### 基础网络、新技术

1. 框架（图片 网络 实现）

－Volly、HttpClient、HttpUrlClient、Okhttp 区别

答：HttpClient 繁琐，且已被抛弃

HttpUrlConnection 2.2以下有重大bug，可读的Inputstream调用close()方法，会导致连接池失败，无法复用。Android 4.0中增加了缓存机制.

Okhttp 实现了连接池、gziping 、缓存，使用Okio实现数据访问存储。

Volley是一个简化网络任务的库。他负责处理请求，加载，缓存，线程，同步等问题。它可以处理JSON，图片，缓存，文本源，支持一定程度的自定义。

（https://www.jianshu.com/p/1b088dd37a2d）

－http https 架构

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。



－图片框架库 使用 比较

Fresco

Universal-Image-Loader

Picasso

－使用：<https://www.jianshu.com/p/c68a3b9ca07a>

Picasso.with(context)

.load(url)

.placeholder(R.drawable.default)

.error(R.drawable.errorimage)

.noFade()

.onlyScaleDown()

.resize(300,400)

.resizeDimen(R.dimen.image\_width, R.dimen.image\_height)

.centerCrop()

.fit()

.rotate(180, 200, 100)

.transform(new BlueTransformation(this))

.priority(Picasso.Priority.HIGH)

.tag("photoTag")

.memoryPolicy(MemoryPolicy.NO\_CACHE, MemoryPolicy.NO\_STORE)

.networkPolicy(NetworkPolicy.NO\_CACHE)

.networkPolicy(NetworkPolicy.OFFLINE)

.setIndicatorsEnabled(true)

.setLoggingEnable(true)

.into(imageview)

.into(new Target(){

public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {

}

@Override

public void onBitmapFailed(Drawable errorDrawable) {

}

@Override

public void onPrepareLoad(Drawable placeHolderDrawable) {

}

})

Picasso.Builder builder = new Picasso.Buidler(this)

builder.download(new CustomerLoader())

builder.memoryCache(new Lurcache(5\*1024));

builder.executor(Executors.newFidxedThreadPool(8));

Picasso pi = builder.buil();

Picasso.setSingletonInstance(pi);

**内存缓存是使用的LRU 策略的缓存实现，它的大小是内存大小的15%,可以自定义它的大小。**

**磁盘缓存是磁盘容量的2%但是不超过50M,不少于5M。**

**处理一个请求的时候，按照这个顺讯检查：memory->disk->network。**

Glide

——

<https://www.jianshu.com/p/93198a80bda7>

－长连 短连 socket/http协议 参数 ／Tcp三次握手具体实现

答：

dns解析

－异步 Future

答：

* OkHttp缓存机制

答：服务端缓存机制（了解）：cache-control、Etag、If-None-Match

Etag对应请求的资源在服务器中的唯一标识（具体规则由服务器决定），比如一张图片，它在服务器中的标识为ETag: W/”ACXbWXd1n0CGMtAd65PcoA==”。

If-None-Match 如果浏览器在Cache-Control:max-age=60设置的时间超时后，发现消息头中还设置了Etag值。然后，浏览器会再次向服务器请求数据并添加In-None-Match消息头，它的值就是之前Etag值。服务器通过Etag来定位资源文件，根据它是否更新的情况给浏览器返回200或者是304。

Etag机制比Last-Modified精确度更高，如果两者同时设置的话，Etag优先级更高。

OKHttp cache Response有两种类型：CacheResponse、NetworkResponse

手动添加拦截器，设置超时时间



<http://blog.csdn.net/briblue/article/details/52920531>

<https://www.jianshu.com/p/797d2d442d27>

大概流程：

<https://www.jianshu.com/p/00d281c226f6>

2、RxJava 线程切换原理 lift 操作符

lift原理：创建新的Subscriber，Obserable

线程切换的原理

subscribeOn() 设置被观察者的所在的线程，只有第一次有效

observeOn()改变调用它之后代码的线程

简单说:最原始的subscribeOn()—调用了—-create(new OperatorSubscribeOn<T>(this, scheduler))—-创建一个代理被观察者—->OperatorSubscribeOn()中实现了call()方法—->call()方法中调用了NewThreadWorker.schedule(Action0)—-Action0被包装称一个RUnnable对象,然后schedule()方法内部使用了线程池,创建一个新的线程,并将包装的Runnable对象传递进去,这样就实现了线程的切换

步骤:

A、原始被观察者调用subscribeOn()方法准备切换线程,(这时候还没切换呢.)产生一个代理被观察者

B、原始订阅者订阅代理被观察者(明面代码上你能看得到的)

C、代理被观察者的onSubscribe.call()方法执行,提供了一个Runnable对象,也就是线程已经被切换了

D、新线程中产生一个新的代理观察者,代理观察者订阅原始被观察者(接下来的动作也都是在新线程中执行)

E、原始被观察者发射数据,这个动作已经是在新线程中执行了

F、代理观察者收到数据,再将数据转发给原始观察者

subscribeOn():多次调用只有第一次的有效.

observeOn():多次指定下游的线程是可以的, 也就是说每调用一次 , 下游的线程就会切换一次

map:转换类型

flatmap:将一个发射数据的Observable变换为多个Observables，然后将它们发射的数据合并后放进一个单独的Observable，合并使用merge操作，故无序

comatmap：与flatmap类似，有序发送

zip: 将多个Observable发送的事件结合到一起，发送属于与最少的数据量相同，其余舍弃。

**merge：**

**filter:**

**interval：间隔时间发送事件**

**sample：取样**

**delay：**

Flowable: 默认缓存128，

观察者Subscriber使用subscription.request(1000)来控制发送, FlowableEmitter.requested()获得当前外部请求数，上游next一下，-1，下游request(n)一下，+n

Backpressure: 目标，控流。几种策略-----

BackpressureStrategy. BUFFER：不限制容量

BackpressureStrategy.DROP：把存不下的事件丢弃

BackpressureStrategy.LATEST：只保留最新的事件

BackpressureStrategy.MISSING

BackpressureStrategy.ERROR

3、设计模式 复习

设计模式 装饰者模式、单例模式、适配器模式、工厂模式

代理模式和装饰模式区别

团长的PPT android中的应用

https://www.jianshu.com/p/fb8d21978e38

4、抽象类和接口 设计的区别

5、MVC、MVVM、MVP 三种模式 复习

ViewModel层的主要责任是表现逻辑和状态，即ViewModel层是连接View层和Model层的。Model层的一些业务逻辑的状态是需要通过ViewModel层暴露给View层来反映给软件使用者的。

6、kotlin 编程

学习视频，了解

<https://www.jianshu.com/p/052fdf84b100>

<https://www.kotlincn.net/docs/reference/android-overview.html>

扩展 静态扩展、扩展的冲突（静态解析，申明类型确定调用）

还有kotlin里lambda表达式跟java的区别

7、ReactNative 学习

搭建环境，再深入一点 —— Note.js

RN中文网<https://reactnative.cn/docs/0.51/getting-started.html#content>

参考简单项目：<https://www.jianshu.com/p/6d0331a39be0>

8、通信加密 对称 非对称 sha1

9、图片在内存的大小控制，影响因素

图片下载控件 逻辑

图片的内存影响

RGB\_8888 每个图片像素占据4个字节

RGB\_565每个图片像素占据2个字节

Android中有四种，分别是：

ALPHA\_8：每个像素占用1byte内存

ARGB\_4444:每个像素占用2byte内存

ARGB\_8888:每个像素占用4byte内存

RGB\_565:每个像素占用2byte内存

Picasso图片加载内存优化方案：

<https://www.jianshu.com/p/6b746c904a49>

总结：1、图片裁剪；2、放弃大图memory Cache；3、DetachView时，去除imageView中图片的引用；4、在新进程查看图片；5、页面滑动停止加载图片；6、对于不透明的图片可以使用RGB\_565来优化内存；7、reference

10、Retrofit

Retrofit上次人家问我原理，这次问更细，retrofit.xx返回给接口，这个实际上返回的是什么。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Java部分

1、各种排序算法

算法的复杂度 计算

快速排序

多路归并排序

冒泡

shell

反转链表

二叉树——两个字节点的最低父节点

——生成二叉树

——前序、中序、后序 遍历

——反转二叉树

字符串 连接函数，考虑位置信息和图片的富文本信息

regular expression match

permutation 1和2

视频推荐系统设计

红黑树

左旋 右旋

增加 删除 节点

堆排序

算法 异或

2、多线程 面试回答整理

同步、异步 (同步方法 同步代码块 同步锁)

阻塞、非阻塞

实现一个阻塞模型

跨进程通信—— contentProvider

**join：**在main线程中A.join()，意味main进程让出CPU，执行完A，才回来执行main，相当于变成串行。Join（0）=join(),join(10)=让出10s，之后恢复并行【本质上是调用了当前线程的wait，执行完在notify】

wait：同步

Thread.yield(): 让出

Thread.sleep():

sleep()方法是Thread类里面的，主要的意义就是让当前线程停止执行，让出cpu给其他的线程，但是不会释放对象锁资源以及监控的状态，当指定的时间到了之后又会自动恢复运行状态。

wait()方法是Object类里面的，主要的意义就是让线程放弃当前的对象的锁，进入等待此对象的等待锁定池，只有针对此对象调动notify方法后本线程才能够进入对象锁定池准备获取对象锁进入运行状态。

对方法加锁

synchronized(this)

3、GC垃圾回收机制 内存回收几种算法 内存分代 算法复习

Devid ART 区别

java虚拟机原理

Dalvik虚拟机

4、Java容器

java中的容器主要有List，Set，Map三种。了解这些容器的基本属性以及底层的实现，对正确高效的使用他们很有效。

List下有ArrayList与LinkedeList。

ArrayList 底层类似数组。

LinkedList 底层类似与链表。

Set下有HashSet，TreeSet，LinkedeHashSet。

HashSet重复的值存储一次，查询快。

TreeSet重复的值存储一次，存储为升序排列。

LinkedHashSet重复的值存储一次，存储为放入顺序，查询快。

Map下有HashMap,TreeMap,LinkedHashMap.

HashMap重复键，存储后一个值，查询快。

TreeMap重复键，存储后一个值，存储为升序排列。

LinkedHashMap重复键，存储后一个值，存储为放入顺序，查询快

Hashmap的实现

5、浅拷贝：copy的链接，clone()

深拷贝:实现clonable接口，彻底的深拷贝

6、线程池

Executors.newFixedThreadPool(XX) 固定大小X个

newSingleThreadExecutor()

newCachedThreadPool

newScheduledThreadPool

7、内部类

**静态内部类：**不可以访问外部类的非静态成员，不能直接访问外部类的非静态成员，只能通过一个对象引用来间接使用。

**非静态内部类:** 非静态内部类可以访问外部类的所有成员（包括外部类中的private成员）——不能定义各种静态的成员（包括静态变量、静态方法、静态代码块和静态内部类），原因在静态变量和静态方法都只需要通过类名就能访问，不必通过任何实例化对象；而普通内部类的初始化要利用外部类的实例化对象，这明显违背了static的设计初衷。

**匿名内部类：**

**局部内部类：**

8、for

Java的for 循环，两种底层的实现，速率，效率比较

9、Java虚拟机

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Android部分

Android 高级面试题及答案<https://www.jianshu.com/p/3864c11bfb46>

1、APK架构、AAPT 打包过程、包内文件目录结构



2、热修复 hotfix 原理 插桩法 几种热修复方案

andfix

美团实现Robust

3、Gradle／Android工程的编译/实现拆包／分库／

dex拆分 细节 MultiDex 优化

multidex 异步加载

multiDex突破65535

Dex

4、泰坦 webview 了解逻辑，面试可问的问题

https://www.jianshu.com/p/3c94ae673e2a

**5、android View ViewGroup 消息分发**

viewGroup的考察(touch事件分发机制) <https://wiki.sankuai.com/pages/viewpage.action?pageId=566344792>

ViewGroup 首先调用 onInterceptTouchEvent

View/ViewGroup 正常流程-> dispatchTouchEvent -> onTouch

对于View（注意！ViewGroup也是View）而言，如果设置了onTouchListener，那么OnTouchListener方法中的onTouch方法会被回调。onTouch方法返回true，则onTouchEvent方法不会被调用（onClick事件是在onTouchEvent中调用）所以三者优先级是onTouch->onTouchEvent->onClick

View 的onTouchEvent 方法默认都会消费掉事件（返回true），除非它是不可点击的（clickable和longClickable同时为false），View的longClickable默认为false，clickable需要区分情况，如Button的clickable默认为true，而TextView的clickable默认为false。

View的描绘原理：<https://wiki.sankuai.com/pages/viewpage.action?pageId=569127518>

onMeasure()和onLayout()方法

6、android 启动 app 新进程 过程

7、android动态化配置框架

8、android组件化实现方案

9、android项目解耦合具体方案

－组件化

－1.app模块插件化，每一个功能模块都类似于一个项目可以独立开发、运行、测试。这些模块UI跳转之间的耦合可以使用URL来进行跳转。

－2.项目本身使用更加合理的框架，例如MVP，不能生搬硬套，更具项目情况合理使用。

- 3. 使用RxAndroid，这样观察者模式

https://www.jianshu.com/p/45cb536be2f4

10、视频播放器 研究，实现

11、券商相关：分时K线图实现

12、Parcelable 详细

13、Android 线程间通信有哪几种方式

1）共享变量（内存）

2）管道

3）handle机制

runOnUiThread(Runnable)

view.post(Runnable)

——多进程：AIDL，内存共享，广播

14、Android View的创建过程，Activity

ActivityManagerService(AMS)是Android中最核心的服务

15、handler

Runnable 和Message 是消息的2种载体,所有的消息都会放入MessageQueue里面,MessageQueue并不是一个真正的队列，而是链表,Looper就是循环在某件事情.Looper不断的从MessageQueue从取出数据，然后交给handler来处理。

当你创建一个新的handler的时候，他会与这个创建它的线程绑定。

对于一个Thread来说MessageQueue,和Looper只有一个。

prepare才是looper创建以及和thread绑定的地方。

looper.loop()方法是整个looper机制启动的地方。

Handler的消息机制，如果当前消息没有了，looper会怎么样？

16、AOP

AOP AspactJ：<https://www.jianshu.com/p/c202853059b4>

AOP 切片框架

17、

Android textview.set（xx），从1执行到100，最后结果是什么，就在主线程中

18、

Xml中的布局怎么转变成Android的view

19、Dalvik 虚拟机 和 ART 虚拟机

<https://blog.csdn.net/jason0539/article/details/50440669>

Dalvik虚拟机 dex->odex->

AOT 取代 JIT

JIT：Just In Time，及时编译技术。JIT会在运行时分析应用程序的代码，识别哪些方法可以归类为热方法，这些方法会被JIT编译器编译成对应的汇编代码，然后存储到代码缓存中，以后调用这些代码时就不用解释执行了，可以直接使用代码缓存中已编译好的汇编代码，能显著提升应用程序的执行效率。

AOT：预编译技术，编译时直接将源代码编译成机器码。安装时间长，字节码编译为机器码耗时长，需要更大的空间。

**Dalvik的垃圾回收机制**：非并发，频繁GC，容易卡顿，采用“标记清除”，内存碎片化，对象分配内存寻址较慢。

ART垃圾回收机制：部分并发。

ART在内存分配上做了优化，比如它开辟了一块名为Large Object Space的内存区域，专门用来存放大对象，同时还引入了一个名为moving collector的技术，专门用来将gc后不连续的物理内存块对其，解决了Dalvik上内存碎片化严重的问题。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

———————其他 工具

1、python 复习

2、正则表达式

3、Lambda表达式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

————————DP业务相关：

1. crash上报、符号化，解析 Python shell脚本 工具化
2. zxing二维码
3. push 保活，定位，手机Donzon状态下如何保活

保活方案调研<https://wiki.sankuai.com/pages/viewpage.action?pageId=775216685>

**Doze：**状态下，进入的状态比较苛刻，1小时的等待期（屏幕半小时后进行大幅度运动监测，接下来半小时无动作才进入doze），AlarmManager6.0新增了两个方法setAndAllowWhileIdle()和setExactAndAllowWhileIdle()在Doze状态下执行定时任务，但是每个App每9min只能执行一次。

**App Standby：**系统通过App Standby机制可以把用户没有使用的app置为空闲状态，当然前提是app没有命中以下条件：

用户明确的启动了该app

app有前台进程（包括前台的activity、前台service，或者正在被其他activity及前台service使用）

app当前没有在锁屏界面或者通知栏有通知

当设备连接到电源时，系统会把所有处于standby的app恢复到正常运行状态，如果设备长期处于空闲状态，系统会每天允许standby的app联网一次。

小结：在App Standby状态下，App很难得到运行，因此Push受到的影响会更大，当用户重新打开App或者充电时就会恢复。

使用GCM，但在国内环境下是不现实的；

Android提供了白名单机制，使app能在Doze和Standby时仍能够访问网络

这种方式类似与请求定位等权限，但是对用户不是特别友好，可以考虑。

其实Doze模式对Push的影响不大，因为很难进入Doze模式，且可以通过setAlarmClock()继续使用定时器。主要是Standby模式影响比较大，如果app长时间没被使用，就无法自启动运行。目前的解决方案主要是通过官方的REQUEST\_IGNORE\_BATTERY\_OPTIMIZATIONS权限让用户加入白名单，其他暂时没有有效的解决方案。

混淆

多渠道打包管理

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

高性能的注意项

0、注意内存泄漏（比如数据库的Corsor对象关闭，content内存泄漏，使用工具Leaks检测）

Thread会造成无法被GC吗？必须手动？

Handler的惯用写法，内存泄漏

Activity中使用非静态内部类导致的，如果其实例的持有对象的生命周期大于其外部类对象，那么就有可能导致内存泄露。

1、android 内存优化 onTrimMemory，主要是提示开发者在内存不足的时候，主动处理资源，释放内存，从而避免Android系统杀死，毕竟热启比冷启耀快很多

2、避免使用Enum，使用SparseArray替代HashMap

3、避免创建不必要的对象，对生命周期较短的临时变量，想办法规避掉每次都要创建，减少GC的次数，也是减少内存抖动，减少了Android系统的卡顿

这样GC回收被强制调用机会就会更少，留给Android系统进行UI渲染或者音频加载的时间就会更多，从而避免了卡顿现象

4、慎用Service，下载使用IntentService

5、将应用分为多个进程，减少被系统杀死的概率（注意后台进程不要与前台UI耦合）

6、尽量提供与设备分辨率匹配的bitmap进入内存，否则，分辨率高的，会被压缩，当然，本身分辨率高的本身占用空间就大

7、lint检查，去除不必要的资源，减小APK大小，使用Proguard混淆（好处？？补充下 ）

8、软引用：内存足＝强，不足，回收

弱引用：GC时即回收，生命周期更短

9、尽量避免使用Enum。枚举相对于静态常量来说，需要两倍甚至更多的内存。

10、布局层级 过度绘制

11、启动 使用异步加载

12、谨慎使用面向对象，避免不必要的额外代码，耗费了资源

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lint 接入、FindBugs、CheckStyle，

打点架构实现

整体架构

动态代理 Proxy InvocationHandler

静态模型，模块化实现， 静态化模型 babel实现方式

material design

管道流

BaseDetectHelper.java

splash页面实现 NovaMainApplication 阅读 皮肤实现

app包大小控制方案 包体大小监控，缩小

查看应用内存 adb shell dmpsms package name

https://mp.weixin.qq.com/s/fz8wcE13N96SQ1MwQL-tNQ