**网络相关**

* Https的原理？
  + 应用层和网络层增加了基于TLS协议的SSL层，通过TLS握手过程交换加密秘钥，后续传输内容使用秘钥进行对称加密，达到安全传输的目的。
* Http三次握手过程？
  + client：syn=1，seq=x
  + server：syn=1，ack=x+1，seq=y
  + client：ack=y+1，seq=z
* Tcp和Udp的区别？
  + Tcp面向链接，可靠的传输层协议。
  + Udp是面向事务，简单不可靠传输层协议。
* Http的报文结构？cookie是用来干嘛的？有哪些响应码，分别都代表什么意思？
  + 请求头、请求行、请求数据，状态行、响应头、响应数据
  + 因为http是无状态协议，cookie用来保存客户端状态。服务端响应头带上Set-Cookie返回cookie信息，客户端请求头Cookie带上cookie信息
  + 206 断点续传、3xx重定向、400 Bad Request、403 Forbidden、404 Not Found、500 Internal Server Error、502 Bad Gateway、503 Service Unavailable、504 Gateway Timeout
* 有自己实现过Socket协议吗？

tcpip和http都是协议是约定好的规范，他们位于网络5层模型的传输层（tcp）和应用层（http），tcpip表示的是一系列协议。规定好的协议总要操作系统实现了才能使用，而socket就是操作系统实现的，tcpip协议族的接口，用于创建一个套接字，可以理解为，通过网络读写的文件描述符，socket、bind、listen、accept、connect一系列都是操作系统提供的接口用于实现tcp协议相关的功能。

**多线程**

* 死锁产生的条件？有哪些死锁类型？
  1. 互斥条件、请求与保持条件、不剥夺条件、循环等待条件
  2. 静态的锁顺序死锁、动态的锁顺序死锁、协作对象之间产生的死锁
* CAS，乐观锁
  1. CAS就是一种乐观锁思想的应用，使用while死循环，直到成功为止，java.util.concurrent.atomic包相关类就是CAS的实现，如AtomicInteger
* AQS
  1. AQS，即AbstractQueuedSynchronizer，提供了一个基于FIFO队列，可以用于构建锁或者其他相关同步装置的基础框架
  2. synchronized是悲观锁，ReentrantLock可利用AQS实现公平锁和非公平锁。
* Sleep()和wait()的区别？
  1. wait会释放锁，sleep等待过程仍然持有锁
  2. sleep是Thread静态方法，可以直接调用。wait是对象方法，而且需要在synchronized块中调用
  3. wait可以被notify和notifyAll唤醒，sleep不行
* Java有哪些线程池？他们的区别是什么？线程池工作流程是怎样的？线程池实现原理是怎样的？Cache线程池有哪些弊端？

**线程池任务执行逻辑**：线程池线程数量未达到核心线程数则新建核心线程，超过核心线程数则插入排队队列等待，若队列已满但未超出线程池容量，新启动临时线程执行任务，若超出线程池容量则执行拒绝策略。

* 1. newFixedThreadPool 核心线程数和线程池大小容量一致并可配置，核心线程已满则进入排队队列等待，因为队列使用容量为Interger.Max的LinkedBlockingQueue，所以不会执行拒绝策略；

newCachedThreadPool核心线程数为0，线程池大小为Max，任务队列使用无任何容量的SynchronousQueue，所以当来任务的会新建线程处理，闲置线程60s之后被回收。特点是闲置时几乎不占用系统资源；

newScheduledThreadPool 核心线程数固定，线程池大小为Max，闲置存活时间为10ms，常用于定时任务执行；newSingleThreadPool 核心线程和线程池大小都为1，任务队列使用容量为Interger.Max的LinkedBlockingQueue，也就是说只有一个任务处于活动状态，其他任务在队列中排队等候依次执行。

* 1. 线程池实现原理：ThreadPoolExecutor中内部类Worker为核心线程实现类，核心逻辑是execute()里面的addWorker()，thread的start()实际上执行的是worker类的runWork()，取出execute()传入的runnable执行，然后一直轮询读取队列里面的任务，如果队列没有任务，则会挂起不占用cpu资源。
  2. 任务较多的时候，一直新建进程执行任务，占用内存资源，有可能会OOM
* 多线程间的通信方式？
  1. synchronized关键字，wait()/notify()
  2. ReentranLock，condition.await()/condition.signal()
* synchronize关键字怎么用的？还知道哪些同步的方式？
  1. 线程同步
     + 方法同步

|  |  |
| --- | --- |
| 1 | public synchronized void method1 |

* + - 代码块同步，同方法同步，锁的类实例

|  |  |
| --- | --- |
| 1 | synchronized(this){ *//TODO }* |

* + - 静态方法同步

|  |  |
| --- | --- |
| 1 | public synchronized static void method3 |

* + - 类代码块同步，同静态方法同步，锁的是类

|  |  |
| --- | --- |
| 1 | synchronized(Test.class){ *//TODO}* |

* + - 对象代码块同步

|  |  |
| --- | --- |
| 1 | synchronized(o) {} |

* 1. ReentranLock、CountDown、线程同步的数据结构ConcurrentHashMap
* Thread直接调用run方法会怎么样？start方法作用是什么？
  1. run只普通类方法，start方法是启动线程，真正实现了多线程运行
* volatile关键字的作用是什么？
  1. 内存可见、禁止重排序，由cpu指令内存屏障实现。
  2. 内存屏障，又称内存栅栏，是一个CPU指令，基本上它是一条这样的指令：  
     1、保证特定操作的执行顺序。  
     2、影响某些数据（或则是某条指令的执行结果）的内存可见性。
* 怎么安全停止一个线程任务？原理是什么？线程池里有类似机制吗？
  1. 标志位或者interrupt()方法
  2. 线程池有shunDown()和shunDownNow()。
     + shutdown()：不会立即终止线程池，而是要等所有任务缓存队列中的任务都执行完后才终止，但再也不会接受新的任务
     + shutdownNow()：立即终止线程池，并尝试打断正在执行的任务，并且清空任务缓存队列，返回尚未执行的任务
* ANR中线程的有几种状态，分别是什么原因造成？
  1. Java中线程状态有NEW、RUNNING、BLOCKED、WAITING、TIMED\_WAITING、TERMINATED
  2. anr中的线程状态有十种，对应Java线程状态的六种

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 | static final Thread.State[] STATE\_MAP = new Thread.State[] {  Thread.State.TERMINATED, *// ZOMBIE*  Thread.State.RUNNABLE, *// RUNNING*  Thread.State.TIMED\_WAITING, *// TIMED\_WAIT*  Thread.State.BLOCKED, *// MONITOR*  Thread.State.WAITING, *// WAIT*  Thread.State.NEW, *// INITIALIZING*  Thread.State.NEW, *// STARTING*  Thread.State.RUNNABLE, *// NATIVE*  Thread.State.WAITING, *// VMWAIT*  Thread.State.RUNNABLE *// SUSPENDED*  }; |

**数据结构**

* HashMap和HashTable的区别？和 ConcurrentHashMap 区别？和LinkedHashMap区别？内部实现原理？
  1. HashMap的key、value允许null，HashTable的key、value不允许null；HashMap的方法线程不同步，HashTable的方法线程同步
  2. ConcurrentHashMap是方法是线程同步的，使用分段加锁的机制实现。segment+lock；1.8以后使用CAS实现线程同步
  3. 数组+双向链表，保持了插入顺序。构造方法的参数accessOrder设置为true则为访问顺序，为false，则为插入顺序
* LRUCache的原理？

使用LinkedHashMap存储数据，accessOrder设置为true，即最近最少访问的放到队尾，最近访问的放到队首，实现最近最少使用算法。

* ArrayList和LinkedList区别？为什么ArrayList不是线程安全的？
  1. ArrayList是的实现是数组，LinkedList的实现是双向链表，ArrayList按索引访问较快，LinkedList插入、删除元素较快
  2. 因为方法没用同步
* 数据库的索引用的什么数据结构？
  1. B+ tree

**虚拟机**

* [垃圾回收机制](https://app.yinxiang.com/shard/s8/nl/1115234/754fb33b-62e2-43f8-b0e0-56a59e0d5225/)？有哪些对象可以作为GC roots？
  1. 根据引用计数算法/可达性算法判断对象是否可以回收，分代回收，标志回收算法
  2. 虚拟机栈（栈帧中的本地变量表）中引用的对象；

方法区中类静态属性引用的对象；

方法区中常量引用的对象；

本地方法栈中JNI（即一般说的Native方法）引用的对象；

* 几种GC类型
  1. GC\_CONCURRENT/kGcCauseBackground，当内存达到一定阀值时会出发GC
  2. GC\_FOR\_MALLOC/kGcCauseForAlloc，当要分配内存的时候发现内存不够的情况下引起的GC
  3. GC\_EXPLICIT/kGcCauseExplicit，当程序中调用System.gc()方法触发
* 跟Art、Dalvik对比
  1. Dalvik是Android平台的虚拟机，2.2以上引入，在程序运行时使用Just-in-time即时编译对dex文件翻译成native code去执行
  2. Apk安装过程中，Dalvik会将得出dex文件执行dexopt进行字节码的优化，而Art会执行dex2oat讲dex字节码翻译成本地机器码，最终都是odex文件。

Art是Android runtime，Dalvik的升级版，4.4以上引入，在程序安装时使用Ahead-of-time预编译用dex2oat生成nativecode。

* 1. 垃圾回收机制，Art对比Dalvik使用部分并发回收，锁住java堆，进行标记回收。
  2. 新增Large-Object-Space内存空间专门存放large object
* Java内存模型？
  1. Java内存模型规定了所有变量存在主内存中，每条线程有自己的工作内存，工作内存存储从主内存拷贝过来的变量。线程对变量的所有操作在工作内存中进行。不同线程之间无法直接访问对方工作内存的变量，线程间变量值的传递需要通过主内存来完成。
* Java内存区域？
  1. 堆（对象，数组），方法区（类信息，静态变量，运行时常量池），虚拟机栈（基本数据类型，方法局部变量，），本地方法栈，程序计数器
* 类加载机制？双亲委托模型？
  1. 把描述类的数据从class文件加载到内存，并对数据进行校验、转换解释和初始化，最终形成可以被虚拟机直接使用的Java类型。类的生命周期包括加载、验证、准备、解析、初始化、使用、卸载。
  2. 如果一个**类加载器**收到了类加载的请求，它首先不会自己尝试加载这个类，而是把请求委托给父加载器去完成，依次向上，所有的类加载器请求最终都应该被传递到顶层的**启动类加载器**中，只有当父加载器在它的搜索范围中没有找到所需的类时，即无法完成该加载，子加载器才会尝试自己去加载此类。

**操作系统**

* 进程与线程的区别？
  + 进程是进程实体运行的过程，是系统进行资源分配和调度的一个独立单位
  + 线程是比进程更小的可独立运行的基本单位，引入是为了减少程序在并发执行过程的开销

**Android系统**

* Binder通信
  + Binder线程池：每个 Server 进程在启动时创建一个 binder 线程池，并向其中注册一个 Binder 线程；之后 Server 进程也可以向 binder 线程池注册新的线程，或者 Binder 驱动在探测到没有空闲 binder 线程时主动向 Server 进程注册新的的 binder 线程。对于一个 Server 进程有一个最大 Binder 线程数限制，默认为16个 binder 线程，例如 Android 的 system\_server 进程就存在16个线程。对于所有 Client 端进程的 binder 请求都是交由 Server 端进程的 binder 线程来处理的

**其它**

* 用过哪些设计模式？DCL单例模式为什么要两次判空？Android里源码或者你用过的开源库都用到了什么设计模式？
  1. builder模式、单例模式、工厂方法模式
  2. 防止jvm即时编译时指令重排序导致的判空异常
  3. Glide使用builder模式、适配器模式、策略模式
* — final关键字的作用？
  1. final修饰的类不可被继承，final修饰的变量不可被修改，final方法不可被重写
  2. 好处是提高性能，使用final关键字jvm会对方法、变量及类进行优化
* 静态内部类和内部类的区别？
  1. 静态内部类不持有外部类引用
* 值传递类问题
* Kotlin为什么能和Java混编

**Android题**

* 你有什么亮点？项目中遇到过什么难题或者坑，怎么解决的？
  1. 擅长架构分层设计，应用的性能优化
  2. Android N的适配问题，项目中使用webview，但Application有需要需要加上android:defaultToDeviceProtectedStorage="true"导致报WebView cannot be used with device protected storage异常，查看资料和源码，通过反射调用context内部方法createCredentialProtectedStorageContext赋值web初始化。
* 做过哪些性能优化？是怎么评测和具体优化的？
* Activity的冷启动流程？AMS的作用？
* 怎么分析内存泄漏？
* View的事件分发机制？滑动冲突怎么解决？
* 自定义View的原理和流程？
* Handler原理？（一般会花式扩展），loop方法死循环，cpu空转？
  1. 消息机制，由Handler/Looper/MessageQueue等构成，一般在主线程处理消息，MessageQueue里面数据结构主要是链表
  2. Android使用epool机制，nativepollonce进入睡眠等待，等到有事件到达时再唤醒
* 有哪些多进程通信方式？Binder机制？为什么使用Binder机制？
  1. AIDL、广播、socket、Messenger、ContentProvider、文件共享
  2. Android系统跨进程通信机制，基于C/S架构，通过**注册服务**，**获取服务**和**使用服务**三个步骤实现。每个进程运行在各自的虚拟地址空间，由用户空间和内核空间组成，各自的用户空间是不能共享，但内核空间在共享的，**进程间通信就是利用进程间可共享的内核空间来完成底层的通信工作**。Client端进程和Server端进程往往采用ioctl等方法与内核空间的驱动进行交互。
  3. Binder数据拷贝只需要一次，而socket需要两次，共享内存方式一次内存拷贝都不需要，但实现复杂。

[](http://gityuan.com/images/binder/java_binder/java_binder.jpg)

* AIDL文件的认识？
  1. aidl，接口定义语言，文件后缀为aidl。程序构建时自动生成对应的同名java文件，里面实现了Binder注册服务和获取服务的代码，分别是Stub里面的attachInterface和asInterface方法，asInterface方法返回内部类Proxy的，即Binder客户端。一般在Service中实现Stub类，并返回asBinder对象为Stub，即Binder服务端。
* Android的生命周期和启动模式相关？
* 你项目中用到哪些开源库？说说其实现原理？（OKhttp、RxJava、Retrofit重点，如果有用到的话）
* Android的打包流程？apk里有哪些东西？签名算法的原理？
  1. 打包流程
     1. aapt工具打包资源文件，生成R.java、resource.arsc和res文件
     2. 处理aidl文件，生成java文件
     3. java compiler编译R.java、java源文件，生成.class文件
     4. 通过dx命令，.class文件生成classes.dex
     5. 通过apkbuilder将resource.arsc、res文件、assets文件和classes.dex一起打成apk包
     6. 通过jarsigner工具，进入签名
     7. 通过zipalign工具，将签名后的apk进行对齐处理。
* 了解哪些插件化技术？
* LinearLayout的布局流程？
* MVC,MVP,MVVM理解？
  1. MVC。View：Xml布局文件，Model：数据模型，数据的获取、存储、状态变化，Controller：对象Activity，处理数据，业务和UI。但是作为View的XML功能太弱，导致Activity充斥了大量view和controller的代码，造成Activity代码臃肿。
  2. MVP。M仍然是数据模型，V是xml和Activity，处理view的绘制和与用户的交互，P是presenter，负责M与V的交互。核心理念是通过一个view接口将presenter与view进行解耦，让Activity只负责view的事务。缺点是，V层和P层存在一定的耦合度，V层修改导致P层也要修改，而且view的接口粒度不好控制，容易造成大量接口存在。UI驱动模型。
  3. MVVM。M仍然是数据模型，V是xml和Activity，VM就是ViewModel，通过数据绑定监听，实现了V层和ViewModel层的解耦
* Android怎么做保活？
* AsyncTask与HandlerThread区别？
  1. AsyncTask内部由线程池和Handler构成，串行执行任务，任务在异步线程里执行，通过handler切换到主线程更新执行结果；
  2. HandlerThread可以用来**执行多个耗时操作而不需要多次启动线程**，里面使用Handler和Looper实现，但任务执行是串行的。
  3. AsyncTask与HanderThread的区别在与AsyncTask使用了线程池并且能在ui线程更新数据，而HandlerThread主要是执行后台线程操作。
* Parcelable与Serializable区别？Parcelable为啥性能高
  1. Parcelable性能更好，实现上也有区别
  2. Parcelable使用正式的序列化处理代替反射，Serializable使用反射创建大量临时对象，可能引起gc频繁回收
* 相同优先级的广播如何保证优先获取到
  1. 动态注册优于静态注册，动态注册先注册的优于后注册，静态注册先扫描的优于后扫描
* requestLayout和invalidate的区别
  1. view的invalidate不会导致ViewRootImpl的invalidate被调用，而是递归调用父view的invalidateChild，直到ViewRootImpl的invalidateChildInParent，然后触发peformTraversals，会导致当前view被重绘,由于mLayoutRequested为false，不会导致onMeasure和onLayout被调用，而OnDraw会被调用，viewgroup的invalidate触发子view的draw(Canvas canvas, ViewGroup parent, long drawingTime)
  2. requestLayout会直接递归调用父窗口的requestLayout，直到ViewRootImpl,然后触发peformTraversals，由于mLayoutRequested为true，会导致onMeasure和onLayout被调用。不一定会触发OnDraw ,t触发onDraw可能是因为在在layout过程中发现l,t,r,b和以前不一样，那就会触发一次invalidate，所以触发了onDraw，也可能是因为别的原因导致mDirty非空（比如在跑动画）
  3. 只要刷新的时候就调用invalidate，需要重新measure就调用requestLayout，后面再跟个invalidate（为了保证重绘）
* MultiDex认识？为什么需要MultiDex？MultiDex分包原理？高版本上还需要分包嘛？
  1. 在程序构建的时候，单个dex超过65536方法，因为dex文件使用short类型来索引dex文件中的方法，最长不能超过65536个。
  2. 程序构建的时候通过dx工具进行dex分包，避免单个dex包超出65536个方法
  3. 拆包
     1. 通过mainDexClasses脚本生成maindexlist.txt，txt里面保存了四大组件等class和直接依赖的class，最后通过create{flavor}{buildType}MainDexClassListTask根据maindexlist生成主dex。
  4. 安装
     1. 在Application的生命周期方法attachBaseContext方法执行dex的读取和安装。通过反射获取BaseDexClassLoader的pathList对象DexPathList，继而获取DexPathList成员变量dexElement数组，反射执行方法makeElements得到apk的分包dex，然后把之前的Element和新加载的Element合并，反射赋值给dexElement。
  5. 5.0Android系统上，art内建分包支持，不需要mutildex。

**算法**

* 排序相关的（快排，分析不同排序区别，时间复杂度等）
* 字符串、数组相关的（滑动窗口、双指针）
* 链表（反转链表）
* 递归、斐波那契数列（爬楼梯）
* 动态规划

**缺口**

* AQS、CAS原理
* 卡顿优化
* opengl
* 自定义view
* 算法
* sparsearray 为什么使用两个数
  + 适合少量数据，去除hash值的存储空间，没有next指针的占用，节省内存
* ui渲染流
  + **choreograhpher**：关键方法postCallBack，向底层注册请求接收Vsync信号，进行屏幕绘制
  + WindowManager、ViewManager是一个接口，WindowManagerImpl是他们的实现类，定义了addView等方法，addView又由WindowManagerGlobal的addView实现，viewRootImpl在WindowManagerGlobal中实例化。
  + Window是抽象类，PhoneWindow继承了Window，提供了一系列窗口的方法，比如设置内容，背景，标题等
  + setContentView的情况
    - Activity的setContentView最终调用了PhoneWindow的setContentView，之后根据布局文件实例话了DecorView。在Activity生命周期方法onResume方法中，调用了WindowManager的addView方法，将decorView交给viewrootimpl。最终调用setVisibility方法触发invalidate方法，执行viewrootimpl的performTraversals方法，然后通过choreographer的postCallback方法向DisplayEventReceiver注册接收vsync信号，接收到信号执行choreographer#FrameDisplayEventReceiver#onVsync方法，最终执行callback，回到ViewRootImpl的doTraversal方法，触发执行performMeasure，performLayout，performDraw等方法完成绘制
  + addView情况
    - 触发requestLayout方法，重新触发绘制流程
* 整
  + 硬件接收点击信号，通过native代码调用InputEventReveiver的dispatchInputEvent方法，ViewRootImple中WindowInputEventReceiver接收到inputEvent，然后通过层层传递，经过DecorView，PhoneWindow，再传递到Activity，之后执行Activity的onDispatchTouchEvent。
* binder机
* 插件化，replugin原理，优缺点
* shareprefrence跨进程访问

原文作者：[HuanPei Tan](https://tanhuanpei.github.io/)

原文链接：[https://tanhuanpei.github.io/2019/03/25/Android面试知识点总结/](https://tanhuanpei.github.io/2019/03/25/Android%E9%9D%A2%E8%AF%95%E7%9F%A5%E8%AF%86%E7%82%B9%E6%80%BB%E7%BB%93/)