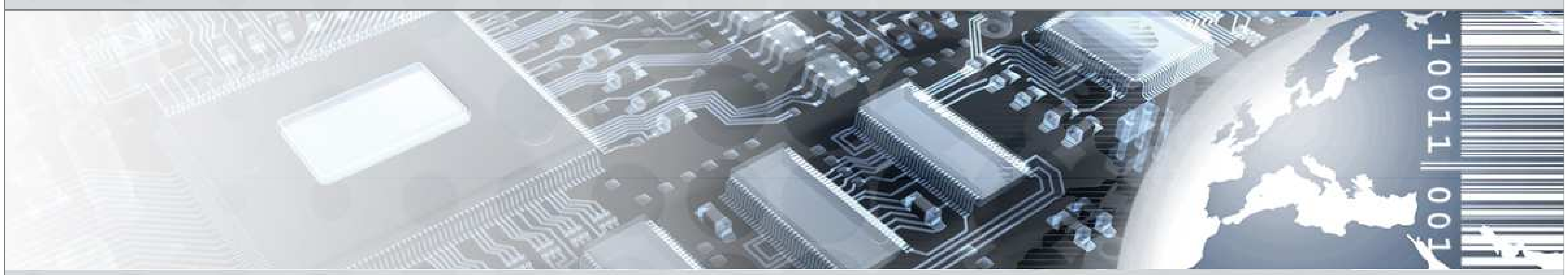


Monitoring und Profiling von Java-Anwendungen

von Rolf Kulemann

Inhalt

1. **Wieso Profiling bzw. Monitoring?**
    1. Monitoring
    2. Profiling
    3. Abgrenzung Monitoring zu Profiling
2. **(Basis)Werkzeuge des JDK**
    1. jps
    2. jstat
    3. jstatd
3. **Exkurs: Java Management Extensions (JMX)**
4. **Exkurs: Java Heaps**
5. **Exkurs: Garbage Collection**
6. **Monitoring mit jconsole**
7. **Profiling mit jvisualvm**
    1. Übersicht Beispiele
    2. Profiling CPU
    3. Profiling Memory

1. Wieso Profiling und Monitoring? Monitoring

- **Def.: Monitoring ist ein Überbegriff für alle Arten der unmittelbaren systematischen Erfassung (*Protokollierung*), *Beobachtung oder Überwachung* eines Vorgangs oder Prozesses mittels technischer Hilfsmittel oder anderer Beobachtungssysteme. Dabei ist die wiederholende Durchführung ein zentrales Element der jeweiligen Untersuchungsprogramme, um anhand von Ergebnisvergleichen Schlussfolgerungen ziehen zu können.**
- ***Die Funktion des Monitorings* besteht darin, bei einem beobachteten Ablauf bzw. Prozess *steuernd einzugreifen*, sofern dieser nicht den gewünschten Verlauf nimmt bzw. bestimmte Schwellwerte unter- bzw. überschritten sind. Monitoring ist deshalb ein Sondertyp des Protokollierens.**

1. Wieso Profiling und Monitoring?
Profiling

- **Def.: Als Profiler werden *Programmierwerkzeuge* bezeichnet, die das Laufzeitverhalten von Software analysieren. Es gibt unterschiedliche Problembereiche in der Softwareentwicklung, die durch *ineffiziente* Programmierung ausgelöst werden.**
- **Ein Profiler hilft dem Entwickler durch Analyse und Vergleich von laufenden Programmen die Problembereiche aufzudecken. Daraus kann man Maßnahmen zur strukturellen und algorithmischen Verbesserung des Quellcodes ableiten.**
- **Profiler können bei einer API Analyse zwecks re-engineering sehr behilflich sein, wenn kein Quellcode vorliegt**
- **Laufzeit oder Speicherprobleme ohne Profiler zu analysieren ist sehr zeitaufwendig und mühsam. I.d.R. wird umständlicher problemspezifisches Logging in die Software eingehführt…**
- **Java Profiler arbeiten i.d.R. mit reversibler Code Instrumentierung zur Laufzeit**

1. Wieso Profiling und Monitoring?
Abgrenzung Monitoring zu Profiling

- **Monitoring wird zur Überwachung und Steuerung eingesetzt**
- **Mit Monitoring ist es möglich grundsätzlich mögliche Probleme im Laufzeitverhalten festzustellen -> Anwendung benötigt viel Speicher oder ist langsam**
- **Mit Profiling analysiert man durch Benutzer oder Monitoring festgestellte Probleme im Detail -> Speichernutzung und Performance und leitet Maßnahmen zur Behebung ab**
- **D.h. Monitoring trägt der Erkenntnis bei und Profiling der Problembehebung**

2. (Basis)Werkzeuge des JDK jps

- **Siehe http://java.sun.com/performance/jvmstat/**
- **Auszug: JVM Process Status Tool - Lists instrumented HotSpot Java virtual machines on a target system. *(formerly jvmps)***

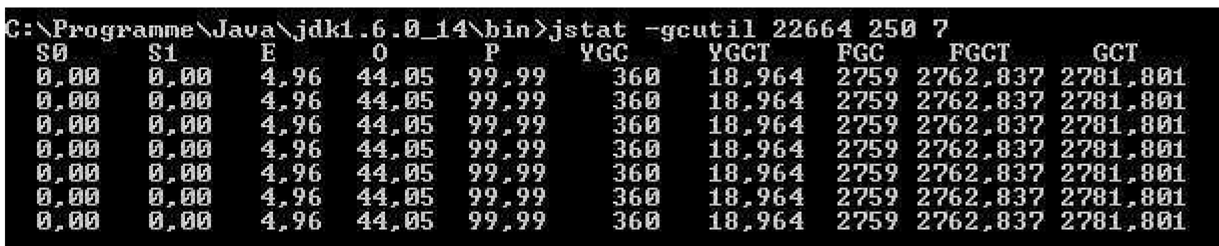
```
C:\Programme\Java\jdk1.6.0_14\bin>jps
24612 Main
22664 org.eclipse.equinox.launcher_1.1.0.v20100507.jar
23268 Jps
10156 JConsole
9172

C:\Programme\Java\jdk1.6.0_14\bin>_
```

- **Wird auch von jconsole und jvisualvm genutzt**
- **Kurz: Listet Java VM Prozesse**
- **Nützlich: Bedingt**

(Basis)Werkzeuge des JDK
jstat

- **Siehe http://java.sun.com/performance/jvmstat/**
- **Auszug: JVM Statistics Monitoring Tool - Attaches to an instrumented HotSpot Java virtual machine and collects and logs performance statistics as specified by the command line options. *(formerly jvmstat)***



C:\Programme\Java\jdk1.6.0_14\bin>jstat -gcutil 22664 250 7

S0	S1	E	O	P	YGC	YGCT	FGC	FGCT	GCT
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801
0,00	0,00	4,96	44,05	99,99	360	18,964	2759	2762,837	2781,801

- ***Kurz: Kommandozeilen Monitoring***
- ***Nützlich: Bedingt, da schwierig zu bedienen. Sehr viele Aufrufvariationen. Ist sicher ein Expertentool und nicht für den täglichen Gebrauch geeignet***
- ***Besser man nutz jconsole/jvsualvm als Frontend***

(Basis)Werkzeuge des JDK
jstatd

- **Siehe http://java.sun.com/performance/jvmstat/**
- **Auszug: JVM jstat Daemon - Launches an RMI server application that monitors for the creation and termination of instrumented HotSpot Java virtual machines and provides a interface to allow remote monitoring tools to attach to Java virtual machines running on the local system. *(formerly perfagent)***
- ***Kurz: Muss auf Zielrechner gestartet sein, wenn man jstat remote auf eine VM anwenden möchte***

3. Exkurs: Java Management Extensions (JMX)

NEO BUSINESS PARTNERS

- **Erwähnenswert, da Monitoring auch Steuerung betrifft**
- **Java Management Extensions (JMX) ist eine vom Java Community Process (JSR-3) entwickelte Spezifikation zur Verwaltung und Überwachung von Java-Anwendungen**
- **Vergleiche SNMP**
- **D.h. es können über die JMX API Management Extensions (MBeans) entwickelt und in einer VM (MBeanServer) bereitgestellt werden**
- **Entsprechende Tools können die Management Extensions, welche von einer VM exponiert werden, listen und aufrufen**
- **Standard VM/JDK stellt schon viele Extensions bereit, welche Grundlage von Werkzeugen wie jconsole und jvisualvm sind**
- **Beispiel folgt**

4. Exkurs: Java Heaps (Heap or non-Heap?)

- **Eden Space (heap): pool from which memory is initially allocated for most objects.**
- **Survivor Space (heap): pool containing objects that have survived GC of eden space.**
- **Tenured Generation (heap): pool containing objects that have existed for some time in the survivor space.**
- **Permanent Generation (non-heap): holds all the reflective data of the virtual machine itself, such as class and method objects. With JVMs that use class data sharing, this generation is divided into read-only and read-write areas.**
- **Code Cache (non-heap): HotSpot JVM also includes a "code cache" containing memory used for compilation and storage of native code.**
- **Siehe auch http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf**

5. Exkurs: Garbage Collection

- **Garbage Collection (GC) ist CPU intensiv**
- **Teilweise reagiert Anwendung nicht, wenn GC aktiv**
- **VM Parameter um Garbage Collector Verhalten zu steuern**
- **Z.B. *-XX:+UseParallelGC -XX:ParallelGCThreads=4***
- **Parallele GC Threads empfohlen bei Muli Core CPUs. VM läuft meist nur in einer CPU (kein natives threading)**
- **Garbage Collection wird von der VM möglichst hinausgezögert -> *Performance***
- **Selektion des Kollektors**
  - –XX:+UseSerialGC
  - –XX:+UseParallelGC
  - –XX:+UseParallelOldGC
  - –XX:+UseConcMarkSweepGC
- **Es gibt also diverse Tuningmöglichkeiten des GCs**
- **Siehe auch**
  - http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html
  - http://download.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html#gddzt

6. Monitoring mit jconsole
JDK_HOME/bin/jconsole

- **jconsole nutz JMX für die Überwachung und Steuerung**
- **Liest also alle registrierten MBeans und listet deren Daten und Methoden**
- **MBean stellt Properties (ro/rw) und Methoden bereit**

Startdialog jconsole

7. Profiling mit jvisualvm
JDK_HOME/bin/jvisualvm

- **Bietet neben Profiling auch Monitoring wie jconsole**

7. Profiling mit jvisualvm
Literatur

Das Tuning bzw. die Analyse von Java Programmen erfordert viel Erfahrung und wissen über potenzielle Poblembereiche

- **„Java Performance Tuning" - O'Reilly von Jack Shirazi**
- **„Effective Java" – Sun von Joshua Bloch**
- **„Bitter Java" – Manning von Bruce A. Tate (lustig)**
- **„Better, Faster, Lighter Java" – O'Reilly von Bruce A. Tate und Justin Gehtland**

Diese Titel habe ich vor vielen Jahren gelesen. Es gibt sicher neue Auflagen bzw. sind die Weisheiten und Fakten in diesen Büchern immer gültig.

7. Profiling mit jvisualvm
Die Beispiele

Die Vorteile von Profiling bei der Analyse von Speichernutzung und Laufzeitverhalten werden anhand folgender Beispiele verdeutlicht

- **XSLT Verarbeitung: Vergleich von 3 Möglichkeiten der Transformation**
  - DOM, alle sagen DOM braucht viel Speicher und CPU. Wirklich?
  - SAX, ist doch eh viel besser als DOM, oder?
  - Stream, die Silberkugel….angeblich gibt es keine
- **String Konkatinierung in Schleife**
  - String, so etwas tut man doch nicht
  - StringBuffer, ja genau
  - StringBuilder, was ist das?

7. Profiling mit jvisualvm
Profiling CPU

Motivation:

- **Man denkt, was dauert den da so lange**
- **Kunde sagt die Anwendung bzw. eine bestimmte Funktion ist langsam**
- **Re-engineering**

7. Profiling mit jvisualvm
Profiling Memory

Motivation:

- **Speicherprobleme in Java? Ja, das gibt es**
- **Stichwort langlebige Objekte in Verbindung mit Caches (statische Maps etc.)**
- **Manchmal findet man allerdings nicht wirklich einen Programmierfehler oder einen ineffizienten Algorithmus. D.h. die Anwendung benötigt naturgemäß viel Speicher**
- **XML/XSL: SAX oder DOM?**
- **Tradeoff Performance <->Speicher**