KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

10/2014

Jahrgang 7

Editorial

Jubiläum

Der eine oder andere wird sich das vielleicht nicht vorstellen können, aber das World Wide Web (W3) hat inzwischen schon mehr als 25 Jahre auf dem Buckel. Das mag vielleicht nur deshalb so sein, weil es seit nunmehr 20 Jahren das World Wide Web Consortium (W3C) gibt, das schützend und mahnend seine Hand über das weltweite Wunder hält.

Die Idee, Informationen miteinander zu verknüpfen, ist mindestens so alt, wie die Erfindung des Katalogs. Aber erst mit der Veröffentlichung des „World Wide Webs" von Robert Cailliau und Tim Berners-Lee wurde aus der Idee eine weltumspannende Wirklichkeit. Allerdings befürchtete Berners-Lee, dass mit der rasanten Akzeptanz des Webs möglicherweise auch eine Aufspaltung einhergehen könnte, die etwa kommerzielle Inhalte von akademischen trennen könnte.

So wurde im Oktober 1994 das World Wide Web Consortium ins Leben gerufen, das die Aufgabe haben sollte, durch Standardisierung eine Diversifizierung des Webs zu verhindern. Ob dies dem Konsortium gelungen ist, sei mal dahingestellt. Aber das Konsortium hat es auf jeden Fall geschafft, das Bewusstsein dafür zu schaffen, dass nur eine einzige Lösung die Vielfalt des Webs garantieren kann.

> „Today we think nothing of watching video and audio natively in the browser, and nothing of running a browser on a phone. We expect to be able to share photos, shop, read the news, and look up information anywhere, on any device. Though they remain invisible to most users, HTML5 and the Open Web Platform are driving these growing user expectations."
>
> Tim Berners-Lee, W3C Director

In den zwanzig Jahren, die am 29. Oktober 2014 gebührend gefeiert wurden [1], hat das W3C vielleicht nicht so prickelnde Innovationen hervorgebracht, wie sich der eine oder andere das vielleicht gewünscht haben mag. Aber die vom W3C gepflegte Basis macht es möglich, dass man inzwischen auf fast jedem Gerät, praktisch einfach so, Musik hören, Videos anschauen, soziale Kontakte pflegen und über Gott und die Welt etwas lesen kann.

Dabei hat es das Konsortium geschafft, dass alle großen Browser-Hersteller an einem Strang ziehen und das Web als „patentfreien" Raum etabliert haben, der sowohl Entwicklern als auch Nutzern einen kostenfreien Zugang zu Applikationen und Informationen aller Art gewährt. Eine beachtliche Leistung, die Erstaunliches wie die Wikipedia, Google und Facebook hervorgebracht hat.

So kann das W3C trotz aller Kritik zurecht mit Stolz auf zwanzig ereignisreiche Jahre zurückblicken. Herzlichen Glückwunsch!

Referenzen

[1] W3C *W3C20 Anniversary Symposium – The Future of the Web* http://www.w3.org/20

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 Frau Natalia Wilhelm via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-16 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 Natalia Wilhelm via E-Mail unter natalia.wilhelm@bookware.de oder telefonisch unter 09131/8903-16.

Copyright

Impressum

KaffeeKlatsch Jahrgang 7, Nummer 10, Oktober 2014
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking
Anzeigen: Natalia Wilhelm
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

MongoDB & Söhne

Die populäre NoSQL-Datenbank und ihre Anbindung an Java 6

von Thomas Bertz

Möchten Sie endlich einmal mit einer *NoSQL*-Datenbank arbeiten? Vielleicht mit *MongoDB*? Und fragen Sie sich, wie Sie diese mit Java ansprechen und in Ihre bestehenden Programme integrieren können? Dieser Artikel soll einen ersten Überblick geben, als Trockenübungen die CRUD-Operationen in MongoDB vorführen und anschließend mit dem Object-Document Mapper (ODM) *Morphia* Lust auf mehr machen.

Java JSON Reloaded

Ein neugieriger Blick auf den JSR 353 9

von Thomas Künneth

JSON ist ein überaus populäres, leichtgewichtiges und sprachunabhängiges Datenaustauschformat. Sein Name ist ein Akronym und bedeutet *JavaScript Object Notation*. Ursprünglich also aus JavaScript entstanden, ist das Erzeugen und Interpretieren solcher Objekte mittlerweile in praktisch jeder modernen Programmiersprache möglich.

Kuchen für Alle!

Über die JavaScript-Bibliothek D3 zur Visualisierung von Daten 13

von Sandy Stehr

Es gibt einige *JavaScript*-Bibliotheken für Torten- und Balkendiagramme auf dem Markt, wie beispielsweise *googleapi*, *Processing* oder *Raphaël*. Als *D3* 2011 herauskam, wurde schnell klar, dass es ein mächtiges Werkzeug sein würde, um Datenvisualisierungen im Web zu erstellen. Doch warum wurde es so erfolgreich, wobei es doch nicht das erste und auch nicht das einzige Werkzeug auf dem Markt ist?

MongoDB & Söhne

Die populäre NoSQL-Datenbank und ihre Anbindung an Java

von Thomas Bertz

Möchten Sie endlich einmal mit einer *NoSQL*-Datenbank arbeiten? Vielleicht mit *MongoDB*? Und fragen Sie sich, wie Sie diese mit Java ansprechen und in Ihre bestehenden Programme integrieren können? Dieser Artikel soll einen ersten Überblick geben, als Trockenübungen die CRUD-Operationen in MongoDB vorführen und anschließend mit dem Object-Document Mapper (ODM) *Morphia* Lust auf mehr machen.

Wieso, weshalb, warum?

2009 veröffentlicht, 2010 im Technologie-Radar von Thoughtworks unter *Trial* (ausprobieren!) und 2013 unter *Adopt* (verwenden!) gelistet, ist MongoDB [1] nach eigenen Angaben die bekannteste und gefragteste NoSQL-Datenbank. Die Installation ist kinderleicht und schnell, und der Zugriff aus Java heraus ist auch kein Problem. Die herausragendste Eigenschaft von MongoDB ist sicherlich die Schemafreiheit. Noch eine ganze Reihe weiterer toller Eigenschaften, welche die Erschaffer zu dem sich aus dem englischen Wort „humongous“ (gigantisch) ableitenden Namen verführt haben, kann man auf der Homepage nachlesen, als da wären: *Map/Reduce*, *Auto-Sharding*, *GridFS* und *Fast In-Place Updates* (Liste nicht vollständig).

Installation, Umgebung und erste Schritte

MongoDB gibt es für jedes gängige Betriebssystem. Nach dem Herunterladen und Installieren der circa 130 MiB großen Datei findet man im Installationsverzeichnis mehrere ausführbare Dateien, wobei anfänglich für uns zwei interessant sein dürften. Zum einen starten wir *mongod*. Dies ist der *Database Management System* (*DBMS*)-Prozess, der den Datenbankdienst zur Verfügung stellt und standardmäßig im Hintergrund auf Port 27017 auf Client-Anfragen lauscht. Beim ersten Start wird sich dieser Prozess höchstwahrscheinlich mit folgendem Fehler beenden.

ERROR: dbpath (\data\db\) does not exist

Damit hat er im allgemeinen Fall Recht, und so legen wir diesen Ordner, in dem die internen Verwaltungsdateien der Datenbank gehalten werden, an und versuchen es – diesmal mit Erfolg – erneut.

Nun können wir mit der zweiten wichtigen Datei *mongo* einen weiteren Prozess starten. Dieser realisiert eine *JavaScript-Shell*, die sich mit dem DBMS-Prozess verbindet und unsere Befehle auf der Kommandozeile entgegennimmt. Wie von relationalen Datenbanken bekannt, lassen sich mit MongoDB mehrere logische Datenbanken einrichten und verwalten. Dies bewerkstelligt man einfach mit *use mydb*. Damit wechselt der Kontext zur Datenbank *mydb* und diese wird auch gleich angelegt, falls sie noch nicht existieren sollte. Alle weiteren Befehle werden ab hier mit dem Präfix *db.* begonnen. Um sich den aktuellen Datenbankkontext anzeigen zu lassen, verwendet man schlicht *db*.

Wie sehen nun die Dokumente aus, die man in MongoDB speichern kann? Und wie steht es mit den CRUDs (*Create, Read, Update, Delete*), also den klassischen Datenbankoperationen (Erzeugen, Lesen, Ändern, Löschen)?

Dokumente sind für MongoDB als *JSON*-Objekte repräsentierbar. Ein simples Dokument, das eine Person beschreibt, könnte folgendermaßen aussehen.

```
p = { vorname: "Donald", nachname: "Knuth", alter: 76}
```

Dieses Dokument kann man mit dem folgenden Befehl in die Datenbank einfügen (CREATE).

```
db.persons.insert(p)
```

Dabei bezeichnet *persons* die *Collection* (Sammlung), in der das Dokument abgelegt wird. Collections entsprechen in etwa den Tabellen bei relationalen Datenbanken mit dem Unterschied, dass Collections bei MongoDB schemafrei sind. Und wie man außerdem erkennen kann, werden Collections *lazy* erzeugt, also erst dann, wenn sie wie hier beispielsweise für ein *insert*-Statement benötigt werden.

Ein Beispiel für eine READ-Operation zum Finden aller Personen, die älter als 70 Jahre sind:

```
db.persons.find({alter: {$gt: 70}})
```

Ein Beispiel für eine simple UPDATE-Operation (hier wird Donald E. Knuths [2] Vorname erweitert):

```
db.persons.update({alter: 76}, {$set: {vorname: "Donald E."}})
```

Und ein Beispiel für eine DELETE-Operation (löscht alle Dokumente, deren Attribut „alter“ 76 ist):

```
db.persons.remove({alter: 76})
```

Anschluss an Java

Wenn man erst einmal verstanden hat, was Dokumente sind und wie mit MongoDB umzugehen ist, dann ist die Abfrage mit Java nicht schwer. Man nimmt dazu den *Mongo-Java*-Treiber in den *BuildPath* auf. Der Treiber ist auf der Dokumentationsseite [3] von MongoDB referenziert. Verbindung zur Datenbank baut man mit den folgenden Zeilen auf.

```
MongoClient client;
client = new MongoClient("localhost");
DB db = mongoClient.getDB("mydb");
DBCollection personsColl = db.getCollection("persons");
```

Ein Beispiel der eigentlichen *find*-Abfrage sähe etwa wie folgt aus.

```
BasicDBObject query = new BasicDBObject();
DBCursor cursor = personsColl.find(query);
try {
  while (cursor.hasNext()) {
    System.out.println(cursor.next());
  }
} finally {
  cursor.close();
}
```

So weit, so gut. Was aber, wenn die Inhalte des Cursors nicht einfach nur ausgegeben, sondern – wir sind ja schließlich in Java – typsicher weiterverarbeitet werden sollen? Schaut man sich das *Interface* von *DBObject* an (denn das gibt *cursor.next()* zurück), erkennt man, dass elf Methoden zu implementieren wären. Das will man nicht für jede seiner Fachklassen tun. Glücklicherweise gibt es bereits einige Bibliotheken, die genau das für uns tun. Und heraus kommt ein *Object-Document Mapper* (ODM), also ein ähnliches Konzept, wie man es von relationalen Datenbanken als *Object-Relational Mapper* (ORM) kennt. Für den vorliegenden Artikel wurde *Morphia* [4] gewählt.

Morphia – ein ODM zwischen Java und MongoDB

Zunächst muss wieder die entsprechende Bibliothek in den *BuildPath* aufgenommen werden. Die *groupId* lautet *org.mongodb.morphia* und die *artifactId* lautet *morphia*. Sie ist über das zentrale *Maven Repository* [5] erhältlich.

Die Verbindung zur Datenbank und das Erzeugen eines „Mapping-Kontextes“, wie man es von ORMs her kennt, realisieren die folgenden Zeilen.

```
MongoClient client;
client = new MongoClient("localhost");
Morphia morphia = new Morphia();
Datastore ds = morphia.createDatastore(mongoClient, "mydb");
morphia.map(Person.class);
```

Und das eigentliche Persistieren eines Fachobjekts beschränkt sich dann auf den Ausdruck:

```
Key<Person> k = ds.save(new Person("Donald", "Knuth", 76));
```

Mit Morphia bleiben die Fachobjekte sauber, übersichtlich und typsicher. Das Mapping einer Fachklasse wird mit der Annotation *@Entity* analog zu *JPA/Hibernate* weitestgehend versteckt, wie das folgende Listing beweist.

```
@Entity("persons")
public class Person {
  @Id ObjectId id;
  private String vorname = "";
  private String nachname = "";
  private int alter;

  /*constructors, getters, setters*/
}
```

Zum Schluss sei noch ein Beispiel mit allen CRUD-Operationen mit Morphia als *JUnit*-Test gezeigt:

```
@Before
public void setup() {
  ds.delete(ds.createQuery(Person.class));
   assertEquals(0, personsColl.count());
}

@Test
public void testCRUD() {
  Person personExpected = new Person(
    "Donald", "Knuth", 76
  );
  Key<Person> k = ds.save(personExpected);
  assertEquals(1, personsColl.count());
  Person personActual =
    ds.find(Person.class).field("_id").equal(k.getId()).get();
  assertEquals(personExpected, personActual);
```

```
        UpdateResults ur =
            ds.update(personExpected, ds.createUpdateOperations(
                Person.class).set("vorname", "Donald E.")
            );
        assertEquals(1, ur.getUpdatedCount());
        personActual = ds.find(Person.class).field("_id").equal(
            k.getId()
        ).get();
        assertEquals("Donald E.", personActual.getVorname());
        ds.delete(personActual);
        assertEquals(0, personsColl.count());
    }
```

Fazit und Ausblick

MongoDB imponiert als eine schnell installier- und einsetzbare, dokumentenorientierte Datenbank. Dieses Konzept zieht sich durch, was daran bemerkbar ist, dass Collections vor Verwendung nicht angelegt werden müssen und MongoDB schemafrei ist. Das Konzept ist leicht erlernbar.

Mit Morphia ist eine komfortable Bibliothek gelungen, welche die Beta-Phase zwar noch nicht hinter sich gelassen hat, die aber zeigt, wie ein leichtgewichtiger ODM aussehen kann. Transaktionssteuerung fehlt leider komplett (wie gesagt: eine 0 steht noch in der Major-Version). Wer das braucht, der sollte sich z. B. *MongoLink* [6] ansehen.

Hier wurden nur einige Eigenschaften erwähnt, mit denen sich MongoDB auf ihrer Homepage brüstet. Lohnenswert wäre eine weitere Untersuchung einiger ihrer Funktionalitäten. Fest steht: Die breite Akzeptanz ist Zeichen ihres Erfolgs, auch wenn Donald E. Knuth das vielleicht anders sieht [7].

Referenzen

[1] MongoDB *Homepage*, https://www.mongodb.org

[2] Wikipedia *Donald E. Knuth*, http://de.wikipedia.org/wiki/Donald_E._Knuth

[3] MongoDB *Java MongoDB Driver*, http://docs.mongodb.org/ecosystem/drivers/java

[4] GitHub *Morphia – Leichtgewichtiger ODM von Java-Objekten zu MongoDB-Dokumenten*, https://github.com/mongodb/morphia/wiki

[5] Maven *The Central Repository*, http://search.maven.org

[6] MongoLink *An Object-Document Mapper (ODM) for Java and MongoDB*, http://mongolink.org

[7] youtube *My advice to young people – Donald Knuth [video]*, http://www.youtube.com/watch?v=75Ju0eM5T2c

Kurzbiografie

Thomas Bertz ist Senior Consultant bei der MATHEMA Software GmbH. Bisher war er als *Build-* und Konfigurationsmanager, Testmanager und Software-Entwickler in den unterschiedlichsten Branchen tätig. In der Vergangenheit hat er hauptsächlich mit C++ und Datenbanken gearbeitet und unterstützt heute Kunden bei der Konzeption und Realisierung von Software-Lösungen im Java Enterprise-Umfeld.

Java JSON Reloaded

Ein neugieriger Blick auf den JSR 353

von Thomas Künneth

JSON ist ein überaus populäres, leichtgewichtiges und sprachunabhängiges Datenaustauschformat. Sein Name ist ein Akronym und bedeutet *JavaScript Object Notation*. Ursprünglich also aus JavaScript entstanden, ist das Erzeugen und Interpretieren solcher Objekte mittlerweile in praktisch jeder modernen Programmiersprache möglich.

Allgemeines

JSON [1] wurde von Douglas Crockford spezifiziert und ist in den beiden konkurrierenden Standards RFC 7159 [2] und ECMA-404 [3] beschrieben. Wie es zu dieser skurrilen Situation kam, hat Web-Veteran Tim Bray in seinem Blog [4] skizziert. Für Java existieren viele bewährte Implementierungen. Mit dem JSR 353, *Java API for JSON Processing*, gibt es aber seit geraumer Zeit eine standardisierte Programmierschnittstelle. Ihre Referenzimplementierung [5] ist in Java EE 7 enthalten. Auf dem Client (Java SE 8) steht sie hingegen noch nicht automatisch zur Verfügung. Um die Bibliothek in eigenen Projekten zu verwenden, fügen Sie das Archiv *javax.json-1.0.4.jar* [6] dem Klassenpfad hinzu.

JSON ist bemerkenswert einfach gehalten. Das Format kennt nur zwei Datenstrukturen: Objekte und *Arrays*. Objekte sind Mengen von Name-Wert-Paaren. Arrays bestehen aus Listen von Werten. Mögliche Typen von Werten sind *string*, *number*, *object*, *array*, *true*, *false*, sowie *null*. Objekte werden durch geschweifte Klammern gekennzeichnet, ihre Name-Wert-Paare mit einem Komma voneinander getrennt. Zwischen dem Namen und dem Wert eines Paares steht ein Doppelpunkt. Die Namen in einem Objekt sind immer *Strings*. Ihre Werte können hingegen einen der oben genannten Typen annehmen, also auch Objekte und Arrays. Letztere werden von eckigen Klammern umschlossen. Ihre Werte sind durch ein Komma voneinander getrennt. Wichtig dabei: jeder Wert kann einen anderen Typ haben. Listing *testdaten.json* zeigt eine vollständige JSON-Datenstruktur.

```
{
  "name" : "Künneth",
  "vorname" : "Thomas",
  "gebdt" : 19700829,
  "verheiratet" : true,
  "hobbys" : ["Retro computing", null, 42]
}
```

Listing *testdaten.json*: Eine JSON-Datenstruktur

JSON-Datenstrukturen wirken üblicherweise kompakter als vergleichbare XML-Konstrukte. Die Vielzahl an sich öffnenden und schließenden *Tags* macht letztere für den Menschen schwerer fassbar. Allerdings lassen sich XML-Daten hervorragend validieren. Und die Lesbarkeit steht bei einem Datenaustauschformat sicher nicht an erster Stelle der Prioritätenliste. Insofern gibt es trotz JSON noch genügend sinnvolle Nutzungsszenarien für XML.

Das Erzeugen und Parsen von JSON-Objekten kann auf zweierlei Weise erfolgen. Man arbeitet entweder mit einen Baum im Speicher, der die Daten repräsentiert. Dieser kann nach Belieben navigiert, analysiert und verändert werden. Der Vorteil dieses Objektmodells, stets auf alle Elemente Zugriff zu haben, wird mit höherem Speicherverbrauch und unter Umständen Geschwindigkeitseinbußen erkauft. Oder man nutzt ein Streaming-Modell, wobei ein Ereignis-getriebener *Parser* stets genau ein Element liest. Immer, wenn beispielsweise der Beginn oder das Ende eines Objekts oder Arrays erkannt wird, oder wenn ein Name oder ein Wert gefunden wird, erzeugt der Parser ein Event, auf das man programmatisch reagieren kann. Nach der Verarbeitung durch die Anwendung fährt der Parser mit dem nächsten Element fort. Das Streaming-Modell ist äußerst praktisch, wenn das Programm keinen Überblick über die gesamte Datenstruktur braucht. Beide Modelle werden im Java Specification Request 353 berücksichtigt.

Das Objektmodell

Das Paket *javax.json* enthält Interfaces, Hilfsklassen und Datentypen für das Lesen, Schreiben und Erzeugen von JSON-Objektgeflechten. Wie leicht sich diese API nutzen lässt, zeigt das Listing *JsonReaderDemo.java*. Es liest eine JSON-Datenstruktur ein und gibt deren Inhalt auf der Konsole aus. Zuerst wird ein Objekt des Typs *javax.json.JsonReader* erzeugt. Dieses liefert durch Aufruf von *read()* eine *JsonStructure*. Sie bildet die Wurzel des Objektbaums. In Abhängigkeit vom Aufbau der Da-

tenstruktur handelt es sich entweder um ein *JsonObject* oder ein *JsonArray*. Beide Typen leiten von *javax.json.JsonStructure* ab.

```
package com.thomaskuenneth.jsontester;

import java.util.function.Consumer;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.json.JsonStructure;
import javax.json.JsonValue;

public class JsonReaderDemo {

  private static final Logger LOGGER =
    Logger.getLogger(
      JsonReaderDemo.class.getName()
    );

  public static void main(String[] args) {
    JsonReader reader =
      Json.createReader(
        JsonReaderDemo.class
        .getResourceAsStream("testdaten.json")
      );
    JsonStructure jsonst = reader.read();
    if (jsonst instanceof JsonObject) {
      JsonObject object = (JsonObject) jsonst;
      LOGGER.log(Level.INFO, object.toString());
      JsonArray array = object.getJsonArray("hobbys");
      if (array != null) {
        array.forEach(new Consumer<JsonValue>() {

          @Override
          public void accept(JsonValue value) {
            LOGGER.log(
              Level.INFO, value.getValueType() + ": "
              + value.toString()
            );
          }

        });
      }
    }
  }
}
```

Listing *JsonReaderDemo.java*: Einlesen und Verarbeiten einer *.json*-Datei

Arrays lassen sich sehr bequem mit der Methode *forEach()* verarbeiten. Ihr wird ein Konsument übergeben, der für jedes Element aufgerufen wird. Die Beispielimplementierung gibt den JSON-Typ sowie den Wert des Feldelements auf der Konsole aus. Das Interface *java.util.function.Consumer* wurde übrigens mit Java 8 eingeführt. Auch das programmatische Erzeugen einer JSON-Struktur ist mit äußerst wenigen Zeilen Quelltext möglich. Wie das aussehen kann, zeigt das Listing *JsonBuilderDemo.java*. Als erstes wird mit *Json.createObjectBuilder()* ein Objekt des Typs *javax.json.JsonObject* erzeugt. Diesem werden durch aufeinander folgende Aufrufe der Methode *add()* beliebige Werte und Unterstrukturen hinzugefügt.

```
package com.thomaskuenneth.jsontester;

import java.util.logging.Logger;

import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonValue;

public class JsonBuilderDemo {

  private static final Logger LOGGER =
    Logger.getLogger(
      JsonBuilderDemo.class.getName()
    );

  public static void main(String[] args) {

    JsonObject model = Json
      .createObjectBuilder()
      .add("name", "Künneth")
      .add("vorname", "Thomas")
      .add("gebdt", 19700829)
      .add("verheiratet", true)
      .add("hobbys", Json.createArrayBuilder()
        .add("Retro computing")
        .add(JsonValue.NULL).add(42)
      ).build();
    LOGGER.info(model.toString());
  }

}
```

Listing *JsonBuilderDemo.java*: Programmatisches Erzeugen eines JSON-Objekts

Das Serialisieren einer JSON-Struktur in eine Datei ist in Java 8 mit wenigen Zeilen Quelltext abgehandelt. *try-with-resources* sorgt hierbei für das Schließen der nicht mehr benötigten Ressourcen.

```
try (FileWriter stWriter = new FileWriter("ausgabe.json");
JsonWriter jsonWriter = Json.createWriter(stWriter)) {
  jsonWriter.writeObject(model);
} catch (IOException e) {
  LOGGER.log(
    Level.SEVERE, "Fehler beim Erzeugen des FileWriter", e
  );
}
```

Streaming API

Besonders effizient lassen sich JSON-Strukturen mit Hilfe eines Parsers verarbeiten. Die Methode *createParser()* der Klasse *javax.json.Json* liefert ein Objekt des Typs *javax.json.stream.JsonParser*. Mit *hasNext()* prüfen Sie, ob noch weitere Elemente einer JSON-Struktur geparst werden können. In diesem Fall liefert *next()* eine Instanz von *JsonParser.Event*. Je nach Art des aufgetretenen Ereignisses können die auf diese Weise ermittelten Werte verarbeitet werden. Die grundsätzliche Vorgehensweise ist im Listing *JsonParserDemo* zu sehen.

```
package com.thomaskuenneth.jsontester;

import java.util.logging.Level;
import java.util.logging.Logger;

import javax.json.Json;
import javax.json.stream.JsonParser;

public class JsonParserDemo {

  private static final Logger LOGGER =
    Logger.getLogger(
      JsonParserDemo.class.getName()
    );

  public static void main(String[] args) {
    JsonParser parser = Json.createParser(
      JsonReaderDemo.class
      .getResourceAsStream("testdaten.json")
    );
    while (parser.hasNext()) {
      JsonParser.Event event = parser.next();
      LOGGER.log(Level.INFO, event.toString());
      switch (event) {
        case START_ARRAY:
        case START_OBJECT:
        case END_ARRAY:
        case END_OBJECT:
        case VALUE_NULL:
        case VALUE_FALSE:
        case VALUE_TRUE:
          // nix zu tun... :-)
          break;
        case KEY_NAME:
        case VALUE_STRING:
        case VALUE_NUMBER:
          LOGGER.log(Level.INFO, parser.getString());
          break;
      }
    }
  }
}
```

Listing *JsonParserDemo.java*: Die Streaming API

Auch das kontinuierliche Erzeugen von JSON-Strukturen wird mit der Streaming API zum Kinderspiel. Die Vorgehensweise ist im Listing *JsonGeneratorDemo.java* zu sehen. Nach dem Erzeugen einer Generator-Klasse mit *createGenerator()* folgt als erstes ein initiales *writeStartObject()*, im Anschluss daran die gewünschten *write...()*-Aufrufe. Beispielsweise schreibt *write(42)* die übergebene Zahl. Arrays leiten Sie mit *writeStartArray()* ein und schließen sie mit *writeEnd()* ab. Ein letztes *writeEnd()* bildet die schließende Klammer zum ersten *writeStartObject()*. Die korrespondierende Ressource wird mit *close()* geschlossen.

```
package com.thomaskuenneth.jsontester;

import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.json.Json;
import javax.json.JsonValue;
import javax.json.stream.JsonGenerator;

public class JsonGeneratorDemo {

  private static final Logger LOGGER =
    Logger.getLogger(
      JsonGeneratorDemo.class.getName()
    );

  public static void main(String[] args) {
    FileWriter writer;
    try {
      writer = new FileWriter("ausgabe2.json");
      JsonGenerator gen = Json.createGenerator(writer);
      gen.writeStartObject().write("name", "Künneth")
      .write("vorname", "Thomas").write("gebdt", 19700829)
      .write("verheiratet", true).writeStartArray("hobbys")
      .write("Retro computing").write(JsonValue.NULL)
      .write(42).writeEnd().writeEnd();
      gen.close();
    } catch (IOException e) {
      LOGGER.log(
        Level.SEVERE,
        "Fehler beim Erzeugen von FileWriter", e
      );
    }
  }

}
```

Listing *JsonGeneratorDemo.java*: Kontinuierliches Generieren einer JSON-Struktur

Fazit

JSR 353 definiert eine sehr leicht nutzbare Schnittstelle für die Verarbeitung von JSON-Strukturen. Auch wenn die Referenzimplementierung noch von Hand den eigenen Projekten hinzugefügt werden muss, lohnt es auf jeden Fall, über einen Einsatz in neuen Projekten nachzudenken.

Skurriler Weise gibt es mit *JEP 198: Light-Weight JSON API* für Java 9 Überlegungen, eine weitere, konkurrierende JSON-API einzuführen. Noch ist nicht allzu viel über diese bekannt. Ob dem JSR 353 damit ein ernsthafter Gegner erwächst, muss die Zukunft zeigen.

Referenzen

[1] JSON *Introducing JSON*
http://www.json.org

[2] Tools.ietf.org *RFC 7159*
http://tools.ietf.org/html/rfc7159

[3] ECMA *Standard ECMA-404*
http://www.ecma-international.org/publications/standards/Ecma-404.htm

[4] TBray *JSON Redux AKA RFC7159*
https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON

[5] JSON Processing *JSON Processing*
https://jsonp.java.net

[6] *javax.json-1.0.4.jar*
http://search.maven.org/remotecontent?filepath=org/glassfish/javax.json/1.0.4/javax.json-1.0.4.jar

Kurzbiografie

Thomas Künneth ist als Senior Consultant für die MATHEMA Software GmbH tätig. Seit annähernd 15 Jahren beschäftigt er sich intensiv mit Java. Er ist Autor der Bücher „Android 4“, „Einstieg in Eclipse 4.4“ und „Java für Windows“ sowie zahlreicher Fachartikel.

Kuchen für Alle!

Über die JavaScript-Bibliothek D3 zur Visualisierung von Daten

von Sandy Stehr

Es gibt einige *JavaScript*-Bibliotheken für Torten- und Balkendiagramme auf dem Markt, wie beispielsweise *googleapi*, *Processing* oder *Raphaël*. Als *D3* 2011 herauskam, wurde schnell klar, dass es ein mächtiges Werkzeug sein würde, um Datenvisualisierungen im Web zu erstellen. Doch warum wurde es so erfolgreich, wobei es doch nicht das erste und auch nicht das einzige Werkzeug auf dem Markt ist?

Was ist D3 und wie funktioniert es?

D3.js oder nur D3 steht für *Data-Driven Documents* und ist eine von Michael Bostock entwickelte Open-Source JavaScript-Bibliothek um Datensätze im Web dynamisch und interaktiv grafisch darzustellen. Dabei nutzt es die aktuellen JavaScript, *SVG*, *HTML5* und *CSS3* Web-Standards und ist somit in allen modernen Web-Browsern (außer *IE8* und darunter) lauffähig.

D3 ermöglicht es auf einfache Weise SVG-Grafiken zu erstellen und die DOM-Struktur (*Document Object Model*) durch Datensätze zu manipulieren. Dabei kann man mit D3 eine einfache HTML-Tabelle erstellen oder ein interaktives SVG-Diagramm mit Animationen, Übergängen und Datenaktualisierungen in Echtzeit.

Das „d3“-Objekt wird ähnlich dem „$“-Objekt in *jQuery* benutzt. Damit lassen sich DOM-Elemente selektieren, neu erstellen und auch wieder löschen.

```
d3.selectAll("p").style("color", "white");
```

Zum Zeichnen braucht man eine SVG-basierte *Canvas*. Diese Canvas und alle anderen SVG-Elemente werden mittels *.append()* an ein Element angehängt, welches vorher, wie im folgenden Beispiel, über dessen ID *item* ausgewählt wird.

```
d3.select("#item").append("svg");
```

Attribute können durch einfache Aneinanderreihung hinzugefügt und verändert werden.

```
d3.select("#item").append("svg")
  .attr("width", 100)
  .attr("height", 100);
```

Styles, Attribute und andere Eigenschaften können aber nicht nur durch einfache Konstanten, sondern auch mittels Funktionen dynamisch verändert werden, z. B. um Paragrafen in einer zufälligen Farbe einzufärben:

```
d3.selectAll("p").style("color", function() {
  return "hsl(" + Math.random() * 360 + ",100%,50%)";
});
```

Allen Attributen und Styles können Animationen angehängt werden per

.transition(),
.delay(), oder
.duration().

Durch Hinzufügen von *.on()* können zudem noch Events angebunden werden. Eine Verkettung von Animationen ist somit auch möglich.

```
d3.selectAll("p").on("mousedown", animateFirstStep);

function animateFirstStep(){
  d3.select(this)
  .transition()
  .delay(0)
  .duration(1000)
  .attr("r", 10)
  .each("end", animateSecondStep);
}

function animateSecondStep(){
  d3.select(this)
  .transition()
  .duration(1000)
  .attr("r", 40);
}
```

Besonders interessant ist aber nun das Hinzufügen ganzer Datensets. Diese können als einfaches *Array* oder als *JSON*-Objekt mit dem Attribut *.data()* eingefügt werden. Mit *.enter()* und *.exit()* werden Elemente hinzugefügt oder gelöscht.

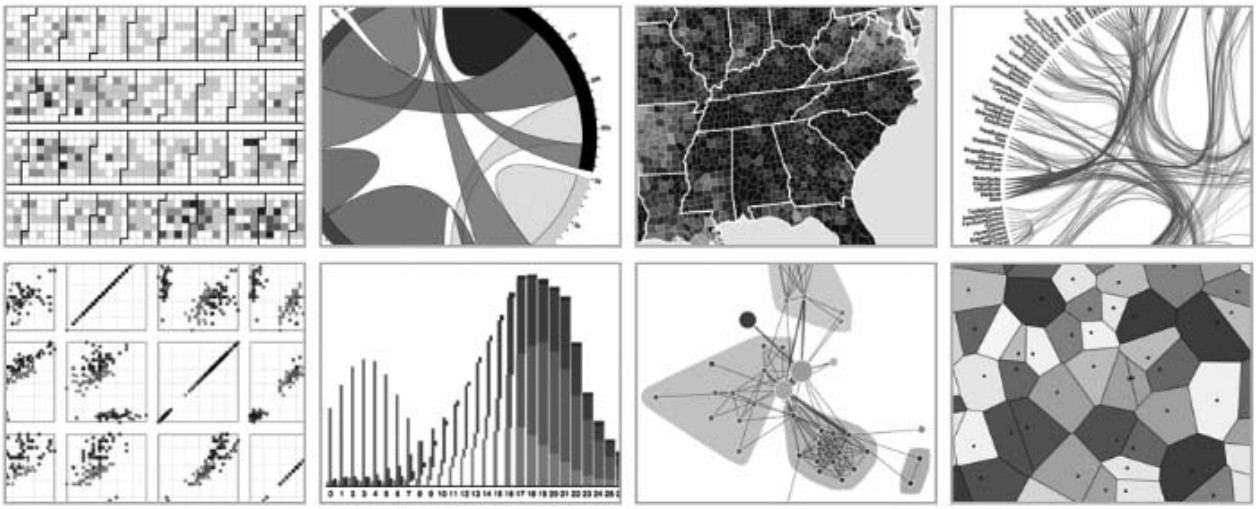


Abbildung 1: Beispiele für interaktive Visualisierungen mit D3

```
var dataset = [ 25, 7, 5, 26, 11, 8, 25, 14, 23, 19, 14, 11, 22,
29, 11, 13, 12, 17, 18, 10];

d3.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("y", function(d){
   return h - (d*2);
   })
```

Vorteile

Der Erfolg von D3 hängt gewiss von mehreren Faktoren und dem Ansatz, den D3 wählt, ab. Einer der Gründe ist sicherlich, dass es offene Standards nutzt und somit nahtlos mit den derzeitigen Web-Technologien funktioniert und daher in allen modernen Web-Browsern ohne weitere *Plugins* lauffähig ist. Es ist nicht auf kleine Bereiche auf einer Web-Seite begrenzt wie *Raphaël.js* oder andere Canvas- oder nur auf SVG-basierende Bibliotheken. Eine umfangreiche Dokumentation, eine große Fan-Gemeinde und besonders die Zugänglichkeit von Mike Bostock mit seinen unzähligen Code-Beispielen erleichtern den Einstieg enorm und tragen zum Erfolg bei.

Nachteile

Natürlich ist nicht alles Gold was glänzt und somit gibt es auch einige Nachteile. So kann die DOM-Manipulation für eine große Anzahl von Einträgen extrem langsam werden. Auch bekommt SVG Performanzprobleme mit einer großen Menge an Elementen. Andererseits sollte man für eine gute Darstellung die Daten ohnehin übersichtlich halten. Für große Datenmengen empfiehlt sich die vorhergehende Verarbeitung in kleinere Datensätze. Außerdem muss man etwas JavaScript lernen, aber so ist das eben mit allen Technologien. Dafür ist die Lernkurve steil und es gibt sehr viele gute Tutorials im Netz, sowie viel Unterstützung in Foren.

Wichtig ist bei D3, dass es eben keine Grafik- oder Datenverarbeitungsbibliothek ist. Es enthält keine vorgefertigten Diagramme, sondern ermöglicht kreativere und innovativere Wege zur Datendarstellung als nur mit den üblichen altmodischen Balken-, Linien- und Tortendiagrammen.

Stattdessen ist es ein Werkzeug, welches die Verbindung zwischen Daten und der Grafikerstellung einfach macht. Es erlaubt die volle Kontrolle über das finale visuelle Ergebnis und gibt unendlich viele Möglichkeiten der Datenvisualisierung.

Referenzen

[1] D3 Data-Driven Documents *Overview*
www.d3js.org

[2] GitHub *D3 Wiki*
https://github.com/mbostock/d3/wiki

[3] Bostock, Michael *Ogievetsky, Vadim, Heer, Jeffrey, D3: Data-Driven Documents*, http://vis.stanford.edu/papers/d3

[4] bl.ocks.org *mbostock's blocks, D3 Anwendungsbeispiele*
http://bl.ocks.org/mbostock

[5] W3C *Scalable Vector Graphics (SVG) 1.1 Specification*
http://www.w3.org/TR/2003/REC-SVG11-20030114

[6] W3C *Document Object Model*
www.w3.org/DOM

Kurzbiografie

Sandy Stehr ist seit 2013 als Entwicklerin und Consultant für die MATHEMA Software GmbH tätig. Ihre Schwerpunkte liegen in der Entwicklung von *Frontends* und *Templates* von Java-basierten Web-Applikationen.

Des Programmierers kleine Vergnügen

Schleifenüberlauf-dilemma

von Michael Wiedeking

Eine *for*-Schleife über eine Sequenz von ganzen Zahlen laufen zu lassen, ist trivial: Initialisierung der Schleifenvariablen, Test, Inkrement – fertig. Nicht ganz so einfach ist es, wenn das letzte zu durchlaufende Element der maximale *Integer*-Wert ist; und noch blöder wird es, wenn man über alle *Integer*-Zahlen iterieren will, natürlich ohne versehentlich in einer Endlosschleife zu landen.

Stellen Sie sich vor, sie wollen über alle nicht negativen ganzen Zahlen, also von 0 bis zur größten darstellbaren Zahl MAX, iterieren. Dabei ergibt sich schon das Problem, dass eine Abfrage mit der kanonischen Form $i < M$ gar nicht möglich ist, weil es ein „maximaleres“ $M = MAX + 1$ schon alleine sprachlich nicht gibt. Der daraus resultierende naive Ansatz

```
for (int i = 0; i ≤ MAX; i += 1) {
  …
}
```

funktioniert aber trotzdem nicht. Denn wenn die Laufvariable das Maximum MAX erreicht hat, läuft das Inkrement i += 1 über. Für dieses neue, nunmehr negative i gilt aber leider auch, dass $i <$ MAX ist. Und damit wird die Schleife natürlich nicht beendet.

Jetzt könnte man sich zu Nutze machen, dass nach MAX eine negative Zahl, also ein Vorzeichenwechsel von Plus nach Minus kommt und die Bedingung zu $i >= 0$ umformen. Da wir aber stets auf der Suche nach allgemeinen Lösungen sind, können wir diese Lösung einfach dadurch zunichte machen, dass wir beispielsweise über alle *Integer* iterieren wollen.

Ein erster Lösungsansatz für dieses Problem ist eigentlich recht simpel, wenngleich er nicht unbedingt einfach einzusehen ist: Man nutzt einfach die „Äquivalenz“

$$x \le y \;\Leftrightarrow\; x - y \le 0 \;\Leftrightarrow\; y - x \ge 0\,.$$

Der umgeformte Ausdruck ist zwar vergleichbar falsch, aber glücklicherweise an einer anderen Stelle. Mit der Differenz kann nämlich bestimmt werden, wie groß der Abstand $\delta = x - y$ zwischen zwei Zahlen ist. Misst man nun den Abstand zwischen dem aktuellen Index i und dem maximalen Wert MAX, so ist dieses $\delta = MAX - i$ immer größer gleich 0, solange i vor oder auf dem End-Index liegt. Das funktioniert auch dann noch, wenn das Inkrement überläuft, da die Differenz $MAX - i$ trotzdem noch korrekt berechnet wird.
Die Schleife lässt sich demnach wie folgt umwandeln:

```
for (int i = 0; MAX – i ≥ 0; i += 1) {
  …
}
```

Findet der Überlauf statt, so liefert die Differenz den Wert –1, und die Schleife wird abgebrochen. Die –1 kommt übrigens dadurch zustande, weil die Differenz von MAX = 0x7FFFFFFF und dessen Nachfolger MIN = MAX + 1 = 0x80000000 unabhängig von der Interpretation der Bit-Muster 0xFFFFFFFF ist. Interpretiert man dies als vorzeichenlose Zahl, so erhält man die maximale Differenz; interpretiert man aber das Vorzeichen-Bit, so erhält man als Ergebnis –1 (was ja auch so sein muss, denn wenn $i = MAX + 1$ ist, muss auch $MAX - i = -1$ gelten). Das Abbruchkriterium hält aber auch stand, wenn es keinen Überlauf gibt, denn für ein beliebiges letztes Element *last* wird die Differenz *last* – i genau dann negativ, wenn $i >$ *last* ist.

Leider funktioniert das auch nicht mehr, wenn man mehr als MAX Elemente hat, denn selbstverständlich kann auch die Subtraktion überlaufen; allein wenn man mit $i = -1$ beginnen und bis MAX iterieren möchte, läuft jene über. War der Abstand zwischen MAX und 0 noch MAX, so ist er jetzt schon MAX + 1, also MIN. Soll die Schleife mit –2 beginnen, so ist die Differenz MIN + 1 usw. Das Abbruchkriterium muss also etwas sorgfältiger formuliert werden: Erst wenn die Differenz genau –1 ist, ist das Ende der Schleife erreicht.

```
for (int i = 0; MAX – i ≠ –1; i += 1) {
  …
}
```

Leider ist auch das unbefriedigend, falls man eine Schrittweite hat, die von 1 verschieden ist. In dem Fall ist MAX natürlich nur noch eine obere Schranke, die möglicherweise nicht genau erreicht wird. Wählt man dann etwa für die untere Grenze –1 und für die obere Grenze MAX bei einer Schrittweite von 7, so ist das letzte $\delta = MAX - i = -5$. Das Problem ist also nicht, dass das δ negativ ist, sondern dass es im negativen Bereich bis $-\Delta$ liegt, wenn Δ

die Schrittweite ist. Die Schleife darf also durchlaufen werden, solange für die Differenz $\delta = MAX - i$ entweder $\delta \geq 0$ oder $\delta < -\Delta$ gilt:

```
for (int i = 0; (MAX – i) ≥ 0 || (MAX – i) < –Δ; i += Δ) {
  …
}
```

Jetzt ergibt sich allerdings ein neues Problem, denn wenn der Abstand (fast) maximal ist, ist diese Bedingung schon für die erste Iteration nicht erfüllt. Will man etwa von MIN bis MAX alle Werte mit einem beliebigen Δ durchlaufen, dann ist $i = MIN$ und $MAX - i = MAX - MIN = -1$, und das ist leider in dem halb offenen Intervall $[-\Delta, 0[$ enthalten und verhindert so insgesamt das Ausführen der Schleife.

Das führt zu der bitteren Erkenntnis, dass dieses Problem allgemein nicht mit einer *for*-Schleife erledigt werden kann, weil es sich bei dem hergeleiteten Bedingungen wohl eher um ein Abbruchkriterium handelt. Deshalb muss leider auch die Entscheidung darüber, ob die Schleife überhaupt durchlaufen wird, separat geprüft werden. Damit erhält man für das erste Element *first*, das letzte Element *last* und die Schrittweite Δ die folgende Schleife.

```
if (first ≤ last) {
  int i = first;
  int δ;
  do {
    …
    i += Δ;
    δ = last – i;
  } while ((δ ≥ 0) || (δ < –Δ));
}
```

Die beiden Bedingungen am Ende sind allerdings nicht ganz so schön, und es stellt sich die Frage, ob das nicht nur mit einem Test erledigt werden könnte. Die Schleife wird beendet, wenn δ in dem kleinen Bereich zwischen –1 und –Δ liegt. Das lässt sich auch herausfinden, wenn man den Ausdruck $(\delta \geq 0) \,||\, (\delta < -\Delta)$ umformt:

$$
\begin{aligned}
&(\delta \geq 0) \,||\, (\delta < -\Delta) && \Leftrightarrow \\
&!!((\delta \geq 0) \,||\, (\delta < -\Delta)) && \Leftrightarrow \\
&!(!(\delta \geq 0) \,\&\&\, !(\delta < -\Delta)) && \Leftrightarrow \\
&!((\delta < 0) \,\&\&\, (\delta \geq -\Delta)) && \Leftrightarrow \\
&!((-\Delta \leq \delta) \,\&\&\, (\delta < 0)) && \Leftrightarrow \\
&!(-\Delta \leq \delta < 0)
\end{aligned}
$$

Aus einem der vergangenen Vergnügen [1] wissen wir, dass derartige Bereichsprüfungen tatsächlich mit nur einem Test geprüft werden können. Aber wir können uns das auf die Schnelle auch herleiten.

Die beiden Tests müssen ja nur deswegen gemacht werden, weil dieses Intervall nicht am „Rand" liegt. Wenn dieses Intervall an den Rand verschoben wird, so kann auf einen der Tests verzichtet werden, weil die Randeigenschaft zwangsläufig immer gilt. Wählt man spaßeshalber den „linken" Rand, so verschiebt man das Intervall derart, dass der Test $-\Delta \leq \delta$ implizit erfüllt ist.

Addiert man die Ungleichung mit $MIN + \Delta$, so erhält man die gewünschte Verschiebung:

$$
\begin{aligned}
&-\Delta + MIN + \Delta \leq \delta + MIN + \Delta < 0 + MIN + \Delta \quad \Leftrightarrow \\
&MIN \leq \delta + MIN + \Delta < MIN + \Delta
\end{aligned}
$$

Da MIN die kleinstmögliche, vorzeichenbehaftete Zahl ist, verbleibt als einziger Test die Bedingung

$$\delta + MIN + \Delta < MIN + \Delta\,.$$

Da MIN und Δ konstant sind, braucht $D = MIN + \Delta$ nur ein einziges Mal vor der Schleife berechnet zu werden. Damit erhält man also ersatzweise für die Schleifenbedingung

```
do {
  …
} while (!(δ + D < D));
```

Wird die Schrittweite Δ dynamisch bestimmt, ist noch darauf zu achten, dass sie positiv ist und insbesondere nicht 0 sein darf. Das kann man sicherheitshalber noch eingangs vor der Schleife prüfen:

```
if ((Δ > 0) && (first ≤ last)) {
  …
}
```

Damit hat man mit nur zwei zusätzlichen Instruktionen pro Schleifendurchlauf eine allgemeine Möglichkeit geschaffen, über ganzzahlige Werte in einem abgeschlossenen Intervall bei beliebiger Schrittweite zu iterieren. Aber keine Sorge: Vor den beiden Instruktionen muss nur derjenige Angst haben, der überhaupt nicht weiß, was für eine Schleife er braucht. Und in diesem speziellen Fall sind zwei (bedingungsfreie) Instruktionen wahrlich nicht zu viel…

Referenzen

[1] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Voll beschränkt!* KaffeeKlatsch, Jahrgang 3, Nr. 4, Seite 13, Bookware, April 2010 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2010-04.pdf

Kurzbiographie

Michael Wiedeking (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt" Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

Kaffeesatz

… ex machina

von Michael Wiedeking

Mir fällt nichts ein. Wie auch? In dem Land, in dem alles zu groß, zu übertrieben und zu laut ist, sitze ich in einer Lobby und warte darauf, dass mir etwas einfällt. Aber der Lärm ist einfach zu laut. Ich könnte zwar in mein Zimmer gehen, aber wenn mir da auch nichts einfällt, habe ich noch nicht einmal eine Entschuldigung.

Jetzt wäre eines dieser Programme nicht schlecht, die Texte ganz alleine schreiben können. Sollen ja dieser Tage schon an jeder Straßenecke zu haben sein. Dann könnte ich mich getrost zurückziehen, morgen früh noch einmal drübergucken und ab in die Redaktion damit.

Da gibt es etwa den Quakebot, der in Los Angeles über Erdbeben berichtet. Bei uns ist das immer eine große Sache, wenn die Erde bebt, aber in Kalifornien… Da berichtet man doch eher darüber, wenn es mal kein Erdbeben gegeben hat. Und eben dort hat ein findiger Journalist ein Programm geschrieben, das die Daten vom U.S. Geological Survey (USGS) in lesenswerte Nachrichten umwandelt.

So hieß es dort [1] am 25. Oktober um 8.24 Uhr:

> *„A shallow magnitude 3.6 earthquake was reported Saturday morning 32 miles south from Lone Pine, Calif., in the Sequoia National Forest, according to the U.S. Geological Survey. The temblor occurred at 7:31 a.m. Pacific time at a depth of zero miles. According to the USGS, the epicenter was 40 miles north-northwest of Ridgecrest, 53 miles east of Porterville and 57 miles east of Lindsay.*
> *In the last 10 days, there have been no earthquakes of magnitude 3.0 or greater centered nearby.“*

Das ist sicherlich nichts, was literarisch auch nur irgendwie von Wert sein könnte, aber kostet den verantwortlichen Autor im Prinzip nur das Lesen des Artikels. Der Rest passiert von ganz alleine, wann immer über den USGS eine neue Meldung eintrudelt. Wenn man sich die anderen Meldungen anschaut [2], sieht man auch, warum jede Meldung mit dem Hinweis „*This information comes from the USGS Earthquake Notification Service and this post was created by an algorithm*“ schließt.

Unabhängig von dem literarischen Wert und der sprachlichen Vielfalt ist obiger Text auf jeden Fall besser zu lesen als

„{"type":"Feature","properties":{"mag":3.61,"pla
ce":"52km S of Lone Pine, California","time":1
414247475080,"updated":1414431037010,"tz":-
420,"url":"http://earthquake.usgs.gov/earthquakes/
eventpage/ci37282264","detail":"http://earthquake.
usgs.gov/earthquakes/feed/v1.0/detail/ci37282264.
geojson","felt":4,"cdi":2.7,"mmi":4.36,"alert":null,"
status":"reviewed","tsunami":null,"sig":202,"net":"c
i","code":"37282264","ids":",ci37282264,","sources
":",ci,","types":",cap,dyfi,focal-mechanism,general-
link,geoserve,nearby-cities,origin,phase-data,scitech-
link,shakemap,","nst":41,"dmin":0.16,"rms":0.1
9,"gap":71,"magType":"ml","type":"earthquake"
,"title":"M 3.6 - 52km S of Lone Pine, Californ
ia"},"geometry":{"type":"Point","coordinates":[-
118.0518333,36.1301667,0]},"id":"ci37282264"}“.

Wann immer ein Erdbeben auftritt, egal wie stark oder schwach, wird eine vergleichbare Meldung versandt. Entweder als *ATOM*, *GeoJSON*, *KML*, *Spreadsheet* oder *QuakeML*. Die Nachrichten sind im Abonnement oder via Web-Service beziehbar. Beispielsweise füllen die Ereignisse für die letzten 30 Tage eine Datei mit 6.3 MB und beschreiben Erdbeben und Tsunamis auf der ganzen Welt. Um also herauszufinden, ob ein Beben in Kalifornien stattgefunden hat, muss man die Suche sowieso auch auf bestimmte Koordinaten einschränken. Und alleine für diese Aufgabe lohnt es sich schon, einen Rechner zu bemühen.

Werden Journalisten durch den Einsatz eines solchen Textroboters nun arbeitslos? Nein, sie werden lediglich von der Last befreit, derart langweilige Ereignisse in Worte fassen zu müssen. Nicht ohne Grund gibt es diese Textroboter inzwischen auch für Sportereignisse. Hilft es beim Verfassen des Kaffeesatzes? Nein, wirklich geistreiche Dinge wird man auch in naher Zukunft noch „von Hand“ schreiben müssen.

Und ist es nicht trotzdem auch eine Hilfe? Ja, auf jeden Fall, denn dank der Textroboter ist mir ja doch noch etwas eingefallen, und ich kann nun getrost ins Bett gehen.

Referenzen

[1] Quakebot *Earthquake: 3.6 quake strikes near Lone Pine*, Los Angeles Times, 25. Oktober 2014, http://www.latimes.com/local/lanow/la-me-earthquake-magnitude-36-lone-pine-20141025-story.html

[2] Los Angeles Times *Earthquakes* http://www.latimes.com/local/earthquakes

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
http://www.jug-bb.de
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

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

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

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

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

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

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

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

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

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

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

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

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

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
http://www.jug-n.de
Kontakt: Frau Natalia Wilhelm
(info@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
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)
Rainer Anglett (ra@jugs.org)

Schweiz

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

.NET User Groups

Deutschland

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

.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
http://www.dodnedder.de
Kontakt: Herr Udo Neßhöver, Frau Ulrike Stirnweiß
(info@dodnedder.de)

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

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

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

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

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

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

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

.NET Developers Group Stuttgart
Tieto Deutschland GmbH
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Herr Michael Niethammer

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

Österreich

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

Software Craftsmanship Communities

Deutschland, Schweiz, Österreich

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

MATHEMA

meet the experts of enterprise infrastructure

Weiterführende Programmierung unter C#
Umstieg auf C# 4.5/5 und Einführung in fortgeschrittene Konzepte
24. – 26. November 2014, 11. – 13. Mai 2015
1.350,– € (zzgl. 19 % MwSt.)

HTML5, CSS3 und JavaScript
9. – 12. März 2015, 21. – 24. September 2015
1.650,– € (zzgl. 19 % MwSt.)

Entwicklung mobiler Anwendungen mit iOS
20. – 22. April 2015, 5. – 7. Oktober 2015
1.250,– € (zzgl. 19 % MwSt.)

Fortgeschrittenes Programmieren mit Java
Ausgewählte Pakete der Java Standard Edition
4. – 6. Mai 2015, 19. – 21. Oktober 2015
1.350,– € (zzgl. 19 % MwSt.)

AngularJS
18. – 19. Mai 2015, 7. – 8. Dezember 2015
950,– € (zzgl. 19% MwSt.)

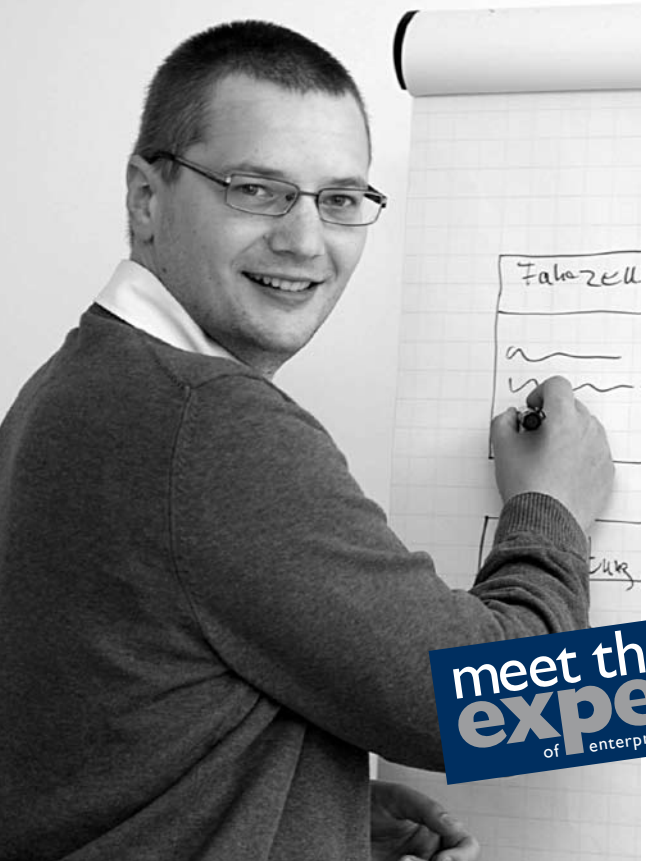
Anwendungsentwicklung mit der Java Enterprise Edition
22. – 26. Juni 2015, 30. Nov – 4. Dez. 2015
2.150,– € (zzgl. 19 % MwSt.)

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

training



18. – 19. Mai 2015, 7. – 8. Dezember 2015,
950,– € (zzgl. 19% MwSt.)

AngularJS

Trainer: Moritz Herrmann

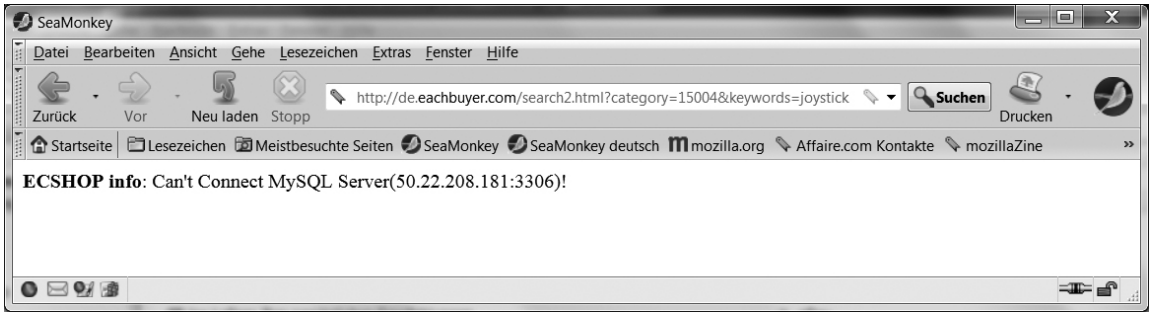
Das JavaScript Framework AngularJS ermöglicht Ihnen die Entwicklung von modernen Web-Applikationen basierend auf dem MVVM-Prinzip. Aufbauend auf unseren umfangreichen Praxiserfahrungen mit AngularJS vermitteln wir Ihnen in diesem Kurs alle wichtigen Grundlagen, Komponenten und Konzepte des Frameworks. Durch viele Beispiele und Übungen können Sie Ihr neu erworbenes Wissen sofort praktisch anwenden und vertiefen.

Selbstverständlich führen wir das Training auch gerne in Ihrem Haus durch. Sprechen Sie uns an und wir entwickeln ein auf Sie persönlich zugeschnittenes Angebot.

Weitere Informationen und Kursanmeldung unter:
http://www.mathema.de/training/katalog/angularjs

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 | Madeline Hörner

Das Allerletzte



Dies ist kein Scherz!
Diese Fehlermeldung 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 November.

Herbstcampus

Wissenstransfer par excellence

31. August – 3. September 2015
in Nürnberg