

# GalaxyPilot —— D.S

生命不熄，战斗不止      QQ: 8117892

## 导航

BlogJava  
首页  
新随笔  
联系  
XML 聚合  
管理

< 2006年4月 >						
日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

## 常用链接

我的随笔  
我的评论  
我的参与  
最新评论

## 留言簿(45)

给我留言  
查看公开留言  
查看私人留言

## 随笔档案(65)

2011年6月 (1)  
2010年12月 (1)  
2010年4月 (2)  
2009年11月 (1)  
2009年3月 (2)  
2009年2月 (1)  
2008年12月 (1)  
2008年11月 (1)  
2008年9月 (3)  
2008年4月 (1)  
2008年1月 (2)  
2007年11月 (1)  
2007年10月 (2)  
2007年6月 (2)  
2007年3月 (3)  
2007年2月 (2)  
2006年12月 (2)  
2006年10月 (2)

## JAVA内存泄漏——内存泄漏原因和内存泄漏检测工具(zt)

### 摘要

虽然Java虚拟机(JVM)及其垃圾收集器(garbage collector，GC)负责管理大多数的内存任务，Java软件程序中还是有可能出现内存泄漏。实际上，这在大型项目中是一个常见的问题。避免内存泄漏的第一步是要弄清楚它是如何发生的。本文介绍了编写Java代码的一些常见的内存泄漏陷阱，以及编写不泄漏代码的一些最佳实践。一旦发生了内存泄漏，要指出造成泄漏的代码是非常困难的。因此本文还介绍了一种新工具，用来诊断泄漏并指出根本原因。该工具的开销非常小，因此可以使用它来寻找处于生产中的系统的内存泄漏。

### 垃圾收集器的作用

虽然垃圾收集器处理了大多数内存管理问题，从而使编程人员的生活变得更轻松了，但是编程人员还是可能犯错而导致出现内存问题。简单地说，GC循环地跟踪所有来自“根”对象（堆栈对象、静态对象、JNI句柄指向的对象，诸如此类）的引用，并将所有它所能到达的对象标记为活动的。程序只可以操纵这些对象；其他的对象都被删除了。因为GC使程序不可能到达已被删除的对象，这么做就是安全的。

虽然内存管理可以说是自动化的，但是这并不能使编程人员免受思考内存管理问题之苦。例如，分配（以及释放）内存总会有开销，虽然这种开销对编程人员来说是不可见的。创建了太多对象的程序将会比完成同样的功能而创建的对象却比较少的程序更慢一些（在其他条件相同的情况下）。

而且，与本文更为密切相关的是，如果忘记“释放”先前分配的内存，就可能造成内存泄漏。如果程序保留对永远不再使用的对象的引用，这些对象将会占用并耗尽内存，这是因为自动化的垃圾收集器无法证明这些对象将不再使用。正如我们先前所说的，如果存在一个对对象的引用，对象就被定义为活动的，因此不能删除。为了确保能回收对象占用的内存，编程人员必须确保该对象不能到达。这通常是通过将对象字段设置为null或者从集合(collection)中移除对象而完成的。但是，注意，当局部变量不再使用时，没有必要将其显式地设置为null。对这些变量的引用将随着方法的退出而自动清除。

概括地说，这就是内存托管语言中的内存泄漏产生的主要原因：保留下来却永远不再使用的对象引用。

### 典型泄漏

既然我们知道了在Java中确实有可能发生内存泄漏，就让我们来看一些典型的内存泄漏及其原因。

### 全局集合

2006年9月 (2)  
2006年8月 (1)  
2006年7月 (1)  
2006年6月 (1)  
2006年5月 (2)  
2006年4月 (13)  
2006年3月 (8)  
2006年2月 (7)

文章档案(1)

2006年7月 (1)

相册


照片

搜索

积分与排名

积分 - 199917

排名 - 93

最新评论 

1. re: FineReport6.5破解[未登录]  
楼主 请发一份给我吧 ambrai  
n1@163.com 谢谢

--ambrain

2. re: FineReport6.5破解[未登录]  
楼主 发一份给我吧 5508144  
0@qq.com

--java

3. re: FineReport6.5破解[未登录]  
求一份破解补丁, 感谢 3834  
75423@qq.com tonybao\_sz  
@163.com

--Tony

4. re: FineReport6.5破解  
求破解文件, 感谢楼主, 请  
发邮箱xiaowu0371@qq.com

--xiaowu0371

5. re: FineReport6.5破解  
楼主, 求共享啊445936705  
@qq.com

--楼主, 求共享啊

6. re: FineReport6.5破解  
楼主学习使用, 能否发一份  
half000@163.com

--half000@163.com

7. re: FineReport6.5破解  
能给我一份破解补丁吗, 感  
谢 hyibing@126.com

在大的应用程序中有某种全局的数据储存库是很常见的，例如一个JNDI树或一个会话表。在这些情况下，必须注意管理储存库的大小。必须有某种机制从储存库中移除不再需要的数据。

这可能有多种方法，但是最常见的一种是周期性运行的某种清除任务。该任务将验证储存库中的数据，并移除任何不再需要的数据。

另一种管理储存库的方法是使用反向链接(referrer)计数。然后集合负责统计集合中每个入口的反向链接的数目。这要求反向链接告诉集合何时会退出口。当反向链接数目为零时，该元素就可以从集合中移除了。

缓存

缓存是一种数据结构，用于快速查找已经执行的操作的结果。因此，如果一个操作执行起来很慢，对于常用的输入数据，就可以将操作的结果缓存，并在下次调用该操作时使用缓存的数据。

缓存通常都是以动态方式实现的，其中新的结果是在执行时添加到缓存中的。典型的算法是：

检查结果是否在缓存中，如果在，就返回结果。

如果结果不在缓存中，就进行计算。

将计算出来的结果添加到缓存中，以便以后对该操作的调用可以使用。该算法的问题（或者说是潜在的内存泄漏）出在最后一步。如果调用该操作时有相当多的不同输入，就将有相当多的结果存储在缓存中。很明显这不是正确的方法。

为了预防这种具有潜在破坏性的设计，程序必须确保对于缓存所使用的内存容量有一个上限。因此，更好的算法是：

检查结果是否在缓存中，如果在，就返回结果。

如果结果不在缓存中，就进行计算。

如果缓存所占的空间过大，就移除缓存最久的结果。

将计算出来的结果添加到缓存中，以便以后对该操作的调用可以使用。通过始终移除缓存最久的结果，我们实际上进行了这样的假设：在将来，比起缓存最久的数据，最近输入的数据更有可能用到。这通常是一个不错的假设。

新算法将确保缓存的容量处于预定义的内存范围之内。确切的范围可能很难计算，因为缓存中的对象在不断变化，而且它们的引用包罗万象。为缓存设置正确的大小是一项非常复杂的任务，需要将所使用的内存容量与检索数据的速度加以平衡。

解决这个问题的另一种方法是使用java.lang.ref.SoftReference类跟踪缓存中的对象。这种方法保证这些引用能够被移除，如果虚拟机的内存用尽而需要更多堆的话。

ClassLoader

--hyibing  
8. re: FineReport6.5破解  
求一份破解补丁, 感谢 3241364@qq.com  
--hyb  
9. re: FineReport6.5破解  
能给我一份破解补丁吗, 感谢 26234032@qq.com  
--风也  
10. re: FineReport6.5破解[未登录]  
能否发份给我啊? email: 446504678@qq.com  
--追风

阅读排行榜

- 1. JAVA内存泄漏——内存泄漏原因和内存泄漏检测工具(zt)(16373)
- 2. 硬盘分区表知识——详解硬盘MBR(12132)
- 3. 浅谈JAVA程序破解(8769)
- 4. JProfiler 4.2 注册分析(7487)
- 5. 注册机(6114)
- 6. IP地址转换成10进制整数(zt)(5252)
- 7. 推荐一套图形报表工具——ChartDirector(5058)
- 8. 通用防SQL注入函数java版(4251)
- 9. Java基础—关于session的详细解释(zt)(4248)
- 10. FineReport6.5破解(4236)

评论排行榜

- 1. Zelix KlassMaster 破解(226)
- 2. FineReport6.5破解(63)
- 3. 破解 Geneious Pro 3.0.6(45)
- 4. CLC Combined Workbench 3.0.3 破解(32)
- 5. clc 生物工程软件破解(23)
- 6. X报表系统的注册机(23)
- 7. CADI 2.92 电梯调试软件破解(13)
- 8. 通用防SQL注入函数java版(12)
- 9. 我说点山寨机的内幕(zt)(11)
- 10. JProfiler 4.2 注册分析(11)

Java ClassLoader结构的使用为内存泄漏提供了许多可乘之机。正是该结构本身的复杂性使ClassLoader在内存泄漏方面存在如此多的问题。ClassLoader的特别之处在于它不仅涉及“常规”的对象引用, 还涉及元对象引用, 比如: 字段、方法和类。这意味着只要有对字段、方法、类或ClassLoader的对象的引用, ClassLoader就会驻留在JVM中。因为ClassLoader本身可以关联许多类及其静态字段, 所以就有许多内存被泄漏了。

确定泄漏的位置

通常发生内存泄漏的第一个迹象是: 在应用程序中出现了OutOfMemoryError。这通常发生在您最不愿意它发生的生产环境中, 此时几乎不能进行调试。有可能是因为测试环境运行应用程序的方式与生产系统不完全相同, 因而导致泄漏只出现在生产中。在这种情况下, 需要使用一些开销较低的工具来监控和查找内存泄漏。还需要能够无需重启系统或修改代码就可以将这些工具连接到正在运行的系统上。可能最重要的是, 当进行分析时, 需要能够断开工具而保持系统不受干扰。

虽然OutOfMemoryError通常都是内存泄漏的信号, 但是也有可能应用程序确实正在使用这么多的内存; 对于后者, 或者必须增加JVM可用的堆的数量, 或者对应用程序进行某种更改, 使它使用较少的内存。但是, 在许多情况下, OutOfMemoryError都是内存泄漏的信号。一种查明方法是不间断地监控GC的活动, 确定内存使用量是否随着时间增加。如果确实如此, 就可能发生了内存泄漏。

详细输出

有许多监控垃圾收集器活动的方法。而其中使用最广泛的可能是使用-Xverbose:gc选项启动JVM, 并观察输出。

[memory ] 10.109-10.235: GC 65536K->16788K (65536K), 126.000 ms  
箭头后面的值 (本例中是16788K) 是垃圾收集所使用的堆的容量。

控制台

查看连续不断的GC的详细统计信息的输出将是非常乏味的。幸好有这方面的工具。JRockit Management Console可以显示堆使用量的图示。借助于该图, 可以很容易地看出堆使用量是否随时间增加。



Figure 1. The JRockit Management Console

甚至可以配置该管理控制台, 以便如果发生堆使用量过大的情况 (或基于其他的事件), 控制台能够向您发送电子邮件。这明显使内存泄漏的查看变得更容易了。

内存泄漏检测工具

还有其他的专门进行内存泄漏检测的工具。JRockit Memory Leak Detector可以用来查看内存泄漏, 并可以更深入地查出泄漏的根源。这个强大的工具是紧密集成到JRockit JVM中的, 其开销非常

小，对虚拟机的堆的访问也很容易。

### 专业工具的优点

一旦知道确实发生了内存泄漏，就需要更专业的工具来查明为什么会发生泄漏。**JVM**自己是不会告诉您的。这些专业工具从**JVM**获得内存系统信息的方法基本上有两种：**JVMTI**和字节码技术(byte code instrumentation)。**Java**虚拟机工具接口(**Java Virtual Machine Tools Interface, JVMTI**)及其前身**Java**虚拟机监视程序接口(**Java Virtual Machine Profiling Interface, JVMPI**)是外部工具与**JVM**通信并从**JVM**收集信息的标准化接口。字节码技术是指使用探测器处理字节码以获得工具所需的信息的技术。

对于内存泄漏检测来说，这两种技术有两个缺点，这使它们不太适合用于生产环境。首先，它们在内存占用和性能降低方面的开销不可忽略。有关堆使用量的信息必须以某种方式从**JVM**导出，并收集到工具中进行处理。这意味着要为工具分配内存。信息的导出也影响了**JVM**的性能。例如，当收集信息时，垃圾收集器将运行得比较慢。另外一个缺点是需要始终将工具连在**JVM**上。这是不可能的：将工具连在一个已经启动的**JVM**上，进行分析，断开工具，并保持**JVM**运行。

因为**JRockit Memory Leak Detector**是集成到**JVM**中的，就没有这两个缺点了。首先，许多处理和分析工作是在**JVM**内部进行的，所以没有必要转换或重新创建任何数据。处理还可以背负(piggyback)在垃圾收集器本身上而进行，这意味着提高了速度。其次，只要**JVM**是使用-Xmanagement选项（允许通过远程**JMX**接口监控和管理**JVM**）启动的，**Memory Leak Detector**就可以与运行中的**JVM**进行连接或断开。当该工具断开时，没有任何东西遗留在**JVM**中，**JVM**又将以全速运行代码，正如工具连接之前一样。

### 趋势分析

让我们深入地研究一下该工具以及它是如何用来跟踪内存泄漏的。在知道发生内存泄漏之后，第一步是要弄清楚泄漏了什么数据--哪个类的对象引起了泄漏？**JRockit Memory Leak Detector**是通过在每次垃圾收集时计算每个类的现有对象的数量来实现这一步的。如果特定类的对象数目随时间而增长（“增长率”），就可能发生了内存泄漏。



图2. **Memory Leak Detector**的趋势分析视图

因为泄漏可能像细流一样非常小，所以趋势分析必须运行很长一段时间。在短时间内，可能会发生一些类的局部增长，而之后它们又会跌落。但是趋势分析的开销很小（最大开销也不过是在每次垃圾收集时将数据包由**JRockit**发送到**Memory Leak Detector**）。开销不



应该成为任何系统的问题——即使是一个全速运行的生产中的系统。

起初数目会跳跃不停，但是一段时间之后它们就会稳定下来，并显示出哪些类的数目在增长。

找出根本原因

有时候知道是哪些类的对象在泄漏就足以说明问题了。这些类可能只用于代码中的非常有限的部分，对代码进行一次快速检查就可以显示出问题所在。遗憾地是，很有可能只有这类信息还并不够。例如，常见到泄漏出在类`java.lang.String`的对象上，但是因为字符串在整个程序中都使用，所以这并没有多大帮助。

我们想知道的是，另外还有哪些对象与泄漏对象关联？在本例中是**String**。为什么泄漏的对象还存在？哪些对象保留了对这些对象的引用？但是能列出的所有保留对**String**的引用的对象将会非常多，以至于没有什么实际用处。为了限制数据的数量，可以将数据按类分组，以便可以看出其他哪些对象的类与泄漏对象(**String**)关联。例如，**String**在**Hashtable**中是很常见的，因此我们可能会看到与**String**关联的**Hashtable**数据项对象。由**Hashtable**数据项倒推，我们最终可以找到与这些数据项有关的**Hashtable**对象以及**String**（如图3所示）。



图3. 在工具中看到的类型图的示例视图

倒推

因为我们仍然是以类的对象而不是单独的对象来看待对象，所以我们不知道是哪个**Hashtable**在泄漏。如果我们可以弄清楚系统中所有的**Hashtable**都有多大，我们就可以假定最大的**Hashtable**就是正在泄漏的那一个（因为随着时间的流逝它会累积泄漏而增长得相当大）。因此，一份有关所有**Hashtable**对象以及它们引用了多少数据的列表，将会帮助我们指出造成泄漏的确切**Hashtabl**。



图4. 界面：**Hashtable**对象以及它们所引用数据的数量的列表

对对象引用数据数目的计算开销非常大（需要以该对象作为根遍历引用图），如果必须对许多对象都这么做，将会花很多时间。如果了解一点**Hashtable**的内部实现原理就可以找到一条捷径。**Hashtable**的内部有一个**Hashtable**数据项的数组。该数组随着**Hashtable**中对象数目的增长而增长。因此，为找出最大的**Hashtable**，我们只需找出引用**Hashtable**数据项的最大数组。这样要快很多。



图5. 界面：最大的Hashtable数据项数组及其大小的清单

更进一步

当找到发生泄漏的Hashtable实例时，我们可以看到其他哪些实例在引用该Hashtable，并倒推回去看看是哪个Hashtable在泄漏。



图 6. 这就是工具中的实例图

例如，该Hashtable可能是由MyServer类型的对象在名为activeSessions的字段中引用的。这种信息通常就足以查找源代码以定位问题所在了。



图7. 检查对象以及它对其他对象的引用

找出分配位置

当跟踪内存泄漏问题时，查看对象分配到哪里是很有用的。只知道它们如何与其他对象相关联（即哪些对象引用了它们）是不够的，关于它们在何处创建的信息也很有用。当然了，您并不想创建应用程序的辅助构件，以打印每次分配的堆栈跟踪(stack trace)。您也不想仅仅为了跟踪内存泄漏而在运行应用程序时将一个分析程序连接到生产环境中。

借助于JRockit Memory Leak Detector，应用程序中的代码可以在分配时进行动态添加，以创建堆栈跟踪。这些堆栈跟踪可以在工具中进行累积和分析。只要不启用就不会因该功能而产生成本，这意味着随时可以进行分配跟踪。当请求分配跟踪时，JRockit 编译器动态插入代码以监控分配，但是只针对所请求的特定类。更好的是，在进行数据分析时，添加的代码全部被移除，代码中没有留下任何会引起应用程序性能降低的更改。



图8. 示例程序执行期间String的分配的堆栈跟踪

结束语

内存泄漏是难以发现的。本文重点介绍了几种避免内存泄漏的最佳实践，包括要始终记住在数据结构中所放置的内容，以及密切监控内存使用量以发现突然的增长。

我们都已经看到了JRockit Memory Leak Detector是如何用于生产中的系统以跟踪内存泄漏的。该工具使用一种三步式的方法来找出泄漏。首先，进行趋势分析，找出是哪个类的对象在泄漏。接下来，看看有哪些其他的类与泄漏的类的对象相关联。最后，进一步

研究单个对象，看看它们是如何互相关联的。也有可能对系统中所有对象分配进行动态的堆栈跟踪。这些功能以及该工具紧密集成到JVM中的特性使您可以以一种安全而强大的方式跟踪内存泄漏并进行修复。

原文出  
处：[http://dev2dev.bea.com/pub/a/2005/06/memory\\_leaks.html](http://dev2dev.bea.com/pub/a/2005/06/memory_leaks.html)

posted on 2006-04-28 10:20 舵手 阅读(16373) 评论(2) 编辑 收藏

评论

# re: JAVA内存泄漏——内存泄漏原因和内存泄漏检测工具(zt)[未登录] 回复 更多评论

顶~收藏!

2008-05-28 17:30 | peter

# re: JAVA内存泄漏——内存泄漏原因和内存泄漏检测工具(zt) 回复 更多评论

qweqwe

2008-10-21 10:46 | 213

[新用户注册](#) [刷新评论列表](#)

找优秀程序员，就在博客园

IT新闻：

- 传雅虎考虑避税剥离阿里巴巴股权：估价140亿
- Office 2003走过十年 比尔·盖茨生日快乐
- SMK开发出可戴着手套输入信息的触控面板
- 摩托罗拉第三季度仅售出10万台Xoom平板电脑
- RIM证实软件高级副总裁托宾数月前离职

**买正版控件，到慧都控件网**  
网址：<http://www.evget.com> 免费电话：400-700-1020

**正在促销**

[www.evget.com](http://www.evget.com) Google 提供的广告

[博客园](#) [博问](#) [IT新闻](#) [Java程序员招聘](#)

标题

姓名

主页

验证码

\* 8044

内容(请不要发表任何与政治相关的内容)

Remember Me?

登录

[使用Ctrl+Enter键可以直接提交]



推荐职位:

- 杭州高级C++软件工程师(新迪数字工程系统)
- 北京.NET软件开发工程师(北京国双科技)
- 北京.NET 研发工程师 (北京捷报数据)
- CRM研发工程师(年薪15W以上) (网易有道)
- 北京C#开发工程师 B/S方向(圣特尔科技)
- 北京高级.NET工程师 (月薪15k) (盛安德科技)

博客园首页随笔:

- 21个设计Web应用程序的最佳 图标集
- LINQ之路 5: LINQ查询表达式
- DriveInfo 类 提供对有关驱动器的信息的访问
- 程序员应知——关注细节
- java多线程 sleep()和wait()的区别

知识库:

- C++ 程序员的 C# 转型手册
- 书上没写的领导守则
- 大型网站后台架构的Web Server与缓存
- 优秀程序设计的18大原则



· 程序员与非程序员的思维差异

网站导航:

[博客园](#) [IT新闻](#) [知识库](#) [C++博客](#) [程序员招聘](#) [管理](#)

Powered by:  
[BlogJava](#)  
Copyright © 舵手