## 高端技术题

### 一.图片

**1、图片库对比**

**2、LRUCache原理**

LruCache是个泛型类，主要原理是：把最近使用的对象用强引用存储在LinkedHashMap中，当缓存满时，把最近最少使用的对象从内存中移除，并提供get/put方法完成缓存的获取和添加。LruCache是线程安全的，因为使用了synchronized关键字。

当调用put()方法，将元素加到链表头，如果链表中没有该元素，大小不变，如果没有，需调用trimToSize方法判断是否超过最大缓存量，trimToSize()方法中有一个while(true)死循环，如果缓存大小大于最大的缓存值,会不断删除LinkedHashMap中队尾的元素，即最少访问的，直到缓存大小小于最大缓存值。当调用LruCache的get方法时，LinkedHashMap会调用recordAccess方法将此元素加到链表头部。

**3、图片加载原理**

**4、自己去实现图片库，怎么做？**

**5、Glide源码解析**

1）Glide.with(context)创建了一个RequestManager，同时实现加载图片与组件生命周期绑定：在Activity上创建一个透明的ReuqestManagerFragment加入到FragmentManager中，通过添加的Fragment感知Activty\Fragment的生命周期。因为添加到Activity中的Fragment会跟随Activity的生命周期。在RequestManagerFragment中的相应生命周期方法中通过liftcycle传递给在lifecycle中注册的LifecycleListener

2）RequestManager.load(url) 创建了一个RequestBuilder<T>对象 T可以是Drawable对象或是ResourceType等

3) RequestBuilder.into(view)

-->into(glideContext.buildImageViewTarget(view, transcodeClass))返回的是一个DrawableImageViewTarget, Target用来最终展示图片的，buildImageViewTarget-->ImageViewTargetFactory.buildTarget()根据传入class参数不同构建不同的Target对象，这个Class是根据构建Glide时是否调用了asBitmap()方法，如果调用了会构建出BitmapImageViewTarget，否则构建的是GlideDrawableImageViewTarget对象。

-->GenericRequestBuilder.into(Target),该方法进行了构建Request，并用RequestTracker.runRequest()

*Request request = buildRequest(target);//构建Request对象，Request是用来发出加载图片的，它调用了buildRequestRecursive()方法以，内部调用了GenericRequest.obtain()方法*

*target.setRequest(request);*

*lifecycle.addListener(target);*

*requestTracker.runRequest(request);//判断Glide当前是不是处于暂停状态，若不是则调用Request.begin()方法来执行Request，否则将Request添加到待执行队列里，等暂停态解除了后再执行*

-->GenericRequest.begin()

1）onSizeReady()--> Engine.load(signature, width, height, dataFetcher, loadProvider, transformation, transcoder,

            priority, isMemoryCacheable, diskCacheStrategy, this) --> a)先构建EngineKey; b) loadFromCache从缓存中获取EngineResource，如果缓存中获取到cache就调用cb.onResourceReady(cached)； c)如果缓存中不存在调用loadFromActiveResources从active中获取，如果获取到就调用cb.onResourceReady(cached)；d)如果active中也不存在，调用EngineJob.start(EngineRunnable), 从而调用decodeFromSource()/decodeFromCache()-->如果是调用decodeFromSource()-->ImageVideoFetcher.loadData()-->HttpUrlFetcher()调用HttpUrlConnection进行网络请求资源-->得于InputStream()后,调用decodeFromSourceData()-->loadProvider.getSourceDecoder().decode()方法解码-->GifBitmapWrapperResourceDecoder.decode()-->decodeStream()先从流中读取2个字节判断是GIF还是普通图，若是GIF调用decodeGifWrapper()来解码，若是普通静图则调用decodeBitmapWrapper()来解码-->bitmapDecoder.decode()

**6、Glide使用什么缓存？**

1) 内存缓存：LruResourceCache(memory)+弱引用activeResources

Map<Key, WeakReference<EngineResource<?>>> activeResources正在使用的资源，当acquired变量大于0，说明图片正在使用，放到activeResources弱引用缓存中，经过release()后，acquired=0,说明图片不再使用，会把它放进LruResourceCache中

2）磁盘缓存：DiskLruCache,这里分为Source(原始图片)和Result（转换后的图片）

第一次获取图片，肯定网络取，然后存active\disk中，再把图片显示出来，第二次读取相同的图片，并加载到相同大小的imageview中，会先从memory中取，没有再去active中获取。如果activity执行到onStop时，图片被回收，active中的资源会被保存到memory中，active中的资源被回收。当再次加载图片时，会从memory中取，再放入active中，并将memory中对应的资源回收。

之所以需要activeResources，它是一个随时可能被回收的资源，memory的强引用频繁读写可能造成内存激增频繁GC，而造成内存抖动。资源在使用过程中保存在activeResources中，而activeResources是弱引用，随时被系统回收，不会造成内存过多使用和泄漏。

**7、Glide内存缓存如何控制大小？**

Glide内存缓存最大空间(maxSize)=每个进程可用最大内存\*0.4（低配手机是   每个进程可用最大内存\*0.33）

磁盘缓存大小是250MB   int DEFAULT\_DISK\_CACHE\_SIZE = 250 \* 1024 \* 1024;

### 二网络和安全机制

1.网络框架对比和源码分析

2.自己去设计网络请求框架，怎么做？

okhttp源码

3.网络请求缓存处理，okhttp如何处理网络缓存的；

(1)网络缓存优先考虑强制缓存，再考虑对比缓存

--首先判断强制缓存中的数据的是否在有效期内。如果在有效期，则直接使用缓存。如果过了有效期，则进入对比缓存。

--在对比缓存过程中，判断ETag是否有变动，如果服务端返回没有变动，说明资源未改变，使用缓存。如果有变动，判断Last-Modified。

--判断Last-Modified，如果服务端对比资源的上次修改时间没有变化，则使用缓存，否则重新请求服务端的数据，并作缓存工作。

（2）okhttp缓存

开启使用Okhttp的缓存其实很简单，只需要给OkHttpClient对象设置一个Cache对象即可，创建一个Cache时指定缓存保存的目录和缓存最大的大小即可。

*//新建一个cache，指定目录为外部目录下的okhttp\_cache目录，大小为100M*

*Cache cache = new Cache(new File(Environment.getExternalStorageDirectory() + "/okhttp\_cache/"), 100 \* 1024 \* 1024);*

*//将cache设置到OkHttpClient中，这样缓存就开始生效了。*

*OkHttpClient client = new OkHttpClient.Builder().cache(cache).build();*

相关的类有：

1）CacheControl( HTTP中的Cache-Control和Pragma缓存控制）：指定缓存规则

2）Cache(缓存类)

3）DiskLruCache(文件化的LRU缓存类）

（1）读取缓存：先获限OkHttpClient的Cache缓存对象，就是上面创建OkHttpClient设置的Cahce; 传Request请求到Cache的get方法查找缓存响应数据Response；构造一个缓存策略，再调用它的get去决策使用网络请求还是缓存响应。若使用缓存，它的cacheResponse不为空,networkRequest为空，用缓存构造响应直接返回。若使用请求，则cacheResponse为空,networkRequest不为空，开始网络请求流程。

Cache的get获取缓存方法，计算request的key值（请求url进行md5加密），根据key值去DisLruCache查找是否存在缓存内容，存则则创建绘存Entry实体。ENTRY\_METADATA代表响应头信息，ENTRY\_BODY代表响应体信息。如果缓存存在，在指定目录下会有两个文件\*\*\*\*.0    \*\*\*\*\*.1分别存储某个请求缓存响应头和响应体信息。

CacheStrategy的get方法：1）若缓存响应为空或 2）请求是https但缓存响应没有握手信息；3）请求和缓存响应都是不可缓存的；4）请求是onCache，并且又包含if-Modified-Since或If-None-Match则不使用缓存； 再计算请求有效时间是否符合响应的过期时间，若响应在有效范围内，则缓存策略使用缓存，否则创建一个新的有条件的请求，返回有条件的缓存策略。

（2）存储缓存流程：从HttpEngine的readResponse()发送请求开始，判断hasBody(userResponse),如果缓存的话，maybeCache()缓存响应头信息，unzip(cacheWritingResponse(storeRequest, userResponse))缓存响应体。

**4.从网络加载一个10M的图片，说下注意事项**

**5.TCP的3次握手和四次挥手**

**6.TCP与UDP的区别**

**7.TCP与UDP的应用**

**8.HTTP协议**

**9.HTTP1.0与2.0的区别**

**10.HTTP报文结构**

**11.HTTP与HTTPS的区别以及如何实现安全性**

**12.如何验证证书的合法性?**

**13.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?**

**14.client如何确定自己发送的消息被server收到?**

**15.谈谈你对WebSocket的理解**

**16.WebSocket与socket的区别**

**17.谈谈你对安卓签名的理解。**

**18.请解释安卓为啥要加签名机制?**

**19.视频加密传输**

**20.App 是如何沙箱化，为什么要这么做？**

**21.权限管理系统（底层的权限是如何进行 grant 的）？**

### 三.数据库

**1.sqlite升级，增加字段的语句**

**2.数据库框架对比和源码分析**

**3.数据库的优化**

**4.数据库数据迁移问题**

### 四.算法

**1.排序算法有哪些？**

**2.最快的排序算法是哪个？**

**3.手写一个冒泡排序**

**4.手写快速排序代码**

**5.快速排序的过程、时间复杂度、空间复杂度**

**6.手写堆排序**

**7.堆排序过程、时间复杂度及空间复杂度**

**8.写出你所知道的排序算法及时空复杂度，稳定性**

**9.二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**

**10给阿里2万多名员工按年龄排序应该选择哪个算法？**

**11.GC算法(各种算法的优缺点以及应用场景)**

**12.蚁群算法与蒙特卡洛算法**

**13.子串包含问题(KMP 算法)写代码实现**

**14一个无序，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**

**15.万亿级别的两个URL文件A和B，如何求出A和B的差集C(提示：Bit映射->hash分组->多文件读写效率->磁盘寻址以及应用层面对寻址的优化)**

**16.百度POI中如何试下查找最近的商家功能(提示：坐标镜像+R树)。**

**17.两个不重复的数组集合中，求共同的元素。**

**18.两个不重复的数组集合中，这两个集合都是海量数据，内存中放不下，怎么求共同的元素？**

**19.一个文件中有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**

**20.一张Bitmap所占内存以及内存占用的计算**

一张图片(bitmap)占用的内存影响因素：图片原始长、宽，手机屏幕密度，图片存放路径下的密度，单位像素占用字节数

bitmapSize=图片长度\*（inTargetDensity手机的density / inDensity图片存放目录的density）\*宽度\*（手机的inTargetDensity / inDensity目标存放目录的density）\*单位像素占用的字节数（图片长宽单位是像素）

1）图片长宽单位是像素：单位像素字节数由其参数BitmapFactory.Options.inPreferredConfig变量决定，它是Bitmap.Config类型，包括以下几种值：ALPHA\_8图片只有alpha值，占用一个字节；ARGB\_4444 一个像素占用2个字节,A\R\G\B各占4bits；ARGB\_8888一个像素占用4个字节，A\R\G\B各占8bits（高质量图片格式，bitmap默认格式）；ARGB\_565一个像素占用2字节，不支持透明和半透明，R占5bit, Green占6bit, Blue占用5bit. 从Android4.0开始该项无效。

2） inTargetDensity 手机的屏幕密度(跟手机分辨率有关系)

inDensity原始资源密度（mdpi:160;   hdpi:240;   xhdpi:320;   xxhdpi:480; xxxhdpi:640）

当Bitmap对象在不使用时，应该先调用recycle()，再将它设置为null，虽然Bitmap在被回收时可通过BitmapFinalizer来回收内存。但只有系统垃圾回收时才会回收。Android4.0之前，Bitmap内存分配在Native堆中，Android4.0开始，Bitmap的内存分配在dalvik堆中，即Java堆中，调用recycle()并不能立即释放Native内存。

1. **2000万个整数，找出第五十大的数字？**

**22.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？**

**23.求1000以内的水仙花数以及40亿以内的水仙花数**

**24. 5枚硬币，2正3反如何划分为两堆然后通过翻转让两堆中正面向上的硬8币和反面向上的硬币个数相同**

**25.时针走一圈，时针分针重合几次**

**26.N\*N的方格纸,里面有多少个正方形**

**27.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？**

### 五.插件化、模块化、组件化、热修复、增量更新、Gradle

**1.对热修复和插件化的理解**

**2.插件化原理分析**

**3.模块化实现（好处，原因）**

**4.热修复,插件化**

**5.项目组件化的理解**

**6.描述清点击 Android Studio 的 build 按钮后发生了什么**

### 六.架构设计和设计模式

**1.谈谈你对Android设计模式的理解**

**2.MVC MVP MVVM原理和区别**

**3.你所知道的设计模式有哪些？**

**4.项目中常用的设计模式**

**5.手写生产者/消费者模式**

**6.写出观察者模式的代码**

**7.适配器模式，装饰者模式，外观模式的异同？**

**8.用到的一些开源框架，介绍一个看过源码的，内部实现过程。**

**9.谈谈对RxJava的理解**

RxJava是基于响应式编程，基于事件流、实现异步操（类似于Android中的AsyncTask、Handler作用）作的库，基于事件流的链式调用，使得RxJava逻辑简洁、使用简单。RxJava原理是基于一种扩展的观察者模式，有四种角色：被观察者Observable 观察者Observer 订阅subscribe 事件Event。RxJava原理可总结为：被观察者Observable通过订阅(subscribe)按顺序发送事件（Emitter)给观察者(Observer)， 观察者按顺序接收事件&作出相应的响应动作。

RxJava中的操作符：

1）defer():直到有观察者(Observer)订阅时，才会动态创建被观察者对象(Observer)&发送事件，通过Observer工厂方法创建被观察者对象，每次订阅后，都会得到一个刚创建的最新的Observer对象，可以确保Observer对象里的数据是最新的。defer()方法只会定义Observable对象，只有订阅操作才会创建对象。

*Observable<T> observable = Observable.defer(new Callable<ObservableSource<? extends T>>() {*

*@Override*

*public ObservableSource<? extends T> call() throws Exception {*

*return Observable.just();*

*}*

*}*

2）timer() 快速创建一个被观察者(Observable)，延迟指定时间后，再发送事件

*Observable.timer(2, TimeUnit.SECONDS)//也可以自定义线程timer(long, TimeUnit, Scheduler)*

*.subscribe(new Observer<Long>() {*

*@Override*

*public void onSubscribe(Disposable d) {*

*}*

*...*

*});*

3) interval() intervalRange() 快速创建一个被观察者对象（Observable)，每隔指定时间就发送事件

*//interval三个参数，参数1：第一次延迟时间 参数2：间隔时间数字 参数3：时间单位*

*Observable.interval(3, 1, TimeUnit.SECONDS).subscribe();*

*//intervalRange五个参数，参数1：事件序列起始点 参数2：事件数量 参数3：第一次延迟时间 参数4：间隔时间数字 参数5：时间单位*

*Observable.intervalRange(3, 10, 2, 1, TimeUnit.SECONDS).subscribe();*

*RxJava的功能与原理实现*

**10.Rxjava发送事件步骤：**

1）创建被观察者对象Observable&定义需要发送的事件

*Observable.create(new ObservableOnSubscribe<T>(){*

*@Override*

*public void subscribe(ObservableEmitter<T> emitter) throws Exception {*

*//定义发送事件的行为*

*}*

*});*

Observable.create()方法实际创建了一个ObservableCreate对象，它是Observable的子类，传入一个ObservableOnSubscribe对象，复写了发送事件行为的subscribe()方法。

2）创建观察者对象Observer&定义响应事件的行为

*Observer observer = new Observer<T>() {*

*@Override*

*public void onSubscribe(Disposable d){//Disposable对象可用于结束事件*

*//默认最先调用*

*}*

*@Override*

*public void onNext(T t){*

*}*

*@Override*

*public void onError(Throwable d){*

*}*

*@Override*

*public void onComplete(){*

*}*

*}*

3）通过subscribe()方法使观察者订阅被观察者

*Observable.subscribe(Observer observer);//实际调用的是ObservableCreate.subscribeActual()方法，具体实现如下*

*protected void subscribeActual(Observer<? super T> observer) {*

*// 1. 创建1个CreateEmitter对象用于发射事件（封装成1个Disposable对象）*

*CreateEmitter<T> parent = new CreateEmitter<T>(observer);*

*// 2. 调用观察者（Observer）的onSubscribe（）*

*observer.onSubscribe(parent);*

*try {*

*// 3. 调用source对象的（ObservableOnSubscribe对象）subscribe（）*

*source.subscribe(parent);*

*} catch (Throwable ex) {*

*Exceptions.throwIfFatal(ex);*

*parent.onError(ex);*

*}*

*}*

**11.RxJava的作用，与平时使用的异步操作来比的优缺点**

**12.说说EventBus作用，实现方式，代替EventBus的方式**

**13.从0设计一款App整体架构，如何去做？**

**14.说一款你认为当前比较火的应用并设计(比如：直播APP，P2P金融，小视频等)**

**15.谈谈对java状态机理解**

**16.Fragment如果在Adapter中使用应该如何解耦？**

**17.Binder机制及底层实现**

**18.对于应用更新这块是如何做的？(解答：灰度，强制更新，分区域更新)？**

**19.实现一个Json解析器(可以通过正则提高速度)**

**20.统计启动时长,标准**

### 七.性能优化

**1.如何对Android 应用进行性能分析以及优化?**

**2.ddms 和 traceView**

**3.性能优化如何分析systrace？**

**4.用IDE如何分析内存泄漏？**

**5.Java多线程引发的性能问题，怎么解决？**

**6.启动页白屏及黑屏解决？**

**7.启动太慢怎么解决？**

**8.怎么保证应用启动不卡顿？**

**9.App启动崩溃异常捕捉**

**10自定义View注意事项**

**11.现在下载速度很慢,试从网络协议的角度分析原因,并优化(提示：网络的5层都可以涉及)。**

**12.Https请求慢的解决办法（提示：DNS，携带数据，直接访问IP）**

**13.如何保持应用的稳定性**

**14.RecyclerView和ListView的性能对比**

**15.ListView的优化**

**16.RecycleView优化**

**17.View渲染**

**18.Bitmap如何处理大图，如一张30M的大图，如何预防OOM**

**19.java中的四种引用的区别以及使用场景**

**20.强引用置为null，会不会被回收？**

### 八.NDK、jni、Binder、AIDL、进程通信有关

**1.请介绍一下NDK**

**2.什么是NDK库?**

**3.jni用过吗？**

**4.如何在jni中注册native函数，有几种注册方式?**

**5.Java如何调用c、c++语言？**

**6.jni如何调用java层代码？**

**7.进程间通信的方式？**

**8.Binder机制**

**9.简述IPC？**

**10.什么是AIDL？**

**11.AIDL解决了什么问题？**

**12.AIDL如何使用？**

**13.Android 上的 Inter-Process-Communication 跨进程通信时如何工作的？**

**14.多进程场景遇见过么？**

**15.Android进程分类？**

**16.进程和 Application 的生命周期？**

**17.进程调度**

**18.谈谈对进程共享和线程安全的认识**

**19谈谈对多进程开发的理解以及多进程应用场景**

1. **什么是协程？**

### 九.framework层、ROM定制、Ubuntu、Linux之类的问题

**1.java虚拟机的特性**

**2.谈谈对jvm的理解**

**3.JVM内存区域，开线程影响哪块内存**

**4.对Dalvik、ART虚拟机有什么了解？**

**5.Art和Dalvik对比**

**6.虚拟机原理，如何自己设计一个虚拟机(内存管理，类加载，双亲委派)**

**7.谈谈你对双亲委派模型理解**

**8.JVM内存模型，内存区域**

**9.类加载机制**

**10.谈谈对ClassLoader(类加载器)的理解**

**11.谈谈对动态加载（OSGI）的理解**

**12.内存对象的循环引用及避免**

**13.内存回收机制、GC回收策略、GC原理时机以及GC对象**

**14.垃圾回收机制与调用System.gc()区别**

**15.Ubuntu编译安卓系统**

**16.系统启动流程是什么？（提示：Zygote进程 –> SystemServer进程 –> 各种系统服务 –> 应用进程）**

**17.大体说清一个应用程序安装到手机上时发生了什么**

**18.简述Activity启动全部过程**

**19.App启动流程，从点击桌面开始**

**20.逻辑地址与物理地址，为什么使用逻辑地址？**

**21.Android为每个应用程序分配的内存大小是多少？**

**22.Android中进程内存的分配，能不能自己分配定额内存？**

**23.进程保活的方式**

**24.如何保证一个后台服务不被杀死？（相同问题：如何保证service在后台不被kill？）比较省电的方式是什么？**

**25.App中唤醒其他进程的实现方式**