1、说下 Activity 生命周期 ？

- 参考解答：在正常情况下，Activity 的常用生命周期就只有如下 7 个
  - onCreate()：表示 Activity **正在被创建**，常用来**初始化工作**，比如调用 setContentView 加载界面布局资源，初始化 Activity 所需数据等；
  - onRestart()：表示 Activity **正在重新启动**，一般情况下，当前 Acitivty 从不可见重新变为可见时，OnRestart 就会被调用；
  - onStart()：表示 Activity **正在被启动**，此时 Activity **可见但不在前台**，还处于后台，无法与用户交互；
  - onResume()：表示 Activity **获得焦点**，此时 Activity **可见且在前台**并开始活动，这是与 onStart 的区别所在；
  - onPause()：表示 Activity **正在停止**，此时可做一些**存储数据、停止动画**等工作，但是不能太耗时，因为这会影响到新 Activity 的显示，onPause 必须先执行完，新 Activity 的 onResume 才会执行；
  - onStop()：表示 Activity **即将停止**，可以做一些稍微重量级的回收工作，比如注销广播接收器、关闭网络连接等，同样不能太耗时；
  - onDestroy()：表示 Activity **即将被销毁**，这是 Activity 生命周期中的最后一个回调，常做**回收工作、资源释放**；
- 延伸：从**整个生命周期**来看，onCreate 和 onDestroy 是配对的，分别标识着 Activity 的创建和销毁，并且只可能有**一次调用**； 从 Activity **是否可见**来说，onStart 和 onStop 是配对的，这两个方法可能被**调用多次**； 从 Activity **是否在前台**来说，onResume 和 onPause 是配对的，这两个方法可能被**调用多次**； 除了这种区别，在实际使用中没有其他明显区别；

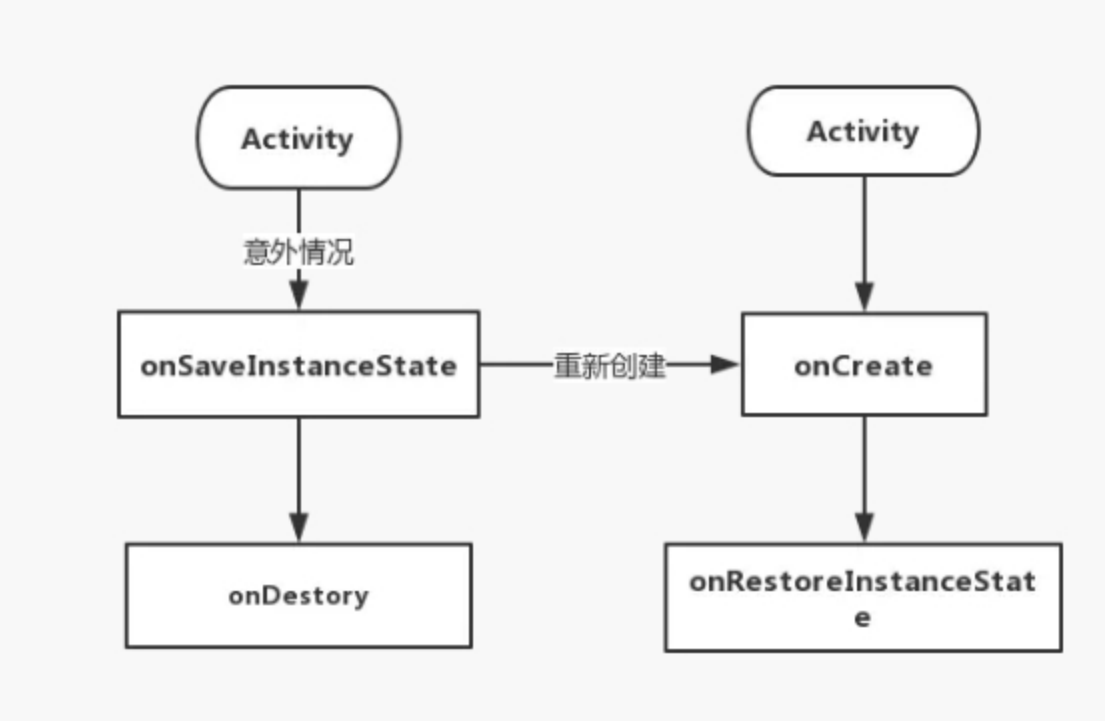
2、Activity A 启动另一个 Activity B 会调用哪些方法？如果 B 是透明主题的又或则是个 DialogActivity 呢 ？

- 参考解答：Activity A 启动另一个 Activity B，回调如下
  - Activity A 的 onPause() → Activity B 的 onCreate() → onStart() → onResume() → Activity A 的 onStop()；
  - 如果 B 是透明主题又或则是个 DialogActivity，则不会回调 A 的 onStop；

3、说下 onSaveInstanceState()方法的作用 ？ 何时会被调用？

- 参考解答：发生条件：异常情况下（**系统配置发生改变时导致 Activity 被杀死并重新创建、资源内存不足导致低优先级的 Activity 被杀死**）

- 系统会调用 onSaveInstanceState 来保存当前 Activity 的状态，此方法调用在 onStop 之前，与 onPause 没有既定的时序关系；
- 当 Activity 被重建后，系统会调用 onRestoreInstanceState，并且把 onSave(简称)方法所保存的 Bundle 对象**同时传参**给 onRestore(简称)和 onCreate()，因此可以通过这两个方法判断 Activity **是否被重建**，调用在 onStart 之后；



4、说下 Activity 的四种启动模式、应用场景 ？

- 参考回答：
  - **standard 标准模式**：每次启动一个 Activity 都会重新创建一个新的实例，不管这个实例是否已经存在，此模式的 Activity 默认会进入启动它的 Activity 所属的任务栈中；
  - **singleTop 栈顶复用模式**：如果新 Activity 已经位于任务栈的栈顶，那么此 Activity 不会被重新创建，同时会回调 **onNewIntent** 方法，如果新 Activity 实例已经存在但不在栈顶，那么 Activity 依然会被重新创建；
  - **singleTask 栈内复用模式**：只要 Activity 在一个任务栈中存在，那么多次启动此 Activity 都不会重新创建实例，并回调 **onNewIntent** 方法，此模式启动 Activity A，系统首先会寻找是否存在 A 想要的任务栈，如果不存在，就会重新创建一个任务栈，然后把创建好 A 的实例放到栈中；
  - **singleInstance 单实例模式**：这是一种加强的 singleTask 模式，具有此种模式的 Activity 只能单独地位于一个任务栈中，且此任务栈中只有唯一一个实例；

5、了解哪些 Activity 常用的标记位 Flags？

- 参考回答：
  - **FLAG_ACTIVITY_NEW_TASK**：对应 singleTask 启动模式，其效果和在 XML 中指定该启动模式相同；
  - **FLAG_ACTIVITY_SINGLE_TOP** ： 对应 singleTop 启动模式，其效果和在 XML 中指定该启动模式相同；
  - **FLAG_ACTIVITY_CLEAR_TOP** ： 具有此标记位的 Activity，当它启动时，在同一个任务栈中所有位于它上面的 Activity 都要出栈。这个标记位一般会和 singleTask 模式一起出现，在这种情况下，被启动 Activity 的实例如果已经存在，那么系统就会回调 onNewIntent。如果被启动的 Activity 采用 standard 模式启动，那么它以及连同它之上的 Activity 都要出栈，系统会创建新的 Activity 实例并放入栈中；
  - **FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS** ： 具有这个标记的 Activity 不会出现在历史 Activity 列表中；

6、说下 Activity 跟 window，view 之间的关系？

- 参考回答：
  - Activity 创建时通过 attach()初始化了一个 Window 也就是 PhoneWindow，一个 PhoneWindow 持有一个 DecorView 的实例，DecorView 本身是一个 FrameLayout，继承于 View，Activty 通过 setContentView 将 xml 布局控件不断 addView()添加到 View 中，最终显示到 Window 于我们交互；

7、横竖屏切换的 Activity 生命周期变化？

- 参考回答：
  - **不设置** Activity 的 android:configChanges 时，切屏会销毁当前 Activity，然后重新加载调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次； onPause()→onStop()→onDestory()→onCreate()→onStart()→onResume()
  - 设置 Activity 的 android:configChanges="**orientation**"，经过机型测试
    - **在 Android5.1 即 API 23 级别下**，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
    - **在 Android9 即 API 28 级别下**，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法
    - **官方纠正后**，原话如下
      - 如果您的应用面向 **Android 3.2 即 API 级别 13** 或更高级别（按照 minSdkVersion 和 targetSdkVersion

属性所声明的级别），则还应声明 "screenSize" 配置，因为当设备在横向与纵向之间切换时，该配置也会发生变化。即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity

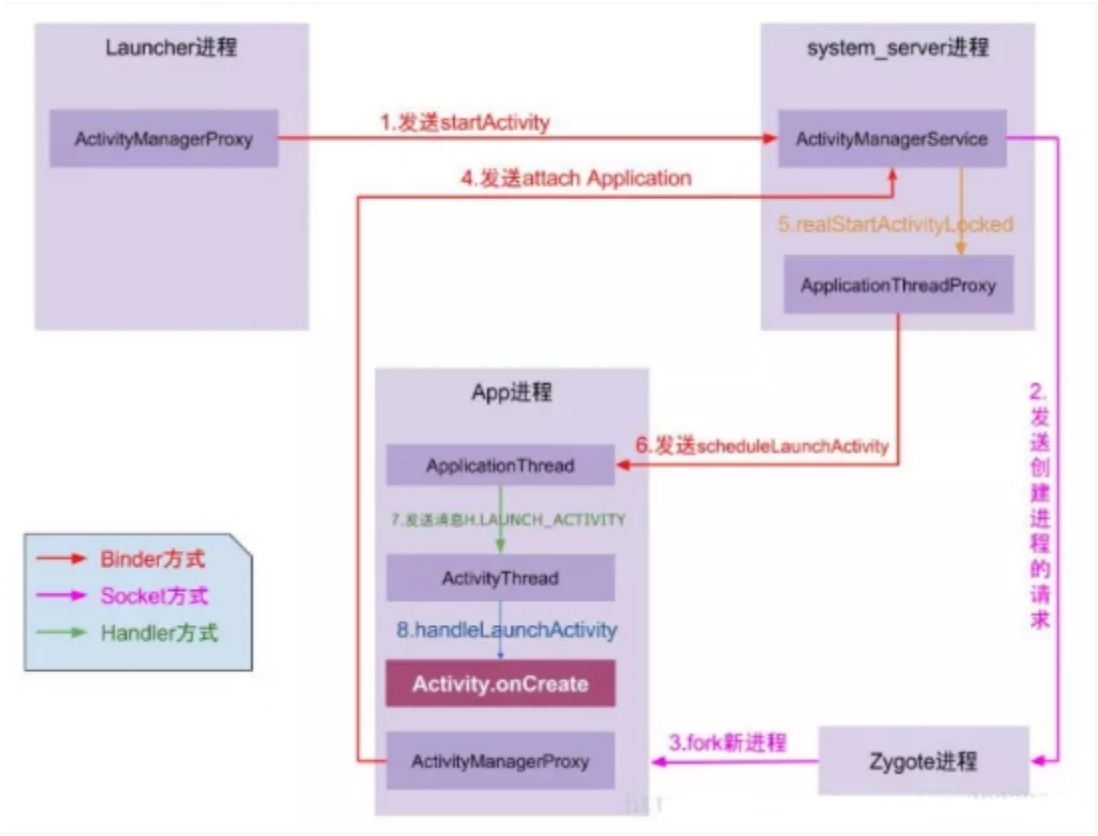
    - 设置 Activity 的 android:configChanges="orientation|keyboardHidden|screenSize"时，机型测试通过，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法；

8、如何启动其他应用的 Activity？

- 参考回答：
    - 在保证有权限访问的情况下，通过隐式 Intent 进行目标 Activity 的 IntentFilter 匹配，原则是：
        - 一个 intent 只有同时匹配某个 Activity 的 intent-filter 中的 action、category、data 才算完全匹配，才能启动该 Activity；
        - 一个 Activity 可以有多个 intent-filter，一个 intent 只要成功匹配任意一组 intent-filter，就可以启动该 Activity；

9、Activity 的启动过程？(重点)

- 参考回答：
    - 点击 App 图标后通过 startActivity 远程调用到 AMS 中，AMS 中将新启动的 activity 以 activityrecord 的结构压入 activity 栈中，并通过远程 binder 回调到原进程，使得原进程进入 pause 状态，原进程 pause 后通知 AMS 我 pause 了
    - 此时 AMS 再根据栈中 Activity 的启动 intent 中的 flag 是否含有 new_task 的标签判断是否需要启动新进程，启动新进程通过 startProcessXXX 的函数
    - 启动新进程后通过反射调用 ActivityThread 的 main 函数，main 函数中调用 looper.prepar 和 lopper.loop 启动消息队列循环机制。最后远程告知 AMS 我启动了。AMS 回调 handleLauncherAcitivyt 加载 activity。在 handlerLauncherActivity 中会通过反射调用 Application 的 onCreate 和 activity 的 onCreate 以及通过 handleResumeActivity 中反射调用 Activity 的 onResume


Launcher进程
ActivityManagerProxy
1.发送startActivity
system_server进程
ActivityManagerService
4.发送attach Application
5.realStartActivityLocked
ApplicationThreadProxy
App进程
6.发送scheduleLaunchActivity
ApplicationThread
7.发送消息H.LAUNCH_ACTIVITY
ActivityThread
8.handleLaunchActivity
Activity.onCreate
ActivityManagerProxy
3.fork新进程
Zygote进程
2.发送创建进程的请求
Binder方式
Socket方式
Handler方式