Wo laufen sie denn?

SAP JVM Profiler

Matthias Braun, Michael Wintergerst

Der SAP JVM Profiler erlaubt es Ihnen, schnell und einfach Java-Applikationen zu analysieren und die Probleme in Ihren Applikationen aufzuzeigen.

Auf den ersten Blick ist Java gerade in Bezug auf Ressourcen-Nutzung und -Verbrauch eine sehr komfortable Programmiersprache: Eine automatische Speicherverwaltung ist integraler Bestandteil der Laufzeitumgebung und der Just-in-Time-Compiler erstellt hocheffizienten Maschinencode zur schnellstmöglichen Ausführung von Java-Programmen.

Leider ist das nur die halbe Wahrheit. Nicht selten haben ungeschickt entworfene Java-Applikationen einen sehr hohen Speicher- und CPU-Verbrauch. Außerdem kommt es leicht zu Situationen, in denen die Synchronisation der verschiedenen Anwendungs-Threads nicht optimal abgestimmt ist, wodurch sich die Programmabarbeitung unnötig verzögert.

SAP JVM & SAP JVM Profiler

Die SAP JVM [SAPJVM,SchWi13] positioniert sich als die virtuelle Maschine (VM) für Entwickler und hochverfügbare Applikationsserver. Gerade bei Entwicklung und Betrieb komplexer und skalierbarer Anwendungen ist eine genaue Analyse des Laufzeit- und Speicherverhaltens von enormer Bedeutung. Um eine einfache und effiziente Analyse Java-basierter Systeme zu ermöglichen und potenzielle Schwachstellen ausfindig zu machen, bietet die SAP JVM eine Reihe zusätzlicher Werkzeuge und Hilfsmittel an, die weit über den üblichen Funktionsumfang gängiger Analyse-Tools hinausgehen. Das mächtigste Werkzeug ist hier der sogenannte SAP JVM Profiler, eine vollständige Monitoring- und Profiling-Lösung inklusive grafischer Aufbereitung.

Zum Funktionsumfang gehört eine detaillierte Garbage Collection (GC)-Analyse, eine Allokations- und Performance-Analyse sowie Datei- und Netzwerk-Traces, mit deren Hilfe das E/A-Verhalten einer JVM beobachtet werden kann. Abrundend bietet ein Synchronisations-Trace die Möglichkeit, Nebenläufigkeit und Kooperation der einzelnen Applikations-Threads im Auge zu behalten.

Download: SAP HANA Cloud

Zum lokalen Entwickeln in der SAP HANA Cloud [HANA] wird die SAP JVM zum freien Download zur Verfügung gestellt: `https://tools.hana.ondemand.com/#cloud`. Die entsprechenden Entwicklerwerkzeuge, wie den SAP JVM Profiler, erhalten Sie über die SAP Eclipse Update Site: `https://tools.hana.ondemand.com/kepler`. Dokumentationen zum SAP JVM Profiler finden Sie integriert in der Eclipse-Hilfe.

Architektur

Der SAP JVM Profiler gliedert sich in zwei wesentliche Komponenten: ein in die SAP JVM integriertes Backend für das Sammeln der Analysedaten sowie ein Eclipse-basiertes Frontend für die grafische Darstellung und Aufbereitung der Informationen. Im Folgenden wird die Bezeichnung SAP JVM Profiler als Synonym für das grafische Frontend verwendet. Das Eclipse-Plug-in können Sie direkt in eine gewöhnliche Eclipse-Entwicklungsumgebung für Java einbinden. Unterstützt werden alle Releases ab Version 3.3. Freigegeben ist das Frontend für die Betriebssysteme Windows, Linux und Mac OS X.

Zu beachten ist, dass der SAP JVM Profiler nur zum Profilen einer Applikation, die auf der SAP JVM läuft, benutzt werden kann. Im Gegensatz zu anderen Profiler-Lösungen wie YourKit [YourKit] oder JProbe [JProbe] baut die SAP-Lösung nicht auf dem Java Virtual Machine Tool Interface [JVMTI] auf. Stattdessen wurde das Profiling-Backend direkt in die Kernkomponenten der SAP JVM (z. B. GC, Thread- und Memory-Management) integriert. Dadurch entstehen gerade im Hinblick auf Laufzeit-Overhead, Speicherplatzbedarf und Präzision der Profiling-Daten enorme Vorteile.

Nehmen wir beispielsweise die GC-Analyse. Eine GC ist sicherlich eine der komplexesten Komponenten einer JVM. Sie spaltet sich in diverse Phasen und Ereignisse auf, die abhängig von der jeweiligen Implementierung (Garbage First, Concurrent-Mark-Sweep, Parallel-Old GC usw.) sind. Durch die direkte Integration des Backends in die GC-Implementierungen lassen sich Detail-Informationen zu allen GC-Phasen effizient ermitteln.

Thread-Annotationen

Ein weiterer Vorteil dieser Architektur ist die Möglichkeit zur Annotation der Profiling-Daten mit benutzerdefinierten Werten. Die SAP JVM bietet ein Java-API an, um den laufenden Thread mit beliebigen String-Schlüssel-Wert-Paaren zu annotieren. Es gibt vier vordefinierte Annotationen: *User*, *Request*, *Session* und *Application*. Für jede dieser Annotationen kann der aktuelle Wert als String für den Thread gesetzt werden. Bei Bedarf können zusätzliche Annotationen definiert werden. Die User-Annotation wird dann von der VM beispielsweise beim Profiling dafür verwendet, die Ressourcen ihren Benutzern zuzuordnen. Darüber hinaus können diese Daten auch dazu dienen, den Profiling-Overhead zu verringern. Der SAP JVM Profiler bietet daher die Möglichkeit, einen Profiling-Trace nur für einen bestimmten Benutzer oder eine bestimmte Applikation zu starten. Alle anderen Benutzer oder Applikationen werden nicht beeinträchtigt.

On-Demand Profiling

Eine weitere wesentliche Eigenschaft, die den SAP JVM Profiler von anderen Profiling-Lösung unterscheidet, ist, dass die SAP JVM „on-the-fly“ in den Profiling-Modus geschaltet werden kann. Das heißt, Sie können in einer produktiv laufenden SAP JVM die Profiling-Funktionalität einschalten, ohne die JVM durchzustarten. Probleme, die in einer JVM auftreten, können damit direkt analysiert werden. Es ist nicht notwendig, die Situationen in einer speziell gestarteten JVM nachzustellen. Diese Funktionalität ist gerade beim Betrieb von produktiven Systemen von enormer Bedeutung.

Eine laufende SAP JVM kann direkt über den SAP JVM Profiler in den Profiling-Modus versetzt werden. Das grafische Frontend bietet dazu eine eigene Eclipse-Perspektive mit den dazugehörigen zentralen *PROFILE*- und *VM EXPLORER*-Views. Der *PROFILE*-View stellt Ihnen Informationen zu allen gestarteten Profiling-Läufen mit den dazugehörigen Traces und geöffneten Statistiken bereit.

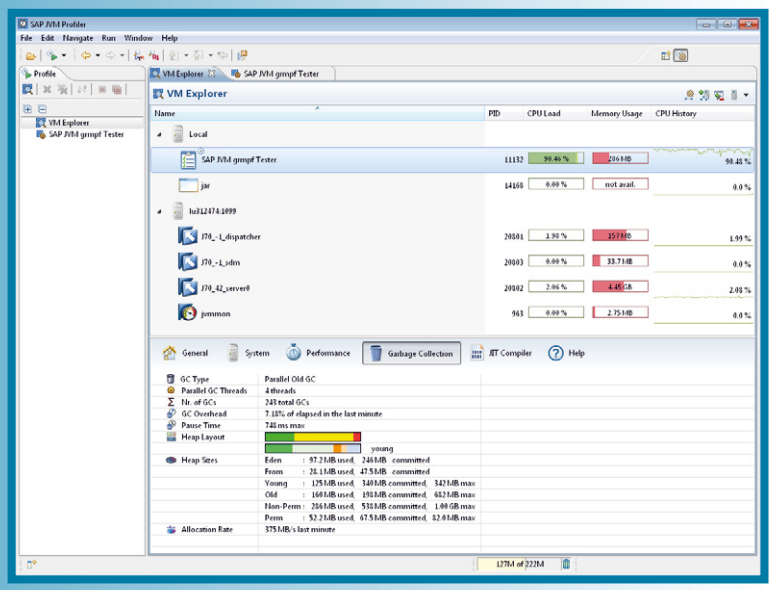

Abb. 1: SAP JVM Profiler – VM Explorer

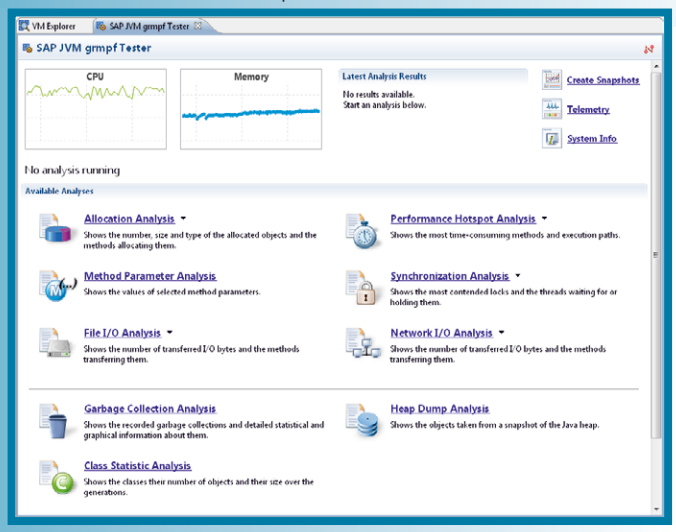

Abb. 2: Connection View – Starten Sie Ihre Analyse

Der *VM EXPLORER* gibt Ihnen einen Überblick über laufende SAP JVM-Prozesse. Ohne zusätzliche Konfiguration werden Ihnen alle auf dem lokalen Rechner vorhandenen SAP JVMs mit detaillierten Monitoring-Informationen angezeigt. Damit der Profiler JVMs auf entfernten Rechnern erkennen und sich mit ihnen verbinden kann, muss auf dem Rechner eine *jvmmond*-Instanz gestartet sein. Das *jvmmond*-Programm finden Sie im bin-Verzeichnis Ihrer SAP JVM Runtime. Es stellt über einen geöffneten TCP-Port Monitoring-Funktionalität bereit. Sie können im *VM EXPLORER* den Rechner eintragen, indem Sie entweder über das Kontext-Menü oder in der Werkzeugleiste *MANAGE HOSTS* aufrufen und im erscheinenden Dialog den neuen Rechner mit Namen oder IP-Adresse samt TCP-Port eintragen.

Neben den lokalen JVMs sehen Sie dann im *VM EXPLORER* auch die JVMs, die auf dem neu eingetragenen Rechner laufen. Die Spalten des *VM EXPLORER*s zeigen Ihnen sowohl statische Werte, wie die Prozess-ID und den Startzeitpunkt, als auch dynamische Werte, wie die aktuelle CPU-Last und Speicher-Belegung. Einige weitere Attribute der SAP JVM können Sie über den Kontextmenü-Eintrag *ADD OR REMOVE COLUMNS* oder über die gleichnamige Schaltfläche in der Werkzeugleiste einblenden.

Die angezeigten Werte sollen Ihnen einen ersten Eindruck vom Zustand der JVMs auf dem System vermitteln. Eine dauerhaft hohe CPU- oder Speicher-Auslastung könnte auf ein ernsthaftes Problem hinweisen. Wenn Sie einen Server-Knoten genauer unter die Lupe nehmen möchten, führen Sie einfach einen Doppelklick auf die zugehörige Zeile im *VM EXPLORER* aus (alternativ über das Kontext-Menü *Connect*), und der Profiler baut daraufhin eine Verbindung zur SAP JVM auf.

Neben der Anzeige von grundlegenden Informationen zur ausgewählten JVM dient der sich öffnende *CONNECTION VIEW* insbesondere als Ausgangspunkt zum Starten und Stoppen von Profiling-Traces. Im oberen Bereich der Übersicht in Abbildung 2 können Sie den zeitlichen Verlauf von CPU-Auslastung und Speicherverbrauch verfolgen.

Profiling-Traces

Der Profiler bietet Ihnen zur Beobachtung der unterschiedlichen Ressourcen mehrere Traces an, die Sie zur Laufzeit je nach Bedarf aktivieren und nach dem gewünschten Analysezeitraum wieder deaktivieren können. Das Profiling-Backend der SAP JVM sammelt bei eingeschaltetem Trace die entsprechenden Analysedaten und schickt sie zur weiteren Verarbeitung zum Frontend. Die durch einen Trace gesammelten Daten der JVM können dann mit Hilfe der im Frontend angebotenen Statistiken eingehend analysiert und ausgewertet werden. Es stehen Ihnen hierbei die in Tabelle 1 aufgeführten Traces und Analysemöglichkeiten zur Verfügung.

Profiling Trace	Beschreibung
Allocation Analysis	Protokolliert sämtliche Objekt-Erzeugungen und bildet somit den Speicherverbrauch über die Zeit ab
Performance Hotspot Analysis	Deckt CPU-intensive Methodenaufrufe und Ausführungspfade auf
Synchronization Trace	Informiert Sie über Blockierungen von Threads aufgrund von Sperren
File & Network I/O Analysis	Erfasst sämtliche E/A-Operationen der SAP JVM
Method Parameter Trace (MPT)	Überwacht die Anzahl der Aufrufe und Über- oder Rückgabeparameter bestimmter Methoden
GC Analysis	Analysiert GC-Operationen inkl. einzelner Phasen und Ereignisse
Class Statistic	Ermittelt die geladenen Klassen und die entsprechende Anzahl von Objekten
Heap Dump	Triggert das Erstellen eines Java-Heap-Dumps

Tabelle 1: Profiling-Traces und Analysemöglichkeiten

Integration Memory Analyzer

Wie oben beschrieben, lassen sich Heap-Dumps über den SAP JVM Profiler erstellen. Diese können in das bekannte HPROF-Format exportiert oder bei installiertem *Memory Analyzer* [MAT] direkt geöffnet werden. Der Memory Analyzer ist ein von SAP initiiertes Open-Source-Projekt zur Unterstützung der Java-Entwickler bei der Analyse von Java-Heaps. Trotz ausgefeilter Speicherverwaltung in der JVM können bei der Programmausführung *Memory-Leaks* entstehen – unter anderem

durch statische Objekte, die transitiv riesige Objekt-Hierarchien referenzieren und damit dauerhaft am Leben halten.

Wachsen Memory-Leaks mit fortschreitender Programmausführung weiter an, wird es früher oder später zu einer schwerwiegenden `OutOfMemoryException` kommen, die in aller Regel ein Durchstarten des JVM-Prozesses erzwingt. Der Memory Analyzer eignet sich besonders zur Analyse solcher Situationen, vorzugsweise sollte er aber schon in der Entwicklungsphase zur Vermeidung besonders Speicher hungriger Objekte eingesetzt werden.

Kombiniert mit dem Allocation-Trace lassen sich sogar die Quellcodestellen aufspüren, an denen die lebenden Objekte erzeugt wurden.

Performance Hotspot Trace

Gemäß dem bekannten Prinzip von Pareto (auch bekannt als 80/20-Regel) ist das Laufzeitverhalten der Anwendung nur durch kleine Anteile des Programmcodes bestimmt. Eine naheliegende Strategie ist demnach, sich bei der Performance-Optimierung auf diese wenigen Stellen im Programm zu konzentrieren, die einen wesentlichen Anteil des CPU-Verbrauchs ausmachen. Leider lassen sich diese sogenannten Hotspots durch Quellcode-Analyse nur sehr schwer auffinden. Der SAP JVM Profiler bietet daher die Möglichkeit, am laufenden System eine Performance-Messung zuzuschalten, aus der sich die in Anspruch genommene CPU-Zeit der ausgeführten Methoden ableiten lässt. Besonders teure Methodenaufrufe können anschließend mühelos aus den aufbereiteten Messdaten extrahiert werden.

Anstatt die Abarbeitungszeit jedes einzelnen Methodenaufrufs gesondert zu erfassen, basiert der Performance Hotspot Trace der SAP JVM auf einem stochastischen Ansatz, dem *Time-based Sampling*. Dabei wird die Anwendung periodisch in kurzen Abständen von ca. 10 Millisekunden angehalten, um den Zustand aller Anwendungs-Threads zu inspizieren. Die erhaltenen Profiling-Daten – bestehend aus dem Callstack des Threads (einschließlich der aktuell ausgeführten Methode) und der verbrauchten CPU-Zeit – werden als *Samples* zusammengefasst und kontinuierlich zum Frontend gesendet. Um Messergebnisse mit bestmöglicher Güte zu erzielen, ist es notwendig, das Sampling über einen genügend langen Zeitraum vorzunehmen.

Bei der sich anschließenden Analyse erlauben die gesammelten Samples die Rekonstruktion der teuersten Ausführungspfade in der Anwendung bis hin zur Ebene von Programmcode-Zeilen. Im Profiler stehen Ihnen hierfür verschiedene Methoden-Statistiken zur Verfügung – unter anderem auch eine Variante, in der Sie nach dem Top-Down-Prinzip die teuersten Pfade in der Aufruf-Hierarchie aufspüren können (s. Abb. 3). Ausgefeilte Filter- und Sortierfunktionen helfen Ihnen dabei, die wesentlichen Aspekte schnell aus der Fülle an Informationen herauszuarbeiten.

In großen Softwaresystemen mit mehreren hundert Threads hat sich das Timebased Sampling – ganz im Gegensatz zu auf Code-Instrumentierung basierenden Ansätzen – als höchst praxistauglich erwiesen. Von den kurzen Pausenzeiten in der Anwendungsausführung abgesehen, die durch das Sampling erforderlich werden, bleibt die Anwendung im Wesentlichen unberührt und verzeichnet einen überschaubaren Overhead von lediglich 10 bis 15 Prozent. Dies ermöglicht auch einen Einsatz im produktiven Modus.

Method Parameter Trace

Ein sehr vielseitig einsetzbares Feature des Profilers ist der Method Parameter Trace (MPT), bei dem die SAP JVM die Übergabeparameter oder Rückgabewerte von zuvor festgelegten Methoden mitprotokolliert. Darüber hinaus wird auch die Ausführungszeit der Methodenaufrufe gemessen, der aktuelle Callstack des ausführenden Threads aufgezeichnet und ein Zeitstempel erzeugt. Mit Hilfe dieser Daten können Sie im Profiler genau zurückverfolgen, in welchem Kontext und zu welchen Zeitpunkten bestimmte Methoden mit den entsprechenden Parametern aufgerufen worden sind.

Wenn Sie beispielsweise für die JDK-Methode `java.io.File InputStream(java.io.File)` den einzigen Eingabeparameter vom Typ `File` erfassen lassen, erhalten Sie einen Überblick über alle Dateien, die innerhalb des Analysezeitraums geöffnet wurden. Beim Ausweisen dieser Objekte in der Statistik kommt ein sogenannter *Modifikator* zum Einsatz, der die (Parameter-)Objekte entweder auf einen Basistyp (`int`, `float` usw.) oder auf `String` abbildet. Für die im Beispiel aufgeführten `File`-Objekte bietet sich etwa die Methode `getName()` an, um diese Abbildung zu gewährleisten. Bei der Ausgabe in der Tabelle werden dann die Dateinamen angezeigt.

Ein weiteres prominentes Anwendungsbeispiel ist das Tracen von SQL-Anweisungen in der Datenbank-Schnittstelle, um kostenintensiven Datenbank-Abfragen auf die Spur zu kommen. Diese spezielle Analyse steht übrigens bereits vorgefertigt im Profiler zur Verfügung und braucht bei Bedarf nur noch aktiviert zu werden. Auch bei der Fehleranalyse leistet der MPT hilfreiche Dienste. So kann er alle Änderungen an `System`-Propertys protokollieren oder detailliert aufschlüsseln, mit welchem `ClassLoader` eine Java-Klasse geladen wurde.

Im Gegensatz zum Performance Hotspot Trace basiert der MPT auf der Instrumentierung der zu tracenden Methode(n). Um die Zeitmessung zu bewerkstelligen, wird bei jedem Methoden-Eintritt und -Austritt ein Zeitstempel generiert und aus dem konfigurierten Parameter der aktuelle Wert abgeleitet. Weil sich dadurch der Anwendungscode ändert und die betroffenen Methoden erneut kompiliert werden müssen, sollten Sie den Einsatz des MPT in produktiven Systemen auf eine überschaubar große Anzahl von Methoden-Parameter beschränken.

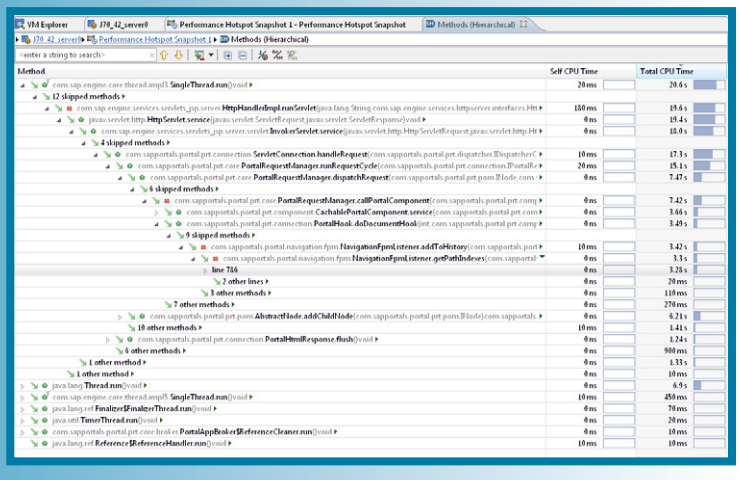

Abb. 3: Hierarchische Methodenstatistik

GC-Analyse

Die automatische Speicherverwaltung (GC), ein zentraler Bestandteil der JVM, übt besonders in Systemen mit einem sehr großen Java-Heap einen nicht zu unterschätzenden Einfluss auf die Gesamt-Performance aus. Die Optimierungsaufgabe liegt generell darin, ein für die spezielle Anwendung optimales Verhältnis zwischen

Abb. 4: GC-Analyse-Graph

Dauer und Häufigkeit der GCs zu erzielen. Dies ist nur dadurch erreichbar, dass Sie die GC-Parameter (also insbesondere die Größe der Heap-Generationen und den Kollektor-Typ) mit dem Lastverhalten der Anwendung abstimmen. Eine unbedachte Wahl dieser Parameter kann schnell zu einem unerwarteten Systemverhalten führen. Wenn beispielweise kurze Antwortzeiten vom System erwartet werden, sind Pausenzeiten im Sekundenbereich für die Durchführung einer GC nicht akzeptabel. In diesem Fall sollte ein Kollektor-Typ zum Einsatz kommen, der mit hoher Wahrscheinlichkeit Full GCs vermeidet, wie der Concurrent-Mark-Sweep (CMS)-Kollektor. Nebenbei wollen auch Heap-Bereiche wie die Permanent Generation richtig dimensioniert sein, um fatale OutOfMemory-Situationen zu vermeiden.

Um das Verhalten der GC in einem konkreten Anwendungsfall genau beobachten und gegebenenfalls Rückschlüsse auf die gewählten GC-Einstellungen ziehen zu können, bietet der SAP JVM Profiler eine detaillierte GC-Analyse an, die es Ihnen ermöglicht, die Auswirkungen jeder einzelnen GC auf das Gesamtsystem bis in die einzelnen GC-Phasen hinein zu untersuchen. Wie bei all den anderen Profiling-Traces können Sie die GC-Informationen direkt von einer laufenden JVM abgreifen. Oder aber Sie laden nachträglich das sogenannte *GC-History File* in den Profiler, in welches eine SAP JVM-Instanz bei gesetzter Option `-XX:+GCHistory` sämtliche GC-Ereignisse archiviert.

Aus den gesammelten GC-Ereignissen generiert Ihnen der Profiler zunächst eine grobe Übersicht, die Ihnen schnell einen Eindruck vermittelt, wie viele GCs stattgefunden haben und wie hoch der Gesamtaufwand dafür war.

In der *GC-Statistik* sind die Daten hingegen tabellarisch bis ins kleinste Detail analysierbar. Neben dem Zeitpunkt und der Ausprägung einer GC (partiell oder vollständig) sind unter anderem auch der Grund für die GC, ihre Auswirkung auf den Heap sowie die maximale Pausenzeit ablesbar. Artefakte, wie GCs mit besonders langen Pausenzeiten oder solche mit unerwünschten *Promotion Failures*, sind auf einen Blick erkennbar. Ebenfalls sehr hilfreich ist die Vergleichsfunktion, mit der Sie die Attribute verschiedener GCs gegenüberstellen können.

Um die Füllstände der einzelnen Heap-Generationen besser im Auge zu behalten, bietet sich die graphische Analyse der GC-Ereignisse im zeitlichen Verlauf an.

Im oberen Graph der Abbildung 4 sind nebenläufige GC-Phasen zusammen mit ihrer Inanspruchnahme an CPU dargestellt. Full GCs stechen durch Spitzen bei der Pausenzeit hervor. Im mittleren Graph ist der typisch sägezahnartige Verlauf des Füllstands in der *Old Generation* deutlich zu erkennen.

Fazit

Der Artikel hat Ihnen einen ersten Einblick in die Analyse-Werkzeuge der SAP JVM gegeben. Der SAP JVM Profiler bietet Ihnen die Möglichkeit, in einfacher Art und Weise in das Laufzeit-Verhalten Ihrer Applikation zu schauen. In einem schon geplanten Artikel werden wir im Detail auf ein weiteres Werkzeug, den SAP JVM Debugger, eingehen. Dieser erlaubt Ihnen, Ihre Java-Applikationen in der SAP HANA Cloud zu debuggen, und ist daher besonders für das Java-Debugging über „Wide Area Network"-Verbindungen ausgelegt.

Literatur und Links

[HANA] `https://help.hana.ondemand.com/`
[JProbe] `http://jprobe.software.informer.com/`
[JVMTI] `http://docs.oracle.com/javase/7/docs/technotes/guides/jvmti/`
[MAT] `http://www.eclipse.org/mat/`
[SAPJVM] `https://help.hana.ondemand.com/help/frameset.htm?da030d10d97610149defa1084cb0b2f1.html`
[SchWi13] R. Schmelter, M. Wintergerst, Die VM für Entwickler und hochverfügbare Applikationsserver, in: JavaSPEKTRUM, 5/2013
[Yourkit] `http://www.yourkit.com/overview/index.jsp`

Matthias Braun widmet sich seit über zehn Jahren der professionellen Softwareentwicklung. Als ein führender Entwickler im SAP JVM-Team weist er ein sehr breitgefächertes und fundiertes Fachwissen in dem Themenkomplex auf.
E-Mail: matthia.braun@sap.com

Michael Wintergerst beschäftigt sich nunmehr seit über zehn Jahren mit dem Thema JVM innerhalb der SAP AG. Er leitet als Development Manager die Entwicklungsabteilung der SAP JVM.
E-Mail: michael.wintergerst@sap.com