## **Android面试常问基础知识点**

**1、四大组件是什么**

1）Activity：用户可操作的可视化界面，为用户提供一个完成操作指令的窗口。一个Activity通常是一个单独的屏幕，Activity通过Intent来进行通信。Android中会维持一个Activity Stack，当一个新Activity创建时，它就会放到栈顶，这个Activity就处于运行状态。

2）Service：服务，运行在手机后台，适合执行不需和用户交互且还需长期运行的任务。

3）ContentProvider：内容提供者，使一个应用程序的指定数据集提供给其他应用程序，其他应用可通过ContentResolver类从该内容提供者中获取或存入数据。它提供了一种跨进程数据共享的方式，当数据被修改后，ContentResolver接口的notifyChange函数通知那些注册监控特定URI的ContentObserver对象。

如果ContentProvider和调用者在同一进程中，ContentProvider的方法(query/insert/update/delete等)和调用者在同一线程中；如果ContentProvider和调用者不在同一进程，ContentProvider方法会运行在它自身进程的一个Binder线程中。

4）Broadcast Receiver: 广播接收者，运用在应用程序间传输信息，可以使用广播接收器来让应用对一个外部事件做出响应。

**2、四大组件的生命周期和简单用法**

1）Activity：onCreate()->onStart()->onResume()->onPause()->onStop()->onDestory()

onCreate()：为Activity设置布局，此时界面还不可见；

onStart(): Activity可见但还不能与用户交互，不能获得焦点

onRestart(): 重新启动Activity时被回调

onResume(): Activity可见且可与用户进行交互

onPause(): 当前Activity暂停，不可与用户交互，但还可见。在新Activity启动前被系统调用保存现有的Activity中的持久数据、停止动画等。

onStop(): 当Activity被新的Activity覆盖不可见时被系统调用

onDestory(): 当Activity被系统销毁杀掉或是由于内存不足时调用

2）Service

a) onBind方式绑定的：onCreate->onBind->onUnBind->onDestory（不管调用bindService几次，onCreate只会调用一次，onStart不会被调用，建立连接后，service会一直运行，直到调用unBindService或是之前调用的bindService的Context不存在了，系统会自动停止Service,对应的onDestory会被调用）

b) startService启动的：onCreate->onStartCommand->onDestory(start多次，onCreate只会被调用一次，onStart会调用多次，该service会在后台运行，直至被调用stopService或是stopSelf)

c) 又被启动又被绑定的服务，不管如何调用onCreate()只被调用一次，startService调用多少次，onStart就会被调用多少次，而unbindService不会停止服务，必须调用stopService或是stopSelf来停止服务。必须unbindService和stopService(stopSelf）同时都调用了才会停止服务。

3）BroadcastReceiver

a) 动态注册：存活周期是在Context.registerReceiver和Context.unregisterReceiver之间，BroadcastReceiver每次收到广播都是使用注册传入的对象处理的。

b) 静态注册：进程在的情况下，receiver会正常收到广播，调用onReceive方法；生命周期只存活在onReceive函数中，此方法结束，BroadcastReceiver就销毁了。onReceive()只有十几秒存活时间，在onReceive()内操作超过10S，就会报ANR。

进程不存在的情况，广播相应的进程会被拉活，Application.onCreate会被调用，再调用onReceive。

4）ContentProvider：应该和应用的生命周期一样，它属于系统应用，应用启动时，它会跟着初始化，应用关闭或被杀，它会跟着结束。

**3、Activity之间的通信方式**

1）通过Intent方式传递参数跳转

2）通过广播方式

3）通过接口回调方式

4）借助类的静态变量或全局变量

5）借助SharedPreference或是外部存储，如数据库或本地文件

**4、Activity各种情况下的生命周期**

1) 两个Activity(A->B)切换(B正常的Activity)的生命周期：onPause(A)->onCreate(B)->onStart(B)->onResume(B)->oStop(A)

这时如果按回退键回退到A  onPause(B)->onRestart(A)->onStart(A)->onResume(A)->oStop(B)

如果在切换到B后调用了A.finish()，则会走到onDestory(A)，这时点回退键会退出应用

2) 两个Activity(A->B)切换(B透明主题的Activity或是Dialog风格的Acivity)的生命周期：onPause(A)->onCreate(B)->onStart(B)->onResume(B)

这时如果回退到A  onPause(B)->onResume(A)->oStop(B)->onDestory(B)

3) Activity(A)启动后点击Home键再回到应用的生命周期：onPause(A)->oStop(A)->onRestart(A)->onStart(A)->onResume(A)

**5、横竖屏切换的时候，Activity 各种情况下的生命周期**

1）切换横屏时：onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume

2) 切换竖屏时：会打印两次相同的log

onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume->onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume

3) 如果在AndroidMainfest.xml中修改该Activity的属性，添加android:configChanges="orientation"

横竖屏切换，打印的log一样，同1)

4) 如果AndroidMainfest.xml中该Activity中的android:configChanges="orientation|keyboardHidden"，则只会打印

onConfigurationChanged->

**6、Activity与Fragment之间生命周期比较**

Fragment生命周期：onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onPause->onStop->onDestoryView->onDestory->onDetach

切换到该Fragment：onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume

按下Power键：onPause->onSaveInstanceState->onStop

点亮屏幕解锁：onStart->onRestoreInstanceState->onResume

切换到其他Fragment: onPause->onStop->onDestoryView

切回到该Fragment: onCreateView->onActivityCreated->onStart->onResume

退出应用：onPause->onStop->onDestoryView->onDestory->onDetach

**7、Activity上有Dialog的时候按Home键时的生命周期**

AlertDialog并不会影响Activity的生命周期，按Home键后才会使Activity走onPause->onStop，AlertDialog只是一个组件，并不会使Activity进入后台。

**8、两个Activity 之间跳转时必然会执行的是哪几个方法？**

前一个Activity的onPause，后一个Activity的onResume

**9、前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法。**

1）前台切换到后台，会执行onPause->onStop，再回到前台，会执行onRestart->onStart->onResume

2) 弹出Dialog，并不会影响Activity生命周期

**10、Activity的四种启动模式对比**

1）standard：标准启动模式（默认），每启动一次Activity，都会创建一个实例，即使从ActivityA startActivity ActivityA,也会再次创建A的实例放于栈顶，当回退时，回到上一个ActivityA的实例。

2) singleTop：栈顶复用模式，每次启动Activity，如果待启动的Activity位于栈顶，则不会重新创建Activity的实例，即不会走onCreate->onStart，会直接进入Activity的onPause->onNewIntent->onResume方法

3) singleInstance: 单一实例模式，整个手机操作系统里只有一个该Activity实例存在，没有其他Actvity,后续请求均不会创建新的Activity。若task中存在实例，执行实例的onNewIntent()。应用场景：闹钟、浏览器、电话

4) singleTask：栈内复用，启动的Activity如果在指定的taskAffinity的task栈中存在相应的实例，则会把它上面的Activity都出栈，直到当前Activity实例位于栈顶，执行相应的onNewIntent()方法。如果指定的task不存在，创建指定的taskAffinity的task,taskAffinity的作用，进入指写taskAffinity的task,如果指定的task存在，将task移到前台，如果指定的task不存在，创建指定的taskAffinity的task. 应用场景：应用的主页面

**11、Activity状态保存于恢复**

Activity被主动回收时，如按下Back键，系统不会保存它的状态，只有被动回收时，虽然这个Activity实例已被销毁，但系统在新建一个Activity实例时，会带上先前被回收Activity的信息。在当前Activity被销毁前调用onSaveInstanceState(onPause和onStop之间保存)，重新创建Activity后会在onCreate后调用onRestoreInstanceState（onStart和onResume之间被调用），它们的参数Bundle用来数据保存和读取的。

保存View状态有两个前提：View的子类必须实现了onSaveInstanceState; 必须要设定Id，这个ID作为Bundle的Key

**12、fragment各种情况下的生命周期**

正常情况下的生命周期：onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onPause->onStop->onDestoryView->onDestory->onDetach

1)Fragment在Activity中replace  onPause(旧)->onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onStop(旧)->onDestoryView(旧)

如果添加到backStack中，调用remove()方法fragment的方法会走到onDestoryView，但不会执行onDetach()，即fragment本身的实例是存在的，成员变量也存在，但是view被销毁了。如果新替换的Fragment已在BackStack中，则不会执行onAttach->onCreate

**13、Fragment状态保存onSaveInstanceState是哪个类的方法，在什么情况下使用？**

在对应的FragmentActivity.onSaveInstanceState方法会调用FragmentController.saveAllState，其中会对mActive中各个Fragment的实例状态和View状态分别进行保存.当Activity在做状态保存和恢复的时候, 在它其中的fragment自然也需要做状态保存和恢复.

**14、Fragment.startActivityForResult是和FragmentActivity的startActivityForResult？**

如果希望在Fragment的onActivityResult接收数据，就要调用Fragment.startActivityForResult， 而不是Fragment.getActivity().startActivityForResult。Fragment.startActivityForResult->FragmentActivitymHost.HostCallbacks.onStartActivityFromFragment->FragmentActivity.startActivityFromFragment。如果request=-1则直接调用FragmentActivity.startActivityForResult，它会重新计算requestCode，使其大于0xfffff。

**15、如何实现Fragment的滑动？**

ViewPager+FragmentPagerAdapter+List<Fragment>

**16、fragment之间传递数据的方式？**

1)在相应的fragment中编写方法，在需要回调的fragment里获取对应的Fragment实例，调用相应的方法；

2）采用接口回调的方式进行数据传递；

a) 在Fragment1中创建一个接口及接口对应的set方法; b) 在Fragment1中调用接口的方法；c)在Fragment2中实现该接口；

3）利用第三方开源框架EventBus

**17、service和activity怎么进行数据交互？**

1）通过bindService启动服务，可以在ServiceConnection的onServiceConnected中获取到Service的实例，这样就可以调用service的方法，如果service想调用activity的方法，可以在service中定义接口类及相应的set方法，在activity中实现相应的接口，这样service就可以回调接口言法；

2）通过广播方式

**18、说说ContentProvider、ContentResolver、ContentObserver 之间的关系**

ContentProvider实现各个应用程序间数据共享，用来提供内容给别的应用操作。如联系人应用中就使用了ContentProvider，可以在自己应用中读取和修改联系人信息，不过需要获取相应的权限。它也只是一个中间件，真正的数据源是文件或SQLite等。

ContentResolver内容解析者，用于获取内容提供者提供的数据，通过ContentResolver.notifyChange(uri)发出消息

ContentObserver内容监听者，可以监听数据的改变状态，观察特定Uri引起的数据库变化，继而做一些相应的处理，类似于数据库中的触发器，当ContentObserver所观察的Uri发生变化时，便会触发它。

**19、请描述一下广播BroadcastReceiver的理解**

BroadcastReceiver是一种全局监听器，用来实现系统中不同组件之间的通信。有时候也会用来作为传输少量而且发送频率低的数据，但是如果数据的发送频率比较高或者数量比较大就不建议用广播接收者来接收了，因为这样的效率很不好，因为BroadcastReceiver接收数据的开销还是比较大的。

**20、广播的分类**

**1）普通广播**：完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，消息传递的效率比较高，并且无法中断广播的传播。

**2）有序广播**：发送有序广播后，广播接收者将按预先声明的优先级依次接收Broadcast。优先级高的优先接收到广播，而在其onReceiver()执行过程中，广播不会传播到下一个接收者，此时当前的广播接收者可以abortBroadcast()来终止广播继续向下传播，也可以将intent中的数据进行修改设置，然后将其传播到下一个广播接收者。 sendOrderedBroadcast(intent, null);//发送有序广播

**3）粘性广播**：sendStickyBroadcast()来发送该类型的广播信息，这种的广播的最大特点是，当粘性广播发送后，最后的一个粘性广播会滞留在操作系统中。如果在粘性广播发送后的一段时间里，如果有新的符合广播的动态注册的广播接收者注册，将会收到这个广播消息，虽然这个广播是在广播接收者注册之前发送的，另外一点，对于静态注册的广播接收者来说，这个等同于普通广播。

**21、广播使用的方式和场景**

**1）App全局监听**：在AndroidManifest中静态注册的广播接收器，一般我们在收到该消息后，需要做一些相应的动作，而这些动作与当前App的组件，比如Activity或者Service的是否运行无关，比如我们在集成第三方Push SDK时，一般都会添加一个静态注册的BroadcastReceiver来监听Push消息，当有Push消息过来时，会在后台做一些网络请求或者发送通知等等。

**2）组件局部监听**：这种主要是在Activity或者Service中使用registerReceiver()动态注册的广播接收器，因为当我们收到一些特定的消息，比如网络连接发生变化时，我们可能需要在当前Activity页面给用户一些UI上的提示，或者将Service中的网络请求任务暂停。所以这种动态注册的广播接收器适合特定组件的特定消息处理。

**22、在manifest 和代码中如何注册和使用BroadcastReceiver?**

1）mainfest中注册:静态注册的广播接收者就是一个常驻在系统中的全局监听器，也就是说如果你应用中配置了一个静态的BroadcastReceiver，而且你安装了应用而无论应用是否处于运行状态，广播接收者都是已经常驻在系统中了。

<*receiver android:name=".MyBroadcastReceiver">*

*<intent-filter>*

*<action android:name="com.smilexie.test.intent.mybroadcastreceiver"/>*

*</intent-filter>*

*</receiver>*

2) 动态注册:动态注册的广播接收者只有执行了registerReceiver(receiver, filter)才会开始监听广播消息，并对广播消息作为相应的处理。

*IntentFilter fiter = new IntentFilter("com.smilexie.test.intent.mybroadcastreceiver");*

*MyBroadcastReceiver receiver = new MyBroadcastReceiver();*

*registerReceiver(receiver, filter);*

//撤销广播接受者的动态注册

unregisterReceiver(receiver);

**23、本地广播和全局广播有什么差别？**

1）LocalBroadcastReceiver仅在自己的应用内发送接收广播，也就是只有自己的应用能收到，数据更加安全。广播只在这个程序里，而且效率更高。只能动态注册，在发送和注册的时候采用LocalBroadcastManager的sendBroadcast方法和registerReceiver方法。

2）全局广播：发送的广播事件可被其他应用程序获取，也能响应其他应用程序发送的广播事件（可以通过 exported–是否监听其他应用程序发送的广播 在清单文件中控制） 全局广播既可以动态注册，也可以静态注册。

**24、AlertDialog,popupWindow,Activity区别**

（1）Popupwindow在显示之前一定要设置宽高，Dialog无此限制。

（2）Popupwindow默认不会响应物理键盘的back，除非显示设置了popup.setFocusable(true);而在点击back的时候，Dialog会消失。

（3）Popupwindow不会给页面其他的部分添加蒙层，而Dialog会。

（4）Popupwindow没有标题，Dialog默认有标题，可以通过dialog.requestWindowFeature(Window.FEATURE\_NO\_TITLE);取消标题

（5）二者显示的时候都要设置Gravity。如果不设置，Dialog默认是Gravity.CENTER。

（6）二者都有默认的背景，都可以通过setBackgroundDrawable(new ColorDrawable(android.R.color.transparent));去掉。

（7）Popupwindow弹出后，取得了用户操作的响应处理权限，使得其他UI控件不被触发。而AlertDialog弹出后，点击背景，AlertDialog会消失。

**25、Application 和 Activity 的 Context 对象的区别**

1）Application Context是伴随应用生命周期；不可以showDialog, startActivity, LayoutInflation

可以startService\BindService\sendBroadcast\registerBroadcast\load Resource values

2）Activity Context指生命周期只与当前Activity有关，而Activity Context这些操作都可以，即凡是跟UI相关的，都得用Activity做为Context来处理。

一个应用Context的数量=Activity数量+Service数量+1（Application数量）

1. **Android属性动画特性**

属性动画（Property Animation）是在 Android 3.0（API 11）后才提供的一种全新动画模式

实现动画效果在Android开发中非常常见，因此Android系统一开始就提供了两种实现动画的方式：

逐帧动画（Frame Animation）

补间动画（ Tweened animation ）

逐帧动画 & 补间动画存在一定的缺点：

a. 作用对象局限：View

即补间动画 只能够作用在视图View上，即只可以对一个Button、TextView、甚至是LinearLayout、或者其它继承自View的组件进行动画操作，但无法对非View的对象进行动画操作

有些情况下的动画效果只是视图的某个属性 & 对象而不是整个视图；

如，现需要实现视图的颜色动态变化，那么就需要操作视图的颜色属性从而实现动画效果，而不是针对整个视图进行动画操作

b. 没有改变View的属性，只是改变视觉效果

补间动画只是改变了View的视觉效果，而不会真正去改变View的属性。

如，将屏幕左上角的按钮 通过补间动画 移动到屏幕的右下角

点击当前按钮位置（屏幕右下角）是没有效果的，因为实际上按钮还是停留在屏幕左上角，补间动画只是将这个按钮绘制到屏幕右下角，改变了视觉效果而已。

c. 动画效果单一

补间动画只能实现平移、旋转、缩放 & 透明度这些简单的动画需求

一旦遇到相对复杂的动画效果，即超出了上述4种动画效果，那么补间动画则无法实现。

属性动画中最基础的 ObjectAnimator

ObjectAnimator 是属性动画中最重要的实行类，创建一个ObjectAnimator只需要通过他的静态工厂方类直接返回一个ObjectAnimator对象。参数包括一个对象和对象的属性名字，但这个属性必须有get和set函数，内部会通过反射机制来调用set函数修改对象属性值，也可以调用setInterpolator设置对应的差值器。

属性动画操作的是属性，所以被操作的view可以响应事件。

属性动画可以对任何对象的属性做动画而不仅仅是View，甚至可以没有对象。除了作用对象进行扩展外，属性动画的效果也加强了，不仅能实现View动画的4中效果，还能实现其它多种效果，这些效果都是通过ValuAnimator或ObjectAnimator、AnimatorSet等来实现的。

区别和特性

1.  补间动画：只产生了一个动画效果，其真实的坐标并没有发生改变（只是改变了View的显示效果而已，并不会真正的改变View的属性）。View做在做动画的时候，它并没有真正的移动它的位置，而是根据动画时间的插值，计算出一个Matrix，然后不停的invalidate，在onDraw中的Canvas上使用这个计算出来的Matrix去draw这个View的内容，并有onLayout中还是原来的位置，所以点击事件只能点击到原来的位置才能触发

2. ObjectAnimator：一般直接用与View，要求作用的View提供该属性（如View的scaleX属性）的getter、setter方法（如setScaleX()方法），可以直接改变view的属性所以View的位置也跟随属性的改变而改变，点击事件的触发位置为动画结束的位置。

3. ValueAnimator：属性动画的核心，这个我理解为数值动画，ObjectAnimator也只不过是通过不断改变的数值然后赋值给相应的属性而已。通过设置初始和终点值，ValueAnimator 会通过相应的Interpolator  和duration 计算出平滑的数值变化，然后可以通过得到的Value进行任意操作

1. **如何导入外部数据库?**

我们平时见到的android数据库操作一般都是在程序开始时创建一个空的数据库，然后再进行相关操作。如果我们需要使用一个已有数据的数据库怎么办呢？  
我们都知道android系统下数据库应该存放在 /data/data/com.\*.\*(package name)/ 目录下，所以我们需要做的是把已有的数据库传入那个目录下。操作方法是用FileInputStream读取原数据库，再用FileOutputStream把读取到的东西写入到那个目录。  
操作方法：1. 把原数据库包括在项目源码的 res/raw 目录下，然后建立一个DBManager类

然后在程序的首个Activity中示例化一个DBManager对象，然后对其执行openDatabase方法就可以完成导入了，可以把一些要对数据库进行的操作写在DBManager类里，然后通过DBManager类的对象调用；也可以在完成导入之后通过一个SQliteDatabase类的对象打开数据库，并执行操作。

1. **LinearLayout、RelativeLayout、FrameLayout的特性及对比，并介绍使用场景。**

**LinearLayout**

LinearLayout按照垂直或者水平的顺序依次排列子元素，每一个子元素都位于前一个元素之后。如果是垂直排列，那么将是一个N行单列的结构，每一行只会有一个元素，而不论这个元素的宽度为多少；如果是水平排列，那么将是一个单行N列的结构。如果搭建两行两列的结构，通常的方式是先垂直排列两个元素，每一个元素里再包含一个LinearLayout进行水平排列。

**RelativeLayout**

RelativeLayout按照各子元素之间的位置关系完成布局。在此布局中的子元素里与位置相关的属性将生效。例如android：layout\_below, android:layout\_above等。子元素就通过这些属性和各自的ID配合指定位置关系。

**FrameLayout**

FrameLayout是五大布局中最简单的一个布局，在这个布局中，整个界面被当成一块空白备用区域，所有的子元素都不能被指定放置的位置，它们统统放于这块区域的左上角，并且后面的子元素直接覆盖在前面的子元素之上，将前面的子元素部分和全部遮挡。

**总结**

（1）RelativeLayout慢于LinearLayout是因为它会让子View调用2次measure过程，而后者只需一次，但是有weight属性存在时，后者同样会进行两次measure。

（2）RelativeLayout的子View如果高度和RelativeLayout不同，会引发效率问题，可以使用padding代替margin以优化此问题。

（3）在不响应层级深度的情况下，使用Linearlayout和FrameLayout而不是RelativeLayout。

结论中的第三条也解释了文章前言中的问题：DecorView的层级深度已知且固定的，上面一个标题栏，下面一个内容栏，采用RelativeLayout并不会降低层级深度，因此这种情况下使用LinearLayout效率更高。

而为开发者默认新建RelativeLayout是希望开发者能采用尽量少的View层级，很多效果是需要多层LinearLayout的嵌套，这必然不如一层的RelativeLayout性能更好。因此我们应该尽量减少布局嵌套，减少层级结构，使用比如viewStub，include等技巧

1. **谈谈对接口与回调的理解**

接口回调是指：  
可以把使用实现了某一接口的类创建的对象的引用，赋给该接口声明的接口变量，那么该接口变量就可以调用被类实现的接口的方法。实际上，当接口变量调用被类实现的接口中的方法时，就是通知相应的对象调用接口的方法，这一过程称为对象功能的接口回调。

指导思想：  
面向对象设计的封装性，模块间要解耦，模块内要内聚

关键字：解耦  
削弱模块间的协作性，也就是耦合性，使得模块之间高度独立，模块间更多的从事着单向的调用工作。一个模块需要某种服务就去找另一个模块，这使得程序呈现出层次性，高层通过接口调用底层，底层提供服务。摈弃模块间你中有我我中有你的暧昧关系。  
关键字：协作  
但我们又需要模块间的协作，所以需要回调（C++中是指针）。如果A对象关心B对象的状态变化，那么给B对象的状态的变化注册接口回调函数，A实现接口，让接口函数通知A，B对象状态的改变，这样在封装了模块变化的同时实现了模块间的协作关系，另辟独径的给对象解耦。  
关键字：变量  
回调函数跟普通函数只是在调用函数时略有区别。一般调用普通函数时，直接写函数名即可。但是在调用所谓“回调”函数时，是把它作为参数传递给另一函数。关键就在于“参数”这两个字。为什么要把某个东西参数化？就是它存在变化的可能性。既然可以把变量做成参数，那么函数也同样可以做成参数，只要它们有变化的可能。对于一成不变的东西，显然直接嵌入便可。

1）定义接口： 定义一个接口、定义其中的抽象方法、抽象方法含有参数(被传递的数据)；

2）编写回调方法： 在定义接口的类中，编写用户回调的方法，要传递一个接口对象实例，让别的类去实现。（相当于为接口成员变量赋值）

3）为抽象方法赋值： 获取一个全局的接口成员变量，在某个事件中使用接口成员变量调用接口中的方法，并且为抽象方法中的参数赋值。（这一步可以在回调方法中实现）

需要一个接口的实现类：

1）让当前Activity实现接口，变成接口的实现类；

2）写一个类去实现接口，实现其中的抽象方法，然后在需要的地方创建一个接口实现类的子类对象

3）直接在当前位置使用匿名对象实现,创建一个接口实例。

回调原理：

接口调用自己的抽象方法，相当于接口的实现类调用实现类中重写的抽象方法；

接口中没有构造函数:

1）接口中是没有构造函数的，不能直接创建对象，只能由实现类创建对象；接口中的成员常量不需要进行初始化，所以不需要构造函数。

2）而抽象类是有构造方法的，为了给子类调用初始化抽象类中的成员变量。

接口的特点：

1）接口用关键字interface表示；类实现接口用implements表示。

2）接口不能实例化：那么，接口如何实例化呢?按照多态的方式，由具体的子类实例化。其实这也是多态的一种，接口多态。

3）接口的子类：要么是抽象类，要么重写接口中的所有抽象方法。

接口中的成员：

成员变量：只能是常量  默认修饰符 public static final(不能被修改)

构造方法：没有，因为接口主要是扩展功能的，而没有具体存在

成员方法：只能是抽象方法  默认修饰符 public abstract(可以省略)

类与类,类与接口以及接口与接口的关系：

类与类： 继承关系，只能单继承，但是可以多层继承

类与接口：实现关系，可以多实现,还可以在继承一个类的同时实现多个接口。

接口与接口：继承关系，可以单继承，也可以多继承。

1. **介绍下SurfView**

1.什么是SurfaceView？

    Surface意为表层、表面，顾名思义SurfaceView就是指一个在表层的View对象。为什么说是在表层呢，这是因为它有点特殊跟其他View不一样，其他View是绘制在“表层”的上面，而它就是充当“表层”本身。SDK的文档 说到：SurfaceView就是在窗口上挖一个洞，它就是显示在这个洞里，其他的View是显示在窗口上，所以View可以显式在 SurfaceView之上，你也可以添加一些层在SurfaceView之上。 从API中可以看出SurfaceView属于View的子类 它是专门为制作游戏而产生的，它的功能非常强大，最重要的是它支持OpenGL ES库，2D和3D的效果都可以实现。创建SurfaceView的时候需要实现SurfaceHolder.Callback接口，它可以用来监听SurfaceView的状态，比如：SurfaceView的改变 、SurfaceView的创建 、SurfaceView 销毁等，我们可以在相应的方法中做一些比如初始化的操作或者清空的操作等等。

Android系统提供了View进行绘图处理，我们通过自定义的View可以满足大部分的绘图需求，但是这有个问题就是我们通常自定义的View是用于主动更新情况的，用户无法控制其绘制的速度，由于View是通过invalidate方法通知系统去调用view.onDraw方法进行重绘，而Android系统是通过发出VSYNC信号来进行屏幕的重绘，刷新的时间是16ms,如果在16ms内View完成不了执行的操作，用户就会看着卡顿，比如当draw方法里执行的逻辑过多，需要频繁刷新的界面上，例如游戏界面，那么就会不断的阻塞主线程，从而导致画面卡顿。而SurfaceView相当于是另一个绘图线程，它是不会阻碍主线程，并且它在底层实现机制中实现了双缓冲机制。

2.如何使用SurfaceView？

        首先SurfaceView也是一个View，它也有自己的生命周期。因为它需要另外一个线程来执行绘制操作，所以我们可以在它生命周期的初始化阶 段开辟一个新线程，然后开始执行绘制，当生命周期的结束阶段我们插入结束绘制线程的操作。这些是由其内部一个SurfaceHolder对象完成的。

SurfaceView它的绘制原理是绘制前先锁定画布（获取画布），然后等都绘制结束以后在对画布进行解锁 ，最后在把画布内容显示到屏幕上。  

通常情况下，使用以下步骤来创建一个SurfaceView的模板：

（1）创建SurfaceView

创建自定义的SurfaceView继承自SurfaceView,并实现两个接口：SurfaceHolder.Callback和Runnable.

（2）初始化SurfaceView

在自定义的MySurfaceView的构造方法中，需要对SurfaceView进行初始化，包括SurfaceHolder的初始化、画笔的初始化等。在自定义的SurfaceView中，通常需要定义以下三个成员变量:

private SurfaceHolder mHolder;

private Canvas mCanvas;//绘图的画布

private boolean mIsDrawing;//控制绘画线程的标志位

  SurfaceHolder，顾名思义，它里面保存了一个对Surface对象的引用，而我们执行绘制方法本质上就是操控Surface。SurfaceHolder因为保存了对Surface的引用，所以使用它来处理Surface的生命周期。（说到底 SurfaceView的生命周期其实就是Surface的生命周期）例如使用 SurfaceHolder来处理生命周期的初始化。

（3）使用SurfaceView

通过SurfaceHolder对象的lockCanvans()方法，我们可以获取当前的Canvas绘图对象。接下来的操作就和自定义View中的绘图操作一样了。需要注意的是这里获取到的Canvas对象还是继续上次的Canvas对象，而不是一个新的对象。因此，之前的绘图操作都会被保留，如果需要擦除，则可以在绘制前，通过drawColor()方法来进行清屏操作。

绘制的时候，在surfaceCreated()方法中开启子线程进行绘制，而子线程使用一个while(mIsDrawing)的循环来不停的进行绘制，在绘制的逻辑中通过lockCanvas()方法获取Canvas对象进行绘制，通过unlockCanvasAndPost(mCanvas)方法对画布内容进行提交。整体代码模板如下：

这里说一个优化的地方，这就是在run方法中。

在我们的draw()方法每一次更新所耗费的时间是不确定的。举个例子 比如第一次循环draw() 耗费了1000毫秒 ，第二次循环draw() 耗时2000毫秒。很明显这样就会造成运行刷新时间时快时慢,可能出现卡顿现象。为此最好保证每次刷新的时间是相同的，这样可以保证整体画面过渡流畅。

这里说一下Thread.yield(): 与Thread.sleep(long millis):的区别：

Thread.yield(): 是暂停当前正在执行的线程对象 ，并去执行其他线程。

Thread.sleep(long millis):则是使当前线程暂停参数中所指定的毫秒数然后在继续执行线程。

3.SurfaceView的使用实例

（1）正弦曲线

要绘制一个正弦曲线，只需要不断修改横纵坐标的值，并让他们满足正弦函数即可。因此，我们需要一个Path对象来保存正弦函数上的坐标点，在子线程的while循环中，不断改变横纵坐标值。

（2）画图板

我们也可以通过使用SurfaceView来实现一个简单的绘图板，绘图的方法与View中进行绘图所使用的方法一样，也是通过Path对象记录手指滑动的路径来进行绘图。4.SurfaceView和View的区别

总的归纳起来SurfaceView和View不同之处有：

1. SurfaceView允许其他线程更新视图对象(执行绘制方法)而View不允许这么做，它只允许UI线程更新视图对象。

2. SurfaceView是放在其他最底层的视图层次中，所有其他视图层都在它上面，所以在它之上可以添加一些层，而且它不能是透明的。

3. 它执行动画的效率比View高，而且你可以控制帧数。

4. SurfaceView在绘图时使用l了双缓冲机制，而View没有。

**31、序列化的作用，以及Android两种序列化的区别**

序列化 (Serialization)将对象的状态信息转换为可以存储或传输的形式的过程。在序列化期间，对象将其当前状态写入到临时或持久性存储区。以后，可以通过从存储区中读取或反序列化对象的状态，重新创建该对象。

简单地说：“序列化”就是将运行时的对象状态转换成二进制，然后保存到流、内存或者通过网络传输给其他端。



1. **差值器，估值器**

**<https://www.jianshu.com/p/2f19fe1e3ca1>**

**这篇文章介绍的十分详细，大家可以系统的学习，了解一下。**

1. **Android中数据存储方式**

[1 使用SharedPreferences存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type1)

[2 文件存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type2)

[3 SQLite数据库存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type3)

[4 使用ContentProvider存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type4)

[5 网络存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type5)

--