这里有人整理了很全的androd目录，很不错。

https://www.kancloud.cn/s1657292627/android\_ios/621895

JAVA基本题：

1、== 和 equal = 》

2、cloneable接口实现原理，浅拷贝or深拷贝

### 3、[HashMap并发问题](https://www.cnblogs.com/lan-ge/p/10290169.html)，HashTable和hashMap区别、LinkedHashMap

### 4、String、StringBuffer和StringBuilder的区别

### 5、TCP和socket连接：

### Tcp：三次握手，四次挥手

### 6、websocket,socket,http

### 主流框架：glide框架、okHttp，Retrofit，RxJava，EventBus，Picasso，ARrouter，GreenDao

Android基础题

|  |  |
| --- | --- |
| Handler消息处理机制的原理 | 异步处理机制、Looper线程为什么不会卡死  Message.obtain()与直接new的区别 |
| 事件分发机制 | ListView嵌套到ScollView会有什么问题吗  怎么派发下去的，intercepTouchEvent的作用（ScrollView监听是否拖动事件，是的话就拦截）  Activity.onTouchEvent方法什么时候会被回调  事件分发的时候，有什么注意事项吗（ANR） |
| Context对象 | 1. Context有多少种类型实例； 2. 一个anroid进程一般会有多少个context实例； |
| View的绘制流程流程 | measure-> layout->draw  measureSpec作用 ：分配一个View能够使用多大的空间   * **高2位**总共可以组成三种情况，分别为：AT\_MOST、UNSPECIFIED、EXACTLY。 * **低30位**所表示的整型数值，表示着，当前控件可用的最大宽度与高度。再详细解释下所谓**可用的最大宽度和高度** |
| Android的启动模式 | 写一个键盘、分享控件、用什么启动模式好？  多任务栈的返回原理：<Android任务和返回栈简单总结>  Intent.FLAG\_ACTIVITY\_NEW\_TASK  Intent.FLAG\_ACTIVITY\_CLEAR\_TASK | Intent.FLAG\_ACTIVITY\_NEW\_TASK  FLAG\_ACTIVITY\_NEW\_TASK | FLAG\_ACTIVITY\_CLEAR\_TOP  区别 |
| ListView和RecyleView区别 |  |
| 布局组件 | 说一说android比较常用的布局组件。  RelateLayout和LinereLayout比较 |
|  |  |

Android高级

线程：

|  |  |
| --- | --- |
| 基本概念 | Sleep与wait区别、 对象锁、同步、ThreadLocal变量 |
| 常见的并发问题 | 累加的时候、HashMap的add元素的时候； 死锁 |
| 线程中synchronize和vilatie的作用和原理 | 写一个单例模式  Valatie： 可见性（内存模型有关（主存、一级缓存、堆、栈））、原子性（a=0，a++）、有序性（汇编指令不一定）；  （对）  Synchronize：确保线程访问互斥和同步代码，并发问题  原理：锁对象概念，放在堆内存里面，进去这个区域的线程必须持有这个锁才可以进入。否则需要竞争和等待该锁释放之后才可以；  Java对锁有一些优化：偏向锁、轻量级锁（同一个时间不存在多个线程竞争）、自旋级锁、重量级锁 |
| Java内存模型，堆和栈 | 内存模型： 寄存器、本地方法区、方法区（静态方法）、堆内存、栈内存  栈只对线程自己可见，存储的是一些局部变量；  堆存储的是一些公共对象内存，所以线程都可见和可访问；  但为了提高效率，栈会把堆的一些数据缓存到栈里面，这样的话多个线程访问就会有同步问题； |
| 线程池的作用 | 1. 为什么要管理线程？线程池设置多大比较合适？   管理和复用线程，有两个原因：   1. 创建、销毁线程需要时间和内存开销   2）线程多了会给调度cpu是算法带来很大的开销；  Cpu密集：n+1， io密集：2n+1 |
| 多进程 | 1. Binder进程间通信机制：<https://www.jianshu.com/p/ce3c7cc0603e、https://www.jianshu.com/p/4ee3fd07da14>   模型构成： 客户端，服务端、serviceManger， binder驱动  实现IPC方式：   1. 上层：注册服务、获取服务、使用服务。 2. 底层：操作内核空间内存，内存映射的方式去实现跨进程传递数据   ibiner线程池、transact()   1. 常见的IPC方式：信号量/共享内存/socket/消息队列 2. Binder的最大好处： 通过内存映射的方式，跨进程传递的时候只需要拷贝一次。 3. ContentProvider的并发线程是多少：16 4. Android跨进程传递数据方式：序列化的方式( Parcebale)和(匿名共享内存)ashmen 5. [service 和 client的实现消息回调](https://www.baidu.com/link?url=sJ_2W8gVU9DtGar6Zx8T4g_CPyNWYm_vue3oQgJzDhih4PNTR587JzLOJ-nL6YpGwWZdXdJZl4UR2GKsWtXWw_&wd=&eqid=9d5e9b6200001a8f000000065f4cb074)   由于跨进程传输客户端的同一对象会在服务端生成不同的对象！所以service端的回掉方式需要借助RemoteCallbackList类去发送。 |
| ContentProvider | 1. [初始化过程](https://www.jianshu.com/p/1b422676c824) （比Application.onCreate()还早，sdk可以考虑在这里初始化） 2. [理解android:multiProcess属性](https://blog.csdn.net/wangbaochu/article/details/52042926) 3. 操作数据库的同步问题；   它就是通过Binder进程间通信机制和匿名共享内存来实现 |
| 系统适配 | 分屏、画中画、异性屏、挖孔屏 |
| AMS、WMS的职责 | <https://www.jianshu.com/p/47eca41428d6> |

Kotlin和RN

|  |  |
| --- | --- |
| 新特性 |  |
| ?与!!引用区别 |  |
| RN | 1. 生命周期 2. 当你调用setState的时候，发生了什么事 3. props和state相同点和不同点 4. shouldComponentUpdate 应该做什么？ 5. redux状态管理的流程 6. 加载bundle的机制 7. Flex布局 8. component和PureComponent的区别？ 9. Immutable.js原理和作用。（保证每个对象不可变，加快对象比较性能） |
|  |  |
|  |  |

RN面试题：https://blog.csdn.net/qq\_33323251/article/details/80014166

UI框架与设计模式：

|  |  |
| --- | --- |
| mvp、mvc、mvvm | MVC:  M：数据Bean和IModel，数据请求一般写在IModel(impl)  V：xml布局文件  C：Activity，直接持有Model引用，在Activity内直接请求数据，然后再绘制UI  MVP  M:Model还是保持MVC的特性，数据Bean和IModel，数据请求一般写在IModel(impl)  View：  View: Activity，XML，相比MVC，不单单的再将xml布局文件视为View，Activity也不再作为一个Controller而是作为View的一部分而存在，这也是解耦的一个体现，也符合了Android设计Activity这个组件的初衷。  P: 持有View和Modle的引用，根据业务逻辑将二者进行梳理放置  MVVM:  M：数据Bean或IModel，和MVP一样  V：Activity和xml布局  VM：Presenter的进化(ViewModel)，只是数据会利用dataBinding框架的双向绑定特性对数据进行绑定，最直观的结果就是我们设置数据不再是拿到某个具体的View，而是利用框架生成的Binding文件setXXX(Bean)，譬如ActivityMainBinding.setUser(user)  MVVM有两种数据绑定方式，databinding 和 LiveData方式。  缺点：绑定的方式框架会比较死板，不利于一些优化，比如说：view的复用 |
| 组件化框架： | 1. 协同开发 2. 支持多进程 3. 无痕化埋点 4. 接入AI的能力，采集分析 5. 换皮肤插件   系统适配： 分屏、横屏、画中画、深色模式  [组件化的思考](http://blog.spinytech.com/2016/12/28/android_modularization/) |
| 多进程和插件框架 |  |
| RxJAva  基于事件流的链式调用 | （理解了观察者模式，逻辑情绪、调用简单） |
| AOP方式 | 无痕化埋点， |
| 三方组件 | Glide，okhttp，evenbus |
| 自定义sdk | 1. 有什么办法自定义sdk做一些初始化工作 |

Andorid性能 （卡顿流畅性、内存问题、耗电）

|  |  |  |  |
| --- | --- | --- | --- |
| 卡顿 | 内存抖动 | OnDraw方法， 避免频繁new，传输载体类,Log | [对象池](http://blog.spinytech.com/2017/01/10/concurrent_object_pool/)  onDraw，log打印日志，上报   1. ​字符串的拼接优先考虑StringBuilder和StringBuffer |
| UI线程避免耗时操作 | Io操作，解析操作 |  |
| 页面布局复杂 | 过度渲染、重复渲染、布局问题   1. 层级要少：减少过度绘制； 2. 硬件加速； | 1. UI布局问题；层级不太深；L布局不要weight，多个L使用一个R方式； |
| ANR问题 | 避免在事前派发过程中， | 出现ANR时，adb pull data/anr/tarces.txt 结合log分析 |
| 内存 | | 1. 内存泄漏原因：   GC只会会清理那些引用链不可达的对象（对象根本不会再调用），但是有些对象是可以确定不会再调用了，但引用链是可达的，gc不会释放。  **如果长生命周期的对象持有短生命周期的引用，就很可能会出现内存泄露**   1. 内存溢出 ： 多进程，webview，RN页面 2. 内存抖动： |  |
| [耗电](App耗电原因分析) | | 1. 引起cpu和gpu大量计算：布局过度渲染、动画、频繁交互、json解析 2. 频繁网络请求、网络切换 3. 定时任务唤醒； 4. 非必要，不要常驻service 5. 传感器和设备问题 | batteryReport  cpu使用率  过度渲染的情况 |
| Apk瘦身 | |  |  |
| 冷、暖、热启动 | |  | 不要只考虑application.onCreate（），注意ContentProvider的onCreate初始化 |

主要研究方向：移动端上一些AI能力在应用层面的落地。

数据埋点采集框架：

1. 没有专门的一个采集进程去处理
2. 埋点和采集是不冲突的，采集是埋点的基础。
3. 线程和进程是什么区别；

共享相同的内存空间；

1. 生成者和消费者模式；
2. ？

Android内存管理：

1内存管理机制（java回收机制）

1）一开始会给每一个进程分配一个“够用”的app，随着应用的运行，会再分配额外的内存；但是都是有限度，如果超过这个内存上限度，会被认为内存溢出而杀死记这个进程；

android更大的让更多进程留在内存里，就是退出app，也不会马上释放内存；而是置为空进程，为下次启动做准备，减少启动时间；

2）、回收；当程序在运行过程当中，遇到内存不够用时，java会自动进行gc回收操作； 会计算哪些对象是可以回收，然后回收释放内存。然后根跟进程可见属性，杀掉一些后台进程。

1. 内存优化：
2. 静态内部类代替匿名内部类；
3. 使用弱引用去持有外部的类；
4. 避免频繁gc： onDraw， Log
5. IO资源要及时释放；
6. 使用cannaryLeak去检测内存泄漏；

ReactNative

1. RN运行原理；
2. Flex布局：

Flex-direction（排列方向）、justifyContent(主轴的排列方式)，alignItems(次轴的排列方式)

1. 组件的生命周期：

1)挂载DOM阶段： componentWillMount –> render() compoonentDidMount()

2)更新阶段：

compoentWIllUpdate –>render->compentDidUpdate()

1. ListView和FlatList性能对比；
2. RN bundle与热更新、按需加载；
3. JAVA与JS同通信；

基本原理：Java层在map注册里面注册方法，注册之后就可以在js里面调用；

跨进程框架：

1. Ibinder通信原理

Servcie端通过Binder驱动把接口方法注册到ServiceManager（ServiceManager是一个系统服务，是一个常驻进程），然后client通过Binder驱动向ServiceManager端请查询相对方方法接口和接口；当客户端调用改方法时，代理对象会通过驱动去调用服务器的方法；服务器的方法被执行后，会通过ServiceManager返回客户端；

Binder驱动： AIDL描述去创建和维护一个共同Binder引用，然后再通过这个binder引用去调用各自进程的java层的binder方法；

AIDL：就是接口标志符，

1. 都有哪些跨进程方法

ContentProvder，共享文件，管道，socket

以技术服人： 帮助别人、公共解决问题的经历；

框架去制约，提高开发效率、控制风险； 每个人的能力有限，

指出错误 ： 不要期望他能马上改过来，就是让他知道有pk成本。后面就会收敛起来。场景：手写模块放在公共ui模块。

不可或缺人才：

做分享：

认知：

1. 一个流程走到底，造成很多不可或缺人才；
2. 性能排查方面，让我想到了很多；

分析题：

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

Paint paint = new Paint();

paint.setColor(Color.BLUE);

paint.setStyle(Style.FILL);

canvas.drawRect(new Rect(0, 0, getWidth(), getHeight()), paint);

paint.setColor(Color.RED);

paint.setStyle(Style.STROKE);

canvas.drawRect(new Rect(10, 10, 100, 30), paint);

paint.setColor(Color.GREEN);

canvas.drawText("Hello", 10, 50, paint);

Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.icon);

canvas.drawBitmap(bitmap, 10, 60, paint);

}