前言：最近在研究Handler的知识，其中涉及到一个问题，如何避免Handler带来的内存溢出问题。在网上找了很多资料，有很多都是互相抄的，没有实际的作用。

本文的内存泄漏检测工具是：LeakCanary  github地址：<https://github.com/square/leakcanary>

**什么是内存泄漏？**

* 内存泄漏是当程序不再使用到的内存时，释放内存失败而产生了无用的内存消耗。内存泄漏并不是指物理上的内存消失，这里的内存泄漏是值由程序分配的内存但是由于程序逻辑错误而导致程序失去了对该内存的控制，使得内存浪费。

**怎样会导致内存泄漏？**

* 资源对象没关闭造成的内存泄漏，如查询数据库后没有关闭游标cursor
* 构造Adapter时，没有使用 convertView 重用
* Bitmap对象不在使用时调用recycle()释放内存
* 对象被生命周期长的对象引用，如activity被静态集合引用导致activity不能释放

**内存泄漏有什么危害？**

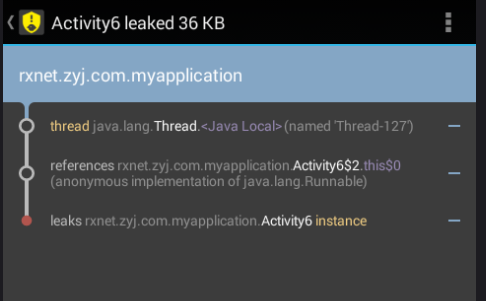
* 内存泄漏对于app没有直接的危害，即使app有发生内存泄漏的情况，也不一定会引起app崩溃，但是会增加app内存的占用。内存得不到释放，慢慢的会造成app内存溢出。所以我们解决内存泄漏的目的就是防止app发生内存溢出。

**1、新建线程引起的Activity内存泄漏**

例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package rxnet.zyj.com.myapplication;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;    public class Activity6 extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_6);            findViewById( R.id.finish).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            new Thread(new Runnable() {              @Override              public void run() {                  try {<br>                    //模拟耗时操作                      Thread.sleep( 15000 );                  } catch (InterruptedException e) {                      e.printStackTrace();                  }              }          }).start();        }  } |

　　运行上面的代码后，点击finish按钮,过一会儿发生了内存泄漏的问题。



**为什么Activity6会发生内存泄漏？**

进入Activity6 界面，然后点击finish按钮，Activity6销毁，但是Activity6里面的线程还在运行，匿名内部类Runnable对象引用了Activity6的实例，导致Activity6所占用的内存不能被GC及时回收。

**如何改进？**

Runnable改为静态非匿名内部类即可。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package rxnet.zyj.com.myapplication;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;    public class Activity6 extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_6);            findViewById( R.id.finish).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            new Thread( new MyRunnable()).start();        }        private static class MyRunnable implements Runnable {            @Override          public void run() {              try {                  Thread.sleep( 15000 );              } catch (InterruptedException e) {                  e.printStackTrace();              }          }      }    } |

**2、Activity添加监听器造成Activity内存泄漏**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package rxnet.zyj.com.myapplication;    import android.app.Activity;  import android.os.Bundle;    public class LeakActivity extends Activity {      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          NastyManager.getInstance().addListener(this);      }  } |

　　这个是在开发中经常会犯的错误，NastyManager.getInstance() 是一个单例，当我们通过 addListener(this) 将 Activity 作为 Listener 和 NastyManager 绑定起来的时候，不好的事情就发生了。

**如何改进？**

想要修复这样的 Bug，其实相当简单，就是在你的 Acitivity 被销毁的时候，将他和 NastyManager 取消掉绑定就好了。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package rxnet.zyj.com.myapplication;    import android.app.Activity;  import android.os.Bundle;    public class LeakActivity extends Activity {      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          NastyManager.getInstance().addListener(this);      }        @Override      protected void onDestroy() {          super.onDestroy();          NastyManager.getInstance().removeListener(this);      }  } |

**3、Handler 匿名内部类造成内存溢出？**

先看着一段代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.os.Handler;  import android.os.Message;  import android.support.v7.app.AppCompatActivity;  import android.util.Log;  import android.view.View;    public class HandlerActivity extends AppCompatActivity {        private final static int MESSAGECODE = 1 ;        private final Handler handler = new Handler(){          @Override          public void handleMessage(Message msg) {              super.handleMessage(msg);              Log.d("mmmmmmmm" , "handler " + msg.what ) ;          }      };        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_handler);            findViewById( R.id.finish ).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            new Thread(new Runnable() {              @Override              public void run() {                  handler.sendEmptyMessage( MESSAGECODE ) ;                  try {                      Thread.sleep( 8000 );                  } catch (InterruptedException e) {                      e.printStackTrace();                  }                  handler.sendEmptyMessage( MESSAGECODE ) ;              }          }).start() ;        }  } |

　　这段代码运行起来后，立即点击 finish 按钮，通过检测，发现 HandlerActivity 出现了内存泄漏。当Activity finish后，延时消息会继续存在主线程消息队列中8秒钟，然后处理消息。而该消息引用了Activity的Handler对象，然后这个Handler又引用了这个Activity。这些引用对象会保持到该消息被处理完，这样就导致该Activity对象无法被回收，从而导致了上面说的 Activity泄露。Handler 是个很常用也很有用的类，异步，线程安全等等。如果有下面这样的代码，会发生什么呢？ handler.postDeslayed ，假设 delay 时间是几个小时… 这意味着什么？意味着只要 handler 的消息还没有被处理结束，它就一直存活着，包含它的 Activity 就跟着活着。我们来想办法修复它，修复的方案是 WeakReference ，也就是所谓的弱引用。垃圾回收器在回收的时候，是会忽视掉弱引用的，所以包含它的 Activity 会被正常清理掉。

**如何避免**

* 使用静态内部类
* 使用弱引用

修改后代码是这样的。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.os.Handler;  import android.os.Message;  import android.support.v7.app.AppCompatActivity;  import android.util.Log;  import android.view.View;    import java.lang.ref.WeakReference;    public class HandlerActivity extends AppCompatActivity {        private final static int MESSAGECODE = 1 ;      private static Handler handler ;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_handler);            findViewById( R.id.finish ).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            handler = new MyHandler( this ) ;            new Thread(new Runnable() {              @Override              public void run() {                  handler.sendEmptyMessage( MESSAGECODE ) ;                  try {                      Thread.sleep( 8000 );                  } catch (InterruptedException e) {                      e.printStackTrace();                  }                  handler.sendEmptyMessage( MESSAGECODE ) ;              }          }).start() ;        }        private static class MyHandler extends Handler {          WeakReference<HandlerActivity> weakReference ;            public MyHandler(HandlerActivity activity ){              weakReference  = new WeakReference<HandlerActivity>( activity) ;          }            @Override          public void handleMessage(Message msg) {              super.handleMessage(msg);              if ( weakReference.get() != null ){                  // update android ui                  Log.d("mmmmmmmm" , "handler " + msg.what ) ;              }          }      }  } |

　　这个Handler已经使用了静态内部类，并且使用了弱引用。但是这个并没有完全解决 HandlerActivity 内存泄漏的问题，罪魁祸首是线程创建的方式出了问题，就像本文的第一个例子一样。改进的方式，是把Runnable类写成静态内部类。

最终完整的代码如下：

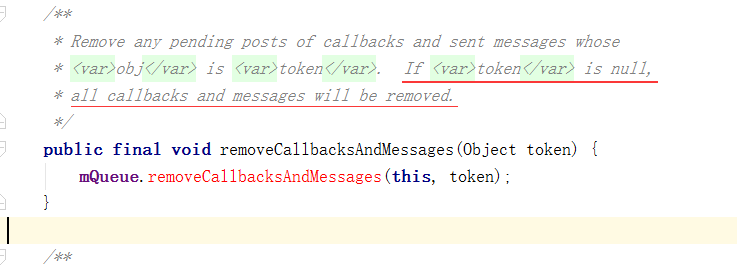
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.os.Handler;  import android.os.Message;  import android.support.v7.app.AppCompatActivity;  import android.util.Log;  import android.view.View;    import java.lang.ref.WeakReference;    public class HandlerActivity extends AppCompatActivity {        private final static int MESSAGECODE = 1 ;      private static Handler handler ;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_handler);            findViewById( R.id.finish ).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            //创建Handler          handler = new MyHandler( this ) ;            //创建线程并且启动线程          new Thread( new MyRunnable() ).start();      }        private static class MyHandler extends Handler {          WeakReference<HandlerActivity> weakReference ;            public MyHandler(HandlerActivity activity ){              weakReference  = new WeakReference<HandlerActivity>( activity) ;          }            @Override          public void handleMessage(Message msg) {              super.handleMessage(msg);              if ( weakReference.get() != null ){                  // update android ui                  Log.d("mmmmmmmm" , "handler " + msg.what ) ;              }          }      }        private static class MyRunnable implements Runnable {            @Override          public void run() {              handler.sendEmptyMessage( MESSAGECODE ) ;              try {                  Thread.sleep( 8000 );              } catch (InterruptedException e) {                  e.printStackTrace();              }              handler.sendEmptyMessage( MESSAGECODE ) ;          }      }  } |

　　等等，还没完呢？

上面这个代码已经有效的解决了Handler，Runnable 引用Activity实例从而导致内存泄漏的问题，但是这不够。因为内存泄漏的核心原因就是这个某个对象应该被系统回收内存的时候，却被其他对象引用，造成该内存无法回收。所以我们在写代码的时候，要始终绷着这个弦。再回到上面这个问题，当当前Activity调用finish销毁的时候，在这个Activity里面所有线程是不是应该在OnDestory()方法里，取消线程。当然是否取消异步任务，要看项目具体的需求，比如在Activity销毁的时候，启动一个线程，异步写log日志到本地磁盘，针对这个需求却需要在OnDestory()方法里开启线程。所以根据当前环境做出选择才是正解。

所以我们还可以修改代码为：在onDestroy() 里面移除所有的callback 和 Message 。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.os.Handler;  import android.os.Message;  import android.support.v7.app.AppCompatActivity;  import android.util.Log;  import android.view.View;    import java.lang.ref.WeakReference;    public class HandlerActivity extends AppCompatActivity {        private final static int MESSAGECODE = 1 ;      private static Handler handler ;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_handler);            findViewById( R.id.finish ).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            //创建Handler          handler = new MyHandler( this ) ;            //创建线程并且启动线程          new Thread( new MyRunnable() ).start();      }        private static class MyHandler extends Handler {          WeakReference<HandlerActivity> weakReference ;            public MyHandler(HandlerActivity activity ){              weakReference  = new WeakReference<HandlerActivity>( activity) ;          }            @Override          public void handleMessage(Message msg) {              super.handleMessage(msg);              if ( weakReference.get() != null ){                  // update android ui                  Log.d("mmmmmmmm" , "handler " + msg.what ) ;              }          }      }        private static class MyRunnable implements Runnable {            @Override          public void run() {              handler.sendEmptyMessage( MESSAGECODE ) ;              try {                  Thread.sleep( 8000 );              } catch (InterruptedException e) {                  e.printStackTrace();              }              handler.sendEmptyMessage( MESSAGECODE ) ;          }      }        @Override      protected void onDestroy() {          super.onDestroy();           //如果参数为null的话，会将所有的Callbacks和Messages全部清除掉。          handler.removeCallbacksAndMessages( null );      }  } |



**4、AsyncTask造成内存泄漏**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | package rxnet.zyj.com.myapplication;    import android.os.AsyncTask;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.util.Log;  import android.view.View;    public class Activity2 extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_2);            findViewById( R.id.finish2).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });              new AsyncTask<String,Integer,String>(){                @Override              protected String doInBackground(String... params) {                  try {                      Thread.sleep( 6000 );                  } catch (InterruptedException e) {                  }                  return "ssss";              }                @Override              protected void onPostExecute(String s) {                  super.onPostExecute(s);                  Log.d( "mmmmmm activity2 " , "" + s ) ;              }            }.executeOnExecutor( AsyncTask.THREAD\_POOL\_EXECUTOR , "" ) ;        }  } |

**为什么？**

上面代码在activity中创建了一个匿名类AsyncTask，匿名类和非静态内部类相同，会持有外部类对象，这里也就是activity，因此如果你在Activity里声明且实例化一个匿名的AsyncTask对象，则可能会发生内存泄漏，如果这个线程在Activity销毁后还一直在后台执行，那这个线程会继续持有这个Activity的引用从而不会被GC回收，直到线程执行完成。

**怎么解决？**

* 自定义静态AsyncTask类
* AsyncTask的周期和Activity周期保持一致。也就是在Activity生命周期结束时要将AsyncTask cancel掉。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | package rxnet.zyj.com.myapplication;    import android.os.AsyncTask;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;    public class AsyncTaskActivity extends AppCompatActivity {        private static MyTask myTask ;      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_asynctask);            findViewById( R.id.finish).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            myTask = new MyTask() ;          myTask.executeOnExecutor( AsyncTask.THREAD\_POOL\_EXECUTOR , "") ;        }        private static class MyTask extends AsyncTask{            @Override          protected Object doInBackground(Object[] params) {              try {                  //模拟耗时操作                  Thread.sleep( 15000 );              } catch (InterruptedException e) {                  e.printStackTrace();              }              return "";          }      }        @Override      protected void onDestroy() {          super.onDestroy();            //取消异步任务          if ( myTask != null ){              myTask.cancel(true ) ;          }      }  } |

**5、Timer Tasks 造成内存泄漏**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.support.v7.app.AppCompatActivity;  import android.view.View;    import java.util.Timer;  import java.util.TimerTask;    public class TimerActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_2);            findViewById( R.id.finish2).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            //开始定时任务          timer();      }        void timer(){          new Timer().schedule(new TimerTask() {              @Override              public void run() {                  while(true);              }          },1000 );  // 1秒后启动一个任务      }  } |

**为什么？**

这里内存泄漏在于Timer和TimerTask没有进行Cancel，从而导致Timer和TimerTask一直引用外部类Activity。

**怎么解决？**

* 在适当的时机进行Cancel。
* TimerTask用静态内部类

**注意：在网上看到一些资料说，解决TimerTask内存泄漏可以使用在适当的时机进行Cancel。经过测试，证明单单使用在适当的时机进行Cancel ， 还是有内存泄漏的问题。所以一定要用静态内部类配合使用。**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package rxnet.zyj.com.myapplication;    import android.os.Bundle;  import android.support.v7.app.AppCompatActivity;  import android.util.Log;  import android.view.View;    import java.util.Timer;  import java.util.TimerTask;    public class TimerActivity extends AppCompatActivity {        private TimerTask timerTask ;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_2);            findViewById( R.id.finish2).setOnClickListener(new View.OnClickListener() {              @Override              public void onClick(View v) {                  finish();              }          });            //开始定时任务          timer();      }        void timer(){          timerTask = new MyTimerTask() ;          new Timer().schedule( timerTask ,1000 );  // 1秒后启动一个任务      }        private static class MyTimerTask extends TimerTask{            @Override          public void run() {              while(true){                  Log.d( "ttttttttt" , "timerTask" ) ;              }          }      }        @Override      protected void onDestroy() {          super.onDestroy();            //取消定时任务          if ( timerTask != null ){              timerTask.cancel() ;          }      }  } |