Konzeption und Entwicklung eines Auswertungstools für die Logdateien des JRockit Garbage Collectors

Student: *Raffael Schmid*
Studiengang: *Informatik*
Zeitraum: *Juli 2011 bis Dezember 2011*

Lagerstrasse 45, Postfach 1568, CH-8021 Zürich
Telefon +41 (0)43 268 25 22, Fax +41 (0)43 268 25 30
rektorat@hsz-t.ch, www.hsz-t.ch

HOCHSCHULE FÜR TECHNIK ZÜRICH

Aufgabenstellung

Ausgangslage

Für die Ermittlung von Java Performance-Problemen braucht es Wissen über die Funktionsweise der Java Virtual Machine (JVM), deren Ressourcenverwaltung (Speicher, I/O, CPU) und das Betriebssystem. Die Verwendung von Tools zur automatisierten Auswertung der Daten kann in den meisten Fällen sehr hilfreich sein.
Die Auswertung von Garbage Collection Metriken kann im laufenden Betrieb durch Profiling (online) gemacht werden, sie ist aber bei allen JVMs auch via Logdatei (offline) möglich. Die unterschiedlichen Charakteristiken der Garbage Collectors bedingen auch unterschiedliche Auswertungs- und Einstellungsparameter.
JRockit ist die Virtual Machine des Weblogic Application Servers und basiert entsprechend auch auf anderen Garbage Collection Algorithmen als die der Sun VM. Aktuell gibt es noch kein Tool, welches die Daten der Logs sammelt und grafisch darstellt.

Ziel der Arbeit

Ziel der Bachelorthesis ist die Konzeption und Entwicklung eines Prototypen für die Analyse von Garbage Collection Logdateien der JRockit Virtual Machine. Die Software wird mittels einer Java Rich Client Technologie implementiert. Zur Konzeption werden die theoretischen Grundlagen der Garbage Collection im Allgemeinen und der JRockit Virtual Machine spezifisch erarbeitet und zusammengestellt.

Aufgabenstellung

Im Rahmen der Bachelorthesis werden vom Studenten folgende Aufgaben durchgeführt:

1. Studie der Theoretischen Grundlage im Bereich der Garbage Collection (generell und spezifisch JRockit Virtual Machine)
2. Stärken- / Schwächen-Analyse der bestehende Rich Client Frameworks (Eclipse RCP Version 3/4, Netbeans)
3. Durchführung einer Anforderungsanalyse für einen Software-Prototyp.
4. Auswahl der zu verwendenden Frameworks
5. Konzeption und Spezifikation des Software-Prototypen (auf Basis des ausgewählten Rich Client Frameworks), der die ermittelten Anforderungen erfüllt.
6. Implementation der Software
7. Bewertung der Software auf Basis der Anforderungen

Erwartete Resultate

Die erwarteten Resultate dieser Bachelorthesis sind:

1. Detaillierte Beschreibung der Garbage Collection Algorithmen der Java Virtual Machine im Generellen und spezifisch der JRockit Virtual Machine.
2. Analyse über Stärken und Schwächen der bestehenden (state of the art) Java Rich Client Technologien
3. Anforderungsanalyse des Software Prototyps
4. Dokumentierte Auswahlkriterien und Entscheidungsgrundlagen
5. Konzept und Spezifikation der Software
6. Lauffähige, installierbare Software und Source-Code
7. Dokumentierte Bewertung der Implementation

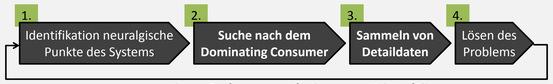
Weiter Informationen

Link zur Arbeit:	https://github.com/schmidic/bachelorthesis
Email:	schmira4@students.zhaw.ch
Zeitraum:	22. Juni 2011 bis 22. Dezember 2011

Grundlagen

Vorgehen Performance Tuning

Die Performanceanalyse ist ein iterativer Prozess und dauert in der Regel so lange, bis die Anforderungen an das System in diesem Bereich erfüllt sind. Eine einzelne Iteration besteht aus den folgenden vier Schritten:



solange Performanceanforderungen nicht erfüllt

Suche nach dem Dominating Consumer

Hohe relative Systemlast

Die erste Frage bei der Suche nach dem Dominating Consumer ist, ob die hohe CPU-Auslastung durch die Applikation oder das System verursacht wird. Bei einer Auslastung durch das System spricht man von einer hohen relativen Systemlast. Dies kann unterschiedliche Gründe haben (exzessive Kontextwechsel, hohes I/O). Zur Analyse dieses Messwertes kann dafür der Task Manager (unter Windows) oder vmstat (unter Linux, Unix) verwendet werden.

Hohe CPU- respektive Core-Last

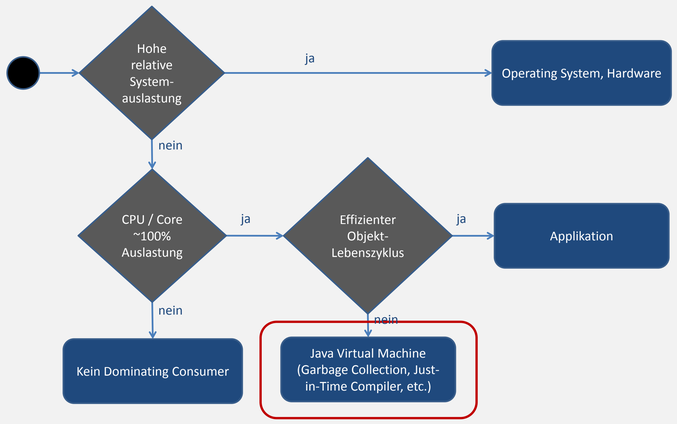
Sofern die relative Systemlast nicht gross ist wird überprüft, ob die CPU Auslastung insgesamt gross ist. Man prüft also, ob es eventuell gar **keinen Dominating Consumer** gibt. Dies würde bedeuten, dass etwas die Threads daran hindert, CPU-Ressourcen zu bekommen. *Dies kann unterschiedliche Gründe haben: Dead Locks, Applikation skaliert nicht, langsame Disks oder langsames Netzwerke, zu kleine Connection- und Thread-Pools, Aurufe auf langsame externe Systeme.*

Effizienter Objekt-Lebenszyklus

Sofern die CPU-Auslastung trotz kleiner relativer Systemlast durch die Applikation hoch ist, muss **Java Virtual Machine** und evt. die **Garbage Collection** angeschaut werden. Dafür gibt es unterschiedliche Herangehensweisen:

•*Memory Analyse (Objektpopulation):* Es wird angeschaut, wie alt die Objekte in den unterschiedlichen Bereichen sind.

•*Garbage Collection Heuristik:* Wann und wie oft werden Garbage Collections durchgeführt, wie lange haben sie gedauert, wieviel Speicher haben sie befreit.



Garbage Collection Tuning

Ziele

Beim Tuning der Garbage Collection verfolgt man eines der drei folgenden Ziele:
• Verbesserung des Durchsatzes
• **kleine und gleichmässige Pausenzeiten**
• geringerer Speicherverbrauch

Das einzige relevante Tuningziel ist in der Regel die Optimierung hinsichtlich gleichmässiger und kurzer Pausenzeiten. Dies ist insbesondere bei Applikationen wichtig, bei denen die Interaktion mit einem Benutzer im Vordergrund steht. Durchsatztuning ist meistens nicht sehr effektiv und der Austausch der CPU kostengünstiger. Arbeitsspeicher-Tuning (damit der Garbage Collector weniger Arbeitsspeicher verbraucht) ist mit der 64-Bit Architektur in den Hintergrund gerückt, da nun normalerweise genügend Speicher angesprochen werden kann.

Garbage Collection auf der JRockit Virtual Machine

Die Grundlage der JRockit Garbage Collection bildet der Tri-Coloring Mark & Sweep Algorithmus. Er wurde hinsichtlich besserer Parallelisierbarkeit und der optimalen Verwendung der Anzahl Garbage Collection Threads optimiert. Die Garbage Collection der JRockit VM arbeitet entweder mit oder ohne Generationen. Es gibt folgende Algorithmen:

Generational Concurrent Mark & Sweep
• Single Concurrent Mark & Sweep
• Generational Parallel Mark & Sweep
• Single Parallel Mark & Sweep

Aufbau des Heaps

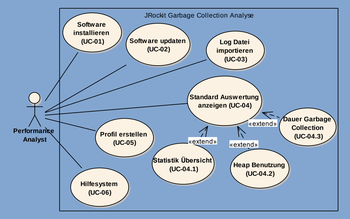
Die JRockit Virtual Machine kommt ohne Permanent-Bereich (hier werden bei der HotSpot VM beispielsweise die Class-Objekte angelegt) aus. JRockit verwendet keine Mark & Copy Algorithmus und benötigt deshalb auch keine Survivor-Regionen. Sofern Generationen angelegt werden, sieht das Schema der Bereiche folgendermassen aus:

In der Young Generation werden die neuen Objekte angelegt, sie wird in thread-lokale Bereiche (jedem Thread gehört ein Bereich) unterteilt, sodass die Allokation nicht synchronisiert werden muss. In der Keep Area befinden sich die Objekte, welche eine Garbage Collection auf der Young Generation überlebt haben. Nach einer nochmaligen Collection wandern sie in die Old Generation und bleiben da, bis sie von einer Old Collection weggeräumt werden.

Realisierung

Anforderungsanalyse

Use Cases



Bezeichnung	Titel	Beschreibung
UC-01	Software installieren	Der Benutzer kann die Software in seiner Entwicklungsumgebung installieren.
UC-02	Software update	Der Benutzer kann die Software aus der Entwicklungsumgebung updaten
UC-03	Logdatei importieren	Der Benutzer kann die sich auf dem Dateisystem befindenden Logdatei importieren.
UC-04	Standardauswertung	Für eine schnelle Übersicht kann der Benutzer die Standardauswertung öffnen.
UC-04.1	Anzeige Statistik Übersicht	Auf dem ersten Tab der Standardauswertung befinden sich verschiedene statistische Auswertungen .
UC-04.2	Anzeige Heap Benutzung	Die Heap Benutzung zeigt dem Benutzer anhand einer Grafik, zu welchem Zeitpunkt wie viel Speicher des Heaps verwendet wurde.
UC-04.3	Anzeige Dauer Garbage Collection	Die Anzeige Dauer Garbage Collection zeigt dem Benutzer über die Zeit wie lange die einzelne Garbage Collection gedauert hat.
UC-05	Profil (benutzerdefinierte Auswertung) erstellen	Der Benutzer kann ein eigenes Profil erstellen. Dem Profil können eigene, benutzerdefinierte Charts hinzugefügt werden. Die Profile sind persistent und können exportiert wie auch importiert werden.
UC-06	Hilfesystem	Der Benutzer kann auf eine indexbasierte und eine kontextsensitive Hilfe zugreifen

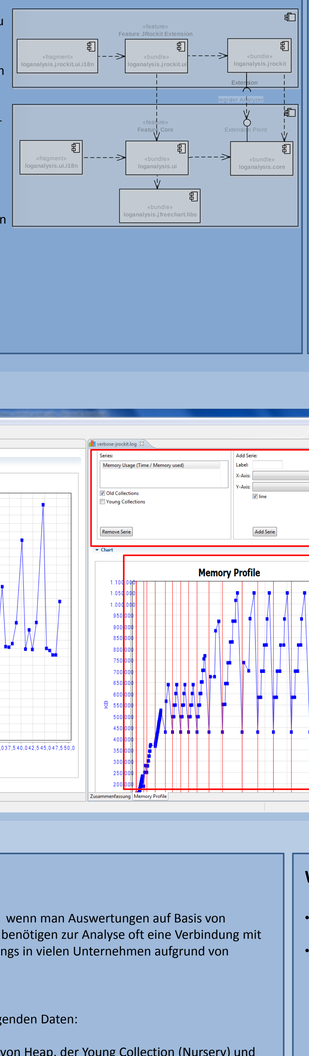
Konzept

Verwendete Frameworks

Auf der Basis der Anforderungen wurde eine Evaluation bestimmter Komponenten gemacht. Aufgrund dieser ergab sich, welche Bibliotheken für die Realisierung verwendet werden: als Rich Client Framework wird Eclipse 3.x verwendet, zur Generierung der Charts hat sich die Verwendung von JFreeChart als leichtgewichtig und mächtig erwiesen.

Architektur

Aufgrund einer Anforderung ist die Applikation so zu konzipieren, dass auch weitere Logformate hinzugefügt werden können. Die Applikation wird deshalb initial (später können andere Erweiterungen hinzu kommen) in die Teile *Basissoftware* und *Erweiterung JRockit* aufgeteilt. Die Kommunikation dazwischen findet über den Eclipse Extension-Point-Mechanismus statt. Die Verantwortlichkeiten sind folgendermassen:



Basissoftware (Core Feature)
• Garbage Collection Log importieren, einlesen
• Profil erstellen, speichern, exportieren, importieren
• Hilfesystem (Features registrieren Erweiterungen)

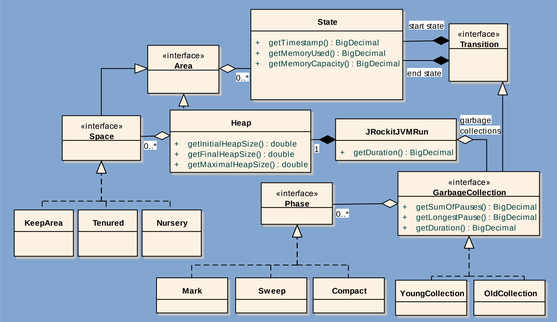
JRockit Extension (JRockit Extension Feature)
• Garbage Collection Log parsen, in eigenem Domänenmodell speichern
•Laden der Daten aus dem Domänenmodell

Parser

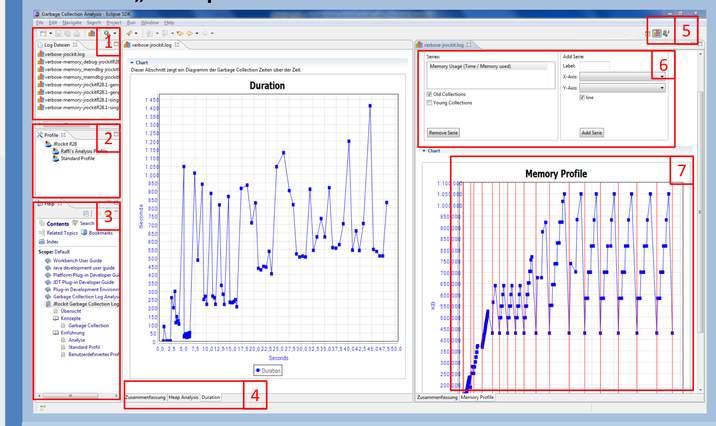
Das Parsen passiert inzwei Schritten: als erstes wandelt der Lexer die Rohdaten in einzelne Tokens um, anschliessend speichert der Syntactic Analyzer die werte im Domänenmodell.

Domänenmodell

Die gelesenen Daten werden anschliessend in Form dieses Domänenmodells gespeichert. Es entspricht der abstrahierten Struktur des JRockit Garbage Collectors.



Proof of „Konzept“



Implementation

Eine Logdatei wird durch den Import in der Ansicht *Logdateien* sichtbar. Mit einem Doppelklick auf diese Datei oder ein Profil wird die Logdatei eingelesen, geparst, analysiert und im entsprechenden Analysefenster geöffnet.

1. Die Ansicht *Logdateien* zeigt alle vom Benutzer importierten Garbage Collection Logs.
2. Die Ansicht *Profile* zeigt die vom Benutzer erstellten Profile, diese können an die Bedürfnisse des Analysten angepasst werden.
3. Die Hilfe zeigt relevante Theman basierend auf einem index. Es gibt auch Hilfethemen die an eine bestimmte Aktion oder ein Fenster gebunden sind (kontextsensitiv).
4. Die geöffnete Standardauswertung zeigt drei verschiedene Tabs mit der Zusammenfassung der Logdatei, der Heap Analyse und des Diagramms Dauer Garbage Collection.
5. Für die Garbage Collection Analyse wurde eine eigene Perspektive definiert, damit der Benutzer andere Perspektiven nicht umstellen muss.
6. Hier kann der Benutzer das in diesem Fenster aktuell sichtbare Chart anpassen.
7. Ein benutzerdefiniertes Chart, kann durch den Benutzer angepasst werden und zeigt verschiedene Datenserien an.

Review

Was das Tool leistet

Die Analysesoftware kann dann verwendet werden, wenn man Auswertungen auf Basis von Garbage Collection Logs machen will. Andere Tools benötigen zur Analyse oft eine Verbindung mit der entsprechenden Virtual Machine, dies ist allerdings in vielen Unternehmen aufgrund von blockierten Ports nicht möglich.

Funktionsumfang

Die Standardauswertung zeigt eine Statistik mit folgenden Daten:
• Garbage Collection Aktivität
• Initiale, durchschnittliche und maximale Kapazität von Heap, der Young Collection (Nursery) und Old Collection.
• Generelle Informationen wie Dauer der Messung, verwendete Zeit in der Garbage Collection (relativ, absolut)
Zusätzlich gibt es Diagramme zur Auswertung des verwendeten Speichers auf dem Heap und zur Dauer der einzelnen Garbage Collections.

Was das Tool nicht leistet

• Die Software kann zum jetzigen Zeitpunkt keine anderen Logdateien, als die der JRockit Virtual Machine Release 28, verarbeiten.
• Der Einsatz der Analysesoftware für das Tuning der Garbage Collection auf der JRockit Virtual Machine macht aktuell nur dann Sinn, wenn die Auswertung mittels JRockit Mission Control (ein Werkzeug von Oracle) nicht möglich ist (Firewall-Restriktionen, etc.) .