# Activity的生命周期和启动模式

Activity确实是除了window，dialog，toast外能见到的界面

Configurations改变，activity会销毁重建

生命周期一定会执行完才进行下一个，所以不能做太耗时的工作

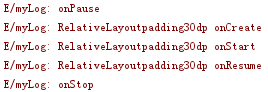
## onRestart注意事项

Onrestart一般是重新从不可见变到可见的时候。如切换home键到别的，又回来

再次回到原activity，onrestart，onstart，onresume

## 两个生命周期先后顺序

**两个activity，启动第二个activity，会先执行第一个的onPause方法，但是onstop方法不紧随其后，所以，不能在onPause中做耗时操作！**



## OnSaveInstanceState调用时机

onSaveInstanceState在onStop之前，但不知道是否在onPause前后

## 资源不足杀死activity优先级

前台Activity > 可见但非前台 > 后台Activity

## 隐式启动

### 需要同时匹配列表中的action、category、data信息，所有都要匹配，否则不能启动。

**Activity中可以有多个intent-filter，一个intent只要匹配任何一组，即可成功启动对应的activity**

### 匹配action

**一个过滤规则可以有多个action，那么只要Intent中的action能够和过滤规则中的任何一个action相同即可匹配成功。没有指定的action，那么匹配失败**

### 匹配category

**因为startactivity或者startactivityForResult的时候会默认为Intent加上“android.intent.category.DEFAULT”这个category，所以必须在intent-filter中指定这个category。**

**Intent可以没有category，也可以匹配成功，一旦有，必须要全部匹配**

### 匹配data

**跟action匹配类似**

**例如，<intent-filter>**

**<data android:mimeType=”video/mpeg” android:scheme=”http” …/>**

**<data android:mimeType=”audio/mpeg” android:scheme=”http”…/>**

**</intent-filter>**

**可以 intent.setDataAndType(Uri.parse(“http://abc”),”video/mpeg”)**

**或者**

**Intent.setDataAndType(Uri.parse(“http://abc”),”audio/mpeg”)**

### 可以用resolveActivity查看是否有没有匹配的activity

# Ipc

## 多进程

Process=“：remote”

Process=“com.example.name.remote”

以“：”开头的进程属于当前应用的私有进程，其他应用的组件不可以和它跑在同一个进程中，不以：开头的进程属于全局进程，其他应用通过ShareUID方式可以和它跑在同一个进程中

### 使用多进程会出现以下问题

1 静态成员和单例模式完全失效

2 线程同步机制完全失效

3 SharePreference的可靠性下降

**4 Application会创建多次**

## Serializable

Private static final long seriaVersionUID，这个值是用来判定是否可以序列化，如果不定义这个值，当类成员变量改动（删除或添加后）那么就好反序列化失败，当定义这个值后，类成员细微改动还是可以序列化成功。

## Binder

Messenger底层是AIDL

## Messenger

Messenge不能传递parcelable对象！！！只能传递系统的！

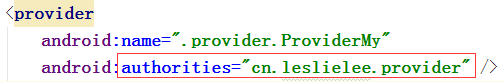
**Ipc中传递的对象不会是同一个，所以通过判断是否是同一个对象进行操作是不可取的。用RemoteCallbackList**

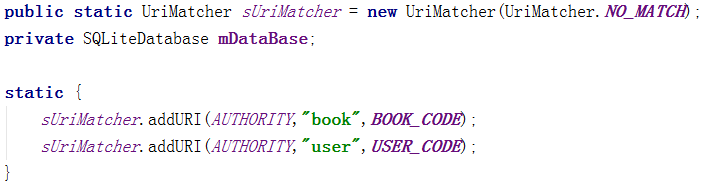
## ContentProvider

继承ContentProvider，操作数据库继承SQLiteOpenHelper，ContentProvider中完成增删查改，用UriMatch判断，查询的是哪个表。

SQLIteOpenHelper，用来获取操作数据库的引用。

（自己写过DEMO的。忘了去翻）







## Socket

Server：

**ServerSocket serverSocket=new ServerSocket（port）；**

//获取客户端连接

**Socket client=serverSocket.accept();**//好像是阻塞

然后获取客户端连接的输入流输出流，输出

Client：

Socket socket=new Socket（”ip”,port）;

然后获取socket的输入流

Eg：new PrintWriter(new BufferedWriter(new OutpetStreamWriter(**socket.getOutputSteam()**)),true)

## Ipc使用总类

### Bundle

### 文件共享

### AIDL

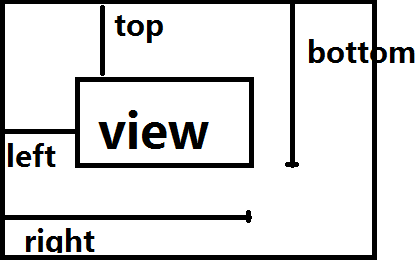
### Messenger

### ContentProvider

### Socket

# View的事件体系

## View的位置参数



**需要注意的是，View在平移的过程中，top和left表示的是原始左上角的位置信息，其值并不会改变，此时发生改变的是x,y、translationX和translationY**

## motionEvent和TouchSlop

MotionEvent，手指接触屏幕后，产生一系列事件，可以获得getX和getY或者getRawX和getRawY获取坐标，第一组获得的是相对于view左上角的坐标，第二组是相对于手机屏幕左上角的坐标

Touchslop是系统所能识别出的被认为是滑动的最小距离。可以用

ViewConfiguration.get(getContext()).getScaledTouchSlop();

## VelocityTracker、GestureDetector和Scroller

### VelocityTracker，速度追踪

VelocityTracker velocityTranker = VelocityTracker.obtain();

velocityTracker.addMovement(event).

**获取速度前，一定要先计算一遍速度！！！速度可以为负**

**velocityTracker.computeCurrentVelocity(time);//然后再获取速度**

**速度=（终点位置-起点位置）/时间段**

**velocityTracker.getXVelocity();**

**velocityTracker.getYVelocity();**

### GestureDetector

手势检测，用于辅助检测用户的单机、滑动、长按、双击等行为

使用，接管View的onTouchEvent行为，

Boolean consume=mGestureDetector.onTouchEvent(event);

Return consume;

### Scroller

用于实现View的弹性滑动。使用View的ScrollBy/scrollTo进行滑动，是瞬间完成的。

Scroller本身不能让View实现弹性滑动，需要和View的computeScroll配合

## View的滑动

### View的三种滑动：

1 通过View自身的ScrollerTo/ScrollerBy

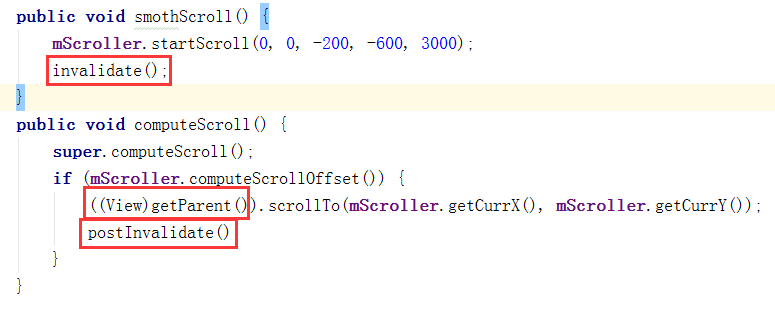
2 给View添加平移效果实现滑动

3 通过改变View的LayoutParam

**ScrollBy和scrollTo只能改变View内容的位置，不能改变View在布局中的位置！！**

**所以用scroller的startScroll,在view自己中使用是没有用的！！！需要获取parent，在开始scroll！！**

**还有滑动的时候必须要配合invalid！**



## View的事件分发

Public Boolean dispatchTouchEvent

Public Boolean onInterceptTouchEvent

Public Boolean onTouchEvent

onTouchEvent在dispatchTouchEvent中调用

### 三个函数的关系，伪代码

Public Boolean dispatchTouchevent（MotionEvent ev）{

Boolean consume = false;

If(onInterceptTouchEvent(ev)){

Consume=true;

}else{

Comsume=child.dispatchTouchEvent(ev);

}

Return consume;

}

### onTouchListener,onTouchEvent,onClickListener优先级

onTouchListener中的onTouch返回true，那么onTouchevent不会被调用

onClickListener优先级最低，在事件传递的最低端

事件传递顺序activity->window->View

Window的唯一实现类是phoneWindow

### 外部拦截

**父容器在onInterceptTouchEvent方法中，首先不能拦截Action\_Down，因为拦截了这个，那么move和up，不管怎么样都不能传递给子元素了，Action\_Move可以根据需要来决定是否拦截，最后是Action\_Up,这里必须要返回false，因为Action\_Up事件本身没有太多意义**

### 内部拦截

**同样action\_Down事件不拦截。**

**Parent.requestDisallowInterceptTouchEvent(true)//不拦截**

# View的工作原理

## 初识ViewRoot和DecorView

ViewRoot对应于ViewRootImpl类，他是连接WindowManager和DecorView的纽带，View的三大流程均是通过ViewRoot来完成的。

## 理解MeasureSpec

View的LayoutParams根据父容器所施加的规则转换成对应的MeasureSpec，然后再根据这个MeasureSpec来测量出View的宽高，但是不一定是最终宽高，**因为View的最终大小在layout中确定**

### MeasureSpec

32位的int值，高2位表示SpecMode，低30位表示SpecSize

SpecMode三类

**UNSPECIFIED**:父容器不对View有任何限制，要多大给多大，这种情况一般用于系统内部，表示一种测量状态

**EXACTLY**:父容器已经检测出View所需要的精确大小，这个时候View的最终大小就是SpecSize所指定的值，match\_parentm模式

**AT\_MOST**:父容器指定了一个可用大小，即SpecSIze，大小不定，但View的大小不能大于这个值，相当于wrap\_content.

MeasureSpec一旦确定后，onMeasure中就可以确定View的测量宽高

## View的工作流程

View的工作流程主要指measure，layout，draw三大流程

## 在activity中获取view宽高的四中方法

1 activity/View#onWindowFocusChanged（会被调用多次）

2 view.post(runnable),通过post将一个runnable投递到消息队列的尾部，然后等待Looper调用此runnable的时候，view也已经初始化好了。

3 viewTreeObserver 的众多回调可以完成这个功能，比如用OnGlobalLayoutListener（实际开发中自己有用到，判断textview）（这个也会被调用多次）

4 view.measure(),通过手动调用测量获得view的宽高，（不懂）

## Layout过程

Layout是确定view本身的位置，onLayout方式则会确定所有子元素的位置

## Draw过程

View的绘制过程遵循如下几步

1. 绘制背景background.draw(canvas)
2. 绘制自己（ondraw）
3. 绘制children（dispatchDraw）
4. 绘制装饰（onDrawScrillBars）

有一个setWithNoDraw这个方法，view默认不启用，ViewGroup默认会启用，当明确知道一个ViewGroup需要通过onDraw来绘制内容的时候，需要显式的关闭WILL\_NOT\_DRAW这个标记

## 自定义View

### 自定义view的分类

1 继承View重写onDraw的方法

2 继承ViewGroup派生特殊的layout

1. 继承特定的view（如Textview）

4 继承特定的viewGourp（如LinearLayout）

### 自定义view须知

1 让view支持wrap\_content

2 支持padding

3 尽量不要在view中使用Handler

1. view中如果有线程或者动画，需要及时停止，onDEtachedFromWIndow回调。
2. view带有滑动嵌套情形时，需要处理好滑动冲突

# 理解remoteViews

RemoteViews在通知栏上的应用(notification)

RemoteVIews在桌面小部件的应用

## pendingIntent

pendingIntent支持三种待定意图：启动activity，启动Service和发送广播，相当与启动acitivty，启动service和发送广播

pendingIntent匹配规则:如果两个PendingIntent他们内部的Intent相同并且requestCode也相同，那么两个PendingIntent就是相同的。这里Intent相同指的是，componentName和intent-filter都相同。

### PengdingIntent的四个Flag

#### FLAG\_ONE-SHOT

当前描述的PendingIntent只能被使用一次，然后它就会被自动calcel，如果后续还有相同的pendingIntent，那么他们的send方法就会调用失败。

#### FLAG\_NO\_CREATE

当前描述的PendingIntent不会主动创建，如果当前PendingIntent之前不存在，那么getActivity,getService和getBroadcast方法会直接放回null。

#### FLAG\_CALCEL\_CURRENT

当前描述的PendingIntent如果已经存在，那么他们都会被cancel，然后系统会创建一个新的PendingIntent，对于通知栏消息来说，那些cancel的消息单击后将无法打开

#### FLAG\_UPDATE\_CURRENT

当前描述的PendingIntent如果已经存在，那么它们都会被更新

# Android中的drawable

Drawable的内部宽高，通过getInstinsicWidth 和getIntrinsicHeight，不是所有的Drawable都有宽高，图片的Drawable，宽高就是图片的宽高，想颜色的Drawable就没有宽高。Drawable的内部宽高不等同于它的大小。一般来说，Drawable是没有大小概念，当作为背景的时候，大小就是view的大小

## shapeDrawable

这个就ring比较特殊，有特殊的属性，需要android：useLevel设置false才能显示

## LayoutDrawable

<layer-list>层次化的drawable的集合，有层次概念，下面的item会覆盖上面的item。

## stateListDrawable

<selector>也是drawable集合，每个drawable对应view的一种状态。

## levelListDrawable

## transitionDrawable

对应<transition> 用于实现两个Drawable之间的淡入淡出效果，设置背景后，通过

TransitionDrawable drawable=(TransitionDrawable)View.getBackground();

Drawable.startTransition(time);

## insetDrawablw

InsetDrawable对应<inset>标签，他可以将其他Drawable内嵌到自己当中，并可以在四周流出一定的间距，通过layerDrawable也可以解决

## ScaleDrawable

对应<scal>标签，可以根据自己等级将指定的drawable缩放到一定比例.

注意，等级为0的时候ScalDrawable不可见，这是默认值。

## ClipDrawable

对应<clip>标签，可以根据自己当前的等级来裁剪另一个Drawable