互联网大型公司（阿里腾讯百度等）android面试题目(有答案)

Android 基础与底层机制

1. 数据库的操作类型有哪些，如何导入外部数据库?

> 数据库的操作类型:增、删、改、查;
> 把原数据库包括在项目源码的 res/raw
> android系统下数据库应该存放在 /data/data/com..（package name）/ 目录下，所以我们需要做的是把已有的数据库传入那个目录下.操作方法是用FileInputStream读取原数据库，再用FileOutputStream把读取到的东西写入到那个目录.

2. 是否使用过本地广播，和全局广播有什么差别?

> 全局广播(BroadcastReceiver)：发出去的广播可以被任何应用程序接收到，也可以接受来自任何应用程序的广播;
> 本地广播(LocalBroadcastManager)：发送的广播只会在自己App内传播，不会泄露给其他App，确保隐私数据不会泄露;其他App也无法向你的App发送该广播，不用担心其他App会来搞破坏;比系统全局广播更加高效;

3. 是否使用过 IntentService，作用是什么， AIDL 解决了什么问题?

> Service是在主线程中而没有另开新的线程执行任务,不能处理耗时操作;
> 而IntentServicee是有替代Service处理耗时操作的作用，从IntentService的源码中我们也可以看到IntentService实际也是通过Handler来进行主线程的UI更新等操作；IntentServicee有一个抽象方法onHandleIntent，在这里可以处理耗时操作;

> AIDL 可以跨进程访问其他应用程序，和其他应用程序通讯，多进程间通讯;

1. Activity、 Window、 View 三者的差别， fragment 的特点?

> Activity是Android应用程序的载体，允许用户在其上创建一个用户界面，并提供用户处理事件的API，如onKeyEvent, onTouchEvent等。 并维护应用程序的生命周期。
> 当我们调用Acitivity的 setContentView方法的时候实际上是调用的Window对象的setContentView方法，所

以我们可以看出Activity中关于界面的绘制实际上全是交给Window对象来做的。
Activity—>Window—>DecorView
Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图） LayoutInflater像剪刀，Xml配置像窗花图纸。

- 在Activity中调用attach，创建了一个Window
- 创建的window是其子类PhoneWindow，在attach中创建PhoneWindow
- 在Activity中调用setContentView(R.layout.xxx)
- 其中实际上是调用的getWindow().setContentView()
- 调用PhoneWindow中的setContentView方法
- 创建ParentView：作为ViewGroup的子类，实际是创建的DecorView(作为 FramLayout的子类）
- 将指定的R.layout.xxx进行填充通过布局填充器进行填充【其中的parent指的>* 就是DecorView】
- 调用到ViewGroup
- 调用ViewGroup的removeAllView()，先将所有的view移除掉
- 添加新的view：addView()
  fragment 特点
- Fragment可以作为Activity界面的一部分组成出现；
- 可以在一个Activity中同时出现多个Fragment，并且一个Fragment也可以在多个Activity中使用；
- 在Activity运行过程中，可以添加、移除或者替换Fragment；
- Fragment可以响应自己的输入事件，并且有自己的生命周期，它们的生命周期会受宿主Activity的生命周期影响。

- 描述一次网络请求的流程（新浪）

- 域名解析
- TCP的三次握手
- 建立TCP连接后发起HTTP请求
- 服务器响应HTTP请求
- 浏览器解析html代码
- 同时请求html代码中的资源（如js、css、图片等）
- 最后浏览器对页面进行渲染并呈现给用户
  参考：

- Handler、 Thread 和 HandlerThread 的差别（小米）

- Thread是一个线程；
- 我们知道Handler是用来异步更新UI的，更详细的说是用来做线程间的通信的，更新UI时是子线程与UI主线程之间的通信。那么现在我们要是想子线程与子线程之间的通信要怎么做呢？当然说到底也是用

Handler+Thread来完成（不推荐，需要自己操作Looper）

- HandlerThread就是（Handler+Thread结合），HandlerThread其实还是一个线程，它跟普通线程有什么不同之处是多了一个Looper，这个是子线程独有的Looper，用来做消息的取出和处理。

- 低版本 SDK 实现高版本 api（小米）

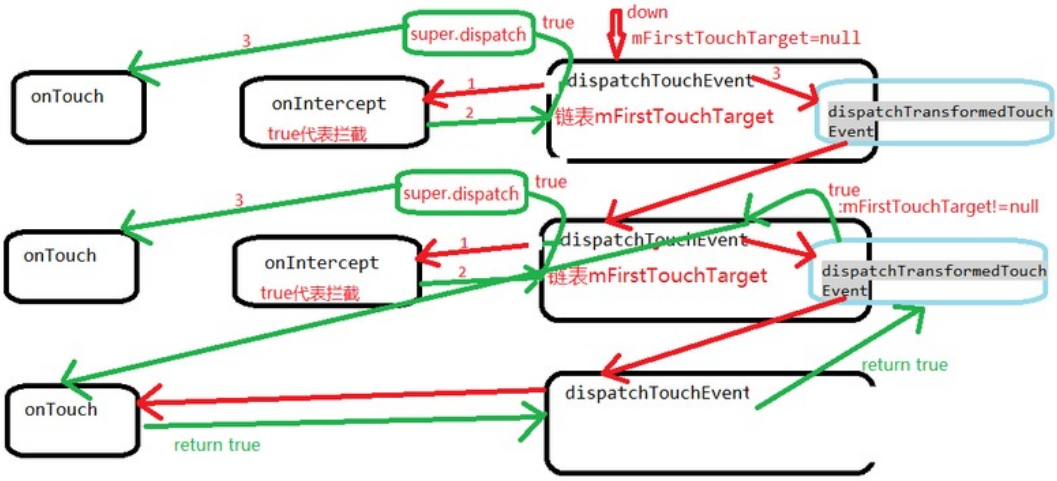
在使用高于minSdkVersion API level的方法需要:

- 用@TargeApi($API_LEVEL) 使可以编译通过, 不建议使用@SuppressLint(“NewApi”);
- 运行时判断API level; 仅在足够高，有此方法的API level系统中，调用此方法;
- 保证功能完整性，保证低API版本通过其他方法提供功能实现。

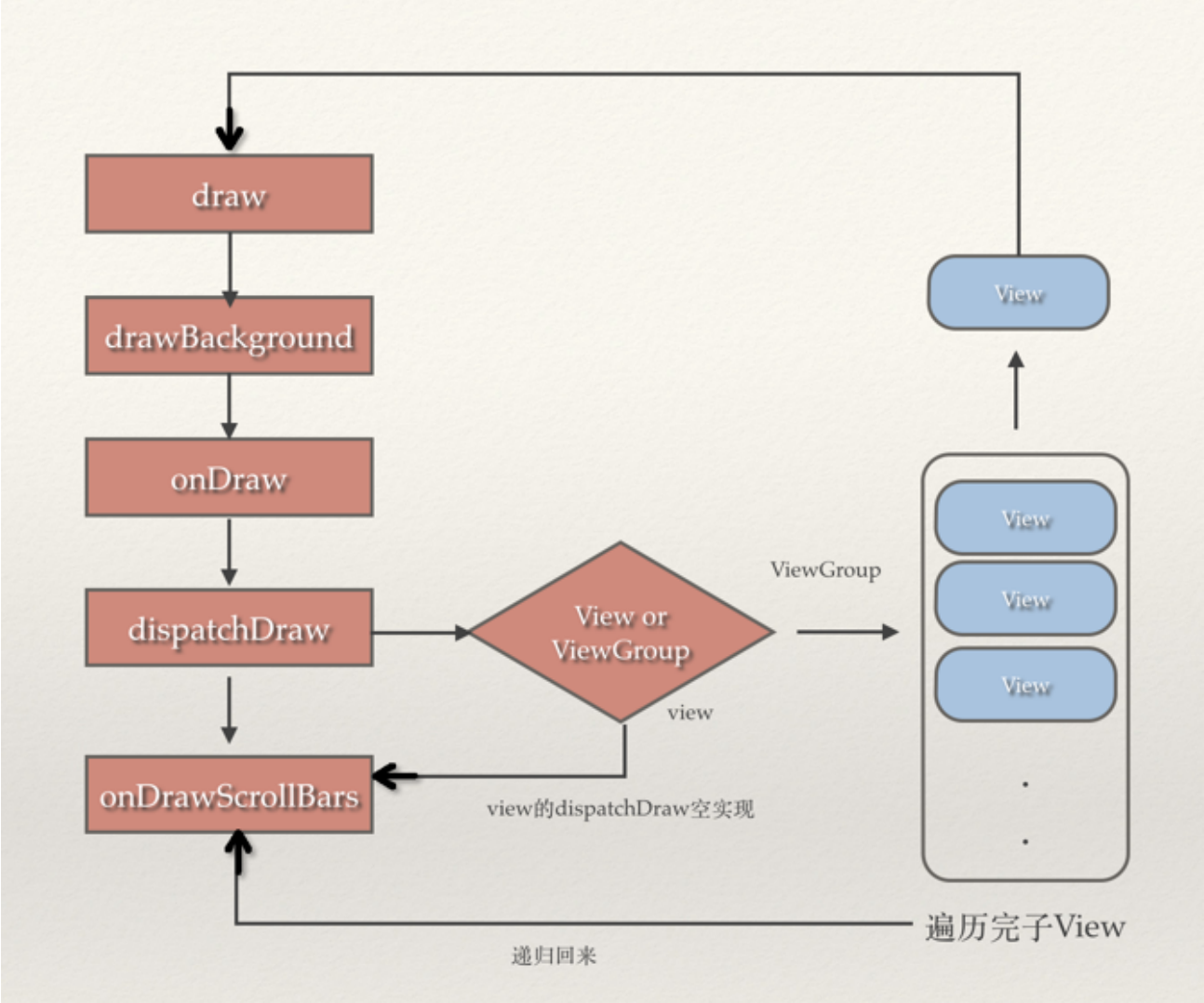
- launch mode 应用场景（百度、小米、乐视）

- standard：每次激活Activity时都会创建Activity实例，并放入任务栈中。这个是默认的；
- singleTop：如果已经存在在栈顶在对方的任务一个同类型的活动实例，不会有任何新的activity创造，而是被发送到一个存在的activity实例通过onNewIntent() 方法的意图，即会重用该实例调用当前activity的onNewIntent() 方法。适合接收通知启动的内容显示页面。
  例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。
- singleTask：如果系统中有一个存在的活动实例，整个任务将实例将被移动到顶部，Intent将通过onnewintent()方法交付。否则，新的activity将被创建并放置在适当的任务中。适合作为程序入口点。
  例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。
- singleInstance适合需要与程序分离开的页面。
  例如闹铃提醒，将闹铃提醒与闹铃设置分离。
  singleInstance不要用于中间页面，如果用于中间页面，跳转会有问题，比如：A -> B (singleInstance) -> C，完全退出后，在此启动，首先打开的是B。

- touch 事件传递流程（小米）



- view 绘制流程（百度）



https://www.jianshu.com/p/5a71014e7b1b

- 什么情况导致内存泄漏（美团）

- **1.单例造成的内存泄漏**
  由于单例的静态特性使得其生命周期和应用的生命周期一样长，如果一个对象已经不再需要使用了，而单例对象还持有该对象的引用，就会使得该对象不能被正常回收，从而导致了内存泄漏。
- **2.非静态内部类创建静态实例造成的内存泄漏**
  例如，有时候我们可能会在启动频繁的Activity中，为了避免重复创建相同的数据资源，可能会出现如下写法：

```
public class MainActivity extends AppCompatActivity {
   private static TestResource mResource = null;
   @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       if(mResource == null){
          mResource = new TestResource();
       }
      //...
   }

 class TestResource {
  //...
   }
}
```

  这样在Activity内部创建了一个非静态内部类的单例，每次启动Activity时都会使用该单例的数据。虽然这样避免了资源的重复创建，但是这种写法却会造成内存泄漏。因为非静态内部类默认会持有外部类的引用，而该非静态内部类又创建了一个静态的实例，该实例的生命周期和应用的一样长，这就导致了该静态实例一直会持有该Activity的引用，从而导致Activity的内存资源不能被正常回收。

解决方法：将该内部类设为静态内部类或将该内部类抽取出来封装成一个单例，如果需要使用Context，就使用Application的Context。

- **3.Handler造成的内存泄漏**

示例：创建匿名内部类的静态对象

```
public class MainActivity extends AppCompatActivity {
   private final Handler handler = new Handler() {
    @Override
   public void handleMessage(Message msg) {
        // ...
    }
  };

  @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      new Thread(new Runnable() {
          @Override
          public void run() {
```

```
                // ...
                handler.sendEmptyMessage(0x123);
            }
        });
    }
}
```

1、从Android的角度
当Android应用程序启动时，该应用程序的主线程会自动创建一个Looper对象和与之关联的MessageQueue。当主线程中实例化一个Handler对象后，它就会自动与主线程Looper的MessageQueue关联起来。所有发送到MessageQueue的Message都会持有Handler的引用，所以Looper会据此回调Handle的handleMessage()方法来处理消息。只要MessageQueue中有未处理的Message，Looper就会不断的从中取出并交给Handler处理。另外，主线程的Looper对象会伴随该应用程序的整个生命周期。
2、 Java角度
在Java中，非静态内部类和匿名类内部类都会潜在持有它们所属的外部类的引用，但是静态内部类却不会。对上述的示例进行分析，当MainActivity结束时，未处理的消息持有handler的引用，而handler又持有它所属的外部类也就是MainActivity的引用。这条引用关系会一直保持直到消息得到处理，这样阻止了MainActivity被垃圾回收器回收，从而造成了内存泄漏。
解决方法：将Handler类独立出来或者使用静态内部类，这样便可以避免内存泄漏。

- **4.线程造成的内存泄漏**

示例：AsyncTask和Runnable
AsyncTask和Runnable都使用了匿名内部类，那么它们将持有其所在Activity的隐式引用。如果任务在Activity销毁之前还未完成，那么将导致Activity的内存资源无法被回收，从而造成内存泄漏。
解决方法：将AsyncTask和Runnable类独立出来或者使用静态内部类，这样便可以避免内存泄漏。

- **5.资源未关闭造成的内存泄漏**

对于使用了BraodcastReceiver，ContentObserver，File，Cursor，Stream，Bitmap等资源，应该在Activity销毁时及时关闭或者注销，否则这些资源将不会被回收，从而造成内存泄漏。
1）比如在Activity中register了一个BraodcastReceiver，但在Activity结束后没有unregister该BraodcastReceiver。
2）资源性对象比如Cursor，Stream、File文件等往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们，以便它们的缓冲及时回收内存。它们的缓冲不仅存在于 java虚拟机内，还存在于java虚拟机外。如果我们仅仅是把它的引用设置为null，而不关闭它们，往往会造成内存泄漏。
3）对于资源性对象在不使用的时候，应该调用它的close()函数将其关闭掉，然后再设置为null。在我们的程序退出时一定要确保我们的资源性对象已经关闭。
4）Bitmap对象不在使用时调用recycle()释放内存。2.3以后的bitmap应该是不需要手动recycle了，内存已经在java层了。

- **6.使用ListView时造成的内存泄漏**

初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的View对象，同时ListView会将这些View对象缓存起来。当向上滚动ListView时，原先位于最上面的Item的View对象会被回收，然后被用来构造新出现在下面的Item。这个构造过程就是由getView()方法完成的，getView()的第二个形参convertView就是被缓存起来的Item的View对象（初始化时缓存中没有View对象则convertView是null）。
构造Adapter时，没有使用缓存的convertView。
解决方法：在构造Adapter时，使用缓存的convertView。

- **7.集合容器中的内存泄露**

我们通常把一些对象的引用加入到了集合容器（比如ArrayList）中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，这样这个集合就会越来越大。如果这个集合是static的话，那情况就更严重了。
解决方法：

在退出程序之前，将集合里的东西clear，然后置为null，再退出程序。

- **8.WebView造成的泄露**

当我们不要使用WebView对象时，应该调用它的destory()函数来销毁它，并释放其占用的内存，否则其长期占用的内存也不能被回收，从而造成内存泄露。
解决方法：
为WebView另外开启一个进程，通过AIDL与主线程进行通信，WebView所在的进程可以根据业务的需要选择合适的时机进行销毁，从而达到内存的完整释放。

- ANR 定位和修正

如果开发机器上出现问题，我们可以通过查看/data/anr/traces.txt即可，最新的ANR信息在最开始部分。

- 主线程被IO操作（从4.0之后网络IO不允许在主线程中）阻塞。
- 主线程中存在耗时的计算
- 主线程中错误的操作，比如Thread.wait或者Thread.sleep等 Android系统会>* 监控程序的响应状况，一旦出现下面两种情况，则弹出ANR对话框
- 应用在5秒内未响应用户的输入事件（如按键或者触摸）
- BroadcastReceiver未在10秒内完成相关的处理
- Service在特定的时间内无法处理完成 20秒
  修正
- 使用AsyncTask处理耗时IO操作。
- 使用Thread或者HandlerThread时，调用Process.setThreadPriority(Process.THREADPRIORITYBACKGROUND)设置优先级，否则仍然会降低程序响应，因为默认Thread的优先级和主线程相同。
- 使用Handler处理工作线程结果，而不是使用Thread.wait()或者Thread.sleep()来阻塞主线程。
- Activity的onCreate和onResume回调中尽量避免耗时的代码
- BroadcastReceiver中onReceive代码也要尽量减少耗时，建议使用IntentService处理。

- 什么情况导致 oom（乐视、美团）

这个跟11好像差不多

- Android Service 与 Activity 之间通信的几种方式

- 通过Binder对象
- 通过Intent
- 通过Broadcast广播
- 自定义接口回调

- Android 各个版本 API 的区别

> http://blog.csdn.net/u012964796/article/details/50662564

- 如何保证一个后台服务不被杀死,比较省电的方式是什么？（百度）

> - 双进程守护(就是开启二个service,二个service是不同的进程中，用aidl监听，有一个service杀死了，另一个service监听到了就重新启动刚杀死的service)，这个有时间时专门写文章介绍；
> - service绑定通知栏成为前台服务，
> - 还有一个就是锁屏时启动一个像素的activity,哈哈，好像某应用就是这么干的；
> - AlarmManager不断启动service
> - 通过jni调用，在c层启动多进程

- Requestlayout， onlayout， onDraw， DrawChild 区别与联系（猎豹）

> requestLayout()方法 ： 会导致调用measure()过程 和 layout()过程 。 将会根据标志位判断是否需要ondraw
> onLayout()方法(如果该View是ViewGroup对象，需要实现该方法，对每个子视图进行布局)
> 调用onDraw()方法绘制视图本身 (每个View都需要重载该方法，ViewGroup不需要实现该方法)
> drawChild()去重新回调每个子视图的draw()方法

- invalidate()和 postInvalidate() 的区别及使用（百度）

> postInvalidate() 方法在非 UI 线程中调用，通知 UI 线程重绘。
> invalidate() 方法在 UI 线程中调用，重绘当前 UI。

- Android 动画框架实现原理

> Animation框架定义了透明度，旋转，缩放和位移几种常见的动画，而且控制的是整个View，实现原理是每次绘制视图时View所在的ViewGroup中的drawChild函数获取该View的Animation的Transformation值，然后调用canvas.concat(transformToApply.getMatrix())，通过矩阵运算完成动画帧，如果动画没有完成，继续调用invalidate()函数，启动下次绘制来驱动动画，动画过程中的帧之间间隙时间是绘制函数所消耗的时间，可能会导致动画消耗比较多的CPU资源，最重要的是，动画改变的只是显示，并不能相应事件。

- Android 为每个应用程序分配的内存大小是多少？（美团）

> android程序内存一般限制在16M，也有的是24M

- Android View 刷新机制（百度、美团）

> 由ViewRoot对象的performTraversals()方法调用draw()方法发起绘制该View树，值得注意的是每次发起绘

图时，并不会重新绘制每个View树的视图，而只会重新绘制那些“需要重绘”的视图，View类内部变量包含了一个标志位DRAWN，当该视图需要重绘时，就会为该View添加该标志位。

调用流程：

mView.draw()开始绘制，draw()方法实现的功能如下：

绘制该View的背景为显示渐变框做一些准备操作(见5，大多数情况下，不需要改渐变框)调用onDraw()方法绘制视图本身 (每个View都需要重载该方法，ViewGroup不需要实现该方法)调用dispatchDraw ()方法绘制子视图(如果该View类型不为ViewGroup，即不包含子视图，不需要重载该方法)值得说明的是，ViewGroup类已经为我们重写了dispatchDraw ()的功能实现，应用程序一般不需要重写该方法，但可以重载父类函数实现具体的功能。

- LinearLayout 对比 RelativeLayout（百度）（Ricky）

RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure，RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。
最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

- 优化自定义 view（百度、乐视、小米）

为了加速你的view，对于频繁调用的方法，需要尽量减少不必要的代码。先从onDraw开始，需要特别注意不应该在这里做内存分配的事情，因为它会导致GC，从而导致卡顿。在初始化或者动画间隙期间做分配内存的动作。不要在动画正在执行的时候做内存分配的事情。

你还需要尽可能的减少onDraw被调用的次数，大多数时候导致onDraw都是因为调用了invalidate().因此请尽量减少调用invaildate()的次数。如果可能的话，尽量调用含有4个参数的invalidate()方法而不是没有参数的invalidate()。没有参数的invalidate会强制重绘整个view。

另外一个非常耗时的操作是请求layout。任何时候执行requestLayout()，会使得Android UI系统去遍历整个View的层级来计算出每一个view的大小。如果找到有冲突的值，它会需要重新计算好几次。另外需要尽量保持View的层级是扁平化的，这样对提高效率很有帮助。

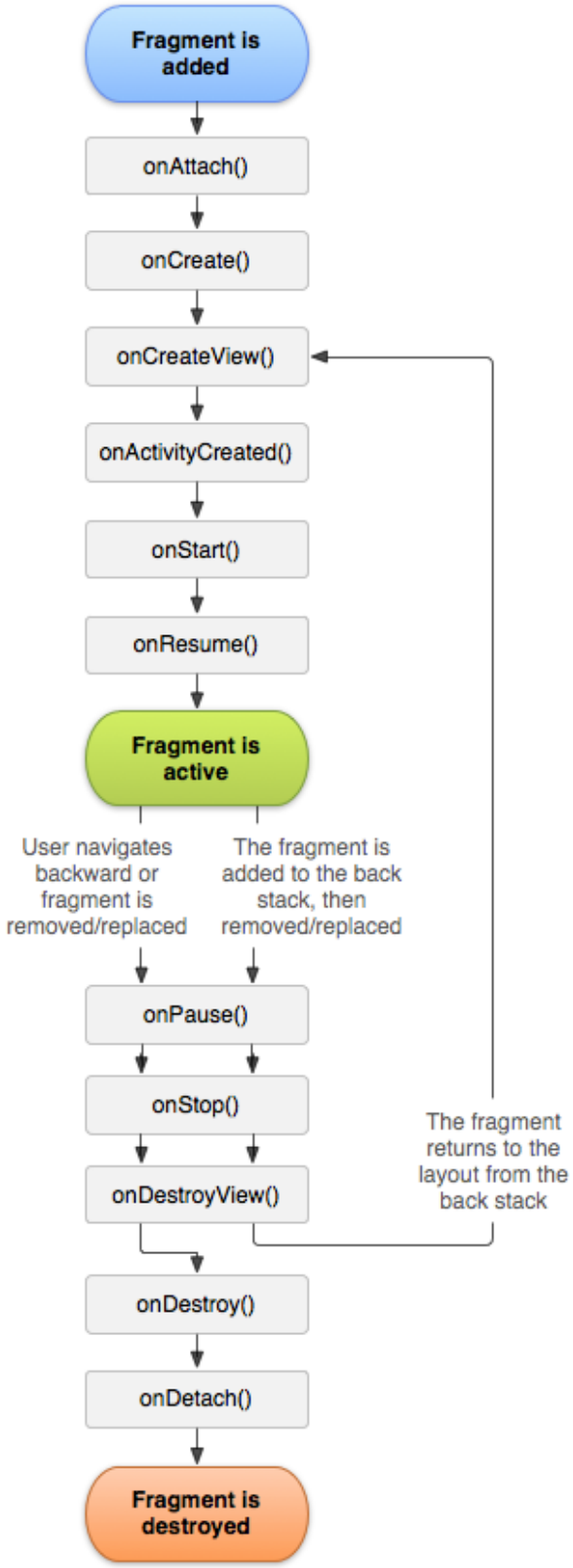
如果你有一个复杂的UI，你应该考虑写一个自定义的ViewGroup来执行他的layout操作。与内置的view不同，自定义的view可以使得程序仅仅测量这一部分，这避免了遍历整个view的层级结构来计算大小。这个PieChart 例子展示了如何继承ViewGroup作为自定义view的一部分。PieChart 有子views，但是它从来不测量它们。而是根据他自身的layout法则，直接设置它们的大小。

- ContentProvider（乐视）

http://blog.csdn.net/coder_pig/article/details/47858489

https://www.jianshu.com/p/f5ec75a9cfea

- fragment 生命周期



- volley 解析（美团、乐视）

> https://www.jianshu.com/p/15e6209d2e6f

- Android Glide 源码解析

> http://blog.csdn.net/guolin_blog/article/details/53759439

- Android 属性动画特性（乐视、小米）

https://www.jianshu.com/p/8076fe970a0c

- Handler 机制及底层实现

http://blog.csdn.net/lmj623565791/article/details/38377229

- Binder 机制及底层实现

http://blog.csdn.net/weijinqian0/article/details/52233529

Java 基础

1. 接口的意义（百度）

> 1、重要性：在Java语言中， abstract class 和interface 是支持抽象类定义的两种机制。正是由于这两种机制的存在，才赋予了Java强大的 面向对象能力。
> 2、简单、规范性：如果一个项目比较庞大，那么就需要一个能理清所有业务的架构师来定义一些主要的接口，这些接口不仅告诉开发人员你需要实现那些业务，而且也将命名规范限制住了（防止一些开发人员随便命名导致别的程序员无法看明白）。
> 3、维护、拓展性：比如你要做一个画板程序，其中里面有一个面板类，主要负责绘画功能，然后你就这样定义了这个类。可是在不久将来，你突然发现这个类满足不了你了，然后你又要重新设计这个类，更糟糕是你可能要放弃这个类，那么其他地方可能有引用他，这样修改起来很麻烦。
> 如果你一开始定义一个接口，把绘制功能放在接口里，然后定义类时实现这个接口，然后你只要用这个接口去引用实现它的类就行了，以后要换的话只不过是引用另一个类而已，这样就达到维护、拓展的方便性。
> 4、安全、严密性：接口是实现软件松耦合的重要手段，它描叙了系统对外的所有服务，而不涉及任何具体的实现细节。这样就比较安全、严密一些（一般软件服务商考虑的比较多）。

1. 抽象类的意义（乐视）

> 抽象类往往用来表征对问题领域进行分析、设计中得出的抽象概念，是对一系列看上去不同，但是本质上相同的具体概念的抽象。具体分析如下：
> 1.因为抽象类不能实例化对象，所以必须要有子类来实现它之后才能使用。这样就可以把一些具有相同属性和方法的组件进行抽象，这样更有利于代码和程序的维护。
> 2.当又有一个具有相似的组件产生时，只需要实现该抽象类就可以获得该抽象类的那些属性和方法。

1. 内部类的作用(百度，乐视)

> 定义：放在一个类的内部的类我们就叫内部类。
> 作用：
> 1.内部类可以很好的实现隐藏，一般的非内部类，是不允许有 private 与protected权限的，但内部类可以
> 2.内部类拥有外围类的所有元素的访问权限
> 3.可是实现多重继承
> 4.可以避免修改接口而实现同一个类中两种同名方法的调用。

2. 父类的静态方法能否被子类重写，为什么？（猎豹）

> 父类的静态方法是不能被子类重写的，其实重写只能适用于实例方法，不能用于静态方法，对于上面这种静态方法而言，我们应该称之为隐藏。
> Java静态方法形式上可以重写，但从本质上来说不是Java的重写。因为静态方法只与类相关，不与具体实现相关。声明的是什么类，则引用相应类的静态方法(本来静态无需声明，可以直接引用)。并且static方法不是后期绑定的，它在编译期就绑定了。换句话说，这个方法不会进行多态的判断，只与声明的类有关。

1. 举 1-2 个排序算法，并使用 java 代码实现（美团）

2. 列举 java 的集合和继承关系（百度、美团）

3. java 虚拟机的特性（百度、乐视）

> Java语言的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用模式Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

4. 哪些情况下的对象会被垃圾回收机制处理掉（乐视、美团、小米）

> Java 垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性。
> （1）超出对象的引用句柄的作用域时，这个引用句柄引用的对象就变成垃圾。
> （2）没有超出对象的引用句柄的作用域时，给这个引用句柄赋值为空时，这个引用句柄引用的对象就变成垃圾。
> （3）创建匿名对象时，匿名对象用完以后即成垃圾。

5. 进程和线程的区别（猎豹）

> 进程和线程都是一个时间段的描述，是CPU工作时间段的描述，不过是颗粒大小不同。
> 1. 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.
> 2. 线程的划分尺度小于进程，使得多线程程序的并发性高。
> 3. 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。
> 4. 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

> 5. 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

1. Java 中==和 equals 的区别， equals 和 hashCode 的区别（乐视）

   > http://blog.csdn.net/tiantiandjava/article/details/46988461

2. ArrayList 和 HashMap 的实现原理（美团，百度）
   https://www.jianshu.com/p/f174d49b391c
3. java 中 int char long 各占多少字节数
   short 2个字节
   int 4个字节
   long 8个字节
4. java int 与 integer 的区别

   > http://blog.csdn.net/chenliguan/article/details/53888018

5. string stringbuffer stringbuilder 区别（小米、乐视、百度）

   > String：适用于少量的字符串操作的情况
   > StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况（非线程安全）
   > StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况（线程安全）
   > 在大部分情况下 StringBuilder > StringBuffer
   > https://www.cnblogs.com/su-feng/p/6659064.html

6. Java 多态（乐视）

   > http://blog.csdn.net/Jian_Yun_Rui/article/details/52937791

7. 什么导致线程阻塞（58、美团）
   http://blog.csdn.net/sinat_22013331/article/details/45740641
8. 抽象类接口区别（360）
   https://www.cnblogs.com/yongjiapei/p/5494894.html
9. 容器类之间的区别（乐视、美团）
   https://www.cnblogs.com/sunliming/archive/2011/04/05/2005957.html
10. Java 中 HashMap 和 HashTable 的区别（乐视、小米）
    https://www.cnblogs.com/lchzls/p/6714335.html
    http://blog.csdn.net/u012050154/article/details/50905364
11. ArrayMap VS HashMap
    http://blog.csdn.net/vansbelove/article/details/52422087

数据结构与算法

1. 堆和栈在内存中的区别是什么(数据结构方面以及实际实现方面)
2. 最快的排序算法是哪个？给阿里 2 万多名员工按年龄排序应该选择哪个算法？堆和树的区别；写出快排代码；链表逆序代码（阿里）
3. 求 1000 以内的水仙花数以及 40 亿以内的水仙花数（百度）
4. 子串包含问题(KMP 算法)写代码实现
5. 万亿级别的两个 URL 文件 A 和 B，如何求出 A 和 B 的差集 C,(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)
6. 蚁群算法与蒙特卡洛算法
7. 写出你所知道的排序算法及时空复杂度，稳定性（小米）

其他

1. 死锁的四个必要条件

> 1）互斥条件，即某个资源在一段时间内只能由一个线程占有，不能同时被两个或两个以上的线程占有
> 2）不可抢占条件，线程所获得的资源在未使用完毕之前，资源申请者不能强行地从资源占有者手中夺取资源，而只能由该资源的占有者线程自行释放
> 3）占有且申请条件，线程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外线程占有，此时该线程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。
> 4）循环等待条件，存在一个线程等待序列{P1，P2，...，Pn}，其中P1等待P2所占有的某一资源，P2等待P3所占有的某一源，.....，而Pn等待P1所占有的的某一资源，形成一个线程循环等待环
> 解决死锁的办法：加锁顺序，死锁检测

1. 常见编码方式； utf-8 编码中的中文占几个字节；数字几个字节

> 一个utf8数字占1个字节，一个utf8英文字母占1个字节，少数是汉字每个占用3个字节，多数占用4个字节。

1. 实现一个 Json 解析器(可以通过正则提高速度)

```
String json = "{name:\"jason\",father:\"jason\",age:18}";
//name:"jason"
//age:18
//\"\\w+\" 字符串属性
Pattern p = Pattern.compile("\\w+:(\"\\w+\"|\\d*)");
Matcher m = p.matcher(json);
while(m.find()){
    String text = m.group();
```

```
9       int dotPos= text.indexOf(":");
10      String key = text.substring(0, dotPos);
11      String value = text.substring(dotPos+1, text.length());
12      //替换字符串的开始结束的双引号
13      value = value.replaceAll("^\\\"|\\\"$", "");
14      System.out.println(key);
15      System.out.println(value);
16  }
17
```

1. Android App 的设计架构： MVC,MVP,MVVM 与架构经验谈（搜狐）
2. 写出观察者模式的代码
3. TCP 的 3 次握手和四次挥手； TCP 与 UDP 的区别

> http://blog.csdn.net/whuslei/article/details/6667471
> 1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接
> 2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付
> 3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的
> UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）
> 4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信
> 5、TCP首部开销20字节;UDP的首部开销小，只有8个字节
> 6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

1. HTTP 协议； HTTP1.0 与 2.0 的区别；HTTP 报文结构
2. HTTP 与 HTTPS 的区别以及如何实现安全性

>