目录

GitHub 托管

https://github.com/JackChan1999/Android-Interview

GitBook 在线阅读

在线阅读，PDF、ePub、Mobi电子书下载

https://www.gitbook.com/book/alleniverson/android-interview/details

Android 面试宝典

内容：JavaSE 基础，JavaSE 高级，Android 基础，Android 高级，Android 项目，项目面试常见问题，面试实战记录，BAT 面试题，Android 最新技术

下载地址：http://download.csdn.net/detail/axi295309066/9842371

Android 项目宝典

本项目宝典收录了一些经典的开源项目，希望借此丰富大家的学习资料，让大家在工作中更加游刃有余。

为了便于大家的学习，本人对所有项目进行测试运行，确认编译通过。并且对这些项目做了一些简单的分类，截取一些有代表性的图片，项目描述中附有百度网盘的下载链接。

本项目宝典以后还会陆续更新，如有不足之处，还请大家提出宝贵的建议。

下载地址：http://download.csdn.net/detail/axi295309066/9872667

Android面试

1. 史上最全 Android 面试资料集合
2. 2016Android某公司面试题
3. 国内一线互联网公司内部面试题库
4. BAT无线工程师面试流程详细解析
5. 阿里面经1
6. Android面试题整理
7. 2016Android某公司面试题
8. Android面试题集合
9. 一个五年Android 开发者百度、阿里、聚美、映客的面试心经
10. 扫清Android面试障碍--面试前的准备
11. Andorid-15k+的面试题
12. Android 开发工程师面试指南
13. 面试不失败-揭秘IT面试各个环节成功的内幕
14. 亲爱的面试官，这个我可没看过！（Android部分）
15. 115个Java面试题及回答
16. interview-about

目录

    - 迭代开发的时候如何向前兼容新旧接口?
    - 手把手教你如何解决as jar包冲突
    - Context原理分析
    - 解决ViewPager.setCurrentItem中间很多页面切换方案
    - 解决字体适配
    - 软键盘顶出去解决方案
    - 机型适配之痛
    - ViewPager和Fragment使用过程中会遇到哪些问题
- 与人事相关面试题
    - 人事面试宝典一之自我介绍
    - 人事面试宝典二之离职
    - 人事面试宝典
- 网络编程
    - Android客户端和服务端如何使用Token和Session
    - 推送原理
- Java面试题
    - 深拷贝浅拷贝
    - 数据库求差
    - Java面试题-1
    - Java面试题-2
    - Java高级软件工程师面试考纲
    - 66道经典的Java基础面试题集锦
    - 115个Java面试题及回答
    - J2SE基础面试核心内容
    - J2SE高级面试核心内容
- 面试技巧
    - 程序员面试宝典
    - 罗永浩新东方万字求职信
    - 我在面试中最喜欢问开发者的问题，和回答思路
- 经验分享
    - 我为什么要离开华为?
    - 工作三年后，我选择离开腾讯
    - 一个程序员的血泪史
    - 我为什么离开锤子科技?
    - 互联网巨头BAT3内部员工的真实状况
    - 扫清Android面试障碍
    - 史上最全 Android 面试资料集合
    - 2016年4月某公司面试题及面试流程
    - 2017届实习生招聘面经
    - 国内一线互联网公司内部面试题库
    - 互联网公司面试经验总结
    - 一个五年Android开发者百度、阿里、聚美、映客的面试心经
    - 腾讯公司程序员面试题及答案详解
    - 面试心得与总结：BAT、网易、蘑菇街
    - 阿里+百度+CVTE面经合集
    - 技术硬碰硬—阳哥带你玩转上海Android招聘市场
    - Android 曲折的求职之路
    - 杭州找 Android 工作的点点滴滴
    - 给培训班出来的一点不成熟的小建议

- Android 暑期实习生面试经验谈

关注我

- Email：619888095@qq.com
- CSDN博客：Allen Iverson
- 新浪微博：AndroidDeveloper
- GitHub：JackChan1999
- GitBook：alleniverson
- 个人博客：JackChan

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作!

微信赞赏支持	支付宝赞赏支持
	

Android视频教程

- 零基础入门Android语法与界面
- Kotlin系统入门与进阶
- 从Java到Kotlin系列精讲
- Kotlin打造完整电商APP 模块化+MVP+主流框架
- Android强化：服务与通信之广播、服务、蓝牙
- Gradle3.0自动化项目构建技术精讲+实战
- React Native技术精讲与高质量上线APP开发
- 双平台真实开发GitHub App React Native技术全面掌握
- Android应用发展趋势必备武器 热修复与插件化
- Android通用框架设计与完整电商APP开发
- 全程MVP手把手 打造IM即时通讯Android APP
- RxJava从源码到应用，移动端开发效率秒提速
- Android开发—高级开发专题系列全套课程
- BAT大咖助力 全面升级Android面试
- Android高级面试 10大开源框架源码解析
- BAT大厂APP架构演进实践与优化之路
- Android架构师之路 网络层架构设计与实战
- Android互动直播APP开发
- 从零开发Android视频点播APP
- 组件化封装思想实战Android App
- 带领新手快速开发Android App
- 快速开发轻量级App

BAT大咖助力，全面升级Android面试

安卓面试全新升级课程，已经帮助很多同学在短时间内拿到了满意的offer，作为Android开发求职路上快速赢取满意offer的必备课程，不管你是什么级别的工程师，都可以通过这门课程的学习，轻松快速搞定Android面试，赢取满意offer！

课程章节

第1章 课程介绍

本章带你了解面试过程中会遇到的问题，个人应该摆正的心态，以及面试官最为看重你的解决问题的思路。

第2章 一线互联网公司初中高Android开发工程师的技能要求

本章对一线互联网公司各个级别Android开发工程师的招聘需求进行深入分析，并带大家清晰完整的了解面试复习与准备思路，做到有的放矢，有侧重点的进行复习与准备。

第3章 Android基础相关面试问题

本章主要从Android的五大组件给大家讲解，讲解的思路主要通过一道道面试题的思路带大家逐题破解，以点带面来复习巩固android基础知识，并会在每道课程结尾处给大家总结补充一些知识点，主要讲解：Fragment 、Service 、Binder 、Activity、 BroadCast 、WebView安全漏洞、android6.0/7.0增加的新功能。...

第4章 异步消息处理机制相关面试问题

Android中异步消息处理在面试中是一定会被问到的，在实战过程中也是非常重要的一个开发手段，我们会从handler、Asynctask、IntentService、HandlerThread给大家详细讲解，主要通过使用、源码机制来给大家深入分析异步消息处理机制。

第5章 View相关面试问题

本章主要从view的绘制、事件分发、listview、属性动画来给大家进行讲解，自定义控件在日常开发中是也是必不可少的，本章主旨就是能了解其中的原理，从而在面试中遇到同类问题能给出相应的思路。

第6章 Android项目构建相关面试问题

开发过程中项目的构建是很重要一环，也是检验你是不是一个合格的android开发工程师的标志，面试中也会经常问到，在这里我们主要通过Android的编译打包、Proguard混淆、git的使用、gradle、渠道包这五个部分给大家分析，带大家了解Android构建的全过程，从而轻松应对这类问题的各种面试与开发。...

第7章 开源框架相关面试问题

本章主要带大家分析现在热门的开源框架主要有网络框架：retrofit/okhttp/volley，图片加载框架fresco/gilde/uil，IOC框架：butterknife/dagger2，分析的思路是从使用到深入源码分析，开源框架可以说是一个高级工程师的试金石，如果对于以上框架很熟悉，并能画出流程图，会让面试官对于刮目相看。...

第8章 Android异常与性能优化相关面试问题

随着现在的android开发业务逻辑不断扩大，对于手机的性能也提出了很高的要求，所以一款app在性能上如果能区别其他app也将脱颖而出，同样如果候选人能对性能优化很熟悉，也将在面试中脱颖而出，本章主要从UI卡顿、内存管理、内存泄漏这几个角度带大家分析性能优化。...

第9章 热门前沿知识相关面试问题

现在Android发展越来越快，对于一些前沿的知识，在面试中我们也是需要做到了解，这章从Android的插件化、热更新、rxjava、进程保活，组件化，签名过程，应用沙盒等方面给大家讲解，主要想做到扩大大家的知识面，让面试官看到你对android的热爱。

第10章 Java高级技术点面试问题

在Android的面试中，面试官通常缺少不了会问一下Java高级技术，本章就会为大家讲解Java相关高级技术面试点，包括GC/回收算法/堆栈/、反射/编译时vs运行时、注解（结合android annotation库）、范型、线程池/并发编程、Socket、IO/NIO、集合框架、类加载器、异常、继承/组合/多态、引用类型/内存泄漏、java虚拟机/d...

第11章 设计模式相关面试问题

设计模式是高级开发者的必备知识，面试中也是经常被问到，本章将结合Android使用场景，讲解常用的设计模式，让大家既掌握Android下设计模式的使用，又可轻松应对面试中关于设计模式的面试问题。包括观察者模式、动态代理 、工厂、策略类、装饰、桥接、单例等常用设计模式。...

第12章 网络协议相关面试问题

网络编程无论在开发中还是在面试中都是非常重要的，在面试中尤其对网络协议问的比较多，本章将会对网络协议进行讲解，包括https/http、dns、tcp/ip以及加密算法。

第13章 算法相关面试问题

算法作为编程的重要部分，在BAT等大公司基本是必考项，本章将结合案例为大家讲解常用常考的算法面试问题，帮助大家提高算法能力的同时轻松应对算法相关的面试。

第14章 课程总结

本章主要总结面试过程的相关技术点。同时也将面试的内容做一个归纳总结，最后非常感谢大家的支持，课程中遇到任何问题都可以在问答区提问，我在那里等着大家，有问必答，也祝愿大家都能尽早的获得一份心仪的offer。

Android高级面试，10大开源框架源码解析

编程最好的学习方法是阅读顶尖工程师的源码！本课程将带你深度剖析Android主流开源框架的源码，让你全面掌握框架的使用场景、内部机制、构造原理、核心类、架构与设计思想等，提升你的代码阅读与分析能力、提高代码设计能力及改造能力，快速突破技术瓶颈，轻松应对Android高级面试与技术难题！

课程章节

第1章 课程介绍

编程最好的学习方法是阅读顶级工程师的源码！本课程将带你深度剖析Android主流开源框架的源码，让你全面掌握框架的使用场景、内部机制、构造原理、核心类、架构与设计思想等，提升你的代码阅读与分析能力、提高代码设计能力及改造能力，快速突破技术瓶颈，轻松应对Android高级面试与技术难题！ ...

- 1-1 课程导学

第2章 Okhttp网络库深入解析和相关面试题分析

本章主要先通过分析OKhttp的简单使用，对于OKhttp的调度器、拦截器、缓存策略、连接池等进行了相应的源码和原理分析，并对于socket、websocket、http缓存、多线程下载、文件下载、https等经典Android面试题进行分析。

- 2-1 okhttp框架流程分析
- 2-2 okhttp同步请求方法
- 2-3 okhttp异步请求方法
- 2-4 okhttp同步请求流程和源码分析
- 2-5 okhttp异步请求流程和源码分析-1
- 2-6 okhttp异步请求流程和源码分析-2
- 2-7 okhttp任务调度核心类dispatcher解析-1
- 2-8 okhttp任务调度核心类dispatcher解析-2
- 2-9 okhttp拦截器流程
- 2-10 okhttp拦截器链介绍
- 2-11 okhttp之RetryAndFollowUpInterceptor解析
- 2-12 okhttp之BridgeInterceptor解析
- 2-13 okhttp缓存策略源码分析：put方法
- 2-14 okhttp缓存策略源码分析：get方法
- 2-15 okhttp拦截器之CacheInterceptor解析
- 2-16 okhttp拦截器之ConnectInterceptor解析-1
- 2-17 okhttp拦截器之ConnectInterceptor解析-2
- 2-18 okhttp连接池：put,get方法
- 2-19 okhttp连接池：connection回收
- 2-20 okhttp拦截器之CallServerInterceptor解析
- 2-21 okhttp面试: Socket-1
- 2-22 okhttp面试: Socket-2
- 2-23 okhttp面试: HttpClient&HttpUrlConnection
- 2-24 okhttp面试: OkHttp来实现WebSocket连接
- 2-25 okhttp面试: WebSocket&轮询相关
- 2-26 okhttp面试: Http缓存、Etag等标示作用

- 2-27 okhttp面试: 断点续传原理&Okhttp如何实现
- 2-28 okhttp面试：多线程下载
- 2-29 okhttp面试：文件上传&Okhttp如何处理文件上传
- 2-30 okhttp面试：如何解析Json类型数据
- 2-31 okhttp面试：Https / 对称加密&不对称加密

第3章 Retrofit网络库深入解析和相关面试题分析

本章主要先通过分析retrofit的使用，对于retrofit的接口、动态代理、适配工厂、数据转换等进行相应的源码和原理分析，并对于retrofit的设计模式、线程切换、Hook、MVC和MVP架构、SP跨进程问题等经典Android面试题进行分析。

- 3-1 retrofit流程分析
- 3-2 retrofit概述
- 3-3 retrofit官网例子解析
- 3-4 retrofit请求过程7步骤详解
- 3-5 静态代理模式讲解
- 3-6 动态代理模式讲解
- 3-7 retrofit网络通信流程8步骤&7个关键成员变量解析
- 3-8 retrofit中builder构建者模式&builder内部类解析
- 3-9 retrofit中baseurl / converter / calladapter解析
- 3-10 retrofit中build方法完成retrofit对象创建流程解析
- 3-11 retrofit中RxjavaCallAdapterFactory内部构造与工作原理解析
- 3-12 retrofit中网络请求接口实例解析
- 3-13 retrofit中serviceMethod对象解析
- 3-14 retrofit中okHttpCall对象和adapt返回对象解析
- 3-15 retrofit中同步请求&重要参数解析
- 3-16 retrofit中异步请求解析
- 3-17 retrofit设计模式解析-1：构建者模式
- 3-18 retrofit设计模式解析-2：工厂模式
- 3-19 retrofit设计模式解析-3：外观模式
- 3-20 retrofit设计模式解析-4：策略模式
- 3-21 retrofit设计模式解析-5：适配器模式
- 3-22 retrofit设计模式解析-6：动态代理模式 / 观察者
- 3-23 retrofit面试题：retfrofit线程切换（异步机制Looper)
- 3-24 retrofit面试题：rxjava和retrofit如何结合进行网络请求
- 3-25 retrofit面试题：Hook与动态代理
- 3-26 retrofit面试题：Android MVC架构优势和缺点
- 3-27 retrofit面试题：MVP优点和缺点
- 3-28 retrofit面试题：sp跨进程&apply和commit方法

第4章 Glide图片库深入解析和相关面试题分析

本章主要先通过分析Glide的使用，对于glide的内存和硬盘缓存、加载策略、如何进行图片网络请求等方面，并将重点放在梳理整个Glide请求的流程，最后对于bitmap、性能优化OOM和三级缓存、Lrucache等Android面试题进行分析。

- 4-1 glide框架流程分析
- 4-2 glide框架介绍-1
- 4-3 glide框架介绍-2

- 4-4 glide图片加载流程和源码分析-1：with方法（requestManager获取）
- 4-5 glide图片加载流程和源码分析-2：with方法（requestManagerRetriever的get方法）
- 4-6 glide图片加载流程和源码分析-3：load方法
- 4-7 glide图片加载流程和源码分析-4：into方法（buildTarget）
- 4-8 glide图片加载流程和源码分析-5：into方法（request建立和begin方法）
- 4-9 glide图片加载流程和源码分析-6：into方法（Loadprovider）
- 4-10 glide图片加载流程和源码分析-7：into方法（硬盘缓存 / 内存缓存）
- 4-11 glide图片加载流程和源码分析-8：into方法（内存缓存的读取）
- 4-12 glide图片加载流程和源码分析-9：into方法（内存缓存的写入）
- 4-13 Glide面试一：bitmap&oom&优化bitmap
- 4-14 Glide面试二：三级缓存&lrucache

第5章 LeakCanary内存泄漏框架解析和相关面试题分析

本章主要先通过leakcanary使用，然后分析内存泄漏产生原因，并对于Leakcanary如何进行泄漏Activity收集策略、转换内存快照、定位内存泄漏位置等分析，最后对于现在业界比较关心的UI流畅度和性能数据上报等进行对应分析。

- 5-1 leakcanary预备知识：android性能优化&Gcroots
- 5-2 leakcanary内存框架：内存泄漏基础&为什么需要leakcanary
- 5-3 android常见内存泄漏分析-1：单例VS非静态内部类
- 5-4 android常见内存泄漏分析-2：handler&解决办法
- 5-5 android常见内存泄漏分析-3：线程&WebView
- 5-6 leakcanary原理分析-1：Leakcanary原理概述和弱引用 / 引用队列
- 5-7 leakcanary原理分析-2：ActivityRefWatcher如何监视Activity
- 5-8 leakcanary原理分析-3：.hprof转换snapshot
- 5-9 leakcanary原理分析-4：查找内存泄漏引用和最短泄漏路径
- 5-10 leakcanary面试题：Application&内存
- 5-11 leakcanary面试题：性能数据上报：网络流量和冷启动
- 5-12 leakcanary面试题：性能数据上报：UI卡顿和内存占用

第6章 butterknife依赖注入框架源码解析

本章从butterknife的基本使用讲起，首先会介绍框架相关注解和APT知识点，然后开始逐步分析butterknife源码，并逐步理清butterknife注入框架的原理，最后提炼butterknife中有关android面试相关问题。

- 6-1 butterknife的引言和基本使用
- 6-2 butterknife原理必备知识点1：注解
- 6-3 butterknife原理必备知识点2：APT工作原理
- 6-4 butterknife原理必备知识点3：反射+运行时注解举例
- 6-5 butterknife原理分析-1：注解处理器如何处理注解和保存注解
- 6-6 butterknife原理分析-2：如何生成findviewByID代码

第7章 blockcanary UI卡顿优化框架源码解析

本章会从blockcanary基本使用讲起，首先会简单介绍ActivityThread / handler / looper相关框架知识点，然后通过分析blockcanary源码，逐步理清blockcanary如何解决UI卡顿的原理，最后会提炼blockcanary中有关android面试相关问题,并总结android性能优化相关问题。...

- 7-1 blockcanary背景 / UI卡顿原理 / UI卡顿常见原因

- 7-2 blockcanary使用 / 阀值参数
- 7-3 blockcanary核心原理实现和流程图简述
- 7-4 blockcanary源码解析-1：框架初始化
- 7-5 blockcanary源码解析-2：stacksampler / cpusampler / start方法
- 7-6 blockcanary面试一：anr场景 / 原因 / 解决
- 7-7 blockcanary面试二：watchdog-anr 如何检测anr
- 7-8 blockcanary面试三：new Thread开启线程的4点弊端
- 7-9 blockcanary面试四：线程间通信：子线程--UI线程
- 7-10 blockcanary面试五：主线程--子线程(handlerThread-IntentService)
- 7-11 blockcanary面试六：多进程的4点好处与问题 / voliate关键字
- 7-12 blockcanary面试七：voliate关键字和单例的写法

第8章 eventbus异步框架源码解析

本章会从eventbus的基本用法开始讲起，主要包括Event、Subscriber、Publisher、ThreadMode几大部分，并结合handler、组件间传递等消息知识点深入分析，然后对比分析eventbus3.0和2.0的区别，并结合eventbus在android面试中遇到的高频问题，对eventbus框架进行总结。...

- 8-1 eventbus框架核心概念：事件传递 / EventBus的优点 / 传统handler通信的两种方式
- 8-2 eventbus框架基本用法
- 8-3 eventbus框架源码解析-1：EventBus对象构建 / 如何进行线程调度
- 8-4 eventbus框架源码解析-2 subscribe注解 / threadMode
- 8-5 eventbus框架源码解析-3：register订阅(上)
- 8-6 eventbus框架源码解析-4：register订阅（中）
- 8-7 eventbus框架源码解析-5：register订阅（下）
- 8-8 eventbus框架源码解析-6：subscribe方法完成订阅（上）
- 8-9 eventbus框架源码解析-7：subscribe方法完成订阅（下）
- 8-10 eventbus框架源码解析-8：发送事件post

第9章 dagger2依赖注入框架源码解析

本章从dagger2的基本使用讲起，首先会介绍框架相关依赖注入的知识点，然后逐步分析dagger2源码，并逐步理清dagger2注入框架原理，并对比分析dagger2与dagger的区别，最后会根据android面试相关问题，给大家总结dagger2的相关知识点。

- 9-1 dagger2引言：依赖注入和使用场景
- 9-2 dagger2四种注入方式和依赖注入总结
- 9-3 dagger2的四种基本注解：@inject注解
- 9-4 dagger2的四种基本注解：@component注解
- 9-5 dagger2的inject和component注解实例和源码分析
- 9-6 dagger2的@Module和@Provides注解
- 9-7 dagger2的@Module和@Provides注解实例和代码分析

第10章 rxjava异步框架源码解析

本章会从rxjava的基本使用讲起：主要包括观察者模式、操作符、线程控制等，然后逐步分析rxjava中的响应式编程原理，最后会结合rxjava在android面试中遇到的高频面试问题，给大家总结rxjava相关知识。

- 10-1 rxjava基本用法和观察者模式：01-传统观察者模式
- 10-2 rxjava观察者模式和基本用法

- 10-3 rxjava如何创建Observable&observer / subscriber
- 10-4 rxjava如何创建subscriber以及如何完成订阅
- 10-5 rxjava操作符之map基本使用
- 10-6 rxjava操作符之map源码探究：lift
- 10-7 rxjava操作符之flatmap
- 10-8 rxjava线程控制：多线程编程准则&Rxjava如何处理多线程&&Schedulers
- 10-9 rxjava线程控制：两个小例子&observeOn和SubscribeOn
- 10-10 rxjava线程控制：SubscribeOn源码剖析
- 10-11 rxjava线程控制：ObserveOn源码剖析&&subscribeOn可以调用几次

第11章 picasso图片框架源码解析

本章从picasso基本用法和配置讲起，逐步分析picasso的源码，并从DownLoader，Dispatcher,service线程池等核心类进行分析，最后根据picasso流程图进行总结，并给大家提炼android面试中有关picasso框架的问题。

- 11-1 picasso框架基本使用API
- 11-2 picasso源码with方法：内存缓存Lrucache和线程池的调度
- 11-3 piacsso源码with：dispatcher如何完成线程切换
- 11-4 picasso源码with：NetworkRequestHandler处理图片请求和回调
- 11-5 picasso源码load方法
- 11-6 picasso源码into方法：Action&BitmapHunter
- 11-7 picasso源码into方法：线程池&PicassoFutureTask
- 11-8 picasso源码into：线程开启如何执行图片加载请求?
- 11-9 picasso源码into：Okhttp和UrlConnectionDownloader下载图片
- 11-10 picasso源码into方法：完成加载

第12章 课程总结

本章将通过对Android面试技巧的梳理，帮助大家整体的认知和提高Android面试能力以及需要做的面试准备等，希望能对大家的面试有所帮助！最后非常感谢大家对课程的认可和支持，祝愿你们都能找到好工作。收到你们的Offer消息，是做好这门课程最大的动力。...

- 12-1 Android面试技巧梳理

Android面试题

Android基本常识

1. 写10个简单的linux命令

命令	作用
mkdir	创建文件夹
rmdir	删除文件夹
mv	移动文件
rm	删除文件
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

```
|  build.gradle       项目Gradle构建脚本
|  gradle.properties  项目Gradle属性文件
|  gradlew            在没有安装gradle的pc上使用,没用
|  gradlew.bat        在没有安装gradle的pc上使用,没用
|  local.properties   指定sdk所在目录
|  settings.gradle    项目Gradle设置文件
|
├─.gradle
├─.idea
├─app
|  |  .gitignore   git忽略文件列表
|  |  app.iml  临时文件,不需要关心
|  |  build.gradle  Module Gradle构建脚本
|  |  proguard-rules.pro    proguard混淆规则
|  |
|  ├─build  构建目录，相当于Eclipse中默认Java工程的bin目录。编译生成的apk在此目录
|  ├─libs   依赖包
|  └─src
|      ├─androidTest  测试相关代码文件夹
|      |  └─java
|      |      └─com
|      |          └─itheima
|      |              └─myapplication
|      |                      ApplicationTest.java
```

```
|       |
|       └main
|           |  AndroidManifest.xml   清单文件
|           |
|           ├assets
|           ├aidl
|           ├java       项目源码
|           |  └com
|           |      └itheima
|           |          └myapplication
|           |                  MainActivity.java
|           |
|           ├jni    放置c代码
|           ├jniLibs    放置so库
|           ├assets
|           └res        资源文件
|               ├drawable  .9图片只能放到drawable目录下
|               ├layout
|               |      activity_main.xml
|               |
|               ├menu
|               |      menu_main.xml
|               |
|               ├mipmap-hdpi            类似drawable-hdpi
|               |      ic_launcher.png
|               |
|               ├mipmap-mdpi            类似drawable-mdpi
|               |      ic_launcher.png
|               |
|               ├mipmap-xhdpi           类似drawable-xdpi
|               |      ic_launcher.png
|               |
|               ├mipmap-xxhdpi          类似drawable-xxdpi
|               |      ic_launcher.png
|               |
|               ├values
|               |      dimens.xml
|               |      strings.xml
|               |      styles.xml
|               |
|               └values-w820dp
|                       dimens.xml
|
├build
└gradle
    └wrapper   gradle wrapper可以看作是对gradle的封装，它可以使得在没有安装gradle的电脑上也可以使用Gradle进行构建
            gradle-wrapper.jar
            gradle-wrapper.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()、runonUITread()、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：

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

缺点：

- 安全问题、隐私问题
- 同质化严重
- 运营商对Android手机仍然有影响
- 山寨化严重
- 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?

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…）	
EFICCID	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
DFGSM	7F20	GSM目录	备注：根据ETSIGSM09.91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	
EFLP语言选择	6F05	LP	语言选择文件	包含一种或多种语言编码
			国际移动用户	包含SIM卡所对应的号段，比如46000代表135

			识别符	－139号段、46002代表1340－1348
EFKC语音加密密钥	6F20	Kc	计算密钥	用于SIM卡的加密、解密
EFPLMNsel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
EFHPLMN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
EFACMmax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
EFSST 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
EFACM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
EFGID1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
EFGID2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
EFPUCT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
EFCBMI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
EFSPN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
EFCBMID	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
EFSUME	6F54	SUME	建立菜单单元	建立SIM卡中的菜单
EFBCCH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
EFACC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
EFFPLMN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
EFLOCI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
EFAD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）
EFPHASE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
DFTELECOM	7F10	电信目录		
EFADN缩位拨号	6F3A	AND	电话簿	用于将电话记录存放在SIM卡中

拨号				
EFFDN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
EFSMS短消息	6F3C	SMS	短消息	用于将短消息记录存放在SIM卡中
EFCCP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
EFMSISDN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
EFSMSP短信息参数	6F42	SMSP	短消息业务参数	包括短信中心号码等信息
EFSMSS短信息状态	6F43	SMSS	短消息状态	这个标识是用来控制流量的
EFLND最后拨号	6F44	LND	最后拨叫号码	存储最后拨叫号码
EFExt1扩展文件1	6F4A	EXT1	扩展文件1	包括AND，MSISDN或LND的扩展数据
EFExt2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AndroidSDK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AndroidAsset 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 debuging 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。对不喜欢命今操作方式的人来说是一个不错的选择。

12.ddms 和traceview的区别

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。

13.补充知识：TraceView的使用

13.1 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:

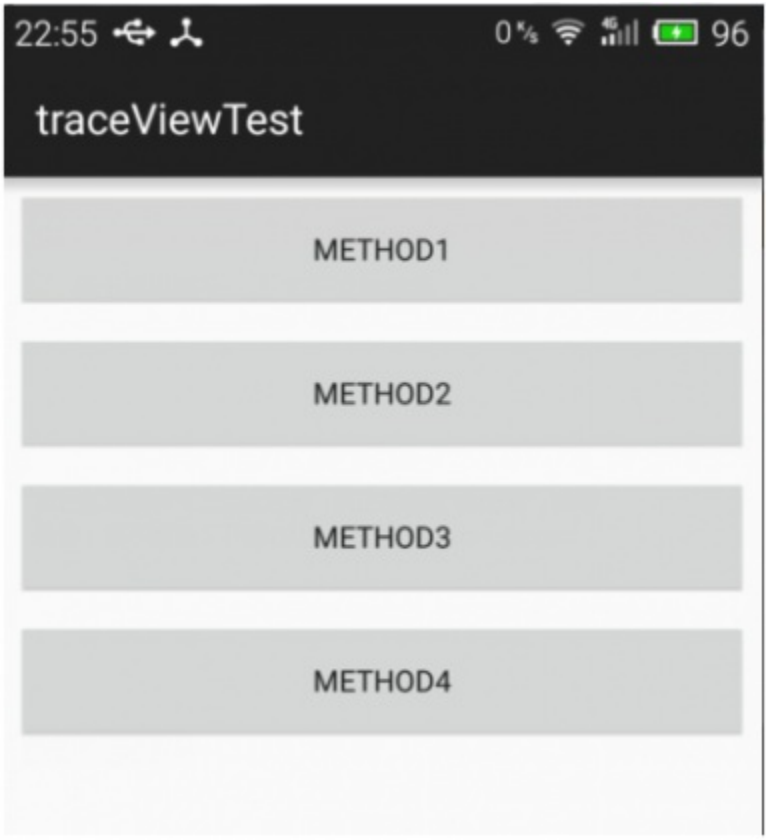
开发者在一些关键代码段开始前调用Android SDK中Debug类的startMethodTracing函数，并在关键代码段结束前调用stopMethodTracing函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是Java线程）的函数执行情况，并将采集数据保存到/mnt/sdcard/下的一个文件中。开发者然后需要利用SDK中的Traceview工具来分析这些数据。

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向读者介绍Debug类以及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) {
```

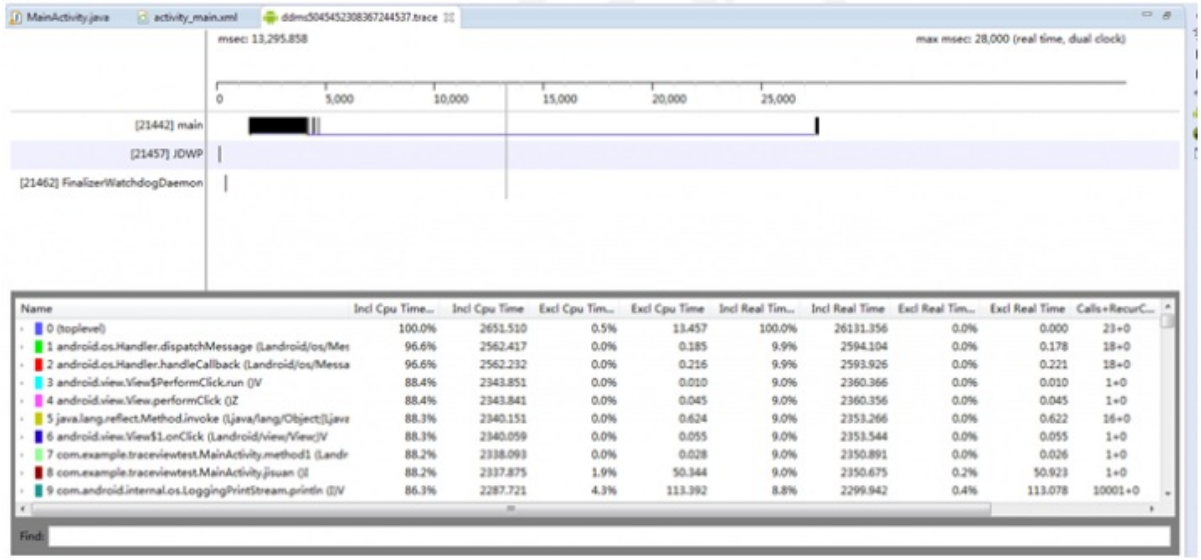
```
                int result = jisuan();
                System.out.println(result);
        }
        private int jisuan() {
                for (int i = 0; i < 10000; i++) {
                        System.out.println(i);
                }
                return 1;
        }

        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。

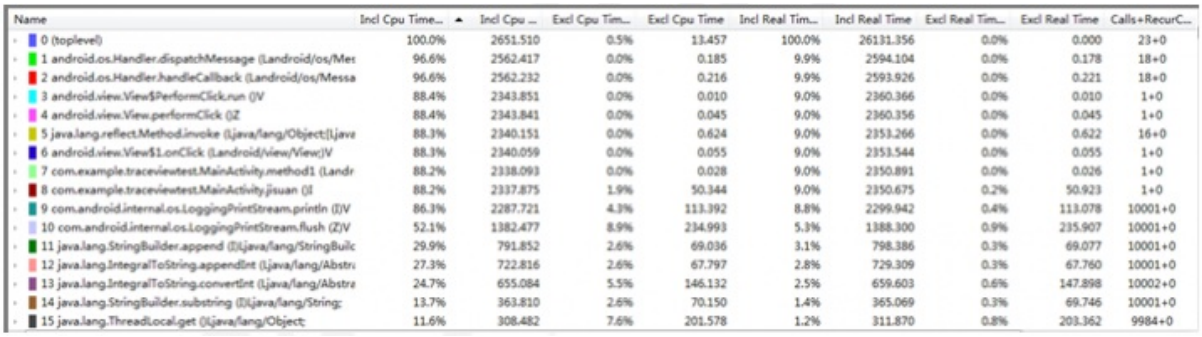


我们主要是分析main线程。

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：



通过分析发现：method1最耗时，耗时2338毫秒。

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。

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FragmentActivitiy，ListActivity，TabAcitivty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.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) {
        super.onSaveInstanceState(outState);
}
```

4.两个Activity之间跳转时必然会执行的是哪几个方法？

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。

5.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。

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。

6.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"
```

7.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。

8.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：

- standard 默认启动模式
- singleTop 栈顶复用模式
- singleTask 栈内复用模式
- singleInstance 单例模式

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

下面我们结合实例一一介绍这四种lanchMode：

8.1 standard

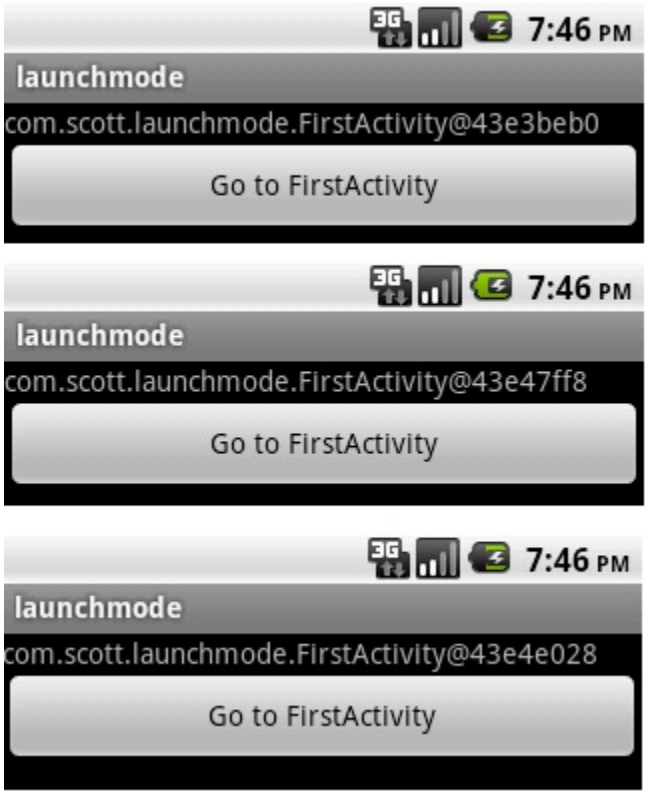
standard模式是默认的启动模式，不用为<activity>配置android:launchMode属性即可，当然也可以指定值为standard。

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：

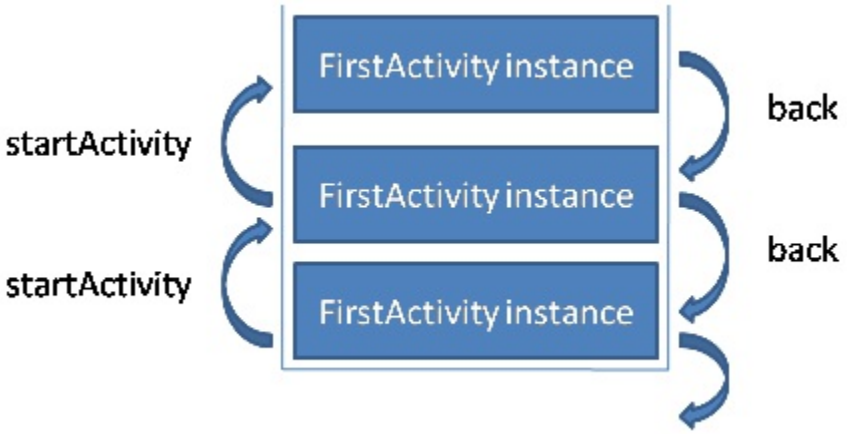
```
public class FirstActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first);
        TextView textView = (TextView) findViewById(R.id.tv);
        textView.setText(this.toString());
        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：

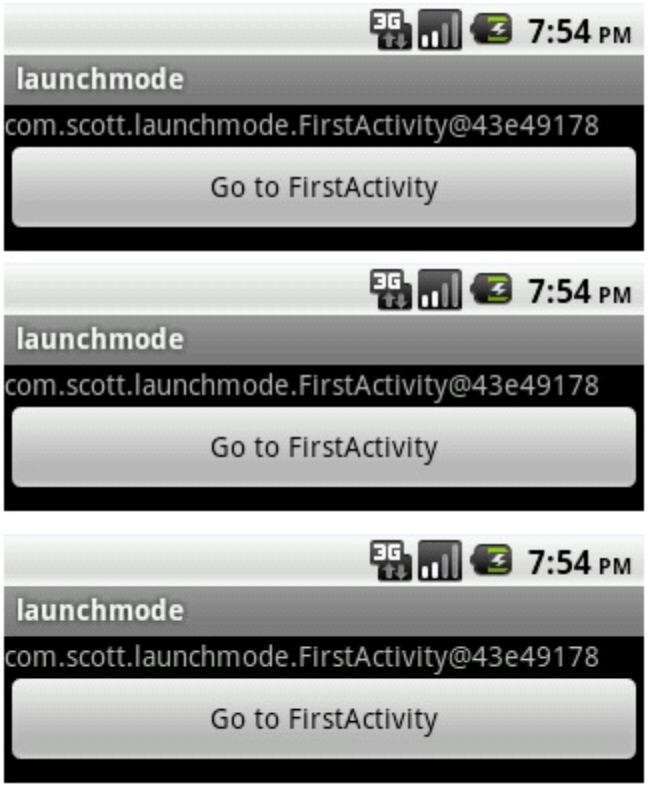


如图所示，每次跳转系统都会在task中生成一个新的FirstActivity实例，并且放于栈结构的顶部，当我们按下后退键时，才能看到原来的FirstActivity实例。

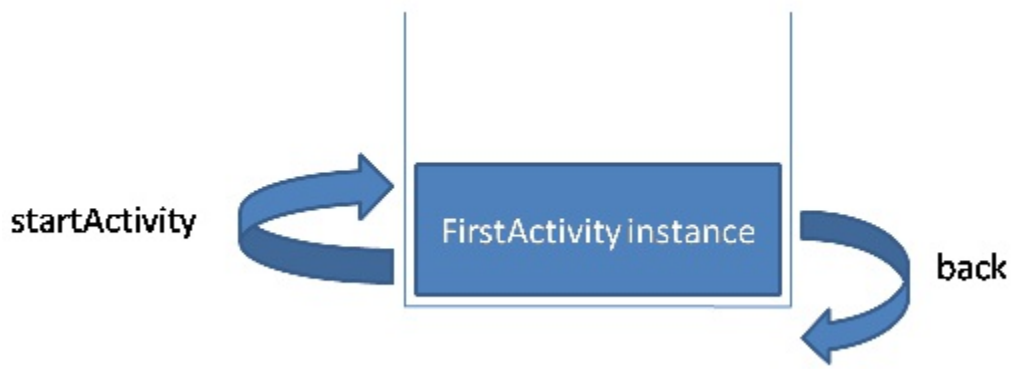
这就是standard启动模式，不管有没有已存在的实例，都生成新的实例。

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：



正如上图所示，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：

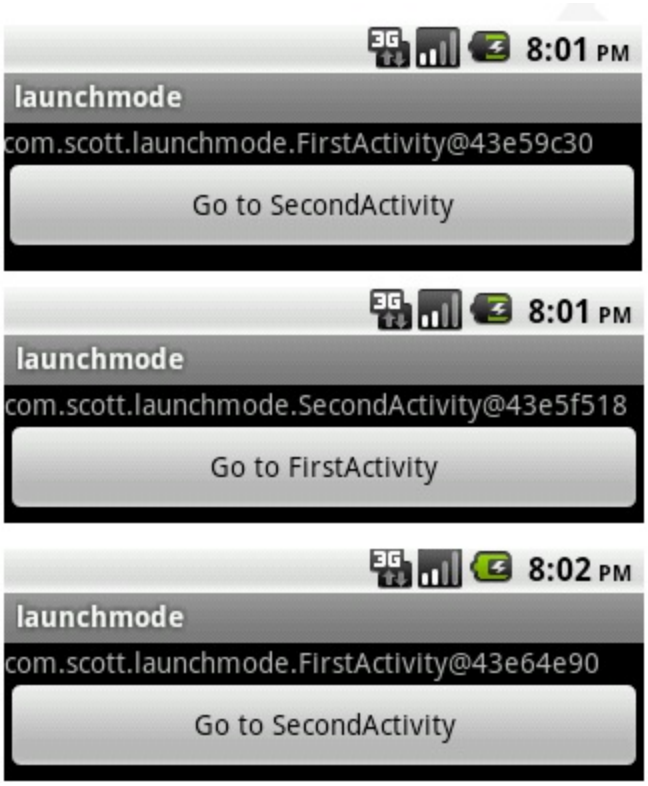
```
public class SecondActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.second);
                TextView textView = (TextView) findViewById(R.id.tv);
        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);
            }
        });
        }
 }
```

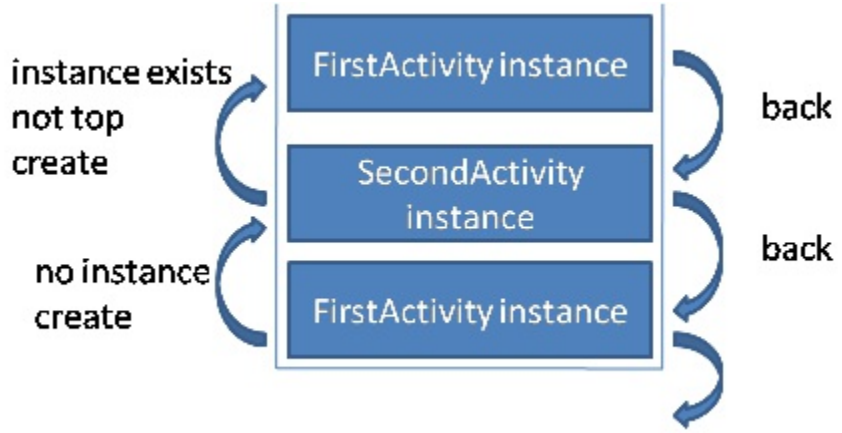
然后将之前的FirstActivity跳转代码改为：

```
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的序列号是不同的，证明从SecondActivity跳转到FirstActivity时生成了新的FirstActivity实例。原理图如下：

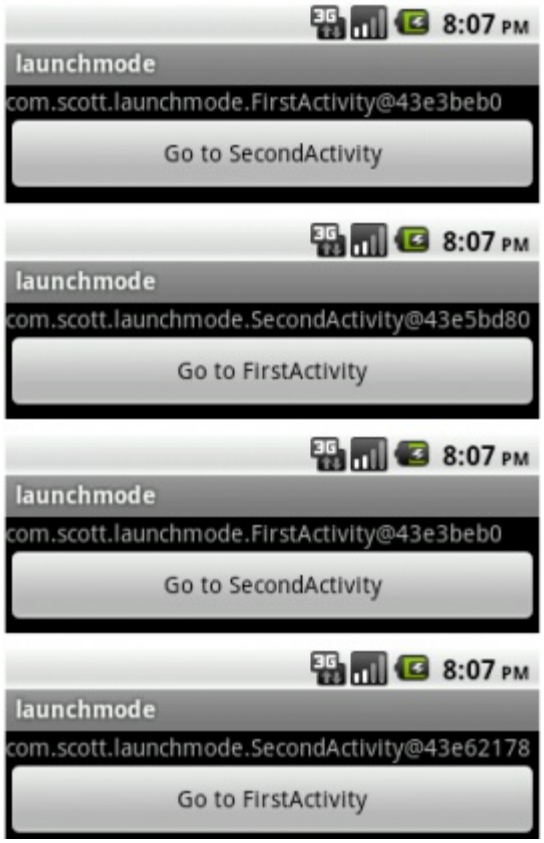


我们看到，当从SecondActivity跳转到FirstActivity时，系统发现存在有FirstActivity实例,但不是位于栈顶，于是重新生成一个实例。

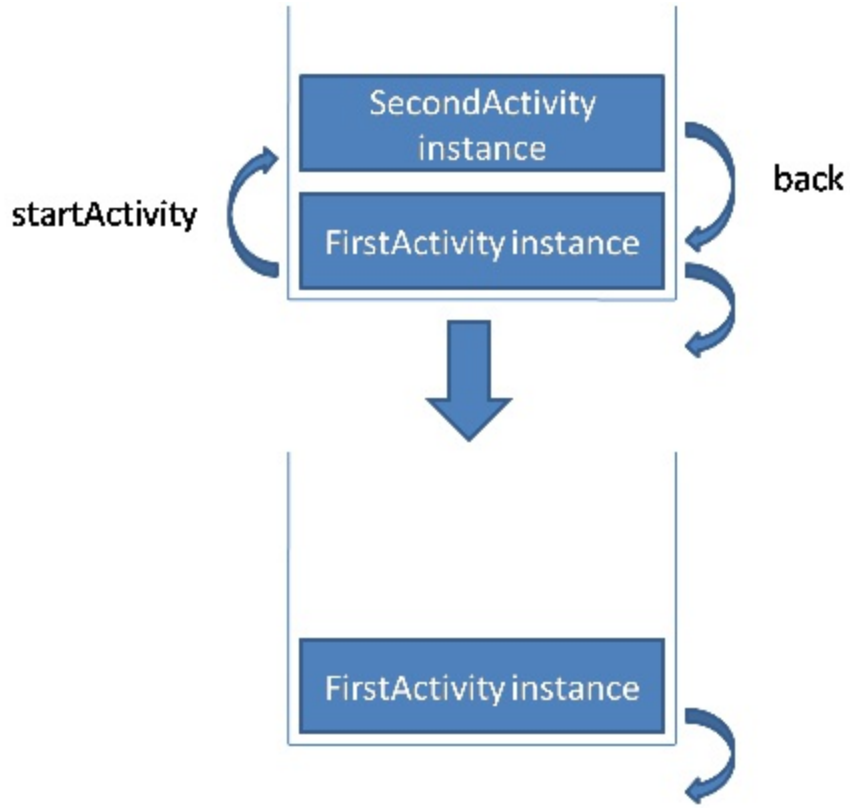
这就是singleTop启动模式，如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例。

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。但是如果我们有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，如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前。

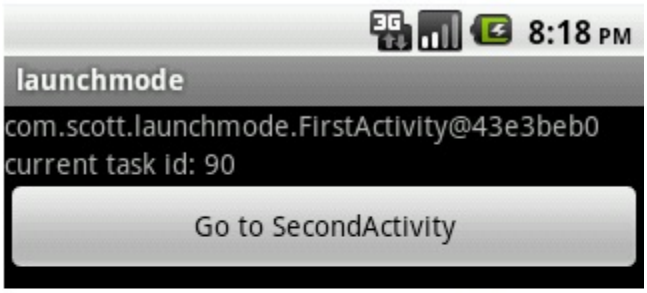
8.4 singleInstance

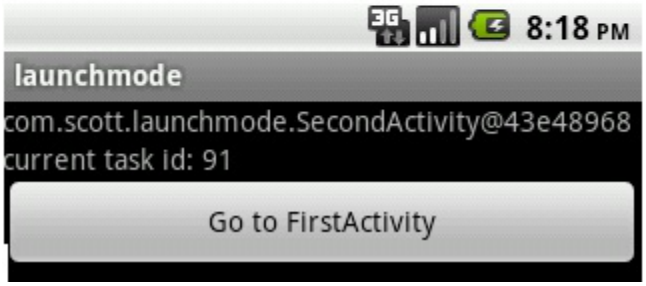
这种启动模式比较特殊，因为它会启用一个新的栈结构，将Activity放置于这个新的栈结构中，并保证不再有其他Activity实例进入。

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：

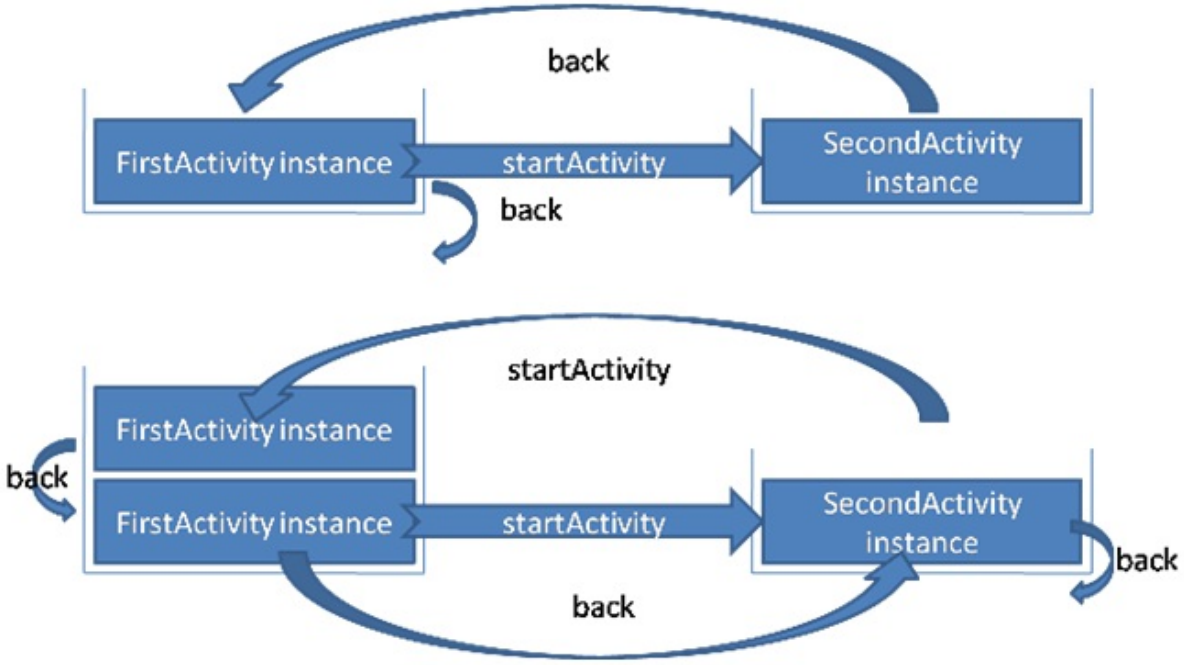
```
TextView taskIdView = (TextView) findViewById(R.id.taskIdView);
taskIdView.setText("current task id: " + this.getTaskId());
```

然后我们再演示一下这个流程：



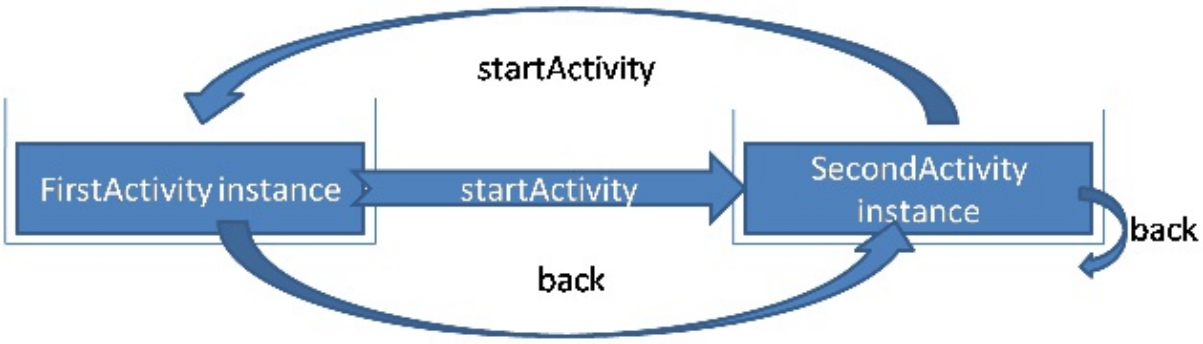


我们发现这两个Activity实例分别被放置在不同的栈结构中，关于singleInstance的原理图如下：



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：



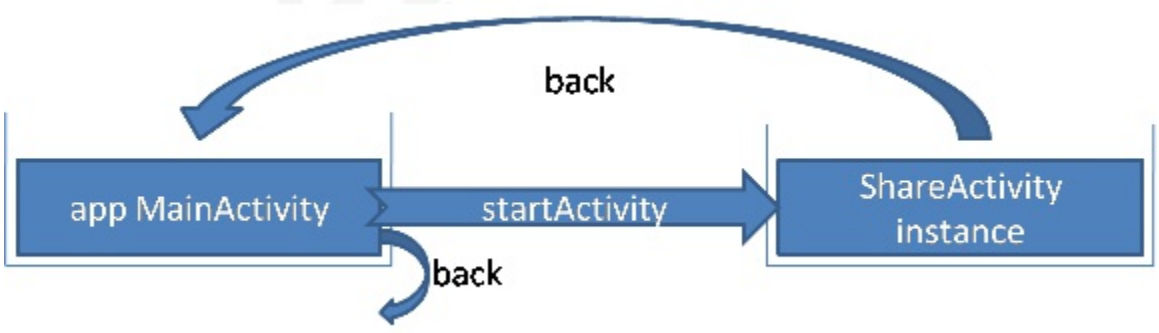
singleInstance启动模式可能是最复杂的一种模式，为了帮助大家理解，我举一个例子，假如我们有一个share应用，其中的ShareActivity是入口Activity，也是可供其他应用调用的Activity，我们把这个Activity的启动模式设置为singleInstance，然后在其他应用中调用。我们编辑ShareActivity的配置：

```
<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做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：



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servic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。

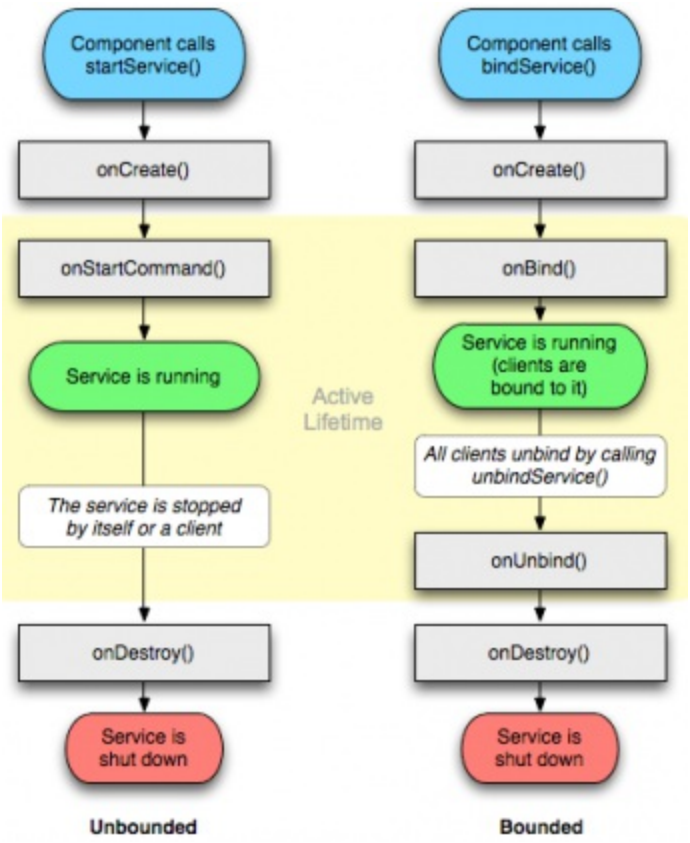
非绑定模式：当第一次调用startService的时候执行的方法依次为onCreate()、onStartCommand()，当Service关闭的时候调用onDestory方法。

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。

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；

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；

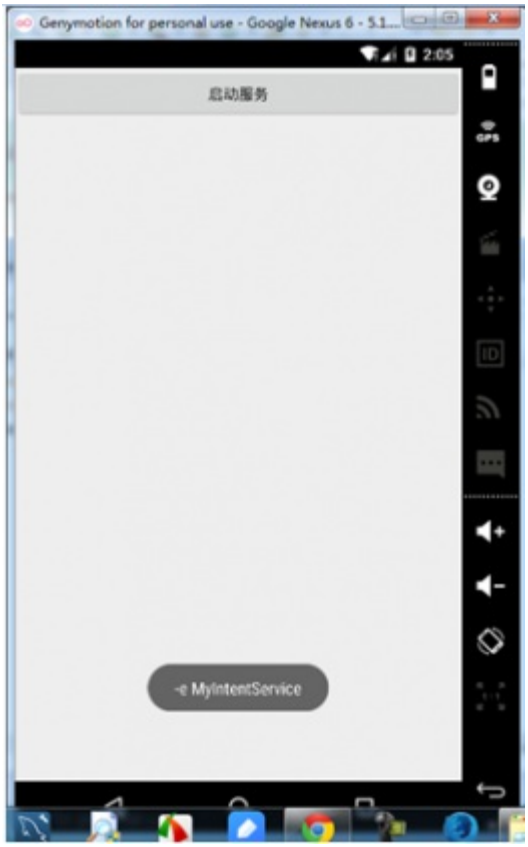
使用IntentService

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.javapublic 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。不过兄弟俩各有各自的本领，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，mainThread （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。

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

a. 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；

b. 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；

c. 最好也不要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；

ContentProvider

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

第三方可以通过ContentResolver来访问该Provider。

2.请介绍下Android的数据存储方式

- File存储
- SharedPreference存储
- ContentProvider存储
- SQLiteDataBase存储
- 网络存储

3.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。

4、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

Android中的布局

1.Android中常用的布局都有哪些

- FrameLayout
- RelativeLayout
- LinearLayout
- AbsoluteLayout
- TableLayout
- GridLayout

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。

ListView

1.ListView如何提高其效率？

- 复用ConvertView
- 自定义静态类ViewHolder
- 使用分页加载
- 使用WeakRefrence引用ImageView对象

2.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。

3.ListView如何实现分页加载

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)

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（flgin）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。

4.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。

5.ListView如何定位到指定位置

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

6.当在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。

```
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);
}
```

7.ListView中如何优化图片

图片的优化策略比较多。

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:

- 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。
- 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
- 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。
- 在getView中做图片转换时，产生的中间变量一定及时释放

异步加载图片基本思想：

- 先从内存缓存中获取图片显示（内存缓冲）
- 获取不到的话从SD卡里获取（SD卡缓冲）
- 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，否则快速滑屏的话会不够流畅。

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。

8.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。

9.Java中引用类型都有哪些

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：

- 首先将softRef的referent（abc）设置为null，不再引用heap中的new String("abc")对象。
- 将heap中的newString("abc")对象设置为可结束的(finalizable)。
- 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 SoftReference 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）。
file:///C:/Users/ADMINI~1/AppData/Local/Temp/msohtmlclip1/01/clip_image002.jpg

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newString("abc")对象设置为可结束的(finalizable)。

2.与软引用和弱引用不同, 先把PhantomRefrence对象添加到它的ReferenceQueue中.然后在释放虚可及的对象。

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。

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

```
InputStreamin = conn.getInputStream();
StringjsonStr = 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

- 通过构造函数将json字符串转换成json对象

```
JSONObject  jsonObject = new JSONObject(jsonStr);
```

- 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：

- 从json对象中获取你所需要的键所对应的值

```
JSONObject json=jsonObject.getJSONObject("weatherinfo");
Stringcity = json.getString("city");
Stringtemp = json.getString("temp")
```

- 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);
  Stringtitle=obj.getString("title");
  Stringdescription=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);
JSONObjectjson = new JSONObject(map);
```

方法2、创建一个json对象，通过put方法添加数据

```
JSONObjectjson=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);
JSONArrayarray=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){
        XmlPullParserparser=Xml.newPullParser();//获取解析器
        parser.setInput(in,"utf-8");
        for(inttype=){   //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.setOuput(put,"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。

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

- 在使用内存的时候，Parcelable 类比Serializable性能高，所以推荐使用Parcelable类。
- 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。
- 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"/>
    <actionandroid: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

```
Intentintent =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

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。

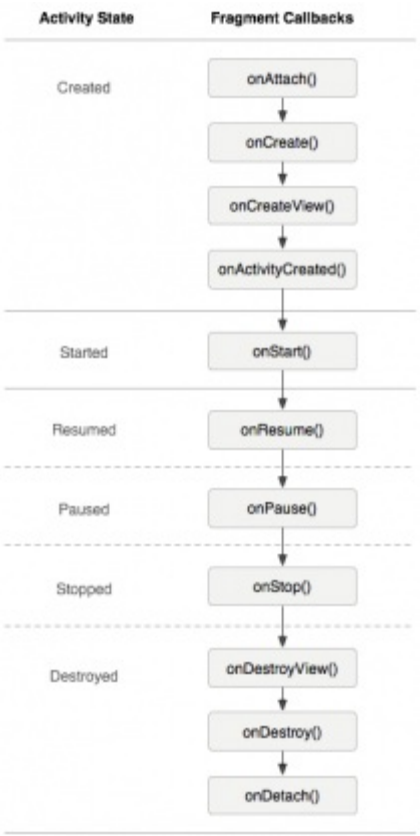
Fragment1、Fragment跟Activity之间是如何传值的

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、描述一下Fragment的生命周期



能否在子线程中更新UI

可以的，在onCreate()的setContentView()后new一个Thread去更新UI是不会报错的，但是延迟1s后再更新UI就会报错，这是因为在onCreate()的时候ViewRoot的requestLayout()方法没有执行，layout布局文件还没有创建完成。而ViewRoot的requestLayout()方法中才会调用checkThread()方法检查当前是否是主线程，不是的话就抛CalledFromWrongThreadException。Android中的定义是：不建议在子线程中更新UI，否则会产生不可预知的错误

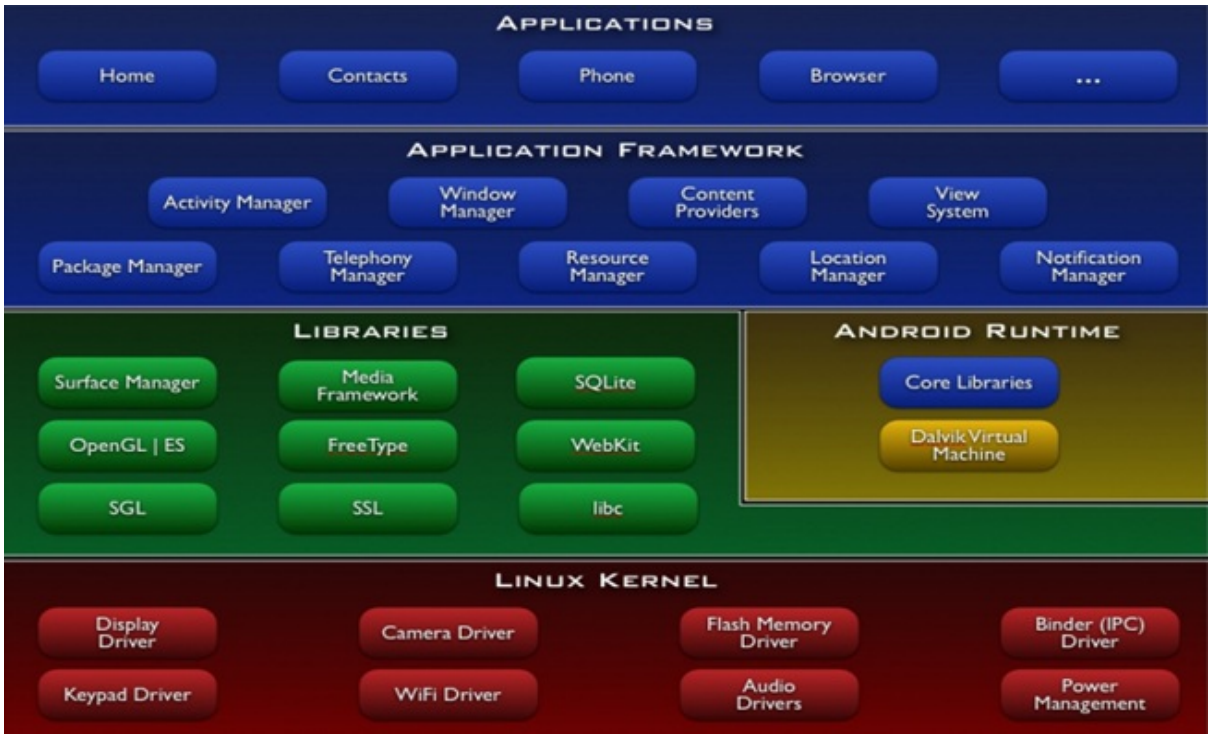
> 原文链接：http://blog.csdn.net/lmj623565791/article/details/24015867/

下面的题目都是楼主在Android交流群大家面试时遇到的，如果大家有好的题目或者好的见解欢迎分享，楼主将长期维护此帖。

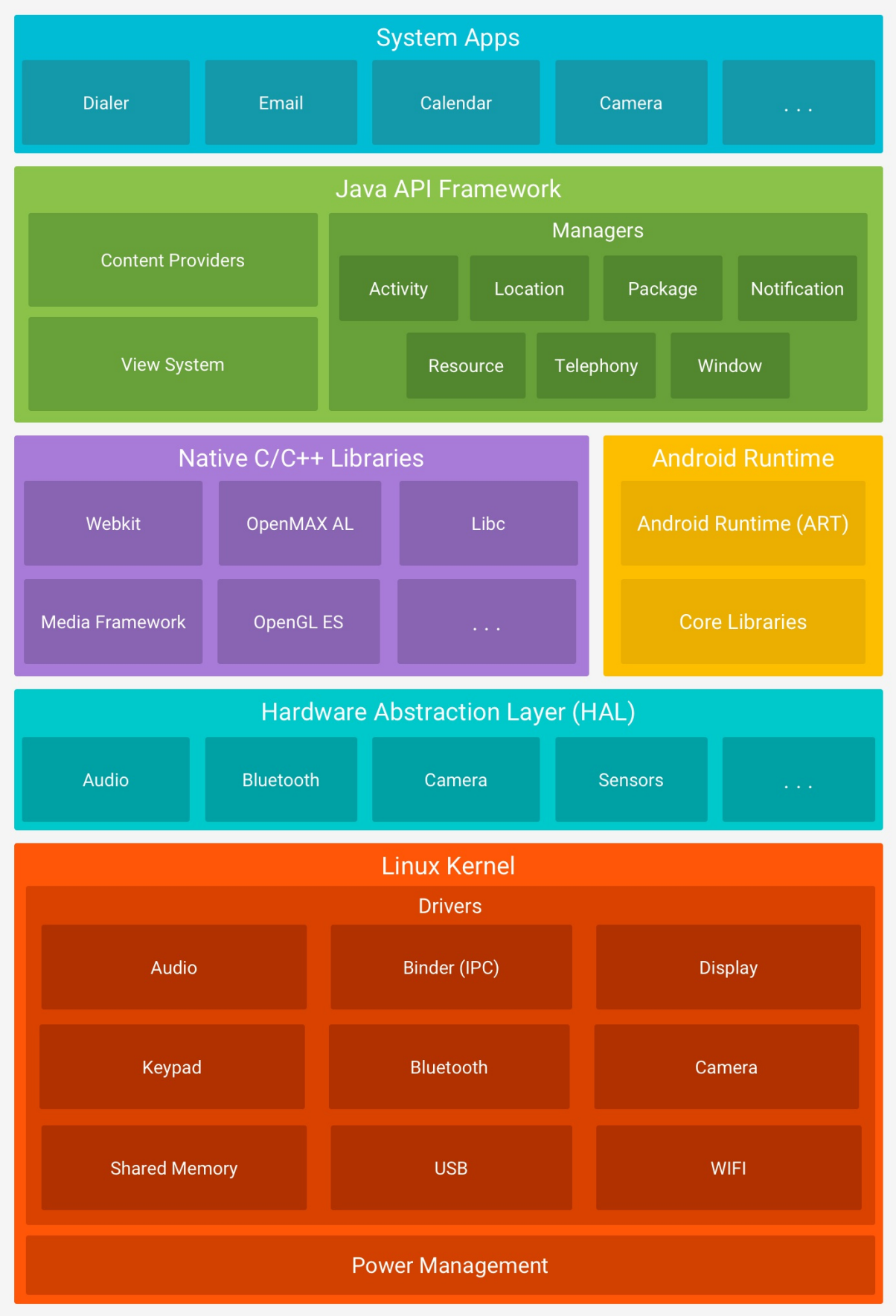
某公司高级面试题

1、详述Android系统架构，包括层与层之间调用、binder、jni、底层文件读写方法

android源码之前分为四层，如下图：



新的android源码分为五层（平台架构），增加了HAL层，如下图：



- Linux Kernel

Linux内核，主要是一些硬件驱动和Binder驱动（IPC进程间通信）

- 硬件抽象层

Android系统的硬件抽象层（Hardware Abstract Layer，HAL）运行用户空间中，它向下屏蔽硬件驱动模块的实现细节，向上提供硬件访问服务。通过硬件抽象层，Android系统分两层来支持硬件设备，其中一层实现在用户空间中，另一层实现在内核空间中。传统的Linux系统把对硬件的支持完全实现在内核空间中，即把对硬件的支持完全实现在硬件驱动模块中。

Android系统里封装内核驱动程序的接口层。对上层提供接口，屏蔽底层驱动实现细节.

本来Linux内核可以负责驱动接口定义和驱动实现，但是受限于GNU License（开源感染性），如果厂商选择驱动接口和实现都在内核空间完成，就必须开放自己的驱动源代码。这是不符合厂商利益的（驱动包含核心硬件参数，与其他厂家竞争的法宝）。所以Google将Linux内核中跟底层硬件操作相关的逻辑封装成HAL层接口，厂商基于接口去实现，不直接在内核空间实现驱动。因为Android系统遵循Apache License，不强制开源。

- Libraries and Android Runtime

原生C/C++库，系统运行的核心库，Android运行时环境，包括核心库和dvm

- Application FrameWork 框架层

提供的是Java API，主要是各种Manager，如WindowManager，ActivityManager等。

硬件访问服务通过硬件抽象层模块来为应用程序提供硬件读写操作。由于硬件抽象层模块是使用C++语言开发的，而应用程序框架层中的硬件访问服务是使用Java语言开发的，因此，硬件访问服务必须通过Java本地接口（Java Native Interface，JNI）来调用硬件抽象层模块的接口。

在Android应用程序框架层开发硬件访问服务的目的是为了让上层的Android应用程序能够访问对应的硬件设备。

- Applications 应用层

我们使用的系统自带App或者自己安装的App都属于应用层

应用程序框架中的基于Java语言的Binder接口是通过JNI来调用基于C/C++语言的Binder运行库来为Java应用程序提供进程间通信服务的

2、描述自己的一个项目，要求画出结构图，UML图，详细描述项目种的技术点，技术难点以及解决方案

3、一道算法

4、谈谈自己项目管理的方法、对敏捷，即原型开发软件开发的理解

基础面试题

1. 请解释下在单线程模型中Message,Handler,MessageQueue,Looper之间的关系。

拿主线程来说，主线程启动时会调用Looper.prepare()方法，会初始化一个Looper，放入Threadlocal中，接着调用Looper.loop()不断遍历MessageQueue，Handler的创建依赖与当前线程中的Looper，如果当前线程没有Looper则必须调用Looper.prepare()。Handler , sendMessage到MessageQueue，Looper不断从MessageQueue中取出消息，回调handleMessage方法。

2. 如果有个100M大的文件，需要上传至服务器中，而服务器form表单最大只能上传2M，可以用什么方法。

这个问题不是很明确我觉得，首先来说使用http协议上传数据，特别在android下，跟form没什么关系。传统的在web中，在form中写文件上传，其实浏览器所做的就是将我们的数据进行解析组装拼成字符串，以流的方式发送到服务器，且上传文件用的都是POST方式，POST方式对大小没什么限制。

回到题目，可以说假设每次真的只能上传2M，那么可能我们只能把文件截断，然后分别上传了。

3. 内存溢出和内存泄漏有什么区别？何时会产生内存泄漏？内存优化有哪些方法？

内存溢出通俗理解就是软件（应用）运行需要的内存，超出了它可用的最大内存。

内存泄漏就是我们对某一内存空间的使用，使用完成后没有释放。

内存优化：Android中容易内存溢出的部分，就是图片的加载，我们可以使用图片的压缩加上使用LruCache缓存的目的来控制图片所能够使用的内存。

还有对于比较耗资源的对象及时的关闭，例如Database Conn， 各种传感器， Service 等等。

4. AsyncTask使用在哪些场景？它的缺陷是什么？如何解决？

AsyncTask 运用的场景就是我们需要进行一些耗时的操作，耗时操作完成后更新主线程，或者在操作过程中对主线程的UI进行更新。

缺陷：AsyncTask中维护着一个长度为128的线程池，同时可以执行5个工作线程，还有一个缓冲队列，当线程池中已有128个线程，缓冲队列已满时，如果此时向线程提交任务，将会抛出RejectedExecutionException。

解决：由一个控制线程来处理AsyncTask的调用判断线程池是否满了，如果满了则线程睡眠否则请求AsyncTask继续处理。

5. Activity用SharedPreferences保存数据，大小有木有限制？

这个真心查不到

Sp的底层是由xml实现的，操作Sp的过程就是xml的序列化和解析的过程.Xml是储存在磁盘上的，因此考虑到IO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明确的要求的，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，其实SP设置的目的就是为了保存用户的偏好和配置信息的，因此建议不要保存太多的数据

6. Activity间通过Intent传递数据大小有没有限制？

貌似是40K。Intent貌似应该是1m，Intent携带数据太大是会报错，比如携带一张大图片是就会发生异常

7. assest文件夹里放文件，对于文件的大小有没有限制？

assets目录更像一个附录类型的目录，Android不会为这个目录中的文件生成ID并保存在R类当中，因此它与Android中的一些类和方法兼容度更低。

同时，由于你需要一个字符串路径来获取这个目录下的文件描述符，访问的速度会更慢。但是把一些文件放在这个目录下会使一些操作更加方便，比方说拷贝一个数据库文件到系统内存中。要注意的是，你无法在Android XML文件中引用到assets目录下的文件，只能通过AssetManager来访问

这些文件。数据库文件和游戏数据等放在这个目录下是比较合适的。另外，网上关于assets和raw的资料都千篇一律了，因此关于这两者中单个文件

大小不能超过1M的**错误**描述也在传播，即如果读取超过1M的文件会报"Data exceeds UNCOMPRESS_DATA_MAX (1314625 vs 1048576)"的IOException，还引申出种种解决方案。个人认为不应该有这样的限制，为了验证这个说法写了个Demo，发现将近5M的压缩包在assets和raw中都能正常访问，因此在这里纠正一下，理论上只要打包不超过Android APK 50M大小的限制都是没有问题的。当然了，不排除是Android很早期的时候因为设备硬件原因aapt在编译的时候对这两个文件夹大小做出了限制，如果是这样，较新版的ADT应该不会出现这种情况。

来自：http://my.eoe.cn/futurexiong/archive/5350.html

8. 启动一个程序，可以主界面点击图标进入，也可以从一个程序中跳转过去，二者有什么区别？

是因为启动程序（主界面也是一个app），发现了在这个程序中存在一个设置为`<category android:name="android.intent.category.LAUNCHER" />`的activity,所以这个launcher会把icon提出来，放在主界面上。当用户点击icon的时候，发出一个Intent:

```
Intent intent = mActivity.getPackageManager().getLaunchIntentForPackage(packageName);
mActivity.startActivity(intent);
```

跳过去可以跳到任意允许的页面，如一个程序可以下载，那么真正下载的页面可能不是首页（也有可能是首页），这时还是构造一个Intent，startActivity.这个intent中的action可能有多种view,download都有可能。系统会根据第三方程序向系统注册的功能，为你的Intent选择可以打开的程序或者页面。所以唯一的一点不同的是从icon的点击启动的intent的action是相对单一的，从程序中跳转或者启动可能样式更多一些。本质是相同的。

9. 程序之间的亲和性的理解

- 默认情况下一个应用的所有Activity都是具有相同的affinity，都是从application中继承，application的affinity默认就是manifest的包名。
- affinity对Activity来说，就像是身份证一样，可以告诉所在的Task，自己属于其中的一员。
- 应用场合：
  - 根据affinity重新为Activity选择合适的宿主Task
  - 与allowTaskReparenting属性配合
  - 启动Activity使用Intent设置了FLAG_ACTIVITY_NEW_TASK标记

10. 同一个程序，但不同的Activity是否可以放在不同的Task任务栈中？

可以放在不同的Task中。需要为不同的activity设置不同的affinity属性，启动activity的Intent需要包含FLAG_ACTIVITY_NEW_TASK标记。

11. 横竖屏切换时候Activity的生命周期。

- 不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次
- 设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次
- 设置Activity的android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

12. AIDL的全称是什么？如何工作？

全称是：Android Interface Define Language

在Android中, 每个应用程序都可以有自己的进程. 在写UI应用的时候, 经常要用到Service. 在不同的进程中, 怎样传递对象呢? 显然, Java中不允许跨进程内存共享.

因此传递对象, 只能把对象拆分成操作系统能理解的简单形式, 以达到跨界对象访问的目的. 在J2EE中,采用RMI的方式, 可以通过序列化传递对象. 在Android中, 则采用AIDL的方式. 理论上AIDL可以传递Bundle,实际上做起来却比较麻烦。

AIDL(Android接口描述语言)是一种接口描述语言; 编译器可以通过aidl文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程的目的. 如果需要在一个Activity中, 访问另一个Service中的某个对象, 需要先将对象转化成AIDL可识别的参数(可能是多个参数), 然后使用AIDL来传递这些参数, 在消息的接收端, 使用这些参数组装成自己需要的对象.AIDL的IPC的机制和COM或CORBA类似, 是基于接口的，但它是轻量级的。它使用代理类在客户端和实现层间传递值. 如果要使用AIDL, 需要完成2件事情: 1. 引入AIDL的相关类; 2. 调用aidl产生的class.

AIDL的创建方法:

AIDL语法很简单,可以用来声明一个带一个或多个方法的接口， 也可以传递参数和返回值。 由于远程调用的需要, 这些参数和返回值并不是任何类型.

下面是些AIDL支持的数据类型:

1. 不需要import声明的简单Java编程语言类型(int,boolean等)

1. String, CharSequence不需要特殊声明

1. List, Map和Parcelables类型, 这些类型内所包含的数据成员也只能是简单数据类型, String等其他比支持的类型.

(另外: 我没尝试Parcelables, 在Eclipse+ADT下编译不过, 或许以后会有所支持

13. 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的进程是dalivk虚拟机进程,每个android程序都运行在自己的进程里面,每个android程序系统都会给他分配一个单独的liunx uid(user id),每个dvm都是linux里面的一个进程.所以说这两个进程是一个进程.


与IPC机制相关的试题
1- Davik进程、linux进程、线程之间的区别？
2-Android 中进程与进程之间如何通信？
与及时通讯相关试题
3-简单阐述一下对XMPP协议理解以及优缺点？
4-简单阐述一下及时推送原理？
5-简要说明一下openfire、smack、spark
6-列出几种常见的解决消息即时获取方案
7-Timer比AlarmManager实现心跳时更耗电原因
与性能优化相关试题
8-对于Android 的安全问题，你知道多少
9-内存泄漏和内存溢出分别是什么？它们有什么...
10-什么情况下会导致内存泄漏
11-说一下如何对Android内存优化？
12-如何对android应用进行内存性能分析
与开源框架相关的试题
13-开发中常用的一些开源框架和平台有哪些？
14-说一下Picasso ImageLoader Fresco Gli...
15-volley、okHttp之间的区别和优缺点
16-说一下Volley框架的实现原理以及优缺点？
与应用发布相关的试题
与工作中开发有关的试题
Android干货程序员


Android中数据存储方式和缓存
1- 请介绍下Android中的数据存储方式（8分）
2- 在项目中，你是如何缓存数据的？（10分）
3- 说说LruCache底层原理（10分）
4- 加载图片的三级缓存原理（10分）
动画相关面试题
5- android中的动画有哪些,它们的特点和区别是...
6- 动画插值器是什么（20分）
7- 请说一下对属性动画的理解（20分）
8- 如何让自定义view中绘制的内容执行动画（20...
9- 介绍一下Matrix的作用（3分）
自定义控件相关面试题
与WebView相关的面试题
25- webview和js的交互（5分）
26- WebView加载数据性能优化？webview加载...
27- 如何使WebView自适应屏幕大小？
Android干货程序员


View的实现和优化相关面试题
1-ListView优化？ListView如何提高其效率？
2-如何实现让ListView的条目显示不一样的布局?
3-ScrollView中嵌入ListView常见问题和解决方...
跨平台开发相关面试题
4-简单描述一下Html5的特点？
屏幕适配相关的面试题
5-屏幕适配的方式有哪些？
6-请问平时开发过程中，你是如何做到多分辨率...
与设计模式相关的面试题
7-简单说说MVC,MVP原理以及它们之间的区别...
8-你一般在开发项目中都使用什么设计模式？如...
9-手写一个单例模式，能不能保证使用时对象的...
与网络相关的面试题
与线程间通信机制相关面试题
与延迟和异步任务相关试题
Android干货程序员


与Activity相关的面试题
1-两个Activity之间跳转时必然会执行的是哪几个...
2-如何退出Activity？如何安全退出已调用多个Ac...
3-设备横竖屏切换的时候，接下来会发生什么（6...
4-后台的activity被系统自动回收的话，怎么在回...
5-activity的四种启动模式，分别代表什么含义，...
6-切换主题功能（8分）
7-简述Activity的管理机制（10分）
与Service相关的面试题
8-说一下IntentService和Service的区别？（8分）
9-请描述一下Service的生命周期（5分）
10-Activity怎么和service绑定，怎么在activity中...
11-Service是否在main thread中执行, service...
12-Service中的onStartCommond四种返回值策...
13-service被kill之后怎么让它重启（8分）
14-Activity如何给Service发送Message？
Android系统相关面试题
与Fragment相关的面试题
Android干货程序员

整个面试题视频如下(持续更新中)：

与IPC机制相关面试题

- 1- Davik进程linux进程线程之间的区别
- 2- aidl实现进程间通信
- 3- messenger实现进程间通信
- 4- ContentProvider实现进程间通信

与性能优化相关试题

- 5- 什么是内存泄漏
- 6- 什么是内存溢出
- 7- 什么情况会导致内存泄漏
- 8- 避免程序的OOM异常
- 9- 线程池原理
- 10- UI性能优化
- 11- 内存优化之字符串优化
- 12- 常见内存优化方式
- 13- 性能分析之hierarchyviewer使用
- 14- 性能分析之Lint规范代码
- 15- 性能分析之规避内存抖动
- 16- 性能分析之内存检测工具介绍

与XMPP相关试题

- 17- 什么是XMPP和XMPP的数据格式
- 18- 及时聊天的展示形式
- 19- TCP和UDP协议
- 20- 极光推送原理
- 21- XMPP的基本概念
- 22- 常见消息推送的解决方案

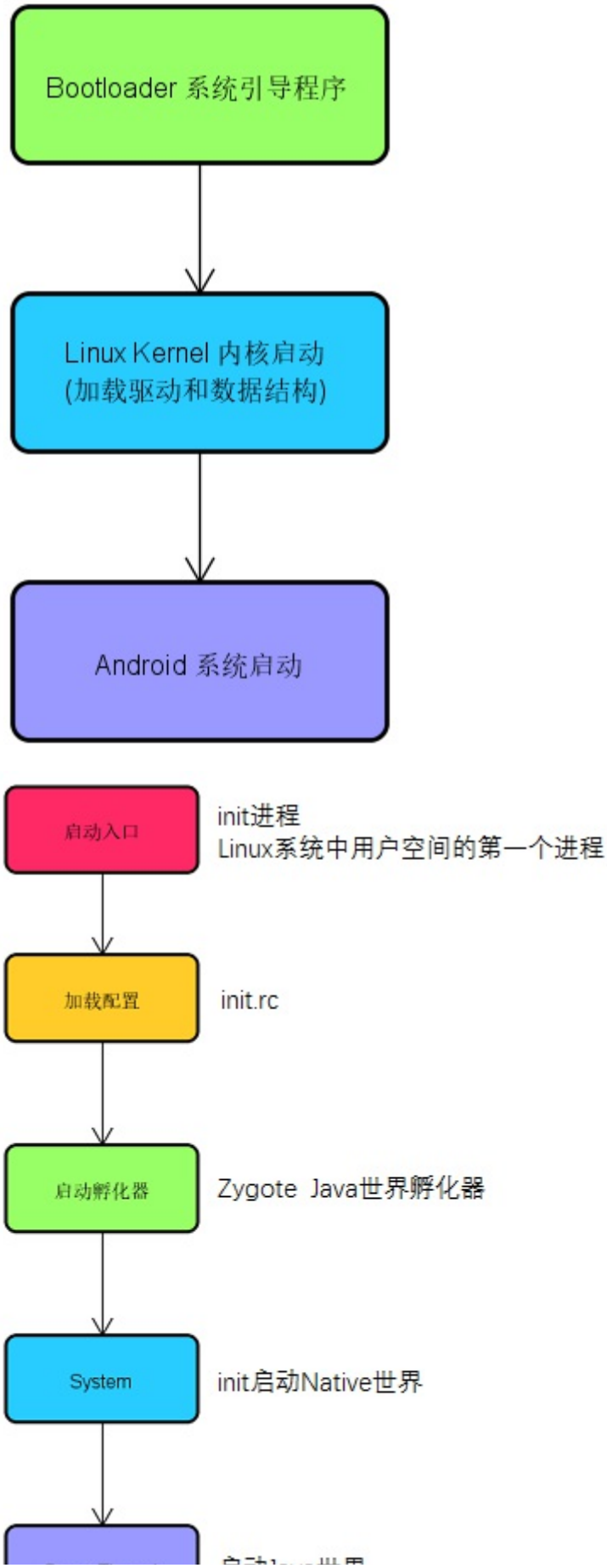
与登录相关试题

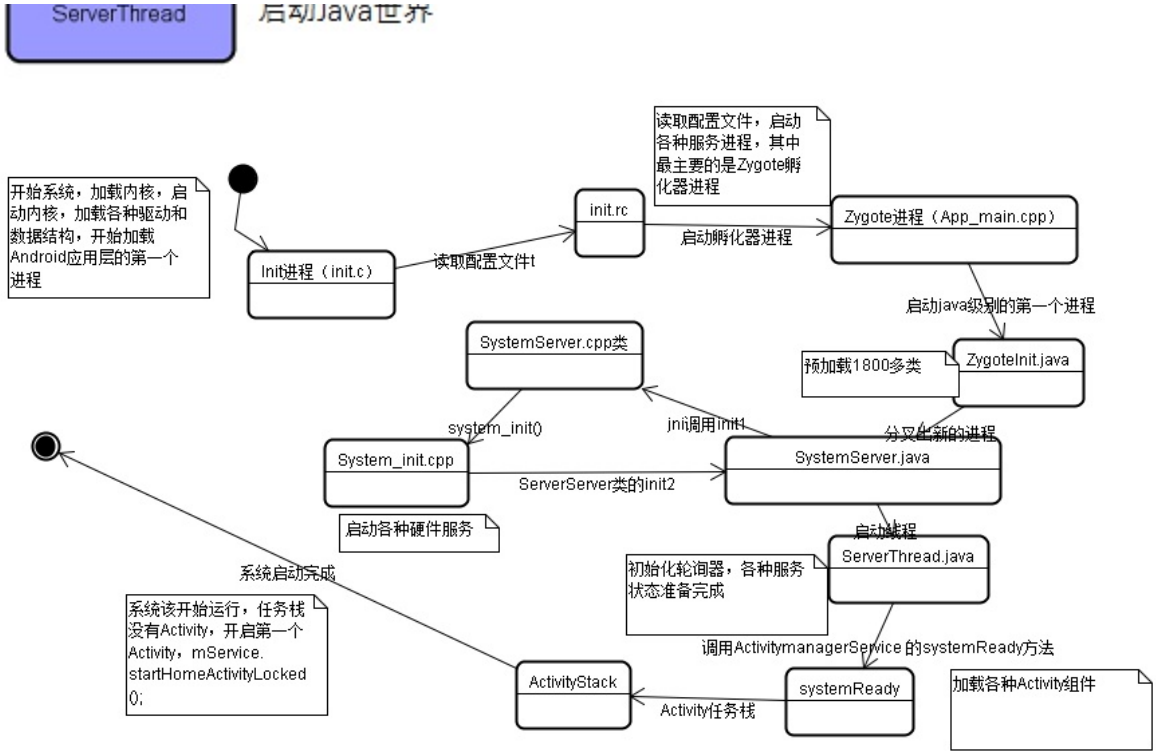
- 23- 微信扫一扫登录内部实现原理
- 24- 腾讯QQ三方登录实现原理
- 25- 登录为什么要使用Token

与开发相关试题

- 26- 迭代开发的时候如何向前兼容新旧接口
- 27- 应用程序的开发流程

Android启动流程


Bootloader 系统引导程序
Linux Kernel 内核启动
(加载驱动和数据结构)
Android 系统启动
启动入口
init进程
Linux系统中用户空间的第一个进程
加载配置
init.rc
启动孵化器
Zygote Java世界孵化器
System
init启动Native世界



在Android系统启动时，第一个启动起来的进程就是Zygote（进程孵化器）进程，然后由Zygote启动SystemServer，再后就是启动ActivityManagerService，WindowManagerService等系统核心服务，这些服务承载着整个Android系统与客户端程序交互的重担。Zygote除了启动系统服务和进程之外，普通的用户进程也由Zygote进程fork而来，当一个应用进程启动起来后，就会加载用户在清单文件（AndroidManifest.xml）中配置的默认加载的Activity，此时加载的入口是 `ActivityThread.main(String[] args)` ，这个方法就类似与C语言中的main()方法，是整个应用程序的入口。

由操作系统的引导文件去运行linux的内核程序，内核程序开始启动的时候会加载各种驱动和数据结构，开始加载android应用层的第一个进程（init进程c代码（system\core\init目录） Init.c）

Zygote进程 --> SystemServer --> 系统服务AMS，WMS，PMS...

1、当引导程序启动Linux内核后，会加载各种驱动和数据结构，当有了驱动以后，开始启动Android系统同时会加载用户级别的第一个进程init（system\core\init.c）代码如下：

```
int main(int argc, char **argv)
{
    // 创建文件夹 挂载
    mount("tmpfs", "/dev", "tmpfs", 0, "mode=0755");
    mkdir("/dev/pts", 0755);

    // 打卡日志
    log_init();

    INFO("reading config file\n");
    // 加载init.rc配置文件
    init_parse_config_file("/init.rc");
}
```

2、加载init.rc文件，会启动一个Zygote进程，此进程是Android系统的一个母进程，用来启动Android的其他服务进程，代码：

```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
```

```
socket zygote stream 666
onrestart write /sys/android_power/request_state wake
onrestart write /sys/power/state on
onrestart restart media
onrestart restart netd
```

3、从c++代码调到java代码:

```
int main(int argc, const char* const argv[])
{
    ...
    // Android运行时环境
    AppRuntime runtime;
    ...
    // Next arg is startup classname or "--zygote"
    if (i < argc) {
        arg = argv[i++];
        if (0 == strcmp("--zygote", arg)) {
            bool startSystemServer = (i < argc) ?
                    strcmp(argv[i], "--start-system-server") == 0 : false;
            setArgv0(argv0, "zygote");
            set_process_name("zygote");
            // 启动java代码
            runtime.start("com.android.internal.os.ZygoteInit",
         ...
}
```

4、ZygoteInit.java 代码:

```
 public static void main(String argv[]) {
        try {
            VMRuntime.getRuntime().setMinimumHeapSize(5 * 1024 * 1024);

            // 加载Android依赖的类
            preloadClasses();
            //cacheRegisterMaps();
            preloadResources();
            ...

            if (argv[1].equals("true")) {
                // 启动系统服务
                startSystemServer();
            } else if (!argv[1].equals("false")) {
           ...
    }

    private static boolean startSystemServer()
         ...
            args = new String[] {
                "--setuid=1000",
                "--setgid=1000",
                "--setgroups=1001,1002,1003,1004,1005,1006,1007,1008,1009,1010,1018,3001,3002,3003,3006",
                "--capabilities=130104352,130104352",
                "--rlimit=8,",
                "--runtime-init",
                "--nice-name=system_server",
                "com.android.server.SystemServer",
          ...

            /* Request to fork the system server process */
            // 母进程开始分叉服务 启动SystemServer
            pid = Zygote.forkSystemServer(
                    parsedArgs.uid, parsedArgs.gid,
```

```
                    parsedArgs.gids, debugFlags, rlimits,
                    parsedArgs.permittedCapabilities,
                    parsedArgs.effectiveCapabilities);
        ..
    }
```

5、SystemServer.java 代码

```
 public static void main(String[] args) {
    ...
    // 加载jni库
    System.loadLibrary("android_servers");
    // 调用native方法
    init1(args);
}
native public static void init1(String[] args);
```

6、SystemServer 对应的c++代码 com_android_server_SystemServer.cpp 代码如下：

```
    // 类似java的抽象方法
    extern "C" int system_init();

    static void android_server_SystemServer_init1(JNIEnv* env, jobject clazz)
    {
        // 转调
        system_init();
    }

    /*
     * JNI registration.
     */
    static JNINativeMethod gMethods[] = {
        /* name, signature, funcPtr */
        // 函数指针 把init1方法映射到android_server_SystemServer_init1
        { "init1", "([Ljava/lang/String;)V", (void*) android_server_SystemServer_init1 },
    };
```

7、system_init 的实现方法在System_init.cpp 代码如下：

```
    extern "C" status_t system_init()
    {
        ...
        // 启动硬件的服务
        if (strcmp(propBuf, "1") == 0) {
            // Start the SurfaceFlinger
            SurfaceFlinger::instantiate();
        }

        AndroidRuntime* runtime = AndroidRuntime::getRuntime();

        LOGI("System server: starting Android services.\n");
        // 启动完硬件服务后，又回到Systemserver的init2方法
        runtime->callStatic("com/android/server/SystemServer", "init2");
        ...
    }
```

8、SystemServer 的init2方法代码：

```
public static final void init2() {
        Slog.i(TAG, "Entered the Android system server!");
```

```
        Thread thr = new ServerThread();
        thr.setName("android.server.ServerThread");
        thr.start();
}
```

9、ServerThread的run方法：

```
public void run() {
        ...
        // 开启Android各种服务并且添加到ServiceManager去管理
        Slog.i(TAG, "Device Policy");
        devicePolicy = new DevicePolicyManagerService(context);
        ServiceManager.addService(Context.DEVICE_POLICY_SERVICE, ottle =

        ...
        // We now tell the activity manager it is okay to run third party
        // code.  It will call back into us once it has gotten to the state
        // where third party code can really run (but before it has actually
        // started launching the initial applications), for us to complete our
        // initialization.
        // 各种服务开启后调用ActivityManagerService.systemReady
        ((ActivityManagerService)ActivityManagerNative.getDefault())
                .systemReady(new Runnable() {
            public void run() {
                Slog.i(TAG, "Making services ready");
```

10、ActivityMangerService的systemReady的方法：

```
    public void systemReady(final Runnable goingCallback) {
            ...
            // 打开第一个Activity
                mMainStack.resumeTopActivityLocked(null);
            }
        }
```

11、ActivityStack的resumeTopActivityLocked方法

```
    final boolean resumeTopActivityLocked(ActivityRecord prev) {
            // Find the first activity that is not finishing.
            // 没有已经打开的Activity next为 null
            ActivityRecord next = topRunningActivityLocked(null);

            // Remember how we'll process this pause/resume situation, and ensure
            // that the state is reset however we wind up proceeding.
            final boolean userLeaving = mUserLeaving;
            mUserLeaving = false;

            if (next == null) {
                // There are no more activities!  Let's just start up the
                // Launcher...

                if (mMainStack) {
                    // 启动lucher应用的锁屏界面
                    return mService.startHomeActivityLocked();
                }
            }
```

12、Android系统启动完成，打开了Luncher应用的Home界面。

面试记录

1.对比ListView跟RecyclerView，RecyclerView有什么优势？

2.了解过哪些网络框架？OkHttp比HttpURLConntaion好在哪里？

3.Retrofit怎么进行二次封装？有哪些好处？

4.自定义View的了解？

5.短信验证码怎么保证登录的安全？

- 密码加密

  md5，非对称加密

- 动态密码
- 验证码
- 有效时间
- TrustZone

https://www.zhihu.com/question/24173904

身份认证有三个方式：你知道的，你持有的，以及你固有的。一般的口令密码之类算第一类（你知道的），持有令牌通行证之类算第二类（你持有的），指纹虹膜等生物特征算第三类（你固有的）。由于获取 / 伪造的难度不同，一般认为第一类的安全性比第二类差，第二类又比第三类差；但需要明确的是，如果只有其中一种都算是弱认证，必须独立使用两种甚至三种才算是强认证。

普通应用比如邮箱的认证方式都是口令或密码这第一类认证，使用短信验证码则是为了提供第二类认证。在安全设定中，做得好的系统会要求关键修改要同时使用两种认证方式，即：使用密码登录，然后修改关键信息比如已经注册过的手机号，还需要先用之前的手机接收验证码；单独获得手机后，是不应该能够登入账户并修改所有信息的，否则就破坏了多种认证方式直接的独立性，进而破坏了系统的安全性。

在智能手机的年代，由于OS开放了短信操作和拦截的接口（Android直接提供，iOS需要越狱），对于一个安装了支付类App的智能手机且绑定账户的SIM卡也安装在同一个手机的情况（绝大部分情况下是这样），短信验证事实上已经退化成了单因子验证，只要智能手机被安装了木马那么这些验证体系就会全线崩溃，攻击者甚至可以只通过钓鱼wifi全部搞定登陆密码、支付密码和短信验证

短信验证码算是短信细分行业里的小众行业，因为价格便宜却能针对性的起到大力宣传作用，自然短信这个行业的竞争也愈演愈烈，各个短信运营商都绞尽脑汁在确保网站短信验证码接口相关措施的严密。

阅信短信验证码平台小编下面从五个方面入手说说阅信短信平台是怎么做的：

- 绑定服务器IP。供应商的短信验证码接口只能识别客户的绑定服务器IP，否则将会报错；
- 开启反投诉策略。短信验证码接口容易受到轰炸，供应商应将同个手机号码在24小时内接收同一根通道短信条数最大值设置为10；

阅信网站短信验证码接口

- 防盗用策略。为了防止客户账号被盗，供应商可以限制短信验证码接口任何时间段的任意条数上限。类似于刚刚上线的App客户，短信日发送上限可以设置成10000条；
- 关闭网页发送短信权限。验证码基本上经由短信验证码接口提交，所以web端的网页提交功能应该关闭；
- 绑定后台登录手机号码。短信接口管理后台可以查看到每一条短信的发送详情，为了保证数据安全，必须要指定专人管理，登录后台要收到手机短信验证码才能登录成功。

当然，验证码安全并不完全由通道供应商单方面控制，从平台开发者本身出发，也需要做好防范机制，例如完善网站入口处的二次验证。

随着App项目的推广，网站或手机短信验证码接口广泛应用在用户验证各方面，大大降低了非法注册，烂注册的数据的出现，所以其安全防范机制尤为重要。

6.线程间通信有哪些方式?

gravity跟layout_gravity区别

静态成员变量运行题

常见五种布局

自动更新原理

Activity生命周期

手写数据表查询年龄最大的十个人

数据存储的几种方式

通过网络请求加载图片并缓存到本地

第二次加载图片时从本地加载

用实际例子问我权重，sp，数据库，然后看了下app，问了哪些新控件使用比较熟，然后问RecyclerView怎么添加分隔线，我说用网上找的工具类。数据库很重视，但是问的几个问题我都只能说出思路。

Android已离职，面试我的估计是java，上来先问了个反射，然后对着简历挑着跟java沾边的东西问，比如线程间通信，MVP跟MVC模式，Cookie，然后问了些安卓的自定义View的绘制流程、事件分发机制、第三方SDK。一开始还问了我做没做过百度地图，我说做过但是没有扩展很多功能，只做了基本的功能实现，然后他问导航做过没，我说没有。实际上他们公司下个项目就要用到百度地图。

公司环境一般，一个大厅用玻璃隔出了一个会议室跟老板办公室，剩下的就是两排桌子面对面坐着8~10个人，员工都比较年轻，很多刚毕业的。面试我的人感觉经验也不丰富，技术水平应该比较一般

1.AIDL简历写了的，抓紧了解清楚怎么使用！ 2.对称加密与非对称加密的区别，进程间通信的所有方式，数据库优化与框架（存入一万条数据怎么优化？），事件分发细节不够清楚(返回true/false/super相应有什么结果？)，内容提供者不熟悉，网络框架源码必须得看一个!

上来狂怼Java基础和Java相关的知识点，问了怎么创建线程？暂停线程的两种方法及区别？AIDL服务怎么暴露方法？工厂模式？观察者模式？http长连接短连接和组成？使用什么请求？数据库？线程池的使用场景？

面试上来先自我介绍，完了就开始聊项目，感觉整个过程都挺随意的，问了挺多问题，但是基本都没有深究，基本没问到太细节的实现。JNI跟混合开发都有被问到，看来还是得了解一点其他知识点给自己涨涨身价

面试官起码三十多岁，问的网络的问题听着都很懵逼，还问到app测试版跟正式版有什么不同？三个gradle有什么区别？对gradle有什么理解？有没有发生过闪退？怎么定位问题？

问了些实际开发的问题，比如pad上图标如何适配，UI给比例图时我们怎么确定文字的大小，集成第三方SDK需要多少时间。

问的比较简单，就自定义View跟事件分发、Activity、服务、网络、进程通信，还简单讲了一下Binder的底层原理。顺利通过技术面试

跟老板谈了下，被问到认为自己技术还有哪些方面需要提升，顺势说目前已经了解过Binder底层，还需要了解更多Android底层的东西和框架的实现原理，以便更好运用和修改。问到职业规划表示自己不光对技术感兴趣，对产品也很感兴趣。然后讲了下对上个公司产品的一些看法，对他们公司产品的看法，提了两个优化的点。基本就顺利过

关了

Imageloader

常用的图片处理框架

Fresco 正在实现了三级缓存，2级内存缓存，一级本地缓存、Glide、Picasso、UIL-ImageLoader

图片占用内存

大小 = 每个像素占用的字节数 * 总像素

Bitmap.Config

- RGB_565
- ARGB_8888

内存溢出

图片乱序错位问题

imageview.setTag(url)

弱引用双向关联

使用volley的NetImageView

四种引用类型

- 强引用
- 弱引用
- 软引用
- 虚引用

三级缓存

1、DiskLrucache 本地/磁盘缓存

2、Lrucache 内存缓存

原理

LinkedHashMap 双向循环链表

- 参数1：初始大小
- 参数2：装载因子
- 参数3：true 按访问顺序排序，false 按插入顺序排序

内存大小

```
int memory = (int) (Runtime.getRuntime().maxMemory()/8);
LruCache<String,Bitmap> cache = new LruCache<String,Bitmap>(memory){
    @Override
    protected int sizeOf(String key, Bitmap value) {
        return value.getByteCount();
    }
```

```
};
```

Lrucache .get(key) --> LinkedHashMap.get(key) : 命中，移动到双向循环链表的的尾部

Lrucache .put(key) --> LinkedHashMap.put(key) : 如果size > maxSize，删除链表header节点直到size < maxSize

3、 网络缓存

4、缓存路径

- getExternalCacheDir()
- getCacheDir()

图片压缩

BitmapFactory.Options

- inJustDecodeBounds
- inSampleSize
- outWidth
- outHeight
- inBitmap

```
/**
  * Decode and sample down a bitmap from resources to the requested width and height.
  *
  * @param res The resources object containing the image data
  * @param resId The resource id of the image data
  * @param reqWidth The requested width of the resulting bitmap
  * @param reqHeight The requested height of the resulting bitmap
  * @param cache The ImageCache used to find candidate bitmaps for use with inBitmap
  * @return A bitmap sampled down from the original with the same aspect ratio and dimensions
  *         that are equal to or greater than the requested width and height
  */
 public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
         int reqWidth, int reqHeight, ImageCache cache) {

     // BEGIN_INCLUDE (read_bitmap_dimensions)
     // First decode with inJustDecodeBounds=true to check dimensions
     final BitmapFactory.Options options = new BitmapFactory.Options();
     options.inJustDecodeBounds = true;
     BitmapFactory.decodeResource(res, resId, options);

     // Calculate inSampleSize
     options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
     // END_INCLUDE (read_bitmap_dimensions)

     // If we're running on Honeycomb or newer, try to use inBitmap
     if (Utils.hasHoneycomb()) {
         addInBitmapOptions(options, cache);
     }

     // Decode bitmap with inSampleSize set
     options.inJustDecodeBounds = false;
     return BitmapFactory.decodeResource(res, resId, options);
 }

/**
  * Decode and sample down a bitmap from a file to the requested width and height.
  *
  * @param filename The full path of the file to decode
```

```
     * @param reqWidth The requested width of the resulting bitmap
     * @param reqHeight The requested height of the resulting bitmap
     * @param cache The ImageCache used to find candidate bitmaps for use with inBitmap
     * @return A bitmap sampled down from the original with the same aspect ratio and dimensions
     *         that are equal to or greater than the requested width and height
     */
    public static Bitmap decodeSampledBitmapFromFile(String filename,
            int reqWidth, int reqHeight, ImageCache cache) {

        // First decode with inJustDecodeBounds=true to check dimensions
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(filename, options);

        // Calculate inSampleSize
        options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

        // If we're running on Honeycomb or newer, try to use inBitmap
        if (Utils.hasHoneycomb()) {
            addInBitmapOptions(options, cache);
        }

        // Decode bitmap with inSampleSize set
        options.inJustDecodeBounds = false;
        return BitmapFactory.decodeFile(filename, options);
    }

    /**
     * Decode and sample down a bitmap from a file input stream to the requested width and height.
     *
     * @param fileDescriptor The file descriptor to read from
     * @param reqWidth The requested width of the resulting bitmap
     * @param reqHeight The requested height of the resulting bitmap
     * @param cache The ImageCache used to find candidate bitmaps for use with inBitmap
     * @return A bitmap sampled down from the original with the same aspect ratio and dimensions
     *         that are equal to or greater than the requested width and height
     */
    public static Bitmap decodeSampledBitmapFromDescriptor(
            FileDescriptor fileDescriptor, int reqWidth, int reqHeight, ImageCache cache) {

        // First decode with inJustDecodeBounds=true to check dimensions
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFileDescriptor(fileDescriptor, null, options);

        // Calculate inSampleSize
        options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

        // Decode bitmap with inSampleSize set
        options.inJustDecodeBounds = false;

        // If we're running on Honeycomb or newer, try to use inBitmap
        if (Utils.hasHoneycomb()) {
            addInBitmapOptions(options, cache);
        }

        return BitmapFactory.decodeFileDescriptor(fileDescriptor, null, options);
    }
    /**
     * Calculate an inSampleSize for use in a {@link android.graphics.BitmapFactory.Options} object when decoding
     * bitmaps using the decode* methods from {@link android.graphics.BitmapFactory}. This implementation calculates
     * the closest inSampleSize that is a power of 2 and will result in the final decoded bitmap
     * having a width and height equal to or larger than the requested width and height.
     *
     * @param options An options object with out* params already populated (run through a decode*
     *            method with inJustDecodeBounds==true
```

```
     * @param reqWidth The requested width of the resulting bitmap
     * @param reqHeight The requested height of the resulting bitmap
     * @return The value to be used for inSampleSize
     */
    public static int calculateInSampleSize(BitmapFactory.Options options,
                                            int reqWidth, int reqHeight) {
        // BEGIN_INCLUDE (calculate_sample_size)
        // Raw height and width of image
        final int height = options.outHeight;
        final int width = options.outWidth;
        int inSampleSize = 1;

        if (height > reqHeight || width > reqWidth) {

            final int halfHeight = height / 2;
            final int halfWidth = width / 2;

            // Calculate the largest inSampleSize value that is a power of 2 and keeps both
            // height and width larger than the requested height and width.
            while ((halfHeight / inSampleSize) > reqHeight
                    && (halfWidth / inSampleSize) > reqWidth) {
                inSampleSize *= 2;
            }

            // This offers some additional logic in case the image has a strange
            // aspect ratio. For example, a panorama may have a much larger
            // width than height. In these cases the total pixels might still
            // end up being too large to fit comfortably in memory, so we should
            // be more aggressive with sample down the image (=larger inSampleSize).

            long totalPixels = width * height / inSampleSize;

            // Anything more than 2x the requested pixels we'll sample down further
            final long totalReqPixelsCap = reqWidth * reqHeight * 2;

            while (totalPixels > totalReqPixelsCap) {
                inSampleSize *= 2;
                totalPixels /= 2;
            }
        }
        return inSampleSize;
        // END_INCLUDE (calculate_sample_size)
    }
```

EventBus

1. 事件总线

组件（activity，fragment，service）间的交互，通信，主线程子线程间的通信

观察者模式，发布订阅，注解，Map<订阅对象，订阅方法>

事件类型EventType，事件响应函数

2. 发布接收事件

- 发布事件post()

EventBus.getDefault().post(new User("mr.simple"), "my_tag");

反射调用，回调

- 注册事件register()

扫描订阅对象的所有方法，包括分类的方法，匹配方法，保存到Map集合中

- 接收事件，事件响应函数

@Subscriber(tag = "my_tag", mode=ThreadMode.POST)

3. EventBus

订阅队列

Map

判断是否是主线程：Looper.getMainLooper() == Looper.myLooper();

Handler mHandler = new Handler(Looper.getMainLooper());

ThreadLocal<PostingThreadState>

存储了▉个事件队列以及事件的状态

```
class PostingThread
{
List<Object> mEventQueue = new ArrayList<Object>();
boolean isMainThread;
boolean isPosting;
}
```

Map<Class, CopyOnWriteArrayList<SubscribeMethod>>

通过订阅对象的字节码拿到订阅对象匹配的方法以及方法的参数，其中threadmode是通过截取字符串获得，EventBus3.0是通过注解的方式拿到threadmode，方法的参数就是EventType

Android EventBus

EventType

方法的参数和tag组成EventType

依赖注入IOC

ImageLoader

面向接口编程，策略模式，BitmapRequest，链式调用 return this Builder模式，单例模式，生产者消费者模式

模板方法模式（算法骨架）

BitmapRequest

图片请求

ImageLoaderConfig

设置一些基本的东西，比如加载中的图片、加载失败的图片、缓存策略

RequestQueue

内部维持着一个PriorityBlockingQueue

```
BlockingQueue<BitmapRequest> mRequestQueue = new PriorityBlockingQueue<BitmapRequest>();
```

RequestDispatcher

HttpUtils

常用的网络请求框架

- Retrofit
- OkHttp
- NoHttp
- Volley
- HttpURLConnection（省电省流量）
- HttpClient（6.0后删除）
- AsyncHttpclient

请求头Map mHeaders，请求参数Map mBodyParams，URLEncoder.encode()

- NetworkExecutor 网络请求线程
- HttpStack Http执行器
- ResponseDelivery Response分发

Http协议

请求Request

- 请求首行
- 请求头
- 空行
- 请求体

响应Response

- 响应首行
- 响应头
- 空行
- 响应体

请求头

响应头

通用头

ContentType

- get 查
- post 改
- put 增 提交表单
- delete 删
- head 只返回首部
- trace 诊断，跟踪http请求是否被修改
- options 请求服务器告知其支持的各种功能

网络模型

TCP/IP参考模型

- 应用层 http，https，ftp
- 传输层 tcp udp
- 网络层 ip地址
- 物理层
- 数据链路层

三次、四次握手

三要素

ip地址，端口号，协议

文件上传

表单数据，Content-Type:multipart/form-data

ORM框架

注解（运行时注解或编译时注解）反射

注解标记表和表中的列，javabean上添加注解，指定一个类对应的表名，一个字段在表中对应的列名

在清单文件中配置数据库名，版本号，还可以配置model

model类可以从清单文件中获取，也可以通过扫描整个应用获取，扫描当前App的dex文件

创建表：通过注解标记表名和字段名，通过反射拿到model的信息来拼接sql语句

数据库操作类

- Selection
- Delete
- Update
- Insert

保证接口安全一般分为两种，一个是防数据篡改，一个是防数据泄漏，防篡改使用摘要验证，防泄漏使用加密

token

token=5位随机数+时间戳

aesEnctrypt（token）

（1）验证时间和服务器时间不能超过3分。

（2）同一个时间戳，随机数只能使用一次。

对称加密

把参数对称加密 aes

签名验证

ras

调用方，要提供公钥，sign（通过双方定义好的算法把摘要和参数计算后的值）

我们把post过来的参数先解密，通过制定的算法算出sign

我们看我们算出sign与调用方传过来的sign是否一致，一致则可以进行业务处理，不一致返回签名失败。

sign算法

secretKey是双方定义的摘要

params是参数的hashmap集合

```
byte[] data=Digest.getDigest(secretKey, params);

public class Digest {

     public static byte[] getDigest(String secretKey, Map<String,String> params) throws Exception{

             Set<String> keySet = params.keySet();

             TreeSet<String> sortSet = new TreeSet<>();

             sortSet.addAll(keySet);

             String keyValueStr = "";

             Iterator<String> it = sortSet.iterator();

             while(it.hasNext()){

                 String key = it.next();

                 String value = params.get(key);

                 keyValueStr += key + value;

             }
```

```
            keyValueStr = keyValueStr + secretKey;

            MessageDigest md = MessageDigest.getInstance("SHA-1");

            return md.digest(keyValueStr.getBytes("utf-8"));

        }

}

boolean b=RsaUtil.verify(data, sign, publicKeyString);
```

b=false 返回签名失败。

以下是rasutil

```
/**

 *

 */

package com.hlmedicals.app.util;

import Java.io.UnsupportedEncodingException;

import java.security.KeyFactory;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.spec.PKCS8EncodedKeySpec;

import java.security.spec.X509EncodedKeySpec;

import com.itextpdf.xmp.impl.Base64;

/**

 * @author dell

 *

 */

public class RsaUtil {

    public static void main (String [] args){

        try {

            byte[]  privateKeyString= IConst.privateKeyString.getBytes("utf-8");

            byte[]  publicKeyString= IConst.publicKeyString.getBytes("utf-8");

            String data1 = "testabc";

            //siyao签名

            byte[] data=data1.getBytes("utf-8");

            byte[] s = sign(data, privateKeyString);
```

```
            boolean b=verify(data,s,publicKeyString);
            System.out.println(b);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    /**
     * 使用私钥对数据进行加密签名
     * @param data 数据
     * @param privateKeyString 私钥
     * @return 加密后的签名
     */
    public static byte[] sign(byte[] data, byte[] privateKeyString) throws Exception {
        KeyFactory keyf = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyf.generatePrivate(new PKCS8EncodedKeySpec(Base64.decode(privateKeyString)));
        java.security.Signature signet = java.security.Signature.getInstance("SHA1withRSA");
        signet.initSign(privateKey);
        signet.update(data);
        byte[] signed = signet.sign();
        return Base64.encode(signed);
    }
    /**
     * 使用公钥判断签名是否与数据匹配
     * @param data 数据
     * @param sign 签名
     * @param publicKeyString 公钥
     * @return 是否篡改了数据
     */
    public static boolean verify(byte[] data, byte[] sign, byte[] publicKeyString) throws Exception {
        KeyFactory keyf = KeyFactory.getInstance("RSA");
        PublicKey publicKey = keyf.generatePublic(new X509EncodedKeySpec(Base64.decode(publicKeyString)));
        java.security.Signature signet = java.security.Signature.getInstance("SHA1withRSA");
```

```
            signet.initVerify(publicKey);

            signet.update(data);

            return signet.verify(Base64.decode(sign));

        }

    }
```

4.http 转成htts

5.后台登录要有验证码，没有验证码 ， 系统会被爆破，验证码不能被多次使用。

我的项目登录成功后把验证码存在session里。要把session验证码设定为空就可以。

6.系统上传文件时候，应该上传白名单的文件类型，防止jsp、jspx在系统中执行导致系统崩掉。

在开发过程中，肯定会有和第三方或者app端的接口调用。在调用的时候，如何来保证非法链接或者恶意攻击呢？

签名

根据用户名或者用户id，结合用户的ip或者设备号，生成一个token。在请求后台，后台获取http的head中的token，校验是否合法（和数据库或者Redis中记录的是否一致，在登录或者初始化的时候，存入数据库/redis）

在使用Base64方式的编码后，Token字符串还是有20多位，有的时候还是嫌它长了。由于GUID本身就有128bit，在要求有良好的可读性的前提下，很难进一步改进了。那我们如何产生更短的字符串呢？还有一种方式就是较少Token的长度，不用GUID，而采用一定长度的随机数，例如64bit，再用Base64编码表示：

```
    var rnd = new Random();

    var tokenData = userIp+userId;

    rnd.NextBytes(tokenData);

    var token = Convert.ToBase64String(tokenData).TrimEnd('=');
```

由于这里只用了64bit，此时得到的字符串为Onh0h95n7nw的形式，长度要短一半。这样就方便携带多了。但是这种方式是没有唯一性保证的。不过用来作为身份认证的方式还是可以的（如网盘的提取码）。

加密

客户端和服务器都保存一个秘钥，每次传输都加密，服务端根据秘钥解密。

客户端：

1、设置一个key（和服务器端相同）

2、根据上述key对请求进行某种加密（加密必须是可逆的，以便服务器端解密）

3、发送请求给服务器

服务器端：

1、设置一个key

2、根据上述的key对请求进行解密（校验成功就是「信任」的客户端发来的数据，否则拒绝响应）

3、处理业务逻辑并产生结果

4、将结果反馈给客户端

第三方支持

比如spring security – oauth

有兴趣的，可以参考这篇帖子

http://wwwcomy.iteye.com/blog/2230265

多数采用OAuth2，不过对于盗cookie没招，走HTTPS吧

api进行token校验或者签名，另外对于防刷，要进行限流

题主要看你如何定义接口安全了，我们可以看看新浪微博，每天无数的爬虫在爬，他的接口安全么？一般意义上安全指的是传输过程中不被盗取，走https基本可以做到，至于有人拿到api文档一类的东西（app被反向），那怎么样也防不了

所以如同楼上 1.对IP进行过滤，频繁访问的进行限制。 2.使用token令牌进行验证，通过后进行业务逻辑。 3.不对参数过渡暴露，传输的文本根据业务级别进行加密。

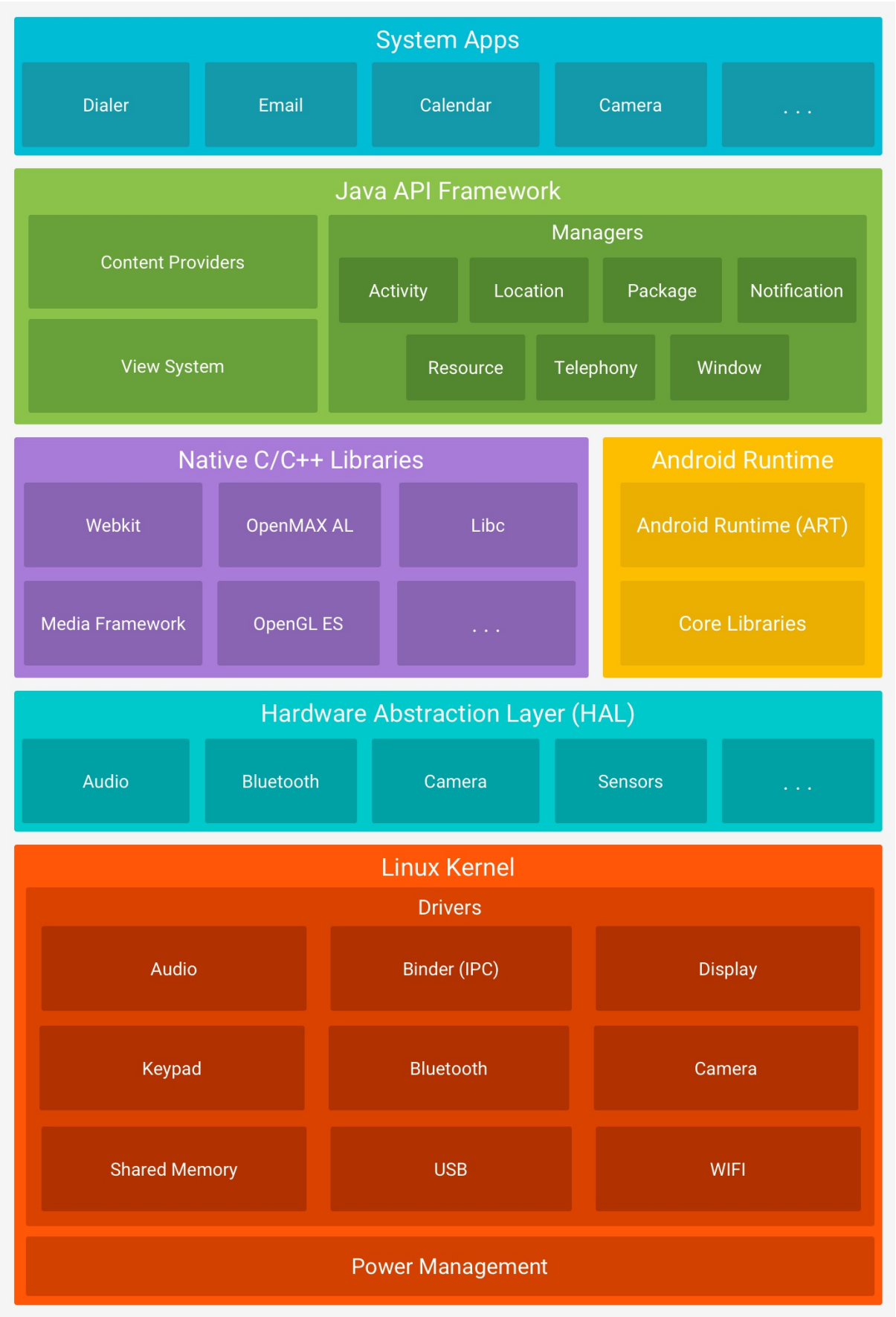
没有绝对的安全，题主可以看看微信开发者文档，看看他们的接口是怎么设计的

题主要看你如何定义接口安全了，我们可以看看新浪微博，每天无数的爬虫在爬，他的接口安全么？一般意义上安全指的是传输过程中不被盗取，走https基本可以做到，至于有人拿到api文档一类的东西（app被反向），那怎么样也防不了

所以如同楼上 1.对IP进行过滤，频繁访问的进行限制。 2.使用token令牌进行验证，通过后进行业务逻辑。 3.不对参数过渡暴露，传输的文本根据业务级别进行加密。

没有绝对的安全，题主可以看看微信开发者文档，看看他们的接口是怎么设计的

Android 是一种基于 Linux 的开放源代码软件栈，为广泛的设备和机型而创建。下图所示为 Android 平台的主要组件。


System Apps
Dialer
Email
Calendar
Camera
. . .
Java API Framework
Content Providers
View System
Managers
Activity
Location
Package
Notification
Resource
Telephony
Window
Native C/C++ Libraries
Webkit
OpenMAX AL
Libc
Media Framework
OpenGL ES
. . .
Android Runtime
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
. . .
Linux Kernel
Drivers
Audio
Binder (IPC)
Display
Keypad
Bluetooth
Camera
Shared Memory
USB
WIFI
Power Management

Linux 内核

Android 平台的基础是 Linux 内核。例如，Android Runtime (ART) 依靠 Linux 内核来执行底层功能，例如线程和低层内存管理。

使用 Linux 内核可让 Android 利用主要安全功能，并且允许设备制造商为著名的内核开发硬件驱动程序。

硬件抽象层 (HAL)

硬件抽象层 (HAL) 提供标准界面，向更高级别的 Java API 框架显示设备硬件功能。HAL 包含多个库模块，其中每个模块都为特定类型的硬件组件实现一个界面，例如相机或蓝牙模块。当框架 API 要求访问设备硬件时，Android 系统将为该硬件组件加载库模块。

Android Runtime

对于运行 Android 5.0（API 级别 21）或更高版本的设备，每个应用都在其自己的进程中运行，并且有其自己的 Android Runtime (ART) 实例。ART 编写为通过执行 DEX 文件在低内存设备上运行多个虚拟机，DEX 文件是一种专为 Android 设计的字节码格式，经过优化，使用的内存很少。编译工具链（例如 Jack）将 Java 源代码编译为 DEX 字节码，使其可在 Android 平台上运行。

ART 的部分主要功能包括：

- 预先 (AOT) 和即时 (JIT) 编译
- 优化的垃圾回收 (GC)
- 更好的调试支持，包括专用采样分析器、详细的诊断异常和崩溃报告，并且能够设置监视点以监控特定字段

在 Android 版本 5.0（API 级别 21）之前，Dalvik 是 Android Runtime。如果您的应用在 ART 上运行效果很好，那么它应该也可在 Dalvik 上运行，但反过来不一定。

Android 还包含一套核心运行时库，可提供 Java API 框架使用的 Java 编程语言大部分功能，包括一些 Java 8 语言功能。

原生 C/C++ 库

许多核心 Android 系统组件和服务（例如 ART 和 HAL）构建自原生代码，需要以 C 和 C++ 编写的原生库。Android 平台提供 Java 框架 API 以向应用显示其中部分原生库的功能。例如，您可以通过 Android 框架的 Java OpenGL API 访问 OpenGL ES，以支持在应用中绘制和操作 2D 和 3D 图形。

如果开发的是需要 C 或 C++ 代码的应用，可以使用 Android NDK 直接从原生代码访问某些原生平台库。

Java API 框架

您可通过以 Java 语言编写的 API 使用 Android OS 的整个功能集。这些 API 形成创建 Android 应用所需的构建块，它们可简化核心模块化系统组件和服务的重复使用，包括以下组件和服务：

- 丰富、可扩展的视图系统，可用以构建应用的 UI，包括列表、网格、文本框、按钮甚至可嵌入的网络浏览器
- 资源管理器，用于访问非代码资源，例如本地化的字符串、图形和布局文件
- 通知管理器，可让所有应用在状态栏中显示自定义提醒
- Activity 管理器，用于管理应用的生命周期，提供常见的导航返回栈

- 内容提供程序，可让应用访问其他应用（例如“联系人”应用）中的数据或者共享其自己的数据

开发者可以完全访问 Android 系统应用使用的框架 API。

系统应用

Android 随附一套用于电子邮件、短信、日历、互联网浏览和联系人等的核心应用。平台随附的应用与用户可以选择安装的应用一样，没有特殊状态。因此第三方应用可成为用户的默认网络浏览器、短信 Messenger 甚至默认键盘（有一些例外，例如系统的“设置”应用）。

系统应用可用作用户的应用，以及提供开发者可从其自己的应用访问的主要功能。例如，如果您的应用要发短信，您无需自己构建该功能，可以改为调用已安装的短信应用向您指定的接收者发送消息。

源码分析相关面试题

- Volley源码剖析
- 注解框架内部实现原理
- okhttp内核剖析
- Android源码编译实现静默安装和静默偷拍

今天就带大家一层层剥光Volley这位懵懂无知少女（14年出生，应该算无知少女）的外衣，带大家轻轻抚摸Volley每一寸肌肤，进入Volley的身体，为达到完美效果，开头录有激情小视频，观看时请自备纸巾，各位现在需要做的就是搬一把椅子，打开电脑，双击Android studio导入Volley源码，随我一起观看录制的激情小视频带着大家一起飞，观看激情小视频过程中，如能让你内心微微一颤有所收获，感受到满满的爱意，点赞或打赏都是最美的祝福。

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频：

- Volley内核分析配套视频一
- Volley内核分析配套视频二
- Volley内核分析配套视频三

通过一个小栗子开始咱们的源码分析

```
RequestQueue queue = Volley.newRequestQueue(this);
     String url ="http://www.baidu.com";
    StringRequest stringRequest = new  StringRequest(Request.Method.GET,
    url,
    new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {

      }
    }, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {

     }
     });
 queue.add(stringRequest);
```

第一部分：一行行分析

```
RequestQueue queue = Volley.newRequestQueue(this);
```

进入源码分析:

```
public static RequestQueue newRequestQueue(Context context) {
   return newRequestQueue(context, (HttpStack)null);
}
```

由以上源码分析可知:

1）该方法有两个参数，第一个Context是上下文，第二个参数为null

```
public static RequestQueue newRequestQueue(Context context, HttpStack stack) {
   File cacheDir = new File(context.getCacheDir(), "volley");
   String userAgent = "volley/0";
   try {
     String network = context.getPackageName();
     PackageInfo queue = context.getPackageManager().getPackageInfo(network, 0);
     userAgent = network + "/" + queue.versionCode;
   } catch (NameNotFoundException var6) {

   }
   if(stack == null) {
     if(VERSION.SDK_INT >= 9) {
        stack = new HurlStack();
   } else {
        stack = new HttpClientStack(AndroidHttpClient.newInstance(userAgent));
   }
 }

 BasicNetwork network1 = new BasicNetwork((HttpStack)stack);
```

```
 RequestQueue queue1 = new RequestQueue(new  DiskBasedCache(cacheDir), network1);
  queue1.start();
  return queue1;
 }
```

由以上源码分析可知:

1）初始化缓存文件，名字叫volley。

2）获取到当前应用包名和包的基本信息，并且给userAgent 重新赋值。

3）stack 传递过来为null，所以直接走if判断里面。

4）在这个if中对版本号进行了判断，如果版本号大于等于9就使得HttpStack对象的实例为HurlStack，如果小于9则实例为HttpClientStack。至于这里为何进行版本号判断，实际代码中已经说明了，请看下文。

5) BasicNetwork是Network接口的实现，BasicNetwork实现了performRequest方法，其作用是根据传入的HttpStack对象来处理网络请求。紧接着new出一个RequestQueue对象，并调用它的start()方法进行启动，然后将RequestQueue返回。RequestQueue是根目录下的一个类，其作用是一个请求调度队列调度程序的线程池。这样newRequestQueue()的方法就执行结束了。

> 第四条如何分析出来请看如下代码:

```
public class HurlStack implements HttpStack
public class HttpClientStack implements HttpStack
```

继承httpStack接口

```
public interface HttpStack {
    HttpResponse performRequest(Request<?> var1, Map<String, String> var2) throws IOException, AuthFailureError;
}
```

接口当中有如下方法，performRequest，大家有没有觉得奇怪都继承同一个方法，并且都实现一样的接口同样的方法，实际上子类实现方法不一样，代码如下:

HttpClientStack代码如下:

```
public HttpResponse performRequest(Request<?> request, Map<String, String> additionalHeaders) throws IOException, Aut
hFailureError {
        HttpUriRequest httpRequest = createHttpRequest(request, additionalHeaders);
        addHeaders(httpRequest, additionalHeaders);
        addHeaders(httpRequest, request.getHeaders());
        this.onPrepareRequest(httpRequest);
        HttpParams httpParams = httpRequest.getParams();
        int timeoutMs = request.getTimeoutMs();
        HttpConnectionParams.setConnectionTimeout(httpParams, 5000);
        HttpConnectionParams.setSoTimeout(httpParams, timeoutMs);
        return this.mClient.execute(httpRequest);
    }
```

HurlStack代码如下：

```
public HttpResponse performRequest(Request<?> request, Map<String, String> additionalHeaders) throws IOException, Aut
hFailureError {
     ......
```

```
        URL parsedUrl1 = new URL(url);
        HttpURLConnection connection = this.openConnection(parsedUrl1, request);

     .......
           return response;
        }
    }
```

由以上得知HttpClientStack走的是httpClient，HurlStack走的是HttpURLConnection，谷歌在高版本当中已经去掉了httpClient，所以这里必须做版本号的判断。

继续源码分析，现在再来看下RequestQueue队列的start方法，如下所示：

```
public void start() {
        this.stop();
        this.mCacheDispatcher = new CacheDispatcher(this.mCacheQueue, this.mNetworkQueue, this.mCache, this.mDelivery
);
        this.mCacheDispatcher.start();

for(int i = 0; i < this.mDispatchers.length; ++i) {
   NetworkDispatcher networkDispatcher = new      NetworkDispatcher(this.mNetworkQueue, this.mNetwork, this.mCache, t
his.mDelivery);
   this.mDispatchers[i] = networkDispatcher;
   networkDispatcher.start();
 }
}
```

以上源码可知：

1）创建了一个CacheDispatcher的实例，然后调用了它的start()方法

2）for循环里去创建NetworkDispatcher的实例，并分别调用它们的start()方法

3）这里的CacheDispatcher和NetworkDispatcher都是继承自Thread的，而默认情况下for循环会执行四次，也就是说当调用了Volley.newRequestQueue(context)之后，就会有五个线程一直在后台运行，不断等待网络请求的到来，其中一个CacheDispatcher是缓存线程，四个NetworkDispatcher是网络请求线程。

> 第三条如何分析出来for循环会执行四次，请看如下代码分析：

```
public RequestQueue(Cache cache, Network network, int threadPoolSize, ResponseDelivery delivery) {
        ......
        this.mDispatchers = new NetworkDispatcher[threadPoolSize];
        ......
    }
```

threadPoolSize大小是4，如何看出来请看如下代码：

```
 public RequestQueue(Cache cache, Network network, int threadPoolSize) {
        this(cache, network, threadPoolSize, new ExecutorDelivery(new Handler(Looper.getMainLooper())));
    }

    public RequestQueue(Cache cache, Network network) {
        this(cache, network, 4);
    }
```

通过构造方法传递，默认是4，如上已经把第一行代码分析完毕，进入第二部分。

```
RequestQueue queue = Volley.newRequestQueue(this);
```

第二部分：添加请求到队列

```
StringRequest stringRequest = new StringRequest(Request.Method.GET,
                url,
new Response.Listener<String>() {
@Override
public void onResponse(String response) {

 }
}, new Response.ErrorListener() {

 @Override
 public void onErrorResponse(VolleyError error) {

 }
 });
 queue.add(stringRequest);
```

调用RequestQueue的add()方法将Request传入就可以完成网络请求操作了。也就是说add()方法的内部是核心代码了。现在看下RequestQueue的add方法，具体如下：

```
public Request add(Request request) {
        request.setRequestQueue(this);
        Set var2 = this.mCurrentRequests;
        synchronized(this.mCurrentRequests) {
            this.mCurrentRequests.add(request);
        }

        request.setSequence(this.getSequenceNumber());
        request.addMarker("add-to-queue");
        if(!request.shouldCache()) {
            this.mNetworkQueue.add(request);
            return request;
        } else {
       Map var7 = this.mWaitingRequests;
       synchronized(this.mWaitingRequests) {
       String cacheKey = request.getCacheKey();
       if(this.mWaitingRequests.containsKey(cacheKey)) {
       Object stagedRequests = (Queue)this.mWaitingRequests.get(cacheKey);
       if(stagedRequests == null) {
         stagedRequests = new LinkedList();
       }
       ((Queue)stagedRequests).add(request);
       this.mWaitingRequests.put(cacheKey, stagedRequests);
      }
     } else {
         this.mWaitingRequests.put(cacheKey, (Object)null);
         this.mCacheQueue.add(request);
     }

       return request;

    }
```

通过以上源码可知：

1）Request是所有请求的基类，是一个抽象类。request.setRequestQueue(this);的作用就是将请求Request关联到当前RequestQueue。然后同步操作将当前Request添加到RequestQueue对象的mCurrentRequests HashSet中做记录。

2）通过request.setSequence(getSequenceNumber());得到当前RequestQueue中请求的个数，然后关联到当前Request。

3）request.addMarker("add-to-queue");添加调试的队列标记。

4）if (!request.shouldCache())判断当前的请求是否可以缓存，如果不能缓存则直接通过mNetworkQueue.add(request);将这条请求加入网络请求队列，然后返回request；如果可以缓存的话则在通过同步操作将这条请求加入缓存队列。

第二部分分析完毕，进入第三部分。

第三部分：CacheDispatcher中的run()方法，代码如下所示：

```
public void run() {
  if(DEBUG) {
     VolleyLog.v("start new dispatcher", new Object[0]);
  }

  Process.setThreadPriority(10);
  this.mCache.initialize();


  while(true) {
       final Request e = (Request)this.mCacheQueue.take();
       e.addMarker("cache-queue-take");
       if(e.isCanceled()) {
         e.finish("cache-discard-canceled");
       } else {
        Entry entry = this.mCache.get(e.getCacheKey());
       if(entry == null) {
         e.addMarker("cache-miss");
         this.mNetworkQueue.put(e);
 } else if(entry.isExpired()) {
       e.addMarker("cache-hit-expired");
       e.setCacheEntry(entry);
       this.mNetworkQueue.put(e);
 } else {
      e.addMarker("cache-hit");
      Response response = e.parseNetworkResponse(new  NetworkResponse(entry.data, entry.responseHeaders));
      e.addMarker("cache-hit-parsed");
      if(entry.refreshNeeded()) {
        e.addMarker("cache-hit-refresh-needed");
        e.setCacheEntry(entry);
        response.intermediate = true;
       this.mDelivery.postResponse(e, response,
       new Runnable() {
        public void run() {
          CacheDispatcher.this.mNetworkQueue.put(e);
       }
```

以上源码可知：

1）首先通过Process.setThreadPriority设置线程优先级

2）mCache.initialize(); 初始化缓存块

3）while(true)循环，表示它一直在等待缓存队列的新请求的出现

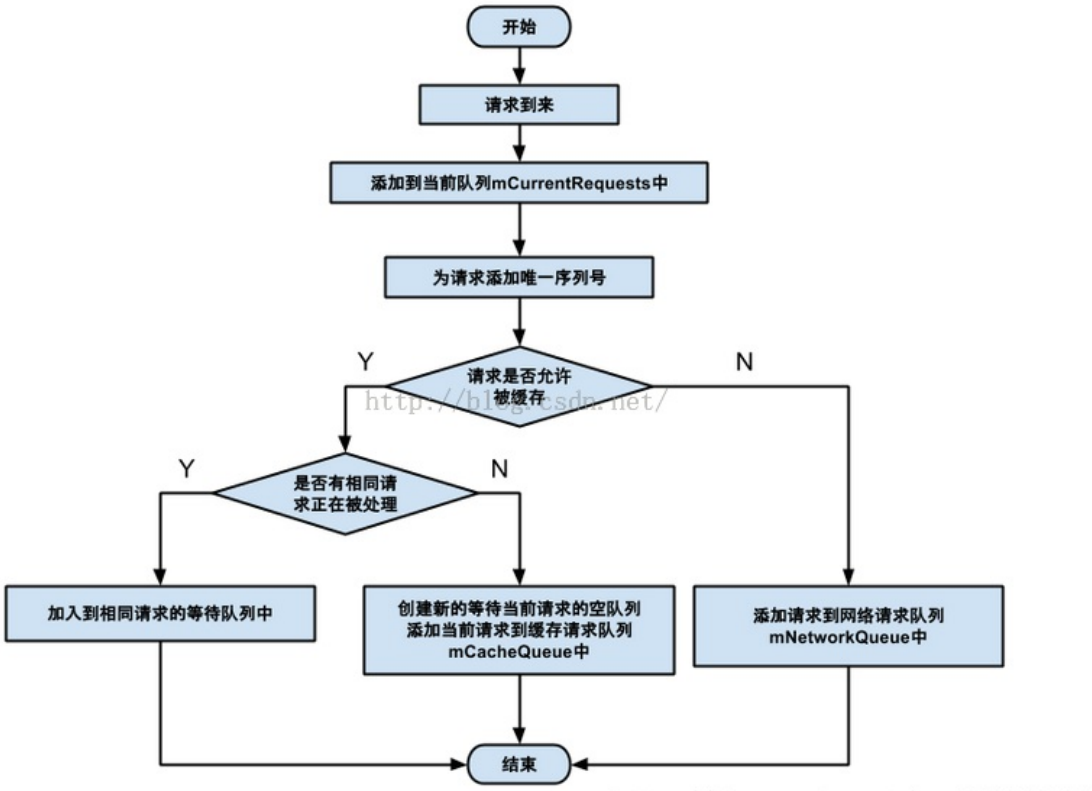
4）接着，先判断这个请求是否有对应的缓存结果，如果没有则直接添加到网络请求队列

5）再判断这个缓存结果是否过期了，如果过期则同样地添加到网络请求队列

6) 接下来便是对缓存结果的处理了，我们可以看到，先是把缓存结果包装成NetworkResponse类，然后调用了Request的parseNetworkResponse;

小结

CacheDispatcher线程主要对请求进行判断，是否已经有缓存，是否已经过期，根据需要放进网络请求队列。同时对相应结果进行包装、处理，然后交由ExecutorDelivery处理。这里以一张流程图显示它的完整工作流程：



第四部分：NetworkDispatcher代码如下所示：

```
public void run() {
   Process.setThreadPriority(10);
   while(true) {
      Request request;
      while(true) {
        try {
         request = (Request)this.mQueue.take();
         break;
        } catch (InterruptedException var4) {

       }
     try {
        request.addMarker("network-queue-take");
        if(request.isCanceled()) {
           request.finish("network-discard-cancelled");
        } else {
           ......
       NetworkResponse e = this.mNetwork.performRequest(request);
       request.addMarker("network-http-complete");
       if(e.notModified && request.hasHadResponseDelivered()) {
         request.finish("not-modified");
        } else {
         Response response = request.parseNetworkResponse(e);
         request.addMarker("network-parse-complete");
```

```
        if(request.shouldCache() && response.cacheEntry != null) {
                            this.mCache.put(request.getCacheKey(), response.cacheEntry);
         request.addMarker("network-cache-written");
        }
 request.markDelivered();
 this.mDelivery.postResponse(request, response);
 }
```

由以上代码分析可知：

1）通过mNetwork.performRequest(request);代码来发送网络请求

2）而Network是一个接口，这里具体的实现之前已经分析是BasicNetwork，所以先看下它的performRequest()方法，如下所示：

```
public NetworkResponse performRequest(Request<?> request) throws VolleyError {
    long requestStart = SystemClock.elapsedRealtime();
    while(true) {
      HttpResponse httpResponse = null;
      Object responseContents = null;
      HashMap responseHeaders = new HashMap();
      try {
       HashMap e = new HashMap();
       this.addCacheHeaders(e, request.getCacheEntry());
       httpResponse = this.mHttpStack.performRequest(request, e);
      ......
      byte[] responseContents1;
      if(httpResponse.getEntity() != null) {
          responseContents1 = this.entityToBytes(httpResponse.getEntity());
      } else {
          responseContents1 = new byte[0];
      }

   long requestLifetime = SystemClock.elapsedRealtime() - requestStart;
   this.logSlowRequests(requestLifetime, request, responseContents1, statusCode2);
   if(networkResponse1 >= 200 && networkResponse1 <= 299) {
     return new NetworkResponse(networkResponse1, responseContents1, responseHeaders1, false);
  }
}
```

由上述代码可知：

1）httpResponse = this.mHttpStack.performRequest(request, e)该方法返回了httpResponse

2）把httpResponse 交给 new NetworkResponse对象进行处理，封装成NetworkResponse对象并返回

3）在NetworkDispatcher#run()方法获取返回的NetworkResponse对象后，对响应解析

4）在解析完了NetworkResponse中的数据之后，又会调用ExecutorDelivery（ResponseDelivery接口的实现类）的postResponse()方法来回调解析出的数据，具体代码如下所示：

```
 this.mDelivery.postResponse(request, response);
```

```
public void postResponse(Request<?> request, Response<?> response, Runnable runnable) {
        request.markDelivered();
        request.addMarker("post-response");
        this.mResponsePoster.execute(new ExecutorDelivery.ResponseDeliveryRunnable(request, response, runnable));
}
```

这里可以看见在mResponsePoster的execute()方法中传入了一个ResponseDeliveryRunnable对象，就可以保证该对象中的run()方法就是在主线程当中运行的了，我们看下run()方法中的代码是什么样的：

```
if(this.mResponse.isSuccess()) {
                    this.mRequest.deliverResponse(this.mResponse.result);
 } else {
                this.mRequest.deliverError(this.mResponse.error);
}
```

以上代码可知

mRequest的deliverResponse或者deliverError将反馈发送到回调到UI线程。这也是你重写实现的接口方法，到目前为止，关于Volley的源码解析完毕。

总结

1）当一个RequestQueue被成功申请后会开启一个CacheDispatcher和4个默认的NetworkDispatcher。

2）CacheDispatcher缓存调度器最为第一层缓冲，开始工作后阻塞的从缓存序列mCacheQueue中取得请求；对于已经取消的请求，标记为跳过并结束这个请求；新的或者过期的请求，直接放入mNetworkQueue中由N个NetworkDispatcher进行处理；已获得缓存信息（网络应答）却没有过期的请求，由Request的parseNetworkResponse进行解析，从而确定此应答是否成功。然后将请求和应答交由Delivery分发者进行处理，如果需要更新缓存那么该请求还会被放入mNetworkQueue中。

3）将请求Request add到RequestQueue后对于不需要缓存的请求（需要额外设置，默认是需要缓存）直接丢入mNetworkQueue交给N个NetworkDispatcher处理；对于需要缓存的，新的请求加到mCacheQueue中给CacheDispatcher处理；需要缓存，但是缓存列表中已经存在了相同URL的请求，放在mWaitingQueue中做暂时处理，等待之前请求完毕后，再重新添加到mCacheQueue中。

4）网络请求调度器NetworkDispatcher作为网络请求真实发生的地方，对消息交给BasicNetwork进行处理，同样的，请求和结果都交由Delivery分发者进行处理。

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频：

- 配套视频

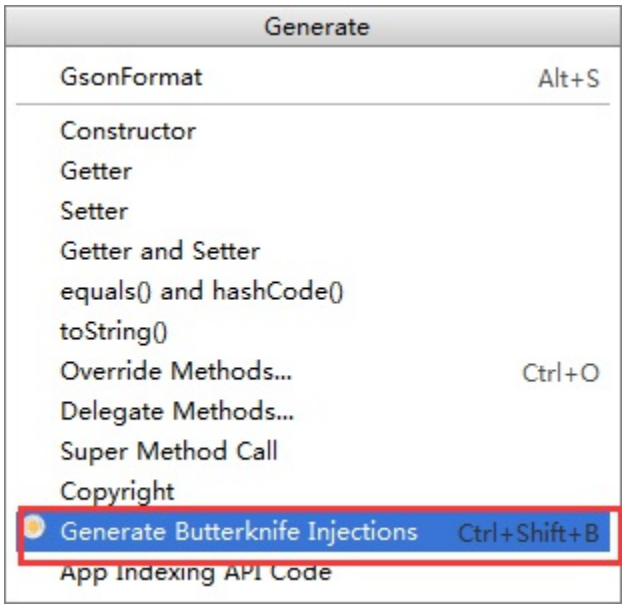
注解框架实现原理，手写ButterKnife实现自己的注解框架

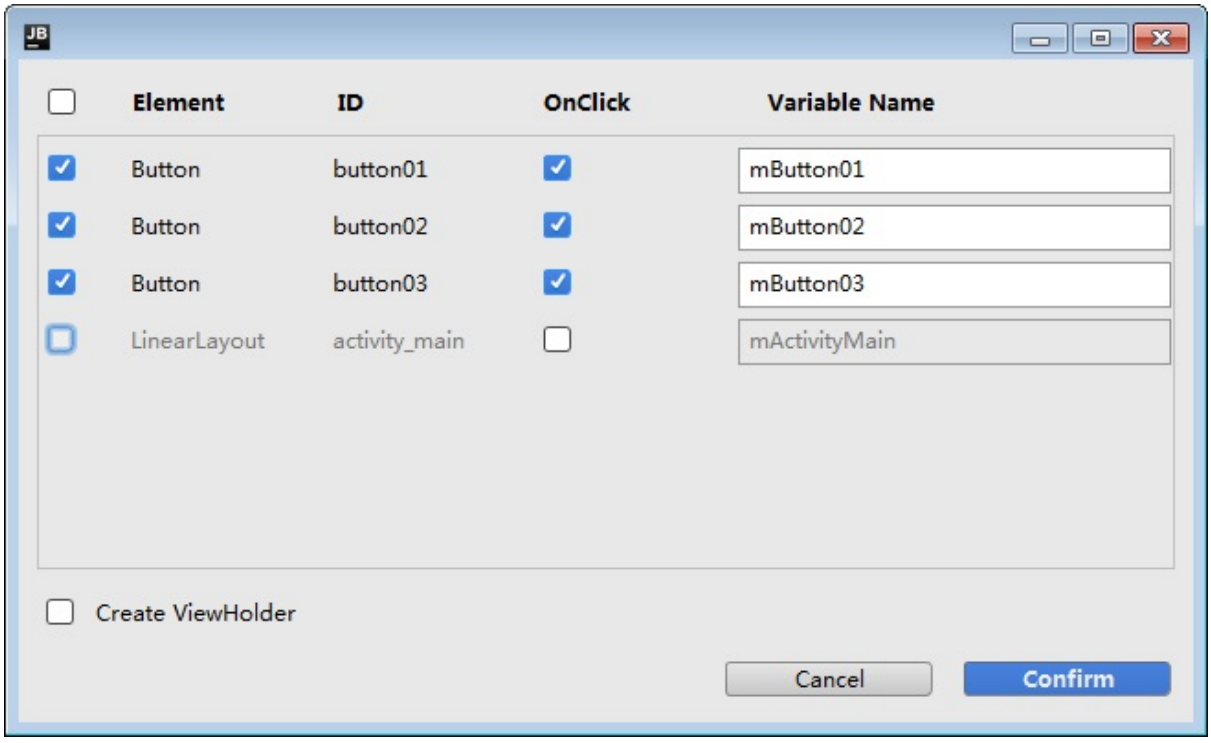
初级程序员使用别人的框架，中级程序员不仅会使用别人的框架还知道内部的实现原理，高级程序员则按需编写自己的框架。添加该模块的目的就是想提交大家的逼格，让大家养成一个动手编写“自主知识产权”框架的意识。

1. 编写 ButterKnife框架

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：

ButterKnife使用如下：

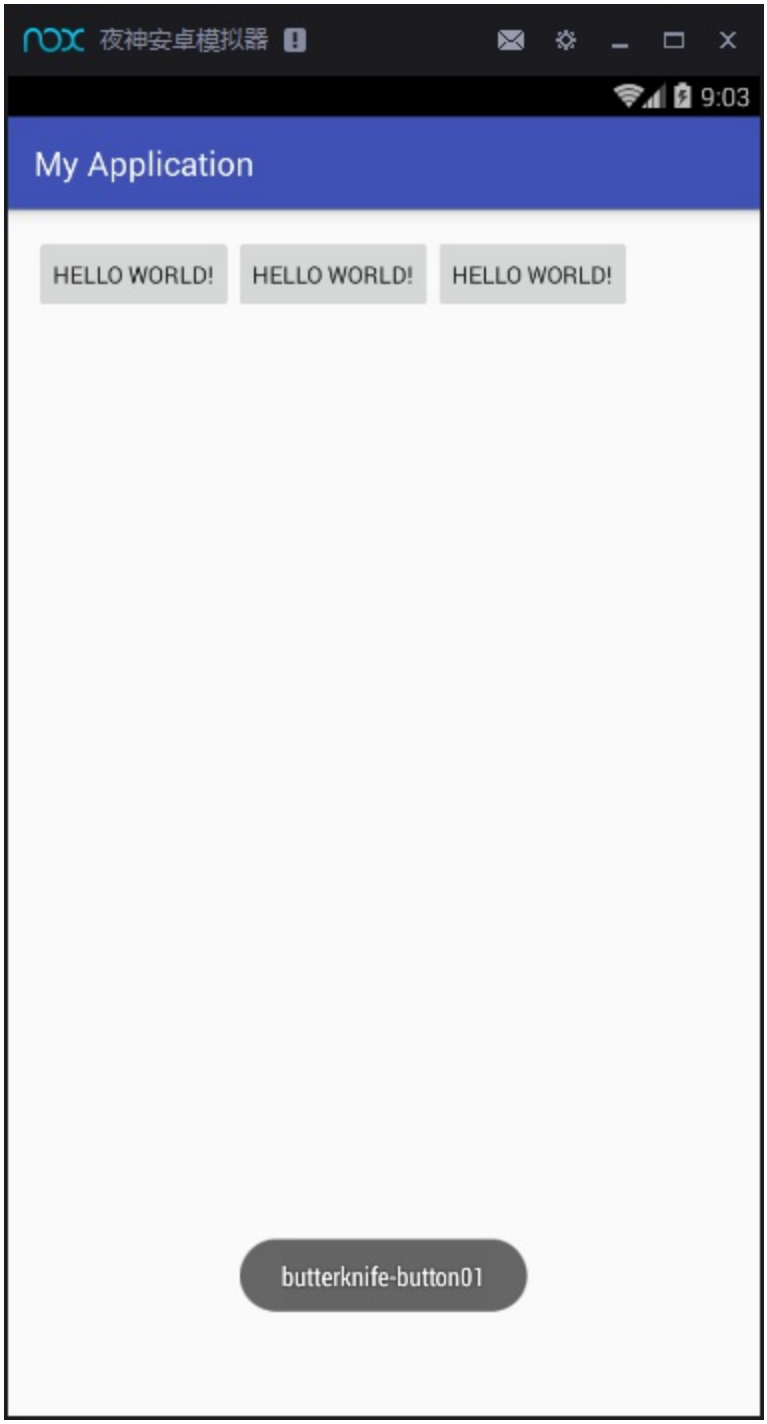




会出现如下代码：

```
    @BindView(R.id.button01)
     Button mButton01;
     @BindView(R.id.button02)
     Button mButton02;
     @BindView(R.id.button03)
     Button mButton03;

         @butterknife.OnClick({R.id.button01, R.id.button02, R.id.button03})
     public void onClick(View view) {
         switch (view.getId()) {
           case R.id.button01:
           Toast.makeText(this, "butterknife-button01", Toast.LENGTH_SHORT).show();
                 break;
           case R.id.button02:
            Toast.makeText(this, "butterknife-button02", Toast.LENGTH_SHORT).show();
                 break;
           case R.id.button03:
           Toast.makeText(this, "butterknife-button03", Toast.LENGTH_SHORT).show();
                 break;
         }
     }
```

点击每个按钮会弹出响应的Toast，如下：



这个就是ButterKnife的用法。

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

```
/**
```

```
* @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>
  * 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
 * 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
 * 3. RetentionPolicy.RUNTIME 注释永久保留，可以被VM加载时加载到内存中
 * 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
 *
 * @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，
       由于我们的当前注解类是想声明在Filed上面
 * 因此这里设置为ElementType.FIELD。
 * 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。
 *
 * @interface 是声明注解类的组合关键字。
 */

@Target({java.lang.annotation.ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface ViewInject {
    public abstract int value();
}
```

【文件】Click.java

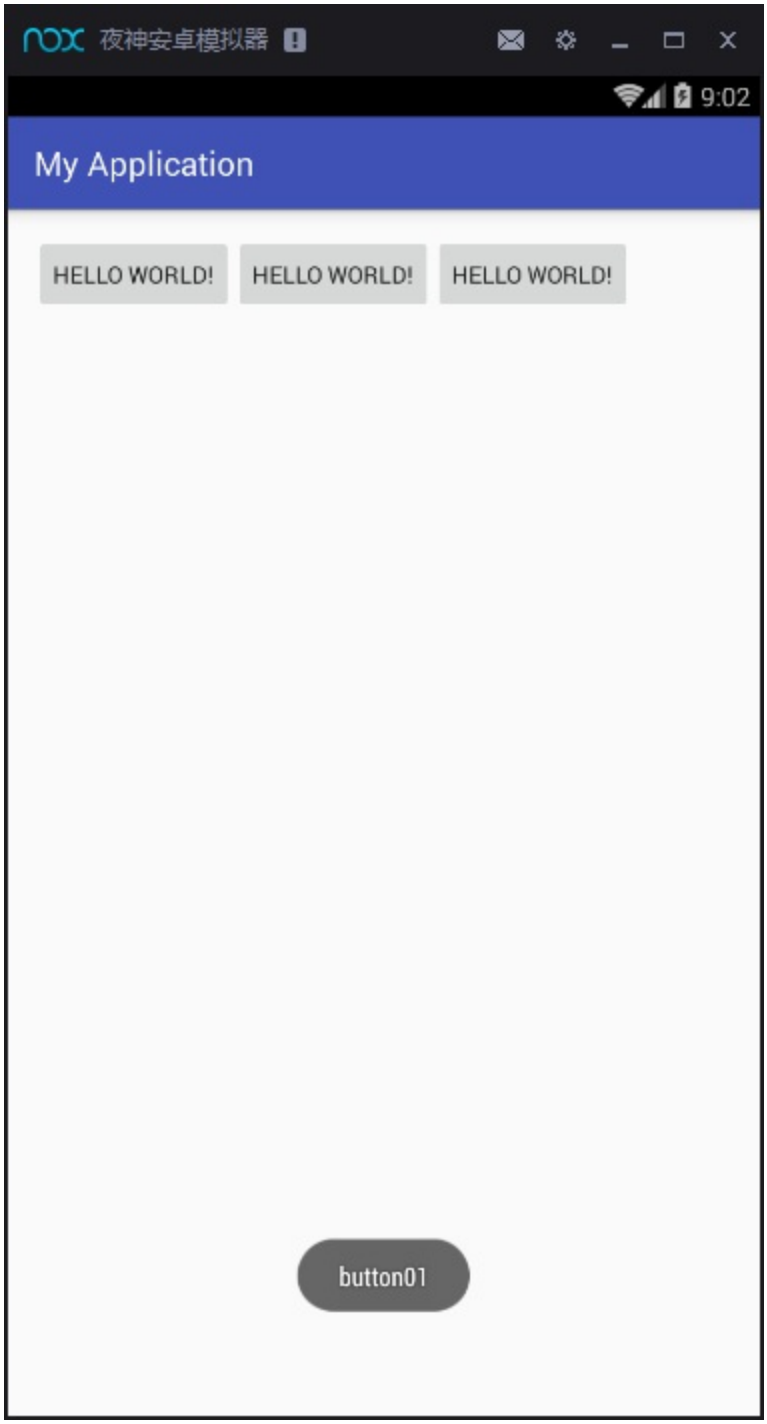
```
@Target({java.lang.annotation.ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface OnClick
{
    public abstract int[] value();
}
```

编写核心方法 ViewUtilsTest.inject(Ativity);

```
public class ViewUtilsTest {
    public static void inject(final Activity activity)
    {
        /**
         * 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(),
         * 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。
         */

        Class clazz = activity.getClass();
        Field[] declaredFields = clazz.getDeclaredFields();
        //一个Activity中可能有多个Field，因此遍历。
        for (int i = 0; i < declaredFields.length; i++) {
            Field field = declaredFields[i];
            //设置为可访问，暴力反射，就算是私有的也能访问到
            field.setAccessible(true);
         //获取到字段上面的注解对象
         ViewInject annotation = (ViewInject)field.getAnnotation(ViewInject.class);
            //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
            if (annotation == null)
            {
                continue;
            }
            //获取注解中的值
            int id = annotation.value();
            //获取控件
            View view = activity.findViewById(id);

            try
            {
                //将该控件设置给field对象
                field.set(activity, view);
```

```
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }

        }
        //获取所有的方法（私有方法也可以获取到）
        Method[] declaredMethods = clazz.getDeclaredMethods();
        for (int i = 0; i < declaredMethods.length; i++) {
            final Method method = declaredMethods[i];
            //获取方法上面的注解
            OnClick annotation = (OnClick)method.getAnnotation(OnClick.class);
            if (annotation == null) {
                //如果该方法上没有注解，循环下一个
                continue;
            }
            //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。
            int[] value = annotation.value();
            for (int j = 0; j < value.length; j++) {
                int id = value[j];

                final View button = activity.findViewById(id);

                button.setOnClickListener(new View.OnClickListener()
                {
                    public void onClick(View v)
                    {
                        try
                        {
                            //反射调用用户指定的方法
                           method.invoke(activity,button);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                });
            }
        }
    }
}
```

咱们自己的注解框架就实现好了，效果如下：



- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

okhttp源码特别特别复杂，类涉及较多，导致本文非常长，我相信没有几个人能把本文看完，所以特意录制了跟文章同步的视频。

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频：

- okhttp内核分析配套视频一

- okhttp内核分析配套视频二
- okhttp内核分析配套视频三

基本使用

从使用方法出发，首先是怎么使用，其次是我们使用的功能在内部是如何实现的.建议大家下载 OkHttp 源码之后，跟着本文，过一遍源码。

官方博客栗子：http://square.github.io/okhttp/#examples

```
OkHttpClient client = new OkHttpClient();

String run(String url) throws IOException {
  Request request = new Request.Builder()
      .url(url)
      .build();

  Response response = client.newCall(request).execute();
  return response.body().string();
}
```

Request、Response、Call 基本概念

上面的代码中涉及到几个常用的类：Request、Response和Call。下面分别介绍：

Request

每一个HTTP请求包含一个URL、一个方法（GET或POST或其他）、一些HTTP头。请求还可能包含一个特定内容类型的数据类的主体部分。

Response

响应是对请求的回复，包含状态码、HTTP头和主体部分。

Call

OkHttp使用Call抽象出一个满足请求的模型，尽管中间可能会有多个请求或响应。执行Call有两种方式，同步或异步

第一步：创建 OkHttpClient对象,进行源码分析：

```
OkHttpClient client = new OkHttpClient();
```

通过okhttp源码分析,直接创建的 OkHttpClient对象并且默认构造builder对象进行初始化

```
public class OkHttpClient implements Cloneable, Call.Factory, WebSocket.Factory {
  public OkHttpClient() {
       this(new Builder());
  }
  OkHttpClient(Builder builder) {
    this.dispatcher = builder.dispatcher;
    this.proxy = builder.proxy;
    this.protocols = builder.protocols;
    this.connectionSpecs = builder.connectionSpecs;
```

```
    this.interceptors = Util.immutableList(builder.interceptors);
    this.networkInterceptors = Util.immutableList(builder.networkInterceptors);
    this.eventListenerFactory = builder.eventListenerFactory;
    this.proxySelector = builder.proxySelector;
    this.cookieJar = builder.cookieJar;
    this.cache = builder.cache;
    this.internalCache = builder.internalCache;
    this.socketFactory = builder.socketFactory;

    boolean isTLS = false;
    ......

    this.hostnameVerifier = builder.hostnameVerifier;
    this.certificatePinner = builder.certificatePinner.withCertificateChainCleaner(
        certificateChainCleaner);
    this.proxyAuthenticator = builder.proxyAuthenticator;
    this.authenticator = builder.authenticator;
    this.connectionPool = builder.connectionPool;
    this.dns = builder.dns;
    this.followSslRedirects = builder.followSslRedirects;
    this.followRedirects = builder.followRedirects;
    this.retryOnConnectionFailure = builder.retryOnConnectionFailure;
    this.connectTimeout = builder.connectTimeout;
    this.readTimeout = builder.readTimeout;
    this.writeTimeout = builder.writeTimeout;
    this.pingInterval = builder.pingInterval;
  }
}
```

第二步：接下来发起 HTTP 请求

```
Request request = new Request.Builder().url("url").build();
okHttpClient.newCall(request).enqueue(new Callback() {
  @Override
  public void onFailure(Call call, IOException e) {

 }

@Override
public void onResponse(Call call, Response response) throws IOException {

}
});
```

第二步：代码流程分析：

```
Request request = new Request.Builder().url("url").build();
```

初始化构建者模式和请求对象，并且用URL替换Web套接字URL。

```
public final class Request {
    public Builder() {
      this.method = "GET";
      this.headers = new Headers.Builder();
    }
    public Builder url(String url) {
      ......

      // Silently replace web socket URLs with HTTP URLs.
      if (url.regionMatches(true, 0, "ws:", 0, 3)) {
```

```
        url = "http:" + url.substring(3);
      } else if (url.regionMatches(true, 0, "wss:", 0, 4)) {
        url = "https:" + url.substring(4);
      }

      HttpUrl parsed = HttpUrl.parse(url);
      ......
      return url(parsed);
    }
    public Request build() {
      ......
      return new Request(this);
    }
}
```

第三步：方法解析：

```
okHttpClient.newCall(request).enqueue(new Callback() {
@Override
public void onFailure(Call call, IOException e) {

}

@Override
public void onResponse(Call call, Response response) throws IOException {

}
});
```

源码分析:

```
public class OkHttpClient implements Cloneable, Call.Factory, WebSocket.Factory {
   @Override
   public Call newCall(Request request) {
    return new RealCall(this, request, false /* for web socket */);
   }



}
```

RealCall实现了Call.Factory接口创建了一个RealCall的实例，而RealCall是Call接口的实现。

异步请求的执行流程

```
final class RealCall implements Call {
   @Override
   public void enqueue(Callback responseCallback) {
   synchronized (this) {
   if (executed) throw new IllegalStateException("Already Executed");
      executed = true;
   }
    captureCallStackTrace();
    client.dispatcher().enqueue(new AsyncCall(responseCallback));
  }
}
```

由以上源码得知：

1） 检查这个 call 是否已经被执行了，每个 call 只能被执行一次，如果想要一个完全一样的 call，可以利用 call#clone 方法进行克隆。

2）利用 client.dispatcher().enqueue(this) 来进行实际执行，dispatcher 是刚才看到的 OkHttpClient.Builder 的成员之一

3）AsyncCall是RealCall的一个内部类并且继承NamedRunnable，那么首先看NamedRunnable类是什么样的，如下：

```
public abstract class NamedRunnable implements Runnable {
  ......

  @Override
  public final void run() {
   ......
    try {
      execute();
    }
    ......
  }

  protected abstract void execute();
}
```

可以看到NamedRunnable实现了Runnbale接口并且是个抽象类，其抽象方法是execute()，该方法是在run方法中被调用的，这也就意味着NamedRunnable是一个任务，并且其子类应该实现execute方法。下面再看AsyncCall的实现：

```
final class AsyncCall extends NamedRunnable {
    private final Callback responseCallback;

    AsyncCall(Callback responseCallback) {
      super("OkHttp %s", redactedUrl());
      this.responseCallback = responseCallback;
    }

    ......
final class RealCall implements Call {
  @Override protected void execute() {
  boolean signalledCallback = false;
  try {
     Response response = getResponseWithInterceptorChain();
  if (retryAndFollowUpInterceptor.isCanceled()) {
     signalledCallback = true;
     responseCallback.onFailure(RealCall.this, new IOException("Canceled"));
  } else {
    signalledCallback = true;
    responseCallback.onResponse(RealCall.this, response);
  }
 } catch (IOException e) {
  ......
  responseCallback.onFailure(RealCall.this, e);

} finally {
    client.dispatcher().finished(this);
  }
}
```

AsyncCall实现了execute方法，首先是调用getResponseWithInterceptorChain()方法获取响应，然后获取成功后，就调用回调的onReponse方法，如果失败，就调用回调的onFailure方法。最后，调用Dispatcher的finished方法。

关键代码：

responseCallback.onFailure(RealCall.this, new IOException("Canceled"));

和

responseCallback.onResponse(RealCall.this, response);

走完这两句代码会进行回调到刚刚我们初始化Okhttp的地方,如下：

```
okHttpClient.newCall(request).enqueue(new Callback() {
   @Override
   public void onFailure(Call call, IOException e) {

   }

   @Override
   public void onResponse(Call call, Response response) throws IOException {

   }
});
```

核心重点类Dispatcher线程池介绍

```
public final class Dispatcher {
  /** 最大并发请求数为64 */
  private int maxRequests = 64;
  /** 每个主机最大请求数为5 */
  private int maxRequestsPerHost = 5;

  /** 线程池 */
  private ExecutorService executorService;

  /** 准备执行的请求 */
  private final Deque<AsyncCall> readyAsyncCalls = new ArrayDeque<>();

  /** 正在执行的异步请求，包含已经取消但未执行完的请求 */
  private final Deque<AsyncCall> runningAsyncCalls = new ArrayDeque<>();

  /** 正在执行的同步请求，包含已经取消单未执行完的请求 */
  private final Deque<RealCall> runningSyncCalls = new ArrayDeque<>();
```

在OkHttp，使用如下构造了单例线程池

```
public synchronized ExecutorService executorService() {
    if (executorService == null) {
      executorService = new ThreadPoolExecutor(0, Integer.MAX_VALUE, 60, TimeUnit.SECONDS,
          new SynchronousQueue<Runnable>(), Util.threadFactory("OkHttp Dispatcher", false));
    }
    return executorService;
  }
```

构造一个线程池ExecutorService：

```
executorService = new ThreadPoolExecutor(
//corePoolSize 最小并发线程数,如果是0的话，空闲一段时间后所有线程将全部被销毁
    0,
//maximumPoolSize: 最大线程数，当任务进来时可以扩充的线程最大值，当大于了这个值就会根据丢弃处理机制来处理
    Integer.MAX_VALUE,
//keepAliveTime: 当线程数大于corePoolSize时，多余的空闲线程的最大存活时间
```

```
    60,
//单位秒
    TimeUnit.SECONDS,
//工作队列,先进先出
    new SynchronousQueue<Runnable>(),
//单个线程的工厂
   Util.threadFactory("OkHttp Dispatcher", false));
```

可以看出，在Okhttp中，构建了一个核心为[0, Integer.MAX_VALUE]的线程池，它不保留任何最小线程数，随时创建更多的线程数，当线程空闲时只能活60秒，它使用了一个不存储元素的阻塞工作队列，一个叫做"OkHttp Dispatcher"的线程工厂。

也就是说，在实际运行中，当收到10个并发请求时，线程池会创建十个线程，当工作完成后，线程池会在60s后相继关闭所有线程。

```
synchronized void enqueue(AsyncCall call) {
    if (runningAsyncCalls.size() < maxRequests && runningCallsForHost(call) < maxRequestsPerHost) {
      runningAsyncCalls.add(call);
      executorService().execute(call);
    } else {
      readyAsyncCalls.add(call);
    }
  }
```

从上述源码分析，如果当前还能执行一个并发请求，则加入 runningAsyncCalls ，立即执行，否则加入 readyAsyncCalls 队列。

Dispatcher线程池总结

1）调度线程池Disptcher实现了高并发，低阻塞的实现 2）采用Deque作为缓存，先进先出的顺序执行 3）任务在try/finally中调用了finished函数，控制任务队列的执行顺序，而不是采用锁，减少了编码复杂性提高性能

这里是分析OkHttp源码，并不详细讲线程池原理，如对线程池不了解请参考如下链接

点我，线程池原理，在文章性能优化最后有视频对线程池原理讲解

```
try {
        Response response = getResponseWithInterceptorChain();
        if (retryAndFollowUpInterceptor.isCanceled()) {
          signalledCallback = true;
          responseCallback.onFailure(RealCall.this, new IOException("Canceled"));
        } else {
          signalledCallback = true;
          responseCallback.onResponse(RealCall.this, response);
        }
      } finally {
        client.dispatcher().finished(this);
      }
```

当任务执行完成后，无论是否有异常，finally代码段总会被执行，也就是会调用Dispatcher的finished函数

```
void finished(AsyncCall call) {
   finished(runningAsyncCalls, call, true);
 }
```

从上面的代码可以看出，第一个参数传入的是正在运行的异步队列，第三个参数为true，下面再看有是三个参数的finished方法：

```
private <T> void finished(Deque<T> calls, T call, boolean promoteCalls) {
    int runningCallsCount;
    Runnable idleCallback;
    synchronized (this) {
      if (!calls.remove(call)) throw new AssertionError("Call wasn't in-flight!");
      if (promoteCalls) promoteCalls();
      runningCallsCount = runningCallsCount();
      idleCallback = this.idleCallback;
    }

    if (runningCallsCount == 0 && idleCallback != null) {
      idleCallback.run();
    }
  }
```

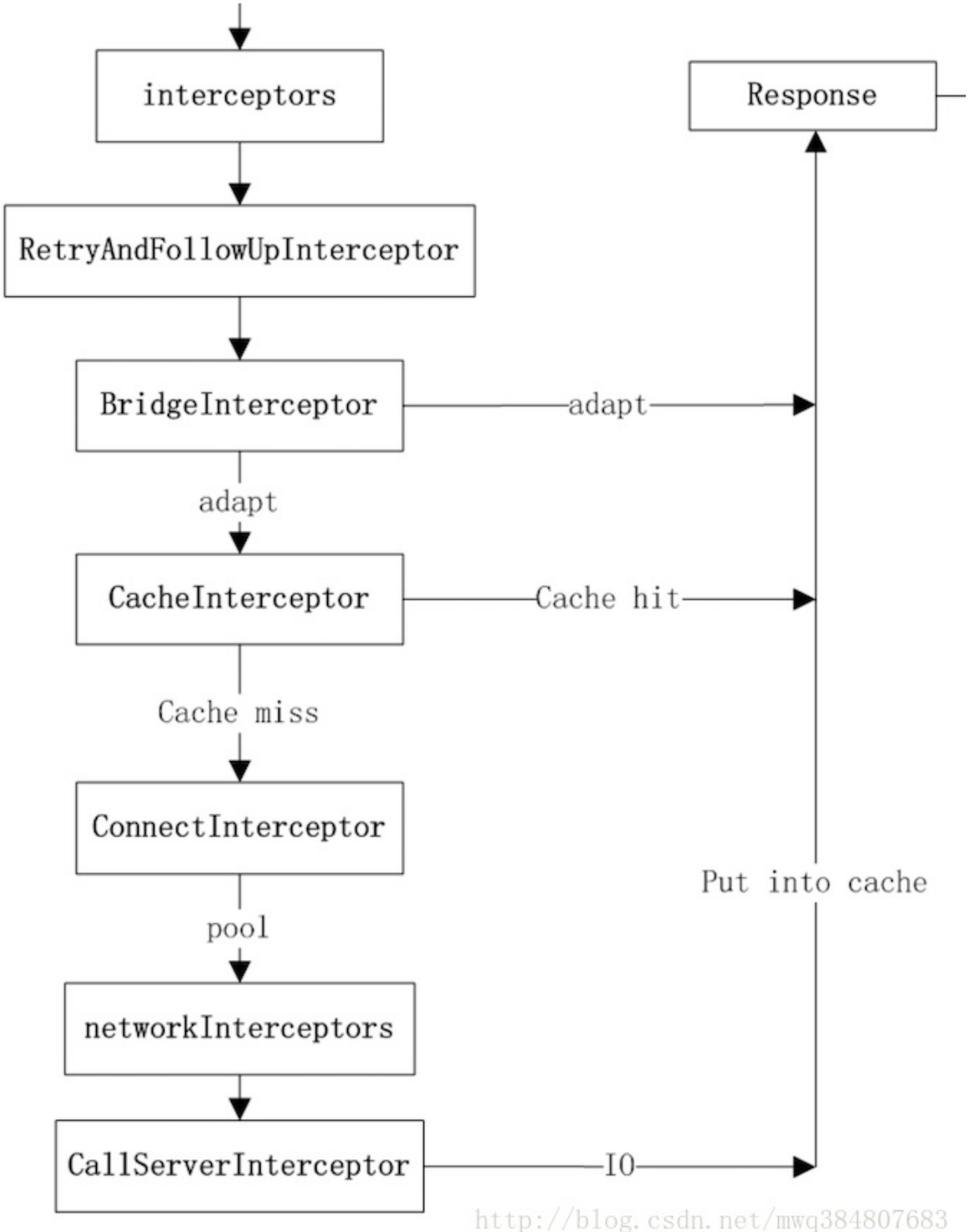
打开源码，发现它将正在运行的任务Call从队列runningAsyncCalls中移除后，获取运行数量判断是否进入了Idle状态，接着执行promoteCalls()函数,下面是promoteCalls()方法：

```
private void promoteCalls() {
    if (runningAsyncCalls.size() >= maxRequests) return; // Already running max capacity.
    if (readyAsyncCalls.isEmpty()) return; // No ready calls to promote.

    for (Iterator<AsyncCall> i = readyAsyncCalls.iterator(); i.hasNext(); ) {
      AsyncCall call = i.next();

      if (runningCallsForHost(call) < maxRequestsPerHost) {
        i.remove();
        runningAsyncCalls.add(call);
        executorService().execute(call);
      }

      if (runningAsyncCalls.size() >= maxRequests) return; // Reached max capacity.
    }
  }
```

主要就是遍历等待队列，并且需要满足同一主机的请求小于maxRequestsPerHost时，就移到运行队列中并交给线程池运行。就主动的把缓存队列向前走了一步，而没有使用互斥锁等复杂编码

核心重点getResponseWithInterceptorChain方法

```
Response getResponseWithInterceptorChain() throws IOException {
    // Build a full stack of interceptors.
    List<Interceptor> interceptors = new ArrayList<>();
    interceptors.addAll(client.interceptors());
    interceptors.add(retryAndFollowUpInterceptor);
    interceptors.add(new BridgeInterceptor(client.cookieJar()));
    interceptors.add(new CacheInterceptor(client.internalCache()));
    interceptors.add(new ConnectInterceptor(client));
    if (!forWebSocket) {
      interceptors.addAll(client.networkInterceptors());
    }
    interceptors.add(new CallServerInterceptor(forWebSocket));

    Interceptor.Chain chain = new RealInterceptorChain(
        interceptors, null, null, null, 0, originalRequest);
    return chain.proceed(originalRequest);
  }
```



1）在配置 OkHttpClient 时设置的 interceptors； 2）负责失败重试以及重定向的 RetryAndFollowUpInterceptor； 3）负责把用户构造的请求转换为发送到服务器的请求、把服务器返回的响应转换为用户友好的响应的 BridgeInterceptor； 4）负责读取缓存直接返回、更新缓存的 CacheInterceptor； 5）负责和服务器建立连接的 ConnectInterceptor； 6）配置 OkHttpClient 时设置的 networkInterceptors； 7）负责向服务器发送请求数据、从服务器读取响应数据的 CallServerInterceptor。

OkHttp的这种拦截器链采用的是责任链模式，这样的好处是将请求的发送和处理分开，并且可以动态添加中间的处理方实现对请求的处理、短路等操作。

从上述源码得知，不管okhttp有多少拦截器最后都会走，如下方法：

```
Interceptor.Chain chain = new RealInterceptorChain(
        interceptors, null, null, null, 0, originalRequest);
return chain.proceed(originalRequest);
```

从方法名字基本可以猜到是干嘛的，调用 chain.proceed(originalRequest); 将request传递进来，从拦截器链里拿到返回结果。那么拦截器Interceptor是干嘛的，Chain是干嘛的呢？继续往下看RealInterceptorChain

RealInterceptorChain类

下面是RealInterceptorChain的定义，该类实现了Chain接口，在getResponseWithInterceptorChain调用时好几个参数都传的null。

```
public final class RealInterceptorChain implements Interceptor.Chain {

   public RealInterceptorChain(List<Interceptor> interceptors, StreamAllocation streamAllocation,
        HttpCodec httpCodec, RealConnection connection, int index, Request request) {
        this.interceptors = interceptors;
        this.connection = connection;
        this.streamAllocation = streamAllocation;
        this.httpCodec = httpCodec;
        this.index = index;
        this.request = request;
  }
  ......

 @Override
 public Response proceed(Request request) throws IOException {
    return proceed(request, streamAllocation, httpCodec, connection);
  }

  public Response proceed(Request request, StreamAllocation streamAllocation, HttpCodec httpCodec,
      RealConnection connection) throws IOException {
    if (index >= interceptors.size()) throw new AssertionError();

    calls++;

    ......

    // Call the next interceptor in the chain.
    RealInterceptorChain next = new RealInterceptorChain(
        interceptors, streamAllocation, httpCodec, connection, index + 1, request);
    Interceptor interceptor = interceptors.get(index);
    Response response = interceptor.intercept(next);

   ......

    return response;
  }

  protected abstract void execute();
}
```

主要看proceed方法，proceed方法中判断index（此时为0）是否大于或者等于client.interceptors(List)的大小。由于httpStream为null，所以首先创建next拦截器链，主需要把索引置为index+1即可；然后获取第一个拦截器，调用其intercept方法。

Interceptor 代码如下：

```
public interface Interceptor {
  Response intercept(Chain chain) throws IOException;
```

```
  interface Chain {
    Request request();

    Response proceed(Request request) throws IOException;

    Connection connection();
  }
}
```

BridgeInterceptor

BridgeInterceptor从用户的请求构建网络请求，然后提交给网络，最后从网络响应中提取出用户响应。从最上面的图可以看出，BridgeInterceptor实现了适配的功能。下面是其intercept方法：

```
public final class BridgeInterceptor implements Interceptor {
  ......

@Override
public Response intercept(Chain chain) throws IOException {
  Request userRequest = chain.request();
  Request.Builder requestBuilder = userRequest.newBuilder();

 RequestBody body = userRequest.body();
 //如果存在请求主体部分，那么需要添加Content-Type、Content-Length首部
 if (body != null) {
      MediaType contentType = body.contentType();
      if (contentType != null) {
        requestBuilder.header("Content-Type", contentType.toString());
      }

      long contentLength = body.contentLength();
      if (contentLength != -1) {
        requestBuilder.header("Content-Length", Long.toString(contentLength));
        requestBuilder.removeHeader("Transfer-Encoding");
      } else {
        requestBuilder.header("Transfer-Encoding", "chunked");
        requestBuilder.removeHeader("Content-Length");
      }
    }

    if (userRequest.header("Host") == null) {
      requestBuilder.header("Host", hostHeader(userRequest.url(), false));
    }

    if (userRequest.header("Connection") == null) {
      requestBuilder.header("Connection", "Keep-Alive");
    }

    // If we add an "Accept-Encoding: gzip" header field we're responsible for also decompressing
    // the transfer stream.
    boolean transparentGzip = false;
    if (userRequest.header("Accept-Encoding") == null && userRequest.header("Range") == null) {
      transparentGzip = true;
      requestBuilder.header("Accept-Encoding", "gzip");
    }

    List<Cookie> cookies = cookieJar.loadForRequest(userRequest.url());
    if (!cookies.isEmpty()) {
      requestBuilder.header("Cookie", cookieHeader(cookies));
    }

  if (userRequest.header("User-Agent") == null) {
      requestBuilder.header("User-Agent", Version.userAgent());
  }
```

```
Response networkResponse = chain.proceed(requestBuilder.build());

HttpHeaders.receiveHeaders(cookieJar, userRequest.url(), networkResponse.headers());

Response.Builder responseBuilder = networkResponse.newBuilder()
        .request(userRequest);

    if (transparentGzip
        && "gzip".equalsIgnoreCase(networkResponse.header("Content-Encoding"))
        && HttpHeaders.hasBody(networkResponse)) {
      GzipSource responseBody = new GzipSource(networkResponse.body().source());
      Headers strippedHeaders = networkResponse.headers().newBuilder()
          .removeAll("Content-Encoding")
          .removeAll("Content-Length")
          .build();
      responseBuilder.headers(strippedHeaders);
      responseBuilder.body(new RealResponseBody(strippedHeaders, Okio.buffer(responseBody)));
    }

    return responseBuilder.build();
  }

  /** Returns a 'Cookie' HTTP request header with all cookies, like {@code a=b; c=d}. */
  private String cookieHeader(List<Cookie> cookies) {
    StringBuilder cookieHeader = new StringBuilder();
    for (int i = 0, size = cookies.size(); i < size; i++) {
      if (i > 0) {
        cookieHeader.append("; ");
      }
      Cookie cookie = cookies.get(i);
      cookieHeader.append(cookie.name()).append('=').append(cookie.value());
    }
    return cookieHeader.toString();
  }
}
```

从上面的代码可以看出，首先获取原请求，然后在请求中添加头，比如Host、Connection、Accept-Encoding参数等，然后根据看是否需要填充Cookie，在对原始请求做出处理后，使用chain的procced方法得到响应，接下来对响应做处理得到用户响应，最后返回响应。接下来再看下一个拦截器ConnectInterceptor的处理。

```
public final class ConnectInterceptor implements Interceptor {
  ......

 @Override
 public Response intercept(Chain chain) throws IOException {
 RealInterceptorChain realChain = (RealInterceptorChain) chain;
Request request = realChain.request();
StreamAllocation streamAllocation = realChain.streamAllocation();

 // We need the network to satisfy this request. Possibly for validating a conditional GET.
 boolean doExtensiveHealthChecks = !request.method().equals("GET");
 HttpCodec httpCodec = streamAllocation.newStream(client, doExtensiveHealthChecks);
 RealConnection connection = streamAllocation.connection();

 return realChain.proceed(request, streamAllocation, httpCodec, connection);
  }
}
```

实际上建立连接就是创建了一个 HttpCodec 对象，它利用 Okio 对 Socket 的读写操作进行封装，Okio 以后有机会再进行分析，现在让我们对它们保持一个简单地认识：它对 java.io 和 java.nio 进行了封装，让我们更便捷高效的进行 IO 操作。

CallServerInterceptor

CallServerInterceptor是拦截器链中最后一个拦截器，负责将网络请求提交给服务器。它的intercept方法实现如下：

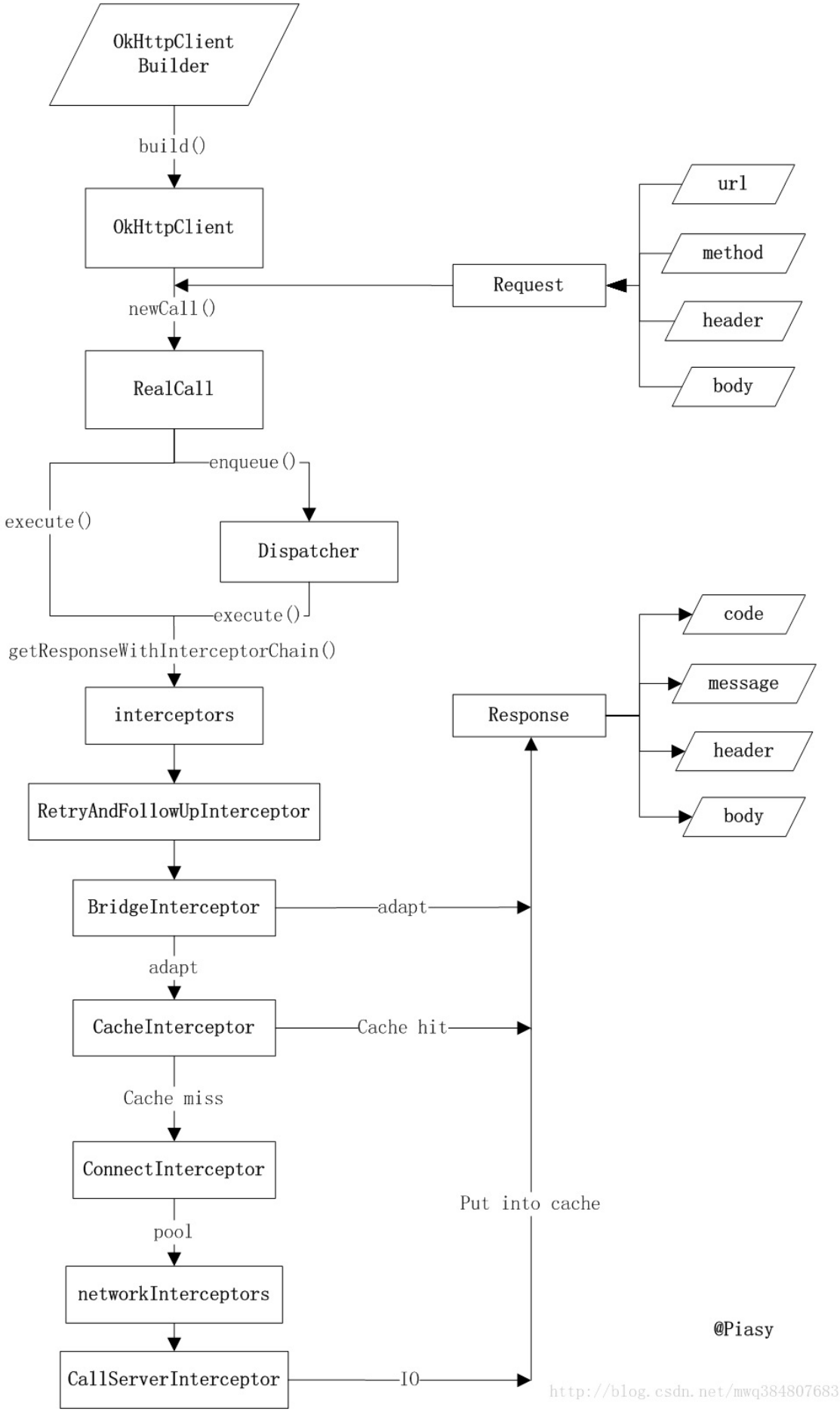
```
@Override
public Response intercept(Chain chain) throws IOException {
    RealInterceptorChain realChain = (RealInterceptorChain) chain;
    HttpCodec httpCodec = realChain.httpStream();
    StreamAllocation streamAllocation = realChain.streamAllocation();
    RealConnection connection = (RealConnection) realChain.connection();
    Request request = realChain.request();

    long sentRequestMillis = System.currentTimeMillis();
    httpCodec.writeRequestHeaders(request);

    Response.Builder responseBuilder = null;
    if (HttpMethod.permitsRequestBody(request.method()) && request.body() != null) {
      // If there's a "Expect: 100-continue" header on the request, wait for a "HTTP/1.1 100
      // Continue" response before transmitting the request body. If we don't get that, return what
      // we did get (such as a 4xx response) without ever transmitting the request body.
      if ("100-continue".equalsIgnoreCase(request.header("Expect"))) {
        httpCodec.flushRequest();
        responseBuilder = httpCodec.readResponseHeaders(true);
      }

      if (responseBuilder == null) {
        // Write the request body if the "Expect: 100-continue" expectation was met.
        Sink requestBodyOut = httpCodec.createRequestBody(request, request.body().contentLength());
        BufferedSink bufferedRequestBody = Okio.buffer(requestBodyOut);
        request.body().writeTo(bufferedRequestBody);
        bufferedRequestBody.close();
      } else if (!connection.isMultiplexed()) {
        // If the "Expect: 100-continue" expectation wasn't met, prevent the HTTP/1 connection from
        // being reused. Otherwise we're still obligated to transmit the request body to leave the
        // connection in a consistent state.
        streamAllocation.noNewStreams();
      }
    }

    httpCodec.finishRequest();

    if (responseBuilder == null) {
      responseBuilder = httpCodec.readResponseHeaders(false);
    }

    Response response = responseBuilder
        .request(request)
        .handshake(streamAllocation.connection().handshake())
        .sentRequestAtMillis(sentRequestMillis)
        .receivedResponseAtMillis(System.currentTimeMillis())
        .build();

    int code = response.code();
    if (forWebSocket && code == 101) {
      // Connection is upgrading, but we need to ensure interceptors see a non-null response body.
      response = response.newBuilder()
          .body(Util.EMPTY_RESPONSE)
          .build();
    } else {
      response = response.newBuilder()
          .body(httpCodec.openResponseBody(response))
          .build();
    }

    if ("close".equalsIgnoreCase(response.request().header("Connection"))
```

```
        || "close".equalsIgnoreCase(response.header("Connection"))) {
      streamAllocation.noNewStreams();
    }

    if ((code == 204 || code == 205) && response.body().contentLength() > 0) {
      throw new ProtocolException(
          "HTTP " + code + " had non-zero Content-Length: " + response.body().contentLength());
    }

    return response;
  }
```

从上面的代码中可以看出，首先获取HttpStream对象，然后调用writeRequestHeaders方法写入请求的头部，然后判断是否需要写入请求的body部分，最后调用finishRequest()方法将所有数据刷新给底层的Socket，接下来尝试调用readResponseHeaders()方法读取响应的头部，然后再调用openResponseBody()方法得到响应的body部分，最后返回响应。

最后总结

OkHttp的底层是通过Java的Socket发送HTTP请求与接受响应的(这也好理解，HTTP就是基于TCP协议的)，但是OkHttp实现了连接池的概念，即对于同一主机的多个请求，其实可以公用一个Socket连接，而不是每次发送完HTTP请求就关闭底层的Socket，这样就实现了连接池的概念。而OkHttp对Socket的读写操作使用的Oklo库进行了一层封装。



- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析

与XMPP相关面试题

- 与XMPP相关试题一
- 与XMPP相关试题二

与性能优化相关面试题

- 与性能优化相关面试题一
- 与性能优化相关面试题二
- 与性能优化相关面试题三
- 与性能优化相关面试题四
- 与性能优化相关面试题五
- 与性能优化相关面试题六
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析

与人事相关面试题

- 人事面试宝典

本文配套视频

- 静默安装原理一
- 静默安装原理二

很多哥们在后台公众号留言，让讲讲如何编译源码，感觉源码开发很神秘，好吧，今天就手把手教大家安装虚拟机编译源码，文字部分的东西就别看了，毕竟源码或者技术这种东西三言两语也讲不明白，直接看视频吧，通过本文我录制的视频希望对大家能起到抛砖引玉的作用，不过这个视频是我在三年前录制的，讲的是通过源码编译不需要root权限实现静默安装和偷拍，然后也把Android系统安装源码给大家分析了，毕竟三年前随堂录制的视频，也比较简单的东西，如果没有收获随便看看就行，有收获点个赞就行，对静默安装感兴趣就从第一个视频开始看起，如果只是单纯对源码编译感兴趣，直接从第二个视频15分钟开始看就行，这样能节约大家的时间，视频时间有点长。

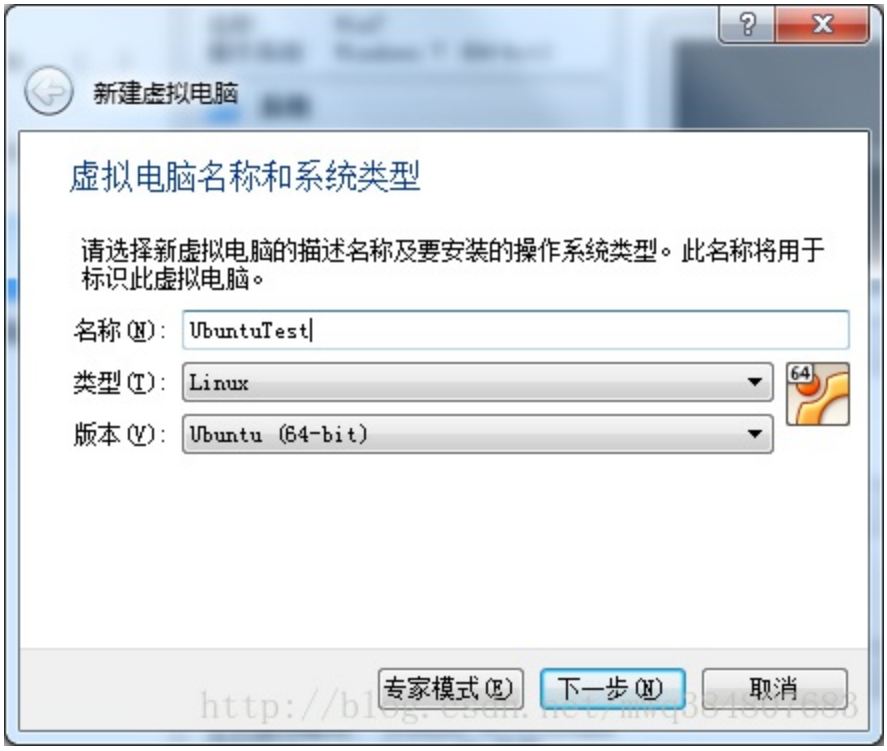
四年前写的静默安装代码，有兴趣看看吧：http://www.apkbus.com/forum.php?mod=viewthread&tid=120895

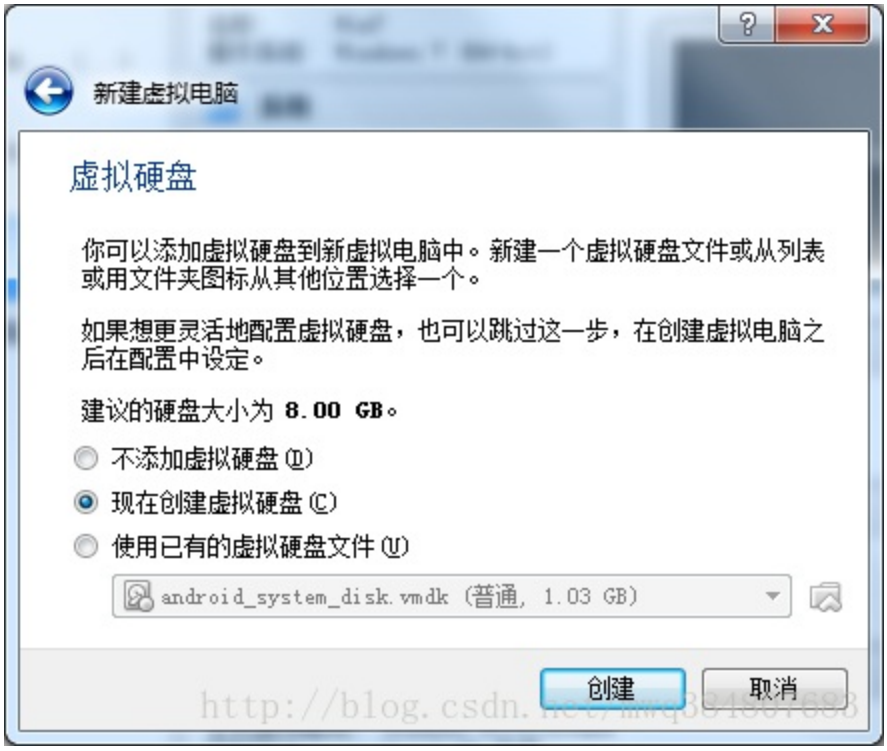
Android源码分析

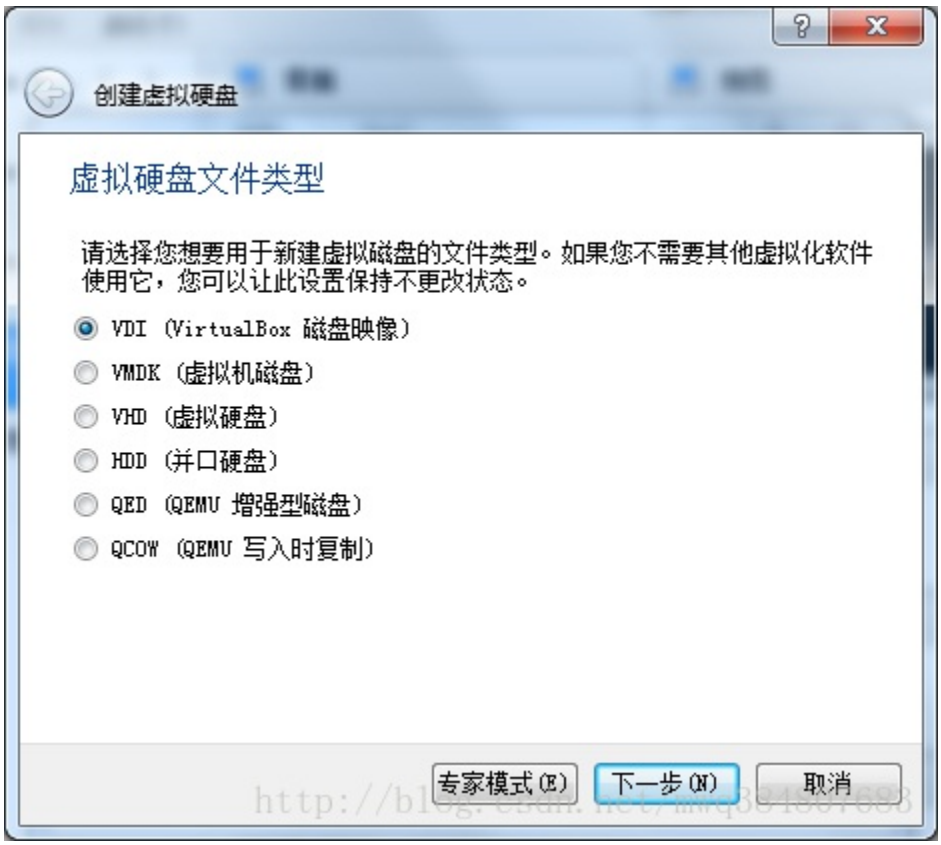
1. 了解虚拟机的的使用

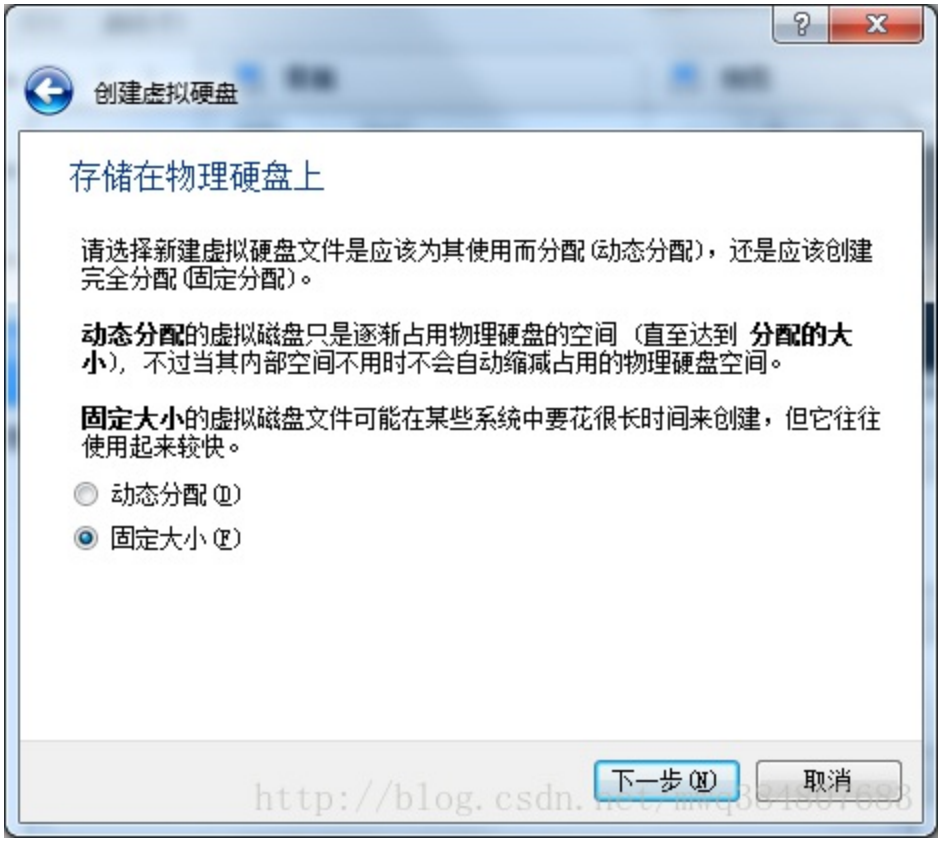
Android 系统的编译环境目前只支持 Linux 以及 Mac OS 两种操作系统 虚拟机可以很方便的搭建不同系统的开发环境 虚拟机可以完整的模拟一台电脑 虚拟机中的系统独立于主系统存在，子系统的损坏与否不影响主系统功能

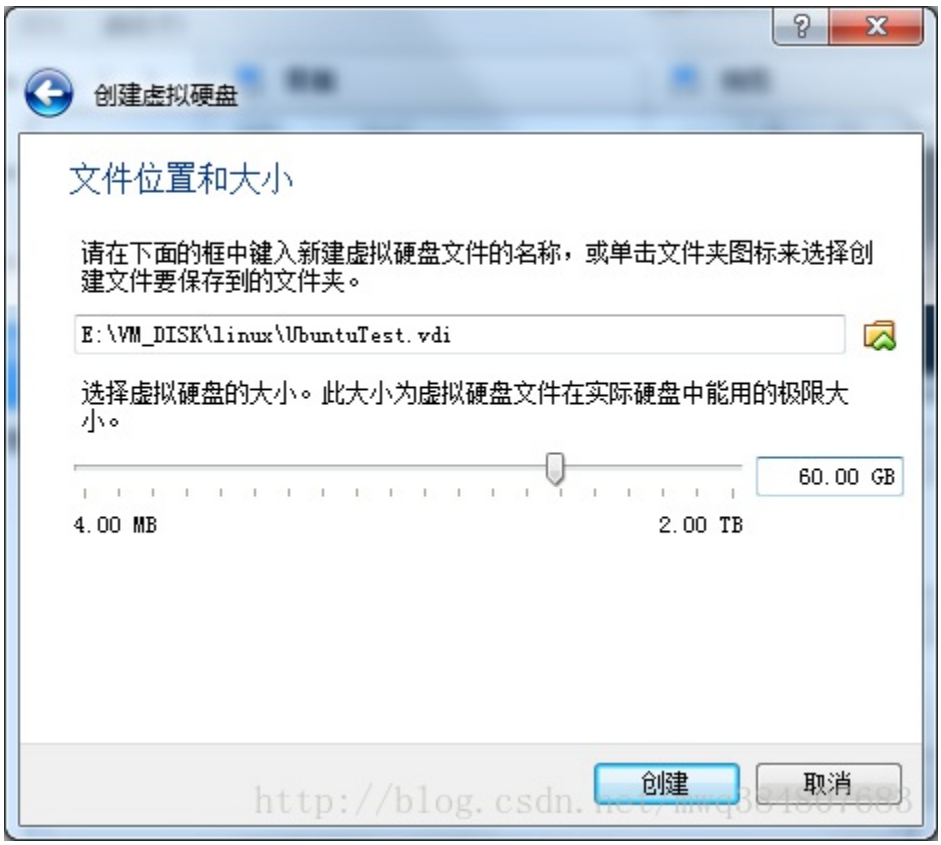
1.1. 了解虚拟机的安装和创建新虚拟机的方法

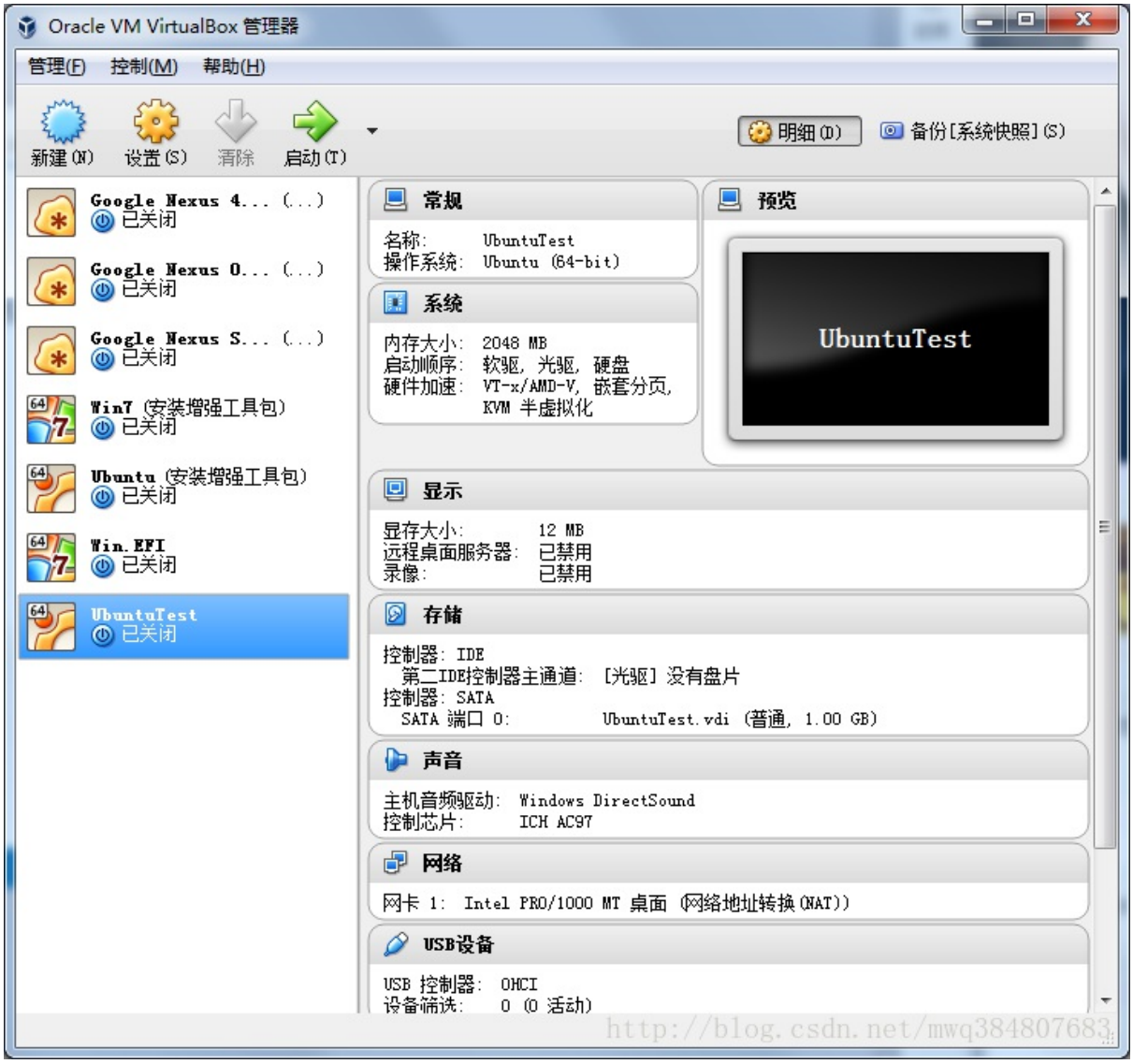


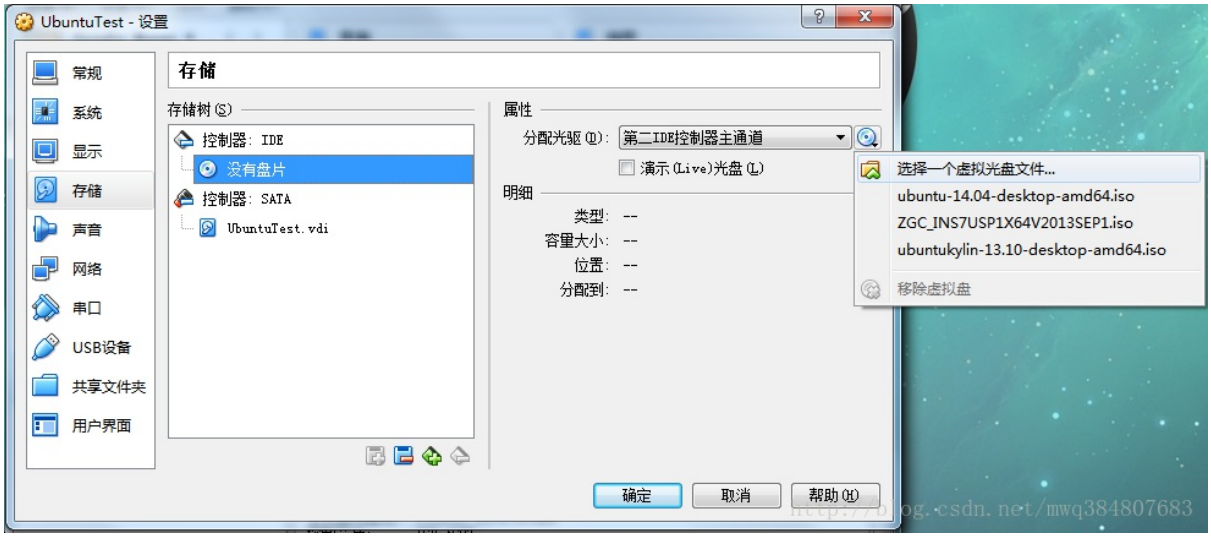

新建虚拟电脑
虚拟硬盘
你可以添加虚拟硬盘到新虚拟电脑中。新建一个虚拟硬盘文件或从列表或用文件夹图标从其他位置选择一个。
如果想更灵活地配置虚拟硬盘，也可以跳过这一步，在创建虚拟电脑之后在配置中设定。
建议的硬盘大小为 8.00 GB。
不添加虚拟硬盘(D)
现在创建虚拟硬盘(C)
使用已有的虚拟硬盘文件(U)
android_system_disk.vmdk (普通, 1.03 GB)
创建
取消

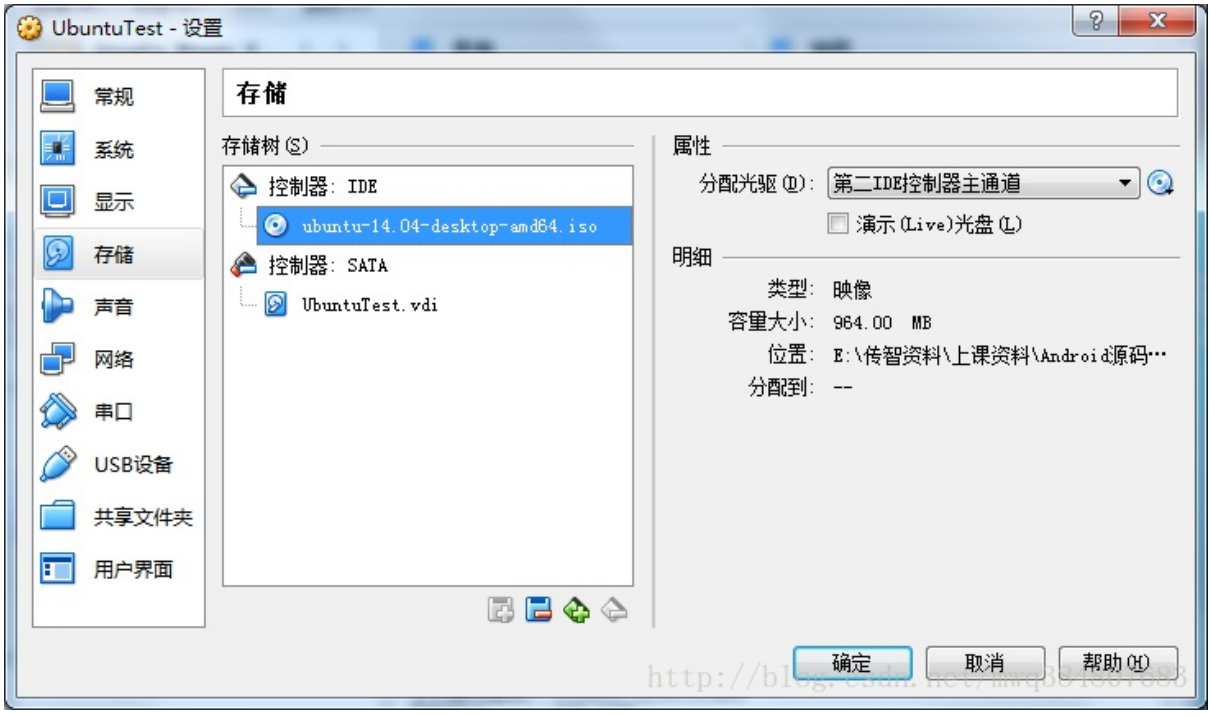

创建虚拟硬盘
虚拟硬盘文件类型
请选择您想要用于新建虚拟磁盘的文件类型。如果您不需要其他虚拟化软件使用它，您可以让此设置保持不更改状态。
VDI (VirtualBox 磁盘映像)
VMDK (虚拟机磁盘)
VHD (虚拟硬盘)
HDD (并口硬盘)
QED (QEMU 增强型磁盘)
QCOW (QEMU 写入时复制)
专家模式(E)
下一步(N)
取消


创建虚拟硬盘
存储在物理硬盘上
请选择新建虚拟硬盘文件是应该为其使用而分配(动态分配)，还是应该创建完全分配(固定分配)。
动态分配的虚拟磁盘只是逐渐占用物理硬盘的空间（直至达到 分配的大小），不过当其内部空间不用时不会自动缩减占用的物理硬盘空间。
固定大小的虚拟磁盘文件可能在某些系统中要花很长时间来创建，但它往往使用起来较快。
动态分配(D)
固定大小(F)
下一步(N)
取消

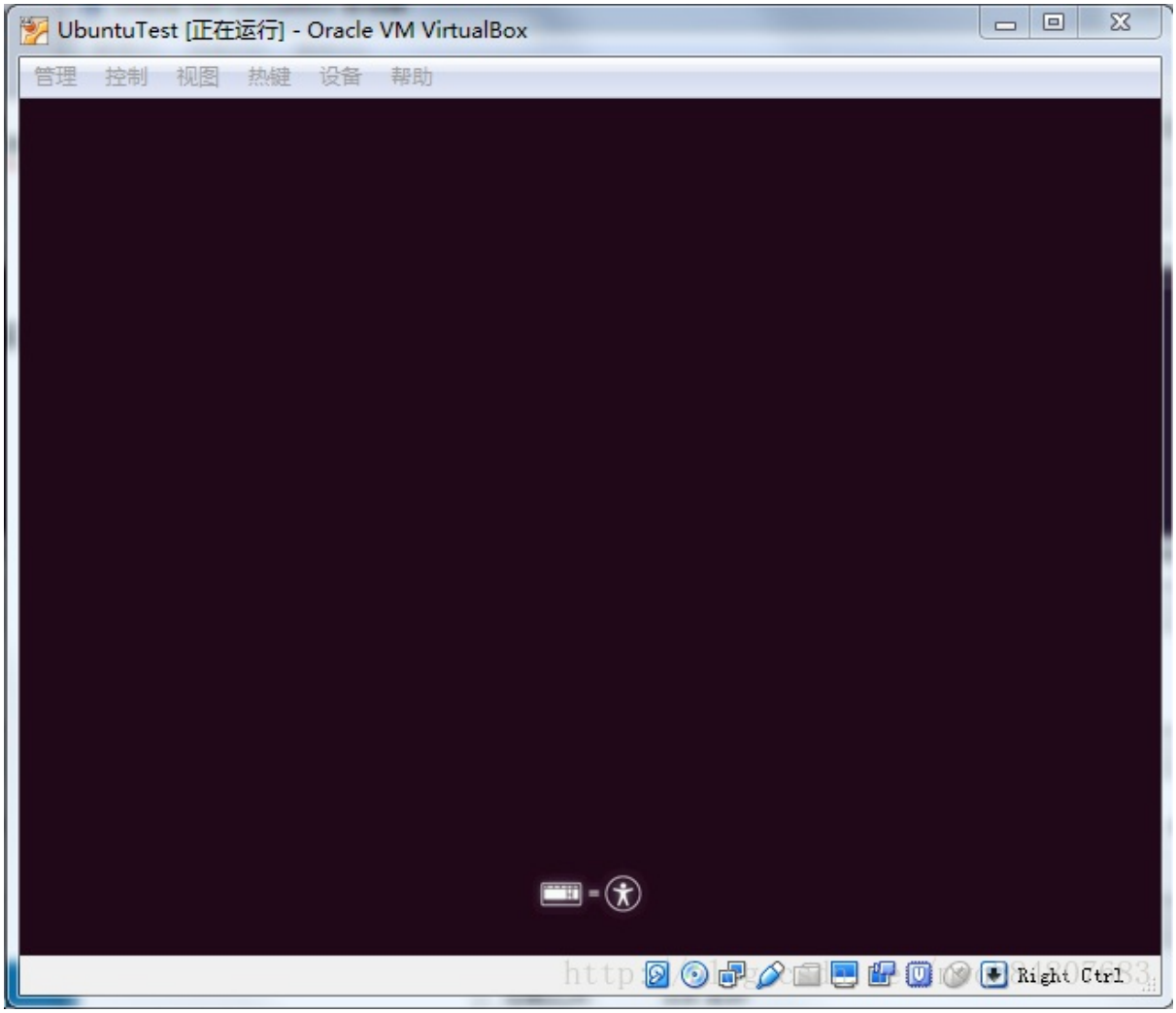

创建虚拟硬盘
文件位置和大小
请在下面的框中键入新建虚拟硬盘文件的名称，或单击文件夹图标来选择创建文件要保存到的文件夹。
E:\VM_DISK\linux\UbuntuTest.vdi
选择虚拟硬盘的大小。此大小为虚拟硬盘文件在实际硬盘中能用的极限大小。
60.00 GB
4.00 MB
2.00 TB
创建
取消


Oracle VM VirtualBox 管理器
管理(F) 控制(M) 帮助(H)
新建(N) 设置(S) 清除 启动(T)
明细(D) 备份[系统快照](S)
Google Nexus 4... (...) 已关闭
Google Nexus O... (...) 已关闭
Google Nexus S... (...) 已关闭
Win7 (安装增强工具包) 已关闭
Ubuntu (安装增强工具包) 已关闭
Win.EFI 已关闭
UbuntuTest 已关闭
常规
名称: UbuntuTest
操作系统: Ubuntu (64-bit)
系统
内存大小: 2048 MB
启动顺序: 软驱, 光驱, 硬盘
硬件加速: VT-x/AMD-V, 嵌套分页, KVM 半虚拟化
预览
UbuntuTest
显示
显存大小: 12 MB
远程桌面服务器: 已禁用
录像: 已禁用
存储
控制器: IDE
第二IDE控制器主通道: [光驱] 没有盘片
控制器: SATA
SATA 端口 0: UbuntuTest.vdi (普通, 1.00 GB)
声音
主机音频驱动: Windows DirectSound
控制芯片: ICH AC97
网络
网卡 1: Intel PRO/1000 MT 桌面 (网络地址转换(NAT))
USB设备
USB 控制器: OHCI
设备筛选: 0 (0 活动)
http://blog.csdn.net/mwq384807683


UbuntuTest - 设置
常规
系统
显示
存储
声音
网络
串口
USB设备
共享文件夹
用户界面
存储
存储树(S)
控制器: IDE
没有盘片
控制器: SATA
UbuntuTest.vdi
属性
分配光驱(D): 第二IDE控制器主通道
演示(Live)光盘(L)
明细
类型: --
容量大小: --
位置: --
分配到: --
选择一个虚拟光盘文件...
ubuntu-14.04-desktop-amd64.iso
ZGC_INS7USP1X64V2013SEP1.iso
ubuntukylin-13.10-desktop-amd64.iso
移除虚拟盘
确定
取消
帮助(H)



1.2. 了解虚拟机里安装Ubuntu系统的方法




安装
欢迎
ગુજરાતી
தமிழ்
తెలుగు
ಕನ್ನಡ
മലയാളം
සිංහල
ภาษาไทย
ລາວ
བོད་ཡིག
မြန်မာစာ
한국어
日本語
中文(繁體)
中文(简体)
试用 UbuntuKylin
安装 UbuntuKylin
您可以直接从此 CD 尝试 UbuntuKylin，而不用对您的电脑作任何更改。
如果您已经准备完毕，您可以与现有系统并存 (或者替代) 方式将 UbuntuKylin 安装到您的电脑上。此过程无需耗时太久。
http://blog.csdn.net/mwq384807683


安装
准备安装 UbuntuKylin
要获得最佳的体验，请确定这台计算机：
有至少 6.2 GB 可用的磁盘空间
已经插入电源
已经连接到互联网
安装中下载更新
UbuntuKylin 使用第三方软件播放 Flash、MP3 和其他媒体，并驱动某些图形和无线网络硬件。其中某些软件是闭源的。这些软件受到其文档中协议条款的约束。
安装这个第三方软件
Fluendo MP3 插件包含 Fraunhofer IIS 和 Technicolor SA 授权的 MPEG Layer-3 音频解码技术。
退出(Q)
后退(B)
继续
http://blog.csdn.net/mwq384807683


安装
安装类型
这台计算机似乎没有安装操作系统。您准备怎么做？
清除整个磁盘并安装 UbuntuKylin
注意：这会删除磁盘上的全部文件。
为了安全起见，将会加密你的新 Ubuntu 安装程序
下一步，你需要选择一个安全密钥。
给新安装的 Ubuntu 使用 LVM
这将启动逻辑分区管理(LVM)，有快照和调整分区大小等功能。
其他选项
您可以自己创建、调整分区，或者为 UbuntuKylin 选择多个分区。
退出(Q)
后退(B)
现在安装(I)
http://blog.csdn.net/mwq384807683


安装
您在什么地方？
Shanghai
后退(B)
继续
http://blog.csdn.net/mwq384807683


安装
键盘布局
选择您的键盘布局：
菲律宾语
芬兰语
哈萨克语
汉语
荷兰语
黑山语
加泰罗尼亚语
汉语
汉语 - Tibetan
汉语 - Tibetan (with ASCII numerals)
汉语 - Uyghur
在这里输入以测试您的键盘
探测键盘布局
后退(B)
继续
http://blog.csdn.net/mwq384807683


安装
您是谁？
您的姓名：
dj
您的计算机名：
dj-VirtualBox
与其他计算机联络时使用的名称。
选择一个用户名：
dj
选择一个密码：
密码强度：较弱
确认您的密码：
自动登录
登录时需要密码
加密我的主目录
后退(B)
继续
http://blog.csdn.net/mwq384807683

Android源码下载

> Android源码下载支持的系统目前只有Ubuntu和Mac OS两种操作系统, 本次以Ubuntu系统为例.
>
> 官方网站: https://source.android.com/source/downloading.html

1. 下载Git(版本控制工具). 调出命令行: ctrl + alt + T

```
sudo apt-get install git
```

1. 安装curl(上传和下载数据的工具).

```
sudo apt-get install curl
```

1. 安装repo(一个基于git的版本库管理工具, 这里用于自动批量下载android整个项目.).

```
// 创建目录
mkdir bin

// 下载repo脚本到本地bin文件夹下
curl https://android.git.kernel.org/repo >~/bin/repo
// 如果上面下载失败, 采用下面这种方式
curl "https://php.webtutor.pl/en/wp-content/uploads/2011/09/repo" >~/bin/repo

// 给所有用户追加可执行的权限
chmod a+x ~/bin/repo

// 临时把repo添加到环境变量中, 方便后面执行.
// 注意: 每次重启ubuntu之后此环境变量失效, 重新配置就可以了.
export PATH=~/bin:$PATH
```

1. 创建文件夹, 用于存放下载的Android源码.

```
// 创建目录
mkdir android_source

// 修改权限
chmod 777 android_source

cd android_source
```

1. 初始化库.

```
// 需要先配置git的用户信息
git config --global user.email "ad_307@yeah.net"
git config --global user.name "liyindong"

repo init -u httpss://android.googlesource.com/platform/manifest -b android-2.3_r1

// 如果上面初始化失败, 用下面的代码
repo init -u git://codeaurora.org/platform/manifest.git -b gingerbread
// 或
repo init -u git://android.git.kernel.org/platform/manifest.git -b  gingerbread
```

出现以下信息成功初始化

```
repo initialized in /home/liyindong/android_source
```

1. 开始同步下载.

```
repo sync
```

注意: 下载过程中, 因为网络问题, 可能会中断下载. 当中断下载时, 继续使用repo sync命令继续下载.

Android源码编译

在编译源码之前需要做一些准备操作, 详细步骤如下:

1. 安装JDK, google官方要求编译2.3源码需要JDK1.6.

- 1). 下载JDK1.6, 下载地址:https://download.oracle.com/otn/java/jdk/6u45-b06/jdk-6u45-linux-x64.bin
- 2). 创建目录.

```
sudo mkdir /usr/java
```

- 3). 把下载好的jdk-6u45-linux-x64.bin拷贝到上面创建的目录下.

```
sudo cp /home/liyindong/jdk-6u45-linux-x64.bin /usr/java
```

- 4). 添加可执行权限.

```
sudo chmod 755 /usr/java/jdk-6u45-linux-x64.bin
```

- 5). 解压.

```
cd /usr/java
sudo ./jdk-6u45-linux-x64.bin
```

- 6). 配置环境变量.

```
export JAVA_HOME=/usr/java/jdk1.6.0_45
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

- 7). 验证是否成功.

```
liyindong@liyindong-VirtualBox:~$ java -version
java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)
```

2. 安装其他编译时依赖的软件.

> **注意: ubuntu自带的源中速度比较慢, 有些软件找不到, 所以需要修改为国内的源, 修改源步骤如下:**

- 1). 备份ubuntu自带的源.

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.old
```

- 2). 修改源文件.

```
sudo gedit /etc/apt/sources.list
```

- 3). 这时会弹出一个文本编辑框, 先删除所有内容, 然后把以下内容拷贝进去, 并保存.

```
deb https://mirrors.163.com/ubuntu/ trusty main restricted universe multiverse
deb https://mirrors.163.com/ubuntu/ trusty-security main restricted universe multiverse
deb https://mirrors.163.com/ubuntu/ trusty-updates main restricted universe multiverse
deb https://mirrors.163.com/ubuntu/ trusty-proposed main restricted universe multiverse
deb https://mirrors.163.com/ubuntu/ trusty-backports main restricted universe multiverse
deb-src https://mirrors.163.com/ubuntu/ trusty main restricted universe multiverse
deb-src https://mirrors.163.com/ubuntu/ trusty-security main restricted universe multiverse
deb-src https://mirrors.163.com/ubuntu/ trusty-updates main restricted universe multiverse
deb-src https://mirrors.163.com/ubuntu/ trusty-proposed main restricted universe multiverse
deb-src https://mirrors.163.com/ubuntu/ trusty-backports main restricted universe multiverse

deb https://mirrors.sohu.com/ubuntu/ trusty main restricted universe multiverse
deb https://mirrors.sohu.com/ubuntu/ trusty-security main restricted universe multiverse
deb https://mirrors.sohu.com/ubuntu/ trusty-updates main restricted universe multiverse
deb https://mirrors.sohu.com/ubuntu/ trusty-proposed main restricted universe multiverse
deb https://mirrors.sohu.com/ubuntu/ trusty-backports main restricted universe multiverse
deb-src https://mirrors.sohu.com/ubuntu/ trusty main restricted universe multiverse
deb-src https://mirrors.sohu.com/ubuntu/ trusty-security main restricted universe multiverse
deb-src https://mirrors.sohu.com/ubuntu/ trusty-updates main restricted universe multiverse
deb-src https://mirrors.sohu.com/ubuntu/ trusty-proposed main restricted universe multiverse
deb-src https://mirrors.sohu.com/ubuntu/ trusty-backports main restricted universe multiverse

deb https://mirrors.oschina.net/ubuntu/ trusty main restricted universe multiverse
deb https://mirrors.oschina.net/ubuntu/ trusty-backports main restricted universe multiverse
deb https://mirrors.oschina.net/ubuntu/ trusty-proposed main restricted universe multiverse
deb https://mirrors.oschina.net/ubuntu/ trusty-security main restricted universe multiverse
deb https://mirrors.oschina.net/ubuntu/ trusty-updates main restricted universe multiverse
deb-src https://mirrors.oschina.net/ubuntu/ trusty main restricted universe multiverse
deb-src https://mirrors.oschina.net/ubuntu/ trusty-backports main restricted universe multiverse
deb-src https://mirrors.oschina.net/ubuntu/ trusty-proposed main restricted universe multiverse
deb-src https://mirrors.oschina.net/ubuntu/ trusty-security main restricted universe multiverse
deb-src https://mirrors.oschina.net/ubuntu/ trusty-updates main restricted universe multiverse
```

- 4). 保存之后, 更新数据源.

```
sudo apt-get update
```

- **执行完上面几步, 数据源就更新完成了, 下面就开始安装编译依赖的软件, 同样, 在终端中以行为单位依次输入以下命令:**

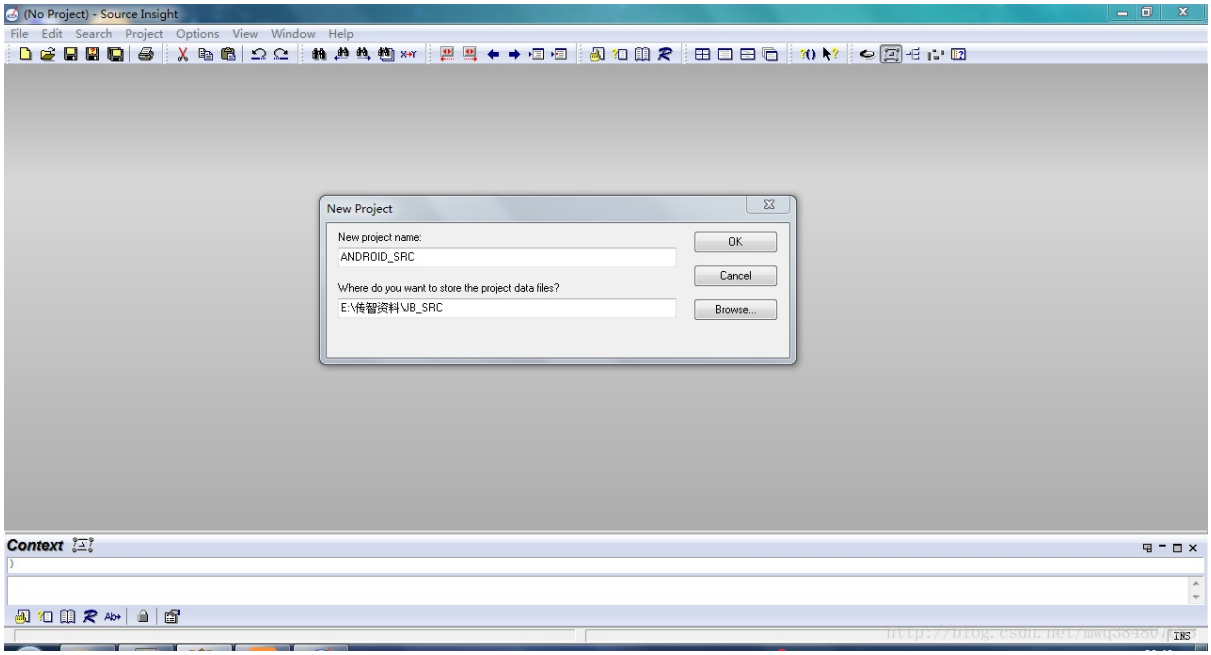
```
sudo apt-get install gnupg
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install gperf
```

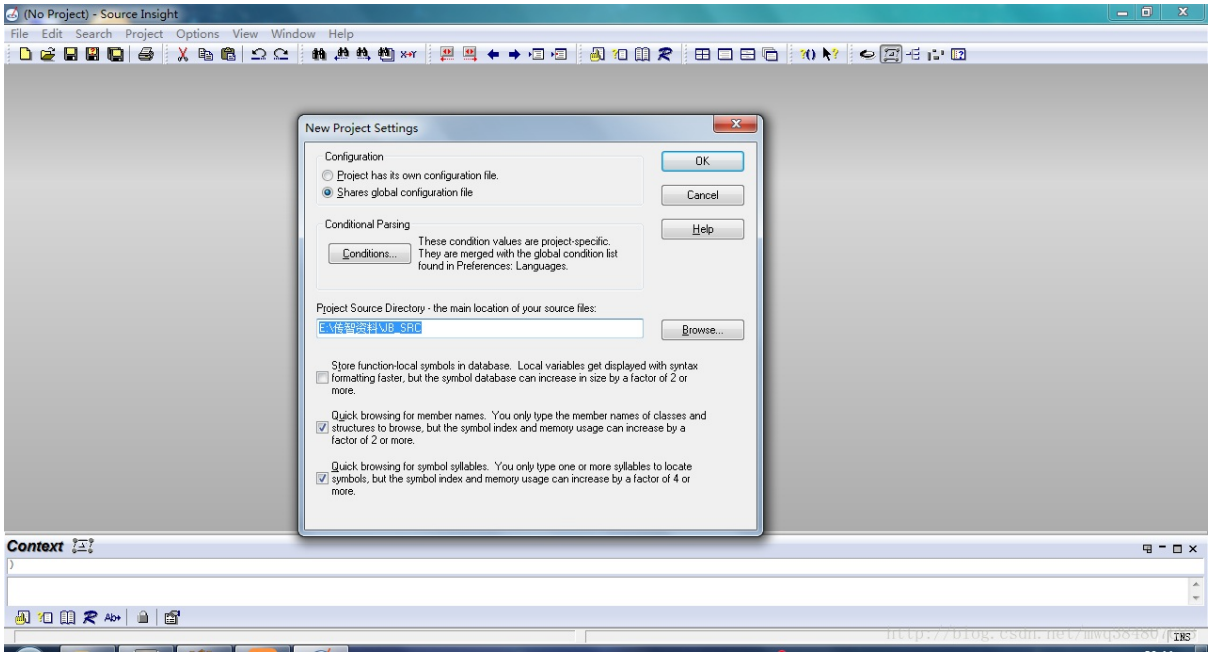
```
sudo apt-get install zip
sudo apt-get install curl
sudo apt-get install build-essential
sudo apt-get install libesd0-dev
sudo apt-get install libwxgtk2.6-dev
sudo apt-get install libsdl-dev
sudo apt-get install lsb-core
sudo apt-get install lib32readline-gplv2-dev
sudo apt-get install g++-multilib
sudo apt-get install lib32z1-dev
sudo apt-get install libswitch-perl
```

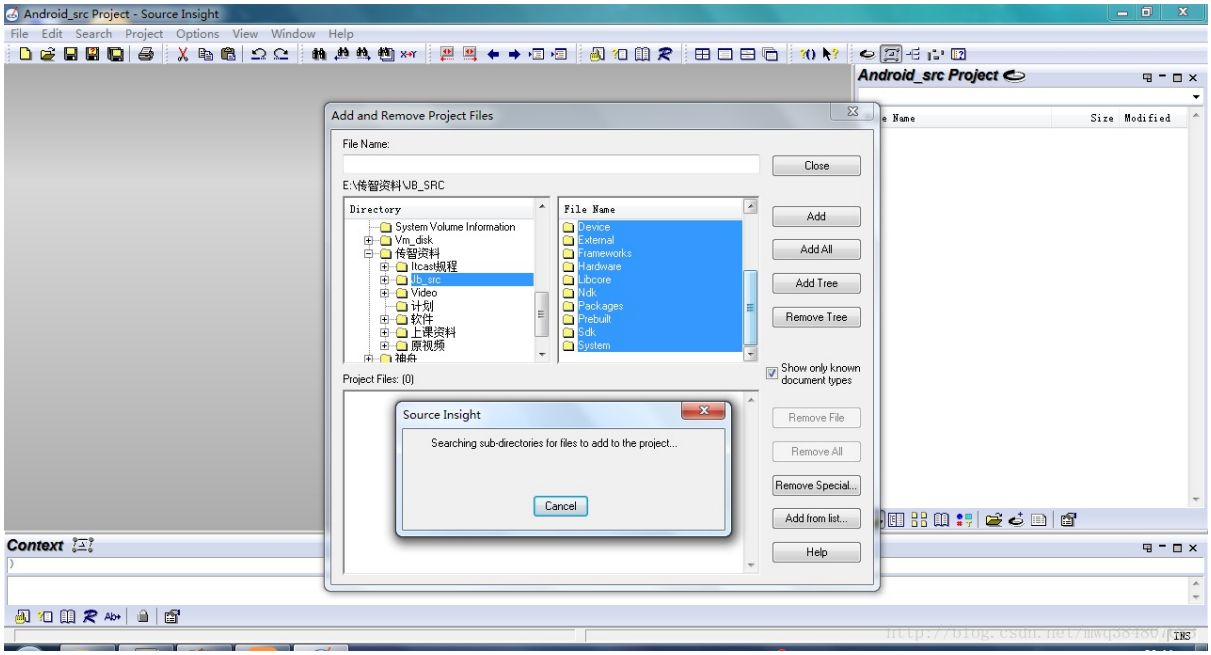
> 3. 开始编译, 在源码的目录下, 执行一下命令:

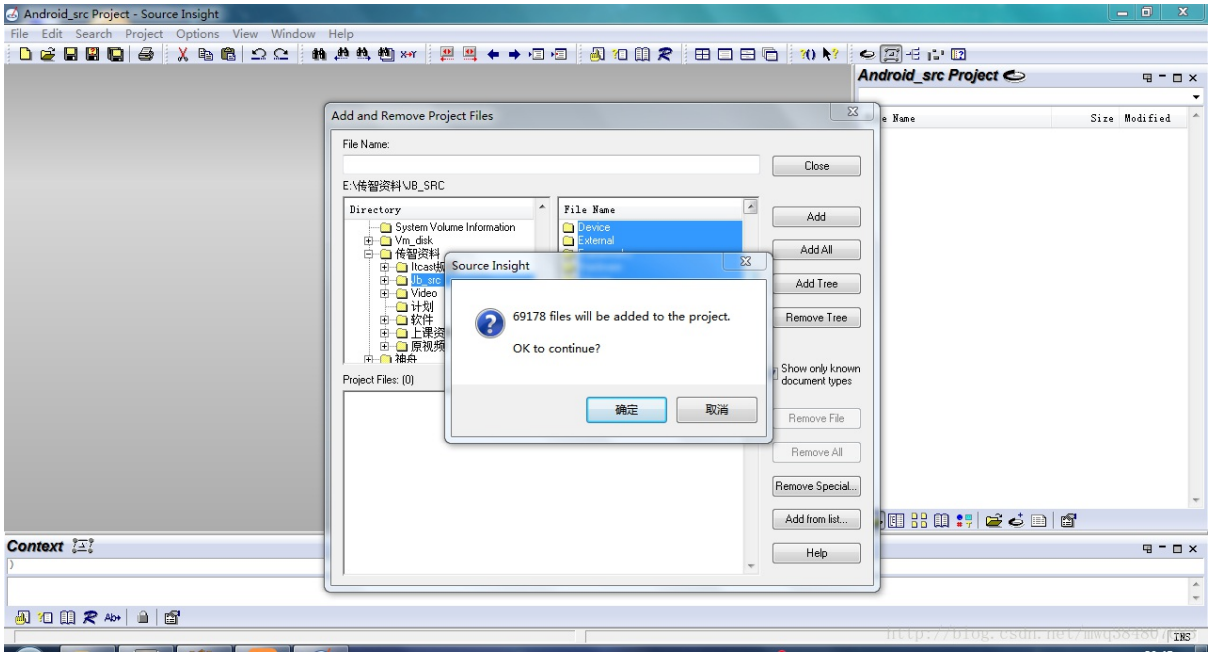
```
make
```

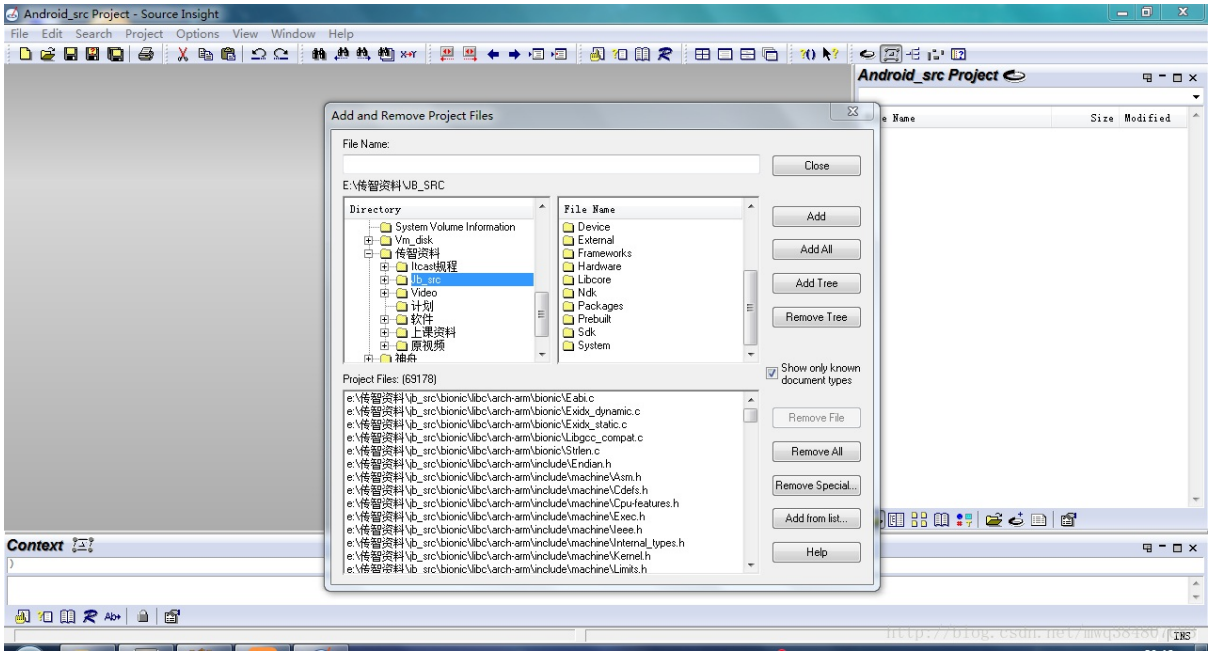
5. 了解SourceInsight的使用方法

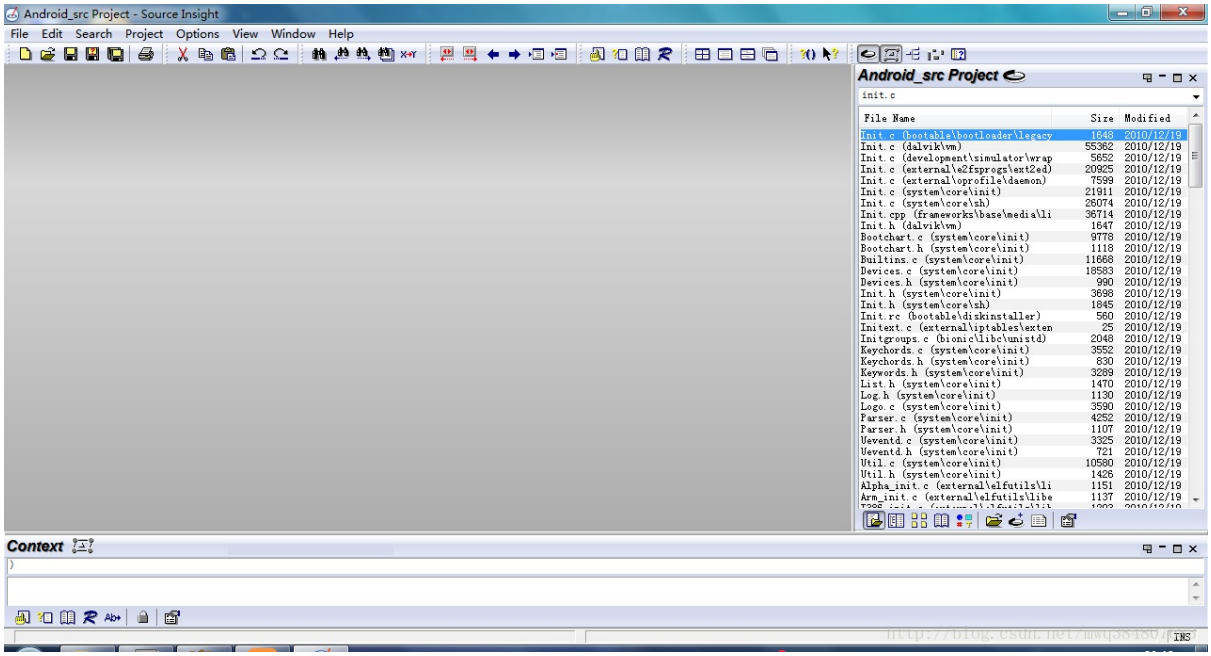



(No Project) - Source Insight
File Edit Search Project Options View Window Help
New Project Settings
Configuration
Project has its own configuration file.
Shares global configuration file
OK
Cancel
Help
Conditional Parsing
Conditions...
These condition values are project-specific. They are merged with the global condition list found in Preferences: Languages.
Project Source Directory - the main location of your source files:
E:\传智资料\JB_SRC
Browse...
Store function-local symbols in database. Local variables get displayed with syntax formatting faster, but the symbol database can increase in size by a factor of 2 or more.
Quick browsing for member names. You only type the member names of classes and structures to browse, but the symbol index and memory usage can increase by a factor of 2 or more.
Quick browsing for symbol syllables. You only type one or more syllables to locate symbols, but the symbol index and memory usage can increase by a factor of 4 or more.
Context


Android_src Project - Source Insight
File Edit Search Project Options View Window Help
Android_src Project
Name
Size
Modified
Add and Remove Project Files
File Name:
Close
E:\传智资料\JB_SRC
Directory
System Volume Information
Vm_disk
传智资料
Itcast规程
JB_SRC
Video
计划
软件
上课资料
原视频
File Name
Device
External
Frameworks
Hardware
Libcore
Ndk
Packages
Prebuilt
Sdk
System
Add
Add All
Add Tree
Remove Tree
Show only known document types
Project Files: (0)
Remove File
Remove All
Remove Special...
Add from list...
Help
Source Insight
Searching sub-directories for files to add to the project...
Cancel
Context


Android_src Project - Source Insight
File Edit Search Project Options View Window Help
Android_src Project
Add and Remove Project Files
File Name:
Close
E:\传智资料\JB_SRC
Directory
File Name
System Volume Information
Vm_disk
传智资料
Itcast规程
Video
计划
软件
上课资料
原视频
Device
External
Add
Add All
Add Tree
Remove Tree
Source Insight
69178 files will be added to the project.
OK to continue?
确定
取消
Project Files: (0)
Show only known document types
Remove File
Remove All
Remove Special...
Add from list...
Help
Context


Android_src Project - Source Insight
File Edit Search Project Options View Window Help
Android_src Project
Add and Remove Project Files
File Name:
Close
E:\传智资料\JB_SRC
Directory
File Name
System Volume Information
Vm_disk
传智资料
Itcast规程
Video
计划
软件
上课资料
原视频
Device
External
Frameworks
Hardware
Libcore
Ndk
Packages
Prebuilt
Sdk
System
Add
Add All
Add Tree
Remove Tree
Project Files: (69178)
e:\传智资料\jb_src\bionic\libc\arch-arm\bionic\Eabi.c
e:\传智资料\jb_src\bionic\libc\arch-arm\bionic\Exidx_dynamic.c
e:\传智资料\jb_src\bionic\libc\arch-arm\bionic\Exidx_static.c
e:\传智资料\jb_src\bionic\libc\arch-arm\bionic\Libgcc_compat.c
e:\传智资料\jb_src\bionic\libc\arch-arm\bionic\Strlen.c
e:\传智资料\jb_src\bionic\libc\arch-arm\include\Endian.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Asm.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Cdefs.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Cpu-features.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Exec.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Ieee.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Internal_types.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Kernel.h
e:\传智资料\jb_src\bionic\libc\arch-arm\include\machine\Limits.h
Show only known document types
Remove File
Remove All
Remove Special...
Add from list...
Help
Context



- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

Activity相关面试题

- onSaveInstanceState源码内核分析
- 深刻剖析activity启动模式-1
- 深刻剖析activity启动模式-2
- 深刻剖析activity启动模式-3
- Activity Task和Process之间的关系
- 为什么service里面startActivity抛异常
- App优雅退出
- onCreate源码分析

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

经常有人问，后台的activity被系统自动回收的话，怎么回到界面的时候恢复数据，通过一个真实案例给大家讲讲如何保存状态，然后带着大家分析onSaveInstanceState的源码。

当前页面侧滑菜单指向专题，用户做了如下操作：

1）当用户按下HOME键时。 2）长按HOME键，选择运行其他的程序时。 3）按下电源按键（关闭屏幕显示）时。 4）从activity A中启动一个新的activity时。 5）屏幕方向切换时，例如从竖屏切换到横屏时。

失去焦点，activity很可能被进程终止！被KILL掉了，这时候就需要能保存当前的状态，不然下次用户再次进来看到的还是新闻，这样用户体验就不够好，代码有删减，我自己项目就这样使用的，解决方案如下：

```
@Override
public void onSaveInstanceState(Bundle outState) {
  outState.putInt("newsCenter_position", newsCenterPosition);
  outState.putInt("smartService_position", smartServicePosition);
  outState.putInt("govAffairs_position", govAffairsPosition);
  super.onSaveInstanceState(outState);
}
```

如上代码可知：

1）界面被回收之后调用onSaveInstanceState方法保存当前的状态，每个侧滑菜单选项都有一个位置。

```
@Override
public void onCreate(Bundle savedInstanceState) {
if (savedInstanceState != null
    && savedInstanceState.containsKey("newsCenter_position")) {
    newsCenterPosition = savedInstanceState
                    .getInt("newsCenter_position");
    smartServicePosition = savedInstanceState
                    .getInt("smartService_position");
    govAffairsPosition = savedInstanceState
                    .getInt("govAffairs_position");
}

    super.onCreate(savedInstanceState);
}
```

由以上代码可知：

1）判断当前Bundle 是否有刚刚我们保存的位置，如果不为空，从当前的Bundle取出来，给每一个位置赋值。

```
public void switchMenu(int type) {
switch (type) {
     case NEWS_CENTER:
     if (newsCenterAdapter == null) {
         newsCenterAdapter = new MenuAdapter(ct, newsCenterMenu);
         newsCenterclassifyLv.setAdapter(newsCenterAdapter);
      } else {
         newsCenterAdapter.notifyDataSetChanged();
      }
     newsCenterAdapter.setSelectedPosition(newsCenterPosition);
      break;
      case SMART_SERVICE:
     if (smartServiceAdapter == null) {
        smartServiceAdapter = new MenuAdapter(ct, smartServiceMenu);
                  smartServiceclassifyLv.setAdapter(smartServiceAdapter);
     } else {
        smartServiceAdapter.notifyDataSetChanged();
     }
         smartServiceAdapter.setSelectedPosition(smartServicePosition);
     break;
    case GOV_AFFAIRS:
    if (govAffairsAdapter == null) {
        govAffairsAdapter = new MenuAdapter(ct, govAffairsMenu);
         govAffairsclassifyLv.setAdapter(govAffairsAdapter);
     } else {
        govAffairsAdapter.notifyDataSetChanged();
```

```
     }
        govAffairsAdapter.setSelectedPosition(govAffairsPosition);
    break;
}
```

以上代码可知：

1）根据当前的位置设置到adapter当中，这样下次用户进来就还是专题了。

总结下savedInstanceState的使用，代码如下：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if(savedInstanceState != null)
            System.out.println("onCreate() : " + savedInstanceState.getString("octopus"));
    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        System.out.println("onRestoreInstanceState() : " + savedInstanceState.getString("octopus"));
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        outState.putString("octopus", "www.baidu.com");
        System.out.println("onSaveInstanceState() : save date www.baidu.com");
    }

}
```

横竖屏切换，打印结果如下：

```
I/System.out( 8167): onSaveInstanceState() : save date www.baidu.com
I/System.out( 8167): onCreate() : www.baidu.com
I/System.out( 8167): onRestoreInstanceState() : www.baidu.com
```

从打印结果可以看出来，当前Activity被系统回收之后,会调用onSaveInstanceState()保存状态，然后在activity判断bundler是否有当前状态，如果只是到这，估计你们就会吐槽没啥含金量，没办法硬着头皮上，接着咱们来分onSaveInstanceState()源码，请看如下代码：

```
 @Override
 public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

 }
```

以上代码可知

1）调用父类Activity源码里面的onSaveInstanceState方法，代码如下：

```
protected void onSaveInstanceState(Bundle outState) {
   outState.putBundle(WINDOW_HIERARCHY_TAG, mWindow.saveHierarchyState());
   Parcelable p = mFragments.saveAllState();
   if (p != null) {
     outState.putParcelable(FRAGMENTS_TAG, p);
    }
    ......
}
```

以上代码可知

1）outState.put一个tag调用了mWindow里面的saveHierarchyState方法，继续分析Window源代码。

2）window是抽象类调用子类PhoneWindow里面的saveHierarchyState方法代码如下：

```
@Override
public Bundle saveHierarchyState() {
  Bundle outState = new Bundle();
  if (mContentParent == null) {
    return outState;
  }

  SparseArray<Parcelable> states = new SparseArray<Parcelable>();
  mContentParent.saveHierarchyState(states);
  outState.putSparseParcelableArray(VIEWS_TAG, states);
  ......
  return outState;
}
```

以上代码可知

1) Bundle outState = new Bundle()初始化Bundle对象，Bundle实现了Parcelable接口。

2）states = new SparseArray()并且把自己放到outState当中。

3）mContentParent.saveHierarchyState(states)，整个View树的顶层视图保存了层级状态代码如下：

```
public void saveHierarchyState(SparseArray<Parcelable> container) {
        dispatchSaveInstanceState(container);
}
```

以上代码可知：

1）调相应的dispatchSaveInstanceState方法，代码如下：

```
protected void dispatchSaveInstanceState(SparseArray<Parcelable> container) {
   if (mID != NO_ID && (mViewFlags & SAVE_DISABLED_MASK) == 0) {
    mPrivateFlags &= ~PFLAG_SAVE_STATE_CALLED;
    Parcelable state = onSaveInstanceState();
    if ((mPrivateFlags & PFLAG_SAVE_STATE_CALLED) == 0) {
      throw new IllegalStateException(
        "Derived class did not call         super.onSaveInstanceState()");
    }
   if (state != null) {
     // Log.i("View", "Freezing #" + Integer.toHexString(mID)
     // + ": " + state);
     container.put(mID, state);
   }
```

```
    }
  }
```

以上代码可知：

1） mID != NO_ID 判断一个View必须有一个id，也就是说你要么在xml里通过android:id指定要么在代码里通过setId，但是你从如上代码压根是看不出来谷歌想干啥，必须全局搜索NO_ID 和 mID ，一般在源码里面都会有谷歌工程师的注释方便我们理解，搜索NO_ID 可知代码如下：

```
/**
   * Used to mark a View that has no ID.
   */
   public static final int NO_ID = -1;
```

原来NO_ID用来标记没有id的View，搜索mID可知原来在如下代码赋值

```
public void setId(@IdRes int id) {
        mID = id;
        if (mID == View.NO_ID && mLabelForId != View.NO_ID) {
            mID = generateViewId();
        }
    }
```

经常当我们看不懂谷歌源码的时候，可以通过曲线救国的方式，看看英文注释，看看源码哪个地方用到当前的类或者方法或者变量，这样就好理解了，好了扯远了，继续分析代码；

2）通过if判断，检测子类是否调用父类的onSaveInstanceState()方法，否则会抛异常，突然看到这才明白，还记得刚刚开始学Android的时候，经常一不小心就把代码里面的super.onCreate(savedInstanceState);这行代码删掉，报了错误还看不懂，原来系统在这里检测了，都怪自己曾经太年轻。

3）container.put(mID, state)这行代码，将state放进SparseArray中，以view自身的id为key，并且从注释来看打印mID的Hex值用来保证每页的id必须是唯一的，难怪每当我给view取id的时候，一个页面有重复的id就会报错，谷歌大婶在这里做判断了，腻害了word哥，总是百思不得其姐，凭啥不让我共用id(因为取名字太难了)，原来是想把id做为key来使用。

4）走到这onSaveInstanceState()，调用如下代码：

```
@CallSuper
protected Parcelable onSaveInstanceState() {
        mPrivateFlags |= PFLAG_SAVE_STATE_CALLED;
        ......
        return BaseSavedState.EMPTY_STATE;
}
```

以上代码可知：

1）设置位标志， 默认不save任何东西，状态为空，这就是为啥我们每次随便写个类继承activity实现onCreate方法的时候可以使用参数savedInstanceState保存状态，因为默认为null，代码如下：

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        savedInstanceState.putString("key","value");
}
```

至此整个savedInstanceState保存状态源码分析完毕。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态
- 深刻剖析activity启动模式(一)
- 深刻剖析activity启动模式(二)

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

Activity的四种启动模式如下：

standard、singleTop、singleTask、singleInstance

我们一边讲理论一边结合案例来全面学习这四种启动模式。 为了打印方便，定义一个基础BaseActivity，在其onCreate方法和onNewIntent方法中打印出当前Activity的日志信息，主要包括所属的task，当前类的hashcode，之后我们进行测试的Activity都直接继承该BaseActivity

```
public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i("maweiqi", "*****onCreate()方法******");
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.h
ashCode());
        dumpTaskAffinity();
    }

    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        Log.i("maweiqi", "*****onNewIntent()方法*****");
        Log.i("maweiqi", "onNewIntent: " + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this
.hashCode());
        dumpTaskAffinity();
    }

    protected void dumpTaskAffinity(){
        try {
            ActivityInfo info = this.getPackageManager()
                 .getActivityInfo(getComponentName(), PackageManager.GET_META_DATA);
            Log.i("maweiqi", "taskAffinity:"+info.taskAffinity);
        } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

standard-默认模式

这个模式是默认的启动模式，即标准模式，在不指定启动模式的前提下，系统默认使用该模式启动Activity，每次启动一个Activity都会重写创建一个新的实例，不管这个实例存不存在，这种模式下，谁启动了该模式的Activity，该Activity就属于启动它的Activity的任务栈中。

配置形式：

```
<activity android:name=".SecondActivity" android:launchMode="standard"/>
```

使用案例：

对于standard模式，android:launchMode可以不进行声明，因为默认就是standard。 SecondActivity的代码如下，入口MainActivity中有一个按钮进入该Activity，这个Activity中又有一个按钮启动SecondActivity。

```
public class MainActivity extends BaseActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    Button btn_standard= (Button)   findViewById(R.id.btn_standard);
    btn_standard.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
       Intent intent = new Intent(MainActivity.this, SecondActivity.class);
       startActivity(intent);
    }
   });
        Log.i("maweiqi", "*****onCreate()方法******");
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.h
ashCode());
    }
}
```

```
public class SecondActivity extends BaseActivity {
    private Button jump;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

        jump= (Button) findViewById(R.id.button3);
        jump.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              Intent intent = new Intent(SecondActivity.this, SecondActivity.class);
              startActivity(intent);
            }
        });
    }

}
```

测试方式：

MainActivity --> SecondActivity --> SecondActivity --> SecondActivity --> SecondActivity

输出的日志如下：

```
  MainActivity TaskId: 2 hasCode:1249333352
SecondActivity TaskId: 2 hasCode:1249526392
SecondActivity TaskId: 2 hasCode:1249424816
SecondActivity TaskId: 2 hasCode:1249439692
SecondActivity TaskId: 2 hasCode:1249459968
```

测试结果：

1）日志输出了四次SecondActivity的和一次MainActivity的，并且所属的任务栈的id都是2，这也验证了谁启动了该模式的Activity，该Activity就属于启动它的Activity的任务栈中这句话.因为启动SecondActivity的是MainActivity，而MainActivity的taskId是2，因此启动的SecondActivity也应该属于id为2的这个task，后续的3个SecondActivity是被SecondActivity这个对象启动的，因此也应该还是2，所以taskId都是2。

2）并且每一个Activity的hashcode都是不一样的，说明他们是不同的实例，即“每次启动一个Activity都会重写创建一个新的实例”

singleTop模式

这个模式下，如果新的activity已经位于栈顶，那么这个Activity不会被重新创建，同时它的onNewIntent方法会被调用，通过此方法的参数我们可以去除当前请求的信息。如果栈顶不存在该Activity的实例，则情况与standard模式相同。需要注意的是这个Activity它的onCreate()，onStart()方法不会被调用，因为它并没有发生改变。

配置形式：

```
<activity android:name=".SingleTopActivity" android:launchMode="singleTop"/>
<activity android:name=".OtherTopActivity" android:launchMode="singleTop"/>
```

使用案例：

```
public class SingleTopActivity extends BaseActivity {
    private Button jump, jump2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_singletop);

        jump = (Button) findViewById(R.id.button);
        jump2 = (Button) findViewById(R.id.button2);
        jump.setOnClickListener(new View.OnClickListener() {
       @Override
        public void onClick(View v) {
        Intent intent = new Intent(SingleTopActivity.this, SingleTopActivity.class);
        startActivity(intent);
        }
      });
       jump2.setOnClickListener(new View.OnClickListener() {
       @Override
        public void onClick(View v) {
          Intent intent = new Intent(SingleTopActivity.this, OtherTopActivity.class);
          startActivity(intent);
          }
       });
        Log.i("maweiqi", "*****onCreate()方法******");
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.h
ashCode());
    }
}
```

```
public class OtherTopActivity extends AppCompatActivity {
    private Button jump;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_other);

        jump= (Button) findViewById(R.id.btn_other);
        jump.setOnClickListener(new View.OnClickListener() {
      @Override
       public void onClick(View v) {
         Intent intent = new Intent(OtherTopActivity.this, SingleTopActivity.class);
         startActivity(intent);
            }
        });
        Log.i("maweiqi", "*****onCreate()方法******");
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.h
ashCode());
    }
```

```
}
```

操作和standard模式类似，直接贴输出日志

```
onCreate: MainActivity TaskId: 3 hasCode:1249332216
onCreate: SingleTopActivity TaskId: 3 hasCode:1249464444
onNewIntent: SingleTopActivity TaskId: 3 hasCode:1249464444
onNewIntent: SingleTopActivity TaskId: 3 hasCode:1249464444
onNewIntent: SingleTopActivity TaskId: 3 hasCode:1249464444
```

测试结果：

1）除了第一次进入SingleTopActivity这个Activity时，输出的是onCreate方法中的日志，后续的都是调用了onNewIntent方法，并没有调用onCreate方法，并且四个日志的hashcode都是一样的。

2）说明栈中只有一个实例。这是因为第一次进入的时候，栈中没有该实例，则创建，后续的三次发现栈顶有这个实例，则直接复用，并且调用onNewIntent方法。

3）那么假设栈中有该实例，但是该实例不在栈顶情况又如何呢？ 我们先从MainActivity中进入到SingleTopActivity，然后再跳转到OtherActivity中，再从OtherActivity中跳回SingleTopActivity，再从SingleTopActivity跳到SingleTopActivity中，看看整个过程的日志。

输出的日志如下：

```
onCreate: SingleTopActivity TaskId: 4 hasCode:1249520904
onCreate: OtherTopActivity TaskId: 4 hasCode:1249420244
onCreate: SingleTopActivity TaskId: 4 hasCode:1249448776
onCreate: SingleTopActivity TaskId: 4 hasCode:1249448776
onNewIntent: SingleTopActivity TaskId: 4 hasCode:1249448776
```

测试结果：

1）从MainActivity进入到SingleTopActivity时，新建了一个SingleTopActivity对象。

2）从SingleTopActivity跳到OtherActivity时，新建了一个OtherActivity，此时task中存在三个Activity，从栈底到栈顶依次是MainActivity，SingleTopActivity，OtherActivity。

3）此时如果再跳到SingleTopActivity，即使栈中已经有SingleTopActivity实例了，但是依然会创建一个新的SingleTopActivity实例，这一点从上面的日志的hashCode可以看出，此时栈顶是SingleTopActivity，如果再跳到SingleTopActivity，就会复用栈顶的SingleTopActivity，即会调用SingleTopActivity的onNewIntent方法。这就是上述日志的全过程。

对以上内容进行总结

standard启动模式是默认的启动模式，每次启动一个Activity都会新建一个实例不管栈中是否已有该Activity的实例。

singleTop模式分3种情况

1）当前栈中已有该Activity的实例并且该实例位于栈顶时，不会新建实例，而是复用栈顶的实例，并且会将Intent对象传入，回调onNewIntent方法

2）当前栈中已有该Activity的实例但是该实例不在栈顶时，其行为和standard启动模式一样，依然会创建一个新的实例

3）当前栈中不存在该Activity的实例时，其行为同standard启动模式standard和singleTop启动模式都是在原任务栈中新建Activity实例，不会启动新的Task

singleTask模式

在这个模式下，如果栈中存在这个Activity的实例就会复用这个Activity，不管它是否位于栈顶，复用时，会将它上面的Activity全部出栈，并且会回调该实例的onNewIntent方法。

配置形式：

```
<activity android:name=".SingleTaskActivity" android:launchMode="singleTask"/>
<activity android:name=".OtherTaskActivity" android:launchMode="singleTask"/>
```

使用案例：

```
public class SingleTaskActivity extends BaseActivity {
    private Button jump,jump2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_task);
        jump = (Button) findViewById(R.id.btn_task);
        jump2 = (Button) findViewById(R.id.btn_other);
        jump.setOnClickListener(new View.OnClickListener() {
      @Override
       public void onClick(View v) {
      Intent intent = new Intent(SingleTaskActivity.this, SingleTaskActivity.class);
      startActivity(intent);
      }
     });
 jump2.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
    Intent intent = new Intent(SingleTaskActivity.this, OtherTaskActivity.class);
     startActivity(intent);
   }
 });
}}
```

```
public class OtherTaskActivity extends BaseActivity {
    private Button jump;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_other_task);

        jump= (Button) findViewById(R.id.btn_other);
        jump.setOnClickListener(new View.OnClickListener() {
      @Override
    public void onClick(View v) {
       Intent intent = new Intent(OtherTaskActivity.this, SingleTaskActivity.class);
       startActivity(intent);
    }
});
}
```

```
}
```

日志输出

```
onCreate: MainActivity TaskId: 5 hasCode:1249321980
onCreate: SingleTaskActivity TaskId: 5 hasCode:1249515136
onCreate: OtherTaskActivity TaskId: 6 hasCode:1249386172
onNewIntent: SingleTaskActivity TaskId: 6 hasCode:1249513244
```

测试结果：

1）从MainActiviyty进入到SingleTaskActivity，再进入到OtherActivity后，此时栈中有3个Activity实例，并且SingleTaskActivity不在栈顶。

2）而在OtherActivity跳到SingleTaskActivity时，并没有创建一个新的SingleTaskActivity，而是复用了该实例，并且回调了onNewIntent方法。并且原来的OtherActivity出栈了，具体见下面的信息，使用命令adb shell dumpsys activity activities可进行查看

```
Running activities (most recent first):
      TaskRecord{3c727e #11 A=com.maweiqi.task U=0 sz=2}
        Run #1: ActivityRecord{5a00d1e u0 com.maweiqi.task/.SingleTaskActivity t11}
        Run #0: ActivityRecord{2dce0b u0 com.maweiqi.task/.MainActivity t11}
```

可以看到当前栈中只有两个Activity，即原来栈中位于SingleTaskActivity 之上的Activity都出栈了。

singleInstance-全局唯一模式

该模式具备singleTask模式的所有特性外，与它的区别就是，这种模式下的Activity会单独占用一个Task栈，具有全局唯一性，即整个系统中就这么一个实例，由于栈内复用的特性，后续的请求均不会创建新的Activity实例，除非这个特殊的任务栈被销毁了。以singleInstance模式启动的Activity在整个系统中是单例的，如果在启动这样的Activiyt时，已经存在了一个实例，那么会把它所在的任务调度到前台，重用这个实例。

singleInstance示例一配置形式：

```
<activity android:name=".MainActivity" android:launchMode="standard">
    <intent-filter>
       <action android:name="android.intent.action.MAIN"/>

      <category android:name="android.intent.category.LAUNCHER"/>
     </intent-filter>
</activity>
 <activity android:name=".SingleInstanceActivity" android:launchMode="singleInstance"/>
```

```
public class MainActivity extends BaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn_standard= (Button) findViewById(R.id.btn_standard);
        btn_standard.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, SingleInstanceActivity.class);
```

```
                startActivity(intent);
            }
        });

    }
}
```

```
public class SingleInstanceActivity extends BaseActivity  {
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_instance);
        Button button4 = (Button) findViewById(R.id.button4);
        button4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(SingleInstanceActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

测试方式：

MainActivity--> SingleInstanceActivity--> MainActivity--> SingleInstanceActivity

日志输出

```
onCreate: MainActivity TaskId: 12 hasCode:201331476
onCreate: SingleInstanceActivity TaskId: 13 hasCode:57987178
onCreate: MainActivity TaskId: 12 hasCode:254253633
onNewIntent: SingleInstanceActivity TaskId: 13 hasCode:57987178
```

测试结果

1）两个SingleInstanceActivity是同一个实例。

2） 第一次进入的MainActivity和第一次进入的SingleInstanceActivity位于不同的task中。

3） 两个MainActivity是位于同一个task中的不同实例。

4）这个结论与预期是相同的，即，singleInstance类型的Activity的实例只能有一个，而且它只允许存在于单独的一个task中。

5）至于为什么两个MainActivity是位于同一个task中的不同实例，那是因为它是standard类型的，我们可以将ActivityTest修改为singleTop等其他类型进行测试。

singleInstance示例二配置形式：

MainActivity的模式改为"singleTop",修改后的manifest如下:

```
<activity android:name=".MainActivity" android:launchMode="singleTop">
    <intent-filter>
       <action android:name="android.intent.action.MAIN"/>

      <category android:name="android.intent.category.LAUNCHER"/>
```

```
    </intent-filter>
</activity>
 <activity android:name=".SingleInstanceActivity" android:launchMode="singleInstance"/>
```

MainActivity进入SingleInstanceActivity,在从SingleInstanceActivity 进入MainActivity,在从MainActivity进入SingleInstanceActivity

日志输出

```
onCreate: MainActivity TaskId: 15 hasCode:201331476
onCreate: SingleInstanceActivity TaskId: 16 hasCode:57987178
onNewIntent: MainActivity TaskId: 15 hasCode:201331476
onNewIntent: SingleInstanceActivity TaskId: 16 hasCode:57987178
```

测试结果

1）两个SingleInstanceActivity是同一个实例。

2）第一次进入的MainActivity和第一次进入的SingleInstanceActivity位于不同的task中。

3）两个MainActivity是同一个实例。

launchMode模式总结

1. standard

在该模式下，Activity可以拥有多个实例，并且这些实例既可以位于同一个task，也可以位于不同的task。

2.singleTop

该模式下，在同一个task中，如果存在该Activity的实例，并且该Activity实例位于栈顶(即，该Activity位于前端)，则调用startActivity()时，不再创建该Activity的示例；而仅仅只是调用Activity的onNewIntent()。否则的话，则新建该Activity的实例，并将其置于栈顶。

3. singleTask

只容许有一个包含该Activity实例的task存在！ 总的来说：singleTask的结论与android:taskAffinity相关(下章在讲),以A启动B来说

1） 当A和B的taskAffinity相同时：第一次创建B的实例时，并不会启动新的task，而是直接将B添加到A所在的task；否则，将B所在task中位于B之上的全部Activity都删除，然后跳转到B中。 2） 当A和B的taskAffinity不同时：第一次创建B的实例时，会启动新的task，然后将B添加到新建的task中；否则，将B所在task中位于B之上的全部Activity都删除，然后跳转到B中。

4. singleInstance

顾名思义，是单一实例的意思，即任意时刻只允许存在唯一的Activity实例，而且该Activity所在的task不能容纳除该Activity之外的其他Activity实例！ 它与singleTask有相同之处，也有不同之处。 相同之处：任意时刻，最多只允许存在一个实例。 不同之处： 1） singleTask受android:taskAffinity属性的影响，而singleInstance不受android:taskAffinity的影响。 2） singleTask所在的task中能有其它的Activity，而singleInstance的task中不能有其他

Activity。 3） 当跳转到singleTask类型的Activity，并且该Activity实例已经存在时，会删除该Activity所在task中位于该Activity之上的全部Activity实例；而跳转到singleInstance类型的Activity，并且该Activity已经存在时，不需要删除其他Activity，因为它所在的task只有该Activity唯一一个Activity实例。

参考链接：http://blog.csdn.net/sbsujjbcy/article/details/49360615

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态
- 深刻剖析activity启动模式(一)
- 深刻剖析activity启动模式(二)

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

实例验证singleTask启动模式

上篇文章将activity的四种启动模式就基本介绍完了。为了加深对启动模式的了解，下面会通过一个简单的例子进行验证。由以上的介绍可知，standard和singleTop这两种启动模式行为比较简单，所以在下面的例子中，通过taskAffinity会对singleTask和singleInstance着重介绍。

验证启动singleTask模式的activity时是否会创建新的任务

这个实例中有三个Activity,分别为：MainActivity，SecondActivity和ThirdActivity。

配置形式：

```
<activity  android:label="@string/app_name"
                  android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>



        <activity android:name=".SecondActivity"
                  android:launchMode="singleTask">
            <intent-filter >
                <action android:name="com.maweiqi.SecondActivity"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>

        <activity android:name=".ThirdActivity"
                  android:label="@string/app_name" >
        </activity>
```

说明：

MainActivity和ThirdActivity都是标准的启动模式，而SecondActivity的启动模式为singleTask。

使用案例：

```
public class MainActivity extends AppCompatActivity {
    private static final String ACTIVITY_NAME = "MainActivity";
    private static final String LOG_TAG = "maweiqi";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        findViewById(R.id.button1).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, SecondActivity.class);

                startActivity(intent);
            }
        });

        int taskId = getTaskId();
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
```

```
    }
}
```

```
public class SecondActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

        findViewById(R.id.button2).setOnClickListener(new View.OnClickListener() {
           @Override
            public void onClick(View v) {
              Intent intent = new Intent(SecondActivity.this, ThirdActivity.class);
              startActivity(intent);
            }
        });

        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() + " TaskId: "
        + getTaskId() + " hasCode:" + this.hashCode());

    }
}
```

```
public class ThirdActivity extends AppCompatActivity {
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() +
        " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
    }
}
```

测试方式：

MainActivity -->SecondActivity -->ThirdActivity。

输出的日志如下：

```
onCreate: MainActivity TaskId: 21 hasCode:199785693
onCreate: SecondActivity TaskId: 21 hasCode:171956091
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
   TaskRecord{838c0b8 #21 A=com.open.android.task1 U=0 sz=2}
      Run #1: ActivityRecord{abf1b0a u0 com.open.android.task1/.SecondActivity t21}
      Run #0: ActivityRecord{4e579b u0 com.open.android.task1/.MainActivity t21}
```

测试结果：

1）MainActivity和SecondActivity是启动在同一个任务中

2）在SecondActivity增加一个taskAffinity属性，如下所示：

```
<activity android:name=".SecondActivity"
```

```
                android:launchMode="singleTask"
                android:taskAffinity="com.maweiqi.second">
        <intent-filter >
                <action android:name="com.maweiqi.SecondActivity"/>
                <category android:name="android.intent.category.DEFAULT"/>
         </intent-filter>
    </activity>
```

测试方式：

MainActivity -->SecondActivity -->ThirdActivity。

输出的日志如下：

```
onCreate: MainActivity TaskId: 24 hasCode:199785693
onCreate: SecondActivity TaskId: 25 hasCode:171956091
onCreate: ThirdActivity TaskId: 25 hasCode:76684615
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
   TaskRecord{844539b #25 A=com.maweiqi.second U=0 sz=2}
      Run #2: ActivityRecord{2eb5348 u0 com.open.android.task1/.ThirdActivity t25}
      Run #1: ActivityRecord{119f0df u0 com.open.android.task1/.SecondActivity t25}
   TaskRecord{b730338 #24 A=com.open.android.task1 U=0 sz=1}
      Run #0: ActivityRecord{2e05e2a u0 com.open.android.task1/.MainActivity t24}
```

测试结果：

1）MainActivity和SecondActivity运行在不同的任务中了

2）ThirdActivity和SecondActivity运行在同一个任务中

在这里便引出了manifest文件中的一个重要属性，taskAffinity。在官方文档中可以得到关于taskAffinity的以下信息

taskAffinity

1）taskAffinity表示当前activity具有亲和力的一个任务（翻译不是很准确，原句为The task that the activity has an affinity for.），大致可以这样理解，这个 taskAffinity表示一个任务，这个任务就是当前activity所在的任务。

2）在概念上，具有相同的affinity的activity（即设置了相同taskAffinity属性的activity）属于同一个任务。

3） 一个任务的affinity决定于这个任务的根activity（root activity）的taskAffinity。

4） 默认情况下，一个应用中的所有activity具有相同的taskAffinity，即应用程序的包名。我们可以通过设置不同的taskAffinity属性给应用中的activity分组，也可以把不同的 应用中的activity的taskAffinity设置成相同的值。

5）为一个activity的taskAffinity设置一个空字符串，表明这个activity不属于任何task。

结果分析：

1）这就可以解释上面示例中的现象了，由第4条可知，MainActivity和SecondActivity具有不同的taskAffinity，MainActivity的taskAffinity为com.open.android.task1，SecondActivity的taskAffinity为com.maweiqi.second。

2）当新启动的activity（SecondActivity）是以FLAG_ACTIVITY_NEW_TASK标志启动时（只有singleTask模式才会保证“single in task”,只使用FLAG_ACTIVITY_NEW_TASK并不保证“single in task”，或者说singleTask包含了FLAG_ACTIVITY_NEW_TASK，反之却不成立），framework会检索是否已经存在了一个affinity为com.maweiqi.second的任务（即一个TaskRecord对象）

3）如果存在这样的一个任务，则检查在这个任务中是否已经有了一个SecondActivity的实例。

3-1）如果已经存在一个SecondActivity的实例，则会重用这个任务和任务中的SecondActivity实例，将这个任务调到前台，清除位于SecondActivity上面的所有Activity，显示SecondActivity，并调用SecondActivity的onNewIntent（）；

3-2）如果不存在一个SecondActivity的实例，会在这个任务中创建SecondActivity的实例，并调用onCreate()方法

3-3）这是在设置了singleTask模式的情况下会这样，在没有设置singleTask模式的情况下（即默认的standard模式）使用FLAG_ACTIVITY_NEW_TASK，并不会清除位于SecondActivity上面的所有Activity，而是会在task的上面重新创建一个SecondActivity。也就是说此时task中有两个SecondActivity。

4）如果不存在这样的一个任务，会创建一个新的affinity为com.maweiqi.second的任务，并且将SecondActivity启动到这个新的任务中

上面讨论的是设置taskAffinity属性的情况,如果SecondActivity只设置启动模式为singleTask,而不设置taskAffinity,即三个Activity的taskAffinity相同,都为应用的包名，那么SecondActivity是不会开启一个新任务的，framework中的判定过程如下:

1）在MainActivity启动SecondActivity时,发现启动模式为singleTask,那么设定他的启动标志为FLAG_ACTIVITY_NEW_TASK

2）然后获得SecondActivity的taskAffinity,即为包名com.open.android.task1检查是否已经存在一个affinity为com.open.android.task1的任务,这个任务是存在的,就是MainActivity所在的任务,这个任务是在启动MainActivity时开启的

3）既然已经存在这个任务,就检索在这个任务中是否存在一个SecondActivity的实例,发现不存在在这个已有的任务中启动一个SecondActivity的实例

为了作一个清楚的比较,列出SecondActivity启动模式设为singleTask,并且taskAffinity设为com.maweiqi.second时的启动过程

1）在MainActivity启动SecondActivity时，发现启动模式为singleTask，那么设定他的启动标志为FLAG_ACTIVITY_NEW_TASK

2）然后获得SecondActivity的taskAffinity,即com.maweiqi.second检查是否已经存在一个affinity为com.maweiqi.second的任务,这个任务是不存在的创建一个新的affinity为com.maweiqi.second的任务,并且将SecondActivity启动到这个新的任务中

framework中对任务和activity的调度是很复杂的,尤其是把启动模式设为singleTask或者以FLAG_ACTIVITY_NEW_TASK标志启动时.所以,在使用singleTask和FLAG_ACTIVITY_NEW_TASK时,要仔细测试应用程序.

实例验证将两个不同app中的不同的singleTask模式的Activity的taskAffinity设成相同

官方文档中提到，可以把不同的 应用中的activity的taskAffinity设置成相同的值，这样的话这两个activity虽然不在同一应用中，却会在运行时分配到同一任务中，下面对此进行验证，在这里，会使用上面的示例,并创建一个新的示例AndroidTaskTest3。AndroidTaskTest3由两个activity组成，分别为MianActivity和OtherActivity，在MianActivity中点击按钮会启动OtherActivity,两个应用代码和布局一样，仅列出清单文件，两份清单文件如下:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.open.android.task1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
       <activity android:name=".SecondActivity"
                  android:launchMode="singleTask"
                  android:taskAffinity="com.maweiqi.second">
            <intent-filter >
                <action android:name="com.maweiqi.SecondActivity"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
        <activity android:name=".ThirdActivity"/>
    </application>
</manifest>
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.open.android.task3">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name=".OtherActivity"
            android:taskAffinity="com.maweiqi.second"
            android:launchMode="singleTask">
        </activity>
    </application>

</manifest>
```

可以看到OtherActivity和SecondActivity的启动模式都被设置为singleTask,并且taskAffinity属性被设置为com.maweiqi.second.现在将这两个应用安装在设备上。执行以下操作:

测试方式：

1）MainActivity --> SecondActivity

2）MainActivity --> OtherActivity

启动AndroidTaskTest应用，在它的MianActivity中点击按钮开启SecondActivity，由上面的介绍可知secondActivity是运行在一个新任务中的，这个任务就是com.maweiqi.second。

然后按Home键回到Launcher，启动AndroidTaskTest3，在启动AndroidTaskTest3的入口MianActivity时，会自动启动新的任务，那么现在一共有三个任务，AndroidTaskTest的MianActivity和SecondActivity分别占用一个任务，AndroidTaskTest3的MianActivity也占用一个任务。

在AndroidTaskTest3的MianActivity中点击按钮启动OtherActivity，那么这个OtherActivity是在哪个任务中呢？

日志输出

```
***AndroidTaskTest应用测试日志输出***
onCreate：MainActivity TaskId: 29 hasCode:193251508
onCreate：SecondActivity TaskId: 30 hasCode:171956091

***AndroidTaskTest3应用测试日志输出***
onCreate：MainActivity TaskId: 31 hasCode:199785693
onCreate：OtherActivity TaskId: 30 hasCode:171956091
```

下面执行adb shell dumpsys activity命令，发现有以下输出：

```
Task id #30
  TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
    Intent { flg=0x10000000 cmp=com.open.android.task1/.SecondActivity }
     Hist #1: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
       Intent { flg=0x10400000 cmp=com.open.android.task3/.OtherActivity }
       ProcessRecord{12090b5 27543:com.open.android.task3/u0a62}
      Hist #0: ActivityRecord{1048af6 u0 com.open.android.task1/.SecondActivity t30}
       Intent { flg=0x10000000 cmp=com.open.android.task1/.SecondActivity }
       ProcessRecord{5bc013e 26035:com.open.android.task1/u0a59}

 Task id #31
   TaskRecord{dce52bb #31 A=com.open.android.task3 U=0 sz=1}
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task3/.MainActivity }
        Hist #0: ActivityRecord{f9e58c5 u0 com.open.android.task3/.MainActivity t31}
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task3/.MainActivity }
          ProcessRecord{12090b5 27543:com.open.android.task3/u0a62}

 Task id #29
      TaskRecord{5b063d8 #29 A=com.open.android.task1 U=0 sz=1}
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task1/.MainActivity }
        Hist #0: ActivityRecord{689947d u0 com.open.android.task1/.MainActivity t29}
          Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.
android.task1/.MainActivity }
          ProcessRecord{5bc013e 26035:com.open.android.task1/u0a59}

Running activities (most recent first):
   TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
        Run #3: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
   TaskRecord{dce52bb #31 A=com.open.android.task3 U=0 sz=1}
        Run #2: ActivityRecord{f9e58c5 u0 com.open.android.task3/.MainActivity t31}
   TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
       Run #1: ActivityRecord{1048af6 u0 com.open.android.task1/.SecondActivity t30}
   TaskRecord{5b063d8 #29 A=com.open.android.task1 U=0 sz=1}
```

```
        Run #0: ActivityRecord{689947d u0 com.open.android.task1/.MainActivity t29}

  mResumedActivity: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
```

结果分析

1）所以由此可见,AndroidTaskTest1的SecondActivity和AndroidTaskTest3的OtherActivity是在同一任务中的

2）由上面adb shell dumpsys activity命令的输出结果还可以看出，AndroidTaskTest1和AndroidTaskTest3这两个应用程序会开启两个进程，他们的所有组件分别运行在独立的进程中

3）AndroidTaskTest1所在进程的进程号为u0a59，AndroidTaskTest3所在进程的进程号为u0a62

4）com.maweiqi.second任务中的两个activity属于不同的应用，并且运行在不同的进程中，这也说明了一个问题：任务（Task）不仅可以跨应用（Application），还可以跨进程（Process）。

实例验证singleTask的另一意义：在同一个任务中具有唯一性

singleTask模式只意味着“可以在一个新的任务中开启”，至于是不是真的会在新任务中开启，在framework中还有其他条件的限制。由上面的介绍可知，这个条件为：是否已经存在了一个由他的taskAffinity属性指定的任务。这一点具有迷惑性，我们在看到singleTask这个单词的时候，会直观的想到它的本意：single in task。即，在同一个任务中，只会有一个该activity的实例。现在让我们进行验证：

为了验证这种情况，需要修改一下上面用到的AndroidTaskTest1示例。增加一个FourthActivity，并且MianActivity，SecondActivity，ThirdActivity和FourthActivity这四个activity都不设置taskAffinity属性，并且将SecondActivity启动模式设为singleTask，这样这四个activity会在同一个任务中开启。代码和软件界面就不列出了，只列出清单文件。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.open.android.task1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
       <activity android:name=".SecondActivity"
                  android:launchMode="singleTask"
                  >
         </activity>
        <activity android:name=".ThirdActivity"/>
        <activity android:name=".FourthActivity"/>
    </application>

</manifest>
```

测试方式：

MianActivity --> SecondActivity --> ThirdActivity --> FourthActivity

日志输出

```
onCreate: MainActivity TaskId: 34 hasCode:199785693
onCreate: SecondActivity TaskId: 34 hasCode:171956091
onCreate: ThirdActivity TaskId: 34 hasCode:240438941
onCreate: FourthActivity TaskId: 34 hasCode:168147209
```

由此可见这四个activity都是在同一个任务中的。再次执行adb shell dumpsys activity命令加以验证：

```
Running activities (most recent first):
  TaskRecord{d114530 #34 A=com.open.android.task1 U=0 sz=4}
     Run #3: ActivityRecord{edae6a3 u0 com.open.android.task1/.FourthActivity t34}
     Run #2: ActivityRecord{f9339a5 u0 com.open.android.task1/.ThirdActivity t34}
     Run #1: ActivityRecord{1a30096 u0 com.open.android.task1/.SecondActivity t34}
     Run #0: ActivityRecord{7e96fa4 u0 com.open.android.task1/.MainActivity t34}
```

测试结果：

1）同样可以说明目前这四个activity都运行在affinity为com.open.android.task1的任务中，即栈中的状态为MainActivity --> SecondActivity --> ThirdActivity --> FourthActivity。

下面执行在FourthActivity中点击按钮启动SecondActivity的操作，注意，SecondActivity的启动模式为singleTask，那么现在栈中的情况如何呢？

日志输出

```
没有日志输出，说明没有调用onCreate方法
```

再次执行adb shell dumpsys activity命令，有以下输出：

```
Running activities (most recent first):
  TaskRecord{d114530 #34 A=com.open.android.task1 U=0 sz=2}
     Run #1: ActivityRecord{1a30096 u0 com.open.android.task1/.SecondActivity t34}
     Run #0: ActivityRecord{7e96fa4 u0 com.open.android.task1/.MainActivity t34}
```

测试结果：

1）这时栈中的状态为MainActivity --> SecondActivity。确实确保了在任务中是唯一的，并且清除了同一任务中它上面的所有Activity。

2）那么这个SecondActivity的实例是重用的上次已有的实例还是重新启动了一个实例呢？可以观察系统Log， 发现系统Log没有改变，还是上面的四条Log。打印Log的语句是在各个Activity中的onCreate方法中执行的，没有打印出新的Log，说明SecondActivity的onCreate的方法没有重新执行，也就是说是重用的上次已经启动的实例，而不是销毁重建。

结果分析：

1）经过上面的验证，可以得出如下的结论：在启动一个singleTask的Activity实例时，如果系统中已经存在这样一个实例，就会将这个实例调度到任务栈的栈顶，并清除它当前所在任务中位于它上面的所有的activity。

实例验证singleInstance的行为

考谷歌官方文档，singleInstance的特点可以归结为以下三条：

1）以singleInstance模式启动的Activity具有全局唯一性，即整个系统中只会存在一个这样的实例

2）以singleInstance模式启动的Activity具有独占性，即它会独自占用一个任务，被他开启的任何activity都会运行在其他任务中（官方文档上的描述为，singleInstance模式的Activity不允许其他Activity和它共存在一个任务中）

3）被singleInstance模式的Activity开启的其他activity，能够开启一个新任务，但不一定开启新的任务，也可能在已有的一个任务中开启

下面对这三个特点分别验证，所使用的示例同样为AndroidTaskTest1，只不过会进行一些修改，下面列出它的清单文件：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.open.android.task1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>



        <activity android:name=".SecondActivity"
                  android:launchMode="singleInstance"
                  >
            <intent-filter>
                <action android:name="com.maweiqi.ACTION_MY"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>

        <activity android:name=".ThirdActivity"/>
        <activity android:name=".FourthActivity"/>
    </application>

</manifest>
```

由上面的清单文件可以知道，该应用包括四个activity，分别为MianActivity，SecondActivity，ThirdActivity，FourthActivity，其中SecondActivity启动模式设置为singleInstance。MianActivity可以开启SecondActivity，SecondActivity可以开启ThirdActivity。 并且为了可以在其他应用中开启SecondActivity，为SecondActivity设置了一个IntentFilter，这样就可以在其他应用中使用隐式Intent开启SecondActivity。

测试方式：

MianActivity --> SecondActivity --> ThirdActivity

为了更好的验证singleInstance的全局唯一性，还需要其他一个应用，新建AndroidTaskTest4。AndroidTaskTest4只需要一个MianActivity，在MainActivity中点击按钮会开启AndroidTaskTest1应用中的SecondActivity。开启AndroidTaskTest1应用中的SecondActivity的代码如下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent();
                intent.setAction("com.maweiqi.ACTION_MY");
                startActivity(intent);
            }
        });
    }
}
```

下面开始验证第一个特点：以singleInstance模式启动的Activity具有全局唯一性，即整个系统中只会存在一个这样的实例

执行如下操作：安装AndroidTaskTest1应用，点击MainActivity中的按钮，开启SecondActivity，可以看到如下log输出：

日志输出

```
onCreate: MainActivity TaskId: 35 hasCode:199785693
onCreate: SecondActivity TaskId: 36 hasCode:171956091
```

执行adb shell dumpsys activity命令，有以下输出：

```
  Running activities (most recent first):
      TaskRecord{2b68544 #36 A=com.open.android.task1 U=0 sz=1}
        Run #1: ActivityRecord{6d9f8c u0 com.open.android.task1/.SecondActivity t36}
      TaskRecord{ae9a62d #35 A=com.open.android.task1 U=0 sz=1}
        Run #0: ActivityRecord{d0429f5 u0 com.open.android.task1/.MainActivity t35}
```

以上可以说明，singleInstance模式的Activity总是会在新的任务中运行(前提是系统中还不存在这样的一个实例)。

下面验证它的全局唯一性，执行以下操作：安装另一个应用AndroidTaskTest4，在开启的MainActivity中点击按钮开启AndroidTaskTest1应用中的SecondActivity。看到打印出一条新的日志：

```
{act=com.maweiqi.ACTION_MY cmp=com.open.android.task1/.SecondActivity} from uid 10063 on display 0
```

执行adb shell dumpsys activity命令，有以下输出：

```
      Running activities (most recent first):
      TaskRecord{2b68544 #36 A=com.open.android.task1 U=0 sz=1}
        Run #2: ActivityRecord{6d9f8c u0 com.open.android.task1/.SecondActivity t36}
      TaskRecord{a7e9abd #37 A=com.open.android.task4 U=0 sz=1}
        Run #1: ActivityRecord{f50eec0 u0 com.open.android.task4/.MainActivity t37}
      TaskRecord{ae9a62d #35 A=com.open.android.task1 U=0 sz=1}
        Run #0: ActivityRecord{d0429f5 u0 com.open.android.task1/.MainActivity t35}
```

测试结果：

1）开启的SecondActivity就是上次创建的编号为6d9f8c的SecondActivity。

2）Log中没有再次输出关于SecondActivity的信息，说明SecondActivity并没有重新创建

结果分析：

以singleInstance模式启动的Activity在整个系统中是单例的，如果在启动这样的Activiyt时，已经存在了一个实例，那么会把它所在的任务调度到前台，重用这个实例。

下面开始验证第二个特点：以singleInstance模式启动的Activity具有独占性，即它会独自占用一个任务，被他开启的任何activity都会运行在其他任务中

重新安装AndroidTaskTest1应用，点击MainActivity中的按钮，开启SecondActivity，在SecondActivity中点击按钮，开启ThirdActivity。可以看到有如下Log输出：

测试方式：

MainActivity --> SecondActivity --> ThirdActivity

日志输出：

```
onCreate: MainActivity TaskId: 42 hasCode:199785693
onCreate: SecondActivity TaskId: 43 hasCode:171956091
onCreate: ThirdActivity TaskId: 42 hasCode:76684615
```

执行adb shell dumpsys activity命令，有以下输出：

```
 Running activities (most recent first):
      TaskRecord{ace0710 #42 A=com.open.android.task1 U=0 sz=2}
        Run #3: ActivityRecord{322319f u0 com.open.android.task1/.ThirdActivity t42}
      TaskRecord{abc9c0e #43 A=com.open.android.task1 U=0 sz=1}
        Run #2: ActivityRecord{903a842 u0 com.open.android.task1/.SecondActivity t43}
      TaskRecord{ace0710 #42 A=com.open.android.task1 U=0 sz=2}
        Run #1: ActivityRecord{e3c0ddf u0 com.open.android.task1/.MainActivity t42}
```

测试结果：

SecondActivity所在的任务为43，被SecondActivity启动的ThirdActivity所在的任务为42，这就说明以singleInstance模式启动的Activity具有独占性，即它会独自占用一个任务，被他开启的任何activity都会运行在其他任务中

下面开始验证第三个特点：被singleInstance模式的Activity开启的其他activity，能够在新的任务中启动，但不一定开启新的任务，也可能在已有的一个任务中开启

有上面对第二个特点的验证可以看到，被SecondActivity启动的ThirdActivity并没有运行在一个新开启的任务中，而是和MainActivity运行在了同一个已有的任务中，那么在什么情况下ThirdActivity才会启动一个新的任务呢？

现在对程序的清单文件做以下修改，为ThirdActivity增加一个属性taskAffinity：

配置如下：

```
<activity android:name=".ThirdActivity"
                  android:taskAffinity="com.maweiqi.second"/>
```

重新安装AndroidTaskTest1应用，执行和上一步中同样的操作：点击MainActivity中的按钮，开启SecondActivity，在SecondActivity中点击按钮，开启ThirdActivity。可以看到有如下输出：

```
onCreate: MainActivity TaskId: 44 hasCode:199785693
onCreate: SecondActivity TaskId: 45 hasCode:171956091
onCreate: ThirdActivity TaskId: 46 hasCode:76684615
```

执行adb shell dumpsys activity命令，有以下输出：

```
Running activities (most recent first):
      TaskRecord{3088b56 #46 A=com.maweiqi.second U=0 sz=1}
        Run #2: ActivityRecord{9eff1ed u0 com.open.android.task1/.ThirdActivity t46}
      TaskRecord{f9bb7d7 #45 A=com.open.android.task1 U=0 sz=1}
        Run #1: ActivityRecord{16c7b28 u0 com.open.android.task1/.SecondActivity t45}
      TaskRecord{e9af8c4 #44 A=com.open.android.task1 U=0 sz=1}
        Run #0: ActivityRecord{5c2ed47 u0 com.open.android.task1/.MainActivity t44}
```

测试结果：

1）被SecondActivity启动的ThirdActivity启动在了一个新的任务中，即在启动ThirdActivity时创建了一个新任务。这就说明被singleInstance模式的Activity A在开启另一activity B时，能够开启一个新任务，但是是不是真的开启新任务，还要受其他条件的限制，这个条件是：当前系统中是不是已经有了一个activity B的taskAffinity属性指定的任务。

其实这种行为和singleTask启动时的情况相同。在Activity的启动模式设置为singleTask时，启动时系统会为它加上FLAG_ACTIVITY_NEW_TASK标志，而被singleInstance模式的Activity开启的activity，启动时系统也会为它加上FLAG_ACTIVITY_NEW_TASK标志，所以他们启动时的情况是相同的，上面再验证singleTask时已经阐述过，现在重新说明一下：

结果分析：

由于ThirdActivity是被启动模式为singleInstance类型的Activity（即SecondActivity）启动的，framework会为它它加上FLAG_ACTIVITY_NEW_TASK标志，这时 framework会检索是否已经存在了一个affinity为com.maweiqi.second（即ThirdActivity的taskAffinity属性）的任务

1）如果存在这样的一个任务，则检查在这个任务中是否已经有了一个ThirdActivity的实例.

1-1）如果已经存在一个ThirdActivity的实例，则会重用这个任务和任务中的ThirdActivity实例，将这个任务调到前台，清除位于ThirdActivity上面的所有Activity，显示ThirdActivity，并调用ThirdActivity的onNewIntent（）。

1-2）如果不存在一个ThirdActivity的实例，会在这个任务中创建ThirdActivity的实例，并调用onCreate()方法

1-3）需要注意（1-1）有一种特殊情况，MainActivity, SecondActivity, ThirdActivity, FourthActivity 都不设置 taskAffinity.FourthActivity 设置为 singleInstance。测试 MainActivity -> SecondActivity -> ThirdActivity -> FourthActivity -> SecondActivity，从 FourthActivity 跳转到 SecondActivity, 是新开的一个 SecondActivity， 不会销毁 原 SecondActivity 上面的 ThirdActivity。

2）如果不存在这样的一个任务，会创建一个新的affinity为com.maweiqi.second的任务，并且将ThirdActivity启动到这个新的任务中.

如果ThirdActivity不设置taskAffinity，即ThirdActivity和MainActivity的taskAffinity相同，都为应用的包名，那么ThirdActivity是不会开启一个新任务的.

framework中的判定过程如下：

1）在SecondActivity启动ThirdActivity时，因为SecondActivity是singleInstance的，所以设定ThirdActivity的启动标志为FLAG_ACTIVITY_NEW_TASK

2）然后获得ThirdActivity的taskAffinity,即为包名com.open.android.task1

3）检查是否已经存在一个affinity为com.open.android.task1的任务，这个任务是存在的，就是MainActivity所在的任务，这个任务是在启动MainActivity时开启的

4） 既然已经存在这个任务,就检索在这个任务中是否存在一个ThirdActivity的实例,发现不存在

5）在这个已有的任务中启动一个SecondActivity的实例

为了作一个清楚的比较，列出ThirdActivity的taskAffinity属性设为com.maweiqi.second时的启动过程

1）在SecondActivity启动ThirdActivity时，因为SecondActivity是singleInstance的，那么设定ThirdActivity的启动标志为FLAG_ACTIVITY_NEW_TASK

2）然后获得ThirdActivity的taskAffinity，即为com.maweiqi.second

3）检查是否已经存在一个affinity为com.maweiqi.second的任务，这个任务是不存在的

4） 创建一个新的affinity为com.maweiqi.second的任务，并且将ThirdActivity启动到这个新的任务

到此singleInstance也介绍完了。

小结

由上述可知，Task是Android Framework中的一个概念，Task是由一系列相关的Activity组成的，是一组相关Activity的集合。Task是以栈的形式来管理的。

我们在操作软件的过程中，一定会涉及界面的跳转。其实在对界面进行跳转时，Android Framework既能在同一个任务中对Activity进行调度，也能以Task为单位进行整体调度。在启动模式为standard或singleTop时，一般是在同一个任务中对Activity进行调度，而在启动模式为singleTask或singleInstance是，一般会对Task进行整体调度。

对Task进行整体调度包括以下操作：

1）按Home键，将之前的任务切换到后台 2）长按Home键，会显示出最近执行过的任务列表 3）在Launcher或HomeScreen点击app图标，开启一个新任务，或者是将已有的任务调度到前台 4）启动singleTask模式的Activity时，会在系统中搜寻是否已经存在一个合适的任务，若存在，则会将这个任务调度到前台以重用这个任务。如果这个任务中已经存在一个要启动的Activity的实例，则清除这个实例之上的所有Activity，将这个实例显示给用户。如果这个已存在的任务中不存在一个要启动的Activity的实例，则在这个任务的顶端启动一个实例。若这个任务不存在，则会启动一个新的任务，在这个新的任务中启动这个singleTask模式的Activity的一个实例。 5）启动singleInstance的Activity时，会在系统中搜寻是否已经存在一个这个Activity的实例，如果存在，会将这个实例所在的任务调度到前台，重用这个Activity的实例（该任务中只有这一个Activity），如果不存在，会开启一个新任务，并在这个新任务中启动这个singleInstance模式的Activity的一个实例。

前面介绍了通过launchMode设置Activity的启动模式。本章接着介绍Activity的启动模式相关内容，讲解的内容是Intent与启动模式相关的Flag，以及android:taskAffinity的属性。

Intent与启动模式相关的Flag简介

这里仅仅对几个常用的与启动模式相关的Flag进行介绍。

FLAG_ACTIVITY_NEW_TASK

很少单独使用FLAG_ACTIVITY_NEW_TASK，通常与FLAG_ACTIVITY_CLEAR_TASK或FLAG_ACTIVITY_CLEAR_TOP联合使用。因为单独使用该属性会导致奇怪的现象，通常达不到我们想要的效果！尽管如何，后面还是会通过"FLAG_ACTIVITY_NEW_TASK示例一"和"FLAG_ACTIVITY_NEW_TASK示例二"会向你展示单独使用它的效果。

FLAG_ACTIVITY_SINGLE_TOP

在google的官方文档中介绍，它与launchMode="singleTop"具有相同的行为。实际上，的确如此！单独的使用FLAG_ACTIVITY_SINGLE_TOP，就能达到和launchMode="singleTop"一样的效果。

FLAG_ACTIVITY_CLEAR_TOP

顾名思义，FLAG_ACTIVITY_CLEAR_TOP的作用清除"包含Activity的task"中位于该Activity实例之上的其他Activity实例。FLAG_ACTIVITY_CLEAR_TOP和FLAG_ACTIVITY_NEW_TASK两者同时使用，就能达到和launchMode="singleTask"一样的效果！

FLAG_ACTIVITY_CLEAR_TASK

FLAG_ACTIVITY_CLEAR_TASK的作用包含Activity的task。使用FLAG_ACTIVITY_CLEAR_TASK时，通常会包含FLAG_ACTIVITY_NEW_TASK。这样做的目的是启动Activity时，清除之前已经存在的Activity实例所在的task；这自然也就清除了之前存在的Activity实例！ 注意：当同时使用launchMode和上面的FLAG_ACTIVITY_NEW_TASK等标签时，以FLAG_ACTIVITY_NEW_TASK为标准。也就是说，代码的优先级比manifest中配置文件的优先级更高！ 下面，通过几个实例加深对这几个标记的理解。

FLAG_ACTIVITY_NEW_TASK标签

配置形式：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.open.android.task5">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
```

```
        <activity android:name="SecondActivity" />
    </application>

</manifest>
```

说明：

在该实例中，有两个MainActivity和SecondActivity。

使用案例：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.button).
                setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, SecondActivity.class);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
    }
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);

        Log.i("maweiqi", "onNewIntent: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());

    }
}
```

注意，跳转的Intent添加了FLAG_ACTIVITY_NEW_TASK标志。

```
public class SecondActivity extends AppCompatActivity {

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Button button = (Button) findViewById(R.id.button2);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(SecondActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
    }
```

```
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);

        Log.i("maweiqi", "onNewIntent: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());

    }
}
```

测试方式：

MainActivity--> SecondActivity --> MainActivity--> SecondActivity

输出的日志如下：

```
onCreate: MainActivity TaskId: 73 hasCode:195694529
onCreate: SecondActivity TaskId: 73 hasCode:54326677
onCreate: MainActivity TaskId: 73 hasCode:121740648
onCreate: SecondActivity TaskId: 73 hasCode:5761837
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
    TaskRecord{e3d17e2 #73 A=com.open.android.task5 U=0 sz=4}
       Run #3: ActivityRecord{a34fa00 u0 com.open.android.task5/.SecondActivity t73}
       Run #2: ActivityRecord{21172da u0 com.open.android.task5/.MainActivity t73}
       Run #1: ActivityRecord{b1e1e64 u0 com.open.android.task5/.SecondActivity t73}
       Run #0: ActivityRecord{f015a6e u0 com.open.android.task5/.MainActivity t73}

    mResumedActivity: ActivityRecord{a34fa00 u0 com.open.android.task5/.SecondActivity t73}
```

测试结果：

1）全部在同一个task中

2）全部创建不同的实例

那如果MainActivity跳转去掉FLAG_ACTIVITY_NEW_TASK？在SecondActivity清单文件添加singleTask，代码和流程跟之前一样，只贴出清单文件配置

配置如下：

```
<activity android:name="SecondActivity"
            android:launchMode="singleTask"/>
```

测试方式：

MainActivity--> SecondActivity --> MainActivity--> SecondActivity

输出的日志如下：

```
onCreate: MainActivity TaskId: 74 hasCode:195694529
```

```
onCreate: SecondActivity TaskId: 74 hasCode:54326677
onCreate: MainActivity TaskId: 74 hasCode:38155814
onNewIntent: SecondActivity TaskId: 74 hasCode:54326677
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
   TaskRecord{46d0de2 #74 A=com.open.android.task5 U=0 sz=2}
      Run #1: ActivityRecord{b31602f u0 com.open.android.task5/.SecondActivity t74}
      Run #0: ActivityRecord{f88d9dc u0 com.open.android.task5/.MainActivity t74}

    mResumedActivity: ActivityRecord{b31602f u0 com.open.android.task5/.SecondActivity t74}
```

测试结果：

1）FLAG_ACTIVITY_NEW_TASK的作用和singleTask具有不同的效果

那如果两个Activity的android:taskAffinity不相同呢？此时会导致什么效果呢？下面，我们通过示例来看看效果。

配置如下：

```
<activity android:name="SecondActivity"
                android:taskAffinity="com.maweiqi.second"/>
```

代码回到最初MainActivity还是添加Flag标记，其他省略，如下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.button).
                setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, SecondActivity.class);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        Log.i("maweiqi", "onCreate: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
    }
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);

        Log.i("maweiqi", "onNewIntent: " + getClass().getSimpleName() +
                " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());

    }
}
```

测试方式：

MainActivity--> SecondActivity --> MainActivity--> SecondActivity

输出的日志如下：

```
onCreate: MainActivity TaskId: 77 hasCode:195694529
onCreate: SecondActivity TaskId: 78 hasCode:54326677
onCreate: MainActivity TaskId: 78 hasCode:38155814
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
    TaskRecord{65dda1d #78 A=com.maweiqi.second U=0 sz=2}
       Run #2: ActivityRecord{845c9ed u0 com.open.android.task5/.MainActivity t78}
       Run #1: ActivityRecord{f813328 u0 com.open.android.task5/.SecondActivity t78}
     TaskRecord{dd8e492 #77 A=com.open.android.task5 U=0 sz=1}
       Run #0: ActivityRecord{7a85f99 u0 com.open.android.task5/.MainActivity t77}

    mResumedActivity: ActivityRecord{845c9ed u0 com.open.android.task5/.MainActivity t78}
```

测试结果：

1）当第二次进入到MainActivity中，再企图从MainActivity中进入到SecondActivity时，没有产生任何效果，仍然停留在MainActivity中！即第二次MainActivity--> SecondActivity压根就没发生！

结果分析：

1）当相互跳转的两个Activity的android:taskAffinity不同时，添加FLAG_ACTIVITY_NEW_TASK：第一次启动SecondActivity时，会新建一个task，并将SecondActivity添加到该task中。这与singleTask产生的效果是一样的！但是，当企图再次从MainActivity进入到SecondActivity时，却什么也没有发生！

2）为什么呢？是因为此时SecondActivity实例已经存在，但是它所在的task的栈顶是MainActivity；而单独的添加FLAG_ACTIVITY_NEW_TASK又不会"删除task中位于SecondActivity之上的Activity实例"，所以就没有发生跳转！

FLAG_ACTIVITY_CLEAR_TOP。

修改MainActivity里面的点击事件如下：

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                        | Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

```
<activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
<activity android:name="SecondActivity"/>
```

测试方式：

MainActivity--> SecondActivity --> MainActivity--> SecondActivity

日志输出

```
onCreate: MainActivity TaskId: 80 hasCode:58656936
onCreate: SecondActivity TaskId: 80 hasCode:212434303
onCreate: MainActivity TaskId: 80 hasCode:121740648
onCreate: SecondActivity TaskId: 80 hasCode:15009890
```

测试结果：

1） MainActivity和SecondActivity在同一个task中！

2） 两个SecondActivity是不同的实例。

结果分析：

这与没有添加FLAG_ACTIVITY_CLEAR_TOP时效果一样！这说明，当相互跳转的两个Activity的android:taskAffinity一样时，不会产生任何效果！

接下来，看看不同android:taskAffinity的情况，代码一致，清单文件修改

文件配置

```
<activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
<activity android:name="SecondActivity"
                  android:taskAffinity="com.maweiqi.second"/>
```

测试方式：

MainActivity--> SecondActivity --> MainActivity--> SecondActivity

日志输出

```
onCreate: MainActivity TaskId: 83 hasCode:195694529
onCreate: SecondActivity TaskId: 84 hasCode:54326677
onCreate: MainActivity TaskId: 84 hasCode:190178689
onCreate: SecondActivity TaskId: 84 hasCode:5761837
```

在命令行中执行以下命令 adb shell dumpsys activity ,有以下输出：

```
Running activities (most recent first):
   TaskRecord{12e942 #84 A=com.maweiqi.second U=0 sz=1}
      Run #1: ActivityRecord{e0674b5 u0 com.open.android.task5/.SecondActivity t84}
    TaskRecord{62b9d53 #83 A=com.open.android.task5 U=0 sz=1}
      Run #0: ActivityRecord{71ec08a u0 com.open.android.task5/.MainActivity t83}

   mResumedActivity: ActivityRecord{e0674b5 u0 com.open.android.task5/.SecondActivity t84}
```

测试结果：

1）MainActivity和SecondActivity在不同task中！

2） 两个SecondActivity是不同实例。

结果分析：

FLAG_ACTIVITY_NEW_TASK和FLAG_ACTIVITY_CLEAR_TOP的使用和android:taskAffinity相关。 在同时使用FLAG_ACTIVITY_NEW_TASK|FLAG_ACTIVITY_CLEAR_TOP的情况下，以A启动B来说

1）当A和B的taskAffinity相同时：添加FLAG_ACTIVITY_NEW_TASK|FLAG_ACTIVITY_CLEAR_TOP没有任何作用。和没有添加时的效果一样！

2）当A和B的taskAffinity不同时：添加FLAG_ACTIVITY_NEW_TASK|FLAG_ACTIVITY_CLEAR_TOP后，如果该activity没有设置启动模式（即默认是standard），或者intent没有设置一个FLAG_ACTIVITY_SINGLE_TOP 的flag，那么这个activity就会销毁，然后重新创建这个实例

总结：

Activity的启动模式

在AndroidManifest.xml中，可以配置每个activity的启动模式：例如： android:launchMode="standard"

（1） standard 标准模式

此模式，不管有没有已存在的实例，都生成新的实例。每次调用startActivity()启动Activity时都会创建一个新的Activity放在栈顶，每次返回都会销毁实例并出栈，可以重复创建。

（2） singletop 单一顶部模式

如果任务栈的栈顶存在这个要开启的activity，不会重新创建新的activity，而是复用已存在的activity。保证栈顶如果存在，则不会重复创建，但如果不在栈顶，那么还是会创建新的实例。 应用场景：浏览器的书签

（3） singletask 单一任务模式

是一个比较严格的模式，在当前任务栈里面只能有一个实例存在，当开启activity的时候，就去检查在任务栈里面是否有实例已经存在，如果有实例存在就复用这个已经存在的activity，并且把这个activity上面的所有的别的activity都清空，复用这个已经存在的activity。 应用场景：BrowserActivity 浏览器界面 如果一个activity的创建需要占用大量的系统资源（cpu，内存）一般配置这个activity为singletask的启动模式。webkit内核(c) 初始化需要大量内存 js解析引擎 html渲染引擎 http解析，下载…减少内存开销，cpu占用。播放器的播放Activity

（4） singleInstance

这种启动模式比较特殊，它会启用一个新的任务栈，activity会运行在自己的任务栈里，这个任务栈里面只有一个实例存在并且保证不再有其他Activity实例进入。在整个手机操作系统里面只有一个实例存在。 应用场景：来电页面。InCallScreenActivity

最后的总结：

实话说，关于任务栈的bug有点多，如果真的添加了启动模式，一定要多多测试，有的地方和官方文档描述完全不相符。 stackoverflow上也有人吐槽。 http://stackoverflow.com/questions/9772927/flag-activity-new-task-clarification-needed

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态
- 深刻剖析activity启动模式(一)
- 深刻剖析activity启动模式(二)
- 深刻剖析activity启动模式(三)
- Activity Task和Process之间的关系
- service里面startActivity抛异常？activity不会

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

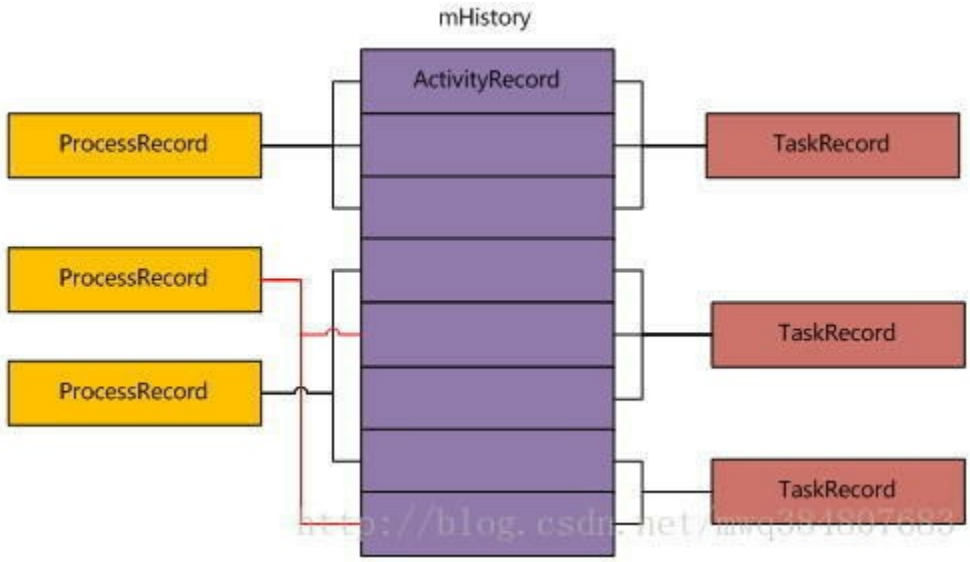
与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案
- 字体适配
- 软键盘顶出去解决方案与人事相关面试题
- 人事面试宝典

AMS提供了一个ArrayList mHistory来管理所有的activity，activity在AMS中的形式是ActivityRecord，task在AMS中的形式为TaskRecord，进程在AMS中的管理形式为ProcessRecord。如下图所示



从图中我们可以看出如下几点规则：

1) 所有的ActivityRecord会被存储在mHistory管理；

2) 每个ActivityRecord会对应到一个TaskRecord，并且有着相同TaskRecord的ActivityRecord在mHistory中会处在连续的位置；

3) 同一个TaskRecord的Activity可能分别处于不同的进程中，每个Activity所处的进程跟task没有关系；

Activity启动时ActivityManagerService会为其生成对应的ActivityRecord记录，并将其加入到回退栈(back stack)中，另外也会将ActivityRecord记录加入到某个Task中。请记住，ActivityRecord，backstack，Task都是ActivityManagerService的对象，由ActivityManagerService进程负责维护，而不是由应用进程维护。

在回退栈里属于同一个task的ActivityRecord会放在一起，也会形成栈的结构，也就是说后启动的Activity对应的ActivityRecord会放在task的栈顶

执行adb shell dumpsys activity命令，发现有以下输出：

```
Task id #30
  TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
    Intent { flg=0x10000000 cmp=com.open.android.task1/.SecondActivity }
     Hist #1: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
       Intent { flg=0x10400000 cmp=com.open.android.task3/.OtherActivity }
       ProcessRecord{12090b5 27543:com.open.android.task3/u0a62}
      Hist #0: ActivityRecord{1048af6 u0 com.open.android.task1/.SecondActivity t30}
       Intent { flg=0x10000000 cmp=com.open.android.task1/.SecondActivity }
       ProcessRecord{5bc013e 26035:com.open.android.task1/u0a59}

 Task id #31
   TaskRecord{dce52bb #31 A=com.open.android.task3 U=0 sz=1}
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task3/.MainActivity }
        Hist #0: ActivityRecord{f9e58c5 u0 com.open.android.task3/.MainActivity t31}
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task3/.MainActivity }
          ProcessRecord{12090b5 27543:com.open.android.task3/u0a62}

 Task id #29
      TaskRecord{5b063d8 #29 A=com.open.android.task1 U=0 sz=1}
```

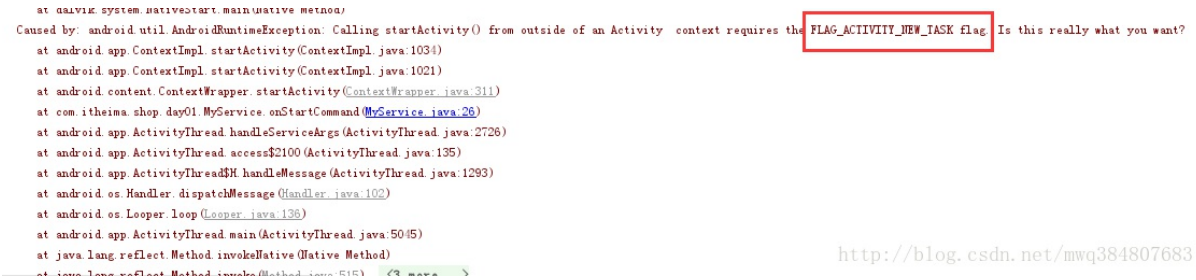
```
      Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.andr
oid.task1/.MainActivity }
        Hist #0: ActivityRecord{689947d u0 com.open.android.task1/.MainActivity t29}
          Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.open.
android.task1/.MainActivity }
          ProcessRecord{5bc013e 26035:com.open.android.task1/u0a59}

Running activities (most recent first):
   TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
        Run #3: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
   TaskRecord{dce52bb #31 A=com.open.android.task3 U=0 sz=1}
        Run #2: ActivityRecord{f9e58c5 u0 com.open.android.task3/.MainActivity t31}
   TaskRecord{7f2f34a #30 A=com.maweiqi.second U=0 sz=2}
       Run #1: ActivityRecord{1048af6 u0 com.open.android.task1/.SecondActivity t30}
   TaskRecord{5b063d8 #29 A=com.open.android.task1 U=0 sz=1}
       Run #0: ActivityRecord{689947d u0 com.open.android.task1/.MainActivity t29}

mResumedActivity: ActivityRecord{a0f9ded u0 com.open.android.task3/.OtherActivity t30}
```

从图对应文字在对应adb输出，这几个基本概念大家估计就清楚了。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

Android面试题-源码分析为什么service里面startActivity抛异常？activity不会

我们有时候需要在service里面启动activity，但是会发现报如下异常：



必须添加FLAG_ACTIVITY_NEW_TASK这个标记就可以了，那么为什么在activity里面不需要呢？接下来通过从源码角度带大家分析。

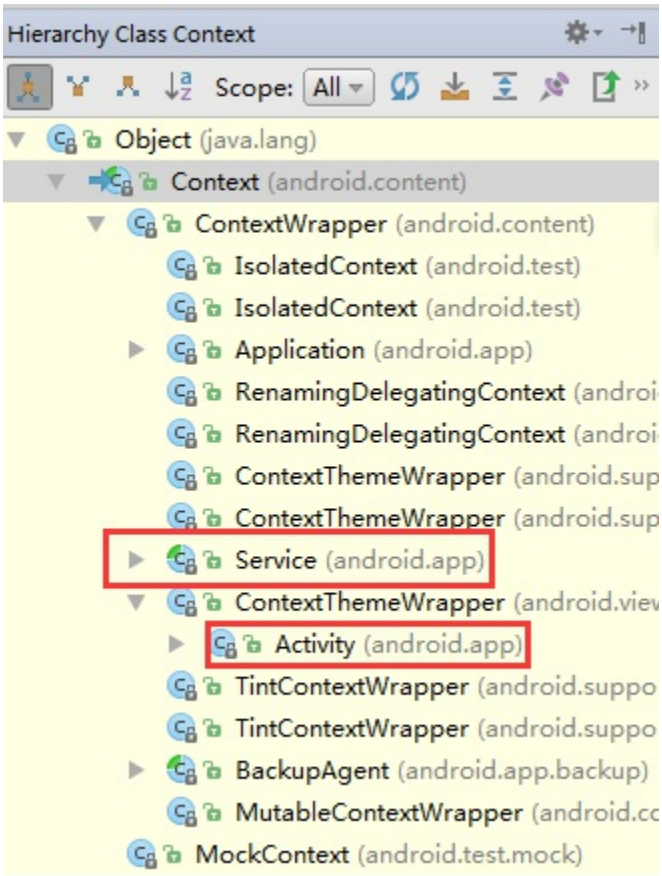
启动activity有两种形式

1）直接调用Context类的startActivity方法；这种方式启动的Activity没有Activity栈，因此不能以standard方式启动，必须加上FLAG_ACTIVITY_NEW_TASK这个Flag,服务就是通过Context调用。

2）调用被Activity类重载过的startActivity方法，通常在我们的Activity中直接调用这个方法就是这种形式；

Context.startActivity源码分析

我们查看Context类的startActivity方法，发现这竟然是一个抽象类；查看Context的类继承关系图如下：



我们看到诸如Activity，Service等并没有直接继承Context，Activity继承了ContextThemeWrapper，Service而是继承了ContextWrapper；

现在从源码分析ContextWrapper的实现：

```
@Override
public void startActivity(Intent intent) {
        mBase.startActivity(intent);
}
```

这个mBase是什么呢？这里我先直接告诉你，它的真正实现是ContextImpl类；至于为什么，有一条思路：在任意mBase打一个断点就能看到实现。

Context.startActivity最终使用了ContextImpl里面的方法，代码如下：

```
 @Override
    public void startActivity(Intent intent, Bundle options) {
        warnIfCallingFromSystemProcess();
        if ((intent.getFlags()&Intent.FLAG_ACTIVITY_NEW_TASK) == 0) {
            throw new AndroidRuntimeException(
                    "Calling startActivity() from outside of an Activity "
                    + " context requires the FLAG_ACTIVITY_NEW_TASK flag."
                    + " Is this really what you want?");
        }
        mMainThread.getInstrumentation().execStartActivity(
                getOuterContext(), mMainThread.getApplicationThread(), null,
                (Activity) null, intent, -1, options);
    }
```

源码分析：

1）大家看看抛出来的异常是不是还是熟悉的味道。

2）通过判断可知当前的intent.getFlags是否带有FLAG_ACTIVITY_NEW_TASK这个标记，如果没有抛出异常，因为源码使用了&运算符,只有两个位都是1，结果才是1，所以可知service没有带FLAG_ACTIVITY_NEW_TASK标记，才抛出异常。

3）真正的startActivity使用了Instrumentation类的execStartActivity方法；继续跟踪：

```
 public ActivityResult execStartActivity(
     Context who, IBinder contextThread, IBinder token, Activity target,Intent intent, int requestCode, Bundle option
s) {
      ......
try {
     ......
     int result = ActivityManagerNative.getDefault()
                .startActivity(whoThread, who.getBasePackageName(), intent,
                   intent.resolveTypeIfNeeded(who.getContentResolver()),
                        token, target != null ? target.mEmbeddedID : null,
                        requestCode, 0, null, options);
            checkStartActivityResult(result, intent);
        } catch (RemoteException e) {
            throw new RuntimeException("Failure from system", e);
        }
        return null;
```

源码分析：

1）到这里我们发现真正调用的是ActivityManagerNative的startActivity方法；

Activity.startActivity源码分析

```
@Override
public void startActivity(Intent intent) {
        this.startActivity(intent, null);
}
```

源码可知：

1）调用当前类的startActivity方法，代码如下：

```
@Override
public void startActivity(Intent intent, @Nullable Bundle options) {
        if (options != null) {
            startActivityForResult(intent, -1, options);
        } else {
            // Note we want to go through this call for compatibility with
            // applications that may have overridden the method.
            startActivityForResult(intent, -1);
        }
    }
```

源码可知

1）调用了startActivityForResult

```
public void startActivityForResult(Intent intent, int requestCode, @Nullable Bundle options) {
   ......
   Instrumentation.ActivityResult ar =
          mInstrumentation.execStartActivity(
          this, mMainThread.getApplicationThread(), mToken, this,
                    intent, requestCode, options);
    ......
```

源码可知

1）可以看到，其实通过Activity和ContextImpl类启动Activity并无本质不同，他们都通过Instrumentation这个辅助类调用到了ActivityManagerNative的方法。

2）只是Activity不会去检查标记，所以并不会抛出异常。

至此源码分析完毕。

1. RxBus优雅式

首先，在基类BaseActivity里，注册RxBus监听：

```
public class BaseActivity3 extends AppCompatActivity {
    Subscription mSubscription;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        initRxBus();
    }
    //接收退出的指令，关闭所有activity
    private void initRxBus() {
        mSubscription = RxBus.getInstance().toObserverable(NormalEvent.class)
                .subscribe(new Action1<NormalEvent>() {
                               @Override
                               public void call(NormalEvent userEvent) {
                                   if (userEvent.getType() == -1) {
                                       finish();
                                   }
                               }
                           },
                        new Action1<Throwable>() {
                            @Override
                            public void call(Throwable throwable) {
                            }
                        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (!mSubscription.isUnsubscribed()) {
            mSubscription.unsubscribe();
        }
    }
}
```

这是事件实体NormalEvent:

```
public class NormalEvent {
    private int type;

    public NormalEvent(int type) {
        this.type = type;
    }

    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }
}
```

新建RxBus类

```
public class RxBus {
```

```
    private static volatile RxBus mInstance;

    private final Subject bus;


    public RxBus()
    {
        bus = new SerializedSubject<>(PublishSubject.create());
    }

    /**
     * 单例模式RxBus
     *
     * @return
     */
    public static RxBus getInstance()
    {

        RxBus rxBus2 = mInstance;
        if (mInstance == null)
        {
            synchronized (RxBus.class)
            {
                rxBus2 = mInstance;
                if (mInstance == null)
                {
                    rxBus2 = new RxBus();
                    mInstance = rxBus2;
                }
            }
        }

        return rxBus2;
    }


    /**
     * 发送消息
     *
     * @param object
     */
    public void post(Object object)
    {

        bus.onNext(object);

    }

    /**
     * 接收消息
     *
     * @param eventType
     * @param <T>
     * @return
     */
    public <T> Observable<T> toObserverable(Class<T> eventType)
    {
        return bus.ofType(eventType);
    }
}
```

最后，在需要退出的地方调用：

```
RxBus.getInstance().post(new NormalEvent(-1));//发送退出指令
```

2. 容器式：

建立一个全局容器，把所有的Activity存储起来，退出时循环遍历finish所有Activity

```
public class BaseActivity extends AppCompatActivity {
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState ) {
        super.onCreate(savedInstanceState);
        ActivityManager.getActivityManager().addActivity(this);
    }
    @Override protected void onDestroy() {
        super.onDestroy();
       //  结束Activity&从栈中移除该Activity
        ActivityManager.getActivityManager().finishActivity();
    }

}



public class ActivityManager {
        // Activity栈
        private static Stack<Activity> activityStack;
        // 单例模式
        private static ActivityManager instance;

        private ActivityManager() {
        }

        /**
         * 单一实例
         */
        public static ActivityManager getActivityManager() {
            if (instance == null) {
                instance = new ActivityManager();
            }
            return instance;
        }

        /**
         * 添加Activity到堆栈
         */
        public void addActivity(Activity activity) {
            if (activityStack == null) {
                activityStack = new Stack<Activity>();
            }
            activityStack.add(activity);
        }

        /**
         * 获取当前Activity（堆栈中最后一个压入的）
         */
        public Activity currentActivity() {
            Activity activity = activityStack.lastElement();
            return activity;
        }

        /**
         * 结束当前Activity（堆栈中最后一个压入的）
         */
        public void finishActivity() {
```

```
            Activity activity = activityStack.lastElement();
            finishActivity(activity);
        }

        /**
         * 结束指定的Activity
         */
        public void finishActivity(Activity activity) {
            if (activity != null) {
                activityStack.remove(activity);
                activity.finish();
                activity = null;
            }
        }

        /**
         * 结束指定类名的Activity
         */
        public void finishActivity(Class<?> cls) {
            for (Activity activity : activityStack) {
                if (activity.getClass().equals(cls)) {
                    finishActivity(activity);
                }
            }
        }

        /**
         * 结束所有Activity
         */
        public void finishAllActivity() {
            for (int i = 0; i < activityStack.size(); i++) {
                if (null != activityStack.get(i)) {
                    activityStack.get(i).finish();
                }
            }
            activityStack.clear();
        }

        /**
         * 退出应用程序
         */
        public void AppExit(Context context) {
            try {
                finishAllActivity();
                //根据进程ID，杀死该进程
                android.os.Process.killProcess(android.os.Process.myPid());
                //退出真个应用程序
                System.exit(0);
            } catch (Exception e) {
            }
        }

}
```

3. 广播式

通过在BaseActivity中注册一个广播，当退出时发送一个广播，finish退出

```
public class BaseActivity2 extends AppCompatActivity {
    private static final String EXITACTION = "action.exit";
    private ExitReceiver exitReceiver = new ExitReceiver();
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        IntentFilter filter = new IntentFilter();
```

```
        filter.addAction(EXITACTION);
        registerReceiver(exitReceiver, filter);
    }
    @Override protected void onDestroy() {
        super.onDestroy(); unregisterReceiver(exitReceiver);
    }
    class ExitReceiver extends BroadcastReceiver {
        @Override public void onReceive(Context context, Intent intent) {
            BaseActivity2.this.finish();
        }
    }

}
```

4. SingleTask

1、设置MainActivity的加载模式为singleTask

```
android:launchMode="singleTask"
```

2、将退出出口放置在MainActivity

```
private boolean mIsExit;
    @Override /** * 双击返回键退出 */
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            if (mIsExit) {
                this.finish();
            } else {
                Toast.makeText(this, "再按一次退出", Toast.LENGTH_SHORT).show();
                mIsExit = true;
                new Handler().postDelayed(new Runnable() {

                    @Override public void run() {
                        mIsExit = false;
                    }
                }, 2000);
            } return true;
        } return super.onKeyDown(keyCode, event);
    }
```

5. SingleTask改版式

第一步设置MainActivity的加载模式为singleTask

```
android:launchMode="singleTask"
```

第二步重写onNewIntent()方法

```
private static final String TAG_EXIT = "exit";
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        if (intent != null) {
            boolean isExit = intent.getBooleanExtra(TAG_EXIT, false);
            if (isExit) { this.finish();
            }
        }
```

```
    }
```

第三步 退出

```
Intent intent = new Intent(this,MainActivity.class); intent.putExtra(MainActivity.TAG_EXIT, true);
startActivity(intent);
```

Activity扮演了一个界面展示的角色，堪称四大组件之首，onCreate是Activity的执行入口，都不知道入口到底干了嘛，还学什么android,所以本文会从源码的角度对其进行分析。

熟悉源码的会发现，真正启动Activity的实现都在ActivityThread，前面的调用过程略过

ActivityThread的方法performLaunchActivity中调用了Instrumentation类中的方法callActivityOnCreate方法，继而调用了TargetActivity中的onCreate方法。

```
private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {
......
Activity activity = null;
activity = mInstrumentation.newActivity( cl, component.getClassName(), r.intent);
......
if (r.isPersistable()) {
     mInstrumentation.callActivityOnCreate(activity, r.state, r.persistentState);
} else {
     mInstrumentation.callActivityOnCreate(activity, r.state);
}
......
}
```

源码可知：

1）通过反射的机制创建的Activity

2）这里的mInstrumentation是类Instrumentation

3）Instrumentation类中的方法callActivityOnCreate方法源码如下：

```
public void callActivityOnCreate(Activity activity, Bundle icicle) {
        prePerformCreate(activity);
        activity.performCreate(icicle);
        postPerformCreate(activity);
    }
```

源码可知：

1）activity.performCreate(icicle)，其中的方法是通过activity，这个activity，形如：Activity activity = 子Activity的对象

2）在Activity类中的方法performCreate(icicle)，源码如下：

```
final void performCreate(Bundle icicle) {
        onCreate(icicle);
        mActivityTransitionState.readState(icicle);
        performCreateCommon();
}
```

源码可知：

1）原来onCreate的生命周期方法是在这里回调的

2）在performCreate方法中调用的onCreate方法是MainActivity中的onCreate方法，那么到此MainActivity中的方法onCreate方法中的参数Bundle savedInstanceState也就知道来源了，此时，MainActivity中的方法也就被调用了。

再次看MainActivity中的方法onCreate:

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
```

super.onCreate(savedInstanceState)，其实这条语句放在子类中的onCreate方法中的任何位置都可，问题只是super.onCreate(savedInstanceState)必须要被执行，所以，最好也就是放在第一行，看起来比较明确，至于为什么，参考onSaveInstanceState源码分析 至此onCreate源码分析完毕。

Service相关面试题

- IntentService源码分析
- Service是否在main thread中执行, service里面是否能执行耗时的操作?
- Android面试题-Service不死之身

IntentService是继承于Service并处理异步请求的一个类，在IntentService内有一个工作线程来处理耗时操作，启动IntentService的方式和启动传统Service一样，同时，当任务执行完后，IntentService会自动停止，而不需要我们去手动控制。另外，可以启动IntentService多次，而每一个耗时操作会以工作队列的方式在IntentService的onHandleIntent回调方法中执行，并且，每次只会执行一个工作线程，执行完第一个再执行第二个，以此类推。

而且，所有请求都在一个单线程中，不会阻塞应用程序的主线程（UI Thread），同一时间只处理一个请求。

IntentService有什么好处呢?

1）我们省去了在Service中手动开线程的麻烦，

2）当操作完成时，我们不用手动停止Service。

接下来让我们来看看如何使用，写一个Demo来模拟两个耗时操作，Operation1与Operation2，先执行1，2必须等1执行完才能执行2:

新建工程，新建一个继承IntentService的类，我这里是IntentServiceDemo.java

```
public class IntentServiceDemo extends IntentService {

    public IntentServiceDemo() {
        //必须实现父类的构造方法
        super("IntentServiceDemo");
    }

    @Override
    public IBinder onBind(Intent intent) {
        System.out.println("onBind");
        return super.onBind(intent);
    }


    @Override
    public void onCreate() {
        System.out.println("onCreate");
        super.onCreate();
    }

    @Override
    public void onStart(Intent intent, int startId) {
        System.out.println("onStart");
        super.onStart(intent, startId);
    }


    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        System.out.println("onStartCommand");
        return super.onStartCommand(intent, flags, startId);
    }


    @Override
    public void setIntentRedelivery(boolean enabled) {
        super.setIntentRedelivery(enabled);
        System.out.println("setIntentRedelivery");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //Intent是从Activity发过来的，携带识别参数，根据参数不同执行不同的任务
        System.out.println("currentThread()=" + Thread.currentThread().getName());
```

```
        String action = intent.getExtras().getString("param");
        if (action.equals("oper1")) {
            System.out.println("Operation1");
        }else if (action.equals("oper2")) {
            System.out.println("Operation2");
        }

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onDestroy() {
        System.out.println("onDestroy");
        super.onDestroy();
    }

}
```

我把生命周期方法全打印出来了，待会我们来看看它执行的过程是怎样的。接下来是Activity，在Activity中来启动IntentService：

```
public class TestActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //可以启动多次，每启动一次，就会新建一个work thread，但IntentService的实例始终只有一个
        //Operation 1
        Intent startServiceIntent = new Intent("com.test.intentservice");
        Bundle bundle = new Bundle();
        bundle.putString("param", "oper1");
        startServiceIntent.putExtras(bundle);
        startService(startServiceIntent);

        //Operation 2
        Intent startServiceIntent2 = new Intent("com.test.intentservice");
        Bundle bundle2 = new Bundle();
        bundle2.putString("param", "oper2");
        startServiceIntent2.putExtras(bundle2);
        startService(startServiceIntent2);
    }
}
```

最后，别忘了配置Service，因为它继承于Service，所以，它还是一个Service，一定要配置，否则是不起作用的

```
<service android:name=".IntentServiceDemo">
      <intent-filter >
          <action android:name="com.test.intentservice"/>
      </intent-filter>
</service>
```

最后来看看执行结果:

tag	Message
System.out	onCreate
System.out	onStartCommand
System.out	onStart
System.out	onStartCommand
System.out	onStart
System.out	Operation 1
System.out	Operation 2
System.out	onDestroy

从结果可以看到，onCreate方法只执行了一次，而onStartCommand和onStart方法执行了两次，开启了两个Work Thread，这就证实了之前所说的，启动多次，但IntentService的实例只有一个，这跟传统的Service是一样的。Operation1也是先于Operation2打印，并且我让两个操作间停顿了2s，最后是onDestroy销毁了IntentService。

IntentService 源码分析

```
@Override
public void onCreate() {
        super.onCreate();
        HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");
        thread.start();
        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

源码可知：

1）实际上是使用了一个 HandlerThread 来维护线程的，

2） HandleThread 中，内部已经维护一个 Looper，这里直接使用 HandlerThread 的 Looper 对象，便于在 IntentService 中去维护消息队列，

3）创建的 mServiceHandler 是属于 HandleThread 这个 WorkerThread 的。

```
private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }

        @Override
        public void handleMessage(Message msg) {
            onHandleIntent((Intent)msg.obj);
            stopSelf(msg.arg1);
        }
    }
```

源码可知：

1）直接把消息交给 onHandleIntent() 方法去执行具体的业务逻辑

2）执行完成之后，立即调用 stopSelf() 方法停止自己

接下来分析start源码

```
 @Override
 public void onStart(Intent intent, int startId) {
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        msg.obj = intent;
```

```
        mServiceHandler.sendMessage(msg);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        onStart(intent, startId);
        return mRedelivery ? START_REDELIVER_INTENT : START_NOT_STICKY;
      }
```

源码可知

1）在 onStartCommand() 中直接调用了 onStart() 方法

2）而上面 stopSelf() 方法使用的 startId 来停止当前的此次任务服务。

3）而 Service 如果被启动多次，就会存在多个 startId ， 当所有的 startId 都被停止之后，才会调用 onDestory() 自我销毁。

我们在看看HandlerThread启动之后的源码

```
@Override
public void run() {
        mTid = Process.myTid();
        Looper.prepare();
        synchronized (this) {
            mLooper = Looper.myLooper();
            notifyAll();
        }
        Process.setThreadPriority(mPriority);
        onLooperPrepared();
        Looper.loop();
        mTid = -1;
    }
```

源码可知

run方法里面添加了锁，这也解释了为什么多次 start 同一个 IntentService 它会顺序执行，全部执行完成之后，再自我销毁。

Android面试题-Service是否在main thread中执行, service里面是否能执行耗时的操作?

Service不是独立的进程，也不是独立的线程，它是依赖于应用程序的主线程的，也就是说，在更多时候不建议在Service中编写耗时的逻辑和操作（比如:网络请求，拷贝数据库，大文件），否则会引起ANR。

如果想在服务中执行耗时的任务。有以下解决方案:

1） 在service中开启一个子线程

```
new Thread(){}.start();
```

2） 可以使用IntentService异步管理服务 参考文章IntentService的使用:
http://blog.csdn.net/mwq384807683/article/details/72549222

Service 和 Activity 在同一个线程，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)

1. 在onStartCommand方法中将flag设置为START_STICKY;

```
return Service.START_STICKY;
```

2. 在xml中设置了android:priority

```
<!--设置服务的优先级为MAX_VALUE-->
<service android:name=".MyService"
         android:priority="2147483647"
         >
</service>
```

3. 在onStartCommand方法中设置为前台进程

```
@Override
   public int onStartCommand(Intent intent, int flags, int startId) {
     Notification notification = new Notification(R.mipmap.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);
       RemoteViews remoteView = new RemoteViews(this.getPackageName(),R.layout.notification);
       remoteView.setImageViewResource(R.id.image, R.mipmap.ic_launcher);
       remoteView.setTextViewText(R.id.text , "Hello,this message is in a custom expanded view");
       notification.contentView = remoteView;
       notification.contentIntent = pendingIntent;
       startForeground(1, notification);
       return Service.START_STICKY;

   }
```

4. 在onDestroy方法中重启service

```
@Override
public void onDestroy() {
        super.onDestroy();
        startService(new Intent(this, MyService.class));
}
```

5. 用AlarmManager.setRepeating(...)方法循环发送闹钟广播,接收的时候调用service的onstart方法

```
Intent intent = new Intent(MainActivity.this,MyAlarmReciver.class);
        PendingIntent sender = PendingIntent.getBroadcast( MainActivity.this, 0, intent, 0);

        // We want the alarm to go off 10 seconds from now.
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(System.currentTimeMillis());
        calendar.add(Calendar.SECOND, 1);
        AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
        //重复闹钟
        /**
         *  @param type
         * @param triggerAtMillis t 闹钟的第一次执行时间，以毫秒为单位
         * go off, using the appropriate clock (depending on the alarm type).
         * @param intervalMillis 表示两次闹钟执行的间隔时间，也是以毫秒为单位
```

```
         * of the alarm.
         * @param operation 绑定了闹钟的执行动作，比如发送一个广播、给出提示等等
         */
        am.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), 2 * 1000, sender);
```

6. 目前市场面的很多三方的消息推送SDK唤醒APP,例如Jpush

总结

这纯粹是面试的时候忽悠一下面试官，不代表着你的Service就永生不死了，只能说是提高了进程的优先级。迄今为止我没有发现能够通过常规方法达到流氓需求(通过长按home键清除都清除不掉)的方法，目前所有方法都是指通过Android的内存回收机制和普通的第三方内存清除等手段后仍然保持运行的方法，有些手机厂商把这些知名的app放入了自己的白名单中，保证了进程不死来提高用户体验（如微信、QQ、陌陌都在小米的白名单中）。如果从白名单中移除，他们终究还是和普通app一样躲避不了被杀的命运。

网络编程

- Android客户端和服务端如何使用Token和Session
- 推送原理
- 阐述一下对XMPP协议理解以及优缺点?
- 简单阐述一下及时推送原理?

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 什么是XMPP配套视频

3-简单阐述一下对XMPP协议理解以及优缺点？

定义：

> XMPP（可扩展消息处理现场协议）的前身是Jabber，由一个开源组织产生的即时通信协议，基于可扩展标记语言（XML）的一种协议，是由IETF(国际互联网小组)通过的网络标准。

- 及时通讯:发送一则消息，对方账号马上即时接收到消息。中间没有出现延时。 底层使用 socket实现/java对tcp/ip协议的实现

基本网络结构

XMPP中定义了三个角色，客户端，服务器，网关。通信能够在这三者的任意两个之间双向发生。服务器同时承担了客户端信息记录，连接管理和信息的路由功能。网关承担着与异构即时通信系统的互联互通，异构系统可以包括SMS（短信），MSN，ICQ等。基本的网络形式是单客户端通过TCP/IP连接到单服务器，然后在之上传输XML。

- 举个例子看看所谓的XML（标准通用标记语言的子集）流是什么样子的?

客户端:

```
<?xmlversion='1.0'?>
<stream:stream
    to='example_com'
    xmlns='jabber:client'
    xmlns:stream='http_etherx_jabber_org/streams'
    version='1.0'>
```

服务器:

```
<?xmlversion='1.0'?>
<stream:stream
    from='example_com'
    id='someid'
    xmlns='jabber:client'
    xmlns:stream='http_etherx_jabber_org/streams'
    version='1.0'>
```

其他通信 客户端:

```
<messagefrom='juliet_example_com'
    to='romeo_example_net'
    xml:lang='zh-cn'>
        <body>Art thou not Romeo, and a Montague?</body>
</message>
```

服务器:

```
<message from='romeo_example_net'
    to='juliet_example_com'
    xml:lang='zh-cn'>
    <body>Neither, fair saint, if either thee dislike.</body>
</message>
```

客户端:

```
</stream:stream>
```

服务器:

```
</stream:stream>
```

- 以文档的观点来看，客户端或服务器发送的所有XML文本连缀在一起，从<stream>到</stream>构成了一个完整的XML文档。其中的stream标签就是所谓的XML Stream。在<stream>与</stream>中间的那些<message>...</message>这样的XML元素就是所谓的XML Stanza（XML节）。 XMPP核心协议通信的基本模式就是先建立一个stream，然后协商一堆安全之类的东西，中间通信过程就是客户端发送XML Stanza，一个接一个的。服务器根据客户端发送的信息以及程序的逻辑，发送XML Stanza给客户端。但是这个过程并不是一问一答的， 任何时候都有可能从一方发信给另外一方。通信的最后阶段是</stream>关闭流，关闭TCP/IP连接。

> XMPP 的消息格式。 XMPP 协议的所有消息都是 XML 格式的，在 XMPP 中定义了 3 个顶层消息:

- 2.1 Presence

  用于确定用户的状态。消息结构举例如下（每个 XML 的 node 还会有很多其他 attribute，为了简单起见这里省略，下同）：

  ```
  <presence from="abc@jabber.org/contact"  to="def@jabber.org/contact">
    <status>online</status>
  </presence>
  ```

- 2.2 Message

  用于在两个用户之间发送消息。消息结构举例如下:

  ```
  <message from="abc@jabber.org/contact" to="def@jabber.org/contact" type=“chat”>
    <body>hello</body>
  </message>
  ```

- 2.3 IQ

  信息/请求，是一个请求－响应机制，管理XMPP服务器上两个用户的转换，允许他们通过相应的XML格式进行查询和响应。

  ```
  <iq from="abc@jabber.org/contact" id=“id11” type=“result”>
  </iq>
  ```

▪常用形式

直接通讯	Peer to Peer p2p 点对点　不经过服务器
在线代理	消息发送给对方 先经过[服务器] 服务器 根据目录账号进行转发
离线代理	消息发送给对方 对方不在线 先经过[服务器] 服务器 使用数据库 暂时保存.当对对方上线,根据目录账号进行转发
离线扩展	对方不在线时候，以其它方式通知用户

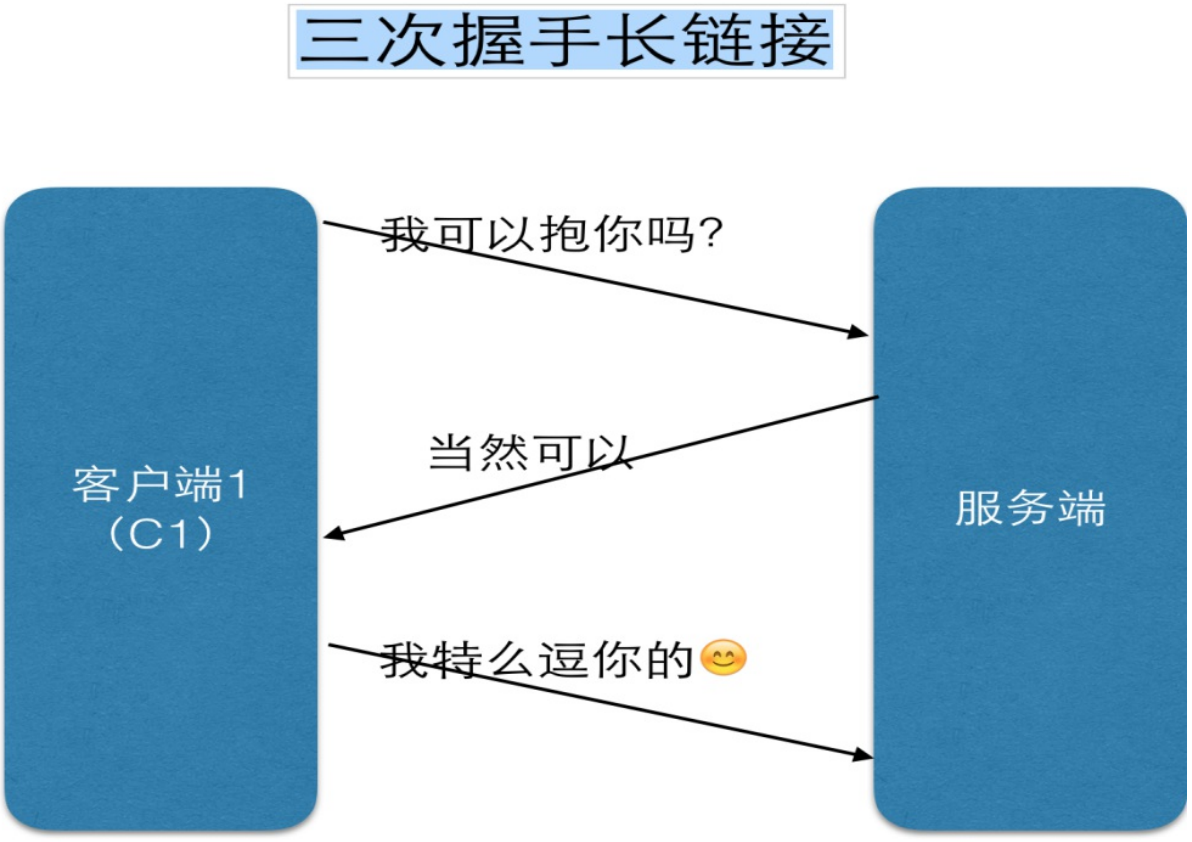
及时聊天的展示形式

Socket	Xmpp 开源
企业 自主研发 在线	企业 二次开发。 在线 离线。

- TCP/IP 与 UDP 差别

	TCP	UDP
面向连接	面向连接	不面向连接
可靠性	可靠	不可靠
数据量	大量数据	少量数据 64K
传输效率	慢	快
应用 由于网络提速	多数使用	少数
java	ServerSocket	DataprogramSocket

> 面向连接：三次握手



- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 极光推送原理配套视频
- xmpp基本概念配套视频
- 配套视频

4-简单阐述一下及时推送原理?

a) 传统获取服务器数据使用的是pull模式，是客户端向服务器请求数据。从客户端发起连接请求，获取到服务器数据后就关闭连接。当连接断开后，服务器就会失去客户端的地址，因此无法主动向客户端发送消息。

b) 推送(push)是服务主动向客户端发送数据。

它的原理是保持一个长连接，当客户端和服务器建立连接后不再断开，这样服务器随时有新消息都可以发送给客户端。

长连接和短连接。所谓长连接，指在一个TCP连接上可以连续发送多个数据包，在TCP连接保持期间，如果没有数据包发送，需要双方发检测包以维持此连接。（心跳连接）

```
while(true) {
    request(timeout);
    request(timeout);
}
```

短连接是指通信双方有数据交互时，就建立一个TCP连接，数据发送完成后，则断开此TCP连接，即每次TCP连接只完成一对

c) 至于如何获取推送消息。由于服务端传来推送消息的时间是不确定的，这里只能等待推送SDK的回调，比如通过注册监听或者广播接收者。不同的厂商的推送SDK可能会有不同的处理方案，以百度推送SDK来说，是通过广播接收者获取推送数据。

5-简要说明一下openfire、smack、spark

openfire是基于XMPP协议的即时通信的服务器端的一个实现，你不需要编写一行服务端的代码，实现一个简单的点对点通信或是简单的群聊.

smack 是XMPP传输协议的Java实现，提供了一套API接口，可以连接服务端。一般用于快速开发手机客户端。

spark是基于smack实现的一个XMPP即时通信客户端(PC端的)。

6-列出几种常见的解决消息即时获取方案

1. 轮询(Pull)方式：客户端定时向服务器发送询问消息，一旦服务器有变化则立即同步消息。
2. SMS(短信消息)(Push)方式：通过拦截SMS消息并且解析消息内容来了解服务器的命令，但这种方式一般用户在经济上很难承受。
3. 持久连接(Push)方式：客户端和服务器之间建立长久连接，这样就可以实现消息的及时行和实时性。

7-Timer比AlarmManager实现心跳时更耗电原因

- Timer

Android 的 Timer 类可以用来计划需要循环执行的任务。 Timer 的问题是它需要用 WakeLock 让 CPU 保持唤醒状态，这样会大量消耗手机电量， 大大减短手机待机时间。这种方式不能满足我们的需求。

- AlarmManager

AlarmManager 是 Android 系统封装的用于管理 RTC 的模块，RTC (Real Time Clock) 是一个独立的硬件时钟，可以在 CPU 休眠时正常运行，在预设的时间到达时，通过中断唤醒 CPU。这意味着，如果我们用 AlarmManager 来定时执行任务，CPU 可以正常的休眠，只有在需要运行任务时醒来一段很短的时间。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

与性能优化相关面试题

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

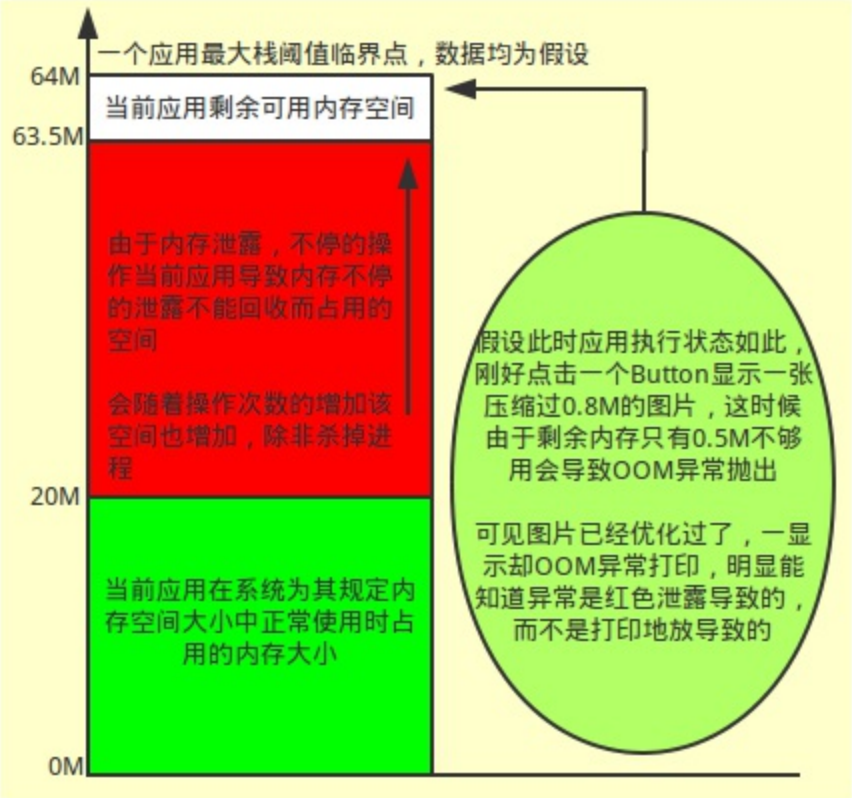
- 人事面试宝典

本文配套视频

- 配套视频
- 配套视频
- 配套视频

9-内存泄漏和内存溢出分别是什么？它们有什么关系？

- 内存泄露是指保存了不可能再被访问的变量引用，导致垃圾回收器无法回收内存。 也就是说：在Java中有些对象的生命周期是有限的，当它们完成了特定的逻辑后将会被垃圾回收；但是，如果在对象的生命周期本来该被垃圾回收时这个对象还被别的对象所持有引用，那就会导致内存泄漏
- 内存溢出是指虚拟机内存耗尽，无法为新对象分配内存，导致引用崩溃。典型的情况为加载多张大图，导致内存耗尽。



- 当某个界面存在内存泄露，反复进入该界面，将导致一直有新对象创建但是无法回收，最终内存耗尽，产生内存溢出。

10-什么情况下会导致内存泄漏

- 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。
- 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。
- 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。 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 mContextRef;
- 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。 针对这种线程导致的内存泄露问题的解决方案：
  - 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）。
  - 在线程内部采用弱引用保存Context引用。

- 查询数据库没有关闭cursor 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，只有在常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。
- 构造Adapter没有复用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。 为什么要复用convertView? 当convertView为空时，用setTag()方法为每个View绑定一个存放控件的ViewHolder对象。当convertView不为空，重复利用已经创建的view的时候，使用getTag()方法获取绑定的ViewHolder对象，这样就避免了findViewById对控件的层层查询，而是快速定位到控件。
- Bitmap不再使用时没有调用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。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 配套视频
- 配套视频
- 线程池原理配套视频

11-说一下如何对Android内存优化上集?

Bitmap OOM（图片优化）

1. 图片处理

a．等比缩放

```
        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。

b．对图片采用软引用，及时地进行recyle()操作

```
SoftReference<Bitmap> bitmap = new SoftReference<Bitmap>(pBitmap);
if(bitmap != null){
    if(bitmap.get() != null && !bitmap.get().isRecycled()){
        bitmap.get().recycle();
        bitmap = null;
    }
}
```

c．设置图片解码格式

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inPreferredConfig = Bitmap.Config.ARGB_4444;
```

d．加载部分图片

```
public void loadPartPic(){
   File file = new File("/mnt/sdcard/dog.jpg");
   try {
     FileInputStream inputStream = new FileInputStream(file);
     BitmapRegionDecoder bitmapRegionDecoder =
            BitmapRegionDecoder.newInstance(inputStream, false);
     //获得图片的宽、高
            BitmapFactory.Options tmpOptions = new BitmapFactory.Options();
     int width = tmpOptions.outWidth;
     int height = tmpOptions.outHeight;
     //设置显示图片的中心区域
     BitmapFactory.Options options = new BitmapFactory.Options();
     options.inPreferredConfig = Bitmap.Config.ARGB_4444;
     Bitmap bitmap = bitmapRegionDecoder.decodeRegion(new Rect(width / 2 - 100 ,
           height / 2 - 100,width / 2 + 100, height / 2 + 100), options);
     System.out.println("bitmap的大小:" + bitmap.getByteCount());
     mIv.setImageBitmap(bitmap);
    } catch (Exception e) {
        e.printStackTrace();
    }
```

2. 图片的缓存

a) 网络缓存 b) 内存缓存 c) 磁盘缓存

3. 使用加载图片框架处理图片，如专业处理加载图片的ImageLoader，glide，picasso等图片加载框架。

通过以上方式可以解决图片OOM异常

UI Review（UI优化）

减少视图层级 减少视图层级可以有效的减少内存消耗，因为视图是一个树形结构，每次刷新和渲染都会遍历一次。

ViewStub标签

此标签可以使UI在特殊情况下，直观效果类似于设置View的不可见性，但是其更大的意义在于被这个标签所包裹的Views在默认状态下不会占用任何内存空间。

```
<ViewStub
        android:id="@+id/mystub"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout="@layout/common_msg" >
 </ViewStub>
```

include标签

可以通过这个标签直接加载外部的xml到当前结构中，是复用UI资源的常用标签。

```
<include
        layout="@layout/activity_main" />
```

merge标签

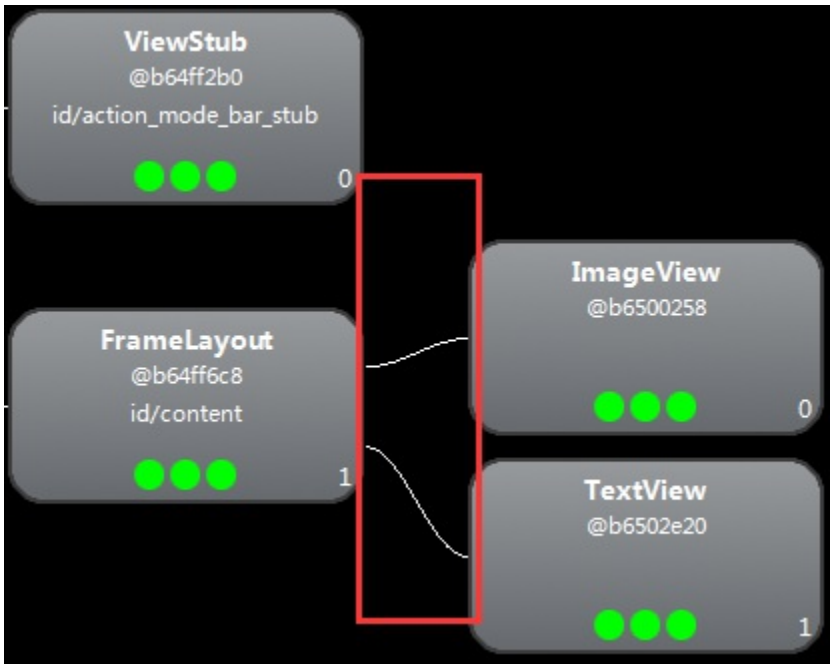
它在优化UI结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个Android Layout的结构。

```
<?xml version="1.0" encoding="utf-8"?>
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/ic_launcher"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:layout_marginTop="10dp"
        android:textSize="20sp"
        android:text="我是小黑马"
        />

</merge>
```

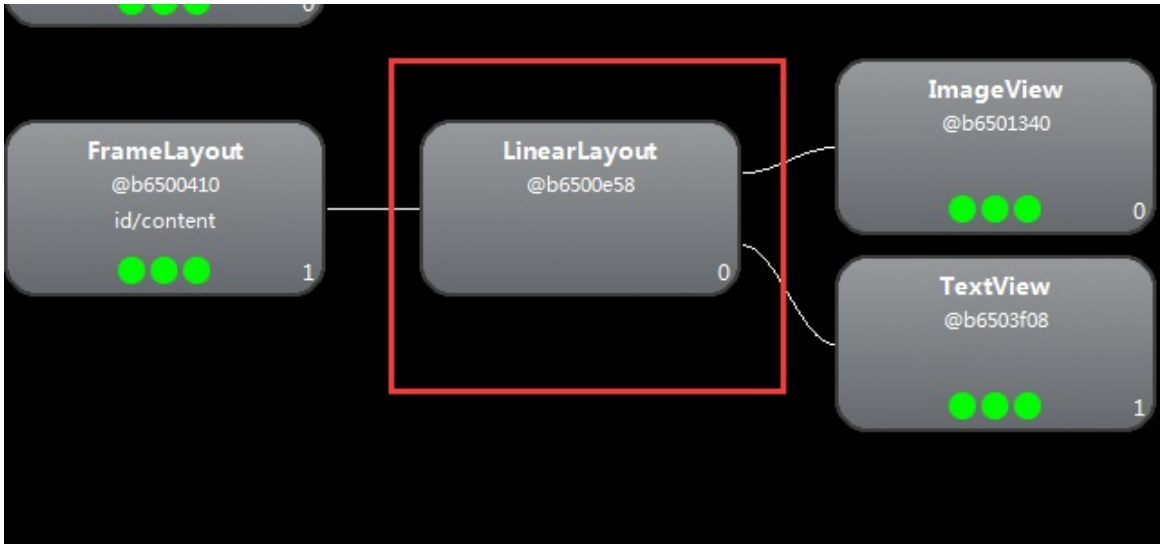
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/ic_launcher" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:layout_marginTop="10dp"
        android:text="我是小黑马"
        android:textSize="20sp" />

</LinearLayout>
```

实现一模一样的效果，我们通过SDK自带工具检测android层级关系

- 使用merge标签，布局少一个层级



- 不使用merge标签，布局多一个层级



Fragment优化 直接使用系统APP包下面的Fragment，不要使用V4包里面的Fragment可以减少层级结构。

使用ThreadPool优化代码（线程池实现原理）

1. new Thread的弊端

执行一个异步任务你还只是如下new Thread吗？

new Thread的弊端如下：

- 每次new Thread新建对象性能差。
- 线程缺乏统一管理，可能无限制新建线程，相互之间竞争，及可能占用过多系统资源导致死机或oom。
- 缺乏更多功能，如定时执行、定期执行、线程中断。

相比new Thread，Java提供的四种线程池的好处在于：

- 重用存在的线程，减少对象创建、消亡的开销，性能佳。
- 可有效控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。
- 提供定时执行、定期执行、单线程、并发数控制等功能。

2. Java 线程池

Java通过Executors提供四种线程池，分别为： newCachedThreadPool() 创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。 newFixedThreadPool() 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。 newScheduledThreadPool() 创建一个定长线程池，支持定时及周期性任务执行。 newSingleThreadExecutor() 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- String字符串优化配套视频
- 其他优化配套视频

String字符串优化

最常见的例子就是当你要频繁操作一个字符串时，使用StringBuffer代替String。 还比如：使用int数组而不是Integer数组。 避免创建短命的临时对象，减少对象的创建就能减少垃圾收集，进而减少对用户体验的影响。

ListView优化

1. Item布局，层级越少越好，使用hierarchyview工具查看优化。
2. 复用convertView
3. 使用ViewHolder
4. item中有图片时，异步加载
5. 快速滑动时，不加载图片
6. item中有图片时，应对图片进行适当压缩
7. 实现数据的分页加载

减少不必要的全局变量

尽量避免static成员变量引用资源耗费过多的实例，比如Context。 因为Context的引用超过它本身的生命周期，会导致Context泄漏。所以尽量使用Application这种Context类型。 你可以通过调用Context.getApplicationContext()或Activity.getApplication()轻松得到Application对象。

Cursor（游标）回收

Cursor是Android查询数据后得到的一个管理数据集合的类，在使用结束以后。应该保证Cursor占用的内存被及时的释放掉，而不是等待GC来处理。并且Android明显是倾向于编程者手动的将Cursor close掉，因为在源代码中我们发现，如果等到垃圾回收器来回收时，会给用户以错误提示。

Receiver（接收器）回收

调用registerReceiver()后未调用unregisterReceiver(). 当我们Activity中使用了registerReceiver()方法注册了BroadcastReceiver，一定要在Activity的生命周期内调用unregisterReceiver()方法取消注册 也就是说registerReceiver()和unregisterReceiver()方法一定要成对出现，通常我们可以重写Activity的onDestory()方法，在onDestory里进行unregisterReceiver操作

Stream/File（流/文件）回收

主要针对各种流，文件资源等等如： InputStream/OutputStream，SQLiteOpenHelper，SQLiteDatabase，Cursor，文件，I/O，Bitmap图片等操作等都应该记得显示关闭。

避免内部Getters/Setters

在Android中，虚方法调用的代价比直接字段访问高昂许多。通常根据面向对象语言的实践，在公共接口中使用Getters和Setters是有道理的，但在一个字段经常被访问的类中宜采用直接访问。 for循环 访问成员变量比访问本地变量慢得多，如下面一段代码：

```
for(int i =0; i < this.mCount; i++)  {}
```

永远不要在for的第二个条件中调用任何方法，如下面一段代码：

```
for(int i =0; i < this.getCount(); i++) {}
```

对上面两个例子最好改为：

```
int count = this.mCount; / int count = this.getCount();
for(int i =0; i < count; i++)  {}
```

- 欢迎关注微信公众号，长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频。

- HierarchyViewer配套视频
- Lint配套视频
- 内存抖动现象配套视频12-如何对android应用进行内存性能分析

Android应用UI性能分析

在使用App时会发现有些界面启动卡顿、动画不流畅、列表等滑动时也会卡顿出现这种情况，可以考虑对UI性能分析。

> 首先要清楚卡顿的原因，有以下几种情况：

- 人为在UI线程中做轻微耗时操作，导致UI线程卡顿
- 布局Layout过于复杂，无法在16ms内完成渲染
- 同一时间动画执行的次数过多，导致CPU或GPU负载过重
- View过度绘制，导致某些像素在同一帧时间内被绘制多次，从而使CPU或GPU负载过重
- View频繁的触发measure、layout，导致measure、layout累计耗时过多及整个View频繁的重新渲染
- 内存频繁触发GC过多（同一帧中频繁创建内存），导致暂时阻塞渲染操作
- 冗余资源及逻辑等导致加载和执行缓慢
- 臭名昭著的ANR

如何分析?

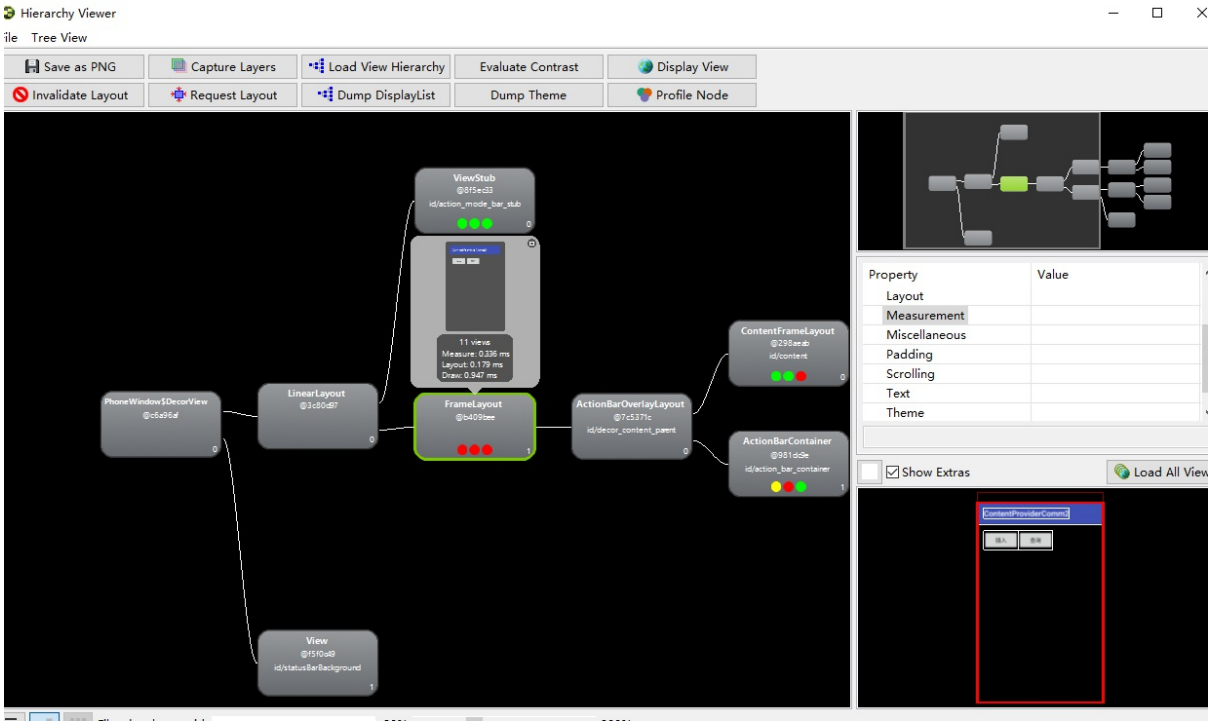
> 分析UI卡顿我们一般都借助工具，通过工具一般都可以直观的分析出问题原因，从而反推寻求优化方案，具体如下细说各种强大的工具

使用HierarchyViewer分析UI性能

我们可以通过SDK提供的工具HierarchyViewer来进行UI布局复杂程度及冗余等分析 通过命令启动HierarchyViewer

```
1  xxx@ThinkPad:~$ hierarchyviewer   //通过命令启动HierarchyViewer
```

接下来Hierarchy window窗口打开：

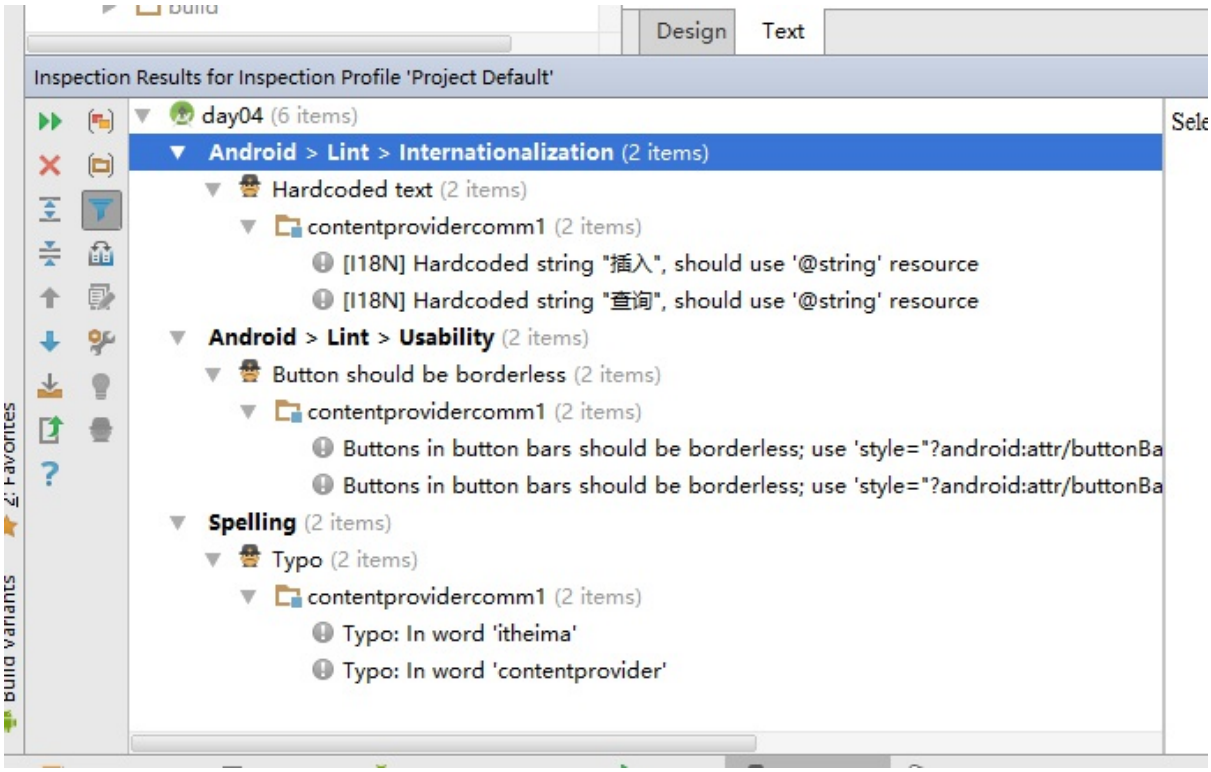


一个Activity的View树，通过这个树可以分析出View嵌套的冗余层级，以及每个View在绘制的使用时长也有表示。

使用Lint进行资源及冗余UI布局等优化

冗余资源及逻辑等也可能会导致加载和执行缓慢，这可以使用Link工具，来发现优化这些问题的

在Android Studio 1.4版本中使用Lint最简单的办法： 就是将鼠标放在代码区点击右键->Analyze->Inspect Code–>界面选择你要检测的模块->点击确认开始检测，等待一下后会发现如下结果：



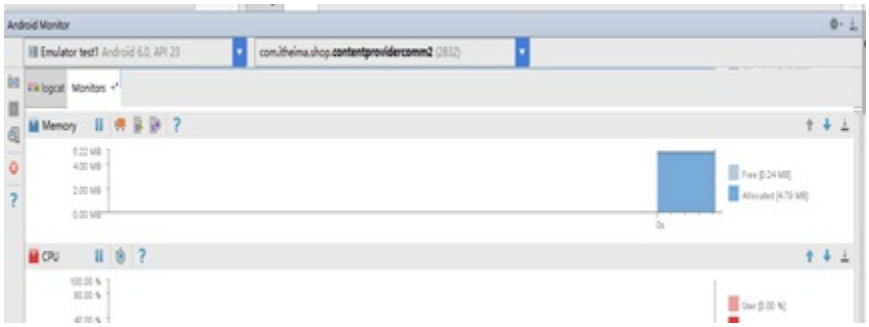
如果存在冗余的UI层级嵌套，会进行高亮显示， 我们根据提示可以点击跳进去进行优化处理掉的。

使用Memory监测及GC打印与Allocation Tracker进行UI卡顿分析

由于Android系统会依据内存中不同的内存数据类型分别执行不同的GC操作，常见应用开发中导致GC频繁执行的原因主要可能是因为短时间内有大量频繁的对象创建与释放操作，也就是俗称的内存抖动现象，或者短时间内已经存在大量内存暂用介于阈值边缘，接着每当有新对象创建时都会导致超越阈值触发GC操作

如何查看?

Android Studio 工具提供内存查看器：



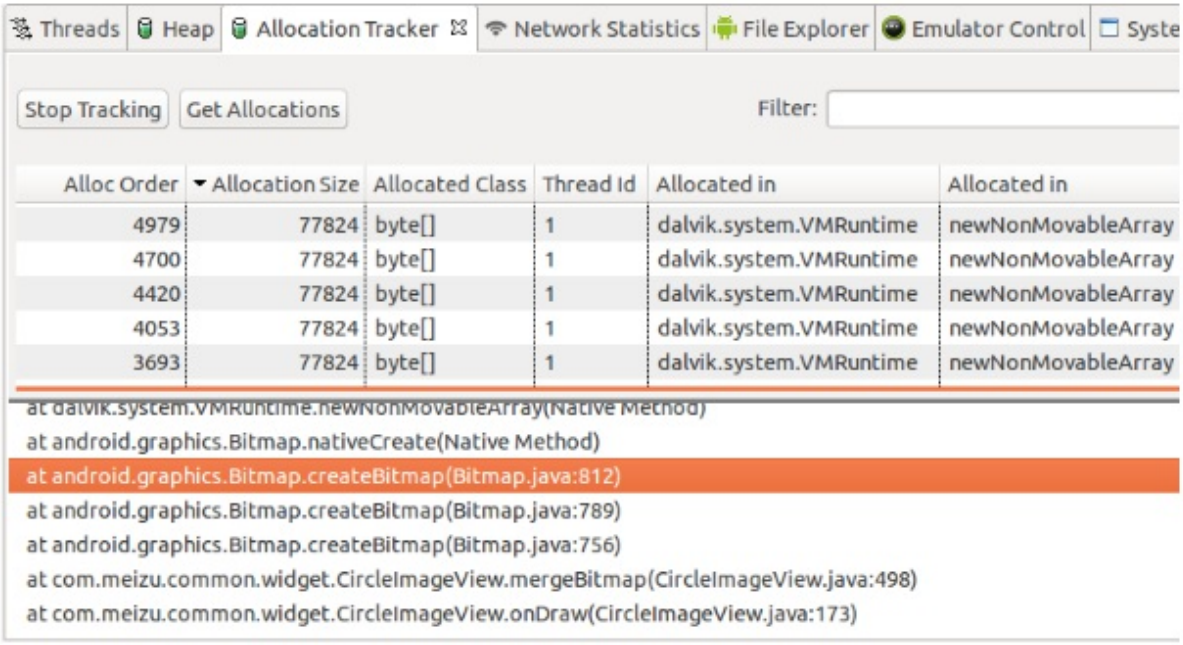
根据内存抖动现象，查看log日志进行分析：

```
//截取其中比较密集一段LogCat，与上图Memory检测到的抖动图对应，其中xxx为应用包名
......
10-06 00:59:45.619 xxx I/art: Explicit concurrent mark sweep GC freed 72515(3MB) AllocSpace objects, 65(2028KB) LOS objects, 8
10-06 00:59:45.749 xxx I/art: Explicit concurrent mark sweep GC freed 5396(193KB) AllocSpace objects, 0(0B) LOS objects, 75% fr
......
10-06 00:59:48.059 xxx I/art: Explicit concurrent mark sweep GC freed 4693(172KB) AllocSpace objects, 0(0B) LOS objects, 75% fr
......
```

> 如何看到，这种不停的大面积打印GC导致所有线程暂停的操作必定会导致UI视觉的卡顿，所以我们要避免此类问题的出现，具体的常见优化方式如下:

- 检查代码，尽量避免有些频繁触发的逻辑方法中存在大量对象分配
- 尽量避免在多次for循环中频繁分配对象
- 避免在自定义View的onDraw()方法中执行复杂的操作及创建对象（譬如Paint的实例化操作不要写在onDraw()方法中等）
- 对于并发下载等类似逻辑的实现尽量避免多次创建线程对象，而是交给线程池处理。

有了上面说明GC导致的性能后我们就该定位分析问题了， 我们可以通过运行DDMS->Allocation Tracker标签打开一个新窗口，然后点击Start Tracing按钮，接着运行你想分析的代码，运行完毕后点击Get Allocations按钮就能够看见一个已分配对象的列表，如下:



根据内存分配情况，进行优化。

写篇文章和录制视频从选题，思考，组织，写作，编辑至少需要一个多小时，而您点赞和转发只需要0.1秒，就可以带给我更大的动力，何乐而不为呢？

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- Monitors使用配套视频
- DDMS-Heap配套视频
- leakcanary配套视频

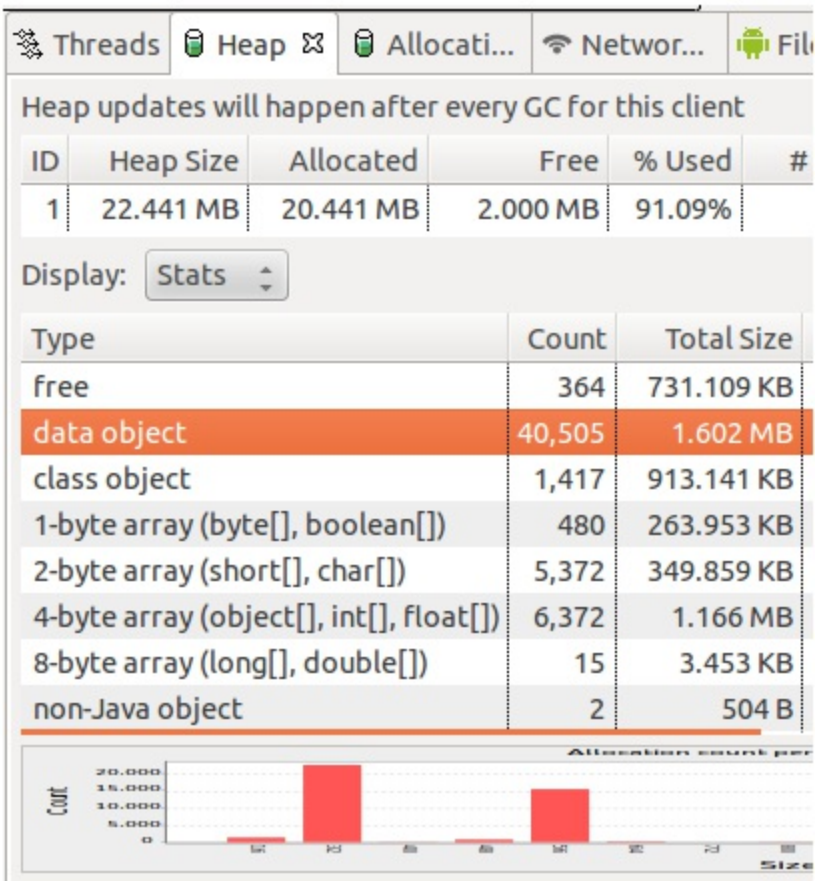
12-如何对android应用进行内存性能分析

如何查看项目中内存泄漏?

察觉方式	场景
AS的Memory窗口	平时用来直观了解自己应用的全局内存情况，大的泄露才能有感知。
DDMS-Heap内存监测工具	同上，大的泄露才能有感知。
dumpsys meminfo命令	常用方式，可以很直观的察觉一些泄露，但不全面且常规足够用。
leakcanary神器	比较强大，可以感知泄露且定位泄露；实质是MAT原理，只是更加自动化了，当现有代码量已经庞大成型，且无法很快察觉掌控全局代码时极力推荐；或者是偶现泄露的情况下极力推荐。

> AS的Memory窗口如下，详细的说明这里就不解释了，很简单很直观（使用频率高）：

> DDMS-Heap内存监测工具窗口如下，详细的说明这里就不解释了，很简单（使用频率不高）：

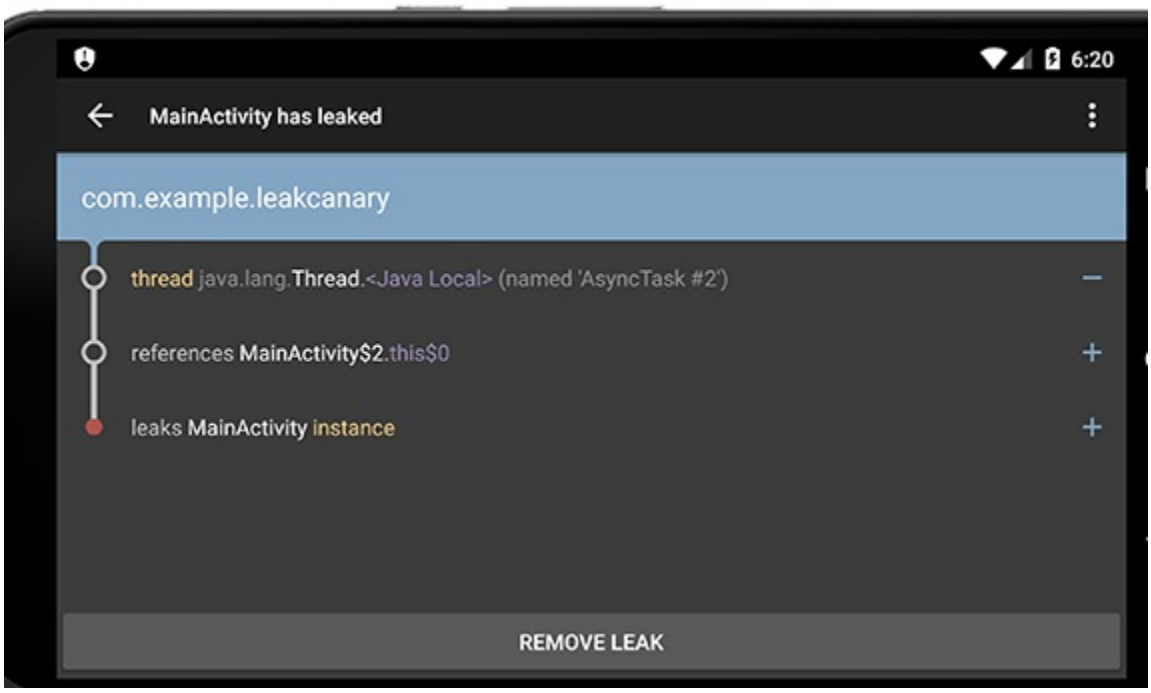


dumpsys meminfo命令如下（使用频率非常高，非常高效，注意：adb shell dumpsys meminfo不跟参数直接展示系统所有内存状态）：

```
[redacted]:~$ adb shell dumpsys meminfo -a com.sina.weibo
Applications Memory Usage (kB):
Uptime: 98283424 Realtime: 99272862

** MEMINFO in pid 25855 [com.sina.weibo] **
                   Pss      Pss   Shared  Private   Shared  Private  Swapped     Heap     Heap     Heap
                 Total    Clean    Dirty    Dirty    Clean    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
  Native Heap        0        0        0        0        0        0        0    20480    13199     7280
  Dalvik Heap    36374        0     3400    36204        0        0    14796    53665    51284     2381
 Dalvik Other      941        0       16      940        0        0       56
       Stack       652        0        0      652        0        0       28
      Ashmem       134        0       52      128        0        0        0
   Other dev         9        0       64        0        0        8        0
     .so mmap     7704     5896      844      340     9668     5896     1756
     .apk mmap    1374     1108        0        0     1832     1108        0
     .ttf mmap     331      104        0        0     1252      104        0
     .dex mmap    8615     6624        0        0     6476     6624        0
     .oat mmap    3609      884        0        0    17336      884        0
     .art mmap    1804       24     1564     1296     7552       24       48
   Other mmap      285        0        8        4      364      256        0
      Unknown    11050        0     1120    10996        0        0     1064
        TOTAL    72882    14640     7068    50560    44480    14904    17748    74145    64483     9661

 Dalvik Details
        .Heap    16808        0        0    16808        0        0        0
         .LOS    16572        0        0    16572        0        0        0
          .GC      893        0        8      892        0        0       56
      .Zygote      462        0     3400      292        0        0    14796
   .NonMoving     2532        0        0     2532        0        0        0
 .IndirectRef       48        0        8       48        0        0        0

 Objects
               Views:      188         ViewRootImpl:        1
         AppContexts:        9           Activities:        4
              Assets:        5         AssetManagers:        5
        Local Binders:       65         Proxy Binders:       35
        Parcel memory:       20          Parcel count:       80
     Death Recipients:        2       OpenSSL Sockets:        3

 Dalvik
        isLargeHeap:    false

 SQL
        MEMORY_USED:     1005
  PAGECACHE_OVERFLOW:      664          MALLOC_SIZE:       62

 DATABASES
      pgsz     dbsz   Lookaside(b)          cache  Dbname
         4       56            325       32/65/19  /data/data/com.sina.weibo/databases/sinamobilead.db
         4      248             92       49/121/8  /data/data/com.sina.weibo/databases/sina_weibo
         4       16             25         2/29/5  /data/data/com.sina.weibo/databases/sinamobileadparams.db
```

Android应用内存泄露leakcanary工具定位分析

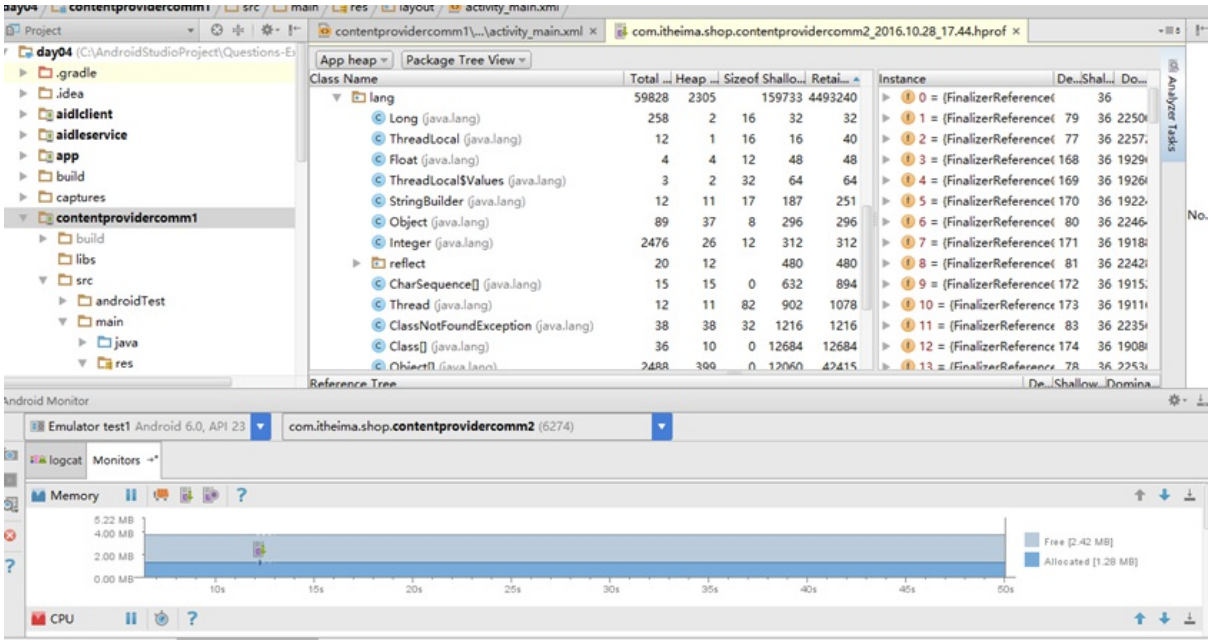
leakcanary是一个开源项目，一个内存泄露自动检测工具，是著名的GitHub开源组织Square贡献的，它的主要优势就在于自动化过早的发觉内存泄露。



关于leakcanary工具的配置使用方式这里不再详细介绍，可以参考官方说明使用：
https://github.com/square/leakcanary

> Android应用内存泄露MAT工具定位分析

Eclipse Memory Analysis Tools是一个专门分析Java堆数据内存引用的工具，我们可以使用它方便的定位内存泄露原因，核心任务就是找到GC ROOT位置即可



写篇文章和录制视频从选题，思考，组织，写作，编辑至少需要一个多小时，而您点赞和转发只需要0.1秒，就可以带给我更大的动力，何乐而不为呢？

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 配套视频

Android性能优化之App应用启动分析与优化

App启动方式

通常来说, 一个App启动也会分如下二中不同的状态:

.1）冷启动

当启动应用时，后台没有该应用的进程，这时系统会重新创建一个新的进程分配给该应用，这个启动方式就是冷启动。冷启动因为系统会重新创建一个新的进程分配给它，所以会先创建和初始化Application类，再创建和初始化MainActivity类（包括一系列的测量、布局、绘制），最后显示在界面上。

2.）热启动

当启动应用时，后台已有该应用的进程（例：按back键、home键，应用虽然会退出，但是该应用的进程是依然会保留在后台，可进入任务列表查看），所以在已有进程的情况下，这种启动会从已有的进程中来启动应用，这个方式叫热启动。热启动因为会从已有的进程中来启动，所以热启动就不会走Application这步了，而是直接走MainActivity（包括一系列的测量、布局、绘制），所以热启动的过程只需要创建和初始化一个MainActivity就行了，而不必创建和初始化Application，因为一个应用从新进程的创建到进程的销毁，Application只会初始化一次。

启动时间的测量

关于Activity启动时间的定义

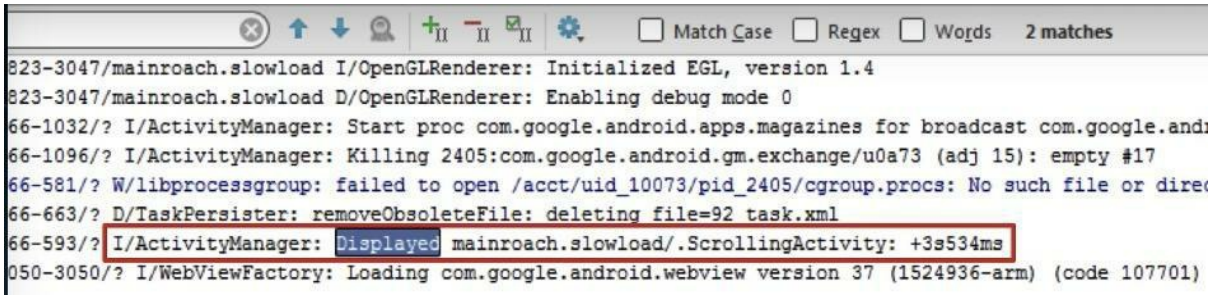
对于Activity来说，启动时，首先执行的是onCreate()、onStart()、onResume()这些生命周期函数，但即使这些生命周期方法回调结束了，应用也不算已经完全启动，还需要等View树全部构建完毕，一般认为，setContentView中的View全部显示结束了，算作是应用完全启动了。

Display Time

从API19之后，Android在系统Log中增加了Display的Log信息，通过过滤ActivityManager以及Display这两个关键字，可以找到系统中的这个Log:

```
$ adb logcat | grep “ActivityManager”
ActivityManager: Displayed com.example.launcher/.LauncherActivity: +999ms
```

抓到的Log如图所示:



那么这个时间，实际上是Activity启动，到Layout全部显示的过程，但是要注意，这里并不包括数据的加载，因为很多App在加载时会使用懒加载模式，即数据拉取后，再刷新默认的UI。

基于上面的启动流程我们尽量做到如下几点

1) Application的创建过程中尽量少的进行耗时操作

2) 首屏Activity的渲染

当前冷启动效果:

Application

Application是程序的主入口，特别是很多第三方SDK都会需要在Application的onCreate里面做很多初始化操作，一般来说我们可以将这些初始化放在一个单独的线程中处理, 为了方便今后管理,

优化的方法，无非是通过以下几个方面:

1）异步初始化 2）后台任务 3）界面预加载

异步初始化

这个很简单，就是让App在onCreate里面尽可能的少做事情，而利用手机的多核特性，尽可能的利用多线程，例如一些第三方框架的初始化，如果能放线程，就尽量的放入线程中，最简单的，你可以直接new Thread()，当然，你也可以通过公共的线程池来进行异步的初始化工作，这个是最能够压缩启动时间的方式

后台任务

因为这个App一般会集成很多三方SDK等服务, 所以Application的onCreate有很多第三方平台的初始化工作…

Application代码如下:

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        LitePal.initialize(this.getApplicationContext());
    }
}
```

使用IntentService不同于Service, 它是工作在后台线程的.

IntentService代码如下:

```
public class InitializeService extends IntentService {
    private static final String ACTION_INIT_WHEN_APP_CREATE = "com.maweiqi";
    public InitializeService(String name) {
        super(name);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if (ACTION_INIT_WHEN_APP_CREATE.equals(action)) {
                performInit();
            }
        }
    }
    private void performInit() {
        LitePal.initialize(this.getApplicationContext());
    }
    public static void start(Context context) {
        Intent intent = new Intent(context, InitializeService.class);
        intent.setAction(ACTION_INIT_WHEN_APP_CREATE);
        context.startService(intent);
    }
}
```

MyApp的onCreate改成:

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        InitializeService.start(this);
    }
}
```

最终的效果

界面预加载

当系统加载一个Activity的时候，onCreate()是一个耗时过程，那么在这个过程中，系统为了让用户能有一个比较好的体验，实际上会先绘制一些初始界面，类似于PlaceHolder。

系统首先会读取当前Activity的Theme，然后根据Theme中的配置来绘制，当Activity加载完毕后，才会替换为真正的界面。所以，Google官方提供的解决方案，就是通过android:windowBackground属性，来进行加载前的配置，同时，这里不仅可以配置颜色，还能配置图片，例如，我们可以使用一个layer-list来作为android:windowBackground要显示的图：

在drawable文件夹下面，做一个logo_splash的背景: start_window.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 底层白色 -->
    <item android:drawable="@color/white" />

    <!-- 顶层Logo居中 -->
    <item>
        <bitmap
            android:gravity="center"
            android:src="@drawable/ic_github" />
```

```
    </item>
</layer-list>
```

在style里面弄一个主题:

```
<style name="StartStyle" parent="AppTheme">
        <item name="android:windowBackground">@drawable/start_window</item>
    </style>
```

在清单文件里面给Activity指定需要预加载的Style:

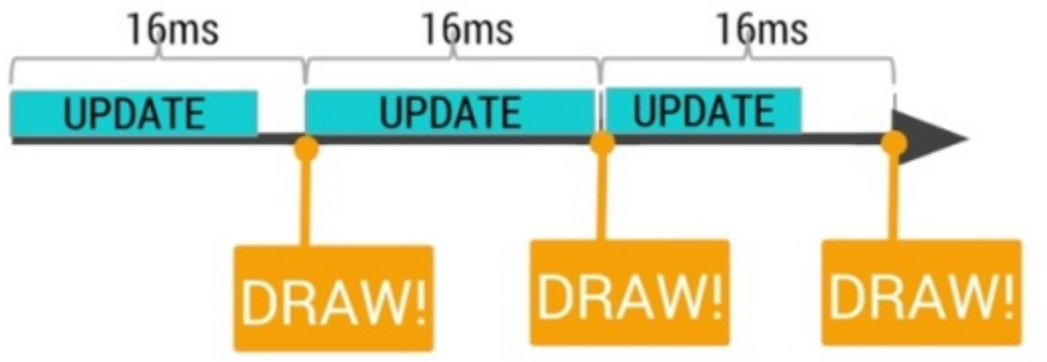
```
<activity android:name=".MainActivity" android:theme="@style/StartStyle">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
```

activity代码如下:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        setTheme(R.style.AppTheme);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

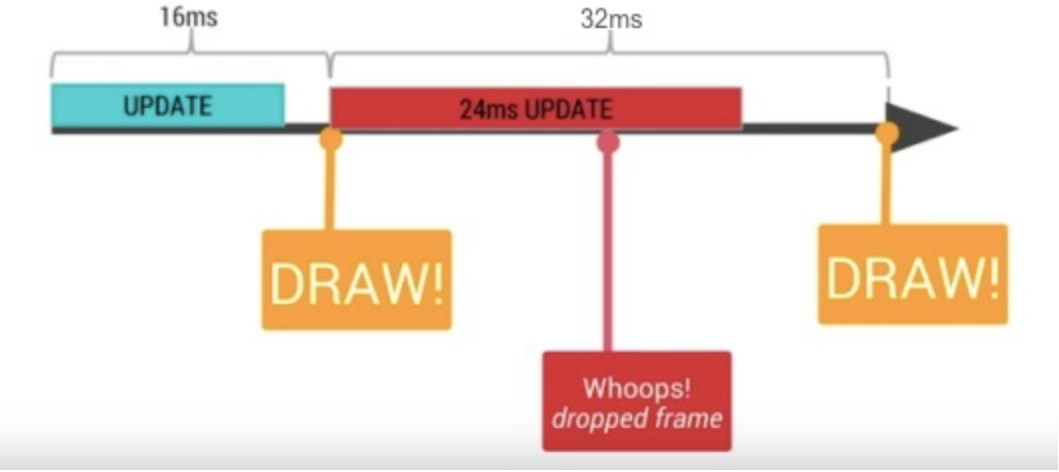
首屏Activity的渲染

Android系统每隔16ms会发出VSYNC信号重绘我们的界面(Activity). 为什么是16ms, 因为Android设定的刷新率是60FPS(Frame Per Second), 也就是每秒60帧的刷新率, 约合16ms刷新一次.就像是这样的:



这就意味着, 我们需要在16ms内完成下一次要刷新的界面的相关运算, 以便界面刷新更新. 然而, 如果我们无法在16ms内完成此次运算会怎样呢?

例如, 假设我们更新屏幕的背景图片, 需要24ms来做这次运算. 当系统在第一个16ms时刷新界面, 然而我们的运算还没有结束, 无法绘出图片. 当系统隔16ms再发一次VSYNC信息重绘界面时, 用户才会看到更新后的图片. 也就是说用户是32ms后看到了这次刷新(注意, 并不是24ms). 这就是传说中的丢帧(dropped frame):



丢帧给用户的感觉就是卡顿, 而且如果运算过于复杂, 丢帧会更多, 导致界面常常处于停滞状态, 卡到爆.

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

与人事相关面试题

- 人事面试宝典

> 现在三四月份，金三银四最好找工作时间，为方便各位找工作，特意收集100道android各个方面的面试题，并且会一一录制视频分享给大家方便大家找工作，面试题分类如下；


Android中数据存储方式和缓存
1- 请介绍下Android中的数据存储方式（8分）
2- 在项目中，你是如何缓存数据的？（10分）
3- 说说LruCache底层原理（10分）
4- 加载图片的三级缓存原理（10分）
动画相关面试题
5- android中的动画有哪些,它们的特点和区别是...
6- 动画插值器是什么（20分）
7- 请说一下对属性动画的理解（20分）
8- 如何让自定义view中绘制的内容执行动画（20...
9- 介绍一下Matrix的作用（3分）
自定义控件相关面试题
与WebView相关的面试题
25- webview和js的交互（5分）
26- WebView加载数据性能优化？webview加载...
27- 如何使WebView自适应屏幕大小？
Android干货程序员


View的实现和优化相关面试题
1-ListView优化？ListView如何提高其效率？
2-如何实现让ListView的条目显示不一样的布局?
3-ScrollView中嵌入ListView常见问题和解决方...
跨平台开发相关面试题
4-简单描述一下Html5的特点？
屏幕适配相关的面试题
5-屏幕适配的方式有哪些？
6-请问平时开发过程中，你是如何做到多分辨率...
与设计模式相关的面试题
7-简单说说MVC,MVP原理以及它们之间的区别...
8-你一般在开发项目中都使用什么设计模式？如...
9-手写一个单例模式，能不能保证使用时对象的...
与网络相关的面试题
与线程间通信机制相关面试题
与延迟和异步任务相关试题
Android干货程序员

本文配套视频

- 配套视频
- 配套视频
- 配套视频
- 配套视频
- 欢迎关注微信公众号,长期推荐技术文章和技术视频

1- Davik进程、linux进程、线程之间的区别?

Linux进程：

- Linux进程，它有独立的内核堆栈和独立的存储空间，它是操作系统中资源分配和调度的最小单位。
- Linux操作系统会以进程为单位，分配系统资源，给程序进行调度。
- Linux操作系统在执行一个程序时，它会创建一个进程，来执行应用程序，并且伴随着资源的分配和释放。

Davik进程：

- Dalvik虚拟机运行在Linux操作系统之上。
- Davik进程就是Linux操作系统中的一个进程，属于linux进程
- 每个Android应用程序进程都有一个Dalvik虚拟机实例。这样做得好处是Android应用程序进程之间不会互相影响，也就是说，一个Android应用程序进程的意外终止，不会影响到其他的应用程序进程的正常运行。

线程：

- 线程是进程的一个实体，是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。
- 线程自己基本上不拥有系统资源,在运行时，只需要必不可少的资源(如程序计数器,一组寄存器和栈)。
- 线程与同属一个进程的其他的线程共享进程所拥有的全部资源。

三者之间的联系：

- Davik进程就是Linux操作系统的一个进程。
- 线程就是进程的一个实体，线程是进程的一部分。一个进程中可以提供多个线程执行控制。

进程和线程的区别：

- 一个程序至少有一个进程,一个进程至少有一个线程.
- 线程的划分尺度小于进程，使得多线程程序的并发性高。
- 进程在执行过程中拥有独立的内存单元，而多个线程共享内存(同属一个进程)，从而极大地提高了程序的运行效率。
- 每个独立的进程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。
- 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

2-Android 中进程与进程之间如何通信?

> aidl机制进程间通信 AIDL: (Android Interface definition language的缩写)它是一种android内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口

AIDL进程间通讯的原理： 通过编写aidl文件来定义进程间通信接口。 编译后会自动生成响应的java文件 服务器将接口的具体实现写在Stub中，用iBinder对象传递给客户端， 客户端bindService的时候，用asInterface的形式将iBinder还原成接口，再调用其接口中的方法来实现通信。

使用Messenger实现进程间通信

> Messenger是基于AIDL实现的。 AIDL使服务器可以并行处理，而Messenger封装了AIDL之后只能串行运行，所以Messenger一般用作消息传递。

- 需要大家注意:
- 区别Messenger和Message。
  - Message是消息，承载了要传递的数据。
  - Messenger是信使，可以发送消息。并且Messenger对象可以通过getBinder方法获取一个Ibinder对象。

Messenger实现原理：

> 服务端（被动方）提供一个Service来处理客户端（主动方）连接，维护一个Handler来创建Messenger，在onBind时返回Messenger的binder。 双方用Messenger来发送数据，用Handler来处理数据。Messenger处理数据依靠Handler，所以是串行的，也就是说，Handler接到多个message时，就要排队依次处理。

使用Messenger实现进程间通信方法如下：

- 首先A应用提供一个Service，创建一个Messenger对象，在onBinder方法里返回messenger.getBinder()生成的IBinder对象； 然后在B应用绑定该Service，在ServiceConnection的onServiceConnected方法获取到IBinder对象；
- 最后在B应用使用获取到的binder对象构造出一个新的Messenger对象，使用该Messenger对象的send方法发送的Message数据，都将被Service里的Messenger对象handlerMessage方法接收到。

内容提供者ContentProvider实现进程间通信

系统四大组件之一，底层也是Binder实现，主要用来为其他APP提供数据。

> 自定义的ContentProvider为外界进程访问的时候， 需要在注册时要提供authorities属性，应用需要访问的时候将属性包装成Uri.parse("content://authorities")。

- 然后实现： ContentProvider的中query，delete，insert等相关方法，进行数据的操作。
- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

与登录相关面试题

- Oauth的实现原理
- Token的实际意义
- 微信扫码登录内部实现原理

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 配套视频

腾讯QQ第三方登录的实现原理?

> Oauth当中的角色:

1. Service Provider（服务提供方）

服务提供方通常是网站，在这些网站当中存储着一些受限制的资源，如照片、视频、联系人列表等。这些网站通常使用用户名和密码来确认用户的身份。比如新浪微博的开放平台就是Service Provider。

1. User（用户）

存放在服务提供方的受保护的资源的所有者。用户持有可以登录服务提供者网站的用户名和密码。用户不希望把自己的资源公开，但是用户却需要将这些资源共享给其他网站或应用程序（如用户希望使用第三方开发的新浪微博客户端来访问自己的资源，但又不希望第三方应用知道自己的用户名和密码）。

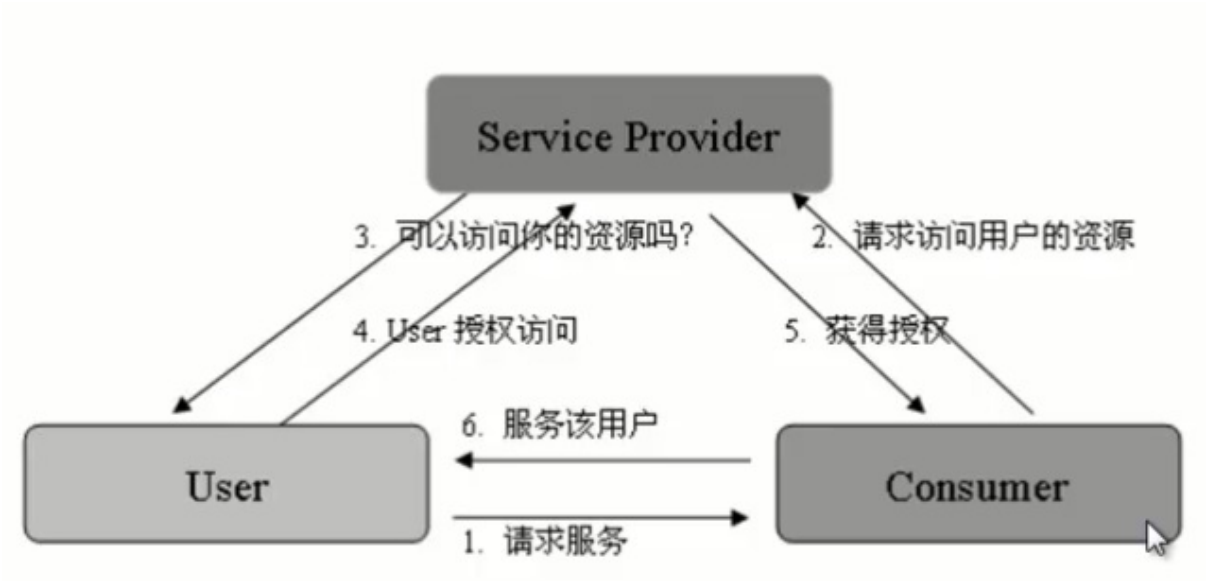
1. 客户端（Client）

要访问服务提供方资源的第三方应用，可以是web应用程序、桌面应用程序或者是手机应用程序。客户端需要得到授权之后才能访问相应的资源。

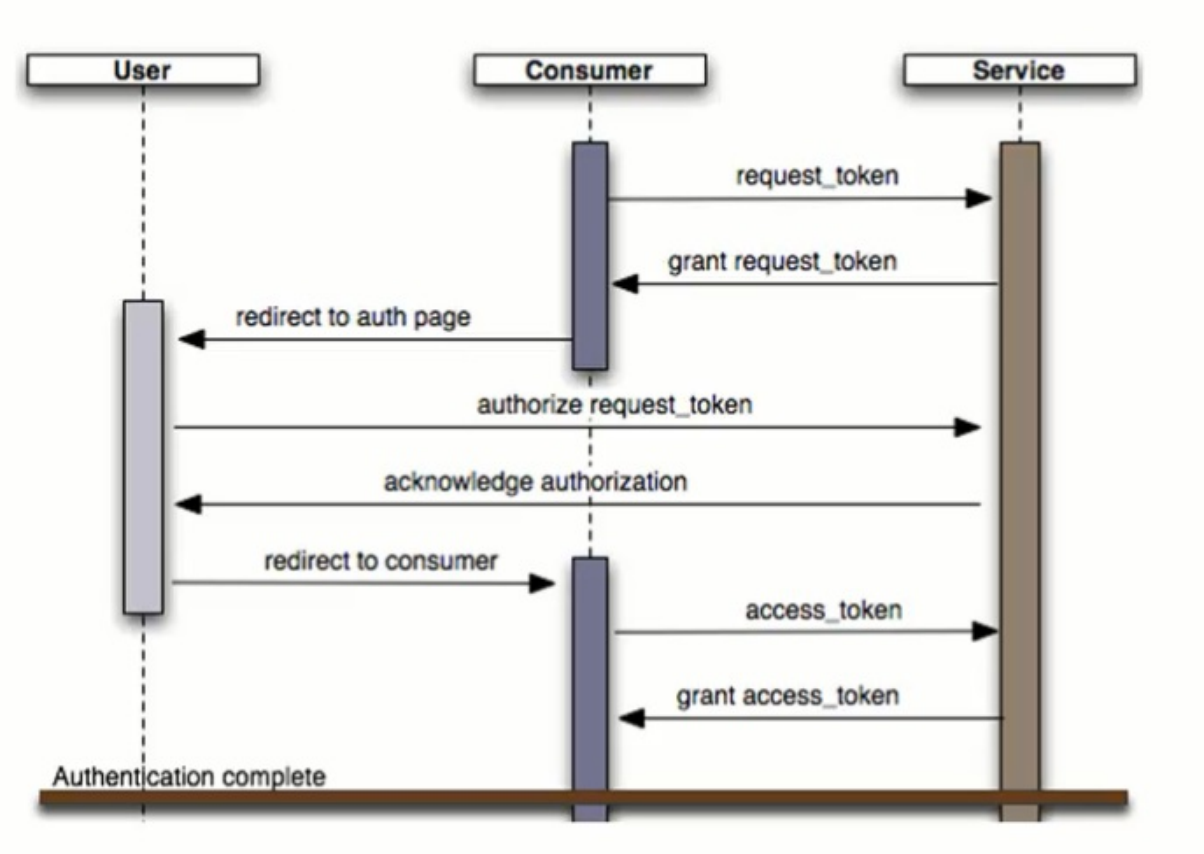
> Oauth的认证和授权过程:

1. 用户使用第三方的客户端（如访问第三方的网站，或使用第三方的应用），想对存放在服务提供者的某些资源进行操作
2. 第三方网站或应用向服务提供方请求一个临时令牌（Request Token）
3. 服务提供方验证第三方的身份后，授予其一个临时令牌
4. 第三方获得临时令牌后，将用户引导至服务提供方的授权页面请求用户授权。在这个过程中将临时令牌和客户端的回调连接发送给服务提供者
5. 用户在服务提供者的授权页面上输入自己的用户名和密码，然后授权该客户端访问相应的资源
6. 授权成功后，服务提供方引导用户返回第三方网站的的网页
7. 客户端根据临时令牌从服务提供方那里获取访问令牌（Access Token）
8. 服务提供方根据临时令牌和用户的授权情况授予客户端访问令牌
9. 客户端使用获取的访问令牌访问存放在服务提供方上的相应的资源

Oauth简单示意图



Oauth全面的示意图



- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

今天文章比较简单，主要是为了录制面试题系列，保证文章的完整性来帮助那些想找工作的哥们，各位高级程序员请勿拍砖,不过把下面三段视频都看完，多多少少会有些收获。。。

本文配套视频：

- 为什么需要token配套视频
- 一分钟登录配套视频
- 一分钟解析登录配套视频

在android开发中，用户登录时，客户端会接收到token值，请描述一下对token的理解？

Token的引入：

Token是在客户端频繁向服务端请求数据，服务端频繁的去数据库查询用户名和密码并进行对比，判断用户名和密码正确与否，并作出相应提示，在这样的背景下，Token便应运而生。

Token的定义：

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

使用Token的目的： Token的目的是为了验证用户登录情况以及减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。

Token的应用：

1. 当用户首次登录成功之后, 服务器端就会生成一个 token 值，这个值，会在服务器保存token值(保存在数据库中)，再将这个token值返回给客户端
2. 客户端拿到 token 值之后,使用sp进行保存
3. 以后客户端再次发送网络请求(一般不是登录请求)的时候,就会将这个 token 值附带到参数中发送给服务器
4. 服务器接收到客户端的请求之后,会取出token值与保存在本地(数据库)中的token值做对比
5. 如果两个 token 值相同， 说明用户登录成功过!当前用户处于登录状态
6. 如果没有这个 token 值, 没有登录成功
7. 如果 token 值不同: 说明原来的登录信息已经失效,让用户重新登录

使用一分钟利用开源中国接口获取登录token

利用一分钟时间解析服务器返回的token数据

懒得写文字了，直接看视频吧，看完会有不小的收获哦

登录和注销 (Cookie, Session)

- 登陆过程
    1. 先把用户名和密码传给服务器(请求头中没有Cookie)
    2. 服务器验证用户名和密码
    3. 验证通过->响应头中返回(Cookies)一个或多个key=value;key1=value1
    4. 客户端保存这些Cookies到本地文件.
- 请求数据过程
    1. 如果没有Cookie, 只能获取到公共信息.
    2. 把Cookie放到请求头中, 可以获取到Cookie对应的账号的隐私数据.

3. 如果Cookie超时了(一小时, 一周, 一个月), 服务器同样不返回隐私数据.

- 注销过程

1. 删除本地的Cookie
2. 以后的请求头中都没有Cookie
3. 服务器不会再知道客户端是谁, 会话结束.

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 配套视频

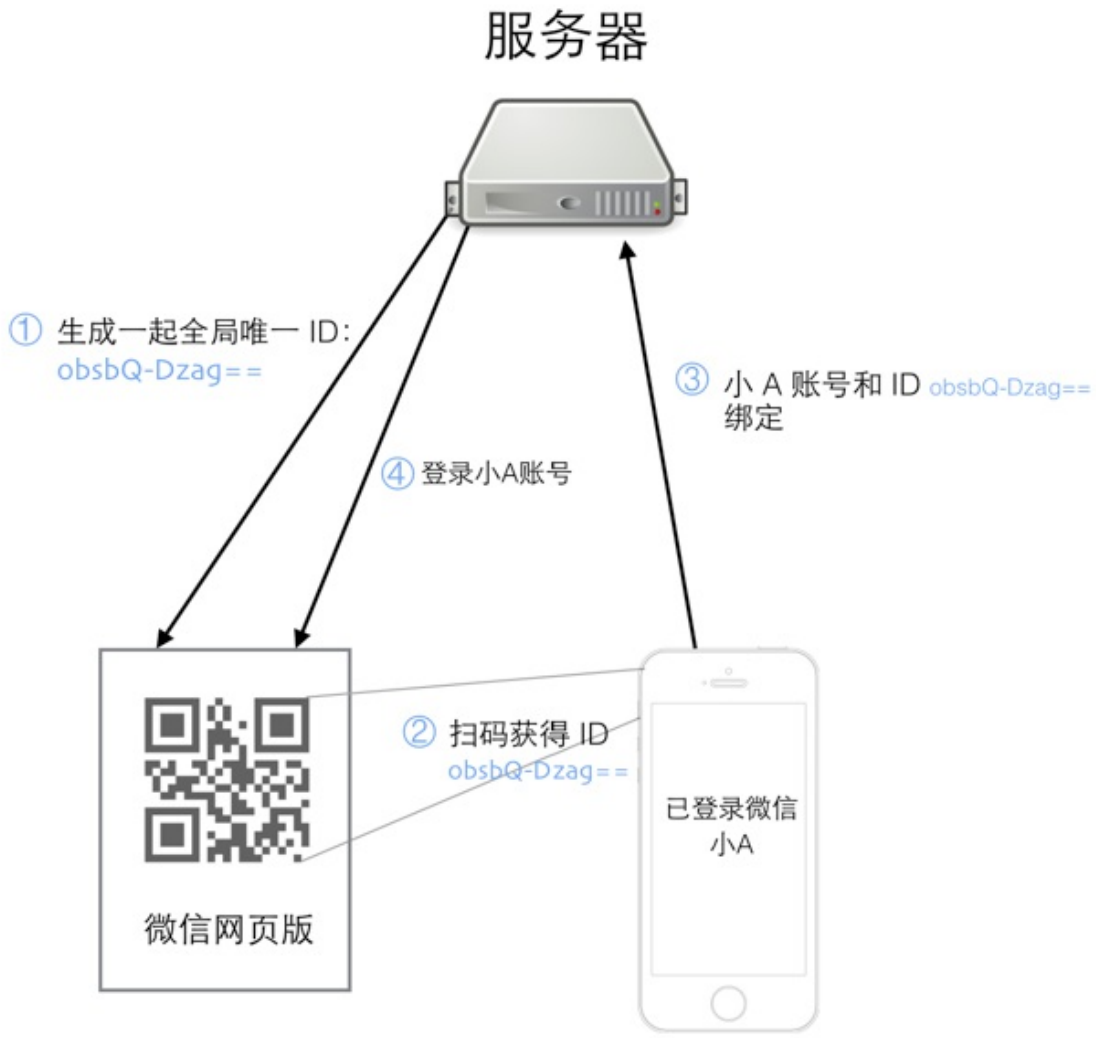
26微信扫码登录内部实现原理?

打开网页版微信，可以看到如下的页面：

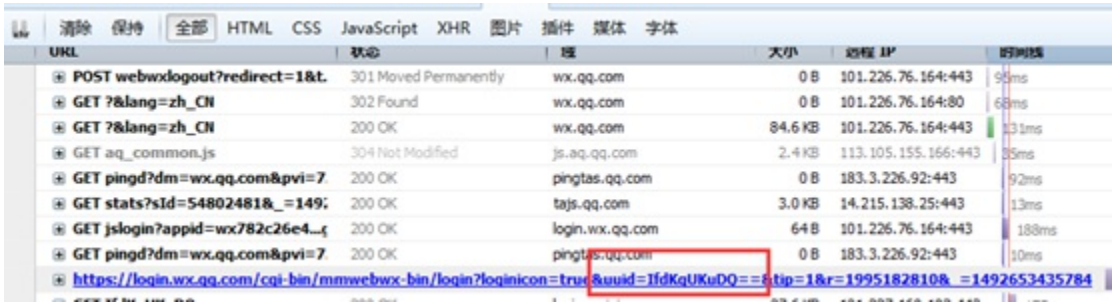
如果你用我查查、支付宝、新浪微博等软件扫码二维码，你会发现此二维码解析出来是如下的网址：

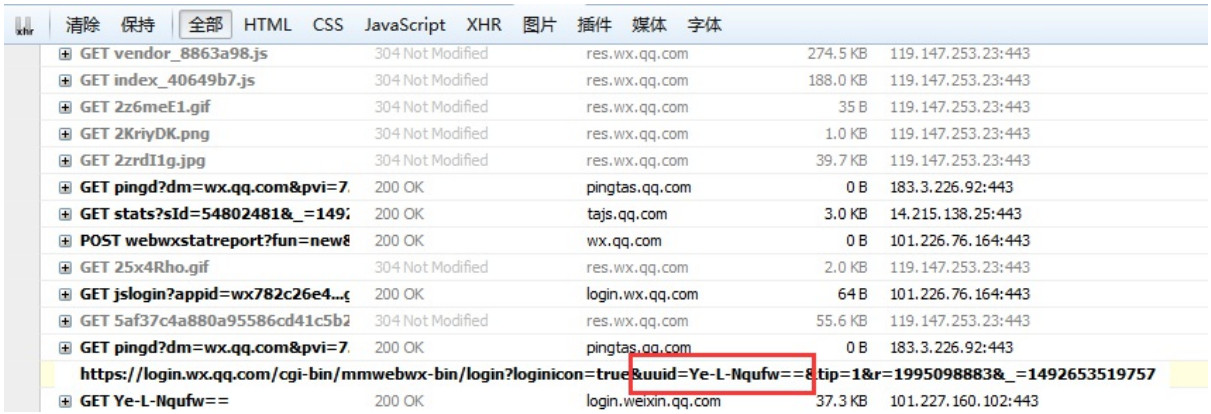
```
https://login.weixin.qq.com/l/obsbQ-Dzag==
```

接下来详细介绍一下扫码登录具体的每个步骤：



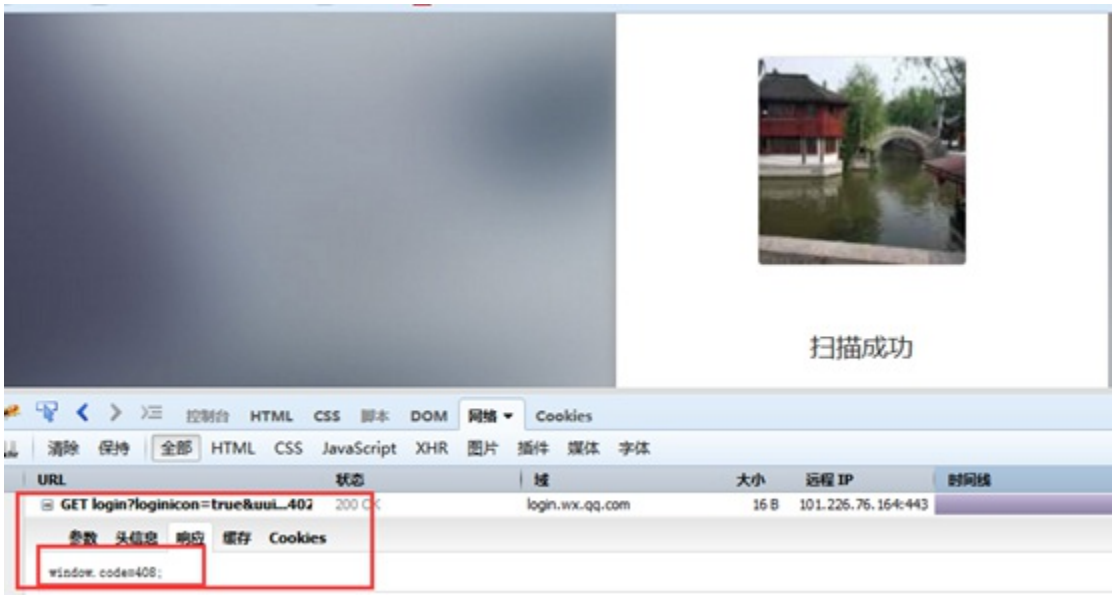
①：用户 A 访问微信网页版，微信服务器为这个会话生成一个全局唯一的 UUID二维码，上面的 URL 中 obsbQ-Dzag== 就是这个 UUID，且每次刷新后都会改变。这样可以保证一个UUID只可以绑定一个账号和密码，确定登录用户的唯一性。我刷新三次，扫描结果如下，其中最后面那串数字就是UUID：此时系统并不知道访问者是谁。



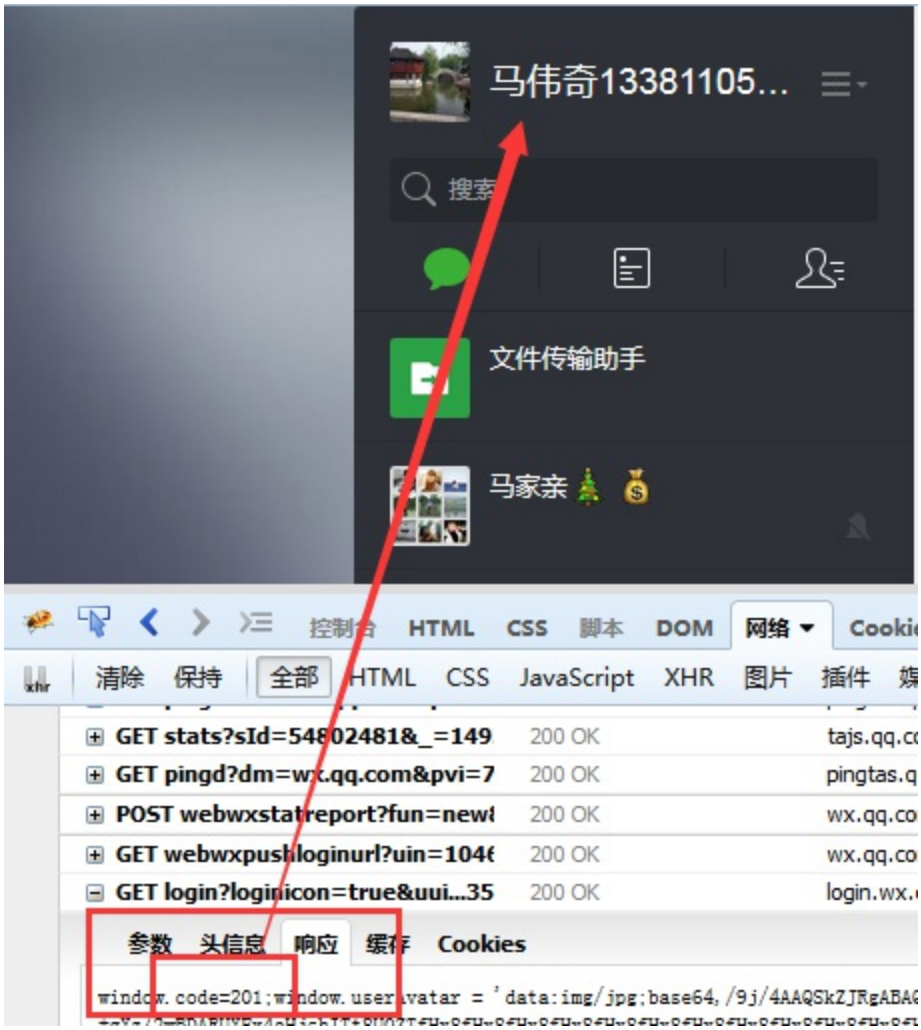


②：除了返回唯一的uid，实际上打开这个页面的时候，浏览器跟服务器还创建了一个长连接，请求uid的扫描记录。如果没有，在特定时长后会接到状态码408（请求超时），表示应该继续下一次请求；如果接到状态码201（服务器创建新资源成功），表示客户端扫描了该二维码。

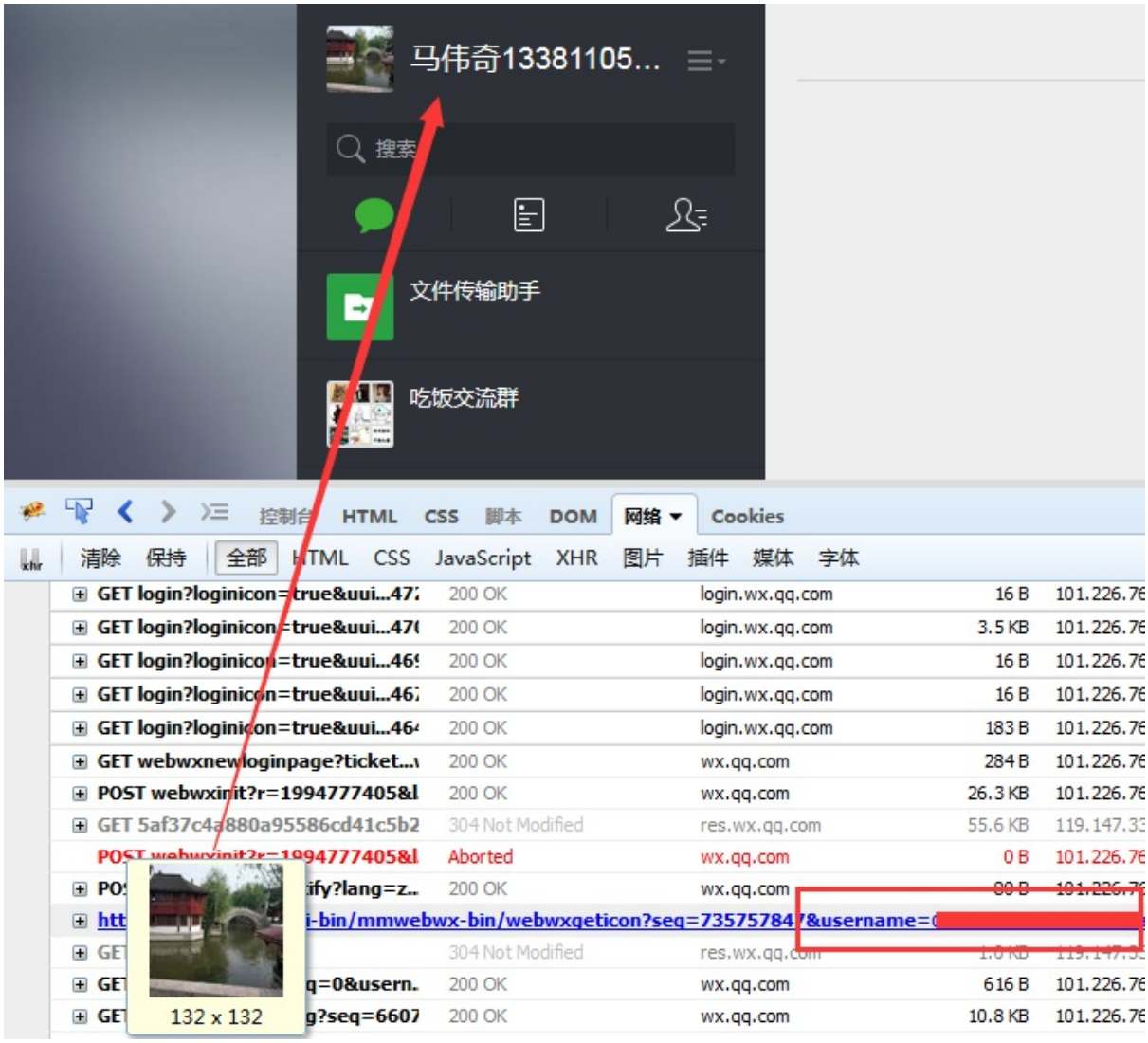
请求超时：返回408



扫码成功：返回201



③：手机上的微信是登录状态，用户点击确认登录后，手机上的微信客户端将微信账号和这个扫描得到的 ID 一起提交到服务器



④：服务器将这个 ID 和用户 A 的微信号绑定在一起，并通知网页版微信，这个 ID 对应的微信号为用户 A，网页版微信加载用户 A 的微信信息，至此，扫码登录全部流程完成

总的来说，微信扫码登录核心过程应该是这样的：浏览器获得一个唯一的、临时的UUID，通过长连接等待客户端扫描带有此UUID的二维码后，从长连接中获得客户端上报给服务器的帐号信息进行展示。并在客户端点击确认后，获得服务器授信的令牌，进行随后的信息交互过程。 在超时、网络断开、其他设备上登录后，此前获得的令牌或丢失、或失效，对授权过程形成有效的安全防护，类似的应用还有扫码支付、扫码加公众号等功能.

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口？
- 手把手教你如何解决as jar包冲突
- Context原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案
- 解决字体适配
- 软键盘顶出去解决方案
- 机型适配之痛

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频。。

- 配套视频

迭代开发的时候如何向前兼容新旧接口？

设计服务器接口时,每一个接口，都带版本号。比如用户登陆接口第 1 版为

```
/1/user/login
```

返回 Json 数据。数据结构改动后，假如 Json 数据只是增加字段，这时接口不用修改。当登陆接口改动太大，会删除或者修改字段。就递增版本号，新添接口：

```
/2/user/login
```

旧的 /1/user/login 接口需要保留,这时旧的客户端使用 /1/user/login，而新的客户端使用 /2/user/login。

在服务端 /1/user/login 和 /2/user/login 进行重构，某些地方调用相同的代码。两个接口并存一段时间后，比如过了 3 个月。估计旧的客户端差不多都升级到新的了，这时旧的 /1/user/login 接口就可以不再维护，直接返回错误码。

比如开源中国开发也是如此，开源中国API接口如下：

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

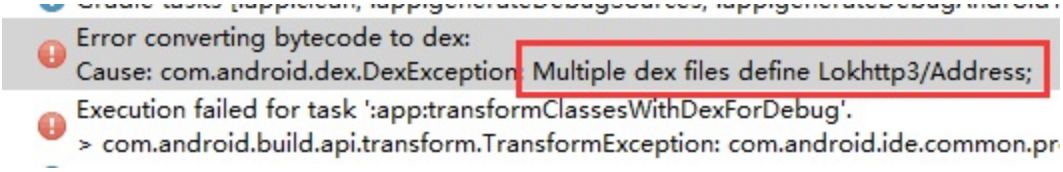
与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

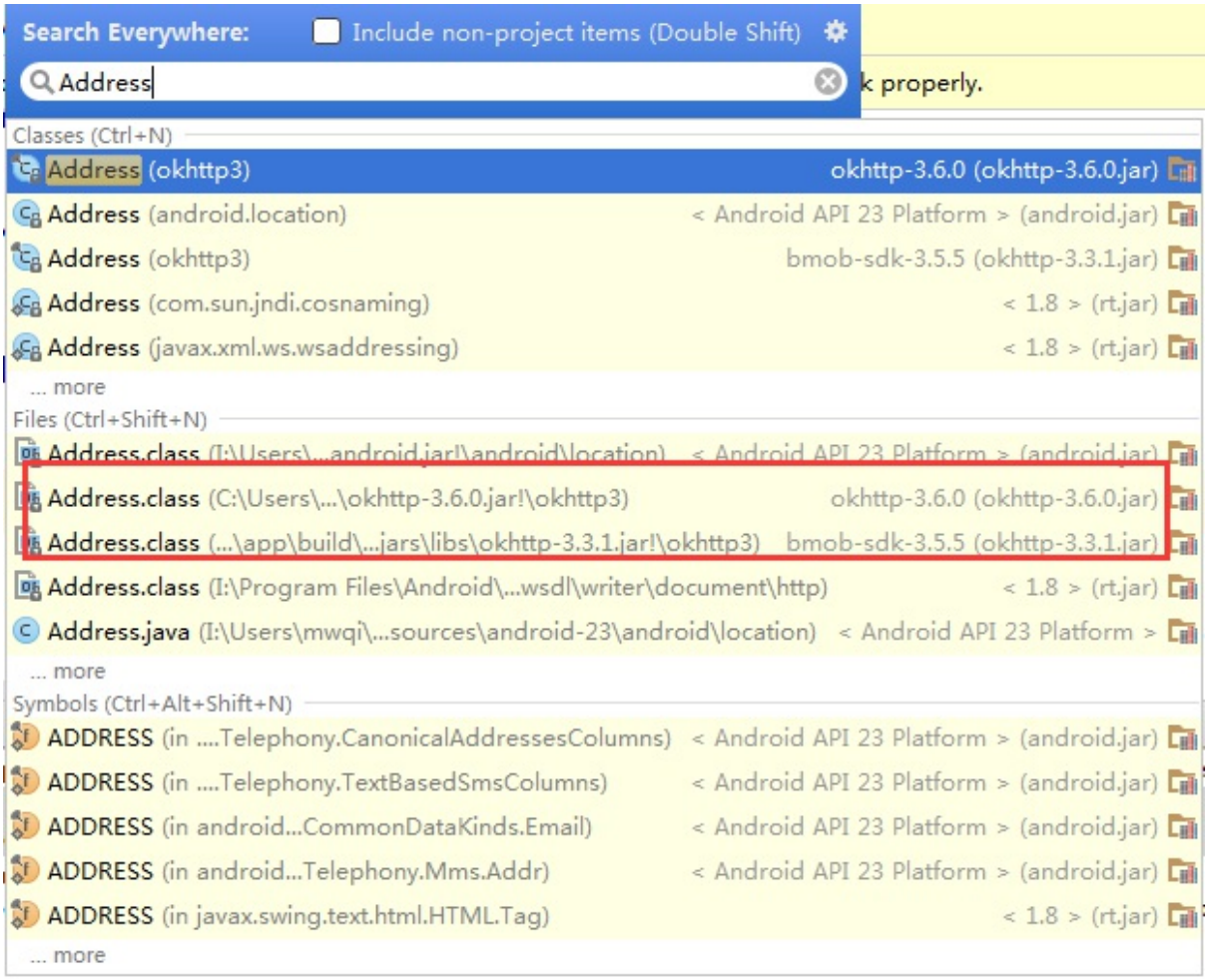
与人事相关面试题

- 人事面试宝典

第一：当出现这个错误就是jar包冲突。

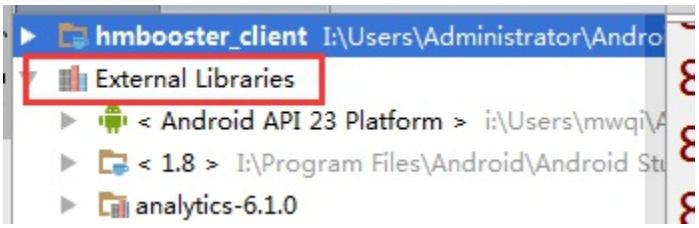


第二：解决办法，双击Shift搜索，会发现如下图片：

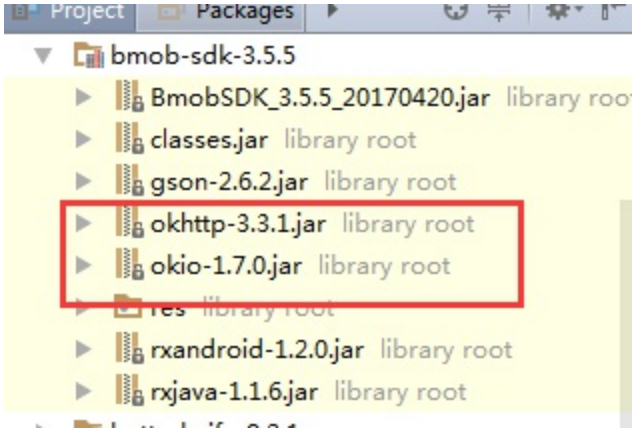


搜索你会发现我的项目里面添加了okhttp3.6，bomb里面也添加了okhttp3.3.1，这时候系统就不知道该用哪个jar冲突了。

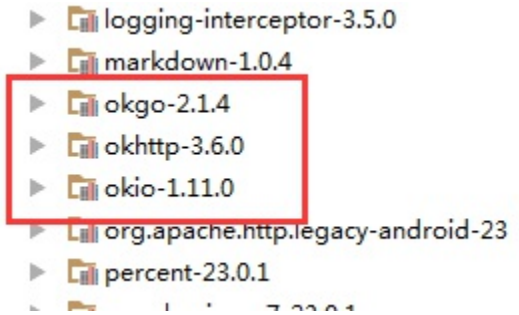
第三：打开项目的扩展库，如下图展示：



会发现bomb自带了okhttp的jar包，如图片展示。



我的项目也再带okhttp的jar包，如图片展示。



第四：找到我的build.gradle文件，去掉刚刚冲突的jar包，如下展示：

```
//bmob-sdk：Bmob的android sdk包，包含了Bmob的数据存储、文件等服务，以下是最新的bmob-sdk:
   compile ('cn.bmob.android:bmob-sdk:3.5.5'){ // gson-2.6.2
        exclude group: 'com.squareup.okhttp3'
        exclude group: 'com.squareup.okio'
        exclude group: 'com.google.code.gson'
//        exclude(module: 'gson')      // 防止版本冲突
    }

 // 网络加载
    compile ('com.squareup.okhttp3:okhttp:3.6.0'){
        exclude group: 'com.squareup.okhttp3'
        exclude group: 'com.squareup.okio'
    }
```

至此解决冲突。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

本文配套视频

- 配套视频

谈一下你对Android中的context的理解，在一个应用程序中有多少个context实例？

1. 什么是Context?

通过金山词霸解释：上下文环境，什么是环境，这个词只可意会不可言传，为了大家更好的理解，举一个栗子，比如：我想点鸡，我在麦当劳跟服务员说我想点鸡，服务员给端上来一只香喷喷的烤鸡，如下图：

但是我换一个环境，去红灯区点鸡，妈咪就会给带来一只呆萌可爱的失足少女，如下图：

这就是环境，一样的东西不同地方，就表示不一样的意思。

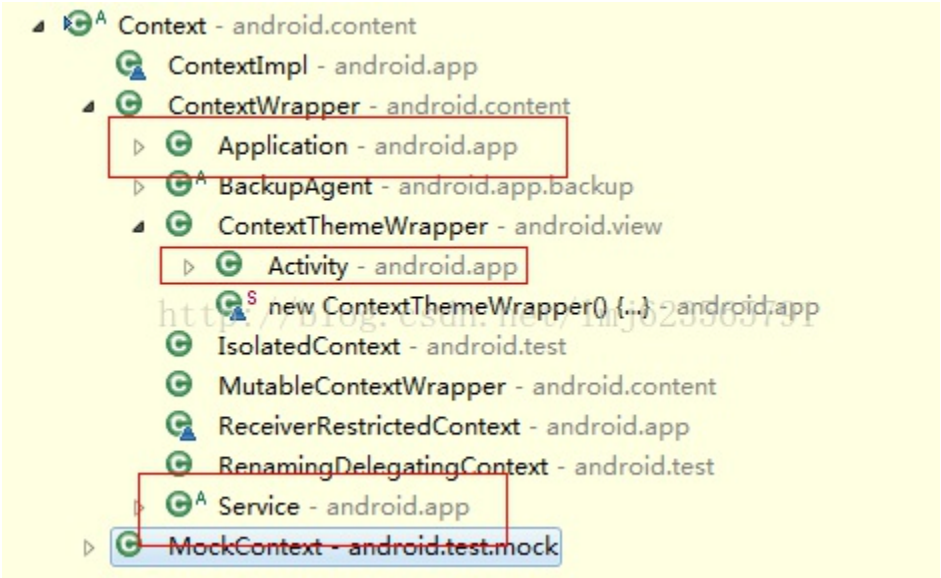
Context，中文直译为“上下文”，SDK中对其说明如下：

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc

从上可知一下三点,即：

- 它描述的是一个应用程序环境的信息，即上下文。
- 该类是一个抽象(abstract class)类，Android提供了该抽象类的具体实现类(后面我们会讲到是ContextIml类)。
- 通过它我们可以获取应用程序的资源和类，也包括一些应用级别操作，例如：启动一个Activity，发送广播，接受Intent信息 等

首先看它们的继承关系



2. 什么时候创建Context实例

熟悉了Context的继承关系后，我们接下来分析应用程序在什么情况需要创建Context对象的？应用程序创建Context实例的情况有如下几种情况：

1) 创建Application 对象时， 而且整个App共一个Application对象 2) 创建Service对象时 3) 创建Activity对象时

因此应用程序App共有的Context数目公式为：

> 总Context实例个数 = Service个数 + Activity个数 + 1（Application对应的Context实例）

1、创建Application对象的Context: 首先新建一个MyApplication并让它继承自Application，然后在AndroidManifest.xml文件中对MyApplication进行指定，如下所示：

```
<application
    android:name=".MyApplication"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ......
</application>
```

指定完成后，当我们的程序启动时Android系统就会创建一个MyApplication的实例,通过如下代码获取到它的实例:

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MyApplication myApp = (MyApplication) getApplication();
        Log.d("TAG", "getApplication is " + myApp);
    }

}
```

可以看到，代码很简单，只需要调用getApplication()方法就能拿到我们自定义的Application的实例了，打印结果如下所示：

```
Text
getApplication is com.example.androidtest.MyApplication@a428db3
```

那么除了getApplication()方法，其实还有一个getApplicationContext()方法，这两个方法看上去好像有点关联，那么它们的区别是什么呢？我们将代码修改一下：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MyApplication myApp = (MyApplication) getApplication();
        Log.d("TAG", "getApplication is " + myApp);
        Context appContext = getApplicationContext();
        Log.d("TAG", "getApplicationContext is " + appContext);
    }
```

```
}
```

同样，我们把getApplicationContext()的结果打印了出来，现在重新运行代码，结果如下图所示：

```
Text
getApplication is com.example.androidtest.MyApplication@a428db3
getApplicationContext is com.example.androidtest.MyApplication@a428db3
```

打印出的结果是一样的呀，连后面的内存地址都是相同的，看来它们是同一个对象。其实这个结果也很好理解，Application本身就是一个Context，所以这里获取getApplicationContext()得到的结果就是MyApplication本身的实例。

那么有的朋友可能就会问了，既然这两个方法得到的结果都是相同的，那么Android为什么要提供两个功能重复的方法呢？实际上这两个方法在作用域上有比较大的区别。getApplication()方法的语义性非常强，一看就知道是用来获取Application实例的，但是这个方法只有在Activity和Service中才能调用的到。那么也许在绝大多数情况下我们都是在Activity或者Service中使用Application的，但是如果在一些其它的场景，比如BroadcastReceiver中也想获得Application的实例，这时就可以借助getApplicationContext()方法了，如下所示：

```
public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        MyApplication myApp = (MyApplication) context.getApplicationContext();
        Log.d("TAG", "myApp is " + myApp);
    }

}
```

也就是说，getApplicationContext()方法的作用域会更广一些，任何一个Context的实例，只要调用getApplicationContext()方法都可以拿到我们的Application对象。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

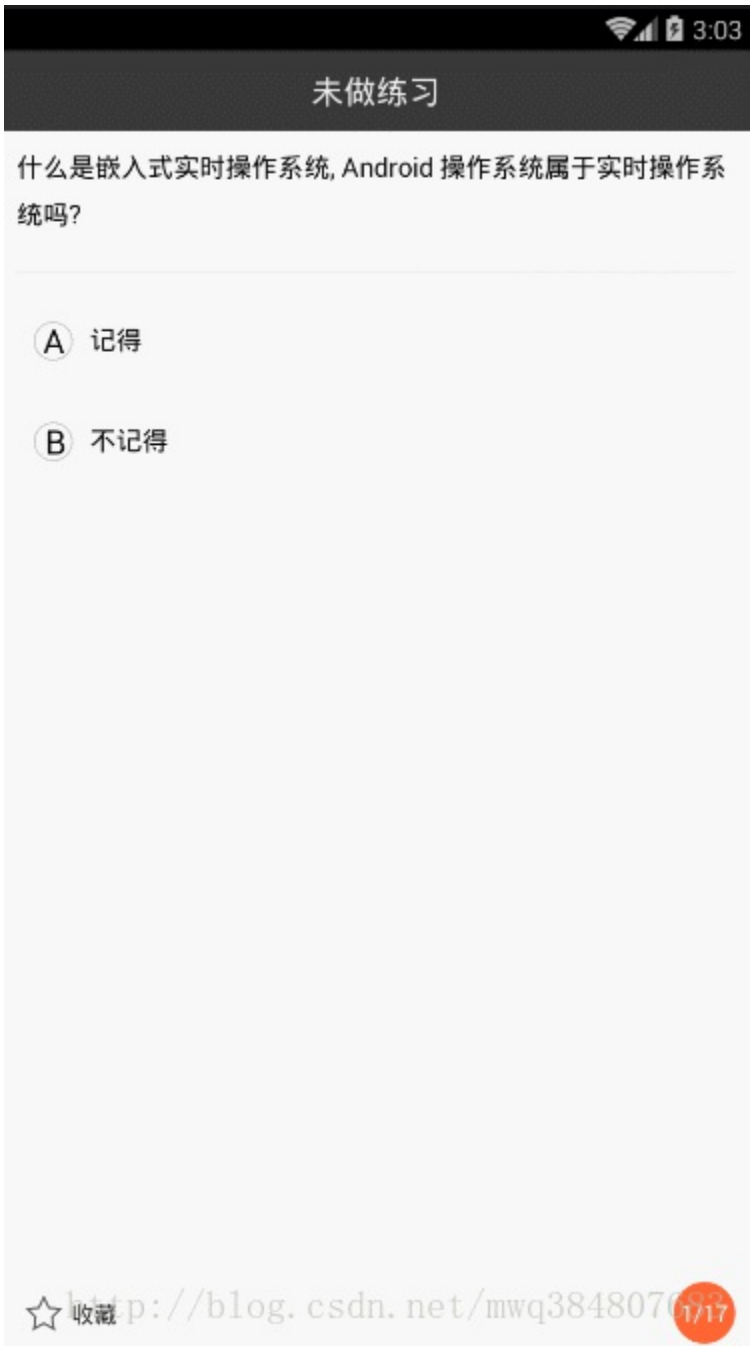
与人事相关面试题

- 人事面试宝典

本文配套视频

- ViewPager.setCurrentItem的bug演示一
- ViewPager.setCurrentItem解决方案二

今天做项目用ViewPager.setCurrentItem 方法，如果两个页面相聚比较远，就会闪瞎我的钛合金双眼，中间切换大概20个页面，如下所示：



setCurrentItem第二个参数设置false，四不四很简单，直接使用如下代码：

```
ViewPager.setCurrentItem(position,false);
```

很不幸的是，使用上面的代码会出现如下效果，扎心了老铁：

从第一题点击切换到第十八题，你会发现页面显示空白，如果从第十个页面切换到第十五个页面没事，平时大家估计没有发现这个bug，一般我们使用ViewPager都是底下5个tab页面，从第一个切换到第五个没事，之前我也以为把第二个参数设置false就行，今天才发现，原来如果当页面比较少的时候，大概十个以内，一般没有问题，如果超过十个页面切换就会出现空白，加载不了数据，扎心了，提出解决方案吧，ViewPager滑动使用的是Scroll，咱们把Scroll的滑动时间duration 设置为0就行。

自定义一个Scroll类，用于控制ViewPager滑动速度：

```
public  class MScroller extends Scroller {

   private static final Interpolator sInterpolator = new Interpolator() {
   public float getInterpolation(float t) {
            t -= 1.0f;
            return t * t * t * t * t + 1.0f;
        }
    };
```

```
  public boolean noDuration;

  public void setNoDuration(boolean noDuration) {
        this.noDuration = noDuration;
    }

  public MScroller(Context context) {
        this(context,sInterpolator);
  }

  public MScroller(Context context, Interpolator interpolator) {
        super(context, interpolator);
  }

    @Override
  public void startScroll(int startX, int startY, int dx, int dy, int duration) {
        if(noDuration)
            //界面滑动不需要时间间隔
            super.startScroll(startX, startY, dx, dy, 0);
        else
            super.startScroll(startX, startY, dx, dy,duration);
    }
}
```

上面代码可知：

1）动态判断页面是否需要滑动，如果不需要滑动，设置滑动时间为0;

为方便使用，定义一个辅助类

```
public class ViewPageHelper {
    ViewPager viewPager;

    MScroller scroller;

    public ViewPageHelper(ViewPager viewPager) {
        this.viewPager = viewPager;
        init();
    }

    public void setCurrentItem(int item){
        setCurrentItem(item,true);
    }

    public MScroller getScroller() {
        return scroller;
    }


    public void setCurrentItem(int item, boolean somoth){
        int current=viewPager.getCurrentItem();
        //如果页面相隔大于1,就设置页面切换的动画的时间为0
        if(Math.abs(current-item)>1){
            scroller.setNoDuration(true);
            viewPager.setCurrentItem(item,somoth);
            scroller.setNoDuration(false);
        }else{
            scroller.setNoDuration(false);
            viewPager.setCurrentItem(item,somoth);
        }
```

```
    }

    private void init(){
        scroller=new MScroller(viewPager.getContext());
        Class<ViewPager>cl=ViewPager.class;
        try {
            Field field=cl.getDeclaredField("mScroller");
            field.setAccessible(true);
            //利用反射设置mScroller域为自己定义的MScroller
            field.set(viewPager,scroller);
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        }catch (IllegalAccessException e){
            e.printStackTrace();
        }
    }

}
```

由上面代码可知：

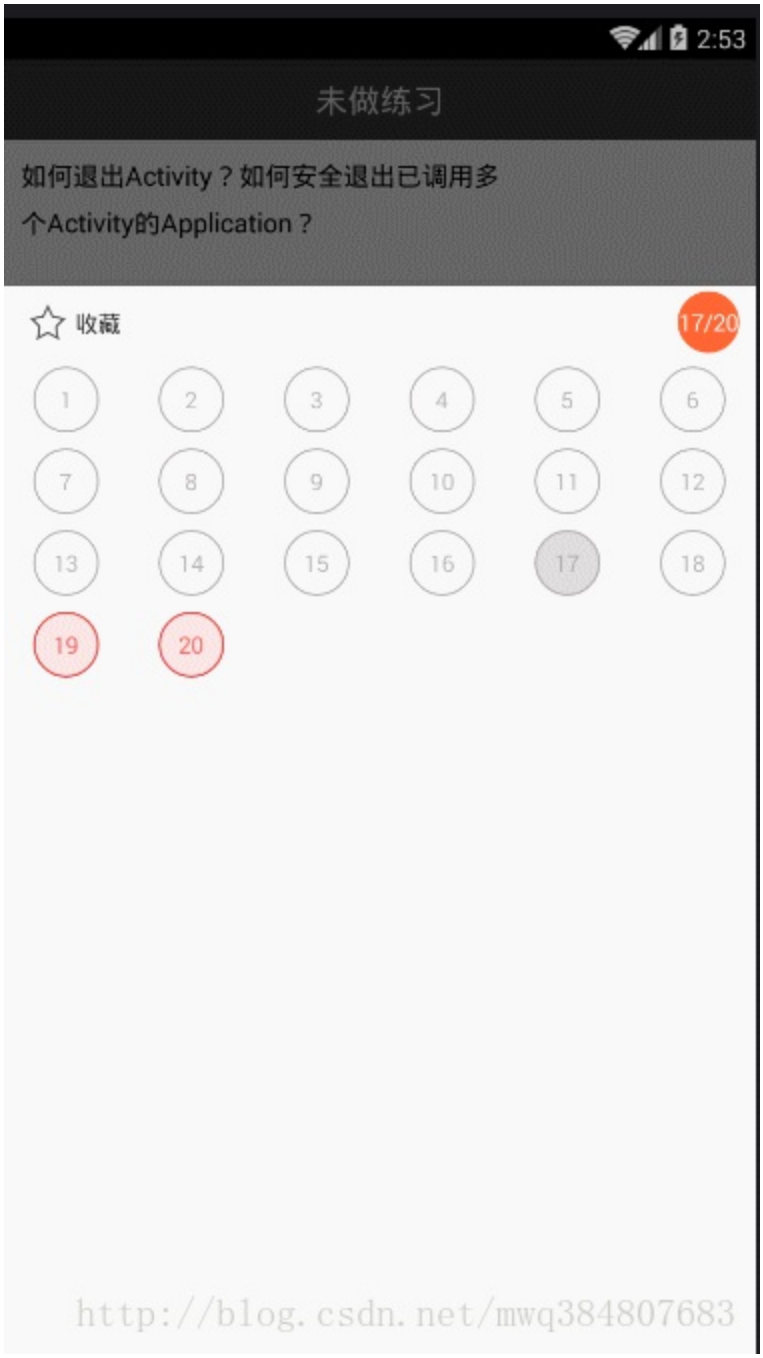
1）Math.abs(current-item)>1 ，通过数学函数判断页面相隔大于1,就设置页面切换的动画的时间为0。

2）这样每次设置页面的时候，通过 helper 就可以自动选择是否有时间间隔了。

3）但是这样有点麻烦，每次还要手动改，而且使用TabLayout或者ViewPagerIndicator的话，它会自动调用ViewPager的方法，无法使用Helper，所以可以采用自定一个ViewPager,代码如下：

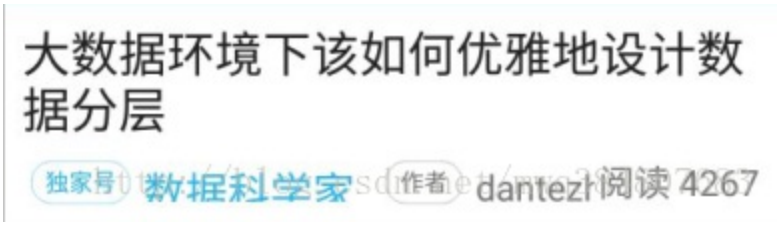
```
public class SuperViewPager extends ViewPager {


    private ViewPageHelper helper;

    public SuperViewPager(Context context) {
        this(context,null);
    }

    public SuperViewPager(Context context, AttributeSet attrs) {
        super(context, attrs);
        helper=new ViewPageHelper(this);

   }

    @Override
    public void setCurrentItem(int item) {
        setCurrentItem(item,true);
    }

    @Override
    public void setCurrentItem(int item, boolean smoothScroll) {
        MScroller scroller=helper.getScroller();
        if(Math.abs(getCurrentItem()-item)>1){
            scroller.setNoDuration(true);
            super.setCurrentItem(item, smoothScroll);
            scroller.setNoDuration(false);
        }else{
            scroller.setNoDuration(false);
            super.setCurrentItem(item, smoothScroll);
        }
    }
}
```

至此完美解决了，ViewPager.setCurrentItem切换页面，效果如下：



- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

Android面试题-解决字体适配



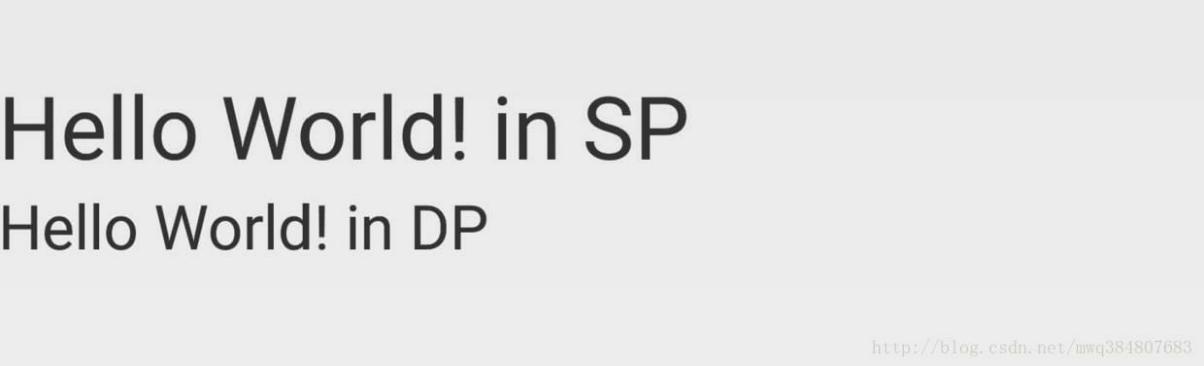
做个简单的例子，先验证一下：

同样的布局代码

```
<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="18sp"
 android:text="Hello World! in SP" />

<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="18dp"
 android:text="Hello World! in DP" />
```

调节设置中显示字体大小

运行后显示样式



回到标题要解决的问题，如果要像微信一样，所有字体都不允许随系统调节而发生大小变化，要怎么办呢？利用Android的Configuration类中的fontScale属性，其默认值为1，会随系统调节字体大小而发生变化，如果我们强制让其等于默认值，就可以实现字体不随调节改变，在工程的Application或BaseActivity中添加下面的代码：

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    if (newConfig.fontScale != 1)//非默认值
        getResources();
    super.onConfigurationChanged(newConfig);
}

@Override
public Resources getResources() {
     Resources res = super.getResources();
     if (res.getConfiguration().fontScale != 1) {//非默认值
        Configuration newConfig = new Configuration();
        newConfig.setToDefaults();//设置默认
        res.updateConfiguration(newConfig, res.getDisplayMetrics());
     }
     return res;
}
```

总结，两种方案解决这个问题： 一是布局宽高固定的情况下，字体单位改用dp表示； 二是通过3中的代码设置应用不能随系统调节，在检测到fontScale属性不为默认值1的情况下，强行进行改变。

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态
- 深刻剖析activity启动模式(一)
- 深刻剖析activity启动模式(二)
- 深刻剖析activity启动模式(三)
- service里面startActivity抛异常？activity不会

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

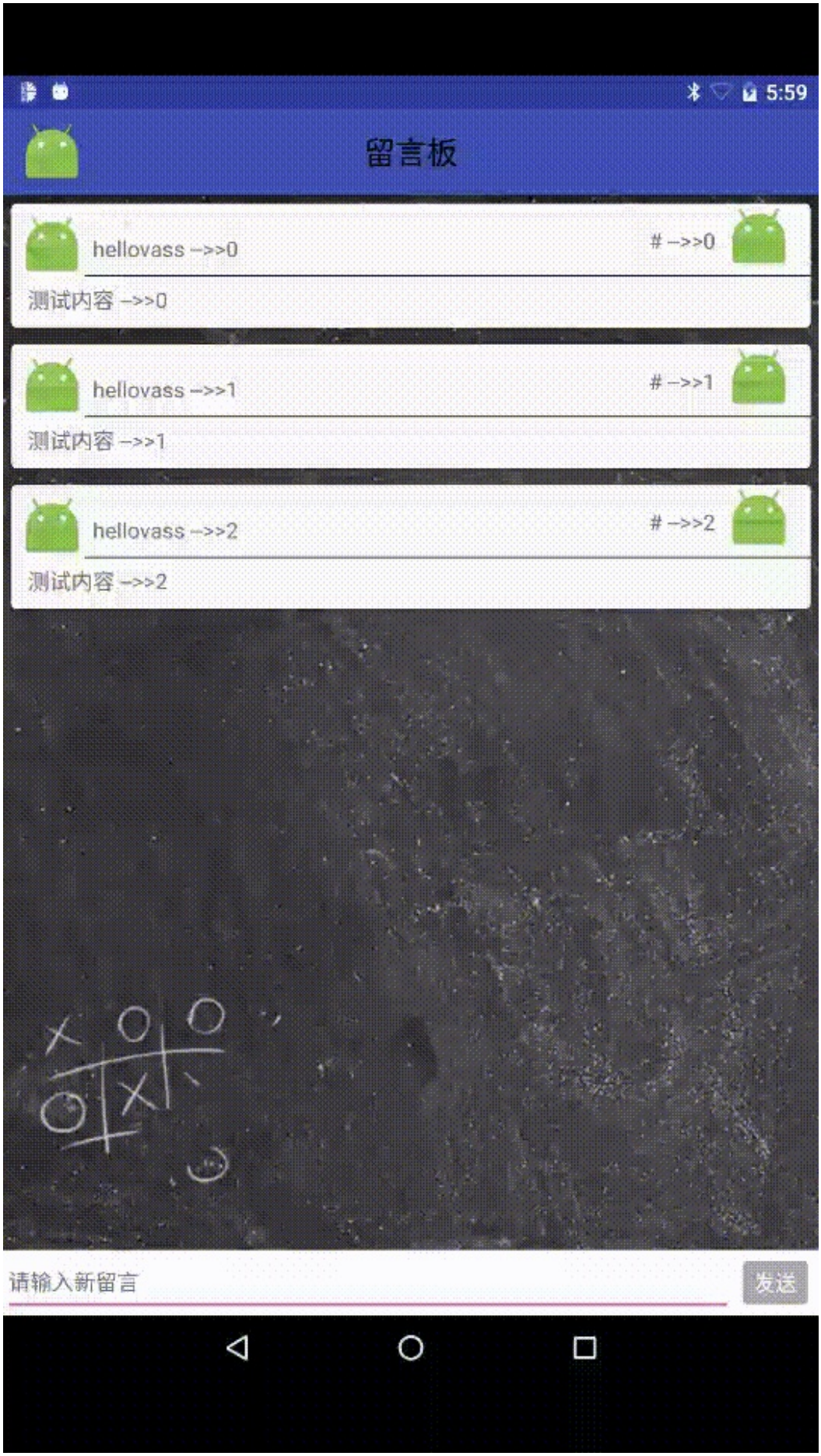
与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

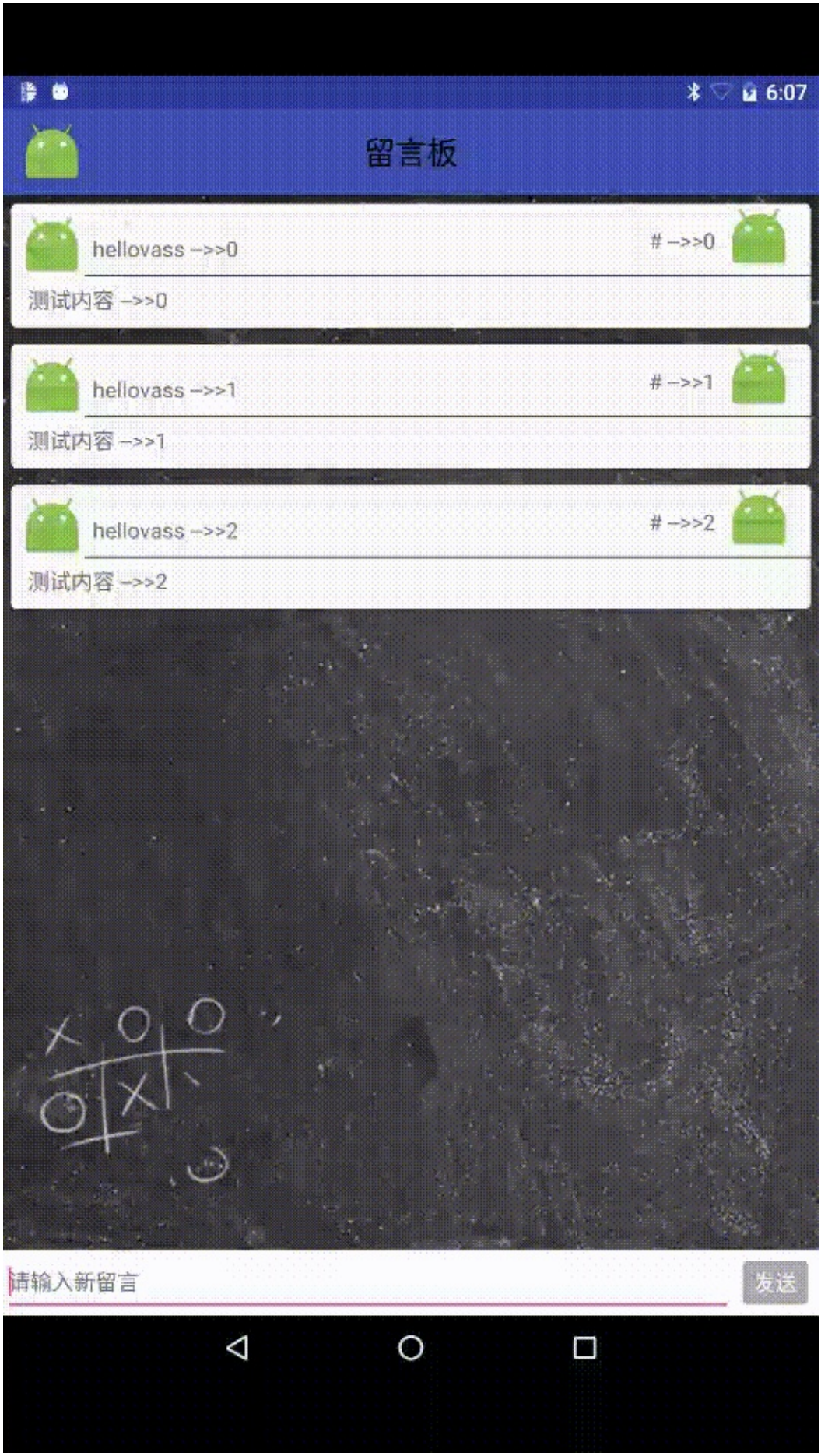
- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案
- 字体适配
- 软键盘顶出去解决方案与人事相关面试题
- 人事面试宝典

错误的打开方式


5:59
留言板
hellovass –>>0
–>>0
测试内容 –>>0
hellovass –>>1
–>>1
测试内容 –>>1
hellovass –>>2
–>>2
测试内容 –>>2
请输入新留言
发送

错误的打开姿势.gif

仔细观察会发现，当软键盘弹出时 **background、recyclerview、toolbar** 被**软键盘**顶上去了！这样的交互简直不能忍，对用户来说也非常突兀。 正确的打开方式


6:07
留言板
hellovass –>>0
–>>0
测试内容 –>>0
hellovass –>>1
–>>1
测试内容 –>>1
hellovass –>>2
–>>2
测试内容 –>>2
请输入新留言
发送

正确的打开姿势.gif

软键盘弹出只是遮盖了 **background** 一部分，**background** 没有被**压缩**。

实现

1. AndroidMainifest.xml 配置文件

```
<activity
   android:name=".MainActivity"
   android:windowSoftInputMode="adjustResize">
```

   非透明状态栏下使用adjustResize和adjustPan，或是透明状态栏下使用 `fitsSystemWindows=true` 属性

   主要实现方法： 在AndroidManifest.xml对应的Activity里添加 `windowSoftInputMode=”adjustPan”` 或是 `android:windowSoftInputMode=”adjustResize”` 属性

推荐一篇软键盘很好的文章：http://blog.csdn.net/smileiam/article/details/69055963

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

由于开源三方定制系统较多，请大家详细描述场景、机型及解决方案，方便其他朋友参考

[问答]-Android开发中有哪些兼容性问题？都是怎么解决的？ [问答] 你在工作中遇到的最复杂的问题或者bug是什么？你是怎么搞定的?

华为P6和P7

场景：使用MIPush，在华为部分手机上无法推送成功。 机型：[华为P6，华为P7] 解决方案：P6和P7是华为的高端机型，不允许推送，防止骚扰用户，无解。

魅族3和魅族4

场景：魅族手机ListView的Item中的EditText无法编辑，点击EditText弹出软键盘后，软键盘会立即自动隐藏 机型：[魅族3，魅族4] 解决方案： 方法一：将ListView换成RecyclerView

方法二：https://github.com/Aspsine/EditTextInListView

HTC M8

场景：HTC M8 从一个Activity 使用QQSDK 登陆, 登陆成功后, 返回Activity结果Activity 被销毁了 机型：HTC M8 等某些带有 虚拟 Menu 键盘的手机 解决方案：后来调查发现是这个Activity是全屏,屏蔽了Menu键盘的黑条. 但是跳转到QQ却把那个Menu的黑条显示了出来, 这导致发生了 screenSize 的变化 从而导致我的Activity销毁了. 知道了这个原因, 在manifest中的 configChanges 添加screenSize 解决了这个问题.

所有android4.4机型

场景：Android4.4系统使用了SystemBarTintManager库修改透明状态栏后，会导致根布局从屏幕顶端开始布局，而不是从ActionBar开始布局 机型：所有android4.4机型 解决方案： 方法一：针对4.4创建一套额外的布局，即layout-v19文件夹，并且在根布局外层再套一层LinearLayout，并在LinearLayout中添加一个属性 android:fitsSystemWindows="true"

方法二：是为4.4及以上添加了paddingTop去适配，添加layout觉得不好适配。

方法三：在Build.VERSION.SDK_INT <= 18的版本中，通过colorDrawable.setAlpha(alpha);设置actionbar背景色透明度的时候，colorDrawable需要设置callback。

```
final Drawable.Callback mDrawableCallback = new Drawable.Callback() {
   @Override
   public void invalidateDrawable(Drawable who) {
      getActionBar().setBackgroundDrawable(who);
   }

   @Override
   public void scheduleDrawable(Drawable who, Runnable what, long when) {
   }

   @Override
   public void unscheduleDrawable(Drawable who, Runnable what) {
   }
 };
    colorDrawable.setCallback(mDrawableCallback);
```

魅族MX3

场景：Camera拍摄，用setPreviewFormat设置成YV12，预览会变成绿屏，实际用getPreviewFormat显示是支持YV12的 机型：魅族MX3 解决方案：没办法只能设置成NV21了

其代表机型为：三星I8258、华为H30-T00、红米等。

Camera常见问题： 1）Intent调用手机内相机程序

```
//创建一个Action为MediaStore.ACTION_IMAGE_CAPTURE的Intent
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
//指定拍照后照片的存储路径
intent.putExtra(MediaStore.EXTRA_OUTPUT, picFile);
//指定照片的质量
intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 100);
//启动系统相机
startActivityForResult(intent,RESULT_CAMERA);
```

如果我们设置了照片的存储路径，那么很可能会遇到一下三种问题：

问题一：onActivityResult方法中的data返回为空（数据表明，93%的机型的data将会是Null，所以如果我们指定了路径，就不要使用data来获取照片，起码在使用前要做空判断）。 问题二：照片无法存储。 如果自定义存储路径是/mnt/sdcard/lowry/，而手机SD卡下在拍照前没有名为lowry的文件夹，那么部分手机拍照后图片不会保存，导致我们无法获得照片，大多数手机的相机遇到文件夹不存在的情况都会自己创建出不存在的文件夹，而个别手机却不会创建，

解决的方法就是在指定存储路径前先判断路径中的文件夹是否都存在，不存在先创建再调用相机。

问题三：照片可以存储，但是名字不对。 file:///mnt/sdcard/123 1.jpg，由于URI的fromFile方法会将路径中的空格用“%20”取代。

其实对于大多数的手机这都不算事，手机在解析存储路径的时候都会将“%20”替换为空格，这样实际上最终的照片名字还是我们当初指定的名字：123 1.jpg，遗憾的是个别手机（如酷派7260）系统自带的相机没有将“%20”读成空格，拍照后的照片的名字是123%201.jpg，我们用路径“file:///mnt/sdcard/123 1.jpg”能找到照片才怪!

```
picFile = new File(ToolUtil.getStorePath(""), "/123 1.jpg");
Log.i("lowry", "===========Uri.fromFile(picFile)==========" +Uri.fromFile(picFile).toString());
```

```
===========Uri.fromFile(picFile)==========file:///mnt/sdcard/123%201.jpg
```

总结：

（1）使用onActivityResult中的intent(data)前要做空判断。 （2）指定拍照路径时，先检查路径中的文件夹是否都存在，不存在时先创建文件夹再调用 相机拍照。 （3）指定拍照存储路径时，照片的命名中不要包含空格等特殊符号。

所有手机

PopupWindow中嵌套EditText，会出现EditText长按无法触发“粘贴”选项，可以改成Dialog嵌套EditText，包括DialogFragment。

三星Note系列,S系列

会调用Activity的onPause和onStop方法.其他手机会保持在onResume状态

酷派8720L

场景：在获取系统相机拍照然后保存在本地有时候会保存不上，获取不到地址。 问题原因：通过调试发现当拍完照返回的时候自己设的成员变量值会被回收，估计就是内存不足的原因。重启机器后就好了。 解决方案：无方案。

所有手机

场景：输入法中的emoji适配，Android4.1之前的系统不支持emoji显示

解决方案：所以对于Android4.1之前的系统，我采用了bitmap来显示emoji。

三星手机

问题： (1) 摄像头拍照后图片数据不一定能返回 ; onActivityResult的data为空 (2) 三星的camera强制切换到横屏 导致Activity重启生命周期 (但是部分机型 配置 android:configChanges 也不能阻止横竖屏切换); (3) APP Activity A调用系统拍照 --> 拍照 --> 在拍好照片的界面做几次横竖屏转换 --> 返回APP界面Activity A ， A 被销毁。

解决方案：如果 activity 的销毁如果无法避免 那么在activity销毁之前调用 onSaveInstanceState 保存图片的路径 当activity重新创建的时候 会将 onSaveInstanceState 保存的文件传递给onCreate()当中 在onCreate当中 检查照片的地址是否存在文件 以此来判定拍照是否成功 Demo 下载地址: http://download.csdn.NET/detail/aaawqqq/7653475

OPPO 手机

场景：OPPO 手机启动 Service 报 SecurityException 解决方案：try catch 该异常

HTC Desire 820、Lenovo A320T

场景：这个问题主要在部分机型的4.X系统上遇见，小图标大小没有按照24dp裁剪，而是采用了桌面图标一样的大小96dp


4:54
2016年11月8日
星期二
大图推送
2016 9月 18, 周日
普通消息，带大图
9月16日 11:00
考拉海购直播
丁爸爸直播首秀
iPhone7终极防水测试
立即围观
iPhone7
奥测
iPhone7
水下开启
#杭州身边事#
下午1:47
杭州:妻子在隔壁剖腹产,他在手术台上紧张...
考拉双11来啦
上午10:45
德爱包税95/罐起，美迪惠尔水库面膜5元...
11.11狂欢❀严选...
2016 11月 7, 周一
11月8日0点 正式开抢，全场特惠低至8折...
没有 SIM 卡
http://blog.csdn.net/mwq384807683

解决方案：按照标准来，小图标大小为24dp，大图标为桌面icon图标大小96dp

魅族5.X手机，大图显示问题

场景：Flyme系统对原生Android源码做了修改，采用BigPictureStyle方式显示大图通知栏的时候，消息与大图重合了，如下图。

解决方案

首先，通过BigPictureStyle来实现大图功能肯定是走不通的，因为事实就摆着行不通的嘛。京东的App肯定是通过RemoteViews来实现的。于是，开始走弯路，尝试通过RemoteViews来展示大图。但是谷歌规定，自定义布局展示的通知栏消息最大高度是64dp。那么，京东的App是怎么实现的？在尝试了各种方法以后，最后又是通过投机取巧的方式解决了问题

```
private void showBigPictureNotificationWithMZ(Context context) {
    NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SER
VICE);
    Notification.Builder builder = new Notification.Builder(context);
    Notification notification = generateNotification(builder);
    notification.bigContentView = mRemoteViews;
    notificationManager.notify(notifyId, notification);
}
```

需要先生成Notification的实例，然后手动给notification.bigContentView赋值，再notify，就可以了

问题二：顶部状态栏(StatusBar)小图标显示异常

场景：当通知来的时候，如果不在通知栏浏览，会在顶部状态栏出现一个向上翻滚动画的通知消息，这条通知消息左边是一个小图标。部分系统这个小图标显示异常，是一个纯灰色的正方形，如下图。


09:43
男士t恤
推荐
活动
全球旗舰
网易严选
母婴
美妆
箱包
共享全世界的实惠好物
拼团买好货
全场包邮1元起团
每日签到
限时购
VIP
会员专享
拼团
立省30元
周末特卖
仅此2天
嗲货好价限量抢
周末
首页
分类
发现
购物车
我的考拉

解决方案

首先产生灰色图标的原因就是5.0系统引入了材料设计，谷歌强制使用带有alpha通道的图标，并且RGB的alpha值必须是0(实测不为0也是可以的，但系统会忽略所有RGB值)。因此，使用JPG的图片是不行的，最好的代替方案就是一张背景透明的PNG图片。

问题三：Android 7.X机型，通知栏小图标显示成灰色

问题详情

这个问题跟第二个有点类似，在7.0系统及以上，有部分应用的小图标是灰色的，大图可以正常显示。碰巧的是，显示异常的小图标，颜色都是灰色的。


11:41
0 KB/s Internet Speed Meter Lite
Speed: 0 B/s "NetEase-5G" 97%
Mobile: 0 MB WiFi: 60.2 MB
网易考拉海购
普通推送
普通消息，不带大图
网易考拉海购
普通推送
普通消息，带普通emoji表情
Google Play Store • 1m
2 updates available
V2EX+ and Snapseed
Android System
USB debugging connected
Touch to disable debugging.
Tap for more options.
打开JSBridge测试页面
http://blog.csdn.net/mwq384807683

解决方案

与小图标显示异常解决方案类似，将小图标替换为透明背景的PNG图片。

问题四：RemoteViews显示异常

问题详情

由于系统提供的通知栏消息类型有时候不能满足要求，部分通知栏消息采用自定义RemoteViews来实现。采用RemoteViews，特别是手动生成Bitmap然后直接传给一个自定义Layout，再通过setContentView方式设置通知栏消息时，会存在各种各样的坑。

Android通知栏的背景色有几种情况，白色、暗色、暗色透明和黑色。如果生成的Bitmap带背景色，这个背景色就很难选择。如果选择黑色背景，那么在白色通知栏的机型上就很难看。因此不能完全在各个系统上面完美展示出来。如果不带背景色，那么字体颜色也面临同样的困惑。试想，如果在白色的背景上显示白色的文字，用户看到白茫茫一片，是什么感受?

另一方面，大部分厂商对原生的Android系统都会有各种各样的改造，通知栏的样式也不例外。如果按照原生的样式来设计，那么在大部分国内厂商的机子上显示都和正常的普通通知栏消息不一样。例如华为6.0系统的机子，原生系统的时间线在右上角，华为的在左边，这样会给用户带来错觉。


通知
开关
时间
15:41
2016/11/9
星期三
15:20
大图推送
普通消息，带大图
9月16日 11:00
考拉海购直播
丁爸爸直播首秀
iPhone7终极防水测试
立即围观
iPhone7
水下开启
iPhone7
实测
15:40
普通推送
09-18
普通消息，带自定义emoji表情
15:20
普通推送
普通消息，不带大图
未插卡
http://blog.csdn.net/mwq384807683

解决方案

详见RemoteViews适配一节。

问题五：通知栏更新频率

问题详情

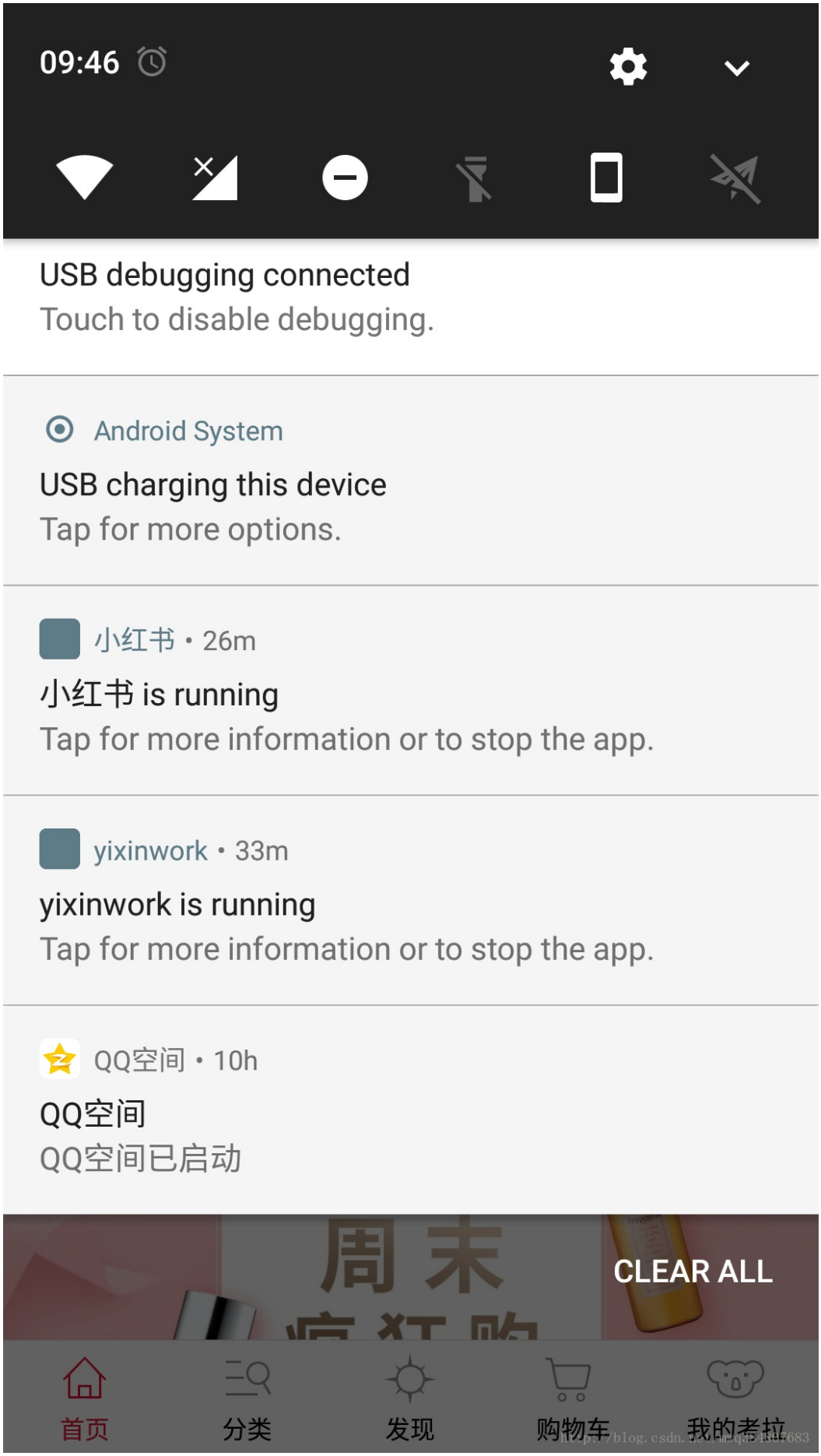
每个应用基本都有自更新的逻辑，App开机的时候提示用户升级，点击升级按钮后在Notification出现一个下载带进度条的通知。应用一般是在开启一个工作线程在后台下载，然后在下载的过程中通过回调更新通知栏中的进度条。我们知道，下载进度的快慢是不可控的，如果每次下载中的回调都去更新通知栏，那么可能几百毫秒、几十毫秒、甚至几毫秒就更新一次通知栏，应用可能就会ANR，甚至崩溃。

解决方案

控制通知栏更新频率，一般控制在0.5s或者1s就可以了。在某一个更新时间间隔内下载的进度回调直接丢弃，需要注意的是下载完成的回调，需要实时回调通知栏消息显示下载完成。

问题六：恶心的后台通知和“守护”通知 问题详情 但凡存在后台通知或者“守护”通知的应用，在7.0系统以后都会原形毕露.


09:44
NetEase News • 8m
女作家环游世界到印度遭奸杀
[早读]不排除战争?美国务卿对朝表态卖的什么药；女作家环游世界到印度第2天遭奸杀；中纪委副书记狠批"商人谋权"在指谁>>
贝贝母婴正品特卖 • 34m
曝光！这些商品都...
早教机9.9元！儿童套装19元！湿巾19元！周末特价..
0 KB/s
Internet Speed Meter Lite
Speed: 0 B/s "1703" 97%
Mobile: 0 MB WiFi: 50 MB
贝贝母婴正品特卖 • 7m
Push Service
Push Service
小红书 • 21m
点击查看详情
首页
分类
发现
购物车
我的考拉


09:46
USB debugging connected
Touch to disable debugging.
Android System
USB charging this device
Tap for more options.
小红书 • 26m
小红书 is running
Tap for more information or to stop the app.
yixinwork • 33m
yixinwork is running
Tap for more information or to stop the app.
QQ空间 • 10h
QQ空间
QQ空间已启动
周末
CLEAR ALL
首页
分类
发现
购物车
我的考拉

解决方案:无

小米推送SDK接入问题

问题详情

为了提升推送到达，考拉接入了小米推送的SDK。小米推送分为通知栏消息和透传消息，通知栏消息属于系统级推送，在MIUI的机子上可以在进程被杀死的情况下也能收到应用推送。然而有个问题，小米认为应用在前台时，不会回调任何方法；小米认为应用在后台的时候，收到通知栏消息的同时，会回调onNotificationMessageArrived方法。这时候就要小心翼翼地处理这条消息了。因为如果你的应用前后台判断逻辑和小米的不一样，那么就有可能小米帮你发了一条通知栏消息，你自己又发了一遍，造成通知栏消息的重复发送(这个坑考拉踩过T_T)。另一方面，在7.0系统的机子上，主标题和小图标的颜色是可以改变的，目前小米推送SDK没有开放这个接口供调用方定制。

解决方案

目前只能解决第一个问题——前后台判断的问题。应用是否在后台可以根据以下代码进行判断。在Android 5.0以上，可以通过ActivityManager.RunningAppProcessInfo判断，Android 5.0及以下版本通过ActivityManager.RunningTaskInfo判断。经测试，这个方案在Android 4.4以上结果是可以完全匹配的。

```
public static boolean isAppInBackgroundInternal(Context context) {
    ActivityManager manager = (ActivityManager) context.getSystemService(Context.ACTIVITY_SERVICE);
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP) {
        List<ActivityManager.RunningAppProcessInfo> runningProcesses = manager.getRunningAppProcesses();
    if (!ListUtils.isEmpty(runningProcesses)) {
    for (ActivityManager.RunningAppProcessInfo runningProcess : runningProcesses) {
      if (runningProcess.importance == ActivityManager.RunningAppProcessInfo.IMPORTANCE_FOREGROUND) {
                    return false;
                }
            }
        }
    } else {
        List<ActivityManager.RunningTaskInfo> task = manager.getRunningTasks(1);
        if (!ListUtils.isEmpty(task)) {
            ComponentName info = task.get(0).topActivity;
            if (null != info) {
                return !isKaolaProcess(info.getPackageName());
            }
        }
    }
    return true;
}
```

Android通知栏适配

RemoteViews适配 由于系统自带的通知栏消息样式不能完全满足产品们脑洞大开的需求，有时候我们需要自定义布局样式展示通知栏消息。Android系统可以将自定义布局通过setContent(7.X系统推荐使用setCustomContentView)设置到Notification.Builder中，来实现样式的更变。setContent方法需要传入一个RemoteViews对象，它是一个普通的数据类型，不是View，作用是供其他进程展示视图。RemoteViews只支持4种基本的布局:

FrameLayout LinearLayout RelativeLayout GridLayout 这些布局下面只支持几种视图控件:

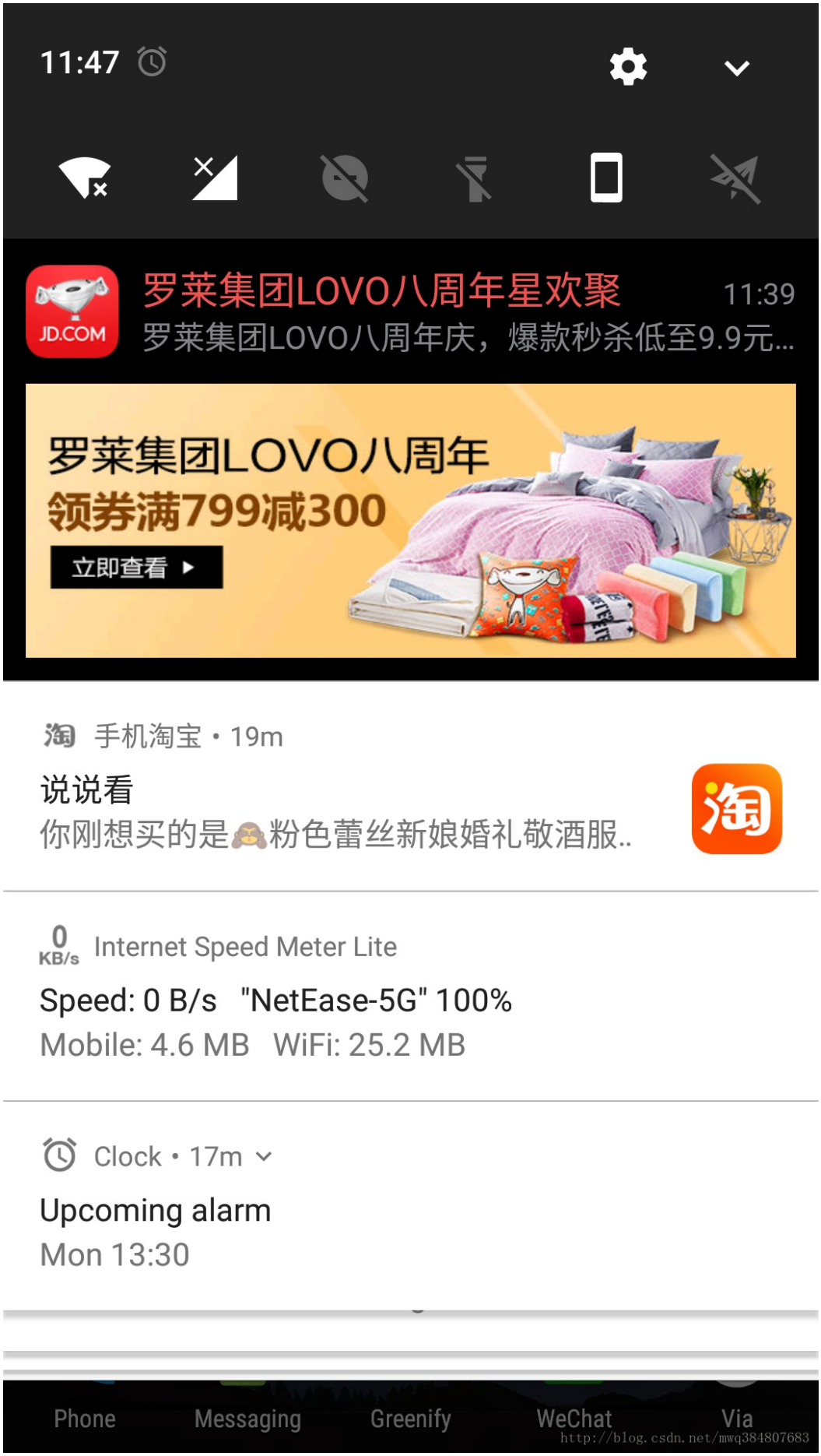
AnalogClock Button Chronometer ImageButton ImageView ProgressBar TextView ViewFlipper ListView GridView StackView AdapterViewFlipper 只能通过上述组合生成一个RemoteViews。

自定义布局与视图

除了上面提到的布局与控件，有没有办法自定义布局与视图呢？我们知道，任何一个View，都可以生成一个Bitmap对象，支持的视图控件里有ImageView，可以通过ImageView.setBitmapResource()将自定义视图设置到一个ImageView中，然后再随便放到一个布局上，就可以实现通知栏消息的任意布局。理想是美好的，但现实是残酷的。使用这种方式自定义的布局，会存在与原生的通知栏消息样式不一致的可能，包括小图标/大图标的大小，字体的大小与颜色，时间的显示方式(不同版本的时间显示位置和样式都不一样)。下面解决一个最关键，也最致命的问题——字体颜色。如果字体颜色和背景颜色一样，那这条通知栏消息就没法看了，如RemoteViews显示异常一节介绍的一样。

解决字体颜色和背景颜色一样的问题有三种解决方案，分别是:

背景色固定不透明，字体颜色与背景色形成反差。（360和京东的做法） 背景色透明，字体颜色采用系统原生的notification_style。 背景色透明，通过特殊方式拿到通知栏字体颜色和字体大小。


11:47
罗莱集团LOVO八周年星欢聚
11:39
JD.COM
罗莱集团LOVO八周年庆，爆款秒杀低至9.9元...
罗莱集团LOVO八周年
领券满799减300
立即查看
手机淘宝 • 19m
说说看
你刚想买的是粉色蕾丝新娘婚礼敬酒服..
0 KB/s
Internet Speed Meter Lite
Speed: 0 B/s "NetEase-5G" 100%
Mobile: 4.6 MB WiFi: 25.2 MB
Clock • 17m
Upcoming alarm
Mon 13:30
Phone
Messaging
Greenify
WeChat
Via
http://blog.csdn.net/mwq384807683

其中，第一种方案简单，能够兼容所有厂商机型。例如京东固定背景色为黑色，字体为红色。这种方式的唯一缺陷是样式上不能与普通通知栏消息重合，在白色背景的通知栏上极为显眼。第二种方式，通过阅读源码可知，系统的通知栏标题和内容采用的颜色分别是@android:color/primary_text_dark和@android:color/secondary_text_dark，但踩过坑之后发现并非所有的机型默认都是这两个颜色，有可能获取不到值。因此这种方案只能作为参考，不能用于实际环境中。最后详细介绍一下第三种方式。

Android默认字体颜色获取

这种方案有一点投机取巧，是网上寻找代替方案时在简书上找到的，作者是hackware。思路就是通过Notification.Builder生成一条空的Notification，但不调用notify()方法，然后通过这条Notification想办法获取里面的布局元素，通过遍历，就能拿到对应的字体和颜色了。具体看代码:

```
private static final String NOTIFICATION_TITLE = "notification_title";
public static final int INVALID_COLOR = -1; // 无效颜色
private static int notificationTitleColor = INVALID_COLOR; // 获取到的颜色缓存
/**
 * 获取系统通知栏主标题颜色，根据Activity继承自AppCompatActivity或FragmentActivity采取不同策略。
 *
 * @param context 上下文环境
 * @return 系统主标题颜色
 */
public static int getNotificationColor(Context context) {
    try {
        if (notificationTitleColor == INVALID_COLOR) {
            if (context instanceof AppCompatActivity) {
                notificationTitleColor = getNotificationColorCompat(context);
            } else {
                notificationTitleColor = getNotificationColorInternal(context);
            }
        }
    } catch (Exception ignored) {
    }
    return notificationTitleColor;
}
/**
 * 通过一个空的Notification拿到Notification.contentView，通过{@link RemoteViews#apply(Context, ViewGroup)}方法返回通知栏消息根布局实例。
 *
 * @param context 上下文
 * @return 系统主标题颜色
 */
private static int getNotificationColorInternal(Context context) {
    Notification.Builder builder = new Notification.Builder(context);
    builder.setContentTitle(NOTIFICATION_TITLE);
    Notification notification = builder.build();
    try {
        ViewGroup root = (ViewGroup) notification.contentView.apply(context, new FrameLayout(context));
        TextView titleView = (TextView) root.findViewById(android.R.id.title);
        if (null == titleView) {
            iteratorView(root, new Filter() {
                @Override
                public void filter(View view) {
                    if (view instanceof TextView) {
                        TextView textView = (TextView) view;
                        if (NOTIFICATION_TITLE.equals(textView.getText().toString())) {
                            notificationTitleColor = textView.getCurrentTextColor();
                        }
                    }
                }
            });
            return notificationTitleColor;
```

```
        } else {
            return titleView.getCurrentTextColor();
        }
    } catch (Exception e) {
        DebugLog.e(e.getMessage());
        return getNotificationColorCompat(context);
    }
}
/**
 * 使用getNotificationColorInternal()方法，Activity不能继承自AppCompatActivity（实测5.0以下机型可以，5.0及以上机型不行），
 * 大致的原因是默认通知布局文件中的ImageView（largeIcon和smallIcon）被替换成了AppCompatImageView，
 * 而在5.0及以上系统中，AppCompatImageView的setBackgroundResource(int)未被标记为RemotableViewMethod，导致apply时抛异常。
 *
 * @param context 上下文
 * @return 系统主标题颜色
 */
private static int getNotificationColorCompat(Context context) {
    try {
        Notification.Builder builder = new Notification.Builder(context);
        Notification notification = builder.build();
        int layoutId = notification.contentView.getLayoutId();
        ViewGroup root = (ViewGroup) LayoutInflater.from(context).inflate(layoutId, null);
        TextView titleView = (TextView) root.findViewById(android.R.id.title);
        if (null == titleView) {
            return getTitleColorIteratorCompat(root);
        } else {
            return titleView.getCurrentTextColor();
        }
    } catch (Exception e) {
    }
    return INVALID_COLOR;
}
private static void iteratorView(View view, Filter filter) {
    if (view == null || filter == null) {
        return;
    }
    filter.filter(view);
    if (view instanceof ViewGroup) {
        ViewGroup viewGroup = (ViewGroup) view;
        for (int i = 0; i < viewGroup.getChildCount(); i++) {
            View child = viewGroup.getChildAt(i);
            iteratorView(child, filter);
        }
    }
}
private static int getTitleColorIteratorCompat(View view) {
    if (view == null) {
        return INVALID_COLOR;
    }
    List<TextView> textViews = getAllTextViews(view);
    int maxTextSizeIndex = findMaxTextSizeIndex(textViews);
    if (maxTextSizeIndex != Integer.MIN_VALUE) {
        return textViews.get(maxTextSizeIndex).getCurrentTextColor();
    }
    return INVALID_COLOR;
}
private static int findMaxTextSizeIndex(List<TextView> textViews) {
    float max = Integer.MIN_VALUE;
    int maxIndex = Integer.MIN_VALUE;
    int index = 0;
    for (TextView textView : textViews) {
        if (max < textView.getTextSize()) {
            // 找到字号最大的字体，默认把它设置为主标题字号大小
            max = textView.getTextSize();
            maxIndex = index;
        }
```

```
            index++;
        }
        return maxIndex;
    }
    /**
     * 实现遍历View树中的TextView，返回包含TextView的集合。
     *
     * @param root 根节点
     * @return 包含TextView的集合
     */
    private static List<TextView> getAllTextViews(View root) {
        final List<TextView> textViews = new ArrayList<>();
        iteratorView(root, new Filter() {
            @Override
            public void filter(View view) {
                if (view instanceof TextView) {
                    textViews.add((TextView) view);
                }
            }
        });
        return textViews;
    }

    private interface Filter {
        void filter(View view);
    }
```

RemoteViews适配方案

获取系统通知标题颜色，如果能够获取到，那么标题、内容和时间的颜色都设置为标题颜色。 获取不到的情况下，遍历系统通知里的所有文字，取字号最大的那条文字的颜色作为标题、内容和时间的颜色。 以上两个步骤的实现在getNotificationColor()方法里。如果还获取不到，那么标题和内容采用Android原生系统提供的，其中标题是@android:color/primary_text_dark，内容是@android:color/secondary_text_dark。 有一点需要说明的是，以上适配只适合在Android 7.0以下系统。Android 7.0+修改了Notification，采用@android:color/primary_text_dark和@android:color/secondary_text_dark已经获取不到颜色值了，考虑到7.0所采用的通知栏主色调是白色，因此目前暂时的解决方案是遇到7.0的系统采用黑色字体。面对众多厂商的源码修改，目前测试有ZUK的7.0系统为暗色背景，暂时的解决方案是根据机型适配。

参考链接：http://iluhcm.com/2017/03/12/experience-of-adapting-to-android-notifications/

ViewPager和Fragment使用过程中会遇到哪些问题

1. 适配器的选择

使用ViewPager加载多个Fragment时，我一般选择FragmentPagerAdapter。 需要大家注意：
FragmentPagerAdapter：该类内的每一个生成的 Fragment 都将保存在内存之中，因此适用于那些相对静态的页，数量也比较少的场景 但是，如果需要处理有很多页，并且数据动态性较大、占用内存较多的情况，这时候，我们选择怎么是的适配器呢？

应该使用FragmentStatePagerAdapter

FragmentStatePagerAdapter：会把已经创建的Fragment进行保存。会保持Fragment的状态 通过源码分析发现：
FragmentStatePagerAdapter这个类抽象出了一个getItem()方法，用于创建对应的Fragment 而FragmentStatePagerAdapter的.instantiateItem()方法实现中，调用了getlitem()方法。 调用该instantiateItem()方法时，判断一下要生成的 Fragment 是否已经生成过了，如果生成过了，就使用旧的，旧的将被 Fragment.attach()；如果没有，就调用 getItem() 生成一个新的，新的对象将被 FragmentTransation.add()。
FragmentStatePagerAdapter会将所有生成的 Fragment 对象通过 FragmentManager 保存起来备用，以后需要该 Fragment 时，都会从 FragmentManager 读取，而不会再次调用 getItem() 方法。

2. Fragment数据的缓存

Fragment在初始化View对象，把该对象作为一个成员变量进行保存。 再一次初始化Fragment对应的View对象时，判断当前成员变量View对象是否为空。 如果为空，创建新的View对象，否则，不在创建。这样就可以做到对Fragemnt进行数据缓存

这样做的好处：避免了每次加载Fragment都要重新创建View，加载数据了。提高了性能，以及减少了内存的开销。

3. ViewPager预加载

我们知道，系统的ViewPager默认提供预加载机制。但是，根据业务需要，取消掉对应的预加载机制。 可以这样做：替换掉系统原生的Viewpager类。将该类中的一个变量mOffscreenPageLimit 设置为0,不进行预加载

4. Fragment嵌套Fragment

在Fragment中嵌套Fragment时，一定要使用getChildFragmentManager();

否则，会在ViewPager中出现fragment不会加载的情况，即fragment出现空白页的情况。

人事面

- 人事面试宝典一之自我介绍
- 人事面试宝典二之离职
- 人事面试宝典

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

可别小瞧人事面试时的各个介绍环节

1. 请你自我介绍一下你自己？

回答提示：一般人回答这个问题过于平常，只说姓名、年龄、爱好、工作经验，这些在简历上都有，其实，企业最希望知道的是求职者能否胜任工作，包括：最强的技能、最深入研究的知识领域、个性中最积极的部分、做过的最成功的事，主要的成就等，这些都可以和学习无关，也可以和学习有关，但要突出积极的个性和做事的能力，说得合情合理企业才会相信。企业很重视一个人的礼貌，求职者要尊重考官，在回答每个问题之后都说一句“谢谢”。企业喜欢有礼貌的求职者。

2. 你觉得你个性上最大的优点是什么？

回答提示：沉着冷静、条理清楚、立场坚定、顽强向上。 乐于助人和关心他人、适应能力和幽默感、乐观和友爱。

3. 说说你最大的缺点？

回答提示：这个问题企业问的概率很大，通常不希望听到直接回答的缺点是什么等，如果求职者说自己小心眼、爱忌妒人、非常懒、脾气大、工作效率低，企业肯定不会录用你。绝对不要自作聪明地回答“我最大的缺点是过于追求完美”，有的人以为这样回答会显得自己比较出色，但事实上，他已经岌岌可危了。企业喜欢求职者从自己的优点说起，中间加一些小缺点，最后再把问题转回到优点上，突出优点的部分。企业喜欢聪明的求职者。

4. 你对加班的看法？

回答提示：实际上好多公司问这个问题，并不证明一定要加班。 只是想测试你是否愿意为公司奉献。 回答样本：如果是工作需要我会义不容辞加班。我现在单身，没有任何家庭负担，可以全身心的投入工作。但同时，我也会提高工作效率，减少不必要的加班

5. 你对薪资的要求？

回 答提示：如果你对薪酬的要求太低，那显然贬低自己的能力；如果你对薪酬的要求太高，那又会显得你分量过重，公司受用不起。一些雇主通常都事先对求聘的职位定下开支预算，因而他们第一次提出的价钱往往是他们所能给予的最高价钱。他们问你只不过想证实一下这笔钱是否足以引起你对该工作的兴趣 。

回答样本一：“我对工资没有硬性要求。我相信贵公司在处理我的问题上会友善合理。我注重的是找对工作机会，所以只要条件公平，我则不会计较太多

回答样本二：我受过系统的软件编程的训练，不需要进行大量的培训。而且我本人也对编程特别感兴趣。因此，我希望公司能根据我的情况和市场标准的水平，给我合理的薪水。

回答样本三：如果你必须自己说出具体数目，请不要说一个宽泛的范围，那样你将只能得到最低限度的数字。最好给出一个具体的数字，这样表明你已经对当今的人才市场作了调查，知道像自己这样学历的雇员有什么样的价值。

6. 除了本公司外，还应聘了哪些公司？

回答提示：很奇怪，这是相当多公司会问的问题，其用意是要概略知道应徵者的求职志向，所以这并非绝对是负面答案，就算不便说出公司名称，也应回答“销售同种产品的公司”，如果应聘的其他公司是不同业界，容易让人产生无法信任的感觉。

7. 你还有什么问题要问吗？

回答提示：企业的这个问题看上去可有可无，其实很关键，企业不喜欢说“没有问题”的人，因为其很注重员工的个性和创新能力。企业不喜欢求职者问个人福利之类的问题，如果有人这样问：贵公司对新入公司的员工有没有什么培训项目，我可以参加吗？或者说贵公司的晋升机制是什么样的？企业将很欢迎，因为体现出你对学习的热情和对公司的忠诚度以及你的上进心。

> 总结一点：把命卖给公司，最后祝大家五一节快乐。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

为什么要离职?

回答提示:

①回答这个问题时一定要小心，就算在前一个工作受到再大的委屈，对公司有多少的怨言，都千万不要表现出来，尤其要避免对公司本身主管的批评，避免面试官的负面情绪及印象；建议此时最好的回答方式是将问题归咎在自己身上，例如觉得工作没有学习发展的空间，自己想在面试工作的相关产业中多加学习，或是前一份工作与自己的生涯规划不合等等，回答的答案最好是积极正面的。

②我希望能获得一份更好的工作，如果机会来临，我会抓住；我觉得目前的工作，已经达到顶峰，即沒有升迁机会。

您在前一家公司的离职原因是什么?

回答提示:

①最重要的是：应聘者要使找招聘单位相信，应聘者在过往的单位的“离职原因”在此家招聘单位里不存在;

②避免把“离职原因”说得太详细、太具体;

③不能掺杂主观的负面感受，如“太辛苦”、“人际关系复杂”、“管理太混乱”、“公司不重视人才”、“公司排斥我们某某的员工”等;

④但也不能躲闪、回避，如“想换换环境”、“个人原因”等;

⑤不能涉及自己负面的人格特征，如不诚实、懒惰、缺乏责任感、不随和等;

⑥尽量使解释的理由为应聘者个人形象添彩;

⑦相关例子：如“我离职是因为这家公司倒闭；我在公司工作了三年多，有较深的感情；从去年始，由于市场形势突变，公司的局面急转直下；到眼下这一步我觉得很遗憾，但还要面对显示，重新寻找能发挥我能力的舞台。”同一个面试问题并非只有一个答案，而同一个答案并不是在任何面试场合都有效，关键在应聘者掌握了规律后，对面试的具体情况进行把握，有意识地揣摩面试官提出问题的心理背景，然后投其所好。

分析：除非是薪资太低，或者是最初的工作，否则不要用薪资作为理由。“求发展”也被考官听得太多，离职理由要根据每个人的真实离职理由来设计，但是在回答时一定要表现得真诚。实在想不出来的时候，家在外地可以说是因为家中有事，须请假几个月，公司又不可能准假，所以辞职。这个答案一般面试官还能接受。

- 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

面试题-史上最全人事面试宝典

源码分析相关面试题

- Volley源码分析
- 注解框架实现原理
- okhttp3.0源码分析
- onSaveInstanceState源码分析
- 静默安装和源码编译

Activity相关面试题

- 保存Activity的状态

与XMPP相关面试题

- XMPP协议优缺点
- 极光消息推送原理

与性能优化相关面试题

- 内存泄漏和内存溢出区别
- UI优化和线程池实现原理
- 代码优化
- 内存性能分析
- 内存泄漏检测
- App启动优化
- 与IPC机制相关面试题

与登录相关面试题

- oauth认证协议原理
- token产生的意义
- 微信扫一扫实现原理

与开发相关面试题

- 迭代开发的时候如何向前兼容新旧接口
- 手把手教你如何解决as jar包冲突
- context的原理分析
- 解决ViewPager.setCurrentItem中间很多页面切换方案

与人事相关面试题

- 人事面试宝典

1. 请你自我介绍一下你自己？

回答提示：一般人回答这个问题过于平常，只说姓名、年龄、爱好、工作经验，这些在简历上都有，其实，企业最希望知道的是求职者能否胜任工作，包括：最强的技能、最深入研究的知识领域、个性中最积极的部分、做过的最成功的事，主要的成就等，这些都可以和学习无关，也可以和学习有关，但要突出积极的个性和做事的能力，说得合情合理企业才会相信。企业很重视一个人的礼貌，求职者要尊重考官，在回答每个问题之后都说一句“谢谢”。企业喜欢有礼貌的求职者。

2. 你觉得你个性上最大的优点是什么？

回答提示：沉着冷静、条理清楚、立场坚定、顽强向上。 乐于助人和关心他人、适应能力和幽默感、乐观和友爱。

3. 说说你最大的缺点？

回答提示：这个问题企业问的概率很大，通常不希望听到直接回答的缺点是什么等，如果求职者说自己小心眼、爱忌妒人、非常懒、脾气大、工作效率低，企业肯定不会录用你。绝对不要自作聪明地回答“我最大的缺点是过于追求完美”，有的人以为这样回答会显得自己比较出色，但事实上，他已经岌岌可危了。企业喜欢求职者从自己的优点说起，中间加一些小缺点，最后再把问题转回到优点上，突出优点的部分。企业喜欢聪明的求职者。

4. 你对加班的看法？

回答提示：实际上好多公司问这个问题，并不证明一定要加班。 只是想测试你是否愿意为公司奉献。 回答样本：如果是工作需要我会义不容辞加班。我现在单身，没有任何家庭负担，可以全身心的投入工作。但同时，我也会提高工作效率，减少不必要的加班

5. 你对薪资的要求？

回 答提示：如果你对薪酬的要求太低，那显然贬低自己的能力；如果你对薪酬的要求太高，那又会显得你分量过重，公司受用不起。一些雇主通常都事先对求聘的职位定下开支预算，因而他们第一次提出的价钱往往是他们所能给予的最高价钱。他们问你只不过想证实一下这笔钱是否足以引起你对该工作的兴趣。

回答样本一：“我对工资没有硬性要求。我相信贵公司在处理我的问题上会友善合理。我注重的是找对工作机会，所以只要条件公平，我则不会计较太多

回答样本二：我受过系统的软件编程的训练，不需要进行大量的培训。而且我本人也对编程特别感兴趣。因此，我希望公司能根据我的情况和市场标准的水平，给我合理的薪水。

回答样本三：如果你必须自己说出具体数目，请不要说一个宽泛的范围，那样你将只能得到最低限度的数字。最好给出一个具体的数字，这样表明你已经对当今的人才市场作了调查，知道像自己这样学历的雇员有什么样的价值。

6. 在五年的时间内，你的职业规划？

回答提示：这是每一个应聘者都不希望被问到的问题，但是几乎每个人都会被问到。比较多的答案是“管理者”。但是近几年来，许多公司都已经建立了专门的技术途径。这些工作地位往往被称作“顾问”、“参议技师”或“高级软件工程师”等等。当然，说出其他一些你感兴趣的职位也是可以的，比如产品销售部经理，生产部经理等一些与你的专业有相关背景的工作。要知道，考官总是喜欢有进取心的应聘者，此时如果说“不知道”，或许就会使你丧失一个好机会。最普通的回答应该是“我准备在技术领域有所作为”或“我希望能按照公司的管理思路发展”。

7. 你朋友对你的评价？

回答提示： 想从侧面了解一下你的性格及与人相处的问题。 回答样本：“我的朋友都说我是一个可以信赖的人。因为，我一旦答应别人的事情，就一定会做到。如果我做不到，我就不会轻易许诺。 回答样本：”我觉的我是一个比较随和的人，与不同的人都可以友好相处。在我与人相处时，我总是能站在别人的角度考虑问题“

8. 你还有什么问题要问吗？

回答提示：企业的这个问题看上去可有可无，其实很关键，企业不喜欢说“没有问题”的人，因为其很注重员工的个性和创新能力。企业不喜欢求职者问个人福利之类的问题，如果有人这样问：贵公司对新入公司的员工有没有什么培训项目，我可以参加吗？或者说贵公司的晋升机制是什么样的？企业将很欢迎，因为体现出你对学习的热情和对公司的忠诚度以及你的上进心。

9. 如果通过这次面试我们单位录用了你，但工作一段时间却发现你根本不适合这个职位，你怎么办？

回答提示：一段时间发现工作不适合我，有两种情况：

1、如果你确实热爱这个职业，那你就要不断学习，虚心向领导和同事学习业务知识和处事经验，了解这个职业的精神内涵和职业要求，力争减少差距；

2、你觉得这个职业可有可无，那还是趁早换个职业，去发现适合你的，你热爱的职业，那样你的发展前途也会大点，对单位和个人都有好处。

10. 在完成某项工作时，你认为领导要求的方式不是最好的，自己还有更好的方法，你应该怎么做？

回答提示：

- 原则上我会尊重和服从领导的工作安排；同时私底下找机会以请教的口吻，婉转地表达自己的想法，看看领导是否能改变想法；
- 如果领导没有采纳我的建议，我也同样会按领导的要求认真地去完成这项工作；
- 还有一种情况，假如领导要求的方式违背原则，我会坚决提出反对意见；如领导仍固执己见，我会毫不犹豫地再向上级领导反映。

11. 如果你的工作出现失误，给本公司造成经济损失，你认为该怎么办？

回答提示：

- 我本意是为公司努力工作，如果造成经济损失，我认为首要的问题是想方设法去弥补或挽回经济损失。如果我无能力负责，希望单位帮助解决；
- 是责任问题。分清责任，各负其责，如果是我的责任，我甘愿受罚；如果是一个我负责的团队中别人的失误，也不能幸灾乐祸，作为一个团队，需要互相提携共同完成工作，安慰同事并且帮助同事查找原因总结经验。
- 总结经验教训，一个人的一生不可能不犯错误，重要的是能从自己的或者是别人的错误中吸取经验教训，并在今后的工作中避免发生同类的错误。检讨自己的工作方法、分析问题的深度和力度是否不够，以致出现了本可以避免的错误

12. 如果你在这次考试中没有被录用，你怎么打算？

回答提示：现在的社会是一个竞争的社会,从这次面试中也可看出这一点,有竞争就必然有优劣,有成功必定就会有失败.往往成功的背后有许多的困难和挫折,如果这次失败了也仅仅是一次而已,只有经过经验经历的积累才能塑造出一个完全的成功者。我会从以下几个方面来正确看待这次失败.

第一、要敢于面对,面对这次失败不气馁,接受已经失去了这次机会就不会回头这个现实,从心理意志和精神上体现出对这次失败的抵抗力。要有自信,相信自己经历了这次之后经过努力一定能行.能够超越自我.

第二、善于反思,对于这次面试经验要认真总结,思考剖析,能够从自身的角度找差距。正确对待自己,实事求是地评价自己,辩证的看待自己的长短得失,做一个明白人.

第三、走出阴影,要克服这一次失败带给自己的心理压力,时刻牢记自己弱点,防患于未然,加强学习,提高自身素质. 第四、认真工作,回到原单位岗位上后,要实实在在、踏踏实实地工作,三十六行,行行出状元,争取在本岗位上做出一定的成绩. 第五、再接再厉,成为软件工程师或网络工程师一直是我的梦想,以后如果有机会我仍然后再次参加竞争.

13. 如果你做的一项工作受到上级领导的表扬，但你主管领导却说是他做的，你该怎样？

回答提示：我首先不会找那位上级领导说明这件事，我会主动找我的主管领导来沟通，因为沟通是解决人际关系的最好办法，但结果会有两种：1.我的主管领导认识到自己的错误，我想我会视具体情况决定是否原谅他；2.他更加变本加厉的来威胁我，那我会毫不犹豫地找我的上级领导反映此事，因为他这样做会造成负面影响，对今后的工作不利。

14. 谈谈你对跳槽的看法？

回答提示:

- 正常的"跳槽"能促进人才合理流动，应该支持;
- 频繁的跳槽对单位和个人双方都不利，应该反对。

15. 工作中你难以和同事、上司相处，你该怎么办？

回答提示:

- 我会服从领导的指挥，配合同事的工作。
- 我会从自身找原因，仔细分析是不是自己工作做得不好让领导不满意，同事看不惯。还要看看是不是为人处世方面做得不好。如果是这样的话 我会努力改正。
- 如果我找不到原因，我会找机会跟他们沟通，请他们指出我的不足。有问题就及时改正。
- 作为优秀的员工，应该时刻以大局为重，即使在一段时间内，领导和同事对我不理解，我也会做好本职工作，虚心向他们学习，我相信，他们会看见我在努力，总有一天会对我微笑的!

16. 假设你在某单位工作，成绩比较突出，得到领导的肯定。但同时你发现同事们越来越孤立你，你怎么看这个问题？你准备怎么办？

回答提示:

- 成绩比较突出，得到领导的肯定是件好事情，以后更加努力
- 检讨一下自己是不是对工作的热心度超过同事间交往的热心了，加强同事间的交往及共同的兴趣爱好。
- 工作中，切勿伤害别人的自尊心
- 不再领导前拨弄是非
- 乐于助人对面

17. 你最近是否参加了培训课程？谈谈培训课程的内容。是公司资助还是自费参加？

回答提示：可以回答一些线上的自我提升的平台,极客学院,慕课网等.

18. 你对于我们公司了解多少?

回答提示：在去公司面试前上网查一下该公司主营业务。如回答：贵公司有意改变策略，加强与国外大厂的OEM合作，自有品牌的部分则透过海外经销商。

19. 请说出你选择这份工作的动机?

回答提示：这是想知道面试者对这份工作的热忱及理解度，并筛选因一时兴起而来应试的人，如果是无经验者，可以强调"就算职种不同，也希望有机会发挥之前的经验"。

20. 你最擅长的技术方向是什么?

回答提示：说和你要应聘的职位相关的课程，表现一下自己的热诚没有什么坏处。

21. 你能为我们公司带来什么呢?

回答提示： 其实我们为公司所做的，也就是为自己所做的，你在为公司不断付出，取得业绩的同时，也是实现了自己价值，自我成为，所以在回答"你能为公司带来什么"时，不妨站在以上角度

22. 最能概括你自己的三个词是什么?

回答提示： 我经常用的三个词是：适应能力强，有责任心和做事有始终，结合具体例子向主考官解释，

23. 你的业余爱好是什么?

回答提示：找一些富于团体合作精神的，这里有一个真实的故事：有人被否决掉，因为他的爱好是深海潜水。主考官说：因为这是一项单人活动，我不敢肯定他能否适应团体工作。

24. 作为被面试者给我打一下分

回答提示：试着列出四个优点和一个非常非常非常小的缺点，（可以抱怨一下设施，没有明确责任人的缺点是不会有人介意的）。

25. 你怎么理解你应聘的职位?

回答提示：把岗位职责和任务及工作态度阐述一下

26. 喜欢这份工作的哪一点?

回 答提示：相信其实大家心中一定都有答案了吧！每个人的价值观不同，自然评断的标准也会不同，但是，在回答面试官这个问题时可不能太直接就把自己心理的话说出来，尤其是薪资方面的问题，不过一些无伤大雅的回答是不错的考虑，如交通方便，工作性质及内容颇能符合自己的兴趣等等都是不错的答案，不过如果这时自己能仔细思考出这份工作的与众不同之处，相信在面试上会大大加分。

27. 为什么要离职?

回答提示：

回答这个问题时一定要小心，就算在前一个工作受到再大的委屈，对公司有多少的怨言，都千万不要表现出来，尤其要避免对公司本身主管的批评，避免面试官的负面情绪及印象；建议此时最好的回答方式是将问题归咎在自己身上，例如觉得工作没有学习发展的空间，自己想在面试工作的相关产业中多加学习，或是前一份工作与自己的生涯

规划不合等等，回答的答案最好是积极正面的。

我希望能获得一份更好的工作，如果机会来临，我会抓住；我觉得目前的工作，已经达到顶峰，即沒有升迁机会。

28. 说说你对行业、技术发展趋势的看法？

回答提示：企业对这个问题很感兴趣，只有有备而来的求职者能够过关。求职者可以直接在网上查找对你所申请的行业部门的信息，只有深入了解才能产生独特的见解。企业认为最聪明的求职者是对所面试的公司预先了解很多，包括公司各个部门，发展情况，在面试回答问题的时候可以提到所了解的情况，企业欢迎进入企业的人是“知己”，而不是“盲人”。

29. 对工作的期望与目标何在？

回答提示：这是面试者用来评断求职者是否对自己有一定程度的期望、对这份工作是否了解的问题。对于工作有确实学习目标的人通常学习较快，对于新工作自然较容易进入状况，这时建议你，最好针对工作的性质找出一个确实的答案，如业务员的工作可以这样回答：“我的目标是能成为一个超级业务员，将公司的产品广泛的推销出去，达到最好的业绩成效；为了达到这个目标，我一定会努力学习，而我相信以我认真负责的态度，一定可以达到这个目标。”其他类的工作也可以比照这个方式来回答，只要在目标方面稍微修改一下就可以了。

30. 说说你的家庭。

回答提示：企业面试时询问家庭问题不是非要知道求职者家庭的情况，探究隐私，企业不喜欢探究个人隐私，而是要了解家庭背景对求职者的塑造和影响。企业希望听到的重点也在于家庭对求职者的积极影响。企业最喜欢听到的是：我很爱我的家庭！我的家庭一向很和睦，虽然我的父亲和母亲都是普通人，但是从小，我就看到我父亲起早贪黑，每天工作特别勤劳，他的行动无形中培养了我认真负责的态度和勤劳的精神。我母亲为人善良，对人热情，特别乐于助人，所以在单位人缘很好，她的一言一行也一直在教导我做人的道理。企业相信，和睦的家庭关系对一个人的成长有潜移默化的影响。

31. 就你申请的这个职位，你认为你还欠缺什么？

回答提示：企业喜欢问求职者弱点，但精明的求职者一般不直接回答。他们希望看到这样的求职者：继续重复自己的优势，然后说：“对于这个职位和我的能力来说，我相信自己是可以胜任的，只是缺乏经验，这个问题我想我可以进入公司以后以最短的时间来解决，我的学习能力很强，我相信可以很快融入公司的企业文化，进入工作状态。”企业喜欢能够巧妙地躲过难题的求职者。

32. 你欣赏哪种性格的人？

回答提示：诚实、不死板而且容易相处的人、有"实际行动"的人。

33. 你通常如何处理别人的批评？

回答提示：①沈默是金。不必说什么，否则情况更糟，不过我会接受建设性的批评；②我会等大家冷静下来再讨论。

34. 你怎样对待自己的失败？

回答提示：我们大家生来都不是十全十美的，我相信我有第二个机会改正我的错误。

35. 什么会让你有成就感？

回答提示：为贵公司竭力效劳；尽我所能，完成一个项目

36. 眼下你生活中最重要的是什么？

回答提示：对我来说，能在这个领域找到工作是最重要的；望能在贵公司任职对我说最重要。

37. 你为什么愿意到我们公司来工作？

回答提示：对于这个问题，你要格外小心，如果你已经对该单位作了研究，你可以回答一些详细的原因，像“公司本身的高技术开发环境很吸引我。”，“我同公司出生在同样的时代，我希望能够进入一家与我共同成长的公司。”“你们公司一直都稳定发展，在近几年来在市场上很有竞争力。”或者“我认为贵公司能够给我提供一个与众不同的发展道路。”这都显示出你已经做了一些调查，也说明你对自己的未来有了较为具体的远景规划。

38. 你和别人发生过争执吗？你是怎样解决的？

回答提示：这是面试中最险恶的问题。其实是考官布下的一个陷阱。千万不要说任何人的过错。应知成功解决矛盾是一个协作团体中成员所必备的能力。假如你工作在一个服务行业，这个问题简直成了最重要的一个环节。你是否能获得这份工作，将取决于这个问题的回答。考官希望看到你是成熟且乐于奉献的。他们通过这个问题了解你的成熟度和处世能力。在没有外界干涉的情况下，通过妥协的方式来解决才是正确答案。

39. 问题：你做过的哪件事最令自己感到骄傲?

回答提示：这是考官给你的一个机会，让你展示自己把握命运的能力。这会体现你潜在的领导能力以及你被提升的可能性。假如你应聘于一个服务性质的单位，你很可能会被邀请去午餐。记住：你的前途取决于你的知识、你的社交能力和综合表现。

40. 你新到一个部门,一天一个客户来找你解决问题,你努力想让他满意，可是始终达不到群众得满意,他投诉你们部门工作效率低,你这个时候怎么作?

回 答提示：

(1)首先，我会保持冷静。作为一名工作人员，在工作中遇到各种各样的问题是正常的，关键是如何认识它，积极应对，妥善处理。

(2)其次，我会反思一下客户不满意的原因。一是看是否是自己在解决问题上的确有考虑的不周到的地方，二是看是否是客户不太了解相关的服务规定而提出超出规定的要求，三是看是否是客户了解相关的规定，但是提出的要求不合理。

(3)再次，根据原因采取相对的对策。如果是自己确有不周到的地方，按照服务规定作出合理的安排，并向客户作出解释；如果是客户不太了解政策规定而造成的误解，我会向他作出进一步的解释，消除他的误会；如果是客户提出的要求不符合政策规定，我会明确地向他指出。

(4)再次，我会把整个事情的处理情况向领导作出说明，希望得到他的理解和支持。(5)我不会因为客户投诉了我而丧失工作的热情和积极性，而会一如既往地牢记为客户服务的宗旨，争取早日做一名领导信任、公司放心、客户满意的职员。

41. 对这项工作，你有哪些可预见的困难？

回答提示：

不宜直接说出具体的困难，否则可能令对方怀疑应聘者不行；

可以尝试迂回战术，说出应聘者对困难所持有的态度——“工作中出现一些困难是正常的，也是难免的，但是只要有坚忍不拔的毅力、良好的合作精神以及事前周密而充分的准备，任何困难都是可以克服。”

分析：一般问这个问题，面试者的希望就比较大了，因为已经在谈工作细节。但常规思路中的回答，又被面试官“骗”了。当面试官询问这个问题的时候，有两个目的。第一，看看应聘者是不是在行，说出的困难是不是在这个职位中一般都不可避免的问题。第二，是想看一下应聘者解决困难的手法对不对，及公司能否提供这样的资源。而不是想了解应聘者对困难的态度。

42. 如果我录用你，你将怎样开展工作？”

回答提示：

如果应聘者对于应聘的职位缺乏足够的了解，最好不要直接说出自己开展工作的具体办法；

可以尝试采用迂回战术来回答，如“首先听取领导的指示和要求，然后就有关情况进行了解和熟悉，接下来制定一份近期的工作计划并报领导批准，最后根据计划开展工作。”

分析：这个问题的主要目的也是了解应聘者的工作能力和计划性、条理性，而且重点想要知道细节。如果向思路中所讲的迂回战术，面试官会认为回避问题，如果引导了几次仍然是回避的话。此人绝对不会录用了。

43. 你希望与什么样的上级共事？

回答提示：

- 通过应聘者对上级的“希望”可以判断出应聘者对自我要求的意识，这既上一个陷阱，又是一次机会；
- 最好回避对上级具体的希望，多谈对自己的要求；
- 如“做为刚步入社会的新人，我应该多要求自己尽快熟悉环境、适应环境，而不应该对环境提出什么要求，只要能发挥我的专长就可以了

分析：这个问题比较好的回答是，希望我的上级能够在工作中对我多指导，对我工作中的错误能够立即指出。总之，从上级指导这个方面谈，不会有大的纰漏。

44. 在完成某项工作时，你认为领导要求的方式不是最好的，自己还有更好的方法，你应该怎么做？

回答提示：

- 原则上我会尊重和服从领导的工作安排；同时私底下找机会以请教的口吻，婉转地表达自己的想法，看看领导是否能改变想法；
- 如果领导没有采纳我的建议，我也同样会按领导的要求认真地去完成这项工作；
- 还有一种情况，假如领导要求的方式违背原则，我会坚决提出反对意见；如领导仍固执己见，我会毫不犹豫地再向上级领导反映。

45. 与上级意见不一是，你将怎么办？

回答提示：

- 一般可以这样回答“我会给上级以必要的解释和提醒，在这种情况下，我会服从上级的意见。”
- 如果面试你的是总经理，而你所应聘的职位另有一位经理，且这位经理当时不在场，可以这样回答：“对于非原则性问题，我会服从上级的意见，对于涉及公司利益的重大问题，我希望能向更高层领导反映。”

分析：这个问题的标准答案是思路1，如果用2的回答，必死无疑。你没有摸清楚改公司的内部情况，先想打小报告，这样的人没有人敢要。

46. 你工作经验欠缺，如何能胜任这项工作?

常规思路:

- 如果招聘单位对应届毕业生的应聘者提出这个问题，说明招聘公司并不真正在乎“经验”，关键看应聘者怎样回答;
- 对这个问题的回答最好要体现出应聘者的诚恳、机智、果敢及敬业;
- 如“作为应届毕业生，在工作经验方面的确会有所欠缺，因此在读书期间我一直利用各种机会在这个行业里做兼职。我也发现，实际工作远比书本知识丰富、复杂。但我有较强的责任心、适应能力和学习能力，而且比较勤奋，所以在兼职中均能圆满完成各项工作，从中获取的经验也令我受益非浅。请贵公司放心，学校所学及兼职的工作经验使我一定能胜任这个职位。” 点评：这个问题思路中的答案尚可。突出自己的吃苦能力和适应性以及学习能力（不是学习成绩）为好。

47. 您在前一家公司的离职原因是什么?

回答提示:

- 最重要的是：应聘者要使找招聘单位相信，应聘者在过往的单位的“离职原因”在此家招聘单位里不存在
- 避免把“离职原因”说得太详细、太具体
- 不能掺杂主观的负面感受，如“太辛苦”、“人际关系复杂”、“管理太混乱”、“公司不重视人才”、“公司排斥我们某某的员工”等
- 但也不能躲闪、回避，如“想换换环境”、“个人原因”等
- 不能涉及自己负面的人格特征，如不诚实、懒惰、缺乏责任感、不随和等
- 尽量使解释的理由为应聘者个人形象添彩；⑦相关例子：如“我离职是因为这家公司倒闭；我在公司工作了三年多，有较深的感情；从去年始，由于市场形势突变，公司的局面急转直下；到眼下这一步我觉得很遗憾，但还要面对显示，重新寻找能发挥我能力的舞台。”同一个面试问题并非只有一个答案，而同一个答案并不是在任何面试场合都有效，关键在应聘者掌握了规律后，对面试的具体情况进行把握，有意识地揣摩面试官提出问题的心理背景，然后投其所好

分析：除非是薪资太低，或者是最初的工作，否则不要用薪资作为理由。“求发展”也被考官听得太多，离职理由要根据每个人的真实离职理由来设计，但是在回答时一定要表现得真诚。实在想不出来的时候，家在外地可以说是因为家中有事，须请假几个月，公司又不可能准假，所以辞职。这个答案一般面试官还能接受。

48. 你工作经验欠缺，如何能胜任这项工作?

回答提示:

- 如果招聘单位对应届毕业生的应聘者提出这个问题，说明招聘公司并不真正在乎“经验”，关键看应聘者怎样回答;
- 对这个问题的回答最好要体现出应聘者的诚恳、机智、果敢及敬业;
- 如“作为应届毕业生，在工作经验方面的确会有所欠缺，因此在读书期间我一直利用各种机会在这个行业里做兼职。我也发现，实际工作远比书本知识丰富、复杂。但我有较强的责任心、适应能力和学习能力，而且比较勤奋，所以在兼职中均能圆满完成各项工作，从中获取的经验也令我受益非浅。请贵公司放心，学校所学及兼职的工作经验使我一定能胜任这个职位。”

分析：这个问题思路中的答案尚可。突出自己的吃苦能力和适应性以及学习能力（不是学习成绩）为好。

49. 为了做好你工作份外之事，你该怎样获得他人的支持和帮助?

回答提示：每个公司都在不断变化发展的过程中；你当然希望你的员工也是这样。你希望得到那些希望并欢迎变化的人，因为这些人明白，为了公司的发展，变化是公司日常生活中重要组成部分。这样的员工往往很容易适应公司的变化，并会对变化做出积极的响应。此外，他们遇到矛盾和问题时，也能泰然处之。下面的问题能够考核应聘者这方面的能力。 据说有人能从容避免正面冲突。请讲一下你在这方面的经验和技巧。

有些时候，我们得和我们不喜欢的人在一起共事。说说你曾经克服了性格方面的冲突而取得预期工作效果的经历。

50. 如果你在这次面试中没有被录用，你怎么打算?

回答提示：现在的社会是一个竞争的社会,从这次面试中也可看出这一点,有竞争就必然有优劣,有成功必定就会有失败.往往成功的背后有许多的困难和挫折,如果这次失败了也仅仅是一次而已,只有经过经验经历的积累才能塑造出一个完全的成功者。我会从以下几个方面来正确看待这次失败.

第一、要敢于面对,面对这次失败不气馁,接受已经失去了这次机会就不会回头这个现实,从心理意志和精神上体现出对这次失败的抵抗力。要有自信,相信自己经历了这次之后经过努力一定能行.能够超越自我.

第二、善于反思,对于这次面试经验要认真总结,思考剖析,能够从自身的角度找差距。正确对待自己,实事求是地评价自己,辩证的看待自己的长短得失,做一个明白人.

第三、走出阴影,要克服这一次失败带给自己的心理压力,时刻牢记自己弱点,防患于未然,加强学习,提高自身素质. 第四、认真工作,回到原单位岗位上后,要实实在在、踏踏实实地工作,三十六行,行行出状元,争取在本岗位上做出一定的成绩. 第五、再接再厉,成为国家公务员一直是我的梦想,以后如果有机会我仍然后再次参加竞争.

51. 假如你晚上要去送一个出国的同学去机场，可单位临时有事非你办不可，你怎么办?

回答提示：我觉得工作是第一位的，但朋友间的情谊也是不能偏废的。这个问题我觉得要按照当时具体的情况来决定。

- 如果我的朋友晚上9点中的飞机，而我的 加班八点就能够完成的话，那就最理想了，干完工作去机场，皆大欢喜。
- 如果说工作不是很紧急，加班仅仅是为了明天上班的时候能把报告交到办公室，那完全可以跟领导打声招呼，先去机场然后回来加班，晚点睡就是了。
- 如果工作很紧急，两者不可能兼顾的情况下，我觉得可以由两种选择。
    - 如果不是全单位都加班的话，是不是可以要其他同事来代替以下工作，自己去机场，哪怕就是代替你离开的那一会儿。
    - 如果连这一点都做不到的话，那只好忠义不能两全了，打电话给朋友解释一下，小心他会理解，毕竟工作做完了就完了，朋友还是可以再见面的。

52. 如果通过这次面试我们单位录用了你，但工作一段时间却发现你根本不适合这个职位，你怎么办?

回答提示：一段时间发现工作不适合我，有两种情况：

1、如果你确实热爱这个职业，那你就要不断学习，虚心向领导和同事学习业务知识和处事经验，了解这个职业的精神内涵和职业要求，力争减少差距；

2、你觉得这个职业可有可无，那还是趁早换个职业，去发现适合你的，你热爱的职业，那样你的发展前途也会大点，对单位和个人都有好处。

53. 你做过的哪件事最令自己感到骄傲?

回答提示：这是考官给你的一个机会，让你展示自己把握命运的能力。这会体现你潜在的领导能力以及你被提升的可能性。假如你应聘于一个服务性质的单位，你很可能会被邀请去午餐。记住：你的前途取决于你的知识、你的社交能力和综合表现。

54. 谈谈你过去做过的成功案例

回答提示：举一个你最有把握的例子，把来龙去脉说清楚，而不要说了很多却没有重点。切忌夸大其词，把别人的功劳到说成自己的，很多主管为了确保要用的人是最适合的，会打电话向你的前一个主管征询对你的看法及意见，所以如果说谎，是很容易穿梆的。

55. 谈谈你过去的工作经验中，最令你挫折的事情

回答提示：曾经接触过一个客户，原本就有耳闻他们以挑剔出名，所以事前的准备功夫做得十分充分，也投入了相当多的时间与精力，最后客户虽然并没有照单全收，但是接受的程度已经出乎我们意料之外了。原以为从此可以合作愉快，却得知客户最后因为预算关系选择了另一家代理商，之前的努力因而付诸流水。尽管如此，我还是从这次的经验学到很多，如对该产业的了解，整个team的默契也更好了。

分析：借此了解你对挫折的容忍度及调解方式。

56. 如何安排自己的时间？会不会排斥加班？

回答提示：基本上，如果上班工作有效率，工作量合理的话，应该不太需要加班。可是我也知道有时候很难避免加班，加上现在工作都采用责任制，所以我会调配自己的时间，全力配合。

分析：虽然不会有人心甘情愿的加班，但依旧要表现出高配合度的诚意。

57. 为什么我们要在众多的面试者中选择你？

回答提示：根据我对贵公司的了解，以及我在这份工作上所累积的专业、经验及人脉，相信正是贵公司所找寻的人才。而我在工作态度、EQ上，也有圆融、成熟的一面，和主管、同事都能合作愉快。

分析：别过度吹嘘自己的能力，或信口开河地乱开支票，例如一定会为该公司带来多少钱的业务等，这样很容易给人一种爱说大话、不切实际的感觉。

58. 对这个职务的期许？

回答提示：希望能借此发挥我的所学及专长，同时也吸收贵公司在这方面的经验，就公司、我个人而言，缔造“双赢”的局面。

分析：回答前不妨先询问该公司对这项职务的责任认定及归属，因为每一家公司的状况不尽相同。以免说了一堆理想抱负却发现牛头不对马嘴。

59. 为什么选择这个职务？

回答提示：：这一直是我的兴趣和专长，经过这几年的磨练，也累积了一定的经验及人脉，相信我一定能胜任这个职务的。 分析：适时举出过去的“丰功伟业”，表现出你对这份职务的熟稔度，但避免过于夸张的形容或流于炫耀。

60. 为什么选择我们这家公司？

回答提示：曾经在报章杂志看过关于贵公司的报道，与自己所追求的理念有志一同。而贵公司在业界的成绩也是有目共睹的，而且对员工的教育训练、升迁等也都很有制度。

分析：去面试前先做功课，了解一下该公司的背景，让对方觉得你真的很有心想得到这份工作，而不只是探探路。

61. 你认为你在学校属于好学生吗？

回答提示：企业的招聘者很精明，问这个问题可以试探出很多问题：如果求职者学习成绩好，就会说：“是的，我的成绩很好，所有的成绩都很优异。当然，判断一个学生是不是好学生有很多标准，在学校期间我认为成绩是重要的，其他方面包括思想道德、实践经验、团队精神、沟通能力也都是很重要的，我在这些方面也做得很好，应该说我是一个全面发展的学生。”如果求职者成绩不尽理想，便会说：“我认为是不是一个好学生的标准是多元化的，我的学习成绩还可以，在其他方面我的表现也很突出，比如我去很多地方实习过，我很喜欢在快节奏和压力下工作，我在学生会组织过 ××活动，锻炼了我的团队合作精神和组织能力。” 有经验的招聘者一听就会明白，企业喜欢诚实的求职者。

62. 请谈谈如何适应办公室工作的新环境？

回答提示

- 办公室里每个人有各自的岗位与职责，不得擅离岗位。
- 根据领导指示和工作安排，制定工作计划，提前预备，并按计划完成。
- 多请示并及时汇报，遇到不明白的要虚心请教。
- 抓间隙时间，多学习，努力提高自己的政治素质和业务水平。

63. 在工作中学习到了些什么？

回答提示：这是针对转职者提出的问题，建议此时可以配合面试工作的特点作为主要依据来回答，如业务工作需要与人沟通，便可举出之前工作与人沟通的例子，经历了哪些困难，学习到哪些经验，把握这些要点做陈述，就可以轻易过关了

64. 有想过创业吗？

回答提示：这个问题可以显示你的冲劲，但如果你的回答是“有”的话，千万小心，下一个问题可能就是“那么为什么你不这样做呢？”

65. 最能概括你自己的三个词是什么？

回答提示：我经常用的三个词是：适应能力强，有责任心和做事有始终，结合具体例子向主考官解释，使他们觉得你具有发展潜力

66. 你认为你在学校属于好学生吗？

回答提示：企业的招聘者很精明，问这个问题可以试探出很多问题：如果求职者学习成绩好，就会说：“是的，我的成绩很好，所有的成绩都很优异。当然，判断一个学生是不是好学生有很多标准，在学校期间我认为成绩是重要的，其他方面包括思想道德、实践经验、团队精神、沟通能力也都是很重要的，我在这些方面也做得很好，应该说我是一个全面发展的学生。”如果求职者成绩不尽理想，便会说：“我认为是不是一个好学生的标准是多元化的，我的学习成绩还可以，在其他方面我的表现也很突出，比如我去很多地方实习过，我很喜欢在快节奏和压力下工作，我在学生会组织过 ××活动，锻炼了我的团队合作精神和组织能力。” 有经验的招聘者一听就会明白，企业喜欢诚实的求职者。

67. 除了本公司外，还应聘了哪些公司？

回答提示：很奇怪，这是相当多公司会问的问题，其用意是要概略知道应徵者的求职志向，所以这并非绝对是负面答案，就算不便说出公司名称，也应回答“销售同种产品的公司”，如果应聘的其他公司是不同业界，容易让人产生无法信任的感觉。

68. 何时可以到职？

回答提示：大多数企业会关心就职时间，最好是回答\'如果被录用的话，到职日可按公司规定上班”，但如果还未辞去上一个工作、上班时间又太近，似乎有些强人所难，因为交接至少要一个月的时间，应进一步说明原因，录取公司应该会通融的

69. 你并非毕业于名牌院校？

回答提示：是否毕业于名牌院校不重要，重要的是有能力完成您交给我的工作,我有什么什么项目经验,如何帮助项目经理解决了问题,不拉不拉。

70. 你怎样看待学历和能力？

回答提示：学历我想只要是大学专科的学历，就表明觉得我具备了根本的学习能力。剩下的，你是学士也好，还是博士也好，对于这一点的讨论，不是看你学了多少知识，而是看你在这个领域上发挥了什么，也就是所说的能力问题。一个人工作能力的高低直接决定其职场命运，而学历的高低只是进入一个企业的敲门砖，如果贵公司把学历卡在博士上，我就无法进入贵公司，当然这不一定只是我个人的损失，如果一个专科生都能完成的工作，您又何必非要招聘一位博士生呢？

- 欢迎关注微信公众号,长期推荐技术文章和技术视频
- 微信公众号名称：Android干货程序员

> 原文链接：马伟奇，http://www.jianshu.com/p/d61b553ff8c9

对于初学者来说，对Token和Session的使用难免会限于困境，开发过程中知道有这个东西，但却不知道为什么要用他？更不知道其原理，今天我就带大家一起分析分析这东西。

我们先解释一下他的含义：

1、Token的引入：Token是在客户端频繁向服务端请求数据，服务端频繁的去数据库查询用户名和密码并进行对比，判断用户名和密码正确与否，并作出相应提示，在这样的背景下，Token便应运而生。

2、Token的定义：Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

3、使用Token的目的：Token的目的是为了减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。

了解了Token的意义后，我们就更明确的知道为什么要用他了。

如何使用Token?

这是本文的重点，在这里我就介绍常用的两种方式。

1、用设备号/设备mac地址作为Token（推荐）

客户端：客户端在登录的时候获取设备的设备号/mac地址，并将其作为参数传递到服务端。

服务端：服务端接收到该参数后，便用一个变量来接收同时将其作为Token保存在数据库，并将该Token设置到session中，客户端每次请求的时候都要统一拦截，并将客户端传递的token和服务器端session中的token进行对比，如果相同则放行，不同则拒绝。

分析：此刻客户端和服务器端就统一了一个唯一的标识Token，而且保证了每一个设备拥有了一个唯一的会话。该方法的缺点是客户端需要带设备号/mac地址作为参数传递，而且服务器端还需要保存；优点是客户端不需重新登录，只要登录一次以后一直可以使用，至于超时的问题是有服务器这边来处理，如何处理？若服务器的Token超时后，服务器只需将客户端传递的Token向数据库中查询，同时并赋值给变量Token，如此，Token的超时又重新计时。

2、用session值作为Token

客户端：客户端只需携带用户名和密码登陆即可。

客户端：客户端接收到用户名和密码后并判断，如果正确了就将本地获取sessionID作为Token返回给客户端，客户端以后只需带上请求数据即可。

分析：这种方式使用的好处是方便，不用存储数据，但是缺点就是当session过期后，客户端必须重新登录才能进行访问数据。

使用过程中出现的问题以及解决方案?

刚才我们轻松介绍了Token的两种使用方式，但是在使用过程中我们还出现各种问题，Token第一种方法中我们隐藏了一个在网络不好或者并发请求时会导致多次重复提交数据的问题。

该问题的解决方案：将session和Token套用，如此便可解决，如何套用呢？请看这段解释：

session是一个在单个操作人员整个操作过程中，与服务器端保持通信的惟一识别信息。在同一操作人员的多次请求当中，session始终保证是同一个对象，而不是多个对象，因为可以对其加锁。当同一操作人员多个请求进入时，可以通过session限制只能单向通行。

本文正是通过使用session以及在session中加入token，来验证同一个操作人员是否进行了并发重复的请求，在后一个请求到来时，使用session中的token验证请求中的token是否一致，当不一致时，被认为是重复提交，将不准许通过。

这就是解决重复提交的方案。

总结：以上是个人对开发中使用Token和session的一点总结，如有叙述不当之处请指正，我将及时改正并感谢，我知道还有更多更好的使用方式，我在这里只是抛砖引玉，希望大家将您的使用方式提出来，我们一起讨论，学习，一起进步，同时也为像我一样对这方面理解薄弱的朋友提供点帮助，谢谢。

极光推送技术原理：移动无线网络长连接

移动互联网应用现状

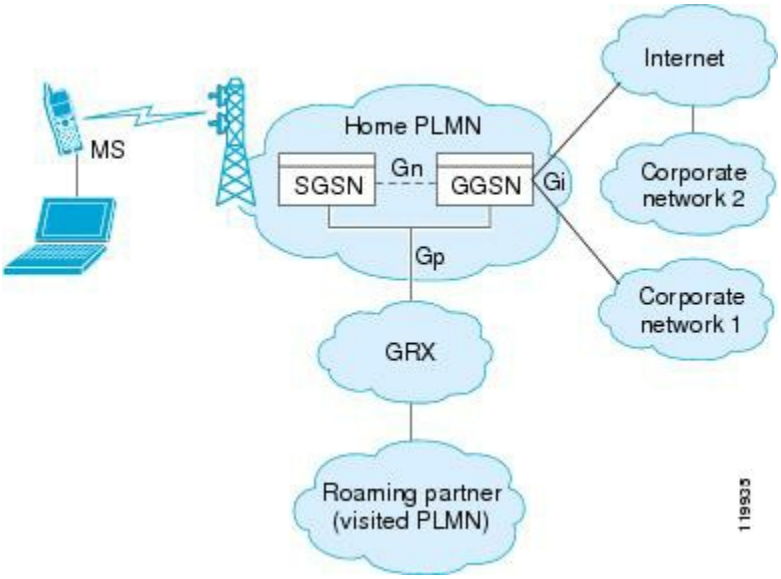
因为手机平台本身、电量、网络流量的限制，移动互联网应用在设计上跟传统 PC 上的应用很大不一样，需要根据手机本身的特点，尽量的节省电量和流量，同时又要尽可能的保证数据能及时到达客户端。

为了解决数据同步的问题，在手机平台上，常用的方法有2种。一种是定时去服务器上查询数据，也叫Polling，还有一种手机跟服务器之间维护一个 TCP 长连接，当服务器有数据时，实时推送到客户端，也就是我们说的 Push。

从耗费的电量、流量和数据送达的及时性来说，Push 都会有明显的优势，但 Push 的实现和维护成本相对较高。在移动无线网络下维护长连接，相对也有一些技术上的难度。本文试图给大家介绍一下我们极光推送在 Android 平台上是如何维护长连接。

移动无线网络的特点

因为 IP v4 的 IP 量有限，运营商分配给手机终端的 IP 是运营商内网的 IP，手机要连接 Internet，就需要通过运营商的网关做一个网络地址转换（Network Address Translation，NAT）。简单的说运营商的网关需要维护一个外网 IP、端口到内网 IP、端口的对应关系，以确保内网的手机可以跟 Internet 的服务器通讯。



图片源自 cisco.com.

NAT 功能由图中的 GGSN 模块实现。

大部分移动无线网络运营商都在链路一段时间没有数据通讯时，会淘汰 NAT 表中的对应项，造成链路中断。

Android 平台上长连接的实现

为了不让 NAT 表失效，我们需要定时的发心跳，以刷新 NAT 表项，避免被淘汰。

Android 上定时运行任务常用的方法有2种，一种方法用 Timer，另一种是AlarmManager。

Timer

Android 的 Timer 类可以用来计划需要循环执行的任务，Timer 的问题是它需要用 WakeLock 让 CPU 保持唤醒状态，这样会大量消耗手机电量，大大减短手机待机时间。这种方式不能满足我们的需求。

AlarmManager

AlarmManager 是 Android 系统封装的用于管理 RTC 的模块，RTC (Real Time Clock) 是一个独立的硬件时钟，可以在 CPU 休眠时正常运行，在预设的时间到达时，通过中断唤醒 CPU。

这意味着，如果我们用 AlarmManager 来定时执行任务，CPU 可以正常的休眠，只有在需要运行任务时醒来一段很短的时间。极光推送的 Android SDK 就是基于这种技术实现的。

服务器设计

当有大量的手机终端需要与服务器维持长连接时，对服务器的设计会是一个很大的挑战。

假设一台服务器维护10万个长连接，当有1000万用户量时，需要有多达100台的服务器来维护这些用户的长连接，这里还不算用于做备份的服务器，这将会是一个巨大的成本问题。那就需要我们尽可能提高单台服务器接入用户的量，也就是业界已经讨论很久了的 C10K 问题。

C2000K

针对这个问题，我们专门成立了一个项目，命名为C2000K，顾名思义，我们的目标是单机维持200万个长连接。最终我们采用了多消息循环、异步非阻塞的模型，在一台双核、24G内存的服务器上，实现峰值维持超过300万个长连接。

后记

稳定维护长连接是推送平台的一个基础，极光推送团队将会在这方面长期投入，以保证用户能有效的节省电量、流量，同时数据能实时送达。

即时通信技术-Websocket

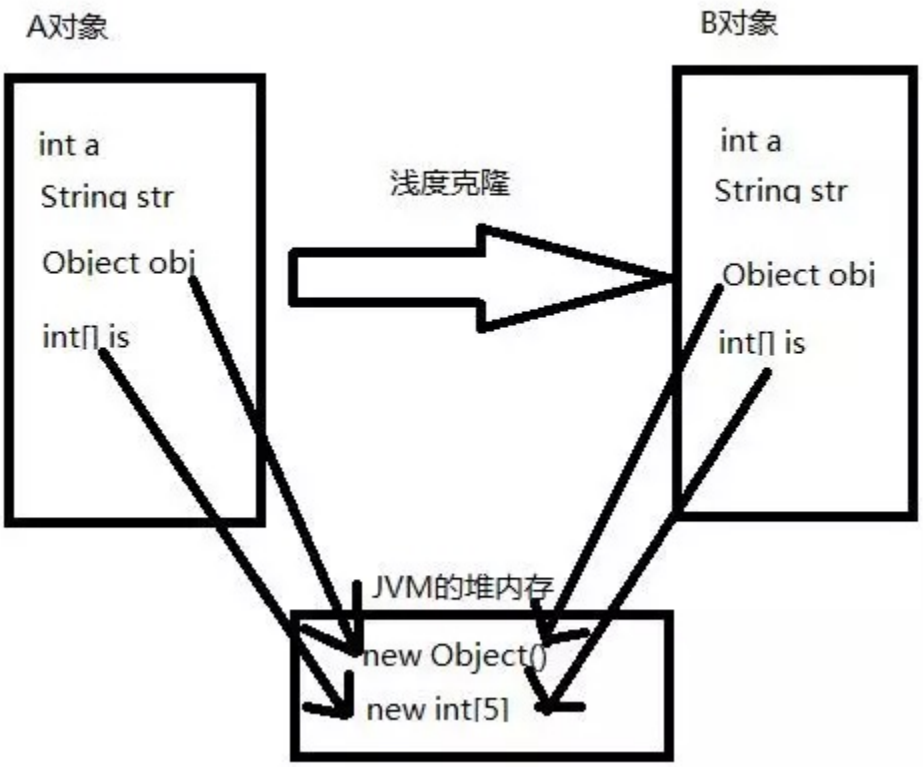
以前不管使用HTTP轮询或使用TCP长连接等方式制作在线聊天系统，都有天然缺陷，随着Html5的兴起，其中有一个新的协议WebSocket protocol，可实现浏览器与服务器全双工通信(full-duplex)，它可以做到：浏览器和服务器只需要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。这个新的协议的特点正好适合这种在线即时通信。

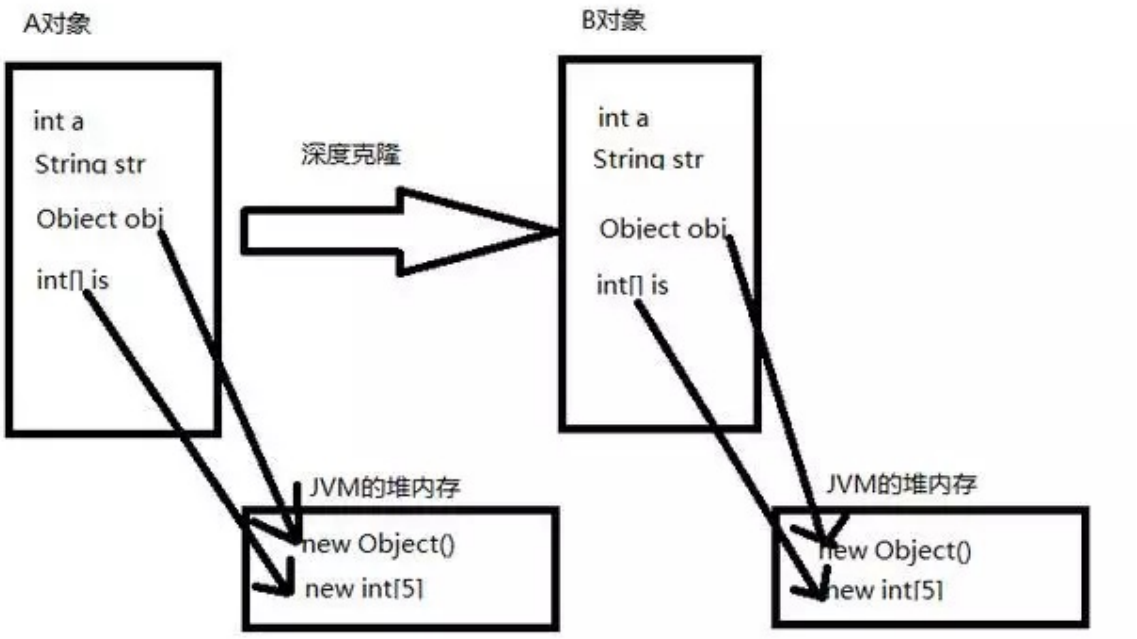
Java面试题

“纵使面试无数，难敌考官吃素“，昨天去乐视面试，遇到一个从来没有遇到过的考题！

请分别实现深度和浅读的对象克隆？

原理：深度克隆和浅度克隆，Object中的克隆方法是浅度克隆。JDK规定了克隆需要满足的一些条件，简要总结一下就是：对某个对象进行克隆，对象的的成员变量如果包括引用类型或者数组，那么克隆的时候其实是不会把这些对象也带着复制到克隆出来的对象里面的，只是复制一个引用，这个引用指向被克隆对象的成员对象，但是基本数据类型是会跟着被带到克隆对象里面去的。而深度可能就是把对象的所有属性都统统复制一份新的到目标对象里面去。简单画个对比图：





实现方式

- 实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆；
- 实现Cloneable接口并重写Object类中的clone()方法，即可实现浅度克隆。

代码

Car.java

```
public class Car implements Serializable {

    private static final long serialVersionUID = 1L;

    private String brand;//品牌
    private int maxSpeed;//最高时速

    public Car(String brand, int maxSpeed) {
        this.brand = brand;
        this.maxSpeed = maxSpeed;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public int getMaxSpeed() {
        return maxSpeed;
    }

    public void setMaxSpeed(int maxSpeed) {
        this.maxSpeed = maxSpeed;
    }
```

```
    @Override
    public String toString() {
        return "Car{" +
                "brand='" + brand + '\'' +
                ", maxSpeed=" + maxSpeed +
                '}';
    }
}
```

Person.java

```
public class Person implements Serializable {
    private static final long serialVersionUID = 1L;

    private String name;//姓名
    private int age;//年龄
    private Car car;//座驾

    public Person(String name, int age, Car car) {
        this.name = name;
        this.age = age;
        this.car = car;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
        this.car = car;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", car=" + car +
                '}';
    }
}
```

Person2.java

```
public class Person2 implements Serializable {
    private static final long serialVersionUID = 1L;
```

```
    private String name;//姓名
    private int age;//年龄
    private Car car;//座驾

    public Person2(String name, int age, Car car) {
        this.name = name;
        this.age = age;
        this.car = car;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
        this.car = car;
    }

    @Override
    protected Object clone(){
        Person2 s= null;
        try {
            s = (Person2) Person2.super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return s;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", car=" + car +
                '}';
    }
}
```

CloneUtil.java

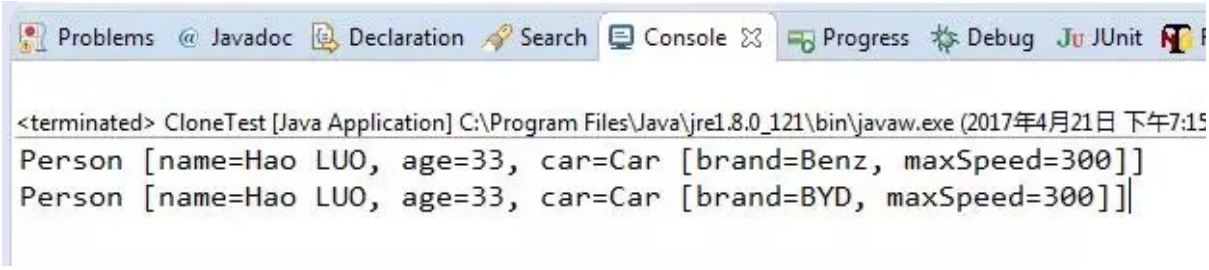
```
public class CloneUtil {

    private CloneUtil() {
        throw new AssertionError();
    }

    public static <T extends Serializable> T clone(T object) throws IOException,
```

```
            ClassNotFoundException {
        // 说明：调用ByteArrayOutputStream或ByteArrayInputStream对象的close方法没有任何意义
        // 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源，这一点不同于对外资源(如文件流)的释放
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(object);

        ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());
        ObjectInputStream ois = new ObjectInputStream(bais);
        return (T) ois.readObject();
    }
}
```

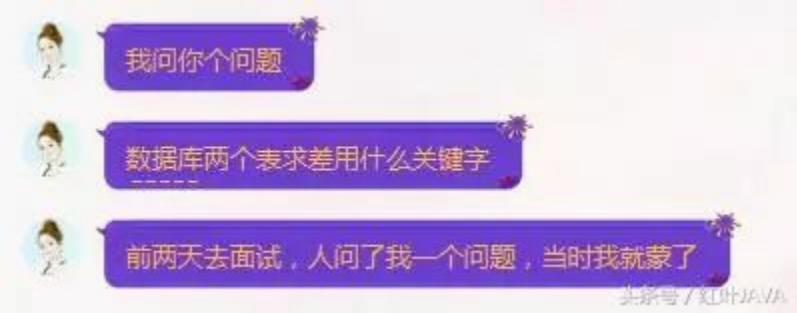
CloneTest

```
public class CloneTest {
    public static void main(String[] args){
        try {
            // 深拷贝
            Person p1 = new Person("xiaoming",18,new Car("Benz",300));
            Person p2 = CloneUtil.clone(p1);
            // 修改克隆的Person对象p2关联的汽车对象的品牌属性
            // 原来的Person对象p1关联的汽车不会受到任何影响，还是Benz
            // 因为在克隆Person对象时其关联的汽车对象也被克隆了

            p2.setAge(25);
            p2.getCar().setBrand("BYD");

            System.out.println(p1);

            Person2 p3 = new Person2("xiaoming",18,new Car("Benz",300));
            Person2 p4 = (Person2) p3.clone();
            p4.setAge(25);
            p4.getCar().setBrand("BYD");

            System.out.println(p3);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

结果



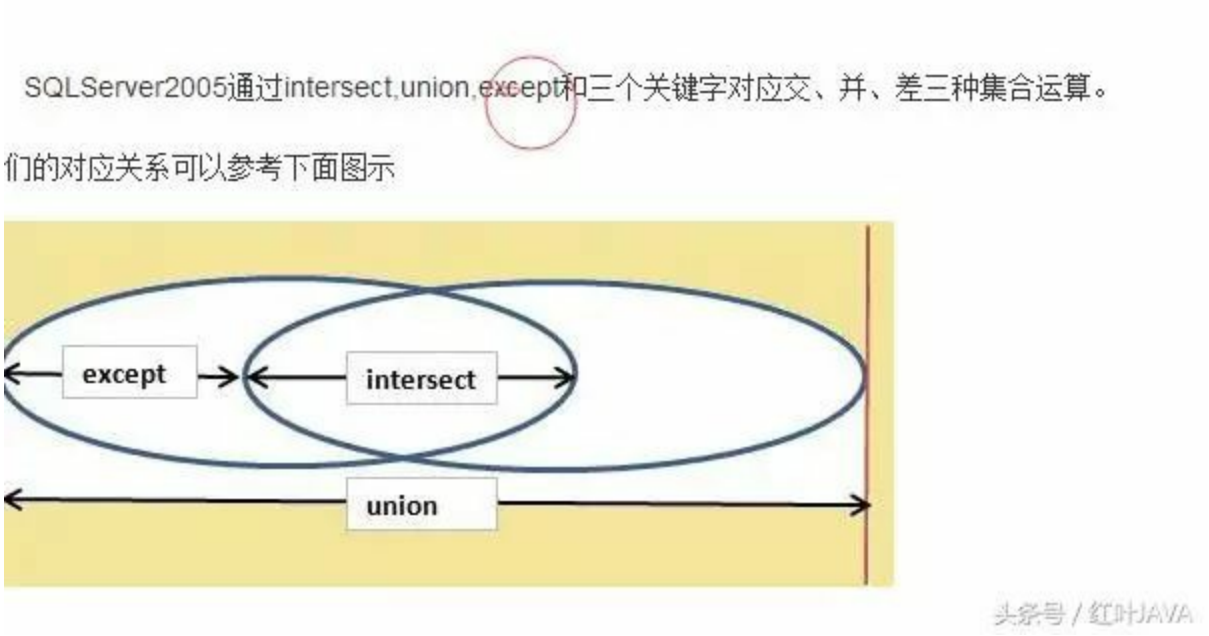
今日，有同学跟我说他在最近在面试的时候，面试官问了他一个很简单的问题，结果他一脸懵逼，他可是有过一年开发经验的，怎么可能会在面试的时候马失前蹄了呢？大家来看下他的问题你们会不会。



是很简单的问题吧

我一看，这问题糟了，我也不知道，是很尴尬，我就给了我们公司一位大牛，结果大牛神秘兮兮的给了一张图

这一下子我就看懂了，但我就是不说，我就也给他回了这张图



这是后来他回我的

但是，不得不说，他还是可以的，一下子他也看懂了，原来这就是个小问题，但是就是这个小细节，他却错失了一次很好的机会。

我也帮大家整理了一些网上面网友遇到过的古怪面试题。可能不全，但也会帮到大家的吧

1、 使用两种方式写出SingleTon.

2、 简单绘制出Sun Hotspot VM的内存结构。并简单说明各自的作用。

3、 实现一个用于生产者-消费者模式的队列。假设队列的固定长度为10，FULL的时候只能消费不能生产，EMPTY的时候只能生产不能消费。

4、 调用A线程的interrupt方法，其在什么情况下会抛出InterruptException（不考虑网络IO的情况）

5、 Sevlet容器如何使用cookie跟踪回话？response.encodeURL(URL)什么情况下生效？

6、 DOM与SAX方式解析XML文件的原理？各自适用的场景是什么？

7、 将一个byte值转化为int值为什么要与0Xff进行&运算？

答案呢，在下面

1.至少有3种写法

2.新生代(eden+2survisor) 老年代 持久代

3.可考虑blockingqueue

4.该线程在wait或者sleep时

5.cookie里边有session id的内容。

6.dom是一口吃，生成节点树~另一个基于事件处理

7.8->32位

试题都很简单，但是越微小越容易出错，编程更是这样，虽然现在代码都会报错，但是我们也应该养成注意细节的好习惯。

1. JRE与JDK的区别?

JRE:java运行时的环境，JDK:包含JRE并且可以查看源码

2. Java中的数据类型都有哪些?

分别是8种基本数据类型:byte、short、int、long、float、double、boolean、char

除8种以外统称为引用类型,例如:String类、Object类、数组及自己创建的类等

3. 等号“==”与equals的区别?

"=="比较的是栈上的值(基本数据类型比较值的方式)

equlas比较的是堆上的值(引用类型比较值的方式)

4. i++与++i的区别?

i++为代码执行后自加1，++i为代码执行前自加1

5. break和continue和return的区别?

break:跳出整个循环，continue:跳出本次循环,进入下一次循环，return:跳出方法,结束这个方法,并返回一个数据

6. int类型和String类型之间的互相转换?

int -->String常用的方法:

- 字符串拼接
- String b = String.valueof(int a)方法

String -->int常用的方法:

- int i = Integer.valueof(String a)方法
- int i = Integer.parseInt(String a)方法

7. &、|与&&、||的区别?

&、| :为位运算符可以进行位运算,符号两边都需要判断才会结束判断,效率低

&&、||:为逻辑判断符用来进行逻辑判断,从左到右判断,一面为否就结束判断,效率高

8. swich()语句中小括号能使用String类型数据么?

- 只用JDK1.7版本以后才可以使用String类型数据
- long类型数据任何版本都不可以使用

9. 循环都有哪些?

- for循环
- for each循环

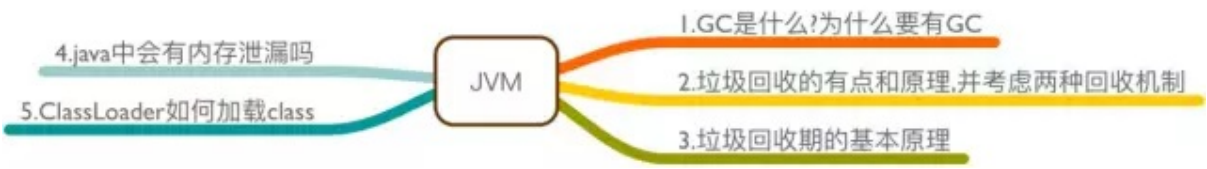
- while循环
- do while 循环 (先执行一次)
- 递归(方法自己调用自己)

10. 类的结构是什么?

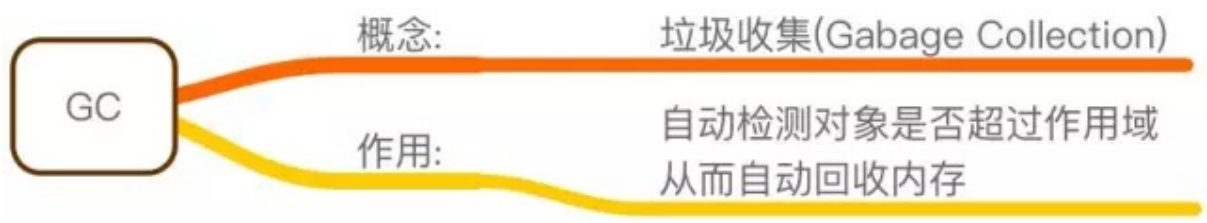
- 成员变量
- 构造器(构造方法)(不可以被static修饰)
- 普通方法
- 代码块
- 内部类

11. 图解java面试题 - JVM

内容大纲

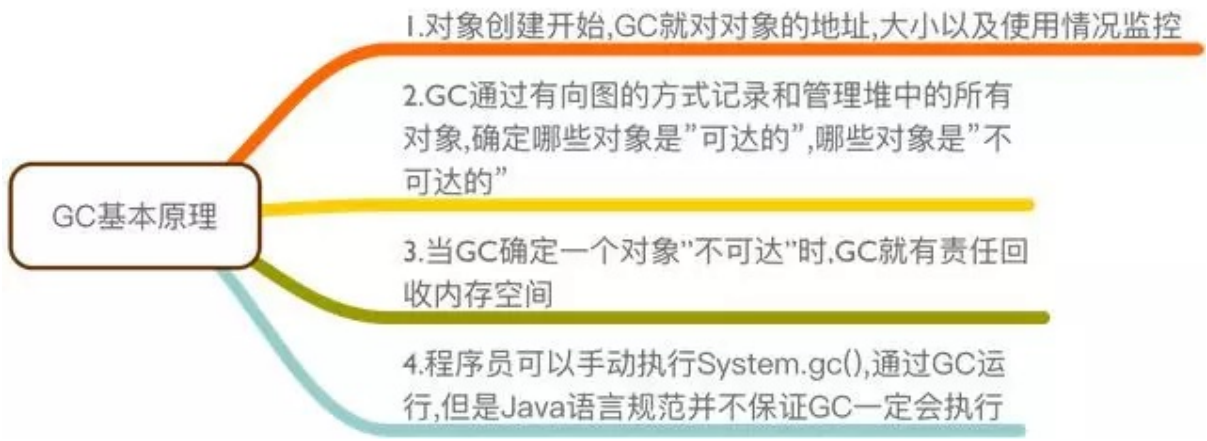


GC是什么?为什么要有GC?

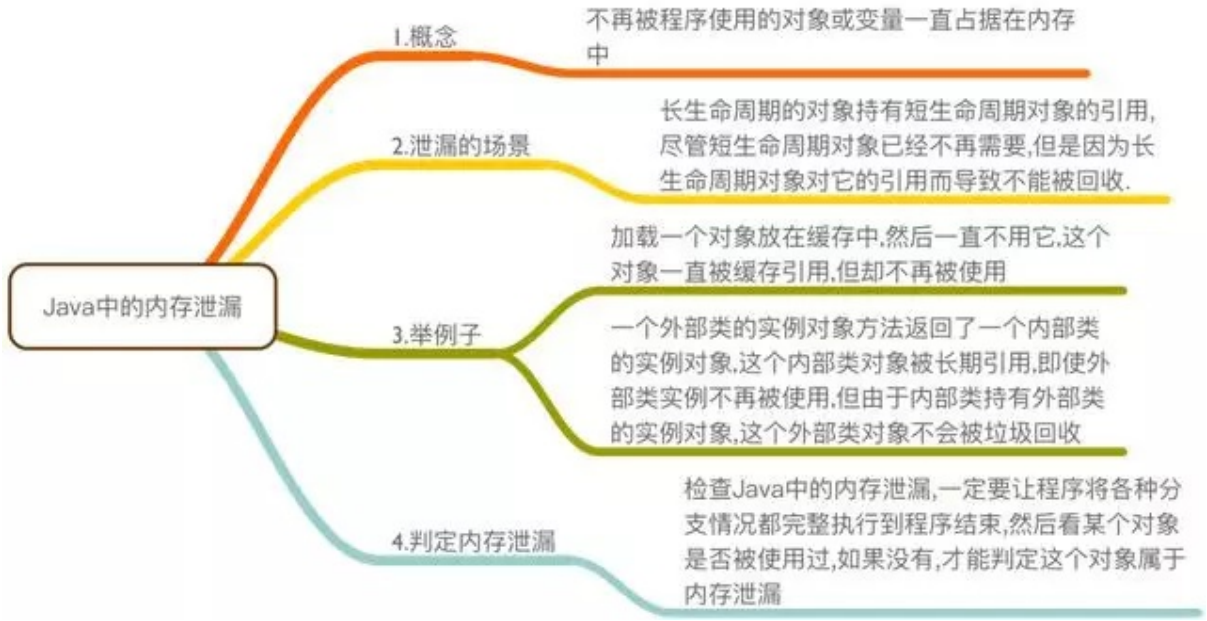


垃圾回收的优点和原理,并考虑两种回收机制

垃圾回收器的基本原理是什么

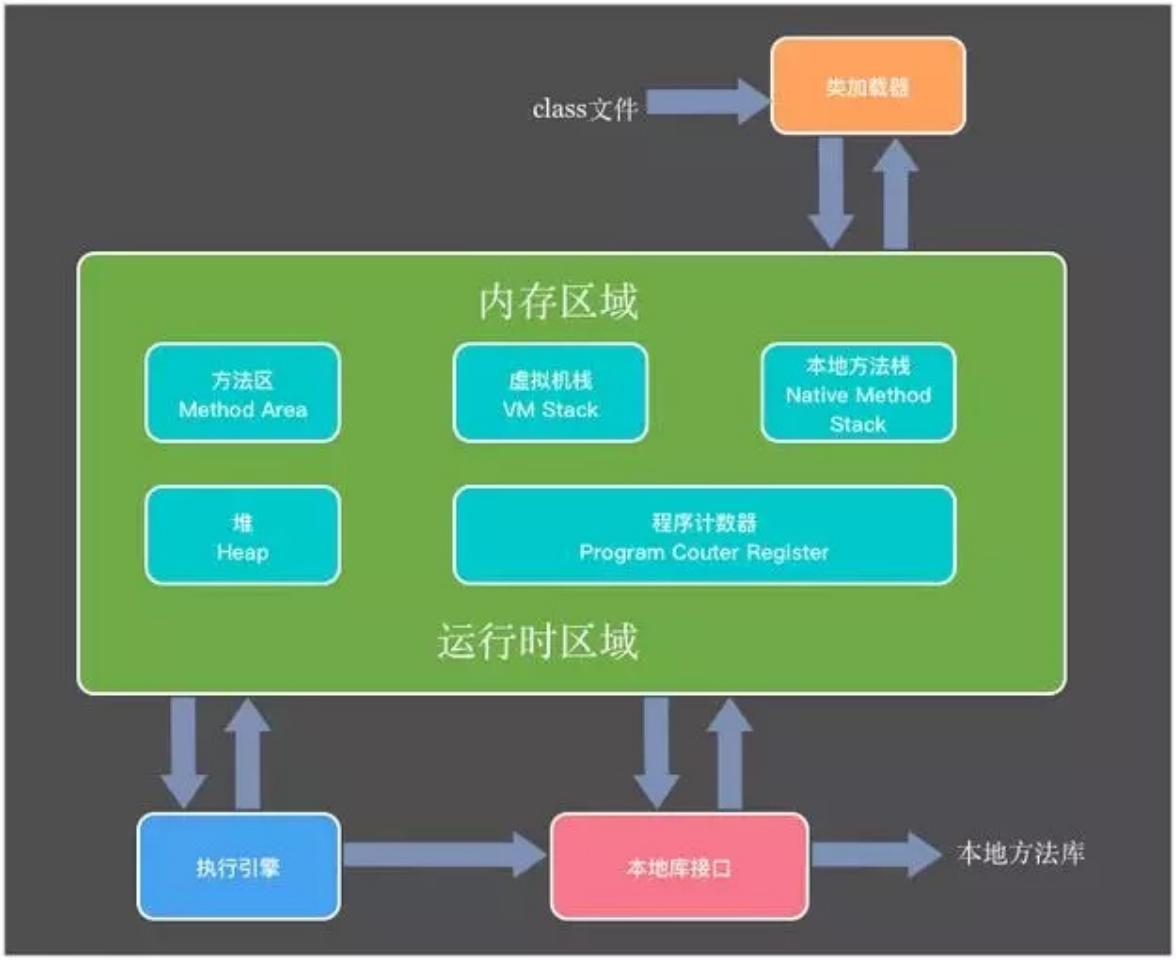


Java中会有内存泄漏吗



ClassLoader如何加载class

JVM内存模型图


class文件
类加载器
内存区域
方法区
Method Area
虚拟机栈
VM Stack
本地方法栈
Native Method
Stack
堆
Heap
程序计数器
Program Couter Register
运行时区域
执行引擎
本地库接口
本地方法库

1.Java泛型中的泛型参数能不能是基本类型，比如ArrayList中的T可不可以是int，为什么？ArrayList中可不可以插入数字，如果可以，请用代码示例，如果不可以，请说明原因？

答：不能，泛型要求能包容的是对象类型，而基本类型在java里不属于对象。

ArrayList<String>中不可以插入数字，因为他的泛型参数规定是String类型，所以只能插入字符串。但是如果非要插入数字，有三种方法可以解决。比如泛型参数改成Integer，或者不声明泛型类型或者是数字和空字符串拼接。

2.什么是值传递和引用传递？

答：值传递：传递的是实际参数的一个副本，这个值可能是基本类型，也可能是引用类型的地址。意思指的是在方法调用时，传递的参数是按值的拷贝传递。

引用传递：传递的是实际参数的地址的一个副本。意思指的是在方法调用时，传递的参数是按引用进行传递，其实传递的引用的地址，也就是变量所对应的内存空间的地址。

在java中，只有值传递.引用传递其实也就是内存地址值的传递。

3.Java集合类框架的基本接口有哪些？请说明各自的用途

- Collection：代表一组对象，每一个对象都是它的子元素。
- Set：不包含重复元素的Collection。
- List：有顺序的collection，并且可以包含重复元素。
- Map：可以把键(key)映射到值(value)的对象，键不能重复。

4.下面的代码中list对象总共扩容了几次，请详细讲述一下ArrayList的扩容规则。

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

for (int i = 0; i < 20; i++) {
    list.add(Integer.toString(i));
}
```

答：总共扩容了2次。它默认是大小（size）是10个，每次扩容都会比较现在的容量是否够用，如果不够用就扩容1.5倍，调用Arrays.copyOf方法，所以每次扩容都是new一个新的数组！如果一个list初始化容量为10，如果需要添加1000个对象，需要初始化11次，而每次都生成新的数组对象，将原来的对象复制到新数组对象里，就是说每次都要丢弃原来的数组对象，这样的操作确实特别费时又占用内存（虽说只是复制对象的引用，但数组对象本身也是需要占用内存的）。看来如果元素很多的话，估计数量初始化一个容量还是很有必要的。

5.请详细讲述一下RandomAccess接口有什么作用

答：RandomAccess用来当标记的，是一种标记接口，接口的非典型用法。意思是，随机访问任意下标元素都比较快。用处，当要实现某些算法时，会判断当前类是否实现了RandomAccess接口，会根据结果选择不同的算法

6. 请详细讲述一下HashMap的内部实现原理。

答：HashMap的底层实现都是数组+链表结构实现的。添加、删除、获取元素时都是先计算hash，根据hash和table.length计算index也就是table数组的下标，然后进行相应操作。HashMap创建时会默认初始化时创建一个默认容量为16的Entry数组，默认加载因子为0.75，同时设置临界值为16*0.75。

7. 请编写一段Java代码，对数组{1, 2, 3, 4, 5, 6, 7, 8, 9, 0}进行随机排序。

答：代码如下

```java
public class RandomSort {
    public static void main(String[] args) {

        //第一步：定义一个数组
        int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
        //第二步：定义两个临时变量。index代表随机的下标，t是临时存储变量
        int index, t;
        //第三步：开始进行随机排序
        for (int i = 0; i < a.length; i++) {
            //生成一个随机下标
            index = new Random().nextInt(3);
            //每次按顺序的遍历的值和随机下标进行值交换
            t = a[i];
            a[i] = a[index];
            a[index] = t;
        }
        //第五步：打印随机排序后的数组
        for (int i : a) {
            System.out.print(i+" ");
        }
    }
}
```

打印后的结果是：

```
8 0 9 2 3 5 4 7 6 1
```

8. 请用您认为最优的算法计算斐波那契数列：F(n)=F(n-1)+F(n-2) (n为自然数，F(0)=0, F(1)=1)。

答：

```java
public class Fibonacci{
    //定义三个变量方法
    public static void main(String[] args) {
        int a = 1, b = 1, c = 0;
        System.out.println("斐波那契数列前20项为：");
        System.out.print(a + "\t" + b + "\t");
        for (int i = 1; i <= 18; i++) {
            c = a + b;
            a = b;
            b = c;
            System.out.print(c + "\t");
            if ((i + 2) % 5 == 0)
                System.out.println();
        }
    }
}
```

9. 请详细讲述一下Java中volatile关键字的作用。

答：用volatile修饰的变量，线程在每次使用变量的时候，都会读取变量修改后的最的值。volatile很容易被误用，用来进行原子性操作。

主要用在多线程，同步变量。 线程为了提高效率，将某成员变量(如A)拷贝了一份（如B），线程中对A的访问其实访问的是B。只在某些动作时才进行A和B的同步。因此存在A和B不一致的情况。volatile就是用来避免这种情况的。volatile告诉jvm， 它所修饰的变量不保留拷贝，直接访问主内存中的（也就是上面说的A)。在Java内存模型中，有main memory，每个线程也有自己的memory (例如寄存器)。为了性能，一个线程会在自己的memory中保持要访问的变量的副本。这样就会出现同一个变量在某个瞬间，在一个线程的memory中的值可能与另外一个线程memory中的值，或者main memory中的值不一致的情况。

一个变量声明为volatile，就意味着这个变量是随时会被其他线程修改的，因此不能将它cache在线程memory中。

10. 写一段代码在遍历 ArrayList 时移除一个元素。

答：我们知道ArrayList的底层是用数组实现的，如果你删除了其中一个元素，那么后边的元素都会向前移动。所以在遍历时如果删除元素，就要小心了。

第一种方法，用数组下标进行遍历，如果需要删除元素，我们从后向前遍历，这样不论有没有元素删除，我们都不会遗漏未被遍历的元素。

第二种方法，我们使用迭代器。

```
Iterator itr = list.iterator();

while(itr.hasNext()) {
      if(...) {
        itr.remove();
      }
}
```

总之，如果你的删除操作比较多的话，建议使用LinkedList。

11. 请详细解释一下Java泛型中<? super T>和<? extends T>的作用和区别。

答：<? super T>表示包括T在内的任何T的父类，<? extends T>表示包括T在内的任何T的子类。

遵循一个原则则是，生产者（Producer）使用extends，消费者（Consumer）使用super。

生产者使用extends

如果你需要一个列表提供T类型的元素（即你想从列表中读取T类型的元素），你需要把这个列表声明成< extends T>，比如List< extends Integer>，因此你不能往该列表中添加任何元素。

消费者使用super

如果需要一个列表使用T类型的元素（即你想把T类型的元素加入到列表中），你需要把这个列表声明成< super T>，比如List< super Integer>，因此你不能保证从中读取到的元素的类型。

即是生产者，也是消费者

如果一个列表即要生产，又要消费，你不能使用泛型通配符声明列表，比如List<Integer>。

12. 请用JDBC写一段访问数据库读取数据的代码，SL语句和驱动自己选择。

```java
//查询数据库中的内容
public class ResultSetDemon01 {

    public static final String DBDRIVER = "oracle.jdbc.driver.OralceDriver";

    public static final String DBURL = "jdbc:oracle:thin:@locast:1521:orcl";

    public static final String DBUSER = "admin";

    public static final String DBPASS = "admin";

    public static void main(String[] args) throws Exception{

        Connection conn = null;//表示数据库连接的对象

        Statement stmt = null;//表示数据库更新操作

        ResultSet result = null;//表示接受数据库查询到的结果

        Class.forName(DBDRIVER);//使用class类加载驱动程序

        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS);//连接数据库

        stmt = conn.createStatement();//tatement接口需要通过connection接口进行实例化操作

        result = stmt.executeQuery("select * from person");//执行sql语句，结果集放在result中

        while(result.next()){//判断是否还有下一行
            String name = result.getString("name");//获取数据库person表中name字段的值
            System.out.println(name);
        }
        result.close();
        stmt.close();
        conn.close();

    }
```

13. 请谈谈Proxy模式，Adapter模式，Decorator模式以及Façade模式分别的使用场景和区别。

答：**Proxy模式**是设计模式中的代理模式，比如我去优衣库买衣服，那么我肯定不会直接去优衣库工厂（生产衣服工厂）去买衣服吧，那么我们直接去门店啊或者代理店去买。这些地方其实就是优衣库造衣工厂的代理。

什么时候开始使用呢？当我们需要使用的对象很复杂或者需要很长时间去构造，这时就可以使用代理模式(Proxy)。例如：如果构建一个对象很耗费时间和计算机资源，代理模式(Proxy)允许我们控制这种情况，直到我们需要使用实际的对象。一个代理(Proxy)通常包含和将要使用的对象同样的方法，一旦开始使用这个对象，这些方法将通过代理(Proxy)传递给实际的对象。 一些可以使用代理模式(Proxy)的情况：

一个对象，比如一幅很大的图像，需要载入的时间很长。

一个需要很长时间才可以完成的计算结果，并且需要在它计算过程中显示中间结果

一个存在于远程计算机上的对象，需要通过网络载入这个远程对象则需要很长时间，特别是在网络传输高峰期。

一个对象只有有限的访问权限，代理模式(Proxy)可以验证用户的权限

代理模式(Proxy)也可以被用来区别一个对象实例的请求和实际的访问，例如：在程序初始化过程中可能建立多个对象，但并不都是马上使用，代理模式(Proxy)可以载入需要的真正的对象。这是一个需要载入和显示一幅很大的图像的程序，当程序启动时，就必须确定要显示的图像，但是实际的图像只能在完全载入后才可以显示！这时我们就可以使用代理模式(Proxy)。

Adapter模式是设计模式的适配器模式,它主要是将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以在一起工作。

使用场景即系统需要使用现有的类，而这些类的接口不符合系统的接口。

想要建立一个可以重用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作。

两个类所做的事情相同或相似，但是具有不同接口的时候。

旧的系统开发的类已经实现了一些功能，但是客户端却只能以另外接口的形式访问，但我们不希望手动更改原有类的时候。

使用第三方组件，组件接口定义和自己定义的不同，不希望修改自己的接口，但是要使用第三方组件接口的功能。

Decorator模式即设计模式中的装饰器模式。它能动态地给一个对象添加一些额外的职责或者行为。就增加功能来说， Decorator模式相比生成子类更为灵活。它提供了改变子类的灵活方案。装饰器模式在不必改变原类文件和使用继承的情况下，动态的扩展一个对象的功能。它是通过创建一个包装对象，也就是装饰来包裹真实的对象。

当用于一组子类时，装饰器模式更加有用。如果你拥有一族子类（从一个父类派生而来），你需要在与子类独立使用情况下添加额外的特性，你可以使用装饰器模式，以避免代码重复和具体子类数量的增加。

Façade模式即设计模式的外观设计模式。它为子系统中的一组接口提供一个一致的界面，Facade模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

应用场景是

当你要为一个复杂子系统提供一个简单接口时。

客户程序与抽象类的实现部分之间存在着很大的依赖性。

当你需要构建一个层次结构的子系统时，使用Facade模式定义子系统中每层的入口点。仅通过facade进行通讯。

14. OOP的基本特性和设计原则有哪些。

答：OOP的三大基本特性是抽象与封装，继承与多态。

1.抽象与封装:

抽象是把系统中需要处理的数据和在这些数据上的操作结合在一起，根据功能、性质和用途等因素抽象成不同的抽象数据类型。每个抽象数据类型既包含了数据，又包含了针对这些数据的授权操作。在面向对象的程序设计中，抽象数据类型是用“类”这种结构来实现的，每个类里都封装了相关的数据和操作。

封装是指利用抽象数据类型和基于数据的操作结合在一起，数据被保护在抽象数据类型的内部，系统的其他部分只有通过包裹在数据之外被授权的操作，才能与这个抽象数据类型进行交互。

2. 继承：

它是与传统方法不同的一个最有特色的方法。它是面向对象的程序中两个类之间的一种关系，即一个类可以从另一个类（即它的父类）继承状态和行为。继承父类的类称为子类。

继承的优越性：通过使用继承，程序员可以在不同的子类中多次重新使用父类中的代码，使程序结构清晰，易于维护和修改，而子类又可以提供一些特殊的行为，这些特殊的行为在父类中是没有的

3.多态：

是指一个程序中同名的方法共存的情况，调用者只需使用同一个方法名，系统会根据不同情况，调用相应的不同方法，从而实现不同的功能。多态性又被称为“一个名字，多个方法”。

设计原则有五种:

单一职责原则(Single-Resposibility Principle)。"对一个类而言，应该仅有一个引起它变化的原因。"本原则是我们非常熟悉地"高内聚性原则"的引申，但是通过将"职责"极具创意地定义为"变化的原因"，使得本原则极具操作性，尽显大师风范。同时，本原则还揭示了内聚性和耦合生，基本途径就是提高内聚性；如果一个类承担的职责过多，那么这些职责就会相互依赖，一个职责的变化可能会影响另一个职责的履行。其实OOD的实质，就是合理地进行类的职责分配。

开放封闭原则(Open-Closed principle)。"软件实体应该是可以扩展的，但是不可修改。"本原则紧紧围绕变化展开，变化来临时，如果不必改动软件实体裁的源代码，就能扩充它的行为，那么这个软件实体设计就是满足开放封闭原则的。如果说我们预测到某种变化，或者某种变化发生了，我们应当创建抽象类来隔离以后发生的同类变化。在Java中，这种抽象是指抽象基类或接口；在C++中，这各抽象是指抽象基类或纯抽象基类。当然，没有对所有情况都贴切的模型，我们必须对软件实体应该面对的变化做出选择。

Liskov替换原则(Liskov-Substituion Principle)。"子类型必须能够替换掉它们的基类型。"本原则和开放封闭原则关系密切，正是子类型的可替换性，才使得使用基类型模块无需修改就可扩充。Liskov替换原则从基于契约的设计演化而来，契约通过为每个方法声明"先验条件"和"后验条件"；定义子类时，必须遵守这些"先验条件"和"后验条件"。当前基于契的设计发展势头正劲，对实现"软件工厂"的"组装生产"梦想是一个有力的支持。

依赖倒置原则(Dependecy-Inversion Principle)。"抽象不应依赖于细节，细节应该依赖于抽象。"本原则几乎就是软件设计的正本清源之道。因为人解决问题的思考过程是先抽象后具体，从笼统到细节，所以我们先生产出的势必是抽象程度比较高的实体，而后才是更加细节化的实体。于是，"细节依赖于抽象"就意味着后来的依赖于先前的，这是自然而然的重用之道。而且，抽象的实体代表着笼而统之的认识，人们总是比较容易正确认识它们，而且本身也是不易变的，依赖于它们是安全的。依赖倒置原则适应了人类认识过程的规律，是面向对象设计的标志所在。

接口隔离原则(Interface-Segregation Principle)。"多个专用接口优于一个单一的通用接口。"本原则是单一职责原则用于接口设计的自然结果。一个接口应该保证，实现该接口的实例对象可以只呈现为单一的角色；这样，当某个客户程序的要求发生变化，而迫使接口发生改变时，影响到其他客户程序的可能生性小。

良性依赖原则。"不会在实际中造成危害的依赖关系，都是良性依赖。"通过分析不难发现，本原则的核心思想是"务实"，很好地揭示了极限编程(Extreme Programming)中"简单设计""各"重构"的理论基础。本原则可以帮助我们抵御"面向对象设计五大原则"以及设计模式的诱惑，以免陷入过度设计(Over-engineering)的尴尬境地，带来不必要的复杂性。

15. 请谈谈Java中的多线程相关的内容以及各种相关的锁。

答：实现Java中的多线程有两种方式。一是直接继承Thread类，二是实现Runnable接口。通常我们实现Runnabale接口方式去做。好处如下：

(1)适合多个相同程序代码的线程去处理同一资源的情况，把虚拟CPU（线程）同程序的代码，数据有效的分离，较好地体现了面向对象的设计思想。

(2)可以避免由于Java的单继承特性带来的局限。我们经常碰到这样一种情况，即当我们要将已经继承了某一个类的子类放入多线程中，由于一个类不能同时有两个父类，所以不能用继承Thread类的方式，那么，这个类就只能采用实现Runnable接口的方式了。

(3)有利于程序的健壮性，代码能够被多个线程共享，代码与数据是独立的。当多个线程的执行代码来自同一个类的实例时，即称它们共享相同的代码。 多个线程操作相同的数据，与它们的代码无关。当共享访问相同的对象是，即它们共享相同的数据。当线程被构造时，需要的代码和数据通过一个对象作为构造函数 实参传递进去，这个对象就是一个实现了Runnable接口的类的实例

Java中多线程的状态是

1.新建状态（New）：新创建了一个线程对象。

2.就绪状态（Runnable）：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3.运行状态（Running）：就绪状态的线程获取了CPU，执行程序代码。

4.阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5.死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

Java多线程锁有对象锁和类锁。对象级别锁是一个机制，当你想同步一个非静态方法或者非静态代码块，让在给定的类实例中只有一个线程来执行这个代码块，这就可以使得实例级别的数据是线程安全的。类级别锁是在所有可变的实例或者运行环境中，类级别锁阻止多线程进入同步块，也就是说，如果运行环境中有DemoClass的100个实例，在任何时刻，只能有DemoClass的一个实例来执行它的demoMethod()方法，所有其他的DemoClass实例在其他线程中只能处于阻塞状态，这使得静态数据是线程安全的。

如果要应聘高级开发工程师职务，仅仅懂得Java的基础知识是远远不够的，还必须懂得常用数据结构、算法、网络、操作系统等知识。因此本文不会讲解具体的技术，笔者综合自己应聘各大公司的经历，整理了一份大公司对Java高级开发工程师职位的考核纲要，希望可以帮助到需要的人。

当前，市面上有《Java XX宝典》类似的图书，而且图书中的内容都着重在讲解Java最为基础的部分，最严重的是，里面有着大量错误的内容，极具误导性。另外，网上也有各种各样的Java面试题，很多也是着重在Java语言基础上。实际上，如果要应聘高级开发工程师职务，仅仅懂得Java的基础知识是远远不够的，还必须懂得常用数据结构、算法、网络、操作系统等知识。因此本文不会讲解具体的技术，笔者综合自己应聘各大公司的经历，整理了一份大公司对Java高级开发工程师职位的考核纲要，希望可以帮助到需要的人。

1 Java基础

1.1 Collection和Map

(1)掌握Collection和Map的继承体系。

(2)掌握ArrayList、LinkedList、Vector、Stack、PriorityQueue、HashSet、LinkedHashSet、TreeSet、HashMap、LinkedHashMap、TreeMap、WeakHashMap、EnumMap、TreeMap、HashTable的特点和实现原理。

(3)掌握CopyOnWriteArrayList、CopyOnWriteArraySet、ConcurrentHashMap的实现原理和适用场景。

1.2 IO

(1)掌握InputStream、OutputStream、Reader、Writer的继承体系。

(2)掌握字节流(FileInputStream、DataInputStream、BufferedInputStream、FileOutputSteam、DataOutputStream、BufferedOutputStream)和字符流(BufferedReader、InputStreamReader、FileReader、BufferedWriter、OutputStreamWriter、PrintWriter、FileWriter)，并熟练运用。

(3)掌握NIO实现原理及使用方法。

1.3 异常

(1)掌握Throwable继承体系。

(2)掌握异常工作原理。

(3)了解常见受检异常(比如FileNotFoundException)、非受检异常(比如NullPointerException)和错误(比如IOError)。

1.4 多线程

(1)掌握Executors可以创建的三种(JAVA8增加了一种，共四种)线程池的特点及适用范围。

(2)掌握多线程同步机制，并熟练运用。

1.5 Socket

(1)掌握Socket通信原理。

(2)熟练使用多线程结合Socket进行编程。

2 Java虚拟机

2.1 JVM内存区域划分

(1)掌握程序计数器、堆、虚拟机栈、本地方法栈、方法区（JAVA8已移除）、元空间（JAVA8新增）的作用及基本原理。

(2)掌握堆的划分：新生代（Eden、Survivor1、Survivor2）和老年代的作用及工作原理。

(3)掌握JVM内存参数设置及调优。

2.2 类加载

(1)掌握类的加载阶段：加载、链接（验证、准备、解析）、初始化、使用、卸载。

(2)掌握类加载器分类及其应用：启动类加载器、扩展类加载器、应用程序类加载器、自定义加载器。

3 J2EE

(1) 掌握JSP内置对象、动作及相关特点和工作原理。

(2) 掌握Servlet的特点和工作原理。

(3) 掌握Spring框架的IOC和AOP实现原理（反射和动态代理）。

(4) 至少掌握一个MVC框架（Spring MVC，Struts等）的工作原理，并熟练运用。

(5) 至少掌握一个ORM框架(Hibernate，MyBatis等)的工作原理，并熟练运用。

4 数据结构与算法

(1)掌握线性表和树的特点并熟练运用。

(2)掌握常用排序和查找算法：插入排序(直接插入排序、希尔排序)、选择排序(直接选择排序、堆排序)、交换排序(冒泡排序、快速排序)、归并排序，顺序查找、二分查找、哈希查找。

(3) 熟练运用常见排序和查找算法思想解决编程问题。

(4)了解几大基本算法：贪心算法、分治策略、动态规划。

5 计算机网络

(1)掌握网络的分层结构，及每层的功能特点。

(2)掌握TCP/IP的通信原理(三次握手、四次挥手)

6 数据库

(1)掌握复杂的SQL语句编写。

(2)掌握数据库的优化（SQL层面和表设计层面）。

(3)至少掌握一款数据库产品。

(4)熟悉高并发、大数据情况下的数据库开发。

7 Web技术

(1)掌握AJAX的工作原理。

(2)至少熟悉一款JS框架(比如JQuery)。

8 设计模式

(1)熟悉常见的设计模式。

(2)会将设计模式理论应用到实际开发中。

9 Linux

(1)熟练运用Linux常见命令。

(2)熟悉Linux操作系统基本概念及特点。

(3)熟悉Shell脚本。

10 操作系统

(1)掌握操作系统的进程管理。

(2)了解操作系统的I/O。

11 正则表达式

(1)掌握常见正则表达式符号。

(2)熟练运用正则表达式解决实际问题(比如匹配电话号码、邮箱、域名等)。

问题：如果main方法被声明为private会怎样？

答案：能正常编译，但运行的时候会提示”main方法不是public的”。

问题：Java里的传引用和传值的区别是什么？

答案：传引用是指传递的是地址而不是值本身，传值则是传递值的一份拷贝。

问题：如果要重写一个对象的equals方法，还要考虑什么？

答案：hashCode。

问题：Java的”一次编写，处处运行”是如何实现的？

答案：Java程序会被编译成字节码组成的class文件，这些字节码可以运行在任何平台，因此Java是平台独立的。

问题：说明一下public static void main(String args[])这段声明里每个关键字的作用

答案：public: main方法是Java程序运行时调用的第一个方法，因此它必须对Java环境可见。所以可见性设置为pulic.

static: Java平台调用这个方法时不会创建这个类的一个实例，因此这个方法必须声明为static。

void: main方法没有返回值。

String是命令行传进参数的类型，args是指命令行传进的字符串数组。

问题：==与equals的区别

答案：==比较两个对象在内存里是不是同一个对象，就是说在内存里的存储位置一致。两个String对象存储的值是一样的，但有可能在内存里存储在不同的地方 .

==比较的是引用而equals方法比较的是内容。public boolean equals(Object obj) 这个方法是由Object对象提供的，可以由子类进行重写。默认的实现只有当对象和自身进行比较时才会返回true,这个时候和==是等价的。String, BitSet, Date, 和File都对equals方法进行了重写，对两个String对象 而言，值相等意味着它们包含同样的字符序列。对于基本类型的包装类来说，值相等意味着对应的基本类型的值一样。

```
public class EqualsTest {
               public static void main(String[] args) {
                              String s1 = “abc”;
                              String s2 = s1;
                              String s5 = “abc”;
                              String s3 = new String(”abc”);
                              String s4 = new String(”abc”);
                              System.out.println(”== comparison : ” + (s1 == s5));
                              System.out.println(”== comparison : ” + (s1 == s2));
                              System.out.println(”Using equals method : ” + s1.equals(s2));
                              System.out.println(”== comparison : ” + s3 == s4);
                              System.out.println(”Using equals method : ” + s3.equals(s4));
               }
}
```

结果：

```
== comparison : true
== comparison : true
Using equals method : true
false
Using equals method :true
```

问题：如果去掉了main方法的static修饰符会怎样?

答案：程序能正常编译。运行时会抛NoSuchMethodError异常。

问题：为什么oracle type4驱动被称作瘦驱动?

答案：oracle提供了一个type 4 JDBC驱动，被称为瘦驱动。这个驱动包含了一个oracle自己完全用Java实现的一个TCP/IP的Net8的实现，因此它是平台独立的，可以在运行时由浏览器下载，不依赖任何客户端 的oracle实现。客户端连接字符串用的是TCP/IP的地址端口，而不是数据库名的tnsname。

问题：介绍一下finalize方法

答案： final: 常量声明。 finally: 处理异常。 finalize: 帮助进行垃圾回收。

接口里声明的变量默认是final的。final类无法继承，也就是没有子类。这么做是出于基础类型的安全考虑，比如String和Integer。这样也使得编译器进行一些优化，更容易保证线程的安全性。final方法无法重写。final变量的值不能改变。finalize()方法在一个对象被销毁和回收前会被调用。finally,通常用于异常处理，不管有没有异常被抛出都会执行到。比如，关闭连接通常放到finally块中完成。

问题：什么是Java API?

答案：Java API是大量软件组件的集合，它们提供了大量有用的功能，比如GUI组件。

问题：GregorianCalendar类是什么东西?

答案：GregorianCalendar提供了西方传统日历的支持。

问题：ResourceBundle类是什么?

答案：ResourceBundle用来存储指定语言环境的资源，应用程序可以根据运行时的语言环境来加载这些资源，从而提供不同语言的展示。

问题：为什么Java里没有全局变量?

答案：全局变量是全局可见的，Java不支持全局可见的变量，因为：全局变量破坏了引用透明性原则。全局变量导致了命名空间的冲突。

问题：如何将String类型转化成Number类型?

答案：Integer类的valueOf方法可以将String转成Number。下面是代码示例：

```
String numString = “1000”;
int id=Integer.valueOf(numString).intValue();
```

问题：SimpleTimeZone类是什么?

答案：SimpleTimeZone提供公历日期支持。

问题：while循环和do循环有什么不同?

答案：while结构在循环的开始判断下一个迭代是否应该继续。do/while结构在循环的结尾来判断是否将继续下一轮迭代。do结构至少会执行一次循环体。

问题：Locale类是什么?

答案：Locale类用来根据语言环境来动态调整程序的输出。

问题：面向对象编程的原则是什么?

答案：主要有三点，多态，继承和封装。

问题：介绍下继承的原则

答案：继承使得一个对象可以获取另一个对象的属性。使用继承可以让已经测试完备的功能得以复用，并且可以一次修改，所有继承的地方都同时生效。

问题：什么是隐式的类型转化?

答案：隐式的类型转化就是简单的一个类型赋值给另一个类型，没有显式的告诉编译器发生了转化。并不是所有的类型都支持隐式的类型转化。

代码示例：

```
int i = 1000;
long j = i; //Implicit casting
```

问题：sizeof是Java的关键字吗?

答案：不是。

问题：native方法是什么?

答案：native方法是非Java代码实现的方法。

问题：在System.out.println()里面,System, out, println分别是什么?

答案：System是系统提供的预定义的final类，out是一个PrintStream对象，println是out对象里面一个重载的方法。

问题：封装，继承和多态是什么?

答案：简单来说，多态是指一个名字多种实现。多态使得一个实体通过一个通用的方式来实现不同的操作。具体的操作是由实际的实现来决定的。

多态在Java里有三种表现方式：方法重载通过继承实现方法重写通过Java接口进行方法重写。

问题：显式的类型转化是什么?

答案：显式的类型转化是明确告诉了编译器来进行对象的转化。

代码示例：

```
long i = 700.20;
int j = (int) i; //Explicit casting
```

问题：什么是Java虚拟机?

答案：Java虚拟机是能移植到不同硬件平台上的软件系统。

问题：类型向下转换是什么?

答案：向下转换是指由一个通用类型转换成一个具体的类型，在继承结构上向下进行。

问题：Java的访问修饰符是什么?

答案：访问权限修饰符是表明类成员的访问权限类型的关键字。使用这些关键字来限定程序的方法或者变量的访问权限。它们包含：

public: 所有类都可以访问 protected: 同一个包内以及所有子类都可以访问 private: 只有归属的类才能访问默认: 归属类及相同包下的子类可以访问

问题：所有类的父类是什么?

答案：Object.

问题：Java的基本类型有哪些?

答案：byte,char, short, int, long, float, double, boolean。

问题：静态类型有什么特点?

答案：静态变量是和类绑定到一起的，而不是类的实例对象。每一个实例对象都共享同样一份静态变量。也就是说，一个类的静态变量只有一份，不管它有多少个对象。类变量或者说静态变量是通过static这个关键字来声明的。类变量通常被用作常量。静态变量通常通过类名字来进行访问。当程序运行的时候这个变量就会创建直到程序结束后才会被销毁。类变量的作用域和实例变量是一样的。它的初始值和成员变量也是一样的，当变量没被初始化的时候根据它的数据类型，会有一个默认值。类似的，静态方法是属于类的方法，而不是类对象，它的调用并不作用于类对象，也不需要创建任何的类实例。静态方法本身就是final的，因为重写只会发生在类实例上，静态方法是和类绑定在一起的，不是对象。父类的静态方法会被子类的静态方法屏蔽，只要原来方法没有声明为final。非静态方法不能重写静态方法，也就是说，你不能在子类中把一个静态方法改成实例方法。

非静态变量在每一个对象实例上都有单独的一份值。

问题：&操作符和&&操作符有什么区别?

答案：当一个&表达式在求值的时候，两个操作数都会被求值，&&更像是一个操作符的快捷方式。当一个&&表达式求值的时候，先计算第一个操作数，如果它返回true才会计算第二个操作数。如果第一个操作数取值为fale,第二个操作数就不会被求值。

问题：Java是如何处理整型的溢出和下溢的?

答案：Java根据类型的大小，将计算结果中的对应低阶字节存储到对应的值里面。

问题：public static void写成static public void会怎样?

答案：程序正常编译及运行。

问题，声明变量和定义变量有什么不同?

答案：声明变量我们只提供变量的类型和名字，并没有进行初始化。定义包括声明和初始化两个阶段String s;只是变量声明，String s = new String("bob"); 或者String s = "bob";是变量定义。

问题：Java支持哪种参数传递类型?

答案：Java参数都是进行传值。对于对象而言，传递的值是对象的引用，也就是说原始引用和参数引用的那个拷贝，都是指向同一个对象。

问题：对象封装的原则是什么?

答案：封装是将数据及操作数据的代码绑定到一个独立的单元。这样保障了数据的安全，防止外部代码的错误使用。对象允许程序和数据进行封装，以减少潜在的干涉。对封装的另一个理解是作为数据及代码的保护层，防止保护层外代码的随意访问。

问题：你怎么理解变量?

答案：变量是一块命名的内存区域，以便程序进行访问。变量用来存储数据，随着程序的执行，存储的数据也可能跟着改变。

问题：数值提升是什么?

答案：数值提升是指数据从一个较小的数据类型转换成为一个更大的数据类型，以便进行整型或者浮点型运算。在数值提升的过程中，byte,char,short值会被转化成int类型。需要的时候int类型也可能被提升成long。long和float则有可能会被转换成double类型。

问题：Java的类型转化是什么?

答案：从一个数据类型转换成另一个数据类型叫做类型转换。Java有两种类型转换的方式，一个是显式的类型转换，一个是隐式的。

问题：main方法的参数里面，字符串数组的第一个参数是什么?

答案：数组是空的，没有任何元素。不像C或者C++，第一个元素默认是程序名。如果命令行没有提供任何参数的话，main方法中的String数组为空,但不是null。

问题：怎么判断数组是null还是为空?

答案：输出array.length的值，如果是0,说明数组为空。如果是null的话，会抛出空指针异常。

问题：程序中可以允许多个类同时拥有都有main方法吗?

答案：可以。当程序运行的时候，我们会指定运行的类名。JVM只会在你指定的类中查找main方法。因此多个类拥有main方法并不存在命名冲突的问题。

问题：静态变量在什么时候加载？编译期还是运行期？静态代码块加载的时机呢？

答案：当类加载器将类加载到JVM中的时候就会创建静态变量，这跟对象是否创建无关。静态变量加载的时候就会分配内存空间。静态代码块的代码只会在类第一次初始化的时候执行一次。一个类可以有多个静态代码块，它并不是类的成员，也没有返回值，并且不能直接调用。静态代码块不能包含this或者super,它们通常被用初始化静态变量。

问题：一个类能拥有多个main方法吗？

答案：可以，但只能有一个main方法拥有以下签名：

```
public static void main(String[] args) {}
```

否则程序将无法通过编译。编译器会警告你main方法已经存在。

问题：简单的介绍下JVM是如何工作的?

答案：JVM是一台抽象的计算机，就像真实的计算机那样，它们会先将.java文件编译成.class文件（.class文件就是字节码文件）,然后用它的解释器来加载字节码。

问题：如果原地交换两个变量的值?

答案：先把两个值相加赋值给第一个变量，然后用得到的结果减去第二个变量，赋值给第二个变量。再用第一个变量减去第二个变量，同时赋值给第一个变量。代码如下：

```
int a=5,b=10;a=a+b; b=a-b; a=a-b;
```

使用异或操作也可以交换。第一个方法还可能会引起溢出。异或的方法如下： int a=5,b=10;a=a+b; b=a-b; a=a-b;

```
int a = 5; int b = 10;
a = a ^ b;
b = a ^ b;
a = a ^ b;
```

问题：什么是数据的封装?

答案：数据封装的一种方式是在类中创建set和get方法来访问对象的数据变量。一般来说变量是private的，而get和set方法是public的。封装还可以用来在存储数据时进行数据验证，或者对数据进行计算，或者用作自省（比如在struts中使用javabean）。把数据和功能封装到一个独立的结构中称为数据封装。封装其实就是把数据和关联的操作方法封装到一个独立的单元中，这样使用关联的这些方法才能对数据进行访问操作。封装提供的是数据安全性,它其实就是一种隐藏数据的方式。

问题：什么是反射API？它是如何实现的?

答案：反射是指在运行时能查看一个类的状态及特征，并能进行动态管理的功能。这些功能是通过一些内建类的反射API提供的，比如Class,Method,Field, Constructors等。使用的例子：使用Java反射API的getName方法可以获取到类名。

问题：JVM自身会维护缓存吗，是不是在堆中进行对象分配，操作系统的堆还是JVM自己管理的堆？为什么？

答案：是的，JVM自身会管理缓存，它在堆中创建对象，然后在栈中引用这些对象。

问题：虚拟内存是什么？

答案：虚拟内存又叫延伸内存，实际上并不存在真实的物理内存。

问题：方法可以同时即是static又是synchronized的吗？

答案：可以。如果这样做的话，JVM会获取和这个对象关联的java.lang.Class实例上的锁。这样做等于：

```
synchronized(XYZ.class) {
}
```

问题：String和StringTokenizer的区别是什么？

答案：StringTokenizer是一个用来分割字符串的工具类。

```
StringTokenizer st = new StringTokenizer(”Hello World”);
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

输出：

```
Hello
World
```

问题：transient变量有什么特点？

答案：transient变量不会进行序列化。例如一个实现Serializable接口的类在序列化到ObjectStream的时候，transient类型的变量不会被写入流中，同时，反序列化回来的时候，对应变量的值为null。

问题：哪些容器使用Border布局作为它们的默认布局？

答案：Window, Frame, Dialog。

问题：怎么理解什么是同步？

答案：同步用来控制共享资源在多个线程间的访问，以保证同一时间内只有一个线程能访问到这个资源。在非同步保护的多线程程序里面，一个线程正在修改一个共享变量的时候，可能有另一个线程也在使用或者更新它的值。同步避免了脏数据的产生。

对方法进行同步：

```
public synchronized void Method1 () {
// Appropriate method-related code.
}
```

在方法内部对代码块进行同步：

```
public myFunction (){
    synchronized (this) {
            // Synchronized code here.
         }
}
```

115个Java面试题及回答

简介 在本教程中,我们将讨论在Java面试中,用人单位用来测试应聘者Java以及面向对象的能力的面试题目.以下章节我们将按照以下结构讨论面试问题，面向对象编程及其特性，Java及其特性的一般问题,集合,垃圾回收，异常处理,Java applets,Swing,JDBC,RMI, Servlet 和 JSP. 来，我们一起出发吧。

115个Java面试题及回答

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。

1. Java面向对象思想

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：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:

- 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）
- 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. Java中的多态

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。

3. Java的异常处理

3.1 Java中异常分为哪些种类

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：

- 当前方法知道如何处理该异常，则用try...catch块来处理该异常。
- 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。

3.2 调用下面的方法，得到的返回值是什么

```
public int getNum(){
     try {
        int a = 1/0;
        return 1;
    } catch (Exception e) {
        return 2;
    } finally{
        return 3;
    }
}
```

代码在走到第3行的时候遇到了一个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。

4. Java的数据类型

4.1 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

数据类型	字节数
byte	1
char	2
short	2
int	4
float	4
double	8
long	8

boolean	1

PS：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

4.2 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。

5. Java的IO

5.1 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。

5.2 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。

5.3 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();
```

6. Java的集合

6.1 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\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\为形参，返回类型为HashMap\，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。

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\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中的数据存储在LinkedHashMap中
                for(Entry<Integer, User> entry : list){
                        linkedHashMap.put(entry.getKey(), entry.getValue());
                }
                //返回结果
                return linkedHashMap;
        }
}
```

6.2 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。

7. Java的多线程

7.1 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。

7.2 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。

7.3 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：

- 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。
- 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。

- 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
- 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
- volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞。
- 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

7.4 分析代码解释原因

```
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 + "]";
```

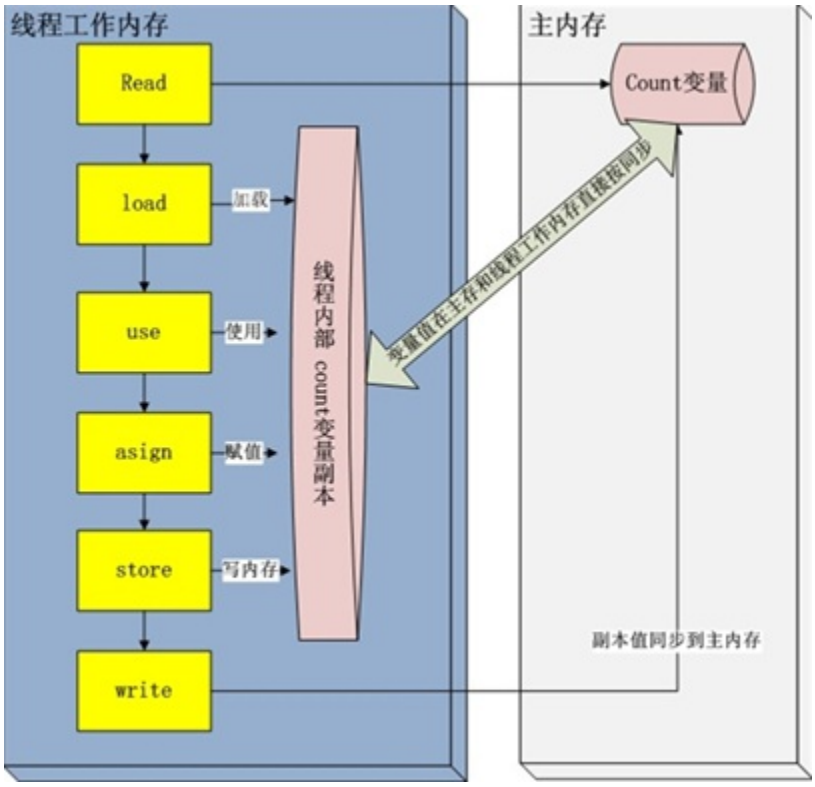
```
        }
}
public class VolatileTest {
        public static void main(String[] args) {
                final Counter counter = new Counter();
                for(int i=0;i<1000;i++){
                        new Thread(new Runnable() {
                                @Override
                                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在堆内存的变量的值，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:



7.5 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。

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

在JDK的java.util.concurrent.Executors中提供了生成多种线程池的静态方法。然后调用他们的execute方法即可。

1. Java中的反射

1.1 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

- Class.forName(className)
- 类名.class
- 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。

2. Java中的动态代理

2.1 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);
```

3. Java中的设计模式

3.1 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

3.2 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。

3.2.1 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;
        }
}
```

3.2.2 懒汉式

```
public class Singleton {
        // 声明变量
        private static volatile Singleton singleton2 = null;

        // 私有构造函数
        private Singleton2() {
        }

        // 提供对外方法
        public static Singleton2 getInstance() {
                if (singleton2 == null) {
                        synchronized (Singleton2.class) {
                                if (singleton == null) {
                                        singleton = new Singleton();
                                }
                        }
                }
                return singleton;
        }
}
```

3.3 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();
        }
}
```

3.4 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);
        }
}
```

3.5 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。

3.6 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();
        }
}
```

3.7 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);
        }
}
```

3.8 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，就这的意思是，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();
        }
}
```

面试技巧

- 程序员面试宝典
- 罗永浩新东方万字求职信
- 我在面试中最喜欢问开发者的问题，和回答思路

行业资深大牛，在实际面试百八十号的程序员工作中，深度总结什么样子的人，比较容易拿到offer！绝对值得一看哦

1. 题要做满

一般来说，程序员面试都会先由HR给一张面试题。面试题一般包含：逻辑题、代码规范、数据库理论、简单算法、高级函数应用、服务器基础。大多数公司出的题目，百度上都有。如果面试的岗位是初级开发，大胆去百度吧，只要填满就能获得好感度。如果面试的是中级开发，这些题只需要写重点就行，对做题时间会有要求。如果面试的是高级开发，在中级开发的要求上，把字写得漂亮些。

2. 面试问答

第一面，基本是技术主管/经理。可能是因为营造优越感，一定会问高级问题，即使需要的岗位是初级岗位。

提问的第一波是面试题上的细节，确定是百度的还是真的懂的。第二波，会随机提问一些技术问题。

PHP面试最喜欢问的就是常用框架和框架之间的差别。

Java面试最喜欢问关于函数构建、算法加密等问题。

前端面试，基本就拿着其他公司的界面设计问多久可以实现。

会涉及一些数据库，基本都是问增改删查的应用、数据库设计的规范。其他的可能会随机选取一些业务场景，让当场做程序设计。

整体的提问过程，能回答上一半就可以进入下一个面试节点。回答的时候态度一定要好，不知道的就回答“以前工作中没怎么用到”，千万不能胡说八道。知道的，也谦虚一点，回答“我过往是如此运用的”。

3. 面试沟通

可能是第二面，也可能是第三面，会有一个看上去和蔼可亲的人来面试。这个人基本上是某个项目负责人，面试的内容是沟通能力。

一开始也会带一点基础技术问题，基本上还是根据面试题扩展的。

然后就会开始问是否可以加班、如果需求发生变更怎么处理、对过往的工作从业务上理解多少、响应Bug会具体怎么做。回答不能啰嗦，要肯定句，比如：可以加班。而不要，模棱两可或者带有场景的回答，比如：不排斥加班，但是无意义的加班不太能接受。

4. 面试情怀

一般部门总监会谈这个问题。这部分，每个人都不同，但是未来规划一定要明确。一般程序员就两个发展：管理和技术专家。一般问这个问题，就是看短期内会不会离职，是不是一个有目标的人。

还会问一些兴趣爱好，看未来能不能融入团队氛围。

接下来就问薪资和待遇要求了。这部分，请根据自己的心走。不要觉得不好意思开口，面试是双向选择的过程。如果面试感觉比较好，当然也可以适当的降低一点要求。切记不要说出让自己下不了台的薪资。

说几个有意思的对话，在以往面试中遇见过的

注意啊，都是面试失败案例，千万别重蹈覆辙

程序员李某面试：

- Q：以前的工作是否用过敏捷开发？
- A：核心就是以人为本。开发设计功能跟着心走。

结果：回答完，面试就结束了。

程序员王某面试：

- Q：你当初做保险行业的，怎么处理用户名、密码、银行卡入库加密这块？
- A：为什么要加密？明文存就好了啊。

结果：认为明文存是理所当然不假思索的，Pass。

程序员马某面试：

- Q：PHP的常用框架了解的有哪些？
- A：ThinkPHP和Yii。
- Q：他们的区别是什么？或者说优缺点？
- A：（沉默了3分钟）其实我没有接触过Yii。

结果：后面的面试也基本是这种一知半解的节奏，Pass。

程序员刘某面试：

- Q：如果产品经理安排了一个任务给你，然后当天就需求变更了，你怎么办？
- A：这种事情绝对不允许发生第二次。

结果：应该会很难融入团队吧，Pass。

程序员面试中的5个杀手锏问题

也许你是个JavaScript巨星，为了防止被那些烦人的猎头骚扰，不得不删除你在LinkedIn上的个人资料。又或者，也许你是一个普通、可靠的合作伙伴，一年到头也只会收到2到3次的面试邀请。

不管你去面试的频率如何，下面这五个问题是每个软件工程师都应该问的——将有助于你确定自己在这家公司长期工作是否会合作愉快。

你们的企业文化是什么？

你每天将会有10至12个小时需要与同事的信仰、价值观和行为打交道。企业文化重视技术吗？尊重软件工程师吗？软件工程师在产品开发上有发言权吗？企业有没有提供便利以便于软件工程师将工作做到最好？

为了找到答案，可以问问企业从开发到测试都喜欢什么工具，Luca Bonmassar，Gild公司的联合创始人和首席技术官建议说（Gild是一个用于查找评估和招聘技术人才的SaaS平台）。如果面试官不能回答，Bonmassar说，“这通常是一个坏兆头”，说明该公司对你重视的技术并没有给予足够的重视。

他还建议询问开发流程：“开发人员的投入有多少会进入到产品？项目经理是否决定了进度的每一个细节？需要构建什么，或者工程团队有没有发言权，有多少发言权？“

询问工程和其他团队之间的关系。Doug Schade，WinterWyman公司软件技术搜索部门的合作伙伴和招聘人员，建议问“在应对项目时，你们公司会给开发人员什么级别的自主性？”Bonmassar说，对软件工程师的反馈缺乏任何机制是一个“危险信号”。

如何衡量我？

你的雇主如何定义你的“成功”与给你的工资和津贴等各种福利息息相关。但是，不同公司的评判标准不同，要满足你觉得不舒服的目标会让你的生活苦不堪言。

有些公司衡量软件工程师看的是他们的努力，比如他们工作了多少小时，提交了多少代码，Ari Weil，Yottaa公司的产品副总裁说（Yottaa是一家自适应的内容分发网络提供商）。也有的用结果来评估软件工程师，如因缺陷而需要召回的代码数量，或在规定时间和预算范围内，小组完成的项目数量。

有什么成长计划?

TonyaShtarkman，Riviera Partners的首席技术招聘人员说（Riviera Partners是一家总部位于旧金山的猎头公司），很多软件工程师觉得“他们在当前公司已经不可能有多大发展了。“她建议软件工程师在面试时要询问是否有一个针对软件工程师的成长计划——允许他们继续晋升，并且有机会让他们参加会议和研讨会来建立新的产品和功能，并受到辅导。

许多软件工程师希望雇主会告知他们最新、最好的技术工具，使他们能够保与时俱进。但Bonmassar警告说，“它通常是一个不好的兆头”，当公司坚持某个极其特殊的技能，并要求能迅速改变的时候，可能要不了多久该公司就会开始找人来代替你。如果说需要更匹配的长期合作，他说，那么可能这家公司现在需要的是“聪明，但不必知道工具和技术每一个细节的人”。

他还建议询问一下，多少外部聘请vs公司内部晋升。这答案能说明很多关于随着企业发展你的成长之路会怎么样的趋势。

你们的发展计划?

如果你正在考虑去创业公司工作，那么你需要了解他们的发展计划：“加入创业公司，总是涉及着一定程度的风险水平，然而创业公司的工程师往往比大企业的工程师不怕风险，“Shtarkman说。“不过，将风险控制在一定的稳定范围内是必需的。”

第一个步骤是调查。Shtarkman建议可以问这样的问题，如“你们的资金消耗率（公司的负现金流）是多少？”，以便于了解公司在没有其他资金和不盈利的情况下能维持多久。Jim Barnett，Glint公司的首席执行官（Glint是一个用于跟踪可以影响保留趋势的网络平台），建议在签署保密协议前可得仔细看清楚。

我会喜欢你们的人吗?

聊到目前的团队成员，“我碰到过一些工程师之所以接受创业公司的offer，纯粹是因为他们与团队融合得非常好——有时候甚至是因为某个人的魅力，”Shtarkman说。“说来说去，公司是由人组成的，如果你不能与你的队友和睦共处，那么当作长期的职业生涯几乎是不可能的。”

试着和公司的内部人士聊天，以便于知道“公司内部管理人员大致的情形，”Barnett说。“他们好合作吗，他们做事征求意见吗，他们提供反馈吗，他或她投资团队成员并帮助他们成长吗？”

试着和团队中你共事的人进行非正式的交谈。问问他们工作中最让他们沮丧的是什么。比起面试官，他们更可能现实地回答你，Shtarkman说。

底线：挖掘得更深一点以了解今后你每天需要共事的人，和你每天要经历的工作流程，而不要只关注薪水。

程序员面试技巧总结

闲聊

在深入代码之前，大多数面试官喜欢聊聊你的背景。

他们想知道：

你对编码认知。你是否知道如何编写好代码？

个人能力/领导力。你是否经历过整个工作流程？你是否修复过并不怎么正确的东西，即使你并不需要这么去做？

沟通。和你交流技术问题是有用的还是痛苦的？

你应该至少说明以下中的一个：

你曾解决的一个有趣的技术问题

你曾克服的一个人际冲突

显示领导力或个人能力的例子

你曾在以往项目中做出的贡献

最喜欢的语言的一些琐事，对这种语言你做了什么，以及你不喜欢它哪里

有关公司产品/业务的问题

关于该公司的工程策略（测试，Scrum，等等）

热爱技术。表达你对你所做的一切感到骄傲，你对自己的选择充满自信，你对语言和工作流有着自己的看法。

沟通

涉及到编码问题的时候，沟通是关键。一个在工作时需要帮助却能和人正确沟通的求职者比那些能轻松解决问题的求职者甚至更好。

了解这是哪种问题。

有两种类型的问题：

编码。面试官希望你能针对问题写出简洁高效的代码。

闲聊。面试官希望能和你聊一聊。话题通常是（1）高水平的系统设计（“如何克隆Twitter？”）或（2）琐事（“Javascript中的hoisting是什么意思？”）。有时候这些琐事中也会引入“实际”问题，例如，“如何迅速排序整数列？好的，如果不是整数，是其他类型的呢…… ”。

如果你开始编写代码，并且面试官并不想多说废话，只想尽快过渡到“实际”问题，那么如果你罗哩叭嗦太多的话，她可能会觉得厌烦。不妨直接问，“是不是为这个问题写代码？”

让人感觉你有团队精神。面试官想知道和你一起工作是什么感觉，会有什么问题，所以要让他们看到你的团队合作性。使用“我们”来代替“我”，例如，“如果那个时候我们做广度优先搜索的话，就能及时/准时得到解决方案。”如果让你选择在纸上还是在白板上编码的话，选白板。这样，你就可以接近面试官，直接面对他提出的问题（而不是和她在桌子两边遥遥相望）。

把自己的想法大声说出来。不是开玩笑，比如说：“我不知道这样做是否有效——但请让我试一试。”如果你不知道怎么办，不知道这个问题该如何解决，那么就说一说你现在的想法。说一说你认为怎么做可能会有效。说一说你认为哪些会有用，以及为什么没用的原因。这同样适用于琐碎的闲聊问题。当面试官要求你解释Javascript闭包的时候，“这与范围有关，不妨把它放到一个函数中”可能会让你得到90％的分数。

不知为不知。如果正在谈论的话题（例如，具体的语言事务，具体的琐事，运行时分析）的确是你不曾涉猎的内容，那么不要不懂装懂。相反，你可以直接说：“我不知道，但我猜$thing，因为……”，因为后面可以通过分析排除其他选项，还可以拿其他语言或问题做例子。

说话不要不经大脑。不要自信地将答案脱口而出。如果是正确的，那么你还是需要时间来考虑如何解释，如果是错的，那会显得你冲动鲁莽。你不是在和人比速度，而且你这么做更有可能因为打断她的话或者妄下结论而惹恼她。

摆脱困境

有时候你会陷入僵局。放松。这并不意味着你已经失败了。请记住，面试官通常更在乎的，是你能否巧妙地从几个不同的角度去揭示问题，而不是一根筋走到底地坚持正确答案。

画图。不要浪费时间在脑袋里思考，可以画到板上。画出几个不同的测试输入。画出你如何手动如愿得到所需的输出。然后想想将你的方法转换成代码。

解决问题的简单版本。不知道如何找到集合中的第4大条目？那么想想如何找到第1大条目，然后试试能否沿用这种方法。

写一个简洁低效的解决方案，然后对其进行优化。竭尽全力。尽一切可能的方法得到某种答案。

讲讲自己的思路。讲一讲你知道什么。讲一讲你认为什么可能工作以及为什么无效的原因。你可能突然会意识到它实际上是可以工作的，或修改版本是有效的。也有可能，你会得到提示。

等待提示。不要用期待的眼光盯着面试官，但可以有短暂的“思考”时间——面试官或许已经决定给你个提示也说不定呢，等待她的提示以免打断她。

考虑空间和运行时的界限。如果你不知道你是否可以优化解决方案，那么就说出来。

例如：

“我必须至少看看所有的条目，我做不到时间复杂度比O(n)还好的了。”

“蛮力方法才能检验所有的可能性。”

“答案将包含n^2数据项，所以我必须至少花费N^2的时间。”

写下你的思路想法

凭空地想很容易自我矛盾。把你的想法写下来，然后再去考虑细节。

调用帮助函数，继续前进。如果你不能或多或少地马上想出如何实现算法，那就跳过它。写一个命名合理的调用函数，例如：“this will do X”，然后继续下一步骤。如果帮助函数非常微不足道，你甚至可以将它忽略。

不要担心语法。不妨一笑而过。如果你非要考虑语法，那就还原到英语。只要向面试官说明稍后会回来整理即可。

预备足够的空间。你可能后面会想要在代码行之间添加代码或笔记。从白板的顶部开始写，并在每一行之间留一条空白。

最后写一个重头检查的标志。不要担心你写的for循环是否应该有“<”或“<=”。在代码的最后画个勾选提醒自己最后再检查一遍。先按自己的思路走。

使用描述性的变量名。想名字需要时间，但可以防止你忘记自己写某段代码的目的。使用namesto_phone_nums_map而不是nums。在名称中说明类型。返回布尔值的函数应该以“is *”，保存列表的Vars应该以“s”结尾。标准化很有意义。

完成之后的整理

浏览解决方案，大声地讲，输入一个例子。当程序运行时记录下变量保存的值——如果你只是记在脑子里，不会让你赢得任何加分。这有助于你发现bug和消除面试官的困惑。

寻找差一错误。你的for循环是不是应该使用“<=”来代替“<”?

测试边缘情况。措施包括空集合，单项目集合或负数。加分点：提一提单元测试!

不要惹人厌烦。有的面试官可能并不在意这些整理步骤。如果你不确定，可以这样说，“我通常会检测一些边缘情况——那么我们接下来是不是做这个呢？“

实践

最后，运行实践问题是没有捷径的。

好记性不如烂笔头。对自己诚实。用笔写可能一开始会让你觉得别扭。但是如果你现在就能克服这个难题，那么当面试的时候，你就不会觉得笨拙和不顺手了。

本文中的实践问题只是提供了每个面试过程的线索要点，没有真正的金科玉律，在真正面试时还需实际问题实际解决。**最后，祝大家面试成功。**

俞校长您好：

我先对照一下新东方最新的招聘要求：

1、有很强的英语水平，英语发音标准

英语水平还好，发音非常标准，我得承认比王强老师的发音差一点。很多发音恐怖的人(宋昊、陈圣元之流)也可以是新东方的品牌教师，我不知道为什么要要求这一条，尽管我没这方面的问题。

2、大学本科或以上学历，英语专业者优先

真不喜欢这么势利的条件，这本来应该是实力、马力之流的学校的要求。

3、有过考TOEFL、GRE的经验

GRE考过两次。

4、有教学经验者，尤其是教过以上科目者优先

教过后来被国家明令禁止的传销课，半年。

5、口齿伶俐，中文表达能力强，普通话标准

岂止伶俐，简直凌厉，普通话十分标准，除了对卷舌音不太在意(如果在意，平舌音也会发错，所以两害相衡取其轻)。

6、具备较强的幽默感，上课能生动活泼

我会让他们开心。

7、具备较强的人生和科学知识，上课能旁征博引

除了陈圣元，我在新东方上过课的老师(张旭、王毅峰、王昆嵩)都和文盲差不多，当然他们还小。说到底，陈圣元的全部知识也只是在于让人看不出他没有知识而已。

8、具备现代思想和鼓动能力，能引导学员为前途奋斗

新东方的学员是最合作，最容易被鼓动的，因为他们来上课的最大目的就是接受鼓动，这个没有问题。

9、年龄在40岁以下

28岁。

下面是我的简历或是自述：

罗永浩，男，1972年生于吉林省和龙县龙门公社。

在吉林省延吉市读初中时，因为生性狷介,很早就放弃了一些当时我讨厌的主课,比如代数、化学、英文,后来只好靠走关系才进了当地最好的一所高中,这也是我刚正不阿的三十来年里比较罕见的一个污点。因为我和我国教育制度格格不入又不肯妥协,1989年高中二年级的时候就主动退学了。有时候我想其实我远比那些浑浑噩噩地从小学读到硕士博士的人更渴望高等教育,我们都知道钱钟书进清华的时候数学是零分(后来经证实其实是15分),卢冀野入东南大学的时候也是数学零分，臧克家去山东国立青岛大学的时候也是差不多的情况。今天的大学校长们有这样的胸襟吗？当然,发现自己文章写的不如钱钟书是多年后的事情了,还好终于发现了。

退学之后基本上我一直都是自我教育(当然我的自我教育远早于退学之前),主要是借助书籍。因为家境还勉勉强强,我得以相对从容地读了几年书,"独与天地精神往来"。

基于"知识分子要活得有尊严,就得有点钱"这样的认识(其实主要是因为书价越来越贵),我从1990年至1994年先后筛过沙子,摆过旧书摊,代理过批发市场招商,走私过汽车,做过期货,还以短期旅游身份去韩国销售过中国壮阳药及其他补品。令人难堪的是做过的所有这些都没有让我"有点钱",实际上,和共同挣扎过的大部分朋友们比起来,我还要庆幸

我至少没有赔钱。

我渐渐意识到我也许不适合经商,对一个以知识分子自许的人来说,这并不是很难接受的事情,除非这同时意味着我将注定贫穷。

1994年夏天,我找了个天津中韩合资企业的工作并被派去韩国学习不锈钢金属点焊技术,1995年夏天回国的时候,很不幸我姐姐也转到了这家天津的公司并担任了副总经理,为了避嫌我只好另谋出路。

1995年8月至1996年初,经一位做传销公司(上海雅婷)的老同学力邀,我讲了半年左右的传销课，深受广大学员爱戴。遗憾的是国家对这种有争议的商业形式采取的不是整顿而是取缔的政策,所以看到形势不对,我们就在强制命令下达之前主动结束了生意。因为那时候我爱上了西方音乐(古典以外的所有形式),大概收有上千张英文唱片,为了听懂他们在唱些什么，我在讲传销课的同时开始学习一度深恶痛绝的英文。我在一个本地的三流私立英语学校上了三个月的基础英语课，后来因为他们巧立名目拒付曾经答应给我的奖金(我去法院起诉过，又被法院硬立名目拒绝受理)，我只好又自学了。

实在不知道困在一个小地方可以做些什么,所以1996年夏天我到天津安顿下来(那时候我很喜欢北京，但是北京房价太丧心病狂了),靠给东北的朋友发些电脑散件以及后来零星翻译一些机械设备的英文技术文章维生,因为生性懒散不觉蹉跎至今。我要感谢那本莫名其妙的预言书"诸世纪",尽管我不是一个迷信的人,但是去年五一我看到那段著名的预言"1999年7月,恐怖的大王将从天而降、、、、、、"的时候还是有些犹豫,我认真地考虑自己可能即将结束的生命里有什么未了的心愿,结果发现只有减肥。从我有记忆以来我就是个痛苦的胖子,因为胖,我甚至不得不隐藏我性格里比较敏感忧郁的一面,因为胖子通常被大众潜意识里不由分说地认为应该嘻嘻哈哈,应该性情开朗,应该徐小平。他们对一个矫矫不群的胖子的性格能够容忍的上限是严肃,再出格一点就不行了，比如忧郁。虽然他们从来不能如此准确地说出这种想法,但是如果看到一个忧郁的胖子,他们就会直觉哪里不对了,他们的这种直觉的本质是，"你是个胖子,你凭什么忧郁呢?你还想怎么样?你已经是个胖子了。"所以很难见到一个肥胖的并且影响广泛的诗人,因为公众不能接受,任凭他的诗歌惨绿无比。当然胖子的痛苦永远不值得同情(除非是因为病理或基因导致),因为他们胖通常是因为缺乏坚强的意志(也许除了丘吉尔)。我就是个典型,我的肥胖完全是因为厌恶运动造成的,我有过十几次失败的减肥经历,我试过节食、锻炼、气功和几乎所有流行过的药物,包括在西方严禁非处方使用的芬弗拉明,我总怀疑我不如小时候开朗是因为误用芬弗拉明造成的,它减肥的药理竟然是通过使人情绪低落从而降低食欲,事实上,它根本就不是研制用来减肥的,它本是用来使轻度狂躁型精神病患者稳定情绪的药。我是中国落后的药检制度的严重受害者。

过了去年的五一节之后,我制定了严格的计划：每天只吃蔬菜、豆腐、全麦面包、鱼肉、橙汁、脱脂牛奶和善存,每天用一个小时跑10公里,也就是标准跑道的25圈。我不得不骄傲的是，我只用了58天就减掉了48斤体重,去掉休息的星期天,几乎是一天一斤。然后我心情平静地迎接了什么事情都没发生的7月。这件事过后我发现其实我还是很有毅力的一个人。但是我不知道我的毅力应该用来做什么,末日虽然没有来,但是新世纪来了,30岁也快来了,这真是一件让人坐立不安的事情。

后来我一度想移民加拿大，所以一边找资料看一边到天津大学夜间开办的口语学习班上课，一个班20多个人，一个外国教师(更多的时候是外国留学生)和我们天南地北地胡聊，除了政治。我一共上了四期这样的班，口语就差不多了，当然还是停留在比较普通的交流水平上，至少我看英文电影时还是需要看字幕，尽管在天津的四年间我看过大概600部英文电影。过了元旦，一个小朋友在和我吃饭的时候突然问我，为什么不去新东方教书,你应该很适合去新东方教书。我说我倒是喜欢讲课,但是一个民办教师有什么前途呢?他说如果年薪百万左右的工作不算前途那他就没什么可说的了。我得说我很吃惊。不管怎么样，我仔细地把我能找到的关于新东方的材料都看了一遍,我觉得这个工作很适合我，尤其是看到杨继老师在网页上说"做一个自由而又敬业的人是我的梦想,新东方是实现它的好地方"的时候。在我尽管懒散无为却又是勤于思考的三十来年里，好像还是第一次看到一个很适合我并且我也有兴趣去做的工作。杨继还转述席勒的话"忠于你年轻时的梦想"。我没看过席勒的东西，光知道有两个能写字儿的席勒,不知道是哪一个说的这话，但是我宁愿把它当成是新东方的精神。我听说教托福和教GRE薪水差不多，但是GRE的学习要苦得多。

我想了想还是选择了GRE，毕竟托福是专门给非英语国家的学生考的，教书的满足感上逊色很多。

旧历新年的时候，因为不确定是不是需要大学文凭才行，我试着写了一封应聘信给俞老师，提到我只有高中文凭，结果得到的答复是欢迎来面试，除了感激我还能说什么呢?我是说即便没有文凭不行我还是会来新东方做教师的，但是可能不得不伪造证件，作为一个比大多数人都更有原则、以知识分子自诩的人，如果可能，我还是希望不搞这些虚假的东西，俞校长的开明使得我不必去做大违我的本性和原则的事情，得以保持了人格的完整，这是我时常感念的。

过了春节处理了一些杂事，很快就到了6月份，我买了本"红宝书"就上山了。鹫峰山上的学习气氛和恶劣条件我都非常喜欢，应该是因为生活有了明确目标的关系吧。但是我很快发现，讲课教师的水平和他们的报酬以及新东方的声誉比起来还是很不理想的。我看到身边大多数的同学对所有的老师评价都很好，听到那些愚蠢的笑话、对ETS肤浅的分析导致的轻浮谩骂和充满种族歧视、宗教歧视的言论的时候，大多数人都笑得很开心。这最终再次有力地证实了我一直怀有的一个看法：任何一个相对优秀的群体里面都是笨蛋居多。无论台下是300名来听传销的社会闲散人员还是300名来听GRE的大学毕业生，对于一个讲课的人来说并没有多少区别，这也是他们在台上信口开河、吹牛放炮的信心来源。当然这里大多数同学专业都很出色都很勤奋刻苦，积极上进，性格上也远比我更具备成功的素质，我只是说他们缺少情趣，他们聪明(至少他们都敢考GRE的数学，这是我想都不敢想的)，但是没有灵气，人品也未必差，只是缺乏独立思考能力。

我只喜欢陈圣元一个人的课，所以后来也就只去上他一个人的课，其他的时候一个人在宿舍背单词。陈圣元除了胡扯闲聊比较有水准之外，治学态度曾经也让我觉得很好，说起charter这个单词的时候，他说为了找到那个填空句子里面表达的意思查遍了所有的词典都找不到满意的解释，最后花了一千多块钱买了一本巨大沉重的韦氏词典(显然是指MerriamWebsters Third New International Dictionary Unabridged)才终于在该词典所列的关于charter的25条释义中的最后一条里找到了答案。说起市面上粗制滥造的填空参考书的时候他很不以为然，"我以三年的教学经验也精心编写了一本，那些作者对题目绝对没有我钻研得深，他们就会胡编乱造然后急忙出版抓紧骗钱，我这本可以说是这方面的集大成者，现在正在印刷当中，很快就可以和大家见面"。由于在山上的时候单词还没怎么背，题目都没做过，所以他这些态度和表现曾经让我很景仰。发现不对头是下山之后开始的，我录了他的全部课堂录音，我听着录音大量做题的时候，才发现他的分析讲解漏洞百出，尽管他批评过去的新东方老师都是拿了正确答案再进行分析讲解，可是他的工作显然也是一样，这样才能解释为什么他总是能用错误的分析推理给你一个正确的答案。另外我发现所有的三流词典，包括英汉词典，都在charter的第一个释义上就解释了他声称在韦氏第三版未删节新国际词典的25条释义的最后一条中找到的答案，"由君主或立法机关发给城市或大学，规定其特权及宗旨的特许状"，所以我也买了本十多斤重的韦氏第三版回来，发现只有13条释义，而且在第2条里就解释了这个问题。现在他的那本填空教程就在我手边，仅在No、4的52道题中，我就找到了18处错误，如果说翻译的错误对学生不重要，那么解题分析的错误也有10处之多。这也最终使得我改了主意，决定做填空老师，本来我想做词汇老师，那样可以海阔天空地胡扯。

我以这样的条件敢来新东方应聘，除了脸皮厚这个最显而易见的表面原因之外，主要还是教填空课的自信。第二次考试之后我一直做填空的备课，最消耗时间的是把NO、4到1994年的全部填空题翻译成中文，400多个句子的翻译居然用了我整整一个月的时间，基本上是一个小时翻译三个句子，当然快的时候两分钟一个，慢的时候几个小时翻不好一句。翻译这些句子是我本来的备课计划之外的工作，最终使我不得不做这个工作的原因是钱坤强和陈圣元那两本"惨不忍睹"的教材。钱坤强的那本就不必说了，此人中文都有问题，尽管我坚信他的英文要远比我的水平高(也许应该说熟练)，但是理论上一个人如果母语都掌握不好(这意味着他对语言本身不敏感)，那么他肯定掌握不好任何其他语言，即便他能熟练运用，也不适合做语言方面的工作，比如文字翻译。他那本超级填空教程在新东方地下室卖了两年都没正式出版，一定程度上说明了这本书的水准。至于我非常喜欢的陈圣元，他在那本书的前言中说道："翻译时尽量体现原文的结构，以便考生能对照原文体会原文句子结构的特征，从而体会结构与答案选项设计之间的关系……这样做会使得句子略显得欧化而不自然……不会去套用貌似华丽实则似是而非的成语。"

作为一个考试学习用的教材，他声称的翻译原则和宗旨是很好的，但是很遗憾，我看到的绝不仅仅是一个欧化或是不自然的问题。首先"体现原文结构"应该表现为翻译成中文之后原文中的各个句子成分最大限度地在译文中充当同样的成分，而不是把所有的成分不变动位置地翻译成对应的中文单词，这样的做法和那些《金山快译》之类的拙劣软件的翻译结果有什么区别呢？实际上《金山快译》这一类翻译软件的译文虽然狗屁不通，但是我们即使看不到原文，通过猜测也能大致明白它想说些什么，这就如同没学过日文的中国人看日文电器说明书里面夹杂的汉字也能隐约猜出大意一样。陈圣元的译文本质上就是这样的一种东西，当然程度上有些区别。

其实欧化并不可怕，尤其是在一本学习用的教材中，即使是在文学中，一些恶性的欧化今天也成了现代白话文的组成部分。陈圣元的译文根本不是欧化的问题，他的译文和钱坤强的一样，最恐怖也让人难以置信的是，作为所谓的译文，如果脱离了原文的对照，没有一个中国人知道这些句子在说什么，我是说这些句子字面上在说什么都没人看得懂，而不是因为句意晦涩难解而使人看不懂它想表达的内涵。

另外，看得懂的很多句子又翻错了，即便看不懂句子的意思真的不影响正确答案的选择，但是作为一本教学参考书，如果参考译文都翻错了，还怎么让学生信服呢？除非是以一种变态的方式，比如"陈老师的译文都翻错了，可是他的GRE考分那么高，可见看不懂句子对答题反而有帮助"这一种。

我的译文体现了这样几个原则，首先由于不是文学翻译，我注意了最大限度地使用原文的结构，使译文中的句子成分尽量充当原文中的对应成分。为了这种对应，有时候会有些比较不符合中文习惯的句子结构，比如一些在英文中可以置后的定语从句按照中文的语法放到了修饰对象的前面之后，句子显得臃肿不堪，另外还可能导致断句困难。针对这样的问题，我在这类句子中大量地使用了括号和破折号，很多时候，如果只读括号外边的内容，读到的就是这个句子完整的主干，那些使句子结构变得复杂的修饰成分都在括号里边，但是如果假定这些括号不存在把它们连起来读，也是一个通顺完整毫无语病的句子，这样和原文对照阅读时对应的成分和原句的主干结构清晰可辨。

在解题思路上我修正了陈圣元的书中所有不严谨的地方，难以置信的是这些不严谨的错误在他的书中竟有三成之多。我的草稿还有很多优点，虽然这些优点是我完成的，但是我不想为了向别人解释"我做的工作牛就牛在、、、、、、"这样的东西浪费太多的时间，所以不再分析了。如果我们都接受"不存在完美的东西"这样一个假设，那么我想说的是，我的这本填空教材是离完美最近的那一个。希望我的坦率不会倒了您的胃口，当然我知道新东方的开明气度才会这样讲话。

如果新东方出版参考书的惟一标准是书的质量，而不权衡其他方面的因素，那么陈圣元的那本书的寿命不会也不应该超过一年。需要做一个效果未必理想的声明是，我并非有意攻击陈圣元，他在课上说起，新东方的同仁们的一个优点是互相不会拆台，也许私下并无深交但是不会互相诋毁，这对事业或是人生的成功起到了相对积极的作用。这种观点虽然不合我的本性，但是我也知道如果大家都是书生意气，新东方也没有今天。所以我接受了他的这个看法，基于这一点，我也不想对他做更多的攻击，很大程度上我对他的看法现在都坦率地说了出来是因为他已经离开了，不存在和睦相处的问题。何况，他出色的幽默感和极佳的亲和力都是我很佩服的，毕竟他是新东方我见过的最喜欢的老师(如果不是惟一喜欢的)。对于他的工作和治学态度，我更多的是感到遗憾。

当然我知道会有一些年轻教师不屑地说，教教GRE，算个屁自学？那好吧。

我想我多半看起来像是个怪物，高中毕业，不敢考数学，居然要来做教师。但是我到新东方应聘不是来做教师的，我是来做优秀教师的，所以不适合以常理判断。即使新东方的声誉和报酬使得它从来都不缺教师，我也知道优秀的教师永远都是不嫌多的，如果新东方从来都不缺优秀教师，那么我也知道更优秀的教师从来都是新东方迫切需要的。

龚自珍劝天公"不拘一格降人才"，如果"不拘一格"的结果是降下了各方面发展严重失衡的，虽然远不是全面但又是十分优秀的畸形人才，谁来劝新东方"不拘一格用人才"呢？想想王强老师的经历，所以我也来试试说服您，我们都知道那个美国老头虽然觉得他很荒唐，但是他还是给了王强老师一个机会去见他，一个机会去说服他，所以我想我需要的也就是这么个机会而已。给我个机会去面试或是试讲吧，我会是新东方最好的老师，最差的情况下也会是"之一"。

本文作者是 Zach Mathew，他是 FreshBooks 的经理。

我面试过很多人，大部分是开发者，部分是产品经理，有时候会面试主管或者副总监。但不管是面试什么级别和什么工种的应聘者，我都会在过程中对他们提出一个相同的要求： 现在，请把我当成一个学生，随便教我点什么东西和知识吧。

什么都行。

可能是什么东西你觉得有意思的，或者你自己在某方面研究比较深的领域。甚至是你最近刚刚学习到的东西，反正是什么都好。你不需要是那方面的专家，但至少能跟我讲明白讲清楚，而且你能够回答我一些基础的问题。

对，现在给你十分钟的时间，把你脑海里想到的东西教给我。

我之所以对面试者提出这个要求，是因为我想知道我能从这个将来的同事身上学习到什么。我也想知道你的团队未来会从你身上学习到什么。

当然，我问这个问题的时候，也想知道你的气质符合不符合我们公司的文化。虽然说 FreshBooks 这个公司并没有具体的规则，但其实每天，无论是实习生或者是管理层，我都会问他们类似问题，而且希望他们能给我满意的回答。

以下是我不久前问自己同事的问题，并从中学习到的事情。

- 我问 Tobi, 他是我团队里的一名开发: 我看到你在代码中正在用 ES6， 你认为它用起来怎么样?
- 我问 Marcus, 他是金融公司的一名分析师: 跟我解释下同期群分析是什么意思？我应该在未来使用这个方法吗？

有的时候，一些初级开发者会问我：我知道的东西，你肯定早就知道了。我没法教你。

不，还是请指点我一下。实际上，当你真正教我的时候，你会吃惊于我多么无知。 而且就算你讲的东西是我早就知道的事情，再听一遍也不是什么大事。

毕竟在那么多次的面试里，肯定会有人告诉我一些我早就知道的东西。那真的无所谓，我真正想从面试里了解到的是：

1. 你是否能和他人进行有效交流？
2. 你能否在研究一件事情的时候透过表面深入内层？
3. 你是否具有有条理阐述一件事情的能力，还是说你是对什么都很不耐烦沟通的人？

会学习是一种能力，能把自己学习到的东西表达给别人也是一种能力。

这不仅仅是为了面试，我的意图是考察你其他的技能和潜能。

在公司内部，我们也经常举办这种「教我点什么」的大会。通常是在周五下午，喝点小酒，大家聚在一起，分享彼此之间从本周工作以及最近的项目中得到的灵感。基本上，这就是一个信息的共享大会。不需要准备 ppt 或者演讲稿什么的，只需要张开嘴，放开心扉，告诉别人你的所得。

你也可以反过来，邀请我教你学习点什么。 这当然可以，如果我要求你教我点什么，你也可以对我提出相同要求。面试是一个双方过程，在我评测你的时候，你也可以评价我。

所以拜托，当我要求你教我点什么东西的时候，你也可以对我提出相同的请求。

上文来源：Medium

我个人感觉，这种「给你十分钟，你随便教我什么东西」的面试思路非常有意思。那身为应聘者，我们该怎么运用这十分钟给面试官留下好印象呢，对爱丽丝来说，其实觉得把这整个场景视为一个小小的自检是否具有有效沟通的过程比较有意思。

我把自己的面试思路分享一下。

首先，你要对面试时间有个预估。对方说给你十分钟，这时间说长不长说短不短。你可以迅速规划一下，比如我先用五分钟的时间讲述我想说东西的具体概念、理论知识和背景感悟。然后，剩下五分钟，可以用来让面试官继续提问或者自己继续补充。切不可自己光说。

其次，你也可以把自己的打算和规划告诉面试官，比如在我最初前几分钟内向你讲述背景知识的时候，请先听我说完，尽量不要打断我的思路。

接着，当你在真正介绍一件事、或传授一个知识的时候，记住一定要有条理，说话慢一些。当回答对方的问题时，多问问对方「我讲清楚没有」，而不是要问对方「你听明白没有」。

最后，面试官都说了他其实并不在乎你讲什么。这是什么意思？这就是说你跟他说怎么烙煎饼和怎么解释引力波，人家根本不在乎。其实主要考察你的交流水平，其实也就是逻辑。

而逻辑简单来说，就是大到小，从浅及深，想清楚影响的因果联系。当对方提出你不懂的问题，可以把他的问题拆解出几个小问题，去解答你懂得的地方。而对于不懂的事情，也不要不懂装懂，反而可以咨询他的意见——相信我，既然他能问出你不懂的问题，就说明他的水平比你高。

让他帮你解释。当面试官比你话多的时候，这绝对是很好的信号。

经验之谈

- 一个程序员的血泪史
- 我为什么离开锤子科技?
- 扫清Android面试障碍
- 史上最全 Android 面试资料集合
- 2016年4月某公司面试题及面试流程
- 2017届实习生招聘面经
- 国内一线互联网公司内部面试题库
- 互联网公司面试经验总结
- 一个五年Android开发者百度、阿里、聚美、映客的面试心经
- 腾讯公司程序员面试题及答案详解
- 面试心得与总结：BAT、网易、蘑菇街
- 阿里+百度+CVTE面经合集
- 技术硬碰硬—阳哥带你玩转上海Android招聘市场

> 原文链接：微信公众号[小李成长笔记]

到 9 月中旬，我在华为工作就满 6 年了。

同事听说我要离职，都很诧异。干的好好的啊，怎么突然说要走？

是啊，在现在的部门这么久了，人和事都很熟了，偶然有些小摩擦，都可以理解和接受。那为什么要走？

华为的 34，胶片文化，强绩效主义，部门墙，研发的低效等等这些，大家说过很多，就不再重复了。在我看来，这些都是微不足道的外因，我完全不认同，但可以理解。

真正推动我下定决心离开的还是内因：我那颗躁动不安的心。

在华为 3 年后，工作真的变成了工作。我常常想不起来昨天都做了些什么，只是模糊记得接了几个电话，帮助客户解决了几个问题，接入了几个莫名其妙的会议，回复了几个邮件。就这样一周过去了，一个月过去了，一年也过去了...

职级在升，工资在涨，年终奖也一年比一年多。可我却越来越心虚，越来越焦虑。

有 2 个我一直在争辩。一个我：现实点，这就是份工作，出卖时间，换取工资而已。干嘛期望那么多？另一个我：还这么年轻，就认命了吗。想想这些年你都干了些啥。每天处理些 stupid 问题，一年年重复自己，你还想像这样过多少年？

2 个我不停的争吵，日子在争吵中慢慢过去。

我知道不能再这样下去了，我要行动起来。我要做准备，离开这个安逸，不再成长的环境。

学习 Andoid 开发，重拾开发乐趣

16 年 Android 很火。我想学 Android 开发，买了《第一行代码》，边看书边搭环境，敲代码。买 vpn，用 google 搜索问题答案。关注了张哥的公众号「stormzhang」，知道了 stackoverflow 网站，github 网站。知道了开源软件。慢慢摸索，年底时也能做个还能用的 app 了。上传到应用商店，看到后台统计数据，每天有上百人下载，也有一些用户评论说不错，很喜欢。感觉很充实，很有成就感。

走出去，接触外面的世界

在华为的日子，每天按部就班工作，大家都很勤奋，专注。只需要埋头干活就可以了。慢慢的，好像和外面的世界隔离了。

当我第一次参加一个武汉产品经理交流活动，看到不大的会议室里坐满了上百名自发参加的朋友，被大家渴望知识和交流的态度感动，原来，我们身边还有这样一群人。

也参加过几次程序员自发组织的编程活动，大家一起结对编程，分享各自的工作流，推荐好用的工具和软件。能感受到他们对自己工作的热爱，让我羡慕感动。大周末的，放弃陪老婆孩子的休息时间，穿越大半个城市，和一群素未谋面的伙伴编程，绝对的真爱。当然，我们也成了网上的好朋友，虽然平时不咋联系，但大家知道，我们都是一类人。

关注互联网大 V，了解另一个世界

这期间关注了好多互联网行业大v的公众号。有耿直风趣的冯大辉，温和幽默的 MacTalk，黑白灰道都熟的 Caoz，硅谷 Airbnb 美女程序员 angena，Android 开发者帅比 stormzhang...每天看他们的文章，看他们写自己的从业感悟，写互联网行业的灰与黑，写曾经的迷茫和奋斗...真实有力，催人奋发。牛人都在奋斗，我还在温室里等死。

偶遇付费社群，见识网赚套路开眼界

转眼到了 17 年，好像一夜之间，知识付费就火了起来。

机缘巧合之下，加入了网友亦仁的生财有道小密圈，亲身经历了圈主 30 天收百万入圈费，见识圈主高超的运营手法，圈友大牛们分享的各种生财，吸粉套路。彻底改变了我对广告，对一些互联网业务的认知。

以往的我，对各种广告很不屑，认为都是骗人的。ppt 教程，excel 教程，这么 low 的东西，我可是 it 专家，哪看的上这些...

选择性无视，不看，不听，不想。

现在我会去想为什么会有这个，有多大市场，针对哪些人群，它们有哪些套路，有哪些值得学习的地方？这个事情有哪些门槛，如果我去做，可以吗。

脑袋瓜被慢慢的打开，不再是以前那个非黑即白，只有 0 和 1 的工科男死脑筋了。

还加入了刘大猫的财富移动城堡。了解到刘大猫，是看了他写的自传文，一个从高中开始就做网站，做淘宝客，做流量变现，27 岁就积累千万财富的大牛人。听他分享各种总结，经验和看法，思路清晰，态度诚恳。不满足现状，完全放弃还很赚钱的淘宝客生意，坚决转型公司和个人发展方向。让我汗颜，想到自己畏惧风险，待在舒适圈里不愿出来，无地自容。

还有好多社群，这些社群让我有了“近距离”观察和交流大牛的机会。他们就像存在你身边，就是你的一个学长或者朋友，你好像看着他们一步步成长为大牛，有时候会想，或许有一天，我也会牛起来。

开发微信小程序，体验流量的威力

网上和群友交流，不能光说不练，你总得有些拿的出手的东西。听再多的套路，经验不如亲自去实践下。

趁着小程序这股东风，自学开发了一个文字转语音的小工具。从 app 名的选择，简介，到搜索关键词，app 界面，功能都做了一些思考。开发上线后，收集用户反馈，和用户交流使用场景，慢慢的每天用户竟然有近 300 了，排名也一直上升到第一名。

有天看后台数据，访问量突然增加了 10 倍，当天新增用户 3.5 k。吓坏了，以为后台出统计故障了。原来是知晓程序公众号当天的文章推荐了。那天好多用户联系我，夸工具好用，确实帮他们解决了一些问题。好开心，体验了一把网红的感觉。

以后咋发展

写到这里，可能很多朋友会问，你写了这么多瞎折腾的经历，和你离职到底有啥关系呢，是要转行搞互联网产品开发吗？老实说，我也不肯定。

我只知道这些经历潜移默化改变了我，让我不再封闭，不再纠结。我庆幸这些经历让现在的我经过华为近 6 年的“摧残”后，心还火热，对未来还留有希望。

工作咋办呢，32 岁“高龄”了，还要去做程序员吗，还有公司要吗？没事，我觉得自己还年轻，如何做产品，还有很多可以学，我也相信，华为 6 年的经历，磨练，纠结，折腾，这些都是我未来宝贵的财富。

终于，我做了选择，不再纠结，重新出发。

> 来源：微信公众号「LJ 说」，id：「LjNotes」。

“你居然要从腾讯离职了！？”

这是身边朋友得知我要离开后的反应，似乎大家都难以理解这样的决定。

从行业环境来看，中国互联网正处于一派繁荣之境；从公司形势来看，也正要准备大刀阔斧地干一番大事业；从个人发展来看，自己在公司也会担任越来越重要的角色。

所有的环境都是好的，更加显得离职的决策不理智。

HR 系统弹窗给出最后的挽留：你确定要提交离职申请吗？

经过各种综合考虑后，我还是点了“确定”按钮，正式从工作了三年的腾讯离职。

一直有朋友问，在腾讯的工作感觉怎么样？

关于这个问题，从来没有好好思考过，觉得当局者迷，尽量做好手上工作就是了。

现在终于有时间梳理一番。

回想起这几年的经历，既有取得成就的喜悦，也有遭受挫折的失落，个中唏嘘，在离开之际，希望与你分享一二。

大公司之病

3 年前，我面试完，从腾讯出来，融入了深南大道熙熙攘攘的下班人群中。

在过天桥的时候，我特意拍了一张腾大的夜景，留作纪念，表示我终于要到腾讯上班了。

虽然还没正式通知，不过凭着面试反馈，我知道自己终究还是要进入这家梦寐已久的公司了。

3 年后，同样是腾讯大厦，我站的位置已经发生改变。

从外面的仰望变成了里面的远眺，心情体会也随之改变。

只要在大型企业工作过的人，都会被大公司病深深困扰着。

你厉害还是平台厉害

BAT 的光环是非常牛逼的，它意味着你进入国内任何一家互联网公司都畅通无阻，它意味着你可以对外分享自己的经验和心得，享受着他人崇拜的目光。

然而，在一家几万人的巨头企业，几乎每个人都是一个普通员工，毫无存在感可言，其中滋味就如人饮水，冷暖自知。

看《权利的游戏》，我在想，龙妈拥有三条喷火巨龙，为什么还要四处斡旋、拉帮结派，直接骑着三条龙到处喷火，不早就征服七大王国了吗。

直到有一幕场景，大龙“Drogo”在斗兽场，被很多小兵拿着矛乱刺，身受重伤，我才反应过来，不管龙妈和她的三条龙再厉害，始终赢不了训练有素的军队。

这就是大企业的一个缩影。

公司征战，并不需要一个能斗天斗地的英雄，而是需要一支能打仗的队伍。

尽管在招聘的时候，大公司往往会筛选出最厉害的一批人，但这并不代表着每个人都举足轻重。

事实上，不管你是清北名校毕业光环加持，还是二三本拼搏多年进入大平台，公司想要的结果其实都一样。

公司希望每个一线的员工坚守着自己一亩三分地，不需要你把控全局，不需要你战略思考，只要努力地当好螺丝钉。

每个人手上分到一小块工作，然后在未来的很长一段时间内，不断地重复着这个工作，成为这个小模块的“专家”。

负责个性化皮肤的，可能几年内都在钻研怎么把更多皮肤卖出去；写文案的，长年累月地追着微博热点写文章；做渠道运营的，风雨无阻地盯着各个渠道把自己的广告上线……

并不是说公司不重视个人创造力，恰恰相反，公司希望的是大家发挥创造力，把自己变成更可靠的螺丝钉，成为一个更靠谱的零部件。

全公司上下形成一股合力，用军队的方式赢得战争。

赢的方式，不是靠武艺高超的英雄，而是让所有人在统一指挥下，移动、格挡、举矛、刺杀，每个动作都如此简单，但千军万马在一起，就能击破对手。

这也就意味着铁打的营盘流水的兵，在大平台工作，一件事情成功之后，很容易让人觉得是因为自己牛逼，其实真正的原因是平台的力量。

同一件事情，放张三能做成，放李四也 OK。

在这里，你不是英雄，你是一个日夜训练着重复动作的小兵。更可悲的是，对于绝大部分人而言，公司压根没计划让你成为一个英雄或将军。

你只是千军万马中的一员，平台缺少了你，马上能找到一个人填补上去，而你一旦离开了平台，就会发现很难再复制以往的成功。

无尽的流程和制度

把大象放进冰箱一共要三步：打开冰箱门、放进大象、关上冰箱门。

但是在大公司，这个流程就远不止三步了。

你要给个报告写清楚把大象放到冰箱的意义和重要性，搞清楚哪种冰箱放哪种大象，把开门动作、放大象的路线描绘清楚，把关门的力度写出来，拿着完整的方案找项目经理去排期，直到有人力来把大象放进冰箱。

一个产品或功能，从无到有需要经历漫长的流程。一个大企业的员工，每天为制度所困。

某些产品一两年内都没有可感知的外观变化，例如微信，有人就会问这么多工作人员都在忙些什么呢。

其实工作人员都很忙，忙在了“流程”和“制度”上。

当你们是一个三五人的创业团队，大家就坐在一起，有事情吼一声就可以。比如说想要做一个功能改变，可能就是抬起头跟对面的开发说要怎么怎么改，半天之后就能在产品上看到了。线上反馈好，就保留，反馈不好就改回来，不过是几个小时的事情。

然而，对于一个巨型产品来说，所有人的 80% 精力并不是在做“正经的工作”。

每个产品经理电脑上都躺着十几份写好的需求文档，在等候着漫长的项目排期。等排期终于到了的时候，有的需求已经不再适用了，或者写它的产品经理已经走了，要是需求和人都还在的话，那就要谢天谢地，守得云开见明月，终于要上线了。

每个开发脑子里都存放着许多改进方案，很多可能就是改一行代码的事情，但是不能擅自改动，所有的改动都应该以产品需求为主，否则出了问题那就闯大祸了。

遇到跨团队、跨部门沟通，更加是考验人的忍耐力，找一个接口人要花上大半天。对方要么不回复，要么回一句“这不是我负责的”。好不容易对接上了，好家伙，群里面出现四五个接口人，每个人都得交待一遍来龙去脉。

除此之外，每个人身上还背负着各种会议、分享、周报月报，PPT 模板成了最受欢迎的文件。

能够安心写代码、写需求的时间，算下来也许还真的没有20%。但工作还是要完成的，于是就只好加班加班，成为了互联网行业的一大特色。

流程制度是一个好东西，也是一个坏东西。

好的地方在于保证企业这条大船高效率地运转，坏的地方在于牺牲了个人效率来满足集体的效率。

逃不过的修罗场

对于大企业工作的朋友，是万万不能问什么时候升职的。就像不能问魏忠贤魏公公什么时候生个小孩，这是要杀头的大罪。

越是受过高等教育的人，越是想着要改变世界。当初怀着远大的志向进入大公司，想要施展拳脚做一件不凡的事情。

但几年后，大多数人的志气早已被磨消。修身齐家治国平天下，他们连修身都做不好。

眼看着身边的朋友在中小公司鹤立鸡群，一路扶摇直上，成为有决策权的管理者，而自己只是数年如一日地坐在小隔间，每周想着如何跟老板汇报工作。

也许离 CEO 的办公室只有十米不到的距离，也许每天还能跟几个高管寒暄一下，似乎离他们好近，但是心里明白这种阶层的差距是一道无法逾越的鸿沟。

HR 在设计个人发展体系的时候，给每个人都提供了两种路径，一个是专业能力晋级，一个是管理职能晋升。

所有人都能在专业能力通道上一步步地打怪升级，最终成为高级产品经理、高级工程师，甚至专家 xx。

但是在管理通道上，坑位就那么几个，而且大企业内具有管理头衔的人流动性远远低于普通员工，于是国企中“一个萝卜一个坑”的现象在创新的互联网企业同样存在。

不少人已经工作十多年，但仍然是一线普通员工。

至于谁能晋升，这个话题，不说也罢。

要是运气不好，赶上了“宫廷大戏”，轻则工作上举步维艰，重则随着失败一方的领导一起离开。

这里就是一个几万人的修罗场，陷于其中的人，个个都身不由己。

外面的世界很精彩?

前面说了那么多，你可能会以为我在痛陈大公司的弊端，但这并非我本意。

我并没有在真正意义上的创业团队工作过，但也有过小团队的经历，加上平时和不少创业团队的朋友交流，对小公司的辛酸也是略知一二。总结起来，不过是几个字：人少事杂、管理混乱、野蛮生长。

人少事杂

在小团队，可能出现最多的头衔是“全栈 xx”，这并不是说明他有多厉害，而是在一个人手不足的团队中，每个人可能都身兼数职。

写前端页面的，可能没人把写好的接口交给你，而是需要自己写服务器脚本、自己调优数据库，还得自己盯着运维数据，宕机了得马上修复。

做产品的，不是只打开 word 来写需求文档，用户调研、交互图得自己做，上线后的运营还得自己跟。

做运营的，更加是无所不包，大到策划一个线上活动，小到做客服回答用户的咨询。

每个人也很忙，似乎什么都能做。

这也是小团队吸引人的噱头，能对付过来的人，就成为一个真正的“全栈”，疲于奔命的人，就什么都做什么都学不精。

管理混乱

小团队是否意味着效率高呢，其实也可能存在更加胡乱的管理。

曾听朋友讲过他的经历。一个普通员工，需要同时向两个领导汇报，而两个领导还经常互掐，于是该朋友就一脸懵逼了，经常接收到两个完全相反的指令。

还有可能，前一周刚刚跟另外一个团队开完会，达成决定做个方案，下周再找他们就发现整个团队被老板裁撤了。

又或者，团队在短时间内爆发性增长，为了融资，为了数字，找来了一批新人，大家都面面相觑，不知道谁该做什么，本来公司也并不是因为业务需要而招新人，所以干脆大家都逛淘宝、刷微博。

如果说大公司内部的身不由己还有章可循，小公司的变化就是充满着惊喜。

野蛮生长

大公司令人艳羡的地方就是有很多现成的基础服务，而在小团队干活的人都会非常痛苦，大多时候都需要自己造轮子。

CDN 网络需要自己搭建，大数据平台需要自己开发，账号体系需要自己建设，支付系统需要从零开始……

一个最简单的例子，大公司有成熟的数据体系，每个可以看各种各样的报表，以便调整运营策略，但是小团队可能看个数据就需要提导数据的需求，等到一两周之后才能看到。

从一片荒芜中把业务从零开始做起来，是一件很锻炼人的事情，但其实背后更多的是资源浪费。

都是围城

万物皆有裂痕，要看到裂痕中照进来的光，而不是裂痕本身。

一位长者曾说过，一个人的命运啊，当然要靠自我奋斗，但也要考虑到历史进程。

历史的进程和外部的环境，都是我们所不能控制的，但是说到自我奋斗的话，大公司却提供了温暖的襁褓。

在正规化中成长

流程制度的反面就是“正规化”。

当有人问我要不要去大公司的时候，我都会回答，如果有合适的机会就去吧。不为别的，就为了体验这种正规化的流程制度。

管理两三个人的时候可以靠命令，管理二三十个人的时候可以凭个人魅力，管理几百人、上万人就只能依赖于流程制度了。

前面说过流程是牺牲个人效率满足集体效率，从个人成长而言，依然能从中学习到受益终身的东西。

你可以知道一个业务从零到一是怎么搭建团队的，各个团队通过什么样的流程进行配合，各司其职代表着每个环节都能产出精品，于是你就知道一个优秀的作品应该是怎么样的，以后碰到类似的场景就有经验了。

总而言之，身处在“正规军”当中，虽然自己只是其中的普通一员，但是也可以耳濡目染地学习到最顶尖的产品是如何打造出来的。

当然，前提是你有心去了解和学习。

站在巨人肩膀上

大公司汇聚了最优秀的资源，包括人才、技术、资金、经验等。

在平时，如果你想了解或钻研某一事物，往往能在内网上找到独家优质的经验分享。

更进一步，可以直接联系这个领域的高手咨询请教。内部使用的软件，上至 CEO 下至电脑维修小哥，都静静地躺在好友列表里，等着你联系。

在大公司工作的人，是真正的能做到聚焦于业务逻辑本身，而不用被琐碎的杂事打扰。

行政上，公司配备了饭堂、班车、体检、节假日福利、家人福利等等，让你能安心地工作。

业务上，有专门的基础服务部门，IT设备、开发组件、大数据平台、安全防御、用户数据等等，都可以拿来马上用。

个人成长上，虽然不是每个人都能平步青云，但是完善的薪酬福利让每个人都能获得相对合理的回报，各种培训让大家都能适当跳出舒适区获得成长。

这些基础服务都是十多年无数人的心血积累，可想而知，站在这样一个巨人的肩膀上，新人可以获得更高速的成长。

围城里外的人

前段时间见到一位大学好友，在外闯荡多年，辗转了几个公司，现在已经是带着小团队的“总监”。

我说，这几年间你升职加薪，走上人生巅峰，让人好生羡慕啊。最重要的是可以根据自己想法去实施一些方案，而不会被束手束脚，跟随着高速成长的公司也能让自己的思维和能力快速成长。

而他谈到小公司的经历，眼中难掩失落，反而羡慕大公司内提供的坚实后盾。更让我惊讶的是他其实已经在找BAT的机会，准备年后就进入巨头企业了，即使放弃管理者的头衔做一个普通员工。

所以你看啊，大公司就是一座围城，外面的人想进去，里面的人想出来。

而他们想进去或想出来的原因，其实都是一样的东西。

我的第一份工作,是某某电力.

求职过程很有意思:有一天辅导员有家国企严招聘,说都去看看;结果去了发现只有几个人在,一位大姐说:人都到齐了吗?那就开始吧.我介绍一下我们公司,你们愿意来的话,就填个表. 完全没有面试….就算是有了第一份工作.

毕业后就去报到了,结果报到时间还没到…玩了几个月才又去.简单地培训了两个星期,做了一个安全生产知识的考试,就给所有人买了火车票,送到了四川一个偏僻的小山沟里.那里有一家火力发电场正在修建.

初到这里,什么也没有.有一片正在修建的房子,刚盖好框架,封了顶,地面都还没有平整,这就是我们的宿舍.每人发了几百块钱,说,开车带你们去镇上买点生活用品吧(工地上有不少的大小货车).然后生活就这样开始了.

规定的上班时间是8点到5点;天气非常热,经常只能在屋子里猫着.还好我们不需要天天下工地.有一段时间比较无聊,每天打游戏,但是没有网,只能打单机版的大菠萝.有的日子也经常需要加班赶进度,因为进入梅雨季度了施工就会受影响;但是我们这些质检岗位是帮不上忙的,我们的日子,是负责编质检报表,厚厚的报告拿过来手写上质检记录,反正师傅让这么填的,天知道有没有真正的检查过….

工地上,一般是不允许夫妻同行的,整个工地上几乎全是一水的大老爷们.下了班,业余生活就基本可以用吃喝嫖赌来概括.当然,我们刚毕业的学生,也就只能参于一下吃,每天轮流着去小镇上少得可怜的几家餐馆去吃.

老师傅们,除了吃喝,还赌.有时候通霄地打麻将.旷工就旷工呗,反正也不会怎么样.辛苦钱,还算可观,加上各种补贴,安全奖金啊,其实不算少.那两个月七七八八加起来是3500一个月,但是在这山里,基本没地儿花.当然,后来因为每天轮流请客大吃大喝,花光了.

当然,还有嫖.谁谁谁去嫖了,基本不是新闻.没办法,没别的娱乐….

到这里一个星期的时候,有天在外面吃完晚饭,有位师傅说,你们要不要我带你们去找个小姐?几个刚毕业的孩子吓尿了.

打那后,我就决定要辞职.我觉得要不了几年,我就会成为这样一个人.

于是,在工地上又呆了两个月,辞职了.

在家玩了一段时间,春节过后,我带着仅剩的1000块钱来到了这个有着几千万人的大城市.(我打肿脸充胖子,给了家里一点钱…)

不再是电焊质检工程师,我的新职业是PHP工程师,月薪2500.买了火车票,还剩下700块.住在一个380一个月的地下室.然后中午在公司吃,早晚都吃那种北京的一块钱的大饼,当时感叹,北方的大饼真大啊….

来了没多久,老板说要招人,一直没招着.我想起有一个兄弟,因为没有拿到毕业证,还呆在学校旁边,没有正经找工作,好像呆在一个什么地方打杂,一个月500.我说,把他招过来吧.然后就是哥俩一起住地下室.

公司很小,偶尔会晚一点发工资.有一天,有个人离职,走的时候对老板哭着说,我500块的时候就跟着你干 ,干了3年,没要求地啥,你给我什么? 那之后不久,我也换了一家公司.

这次换了之后,看起来公司会靠谱一点,因为有某知名网站的CTO坐阵.但是那时候显示鉴别水平太低了…这段时间里其实一直被教育,你的努力,老板看得见.但是事实是,老板看见了,就只会开心一下而已,又能怎么样呢….我从来没想过为自己争取点什么.

新的公司,老板同时还有房地产业务和广告公司业务.租的地方很宽敞的样子,搞了个神秘气派的会议室,三天两头拉着据说是政府高官来开会,老板也总是一幅指点江山的样子,说是在通州弄了片地搞开发,但从没见过有啥进展.广告公司呢,好奇葩,是个清华美女负责的,是老板的…小三.

广告公司在18楼,有几个设计师,我认识一个,高高瘦瘦的男孩,我悄悄打听了一下,他的月薪是….1400!

后来的离开也有戏剧性,有一天快下班时候,来了几个警察,有几个律师来了.原来是有老板们的分歧,都说公司是自己的,要我把服务器代码复制出来交出来,要是交给另一方的话,要告我侵占公司财产….

得,我写个代码还要写进监狱啊,那我辞职还不好吗…..

我又失业了.

这次,我一定要找个更靠谱的人介绍个工作.

所以,我找了一个我认识的某知名博客网站的php开发经理介绍,当时他刚刚要离开,去南京创办一家在线旅游网站(没几年就美股上市了)

大牛介绍的工作,从名字上绝对靠谱,是某某电视台的网络中心.在农村小镇,你要说你是这电视台来的啊,镇长马上得来请你吃饭.

面试过程也简单:你是介绍过来的呀,那不用面了.你期望月薪多少?8000?好吧,没问题,现在还在职吗?不在职?那你下周一就来吧.

入职是在月底了,没过几天,有个妹子拿了个表过来登记身份证号,然后两天后就发了5000块钱现金.我心想,这国家单位就是不一样啊…我才上了几天班,就发钱了… 没几个月我就知道,这是劳务费.我们根本不属于编制人员.编制,是在这个单位的生命线.有编制,你就不用干活,有活让外包做就行了.几乎每周都有各种公司来讲方案,而编制类的老大们就只是做决定,买哪个系统. 这里做点事很难.会议很多,效率很低,很讲究级别.有个会议,一个总监没空去,叫一个下属去代开.进门就被轰了出来:你什么级别啊,这是你能来的会吗?

当时是离奥运还有一年四个月,我们的工作就是上线一个相关的站点,要在离奥运一年前上线.还好时间快到的时候,上线了.

接下来的几个月,一直没有发工资,连劳动合同也没有,当然,这之前两家也都没有.期间问了几次,但没答复.有次问急了,就安排去一个考试,还是北京人事局组织的,我没考不过,因为考的是政治和英语.没有英语专八和考研辅导是没指望的.

我和那个登记身份证的妹子熟了起来.我问她,你发工资了吗?她说,这几个月还没有呢,我算实习.我大学在比利时上的,这边不认.所以我只能在这实习,好把档案挂这.我说,那一个月多少钱?200. 我目瞪口呆.依你开的车,200就够养一天吧?妹子说,我来上班是为了有点事情做,也不是为了钱.

好吧,你家住王府井,你开A6上班,我可不是,我老家房子都摇摇欲坠随时要塌了,我上班做事就是为了钱.

我打电话给当时负责的大部分的老大,我诚恳地希望能结一部分钱.他在电话里说,你不是在找人打听吗?听说你不是要走吗?你牛B你就走啊? 我录了音,但后几年之后觉得已经没什么必要了.丑陋的人到处有,犯不着揭开那些不痛快的记忆.

我决定去申请劳动仲裁. 我问那位大哥,你介绍我过去,我去申请劳动仲裁,会对你有什么不好的影响吗? 他说,你放心去吧,干活拿钱,天经地义.

其实,我也没啥选择,人总得吃饭.要不然我怎么办,兜里没钱了,难道滚回老家去种地吗?

2007年,我在劳动局大厅犹豫了5分钟,在想,要不要问问,我可不可以满足那个"农民工免仲裁费"的条件呢?

一个月后,开庭. 我见识了各种无赖的狡辩.比如说,现在说,他是实习生,实习生没有工资(已经毕业的就不再是实习生了),现在已经过了60天的诉讼时限了(完幸书记官看了一下日历,我提交材料的时候刚好是第60天).

我已经忘了是当庭宣布了结果还是后来通知了. 只记得,是电视台的法务通知我去找财务领钱.去的时候各种忐忑,心想平时找各位头们可是千辛万苦, 这次能顺利吗?

还好,财务已经等在那里,数完钱,签字走人. 回来后,法务打电话给我,很生气,不想见你. 我心里想,我已经不生气了,我再也不想和贵台有任何交集.

这次之后,我终于换到了一家靠谱的外企,虽然没几年他也从中国互联网中出局了.

记我那些可怕的职业经历.

原文链接：http://blogread.cn/it/article/7712?f=wb

我在2015年3月入职锤子科技，最近几天离职，现在特别想把这不到两年的时间里的经历和我对这家公司的想法写下来。最近一段时间公司发生了大面积的裁员，但是我并不属于这一次陆陆续续的裁员的范围，而是自己提出离职的，最后发生了一些不愉快的事情，后面也会提到。

我2012年本科毕业的时候对自己要去什么样的公司完全没有概念，我的专业是软件工程，但是当时不想去任何一家IT 公司，于是我选择了出国留学。锤子科技是2012年5月份创立的，我在大学期间是一名典型的罗粉，但就是这样我还是没有太关心这家由老罗创立的公司，心里甚至觉得老罗有玩票的嫌疑，而且那个时候出国的手续基本办完了，所以也有点行色匆匆的意思，就没有特别关注这家公司。

锤子科技的第一个ROM是在2013年3月份发布的，当时人在国外，全程看了发布会的回放，我第一时间刷了这个ROM，虽然问题多多，但是非常喜欢这个系统的很多细节。我第一次动念头想要去锤子科技上班是T1 发布会之后，当时T1 发布之后我也是属于第一批预定的用户。因为我是南方人，回国后找工作简历直接先投了锤子科技，因为如果不行，我是不太可能来北京工作的。当然从面试到最后入职都很顺利，我入职的时候差不多是T2 这个项目刚开始的时候。说到当时入职锤子科技我还是比较感激公司的，因为这是我的第一份工作，之前没有任何Android 相关的开发经验。入职之后知道公司在当时项目非常紧张，需要来了就能干活的人·。想到公司愿意招我这个新人，我心里是有几分侥幸和感谢的。

锤子科技在科技圈的公司里可能算名气比较大的企业了，对于它的各种说法和想法都很多，这两年时间确实和公司一起经历了很多。但是在这个时候我最想提的一个点是公司的一次营销活动，就是让大家参与进来以“天生骄傲”为题写一些自己天生骄傲的经历。我觉得这个可能是这个公司最吸引我的点，也是我作为一个罗粉，老罗最吸引我的点，就是有一种永远要做正确的事情的骄傲。

我还记得的一个“天生骄傲”的故事是老罗转发的一则，一个人因为工作原因用车很多，他开的是自己的车但是公司会给他报油费，他每次会特别留心记下哪些是因为工作原因用车，哪些不是，每次向公司报销的时候，会去掉平时非工作用车的里程数。·我记得原作者的表达要更让我震撼一些，我自己转述的有点啰嗦，反正意思是差不多的。刚到公司那一会我的确相信这是一家天生骄傲的公司，并且可能是由内而外的，并不简单是公司的营销策略。

因为就我当时的观察，来锤子科技入职的同事，有一部分是和我一样的罗粉，他们可能是被相同的价值观吸引来的。还有一部分在入职之前并不知道老罗是谁，但是加入公司之后对这种价值观是很认同的，当然这肯定不包括所有的同事，但我想在我入职那会儿应该包括了大部分同事。但是现在是不是这样，甚至是否公司还能相信自己天生骄傲，我都是不敢确信的。

我一直觉得，我的性格是那种，如果在有选择自由的情况下，还愿意做自己不喜欢做的事情，那肯定是有一种我认为正确的价值观或者理想之类的东西在激励我，我相信这也是很多人的情况。我在锤子科技工作的第一年，有一段时间项目特别紧，几乎每天都要工作到9 到10 点钟。我是很不喜欢加班的，工作做完都是想尽早回家做自己的事情，毕竟工作这么忙了，难得有时间干自己喜欢干的事情，说白了我理解的工作就是自己生活的保障，并没有什么特别大的野心要在并不是自己兴趣的工作上有什么作为。虽然这样，只要是觉得有必要在当天做完的工作，不管到多晚我都会做完的，当时让我有这种决心的，可能就是那种对公司的认同感。

我入职的时候，面试我的软件副总裁是跟我说过公司的工作时间制度的，锤子科技是采用那种弹性工作制，虽然有一个规定的早十点到晚七点的工作时间，但是只要能保证工作做完，并不强制一定要十点到，七点走，但是如果规定时间内工作做不完，就得自觉加班。这是当时面试我的软件这一块的副总裁原话跟我说的。

说实话，我当时对这种工作制度的想法是很理想化的，非常认同，并且也是这么做的。刚入职那会安排我的工作是先理解代码，看文档，刚开始的一周我基本晚上都是7 点准时走，说实话一天时间里有些代码我翻来覆去能看两三遍，基本也都能看懂，而且我也不觉得再多看一两个小时效果会好，毕竟按我学习东西的节奏我不想一天时间看太多东西。但是第二周我们20几个人的组的经理马上找我谈话了，我立刻就意识到有些事情可能并不是我想象的那样，不过他最后给出的理由我也算勉强接受了，他给的理由是要和组里的同事保持步调一致，不然可能有时候找不到你。当然在之后差不多两年的工作时间里，我慢慢对弹性工作制的认识从理想化的状态变成了公司不愿意给加班费，因为所谓的弹性工作制就是一种可以让你每天加班到10点但是不用付薪水的制度。

关于工作时间，甚至做到了你没事情做也得在公司待着的地步，感觉就是如果7 点准时走会有人不爽，而这个不爽的人就是我们软件部门实际的负责人，这个人后来升到了软件副总裁的位置，在他升上去之前有些东西还处于含含糊糊的地带，等到他升上去了之后这些都变成了实质的制度。顺便一说，我在公司的两年多时间，弹性工作制从来都是从7 点往后弹的，这倒不是说我们真的有多忙，只是如果你每天的工作时间恰好只有9 个小时，那就会有人找你谈话，不管你的工作做没做完，做得好不好。

当然我还是愿意相信公司的很多同事是从理想化的角度来理解弹性工作制的，包括当时面试我的前副总裁，他是一位很和蔼的中年大叔，大概好像几个月前退休了。可是为什么在我眼里公司慢慢变得不那么骄傲了呢？甚至我现在确信公司已经不再有脸面在自己内部员工面前说我们是一家天生骄傲的公司了，为什么会变呢？这可能得好好说一说我们这个组的经理，从他可能能侧面说明这个问题的原因。

我们的这位经理是在原来他的经理升上去之后被提拔的，可能就是所谓的自己人吧，但是这个人到底能力怎么样，我觉得可能公司并没有明确考核过，或者甚至本身就没有考核这一步吧，毕竟我对公司管理不是很清楚。我们的软件组里其实并没有明确区分技术和管理的Leader，可以理解为我们的经理需要兼顾这两个方面的工作。

我们组大概维护手机软件部分的大大小小几个模块，锤子的软件部分可以说是很出彩的，但是说我们公司软件的技术多么牛逼，我真不敢说，因为我们的工作主要在于实现产品提出的需求，所以只要能实现了产品功能和性能上的需求，并没有人在意你是怎么实现的，至少我们组完全没有所谓软件架构，如果你负责的某一个小模块需要实现一个新需求，那么从设计方案到软件架构到coding很有可能都是你一个人完成的，公司很有可能没有第二个人知道你是怎么实现的。

这里说一下在我眼里我们组的经理每天的工作似乎就是满足于不挨他上级的骂，所以像整个组的bug 数量和新需求是否超期这些敏感数据是他最关心的。经常遇到的情况是，有一些bug 比较不好分析，或者需要仔细分析前因后果再做合理修改，但是他会在你分析bug 期间催你，经常用到的句式是：“某某，你那个bug 优先级比较高，快点看一下”，好像我们没有在看一样。而且有时候为了降低bug 数量，他还要经常帮我们解bug，有时候他不好意思让我们来加班，自己周六跑到公司解bug，然后我们周一来了发现他解得不对还要帮他擦屁股。倒不是说他技术有多差，只是组里那么多模块的代码他并不都熟悉。只要当天bug 数量降低了，他就满意了。不过话又说回来帮组里同事解bug 也算是他完成他技术Leader的工作吧。

另外虽然他是大组经理，但是其实下面按模块还是分了几个小组的，一般小组内的工作都是我们组长分的。我们小组组长的方式我是很认同的，有的时候他并不会主动给我们分bug，我们会自己从他那里取，这种方法看似放任，但其实我们组长是基于对我们的了解以及建立在这种了解之上的信任才这么做的，这种方式和公司的弹性工作制本质上是类似的，我觉得我们小组内这样合作的方式是很合适并且效率很高的。然而我们这位经理经常越过我们组长给我们分bug，有的时候他不太了解我们组内工作的分配，这样分bug反而会给组内同事造成困扰和压力。在工作的这几年时间内，我们的经理似乎对我们完全没有什么具体的了解，只是象征性的组织过几次团建，吃过几次饭而已。

我想这对他做管理方面的工作是很致命的，因为他要做的不只是每天催我们尽快解bug 而已。也有很多同事为他解释，说他是工程师出生，人又内向，所以有时候表达不到位，但是我想说的是，既然这样，他为什么能做20 几个人团队的经理？他似乎只是作为一个单向的通道存在，项目催的紧了，他就也来催我们一下。其实问题可能很简单，因为他是自己人，有的时候看到他被骂心里其实也很难受，但是这样能力的人当上经理，可能也是从侧面反映了，为什么我越来越感到公司不那么天生骄傲了。

我写这些的目的是很明确的，字里行间都能看出我是带着对公司管理层的怨恨的，因为他们可能就是锤子科技变成今天这样的罪魁祸首。这里说一下我离职的原因，今年公司软件团建的时候采取了很幼稚的形式（发小学的时候用的那种大红奖状，完了竟然没有奖金）奖励了一些加班时间最多，解bug数量最多的同事，我对这样的价值取向是很不认同的，因为这对不用加班就可以把工作做完甚至做得更好的同事是不公平的。

借着酒劲，我在公司群里说了很难听的话，顺便在当晚老罗和我们互动的微信群里问了过年红包还会不会发，结果是直接被退群。当然我后来很后悔，本来年后我也打算离职回老家了，还是忍一忍的好，毕竟后面经历的事情都非常荒诞。我再到公司的时候发现我的办公电脑被收走了，据说公司邮箱也被封了，据说是经理怕我再做出可怕的事

情说出可怕的话。但是公司HR对我说公司的正常裁员已经结束了，也不会开我，这里我真想说难道让我把电脑搬回来继续干吗？我实在没有心力和公司耗下去，就自己提了离职。

我在职期间有一次经理找我聊天问我为什么最近工作不积极对公司有什么不满，我主要对他说了两点，一是公司取消本来就屈指可数的中秋节红包福利为什么不在公司层面通知一下（我和同事等了一天发红包，虽然锤科福利不多，但是逢年过节是直接给现金红包的，每次领红包都很开心），二是公司裁员为什么采取遮遮掩掩的态度，因为即使你不说外面的人一样知道，反而如果你对内部员工都不开诚布公，却会造成我们的恐慌和懈怠。

经理的答复是，这个不可能按你想象得那么理想化的，其他公司都是这样的。其实从他这番话也大概能了解为什么公司在做出这些决定的时候是这样的态度，因为公司的管理层似乎都被他这样的人占领了，他这句话在我看来好像就是锤子科技变得不再特殊的宣言吧。这里想告诉那些还想来锤子科技入职的朋友，了解一家公司不能光看公司自己的官博。

其实最后我还想说一句，锤子科技没有做错任何事情，因为站在公司角度来看，简直可以用为了盈利把所有问题都说通，但是这家公司已经不能再公开宣称自己天生骄傲了。也许正是被天生骄傲的价值观吸引来的，在离开的时候看清公司的现状才不会有任何遗憾。

【导读】在中国互联网竞争加剧，让巨头们对用户的竞争很激烈，对人才的竞争更激烈。中国4家市值超过100亿美元的互联网巨头公司BAT3，对待员工的方式，以及组织架构都各有不同，因此造成了各自风格鲜明的企业文化，究竟巨头内部员工生活工作状态如何，是什么原因造成这种情况的，值得我们实地考察一番。本文整理自“每日CEO说”。

阿里巴巴员工：我感觉我是在为自己做事，不是为公司

王小炜(虾米网创始人，原阿里巴巴员工)：阿里巴巴有强大的HR体系，我目前我在国内看到效率最高的体系。整个HR体系几乎每一件具体的事情都有办法和具体落实的流程，具体到战略怎么制订，目标怎么样分解，考核的指标定的是不是准确等，这都是HR的事情。而对于人的方面所有的业务主管，基本上业绩只占到我们的50%，有30%是团队，还有20%是文化。

阿里巴巴是所有互联网巨头中最喜欢给员工灌输价值观、营造企业文化的公司。“江湖”传言阿里的人，工程师都被洗脑，很多猎头也表示阿里的人最难挖。马云一直强调要有自己的DNA，同时也强调value的重要性，同时有一句话让人关注，一万人的believe就是信仰。他能把自己的梦想和价值观源源不断地灌输给你，用其“扭曲磁场”使得你产生认同并与之共同努力。对于不懂技术的领导者而言，他不会跟你坐下来讨论技术问题或者产品问题，他更像是一种精神存在，一盏指明灯，告诉你要往哪里走。这样的文化收获了什么?某位阿里员工表示：“阿里巴巴也没有宗教式的崇拜，我们崇拜的是自己，崇拜自己的事业。”阿里员工难挖，很重要的一点是因为他们认为在阿里是在为自己做事情，当然，阿里巴巴工资高也是不争的事实。

百度员工：推崇狼性，公司内部压力太大

3B大战之后，百度在内部反思中提出要推广狼性文化，要求员工要以结果导向。百度一些老员工说，“在百度比较怕的是新人，老员工和新员工永远在一个起跑线上在竞争，论功劳，不论苦劳，优秀人才可以不拘一格地被提拔上去。对于新人而言，这是一个很好的竞争环境;然而对于老员工而言，来自内部的竞争压力未免也太大了。”

对于竞争激烈的互联网公司而言，提倡狼性文化完全无可厚非，但是这种狼性应该体现在对问题的处理、竞争对手争夺中，要干劲求结果;对内部也狼，就很不人性了。百度的狼性文化让所有员工都保持了一颗时刻学习的心，也把很多老员工推向了竞争对手的怀里。

360员工：进公司容易出去难

某位360在职员工表示，在360：“做错了不会被开掉，甚至由于管理机制的松散也不会被记录在案，但一顿痛骂是在所难免的。这种暴力的风格像病毒一样自上而下传下来，每一级的Leader都被深深传染，大家都充满戾气，对下属犯的哪怕很小的错误都很容易大发雷霆、说脏话、拍桌子甚至摔东西。除了周鸿祎身边的极少数人，其他人都时

常被骂得狗血喷头，颜面扫地。”

不过，360也有很多优点，否则早就招不到人了。360跟水浒梁山很像的一点还在于：即便你是小喽啰，也可能天天看到宋江——周鸿祎。该员工还表示：“360的上级会像家长那样，对个人表现出极大的关心，老周会突然跑到你身后看着你的电脑屏幕拍拍肩膀问你最近怎么样?离开时随手拿起你桌上的零食边走边吃。”如果你觉得这是老周信任你的表现，那你可就大错特错了。因为“周鸿祎则压根儿就从来没信任过底下人，对总监以下(包括任职期限不长的很多总监)都是进行信息屏蔽的。”

在这样的文化下，新人可能因为周鸿祎突然的一个问候产生有存在感，也可能因为一次狗血淋头的批骂而甩手走人。新人可以随时打破规则与流程做出成绩，但是一年多以后就会因为信任天花板感到迷茫。而当你迷茫想走时，梁山泊的特质再一次显现了：“任何一个员工要跳槽都要经过几道关口的反复盘问，一直要谈到齐向东，如果在比较重要的岗位，周鸿祎会亲自约谈。去哪里?为什么要走?对什么不满?待遇提高一些是否愿意留下来?如果发现你是要去竞争对手阵营就会更难放行。如果你是真的不喜欢这里或有非常好的选择决意离开，就会非常麻烦。据说需要签很多约束性文件，让你感到跳槽风险很大，弄得就像离一次婚一样，麻烦到让你最后放弃跳槽为止。”

腾讯员工：注重对我的培养，却不懂得如何留住我

曾就职于腾讯的董先生说：“腾讯内部环境氛围都不错，待遇上也不错，整体员工素质也很高。”在腾讯的企业文化里，对员工的发展坚守的是“重视员工成长”，如果你是毕业生，在腾讯的工作幸福度将会非常高。原因是“你可以参加新生培训，同时还可以接触到强势的平台资源。大量的新人虽然是刚刚毕业做产品，哪怕全新做一个产品，在QQ tips和其他交叉推广资源的拉动下瞬间用户量过百万那是常事。这样的成就感是在其他公司比较难享受到的。”

问题是：对于新人而言，腾讯的架构太成熟了，在这里个人的价值被平台价值掩盖，因此不少新人还是会选择离开腾讯，原因很简单，在这找不到自己曾经梦想的那种激情。 于是，新人刚刚成熟后，也想要离职;但是对比360的层层盘问，在腾讯想走人则轻松的多。董先生他的领导重金挖过来的人才，但是在最后的离职面谈中，HR更像是例行公事，并没有做深入沟通。这样一来，腾讯花在员工培训上的成果很可能是为他人做嫁衣裳。因此，腾讯的企业文化看起来很好，但是如何落到实处很重要。

扫清Android面试障碍

怎样快速突破初级瓶颈，变身高级开发？怎样在短时间内提高自我身价，月薪提高50%？你是否是个代码高手，面试中却发挥不出来，想进阶却摸不着头脑。博主在互联网行业摸爬滚打，百面成钢。特来总结与分享自己面试的心路历程和经验。

本系列将分为四大部分切入，包括**第一部分面试前的准备**：从长期来看要做点什么来提高自己的身价，短期内怎样迅速进入面试状态，以及如何准备简历才能吸引HR的眼球。**第二部分是安卓初级中级高级面试题**，大概会有几十个问题，基本覆盖了小白到大牛进阶的所有会被问到的问题。**第三部分是除了安卓经验**，你还会被问到哪些问题。**第四部分是关于 offer的选择**，以及一起探讨未来的职业规划。从这四部分看来，已经涵盖了一个程序员的方方面面。我一直觉得面试也是一种考试，只要是考试就有套路，有套路就能被总结，能被总结就能被练习，就好比高考模拟卷或者培训班，能教你的也就是反复练习。

先交代一下背景：博主本科毕业于国内某985大学，10年大二开始接触Android开发，参加了两届谷歌举办的全国大学生安卓应用竞赛并获奖。实习和工作都在上海的互联网公司做app开发。曾误入 iOS深处，沉醉不知归路。后来惊醒还是转投回安卓的世界中去。目前工作三年。本文的目的在于总结回顾上一波的面试，给未来的自己看看当时的努力与心情，更重要的是希望对业内还在迷茫中想要提高自己的开发们指点迷津。

前两个月，我面试了上海多家互联网公司，按规模上至阿里携程新美大，中至爱奇艺安居客B站平安科技，下至格瓦拉触宝科技蜻蜓fm，偏至招行信用卡微鲸科技等。并非有意要面那么多，只是一开始行动力太旺盛不小心简历投多了。秉承“自己投的简历，跪着也要面完”；以及想一窥上海互联网公司发展情况等多种心理下，横扫了各大Android面试题。经验积累了不少，从简历准备到技术准备再到扯淡环节，甚至于怎样针对不同的公司不同的面试官提出不一样的问题等等。

这整个过程不是一帆风顺的，就像给程序做优化一样，一点点完善、一点点让自己的表现趋近完美。我当时甚至会在家对着镜子针对可能被问到的问题，练习说话的语速和表情。请理解一个心机 girl 总会想太多，比如有的问题面试官或许会觉得这道题是他的杀手锏，那你就不要表现出一副做了充分准备、分分钟秒杀他的气势。毕竟咱面的是普通开发，最后还得在人手底下做事呢 ~ 这种情况下最好的办法是面露一点点思考与回忆交加的表情，一点点说出对方希望你说出来的答案；而不是一股脑全说完了。当然了，不同的面试官有不同的套路，比如我最欣赏携程的一个初面面试官，能感觉到那是一个智商超群的人。和聪明人讲话，你就不用再拐弯抹角了。Anyway，假如这个面试官最后露出了满意的笑容，那就恭喜你成功了

面试真的是个体力活，有两次我都想放弃了懒得再面了，让我继续坚持下去的除了有始有终的信念以外，还有照云兄一句“ 所谓自由就是随时拥有说'不'的权利。” 我理解就是给自己多一种选择。这一波面试下来，最遗憾的还是阿里几个想让我去的部门都在杭州，但我是上海人还要在交大在职读研。另外，上海的BAT Android岗位实在是太少了，腾讯基本就不在上海招。曾有冲动为了工作收起行囊就那么离家去杭州，但理智最终制止了我。阿里现在有点像软件界的华为，基本打来的电话都在晚上八点，有种血汗工厂的感觉。总体来说，上海的互联网公司规模都没成气候，还是中小企业居多。看了好几年上海互联网的起起落落，唯有”恨其不争“能形容这种感受了。PS：我最后选择了一份”钱多事少离家近“的工作：请不必问我现在何处，因为我一定会对公司和薪资守口如瓶。

话不多扯，上文的各种体会是为了下文抛砖引玉。这个系列的文章除了全面，更重要的补全除技术以外的短板。比如，当一个面试官问你：**你的职业规划是什么**。相信每个面试者都被问过。

大部分人的内心OS想必是： what，职业规划是什么？ what，这个问题究竟有什么意义？ what，我该怎么回答他？

如果技术面过了，栽在这种问题上岂不吃亏？相反假如你答得好，却会为你的薪资增加筹码。其实这个问题究竟该怎么回答，要具体情况具体分析。首先要看这家公司处于它这个行业的什么位置，它未来的公司战略规划是什么，它想招进来一个怎样的人，它希望你发挥的是什么作用，坐在你对面的是app leader还是CTO 抑或是HR？都会有不同的最佳回答。因为每个人想听到的是不一样的“实话”，实话打引号表示经过包装以后的答案。

关键就是你得理解这话背后的意思，面试官为什么要问这个问题呢，他想知道的无非是：

1. 你会在这家公司呆多久？
2. 你未来可能处于公司什么样的角色中。
3. 你能否适应公司现有文化？
4. 你能为公司发挥多少价值。
5. 面试官还有自己的小算盘。

如果坐在你对面的是app leader，你说你以后想做到这里的技术leader，那你一定是撞枪口上了，除非他就是要离职前招个leader进来。请注意这种情况是很有可能发生的。大多数面试官进来不会介绍他的职位，面试level 也未必和面试顺序正相关。这就好像在一个暗箱中博弈，通过察言观色和判断得到的信息是有限的，知己知彼你才能不出错，然后才是怎样最大化说出对方想听到的接近答案。

面试前的准备

有的时候我们在职场上会碰到各种各样的坑…… 不管是创业公司倒闭啦，新同事合不来啦，氛围太差受不了啦。如果说女人的安全感来自于独立，那么程序员的安全感就是手中有技术，心中有源码。想飞得更高，有哪些是你平时可以做的呢？

本篇接上一篇 扫清面试障碍。上篇主要抛砖引玉，提到了许多技术和非技术相关的面试技巧等。本篇讲述面试前的准备，这不仅是技术实力的积累，还有相当一部分软实力的积累。即使是暂时没有跳槽想法的技术人，也应当生于忧患之中。行业的发展非常迅速，逆水行舟不进则退。危机意识很重要，厚积薄发才能更上一层楼。

友情刺激大家： 《高二Android大牛是这样炼成的》

面试前的准备可分为长期准备，和短期准备。首先声明，文中提到的书籍以及公众号，博客等，都是我自己觉得特别优秀的，绝没收任何广告费

长期准备：

1. 订阅几个高质量的公众号

很多大厂都有自己的微信订阅号，很多人的阅读时间非常琐碎，那么你就可以在地铁上、在等公交车的间隙去刷刷最新技术的发展。订阅号贵精不贵多，某个订阅号里的文章一天也最多一篇，不然就容易粗制滥造。这里推荐几个高质量的android公众号：

腾讯Bugly: weixinBugly。

是腾讯bugly，专门做性能监控的平台，里面的文章也都是总结bugly bbs上企鹅工程师文章的精华。发送频率很低，质量都不错。后期面试中很多关于内存泄漏的答案都来自于此。

移动开发前线：bornmobile。info主编徐川个人的微信公众号，里面的文章有原创也有很多精华集锦。关键是都比较严肃，不像某些个人公众号发展到后期变个人的独白段子手，干货也多，由于大都是搬很多个人博客和采访，所以质量还不错。

Android程序员：androidtrending。公众号所有者应该是汤涛。虽然Android技术公众号不少，个人感觉这个公众号是比较能紧跟潮流的。也没有什么废话而是纯技术。

程序媛：programgirl。一个程序媛发声的平台，更新不定期，`欢迎所有女程序员都来投稿，投稿地址：chaojimiaomiao@qq.com`. 比如本期android面试就会连载在里面。

2. 加入一个本地android组织

大学时期我开始沉迷于android技术，但当时不善社交自命清高，往往是孤军奋战。回过头看，一个人的力量是有限的，只有交流才能进步，闭关锁国是没前途的。结识一批有同样追求的同行，甚至有很多比自己厉害许多的专家人才，你才能看到你未来努力的方向。人外有人，天外有天，千万别做井底之蛙。信息时代信息的交换是有价值的，小伙伴们一起努力也会将你带入一个好的氛围。

但是要谨慎地选择伙伴。我知道男生聚在一起最容易犯的错误是互相影响，比如大家一起去打一盘游戏啦，各种吹牛或恭维啦，，这些都是不可取的。低调而务实的小伙伴们聚在一起才有创造力，你们共同完成一个梦想一件事。

这里举个例子，https://github.com/LittleFriendsGroup/ 里的AndroidSdkSourceAnalysis。 他们发起了一个开源项目：18天认领android源码解析。这就是共同进步共同成长的典型呐~ 朋友应当是志同道合的，而非没目标的乌合之众。

3. 看几本必看的进阶书

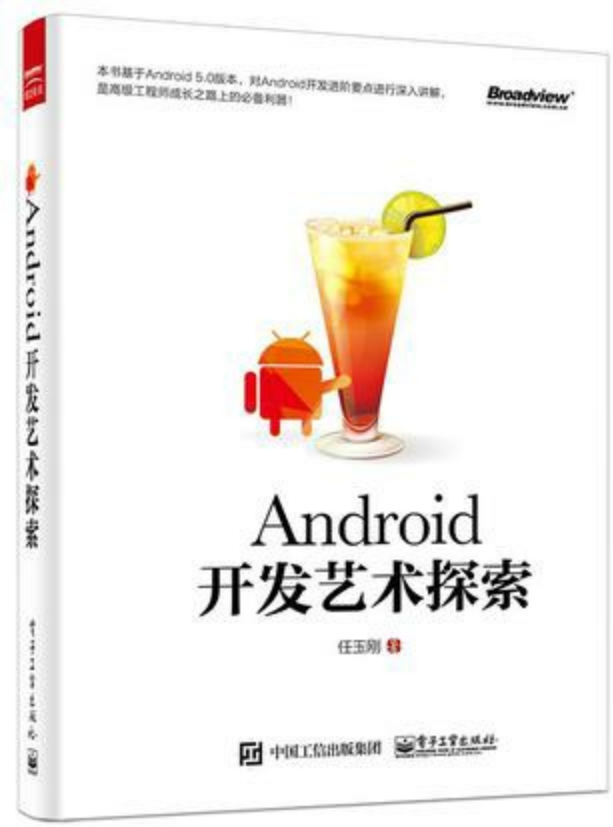
市面上入门书籍泛滥，而适合中高级的主流技术书实在太少。一方面是能写这种书的大牛懒得写，据我所知每卖出一本技术书，作者才拿到4块钱稿费。而中高级的受众面毕竟窄，即使热卖，卖出一万本，那么所拿稿费也不过是4万。然而有这种水平的开发者根本不缺这几万块钱。写书是一件特别耗费精力的事，除了内容还要排版，吃力不讨

好。更何况还会面临盗版横行的情况，就更不值了。写书仅剩的吸引人的价值，恐怕是成书后带来的名声与成就感。由于这种客观情况的存在，市面上粗制滥造的书实在太多，我在此帮大家剔除糟粕，列举几本我觉得值得的书。

推荐书单：

《Android开发艺术探索》任玉刚

业界良心！里面的很多东西面试都被问到了！



《App研发录》包建强

App研发录：架构设计、Crash分析和竞品技术分析。这是一本非常务实的书，书里提到的内容都可以直接拿来用在项目上。这个作者原先是搞.net的，工作十多年经验非常丰富。

摘录一段读后感：

> “落地”，是阅读此书的最大感受，大到技术架构、竞品分析、发布流程，小到每个Crash、员工座位安排、开展技术分享、简历筛选，作者总在寻求落地的解决方案，细细阅读，你不觉得这种方案有多么高大上，不觉得背后有多么华丽的理论支撑，但是你会觉得，一切都是踏踏实实、真真切切可落地实践的，为工作带来不少的实践指导。

《Android内核设计思想》

这本书写得很深，基本是这三本书里最接近底层的书了，读起来也比较晦涩。需要有一定的理解能力。 全书从操作系统的基础知识入手，全面剖析进程/线程、内存管理、Binder机制、GUI显示系统、多媒体管理、输入系统等核心技术在Android中的实现原理。书中讲述的知识点大部分来源于工程项目研发，因而具有较强的实用性，希望可以让读者“知其然，更知其所以然”。全书分为编译篇、系统原理篇、应用原理篇、系统工具篇共4篇22章，基本涵盖了参与Android开发所需具备的知识。

4. 收藏几个博客，紧跟几个专家

有一阵我很迷茫，不知道继续进步的方向，也不知道我能达到什么样的程度。我甚至因为找不到安卓上进步的空间而横向求索，玩了一年的iOS开发。就在这时，我的偶像来了。偶然间一次搜索，翻到了 hukai.me。（当然发现作者长得帅也是一个激动的原因）

原来安卓还可以这么玩~！可以像胡凯一样紧跟谷歌的最新视频教程并翻译它们。 也可以像 代码家 那样写控件造轮子。代码家甚至因为轮子造得好而被谷歌发现，发出了工作邀请。

具体你可以关注哪些安卓的优秀博客可以见 这个链接:

博客地址

- 柳志超博客
- 代码家
- 胡凯
- 禅
- Trinea
- 方杰
- 技术小黑屋
- 程序亦非猿
- 脉脉不得语
- 陈启超的博客

5. 写自己的独立技术博客

见贤思齐焉。

有了那么几个专家榜样，我的视野突然变开阔了，也想变成像他们一样厉害的人。这个时候你已经阅读了几本书，也有了一定的积累，自然而然也会想发表自己的看法和见解。

总之，克服瓶颈期的最好的办法就是六个字——总结、归纳、演绎。把你的收获都写到自己的博客上吧。最好是独立的博客，一方面你会收获到折腾独立博客的乐趣，另一方面它也是你的个人品牌，假如你能长期积累、坚持的话。独立博客唯一的缺点是导流。当然，你对自己的技术总结，目标是为了精益求精，为了自己的进步成长，至于读者多不多，倒是其次的。这个后期在面试中也大有裨益。别人刚认识你的时候对你不了解，你扔出一个博客地址，胜过你的千言万语。

没事的时候就去写写博客吧，记住**不要写很多生活琐事**，而是**记录进步**。别人不是为了看你的个人秀而来到你的空间，而是搜索到你对技术新的见解，觉得对他可能会有帮助才过来的。牢骚什么的，写到你的社交网站上去。

有的人很勤奋，博客天天更新。有量而没有质，感觉很空虚。我主张坚持精品博文才是正道，，毕竟信息已经那么泛滥了，不要再为互联网制造过剩的垃圾辣。。。（真的不是教你懒） 想要写一个优秀的博客，不仅要有实际的项目积累和广泛的阅读积累以外，uml类图，流程图，顺序图等，也是你必不可少的好助手。假如再加上**思维导图**，不仅使你的思路更清晰，读者也能一目了然，有一个全局观。现在就开始试试这些工具吧，会让你的博客更严谨更正式，更熠熠生辉。

6. 看源码

这里列举几个你必须要看的源码:

- Binder
- LruCache
- Handler
- AsyncTask

- EventBus

不要怀疑，这些在面试中一定会被问到的。前期你看过源码也会淡忘的，先有一个大致的印象就行。

7. 提交自己的开源代码

写什么样的开源代码，有这么几个方向。

一是你博客中的例子。一个好的技术博客不仅有图有文字，假如有例子，更胜过苍白的自然语言。百闻不如一见，说得再多也不如实际一个例子。

二是自定义的控件基础库。这个主要集中在酷炫的动画，还有交互更简洁的自定义控件上。这方面的大牛比如很多国外的库，还有国内代码家博客上提到的开源库~

三是有关测试方面的工具。像leakCannary, blockCannary等，能让开发者更好地测试内存卡顿泄漏等。

四是一些反编译的工具。

五是一个完整的上架作品。

短期准备：

1. 去NewCoder刷题

这个网站主要包括了几套公司真题，比如有 Android的，算法讲解，数据结构。

有时间的可以去上面刷刷题，题量较大

2. 去极客学院阅读几本微书

没时间阅读厚重的大部头怎么办？有一些不出版的非纸质微书也是极好的选择。极客学院有几个系列 写得不错，比如《深入理解 Android 卷》系列，《Java 集合学习指南》等。

做工程的开发者容易陷入埋头写代码的怪圈，实际上多看一些技术书，吸收别人的思想精华进步会更快。

3. 看几篇分析源码的文章

在长期准备中，你已经接触过一些源码了；当然你可能跳槽时间比较紧，那不如关注一下这个开源项目吧，里面包括了大量的 android源码分析。

ANDROID SDK 源码解析：AndroidSdkSourceAnalysis

如果你有时间，你也可以参与到这个项目中去。

4. 梳理一下知识体系，找出薄弱环节

总结、总结，再总结。

总结是一个人非常重要的品质，在总结中思考自己的不足，完善自我。不仅适用于面试，也适用于职场，日常生活等等。常思考，才可能变成一个智慧的人。

具体有哪些可能的薄弱环节，可以看下一篇整理的30个发散性问题。

简历的准备：

写好简历也是一个重要的环节。程序员最好常备一份简历，以备不时之需。另一方面，更新简历的过程中也会明白，哪些经历是无用功，哪些是加分项。

有几点

1、尽量避免主观表述，少一点语义模糊的形容词，除非是大公司大牛，已经有成果撑腰，否则慎用「熟悉…」、「使用过…」

2、多一点表意清楚，语气肯定的数量词、名词、成果描述。一定要将自己的优势和期望明晰地表达出来，便于招聘方能对候选人有更准确的定位:

- 介绍技术：最近几份工作经历中所参与过的产品、项目、角色
- 在工作中做的项目的技术细节
- 克服过的技术难点与细节
- 感兴趣的技术
- 在程序马拉松上参加的项目或者是业余的个人项目（+link）
- 如果领导过技术团队，写下带的团队的规模与管理风格
- 介绍自己：过往有特点经历、擅长的方向、对互联网的理解、职业发展规划

3、突出重点。重要的放前面，不重要的经历往后排。尽量注明每段经历的时间，不然技术官或hr会一头雾水。比如你面试 Android开发，iOS的经历就放后面一笔带过，另外时间越远的优先级也越低。对了，记得把联系方式写在开头，不然对方联系不到你，可就把你忘了。

3、试试用markdown语法，注意下排版，预览再提交，版面整洁、干净，也是加分项。

4、HR/技术负责人更喜欢看到一份显示「职业上升趋势」的简历。比如你做开发五年，跳过两三次槽。那么你的职位，你所做的技术难度应当是越来越具挑战，而不是依然罗列各种琐碎的需求实现。

5、牛人讲结果，普通人讲过程。话虽如此，你只要不太啰嗦，该列出的事情还是要列出来的。不仅如此，你在该项目中发挥了什么作用，有什么反思，也可以写出来。记住，你出卖的不仅是技术，还有能力。

千万别…

1、真实的反映你的工作，别浮夸。 因为你的面试官之后一定会问起浮夸的事情，假如你不能自圆其说，会留下非常不好的印象。

2、记得最好不要出现“**精通**”等词汇。 很少有人能精通JAVA，精通数据库，这样的字眼基本等于自杀。好的办法是阐述事实。比如用JAVA开发过什么系统，在什么项目中深入接触数据库等等。

3、简历上的所有技术，你必须谙熟于心。 甚至你应当练习几遍，找出可能发散出去的所有问题。只有你非常熟悉，并且在面试中足够自信，你才有能力引导面试官问到你想要的坑中。

4、简历长度不要超过三页。最好是两页这样一目了然，当然如果从业经历比较长，也可以多写点。

5、别撒谎。天下没有不透风的墙。假如你不想让人知道什么，你可以不写；假如你在原公司的锻炼机会太少，那么你可以将经历写到自己的开源项目中去。

三十道android开放题

为姗姗来迟的安卓面试题表示抱歉。 大家可以先思考一下这些问题，答案会过一阵在后文放出。

基础题

- Activity生命周期和启动模式，以及使用场景
- Service的生命周期，如何启用/停用Service
- 可能导致OOM内存溢出的情况有哪些，怎么解决
- Android中的动画有哪几类，它们的特点和区别是什么
- 注册广播几种方式
- 如何自定义View
- 程序crash的所有可能原因
- android动画
- Android操作系统分层

中级题

- Android事件处理机制
- 内存泄露的案例
- 图片加载库的使用及比较，内在逻辑分析
- 网络库的使用和比较
- android线程间通信
- android进程间通信。Binder源码与Looper关系等
- 设计模式在android中的应用，手写几个典型模式
- annotation 以及java反射机制
- Java虚拟机
- Java几种线程池

高级题

- 热修复技术，原理及其应用
- react native
- 几种架构模式
- 新建一个工程，需要加入哪些库和模块，该怎么设计

有关测试：

- DDMS + MAT
- monkey test和logcat命令的常用过滤参数
- android作优化有哪些可以考虑的方向？

开放题

- 当我在浏览器中输入一个url，世界发生了什么。
- http和https的区别
- 几种加密方式区别，对称与非对称。
- 你还了解什么其他的语言？和Java对比一下。

> 原文链接：冰洁的技术博客，http://www.bingjie.me/

最近看到很多人都在找工作, 而且很多人都感觉今年找工作比去年难很多, 竞争力也增加不少, 因此激发我整理这份资料, 希望能帮到正在找或者准备找工作的童鞋们.

首先我们能否获得一个面试机会, 那肯定是从简历开始, 简历需要做好功夫, 一份好的简历才足够吸引企业得到面试机会, 接着就是面试了, 面试前必须要先做好准备, 多看一下前辈们总结面试题, 有哪一方面不足的地方赶紧补充一下, 还有要了解一下你即将面试那家公司.

> 感谢@Android开发日常(专注分享 Android 优质开源项目以及高质量开发资料) 支持

教你写简历

- 你真的会写简历么?
- 80% 以上简历都是不合格的
- 推荐两个技术简历模板
- 精益技术简历之道——改善技术简历的 47 条原则
- 关于程序员求职简历
- 程序员简历模板列表

面试题

- 国内一线互联网公司内部面试题库
- Android 开发工程师面试指南
- 一个五年 Android 开发者百度, 阿里, 聚美, 映客的面试心经
- 整理常见 Android 面试问题
- 2016 Android 某公司面试题
- 面试后的总结
- Android 面试题整理
- Android interview questions for 2-5 yrs experienced
- Android interview questions
- 40 个 Android 面试题
- Android 名企面试题及涉及知识点整理
- 亲爱的面试官，这个我可没看过！（Android部分）

做题

看完面试题之后那就来做一下面试题目吧, 目前找到两个网站

- SillGun(国外网站, 自备梯子)
- 牛客网

聊面试

(帅张)stormzhang 跟你谈一下面试那些事儿

- 面试时企业最看中你什么能力?
- 我面试到底问什么?
- Android 面试那些事儿

知乎讨论

- 面试时, 问哪些问题能试出一个 Android 应用开发者真正的水平?
- 我用个假简历去面试 android 的结果为什么会这样?
- 怎么准备Android面试?

互联网招聘平台

- 拉勾-专注互联网职业机会
- 简寻-让职位推荐更精准
- 100 offer-帮最好的互联网人发现更好的offer
- BOSS 直聘-互联网招聘神器
- LinkedIn (领英)
- 哪上班

感谢

非常感谢上面分享面试资料以及面试经验的前辈们! 有前辈在前面带路, 我们后辈真心感到幸福.

祝福

最后祝正在找工作的的童鞋们, 马到成功, 心想事成, 事事如意!

> 原文链接：G军仔，http://www.jianshu.com/p/d1efe2f31b6d

2016年4月某公司面试题及面试流程

1. 静态内部类、内部类、匿名内部类，为什么内部类会持有外部类的引用？持有的引用是this？还是其它？

- 静态内部类：使用static修饰的内部类
- 内部类：就是在某个类的内部又定义了一个类，内部类所嵌入的类称为外部类
- 匿名内部类：使用new生成的内部类
- 因为内部类的产生依赖于外部类，持有的引用是类名.this

2. ArrayList和Vector的主要区别是什么？

ArrayList在Java1.2引入，用于替换Vector Vector：线程同步，当Vector中的元素超过它的初始大小时，Vector会将它的容量翻倍 ArrayList：线程不同步，但性能很好，当ArrayList中的元素超过它的初始大小时，ArrayList只增加50%的大小 java集合类框架

http://blog.csdn.net/axi295309066/article/details/54089986

3. Java中try catch finally的执行顺序

先执行try中代码发生异常执行catch中代码，最后一定会执行finally中代码

4. switch是否能作用在byte上，是否能作用在long上，是否能作用在String上？

switch支持使用byte类型，不支持long类型，String支持在java1.7引入

5. Activity和Fragment生命周期有哪些？

- Activity

onCreate→onStart→onResume→onPause→onStop→onDestroy

- Fragment

onAttach→onCreate→onCreateView→onActivityCreated→onStart→onResume→onPause→onStop→onDestroyView→onDestroy→onDetach

6. onInterceptTouchEvent()和onTouchEvent()的区别？

onInterceptTouchEvent()用于拦截触摸事件 onTouchEvent()用于处理触摸事件

7. RemoteView在哪些功能中使用

APPwidget和Notification中

8. SurfaceView和View的区别是什么？

SurfaceView中采用了双缓存技术，在单独的线程中更新界面 View在UI线程中更新界面

9. 讲一下android中进程的优先级？

- 前台进程
- 可见进程
- 服务进程
- 后台进程
- 空进程

10. 代码查错题，没记下来

tips：静态变量持有Activity引用会导致内存泄露

11. 一面

service生命周期，可以执行耗时操作吗？ JNI开发流程 Java线程池，线程同步 自己设计一个图片加载框架 自定义View相关方法 http ResponseCode 插件化，动态加载 性能优化，MAT AsyncTask原理 65k限制 Serializable和Parcelable 文件和数据库哪个效率高 断点续传 WebView和JS Android基础——Service Android基础——IntentService Android开发指导——Service Android开发指导——绑定Service Android开发指导——进程间通信AIDL Android面试基础知识总结（一） Android面试——APP性能优化 Android中Java和JavaScript交互 WebView 远程代码执行漏洞浅析 WebView中的Java与JavaScript提供【安全可靠】的多样互通方案

12. 二面

所使用的开源框架的实现原理，源码 没看过，被pass了 去面试之前把用到的开源框架源码分析一定要看看啊 codekk：开源框架源码解析 2016Android某公司面试题

2017届实习生招聘面经（今日头条，腾讯，阿里，360）

1. 阿里内推

在三月的某一天，当我还沉浸在代码世界的时候，突然一声铃声响，拿起手机一看，杭州电话==大三春招第一次面试开始了。

阿里一面

问的问题不多，也就26分钟的样子

你用过哪些集合类？==太多了，随便说了些 那你说说ArrayList，LinkedList的区别（还是挺简单的，一般用过的都说会）。 说说hashMap是怎样实现的（这个之前看过，顺利回答上。还回答了多线程的问题出现的原因，面试官表示很惊讶的样子） 说说可重入锁

说说view绘制过程和事件分发机制，我大概回答了下。然后面试官又问：onTouch和onTouchEvent是什么区别？如果我重写了ontouch和onClick，它们的调用顺序是怎样的？什么时候会不调用onClick？

handler的是怎样实现的？

由于项目里面用到了picasso，所以最后问了下picasso实现原理。 一面结束，最后面试官居然问我是不是第一次面试== 估计是帮紧张了。不过一面过程中面试官心情还不错，都是笑着问的。

当天晚上接到二面，面试官太累了，约我第二天面试。

阿里二面

二面气氛一直不对，感觉面试官非常严肃，一来就感觉很有压力

自我介绍

操作系统里面线程和进程的区别（挺基础的），接着麻烦就来了；我说完大致区别后，他就问，你说进程里面线程是共享内存的，那么一个进程最大能占多少内存？（懵逼，这是什么意思？考的分页知识？）。然后这里我想了一下，说应该和硬件有关，他继续问，有什么关系？（应该和地址总线有关，当时没想起，他叫我再想想，要是你设计的系统，应该和什么有关，还是没答上）。

你项目中图片是怎么处理的？回答：picasso，顺便说了下picasso原理。然后又问：那么picasso里面有多少个线程来加载图片？要是网络不同，线程数目分别是多少？

布局优化（这里开始说错了一点，然后面试官很生气的样子，自我感觉就要挂了） 项目中有哪些优化？

最后果然挂了（惨痛的经历，不过为后面打下了很好的基础，至少不 怎么紧张了）

然后后面就没有面试了

直到4月腾讯面试

2. 腾讯面经

腾讯是走的正常渠道，到成都现场面试

一面

面试场地是在一个宾馆里面，一对一面试，face to face还是有点紧张的

自我介绍

java多态你了解多少？

你说说重写和重载区别，然后拿了纸笔，手写一个能体现多态的例子

说说java在运行main函数之前做了哪些工作？

这个我居然从启动虚拟机→加载类→初始化类一直说到执行Main

你对大尾小尾了解多少？

我反问：您说的是大小端么？ 他说对，然后我正准备给他解释的时候，他又拿了一张纸：用java写一个判断大小尾的程序

java静态方法能不能被重写？ 答：不能。 问：为什么？ 为什么java静态方法不能调用普通方法？普通方法能调用静态方法？（其实还是实例引用问题） java内存模型和GC机制

其实腾讯面试官感觉都很nice，他称呼我 都用您。感觉怪怪的，而且礼仪非常好。最后面完后，我问我面试得怎样？他说你了解的知识还是挺宽的，然后问了我一句要不要去做游戏？当然要啊！

然后就走了。然后就没有然后了，晚上查状态是不适合。

霸面一面

腾讯面试后，感觉不怎么死心，又跑去长沙霸面了，我一去，HR说移动端基本已经招满了，你可以把简历放在这儿，要是有面试机会的话，我会通知你。然后我心情失落地回去了。

当我刚到住处，刚出电梯，HR就来电话了，叫我去面试

我那个开心啊，把平时20分钟的路程当成10分钟不对跑过去，直接一面。

一面面试官也很nice，还惊讶我从重庆来

Hashmap原理 hashcode和equals还有==的关系 用hashmap实现hashset。。我之前看过的，忘记了。然后按照我的想法回答了。（最后面试官告诉了我该怎样实现==） 内部类访问外部类的变量有什么问题？

android里面onStop和onPause本质区别。什么时候可以存数据？ 两个单链表寻找有没有交点，然后再寻找交点位置 android oom怎么解决 还问了一些项目的问题 其实中间还问了几个算法，忘记是什么了，后面想到了的话会加上的。

二面

二面面试官感觉很牛的样子，一直技术轰炸

告诉我你所直到的所有关于java虚拟机的东西，我说了好久好久。还说了新生代大概什么时候会加入老年代 binder机制 handler原理， Message，loop，messageQueue关系，handler内存泄露问题。 TCP三次握手，用纸画出来 为什么TCP是可靠的，UDP早不可靠的？为什么UDP比TCP快？ 面试官看到了我的项目，然后问了我一个用到的框架的原理，还问了我里面的很多细节，估计是以为我直接看的别人的博客了解到的这些知识，还好我是自己看了源码 算法：几百万个QQ号，找出前100个消费最高的QQ号。直接小顶堆什么的 android四大组件，这里扩展了很多，毕竟非常熟悉，还说了很多坑，很多实现原理（比如activity start原理） 还问了优缺点 （也有一些问题忘记了） 这次面试很久，忘记带水了，出来我直接喝完了一瓶怡宝 这交自我感觉答得不错，然后过了2个小时就收到HR面通知

HR面

自我介绍 项目里面怎么解决安全问题的？好可怕，会技术的HR 有没有女朋友? 家在哪里 有没有亲戚在腾讯？ 我问了下要是过了的话大概会在哪里实习？HR说在深圳，还问我有什么问题么？我说没有，我爸妈也在那边，然后他在我简历上面记了一下。 为什么要学习android? HR面就10多分钟，很快，和我一起面试的还有几个学生，也都是10来分钟，然后HR叫我等结果

然后等啊等，等到现在还没有结果

3. 360面试

360全程视频面试加写代码什么的

一面

写一个adapter，我后面忘记了getView的一个参数，一直在那里想，面试官问我是不是在编译器里面写，我说我在想怎么写。 hashmap原理 java可重入锁 排序算法和稳定性，快排什么时候情况最坏？ 一个获全国奖的项目问了我20分钟，特别是service不被杀死的方法，我说了4种才放过我，还问了我具体实现，特别是在JNI里面实现的时候 项目中界面适配，自定义过view没有？ NFC读卡，这个是我的项目，我说了具体实现，然后就放过我了 我项目中用了google map 和定位，他问怎么定位的？居然问了我具体API，我还说了里面的坑，国产手机阉割了一部分的问题 一面大概1个半小时，头昏脑涨，然后面试官并不放过我，叫我等等。他去叫二面面试官

二面

http协议了解多少，说说里面的协议头部有哪些字段？ https了解多少？为什么百度全部都用了https包括首页 散列表的基础知识，里面也问了hashmap（可见hashmap重要性） 项目问题，几个项目都问了，什么分工啊什么的 问了我很多项目中开发的问题，还好基本都答出来了，二面基础知识基本没多少，都是项目问题 二面接近一个半小时，还好在寝室面试，边面边喝水，二面脑袋都是糊的

二面完后，10分钟打电话通知一周内有HR面

HR面

HR面的时候，我正在火车上，HR说只有15分钟，我说当场面了，因为我那个时候正停在一个大站里面，要停半个小时

自我介绍，问了我所有项目的分工问题和设计等问题，好几个项目，这里就花了接近20分钟，然后火车开走了，然后大家都知道，悲催了，没信号

等到我有信号的时候，再给HR打电话约好第二天继续面。 第二天

继续项目分工 中兴实习情况？为什么最后没留下？（要读书啊） 开发的一些规范 投了XX公司和XX公司没有？为什么没投XX公司？哎，这里太年轻别坑了 怎么看待3Q大战（大姐，这个我怎么来说呢？） 问了我实习时间，希望实习的地点，希望做哪方面？ 你觉得你一面和二面哪一面成绩更好？每一面大概多少分 优缺点 每个问题都问了很久，因为每个过后都接着往下问了的。整个HR面都1小时13分钟，累啊！！ 说好的15分钟呢

然后N天后，收到360拒信。

4. 今日头条

今日头条也是我唯一过的公司，一面还好，二面全程技术轰炸，HR面聊得挺好，虽然有点短

今日头条我是内推的，N天后给我发邮件和电话约面试，本来是北京面试的，结果去不了，就电话面试了。

两次技术面试也是接近2个小时 。一面面试官面完后，叫我去吃饭，过会儿继续面，天真的我以为已经二面了，然而并不是，这个时候面试官还是建议去北京面试，过的机会大些，这个时候哪还有心情吃饭，一直等面试官的电话，结果继续面的时候直接写了一个代码就OK了，代码是在一个矩阵是查找有没有某个数，矩阵从左到右依次增大（忘记是增大还是减小了），从上到下也一样。由于电脑问题，我还是翻墙去写代码的，写代码的时候，由于网不稳定，还经常断

写完直接叫我等二面，过了会儿二面面试官马上来了电话

二面面试官感觉很随和，从Java的用法问到了虚拟机，问到了操作系统，最后深入问到了一个编译原理。还问了一些C语言的东西。 还问了排序。 然后可能是因为我所有东西是自学的，面试官在问之前都问了我了解不，不了解就重新换一个。还问了一些图论最短路径问题（还好面试前不久做了一个比赛，华为的未来寻路，就是最短路径问题），这个答得还行，说了一些经典算法，还有一些只能算法，还有一些改进等。

然后就是我项目中的问题，因为我用了rxjava，picasso，retrofit等开源项目，所以面试官问了我retrofit是如何处理注解的，我直接讲了源码，其过我博客里面也写过了这个，然后面试官可能发现我了解，就直接跳过了这个。然后问了我rxjava的东西，我结合博客看了部分源码，还问了rxjava里面用了大量的<? super T>这些，是什么意思。

过后问我了解github上的一些开源项目不，我说了解一些，然后就是butterknife了，然后我回答错了，以为是注解+反射。后面挂电话后，找了时间分析了源码，还真不是反射==butterknife分析在这里。

今日头条感觉是我面试问题水平最高的，不再局限于基础知识，问了很多很深入的东西。感觉面试官都是根据我具体情况问的，随手丢出问题，直到我回答不上。总以为二面会挂。

结果在某天中午接到了HR的电话，由于那个时候有事，重新约了时间== 当时都有HR面恐惧症了，因为前面两次HR面后都没消息了==

HR面

时间很短，几个问题而已，回答完就叫我等结果，说5月中旬会出结果。

国内一线互联网公司内部面试题库

以下面试题来自于百度、小米、乐视、美团、58、猎豹、360、新浪、搜狐内部题库

熟悉本文中列出的知识点会大大增加通过前两轮技术面试的几率。

GitHub：https://github.com/JackyAndroid/AndroidInterview-Q-A

GitBook：https://www.gitbook.com/book/jackyandroid/androidinterview/details

掘金: https://gold.xitu.io/user/562dc7cc60b20fc9817962a2

目录

- Touch事件传递流程-小米
- view绘制流程-百度
- 多线程-360
- 线程同步-百度
- 什么情况导致内存泄漏-美团
- ANR定位和修正
- 什么情况导致oom-乐视-美团
- Android Service与Activity之间通信的几种方式
- Android各个版本API的区别
- Android代码中实现WAP方式联网-360
- 如何保证service在后台不被kill
- Requestlayout，onlayout，onDraw，DrawChild区别与联系-猎豹
- invalidate()和postInvalidate()的区别及使用-百度和postInvalidate()的区别及使用-百度)
- Android动画框架实现原理
- Android为每个应用程序分配的内存大小是多少？-美团
- Android View刷新机制-百度-美团
- LinearLayout对比RelativeLayout-百度
- 优化自定义view百度-乐视-小米
- ContentProvider-乐视
- fragment生命周期
- volley解析-美团-乐视
- Android Glide源码解析
- Android 设计模式
- 架构设计-搜狐
- Android属性动画特性-乐视-小米
- 专题
- 性能优化
- 架构分析
- 阿里巴巴
- 腾讯

java

接口的意义-百度

规范、扩展、回调、通知机制

抽象类的意义-乐视

为其子类提供一个公共的类型，封装子类中得重复内容 定义抽象方法，子类虽然有不同的实现 但是定义是一致的

内部类的作用-乐视

1. 内部类可以用多个实例，每个实例都有自己的状态信息，并且与其他外围对象的信息相互独立
2. 在单个外围类中，可以让多个内部类以不同的方式实现同一个接口，或者继承同一个类
3. 创建内部类对象的时刻并不依赖于外围类对象的创建
4. 内部类并没有令人迷惑的“is-a”关系，他就是一个独立的实体
5. 内部类提供了更好的封装，除了该外围类，其他类都不能访问

父类的静态方法能否被子类重写-猎豹

不能，子类继承父类后，用相同的静态方法和非静态方法，这时非静态方法覆盖父类中的方法（即方法重写），父类的该静态方法被隐藏（如果对象是父类则调用该隐藏的方法），另外子类可继承父类的静态与非静态方法，至于方法重载我觉得它其中一要素就是在同一类中，不能说父类中的什么方法与子类里的什么方法是方法重载的体现

java排序算法-美团

http://blog.csdn.net/qy1387/article/details/7752973

列举java的集合和继承关系-百度-美团

java虚拟机的特性-百度-乐视

Java语言的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用模式Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

哪些情况下的对象会被垃圾回收机制处理掉-美团-小米

Java 垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性。

进程和线程的区别-猎豹-美团

简而言之，一个程序至少有一个进程，一个进程至少有一个线程。

线程的划分尺度小于进程，使得多线程程序的并发性高。

另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源分配和调度的一个独立单位

线程是进程的一个实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源(如程序计数器，一组寄存器和栈)，但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。

一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行。

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。如果有兴趣深入的话，我建议你们看看《现代操作系统》或者《操作系统的设计与实现》。对就个问题说得比较清楚。

java中==和equals和hashCode的区别-乐视

http://blog.csdn.net/tiantiandjava/article/details/46988461

常见的排序算法时间复杂度-小米

排序法	最差时间分析	平均时间复杂度	稳定度	空间复杂度
冒泡排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
快速排序	$O(n^2)$	$O(nlog_2n)$	不稳定	$O(log_2n)$~O(n)
选择排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
二叉树排序	$O(n^2)$	O(nlog2n)	不一顶	O(n)
插入排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	不稳定	O(1)
希尔排序	O	O	不稳定	O(1)

HashMap的实现原理-美团

1.HashMap概述：

HashMap是基于哈希表的Map接口的非同步实现。此实现提供所有可选的映射操作，并允许使用null值和null键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

2.HashMap的数据结构：

在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap也不例外。HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

从上图中可以看出，HashMap底层就是一个数组结构，数组中的每一项又是一个链表。当新建一个HashMap的时候，就会初始化一个数组。

状态机

http://www.jdon.com/designpatterns/designpattern_State.htm

int-char-long各占多少字节数

类	位数	字节数
byte	8	1

short	16	2
int	32	4
long	64	8
float	32	4
double	64	8
char	16	2

int与integer的区别

http://www.cnblogs.com/shenliang123/archive/2011/10/27/2226903.html

string-stringbuffer-stringbuilder区别-小米-乐视-百度

String 字符串常量

StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

简要的说， String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象， 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用String ， 因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后，JVM 的 GC 就会开始工作，那速度是一定会相当慢的。

而如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuffer ，特别是字符串对象经常改变的情况下。而在某些特别情况下， String 对象的字符串拼接其实是被 JVM 解释成了 StringBuffer 对象的拼接，所以这些时候 String 对象的速度并不会比 StringBuffer 对象慢，而特别是以下的字符串对象生成中， String 效率是远要比 StringBuffer 快的：

```
String S1 = "This is only a" + "simple" + " test";
StringBuffer Sb = new StringBuffer("This is only a").append("simple").append("test");
```

你会很惊讶的发现，生成 String S1 对象的速度简直太快了，而这个时候 StringBuffer 居然速度上根本一点都不占优势。其实这是 JVM 的一个把戏，在 JVM 眼里，这个 `String S1 = “This is only a” + “ simple” + “test”;` 其实就是：`String S1 = “This is only a simple test”;` 所以当然不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的 String 对象的话，速度就没那么快了，譬如：

```
String S2 = “This is only a”;
String S3 = “ simple”;
String S4 = “ test”;
String S1 = S2 +S3 + S4;
```

这时候 JVM 会规规矩矩的按照原来的方式去做

在大部分情况下 StringBuffer 快于 String

StringBuffer

Java.lang.StringBuffer线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。

可将字符串缓冲区安全地用于多个线程。可以在必要时对这些方法进行同步，因此任意特定实例上的所有操作就好像是以串行顺序发生的，该顺序与所涉及的每个线程进行的方法调用顺序一致。

StringBuffer 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串缓冲区中。append 方法始终将这些字符添加到缓冲区的末端；而 insert 方法则在指定的点添加字符。

例如，如果 z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。

在大部分情况下 StringBuilder 快于 StringBuffer

java.lang.StringBuilder

java.lang.StringBuilder一个可变的字符序列是5.0新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同

java多态-乐视

Java多态性理解

Java中多态性的实现

什么是多态

面向对象的三大特性：封装、继承、多态。从一定角度来看，封装和继承几乎都是为多态而准备的。这是我们最后一个概念，也是最重要的知识点。

多态的定义：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

实现多态的技术称为：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实 际类型，根据其实际的类型调用其相应的方法。

多态的作用：消除类型之间的耦合关系。

现实中，关于多态的例子不胜枚举。比方说按下 F1 键这个动作，如果当前在 Flash 界面下弹出的就是 AS 3 的帮助文档；如果当前在 Word 下弹出的就是 Word 帮助；在 Windows 下弹出的就是 Windows 帮助和支持。同一个事件发生在不同的对象上会产生不同的结果。

下面是多态存在的三个必要条件，要求大家做梦时都能背出来!

多态存在的三个必要条件

- 要有继承
- 要有重写
- 父类引用指向子类对象

多态的好处:

1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。

2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图8.3 所示。图中超类Shape规定了两个实现多态的接口方法，computeArea()以及computeVolume()。子类，如Circle和Sphere为了实现多态，完善或者覆盖这两个接口方法。

4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。

5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

Java中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。

什么导致线程阻塞-58-美团

线程的阻塞

为了解决对共享存储区的访问冲突，Java 引入了同步机制，现在让我们来考察多个线程对共享资源的访问，显然同步机制已经不够了，因为在任意时刻所要求的资源不一定已经准备好了被访问，反过来，同一时刻准备好了的资源也可能不止一个。为了解决这种情况下的访问控制问题，Java 引入了对阻塞机制的支持

阻塞指的是暂停一个线程的执行以等待某个条件发生（如某资源就绪），学过操作系统的同学对它一定已经很熟悉了。Java 提供了大量方法来支持阻塞，下面让我们逐一分析。

1.sleep() 方法：sleep() 允许 指定以毫秒为单位的一段时间作为参数，它使得线程在指定的时间内进入阻塞状态，不能得到CPU时间，指定的时间一过，线程重新进入可执行状态。

典型地，sleep() 被用在等待某个资源就绪的情形：测试发现条件不满足后，让线程阻塞一段时间后重新测试，直到条件满足为止。

2.suspend() 和 resume() 方法：两个方法配套使用，suspend()使得线程进入阻塞状态，并且不会自动恢复，必须其对应的resume() 被调用，才能使得线程重新进入可执行状态。典型地，suspend() 和 resume() 被用在等待另一个线程产生的结果的情形：测试发现结果还没有产生后，让线程阻塞，另一个线程产生了结果后，调用 resume() 使其恢复。

3.yield() 方法：yield() 使得线程放弃当前分得的 CPU 时间，但是不使线程阻塞，即线程仍处于可执行状态，随时可能再次分得 CPU 时间。调用 yield() 的效果等价于调度程序认为该线程已执行了足够的时间从而转到另一个线程.

4.wait() 和 notify() 方法：两个方法配套使用，wait() 使得线程进入阻塞状态，它有两种形式，一种允许 指定以毫秒为单位的一段时间作为参数，另一种没有参数，前者当对应的 notify() 被调用或者超出指定时间时线程重新进入可执行状态，后者则必须对应的 notify() 被调用.

初看起来它们与 suspend() 和 resume() 方法对没有什么分别，但是事实上它们是截然不同的。区别的核心在于，前面叙述的所有方法，阻塞时都不会释放占用的锁（如果占用了的话），而这一对方法则相反。

上述的核心区别导致了一系列的细节上的区别。

首先，前面叙述的所有方法都隶属于 Thread 类，但是这一对却直接隶属于 Object 类，也就是说，所有对象都拥有这一对方法。初看起来这十分不可思议，但是实际上却是很自然的，因为这一对方法阻塞时要释放占用的锁，而锁是任何对象都具有的，调用任意对象的 wait() 方法导致线程阻塞，并且该对象上的锁被释放。而调用 任意对象的notify()方法则导致因调用该对象的 wait() 方法而阻塞的线程中随机选择的一个解除阻塞（但要等到获得锁后才真正可执行）。

其次，前面叙述的所有方法都可在任何位置调用，但是这一对方法却必须在 synchronized 方法或块中调用，理由也很简单，只有在synchronized 方法或块中当前线程才占有锁，才有锁可以释放。同样的道理，调用这一对方法的对象上的锁必须为当前线程所拥有，这样才有锁可以释放。因此，这一对方法调用必须放置在这样的 synchronized 方法或块中，该方法或块的上锁对象就是调用这一对方法的对象。若不满足这一条件，则程序虽然仍能编译，但在运行时会出现IllegalMonitorStateException 异常。

wait() 和 notify() 方法的上述特性决定了它们经常和synchronized 方法或块一起使用，将它们和操作系统的进程间通信机制作一个比较就会发现它们的相似性：synchronized方法或块提供了类似于操作系统原语的功能，它们的执行不会受到多线程机制的干扰，而这一对方法则相当于 block 和wakeup 原语（这一对方法均声明为 synchronized）。它们的结合使得我们可以实现操作系统上一系列精妙的进程间通信的算法（如信号量算法），并用于解决各种复杂的线程间通信问题。

关于 wait() 和 notify() 方法最后再说明两点：

第一：调用 notify() 方法导致解除阻塞的线程是从因调用该对象的 wait() 方法而阻塞的线程中随机选取的，我们无法预料哪一个线程将会被选择，所以编程时要特别小心，避免因这种不确定性而产生问题。

第二：除了 notify()，还有一个方法 notifyAll() 也可起到类似作用，唯一的区别在于，调用 notifyAll() 方法将把因调用该对象的 wait() 方法而阻塞的所有线程一次性全部解除阻塞。当然，只有获得锁的那一个线程才能进入可执行状态。

谈到阻塞，就不能不谈一谈死锁，略一分析就能发现，suspend() 方法和不指定超时期限的 wait() 方法的调用都可能产生死锁。遗憾的是，Java 并不在语言级别上支持死锁的避免，我们在编程中必须小心地避免死锁。

以上我们对 Java 中实现线程阻塞的各种方法作了一番分析，我们重点分析了 wait() 和 notify() 方法，因为它们的功能最强大，使用也最灵活，但是这也导致了它们的效率较低，较容易出错。实际使用中我们应该灵活使用各种方法，以便更好地达到我们的目的。

抽象类接口区别-360

1.默认的方法实现

抽象类可以有默认的方法实现完全是抽象的。接口根本不存在方法的实现

2.实现

子类使用extends关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。

子类使用关键字implements来实现接口。它需要提供接口中所有声明的方法的实现

3.构造器

抽象类可以有构造器

接口不能有构造器

4.与正常Java类的区别

除了你不能实例化抽象类之外，它和普通Java类没有任何区

接口是完全不同的类型

5.访问修饰符

抽象方法可以有public、protected和default这些修饰符

接口方法默认修饰符是public。你不可以使用其它修饰符。

6.main方法

抽象方法可以有main方法并且我们可以运行它

接口没有main方法，因此我们不能运行它。

7.多继承

抽象类在java语言中所表示的是一种继承关系，一个子类只能存在一个父类，但是可以存在多个接口。

8.速度

它比接口速度要快

接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。

9.添加新方法

如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。

如果你往接口中添加方法，那么你必须改变实现该接口的类。

容器类之间的区别-乐视-美团

http://www.cnblogs.com/yuanermen/archive/2009/08/05/1539917.html
http://alexyyek.github.io/2015/04/06/Collection http://tianmaying.com/tutorial/java_collection

内部类

http://www.cnblogs.com/chenssy/p/3388487.html

hashmap和hashtable的区别-乐视-小米

http://www.233.com/ncre2/JAVA/jichu/20100717/084230917.html

ArrayMap对比HashMap

http://lvable.com/?p=217

Android

如何导入外部数据库

把原数据库包括在项目源码的 res/raw

android系统下数据库应该存放在 /data/data/packagename/ 目录下，所以我们需要做的是把已有的数据库传入那个目录下.操作方法是用FileInputStream读取原数据库，再用FileOutputStream把读取到的东西写入到那个目录.

本地广播和全局广播有什么差别

因广播数据在本应用范围内传播，不用担心隐私数据泄露的问题。 不用担心别的应用伪造广播，造成安全隐患。相比在系统内发送全局广播，它更高效。

intentService作用是什么，AIDL解决了什么问题-小米

生成一个默认的且与主线程互相独立的工作者线程来执行所有传送至onStartCommand() 方法的Intetnt。

生成一个工作队列来传送Intent对象给你的onHandleIntent()方法，同一时刻只传送一个Intent对象，这样一来，你就不必担心多线程的问题。在所有的请求(Intent)都被执行完以后会自动停止服务，所以，你不需要自己去调用stopSelf()方法来停止。

该服务提供了一个onBind()方法的默认实现，它返回null

提供了一个onStartCommand()方法的默认实现，它将Intent先传送至工作队列，然后从工作队列中每次取出一个传送至onHandleIntent()方法，在该方法中对Intent对相应的处理。

AIDL (Android Interface Definition Language) 是一种IDL 语言，用于生成可以在Android设备上两个进程之间进行进程间通信(interprocess communication， IPC)的代码。如果在一个进程中（例如Activity）要调用另一个进程中（例如Service）对象的操作，就可以使用AIDL生成可序列化的参数。

AIDL IPC机制是面向接口的，像COM或Corba一样，但是更加轻量级。它是使用代理类在客户端和实现端传递数据。

Activity/Window/View三者的差别，fragment的特点-360

Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图） LayoutInflater像剪刀，Xml配置像窗花图纸。

1. 在Activity中调用attach，创建了一个Window
2. 创建的window是其子类PhoneWindow，在attach中创建PhoneWindow
3. 在Activity中调用setContentView(R.layout.xxx)
4. 其中实际上是调用的getWindow().setContentView()
5. 调用PhoneWindow中的setContentView方法
6. 创建ParentView：作为ViewGroup的子类，实际是创建的DecorView(作为FramLayout的子类）
7. 将指定的R.layout.xxx进行填充，通过布局填充器进行填充[其中的parent指的就是DecorView]
8. 调用到ViewGroup
9. 调用ViewGroup的removeAllView()，先将所有的view移除掉
10. 添加新的view：addView()

fragment 特点

- Fragment可以作为Activity界面的一部分组成出现
- 可以在一个Activity中同时出现多个Fragment，并且一个Fragment也可以在多个Activity中使用
- 在Activity运行过程中，可以添加、移除或者替换Fragment
- Fragment可以响应自己的输入事件，并且有自己的生命周期，它们的生命周期会受宿主Activity的生命周期影响

描述一次网络请求的流程-新浪

Handler，Thread和HandlerThread的差别-小米

http://blog.csdn.net/guolin_blog/article/details/9991569

http://droidyue.com/blog/2015/11/08/make-use-of-handlerthread/

从Android中Thread（java.lang.Thread → java.lang.Object）描述可以看出，Android的Thread没有对Java的Thread做任何封装，但是Android提供了一个继承自Thread的类HandlerThread（android.os.HandlerThread → java.lang.Thread），这个类对Java的Thread做了很多便利Android系统的封装。

android.os.Handler可以通过Looper对象实例化，并运行于另外的线程中，Android提供了让Handler运行于其它线程的线程实现，也是就HandlerThread。HandlerThread对象start后可以获得其Looper对象，并且使用这个Looper对象实例Handler。

低版本SDK实现高版本api-小米

自己实现或@TargetApi annotation

Ubuntu编译安卓系统-百度

1. 进入源码根目录
2. build/envsetup.sh
3. lunch
4. full(编译全部)
5. userdebug(选择编译版本)
6. make -j8(开启8个线程编译)

LaunchMode应用场景-百度-小米-乐视

standard，创建一个新的Activity。

singleTop，栈顶不是该类型的Activity，创建一个新的Activity。否则，onNewIntent。

singleTask，回退栈中没有该类型的Activity，创建Activity，否则，onNewIntent+ClearTop。

注意:

1. 设置了singleTask启动模式的Activity，它在启动的时候，会先在系统中查找属性值affinity等于它的属性值taskAffinity的Task存在； 如果存在这样的Task，它就会在这个Task中启动，否则就会在新的任务栈中启动。因此， 如果我们想要设置了singleTask启动模式的Activity在新的任务中启动，就要为它设置一个独立的taskAffinity属性值。
2. 如果设置了singleTask启动模式的Activity不是在新的任务中启动时，它会在已有的任务中查看是否已经存在相应的Activity实例， 如果存在，就会把位于这个Activity实例上面的Activity全部结束掉，即最终这个Activity 实例会位于任务的Stack顶端中。
3. 在一个任务栈中只有一个singleTask启动模式的Activity存在。他的上面可以有其他的Activity。这点与singleInstance是有区别的。

singleInstance，回退栈中，只有这一个Activity，没有其他Activity。

singleTop适合接收通知启动的内容显示页面。

例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。

singleTask适合作为程序入口点。

例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。

singleInstance应用场景:

闹铃的响铃界面。 你以前设置了一个闹铃：上午6点。在上午5点58分，你启动了闹铃设置界面，并按 Home 键回桌面；在上午5点59分时，你在微信和朋友聊天；在6点时，闹铃响了，并且弹出了一个对话框形式的 Activity(名为 AlarmAlertActivity) 提示你到6点了(这个 Activity 就是以 SingleInstance 加载模式打开的)，你按返回键，回到的是微信的聊天界面，这是因为 AlarmAlertActivity 所在的 Task 的栈只有他一个元素， 因此退出之后这个 Task 的栈空了。如果是以 SingleTask 打开 AlarmAlertActivity，那么当闹铃响了的时候，按返回键应该进入闹铃设置界面。

Touch事件传递流程-小米

Android-三张图搞定Touch事件传递机制

View绘制流程-百度

公共技术点之 View 绘制流程

多线程-360

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)，View.postDelay(Runnable,long)
- Handler
- AsyncTask

线程同步-百度

Java基础笔记 – 线程同步问题 解决同步问题的方法 synchronized方法 同步代码块

Android线程间交互（Java synchronized & Android Handler）

单例

```
public class Singleton{
private volatile static Singleton mSingleton;
private Singleton(){
}
public static Singleton getInstance(){
  if(mSingleton == null){\\A
    synchronized(Singleton.class){\\C
     if(mSingleton == null)
      mSingleton = new Singleton();\\B
      }
    }
    return mSingleton;
  }
}
```

什么情况导致内存泄漏-美团

1.资源对象没关闭造成的内存泄漏

描述：资源性对象比如(Cursor，File文件等)往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们，以便它们的缓冲及时回收内存。它们的缓冲不仅存在于 java虚拟机内，还存在于java虚拟机外。如果我们仅仅是把它的引用设置为null，而不关闭它们，往往会造成内存泄漏。因为有些资源性对象，比如 SQLiteCursor(在析构函数finalize()，如果我们没有关闭它，它自己会调close()关闭)，如果我们没有关闭它，系统在回收它时也会关闭它，但是这样的效率太低了。因此对于资源性对象在不使用的时候，应该调用它的close()函数，将其关闭掉，然后才置为null.在我们的程序退出时一定要确保我们的资源性对象已经关闭。

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，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险。

2.构造Adapter时，没有使用缓存的convertView

描述：以构造ListView的BaseAdapter为例，在BaseAdapter中提供了方法：`public View getView(int position, ViewconvertView, ViewGroup parent)` 来向ListView提供每一个item所需要的view对象。初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的 view对象，同时ListView会将这些view对象缓存起来。当向上滚动ListView时，原先位于最上面的list item的view对象会被回收，然后被用来构造新出现的最下面的list item。这个构造过程就是由getView()方法完成的，getView()的第二个形参View convertView就是被缓存起来的list item的view对象(初始化时缓存中没有view对象则convertView是null)。由此可以看出，如果我们不去使用 convertView，而是每次都在getView()中重新实例化一个View对象的话，即浪费资源也浪费时间，也会使得内存占用越来越大。 ListView回收list item的view对象的过程可以查看: android.widget.AbsListView.java → voidaddScrapView(View scrap) 方法。 示例代码:

```
public View getView(int position, ViewconvertView, ViewGroup parent) {
View view = new Xxx(...);
...
return view;
}
```

修正示例代码:

```
public View getView(int position, ViewconvertView, ViewGroup parent) {
View view = null;
if (convertView != null) {
view = convertView;
populate(view, getItem(position));
...
} else {
view = new Xxx(...);
...
}
return view;
}
```

3.Bitmap对象不在使用时调用recycle()释放内存

描述：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，当它不在被使用的时候，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。可以看一下代码中的注释:

```
/**
 * Free up the memory associated with thisbitmap's pixels, and mark the
 * bitmap as "dead", meaning itwill throw an exception if getPixels() or
 * setPixels() is called, and will drawnothing. This operation cannot be
 * reversed, so it should only be called ifyou are sure there are no
 * further uses for the bitmap. This is anadvanced call, and normally need
 * not be called, since the normal GCprocess will free up this memory when
 * there are no more references to thisbitmap.
 */
```

4.试着使用关于application的context来替代和activity相关的context

这是一个很隐晦的内存泄漏的情况。有一种简单的方法来避免context相关的内存泄漏。最显著地一个是避免context逃出他自己的范围之外。使用Application context。这个context的生存周期和你的应用的生存周期一样长，而不是取决于activity的生存周期。如果你想保持一个长期生存的对象，并且这个对象需要一个context,记得使用application对象。你可以通过调用 Context.getApplicationContext() or Activity.getApplication()来获得。更多的请看这篇文章如何避免Android内存泄漏。

5.注册没取消造成的内存泄漏

一些Android程序可能引用我们的Anroid程序的对象(比如注册机制)。即使我们的Android程序已经结束了，但是别的引用程序仍然还有对我们的Android程序的某个对象的引用，泄漏的内存依然不能被垃圾回收。调用registerReceiver后未调用unregisterReceiver。

比如:假设我们希望在锁屏界面(LockScreen)中，监听系统中的电话服务以获取一些信息(如信号强度等)，则可以在LockScreen中定义一个 PhoneStateListener的对象，同时将它注册到TelephonyManager服务中。对于LockScreen对象，当需要显示锁屏界面的时候就会创建一个LockScreen对象，而当锁屏界面消失的时候LockScreen对象就会被释放掉。 但是如果在释放 LockScreen对象的时候忘记取消我们之前注册的PhoneStateListener对象，则会导致LockScreen无法被垃圾回收。如果不断的使锁屏界面显示和消失，则最终会由于大量的LockScreen对象没有办法被回收而引起OutOfMemory，使得system_process 进程挂掉。

虽然有些系统程序，它本身好像是可以自动取消注册的(当然不及时)，但是我们还是应该在我们的程序中明确的取消注册，程序结束时应该把所有的注册都取消掉。

6.集合中对象没清理造成的内存泄漏

我们通常把一些对象的引用加入到了集合中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，这样这个集合就会越来越大。如果这个集合是static的话，那情况就更严重了。

ANR定位和修正

如果开发机器上出现问题，我们可以通过查看/data/anr/traces.txt即可，最新的ANR信息在最开始部分。

- 主线程被IO操作（从4.0之后网络IO不允许在主线程中）阻塞。
- 主线程中存在耗时的计算
- 主线程中错误的操作，比如Thread.wait或者Thread.sleep等 Android系统会监控程序的响应状况，一旦出现下面两种情况，则弹出ANR对话框
- 应用在5秒内未响应用户的输入事件（如按键或者触摸）
- BroadcastReceiver未在10秒内完成相关的处理
- Service在特定的时间内无法处理完成 20秒
- 使用AsyncTask处理耗时IO操作。
- 使用Thread或者HandlerThread时，调用 `Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND)` 设置优先级，否则仍然会降低程序响应，因为默认Thread的优先级和主线程相同。
- 使用Handler处理工作线程结果，而不是使用Thread.wait()或者Thread.sleep()来阻塞主线程。
- Activity的onCreate和onResume回调中尽量避免耗时的代码
- BroadcastReceiver中onReceive代码也要尽量减少耗时，建议使用IntentService处理

什么情况导致oom-乐视-美团

Android内存优化之OOM

- 使用更加轻量的数据结构
- Android里面使用Enum
- Bitmap对象的内存占用
- 更大的图片
- onDraw方法里面执行对象的创建
- StringBuilder

Service与Activity之间通信的几种方式

- 通过Binder对象
- 通过broadcast(广播)的形式

Android各个版本API的区别

http://blog.csdn.net/lijun952048910/article/details/7980562

Android代码中实现WAP方式联网-360

http://blog.csdn.net/asce1885/article/details/7844159

如何保证service在后台不被Kill

一、onStartCommand方法，返回START_STICKY

1.START_STICKY

在运行onStartCommand后service进程被kill后，那将保留在开始状态，但是不保留那些传入的intent。不久后service就会再次尝试重新创建，因为保留在开始状态，在创建 service后将保证调用onstartCommand。如果没有传递任何开始命令给service，那将获取到null的intent。

2.START_NOT_STICKY

在运行onStartCommand后service进程被kill后，并且没有新的intent传递给它。Service将移出开始状态，并且直到新的明显的方法（startService）调用才重新创建。因为如果没有传递任何未决定的intent那么service是不会启动，也就是期间onstartCommand不会接收到任何null的intent。

3.START_REDELIVER_INTENT

在运行onStartCommand后service进程被kill后，系统将会再次启动service，并传入最后一个intent给onstartCommand。直到调用stopSelf(int)才停止传递intent。如果在被kill后还有未处理好的intent，那被kill后服务还是会自动启动。因此onstartCommand不会接收到任何null的intent。

二、提升service优先级

在AndroidManifest.xml文件中对于intent-filter可以通过 `android:priority = "1000"` 这个属性设置最高优先级，1000是最高值，如果数字越小则优先级越低，同时适用于广播。

三、提升service进程优先级

Android中的进程是托管的，当系统进程空间紧张的时候，会依照优先级自动进行进程的回收。Android将进程分为6个等级，它们按优先级顺序由高到低依次是:

1. 前台进程(FOREGROUND_APP)
2. 可视进程(VISIBLE_APP)
3. 次要服务进程(SECONDARY_SERVER)
4. 后台进程 (HIDDEN_APP)
5. 内容供应节点(CONTENT_PROVIDER)
6. 空进程(EMPTY_APP)

当service运行在低内存的环境时，将会kill掉一些存在的进程。因此进程的优先级将会很重要，可以使用 startForeground 将service放到前台状态。这样在低内存时被kill的几率会低一些。

四、onDestroy方法里重启service

service +broadcast 方式，就是当service走ondestory的时候，发送一个自定义的广播，当收到广播的时候，重新启动service；

五、Application加上Persistent属性

六、监听系统广播判断Service状态

通过系统的一些广播，比如：手机重启、界面唤醒、应用状态改变等等监听并捕获到，然后判断我们的Service是否还存活，别忘记加权限啊。

Requestlayout，onlayout，onDraw，DrawChild区别与联系-猎豹

requestLayout()方法 ： 会导致调用measure()过程 和 layout()过程 。 将会根据标志位判断是否需要ondraw

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)

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)

drawChild()去重新回调每个子视图的draw()方法

invalidate()和postInvalidate()的区别及使用-百度

http://blog.csdn.net/mars2639/article/details/6650876

Android动画框架实现原理

Animation框架定义了透明度，旋转，缩放和位移几种常见的动画，而且控制的是整个View，实现原理是每次绘制视图时View所在的ViewGroup中的drawChild函数获取该View的Animation的Transformation值，然后调用 `canvas.concat(transformToApply.getMatrix())`，通过矩阵运算完成动画帧，如果动画没有完成，继续调用invalidate()函数，启动下次绘制来驱动动画，动画过程中的帧之间间隙时间是绘制函数所消耗的时间，可能会导致动画消耗比较多的CPU资源，最重要的是，动画改变的只是显示，并不能相应事件。

Android为每个应用程序分配的内存大小是多少-美团

android程序内存一般限制在16M，也有的是24M

View刷新机制-百度-美团

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：

mView.draw()开始绘制，draw()方法实现的功能如下：

1. 绘制该View的背景
2. 为显示渐变框做一些准备操作(见5，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。

LinearLayout和RelativeLayout性能对比-百度

1. RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure
2. RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。

3. 在不影响层级深度的情况下，使用LinearLayout和FrameLayout而不是RelativeLayout。

最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

优化自定义view百度-乐视-小米

为了加速你的view，对于频繁调用的方法，需要尽量减少不必要的代码。先从onDraw开始，需要特别注意不应该在这里做内存分配的事情，因为它会导致GC，从而导致卡顿。在初始化或者动画间隙期间做分配内存的动作。不要在动画正在执行的时候做内存分配的事情。

你还需要尽可能的减少onDraw被调用的次数，大多数时候导致onDraw都是因为调用了invalidate().因此请尽量减少调用invaildate()的次数。如果可能的话，尽量调用含有4个参数的invalidate()方法而不是没有参数的invalidate()。没有参数的invalidate会强制重绘整个view。

另外一个非常耗时的操作是请求layout。任何时候执行requestLayout()，会使得Android UI系统去遍历整个View的层级来计算出每一个view的大小。如果找到有冲突的值，它会需要重新计算好几次。另外需要尽量保持View的层级是扁平化的，这样对提高效率很有帮助。

如果你有一个复杂的UI，你应该考虑写一个自定义的ViewGroup来执行他的layout操作。与内置的view不同，自定义的view可以使得程序仅仅测量这一部分，这避免了遍历整个view的层级结构来计算大小。这个PieChart 例子展示了如何继承ViewGroup作为自定义view的一部分。PieChart 有子views，但是它从来不测量它们。而是根据他自身的layout法则，直接设置它们的大小。

ContentProvider-乐视

http://blog.csdn.net/coder_pig/article/details/47858489

Fragment生命周期

volley解析-美团-乐视

http://a.codekk.com/detail/Android/grumoon/Volley%20%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90

Glide源码解析

http://www.lightskystreet.com/2015/10/12/glide_source_analysis/ http://frodoking.github.io/2015/10/10/android-glide/

Android设计模式

http://blog.csdn.net/bboyfeiyu/article/details/44563871

架构设计-搜狐

http://www.tianmaying.com/tutorial/AndroidMVC

Android属性动画特性-乐视-小米

如果你的需求中只需要对View进行移动、缩放、旋转和淡入淡出操作，那么补间动画确实已经足够健全了。但是很显然，这些功能是不足以覆盖所有的场景的，一旦我们的需求超出了移动、缩放、旋转和淡入淡出这四种对View的操作，那么补间动画就不能再帮我们忙了，也就是说它在功能和可扩展方面都有相当大的局限性，那么下面我们就来看看补间动画所不能胜任的场景。

注意上面我在介绍补间动画的时候都有使用“对View进行操作”这样的描述，没错，补间动画是只能够作用在View上的。也就是说，我们可以对一个Button、TextView、甚至是LinearLayout、或者其它任何继承自View的组件进行动画操作，但是如果我们想要对一个非View的对象进行动画操作，抱歉，补间动画就帮不上忙了。可能有的朋友会感到不能理解，我怎么会需要对一个非View的对象进行动画操作呢？这里我举一个简单的例子，比如说我们有一个自定义的View，在这个View当中有一个Point对象用于管理坐标，然后在onDraw()方法当中就是根据这个Point对象的坐标值来进行绘制的。也就是说，如果我们可以对Point对象进行动画操作，那么整个自定义View的动画效果就有了。显然，补间动画是不具备这个功能的，这是它的第一个缺陷。

然后补间动画还有一个缺陷，就是它只能够实现移动、缩放、旋转和淡入淡出这四种动画操作，那如果我们希望可以对View的背景色进行动态地改变呢？很遗憾，我们只能靠自己去实现了。说白了，之前的补间动画机制就是使用硬编码的方式来完成的，功能限定死就是这些，基本上没有任何扩展性可言。

最后，补间动画还有一个致命的缺陷，就是它只是改变了View的显示效果而已，而不会真正去改变View的属性。什么意思呢？比如说，现在屏幕的左上角有一个按钮，然后我们通过补间动画将它移动到了屏幕的右下角，现在你可以去尝试点击一下这个按钮，点击事件是绝对不会触发的，因为实际上这个按钮还是停留在屏幕的左上角，只不过补间动画将这个按钮绘制到了屏幕的右下角而已。

专题

性能优化

Android性能优化典范 - 第1季

1. **Render Performance** Android系统每隔16ms发出VSYNC信号，触发对UI进行渲染，如果每次渲染都成功，这样就能够达到流畅的画面所需要的60fps，为了能够实现60fps，这意味着程序的大多数操作都必须在16ms内完成。我们可以通过一些工具来定位问题，比如可以使用HierarchyViewer来查找Activity中的布局是否过于复杂，也可以使用手机设置里面的开发者选项，打开Show GPU Overdraw等选项进行观察。你还可以使用TraceView来观察CPU的执行情况，更加快捷的找到性能瓶颈。

2. **Understanding Overdraw** Overdraw(过度绘制)描述的是屏幕上的某个像素在同一帧的时间内被绘制了多次。在多层次的UI结构里面，如果不可见的UI也在做绘制的操作，这就会导致某些像素区域被绘制了多次。这就浪费大量的CPU以及GPU资源。Overdraw有时候是因为你的UI布局存在大量重叠的部分，还有的时候是因为非必须的重叠背景。例如某个Activity有一个背景，然后里面的Layout又有自己的背景，同时子View又分别有自己的背景。仅仅是通过移除非必须的背景图片，这就能够减少大量的红色Overdraw区域，增加蓝色区域的占比。这一措施能够显著提升程序性能。

3. **Understanding VSYNC** Refresh Rate：代表了屏幕在一秒内刷新屏幕的次数，这取决于硬件的固定参数，例如60Hz。Frame Rate：代表了GPU在一秒内绘制操作的帧数，例如30fps，60fps。通常来说，帧率超过刷新频率只是一种理想的状况，在超过60fps的情况下，GPU所产生的帧数据会因为等待VSYNC的刷新信息而被Hold住，这样能够保持每次刷新都有实际的新的数据可以显示。但是我们遇到更多的情况是帧率小于刷新频率。

4. **Tool:Profile GPU Rendering** 性能问题如此的麻烦，幸好我们可以有工具来进行调试。打开手机里面的开发者选项，选择Profile GPU Rendering，选中On screen as bars的选项。

5. **Why 60fps?** 我们通常都会提到60fps与16ms，可是知道为何会是以程序是否达到60fps来作为App性能的衡量标准吗？这是因为人眼与大脑之间的协作无法感知超过60fps的画面更新。开发app的性能目标就是保持60fps，这意味着每一帧你只有16ms=1000/60的时间来处理所有的任务。

6. **Android， UI and the GPU** 在Android里面那些由主题所提供的资源，例如Bitmaps，Drawables都是一起打包到统一的Texture纹理当中，然后再传递到GPU里面，这意味着每次你需要使用这些资源的时候，都是直接从纹理里面进行获取渲染的。当然随着UI组件的越来越丰富，有了更多演变的形态。例如显示图片的时候，需要先经过CPU的计算加载到内存中，然后传递给GPU进行渲染。文字的显示更加复杂，需要先经过CPU换算成纹理，然后再交给GPU进行渲染，回到CPU绘制单个字符的时候，再重新引用经过GPU渲染的内容。动画则是一个更加复杂的操作流程。为了能够使得App流畅，我们需要在每一帧16ms以内处理完所有的CPU与GPU计算，绘制，渲染等等操作。

7. **Invalidations， Layouts， and Performance** 任何时候View中的绘制内容发生变化时，都会重新执行创建DisplayList，渲染DisplayList，更新到屏幕上等一系列操作。这个流程的表现性能取决于你的View的复杂程度，View的状态变化以及渲染管道的执行性能。举个例子，假设某个Button的大小需要增大到目前的两倍，在增大Button大小之前，需要通过父View重新计算并摆放其他子View的位置。修改View的大小会触发整个HierarcyView的重新计算大小的操作。如果是修改View的位置则会触发HierarchView重新计算其他View的位置。如果布局很复杂，这就会很容易导致严重的性能问题。我们需要尽量减少Overdraw。

8. **Overdraw， Cliprect， QuickReject** 我们可以通过canvas.clipRect()来帮助系统识别那些可见的区域。这个方法可以指定一块矩形区域，只有在这个区域内才会被绘制，其他的区域会被忽视。这个API可以很好的帮助那些有多组重叠组件的自定义View来控制显示的区域。同时clipRect方法还可以帮助节约CPU与GPU资源，在clipRect区域之外的绘制指令都不会被执行，那些部分内容在矩形区域内的组件，仍然会得到绘制。

9. **Memory Churn and performance** 执行GC操作的时候，所有线程的任何操作都会需要暂停，等待GC操作完成之后，其他操作才能够继续运行。Memory Churn内存抖动，内存抖动是因为大量的对象被创建又在短时间内马上被释放。瞬间产生大量的对象会严重占用Young Generation的内存区域，当达到阀值，剩余空间不够的时候，也会触发GC。即使每次分配的对象占用了很少的内存，但是他们叠加在一起会增加Heap的压力，从而触发更多其他类型的GC。这个操作有可能会影响到帧率，并使得用户感知到性能问题。

10. **Garbage Collection in Android** 原始JVM中的GC机制在Android中得到了很大程度上的优化。Android里面是一个三级Generation的内存模型，最近分配的对象会存放在Young Generation区域，当这个对象在这个区域停留的时间达到一定程度，它会被移动到Old Generation，最后到Permanent Generation区域。如果不小心在最小的for循环单元里面执行了创建对象的操作，这将很容易引起GC并导致性能问题。通过Memory Monitor我们可以查看到内存的占用情况，每一次瞬间的内存降低都是因为此时发生了GC操作，如果在短时间内发生大量的内存上涨与降低的事件，这说明很有可能这里有性能问题。我们还可以通过Heap and Allocation Tracker工具来查看此时内存中分配的到底有哪些对象。

11. **Performance Cost of Memory Leaks** 内存泄漏指的是那些程序不再使用的对象无法被GC识别，这样就导致这个对象一直留在内存当中，占用了宝贵的内存空间。显然，这还使得每级Generation的内存区域可用空间变小，GC就会更容易被触发，从而引起性能问题。

12. **Memory Performance** 通常来说，Android对GC做了大量的优化操作，虽然执行GC操作的时候会暂停其他任务，可是大多数情况下，GC操作还是相对很安静并且高效的。但是如果我们对内存的使用不恰当，导致GC频繁执行，这样就会引起不小的性能问题。

13. **Tool - Memory Monitor** Android Studio中的Memory Monitor可以很好的帮助我们查看程序的内存使用情况。

14. **Battery Performance** 我们应该尽量减少唤醒屏幕的次数与持续的时间，使用WakeLock来处理唤醒的问题，能够正确执行唤醒操作并根据设定及时关闭操作进入睡眠状态。某些非必须马上执行的操作，例如上传歌曲，图片处理等，可以等到设备处于充电状态或者电量充足的时候才进行。触发网络请求的操作，每次都会保持无线信号持续一段时间，我们可以把零散的网络请求打包进行一次操作，避免过多的无线信号引起的电量消耗。关于网络请求引起无线信号的电量消耗

15. **Understanding Battery Drain on Android** 使用WakeLock或者JobScheduler唤醒设备处理定时的任务之后，一定要及时让设备回到初始状态。每次唤醒无线信号进行数据传递，都会消耗很多电量，它比WiFi等操作更加的耗电

16. **Battery Drain and WakeLocks** 这正是JobScheduler API所做的事情。它会根据当前的情况与任务，组合出理想的唤醒时间，例如等到正在充电或者连接到WiFi的时候，或者集中任务一起执行。我们可以通过这个API实现很多免费的调度算法。

Android性能优化典范 - 第2季

1. **Battery Drain and Networking** 我们可以有针对性的把请求行为捆绑起来，延迟到某个时刻统一发起请求。这部分主要会涉及到Prefetch(预取)与Compressed(压缩)这两个技术。对于Prefetch的使用，我们需要预先判断用户在此次操作之后，后续零散的请求是否很有可能会马上被触发，可以把后面5分钟有可能会使用到的零散请求都一次集中执行完毕。对于Compressed的使用，在上传与下载数据之前，使用CPU对数据进行压缩与解压，可以很大程度上减少网络传输的时间。

2. **Wear & Sensors** 首先我们需要尽量使用Android平台提供的既有运动数据，而不是自己去实现监听采集数据，因为大多数Android Watch自身记录Sensor数据的行为是有经过做电量优化的。其次在Activity不需要监听某些Sensor数据的时候需要尽快释放监听注册。还有我们需要尽量控制更新的频率，仅仅在需要刷新显示数据的时候才触发获取最新数据的操作。另外我们可以针对Sensor的数据做批量处理，待数据累积一定次数或者某个程度的时候才更新到UI上。最后当Watch与Phone连接起来的时候，可以把某些复杂操作的事情交给Phone来执行，Watch只需要等待返回的结果。

3. **Smooth Android Wear Animation** 在Android里面一个相对操作比较繁重的事情是对Bitmap进行旋转，缩放，裁剪等等。例如在一个圆形的钟表图上，我们把时钟的指针抠出来当做单独的图片进行旋转会比旋转一张完整的圆形图的所形成的帧率要高56%。

4. **Android Wear Data Batching** 仅仅在真正需要刷新界面的时候才发出请求，尽量把计算复杂操作的任务交给Phone来处理，Phone仅仅在数据发生变化的时候才通知到Wear，把零碎的数据请求捆绑一起再进行操作。

5. **Object Pools** 使用对象池技术有很多好处，它可以避免内存抖动，提升性能，但是在使用的时候有一些内容是需要特别注意的。通常情况下，初始化的对象池里面都是空白的，当使用某个对象的时候先去对象池查询是否存在，如果不存在则创建这个对象然后加入对象池，但是我们也可以在程序刚启动的时候就事先为对象池填充一些即将要使用到的数据，这样可以在需要使用到这些对象的时候提供更快的首次加载速度，这种行为就叫做预分配。使用对象池也有不好的一面，程序员需要手动管理这些对象的分配与释放，所以我们需要慎重地使用这项技术，避免发生对象的内存泄漏。为了确保所有的对象能够正确被释放，我们需要保证加入对象池的对象和其他外部对象没有互相引用的关系。

6. **To Index or Iterate?** for index的方式有更好的效率，但是因为不同平台编译器优化各有差异，我们最好还是针对实际的方法做一下简单的测量比较好，拿到数据之后，再选择效率最高的那个方式。

7. **The Magic of LRU Cache** 使用LRU Cache能够显著提升应用的性能，可是也需要注意LRU Cache中被淘汰对象的回收，否者会引起严重的内存泄露。

8. **Using LINT for Performance Tips** Lint已经集成到Android Studio中了，我们可以手动去触发这个工具，点击工具栏的Analysis → Inspect Code，触发之后，Lint会开始工作，并把结果输出到底部的工具栏，我们可以逐个查看原因并根据指示做相应的优化修改。

9. **Hidden Cost of Transparency** 通常来说，对于不透明的View，显示它只需要渲染一次即可，可是如果这个View设置了alpha值，会至少需要渲染两次。

10. **Avoiding Allocations in onDraw()** 首先onDraw()方法是执行在UI线程的，在UI线程尽量避免做任何可能影响到性能的操作。虽然分配内存的操作并不需要花费太多系统资源，但是这并不意味着是免费无代价的。设备有一定的刷新频率，导致View的onDraw方法会被频繁的调用，如果onDraw方法效率低下，在频繁刷新累积的效应

下，效率低的问题会被扩大，然后会对性能有严重的影响。

11. **Tool: Strict Mode** Android提供了一个叫做Strict Mode的工具，我们可以通过手机设置里面的开发者选项，打开Strict Mode选项，如果程序存在潜在的隐患，屏幕就会闪现红色。我们也可以通过StrictMode API在代码层面做细化的跟踪，可以设置StrictMode监听那些潜在问题，出现问题时如何提醒开发者，可以对屏幕闪红色，也可以输出错误日志。

12. **Custom Views and Performance** Useless calls to onDraw()：我们知道调用View.invalidate()会触发View的重绘，有两个原则需要遵守，第1个是仅仅在View的内容发生改变的时候才去触发invalidate方法，第2个是尽量使用ClipRect等方法来提高绘制的性能。Useless pixels：减少绘制时不必要的绘制元素，对于那些不可见的元素，我们需要尽量避免重绘。Wasted CPU cycles：对于不在屏幕上的元素，可以使用Canvas.quickReject把他们给剔除，避免浪费CPU资源。另外尽量使用GPU来进行UI的渲染，这样能够极大的提高程序的整体表现性能。

13. **Batching Background Work Until Later** 1.AlarmManager 使用AlarmManager设置定时任务，可以选择精确的间隔时间，也可以选择非精确时间作为参数。除非程序有很强烈的需要使用精确的定时唤醒，否者一定要避免使用他，我们应该尽量使用非精确的方式。2.SyncAdapter 我们可以使用SyncAdapter为应用添加设置账户，这样在手机设置的账户列表里面可以找到我们的应用。这种方式功能更多，但是实现起来比较复杂。我们可以从这里看到官方的培训课程：http://developer.android.com/training/sync-adapters/index.html 3.JobSchedulor 这是最简单高效的方法，我们可以设置任务延迟的间隔，执行条件，还可以增加重试机制。

14. **Smaller Pixel Formats** Android的Heap空间是不会自动做兼容压缩的，意思就是如果Heap空间中的图片被收回之后，这块区域并不会和其他已经回收过的区域做重新排序合并处理，那么当一个更大的图片需要放到heap之前，很可能找不到那么大的连续空闲区域，那么就会触发GC，使得heap腾出一块足以放下这张图片的空闲区域，如果无法腾出，就会发生OOM。

15. **Smaller PNG Files** 尽量减少PNG图片的大小是Android里面很重要的一条规范。相比起JPEG，PNG能够提供更加清晰无损的图片，但是PNG格式的图片会更大，占用更多的磁盘空间。到底是使用PNG还是JPEG，需要设计师仔细衡量，对于那些使用JPEG就可以达到视觉效果的，可以考虑采用JPEG即可。

16. **Pre-scaling Bitmaps** 对bitmap做缩放，这也是Android里面最遇到的问题。对bitmap做缩放的意义很明显，提示显示性能，避免分配不必要的内存。Android提供了现成的bitmap缩放的API，叫做createScaledBitmap()

17. **Re-using Bitmaps** 使用inBitmap属性可以告知Bitmap解码器去尝试使用已经存在的内存区域，新解码的bitmap会尝试去使用之前那张bitmap在heap中所占据的pixel data内存区域，而不是去问内存重新申请一块区域来存放bitmap。利用这种特性，即使是上千张的图片，也只会仅仅只需要占用屏幕所能够显示的图片数量的内存大小。

18. **The Performance Lifecycle** Gather：收集数据，Insight：分析数据，Action：解决问题

Android性能优化典范 - 第3季

1. **Fun with ArrayMaps** 为了解决HashMap更占内存的弊端，Android提供了内存效率更高的ArrayMap。它内部使用两个数组进行工作，其中一个数组记录key hash过后的顺序列表，另外一个数组按key的顺序记录Key-Value值

2. **Beware Autoboxing** 有时候性能问题也可能是因为那些不起眼的小细节引起的，例如在代码中不经意的“自动装箱”。我们知道基础数据类型的大小：boolean(8 bits)， int(32 bits)， float(32 bits)，long(64 bits)，为了能够让这些基础数据类型在大多数Java容器中运作，会需要做一个autoboxing的操作，转换成Boolean，Integer，Float等对象

3. **SparseArray Family Ties** 为了避免HashMap的autoboxing行为，Android系统提供了SparseBoolMap，SparseIntMap，SparseLongMap，LongSparseMap等容器。

4. **The price of ENUMs** Android官方强烈建议不要在Android程序里面使用到enum。
5. **Trimming and Sharing Memory** Android系统提供了一些回调来通知应用的内存使用情况，通常来说，当所有的background应用都被kill掉的时候，forground应用会收到onLowMemory()的回调。在这种情况下，需要尽快释放当前应用的非必须内存资源，从而确保系统能够稳定继续运行。Android系统还提供了onTrimMemory()的回调，当系统内存达到某些条件的时候，所有正在运行的应用都会收到这个回调
6. **DO NOT LEAK VIEWS** 避免使用异步回调，避免使用Static对象，避免把View添加到没有清除机制的容器里面
7. **Location & Battery Drain** 其中存在的一个优化点是，我们可以通过判断返回的位置信息是否相同，从而决定设置下次的更新间隔是否增加一倍，通过这种方式可以减少电量的消耗
8. **Double Layout Taxation** 布局中的任何一个View一旦发生一些属性变化，都可能引起很大的连锁反应。例如某个button的大小突然增加一倍，有可能会导致兄弟视图的位置变化，也有可能导致父视图的大小发生改变。当大量的layout()操作被频繁调用执行的时候，就很可能引起丢帧的现象。
9. **Network Performance 101** 减少移动网络被激活的时间与次数，压缩传输数据
10. **Effective Network Batching** 发起网络请求与接收返回数据都是比较耗电的，在网络硬件模块被激活之后，会继续保持几十秒的电量消耗，直到没有新的网络操作行为之后，才会进入休眠状态。前面一个段落介绍了使用Batching的技术来捆绑网络请求，从而达到减少网络请求的频率。那么如何实现Batching技术呢？通常来说，我们可以会把那些发出的网络请求，先暂存到一个PendingQueue里面，等到条件合适的时候再触发Queue里面的网络请求。
11. **Optimizing Network Request Frequencies** 前面的段落已经提到了应该减少网络请求的频率，这是为了减少电量的消耗。我们可以使用Batching，Prefetching的技术来避免频繁的网络请求。Google提供了GCMNetworkManager来帮助开发者实现那些功能，通过提供的API，我们可以选择在接入WiFi，开始充电，等待移动网络被激活等条件下再次激活网络请求。
12. **Effective Prefetching** 类似上面的情况会频繁触发网络请求，但是如果我们能够预先请求后续可能会使用到网络资源，避免频繁的触发网络请求，这样就能够显著的减少电量的消耗。可是预先获取多少数据量是很值得考量的，因为如果预取数据量偏少，就起不到减少频繁请求的作用，可是如果预取数据过多，就会造成资源的浪费。

Android性能优化典范 - 第4季

1. **Cachematters for networking** 想要使得Android系统上的网络访问操作更加的高效就必须做好网络数据的缓存。这是提高网络访问性能最基础的步骤之一。从手机的缓存中直接读取数据肯定比从网络上获取数据要更加的便捷高效，特别是对于那些会被频繁访问到的数据，需要把这些数据缓存到设备上，以便更加快速的进行访问。
2. **Optimizing Network Request Frequencies** 首先我们要对网络行为进行分类，区分需要立即更新数据的行为和其他可以进行延迟的更新行为，为不同的场景进行差异化处理。其次要避免客户端对服务器的轮询操作，这样会浪费很多的电量与带宽流量。解决这个问题，我们可以使用Google Cloud Message来对更新的数据进行推送。然后在某些必须做同步的场景下，需要避免使用固定的间隔频率来进行更新操作，我们应该在返回的数据无更新的时候，使用双倍的间隔时间来进行下一次同步。最后更进一步，我们还可以通过判断当前设备的状态来决定同步的频率，例如判断设备处于休眠，运动等不同的状态设计各自不同时间间隔的同步频率。
3. **Effective Prefetching** 到底预取多少才比较合适呢？一个比较普适的规则是，在3G网络下可以预取1-5Mb的数据量，或者是按照提前预期后续1-2分钟的数据作为基线标准。在实际的操作当中，我们还需要考虑当前的网络速度来决定预取的数据量，例如在同样的时间下，4G网络可以获取到12张图片的数据，而2G网络则只能拿到3张图片的数据。所以，我们还需要把当前的网络环境情况添加到设计预取数据量的策略当中去。判断当前设备的状态与网络情况，可以使用前面提到过的GCMNetworkManager。

4. **Adapting to Latency** 一个典型的网络操作行为，通常包含以下几个步骤：首先手机端发起网络请求，到达网络服务运营商的基站，再转移到服务提供者的服务器上，经过解码之后，接着访问本地的存储数据库，获取到数据之后，进行编码，最后按照原来传递的路径逐层返回。常来说，我们可以把网络请求延迟划分为三档：例如把网络延迟小于60ms的划分为GOOD，大于220ms的划分为BAD，介于两者之间的划分为OK（这里的60ms，220ms会需要根据不同的场景提前进行预算推测）。

5. **Minimizing Asset Payload** 为了能够减小网络传输的数据量，我们需要对传输的数据做压缩的处理，这样能够提高网络操作的性能。首先需要做的是减少图片的大小，其次需要做的是减少序列化数据的大小。

6. **Service Performance Patterns** Service是Android程序里面最常用的基础组件之一，但是使用Service很容易引起电量的过度消耗以及系统资源的未及时释放。避免错误的使用Service，例如我们不应该使用Service来监听某些事件的变化，不应该搞一个Service在后台对服务器不断的进行轮询(应该使用Google Cloud Messaging)。如果已经事先知道Service里面的任务应该执行在后台线程(非默认的主线程)的时候，我们应该使用IntentService或者结合HanderThread，AsycnTask Loader实现的Service。

7. **Removing unused code** Android为我们提供了Proguard的工具来帮助应用程序对代码进行瘦身，优化，混淆的处理。它会帮助移除那些没有使用到的代码，还可以对类名，方法名进行混淆处理以避免程序被反编译。

8. **Removing unused resources** 所幸的是，我们可以使用Gradle来帮助我们分析代码，分析引用的资源，对于那些没有被引用到的资源，会在编译阶段被排除在APK安装包之外，要实现这个功能，对我们来说仅仅只需要在build.gradle文件中配置shrinkResource为true就好了

9. **Perf Theory: Caching** 当我们讨论性能优化的时候，缓存是最常见最有效的策略之一。无论是为了提高CPU的计算速度还是提高数据的访问速度，在绝大多数的场景下，我们都会使用到缓存。

10. **Perf Theory: Approximation(近似法)** 例如使用一张比较接近实际大小的图片来替代原图，换取更快的加载速度。所以对于那些对计算结果要求不需要十分精确的场景，我们可以使用近似法则来提高程序的性能。

11. **Perf Theory: Culling(遴选，挑选)** 一个提高性能的方法是逐步对数据进行过滤筛选，减小搜索的数据集，以此提高程序的执行性能。例如我们需要搜索到居住在某个地方，年龄是多少，符合某些特定条件的候选人，就可以通过逐层过滤筛选的方式来提高后续搜索的执行效率。

12. **Perf Theory: Threading** 使用多线程并发处理任务，从某种程度上可以快速提高程序的执行性能。对于Android程序来说，主线程通常也成为UI线程，需要处理UI的渲染，响应用户的操作等等。

13. **Perf Theory: Batching** 网络请求的批量执行是另外一个比较适合说明batching使用场景的例子，因为每次发起网络请求都相对来说比较耗时耗电，如果能够做到批量一起执行，可以大大的减少电量的消耗。

14. **Serialization performance** 数据序列化的行为可能发生在数据传递过程中的任何阶段，例如网络传输，不同进程间数据传递，不同类之间的参数传递，把数据存储到磁盘上等等。通常情况下，我们会把那些需要序列化的类实现Serializable接口(如下图所示)，但是这种传统的做法效率不高，实施的过程会消耗更多的内存。但是我们如果使用GSON库来处理这个序列化的问题，不仅仅执行速度更快，内存的使用效率也更高。Android的XML布局文件会在编译的阶段被转换成更加复杂的格式，具备更加高效的执行性能与更高的内存使用效率。

15. **Smaller Serialized Data** 数据呈现的顺序以及结构会对序列化之后的空间产生不小的影响。

16. **Caching UI data** 缓存UI界面上的数据，可以采用方案有存储到文件系统，Preference，SQLite等等，做了缓存之后，这样就可以在请求数据返回结果之前，呈现给用户旧的数据，而不是使用正在加载的方式让用户什么数据都看不到，当然在请求网络最新数据的过程中，需要有正在刷新的提示。至于到底选择哪个方案来对数据进行缓存，就需要根据具体情况来做选择了。

17. **CPU Frequency Scaling** 调节CPU的频率会执行的性能产生较大的影响，为了最大化的延长设备的续航时间，系统会动态调整CPU的频率，频率越高执行代码的速度自然就越快。我们可以使用Systrace工具来导出CPU的执行情况，以便帮助定位性能问题。

Android性能优化典范 - 第5季

1. **Threading Performance** AsyncTask: 为UI线程与工作线程之间进行快速的切换提供一种简单便捷的机制。适用于当下立即需要启动，但是异步执行的生命周期短暂的使用场景。HandlerThread: 为某些回调方法或者等待某些任务的执行设置一个专属的线程，并提供线程任务的调度机制。ThreadPool: 把任务分解成不同的单元，分发到各个不同的线程上，进行同时并发处理。IntentService: 适合于执行由UI触发的后台Service任务，并可以把后台任务执行的情况通过一定的机制反馈给UI。

2. **Understanding Android Threading** 通常来说，一个线程需要经历三个生命阶段：开始，执行，结束。线程会在任务执行完毕之后结束，那么为了确保线程的存活，我们会在执行阶段给线程赋予不同的任务，然后在里面添加退出的条件从而确保任务能够执行完毕后退出。

3. **Memory & Threading** 不要在任何非UI线程里面去持有UI对象的引用。系统为了确保所有的UI对象都只会被UI线程所进行创建，更新，销毁的操作，特地设计了对应的工作机制(当Activity被销毁的时候，由该Activity所触发的非UI线程都将无法对UI对象进行操作，否者就会抛出程序执行异常的错误)来防止UI对象被错误的使用。

4. **Good AsyncTask Hunting** AsyncTask虽然提供了一种简单便捷的异步机制，但是我们还是很有必要特别关注到他的缺点，避免出现因为使用错误而导致的严重系统性能问题。

5. **Getting a HandlerThread** HandlerThread比较合适处理那些在工作线程执行，需要花费时间偏长的任务。我们只需要把任务发送给HandlerThread，然后就只需要等待任务执行结束的时候通知返回到主线程就好了。另外很重要的一点是，一旦我们使用了HandlerThread，需要特别注意给HandlerThread设置不同的线程优先级，CPU会根据设置的不同线程优先级对所有的线程进行调度优化。

6. **Swimming in Threadpools** 线程池适合用在把任务进行分解，并发进行执行的场景。通常来说，系统里面会针对不同的任务设置一个单独的守护线程用来专门处理这项任务。

7. **The Zen of IntentService** 默认的Service是执行在主线程的，可是通常情况下，这很容易影响到程序的绘制性能(抢占了主线程的资源)。除了前面介绍过的AsyncTask与HandlerThread，我们还可以选择使用IntentService来实现异步操作。IntentService继承自普通Service同时又在内部创建了一个HandlerThread，在onHandlerIntent()的回调里面处理扔到IntentService的任务。所以IntentService就不仅仅具备了异步线程的特性，还同时保留了Service不受主页面生命周期影响的特点。

8. **Threading and Loaders** 当启动工作线程的Activity被销毁的时候，我们应该做点什么呢？为了方便的控制工作线程的启动与结束，Android为我们引入了Loader来解决这个问题。我们知道Activity有可能因为用户的主动切换而频繁的被创建与销毁，也有可能是因为类似屏幕发生旋转等被动原因而销毁再重建。在Activity不停的创建与销毁的过程当中，很有可能因为工作线程持有Activity的View而导致内存泄漏(因为工作线程很可能持有View的强引用，另外工作线程的生命周期还无法保证和Activity的生命周期一致，这样就容易发生内存泄漏了)。除了可能引起内存泄漏之外，在Activity被销毁之后，工作线程还继续更新视图是没有意义的，因为此时视图已经不在界面上显示了。

9. **The Importance of Thread Priority** 在Android系统里面，我们可以通过android.os.Process.setThreadPriority(int)设置线程的优先级，参数范围从-20到24，数值越小优先级越高。Android系统还为我们提供了以下的一些预设值，我们可以通过给不同的工作线程设置不同数值的优先级来达到更细粒度的控制。

10. **Profile GPU Rendering : M Update** 从Android M系统开始，系统更新了GPU Profiling的工具来帮助我们定位UI的渲染性能问题。早期的CPU Profiling工具只能粗略的显示出Process，Execute，Update三大步骤的时间耗费情况。

官方性能优化系列教程

架构分析

MVVM

MVP

阿里面试题

进程间通信方式

1. 通过Intent在Activity、Service或BroadcastReceiver间进行进程间通信，可通过Intent传递数据
2. AIDL方式
3. Messenger方式
4. 利用ContentProvider
5. Socket方式
6. 基于文件共享的方式

什么是协程

我们知道多个线程相对独立，有自己的上下文，切换受系统控制；而协程也相对独立，有自己的上下文，但是其切换由自己控制，由当前协程切换到其他协程由当前协程来控制。

内存泄露是怎么回事

由忘记释放分配的内存导致的

程序计数器，引到了逻辑地址（虚地址）和物理地址及其映射关系

虚拟机中的程序计数器是Java运行时数据区中的一小块内存区域，但是它的功能和通常的程序计数器是类似的，它指向虚拟机正在执行字节码指令的地址。具体点儿说，当虚拟机执行的方法不是native的时，程序计数器指向虚拟机正在执行字节码指令的地址；当虚拟机执行的方法是native的时，程序计数器中的值是未定义的。另外，程序计数器是线程私有的，也就是说，每一个线程都拥有仅属于自己的程序计数器。

数组和链表的区别

数组是将元素在内存中连续存放，由于每个元素占用内存相同，可以通过下标迅速访问数组中任何元素。但是如果要在数组中增加一个元素，需要移动大量元素，在内存中空出一个元素的空间，然后将要增加的元素放在其中。同样的道理，如果想删除一个元素，同样需要移动大量元素去填掉被移动的元素。如果应用需要快速访问数据，很少或不插入和删除元素，就应该用数组。

链表恰好相反，链表中的元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起。比如：上一个元素有个指针指到下一个元素，以此类推，直到最后一个元素。如果要访问链表中一个元素，需要从第一个元素开始，一直找到需要的元素位置。但是增加和删除一个元素对于链表数据结构就非常简单了，只要修改元素中的指针就可以了。如果应用需要经常插入和删除元素你就需要用链表数据结构了。

二叉树的深度优先遍历和广度优先遍历的具体实现

http://www.i3geek.com/archives/794

堆的结构

年轻代（Young Generation）、年老代（Old Generation）和持久代（Permanent Generation）。其中持久代主要存放的是Java类的类信息，与垃圾收集要收集的Java对象关系 不大。年轻代和年老代的划分是对垃 圾收集影响比较大的。

bitmap对象的理解

http://blog.csdn.net/angel1hao/article/details/51890938

什么是深拷贝和浅拷

浅拷贝：使用一个已知实例对新创建实例的成员变量逐个赋值，这个方式被称为浅拷贝。 深拷贝：当一个类的拷贝构造方法，不仅要复制对象的所有非引用成员变量值，还要为引用类型的成员变量创建新的实例，并且初始化为形式参数实例值。这个方式称为深拷贝

对象锁和类锁是否会互相影响

对象锁：Java的所有对象都含有1个互斥锁，这个锁由JVM自动获取和释放。线程进入synchronized方法的时候获取该对象的锁，当然如果已经有线程获取了这个对象的锁，那么当前线程会等待；synchronized方法正常返回或者抛异常而终止，JVM会自动释放对象锁。这里也体现了用synchronized来加锁的1个好处，方法抛异常的时候，锁仍然可以由JVM来自动释放。

类锁： 对象锁是用来控制实例方法之间的同步，类锁是用来控制静态方法（或静态变量互斥体）之间的同步。其实类锁只是一个概念上的东西，并不是真实存在的，它只是用来帮助我们理解锁定实例方法和静态方法的区别的。我们都知道，java类可能会有很多个对象，但是只有1个Class对象，也就是说类的不同实例之间共享该类的Class对象。Class对象其实也仅仅是1个java对象，只不过有点特殊而已。由于每个java对象都有1个互斥锁，而类的静态方法是需要Class对象。所以所谓的类锁，不过是Class对象的锁而已。获取类的Class对象有好几种，最简单的就是MyClass.class的方式。

类锁和对象锁不是同1个东西，一个是类的Class对象的锁，一个是类的实例的锁。也就是说：1个线程访问静态synchronized的时候，允许另一个线程访问对象的实例synchronized方法。反过来也是成立的，因为他们需要的锁是不同的。

looper架构

http://wangkuiwu.github.io/2014/08/26/MessageQueue/

自定义控件原理

http://www.jianshu.com/p/988326f9c8a3

binder工作原理

Binder是客户端和服务端进行通讯的媒介

ActivityThread，Ams，Wms的工作原理

ActivityThread: 运行在应用进程的主线程上，响应 ActivityManangerService 启动、暂停Activity，广播接收等消息。

ams:统一调度各应用程序的Activity、内存管理、进程管理

Java中final，finally，finalize的区别

- final 用于声明属性，方法和类， 分别表示属性不可变， 方法不可覆盖， 类不可继承
- finally 是异常处理语句结构的一部分，表示总是执行
- finalize 是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等. JVM不保证此方法总被调用

一个文件中有100万个整数，由空格分开，在程序中判断用户输入的整数是否在此文件中。说出最优的方法

两个进程同时要求写或者读，能不能实现？如何防止进程的同步？

volatile 的意义？

防止CPU指令重排序

单例

```
public class Singleton{
private volatile static Singleton mSingleton;
private Singleton(){
}
public static Singleton getInstance(){
  if(mSingleton == null){\\A
    synchronized(Singleton.class){\\C
     if(mSingleton == null)
      mSingleton = new Singleton();\\B
     }
    }
    return mSingleton;
  }
}
```

Given a string, determine if it is a palindrome（回文，如果不清楚，按字面意思脑补下）, considering only alphanumeric characters and ignoring cases.

For example, "A man, a plan, a canal: Panama" is a palindrome. "race a car" is not a palindrome.

Note: Have you consider that the string might be empty? This is a good question to ask during an interview. For the purpose of this problem, we define empty string as valid palindrome.

```
public boolean isPalindrome(String palindrome){
        char[] palindromes = palidrome.toCharArray();
         if(palindromes.lengh == 0){
             return true
         }
         Arraylist<Char> temp = new Arraylist();
         for(int i=0;i<palindromes.length;i++){
         if((palindromes[i]>'a' && palindromes[i]<'z')||palindromes[i]>'A' && palindromes[i]<'Z')){
         temp.add(palindromes[i].toLowerCase());
         }
        }
         for(int i=0;i<temp.size()/2;i++){
         if(temp.get(i) != temp.get(temp.size()-i)){
         //
         return false;
         }
         }
```

```
            return true;
    }
```

烧一根不均匀的绳，从头烧到尾总共需要1个小时。现在有若干条材质相同的绳子，问如何用烧绳的方法来计时一个小时十五分钟呢

用两根绳子，一个绳子两头烧，一个一头烧。

腾讯

- 2000万个整数，找出第五十大的数字？ 冒泡、选择、建堆
- 从网络加载一个10M的图片，说下注意事项 图片缓存、异常恢复、质量压缩
- 自定义View注意事项 渲染帧率、内存
- 项目中常用的设计模式 单例、观察者、适配器、建造者
- JVM的理解 http://www.infoq.com/cn/articles/java-memory-model-1

这是一位攻城狮面试了近十家互联网公司总结下来的经验之谈：

我现在主要的方向是Java服务端开发，把遇到的问题和大家分享一下，也谈谈关于技术人员如何有方向的提高自己，做到有的放矢。

面试遇到的问题

1. 百度

百度最近真是炙手可热，贴吧事件刚结束，医疗竞价排名又闹得沸沸扬扬，一些论坛上连带程序员都开始招黑了，友谊的小船可是说翻就翻。

说回面试，百度面了两次，分别是百度糯米和金融事业部，百度目前只有这两个部门的招聘岗位和我比较匹配。面试都在西二旗的百度新总部，园区还在施工，离地铁也比较远，需要打车过去。

面试官自带电脑，整个面试过程都在记录，首先详细询问了最近一份工作项目的架构和工作内容，面试主要围绕工作中用到的组件和中间件技术来扩展，考察掌握程度。

> MySQL InnoDB存储的文件结构
>
> 索引树是如何维护的?
>
> 数据库自增主键可能的问题
>
> Redis的并发竞争问题如何解决了解Redis事务的CAS操作吗
>
> **分析线程池的实现原理和线程的调度过程**
>
> 动态代理的几种方式
>
> Spring AOP与IOC的实现
>
> 为什么CGlib方式可以对接口实现代理?
>
> RMI与代理模式
>
> Dubbo的底层实现原理和机制
>
> 描述一个服务从发布到被消费的详细过程
>
> **算法方面考察了一个简单的数组就地去重问题，用丢弃数组尾部元素的方式实现**

百度金融的面试安排在了周六，最近应该在各种扩张，各个招聘网站随处可见招聘启事。一面面试官很赞，态度认真，有些问题没有思路会给你提示，交流的不错，二面被告知缺少金融支付背景，不过作为一名工作不到两年的新人，我觉得被Pass主要原因应该是工作经验比较少，教育背景也不太亮眼。

> **分布式系统怎么做服务治理**
>
> 接口的幂等性的概念
>
> Maven出现版本冲突如何解决
>
> JVM垃圾回收机制，何时触发MinorGC等操作
>
> 新生代和老生代的内存回收策略
>
> Eden和Survivor的比例分配等
>
> Synchronized和Lock的区别

两次面试，感觉百度的流程比较严格，面试官挺不错的，简单可信赖，虽然工作中一般都用谷歌。

这是一个段子：

有次面百度，我提到了一个比赛，面试官很感兴趣，想搜一下，于是先用百度搜了一下关键字，首屏没有找到，面试官面不改色，熟练的打开了谷歌输入关键字，发现第一个就是官方网站。

2. 阿里巴巴

在内推网上收到了阿里菜鸟和阿里云安全部门的面试，后来参加了阿里云的面试。阿里的面试安排的很快，这次止步二面，两轮面试都是电面。听朋友说阿里五轮面试，四轮技术一轮HR，技术面试是部门的几个同事交叉面试，也有了了解。

一面总体上还是围绕项目架构、Java基础、JVM、并发编程、数据库操作、中间件技术和Dubbo服务治理框架等展开，

可能因为是云安全部门，有一半时间在考察JVM，还提问了一些编译优化的知识，一面结束后很快安排了二面，相对一面，二面的问题更深入，问题比较刨根问底，更加注重对一些技术细节的理解和把握。

比如数据库操作，面试官会详细的问你数据库插入和删除一条数据的过程在底层是如何执行的，项目里配置了读写分离，也会比较深入的就实现方法和底层逻辑展开讨论。

> JVM内存分代
>
> Java 8的内存分代改进
>
> 深入分析了Classloader，双亲委派机制
>
> JVM的编译优化
>
> 对Java内存模型的理解，以及其在并发中的应用
>
> 指令重排序，内存栅栏等
>
> HashMap的并发问题
>
> 了解LinkedHashMap的应用吗

> 在工作中遇到过哪些设计模式，是如何应用的

阿里的岗位大都在杭州，面试结束特意关注了一下那边的生活成本，目前杭州房子均价不到两万，相比浙江一些县市的房价都破两万，杭州的房价应该比较正常。如果拿到阿里和网易等几家互联网公司的高薪，买房和生活的确比北京要轻松很多，果断决定再沉淀一段时间，两年后P7再战。

3. 优酷土豆

优酷的面试都是二对一，每轮面试两个面试官，一面比较顺利，主要是Java基础，Spring原理，Java NIO，并发和集合框架等，可能是因为视频网站，优酷考察网络原理的知识多，比如TCP/IP协议、长连接与短连接等。

一面提到了自己可能会在下半年学习大数据与机器学习相关的知识，二面就在这上面栽了跟头，问了很多海量数据的问题。

> TCP/IP协议
>
> 长连接与短连接
>
> mapreduce过程
>
> 多路归并的时间复杂度
>
> 海量url去重类问题
>
> Java NIO使用
>
> 倒排索引的原理

> 对分词技术的了解

面试中给了一个具体场景，考察对MapReduce过程的理解，比如Map阶段和Reduce阶段是如何进行的等，Reduce阶段面试官希望分析给出一个多路归并的时间复杂度，用外排序的知识简单分析了一下，回答的不太好。回来以后搜索了胜者树和败者树的优化，发现这里面的内容还挺多，深刻体会到有些知识点如果平时掌握的不够全面深刻，很难信手拈来。

4. 搜狐新闻

搜狐最近应该是没有招聘计划，面试等待时间比较长。做了笔试题，一面是个和我年纪相仿的面试官，针对笔试和简历提问了一些基础问题，聊得挺投机，二面技术经理就比较偏架构和中间件的应用，提问了项目，主要考察了服务治理和消息队列等中间件使用的问题：

> 消息中间件如何解决消息丢失问题
>
> Dubbo的服务请求失败怎么处理
>
> 重连机制会不会造成错误
>
> 对分布式事务的理解
>
> 深入分析几个设计模式

面试最后提问了一个不定长字符串转为定长字符串的问题，刚刚面过优酷，这个简单的问题被我想复杂了，没有Get到面试官的点，考虑了唯一性、性能等，扯了一大堆。也提醒一下大家，面试过程中要保持清醒，不要有思维定式，除非是底层研发岗位，社招对算法的考察不会特别难，用正常的思路去解决就可以。

5. 58赶集

58总部在798附近，全天有班车可以过去。总体上，感觉面试官的问题非常接地气，三轮技术面，大部分是实际场景的算法和系统设计类问题：

> HTTP请求的报文格式Spring的事务实现原理
>
> 实际场景问题，大量用户数据如何在内存中排序和去重
>
> 缓存机器增删如何对系统影响最小，一致性哈希的实现
>
> Redis持久化的几种方式
>
> Redis的缓存失效策略
>
> 实际场景问题解决，典型的TOP K问题
>
> 实际场景问题，海量登录日志如何排序和处理SQL操作，主要是索引和聚合函数的应用

三面面试官提问了一些优点和缺点的自我评价类问题，简单交流以后对我给出了一些中肯的建议，非常感谢。

6. 国美在线

国美在线面试最开始是部门经理沟通，在知道我毕业不满两年以后，重新去做了一份笔试题，主要考察Java基础，数据库，设计模式以及数据结构，要求写出B-Tree的节点结构，算法题目是一道等概率抽奖的题目，用蓄水池抽样算法解决了。

> SQL语句编写
>
> MySQL的几种优化
>
> Spring行级锁
>
> Spring衍生的相关其他组件整理
>
> RMI的几种协议和实现框架
>
> BTree相关的操作
>
> 数据库锁表的相关处理
>
> 考察跳台阶问题

和面试官的交流比较轻松，面试官提示我要加强数据库操作的掌握，另外面试过程中询问了一些工作中用到框架和组件的版本等细节问题，平时没太关注，

后来思考了一下，对开源组件的应用，版本的管理很重要，不注意可能会发生一些诡异的问题。

7. 去哪儿网，口袋购物等公司

除了上面的公司，还参加过去哪儿网、口袋购物、链家等几家公司的面试。去哪儿网中规中矩，口袋购物的工作环境非常不错。链家网最近有新浪的鸟哥加入任技术总监，在IT圈子里挺火，面试了链家旗下的两个租房部门，技术氛围不错。

几家公司的模式和问题都类似，注重对基础和编程能力的考察，以及对分布式系统设计和架构的理解。值得一提的是一家创业公司的面试，过程十分简单粗暴。没有自我介绍，面试官看完简历就在白板上提了一个多线程调度问题，递过来MAC就开始敲代码。

写完以后我表示这题目意义不大，问了Redis，要求十五分钟实现一个LRUCache，再次现场写代码。写到一半面试官看没问题就打断了，问对公司有什么想了解的，等了一会让我回去了，就这么被Pass，创业公司效率果然高。

面试中要保持清醒，比如被问到十万个ip段查找这个问题，首先是一个典型的查找问题，明确了这个，就可以针对性的选择相关的算法实现，如二分查找、二叉查找树等。推荐画图表达的方式，做过的项目架构，各种框架和中间件的设计实现，通过画图的方式都可以很好的阐述，可以随身带着纸和笔，面试本来就是一次很好的学习过程，一些问题也可以记录下来。

一般来说，面试过程类似一个寻路算法，交流过程中如果提到了面试官感兴趣的某一点，就会就这个点展开，然后一直提出问题到你不能回答为止，或者你特别牛在这个领域直接秒杀面试官，这样一条路线走通，再换下一条路线。

攻城狮如何用正确的姿势提高技术水平

一般来说，主流互联网公司都在用的就是业内比较成熟和流行的技术，最简单的方式就是看招聘要求，虽然大部分公司的Job Description都有抄袭的嫌疑，但是多比较几个招聘，还是可以了解主流互联网公司的技术方向。下面是从从拉勾上找的几个招聘要求：

百度核心业务部门：

-3年以上互联网公司\大型企业软件的研发经验，擅长web系统研发，熟悉互联网商业产品，对性能优化、架构设计有一定理解者优先

-对Java面向对象软件结构有深入理解以及很强的应用能力，能够熟练应用JavaEE等WEB开发技术，熟练掌握Spring等主流的开发框架

-掌握数据库的理论知识，熟练使用MySQL等主流数据库。熟悉Memcached、Redis等Key-Value存储系统者更佳

-熟练使用shell、python等至少一种脚本编程

-熟练掌握系统故障分析和性能调优的方法和工具，能够快速定位--并解决问题，保证线上服务稳定、高效

-思路开阔，善于思考，能够对自己负责研发的产品提出改进建议

-对技术有热情，关注业界新知，喜欢和他人交流和分享

阿里巴巴：

岗位要求:
1.本科或以上学历，计算机软件或相关专业，4年以上互联网公司Java开发经验。
2.熟悉Java/JEE， 基础扎实，熟练掌握常用Java技术框架，能编写高质量简洁、清晰的代码。
3.对于Java基础技术体系（包括JVM、类装载机制、JUC、NIO、网络）有深入的理解和实践经验。
4.熟悉底层中间件、分布式技术（包括RCP框架、缓存、消息系统、热部署、JMX等），对CAP定论有深入的理解。
5.对于高并发、高可用、高性能、大数据处理有过实际项目产品经验者优先考虑。
6.具有比较强的问题分析和处理能力，热衷于技术，有一定的技术代码癖,github、stackoverflow活跃者优先考虑。

美团酒店事业部：

1、本科及以上学历，扎实的计算机专业基本功。
2、两年以上 Java 开发经验，精通 Java 及面向对象设计开发，对部分 Java 技术有深入研究，研究过优秀开源软件的源码并有心得者优先。
3、熟悉常见设计模式，精通 Spring， MyBatis等流行开源框架。
4、精通 MySQL 应用开发，熟悉数据库原理和常用性能优化技术，以及 NoSQL，Queue 的原理、使用场景以及限制。
5、参与过大型复杂分布式互联网 WEB 系统的设计开发者优先，拥有和工作年限相称的广度和（或）深度。
6、有较强的逻辑思维能力，善于分析、归纳、解决问题；能够独立或带队进行项目开发。
7、有业务系统开发经验的. 对原有业务系统有深度和广度的了解, 以及对原有系统的改进意见(包括业务架构, 业务流程等)。
8、有酒店、旅游行业背景开发经验者优先。

既然是社招，工作经验是必须的，三年以上最好，上面的几个JD里也体现了。然后是技术方面，结合自己的体会，总结下面几点：

基础知识必须要扎实

语言基础，计算机基础，算法和基本的Linux运维等

针对Java语言，需要对集合类，并发包，IO/NIO，JVM，内存模型，泛型，异常，反射等都有比较深入的了解，最好是学习过部分源码。这些知识点都是相通的，在面试中也可以体现。

从源码的角度，可以深入到哈希表的实现，拉链法以外的哈希碰撞解决方法，如何平衡内部数组保证哈希表的性能不会下降等；

从线程安全的角度，可以扩展到HashTable、ConcurrentHashMap等其他的数据结构，可以比较两种不同的加锁方式，RetreenLock的实现和应用，继续深入可以考察Java内存模型，Volitale原语，内存栅栏等；横向扩展可以考察有序的Map结构如TreeMap、LinkedHashMap，继而考察红黑树，LRU缓存，HashMap的排序等知识。

Java方向的中高级职位，会比较重视对虚拟机的掌握，诸如类加载机制，内存模型等，这些在程序的优化和并发编程中都非常重要。

算法方面，基本的排序和查找算法，对递归，分治等思想的掌握。如果算法基础不太好，推荐《编程珠玑》等，每一章都很经典。

计算机基础方面，比如TCP/IP协议和操作系统的知识也是必备的，这些都是大学计算机专业的基础课，也是做开发基本的素养。

系统设计能力

设计模式，造轮子的能力，各种缓存和数据库应用，缓存，中间件技术，高并发和高可用的分布式系统设计等。

大型互联网公司每天要面对海量的请求，都会考察分布式系统的架构和设计，如何构建高并发高可用的系统。另外因为用户基数比较大，一个细微的优化可能会给带来很大的收益，所以对一些技术栈的掌握要求都比较深入。比如对MySQL数据库，需要知道相关的配置和优化，业务上来以后如何分库分表，如何合理的配置缓存，一个经验丰富的服务端开发人员，也应该是一个称职的DBA。

对常用的开发组件，比如中间件，RPC框架等都要有一定的了解，虽然工作中可能用不到我们自己造轮子，但是掌握原理才会得心应手。这部分知识主要靠工作积累，推荐《大型网站技术架构与Java中间件实践》，还有曾贤杰的《大型网站系统架构与实践》，里面对大型网站的演变，服务治理和中间件的使用做了很详细的阐述。

作为业务开发人员，有必要了解压力测试相关的指标，比如QPS，用户平均等待时间等，可以帮助你更好的了解自己的系统。

软性指标

快速学习，良好的沟通能力，以及对相关行业的了解。

公司招聘会比较看重一个人的学习能力，是不是值得培养，很多公司校招的毕业生薪资会倒挂工作多年的老员工，也是这样。像沟通习惯，逻辑分析能力，这些都属于软实力，短时间内很难提高，需要长期的养成和持续不断的投入。好多公司还会看重所在行业，虽然是做业务，但是对产品和行业的了解也很重要。比如互联网金融类公司的岗位，如果有过支付和银行相关的系统开发经验肯定会有加分，这点和每个人的长期规划有关。

有了方向，接下来就是如何提高，说一些自己的感想。

很多时候，除非你的工作内容就是要应对高并发，海量用户等场景，否则通过加班或者说重复性的工作，其实很难有提高。技术人员最直接的提高方式，还是需要跳出来，在工作以外审视自己，比如广泛的阅读技术书籍，多去论坛和各路牛人交流，了解主流互联网公司的技术栈，有针对性的去学习和了解。同时也可以适当的了解一些产品或者设计的知识，以点带面，复合人才肯定更受欢迎，对待面试，要像和妹子约会一样，表现自己平常的一面就可以了。

花絮

本文为完整版，加了一些彩蛋哦！文末有面试和必备的技能点总结。

> 也许会有人感叹某些人的运气比较好，但是他们不曾知道对方吃过多少苦，受过多少委屈。某些时候就是需要我们用心去发现突破点，然后顺势而上，抓住机遇，那么你将会走向另外一条大道，成就另外一个全新的自我。

先简单说说我最近的面试经历吧。面试的公司很多，其中有让我心血沸腾的经历，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。

> 下文真的很长，你可以把这篇文章当做看小说一样，快速浏览一下，但是希望你能将文中提到的那些技能掌握。那也就不枉费我花了一两天时间专门整理这些。我的这些经验仅供参考，希望你能做的比我好，同时希望你在以后的面试中能轻松应对。

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:

1. A类： BAT公司，非常靠谱，各项待遇都是很优厚的
2. 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
3. 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，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。

> 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，新闻中也纰漏了很多，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。

> 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 “X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。我实在想不出如何能实现这样的效果。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会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onMeasuer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一些待遇什么的，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。

之后再来复试的时候，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）。

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，这个可以适当的要高点，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，以下就总结性的说说，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 内存管理的东西，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）。电梯上不去了，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，可以说说思路和想法，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。

在此我真心后悔当时的冲动，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），总结之后才能更近一步。

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崩腾，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识。

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，没想到竟然这个严格。

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？噔噔噔……

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->子 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，但是就因为没有留意，所以就没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。问了好一个美女才回到前台，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的，千万别干坏事哦，会被查出来的，哈哈哈）。

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，毕竟孰能生巧）。

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。

1. 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
2. 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

3. 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。

如果你有面试的疑问或者困惑，可以加我的微信公众账号:mianshishuo, 可以扫描下方二维码，一起来吐槽面试中的感受。我将不定期分享最新的 Android 面试题与面试经验。也可以将你们的面试经验与问题发给我一同讨论，非常感谢。

今天给大家带来的是腾讯的面试题，觉得有用的亲，赏个脸，轻点上面蓝色黑马程序员几个字关注我吧！

1、腾讯笔试题：const的含义及实现机制const的含义及实现机制，比如：const int i,是怎么做到i只可读的？

const用来说明所定义的变量是只读的。

这些在编译期间完成，编译器可能使用常数直接替换掉对此变量的引用。

2、腾讯笔试题：买200返100优惠券，实际上折扣是多少？

到商店里买200的商品返还100优惠券（可以在本商店代替现金）。请问实际上折扣是多少？

由于优惠券可以代替现金，所以可以使用200元优惠券买东西，然后还可以获得100元的优惠券。

假设开始时花了x元，那么可以买到 x + x/2 + x/4 + ...的东西。所以实际上折扣是50%.（当然，大部分时候很难一直兑换下去，所以50%是折扣的上限） 如果使用优惠券买东西不能获得新的优惠券，那么总过花去了200元，可以买到200+100元的商品，所以实际折扣为 200/300 = 67%.

3、腾讯笔试题：tcp三次握手的过程，accept发生在三次握手哪个阶段？

accept发生在三次握手之后。

第一次握手：客户端发送syn包(syn=j)到服务器。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个ASK包（ask=k）。

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)。

三次握手完成后，客户端和服务器就建立了tcp连接。这时可以调用accept函数获得此连接。

4、腾讯笔试题：用UDP协议通讯时怎样得知目标机是否获得了数据包用UDP协议通讯时怎样得知目标机是否获得了数据包？

可以在每个数据包中插入一个唯一的ID，比如timestamp或者递增的int。

发送方在发送数据时将此ID和发送时间记录在本地。

接收方在收到数据后将ID再发给发送方作为回应。

发送方如果收到回应，则知道接收方已经收到相应的数据包；如果在指定时间内没有收到回应，则数据包可能丢失，需要重复上面的过程重新发送一次，直到确定对方收到。

5、腾讯笔试题：统计论坛在线人数分布 求一个论坛的在线人数，假设有一个论坛，其注册ID有两亿个，每个ID从登陆到退出会向一个日志文件中记下登陆时间和退出时间，要求写一个算法统计一天中论坛的用户在线分布，取样粒度为秒。

一天总共有 360024 = 86400秒。

定义一个长度为86400的整数数组int delta[86400]，每个整数对应这一秒的人数变化值，可能为正也可能为负。开始时将数组元素都初始化为0。

然后依次读入每个用户的登录时间和退出时间，将与登录时间对应的整数值加1，将与退出时间对应的整数值减1。

这样处理一遍后数组中存储了每秒中的人数变化情况。

定义另外一个长度为86400的整数数组int online_num[86400]，每个整数对应这一秒的论坛在线人数。

假设一天开始时论坛在线人数为0，则第1秒的人数online_num[0] = delta[0]。第n+1秒的人数online_num[n] = online_num[n-1] + delta[n]。

这样我们就获得了一天中任意时间的在线人数。

6、腾讯笔试题：从10G个数中找到中数 在一个文件中有 10G 个整数，乱序排列，要求找出中位数。内存限制为2G。

不妨假设10G个整数是64bit的。

2G内存可以存放256M个64bit整数。

我们可以将64bit的整数空间平均分成256M个取值范围，用2G的内存对每个取值范围内出现整数个数进行统计。这样遍历一边10G整数后，我们便知道中数在那个范围内出现，以及这个范围内总共出现了多少个整数。

如果中数所在范围出现的整数比较少，我们就可以对这个范围内的整数进行排序，找到中数。如果这个范围内出现的整数比较多，我们还可以采用同样的方法将此范围再次分成多个更小的范围（256M=2^28，所以最多需要3次就可以将此范围缩小到1，也就找到了中数）。

7、腾讯笔试题：两个整数集合A和B，求其交集两个整数集合A和B，求其交集。

- 读取整数集合A中的整数，将读到的整数插入到map中，并将对应的值设为1。
- 读取整数集合B中的整数，如果该整数在map中并且值为1，则将此数加入到交集当中，并将在map中的对应值改为2。

通过更改map中的值，避免了将同样的值输出两次。

8、腾讯笔试题：找出1到10w中没有出现的两个数字 有1到10w这10w个数，去除2个并打乱次序，如何找出那两个数？

申请10w个bit的空间，每个bit代表一个数字是否出现过。

开始时将这10w个bit都初始化为0，表示所有数字都没有出现过。

然后依次读入已经打乱循序的数字，并将对应的bit设为1。

当处理完所有数字后，根据为0的bit得出没有出现的数字。

首先计算1到10w的和，平方和。

然后计算给定数字的和，平方和。

两次的到的数字相减，可以得到这两个数字的和，平方和。

所以我们有

x + y = n

x^2 + y^2 = m

解方程可以得到x和y的值。

9、腾讯笔试题：需要多少只小白鼠才能在24小时内找到毒药有1000瓶水，其中有一瓶有毒，小白鼠只要尝一点带毒的水24小时后就会死亡，至少要多少只小白鼠才能在24小时时鉴别出那瓶水有毒？

最容易想到的就是用1000只小白鼠，每只喝一瓶。但显然这不是最好答案。

既然每只小白鼠喝一瓶不是最好答案，那就应该每只小白鼠喝多瓶。那每只应该喝多少瓶呢？

首先让我们换种问法，如果有x只小白鼠，那么24小时内可以从多少瓶水中找出那瓶有毒的？

由于每只小白鼠都只有死或者活这两种结果，所以x只小白鼠最大可以表示2^x种结果。如果让每种结果都对应到某瓶水有毒，那么也就可以从2^x瓶水中找到有毒的那瓶水。那如何来实现这种对应关系呢？

第一只小白鼠喝第1到2^(x-1)瓶，第二只小白鼠喝第1到第2^(x-2)和第2^(x-1)+1到第2^(x-1) + 2^(x-2)瓶....以此类推。

回到此题，总过1000瓶水，所以需要最少10只小白鼠。

10、腾讯笔试题：根据上排的数填写下排的数，并满足要求。

根据上排给出十个数，在其下排填出对应的十个数，要求下排每个数都是上排对应位置的数在下排出现的次数。上排的数：0，1，2，3，4，5，6，7，8，9。

11、腾讯笔试题：判断数字是否出现在40亿个数中？

给40亿个不重复的unsigned int的整数，没排过序的，然后再给几个数，如何快速判断这几个数是否在那40亿个数当中?

答案：unsigned int 的取值范围是0到2^32-1。我们可以申请连续的2^32/8=512M的内存，用每一个bit对应一个unsigned int数字。首先将512M内存都初始化为0，然后每处理一个数字就将其对应的bit设置为1。当需要查询时，直接找到对应bit，看其值是0还是1即可。

之前实习的时候就想着写一篇面经，后来忙就给忘了，现在找完工作了，也是该静下心总结一下走过的路程了，我全盘托出，奉上这篇诚意之作，希望能给未来找工作的人一点指引和总结，也希望能使大家少走点弯路，如果能耐心读完，相信对你会找到你需要的东西。

先说一下LZ的基本情况，LZ是四川某985学校通信专业的研究生（非计算机），大学阶段也就学了C语言，根本没想过最后要成为码农。大四才开始学java，研一下开始学android，所以LZ觉得自己开始就是一个小白，慢慢成长起来的。

1. 心态

心态很重要！ 心态很重要！ 心态很重要！

重要的事情说三遍，这一点我觉得是必须放到前面来讲。

找工作之前，有一点你必须清楚，就是找工作是一件看缘分的事情，不是你很牛逼，你就一定能进你想进的公司，都是有一个概率在那。如果你基础好，项目经验足，同时准备充分，那么你拿到offer的概率就会比较高；相反，如果你准备不充分，基础也不好，那么你拿到offer的概率就会比较低，但是你可以多投几家公司，这样拿到offer的几率就要大一点，因为你总有运气好的时候。所以，不要惧怕面试，刚开始失败了没什么的，多投多尝试，面多了你就自然能成面霸了。得失心也不要太重，最后每个人都会有offer的。

还有一个对待工作的心态，有些人可能觉得自己没有动力去找一个好工作。其实你需要明白一件事情，你读了十几二十年的书，为的是什么，最后不就是为了找到一个好工作么。现在到了关键时刻，你为何不努力一把呢，为什么不给自己一个好的未来呢，去一个自己不满意的公司工作，你甘心吗?

想清楚这一点，我相信大多数人都会有一股干劲了，因为LZ刚刚准备开始找实习的时候，BAT这种公司想都不敢想，觉得能进个二线公司就很不错了，后来发现自己不逼自己一把，你真不知道自己有多大能耐，所以请对找工作保持积极与十二分的热情，也请认真对待每一次笔试面试。

2. 基础

基础这东西，各个公司都很看重，尤其是BAT这种大公司，他们看中人的潜力，他们舍得花精力去培养，所以基础是重中之重。之前很多人问我，项目经历少怎么办，那就去打牢基础，当你的基础好的发指的时候，你的其他东西都不重要了。

基础无外乎几部分：语言（C/C++或java），操作系统，TCP/IP，数据结构与算法，再加上你所熟悉的领域。这里面其实有很多东西，各大面试宝典都有列举。

在这只列举了Android客户端所需要的和我面试中所遇到的知识点，尽量做到全面，如果你掌握了以下知识点，去面android客户端应该得心应手。

J2SE基础

1.九种基本数据类型的大小，以及他们的封装类。 2.Switch能否用string做参数？ 3.equals与==的区别。 4.Object有哪些公用方法？ 5.Java的四种引用，强弱软虚，用到的场景。 6.Hashcode的作用。 7.ArrayList、LinkedList、Vector的区别。 8.String、StringBuffer与StringBuilder的区别。 9.Map、Set、List、Queue、Stack的特点与用法。 10.HashMap和HashTable的区别。 11.HashMap和ConcurrentHashMap的区别，HashMap的底层源码。 12.TreeMap、HashMap、LindedHashMap的区别。 13.Collection包结构，与Collections的区别。 14.try catch finally，try里有return，finally还执行么？ 15.Excption与Error包结构。OOM你遇到过哪些情况，SOF你遇到过哪些情况。 16.Java面向对象的三个特征与含义。 17.Override和Overload的含义去区别。 18.Interface与abstract类的区别。 19.Static class 与non static class的区别。 20.java多态的实现原理。 21.实现多线程的两种方法：Thread与Runable。 22.线程同步的方法：sychronized、lock、reentrantLock等。 23.锁的等级：方法锁、对象锁、类锁。 24.

写出生产者消费者模式。 25.ThreadLocal的设计理念与作用。 26.ThreadPool用法与优势。 27.Concurrent包里的其他东西：ArrayBlockingQueue、CountDownLatch等等。 28.wait()和sleep()的区别。 29.foreach与正常for循环效率对比。 30.Java IO与NIO。 31.反射的作用于原理。 32.泛型常用特点，List能否转为List。 33.解析XML的几种方式的原理与特点：DOM、SAX、PULL。 34.Java与C++对比。 35.Java1.7与1.8新特性。 36.设计模式：单例、工厂、适配器、责任链、观察者等等。 37.JNI的使用。

Java里有很多很杂的东西，有时候需要你阅读源码，大多数可能书里面讲的不是太清楚，需要你在网上寻找答案。

推荐书籍：《java核心技术卷I》《Thinking in java》《java并发编程》《effictive java》《大话设计模式》

JVM

1.内存模型以及分区，需要详细到每个区放什么。 2.堆里面的分区：Eden，survival from to，老年代，各自的特点。 3.对象创建方法，对象的内存分配，对象的访问定位。 4.GC的两种判定方法：引用计数与引用链。 5.GC的三种收集方法：标记清除、标记整理、复制算法的原理与特点，分别用在什么地方，如果让你优化收集方法，有什么思路？ 6.GC收集器有哪些？CMS收集器与G1收集器的特点。 7.Minor GC与Full GC分别在什么时候发生？ 8.几种常用的内存调试工具：jmap、jstack、jconsole。 9.类加载的五个过程：加载、验证、准备、解析、初始化。 10.双亲委派模型：Bootstrap ClassLoader、Extension ClassLoader、ApplicationClassLoader。 11.分派：静态分派与动态分派。

JVM过去过来就问了这么些问题，没怎么变，内存模型和GC算法这块问得比较多，可以在网上多找几篇博客来看看。

推荐书籍：《深入理解java虚拟机》

操作系统

1.进程和线程的区别。 2.死锁的必要条件，怎么处理死锁。 3.Window内存管理方式：段存储，页存储，段页存储。 4.进程的几种状态。 5.IPC几种通信方式。 6.什么是虚拟内存。 7.虚拟地址、逻辑地址、线性地址、物理地址的区别。

因为是做android的这一块问得比较少一点，还有可能上我简历上没有写操作系统的原因。

推荐书籍：《深入理解现代操作系统》

TCP/IP

1.OSI与TCP/IP各层的结构与功能，都有哪些协议。 2.TCP与UDP的区别。 3.TCP报文结构。 4.TCP的三次握手与四次挥手过程，各个状态名称与含义，TIMEWAIT的作用。 5.TCP拥塞控制。 6.TCP滑动窗口与回退N针协议。 7.Http的报文结构。 8.Http的状态码含义。 9.Http request的几种类型。 10.Http1.1和Http1.0的区别 11.Http怎么处理长连接。 12.Cookie与Session的作用于原理。 13.电脑上访问一个网页，整个过程是怎么样的：DNS、HTTP、TCP、OSPF、IP、ARP。 14.Ping的整个过程。ICMP报文是什么。 15.C/S模式下使用socket通信，几个关键函数。 16.IP地址分类。 17.路由器与交换机区别。

网络其实大体分为两块，一个TCP协议，一个HTTP协议，只要把这两块以及相关协议搞清楚，一般问题不大。

推荐书籍：《TCP/IP协议族》

数据结构与算法

1.链表与数组。 2.队列和栈，出栈与入栈。 3.链表的删除、插入、反向。 4.字符串操作。 5.Hash表的hash函数，冲突解决方法有哪些。 6.各种排序：冒泡、选择、插入、希尔、归并、快排、堆排、桶排、基数的原理、平均时间复杂度、最坏时间复杂度、空间复杂度、是否稳定。 7.快排的partition函数与归并的Merge函数。 8.对冒泡与快排的

改进。 9.二分查找，与变种二分查找。 10.二叉树、B+树、AVL树、红黑树、哈夫曼树。 11.二叉树的前中后续遍历：递归与非递归写法，层序遍历算法。 12.图的BFS与DFS算法，最小生成树prim算法与最短路径Dijkstra算法。 13.KMP算法。 14.排列组合问题。 15.动态规划、贪心算法、分治算法。（一般不会问到） 16.大数据处理：类似10亿条数据找出最大的1000个数...等等

算法的话其实是个重点，因为最后都是要你写代码，所以算法还是需要花不少时间准备，这里有太多算法题，写不全，我的建议是没事多在OJ上刷刷题（牛客网、leetcode等），剑指offer上的算法要能理解并自己写出来，编程之美也推荐看一看。

推荐书籍：《大话数据结构》《剑指offer》《编程之美》

Android

1.Activity与Fragment的生命周期。 2.Acitivty的四中启动模式与特点。 3.Activity缓存方法。 4.Service的生命周期，两种启动方法，有什么区别。 5.怎么保证service不被杀死。 6.广播的两种注册方法，有什么区别。 7.Intent的使用方法，可以传递哪些数据类型。 8.ContentProvider使用方法。 9.Thread、AsycTask、IntentService的使用场景与特点。 10.五种布局：FrameLayout、LinearLayout、AbsoluteLayout、RelativeLayout、TableLayout、GridLayout，各自特点及绘制效率对比。 11.Android的数据存储形式。 12.Sqlite的基本操作。 13.Android中的MVC模式。 14.Merge、ViewStub的作用。 15.Json有什么优劣势。 16.动画有哪两类，各有什么特点？ 17.Handler、Loop消息队列模型，各部分的作用。 18.怎样退出终止App。 19.Asset目录与res目录的区别。 20.Android怎么加速启动Activity。 21.Android内存优化方法：ListView优化，及时关闭资源，图片缓存等等。 22.Android中弱引用与软引用的应用场景。 23.Bitmap的四中属性，与每种属性队形的大小。 24.View与View Group分类。自定义View过程：onMeasure()、onLayout()、onDraw()。 25.Touch事件分发机制。 26.Android长连接，怎么处理心跳机制。 27.Zygote的启动过程。 28.Android IPC:Binder原理。 29.你用过什么框架，是否看过源码，是否知道底层原理。 30.Android5.0、6.0新特性。

Android的话，多是一些项目中的实践，使用多了，自然就知道了，还有就是多逛逛一些名人的博客，书上能讲到的东西不多。另外android底层的东西，有时间的话可以多了解一下，加分项。

推荐书籍：《疯狂android讲义》《深入理解android》

其他综合性的书籍也需要阅读，推荐：《程序员面试笔试宝典》《程序员面试金典》。另外“牛客网www.newcoder.com”是个好地方，里面有各种面试笔试题，也有自己在线的OJ，强烈推荐，还有左程云老师的算法视屏课（已经出书了），反正我看了之后对我帮助很大（这不是植入广告）。

3. 项目

关于项目，这部分每个人的所做的项目不同，所以不能具体的讲。项目不再与好与不好，在于你会不会包装，有时候一个很low的项目也能包装成比较高大上的项目，多用一些专业名词，突出关键字，能使面试官能比较容易抓住重点。在聊项目的过程中，其实你的整个介绍应该是有一个大体的逻辑，这个时候是在考验你的表达与叙述能力，所以好好准备很重要。

面试官喜欢问的问题无非就几个点：

1.XXX（某个比较重要的点）是怎么实现的？ 2.你在项目中遇到的最大的困难是什么，怎么解决的？ 3.项目某个部分考虑的不够全面，如果XXXX，你怎么优化？ 4.XXX（一个新功能）需要实现，你有什么思路？

其实你应该能够预料到面试官要问的地方，请提前准备好，如果被问到没有准备到的地方，也不要紧张，一定要说出自己的想法，对不对都不是关键，主要是有自己的想法，另外，你应该对你的项目整体框架和你做的部分足够熟悉。

4. 其他

你应该问的问题

面试里，最后面完之后一般面试官都会问你，你有没有什么要问他的。其实这个问题是有考究的，问好了其实是有加分的，一般不要问薪资，主要应该是：关于公司的、技术和自身成长的。

以下是我常问的几个问题，如果需要可以参考：

1.贵公司一向以XXX著称，能不能说明一下公司这方面的特点？ 2.贵公司XXX业务发展很好，这是公司发展的重点么？ 3.对技术和业务怎么看？ 4.贵公司一般的团队是多大，几个人负责一个产品或者业务？ 5.贵公司的开发中是否会使用到一些最新技术？ 6.对新人有没有什么培训，会不会安排导师？ 7.对Full Stack怎么看？ 8.你觉得我有哪些需要提高的地方？

知识面

除了基础外，你还应该对其他领域的知识有多少有所涉猎。对于你所熟悉的领域，你需要多了解一点新技术与科技前沿，你才能和面试官谈笑风生。

软实力

什么是软实力，就是你的人际交往、灵活应变能力，在面试过程中，良好的礼节、流畅的表达、积极的交流其实都是非常重要的。很多公司可能不光看你的技术水平怎么样，而更看重的是你这个人怎么样的。所以在面试过程中，请保持诚信、积极、乐观、幽默，这样更容易得到公司青睐。

很多时候我们都会遇到一个情况，就是面试官的问题我不会，这时候大多数情况下不要马上说我不会，要懂得牵引，例如面试官问我C++的多态原理，我不懂，但我知道java的，哪我可以向面试官解释说我知道java的，类似的这种可以往相关的地方迁移（但是需要注意的是一定不要不懂装懂，被拆穿了是很尴尬的），意思就是你要尽可能的展示自己，表现出你的主动性，向面试官推销自己。

还有就是遇到智力题的时候，不要什么都不说，面试官其实不是在看你的答案，而是在看你的逻辑思维，你只要说出你自己的见解，有一定的思考过程就行。

5. 面经

LZ应聘的职位都是android客户端开发。

面经其实说来话长，包括实习的话面过的公司有：CVTE、腾讯、阿里、百度、网易、蘑菇街、小米。最早得追溯到到今年3月份，那时候刚过完年，然后阿里的实习内推就开始了，我基本都没什么准备，就突如其来的接到了人生中第一个面试电话。

阿里实习内推一面

电话面试， 由于是第一次面试，所以非常紧张，项目都没怎么说清楚。然后面试官就开始问项目细节了，这里我关于一个项目细节和面试官有不同的看法，面试官说我这样做有问题，然后我说我们确实是这样做的，并没有出什么错，差点和面试官吵起来，最后我还是妥协了。然后问了我一个怎么对传输的数据加密，我答的很挫，然后面试官就开始鄙视我：你这个基础不好，那个基础不好，那你说说你还有其他什么优势没？Blabla紧张的说了一些……只面了30分钟不到，然后妥妥的就挂了。

经过这次面试突然感觉人生的艰辛，几天后我们教研室的其他同学陆续开始了面试，他们都很顺利，其中我的室友（单程车票）很顺利的拿到了offer，他是个大神，然后我就压力无比的大。制定了整套复习计划，从早上9点看书看到晚上10点。

到了3月15号左右有CVTE面试，第一次面试是群面，比较坑，坐了一个小时的车过去群面了5分钟，没什么好说的。

CVTE实习面

在自我介绍和项目后，面试官开始问一些java基础，object有哪些方法？这个还能说了一些。问hashmap有多大，这个当时一脸茫然，还sb的答了一个65535。然后面试官让我写三分钟内写一个二分查找，当时也是第一次手写代码，并且还计时，完全没经验，最后超时写了出来。中间又问了我一堆基础，都答得不是很完整。最后问我遇到过OOM的情况没有，什么情况下会OOM。这个也没答出来，然后又妥妥的挂了。

这次经历告诉我，我是缺少面试经验，和现场写代码的能力，基础还需要多加强。所以我开始各种准备，在一个月的时间里看了四本面试书（程序员面试宝典、java程序员面试宝典、程序员面试笔试宝典、剑指offer），把所有关于数据结构和算法的东西用代码写了一遍。

然后到了四月初，腾讯来了，我最开始还是非常向往腾讯的，但就当时那个情况，我对自己不报太大希望，觉得能进BAT这样的顶级公司是个奢侈的梦想。

腾讯的面试是在一个5星级酒店里面，逼格高大上，感觉问的东西也比较多，感觉喜欢问智力题，但是我没遇到。

腾讯实习1面

50分钟左右， 面试的时候还是有些紧张的，但是运气好，遇到了一个学校的师兄，他一直叫我不要紧张。几个比较关键的问题：死锁的必要条件，怎么解决，java和c++比有什么优势，java同步方法，activity生命周期，中间让我设计了个银行排队系统，我说了一堆。然后让我写了一个计算一个int里面二进制有几个1，然后我用最高效的方法（n=n&n-1）写出来之后，面试官有点意外，还说没见过这么写的，让我跟他解释一下。后面就是拉拉家常，问我对工作地点怎么看，让我对比qq和微信，一面出来之后，面试官让我留意通知，心想是过了，其实发挥的不怎么好。

就在会学校的路上，都要到学校了，收到了腾讯二面的通知，下午3点。然后我又跑回去二面。

腾讯实习2面

二面是一个很严肃的人，看上去就比较资深那种，一直都不笑，后面才知道是手机管家T4的专家。一开始就问我项目里，心跳包是怎么设计的，我项目里并没有用心跳，然后只能跟他说没做，问我用json传输数据有什么不好（我只知道用哪想过有什么不好）。又问了http和socket的区别，两个协议哪个更高效一点，遇到过java内存泄露没有，用过哪些调试java内存工具，java四种引用。多数都是项目上的东西，基础的东西没问太多，然后感觉自己答的不是很好，很多都不知道，而且还答错了。其实我感觉我应该是过不了的，但是最后我问问题的时候，我让他评价下我的表现，他说不好评价，我自己说了一堆，说在学校里确实见识到的东西比较少，很多东西没考虑全面，然后他表示赞同，和我探讨了一番，我觉得最后这个问题给我加了不少分。二面也面了50分钟左右。

回来后发现我的状态一直没变，而他们二面完了的都到了HR面了，我以为我已经挂定了，后来在一天晚上12点的时候，惊喜的收到了第二天HR面的短信，当晚上几乎高兴得一晚上没睡着觉。

腾讯实习3面（HR）

就是hr面，也就面了十几分钟，聊聊天，问问哪的人，未来什么打算的等等，基本不怎么挂人就不详细写了。

就这样拿到了人生中第一个实习offer。

后面找实习的心就放松了，没有复习了。然后到了5月5号，阿里来了。对阿里也只是想去面一面的心态了，因为已经有腾讯的offer了，就没想太多。

阿里实习1面

面过腾讯之后发现自己已经比较淡定了，面试得时候能够比较好的交谈了。这一面也遇到一个比较好的面试官，能很轻松的和他交流。主要的问题是android的：activity的生命周期、activity的四种启动模式（当时忘了一些没答全）、线性布局和相对布局、多线程请求，java GC算法与GC方法，内存模型，有一个比较特别的问题是问我微信的朋友圈怎么设计，然后我把思路跟他说了，其他的就是问了项目相关的了。还问了我一个觉得技术深度重要还是技术宽度重要，一面感觉还是比较基础的。

阿里实习2面

这一面就比较虐心，碰到一个阿里云的CTO，一上去项目看都不看，直接问我写过多少行代码，我说至少3、4万行，然后他让我写了两个题：一个找素数，一个递归求阶层，对我也算手下留情（他后来让我同学写AVL树的插入算法，想想也是醉了）。后面就各种基础了，java的基础挨个问了一遍，比较关键多线程实现，锁的几种等级等，反射的用法，wait()和sleep()（讨论这个的时候他把我说晕了），Java还好，多数能应付，然后他就开始问c++的了。虽然是基础，但是lz忘了差不多了，什么指针数组和数组指针，虚函数，多态实现（这个我扯到java上了）等等，问了很多，很多都没答上来，然后他说我基础不太好（我想说我简历上写的了解C++，为什么要追着我问TT）。

就这样出来了，本来以为挂了，后面被通知过了。同学都只有2面技术面，我居然多了一面，叫交叉面试，心想这下肯定完了。

阿里实习3面

这一面遇到了后面我去实习时候的部门boss，人非常好，来的时候走的时候都要和我握手，非常的平易近人。这一面还是问项目上的一些东西居多，基础就问了个java多线程，各个排序的时间复杂度、思想。技术问了半个小时，后面半个小时就开始各种聊人生了（@_@），我家是哪的，父母干嘛的，中学怎么样，大学怎么样，等等，完全就不像是技术面嘛（后来才知道，我一个同学一开始来就和他聊人生，还聊过了。再次感叹找工作是看缘分呐）。

阿里实习4面（HR）

阿里hr比腾讯hr面专业，面了一个小时，把我的生活经历趴了一遍，（问了类似你的优缺点，最让你高兴的一件事，最让你伤心的一件事，你的职业规划，你的理想等等，这种，现在想不起来了）也没什么特别好说的。

面完后第二天去圆桌签offer，就这样又拿到了阿里的实习offer。

LZ后面衡量了杭州阿里B2B和广州腾讯MIG，最后选择去了阿里，因为在总部，感觉大boss人比较好，发展前途可能不错，而且留下来的几率比较大，而腾讯是一个分部门，感觉可能不是很有前景（但是后来了解到其实广州腾讯MIG发展前景非常好，环境也非常和谐，我同学去实习的都留下来了。哎，只能感叹选择是个大问题）。在阿里实习的两个月时间也挺愉快的，学到了不少东西，也认识了很好的师兄和主管，只因最后被拥抱了变化没有拿到正式offer。

实习面经就已经写完了，后面是正式找工作的经历，主要是内推比较多：腾讯、网易、蘑菇街、小米，校招就面了家百度。

在阿里实习的时候，面了网易和蘑菇街。

网易面试是我面了这么多中，问得最专业的了。

网易内推1面

电话面，一天在里中午休息的时候面的。这一面我面得很烂，由于在阿里实习，面试官恰好也在阿里呆过，问了我在阿里学到了哪些东西，看过哪些框架，看过源码没有，我支支吾吾说了一些，面试官不太满意（我表示我都说不全啊，在阿里就来了不久，哪那么多时间看源码）。项目各种细节问一通之后，开始问基础，Http报文结构，Handler、Looper模型，ThreadLocal（这个LZ当时没答上来），怎么使service不被杀死，android内存优化，自己实现线程队列模型，问我怎么设计（这个当时被前面的问题问蒙了，直接说不知道了），面了20+分钟，感觉答得都不怎么好，然后面试官问我说还有没有什么比较擅长的他没有问道的，我就把android Framework里zygote的启动和Binder通信说了一遍（这里强行装了一次逼）。

面完之后本以为挂定了，然后师姐跟我说居然过了，也是够神奇，我觉得是我后面补充的内容救了我。

网易内推2面

二面是现场面，就在阿滨江区的隔壁。时间是一天中午，吃了饭就到了隔壁。面试官是个比较年轻人，可能大不了我几岁，也是非常好说话，开始也是聊项目，我把在阿里做的app和自己写的小框架拿出来，他就指着上面各种问，这里怎么实现，会有什么问题，你怎么解决，然后他描述了一个场景说，两个activity，前面的是个dialog activity，怎么在dialog activity存在的情况下改变后面的activity（lz答的用广播）。android怎么解决缓存，要是内存超了怎么办？然后扯到了JVM，GC判定算法与方法，哪个区域用什么GC算法，怎么改进复制算法。然后是基础，也像一面一样问了一些，hashmap和concurrntHashmap的区别、泛型能否强制转换。然后是算法，问了快排和归并的平均时间复杂度与最差时间复杂度，出了个算法题：怎么找到一个随机数组的前50大数、中间50大数，（这个用最小堆和partition函数），复杂度是多少。

面完之后其实感觉还不错，基本都打答上来了，顺利进入三面。

网易内推3面（HR）

hr面也是现场，也聊了很多，问我为什么要从阿里来网易，有什么打算，你看中网易的什么（主要是针对我是在阿里实习来问的，我就讲了一堆网易的优势），让来杭州工作愿不愿意。还跟我说了，这次内推是优中选优，有名额限制，如果没有通过，请继续关注网易校招。

后面让师姐查了下状态，状态显示是三面已通过。但是最后没有收到offer，还是有点小失望。

蘑菇街面试感觉比较基础，没有什么技术难度。

蘑菇街内推1面

电话面，也是在一个中午面的。18分钟，问了一些项目，主要是问基础、问得非常基础：Arraylist与LinkedList区别，String与StringBuffer用法，HashMap与HashTable区别，Synchronized用法等等等等（非常基础），这不一一列举了，然后很顺利的就过了。

2面是在20天后了，也不知道蘑菇街出了什么岔子。

蘑菇街内推2面

也是电话面，CTO面试，就整体聊了项目，我在项目中学到了什么，遇到什么困难怎么解决的，在阿里实习学到了哪些东西，有看过源码么，我的优缺点，我为什么选择蘑菇街，我了解蘑菇街哪些东西。最后答完感觉自己答得还行但是也没有过，不知道为什么。

小米是投的内推精英计划，50个名额，解决北京户口。

小米内推1面

电话面，大概40分钟，面试的时候那边很吵，不过幸好面试官语速慢，而且我答完一个问题后，面试官会和我交流哪里没有答好。没有问项目，就问了基础，问题也不多：HashMap删除元素的方法，for each和正常for的用在不同数据结构（ArrayList、set、hashmap）上的效率区别（LZ表示没有看过源码，不知道），static class和non-static class的区别，一个大文件几个GB，怎么实现复制（这个也没有答好）。然后问了两个算法：之前一个出现过，另一个是在git里面，如果有n个分支，m次commit怎么找到任意两个节点共同的那个父节点（这个当时我想错了，想到二叉树上去了，没有答好）。然后让两个算法用代码实现，1个小时内写好email给他。

小米面了以后也杳无音信，估计也是要求很高，毕竟解决北京户口。

其实在阿里实习的时候很早就开始投简历了，因为出去实习一段时间后，感觉还是很想留在成都（因为lz是四川人）。腾讯我没有参加校招面试，直接走的内推流程。

腾讯1面

电话面，7月20+号，很水，就问了项目，聊了可能有十多分钟，然后面试官说，内推没有什么作用，还是要走校招面试（我觉得他可能是有其他事情，想节省时间），你在实习不能回来，还是要现场面一次才行，然后就留了个电话让我校招联系他，这样就完了。

2面是在我回学校后了。

腾讯2面

9月6号我回学校之后，下午3点接到电话，让我晚上7点去腾讯现场面的（我在想为何是在晚上，lz学校到腾讯要2个小时，还让不让人回来了），当时紧张得要死，因为刚从阿里回来不久，都没怎么好好准备基础，在地铁上看了两本基础书，亚历山大。面试是在腾讯里面，微信部门，面试官是个中年人（现在是LZ的主管），看起来还是比较沉稳的那种。也没问基础技术问题，就聊项目细节和一些可优化的地方，然后把lz的简历看了翻了一遍，问了一遍，然后就是问我在阿里学到了什么，为什么当时选择了阿里（这时候肯定要各种跪舔啊）。然后后来他说他是做ios的，我在想难怪不问我基础。

面完了说一周之内通知我结果，也没报太大希望，感觉并不太对口，因为搞不懂为什么是做ios的来面我。

两天之后，在阿里HRG电话通知我拥抱变化之后，几乎同一时间，腾讯电话通知我拿到了成都offer，我只能感叹太巧了（大概这大半辈子的运气都花光了）。

后来校招开始后，只面了百度一家公司，百度确实比较重视基础与算法，看中技术。

百度1面

大概1个小时，又是个做ios的师兄面试我，自然就只能聊项目了，我给他展示了我做的app后，也问了些技术问题，缓存怎么做的，内存溢出怎么处理。然后两个算法题：把一个数组中奇数放前面，偶数放后面，这个要求写出来。另一个是3亿条IP中，怎么找到次数出现最多的5000条IP。最后问了是否愿意去北京，对于技术的看法。

百度2面

50分钟，写个4个程序题：反转链表、冒泡排序、生产者消费者，这三个都还好写，很快的写出来了，还有一个题是在一组排序数中，给定一个数，返回最接近且不大于这个数的位置，要求时间在O(logn)（这个想了一会，用二分查找，然后特殊处理了一下），最后他看不懂，要我一步一步解释。花了好一整子，最后问了个java反射，就让我走了。百度果然是重视算法。

百度3面

这一面应该是个技术高层，笼统的问了我一下项目的问题，然后问了几个基础：java反射机制；android动画有哪些，什么特点？TCP/IP层次架构，每层的作用与协议；TCP拥塞控制；滑动窗口是怎么设计的，有什么好处；android的布局都有哪些。问完这些之后，然后就是有点类似于HR的聊天了：如果这次面试过了你觉得是因为什么原因，没过呢？你觉得百度怎么样？你对技术路线什么打算？有些和前面重复的就不写了。然后他让我问他问题，我就连续问了5、6个问题，最后愉快的走了。

百度这两天给结果。

6. 写在最后

关于选择

LZ当时实习的时候，杭州阿里和广州腾讯选择去了阿里，但是却因为拥抱变化没有留下来，相反这边在腾讯实习的同学却很顺利。但是也是因为没有去广州腾讯，最后我能留在成都腾讯。选择是一件非常重要的事情，它决定着你的未来，但是也有一点你得知道：塞翁失马焉知非福，现在看起来不太好的选择，不一定将来就好，未来有太多未知数。

心怀感恩

其实一路走来，我也是在成长，从最初的不自信，到了最后面试一切都比较冷静与沉着。我一直相信，机会是留给有准备的人，所以，请提早准备，越早越好。我很感激能有那么多人帮助我和肯定我，没有最初腾讯的肯定，我肯定不会走的这么顺利，所以我很感恩哪些让我通过的人，也感谢我们实验室的兄弟姐妹，给了我良好的学习成长环境，心怀感恩才能好运常在。

找工作其实就像是一场战役，前面我们经历了高考或者考研，现在是找工作，你不在这个时候搏一搏，怎么对得起你之前的努力。不要担心找不到好工作，你要相信：

天道酬勤!

早上11点，刚刚收到阿里的offer，也算是给自己三个月的春招画上了一个还算圆满的句号。

先介绍一下本人的基本情况：陕西普通一本非计算机专业大三学生，非985211。主要技能C/C++/Java/数据结构/算法。

这三个月的经历，首先确实在技术上确实通过不断的面试有了很大的提升，其次在各种面试经验上也有了一些心得。

眼看春招已经基本结束，所以将这段时间的经历写在牛客上，希望能够帮助大家，在秋招上斩获更满意的offer。

所以就拿我收到offer的三家重点说说。（按时间排序）

一、CVTE：

CVTE的实习生招聘非常早，笔试完了就收到通知，预约了面试时间就早早去了，两轮技术面试一轮HR面试，进行的很顺利，3月26号就已经发了offer。

一面：面试官非常亲切，其实我当时是第一次参加现场面试，楼主比较怂，其实现场紧张的不要不要的，自我介绍的时候说完了姓名年龄和专业之后，就卡住说不下去了。面试官看在眼里疼在心上，于是很关心的对我说，没事没事不要紧张，这样吧，我们先写两个算法放松一下（Excuse me？）不过好在面试官出的题很简单，第一个是二分查找，我用递归和非递归各写了一遍，重点就在于下标的控制；另一道是在N个数中求前M大个数，其实也很简单，思路就是使用快速排序的思想，每一次当把一个数字放在正确的位置上的时候跟M进行比较，其实在剑指offer上有原题。好在寒假的时候把剑指offer刷了很多遍，所以很快也写出来了。我个人觉得在写代码之前，有很多事情需要跟面试官进行交流，比如函数的参数、返回值、还有一些异常情况的处理，提前跟面试官约定好。比如我在写代码之前，就问面试官：假设参数是`(int *arr, int length, int m)`，在这种情况下，可能会有四种情况会导致程序出现异常情况

- arr ==NULL
- length <= 0
- m > length
- m <=0

询问他在这四种情况下我们该如何处理？"面试官听完之后一下就有兴趣了，及时跟面试官沟通，一方面是防止思路与面试官预期的差距太大，面试官出的题，因此他有责任让面试者明白他的意思，另一方面表现我们积极思考，向面试官展示我们的思考能力。可能之前的算法写的太顺利，所以给自己建立了蜜汁自信，并且面试官对我的第一印象也好，后面的问题回答的很轻松，有struts2和SpringMVC的区别、Spring中IoC和AOP的理解，不过在数据库方面被难住了，在MySQL中如何定为查询效率较慢的SQL语句，比如慢查询日志、EXPLAIN关键字还有PROFILES等。但是总得来说，一面进行的很顺利。

二面：CVTE的面试流程是，如果一面的面试官觉得通过了，就会示意现场接待的姐姐，姐姐会安排在场外休息一下，二面大概在十分钟之后进行；而如果觉得不符合要求，姐姐就会亲切地告诉面试者，今天的面试结束了，可以先回去等消息。等了十分钟之后，二面如约而至，二面是一个年纪稍大，但是很有风度的中年人，事后学长说那是他们部门的BOSS，面试官让我设计了一个场景，青蛙爬井，就是画画UML，两个类图，和他们的关系。最后扩展成面向接口的思维，不得不说BOSS确实功力深厚，纠正了我很多问题，最后才勉强满意。然后就是分析项目，挑了一个我比较熟悉的，问了很多问题，比如页面的跳转关系、我所做的功能模块，让我一边画图一边解释，我自认项目准备的还算充分，因为都是自己做的，所以这部分也算顺利。后面就没有什么技术问题，问了一下我什么时候能来实习，还有在校的经历，同学之间是如何评价我的，然后就结束了。

HR面：晚上回到宿舍就有短信通知，第二天参加终面，当时还在跟女朋友看电影啊，荒原猎人。正看到小李子跟熊搏斗，女朋友吓得不敢看，广州的电话来了，让我预约第二天的终面并且填一个单子。慌慌张张跑回去完成。印象里面那个问卷问的很全面，比如家庭状况、为什么选择去广州、什么情况下会放弃这份工作、小时候印象最深的

一件事情、列举出近期让你伤心的事情以及你是如何处理不良情绪的。如果各位到了这一步，一定要谨慎作答，这些问题都会列入到综合评测中。后面的面试就是把这些问题现场问一遍，说是HR面，但我总感觉是高管面，一个高管面三个面试者。

面试出来之后，等做到地铁上，广州负责的CVTE校招的学长就已经告诉我通过了，效率非常高。3月26号拿到的offer，这是春招的第一个offer，当时的心情还是很激动的。

二、百度

百度是学长内推，技术面是两轮技术面试。

一面：简单的自我介绍，因为是电话面试，所以流畅了很多（你们懂的）。一个小时满满的技术问题，所以就不用向上面再赘述了，直接上干货

1.是否了解动态规划

动归，本质上是一种划分子问题的算法，站在任何一个子问题的处理上看，当前子问题的提出都要依据现有的类似结论，而当前问题的结论是后 面问题求解的铺垫。任何DP都是基于存储的算法，核心是状态转移方程。

2.JVM调优

其实我没有实际的调优经验，但是我主要介绍了一下JVM的分区、堆的分代以及回收算法还有OOM异常的处理思路

3.分别介绍一下Struts2和Spring

不用多说，这方面比我答得好的同学肯定大有人在，就不出丑了

4.职责链模式（设计模式）

GoF经典设计模式的一种

5.实践中如何优化MySQL

我当时是按以下四条依次回答的，他们四条从效果上第一条影响最大，后面越来越小。

- SQL语句及索引的优化
- 数据库表结构的优化
- 系统配置的优化
- 硬件的优化

6.什么情况下设置了索引但无法使用

- 以“%”开头的LIKE语句，模糊匹配
- OR语句前后没有同时使用索引
- 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）

7.SQL语句的优化

- order by要怎么处理
- alter尽量将多次合并为一次
- insert和delete也需要合并

8.索引的底层实现原理和优化

B+树，经过优化的B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

9.HTTP和HTTPS的主要区别

10.Cookie和Session的区别

11.如何设计一个高并发的系统

- 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引的优化
- 使用缓存，尽量减少数据库 IO
- 分布式数据库、分布式缓存
- 服务器的负载均衡

12.linux中如何查看进程等命令

13.两条相交的单向链表，如何求他们的第一个公共节点

很简单的链表题目，博客上的做法一搜一大把，我记得当时答在兴头上，又给面试官解释了一下如何求单向局部循环链表的入口，链表中很经典的问题（其实链表也就那几个常用算法，比如逆制、求倒数第K个节点，判断是否有环等）

大概八十分钟吧，最后问面试官有没有对我的意见或者建议，面试官说觉得我今晚的面试表现比简历上写的更出色。。。对面试官的好感度瞬间飙升

二面：可能一面面试官对我的评价还算不错，二面面试官一口气考了我11个设计模式（手动微笑），对，是11个设计模式，有直接提问，也有在场景设计中引导我使用。总共加起来11个，分别是：单例模式、简单工厂模式、工厂模式、抽象工厂模式、策略模式、观察者模式、组合模式、适配器模式、装饰模式、代理模式、外观模式。然后就是设计一个公司下有部门、部门下有经理和员工，经理可以管理经理和员工这样的一个模型，组合模式一套用就行了。后面还问了几个非技术问题，比如和产品、测试发生矛盾了怎么处理，答应的任务发现完成不了该如何处理等，大家如果遇到请随意装逼。

百度给我最大的印象就是每次新的面试，面试官一定会问什么时候能来实习，能够实习多长时间，答案符合要求了才进行下面的面试。据说百度今年要求6个月的实习时间，回答两三个月的都再无下文。

机智的我不管哪家公司问都是说从即日起到大四毕业，中间出了期末考试和毕业设计，都能参与实习，近期就能参加实习。不是故意想骗人，只是目前还没有和HR谈条件的筹码，这样说了就算最后去不了，也算是一次面试机会，如果一开始就拒绝的话，连面试机会都没有。等到技术面试结束了，跟一开始比，就有了谈条件的筹码，这个时候大家再根据情况合理要价

三月的CVTE、四月的百度，现在该说五月的阿里了

三、阿里巴巴

阿里巴巴其实同样的简历，在内推阶段直接简历被刷，非常尴尬。但是有幸笔试蜜汁通过，所以收到了参加现场面试的通知。

一面：一面的面试官长得超级像张家辉，整个面试过程，我满脑子都是《激战》里面的老拳王，当他朝我提问的时候，我就想到激战里面的经典台词“怕输，你就会输一辈子”。不知道面试官老师有没有看出我表情的异样~闲话不多说，上干货。

1. 二叉树的遍历方式，前序、中序、后序和层序

二叉树本身就是一个递归的产物，那前序举例，访问根节点，然后左节点，再右节点，如果左节点是一棵子树，那么就先访问左子树的根节点，再访问左子树的左节点，依次递归；而层序，使用队列进行辅助，实现广度优先搜索

2. volatile关键字

给大家推荐两本书：《Java多线程实战》和《Java并发编程的艺术》，这会儿已经三点了，脑子有点乱书名可能未必无误。对Java实现多线程描述的非常详细。现场跟面试官老师扯了很多，我在这里挑主要的说

volatile关键字是Java并发的最轻量级实现，本质上有两个功能，在生成的汇编语句中加入LOCK关键字和内存屏障

作用就是保证每一次线程load和write两个操作，都会直接从主内存中进行读取和覆盖，而非普通变量从线程内的工作空间（默认各位已经熟悉Java多线程内存模型）

但它有一个很致命的缺点，导致它的使用范围不多，就是他只保证在读取和写入这两个过程是线程安全的。如果我们对一个volatile修饰的变量进行多线程下的自增操作，还是会出现线程安全问题。根本原因在于volatile关键字无法对自增进行安全性修饰，因为自增分为三步，读取-》+1-》写入。中间多个线程同时执行+1操作，还是会出现线程安全性问题。

3. synchronized

- 锁的优化：偏向锁、轻量级锁、自旋锁、重量级锁
- 锁的膨胀模型，以及锁的优化原理，为什么要这样设计
- 与Concurrent包下的Lock的区别和联系

Lock能够实现synchronized的所有功能，同时，能够实现长时间请求不到锁时自动放弃、通过构造方法实现公平锁、出现异常时synchronized会由JVM自动释放，而Lock必须手动释放，因此我们需要把unLock()方法放在finally{}语句块中

4. concurrentHashMap

两个hash过程，第一次找到所在的桶，并将桶锁定，第二次执行写操作。

而读操作不加锁，JDK1.8中ConcurrentHashMap从1600行激增到6000行，中间做了很多细粒度的优化，大家可以查一下。

5. 锁的优化策略

- 读写分离
- 分段加锁
- 减少锁持有的时间
- 多个线程尽量以相同的顺序去获取资源

等等，这些都不是绝对原则，都要根据情况，比如不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。这部分跟面试官谈了很久

6. 操作系统

这部分基本是跪着跟面试官交流的，因为非计算机专业，对这个楼主确实比较欠缺

不过好在前面的表现还可以，顺利通过了。

二面：十分钟休息，二面面试官先问了一些无关紧要的问题，比如学校的专业课、平时如何学习新技术等等，然后切入正题，让我选一个熟悉的项目，三分钟画出大体架构图，我在项目部分的准备还算充分，但是面试官真的水平非常高。

在项目部分，可能是我整个阿里面试过程中最提心吊胆的，缓存的使用，如果现在需要实现一个简单的缓存，供搜索框中的ajax异步请求调用，使用什么结构？我回答ConcurrentHashMap，可是内存中的缓存不能一直存在，用什么算法定期将搜索权重较低的entry去掉？我说先按热度递减放进一个CopyOnWriteArrayList中，保留前多少个然后再存回ConcurrentHashMap中，面试官说效率过低，有没有更高效的算法，我假装冥思苦想（用假装其实是因为，确实想不到方法）

后来面试官说其实这个问题有点难了，换一个，又跟我扯到线程的问题，大体就跟一面面试官问的差不多，就不赘述了。这部分感觉面试官还比较满意，就问题TCP如何保证安全性，我说三次握手、四次回收、超时重传、保序性、奇偶校验、去重、拥塞控制。还讲了滑动窗口模型。

后面又考了一些红黑树的问题，问到B+数，还有JDK1.8中对HashMap的增强，如果一个桶上的节点数量过多，链表+数组的结构就会转换为红黑树。

面试官问我项目中使用的单机服务器，如果将它部署成分布式服务器？我当时心里一惊，这个问题确实没有准备过，眼看就要被问死了，临时抖了个机灵，说有一次跟一个师兄尝试这么做的时候，遇到了session共享问题，然后成功地把面试官引向了session共享的问题，跟他讨论了10分钟左右的分布式系统中如何做到session共享。后面面试官可能也觉得我这部分

手写一个线程安全的单例模式，经典的不能再经典，没什么好说的，懒汉饿汉随便选一个。

还有一些MySQL的常见优化方式、定为慢查询等，回答的七七八八，之前面试总结的问题还有印象，所以感谢自己有面试完及时总结的习惯。

最后问了问我平时都如何学习、最近都在看什么书，来实习的话学校的考试如何解决等等。

面试官告诉我他的问题已经问完了，我看没有让我提问的意思，所以我起身跟面试官握了个手（我在参加现场面试的时候有这样的习惯，握手的同时跟面试官强调，“我很珍惜这次面试机会”）

二面出来之后挺紧张的，感觉自己答的还是有很多漏洞，可能面试官虽然发现了，但是觉得我态度不错，虚心学习，所以还是很幸运到了HR面，HR面就不多说了，只要不跟HR乱提要求，比如不考虑某某城市之类的作死要求，再跟HR好好谈谈我们对知识的渴望、希望得到锻炼的决心，我觉得都没什么问题（网易除外，都是泪）

我还重点给HR讲了一下我对项目的反思，哪些地方可以做的更好。又把一次学习新技术时间又很紧的尽力夸大了一下，HR听完之后表示非常羡慕我们这样搞技术的，感觉每天做的事情都很有激情。

最后就是经过三天的等待，顺利拿到阿里巴巴的实习offer

楼主从三月初到现在，基本能叫的出来的公司都参加了各种面试，很惭愧拿到offer的只有这三家。但是经过大大小小二三十次的面试，我觉得对一个后台程序员来说，重要的不只是语言，还有数据结构算法、网络基础、并发、数据库、设计模式、操作系统、linux等等很多很多技术需要掌握。我就有很强烈的感觉，单论Java，在楼主的周围其实有很多比楼主强得多的人，可是他们有的人面试一直不顺利，原因就在于其他的知识点相对薄弱。这点在阿里巴巴面试中就体现的很深刻。最后HR问我作为非计算机专业学生，什么专业课没有学到最让我遗憾？我回答网络基础、操作系统、计算机组成原理和系统的数据库知识体系。虽然侥幸拿到了阿里巴巴的offer，但这一次的面试让我深深地看到了自己差距。跟二面面试官交流的时候，他考我项目，我就拼命想把他往框架上拉，想解释hibernate和Spring还有mybatis，结果面试官一次也没有上当。每当我说这些的时候，面试官就会打断我，说我不用解释框架，我们就建立在这些东西都双方都清楚的基础上。所以真心劝各位准备面试的朋友们，多重视基础。基础能够决定学习能力和思维方式，而学习能力和思维方式最终决定一个程序员能走多远。

以上是我对春招面试的部分总结，手上还有十份左右的手写面经，都是每次面试完当天晚上自己总结的，有需要的朋友可以私信我。

最后祝看完这篇文章额所有朋友，找到自己心仪的工作，程序员都不容易。

最后的最后，牛客在过年到现在这段时间对我的帮助非常大，有我寒假怒刷2000题，也有后面疯狂提交代码（疯狂报错），是金子总会发光，在我的带动下，整个实验室都在刷题，总之祝牛客网越办越好

1. 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 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。服务部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，黑马的每一位达人，在黑马拼命码代码的同事，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！

2. 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”:funk:，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、TableLagout

Q：请介绍下Android的数据存储方式。 A：SharedPreference、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。同时将我们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FragmentSupportManager或者FragmentManager，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，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RunntimeException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RunntimeException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吧这些问题给解决了。

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Davlik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。我们用用的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大图片也展示出来。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WeakRefrence(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!

这家公司所在楼有两层是高德地图的!

阳哥冒死偷拍的笔试题!

阳哥录用通知邮件!

3. 挑战公司No.3：中阜投融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开发的各种视频，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发的框架，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淋尽致，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。

录用Offer!

4. 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、TableLagout。

Q：请介绍下Android的数据存储方式？ A：SharedPreference、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：

- Android Interface Definition Language
- 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。
- 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：

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    SystemClock.sleep(1000);
                }
            }
        }).start();
    }
```

```
}
```

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已经做好了的，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！

5. 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

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：

```java class A{ int a =1; double d = 2.0; void show(){ System.out.println("Class A:a="+a+"\td="+d); } } class B extends A{ float a = 3.0f; String d = "Hello World!"; void show(){ super.show(); System.out.println("Class B:a="+a+"\td="+d); } }
```

(1)若在应用程序的main方法中有以下语句：

```java
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，这是也是为节约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“类装载器有载入类的需求时，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”。

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。onMesure、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，顾名知义就是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registReceiver方法来注册。Service运行在后天进程中，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Jave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：

```
typedef struct {
    const char* name;  //Java中函数的名字
    const char* signature;  //用字符串描述的函数的参数和返回值
    void* fnPtr;  //指向C函数的函数指针
} 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。

录用Offer!

笔试题（在前台眼皮底下偷拍~）

6. 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。

- 个人基本信息（姓名、年龄、老家、居住地等）
- 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
- 自己为何来贵公司面试
- 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.findviewByld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findViewByld(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。

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！

个人近况

本人博客，先说下博主最近近况，今年 2 月份从信和财富出来，去了一家创业公司结果不堪 996 的压榨，5 月底毅然离职没想到目前市场这么萧条，怪自己太作，有好的机会不好好把握，非得出来受虐哈，人都是犯贱的……所以目前整理几家去过的公司以免以后被坑。

金开门（好贷网旗下孵化创业公司）

这公司是在 BOSS 直聘上投的

总体面试还算不错吧 Android 技术那面一般也不会问特别深主要是最新的主流技术一般会问下，还有就是之前的项目会大致问一下

接下来是总监面，总监是个蛮不错的人，满有亲和力的，大概就是聊推送这一块的，还有支付，因为这公司主要业务是聚合支付相关的，总体还 OK

接着是 HR 谈薪水还有介绍公司近况，貌似最近一直是 995 的节奏

最后是大 Boss 面貌似很屌的说了一句目前我们就是 996 的节奏（应该是试探我的），我觉得跟他也没啥好谈，他一幅咄咄逼人的气势，总体感觉 BOSS 应该是个坑比，这个人的感觉貌似跟博主之前在16年遇到的创业公司的老板一个鸟样，所以就没有后续了...

音悦台

这个也是在 Boss 直聘上约的，公司就在三里屯 SOHO 公司主要业务主打 MV 的剩下的我就不多说了，前几年业务还是挺火的

HR 人还是很不错的，公司的环境神马的都没得说，妹子也多，没给offer确实感觉挺遗憾的

首先光技术面问的就蛮深入的，基本最近貌似招人都比较苛刻~多线程，线程池，handler，Looper 源码层，activity 源码，四种启动模式，生命周期，View 的绘制流程，自定义 view，手势传递问的最复杂也最多 还有一些开源项目相关的问题吧 okhttp，glide，eventbus 相关的

1905 电影网

这个是在拉勾上投的，公司在西直门 我敢说这个面试官是这么多年我遇到的最能装逼一个，当然人家技术也蛮不错的，你不会的，遇到问题的，人家也耐心给你讲解哟，无形装逼，最为致命啊！如果你技术不是很好的话千万不要去这家公司找虐

博主之前有个朋友也来过这家面试，貌似最后给说开不了他的工资，还跟他说来面试很多 给我种感觉 就是面试造核弹，工作拧螺丝？最后还问了我项目里有啥亮点 问题蛮多的好多都忘了，大致记住几个

Glide ， Picasso 都分别有几个线程池

Asynctask 源码，为什么 android4.0 以后是串行

OnMeasure 方法几个参数对应含义（这个题问的最多的所以我把答案贴上O(∩_∩)O~

首先我们要理解的是 widthMeasureSpec, heightMeasureSpec 这两个参数是从哪里来的？onMeasure() 函数由包含这个 View 的具体的 ViewGroup 调用，因此值也是从这个ViewGroup 中传入的。这里我直接给出答案：子类 View 的这两个参数，由 ViewGroup 中的 layout_width，layout_height 和 padding 以及 View 自身的 layout_margin 共同决定。权值 weight 也是尤其需要考虑的因素，有它的存在情况可能会稍微复杂点。

了解了这两个参数的来源，还要知道这两个值的作用。我们只取 heightMeasureSpec 作说明。这个值由高 32 位和低 16 位组成，高 32 位保存的值叫 specMode，可以通过如代码中所示的 MeasureSpec.getMode() 获取；低 16 位为 specSize，同样可以由MeasureSpec.getSize() 获取。那么 specMode 和 specSize 的作用有是什么呢？要想知道这一点，我们需要知道代码中的最后一行，所有的 View 的 onMeasure() 的最后一行都会调用 setMeasureDimension() 函数的作用——这个函数调用中传进去的值是 View 最终的视图大小。也就是说 onMeasure() 中之前所作的所有工作都是为了最后这一句话服务的。

我们知道在 ViewGroup 中，给 View 分配的空间大小并不是确定的，有可能随着具体的变化而变化，而这个变化的条件就是传到 specMode 中决定的，specMode 一共有三种可能：

MeasureSpec.EXACTLY：父视图希望子视图的大小应该是 specSize 中指定的。

MeasureSpec.AT_MOST：子视图的大小最多是 specSize 中指定的值，也就是说不建议子视图的大小超过 specSize 中给定的值。

MeasureSpec.UNSPECIFIED：我们可以随意指定视图的大小。）

- 广播怎么不跨进程
- Rxjava 操作符
- Rxjava 1 2的区别
- 还有问了轮播怎么让用户按下三秒之后继续翻页
- 还有五种进程级别
- 多线程下载，3个线程如何下载10M的文件
- 两列 Recyclerview 如果是表格布局怎么添加 header view
- Thread 和 intent service

最牛B的一个问题是类似天猫这种大厂APP实现的全局应用代理是怎么实现的 （本意就是类似于推送的时候处理推送的逻辑不写一大堆switch case，而是在入口处动态去配置就可以）

凡普金科（普惠金融旗下）

这个是在拉勾上投的，公司在银河 SOHO

当时面试地点其实是发的有问题的，前台大门明明在A座嘛，你非得发个 D 座那边的位置，结果那边的门锁了，我敲了半天才有人开，我才知道走错门了应该从 A 座的电梯上来，可是就是发的 D 座，这里吐槽下。。。然后前台妹子给我的笔试题居然是 Java 的（貌似给错了）

面试的深度基本跟 1905 那哥们差不多，也是 activity 启动模式跟手势传递还有 Looper 的源码那块问的比较多只是这个人最后问了一个尺子的效果：附上项目地址哈，类似这个地址 demo 的实现效果只是年龄换成了金额（毕竟是做金融的公司 当然这样咯）

只说还有复试，但是也是没下文了

映社（木蚂蚁）

这个公司绝对是坑比中的战斗机，去了就让你一直等啊等，等到花都谢了的那种 PS：他们现在的项目主要是做直播的产品叫“映社”（有种抄袭映客的嫌疑~~） 去的时候公司前台都没人，打电话也没人接，后来一个快递小哥进门了我和另外的一个也是面试的才进去

首先是有笔试题的话说蛮弱智的（做完感觉也不会怎么看，完全就是浪费时间啊啊）然后那哥们把你领到一个类似小会议室的屋子里，这哥们给人的感觉技术也很一般，没有之前面的那么强势，基本都是照着简历问的，问直播跟FFmpeg那块偏多，贝塞尔曲线？自定义 View，偶尔穿插下 retrofit，Rxjava，热修复神马的，面完之后就出去了让你一直等啊等，等了快 40 分钟的时候进来说总监在开会

这个公司真特么的是个奇葩，你约人的时候不会挑个没会的时间么，貌似拉勾上有个面php 的哥们跟我一样也是被搁置一直等啊等，真是日了狗了！最后来了一句改天复试吧

只说还有复试，让我来我也不会来了

Melons (北京知行远科技)

这家公司是我在拉勾上投的，公司成立于2016年太初创了（我能怎么办，我也很绝望啊，貌似最近拉勾的公司比较少，稀里糊涂就投了

Boss 也是做 android 的，而且还是前最美应用的联合创始人，技术出身还是蛮不错的公司早 10 晚 8 做海外项目

但是目前的状况是跟别人挤在一间办公室里，那个隔壁组的貌似是 Google 天气的团队

技术面还是跟之前的那几家差不多，基本都不会看你做过的项目就咔咔的问底层源码咯，唯一不同的是启动模式那块多问了 taskAffinity 这个属性，我确实是没用过，面试官拿着 macbook 一个一个的循序渐进的问着，面试流程大概一个半小时左右，然后跟 boss 聊了聊薪资和之前为什么离职，因为是早上十点半约的，一直聊到了中午 12 点 40多

我中午饭都没吃，然后紧接着就去中关村准备下午那家的面试

PS:今天还下着雨，挺苦逼的

目测不会发 offer，可能是小公司给不起薪资

NewsDog (公司名字就叫这个薪资标的还挺高)

这家公司是我在拉勾上投的，公司应该是 B 轮了已经 因为是约的是下午两点，而且刚从 Melons 那里面完就来了，所以去这家公司的时候连中午饭都没吃，让前台给接了杯水暂时压压惊。。。

看简介公司应该是做海外市场主要是信息推荐跟数据挖掘的业务（不知道他们现在的产品是啥）

技术面主要是根据简历去问的，比较在意内存泄漏，内存优化还有 View 的过渡绘制这一块的东西，还有就是问了问图片开源库 Picasso v/s Imageloader v/s Fresco vs Glide 区别以及如何去选择吧，还有 eventbus 的源码以及注解的优点，其它的大概就是还问了问项目的难点之类的

比较操蛋的是没有讨论薪资，然后就直接送客了，不造差在哪里

这样的公司也是比较无语的，面试官给人的感觉是屌的一逼，有点高高在上了

曙光无限

曙光无限 这家公司是在 boss 上约的，公司地址在回龙观东大街的腾讯众创空间（办公楼的环境蛮好的），公司主打产品是海外的项目，旗下产品几十种还是蛮多的

- 第一面： 只是人事先照着简历初略的聊了聊以前的项目经验，由于公司是做海外滤镜软件的，可能对图片算法这块要求蛮高的，福利这块目前是采取接近避税的方式，第一年还不给交住房公积金，貌似还需要第二面总监面，而且还要上机写demo...
- 目前 android 行情： 从以往的面试分析来看基本 android 的行情接近饱和状态，薪资这块基本稍微要高点的话直接就给你 pass 然后可能用其它人候选人去对比，市场的行情还真是惨淡
- 后续： 没有通知进一步的面试

遇见科技

遇见科技 这家公司是在 boss 上约的，公司地址在知春路附近，公司的办公环境也还不错哈，项目应该是一款社交软件，貌似起步还是蛮早的，已经做了几年了

第一面： 主要是技术面，问的以简历的内容为主还有面试官会看以往做过的项目（现在看项目的公司确实不多了）比较在意的是之前做过的项目整体的流程，整体架构设计模式还有业务这块的详情，基本都是围绕做的项目这块的技术点来的涉及的知识点也基本涵盖了目前比较流行的开源组件，还有会问一些关于同类框架之前的区别与对比：比如 volley 与 okhttp，图片框架，数据库 greenDao，realm，litepal 等等性能方面的问题

第二面： 第一面没什么问题之后会和 HR 进一步沟通，主要介绍了公司目前的产品方向还有项目节奏，福利待遇神马之类的

总监面： 能见到总监也基本很不容易了，基本也是聊了聊以往的项目，可能比较看重的是解决问题的能力，会问擅长哪方面（Ui 还是业务？）项目难点等等。。。

后续： 没有通知是否给 offer（难道是薪资问题？？现在市场要到 20 K左右貌似就要考虑考虑了）

邻动

邻动 这家公司是在 boss 上约的，公司地址也是在知春路附近，公司的办公环境没的说，门口摆着各种零食饮料，面试等待的过程，前台妹子还给了一杯饮料喝，公司主要做视频方向的项目，目前已知产品叫“快牙”

第一面： 主要是技术面，基本问的跟之前遇到的问题一样，其中回答的不是很好的问题户要是 MessageQue 的源码实现（我回答错了，应该是链表）还有自定义线程池（应该是问线程池那几个参数），但是公司的技术要求可能希望更倾向于有 FFmpeg 相关经验还有做过视频剪切，裁剪之类的经验吧，问完就送客了....内心其实还是挺喜欢做视频这块的项目

元宝亿家

元宝亿家 这家公司是在 boss 上约的，公司地址在东直门，去了直接在前台填表，然后一个目测像总监的人直接面试，他们现在的项目是采用 MVP 写的应该是想找个人快速接手

第一面： 主要是技术面，问的东西感觉还好，但是感觉自己发挥的不是很好，Java String 类的底层源码，Hashmap 实现原理，Android 广播 Service 相关的，ANR，gson 高级用法（比如序列化的时候如何排除某个字段），项目里用到的设计模式，android 手势机制用到了什么设计模式（是责任链模式，这个我回答错了），内存泄漏和内存溢出，子线程不能更新 view 的机制，Rxjava retrofit okhttp，给我印象比较深的是问了 mac 上 pwd 这个命令是干嘛的（我用了这么久mac 确实没有用过这个命令，是显示当前文件全路径的）还有用没用过 Home brew，最后问了问 Git 相关的命令 pull 跟 fetch

- git rm a.a 移除文件(从暂存区和工作区中删除)
- git rm --cached a.a 移除文件(只从暂存区中删除)
- PS:技术很耐心的给我讲解了我没能答对的问题

- 总结： 感觉自己跟目前市场上需求真正意义的 Android 高级工程职位还是有一定差距的，好多东西还是欠缺好多，还要继续恶补了，fighting...

约了第二天复试，信心严重受挫，不知道能不能谈拢...

写在前面的话

我从 14 年毕业到现在一直待一个三线城市，就用 C 市 代替吧。地方很小，适合居住，但不适合 it 开发，城市很小、圈子很小，it 不发达，想要在 it 上面有出路的还是得去北上广深大城市。我在这个城市呆了三年左右由于自己的一些私事所以趁机就出来想找个大城市呆呆，原本打算去其他城市的，后来稀里糊涂的来到了杭州，在朋友这呆了半个月，直到找到工作。我是 17 年 3 月 25 号就辞职了，递交了辞职申请之后然后就跑去云南玩了一圈之后才想到要找工作的，然后就来杭州了，以上就是大概背景，接下来就写写关于在杭州找一份关于 Android 开发的工作中所遇到的人和事，不看不知道，原来世界真的很大各种人都有，果真印证了一句话：林子大了，什么鸟都有。

简历

面试之前，当然得准备一份简历啦，我的简历是当年刚毕业的时候写的一份简历，这里面用到的模板是 乔布简历 http://cv.qiaobutang.com/ 里面的简历模板不错（哈哈，这不是给它打广告的，我一直用这个，感觉不错就推荐了）。简历模板找到了，下面就是内容了，俗话说，要想找到好工作，一个好的简历必不可少的。因为公司越大的话，投递的人肯定越多，HR 筛选的时间就少了，所以简历有亮点就能打动HR，这样才能有面试资格，有了这个面试资格后才有可能得到这个工作机会，有的人写了简历投给公司后，就像石沉大海一样，毫无音讯，所以，如果有小伙伴，投了简历但是没有回应，不妨修改一下简历，但这里修改简历不是要你去造假，这里面有个梗，待会说~，写完了简历，接下来就是投简历了，有几个渠道可以找工作：

- 内推，内推的质量是最高的，但也是最难的。
- 第三方招聘网站，如 拉钩，51Job，智联招聘，猎聘同道等。

就以我而言，使用上面四种方式进行对比，拉勾网上面公司质量还是不错的，但是HR筛选简历这关有点问题，里面给出的筛选不通过理由都是一样的。51job 和智联招聘两者类似，都差不多，我就是在智联招聘上面找到工作的。猎聘同道里面猎头比较多，我第一次面试就是上面的猎头进行联系的。总的而言，前三个多投投简历，重点放在智联招聘和拉勾网上面，其他也可以稍微投投~

面试

经过上面简历这个步骤，相信我们能够接到一些公司的面试邀请的，在接受公司面试邀请之前，我们得复习面试中所遇到的一些基本知识，主要有 Java 和 Android 这两方面的面试知识。

- Java 基础知识，主要有面向对象三大特性及理解，接口与抽象类、泛型、线程池、集合框架、设计模式、常用算法等知识点。
- Android 知识，主要就是一些常用的知识，待会儿给出一些链接。

以上是专业知识准备，还得准备一些其他人文方面的知识，譬如自我介绍啊、兴趣爱好啊之类的。

有了上面的知识基础，我们就可以上面进行面对面接触啦，我总共大概用了 10 天左右时间，面试了大概 15 家公司，其中有三家是明确拒绝我的，还有四家是我明确拒绝他的，还有几家我对比了一下，然后选择了一个性价比比较高的公司的。在这些公司里面，花样百出，有的公司不知道怎么想的，想花一年工作经验的工资找一个三年工作经验的人，这是典型的想得美。还有的公司忽悠你，就是变相的让你加班，我问他工资能给多少，他说看你能力而定，能力多大，工资多高，我说具体个数，如果我面试通过了，你能不能给我准确的数，他就不说话了，而且是早上 10 点上班，晚上 9 点下班，呵呵，不评价，忽悠应届生呢吧~

印象最深的一家公司，地址是 https://www.lagou.com/gongsi/191783.html 没错，里面的评论就是我评论的，刚进去，给人的感觉，公司环境还不错，宽敞明亮的，然后 HR 给了我 A4 纸，正反面，填写个人的信息，详细程度令人咋舌。好不容易花了几分钟填完之后又给我整整三张面试题，没错，是整整三张，题目很多很多，让我做，哎，

我也好忽悠，第一次碰到这种情况，所以就按部就班老老实实的做完了，花了 20 分钟，做之前还把我的手机给收了，娘的，当成学校考试呢啊？？？更奇葩的还在后面，做完面试题目之后，就开始了面试，那个面试官好像是子公司分责人吧，类似于总经理吧，看着我的简历，竟然问我有没有造假？？？WTF！！！还跟我说，他要想查的话很快就能查到了，我就无语了，我的简历竟然能让他怀疑我造假了，我的简历是有多雷人啊！！！接着就开始问我各种知识点，回答出来 95 以上吧，有几个平时没接触过，所以不知道怎么回答，最后面试结束了，没什么问题就开始讨论工资的问题，他看了我的期望薪水，问我，为什么翻了一倍？我跟他说，我以前呆的城市，非常小，基本连三线都不到，房价只有几千块，跟杭州能比吗？？？然后他就无语了，我就问他，为什么杭州房价比 C 房价高出 4~5 倍，你还想工资都差不多？？？面试简章上面的薪水范围跟实际给出的范围严重不符，我估计这家公司就是想把人先忽悠过去，然后开始各种压价，这太他么的可耻了，最后果断被我给拒绝了，而且是当面拒绝，没有留有情面，给再多也不会去的，这是情怀问题，感觉对程序员不尊重！！！以后大伙找公司，注意这家公司，过来人的经验~

上面就是我遇到的印象比较深刻的一家公司。接下来我们就来总结一下面试过程中提出的各种问题，如果有需要的小伙伴可以参考一下。

面试问题

关于人文方面的问题

- 先介绍一下你自己？
- 你有什么兴趣爱好？
- 你平常空闲时间会干什么，看哪些书，有什么心得体会？
- 你为什么要从上家公司离职？
- 如果面试过了的话，就会问你的期望薪资，然后就开始各种压榨你。

关于 Java 方面的问到的知识点

- 面向对象的三大特性，如何理解其中的多态？
- JVM 的内存模型？
- String、StringBuilder、StringBuffer 的区别，StringBuffer 是如何实现线程安全的？
- 了解过 HTTP 吗？说说它的特点，它里面有哪些方法，有了解过吗？知道 HTTPS 吗？这两者有什么区别？
- 你平常是怎么进行加密的？MD5 加密是可逆的吗？
- 接口与抽象类的区别？static 方法可以被覆盖吗？为什么？
- 创建线程的方式，他们有什么区别？知道线程池吗？说说对线程池的理解？
- 你了解过 Java 的四种引用吗？分别代表什么含义，他们有什么区别？
- Java 中关于 equals 和 hashcode 的理解？
- 关于 Java 中深拷贝和浅拷贝的区别？
- 简单的说下 Java 的垃圾回收？
- 了解过 Java 的集合吗？说说 HashMap 的底层实现原理？ArrayList 和 LinkedList 的区别？Java 集合中哪些是线程安全的？
- 如何实现对象的排序？
- 知道 ThreadLocal 吗？说说对它的理解？
- 在你写代码的过程中有使用过设计模式吗？你知道哪些？为什么要这样用，能解决什么问题？
- 了解注解吗？了解反射吗？为什么要使用反射？
- 数据结构中常用排序算法？

以上就是关于 Java 所问道的知识点，记得不是太清楚了，待补充。。。

关于 Android 方面的问到的知识点

- Activity 的生命周期是什么？ onPause 和 onStop 有什么区别？
- Android 五种布局的性能对比？
- Android 四大组件是什么？分别说说对它们的理解？
- 关于 Service 的理解？它的启动方式有什么区别？
- 了解 fragment 吗？说说你对它的理解？
- 自定义过 view 吗？它的步骤是什么？说说你自定义 view 过程中出现的问题，以及是如何解决的？
- 刷新 view 的几种方式，他们有什么区别？
- Android 实现数据存储的几种方式？
- 如何实现 Android 中的缓存的，通过使用第三方库和自定义来分别说明一下缓存技术的实现？
- 如何实现 Activity 与 fragment 的通信？
- Android 5.0、6.0、7.0 新特性？
- Android 中的动画分类？
- 你以前是如何进行屏幕适配的？
- 说说 Activity 创建过程？
- Android 中如何与 JS 交互的？
- 了解 APP 的启动流程？
- 你知道哪些图片加载库？他们有什么区别？ImageLoader 的内部缓存机制是什么？是如何实现的？
- Android 中是如何实现异步通信的？
- 说说 Handler 内部实现原理？
- 使用过 AsyncTask 吗？说说它的内部实现原理？它有什么缺陷？如何改进？
- 知道 JNI、Binder 吗？说说你对它们的理解？
- 如何实现进程间的通信？
- 说说 Android view 和 viewGroup 的事件分发机制？
- 你开发过程中使用到了哪些第三方库？了解过他们的源码吗？
- 你了解广播吗？它与 EventBus 有什么区别？能互相实现吗？
- 你们网络请求是如何实现的？知道 Volley 吗？内部实现流程是什么？它与 OKHttp 有什么区别？
- 你了解哪些第三方功能？知道推送吗？它的原理是什么？
- 接触过 MVP 模式吗？说说看对它的认识？
- 知道 Android 中的多渠道打包吗？
- Android 签名机制的原理？反编译解压后的文件夹所包含的内容有哪些？
- 你了解过模块化、组件化开发吗？
- 开始开发 APP 如何进行架构？
- APP 工程模块是如何划分的？你是如何进行封装的？
- APP 是如何进行优化的？知道 OOM 吗？如何解决内存泄漏？

以上就是我这次面试过程中涉及到的一些关于 Android 方面的知识点，有点模糊了，全凭记忆，待补充....

经过上面的几个阶段，历时半个月，最终我找到了一家比较心仪的公司，整体的性价比个人感觉比较高，符合我的期望。以上便是我这次来杭州面试的点点滴滴，希望对有需求的小伙伴一些帮助~

请记住一点，薪水并不是唯一所要关注的重点，关键还得看看公司环境、领导、同事相处愉快不愉快？要不然给你再多的薪水，每天干的不爽，那不是很悲哀？

最后我会提供一些我面试准备阶段复习所用到的一些基础知识点链接，面试必问的一些基础原理一定得知道，不能含糊，要不然面试过程中必定会露马脚。有需要的小伙伴可以参考一下。

相关链接

- Java面试题集
- Android 名企面试题及涉及知识点整理

- 40个 Android 面试题

那些IT培训出来的Android工程师，希望你面试时涨点记性

这几天，公司在前程无忧上发布了招聘 Andriod 工程师广告。不到 3 个小时， hr 就抱怨说投递 Andriod 工程师的简历已经多达 300 份了。不得已将 Andriod 工程师招聘就下架，然后就去筛选简历了。

我也顺便看了看公司的要求，写的很简单，主要有：

> 经验：1 年以上。
>
> 有开发过蓝牙相关项目经验优先。
>
> 学历：大专及以上。

不知道 hr 和部门经理花费了多少时间挑选了 10 个人出来了。然后就预约了他们过来面试。

很荣幸，经理让我出一点面试题，还特意嘱咐，毕竟我们是软硬件的方案公司，已经有成熟的架构了。app 的难度不大，只要后面肯学是一样。就把第一轮面试的任务交给我。这里我并不是歧视培训机构出来的 Andriod 工程师，而是这几天面试下来，让我觉得很不可思议。如果有的人再用心一点，或许 offer 就是你的了。也希望不论你是正常毕业出来找工作，还是培训出来，自学的，或者中途转行的，都涨一点记性。

先说说笔试部分

有两道题基本上回答的很令人无语。

1、写一写自定义 view 的思路。

有几个人直接写了一个 onMeasure() 方法放那里了，就写了这几个单词，难道就不用多写一点解释。

我一看简历的工作经验，不是 2 年，就是 2 年半，还有 3 年的，怎么一个自定义 view 都没有遇到过？真的是让我怀疑你的工作经验是怎么来的。

重点是：有一个面试者就直接向我坦诚了自己是培训出来的。我瞬间就恍然了很多。我也是个打工的，没必要去指责他，只是给他讲了讲自定义 view，简历应该如何如何。这哥们居然临走时感谢我，要了我的微信。

2、有没有访问过公司官网？如果有，谈谈你的意见。（这一题是 hr 要求加上去的。）

结果很失望，只有 3 个人说访问过。

你去别人家公司面试，就不去访问别人一下官网，更何况，你投简历的时候，你就不用看公司简介，上上别人家公司官网。就算你是群投，收到面试通知后，都不用好好准备一下吗？去官网看看公司文化，团队，产品，特别是产品，大概就知道会用到哪些技术。

再说说口头问的部分

1、搞清楚自己的开发工具

1) 请问你现在开发使用的什么工具？

面试者：Android Studio

2) 那你现在主要使用哪个版本开发？

面试者：2.4

一瞬间，我直接懵了。Google 才放出正式的 2.3 版本。你就用起 2.4 呢？后面还补充告诉我，自己去官网下载的，一直在使用

2、不要刻意去讨好公司，技术的知识点不确定就不要随便回答。

1) 你在简历上说自己做过蓝牙相关的项目，那你告诉我，一般使用蓝牙需要哪几个权限？

面试者：好像两个，两三个吧？

2) 那你能不能说一下？

面试者：我都是直接复制粘贴的

唉，你让我说什么好。你要是做过蓝牙相关开发，哪怕是你忘记了权限，你也可以说一下蓝牙连接的流程。就算以前没做过，招聘的岗位都告诉你了，有做过蓝牙项目 app 的优先，你就可以去补充一下 Andriod 关于蓝牙的知识点啊?

3、别把别人上架的 app 当成自己的。

1) 你有没有 app 上架过？

面试者：有

等他在应用市场找给我的时候，我一看就傻眼了，下载量破 500 万了，一看开发者就不是你。唉，我当时就想，兄弟啊，你没有可以告诉我，就算找一个别人的，能不能不要这么多下载量的？更何况，国内 Andriod 应用市场这么多，想自己的 app 上架都不是什么难事。

更可况，面试要求你，也没有说非要你上架 app。毕竟我们公司的 app 的难度还好。

4、Andoid 6.0 权限的处理。

1）在实际开发中，你说如何处理 6.0 以上手机权限问题的。

面试者：我们开发的 app 不需要适配 6.0 啊。

2）要是客户的手机是 6.0 的，客户要求你的 app 项目适配 6.0 的呢?

面试者：不会吧，都不用适配 6.0 的

你不是都有两三年开发经验了？ 6.0 以上权限适配属于最基本的知识。随便说个思路，先申明权限，到用的时候，对高版本手机进行判断撒的……都是可以的，实际开发项目时，又不是不让你上网去学习研究。

唉……不知道该怎么说好了！

拒绝做面试题的！

有两个过来面试的，直接告诉我:

我写不好，能不能直接说啊！

我也只好同意了，毕竟从那么多简历里面，把你们筛选出来是多么地不容易。结果，好是令人失望。

如果你没有过硬的技术，请不要随便拒绝面试题。

后记

那个坦诚告诉我是培训出来的人，晚上发信息告诉我一些信息。

他们说 Andriod 面试，不用做题的？

现在企业喜欢经验多的，我们都是被要求写几年经验。这样才有面试机会。

你们招聘公司职位要求的技术不是随便写的吗？

大家都不容易，但是做技术这一行，还是需要硬实力。再怎么包装你有几年经验，拿到面试通知时，为什么不去好好准备面试，去背诵知识点。你连最起码的别人公司的官网都不上，你又有什么话语权？

哪怕你没有 app 上过架，确实是刚毕业，转行，或者刚培训出来的，如果临时抱佛脚，复习了公司要求的知识，这个 offer 就属于你的了。毕竟还有 3 个月试用期。

不论什么原因让你的简历如何完美，但是还请你的技术能够跟上。我所知道的，同事，朋友做 IT 的，有的是跨专业，有的是高中，有的是培训出来的，既然他们都行，请你也行！

作为一个双非渣硕，历经两个月的时间，面试了大大小小公司的 Android 实习生岗位，最近终于结束了面试状态，决定好好把面试问题以及相关经验整理下来，顺便附带自己的学习经验与准备过程，攒攒人品，为秋招再战。

前言

2016 年开始接触 Android，从刚开始接触就不断地听到 Android 市场饱和，工作难找等消息。虽然当时也非常迷茫，不过由于第一次深入接触编程语言，再加上自己的一点兴趣，就一直坚持下来了。

通过两个月的面试经历，确实发现 Android 岗位比较少，而且通常要求比较高，不仅需要 Android 开发经验，往往还需要会 React Native,JavaScript 等，甚至还期望你能具有 IOS 开发经验。

不过作为应届生还是有些优势的，那就是一些一线的互联网公司还是比较看中个人基础 以及发展潜力的，所以如果能在自己的专业方向上具有扎实的基础，1-2个实际开发项目以及个人的兴趣，还是能够找到一个满意的 Android 岗位的工作的。目前这些素质，自己也很欠缺，通过下面的面试经历就可以看出来，不过最起码有个努力的目标，可以好好准备为秋招做准备。

面试经验

自己大大小小投了也有 20 多家公司，不过经历简历筛选以及笔试淘汰，最终就经历了7家公司的面试。下面我就把自己面试中问到的问题贴出来供大家参考，一些具体项目相关的就不贴了。

阿里巴巴

阿里是 3 月初开始投的，是自己第一次面试大型的互联网公司，当时自己的准备也不够充分，表现不是很好，经历了三次技术面，最后挂了。

阿里一面

- 排序，快速排序的实现
- 树：B+树的介绍
- 图：有向无环图的解释
- TCP/UDP 的区别，滑动窗口，如何确保有效性
- volatile
- synchronized 与 Lock 的区别
- Java 线程池
- Java 中对象的生命周期
- 类加载机制
- 双亲委派模型
- Android 事件分发机制
- MVP 模式
- RxJava

阿里二面

- 抽象类和接口的区别
- synchronized 与 lock
- 集合 Set 实现 Hash 怎么防止碰撞
- JVM 内存区域 开线程影响哪块内存
- 垃圾收集机制 对象创建，新生代与老年代

- 二叉树 深度遍历与广度遍历
- B树、B+树
- 消息机制

阿里三面

- 项目介绍
- 项目中做了些什么？主要解决的问题
- 为什么选择 Retrofit，RxJava
- RxJava 的特点
- 进程调度
- 进程与线程
- 死锁
- 进程状态
- JVM 内存模型
- 并发集合了解哪些
- ConCurrentHashMap 实现
- CAS 介绍
- 锁 synchronized，lock
- 开启线程的三种方式，run() 和 start() 方法区别
- 线程池
- 常用数据结构简介
- 判断环
- 排序，堆排序实现
- 链表反转
- 海量数据 字典查找
- 平时看什么书

网易游戏

网易游戏当时投的时候就没抱希望，招聘信息上明确指定只招固定的那几所985高校，就随便投了，没想到笔试都没做就直接打电话面试了，不过问的问题确实很深入，结果显然，一面就挂了。

网易游戏一面

- 集合
- concurrenthashmap
- volatile
- synchronized 与 Lock
- Java 线程池
- wait/notify
- NIO
- 垃圾收集器
- Activity 生命周期
- AlertDialog,popupWindow,Activity 区别

远景能源

远景能源是一家互联网能源公司，给出的实习待遇是相当好，经室友推荐就投了简历。最后流程走完，得知挂在了二面上，大概原因就是没有拿得出手的项目，实际项目经验不足。

远景电话面

- 四大组件
- Android 中数据存储方式
- 微信主页面的实现方式
- 微信上消息小红点的原理
- 做的项目，一个印象深刻的问题
- 看的技术博客，印象深刻的

远景现场一面

- 两个不重复的数组集合中，求共同的元素。
- 上一问扩展，海量数据，内存中放不下，怎么求出。
- Java 中 String 的了解。
- ArrayList 与 LinkedList 区别
- 堆排序过程，时间复杂度，空间复杂度
- 快速排序的时间复杂度，空间复杂度

远景现场二面

问的都是一些项目问题，比较宽泛，没问具体技术点

今日头条

今日头条是在四月初投的，当时找了一个月，都没拿到拿得出手的 offer，有点心烦意乱，就又海投了一波。4.18 做了今日头条的面试，4.25 进行的视频面试。一共进行了 3 轮视频面试，头条的面试官很好，看的出来头条的技术是很强的，也很注重算法。三轮技术面后，hr 通知通过，给了个秋招终面直通卡，具体发不发 offer 两周内答复。这两天已经陆续开始发 offer 了，目前还没收到，可能备胎了吧。

今日头条一面

- 数据结构中堆的概念，堆排序
- 死锁的概念，怎么避免死锁
- ReentrantLock
- synchronized
- volatile
- HashMap
- singleTask 启动模式
- 用到的一些开源框架，介绍一个看过源码的，内部实现过程。
- 消息机制实现

今日头条二面

- synchronized 与 ReentrantLock
- ReentrantLock 的内部实现
- 用到的一些开源框架，介绍一个看过源码的，内部实现过程。

- Java 中异常
- App 启动崩溃异常捕捉
- 事件传递机制的介绍
- ListView 的优化
- 今日头条推荐新闻去重，推荐的时候去掉用户已经看过的新闻。
- 二叉树，给出根节点和目标节点，找出从根节点到目标节点的路径。手写算法
- 模式 MVP，MVC 介绍
- 断点续传的实现

今日头条三面

- 集合的接口和具体实现类，介绍
- TreeMap 具体实现
- synchronized 与 ReentrantLock
- 手写生产者/消费者模式
- 逻辑地址与物理地址，为什么使用逻辑地址
- volatile
- 一个无序，不重复数组，输出 N 个元素，使得 N 个元素的和相加为 M，给出时间复杂度、空间复杂度。手写算法
- Android 进程分类
- 前台切换到后台，然后再回到前台，Activity 生命周期回调方法。弹出 Dialog，生命值周期回调方法。
- Activity 的启动模式

触宝科技

触宝科技是一家上海的互联网公司，是通过实习僧上简历投递获得的这次面试机会，一共进行了两轮电话面试，最终获得了 offer。触宝科技的 hr 人很好，时间观念很强，整个流程走的比较顺利。

触宝科技一面

- RxJava 的作用，与平时使用的异步操作来比，优势
- Android 消息机制原理
- Binder 机制介绍
- 为什么不能在子线程更新 UI

触宝科技二面

- JVM 内存模型
- Android 中进程内存的分配，能不能自己分配定额内存
- 垃圾回收机制与调用 System.gc() 区别
- Android 事件分发机制
- 断点续传的实现
- RxJava 的作用，优缺点

爱奇艺

爱奇艺也是通过实习僧上简历投递获得的机会，本来不抱希望，结果过了 10 天左右约我面试。面了大概一个小时，聊得还不错，最后第二天通知我挂了，有点不知所措，可能是实习时间达不到要求吧（只能这样安慰自己了）。

爱奇艺一面

- RxJava 的功能与原理实现
- RecycleView 的使用，原理，RecycleView 优化
- ANR 的原因
- 四大组件
- Service 的开启方式
- Activity 与 Service 通信的方式
- Activity 之间的通信方式
- HashMap 的实现，与 HashSet 的区别
- JVM 内存模型，内存区域
- Java 中同步使用的关键字，死锁
- MVP 模式
- Java 设计模式，观察者模式
- Activity 与 Fragment 之间生命周期比较
- 广播的使用场景

携程

携程是 3 月份投的内推，结果挂掉了，后来通过笔试又获得的机会，笔试完快一个月才收到的通知，本来都快忘记了。既然通知了，就去面了。今天刚去面的，经过 2轮技术面，1 轮 hr 面，第二轮技术面是总监面，主要聊了聊项目上的问题。下周一会根据排名发 offer，现在暂时进入备胎池。

携程一面

- Activity 启动模式
- 广播的使用方式，场景
- App 中唤醒其他进程的实现方式
- AndroidManifest 的作用与理解
- List,Set,Map 的区别
- HashSet 与 HashMap 怎么判断集合元素重复
- Java 中内存区域与垃圾回收机制

携程二面

- EventBus 作用，实现方式，代替 EventBus 的方式
- Android 中开启摄像头的主要步骤
- Github 上传了哪些东西，代码量

学习资料

从 Android 开发工程师的角度来讲，我自己主要准备了以下几个方面的内容：

1.Java

Java 基础，如集合，反射，注解，IO，NIO，其中集合很重要，面试常问。

JVM，如内存区域，内存模型，垃圾回收机制的算法，收集器，类加载机制。

Java 并发，如 sychronized,lock,volatile，生产者/消费者模式，线程池，死锁。

2.Android

Android 基础，如四大组件，Binder 机制，消息机制，事件分发机制，自定义 View 过程。

Android 开源库，如 Retrofit,RxJava 等原理实现，优缺点，以及使用。

3.数据结构

链表，栈，队列，排序，树，图，以及其中涉及到的一些算法题目。

4.设计模式

单例模式，观察者模式，建造者模式，工厂模式，装饰者模式等。

5.操作系统

进程与线程，进程通信，进程调度，分页存储，分段存储，虚拟内存等。

下面介绍以下我看过的一些书籍。

Java

疯狂 Java 讲义(有些人说不好，自己看着还行吧，可以看核心卷，别人都推荐，我没看过)

Thinking in Java(看了一部分，没看完，建议有一定基础再看)

深入理解 Java 虚拟机(很好的一本书，必看)

Head First 设计模式(非常生动 的讲述设计模式)

Java 多线程变成核心技术(讲述 Java 中多线程的一些问题，比较基础)

Effective Java(78条开发中会用到的实际经验，很好，还没看完)

Android

Android 群英传(很基础，通俗易懂)

Android 开发艺术探索(面试必备，内容都深入浅出)

数据结构

大话数据结构(讲述各种数据结构的概念，算法实现是 C,可以作为入门书籍看)

剑指offer(面试必备，面试的时候好多 上面的题目)

其他的没看了，不过可以推荐一个网上视频课程，讲的很好——数据结构

操作系统

现代操作系统(需要耐心的看，书也挺厚，暂时没看完)

推荐个网上课程——操作系统

总结

找工作是个很辛苦的事情，而且一般周期都比较长，有时候即看个人技术，也看运气。第一次找工作，最后的结果虽然不尽如人意，不过收获远比 offer 大。接下来就是针对自己的不足，好好努力了。

关于面经

杭州找 Android 工作的点点滴滴

那些IT培训出来的Android工程师，希望你面试时涨点记性

为跳槽的你献计献策（Android）

想编程，是勤奋自学还是去培训班学习?

更多面试吐槽，面试题，请看这个专题

《Android面试专辑》