ENTAGO

Reaktive Programmierung in Java (Reactive Programming)

Was ist reaktive Programmierung und welches Problem wird dadurch gelöst? Der vorliegende Artikel beschreibt die reaktive Programmierung und zeigt die Verwendung in Java. Die Funktionsweise der populären Frameworks für reaktive Programmierung in Java werden kurz vorgestellt. Abgeschlossen wird mit den Vor- und Nachteilen und wann sich reaktive Programmierung genau lohnt.

Um zu verstehen, welchen Vorteil reaktive Programmierung in Java bringt, muss zunächst die Problemstellung beschrieben werden. Der etablierte Weg der imperativen Programmierung bei Web-Anwendungen in Java verwendet eine Vorgehensweise, die sich «Thread-Per-Request» nennt. Dabei wird bei jedem Aufruf ein Thread verwendet. Dieser Thread wird benutzt, bis der Aufrufer eine Antwort vom Server erhält. Um genug Aufrufer bedienen zu können, hält der Server einen Thread-Pool bereit.

Abbildung 1 illustriert dieses Modell. Von links kommen Aufrufe zum Server. Im Thread-Pool ist festgelegt, wie viele Threads den Aufrufen zur Verfügung stehen. Danach findet der eigentliche Aufruf statt, hier im Beispiel ein Zugriff auf die Datenbank.

Skaliert werden kann durch Erhöhen der Threads im Thread-Pool. Bei Java übernimmt die JVM das Mapping von Java-Threads zu Betriebssystem-Threads.

Das Scheduling und Time-Slicing übernimmt dann das Betriebssystem.

Diese Herangehensweise hat sich über viele Jahre bewährt. Bei modernen Web-Anwendungen werden allerdings immer mehr Aufrufe pro Benutzer verwendet, da eine Applikation sich normalerweise aus mehreren Teilen zusammensetzt, die jeweils durch eigene Aufrufe geladen werden. Des Weiteren steigen Benutzerzahlen pro Web-Anwendung.

Bei dem Thread-Per-Request Modell werden dadurch immer mehr Threads verwendet.

Eine weitere Herausforderung ist, dass der Thread für die Dauer des Aufrufs nicht anders verwendet werden kann. Falls während der Verarbeitung wie in Abbildung 1 ein Datenbankaufruf stattfindet, der lange für die Verarbeitung braucht, bleibt während der gesamten Verarbeitung der Thread in Verwendung.

Hat die Datenbank sogar temporär sehr lange Zugriffszeiten, kann der Thread-Pool komplett aufgebraucht werden. Im schlimmsten Fall werden alle zur Verfügung stehenden Threads verbraucht. Damit steht die Web-Applikation still und kann nicht mehr aufgerufen werden.

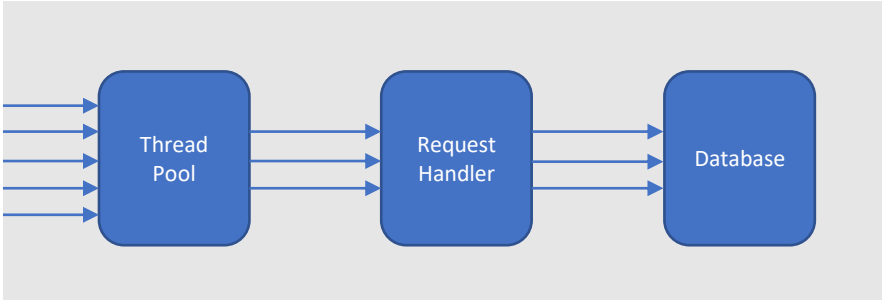

Abbildung 1: Thread-Per-Request Modell

ENTAGO

Definition

Reaktive Programmierung versucht die eingangs beschriebene Herausforderung zu lösen. Statt einem Thread-Pool werden bei reaktiven Frameworks normalerweise so viele Threads verwendet, wie CPU-Kerne zur Verfügung stehen. Deshalb darf keine Operation auf diesen Threads blockieren, wie es bei dem Thread-Per-Request Modell der Fall ist.

Um das zu erreichen, gibt jeder Aufruf im Programmcode sofort eine Antwort in Form eines Streams zurück. Ein Beispiel, welches das visualisiert: Aus einer Datenbank sollen Kunden geladen werden, die bestimmte Kriterien erfüllen:

```
SELECT * FROM KUNDE k
WHERE k.city = 'ZUERICH'
```

Bei imperativer Programmierung würde der Request an die Datenbank geschickt werden, die Datenbank sammelt die Kunden zusammen und meldet sich dann mit der Liste der Kunden. Für diese Zeit ist der Thread blockiert, er kann keine andere Aufgabe übernehmen. Bei reaktiver Programmierung wird der Request an die Datenbank gesendet, als Antwort kommt sofort ein Stream – es wird nicht blockiert. Wenn in der Datenbank der erste Kunde gefunden wird, welcher dem Kriterium entspricht, wird dieser als Element in den Stream geschickt.

Sobald alle Kundendaten auf den Stream geschickt wurden, wird als letztes Element ein «Completion Signal» auf den Stream geschickt. Dann weiss der aufrufende Programmcode, dass die Verarbeitung abgeschlossen ist und keine weiteren Daten mehr kommen.

Reaktive Programmierung ist programmieren mit asynchronen Streams in einer nicht-blockierenden Umgebung.

Reactive in Java

In Java bieten verschiedene Frameworks reaktive Programmierung, beispielsweise RxJava und Reactor. Die Frameworks haben gemein, dass sie die vier verschiedenen Typen implementieren, welche in einer Arbeitsgruppe namens „The Reactive Streams Initiative“ entstanden sind:

- `Subscriber`
- `Publisher`
- `Subscription`
- `Processor`

Erwähnenswert hierbei ist, dass das Interface `Subscription` eine Methode `request()` anbietet. Wenn sich der `Subscriber` mit dem `Publisher` verbindet, erhält er die `Subscription`. Danach kann der Subscriber durch `request()` steuern, wie viele Elemente er für die Verarbeitung erhält. Somit liegt die Verantwortung für die Menge der Elemente beim `Subscriber`. Dieses Vorgehen wird auch «**Backpressure**» genannt. Dadurch wird sichergestellt, dass ein `Publisher` einem `Subscriber` nicht mehr Elemente liefert, als er verarbeiten kann.

Abbildung 2 zeigt ein Beispiel für den Aufbau von Streams und den Elementen, die darauf verwendet werden. Der obere Stream sendet beispielsweise Zahlenwerte, die ersten drei Symbole zeigen die Werte 1, 2 und 5, die auf dem Stream als Elemente versandt werden. Das rote X zeigt einen Fehler. Bei RxJava und Reactor sind Errors (Exceptions) sogenannte «First-class citizens» und müssen entsprechend behandelt werden. Das letzte Symbol, der schwarze Balken, symbolisiert das «Completion Signal». Damit weiss der Aufrufer, dass alle Elemente gesendet wurden und der Stream somit abgeschlossen ist.

Der untere Teil der Abbildung zeigt eine Operation auf dem Stream. Auf dem Stream werden die Zahlenwerte 1, 3, 2 und 5 gesendet. Die Operation «filter» leitet nur die Werte weiter, die grösser als 2 sind. Als Rückgabe der Operation «filter» entsteht ein neuer Stream mit diesen Elementen.

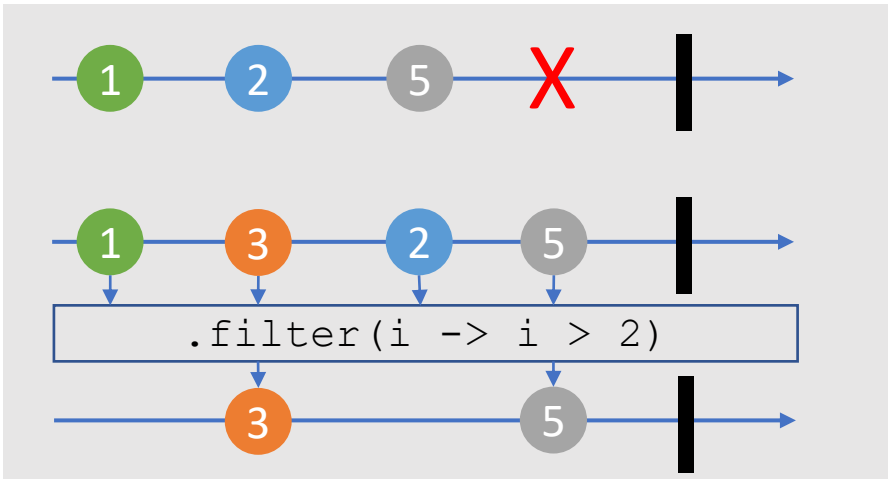

Abbildung 2: Beispiel Streams und Funktion

ENTAGO

Debugging und Fehleranalysen

Der Nachteil der imperativen Programmierung in Java wurde eingangs genannt – bei (sehr) vielen parallelen Aufrufen wird die Hardware nicht optimal genutzt, man spricht auch davon, dass diese Vorgehensweise „teuer“ ist. Dieser Nachteil ist aber auch gleichzeitig der Vorteil: Die Herangehensweise ist einfach zu verstehen, der Programmcode lässt sich einfach lesen. Da pro Request ein Thread verwendet wird, sind auch alle Abläufe mit dem Kontext des Threads verbunden. Dadurch ist speziell Debugging bei imperativer Programmierung sehr gut möglich. Der Stack gehört zum Kontext des Threads, im Fehlerfall lässt sich der Stacktrace wie ein Buch lesen, potentielle Fehler lassen sich somit gut identifizieren.

Bei reaktiver Programmierung in Java wird ein Framework verwendet. Die Frameworks bieten die oben genannten Sprach-Konstrukte, um reaktive Programmierung zu ermöglichen. Dadurch wird der Code aber auch direkt komplizierter, da nun immer mit Subscribern und Publishern als Über- und Rückgabe statt mit Klassen gearbeitet wird.

Debugging und Fehleranalysen werden deshalb ebenfalls komplizierter, werden aber nach und nach durch Tools besser unterstützt.

Relationale Datenbanken in Java

Eine weitere Herausforderung von reaktiver Programmierung in Java ist die Verwendung von relationalen Datenbanken. Dafür wird in Java gewöhnlich JDBC verwendet. JDBC ist allerdings synchron und verwendet einen Thread-Pool. Datenbanken-Aufrufe werden in einer Queue gesammelt, für die Durchführung werden Threads aus einem Pool verwendet. Sind alle Threads aufgebraucht, blockiert JDBC, bis wieder Threads frei werden.

Vorteile	Nachteile
Einfach skalierbar und erreichbar, da keine Threads blockieren	Initial flache Lernkurve, erfordert Umdenken
Parallele Abfragen einfach implementierbar	Debugging und Fehleranalysen sind schwerer und ungewohnt
Deklarative Programmierung	Herausforderungen wie JDBC

Abbildung 3: Vor- und Nachteile von reaktiver Programmierung in Java

Bei reaktiver Programmierung darf aber kein Bestandteil blockieren. Für Unternehmen, die in der Datenhaltung auf relationale Datenbanken setzen, war somit die Einführung von reaktiver Programmierung in Java bis dato keine Option.

Dieses Problem wird durch r2dbc gelöst. R2DBC steht für «Reactive Relational Database Connectivity», es ist eine API Spezifikation für reaktive Treiber von relationalen Datenbanken. Es gibt bereits Treiber für h2, mariadb, mssql, mysql und postgres.

Wann lohnt sich reaktive Programmierung in Java?

Die kurze Antwort ist: Wenn das Thread-per-Request Modell nicht mehr ausreicht und durch das reaktive Programmiermodell die Hardware maximal ausgenutzt werden soll. Dem gegenüber steht ein (hauptsächlich anfangs) kompliziertes Programmiermodell, erschwertes Debugging und Fehleranalysen. Einige populäre Programmierer in der Java-Szene nennen besser lesbaren Code als weiteren Vorteil der reaktiven Programmierung. Statt iterativ vorzugehen, wird deklarativ beschrieben, was mit eingehenden Elementen auf Streams passiert. Die Verarbeitung ist «stateless».

Um in diese Denkweise zu gelangen, wird allerdings viel Zeit benötigt. Ein Team, welches wenig oder kaum Erfahrung in reaktiver Programmierung hat, würde beispielsweise für die Entwicklung von Anforderungen länger brauchen.

Abbildung 3 listet die Vor- und Nachteile auf, die in dem Artikel behandelt wurden.

Eventuell regt der Artikel an, zu eruieren, ob und wie ein Vorteil bei reaktiver Programmierung in der aktuellen Systemlandschaft geschaffen werden kann.

Gerne besprechen wir mit Ihnen zusammen die Vor- und Nachteile der reaktiven Programmierung bei Ihren Java-basierten Systemen. Kontaktieren Sie uns gerne unverbindlich.

Moritz Eberhard
Senior Software Engineer

ENTAGO

Entago AG
Buckhauserstrasse 34
CH-8048 Zurich

info@entago.ch

www.entago.ch