[**Java**](http://lib.csdn.net/base/javase)**基础——内存泄露、内存溢出**

(2010-09-15 15:56:26)

**要点**

* 内存泄露是指程序中间动态分配了内存，但在程序结束时没有释放这部分内存，从而造成那部分内存不可用的情况，重启计算机可以解决，但也有可能再次发生内存泄露，内存泄露和硬件没有关系，它是由软件设计缺陷引起的。
* 内存泄漏可以分为4类：

1) 常发性内存泄漏。发生内存泄漏的代码会被多次执行到，每次被执行的时候都会导致一块内存泄漏。

2) 偶发性内存泄漏。发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生。常发性和偶发性是相对的。对于特定的环境，偶发性的也许就变成了常发性的。所以[**测试**](http://lib.csdn.net/base/softwaretest)环境和测试方法对检测内存泄漏至关重要。

3) 一次性内存泄漏。发生内存泄漏的代码只会被执行一次，或者由于[**算法**](http://lib.csdn.net/base/datastructure)上的缺陷，导致总会有一块仅且一块内存发生泄漏。比如，在类的构造函数中分配内存，在析构函数中却没有释放该内存，所以内存泄漏只会发生一次。

4) 隐式内存泄漏。程序在运行过程中不停的分配内存，但是直到结束的时候才释放内存。严格的说这里并没有发生内存泄漏，因为最终程序释放了所有申请的内存。但是对于一个服务器程序，需要运行几天，几周甚至几个月，不及时释放内存也可能导致最终耗尽系统的所有内存。所以，我们称这类内存泄漏为隐式内存泄漏。

* 内存溢出类型：

1) java.lang.OutOfMemoryError: PermGen space

PermGen space 的全称是 Permanent Generation space, 是指内存的永久保存区域。这块内存主要是被JVM存放Class和Meta信息的，Class在被Loader时就会被放到PermGenspace中，它和存放类实例(Instance)的Heap区域不同，GC不会在主程序运行期对PermGen space进行清理。

JVM由XX:PermSize设置非堆内存初始值，默认是物理内存的1/64；

JVM由XX:MaxPermSize设置最大非堆内存的大小，默认是物理内存的1/4。

**该错误常见场合：**

a) 应用中有很多Class，web服务器对JSP进行precompile时。

b) Webapp下用了大量的第三方jar,其大小超过了JVM默认的大小(4M)时。

2) java.lang.OutOfMemoryError:[**Java**](http://lib.csdn.net/base/java)heap space （堆空间）

在JVM中如果98%的时间是用于GC且可用的Heap size 不足2%的时候将抛出此异常信息。

JVM初始分配的内存由-Xms指定，默认是物理内存的1/64；

JVM最大分配的内存由-Xmx指定，默认是物理内存的1/4。

JVM内存的最大值跟[**操作系统**](http://lib.csdn.net/base/operatingsystem)有很大的关系。32位处理器虽然可控内存空间有4GB，但是具体的操作系统会给一个限制，这个限制一般是2GB-3GB（一般来说Windows系统下为1.5G-2G，[**Linux**](http://lib.csdn.net/base/linux)系统下为2G-3G），而64bit以上的处理器就不会有限制了。

*注意：如果Xms超过了Xmx值，或者堆最大值和非堆最大值的总和超过了物理内存或者操作系统的最大限制都会引起服务器启动不起来。*

**该错误常见场合：**

a) Web上传文件时。

b) 开启大型文件或从[**数据库**](http://lib.csdn.net/base/mysql)中一次取了太多的数据。 

**相关问题**

**1. Q: Java中会存在内存泄漏吗？** A:  Java中也存在内存泄露。当被分配的对象可达但已无用（未对作废数据内存单元的引用置null）即会引起。

         如：

Java代码

1. Vector v=**new** Vector(10);
2. **for** (**int** i=1;i<100; i ) {
3. Object o=**new** Object();
4. v.add(o);
5. o=**null**;
6. }
7. // 此时，所有的Object对象都没有被释放，因为变量v引用这些对象。
8. // 对象加入到Vector后，还必须从Vector中删除，最简单释放方法就是将Vector对象设置为null。

Vector v=new Vector(10); for(int i=1;i<100; i ) { Object o=new Object(); v.add(o); o=null; } // 此时，所有的Object对象都没有被释放，因为变量v引用这些对象。 // 对象加入到Vector后，还必须从Vector中删除，最简单释放方法就是将Vector对象设置为null。

**2. Q: 内存泄露、溢出的异同？**

     A: 同：都会导致应用程序运行出现问题，性能下降或挂起。

         异：

         1) 内存泄露是导致内存溢出的原因之一；内存泄露积累起来将导致内存溢出。

         2) 内存泄露可以通过完善代码来避免；内存溢出可以通过调整配置来减少发生频率，但无法彻底避免。

**3. 如何检测内存泄露？**

 A: 可以通过一些性能监测分析工具，如 JProfiler、Optimizeit Profiler。

**4. Q: 如何避免内存泄露、溢出？** A: 1) 尽早释放无用对象的引用。

          好的办法是使用临时变量的时候，让引用变量在退出活动域后自动设置为null，暗示垃圾收集器来收集该对象，防止发生内存泄露。

          2) 程序进行字符串处理时，尽量避免使用String，而应使用StringBuffer。

          因为每一个String对象都会独立占用内存一块区域，如：

Java代码

1. String str = "aaa";
2. String str2 = "bbb";
3. String str3 = str   str2;
4. // 假如执行此次之后str , str2再不被调用，那么它们就会在内存中等待GC回收；
5. // 假如程序中存在过多的类似情况就会出现内存错误；

String str = "aaa";String str2 = "bbb"; String str3 = str str2; // 假如执行此次之后str , str2再不被调用，那么它们就会在内存中等待GC回收； // 假如程序中存在过多的类似情况就会出现内存错误；

         3) 尽量少用静态变量。

         因为静态变量是全局的，GC不会回收。

         4) 避免集中创建对象尤其是大对象，如果可以的话尽量使用流操作。

         JVM会突然需要大量内存，这时会触发GC优化系统内存环境； 一个案例如下：

Java代码

1. // 使用jspsmartUpload作文件上传，运行过程中经常出现java.outofMemoryError的错误，
2. // 检查之后发现问题：组件里的代码
3. m\_totalBytes = m\_request.getContentLength();
4. m\_binArray = **new byte**[m\_totalBytes];
5. // totalBytes这个变量得到的数极大，导致该数组分配了很多内存空间，而且该数组不能及时释放。
6. // 解决办法只能换一种更合适的办法，至少是不会引发outofMemoryError的方式解决。
7. // 参考：http://bbs.xml.org.cn/blog/more.asp?name=hongrui&id=3747

// 使用jspsmartUpload作文件上传，运行过程中经常出现java.outofMemoryError的错误， // 检查之后发现问题：组件里的代码 m\_totalBytes = m\_request.getContentLength(); m\_binArray = newbyte[m\_totalBytes]; // totalBytes这个变量得到的数极大，导致该数组分配了很多内存空间，而且该数组不能及时释放。 // 解决办法只能换一种更合适的办法，至少是不会引发outofMemoryError的方式解决。 // 参考：http://bbs.xml.org.cn/blog/more.asp?name=hongrui&id=3747

        5) 尽量运用对象池技术以提高系统性能。

         生命周期长的对象拥有生命周期短的对象时容易引发内存泄漏，例如大集合对象拥有[**大数据**](http://lib.csdn.net/base/hadoop)量的业务对象的时候，可以考虑分块进行处理，然后解决一块释放一块的策略。

         6) 不要在经常调用的方法中创建对象，尤其是忌讳在循环中创建对象。

         可以适当的使用hashtable，vector 创建一组对象容器，然后从容器中去取那些对象，而不用每次new之后又丢弃。

         7) 优化配置。

**5. Q: 内存溢出的解决方案？** A: 一是从代码层面进行优化完善，尽量避免该情况发生；

         二是调整优化服务器配置：

         1) 设置-Xms、-Xmx相等；

         2) 设置NewSize、MaxNewSize相等；

         3) 设置Heap size, PermGen space:

            Tomcat 的配置示例：修改%TOMCAT\_HOME%/bin/catalina.bat or catalina.sh

             在“echo "Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

Cmd代码

1. set JAVA\_OPTS=-Xms800m -Xmx800m -XX:PermSize=128M -XX:MaxNewSize=256m -XX:MaxPermSize=256m

-----------------------------/////////////////////////////

[**内存泄露与溢出区别，何时产生内存泄露？**](http://lz12366.javaeye.com/blog/683468)

**文章分类:**[**Java编程**](http://www.javaeye.com/blogs/category/java)

内存溢出是由于jvm虚拟内存不够！   
而泄漏则是另种概念！1   
  
下面实在摘抄的内容！！   
  
下面，我们就可以描述什么是内存泄漏。在Java中，内存泄漏就是存在一些被分配的对象，这些对象有下面两个特点，首先，这些对象是可达的，即在有向图中，存在通路可以与其相连；其次，这些对象是无用的，即程序以后不会再使用这些对象。如果对象满足这两个条件，这些对象就可以判定为Java中的内存泄漏，这些对象不会被GC所回收，然而它却占用内存。   
  
在C++中，内存泄漏的范围更大一些。有些对象被分配了内存空间，然后却不可达，由于C++中没有GC，这些内存将永远收不回来。在Java中，这些不可达的对象都由GC负责回收，因此程序员不需要考虑这部分的内存泄露。   
  
通过分析，我们得知，对于C++，程序员需要自己管理边和顶点，而对于Java程序员只需要管理边就可以了(不需要管理顶点的释放)。通过这种方式，Java提高了编程的效率。   
  
  
因此，通过以上分析，我们知道在Java中也有内存泄漏，但范围比C++要小一些。因为Java从语言上保证，任何对象都是可达的，所有的不可达对象都由GC管理。   
  
对于程序员来说，GC基本是透明的，不可见的。虽然，我们只有几个函数可以访问GC，例如运行GC的函数System.gc()，但是根据Java语言规范定义， 该函数不保证JVM的垃圾收集器一定会执行。因为，不同的JVM实现者可能使用不同的算法管理GC。通常，GC的线程的优先级别较低。JVM调用GC的策略也有很多种，有的是内存使用到达一定程度时，GC才开始工作，也有定时执行的，有的是平缓执行GC，有的是中断式执行GC。但通常来说，我们不需要关心这些。除非在一些特定的场合，GC的执行影响应用程序的性能，例如对于基于Web的实时系统，如网络游戏等，用户不希望GC突然中断应用程序执行而进行垃圾回收，那么我们需要调整GC的参数，让GC能够通过平缓的方式释放内存，例如将垃圾回收分解为一系列的小步骤执行，Sun提供的HotSpot JVM就支持这一特性。   
  
下面给出了一个简单的内存泄露的例子。在这个例子中，我们循环申请Object对象，并将所申请的对象放入一个Vector中，如果我们仅仅释放引用本身，那么Vector仍然引用该对象，所以这个对象对GC来说是不可回收的。因此，如果对象加入到Vector后，还必须从Vector中删除，最简单的方法就是将Vector对象设置为null。

Java代码

1. Vector v=new Vector(10);
2. for (int i=1;i<100; i++)
3. {
4. Object o=new Object();
5. v.add(o);
6. o=null;
7. }

Vector v=new Vector(10);

for (int i=1;i<100; i++)

{

  Object o=new Object();

  v.add(o);

  o=null;

}

//此时，所有的Object对象都没有被释放，因为变量v引用这些对象

**怎样解决内存溢出**

　一、内存溢出类型

　　1、java.lang.OutOfMemoryError:PermGen space

　　JVM管理两种类型的内存，堆和非堆。堆是给开发人员用的上面说的就是，是在JVM启动时创建;非堆是留给JVM自己用的，用来存放类的信息的。它和堆不同，运行期内GC不会释放空间。如果web app用了大量的第三方[jar](http://www.chinabyte.com/keyword/JAR/)或者应用有太多的class文件而恰好MaxPermSize设置较小，超出了也会导致这块内存的占用过多造成溢出，或者tomcat热部署时侯不会清理前面加载的环境，只会将context更改为新部署的，非堆存的内容就会越来越多。

　　PermGen space的全称是PermanentGeneration space,是指内存的永久保存区域，这块内存主要是被JVM存放Class和Meta信息的,Class在被Loader时就会被放到PermGen space中，它和存放类实例(Instance)的Heap区域不同,GC(GarbageCollection)不会在主程序运行期对PermGen space进行清理，所以如果你的应用中有很CLASS的话,就很可能出现PermGenspace错误，这种错误常见在[web服务器](http://www.chinabyte.com/keyword/web%E6%9C%8D%E5%8A%A1%E5%99%A8/)对JSP进行pre compile的时候。如果你的WEB APP下都用了大量的第三方jar, 其大小超过了jvm默认的大小(4M)那么就会产生此错误信息了。

　　一个最佳的配置例子：(经过本人验证，自从用此配置之后，再未出现过tomcat死掉的情况)

　　set JAVA\_OPTS=-Xms800m -Xmx800m -XX:PermSize=128M-XX:MaxNewSize=256m -XX:MaxPermSize=256m

　　2、java.lang.OutOfMemoryError:Javaheap space

　　第一种情况是个补充，主要存在问题就是出现在这个情况中。其默认空间(即-Xms)是物理内存的1/64，最大空间(-Xmx)是物理内存的1/4。如果内存剩余不到40%，JVM就会增大堆到Xmx设置的值，内存剩余超过70%，JVM就会减小堆到Xms设置的值。所以[服务器](http://server.chinabyte.com/)的Xmx和Xms设置一般应该设置相同避免每次GC后都要调整虚拟机堆的大小。假设物理内存无限大，那么JVM内存的最大值跟[操作系统](http://soft.chinabyte.com/os/)有关，一般32位机是1.5g到[3g](http://www.chinabyte.com/keyword/3G/)之间，而64位的就不会有限制了。

　　注意：如果Xms超过了Xmx值，或者堆最大值和非堆最大值的总和超过了物理内存或者操作系统的最大限制都会引起服务器启动不起来。

　　垃圾回收GC的角色

　　JVM调用GC的频度还是很高的，主要两种情况下进行垃圾回收：

　　当应用程序线程空闲;另一个是java内存堆不足时，会不断调用GC，若连续回收都解决不了内存堆不足的问题时，就会报out of memory错误。因为这个异常根据系统运行环境决定，所以无法预期它何时出现。

　　根据GC的机制，程序的运行会引起系统运行环境的变化，增加GC的触发机会。

　　为了避免这些问题，程序的设计和编写就应避免垃圾对象的内存占用和GC的开销。显示调用System.GC()只能建议JVM需要在内存中对垃圾对象进行回收，但不是必须马上回收，

　　一个是并不能解决内存资源耗空的局面，另外也会增加GC的消耗。

　　二、JVM内存区域组成

　　简单的说java中的堆和栈

　　java把内存分两种：一种是栈内存，另一种是堆内存

　　1。在函数中定义的基本类型变量和对象的引用变量都在函数的栈内存中分配;

　　2。堆内存用来存放由new创建的对象和数组

　　在函数(代码块)中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量所分配的内存空间;在堆中分配的内存由java虚拟机的自动垃圾回收器来管理

　　堆的优势是可以动态分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的。缺点就是要在运行时动态分配内存，存取速度较慢;

　　栈的优势是存取速度比堆要快，缺点是存在栈中的数据大小与生存期必须是确定的无灵活性。

　　java堆分为三个区：New、Old和Permanent

　　GC有两个线程：

　　新创建的对象被分配到New区，当该区被填满时会被GC辅助线程移到Old区，当Old区也填满了会触发GC主线程遍历堆内存里的所有对象。Old区的大小等于Xmx减去-Xmn

　　java栈存放

　　栈调整：参数有+UseDefaultStackSize -Xss256K，表示每个线程可申请256k的栈空间

　　每个线程都有他自己的Stack

　　三、JVM如何设置虚拟内存

　　提示：在JVM中如果98%的时间是用于GC且可用的Heap size 不足2%的时候将抛出此异常信息。

　　提示：Heap Size 最大不要超过可用物理内存的80%，一般的要将-Xms和-Xmx选项设置为相同，而-Xmn为1/4的-Xmx值。

　　提示：JVM初始分配的内存由-Xms指定，默认是物理内存的1/64;JVM最大分配的内存由-Xmx指定，默认是物理内存的1/4。

　　默认空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制;空余堆内存大于70%时，JVM会减少堆直到-Xms的最小限制。因此服务器一般设置-Xms、-Xmx相等以避免在每次GC 后调整堆的大小。

　　提示：假设物理内存无限大的话，JVM内存的最大值跟操作系统有很大的关系。

　　简单的说就32位[处理器](http://www.chinabyte.com/keyword/%E5%A4%84%E7%90%86%E5%99%A8/)虽然可控内存空间有4GB,但是具体的操作系统会给一个限制，

　　这个限制一般是2GB-3GB(一般来说Windows系统下为1.5G-2G，[Linux](http://www.chinabyte.com/keyword/Linux/)系统下为2G-3G)，而64bit以上的处理器就不会有限制了

　　提示：注意：如果Xms超过了Xmx值，或者堆最大值和非堆最大值的总和超过了物理内存或者操作系统的最大限制都会引起服务器启动不起来。

　　提示：设置NewSize、MaxNewSize相等，"new"的大小最好不要大于"old"的一半，原因是old区如果不够大会频繁的触发"主" GC ，大大降低了性能

　　JVM使用-XX:PermSize设置非堆内存初始值，默认是物理内存的1/64;

　　由XX:MaxPermSize设置最大非堆内存的大小，默认是物理内存的1/4。

　　解决方法：手动设置Heap size

　　修改TOMCAT\_HOME/bin/catalina.bat

　　在“echo "Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

　　JAVA\_OPTS="-server -Xms800m -Xmx800m -XX:MaxNewSize=256m"

　　四、性能检查工具使用

　　定位内存泄漏：

　　JProfiler工具主要用于检查和跟踪系统(限于Java开发的)的性能。JProfiler可以通过时时的监控系统的内存使用情况，随时监视垃圾回收，线程运行状况等手段，从而很好的监视JVM运行情况及其性能。

　　1.[应用服务](http://www.chinabyte.com/keyword/%E5%BA%94%E7%94%A8%E6%9C%8D%E5%8A%A1/)器内存长期不合理占用，内存经常处于高位占用，很难回收到低位;

　　2. 应用服务器极为不稳定，几乎每两天重新启动一次，有时甚至每天重新启动一次;

　　3. 应用服务器经常做Full GC(GarbageCollection)，而且时间很长，大约需要30-40秒，应用服务器在做Full GC的时候是不响应客户的交易请求的，非常影响系统性能。

　　因为开发环境和产品环境会有不同，导致该问题发生有时会在产品环境中发生，通常可以使用工具跟踪系统的内存使用情况，在有些个别情况下或许某个时刻确实是使用了大量内存导致out of memory，这时应继续跟踪看接下来是否会有下降，

　　如果一直居高不下这肯定就因为程序的原因导致内存泄漏。

　　五、不健壮代码的特征及解决办法

　　1、尽早释放无用对象的引用。好的办法是使用临时变量的时候，让引用变量在退出活动域后，自动设置为null，暗示垃圾收集器来收集该对象，防止发生内存泄露。

　　对于仍然有指针指向的实例，jvm就不会回收该资源,因为垃圾回收会将值为null的对象作为垃圾，提高GC回收机制效率;

　　2、我们的程序里不可避免大量使用字符串处理，避免使用String，应大量使用StringBuffer，每一个String对象都得独立占用内存一块区域;

　　String str = "aaa";

　　String str2 = "bbb";

　　String str3 = str + str2;//假如执行此次之后str ,str2以后再不被调用,那它就会被放在内存中等待Java的gc去回收,程序内过多的出现这样的情况就会报上面的那个错误,建议在使用字符串时能使用StringBuffer就不要用String,这样可以省不少开销;

　　3、尽量少用静态变量，因为静态变量是全局的，GC不会回收的;

　　4、避免集中创建对象尤其是大对象，JVM会突然需要大量内存，这时必然会触发GC优化系统内存环境;显示的声明数组空间，而且申请数量还极大。

　　这是一个案例想定供大家警戒

　　使用jspsmartUpload作文件上传,运行过程中经常出现java.outofMemoryError的错误，

　　检查之后发现问题：组件里的代码

　　m\_totalBytes = m\_request.getContentLength();

　　m\_binArray = new byte[m\_totalBytes];

　　问题原因是totalBytes这个变量得到的数极大，导致该数组分配了很多内存空间，而且该数组不能及时释放。解决办法只能换一种更合适的办法，至少是不会引发outofMemoryError的方式解决。参考：http://bbs.xml.org.cn/blog/more.asp?name=hongrui&id=3747

　　5、尽量运用对象池技术以提高系统性能;生命周期长的对象拥有生命周期短的对象时容易引发内存泄漏，例如大集合对象拥有大数据量的业务对象的时候，可以考虑分块进行处理，然后解决一块释放一块的策略。

　　6、不要在经常调用的方法中创建对象，尤其是忌讳在循环中创建对象。可以适当的使用hashtable，vector 创建一组对象容器，然后从容器中去取那些对象，而不用每次new之后又丢弃

　　7、一般都是发生在开启大型文件或跟[数据库](http://com.chinabyte.com/%E6%95%B0%E6%8D%AE%E5%BA%93/)一次拿了太多的数据，造成 Out OfMemory Error 的状况，这时就大概要计算一下数据量的最大值是多少，并且设定所需最小及最大的内存空间值。