

目录 /CONTENTS

01

JIT优化

02

内存优化

03

监控及工具

04

经验值

GCRoots对象

所有正在运行的线程的栈上的引用变量。所有的全局变量。所有ClassLoader。。。

- 1.System Class
- 2.JNI Local
- 3.JNI Global
- 4.Thread Block
- 5.Busy Monitor
- 6.Java Local
- 7.Native Stack
- 8.Unfinalized
- 9.Unreachable
- 10.Java Stack Frame
- 11.Unknown

栈帧的解释

Java虚拟机栈（Java Virtual Machine Stacks）是线程私有的，它的生命周期与线程相同。虚拟机栈描述的是Java方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

方法区

与Java堆一样，是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

```
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.log4j.Logger;

public class HelloWorld {
    private static Logger LOGGER = Logger.getLogger(HelloWorld.class.getName());
    public void sayHello(String message) {
        SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.YYYY");
        String today = formatter.format(new Date());
        LOGGER.info(today + ": " + message);
    }
}
```

堆区

- Object:Hello World
- Object:SimpleDateFormat
- Object:String
- Object:Logger

方法区

- Class:SimpleDateFormat
- ...
- Class: Logger
- ...
- Class:Hello World
- Method: sayHello()
- ...

栈区，线程1-main

- 参数
- 变量
- 原始类型：行号

01

JIT优化

JUST IN TIME OPTIMIZATION



常规优化

- 1、禁用System.gc
- 2、逃逸分析与标题替换
- 3、关闭偏向锁优化
- 4、指针压缩
- 5、getter方法优化

JIT优化

- 1、开启服务端模式
- 2、增加内联函数的可能性
- 3、提高编译的可能性
- 4、降低线程优先级
- 5、热度衰减与半衰周期

常规优化

- 1、-XX:-DisableExplicitGC，禁用了System.gc()的显示调用
- 2、逃逸分析默认是启用的，-XX:+DoEscapeAnalysis。后续有三种优化会进行：栈内分配，同步消除，标量替换
- 4、偏向锁，关闭：-XX:-UseBiasedLocking
-XX:+UseBiasedLocking -XX:BiasedLockingStartupDelay=0
- 5、指针压缩，-XX:+UseCompressedOops
- 6、getter方法优化，-XX:UseFastAccessorMethods

常规优化

分析对象动态作用域：当一个对象在方法中被定义后，它可能被外部方法所引用，例如作为调用参数传递到其他方法中，称为方法逃逸。甚至还有可能被外部线程访问到，譬如赋值给类变量或可以在其他线程中访问的实例变量，称为线程逃逸。

栈上分配（**Stack Allocation**）：如果确定一个对象不会逃逸出方法之外，那让这个对象在栈上分配内存将会是一个很不错的主意。由于**HotSpot**虚拟机目前的实现方式导致栈上分配实现起来比较复杂，因此在**HotSpot**中暂时还没有做这项优化。

同步消除（**Synchronization Elimination**）：线程同步本身是一个相对耗时的过程，如果逃逸分析能够确定一个变量不会逃逸出线程，无法被其他线程访问，那这个变量的读写肯定就不会有竞争，对这个变量实施的同步措施也就可以消除掉。

标量替换（**Scalar Replacement**）：标量（**Scalar**）是指一个数据已经无法再分解成更小的数据来表示了，**Java**虚拟机中的原始数据类型（**int**、**long**等数值类型以及**reference**类型等）都不能再进一步分解，它们就可以称为标量。相对的，如果一个数据可以继续分解，那它就称作聚合量

（**Aggregate**），**Java**中的对象就是最典型的聚合量。如果把一个**Java**对象拆散，根据程序访问的情况，将其使用到的成员变量恢复原始类型来访问就叫做标量替换。如果逃逸分析证明一个对象不会被外部访问，并且这个对象可以被拆散的话，那程序真正执行的时候将可能不创建这个对象，而改为直接创建它的若干个被这个方法使用到的成员变量来代替。

JIT优化

- 1、服务端模式，-server
- 2、final的函数是向编译器建议可以内联，启动参数不宜设置
- 3、提高编译的可能性，小方法，-XX:CompileThreshold=10000
- 4、线程优先级，Linux不能设置，需要root权限
- 5、热度衰减与半衰周期

OSR编译阈值

A、调用计数器，即方法被调用的次数，CompileThreshold，该值是指当方法被调用多少次后，就编译为机器码，client模式默认为1500次，server模式默认为1万次，可以在启动时添加-XX:CompileThreshold=10000来设置该值。

B、回边计数器，即方法中循环执行部分代码的执行次数，OnStackReplacePercentage，该值用于/参与计算是否触发OSR编译的阈值，client默认为933，server默认为140，可以通过-XX:OnStackReplacePercentage=140来设置。

client模式下的计算规则为

$\text{CompileThreshold} * \text{OnStackReplacePercentage} / 100$ ，

server模式下计算规则为

$\text{CompileThreshold} * (\text{OnStackReplacePercentage} - \text{InterpreterProfilePercentage}) / 100$ 。
InterpreterProfilePercentage，默认为33。

A photograph of a person's hands typing on a silver laptop keyboard. The laptop is open, and the screen displays a blurred image. To the left of the laptop, there is a green semi-circular overlay containing text. In the background, a glass pen holder with several pens is visible on a desk.

02

内存优化

MEMORY OPTIMIZATION

内存优化

- 1、将新对象预留在年轻代
- 2、让大对象进入年老代
- 3、设置对象进入年老代的年龄
- 4、稳定的 Java 堆
- 5、增大吞吐量提升系统性能
- 6、使用非占有的垃圾回收器

内存优化

- 1、将新对象预留在年轻代，`-XX:TargetSurvivorRatio=90`
- 2、让大对象进入年老代，`-XX:PetenureSizeThreshold=1000000`，1M
- 3、设置对象进入年老代的年龄，`-XX:MaxTenuringThreshold=31`
- 4、稳定的 Java 堆，Xmx与Xms相同
- 5、增大吞吐量提升系统性能，`-XX:+UseParallelGC`，`-XX:+UseParallelOldGC`，`-XX:ParallelGC-Threads`（CPU核心数相等）
- 6、使用非占有的垃圾回收器，`-XX:+UseConcMarkSweepGC`

03

监控及工具

MONITORING AND TOOLS



jps：虚拟机进程状况工具

它的功能也和ps命令类似：可以列出正在运行的虚拟机进程，并显示虚拟机执行主类（Main Class,main（）函数所在的类）名称以及这些进程的本地虚拟机唯一ID（Local Virtual Machine Identifier,LVMID）。jps可以通过RMI协议查询开启了RMI服务的远程虚拟机进程状态，hostid为RMI注册表中注册的主机名。

选 项	作 用
-q	只输出 LVMID，省略主类的名称
-m	输出虚拟机进程启动时传递给主类 main() 函数的参数
-l	输出主类的全名，如果进程执行的是 Jar 包，输出 Jar 路径
-v	输出虚拟机进程启动时 JVM 参数

jstat：虚拟机统计信息监视工具

用于监视虚拟机各种运行状态信息的命令行工具。它可以显示本地或者远程虚拟机进程中的类装载、内存、垃圾收集、JIT编译等运行数据，在没有GUI图形界面，只提供了纯文本控制台环境的服务器上，它将是运行期定位虚拟机性能问题的首选工具。

选 项	作 用
-class	监视类装载、卸载数量、总空间以及类装载所耗费的时间
-gc	监视 Java 堆状况，包括 Eden 区、两个 survivor 区、老年代、永久代等的容量、已用空间、GC 时间合计等信息
-gccapacity	监视内容与 -gc 基本相同，但输出主要关注 Java 堆各个区域使用到的最大、最小空间
-gcutil	监视内容与 -gc 基本相同，但输出主要关注已使用空间占总空间的百分比
-gccause	与 -gcutil 功能一样，但是会额外输出导致上一次 GC 产生的原因
-gcnw	监视新生代 GC 状况
-gcnewcapacity	监视内容与 -gcnew 基本相同，输出主要关注使用到的最大、最小空间
-gcold	监视老年代 GC 状况
-gcoldcapacity	监视内容与 -gcold 基本相同，输出主要关注使用到的最大、最小空间
-gcpermcapacity	输出永久代使用到的最大、最小空间
-compiler	输出 JIT 编译器编译过的方法、耗时等信息
-printcompilation	输出已经被 JIT 编译的方法

jmap : Java内存映像工具

map的作用并不仅仅是为了获取dump文件，它还可以查询finalize执行队列、Java堆和永久代的详细信息，如空间使用率、当前用的是哪种收集器等。

选 项	作 用
-dump	生成 Java 堆转储快照。格式为：-dump:[live,]format=b, file=<filename>，其中 live 子参数说明是否只 dump 出存活的对象
-finalizerinfo	显示在 F-Queue 中等待 Finalizer 线程执行 finalize 方法的对象。只在 Linux / Solaris 平台下有效
-heap	显示 Java 堆详细信息，如使用哪种回收器、参数配置、分代状况等。只在 Linux / Solaris 平台下有效
-histo	显示堆中对象统计信息，包括类、实例数量、合计容量
-permstat	以 ClassLoader 为统计口径显示永久代内存状态。只在 Linux / Solaris 平台下有效
-F	当虚拟机进程对 -dump 选项没有响应时，可使用这个选项强制生成 dump 快照。只在 Linux / Solaris 平台下有效

其他工具

jinfo : Java配置信息工具

作用是实时地查看和调整虚拟机各项参数。使用-sysprops选项把虚拟机进程System.getProperties () 的内容打印出来。

jhat : 虚拟机堆转储快照分析工具

jhat内置了一个微型的HTTP/HTML服务器，生成dump文件的分析结果后，可以在浏览器中查看。

HSDIS : JIT生成代码反汇编

HSDIS是一个HotSpot虚拟机JIT编译代码的反汇编插件，它包含在HotSpot虚拟机的源码之中，但没有提供编译后的程序。

可视化工具

JConsole : Java监视与管理控制台

JConsole (Java Monitoring and Management Console) 是一种基于JMX的可视化监视、管理工具。它管理部分的功能是针对JMX MBean进行管理，由于MBean可以使用代码、中间件服务器的管理控制台或者所有符合JMX规范的软件进行访问。

VisualVM : 多合一故障处理工具

VisualVM (All-in-One Java Troubleshooting Tool) 是到目前为止随JDK发布的功能最强大的运行监视和故障处理程序。VisualVM的还有一个很大的优点：不需要被监视的程序基于特殊Agent运行，因此它对应用程序的实际性能的影响很小，使得它可以直接应用在生产环境

JMC , Oracle Java Mission Control 是一个用于对 Java 应用程序进行管理、监视、概要分析和故障排除的工具套件。首次安装时，Java Mission Control 包括 JMX 控制台和 Java 飞行记录器。从 Mission Control 中可以轻松安装更多插件

JITWatch

安装：

```
git clone git@github.com:AdoptOpenJDK/jitwatch.git  
cd jitwatch  
mvn clean install -DskipTests=true
```

运行：launchUI.bat

使用：XX:+UnlockDiagnosticVMOptions -XX:+TraceClassLoading -XX:+LogCompilation -XX:+PrintAssembly

查看结果。

A high-angle photograph of two people working at a desk. One person, wearing a blue striped shirt, is pointing at a laptop screen. The other person, also in a blue striped shirt and wearing glasses, is writing in a notebook. The desk is white and has a laptop, a notebook, and a glass of water. A green semi-transparent overlay covers the left side of the image, containing the text '04 经验值 EXPERIENCE VALUE'.

04

经验值

EXPERIENCE VALUE

服务器：8 cup, 8G mem

e.g.

```
java -Xmx3550m -Xms3550m -Xss128k -XX:NewRatio=4 -XX:SurvivorRatio=4 -XX:MaxPermSize=16m -XX:MaxTenuringThreshold=0
```

调优方案：

-Xmx5g：设置JVM最大可用内存为5G。

-Xms5g：设置JVM初始内存为5G。此值可以设置与-Xmx相同，以避免每次垃圾回收完成后JVM重新分配内存。

-Xmn2g：设置年轻代大小为2G。整个堆内存大小 = 年轻代大小 + 年老代大小 + 持久代大小。持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。

-XX:+UseParNewGC：设置年轻代为并行收集。可与CMS收集同时使用。JDK5.0以上，JVM会根据系统配置自行设置，所以无需再设置此值。

-XX:ParallelGCThreads=8：配置并行收集器的线程数，即：同时多少个线程一起进行垃圾回收。此值最好配置与处理器数目相等。

-XX:SurvivorRatio=6：设置年轻代中Eden区与Survivor区的大小比值。根据经验设置为6，则两个Survivor区与一个Eden区的比值为2:6，一个Survivor区占整个年轻代的1/8。

-XX:MaxTenuringThreshold=30：设置垃圾最大年龄（次数）。如果设置为0的话，则年轻代对象不经过Survivor区直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值 设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概率。设置为30表示 一个对象如果在Survivor空间移动30次还没有被回收就放入年老代。

-XX:+UseConcMarkSweepGC：设置年老代为并发收集。测试配置这个参数以后，参数-XX:NewRatio=4就失效了，所以，此时年轻代大小最好用-Xmn设置，因此这个参数不建议使用。