### **1. RxBus优雅式**

首先，在基类BaseActivity里，注册RxBus监听：

public class BaseActivity3 extends AppCompatActivity {
 Subscription mSubscription;

 @Override
 public void onCreate(@Nullable Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 initRxBus();
 }
 //接收退出的指令，关闭所有activity
 private void initRxBus() {
 mSubscription = RxBus.getInstance().toObserverable(NormalEvent.class)
 .subscribe(new Action1<NormalEvent>() {
 @Override
 public void call(NormalEvent userEvent) {
 if (userEvent.getType() == -1) {
 finish();
 }
 }
 },
 new Action1<Throwable>() {
 @Override
 public void call(Throwable throwable) {
 }
 });
 }

 @Override
 protected void onDestroy() {
 super.onDestroy();
 if (!mSubscription.isUnsubscribed()) {
 mSubscription.unsubscribe();
 }
 }
}

这是事件实体NormalEvent:

public class NormalEvent {
 private int type;

 public NormalEvent(int type) {
 this.type = type;
 }

 public int getType() {
 return type;
 }

 public void setType(int type) {
 this.type = type;
 }
}

新建RxBus类

public class RxBus {

 private static volatile RxBus mInstance;

 private final Subject bus;

 public RxBus()
 {
 bus = new SerializedSubject<>(PublishSubject.create());
 }

 /\*\*
 \* 单例模式RxBus
 \*
 \* @return
 \*/
 public static RxBus getInstance()
 {

 RxBus rxBus2 = mInstance;
 if (mInstance == null)
 {
 synchronized (RxBus.class)
 {
 rxBus2 = mInstance;
 if (mInstance == null)
 {
 rxBus2 = new RxBus();
 mInstance = rxBus2;
 }
 }
 }

 return rxBus2;
 }

 /\*\*
 \* 发送消息
 \*
 \* @param object
 \*/
 public void post(Object object)
 {

 bus.onNext(object);

 }

 /\*\*
 \* 接收消息
 \*
 \* @param eventType
 \* @param <T>
 \* @return
 \*/
 public <T> Observable<T> toObserverable(Class<T> eventType)
 {
 return bus.ofType(eventType);
 }
}

最后，在需要退出的地方调用：

 RxBus.getInstance().post(new NormalEvent(-1));//发送退出指令

### **2. 容器式：**

建立一个全局容器，把所有的Activity存储起来，退出时循环遍历finish所有Activity

public class BaseActivity extends AppCompatActivity {
 @Override
 public void onCreate(@Nullable Bundle savedInstanceState ) {
 super.onCreate(savedInstanceState);
 ActivityManager.getActivityManager().addActivity(this);
 }
 @Override protected void onDestroy() {
 super.onDestroy();
 // 结束Activity&从栈中移除该Activity
 ActivityManager.getActivityManager().finishActivity();
 }

}

public class ActivityManager {
 // Activity栈
 private static Stack<Activity> activityStack;
 // 单例模式
 private static ActivityManager instance;

 private ActivityManager() {
 }

 /\*\*
 \* 单一实例
 \*/
 public static ActivityManager getActivityManager() {
 if (instance == null) {
 instance = new ActivityManager();
 }
 return instance;
 }

 /\*\*
 \* 添加Activity到堆栈
 \*/
 public void addActivity(Activity activity) {
 if (activityStack == null) {
 activityStack = new Stack<Activity>();
 }
 activityStack.add(activity);
 }

 /\*\*
 \* 获取当前Activity（堆栈中最后一个压入的）
 \*/
 public Activity currentActivity() {
 Activity activity = activityStack.lastElement();
 return activity;
 }

 /\*\*
 \* 结束当前Activity（堆栈中最后一个压入的）
 \*/
 public void finishActivity() {
 Activity activity = activityStack.lastElement();
 finishActivity(activity);
 }

 /\*\*
 \* 结束指定的Activity
 \*/
 public void finishActivity(Activity activity) {
 if (activity != null) {
 activityStack.remove(activity);
 activity.finish();
 activity = null;
 }
 }

 /\*\*
 \* 结束指定类名的Activity
 \*/
 public void finishActivity(Class<?> cls) {
 for (Activity activity : activityStack) {
 if (activity.getClass().equals(cls)) {
 finishActivity(activity);
 }
 }
 }

 /\*\*
 \* 结束所有Activity
 \*/
 public void finishAllActivity() {
 for (int i = 0; i < activityStack.size(); i++) {
 if (null != activityStack.get(i)) {
 activityStack.get(i).finish();
 }
 }
 activityStack.clear();
 }

 /\*\*
 \* 退出应用程序
 \*/
 public void AppExit(Context context) {
 try {
 finishAllActivity();
 //根据进程ID，杀死该进程
 android.os.Process.killProcess(android.os.Process.myPid());
 //退出真个应用程序
 System.exit(0);
 } catch (Exception e) {
 }
 }

}

### **3. 广播式**

通过在BaseActivity中注册一个广播，当退出时发送一个广播，finish退出

public class BaseActivity2 extends AppCompatActivity {
 private static final String EXITACTION = "action.exit";
 private ExitReceiver exitReceiver = new ExitReceiver();
 @Override protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 IntentFilter filter = new IntentFilter();
 filter.addAction(EXITACTION);
 registerReceiver(exitReceiver, filter);
 }
 @Override protected void onDestroy() {
 super.onDestroy(); unregisterReceiver(exitReceiver);
 }
 class ExitReceiver extends BroadcastReceiver {
 @Override public void onReceive(Context context, Intent intent) {
 BaseActivity2.this.finish();
 }
 }

}

### **4. SingleTask**

1、设置MainActivity的加载模式为singleTask

android:launchMode="singleTask"

2、将退出出口放置在MainActivity

private boolean mIsExit;
 @Override /\*\* \* 双击返回键退出 \*/
 public boolean onKeyDown(int keyCode, KeyEvent event) {
 if (keyCode == KeyEvent.KEYCODE\_BACK) {
 if (mIsExit) {
 this.finish();
 } else {
 Toast.makeText(this, "再按一次退出", Toast.LENGTH\_SHORT).show();
 mIsExit = true;
 new Handler().postDelayed(new Runnable() {

 @Override public void run() {
 mIsExit = false;
 }
 }, 2000);
 } return true;
 } return super.onKeyDown(keyCode, event);
 }

### **5. SingleTask改版式**

第一步设置MainActivity的加载模式为singleTask

android:launchMode="singleTask"

第二步重写onNewIntent()方法

private static final String TAG\_EXIT = "exit";
 @Override
 protected void onNewIntent(Intent intent) {
 super.onNewIntent(intent);
 if (intent != null) {
 boolean isExit = intent.getBooleanExtra(TAG\_EXIT, false);
 if (isExit) { this.finish();
 }
 }
 }

第三步 退出

Intent intent = new Intent(this,MainActivity.class); intent.putExtra(MainActivity.TAG\_EXIT, true);
startActivity(intent);