## 1：Activity生命周期?

相信不少朋友也已经看过这个流程图了，也基本了解了Activity生命周期的几个过程，我们就来说一说这几个过程。

1.启动Activity：系统会先调用onCreate方法，然后调用onStart方法，最后调用onResume，Activity进入运行状态。

2.当前Activity被其他Activity覆盖其上或被锁屏：系统会调用onPause方法，暂停当前Activity的执行。

3.当前Activity由被覆盖状态回到前台或解锁屏：系统会调用onResume方法，再次进入运行状态。（被动）

4.当前Activity转到新的Activity界面或

按Home键回到主屏，自身退居后台：系统会先调用onPause方法，然后调用onStop方法，进入停滞状态。（主动）

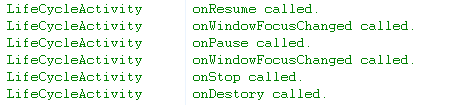
5.用户后退回到此Activity：系统会先调用onRestart方法，然后调用onStart方法，最后调用onResume方法，再次进入运行状态。

6.当前Activity处于被覆盖状态或者后台不可见状态，即第2步和第4步，系统内存不足，杀死当前Activity，而后用户退回当前Activity：再次调用onCreate方法、onStart方法、onResume方法，进入运行状态。

7.用户退出当前Activity：系统先调用onPause方法，然后调用onStop方法，最后调用onDestory方法，结束当前Activity。

ps:

1.onWindowFocusChanged方法：在Activity窗口获得或失去焦点时被调用，例如创建时首次呈现在用户面前；当前Activity被其他Activity覆盖；当前Activity转到其他Activity或按Home键回到主屏，自身退居后台；用户退出当前Activity。以上几种情况都会调用onWindowFocusChanged，并且当Activity被创建时是在onResume之后被调用，当Activity被覆盖或者退居后台或者当前Activity退出时，它是在onPause之后被调用，如图所示：



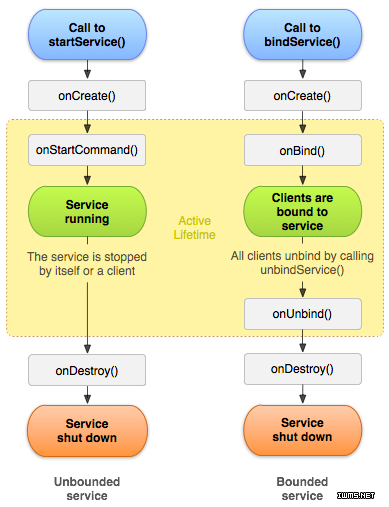
2.onSaveInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，此方法会被调用；(2)在用户改变屏幕方向时，此方法会被调用；(3)在当前Activity跳转到其他Activity或者按Home键回到主屏，自身退居后台时，此方法会被调用。第一种情况我们无法保证什么时候发生，系统根据资源紧张程度去调度；第二种是屏幕翻转方向时，系统先销毁当前的Activity，然后再重建一个新的，调用此方法时，我们可以保存一些临时数据；第三种情况系统调用此方法是为了保存当前窗口各个View组件的状态。onSaveInstanceState的调用顺序是在onPause之前。

3.onRestoreInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，然后用户又回到了此Activity，此方法会被调用；(2)在用户改变屏幕方向时，重建的过程中，此方法会被调用。我们可以重写此方法，以便可以恢复一些临时数据。onRestoreInstanceState的调用顺序是在onStart之后。

<http://blog.csdn.net/liuhe688/article/details/6733407>

## 2：service生命周期?

这里要注意service有两种启动方式，startService()和bindService()



Paste\_Image.png

## 3：如何理解Activity，View，Window三者之间的关系?

这个问题真的很不好回答。所以这里先来个算是比较恰当的比喻来形容下它们的关系吧。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。

1：Activity构造的时候会初始化一个Window，准确的说是PhoneWindow。

2：这个PhoneWindow有一个“ViewRoot”，这个“ViewRoot”是一个View或者说ViewGroup，是最初始的根视图。

3：“ViewRoot”通过addView方法来一个个的添加View。比如TextView，Button等

4：这些View的事件监听，是由WindowManagerService来接受消息，并且回调Activity函数。比如onClickListener，onKeyDown等。

## 4：Activity的几种LaunchMode及使用场景

standard 模式

这是默认模式，每次激活Activity时都会创建Activity实例，并放入任务栈中。使用场景：大多数Activity。

singleTop 模式

如果在任务的栈顶正好存在该Activity的实例，就重用该实例( 会调用实例的 onNewIntent() )，否则就会创建新的实例并放入栈顶，即使栈中已经存在该Activity的实例，只要不在栈顶，都会创建新的实例。使用场景如新闻类或者[阅读](http://book.2cto.com/)类App的内容页面。

singleTask 模式

如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的 onNewIntent() )。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移出栈。如果栈中不存在该实例，将会创建新的实例放入栈中。使用场景如[浏览器](http://www.2cto.com/os/liulanqi/)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 模式

在一个新栈中创建该Activity的实例，并让多个应用共享该栈中的该Activity实例。一旦该模式的Activity实例已经存在于某个栈中，任何应用再激活该Activity时都会重用该栈中的实例( 会调用实例的 onNewIntent() )。其效果相当于多个应用共享一个应用，不管谁激活该 Activity 都会进入同一个应用中。使用场景如闹铃提醒，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。

## 5：View的绘制流程

measure过程

layout过程

draw过程

<http://gold.xitu.io/entry/56af0ba0c24aa800547b60ea>

<http://blog.csdn.net/yanbober/article/details/46128379/>

## 6：Touch事件的传递机制

public boolean dispatchTouchEvent(MotionEventev); //用来分派event

public boolean onInterceptTouchEvent(MotionEventev);//用来拦截event

public boolean onTouchEvent(MotionEventev);//用来处理event

其中Activity和View控件(TextView)拥有分派和处理事件方法，View容器(LinearLayout)具有分派，拦截，处理事件方法。这里也有个比喻：领导都会把任务向下分派，一旦下面的人把事情做不好，就不会再把后续的任务交给下面的人来做了，只能自己亲自做，如果自己也做不了，就只能告诉上级不能完成任务，上级又会重复他的过程。另外，领导都有权利拦截任务，对下级隐瞒该任务，而直接自己去做，如果做不成，也只能向上级报告不能完成任务。

<http://blog.csdn.net/morgan_xww/article/details/9372285>

## 7：Android中的几种动画

曾被问到Android中有几种动画，这个问题也好难回答。Android3.0之前有2种，3.0后有3种。

FrameAnimation(逐帧动画)：将多张图片组合起来进行播放，类似于早期电影的工作原理，很多App的loading是采用这种方式。

TweenAnimation(补间动画)：是对某个View进行一系列的动画的操作，包括淡入淡出(Alpha)，缩放(Scale)，平移(Translate)，旋转(Rotate)四种模式。

PropertyAnimation(属性动画)：属性动画不再仅仅是一种视觉效果了，而是一种不断地对值进行操作的机制，并将值赋到指定对象的指定属性上，可以是任意对象的任意属性。

<http://blog.csdn.net/yanbober/article/details/46481171>

## 8：Android中跨进程通讯有几种方式

1：访问其他应用程序的Activity

如调用[系统](http://www.2cto.com/os/)通话应用

IntentcallIntent=newIntent(Intent.ACTION\_CALL,Uri.parse("tel:12345678");

startActivity(callIntent);

2：Content Provider

如访问系统相册

3：广播(Broadcast)

如显示系统时间

4：AIDL服务

## 9：AIDL理解

<http://bbs.51cto.com/thread-1086040-1.html>

<http://blog.csdn.net/u011974987/article/details/51243539>

跨进程通信只能通过server来完成，AIDL也就是利用的binder通信而已

## 10：Handler的原理

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

<http://blog.csdn.net/guolin_blog/article/details/9991569>

异步消息处理线程

## 11：Binder机制原理

<http://blog.csdn.net/boyupeng/article/details/47011383>

Binder是Android系统进程间通信（IPC）方式之一

Binder使用Client-Server通信方式，安全性好，简单高效，再加上其面向对象的设计思想，独特的接收缓存管理和线程池管理方式，成为Android进程间通信的中流砥柱。

## 12：热修复的原理

<https://mp.weixin.qq.com/s?__biz=MzI1MTA1MzM2Nw==&mid=400118620&idx=1&sn=b4fdd5055731290eef12ad0d17f39d4a&scene=1&srcid=1106Imu9ZgwybID13e7y2nEi#wechat_redirect>

<http://blog.csdn.net/lmj623565791/article/details/49883661>

## 13：设计一套图片异步加载缓存方案

<http://www.cnblogs.com/zyw-205520/p/4997863.html>

<http://blog.csdn.net/boyupeng/article/details/47127605>

内存LruCache+磁盘DiskLruCache+网络Volley

## 14：Android内存泄露及管理

<http://gold.xitu.io/entry/56d64b9e816dfa005943a55c>

<https://news.realm.io/cn/news/droidcon-farber-improving-android-app-performance/>

## 15：Activity与Fragment通信

<http://gold.xitu.io/entry/56a87b2b2e958a0051906227>

## 16：Fragment的那些坑

<http://www.jianshu.com/p/d9143a92ad94>

<http://www.jianshu.com/p/fd71d65f0ec6>

<http://www.jianshu.com/p/38f7994faa6b>

## 17：Android UI适配

<http://www.jianshu.com/p/145fc61011cd>

<http://www.jianshu.com/p/ec5a1a30694b>

<http://blog.csdn.net/lmj623565791/article/details/49990941>

<http://blog.csdn.net/zhengjingle/article/details/51742839>

## 18：Http Https

<http://www.jianshu.com/p/93fdebe5fef1>

## 19：网络请求优化

<http://www.jianshu.com/p/3141d4e46240>

## 20：[数据库](http://www.2cto.com/database/)优化

<http://www.jianshu.com/p/3b4452fc1bbd>

## 21：图片优化

<http://www.jianshu.com/p/5bb8c01e2bc7>

## 22：HybridJAVA 与JS交互

<http://droidyue.com/blog/2014/09/20/interaction-between-java-and-javascript-in-android/>

## 23：单例

<http://www.jianshu.com/p/a956024629cb>

## 24：Java GC原理

<http://blog.csdn.net/ithomer/article/details/6252552>

## 25：ANR

<http://www.jianshu.com/p/124f3b75e164>

## 26：Volley

<http://www.jianshu.com/p/9e17727f31a1>

## 27：JAVA注解反射原理

<http://www.jianshu.com/p/3968ffabdf9d>

<http://www.cnblogs.com/cr330326/p/5695474.html>

## 28：算法

<http://www.jianshu.com/p/ae97c3ceea8d>

## 29：框架模式

<http://gold.xitu.io/entry/56ebb4ad5bbb50004c440972>

## 30：RxJava

<http://www.jianshu.com/p/0cd258eecf60>

<http://www.jianshu.com/p/c95e29854cb1>

RxAndroid是RxJava的一个针对Android平台的扩展，主要用于 Android 开发

特点：响应式编程、异步、链式调用

## 31: ****Fragment的生命周期****



<http://www.cnblogs.com/smyhvae/p/3983234.html>

## 32: java设计模式

设计模式分为三种类型，共23种：

**创建型模式**：[单例模式](https://github.com/youlookwhat/DesignPattern#3-%E5%8D%95%E4%BE%8B%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F)、抽象工厂模式、建造者模式、[工厂模式](https://github.com/youlookwhat/DesignPattern#2-%E5%B7%A5%E5%8E%82%E6%A8%A1%E5%BC%8F)、原型模式。

**结构型模式**：[适配器模式](https://github.com/youlookwhat/DesignPattern#5-%E9%80%82%E9%85%8D%E5%99%A8%E6%A8%A1%E5%BC%8F)、桥接模式、[装饰模式](https://github.com/youlookwhat/DesignPattern#7-%E8%A3%85%E9%A5%B0%E8%80%85%E6%A8%A1%E5%BC%8F)、组合模式、[外观模式](https://github.com/youlookwhat/DesignPattern#8-%E5%A4%96%E8%A7%82%E6%A8%A1%E5%BC%8F)、享元模式、代理模式。

**行为型模式**：[模版方法模式](https://github.com/youlookwhat/DesignPattern#9-%E6%A8%A1%E6%9D%BF%E6%96%B9%E6%B3%95%E6%A8%A1%E5%BC%8F)、[命令模式](https://github.com/youlookwhat/DesignPattern#6-%E5%91%BD%E4%BB%A4%E6%A8%A1%E5%BC%8F)、迭代器模式、[观察者模式](https://github.com/youlookwhat/DesignPattern#1-%E8%A7%82%E5%AF%9F%E8%80%85%E6%A8%A1%E5%BC%8F)、中介者模式、备忘录模式、解释器模式、[状态模式](https://github.com/youlookwhat/DesignPattern#10-%E7%8A%B6%E6%80%81%E6%A8%A1%E5%BC%8F)、[策略模式](https://github.com/youlookwhat/DesignPattern#4-%E7%AD%96%E7%95%A5%E6%A8%A1%E5%BC%8F)、职责链模式(责任链模式)、访问者模式。

<http://www.jianshu.com/p/61b67ca754a3>

## 33: list、map、set的区别和使用

<http://blog.csdn.net/abbuggy/article/details/7720666>