Java Unterstützung für Multithreading von den Versionen 1.0 bis 7

Wolfgang Nast
MT AG
Ratingen

Schlüsselworte

Java Multithreading Corncurent

Einleitung

Java hat seit der Version 1.0 Unterstützung für Multithreading. Die Unterstützung der konkurrierenden Ausführung wurde von Version zu Version verbessert. Durch Prozessoren mit mehreren Kernen und dem Hyperthreading ist die Nebenläufigkeit der Systeme nicht mehr nur für die Server von Bedeutung. Es werden die Ideen und Umsetzungen vorgestellt, die in Java realisiert und erweitert wurden über die Versionen.

Version 1.0 Zentrale Grundlagen

Die Thread-Klasse ist die zentrale Klasse. Jeder Thread in Java wird in einer Thread Instanz gehalten, auch der Hauptthread, der beim Start der Main-Methode verwendet wird. Mit dem Befehl `Thread.currentThread()` kann man sich den aktuellen Thread holen. Jeder Thread hat einen Status dieser beginnt mit dem Status `New` wenn die Instanz noch nicht gestartet wurde. Der Status `TERMINATED` zeigt, dass der Thread beendet wurde, nach dem er gelaufen war. Die Threads werden mit der Methode `start()` gestartet. Hier ist der Status dann `RUNNABLE` und kann sich während der Ausführung in folgende verändern: `WAITING, TIMED_WAITING, BLOCKED`. Ein Thread wird durch das beenden der run Methode beendet. Ein Thread, der als Daemon gekennzeichnet wurde braucht nicht beendet werden, damit das Java-Programm beendet wird. Wenn alle normalen Threads beendet wurden, wird auch das Programm beendet. Hier muss die Main-Methode nicht die letzte sein.

Ein Thread kann mit einem andern Thread interagieren. Mit der Methode `join()` kann man auf das Beenden des Threads warten. Mit der Methode `interrupt()` kann man dem anderen Thread signalisieren, dass dieser sich beenden soll. Dabei wird dieser aus Methoden, die warten oder blocken, mit einer InterrutedException geweckt. Ein laufender Thread kann mit `Thread.interrupted()` testen, ob er sich beenden soll. Mit `Thread.yield()` gibt der aktuell ausgeführte Thread seine Ausführung an einen anderen „laufenden“ Thread ab. Das Heißt, sind mehr lauffähige Threads vorhanden als freie Kerne, so muss die Ausführung aller Threads warten. Um hier zwischen den Threads in der Ausführung zu wechseln gibt es `yield()`. Mit Thread.sleep() wartet der aktuelle Thread die angegebene Zeit oder wird durch einen Interrupt geweckt.

In Java 1.0 gibt es noch die Methoden zum Steuern von anderen Threads, die nicht mehr verwendet werden sollen, weil sie dem kooperativen Gedanken wiedersprechen und auch besonders bei der Freigabe von Systemressourcen Probleme machen. Es sind ~~`suspend()`~~, ~~`resume()`~~, ~~`stop()`~~ und ~~`destroy()`~~.

Mit dem Schlüsselwort `synchronized` kann man einen Block oder einem Methode synchronisieren. Das heißt, es kann nur ein Thread in diesem Block ausgeführt werden. Alle anderen

Threads müssen vor dem Block warten, bis der Block vom Thread wieder freigegeben wird. Bei einem Block ist das Objekt als Parameter anzugeben, gegen das synchronisiert wird. Bei einer Methode ist es `this` (die Instanz) und bei einer `static` Methode ist es die Klasseninstanz. In Java wird jedem Objekt ein Monitor zugeordnet. Über die Monitore wird die Synchronisation vorgehalten. Ein Monitor weiß, wie viele Threads warten. Nur in den `synchronized` Blöcken dürfen die folgenden Methoden des `Object` verwendet werden: `wait(), notify(), notifyAll()`. Mit `wait` wartet man auf ein `notify` eines anderen Threads. Mit `notifyAll` werden alle wartenden Threads aufgeweckt. Mit Java 5 wurden allgemeinere Klassen eingeführt.

Objekt mit dem Schlüsselwort `volatile` dürfen von unterschiedlichen Threads manipuliert werden und wurden häufig zum Datenaustausch zwischen Threads verwendet. Mit Java 5 wurden hier bessere Möglichkeiten umgesetzt.

Version 1.2

Mit Einführung der Collection Klassen wurden die Schwäche der `synchronized` Klassen `Vector` und `Hashtable` aufgehoben durch die Einführung vom `List` und `Map`. Es gibt keine Collections, die in mehreren Threads geändert und gelesen werden können. Mit dem Einfügen oder Löschen eines Elementes werden alle Itteratoren ungültig.

Version 1.4

Änderung am Speichermodell. Hat Auswirkungen auf Objekte die das Schlüsselwort `volatile` verwenden. Hier wird der Cache der einzelnen Threads in `synchronized` Blöcken anders behandelt. Und die Outoforderexecution von arithmetischen Funktionen wurde für volatile Objekte abgeschaltet.

Version 5

Concurrent Klassen wurden eingeführt. `ReentrantLock` ist die Implementierung eines Locks, der wie `synchonized` einen Programbereich schützt, damit nur ein Thread diesen ausführt. Die anderen müssen warten. Ein Lock kann auch als fairer Lock verwendet werden, dann ist die Ausführungsreihenfolge der wartenden Threads, wie die Reihenfolge wie sie warten und nicht willkürlich wie bei `synchronized`. Mit `lock()` oder `tryLock()` wird der Lock gesichert und mit `unlock()` wird der Lock wieder freigegeben. Zu einem Lock kann man mit der Methode `newCondition()` ein Condition Object holen, das mit der Methode `await()` genauso wie `wait()` des Objektes auf eine Signalisierung wartet. Mit der Methode `singal()` wird der wartende Thread aufgeweckt wie mit `notify()` des Objektes. Auch für `notifyAll()` gibt es die Entsprechung `signalAll()`, damit werden alle wartenden Conditions aufgeweckt. Eine weiterer Lock ist umgesetzt als ReentrantReadWriteLock bestehend aus zwei Locks einen zum Lesen und einen zum Schreiben, die voneinander abhängig sind. Als Verbesserung zu volatile sind die atomic Klassen eingeführt worden. Passend zu den einfache Typen boolean, int und long gibt es atomare Klassen, die direkte ändernde Zugriffe aus mehreren Threads zulassen. Für Arrays und Referenzen gibt es auch atomare Klassen. Mit den Queues wurden Klassen eingeführt, um Daten zwischen Threads auszutauschen.

Wichtig sind Future und Excecutor um von den realen Threads unabhänig zu werden.

Die Management-Objekte bieten auch Informationen für das eigene Programm nicht nur für den JMX-Client wie JConsole.

JConsole als Tool um ein laufendes Programm zu untersuchen. Hier können die Threads und ihre Threaddumps angesehen werden.

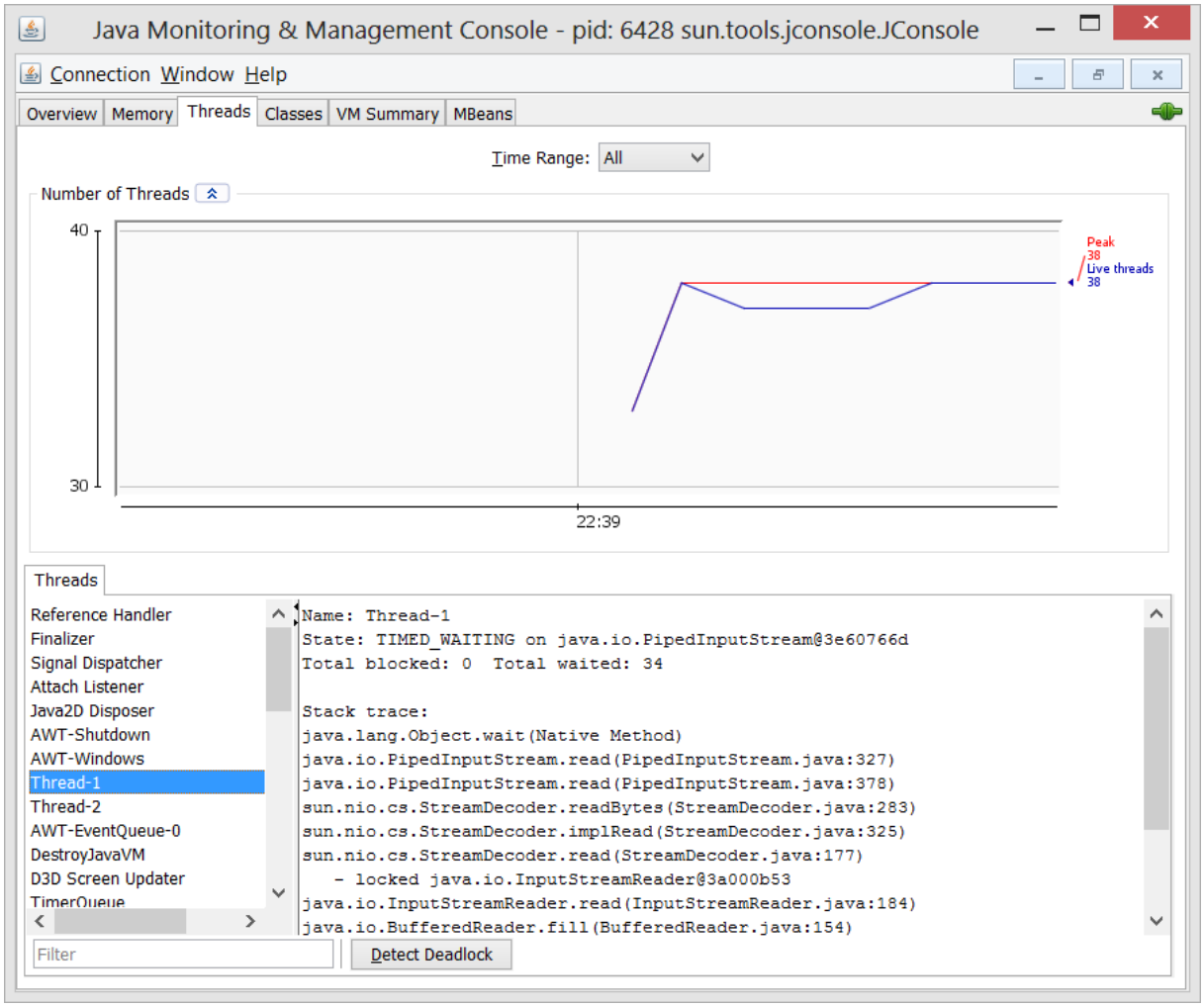

Abb. 1: JConsole

Version 6

Verbesserungen in den Collections. In den ManagementObjekten können jetzt auch Deadlocks mit Locks gefunden werden.

JVisualVM kam mit Update 10. Hier können die Threads und ihr Status über die Zeit betrachtet werden.

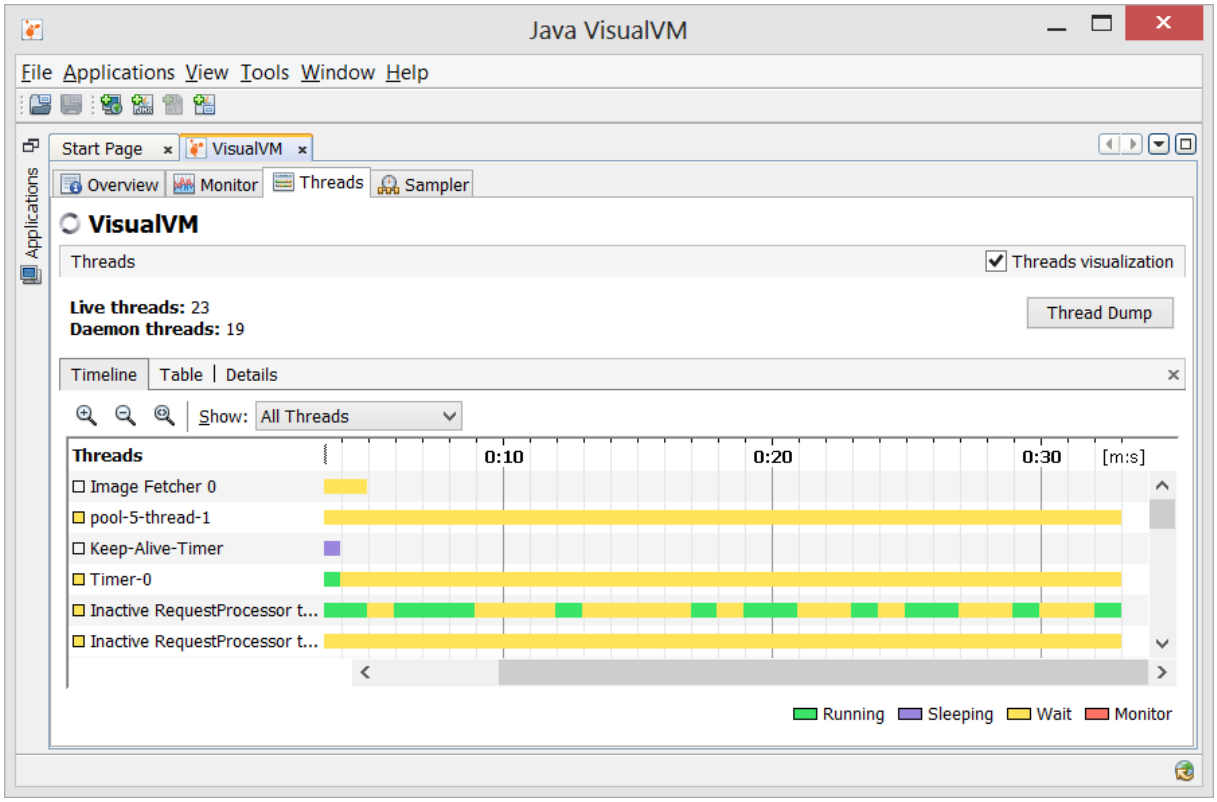

Abb. 2: JVisualVM

Version 7

Das Fork/Join-Framework eingeführt.

Neues Tool ist seit Update 40 JMC, welches die meisten Möglichkeiten bietet sich die Threads eines laufenden Java Programm anzusehen.

Die Threaddarstellungen der einzelnen IDEs werden in ihren aktuellen Versionen gezeigt.

Kontaktadresse:

Wolfgang Nast

MT AG

Balcke-Dürr-Allee 9
D-40882 Ratingen

Telefon: +49 (0) 2102 30961-0

Fax: +49 (0) 2102 30961- 101

E-Mail Wolfgang.Nast@mt-ag.com

Internet: www.mt-ag.com