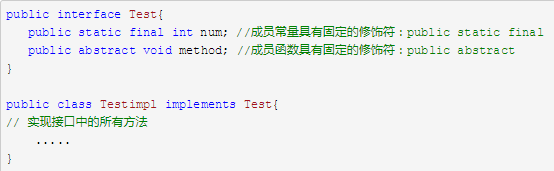


# Java基础

## 接口

定义和实现：



接口中固定的修饰符可以省略

## 接口回调

接口回调：

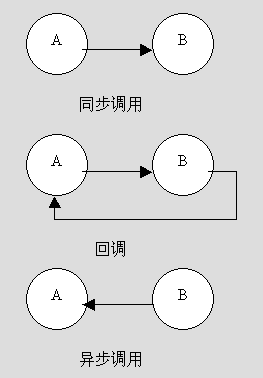
定义：把实现了某一接口（或抽象类）的类所创建的对象的引用 赋予 该接口声明的变量，通过该接口变量 调用 该实现类对象的实现的接口方法

作用：比如A类中有B类，B类中又要更新A类信息的时候就有用了，或者说异步的时候，如果在某个事件发生时得到通知，就可以定义一个接口。

分类：同步回调、回调、异步回调

1. 同步回调：是一种阻塞式调用，调用方法要等待对方执行完毕后才返回，它是一种单向调用
2. 回调：是一种双向调用模式，被调用方在接口被调用时也会调用对方的接口
3. 异步回调：是一种类是于消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种信息或发生某种时间时，会主动通知客户方（即调用客户方的接口）。

回调和异步调用的关系非常紧密，通常使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。Java中经常使用的是回调



比较经典的使用回调的方式：

* Class A 中实现接口InA-----背景1
* Class A中包含一个Class B的引用b----背景2
* Class B中有一个参数为InA的方法Test(InA a)----背景3
* A的对象a调用B的方法传入自己，Test(a)，注册，------这一步相当于you call me
* 然后b就可以在Test方法中调用InA的方法-----这一步相当于I call you back

打电话的例子：异步加回调的方式

有一天小王遇到一个问题，问题是“1+1=？”，就打电话问小李，小李一下子也不知道，就跟小王说，等我办完手上的事情，就去想想答案，小王也不会傻傻的拿着电话去等小李的答案，于是小王就对小李说，你知道答案之后就打我电话告诉我，于是挂了电话，办自己的事情去了，过了一个小时，小李打了小王的电话，告诉他答案是2

实例参考：[接口回调实例](http://blog.csdn.net/xiaanming/article/details/8703708/)

//接口声明的变量

1、Com com;

//实现了Com接口的类（Com1）所创建的对象的引用 赋予给 该接口声明的变量

2、Com com = new Com1；

//通过该接口变量（com）调用 该实现类对象（Com1）的实现的接口方法（carson()）

3、Com.carson();

代码：

1、

public interface Com {  
 void dosth();  
}

2、

Com com = new A();

3、

com.dosth();

类实现了接口或抽象类的类A：

class A implements Com{  
  
 @Override  
 public void dosth() {  
  
 }  
}

第2、3步可以用匿名内部类的方式使用：

new Com(){  
  
 @Override  
 public void dosth() {  
  
 }  
};

## 序列化Parcelable

定义：将对象的状态信息转换为可以存储或传输的形式的过程

作用：

1. 永久性保存对象，保存对象的字节序列到本地文件中
2. 通过序列话对象在网络中传递对象
3. 通过序列化在进程间传递对象

应用场景：需要在多个部件（activity或service）之间通过Intent传递数据，简单类型（如数字、字符串）的可以直接放入Intent。复杂类型的必须实现parcelable接口

## UML类图

SimpleURL：UML自动生成的插件，可以在Android Studio中方便画图。

# GitHub上Android开源项目分类汇总

简书：

[GitHub开源项目分类汇总](http://www.jianshu.com/p/4b307a94ee79)

# 优秀博客主

简书：[Carson\_Ho](http://www.jianshu.com/u/383970bef0a0)

CSDN：[郭霖](http://my.csdn.net/sinyu890807)、[鸿翔](http://my.csdn.net/lmj623565791)、[赵凯强](http://my.csdn.net/bz419927089)

[stormzhang](http://stormzhang.com/posts/)

# Android Studio关联源码方法

[Android Studio 如何关联源码（关联想要的任何源码）](http://blog.csdn.net/bruceyangjie/article/details/52366661)

# Android基础

## 组件

### 1.1.1 Activity

#### 1.1.1.1 生命周期



### 1.1.2 Service

博客：[Android四大组件：service服务史上最全面解析](http://www.jianshu.com/p/d963c55c3ab9)

Service生命周期：<http://www.jianshu.com/p/8d0cde35eb10>

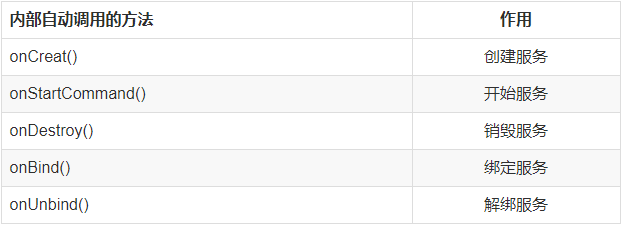
#### 1.1.2.1 Service生命周期

Service生命周期常用方法：

* 4个手动调用方法



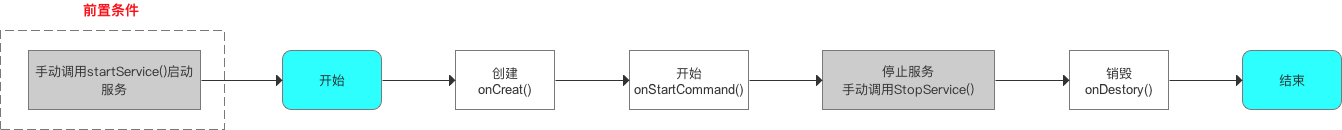
* 5个内部自动调用的方法



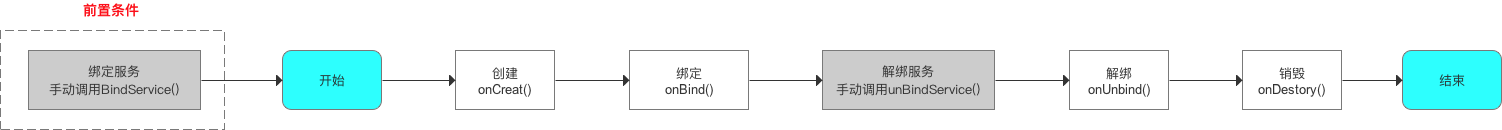
onStartCommand()方法中调用了onStart()方法

生命周期方法具体介绍：

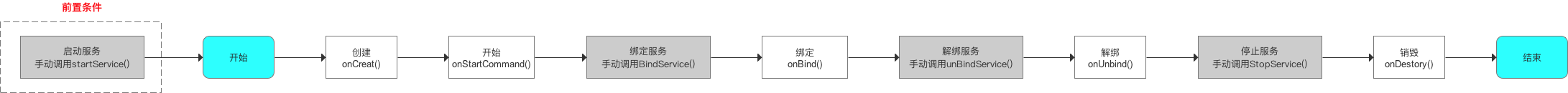
* 只使用startService()启动服务的生命周期



* 只使用BindingService()启动的生命周期



* 同时使用StartService()和BindService()绑定的生命周期



特别注意:

* startService()和stopService()只能开启和关闭service，不能操作service
* bindService()和unbindService()可以操作service
* startService开启的服务，调用者退出后Service仍然存在
* BindService开启的服务，调用这退出后service随着调用者一起被销毁

### 1.1.3 BroadcastReceiver

博客：[Android四大组件：BroadcastReceiver全面解析](http://www.jianshu.com/p/ca3d87a4cdf3)

#### 1.1.3.1 BroadcastReceiver生命周期

### 1.1.4 ContentProvider

优秀博客：基本涵盖Contentprovider所有知识<http://www.jianshu.com/p/ea8bc4aaf057>

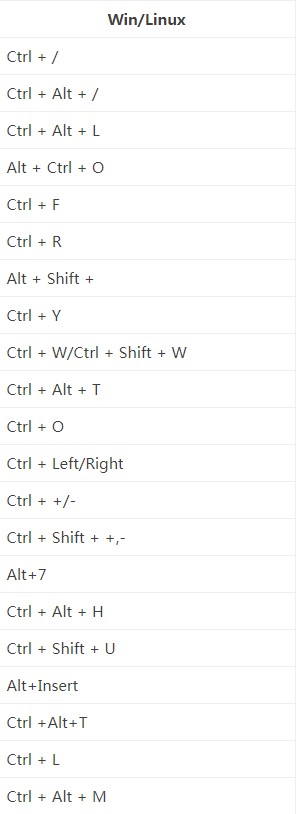
### 1.1.5 Fragment

基础用法：<http://www.jianshu.com/p/2bf21cefb763>

Activity和Fragment之间的通信：<http://www.jianshu.com/p/825eb1f98c19>

## 四大存储方式

## 1.3 Android studio快捷键



快速跳到下一行：shift+enter

## Context理解

Context是一个抽象类，源码解释：

*/\*\*  
 \* Interface to global information about an application environment. This is  
 \* an abstract class whose implementation is provided by  
 \* the Android system. It  
 \* allows access to application-specific resources and classes, as well as  
 \* up-calls for application-level operations such as launching activities,  
 \* broadcasting and receiving intents, etc.  
 \*/*

翻译：是应用程序环境的全局信息接口，是一个抽象类，它的实现是由Android系统提供，允许访问特定应用程序的资源和类（即是跨应用资源访问，获取已安装应用的context），以及用于应用程序级操作（启动活动、广播、接收intent）的回调。

### Context类中的抽象方法

以下方法的具体实现，要看其实现类中是怎么实现的

#### getAssets()

*/\*\*  
 \* Returns an AssetManager instance for the application's package.  
 \* <p>  
 \* <strong>Note:</strong> Implementations of this method should return  
 \* an AssetManager instance that is consistent with the Resources instance  
 \* returned by {****@link*** *#getResources()}. For example, they should share the  
 \* same {****@link*** *Configuration} object.  
 \*  
 \** ***@return*** *an AssetManager instance for the application's package  
 \** ***@see*** *#getResources()  
 \*/*public abstract AssetManager getAssets();

返回一个应用程序包的AssetManager实例

#### getResource()

*/\*\*  
 \* Returns a Resources instance for the application's package.  
 \* <p>  
 \* <strong>Note:</strong> Implementations of this method should return  
 \* a Resources instance that is consistent with the AssetManager instance  
 \* returned by {****@link*** *#getAssets()}. For example, they should share the  
 \* same {****@link*** *Configuration} object.  
 \*  
 \** ***@return*** *a Resources instance for the application's package  
 \** ***@see*** *#getAssets()  
 \*/*public abstract Resources getResources();

返回一个应用程序包的Resources实例

#### getPackageManager()

*/\*\* Return PackageManager instance to find global package information. \*/*public abstract PackageManager getPackageManager();

返回一个找到全局包信息的PackageManager实例

#### getContentResolver()

*/\*\* Return a ContentResolver instance for your application's package. \*/*public abstract ContentResolver getContentResolver();

返回一个应用程序包的ContentResolver实例

#### getMainLooper()

*/\*\*  
 \* Return the Looper for the main thread of the current process. This is  
 \* the thread used to dispatch calls to application components (activities,  
 \* services, etc).  
 \* <p>  
 \* By definition, this method returns the same result as would be obtained  
 \* by calling {****@link*** *Looper#getMainLooper() Looper.getMainLooper()}.  
 \* </p>  
 \*  
 \** ***@return*** *The main looper.  
 \*/*public abstract Looper getMainLooper();

为当前进程的主线程返回Looper，主线程是用于调度应用程序组件（activities，services等）的线程

#### getApplicationContext()

*/\*\*  
 \* Return the context of the single, global Application object of the  
 \* current process. This generally should only be used if you need a  
 \* Context whose lifecycle is separate from the current context, that is  
 \* tied to the lifetime of the process rather than the current component.  
 \*  
 \* <p>Consider for example how this interacts with  
 \* {****@link*** *#registerReceiver(BroadcastReceiver, IntentFilter)}:  
 \* <ul>  
 \* <li> <p>If used from an Activity context, the receiver is being registered  
 \* within that activity. This means that you are expected to unregister  
 \* before the activity is done being destroyed; in fact if you do not do  
 \* so, the framework will clean up your leaked registration as it removes  
 \* the activity and log an error. Thus, if you use the Activity context  
 \* to register a receiver that is static (global to the process, not  
 \* associated with an Activity instance) then that registration will be  
 \* removed on you at whatever point the activity you used is destroyed.  
 \* <li> <p>If used from the Context returned here, the receiver is being  
 \* registered with the global state associated with your application. Thus  
 \* it will never be unregistered for you. This is necessary if the receiver  
 \* is associated with static data, not a particular component. However  
 \* using the ApplicationContext elsewhere can easily lead to serious leaks  
 \* if you forget to unregister, unbind, etc.  
 \* </ul>  
 \*/*public abstract Context getApplicationContext();

返回当前进程的单个、全局Application对象的上下文，通常情况只能在需要一个上下文的情况下使用，该生命周期与当前上下文的生命周期是分开的。这个全局对象的上下文和进程的生命周期相关，而不是当前组件的生命周期。

#### getClassLoader()

*/\*\*  
 \* Return a class loader you can use to retrieve classes in this package.  
 \*/*public abstract ClassLoader getClassLoader();

返回一个可以用来取回此包中的类的类加载器

#### getPackagetName()

*/\*\* Return the name of this application's package. \*/*public abstract String getPackageName();

返回应用程序包的包名

#### getApplicationInfo()

*/\*\* Return the full application info for this context's package. \*/*public abstract ApplicationInfo getApplicationInfo();

返回这个context包的完整的应用程序信息

#### getPackageResourcePath()

*/\*\*  
 \* Return the full path to this context's primary Android package.  
 \* The Android package is a ZIP file which contains the application's  
 \* primary resources.  
 \*  
 \* <p>Note: this is not generally useful for applications, since they should  
 \* not be directly accessing the file system.  
 \*  
 \** ***@return*** *String Path to the resources.  
 \*/*public abstract String getPackageResourcePath();

返回这个Android应用包的完整路径，这个Android包是一个包含了应用resources的一个zip压缩文件。其实就是APK的路径。

#### getPackageCodePath()

*/\*\*  
 \* Return the full path to this context's primary Android package.  
 \* The Android package is a ZIP file which contains application's  
 \* primary code and assets.  
 \*  
 \* <p>Note: this is not generally useful for applications, since they should  
 \* not be directly accessing the file system.  
 \*  
 \** ***@return*** *String Path to the code and assets.  
 \*/*public abstract String getPackageCodePath();

同上，不过这个zip压缩包包含的是这个应用主要的code和assets，这里要明白assets和resources的区别，详见1.6.1 apk文件的生成

#### getSystemService()

*/\*\*  
 \* Return the handle to a system-level service by name. The class of the  
 \* returned object varies by the requested name. Currently available names  
 \* are:  
 \*  
 \* <dl>  
 \* <dt> {****@link*** *#WINDOW\_SERVICE} ("window")  
 \* <dd> The top-level window manager in which you can place custom  
 \* windows. The returned object is a {****@link*** *android.view.WindowManager}.  
 \* <dt> {****@link*** *#LAYOUT\_INFLATER\_SERVICE} ("layout\_inflater")  
 \* <dd> A {****@link*** *android.view.LayoutInflater} for inflating layout resources  
 \* in this context.  
 \* <dt> {****@link*** *#ACTIVITY\_SERVICE} ("activity")  
 \* <dd> A {****@link*** *android.app.ActivityManager} for interacting with the  
 \* global activity state of the system.  
 \* <dt> {****@link*** *#POWER\_SERVICE} ("power")  
 \* <dd> A {****@link*** *android.os.PowerManager} for controlling power  
 \* management.  
 \* <dt> {****@link*** *#ALARM\_SERVICE} ("alarm")  
 \* <dd> A {****@link*** *android.app.AlarmManager} for receiving intents at the  
 \* time of your choosing.  
 \* <dt> {****@link*** *#NOTIFICATION\_SERVICE} ("notification")  
 \* <dd> A {****@link*** *android.app.NotificationManager} for informing the user  
 \* of background events.  
 \* <dt> {****@link*** *#KEYGUARD\_SERVICE} ("keyguard")  
 \* <dd> A {****@link*** *android.app.KeyguardManager} for controlling keyguard.  
 \* <dt> {****@link*** *#LOCATION\_SERVICE} ("location")  
 \* <dd> A {****@link*** *android.location.LocationManager} for controlling location  
 \* (e.g., GPS) updates.  
 \* <dt> {****@link*** *#SEARCH\_SERVICE} ("search")  
 \* <dd> A {****@link*** *android.app.SearchManager} for handling search.  
 \* <dt> {****@link*** *#VIBRATOR\_SERVICE} ("vibrator")  
 \* <dd> A {****@link*** *android.os.Vibrator} for interacting with the vibrator  
 \* hardware.  
 \* <dt> {****@link*** *#CONNECTIVITY\_SERVICE} ("connection")  
 \* <dd> A {****@link*** *android.net.ConnectivityManager ConnectivityManager} for  
 \* handling management of network connections.  
 \* <dt> {****@link*** *#WIFI\_SERVICE} ("wifi")  
 \* <dd> A {****@link*** *android.net.wifi.WifiManager WifiManager} for management of Wi-Fi  
 \* connectivity. On releases before NYC, it should only be obtained from an application  
 \* context, and not from any other derived context to avoid memory leaks within the calling  
 \* process.  
 \* <dt> {****@link*** *#WIFI\_P2P\_SERVICE} ("wifip2p")  
 \* <dd> A {****@link*** *android.net.wifi.p2p.WifiP2pManager WifiP2pManager} for management of  
 \* Wi-Fi Direct connectivity.  
 \* <dt> {****@link*** *#INPUT\_METHOD\_SERVICE} ("input\_method")  
 \* <dd> An {****@link*** *android.view.inputmethod.InputMethodManager InputMethodManager}  
 \* for management of input methods.  
 \* <dt> {****@link*** *#UI\_MODE\_SERVICE} ("uimode")  
 \* <dd> An {****@link*** *android.app.UiModeManager} for controlling UI modes.  
 \* <dt> {****@link*** *#DOWNLOAD\_SERVICE} ("download")  
 \* <dd> A {****@link*** *android.app.DownloadManager} for requesting HTTP downloads  
 \* <dt> {****@link*** *#BATTERY\_SERVICE} ("batterymanager")  
 \* <dd> A {****@link*** *android.os.BatteryManager} for managing battery state  
 \* <dt> {****@link*** *#JOB\_SCHEDULER\_SERVICE} ("taskmanager")  
 \* <dd> A {****@link*** *android.app.job.JobScheduler} for managing scheduled tasks  
 \* <dt> {****@link*** *#NETWORK\_STATS\_SERVICE} ("netstats")  
 \* <dd> A {****@link*** *android.app.usage.NetworkStatsManager NetworkStatsManager} for querying network  
 \* usage statistics.  
 \* <dt> {****@link*** *#HARDWARE\_PROPERTIES\_SERVICE} ("hardware\_properties")  
 \* <dd> A {****@link*** *android.os.HardwarePropertiesManager} for accessing hardware properties.  
 \* </dl>  
 \*  
 \* <p>Note: System services obtained via this API may be closely associated with  
 \* the Context in which they are obtained from. In general, do not share the  
 \* service objects between various different contexts (Activities, Applications,  
 \* Services, Providers, etc.)  
 \*  
 \** ***@param*** *name The name of the desired service.  
 \*  
 \** ***@return*** *The service or null if the name does not exist.  
 \*  
 \** ***@see*** *#WINDOW\_SERVICE  
 \** ***@see*** *android.view.WindowManager  
 \** ***@see*** *#LAYOUT\_INFLATER\_SERVICE  
 \** ***@see*** *android.view.LayoutInflater  
 \** ***@see*** *#ACTIVITY\_SERVICE  
 \** ***@see*** *android.app.ActivityManager  
 \** ***@see*** *#POWER\_SERVICE  
 \** ***@see*** *android.os.PowerManager  
 \** ***@see*** *#ALARM\_SERVICE  
 \** ***@see*** *android.app.AlarmManager  
 \** ***@see*** *#NOTIFICATION\_SERVICE  
 \** ***@see*** *android.app.NotificationManager  
 \** ***@see*** *#KEYGUARD\_SERVICE  
 \** ***@see*** *android.app.KeyguardManager  
 \** ***@see*** *#LOCATION\_SERVICE  
 \** ***@see*** *android.location.LocationManager  
 \** ***@see*** *#SEARCH\_SERVICE  
 \** ***@see*** *android.app.SearchManager  
 \** ***@see*** *#SENSOR\_SERVICE  
 \** ***@see*** *android.hardware.SensorManager  
 \** ***@see*** *#STORAGE\_SERVICE  
 \** ***@see*** *android.os.storage.StorageManager  
 \** ***@see*** *#VIBRATOR\_SERVICE  
 \** ***@see*** *android.os.Vibrator  
 \** ***@see*** *#CONNECTIVITY\_SERVICE  
 \** ***@see*** *android.net.ConnectivityManager  
 \** ***@see*** *#WIFI\_SERVICE  
 \** ***@see*** *android.net.wifi.WifiManager  
 \** ***@see*** *#AUDIO\_SERVICE  
 \** ***@see*** *android.media.AudioManager  
 \** ***@see*** *#MEDIA\_ROUTER\_SERVICE  
 \** ***@see*** *android.media.MediaRouter  
 \** ***@see*** *#TELEPHONY\_SERVICE  
 \** ***@see*** *android.telephony.TelephonyManager  
 \** ***@see*** *#TELEPHONY\_SUBSCRIPTION\_SERVICE  
 \** ***@see*** *android.telephony.SubscriptionManager  
 \** ***@see*** *#CARRIER\_CONFIG\_SERVICE  
 \** ***@see*** *android.telephony.CarrierConfigManager  
 \** ***@see*** *#INPUT\_METHOD\_SERVICE  
 \** ***@see*** *android.view.inputmethod.InputMethodManager  
 \** ***@see*** *#UI\_MODE\_SERVICE  
 \** ***@see*** *android.app.UiModeManager  
 \** ***@see*** *#DOWNLOAD\_SERVICE  
 \** ***@see*** *android.app.DownloadManager  
 \** ***@see*** *#BATTERY\_SERVICE  
 \** ***@see*** *android.os.BatteryManager  
 \** ***@see*** *#JOB\_SCHEDULER\_SERVICE  
 \** ***@see*** *android.app.job.JobScheduler  
 \** ***@see*** *#NETWORK\_STATS\_SERVICE  
 \** ***@see*** *android.app.usage.NetworkStatsManager  
 \** ***@see*** *android.os.HardwarePropertiesManager  
 \** ***@see*** *#HARDWARE\_PROPERTIES\_SERVICE  
 \*/*public abstract Object getSystemService(@ServiceName @NonNull String name);

通过名称返回系统级服务的句柄，返回对象的类根据请求的名称而异，目前能请求到的系统服务有以上几种。

## Android应用从项目源码到启动过程

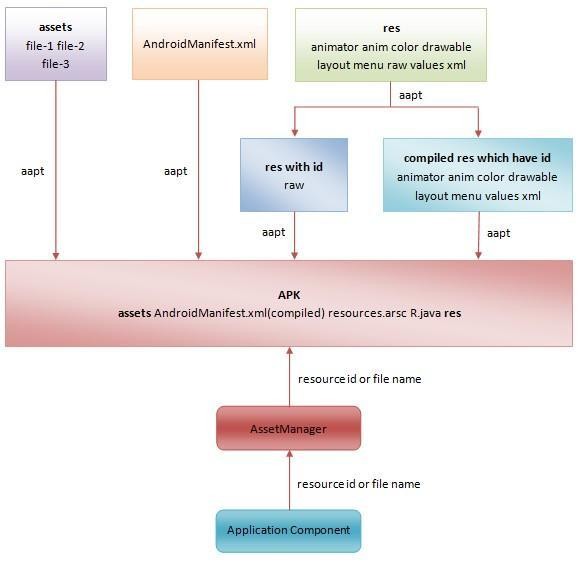
### 1.6.1 apk文件的生成

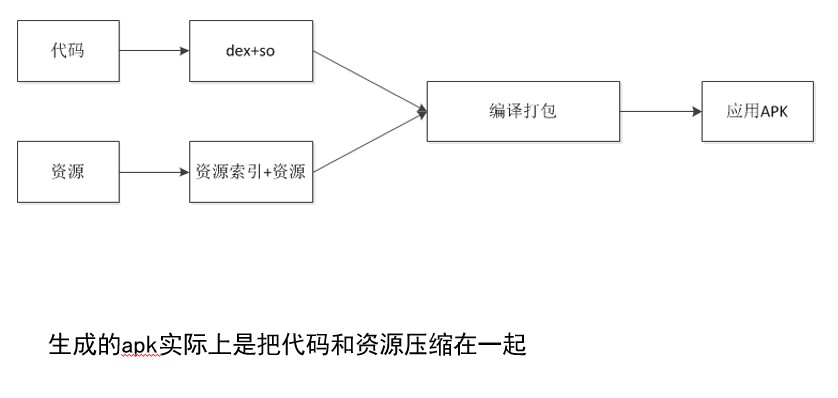


资源打包的过程：

aapt是Android asset packaging tool（Android资源打包工具）

开发APP需要代码和资源。 下图是资源打包过程和查找图：





除了assets资源和res/raw资源被原封不动的打包进apk之外，其他资源都会被编译或者处理，.xml文件会被编译为二进制的xml,解压apk之后无法打开这些xml文件。

除了assets资源外，其他资源都会被赋予一个资源ID

aapt负责编译和打包资源，编译完成后，会生成一个resources.arsc文件和一个R.java，前者是保存一个资源索引库，后者定义了一个资源的ID常量，供代码中索引资源，AndroidManifest.xml文件也会被编译成二进制的xml文件，然后打包到apk中

应用程序在运行时最终通过AssetManager来访问资源，或通过资源ID来访问，或者通过文件名来访问。

res目录和assets目录的区别：

1、引用资源的方式不同

res目录中的所有资源都会在R类的相应子类中生成对应的int类型变量，需要使用这些变量来引用资源。例如res/layout/main.xml是布局文件资源，会在R.layout类中生成一个名为main的变量，所以需要使用R.layout.main来引用main.xml。assets目录中的资源需要直接使用文件名来引用，例如assets/xyz.xml、assets/abc/test.db是assets目录中的两个资源文件，其中abc是assets中的子目录。可以使用如下的代码引用这两个资源文件，并返回与资源文件对应的InputStream对象，剩下的操作就是普通的Java代码输入流操作了。InputStream is1=getResources().getAssets().open("xyz.xml")；InputStream is2=getResources().getAssets().open("abc/test.db");

2、处理方式不同

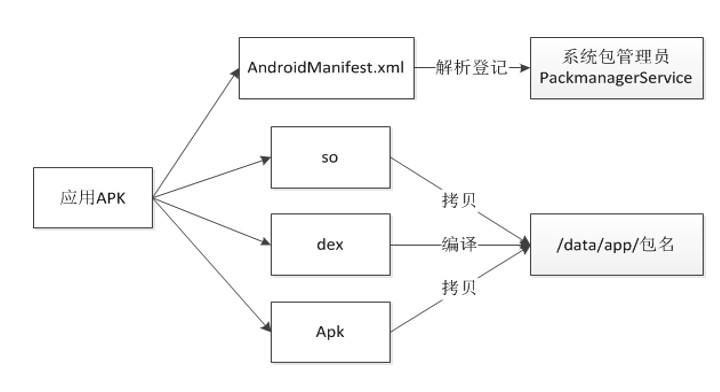
res目录中的资源子目录除了raw外，其他资源目录中的资源文件都会被编译，这也是为什么 将APK文件解压后无法直接查看XML格式资源文件内容的原因。而assets与res/raw目录中的资源文件不会做任何处理，所以将APK解压后，这两个目录中的资源文件都会保持原样.

3、子目录不同

res目录只能有一层子目录，而且这些子目录必须是预定义的，如res/layout、res/values等都是合法的，而res/abc,res/xyz并不是合法的资源目录，在assets目录中可以建任意层次的子目录(只受操作系统的限制)

### 1.6.2 apk文件的安装

安装apk主要包含解析Androidmanifest.xml文件，拷贝，编译dex几个主要步骤：

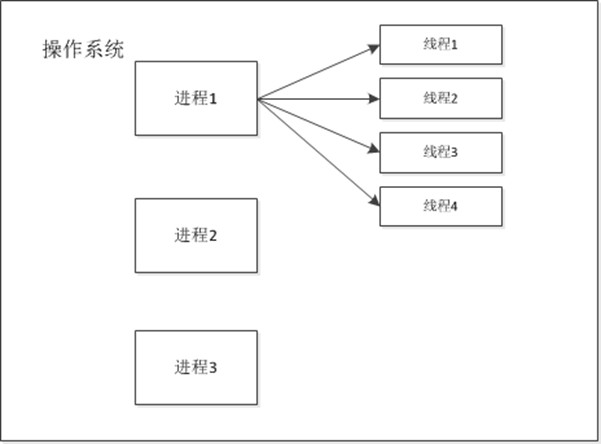


安装完成后base.apk是原始的apk，主要用于加载资源，或者解释执行；lib是apk中所有的C/C++生成的so文件；oat是dex文件(Java代码)编译优化后的虚拟机执行文件

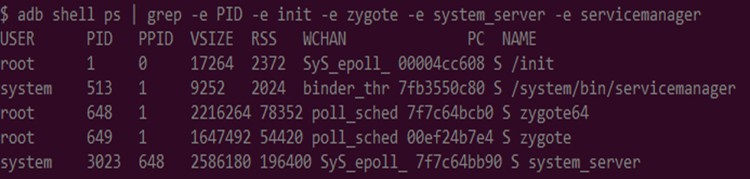
### 1.6.3应用的启动

进程：具有一定独立功能的程序，是系统进行资源分配和调度的一个独立单位，重点是系统调度和单独的单位，也就是说进程是可以独立运行的一段程序，大多数情况下，APP会运行在一个独立的Linux进程中。

线程：是进程的一个实体，是CPU调度和分配的最小单位，是比进程更小的能独立运行的基本单位，线程自己基本上不拥有系统资源。默认情况下APP一开始都会运行在同一个进程中的同一个线程，也就是主线程或UI线程。



/init:开机以后的第一个进程，负责解析init.rc脚本，启动守护进程，创建zygote进程



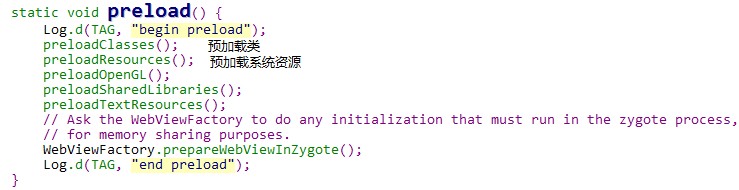
PID是该进程的ID，PPID是父进程的ID，UID是用户ID

zygote/zygote64:所有Android应用进程的父进程

/system/bin/servicemanager:binder通信机制守护进程，总管进程

system\_service:Android系统核心进程，负责创建核心的系统service，包括AMS（Activity Manager Service）/PMS(Package Manager Service)/WMS(Window Manager Service)

zygote进程：所有应用进程的父进程



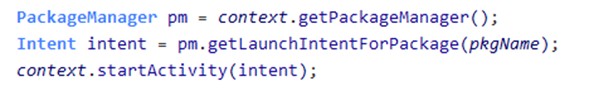
1、预加载Android提供的class：解决编译时框架依赖的原理

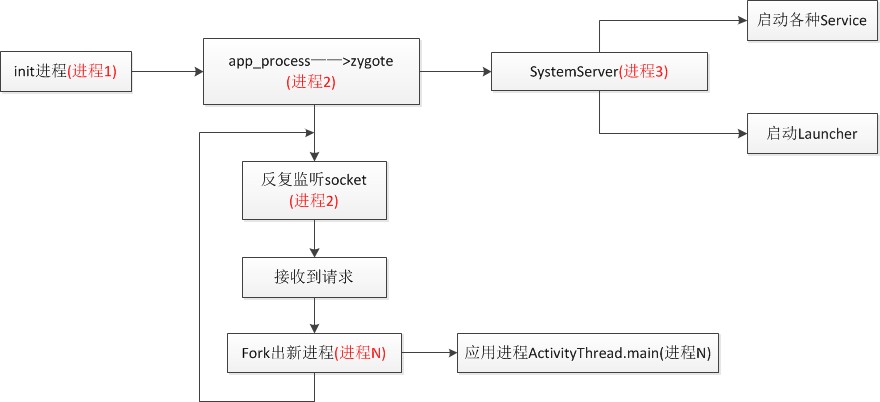
2、预加载系统资源：应用进程中直接访问系统资源的出处

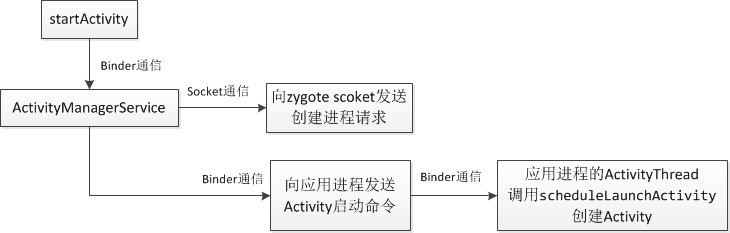
3、死循环监听socket，等待AMS的fork请求（即使请求新建一个进程）

应用进程fork自zygote进程，进入到子进程以后，会将预加载的类和资源一起带到子进程

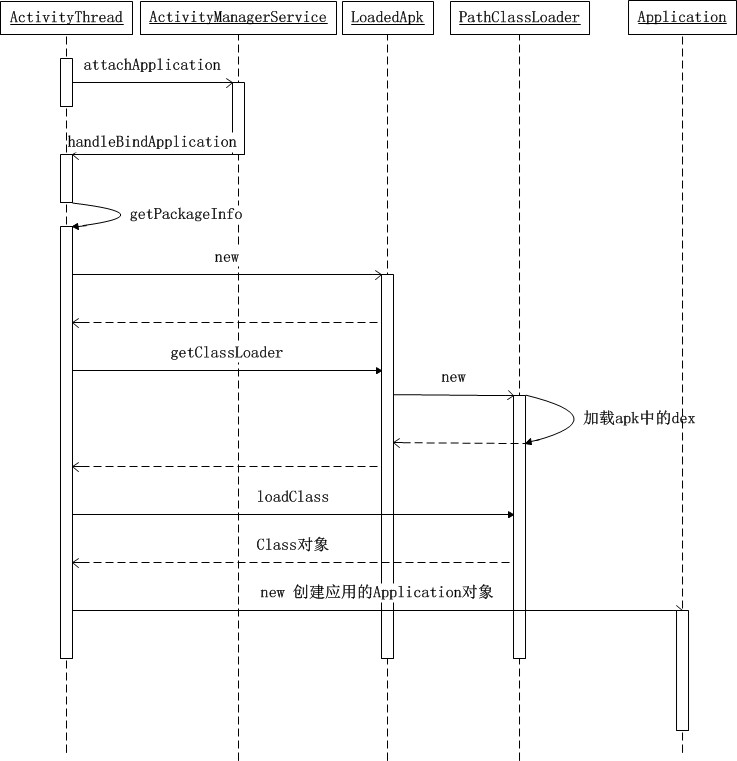
应用启动代码：







进程的启动：是一个通用的步骤，所有的应用在能开始运行之前，都需要经历一次zygote进程fork的过程，fork完成之后该进程和AMS之间没有任何联系，系统无法管理该应用进程，因此进程创建结束以后，应用进程需要主动和AMS建立联系，将其纳入AMS的管理范围。



应用启动过程总结：

1、AMS向zygote进程发起fork命令

2、zygote进程fork进程

3、进程主线程以ActivityThread.main作为入口开始应用进程的初始化，创建主线程的消息死循环，主线程一切皆是消息

4、应用进程向AMS尽心注册，加入AMS管理

5、应用进程加载odex（代码）和apk（资源）

6、进入应用定义的Application

7、启动Activity

大多数情况下一个APP运行在一个进程中，除非在AndroidManifest.xml文件中配置Android：process属性，或者通过native代码fork进程

应用的主线程：一切皆是消息，应用存活的基础（死循环）



## 1.6 常见Bug

## 1.7 UI

### 1.7.1 ListView

使用ListView主要包括三个方面：ListView布局，数据，数据和布局之间的适配（adapter），但是ListView运行效率很低，因为在adapter中执行getView()方法时，每次都会将布局重新加载一边，getView()方法是在每个子项滚动到屏幕内的时候会被调用。当ListView滚动过快的时候，就会造成性能的瓶颈，解决方法：

1. 使用convertView对加载好的布局进行缓存
2. 使用viewHolder对布局中所有的控件实例进行缓存

Android中提供了很多适配器的实现类，最好用的是ArrayAdapter，因为它可以通过泛

来指定要适配的数据类型

### 1.7.2 RecycleView

ListView缺点：性能较低，需要手动优化；不能横向滚动

RecycleView是新增的控件，定义在了support包中，需要添加依赖

使用方法：

1. 在build.gradle中添加依赖
2. 同步，Sync Now

### 1.7.3 Android布局的两种方式

Xml和Java代码相结合好处：代码的解耦，图形化布局界面，布局后立即得到反馈

纯Java代码实现好处：运行的过程中动态的改变界面，xml布局是静态的。

#### 1.7.3.1 xml和Java代码结合

#### 1.7.3.2 纯Java代码实现

Java代码实现布局时，方法传入的参数时px，需要进行提前转化。

### 1.7.4 .9图

Android studio已经集成了.9图的功能，制作.9图可以使用Android studio在线制作

.9图中上边和左边表示图片能够拉伸的区域，下边和右边表示图片中内容显示区域

.9图的制作的作用：

1. 确定图片中哪些地方可以拉伸，应用于消息框等
2. 确定图片中哪些地方不可以拉伸（需要多根线段设置），应用于背景图片中某一部分不拉伸

### 1.7.5 动画和插值器、估值器

优秀博客：基本已经把插值器和估值器讲清楚，下面只是重点部分总结

自定义插值器和估值器：<http://blog.csdn.net/carson_ho/article/details/72863901>

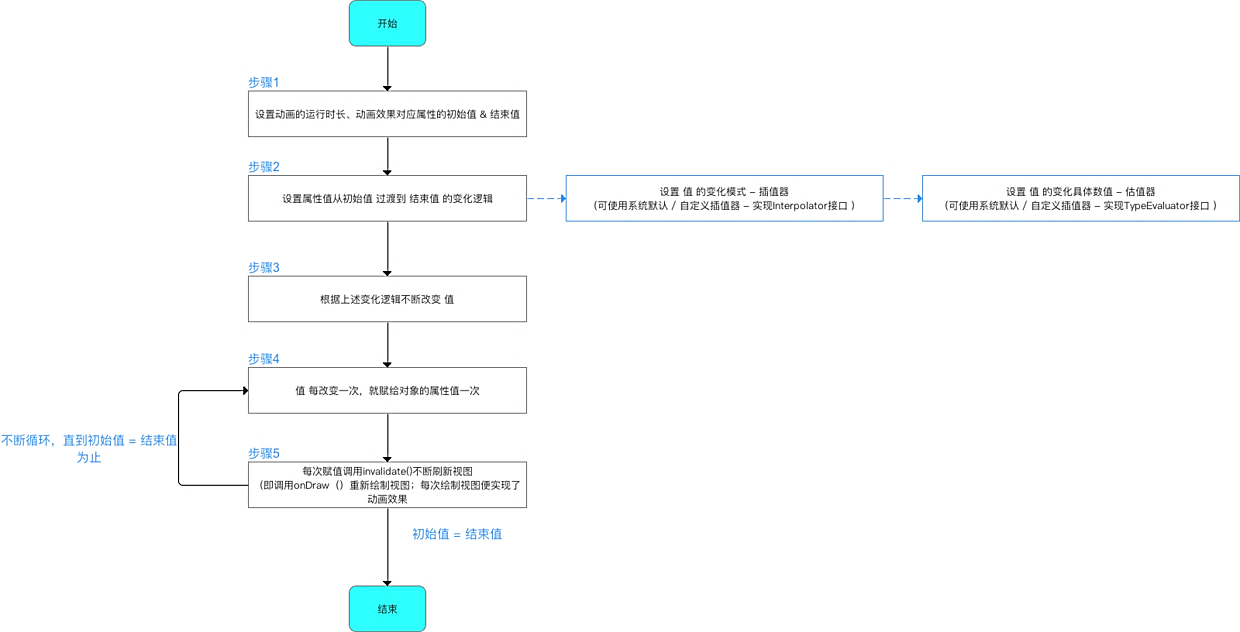
基础知识：

补间、帧、属性动画对比：<http://blog.csdn.net/nanxuan521/article/details/52620147>

Animation、Animator区别：<http://blog.csdn.net/cp25807720/article/details/21833221>

属性动画基础知识：<http://blog.csdn.net/guolin_blog/article/details/43536355>

Android补间动画和属性动画实现原理图：



#### 1.7.5.1 插值器及自定义插值器

##### 1.7.5.1.1 插值器

定义：一个接口

public interface Interpolator extends TimeInterpolator {}

public interface TimeInterpolator {  
 float getInterpolation(float var1);  
}

作用：设置属性值从初始值过渡到结束值的变化规律，具体属性值由估值器得到

应用场景：实现非线性运动的动画效果

两种使用方式：xml或者Java代码中设置，xml中设置插值器时只需要传入对应的插值器资源ID即可，Android系统内置了9种常用插值器；Java代码中设置插值器时，只需创建对应的插值器对象即可，系统有默认的插值器对象（先加速后减速）。

##### 1.7.5.1.2 自定义插值器

本质：根据动画的进度（0%-100%）计算出当前属性值改变的百分比

具体使用：自定义插值器需要实现Interpolator/TimeInterpolator接口&复写getInterpolator()方法

1. 补间动画实现Interpolator接口；属性动画实现TimeInterpolator接口
2. TimeInterpolator接口是属性动画中新增的，用于兼容Interpolator接口，这使得所有过去的Interpolator实现类可以直接在属性动画中使用

#### 1.7.5.2 估值器及自定义估值器

##### 1.7.5.2.1 估值器

定义：一个接口

作用：设置属性值从初始值过渡到结束值变化的具体数值

1. 插值器决定值的变化规律，即是决定变化趋势；具体变化数值交给估值器

2、估值器是属性动画特有的属性

##### 1.7.5.2.2 自定义估值器

本质：根据插值器计算出当前属性值改变的百分比以及初始值、结束值来计算当前属性具体的数值

如：动画进行了50%（初始值=100，结束值=200 ），那么匀速插值器计算出了当前属性值改变的百分比是50%，那么估值器则负责计算当前属性值 = 100 + （200-100）x50% = 150.

具体使用：自定义估值器需要实现TypeEvaluetor()接口&覆写evaluate()方法

#### 1.7.5.3 Android属性动画

优秀博客：[Android属性动画：这是一篇很详细的属性动画 总结&攻略](http://www.jianshu.com/p/2412d00a0ce4)

### 1.7.6 view绘制流程&自定义view

View的绘制顺序和过程有三个：measure、layout、draw

作用：

Measure：测量view的宽和高

Layout：计算view视图的位置

Draw：绘制view视图

#### 1.7.6.1 measure测量过程详解

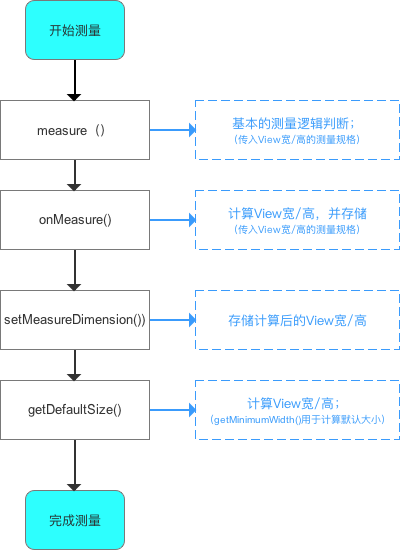
Measure过程根据类型分为两种：

1. 单一view：只测量自身一个view
2. ViewGroup时：对ViewGroup视图中所有的子view都进行测量（即是遍历所有子view的measure方法，然后子view再递归执行这个过程）

##### 1.7.6.1.1 单一view测量流程

应用场景：没有现成的view，需要自己实现，如制作一个支持网络加图片的ImageView

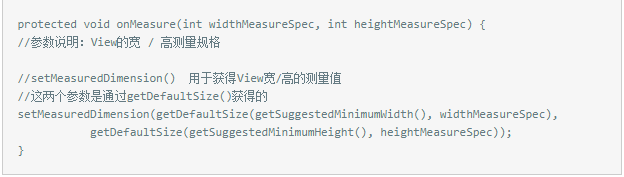
测量流程：



measure源码分析：作用是基本测量逻辑的判断；调用onMeasure方法



onMeaure源码方法：作用是调用getDefaultSize定义对View尺寸的测量逻辑；调用setMeasuredDimension存储测量后的宽和高，是重写onMeasure方法的最终目的

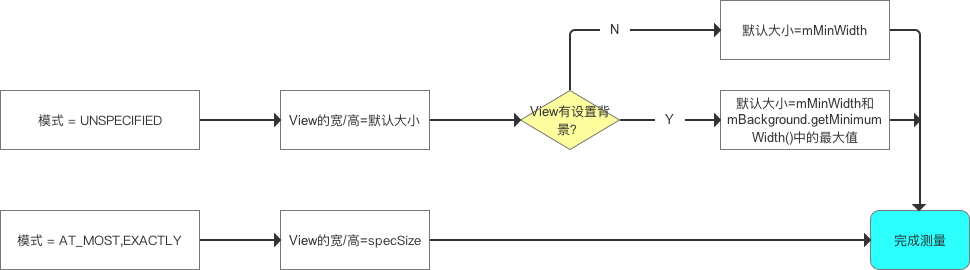


SetMeasuredDimensio有两个参数，指两个参数是由getDefaultSize获得的，getDefaultSize作用是根据view的测量规格计算出View的宽和高

getDefalutSize源码：



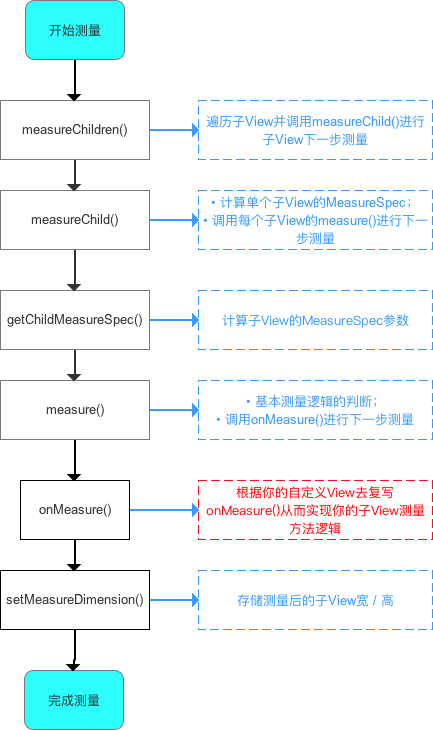
计算逻辑如下：



##### 1.7.6.1.2 viewGroup的测量流程：

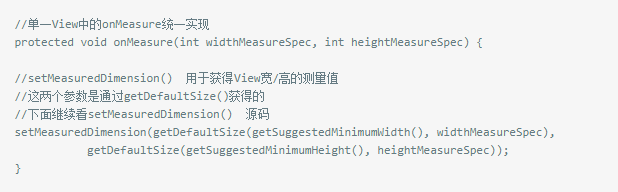
应用场景：自定义viewGroup一般是利用现有的组件根据特定的布局方式来组成新的组件，大多数继承自viewGroup或各种Layout

原理：遍历所有子view进行子view的测量，将所有子view的尺寸进行合并，最终得到viewGroup的测量值



大致流程和单一view是一样的，区别是单一view的measure过程对onMeasure做了统一的代码实现，

注：在单一view measure的过程中，getDefaultSize只是简单的测量了宽高值，在实际使用时有时需要更精确的测量，有时候也需要重写onMeasure方法。



但是自定义viewGroup需要重写onMeasure方法。原因：

1. viewGroup是抽象类，自身并没有重写view的onMeasure方法：
2. 因为不同的viewGroup子类（LinearLayout、RealtiveLayout或自定义ViewGroup子类等）具有不同的布局特性，这导致他们的子view的测量方法各有不同，onMeasure作用是测量view的宽和高，因此viewGroup无法对onMeasure作同意的实现

所以在自定义view中，关键在于复写onMeasure方法

复写套路：

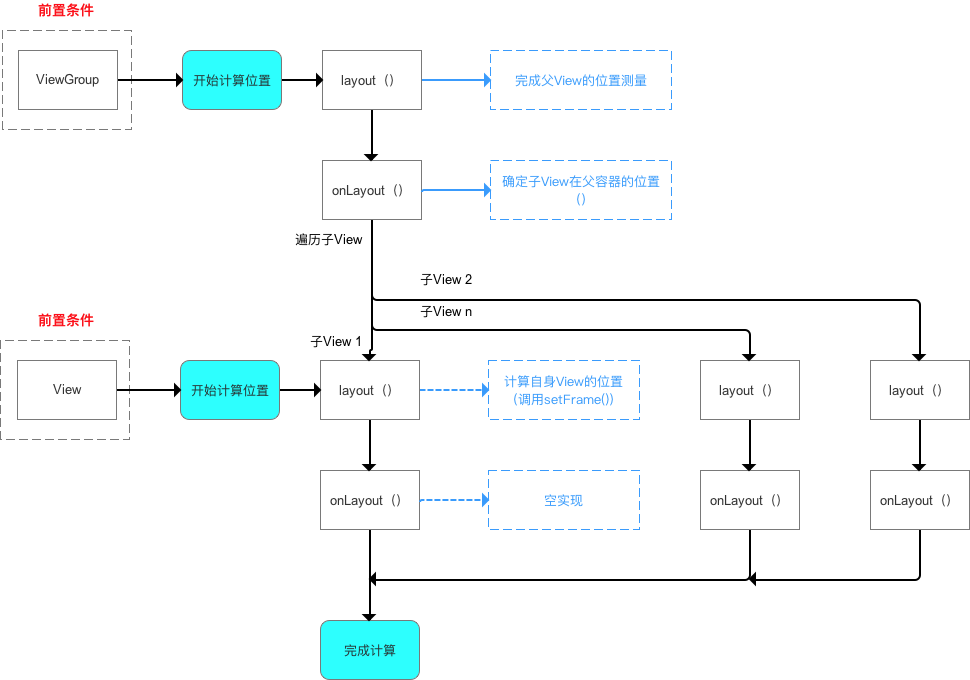


#### 1.7.6.2 layout过程详解

作用：计算view视图的位置（计算view四个顶点的位置：left、top、right、bottom）

分为两种情况：单一view和viewGroup的Layout流程

1. 单一view：计算本身View的位置
2. viewGroup：除了计算本身view的位置外，还需确定子view在父容器中的位置（递归计算每一个子视图的位置）



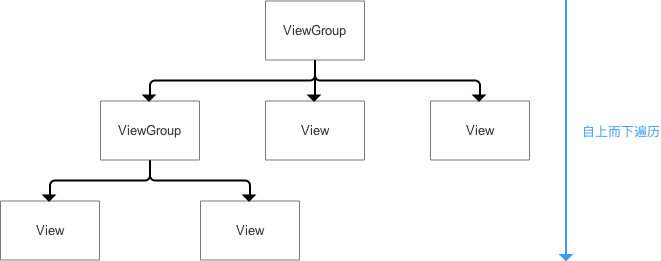
##### 1.7.6.2.1 单一view Layout流程

看上图即可

##### 1.7.6.2.2 viewGroup的Layout过程

步骤：

1. viewGroup调用layout计算自身的位置
2. viewGroup调用onLayout遍历子view并调用子view的layout确定自身子view的位置



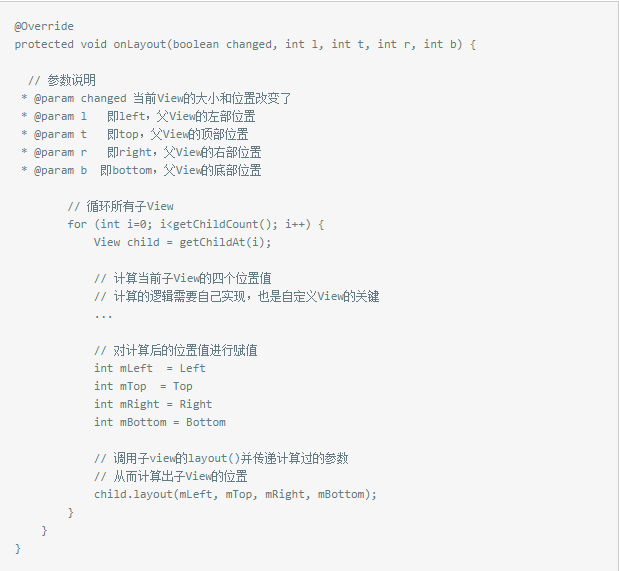
注意点：

1. 无论系统提供的Linearlayout还是自定义的view视图，都需要继承自viewGroup
2. 假如需要确定该viewGroup包含所有子view在父容器中的位置，需要重写onLayout方法（onLayout方法在viewGroup中被定义为抽象方法）

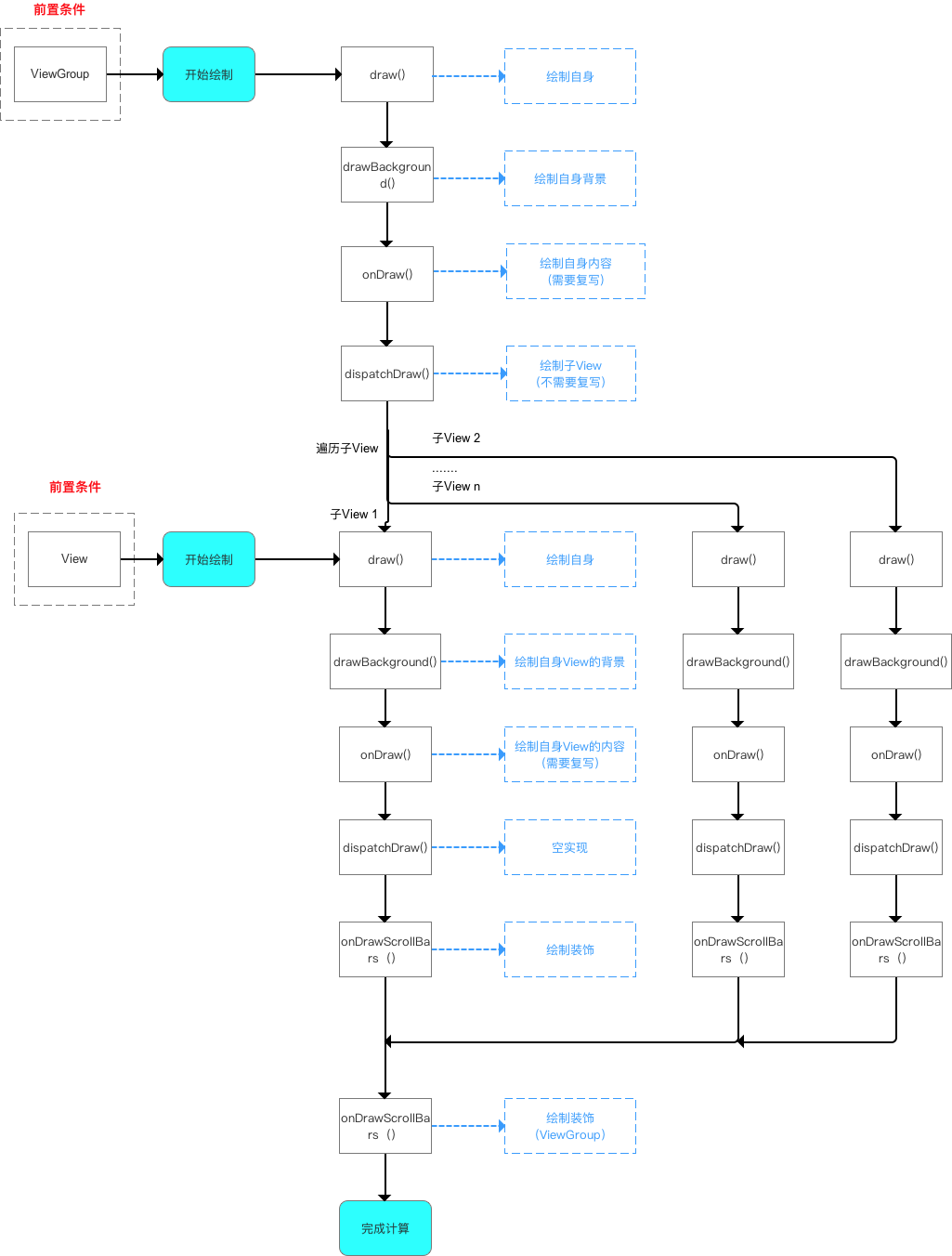
在viewGroup调用layout计算出自己的位置后，需要ViewGroup调用OnLayout遍历子view并调用子view的layout确定自身子view的位置，因为ViewGroup中子view的布局不一样所以要自己重写onLayout方法

所以重写ViewGroup的onLayout的本质是：遍历子view并调用子view的Layout确定子view的位置：

复写套路：



#### 1.7.6.3 draw过程详解



单一View中onDraw默认空实现

自定义viewGroup需要复写onDraw方法

## 1.8 屏幕适配

优秀博客：<http://www.jianshu.com/p/ec5a1a30694b>

Google的官方权威适配文档

郭霖：[Android官方提供的支持不同屏幕大小的全部方法](http://blog.csdn.net/guolin_blog/article/details/8830286)

Stormzhang：[Android屏幕适配](http://stormzhang.com/android/2014/05/16/android-screen-adaptation/)

鸿翔：[Android屏幕适配方案](http://blog.csdn.net/lmj623565791/article/details/45460089)

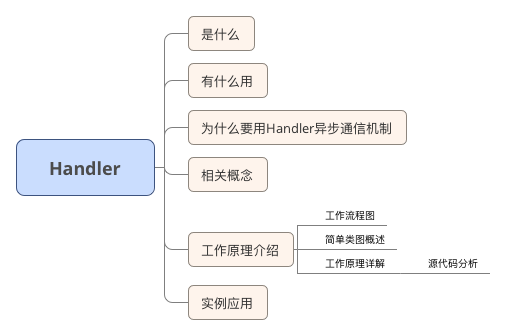
凯子：[Android屏幕适配全攻略（最权威的官方适配指导）](http://blog.csdn.net/zhaokaiqiang1992/article/details/45419023)

## 1.9 多线程编程

### 1.9.1 Android异步消息处理机制

优秀博客：[Android开发：Handler异步通信机制全面解析](http://www.jianshu.com/p/9fe944ee02f7)

目录：



定义：Android提供的一套消息传递机制

作用：实现了子线程对UI线程的更新，实现异步消息的处理：

1. 在新启动的线程中发送消息
2. 在主线程中获取并处理信息

为什么要用Handler：

1. 为了保证Android的UI操作是线程安全的，Android规定了只允许UI线程修改该activity里的UI组件
2. 但在实际开发中，必然会用到多个线程并发操作UI组件，这将导致UI操作的线程不安全

所以问题在于如何同时满足：

1. 保证线程安全
2. 使得多个线程并发操作UI组件

Handler消息传递机制就是解决这个问题的。

相关概念

主线程（UI线程）：

定义：当程序第一次启动的时候，Android会同时启动一条主线程（Main Thread）

作用：主线程主要负责处理和UI相关的事件，主线程又叫UI线程

子线程负责一些比较耗时的操作（联网、取数据、SD卡数据加载等操作），而主线程和子线程之间的通信就靠handler

Message

定义：消息，理解为线程间通讯的数据单元（handler接受和处理的消息对象）

例如后台线程在处理数据完毕后需要更新UI，则可发送一条包含更新信息的Message给UI线程

Message Queue

定义：消息队列

作用：用来存放通过Handler发送过来的消息，按照先进先出执行

Handler

定义：handler是message的主要处理者

作用：负责将Message添加到消息队列&处理Looper分配过来的Message

Looper

定义：循环器，扮演Message Queue和handler之间的桥梁

作用：主要负责消息循环：循环取出Message Queue的Message；消息派发：将取出的Message交付给相应的handler

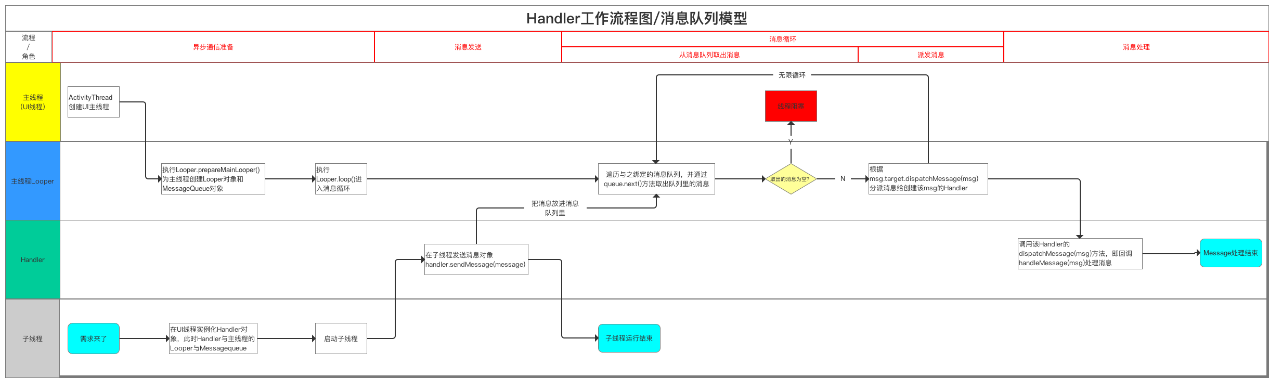
1. 每个线程中只能拥有一个Looper，但是一个Looper可以和多个线程的Handler绑定起来，也就是说多个线程可以往一个Looper所持有的MessageQueue中发送消息。

引申出的问题：当一个Activity中有多个handler时，message是如何传递的？即是Handler发送一个消息给Looper的MessageQueue，Looper从MessageQueue中取得消息时是怎么判别这个消息是从哪一个handler发送过来的。

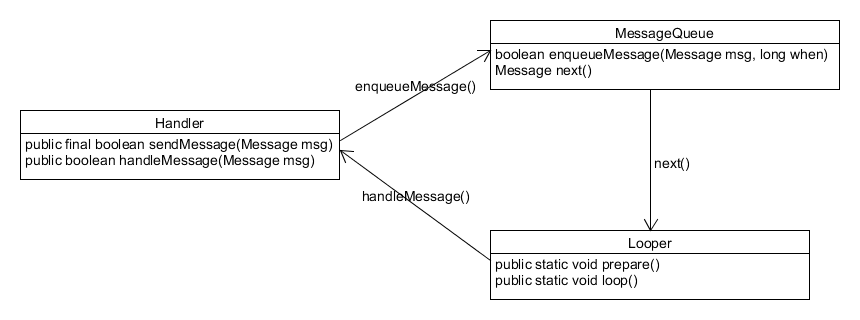
具体解答参考：<http://blog.csdn.net/u010680097/article/details/52142396>

1. handler在创建的时候可以显示指定的Looper，这样handler在调用sendMessage()投递消息时会将消息添加到指定的Looper的MessageQueue队列中，如果不指定Looper，handler默认绑定的是创建它的线程的Looper，这种情况是最常见的。

Handler异步通信机制工作流程：



Handler、Looper、MessageQueue关系类图



Handler

1. 提供sendmessage()方法，把消息放入消息队列
2. 提供handlerMessage()方法，定义各种消息的处理方式

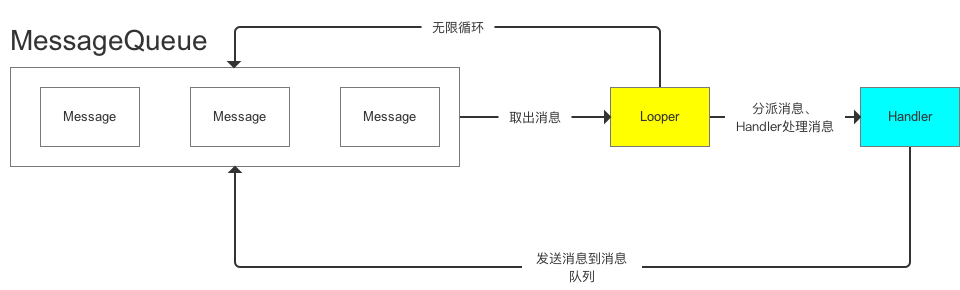
Looper

1. Looper.prepare()：实例化Looper对象；为当前线程生成一个消息队列
2. Looper.loop()：循环从消息队列中获取消息，交给Handler处理；此时Looper处于无线循环中，不停的从MessageQueue中获取Message；如果没有消息就会阻塞

MessaegQueue

1. 提供enqueueMessage方法，将消息根据时间放置到队列中
2. 提供Next方法，从队列中获取消息，没有消息时就会阻塞

Handler工作流程：



异步通信传递机制主要包括异步通信的准备、消息发送、消息循环、消息处理

1. 异步通信的准备：

包括Looper对象的创建&实例化、MessageQueue队列的创建和Handler的实例化，前两步一般Android已经完成，Handler实例化需要我们自己来，Handler实例化时如果继承自Handler的类不是static，可能发生内存泄漏，源码中有显示，Handler初始化时，可通过构造方法自动关联Looper和相应的MessageQueue

1. 消息发送：

Handler将消息发送到消息队列中

1. 消息循环：

Looper执行Looper.loop()进入消息循环，在此过程中不断从MessageQueue中取出消息，并将取出的消息派发给创建该消息的Handler

1. 消息处理

调用Handler的dispatchMassage(msg)方法，即回调handlerMessage(msg)处理消息

Handler向MessageQueue发送消息：post和send两种方式，具体用法参考以下博客

1. 相比send方法，post方法最大的不同在于，更新UI操作可以直接在重写的run方法定义
2. Runnable并没有创建线程，而是发送了一条消息，详见博客：

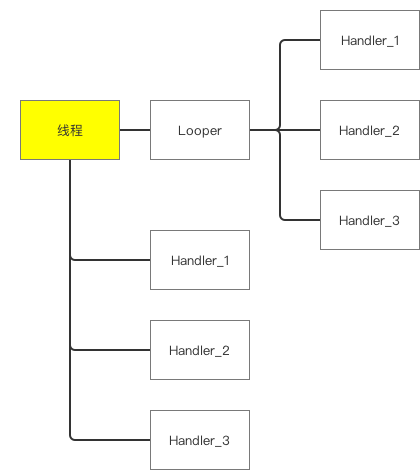
源码解析详见博客：<http://www.jianshu.com/p/9fe944ee02f7>

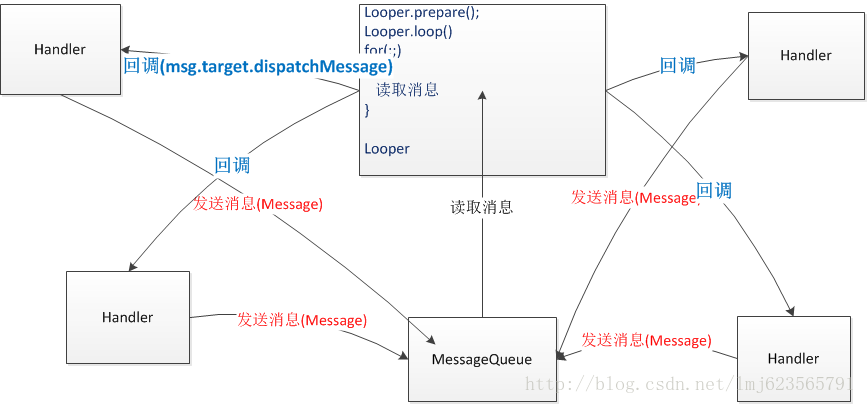
Thread、Looper、Handler之间的对应关系

1、一个线程只能有一个Looper，可以有多个Handler

2、一个Looper可以绑定多个Handler

3、一个Handler只能绑定一个Looper





### 1.9.2 AsyncTask

先明白线程池的概念：

Android中的线程池的概念来源于Java中的Executor，它是一个接口，真正线程池的实现是ThreadPoolExecutor，它提供了一系列的的参数来配置线程池，通过不同的参数可以配置不同的线程池，AsyncTask的线程池也是通过ThreadPoolExecutor来实现的，

线程池的好处：

1. 重用线程池中的线程，避免因为线程的创建和销毁所带来的性能开销
2. 能有效的控制最大线程并发数，避免大量线程之间因为相互抢占资源而导致的阻塞现象
3. 能够对线程进行简单的管理，并提供定时执行以及指定间隔循环执行等功能。

资源：参见《Android开发艺术探索》

### 1.9.3 线程同步和线程安全

### 1.9.4 HandlerThread和IntentService

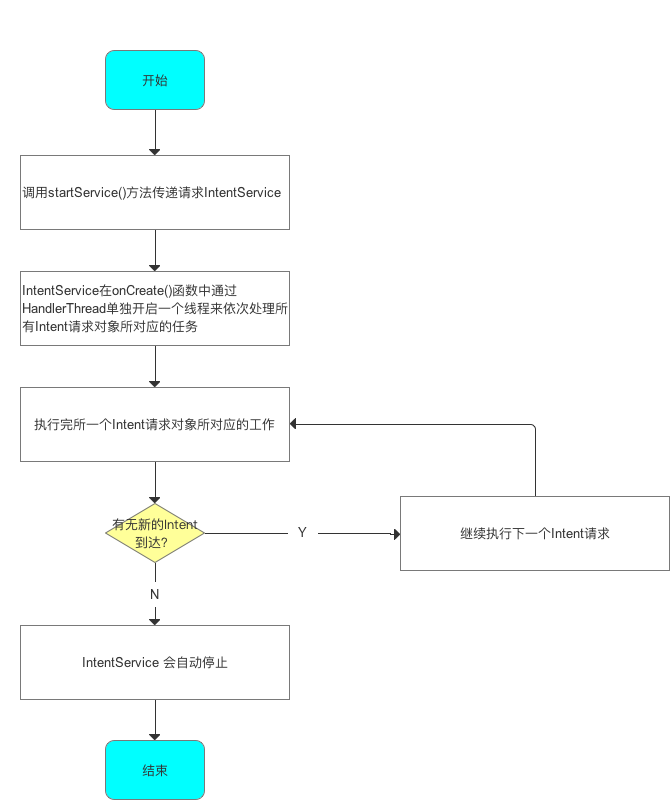
为什么存在HandlerThread？

子线程和主线程之间的通信可以用Handler和Thread实现，但是实现子线程和子线程之间的通信就要用HandlerThread，因为HandlerThread中有自己的线程，并且创建了消息队列，有自己的Looper，可以让我们在自己的线程中分发和处理消息，并对外提供自己的Looper对象的get方法。HandlerThread自带的Looper使得它可以通过消息队列，来重复使用当前线程，节省系统资源开销，这是优点也是缺点，每一个任务都会以队列的方式逐个执行，一旦队列中有某个任务执行时间过长，那么就会导致后续的任务都会被延时处理，这也是IntentService中用到的。

为什么存在IntentService？

当需要在service中执行耗时的操作，因为IntentService继承自service，是运行在主线程中的，由于不能在主线程中执行耗时的操作，所以要在service中新建一个线程，执行耗时的操作，但是IntentService通过HandlerThread单独开启了一个线程，由于只有一个线程，所以每个耗时的操作都会以队列的方式在IntentService的onHandlerIntent回调方法中依次执行，执行完后自动结束。

常用IntentService中线程任务需要按顺序、后台执行的使用场景：离线下载



IntentService用法：

1. 定义intentService的子类：传入线程名称，复写onHandlerIntent方法
2. 在manifest.xml中注册服务
3. 在Activity中开启service服务

### 1.9.5 Handler引起内存泄漏解决方式

详见：[Android开发：详解Handler的内存泄漏](http://www.jianshu.com/p/ed9e15eff47a)

## 1.10 网络请求

博客：[Android主流网络请求开源库对比（Android-Async-Http）、velly，okHttp、Retrofit](http://www.jianshu.com/p/050c6db5af5a)

### 1.10.1 Http协议定义

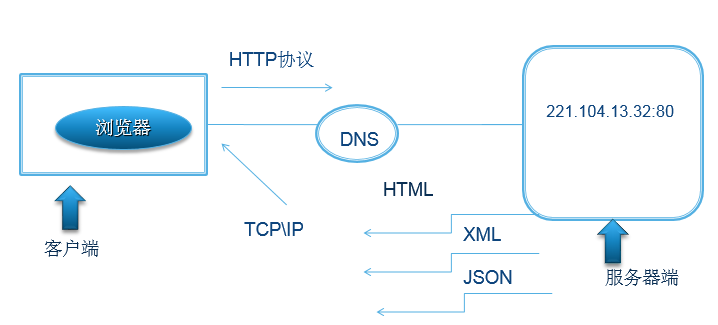
* www是以Internet作为传输媒介的一个应用系统，www网上基本的传输单位是web网页。www的工作是基于客户机/服务器计算模型，由于web浏览器和web服务器构成，两者之间采用超文本传输协议Http进行通信
* Http协议是基于TCP/IP协议之上的协议，是web浏览器和web服务器之间的应用层的协议，是通用、无状态的面向对象的协议
* 实现网络互联需要解决的问题：

1. 浏览器和服务器是通过什么来连接的
2. 连接方式是怎么实现的

* 通过Internet去发送到服务器当中，internet内部可以通过三种方式来实现发送信息和数据：

1. Http协议，是工作当中最常用的，是建立在TCP/IP基础上实现的
2. FTP协议
3. TCP/IP协议，是最低层的协议，其他的方式必须要通过它实现，但是实现这种方式必须要实现socket编程，这种方法用来上传一些比较大的文件、视频，进行断点续传的操作

### 1.10.2 Http协议实例剖析



流程分析：

1. 在浏览器客户端中得到用户输入的内容
2. 浏览器得到这个网址之后，内部会将这个域名发送到DNS上，进行域名解析，得到它的IP之后就会链接到指定的服务器上，假如服务器地址是:221.104.13.32:80，从浏览器到服务器端口它使用到最底层的TCP/IP协议
3. 实现TCP/IP协议用socket来完成，使用socket的套接字
4. 服务器端的80端口监听客户端的链接，这样客户端到服务器就链接上了

Http请求体内容：



Http响应的内容：



Http返回请求数据的三种方式：

服务器接收到请求之后，并按照这些请求的路线找到对应的页面，进一步找到对应的网页内容，返回给客户端。服务器返回给客户端的内容有三种方式：

1. HTML代码内容返回
2. 以XML字符串的形式返回，在Android开发中这种形式比较多
3. 以JSON对象形式返回，在网络流量上考虑JSON比XML方式要好一些，便于解析

### 1.10.3 Http get和post请求数据

参考：[Http中GET和POST的区别](http://www.jianshu.com/p/678ff764a253)

GET和POST：Http协议中的两种发送请求的方法

Http底层是TCP/IP，所以GET和POST的底层也是TCP/IP，也就是说GET/POST都是TCP链接。GET和POST能做的事情是一样的，如果给GET加上request body，给POST加上URL参数，技术上是完全行的通的。

区别：

在万维网世界中，TCP就像汽车，用来运输数据，它很可靠，从来不会发生丢件的现象。但是如果在路上跑的都是一模一样的汽车，那就看起来一片混乱，送急件的汽车可能被前面装满货物的汽车拦堵在路上，整个交通系统一定会瘫痪。为了避免这种情况，交通规则HTTP诞生了，HTTP给汽车运输设定了几个服务类别，GET、POST、PUT、DELETE等，HTTP规定，当执行GET请求的时候，要给汽车贴上GET标签（设置method为GET），而且要求把传输数据放在车顶上（URL）以方便记录。如果是POST请求，就要在车上贴上POST标签，并把货物放在车厢里，也可以在GET的时候往车厢里偷偷藏点货物，但是很不光彩；也可以在 POST的时候在车顶也放一些数据，让人就得傻乎乎。HTTP只是个行为准则，而TCP才是GET和POST怎么实现的基本。

HTTP对GET和POST参数的传递渠道（URL或request body）提出了要求，但是一些“标准答案”中对于参数大小的限制是从哪来的？

在万维网世界中，还有一个重要角色：运输公司。不同的浏览器（发送http请求）和服务器（接受http请求）就是不同的运输公司。虽然在理论上可以在车顶上无限堆货物（URL中无限加参数），但是运输公司不同意，因为装货和卸货是有很大的成本的，他们会限制单次运输量来控制风险，数据量太大对浏览器和服务器都是很大的负担。业界不成文的规定是，（大多数）浏览器通常都会限制URL长度在2K字节，而（大多数）服务器最多处理64K大小的URL，超过的部分，恕不处理。如果使用GET服务，在request body中偷偷藏了数据，不同服务器的处理方式也是不同的，有些服务器会帮你卸货，读出数据，有些服务器会直接忽略，所以，虽然GET可以带request body，也不能保证一定能被接收到。

GET和POST本质上就是TCP链接，并无差别，但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用的过程中体现一些不同

另一个重大区别：

GET产生一个TCP数据包；POST产生两个TCP数据包。

对于GET方式的请求，浏览器会把http header和data一并发送，服务器响应200（返回数据）；对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 OK（返回数据）。也就是说GET只需要汽车跑一趟就可以把货物送到了，POST需要跑两趟，第一次是先和服务器打好招呼，第二次再把货物送过去。因为POST需要两步，时间消耗要多一点，从这个方面看，看起来GET比POST更有效。有的团队推荐用GET替换POST来优化网站性能，但是需谨慎：

1. GET和POST都有自己的语义，不能随便乱用
2. 据研究，在网络环境好的时候，发一次包的时间和发两次包的事件差别基本可以无视，而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点
3. 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

### 1.10.4 URL的定义和组成

* Uniform Resource Locator统一资源定位符
* URL组成部分：<http://www.mbalib.com/china/index.htm>
* <http://：代表超文本传输协议>
* www：代表一个万维网服务器
* mbalib.com/：服务器的域名，或服务器名称
* china/：子目录，类似于我们的文件夹
* index.htm：文件夹中的文件
* /china/index.xml统称为URL路径

### 1.10.5 XML和JSON解析

#### 1.10.5.1 XML解析

定义：

XML（extensible markup language）,是一种数据标记语言&传输格式，可扩展性指可以根据自己需要自定义标记名，只要合理嵌套，符合规定就行，不像HTML的标记名都定义好了，有固定的作用。

作用：

1. 对数据进行标记（结构化数据）
2. 对数据进行存储
3. 对数据进行传输

与HTML的区别：html用于显示信息，xml用于存储&传输信息

解析方式：

* 基于文档驱动方式：xml DOM（xml Document Object Model）xml文件对象模型，定义了访问和操作xml文档元素的方法和接口。

工作原理：DOM是基于树形结构的节点的文档驱动方法，使用DOM对xml进行解析时，首先解析器读入整个XML文档到内存中，然后解析全部文件，并将文件分为独立的元素、属性等，以树结构的形式在内存中对XML文件进行表示，开发人员通过使用DOM API遍历XML树，根据修改文档或检索所需数据

优点：

1. 由于整棵树在内存中，因此可以对XML文档随机访问
2. 可以对xml文档进行修改操作
3. 较SAX和PULL，DOM使用较简单

缺点：

1. 整个文档必须一次性解析完
2. 由于整个文档都需要载入内存，对于大文档成本高

* 事件驱动方式：SAX（Simple API XML）和PULL方式，都是基于流（stream）操作文件，然后根据节点事件回调开发者编写的处理程序。因此Pull和SAX都比较节约内存资源，不会像DOM那样要把所有节点以树的形式展现在内存中，但是PULL比SAX更简明，而且不需要扫描完整个流

SAX和PULL的区别：

1. SAX解析器的工作方式是自动将事件推入注册的事件处理器进行处理，因此不能控制事件的处理主动结束；
2. Pull解析器的工作方式为允许你的应用程序代码主动从解析器中获取事件，因为是主动获取事件，因此可以在满足了需要的条件后不再获取事件，结束解析。这是主要的区别。

如果在一个XML文档中，只需要前面一部分的数据，但是SAX方式和DOM方式对整个文档进行解析，中间不能终止暂停，尽管XML文档中后面的大部分数据都不需要解析，这就浪费了处理资源。使用PULL方式正合适，

PULL解析器也提供了类似SAX的事件，开始文档START\_DOCUMENT和结束文档END\_DOCUMENT，开始元素START\_TAG和结束元素END\_TAG，遇到元素内容TEXT等，但需要调用next()方法提取他们（主动提取事件）。

PULL解析经常使用到的类：

XmlPullParserFactory：工厂类，用来创建PULL解析器

XmlPullParser：PULL解析器

XmlPullParserException：异常

XmlSerializer：该类用来将指定的对象转化为XML文档

#### 1.10.5.2 JSON解析

网站工具：<https://www.bejson.com/>。可以进行解析、编辑、生成JavaBean等一系列操作，也可用Android Studio自带的工具GsonFormat，生成JavaBean，具体参考：

[Android studio插件GsonFormat-自动生成JavaBean](http://www.jianshu.com/p/d34c5e7c8227)

定义：JavaScript Object Notation，JavaScript的对象表示法，是一种轻量级的文本数据交换格式

作用：用于数据的标记、存储和传输

特点：

* 轻量级的文本数据交换格式
* 独立于语言和平台
* 具有自我描述性
* 读写速度快，解析简单

语法：JSON值

* 名称/值
* 数组
* 对象

解析方法：

* 基于事件驱动：Gson解析和Jackson解析，根据所需要取的数据通过建立一个对应于JSON数据的JavaBean类就可以通过简单的操作解析出所需JSON数据
* 基于文档驱动：Android Studio自带org.json解析，先全部文件读入内存中，再遍历所有数据，根据需要检索想要的数据。

GSON、Jackson、AndroidStudio自带org.son解析三类方式对比：

Org.son：

* 原理：基于文档驱动
* 优点：无
* 缺点：解析XML文件时会将整个XML文件的内容解析成树形结构存放在内存中并创建新对象，比较消耗时间和内存，解析速度和效率慢，解析方式和性能完败GSON

GSON：

* 原理：基于事件驱动
* 优点：解析方法简单、解析效率高、占内存少、灵活性高
* 使用情景：适用于需要处理大型JSON文档，JSON文档结构复杂的场合

Jackson：

* 原理：基于事件驱动
* 优点：解析效率高、在数据量大的情况下优势尤为明显，占存少
* 缺点：必须完全解析文档，如果想要按需解析的话可以拆分Json来读取，操作和解析方法复杂
* 使用情景：适用于需要处理超大型JSON文档，不需要对JSON文档进行按需解析，性能要求较高的场合

## 其他待整理

### Intent

显示调用（PackageName/ClassName）：指定Intent组件名称来实现，一般在相同的应用程序内部实现。

隐式调用（Action）：通过Intent Filter实现，一般用在没有指明目标组件名称的前提下，一般用于不同应用程序之间

### JNI和NDK

优秀博客：[Android：JNI和NDK到底是什么（含实例教学）](http://www.jianshu.com/p/38859f881888)

# Android提高

## 1.1 设计模式

# Git使用

## Git基本概念

版本控制：记录若干文件变化，以便将来查阅特定版本修订情况的系统。

文件的三种状态：对任何一个文件，在git内都有三种状态：已提交（committed），已修改（modified），已暂存（staged）。已提交：表示该文件已经被安全的保存到了本地数据库中；已修改：表示已经修改了某个文件，但是还没有提交保存；已暂存：表示把已修改的文件放在下次提交时要保存的清单中

每个项目都有一个Git目录（.git）它是Git用来保存元数据和对象数据的地方，每次克隆镜像仓库的时候，实际拷贝的就是这个目录里面的数据

Git工作流程：在工作目录中修改某些文件；对修改后的文件进行快照，然后保存到暂存区域；提交更新，将保存在暂存区域的文件快照永久转存到Git目录中。

对现有的某个项目开始使用git管理：git init；.git文件是隐藏的，用于保存下载下来的所有版本记录想要看里面的内容需要在文件夹下显示。

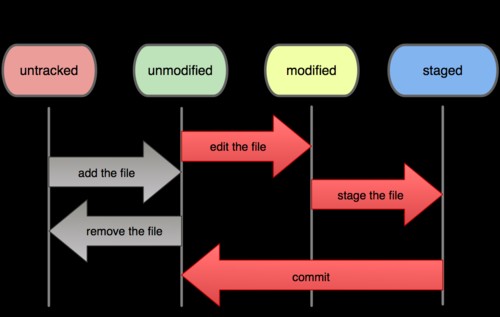
如果当前目录有几个文件想要纳入版本控制，需要git add命令告诉git开始追踪这些文件，然后提交

从现有仓库克隆：git clone [url]；

$ git clone git://github.com/schacon/grit.git mygrit

git支持许多数据传输协议，git：//；http(s)://；user2server：/path.git表示的ssh传输协议

文件状态生命周期：已跟踪和未跟踪，已跟踪的文件时指本来就纳入版本控制管理的文件。



git status命令：用于检查当前文件的状态

git add命令：用于跟踪新文件

git commit命令：用于提交文件

git diff命令：查看不同（difference）

git log命令：显示从最近到最远的提交日志

git reset --hard HEAD^：退回到上一个版本，如果想在退回到上一个版本之后再回到退回之前的版本，git reset --hard [commit id](退回之前的版本的commit ID的前几位即可，通过git log可以查看)，HEAD表示当前版本，HEAD^表示上一个版本，上上一个版本是HEAD^^表示，往上100个版本用HEAD~100表示

git reflog查看提交历史，查看记录每一次命令，前面显示的是commit id，可以通过commit id恢复到以往的任意一个版本

git checkout -- file：恢复到最近一次git commit或者git add之前的状态

cat：查看文档内容

## 建立本地git和远程GitHub仓库的联系

具体怎么创建联系的参照《Git教程By廖雪峰》

git仓库和GitHub之间的传输是通过SSH加密的

git push -u origin master：第一次推送master分支所有内容

git push origin master：推送最新修改

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快

.gitignore文件：是列出要忽略的文件模式

所有空行或者以注释符号#开头的行都会被Git忽略

/说明要忽略的是目录；

\*匹配零个或多个任意字符

[abc]匹配任何一个列在方括号中的内容，[0-9]表示所有在这两个字符范围内的都可以匹配

？只匹配一个任意字符

## 分支管理

查看分支：git branch

创建分支：git branch name

切换分支：git checkout name

创建+切换分支：git checkout -b name

合并某分支到当前分支：git merge name

删除分支：git branch -d name

解决冲突

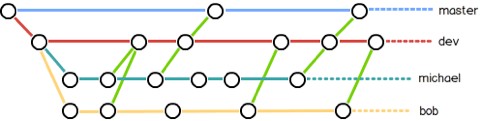
当git无法自动合并分支时，就必须首先解决冲突，解决冲突后，再提交，合并完成

git log --graph命令可以看到分支合并图

## 分支策略

master分支是非常稳定的，仅仅用来发布新版本，平时不能在上面干活，

干活都在dev分支上，dev分支是不稳定的，到某个版本发布时，再把dev分支合并到master分支上，在master分支发布11.0版本



合并分支，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

## Bug分支

修复bug时，通过创建新的bug分支进行修复，然后合并，最后删除

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场

## Feature分支

开发新功能，最好建立一个新分支，如果要丢弃一个没有被合并过的分支，可以通过git branch -D name强行删除

## 多人协作

查看远程库信息，使用git remote -v

本地新建的分支如果不推送到远程，对其他人就是不可见的

从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交

在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地分支和远程分支的名字最好一致

建立本地分支和远程分支的关联，使用 gitbranch --set-upstream branch-name origin/branch-name

从远程抓取分支，使用git pull，如果有冲突，要首先处理冲突。

## 创建标签

命令git tag name 用于新建一个标签，默认HEAD，也可制定一个commit id

-a tagname -m "bababababab"可以指定标签信息

命令git tag可以查看所有标签

## 操作标签

命令git push origin tagname可以推送一个本地标签

命令git push origin --tags可以推送全部未推送过的本地标签

命令git tag -d tagname 可以删除一个本地标签

命令git push origin ：refs/tags/tagname可以删除一个远程标签

## 使用github

github上可以任意Fork开源仓库

自己拥有Fork后的仓库的读写权限

可以推送pull request给官方仓库来贡献代码

# adb使用

什么是adb：借助adb工具，可以管理设备或手机模拟器的状态，进行手机操作：安装软件、系统升级、运行shell命令等，简单而言就是连接Android设备和pc端的桥梁，可以让用户在电脑上对手机进行全面的操作。

## 1.1 adb常用基本命令

adb root：获取管理员权限

adb logcat -s 标签名：在命令行中查看LOG信息

adb start-server：开启adb服务

adb kill-server：终止服务

adb devices：列出所有连接到adb的设备，包括真机和模拟器

adb -s <serialNumber>：指定adb指令的目标

adb version：显示adb的版本号

adb install <apk file name>：将指定的APP安装到唯一连接到adb的设备上，后边的参数为安装包的文件名

adb install -r <apk file name>：安装应用程序，但是保留原有应用程序的所有数据

adb uninstall <package name>：将包名指定的app从目标设备上卸载，后面的参数为应用程序包名

adb uninstall -k <package name>：卸载指定包名的APP，但是保留配置文件和缓存

adb push <local> <remote>：将本机的文件或目录拷贝到本机中

adb shell：进入到目标设备的linux环境，在该环境中可以执行Linux命令，linux shell环境中执行ps可以查看Android设备中运行的所有进程。执行exit可以退出linux shell环境

## 1.2 adb logcat命令查看并过滤Android输出log

Android输出的每一条日志都有一个标记和优先级与其关联：就是tag和优先级

ctrl+c中断输出log记录

adb logcat：查看全部日志

adb logcat > c:\test.log ：将日志保存到文件test.log中

例如：仅输出标记为“ActivityManager”且优先级大于等于“Info”和标记为“PowerManagerService”并且优先级大于等于“Debug”的日志

adb logcat ActivityManager：I PowerManagerService：D \*：S

\*：S用于设置所有标记的日志优先级为S，这样可以确保仅输出符合条件的日志

# 待学习

评审操作指导：<http://smartphone/repositories/DocsAndTools/Docs/SW_Team/MobileInternet/%E4%BA%92%E8%81%94%E7%BD%91%E5%BA%94%E7%94%A8%E7%BB%84/%E6%B5%81%E7%A8%8B%E4%B8%8E%E6%96%87%E6%A1%A3/>

Android-Java编程规范：

<http://smartphone/repositories/DocsAndTools/Docs/SW_Team/MobileInternet/%E4%BA%92%E8%81%94%E7%BD%91%E5%BA%94%E7%94%A8%E7%BB%84/%E8%B5%84%E6%96%99%E5%BA%93/02.%E4%BB%A3%E7%A0%81%E5%9F%BA%E7%A1%80/1.%E5%BC%80%E5%8F%91%E8%A7%84%E8%8C%83/1.vivo%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83/>

# 专利

专利点子从哪来：

1. 手机软件痛点，
2. 看别人专利，佰腾专利查询

专利的查询要看看别人有没有涉及到，就像写的《一种便捷设置音量和屏幕亮度的方法和智能终端》，乐视有一篇《无物理滑动实体按键的移动终端》，这篇专利中的应用场景已经包括了电容传感器的应用场景，所以就失去了创新性。

《一种智能提醒的移动终端设备》：

用户设置提醒时的待办事项可以分为两类，一种是主动要做的事项；一种是被动要做的事项。其中应用最多的是火车、飞机、轮船等定时出发类的事项。出发类事项一般情况下都会在订票的时候发送短信，如12306、深圳航空等，但如果不是使用第三方软件定的火车票或者机票，不会在出发前提醒乘客，本发明的主要目的就是自动获取乘客订票时12306发送的票务信息，并提取乘车时间和乘车地点，根据乘车时间系统自动设置提醒。

发散点：

1. 默认的是出行提醒，其实只要是被动式的要做的事项都可以设置
2. 提醒信息上还可以根据用户当前的位置信息，提醒用户到达乘车地点需要的时间，以及路径规划等。

**专利点子和关键词：**

一种基于WIFI或蓝牙的智能来电系统，面部识别，手机移动热点、根据用户使用习惯设置铃声、

**手机组件：**

* 传感器：
  1. 光线传感器：应用屏幕自动调节亮度，拍照自动平衡，配合距离传感器检测手机是否在口袋
  2. 距离传感器：红外LED灯发射红外线，被近距离物体反射后测距，有效10cm
  3. 重力传感器
  4. 加速度传感器
  5. 磁场传感器
  6. 陀螺仪
  7. GPS
  8. 指纹传感
* 蓝牙、WIFI、热点

# 零散杂乱知识

Log.*e*(*TAG*, "attachBaseContext: ", new Throwable());

打印栈内信息。

# Vivo考试

常识：

CheckBox被选择的事件setOnCheckChangeListener

LinearLayout默认是Horizontal排列

分析内存泄漏问题，主要使用MAT工具

View更新过程中，哪些阶段能获取到view的准确大小：onFinishInflate，onLayout，onDraw。

Android进程的重要性排序？前台进程（正在和用户交互），可见进程，服务进程，后台进程，空进程

栈是一块连续的内存区域，大小是由操作系统决定的；堆是不连续的区域，堆空间比较灵活也特别大

Java四种引用类型及其强弱顺序：强引用、软引用(只有在内存不足时jvm才回收该对象)、弱引用(当jvm垃圾回收时，无论内存是否充足，都会回收被弱引用关联的对象)、虚引用

Android中文件操作模式：MODE\_WORLD-READABLE(创建出来的文件可以被其他程序读取)、MODE\_APPEND(若文件不存在就创建文件，若文件存在则在文件末尾添加内容)、MODE\_PRIVATE(只能被创建这个文件的当前应用程序访问，若文件不存在则创建文件，若文件存在，则会覆盖掉原来的文件)、MODE\_WORLD\_WRITEABLE(允许其他应用写入)

添加第一个appwidget小部件，会执行的方法：onEnabled、onUpdate、onReceive

SparseArray（稀疏数组）：key智能是int型、value是object类型、比HashMap更省内存，在某些条件下性能更好、在存储和读取数据的时候，使用二分法查找。

RatingBar组件中不能用属性直接设置的是“五角星的色彩”

autoText是对TextView中的字体进行矫正

singleLine单行显示、lineSpacingMultiplier行间距的倍数、lineSpacingExtra行间距数值、lines多行显示

Android资源分类：直接访问资源，都保存在res文件下，使用R.进行访问；原生资源放在assets下，不能用R.进行访问，只能通过Assetmanager以二进制流的方式进行访问。

SurfaceView播放视频时需要Mediaplayer配合，使用SurfaceView播放视频时可以改变播放的位置和大小；videoView播放视频不能改变播放位置和大小，不需要MediaPlayer配合。

短信内容提供者的主机名是：sms

Android不支持静态注册的广播android.intent.action.SCREEN\_ON/SCREEN\_OFF/BATTERY\_CHARGED/CONFIGURATION\_CAHANGED/TIME\_TICK

WebView三大组件：WebSettings(对WebView进行各种设置)、WebViewClient(帮助WebView处理各种通知、请求事件)、WebChromeClient(帮助WebView处理JS对话框，网址图标，网址标题和加载进度等)

SetResultData()修改广播接受者的数据，只能存储一个String

代码中获取注册广播事件的方法getAction()

Android版本系统中的基本菜单：PopupMenu弹出菜单、ContextMenu上下文菜单、OptionMenu选项菜单

Android单线程模式必须遵守的规则：不要阻塞UI线程、不要在UI线程之外访问Android UI工具包

GLSurFaceView不能直接从内存或者DMA等硬件接口中获取图像数据

250题

Android 系统对哪些对象提供了资源池？Message、AsyncTask

Java中引用类型及其作用？Java中对图片缓存常用软饮用

Java线程之间通信的方式？nitifyAll()、notify()、wait()

View 的绘制流程？

Android中的view hierarchy使用到的核心设计模式？composite

Bindservice方式启动service，生命周期会回调哪些方法？

333题

350

450

# 疑问：

# 分享

BroadcastReceiver：参考Carson\_Ho博客

View的绘制流程和事件分发机制：参考“故事”和本文档

# 其他

【分配成功】司向军 分配的服务器信息为==>服务器地址为：172.20.200.227 用户名为：sixiangjun 服务器登陆密码：123 服务器SAMBA挂载密码:j0ukvspp 温馨提示：新员工入职请看： <http://192.168.2.27/mediawikis/index.php/%E6%96%B0%E5%91%98%E5%B7%A5%E5%85%A5%E8%81%8C%E6%8C%87%E5%AF%BC>

软件类常用工具请访问 <http://it.vivo.xyz/>

软件工程师常用信息请看 <http://192.168.2.27/mediawikis/index.php/%E5%88%86%E7%B1%BB:%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%B8%B8%E7%94%A8%E4%BF%A1%E6%81%AF>

使用服务器下载代码异常、没有权限请找配置组田慧玲,如果是更换服务器,请大家在收到新服务器的账好后,在在三天内将老服务的资料转移过去,谢谢!!有任何异常请与余建辉联系 手机号码:13714806625