**Android生命周期的学习**

1. Activity生命周期的4中状态和7个方法

生命周期为： 1.完整onCreate() 只会调用一次，在创建时，进行初始化。当此前的Activity由于内存不够被杀死的话。返回时，还是会调用onCreate()

,onStart() Activity处于可见状态

,onResume() Activity获得焦点

,onPause 暂停状态，失去焦点

,onStop 不可见状态

,onDestory 销毁状态

2.可见生命周期，有一个活动区启动另一个活动、点击home建，点击menu返回后台。由onStart() 到 onStop,当活动返回时，调用 onRestart() onStart() onResume()。

3,前台生命周期，弹出一个dialog,后面的活动还是可见，只是失去焦点。onResume() ,onPasue()，dialog消失时调用，onResume()方法。

补充：超级完整周期：onCreate() ,一般都是在create中进行恢复数据

,onStart(),

onRestoreInstanceState()，此方法调用只有活动被系统杀死时才可以调用恢复，如旋转屏幕，内存不够杀死。

onResume(),

onAttachedToWindow(),活动添加到窗口

onPause(),

onSaveInstanceState()，在活动变得不可见时，调用此方法保存活动的状态，以便恢复

onStop(),

onDestory()

onDetachedFromWindow(),活动从窗口中移除

onRestart()

1. Android的框架层：由上向下：应用层 ：包含home，contacts，phone，等程序

应用框架层：有activitiy Manager,window Manager等

系统运行库层：一些C/C++库，android运行库和虚拟机

Linux内核层：蓝牙驱动，显示驱动，wifi等为上层提供硬件支持。

**Android的四大组件**

**Activity:**

1. Activity:所有可见的都是Activity。
2. 任务栈：activitiy每启动一个，就会添加到任务栈中。每一个可见的activity都处于栈顶，当销毁一个活动时，activity会从栈顶移除，出栈。
3. Activity的四种启动模式：

1.standard（默认）标准模式，使用startActivity（intent）启动一次Activity，就会创建一个实例存储在任务栈中，然后返回时，会有多个重复的Activity。

2.singleTop模式，如果此时activity处于栈顶时，那么跳转到这个activity时不会再次创建一个新的实例，而是使用这个实例。

3.singleTask模式，跳转Activity时，会现在栈中查看有无此活动的实例，如果有的话，则将此activity之上的所有活动全部出栈，此时此活动处于栈顶，跳转完成。反之，创建此activity实例。

4.singleInstance模式，也是系统单例模式。如果此activity模式是singleInstance，则在启动一个活动时，会创建一个task栈存放新创建的活动。此模式下一个栈中，只允许一个activity存在。

1. 在Activity之外进行页面跳转的时候，Intent需要添加FLAG,标记，如intent. addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK)。 （因为activity中以默认加上）
2. Intent:意图。启动方式，1.component，显示跳转。

2.action，category行为和类别配合使用。隐式跳转

3.data，type.类别和数据，跳转时可以携带，打开网页，打电话等。

6) Activity隐式调用时，必须要写<category android:name="android.intent.category.DEFAULT" />不添加的话就会报错。

**Service:**

1. Service：后台进行的一切不可见，不用用户知道的操作，如下载，音乐播放等。
2. Service两种启动方式：

1.通过startService(),启动，stopService()关闭，onCreate,onStartCommand,onDestory.

2.通过bindService(),启动,unbindService关闭，onCreate,onbind,onUnbind,onDestory

1. Android 5.0时启动service必须要显示调用，想要隐式调用必须指明包名。
2. Service与线程的区别，service是不可见的但是也运行在主线程，不能直接在service中作耗时操作，不然还是会导致ANR(application not responding).所以正确的做法是在service中开一个线程来进行耗时操作。
3. 为什么不在activity中做耗时操作呢？因为activity要与用户交互频繁，一旦被销毁，就不能对线程进行控制，而service可以与多有的activity进行交互，可以很方便的进行控制。
4. 进程分为：前台进程，可见进程，服务进程，后台进程，空进程。
5. 前台service：因为service的优先级较低，当内存不足时，有可能会被杀死的。所以要将service变成前台service.,使用NotificationCompat.Bulider 来创建信息，向下兼容，使用build()生成Notification对象。然后使用NotificationManager对象的notify方法来显示通知，在使用startForeground()方法来设置为前台服务。
6. 远程服务：在androidManifest.xml中的service设置process=”:remote”表名为远程服务，不在一个进程中。或者创建AIDL，实现进程间通信。

**BroadcastReceiver：**

1. 静态注册：在程序没有启动的情况下，就能接收到广播信息，在AndroidManifest中注册，然后在代码中继承BroadcastReceiver类重写onReceive方法
2. 动态注册：只有程序运行时才能接受到广播，较安全，也是要实现一个接收器，在代码中registerReceiver，unregister..注册和注销。使用sendBroadcast发送广播
3. 广播分类：可分为有序广播和无序广播。有序广播按照优先级进行发送一个一个接收，可以在manifest中设置优先级-1000~1000之间。无序广播，都可以接受到信息。
4. 广播的截断：只有有序广播可以被截断，使用方法abortBroadcast()
5. 系统广播：系统中默认的广播信息，如开机启动，电池电量，网络改变。
6. NFC：近距离无线通信，1.在AndroidManifest中声明nfc权限。2.在获取nfc的activity中获取注册action信息

**ContentProvider:**

1. 主要用来提供数据的共享，实现一个应用对另一个应用的增删改查。
2. 继承ContentProvider，在AndroidManifest中注册，填写authority=包名+类名
3. 在ContentProvider中初始化UriMatcher,添加数据。
4. 重写相应的增删改查方法。
5. MimeType类型返回为，vnd.android.cursor.dir/person，多条数据 ，vnd.android.cursor.item/person，单条数据返回
6. 在activity查询时，获取contentResolve对象，调用query方法，传入uri
7. Uri为content://包名/类名/数据库列

**Android中的MVVM与DataBinding**

1. Build Environment:在app的module中bulid.gradle配置dataBinding
2. 布局中根布局为layout，data元素里面有两个属性 1.variable 变量名用来绑定数据

2.type类型名表示绑定的类，在接下来的控件中进行绑定如：textView的text=”@{…}”

1. 然后在activity中，会自动生成刚才binding的layout布局,转换成为pascal以Binding结尾，如activity\_main ---》 ActivityMainBinding amb
2. 然后调用setcontent()进行绑定，调用amb.setVarible()。进行绑定
3. 事件处理两种方式：1.方法引用：如onclick=“test”,一个重要的好处是表达式在编译期间就被确定了。

2.监听器绑定： lambda表达式

1. Import，就像在java代码中一样，dataBinding在xml的data元素下也可以进行导包，因为java.lang包默认导入故可以直接使用。
2. 类名重复时，可以添加alias标签
3. 导入的类型，也可以用于variable元素中
4. 可以在xml中直接使用带id的view，应用时采用驼峰命名法，系统自动生成为public final TextView t; 比findviewbyId速度快
5. Bingding类会自动生成一个context对象，也就是rootView的getContext()的返回值
6. binding类名自动生成，可以通过修改在data元素下的class标签
7. dataBinding支持include，但是不能再merge节点中使用，在布局中使用。
8. 常用表达式，1.数学表达式，2字符串拼接+，3逻辑运算&&||，4一元++，--，！，~，

5．比较运算，6.instanceOf，7.方法调用，8.类型转换

9.数组存取，10,.三元运算符?;;

1. 在使用上述表达式时，注意转义字符的使用，如<--&lt, >-- &gt
2. 不支持的操作符为this，super，new，泛型调用
3. Null合并运算??，在非null的时候选择左边，反之右边。User.lastName??user.firstName

**Android的基本UI学习**

1. Android的布局分为，四大布局：

1.FrameLayout：最简单的布局管理器，默认view放在左上角，视图会层叠排放。

2.LinearLayout:线性布局，分为水平和垂直

3.RelativeLayout:相对布局，以父布局或者子view的位置，进行布局

4.GridLayout:网格布局， 在android4.0中引用，很灵活，简化布局，减少嵌套

1. 布局的优化：多层嵌套可对应用程序性能和响应能力造成影响。
2. 嵌套最好控制在10层以内，最好使用merge和include结合。
3. 布局中的view不能超过80个
4. 使用Lint工具来分析布局：android studio中已经集成了lint工具
5. Merge标签可以减少视图间的多余，如主布局为垂直布局，include的布局也是垂直布局，那么我们可以再include的布局使用merge标签，引用时自动优化。
6. ViewStub标签，好处是当你需要时才会加载。不常用的布局如进度条，显示错误信息等可以使用，viewstub是个不可见的，大小为0的view。想要使用时，可以findviewbyid在setVisibility,或者直接find。.inflate().

具体的使用：1.在布局中使用layout：指向特定的布局，inflateid：特定的view

2.在代码中显示使用inflate或者setVisibility

1. Fragment：1.fragment的生命周期
2. Fragment添加到用户界面的两种方式：

1.静态加载，直接在布局中引用，指定name元素，每个fragment都必须使用唯一的标示符，id或者tag.

2.动态加载，在程序中加载，使用fragmentTransaction的add,replace,hide,show方法

1. FragmentManager的管理使用，在提交之前调用addToBackStack(null)返回栈
2. Commit()方法不会立即提交，而是异步操作，也可以使用exectuPendingTransactions()来立即执行commit方法。
3. Fragment与Activity的通信，在fragment中使用getActivity().findViewById.在activity中可以getFragmentManager.findFragmentByid/tag
4. 在fragment中使用菜单，要现在onCreat中调用setHasOptionsMenu(true)
5. 使用popBackStack()可以使当前fragmentManager中的fragment弹栈。
6. ListFragment：一个自带listview，和点击事件的fragment，布局包含listview，id为@id/android:list指定，
7. DialogFragment的使用：好处：屏幕旋转时的数据保存，可以直接使用AlertDialog和dialog重组

**TextView:**

1. textStyle:normal,bold(加粗)，italic(斜体)
2. lineSpacingExtra,行间距 ，lineSpacingMultiplier 行间距的倍数
3. shadowColor设置阴影颜色，
4. shadowRadius设置阴影模糊程度，设置0.1为字体颜色，建议设置3.0
5. shadowDx水平方向的偏移
6. shadowDy竖直方向的偏移
7. drawableXXX，（top，bottom。。）可以给textView设置不同方向的图片
8. autoLink,（all,emil,phone,）点击可以跳转
9. 可以使用Html.fromHtml来转换超文本
10. SpannableString & SpannableStringBuilder定制文本
11. EditText使用spannableString来输入图片
12. Button的状态的改变，stateListDrawable，使用selector标签设置选中状态，shape修改颜色。

**ListView和Adapter:**

1. Adapter:BaseAdapter的使用，ArrayAdapter,simpleDateAdapter
2. Adapter的优化：复用布局以及创建viewHolder.。adapter支持复用机制
3. Listview和button焦点抢占问题，解决问题给button设置focusable为false
4. 问题？切换fragment导致，listview数据重复？因为list集合重复添加了数据，在mainactivity中一直持有Fragment的引用，故list集合也就存活，而在viewCreate中初始化数据，导致会重复添加数据
5. ExpandableListView的简单使用以及BaseExpandableListAdapter
6. viewFlipper实现图片轮播图的两种方式。Xml中定义，代码中addView（）
7. 自定义重写onTouchEvent和手势实现ViewFlipper的滑动
8. GestureDetector手势的使用：1.onDown 2.onFling 快速移动，并松开 3.onScroll 4.onShowPress

5.onSingTabUp 手指离开触摸屏的一刹那

1. 如何在fragment中使用onTouchEvent？使用接口回调的方式在activity中进行事件的分发dispatchTouchEvent()中进行事件分发

**Android的数据存储**

1. 数据存储的四种方式：sharedPreferences,Sqlite,文件存储，contentProvider
2. sharedPreferences :是一种轻量性的数据存储，使用键值对存储。主要存储的是用户偏好，不能存储大量数据。保存为/data/data/<package\_name>/shared\_prefs/xxx.xml.
3. sharedPreferences:获取别的应用的数据，用的很少。1.首先调用方法获取context对象，createPackageContext(“”,)。2.在别的程序中创建sharedPreferences对象获取数据。

注意：权限为MODE\_WORLD\_READABLE

1. 使用MD5对sharedPreference数据进行加密：1.MD5不可逆。2.MD5不唯一，一个原始数据只对应一个MD5值，但是一个MD5值可以对应对个原始数据。
2. 文本存储：读取到文本中，位置在/data/data/<package\_name>/files/xxx.xx,其中两个重要的方法，1.openFileInput(fileName)返回输入流。2。openFileOutput(filename,mode)返回输出流，创建文本。
3. SD上的存储：上述的文本存储时存储到手机内存中，而大的数据则应该存储到sd卡上。
4. 数据库：SQLite的使用。：轻量型的数据库，支持事务操作。只支持5中数据类型，null,INTEGER,REAL,TEXT,BLOB
5. SQLite的事务：beginTransaction,setTransactionSuccessful,endTransaction,当数据写入数据库，然后提交成功时，数据写入数据库。否则，事件回滚，前面的事件不会成功。
6. SQlite的大数据存储：BLOB存储的是二进制的文件，如图片，视频等，一般我们不把数据存储在数据库中的，而是存储的是地址。
7. SQLite数据库升级方案：当数据库的版本号改变时会自动调用onUpgragde()方法进行升级。 1. 在使用onUpgragde方法中使用db.execSQL(" drop table my");

onCreate(db);

2.如果要新增一个column，使用Alter Table xx ADD COLUMN xx;

3.如果本版本为3，要从1升级到3 ，在onUpgragde中使用switch执行创建数据库的方法。

**Android中线程与异步**

1. Android在启动时会启动一个main线程，主线程，也叫做UI线程。在UI线程中不能做耗时操作，如网络请求，图片处理，数据库等，否则会造成线程的阻塞，超过5秒就会ANR.
2. UI更新：android中的ui更新不能在非主线程中进行操作，否则会报异常。所以这里就可以使用Handler类或者AsyncTask或者runOnUiThread()来进行ui的更新。
3. Handler：用来发送与处理消息，handle要正常工作时，在当前线程中要有一个Looper对象。

Message:handler接受处理的消息对象

MessageQueue:消息队列，先进先出管理Message,,初始化Looper时，会初始化一个messageQueue

Looper:每个线程只能够有一个Looper.管理MessageQueue,不断的将Message分发给对应的Handler处理。

1. Handler的两种方式：
2. 在主线程中：因为在主线程中，系统默认初始化了一个looper对象，所以直接创建handler对象就可以了。
3. 在子线程中：我们要自己初始化Looper对象，抵用Looper.prepare()方法创建按Looper对象，其构造默认初始化MessageQueue。创建Handler重写handleMessage方法。调用Looper.loop()启动Looper遍历MessageQueue.
4. AsyncTask:和handler一样用于处理异步操作的类，不过更加的轻量，后台是一个线程池。

几个重要的方法：onPreExecute(),初始化操作

doInbackground(params--), 线程池，异步操作

publishProgress(progess--),更新数据

onPostExecute(Result),结果值

onProgressUpdate(Porg),只有publish,,调用时才会调用

onCancelled()—用户取消线程操作，调用

1. AsyncTask注意：1. task实例必须在UI线程中创建

2. exectu方法必须在UI线程中创建

1. 不要手动调用opeExecute()..等方法
2. 该task实例只能被创建一次，对此调用会出现异常。
3. 线程池：分为四种，通过java的Executors提供。

Java通过Executors提供四种线程池，分别为：  
newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。  
newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

ExecutorService cachedThreadPool= Executors.newCachedThreadPool();

1. New Thread弊端 ：

a. 每次new Thread新建对象性能差。  
b. 线程缺乏统一管理，可能无限制新建线程，相互之间竞争，及可能占用过多系统资源导致死机或oom。  
c. 缺乏更多功能，如定时执行、定期执行、线程中断。  
相比new Thread，Java提供的四种线程池的好处在于：  
a. 重用存在的线程，减少对象创建、消亡的开销，性能佳。  
b. 可有效控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。  
c. 提供定时执行、定期执行、单线程、并发数控制等功能。

1. 线程同步：1.同步方法（java中每个对象java的每个对象都有一个内置锁，当用此关键字修饰方法时， 内置锁会保护整个方法。在调用该方法前，需要获得内置锁，否则就处于阻塞状态。）

2.同步代码块。（同步是一种高开销的操作，因此应该尽量减少同步的内容。

通常没有必要同步整个方法，使用synchronized代码块同步关键代码即可。）

3.使用volatile关键字，修饰field变量，成员变量。不能修饰final变量（可见性）

4.使用重入锁实现线程同步。Lock, ReentrantLock类是可重入、互斥、实现了Lock接口的锁. ReentrantLock() : 创建一个ReentrantLock实例

lock() : 获得锁

unlock() : 释放锁

1. 使用局部变量的方式，ThreadLocal类，

创建对象，ThreadLocal<Integer> l = new ThreadLocal<>();

方法：l.get()获得当前数据

2. initialValue ()初始化的数据

3.set()将此线程局部变量的当前线程副本中的值设置为value

**Android中的动画**

1. Android中的动画：帧动画，补间动画，属性动画三大类。当然还有其他的动画如：布局动画，android5.0揭露动画，转场动画，vector矢量动画等。
2. 帧动画：使用一连串的图片的动画，在xml使用标签animation-list设置图片，然后给imagview的背景设置该xml，在代码中启动动画。
3. 补间动画：四种，alpha，translate，rotate，scale。在xml,anim文件夹定义，代码中使用AniamtionUtils.loadAnimation();
4. 补间动画的插入器，Interpolator.
5. 补间动画的缺陷：view并不是真正的移动，旋转，缩放，和透明变化，只是绘图的变化而已，而view本身还在原来的位置。而且补间动画只能对view进行动画设置，有局限性。属性动画解决了这个问题。
6. 属性动画:Animator--- AnimatorSet, ValueAnimator,---ObjectAnimator,TimeAnimat
7. 属性动画实现移动，旋转缩放，透明等:

ObjectAnimator o = ObjectAnimator.offFloat(img,”alpha”,1f,0f,1f);,

第一个参数为任意对象，第二个参数为属性值，而且可以为任意属性，然后ObjectAnimator会去寻找alpha的get,set方法，有的话就给他进行设置。

所以view 的第二个参数可以使alpha，rotation，translationX/Y/Z，scaleX/Y.

1. 组合动画：使用AnimatorSet，的方法play(VlaueAnimator) 返回AnimatorSet.Builderde四个方法为：

after(Animator anim) 将现有动画插入到传入的动画之后执行

after(long delay) 将现有动画延迟指定毫秒后执行

before(Animator anim) 将现有动画插入到传入的动画之前执行

with(Animator anim) 将现有动画和传入的动画同时执行

1. 属性动画监听器：addListener(new AnimatorListenerAdapter)自己随意实现，动画的监听，开始，结束。
2. 属性动画xml编写：在res/animator/

标签animator 对应ValueAnimator , 标签objectAnimator 对应 ObjectAnimator

最后在代码中的实现时， Animator animator = AnimatorInflater.loadAnimator(context, R.animator.anim\_file);

animator.setTarget(view);

animator.start();

1. 属性动画的TypeEvaluator: 告诉动画如何从初始值过渡到结束值。上述使用的offInt和offFloat等都是使用的Int和Float系统内置的计数器，当然也可以自己定制 offObject,继承自TypeEvaluator。

内部evaluate(float fraction ,Object start ,Object end) fraction值为动画完成的进度

1. 属性动画Interpolator的使用：接口TimeInterpolator,里面一个getInterpolator（float input）方法，input是系统根据动画时长计算的值范围是0-1。而fraction的值就是更据input计算出来的。
2. ViewPropertyAnimator类在3.1时引入,也是只对于view使用，如textview.animate().appha(0f).setDuration(1000);自动调用

**Android中图片的处理**

1. Android中13中drawable的使用：
2. ColorDrawable:使用一个固定的颜色填充画笔来绘制到画布上。Xml为color标签
3. NinePatchDrawable，.9图，androidstudio。9图应该放在drawable目录下，xml中标签为 nine-patch
4. shapeDrawable,xml标签为shape,进行一些图形的变换
5. gradientDrawable,xml为gradient,其中的元素，angle为45度倍数，linear线性渐变才有效果，而gradientRadius,只有radial，sweep,发散和平铺渐变有效
6. BitmapDrawable：xml-<bitmap>,tileMode,三种模式，disable拉伸平铺，repaeat平铺，clamp原图大小，mirror镜像平铺
7. InsertDrawable：xml-<insert>将一个drawable插入另一个drawable中
8. ClipDrawable：xml,<clip>,android中的进度条就是这样实现的，从位图上裁剪一部分。通过level值控制，0~10000。
9. RotateDrawable：xml--<roatate>，旋转，也通过setLevel设置旋转，0~10000
10. AnimationDrawable：xml--<animation-list>,帧动画
11. LayerDrawable:xml--<layer-list>,绘制在不同的层。
12. TransitionDrawable:xml--<transition>,只管理两层的Drawable并且提供了透明度变化的动画，startTransiton方法启动切换动画。
13. LevelListDrawable:xml--<level-list>，通过设置不同的level来西那是图片。
14. stateListDrawable:xml--<selector>，状态改变。
15. Bitmap的相关概念：
16. Drawable：通用的图形对象，用于装载常用格式的图像，既可以是PNG，JPG这样的图像， 也是前面学的那13种Drawable类型的可视化对象！我们可以理解成一个用来放画的——画框！
17. Bitmap(位图)：我们可以把他看作一个画架，我们先把画放到上面，然后我们可以 进行一些处理，比如获取图像文件信息，做旋转切割，放大缩小等操作！
18. Canvas(画布)：如其名，画布，我们可以在上面作画(绘制)，你既可以用Paint(画笔)， 来画各种形状或者写字，又可以用Path(路径)来绘制多个点，然后连接成各种图形！
19. Matrix(矩阵)：用于图形特效处理的，颜色矩阵(ColorMatrix)，还有使用Matrix进行图像的 平移，缩放，旋转，倾斜等！
20. BitmapFactory类是bitmap工厂类，用来创建bitmap的实例，而bitmap类构造是私有的，不能实例化，只能使用jni实例化。

几个重要方法：1.decodeFile(str) 本地解析出来bitmap对象

2.decodeResource()资源中解析

3.decodeByteArray()从字节数组解析

4.decodeStream()从流中解析

1. BitmapFactory.Options类用来对bitmap进行设置的

boolean inJustDecodeBounds——如果设置为true，不获取图片，不分配内存，但会返回图片的高宽度信息。

int inSampleSize——图片缩放的倍数。如果设为4，则宽和高都为原来的1/4，则图是原来的1/16。

int outWidth——获取图片的宽度值

int outHeight——获取图片的高度值

int inDensity——用于位图的像素压缩比

int inTargetDensity——用于目标位图的像素压缩比（要生成的位图）

oolean inScaled——设置为true时进行图片压缩，从inDensity到inTargetDensity。

1. Bitmap方法：

***普通方法***

* public boolean **compress** (Bitmap.CompressFormat format, int quality, OutputStream stream) 将位图的压缩到指定的OutputStream，可以理解成将Bitmap保存到文件中！ **format**：格式，PNG，JPG等； **quality**：压缩质量，0-100，0表示最低画质压缩，100最大质量(PNG无损，会忽略品质设定) **stream**：输出流 返回值代表是否成功压缩到指定流！
* void **recycle**()：回收位图占用的内存空间，把位图标记为Dead
* boolean **isRecycled**()：判断位图内存是否已释放
* int **getWidth**()：获取位图的宽度
* int **getHeight**()：获取位图的高度
* boolean **isMutable**()：图片是否可修改
* int **getScaledWidth**(Canvas canvas)：获取指定密度转换后的图像的宽度
* int **getScaledHeight**(Canvas canvas)：获取指定密度转换后的图像的高度

***静态方法****：*

* Bitmap **createBitmap**(Bitmap src)：以src为原图生成可变得新图像
* Bitmap **createScaledBitmap**(Bitmap src, int dstWidth,int dstHeight, boolean filter)：以src为原图，创建新的图像，指定新图像的高宽以及是否变。
* Bitmap **createBitmap**(int width, int height, Config config)：创建指定格式、大小的位图
* Bitmap **createBitmap**(Bitmap source, int x, int y, int width, int height)以source为原图，创建新的图片，指定起始坐标以及新图像的高宽。
* public static Bitmap **createBitmap**(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)

1. 创建Bitmap对象的两种方式，1.使用BitmapFactory,,,,,,,,,2.使用BitmapDrawable
2. 截取图片的一部分，使用Bitmap.createBitmap()创指定大小的图片
3. oom问题，如何避免bitmap的oom？
   1. 采用低内存的编码方式，设置BitmapFactory.Options 的inPreferredConfig属性，默认为Bitmap.Config.ARGB\_8888,(每个像素占据8位#FFFFFFFF),默认设置为Bitmap.Config.ARGB-4444, ,(每个像素占据4位#fffff)
   2. 图片压缩:使用inSampleSize进行压缩
   3. 及时回收，设置为null,或者调用recycle
4. Paint使用
5. Canvas使用

Canvas(): 创建一个空的画布，可以使用setBitmap()方法来设置绘制具体的画布。

Canvas(Bitmap bitmap): 以bitmap对象创建一个画布，将内容都绘制在bitmap上，因此bitmap不得为null。

Drawxxx()方法。

Clipxxx();

Save()保存，然后就可以进行canvas的平移，缩放，旋转，裁剪等操作

Restore（）；恢复之前的状态，防止save后的行为影响。

translate(float dx, float dy)： 平移，将画布的坐标原点向左右方向移动x，向上下方向移动y.canvas的默认位置是在（0,0）

scale(float sx, float sy)：扩大，x为水平方向的放大倍数，y为竖直方向的放大倍数

rotate(float degrees)：旋转，angle指旋转的角度，顺时针旋转

1. Path路径：
2. SetMaskFilter(maskFilter)对画笔设置，其子类BlurMaskFilter,模糊效果，EmbossMaskFilter()浮雕效果。 在API14以上时，要关闭硬件加速，如activity根节点设置android:hardwareAccelerated="false"，或者view的ondraw,设置view.setLayerType(View.LAYER\_TYPE\_HARDWARE, null);
3. Xfermode的使用：子类PorterDuffXfermode(MODE m),给画笔设置，有18中模式。

1,ADD，两个区域全部范围，饱和度增加

2.Clear,啥都没有

3.DARKEN,两个区域全部范围，交集处颜色加深

4.MULTIPLY,两个区域的交集，颜色加深

5. DST，只保留目标图

6,SRC,只保留原图

7.SRC\_IN,,交际部分，保留原图

8.DST\_IN,交集部分，保留目标图

9.SRC\_OUT,补集，保留原图

10,.DST\_OUT，补集，保留目标图

11.SRC\_OVER, 两个区域全部范围，src在上

12.DST\_OVER, 两个区域全部范围，DST在上

13.SRC\_ATOP,保留交集图src部分和dst补集

14DST\_ATOP,保留交集图dst部分和src补集

15.SCREEN, 两个区域全部范围，交集处颜色透明

16.OVER\_LAY, 两个区域全部范围，交集处颜色叠加

17.XOR，两个区域交集之外范围

18.LIGHTEN取两图层全部区域，点亮交集部分颜色

1. 使用xfermode自定义view实现圆形头像，炫酷的字体view
2. Canvas中的save，是保存的坐标的状态，默认ALL\_SAVE\_FLAG,==MATRIX\_SAVE\_FLAG|CLIP\_SAVE\_FLAG

restore恢复的也是坐标的状态，图形不会清除，恢复最近的一次save保存，也就是栈顶。

而saveLayerXX()表示的是，重新再开一个图层，绘制。默认ALL\_SAVE\_FLAG,==MATRIX\_SAVE\_FLAG|CLIP\_SAVE\_FLAG|HAS\_ALPHA\_LAYER\_SAVE\_FLAG

RestoreToCount(int); 恢复栈中的指定的layer，其之上的layer出栈，清除

1. Canvas的剪切方法：clipPath/Rect/Region.

而canvas的剪切方法针对的时画布，而不是图形，只能在画图之前起作用，画图之后不起作用。

clipPath(path)根据路径剪切图片，可以制作圆形头像，剪切各种图形

clipRect（）剪切矩形的范围

clipRegion（）剪切一个区域范围，

Region.op值为:

REPLACE：不论A和B的集合状况，B的范围将全部进行显示，如果和A有交集，则将覆盖A的交集范围；***DIFFERENCE****：A和B的****差集****范围，即A - B，只有在此范围内的绘制内容才会被显示；*

***INTERSECT****：即A和B的****交集****范围，只有在此范围内的绘制内容才会被显示*

***UNION****：即A和B的****并集****范围，即两者所包括的范围的绘制内容都会被显示；*

***XOR****：A和B的****补集****范围，此例中即A除去B以外的范围，只有在此范围内的绘制内容才会被显示；*

***REVERSE\_DIFFERENCE****：B和A的****差集****范围，即B - A，只有在此范围内的绘制内容才会被显示；*

***REPLACE****：不论A和B的集合状况，B的范围将全部进行显示，如果和A有交集，则将覆盖A的交集范围；*

1. Paint的colorFilter的使用：3个子类

ColorMatrixColorFilter:饱和度，亮度

LightingColorFilter

PorterDuffColorFilter