



一种提高Android应用进程存活率新方法

📅 2016-06-19 📁 TECH > ANDROID 🏷️ #ANDROID, #存活率, #进程 📄 11704 ARTICLE.HITS

👤 SKYSERAPH

基础知识

Android 进程优先级

1 进程优先级等级一般分法

- Active process
- Visible Process
- Service process
- Background process
- Empty process

2 Service技巧

- onStartCommand返回START_STICKY
- onDestroy中startSelf
- Service后台变前置, setForeground(true)
- android:persistent = "true"

3 进程优先级号

ProcessList.java

```
1 // Adjustment used in certain places where we don't know it yet.
2 // (Generally this is something that is going to be cached, but we
3 // don't know the exact value in the cached range to assign yet.)
4 static final int UNKNOWN_ADJ = 16;
```

Catalogue

1. 基础知识

- 1.1. Android 进程优先级
- 1.2. Android Low Memory Killer
- 1.3. 查看某个App的进程
- 1.4. Linux AM命令
- 1.5. NotificationListenerService
- 1.6. Android账号和同步机制
- 1.7. Android多进程

2. 现有方法

- 2.1. 网络连接保活方法
- 2.2. 双service(通知栏) 提高进程优先级
- 2.3. Service及时拉起
- 2.4. 守护进程/进程互拉
- 2.5. Linux Am命令开启后台进程
- 2.6. NotificationListenerService通知
- 2.7. 前台浮窗

3. 新方法(AccountSync)

- 3.1. 思路
- 3.2. 效果
- 3.3. 风险
- 3.4. 实现 (核心代码)

4. Refs

5. 后记

```
5 // This is a process only hosting activities that are not visible,
6 // so it can be killed without any disruption.
7 static final int CACHED_APP_MAX_ADJ = 15;
8 static final int CACHED_APP_MIN_ADJ = 9;
9 // The B list of SERVICE_ADJ -- these are the old and decrepit
10 // services that aren't as shiny and interesting as the ones in the A
11 static final int SERVICE_B_ADJ = 8;
12 // This is the process of the previous application that the user was
13 // This process is kept above other things, because it is very common
14 // switch back to the previous app. This is important both for recen
15 // task switch (toggling between the two top recent apps) as well as
16 // UI flow such as clicking on a URI in the e-mail app to view in the
17 // and then pressing back to return to e-mail.
18 static final int PREVIOUS_APP_ADJ = 7;
19 // This is a process holding the home application -- we want to try
20 // avoiding killing it, even if it would normally be in the backgroun
21 // because the user interacts with it so much.
22 static final int HOME_APP_ADJ = 6;
23 // This is a process holding an application service -- killing it wil
24 // have much of an impact as far as the user is concerned.
25 static final int SERVICE_ADJ = 5;
26 // This is a process with a heavy-weight application. It is in the
27 // background, but we want to try to avoid killing it. Value set in
28 // system/rootdir/init.rc on startup.
29 static final int HEAVY_WEIGHT_APP_ADJ = 4;
30 // This is a process currently hosting a backup operation. Killing i
31 // is not entirely fatal but is generally a bad idea.
32 static final int BACKUP_APP_ADJ = 3;
33 // This is a process only hosting components that are perceptible to
34 // user, and we really want to avoid killing them, but they are not
35 // immediately visible. An example is background music playback.
36 static final int PERCEPTIBLE_APP_ADJ = 2;
37 // This is a process only hosting activities that are visible to the
38 // user, so we'd prefer they don't disappear.
39 static final int VISIBLE_APP_ADJ = 1;
40 // This is the process running the current foreground app. We'd real
41 // rather not kill it!
42 static final int FOREGROUND_APP_ADJ = 0;
43 // This is a process that the system or a persistent process has boun
44 // and indicated it is important.
45 static final int PERSISTENT_SERVICE_ADJ = -11;
46 // This is a system persistent process, such as telephony. Definitel
47 // don't want to kill it, but doing so is not completely fatal.
48 static final int PERSISTENT_PROC_ADJ = -12;
49 // The system process runs at the default adjustment.
50 static final int SYSTEM_ADJ = -16;
51 // Special code for native processes that are not being managed by th
```

```
52 // don't have an oom adj assigned by the system).
53 static final int NATIVE_ADJ = -17;
54
55
56
57
58
59
60
61
62
63
64
65
66
67
```

Android Low Memory Killer

Android系统内存不足时，系统会杀掉一部分进程以释放空间，谁生谁死的这个生死大权就是由LMK所决定的，这就是Android系统中的Low Memory Killer，其基于Linux的OOM机制，其阈值定义如下面所示的lowmemorykiller文件中，当然也可以通过系统的init.rc实现自定义。

lowmemorykiller.c

```
1 static uint32_t lowmem_debug_level = 1;
2 static int lowmem_adj[6] = {
3     0,
4     1,
5     6,
6     12,
7 };
8 static int lowmem_adj_size = 4;
9 static int lowmem_minfree[6] = {
10     3 * 512, /* 6MB */
11     2 * 1024, /* 8MB */
12     4 * 1024, /* 16MB */
13     16 * 1024, /* 64MB */
14 };
15 static int lowmem_minfree_size = 4;
```

① 在Low Memory Killer中通过进程的oom_adj与占用内存的大小决定要杀死的进程，oom_adj值越小越不容易被杀死。其中，lowmem_minfree是杀进程的时机，谁被杀，则取决于lowmem_adj，具体值

得含义参考上面 *Android 进程优先级* 所述。

② 在init.rc中定义了init进程（系统进程）的oom_adj为-16，其不可能被杀死（init的PID是1），而前台进程是0（这里的前台进程是指用户正在使用的Activity所在的进程），用户按Home键回到桌面时的优先级是6，普通的Service的进程是8。

init.rc

```
1  # Set init and its forked children's oom_adj.
2      write /proc/1/oom_adj -16
```

关于Low Memory Killer的具体实现原理可参考Ref-2。

查看某个App的进程

步骤（手机与PC连接）

1. adb shell
2. ps | grep 进程名
3. cat /proc/pid/oom_adj //其中pid是上述grep得到的进程号



```
Windows PowerShell
版权所有 (C) 2009 Microsoft Corporation。保留所有权利。

PS C:\Users\Administrator> adb shell
shell@virgo:/ $
shell@virgo:/ $ ps | grep cn.
u0_a501 10291 347 885516 83540 sys_epoll_ 00000000 $ cn.
shell@virgo:/ $
shell@virgo:/ $ cat /proc/10291/oom_adj
0
shell@virgo:/ $
```

Linux AM命令

am命令：在Android系统中通过adb shell 启动某个Activity、Service、拨打电话、启动浏览器等操作Android的命令。其源码在Am.java中，在shell环境下执行am命令实际是启动一个线程执行Am.java中的主函数（main方法），am命令后跟的参数都会当做运行时参数传递到主函数中，主要实现在Am.java的run方法中。

拨打电话

命令：am start -a android.intent.action.CALL -d tel:电话号码

示例：am start -a android.intent.action.CALL -d tel:10086

打开一个网页

命令：am start -a android.intent.action.VIEW -d 网址

示例：am start -a android.intent.action.VIEW -d <http://www.skyseraph.com>

启动一个服务

命令：am startservice <服务名称>

示例：am startservice -n com.android.music/ com.android.music.MediaPlaybackService

NotificationListenerService

“A service that receives calls from the system when new notifications are posted or removed, or their ranking changed.” From Google

用来监听到通知的发送以及移除和排名位置变化,如果我们注册了这个服务,当系统任何一条通知到来或者被移除掉,我们都能通过这个service来监听到,甚至可以做一些管理工作。

Android账号和同步机制

属于Android中较偏冷的知识,具体参考 Ref 3 /4 /5

Android多进程

- **实现**：android:process
- **好处**：一个独立的进程可以充分利用自己的RAM预算,使其主进程拥有更多的空间处理资源。此外,操作系统对待运行在不同组件中的进程是不一样的。这意味着,当系统运行在低可用内存的条件时,并不是所有的进程都会被杀死
- **大坑**：每一个进程将有自己的Dalvik VM实例,意味着你不能通过这些实例共享数据,至少不是传统意义上的。例如,静态字段在每个进程都有自己的值,而不是你倾向于相信的只有一个值。
- 更多详细请参考Ref 9

现有方法

网络连接保活方法

A. GCM

B. 公共的第三方push通道(信鸽等)

C. 自身跟服务器通过轮询,或者长连接

具体实现请参考 微信架构师杨干荣的“微信Android客户端后台保活经验分享”(Ref-1)。

双service(通知栏) 提高进程优先级

思路：(API level > 18)

- 应用启动时启动一个假的Service (FakeService), startForeground(), 传一个空的Notification
- 启动真正的Service (AlwaysLiveService), startForeground(), 注意必须相同Notification ID
- FakeService stopForeground()

效果：通过adb查看,运行在后台的服务其进程号变成了1（优先级仅次于前台进程）

风险：Android系统前台service的一个漏洞，可能在6.0以上系统中修复

实现：核心代码如下

- AlwaysLiveService 常驻内存服务

```
1  @Override
2      public int onStartCommand(Intent intent, int flags, int startId) {
3          startForeground(R.id.notify, new Notification());
4          startService(new Intent(this, FakeService.class));
5          return super.onStartCommand(intent, flags, startId);
6      }
```

- FakeService 临时服务

```
1
2  public class FakeService extends Service {
3      @Nullable
4      @Override
5      public IBinder onBind(Intent intent) {
6          return null;
7      }
8      @Override
9      public int onStartCommand(Intent intent, int flags, int startId)
10         startForeground(R.id.notify, new Notification());
11         stopSelf();
12         return super.onStartCommand(intent, flags, startId);
13     }
14     @Override
15     public void onDestroy() {
16         stopForeground(true);
17         super.onDestroy();
18     }
19 }
20
```

Service及时拉起

AlarmReceiver , ConnectReceiver , BootReceiver等

- Service设置（见上面基础部分）
- 通过监听系统广播，如开机，锁屏，亮屏等重新启动服务
- 通过alarm定时器，启动服务

守护进程/进程互拉

在分析360手机助手app时，发现其拥有N多个进程，一个进程kill后会被其它未kill的进程拉起，这也是一种思路吧，虽然有点流氓~

守护进程一般有这样两种方式：

- 多个java进程守护互拉
- 底层C守护进程拉起App上层/java进程

Linux Am命令开启后台进程

一种底层实现让进程不被杀死的方法，在Android4.4以上可能有兼容性问题，具体参考Ref-7

NotificationListenerService通知

一种需要用户允许特定权限的系统拉起方式，4.3以上系统

前台浮窗

有朋友提出一种应用退出后启动一个不可交互的浮窗，个人觉得这种方法是无效的，读者有兴趣可以一试，参考[Pixel-Activity-Keep-Alive](#)

新方法(AccountSync)

思路

利用Android系统提供的账号和同步机制实现

效果

- 通过adb查看,运行在后台的服务其进程号变成了1（优先级仅次于前台进程），能提高进程优先级，对比如下图

```

PS C:\Users\Administrator> adb shell
shell@virgo:/ $
shell@virgo:/ $ ps | grep cn.r.igamanager:core
u0_a502  9751  347   849848 79412 sys_epoll_ 00000000 $ cn.r.igamanager:core
shell@virgo:/ $
shell@virgo:/ $ cat /proc/9751/oom_adj
0
shell@virgo:/ $
shell@virgo:/ $ cat /proc/9751/oom_adj
4
shell@virgo:/ $

```

应用处于前台

按back键退出应用

正常情况

```

PS C:\Users\Administrator> adb shell
shell@virgo:/ $ ps | grep cn.r.igamanager:core
u0_a501  10291 347   885156 80496 sys_epoll_ 00000000 $ cn.r.igamanager:core
shell@virgo:/ $
shell@virgo:/ $
shell@virgo:/ $ cat /proc/10291/oom_adj
0
shell@virgo:/ $ cat /proc/10291/oom_adj
0
shell@virgo:/ $ cat /proc/10291/oom_adj
1
shell@virgo:/ $

```

应用处于前台

按back键退出应用

采用AccountSyncAdapter方法后

- 进程被系统kill后，可以由syn拉起

风险

- SyncAdapter时间进度不高，往往会因为手机处于休眠状态，而时间往后调整，同步间隔最低为1分钟
- 用户可以单独停止或者删除，有些手机账号默认是不同步的，需要手动开启

实现 (核心代码)

“

1 建立数据同步系统 (ContentProvider)

通过一个ContentProvider用来作数据同步，由于并没有实际数据同步，所以此处就直接建立一个空的ContentProvider即可

```

1 public class XXAccountProvider extends ContentProvider {
2     public static final String AUTHORITY = "包名.provider";
3     public static final String CONTENT_URI_BASE = "content://" + AUTHORITY;
4     public static final String TABLE_NAME = "data";
5     public static final Uri CONTENT_URI = Uri.parse(CONTENT_URI_BASE);
6     @Override
7     public boolean onCreate() {
8         return true;
9     }
10 }

```



```
9      }
10     @Nullable
11     @Override
12     public Cursor query(Uri uri, String[] projection, String selectio
13                        String[] selectionArgs, String sortOrder) {
14         return null;
15     }
16     @Nullable
17     @Override
18     public String getType(Uri uri) {
19         return new String();
20     }
21     @Nullable
22     @Override
23     public Uri insert(Uri uri, ContentValues values) {
24         return null;
25     }
26     @Override
27     public int delete(Uri uri, String selection, String[] selectionAr
28         return 0;
29     }
30     @Override
31     public int update(Uri uri, ContentValues values, String selection
32         return 0;
33     }
34 }
35
36
37
38
39
40
```

然后再Manifest中声明

```
1 <provider
2     android:name="**.XXAccountProvider"
3     android:authorities="@string/account_auth_provider"
4     android:exported="false"
5     android:syncable="true"/>
```



2 建立Sync系统 (SyncAdapter)

通过实现SyncAdapter这个系统服务后, 利用系统的定时器对程序数据ContentProvider进行更新, 具体步骤为:

- 创建Sync服务

```

1
2  public class XXSyncService extends Service {
3      private static final Object sSyncAdapterLock = new Object();
4      private static XXSyncAdapter sSyncAdapter = null;
5      @Override
6      public void onCreate() {
7          synchronized (sSyncAdapterLock) {
8              if (sSyncAdapter == null) {
9                  sSyncAdapter = new XXSyncAdapter(getApplicationContext
10             }
11         }
12     }
13     @Override
14     public IBinder onBind(Intent intent) {
15         return sSyncAdapter.getSyncAdapterBinder();
16     }
17     static class XXSyncAdapter extends AbstractThreadedSyncAdapter {
18         public XXSyncAdapter(Context context, boolean autoInitialize)
19             super(context, autoInitialize);
20     }
21     @Override
22     public void onPerformSync(Account account, Bundle extras, Str
23         getContext().getContentResolver().notifyChange(XXAccountP
24     }
25 }
26 }
27

```

- 声明Sync服务

```

1  <service
2      android:name="*.XXSyncService"
3      android:exported="true"

```

```

4         android:process=":core">
5         <intent-filter>
6             <action
7                 android:name="android.content.SyncAdapter"/>
8         </intent-filter>
9         <meta-data
10             android:name="android.content.SyncAdapter"
11             android:resource="@xml/sync_adapter"/>
12     </service>

```

其中sync_adapter为：

```

1     <sync-adaptor xmlns:android="http://schemas.android.com/apk/res/androi
2         android:accountType="@string/account_auth_type"
3         android:allowParallelSyncs="false"
4         android:contentAuthority="@string/account_auth_provide"
5         android:isAlwaysSyncable="true"
6         android:supportsUploading="false"
7         android:userVisible="true"/>

```

参数说明：

android:contentAuthority 指定要同步的ContentProvider在其AndroidManifest.xml文件中有个android:authorities属性。

android:accountType 表示进行同步的账号的类型。

android:userVisible 设置是否在“设置”中显示

android:supportsUploading 设置是否必须notifyChange通知才能同步

android:allowParallelSyncs 是否支持多账号同时同步

android:isAlwaysSyncable 设置所有账号的isSyncable为1

android:syncAdapterSettingsAction 指定一个可以设置同步的activity的Action。

- 账户调用Sync服务

首先配置好Account（第三步），然后再通过ContentProvider实现手动更新

```

1     public void triggerRefresh() {
2         Bundle b = new Bundle();
3         b.putBoolean(ContentResolver.SYNC_EXTRAS_MANUAL, true);
4         b.putBoolean(ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
5         ContentResolver.requestSync(
6             account,

```

```
7             CONTENT_AUTHORITY,  
8             b);  
9     }
```

添加账号

```
1 Account account = AccountService.GetAccount();  
2 AccountManager accountManager = (AccountManager) context.getSystemService  
3 accountManager.addAccountExplicitly(...)
```



同步周期设置

```
1 ContentResolver.setIsSyncable(account, CONTENT_AUTHORITY, 1);  
2 ContentResolver.setSyncAutomatically(account, CONTENT_AUTHORITY, true)  
3 ContentResolver.addPeriodicSync(account, CONTENT_AUTHORITY, new Bundle
```



“

3 建立账号系统 (Account Authenticator)

通过建立Account账号，并关联SyncAdapter服务实现同步

- 创建Account服务

```
1 public class XXAuthService extends Service {  
2     private XXAuthenticator mAuthenticator;  
3     @Override  
4     public void onCreate() {  
5         mAuthenticator = new XXAuthenticator(this);  
6     }  
7     private XXAuthenticator getAuthenticator() {  
8         if (mAuthenticator == null)  
9             mAuthenticator = new XXAuthenticator(this);  
10        return mAuthenticator;  
11    }  
12    @Override  
13    public IBinder onBind(Intent intent) {
```

```
14         return getAuthenticator().getIBinder();
15     }
16     class XXAuthenticator extends AbstractAccountAuthenticator {
17         private final Context context;
18         private AccountManager accountManager;
19         public XXAuthenticator(Context context) {
20             super(context);
21             this.context = context;
22             accountManager = AccountManager.get(context);
23         }
24         @Override
25         public Bundle addAccount(AccountAuthenticatorResponse respons
26             throws NetworkErrorException {
27             // 添加账号 示例代码
28             final Bundle bundle = new Bundle();
29             final Intent intent = new Intent(context, AuthActivity.cl
30             intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_
31             bundle.putParcelable(AccountManager.KEY_INTENT, intent);
32             return bundle;
33         }
34         @Override
35         public Bundle getAuthToken(AccountAuthenticatorResponse respo
36             throws NetworkErrorException {
37             // 认证 示例代码
38             String authToken = accountManager.peekAuthToken(account,
39             //if not, might be expired, register again
40             if (TextUtils.isEmpty(authToken)) {
41                 final String password = accountManager.getPassword(ac
42                 if (password != null) {
43                     //get new token
44                     authToken = account.name + pa
45                 }
46             }
47             //without password, need to sign again
48             final Bundle bundle = new Bundle();
49             if (!TextUtils.isEmpty(authToken)) {
50                 bundle.putString(AccountManager.KEY_ACCOUNT_NAME, acc
51                 bundle.putString(AccountManager.KEY_ACCOUNT_TYPE, acc
52                 bundle.putString(AccountManager.KEY_AUTHTOKEN, authTo
53                 return bundle;
54             }
55             //no account data at all, need to do a sign
56             final Intent intent = new Intent(context, AuthActivity.cl
57             intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_
58             intent.putExtra(AuthActivity.ARG_ACCOUNT_NAME, account.na
59             bundle.putParcelable(AccountManager.KEY_INTENT, intent);
60             return bundle;
```

```
61         }
62         @Override
63         public String getAuthTokenLabel(String authTokenType) {
64             //         throw new UnsupportedOperationException();
65             return null;
66         }
67         @Override
68         public Bundle editProperties(AccountAuthenticatorResponse res) {
69             return null;
70         }
71         @Override
72         public Bundle confirmCredentials(AccountAuthenticatorResponse res) {
73             throws NetworkErrorException {
74             return null;
75         }
76         @Override
77         public Bundle updateCredentials(AccountAuthenticatorResponse res) {
78             throws NetworkErrorException {
79             return null;
80         }
81         @Override
82         public Bundle hasFeatures(AccountAuthenticatorResponse res) {
83             throws NetworkErrorException {
84             return null;
85         }
86     }
87 }
88
89
90
91
92
93
94
95
96
97
98
99
```

- 声明Account服务

```
1     <service
2         android:name="*.XXAuthService"
```

```
3         android:exported="true"
4         android:process=":core">
5         <intent-filter>
6             <action
7                 android:name="android.accounts.AccountAuthent
8             </intent-filter>
9         <meta-data
10             android:name="android.accounts.AccountAuthenticator"
11             android:resource="@xml/authenticator"/>
12     </service>
```

其中authenticator为：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <account-authenticator xmlns:android="http://schemas.android.com/apk/r
3      android:accountType="@string/account_auth_type"
4      android:icon="@drawable/icon"
5      android:smallIcon="@drawable/icon"
6      android:label="@string/app_name"
7  />
```

- 使用Account服务
同SyncAdapter，通过AccountManager使用
 - 申请Token主要是通过 [AccountManager.getAuthToken\(\)](#)系列方法
 - 添加账号则通过 [AccountManager.addAccount\(\)](#)
 - 查看是否存在账号通过 [AccountManager.getAccountsByType\(\)](#)

Refs

1. [微信Android客户端后台保活经验分享](#)
2. [Android Low Memory Killer原理](#)
3. [stackOverflow 上介绍的双Service方法](#)
4. [Write your own Android Sync Adapter](#)

5. Write your own Android Authenticator

6. Android developer

- [android.accounts](#)
- [AccountManager](#)
- [AbstractAccountAuthenticator](#)
- [AccountAuthenticatorActivity](#)
- [Creating a Sync Adapter](#)

7. Android篇从底层实现让进程不被杀死（失效Closed）

8. Android 4.3+ NotificationListenerService 的使用

9. Going multiprocess on Android

后记

“

2016.5.24

1. 本文发布时间写错了，5.19手贱成了6.19，就酱紫吧，懒得改了，五月份看过的童鞋就当狠狠滴穿越了一把吧，O(∩_∩)O哈哈哈~~

2. 本文在[V2EX](#)、[稀土掘金](#)、[博客园](#)、[CSDN](#)等等诸多网站上有转载或发布，收到了很多评论和讨论，其中有一部分以“天下兴亡匹夫有责”的心态批判笔者等同类开发者把Android生态给搞坏了，提到iOS的诸多好处等等，阐述几点个人观点：

① 据笔者研究，目前双Service拉起的方式在国内排前几的应用(微信/支付宝等等)中都有用到，进程互拉方式在360手机助手、应用宝等应用中有用到，这些才是真正黑科技，笔者提到的方法仅仅是取巧性的用到了Android系统提供的方法，谈不上XXX~~

② iOS的封闭造就其天然的优势,不存在这些问题; 而Android的开源, 有诸多问题但不可否认的是其促进了技术的发展, 科技的发展甚至人类的进步。 物极必反, 很多事情都是双刃剑~

③ 后来经一些网友提醒, 发现所谓提异议的这群家伙都是产品汪, 半吊子技术, 所以XXOO~~

④ 法海无涯, 技术无边, 风涯无罪, 南无阿弥陀佛~~

本文首发于skyseraph.com: “一种提高Android应用进程存活率新方法”
同步发表/转载 [cnBlogs](#) / [CSDN](#) / [伯乐在线](#) ...

By [SkySeraph](#)-2016

版权声明

SkySeraph by SkySeraph is licensed under a [Creative Commons BY-NC-ND 4.0 International License](#).

由Bob创作并维护的SkySeraph博客采用[创作共用保留署名-非商业-禁止演绎4.0国际许可证](#).

本文首发于SkySeraph博客 (<http://skyseraph.com>), 版权归作者所有, 欢迎转载, 但未经作者同意必须保留此段声明, 且在文章页面明显位置给出原文连接, 否则保留追究法律责任的权利。

微信扫码打赏SkySeraph



如果您愿意捐助其它金额[请戳我](#)~~, 扫码支付宝/微信

本文永久链接: <http://skyseraph.com/2016/06/19/Android/一种提高Android应用进程存活率新方法/>

分享到: [QQ空间](#) [新浪微博](#) [腾讯微博](#) [人人网](#) [微信](#)

[Comments](#)

NEWER

Select all the cell in UITableView or UICollectionView problem

OLDER

一道Android OpenGL笔试题

RECENT

[TECH > ANDROID](#)

UIAUTOMATOR2.0升级填坑记

2017-06-04

[SKYSERAPH > SHARING](#)

那些年，从博客到出书的博主

2017-05-27

[TECH > ANDROID](#)

APPUIM源码剖析(BOOTSTRAP)

2017-01-26

[SKYSERAPH > LIFE](#)

而立之年，未开始的创业路

2016-10-31

[TECH > TOOLS](#)

JENKINS GITLAB持续集成打包平台搭建

2016-07-18

CATEGORIES

[▸ SkySeraph \(8\)](#)[▸ Life \(7\)](#)[▸ Sharing \(1\)](#)[▸ Tech \(17\)](#)[▸ Android \(4\)](#)[▸ CV \(5\)](#)

- [Media](#) (1)
- [NFC](#) (1)
- [OpenGL](#) (1)
- [Tools](#) (3)
- [iOS](#) (2)

TAG CLOUD

AR/VR [Android](#) [Appuim](#) [CI](#) [CV/MV](#) [DIP](#) [Git](#) [Gitlab](#) [H264](#) [Jenkins](#) [Life](#) [Linux](#) [Movie](#) [NFC](#) [OpenCV](#)
[OpenGL](#) [Plan](#) [Project](#) [Reading](#) [Sharing](#) [SkySeraph](#) [Swift](#) [Tools](#) [Travel](#) [UiAutomator](#) [adb](#) [iOS](#) [jrtp lib](#)
[live555](#) [图像特征](#) [图像算法](#) [存活率](#) [彩色图像](#) [机器视觉](#) [流媒体](#) [瑕疵检测](#) [著书](#) [进程](#)

LINKS

- [SkySeraph-cnBlogs](#)

© 2017 SkySeraph

Powered by [Hexo](#). Modified by [SkySeraph](#). Total 68540 views.