

常见的内存泄漏原因及解决方法



李俊的博客

□ 关注

2016.07.21 00:24* 字数 2505 阅读 34354 评论 3 喜欢 22

(Memory Leak, 内存泄漏)

为什么会产生内存泄漏？

当一个对象已经不需要再使用本该被回收时，另外一个正在使用的对象持有它的引用从而导致它不能被回收，这导致本该被回收的对象不能被回收而停留在堆内存中，这就产生了内存泄漏。

内存泄漏对程序的影响？

内存泄漏是造成应用程序OOM的主要原因之一。我们知道Android系统为每个应用程序分配的内存是有限的，而当一个应用中产生的内存泄漏比较多时，这就难免会导致应用所需要的内存超过系统分配的内存限额，这就造成了内存溢出从而导致应用Crash。

如何检查和分析内存泄漏？

因为内存泄漏是在堆内存中，所以对我们来说并不是可见的。通常我们可以借助MAT、LeakCanary等工具来检测应用程序是否存在内存泄漏。

- 1、MAT是一款强大的内存分析工具，功能繁多而复杂。
- 2、LeakCanary则是由Square开源的一款轻量级的第三方内存泄漏检测工具，当检测到程序中产生内存泄漏时，它将以最直观的方式告诉我们哪里产生了内存泄漏和导致谁泄漏了而不能被回收。

常见的内存泄漏及解决方法

1、单例造成的内存泄漏

由于单例的静态特性使得其生命周期和应用的生命周期一样长，如果一个对象已经不再需要使用了，而单例对象还持有该对象的引用，就会使得该对象不能被正常回收，从



而导致了内存泄漏。

示例：防止单例导致内存泄漏的实例

```
// 使用了单例模式
public class AppManager {
    private static AppManager instance;
    private Context context;
    private AppManager(Context context) {
        this.context = context;
    }
    public static AppManager getInstance(Context context) {
        if (instance != null) {
            instance = new AppManager(context);
        }
        return instance;
    }
}
```

这样不管传入什么Context最终将使用Application的Context，而单例的生命周期和应用的一样长，这样就防止了内存泄漏。？？？

2、非静态内部类创建静态实例造成的内存泄漏

例如，有时候我们可能会在启动频繁的Activity中，为了避免重复创建相同的数据资源，可能会出现如下写法：

```
public class MainActivity extends AppCompatActivity {

    private static TestResource mResource = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if(mResource == null){
            mResource = new TestResource();
        }
        //...
    }

    class TestResource {
        //...
    }
}
```

这样在Activity内部创建了一个非静态内部类的单例，每次启动Activity时都会使用该单例的数据。虽然这样避免了资源的重复创建，但是这种写法却会造成内存泄漏。因为非

静态内部类默认会持有外部类的引用，而该非静态内部类又创建了一个静态的实例，该实例的生命周期和应用的一样长，这就导致了该静态实例一直会持有该Activity的引用，从而导致Activity的内存资源不能被正常回收。

解决方法：将该内部类设为静态内部类或将该内部类抽取出来封装成一个单例，如果需要使用Context，就使用Application的Context。

3、Handler造成的内存泄漏

示例：创建匿名内部类的静态对象

```
public class MainActivity extends AppCompatActivity {

    private final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            // ...
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        new Thread(new Runnable() {
            @Override
            public void run() {
                // ...
                handler.sendMessage(0x123);
            }
        });
    }
}
```

1、从Android的角度

当Android应用程序启动时，该应用程序的主线程会自动创建一个Looper对象和与之关联的MessageQueue。当主线程中实例化一个Handler对象后，它就会自动与主线程Looper的MessageQueue关联起来。所有发送到MessageQueue的Message都会持有Handler的引用，所以Looper会据此回调Handler的handleMessage()方法来处理消息。只要MessageQueue中有未处理的Message，Looper就会不断的从中取出并交给Handler处理。另外，主线程的Looper对象会伴随该应用程序的整个生命周期。

2、Java角度

在Java中，非静态内部类和匿名类内部类都会潜在持有它们所属的外部类的引用，但是静态内部类却不会。

对上述的示例进行分析，当MainActivity结束时，未处理的消息持有handler的引用，而handler又持有它所属的外部类也就是MainActivity的引用。这条引用关系会一直保持直到消息得到处理，这样阻止了MainActivity被垃圾回收器回收，从而造成了内存泄漏。
解决方法：将Handler类独立出来或者使用静态内部类，这样便可以避免内存泄漏。

4、线程造成的内存泄漏

示例：AsyncTask和Runnable

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        new Thread(new MyRunnable()).start();
        new MyAsyncTask(this).execute();
    }

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

        // ...

        public MyAsyncTask(Context context) {
            // ...
        }

        @Override
        protected Void doInBackground(Void... params) {
            // ...
            return null;
        }

        @Override
        protected void onPostExecute(Void aVoid) {
            // ...
        }
    }

    class MyRunnable implements Runnable {
        @Override
        public void run() {
            // ...
        }
    }
}
```

AsyncTask和Runnable都使用了匿名内部类，那么它们将持有其所在Activity的隐式引

用。如果任务在Activity销毁之前还未完成，那么将导致Activity的内存资源无法被回收，从而造成内存泄漏。

解决方法：将AsyncTask和Runnable类独立出来或者使用静态内部类，这样便可以避免内存泄漏。

5、资源未关闭造成的内存泄漏

对于使用

了BroadcastReceiver, ContentObserver, File, Cursor, Stream, Bitmap等资源，应该在Activity销毁时及时关闭或者注销，否则这些资源将不会被回收，从而造成内存泄漏。

- 1) 比如在Activity中register了一个BroadcastReceiver，但在Activity结束后没有unregister该BroadcastReceiver。
- 2) 资源性对象比如Cursor, Stream、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，往往会造成内存泄漏。
- 3) 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，然后再设置为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。
- 4) Bitmap对象不在使用时调用recycle()释放内存。2.3以后的bitmap应该是不需要手动recycle了，内存已经在java层了。

6、使用ListView时造成的内存泄漏

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Item的View对象会被回收，然后被用来构造新出现在下面的Item。这个构造过程就是由getView()方法完成的，getView()的第二个形参convertView就是被缓存起来的Item的View对象（初始化时缓存中没有View对象则convertView是null）。

构造Adapter时，没有使用缓存的convertView。

解决方法：在构造Adapter时，使用缓存的convertView。

7、集合容器中的内存泄露

我们通常把一些对象的引用加入到了集合容器（比如ArrayList）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。

解决方法：在退出程序之前，将集合里的东西clear，然后置为null，再退出程序。

8、WebView造成的泄露

当我们不要使用WebView对象时，应该调用它的destory()函数来销毁它，并释放其占用的内存，否则其长期占用的内存也不能被回收，从而造成内存泄露。

解决方法：为WebView另外开启一个进程，通过AIDL与主线程进行通信，WebView所在的进程可以根据业务的需要选择合适的时机进行销毁，从而达到内存的完整释放。

如何避免内存泄漏？

1、在涉及使用Context时，对于生命周期比Activity长的对象应该使用Application的Context。凡是使用Context优先考虑Application的Context，当然它并不是万能的，对于有些地方则必须使用Activity的Context。对于Application，Service，Activity三者的Context的应用场景如下：

其中，NO1表示Application和Service可以启动一个Activity，不过需要创建一个新的task任务队列。而对于Dialog而言，只有在Activity中才能创建。除此之外三者都可以使用。

2、对于需要在静态内部类中使用非静态外部成员变量（如：Context、View），可以在静态内部类中使用弱引用米引用外部类的变量来避免内存泄漏。

3、对于不再需要使用的对象，显示的将其赋值为null，比如使用完Bitmap后先调用recycle()，再赋为null。

4、保持对对象生命周期的敏感，特别注意单例、静态对象、全局性集合等的生命周期。

5、对于生命周期比Activity长的内部类对象，并且内部类中使用了外部类的成员变量，可以这样做避免内存泄漏：

- 1) 将内部类改为静态内部类
- 2) 静态内部类中使用弱引用来引用外部类的成员变量

参考

Android内存泄漏总结

Android开发

举报文章 © 著作权归作者所有



李俊的博客

写了 14758 字，被 44 人关注，获得了 87 个喜欢

关注

Android开发技术笔记！


喜欢 | 22

更多分享





下载简书 App ▶


随时随地发现和创作内容




被以下专题收入，发现更多相似内容

 Android面试

 内存泄露

 安卓性能优化

 优秀文章

Part1_Android内存泄漏总结

Android 内存泄漏总结 内存管理的目的就是让我们在开发中怎么有效的避免我们的应用出现内存泄漏的问题。内存泄漏大家都不陌生了，简单粗俗的讲...

 _痞子

Application	
NO1	NO1
NO	NO
YES	YES
YES	YES
YES	YES
YES	YES
YES	YES
YES	YES
YES	YES

java与android 内存泄漏总结


内存管理的目的就是让我们在开发中怎么有效的避免我们的应用出现内存泄漏的问题。内存泄漏大家都不陌生了，简单粗俗的讲，就是该被释放的对象没...

 宇宙只有巴掌大

	Application	
	NO1	NO1
	NO	NO
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES

Android 内存泄漏总结


Android 内存泄漏总结 内存管理的目的就是让我们在开发中怎么有效的避免我们的应用出现内存泄漏的问题。内存泄漏大家都不陌生了，简单粗俗的讲...

 apkcore

	Application	
	NO1	NO1
	NO	NO
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES

Android 内存泄漏总结


内存管理的目的就是让我们在开发中怎么有效的避免我们的应用出现内存泄漏的问题。内存泄漏大家都不陌生了，简单粗俗的讲，就是该被释放的对象没...

 DreamFish



Android中常见的内存泄漏汇总

###集合类泄漏 集合类如果仅仅有添加元素的方法，而没有相应的删除机制，导致内存被占用。如果这个集合类是全局性的变量 (比如类中的静态属性...

 RunningTeemo

	Application	
	NO1	NO1
	NO	NO
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES
	YES	YES


每日一词 尖刻的，爱挖苦的。

 湖边娱乐办公室郑谦毅



开封记

一年春节，一次庙会。早闻开封的庙会的繁闹，便早早的与朋友去了大宋武侠城万岁山的春节庙会，不想竟如此热闹，古色古香的开封城中过年浓浓的气息...

 九命猫书



深耕to B市场，酷开VR再获CITE创新产品奖！

4月9日晚，VR行业传来好消息，在深圳会展中心举办的第五届中国电子信息

博览会(简称CITE 2017)创新之夜上，酷开VR一体机随意门获得...



互联网IT



说说理想中的爱情

临高考，有同学问我：你的梦想是什么？我俩当时正在打篮球，我想也不想就把球传给他并大声回答说：想做一大厨，然后做出各种美食，抓住一个好男人的...



潺禅

