[1.环境搭建，开发相关 5](#_Toc28248233)

[1.1背景相关与系统架构分析 5](#_Toc28248234)

[1.1.1系统特性： 5](#_Toc28248235)

[1.1.2架构的简单理解： 5](#_Toc28248236)

[1.2 开发环境搭建 6](#_Toc28248237)

[1.2.1 环境搭建 6](#_Toc28248238)

[1.2.2相关术语的解析 6](#_Toc28248239)

[1.2.3 打包 6](#_Toc28248240)

[1.3 Git版本管理 6](#_Toc28248241)

[1.3.1 Git是什么 6](#_Toc28248242)

[1.3.2 创建本地仓库与代码托管 6](#_Toc28248243)

[1.4 界面原型设计 7](#_Toc28248244)

[1.5 工程相关解析 7](#_Toc28248245)

[1.5.1 res 7](#_Toc28248246)

[1.5.2 java 8](#_Toc28248247)

[1.6 Android程序签名打包 10](#_Toc28248248)

[1.6.1 什么是签名，有什么用 10](#_Toc28248249)

[1.6.2 如何签名 11](#_Toc28248250)

[1.7 反编译apk获得代码与资源 11](#_Toc28248251)

[2.常用UI控件与事件处理机制 11](#_Toc28248252)

[2.1 View与ViewGroup的概念 11](#_Toc28248253)

[2.2布局 12](#_Toc28248254)

[2.2.1 ConstraintLayout 约束布局 12](#_Toc28248255)

[2.2.1.1 为什么要用ConstraintLayout 12](#_Toc28248256)

[2.2.1.2 约束布局的使用方法 13](#_Toc28248257)

[2.2.2 LinearLayout 线性布局 13](#_Toc28248258)

[2.2.2.1 权重 13](#_Toc28248259)

[2.2.2.2 分割线 13](#_Toc28248260)

[2.2.3 RelativeLayout 相对布局 14](#_Toc28248261)

[2.2.3.1 根据父容器定位 16](#_Toc28248262)

[2.2.3.2 根据兄弟组件定位 16](#_Toc28248263)

[2.2.3.2 margin与padding的区别 17](#_Toc28248264)

[2.2.4 TableLayout 表格布局 17](#_Toc28248265)

[2.2.4.1 如何确定行数和列数 17](#_Toc28248266)

[2.2.4.2 常用属性 18](#_Toc28248267)

[2.2.5 FrameLayout 帧布局 18](#_Toc28248268)

[2.2.5.1 常用属性 18](#_Toc28248269)

[2.2.6 GridLayout 网格布局 18](#_Toc28248270)

[2.2.7 AbsoluteLayout 绝对布局 19](#_Toc28248271)

[2.3 常用基本控件 20](#_Toc28248272)

[2.4 日期类控件 20](#_Toc28248273)

[2.5 adapter控件 20](#_Toc28248274)

[2.6 提示类控件 20](#_Toc28248275)

[2.7 菜单类控件 20](#_Toc28248276)

[2.8 其他控件 20](#_Toc28248277)

[2.9 事件处理的机制 20](#_Toc28248278)

[3.Android四大组件和Intent组件 20](#_Toc28248279)

[3.1 Actviity 活动 20](#_Toc28248280)

[3.1.1 Activity的概念与Activity的生命周期图 20](#_Toc28248281)

[3.1.2 Activity的创建流程 22](#_Toc28248282)

[3.1.3 onCreate() 参数详解 22](#_Toc28248283)

[3.1.4 启动一个Activity的几种方式 23](#_Toc28248284)

[3.1.4.1 显示启动 23](#_Toc28248285)

[3.1.4.2 隐式启动 23](#_Toc28248286)

[3.1.4.3 绝对路径启动 24](#_Toc28248287)

[3.1.5 横竖屏切换与状态保存的问题 24](#_Toc28248288)

[3.1.6 Activity间的数据传递 25](#_Toc28248289)

[3.1.7 完全退出App的方法 25](#_Toc28248290)

[3.1.8 Activity的四种加载模式详解 26](#_Toc28248291)

[3.1.8.1 standard模式 26](#_Toc28248292)

[3.1.8.2 singleTop模式 26](#_Toc28248293)

[3.1.8.3 singleTask模式 27](#_Toc28248294)

[3.1.8.4 singleInstance模式 27](#_Toc28248295)

[3.1.9 activity线程之间的关系 27](#_Toc28248296)

[3.2 Service 服务 27](#_Toc28248297)

[3.2.1 线程 27](#_Toc28248298)

[3.2.2 线程的生命周期 28](#_Toc28248299)

[3.2.3 创建线程的三种方式 28](#_Toc28248300)

[3.2.4 service的生命周期 29](#_Toc28248301)

[3.2.5 service的使用 29](#_Toc28248302)

[3.2.5.1 生命周期详解 29](#_Toc28248303)

[3.2.5.2 StartService启动Service 30](#_Toc28248304)

[3.2.5.3 BindService启动Service 30](#_Toc28248305)

[3.2.5.4 StartService启动Service后bindService绑定 31](#_Toc28248306)

[3.2.6 生命周期的实现 31](#_Toc28248307)

[3.2.7 结论 31](#_Toc28248308)

[3.2.8 IntentService的使用 31](#_Toc28248309)

[3.2.9 Activity与Service通信 32](#_Toc28248310)

[3.2.10 前台服务即状态栏弹框 32](#_Toc28248311)

[3.2.11 后台定时service 33](#_Toc28248312)

[3.2.12 AIDL跨进程通信 33](#_Toc28248313)

[3.3 BroadcastReceiver 广播 33](#_Toc28248314)

[3.3.1 BroadcastReceiver是什么 33](#_Toc28248315)

[3.3.2 两种广播类型 34](#_Toc28248316)

[3.3.3 接收系统广播 34](#_Toc28248317)

[3.3.4 发送广播 35](#_Toc28248318)

[3.3.5 本地广播 35](#_Toc28248319)

[3.3.6 常用的系统广播总结 36](#_Toc28248320)

[3.4 ContentProvider 内容提供者 39](#_Toc28248321)

[3.4.1 什么是ContentProvider 39](#_Toc28248322)

[3.4.2 使用系统提供的ContentProvider 40](#_Toc28248323)

[3.4.3 自定义ContentProvider 41](#_Toc28248324)

[3.4.4 通过ContentObserver监听ContentProvider的数据变化 42](#_Toc28248325)

[3.4.5 其他ContentProvider 42](#_Toc28248326)

[3.5 Intent 意图 43](#_Toc28248327)

[3.5.1 四大组件间的枢纽 43](#_Toc28248328)

[3.5.2 Intent的七个属性 43](#_Toc28248329)

[3.5.2.1 ComponentName(组件名称) 43](#_Toc28248330)

[3.5.1.2 Action(动作) 43](#_Toc28248331)

[3.5.1.3 Category(类别) 44](#_Toc28248332)

[3.5.1.4 Data(数据)，Type(MIME类型) 44](#_Toc28248333)

[3.5.1.5 Extras(额外) 44](#_Toc28248334)

[3.5.1.6 Flags(标记) 45](#_Toc28248335)

[3.5.3 隐式Intent详解 45](#_Toc28248336)

[3.5.4 常用系统Intent合集 45](#_Toc28248337)

[3.5.5 Intent数据的传递 50](#_Toc28248338)

[3.5.5.1 Intent传递简单数据 50](#_Toc28248339)

[3.5.5.2 Intent传递数组 50](#_Toc28248340)

[3.5.5.3 Intent传递集合 50](#_Toc28248341)

[3.5.5.4 Intent传递对象 51](#_Toc28248342)

[3.5.5.5 Intent传递Bitmap 52](#_Toc28248343)

[4. Fragment 53](#_Toc28248344)

[4.1 什么是Fragment 53](#_Toc28248345)

[4.2 Fragment的生命周期 54](#_Toc28248346)

[4.3 Fragment加载方式 55](#_Toc28248347)

[4.3.1 静态加载Fragment 55](#_Toc28248348)

[4.3.2 动态加载Fragment 56](#_Toc28248349)

[4.4 Fragment与Activity的交互 56](#_Toc28248350)

[4.5 简单的底部导航栏实现 56](#_Toc28248351)

[4.6 Fragment与ViewPager的联合实现 56](#_Toc28248352)

[5.数据存储 57](#_Toc28248353)

[5.1 文件存储读写 57](#_Toc28248354)

[5.2 SharedPreference 57](#_Toc28248355)

[5.3 SQLite 57](#_Toc28248356)

[6.网络编程 57](#_Toc28248357)

[6.1 HTTP详解 57](#_Toc28248358)

[6.2 XML数据解析 57](#_Toc28248359)

[6.3 Json数据解析 57](#_Toc28248360)

[6.4 Android文件上传下载 57](#_Toc28248361)

[6.5 调用WebService 57](#_Toc28248362)

[6.6 WebView 57](#_Toc28248363)

[6.7 Socket 57](#_Toc28248364)

[7.绘图与动画 57](#_Toc28248365)

[7.1 Drawable 57](#_Toc28248366)

[7.2 Bitmap 58](#_Toc28248367)

[7.3 绘图相关API 58](#_Toc28248368)

[7.4 动画 58](#_Toc28248369)

[8.多媒体开发 58](#_Toc28248370)

[8.1 SoundPool播放音效 58](#_Toc28248371)

[8.2 MediaPlayer播放音视 58](#_Toc28248372)

[8.3 Careme拍照 58](#_Toc28248373)

[8.4 MediaRecord录音 58](#_Toc28248374)

[9.系统API，地图定位 58](#_Toc28248375)

[9.1 TelephonManager 电话管理器 58](#_Toc28248376)

[9.2 SmsManager短信管理器 58](#_Toc28248377)

[9.3 AudioManager 音频管理器 58](#_Toc28248378)

[9.4 Vibrator震动器 58](#_Toc28248379)

[9.5 AlarmManager 闹钟服务 58](#_Toc28248380)

[9.6 PowerManager 电源服务 58](#_Toc28248381)

[9.7 WindowManager 窗口管理服务 59](#_Toc28248382)

[9.8 LayoutIflater布局管理器 59](#_Toc28248383)

[9.9 WallpaperManager壁纸管理器 59](#_Toc28248384)

[9.10 传感器 59](#_Toc28248385)

[9.11 GPS 59](#_Toc28248386)

[10.项目实战 59](#_Toc28248387)

# 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

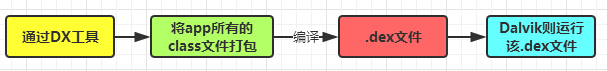
**5.**采用软件叠层方式构建

### 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，参考链接：<https://blog.csdn.net/xuw_xy/article/details/89524335>

### 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，参考链接：<https://www.cnblogs.com/lsdb/p/9337342.html>

## 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！

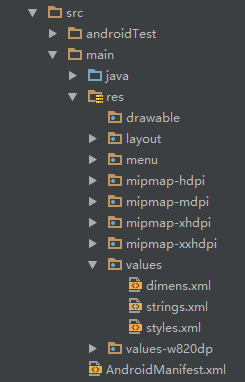
### 1.3.2 创建本地仓库与代码托管

略，未来会单独来补充

## 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

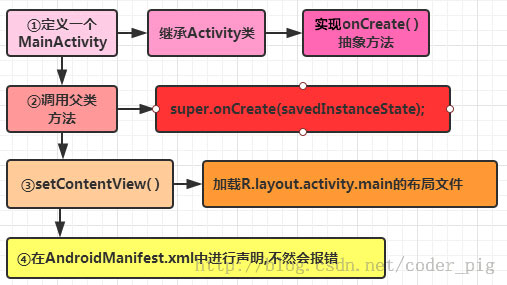
嗯，知道有什么资源，接下来就来了解该怎么用了： 前面也说了，我们所有的资源文件都会在R.java文件下生成一个资源id，我们可以通过这个资源id来完成资源的访问.

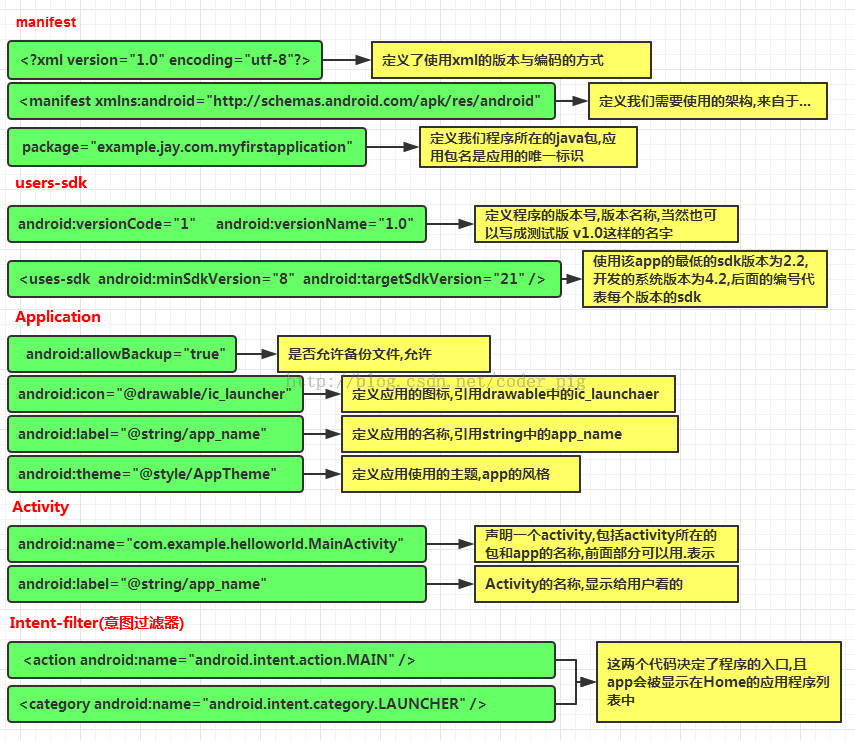
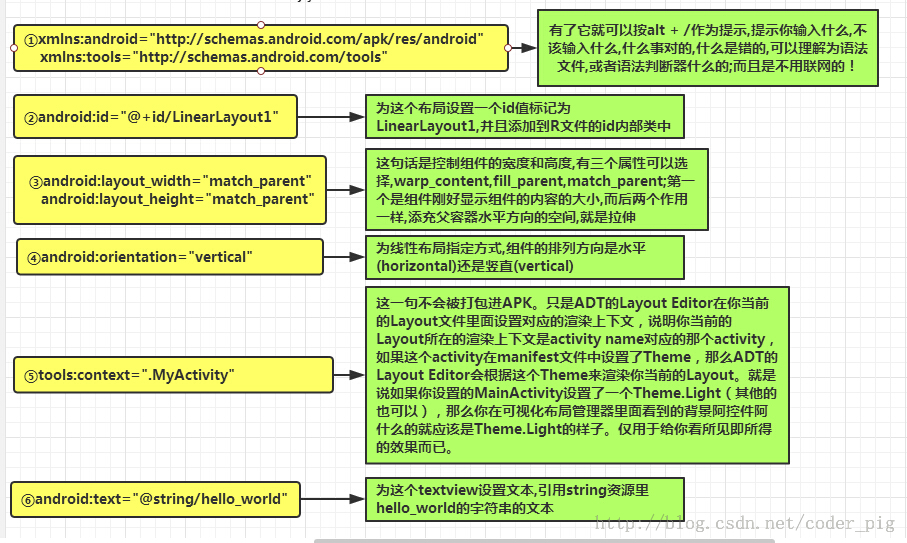
例如：R.string.app\_name

### 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**

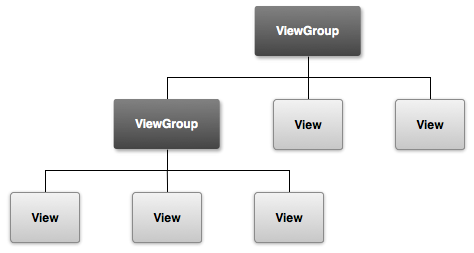
# 2.常用UI控件与事件处理机制

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

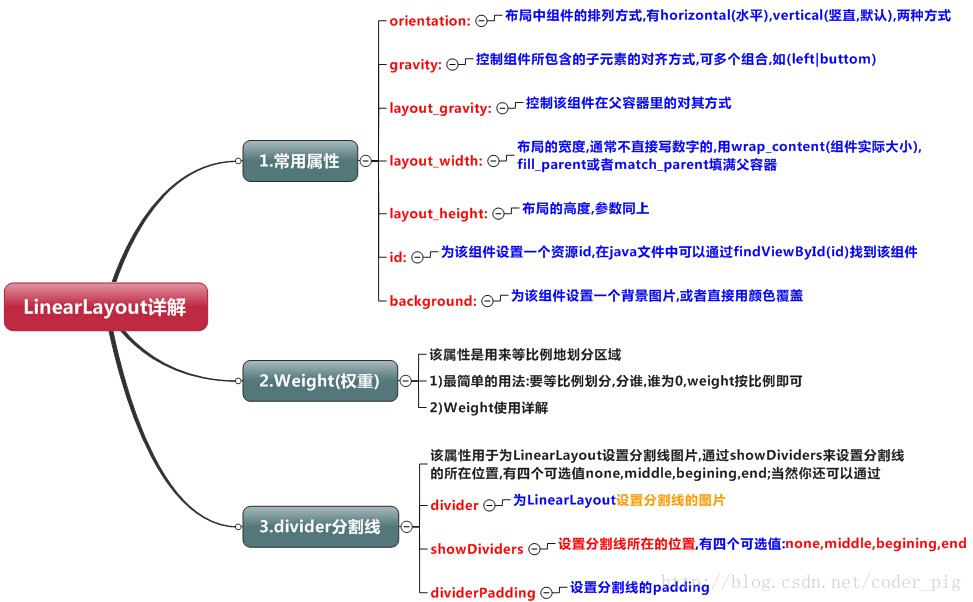
在开发过程中经常能遇到一些复杂的UI，可能会出现布局嵌套过多的问题，嵌套得越多，设备绘制视图所需的时间和计算功耗也就越多。有些人考虑到了嵌套布局带来的风险，所以用一个RelativeLayout来装下所有的控件。那么问题来了，既然用RelativeLayout可以解决问题，为什么还要使用ConstraintLayout呢？因为ConstraintLayout使用起来比RelativeLayout更灵活，性能更出色！还有一点就是ConstraintLayout可以按照比例约束控件位置和尺寸，能够更好地适配屏幕大小不同的机型。

#### 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**

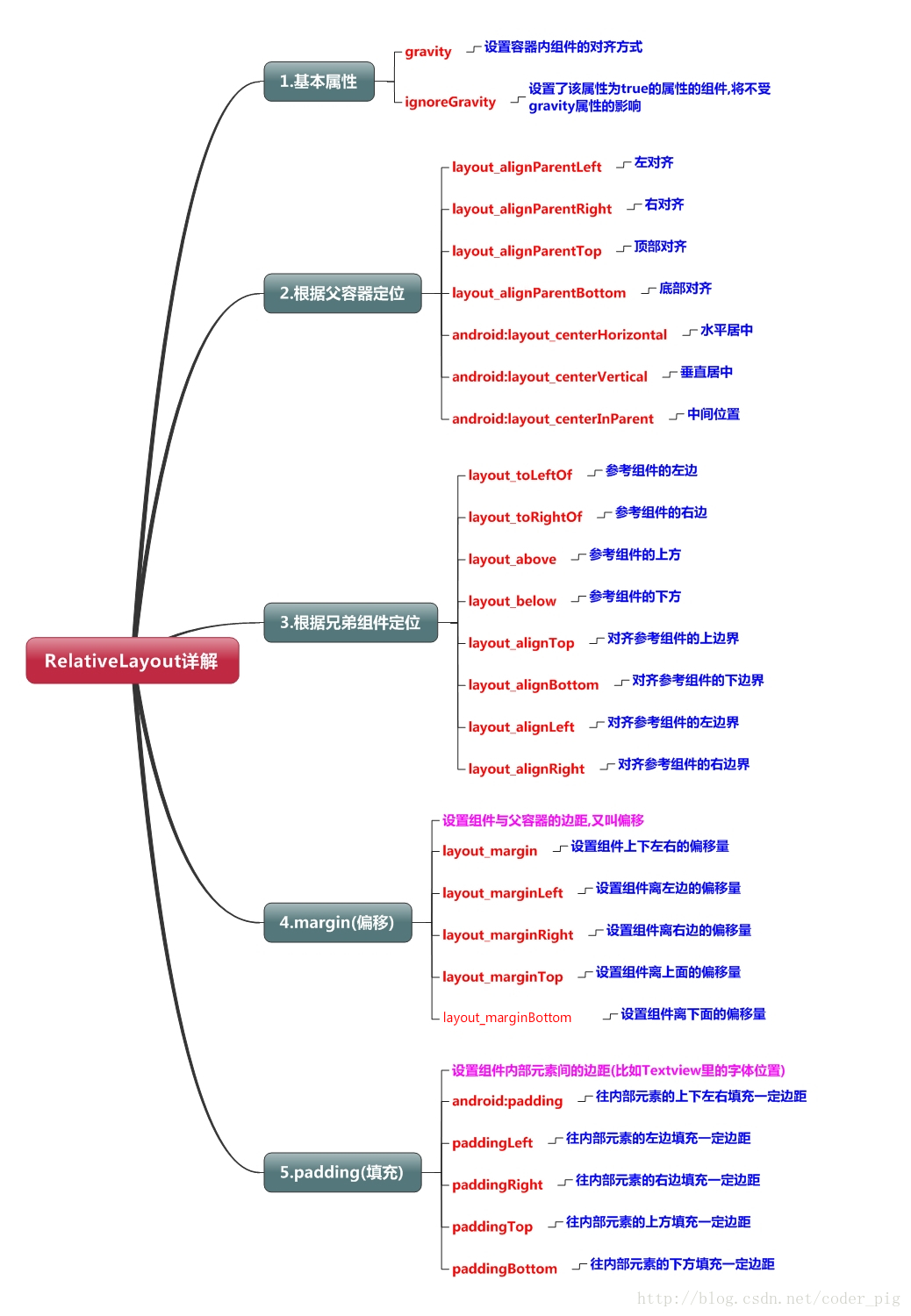
**注意：**使用Layout\_gravity的一个很重要的问题!!! 问题内容: 在一个LinearLayout的水平方向中布置两个TextView,想让一个左,一个右,怎么搞?

**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**

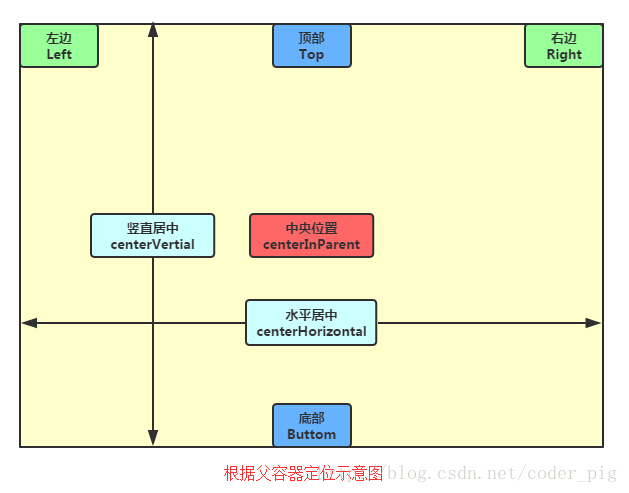
### 2.2.3 RelativeLayout 相对布局

**重要**

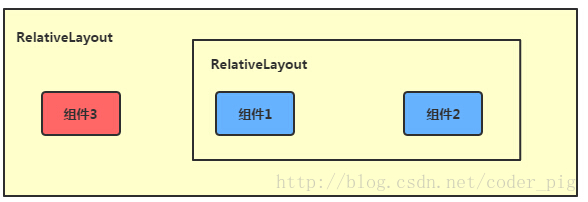
在上一节中我们对LinearLayout进行了详细的解析，LinearLayout也是我们 用的比较多的一个布局,我们更多的时候更钟情于他的weight(权重)属性，等比例划分，对屏幕适配还是 帮助蛮大的;但是使用LinearLayout的时候也有一个问题，就是当界面比较复杂的时候，需要嵌套多层的 LinearLayout,这样就会降低UI Render的效率(渲染速度),而且如果是listview或者GridView上的 item,效率会更低,另外太多层LinearLayout嵌套会占用更多的系统资源,还有可能引发stackoverflow; 但是如果我们使用RelativeLayout的话,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

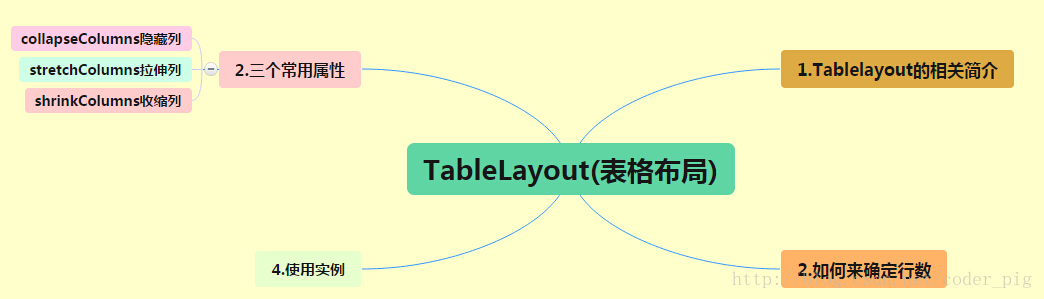
#### 2.2.3.2 margin与padding的区别

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—3.1.5为基础，后续为进阶和精通，按需阅读**

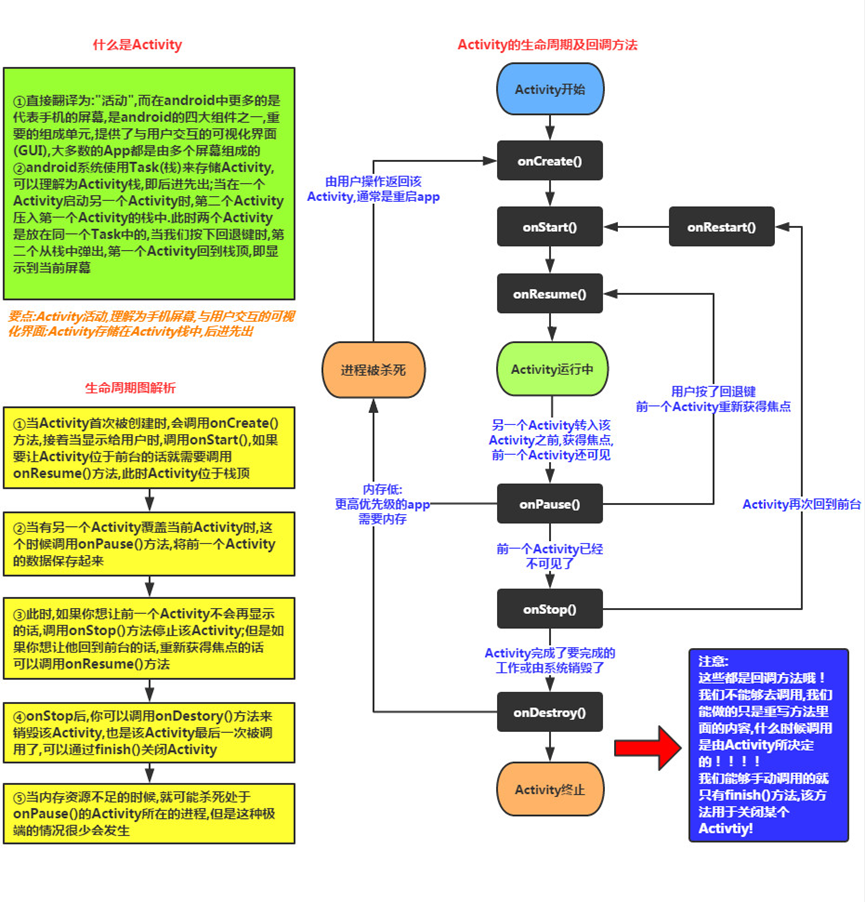
### 3.1.1 Activity的概念与Activity的生命周期图

**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**

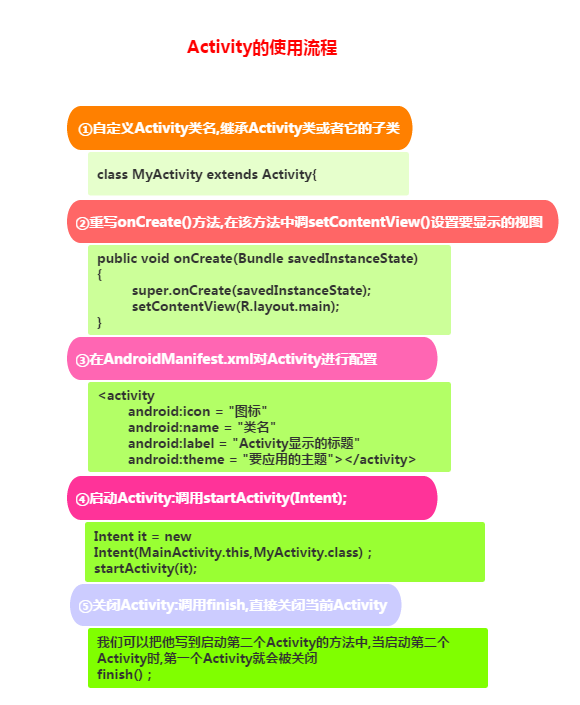
**2.一个App允许有多个Activity**

****

**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);

}

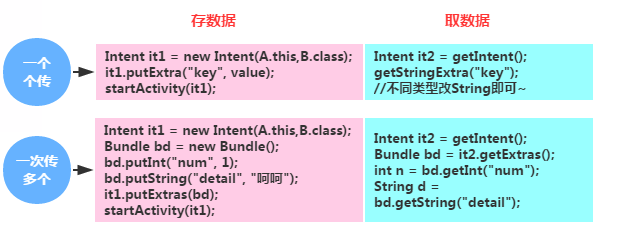
else if (this.getResources().getConfiguration().orientation ==Configuration.ORIENTATION\_PORTRAIT) {

setContentView(R.layout.竖屏);

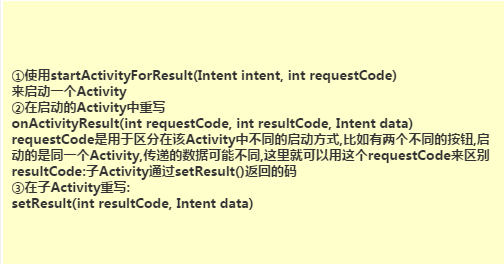
}

### 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);

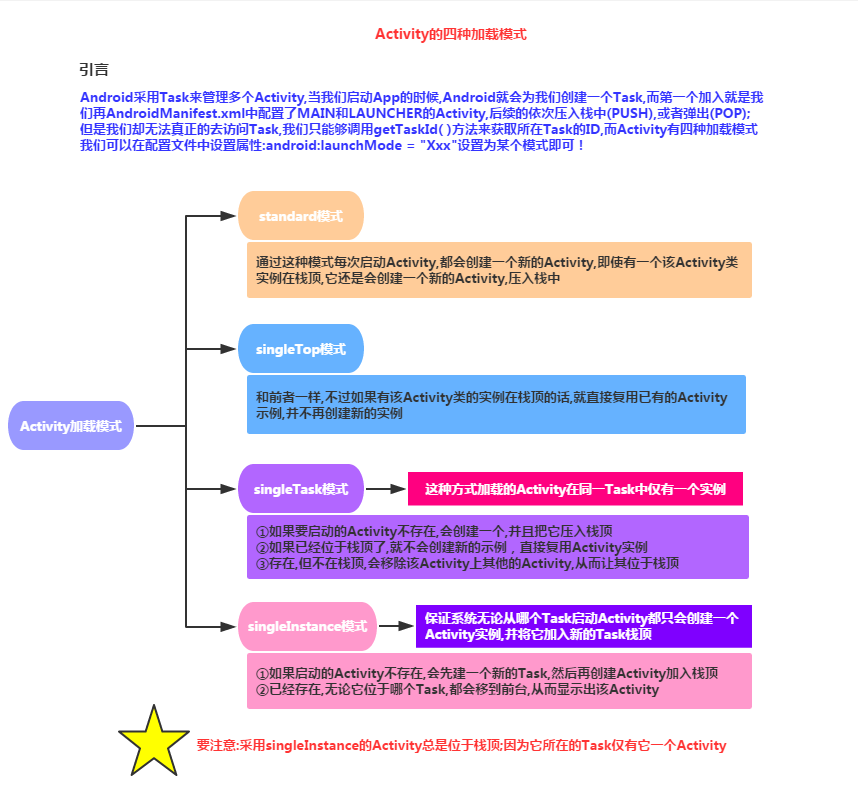
activityMgr.killBackgroundProcesses(context.getPackageName());

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

**3.2.1—3.2.7为基础，后续为进阶和精通，按需阅读**

### 3.2.1 线程

学习service之前先了解下线程相关概念

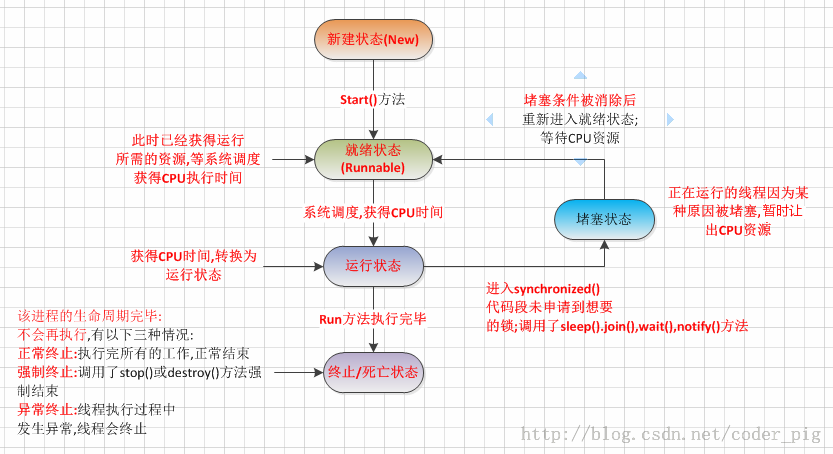
**程序**：为了完成特定任务，用某种语言编写的一组指令集合(一组静态代码)

**进程**：运行中的程序，系统调度与资源分配的一个独立单位，操作系统会 为每个进程分配一段内存空间！程序的依次动态执行，经历代码的加载，执行， 执行完毕的完整过程！

**线程**：比进程更小的执行单元，每个进程可能有多条线程，线程需要放在一个 进程中才能执行，线程由程序负责管理，而进程则由系统进行调度！

**多线程的理解**：并行执行多个条指令，将CPU时间片按照调度算法分配给各个 线程，实际上是分时执行的，只是这个切换的时间很短，用户感觉到"同时"而已！

### 3.2.2 线程的生命周期



### 3.2.3 创建线程的三种方式

**1.继承Thread类**

**2.实现Runnable接口**

**3.实现Callable接口**

如果：使用的是2创建的线程的话，可以直接这样启动：

new **Thread**(myThread).start();

但更多的时候我们喜欢使用匿名类，即下面这种写法：

new **Thread**(new **Runnable**(){

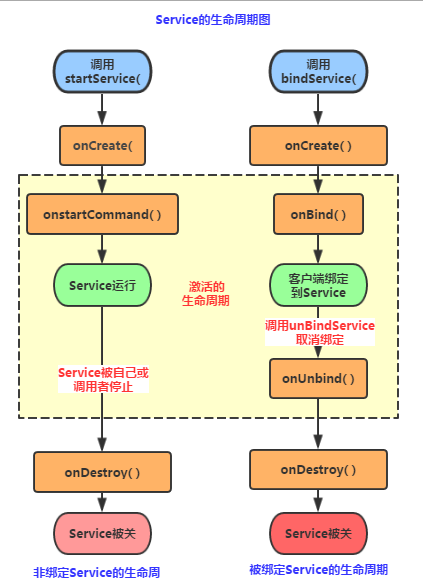
public void run();

}).start();

线程与service并没有太大的关系，不要把这两个混淆了！ Thread是线程，程序执行的最小单元，分配CPU的基本单位！ 而Service则是Android提供一个允许长时间留驻后台的一个组件，最常见的 用法就是做轮询操作！或者想在后台做一些事情，比如后台下载更新！ 记得别把这两个概念混淆！

关于线程先讲到这里，后续会单独写一个单元来归纳线程相关

### 3.2.4 service的生命周期



### 3.2.5 service的使用

好的，从上图的生命周期，我们可以知道，Android中使用Service的方式有两种：

**1）StartService()启动Service**

**2）BindService()启动Service**

PS:还有一种，就是启动Service后，绑定Service！

#### 3.2.5.1 生命周期详解

**onCreate()**：当Service第一次被创建后立即回调该方法，该方法在整个生命周期 中只会调用一次！

**onDestory()**：当Service被关闭时会回调该方法，该方法只会回调一次！

**onStartCommand**(intent,flag,startId)：早期版本是onStart(intent,startId), 当客户端调用**startService**(Intent)方法时会回调，可多次调用StartService方法， 但不会再创建新的Service对象，而是继续复用前面产生的Service对象，但会继续回调 **onStartCommand()**方法！

IBinder **onOnbind**(intent)：该方法是Service都必须实现的方法，该方法会返回一个 IBinder对象，app通过该对象与Service组件进行通信！

**onUnbind(**intent)：当该Service上绑定的所有客户端都断开时会回调该方法！

#### 3.2.5.2 StartService启动Service

①首次启动会创建一个Service实例,依次调用onCreate()和onStartCommand()方法,此时Service 进入运行状态,如果再次调用StartService启动Service,将不会再创建新的Service对象, 系统会直接复用前面创建的Service对象,调用它的onStartCommand()方法！

②但这样的Service与它的调用者无必然的联系,就是说当调用者结束了自己的生命周期, 但是只要不调用stopService,那么Service还是会继续运行的!

③无论启动了多少次Service,只需调用一次StopService即可停掉Service

#### 3.2.5.3 BindService启动Service

①当首次使用bindService绑定一个Service时,系统会实例化一个Service实例,并调用其onCreate()和onBind()方法,然后调用者就可以通过IBinder和Service进行交互了,此后如果再次使用bindService绑定Service,系统不会创建新的Sevice实例,也不会再调用onBind()方法,只会直接把IBinder对象传递给其他后来增加的客户端!

②如果我们解除与服务的绑定,只需调用unbindService(),此时onUnbind和onDestory方法将会被调用!这是一个客户端的情况,假如是多个客户端绑定同一个Service的话,情况如下 当一个客户完成和service之间的互动后，它调用 unbindService() 方法来解除绑定。当所有的客户端都和service解除绑定后，系统会销毁service。（除非service也被startService()方法开启）

③另外,和上面那张情况不同,bindService模式下的Service是与调用者相互关联的,可以理解为 "一条绳子上的蚂蚱",要死一起死,在bindService后,一旦调用者销毁,那么Service也立即终止!

通过BindService调用Service时调用的Context的bindService的解析 bindService(Intent Service,ServiceConnection conn,int flags)

service:通过该intent指定要启动的Service

conn:ServiceConnection对象,用户监听访问者与Service间的连接情况, 连接成功回调该对象中的onServiceConnected(ComponentName,IBinder)方法; 如果Service所在的宿主由于异常终止或者其他原因终止,导致Service与访问者间断开 连接时调用onServiceDisconnected(CompanentName)方法,主动通过unBindService() 方法断开并不会调用上述方法!

flags:指定绑定时是否自动创建Service(如果Service还未创建), 参数可以是0(不自动创建),BIND\_AUTO\_CREATE(自动创建)

#### 3.2.5.4 StartService启动Service后bindService绑定

如果Service已经由某个客户端通过StartService()启动,接下来由其他客户端 再调用bindService(）绑定到该Service后调用unbindService()解除绑定最后在 调用bindService()绑定到Service的话,此时所触发的生命周期方法如下:

**onCreate( )->onStartCommand( )->onBind( )->onUnbind( )->onRebind( )**

PS:前提是:onUnbind()方法返回true!!! 这里或许部分读者有疑惑了,调用了unbindService后Service不是应该调用 onDistory()方法么!其实这是因为这个Service是由我们的StartService来启动的 ,所以你调用onUnbind()方法取消绑定,Service也是不会终止的!

得出的结论: 假如我们使用bindService来绑定一个启动的Service,注意是已经启动的Service!!! 系统只是将Service的内部IBinder对象传递给Activity,并不会将Service的生命周期 与Activity绑定,因此调用unBindService( )方法取消绑定时,Service也不会被销毁！

### 3.2.6 生命周期的实现

代码部分会上传到Github，后续会增加相关代码，算是另一个坑吧

### 3.2.7 结论

**Step 1**:在自定义的Service中继承Binder,实现自己的IBinder对象

**Step 2**:通过onBind( )方法返回自己的IBinder对象

**Step 3**:在绑定该Service的类中定义一个ServiceConnection对象,重写两个方法, onServiceConnected和onDisconnected！然后直接读取IBinder传递过来的参数即可!

使用BindService绑定Service,依次调用onCreate(),onBind()方法, 我们可以在onBind()方法中返回自定义的IBinder对象;再接着调用的是 ServiceConnection的onServiceConnected()方法该方法中可以获得 IBinder对象,从而进行相关操作;当Service解除绑定后会自动调用 onUnbind和onDestroyed方法,当然绑定多客户端情况需要解除所有 的绑定才会调用onDestoryed方法进行销毁哦！

### 3.2.8 IntentService的使用

如果我们直接把 耗时线程放到Service中的onStart()方法中，虽然可以这样做，但是很容易 会引起ANR异常(Application Not Responding)，而Android的官方在介绍 Service有下面这样一段话：

**1.Service不是一个单独的进程,它和它的应用程序在同一个进程中**

**2.Service不是一个线程,这样就意味着我们应该避免在Service中进行耗时操作**

于是乎，Android给我们提供了解决上述问题的替代品，就是下面要讲的IntentService； IntentService是继承与Service并处理异步请求的一个类,在IntentService中有 一个工作线程来处理耗时操作,请求的Intent记录会加入队列

**工作流程：**

客户端通过startService(Intent)来启动IntentService; 我们并不需要手动地区控制IntentService,当任务执行完后,IntentService会自动停止; 可以启动IntentService多次,每个耗时操作会以工作队列的方式在IntentService的 onHandleIntent回调方法中执行,并且每次只会执行一个工作线程,执行完一，再到二这样!

**总结：**

当一个后台的任务,需要分成几个子任务,然后按先后顺序执行,子任务 (简单的说就是异步操作),此时如果我们还是定义一个普通Service然后 在onStart方法中开辟线程,然后又要去控制线程,这样显得非常的繁琐; 此时应该自定义一个IntentService然后再onHandleIntent()方法中完成相关任务！

### 3.2.9 Activity与Service通信

我们前面的操作都是通过Activity启动和停止Service，假如我们启动的是一个下载 的后台Service，而我们想知道Service中下载任务的进度！那么这肯定是需要Service 与Activity进行通信的，而他们之间交流的媒介就是Service中的onBind()方法！ 返回一个我们自定义的Binder对象！

**基本流程如下**：

1.自定义Service中，自定义一个Binder类，然后将需要暴露的方法都写到该类中！

2.Service类中，实例化这个自定义Binder类，然后重写onBind()方法，将这个Binder对象返回！

3.Activity类中实例化一个ServiceConnection对象，重写onServiceConnected()方法，然后 获取Binder对象，然后调用相关方法即可！

### 3.2.10 前台服务即状态栏弹框

学到现在，我们都知道Service一般都是运行在后来的，但是Service的系统优先级 还是比较低的，当系统内存不足的时候，就有可能回收正在后台运行的Service， 对于这种情况我们可以使用前台服务，从而让Service稍微没那么容易被系统杀死， 当然还是有可能被杀死的...所谓的前台服务就是状态栏显示的Notification！

在自定义的Service类中，重写onCreate()，然后根据自己的需求定制Notification； 定制完毕后，调用startForeground(1,notification对象)即可！

核心代码如下：

public void onCreate()

{

super.onCreate();

Notification.Builder localBuilder = new Notification.Builder(this);

localBuilder.setContentIntent(PendingIntent.getActivity(this, 0, new Intent(this, MainActivity.class), 0));

localBuilder.setAutoCancel(false);

localBuilder.setSmallIcon(R.mipmap.ic\_cow\_icon);

localBuilder.setTicker("Foreground Service Start");

localBuilder.setContentTitle("一个前台服务");

localBuilder.setContentText("正在运行...");

startForeground(1, localBuilder.getNotification());

}

### 3.2.11 后台定时service

这部分暂略，基本思路是后台每隔一定时间就执行相应的操作，但android对于省电的优化导致定时不准确，所以基本放弃了用service来更新服务。

### 3.2.12 AIDL跨进程通信

这部分未来会单独开一单元来写

## 3.3 BroadcastReceiver 广播

### 3.3.1 BroadcastReceiver是什么

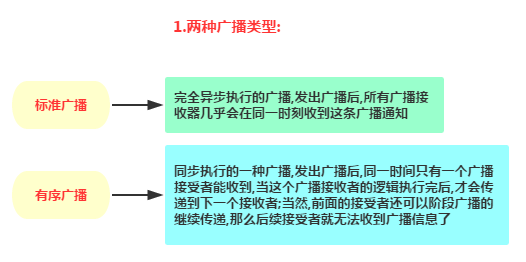
Broadcast直译广播，我们举个形象的例子来帮我理解下BroadcastReceiver，记得以前读书 的时候，每个班级都会有一个挂在墙上的大喇叭，用来广播一些通知，比如，开学要去搬书，广播： "每个班级找几个同学教务处拿书"，发出这个广播后，所有同学都会在同一时刻收到这条广播通知， 收到，但不是每个同学都会去搬书，一般去搬书的都是班里的"大力士"，这群"大力士"接到这条 广播后就会动身去把书搬回可是！

——好吧，上面这个就是一个广播传递的一个很形象的例子：

大喇叭--> 发送广播 --> 所有学生都能收到广播 --> 大力士处理广播

回到我们的概念，其实BroadcastReceiver就是应用程序间的全局大喇叭，即通信的一个手段， 系统自己在很多时候都会发送广播，比如电量低或者充足，刚启动完，插入耳机，输入法改变等， 发生这些时间，系统都会发送广播，这个叫系统广播，每个APP都会收到，如果你想让你的应用在接收到 这个广播的时候做一些操作，比如：系统开机后，偷偷后台跑服务~哈哈，这个时候你只需要为你的应用 注册一个用于监视开机的BroadcastReceiver，当接收到开机广播就做写偷偷摸摸的勾当~ 当然我们也可以自己发广播，比如：接到服务端推送信息，用户在别处登录，然后应该强制用户下线回到 登陆界面，并提示在别处登录~

### 3.3.2 两种广播类型

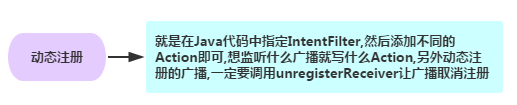


### 3.3.3 接收系统广播

**1）**两种注册广播的方式

前面也讲了，系统在某些时候会发送相应的系统广播，下面我们就来让我们的APP接收系统广播， 接收之前，还需要为我们的APP注册广播接收器哦！

而注册的方法又分为以下两种：动态与静态！



//核心部分代码：MyBRReceiver：自定义的广播类

myReceiver = new MyBRReceiver();

IntentFilter itFilter = new IntentFilter();

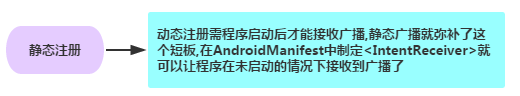
itFilter.addAction("android.net.conn.CONNECTIVITY\_CHANGE");//创建网络变化广播

registerReceiver(myReceiver, itFilter);

//别忘了将广播取消掉哦~ ，写在ondestory中

unregisterReceiver(myReceiver);

动态注册有个缺点就是需要程序启动才可以接收广播，假如我们需要程序 没有启动，但是还是能接收广播的话，那么就需要注册静态广播了！



在AndroidManifest.xml中对该BroadcastReceiver进行注册，添加开机广播的intent-filter!

别忘了加上android.permission.RECEIVE\_BOOT\_COMPLETED的权限哦！

<receiver android:name=".BootCompleteReceiver">

<intent-filter>

<action android:name = "android.intent.cation.BOOT\_COMPLETED">

</intent-filter>

</receiver>

<!-- 权限 -->

<uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED"/>

### 3.3.4 发送广播

**注意！注意！注意！重要的事说三遍**

**不要在广播里添加过多逻辑或者进行任何耗时操作,因为在广播中是不允许开辟线程的, 当onReceiver( )方法运行较长时间(超过10秒)还没有结束的话,那么程序会报错(ANR), 广播更多的时候扮演的是一个打开其他组件的角色,比如启动Service,Notification提示, Activity等！**

发送广播前，要先定义一个接收器，先确定目标，然后再告白！~(●'◡'●)~



发送广播的方法：

sendBroadcast(new Intent("广播的全路径"));

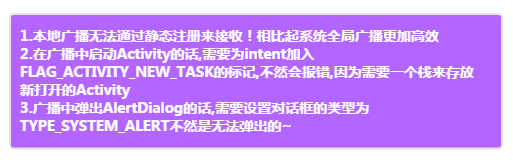
### 3.3.5 本地广播

我们对BroadcastReceiver已经有了一个初步的了解了，知道两种广播类型：标准与有序， 动态或静态注册广播接收者，监听系统广播，自己发送广播！已经满足我们的基本需求了~ 但是前面写的广播都是全局广播！这同样意味着我们APP发出的广播，其他APP都会接收到， 或者其他APP发送的广播，我们的APP也同样会接收到，这样容易引起一些安全性的问题！而 Android中给我们提供了本地广播的机制，使用该机制发出的广播只会在APP内部传播，而且 广播接收者也只能收到本应用发出的广播！

用法：



注意：



### 3.3.6 常用的系统广播总结

**intent.action.AIRPLANE\_MODE;**

//关闭或打开飞行模式时的广播

**Intent.ACTION\_BATTERY\_CHANGED;**

//充电状态，或者电池的电量发生变化

//电池的充电状态、电荷级别改变，不能通过组建声明接收这个广播，只有通过Context.registerReceiver()注册

**Intent.ACTION\_BATTERY\_LOW;**

//表示电池电量低

**Intent.ACTION\_BATTERY\_OKAY;**

//表示电池电量充足，即从电池电量低变化到饱满时会发出广播

**Intent.ACTION\_BOOT\_COMPLETED;**

//在系统启动完成后，这个动作被广播一次（只有一次）。

**Intent.ACTION\_CAMERA\_BUTTON;**

//按下照相时的拍照按键(硬件按键)时发出的广播

**Intent.ACTION\_CLOSE\_SYSTEM\_DIALOGS;**

//当屏幕超时进行锁屏时,当用户按下电源按钮,长按或短按(不管有没跳出话框)，进行锁屏时,android系统都会广播此Action消息

**Intent.ACTION\_CONFIGURATION\_CHANGED;**

//设备当前设置被改变时发出的广播(包括的改变:界面语言，设备方向，等，请参考Configuration.java)

**Intent.ACTION\_DATE\_CHANGED;**

//设备日期发生改变时会发出此广播

**Intent.ACTION\_DEVICE\_STORAGE\_LOW;**

//设备内存不足时发出的广播,此广播只能由系统使用，其它APP不可用？

**Intent.ACTION\_DEVICE\_STORAGE\_OK;**

//设备内存从不足到充足时发出的广播,此广播只能由系统使用，其它APP不可用？

**Intent.ACTION\_DOCK\_EVENT;**

//发出此广播的地方frameworks\base\services\java\com\android\server\DockObserver.java

**Intent.ACTION\_EXTERNAL\_APPLICATIONS\_AVAILABLE;**

////移动APP完成之后，发出的广播(移动是指:APP2SD)

**Intent.ACTION\_EXTERNAL\_APPLICATIONS\_UNAVAILABLE;**

//正在移动APP时，发出的广播(移动是指:APP2SD)

**Intent.ACTION\_GTALK\_SERVICE\_CONNECTED;**

//Gtalk已建立连接时发出的广播

**Intent.ACTION\_GTALK\_SERVICE\_DISCONNECTED;**

//Gtalk已断开连接时发出的广播

**Intent.ACTION\_HEADSET\_PLUG;**

//在耳机口上插入耳机时发出的广播

**Intent.ACTION\_INPUT\_METHOD\_CHANGED;**

//改变输入法时发出的广播

**Intent.ACTION\_LOCALE\_CHANGED;**

//设备当前区域设置已更改时发出的广播

**Intent.ACTION\_MANAGE\_PACKAGE\_STORAGE;**

**Intent.ACTION\_MEDIA\_BAD\_REMOVAL;**

//未正确移除SD卡(正确移除SD卡的方法:设置--SD卡和设备内存--卸载SD卡)，但已把SD卡取出来时发出的广播

//广播：扩展介质（扩展卡）已经从 SD 卡插槽拔出，但是挂载点 (mount point) 还没解除 (unmount)

**Intent.ACTION\_MEDIA\_BUTTON;**

//按下"Media Button" 按键时发出的广播,假如有"Media Button" 按键的话(硬件按键)

**Intent.ACTION\_MEDIA\_CHECKING;**

//插入外部储存装置，比如SD卡时，系统会检验SD卡，此时发出的广播?

**Intent.ACTION\_MEDIA\_EJECT;**

//已拔掉外部大容量储存设备发出的广播（比如SD卡，或移动硬盘）,不管有没有正确卸载都会发出此广播?

//广播：用户想要移除扩展介质（拔掉扩展卡）。

**Intent.ACTION\_MEDIA\_MOUNTED;**

//插入SD卡并且已正确安装（识别）时发出的广播

//广播：扩展介质被插入，而且已经被挂载。

**Intent.ACTION\_MEDIA\_NOFS;**

**Intent.ACTION\_MEDIA\_REMOVED;**

//外部储存设备已被移除，不管有没正确卸载,都会发出此广播？

// 广播：扩展介质被移除。

**Intent.ACTION\_MEDIA\_SCANNER\_FINISHED;**

//广播：已经扫描完介质的一个目录

**Intent.ACTION\_MEDIA\_SCANNER\_SCAN\_FILE;**

**Intent.ACTION\_MEDIA\_SCANNER\_STARTED;**

//广播：开始扫描介质的一个目录

**Intent.ACTION\_MEDIA\_SHARED;**

// 广播：扩展介质的挂载被解除 (unmount)，因为它已经作为 USB 大容量存储被共享。

**Intent.ACTION\_MEDIA\_UNMOUNTABLE;**

**Intent.ACTION\_MEDIA\_UNMOUNTED**

// 广播：扩展介质存在，但是还没有被挂载 (mount)。

**Intent.ACTION\_NEW\_OUTGOING\_CALL;**

**Intent.ACTION\_PACKAGE\_ADDED;**

//成功的安装APK之后

//广播：设备上新安装了一个应用程序包。

//一个新应用包已经安装在设备上，数据包括包名（最新安装的包程序不能接收到这个广播）

**Intent.ACTION\_PACKAGE\_CHANGED;**

//一个已存在的应用程序包已经改变，包括包名

**Intent.ACTION\_PACKAGE\_DATA\_CLEARED;**

//清除一个应用程序的数据时发出的广播(在设置－－应用管理－－选中某个应用，之后点清除数据时?)

//用户已经清除一个包的数据，包括包名（清除包程序不能接收到这个广播）

**Intent.ACTION\_PACKAGE\_INSTALL;**

//触发一个下载并且完成安装时发出的广播，比如在电子市场里下载应用？

**Intent.ACTION\_PACKAGE\_REMOVED;**

//成功的删除某个APK之后发出的广播

//一个已存在的应用程序包已经从设备上移除，包括包名（正在被安装的包程序不能接收到这个广播）

**Intent.ACTION\_PACKAGE\_REPLACED;**

//替换一个现有的安装包时发出的广播（不管现在安装的APP比之前的新还是旧，都会发出此广播？）

**Intent.ACTION\_PACKAGE\_RESTARTED;**

//用户重新开始一个包，包的所有进程将被杀死，所有与其联系的运行时间状态应该被移除，包括包名（重新开始包程序不能接收到这个广播）

**Intent.ACTION\_POWER\_CONNECTED;**

//插上外部电源时发出的广播

**Intent.ACTION\_POWER\_DISCONNECTED;**

//已断开外部电源连接时发出的广播

**Intent.ACTION\_PROVIDER\_CHANGED;**

**Intent.ACTION\_REBOOT;**

//重启设备时的广播

**Intent.ACTION\_SCREEN\_OFF;**

//屏幕被关闭之后的广播

**Intent.ACTION\_SCREEN\_ON;**

//屏幕被打开之后的广播

**Intent.ACTION\_SHUTDOWN;**

//关闭系统时发出的广播

**Intent.ACTION\_TIMEZONE\_CHANGED;**

//时区发生改变时发出的广播

**Intent.ACTION\_TIME\_CHANGED;**

//时间被设置时发出的广播

**Intent.ACTION\_TIME\_TICK;**

//广播：当前时间已经变化（正常的时间流逝）。

//当前时间改变，每分钟都发送，不能通过组件声明来接收，只有通过Context.registerReceiver()方法来注册

**Intent.ACTION\_UID\_REMOVED;**

//一个用户ID已经从系统中移除发出的广播

**Intent.ACTION\_UMS\_CONNECTED;**

//设备已进入USB大容量储存状态时发出的广播？

**Intent.ACTION\_UMS\_DISCONNECTED;**

//设备已从USB大容量储存状态转为正常状态时发出的广播？

**Intent.ACTION\_USER\_PRESENT;**

**Intent.ACTION\_WALLPAPER\_CHANGED;**

//设备墙纸已改变时发出的广播

## 3.4 ContentProvider 内容提供者

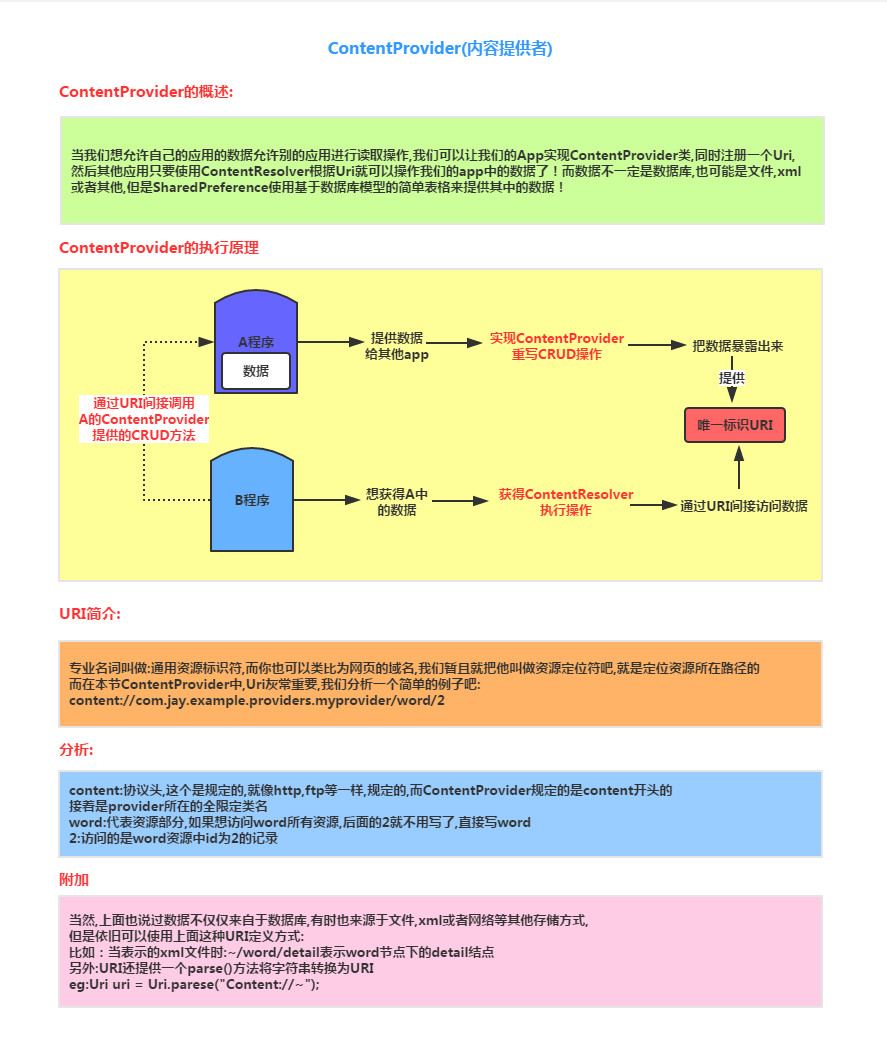
**3.4.1—3.4.4 是基础，之后是进阶和精通，按需阅读**

### 3.4.1 什么是ContentProvider

本节给大家带来的是Android四大组件中的最后一个——ContentProvider(内容提供者)，可能部分读者有疑问了，"Android不是有五大组件的吗？还有个Intent呢？"对的，众所周知四大组件有五个不是很正常的吗！说回这个ContentProvider，我们什么时候 会用到他呢？有下面这两种：

**1.我们想在自己的应用中访问别的应用，或者说一些ContentProvider暴露给我们的一些数据， 比如手机联系人，短信等！我们想对这些数据进行读取或者修改，这就需要用到ContentProvider了！**

**2.我们自己的应用，想把自己的一些数据暴露出来，给其他的应用进行读取或操作，我们也可以用 到ContentProvider，另外我们可以选择要暴露的数据，就避免了我们隐私数据的的泄露！**



### 3.4.2 使用系统提供的ContentProvider

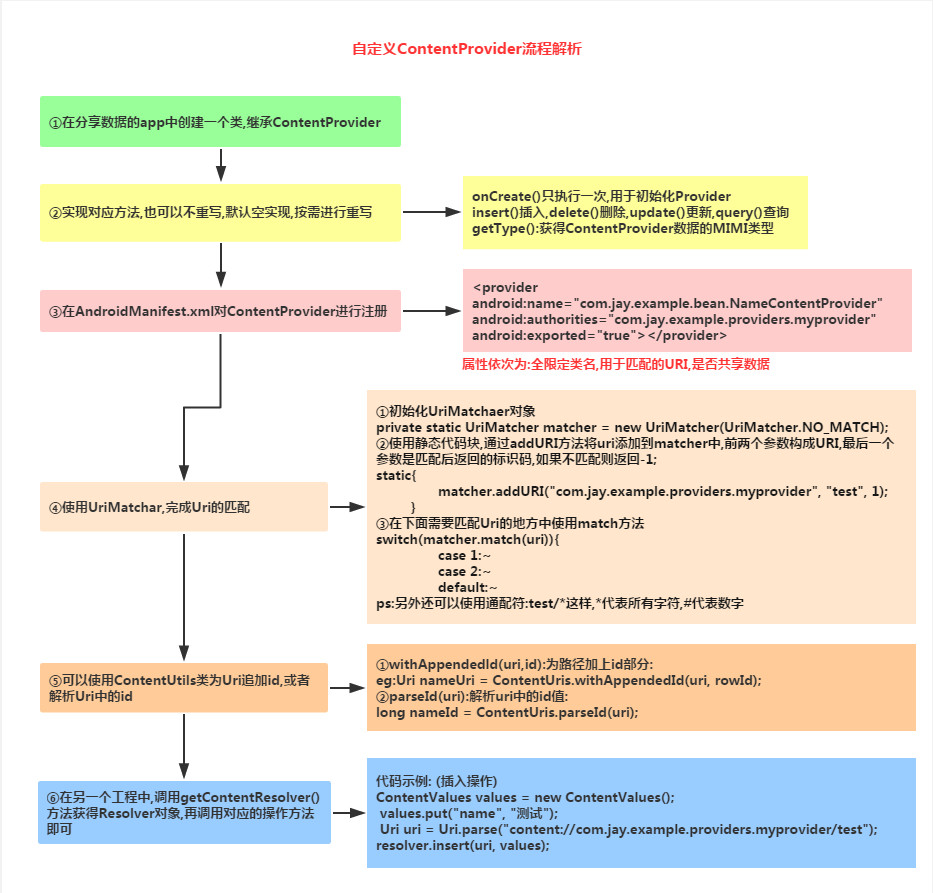
其实很多时候我们用到ContentProvider并不是自己暴露自己的数据，更多的时候通过 ContentResolver来读取其他应用的信息，最常用的莫过于读取系统APP，信息，联系人， 多媒体信息等！如果你想来调用这些ContentProvider就需要自行查阅相关的API资料了！ 另外，不同的版本，可能对应着不同的URL！这里给出如何获取URL与对应的数据库表的字段， 这里以最常用的联系人为例，其他自行google~

①来到系统源码文件下:all-src.rar -> TeleponeProvider -> AndroidManifest.xml查找对应API

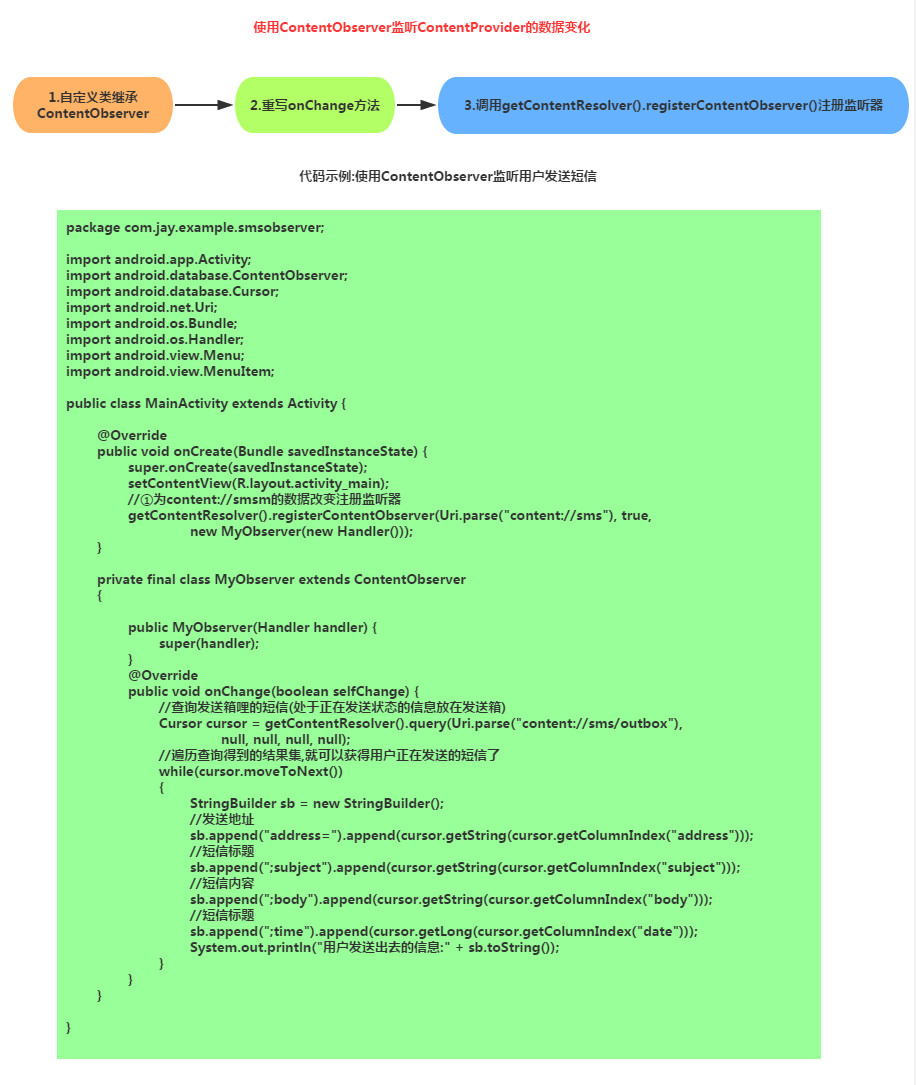
②打开模拟器的file exploer/data/data/com.android.providers.contacts/databases/contact2.db 导出后使用SQLite图形工具查看，三个核心的表:raw\_contact表，data表，mimetypes表！

### 3.4.3 自定义ContentProvider

我们很少会自己来定义ContentProvider，因为我们很多时候都不希望自己应用的数据暴露给 其他应用，虽然这样，学习如何ContentProvider还是有必要的，多一种数据传输的方式，是吧~



### 3.4.4 通过ContentObserver监听ContentProvider的数据变化



### 3.4.5 其他ContentProvider

**待定。To Be Continue. . .**

## 3.5 Intent 意图

### 3.5.1 四大组件间的枢纽

Android通信的桥梁，比如我们可以通过:

**startActivity(Intent)/startActivityForResult(Intent)**：来启动一个Activity

**startService(Intent)/bindService(Intent)**：来启动一个Service

**sendBroadcast**：发送广播到指定BroadcastReceiver

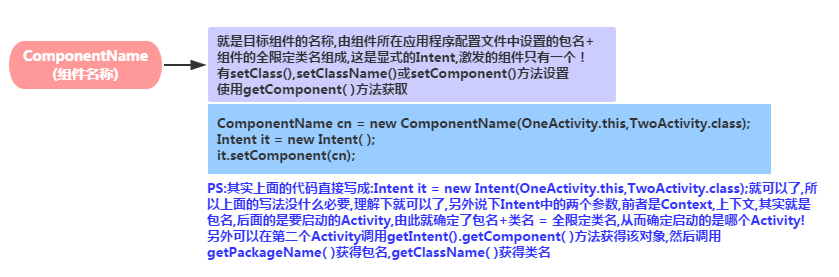
另外别忘了我们在注册四大组件时，写得很多的Intent-Filter哦~

**显式Intent**：通过组件名指定启动的目标组件,比如startActivity(new Intent(A.this,B.class)); 每次启动的组件只有一个~

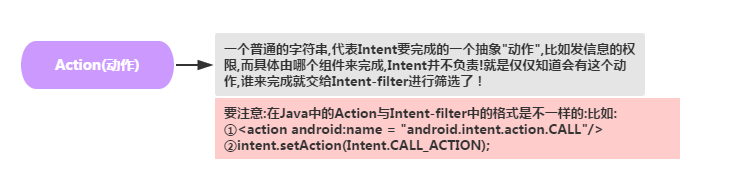
**隐式Intent**:不指定组件名,而指定Intent的Action,Data,或Category,当我们启动组件时, 会去匹配AndroidManifest.xml相关组件的Intent-filter,逐一匹配出满足属性的组件,当不止一个满足时, 会弹出一个让我们选择启动哪个的对话框~

### 3.5.2 Intent的七个属性

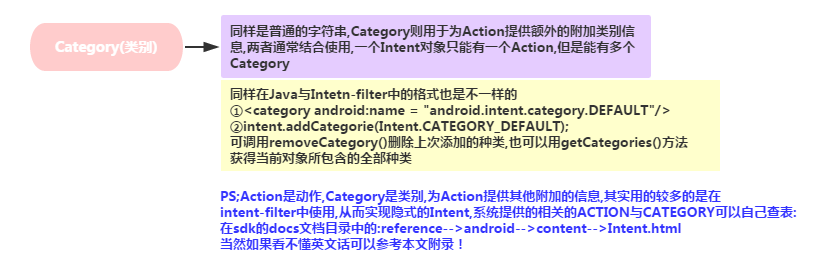
#### 3.5.2.1 ComponentName(组件名称)



#### 3.5.1.2 Action(动作)



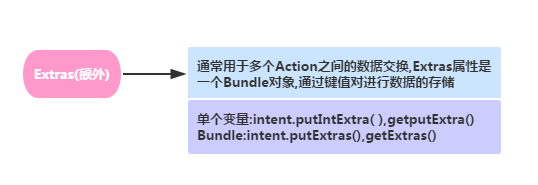
#### 3.5.1.3 Category(类别)



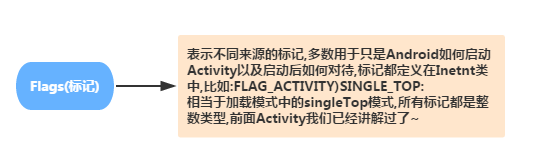
#### 3.5.1.4 Data(数据)，Type(MIME类型)



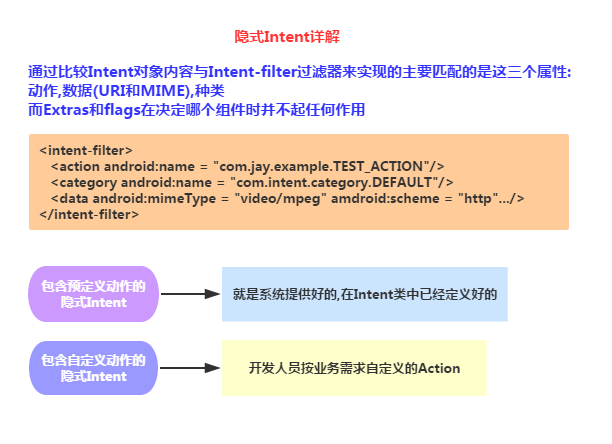
#### 3.5.1.5 Extras(额外)



#### 3.5.1.6 Flags(标记)



### 3.5.3 隐式Intent详解



### 3.5.4 常用系统Intent合集

//===============================================================

//1.拨打电话

// 给移动客服10086拨打电话

Uri uri = Uri.parse("tel:10086");

Intent intent = new Intent(Intent.ACTION\_DIAL, uri);

startActivity(intent);

//===============================================================

//2.发送短信

// 给10086发送内容为“Hello”的短信

Uri uri = Uri.parse("smsto:10086");

Intent intent = new Intent(Intent.ACTION\_SENDTO, uri);

intent.putExtra("sms\_body", "Hello");

startActivity(intent);

//3.发送彩信（相当于发送带附件的短信）

Intent intent = new Intent(Intent.ACTION\_SEND);

intent.putExtra("sms\_body", "Hello");

Uri uri = Uri.parse("content://media/external/images/media/23");

intent.putExtra(Intent.EXTRA\_STREAM, uri);

intent.setType("image/png");

startActivity(intent);

//===============================================================

//4.打开浏览器:

// 打开百度主页

Uri uri = Uri.parse("http://www.baidu.com");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

//===============================================================

//5.发送电子邮件:(阉割了Google服务的没戏!!!!)

// 给someone@domain.com发邮件

Uri uri = Uri.parse("mailto:someone@domain.com");

Intent intent = new Intent(Intent.ACTION\_SENDTO, uri);

startActivity(intent);

// 给someone@domain.com发邮件发送内容为“Hello”的邮件

Intent intent = new Intent(Intent.ACTION\_SEND);

intent.putExtra(Intent.EXTRA\_EMAIL, "someone@domain.com");

intent.putExtra(Intent.EXTRA\_SUBJECT, "Subject");

intent.putExtra(Intent.EXTRA\_TEXT, "Hello");

intent.setType("text/plain");

startActivity(intent);

// 给多人发邮件

Intent intent=new Intent(Intent.ACTION\_SEND);

String[] tos = {"1@abc.com", "2@abc.com"}; // 收件人

String[] ccs = {"3@abc.com", "4@abc.com"}; // 抄送

String[] bccs = {"5@abc.com", "6@abc.com"}; // 密送

intent.putExtra(Intent.EXTRA\_EMAIL, tos);

intent.putExtra(Intent.EXTRA\_CC, ccs);

intent.putExtra(Intent.EXTRA\_BCC, bccs);

intent.putExtra(Intent.EXTRA\_SUBJECT, "Subject");

intent.putExtra(Intent.EXTRA\_TEXT, "Hello");

intent.setType("message/rfc822");

startActivity(intent);

//===============================================================

//6.显示地图:

// 打开Google地图中国北京位置（北纬39.9，东经116.3）

Uri uri = Uri.parse("geo:39.9,116.3");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

//===============================================================

//7.路径规划

// 路径规划：从北京某地（北纬39.9，东经116.3）到上海某地（北纬31.2，东经121.4）

Uri uri = Uri.parse("http://maps.google.com/maps?f=d&saddr=39.9 116.3&daddr=31.2 121.4");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

//===============================================================

//8.多媒体播放:

Intent intent = new Intent(Intent.ACTION\_VIEW);

Uri uri = Uri.parse("file:///sdcard/foo.mp3");

intent.setDataAndType(uri, "audio/mp3");

startActivity(intent);

//获取SD卡下所有音频文件,然后播放第一首=-=

Uri uri = Uri.withAppendedPath(MediaStore.Audio.Media.INTERNAL\_CONTENT\_URI, "1");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

//===============================================================

//9.打开摄像头拍照:

// 打开拍照程序

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

startActivityForResult(intent, 0);

// 取出照片数据

Bundle extras = intent.getExtras();

Bitmap bitmap = (Bitmap) extras.get("data");

//另一种:

//调用系统相机应用程序，并存储拍下来的照片

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

time = Calendar.getInstance().getTimeInMillis();

intent.putExtra(MediaStore.EXTRA\_OUTPUT, Uri.fromFile(new File(Environment

.getExternalStorageDirectory().getAbsolutePath()+"/tucue", time + ".jpg")));

startActivityForResult(intent, ACTIVITY\_GET\_CAMERA\_IMAGE);

//===============================================================

//10.获取并剪切图片

// 获取并剪切图片

Intent intent = new Intent(Intent.ACTION\_GET\_CONTENT);

intent.setType("image/\*");

intent.putExtra("crop", "true"); // 开启剪切

intent.putExtra("aspectX", 1); // 剪切的宽高比为1：2

intent.putExtra("aspectY", 2);

intent.putExtra("outputX", 20); // 保存图片的宽和高

intent.putExtra("outputY", 40);

intent.putExtra("output", Uri.fromFile(new File("/mnt/sdcard/temp"))); // 保存路径

intent.putExtra("outputFormat", "JPEG");// 返回格式

startActivityForResult(intent, 0);

// 剪切特定图片

Intent intent = new Intent("com.android.camera.action.CROP");

intent.setClassName("com.android.camera", "com.android.camera.CropImage");

intent.setData(Uri.fromFile(new File("/mnt/sdcard/temp")));

intent.putExtra("outputX", 1); // 剪切的宽高比为1：2

intent.putExtra("outputY", 2);

intent.putExtra("aspectX", 20); // 保存图片的宽和高

intent.putExtra("aspectY", 40);

intent.putExtra("scale", true);

intent.putExtra("noFaceDetection", true);

intent.putExtra("output", Uri.parse("file:///mnt/sdcard/temp"));

startActivityForResult(intent, 0);

//===============================================================

//11.打开Google Market

// 打开Google Market直接进入该程序的详细页面

Uri uri = Uri.parse("market://details?id=" + "com.demo.app");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

//===============================================================

//12.进入手机设置界面:

// 进入无线网络设置界面（其它可以举一反三）

Intent intent = new Intent(android.provider.Settings.ACTION\_WIRELESS\_SETTINGS);

startActivityForResult(intent, 0);

//===============================================================

//13.安装apk:

Uri installUri = Uri.fromParts("package", "xxx", null);

returnIt = new Intent(Intent.ACTION\_PACKAGE\_ADDED, installUri);

//===============================================================

//14.卸载apk:

Uri uri = Uri.fromParts("package", strPackageName, null);

Intent it = new Intent(Intent.ACTION\_DELETE, uri);

startActivity(it);

//===============================================================

//15.发送附件:

Intent it = new Intent(Intent.ACTION\_SEND);

it.putExtra(Intent.EXTRA\_SUBJECT, "The email subject text");

it.putExtra(Intent.EXTRA\_STREAM, "file:///sdcard/eoe.mp3");

sendIntent.setType("audio/mp3");

startActivity(Intent.createChooser(it, "Choose Email Client"));

//===============================================================

//16.进入联系人页面:

Intent intent = new Intent();

intent.setAction(Intent.ACTION\_VIEW);

intent.setData(People.CONTENT\_URI);

startActivity(intent);

//===============================================================

//17.查看指定联系人:

Uri personUri = ContentUris.withAppendedId(People.CONTENT\_URI, info.id);//info.id联系人ID

Intent intent = new Intent();

intent.setAction(Intent.ACTION\_VIEW);

intent.setData(personUri);

startActivity(intent);

//===============================================================

//18.调用系统编辑添加联系人（高版本SDK有效）：

Intent it = newIntent(Intent.ACTION\_INSERT\_OR\_EDIT);

it.setType("vnd.android.cursor.item/contact");

//it.setType(Contacts.CONTENT\_ITEM\_TYPE);

it.putExtra("name","myName");

it.putExtra(android.provider.Contacts.Intents.Insert.COMPANY, "organization");

it.putExtra(android.provider.Contacts.Intents.Insert.EMAIL,"email");

it.putExtra(android.provider.Contacts.Intents.Insert.PHONE,"homePhone");

it.putExtra(android.provider.Contacts.Intents.Insert.SECONDARY\_PHONE,"mobilePhone");

it.putExtra( android.provider.Contacts.Intents.Insert.TERTIARY\_PHONE,"workPhone");

it.putExtra(android.provider.Contacts.Intents.Insert.JOB\_TITLE,"title");

startActivity(it);

//===============================================================

//19.调用系统编辑添加联系人（全有效）：

Intent intent = newIntent(Intent.ACTION\_INSERT\_OR\_EDIT);

intent.setType(People.CONTENT\_ITEM\_TYPE);

intent.putExtra(Contacts.Intents.Insert.NAME, "My Name");

intent.putExtra(Contacts.Intents.Insert.PHONE, "+1234567890");

intent.putExtra(Contacts.Intents.Insert.PHONE\_TYPE,Contacts.PhonesColumns.TYPE\_MOBILE);

intent.putExtra(Contacts.Intents.Insert.EMAIL, "com@com.com");

intent.putExtra(Contacts.Intents.Insert.EMAIL\_TYPE, Contacts.ContactMethodsColumns.TYPE\_WORK);

startActivity(intent);

//===============================================================

//20.打开另一程序

Intent i = new Intent();

ComponentName cn = new ComponentName("com.example.jay.test",

"com.example.jay.test.MainActivity");

i.setComponent(cn);

i.setAction("android.intent.action.MAIN");

startActivityForResult(i, RESULT\_OK);

//===============================================================

//21.打开录音机

Intent mi = new Intent(Media.RECORD\_SOUND\_ACTION);

startActivity(mi);

//===============================================================

//22.从google搜索内容

Intent intent = new Intent();

intent.setAction(Intent.ACTION\_WEB\_SEARCH);

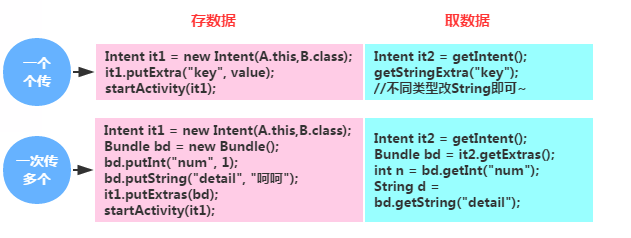
intent.putExtra(SearchManager.QUERY,"searchString")

startActivity(intent);

//===============================================================

### 3.5.5 Intent数据的传递

#### 3.5.5.1 Intent传递简单数据



#### 3.5.5.2 Intent传递数组

**写入数组**：bd.putStringArray("StringArray", new String[]{"呵呵","哈哈"});

//可把StringArray换成其他数据类型,比如int,float等等...

**读取数组**：String[] str = bd.getStringArray("StringArray")

#### 3.5.5.3 Intent传递集合

1）**List<基本数据类型或String>**

写入集合：intent.putStringArrayListExtra(name, value)

读取集合：intent.getStringArrayListExtra(name)

2）**List< Object>**

将list强转成Serializable类型,然后传入(可用Bundle做媒介)

写入集合：putExtras(key, (Serializable)list)

读取集合：(List<Object>) getIntent().getSerializable(key)

PS:Object类需要实现Serializable接口

3）**Map<String, Object>,或更复杂的**

解决方法是：外层套个List

//传递复杂些的参数

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("key1", "value1");

map1.put("key2", "value2");

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();

list.add(map1);

Intent intent = new Intent();

intent.setClass(MainActivity.this,ComplexActivity.class);

Bundle bundle = new Bundle();

//须定义一个list用于在budnle中传递需要传递的ArrayList<Object>,这个是必须要的

ArrayList bundlelist = new ArrayList();

bundlelist.add(list);

bundle.putParcelableArrayList("list",bundlelist);

intent.putExtras(bundle);

startActivity(intent);

#### 3.5.5.4 Intent传递对象

传递对象的方式有两种：将对象转换为Json字符串或者通过Serializable,Parcelable序列化 不建议使用Android内置的抠脚Json解析器，可使用fastjson或者Gson第三方库！

1）**将对象转换为Json字符串**

Gson解析的例子：

**Model:**

public class Book{

private int id;

private String title;

//...

}

public class Author{

private int id;

private String name;

//...

}

**写入数据**：

Book book=new Book();

book.setTitle("Java编程思想");

Author author=new Author();

author.setId(1);

author.setName("Bruce Eckel");

book.setAuthor(author);

Intent intent=new Intent(this,SecondActivity.class);

intent.putExtra("book",new Gson().toJson(book));

startActivity(intent);

**读取数据：**

String bookJson=getIntent().getStringExtra("book");

Book book=new Gson().fromJson(bookJson,Book.class);

Log.d(TAG,"book title->"+book.getTitle());

Log.d(TAG,"book author name->"+book.getAuthor().getName());

2）**使用Serializable,Parcelable序列化对象**

1.Serializable实现:

①业务Bean实现：Serializable接口,写上getter和setter方法

②Intent通过调用putExtra(String name, Serializable value)传入对象实例 当然对象有多个的话多个的话,我们也可以先Bundle.putSerializable(x,x);

③新Activity调用getSerializableExtra()方法获得对象实例: eg:Product pd = (Product) getIntent().getSerializableExtra("Product");

④调用对象get方法获得相应参数

2.Parcelable实现:

一般流程:

①业务Bean继承Parcelable接口,重写writeToParcel方法,将你的对象序列化为一个Parcel对象;

②重写describeContents方法，内容接口描述，默认返回0就可以

③实例化静态内部对象CREATOR实现接口Parcelable.Creator

④同样式通过Intent的putExtra()方法传入对象实例,当然多个对象的话,我们可以先 放到Bundle里Bundle.putParcelable(x,x),再Intent.putExtras()即可

一些解释:

通过writeToParcel将你的对象映射成Parcel对象，再通过createFromParcel将Parcel对象映射 成你的对象。也可以将Parcel看成是一个流，通过writeToParcel把对象写到流里面， 在通过createFromParcel从流里读取对象，只不过这个过程需要你来实现，因此写的 顺序和读的顺序必须一致。

#### 3.5.5.5 Intent传递Bitmap

bitmap默认实现Parcelable接口,直接传递即可

实现代码：

Bitmap bitmap = null;

Intent intent = new Intent();

Bundle bundle = new Bundle();

bundle.putParcelable("bitmap", bitmap);

intent.putExtra("bundle", bundle);

# 4. Fragment

## 4.1 什么是Fragment

Fragment是Android3.0后引入的一个新的API，他出现的初衷是为了适应大屏幕的平板电脑， 当然现在他仍然是平板APP UI设计的宠儿，而且我们普通手机开发也会加入这个Fragment， 我们可以把他看成一个小型的Activity，又称Activity片段！想想，如果一个很大的界面，我们 就一个布局，写起界面来会有多麻烦，而且如果组件多的话是管理起来也很麻烦！而使用Fragment 我们可以把屏幕划分成几块，然后进行分组，进行一个模块化的管理！从而可以更加方便的在 运行过程中动态地更新Activity的用户界面！另外Fragment并不能单独使用，他需要嵌套在Activity 中使用，尽管他拥有自己的生命周期，但是还是会受到宿主Activity的生命周期的影响，比如Activity 被destory销毁了，他也会跟着销毁！Fragment需要嵌套在Activity中使用,当然也可以嵌套到另外一个Fragment中,但这个被嵌套 的Fragment也是需要嵌套在Activity中的,间接地说,Fragment还是需要嵌套在Activity中!! 受寄主Activity的生命周期影响,当然他也有自己的生命周期!另外不建议在Fragment里面 嵌套Fragment因为嵌套在里面的Fragment生命周期不可控!!!

官方文档说创建Fragment时至少需要实现三个方法:onCreate( ),onCreateView( ),OnPause( ); 不过貌似只写一个onCreateView也是可以的...

Fragment的生命周期和Activity有点类似:三种状态:

**Resumed**:在允许中的Fragment可见

**Paused**:所在Activity可见,但是得不到焦点

**Stoped**: ①调用addToBackStack(),Fragment被添加到Bcak栈 ②该Activity转向后台,或者该Fragment被替换/删除

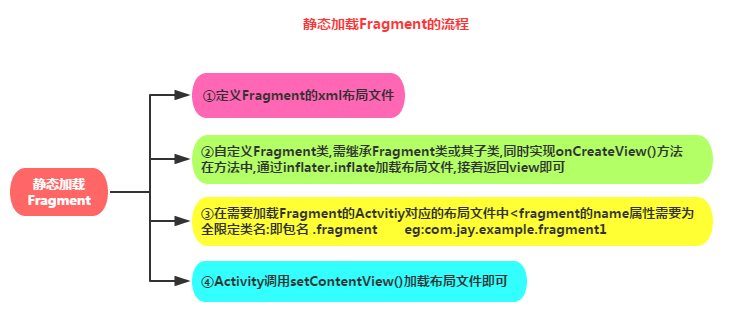
ps:停止状态的fragment仍然活着(所有状态和成员信息被系统保持着),然而,它对用户 不再可见,并且如果activity被干掉,他也会被干掉.

## 4.2 Fragment的生命周期



## 4.3 Fragment加载方式

### 4.3.1 静态加载Fragment



**Step 1**:定义Fragment的布局，就是fragment显示内容的

**Step 2**:自定义一个Fragment类,需要继承Fragment或者他的子类,重写onCreateView()方法 在该方法中调用:inflater.inflate()方法加载Fragment的布局文件,接着返回加载的view对象

public class Fragmentone extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment1, container,false);

return view;

}

}

**Step 3**:在需要加载Fragment的Activity对应的布局文件中添加fragment的标签， 记住，name属性是全限定类名哦，就是要包含Fragment的包名，如:

<fragment

android:id="@+id/fragment1"

android:name="全路径

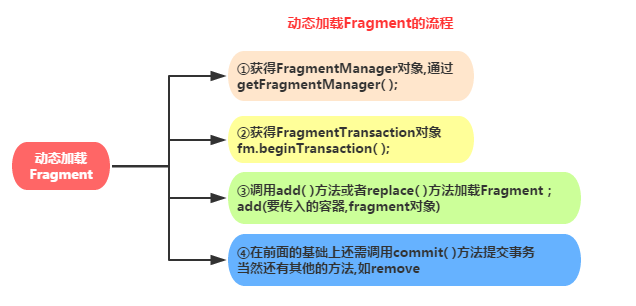
android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1" />

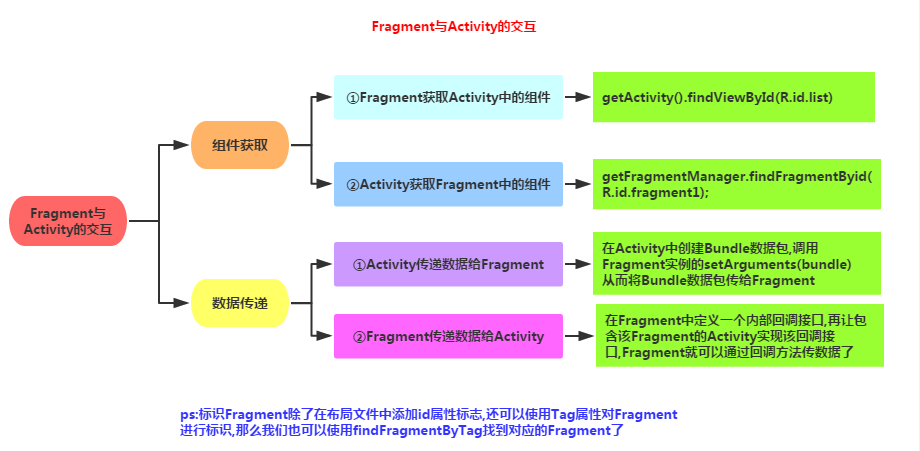
**Step 4**: Activity在onCreate( )方法中调用setContentView()加载布局文件即可!

### 4.3.2 动态加载Fragment



**getFragmentManager().beginTransaction().replace(R.id.LinearLayout1, new Fragment()).commit();**

## 4.4 Fragment与Activity的交互



## 4.5 简单的底部导航栏实现

待定，未来会更新到GitHub

## 4.6 Fragment与ViewPager的联合实现

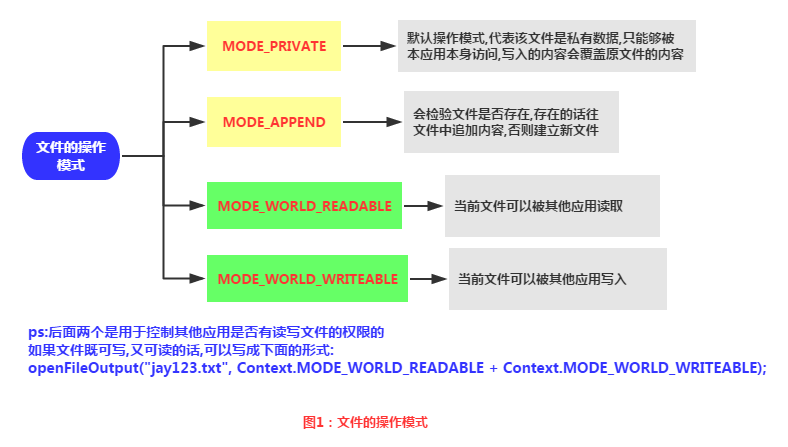
待定，未来会更新到GitHub

# 5.数据存储

## 5.1 文件存储读写

### 5.1.1 Android文件的操作模式

学过Java的同学都知道，我们新建文件，然后就可以写入数据了，但是Android却不一样，因为Android是 基于Linux的，我们在读写文件的时候，还需加上文件的操作模式，Android中的操作模式如下：



### 5.1.2 文件的相关操作方法



读取SD卡的相关操作和上面一致只是多了个判断是否有SD卡和是否有SD卡权限。并且现在手机存储空间越来越大所以这里略过。

### 5.1.3 读取raw和assets文件夹下的文件

相信大家对两个文件夹并不陌生，如果我们不想自己的文件被编译成二进制文件的话， 我们可以把文件放到这两个目录下，而两者的区别如下：

**res/raw**：文件会被映射到R.java文件中，访问的时候直接通过资源ID即可访问，而且 他不能有目录结构，就是不能再创建文件夹

**assets**：不会映射到R.java文件中，通过AssetManager来访问，能有目录结构，即， 可以自行创建文件夹

读取文件资源：

**res/raw**：InputStream is =getResources().openRawResource(R.raw.filename);

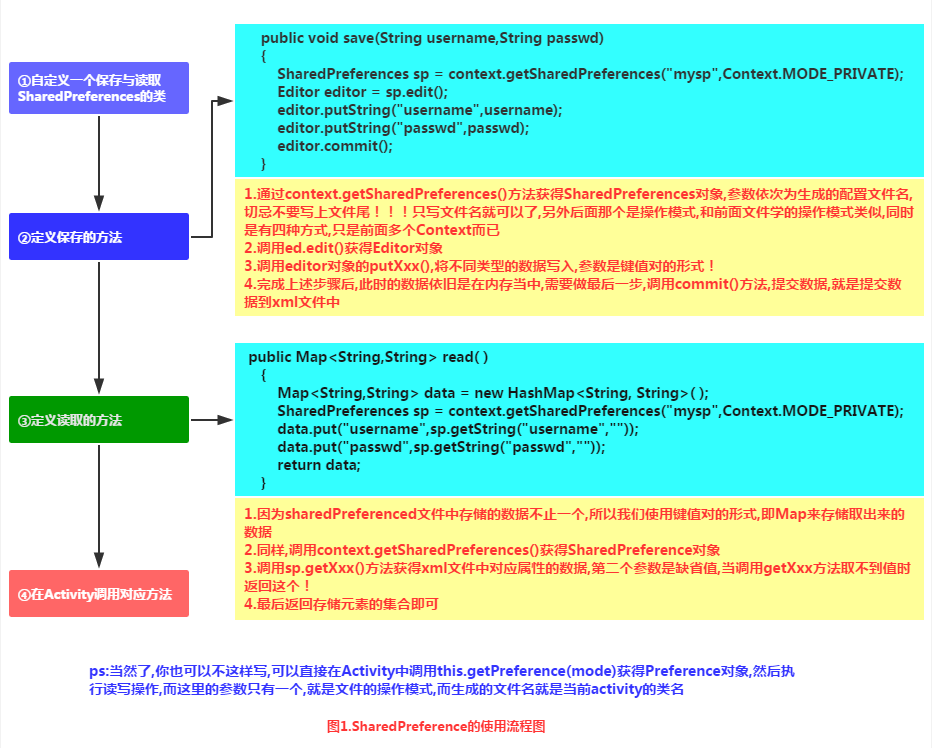
**assets：**AssetManager am = getAssets(); InputStream is = am.open("filename");

## 5.2 SharedPreference

### 5.2.1 什么是SharedPreference

使用SharedPreferences(保存用户偏好参数)保存数据， 当我们的应用想要保存用户的一些偏好参数，比如是否自动登陆，是否记住账号密码,是否在Wifi下才能 联网等相关信息,如果使用数据库的话,显得有点大材小用了！我们把上面这些配置信息称为用户的偏好设置，就是用户偏好的设置，而这些配置信息通常是保存在特定的文件中！比如windows使用ini文件， 而J2SE中使用properties属性文件与xml文件来保存软件的配置信息;而在Android中我们通常使用 一个轻量级的存储类——SharedPreferences来保存用户偏好的参数！SharedPreferences也是使用xml文件, 然后类似于Map集合,使用键-值的形式来存储数据;我们只需要调用SharedPreferences的getXxx(name), 就可以根据键获得对应的值！使用起来很方便！

### 5.2.2 SharedPreference使用方法



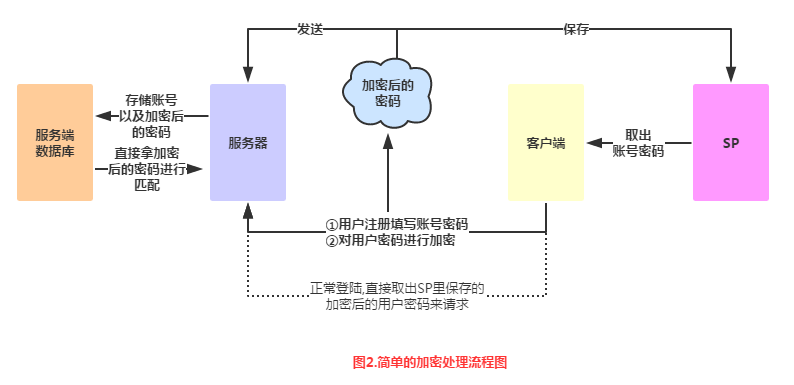
### 5.2.3 读取其他应用的SharedPreferences

获得其他app的Context,而这个Context代表访问该app的全局信息的接口,而决定应用的唯一标识 是应用的包名,所以我们可以通过应用包名获得对应app的Context 另外有一点要注意的是：其他应用的SP文件是否能被读写的前提就是SP文件是否指定了可读或者 可写的权限，我们上面创建的是MODE\_PRIVATE的就不可以了~所以说你像读别人的SP里的数据， 很难，另外，一些关键的信息，比如密码保存到SP里，一般都是会做加密的，所以只能自己写自己玩~ 等下会讲下常用的MD5加密方法！



### 5.2.4 数据加密

之所以写在这个地方是引申阅读，因为经常需要用到加密数据保存密码，这里只写一个常用的加密方式其他可自行百度。



1）MD5是什么？

Message Digest Algorithm MD5（中文名为消息摘要算法第五版）为计算机安全领域广泛 使用的一种散列函数，用以提供消息的完整性保护——摘自《百度百科》 简单点说就是一种加密算法，可以将一个字符串，或者文件，压缩包，执行MD5加密后， 就可以生产一个固定长度为128bit的串！这个串基本唯一！另外我们都知道:一个十六进制 需要用4个bit来表示，那么对应的MD5的字符串长度就为：128 / 4 = 32位了！另外可能 你看到一些md5是16位的，只是将32位MD5码去掉了前八位以及后八位！

2）MD5能破解吗？

答：MD5不可逆，就是说没有对应的算法，无法从生成的md5值逆向得到原始数据！ 当然暴力破解除外，简单的MD5加密后可以查MD5库~

3）MD5值唯一吗？

答：不唯一，一个原始数据只对应一个MD5值，但是一个MD5值可能对应多个原始数据！

## 5.3 SQLite

### 5.3.1 SQLite是什么？

①SQLite是一个轻量级的关系型数据库，运算速度快，占用资源少，很适合在移动设备上使用， 不仅支持标准SQL语法，还遵循ACID(数据库事务)原则，无需账号，使用起来非常方便！

②前面我们学习了使用文件与SharedPreference来保存数据,但是在很多情况下， 文件并不一定是有效的,如多线程并发访问是相关的；app要处理可能变化的复杂数据结构等等！ 比如银行的存钱与取钱！使用前两者就会显得很无力或者繁琐，数据库的出现可以解决这种问题， 而Android又给我们提供了这样一个轻量级的SQLite，为何不用？

③SQLite支持五种数据类型:NULL,INTEGER,REAL(浮点数),TEXT(字符串文本)和BLOB(二进制对象) 虽然只有五种,但是对于varchar,char等其他数据类型都是可以保存的;因为SQLite有个最大的特点: 你可以各种数据类型的数据保存到任何字段中而不用关心字段声明的数据类型是什么,比如你 可以在Integer类型的字段中存放字符串,当然除了声明为主键INTEGER PRIMARY KEY的字段只能够存储64位整数！ 另外， SQLite 在解析CREATE TABLE 语句时， 会忽略 CREATE TABLE 语句中跟在字段名后面的数据类型信息如下面语句会忽略 name字段的类型信息： CREATE TABLE person (personid integer primary key autoincrement, name varchar(20))

**小结下特点：**

SQlite通过文件来保存数据库，一个文件就是一个数据库，数据库中又包含多个表格，表格里又有 多条记录，每个记录由多个字段构成，每个字段有对应的值，每个值我们可以指定类型，也可以不指定类型(主键除外)

我们先来说下几个 我们在使用数据库时用到的三个类：

**SQLiteOpenHelper**：抽象类，我们通过继承该类，然后重写数据库创建以及更新的方法， 我们还可以通过该类的对象获得数据库实例，或者关闭数据库！

**SQLiteDatabase**：数据库访问类：我们可以通过该类的对象来对数据库做一些增删改查的操作

**Cursor**：游标，有点类似于JDBC里的resultset，结果集！可以简单理解为指向数据库中某 一个记录的指针！

### 5.3.2 使用SQLiteOpenHelper类创建数据库与版本管理

对于涉及数据库的app,我们不可能手动地去给他创建数据库文件,所以需要在第一次启用app 的时候就创建好数据库表;而当我们的应用进行升级需要修改数据库表的结构时,这个时候就需要 对数据库表进行更新了;对于这两个操作,安卓给我们提供了SQLiteOpenHelper的两个方法, **onCreate( )**与**onUpgrade( )**来实现

方法解析：

**onCreate(database)**:首次使用软件时生成数据库表

**onUpgrade(database,oldVersion,newVersion)**:在数据库的版本发生变化时会被调用， 一般在软件升级时才需改变版本号，而数据库的版本是由程序员控制的，假设数据库现在的 版本是1，由于业务的变更，修改了数据库表结构，这时候就需要升级软件，升级软件时希望 更新用户手机里的数据库表结构，为了实现这一目的，可以把原来的数据库版本设置为2 或者其他与旧版本号不同的数字即可！

### 5.3.3 使用SQLite

#### 5.3.3.1 创建表和索引

为了创建表和索引，需要调用 SQLiteDatabase 的 execSQL() 方法来执行 DDL 语句。如果没有异常，这个方法没有返回值。

例如，你可以执行如下代码：

**db.execSQL("CREATE TABLE mytable (\_id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT, value REAL);");**

这条语句会创建一个名为 mytable 的表，表有一个列名为 \_id，并且是主键，这列的值是会自动增长的整数（例如，当你插入一行时，SQLite 会给这列自动赋值），另外还有两列：title( 字符 ) 和 value( 浮点数 )。 SQLite 会自动为主键列创建索引。

通常情况下，第一次创建数据库时创建了表和索引。如果你不需要改变表的 schema，不需要删除表和索引 . 删除表和索引，需要使用 execSQL() 方法调用 DROP INDEX 和 DROP TABLE 语句。

#### 5.3.3.2 给表添加数据

上面的代码，已经创建了数据库和表，现在需要给表添加数据。有两种方法可以给表添加数据。

像上面创建表一样，你可以使用 execSQL() 方法执行 INSERT, UPDATE, DELETE 等语句来更新表的数据。execSQL() 方法适用于所有不返回结果的 SQL 语句。例如：

**db.execSQL("INSERT INTO widgets (name, inventory)"+ "VALUES ('Sprocket', 5)");**

另一种方法是使用 SQLiteDatabase 对象的 insert(), update(), delete() 方法。这些方法把 SQL 语句的一部分作为参数。示例如下：

**ContentValues cv=new ContentValues();**

**cv.put(Constants.TITLE, "example title");**

**cv.put(Constants.VALUE, SensorManager.GRAVITY\_DEATH\_STAR\_I);**

**db.insert("mytable", getNullColumnHack(), cv);**

update（）方法有四个参数，分别是表名，表示列名和值的 ContentValues 对象，可选的 WHERE 条件和可选的填充 WHERE 语句的字符串，这些字符串会替换 WHERE 条件中的“？”标记。update() 根据条件，更新指定列的值，所以用 execSQL() 方法可以达到同样的目的。

WHERE 条件和其参数和用过的其他 SQL APIs 类似。例如：

**String[] parms=new String[] {"this is a string"};**

**db.update("widgets", replacements, "name=?", parms);**

delete() 方法的使用和 update() 类似，使用表名，可选的 WHERE 条件和相应的填充 WHERE 条件的字符串。

#### 5.3.3.3 查询数据库

类似 INSERT, UPDATE, DELETE，有两种方法使用 SELECT 从 SQLite 数据库检索数据。

使用 rawQuery() 直接调用 SELECT 语句；

使用 query() 方法构建一个查询。

Raw Queries

正如 API 名字，rawQuery() 是最简单的解决方法。通过这个方法你就可以调用 SQL SELECT 语句。例如：

**Cursor c=db.rawQuery( "SELECT name FROM sqlite\_master WHERE type='table' AND name='mytable'", null);**

在上面例子中，我们查询 SQLite 系统表（sqlite\_master）检查 table 表是否存在。返回值是一个 cursor 对象，这个对象的方法可以迭代查询结果。

如果查询是动态的，使用这个方法就会非常复杂。例如，当你需要查询的列在程序编译的时候不能确定，这时候使用 query() 方法会方便很多。

Regular Queries

query() 方法用 SELECT 语句段构建查询。SELECT 语句内容作为 query() 方法的参数，比如：要查询的表名，要获取的字段名，WHERE 条件，包含可选的位置参数，去替代 WHERE 条件中位置参数的值，GROUP BY 条件，HAVING 条件。

除了表名，其他参数可以是 null。所以，以前的代码段可以可写成：

**String[] columns={"ID", "inventory"};**

**String[] parms={"snicklefritz"};**

**Cursor result=db.query("widgets", columns, "name=?",parms, null, null, null);**

#### 5.3.3.4 使用游标

不管你如何执行查询，都会返回一个 Cursor，这是 Android 的 SQLite 数据库游标，使用游标，你可以：

通过使用 getCount() 方法得到结果集中有多少记录；

通过 moveToFirst(), moveToNext(), 和 isAfterLast() 方法遍历所有记录；

通过 getColumnNames() 得到字段名；

通过 getColumnIndex() 转换成字段号；

通过 getString()，getInt() 等方法得到给定字段当前记录的值；

通过 requery() 方法重新执行查询得到游标；

通过 close() 方法释放游标资源；

例如，下面代码遍历 mytable 表

**Cursor result=db.rawQuery("SELECT ID, name, inventory FROM mytable");**

**result.moveToFirst();**

**while (!result.isAfterLast()) {**

**int id=result.getInt(0);**

**String name=result.getString(1);**

**int inventory=result.getInt(2);**

**// do something useful with these**

**result.moveToNext();**

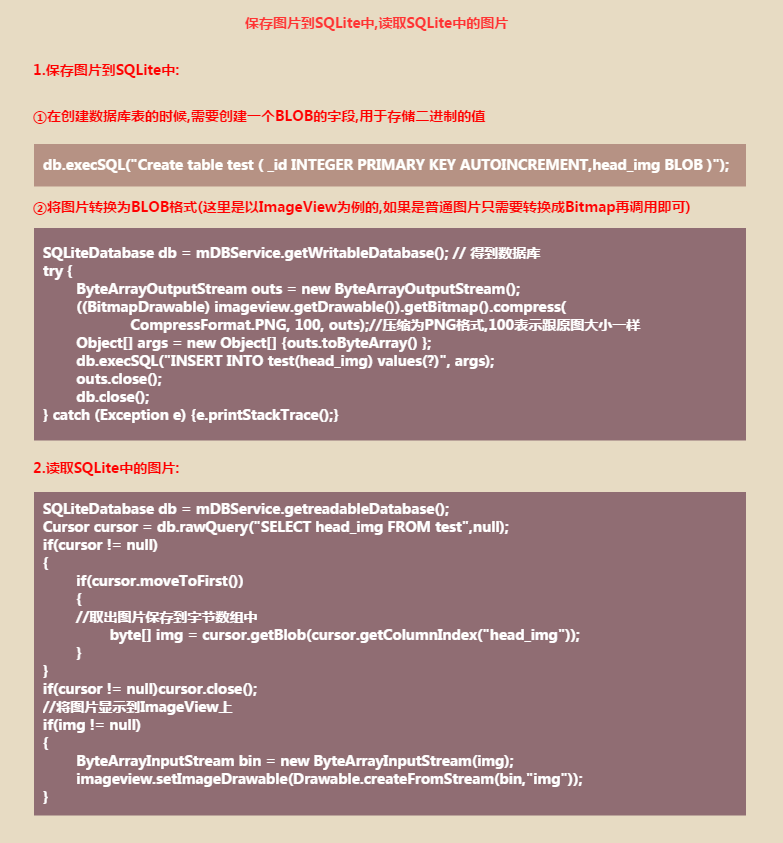
**}**

**result.close();**

### 5.3.4 事务



### 5.3.5 保存大文件



# 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