Node.js + Oracle-Datenbank = "node-oracledb" = Cool!

Autor: Carsten Czarski, ORACLE Deutschland B.V. & Co KG

Im Januar 2015 hat Oracle die Early Adopter-Phase des ersten offiziellen Node.js-Treiber ***node-oracledb*** *für die Oracle-Datenbank gestartet – seit August liegt der Treiber als Production-Version vor.*

Der Treiber kann von GitHub oder dem OTN heruntergeladen werden und erlaubt die nahtlose Einbindung der Oracle-Datenbank in Node.js Programme. Dieser Artikel stellt Node.js kurz vor, erläutert die Besonderheiten und zeigt dann auf, wie man mit dem neuen Treiber arbeitet und Node.js-Programme für die Oracle-Datenbank schreibt.

Javascript auf dem Server: node.js

Node.js ist eine offene, auf Javascript basierende, Plattform für serverbasierende Anwendungen. Als Grundlage wurde die Javascript-Engine "V8" verwendet, die ursprünglich für Google Chrome entwickelt wurde. Mit Node.js programmiert man also Javascript – auf dem Server.

Node.js (im folgenden "Node" genannt) ist in der Entwicklergemeinde mittlerweile sehr populär, was man vor allem an den zahlreichen Community-Erweiterungen erkennen kann. Für den Package Manager (NPM = *Node Packaged Modules*), mit dem die Erweiterungen der Node-Bibliothek verwaltet werden können, stehen mittlerweile fast 120.000 Pakete bereit.

Das Besondere an Node ist aber das ereignisgetriebene, asynchrone Programmiermodell. Das bedeutet, dass das Programm bei einem blockierenden Aufruf nicht wartet, sondern unmittelbar weiterarbeitet. Der (nun nicht mehr blockierende) Aufruf bekommt eine *Callback-Funktion* übergeben, die nach Abschluß aufgerufen wird. Intern verwendet Node dazu eine *Event-Queue*. Die Abbildungen 1 und 2 verdeutlichen diesen Prozess im Vergleich zum normalen, prozeduralen Modell.

```
myjson = http.get('http://server/path');
console.print(myjson)
console.print('Ende')
```

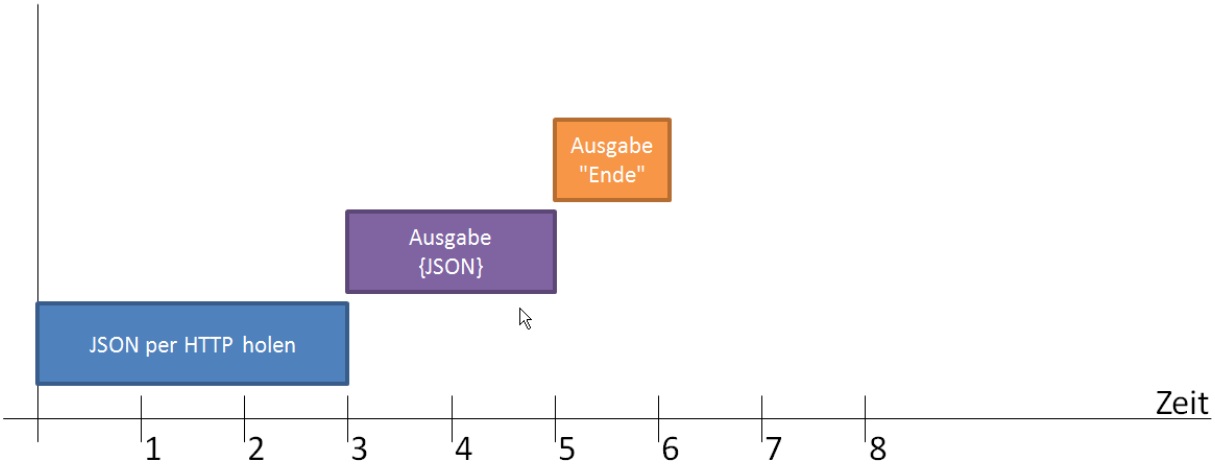

Abbildung 1: Normales, prozedurales Programmiermodell

```
http.get(
  'http://server/path',
  function (result) {
    console.print(result);
  }
);
console.print('Ende')
```

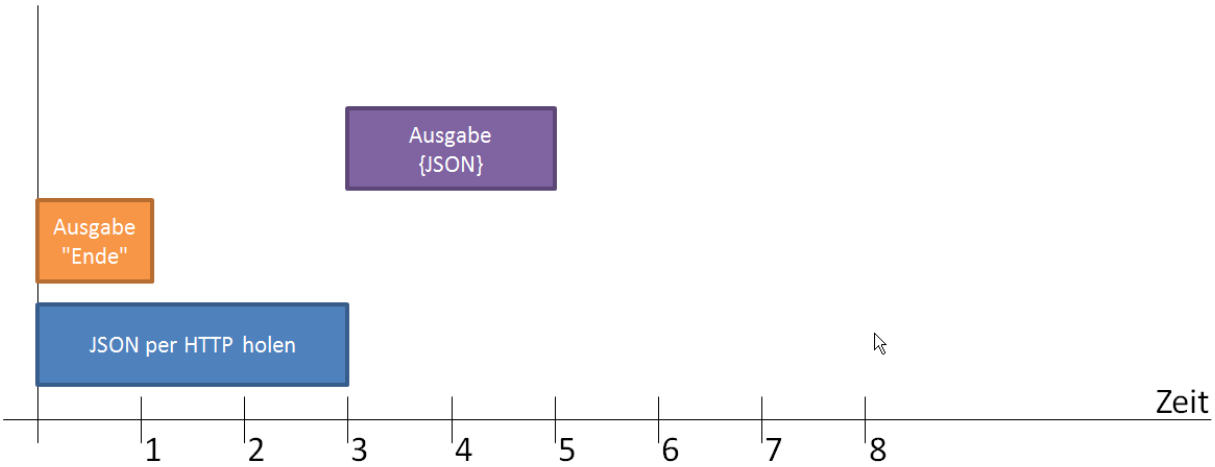

Abbildung 2: Eventgetriebenes Modell von Node.js

Alle blockiereden Aufrufe (Netzwerk, lokales Dateisystem, etc.), werden mit Node asynchron ausgeführt - das Programm kann also weiter arbeiten. Man bemerkt deutlich das Ziel von Node.js: Die CPU soll möglichst ausgelastet werden – der Programmthread soll nur dann warten, wenn gerade *wirklich nichts* zu tun ist. Besonders beliebt ist Node für das Aufsetzen webbasierter Dienste, kommt diese Architektur deren Anforderungen doch besonders entgegen.
Als Entwickler muss man sich an die asynchrone Art der Programmierung erst mal gewöhnen - Code muss anders organisiert werden als in einem klassischen C, Java oder PL/SQL-Programm. Im Internet sind zahlreiche Tutorials und Beispielprogramme für Node.js verfügbar, daher konzentriert sich der Artikel im folgenden auf die Kombination mit der Oracle-Datenbank.

Node.js und die Oracle-Datenbank

Auch für Node braucht man zur Arbeit mit der Oracle-Datenbank einen Treiber. Dieser stellt eine Programmierschnittstelle (API) bereit, mit welcher der Entwickler eine Datenbankverbindung öffnen und SQL oder PL/SQL ausführen kann. Die Oracle-Datenbankzugriffe finden in Node-typischer Manier asynchron statt. Das bedeutet, dass der Entwickler jedem Aufruf (Datenbankverbindung öffnen, SQL Ausführen, Commit, Rollback) einen Callback übergeben muss.
Der Oracle-Treiber **node-oracledb** für die Oracle-Datenbank ist seit Januar 2015 auf GitHub verfügbar (siehe "Weitere Informationen"). Die Schritte zur Installation sind auf der Webseite beschrieben - nach dem Setzen einiger Umgebungsvariablen reicht der Aufruf des Kommandos **npm install oracledb** aus, um den Treiber in die vorhandene Node-Umgebung zu integrieren. Danach kann es auch schon losgehen.

Das erste Programm: Arbeiten mit der Tabelle EMP

Das erste Node-Programm soll schlicht eine Zeile der Tabelle EMP selektieren und auf der Konsole ausgeben.

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

oracledb.getConnection({
    user          : "scott",
    password      : "*****",
    connectString : "dbserver:1521/orcl"
  },

  // :
```

```
  // :

  function(err, connection)  {
    if (err) {console.error(err.message); return;}
    connection.execute(
      "SELECT * from EMP where EMPNO = 7839",
      [],
      function(err, result) {
        if (err) {console.log('%s', err.message); return;}
        console.log(result.rows);
      }
    );
  }
);
// Achten Sie bei der Ausführung auf dieses Kommando!
console.log("Finish????");
```

Die asynchrone Natur der Node-Programme fällt im Code sofort auf. Auf der obersten Ebene befindet sich im Grunde genommen nur ein einziger Aufruf: **oracledb.getConnection**. Diese bekommt als Parameter eine (hier anonyme) Javascript-Funktion als Callback übergeben – was bedeutet, dass deren Code *nach* erfolgreichem Verbindungsaufbau ausgeführt werden soll. In der Zwischenzeit läuft das "Hauptprogramm" weiter. Die Callback-Funktion selbst enthält wiederum asynchrone Aufrufe mit Callback-Funktionen; die innerste schließlich gibt die Ergebnisse aus.

Die letzte Code-Zeile ist interessant: In einem "normalen" Programm wüssten wir, dass diese Ausgabe **Finish???** als Letztes erfolgt. Das asynchrone Node-Programm liefert dagegen ein anderes Ergebnis.

```
$ node emp.js
Finish????
[ [ 7839,
    'KING',
    'PRESIDENT',
    null,
    Tue Nov 17 1981 00:00:00 GMT+0100 (CET),
    5000,
    null,
    10 ] ]
```

Die "Übersetzung" der Oracle-Datentypen auf Javascript Objekte findet im Datenbanktreiber automatisch statt. Das Programm gibt die Javascript-Objekte direkt

auf die Konsole aus – was zu einer JSON-Ausgabe führt. Was der Treiber alles kann, ist in der Dokumentation beschrieben ("Weitere Informationen").

Das zweite Programm: Webserver für die Tabelle EMP

Das zweite Node-Programm macht aus diesem Beispiel einen einfachen Webservice. Dazu braucht es einen HTTP-Server, der mit dem freien Node-Modul **express** jedem Node-Entwickler zur Verfügung steht. Es wird in der Node-Umgebung auf Betriebssystem-Ebene wie folgt installiert.

```
$ npm install express
```

Anschließend kann der einfache Webserver gebaut werden.

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

var pool;

//
// Diese Javascript-Funktion behandelt den HTTP-Request /emp/*
//
function processEmp(req, res) {
  pool.getConnection(function(err, connection){
    connection.execute(
      "select * from emp where (empno=:1 or empno is null)",
      [req.params[0]],
      function(err, results) {
        connection.release(function (err) {}),
        res.send(JSON.stringify(results.rows));
      }
    )
  })
}

//
// Diese Javascript-Funktion startet den Server
//
function startServer () {
  var app =  express();
  app.get ("/emp/*", processEmp);
  var server = app.listen(9000, function () {
    console.log('Table EMP REST Service started.');
  });
}


// :
```

```
// :


//
// Programmstart. Oracle Connection Pool und Webserver starten
//
oracledb.createPool(
  {
    user          : "scott",
    password      : "tiger",
    connectString : "sccloud033:1521/orcl",
    poolMin       : 10,
    poolMax       : 20
  },
  function(err, ppool){
    pool = ppool;
    startServer();
  }
);
```

Das Programm wird gestartet ...

```
$ node emp_rest.js
Table EMP REST Service listening at http://0.0.0.0:9000
```

Nun kann man mit dem Browser den neuen Node-Service nutzen – bei Abruf der URL **/emp/{empno}** bekommt man die entsprechende Zeile der Tabelle EMP im JSON-Format zurückgeliefert.

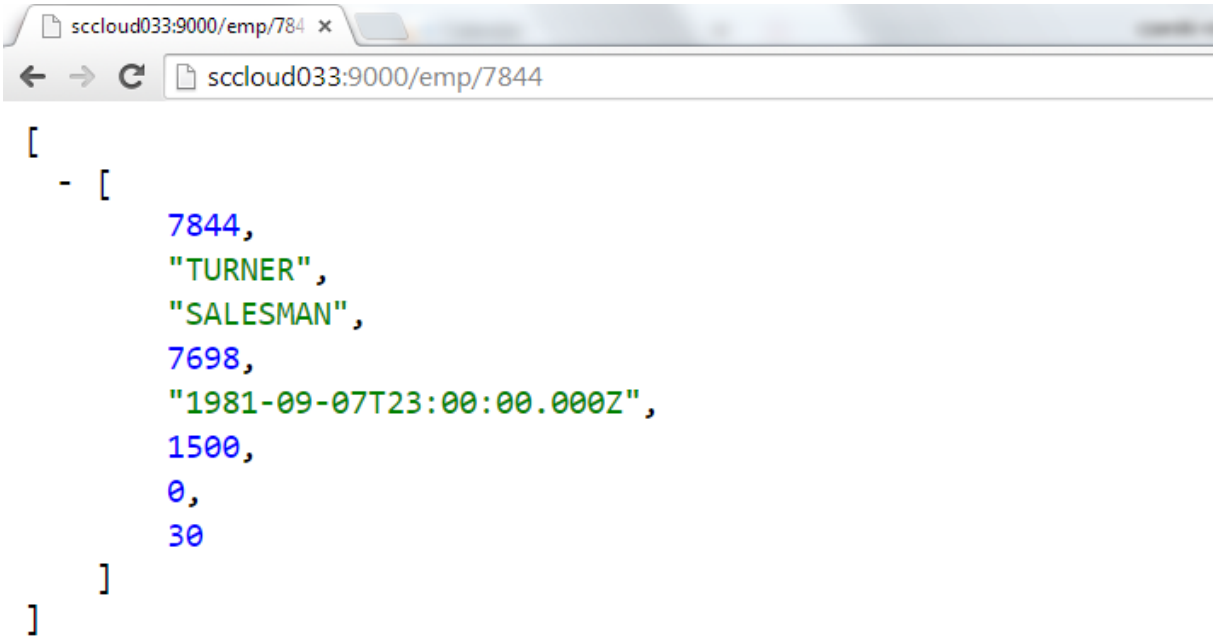

Abbildung 3: Der node.js Service in Aktion!

Aufmerksamen Lesern dürfte aufgefallen sein, dass das zweite Node.js Skript einen Connection Pool zur Verwaltung der Datenbankverbindungen nutzt. Die Attribute **poooMin** und **poolMax** legen dabei fest, mit wievielen Datenbankverbindungen der Connection Pool gestartet wird und wieviele im laufenden Betrieb neu geöffnet werden dürfen. Ist das konfigurierte Maximum erreicht und es wird eine neue Datenbankverbindung angefragt, so muss der Thread warten, bis eine Verbindung frei wird. In der Praxis ist das Nutzen des Connection-Pools eigentlich immer zu empfehlen. Gerade bei einem Webserver, bei dem die Last in einem Moment sehr hoch, in einem anderen sehr niedrig sein kann, stellt der Connection-Pool erst sicher, dass die Datenbank nicht überlastet werden kann.

Ausblick: Weitere Möglichkeiten mit Node.js

Das wirklich besondere Merkmal der Node.js Plattform ist die bereits eingangs erwähnte breite Entwicklercommunity und die große Anzahl verfügbarer Bibliotheken. Mit Node.js sind weit mehr Dinge möglich als das Ausliefern von JSON-Daten per REST-Schnittstelle – drei weitere Möglichkeiten seien hier genannt.

- Node.js bietet eine fertige Unterstützung für die **HTML5 Websocket-Technologie** an; das Aufsetzen eines Websocket-Servers ist recht schnell erledigt. Websockets erlauben es, Daten asynchron vom Server zum Browser zu schicken und so bspw. eine Webseite zu aktualisieren – das ist wesentlich ressourcensparender als ein ständiges Polling.
- Node.js bietet einfach nutzbare Pakete zum Umgang mit Mailservern – das regelmäßige Abrufen von Emails und Speichern in der Datenbank ist mit node.js sehr einfach.
- Das **Webshot**-Paket erlaubt es, einen Screenshot einer Webseite anzufertigen und in der Datenbank abzulegen – serverbasiert und ohne manuelle Interaktion.

Fazit

Mit dem neuen Treiber **node-oracledb** finden Entwickler des freien, Javascript-Frameworks nun auch eine offizielle Unterstützung zum Umgang mit der Oracle-Datenbank. Node.js erlaubt das schnelle und einfache Aufsetzen serverbasierter

Dienste – die umfangreiche Funktionsbibliothek (NPM) macht das Einbinden zusätzlicher Funktionalität einfach.

Weitere Informationen

[1] Node.js Developer Center im OTN
http://www.oracle.com/technetwork/database/database-technologies/node_js/index.html

[2] node-oracledb auf GitHub
https://github.com/oracle/node-oracledb

[3] Node.js im Web
http://nodejs.org

[4] Blog: JSON, REST und die Oracle-Datenbank
http://json-rest-oracledb.blogspot.com

Carsten Czarski
Carsten.Czarski@oracle.com
http://twitter.com/cczarski - http://sql-plsql-de.blogspot.com