https://mp.weixin.qq.com/s?__biz=MzI0MjE3OTYwMg==&mid=2649550710&idx=1&sn=a39775baa4e7a71bd8493fb9cdfb6702&chksm=f118040bc66f8d1de418bdd36d482828f4c0afe6e22e4064a8f32a09c3c72a162af229b34def&scene=21#wechat_redirect

转眼就到了2017年11月，离2018剩下不到60天了，很多人估计正在跳槽的路上，整理一份Android高级开发工程师面试集锦，对照这些问题进行复习，将会事半功倍。废话不多说，直接上干货，由于是整理于网络，就不标明原创，希望大家多多分享和转发给有需要的同学。

阿里巴巴

- LRUCache原理

  https://www.jianshu.com/p/b49a111147ee

  LRUCache用于Android中做内存缓存，实现是使用了LinkedHashMap维护了强引用对象，并通过构造方法为LinkedHashMap指定了访问顺序结构，LinkedHashMap是使用了数组+双向列表的方式来实现的，我们可以给LinkedHashMap缓存的对象指定大小，一般为可用内存的1/8，当缓存的对象超过该指定大小时，会运用LUR即最近最少使用算法删除LinkedHashMap缓存的队尾元素。

  其实，在对缓存的get操作中，会把缓存对象取出，并移动到队头；当调用put方法插入对象时，会把对象插入到队头，并重新缓存计算大小，如果超出缓存指定大小，则移除相应缓存对象。

  get、put、remove方法中对LinkedHashMap的操作，都使用了synchronized保证线程安全。

- 图片加载原理

  http://blog.csdn.net/qq_34168374/article/details/54407976

  图像显示牵涉到CPU、GPU、Display,CPU负责计算图片的形状和文字的纹理；GPU负责渲染图片的颜色；Display就是屏幕显示的图像 ，Vsync是显卡输出频率和屏幕刷新频率同步的意思。16ms。

- 模块化实现（好处，原因）

https://www.cnblogs.com/baronzhang/p/6861258.html

好处：

1.方便多团队并行开发测试。

2.模块间解耦，方便升级。

3.可单独编译打包某一模块，增加复用性，提升开发效率。

- JVM内存模型、内存区域

https://www.jianshu.com/p/51e4051de38c

JVM内存区域可分为：

1.堆(公共)：存放对象

2.方法区(公共)：存放加载类的信息、常量、静态常量等，即编译以后的代码等数据。

3.虚拟机栈：局部变量表、操作数栈、方法返回值等。

4.本地方法栈：和虚拟机栈类似，只是为Native方法服务。

5.程序计数器：通过它来选去吓一跳需要执行的字节码指令

JVM内存模型：

规定了所有的变量都存储在主内存中，每条线程中还有自己的工作内存，线程的工作内存中保存了被该线程所使用到的变量（这些变量是从主内存中拷贝而来）。线程对变量的所有操作（读取、赋值）都必须在工作内存中进行。不同线程之间也是无法直接访问对象内存中的变量，线程间变量值的传递均需要通过主内存来完成。

- 统计启动时长,标准

https://www.jianshu.com/p/c967653a9468

启动分为冷启动和热启动，

本地的话可以通过 adb shell am start -w packagename/activity查看启动时长，totaltime就是App的启动时间

线上的话，如果是热启动：起点可以在Activity的onStart方法，截止点为Activity的onWindowFocusChanged中（onResume时布局的measure、layout\onDraw还没有执行）

冷启动的话：起点为Application的attachBaseContext方法(Application的构造方法没有初始化Context)，截止点为Activitiy的onwindowFocusChanged中

- 如何保持应用的稳定性

1.写代码时严格控制代码质量，code review

2.使用Android Studio的代码分析工具，优化代码

3.尽量使用系统提供的方法，系统没有提供的方法才自己实现

4.使用LeakCanary检测内存泄漏，并优化

5.对App进行压力测试

- ThreadLocal 原理

http://cmsblogs.com/?p=2442

ThreadLocal是为了实现在不同线程种有自己的状态而引入的一个机制，实现原理是每个Thread中都会有一个ThreadLocalMap实例，ThreadLocalMap里用一个Enery[] 来保存数据的，其中index是通过ThreadLocal算出来的，散列冲突是通过开放定址法解决的，Enery存储的key是ThreadLocal，value存储的是真实值。ThreadLocal的set、get、remove操作都是针对其操作。所以set操作就是先得到当前线程的ThreadLocalMap，然后把ThreadLocal为key,设置的值为value存储下来，而get操作就是先得到当前线程的ThreadLocalMap，然后根据ThreadLocal为key去取得value值。

- 谈谈classloader

https://www.2cto.com/kf/201608/533778.html

ClassLoader就是专门用来处理类加载工作的，也叫类加载器。

在Android中会有一个BootClassLoader实例，用于加载一些系统Framework层需要的类；另外当App启动的时候还会需要一个PathClassLoad。

创建ClassLoader实例需要使用一个现有的ClassLoader实例作为新创建实例的Parent，这样以来，一个Android应用，就可以被一颗树关联起来，这就是ClassLoader的双亲模式。

当ClassLoader需要加载一个类时，会先看是否已经加载过该类，有就返回；没有就查询Parent是否已经加载过该类，如果加载过，就直接返回Parent加载的类；如果继承路线上的ClassLoader都没有加载，才由Child执行类的加载工作。

同一Class = 相同的Class Name+Package Name+ClassLoader。

Android中动态加载类遇到的问题：

1.Android中的许多组建类，如Activity、Service需要在清淡文件中注册，如果动态加载进来的组建，没有在清单文件中注册时无法使用的。

http://blog.csdn.net/jiangwei0910410003/article/details/48104455

可以使用两种方式来解决：

反射机制来实现：不需要关心Activity生命周期，但需要在清淡文件中声明

代理机制来实现：不需要声明多个Activity，只要声明一个代理Activity就可以了，但声明周期需要自己管理

2.Android中的Res资源会动态生成R.id，而动态加载进来的类，用到的R.id和现有的Resource实例中保存的资源Id对应不上

http://blog.csdn.net/jiangwei0910410003/article/details/47679843

通过反射，调用AssetManager中的addAssetPath方法

- 动态布局

Android中的布局一般分为两种，静态布局和动态布局。

静态布局XMl写死的布局；动态布局就是通过Java代码控制的布局。

推荐使用静态布局，方便查看。

- 热修复,插件化

https://www.jianshu.com/p/e61a4d10e122

热修复技术：修复线上bug，并不嫩动态添加功能。例如：Tinker

插件化技术：可以动态地添加功能到客户端，例如DroidPlugin

热修复的方式：1.方法级别的替换，如AndFix；2.类级别的替换，如Hotfix

热补丁和插件化技术有很多技术难点：如何将程序代码加入到已经安装在用户手机中的App运行，如何加载图片、布局文件等资源，如何加载so文件，还要AndroidManifest.xml在安装到客户手机中之后就无法往里面加入东西的问题。

插件化详解：http://blog.csdn.net/jiangwei0910410003/article/details/48104581

- 性能优化,怎么保证应用启动不卡顿

http://blog.csdn.net/hpc19950723/article/details/71170345

App的启动包括冷启动和热启动，我们主要是要优化冷启动。

App冷启动的优化，主要包括两个方面Application的初始化+MainActivity的界面加载绘制优化。

Application的初始化优化：尽量减少在Application的构造方法、attachBaseContext、onContext中的初始化工作，就算需要有，尽量放到子线

程或者延迟到MainActivity。

MainActivity的界面加载绘制优化：优化MainActivity的布局，使用ViewStub减少没有必要立即显示的视图。另外可以借助SplashActivity优化。比如，在SplashActivity显示广告的同时，准备MainActivity的数据，然后跳转MainActivity时把准备好的数据通过Intent传递过去。另外，还可以把启动页做成Fragment，在MainActivity添加它，在启动页倒计时的时间，进行网络请求，请求成功以后再加载MainActivity的正式布局，MainActivity的真实布局通过ViewStub来延迟加载。

- 怎么去除重复代码

具体情况需要具体分析

Android来说：定义基Activity和Fragment，把大量公共的方法可以抽取出来，放在基类，比如Context初始化、统计等。另外，功能相似的页面，可以多考虑Fragment实现，而不是Activity实现，因为这样方便复用。

Java来说：提炼方法、抽象基类、提取常量等

布局来说：可重用的代码可以单独抽离出来，作为布局文件，然后使用include方式引入。

- SP是进程同步的吗?有什么方法做到同步

https://www.jianshu.com/p/875d13458538

SharedPreference是可以做到多进程同步的，不过需要使用MODE_MULIT_PROCESS，即跨进程模式，在这种模式下，每次调用getSharedPreferences都会重新加载文件，而不去使用内存缓存。

但是多进程读取同一文件时不安全的，所以不建议这么使用，替换方案建议使用ContentProvider来实现，如果之前已经使用SP，则可以自定义一个Sp代理类，当处于跨进程存取数据时，使用SP的代理类去操作，然后它的具体实现是通过ContentProvider，类似于AIDL，这样不用修改原有逻辑。

- 介绍下Surface View

http://blog.csdn.net/listening_music/article/details/6860786

1.SurfaceView允许其他线程修改UI，而View只能在主线程中更新UI

2.SurfaceView是放在其他最底层的视图层次中，所以其他视图都在它上面，可以在它上面添加一些层，而且它不能是透明的。

3.它执行动画的效率要比View高，并且它可以控制帧数

4.SurfaceView的使用比View复杂，占用资源也比View多，除非View完成不能完成，否则最好不要使用SurfaceView

- HashMap、LinkedHashMap、ConcurrentHashMap、SparseArray、ArrayMap 的实现原理

http://blog.csdn.net/justloveyou_/article/details/62893086

HashMap是使用使用散列表实现的，可以看做一个链表数组，是使用拉链法来解决散列冲突的。

http://blog.csdn.net/justloveyou_/article/details/71713781

LinkedHashMap继承于HashMap，相当于HashMap和双向链表的结合体。

https://www.cnblogs.com/chengxiao/p/6842045.html

ConcurrentHashMap主要使用了“分段锁”的思想，在同一段内的线程才会竞争锁。

http://blog.csdn.net/wzy_1988/article/details/51559012

SparseArray是使用两个数组来实现的，一个Keys，一个Values，其中Keys是一个存储key的素组，它是有序排列的，所以可以使用二分查找很快地确认key在Keys所处位置index，然后并把value插入到Values的index对应位置。

http://lvable.com/?p=217

ArrayMap也是使用两个数组来实现的，一个存放Item Hash值，一个存放Entry，Entry包括key 、value,当存放一个数据时，是先算得Item的Hash值，确定在item hash数组的index，然后把key、value存放到对应的Entry中；取值类似，得到item hash中的index是使用的二分查找。

- BroadcastReceiver，LocalBroadcastManager区别

http://www.trinea.cn/android/localbroadcastmanager-impl/

1.BroadcastReceiver可以应用于应用间和应用内通信，而LocalBroadcastManager只能应用于应用内通信。这样也体现了LocalBroadcastManager的安全性更高。

2.BroadcastReceiver是使用Binder实现的，而LocalBroadcastManager是基于Handler来实现的。所以LocalBroadcastManager的效率更高。

- Bundle 机制

http://blog.csdn.net/a553181867/article/details/51166600

1.可以看成一个特殊的Map，支持key-value存储方式，内部是使用ArrayMap来实现的

2.实现了Parcelable，支持进程间通信

3.支持传递基本类型的数据、Parceable、Seralizable类型

- Handler 机制

http://blog.csdn.net/qian520aoL/article/details/78262289?locationNum=2&fps=1

1.主要牵涉到的类有Message、Handler、Looper、MessageQueue

2.功能

1.Message:携带信息，Handler会作为它的target，Runnable会作为它的callback，它会被Handler发送出去，然后存放在MessageQueue中，Looper会遍历MessageQueue取出Message，然后处理分发。

2.Handler:主要是发送Message和处理Message。

3.Looper:用于从MeesageQueue循环取出Message去处理

4.MessageQueue:用于存放Meesage

3.一个线程中：Message可以有多个，Handler可以有多个，Looper只有一个，MesageQueue只有一个

4.流程：

1.新线程中本身没有Looper和MessageQueue，所以需要先调用Looper.prepare()方法，该方法会创建Looper对象，并用ThreadLocal存放起来，Looper的构造方法中会创建MessageQueue，这样Looper就和Thread绑定起来了，MessageQueue和Looper也绑定起来了。

2.然后创建Handler对象，并实现其handleMessage方法，用于处理Message消息

3.然后调用Looper.loop，从MessageQueue不断取出Message来处理

4.Handler.sendMessage或者View.post的实际，都是把Message加入到MessageQueue中

5.当Looper.loop从MessageQueue不断取出Message来处理时，会先判断message.callback，即是否是View.post方式发送消息的，如果是，则会调用call方法；

如果不是，判断Handler.Callback call是否为空，如果不是空，则调用其handleMessage方法；如也会空，则调用Message.target即Handler的handleMessage方法

5.处理消息所在的线程，是Looper运行的线程，而Looper运行的线程需要看Handler的创建过程，当以默认方式创建Looper，则消息的处理就在当前线程；如果创建Handler时，传入了特定的Looper，则消息的处理在特定Looper运行的线程中。比如在子线程中创建Handler，但是传入了主线程的Looper，则消息会被加入到主线程中的MessageQueue中，所以消息的处理，是在主线程中。

- Android 事件传递机制

  https://www.jianshu.com/p/e99b5e8bd67b

1.事件分发可以分为三层：Activity、ViewGroup、View

2.主要牵涉都到三个方法：dispatchTouchEvent、onInterceptTouchEvent、onTouchEvent

3.牵涉到的三个方法：

1.dispatchTouchEvent：处理事件的分发，如果是true，表示消费了，不传递；如果为false，则调用父控件的onTouchEvent方法，Activity中的话，则直接消费。如果是super ,则调用子控件的dispatchTouchEvent完成事件分发。

2.onInterceptTouchEvent:处理事件的拦截，ViewGroup特有，默认为false，如果返回true，事件不会继续往子控件分发，而是交给自身的onTouchEvent处理

3.onTouchEvent：处理事件，如果是true，则表示事件消费了，则不传递；否则调用父控件的onTouchEvent方法

4.各层方法的实现

1.Activity的dispatchTouchEvent：调用了Window的方法，其实就是调用了ViewGroup的diapatchTouchEvent方法，如果该方法返回了true，则直接返回，表示消费了；如果该方法返回了false，则继续调用Activity的onTouchEvent

2.Activity的onTouchEvent方法：没什么意思

3.ViewGroup的dispatchTouchEvent：主要是判断是否需要拦截，如果不需要拦截，则直接调用子控件的dispathTouchEvent方法，分发事件到子控件，等子控件处理完后，再调用自身的onTouchEvent，所以子控件的dispatchTouchEvent方法和子控件的onTouchEvent方法都会在其前面调用，这样就形成了U型。如果需要拦截，则不会调用子控件的dispatchTouchEvent。如果没有子控件去消费该控件，则mFirstTouchTarget是空，则会调用super.dispatchTouchEvent，即View的dispatchTouchEvent方法，调用ViewGroup的onTouch方法（触发触摸事件）和onTouchEvnet方法(会触发点击事件)

4.ViewGroup的onTouchEvent：ViewGroup自身没有该方法的实现，不过View有onTouchEvent的实现，其实就是根据各种逻辑，处理click、double click等事件

5.ViewGruop的onInterceptTouchEvent:是否需要拦截子控件的事件，true表示拦截，默认为false。

6.View的dispatchTouchEvent:会先调用onTouch方法，如果该方法没有返回true，则调用onTouchEvent方法。这也佐证了onTouch方法在onClick方法之前调用。

7.View的onTouchEvent：和4一样

- App启动流程，从点击桌面开始

https://www.jianshu.com/p/a72c5ccbd150

1.桌面本身是一个Launcher应用，也是一个Launcher进程

2.点击App图标时，会通过Binder方式跟system_server进程通信，告诉AMS，需要启动的包名等信息

3.system_server进程会通过socket方式通知zgote进程fork一个App进程出来，并调用ActivityThread.main方法，然后调用Looper.loop开启App进程主线程消息循环

4.App进程通过Binder方式把attach Application告知system_server的AMS

5.system_server进程等AMS通过Binder方式把scheduleLaunchActivity告知App进程等Application Thread(Binder线程池)

6.Application Thread通过Handler发送H.LAUNCH_ACTIVITY消息通知主线程中的ActivityThread去启动Activity

7.ActivityThread调用handleLaunchActivity去调用Activity的onCreate方法

- 对 Dalvik、ART 虚拟机有基本的了解；

https://www.jianshu.com/p/58f817d176b7

1.Android4.4时ART替换了Dalvik

2.Dalvik运行的是dex，jvm运行java字节码，ART运行的是机器码

3.ART和Dalvik的比较

1.ART性能更好、电池续航能力更强，支持更低的硬件

2.ART所占存储空间更大

3.ART安装时间较长

- Java线程池

http://www.importnew.com/19011.html

1.线程池中牵涉到的几个类的关系，ThreadPoolExcutor继承AbstractExecutorService抽象类, AbstractExecutorService实现了ExecutorService接口，ExecutorService接口继承了Executor接口。

2.线程池的四种状态：初始化以后是RUNNING状态、调用了shutdown方法，则处于SHUTDOWN状态、调用了shutdownNow方法，则处于STOP状态、当处于SHUTDOWN或STOP状态，所有工作线程已经被销毁，任务缓存队列被清空或者执行完以后，则处于TERMINATED状态

3.线程池处理任务的策略

1.线程池被初始化以后，默认是没有线程的。

2.当线程池中的线程数量小于corePoolSize时，则每来一个任务，则创建一个线程去执行这个任务

3.当线程池中的线程数量大于等于corePoolSize，则每来一个任务，会尝试将其添加到任务缓存队列当中，若添加成功，则该任务会等待空闲线程将其取出执行；若挺假失败（一般来说是任务队列已满），则会尝试创建新的线程去执行这个任务

4.当线程池中的线程数达到manimumPoolSize时，则会采取任务拒绝策略进行处理

5.如果线程池中的线程数大于corePoolSize时，如果某个线程空闲时间超过keepAliveTime，线程将被终止，直至线程池中的线程数目不大于corePoolSize；如果允许为核心池中的线程设置存活时间，那么核心池中的线程空闲时间超过keepAliveTime，线程就会被终止。

4.Exectors类提供的几种静态方法创建线程池：

1.Exectors.newFixedThreadPool:corePoolSize和maximumPoolSize一样

2.Exectors.newSingleTheadExecutor:corePoolSize和maximumPoolSize都为1

3.Exectors.newCacedThreadPool:corePoolSize为0，maximumPoolSize为Integer.MAX_VALUE，keepAlivetime为60s,也就是说，当线程空闲60秒时，就会线程线程。

4.Exectors.newScheduledThreadPool:corePoolSize可以指定，maximumPoolSize为Integer.MAX_VALUE

5.根据任务类型来配置线程池

1.如果是CPU密集型任务，就需要尽量压榨CPU，避免过多的线程切换，参考值可以设置为：CPU数+1或者CPU数*2

2.如果是IO密集型任务，尽量多一些线程，当IO线程处于等待的时候，其他线程可以做别的事情，参考值可以设置为：2*CPU数 或 cpu数 / (1-阻塞系数)，阻塞系数可为0.8或0.9

- 画出 Android 的大体架构图

http://blog.csdn.net/wang5318330/article/details/51917092

Android的系统架构主要分为四层：

1.应用层(Applications):系统提供的程序包，比如电子邮件，以及自身编写的App

2.应用框架层(Applications Framework):是Android应用开发的基础，比如内容提供者、包提供者、通知管理器等

3.系统库(Libraries)和Android运行时(Android Runtime)：系统库包括Sqlite、Webkit等，Android运行时包括核心库和虚拟机。

4.Linux内核(Linux Kernel):提供安全性、内存管理、进程管理等。

- Android进程和生命周期；

http://blog.csdn.net/furongkang/article/details/6988589

1.Android中根据重要性可分为以下五种进程

1.前台进程：有Activity处于当前与用户交互等进程

2.可视进程：没有与用户可交互等Activity，但是它是可见的(比如调用了onPause而没有调用onStop的情况)

3.服务进程：调用了StartServer的进程

4.后台进程：包含不为用户可见的Activity (即stop被调用了)

5.空进程：主要是作为缓存以改善再次于其中启动的时间

2.优先级为上面从高到低，系统会尽可能长的延续一个应用程序进程，但是内存过低时，然后会移除一些进程，顺序是上面的从下到上。

- TCP/UDP的区别

http://blog.csdn.net/li_ning_/article/details/52117463

1.TCP面向连接；UDP是无连接的，即发送数据之前不需要建立连接

2.TCP提供可靠服务。也就是说，通过TCP连接传送的数据，无差错、不丢失、不重复，且按序到达；UDP尽最大努力交付，即不保证可靠交付

3.TCP面向字节流，实际上是TCP把数据堪称一串无结构的字节流；UDP是面向报文的，UDP没有拥塞控制，因此网络出现拥塞不会使发送速度降低（对实时应用很有用，如IP电话、实时视频等）

4.每条TCP连接只能是点到点到；UDP支持一对一，一对多，多对一，多对多点交互通信

5.TCP首部开销20个字节；UDP的首部开销，只有8个字节

6.TCP的逻辑通信信道是全双工点可靠信道；UDP则是不可信信道

- Https请求慢的解决办法

1.不用域名请求，省掉DNS解析，IP直连接

2.如果有多台服务器，根据逻辑选择合适的IP请求

3.使用优秀的网络请求框架

4.优化Https相关的内容，比如减少完全握手、替换RSA算法。

- GC回收策略

https://www.cnblogs.com/xiaoxi/p/6486852.html

http://blog.csdn.net/rabbit_in_android/article/details/50386954

1.判断对象是否可回收的算法

1.引用计数法(解决不了相互引用的问题)

2.可达性分析法(GCRoots)

2.对象的四种状态

1.强引用

2.软引用

3.弱引用

4.虚引用

3.垃圾收集算法

1.标记-清除（Mark-Sweep）算法

2.复制（Copying）算法

3.标记-整理（Mark-compact）算法

4.分代手机算法

4.Dalvik虚拟机是使用的Mark-Sweep即标记-清除算法

- 描述清点击 Android Studio 的 build 按钮后发生了什么

http://blog.csdn.net/yangxi_pekin/article/details/78614074

主要有以下5步：

1.gradle检测module以来的所有libray是否准好好，如果一个module依赖另外一个module，则另外一个module也要被编译

2.资源和Manifest文件被打包

3.处理编译器的注解，源码被编译成字节码

4.处理所有带transform前缀的task

5.library生成.arr文件，application生成.apk文件

编译过程：

aapt将AndroidManifest.xml和res下的资源编译生成R.java；aidl工具生成对应的Java Interfaces；将src和通过aapt生成的R.java、.aidl文件通过javaC命令去生成.class文件；使用dex tool将class文件转化成Dalvik byte code。这时候，将所有class文件、第三方jar、没有编译过的图片和编译过的图片都包括。dex文件传给apkbuilder去打包成.apk;最后采用zipalign tool打入签名

- 大体说清一个应用程序安装到手机上时发生了什么；

1.Android系统会为其分配一个UID

2.读取AndroidManifest文件，授权App权限

3.xxx

- App 是如何沙箱化，为什么要这么做；

http://blog.csdn.net/ljheee/article/details/53191397

Android沙箱主要基于以下三个原理来实现的：

1.标准的Linux进程隔离

2.大多数进程拥有唯一的用户ID（UID）

3.严格限制文件系统权限

如何访问其他App的资源?

1.声明适当的manifest

2.与其他受信任的应用程序在同一进程中

为什么要这样做?

1.把App之间隔离起来，保证了App的安全性

- Android 上的 Inter-Process-Communication (IPC)跨进程通信时如何

工作的；

1.跨进程的场景：

1.应用内，通过在清单文件中process指定组建运行进程，应用内两个进程通信

2.应用间，两个应用之间通信，也就是两个进程之间通信

2.多进程可能造成的问题

1.静态成员和单例模式完全失效

2.线程同步机制完全失效

3.SharedPreference的可靠性下降

4.Application会多次创建

3.安卓中的IPC方式有：

名称	优点	缺点
Bundle	简单易用	只能传递B
文件共享	简单易用	不适合高并
Messenger	功能一般，支持一对多串行通信，支持实时通信	不能很好处 Bundle支持
AIDL	功能强大，支持一对多并行通信，支持实时通信	使用稍复杂
ContentProvider	在数据源方面访问强大，支持一对多并发数据共享，Call方法扩展其它操作	可以理解为
Socket	功能强大，支持一对多并发实时通信	实现细节稍
广播		

- MVP模式

http://blog.csdn.net/lmj623565791/article/details/46596109

1.View对应Activity，负责View的绘制以及与用户交互

2.Model依然是业务逻辑和实体模型

3.Presenter负责完成View与Model间的交互

- synchronized与Lock的区别

http://blog.csdn.net/u012403290/article/details/64910926?locationNum=11&fps=1

1.线程的5大状态

1.新建状态：线程创建了，但是还没有调用start方法

2.就绪状态：调用了start方法，但是并不是说只要调用start方法线程马上就会变成当前线程。值得一体的是，线程在睡眠和挂起中恢复的时候也会进入就绪状态。

3.运行状态：线程被设置为当前线程，开始执行run方法，这就是线程进入运行状态

4.阻塞转台：线程被暂停，比如说调用sleep方法后线程就进入阻塞状态

5.死亡状态：线程执行结束

2.synchronized和Lock的区别

类别	Synchronized	Lock
存在层次	Java关键字，Jvm层面上的	是一
锁的释放	1.以获取锁的线程执行同步代码，释放锁；2.线程执行发生异常	在fin
锁的获取	假设A线程获得锁，B线程等待；如果A线程阻塞，B线程h会一直	分情
锁的状态	无法判断	可以
锁类型	可重入、不可中断、非公平	可重
性能	少量同步	大量

- Recycleview Listview 的区别,实现原理、性能

https://cloud.tencent.com/developer/article/1005658

1.ListView原理

1.RecycleBin机制

2.会进行两次onLayout，第一次onLayout时会调用getView方法去加载item的布局，第二次onLayout会先detach所有的View然后再从缓存中add所有的View

2.RecycleView原理

1.RecycleView可局部刷新

2.自动实现了tag，解决错位问题

http://blog.csdn.net/qq_23012315/article/details/50807224

- 屏幕适配方案

https://www.jianshu.com/p/ec5a1a30694b

1.针对不同分辨率的手机，放置多套资源。或者在大分辨率资源文件夹下放一套资源（这样低分辨率手机会浪费资源）

2.尽量使用dp，少使用px

3.使用Java代码结合LayoutParams动态计算

4.使用线性布局的weight

5.推荐使用约束布局等相对灵活的布局，尽量不要使用绝对布局等

6.像TextView控件不知道具体大小时，使用wrap_content

7..9图片

8.根据屏幕配置来加载相应的UI布局

- Fragment生命周期

http://blog.csdn.net/lmj623565791/article/details/37970961

1.为什么要使用Fragment?

1.适配平板

2.便于复用

3.封装组件、解耦

2.Fragment的生命周期

1.onAttach:Fragment与Activity绑定时调用

2.onCreate

3.onCreateView：创建该Fragment的视图

4.onActivityCreated：当Activity的onCreated方法返回时调用

5.onStart

6.onResume

7.onPause

8.onStop

9.onDestroyView：与onCreateView相对应，当该Fragment的视图被移除时调用

10.onDestroy

11.onDetach：与onAttach相对应，当Fragment与Activity关联被取消时调用

3.Fragment与Activity的交互

1.Activity与Fragment通信

1.使用Bundle传值

2.在Activity中管理Fragment的引用，调用Fragment的public方法

3.使用EventBus、Handler来通信

2.Fragment与Activity通信

1.通过getActivity得到Activity引用，然后强转类型，调用public方法，不过需要判断Activity类型

2.在Activity创建Fragment以后，设置Handler到Framgent中，就可以在Fragment使用Handler通信

3.上述2中的Handler可改为接口，则变为接口回掉方式

4.通过EventBus等方式

- 权限管理系统（底层的权限是如何进行 grant 的）

http://blog.csdn.net/wxlinwzl/article/details/38395589

1.从AndroidManifest.xml中提取permission信息

2.获取Pageage中的证书，验证，并将签名信息保存在Package结构中

3.如果是普通package,那么清除package的mAdoptPermission字段信息

4.如果AndroidManifest.xml中指定了shared user，那么先查看全局list中（mSharedUsers）是否该uid对应的SharedUserSetting数据结构，若没有泽分配一个uid，创建ShareUserSetting并保存到全局lst(mSharedUsers)中

5.创建PackageSetting数据结构。并将PackageSettings和SharedUserSetting进行绑定。其中PackageSetting保存了SharedUserSetting结构；而SharedUserSetting中会使用PackageSetting中的签名信息填充自己内部的签名信息，并将PackageSettings添加到一个队列中，表示PackageSetting为其中的共享者之一。

6.如果mAdoptPermission不为空，那么处理permission的领养

7.添加自定义权限。将package中定义的permissionGroup添加到全局的列表mPermissionGroups中去，将package中定义的permission添加全局的列表中去。

8.清除不一致的permission信息

9.对每一个package进行轮询,并进行permission授权

- Java中对象的生命周期

http://blog.csdn.net/sodino/article/details/38387049

1.对象的生命周期主要有如下几个阶段

1.创建阶段(Created)：为对象分配内存空间、开始构造对象、从超类到子类对static成员进行初始化、超类到子类对static成员进行初始化、超类成员变量按顺序初始化，递归调用超类对构造方法、子类成员变量按顺序初始化，子类构造方法调用

2.应用阶段(In Use)：对象至少被一个强引用持有

3.不可见阶段(Invisible)：程序的执行已经超出了该对象的作用域（一般编译过程就会报错）

4.不可达阶段(Unreachable)：该对象不再被任务强引用所持有

5.收集阶段(Collected)：当GC发现处理不可达阶段并且GC已经对该对象的内存空间重新分配做好准备时，则对像进入了收集极端。如果该对象已经重写了finalize方法，则会去执行该方法的终端操作。

6.终结阶段(Finalized)：当对象执行完finalize方法后仍然处于不可达状态时，则该对象进入终结阶段。在该阶段时等待垃圾回收器对该对象空间进行回收。

7.对象控件重分配阶段(De-allocated)：GC对该对象所占用的内存空间进行回收或者再分配，则该对象彻底消失。

2.不要重载 finazlie方法

1.会影响JVM的对象分配和回收速度

2.可能造成该对象的再次“复活”

- volatile

http://www.importnew.com/24082.html

1.java内存模型：工作内存、主内存

2.并发的三大概念：原子性、可见性、有序性

3.volatile修饰得变量具有下面两层含义

1.保证了变量的可见性：强制将修改的值，立即写入主存；当别的线程修改值，当前线程的工作线程的缓存变量失效，需要从主内存中读取。

2.禁止进行指令重排序：指令重排序时，前面代码肯定全部在之前；后面的代码肯定在之后。

4.volatile并不能保证原子性

5.volatile实现原理：

1.可见性：当进行写操作时，发送一条Lock前缀的指令，将缓存数据写到主内存中，而嗅探会在总线上传播的数据检查自己的缓存是否过期，发现缓存内存地址被修改，则设置为无效转改，当处理器对这个数据进去读取时，会强制从系统内存把数据读取到处理器缓存中。

2.有序性：Lock前缀指令实际上相当于一个内存屏障，它确保指令重排序时不会把其后面的指令排到内存屏障之前到位置，也不会把前面的指令排到内存屏障后面。

6.volatile和synchronized的区别

synchronized是防止多个线程同时执行一段代码，所以能保证原子性，但是效率上会有影响；volatile性能上优于synchronized，但是不能保证原子性，

- Service启动方式，有什么不同？一起混合使用

1.有两种启动方式，startService和bindService

2.两种启动方式的区别

1.生命周期：startService方式，Service的生命周期跟启动它的Activity没有绑定关系；bindService方式，Service的生命周期跟启动它的Activity绑定，Activity退出时，Service也会停止

2.通信：startService方式，Activity和Service不能直接通信；bindServcie方式，Activity和Service可以直接通信。

3.停止服务方式：startService调用stopService，多次调用只有第一次有用；bindService方式调用unBindService，多次调用会报错

- 排序

https://www.cnblogs.com/xiao-chuan/archive/2016/10/31/6014945.html

1.排序种类

1.插入排序：直接插入排序、二分法插入排序、希尔排序

2.选择排序：简单选择排序、堆排序

3.交换排序：冒泡排序、快速排序

4.归并排序

5.基数排序

2.时间复杂度

1.直接插入排序：最好时间复杂度O(n)，最坏O(n2)，平均时间复杂度O(n2)

2.二分法插入排序：最好O(n)，最坏O(n2)，平均O(n2)

3.希尔排序：平均O(nlngn)

4.简单选择排序：O(n2)

5.堆排序：O(nlngn)

6.冒泡排序：O(n2)

7.快速排序：O(nlngn)

8.归并排序：O(nlngn)

9.基数排序：O(d(n+r)),d为位数，r为基数

3.稳定性

1.稳定：冒泡排序、插入排序、归并排序和基数排序

2.不稳定：选择排序、快速排序、希尔排序、堆排序

4.排序算法的选择

1.数据规模小

1.待排序列基本有序的情况下，可以选择直接插入排序

2.对稳定性不做要求推荐简单选择排序，对稳定性有要求用插入或冒泡排序

2.数据规模不是很大

1.完全可以用内存空间，序列杂乱无序，对稳定性没有要求，快速排序，此时要付出log(N)的额外空间

2.序列本身可能有序，对稳定性有要求，空间允许下，用归并排序

3.数据规模很大

1.对稳定性有要求，考虑归并排序

2.对稳定性没要求，用堆排序

- 进程与线程

http://blog.csdn.net/luoweifu/article/details/46595285

1.线程：是程序执行中的一个单一的顺序控制流程，是程序执行流的最小单元，是处理器调度和分派的基本单位。它由线程ID、当前指令指针(PC)、寄存器和堆栈组成。

2.进程：是一个具有独立功能的程序在一个数据集上的一次动态执行的过程，是操作系统进行资源分配和调度的一个独立单位，是应用程序运行的载体。它由内存空间(代码、数据、进程空间、打开的文件)和一个或多个线程组成。

3.进程和线程的关系：一个进程可以有一个或多个线程，各个线程之间共享程序的内存空间(进程的内存空间)

4.进程和线程的区别

1.线程是程序执行的最小单位；进程是操作系统资源分配的最小单位

2.一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行线路

3.进程之间相互独立，但同一进程下的各个线程之间共享进程的内存空间(包括代码段、数据集、堆等)及一些进程级的资源(如打开的文件和信号)，进程内的线程对其他建成不可见。

4.调度和切换：线程上下文切换比进程上下文切换要快得多

5.用户线程和内核线程的模型：1对1，多对1，多对多(主流现在是多对多)

6.进程和线程的状态：创建、就绪、运行、阻塞、退出

- 进程调度

https://chyyuu.gitbooks.io/simple_os_book/content/zh/chapter-4/process_schedule_principal.html

1.目的：充分利用计算机系统中的CPU资源

2.进程调度的指标：CPU利用率、吞吐量、周转时间、等待时间、响应时间

3.方式：可抢占式和不可抢式

4.策略：先来先服务、短作业优先、时间片轮转、高响应比优先、多级反馈队列、最高优先级先调度算法

- 开启线程的三种方式,run()和start()方法区别

1.开启线程的三种方式

1继承Thread类创建线程类，然后实现run方法，调用start方法开启线程

2.通过实现Runable接口创建线程类，定义Runnable实现类，实现run方法，创建Runnable实现类对象，作为Thread的target来创建Thread，然后调用线程对象的start方法，开启线程

3.

- 动态加载
- OSGI
- 视频加密传输
- 系统启动流程 Zygote进程 –> SystemServer进程 –> 各种系统服务 –> 应用进程
- 树：B+树的介绍
- 图：有向无环图的解释
- 双亲委派模型
- RxJava
- 抽象类和接口的区别
- 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
- 死锁
- 并发集合了解哪些
- CAS介绍
- 常用数据结构简介
- 判断环（猜测应该是链表环）
- 排序，堆排序实现
- 链表反转
- 线程间操作 List

腾讯

- synchronized用法
- volatile用法
- 动态权限适配方案，权限组的概念
- 网络请求缓存处理，okhttp如何处理网络缓存的
- 图片加载库相关，bitmap如何处理大图，如一张30M的大图，如何预防OOM

- 进程保活
- listview图片加载错乱的原理和解决方案

- https相关，如何验证证书的合法性，https中哪里用了对称加密，哪里用了非对称加密，对加密算法（如RSA）等是否有了解
  http://blog.csdn.net/duanbokan/article/details/50847612

客户端在对服务器say hello之后，服务器将公开密钥证书发送给客户端，注意这个证书里面包含了公钥+各种信息+签名（私钥对各种信息加密后生成签名），客户端收到公开密钥证书后，相当于收到了一个包裹里面有公钥+各种信息+签名，怎么样使用这三个数据来校验尼，很简单，公钥加密，私钥解，私钥加密公钥也可以解，只要利用公钥对签名进行解密，然后最和各种信息做比较就可以校验出证书的合法性。

对称加密：使用同一个密钥进行加密和解密；非对称加密：公钥加密，私钥解密。

Https的通信过程使用了对称加密；建立连接过程，使用了非对象加密。

滴滴

- MVP
- 广播（动态注册和静态注册区别，有序广播和标准广播）
- service生命周期
- handler实现机制（很多细节需要关注：如线程如何建立和退出消息循环等等）
- 多线程（关于AsyncTask缺陷引发的思考）
- 数据库数据迁移问题
- 设计模式相关（例如Android中哪里使用了观察者模式，单例模式相关）
- x个苹果，一天只能吃一个、两个、或者三个，问多少天可以吃完
- TCP与UDP区别与应用（三次握手和四次挥手）涉及到部分细节（如client如何确定自己发送的消息被server收到） HTTP相关 提到过

Websocket 问了WebSocket相关以及与socket的区别

- 是否熟悉Android jni开发，jni如何调用java层代码
- 进程间通信的方式
- java注解
- 计算一个view的嵌套层级
- 项目组件化的理解
- 多线程断点续传原理
- Android系统为什么会设计ContentProvider，进程共享和线程安全问题
- jvm相关
- Android相关优化（如内存优化、网络优化、布局优化、电量优化、业务优化）
- EventBus实现原理

美团

- static synchronized 方法的多线程访问和作用，同一个类里面两个synchronized方法，两个线程同时访问的问题
- 内部类和静态内部类和匿名内部类，以及项目中的应用
- handler发消息给子线程，looper怎么启动
- View事件传递
- activity栈
- 封装view的时候怎么知道view的大小
- arraylist和linkedlist的区别，以及应用场景
- 怎么启动service，service和activity怎么进行数据交互
- 下拉状态栏是不是影响activity的生命周期，如果在onStop的时候做了网络请求，onResume的时候怎么恢复
- view渲染

今日头条

- 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和volatile（n面）

- 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
- ReentrantLock的内部实现
- App启动崩溃异常捕捉
- 事件传递机制的介绍
- ListView的优化
- 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
- 模式MVP，MVC介绍
- 断点续传的实现
- 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
- 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

爱奇艺

- 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

百度

- Bitmap 使用时候注意什么?
- Oom 是否可以try catch ?
- 内存泄露如何产生?
- 适配器模式，装饰者模式，外观模式的异同?
- ANR 如何产生?
- String buffer 与string builder 的区别?
- 如何保证线程安全?
- java四中引用
- Jni 用过么?
- 多进程场景遇见过么?
- 关于handler，在任何地方new handler 都是什么线程下
- sqlite升级，增加字段的语句
- bitmap recycler 相关
- 强引用置为null，会不会被回收?
- glide 使用什么缓存?
- Glide 内存缓存如何控制大小?
- 如何保证多线程读写文件的安全?

携程

- 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
- 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

网易

- 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

小米

- String 为什么要设计成不可变的?
- fragment 各种情况下的生命周期
- Activity 上有 Dialog 的时候按 home 键时的生命周期
- 横竖屏切换的时候，Activity 各种情况下的生命周期
- Application 和 Activity 的 context 对象的区别
- 序列化的作用，以及 Android 两种序列化的区别。
- List 和 Map 的实现方式以及存储方式。
- 静态内部类的设计意图。
- 线程如何关闭，以及如何防止线程的内存泄漏

360

- 软引用、弱引用区别
- 垃圾回收
- 多线程：怎么用、有什么问题要注意；Android线程有没有上限，然后提到线程池的上限
- JVM

- 锁
- OOM，内存泄漏
- ANR怎么分析解决
- LinearLayout、RelativeLayout、FrameLayout的特性、使用场景
- 如何实现Fragment的滑动
- ViewPager使用细节，如何设置成每次只初始化当前的Fragment，其他的不初始化
- ListView重用的是什么
- 进程间通信的机制
- AIDL机制
- AsyncTask机制
- 如何取消AsyncTask
- 序列化
- Android为什么引入Parcelable
- 有没有尝试简化Parcelable的使用
- AIDL机制
- 项目：拉活怎么做的
- 应用安装过程

某海外直播公司

- 线程和进程的区别?
- 为什么要有线程，而不是仅仅用进程?
- 算法判断单链表成环与否?
- 如何实现线程同步?
- hashmap数据结构?
- arraylist 与 linkedlist 异同?
- object类的equal 和hashcode 方法重写，为什么?
- hashmap如何put数据（从hashmap源码角度讲解）?
- 简述IPC?
- fragment之间传递数据的方式?
- 简述tcp四次挥手?
- threadlocal原理

- 内存泄漏的可能原因?
- 用IDE如何分析内存泄漏?
- OOM的可能原因?
- 线程死锁的4个条件?
- 差值器&估值器
- 简述消息机制相关
- 进程间通信方式?
- Binder相关?
- 触摸事件的分发?
- 简述Activity启动全部过程?
- okhttp源码?
- RxJava简介及其源码解读?
- 性能优化如何分析systrace?
- 广播的分类?
- 点击事件被拦截，但是相传到下面的view，如何操作?
- Glide源码?
- ActicityThread相关?
- volatile的原理
- synchronize的原理
- lock原理
- 翻转一个单项链表
- string to integer
- 合并多个单有序链表（假设都是递增的）

其他公司

- 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
- 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
- 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
- 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