目录

[1.singleTask是出生在一个实例中，允许其他Activity存在于这个task中而singleInstance出生在在一个Task中而且独占这个Activity 1](#_Toc385353619)

[2. android.os.Process.killProcess(pid) 和 System.exit(int code) 1](#_Toc385353620)

[3: yangziPreviousVolumelsScrollLayout和scrollLayoutContainer.getChildAt(1) 的区别 2](#_Toc385353621)

[4:ScrollLayout 要在addView之后设置 setToScreen之后才能看见效果 2](#_Toc385353622)

[5:改变图片的大小 2](#_Toc385353623)

[6．想要容器来装载第2个子View时，容器应该选择FrameLayout 3](#_Toc385353624)

[7．Android分辨率控制 3](#_Toc385353625)

[8. 代码控制进入入口Activity 3](#_Toc385353626)

[9 .打开assets中的图片，并设置背景 3](#_Toc385353627)

[10.判断SD卡是否挂载 4](#_Toc385353628)

[11.检查SD卡剩余空间,单位是byte 4](#_Toc385353629)

[12. ScrollLayout布局问题 4](#_Toc385353630)

[13.fragment的生命周期 4](#_Toc385353631)

[14．清除cookie 5](#_Toc385353632)

[15.theme主题文件 5](#_Toc385353633)

[16.attr属性文件的作用 6](#_Toc385353634)

[17.saveInstance机制 7](#_Toc385353635)

[18.view的绘制过程 9](#_Toc385353636)

[19.SlideMenu小结 10](#_Toc385353637)

[20.获取本地化语言 10](#_Toc385353638)

[21.3.0以上menu 10](#_Toc385353639)

[22.屏幕密度 density 11](#_Toc385353640)

[23.获取当前主题下定义的某个属性的值 11](#_Toc385353641)

[24.为单独view设置上下文菜单 11](#_Toc385353642)

[25.selector 11](#_Toc385353643)

[26:判断某个activity时候存在 12](#_Toc385353644)

[27.自定义自己的style 12](#_Toc385353645)

[28 Canvas和Matrix 13](#_Toc385353646)

# 1.singleTask是出生在一个实例中，允许其他Activity存在于这个task中而singleInstance出生在在一个Task中而且独占这个Activity

# 2. android.os.Process.killProcess(pid) 和 System.exit(int code)

需要注意的是： android.os.Process.killProcess(pid) 和 System.exit(int code)会导致进程非正常退出，进程退出时不会去执行Activity的onPause、onStop和onDestroy方法，那么进程很有可能错过了保存数据的机会。因此，这两个方法最好使用在出现异常的时候！

# 3: yangziPreviousVolumelsScrollLayout和scrollLayoutContainer.getChildAt(1) 的区别

objs[1] =yangziPreviousVolumelsScrollLayout;

和

objs[1] = scrollLayoutContainer.getChildAt(1);

的区别 是父将子移除，重新生成新的，虽然yangziPreviousVolumelsScrollLayout所对应的内存块还存在但是 已经和scrollLayoutContainer脱离关系

# 4:ScrollLayout 要在addView之后设置 setToScreen之后才能看见效果

# 5:改变图片的大小

android.view.ViewGroup.LayoutParams lp = imageView.getLayoutParams();

**if**(advertise.abstracz != **null**)

{

lp.height = layoutHeight / 3 + 10;

lp.width = (**int**) (lp.height \* 1.1);

}**else**

{

lp.height = LayoutParams.*FILL\_PARENT*;

lp.width = LayoutParams.*FILL\_PARENT*;

}

# 6．想要容器来装载第2个子View时，容器应该选择FrameLayout

# 7．Android分辨率控制

对于不同分辨率，不同尺寸屏幕大小使用dp值，再加上代码控制布局位置可以控制整体布局

而对于相同分辨率，但是不同屏幕尺寸，不仅要根据代码控制布局，还要通过设置px值来控制字体的大小

# 8. 代码控制进入入口Activity

Intent intent = **new** Intent(Intent.*ACTION\_MAIN*);

intent.addCategory(Intent.*CATEGORY\_LAUNCHER*);

intent.setComponent(**new** ComponentName(getPackageName(), getPackageName() + ".app.entry.BootActivity"));

startActivity(intent);

# 9 .打开assets中的图片，并设置背景

InputStream in = getResources().getAssets().open("boot.png");

SoftReference<Drawable> drawable = **new** SoftReference<Drawable>(**new** BitmapDrawable(in));

layout.setBackgroundDrawable(drawable.get());

# 10.判断SD卡是否挂载

**public** **boolean** isSDCardMounted() {

String state = Environment.*getExternalStorageState*();

**if**(!Environment.*MEDIA\_MOUNTED*.equals(state)) {

**return** **false**;

}

**return** **true**;

}

# 11.检查SD卡剩余空间,单位是byte

**public** **long** freeSize() {

File file = Environment.*getExternalStorageDirectory*();

StatFs sf = **new** StatFs(file.getPath());

**return** ((**long**)sf.getAvailableBlocks())\*((**long**)sf.getBlockSize());

}

# 12. ScrollLayout布局问题

要保证它的父类以及父类的父类布局要明确，是具体的大小或者是fill\_parent

# 13.fragment的生命周期

activity:onCreate

fragment:onAttach

fragment:onCreate

fragment:onCreateView

fragment:onActivityCreated

fragment:onStart

activity:onStart

activity:onResume

fragment:onResume

fragment:onPause

activity:onPause

# 14．清除cookie

//清除cookie \*\*\*\*\*\*\*\*\*\*\*\*务必清除cookie

CookieSyncManager.*createInstance*(getApplicationContext());

CookieManager.*getInstance*().removeAllCookie();

# 15.theme主题文件

在themes.xml文件中<style name="Theme.Sherlock.Light.Dialog"/>

在代码中引用到该样式 的变量名是 Theme\_Sherlock\_Light\_Dialog

在我们的布局文件中引用样式或者注意写法有点区别：

前者：style="@style/MyAttrsViewStyle" MyAttrsViewStyle是定义的样式名

后者：style="?attr/actionBarStyle" actionBarStyle是在theme中定义的样式

Style文件和theme可以相互引用各自的属性值（字段标签都是在attr文件中定义的）

# 16.attr属性文件的作用

可以为布局(layout)文件提供属性,也可以为主题(theme)文件提供属性\

不同的Activity的布局或者属性如何才能找到自己相对应的属性所设置的值，以下提供两个方法:

1.定义自己的view

TypedArray a = context.obtainStyledAttributes(attrs,

R.styleable.*PullToRefresh*);

**int** textColor = a.getColor(R.styleable.*PullToRefresh\_textColor*,

0XFFFFFFFF);

**float** textSize = a.getDimension(R.styleable.*PullToRefresh\_textSize*, 36);

attrs是定义VIEW时的构造函数的属性参数

2.定义的theme文件

TypedArray a = mActivity.getTheme().obtainStyledAttributes(R.styleable.*SherlockTheme*);

mIsFloating = a.getBoolean(R.styleable.*SherlockTheme\_android\_windowIsFloating*, **false**);

此之前是通过对Activity设置theme,再通过Activity获取theme文件

# 17.saveInstance机制

先看Application Fundamentals上的一段话：

 Android calls onSaveInstanceState() before the activity becomes vulnerable to being destroyed by the system, but does not bother calling it when the instance is actually being destroyed by a user action (such as pressing the BACK key)

从这句话可以知道，当某个activity变得“容易”被系统销毁时，该activity的onSaveInstanceState就会被执行，除非该activity是被用户主动销毁的，例如当用户按BACK键的时候。

注意上面的双引号，何为“容易”？言下之意就是该activity还没有被销毁，而仅仅是一种可能性。这种可能性有哪些？通过重写一个activity的所有生命周期的onXXX方法，包括onSaveInstanceState和onRestoreInstanceState方法，我们可以清楚地知道当某个activity（假定为activity A）显示在当前task的最上层时，其onSaveInstanceState方法会在什么时候被执行，有这么几种情况：

1、当用户按下HOME键时。

这是显而易见的，系统不知道你按下HOME后要运行多少其他的程序，自然也不知道activity A是否会被销毁，故系统会调用onSaveInstanceState，让用户有机会保存某些非永久性的数据。以下几种情况的分析都遵循该原则

2、长按HOME键，选择运行其他的程序时。

3、按下电源按键（关闭屏幕显示）时。

4、从activity A中启动一个新的activity时。

5、屏幕方向切换时，例如从竖屏切换到横屏时。

在屏幕切换之前，系统会销毁activity A，在屏幕切换之后系统又会自动地创建activity A，所以onSaveInstanceState一定会被执行

总而言之，onSaveInstanceState的调用遵循一个重要原则，即当系统“未经你许可”时销毁了你的activity，则onSaveInstanceState会被系统调用，这是系统的责任，因为它必须要提供一个机会让你保存你的数据（当然你不保存那就随便你了）。

**至于onRestoreInstanceState方法，需要注意的是，onSaveInstanceState方法和onRestoreInstanceState方法“不一定”是成对的被调用的，（本人注：我昨晚调试时就发现原来不一定成对被调用的！）**

onRestoreInstanceState被调用的前提是，activity A“确实”被系统销毁了，而如果仅仅是停留在有这种可能性的情况下，则该方法不会被调用，例如，当正在显示activity A的时候，用户按下HOME键回到主界面，然后用户紧接着又返回到activity A，这种情况下activity A一般不会因为内存的原因被系统销毁，故activity A的onRestoreInstanceState方法不会被执行

另外，onRestoreInstanceState的bundle参数也会传递到onCreate方法中，你也可以选择在onCreate方法中做数据还原。

至于这两个函数的使用，给出示范代码（留意自定义代码在调用super的前或后）：

@Override  
public void onSaveInstanceState(Bundle savedInstanceState) {  
        savedInstanceState.putBoolean("MyBoolean", true);  
        savedInstanceState.putDouble("myDouble", 1.9);  
        savedInstanceState.putInt("MyInt", 1);  
        savedInstanceState.putString("MyString", "Welcome back to Android");  
        // etc.  
        super.onSaveInstanceState(savedInstanceState);  
}  
  
@Override  
public void onRestoreInstanceState(Bundle savedInstanceState) {  
        super.onRestoreInstanceState(savedInstanceState);  
  
        boolean myBoolean = savedInstanceState.getBoolean("MyBoolean");  
        double myDouble = savedInstanceState.getDouble("myDouble");  
        int myInt = savedInstanceState.getInt("MyInt");  
        String myString = savedInstanceState.getString("MyString");  
}

# 18.view的绘制过程

在setContentView的时候将整个视图对象树给构建出来，调用各个View或者ViewGroup的构造函数

而onMeasure(),onLayout(),onDraw()都是在Activity的onCreate()执行完成之后开始

当前自己的view先onDraw 然后在dispatchDraw 画自己的子类，可以由系统自己完成

而事件分发过程（每一次事件传递过来都是这个顺序） 是先dispatchTouchEvent 先分发，如果是viewgroup的话，先onInterceptTouchEvent(只调一次)，再onTouchEvent

# 19.SlideMenu小结

BaseActivity:①:构造SlideMenu

a:构造CustomViewAbove

b:构造CustomViewBehind

②:构造CustomViewBehind的内容(Fragment)

③:设置SlideMenu的一些参数属性供CustomViewAbove和CustomViewBehind布局和绘制使用

# 20.获取本地化语言

Locale.*getDefault*().getLanguage().equals("en")

# 21.3.0以上menu

1.将menu显示在脚部

<item android:id=*"@+id/menu\_add"*

android:icon=*"@drawable/ic\_menu\_compose"*

android:title=*"@string/menu\_add\_button"*

android:showAsAction=*"always"* />

# 22.屏幕密度 density

TextView tvcolor = **new** TextView(NoteView.**this**);

tvcolor.setPadding(0, 5, 0, 0);

tvcolor.setWidth((**int**)(mContext.getResources().getDisplayMetrics().density \* 6 +0.5f));

tvcolor.setHeight((**int**)(mContext.getResources().getDisplayMetrics().density \* 48 +0.5f));

比如同样尺寸的屏幕 ，分辨率是480 \* 800 ，它的密度是1

分辨率是 960\*1600 ，它的密度就为2

# 23.获取当前主题下定义的某个属性的值

a 代码获取

TypedArray ta = **this**.getTheme()

.obtainStyledAttributes(**new** **int**[] {

com.android.internal.R.attr.*scrollbarSize*

});

int *rightPadding* = ta.getDimensionPixelSize(0, 6);

ta.recycle();

b xml文件

android:paddingRight="?android:attr/scrollbarSize"

# 24.为单独view设置上下文菜单

registerForContextMenu 🡪 onCreateContextMenu

# 25.selector

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<selector xmlns:android=*"http://schemas.android.com/apk/res/android"*>

<item android:state\_enabled=*"false"*

android:drawable=*"@drawable/btn\_fm\_micro\_disable"* />

<item android:state\_pressed=*"true"*

android:drawable=*"@drawable/btn\_fm\_micro\_pressed"* />

<item android:state\_focused=*"true"*

android:drawable=*"@drawable/btn\_fm\_micro\_pressed"* />

<item android:drawable=*"@drawable/btn\_fm\_micro"* />

</selector>

注意：android:state\_enabled=*"false"*

下面一项不写android:state\_enabled=*"true"* 如果写了会导致pressed和focus没有效果

# 26:判断某个activity时候存在

PackageManager pm = getPackageManager();

Intent intent = **new** Intent();

ComponentName compoentName =

**new** ComponentName("com.tinno.otacheck","com.tinno.otacheck.OtaCheckActivity");

intent.setComponent(compoentName);

ResolveInfo ri = pm.resolveActivity(intent, PackageManager.*MATCH\_DEFAULT\_ONLY*);

**if**(ri != **null**)

{

startActivityForResult(intent, 888);

}

# 27.自定义自己的style

1．在values文件夹下创建一个名为styles.xml的xml文件，在xml文件中进行定义：

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<resources>

<style name=*"widthAndheight"*>

<item name=*"android:layout\_width"*>fill\_parent</item>

<item name=*"android:layout\_height"*>wrap\_content</item>

</style>

</resources>

2．引用style中的name

<TextView

android:text=*"@string/title01"*

android:id=*"@+id/TextView01"*

style=*"@style/ widthAndheight "*>

</TextView>

# 28 Canvas和Matrix

抽象的说pre方法是向前"生长", post方法是向后"生长"，具体拿个例子来说，比如一个matrix调用了下列一系列的方法：

matrix.preScale(0.5f, 1);  matrix.preTranslate(10, 0); matrix.postScale(0.7f, 1);  matrix.postTranslate(15, 0); 则坐标变换经过的4个变换过程依次是：

**translate(10, 0) -> scale(0.5f, 1) -> scale(0.7f, 1) -> translate(15, 0),**

所以对matrix方法的调用顺序是很重要的，不同的顺序往往会产生不同的变换效果。pre方法的调用顺序和post方法的互不影响，即以下的方法调用和前者在真实坐标变换顺序里是一致的， matrix.postScale(0.7f, 1); matrix.preScale(0.5f, 1);  matrix.preTranslate(10, 0); matrix.postTranslate(15, 0);

而matrix的set方法则会对先前的pre和post操作**进行刷除**，而后再设置它的值，比如下列的方法调用:

matrix.preScale(0.5f, 1);  matrix.postTranslate(10, 0); matrix.**setScale**(1, 0.6f); matrix.postScale(0.7f, 1); matrix.preTranslate(15, 0); 其坐标变换顺序是

**translate(15, 0) -> scale(1, 0.6f) ->  scale(0.7f, 1).**

**setScale重新设置了矩阵的值，之前的两个变换是无效的了，所以最终的显示效果只有三个变换效果。**

**Canvas里scale, translate, rotate, concat方法都是pre方法**，如果要进行更多的变换可以先从canvas获得matrix, 变换后再设置回canvas.