# 一键结束Activity

具体方式总结如下：



##### 方法一：通过Android组件退出Activity

方式：将App的入口activity设置为SingleTask模式，然后在入口activity回调的onNewIntent中关闭自身即可

1 这里简单地说一下activity的四种启动模式



代码实例：

@Override

protected void onNewIntent(Intent intent) {

super.onNewIntent(intent);

if (intent != null) {

// 是否退出App的标识

boolean isExitApp = intent.getBooleanExtra("exit", false);

if (isExitApp) {

// 关闭自身

this.finish();

}

}

}

1. Activity的onNewIntent()方法何时会被调用?

前提：ActivityA已经启动过,处于当前应用的Activity堆栈中

* 当ActivityA的LaunchMode为SingleTop时，如果ActivityA在栈顶,且现在要再启动ActivityA，这时会调用onNewIntent()方法
* 当ActivityA的LaunchMode为SingleInstance,SingleTask时,如果已经ActivityA已经在堆栈中，那么此时会调用onNewIntent()方法
* 当ActivityA的LaunchMode为Standard时，由于每次启动ActivityA都是启动新的实例，和原来启动的没关系，所以不会调用原来ActivityA的onNewIntent方法

**优点：**

使用简单 & 方便

**缺点：**

* 规定 App的入口Activity采用SingleTask启动模式
* 使用范围局限：只能结束当前任务栈的Activity，若出现多任务栈（即采用SingleInstance启动模式）则无法处理

**应用场景：**

单任务栈

##### 方法二：采用Activity启动标记位

方式：对入口activity使用2种标记位

* Intent.FLAG\_ACTIVITY\_CLEAR\_TOP：销毁目标Activity和它之上的所有Activity，重新创建目标Activity
* Intent.FLAG\_ACTIVITY\_SINGLE\_TOP：若启动的Activity位于任务栈栈顶，那么此Activity的实例就不会重建，而是重用栈顶的实例( 调用onNewIntent()）

使用实例：

Intent intent = new Intent();

intent.setClass(Activity2.this, MainActivity.class);

// 设置标记位

intent.setFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);

// 步骤1：该标记位作用：销毁目标Activity和它之上的所有Activity，重新创建目标Activity

intent.addFlags(Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

// 步骤2：若启动的Activity位于任务栈栈顶，那么此Activity的实例就不会重建，而是重用栈顶的实例( 调用实例的 onNewIntent() )

// 在步骤1中：MainActivity的上层的Activity2会被销毁，此时MainActivity位于栈顶；由于步骤2的设置，所以不会新建MainActivity而是重用栈顶的实例&调用实onNewIntent()

// 传入自己设置的退出App标识

intent.putExtra("exit", true);

startActivity(intent);

在MainActivity中的onNewIntent里结束当前activity即可

**优点：**

使用简单 & 方便

**缺点：**

使用范围局限：只能结束当前任务栈的Activity，若出现多任务栈（即采用SingleInstance启动模式）则无法处理

**应用场景：**

Activity单任务栈

##### 方法三：通过系统任务栈

方式：通过ActivityMananger获取当前系统的任务栈，把栈内所有Activity逐个退出

代码实例：

@TargetApi(Build.VERSION\_CODES.LOLLIPOP)

// 1. 通过Context获取ActivityManager

ActivityManager activityManager = (ActivityManager) context

.getApplicationContext().getSystemService(Context.ACTIVITY\_SERVICE);

// 2. 通过ActivityManager获取任务栈

List<ActivityManager.AppTask> appTaskList = activityManager.getAppTasks();

// 3. 逐个关闭Activity

for (ActivityManager.AppTask appTask : appTaskList) {

appTask.finishAndRemoveTask();

}

// 4. 结束进程

// System.exit(0);

**优点：**

使用简单、方便

**缺点：**

* 使用范围局限：只能结束当前任务栈的Activity，若出现多任务栈（即采用SingleInstance启动模式）则无法处理
* 对 Android 版本要求较高：Android 5.0以上

**应用场景：**

Android 5.0以上的 Activity单任务栈

##### 方法四：通过Broadcastreceiver监听广播

方式：在每个 Activity 里注册广播接收器（响应动作 = 关闭自身）；当需要退出 App 时 发送广播请求即可

具体实现：

1. 自定义广播接收器

public class ExitAppReceiver extends BroadcastReceiver {

private Activity activity;

public ExitAppReceiver(Activity activity){

this.activity = activity;

}

@Override

public void onReceive(Context context, Intent intent) {

activity.finish();

}

}

1. 在每个activity里注册广播接收器，相应动作就是关闭自身activity

public class Activity extends AppCompatActivity {

private ExitAppReceiver mExitAppReceiver；

// 1. 在onCreate（）中注册广播接收器

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mExitAppReceiver = new ExitAppReceiver(this);

registerReceiver(mExitAppReceiver

,newIntentFilter(BaseApplication.EXIT));

}

// 1. 在onDestroy（）中注销广播接收器

protected void onDestroy() {

super.onDestroy();

unregisterReceiver(mExitAppReceive);

}

}

1. 当需要退出app时，发送广播请求

context.sendBroadcast(new Intent(BaseApplication.EXIT));

**优点：**

应用场景广泛：兼顾单 / 多任务栈 & 多启动模式的情况

**缺点：**

实现复杂：需要在每个 Activity 里注册广播接收器

**应用场景：**

任意情况下的一键退出 App，但无法终止 App 进程

##### 方法五：自身实现，创建链表

方式：通过在Applicaiton子类中创建一个链表，保存正在运行的activity的实例，当需要一键退出app，把链表里的所有activity实例逐个退出

具体实现：

1. 在Applicaition子类中创建链表

public class Carson\_BaseApplicaiton extends Application {

// 此处采用 LinkedList作为容器，增删速度快

public static LinkedList<Activity> activityLinkedList;

@Override

public void onCreate() {

super.onCreate();

activityLinkedList = new LinkedList<>();

registerActivityLifecycleCallbacks(

new ActivityLifecycleCallbacks() {

@Override

public void onActivityCreated(Activity activity

, Bundle savedInstanceState) {

Log.d(TAG, "onActivityCreated: "

+ activity.getLocalClassName());

activityLinkedList.add(activity);

// 在Activity启动时（onCreate()） 写入Activity实例到容器内

}

@Override

public void onActivityDestroyed(Activity activity) {

Log.d(TAG, "onActivityDestroyed: "

+ activity.getLocalClassName());

activityLinkedList.remove(activity);

// 在Activity结束时（Destroyed（）） 写出Activity实例

}

@Override

public void onActivityStarted(Activity activity) {}

@Override

public void onActivityResumed(Activity activity) {}

@Override

public void onActivityPaused(Activity activity) {}

@Override

public void onActivityStopped(Activity activity) {}

@Override

public void onActivitySaveInstanceState(Activity activity

, Bundle outState) {}

});

}

public void exitApp() {

Log.d(TAG, "容器内的Activity列表如下 ");

// 先打印当前容器内的Activity列表

for (Activity activity : activityLinkedList) {

Log.d(TAG, activity.getLocalClassName());

}

Log.d(TAG, "正逐步退出容器内所有Activity");

// 逐个退出Activity

for (Activity activity : activityLinkedList) {

activity.finish();

}

// 结束进程

// System.exit(0);

}

}

1. 需要一键退出app时，退出

private Carson\_BaseApplicaiton app;

app = (Carson\_BaseApplicaiton)getApplication();

app.exitApp();

**优点：**

应用场景广泛：兼顾单 / 多任务栈 & 多启动模式的情况

缺点：

需要 Activity 经历正常的生命周期，即创建时调用onCreate（），结束时调用onDestroy（）

因为只有这样经历正常的生命周期才能将 Activity正确写入 & 写出 容器内

**应用场景：**

任意情况下的一键退出 App 实现

##### 方法六：使用RxBus

原理：使用 RxBus当作事件总线，在每个 Activity里注册RxBus订阅（响应动作 = 关闭自身）；当需要退出App时 发送退出事件请求即可。

具体实现：

1. 在每个activity里注册RxBus订阅，相应动作为关闭自身

public class Activity extends AppCompatActivity {

private Disposable disposable;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity2);

// 注册RxBus订阅

disposable = RxBus.getInstance().toObservable(String.class)

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

// 响应动作 = 关闭自身

if (s.equals("exit")){

finish();

}

}

});

}

// 注意一定要取消订阅

@Override

protected void onDestroy() {

if (!disposable.isDisposed()){

disposable.dispose();;

}

}

1. 当需要退出app时，发送退出事件

RxBus.getInstance().post("exit");

System.exit(0);

**优点：**

可与 RxJava & RxBus 相结合

**缺点：**

实现复杂：RxBus 本身的实现难度 & 需要在每个Activity注册和取消订阅 RxBus 使用

**应用场景：**

需要与RxJava 结合使用时

**若项目中没有用到RxJava & RxBus 不建议使用**

上述方法只是结束了所有的activity，应用的进程还是没有结束

### 2 一键结束当前App进程

作用：结束当前activity和结束app进行，即在结束了app的所有activity后，一键退出app进程

方式：主要采用 Dalvik VM本地方法

具体实现方式

* 方式1：android.os.Process.killProcess（）

android.os.Process.killProcess(android.os.Process.myPid()) ；

* 方式2：System.exit()

// System.exit() = Java中结束进程的方法：关闭当前JVM虚拟机

System.exit(0);

**System.exit(0)和System.exit(1)的区别：**

// 1. System.exit(0)：正常退出；

// 2. System.exit(1)：非正常退出，通常这种退出方式应该放在catch块中。

##### 特殊案例分析：

**假设场景：**当前 Activity ≠ 当前任务栈最后1个Activity时，调用上述两个方法会出现什么情况呢？（即Activity1 - Activity2 -Activity3（在Activity3调用上述两个方法））

**结果为：**

1. 结束Activity3（当前 Activity ）& 结束进程   
2. 再次重新开启进程 & 启动 Activity1 、 Activity2

即在Android 中，调用上述Dalvik VM本地方法结果是：

* 结束当前 Activity & 结束进程
* 之后再重新开启进程 & 启动 之前除当前 Activity 外的已启动的 Activity

**原理：**

Android中的ActivityManager时刻监听着进程，一旦发现进程被非正常结束，它将会试图去重启这个进程

### 总结

在 需要实现 一键退出 App 功能时，实际上是需要完成2个步骤：

* 步骤1：一键结束当前App所有的Activity
* 步骤2：一键结束当前App进程