**Android高级面试题**

1、View和ViewGroup的关系

答：Android的UI界面都是由View和ViewGroup及其派生类组合而成的。

　其中，View是所有UI组件的基类，而 ViewGroup是容纳这些组件的容器，其本身也是从View派生出来的.

　View对象是Android平台中用户界面体现的基础单位。

　View类是它称为“widgets(工具)”的子类的基础，它们提供了诸如文本输入框和按钮之类的UI对象的完整实现。

　ViewGroup类同样为其被称为“Layouts(布局)”的子类奠定了基础，它们提供了象流式布局、表格布局以及相对布局之类的布局架构。

　一般来说，开发Android应用程序的UI界面都不会直接使用View和ViewGroup，而是使用这两大基类的派生类。

2、如何定位和解决Andorid的内存溢出问题

答：一、Android的内存机制

android应用层是由java开发的，android的davlik虚拟机与jvm也类似，只不过它是基于寄存器的。在java中，通过new为对象分配内存，所有对象在java堆内分配空间；而内存的释放是由垃圾收集器（GC）来回收的。 Java采用了有向图的原理。Java将引用关系考虑为图的有向边，有向边从引用者指向引用对象。线程对象可以作为有向图的起始顶点，该图就是从起始顶点(GC roots)开始的一棵树，根顶点可以到达的对象都是有效对象，GC不会回收这些对象。如果某个对象 (连通子图)与这个根顶点不可达(注意，该图为有向图)，那么我们认为这个(这些)对象不再被引用，可以被GC回收。

二、Android的内存溢出原因

1、内存泄露导致

由于我们程序的失误，长期保持某些资源（如Context）的引用，垃圾回收器就无法回收它，当然该对象占用的内存就无法被使用，这就造成内存泄露。

Android 中常见就是Activity被引用在调用finish之后却没有释放，第二次打开activity又重新创建，这样的内存泄露不断的发生,则会导致内存的溢出。

Android的每个应用程序都会使用一个专有的Dalvik虚拟机实例来运行，它是由Zygote服务进程孵化出来的，也就是说每个应用程序都是在属于自己的进程中运行的。Android为不同类型的进程分配了不同的内存使用上限，如果程序在运行过程中出现了内存泄漏的而造成应用进程使用的内存超过了这个上限，则会被系统视为内存泄漏，从而被kill掉，这使得仅仅自己的进程被kill掉，而不会影响其他进程.

2、占用内存较多的对象

保存了多个耗用内存过大的对象（如Bitmap）或加载单个超大的图片，造成内存超出限制。

三、常见的内存泄漏问题及其解决方案

1、引用没释放造成的内存泄露

1.1注册没取消造成的内存泄露

1.2集合中对象没清理造成的内存泄露

1.3 static

1.4、线程（内部类的使用）

3、Handler机制的原理，内部是如何实现的，消息队列的实现机制？

答：andriod提供了Handler 和 Looper 来满足线程间的通信。Handler先进先出原则。Looper类用来管理特定线程内对象之间的消息交换(MessageExchange)。

1)Looper: 一个线程可以产生一个Looper对象，由它来管理此线程里的MessageQueue(消息队列)。

2)Handler: 你可以构造Handler对象来与Looper沟通，以便push新消息到MessageQueue里;或者接收Looper从Message Queue取出)所送来的消息。

3) Message Queue(消息队列):用来存放线程放入的消息。

4)线程：UIthread 通常就是main thread，而Android启动程序时会替它建立一个MessageQueue。

4、APK之间互相传递数据

答：Android给每个APK进程分配一个单独的用户空间,其manifest中的userid就是对应一个Linux用户

(Android 系统是基于Linux)的.

所以不同APK(用户)间互相访问数据默认是禁止的.

但是它也提供了2种APK间共享数据的形式:

1. Share Preference. / Content Provider

APK可以指定接口和数据给任何其他APK读取. 需要自己实现接口和Share的数据.

本文对于这个不做详细解释

2. Shared User id

通过Shared User id,拥有同一个User id的多个APK可以配置成运行在同一个进程中.所以默认就是

可以互相访问任意数据. 也可以配置成运行成不同的进程, 同时可以访问其他APK的数据目录下的

数据库和文件.就像访问本程序的数据一样.

比如某个公司开发了多个Android 程序, 那么可以把数据,图片等资源集中放到APK  A中去. 然后

这个公司的所有APK都使用同一个User ID, 那么所有的资源都可以从APK A中读取.

举个例子:

APK A 和APK B 都是C公司的产品,那么如果用户从APK A中登陆成功.那么打开APK B的时候就不用

再次登陆. 具体实现就是 A和B设置成同一个User ID:

    \* 在2个APK的AndroidManifest.xml 配置User ID:

    <manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.android.demo.a1"

    android:sharedUserId="com.c">

   这个"com.c" 就是user id, 然后packagename APK A就是上面的内容,  APK B可能

   是"com.android.demo.b1" 这个没有限制

这个设定好之后, APK B就可以像打开本地数据库那样 打开APK A中的数据库了.

APK A把登陆信息存放在A的数据目录下面. APK B每次启动的时候读取APK A下面的数据库

判断是否已经登陆:

APK B中的代码:

            friendContext = this.createPackageContext(

                    "com.android.demo.a1",

                    Context.CONTEXT\_IGNORE\_SECURITY);

通过A的package name 就可以得到A的 packagecontext

通过这个context就可以直接打开数据库

5、简述Android的启动过程

**答：第一步：系统引导bootloader，即RK29xxLoaderXXX.bin文件**

       加电后，CPU将先执行 bootloader程序，然后bootloader首先会读寄存器地址base + APP\_DATA1的内容， 根据这个地址的值决定是否进入recovery模式或者其它模式。bootloader还会读取MISC分区第一块的内容， 决定进入recovery模式还是升级基带Baseband Processor（BP）或做其它事情

而上述寄存器与分区的值是有按键触发或者软件触发的。

a)       开机按reset+返回键，系统进入recovery模式，加载recovery.img，recovery.img包含内核，基本的文件系统，用于工程模式的烧写

b)       开机按Power，正常启动系统，加载boot.img，boot.img包含内核，基本文件系统，用于正常启动机器（以下只分析正常启动的情况）

**第二步： 启动内核kernel**

1)       源码：kernel/\*

2)       说明：kernel由bootloader加载

**第三步：    文件系统(rootfs)及应用初始化（init）**

1)       源码：system/core/init/\*

2)       配置文件：system/rootdir/init.rc，

3)       说明：init是一个由内核启动的用户级进程，它按照init.rc中的设置执行：启动服务（这里的服务指linux底层服务，如adbd提供adb支持，vold提供SD卡挂载等），执行命令和按其中的配置语句执行相应功能

**第四步：   重要的后台程序zygote**

1)       源码：frameworks/base/cmds/app\_main.cpp等

2)       说明：zygote是一个在init.rc中被指定启动的服务，该服务对应的命令是/system/bin/app\_process

a)       建立Java Runtime，建立虚拟机

b)       建立Socket接收ActivityManangerService的请求，用于Fork应用程序

c)       启动SystemServer

**第五步： 系统服务system server**

1)       源码：frameworks/base/services/java/com/android/server/SystemServer.java

2)       说明：被zygote启动，通过System Manager管理android的服务（这里的服务指frameworks/base/services下的服务，如卫星定位服务，剪切板服务等）

**第六步：桌面launcher**

1)       源码：ActivityManagerService.java为入口，packages/apps/launcher\*实现

2)       说 明：系统启动成功后SystemServer使用xxx.systemReady()通知各个服务，系统已经就绪，桌面程序Home就是在 ActivityManagerService.systemReady()通知的过程中建立的，最终调用 ()启launcher

**第七步：   解锁**

1)       源码：

frameworks/policies/base/phone/com/android/internal/policy/impl/\*lock\*

2)       说 明：系统启动成功后SystemServer调用wm.systemReady()通知WindowManagerService，进而调用 PhoneWindowManager，最终通过LockPatternKeyguardView显示解锁界面，跟踪代码可以看到解锁界面并不是一个 Activity，这是只是向特定层上绘图，其代码了存放在特殊的位置

**第八步：   开机自启动的第三方应用程序**

1)       源码：

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

2)       说 明：系统启动成功后SystemServer调用ActivityManagerNative.getDefault().systemReady()通知ActivityManager启动成功，ActivityManager会通过置变量mBooting，通知它的另一线程，该线程会发送广播android.intent.action.BOOT\_COMPLETED以告知已注册的第三方程序在开机时自动启动。

**第九步： 总结**综上所述，系统层次关于启动最核心的部分是zygote(即app\_process)和system server，zygote它负责最基本的虚拟机的建立，以支持各个应用程序的启动，而systemserver用于管理android后台服务，启动步骤及顺序。

6、如何加载ndk库？如何在jni中注册native函数，有几种注册方法？

答：注册native函数有两种方法：静态注册和动态注册。

1、静态注册方法

根据函数名找到对应的JNI函数:Java层调用函数时，会从对应的JNI中寻找该函数，如果没有就会报错，如果存在则会建立一个关联联系，以后在调用时会直接使用这个函数，这部分的操作由虚拟机完成。

静态方法就是根据函数名来遍历java和jni函数之间的关联，而且要求jni层函数的名字必须遵循

特定的格式，其缺点在于：

1）javah生成的jni层函数特别长；

2）初次调用native函数时要根据名字搜索对应的jni层函数来建立关联联系，这样影响效率。

2、动态注册方法

JNI 允许你提供一个函数映射表，注册给Jave虚拟机，这样Jvm就可以用函数映射表来调用相应的函数，

就可以不必通过函数名来查找需要调用的函数了。

Java与JNI通过JNINativeMethod的结构来建立联系，它在jni.h中被定义，其结构内容如下：

typedef struct {

const char\* name;

const char\* signature;

void\* fnPtr;

} JNINativeMethod;

第一个变量name是Java中函数的名字。

第二个变量signature，用字符串是描述了函数的参数和返回值

第三个变量fnPtr是函数指针，指向C函数。

当java通过System.loadLibrary加载完JNI动态库后，紧接着会查找一个JNI\_OnLoad的函数，如果有，就调用它，

而动态注册的工作就是在这里完成的。

1)JNI\_OnLoad()函数

JNI\_OnLoad()函数在VM执行System.loadLibrary(xxx)函数时被调用，它有两个重要的作用：

指定JNI版本：告诉VM该组件使用那一个JNI版本(若未提供JNI\_OnLoad()函数，VM会默认该使用最老的JNI 1.1版)，如果要使用新版本的JNI，

例如JNI 1.4版，则必须由JNI\_OnLoad()函数返回常量JNI\_VERSION\_1\_4(该常量定义在jni.h中) 来告知VM。

初始化设定，当VM执行到System.loadLibrary()函数时，会立即先呼叫JNI\_OnLoad()方法，因此在该方法中进行各种资源的初始化操作最为恰当，

2)RegisterNatives

RegisterNatives在AndroidRunTime里定义

syntax:

jint RegisterNatives(jclass clazz, const JNINativeMethod\* methods,jint nMethods)

7、如何收集软件崩溃信息？

答：1、Android异常捕获接口

2、设置线程捕获异常

3、通过网络发送Log

8、简述软件编译流程和发布流程

答：

9、自定义控件相关

答：1、了解View的工作原理

2、 编写继承自View的子类

3、 为自定义View类增加属性

4、 绘制控件

5、 响应用户消息

6 、自定义回调函数

一、View结构原理

Android系统的视图结构的设计也采用了组合模式，即View作为所有图形的基类，Viewgroup对View继承扩展为视图容器类。

View定义了绘图的基本操作

基本操作由三个函数完成：measure()、layout()、draw()，其内部又分别包含了onMeasure()、onLayout()、onDraw()三个子方法。具体操作如下：

1、measure操作

     measure操作主要用于计算视图的大小，即视图的宽度和长度。在view中定义为final类型，要求子类不能修改。measure()函数中又会调用下面的函数：

     （1）onMeasure()，视图大小的将在这里最终确定，也就是说measure只是对onMeasure的一个包装，子类可以覆写onMeasure()方法实现自己的计算视图大小的方式，并通过setMeasuredDimension(width, height)保存计算结果。

2、layout操作

     layout操作用于设置视图在屏幕中显示的位置。在view中定义为final类型，要求子类不能修改。layout()函数中有两个基本操作：

     （1）setFrame（l,t,r,b），l,t,r,b即子视图在父视图中的具体位置，该函数用于将这些参数保存起来；

     （2）onLayout()，在View中这个函数什么都不会做，提供该函数主要是为viewGroup类型布局子视图用的；

3、draw操作

     draw操作利用前两部得到的参数，将视图显示在屏幕上，到这里也就完成了整个的视图绘制工作。子类也不应该修改该方法，因为其内部定义了绘图的基本操作：

     （1）绘制背景；

     （2）如果要视图显示渐变框，这里会做一些准备工作；

     （3）绘制视图本身，即调用onDraw()函数。在view中onDraw()是个空函数，也就是说具体的视图都要覆写该函数来实现自己的显示（比如TextView在这里实现了绘制文字的过程）。而对于ViewGroup则不需要实现该函数，因为作为容器是“没有内容“的，其包含了多个子view，而子View已经实现了自己的绘制方法，因此只需要告诉子view绘制自己就可以了，也就是下面的dispatchDraw()方法;

     （4）绘制子视图，即dispatchDraw()函数。在view中这是个空函数，具体的视图不需要实现该方法，它是专门为容器类准备的，也就是容器类必须实现该方法；

     （5）如果需要（应用程序调用了setVerticalFadingEdge或者setHorizontalFadingEdge），开始绘制渐变框；

     （6）绘制滚动条；

      从上面可以看出自定义View需要最少覆写onMeasure()和onDraw()两个方法。

二、View类的构造方法

创建自定义控件的3种主要实现方式:

1）继承已有的控件来实现自定义控件: 主要是当要实现的控件和已有的控件在很多方面比较类似, 通过对已有控件的扩展来满足要求。

2）通过继承一个布局文件实现自定义控件，一般来说做组合控件时可以通过这个方式来实现。

    注意此时不用onDraw方法，在构造广告中通过inflater加载自定义控件的布局文件，再addView(view)，自定义控件的图形界面就加载进来了。

3）通过继承view类来实现自定义控件，使用GDI绘制出组件界面，一般无法通过上述两种方式来实现时用该方式。

四、控件绘制 onDraw()

六、自定义View的方法

onFinishInflate() 回调方法，当应用从XML加载该组件并用它构建界面之后调用的方法

onMeasure() 检测View组件及其子组件的大小

onLayout() 当该组件需要分配其子组件的位置、大小时

onSizeChange() 当该组件的大小被改变时

onDraw() 当组件将要绘\_制它的内容时

onKeyDown 当按下某个键盘时

onKeyUp  当松开某个键盘时

onTrackballEvent 当发生轨迹球事件时

onTouchEvent 当发生触屏事件时

onWindowFocusChanged(boolean)  当该组件得到、失去焦点时

onAtrrachedToWindow() 当把该组件放入到某个窗口时

onDetachedFromWindow() 当把该组件从某个窗口上分离时触发的方法

onWindowVisibilityChanged(int): 当包含该组件的窗口的可见性发生改变时触发的方法

10、AIDL怎么用？全称是什么？

答：AIDL：Android Interface Definition Language,即Android接口定义语言。

Android系统中的进程之间不能共享内存，因此，需要提供一些机制在不同进程之间进行数据通信。

为了使其他的应用程序也可以访问本应用程序提供的服务，Android系统采用了远程过程调用（Remote Procedure Call，RPC）方式来实现。与很多其他的基于RPC的解决方案一样，Android使用一种接口定义语言（Interface Definition Language，IDL）来公开服务的接口。我们知道4个Android应用程序组件中的3个（Activity、BroadcastReceiver和ContentProvider）都可以进行跨进程访问，另外一个Android应用程序组件Service同样可以。因此，可以将这种可以跨进程访问的服务称为AIDL（Android Interface Definition Language）服务。

11、图片过大如何处理？压缩后图片有多大？ 分辨率压缩、质量压缩

答：android图片压缩总结

总结来看，图片有三种存在形式：硬盘上时是file，网络传输时是stream，内存中是stream或bitmap，所谓的质量压缩，它其实只能实现对file的影响，你可以把一个file转成bitmap再转成file，或者直接将一个bitmap转成file时，这个最终的file是被压缩过的，但是中间的bitmap并没有被压缩（或者说几乎没有被压缩，我不确定），因为bigmap在内存中的大小是按像素计算的，也就是width \* height，对于质量压缩，并不会改变图片的像素，所以就算质量被压缩了，但是bitmap在内存的占有率还是没变小，但你做成file时，它确实变小了；

而尺寸压缩由于是减小了图片的像素，所以它直接对bitmap产生了影响，当然最终的file也是相对的变小了；

12、OOM怎么办？

13、平时看什么书？去哪些论坛看？

答：1、anddev

国外非常好的一个Android开发者论坛，论坛版块划分完全面向开发者，从入门到进阶话题很全面，版主的水平也非常高，经常会出一些教程。

地址：[http://www.anddev.org](http://www.anddev.org/)

2、 helloAndroid

以教程为最大特色的国外网站对大家系统学习Android知识非常有帮助

地址：[http://www.helloandroid.com](http://www.helloandroid.com/)

3、安卓之家

国内专注于android开发的论坛，刚开不久，相对来说比较冷清，但论坛的理念所在我比较认可，感觉向市委Android开发者提供了一站式服务

地址：[http://www.androidzj.com](http://www.androidzj.com/)

4、 ACC 开发者论坛

国内的Android开发论坛，相对eoe要冷清得多，不过国人开源项目CoolReader在这里发布，给论坛增色不少。

地址：[http://androidos.cc](http://androidos.cc/)

5、安卓视线 - Android开源项目分享平台

引用一下网站的介绍：Android是Google开发的基于Linux平台的开源手机操作系统。Android为我们勾画了一个美好的移动互联网前景，学习Android开源代码是掌握Android的一个最佳途径，我们希望每个对Android感兴趣的人都能够在这里掌握更多的知识、获得更多的灵感。

每天都有最新的Android开源项目推荐，对于大家学习优秀开源项目，开拓思路非常有帮助。

地址：[http://www.androideye.com](http://www.androideye.com/)

6、安卓航班网

很不错的安卓开发者论坛，有很多安卓学习的资料， 也有很多android源码项目可以下载，非常方便。适合初学者和高手进阶的好地方！

 地址：[http://www.apkway.com](http://www.apkway.com/)

14、谈谈你对Android应用开发市场前景，规范，整体架构等等的感想