# Android内存泄露之资源

泡在网上的日子 发表于 2014-11-22 20:43 第 994 次阅读 内存

1

来源 http://blog.csdn.net/zhuanglonghai/article/details/38421253

资源内存泄露主要是资源申请未释放,还有资源没有重复使用。

第一种解决这部分问题的关键在于申请资源后能保证能释放资源。

第二种利用复用机制优化,如池的概念。

# 1.引用资源没有释放

代码如下:

```
private final class SettingsObserver implements Observer {
   public void update(Observable o, Object arg) {
        // todo ...
   }
}
ContentQueryMap.getInstance().addObserver(new SettingsObserver(
```

这段代码正常吗

### 答案是否定

存在的问题是没有办法注销观察者(SettingsObserver),这样带来的问题是没有办法释放该观察者。那么这个对象将伴随整个单例生命周期,无形中就泄露一个SettingsObserver的内存。

# 1.1注册未取消造成的内存泄露

这种Android的内存泄露比纯Java的内存泄露还要严重, 因为其他一些Android程序可能引用我们的 Anroid程序的对象(比如注册机制)。 即使我们的Android程序已经结束了,但是别的引用程序仍然还有 对我们的Android程序的某个对象的引用,泄露的内存依然不能被垃圾回收。 还是我们的对象周期不一致 引起的。 例如: BroadcastReceiver对象注册 ... has leaked IntentReceiver ... Are you missing a call to unregisterReceiver()? Observer对象 被观察对象生命周期和观察者的生命周期不一致 不观察的时候需要需要注销

# 1.2集合中对象没清理造成的内存泄露

我们通常把一些对象的引用加入到了集合中,当我们不需要该对象时,并没有把它的引用从集合中清理掉,这样这个集合就会越来越大。

如果这个集合是static的话,那情况就更严重了

在add 情况也要记得 remove

# 2.资源对象没关闭造成的内存泄露

资源性对象比如(Cursor,File文件等)往往都用了一些缓冲,我们在不使用的时候,应该及时关闭它们,以便它们的缓冲及时回收内存。它们的缓冲不仅存在于java虚拟机内,还存在于java虚拟机外。 如果我们仅仅是把它的引用设置为null,而不关闭它们,往往会造成内存泄露。 因为有些资源性对象, 比如 SQLiteCursor(在析构函数finalize(),如果我们没有关闭它,它自己会调close()关闭), 如果我们没有关闭它,系统在回收它时也会关闭它,但是这样的效率太低了。 因此对于资源性对象在不使用的时候,应该调用它的close()函数,将其关闭掉,然后才置为null. 在我们的程序退出时一定要确保我们的资源性对象已经关闭。 程序中经常会进行查询数据库的操作,但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小, 对内存的消耗不容易被发现,只有在常时间大量操作的情况下才会复现内存问题,这样就会给以后的测试和问题排查带来困难和风险 代码如下:

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

# 合理的写法:

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

### 3.一些不良代码造成的内存压力

有些代码并不造成内存泄露,但是它们, 或是对没使用的内存没进行有效及时的释放, 或是没有有效的利用已有的对象而是频繁的申请新内存, 对内存的回收和分配造成很大影响的,容易迫使虚拟机不得不给该应用进程分配更多的内存,造成不必要的内存开支。

#### 如果优化呢

频繁的申请内存对象和和释放对象,可以考虑Pool 池。 多线程可以考虑线程池 频繁的申请对象释放对象。可以考虑对象池。 例如: AbsListView 中RecycleBin 类是view对象池 频繁的链接资源和释放资源。可以考虑链接资源池。

### 3.1, Bitmap没调用recycle()

Bitmap对象在不使用时,我们应该先调用recycle()释放内存,然后才它设置为null. 虽然recycle()从源码上看,调用它应该能立即释放Bitmap的主要内存,但是测试结果显示它并没能立即释放内存。但是我它应该还是能大大的加速Bitmap的主要内存的释放。

# 3.2,构造Adapter时,没有使用缓存的convertView

以构造ListView的BaseAdapter为例,在BaseAdapter中提共了方法: public View getView(int

position, View convertView, ViewGroup parent) 来向ListView提供每一个item所需要的view对象。 初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的view对象, 同时ListView会将这些view对象缓存起来。当向上滚动ListView时, 原先位于最上面的list item的view对象会被回收,然后被用来构造新出现的最下面的list item。 这个构造过程就是由getView()方法完成的, getView()的第二个形参 View convertView就是被缓存起来的list item的view对象 (初始化时缓存中没有view对象则 convertView是null)。

### 如下图:

```
- -
AbsListView... 🛭 🔝 ViewGroup.class
                               I HotPic.iava
                                            A HotPicActivi...
                                                           HotPicAdapte...
                                                                           I HotPicFragme...
                                                                                           HotPicDetail...
 2220
 2221
              Uparam position The position to display
 2222
               @param isScrap Array of at least 1 boolean, the first entry will become true if
 2223
                                the returned view was taken from the scrap heap, false if otherwise.
 2224
 2225
              @return A view displaying the data associated with the specified position
 222€
 2227
           Viow obtainView(int position, boolean[] isScrap) {
    Trace.traceHegin(Trace_TRACE_TAG_VIEW, "chtainview");
 2228
 2229
 2230
                View compression:
 2231
 2232
 2233
                scrapView = mRecycler.getTransientStateView(position);
 2234
                it (scrapView == null) {
 2235
                    scrapView - mRecycler.getScrapView(nominated)
 223€
 2237
 2238
 2239
                if (scrapView != null) {
                     child = mAdapter.getView(position, scrapView, this)
 2240
 2241
                    if (child.getImportantForAccessibility() == IMFORTANT_FOR_ACCESSIBILITY_AUTO) {
    child.setImportantForAccessibility(IMFORTANT_FOR_ACCESSIBILITY_YES);
 2242
 2243
 2244
 2245
 2246
                     if (child !- scrapView)
 2247
                         mRecycler.addScrapView(scrapView, position);
 2240
                         if (mCacheColorHint != 0) {
 2249
                              child.setDrawingCacheBackgroundColor(mCacheColorHint);
 2250
 2251
                     } else {
 2252
                         isScrap[0] = true;
 2253
 2254
                         // Clear any system-managed transient state so that we can
 2255
                         // recycle this view and bind it to different data.
 2256
                         if (child.isAccessibilityFocused()) {
 2257
                              child.clearAccessibilityFocus();
 2258
 2259
 2260
                         child.dispatchFinishTemporaryDetach();
 2261
 2262
                | else
```

由上图可以看出,如果我们不去使用convertView,而是每次都在getView()中重新实例化一个View对象的话,即浪费时间,也造成内存垃圾,给垃圾回收增加压力,如果垃圾回收来不及的话,虚拟机将不得不给该应用进程分配更多的内存,造成不必要的内存开支。 ListView回收list item的view对象的过程可以查看: android.widget.AbsListView.java --> obtainView 方法。

### 在使用过程中iava代码如下:

```
7
              final LayoutInflater inflater = (LayoutInflater) mConte
                      .getSystemService(Context.LAYOUT INFLATER SERVI
 8
 9
              convertView = inflater.inflate(R.layout.list item icon
              holder = new ViewHolder();
10
              holder.icon = (ImageView) convertView.findViewById(R.id
11
              holder.text = (TextView) convertView.findViewById(R.id.
12
              convertView.setTag(holder);
13
          } else {
14
              holder = (ViewHolder) convertView.getTag();
15
16
         holder.icon.setImageDrawable(mDefaultDrawable);
17
         holder.text.setText(mData[position]);
18
          return convertView;
19
20
      static class ViewHolder {
21
          ImageView icon;
22
         TextView text;
23
      }
24
```

# 优化点

- 1.复用convertView, convertView在ListView有RecycleBin的对象池维护。
- 2.ViewHolder出现减少findViewByld 的调用

代 位

置: https://github.com/loyabe/Docs/tree/master/%E5%86%85%E5%AD%98%E6%B3%84%E9%9C%B2