**android内存优化**

**内存泄漏**

**内存泄漏：**是指程序在申请内存后，无法释放已申请的内存空间。内存泄漏简单地说就是申请了一块内存空间，使用完毕后没有释放掉。它的一般表现方式是程序运行时间越长，占用内存越多，最终用尽全部内存，整个系统崩溃。由程序申请的一块内存，且没有任何一个指针指向它，那么这块内存就泄露了。内存泄漏一般不会导致程序异常，但它会导致程序的内存占用过大，这将提高内存溢出的几率。所以，内存泄露是内存溢出的一种诱因，不是唯一因素。  
内存泄漏是造成应用程序OOM的主要原因之一

**导致内存泄漏的原因：**

**1、静态变量导致的内存泄漏**

**静态变量：**静态变量是在类被load的时候分配内存的，并且存在于方法区。当类被卸载的时候，静态变量被销毁。只要静态变量没有被销毁也没有置null，其对象一直被保持引用，也即引用计数不可能是0，因此不会被垃圾回收。  
**导致内存泄漏的代码如下：**

public class YouhuaTestActivity extends AppCompatActivity {

private static Context mContext;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_youhua\_test);

mContext = this;

}

}

上述代码Activity无法正常销毁，因为静态变量mContext引用了它。

**2、**[**单列模式**](http://www.jianshu.com/p/0dd09187baeb)**导致的内存泄漏**

当单列中传入一个Activity的Context后，该单列就持有Activity的引用，我们知道单例的生命周期和Application的一样长，所以当Activity退出时它的内存并不会被回收。  
所以，单例中的Context最好使用Application的Context,这样可以有效防止内存泄漏，如下：

private AppManager(Context context) {

this.context = context.getApplicationContext();

}

这样的话，无论传入生命类型的Context，最终单例使用的都是Application的Context

**3、属性动画导致的内存泄漏**

属性动画中有一类无限循环的动画，如果在Activity中播放此动画且没有在onDestory中停止动画，即使已经无法在界面上看到动画效果，动画也会一直播放下去，并且这个时候Activity的View会被动画持有，而View有持有了Activity，最终导致Activity无法释放。这种泄漏的解决办法是在onDestory中调用animator.cancal()来停止动画。

**4、Handler造成的内存泄漏**

大家平时开发中喜欢在Activity中直接new一个Handler的匿名内部类，在Java中，非静态的内部类或者匿名类会隐式的持有其外部类的引用，而静态的内部类则不会。这样造成匿名内部类持有一个外部类(通常是Activity)的引用(不然怎么更新ui)，但是Handler常常伴随着一个执行耗时操作的异步线程(如下载多张图片)，如果在完成耗时操作之前，Activity退出，异步线程持有handler的引用，handler持有Activity的引用，从而导致内存泄漏。如下：

public class MainActivity extends AppCompatActivity {

private Handler mHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

// do something

}

};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

loadData();

}

private void loadData(){

//...do request

Message message = Message.obtain();

mHandler.sendMessage(message);

}

}

我们可以通过下面的方法来规避上述问题：

public class MainActivity extends AppCompatActivity {

/\*\*

\* 创建一个静态Handler内部类，然后对Handler持有的对象使用弱引用，这样在回收时也可以回收Handler持有的对象，

\* 这样虽然避免了Activity泄漏，不过Looper线程的消息队列中还是可能会有待处理的消息，

\* 所以我们在Activity的Destroy时或者Stop时应该移除消息队列中的消息，

\*/

private MyHandler mHandler = new MyHandler(this);

private TextView mTextView ;

private static class MyHandler extends Handler {

private WeakReference<Context> reference;

public MyHandler(Context context) {

reference = new WeakReference<>(context);

}

@Override

public void handleMessage(Message msg) {

MainActivity activity = (MainActivity) reference.get();

if(activity != null){

activity.mTextView.setText("");

}

}

}

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mTextView = (TextView)findViewById(R.id.textview);

loadData();

}

private void loadData() {

//...request

Message message = Message.obtain();

mHandler.sendMessage(message);

}

@Override

protected void onDestroy() {

super.onDestroy();

// 移除消息队列中所有消息和所有的Runnable,

mHandler.removeCallbacksAndMessages(null);

// 当然，也可以使用mHandler.removeCallbacks();或mHandler.removeMessages();来移除指定的Runnable和Message。

}

}

**5、上下文对象导致的内存泄漏**

* 使用application的context来替代activity相关的context。  
  尽量避免activity的context在自己的范围外被使用，这样会导致activity无法释放。不要让生命周期长于Activity的对象持有到Activity的引用
* 在Android中，Application Context的生命周期和应用的生命周期一样长，而不是取决于某个Activity的生命周期。  
  如果想保持一个长期生命的对象，并且这个对象需要一个 Context，就可以使用Application对象。可以通过调用Context.getApplicationContext()方法或者 Activity.getApplication()方法来获得Application对象。
* Drawable的对象的内部Callback持有activity的引用，当Activity finish()之后，静态成员drawable始终持有这个Activity的引用，导致内存释放不了。
* Activity内部如果有一个Context的成员变量，将导致Context引用指向的Activity对象释放不了,见上文：静态变量导致的内存泄漏

**6、线程造成的内存泄漏**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

/\*\*

\* 错误的做法

\* 异步任务和Runnable都是一个匿名内部类，因此它们对当前Activity都有一个隐式引用。

\* 如果Activity在销毁之前，任务还未完成，那么将导致Activity的内存资源无法回收，造成内存泄漏

\*/

new AsyncTask<Void, Void, Void>() {

@Override

protected Void doInBackground(Void... params) {

SystemClock.sleep(10000);

return null;

}

}.execute();

new Thread(new Runnable() {

@Override

public void run() {

SystemClock.sleep(10000);

}

}).start();

/\*\*

\* 正确的做法：使用静态内部类

\* 避免了Activity的内存资源泄漏，当然在Activity销毁时候也应该取消相应的任务AsyncTask::cancel()，避免任务在后台执行浪费资源

\*/

new Thread(new MyRunnable()).start();

new MyAsyncTask(this).execute();

}

static class MyAsyncTask extends AsyncTask<Void, Void, Void> {

private WeakReference<Context> weakReference;

public MyAsyncTask(Context context) {

weakReference = new WeakReference<>(context);

}

@Override

protected Void doInBackground(Void... params) {

SystemClock.sleep(10000);

return null;

}

@Override

protected void onPostExecute(Void aVoid) {

super.onPostExecute(aVoid);

MainActivity activity = (MainActivity) weakReference.get();

if (activity != null) {

//...

}

}

}

static class MyRunnable implements Runnable{

@Override

public void run() {

SystemClock.sleep(10000);

}

}

}

更多线程内容请查看[Android中的线程池 ThreadPoolExecutor](http://www.jianshu.com/p/3da543063b8c)

**7、**[**webview**](http://www.jianshu.com/p/32d48ca7d0e0)**导致的内存泄漏优化**

请参考：  
<http://lipeng1667.github.io/2016/08/06/memory-optimisation-for-webview-in-android/>

**8、其他一些导致内存泄漏的原因**

1、[Bitmap](http://www.jianshu.com/p/98c88f9ceafa)对象使用完后，忘记了调用recycle()方法销毁；  
2、解析图片的时候忘记了设置采样率 详见下文  
3、[自定义View](http://www.jianshu.com/p/7468e038825a)时TypedArray使用完后忘记调用recycle()方法释放内存  
4、[ListView](http://www.jianshu.com/p/b7741023bc6f)的适配器类中没有复用convertView  
5、未采用软引用等

* 软引用：  
  如果一个对象只具有软引用，那么如果内存空间足够，垃圾回收器就不会回收它；如果内存 空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。软引用可以和一个引用队 列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。
* 弱引用：  
  如果一个对象只具有弱引用，那么在垃圾回收器线程扫描的过程中，一旦发现了只具有弱引 用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。弱 引用也可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联 的引用队列中。
* 弱引用与软引用的根本区别在于：  
  只具有弱引用的对象拥有更短暂的生命周期，可能随时被回收。而只具有软引用的对象只有当内存不够的时候才被回收，在内存足够的时候，通常不被回收。

6、即时关闭InputStream/OutputStream。  
7、注册某个对象后没有注销，如[广播](http://www.jianshu.com/p/57a1899c17fb)  
8、集合对象没清理造成的内存泄漏。把大量对象的引用放入集合中，但我们不需要该对象时，记得从集合中将不需要的引用清理掉，同理，当对象不需要时，记得将对象的引用设置为null。  
9、资源文件未关闭造成内存泄漏。最常见的是文件流执行完读写操作后，忘记关闭了输入流，输出流；数据库、Content Provider操作完后Cursor忘记了close等等。

**内存溢出**

**内存溢出：**是指程序在申请内存时，没有足够的内存空间供其使用，出现out of memory；比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。

**如何避免内存溢出**

**1、减小对象的内存占用**

避免OOM的第一步就是要尽量减少新分配出来的对象占用内存的大小，尽量使用更加轻量的对象

* 使用更加轻量的数据结构
* 避免在Android里面使用Enum
* 减小Bitmap对象的内存占用
* [inSampleSize：缩放比例](http://www.jianshu.com/p/98c88f9ceafa)，在把图片载入内存之前，我们需要先计算出一个合适的缩放比例，避免不必要的大图载入。
* decode format：解码格式，选择ARGB\_8888/RBG\_565/ARGB\_4444/ALPHA\_8，存在很大差异。

**2、内存对象的重复利用，减少对象的重复创建，从而减少内存的分配与回收**

* 复用系统自带的资源  
  字符串/颜色/图片/动画/样式以及简单布局等这些都可以重复利用
* 列表中ConvertView的复用
* [Bitmap对象的复用](http://www.jianshu.com/p/635fceca82d3)  
  在ListView与GridView等显示大量图片的控件里面需要使用[LRU的机制来缓存处理好的Bitmap](http://www.jianshu.com/p/635fceca82d3)
* [避免在onDraw方法里面执行对象的创建](http://www.jianshu.com/p/ab15167c48da)
* 善用 StringBuilder