第三关的时候，问我一些工作经历，然后问问职业发展规划，平时的兴趣爱好，以及你觉得你和其他人有什么优势。我挺好奇的，为什么最后的这些面试官都要问类似的问题，之后从一个关系还不错的猎头那里了解到，其实他们也就是了解下以后的动向，以及看看这个人的人品。关于优势我是这么说的：我说到公司后可以先接触一些业务层面的东西，然后慢慢再走架构路线，之后可以负责主要核心模块。其实和上面的回答一样，这基本就是所说的套路。他们可以用套路，我们为何不可呢？嘿嘿，别学我，自己根据实际情况来。

本以为就结束了，没想到他们说 CTO 不在，可能还有复试，先让 HR 大美女跟我谈谈。HR 慢条斯理的跟我说了些待遇什么的，了解了下我的状况，问我要多少。我基本和上一个公司说的一个样。

之后再复试的时候，这个大美女 HR 给我了一些建议，说这个 CTO 是阿里出来的，喜欢会说话的人，想到什么就说什么，别紧张。在这面的时候，我就很放松，该怎么说就怎么说，他也问了一些职业发展规划，以及我的经历，基本 10 来分钟就结束。我只想说大美女 HR 真真是体贴入微，感觉很 Nice，这轮基本也顺利过了。之后这个 HR 直接说我被评为 T5，但是以后可以继续努力，我也欣然接受了。不管怎么样，反正拿到 offer 再说，之后慢慢对比。

楚楚街

说起这第三家 楚楚街 我就一肚子火，也不是说第三家不好，只是在去的路上让我备受折磨。从大屯路东 到 知春路，坐地铁应该几十分钟就到了。当时已经快四点了，5 点面试，然后我就打算坐车去（不想再挤地铁了，想轻轻松松的过去），特么的为了省那几块钱，我选择拼车，在路上本以为只需要最多一个小时就到，没想到花了我 1 个半小时(只能感叹北京的车真多，路上堵的不行不行的)。哎，到他们公司的时候都快 6 点了，还好我提前在电话里和 HR 说过，他们说 6 点也是可以的。于是第三个面试就开始了。

首先过来第一位面试官，看样子应该是 Android 技术 leader，开始问了我一些基础的面试题，比如：View 的事件分发机制，View 的绘图，ListView 的实现原理（这个应该是几年前面试的时候经常问题，没想到现在也能遇见）。聊了好一会，然后他拿出他们的客户端给我演示了一个页面，说这个界面比较卡顿，让我分析下原因。我看过后，提出了几个有效的检测卡顿的方案，他们的这个界面主要是 Listview 的 item 里面包含了一个 viewpager，然后 viewpager 的 item 里面有一个大 view，上面有 N 多图片 + 动画效果，因此实现起来很麻烦，最后导致性能卡顿（不得不说产品同学，你的想象力真丰富啊，有没有考虑过研发同学的心情）。然后，他感觉得到了共鸣，因此接下来说话就比较放松了，他说和我年龄差不多，感觉我还是很厉害的（我不禁惶恐不安，我感觉还行，但是应该不是他说的很厉害，可能只是工作时间长了，该积累下来的东西大部分都有了），互留了微信，方便以后的交流（事实是没有啥交流的，只是当你面试通过后，可以有一个拉你入伙的渠道，嘿嘿，不晓得对不对）。

第二个进来的面试官长得挺帅气的，手上戴着戒指（之所以提到这个，是因为在我在我的印象中这个最亮眼，很多多次在和他交流的过程中，我都比较紧张，我就盯着这块看用来放松，说真的如果看着对方的眼睛，双方可能都不会自在，当然除非你很有自信的时候是可以的）。开始简单问了下工作经历，然后就开始聊技术，第一个就是问我知道不知道 二分法，我当时楞了一下，猛然间反应不过来，最后专门确认问了下是不是 二分查找。然后我说在一个数组里面每次查找的时候从中间点开始对比，大于就右边找，小于就左边找，顺带提了一句这要在一个顺序的数组里面。然后面试官就说，二分查找还得每次先排一次序？我当时说是的，结果就感觉很 2，可能没理解清楚面试官表达的是什么或者说我的表达有问题，其实我想说最开始的数组就是一个有序数组，但是面试官可能误解了我的意思，以为每次查到后，都要先排一次序（只能说悲催啊）。

这个问题过了后就再问了我一个问题：『你来说说 Java 的内存管理。』这个问题在一两年前上就栽过跟头，所以当时专门看过相关文章。但是当我回答的时候，由于长时间没怎么看过了，记忆有点松动，大体的说出来了，但是不够准确（回去后就好好补充了下，在之后的面试过程中遇到的概率还是非常大的，尤其在第二面的时候）。然后他问我要多少薪资，我当时说 XX，然后他就问我是不是可以低一些呢？我开始说可以低一点，但是当他问低多少的时候，我心想上面两个公司的 offer 基本感觉到手了，这个可以适当的高点，能给就来，给不了那就算了（我事后想想才明白，这种 2B 的想法绝对不能有，要时刻保持低调，把握住任何一次机会）。最后他说，我得对得起兄弟们（怎么说呢？估计是刚回答的时候不是特别的满意，还有感觉我要的太高了），你这个薪资我没法跟上面谈。然后可想而知，当然肯定没有结果了。

因此奉劝各位，要时刻保持低调，谦虚谨慎，莫要装 B，否则肯定遭雷劈，我这就是一个活生生的例子。

第二轮 B 类公司面试：

面试有很多，说起来可能会长篇大论，下面就总结性的说说，不再说明具体细节，只说我们之后在面试的时候应该注意的地方，以及他们对应聘者的要求。

映客 && 蘑菇街

映客直播在望京 soho, 很高大上的地方，t1,t2,t3 分别对应从低到高的大楼。到公司后，感觉还可以，第一个面我的人是一个技术，基本就问到一些 Android 的面试题，没有任何悬念就过了，第二面的时候，感觉那个人还是比较随和的，问了 Java 内存管理的東西，以及一些其他的问题，最后还都聊得挺开心，第三面的时候直接就是 HR 谈薪资，很容易就过了。

在望京 soho 还去过 蘑菇街，里面的人技术比较好，我当时过去的时候已经 6 点了。那个面试官就跟我聊人生理想，提到一些 Android 系统原理性的东西，但是感觉回答的不是很好。面试官感觉还是很不错的，然后给我说你以后要多看看例如 handler 原理，windowManager 的东西，并且从源码上去分析，网络上的理论知识还是要结合实践的，真是受教了。这部分我有点弱，虽然知道原理，但是看过源码的东西还是很少的，以后需要注重补充。他说他才是高级，我要应聘的这个 架构师肯定是不行的，问我是否愿意做其他的，我当然表示愿意了，现在要综合提升能力，

才能往更高层走。

最后的最后，他很搞笑的跟我说：『我这人真不骗人』。我还纳闷啥意思，最后他说：『今天已经很晚了，第二轮的面试官不在，我明天给你向上反馈下（从之后的一个同事的口中才明白，一般说第二轮的面试官不在，基本就是说你没戏，很委婉的一种说法而已）』。

结束后我看了一下表，我晕，一面就面试我了一个半小时，真特么无语了。不过收获还是很大的，知道自己的不足后，就知道需要补充哪些东西了。

乐视

去了一趟姚家园的乐视，只能说看着挺风光的，但是进去后，特么的真虐人。

电梯分区，还只能在一边的乘坐，很不赶巧的是我去的时间刚好是 10 点，对于他们公司来说这就是高峰期，电梯根本排不上队，而且乱糟糟的（之前在 X 游的时候，大家都是排队的，这边没有，可能地方太小了，排不开吧）。电梯上不去，看来只能跟一些人爬楼梯，一直爬到 9 层，感觉都喘不过气了。

上去后一个很美的 HR（长腿姐）带我找面试官，然后表示没有会议室，原来的会议室都变成工位了，所以让我先在一个小角落呆着（保洁阿姨的专属位置），过了好一会面试官姗姗来迟，也是一些非常基础性的东西，最主要的是他们提到了推送，怎么实现，已经存活情况说了一些。

第二个面试官也是特么来得晚，等了 N 久，闲的无聊就和保洁阿姨聊天，顺带看看他们的办公环境，只能说真心挤得慌。第二位面试官来了后就看看我的经历，因为第一轮的技术面都过了，因此简单聊了下，就说说他们的发展前景，要做海外产品。听我的兴致勃勃，很开心，然后让我等会。

他们基本都去吃饭了，留下了我在那里干等，然后来了一个 HR 的小妹妹，跟我谈薪资以及经历，貌似对我一两年换工作有很大意见，哥就好好给她普及了一番互联网界的基础知识。没想到就在快要搞定的时候，这个小妹妹的老大过来了，然后就看见一个身材超棒，腿很长的漂亮姐姐 HR（长腿姐），坐在我的对面（小妹妹示意我这是她的老大）。瞬间不爽了，都马上谈完了，结果换人再来，真无语了。只能将刚刚的辉煌时刻再来装 B 一次，然后谈薪资神马的，给的也不是很多，我要 XX，她说那么多，只能给我薪资范围最低的一个档次。好吧，就接着吧，然后非要我先

填写一份背景调查表，如果没有问题后，才给我发 offer，我看到美女拿着那份很大的纸张，瞬间无语了。

我当时就不怎么开心，然后长腿姐毕竟老练的很问到：『说你是不是有事？』。我说是的，待会 1 点还有其他地方的面试，然后她说：『那你先回去吧，这个表格发你邮箱，你写好后发给我。』然后长腿姐就送我出去，我又特么的一路爬楼梯下去（9 层啊），电梯等了 N 久都下不去。

接下来说几个有意思的公司

新浪

新浪位于理想国际大厦，记得几年前去新浪面试的时候，傻傻的都没准备就去了，结果第一关就挂了。

这次是下午去，外面还飘着毛毛细雨。去了后竟然特么的让我做面试题，哥已经不做面试题很多年。但是想起了之前的经历，还是老老实实写写，据我估计面试的哥们应该会问上面的东西。还好这次做了万全的准备，刷了 N 多面试题，补充了基础的数据结构理论知识。写起来如行云流水，嗖嗖嗖的没几分钟就完了。

第一个面试的哥们看看卷子，没啥意见，然后问最后一道纠错编程题有没有什么问题，我虽然指出了几个错误，但是感觉他还不是特别满意。因此我仔细看了下，原来是一个静态变量引用了 Activity 的上下文，然后指出，他再问了一些偏底层的东西以及性能优化的地方，轻轻松松就过了。

等到第二面的时候，这个人一看就是技术大牛，问了很多 Java 层面的东西，多态，抽象类，多线程，内存管理等等。我感觉回答的不是太好，多态那有点问题，其他的应该还可以。

然后就进入了第三面，第三面的面试官应该是部门负责人，问了工作经历上的事情以及兴趣爱好，之后的发展方向，想做什么层面的。最后很不幸的是在等待第四面的时候，最开始给我题的美眉告诉我时间很晚了，让我先回去，之后等消息。

至少这次来比第一次高级了很多，不至于第一轮就被刷下去。最后分析了下原因，还是薪资要的太高了，尤其是这类公司。

滴滴

滴滴位于西二旗，应该有两个办公地点，其实我一直很想去滴滴，福利待遇很不错。一年前去过一次，很可惜在

第一轮的时候，因为在某些适配方面回答的不是太好，因此失去了机会。

这次已经准备很多了，进来后还是在去年的位置上坐下等面试官。说实话感觉滴滴成长的很快，办公环境都变的更漂亮了，哈哈。

这个面试官一看就是一个技术宅，开始对我各种炮轰。面试题一个接一个的，在我连续回答十来个题后，看见他还在问，记得在提及到 volatile 的作用的时候，我就开始不爽了，这个东西记得之前在源码里面见过，但是具体的一时说不上来，看着他那样子，埋头在纸上给我出题，我就不怎么配合了。面试了那么多家，就你问了 N 多问题，还有完没完了（其实这也算是抗压的一种面试方式）？我直接说不知道，然后他再问了几个基础性的东西，我想都不想直接说不知道，他貌似已经看出来我已经很不爽了，然后说，那你说说你项目中有没有比较 NB 或者比较有亮点的地方。我的回答直接是：没有。然后他也就不怎么问了，说那先这样。我说：好，就这样，我先走了。然后潇洒的离开滴滴。

现在想想真特么的很 2B，应该低调低调再低调。也可能是那天下午太累了，上午面试了两家，而且已经拿到两家的 offer 了，还都不错，在这特么憋屈，才表现的如此差劲。其实对于问题，知道的话就好好说，不知道的话，可以说说思路 and 想法，然后说说以后会怎么做，利用迂回包抄策略去应答，准没错。至少给面试官知道你还是可以动脑子的人。

在此我真心想悔当时的冲动，向滴滴那位面试官表示歉意。其实不用那样的，我们只需在面试的时候尽力表现自我就可以，以后切莫带着情绪去看待或者回答问题。

对于人生中的很多问题也是这样的，这次栽倒坑里去了（用我老大的话来说，你不在这里踩坑，总有一天也会在另外一个地方踩到，到那时候的损失就不可估计，趁着年轻多多历练自己），总结之后才能更进一步。

百度外卖

百度外卖现在已经不属于百度了，而是单独分出来。

我的一个同事去了百度外卖，我感觉他的能力和我差不多，我就让他推荐了。

去后，上了一个很长的台阶（感觉很庄重的样子），要刷卡才能进去。等了好长时间，面试官把我领到楼下的公

共办公桌，就是那种中间空地，周围都是楼层，能看见其他人在楼层间走动。一个年龄见长的面试官，开始感觉挺随和的，然后说跟我聊聊 Android 基础。

第一个问就是：『咱们先来谈谈 Android 的四大组件。』我彻底懵逼了，尼玛，跟我谈四大组件，有意思么？没想到一直到最后都跟我谈这些，一个接一个的问。说到广播那块，关于一个 app 被杀掉进程后，是否还能收到广播的问题纠结了好久。

然后让我画我之前设计的架构图，我就随便画了画，但是没想到这个看起来很好的面试官让我大跌眼镜，他用鄙夷的笑容告诉我：『你这也太初级了。』我当时心里有几万只草泥马在奔腾，你都 30+了，就不知道鼓励新人啊，我都说过我刚做架构的时间不长，而且鄙视我，有本事你也弄一个架构给我看看啊，一点不尊重我们年轻一辈的劳动成果。也许就怪我当时我真就按照他说的草草画几笔吧，没怎么认真对待。我去其他公司面试的时候，虽然这个图不怎么样，但是至少能解决某些领域的问题，其他面试官都很谦虚。这个百度外卖的面试官，真不是我喜欢的领导，如果以后真让他来带我，那就真完蛋了，很多时候我们都是因为某些人扼杀了我们最初美好的萌芽，而从此失去了创新的意思。

很庆幸的是我在 K 公司的时候，老大一直鼓励我创新，遇到想做的就去做，所以一路下来，虽然很累，但是干的很开心。

所以每当有人问当初为什么选择 K 公司的时候，我都会自豪的说：『我的老大很不错，我在那里很舒服，很开心』。记得在我离开的时候老大给我最后劝告就是：『你要时刻反思自己此刻是不是已经被别人洗脑了。』

第三轮：

1. 百度

百度位于海淀区上地十街附近，有很多大厦。我去的是一个做国外工具的部门，去了后，被百度的环境和氛围震惊到了，在一个很大的技术园区，有网易，百度，腾讯公司，对面还有一个大楼正在修建，估计会是另外一个互联网公司的场地。

进入大厦里面后，由于还没来得及吃饭，边吃手里的饼，边浏览下百度的外围办公区。进入百度的大楼后，两个

入口都设有刷卡机。

在空闲区等了好一会，然后一个人带我进入大厦。在进去之前，到前台那块面试官输入自己的邮箱账号，然后让我填写其他登记信息，我印象最深的是显示器上边贴着一个纸条，说：请离开的时候在此登记，否则会进入百度的黑名单（意思就这样，具体记不清了）。当时震惊了半天，没想到竟然这么严格。

和面试官进入大楼里面后，只记得的印象是：很整洁，高大。出楼梯后，脚踩着厚厚的地毯，稍微走快点，都感觉很松弛，脚下如踩棉花一样。

为什么有地毯，而不是地板砖——到了夏天很多漂亮的长腿美女穿着高跟鞋踩在地板砖上是一个怎么样的体验呢？噤噤噤.....

我在等候区等到第一个面试官，然后我们简单聊了下 Android 技术，其中有两点有必要提下：

其中一点是：说说 View 的事件分发机制。然后我就说了好多，从 WindowManager->Window->DecorView->View。最后我说当所有的 View 都不处理事件，事件会最后会传递到 Activity 的 onTouchEvent 上。然后面试官立刻说：『哈？你这是颠覆我的三观啊？』然后我意识到可能有问题，但是记得《Android 艺术开发探索》上确实写过到 Activity，但是不是到 onTouchEvent 还真没底。面试官很自信的样子，让我颤抖了。但是随着我的坚信，面试官说：『不行，我不能冤枉你是不！』立刻在手边的 MBP 上看了一下，自言自语感叹道：『还真有啊！』我顿时无语了。

另外一点是：问我 Service 上能不能弹出对话框。对于这个问题，我印象最深刻了，记得一年前的时候，在另外一个公司就因为这个问题让我尴尬万分，回去后专门对这块进行补充。我的回答是可以的，但是面试官面带差异的表情告诉我这是不行的，Dialog 必须要依附于 Window 才能显示出来。然后我的解释会让面试官郁闷一会：我说这个是可以弹出的，我之前也专门试过，不过他弹出是有条件的。条件是：

必须在 Manifest 里面注册系统权限

在显示 dialog 的时候必须要加一个 flag.

我的理由是：系统对话框可以在低电量的时候弹出对话框，我们同样也可以采用该方式来实现。

面试官语塞，然后给我说 Dialog 是必须要依附在 Window 上，Toast 其实也是一个 Window。我听着这些话，就想起以前看过的一篇文章上也确实是这么说的。估计该面试官回去要好好补充下一些知识了哦。然后该面试官让我不能用 ArrayList,用数组 写一个队列。这块刚好我在之前项目中特意用了一下，写的时候，主要有三个方法：put(), get(),peek()。然后考虑下队列的特性，一端进入，一端出去。我当时遇到了盲点，没怎么写完，最后给面试官说了下思路，大体是对的。但是关于选择位置那块没怎么想好。不过这不阻碍我进入第二轮。

第二轮面试的时候，面试官带了很多纸张，我瞬间压力山大，知道不太妙。不出所料，这个面试官，从动画实现原理，到 handler 实现原理，一步步深入各种原理，当我感觉回答的不错的时候，然后他就顺着我的问题继续深入。我只能说我尽力了，有些东西，平时开发的时候真心不注意，但是就是因为没有留意，所以就没办法继续回答他的问题。

面试官把我带出大厦的那一刻，我心情很不好，很可惜没进入百度，之后应该需要准备很多东西。我要说，我还会再来的，哈哈哈！最后也归还身上的一个牌子到前台后，省的被拉入到黑名单（好吓人的样子）。

以后有时间多看看原理性的东西，最好整理一个自己的博客，写上自己的一些看法和感悟，这样记得最深刻，即使几年后也不会遗忘，只是看看别人总结的东西，真的就不怎么记得住。

关于博客可以使用 Hexo, 我的博客也是如此，可以整理一些自己的东西与心得。

2.阿里

这次去的是一个阿里的高德部门，在望京 Soho 附近的首开广场。去了以后首先找厕所，你们知道么？厕所竟然从大厦楼层的一个角转了一大半圈才找到，回来后进入找不到前台了..... 瞬间无语了。问了好几个美女才回到前台，然后接待我的 HR 美女貌似等得不太耐烦了（宝宝心里苦，厕所好远，都找不到回来的路了）。在一个小型会议室等待面试官，看了下布置氛围和环境，感觉太棒了，很多东西都体贴入微。

回顾上次阿里的悲痛遭遇

其实这是我第二次来这边面试了，上一次过来的时候，是刚过完年。提到这里我就苦不堪言，为何如此说呢？当时是 2016 年 2 月 15 日，因为我参加好朋友的婚礼（不得不说，我这个年纪的人都开始结婚了，这次回去有 4 个好朋友都结婚，可想而知，一场完了以后还有另一场，虽然累，但是值得）推迟了好几天才回北京，在参加同学婚礼的时

候接收到阿里高德部门的面试邀请。回到北京的当天是 12 点多，然后回家，一个关系非常好的朋友说今天她们要宴请公司的人吃饭，因为她们结婚了，让我帮忙弄个 MTV。我想这是朋友的终身大事，因此必须要好好干。

我下午 4 点是阿里高德的面试，因此时间很紧促。我凭借我大学的技能在两个小时内搞定这个 MTV，总体来说还不错，就迅速发给朋友，弄完已经 3 点了，然后打车立刻去首开广场。

高德的面试是 4 点钟，匆匆赶到后，就等待面试官。面试很不理想，因为什么都没有准备，而且心力憔悴。面试官问的是一些基础的 Java 问题，很可惜我没怎么回答好。于是就深深的浪费了一次机会，之后和朋友提起此事，无比后悔，当时其实是可以和 HR 电话再约一个时间的。

这次对我的打击很大很大，因为这是我这么多年第一次面试 BAT 的职位，一上来就受挫，很不是滋味。我在这里失利后我就各种准备资料，增强自己的能力，面试前必须要刷题，虽然简单，但是不失为一种方法，虽然不一定有用，但是会加深印象，尤其是去 BAT 这些公司，一定要准备好，否则就别浪费机会，这就是我的教训和经验。

为了 6 月份的这次面试策划了很久。以前对什么可能都不是很上心，但是这个事件深深的刺激我了。

第一个面试官来了后问了一些基本问题，很顺利就进入到第二轮面试。

第二轮也基本是技术面试，问了一些 Android 基础和 Java 基础以及内存管理。

第三轮的面试官应是部门负责人，看起来很好说话的，问了一些经历和基本情况后，问我薪资要多少以及之后的发展方向。我说要 XX，之后希望在架构方面发展，但是也可以从业务开始。貌似这里回答的不怎么好。然后让我留了他的联系方式，我知道很有戏哦。

因为我在进入 K 公司的时候也是这样的，老大感觉我很不错，于是留了微信后，我基本就顺利入职。

回去后的一两天还是很焦虑的，但是我知道大公司都是有流程的，因此我告诉自己不要焦急。过了一两天后他主动加我微信，然后问了些基本情况后，就说他要做最后的总结，让我等着，最迟一周后就有消息。我感觉希望超大的，开心了好久，本以为就可以这样过去。但是一周时间过去了，没人通知我，我开始焦急了，于是我开始主动和他说话，反思自己是否有什么地方做的不好。

经过很多面试后我总结出了结论就是要薪资太高了，于是我在微信里面给他说，只要能过去，薪资低点也是可以

的。但是问了他好几次，他都没有回话，看着微信消息记录，都是我发给他，而他没有任何回复，已经过去好多天了，我知道没希望了，他说不管怎么样都会给我回复的，但是我真绝望了。

就像相亲一样，遇到一个不错的美女，开始都一起聊得很不错，她开始加你好友，并且和你说看好你，不管能不能做女朋友，她之后一定会回复，但是苦苦等待一段时间后，不管你怎么给她说话，但是她就是不理你。可能她真的忙，但是也不可能连续一两天都这么忙吧。于是你知道没结果，因为无言等同于没有希望。为了避免一些幻想的存在，你会将她删除掉，不想留下任何关于他的信息。

同样我也是把这个阿里高德的老大的联系方式删掉，微信也删掉。在我失去希望的时候，过了几天看见他要主动加我，但是我想可能只是安慰的话语，最多告诉我，我不适合他们的职位，因此我为了避免尴尬，直接删除那个加我好友的请求（如果说真的合适的话，应该会很重视你的，不可能好几天都回复，怎么有一种备胎的感觉，呜呜呜，我不想被发好人卡，宁愿做高傲的兔子，也不想做纸老虎，虽然尽管只是纸老虎，但是也会拥有属于它的一片森林）。

于是阿里的这次机会就失去了。

总结后的结论就是：去大公司要的薪资不要太高，否则对方只能感谢你的到来，因为比你优秀的人太多了。

聚美优品

聚美优品 位于东四十条地铁站附近。路过一个竹亭子后，进入大厦里面需要用身份证在前台那块登记后给我一个纸条，上面写着我的身份证信息，然后在门禁卡附近刷二维码进入（真担心个人信息泄露哦，当然一般情况下没人会关注你是谁，千万别干坏事哦，会被查出来的，哈哈）。

推荐我去聚美优品的同事接我上去后，带我到前台填写基本信息。我只写了最基本的信息，然后她说，你就写这么点啊。我说，其实这些信息够用了，写那么多没用，还会暴露你的个人信息。面试成功后，如果有需要可以写详细些，但是一般去面试最好别写身份证信息。工作经历基本也只是最近两个，之前的就不用写了，写那么多没什么用，简历中都会有的。

记得刚工作那会，傻傻的全写了，真耽误了不少时间。过了一会，她把我交给 漂亮的 HR 温柔姐，然后就先忙去了。温柔姐告诉我一般情况下有两轮基本就过了，先让架构师老大直接面我，让我先等候。

过了一会温柔姐不好意思的跟我说架构老大先让一个技术面我，问我是否有意见，我当然没意见了，这是很标准的面试流程（如果你有意见，建议还是别说太多的话，基本都这样的，要淡定）。

一面技术给我一种很成熟的感觉，开始问了我一些基础技术问题，外加 Java 内存管理知识。后给我出了一道算法题，说有一个数组最多存储 6 个数，如果有普通用户的话，存储四个 vip 的客户，另外两个是普通用户（留出一定的空间给普通用户），让考虑全面点（一般都是结合实际场景，让你写出一个算法，要具备的能力就是抽象，处理问题的思路与细节，还有最基本的编码功底）。

然后我就考虑各种情况，第一种是非空情况，然后下面就是几个大的 if else，至少四个条件，基本涵盖了全部情况，然后每个条件里面写上对应的存储数据的过程。由于我的四个大条件都把距离占的差不多了，在写里面细节的时候，用中文描述。过了一会他回来后，看了下说：『你这个还有中文啊！』我尴尬的笑着说：『我先写条件的，最后发现没有空位了，只能用文字代替了，你看我正在另外一个纸上写全部的完整算法。』指了指纸上刚写一小半的代码。他也会心一笑，并指出算法上应该改进的地方，基本 ok 啦。

然后等第二轮的面试，看起来更成熟，但是说话有一种很亲近的感觉。问了基本情况，然后拿出他们的 app 让我看看首页的实现效果，说说怎么实现的。对于这种情况，基本就是考察你的抽象能力，以及分析问题的能力。我先说出使用 ListView 的 header, footview, 然后使用 ListView 的 type 来实现。然后简单说了一些性能优化的东西，该面试官提出我的做法可能会存在性能瓶颈。其实他说出这块是在指导我说这块会有问题，我当然明白他的意思，于是说这块采用 recyclerview + fresco 来实现，可以有效的改善问题（其实提到这些，就说明你看过很多新技术了，有时间最好还是要自己练练这些东西，毕竟熟能生巧）。

他也没深究，基本就感觉不错，开始谈了谈他们的目前状况，以及即将遇到的问题。他在只言片语中都把我当做内部人看，我也心里感觉很舒服。最后告诉我如果我愿意，他就向上报备了，意思是可以继续下一轮。当时他问到我的薪资的时候，因为之前已经说了 N 多次，有的成功，有的感觉很亏，于是这次我并没有说，只是笑笑，而对方说：『那就按照年薪算吧，你打算要多少呢？』我当时什么也没有多想，然后就说：『我希望在我现有的薪资基础上，能上涨 15% - 20%。』他经过在手机上一阵比划后，告诉我可以达到我的预期效果。整个过程感觉很愉悦。

因为面过了一些，并有 offer，但是还是想多看看，结果把自己搞的疲惫不堪。但是最后的最后，温柔姐给我打电话说面试通过。

最终结果

最终我辞职后在家休息几天，没事的时候去咖啡馆看看书，上上网，好好过几天轻松的日子，然后再说定去哪里工作。

总结：面试和必备的技能

这里只简单列举一些东西，可能不是特别全，但是却特别适用，也不一定按照下面的流程，有可能是穿插的，也有可能都有，根据公司的规模以及面试官的心情而定（哈哈哈，你们就自求多福吧）。建议大家还是要将下面的东西全部掌握，没事写写代码，练练手，在项目中能用到的地方一定要用，有可能会遇到很多坑，一定要自己想办法填坑，之后回忆起这段经历，肯定可以敢理直气壮的跟别人讨论。如果你说的头头是道，那么对方会先输一层，然后在心里对你佩服。

一般情况下第一轮都是基础面试，需要扎实的基础

最常用的 Android 基础知识

Java 基础知识

了解一些 常用东西的原理，例如：handler，thread 等

项目中的技术点

第二轮的时候需要了解更深层次的东西

Android 事件分发机制原理

Android 绘图机制原理

WindowManager 的相关知识

进程间传输方式

Java 内存管理机制

一些常用的 list,map 原理，以及子类之间的差别

能进入第三轮基本没什么问题，但是要注意以下问题

该轮一般是 老大或者部门负责人，问的问题一般都看 深度与广度

当问及薪水的时候，要说一个合适的，小公司随意，大公司一定要慎重，当心里没底的时候，可以告诉对方，让对方给一个合理的薪资。一般都是在原工资基础之上增长，听猎头说一般涨幅都在 15%-30%，超 NB 的可以要 30% 及以上，如果感觉自己还不错的，挺厉害的，建议最高 20%，一般人就定在 15% 左右最靠谱。公司内部一般有一套机制，根据公司情况而定。

我们的面试原则就是拿到合理薪资，得到 offer

个人发展情况，这个问题很难回答，如果和公司方向不符合，极有可能和公司无缘。建议多试探性的问问公司缺少什么，你能否给予公司对应的东西。当然对于有自我追求的人，那可以放心大胆的提。我的方向就是架构师，哈哈，挺极端的，别学我哦。我感觉选择都是双向的，因此我知道自己需要的是什么。

你最擅长什么 UI 还是其他什么？这个问题更不好回答。你要说你擅长 UI，是不是意味着你其他能力就不行？虽然我不知道面试官的用意，但是我能感觉到，这个问题不是那么好回答，我会回答说自己都行，来什么业务接什么需求。可能回答不太好，总之和公司的职位吻合就行，这样总不至于出错吧。

二、近期 Android 面试经历总结

原文来自：<http://www.jianshu.com/p/a87349270792>

看一看自己已经有两三个月没有更新博客了，其实一开始的时候，主要原因是加班严重，自己业余时间都用来做一个小 APP 了。然后公司依然每况日下，工资拖了两个月（我真的感觉自己已经够能坚持的了！）。于是自己也开始加入到找工作的行列了。首先自己是在某勾网投简历。搞的我真的是快怀疑人生了。每天都投一些公司，但是都石沉大海。然后我觉得是自己的简历写的有问题。然后找各种朋友帮忙修改，然后又持续在某勾网投了一个星期。纳尼？纳尼？纳尼？难道看我只有半年工作经验？然后给朋友抱怨了一番。被安利去了其他的招聘网站。面试也终于是步入

正轨！就这样白白耽误了我 2 个星期的时间。

因为自己刚刚毕业半年，其实很多公司都有工作经验限制，大公司的话简历确实不好过。但是创业公司不怎么 care 这些东西。自己前前后后面了 5, 6 家公司吧。所以还是建议各位同学，不要像我一样浪，毕业就跑来创业公司，想要独当一面。我来捡几家我觉得有代表性来说吧。

网易

首先在被某勾网整的精神崩溃的时候，因为有朋友在网易做 HR，所以简历给过了（其他 BAT 应该都没办法过简历）。然后面试：（BTW：到现在为止我觉得网易的面试是比较有水平的，问题由浅入深，慢慢触及到自己知识的底线。）

第一面面试的内容大多还都是简历上写的相关内容。

自我介绍：简单介绍了一下自己，还有自己做过的 OpenGL 视图库和骨骼动画播放引擎讲了一些。但是面试官应该是不怎么熟悉这些的。不过也算突出了自己的特点了。

然后应该是想看看我基本界面编辑会不会，然后问了常用的布局有哪些？这个问题很简单嘛，自己把 FrameLayout，LinearLayout，RelativeLayout 讲了一通，然后又讲了 Coordinatorlayout 和 ConstraintLayout。

然后他问了开发是使用什么工具，怎样调试程序，和解决 bug 的。自己把日常的做法讲了一遍。包括各种断点的使用方法。内存泄漏的检测方法。内存抖动如果查找到原因和如何使用 TraceView 来进行性能调优的。然后又问了一些具体的操作方法，估计是想看看是不是我编的吧。

然后开始问架构的东西。主要是 MVC 和 MVP，当然更主要是 MVP 喽。然后自己把 MVP 的架构讲了一下，以及在项目中遇到的 MVP 架构不合理的地方 和自己认为对这个不合理地方的改进。然后讲了自己针对公司项目的 MVP 架构，写的 Android Studio 插件。

然后问了我简历里写的技术细节的东西。比如屏幕适配。项目中为什么使用 OpenGL。礼物动画为什么使用 SurfaceView，它与 View 有什么区别等等。

最后先让我说我看过的一些 Android 的源码，然后我讲了一下 Activity，PhoneWindow，View 他们的关系。还

有 Touch 事件的分发过程，自己还说看过 Handler 消息传递机制，不过他没有让去细讲，估计听了前两个讲的挺细的觉得够了吧。然后他又问了一下 Activity 从 Launcher 的启动过程。这个自己明确表明没有看过源码，但是自己也简单的分析了一下可能过程，感觉面试官还算认可。之后查询相关资料发现自己分析的有些地方还是不对的，而且还有很多细节没有分析到（囧）。

整个过程面试下来感觉还是不错的。面试官的问题由浅入深。而且针对自己项目细节都有针对的问题。面试到最后面试官说之后会有主管给我面试。我觉得一面应该是过了吧。然后收到回复是面试哥觉得我去这个部门太委屈了，因为是游戏辅助 APP，然后让 HR 姐姐把我推荐到云音乐，非常感谢面试哥，人真的蛮好，也谢谢 HR 姐姐帮忙推荐其他部门。但是无奈云音乐并不缺人，但这里也不要我了.....不过对方招聘岗位本来就是高级工程师。可能感觉我经验还是不够吧。

某软件公司

这个软件公司相对于网易的面试要简单一些。主要问题偏向于解决他们项目中面临的问题。

自我介绍：当然这个还是那些内容嘛。

他又问了一些基础的东西。这个我也记不大清了。

然后又问了 MVP 架构的东西。

然后他开始问一些问题，应该是他们项目中遇到的问题。讲了一个需求“要在手机上显示一个表格，表格可能 100*100，要在这些表格里显示数据，手机屏幕一般只能显示 4 列，这个要怎样做。是使用 View 用 Canvas 绘制上去，还是使用 ViewGroup 添加 View 进去？”。自己给出了使用 RecyclerView 然后自定义 LayoutManager 的方式实现（因为自己这样做过嘛，详情可以看我的把 RecyclerView 撸成马蜂窝）。他听了还是比较满意的。

还有一个是手机和平板适配的办法。这个其实很简单嘛。Android 官方有很好的方案。layout-swxxx+Fragment 嘛。然后接着问了 Fragment 使用的具体方法都有哪些。这个也不难嘛。什么 Fragment+FragmentAdapter+ViewPager，或者使用 FragmentManager 来通过 Transaction 来操作 Fragment 等等。

最后问了一下工作模式。一个需求拿到手，要怎样把需求做出来。这样的工作流程是怎样的？这个自己从需求分析到与其他部门商定通信协议，然后再分析需求要注意的 case，最后再实际编码，都讲了一下。他也比较满意的。

整体下来，这个公司偏向于应用和解决问题的能力。具体的原理没有去问太多。过的也是挺轻松的。不过这里可以看出无论是大公司和创业公司，对 MVP 都很有比较高的要求的。还有就是简历中自己写的东西一定要能讲出个 123。这个家公司还是比较注重人才的。开的薪水还是不错的。但是最后我聊了聊工作内容，自己并不是很感兴趣。感觉对于自身的成长帮助不是很大，所以并没有准备去。

某创业公司

这家创业公司的面试其实只能算是缘分吧。可能就是传说中的非常适合吧。他们是做 VR 的，所以比较关注我的 OpenGL 技能（其他公司都对我这一点不以为然）。而且他们会去做各种动画效果，所以也知道骨骼动画是个什么东西。而我从零开发了一个骨骼动画引擎，他们也知道这个的难度，虽然是个 2D 的，但是原理大致相同，自然也是十分青睐。而且他们是要做 SDK，而我也有开发 SDK 的经历。

主要问的问题都是针对于 OpenGL 和我写的骨骼动画引擎和细节，还有 SDK 的设计这三方面。在这个面试过程中也感受出了他们公司技术还是可以的。因为自己搞 OpenGL 事件比较短，理解还不是很深，面试哥从 OpenGL 怎样工作的，和 GPU 怎样工作的给我讲了一通.....在理解深度上被完爆.....

之后几轮技术面差不多同样内容。最后就是创业公司的尿性嘛，副总裁、CEO 随便面面就 OK 了。

其他公司

其他还有几家公司。

其中一家倾向于 ROM 开发，也就是 Framework 开发工程师，我觉得问的问题对于我们应用的开发的没有参考价值。简直不是一个世界的！

还有几家外包公司，都没有技术面，做个笔试题 OK 了.....

还有一家公司昨天面试的，上来问了一个充满杀气的问题：你最擅长什么！听到这个问题，我的第一反应是一脸懵逼！第二反应：这是要怼我了！无论回答什么他都会在这个方向问到死。可能一个简单的 API，记不住都会让他认

为，这么简单的东西你都不会还说擅长？而往往就是一些简单的 API，谁会去背那些东西。不过遇到了我也是认命了！然后我就说了个 UI 绘制方面。然后他又问了我在这方面做过的最复杂的东西是什么。然后我讲了自己做的 OpenGL 视图库和骨骼动画引擎。不过他好像并不怎么关心。应该是没做过 OpenGL 吧，也没做过什么复杂的动画效果吧，不知道要问什么。然后又问还做过什么，然后我把自己做的自定义 View 说了一下，还有就是我的 RecyclerView 的马蜂窝布局管理器（一个自定义的 LayoutManager）。但是在这个过程中他愣是把我说的自定义 LayoutManager 理解成了自己自定义的一个 ViewGroup！然后问的问题我都感觉不是我这个 LayoutManager 所能管的内容。然后弄了半天才发现，面试哥一直理解的有出入.....真是囧！整个面试过程下来感觉天南地北。还有他问我用的 OpenGL 是用的什么容器？我说 GLSurfaceView。好像他听成了 SurfaceView，然后问 SurfaceView 和 View 有什么区别。一开始这里是我没有听清，因为紧接着他问的 OpenGL 在什么容器嘛，所以我把 GLSurfaceView 的工作原理讲了半天.....面试哥听的好像也是云里雾里，然后才知道，他问的 SurfaceView，然后我又把 SurfaceView 的讲了一通.....整个面试我觉得面试哥和我都不舒服，所以我觉得应该不会过吧。

总结

面试下来也有些感悟吧。

首先，基础知识，这些都是在面试中必然涉及的问题。可能我上面并没有列点来具体说明，但是多少都有涉及。而且我即使列出了这些，你也不一定会遇到，其实这个范围太大了，所以自己要注意对于细节的追求。如果是要面试了，那么就自己找一些，都有很多篇文章来介绍这些知识点了。

其次的面试内容要取决于面试官了，我感觉大概分为三种面试官。

第一种是那种会根据你的简历来面试的。这也是大多数。

第二种会根据自己的需求来面试的。

第三种，根据你叼不叼来面试的。

针对于第一种我觉得首先是对自己的简历严格把关，写在上面的内容一定要可以说出 123。对于程序员来说语言表达能力比较差，比如我就是这样。所以自己在面试之前先试着说一说，因为面试过程你只能通过说来让对方知道你

所做的东西，你能做出来的一定要说出来。据说有些人没做过也能说出来。不过我是没有这个能力。

针对于第二种其实要看自己的应变能力。因为你很难预测到对方可能会给你提出怎样的问题。有的时候你一听到问题没有办法马上想到解决的办法。那么这里有一个小技巧，你可以说没有听清楚，中间有一点信号不好，让面试官重复一遍问题，来为自己争取思考时间。如果还没有想出来，那么你可以先分析提出的问题，然后渐渐接近问题的答案。

针对于第三种，我自己之前也没有准备过。第三种面试官非常喜欢问的问题是：你擅长什么？你在哪方面有过人之处？你对哪方面比较精通？其实遇到这种面试官就心中默念阿弥陀佛吧！因为这种面试官要求非常高。他希望的是你对一个领域研究的非常深入，你一个人可以解决这个领域所有的问题，而且希望你是在这个领域做出过一定突破。比如我们常用的什么加载库，你优化它，结果快了多少，就是 1%都可以。这个可不是一般的人能够达到的。而大多数情况下我们都是各个领域都涉及一点，虽然可能看过源码，有一定理解的深度，但远达不到可以解决这个领域所有问题的程度，更没有这种技术突破。像有些第三方库源码我也去看过，但自认为达不到他们所谓的擅长和精通。不过针对于这种面试官我觉得也是值得准备一下的。不然自己真的是会一脸懵逼。有的时候就是遇到这样范范的问题，你就已经不知道要如何说话了。然后说错一句话就会被各种完爆，脑海中回响各种 Enemy is Legendary！之后几天估计都要怀疑人生了.....那么下面说一下准备方法吧。不过自己也是刚刚遇到这种面试官。也没有确定是否正确。也是给自己的计划。

选一个自己相对比较擅长的领域。

基础要 背！平时我并不太去留意要记住各种 API，但是这里就是要背过。不然面试官就会想“这么简单的 API 都含糊不清还说擅长？”就是真的面试的时候记不清了，也不要打磕，要非常自信的说个差不多的，不要说“好像.....”这样的话。要的就是自信，因为这个 API 面试官也不一定记得那么清楚。不过自己能背过才真的有底气。

试着去了解这个领域市面上的技术。一般的话就是一些库或者框架。这里要记住，不要急着去看源码，要先掌握这些技术都有哪些优缺点，尤其是缺点！因为我们经常因为一个库有什么优点而去使用它，但是缺点往往是我们容易

忽略的地方。而知不知道这个库的缺点，是你能不能驾驭这个库的一个关键。

如果有时间的话，研究其中一个众所周知的库的源码。并试图找到它缺点的原因，并找到其解决方法。当然你如果正要准备面试了，肯定没有这个时间。那么就找一些现成的相关的文章来看看吧。然后记住。

结果

近期的面试过程就是差不多这个样子。而且对于我这种刚刚毕业半年，很多公司连简历都比较难过。所以也没有一些非常大的公司的面试机会。因为大公司社招和校招分的很清楚。最终自己决定去那家 VR 的创业公司了，原因有以下几点：我本放荡不羁爱自由，第一当然是兴趣，第二呢希望补充 Android 方面 NDK 的技术，第三希望可以做一些 OpenGL 领域更深入研究，而 VR 正是对其要求非常高，自己也想去挑战这方面的技术瓶颈。下次面试的时候可以非常有底气的回答最擅长什么这样的问题。

9. BAT 面试题

1. 腾讯 2016 年面试题

1. 已知一棵二叉树，如果先序遍历的节点顺序是：ADCEFGHB，中序遍历是：CDFEGHAB，则后序遍历结果为：（ D ）

- A. CFHGEBDA
- B. CDFEGHBA
- C. FGHCDEBA
- D. CFHGEDBA

知识点

对于二叉树的遍历方式一般分为三种先序、中序、后序三种方式：

先序遍历（根左右）

若二叉树为空，则不进行任何操作：否则

- 1、访问根结点。
- 2、先序方式遍历左子树。
- 3、先序遍历右子树。

中序遍历（左根右）

若二叉树为空，则不进行任何操作：否则

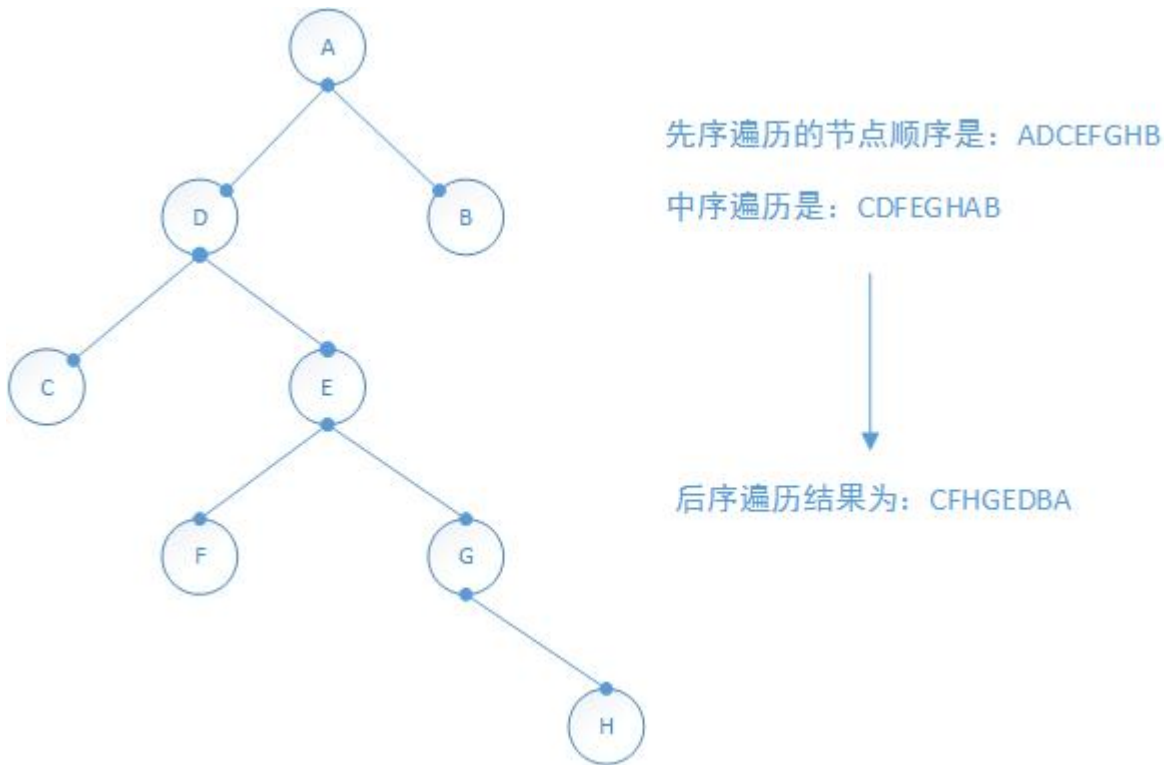
- 1、中序遍历左子树。
- 2、访问根结点。
- 3、中序遍历右子树。

后序遍历（左右根）

若二叉树为空，则不进行任何操作：否则

- 1、后序遍历左子树。
- 2、后序遍历右子树。
- 3、访问根结点。

因此，根据题目给出的先序遍历和中序遍历，可以画出二叉树：



2. 下列哪两个数据结构，同时具有较高的查找和删除性能？（ CD ）

- A. 有序数组
- B. 有序链表
- C. AVL 树
- D. Hash 表

知识点：

数据结构	查找	插入	删除	遍历
数组	$O(N)$	$O(1)$	$O(N)$	—
有序数组	$O(\log N)$	$O(N)$	$O(N)$	$O(N)$
链表	$O(N)$	$O(1)$	$O(N)$	—
有序链表	$O(N)$	$O(N)$	$O(N)$	$O(N)$
二叉树（一般情况）	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(N)$
二叉树（最坏情况）	$O(N)$	$O(N)$	$O(N)$	$O(N)$
平衡树（一般情况和最坏情况）	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(N)$
哈希表	$O(1)$	$O(1)$	$O(1)$	—

平衡二叉树的查找，插入和删除性能都是 $O(\log N)$ ，其中查找和删除性能较好；哈希表的查找、插入和删除性能都是 $O(1)$ ，都是最好的。所以最后的结果选择：CD

3. 下列排序算法中，哪些时间复杂度不会超过 $n \log n$ ？（BC）

- A. 快速排序
- B. 堆排序
- C. 归并排序
- D. 冒泡排序

知识点

各种常用排序算法						
类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$	$O(n \log_2 n)$	不稳定
归并排序		$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数

根据上图，观察平均情况，最好最差情况的时间复杂度基本可以知道答案了，最后结果选择：BC。

4. 初始序列为 1 8 6 2 5 4 7 3 一组数采用堆排序，当建堆（小根堆）完毕时，堆所对应的二叉树中序遍历序列为：（ A ）

A. 8 3 2 5 1 6 4 7

B. 3 2 8 5 1 4 6 7

C. 3 8 2 5 1 6 7 4

D. 8 2 3 5 1 4 7 6

初始化序列：1 8 6 2 5 4 7 3，小根堆就是要求结点的值小于其左右孩子结点的值，左右孩子的大小没有关系，那么小根堆排序之后为：1 2 4 3 5 6 7 8；

中序遍历：左根右，故遍历结果为：8 3 2 5 1 6 4 7

故最后选择的结果：A

5. 当 $n = 5$ 时，下列函数的返回值是：（A）

```
int foo(int n)
{
    if(n<2)return n;

    return foo(n-1)+foo(n-2);
}
```

A . 5

B . 7

C . 8

D . 1

6. S 市 A , B 共有两个区，人口比例为 3 : 5 ，据历史统计 A 区的犯罪率为 0.01% ， B 区为 0.015% ，现有一起新案件发生在 S 市，那么案件发生在 A 区的可能性有多大？（C）

A . 37.5%

B . 32.5%

C . 28.6%

D . 26.1%

这道题首先得了解犯罪率是什么？犯罪率就是犯罪人数与总人口数的比。因此可以直接得出公式： $(3 * 0.01\%) / (3 * 0.01\% + 5 * 0.015\%) = 28.6\%$

当然如果不好理解的话，我们可以实例化，比如 B 区假设 5000 人，A 区 3000 人，A 区的犯罪率为 0.01%，那么 A 区犯罪人数为 30 人，B 区的犯罪率为 0.015%，那么 B 区的犯罪人数为 75 人，求发生在 A 区的可能性，

就是说 A 区的犯罪人数在总犯罪人数的多少，也就是 $30/(30+75)=0.2857$

当然，也可以回归到我们高中遗忘的知识：

假设 C 表示犯案属性

在 A 区犯案概率： $P(C|A)=0.01\%$

在 B 区犯案概率： $P(C|B)=0.015\%$

在 A 区概率： $P(A)=3/8$

在 B 区概率： $P(B)=5/8$

犯案概率： $P(C) = (3/8 \times 0.01\% + 5/8 \times 0.015\%)$

根据贝叶斯公式： $P(A|C) = P(A,C) / P(C) = [P(C|A) P(A)] / [P(C|A) P(A) + P(C|B) P(B)]$ 也可以算出答案来

故，最后结果选择为：C

7. Unix 系统中，哪些可以用于进程间的通信？（ABCD）

- A . Socket
- B . 共享内存
- C . 消息队列
- D . 信号量

知识点

管道（Pipe）及有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux 除了支持 Unix 早期信号语义函数 `sigal` 外，还支持语义符合 Posix.1 标准的信号函数 `sigaction`（实际上，该函数是基于 BSD 的，BSD 为了实现可靠信号机制，又能够统一对外接口，用 `sigaction`

函数重新实现了 signal 函数)；

报文 (Message) 队列 (消息队列)：消息队列是消息的链接表，包括 Posix 消息队列 system V 消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

共享内存：使得多个进程可以访问同一块内存空间，是最快的可用 IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

信号量 (semaphore)：主要作为进程间以及同一进程不同线程之间的同步手段。

套接口 (Socket)：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由 Unix 系统的 BSD 分支开发出来的，但现在一般可以移植到其它类 Unix 系统上：Linux 和 System V 的变种都支持套接字。

故最后选择的结果为：ABCD

8. 静态变量通常存储在进程哪个区？ (C)

- A . 栈区
- B . 堆区
- C . 全局区
- D . 代码区

静态变量的修饰关键字：static，又称静态全局变量。故最后选择的结果为：C

9. 如何提供查询 Name 字段的性能 (B)

- A . 在 Name 字段上添加主键
- B . 在 Name 字段上添加索引
- C . 在 Age 字段上添加主键

D. 在 Age 字段上添加索引

10. IP 地址 131.153.12.71 是一个 (B) 类 IP 地址

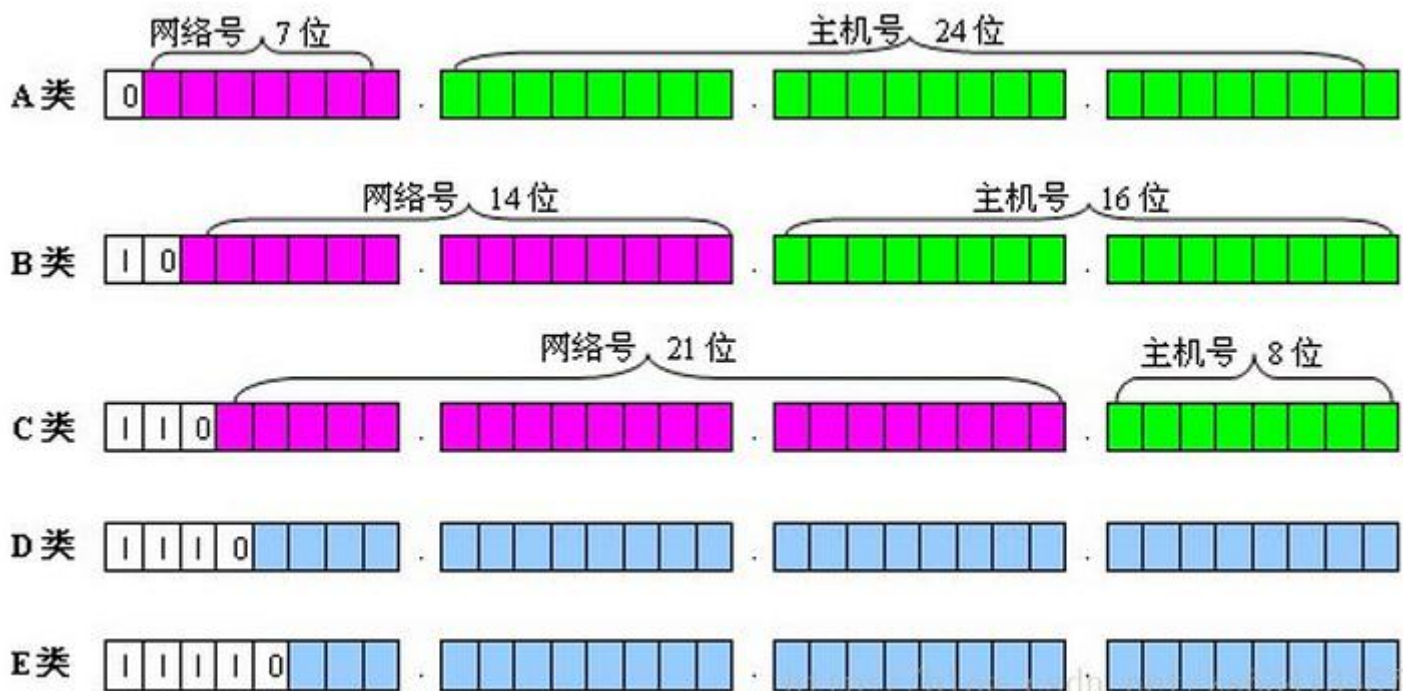
A. A

B. B

C. C

D. D

知识点



11. 浏览器访问某页面，HTTP 协议返回状态码为 403 时表示：(B)

A. 找不到该页面

B. 禁止访问

C. 内部服务器访问

D. 服务器繁忙

12. 如果某系统 $15 \times 4 = 112$ 成立，则系统采用的是 (A) 进制

A . 6

B . 7

C . 8

D . 9

这题因为是选择题，我们可以直接从 A 的选项开始，假设是 6 进制的，我们把等式 $15_6 \times 4_6 = 112_6$ 转为十进制，就是 $11_6 \times 4_6 = 44_6$ ，最后验证等式是否成立，明显等式是成立的，因此答案已经出来了，选择 A。

当然我们也可以假设是 X 进制，且我们知道 X 大于 5，则： $(x+5)_x \times 4_x = xx_x + x_x + 2_x$ ，所以最后计算的结果也为 6。

13. TCP 和 IP 分别对应了 OSI 中的哪几层？ (CD)

A. Application layer

B. Presentation layer

C. Transport layer

D. Network layer

知识点

ISO/OSI	TCP/IP	
应用层	应用层	传递对象： 报文
表示层		
会话层		SMTP FTP TELNET DNS TFTP RPC 其他
运输层	传输层	传输协议分组 TCP UDP
网络层	网际网层 (IP层)	IP数据报 IP(ICMP等) ARP RARP
数据链路层	网络接口	帧 网络接口协议(链路控制和媒体访问)
物理层	硬件 (物理网络)	以太网 令牌环 X.25网 FDDI 其他网络

14. 一个栈的入栈序列是 A, B, C, D, E, 则栈的不可能的输出序列是? (C)

A. EDCBA

B. DECBA

C. DCEAB

D. ABCDE

堆栈分别是先进后出,后进先出,

选项 a 是 abcde 先入栈,然后依次出栈,正好是 edcba

选项 b 是 abcd 先依次入栈,然后 d 出栈, e 再入栈, e 出栈

选项 c 是错误的,不可能 a 先出栈

选项 d 是 a 入栈,然后 a 出栈; b 再入栈, b 出栈.依此类推

最后的结果选择 C。

15. 同一进程下的线程可以共享以下？（BD）

- A . stack
- B . data section
- C . register set
- D . file fd

知识点

线程共享的内容包括：

- 1.进程代码段
- 2.进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)
- 3.进程打开的文件描述符、
- 4.信号的处理器、
- 5.进程的当前目录和
- 6.进程用户 ID 与进程组 ID

线程独有的内容包括：

- 1.线程 ID
- 2.寄存器组的值
- 3.线程的堆栈
- 4.错误返回码
- 5.线程的信号屏蔽码

所以选择为 BD。

16. 对于派生类的构造函数，在定义对象时构造函数的执行顺序为？（D）

- 1：成员对象的构造函数
- 2：基类的构造函数
- 3：派生类本身的构造函数

- A . 123
- B . 231
- C . 321
- D . 213

17. 递归函数最终会结束，那么这个函数一定？（B）

- A. 使用了局部变量
- B. 有一个分支不调用自身
- C. 使用了全局变量或者使用了一个或多个参数
- D. 没有循环调用

18. 编译过程中，语法分析器的任务是（BCD）

- A. 分析单词是怎样构成的
- B. 分析单词串是如何构成语言和说明的
- C. 分析语句和说明是如何构成程序的
- D. 分析程序的结构

知识点

1.词法分析（lexical analysis）

词法分析是编译过程的第一个阶段。这个阶段的任务是从左到右的读取每个字符，然后根据构词规则识别单词。词法分析可以用 lex 等工具自动生成。

2. 语法分析 (syntax analysis)

语法分析是编译过程的一个逻辑阶段。语法分析在词法分析的基础上，将单词序列组合成各类语法短语，如“程序”，“语句”，“表达式”等等。语法分析程序判断程序在结构上是否正确。

3. 语义分析 (semantic analysis)

属于逻辑阶段。对源程序进行上下文有关性质的审查，类型检查。如赋值语句左右端类型匹配问题。

所以 BCD 都属于词法分析，选择结果为 BCD。

19. 同步机制应该遵循哪些基本准则？ (ABCD)

- A . 空闲让进
- B . 忙则等待
- C . 有限等待
- D . 让权等待

20. 进程进入等待状态有哪几种方式？ (D)

- A. CPU 调度给优先级更高的线程
- B. 阻塞的线程获得资源或者信号
- C. 在时间片轮转的情况下，如果时间片到了
- D. 获得 spinlock 未果

21. 设计模式中，属于结构型模式的有哪些？ (BC)

- A. 状态模式

- B. 装饰模式
- C. 代理模式
- D. 观察者模式

10. Android 最新技术

提示：考虑到大家使用下面文章链接的方便性，因此阳哥给每个链接生成了一个二维码，大家将二维码扫描即可

打开网页，阅读原文。

1. Android 架构思考(模块化、多进程)

http://blog.spinytech.com/2016/12/28/android_modularization/



2. Android 热修复

<http://mp.weixin.qq.com/s/GuzbU1M1LY1VKmN7PyVbHQ>



3. Android 性能优化

<http://www.jianshu.com/p/b3b09fa29f65>



4. Android 动态加载

<http://www.jianshu.com/p/b3b09fa29f65>



5. Android AOP

<http://mp.weixin.qq.com/s/SyFe2CgKW51ROAcFHd0a5Q>



6. MVP+Retrofit+RxJava 网络请求框架

<http://www.jianshu.com/p/7b839b7c5884>



7. RxJava

<http://www.jianshu.com/p/be806db4d1e7>



8. 安居客 Android 项目架构演进

<http://www.cnblogs.com/baronzhang/p/6442047.html>



注意：进入该网页后可能需要点击 back 键才能看到正文。

9. Android 进程保活

<http://www.jianshu.com/p/1da4541b70ad>



10. Android 状态栏

<http://mp.weixin.qq.com/s/J2hAWzR4ilJrbqIXEdDFjg>



11. Android Context 的种类

<https://possiblemobile.com/2013/06/context/>



附录：更新记录

时间	类型	操作人	备注
2015 年 5 月 20 号	创建文档	王震阳	新建
2015 年 8 月 7 号	完成文档	同上	第一版收尾
2015 年 8 月 8 号	公布网络	同上	地址： http://bbs.itheima.com/thread-223247-1-1.html
2015 年 10 月 13 号	添加试题	同上	改动较大，以此版为准
2015 年 11 月 24 号	添加 20 余道上海 Android2 期学生面试题	同上	改动很大升级版本号为 Version 2.0
2015 年 11 月 25 号	添加面试实战记录	同上	新添加面试实战内容
2015 年 11 月 26 号	添加项目技能点	同上	新添加 Android 项目知识
2015 年 11 月 29 号	添加开源框架原理	高大冬、徐鑫	修改版本号为 Version 2.5
2016 年 1 月 18 号	添加项目面试常见问题	高大冬、施鹏远、王震阳	修改版本号为 Version 2.6
2016 年 1 月 20 号	添加自己编写框架之	王震阳	修改版本号为 Version 2.7

	ViewUtils 框架		
2017-2-23	添加由郑晶晶、刘高赛、 卢晓等同事收集的面试题	同上	
2017-2-24	添加他人面试心得	同上	修改版本号为 V3.0
2017-2-25	添加网络部分内容	郑晶晶	版本号 V3.0
2017-2-26	添加 BAT 面试题	王震阳	版本号 V3.0
2017-2-26	添加 Android 最新技术	王震阳	版本号 V3.0

暂到此，未完待续！

2017 年 2 月 26 日 星期天 晴

上海市浦东新区航都路 18 号