### 源码分析相关面试题

* [Volley源码分析](http://www.jianshu.com/p/ec3dc92df581)
* [注解框架实现原理](http://www.jianshu.com/p/20da6d6389e1)
* [okhttp3.0源码分析](http://www.jianshu.com/p/9ed2c2f2a52c)
* [onSaveInstanceState源码分析](http://www.jianshu.com/p/cbf9c3557d64)
* [静默安装和源码编译](http://www.jianshu.com/p/2211a5b3c37f)

### Activity相关面试题

* [保存Activity的状态](http://www.jianshu.com/p/cbf9c3557d64)

### 与XMPP相关面试题

* [XMPP协议优缺点](http://www.jianshu.com/p/2c04ac3c526a)
* [极光消息推送原理](http://www.jianshu.com/p/d88dc66908cf)

### 与性能优化相关面试题

* [内存泄漏和内存溢出区别](http://www.jianshu.com/p/5dd645b05c76)
* [UI优化和线程池实现原理](http://www.jianshu.com/p/c22398f8587f)
* [代码优化](http://www.jianshu.com/p/ebd41eab90df)
* [内存性能分析](http://www.jianshu.com/p/2665c31b9c2f)
* [内存泄漏检测](http://www.jianshu.com/p/1514c7804a06)
* [App启动优化](http://www.jianshu.com/p/f0f73fefdd43)
* [与IPC机制相关面试题](http://www.jianshu.com/p/de4793a4c2d0)

### 与登录相关面试题

* [oauth认证协议原理](http://www.jianshu.com/p/2a6ecbf8d49d)
* [token产生的意义](http://www.jianshu.com/p/9b7ce2d6c195)
* [微信扫一扫实现原理](http://www.jianshu.com/p/a9d1f21bd5e0)

### 与开发相关面试题

* [迭代开发的时候如何向前兼容新旧接口](http://www.jianshu.com/p/cbecadec98de)
* [手把手教你如何解决as jar包冲突](http://www.jianshu.com/p/30fdc391289c)
* [context的原理分析](http://www.jianshu.com/p/2706c13a1769)
* [解决ViewPager.setCurrentItem中间很多页面切换方案](http://www.jianshu.com/p/38ab6d856b56)

### 与人事相关面试题

* [人事面试宝典](http://www.jianshu.com/p/d61b553ff8c9)

### 本文配套视频

* [配套视频](https://v.qq.com/x/page/v0396aro8d1.html)

## Android性能优化之App应用启动分析与优化

### App启动方式

通常来说, 一个App启动也会分如下二中不同的状态:

.1）冷启动

当启动应用时，后台没有该应用的进程，这时系统会重新创建一个新的进程分配给该应用，这个启动方式就是冷启动。冷启动因为系统会重新创建一个新的进程分配给它，所以会先创建和初始化Application类，再创建和初始化MainActivity类（包括一系列的测量、布局、绘制），最后显示在界面上。

2.）热启动

当启动应用时，后台已有该应用的进程（例：按back键、home键，应用虽然会退出，但是该应用的进程是依然会保留在后台，可进入任务列表查看），所以在已有进程的情况下，这种启动会从已有的进程中来启动应用，这个方式叫热启动。热启动因为会从已有的进程中来启动，所以热启动就不会走Application这步了，而是直接走MainActivity（包括一系列的测量、布局、绘制），所以热启动的过程只需要创建和初始化一个MainActivity就行了，而不必创建和初始化Application，因为一个应用从新进程的创建到进程的销毁，Application只会初始化一次。

### 启动时间的测量

关于Activity启动时间的定义

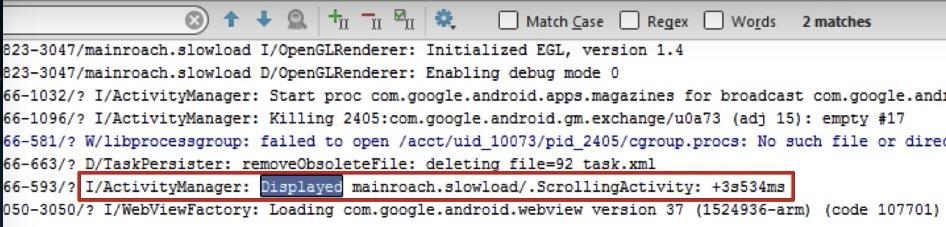
对于Activity来说，启动时，首先执行的是onCreate()、onStart()、onResume()这些生命周期函数，但即使这些生命周期方法回调结束了，应用也不算已经完全启动，还需要等View树全部构建完毕，一般认为，setContentView中的View全部显示结束了，算作是应用完全启动了。

Display Time

从API19之后，Android在系统Log中增加了Display的Log信息，通过过滤ActivityManager以及Display这两个关键字，可以找到系统中的这个Log：

$ adb logcat | grep “ActivityManager”  
ActivityManager: Displayed com.example.launcher/.LauncherActivity: +999ms

抓到的Log如图所示：



img

那么这个时间，实际上是Activity启动，到Layout全部显示的过程，但是要注意，这里并不包括数据的加载，因为很多App在加载时会使用懒加载模式，即数据拉取后，再刷新默认的UI。

### 基于上面的启动流程我们尽量做到如下几点

1. Application的创建过程中尽量少的进行耗时操作
2. 首屏Activity的渲染

当前冷启动效果:



img

#### Application

Application是程序的主入口，特别是很多第三方SDK都会需要在Application的onCreate里面做很多初始化操作，一般来说我们可以将这些初始化放在一个单独的线程中处理, 为了方便今后管理,

优化的方法，无非是通过以下几个方面：

1）异步初始化 2）后台任务 3）界面预加载

异步初始化

这个很简单，就是让App在onCreate里面尽可能的少做事情，而利用手机的多核特性，尽可能的利用多线程，例如一些第三方框架的初始化，如果能放线程，就尽量的放入线程中，最简单的，你可以直接new Thread()，当然，你也可以通过公共的线程池来进行异步的初始化工作，这个是最能够压缩启动时间的方式

后台任务

因为这个App一般会集成很多三方SDK等服务, 所以Application的onCreate有很多第三方平台的初始化工作…

Application代码如下：

public class MyApp extends Application {  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 LitePal.initialize(this.getApplicationContext());  
 }  
}

使用IntentService不同于Service, 它是工作在后台线程的.

IntentService代码如下：

public class InitializeService extends IntentService {  
 private static final String ACTION\_INIT\_WHEN\_APP\_CREATE = "com.maweiqi";  
 public InitializeService(String name) {  
 super(name);  
 }  
  
 @Override  
 protected void onHandleIntent(Intent intent) {  
 if (intent != null) {  
 final String action = intent.getAction();  
 if (ACTION\_INIT\_WHEN\_APP\_CREATE.equals(action)) {  
 performInit();  
 }  
 }  
 }  
 private void performInit() {  
 LitePal.initialize(this.getApplicationContext());  
 }  
 public static void start(Context context) {  
 Intent intent = new Intent(context, InitializeService.class);  
 intent.setAction(ACTION\_INIT\_WHEN\_APP\_CREATE);  
 context.startService(intent);  
 }  
}

MyApp的onCreate改成:

public class MyApp extends Application {  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 InitializeService.start(this);  
 }  
}

最终的效果



img

界面预加载

当系统加载一个Activity的时候，onCreate()是一个耗时过程，那么在这个过程中，系统为了让用户能有一个比较好的体验，实际上会先绘制一些初始界面，类似于PlaceHolder。

系统首先会读取当前Activity的Theme，然后根据Theme中的配置来绘制，当Activity加载完毕后，才会替换为真正的界面。所以，Google官方提供的解决方案，就是通过android:windowBackground属性，来进行加载前的配置，同时，这里不仅可以配置颜色，还能配置图片，例如，我们可以使用一个layer-list来作为android:windowBackground要显示的图：

在drawable文件夹下面 , 做一个logo\_splash的背景: start\_window.xml

<?xml version="1.0" encoding="utf-8"?>  
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">  
 <!-- 底层白色 -->  
 <item android:drawable="@color/white" />  
  
 <!-- 顶层Logo居中 -->  
 <item>  
 <bitmap  
 android:gravity="center"  
 android:src="@drawable/ic\_github" />  
 </item>  
</layer-list>

在style里面弄一个主题:

<style name="StartStyle" parent="AppTheme">  
 <item name="android:windowBackground">@drawable/start\_window</item>  
 </style>

在清单文件里面给Activity指定需要预加载的Style：

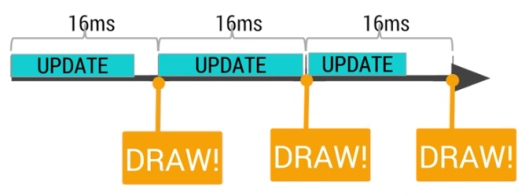
<activity android:name=".MainActivity" android:theme="@style/StartStyle">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
  
 <category android:name="android.intent.category.LAUNCHER"/>  
 </intent-filter>  
 </activity>

activity代码如下：

public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
  
 setTheme(R.style.AppTheme);  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 }  
}

#### 首屏Activity的渲染

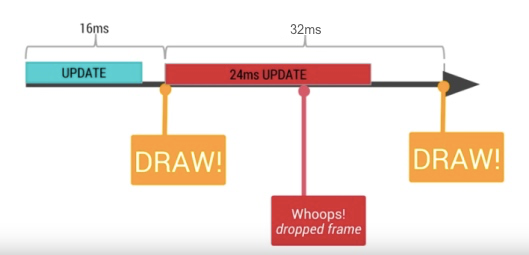
Android系统每隔16ms会发出VSYNC信号重绘我们的界面(Activity). 为什么是16ms, 因为Android设定的刷新率是60FPS(Frame Per Second), 也就是每秒60帧的刷新率, 约合16ms刷新一次.就像是这样的:



img

这就意味着, 我们需要在16ms内完成下一次要刷新的界面的相关运算, 以便界面刷新更新. 然而, 如果我们无法在16ms内完成此次运算会怎样呢?

例如, 假设我们更新屏幕的背景图片, 需要24ms来做这次运算. 当系统在第一个16ms时刷新界面, 然而我们的运算还没有结束, 无法绘出图片. 当系统隔16ms再发一次VSYNC信息重绘界面时, 用户才会看到更新后的图片. 也就是说用户是32ms后看到了这次刷新(注意, 并不是24ms). 这就是传说中的丢帧(dropped frame):



img

丢帧给用户的感觉就是卡顿, 而且如果运算过于复杂, 丢帧会更多, 导致界面常常处于停滞状态, 卡到爆.

* 欢迎关注微信公众号,长期推荐技术文章和技术视频

微信公众号名称：Android干货程序员

img

img