# 一份关于jvm内存调优及原理的学习笔记

JVM

一.虚拟机的基本结构

1.jvm整体架构

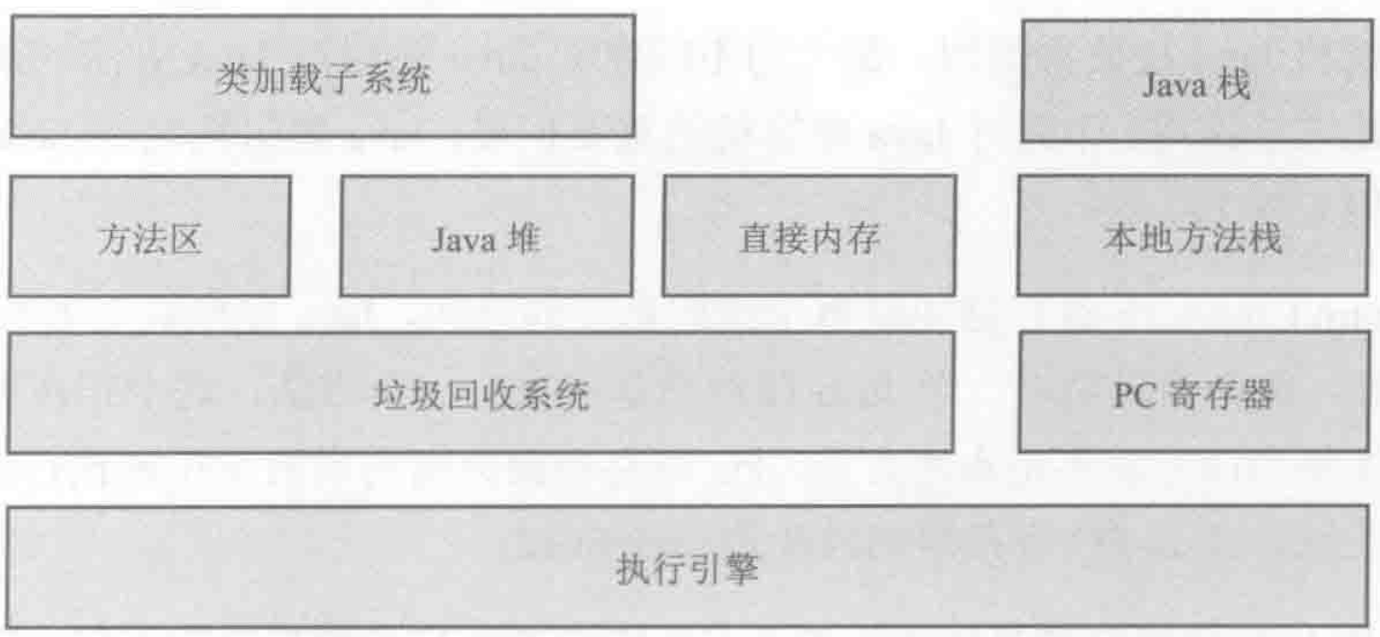


图 2.1 Java 虚拟机的基本结构

类加载子系统：负责从文件系统或者网络中加载class信息，存入方法区中。

方法区（Perm）：存放加载后的class信息，包括静态方法，jdk1.6以前包含了常量池。

参数：-XX:PermSize初始值 -XX:MaxPermSize最大值

Java堆（Heap）：java工程的主要内存工作区域，所有线程共享，jdk1.7以后包含了常量池。参数： -Xms初始值 -Xmx最大值

直接内存：java堆外，直接向系统申请的内存区间，允许NIO库使用。申请空间慢，读写快。默认下最大可用空间等于堆的最大可用空间。在server模式下，读写速度是堆的10倍。

参数：-XX:MaxDirectMemorySize 最大值

垃圾回收器：

Java栈：线程私有，用于存放局部变量，方法参数，同时和java方法的调用返回密切相关。

参数：-Xss最大值

昵称：RUN_TIME
园龄：2年6个月
粉丝：14
关注：2
+加关注

< 2017年10月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 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 | 31 | 1 | 2 | 3 | 4 |

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

Could Not Connect(1)
get(1)
http(1)
httpclient(1)
java(1)
JavaScript(1)
post(1)
tomcat负载均衡(1)
weblogic(1)
webservice(1)
更多

随笔分类

工作经历(5)
学习笔记(2)
有点趣儿

随笔档案

本地方法栈：和java栈类似，主要用于本地方法调用。

PC寄存器：线程私有

执行引擎：

Java [-options] class [args...]

其中-options是java虚拟机的启动参数，args是传递给main方法的参数、

2.java堆

根据垃圾回收机制的不同，java堆有可能拥有不同的结构，常见的java堆分为新生代和老年代。其中新生代存放刚创建的对象及年龄不大的对象，老年带存放着在新生代中经历过多次回收后还存在的对象。

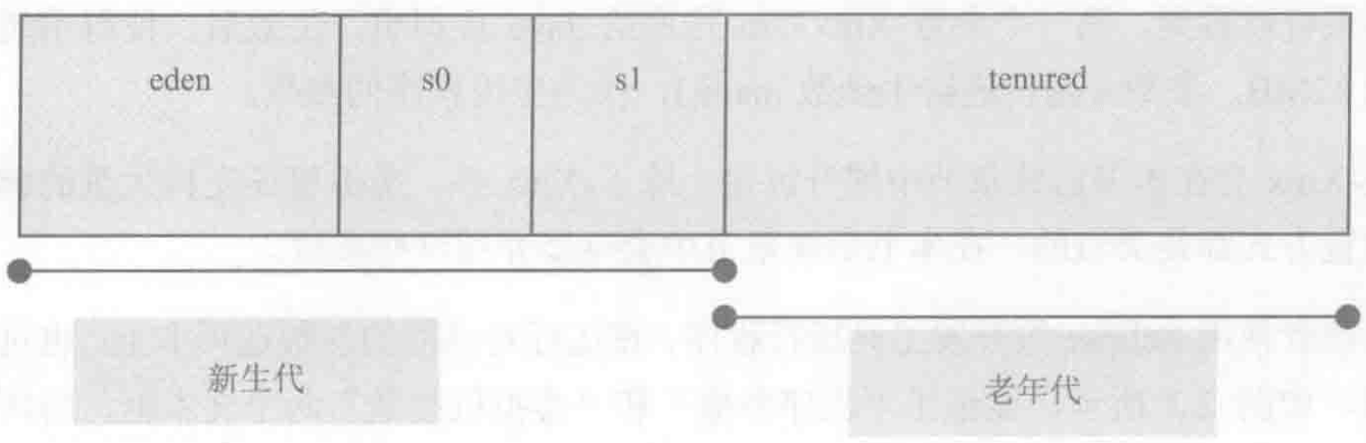


图 2.3　堆空间的一般结构

对象晋升过程：

新生代分为eden区s0,s1区（from，to）。多数情况下对象首先分配在eden区，在一次新生代回收后，存活下来的对象存入s0或s1区。每经过一次新生代的回收，对象的年龄加1。默认情况下年龄达到15的对象将晋升至老年代。如果在第一次回收的时候，存活的对象大于s0（s1）空间，将直接晋升至老年代，如果在为对象第一次分配空间时，对象空间大于eden空间的话，对象也直接分配到老年代。

3.java栈

Java栈和数据结构中的栈有着类似的含义，先进后出，只支持入栈和出栈操作。Java栈中保存的只要内容是栈帧，每一次进行函数调用，都会有一个对应的栈帧被压入栈中，函数调用结束，都会有一个栈帧被弹出栈。

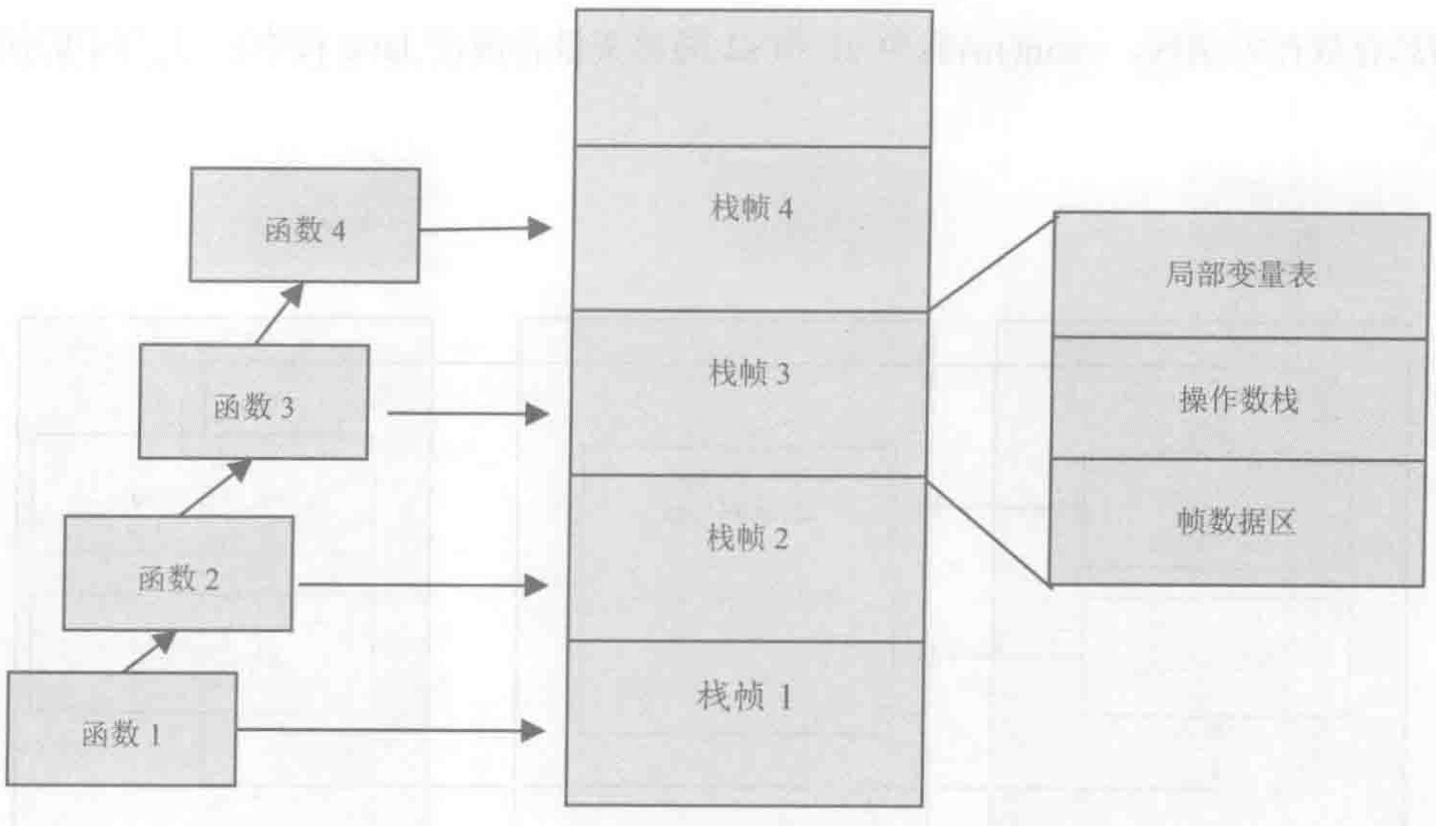


图 2.5　帧栈和函数调用

2017年5月 (1)
2016年8月 (1)
2016年4月 (1)
2015年7月 (1)
2015年6月 (2)
2015年4月 (3)

## 最新评论

1. Re:设计模式六大原则

面向对象六大原则

--清香白莲素还真

2. Re:java反射机制性能优化

@GoQC这个的话最好是人为规避这个问题，如果非要代码解决的话，可以再嵌套个map，拿参数的类型数组（Class[] cs）做键 。取值的时候通过对比cs的内容来判断。不知道你有没有什么比较好的解决.....

--RUN_TIME

3. Re:java反射机制性能优化

mmap.put(m.getName(), m.getGenericReturnType().toString());//遍历出所有的方法，将方法名和返回类型存在静态的map中（缓存）重载的方法怎么办.....

--GoQC

4. Re:java反射机制性能优化

Method m = metMap.get(clazz+"_"+method+"_"+size);//用于区分重载的方法如果重载的方法参数个数一样只是类型不同呢。。。...

--GoQC

5. Re:java反射机制性能优化

依赖注入 啊

--查克拉的觉醒

## 阅读排行榜

1. 一份关于jvm内存调优及原理的学习笔记(11093)
2. js调用百度地图接口(9717)
3. java反射机制性能优化(5070)
4. WebLogic部署集群和代理服务器(4361)
5. Apache+Tomcat部署负载均衡（或集群）(3952)

## 评论排行榜

1. js调用百度地图接口(6)
2. java反射机制性能优化(4)
3. 浅谈http请求数据分析(4)
4. WebLogic部署集群和代理服务器(3)
5. Apache+Tomcat部署负载均衡（或集群）(2)

栈帧

每一个栈帧中包含局部变量表，操作数栈和帧数据区。

栈上分配

栈上分配的基本思想，是将线程私有的对象，打散分配到栈上，分配在栈上的函数调用结束后对象会自行销毁，不需要垃圾回收接入，从而提升性能。对于大量的零散小对象，栈上分配提供了一种很好的对象分配优化策略，但由于和堆空间相比，栈空间较小，因此大对象无法也不适合在栈上分配

栈上分配依赖逃逸分析和标量替换的实现，同时必须在server模式下才能启用。参数-XX:+DoEscapeAnalysis启用逃逸分析　-XX:+EliminateAllocations开启标量替换(默认打开).

例：-server -Xms 100m -Xmx 100m -XX:+DoEscapeAnalysis -XX:+EliminateAllocations



二．Jvm常用参数

1.GC参数

-XX:+PrintGC　每次触发GC的时候打印相关日志

-XX:+PrintGCDetails　更详细的GC日志

-XX:+PrintHeapAtGC　每次GC时打印堆的详细详细信息

-XX:+PrintGCApplicationConcurrentTime　打印应用程序执行时间

-XX:+PrintGCApplicationAtoppedTime　打印应用程序由GC引起的停顿时间

-XX:+PrintReferenceGC　跟踪系统内的软引用，弱引用，虚引用和finallize队列。

1.类跟踪

-verbose:class　跟踪类的加载和卸载

-XX:+TraceClassLoading　单独跟踪类加载

-XX:+TraceClassUnloading　单独跟踪类卸载

-XX:+PrintClassHistogram　查看运行时类的分布情况，使用时在控制台按ctrl+break

2.系统参数查看

-XX:+PrintVMOptions　　运行时，打印jvm接受的命令行显式参数

-XX:+PrintCommandLineFlags　打印传递jvm的显式和隐式参数

-XX:+PrintFlagsFinal　打印所有系统参数值

3.堆

-Xms　堆初始值

-Xmx　堆最大可用值

-Xmn　新生代大小，一般设为整个堆的1/3到1/4左右

-XX:SurvivorRatio　设置新生代中eden区和from/to空间的比例关系n/1

-XX:NewRatio　设置老年代与新生代的比

想要合理的分配堆内存，需要了解对象的晋升过程，可以参照上面介绍堆空间架构时，对对象晋升过程的描述。

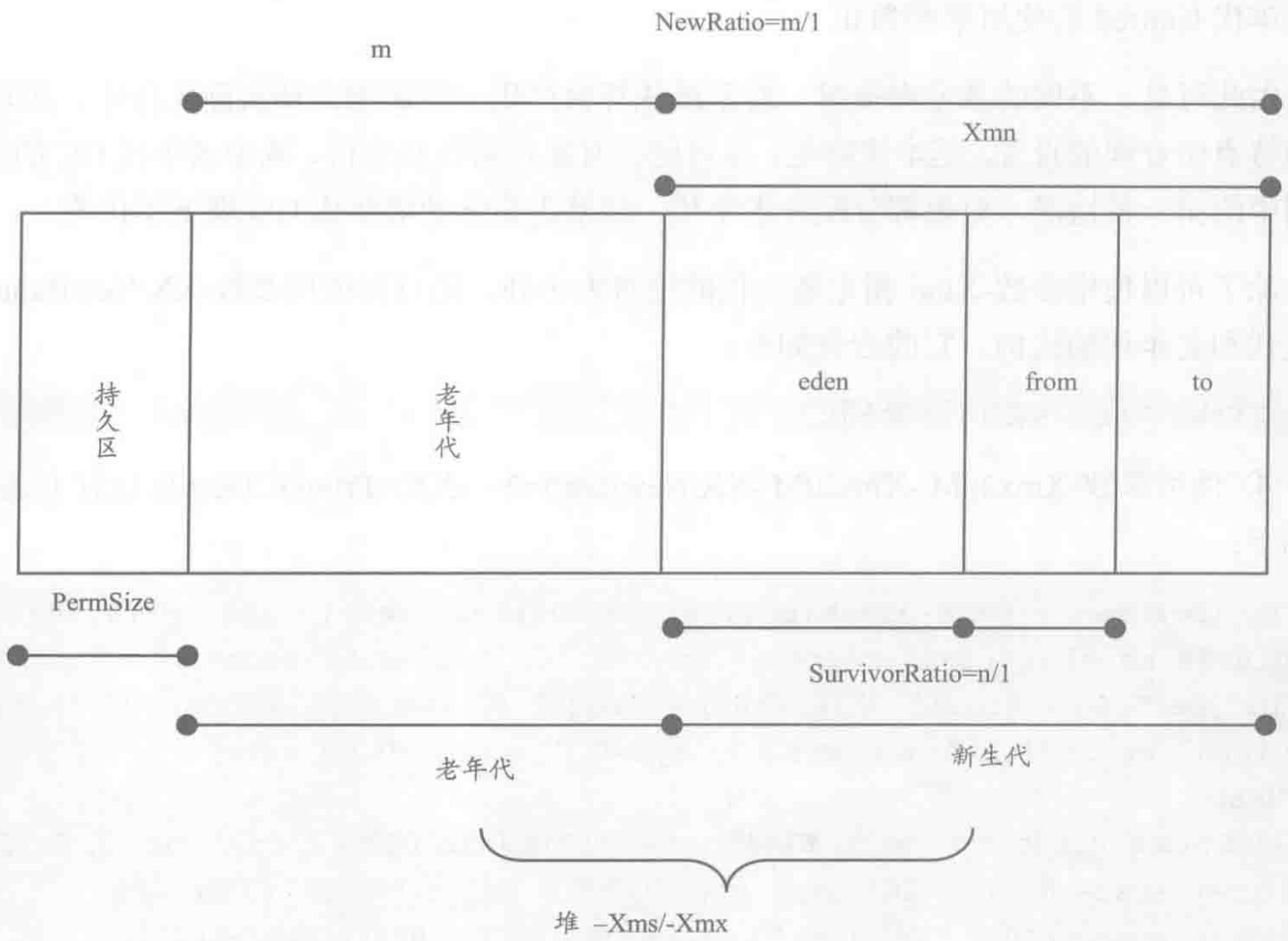


图 3.2　堆的分配参数示意图

基本策略：堆的不同分布情况，对系统会产生一定的影响。尽可能将对象预留在新生代，减少老年代GC的次数（通常老年回收起来比较慢）。实际工作中，通常将堆的初始值和最大值设置相等，这样可以减少程序运行时进行的垃圾回收次数和空间扩展，从而提高程序性能。

4.非堆

-XX:PermSize　方法区（永久区）初始值

-XX:MaxPermSize　方法区（永久区）最大值

-Xss　设置栈空间大小

-XX:MaxDirectMemorySize　直接内存最大可用空间，设置不当可能导致系统OOM

5.虚拟机工作模式

-client　默认工作模式

-server　server工作模式，启动虚拟机时需要显式指定

与client模式相比，server模式启动较慢，会尝试搜集更多的系统性能信息，使用更复杂的优化算法对程序进行优化，server模式下系统完全启动并进入稳定期后，执行速度远远快于client模式，适合长期后台运行的系统。Client模式更适合运行时间不长，又追求启动速度的客户端程序。

三．Jvm性能监控工具

1.JConsole

内存监控，线程监控，类加载情况，虚拟机信息

2.Visual VM

线程dump和分析，性能分析，内存快照分析，BTrace

3.Mission Control

MBean服务器，飞行记录器

四．分析java堆

1.常见的内存溢出原因及解决思路

(1)堆溢出：设置-Xmx调整最大可用堆空间

(2)直接内存溢出：可能是系统内存空间不足，同时没达到参数默认的上限，没有触发GC导致OOM，解决方法是通过-XX:MaxDirectMemorySize 来限制最大内存。

(3)过多线程导致OOM：由于每开启一个线程都会给这个线程分配一个栈，因此当线程数达到一定程度，系统空间不足的时候就会内存溢出，可以尝试减少堆空间，或者可以通过设置参数-Xss限制每个栈的大小。

(4)永久区溢出：系统加载的类过多，导致永久区溢出，通过-XX:MaxPermSize来设置永久区最大可用空间。

(5)GC效率低下引起的OOM：GC是内存回收的关键，回收效率低很有可能引起内存溢出，可以通过合理的分配堆（包括新生代和老年代）空间去解决。

2.String造成的内存泄漏

内存泄漏是指，不再使用的对象占据内存不释放，导致可用内存不断减小，最终引起内存泄漏。在Java1.6中String.subString()方法就存在这样的问题。

SubString中新生成的对象并没有从value中获取自己需要的那部分，而是直接简单的使用了相同的引用，只是修改了offset和count，以此来确定新的String对象的值。当原始字符串还在用的时候这种情况是没有问题的，并且共用value还节省了部分的空间，但是一旦原始字符串被回收，value中多余的部分就造成了空间浪费。

3.浅堆和深堆

浅堆：是指一个对象本身所消耗的内存，不包括其内部引用的对象的大小。

深堆：是指对象的保留集中所有对象浅堆的大小之和。

保留集：是指当对象A被垃圾回收后，可以释放的所有对象的集合（包括A本身），通俗的讲就是，仅被对象A所持有的对象的集合。

4.OQL查询语句

类似于sql语法的查询语句，可以在堆中进行对象的查找和筛选。

.....

分类: 学习笔记



RUN_TIME
关注 - 2
粉丝 - 14
+加关注



« 上一篇：浅谈http请求数据分析
» 下一篇：java反射机制性能优化

posted @ 2016-04-29 09:19 RUN_TIME 阅读(11094) 评论(0) 编辑 收藏

**最新IT新闻**:

· 阿里云双11活动开始预热 云服务器限时2折起
· 传百度正在缩小埃及分支机构规模：已裁员超30人
· 海洋温度模型的缺陷表明，气候变化可能比预期更糟
· Google宣布为Pixel 2/Pixel 2 XL提供两年保修服务
· 滴滴更换出租车派车模式后：乘客打不到车，司机接不到单

» 更多新闻...



**最新知识库文章**:

· 实用VPC虚拟私有云设计原则
· 如何阅读计算机科学类的书
· Google 及其云智慧
· 做到这一点，你也可以成为优秀的程序员
· 写给立志做码农的大学生

» 更多知识库文章...