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 8554 ARTICLE.HITS
SKYSERAPH

Catalogue

基础知识

Android 进程优先级

1 进程优先级等级一般分法

- Activte 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，setForground(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;
```

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

```
52  static final int FOREGROUND_APP_ADJ = 0;
53
54  // This is a process that the system or a persistent process has boun
55  // and indicated it is important.
56  static final int PERSISTENT_SERVICE_ADJ = -11;
57
58  // This is a system persistent process, such as telephony.  Definitel
59  // don't want to kill it, but doing so is not completely fatal.
60  static final int PERSISTENT_PROC_ADJ = -12;
61
62  // The system process runs at the default adjustment.
63  static final int SYSTEM_ADJ = -16;
64
65  // Special code for native processes that are not being managed by th
66  // don't have an oom adj assigned by the system).
67  static final int NATIVE_ADJ = -17;
```

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，其不可能会被杀死(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

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  public class FakeService extends Service {
 2      @Nullable
 3      @Override
 4      public IBinder onBind(Intent intent) {
 5          return null;
 6      }
 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
15      @Override
16      public void onDestroy() {
17          stopForeground(true);
18          super.onDestroy();
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

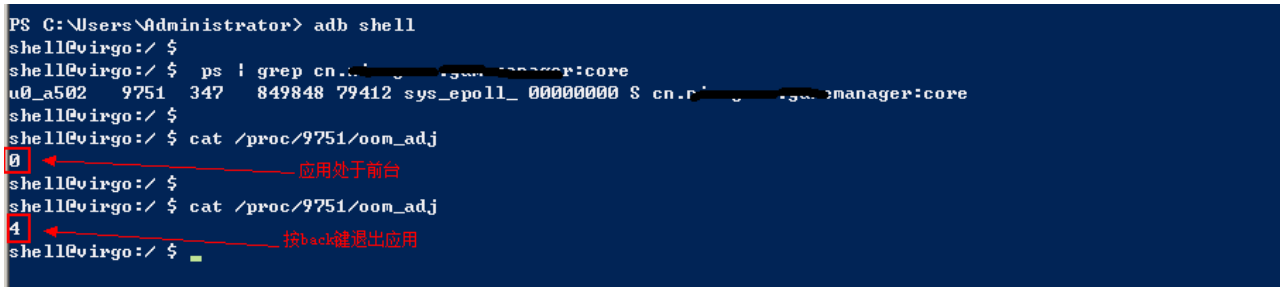
新方法(AccountSync)

思路

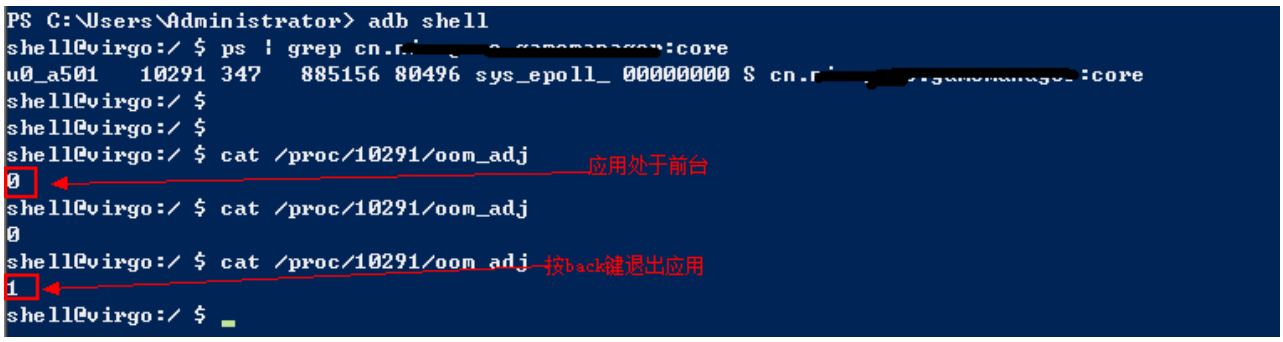
利用Android系统提供的账号和同步机制实现

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



正常情况



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://" + AUTH
4      public static final String TABLE_NAME = "data";
5      public static final Uri CONTENT_URI = Uri.parse(CONTENT_URI_BASE
6
7      @Override
8      public boolean onCreate() {
```

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

```
public class XXSyncService extends Service {
    private static final Object sSyncAdapterLock = new Object();
    private static XXSyncAdapter sSyncAdapter = null;
    @Override
    public void onCreate() {
        synchronized (sSyncAdapterLock) {
            if (sSyncAdapter == null) {
                sSyncAdapter = new XXSyncAdapter(getApplicationContex
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return sSyncAdapter.getSyncAdapterBinder();
    }

    static class XXSyncAdapter extends AbstractThreadedSyncAdapter {
        public XXSyncAdapter(Context context, boolean autoInitialize)
            super(context, autoInitialize);
        }
        @Override
        public void onPerformSync(Account account, Bundle extras, Str
            getContext().getContentResolver().notifyChange(XXAccountP
        }
    }
}
```

- 声明Sync服务

```
<service
        android:name="**.XXSyncService"
        android:exported="true"
```

```
        android:process=":core">
        <intent-filter>
                <action
                        android:name="android.content.SyncAdapter"/>
        </intent-filter>
        <meta-data
                android:name="android.content.SyncAdapter"
                android:resource="@xml/sync_adapter"/>
 </service>
```

其中sync_adapter为：

```
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/androi
              android:accountType="@string/account_auth_type"
              android:allowParallelSyncs="false"
              android:contentAuthority="@string/account_auth_provide"
              android:isAlwaysSyncable="true"
              android:supportsUploading="false"
              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

```
public void triggerRefresh() {
        Bundle b = new Bundle();
        b.putBoolean(ContentResolver.SYNC_EXTRAS_MANUAL, true);
        b.putBoolean(ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
        ContentResolver.requestSync(
                        account,
```

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

添加账号

```
1   Account account = AccountService.GetAccount();
2   AccountManager accountManager = (AccountManager) context.getSystemServ
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
4       @Override
5       public void onCreate() {
6           mAuthenticator = new XXAuthenticator(this);
7       }
8
9       private XXAuthenticator getAuthenticator() {
10          if (mAuthenticator == null)
11              mAuthenticator = new XXAuthenticator(this);
12          return mAuthenticator;
13      }
```

```
14
15      @Override
16      public IBinder onBind(Intent intent) {
17          return getAuthenticator().getIBinder();
18      }
19
20      class XXAuthenticator extends AbstractAccountAuthenticator {
21          private final Context context;
22          private AccountManager accountManager;
23          public XXAuthenticator(Context context) {
24              super(context);
25              this.context = context;
26              accountManager = AccountManager.get(context);
27          }
28
29          @Override
30          public Bundle addAccount(AccountAuthenticatorResponse respons
31                  throws NetworkErrorException {
32                          // 添加账号 示例代码
33              final Bundle bundle = new Bundle();
34              final Intent intent = new Intent(context, AuthActivity.cl
35              intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_
36              bundle.putParcelable(AccountManager.KEY_INTENT, intent);
37              return bundle;
38          }
39
40          @Override
41          public Bundle getAuthToken(AccountAuthenticatorResponse respo
42                  throws NetworkErrorException {
43                          // 认证 示例代码
44              String authToken = accountManager.peekAuthToken(account,
45              //if not, might be expired, register again
46              if (TextUtils.isEmpty(authToken)) {
47                  final String password = accountManager.getPassword(ac
48                  if (password != null) {
49                      //get new token
50                                          authToken = account.name + pa
51                  }
52              }
53              //without password, need to sign again
54              final Bundle bundle = new Bundle();
55              if (!TextUtils.isEmpty(authToken)) {
56                  bundle.putString(AccountManager.KEY_ACCOUNT_NAME, acc
57                  bundle.putString(AccountManager.KEY_ACCOUNT_TYPE, acc
58                  bundle.putString(AccountManager.KEY_AUTHTOKEN, authTo
59                  return bundle;
60              }
```

```
61
62              //no account data at all, need to do a sign
63              final Intent intent = new Intent(context, AuthActivity.cl
64              intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_
65              intent.putExtra(AuthActivity.ARG_ACCOUNT_NAME, account.na
66              bundle.putParcelable(AccountManager.KEY_INTENT, intent);
67              return bundle;
68          }
69
70          @Override
71          public String getAuthTokenLabel(String authTokenType) {
72  //              throw new UnsupportedOperationException();
73              return null;
74          }
75
76          @Override
77          public Bundle editProperties(AccountAuthenticatorResponse res
78              return null;
79          }
80
81          @Override
82          public Bundle confirmCredentials(AccountAuthenticatorResponse
83                  throws NetworkErrorException {
84              return null;
85          }
86
87          @Override
88          public Bundle updateCredentials(AccountAuthenticatorResponse
89                  throws NetworkErrorException {
90              return null;
91          }
92
93          @Override
94          public Bundle hasFeatures(AccountAuthenticatorResponse respon
95                  throws NetworkErrorException {
96              return null;
97          }
98      }
99  }
```

- 声明Account服务

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

```
        android:exported="true"
        android:process=":core">
        <intent-filter>
                <action
                        android:name="android.accounts.AccountAuthent
        </intent-filter>
        <meta-data
                android:name="android.accounts.AccountAuthenticator"
                android:resource="@xml/authenticator"/>
 </service>
```

其中authenticator为：

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

- 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

微信扫码打赏SkySeraph

￥1.68 ￥5.20 ￥13.14



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 UITabView or UICollectionView problem

OLDER

一道Android OpenGL笔试题

RECENT

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

TECH › IOS
SELECT ALL THE CELL IN UITABVIEW OR UICOLLECTIONVIEW PROBLEM
2016-06-30

TECH › ANDROID
一种提高ANDROID应用进程存活率新方法
2016-06-19

TECH › OPENGL
一道ANDROID OPENGL笔试题
2016-05-05

CATEGORIES

- SkySeraph (4)
  - Life (3)
  - Reading (1)
- Tech (13)
  - Android (2)
  - CV (5)
  - Media (1)
  - NFC (1)

- OpenGL (1)
- Tools (1)
- iOS (2)

TAG CLOUD

AR/VR Android CI CV/MV DIP Gitlab H264 Jenkins Life NFC OpenCV OpenGL Project Reading SkySeraph Swift iOS jrtplib 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 进程

LINKS

- SkySeraph-cnBlogs