[1.环境搭建，开发相关 3](#_Toc28100729)

[1.1背景相关与系统架构分析 3](#_Toc28100730)

[1.1.1系统特性： 3](#_Toc28100731)

[1.1.2架构的简单理解： 3](#_Toc28100732)

[1.2 开发环境搭建 3](#_Toc28100733)

[1.2.1 环境搭建 3](#_Toc28100734)

[1.2.2相关术语的解析 3](#_Toc28100735)

[1.2.3 打包 3](#_Toc28100736)

[1.3 Git版本管理 3](#_Toc28100737)

[1.3.1 Git是什么 3](#_Toc28100738)

[1.3.2 创建本地仓库与代码托管 3](#_Toc28100739)

[1.4 界面原型设计 3](#_Toc28100740)

[1.5 工程相关解析 3](#_Toc28100741)

[1.5.1 res 3](#_Toc28100742)

[1.5.2 java 3](#_Toc28100743)

[1.6 Android程序签名打包 3](#_Toc28100744)

[1.6.1 什么是签名，有什么用 3](#_Toc28100745)

[1.6.2 如何签名 3](#_Toc28100746)

[1.7 反编译apk获得代码与资源 3](#_Toc28100747)

[2.常用UI控件与事件处理机制 4](#_Toc28100748)

[2.1 View与ViewGroup的概念 4](#_Toc28100749)

[2.2布局 4](#_Toc28100750)

[2.2.1 ConstraintLayout 约束布局 4](#_Toc28100751)

[2.2.1.1 为什么要用ConstraintLayout 4](#_Toc28100752)

[2.2.1.2 约束布局的使用方法 4](#_Toc28100753)

[2.2.2 LinearLayout 线性布局 4](#_Toc28100754)

[2.2.2.1 权重 4](#_Toc28100755)

[2.2.2.2 分割线 4](#_Toc28100756)

[2.2.3 RelativeLayout 相对布局 4](#_Toc28100757)

[2.2.3.1 根据父容器定位 4](#_Toc28100758)

[2.2.3.2 根据兄弟组件定位 4](#_Toc28100759)

[2.2.3.2 margin与padding的区别 4](#_Toc28100760)

[2.2.4 TableLayout 表格布局 4](#_Toc28100761)

[2.2.4.1 如何确定行数和列数 4](#_Toc28100762)

[2.2.4.2 常用属性 4](#_Toc28100763)

[2.2.5 FrameLayout 帧布局 4](#_Toc28100764)

[2.2.5.1 常用属性 4](#_Toc28100765)

[2.2.6 GridLayout 网格布局 4](#_Toc28100766)

[2.2.7 AbsoluteLayout 绝对布局 4](#_Toc28100767)

[2.3 常用基本控件 4](#_Toc28100768)

[2.4 日期类控件 4](#_Toc28100769)

[2.5 adapter控件 4](#_Toc28100770)

[2.6 提示类控件 4](#_Toc28100771)

[2.7 菜单类控件 4](#_Toc28100772)

[2.8 其他控件 4](#_Toc28100773)

[2.9 事件处理的机制 4](#_Toc28100774)

[3.Android四大组件和Intent组件 4](#_Toc28100775)

[3.1 Actviity 活动 4](#_Toc28100776)

[3.1.1 Activity的概念与Activity的生命周期图 4](#_Toc28100777)

[3.1.2 Activity的创建流程 4](#_Toc28100778)

[3.1.3 onCreate() 参数详解 4](#_Toc28100779)

[3.1.4 启动一个Activity的几种方式 4](#_Toc28100780)

[3.1.4.1 显示启动 4](#_Toc28100781)

[3.1.4.2 隐式启动 4](#_Toc28100782)

[3.1.4.3 绝对路径启动 4](#_Toc28100783)

[3.1.5 横竖屏切换与状态保存的问题 4](#_Toc28100784)

[3.1.6 Activity间的数据传递 4](#_Toc28100785)

[3.1.7 完全退出App的方法 5](#_Toc28100786)

[3.1.8 Activity的四种加载模式详解 6](#_Toc28100787)

[3.1.8.1 standard模式 6](#_Toc28100788)

[3.1.8.2 singleTop模式 6](#_Toc28100789)

[3.1.8.3 singleTask模式 7](#_Toc28100790)

[3.1.8.4 singleInstance模式 7](#_Toc28100791)

[3.1.9 activity线程之间的关系 7](#_Toc28100792)

[3.2 Service 服务 7](#_Toc28100793)

[3.3 BroadcastReceiver 广播 7](#_Toc28100794)

[3.4 ContentProvider 内容提供者 7](#_Toc28100795)

[3.5 Intent 意图 7](#_Toc28100796)

[4.Fragment 7](#_Toc28100797)

[5.数据存储 7](#_Toc28100798)

[5.1 文件存储读写 7](#_Toc28100799)

[5.2 SharedPreference 7](#_Toc28100800)

[5.3 SQLite 7](#_Toc28100801)

[6.网络编程 7](#_Toc28100802)

[6.1 HTTP详解 7](#_Toc28100803)

[6.2 XML数据解析 7](#_Toc28100804)

[6.3 Json数据解析 7](#_Toc28100805)

[6.4 Android文件上传下载 7](#_Toc28100806)

[6.5 调用WebService 7](#_Toc28100807)

[6.6 WebView 7](#_Toc28100808)

[6.7 Socket 7](#_Toc28100809)

[7.绘图与动画 8](#_Toc28100810)

[7.1 Drawable 8](#_Toc28100811)

[7.2 Bitmap 8](#_Toc28100812)

[7.3 绘图相关API 8](#_Toc28100813)

[7.4 动画 8](#_Toc28100814)

[8.多媒体开发 8](#_Toc28100815)

[8.1 SoundPool播放音效 8](#_Toc28100816)

[8.2 MediaPlayer播放音视 8](#_Toc28100817)

[8.3 Careme拍照 8](#_Toc28100818)

[8.4 MediaRecord录音 8](#_Toc28100819)

[9.系统API，地图定位 8](#_Toc28100820)

[9.1 TelephonManager 电话管理器 8](#_Toc28100821)

[9.2 SmsManager短信管理器 8](#_Toc28100822)

[9.3 AudioManager 音频管理器 8](#_Toc28100823)

[9.4 Vibrator震动器 8](#_Toc28100824)

[9.5 AlarmManager 闹钟服务 8](#_Toc28100825)

[9.6 PowerManager 电源服务 8](#_Toc28100826)

[9.7 WindowManager 窗口管理服务 8](#_Toc28100827)

[9.8 LayoutIflater布局管理器 8](#_Toc28100828)

[9.9 WallpaperManager壁纸管理器 8](#_Toc28100829)

[9.10 传感器 8](#_Toc28100830)

[9.11 GPS 8](#_Toc28100831)

[10.项目实战 8](#_Toc28100832)

# 1.环境搭建，开发相关

## 1.1背景相关与系统架构分析

我们可能有这样的一个疑问： 那么多的系统版本，我们开发的时候要针对哪个版本进行开发？ 这就是作为一个Android必须面对的Android的"碎片化"问题了，而这个问题又分为两个： ①系统碎片化：我们开发App时可能需要做到低版本兼容，比如，最低兼容至2.3版本；由于各种Rom定制的盛行，国人都喜欢对原生系统做一些更改，这导致了在原生系统上可行，而在定制Rom上不可行的问题，比如相机调用~ ②屏幕碎片化：市面上各种各样屏幕尺寸的手机，4.3寸，4.5寸，4.7寸，5.0寸，5.3寸...等等，除了手机外，还有Android平板，所以开发时我们可能要处理这个屏幕适配的问题，当然，刚学我们并不需要去考虑这些复杂的东西，后续实际开发我们再来深究！

### 1.1.1系统特性：

**1.** 应用程序框架支持组件的重用与替换（app发布时遵守了框架的约定，其他app也可以使用该模块）

**2.**Dalvik虚拟机:专门为移动设备优化 -集成的浏览器:开源的WebKit引擎

**3.**SQLite结构化的数据存储

**4.**优化的图形库,多媒体支持,GSM电话技术,蓝牙等

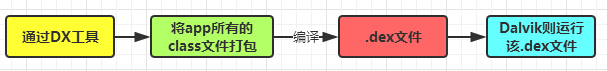
**5.**采用软件叠层方式构建

### 1.1.2架构的简单理解：

1.Application(应用程序层) 我们一般说的应用层的开发就是在这个层次上进行的，当然包括了系统内置的一组应用程序，使用的是Java语言

2.Application Framework(应用程序框架层) 无论系统内置或者我们自己编写的App，都需要使用到这层，比如我们想弄来电黑名单，自动挂断电话，我们就需要用到电话管理(TelephonyManager) 通过该层我们就可以很轻松的实现挂断操作，而不需要关心底层实现

3.Libraries(库) + Android Runtime(Android运行时) Android给我们提供了一组C/C++库，为平台的不同组件所使用，比如媒体框架；而Android Runtime则由Android核心库集 + Dalvik虚拟机构成，Dalvik虚拟机是针对移动设备的虚拟机，它的特点:不需要很快的CPU计算速度和大量的内存空间;而每个App都单独地运行在单独的Dalvik虚拟机内每个app对于一条Dalvik进程）而他的简单运行流程如：



4.Linux内核 这里就是涉及底层驱动的东西了，一些系统服务，比如安全性，内存管理以及进程管理等

## 1.2 开发环境搭建

### 1.2.1 环境搭建

略

### 1.2.2相关术语的解析

**1.**AVD：(android virtual machine):安卓虚拟设备,就是安卓的模拟器

**2.**ADT：(android development tools)安卓开发工具

**3.**SDK：(software development kit)软件开发工具包,就是安卓系统,平台架构等的工具集合

**4.**DDMS：(dalvik debug monitor service)安卓调试工具

**5.**adb：安卓调试桥,在sdk的platform-tools目录下,功能很多,命令行必备

**6.**DX工具：将.class转换成.dex文件

**7.**AAPT：(android asset packing tool),安卓资源打包工具

**8.**R.java文件：由aapt工具根据App中的资源文件自动生成,可以理解为资源字典

**9.**AndroidManifest.xml：app包名 + 组件声明 + 程序兼容的最低版本 + 所需权限等程序的配置文件

### 1.2.3 打包

略

## 1.3 Git版本管理

### 1.3.1 Git是什么

一个分布式版本控制系统，和SVN类似，但远比SVN强大的一个版本控制系统 ①Git可以方便的在本地进行版本管理，如同你本地有一个版本管理服务器一样 我们可以选择在合适的时间将本地版本推送到统一的版本管理服务器 ②Git每次会提取整个代码仓库的完整镜像，相当于对整个代码仓库都进行了一次备份，这样计时版本服务器除了问题，我们可以直接采用本地仓库恢复！结合本地版本管理功能，远程版本管理服务器出问题了，我们依然能继续写自己的代码，当他恢复的时候我们再提交我们的本地版本！ Git研发初期是为了更好的管理Linux内核，不过现在已经广泛应用于各种项目中！

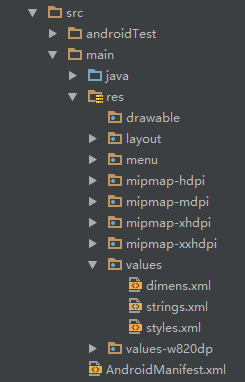
### 1.3.2 创建本地仓库与代码托管

略

## 1.4 界面原型设计

我们常说用户体验用户体验，用户使用你的软件，第一个会接触的是什么？没错，图形化界面(GUI)，简称UI，对于用户而言，最直观，给用户留下第一印像的是往往是程序的界面，而非功能！人，总喜欢美的东西，对吧？ 假如一样的功能，决定用户取向的，往往是UI！精美的UI！由此，一个产品的UI是非常重要的，而产品的界面原型设计一般是由公司的产品经理+美工来完成的，需求分析 ->界面原型设计 ->我们来写代码！可能你觉得界面原型对我们而言并不没什么作用，但假如你以后想自己开发App呢？又或者你升做产品经理呢？嘿嘿！世事无绝对，以后的事，谁知道呢？

## 1.5 工程相关解析



### 1.5.1 res

说到这个**res**目录，另外还有提下这个**assets**目录，虽然这里没有，但是我们可以自己创建，两者的区别在于是否前者下所有的资源文件都会在R.java文件下生成对应的资源id，而后者并不会；前者我们可以直接通过资源id访问到对应的资源；而后者则需要我们通过AssetManager以二进制流的形式来读取！对了，这个R文件可以理解为字典，res下每个资源都都会在这里生成一个唯一的id！存放我们各种资源文件的地方，有图片，字符串，动画，音频等，还有各种形式的XML文件

**1.图片**

**drawable**：存放各种位图文件，(.png，.jpg，.9png，.gif等)除此之外可能是一些其他的drawable类型的XML文件

**mipmap-hdpi**：高分辨率，一般我们把图片丢这里

**mipmap-mdpi**：中等分辨率，很少，除非兼容的的手机很旧

**mipmap-xhdpi**：超高分辨率，手机屏幕材质越来越好，以后估计会慢慢往这里过渡

**mipmap-xxhdpi**：超超高分辨率，这个在高端机上有所体现

**2.布局**

**layout**：该目录下存放的就是我们的布局文件，另外在一些特定的机型上，我们做屏幕适配，比如480\*320这样的手机，我们会另外创建一套布局，就行：layout-480x320这样的文件夹！

**3.菜单**

**menu**：在以前有物理菜单按钮，即menu键的手机上，用的较多，现在用的并不多，菜单项相关的资源xml可在这里编写，不知道谷歌会不会出新的东西来替代菜单了~

**4. values**

**demens.xml**：定义尺寸资源

**string.xml**：定义字符串资源

**styles.xml**：定义样式资源

**colors.xml**：定义颜色资源

**arrays.xml**：定义数组资源

**attrs.xml**：自定义控件时用的较多，自定义控件的属性！

theme主题文件，和styles很相似，但是会对整个应用中的Actvitiy或指定Activity起作用，一般是改变窗口外观的！可在Java代码中通过setTheme使用，或者在Androidmanifest.xml中为<application...>添加theme的属性！

PS:你可能看到过这样的values目录：values-w820dp，values-v11等，前者w代表平板设备，820dp代表屏幕宽度；而v11这样代表在API(11)，即android 3.0后才会用到的！

**5. raw**

用于存放各种原生资源(音频，视频，一些XML文件等)，我们可以通过openRawResource(int id)来获得资源的二进制流！其实和Assets差不多，不过这里面的资源会在R文件那里生成一个资源id而已

**6.动画**

属性动画和补间动画：

animator：存放属性动画的XML文件

**anim**：存放补间动画的XML文件

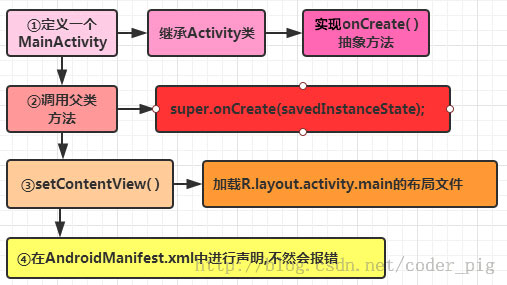
嗯，知道有什么资源，接下来就来了解该怎么用了： 前面也说了，我们所有的资源文件都会在R.java文件下生成一个资源id，我们可以通过这个资源id来完成资源的访问.

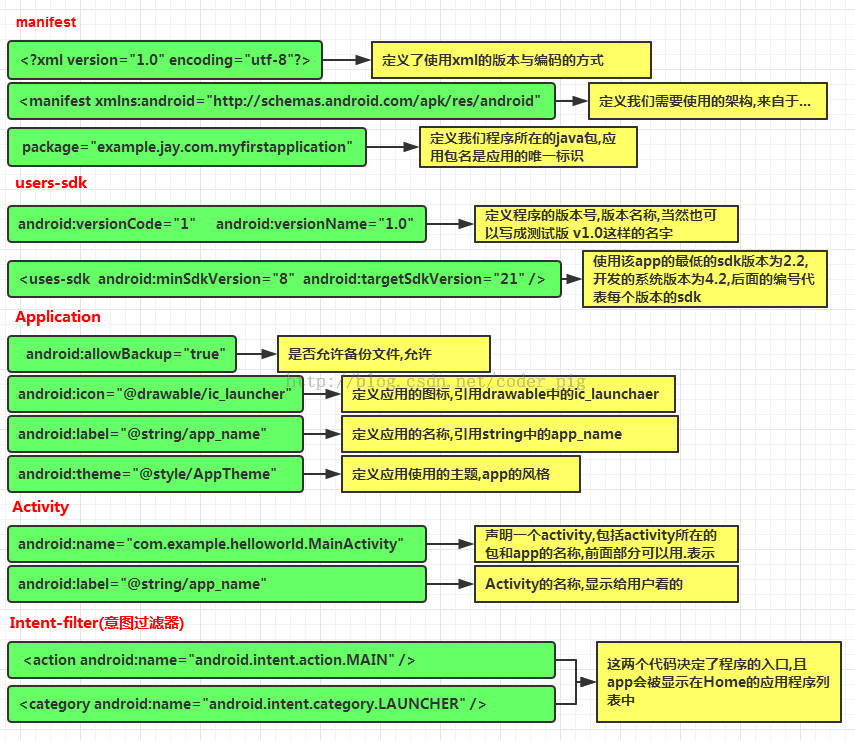
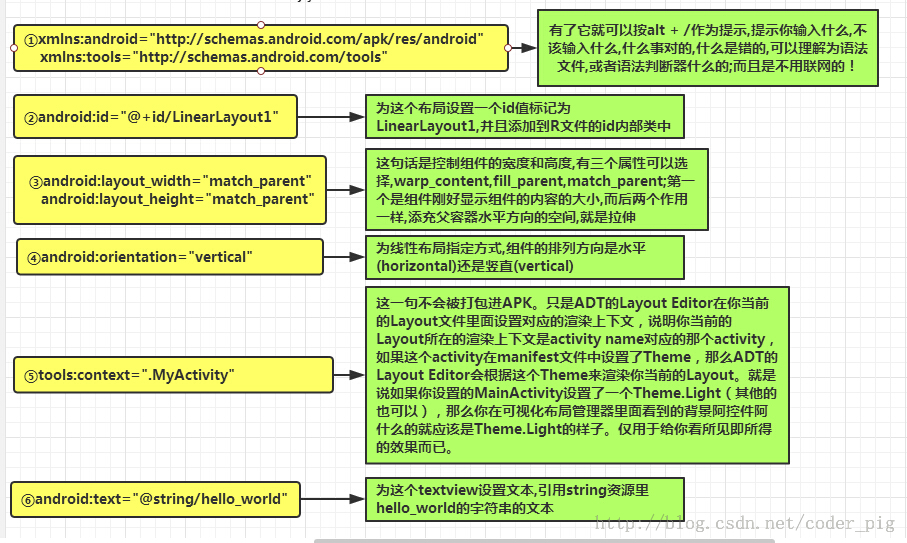
例如：R.string.app\_name

### 1.5.2 java

java：我们写Java代码的地方，业务功能都在这里实现

java文件与布局文件以及AndroidMainfest.xml的关系





## 1.6 Android程序签名打包

### 1.6.1 什么是签名，有什么用

1.应用程序升级：如果你希望用户无缝升级到新的版本，那么你必须用同一个证书进行签名。这是由于只有以同一个证书签名，系统才会允许安装升级的应用程序。如果你采用了不同的证书，那么系统会要求你的应用程序采用不同的包名称，在这种情况下相当于安装了一个全新的应用程序。如果想升级应用程序，签名证书要相同，包名称要相同！

2.应用程序模块化： Android系统可以允许同一个证书签名的多个应用程序在一个进程里运行，系统实际把他们作为一个单个的应用程序，此时就可以把我们的应用程序以模块的方式进行部署，而用户可以独立的升级其中的一个模块。

3.代码或者数据共享： Android提供了基于签名的权限机制，那么一个应用程序就可以为另一个以相同证书签名的应用程序公开自己的功能。以同一个证书对多个应用程序进行签名，利用基于签名的权限检查，你就可以在应用程序间以安全的方式共享代码和数据了。 不同的应用程序之间，想共享数据，或者共享代码，那么要让他们运行在同一个进程中，而且要让他们用相同的证书签名。 ————上述内容摘自:android 为什么需要签名

### 1.6.2 如何签名

略

## 1.7 反编译apk获得代码与资源

反编译Apk"，看上去好像好像很高端的样子，其实不然，就是通过某些反编译软件，对我们的APK进行反编译，从而获取程序的源代码，图片，XML资源等文件；不知道你有没有这样做过，看到一个别人的一个APP界面做得很精美，或者你看上别人的图片素材，简单点的，我们可以下载别人的APK，然后改下后缀名，改成xxx.zip，然后解压。但是，这种方法，获得的只会是一些.png，或者.jpg这样的位图文件资源，如果是xml类的资源，打开我们会发现是乱码，并且假如我们想看APK程序的Java代码，也是行不通的，因为他们都打被打包到classes.dex文件中！ 但是反编译可以解决你的需要~另外，切勿拿反编译来做违法的事，比如把人家的APK重新打包后使用自己的签名然后发布到相关市场...另外，我们是参考别人的代码，而不是完全拷贝！！！切记！！

这里推荐一个工具傻瓜化反编译：**Android killer**

# 2.常用UI控件与事件处理机制

如果你读到了这里说明你已经对安卓系统有一个整体的认识了，下面开始第二部分，也就是系统自带的UI。**注意：本章需要对编程有一定基础，如果基础不好，建议学习Java基础知识，有概念后再来看这一章。**

**本章目前只写到布局，后续会增加各个空间以及事件处理的机制，算是留下一个坑吧**

## 2.1 View与ViewGroup的概念

在Android APP中，所有的用户界面元素都是由View和ViewGroup的对象构成的。View是绘制在屏幕上的用户能与之交互的一个对象。而ViewGroup则是一个用于存放其他View（和ViewGroup）对象的布局容器！如图：



定义你的布局，你可以在代码中实例化View对象并且开始构建你的树，但最容易和最高效的方式来定义你的布局则是使用一个XML文件，用XML来构成布局更加符合人的阅读习惯，而XML类似与HTML 使用XML元素的名称代表一个View。所以< TextView >元素会在你的界面中创建一个TextView控件，而一个< LinearLayout >则会创建一个LinearLayout的容器！

**总结：**Android里的图形界面都是由View和ViewGroup以及他们的子类构成的： View：所有可视化控件的父类,提供组件描绘和时间处理方法 ViewGroup： View类的子类，可以拥有子控件,可以看作是容器 Android UI中的控件都是按照这种层次树的结构堆叠得，而创建UI布局的方式有两种， 自己在Java里写代码或者通过XML定义布局，后者显得更加方便和容易理解！ 也是我们最常用的手段！另外我们一般很少直接用View和ViewGroup来写布局，更多的 时候使用它们的子类控件或容器来构建布局！

## 2.2布局

### 2.2.1 ConstraintLayout 约束布局

**推荐重点掌握，但学习之前需要了解其他布局，学习这一章之前推荐先看其他布局**

参考链接：<https://www.jianshu.com/p/17ec9bd6ca8a>

#### 2.2.1.1 为什么要用ConstraintLayout

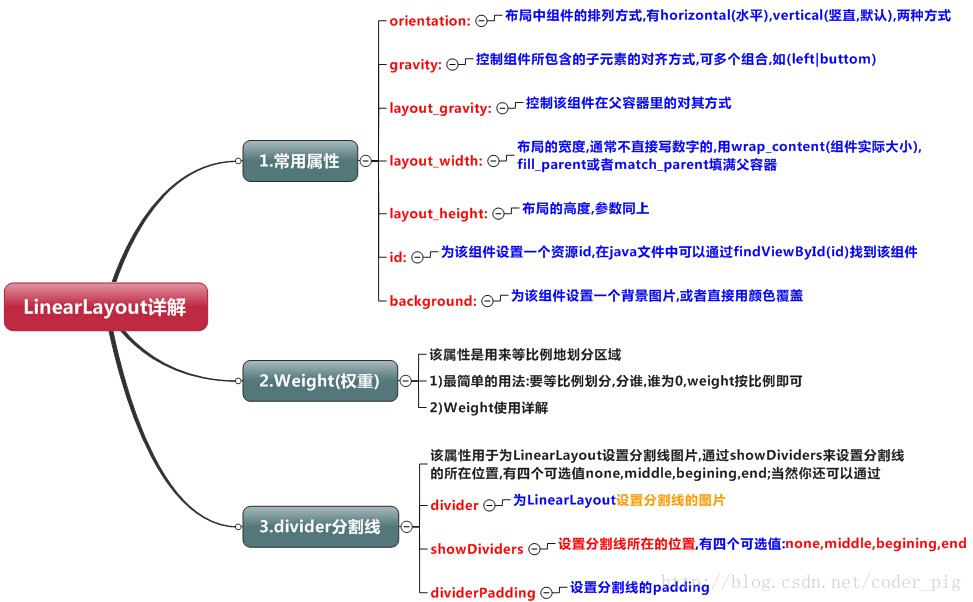
在开发过程中经常能遇到一些复杂的UI，可能会出现布局嵌套过多的问题，嵌套得越多，设备绘制视图所需的时间和计算功耗也就越多。有些人考虑到了嵌套布局带来的风险，所以用一个RelativeLayout来装下所有的控件。那么问题来了，既然用RelativeLayout可以解决问题，为什么还要使用ConstraintLayout呢？因为ConstraintLayout使用起来比RelativeLayout更灵活，性能更出色！还有一点就是ConstraintLayout可以按照比例约束控件位置和尺寸，能够更好地适配屏幕大小不同的机型。

#### 2.2.1.2 约束布局的使用方法

此处可直接在网页查看，未来可能会添加

### 2.2.2 LinearLayout 线性布局

**重要**



#### 2.2.2.1 权重

在java中设置权重：

**setLayoutParams(new LayoutParams(LayoutParams.FILL\_PARENT, LayoutParams.WRAP\_CONTENT, 1))**;

#### 2.2.2.2 分割线

**显示分割线：**

1. **直接加一个view设置成相应的颜色**
2. **设置divider属性**

**1)android:divider设置作为分割线的图片**

**2)android:showDividers设置分割线的位置,none(无),beginning(开始),end(结束),middle(每两个组件间)**

**3)dividerPadding设置分割线的Padding**

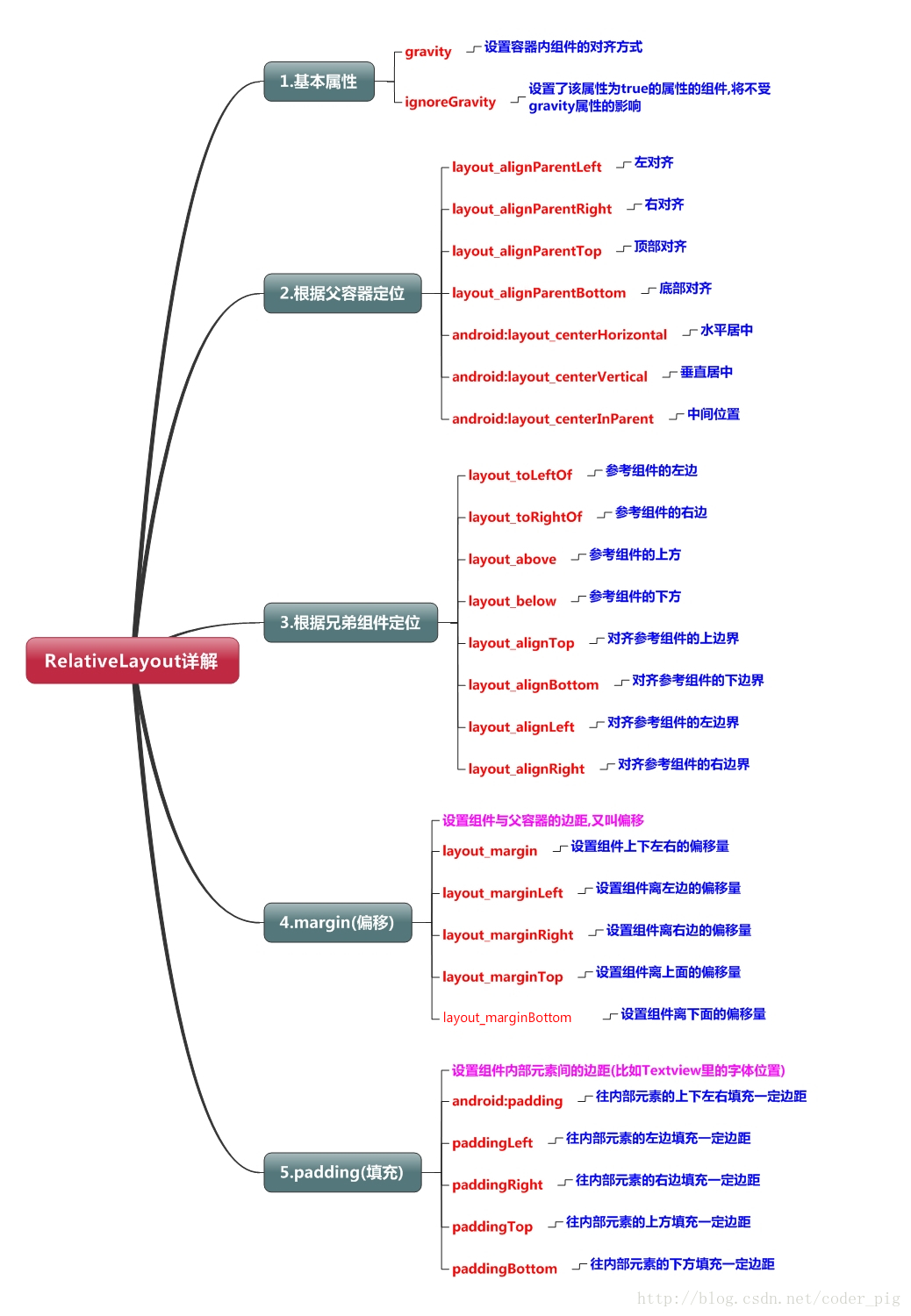
**注意：**使用Layout\_gravity的一个很重要的问题!!! 问题内容: 在一个LinearLayout的水平方向中布置两个TextView,想让一个左,一个右,怎么搞?

**当 android:orientation="vertical" 时， 只有水平方向的设置才起作用，垂直方向的设置不起作用。 即：left，right，center\_horizontal 是生效的。 当 android:orientation="horizontal" 时， 只有垂直方向的设置才起作用，水平方向的设置不起作用。 即：top，bottom，center\_vertical 是生效的**

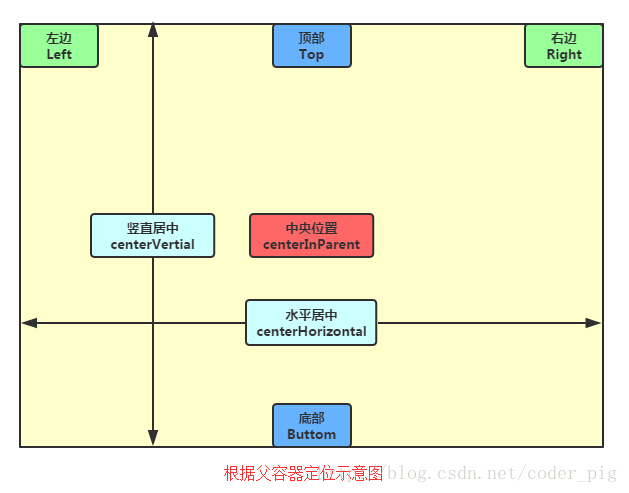
### 2.2.3 RelativeLayout 相对布局

**重要**

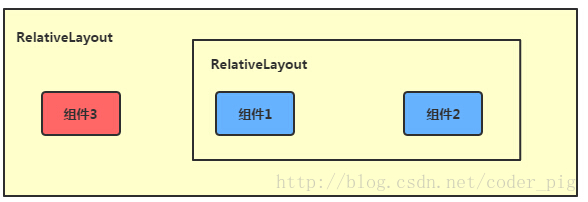
在上一节中我们对LinearLayout进行了详细的解析，LinearLayout也是我们 用的比较多的一个布局,我们更多的时候更钟情于他的weight(权重)属性，等比例划分，对屏幕适配还是 帮助蛮大的;但是使用LinearLayout的时候也有一个问题，就是当界面比较复杂的时候，需要嵌套多层的 LinearLayout,这样就会降低UI Render的效率(渲染速度),而且如果是listview或者GridView上的 item,效率会更低,另外太多层LinearLayout嵌套会占用更多的系统资源,还有可能引发stackoverflow; 但是如果我们使用RelativeLayout的话,可能仅仅需要一层就可以完成了。



#### 2.2.3.1 根据父容器定位



#### 2.2.3.2 根据兄弟组件定位

恩,先说下什么是兄弟组件吧,所谓的兄弟组件就是处于同一层次容器的组件, 图中的组件1,2就是兄弟组件了,而组件3与组件1或组件2并不是兄弟组件,所以组件3不能通过 组件1或2来进行定位,比如layout\_toleftof = "组件1"这样是会报错的！切记！

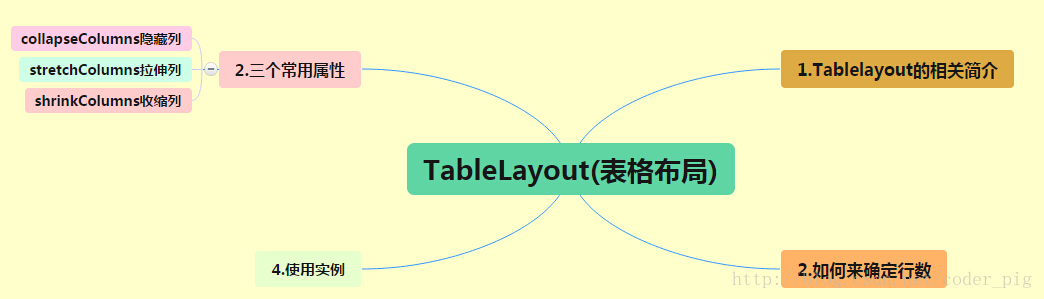
#### 2.2.3.2 margin与padding的区别

初学者对于这两个属性可能会有一点混淆，这里区分下： 首先margin代表的是偏移,比如marginleft = "5dp"表示组件离容器左边缘偏移5dp; 而padding代表的则是填充,而填充的对象针对的是组件中的元素,比如TextView中的文字 比如为TextView设置paddingleft = "5dp",则是在组件里的元素的左边填充5dp的空间！ margin针对的是容器中的组件，而padding针对的是组件中的元素，要区分开来！

**平时我们设置margin的时候都习惯了是正数的, 其实是可以用负数的**

### 2.2.4 TableLayout 表格布局

**非必须掌握，换而言之不重要**



#### 2.2.4.1 如何确定行数和列数

在h5中我们可以通过< table >< tr >< td >就可以生成一个HTML的表格, 而Android中也允许我们使用表格的方式来排列组件,就是行与列的方式,就说我们这节的TableLayout! 但却不像我们后面会讲到的Android 4.0后引入的GridLayout(网格)布局一样,直接就可以设置多少行与多少列！

①如果我们直接往TableLayout中添加组件的话,那么这个组件将占满一行！！！

②如果我们想一行上有多个组件的话,就要添加一个TableRow的容器,把组件都丢到里面！

③tablerow中的组件个数就决定了该行有多少列,而列的宽度由该列中最宽的单元格决定

④tablerow的layout\_width属性,默认是fill\_parent的,我们自己设置成其他的值也不会生效！！！ 但是layout\_height默认是wrapten——content的,我们却可以自己设置大小！

⑤整个表格布局的宽度取决于父容器的宽度(占满父容器本身)

⑥有多少行就要自己数啦,一个tablerow一行,一个单独的组件也一行！多少列则是看tableRow中 的组件个数,组件最多的就是TableLayout的列数

#### 2.2.4.2 常用属性

android:collapseColumns:设置需要被隐藏的列的序号

android:shrinkColumns:设置允许被收缩的列的列序号

android:stretchColumns:设置运行被拉伸的列的列序号

以上这三个属性的列号都是从0开始算的,比如shrinkColunmns = "2",对应的是第三列！

可以设置多个,用逗号隔开比如"0,2",如果是所有列都生效,则用"\*"号即可

除了这三个常用属性,还有两个属性,分别就是跳格子以及合并单元格,这和HTML中的Table类似:

android:layout\_column="2":表示的就是跳过第二个,直接显示到第三个格子处,从1开始算的!

android:layout\_span="4":表示合并4个单元格,也就说这个组件占4个单元格

### 2.2.5 FrameLayout 帧布局

**非必须掌握，换而言之不重要**

FrameLayout(帧布局)可以说布局中最为简单的一个布局,这个布局直接在屏幕上开辟出一块空白的区域,当我们往里面添加控件的时候,会默认把他们放到这块区域的左上角,而这种布局方式却没有任何的定位方式,所以它应用的场景并不多;帧布局的大小由控件中最大的子控件决定,如果控件的大小一样大的话,那么同一时刻就只能看到最上面的那个组件!后续添加的控件会覆盖前一个!虽然默认会将控件放置在左上角,但是我们也可以通过layout\_gravity属性,指定到其他的位置!

#### 2.2.5.1 常用属性

FrameLayout的属性很少就两个,但是在说之前我们先介绍一个东西:

前景图像:永远处于帧布局最上面,直接面对用户的图像,就是不会被覆盖的图片。

**android:foreground**:\*设置改帧布局容器的前景图像

**android:foregroundGravity**:设置前景图像显示的位置

### 2.2.6 GridLayout 网格布局

**非必须掌握，换而言之不重要**

和前面所学的TableLayout(表格布局) 有点类似,不过他有很多前者没有的东西,也更加好用,

**1·**可以自己设置布局中组件的排列方式

**2·**可以自定义网格布局有多少行,多少列

**3·**可以直接设置组件位于某行某列

**4·**可以设置组件横跨几行或者几列



### 2.2.7 AbsoluteLayout 绝对布局

之所以把这个放到最后,是因为绝对布局,我们基本上都是不会使用的,当然你也可以直接跳过,不过作为一个喜欢增长姿势的程序员,我们还是可以了解这个AbsoluteLayout布局的, Android 中我们用这个少的原因,就是因为我们开发的应用需要在很多的机型上面进行一个适配,如果你 使用了这个绝对布局的话,可能你在4寸的手机上是显示正常的,而换成5寸的手机,就可能出现偏移 和变形,所以的话,这个还是不建议使用了,当然,如果你不会写代码的话,又想玩玩android,那么写 布局的时候就可以通过ADT把需要的组件,拖拉到界面上!这个AbsoluteLayout是直接通过X,Y坐标来 控制组件在Activity中的位置的！另外这个但单位是dp!

至此安卓的所有布局学习完毕，这几个布局推荐掌握[2.2.1 ConstraintLayout 约束布局](#_2.2.1_ConstraintLayout_约束布局)以及[2.2.2 LinearLayout 线性布局](#_2.2.2_LinearLayout_线性布局)和[2.2.3 RelativeLayout 相对布局](#_2.2.3_RelativeLayout_相对布局)，其他不推荐掌握了解就行

## 2.3 常用基本控件

## 2.4 日期类控件

## 2.5 adapter控件

## 2.6 提示类控件

## 2.7 菜单类控件

## 2.8 其他控件

## 2.9 事件处理的机制

# 3.Android四大组件和Intent组件

## 3.1 Actviity 活动

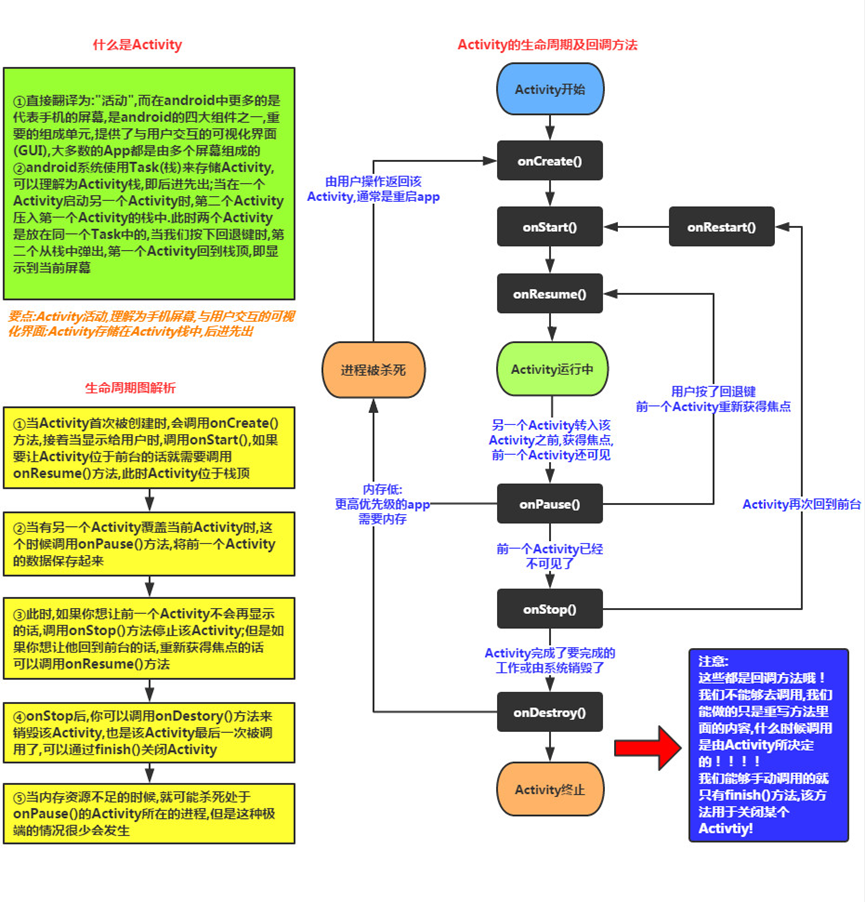
### 3.1.1 Activity的概念与Activity的生命周期图

**极度重要，必问**

Activity是一个应用程序的组件，他在屏幕上提供了一个区域，允许用户在上面做一些交互性的操作， 比如打电话，照相，发送邮件，或者显示一个地图！Activity可以理解成一个绘制用户界面的窗口， 而这个窗口可以填满整个屏幕，也可能比屏幕小或者浮动在其他窗口的上方！

**1.Activity用于显示用户界面，用户通过Activity交互完成相关操作**

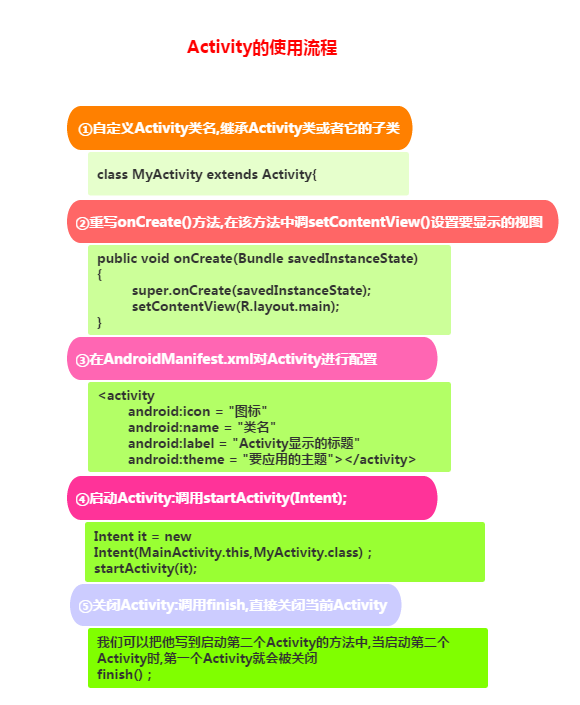
**2.一个App允许有多个Activity**

****

**1. onPause()和onStop()被调用的前提是： 打开了一个新的Activity！而前者是旧Activity还可见的状态；后者是旧Activity已经不可见！**

**2. 另外，亲测：AlertDialog和PopWindow是不会触发上述两个回调方法的~**

### 3.1.2 Activity的创建流程



切记，Android中的四大组件，只要你定义了，无论你用没用，都要在AndroidManifest.xml对 这个组件进行声明，不然运行时程序会直接退出，报ClassNotFindException...

### 3.1.3 onCreate() 参数详解

在重写Act的onCreate()方法时会发现，这玩意有两个参数：

要用两个参数的方法，先要在配置文件中为我们的Activity设置一个属性：

**android:persistableMode="persistAcrossReboots"**

然后我们的Activity就拥有了持久化的能力了，一般我们会搭配另外两个方法来使用：

public void **onSaveInstanceState**(Bundle outState, PersistableBundle outPersistentState)

public void **onRestoreInstanceState**(Bundle savedInstanceState, PersistableBundle persistentState)

前一个方法会在下述情形中被调用：

1·点击home键回到主页或长按后选择运行其他程序

2·按下电源键关闭屏幕

3·启动新的Activity

4·横竖屏切换时，肯定会执行，因为横竖屏切换的时候会先销毁Act，然后再重新创建

重要原则：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（你可以保存也可以不保存）。

而后一个方法，和onCreate同样可以从取出前者保存的数据： 一般是在onStart()和onResume()之间执行！ 之所以有两个可以获取到保存数据的方法，是为了避免Act跳转而没有关闭， 然后不走onCreate()方法，而你又想取出保存数据~

### 3.1.4 启动一个Activity的几种方式

#### 3.1.4.1 显示启动

①最常见的：

**startActivity**(new Intent(当前Act.this,要启动的Act.class));

②通过Intent的ComponentName：

ComponentName cn = new ComponentName("当前Act的全限定类名","启动Act的全限定类名") ;

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

intent.setComponent(cn) ;

**startActivity**(intent) ;

③初始化Intent时指定包名：

**Intent** intent = new Intent("android.intent.action.MAIN");

intent.setClassName("当前Act的全限定类名","启动Act的全限定类名");

**startActivity**(intent);

#### 3.1.4.2 隐式启动

通过Intent-filter的Action,Category或data来实现 这个是通过Intent的 intent-filter\*\*来实现的，这个Intent那章会详细讲解！ 这里知道个大概就可以了！

#### 3.1.4.3 绝对路径启动

**Intent** intent = getPackageManager().getLaunchIntentForPackage("apk第一个启动的Activity的全限定类名") ;

if(intent != null) **startActivity**(intent) ;

### 3.1.5 横竖屏切换与状态保存的问题

**重要，必问**

**App横竖屏切换的时候会销毁当前的Activity然后重新创建一个，你可以自行在生命周期 的每个方法里都添加打印Log的语句，来进行判断，又或者设一个按钮一个TextView点击按钮后，修改TextView 文本，然后横竖屏切换，会神奇的发现TextView文本变回之前的内容了！ 横竖屏切换时Act走下述生命周期：**

**onPause-> onStop-> onDestory-> onCreate->onStart->onResume**

**禁止屏幕横竖屏自动切换**：

很简单，在AndroidManifest.xml中为Act添加一个属性： android:screenOrientation， 有下述可选值：

**unspecified**:默认值 由系统来判断显示方向.判定的策略是和设备相关的，所以不同的设备会有不同的显示方向。

**landscape**:横屏显示（宽比高要长）

**portrait**:竖屏显示(高比宽要长)

**user**:用户当前首选的方向

**behind**:和该Activity下面的那个Activity的方向一致(在Activity堆栈中的)

**sensor**:有物理的感应器来决定。如果用户旋转设备这屏幕会横竖屏切换。

**nosensor**:忽略物理感应器，这样就不会随着用户旋转设备而更改了（"unspecified"设置除外）。

**横竖屏时想加载不同的布局**：

1）准备两套不同的布局，Android会自己根据横竖屏加载不同布局： 创建两个布局文件夹：layout-land横屏,layout-port竖屏 然后把这两套布局文件丢这两文件夹里，文件名一样，Android就会自行判断，然后加载相应布局了！

2 )自己在代码中进行判断，自己想加载什么就加载什么：

我们一般是在onCreate()方法中加载布局文件的，我们可以在这里对横竖屏的状态做下判断，关键代码如下：

if (this.getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE){

setContentView(R.layout.横屏);

}

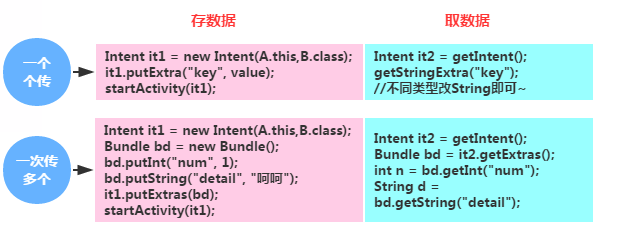
else if (this.getResources().getConfiguration().orientation ==Configuration.ORIENTATION\_PORTRAIT) {

setContentView(R.layout.竖屏);

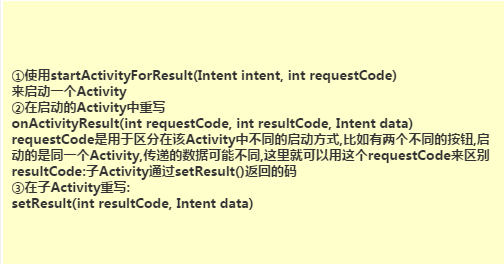
}

### 3.1.6 Activity间的数据传递

前一个act传给后一个act



后一个act传给前一个act



### 3.1.7 完全退出App的方法

/\*\*

\* 退出应用程序

\*/

public void AppExit(Context context) {

try {

ActivityCollector.finishAll();

ActivityManager activityMgr = (ActivityManager) context

.getSystemService(Context.ACTIVITY\_SERVICE);

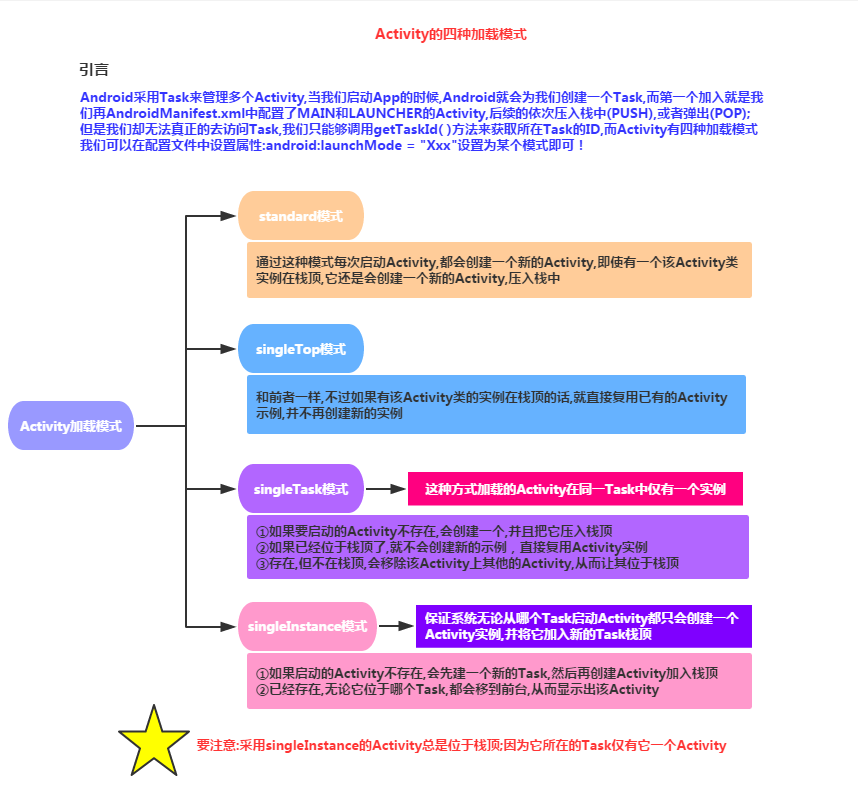
activityMgr.killBackgroundProcesses(context.getPackageName());

System.exit(0);

} catch (Exception ignored) {}

}

### 3.1.8 Activity的四种加载模式详解



#### 3.1.8.1 standard模式

标准启动模式，也是activity的默认启动模式。在这种模式下启动的activity可以被多次实例化，即在同一个任务中可以存在多个activity的实例，每个实例都会处理一个Intent对象。如果Activity A的启动模式为standard，并且A已经启动，在A中再次启动Activity A，即调用startActivity（new Intent（this，A.class）），会在A的上面再次启动一个A的实例，即当前的桟中的状态为A-->A。

#### 3.1.8.2 singleTop模式

如果一个以singleTop模式启动的Activity的实例已经存在于任务栈的栈顶， 那么再启动这个Activity时，不会创建新的实例，而是重用位于栈顶的那个实例， 并且会调用该实例的onNewIntent()方法将Intent对象传递到这个实例中。 举例来说，如果A的启动模式为singleTop，并且A的一个实例已经存在于栈顶中， 那么再调用startActivity（new Intent（this，A.class））启动A时， 不会再次创建A的实例，而是重用原来的实例，并且调用原来实例的onNewIntent()方法。 这时任务栈中还是这有一个A的实例。如果以singleTop模式启动的activity的一个实例 已经存在与任务栈中，但是不在栈顶，那么它的行为和standard模式相同，也会创建多个实例。

#### 3.1.8.3 singleTask模式

只允许在系统中有一个Activity实例。如果系统中已经有了一个实例， 持有这个实例的任务将移动到顶部，同时intent将被通过onNewIntent()发送。 如果没有，则会创建一个新的Activity并置放在合适的任务中。

#### 3.1.8.4 singleInstance模式

保证系统无论从哪个Task启动Activity都只会创建一个Activity实例,并将它加入新的Task栈顶 也就是说被该实例启动的其他activity会自动运行于另一个Task中。 当再次启动该activity的实例时，会重用已存在的任务和实例。并且会调用这个实例 的onNewIntent()方法，将Intent实例传递到该实例中。和singleTask相同， 同一时刻在系统中只会存在一个这样的Activity实例。

### 3.1.9 activity线程之间的关系

这部分未来补充，保持更新

## 3.2 Service 服务

## 3.3 BroadcastReceiver 广播

## 3.4 ContentProvider 内容提供者

## 3.5 Intent 意图

# 4.Fragment

# 5.数据存储

## 5.1 文件存储读写

## 5.2 SharedPreference

## 5.3 SQLite

# 6.网络编程

## 6.1 HTTP详解

## 6.2 XML数据解析

## 6.3 Json数据解析

## 6.4 Android文件上传下载

## 6.5 调用WebService

## 6.6 WebView

## 6.7 Socket

# 7.绘图与动画

## 7.1 Drawable

## 7.2 Bitmap

## 7.3 绘图相关API

## 7.4 动画

# 8.多媒体开发

## 8.1 SoundPool播放音效

## 8.2 MediaPlayer播放音视

## 8.3 Careme拍照

## 8.4 MediaRecord录音

# 9.系统API，地图定位

## 9.1 TelephonManager 电话管理器

## 9.2 SmsManager短信管理器

## 9.3 AudioManager 音频管理器

## 9.4 Vibrator震动器

## 9.5 AlarmManager 闹钟服务

## 9.6 PowerManager 电源服务

## 9.7 WindowManager 窗口管理服务

## 9.8 LayoutIflater布局管理器

## 9.9 WallpaperManager壁纸管理器

## 9.10 传感器

## 9.11 GPS

# 10.项目实战