Node.js der Alleskönner

Kai Donato
MT AG
Ratingen

Schlüsselworte

JavaScript, Node.js, NPM, Express, Webserver, oracledb

Einleitung

Node.js ist nach seiner Veröffentlichung im Jahre 2009 in aller Munde und erfreut sich auf Grund seiner Eigenschaften für die serverseitige Verarbeitung von JavaScript einer großen Popularität.

In diesem Vortrag werden die Grundzüge der Entwicklung mit Node.js sowie einige praktische Beispiele vorgestellt. Ein Webserver in kürzester Zeit entwickeln, sowie in Verbindung mit dem „oracledb"-Treiber zu einer einfachen REST-Schnittstelle transformieren. Das Erweitern einer APEX-Applikation durch mehrere kleine Tools, die sich mit Node.js bereitstellen lassen. All dies wird ihnen in diesem Vortrag anschaulich gezeigt.

Basics von Node.js

Node.js ist im Grunde genommen nur eine Möglichkeit JavaScript auf einem Server auszuführen. „Nur" heißt aber in diesem Fall nicht, dass es nicht mächtig genug ist, um in jeder Ihrer Applikationen Fuß fassen könnte. Es eröffnet dem Entwickler ganz neue Einsatzmöglichkeiten und hilft dabei Probleme oder Anforderungen unter Umständen deutlich effizienter zu lösen.

Um mit der Entwicklung einer Node.js-Applikation zu beginnen, machen wir uns einem mitgelieferten Paketmanager zunutze, der für uns bei der Verwaltung der einbezogenen Pakete und unseres eigenen Quellcodes unter die Arme greift:

Abb. 1: Initialisieren eines Node.js-Projekts

Während des Initialisierungsprozesses werden alle nötigen Informationen zu unserer neu angelegten Applikation abgefragt und anschließend in einem JSON-File hinterlegt (package.json). Diese Datei sorgt dafür, dass Informationen, wie der Autor, die Lizenz, der Einstiegspunkt und die Version sowie der Name der Anwendung stets beim Projekt im Verzeichnis vorliegt. Des Weiteren wird diese Datei noch unsere Referenzen zu den genutzten (Fremd-) Bibliotheken beinhalten, die für den produktiven und/oder den entwicklungsspezifischen Einsatz vorgesehen sind. Dazu später mehr.

Installation des ersten Pakets

Installieren wir nun ein Paket mittels mitgeliefertem Paket Manager, so lädt dieser für uns alle abhängigen Ressourcen sowie den Quellcode in das Verzeichnis „node_modules“ herunter. Von nun an steht das Paket für unsere Applikation zur Verfügung.

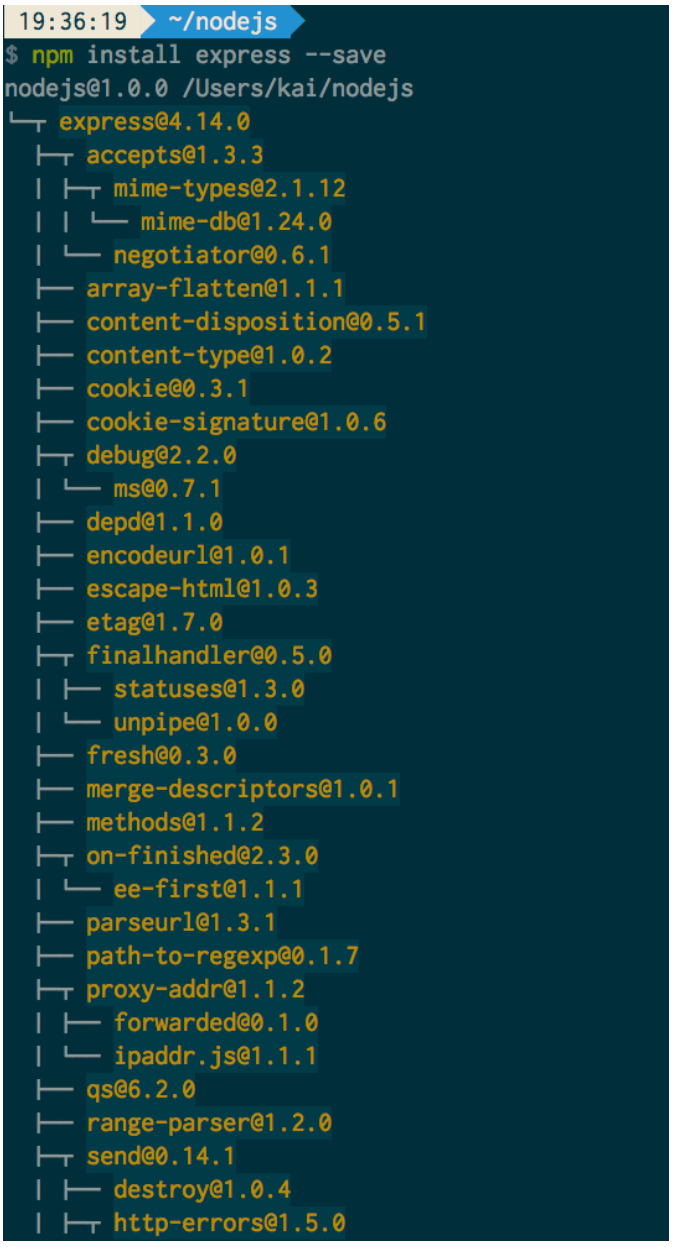

Abb. 2: Installation unseres ersten NPM-Pakets

In einer übersichtlichen Darstellung zeigt uns NPM alle Abhängigkeiten, die es automatisch installiert. Übrigens, jedes dieser Pakete ist in der Grundstruktur aufgebaut, wie unsere Applikation. Inklusive der

„package.json“-Datei, die wiederum alle Abhängigkeiten dieses Pakets besitzt, werden alle Bibliotheken strukturiert im „node_modules“-Ordner abgelegt:

Abb. 3: Ordnerstruktur des "node_modules"-Verzeichnisses

Nun haben wir durch den in Abbildung 2 gezeigten Befehl „**npm install express --save**“ unser erstes Paket installiert und mit dem Parameter „--save“ in unserer „package.js“-Datei persistiert.

Erstellen unseres einfachen Webservers

Mit wenigen Handgriffen haben wir nun das Paket installiert, das wir benötigen, um unseren ersten eigenen Webserver aufzubauen. Mit dem folgenden Code-Snippet können wir unserer Applikation („index.js) Leben einhauchen:

```
1  var express = require('express')
2  var app = express()
3
4  app.get('/', function (req, res) {
5    res.send('Hello World')
6  })
7
8  app.listen(3000)
```

Abb. 4: Beispiel-Code des minimalen Webservers mit „express“

Mit dem folgenden Kommando können wir unsere Applikation starten:

```
node start index.js
```

Fertig ist unser erster kleiner Webserver! Wir können nun im Browser prüfen, ob unserer Webserver auf Anfragen reagiert. Hierzu öffnen wir den Browser und geben in der Adresszeile „http://localhost:3000“ ein.

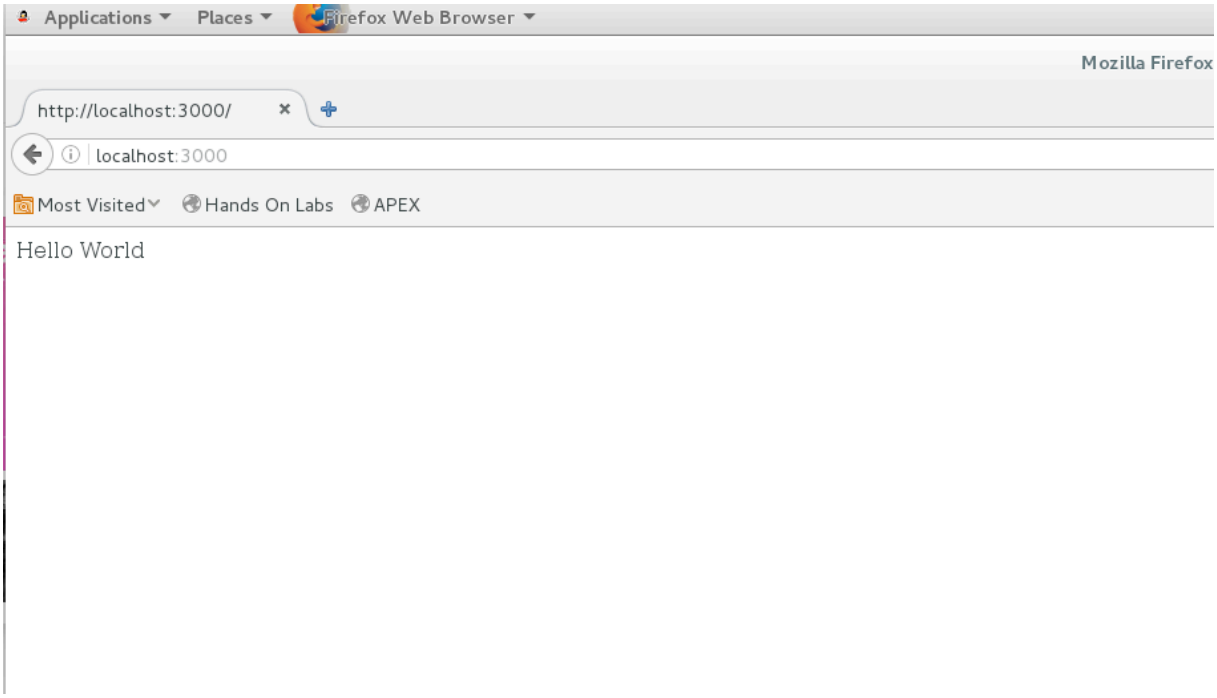

Abb. 5: Ergebnis des minimalen Webservers mit „express“

Anbinden an die Oracle-Datenbank

Da uns die einfache Funktionalität des Webservers noch nicht ausreicht, möchten wir nun auch noch zu einer Datenbank verbinden. Um dies zu tun, installieren wir neben unserem bereits vorhandenen Paket das „oracledb“-Paket, welches von oracle zur Verfügung gestellt wird. Hierzu wird lediglich der InstantClient o.Ä. von Oracle auf dem ausführenden System benötigt, sowie die entsprechenden Bibliotheken zum Kompilieren der notwendigen und mitgelieferten Pakete.

Mit dem Befehl „**npm install oracledb --save“** kann nun die Installation des Datenbanktreibers für Node.js vorgenommen werden. Ein Quellcode-Beispiel zum Testen der Funktionalität des Treibers können wir der Projektseite bei „GitHub“ oder „npmjs.org“ entnehmen:

```
var oracledb = require('oracledb');
oracledb.getConnection(
  {
    user          : "hr",
    password      : "welcome",
    connectString : "localhost/XE"
  },
  function(err, connection)
  {
    if (err) { console.error(err.message); return; }

    connection.execute(
      "SELECT department_id, department_name " +
        "FROM departments " +
        "WHERE manager_id < :id",
      [110],  // bind value for :id
      function(err, result)
      {
        if (err) { console.error(err.message); return; }
        console.log(result.rows);
      });
  });
```

Abb. 6: Beispiels-Code für eine Abfrage der Oracle-Datenbank mit Node.js

Mit dem erneuten Ausführen unseres Skripts können wir sehen, wie die abgefragten Daten in der Konsole ausgegeben werden.

Wie können wir diese beiden vorgestellten Bibliotheken sinnvoll verbinden?

Wir nehmen also den Code der beiden vorherigen Beispiele und verbinden diese:

```
var oracledb = require('oracledb');
var express = require('express');
var app = express();

oracledb.getConnection({
    user: "hr",
    password: "welcome",
    connectString: "localhost/orcl"
  },
  function(err, connection) {
    if (err) {
      console.error(err.message);
      return;
    }
    app.get('/id/:ID', function(req, res) {
      connection.execute(
        "SELECT department_id, department_name " +
        "FROM departments " +
        "WHERE manager_id < :id", [req.params.ID],
        function(err, result) {
          if (err) {
            console.error(err.message);
            return;
          }
          res.send(result.rows);
          console.log(result.rows);
        });
    })
    var server = app.listen(3000, function() {
      console.log('Webserver started on Port ' + server.address().port +
        '!');

    });
  });
```

Abb. 7: Kombinierte Node.js-Applikation mit „express“ und „oracledb“

Wenn wir diesen Quellcode nun ausführen, können wir parametrisierte Abfragen über „express“ an „oracledb“ weiterleiten und die Antwort der Datenbankabfrage wiederum zurückgeben und per Webserver-Response an den Endbenutzer im Browser darstellen. Eben eine einfache REST-API!

Sinnvolle Einsatzmöglichkeiten von Node.js im Oracle Application Express

Da wir nun eine Node.js-Anwendung mit Anbindung an die Oracle-Datenbank haben, und diese sogar als REST-API zur Verfügung stellen können, schadet ein Blick über den Tellerrand hinaus nicht. Mit einem Blick auf die Vielfalt der mittlerweile über 340000 Bibliotheken, die zum Großteil kostenlos und frei verfügbar zur Verfügung stehen, scheinen die Möglichkeiten nahezu unbegrenzt.

Schauen wir uns also einmal mögliche Szenarien an:

- WebSockets anhand des „websocket“-Pakets
- PDF oder Office-Generierung
- Chatbots
- Email-Server
- Automatisierte Test-Suites
- Desktop-Applikationen mit „Electron“

All diese Beispiele haben zumindest einen gemeinsamen Nenner. Die Schnittstelle zu diversen Diensten. Wer Node.js in Kombination mit APEX nutzen möchte, muss sich zunächst Gedanken machen, wie diese Komponenten miteinander kommunizieren sollen. Hierzu ist unser Beispiel zu „express“ und „oracledb“ bereits ein wertvoller Denkanstoß, der bei den meisten Szenarien berücksichtigt werden sollte.

Kontaktadresse:

Kai Donato

MT AG

Balcke-Dürr-Allee, 9

D-40882 Ratingen

Telefon +49 2102 30961-0

E-Mail kai.donato@mt-ag.com

Internet: www.mt-ag.com