**Android基础**

**Android启动流程全解析：https://tower.im/teams/603160/todos/22/**

1. **Activity和Fragment的生命周期**

Activity.

启动Activity：onCreate() -> onStart() -> onResume()

退居后台：onpause() -> onStop()

重返前台：onStart() -> onResume()

后台运行且内存不足： onPause() -> onStop() -> Kill Process

重启该Activity：onCreate() -> onStart() -> onResume()

启动新Activity,并把旧Activity finish()掉：旧Activity的onPause() -> 新Activity的onCreate() -> onStart() -> onResume() -> 旧Activity的onStop() -> onDestory()， 因此不要在onPause()做耗时的逻辑操作。避免卡顿、ANR异常。

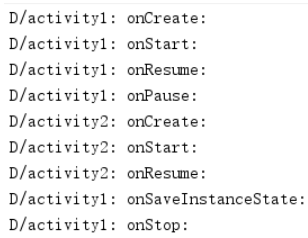
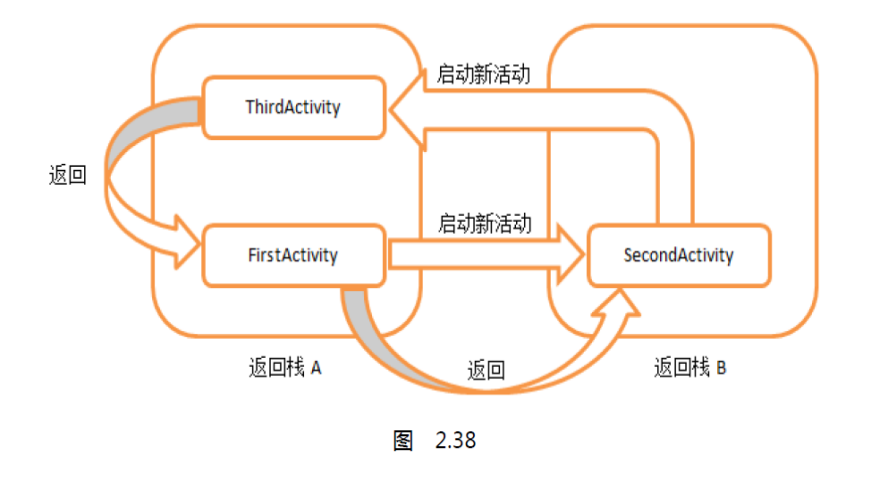
锁屏：onPause() -> onStop()

解屏：onStart() -> onResume()

Fragment.

onAttach() -> onCreate() -> onCreateView() -> onActivityCreate() -> onStart() ->onResume() -> onPause() -> onStop() -> onDestoryView() –> onDestory() ->onDetach()

1. **Activity的四种启动模式及其特点**
2. Standard：标准模式，每次创建Activity都会压进任务栈栈顶。不管该Activity是否存在于任务栈内。 完整生命周期流程。执行完下一Act..y生命周期后，onSaveInstanceState；
3. SingleTop：栈顶复用模式，每次创建Activity时都会检查该Activity和栈顶Activity是否相同，如果是同一个Activity实例则复用该栈顶Activity，不在重新创建Activity实例和执行生命周期方法。
4. SingleTask：栈内复用模式，每次 创建Activity实例时都会检测任务栈内是否存在相同Activity实例，如果有则复用该Activity实例并执行onNewIntent方法，并且将该Activity实例之上的Activity移除出栈。没有则跟standard。过程：先把中间的Acty-onDestory，然后当前Acty-pause、复用的Acty-onNewIntent、onRestart、onStart、onResume；再把上一个Act销毁；
5. SingleInstance：加强版的SingleTask模式，每次创建一个Activity实例都会重新创建一个新的任务栈来维护，实例独占任务栈。其他实例则在标准任务栈上维护。（主栈、副栈）



1. **Activity的缓存方法**
2. B两个Activity，A进入B后A实例可能会被内存回收（系统强行回收），导致返回A时不掉用onRestart方法而是调用onCreate方法。导致原有数据丢失。

**\*\* onSaveInstanceState()**

这时，我们可以重写onSaveInstanceState()方法保存临时数据、View的状态。--当Activity可能被销毁的情况：

--按Home键回主界面

--长按Home键跳转App

--按下电源键

--启动新Activity

--横竖屏切换（未指定configChange的情况）

@Override

protected void onSaveInstanceState(Bundle outState) {

super.onSaveInstanceState(outState);

outState.putString("anAnt","Android");

}

super.onSaveInstanceState()已帮我们实现部分UI状态存储。

onSaveInstantState()尽量用来保存临时信息、UI状态。。不要用来做持久化存储。

**\*\* onRestoreInstanceState()**

onSaveInstateState和onRestoreInstateState()不一定成对触发。

onRestoreIntanceState()的buddle也会传入onCreate方法中。我们也可以在onCreate方法上进行数据还原。 还有onRestoreInstateState()方法在onStart()之后执行。

@Override

public void onSaveInstanceState(Bundle savedInstanceState) {

savedInstanceState.putBoolean("MyBoolean", true);

savedInstanceState.putDouble("myDouble", 1.9);

savedInstanceState.putInt("MyInt", 1);

savedInstanceState.putString("MyString", "Welcome back to Android");

// etc.

super.onSaveInstanceState(savedInstanceState);

}

@Override

public void onRestoreInstanceState(Bundle savedInstanceState) {

super.onRestoreInstanceState(savedInstanceState);

boolean myBoolean = savedInstanceState.getBoolean("MyBoolean");

double myDouble = savedInstanceState.getDouble("myDouble");

int myInt = savedInstanceState.getInt("MyInt");

String myString = savedInstanceState.getString("MyString");

}

1. **Service的生命周期，启动的两种方法**

在上下文环境下使用

1. Context.bindService(Intent service, ServiceConnection conn, int flag)

**onCreate -> onBind -> onUnbind -> onDestory**

1. Context.StartService(intent Service)

**onCreate -> onStartCommand -> onDestory**

bindService -- Service生命周期与Context绑定，Context销毁，Service就销毁。

StartService --Service生命周期不与Context关联，Service不会随Context销毁而销毁。

但无论哪种方式启动，都需要在manifest清单文件上注册/声明：

<service

android:name=".packnameName.youServiceName"

android:enabled="true" />

Ps：ServiceConnection 重写onServiceConnected和onServiceConnected方法。

1. **怎样保证Service不被杀死**
2. Service的onStartCommand方法上返回值设置成**START\_STICKY。**

kill 后会被重启（等待5秒左右），重传Intent，保持与重启前一样

1. 提升Service进程的优先级。

在startCommand上调用startForeGround()使服务状态变为前台服务。在onDestory()上调用stopForeGround()。像手机播放器一样不管手机如何休眠，运行在后台依然不停播放。

1. 在onDestory()上重启Service

在onDestory方法上再startService。

1. 通过广播监听Service状态。

通过service + brocast的方式当service执行onDestory时，发送自定义广播，Recever接收到广播后重新启动Service。

1. 放一个像素在前台。（手机QQ）
2. **广播的两种注册方法，区别？**

静态注册：Manifest清单文件上对广播进行注册声明。

动态注册：在Activity组件上用代码进行动态注册。

区别：静态注册在App退出时Receiver仍然能够接收广播。

动态注册当App退出时，广播就无法被接收。

1. **Intent的使用方法，能传递什么？**

Intent是组件间进行交互的重要方式。通过Intent机制，创建Activity、Service、BrocastReceiver组件，还可以在组件间传递数据。

显式intent：new intent时就指定想要打开的组件。

隐士intent：通过一系列的匹配动作，action、category来匹配过滤出相应的组件。

隐式intent还可以打开其他应用的Activity组件，比如浏览器、拨号。。

常用方法：

it.putExtras("bd",“xxx”);

intent.getExtra("bd")

传递类型：

基本数据类型及其数组、String/CharSeq及其数组、Parcelable\Serializable序列化对象。

1. **ContentProvider使用方法**

内容提供者是用来供给其他应用程序访问当前应用的数据的一个组件。实现不同的应用程序之间进行数据共享的功能。

使用：

在当前应用上创建ContentProvider[[1]](#footnote-0)内容提供者，并实现他的 增删改查和onCreate、getType方法，并且通过uriMatach添加他的Uri匹配路径。

在其他应用上通过Context.getContentResolver获取内容解析者，调用他的增删改查方法并传入匹配的Uri和匹配参数即可操作其他应用的数据。

1. **Thread、AsycTask、IntentService的使用场景及特点**

一般开发中执行一些耗时操作或异步操作（加载网络、访问数据库）时，通常需要使用Thread类创建一个新线程来执行。

IntentService是本质上是Service，基于Service+Handler 只是在Service上又做了一些异步的多线程的封装。我们可以在onHandleIntent上面执行耗时操作，执行完毕后自动销毁Service。

AsyncTask基于Thread池 和 Handler，帮助我们进行了异步多线程的封装。能够避免程序线程过多导致的性能开销过大。主要方法：onPreExecute()-主 doInBackground()-子 onPostExecute()-主 onProgressUpdate() –主，需要在doInBackgrond()上调用publishProgress()..

1. **五大布局：FrameLayout、LinearLayout、AbsoluteLayout、RelativeLayout、TableLayout（ConstraintLayout 加强可视化操作、可视化编写界面）**

FrameLayout：帧布局，子控件默认叠加在左上角。我们可以通过layout\_gravity控制子控件的相对位置。

Linearlayout：线性布局，以一行或一列为一个控件独占的布局，需要Orientation指定具体排列方式。

RelativeLayout：相对布局，布局内部可放置多个控件，每个控件间的相对位置可以相对排列。常用：layout\_centerInparent/alignParentRight..

AbsoulteLayout：绝对布局，放置多个空间，每个控件自定义x，y坐标。会有屏幕适配问题。

TableLayout：表格布局，将子控件分配到行或列上，一个TableLayout有多个TableRow组成。

ConstraintLayout：约束布局，android新加入的布局，用于强化图形界面手动绘制界面，加强可视化操作。 （强化可视化操作，通过图形界面去绘制我们的布局我们的界面元素）

1. **Android数据存储形式**

SharePreference：轻量级的数据存储，本质是一个xml文件，通过键值对存储数据。

File文件存储：也就是我问说文件IO，用于存储大量数据，需要解决数据同步更新问题。

Sqlite数据库：Andoroid提供轻量级的数据库数据存储，支持常用的Sql语句。Android提供SqliteDatabase这个类，封装了一些API，让我们更方便的操作数据库（db.insert/update/delete/rawquery-sql查询/query-api查询）。

ContentProvider：内容提供者存储数据，用于提供给其他应用来进行的数据操作。暴露URI。

MMKV：总言之，常规文件操作需要从磁盘到内核缓存再到用户空间的两次数据拷贝。而mmap操控文件，只需要从磁盘地址映射到用户空间只经历一次拷贝过程。说白了，mmap的关键点是**实现了用户空间和内核空间的数据直接交互**而省去了空间不同数据不通的繁琐过程。因此mmap效率更高。

**MMAP特点：内存映射**

由上文讨论可知，mmap优点共有一下几点：

1、对文件的读取操作跨过了页缓存，减少了数据的拷贝次数，用内存读写取代I/O读写，提高了文件读取效率。

2、实现了用户空间和内核空间的高效交互方式。两空间的各自修改操作可以直接反映在映射的区域内，从而被对方空间及时捕捉。

3、提供进程间共享内存及相互通信的方式。不管是父子进程还是无亲缘关系的进程，都可以将自身用户空间映射到同一个文件或匿名映射到同一片区域。从而通过各自对映射区域的改动，达到进程间通信和进程间共享的目的。

同时，如果进程A和进程B都映射了区域C，当A第一次读取C时通过缺页从磁盘复制文件页到内存中；但当B再读C的相同页面时，虽然也会产生缺页异常，但是不再需要从磁盘中复制文件过来，而可直接使用已经保存在内存中的文件数据。

4、可用于实现高效的大规模数据传输。内存空间不足，是制约大数据操作的一个方面，解决方案往往是借助硬盘空间协助操作，补充内存的不足。但是进一步会造成大量的文件I/O操作，极大影响效率。这个问题可以通过mmap映射很好的解决。换句话说，但凡是需要用磁盘空间代替内存的时候，mmap都可以发挥其功效。

1. **Sqlite的基本操作**

一般我们使用Sqlite数据库，都会自己写一个类MySQLiteHelper继承SQLiteOpenHelper并实现抽象方法：onCreate() --创建表 onUpgrade() –版本号提升, 更新数据库.

后期通过helper.getReadableDatabse()/getWritableDatabase() 获取数据库db，增删改查!

public class StudentDatabaseHelper extends SQLiteOpenHelper {  
 private Context mContext;  
 private static final String CREATE\_STU = "create table if not exists stu(\_id integer primary key autoincrement," +  
 "name text, sex text, class text, born text, address text, phone text)";  
  
 public StudentDatabaseHelper(Context context, String name,  
 SQLiteDatabase.CursorFactory factory, int version) {  
 super(context, name, factory, version);  
 mContext = context;  
 }  
 @Override  
 public void onCreate(SQLiteDatabase db) {  
 db.execSQL(CREATE\_STU);  
 Toast.makeText(mContext, "Create STU success!~", Toast.LENGTH\_SHORT).show();  
 }  
 @Override  
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
  
 }  
}

public boolean onCreate() {  
 dbHelper = new StudentDatabaseHelper(getContext(), "stu.db", null, 1);  
 return true;  
}  
  
public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String selection, @Nullable String[] selectionArgs, @Nullable String sortOrder) {  
 SQLiteDatabase db = dbHelper.getReadableDatabase();  
 Cursor cursor = db.query("stu",projection,selection,selectionArgs,null,null,sortOrder);  
 return cursor;  
}

1. **Android中的 MVC**

View 视图层

Controller 控制层

Model 模型层

接收用户操作，将数据指令传递给Controller层。然后Controller完成业务逻辑后，请求Model层状态改变，Model层将新数据发送到View层，用户界面得到反馈。

1. **Include、merge、viewstub的作用**

**布局优化!**

**-- <include>:**包含布局，当我们写界面写得比较大时，我们可以将界面划分出几个独立的子布局，然后再通过include标签将这些布局包含进来。

**-- <merge>:**用来减少我们布局层级嵌套的问题，比如我们在一个界面上include上另一个布局，这时候include布局上面的顶层的Linearlayout、RelativeLayout就无意义了，而且他还会消耗系统资源。这时我们可以使用Merge标签，减少视图层级。

**--<viewstub>:**布局需要时使用，通过ViewStub标签包含的内容初始化时不会被加载，可达到节省内存的目的，后期通过代码可以动态令布局内容显示。

<ViewStub

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout="@layout/progressbar"

android:id="@+id/viewStub"/>

代码动态显示：

1. ((ViewStub) findViewById(R.id.stub\_import)).setVisibility(View.VISIBLE);
2. // or
3. View importPanel = ((ViewStub) findViewById(R.id.stub\_import)).inflate();
4. **Json的优势**

首先Json是一种轻量级的数据交换格式，数据以键值对来存储。

Json优势：

1. 数据格式简单、易于读写，没xml那么沉余多标签。
2. 数据格式带压缩，占用带宽小。
3. 支持多种语言，便于服务器端解析和处理。

缺点：

使用广度没xml那么高，兼容性没xml格式数据那么好。

1. **动画主要有哪几类？各有什么特点？**

补间动画、帧动画、属性动画

**补间动画：**一般分为4种动画方式，alpha（透明渐变）、translate（平移）、scale（缩放）、rotate（旋转），还能组合Set。通过指定View控件或布局的开始跟结束时状态和变化时间，就可以完成一系列动画效果。 实现方式也有两种：Xml文件 和 Java代码

Xml：

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

<scale xmlns:android="http://schemas.android.com/apk/res/android"

android:duration="1000"

android:fromXScale="0.0"

android:fromYScale="0.0"

android:pivotX="50%"

android:pivotY="50%"

android:toXScale="1.0"

android:toYScale="1.0"/>

//MainActivity加载

Animation animation = AnimationUtils.loadAnimation(mContext, R.anim.alpha\_anim);

img = (ImageView) findViewById(R.id.img);

img.startAnimation(animation);

Java代码：

Button mButton = (Button) findViewById(R.id.Button);

// 步骤1:创建 需要设置动画的 视图View

Animation rotateAnimation = new RotateAnimation(0,270,Animation.RELATIVE\_TO\_SELF,0.5f,Animation.RELATIVE\_TO\_SELF,0.5f);

// 步骤2：创建旋转动画的对象 & 设置动画效果：旋转动画对应的Animation子类为RotateAnimation

rotateAnimation.setDuration(3000);

// 固定属性的设置都是在其属性前加“set”，如setDuration（）

mButton.startAnimation(rotateAnimation);

// 步骤3：播放动画

缺陷：

作用对象局限，只能作用View或布局，无法改变视图的某事属性，不如颜色。

没有真正改变View的属性，不会真正改变位置效果，只是视觉效果。

动画效果单一，只能是4种动画效果或跟他们的组合。

**帧动画：**

帧动画是Android动画中比较容易实现的一种动画，其核心是通过一张张图片形成帧从而形成连续播放的动画效果。类似于gif动画一样。（animation-list、AnimationDrawable）

实现：

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

<animation-list xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:drawable="@drawable/a\_0"

android:duration="100" />

<item

android:drawable="@drawable/a\_1"

android:duration="100" />

<item

android:drawable="@drawable/a\_2"

android:duration="100" />

</animation-list>

//MainActivity实现

ImageView img= (ImageView) findViewById(R.id.animation1);

img.setImageResource(R.drawable.frame\_anim1);

AnimationDrawable animationDrawable1 = (AnimationDrawable) img.getDrawable();

animationDrawable1.start();

**属性动画：**

新SDK提供的一种动画类型，弥补帧动画和补间动画的不足。

原理：在一定时间间隔内，通过不断对值进行改变。不断将该值赋给对象属性，从而实现对象在该属性上的动画效果。核心类：ValueAnimation、ObjectAnimation(基于ValueAnimation)。

**使用动画的注意事项**

1. **内存耗尽**：使用**帧动画**的时候防止因为图片过多导致 OOM。
2. **View 动画并没有真正改变 View 的位置**：View 动画并没有真正改变 View 的属性，即 View 动画执行之后并未改变 View 的真实布局属性值。譬如我们在布局中有一个 Button 在屏幕上方，我们设置了平移动画移动到屏幕下方然后保持动画最后执行状态呆在屏幕下方，这时如果点击屏幕下方动画执行之后的 Button 是没有任何反应的，而点击原来屏幕上方没有 Button 的地方却响应的是点击 Button 的事件。
3. **内存泄漏**：使用属性动画的时候，当使用**无限循环动画**，需要在 **Activity 退出的时候停止**动画，不然可能会因为无法释放资源而导致 Activity 内存泄漏。
4. **使用 dp 而不是 px**：因为 px 在不同设备上面的兼容问题，使用动画的时候尽量使用 dp 作为单位。
5. **硬件加速**：使用硬件加速可以提升动画的流畅性。

1. **Handler、Loop消息队列模型，各部分作用。**

讲到消息队列，就涉及到MessageQueue消息队列、Message消息、Handler、Looper轮询器。

之所以需要Handler消息机制，因为我们开发中一般都会遇到需要加载网络、加载数据库等等的一些耗时操作，而我们的主线程是不可以被阻塞的，也就是不能够进行耗时操作。这时候我们就需要开启子线程完成耗时操作，完成后需要更新UI。这是候就用到了Handler消息机制。通过发送Message消息，将数据传递到主线程中让其更新UI。

Message：消息，用来在线程间传递数据的工具。携带少量数据，在线程间进行数据交互。.arg1、.arg2（int）、.obj、.what（obj）来传递数据。

MessageQueue：消息队列（单链表维护），用来存放handler发送过来的消息，并等待Looper轮询器轮询并处理。值得注意，消息队列是线程私有的。

Handler：用于来送和处理消息。发送消息一般为：handler.sendMessage()方法。发送的消息一般都会到达Handler的handleMessage（）方法中。

Looper：轮询器，是每个线程中MessageQueue的管家。调用looper.loop()之后，looper会一直检测一直轮询查看MessageQueue消息队列中是否存在消息，如果存在则取出消息，并传递到handleMessage（）方法中。

**Msg发送的时候，会将当前Handler设置为Target，等到Loop轮训到时调用target.dispatchHandleMag()方法。**

**如果在子线程Thread里边new Handler，使用子线程的handler收发message时，必须先Looper.preper(里边创建Looper并设置到ThreadLocal上，并且会初始化MagQueue)。**

**Thread -> Looper -> MessageQueue -> Handler**

**1 -> 1 -> 1 -> N**

1. **怎样退出终止App**
2. 容器式，在Application下创建一个容器（集合）来管理（存放）所有Activity，需要退出应用时，将容器内所有的Activity全部Finish掉即可。
3. 广播式，在BaseActivity中注册广播，需要退出应用时发出广播，Activity接收到后finish()掉。
4. 通过API杀死进程：*android.os.Process.killProcess(android.os.Process.myPid());*
5. Receiver + SingleTask 式，SplashActivity –> HomeActivity(SingleTask栈内唯一) +…。后期需要退出应用是只需打开HomeActivity并发送退出广播，应用就会退出。
6. **Assest目录和Res目录区别**

Res目录下的文件会被映射到R.java文件当中。后期我们需要用到资源文件就能够通过R文件访问。

Assest目录下的文件则不会映射到R.java文件当中。需要用到资源文件时需要通过AssestManager来访问。

1. **Android如何加速启动APP、Activity**
2. 避免在onCreate()方法上进行大量逻辑或耗时操作。因为执行onCreate方法时，界面还没有显示出来，屏幕处于黑屏状态。一旦**执行过多耗时操作，则黑屏时间过长**，降低用户体验。
3. A页面启动B页面时，不要在onPause方法上执行的耗时操作。因为B页面的生命周期必须要等到A页面的onPause方法执行后才执行。
4. 尽量减少在主线程的阻塞时间。耗时操作放在子线程上执行。
5. 优化布局文件，减少布局层次结构。布局层次结构过大消耗系统绘制性能。
6. 提高ListView、RecyclerView、Adapter的效率：复用converView、使用ViewHolder、缓存item数据、数据分段分页显示。
7. **Android内存优化方法：ListView优化、及时关闭资源、图片缓存…**
8. 图片内存的优化：

-- 在Bitmap对象上做处理，添加**软引用或弱引用**。

-- 使用**三级缓存**机制。图片加载进内存时使用**LRUCache “最近最少使用淘汰算法”** 的缓存机制来保存Bitmap对象。

-- 大图片需要进行**压缩处理**（BitmapFactory.Options（）表示要缩小原来的几分之一）

1. 其他的要注意：

-- 及时关闭资源对象，比如IO流数据库Cursor..

-- ListView、RecycleView复用converView、使用ViewHolder、缓存每个item图片、数据。。

1. **Android中弱引用与软引用的应用场景。**

-- 软引用：持有软引用的对象，一般jvm虚拟机都不会对他进行垃圾回收，除非系统内存严重不足时才对该对象进行回收。

-- 弱引用：仅仅持有弱引用的对象，jvm虚拟机都会在下次垃圾回收并扫描到该对象存在就会对其进行回收处理，不管当前系统内存是否足够。

-- 软引用我们一般会使用在图片缓存上面，Bitmap对象进行软引用处理，系统内存不足是就能够回收图片这部分占用的内存，从而避免内存溢出的发生。

-- 弱引用我们一般会使用在内部类对外部类的引用持有上。例如我们在Activity内部创建了Handler的内部类，内部类当中需要持有外部类引用，我们就可以对外部类的Activity进行弱引用处理，避免后期Activity销毁后无法进行内存回收的处境。

1. **Bitmap的4种属性，每种属性队形的大小**

-- 宽、高、单位像素所占用的字节数。

-- Bitmap对象用BitmapFactory进行创建。

-- 可以使用BitmapFactory.option()来对原图片进行压缩（传入options.inSampleSize）。

Options.inJustDecodeBounds = true，表示该方法只获取图片宽高等信息，不获取完整图片流。

然后针对Options .outWidth/height对自定义控件描绘。

**在Android 中图片有四种属性，分别是：**

**ALPHA\_8：每个像素占用1byte 内存**

**ARGB\_4444：每个像素占用2byte 内存**

**ARGB\_8888：每个像素占用4byte 内存（默认）**

**RGB\_565：每个像素占用2byte 内存(没有alpha 属性)**

1. **View与ViewGroup分类自定义View过程：onMeasure()、onLayout()、onDraw()**

自定义控件流程：

1. **自定义属性**
   1. 分析完成控件显示所需属性值。
   2. 在res/values/attrs.xml上定义属性声明。
   3. 在layout布局中使用
   4. 在View的构造函数中去获取
2. **重写方法**
   1. onMeasure() 测量自定义View宽高
   2. onLayout() 布局控件在屏幕中的位置
   3. onDraw() 绘制控件
3. **处理Touch触摸事件**
   1. onInterceptTouchEvent 拦截事件
   2. onTouchEvent() 处理用户触摸、滑动、点击事件。
4. View的状态保存与恢复
5. **Touch事件的分发机制**
   1. dispatchTouchEvent(MotionEvent ev) 分发事件
   2. onInterceptTouchEvent(MotionEvent ev) 拦截事件

--True -> 拦截调用onTouchEvent。

--Flase -> 向下传递事件调用下层dispatchTouchEvent()

* 1. onTouchEvent(MotionEvent ev) 处理事件

--True -> 消费事件，不再向上传递。

--Flase –> 不消费事件，事件继续向上传递。

1. **Android长连接，怎么处理心跳机制**

移动运营商为了减轻负荷，在客户端和服务器端一段时间没有通信时，就会中断通信服务。因此，客户端应用不得不定时发送一个心跳报文到服务器端，以确保连接的有效性 一般我们用来维护推送的长连接。（长连接比较耗电）

iOS推送：iOS的推送是通过系统来维护，也就是说iOS在系统级别上维护了一个客户端应用到苹果服务器的长连接。然后再将需要的推送的消息先推送到苹果服务器，再通过系统长连接推送到手机应用当中。

好处：

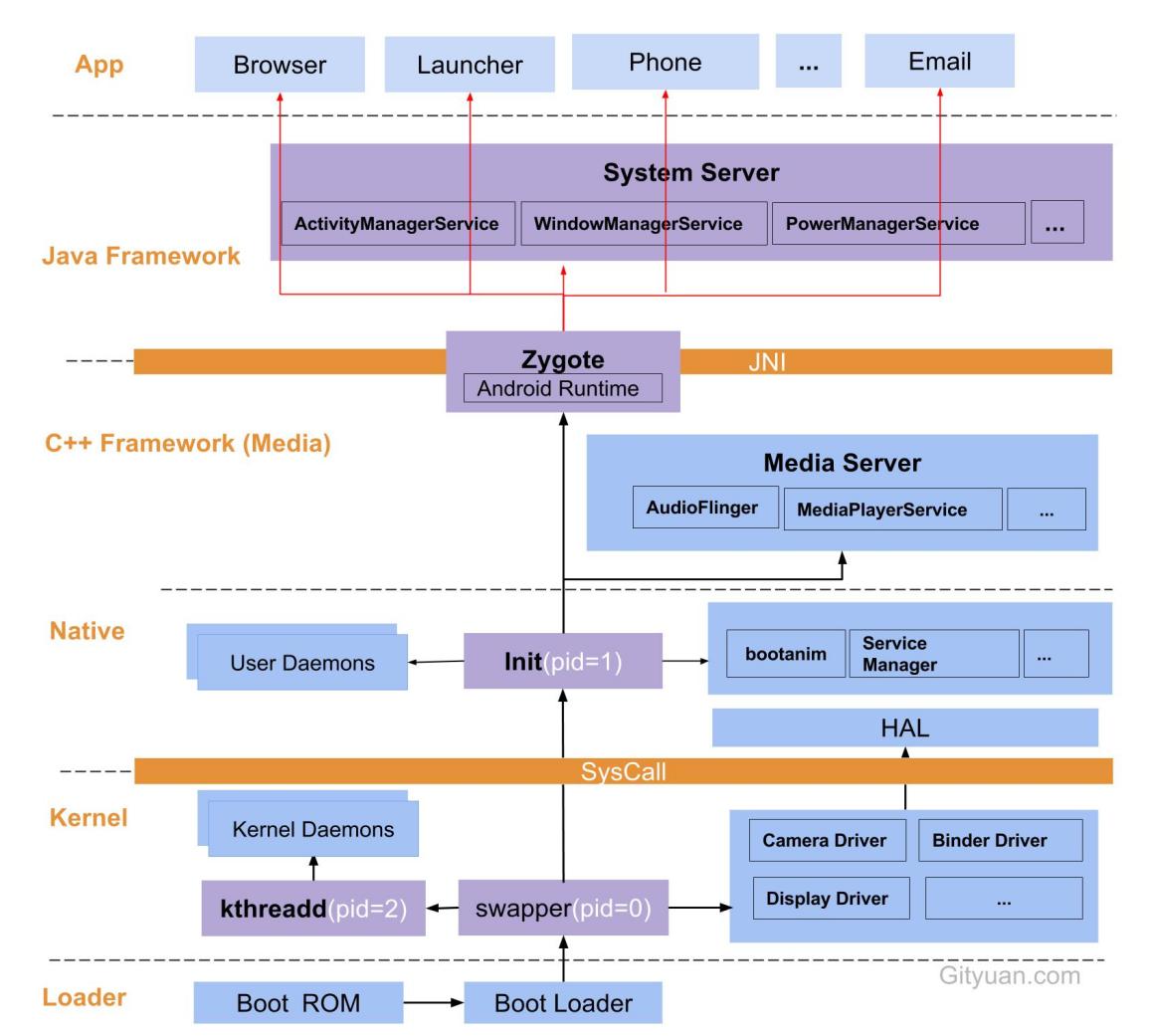
* + - 1. 稳定，因为长连接是系统进程，不会被杀死。
      2. 省电，系统只需维护一个长连接即可。不用每个应用都各自维护链接。
      3. 安全，只有苹果开发者才能够进行消息推送。

Android推送：android长连接是由各自应用来维护的。Google也有推出过推送框架，但在国内使用不了。而最近工信部也联合各大手机厂商（华为、ov、小米）和推送厂商（个推、极光）等等联合来推出一套android的统一推送服务。目的在于管理国内应用的消息推送问题。统一由终端厂商提供系统推送服务各厂商实现推送接口的统一。允许开发者接入，降低学习成本，提高用户体验。

1. **Zygote的启动过程**
   * + 1. Loader层：按下电源键开始，初始化硬件信息，将引导程序加载进内存，进入Linux Kernel层。
       2. Linux Kernel层：正式开始进入Android系统，创建系统第一个进程 –Swapper/idle进程。这是系统的第一个进程，用于初始化进程管理、内存管理和加载各种驱动。(并且Kernel还会创建一个系统内核进程Kthreadd), 进入Native层。
       3. Native层：Native层会孵化出init进程，该进程是用户空间守护进程，是所有用户进程都由init进程直接或间接的孵化而来。Init进程主要负责启动serviceManager（管理binder IPC通信的服务）、bootanim（开机动画）、等等的一些底层服务。并且孵化logd、installd、adbd等用户守护进程。 最最最重要的是，init进程通过解析init.rc脚本文件后孵化出了Zygote进程！-- Android系统的第一个Java虚拟机进程。

进入Framework层。

* + - 1. Framework层：在Zygote进程中加载JVM虚拟机，创建System Server，System Server负责启动和管理整个JavaFramework，包括ActivityManager、PowerManager等服务。
      2. APP层： Zygote创建的第一个应用层进程Launcher，就是我们开机后看到的桌面。Zygote还会创建浏览器、拨号、短信。。等等的一些系统应用。每个应用一般都会运行的单独的一个进程中。还有其他的应用进程都是由Zygote进程创建的！



**Launch的Pause程序 -> 唤醒Zygote -> 创建App进程 -> ActivityThread.main() -> attach() -> attachApplication -> HandleLaucherActivity() -> performLaucherActivity() -> onCreate..setContentView()..initDecorView()....onStart() -> handleResumeActivity() -> preformResumeActivity() -> ViewRoot..WindowManagerSession..doTravers -> decorView.makeVisiable()**

1. **Android IPC Binder原理（思路：4角色、举例App调用Server服务方法，详28.2）**

Binder是Android中的一种IPC跨进程通信的方式。

在我们应用开发中，Binder 一般是作为我们进程间通信的媒介。比如我们bindService的时候，Server端就会返回一个binder对象到客户端，客户端拿到在这个binder对象可以将他转为对应的业务binder Proxy对象，这样就能够获取Server端提供的方法或服务（方法、数据。。）

Binder是采用客户端-服务器的通信方式。Binder进程通信涉及到4个主要角色：Client、Server、ServerManager、Binder驱动。其中Client、Server、ServerManager是定义在用户空间上的，而Binder驱动则定义在内核空间中。

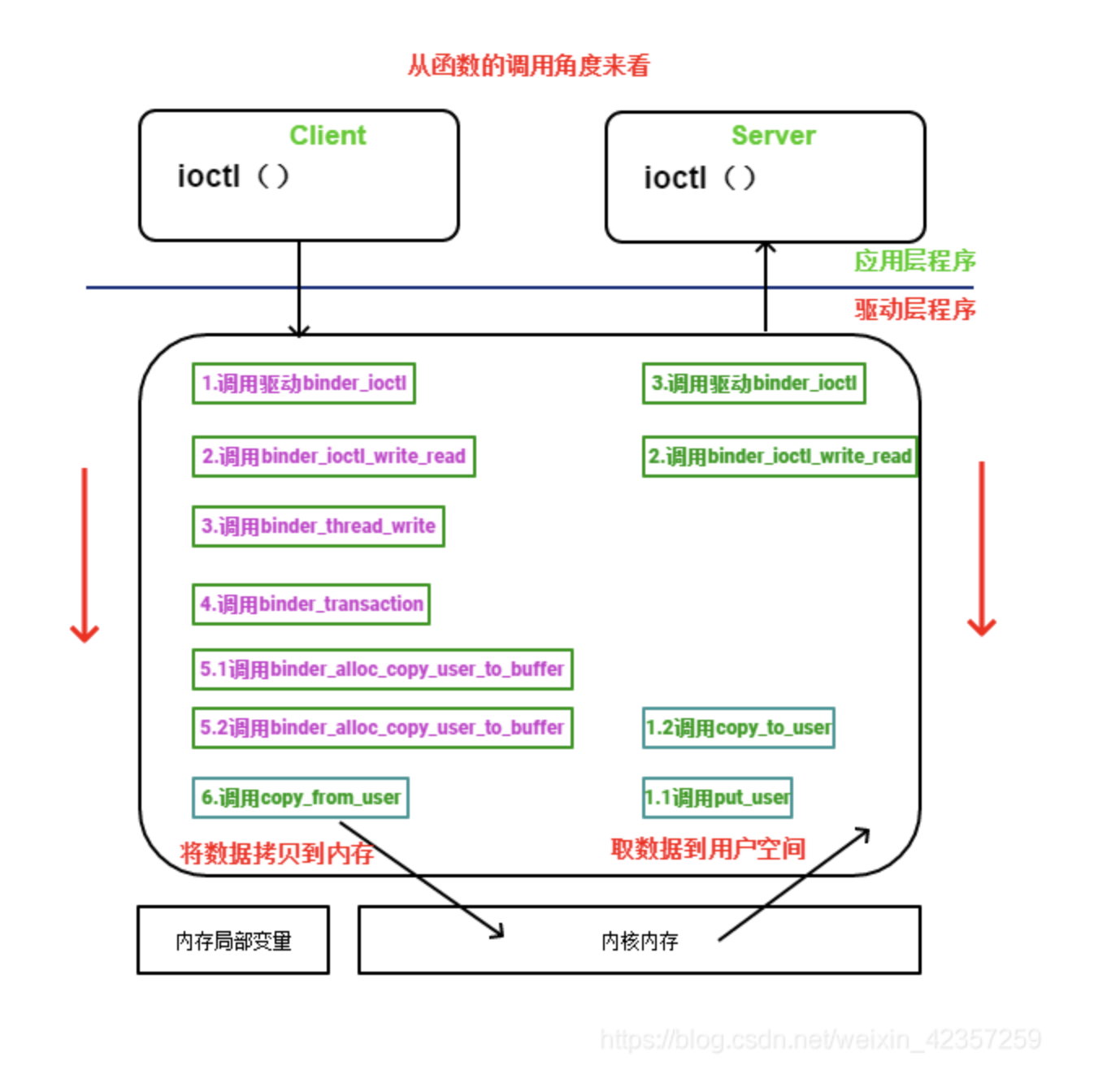
ServerManager（Native层c++）用于管理系统中的IPC服务。是Binder通信机制的大管家，用来给我们的Client、Server提供服务接口。Server进程要先在ServerManager上注册，然后Client向ServerManager获取Server服务。

Binder驱动：两个进程的进程空间有不同的虚拟地址映射规则，内存是不共享的，无法直接通信。这叫做“进程隔离”。Linux 把进程空间划分为用户空间和内核空间，分别运行用户程序和系统内核。用户空间和内核空间虽也是隔离的，但可以通过 copy\_from\_user 将数据从用户空间拷贝到内核空间，通过 copy\_to\_user 将数据从内核空间拷贝到用户空间（内核空间会先开辟一缓存块、直接映射到接收者进程上、实际只一次拷贝）。而 binder 使用了 mmap存储方案，只需调 copy\_from\_user 拷贝一次即可。（在用户空间的两进程不能共享内存，只能通过内核层共享io；区分实时内存和SD存储区别）。

-- Client、Server、ServerManager定义在用户空间；Binder驱动定义在内核空间。

--Binder驱动程序通过设备文件/dev/binder与用户空间进行通信。Client、Server、ServerManager通过open、ioctl() 文件操作函数与Binder驱动进行通信。

--Client与Server之间的通信是由Binder驱动间接完成的。



AIDL主要为了方便我们开发者实现Binder进程通信，系统为我们生成底层代码。

1. 用aidl语言定义需要调用的接口函数。
2. Server端实现这些方法。
3. Client端获取Server端Binder对象 并调用这些方法。

***实现了MyAIDL接口，需要明白两个点：***

1: 客户端调用的MyAIDL.Stub.asInterface(Binder对象).add(a, b), 实际会先判断server和client是否在同一个进程，如果在，就直接调用server的方法，此时不牵涉到Binder通信；如果不在同一个进程，就返回一个代理对象Proxy。

2: onTransact方法会根据底层Binder驱动传递过来的数据做相应的处理操作，其中包括调用server的方法。



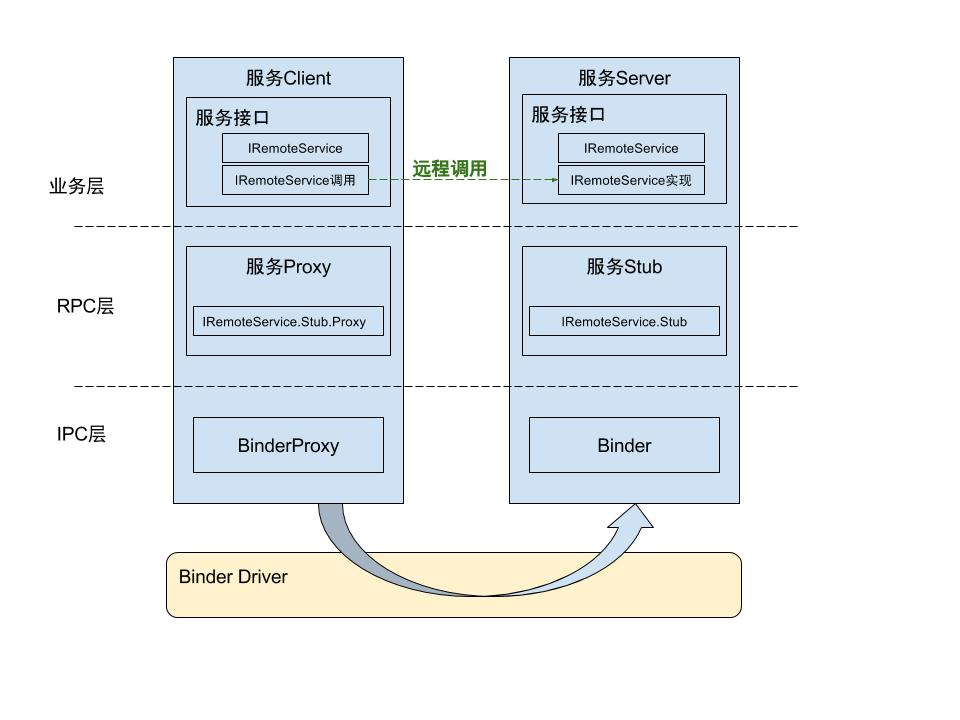
**Binder：应用层间的进程通信都用Binder进行通信。**

**Socket：Zygote进程与SystemServer进程(PowerMgS、WindowMgS..)的通信用Socket。一方面，因为ServerManager进程与Zygote进程都是由init进程创建的，无法保证Zygote进程创建完后ServerManager进程已创建并完成初始化动作；另一方面，因为Zygote创建线程是通过fork方式创建、fork不允许在多线程上操作、而Binder又是多线程处理，所以不用Binder，否则会存在死锁(相互等待资源)。**

**文件共享：**

**intent组件：基于Binder，最大传输数据为1MB（内核Binder缓存块为1MB、单进程共享），因此我们常说intent不适宜传递大数据。**

**IPC、LPC、RPC：进程间通信、本地通信、远程通信(Socket、接口)，RPC是IPC的一种。**



举个栗子：应用与系统服务通信



mWindowSession就是客户端拿到服务端的Binder Proxy代理对象，来调用系统服务。

同理，mWindow是客户端给服务端的Stub，服务端可以拿mWindow这个代理调用客户端的接口。

流程：Server实现具体的Binder并注册至ServiceManager -> Client通过ServiceManager拿到Server上Binder的Proxy对象 -> 在Client通过Proxy调用Server暴露出来的方法或服务

**28.2、ServerManager和SystemManager的区别**

ServerManager是Native层的进程，SystemManager是Framework层的进程。

ServerManager是Native服务，只有一个功能就是提供IPC服务，即为进程间提供Binder通信服务。

SystemManager是系统的服务集合，包括很多系统服务如PowerManagerS、WindowManagerS...

我们在App中调用相关Server服务方法，本质上就是通过ServerManager找到对应的注册过的Binder Proxy，最终在Server所在的Binder进程上执行相关方法，处理返回结果至我们App。-进程间通信。

1. **使用过什么框架、是否读过源码？底层实现原理**

<https://segmentfault.com/a/1190000005073746>

图片缓存：DiskLruCache

图片加载：Glide

图片处理：Glide-transformations

网络请求：

OkHttp -> 一个http与http/2的客户端

四大核心类：OkHttpClient、Request、Call 和 Response; 请求队列、连接池、拦截器（AppicationIntercept、RetryAndFllowIntercepter、BridgeIntercepter、CacheInterIntercepter）

Volley -> Google推出的Android异步网络请求框架和图片下载框架

Retrofit -> 类型安全的Http客户端

网络解析：

Gson -> Json和Java bean 的转化

Jackson -> Json、Xml、Java bean的相互转换。

数据库：

OrmLite

依赖注入：

ButterKnife

Dagger2

响应式编程：

RxJava

RxAndroid

RxBinding

性能优化：Studio上的Memeory Monitor

事件总线：EventBus

测试框架：Mockito(Java的Mocking单元测试)、Robotium( UI测试)

调试框架：Stetho

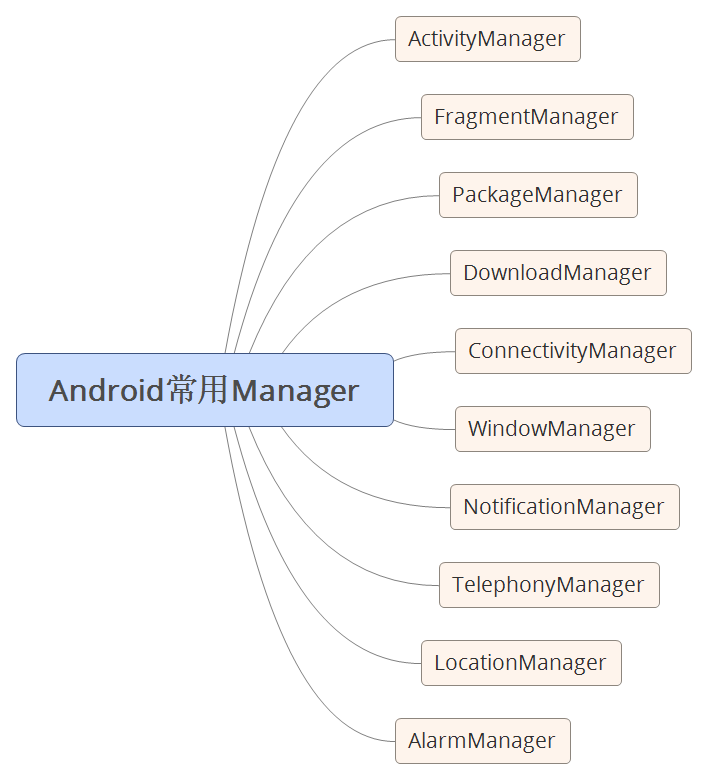
1. **如何实现一个网络框架(参考Volley)**

1.缓存队列,以url为key缓存内容可以参考Bitmap的处理方式，这里单独开启一个线程。

2.网络请求队列，使用线程池进行请求。

3.提供各种不同类型的返回值的解析如String，Json，图片等等。

1. **Android常用Manager**



1. **Android5.0、Android6.0新特性**

Android5.0：

--新增MaterialDesign设计风格

--支持64位ART虚拟机（新虚拟机，预编译技术，安装时就将字节码编译成机器码）

Android6.0：

--支持快速充电的切换

--曾强应用权限管理

--新增大量过度动画，用户体验更流畅。

Android7.0：

--支持分屏多任务

--增加VR支持

--新增Instant Apps（即时应用）功能

--支持Java8语言的新特性

Android8.0：

--画中画模式

--官方支持Kotlin编程语言。

--加强后台App管理

--值得注意的是，对于App的加速完全是基于系统层面，运用到的技术包括代码本地化、并发垃圾收集压缩等等，而App本身则没有任何变化。

-- Android O对于续航和硬件资源的优化，新系统通过更加智能的后台任务管理逻辑，限制了一些大量消耗资源的后台服务长期运行，借此在提升电池续航的同时也释放了更多内存资源。

33．理一下我们热修复的方案

* 修复有bug的类，生成dex补丁包；
* 通过反射机制得到PathClassLoader的成员你变量PathList字段(通过上面分析知道，PathList是PathClassLoader父类BaseDexClasLoader中的)
* 然后再反射PathList获取它的dexElements字段(是一个存放dex的Element数组)
* 将我们生成的dex补丁包，插入到dexElements的数组的最前端

**具体实现步骤**

1. 通过反射机制拿到”PathClassLoader”中的”PathList”对象
2. 通过反射机制拿到”PathList”对象中的”dexElements”数组
3. 通过”DexClassLoader” 加载我们的xxx.dex补丁包
4. 通过反射机制拿到”DexClassLoader”中的”PathList”对象
5. 通过反射机制拿到”PathList”对象中的”dexElements”数组
6. 将”DexClassLoader”的”dexElements”插入”PathClassLoader”的dexElements的前面

Ps：上面也说过，PathClassLoader和DexClassLoader均继承自BaseDexClassLoader,重要的方法都在BaseDexClassLoader中，包括”PathList”，所以我们只要反射BaseDexClassLoader就可以了。

安装补丁，就是通过**反射机制**实现，如果不熟悉反射机制，下面的代码可能会让你像坐过山车一样晕头转向。像了解反射机制的朋友可以看下我的上一篇文章[java/android中的反射机制](http://blog.csdn.net/u013366008/article/details/77833414" \t "_blank) 。

**你是如何自学Android**

首先是去图书馆或者上京东、亚马逊买些评价比较好的入门书去看。然后翻墙上Google看看Android 的官方文档跟Api。再上网找些视频看看，结合书本跟视频上的例子不断的去敲代码，加深印象。然后看别人的博客，看看被人对一些技术的看法和理解。平时有空做一些项目，向github提交代码，在看看别人的一些开源框架，开源项目。多学习学习。觉得自己基础掌握的不错之后，在去开始看进阶的书，像设计模式、别人项目框架、代码封装、第三方类库，以及看源码，看完源码学习到一些思想，然后再尝试着写点代码做点小东西，想想代码的提升。

1. [↑](#footnote-ref-0)