# 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**

https://www.cnblogs.com/cr330326/p/5585021.html
https://www.jianshu.com/p/448570005e81

**4、自己去实现图片库，怎么做？**

https://blog.csdn.net/guolin_blog/article/details/53759439

**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;

Android 中 onTouch 与 onClick 两种监听的完全解析

https://blog.csdn.net/l1530304349/article/details/53036276

**二网络和安全机制**

1. 网络框架对比和源码分析
2. https://blog.csdn.net/small_and_smallworld/article/details/72811227
   https://www.jianshu.com/p/86fff768b063

3. 自己去设计网络请求框架，怎么做？
   https://blog.csdn.net/l1530304349/article/details/53407634

okhttp 源码
https://www.jianshu.com/p/37e26f4ea57b

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。

https://www.jianshu.com/p/73935e04a213

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
https://www.cnblogs.com/lzq198754/p/5780310.html

9.HTTP1.0 与 2.0 的区别
10.HTTP 报文结构
https://www.cnblogs.com/zhuifeng/p/4072248.html

11.HTTP 与 HTTPS 的区别以及如何实现安全性
https://www.cnblogs.com/wqhwe/p/5407468.html

https://blog.csdn.net/weixin_40423553/article/details/79202977

12.如何验证证书的合法性?
https://www.cnblogs.com/funny11/p/6978908.html

https://www.jianshu.com/p/ee919760b5f5

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?
https://blog.csdn.net/tenfyguo/article/details/40922813
https://blog.csdn.net/tenfyguo/article/details/40958727

14.client 如何确定自己发送的消息被 server 收到?
https://blog.csdn.net/yinxianluo/article/details/8932080

15.谈谈你对 WebSocket 的理解
https://blog.csdn.net/jing_80/article/details/82111423
https://blog.csdn.net/qq_33566350/article/details/75508219

16.WebSocket 与 socket 的区别

https://www.cnblogs.com/zyy1688/p/10002089.html
https://blog.csdn.net/wwd0501/article/details/54582912

**17.谈谈你对安卓签名的理解。**
https://blog.csdn.net/glzhu2/article/details/73174436
https://www.cnblogs.com/jpfss/p/9889630.html

**18.请解释安卓为啥要加签名机制?**
https://blog.csdn.net/hubert_bing/article/details/69964740
https://blog.csdn.net/u014225510/article/details/50937013

1、应用程序升级，验证 app 的唯一性，包名和签名都一致才允许升级。

2、应用程序模块化，可以模块化部署多个应用到一个进程，只要他们的签名一样。

3、代码或者数据共享，同一个签名有相同的权限，可以共享数据和代码。

**19.视频加密传输**
https://blog.csdn.net/shenshibaoma/article/details/79003854
要处理这个问题首先我们要知道为何视频需要进行加密？因为 **盗链** 和 **盗播** 的存在，让版权价值大打折扣。用户通过一次付费行为，就可以拿到付费视频的播放 URL，将播放 URL 进行二次分发，这种行为叫做盗链；用户直接将视频下载到本地，然后再进行二次上传分发，这种行为叫做盗播
常用的视频加密技术有：

**1、分片加密**

基于苹果公司的 HLS 协议，服务器需要切片出 TS 文件并进行 AES 加密，生成 m3u8 索引文件，然后客户端下载到本地给播放器播放。

**2、文件的前中后加密**

这三段加密都涉及到文件的读取，尤其对于大文件的读取，我们使用 java 的 RandomAccessFile（随机访问文件）
这个类来进行文件的加密和解密.

**20.App 是如何沙箱化，为什么要这么做？**
https://blog.csdn.net/ljheee/article/details/53191397

沙箱是为 app 提供隔离环境的一种安全机制，严格控制执行的程序所访问的资源，以确保系统的安全，让 app 在独立的进程中执行任务，让其不能访问外部进程的资源，这样一个应用出问题了，其他的应用进程能够保证不被影响。

**21.权限管理系统（底层的权限是如何进行 grant 的）？**
应用程序在应用层的 AndroidManifest.xml 中所申请的权限将会在 Android 系统启动时，经过解析后，逐步映射到内核层的组 ID 和用户 ID，最终由内核层的 setgid()

和 setuid()函数设置后才能执行；在进行权限申请时，6.0 以上需要动态代码申请，6.0 以下直接在 AndroidManifest.xml 注册即可。

Gc 回收算法？
https://www.jianshu.com/p/fe6d8318d0ae

java 内存区域及内存溢出异常
https://www.jianshu.com/p/e956b9de864e
https://www.jianshu.com/p/278e611f87eb

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

https://www.jianshu.com/p/a34fc64f84b2

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。

**21.　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
20. 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