**二组模拟面试汇总**

**一、**

**·Android通信机制**

第一种：使用handler来进行线程之间通信

第二种：AsyncTask--线程之间通信

第三种：intent--(Activity和Activity之间跳转)

第四种：BroadCastReciver广播的发送与接收--组件之间通信

第五种：Notifation通知栏信息

第六种：socket

第七种：Notification

第八种：EventBus--发布/订阅事件总线

**二、**

**·EventBus**

**三、**

**·下单支付流程**

**四、**

**·泛型的理解及使用**

泛型是[程序设计语言](https://baike.so.com/doc/5797287-6010082.html" \t "https://baike.so.com/doc/_blank)的一种特性。允许程序员在强类型程序设计语言中编写代码时定义一些可变部分，那些部分在使用前必须作出指明。[泛型类](https://baike.so.com/doc/1782661-1885184.html" \t "https://baike.so.com/doc/_blank)是引用类型，是堆对象，主要是引入了类型参数这个概念。

****泛型之前****

在Java增加泛型类型之前，通用程序的设计就是利用***继承***实现的，例如，ArrayList类只维护一个Object引用的数组，Object为所有类***基类***。当我们获取一个值的时候，必须进行强制类型转换。

所以，从Java 5开始，ArrayList在使用时可以加上一个****类型参数（type parameter），这个类型参数用来指明ArrayList中的元素类型。****

所谓**泛型类**（generic class）就是具有一个或多个类型参数的类。

所谓**泛型方法**，就是带有类型参数的方法，它既可以定义在泛型类中，也可以定义在普通类中。

**五、**

**·注解及使用**

# ButterKnife 简单使用

****（1）ButterKnife是什么？****

在开发过程中，我们总是会写大量的findViewById和点击事件，像初始view、设置view监听这样简单而重复的操作让人觉得特别麻烦，当然不会偷懒的程序员不是好程序员，自然也出现了相应的解决方案--依赖注入。而ButterKnife则是依赖注入中相对简单易懂的很不错的开源框架，（其实ButterKnife也不算严格意义上的依赖注入，后面文章中会做分析）。但ButterKnife作为JakeWharton大神写的注解框架被广泛应用于android开发中，自然也有它的过人之处。下面对它的使用过程进行描述。

****（2）它的具体优势：****

1.强大的View绑定和Click事件处理功能，简化代码，提升开发效率

2.方便的处理Adapter里的ViewHolder绑定问题

3.运行时不会影响APP效率，使用配置方便

4.代码清晰，可读性强

****（3）ButterKnife使用中有哪些注意的点呢？****

①：Activity ButterKnife.bind(this);必须在setContentView();之后，且父类bind绑定后，子类不需要再bind

②：Fragment ButterKnife.bind(this, mRootView);

③：属性布局不能用private or static 修饰，否则会报错

④：setContentView()不能通过注解实现。

**六、**

**·缓存策略**

# Android 缓存策略LruCache和DiskLruCache学习

LruCache和DiskLruCache是采用了LRU(Least Recently Used)近期最少使用算法的两种缓存。LruCache内存缓存，DiskLruCache存储设备缓存；

**LruCache**是线程安全的；****

**七、**

**·wait, notify , notifyAll 是Thread 还是Object**

sleep来自Thread类，sleep方法没有释放锁，sleep不出让系统资源；sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到只能调用interrupt()强行打断。

Sleep需要捕获异常,而wait不需要

# wait()、notify()和notifyAll()是 **Object**类 中的方法

1）wait()、notify()和notifyAll()方法是本地方法，并且为final方法，无法被重写。

2）调用某个对象的wait()方法能让当前线程阻塞，并且当前线程必须拥有此对象的monitor（即锁）

3）调用某个对象的notify()方法能够唤醒一个正在等待这个对象的monitor的线程，如果有多个线程都在等待这个对象的monitor，则只能唤醒其中一个线程；

4）调用notifyAll()方法能够唤醒所有正在等待这个对象的monitor的线程；

\*wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用；

**八、**

**·Picaso和 Glide 的区别**

****磁盘缓存****

Picasso和Glide在磁盘缓存策略上有很大的不同。Picasso缓存的是全尺寸的，而Glide缓存的是跟ImageView尺寸相同的。

****内存开销****

原因在于Picasso是加载了全尺寸的图片到内存，然后让GPU来实时重绘大小。而Glide加载的大小和ImageView的大小是一致的，因此更小。

****Image质量的细节****

Glide加载的图片没有Picasso那么平滑，

Glide可以加在GIF动态图，而Picasso不能。

因为Glide和Activity/Fragment的生命周期是一致的，因此gif的动画也会自动的随着Activity/Fragment的状态暂停、重放。Glide 的缓存在gif这里也是一样，调整大小然后缓存。

****加载速度****

Glide的这种方式优点是加载显示非常快，需要更大的空间来缓存。而Picasso的方式则因为需要在显示之前重新调整大小而导致一些延迟，

****库的大小****

Picasso (v2.5.1)的大小约118kb，而Glide (v3.5.2)的大小约430kb。 Picasso和Glide的方法个数分别是840和2678个。

****总结****

Glide和Picasso都是非常完美的库。Glide加载图像以及磁盘缓存的方式都要优于Picasso，速度更快，并且Glide更有利于减少OutOfMemoryError的发生，GIF动画是Glide的杀手锏。不过Picasso的图片质量更高。

**一、**

**·擅长的项目类型**

**二、**

**·Material Design新控件**

Material Design（材质设计）是Google在2014年I/O大会上发布的一种新的设计规范，只支持5.0及其以上的设备。

**三、**

**·动画分类**

**,在国内的很多网站上动画都分为两种:帧动画和补间动画,但是在安卓的官网上在3.0,又加入一种属性动画.也就是现在有3种动画.**

**1: Property Animation(属性动画别加入在3.0)**

**2: View Animation (补间动画分为:透明,旋转,位移,缩放)**

**注意:补间动画是假动画,比如一个位移从A移动到B,如果给他设置监听事件,他的点击区域永远都在A,但是属性动画,如果给他设置点击事件,同样从A移动到B,他的点击区域就在B.**

**3: Drawable Animation(帧动画**帧动画类似于放电影，通过播放已经排列放好的图片来实现。**)**

   两种，一种是补间动画，还有一种是帧动画，帧动画类似于放电影，通过播放已经排列放好的图片来实现。

补间动画的实现定义开始和结束，中间由系统计算得出效果有透明，位移，放大缩小等等。

自定义录音或者进度条过程，会应用到帧动画，补间动画，一般用于应用的欢迎界面。

(菜单弹出，例如赞和评论，是个动画过程。)

**四、**

**·Activity启动模式**

**五、**

**·为什么要使用Retrofit+RxJava**

**六、**

**·多线程的实现方式**

**一：**

**·擅长的项目类型“”**

**二：**

**·购物车数据同步的实现**

**三：**

**·Okhttp切换成Retrofit+rxJava的原因**

1、**Okhttp**优点

支持SPDY, 可以合并多个到同一个主机的请，使用连接池技术减少请求的延迟(如果SPDY是可用的话) ，

使用GZIP压缩减少传输的数据量，缓存响应避免重复的网络请求、拦截器等等。

2、**Okhttp**缺点

第一缺点是消息回来需要切到主线程，主线程要自己去写，第二传入调用比较复杂。

**\*\*\***

Retrofit+RxJava的优点很明显：封装了OkHttp,使用了注解，链式调用，支持Gson解析,可以实现线程之间的快速切换；处理数据简洁易懂，易于进行元素间的变换；可以简单处理大量的嵌套异步回调等。

进行了封装，优点是调用方便，代码量少，链式结构清晰。

Rxjava**简洁，异步，**解决蜜汁锁紧，嵌套循环；

**四：**

**·Retrofit1.0和Retrofit2.0的区别**

Retrofit1中的setConverter，2.0换以addConverterFactory，用于支持Gson转换。

# **1 背压处理**

1.0，关于背压最大的遗憾是集中在Obserable这个类中处理，导致有的Observable支持背压，有的不支持。

为了解决这种缺憾，新的版本把支持背压和不支持背压的Observable 区分开来。

Rxjava2.0，出现了两种观察者模式：

Observable(被观察者)/Observer（观察者）   
Flowable(被观察者)/Subscriber(观察者)

RxJava2.X中， Observeable用于订阅Observer ，是不支持背压的，而 Flowable用于订阅Subscriber ，是支持背压(Backpressure)的。

# **2 操作符名称变动**

操作符相关的变动基本都只是包名或类名的改动。大家经常用到的就是Action和Funtion。

**五：**

**·什么是背压**

RxJava背压（Backpressure）机制

在RxJava中，可以通过对Observable连续调用多个Operator组成一个调用链，其中数据从上游向下游传递。当上游发送数据的速度大于下游处理数据的速度时，就需要进行Flow Control了。如果不进行Flow Control，就会抛出MissingBackpressureException异常。

这就像小学做的那道数学题：一个水池，有一个进水管和一个出水管。如果进水管水流更大，过一段时间水池就会满（溢出）。这就是没有Flow Control导致的结果。

再举个例子，在 RxJava1.x 中的 observeOn， 因为是切换了消费者的线程，因此内部实现用队列存储事件。在 Android 中默认的 buffersize 大小是16，因此当消费比生产慢时， 队列中的数目积累到超过16个，就会抛出MissingBackpressureException。

如果你想了解更多关于背压的知识，请参考：

[http://blog.csdn.net/jdsjlzx/article/details/52717636](http://blog.csdn.net/jdsjlzx/article/details/52717636" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
[http://www.jianshu.com/p/2c4799fa91a4](http://www.jianshu.com/p/2c4799fa91a4" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)

**六：**

**·RxJava使用完对象之后怎么处理**

**（内存泄漏相关问题）**

**软引用**

**@软引用和弱引用**。

如果一个对象只具有软引用，那么如果内存空间足够，垃圾回收器就不会回收它；如果内存 空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。软引用可以和一个引用队 列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。

如果一个对象只具有弱引用，那么在垃圾回收器线程扫描的过程中，一旦发现了只具有弱引 用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。弱 引用也可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联 的引用队列中。

弱引用与软引用的根本区别在于：只具有弱引用的对象拥有更短暂的生命周期，可能随时被回收。而只具有软引用的对象只有当内存不够的时候才被回收，在内存足够的时候，通常不被回收。

**七：**

**·RxJava操作符及使用**

[1. RxJava使用介绍](http://blog.csdn.net/jdsjlzx/article/details/51485778" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/360" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
 [2. RxJava操作符](http://www.stay4it.com/course/27" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Creating Observables(Observable的创建操作符)](http://blog.csdn.net/jdsjlzx/article/details/51485861" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/359" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Transforming Observables(Observable的转换操作符)](http://blog.csdn.net/jdsjlzx/article/details/51493772" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/369" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Filtering Observables(Observable的过滤操作符)](http://blog.csdn.net/jdsjlzx/article/details/51489061" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/371" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Combining Observables(Observable的组合操作符)](http://blog.csdn.net/jdsjlzx/article/details/52415615" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/370" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Error Handling Operators(Observable的错误处理操作符)](http://blog.csdn.net/jdsjlzx/article/details/52448173" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/372" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Observable Utility Operators(Observable的辅助性操作符)](http://blog.csdn.net/jdsjlzx/article/details/54768360" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/364" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Conditional and Boolean Operators(Observable的条件和布尔操作符)](http://blog.csdn.net/jdsjlzx/article/details/54767751" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/374" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• Mathematical and Aggregate Operators(Observable数学运算及聚合操作符)](http://blog.csdn.net/jdsjlzx/article/details/51489793" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/368" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
  [• 其他如observable.toList()、observable.connect()、observable.publish()等等；](http://blog.csdn.net/jdsjlzx/article/details/53365839" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/366" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
[3. RxJava Observer与Subcriber的关系](http://blog.csdn.net/jdsjlzx/article/details/51534504" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/367" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
[4. RxJava线程控制（Scheduler）](http://blog.csdn.net/jdsjlzx/article/details/51685769" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/365" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)   
[5. RxJava 并发之数据流发射太快如何办（背压（Backpressure））](http://blog.csdn.net/jdsjlzx/article/details/51868640" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank) [【视频教程】](http://www.stay4it.com/course/27/learn" \l "lesson/373" \t "https://blog.csdn.net/jdsjlzx/article/details/_blank)

**八：**

**·简述观察者模式**

**（Java中类是被观察者，接口是观察者）**

**观察者设计模式(自定义控件的时候设置点击事件,)**

       所谓观察者模式，举个例子现在许多购房者都密切观察者房价的变化，当房价变化时，所有购房者都能观察到，以上的购房者属于观察者，这便是观察者模式。

java中可以借助Observable类和Observer接口轻松实现以上功能。当然此种模式的实现也不仅仅局限于采用这两个类。

观察者模式本质上就是设置监听:通过监听来观察发生变化,通过回调返回,

首先需要定义一个接口,接口里面需要定义一个方法,方法的参数就是需要返回的参数,

其次需要定义一个监听设置的方法

最后在需要监听的地方设置监听

在需要调用的的地方首先new出对象,调用监听的方法,设置内部实现类,实现方法,从而达到监听的目的.

**九：**

**·接口回调如何进行+使用场景**

**（MVP,RecyclerView的条目点击事件,Fragment通信）**

**十：**

**·String | StringBuffer | StringBuilder 区别**

JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，

即包含多个字符的字符数据。这个String类提供了数值不可改变的字符串。而这个

StringBuffer类提供的字符串进行修改。

StringBuilder与StringBuffer的区别 。

StringBuffer

1，一个类似于 String 的字符串缓冲区，对它的修改不会像String那样重创建对象。

2，使用append()方法修改Stringbuffer的值，使用toString()方法转换为字符串。

Stringbuild是jdk1.5后用来替换stringBuffer的一个类，大多数时候可以替换

StringBuffer。和StringBuffer的区别在于Stringbuild是一个单线程使用的类，不值

执行线程同步所以比StringBuffer的速度快，效率高。

**十一：**

**·异常的分类：**

答：编译时异常： 程序正确，但因为外在的环境条件不满足引发。例如：用户错

误及I/O问题----程序试图打开一个并不存在的远程Socket端口。这不是程序本

身的逻辑错误，而很可能是远程机器名字错误(用户拼写错误)。对商用软件系统，

程序开发者必须考虑并处理这个问题。Java编译器强制要求处理这类异常，如果

不捕获这类异常，程序将不能被编译。

运行期异常： 这意味着程序存在bug，如数组越界，0被除，入参不满足规范.....

这类异常需要更改程序来避免，Java编译器强制要求处理这类异常。

异常的继承结构：

基类为Throwable，Error和Exception继承Throwable，RuntimeException和IOException等继承Exception，具体的RuntimeException继承RuntimeException。

Error和RuntimeException及其子类成为未检查异常（unchecked），其它异常成为已检查异常（checked）。

1、 SQLException：操作数据库异常类。解决方案：检查插入的数据是否重复。  
2、 ClassCastException：数据类型转换异常。解决方案：注意相应的组件在初始化的时候，是否相对应。  
3、 NumberFormatException：字符串转换为数字类型时抛出的异常。解决方案：检查数据是否正确。

4. java.lang.NullPointerException

5. java.lang.ClassNotFoundException　　异常的解释是"指定的类不存在"。  
6. java.lang.ArithmeticException　　这个异常的解释是"数学运算异常"，比如程序中出现了除以零这样的运算就会出这样的异常。  
7. java.lang.ArrayIndexOutOfBoundsException "数组下标越界"，

8. java.lang.IllegalArgumentException这个异常的解释是"方法的参数错误"，

9. java.lang.IllegalAccessException 这个异常的解释是"没有访问权限"，