Java-Diagnose mit dem Oracle Enterprise Manager 13c

Marcus Schröder
Oracle Deutschland B.V. & Co KG
Nürnberg

Schlüsselworte
EM13c FMW Java Diagnose

Einleitung

Mit dem Oracle Enterprise Manager 13c liefert Oracle die nächste Generation des erfolgreichen Manageability-Produkts. Einen interessanten Teilbereich der Lösung stellt das Thema Java-Diagnose dar. Die Funktionalität wurde in der Version 13c des Enterprise Managers vollständig überarbeitet. Der folgende Vortrag befasst sich mit den Neuerungen, zeigt Unterschiede zu älteren Versionen und gibt Praxistipps für die Nutzung

Was ist der Oracle Enterprise Manager Cloud Control 13c?

Oracle Cloud Control ist das zentrale System-Management-Tool von Oracle. Es dient dazu, nahezu alle Oracle-Produkte zu überwachen, diagnostizieren und im Lifecycle-Management zu unterstützen. Die Kurzbezeichnung für den Oracle Enterprise Manager Cloud Control 13c lautet EM13c oder auch OEM13c (Oracle Enterprise Manager). EM13c ist ein zentrales Management-Tool, d. h. es wird in einer unabhängigen Infrastruktur betrieben. EM13c ist als Java-EE-Webanwendung konzipiert und verwendet eine Oracle-Datenbank als Repository, den Oracle WebLogic-Server als Java-EE-Applikationsserver und Java-SE-Applikationen als Management Agent.

Ein wichtiger Bereich des EM13c ist die Diagnose von Java-basierten Applikationsservern und darauf installierten Anwendungen. Bevor wir in diesem Artikel näher auf die Lösung eingehen, wird erörtert, was Java-Diagnose ist, wofür es verwendet wird und welche allgemeinen Lösungen am Markt existieren.

Java-Diagnose Basics

Im Kern geht es bei der Java-Diagnose um das Monitoren und Verwalten von Java-Applikationen, die als Stand-Alone-Applikation oder Java EE implementiert sind. Die Java-Diagnose ist ein Teilbereich des Application Performance Management, welches wiederum im Bereich IT-Systemmanagement zu finden ist.

In einer Java-Applikation bzw. in dem darunterliegenden Java-Container können unterschiedliche Ausprägungen von Fehlern/Ausnahmen auftreten. Diese Fehler/Ausnahmen können zu einem Ausfall der Anwendungen bzw. zu Performance-Einbußen führen. Typische Fehler sind:

- Programmabbruch mit einer nicht abgefangenen Java-Fehlermeldung
- Out-of-Memory-Fehler, der Java-Speicher ist nicht ausreichend dimensioniert und das Programm stürzt ab
- Speicher-Lecks, das Programm benötigt immer mehr Speicher, bis die Obergrenze des zugewiesenen Speichers erreicht wird
- Stuck-Threads, Programm-Threads behindern sich gegenseitig in der weiteren Ausführung
- Performance-Probleme, die innerhalb des Programms oder bei Zugriffen auf Ressourcen auftreten

Diese Fehler/Ausnahmen zu finden, ist sehr zeitaufwendig. Im Allgemeinen gilt, je später der Fehler in einem Programm-Lebenszyklus auftritt, desto komplizierter ist er zu finden. Um den Entwicklern

und Operatoren der Java-Anwendung bei der Fehlersuche zu unterstützen, existiert eine Vielzahl von Tools und Schnittstellen innerhalb des JDKs/JREs. Die Nutzung ermöglicht die Diagnose ohne externe Tools. Vorteile: Keine zusätzlichen Kosten und direkter Zugriff auf die Tools/Schnittstellen. Nachteil: Wenn die Anzahl der überwachten JVMs steigt, steigt auch die Komplexität der Überwachung, dies ist oft mit „Bordmitteln“ nicht mehr zu leisten.

Das haben Softwarehersteller erkannt und eine Vielzahl von Diagnosetools entwickelt, die sich der JDK-/JRE-Schnittstellen (Java Runtime Environment/Java Development Kit) bedienen bzw. Java-Mechanismen verwenden, um komplexe Szenarien überwachen zu können. Ein weiterer Punkt ist, dass die Java-Applikationen/Java-EE-Server auf externe Schnittstellen zugreifen, die in die Überwachung einbezogen werden müssen. Dies sind zum Beispiel: Datenbanken, Datei-Systeme, Messaging-Services, Web-Services etc., die in Java oder anderen Programmiersprachen implementiert werden.

Die Implementierung der externen Diagnosetools bedient sich zwei unterschiedlichen Ansätzen:

- Byte-Code-Instrumentierung (BCI)
- API-Nutzung

Bei der BCI wird Byte-Code während der Laufzeit bzw. beim Laden der Klasse hinzugefügt. Dieser Byte-Code wird zusammen mit dem Java-Objekt ausgeführt und ermittelt Informationen zum Programmablauf. Vorteil: Man erhält sehr detaillierte Informationen und kann eine Java-Transaktion vollständig verfolgen. Nachteil: Wendet man diese Technik auf alle Programmkomponenten an, wird ein erheblicher Mehraufwand verursacht, der sich in der Performance widerspiegelt. Daher werden bei der BCI nur zentrale Programmkomponenten bzw. Teilbereiche überwacht. Zusätzlich ist BCI eine, wenn auch gewollte, Änderung des ablaufenden Programmes. Es kann in seltenen Fällen dazu führen, dass diese Änderungen ebenfalls Fehler nach sich ziehen.

Bei der API-Nutzung wird direkt auf die in der JRE implementierten Schnittstellen zugegriffen, um an die gewünschten Informationen/Metriken zu gelangen. Durch die Performance-Optimierung der JRE bekommt man durch die API einen geringeren Detaillierungsgrad der Abläufe als mit dem BCI-Ansatz. Eine andere Methode ist das Erzeugen von Thread-Dumps. Der Thread-Dump ist ein Momentanbild des aktuellen Java-Threads, dieses Abbild kann zum Beispiel in eine Datei geschrieben werden. Die Dumps enthalten alle Informationen des aktuellen Aufrufs, diese Informationen stellen eine aktuelle Momentaufnahme dar. Vorteile: Überwachung aller Komponenten, geringer Performance-Impact, kein Neustart des Java-EE-Servers. Nachteil: Momentaufnahmen ermöglichen keine durchgängige Verfolgung von Transaktionen/Programmabläufen.

Die Java-Diagnose im EM13c bedient sich unter anderem der Thread-Dump-Methode, da die Architektur des EM13c auf die hochskalierbare Überwachung von Produktionsumgebungen ausgelegt ist.

Java-Diagnose mit dem Oracle Workload-Explorer im EM13c

Der Workload-Explorer ist eine neue Darstellung der ermittelten Statistiken der Java-Diagnose. Das Diagnosebild wird mit regelmäßigen Thread-Dumps erstellt. Diese werden per Default alle zwei Sekunden durchgeführt und von dem EM13c-Agenten an das zentrale Repository übermittelt. Mit den gewonnenen Informationen werden folgende Fehlerursachen identifiziert:

1. Performance-Bottlenecks im Java Code
2. Sich gegenseitig behindernde Threads (Stuck-Threads)
3. Analyse von Schnittstellenproblemen (JDBC, JMS, RMI etc.)

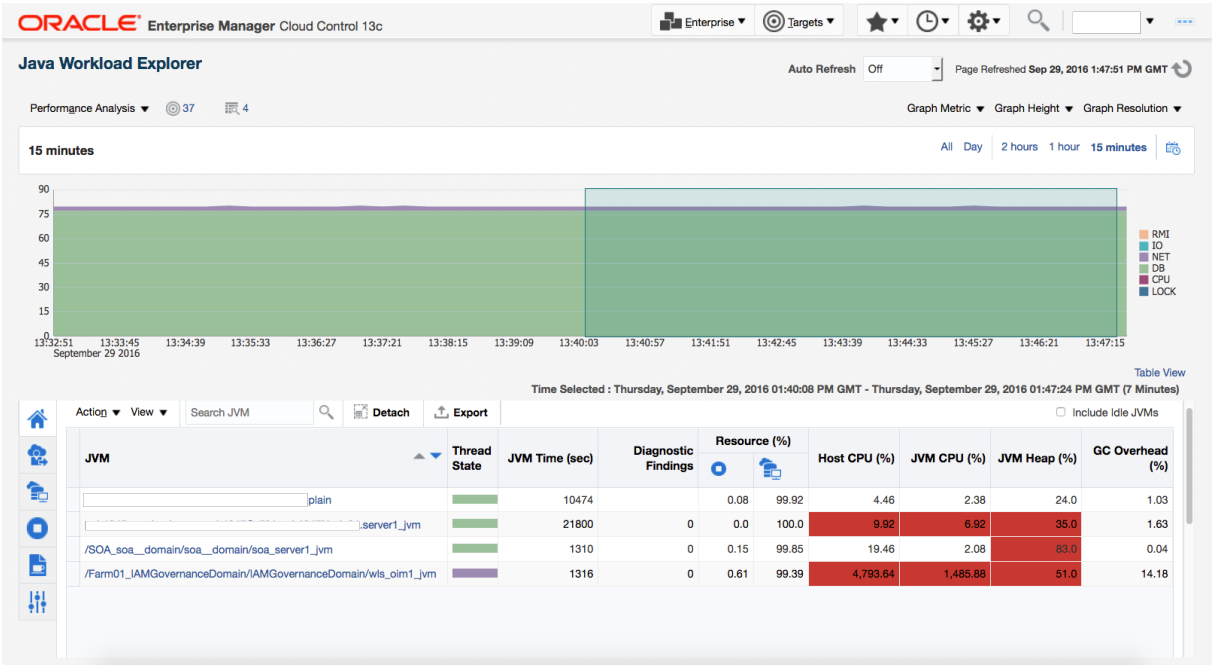

Abb. 1: EM13c Java Workload Explorer Übersicht der JVMs

Der Workload-Explorer liefert alle Details zu Performance-Statistiken und liefert Informationen zu:

- JVM-Statistiken
- OS-Informationen
- Garbage-Collection
- Server-Requests
- Client-Requests (RMI, Web)
- Statistiken zu Methoden, Klassen und Packages
- Statistiken zu Externen-Ressourcen-Anfragen wie Datenbank, Dateien etc.

Diese Statistiken können vom Server-Request bis zum Datenbank-Call verfolgt werden. So ist es zum Beispiel möglich, das SQL-Statement auszulesen und ohne Medienunterbrechung direkt in der Datenbank-Diagnose weiter zu analysieren, dies ist auch in umgekehrter Reihenfolge möglich. Langlaufende SQL-Statements werden von DBAs aufgespürt, und die dafür verantwortliche Java-

Methode mittels EM13c Workload-Explorer zugewiesen. So hat der Java-Entwickler genügend Informationen, um die Ursache zu ermitteln.

Um ein Bild der allgemeinen Informationen zu erhalten, wurde eine Java-Workload-Reporting-Funktion implementiert. In dem Report kann der Benutzer über einen maximalen Zeitraum von einer Stunde und über die Anzahl von 10 JVMs einen Report erstellen. In den gewonnenen Reports werden alle Statistiken im Kontext und übersichtlich dargestellt. Der Report wird im HTML-Format bereitgestellt und kann an jedem beliebigen Ort gespeichert werden.

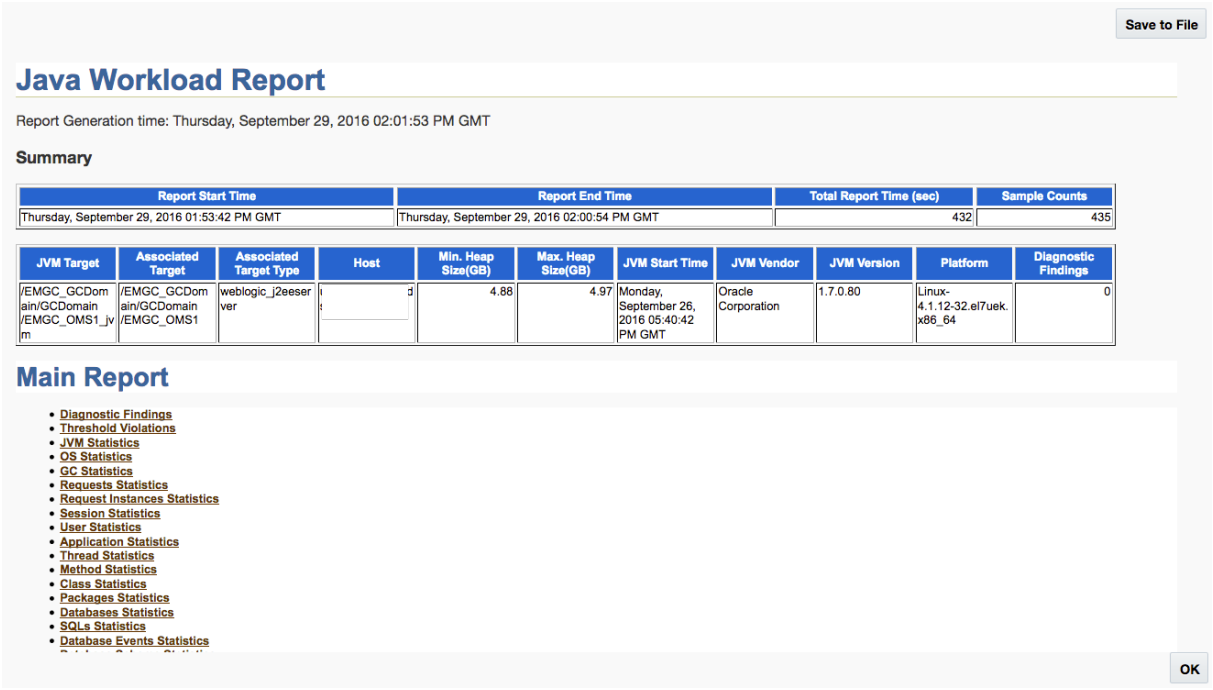

Abb. 3: EM13c Darstellung des Java Workload Reports

Java Heap Memory Analyse mit dem Oracle EM13c

Ein weiterer wichtiger Punkt im Kontext der Java-Applikationsanalyse stellt die Überwachung und Auswertung des Java-Speichers dar. Um eine höhere Transparenz der Nutzung des Java-Heap-Speichers zu erhalten, existieren verschiedene Ansätze. Eine Möglichkeit ist das Auslesen des Speichers mittels Profiler-Interface (JVMPI) bzw. Tool-Interface (JVMTI). Man gewinnt aktuelle Informationen über Speichernutzung und die Gruppierung auf die verschiedenen geladenen Objekte. Dies ist ausreichend, um gewisse Tendenzen bezüglich Speicherzuwächsen zu erkennen.

Im Oracle EM13c werden alle Informationen übersichtlich abgebildet. Im Bereich Garbage Collection werden die Bereinigungsintervalle dargestellt, Class Loading Data gibt an, welche und wie viele Klassen in den Speicher geladen wurden, und im Class Histogramm werden nach einem festen Schema Speicherabbilder erzeugt. Der letzte Punkt dient dem Auffinden von Java-Heap-Speicher-Lecks. Der Benutzer definiert einen Zeitraum, in dem wiederholt Speicherabbilder erzeugt werden. Diese werden im EM13c Repository abgespeichert und können zu einem späteren Zeitpunkt analysiert werden. Durch dieses Vorgehen können Java-Heapspeicher-Lecks gefunden werden, die nach einer langen Laufzeit auftreten.

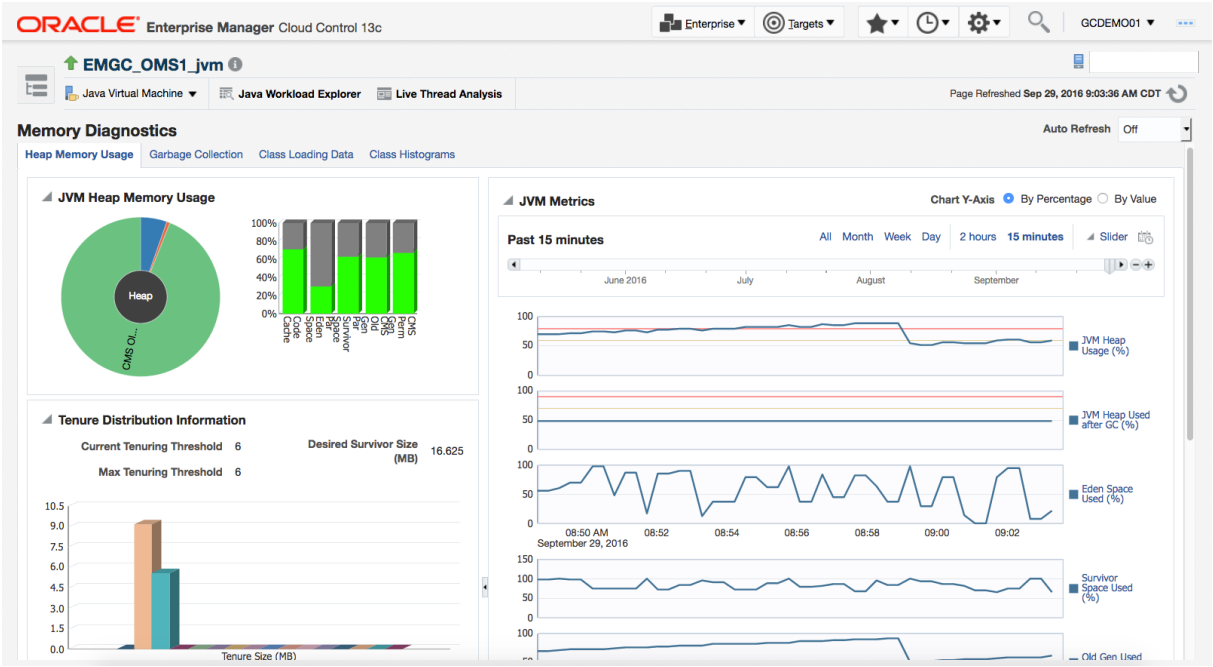

Abb. 2: EM13c Memory Diagnose Statistiken

Wurde ermittelt, dass es sich um ein Speicherproblem handelt, wird im nächsten Schritt die Ursache identifiziert. Dafür reichen die bisher aufgezählten Ansätze nicht aus! Um einen größeren Detailierungsgrad zu erhalten, muss ein vollständiges Speicherabbild erzeugt werden. Das Abbild eines Java-Speichers kann sehr groß werden. Bei Systemen, die Speichergrößen im zweistelligen Gigabyte-Bereich aufweisen, muss das Erzeugen von Heap-Dumps sorgfältig geplant werden. Es sollte ausreichend Speicherplatz auf den lokalen Festplatten vorhanden sein, um zum Beispiel mehrere Dumps speichern zu können. Um ein genaues Bild der Ursache zu erhalten, reicht es nicht aus, ein oder zwei Heap-Dumps zu erzeugen, in der Praxis sind es oft zehn und mehr.

Im EM13c werden die erzeugten Heap-Dumps in ein lokales Verzeichnis oder ein Shared-File-System geschrieben. Diese mehrere Gigabyte großen Dateien werden entweder sofort nach dem Schreiben oder zu einem späteren Zeitpunkt in das EM13c Repository geladen. Die für den Nutzer variable Upload-Zeit wurde implementiert, um ggf. auftretende Engpässe im Netzwerk nicht zusätzlich zu erhöhen. Das Hochladen der Dateien kann zum Beispiel nachts erfolgen. Das EM13c Repository muss für diese Fälle ausreichend dimensioniert sein. Alternativ kann eine externe Oracle-Datenbank für das Speichern der Heap-Dumps verwendet werden. Alles unter der Voraussetzung, dass die Analyse im Enterprise Manager Cloud Control stattfindet. Werden externe Tools verwendet, kann der Heap-Dump direkt im HPROF-Format abgespeichert werden und mit den entsprechenden Tools analysiert werden. Der EM13c bietet alle Möglichkeiten, die ein modernes Heap-Analyse-Tool bereitstellen muss: Automatische Analysen, Leak-Kandidaten etc. Nach erfolgreicher Ursachenanalyse können die geladenen Heap-Dumps aus dem Repository bzw. aus der externen Datenbank gelöscht werden.

Fazit

Der EM13c bietet neben seinen hervorragenden Eigenschaften in den Bereichen Überwachung, Benachrichtigungen, Service Level Management, Reporting und Lifecycle-Management auch im Diagnosebereich eine ausgereifte Lösung, um Probleme im Java-Applikations-Bereich zu lösen. Im Betriebsumfeld, in welchem es meist zu viele verschiedene Werkzeuge gibt, sollte möglichst auf integrierte und zentrale Lösungen zurückgegriffen werden.

Kontaktadresse:
Marcus Schröder
Oracle Deutschland B.V. & Co KG
Lina-Ammon-Str. 19
90471 Nürnberg

Telefon/Fax: +49 (0)911 98182471
E-Mail: marcus.schroeder@oracle.com
Internet: www.oracle.com