Nr.02 | Sommer 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

Java im Aufwind

VisualVM
Unbekannte Kostbarkeiten des SDK

Grails
Die Suche ist vorbei

Microsoft und Java
Frei verfügbare Angebote für Software-Entwickler

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Sonderdruck

iJUG Verbund

Interview mit John Duimovich, Distinguished Engineer in der IBM Software Group mit Schwerpunkt „Java Virtual Machines & Embedded Java", IBM Canada, Seite 16

„Java besitzt immer noch ein enormes Potenzial …" Stefan Koospal, Vorsitzender der Sun User Group Deutschland, im Gespräch mit Java aktuell, Seite 42

Der Sport ist in manchen Bereichen hilfreich für die Entwicklung von Software. Was wir vom America's Cup lernen können, Seite 44

Unbekannte Kostbarkeiten des SDK
Heute: VisualVM

Bernd Müller, Ostfalia, Hochschule für angewandte Wissenschaften

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des SDK vorstellen: die unbekannten Kostbarkeiten.

Das Java SDK enthält bereits seit den allerersten Versionen eine Reihe von Kommandozeilen-Werkzeugen für die Untersuchung von Anwendungen, wie etwa „jps", „jstat", „jstatd", „jinfo", „jhat", „jmap" und „jstack". Mit dem SDK 5 wurde „jconsole" eingeführt, ein Monitoring-Werkzeug, das Informationen über die Performanz und den Ressourcen-Verbrauch von laufenden Java-Anwendungen grafisch darstellt. Mit dem Update 7 des SDK 6 wurde VisualVM eingeführt, nach eigenen Angaben das „All-in-One Java Troubleshooting Tool". VisualVM entstand im Rahmen des NetBeans-Projekts, ist als Stand-alone-Anwendung unter [1] herunterzuladen und – wie bereits erwähnt – in neueren SDKs enthalten.

VisualVM im Überblick

Als Kurzcharakterisierung von VisualVM geben wir die Beschreibung von [1] wieder: „VisualVM is a visual tool integrating several commandline JDK tools and lightweight profiling capabilities. Designed for both production and development time use, it further enhances the capability of monitoring and performance analysis for the Java SE platform."

Im Rahmen dieses Artikels können wir nur einen Überblick über die VisualVM geben und raten dem Leser, nach der Lektüre des Artikels selbst einen Blick auf dieses recht unbekannte, aber nützliche Werkzeug zu werfen. Nach dem Start von VisualVM (Programm „jvisualvm" im „bin"-Verzeichnis des SDK) erscheint das Hauptfenster. Wir verwenden VisualVM in der Version des SDK 6 Update 26.

Man erkennt im linken Teilfenster den Tab „Applications" mit den vier Knoten „Local", „Remote", „VM Coredumps" und „Snapshots". VisualVM erlaubt das Monitoring und Profiling von lokalen und entfernten JVMs. Um letztere mit VisualVM nutzen zu können, müssen einige zusätzliche Vorkehrungen getroffen werden, auf die wir hier jedoch nicht eingehen können. Weiterhin besteht die Möglichkeit, Coredumps zu analysieren sowie Snapshots zu erzeugen und ebenfalls zu analysieren.

Auf der rechten Seite erkennt man im Tab „Start Page" einige Überschriften, die Web-Links darstellen. Wir empfehlen dem Leser zum Einstieg in die SDK-Bordwerkzeuge folgende Links:

- Getting Started with VisualVM
- Troubleshooting Guide for Java SE 6
- Monitoring and Managing Java SE 6

Zurück zu den lokalen Anwendungen. Man erkennt VisualVM selbst, GlassFish, JMeter und Eclipse. Die zu monitorende Anwendung ist durch einen Doppelklick auszuwählen. Im rechten Teilfenster wird dann ein neuer Tab mit Informationen zur ausgewählten Anwendung dargestellt. Der Name des Tabs entspricht dem im linken Teilfenster selektierten Knoten. Der Tab selbst ist wiederum in Unter-Tabs aufgeteilt. Abbildung 1 zeigt den initialen Tab „Overview".

Die Tabs „JVM arguments" und „System properties" sollten die erste Anlaufstelle sein, wenn sich eine Anwendung unerklärlich verhält. Eventuell ist lediglich ein Property falsch definiert.

Monitoring von Anwendungen

Bei Server-Anwendungen kann es zu Speicherplatz-Problemen kommen, die sich durch einen „OutOfMemory"-Error oder häufige Garbage-Collection-Pausen bemerkbar machen. Zur Analyse derartiger Probleme können hochwertige kommerzielle Produkte – für eine erste (und eventuell ausreichende) Analyse aber auch VisualVM eingesetzt werden. In einem kleinen Beispiel wollen wir Objekte auf dem Heap anlegen und wieder freigeben, um den Einsatz des Garbage Collectors zu erzwingen und die Speicherplatz-Belegung und den Einsatz des Garbage Collectors mit VisualVM visualisieren zu können.

Zunächst benötigen wir ein Programm, um Speicherplatz (sinnlos) anzulegen und wieder freizugeben. Wir wählen JAX-RS als Implementierungs-Alternative, da eine Anwendung mit Swing oder ein Menü über „System.in/System.out" aufwändiger wäre. Der folgende Code legt pro Aufruf von „waste()" eine Liste von String-Arrays wachsender Länge an und erzeugt durch eine unglückliche Verwendung von „+" zur String Concatenation zusätzlichen Müll. Da die Liste eine lokale Variable der Methode ist, kann die ganze Struktur nach dem Methodenaufruf „garbage-collected" werden. Die Details der Implementierung sind nicht von Interesse (siehe Listing 1).

Durch die Definition als REST-Service und entsprechende Konfiguration von Anwendungsnamen (waste) und REST-Anwendungspfad (rest) kann mit jedem Browser, mit „wget" oder „curl" und dem folgenden URL der Speicher in Anspruch genommen werden: http://localhost:8080/waste/rest/waste/100. Dabei ist die Größe der Speicherstruktur frei wählbar. Auch der explizite Aufruf des Garbage Collectors kann einfach über REST realisiert werden (siehe Listing 2)

Der Garbage Collector kann nun ebenfalls sehr einfach aufgerufen werden: http://localhost:8080/waste/rest/gc. Für den REST-erfahrenen Leser: Wir sind uns darüber im Klaren, dass „GET" in beiden

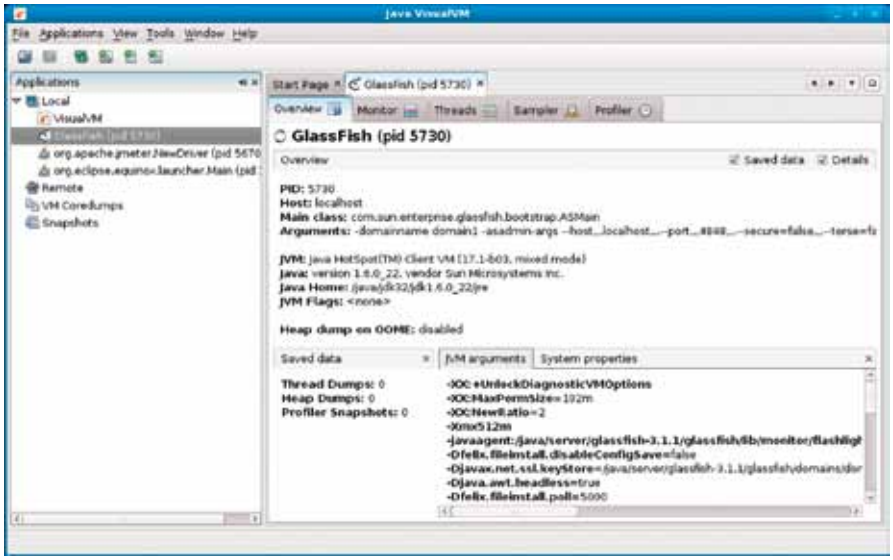

Abbildung 1: Overview mit „JVM arguments“ und „System properties“

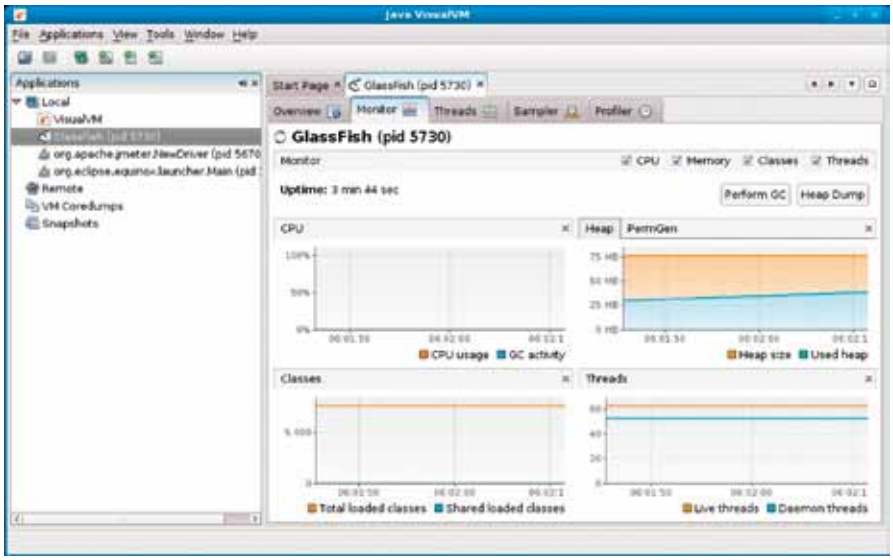

Abbildung 2: Monitor-Tab mit vier Teilbereichen

Fällen nicht die korrekte und von REST intendierte Alternative ist. Sie ist aber am einfachsten zu verwenden.

Zurück zur VisualVM. Unter dem Tab „Monitor“ erscheint eine Anzeige mit den vier Bereichen „CPU“, „Heap/PermGen“, „Classes“ und „Threads“ (siehe Abbildung 2). Da die Darstellung der Klassen und Threads für unser Beispiel irrelevant ist, können diese beiden Teilbereiche entfernt werden. Mit den beiden verbliebenen Bereichen „CPU“ und „Heap“ starten wir nun unseren Versuch. Mit „Jmeter“, einem Werkzeug für Load-Tests und Performanz-Messungen der Apache Software Foundation [2], rufen wir mehrfach die URL „http://localhost:8080/waste/rest/waste/{size}“ auf. Der konkrete Wert von „size“ ist nicht relevant und muss für die jeweilige Umgebung eventuell angepasst werden, um die gewünschte Speichernutzung zu erzwingen. Abbildung 3 zeigt die Darstellung in VisualVM.

Man erkennt, dass der mehrmalige Aufruf der Methode entsprechende Speicher-Allokationen verursacht hat. Um insgesamt mit dem Speicher auskommen zu können, hat die JVM den Garbage Collector mehrfach aufgerufen (Kurve „GC activity“ im linken Diagramm). Die JVM hat außerdem den Heap-Bereich auf 500 MB erweitert, was dem (konfiguriert) maximal verfügbaren Heap entspricht.

Das Plug-in „Visual GC“

Die Reihe „Unbekannte Kostbarkeiten“ stellt SDK-Bordmittel vor. Das Thema „Speichernutzung und Garbage Collection mit VisualVM“ wäre aber ohne das Plug-in „Visual GC“ nicht annähernd vollständig. VisualVM erlaubt durch einen einfachen Plug-in-Mechanismus die Erweiterung um verschiedene Aspekte. Eine Reihe von Plug-ins können von [1] heruntergeladen und installiert werden. Über das Menü „Tools -> Plugins“ geschieht dies über wenige Klicks, sodass durchaus von einem SDK-Bordmittel gesprochen werden kann. Wir gehen außerdem davon aus, dass in einer zukünftigen Version von VisualVM dieses Plug-in bereits enthalten sein wird.

```
@Path(„/“)
public class WasteService {

  @GET
  @Path(„waste/{size}“)
  @Produces(MediaType.TEXT_PLAIN)
  public String waste(@PathParam(„size“) int size) {
    List<String[]> waste = new ArrayList<String[]>();
    for (int i = 1; i < size; i++) {
      waste.add(consume(i));
    }
    return „wasted“;
  }

  private String[] consume(int i) {
    String[] waste = new String[i];
    waste[0] = new String(„1“);
    int digit = 1;
    for (int j = 1; j < waste.length; j++) {
      waste[j] = waste[j-1] + (++digit) % 10;
    }
    return waste;
  }
}
```

Listing 1

```
@Path(„/“)
public class WasteService {
  ...
  @GET
  @Path(„gc“)
  @Produces(MediaType.TEXT_PLAIN)
  public String gc() {
    System.gc();
    return „gc called“;
  }
  ...
}
```

Listing 2

Abbildung 5 zeigt VisualVM nach der Installation von Visual GC und dem geöffneten Tab „Visual GC".

Die Standard-Darstellung des Heap in VisualVM unterscheidet nicht zwischen den verschiedenen Heap-Bereichen (siehe Abbildung 3). Mit Visual GC ist es möglich, diese Bereiche einzeln zu beobachten. In Abbildung 4 ist der Bereich für die Old Generation und die Young Generation zu erkennen. Die Young Generation ist wiederum in die Bereiche „Eden", „Survivor 0" und „Survivor 1" unterteilt. Oracles VM erlaubt unter anderem mit den Startoptionen „-Xmn", „-XX:NewSize", „-XX:MaxNewSize", „-XX:NewRatio" und „-XX:SurvivorRatio" das Fein-Tuning des Heap. Um diese Optionen sinnvoll einsetzen zu können, muss zunächst das Verhalten dieser Speicher-Bereiche auf potenzielle Probleme hin analysiert werden. Die Verwendung von VisualVM ist eine Möglichkeit, dies zu tun.

Fazit

Wir haben die verborgene Kostbarkeit „VisualVM" vorgestellt, die seit Java 6 Update 7 im SDK enthalten ist. Mit ihr ist es möglich, den Speicherbedarf einer Anwendung und das Garbage-Collection-Verhalten einer VM zu analysieren. Leider war nur ein kleiner Einblick in die Features von VisualVM möglich. Nicht vorgestellt haben wir das Profiling. So können etwa für eine Klasse die Anzahl der Instanzen und deren Speicherbedarf sowie die Laufzeit einer Methode ermittelt werden. Durch weitere Plug-ins, etwa für das Monitoring des GlassFish-Application-Servers oder die Anzeige von MBeans-Details, kann VisualVM mit zusätzlicher Funktionalität versehen werden.

Wir raten, VisualVM auszuprobieren und sich ein eigenes Bild von den zur Verfügung stehenden Möglichkeiten zu machen. Für weitere Informationen zu den genannten Themen ist das Buch von Hunt und John empfohlen [3].

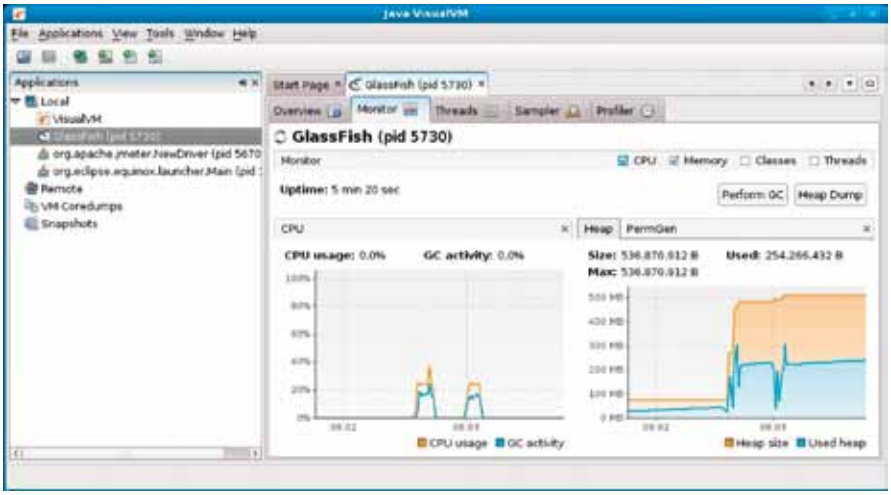

Abbildung 3: CPU- und Heap-Auslastung für den Test

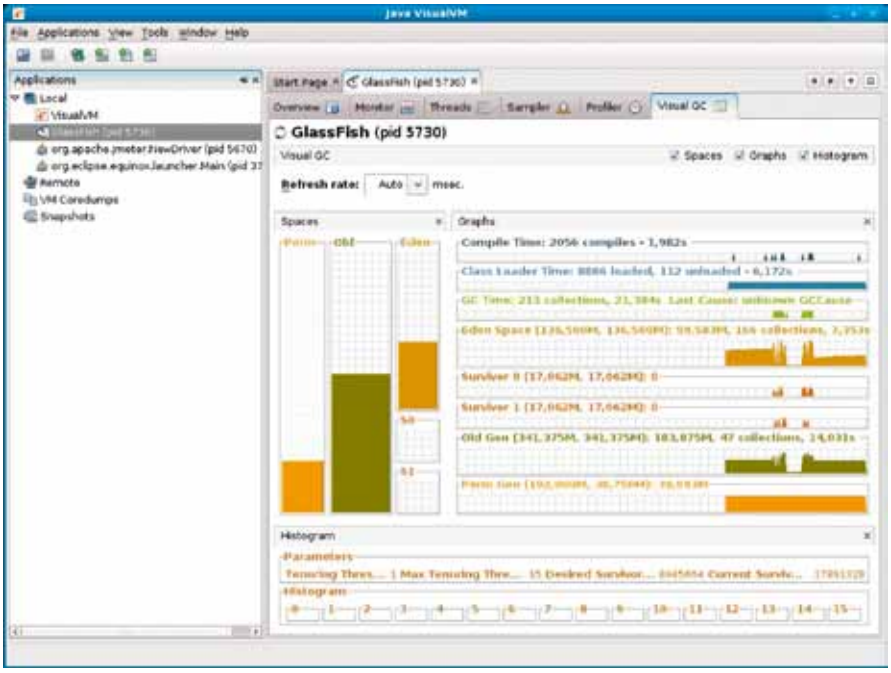

Abbildung 4: Das Tab des Plug-ins Visual GC

Weitere Informationen

[1] http://visualvm.java.net
[2] http://jmeter.apache.org
[3] Charlie Hunt, Binu John. Java Performance. Addison Wesley, 2012

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/go/shop

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

Ja, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

Ja, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.