FT-Ausarbeitung

Java Virtual Machine

Erläuterungen der Funktionsweisen

Alexander Rieppel

28. Mai 2014

5AHITT

Inhaltsverzeichnis

1 Einführung

Die Java Virtual Machine (JVM) wird innerhalb der Java Laufzeitumgebung (JRE) angeboten, welche für die Ausführung des Byte-Codes von Java-Programmen verantwortlich ist. Im Normalfall wird jedes gestartete Programm in seiner eigenen virtuellen Umgebung ausgeführt. Den restlichen Teil der JRE stellen die Java-Klassenbibliotheken dar. Die JVM dient als Schnittstelle zwischen Maschine und Betriebssystem und ist sowohl für Microsoft Windows, Linux, Mac OS X und andere Plattformen verfügbar.

Die JVM besteht aus folgenden Teilen:

- Classloader und Resolver
- Garbage Collector
- Execution Engine

1.1 Funktionsweise

Die Java Virtual Machine ist dazu da, um den Byte-Code der vom Java-Compiler erzeugt wurde, auf der jeweiligen Plattform auf der er ausgeführt werden soll ausführbar zu machen. Dazu werden die verschiedenen Byte-Code Dateien die erzeugt wurden (zu erkennen an Dateieindung “class“) während der Laufzeit in die, vom jeweiligen Rechner verstandene Maschinensprache übersetzt. Die JVM arbeitet im Prinzip wie ein Interpreter, allerdings um einiges schneller, da nicht mehr auf Syntaxfehler geachtet werden muss.

1.2 Sicherheit

Die JVM ist neben der Plattformunabhängigkeit auch für ein gewisses Maß an Sicherheit verantwortlich. Sie über wacht während der Lauf-

zeit die Ausführung der Programme, verhindert Pufferüberläufe, welche zu unvorhersehbarem Verhalten wie zum Beispiel ein Absturz des Programmes führen können. Diese Überwachung ist allerdings ziemlich einfach, da Java Zeiger nur implizit unterstützt.

1.3 Optimierungsverfahren

Um die Performance von Java-Programmen zu erhöhen, setzen die meisten Java-VMs sogenannte JIT-Compiler (Just-In-Time-Compiler oder JITC) ein, die unmittelbar während des Programmablaufs den Bytecode "genau zur rechten Zeit" dauerhaft in Maschinensprache übersetzen. Eine Weiterentwicklung ist der Hotspot-Optimizer, welcher dynamische Optimierung anbietet.

1.3.1 Dynamische Optimierung

Oft ist zum Zeitpunkt der Kompilierung nicht bekannt, welche konkreten Eingaben eine Software verarbeiten muss. Demnach muss die Software mit allen Arten von Eingaben zurechtkommen. Die Eingabe wird deshalb in Variablen gespeichert. Nach dem Start des Programms werden jedoch viele Variablen nicht mehr geändert. Folglich sind diese - von einem Zeitpunkt kurz nach dem Start an - Konstanten. Wird nun erst nach diesem Zeitpunkt die Software für die System-Architektur kompiliert (dies ist bei Java Hotspot der Fall), so können diese Konstanten berücksichtigt werden. Bestimmte Verzweigungen, die nur von solchen „Halbkonstanten" abhängig sind, sind dann für immer eindeutig und stellen somit kein Risiko für eine falsche Sprungvorhersage dar. Ein solcher Programmcode kann also schneller ablaufen als zu früh compilierter Code.

1.3.2 Implementierung in Hardware

Ausführungen in Hardware sind Java-Prozessoren, Mikroprozessoren, die Java-Bytecode als Maschinensprache verwenden. Sie konnten sich gegen die schnelle Steigerung der Leistungsfähigkeit von Standard-PC und JVM nicht durchsetzen.

1.4 Threads

Die Java VM schottet die in ihr laufenden Prozesse vom Betriebssystem ab (Green Threads). Sie bildet standardmäßig Java-Threads durch Threads des Betriebssystems ab. Nur in Ausnahmefällen erfolgt das Thread-Management durch die Java VM. Somit ist es auch möglich, auf einem Betriebssystem, das kein Multithreading unterstützt, eine Java VM mit vollem Funktionsumfang anzubieten. Die Java VM hat stets volle und standardkonforme Kontrolle über die Java-Threads, d.h. der Programmierer braucht nicht die betriebssystemspezifischen Multi-Threading/Tasking/Processing-Eigenschaften zu berücksichtigen und kann sich stets auf die JRE verlassen. Nachteil ist, dass Probleme, die von einem Thread ausgehen, seitens des Betriebssystems dem gesamten Prozess zugeordnet werden. Gängige Betriebssysteme (wie zum Beispiel Linux, Windows) erlauben Kontrolle über diese „nativen“ Threads allenfalls über Software von Drittanbietern, wie beispielsweise die mit dem JDK mitgelieferte VisualVM. Standardwerkzeuge wie zum Beispiel der Windows Taskmanager zeigen Systemthreads jedoch nicht an.

1.5 JVM kompatible Sprachen

Neben Java gibt es auch andere Sprachen, die als Programmiersprachen für JVM-Programme benutzt werden können. Unter anderem folgende Sprachen können auf einer JVM laufen:

- Clojure, ein Lisp-Dialekt
- Ceylon
- Erjang, ein Erlang-Dialekt für die JVM
- Free Pascal, das auch unter der JVM einen Großteil der Object-Pascal-Konstrukte unterstützt
- Groovy, eine dynamisch typisierte Programmiersprache
- JRuby, eine 100
- Jython (früher: JPython), ist eine reine Java-Implementierung der Programmiersprache Python

- Scala, eine Sprache, die Eigenschaften von Java mit funktionaler Programmierung vereint
- Kotlin, eine 2011 vorgestellte Sprache von JetBrains

Auch gibt es eine Reihe von Skript-Sprachen, die von Java aus aufrufbar sind. Dazu gehört JavaScript (mittlerweile standardisiert als ECMAScript) mit dem „Rhino“-Interpreter (ein Mozilla-Projekt) sowie JavaFX als eine Skript-Sprache vor allem für grafische Elemente.[Wik][Ora]

Literaturverzeichnis

[Ora] Oracle: *Java Virtual Machine* - *http://docs.oracle.com/javase/7/docs/technotes/guides/vm/index.html?intcmp=3170*

[Wik] Wikipedia: *Java Virtual Machine* - *http://de.wikipedia.org/wiki/Java_Virtual_Machine*