最近在工作中处理了一些内存泄露的问题，在这个过程中我尤其发现了一些基本的问题反而忽略导致内存泄露，比如静态变量，cursor关闭，流关闭，线程，定时器，反注册，bitmap等等，我稍微统计并总结了一下，当然了，这些问题这么说起来比较笼统，接下来我会根据问题，把一些实例代码贴出来，一步一步分析，在具体的场景下，用行之有效的方法，找出泄露的根本原因，并给出解决方案。  
    现在，就从cursor关闭的问题开始把，谁都知道cursor要关闭，但是往往相反，人们却常常忘记关闭，因为真正的应用场景可能并非理想化的简单。  
**1. 理想化的cursor关闭**

|  |  |
| --- | --- |
| 1  2  3  4 | // Sample Code  Cursor cursor = db.query();  List<String> list = convertToList(cursor);  cursor.close(); |

    这是最简单的cursor使用场景，如果这里的cursor没有关闭，我想可能会引起万千口水，一片骂声。  
    但是实际场景可能并非如此，这里的cursor可能不会关闭，至少有以下两种可能。

**2. Cursor未关闭的可能**  
     (1). cursor.close()之前发生异常。  
     (2). cursor需要继续使用，不能马上关闭，后面忘记关闭了。

**3. Cursor.close()之前发生异常**  
     这个很容易理解，应该也是初学者最开始碰到的常见问题，举例如下:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try {      Cursor c = queryCursor();      int a = c.getInt(1);      ......      // 如果出错,后面的cursor.close()将不会执行      ......      c.close();  } catch (Exception e) {  } |

　　正确写法应该是:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Cursor c;  try {      c = queryCursor();      int a = c.getInt(1);      ......      // 如果出错,后面的cursor.close()将不会执行      //c.close();  } catch (Exception e) {  } finally{      if (c != null) {          c.close();      }  } |

    很简单，但是需要时刻谨记。

**4. Cursor需要继续使用，不能马上关闭**  
    有没有这种情况？怎么办？  
    答案是有，CursorAdapter就是一个典型的例子。  
    CursorAdapter示例如下:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | mCursor = getContentResolver().query(CONTENT\_URI, PROJECTION,  null, null, null);  mAdapter = new MyCursorAdapter(this, R.layout.list\_item, mCursor);  setListAdapter(mAdapter);  // 这里就不能关闭执行mCursor.close(),  // 否则list中将会无数据 |

**5. 这样的Cursor应该什么时候关闭呢？**  
    这是个可以说好回答也可以说不好回答的问题，那就是在Cursor不再使用的时候关闭掉。  
    比如说，  
    上面的查询，如果每次进入或者resume的时候会重新查询执行。  
    一般来说，也只是这种需求，很少需要看不到界面的时候还在不停地显示查询结果，如果真的有，不予讨论，记得最终关掉就OK了。  
    这个时候，我们一般可以在onStop()方法里面把cursor关掉（同时意味着你可能需要在onResume()或者onStart()重新查询一下）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  protected void onStop() {      super.onStop();      // mCursorAdapter会释放之前的cursor，相当于关闭了cursor      mCursorAdapter.changeCursor(null);  } |

　　我专门附上CursorAdapter的changeCursor()方法源码，让大家看的更清楚，免得不放心changeCursor(null)方法：

|  |  |
| --- | --- |
| 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 | /\*\*   \* Change the underlying cursor to a new cursor. If there is an existing cursor it will be   \* closed.   \*   \* @param cursor The new cursor to be used   \*/  public void changeCursor(Cursor cursor) {      Cursor old = swapCursor(cursor);      if (old != null) {          old.close();      }  }    /\*\*   \* Swap in a new Cursor, returning the old Cursor.  Unlike   \* {@link #changeCursor(Cursor)}, the returned old Cursor is <em>not</em>   \* closed.   \*   \* @param newCursor The new cursor to be used.   \* @return Returns the previously set Cursor, or null if there wasa not one.   \* If the given new Cursor is the same instance is the previously set   \* Cursor, null is also returned.   \*/  public Cursor swapCursor(Cursor newCursor) {      if (newCursor == mCursor) {          return null;      }      Cursor oldCursor = mCursor;      if (oldCursor != null) {          if (mChangeObserver != null) oldCursor.unregisterContentObserver(mChangeObserver);          if (mDataSetObserver != null) oldCursor.unregisterDataSetObserver(mDataSetObserver);      }      mCursor = newCursor;      if (newCursor != null) {          if (mChangeObserver != null) newCursor.registerContentObserver(mChangeObserver);          if (mDataSetObserver != null) newCursor.registerDataSetObserver(mDataSetObserver);          mRowIDColumn = newCursor.getColumnIndexOrThrow("\_id");          mDataValid = true;          // notify the observers about the new cursor          notifyDataSetChanged();      } else {          mRowIDColumn = -1;          mDataValid = false;          // notify the observers about the lack of a data set          notifyDataSetInvalidated();      }      return oldCursor;  } |

6. **实战AsyncQueryHandler中Cursor的关闭问题**  
    AsyncQueryHandler是一个很经典很典型的分析Cursor的例子，不仅一阵见血，能举一反三，而且非常常见，为以后避免。  
    AsyncQueryHandler文档参考地址:  
    <http://developer.android.com/reference/android/content/AsyncQueryHandler.html>    下面这段代码是Android2.3系统中Mms信息主页面ConversationList源码的一部分,大家看看Cursor正确关闭了吗？

|  |  |
| --- | --- |
| 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 | private final class ThreadListQueryHandler extends AsyncQueryHandler {      public ThreadListQueryHandler(ContentResolver contentResolver) {          super(contentResolver);      }        @Override      protected void onQueryComplete(int token, Object cookie, Cursor cursor) {          switch (token) {          case THREAD\_LIST\_QUERY\_TOKEN:              mListAdapter.changeCursor(cursor);              setTitle(mTitle);              ... ...              break;            case HAVE\_LOCKED\_MESSAGES\_TOKEN:              long threadId = (Long)cookie;              confirmDeleteThreadDialog(new DeleteThreadListener(threadId, mQueryHandler,                      ConversationList.this), threadId == -1,                      cursor != null && cursor.getCount() > 0,                      ConversationList.this);              break;            default:              Log.e(TAG, "onQueryComplete called with unknown token " + token);          }      }  }    @Override  protected void onStop() {      super.onStop();        mListAdapter.changeCursor(null);  } |

    大家觉得有问题吗？  
    主要是两点：  
    (1). THREAD\_LIST\_QUERY\_TOKEN分支的Cursor正确关闭了吗？  
    (2). HAVE\_LOCKED\_MESSAGES\_TOKEN分支的Cursor正确关闭了吗？  
    根据前面的一条条分析，答案是：  
    (1). THREAD\_LIST\_QUERY\_TOKEN分支的Cursor被传递到了mListAdapter了，而mListAdapter在onStop里面使用changeCursor(null),当用户离开当前Activity，这个Cursor被正确关闭了，不会泄露。  
    (2). HAVE\_LOCKED\_MESSAGES\_TOKEN分支的Cursor（就是参数cursor），只是作为一个判断的一个条件，被使用后不再使用，但是也没有关掉，所以cursor泄露，在StrictMode监视下只要跑到这个地方都会抛出这个错误：

|  |  |
| --- | --- |
| 1  2  3  4 | E/StrictMode(639): A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.  E/StrictMode(639): java.lang.Throwable: Explicit termination method 'close' not called  E/StrictMode(639): at dalvik.system.CloseGuard.open(CloseGuard.java:184)  ... ... |

　　在Android4.0 JellyBean中谷歌修正了这个泄露问题，相关代码如下:

|  |  |
| --- | --- |
| 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 | private final class ThreadListQueryHandler extends ConversationQueryHandler {      public ThreadListQueryHandler(ContentResolver contentResolver) {          super(contentResolver);      }        @Override      protected void onQueryComplete(int token, Object cookie, Cursor cursor) {          switch (token) {          case THREAD\_LIST\_QUERY\_TOKEN:              mListAdapter.changeCursor(cursor);                ... ...                break;            case UNREAD\_THREADS\_QUERY\_TOKEN:              // 新增的UNREAD\_THREADS\_QUERY\_TOKEN分子和HAVE\_LOCKED\_MESSAGES\_TOKEN分支也是类似的情况，cursor在jellybean中被及时关闭了              int count = 0;              if (cursor != null) {                  count = cursor.getCount();                  cursor.close();              }              mUnreadConvCount.setText(count > 0 ? Integer.toString(count) : null);              break;            case HAVE\_LOCKED\_MESSAGES\_TOKEN:              @SuppressWarnings("unchecked")              Collection<Long> threadIds = (Collection<Long>)cookie;              confirmDeleteThreadDialog(new DeleteThreadListener(threadIds, mQueryHandler,                      ConversationList.this), threadIds,                      cursor != null && cursor.getCount() > 0,                      ConversationList.this);              // HAVE\_LOCKED\_MESSAGES\_TOKEN分支中的cursor在jellybean中被及时关闭了              if (cursor != null) {                  cursor.close();              }              break;            default:              Log.e(TAG, "onQueryComplete called with unknown token " + token);          }      }  }      @Override  protected void onStop() {      super.onStop();      mListAdapter.changeCursor(null);  } |

　　是不是小看了AsyncQueryHandler，谷歌在早期的版本里面都有一些这样的代码，更何况不注意的我们呢，实际上网上很多使用AsyncQueryHandler举例中都犯了这个错误，看完这篇文章后，以后再也不怕AsyncQueryHandler的cursor泄露了，还说不定能解决很多你现在应用的后台strictmode的cursor not close异常问题。

**7. 小结**  
    虽然我觉得还有很多cursor未关闭的情况没有说到，但是根本问题都是及时正确的关闭cursor。  
    内存泄露cursor篇是我工作经验上的一个总结，专门捋清楚后对我自己对大家觉得都很有帮助，让复杂的问题本质化，简单化！