1. PhoneWindow 实例是在哪个类哪个方法中实例化的

ActivityThread执行main方法启动入口，实例化Activity，ActivityThread调用performLaunchActivity，具体由Instrumentation.newActivity实例化出activity，然后调用activity的attach方法new PhoneWindow，初始化activity中一些参数

2.Application/ Activity启动流程：

0812补充：

Launcher启动activity：

1. 调用startActivityForResult

2. Instrumentation启动activity，获取AMS服务，记录activity，launcherActivity进入onpause状态

3. AMS查询是否存在新的activity的进程，如果已经有该activity在后台，唤醒，否则zygote fork出一个子进程，调用activityThread的main方法，Application启动流程是从ActivityThread main方法开始的，所在的mian线程启动一个

1.Looper.prepareMainLoop

2.new ActivityThread，（会创建一个ApplicationThread）

首先在ActivityThread的主线程，生成一个ApplicationThread对象（本地代理），这个对象式能够对外提供远程服务的，换句话说别的进程可以通过这个对象的本地代理与我这个ActivityThread的主线程（ApplicationThread对象）通信

3 AMS调用attachApplication，将此activityThread的本地代理ApplicationThread交给AMS管理

4 AMS通过binder通信调用本地代理ApplicationThread接口方法scheduleLaunchActivity通知activityThread的H对象handleMessage执行ActivityThread#performLauchActivity创建activity

3.2.1 后续所有操作都是由AMS通过binder使用本地代理ApplicationThread发送消息给activityThread#H（Handler对象），H通过handleMessage调用ActivityThread的方法

5 由Instrumentation.newActivity实例化出activity

6 然后调用activity的attach方法初始化activity变量

4.Looper.loop开启消息队列

Activity启动Activity：

1. 调用startActivityForResult

2. Instrumentation启动activity，获取AMS服务，记录activity，launcherActivity进入onpause状态

3. AMS查询是否存在新的activity的进程，获取 ApplicationThread对象（本地代理）然后AMS通过binder通信调用本地代理ApplicationThread接口方法scheduleLaunchActivity通知activityThread的H对象handleMessage执行ActivityThread#performLauchActivity创建activity，回调onCreate,onstart,onResume

4. ActivityRecord记录的启动的activity进入onStop状态

4 为什么主线程不会因为looper.loop卡死

Looper.loop方法

1.for（）死循环，获取MessageQueue里面的msg

2.next启动for（）死循环等待消息，如果有消息就出队，没有就一直等待，使用linux的epoll模型阻塞，但是不会卡住

主要原因有2个

1. epoll模型   
当没有消息的时候会epoll.wait，等待句柄写的时候再唤醒，这个时候其实是阻塞的。释放cpu，进入休眠，不会占用CPU

2. 所有的ui操作都通过handler来发消息操作。   
比如屏幕刷新16ms一个消息，你的各种点击事件，所以就会有句柄写操作，唤醒上文的wait操作，所以不会被卡死了。

5. selector实现原理

Selector主要是一个statueListDrawable，对不同状态返回不同的drawable对象

# 6. Android View动画

启动动画的时候将View绑定一个Animation，调用invalidate请求重绘，每次绘图时View所在的ViewGroup中的dispathDraw，最终执行的view的draw方法，然后调用applyTransformation执行绘制，判断是否绘制完成，没有将继续调用invalidate

综上，我们稍微整理一下：

1. 首先，当调用了 View.startAnimation() 时动画并没有马上就执行，而是通过 invalidate() 层层通知到 ViewRootImpl 发起一次遍历 View 树的请求，而这次请求会等到接收到最近一帧到了的信号时才去发起遍历 View 树绘制操作。
2. 从 DecorView 开始遍历，绘制流程在遍历时会调用到 View 的 draw() 方法，当该方法被调用时，如果 View 有绑定动画，那么会去调用applyLegacyAnimation()，这个方法是专门用来处理动画相关逻辑的。
3. 在 applyLegacyAnimation() 这个方法里，如果动画还没有执行过初始化，先调用动画的初始化方法 initialized()，同时调用 onAnimationStart() 通知动画开始了，然后调用 getTransformation() 来根据当前时间计算动画进度，紧接着调用 applyTransformation() 并传入动画进度来应用动画。
4. getTransformation() 这个方法有返回值，如果动画还没结束会返回 true，动画已经结束或者被取消了返回 false。所以 applyLegacyAnimation() 会根据 getTransformation() 的返回值来决定是否通知 ViewRootImpl 再发起一次遍历请求，返回值是 true 表示动画没结束，那么就去通知 ViewRootImpl 再次发起一次遍历请求。然后当下一帧到来时，再从 DecorView 开始遍历 View 树绘制，重复上面的步骤，这样直到动画结束。
5. 有一点需要注意，动画是在每一帧的绘制流程里被执行，所以动画并不是单独执行的，也就是说，如果这一帧里有一些 View 需要重绘，那么这些工作同样是在这一帧里的这次遍历 View 树的过程中完成的。每一帧只会发起一次 perfromTraversals() 操作。

实际上，**补间动画仅仅是对 View 在视觉效果上做了移动、缩放、旋转和淡入淡出的效果，其实并没有真正改变 View 的属性**

# 7. Android Handler同步屏障机制

# Handler中的Message可以分为两类：同步消息、异步消息

一般情况下这两种消息的处理方式没什么区别，只有在设置了同步屏障时才会出现差异

通常我们使用Handler发消息时，这些消息都是同步消息，如果我们想发送异步消息，那么在创建Handler时使用以下构造函数中的其中一种(async传true)，那么这个handler将异步发送消息

public Handler(boolean async);

public Handler(Callback callback, boolean async);

public Handler(Looper looper, Callback callback, boolean async);

设置同步屏障：

使用MessageQueue.postSyncBarrier函数，返回的值用于解除该屏障

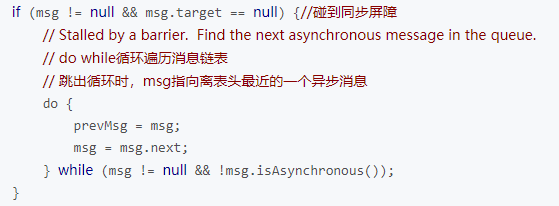
接触同步屏障：

使用removeSyncBarrier(int token) 函数

从代码层面上来讲，同步屏障就是一个Message，一个target字段为空的Message，该方法用于MessageQueue，多个Handler可能发送不同同步/异步消息，那么将忽略同步屏障后面的同步消息，优先取用异步消息

## 同步屏障的工作原理

同步屏障只在Looper死循环获取待处理消息时才会起作用，也就是说同步屏障在MessageQueue.next函数中发挥着作用



**同步屏障为Handler消息机制增加了一种简单的优先级机制，异步消息的优先级要高于同步消息**