一、基础题

1、谈谈 Activity 的生命周期

2、谈谈 Service 的生命周期

3、如何启用 Service，如何停用 Service？

4、Service 的 onXXX 方法能否执行耗时操作?

5、Service 有哪些启动方法，有什么区别，怎样停用 Service？

6、什么时候使用 Service？

7、什么是 IntentService？有何优点？

8、Android 系统优缺点

9、谈谈 Android 常用布局

10、请描述 Broadcast Receiver。

11、注册广播有几种方式，这些方式有何优缺点？

12、请介绍下 ContentProvider 是如何实现数据共享的。

13、说说 Intent 和 Intent Filter。

14、Intent 对象中包含七个常用属性，请写出这七个属性。

15、Android 的数据存储有哪些种类？

16、手写代码向内存卡保存字符串 Hello Android 至 a.txt 文件

17、手写代码读取内存卡中的 a.txt 文件

18、手写代码读取 SD 存卡中 Downloads 文件夹中的 a.txt 文件内容

19、手写代码将 Hello Android 保存至 SD 存卡的 Downloads 文件夹的 a.txt 文件中

20、手写代码解析以下 xml 为一个 ArrayList 集合。

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<users>
<user id="1001">
 <name>张飞</name>
 <password>1234</password>
 <phone>68357788</phone>
 <email>zhangfei@qq.com</email>
</user>
<user id="1002">
 <name>王菲</name>
 <password>1234</password>
 <phone>68995577</phone>
 <email>wangfei@qq.com</email>
</user>
```

21、手写代码读取当前包内的 students.db 中 student 表的一组学生记录。

student 表包含 id、name、sex、email 四个字段。结果保存 ArrayList 集合中

22、谈谈 Fragment 的生命周期

23、谈谈 Fragment 与 Activity 通信

24、ActivityA 启动 ActivityB 时，调用了两个 Activity 的哪些生命周期方法？调用的次序是？

25、简单描述 AIDL

26、谈谈 Android 的 IPC（进程间通信）机制

二、重要题

1、谈谈 Android 的线程间通信机制

2、Service 和 Thread 的区别

3、简述触摸事件分发机制

4、谈谈 View 绘图原理

5、Activity 在屏幕旋转时的生命周期

6、Activity 的启动模式有哪些？是什么含义？

7、跟 Activity 和 Task 有关的 Intent 启动方式有哪些？其含义？

8、谈谈 ListView 的优化，要求手写 getView 方法

9、谈谈 RecyclerView 多种布局的实现

三、提高题

1、简述 Android 消息传递的方式都有哪些？

2、简述广播与通过 Intent 向组件传递消息之间的异同点。

3、简述通知与广播之间在消息传递方面的不同。

4、如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

5、如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

6、View 如何刷新？简述什么是双缓冲？

一、基础题答案

1、谈谈 Activity 的生命周期

Activity 的生命周期就是 Activity 从被创建到被销毁的过程。这个过程中包括以下状态：

1、运行态：Activity 处于屏幕最上端且和用户能交互的状态。

2、暂停态：Activity 可见但不能与用户交互。

3、停止态：被其它 Activity 完全遮挡时。

4、销毁状态：Activity 被系统回收。

Android 提供了如下回调方法用于 Activity 不同状态切换时的操作，理解并合理地使用这些回调方法可以开发出高性能、低功耗的 app。

1)onCreate(0

Activity实例被系统后第一个被回调的方案。在该方法中，可以完成

a) 设置Activity管理的布局；

b) 实例化布局中的View；

c) 设置UI事件的各种监听；

d) 注册广播接收者；

e) 加载(包括从服务端获取的)数据；

f) 通过Bundle类型的参数获取前任Activity储存的数据。

2)onStart()

Activity可见时调用。

1、 onCreate()-onStart()

2、 onStop()->onRestart()->onStart()

在该方法中可以：

a) 启动Service；

b) 注册传感器等事件监听；

c) 启动动画

3)onResume()

Activity开始与用户交互时调用（无论是启动还是重新启动一个活动，该方法总是被调用的）。

onStart()-onResume()

onPause()-onResume()

4)onPause()

当 Activity 处于暂停、停止或被回收时，都会回调本方法。
该方法用于重要数据的持久化。

5)onStop()

Activity被停止并转为不可见阶段及后续的生命周期事件时调用。
处于停止状态的Activity中的所有数据和界面状态均保存在内存中。

6)onRestart()

重新启动Activity时调用。该活动仍在栈中，而不是启动新的活动。

7)onDestroy()

Activity被完全从系统内存中移除时调用，该方法被调用。
在该方法中释放相关资源。如网络框架、取消广播接收者的注册。

2、谈谈 Service 的生命周期

服务常用生命周期回调方法如下：

1、onCreate()

该方法在服务被创建时调用，该方法只会被调用一次，无论调用多少次 startService()或 bindService()方法，服务也只被创建一次。

2、onDestroy()

该方法在服务被终止时调用。

3、onStartCommand()

只有采用 startService 方法启动服务时才会回调本方法。本方法在服务开始运行时被调用。多次调用 startService 方法尽管不会多次创建服务，但 onStartCommand 方法会被多次调用。

4、onBind()

只有采用 bindService 方法启动服务时才会回调本方法。本方法在调用者与服务被绑定时被回调，当调用者与服务已经绑定，多次调用 bindService 方法并不会导致该方法被

多次调用。

5、onUnbind()

只有采用 bindService 方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用

6、onRebind()

当解除绑定时，onUnbind 方法返回 true 时，再次绑定才会回调 onRebing 方法。

3、如何启用 Service，如何停用 Service？

步骤 1、继承 Service 类

```
public class SMSService extends Service {}
```

步骤 2、在项目清单中注册 Service

在 AndroidManifest.xml 文件中的<application>节点里对服务进行配置:<service Android:name=".SMSService" />

步骤 3、启动 Service

调用 ContextWrapper.startService()启用服务，调用者与服务之间没有关连，即使调用者退出了，服务仍然运行。

如果采用 startService()方法启动服务，在服务未被创建时，系统会先调用服务的 onCreate()方法，接着调用 onStartCommand 方法。如果调用 startService()方法前服务已经被创建，多次调用 startService 方法并不会导致多次创建服务，但会导致多次调用 onStartCommand 方法。

采用 startService()方法启动的服务，只能调用 Context.stopService()方法结束服务，服务结束时会调用 onDestroy 方法。

提示： Android 不建议使用 Service.onStart()是建议调用 onStartCommand 方法。

步骤 3、绑定并启动 Service

ContextWrapper.bindService()启动。

使用 bindService()方法启用服务，调用者与服务绑定在了一起，调用者一旦解除绑定，服务若没有组件绑定，也将终止。

如果采用 ContextWrapper.bindService()方法启动服务，在服务未被创建时，系统会先调用服务的 onCreate()方法，接着调用 onBind()方法。这个时候调用者和服务绑定在一起，调用者退出了，系统就会先调用服务的 onUnbind()方法，接着调用 onDestroy()方

法。

如果调用 bindService()方法前服务已经被绑定，多次调用 bindService()方法并不会导致多次创建服务及绑定(也就是说 onCreate()和 onBind()方法并不会被多次调用)。如果调用者希望与正在绑定的服务解除绑定，可以调用 unbindService()方法，调用该方法也会导致系统调用服务的 onUnbind()-->onDestroy()方法。

4、Service 的 onXXX 方法能否执行耗时操作?

默认情况,如果没有显示的指定 Service 所运行的进程, Service 和 Activity 是运行在当前 app 所在进程的 main thread 里面。

Service 里面不能执行耗时的操作(网络请求,拷贝数据库,大文件)

耗时操作放在子线程中执行。

特殊情况 ,可以在清单文件配置 service 执行所在的进程 ,让 service 在另外的进程中执行

<service Android:process="cn.ucai.xxx"></service>

5、Service 有哪些启动方法，有什么区别，怎样停用 Service？

通常有两种方式启 7 个 Service,他们对 Service 生命周期的影响是不一样的。

1、通过 startService

Service 会经历 onCreate 到 onStart，然后处于运行状态，stopService 的时候调用 onDestroy 方法。

如果是调用者自己直接退出而没有调用 stopService 的话,Service 会一直在后台运行。

2、通过 bindService

Service 会运行 onCreate，然后是调用 onBind， 这个时候调用者和 Service 绑定在一起。调用者退出了，Srevice 就会调用 onUnbind->onDestroyed 方法。

所谓绑定在一起就共存亡了。调用者也可以通过调用 unbindService 方法来停止服务，这时候 Srevice 就会调用 onUnbind->onDestroyed 方法。

需要注意的是如果这几个方法交织在一起的话，会出现什么情况呢？

一个原则是 Service 的 onCreate 的方法只会被调用一次，就是你无论多少次的 startService 又 bindService，Service 只被创建一次。

如果先是 bind 了,那么 start 的时候就直接运行 Service 的 onStart 方法,如果先是 start，那么 bind 的时候就直接运行 onBind 方法。

如果 service 运行期间调用了 bindService,这时候再调用 stopService 的话,service 是不会调用 onDestroy 方法的,service 就 stop 不掉了,只能调用 UnbindService, service 就会被销毁

如果一个 service 通过 startService 被 start 之后，多次调用 startService 的话，service 会多次调用 onStart 方法。多次调用 stopService 的话，service 只会调用一次 onDestroyed 方法。

如果一个 service 通过 bindService 被 start 之后，多次调用 bindService 的话，service 只会调用一次 onBind 方法。

多次调用 unbindService 会抛出异常。

在 Service 中调用 stopSelf 方法停止自身。

6、什么时候使用 Service？

拥有 Service 的进程具有较高的优先级

Android 系统会尽量保持拥有 Service 的进程运行，只要在该 Service 已经被启动

(start)或者客户端连接(bindService)到它。当内存不足时，需要保持，拥有 service 的进程具有较高的优先级。
1．如果 Service 正在调用 onCreate， onStartCommand 或者 onDestory 方法，那么用于当前 Service 的进程相当于前台进程以避免被 killed。
2．如果当前 Service 已经被启动，拥有它的进程则比那些用户可见的进程优先级低一些，但是比那些不可见的进程更重要，这就意味着 Service 一般不会被 killed.
3．如果客户端已经连接到 Service (bindService),那么拥有 Service 的进程则拥有最高的优先级，可以认为 Service 是可见的。
4． 如果 Service 可以使用 startForeground(true)方法来将 Service 设置为前台状态，那么系统就认为是对用户可见的，不会在内存不足时 killed。
如果有其它应用组件作为 Service,Activity 等运行在相同的进程中，那么将会增加该进程的重要性。
1.Service 的特点可以是可以在后台一直运行,可以在 Service 里面创建线程去完成耗时的操作。
2.BroadcastReceiver 捕获到一个事件之后,可以起一个 Service 来完成一个耗时的操作。
BroadcastReceiver 生命周期和响应时间很短。
3.远程的 Service 如果被启动起来,可以被多次 bind， 但不会重新 create.手机的人脸识别的 Service 可以被图库使用,可以被摄像机、照相机等程序使用.
4、检查项目的版本更新的代码应放在 Service 的工作线程中执行，当有新版本需要更新应用时，在 Service 的工作线程中向 Activity 发通知，提醒更新版本。

7、什么是 IntentService？有何优点？

普通的 service ,默认运行在 ui main 主线程
Sdk 给我们提供的方便的,带有异步处理的 service 类,
异步处理的方法 OnHandleIntent()
OnHandleIntent() 处理耗时的操作
Acitivity 的进程，当处理 Intent 的时候，会产生一个对应的 Service
Android 的进程处理器尽可能的不 kill 掉 IntentService。
非常容易使用。
重要提示：
我使用的 OkHttp 框架后，无需使用 IntentService，直接在 Service 中使用即可。IntentService 还需要使用 Handler 和 Message 在 onHandleIntent 方法与主线程之间通信。
而我自己封装的 OkHttp 框架已实现了工作线程与主线程之间的通信。

8、Android 系统优缺点

一、优点

(一)开放性

在优势方面，Android 平台首先就是其开发性，开发的平台允许任何移动终端厂商加入到 Android 联盟中来。显著的开放性可以使其拥有更多的开发者，随着用户和应用的日益丰富，一个崭新的平台也将很快走向成熟。开放性对于 Android 的发展而言，有利于积累人气，

这里的人气包括消费者和厂商，而对于消费者来讲，最大的受益是丰富的软件资源。开放的平台也会带来更大竞争，如此一来，消费者将可以用更低的价位购得心仪的手机。

(二)挣脱运营商的束缚

在过去很长的一段时间，特别是在欧美地区，手机应用往往受到运营商制约，使用什么功能接入什么网络，几乎都受到运营商的控制。从去年 iPhone 上市 ，用户可以更加方便地连接网络，运营商的制约减少。随着 EDGE、HSDPA 这些 2G 至 3G 移动网络的逐步过渡和提升，手机随意接入网络已不是运营商口中的笑谈，当你可以通过手机 IM 软件方便地进行即时聊天时，再回想不久前天价的彩信和图铃下载业务，是不是像噩梦一样?互联网巨头 Google 推动的 Android 终端天生就有网络特色，将让用户离互联网更近。

(三)丰富的硬件选择

这一点还是与 Android 平台的开放性相关，由于 Android 的开放性，众多的厂商会推出千奇百怪，功能特色各具的多种产品。功能上的差异和特色，却不会影响到数据同步、甚至软件的兼容，好比你从诺基亚 Symbian 风格手机 一下改用苹果 iPhone ，同时还可将 Symbian 中优秀的软件带到 iPhone 上使用、联系人等资料更是可以方便地转移，是不是非常方便呢?

(四)不受任何限制的开发商

Android 平台提供给第三方开发商一个十分宽泛、自由的环境，不会受到各种条条框框的阻扰，可想而知，会有多少新颖别致的软件会诞生。

Android 系统可以深度定制，使得很多厂商在 Android 原生系统基础上能设计自己具有自己特色的手机系统。并且，开放性可以缩短开发周期，降低开发成本。

(五)无缝融合 Google 应用

如今叱诧互联网的 Google 已经走过 10 年度历史，从搜索巨人到全面的互联网渗透，Google 服务如地图、邮件、搜索等已经成为连接用户和互联网的重要纽带，而 Android 平台手机将无缝结合这些优秀的 Google 服务。

(六)发开门槛和成本低

Android 应用层使用 Java 开发，使得众多的 Java 开发企业和工程师可以低成本地转入 Android 开发。

由于 Android 开发无需付费，因此有效减少了企业开发的开发成本。

二、缺点

(一)安全和隐私

由于手机与互联网的紧密联系，个人隐私很难得到保守。除了上网过程中经意或不经意留下的个人足迹，Google 这个巨人也时时站在你的身后，洞穿一切，因此，互联网的深入将会带来新一轮的隐私危机。

(二)首先开卖 Android 手机的不是最大运营商

众所周知，T-Mobile 在 23 日，于美国纽约发布 了 Android 首款手机 G1。但是在北美市场，最大的两家运营商乃 AT&T 和 Verizon，而目前所知取得 Android 手机销售权的仅有 T-Mobile 和 Sprint，其中 T-Mobile 的 3G 网络相对于其他三家也要逊色不少，因此，用户可以买账购买 G1，能否体验到最佳的 3G 网络服务则要另当别论了!

(三)运营商仍然能够影响到 Android 手机

在国内市场，不少用户对购得移动定制机不满，感觉所购的手机被人涂画了广告一般。这样的情况在国外市场同样出现。Android 手机的另一发售运营商 Sprint 就将在其机型中内置其手机商店程序。

(四)同类机型用户减少

在不少手机论坛都会有针对某一型号的子论坛，对一款手机的使用心得交流，并分享软件资源。而对于 Android 平台手机，由于厂商丰富，产品类型多样，这样使用同一款机型的用户越来越少，缺少统一机型的程序强化。举个稍显不当的例子，现在山寨机泛滥，品种各异，就很少有专门针对某个型号山寨机的讨论和群组，除了哪些功能异常抢眼、颇受追捧的机型以外。

(五)过分依赖开发商缺少标准配置

在使用 PC 端的 Windows Xp 系统的时候，都会内置微软 Windows Media Player 这样一个浏览器程序，用户可以选择更多样的播放器，如 Realplay 或暴风影音等。但入手开始使用默认的程序同样可以应付多样的需要。在 Android 平台中，由于其开放性，软件更多依赖第三方厂商，比如 Android 系统的 SDK 中就没有内置音乐 播放器，全部依赖第三方开发，缺少了产品的统一性。

(六)碎片化

Android 手机屏幕尺寸的多样化，使得开发应用程序必须考虑适配更多的机型。

(七)升级慢

Android 系统升级后，很多厂商、个人并没有及时升级到最新版本。这一点 IOS 比 Android 要好得多。

使得应用程序必须考虑兼容老版本的系统。

9、谈谈 Android 常用布局

常用六种布局方式，分别是：

FrameLayout（帧布局）
LinearLayout （线性布局）
AbsoluteLayout（绝对布局）
RelativeLayout（相对布局）
TableLayout（表格布局）。
GridLayout（网格布局）

1、FrameLayout

特点：

所有 view 依次都放在左上角，会重叠。

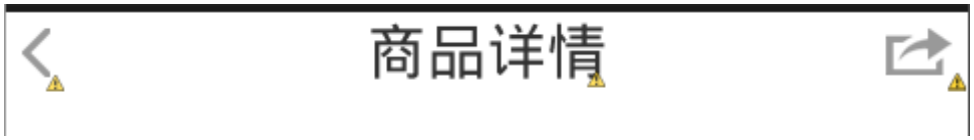
提供了 gravity 属性设置布局内子 view 的对齐方式。

子 view 中可使用 layout_gravity 属性设置在 FrameLayout 中的对齐方式。

应用场景：

1、使用频率较高。
2、作为占位符，例如在 Fragment 中作为占位符。
3、需要两个 view 叠加在一起时，如设置切换效果的两个 view，一个 view 表示开启，一个 view 表示关闭。

4、简单对齐时，靠左、靠右、居中，如页面的标题行，标题行左边有一个返回按钮、中间是显示标题的 TextView、右边有一个发送、分享之类的按钮。如下图所示：



2、LinearLayout

特点：

1、线性布局的控件按方向线性排列，方向分为
垂直布局（Android:orientation="vertical"）和
水平布局（Android:orientation="horizontal" ）。

当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；

水平布局时，只有一行，每一个元素依次向右排列。超出一行的 view 被裁剪。

2、weight 属性，权重，控件在屏幕上所占位置的比例。

1)weight 值为 0 时，控件在屏幕上所占空间由 width 和 height 设置。

2)weight 值为 1 时，

(1)控件的宽、高值为固定值或 wrap_content， weight 值与控件所占空间成正比。

(2)控件的宽、高值为 match_parent 时，weight 值与控件所占空间成反比。

3、垂直方向，不能使用 layout_gravity=”center_vertical 设置子 view 垂直居中。

水平方向，不能使用 layout_gravity=”center_horizontal 设置子 view 水平居中。

应用场景：

1、使用频率很高。
2、适合线性排列的布局。
3、复杂布局需要嵌套布局。
4、利用 weight 可以方便地实现多屏适配。
5、利用 weight 可以实现底部(或顶部)加显示内容填充剩余空间的效果。

3、AbsoluteLayout

特点：

绝对布局用 X,Y 坐标来指定元素的位置，这种布局方式也比较简单，但是在屏幕旋转时，往往会出问题，而且多个元素的时候，计算比较麻烦。

不利于多屏适配。

应用场景：

企业开发不用。

4、RelativeLayout

特点：

1、相对布局是以某个元素为参照物来定位的布局方式。参照物可以是父容器布局，也可以是容器中的某个子 view。

默认 view 在父容器的左上，允许 view 叠加。

应用场景：

1、使用频率很高。
2、帧布局应用场景中的 2、3、4。
3、可以实现底部(或顶部)菜单加显示内容填充剩余空间的效果。
参见 RelativeDemo.zip。

5、TableLayout

特点：

1、表格布局，TableLayout 表示多行多列
2、TableRow 表示表格的一行。
3、每个控件占一列。
4、strenchColumn 属性设置拉伸的列的索引。
5、控件的宽高属性可以省略。

应用场景：

1、适合多行多列，有规律的界面。
2、可以实现行的简单多屏适配。
3、使用频率：一般。

6、GridLayout

特点：

1、支持线性布局的排列方式。
2、默认横向布局，控件可折行显示。
3、可设置每行最大列数。
4、可设置最大行数。
5、控件可省略宽高的设置。
6、支持控件跨行、跨列显示。

应用场景：

sdk4.0 及其以上的 Android 手机系统。
考虑到兼容低版本的手机系统，目前使用频率很低。

10、请描述 Broadcast Receiver。

Broadcast Receiver用于接收并处理广播通知。多数的广播是系统发起的，如地域变换、电量不足、来电来信等。程序也可以播放一个广播。程序可以有任意数量的broadcast receivers来响应它觉得重要的通知。broadcast receiver可以通过多种方式通知用户：启动activity、使用NotificationManager、开启背景灯、振动设备、播放声音等，最典型的是在状态栏显示一个图标，这样用户就可以点它打开看通知内容。通常我们的某个应用或系统本身在某些事件（电池电量不足、来电来短信）来临时会广播一个Intent出去，我们可以利用注册一个Broadcast Receiver来监听到这些Intent并获取Intent中的数据。

11、注册广播有几种方式，这些方式有何优缺点？

答：首先写一个类要继承BroadcastReceiver
第一种:在清单文件中声明,添加
第二种使用代码进行注册如:
两种注册类型的区别是：

1)第一种是非常驻型广播，也就是说广播跟随组件的生命周期。

2)第二种是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。

开发中尽量用非常驻型广播，可以节省系统资源。

12、请介绍下 ContentProvider 是如何实现数据共享的。

一个程序可以通过实现一个Content provider的抽象接口将自己的数据完全暴露出去，而且Content providers是以类似数据库中表的方式将数据暴露。Content providers存储和检索数据，通过它可以让所有的应用程序访问到，这也是应用程序之间唯一共享数据的方法。

要想使应用程序的数据公开化，可通过2种方法：创建一个属于你自己的Content provider或者将你的数据添加到一个已经存在的Content provider中，前提是有相同数据类型并且有写入Content provider的权限。

如何通过一套标准及统一的接口获取其他应用程序暴露的数据？Android提供了ContentResolver，外界的程序可以通过ContentResolver接口访问ContentProvider提供的数据。

13、说说 Intent 和 Intent Filter。

1、Intent 的作用

Android 中通过 Intent 实现组件的调用与激活。

Intent 可以携带数据，在组件之间传递数据。

2、Intent filter

意图过滤器，设置组件被启动的条件。包含以下属性：

1)action: 动作

2)data: 数据 uri

3)category :类别

1)Action 匹配

Action 是一个用户定义的字符串,用于描述一个 Android 应用程序组件,一个 Intent Filter 可以多个 Action。在 AndroidManifest.xml 的 Activity 定义时可以在其<intent-filter >节点指定一个 Action 列表用于标示 Activity 所能接受的“动作”。

启动该 Activity 时只要符合列表中的一个 acion 值，即可启动该 Acivity。

Android 预定义了一系列的 Action 分别表示特定的系统动作。这些 Action 通过常量的方式定义在 android.content. Intent 中，以“ACTION_”开头。可以在 Android 提供的文档中找到它们的详细说明。

2)URI 数据匹配

一个 Intent 可以通过 URI 携带外部数据给目标组件。在 <intent-filter >节点中，通过 <data/>节点匹配外部数据。

mimeType 属性指定携带外部数据的数据类型；

scheme 指定协议；

host、port、path 指定数据的位置、端口、和路径。如下：

```
<data
    android:mimeType="mimeType"
    android:scheme="scheme"
    android:host="host"
    android:port="port"
    android:path="path"/>
```

例如：

电话的 uri tel: 12345

http://www.baidu.com

自定义的 uri ucai://www.sxt/person/10

如果在 Intent Filter 中指定了这些属性，那么只有所有的属性都匹配成功时 URI 数据匹配才会成功。

3)Category 类别匹配

<intent-filter >节点中可以为组件定义一个 Category 类别列表，当 Intent 中包含这个列表的所有项目时 Category 类别匹配才会成功。

默认是 DEFAULT

14、Intent 对象中包含七个常用属性，请写出这七个属性。

答：

1、mComponent：组件，类型是 Component Name；

2、mAction：动作(Action)类型是 String；

3、mData：数据(Data)，类型是 Uri；

4、mCategories：分类(Category)，类型是 ArraySet<String>；

5、mExtras：额外信息（Extra），类型是 Bundle；

6、mFlags：标志（Flags），类型是 int。

7、mType:mData 的类型，String 类型。

同时设置 data 和 Type 时，必须调用：

```
intent.setDataAndType(Uri data,String type);
```

而不能分别调用以下两个方法：

```
intent.setData();
intent.setType();
```

15、谈谈 Android 的数据存储

答：

1) SharedPreperences：首选项，存储键值对类型的数据，适合存储配置信息等少量

的数据。数据存储的位置：data/data/包名/shared_pref

2) 文件存储，分为内存储和外存储：

a) 内存储文件：属于文件存储，适合保存数据量比较大的场景。文件的存储位置：

Data/data/包名/files

b) 外存储：媒体扫描器可以访问的存储设备，若是文件类型则 SD 卡是重要的外存储器，适合存放大容量的数据，如音频、视频、图片等。

3) 数据库存储：数据存放在 SQLite 数据库中，存储的位置：

Data/data/包名/databases

4) ContentProvider：

当本应用向其它应用共享其数据时，虽然其它方法也可以对外共享数据，但数据访问方式会因数据存储的方式而不同，如：采用文件方式对外共享数据，需要进行文件操作读写数据；采用 sharedpreferences 共享数据，需要使用 sharedpreferences API 读写数据。而使用 ContentProvider 共享数据的好处是统一了数据访问方式。

5) 网络存储

通过网络与远程的手机或服务器进行数据交互。常见的操作与从服务端下载数据、图片，向服务端注册用户信息、上传头像等。

22、谈谈 Fragment 的生命周期

答：

1、Fragment 的生命周期方法与其宿主 Activity 联系十分紧密。

2、Activity 直接影响它所包含的 fragment 的生命周期，所以对 Activity 的某个生命周期方法的调用也会产生对 fragment 相同方法的调用。

3、Fragment 比 Activity 还要多出几个生命周期回调方法，这些额外的方法是为了与 Activity 的交互而设立。

4、Fragment 各生命周期方法

1)onAttach 方法：Fragment 对象与宿主 Activity 绑定时回调 onAttach 方法。

当已 fragment 标签嵌入 Activity 布局时，

2)onAttach、onCreate 和 onCreateView 三个方法在宿主 Activity.onCreate 方法的 setContentView 方法执行中被回调。

3)onCreate 方法：当 Activity 和 Fragment 是从保存状态重新创建时，可从本方法中获取保存的数据。

onCreate 方法中可以编写非 UI 操作的代码，如在后台线程中执行下载操作等。

4)onCreateView 方法用于创建 Fragment 的布局。

5) onActivityCreated 方法在宿主 Activity 的 onCreated 方法回调后调用。

onActivityCreated()回调时，宿主 Activity 和碎片的视图都已创建完毕。

6)onStart 方法是 Fragment 被用户看见时调用。

7)onResume 方法是 Fragment 能与用户交互时调用。

8)onPause 方法是 Fragment 进入暂停状态时回调，或 Fragment 进入停止、销毁状态时也被回调。

9)onStop 方法是 Fragment 进入停止状态时回调，或 Fragment 销毁时会被回调。

10) onDestroyView 方法：当 fragment 的 layout 被销毁时被调用，onDestroyView 在宿主 Activity 的 onStop 方法之后调用。

11)onDestroy 方法在宿主 Activity 的 onDestroy 方法之前调用，onDestroy 方法回调时，

碎片仍附加在宿主 Activity 上，但不能被调用。

12)当 fragment 被从 Activity 中删掉时回用 onDetach 方法，onDetach 是 Fragment 的最后一个生命周期方法，该方法执行后，碎片就不会与宿主 Activity 绑定，不再拥有视图，所有的资源将被释放。

23、Fragment 与 Activity 通信

答：

1、通过 setArguments 方法在启动的 Fragment 中设置传递的数据；在 Fragment 中用 getArguments 方法取出数据。

2、通过定义接口并在宿主 Activity 中实现接口，为 Fragment 暴露 Activity 中相应的方法、成员变量，与 Fragment 实现通信。

参见 F:\0-Android\0-优才教育\201603 班\就业\doc\code\day16_07

3、 通过广播在两个 Fragment 之间传递数据。

24、ActivityA 启动 ActivityB 时，调用了两个 Activity 的哪些生命周期方法？调用的次序是？

答：依次调用了 A.onPause()->B.onCreate()->B.onStart()->B.onResume()->A.onStop()

追问：为什么 A.onStop()最后调用？

答：为了使 ActivityB 尽快启动。

25、谈谈 Android 的 IPC（进程间通信）机制

IPC是内部进程通信的简称， 是共享"命名管道"的资源。Android中的IPC机制是为了让Activity和Service之间可以随时的进行交互，故在Android中该机制，只适用于Activity和Service之间的通信，类似于远程方法调用，类似于C/S模式的访问。通过定义AIDL接口文件来定义IPC接口。Servier端实现IPC接口，Client端调用IPC接口本地代理。

26、简单描述 AIDL

答：由于每个应用程序都运行在自己的进程空间，并且可以从应用程序UI运行另一个服务进程，而且经常会在不同的进程间传递对象。在Android平台，一个进程通常不能访问另一个进程的内存空间，所以要想对话，需要将对象分解成操作系统可以理解的基本单元，并且有序的通过进程边界。

通过代码来实现这个数据传输过程是冗长乏味的，Android提供了AIDL工具来处理这项工作。

AIDL (Android Interface Definition Language)用于生成可以在Android设备上两个进程之间进行进程间通信(IPC)的代码。如果在一个进程中（例如Activity）要调用另一个进程中（例如Service）对象的操作，就可以使用AIDL生成可序列化的参数。

AIDL支持的数据类型:

1. 不需要 import 声明的简单 Java 编程语言类型(int,boolean 等)

2. String, CharSequence 不需要特殊声明

3. List, Map 和 Parcelable 类型, 这些类型内所包含的数据成员也只能是简单数据类型, String 等其它支持的类型.

二、重要题答案

1、谈谈 Android 的线程间通信机制(基础)

1)Android 的线程设计模式

(1)不要阻塞 UI 线程，达到 5 秒后应用会报错。
(2)不允许让非 UI 线程修改 UI。

2)Android 线程间通信机制

使用 MessageQueue(消息队列)的数据结构存放线程间的消息。

使用 Looper 管理消息队列，将消息从 MessageQueue 取出交给 Handler.handleMessage 方法处理。

使用 Handler 的 sendMessage 等方法发送消息，使用 Handler.handleMessage 方法接收 Looper 交给的消息，并处理消息。

3)线程间通信方式包括

1)Handler，该方法可以在线程间发送/接收消息。
2)用 Handler.post 方法向 Handler 工作的线程发送一个实现了 Runnable 接口的对象。
3)用 Activity 类的 runOnUiThrad 方法向主线程发送一个实现了 Runnable 接口的对象。
4)AsyncTask 实现一个工作线程与主线程的通信。由工作线程向主线程发送消息。

4)详细说明

1)Message Queue(消息队列)：用来存放通过 Handler 发布的消息，数据结构为队列。通常附属于某一个创建它的线程，可以通过 Looper.myQueue()得到当前线程的消息队列

2)Handler：可以发布或者处理一个消息或者操作一个 Runnable，通过 Handler 发布消息，消息将只会发送到与它关联的消息队列，当然也只能处理该消息队列中的消息。这两种消息都会插在 message queue 队尾并按先进先出执行。但通过这两种方法发送的消息执行的方式略有不同：通过 sendMessage 发送的是一个 message 对象,会被 Handler 的 handleMessage()函数处理；而通过 post 方法发送的是一个 runnable 对象，则会自己执行。

3)Looper：是 Handler 和消息队列之间通讯桥梁，程序组件首先通过 Handler 把消息传递给 Looper，Looper 把消息放入队列。Looper 也把消息队列里的消息广播给所有的 Handler，Handler 接受到消息后调用 handleMessage 进行处理

4)Looper

Looper 是每条线程里的 Message Queue 的管家。Android 没有 Global 的 Message Queue，而 Android 会自动替主线程(UI 线程)建立 Message Queue，但在子线程里并没有建立 Message Queue。所以调用 Looper.getMainLooper()得到的主线程的 Looper 不为 NULL，但调用 Looper.myLooper() 得到当前线程的 Looper 就有可能为 NULL。对于子线程

使用 Looper，API Doc 提供了正确的使用方法：这个 Message 机制的大概流程：

1. 在 Looper.loop()方法运行开始后，循环地按照接收顺序取出 Message Queue 里面的非 NULL 的 Message。

2. 一开始 Message Queue 里面的 Message 都是 NULL 的。当 Handler.sendMessage(Message)到 Message Queue，该函数里面设置了那个 Message 对象的 target 属性是当前的 Handler 对象。随后 Looper 取出了那个 Message，则调用该 Message 的 target 指向的 Hander 的 dispatchMessage 函数对 Message 进行处理。在 dispatchMessage 方法里，如何处理 Message 则由用户指定，三个判断，优先级从高到低：

1) Message 里面的 Callback，一个实现了 Runnable 接口的对象，其中 run 函数做处理工作；

2) Handler 里面的 mCallback 指向的一个实现了 Callback 接口的对象，由其 handleMessage 进行处理；

3) 处理消息 Handler 对象对应的类继承并实现了其中 handleMessage 函数，通过这个实现的 handleMessage 函数处理消息。

由此可见，我们实现的 handleMessage 方法是优先级最低的！

2、Service 和 Thread 的区别

Service 是系统的组件，它由系统进程托管（servicemanager）；它们之间的通信类似于 Client 和 Server，是一种轻量级的 ipc 通信，这种通信的载体是 Binder，它是在 linux 层交换信息的一种 ipc。而 Thread 是由本应用程序托管。

1)Thread

Thread 是程序执行的最小单元，它是 CPU 分配的基本单位。可以用 Thread 来执行一些异步操作。

2)Service

Service 是 Android 的一种机制，当它运行的时候如果是 Local Service，那么对应的 Service 是运行在主进程的 main 线程。如：onCreate，onStart 这些函数在被系统调用的时候都是在主进程的 main 线程上运行。如果是 Remote Service，那么对应的 Service 则是运行在独立进程的 main 线程上。

既然这样，那么我们为什么要用 Service 呢？其实这跟 Android 的系统机制有关，我们先拿 Thread 来说。Thread 的运行是独立于 Activity 的，也就是说当一个 Activity 被 finish 之后，如果你没有主动停止 Thread 或者 Thread 里的 run 方法没有执行完毕的话，Thread 会一直执行。因此这里会出现一个问题：当 Activity 被 finish 之后，你不再持有该 Thread 的引用。另一方面，你没有办法在不同的 Activity 中对同一 Thread 进行控制。

举个例子：如果你的 Thread 需要不停地隔一段时间就要连接服务器做某种同步的话，该 Thread 需要在 Activity 没有 start 的时候也在运行。这个时候当你 start 一个 Activity 就没有办法在该 Activity 里面控制之前创建的 Thread。因此你便需要创建并启动一个 Service ，在 Service 里面创建、运行并控制该 Thread，这样便解决了该问题（因为任何 Activity 都可以控制同一 Service，而系统也只会创建一个对应 Service 的

实例）。

因此你可以把 Service 想象成一种消息服务，而你可以在任何有 Context 的地方调用 startService、stopService、bindService， unbindService，来控制它，你也可以在 Service 里注册 BroadcastReceiver，在其它地方通过发送 broadcast 来控制它，这些都是 Thread 做不到的。

3、简述触摸事件分发机制

如下图所示：

1)Activity.dispatchTouchEvent()将触摸事件分发给 ViewGroup.dispatchTouchEvent()

2) ViewGroup.dispatchTouchEvent()返回 false，则事件传递给 Activity.onTouchEvent()

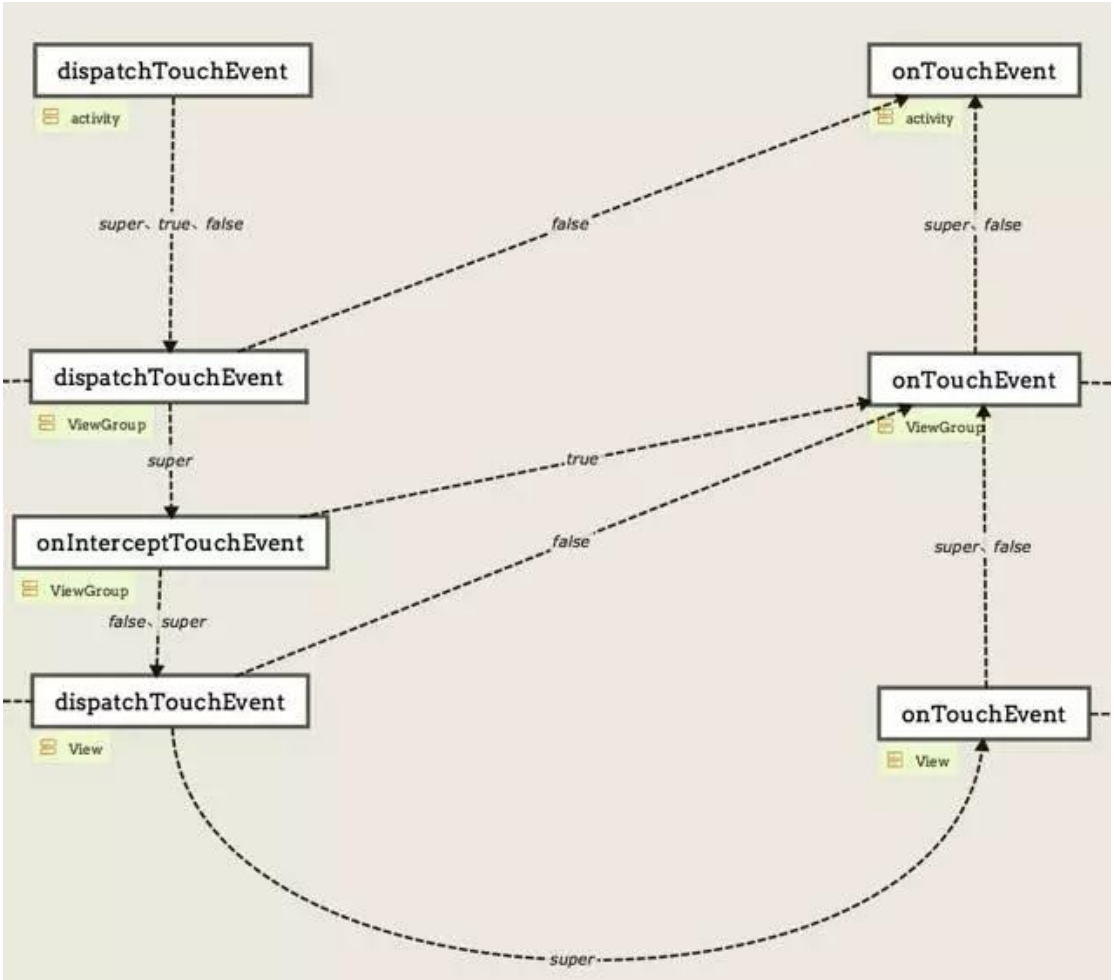
3) ViewGroup.dispatchTouchEvent()返回 true，则事件传递给 ViewGroup.onIterceptTouchEvent()

4) ViewGroup.onIterceptTouchEvent()返回 true，则拦截事件，事件交给 ViewGroup.onTouchEvent()

5) ViewGroup.onIterceptTouchEvent()返回 false，则事件传递给下层的 ViewGroup 或 View 的 dispatchTouchEvent

6)View.dispatchTouchEvent()将事件传递给 View.onTouchEvent()

7) View.onTouchEvent()返回 true，则后续事件继续传递给本 View.onTouchEvent()，若返回 super 或 false，则后续事件传递给父容器.onTouchEvent()，依次类推，直至上传给 Activity.onTouchEvent()

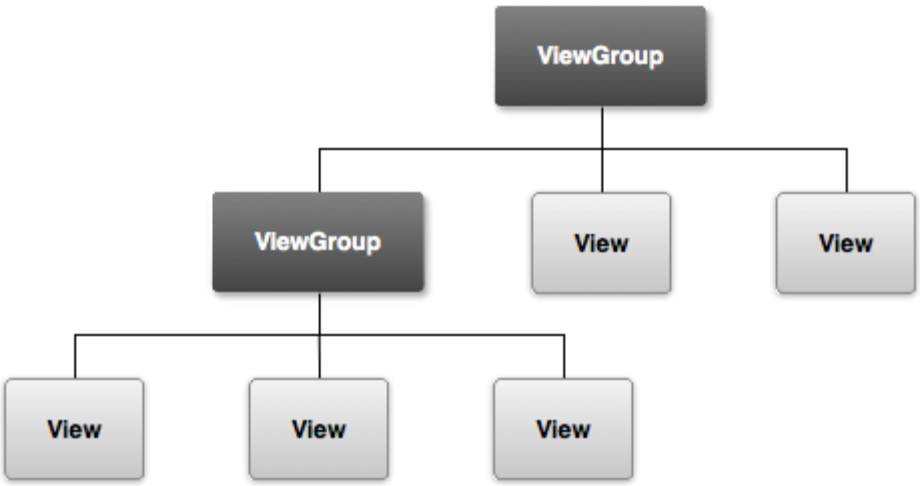


更详细请参考:《图解 Android 事件分发机制》

4、谈谈 View 绘制原理

答:

Android 系统的视图结构的设计采用了组合模式，即 View 作为所有图形的基类，Viewgroup 对 View 继承扩展为视图容器类，由此就得到了视图部分的基本结构--树形结构



View 树的绘制是一个递归的过程，从 ViewGroup 一直向下遍历，直到所有的子 view 都完成绘制，这一切的源头是在 View 树的源头——ViewRoot！，ViewRoot 中包含了窗口的总容器 DecorView，ViewRoot 中的 performTraversal()方法会依次调用 decorView 的 measure、layout、draw 方法，从而完成 view 树的绘制。

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。

4、ViewGroup 中的扩展操作：

Viewgroup 是一个抽象类。

1)对子视图的 measure 过程

(1)measureChildren()，内部使用一个 for 循环对子视图进行遍历，分别调用子视图

的 measure()方法；

(2)measureChild()，为指定的子视图 measure，会被 measureChildren 调用；

(3)measureChildWithMargins()，为指定子视图考虑了 margin 和 padding 的 measure；

以上三个方法是 ViewGroup 提供的 3 个对子 view 进行测量的参考方法，设计者需要在实际中首先覆写 onMeasure()，之后再对子 view 进行遍历 measure，这时候就可以使用以上三个方法，当然也可以自定义方法进行遍历。

2)对子视图的 layout 过程

在 ViewGroup 中 onLayout()被定义为 abstract 类型，也就是具体的容器必须实现此方法来安排子视图的布局位置，实现中主要考虑的是视图的大小及视图间的相对位置关系，如 gravity、layout_gravity。

3)对子视图的 draw 过程

(1)dispatchDraw()，该方法用于对子视图进行遍历然后分别让子视图分别 draw，方法内部会首先处理布局动画（也就是说布局动画是在这里处理的），如果有布局动画则会为每个子视图产生一个绘制时间，之后再有一个 for 循环对子视图进行遍历，来调用子视图的 draw 方法（实际为下边的 drawChild()）；

(2)drawChild()，该方法用于具体调用子视图的 draw 方法，内部首先会处理视图动画（也就是说视图动画是在这里处理的），之后调用子视图的 draw()。

从上面分析可以看出自定义 viewGroup 的时候需要最少覆写 onMeasure()和 onLayout()方法，其中 onMeasure 方法中可以直接调用 measureChildren 等已有的方法，而 onLayout 方法就需要设计者进行完整的定义；一般不需要覆写。

5、invalidate()方法

invalidate()方法会导致 View 树的重新绘制，而且 view 中的状态标志 mPrivateFlags 中有一个关于当前视图是否需要重绘的标志位 DRAWN，也就是说只有标志位 DRAWN 置位的视图才需要进行重绘。当视图调用 invalidate()方法时，首先会将当前视图的 DRAWN 标志置位，之后有一个循环调用 parent.invalidateChildinParent()，这样会导致从当前视图依次向上遍历直到根视图 ViewRoot，这个过程会将需要重绘的视图标记 DRAWN 置位，之后 ViewRoot 调用 performTraversals()方法，完成视图的绘制过程。

5、Activity 在屏幕旋转时的生命周期

答：

1)不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次。

2)设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次。

3)设置Activity的android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法。

6、Activity 的启动模式有哪些？是什么含义？

答：在android里，有4种activity的启动模式，分别为：

“standard”（默认）

“singleTop”

“singleTask”

“singleInstance”

当应用运行起来后就会开启一条线程，线程中会运行一个任务栈，当Activity实例创建后就会放入任务栈中。Activity启动模式的设置在AndroidManifest.xml文件中，通过配置Activity的属性android:launchMode=""设置。

1. Standared 模式（默认）

默认创建的Activity的模式，这种模式的特点：创建了Activity实例，一旦激活该Activity，则会向任务栈中加入新创建的实例，退出Activity则会在任务栈中销毁该实例。

2. SingleTop 模式

这种模式会考虑当前要激活的Activity实例在任务栈中是否正处于栈顶，如果处于栈顶则无需重新创建新的实例，会重用已存在的实例，否则会在任务栈中创建新的实例。

3. SingleTask 模式

如果任务栈中存在该模式的Activity实例，则把栈中该实例以上的Activity实例全部移除，调用该实例的newInstance()方法重用该Activity，使该实例处於栈顶位置，否则就重新创建一个新的Activity实例。

4. SingleInstance 模式

当该模式Activity实例在任务栈中创建后，只要该实例还在任务栈中，即只要激活的是该类型的Activity，都会通过调用实例的newInstance()方法重用该Activity，此时使用的都是同一个Activity实例，它都会处于任务栈的栈顶。此模式一般用于加载较慢的，比较耗性能且不需要每次都重新创建的Activity。

7、跟 Activity 和 Task 有关的 Intent 启动方式有哪些？其含义？

核心的 Intent Flag 有：

FLAG_ACTIVITY_NEW_TASK
FLAG_ACTIVITY_CLEAR_TOP
FLAG_ACTIVITY_SINGLE_TOP
FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

FLAG_ACTIVITY_NEW_TASK

将目标 Activity 启动到新的任务栈，相当于 Activity 的 Launch mode 的 singleInstance 模式。

在Service、Broadcast中启动Activity时，必须使用FLAG_ACTIVITY_NEW_TASK标志，否则会出现异常。

使用了FLAG_ACTIVITY_NEW_TASK标志，若activity与已存在的task有相同affinity，则该Activity会启动在这个Task的中。

这个标志一般用于呈现“启动”类型的行为：它们提供用户一系列可以单独完成的事情，与启动它的Activity完全无关。

FLAG_ACTIVITY_CLEAR_TOP

如果设置，并且这个Activity已经在当前的Task中运行，因此，不再是重新启动一个这个Activity的实例，而是在这个Activity上方的所有Activity都将关闭，然后这个Intent会作为一个新的Intent投递到老的Activity（现在位于顶端）中。

FLAG_ACTIVITY_SINGLE_TOP

如果设置，并且这个Activity已经在当前的Task中运行，因此，不再是重新启动一个这个Activity的实例，而是在这个Activity上方的所有Activity都将关闭，然后Intent传递到该Activity的onNewInent方法（现在位于顶端）中。

FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

如果设置该属性，并且这个activity在一个新的task中正在被启动或者被带到一个已经存在的task的顶部，这时这个activity将会被作为这个task的首个页面加载。这将会导致拥有这个应用的affinities的task处于一个合适的状态(移动activity到这个task或者activity从中移出)，或者简单的重置这个task到它的初始状态。

8、谈谈 ListView 的优化

答：

1、如果自定义适配器，那么在 getView 方法中要考虑方法传进来的参数 convertView 是否为 null，如果为 null 就创建 convertView 并返回，如果不为 null 则直接使用。在这个方法中尽可能少创建 view。

2、给 convertView 设置 tag（setTag（）），传入一个 viewHolder 对象，用于缓存要显示的数据，可以达到图像数据异步加载的效果。

3、如果 listview 需要显示的 item 很多，就要考虑分页加载。比如一共要显示 100 条或者更多的时候，我们可以考虑先加载 20 条，等用户拉到列表底部的时候再去加载接下来的 20 条。

扩展：

Android 提供的 RecyclerView 具有 ViewHolder 并且自动封装了 ViewHolder。

(1)RecylerView 封装了 Viewholder 的回收复用，即 RecylerView 标准化了 ViewHolder，编写 Adapter 面向的是 ViewHolder 而不再是 View 了，复用的逻辑被封装了，写起来更加简单。

(2)提供了一种插拔式的体验，高度的解耦，异常的灵活，针对一个 Item 的显示 RecylerView 专门抽取出了相应的类，来控制 Item 的显示，使其的扩展性非常强。例如：你想控制横向或者纵向滑动列表效果可以通过 LinearLayoutManager 这个类来进行控制(与 GridView 效果对应的是 GridLayoutManager,与瀑布流对应的还有 StaggeredGridLayoutManager 等)，也就是说 RecylerView 不再拘泥于 ListView 的线性展示方式，它也可以实现 GridView 的

效果等多种效果。你想控制 Item 的分隔线，可以通过继承 RecylerView 的 ItemDecoration 这个类，然后针对自己的业务需求去抒写代码。

(3)可以控制 Item 增删的动画。

(4)ViewHolder 可扩展

(5)RecyclerView 象 ListView 一样能回收、复用 item，并能方便地显示多种布局类型的 item。

(6)RecyclerView.Adapter.onCreateViewHolder() 相当于 BaseAdapter.getView() 中创建 ViewHolder 对象的代码块。创建的 ViewHolder 系统将自动缓存并复用。

(7)RecyclerView.Adpter.onBindViewHolder() 相当于 BaseAdapter.getView 中显示 ViewHolder 中的 view 的代码块。

9、谈谈 RecyclerView 多种布局的实现

1、为每一个布局类型创建一个 ViewHolder，父类是 RecyclerView.ViewHolder

2、在适配器中定义所有布局的类型常量 int 类型。

3、在 getItemType 方法中根据 position 返回不同的布局类型常量

4、在 onCreateViewHolder 方法中根据 viewType 创建相应类型的布局。

5、在 onBindViewHolder 中根据 getItemType(position)的值在不同的 ViewHolder 中显示数据。

三、提高题答案

1、简述 Android 消息传递的方式都有哪些？

a) Activity 内部可以通过 Handler、Message 实现线程之间的消息传递。
b) 应用之间、组件之间可以通过 Intent 传递消息。
c) 应用或组件之间可以通过广播传递消息。
d) 应用或组件之间可通过通知传递消息，通知的消息在通知栏显示。
e) 组件和 Service 之间可以通过绑定 Service 传递消息。

2、简述广播与通过 Intent 向组件传递消息之间的异同点。

1)都是通过 Intent 启动目标组件和传递数据。

2)Intent 启动组件可以在同一应用内也可以跨应用启动目标组件。广播还可以在组件和非组件之间传递消息。

3)广播均是通过隐式意图启动目标组件。

3、简述通知与广播之间在消息传递方面的不同。

1)通知的消息直接显示在通知栏；广播的消息不显示，而是在代码中传递。

2)通知可以跨应用、在同一应用内各组件之间传递消息。广播可以在组件甚至是非组件之间传递消息。

4、如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

方法(1)：将app中的所有task栈底的Activity的launchmode设置为single task，当需要关闭所有activity时，启动每个task栈底的activity，并传送一个关闭该activity的标识。则task的所有activity给清除，然后栈底的activity再调用finish关闭自身。

方法(2)将所有的Activity放在一个List集合中，关闭app时，将该集合所有的元素执行finish方法。

5、如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

答：重写onSaveInstanceState()方法，在此方法中保存需要保存的数据，该方法将会在activity被回收之前调用。通过重写onRestoreInstanceState()方法可以从中提取保存好的数据

6、View 如何刷新？简述什么是双缓冲？

android中实现view的更新有两个方法，一个是invalidate，另一个是postInvalidate，其中前者是在UI线程自身中使用，而后者在非UI线程中使用。

双缓冲

闪烁是图形编程的一个常见问题。当进行复杂的绘制操作时会导致呈现的图像闪烁或具有其他不可接受的外观。双缓冲的使用解决这些问题。双缓冲使用内存缓冲区来解决由多重绘制操作造成的闪烁问题。当使用双缓冲时，首先在内存缓冲区里完成所有绘制操作，而不是在屏幕上直接进行绘图。当所有绘制操作完成后，把内存缓冲区完成的图像直接复制到屏幕。因为在屏幕上只执行一个图形操作，所以消除了由复杂绘制操作造成的图

像闪烁问题。
在android中实现双缓冲,可以使用一个后台画布backcanvas,先把所有绘制操作都在这上面进行。等图画好了，然后在把backcanvas拷贝到
与屏幕关联的canvas上去,如下:

```
Bitmap bitmapBase = new Bitmap()
Canvas backcanvas = new Canvas(bitmapBase)
backcanvas.draw()...//画图
Canvas c = lockCanvas(null);
c.drawbitmap(bitmapBase);//把已经画好的图像输出到屏幕上
unlock(c)....
```