KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
11/2017
Jahrgang 10

Alles eine Frage der geschickten Sprünge

Der Weg ist das Ziel: Zugriffsoptimierung in Neo4J

von Ilker Yümsek

Akka

Nebenläufige, verteilte und fehlertolerante Applikationen mit Akka schreiben

von Dr. Martin Fluch

„Partageons ce qui nous départage"

Können wir zusammen, ohne Finanzierung, ohne Auftrag und in drei Monaten, ein Buch über unsere Firmenkultur schreiben?

Eine Buchrezension
von Tim Bourguignon

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

Sie können die elektronische Form des KaffeeKlatsch
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den

Versand des KaffeeKlatsch verwendet.

Editorial
Die liebe Vorweihnachtszeit

In der Adventszeit ist es wieder soweit. Unzählige Weihnachtsmärkte öffnen ihre Tore und – eventuell kennen Sie das auch – man weiß nicht, welchen man besuchen soll. (Vorausgesetzt natürlich man möchte überhaupt einen besuchen.) Und auch hierbei lässt sich mittlerweile App-Hilfe schaffen. So existieren in den App Stores verschiedene Weihnachtsmarkt-Apps, die Ihnen den Weg zum nächsten Glühweinstand weisen. Und wenn diese zudem noch von der Usability und den Funktionen her überzeugen können, ist die vorweihnachtliche Stimmung auch gerettet. Möglicherweise lassen sich über solche Apps sogar Weihnachtsmärkte finden, die einem bisher gar nicht bekannt waren und die sich dann als wahre Schmuckstücke entpuppen. Ausprobieren kann man es ja einmal. Bevor Sie sich jedoch zu einem Weihnachtsmarkt aufmachen, werfen Sie vorher noch einen Blick in die Themen unserer aktuellen Ausgabe!

In „*Alles eine Frage der geschickten Sprünge*" (Seite 6) geht Ilker Yümsek mit seinem Thema zur Neo4J-Datenbank in die dritte Runde. Dieses Mal erläutert er, wie die Zugriffsoptimierung bei der Datenstruktur in Neo4J funktioniert und sich die Arbeitsweise der Neo4J-Komponenten in Bezug auf den Datenzugriff darstellt.

Um skalierbare und fehlertolerante Applikationen zu schreiben, gibt es verschiedene Frameworks. Martin Fluch widmet sich in seinem Artikel dem Framework „*Akka*" (Seite 16) und stellt das auf dem Aktormodel basierende Framework anhand einer kleinen Anwendung vor.

Tim Bourguignon befasst sich in „*Partageons ce qui nous départage*" (Seite 21) mit dem gleichnamig veröffentlichten Buch des Unternehmens Octo. Da das Buch nie ins Deutsche übersetzt wurde, fasst er hier einige Aspekte der Firmenkultur Octos zusammen und stellt deren Besonderheiten heraus.

Etwas Unterhaltsames haben wir auch wieder in unserer Rubrik „*Das Allerletzte*" für Sie.

Ich wünsche Ihnen viel Spaß beim Lesen und eine schöne Adventszeit.

Ihr Oliver Klosa
Chefredakteur

Beitragsinformation

Der KaffeeKlatsch dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KaffeeKlatsch veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KaffeeKlatsch zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KaffeeKlatsch-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde. Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.
Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KaffeeKlatsch unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Herrn Oliver Klosa via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-0 gebucht werden.

Abonnement

Der KaffeeKlatsch erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KaffeeKlatsch via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Oliver Klosa via E-Mail unter redaktion@bookware.de oder telefonisch unter 09131/8903-0.

Copyright

Das Copyright des KaffeeKlatsch liegt vollständig bei der Bookware. Wir gestatten die Übernahme des KaffeeKlatsch in Datenbestände, wenn sie ausschließlich privaten Zwecken dienen. Das auszugsweise Kopieren und Archivieren zu gewerblichen Zwecken ohne unsere schriftliche Genehmigung ist nicht gestattet.

Sie dürfen jedoch die unveränderte PDF-Datei gelegentlich und unentgeltlich zu Bildungs- und Forschungszwecken an Interessenten verschicken. Sollten diese allerdings ein dauerhaftes Interesse am KaffeeKlatsch haben, so möchten wir diese herzlich dazu einladen, das Magazin direkt von uns zu beziehen. Ein regelmäßiger Versand soll nur über uns erfolgen.

Bei entsprechenden Fragen wenden Sie sich bitte per E-Mail an copyright@bookware.de.

Impressum

KaffeeKlatsch Jahrgang 10, Nummer 11, November 2016
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Software GmbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-99
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking / Dr. Oliver Klosa
Anzeigen: Dr. Oliver Klosa
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Alles eine Frage der geschickten Sprünge

Der Weg ist das Ziel: Zugriffsoptimierung in Neo4J

von Ilker Yümsek Seite 6

In diesem Beitrag wird anhand der Daten aus dem Artikel der KaffeeKlatsch-Ausgabe von September 2017 beschrieben, wie die Zugriffsoptimierung bei der Datenstruktur in Neo4J grundsätzlich funktioniert. Der Schwerpunkt liegt dabei nicht in der Darstellung aller Möglichkeiten für die Zugriffsoptimierung. Das Ziel ist viel mehr, die grundsätzliche Arbeitsweise der Neo4J-Komponenten bzgl. des Datenzugriffes darzulegen.

Akka

Nebenläufige, verteilte und fehlertolerante Applikationen mit Akka schreiben

von Dr. Martin Fluch Seite 16

Von modernen Applikationen wird in der heutigen Zeit erwartet, dass sie skalierbar und fehlertolerant sind. Akka ist ein in Scala geschriebenes Framework, welches basierend auf dem Aktormodel eine Sammlung von Werkzeugen für Scala und Java bereitstellt, um genau dieses Ziel zu erreichen.

„Partageons ce qui nous départage"

Können wir zusammen, ohne Finanzierung, ohne Auftrag und in drei Monaten, ein Buch über unsere Firmenkultur schreiben?

Eine Buchrezension von Tim Bourguignon Seite 21

Vier Mal war die Firma Octo Technology auf dem Podium der *„Great Place To Work"*-Awards in Frankreich und das zwei Mal als Sieger. Seit 2011 wurde diese französische Beratungsfirma mehrmals für ihre angenehme Arbeitsatmosphäre, ihre Kultur und ihr Flair geehrt. Gerne wäre man eine Maus, um diese Firma von innen erleben zu dürfen. Das ist aber nun nicht mehr nötig: Ende 2010 publizierten die Berater der Firma ihr eigenes Buch mit dem Titel *„Partageons ce qui nous departage"*, zu Deutsch: *„Teilen wir, was uns unterscheidet"*.

Beitragsinfo 4

User Groups 24

Das Allerletzte 27

Alles eine Frage der geschickten Sprünge

Der Weg ist das Ziel: Zugriffsoptimierung in Neo4J

In diesem Beitrag wird anhand der Daten aus dem Artikel der KaffeeKlatsch-Ausgabe von September 2017 beschrieben, wie die Zugriffsoptimierung bei der Datenstruktur in Neo4J grundsätzlich funktioniert. Der Schwerpunkt liegt dabei nicht in der Darstellung aller Möglichkeiten für die Zugriffsoptimierung. Das Ziel ist viel mehr, die grundsätzliche Arbeitsweise der Neo4J-Komponenten bzgl. des Datenzugriffes darzulegen. Mehr Informationen über die gesamte Bandbreite der Tuning-Möglichkeiten sind unter [2] und [3] verfügbar.

Der Anwendungsfall

Der Anwendungsfall für die Demonstration der Arbeitsweise baut auf die aufgesetzte Graph-Datenstruktur aus einem früheren Artikel [1].

Zur Erinnerung: Es ging dabei um die Carsharing-Informationen der DB mit folgenden Entitäten:

- Stationen
- Fahrzeuge
- Tarifklassen
- Buchungen

Die Entitäten waren dabei folgendermaßen strukturiert:

- Über unterschiedliche Stationen können Buchungen betätigt werden.
- Jede Station hat einen Betreiber
- In einer Buchung wird ein Fahrzeug ausgeliehen.
- Das Fahrzeug wird von einer Station abgeholt und in einer Station abgegeben
- Jede Buchung ist einer Tarifklasse zugeordnet.

In der finalen Graph-Datenstruktur wurden dann in Neo4J folgende Entitäten mit folgenden Beziehungen realisiert:

- Einer Buchung liegt immer ein Fahrzeug zugrunde (Beziehung REFERS_TO).
- Eine Buchung zeigt außerdem auf eine Tarifklasse (Beziehung ACCORDING_TO).
- Eine Tarifklasse baut auf eine Haupttarifklasse (Beziehung BELONGS_TO).
- Ein Fahrzeug wird über eine Station ausgeliehen (Beziehung WAS_BOOKED_IN)

Erst Zugriffspfade, dann das Schema

Bei der Aufsetzung einer Graph-Datenbank wie Neo4J sollte man folgenden Aspekt unbedingt berücksichtigen: Neo4J ist grundsätzlich schemafrei. Dies ermöglicht eine gewisse Freiheit bei der Verwaltung von „grauen Zonen", die fachlich und technisch noch nicht vollständig definiert sind. Diese Freiheit sollte man aber auf gar keinen Fall so nutzen, dass man die Graph-Datenstruktur komplett schemafrei lässt. Um ein betreibbares und im erwünschten Maße verfügbares System sicherzustellen, ist eine gründliche Konzeption der Graph-Datenstruktur unerlässlich. Im Rahmen dieser Konzeption müssen dann die Stärken der Datenbank und anhand deren eine performante Datenstruktur – insbesondere bei Lese-Zugriffen – im Fokus stehen. Wertvolle Tipps bzgl. der Aufsetzung einer Neo4J-Datenbank liefert ein Beitrag im Neo4J-Blog [4]. Wenn die grundsätzliche Graph-Datenstruktur konzipiert ist, kommen erst die Schritte für die Optimierung des Datenzugriffes. In diesem Bereich ist immer ein gesundes Maß einzuhalten. Weder Indizes über alle Nodes hinweg in gesamter Attribut-Bandbreite noch auf die IDs der Knoten eingeschränkt. Beim Finden der notwendigen Indizierungen stellen die Zugriffspfade die Grundlage dar. Mit dem Begriff Zugriffspfad ist hier die Art und Weise des Datenzugriffes (lesend und schreibend) gemeint. Anhand der Zugriffspfade, die fachlich und technisch für den Einsatz relevant sind, können die notwendigen Optimierungen getrieben werden. Dabei kommen diverse Werkzeuge/Komponenten, die Neo4J mitbringt, zum Einsatz. Im Folgenden werden diese Werkzeuge/Komponenten beschrieben. Außerdem ist es empfehlenswert, sich mit den Such-Algorithmen im Bereich der Graph-Datenstrukturen auseinanderzusetzen (siehe [5], [6], [7] und [8]).

Cypher

Die Abfragesprache Cypher bringt zwei Anweisungen mit, die für die Optimierung des Datenzugriffes eingesetzt werden: EXPLAIN und PROFILE. Bei beiden Anweisungen braucht man sie nur am Anfang einer regulären Cypher-Anweisung zu stellen.

- EXPLAIN untersucht dabei den Ausführungsplan (Execution-Plan) ohne die Cypher-Anweisung durchzuführen.
- PROFILE führt die Anweisung durch und ermittelt, welche Operatoren mit welcher Menge (Zeilenanzahl) von Daten in Berührung kommen und wie lange sie brauchen. PROFILE ist – wie es sich selbst schon erklärt – ressourcenintensiver als EXPLAIN und sollte deswegen mit Vorsicht eingesetzt werden.

Ausführungsplan

Der Ausführungsplan umfasst die einzelnen Schritte im Rahmen der Verarbeitung der Anweisung und liefert uns dabei Details über die

- Operatoren, die zum Einsatz kommen
- Datenmenge, die in jedem Schritt verarbeitet wird
- Ressourcen, die in Anspruch genommen werden und
- Laufzeit, die verbraucht wird.

Im Falle von EXPLAIN sind die Werte in einem Ausführungsplan Schätzungen, bei PROFILE ermittelte Werte.

Operatoren

Bei jedem Schritt in einem Ausführungsplan kommt ein sogenannter Operator zum Einsatz. Die Operatoren stehen für die unterschiedlichen Arten der Datenverarbeitung (Suche, Schreiben, Umwandeln etc.).

Planner

Damit ein Ausführungsplan zusammengestellt werden kann, müssen natürlich unterschiedliche Gegebenheiten analysiert und dabei in Erwägung gezogen werden, welchen Weg man bei der Verarbeitung der Daten nimmt. In diesem Bezug kommt dann der sogenannte Planner ins Spiel. Der Planner ermittelt anhand der Gegebenheiten, welche Verarbeitungsschritte mit welchen Operatoren durchgeführt werden muss. In Neo4J gibt es zwei Arten von Planner:

- COST: Der Cost-Planner berücksichtigt statistische Daten (inkl. Indizes) aus der Datenbank, um den günstigsten Weg zu ermitteln.
- RULE: Der Rule-Planner hat gewisse vorgegebene Regeln, die den Ausführungsplan bestimmen. Dabei berücksichtigt er zwar auch die Indizes, aber nicht die Statistiken.

Cost-Planner ist der Standard-Planner und ist als Default-Einstellung konfiguriert.

Wenn per EXPLAIN der Ausführungsplan ermittelt wird, werden die Schritte in der Ausgabe so dargestellt, dass die Reihenfolge der Schritte von unten nach oben ist. Außerdem sieht man am Anfang die Grund-Informationen über die Umgebung und über die ausgeführte Anweisung (DB, Planner-Art usw.):

Hinweis: Aus Platzgründen wurden in den Konsolen-Ausgaben die zusammengehörigen Spalten, wo es notwendig war, getrennt voneinander dargestellt. Dabei wurde als Trennzeichen „///“ verwendet, sodass unterhalb der jeweiligen Zeile mit demselben Trennzeichen beginnend die übrigen Spalten dargestellt wurden.

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.fuelType="Strom" RETURN *;
+-----------------------------------------------------------------///
| Plan      | Statement   | Version      |                        ///
+-----------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2" |                        ///
+-----------------------------------------------------------------///
///--------------------------------+
/// Planner | Runtime       | Time |
///--------------------------------+
/// „COST"  | „INTERPRETED" | 11   |
///--------------------------------+

+-----------------+----------------+-----------------------------///
| Operator        | Estimated Rows | Identifiers                 ///
+-----------------+----------------+-----------------------------///
| +ProduceResults |            296 | v                           ///
+-----------------+----------------------------------------------///
| +NodeIndexSeek  |            296 | v                           ///
+-----------------+----------------+-----------------------------///
///+--------------------+
///| Other              |
///+--------------------+
///|                    |
///+--------------------+
///| :VEHICLE(fuelType) |
///+--------------------+
```

Indizierungen

In Neo4J werden die Indizes über eine CREATE-Anweisung erstellt und über eine DROP-Anweisung ent-

fernt. Dabei liegt einem Index immer ein Label kombiniert mit einem oder mehreren Attributen zugrunde:

```
CREATE INDEX ON (VEHICLE:fuelType);
DROP INDEX ON (VEHICLE:fuelType);
```

Nach einer CREATE-Anweisung wird dabei nur die Erstellung des Index-Eintrags gestartet, die ggf. Zeit in Anspruch nimmt. Über die mitgelieferte Prozedur db.indexes() können die Indizes so aufgelistet werden, dass es sichtbar ist, ob der Index verfügbar ist (ONLINE) oder nicht (POPULATING):

```
neo4j> call db.indexes();
+-----------------------------------------------------------------------------------///
| description                                                                        ///
+-----------------------------------------------------------------------------------///
| „INDEX ON :BOOKING(bookingID)“                                                     ///
| „INDEX ON :BOOKING(technicalIncomeChannel)“                                        ///
| „INDEX ON :VEHICLE(capacityAmount)“                                                ///
| „INDEX ON :VEHICLE(kw)“                                                            ///
+-----------------------------------------------------------------------------------///
///-----------------------------------------+
///| state        | type                    |
///-----------------------------------------+
///| „POPULATING“ | „node_label_property“ |
///| „ONLINE“     | „node_label_property“ |
///| „ONLINE“     | „node_label_property“ |
///| „ONLINE“     | „node_label_property“ |
///-----------------------------------------+
```

Beispiele für die Zugriffsoptimierung

Im Folgenden wird nun demonstriert, wie die Komponenten bei der Zugriffsoptimierung eingesetzt werden und wie die Arbeitsweise bei der Ermittlung des Ausführungsplans grundsätzlich aussieht. Dabei bildet die indexbasierte Optimierung mit den Operatoren *NodeIndexSeek* und *NodeIndexSeekByRange* den Schwerpunkt. Detaillierte Informationen zum Thema Ausführungsplan und Operatoren sind unter dem Link [9] zu finden.

Navigieren durch die Labels

Im initialen Zustand einer Neo4J-Datenbank ist noch keine Indizierung vorhanden, außer dass die vorhandenen Labels bereits indiziert sind. Aus dem Grund werden z. B. bei folgender Abfrage die indizierten Labels als Einstiegspunkt verwendet (*Operator NodeByLabelScan*):

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.capacityAmount='90 l' RETURN *;
+-----------------------------------------------------------------------------///
| Plan      | Statement   | Version                                          ///
+-----------------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“                                     ///

+-----------------------------------------------------------------------------///
///----------------------------------+
///| Planner | Runtime       | Time |
///----------------------------------+
///| „COST“  | „INTERPRETED“ | 15   |
///----------------------------------+

+------------------+----------------+---------------------------///
| Operator         | Estimated Rows | Ident.                    ///
+------------------+----------------+---------------------------///
| +ProduceResults  |            177 | v                         ///
+----------------+-----------------------------------------------///
| +Filter        |            177 | v                           ///
+----------------+-----------------------------------------------///
| +NodeByLabelScan |           1773 | v                         ///
+------------------+----------------+---------------------------///
///+-------------------------------------+
///| Other                               |
///+-------------------------------------+
///|                                     |
///+-------------------------------------+
///| v.capacityAmount == {  AUTOSTRING0} |
///+-------------------------------------+
///| :VEHICLE                            |
///+-------------------------------------+

0 rows available after 15 ms, consumed after another 0 ms

neo4j> MATCH (v:VEHICLE) RETURN count(v) AS counts
order by 1;
+--------+
| counts |
+--------+
| 1773   |
+--------+
```

Die Spalte Estimated Rows gibt an, wie viele Zeilen im Worst-Case durchlaufen werden müssen, um den (die) gewünschten Knoten zu ermitteln. Weil die oben ermittelte Menge (1.773) die Gesamtanzahl der Knoten mit dem Label VEHICLE ist, hat der Ausführungsplan natürlich wesentliches Optimierungspotenzial. Die grundsätzliche Frage bei diesem Sachverhalt ist, wie die Anzahl der gelesenen Zeilen reduziert werden kann. Ein Blick auf die Diversität bei den Werten in der Spalte capacityAmount stellt für die Beantwortung dieser Frage eine gute Grundlage dar:

```
neo4j> MATCH (v:VEHICLE) RETURN
   distinct v.capacityAmount AS capacity order by 1;
+----------+
| capacity |
+----------+
| „60 l“   |
| NULL     |

| „52 l“   |
| „45 l“   |
| „42 l“   |
| „55 l“   |
| „75 l“   |
| „35 l“   |
| „90 l“   |
| „32 l“   |
| „70 l“   |
| „56 l“   |
| „66 l“   |
| „10 l“   |
| „63 l“   |
| „48 l“   |
+----------+

17 rows available after 27 ms, consumed after another 1 ms

neo4j> MATCH (v:VEHICLE) WITH
   distinct v.capacityAmount AS capacity
      RETURN COUNT(capacity) order by 1;
+-----------------+
| COUNT(capacity) |
+-----------------+
| 16              |
+-----------------+

1 row available after 22 ms, consumed after another 0 ms
```

Die Daten zeigen, dass eine Indizierung über die Spalte *capacityAmount* aufgrund der vernünftigen Verteilung der unterschiedlichen Werte Sinn macht:

```
neo4j> CREATE INDEX ON :VEHICLE(capacityAmount);
0 rows available after 3 ms, consumed after another 0 ms
Added 1 indexes
```

Übersichtlichkeit macht Hoffnung: NodeIndexSeek

Durch die durchgeführte Indizierung sind die 1.773 Entitäten mit 16 unterschiedlichen Werten bzgl. des Attributes *capacityAmount* „optimiert“ abgelegt.
Diese Sachlage führt dann zur folgenden Schlussfolgerung in einem Ausführungsplan: Es würde durchschnittlich ca. 105 Zugriffe kosten, bis die Entität(en) mit einem vorgegebenen capacityAmount-Wert ermittelt werden.

```
neo4j> EXPLAIN MATCH (v:VEHICLE) where
v.capacityAmount=‘90 l‘ RETURN *;
+-----------------------------------------------------------------///
| Plan      | Statement   | Version                                ///
+-----------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“                           ///
+-----------------------------------------------------------------///
///-----------------------------------+
///| Planner | Runtime       | Time |
///-----------------------------------+
///| „COST“  | „INTERPRETED“ | 16   |
///-----------------------------------+

+-----------------+----------------+----------------------------///
| Operator        | Estimated Rows | Ident.                     ///
+-----------------+----------------+----------------------------///
| +ProduceResults |            105 | v                          ///
+-----------------+----------------------------------------------///
| +NodeIndexSeek  |            105 | v                          ///
+-----------------+----------------+----------------------------///
///+--------------------------+
///| Other                    |
///+--------------------------+
///|                          |
///+--------------------------+
///| :VEHICLE(capacityAmount) |
///+--------------------------+

0 rows available after 16 ms, consumed after another 0 ms

neo4j> MATCH (v:VEHICLE) WITH v.capacityAmount AS ca,
   v as vehicle WITH ca as caValue, count(vehicle) as
      caC RETURN avg(caC);
+--------------------+
| avg(caC)           |
+--------------------+
| 104.29411764705884 |
+--------------------+

1 row available after 18 ms, consumed after another 0 ms
```

Wenn nach zwei Kriterien gesucht wird, kommt es natürlich darauf an, wie die Zugriffspfade im Vergleich abschneiden. Als Beispiel für so einen Fall wird nun das Attribut kw genommen:

```
neo4j> MATCH (v:VEHICLE) WITH distinct v.kw
   AS kw RETURN COUNT(kw) order by 0;
+-----------+
| COUNT(kw) |
+-----------+
| 30        |
+-----------+
```

```
1 row available after 18 ms, consumed after another 0 ms
neo4j> MATCH (v:VEHICLE) RETURN distinct v.kw order
by v.kw;
+------+
| v.kw |
+------+
| 0    |
| 17   |
| 20   |
| 30   |
| 35   |
| 44   |
| 50   |
| 51   |
| 55   |
| 63   |
| 66   |
| 68   |
| 70   |
| 74   |
| 75   |
| 78   |
| 80   |
| 81   |
| 84   |
| 85   |
| 88   |
| 95   |
| 100  |
| 103  |
| 110  |
| 111  |
| 118  |
| 120  |
| 125  |
| 132  |
+------+
```

Wenn nach einem kw-Wert ohne Indizierung gesucht wird, muss man die gesamte Bandbreite der Daten in Betracht ziehen. Dabei gibt es bei 1.773 Entitäten 30 unterschiedliche Werte bzgl. des Attributes kw. Wenn diese Entitäten durch einen Index organisiert vorliegen, kostet es durchschnittlich ca. 59 Schritte bis die Entität(en) mit einem vorgegebenen kw-Wert ermittelt werden. Auch wenn dieselbe Optimierung für das Attribut *capacityAmount* vorliegt, wird *kw* als günstigerer Einstiegspunkt genommen:

```
neo4j> MATCH (v:VEHICLE) WITH v.kw AS kw,
   v as vehicle WITH kw as kwValue, count(vehicle)
      as kwCount RETURN avg(kwCount);
+--------------------+
| avg(kwCount)       |
+--------------------+

| 59.099999999999994 |
+--------------------+
```

```
1 row available after 11 ms, consumed after another 0 ms

neo4j> EXPLAIN MATCH (v:VEHICLE) where
v.capacityAmount='90 l' and v.kw = 63 RETURN *;
+-----------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-----------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2"                            ///
+-----------------------------------------------------------------///
///-------------------------------+
///| Planner | Runtime       | Time |
///-------------------------------+
///| „COST"  | „INTERPRETED" | 37   |
///-------------------------------+

+-----------------+----------------+----------------------------///
| Operator        | Estimated Rows | Ident.                     ///
+-----------------+----------------+----------------------------///
| +ProduceResults |              3 | v                          ///
+----------------+-----------------------------------------------///
| +Filter         |              3 | v                          ///
+----------------+-----------------------------------------------///
| +NodeIndexSeek  |             59 | v                          ///
+-----------------+----------------+----------------------------///
///+-----------------------------------+
///| Other                             |
///+-----------------------------------+
///|                                   |
///+-----------------------------------+
///| v.capacityAmount == {  AUTOSTRING0} |
///+-----------------------------------+
///| :VEHICLE(kw)                      |
///+-----------------------------------+

0 rows available after 37 ms, consumed after another 0 ms
```

Besser, aber nicht immer: Suchen nach einer Range

Wenn nach einem Nummern- bzw. Namensraum („>", „<", „STARTS WITH" usw.) gesucht wird, dann kann man einen bestimmten Teil der Daten unberücksichtigt lassen. In so einem Fall kommt der Operator *NodeIndexSeekByRange* zum Einsatz:

```
1 row available after 21 ms, consumed after another 0 ms
   neo4j> EXPLAIN MATCH (v:VEHICLE)
      where v.kw > 20 RETURN *;
+-----------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-----------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2"                            ///
+-----------------------------------------------------------------///
///-------------------------------+
///| Planner | Runtime       | Time |
```

```
///----------------------------------+
///| „COST“  | „INTERPRETED“ | 10   |
///----------------------------------+

+-----------------------+----------------------------------------///
| Operator              | Estimated Rows                         ///
+-----------------------+----------------------------------------///
| +ProduceResults       |            51                          ///
+-----------------------------------------------------------------///
| +NodeIndexSeekByRange |            51                          ///
+-----------------------+----------------------------------------///
///+-------------+----------------------------+
///| Identifiers | Other                      |
///+-------------+----------------------------+
///| v           |                            |
///+-------------+----------------------------+
///| v           | :VEHICLE(kw) > {  AUTOINT0} |
///+-------------+----------------------------+

0 rows available after 10 ms, consumed after another 0 ms
```

Bei der Schätzung durch den Planner spielt dabei die vorgegebene Range keine Rolle:

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.kw > 120 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“                            ///
+-------------------------------------------------------------------///
///----------------------------------+
///| Planner | Runtime       | Time |
///----------------------------------+
///| „COST“  | „INTERPRETED“ | 9    |
///----------------------------------+

+-----------------------+----------------------------------------///
| Operator              | Estimated Rows                         ///
+-----------------------+----------------------------------------///
| +ProduceResults       |            51                          ///
+-----------------------------------------------------------------///
| +NodeIndexSeekByRange |            51                          ///
+-----------------------+----------------------------------------///
///+-------------+----------------------------+
///| Identifiers | Other                      |
///+-------------+----------------------------+
///| v           |                            |
///+-------------+----------------------------+
///| v           | :VEHICLE(kw) > {  AUTOINT0} |
///+-------------+----------------------------+

0 rows available after 9 ms, consumed after another 0 ms
```

Es scheint so zu sein, dass ein Optimierungsfaktor von dem Schätzungswert im Falle einer exakten Wertvorgabe (59) abgezogen wird, wodurch man dann den Schätzungswert 51 bekommt. Auf jeden Fall ist der Operator *NodeIndexSeekByRange* (im Falle von numerischen Werten) optimistischer und performant. Hier sind einige Testzugriffe für beide Fälle (NodeIndexSeek und NodeIndexSeekByRange):

```
neo4j> PROFILE MATCH (v:VEHICLE)
   where v.kw = 120 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version     | Planner                     ///
+-------------------------------------------------------------------///
| „PROFILE“ | „READ_ONLY“ | „CYPHER 3.2“ | „COST“                     ///
+-------------------------------------------------------------------///
///----------------------------------------+
///| Runtime       | Time | DbHits | Rows |
///----------------------------------------+
///| „INTERPRETED“ | 7    | 0      | 18   |
///----------------------------------------+

+-----------------+----------------+------+----------------------///
| Operator        | Estimated Rows | Rows | DB Hits              ///
+-----------------+----------------+------+----------------------///
| +ProduceResults |             59 |   18 |    0                 ///
+----------------+------+-----------------------------------------///
| +NodeIndexSeek  |             59 |   18 |   19                 ///
+-----------------+----------------+------+----------------------///
///+-----------+-------------+--------------+
///| Cache H/M | Identifiers | Other        |
///+-----------+-------------+--------------+
///|       0/0 | v           |              |
///+-----------+-------------+--------------+
///|       0/0 | v           | :VEHICLE(kw) |
///+-----------+-------------+--------------+

neo4j> PROFILE MATCH (v:VEHICLE)
   where v.kw > 100 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version     | Planner                     ///
+-------------------------------------------------------------------///
| „PROFILE“ | „READ_ONLY“ | „CYPHER 3.2“ | „COST“                     ///
+-------------------------------------------------------------------///
///----------------------------------------+
///| Runtime       | Time | DbHits | Rows |
///----------------------------------------+
///| „INTERPRETED“ | 63   | 0      | 76   |
///----------------------------------------+

+-----------------------+----------------+-----------------------///
| Operator              | Estimated Rows | Rows                  ///
+-----------------------+----------------+-----------------------///
| +ProduceResults       |             51 |   76                  ///
+----------------+-----------------------------------------------///
| +NodeIndexSeekByRange |             51 |   76                  ///
```

```
+-----------------------+----------------+-----------------------///
///+---------+-----------+---------------------------------------///
///| DB Hits | Cache H/M | Identifiers                           ///
///+---------+-----------+---------------------------------------///
///|       0 |       0/0 | v                                     ///
///+---------+-----------+---------------------------------------///
///|      77 |       0/0 | v                                     ///
///+---------+-----------+---------------------------------------///
///+-----------------------------+
///| Other                       |
///+-----------------------------+
///|                             |
///+-----------------------------+
///| :VEHICLE(kw) > {  AUTOINT0} |
///+-----------------------------+

neo4j> PROFILE MATCH (v:VEHICLE)
   where v.kw > 20 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „PROFILE“ | „READ_ONLY“ | „CYPHER 3.2“                            ///
+-------------------------------------------------------------------///
///-------------------------------------------------+
///| Planner | Runtime       | Time | DbHits | Rows |
///-------------------------------------------------+
///| „COST“  | „INTERPRETED“ | 27   | 0      | 1761 |
///-------------------------------------------------+

+-----------------------+----------------+-----------------------///
| Operator              | Estimated Rows | Rows                    ///
+-----------------------+----------------+-----------------------///
| +ProduceResults       |             51 | 1761                    ///
+----------------+------------------------------------------------///
| +NodeIndexSeekByRange |             51 | 1761                    ///
+-----------------------+----------------+-----------------------///
///+---------+-----------+---------------------------------------///
///| DB Hits | Cache H/M | Identifiers                           ///
///+---------+-----------+---------------------------------------///
///|       0 |       0/0 | v                                     ///
///+---------+-----------+---------------------------------------///
///|    1762 |       0/0 | v                                     ///
///+---------+-----------+---------------------------------------///
///+-----------------------------+
///| Other                       |
///+-----------------------------+
///|                             |
///+-----------------------------+
///| :VEHICLE(kw) > {  AUTOINT0} |
///+-----------------------------+

neo4j> PROFILE MATCH (v:VEHICLE)
   where v.kw = 51 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version      | Planner                    ///
+-------------------------------------------------------------------///
| „PROFILE“ | „READ_ONLY“ | „CYPHER 3.2“ | „COST“                     ///
+-------------------------------------------------------------------///
///----------------------------------------+
///| Runtime       | Time | DbHits | Rows |
///----------------------------------------+
///| „INTERPRETED“ | 15   | 0      | 127  |
///----------------------------------------+

+-----------------+----------------+------+-----------------------///
| Operator        | Estimated Rows | Rows | DB Hits                 ///
+-----------------+----------------+------+-----------------------///
| +ProduceResults |             59 |  127 |    0                    ///
| |               +----------------+------+---------------------------///
| +NodeIndexSeek  |             59 |  127 |  128                    ///
+-----------------+----------------+------+-----------------------///
///+-----------+-------------+--------------+
///| Cache H/M | Identifiers | Other        |
///+-----------+-------------+--------------+
///|       0/0 | v           |              |
///+-----------+-------------+--------------+
///|       0/0 | v           | :VEHICLE(kw) |
///+-----------+-------------+--------------+
```

Dieses bessere Abschneiden von *NodeIndexSeekByRange* bei numerischen Werten wird auch im folgenden Beispiel deutlich: Wenn nach einem exakten Brennstoff-Wert und einer Range für den kw-Wert gesucht wird, liefert der Operator *NodeIndexSeekByRange* für das Attribut kw den besseren Einstiegspunkt:

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.fuelType="Strom“ and v.kw > 20 RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version      | Planner                    ///
+-------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“ | „COST“                     ///
+-------------------------------------------------------------------///
///-----------------------+
///| Runtime       | Time |
///-----------------------+
///| „INTERPRETED“ | 4    |
///-----------------------+

+-----------------------+----------------+-----------------------///
| Operator              | Estimated Rows | Ident.                  ///
+-----------------------+----------------+-----------------------///
| +ProduceResults       |              9 | v                       ///
+----------------+------------------------------------------------///
| +Filter        |              9 | v                             ///
+----------------+------------------------------------------------///
| +NodeIndexSeekByRange |             51 | v                     ///
+-----------------------+----------------+-----------------------///
///+-------------------------------+
///| Other                         |
///+-------------------------------+
///|                               |
```

```
///+-------------------------------+
///| v.fuelType == {  AUTOSTRING0} |
///+-------------------------------+
///| :VEHICLE(kw) > {  AUTOINT1}   |
///+-------------------------------+

0 rows available after 4 ms, consumed after another 0 ms
```

Dies bedeutet aber nicht, dass der Operator *NodeIndexSeekByRange* dem Operator *NodeIndexSeek* immer bevorzugt wird und bessere Schätzungswerte liefert. Es ist immer entscheidend, wie die vorhandenen Werte/Mengen/Varianten in der Datenstruktur aussehen. Dies zeigt das nachfolgende Beispiel, in dem auch die Performance beim rangebasierten, textuellen Suchen mit *NodeIndexSeekByRange* gezeigt wird:

Wenn die Verarbeitungsschritte auf Basis des Brennstoffes untersucht werden, liegt der Schätzwert (im Falle einer exakten Wertvorgabe) im Bereich von 296. Im Falle einer Range (textbasierte Suche) ist er im Bereich von 1.755.

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.fuelType="Strom" RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2"                            ///
+-------------------------------------------------------------------///
///--------------------------------+
///| Planner | Runtime       | Time |
///--------------------------------+
///| „COST"  | „INTERPRETED" | 11   |
///--------------------------------+

+-----------------+----------------+----------------------------///
| Operator        | Estimated Rows | Identifiers                ///
+-----------------+----------------+----------------------------///
| +ProduceResults |            296 | v                          ///
+----------------+-----------------------------------------------///
| +NodeIndexSeek  |            296 | v                          ///
+-----------------+----------------+----------------------------///
///+--------------------+
///| Other              |
///+--------------------+
///|                    |
///+--------------------+
///| :VEHICLE(fuelType) |
///+--------------------+

neo4j> MATCH (v:VEHICLE) WITH v.fuelType AS ft,
   v as vehicle WITH ft as ftValue, count(vehicle)
      as ftC RETURN avg(distinct ftC);
+------------------+
| avg(distinct ftC) |
+------------------+
| 295.5            |
+------------------+

1 row available after 23 ms, consumed after another 0 ms

neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.fuelType STARTS WITH „Strom" RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version      | Planner                    ///
+-------------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2" | „COST"                     ///
+-------------------------------------------------------------------///
///-----------------------+
///| Runtime       | Time |
///-----------------------+
///| „INTERPRETED" | 77   |
///-----------------------+

+-----------------------+----------------+-------------------------///
| Operator              | Estimated Rows | Ident.                  ///
+-----------------------+----------------+-------------------------///
| +ProduceResults       |           1755 | v                       ///
+----------------+-------------------------------------------------///
| +NodeIndexSeekByRange |           1755 | v                       ///
+-----------------------+----------------+-------------------------///
///+-----------------------------------------------+
///| Other                                         |
///+-----------------------------------------------+
///|                                               |
///+-----------------------------------------------+
///| :VEHICLE(fuelType STARTS WITH {  AUTOSTRING0}) |
///+-----------------------------------------------+
```

Werden die Verarbeitungsschritte auf Basis der Tankkapazität untersucht, liegt der Schätzwert (im Falle einer exakten Wertvorgabe) im Bereich von 105. Im Falle einer Range (textbasierte Suche) ist er im Bereich von 1.389.

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.capacityAmount="42 l" RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „EXPLAIN" | „READ_ONLY" | „CYPHER 3.2"                            ///
+-------------------------------------------------------------------///
///--------------------------------+
///| Planner | Runtime       | Time |
///--------------------------------+
///| „COST"  | „INTERPRETED" | 15   |
///--------------------------------+

+----------------+----------------+----------------------------///
```

```
| Operator        | Estimated Rows | Ident.                          ///
+-----------------+----------------+-----------------------------///
| +ProduceResults |            105 | v                               ///
+----------------+-------------------------------------------///
| +NodeIndexSeek  |            105 | v                               ///
+-----------------+----------------+-----------------------------///
///+-------------------------+
///| Other                   |
///+-------------------------+
///|                         |
///+-------------------------+
///| :VEHICLE(capacityAmount) |
///+-------------------------+

0 rows available after 15 ms, consumed after another 0 ms

neo4j> MATCH (v:VEHICLE) WITH v.capacityAmount AS ca,
   v as vehicle WITH ca as caValue, count(vehicle)
      as caC RETURN avg(caC);
+--------------------+
| avg(caC)           |
+--------------------+
| 104.29411764705884 |
+--------------------+

1 row available after 25 ms, consumed after another 0 ms
neo4j> EXPLAIN MATCH (v:VEHICLE) where
v.capacityAmount STARTS WITH „42“ RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“                            ///
+-------------------------------------------------------------------///
///-------------------------------+
///| Planner | Runtime       | Time |
///-------------------------------+
///| „COST“  | „INTERPRETED“ | 15   |
///-------------------------------+

+-----------------------+----------------+-----------------------///
| Operator              | Estimated Rows | Ident.                  ///
+-----------------------+----------------+-----------------------///
| +ProduceResults       |           1389 | v                       ///
+----------------+-------------------------------------------///
| +NodeIndexSeekByRange |           1389 | v                       ///
+-----------------------+----------------+-----------------------///
///+-----------------------------------------+
///| Other                                   |
///+-----------------------------------------+
///|                                         |
///+-----------------------------------------+
///| :VEHICLE(capacityAmount STARTS WITH     |
///|                        {  AUTOSTRING0}) |
///+-----------------------------------------+
0 rows available after 15 ms, consumed after another 0 ms
```

Bei einer textuellen Range-Suche für beide Attribute (*capacityAmount* und *fuelType*) wird dann der günstigste Punkt für den Einstieg genommen und danach nach der zweiten Bedingung gefiltert:

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.capacityAmount STARTS WITH „42“
      and v.fuelType STARTS WITH „Strom“ RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version      | Planner                    ///
+-------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“ | „COST“                     ///
+-------------------------------------------------------------------///
///-----------------------+
///| Runtime       | Time |
///-----------------------+
///| „INTERPRETED“ | 51   |
///-----------------------+

+-----------------------+----------------+-----------------------///
| Operator              | Estimated Rows | Ident.                  ///
+-----------------------+----------------+-----------------------///
| +ProduceResults       |           1374 | v                       ///
+----------------+-------------------------------------------///
| +Filter               |           1374 | v                       ///
+----------------+-------------------------------------------///
| +NodeIndexSeekByRange |           1389 | v                       ///
+-----------------------+----------------+-----------------------///
///+-----------------------------------------+
///| Other                                   |
///+-----------------------------------------+
///|                                         |
///+-----------------------------------------+
///| StartsWith(v.fuelType,{AUTOSTRING1})    |
///+-----------------------------------------+
///| :VEHICLE(capacityAmount STARTS WITH     |
///|                        {AUTOSTRING0}) |
///+-----------------------------------------+

0 rows available after 51 ms, consumed after another 0 ms
```

Wenn aber die Bedingung bzgl. des Attributes fuelType eine exakte Wertvorgabe ist, dann wird der Operator *NodeIndexSeek* dem Operator *NodeIndexSeekByRange* als Einstiegspunkt bevorzugt, weil er in der Schätzung besser liegt:

```
neo4j> EXPLAIN MATCH (v:VEHICLE)
   where v.capacityAmount STARTS WITH „42“
      and v.fuelType=“Strom“ RETURN *;
+-------------------------------------------------------------------///
| Plan      | Statement   | Version                                 ///
+-------------------------------------------------------------------///
| „EXPLAIN“ | „READ_ONLY“ | „CYPHER 3.2“                            ///
+-------------------------------------------------------------------///
///-------------------------------+
///| Planner | Runtime       | Time |
///-------------------------------+
///| „COST“  | „INTERPRETED“ | 35   |
///-------------------------------+
+-----------------+----------------+-----------------------------///
| Operator        | Estimated Rows | Ident.                          ///
```

```
+----------------+----------------+-----------------------------///
| +ProduceResults |          231 | v                              ///
+----------------+-----------------------------------------------///
| +Filter        |          231 | v                              ///
+----------------+-----------------------------------------------///
| +NodeIndexSeek |          296 | v                              ///
+----------------+----------------+-----------------------------///
///+-----------------------------------------------+
///| Other                                         |
///+-----------------------------------------------+
///|                                               |
///+-----------------------------------------------+
///| StartsWith(v.capacityAmount,{  AUTOSTRING0}) |
///+-----------------------------------------------+
///| :VEHICLE(fuelType)                            |
///+-----------------------------------------------+

0 rows available after 35 ms, consumed after another 0 ms
```

Fazit

Es ist essentiell, beim Aufsetzen und Betreiben einer Neo4J-Datenbank sich mit der Datenstruktur und mit der Optimierung des Datenzugriffes auseinanderzusetzen und die vorgehabten/erfolgten Zugriffe mit einem kritischen Blick zu betrachten. Auch wenn die Feststellungen/Interpretationen in diesem Artikel nicht vollständig sind und nur einen Teilaspekt abdecken, so haben die Leserinnen und Leser einen ersten Eindruck von der Arbeitsweise der Operatoren/Planner in Neo4J erhalten.

Referenzen

[1] KaffeeKlatsch *September 2017*, http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2017-09.pdf

[2] The Neo4J Developer Manual *Indexes*, http://neo4j.com/docs/developer-manual/current/cypher/schema/index/

[3] The Neo4J Developer Manual *Query tuning*, http://neo4j.com/docs/developer-manual/current/cypher/query-tuning/

[4] The dark side Neo4J, https://neo4j.com/blog/dark-side-neo4j-worst-practices

[5] Neo4J *Graph Algorithms*, https://neo4j-contrib.github.io/neo4j-graph-algorithms/

[6] Neo4J *Efficient Graph Algorithms in Neo4j*, https://neo4j.com/blog/efficient-graph-algorithms-neo4j

[7] Neo4J *Graph Search-Algorithms*, https://neo4j.com/blog/graph-search-algorithm-basics

[8] Neo4J *Graph Compute with Neo4j*, https://neo4j.com/blog/graph-compute-neo4j-algorithms-spark-extensions

[9] Neo4J *Execution-Plans*, http://neo4j.com/docs/developer-manual/current/cypher/execution-plans

Kurzbiografie

Ilker Yümsek arbeitet als Spezialist IT-Beratung bei Schaeffler Technologies AG & Co. KG im Bereich der System-Integration. Er beschäftigt sich seit 2001 mit der Konzeption und Entwicklung von Software-Lösungen im Enterprise-Umfeld. Besonderes Interesse hat er an der Datenverarbeitung und -integration sowie an Graph-Datenbanken.

Herbstcampus

Wissenstransfer par excellence

4.– 6. September 2018 in Nürnberg

Akka

Nebenläufige, verteilte und fehlertolerante Applikationen mit Akka schreiben

von Dr. Martin Fluch

Von modernen Applikationen wird in der heutigen Zeit erwartet, dass sie skalierbar und fehlertolerant sind. Akka ist ein in Scala geschriebenes Framework, welches basierend auf dem Aktormodel eine Sammlung von Werkzeugen für Scala und Java bereitstellt, um genau dieses Ziel zu erreichen.

Die Vorgeschichte

Das Aktormodel, auf dem Akka [1] basiert, ist nicht neu. Es wurde von Carl Hewitt und seinen Kollegen im Jahr 1973 entwickelt, um ein effektives Architekturmodel für KI-Systeme bereitzustellen, die eine hochgradige Parallelisierung erfordern [2]. Der Kern des Architekturmodels sind Aktoren, welche in diesem Model die elementare Einheit für die Verarbeitung, Speicher und die Kommunikation darstellen [3]. Aktoren sind isoliert voneinander, teilen keinen veränderlichen Zustand und kommunizieren durch asynchrone Nachrichten. Dabei wird ihr Verhalten durch die folgenden drei Axiome bestimmt, welche angeben, wie ein Aktor auf eine Nachricht reagieren kann, die er bekommt, nämlich:

1. Er kann weitere Aktoren erzeugen.
2. Er kann Nachrichten an andere Aktoren schicken.
3. Er kann bestimmen, wie er auf die Nachricht, die er als nächstes erhält, reagieren wird.

Es ist wichtig zu verstehen, dass innerhalb eines Aktors die Verarbeitung immer sequenziell geschieht, da ein Aktor nie zwei Nachrichten gleichzeitig verarbeitet. Parallelisierung erreicht man erst dadurch, dass man ein Problem an viele Aktoren verteilt, damit sie dieses unabhängig voneinander verarbeiten können.

Unabhängig von Hewitts Arbeit wurde 1986 beim Telekommunikationsunternehmen Ericsson die Sprache Erlang [4] für ihre Produkte entwickelt. Ziel war es, eine Sprache zu entwickeln, bei der Nebenläufigkeit und Fehlertoleranz integraler Bestandteil der Sprache sind. Prozesse in Erlang sind die Aktoren in Hewitts Model und die Sprache bietet ein Überwachungskonzept für seine Prozesse, die eine überragende Fehlertoleranz ermöglichen. Ein legendäres Beispiel für fehlertolerante Software, die in Erlang geschrieben wurde, ist der ATM-Switch AXD30 von Ericsson. Die Software besteht aus mehr als 2 Millionen Zeilen Code in Erlang, dem eine Zuverlässigkeit von 99,9999999% („nine nines reliability") nachgesagt wird [5, s. 203].

Andere Beispiele für den Einsatz von Erlang sind Facebooks Chat, bei dessen Onlinegang 2008 das System über Nacht die Last von 70 Millionen Benutzer verkraften musste [6], sowie der Message-Broker RabbitMQ [7] und die dokumentorientierte Datenbank CouchDB [8].

Etwa 2006/2007 fanden die Ideen, welche Erlang zum Aktormodel hatte, durch die Arbeit von Phillip Haller Einzug in Scala und schließlich schrieb Jonas Bonér Akka als eine Neuimplementation der Aktor-Core-Implementation von Scala [9]. Mit Scala 2.10.0 hat Akka die ursprünglichen Aktor-Bibliotheken in Scala offiziell abgelöst [11]. Andere Einsatzgebiete, allesamt ebenfalls Open Source, sind zum Beispiel die Big-Data-Datenfluss-Engine Gearpump [12], welche ursprünglich von Intel entwickelt wurde, sowie das Stream-Processing-Framework Apache Flink [13] oder das Lagom-Framework zum Schreiben von reaktiver Microservice-Systeme in Java oder Scala [14].

Akka ist ein vielseitiges Framework. Selbst ein Buch wie Akka in action [15] kann nur einen Überblick geben, wie Akka das Aktormodel implementiert, welche Möglichkeiten es bietet, verteilte und fehlertolerante Applikationen zu schreiben und welche Strategien sich anbieten, Probleme mit dem Aktormodel zu lösen. Insofern kann auch dieser Artikel schon nur an der Oberfläche der Oberfläche der Thematik kratzen. Wir werden dies anhand von Beispielen in Java unternehmen. Akka benötigt dabei mindestens Java 8.

Hello World! in Akka

Um eine minimale Anwendung in Akka zu schreiben, benötigt man nur eine Abhängigkeit zu einem Artefakt und keine weitere Konfiguration:

```
<dependency>
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-actor_2.12</artifactId>
  <version>2.5.6</version>
</dependency>
```

Aktoren sind leichtgewichtige Objekte und benötigen als Lebensraum ein Aktorsystem. Dieses muss in Akka einmal gestartet werden. Hierzu gibt es in der Klasse *akka.actor.ActorSystem* die statische Methode create. Dieses startet das Akka-System, welches die Verwaltung und Kommunikation aller Aktoren übernimmt. Ein Aktorsystem kann explizit durch den Aufruf der Methode *terminate* wieder beendet werden:[1]

```
ActorSystem system = ActorSystem.create();
try {
   // Erzeuge Aktoren, starte Verarbeitung
   // und warte auf Beendigung.
} finally{
   system.terminate();
}
```

Ein Aktor ist nun eine Klasse, die von der Klasse *akka.actor.AbstractActor* ableitet. Diese abstrakte Klasse bietet die nötige Schnittstelle, damit sich der Aktor in das Aktorsystem einfügen kann. Es muss mindestens die abstrakte Methode *createReceive* überschrieben werden, in der bestimmt wird, wie der Aktor auf unterschiedliche Nachrichten reagieren soll. Dabei ist eine Nachricht einfach nur ein beliebige, unveränderliche Klasse. Ein Beispiel:

```
public class Greeter extends Actor {
   private String template;
   public Greeter(String template) {
      this.template = template;
   }

@Override
   public Receive createReceive() {
      return receiveBuilder()
         .match(GreetingMsg.class, this::greet)
         .build();
   }

   public void greet(GreetingMsg msg) {
      String greeting = String.format(
         template, msg.name);
      System.out.println(greeting);
   }
}

public class GreetingMsg {
   public final String name;
   public GreetingMsg(String name) {
      this.name = name;
   }
}
```

1 Im Allgemeinen wird man allerdings nicht, wie in diesem Beispiel, explizit auf die Beendigung der Aufgaben warten, um die terminate-Methode aufzurufen, sondern dies wird implizit geschehen.

Einen Aktor erzeugt man allerdings nicht mit dem *new*-Operator von Java, sondern Akka bietet unterschiedliche Factory-Methoden hierfür an. Damit diese wissen, welchen Aktor und welche Parameter für den Aufruf des Konstruktors verwendet werden müssen, gibt es die Klasse *akka.actor.Props*:

```
Props greeterProps = Props.create(
   Greeter.class,„Hallo %s!“
);
```

Nun kann man damit einen Aktor erzeugen, zum Beispiel:

```
ActorRef greeterActor =
   system.actorOf(greeterProps, „greeter“);
```

wobei system die Instanz des Aktorsystems aus dem obigen Beispiel und der zweite Parameter ein (fast) freiwählbarer Name ist. Wie zu sehen ist, erhält man nicht eine Instanz von Greeter, sondern eine Instanz von *akka.actor.ActorRef* zurück. Eine Nachricht an diesen Aktor kann man nun wie folgt schicken:

```
GreetingMsg greetingMsg = new GreetingMsg(„Welt“);
greeterActor.tell(msg, null);
```

Dies führt dann dazu, dass Akka das Objekt *greetingMsg* an die Methode greet der erzeugten Instanz des Aktors Greeter schickt und dieser dann „Hallo Welt!“ ausgibt.

Was im Hintergrund geschieht und geschehen kann

Das ist schon ein wenig Aufwand, um *„Hallo Welt!“* auszugeben. Interessant ist, was im Hintergrund passiert. Zuerst ist anzumerken, dass die Verarbeitung der Nachricht asynchron geschieht. Zu unserem Greeter-Aktor gehört eine Mailbox, in die Akka unsere Nachricht einstellt. Sobald die Nachricht eingestellt ist, kehrt die Programmausführung aus der *tell*-Methode zurück. Akka verwaltet Thread-Pools, welche sich um die Abarbeitung von Nachrichten in den Mailboxen von Aktoren kümmert. Dabei wird immer sicher gestellt, dass nie mehr als eine Nachricht von einem Aktor bearbeitet wird.[2]

Die Reihenfolge, in welcher die Nachrichten in einer Mailbox ankommen, ist nur für Nachrichten, die

2 Das heißt, zu jeder beliebigen Zeit wird eine Instanz eines Aktors immer nur von höchstens einem Thread besucht.

aus einem Thread geschickt werden, garantiert: in diesem Fall kommen die Nachrichten genau in der Reihenfolge an, wie sie gesendet worden sind. Zugestellt wird nach dem Best-Effort-Prinzip, wobei eine Nachricht nicht doppelt zugestellt wird.

In unserem Beispiel befindet sich die Instanz des Greeter-Aktors in der gleichen JVM, in der auch der aufrufende Thread beheimatet ist. Das ist aber nicht zwingend notwendig. Ein *ActorRef*-Objekt kann aber auch auf einen Aktor verweisen, welcher in einem Aktorsystem in einer anderen JVM lebt. Diese JVM kann sogar auf einem anderen Server und die Kommunikation über das Netzwerk laufen, ohne dass es zum Senden der *greetingMsg*-Nachricht irgendetwas anders machen müsste.

Eine weitere Konsequenz ist, dass der Sender einer Nachricht nur eine Referenz auf einen Aktor und nicht den Aktor selbst in Händen hält und dass Akka bei Bedarf die Instanz eines Aktors austauschen kann, ohne dass der Aufrufer etwas davon mitbekommt. Dadurch kann bei der Behandlung von Fehlerfällen profitiert werden.

Hierarchische Strukturen

Im einführenden Beispiel haben wir die Methode *actorOf* der Klasse *ActorSystem* benutzt, um einen Aktor zu erzeugen. In der Einleitung sagt das erste Axiom über Aktoren, dass ein Aktor selbst Aktoren erzeugen kann. In Akka geschieht dies über den *Aktor*-Kontext. Diesen erhält man über die Methode *getContext* der Klasse *AbstractActor*. Dieser Kontext hat ebenfalls eine *actorOf*-Methode, mit der man einen Aktor genauso wie mit der gleichnamigen Methode aus der Klasse *ActorSystem* erzeugen kann. Worin liegt der Unterschied?

Jeder Aktor in Akka hat einen Parent-Aktor. Für Aktoren, die mit Methode *ActorContext.actorOf*-Methode erzeugt wurden, ist dies der Aktor, dessen Kontext man zur Erzeugung verwendet hat. Die Aktoren, die mit *ActorSystem.actorOf*-Methode erzeugt wurden, haben den „*guardian*"-Aktor als *Parent*-Aktor. Je Aktorsystem gibt es genau einen „*guardian*"-Aktor. Daraus ergibt sich, dass alle Aktoren in einem Aktorsystem eine hierarchische Baumstruktur bilden.[3] Aktoren, die den gleichen Parent-Aktor haben, werden die Kinder dieses Aktors genannt.

Jeder Aktor hat einen Namen, den man als zusätzlichen Parameter beim Aufruf von der *actorOf*-Methode angeben kann oder der automatisch generiert wird, falls keiner angegeben wurde. Der „*guardian*"-Aktor hat immer den Namen „*user*". Aktoren, die den gleichen *Parent*-Aktor haben, müssen unterschiedliche Namen haben. Dadurch erreicht man, dass man Aktoren in einem Aktorsystem ähnlich wie in einem Unix-Filesystem eindeutig benennen kann, indem man ihre Namen durch Schrägstriche zu einem Aktor-Pfad verbindet. Zum Beispiel hätte der *Greeter*-Aktor aus unserem Beispiel den Pfad „*/user/greeter*" und der „*guardian*"-Aktor den Pfad „*/user*".[4]

Supervision

Wie auch Erlang benutzt Akka die hierarchische Struktur, um eine systematische Fehlerbehandlung zu ermöglichen. Tritt bei der Verarbeitung ein Fehler auf, d. h., es wird eine Exception erzeugt, die nicht im Aktor gefangen wird, dann wird die Verarbeitung der Nachrichten in der Mailbox des Aktors angehalten und der *Parent*-Aktor über die Exception informiert.[5] In seiner Konfiguration ist nun vorgegeben, wie darauf reagiert werden soll. Es gibt essentiell vier Möglichkeiten:

1. Die Exception kann ignoriert werden.
2. Der Aktor, in dem die Exception aufgetreten ist, kann neu gestartet werden.
3. Der Aktor, in dem die Exception aufgetreten ist, kann gestoppt werden.
4. Der Fehler kann weiter zum übergeordneten Aktor eskaliert werden.

Wird die Exception ignoriert, fährt der Aktor mit der Bearbeitung der Nachrichten in seiner Mailbox einfach fort.

Ein Neustart bedeutet, dass der betroffene Aktor und rekursiv alle Aktoren, die in der Hierarchie unter ihm sind, gestoppt werden und der betroffene Aktor durch eine neue erzeugte Instanz ersetzt wird, die anhand des ursprünglich verwendeten Props-Objekts erzeugt wird.[6]

Im dritten Fall werden der betroffene Aktor und rekursiv alle Aktoren, die in der Hierarchie unter ihm sind, gestoppt, ohne dass der gestoppte Aktor durch eine neue Instanz ersetzt wird. Alle Nachrichten in

3 Auch der „*guardian*"-Aktor hat aus technischen Gründen noch einen übergeordneten richtigen Aktor, der „*root guardian*"-Aktor. Auch dieser benötigt noch eine übergeordnete Instanz, die aber kein richtiger Aktor mehr ist und als „the one who walks the bubbles of space-time" genannt wird.

4 Der „*root guardian*"-Aktor hat den Pfad „*/*".

5 Der Parent-Aktor ist nicht unbedingt der Aktor, der die Nachricht gesendet hat, die zu dem Fehler geführt hat.

6 Das heißt, der aktuell vorhandene State des Aktors wird verworfen.

der zugehörigen Mailbox werden in die systemweite *DeadLetter*-Mailbox umgeleitet. Ebenfalls werden weitere Nachrichten, die an den gestoppten Aktor geschickt werden, ohne Fehlermeldung an die *DeadLetter*-Maibox umgeleitet.

Damit nun Aktoren im dritten Fall nicht einfach von der Anwendung unbemerkt sterben und mehr und mehr Nachrichten ins Nirvana laufen, bietet Akka eine Überwachungsmöglichkeit. Der Aktor-Kontext hat eine *watch*-Methode, mit der ein Aktor sein Interesse bekunden kann, von Akka informiert zu werden, wenn der angegebene Aktor gestoppt wird. Die Information wird per Nachricht vom Typ *akka.actor.Terminated* mitgeteilt, auf die der interessierte Aktor wie gewohnt hören und dann reagieren kann.

Man kann die Supervision-Strategie eines Aktors konfigurieren, indem man die Methode *AbstractActor.supervisorStrategy* überschreibt:

```
@Override
public SupervisorStrategy supervisorStrategy() {
  return new OneForOneStrategy(
    DeciderBuilder.match(
      DbConnectionBrokenException.class,
      e -> SupervisorStrategy.restart()
    )
    .match(
      DbNodeDownException.class,
       e -> SupervisorStrategy.stop()
    )
    .match(
      IllegalArgumentException.class,
      e->SupervisorStrategy.resume()
    )
    .build()
  );
}
```

Exceptions, für die keine Angaben gemacht werden, werden automatisch eskaliert. Wird die *supervisorStrategy*-Methode nicht überschrieben, dann greift eine Default-Strategie, die essentiell bei allen Fehlern, die vom Typ *java.lang.Exception* sind, den betroffenen Aktor neu startet und in allen anderen Fällen[7] den Fehler eskaliert. Fehler, die vom „guardian"-Aktor eskaliert werden, führen dazu, dass das Aktorsystem beendet wird.[8]

7 Insbesondere sind dies alle Fehler vom Typ JAVA.LANG.ERROR.

8 Neben der ONEFORONESTRATEGY gibt es auch noch eine ALLFORONESTRATEGY, die die Fehlerbehandlungsentscheidung auf alle Geschwister-Aktoren ausdehnt.

Durch dieses hierarchische Überwachungskonzept wird es möglich, die Geschäftslogik sauber von der Fehlerbehandlungslogik zu trennen und feingranular auf unterschiedliche Fehlersituationen zu reagieren.

Erweiterungen von Akka

Die Funktionalität von Akka ist in unterschiedlichen Modulen und Bibliotheken zusammengefasst, die die Kern-Funktionalität, die durch das Artefakt *akka-actor* bereitgestellt wird, erweitert.

Akka-Systeme auf unterschiedlichen Rechnern können miteinander kommunizieren. Die nötige Funktionalität wird von dem Artefakt *akka-remote* bereitgestellt, welches bei entsprechender Konfiguration eine transparente Kommunikation über TCP ermöglicht. Entweder können Aktoren von einem Aktorsystem *remote* auf einem anderen Aktorsystem gestartet werden oder bestehende Aktoren können mit Hilfe der Methode *actorSelection* des Aktor-Kontexts adressiert und dann angesprochen werden, ähnlich zu:

```
ActorSelection selection = context
  .actorSelection(
    „akka.tcp://app@10.0.0.1:2552"
      + „/user/serviceA/worker"
  );
selection.tell(„Hallo ServiceA-Worker!", getSelf());
```

Basierend auf dieser *Remoting*-Funktionalität gibt es dann Module, die es ermöglichen, Aufgaben in Cluster von Aktorsystemen zu bilden, um auf ihnen unterschiedliche Aufgaben bearbeiten zu lassen.

Ein weiteres Akka-Modul unterstützt die Verarbeitung von Daten in Streams. Auf diesem Modul baut dann *akka-http* auf, welches die Erstellung von HTTP-Endpunkten erlaubt. Des Weiteren sei noch das Modul *akka-persistence* erwähnt, welches über Event Sourcing-Aktoren bereitstellt, die ihren Status persistent halten können, d. h., die nach einem Neustart ihren letzten Status wieder automatisch rekonstruieren können.

Zum Abschluss

Dieser Artikel konnte das Thema Aktormodel und Akka nur ganz oberflächlich einführen und die Möglichkeiten, die Akka bietet, nur schemenhaft anreißen. Auf der Homepage von Akka [1] gibt es eine umfangreiche Dokumentation, die fast parallel für Scala und Java gehalten ist. Des Weiteren gibt es unterschiedliche Bücher, zum Beispiel „*Akka in action*" [15], die

einen tieferen Einblick in Akka und das Aktormodel bieten. Gerade das Programmiermodel erfordert ein Umdenken, welches durch seine hochgradige Asynchronität auf den ersten Blick ungewohnt ist. Aber die erfolgreichen Projekte mit Erlang oder Akka haben gezeigt, dass es sich lohnt, sich mit diesem Programmiermodel zu beschäftigen, um skalierbare und fehlertolerante Applikationen zu entwickeln.

Referenzen

[1] Akka
https://www.akka.io

[2] Hewitt, Carl; Bishop, Peter; Steiger, Richard *A universal modular ACTOR formalism for artificial intelligence*, Proceeding, IJCAI, 1973

[3] Hewitt, Carl; Meijer, Erik; Szyperski, Clemens
https://channel9.msdn.com/Shows/Going+Deep/Hewitt-Meijer-and-Szyperski-The-Actor-Model-everything-you-wanted-to-know-but-were-afraid-to-ask

[4] Erlang *Programming Language*
https://www.erlang.org

[5] Armstrong, Joe *Making reliable distributed systems in the presence of software errors*, PhD Thesis, 2003

[6] Letuchy.Eugene *Facebook Chat*,
https://www.facebook.com/note.php?note_id=14218138919

[7] RabbitMQ, https://www.rabbitmq.com

[8] Appache *CouchDB*:
http://couchdb.apache.org

[9] Dougherty, Rich *Early Akka History*,
https://groups.google.com/forum/#!topic/akka-user/1zXAKgOn5UU, 2013

[10] Bonér, Jonas *Introducing Akka – Simpler scalability, fault-tolerance, concurrency & remoting through actors*,
http://jonasboner.com/introducing-akka, 2010

[11] Jovanovic, Vojin; Haller, Phillip
http://docs.scala-lang.org/overviews/core/actors-migration-guide.html

[12] Apache *Gearpump*
https://github.com/apache/incubator-gearpump

[13] Apache *Flink*
https://cwiki.apache.org/confluence/display/FLINK/Akka+and+Actors

[14] Lagom Framework
https://www.lagomframework.com

[15] Roestenburg, Raymond; Bakker, Rob; Williams, Rob *Akka in action*, Manning, 2017

Kurzbiografie

Dr. Martin Fluch, Mathematiker, ist seit März 2015 als Software-Entwickler für MATHEMA Software GmbH in Erlangen tätig. Seine Themenschwerpunkte umfassen die Java Standard und Enterprise Edition (JavaSE und JavaEE). Dabei interessieren ihn besonders die Abstraktion von Lösungskonzepten in Architekturmuster und deren Anwendung.

Herbstcampus

Wissenstransfer par excellence

4.– 6. September 2018 in Nürnberg

„Partageons ce qui nous départage“

Können wir zusammen, ohne Finanzierung, ohne Auftrag und in drei Monaten, ein Buch über unsere Firmenkultur schreiben?

Eine Buchrezension von Tim Bourguignon

Vier Mal war die Firma Octo Technology [1] auf dem Podium der „*Great Place To Work*“-Awards [2] in Frankreich und das zwei Mal als Sieger. Seit 2011 wurde diese französische Beratungsfirma mehrmals für ihre angenehme Arbeitsatmosphäre, ihre Kultur und ihr Flair geehrt. Gerne wäre man eine Maus, um diese Firma von innen erleben zu dürfen. Das ist aber nun nicht mehr nötig: Ende 2010 publizierten die Berater der Firma ihr eigenes Buch mit dem Titel „*Partageons ce qui nous departage*“ [3], zu Deutsch: „*Teilen wir, was uns unterscheidet*“.

Dieses Buch ist als Wette auf die Welt gekommen: „Können wir zusammen (20 Autoren) ohne Finanzierung, ohne Auftrag und in 3 Monaten, ein Buch über unsere Firmenkultur schreiben?“ Wette gewonnen!

Da dieses Buch schon längst ausverkauft ist und nie ins Deutsche übersetzt werden wird, fühle ich mich verpflichtet, eine Zusammenfassung zu schreiben. Was macht Octo denn nun anders?

Übergreifender Austausch

Octo legt einen großen Wert auf Austausch und Retrospektive. Viele Berater sind nahezu 100 Prozent beim Kunden, jedoch gibt es viele übergreifende Tätigkeiten und Hilfsbereitschaft untereinander. Dabei setzen sie auf die folgenden drei Werkzeuge: das „*Perfection Game*“, „*360°-Feedback*“ und „*Trainingsräume*“.

1. Durch das „*Perfection Game*“ helfen sich Kollegen, ihre Werke zu verbessern. Über jedes „*Deliverable*“ wird iteriert, ohne Ausnahme. Kollegen reviewen sich gegenseitig und bewerten die Arbeit der Kollegen. Dabei ist Positivität das Motto und die Bewertungen werden nicht weiter kommentiert: „Ich fand X und Y gut, deshalb gebe ich dir die Note 2,0. Wenn A und B auch drin gewesen wären, hättest du von mir eine 1,0 bekommen“.
2. Insbesondere im Leadership und Management wird ein 360°-Feedback eingesetzt. Das 360°-Feedback ist eine Methode zur Einschätzung der Kompetenzen und Leistungen aus unterschiedlichen Perspektiven, wie z. B. aus dem Blickwinkel der Mitarbeiter, Vorgesetzten, Kollegen, Teammitglieder und Kunden.
3. Die Octo-Mitarbeiter organisieren „*Mastermind-Treffen*“, auch „*Trainingsräume*“ genannt. Dabei trifft sich regelmäßig eine Gruppe von Leuten mit einem ähnlichen Profil, um sich über die eigenen Probleme auszutauschen und sich gegenseitig zu beraten. Dies entspricht nach Ansicht des Autors der Definition der „*Mastermind-Gruppe*“ [4].

Kultur und Mindset

Bei Octo wird zudem großer Wert auf Kultur und Mindset gelegt. Die Firma hat mittlerweile mehrere hundert Mitarbeiter, verhält sich aber immer noch wie eine kleine Familie.

4. Als erstes Gesetz gilt: „Kollegen kommen zuerst, dann die Kunden“. Die Kollegen sind das Allerwichtigste. Wenn es einem Kollegen beim Kunden nicht gut geht, er sich langweilt oder nichts mehr lernt, steigt die Firma auf die „Barrikaden“ und sorgt für

Abwechslung. Kundenprojekte werden nicht einfach akzeptiert, weil es G€ld bringt, sondern weil es die wichtigsten Ressourcen der Firma voranbringt.

5. Eine große Bedeutung hat das Empowerment im Unternehmen. Wenn jemand eine Idee hat, dann wird er oder sie „empowert“, es selbst umzusetzen. Nur so bekommen Aktionen mehr Gewicht als Wörter.

6. Der beste Weg zu lernen ist, jemandem über die Schultern zu schauen und dabei Multiplikatoren zu generieren. Deshalb wird Wissen nicht vorrangig via Dokumente akquiriert, sondern via Beobachtung und Coaching. In Workshops und Trainings ist z. B. so gut wie immer ein „Lehrling“ dabei.

7. *One-on-ones:* Feedback ist für die Octo-Mitarbeiter sehr wichtig. One-on-one-Meetings mit den Managern sind gesetzt: 30 Minuten pro Woche, jede Woche. Dort wird sich sowohl über die aktuelle Mission ausgetauscht als auch über das, was in der eigenen Firma passiert.

8. *Vertrauen* wird ebenfalls sehr großgeschrieben. Octo hatte 2011 (Erstellungszeitpunkt des Buches) 140 Mitarbeiter. Fünf Jahre später sind es 377 gewesen. 2011 hatte Octo noch keine Zugriffsbeschränkungen im internen Customer-Relationship-Management-Tool. Jeder durfte alles sehen. Wer was macht, wieviel pro Mission abgerechnet wurde etc. Allerdings waren die Gehälter noch geheim. Wie dies heutzutage aussieht, ließ sich nicht mehr herausfinden.

Arbeitstipps

Das Buch gibt auch praktische Tipps. Es scheint, als ob dieses Werk ein Teil der Einstellungslektüre für neue Mitarbeiter wäre... (was es übrigens bestimmt ist).

9. Gar nicht erst zu fragen oder ein „Nein“ als Antwort auf eine Frage zu erhalten, erzielen das gleiche Ergebnis. Doch im Gegensatz zum Nichtfragen birgt eine Frage die Möglichkeit auf eine positive Antwort. Mit anderen Worten: Frechheit lohnt sich und wird oft belohnt. Somit hat Octo Neil Armstrong selbst für eine interne Konferenz nach Paris einladen können. Bei Octo lebt der Spruch: *„Ein „Nein“ als Antwort ist schon gewonnen“*.

10. *Meeting-Tools*:

- Könnt ihr ein Bild als Protokoll schicken? So erklären die Octo-Mitarbeiter ein gutes Meeting: Wenn das Protokoll live entsteht, dann am besten auf dem Flipchart, an der Wand oder dem Whiteboard. Alles andere lässt Interpretationsspielraum, was ein eher schlechteres Ergebnis liefert.
- Um zu vermeiden, dass ein Termin ohne weitere Aktionen endet, sind *„Actionables“* ein Muss nach jeder Diskussion. Dies können auch Mini-Actionnables sein, aber es muss eins geben.
- Das *„Intention Check“*-Spiel ist allen Mitarbeitern bekannt. In jeder Diskussion kann jederzeit jemand eine *„Willenserklärung“* abfragen. Dann muss der andere erklären, wo er hin will. Dies verhindert, dass Diskussionen ab und zu vom Thema abdriften, dass sich jemand in der Hitze des Moments von seinen Emotionen überrollen lässt oder dass eine Diskussion aus der Bahn läuft.
- Das *„Return On Time Invested“* wird nach jedem Meeting analysiert. Jeder Teilnehmer darf von -2 bis +2 eine Bewertung abgeben. Eine Note im negativen Bereich weist darauf hin, dass der Teilnehmer lieber etwas anderes gemacht hätte. 0 ist der Break-Even-Point. Bei Bewertungen im positiven Bereich war die Zeit also gut investiert.
- Auch ein *„Spaß-o-Meter“* wird eingesetzt. Ziel ist es, niedrigere Noten so früh wie möglich zu identifizieren und anzugehen, weil glückliche Mitarbeiter immer effektiver und effizienter sind.

Wissen konsolidieren

Bei Octo wird viel Wissen über Mailing-Lists, Wikis, Chats etc. ausgetauscht. Dies wirkt sich als Konsolidierung für deren Blog oder Bücher aus.

Projekte, wie dieses Buch zu schreiben, werden auch als „normale“ Projekte behandelt. Mit Zielen, Sichtbarkeit, Aufmerksamkeit und Interesse bis hoch zum Management.

Octo hat mehrere Bücher wie dieses publiziert und alle Mitarbeiter schreiben regelmäßig auf dem Firmenblog. Die Mehrheit ihrer Kundenakquise entstammt mittlerweile durch diese Offenheit mit Informationen. Aus ihrer Sicht lohnt es sich definitiv, viel zu „geben“.

Fazit

Insgesamt gibt es 44 größere oder kleinere Tipps, die so getauscht werden. Wenn ich nur eins mitnehmen dürfte/könnte, wäre es das *„Perfektion Game“*. Ich bin ganz klar schon ein Spieler dieses Spiels, aber ich sehe es immer noch zu selten in unseren Firmen institutionalisiert.

Ich habe mich persönlich mit Consultants von Octo unterhalten, wobei mittlerweile neue Werkzeuge, wie zum Beispiel die 20 Prozent „*Innovationszeit*" von Google oder die freie Buchbestellungs-Politik (jeder darf ein Buch bestellen, ohne nachfragen zu müssen), eingesetzt werden. Damit erreicht Octo seit Jahren 20 Prozent Wachstum und 10 Prozent Gewinn pro Jahr.

Seitdem hat Octo auch andere Bücher publiziert, unter anderem „*Culture Code*" [5], ein Buch über die Rolle des Tech Leads. Ich finde es unglaublich interessant, wie diese Firma ihr Know-how damit konsolidiert. Wäre auch Ihre Firma in der Lage, so etwas zu produzieren? Und wenn nicht, warum?

Was nehmen Sie von dieser Liste mit? Was fehlt in der Liste? Mit welchen Mitteln würden Sie Ihre Firmen am besten ändern? Jetzt sind Sie dran.

Referenzen

[1] Octo *Octo Technology*
http://www.octo.com/en

[2] Great places to work
http://www.greatplacetowork.net

[3] Octo *Partageons ce qui nous départage*
http://www.octo.com/fr/publications/10-partageons-ce-qui-nous-departage

[4] wikiwand *Mastermind Gruppe*
https://www.wikiwand.com/en/Mastermind_group

[5] Octo *Culture Code*
http://www.octo.com/fr/publications/20-culture-code

Kurzbiografie

Timothée Bourguignon ist als Chief Learning Officer & Agile Coach für MATHEMA Software GmbH tätig. Sein Spezialgebiet ist die Anwendungsprogrammierung mit dem .NET-Framework. Daneben beschäftigt er sich mit den Neuerungen der .NET-Welt und deren Communities. Als Agilist, Scrum Master und Scrum Coach liegt sein Schwerpunkt auf agilen Methoden, guter Teamarbeit und hoher Arbeitsqualität. Hierbei setzt er insbesondere auf die Philosophie des Software-Craftsmanships.

Herbstcampus

Wissenstransfer par excellence

4.– 6. September 2018 in Nürnberg

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch? Dann geben Sie uns doch bitte Bescheid.
Bookware, Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0, Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

Deutschland

JUG Berlin Brandenburg
Kontakt: Oliver B. Fischer
(o.b.fischer@swe-blog.net)
http://jug-berlin-brandenburg.de/kontakt.html

Java UserGroup Bremen
Kontakt: Rabea Gransberger (rgransberger@gmx.de)
http://www.jugbremen.de

JUG DA
Java User Group Darmstadt
Kontakt: jug-da-orga@googlegroups.com
http://www.jug-da.de

Java User Group Saxony
Java User Group Dresden
Kontakt: Falk Hartmann
(falk.hartmann@jugsaxony.org)
http://www.jugsaxony.de

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Kontakt: Heiko Sippel (info@rheinjug.de)
http://www.rheinjug.de

JUG Deutschland e.V.
Java User Group Deutschland e.V.
Kontakt: Stefan Koospal (office@java.de)
http://www.java.de

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Kontakt: Heiko Sippel (heiko.sippel@ruhrjug.de)
http://www.ruhrjug.de

JUGF
Java User Group Frankfurt
Kontakt: Alexander Culum (alexander.culum@web.de)
http://www.jugf.de

JUG Görlitz
Java User Group Görlitz
Kontakt: (kontakt@jug-gr.de)
http://www.jug-gr.de

JUG Hamburg
Java User Group Hamburg
http://www.jughh.org

JUG Karlsruhe
Java User Group Karlsruhe
Kontakt: (jugkarlsruhe@gmail.com)
http://jug-karlsruhe.de

JUGC
Java User Group Köln
Kontakt: Michael Hüttermann
(michael@huettermann.net)
http://www.jugcologne.org

jugm
Java User Group München
Kontakt: Andreas Haug (ah@jugm.de)
http://www.jugm.de

JUG Münster
Java User Group für Münster und das Münsterland
Kontakt: Thomas Kruse (tkjugi@sforce.org)
http://www.jug-muenster.de

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
Kontakt: (info@jug-n.de)
http://www.jug-n.de

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

JUGS e.V.
Java User Group Stuttgart e.V. , c/o Dr. Michael Paus
Kontakt: Dr. Micheal Paus (mp@jugs.org),
Hagen Stanek (hs@jugs.org),
Rainer Anglett (ra@jugs.org)
http://www.jugs.org

Schweiz

JUGS
Java User Group Switzerland
http://www.jug.ch (info@jug.ch)

.NET User Groups

Deutschland

.NET User Group Bonn
.NET User Group "Bonn-to-Code.Net"
Kontakt: Roland Weigelt (mail@bonn-to-code.net)
http://www.bonn-to-code.net

.NET User Group Dortmund (Do.NET)
c/o BROCKHAUS AG
http://do-dotnet.de
Kontakt: Paul Mizel (pmizel@do-dotnet.de)

Die Dodnedder
.NET User Group Franken
Kontakt: Udo Neßhöver, Ulrike Stirnweiß
(info@dodnedder.de)
http://www.dodnedder.de

.NET UserGroup Frankfurt
http://www.dotnet-usergroup.de

.NET User Group Friedrichshafen
Kontakt: Tobias Allweier (info@dotnet-fn.de)
http://www.dotnet-fn.de

.NET User Group Hannover
http://www.dnug-hannover.de
Kontakt: (dnug@indisoftware.de)

INdotNET
Ingolstädter .NET Developers Group
Kontakt: Gregor Biswanger
(gregor.biswanger@web-enliven.de)
http://www.indot.net

DNUG-Köln
DotNetUserGroup Köln
Kontakt: Albert Weinert (info@der-albert.com)
http://www.dnug-koeln.de

.NET User Group Leipzig
Kontakt: Alexander Groß (agross@dotnet-leipzig.de)
Torsten Weber (tweber@dotnet-leipzig.de)
http://www.dotnet-leipzig.de

.NET Developers Group München
Kontakt: Hardy Erlinger (hardy_erlinger@hotmail.com)
http://www.munichdot.net

.NET User Group Oldenburg
c/o Hilmar Bunjes und Yvette Teiken
Hilmar Bunjes (hilmar.bunjes@dotnet-oldenburg.de)
Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)
http://www.dotnet-oldenburg.de

.NET Developers Group Stuttgart
Kontakt: Michael Niethammer
(GroupLeader@devgroup-stuttgart.net)
http://www.devgroup-stuttgart.net

.NET Developer-Group Ulm
c/o artiso solutions GmbH
Kontakt: Thomas Schissler (tschissler@artiso.com)
http://www.dotnet-ulm.de

Österreich

.NET User Group Austria
c/o Global Knowledge Network GmbH,
Kontakt: Christian Nagel (ug@christiannagel.com)
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx

Software Craftsmanship Communities

Deutschland, Schweiz, Österreich

Softwerkskammer – Mehrere regionale Gruppen und Themengruppen unter einem Dach
Kontakt: Nicole Rauch (nicole.m@gmx.de)
http://www.softwerkskammer.org

Metropolregion Nürnberg
JUG

Die Java User Group
Metropolregion Nürnberg
trifft sich regelmäßig einmal im Monat.
Thema und Ort werden über
www.jug-n.de
bekannt gegeben.

Weitere Informationen
finden Sie unter:
www.jug-n.de

MATHEMA

meet the experts of enterprise infrastructure

- **Einführung in die Funktionale Programmierung**
  11. Dezember 2017, 950,– € (zzgl. 19 % MwSt.)
- **HTML5, CSS3 und JavaScript**
  22. Januar 2018, 25. Juni 2018, 1.650,– € (zzgl. 19 % MwSt.)
- **Entwicklung mobiler Anwendungen mit iOS**
  5. Februar 2018, 16. Juli 2018, 1.250,– € (zzgl. 19 % MwSt.)
- **Einführung in die objektorientierte Programmiersprache Java – Eine praxisnahe Einführung**
  12. Februar 2018, 9. Juli 2018, 2.150,– € (zzgl. 19 % MwSt.)
- **Enterprise JavaBeans (EJB) – Entwicklung von Geschäftslogik-Komponenten**
  12. März 2018, 1.650,– € (zzgl. 19 % MwSt.)
- **Scrum Basics**
  950,– € (zzgl. 19 % MwSt.)
- **Scrum im Großen** – Agiles Organisationsdesign nach LeSS, 950,– € (zzgl. 19 % MwSt.)

Lesen bildet. **Training macht fit.**

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

MATHEMA

„Die Herausforderung, jeden Tag etwas Neues zu lernen, habe ich gesucht und bei MATHEMA gefunden."

Tim Bourguignon, Senior Consultant

Wir sind ein Consulting-Unternehmen mit Schwerpunkt in der Entwicklung unternehmenskritischer, verteilter Systeme und Umsetzung von Service-orientierten Architekturen und Applikationen von Frontend bis Backend. Darüber hinaus ist uns der Wissenstransfer ein großes Anliegen: Wir verfügen über einen eigenen Trainingsbereich und unsere Consultants sind regelmäßig als Autoren in der Fachpresse sowie als Speaker auf zahlreichen Fachkonferenzen präsent.

join the experts of enterprise infrastructure

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de | Ansprechpartnerin:
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de | Britta Kolb

Das Allerletzte

Dies ist kein Scherz!
Diese Meldung wurde tatsächlich in der freien Wildbahn angetroffen.
Ist Ihnen auch schon einmal ein Exemplar dieser Gattung über den Weg gelaufen?
Dann scheuen Sie sich bitte nicht, uns das mitzuteilen.

Der nächste KaffeeKlatsch erscheint im Dezember.

Herbstcampus

Wissenstransfer par excellence

4. – 6. September 2018
in Nürnberg