# 内存溢出(out of memory)

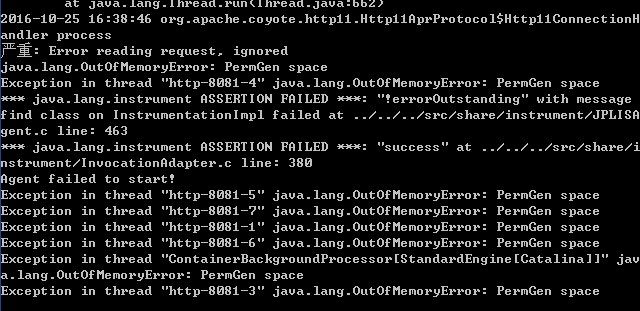
通俗理解就是内存不够，通常在运行大型软件或游戏时，软件或游戏所需要的内存远远超出了你主机内安装的内存所承受大小，就叫内存溢出。此时软件或游戏就运行不了，系统会提示内存溢出，有时候会自动关闭软件，重启电脑或者软件后释放掉一部分内存又可以正常运行该软件或游戏一段时间。

## 内存溢出现象：

第一种OutOfMemoryError： PermGen space

发生这种问题的原意是程序中使用了大量的jar或class，使java虚拟机装载类的空间不够，与Permanent Generation space有关。解决这类问题有以下两种办法：

增加java虚拟机中的XX:PermSize和XX:MaxPermSize参数的大小，其中XX:PermSize是初始永久保存区域大小，XX:MaxPermSize是最大永久保存区域大小。如针对tomcat6.0，在catalina.sh 或catalina.bat文件中一系列环境变量名说明结束处（大约在70行左右） 增加一行： JAVA\_OPTS=" -XX:PermSize=64M -XX:MaxPermSize=128m" 如果是windows服务器还可以在系统环境变量中设置。感觉用tomcat发布sprint+struts+hibernate架构的程序时很容易发生这种内存溢出错误。使用上述方法，我成功解决了部署ssh项目的tomcat服务器经常宕机的问题。



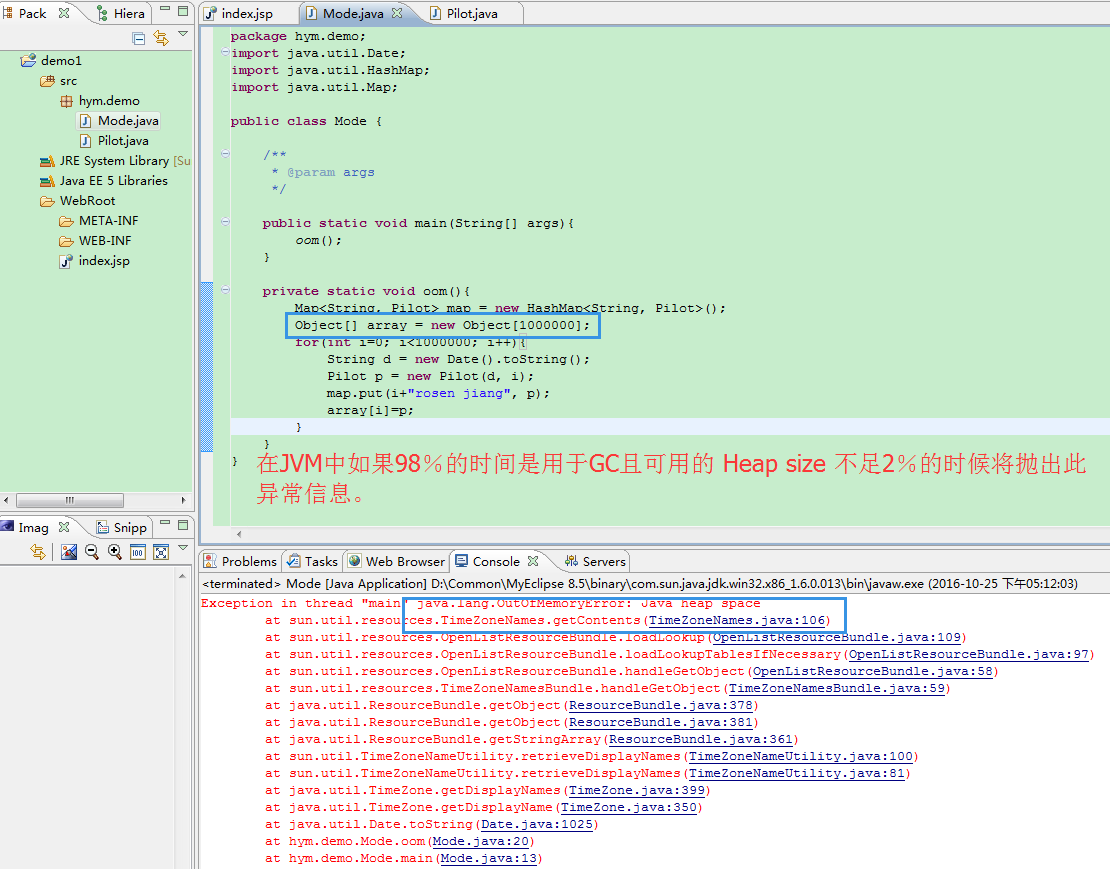
清理应用程序中web-inf/lib下的jar，如果tomcat部署了多个应用，很多应用都使用了相同的jar，可以将共同的jar移到tomcat共同的lib下，减少类的重复加载。这种方法是网上部分人推荐的，我没试过，但感觉减少不了太大的空间，最靠谱的还是第一种方法。

第二种OutOfMemoryError： Java heap space

发生这种问题的原因是java虚拟机创建的对象太多，在进行垃圾回收之间，虚拟机分配的到堆内存空间已经用满了，与Heap space有关。解决这类问题有两种思路：

检查程序，看是否有死循环或不必要地重复创建大量对象。找到原因后，修改程序和算法。 我以前写一个使用K-Means文本聚类算法对几万条文本记录（每条记录的特征向量大约10来个）进行文本聚类时，由于程序细节上有问题，就导致了Java heap space的内存溢出问题，后来通过修改程序得到了解决。

增加Java虚拟机中Xms（初始堆大小）和Xmx（最大堆大小）参数的大小。如：set JAVA\_OPTS= -Xms256m -Xmx512m -XX:PermSize=64m -XX:MaxPermSize=128m



第三种OutOfMemoryError：unable to create new native thread

在java应用中，有时候会出现这样的错误：OutOfMemoryError: unable to create new native thread.这种怪事是因为JVM已经被系统分配了大量的内存(比如1.5G)，并且它至少要占用可用内存的一半。

## 出现的原因

引起内存溢出的原因有很多种，常见的有以下几种:

 1、内存中加载的数据量过于庞大，如一次从数据库取出过多数据;

 2、集合类中有对对象的引用，使用完后未清空，使得JVM不能回收;

 3、代码中存在死循环或循环产生过多重复的对象实体;

 4、使用的第三方软件中的BUG;

 5、启动参数设定的过小;

## 解决办法：

第一步，就是修改JVM启动参数，直接增加内存。

第二步，检查错误日志，查看"OutOfMemory"错误前是否有其它异常或错误。

第三步，安排有经验的编程人员对代码进行走查和分析，找出可能发生内存溢出的位置。

第四步，使用内存查看工具动态查看内存使用情况。

## MAT检查：

1. 生成dump
2. 找出溢出源

■Dominator Tree（支配树）：列出每个对象(Object instance)与其引用关系的树状结构，还包含了占有多大内存，所占百分比

■Histogram视图（截图里柱子那个，边上的是Dominator Tree ）：列出每個class产生了多少個实例，以及占有多大内存，所占百分比

■可以很容易找出站内存最多的几个类，根据Retained Heap排序，找出前几个。

■可以分不同的维度来查看类的Histogram视图，Group by class、Group by superclass、Group by class loader、Group by package

■只要有溢出，时间久了，溢出类的实例数量或者其占有的内存会越来越多，排名也就越来越前，通过多次对比不同时间点下的Histogram图对比就能很容易把溢出类找出来。

3、定位溢出源

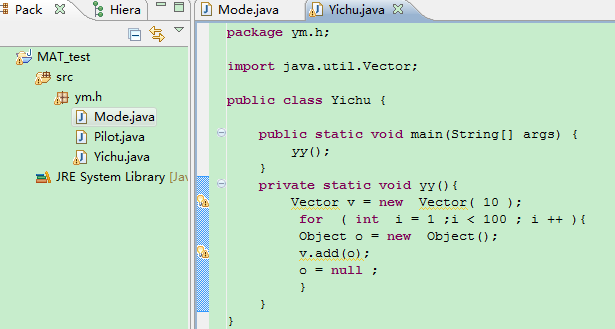
通过Pat通过Path to GC Roots或者Merge Shortest Paths to GC Roots

# 内存泄漏

也称作"存储渗漏"，用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元。直到程序结束。(其实说白了就是该内存空间使用完毕之后未回收)即所谓内存泄漏。

内存泄漏形象的比喻是"操作系统可提供给所有进程的存储空间正在被某个进程榨干"，最终结果是程序运行时间越长，占用存储空间越来越多，最终用尽全部存储空间，整个系统崩溃。所以"内存泄漏"是从操作系统的角度来看的。这里的存储空间并不是指物理内存，而是指虚拟内存大小，这个虚拟内存大小取决于磁盘交换区设定的大小。由程序申请的一块内存，如果没有任何一个指针指向它，那么这块内存就泄漏了。

一般来说内存泄漏有两种情况。一种情况如在C/C++ 语言中的，在堆中的分配的内存，在没有将其释放掉的时候，就将所有能访问这块内存的方式都删掉（如指针重新赋值）；另一种情况则是在内存对象明明已经不需要的时候，还仍然保留着这块内存和它的访问方式（引用）。第一种情况，在 Java 中已经由于垃圾回收机制的引入，得到了很好的解决。所以， Java 中的内存泄漏，主要指的是第二种情况。



实例解释：在这个例子中，代码栈中存在Vector 对象的引用 v 和 Object 对象的引用 o 。在 For 循环中，我们不断的生成新的对象，然后将其添加到 Vector 对象中，之后将 o 引用置空。问题是当 o 引用被置空后，如果发生 GC ，我们创建的 Object 对象是否能够被 GC 回收呢？答案是否定的。因为， GC 在跟踪代码栈中的引用时，会发现 v 引用，而继续往下跟踪，就会发现 v 引用指向的内存空间中又存在指向 Object 对象的引用。也就是说尽管 o 引用已经被置空，但是 Object 对象仍然存在其他的引用，是可以被访问到的，所以 GC 无法将其释放掉。如果在此循环之后， Object 对象对程序已经没有任何作用，那么我们就认为此 Java 程序发生了内存泄漏。