Rumor: Implementierung von Übung 1 in Node (JavaScript)

Felix Blechschmitt

Dieses Dokument enthält Informationen über den Aufbau und die Funktionen von rumor.

Build status (by TravisCI)

Getting Started

Das Projekt wurde in der Programmiersprache JavaScript nach dem Standard EcmaScript 6 (ES6) entwickelt. Dabei wurde der Interpreter node in der Version v4.4.4 verwendet.

Installation

Zunächst müssen alle notwendigen Abhängigkeiten installiert werden:

```
$ npm install
```

Usage

Zum Übersetzen der Anwendung sowie zum Starten der einzelnen Applikationen werden NPM-Skripte verwendet, die in der Datei package.json definiert werden.

Ein solches Skript wird über folgendes Kommando ausgeführt, dabei können optional Parameter übergeben werden:

```
$ npm run <SkriptName> [-- <Parameter>]
```

Des Weiterhen werden zusätzliche Shell-Skripte zur Verfügung gestellt mit denen eine Testsuite zum Durchführen von Experimenten gestartet werden kann sowie ein Skript zum sofortigen Beenden aller gestarteter Node-Prozesse. Dieses Skript kann im Falle eines Fehlers ausgeführt werden, falls die Prozesse in einem ungewollten Zustand nicht mehr normal beendet werden können.

Build-Prozess

Node ist nicht in der Lage JavaScript-Dateien, die im ES6-Standard entwickelt wurden, direkt auszuführen. Diese müssen zunächst in den ES5-Standard übersetzt (transpiliert) werden. Hierzu wird babel verwendet.

Der Übersetzungsvorgang kann über das NPM-Skript “build” angestoßen werden. Dabei wird zunächst das möglicherweise bereits vorhandene Verzeichnis “build” geleert bzw. ein solches Verzeichnis erstellt. Anschließend werden alle JavaScript-Dateien von babel in den Standard ES5 übersetzt und inklusive Source Map im Verzeichnis “build” abgelegt.

```
$ npm run build
```

Starten eines Netzwerkknotens

Die Hauptanwendung des Projekts ist ein Netzwerkknoten, der Gerüchte – also Nachrichten – entgegen nimmt und diese an seine Nachbarn verteilt. Ein solcher Knoten kann in einem interaktiven Modus gestartet werden, indem keine Parameter übergeben werden. Dann werden alle benötigten Parameter über einen CLI-Dialog erfragt:

```
$ npm run start
```

Alternativ können die notwendigen Parameter direkt beim Start übergeben werden, sodass die Anwendung ohne weitere Interkation mit dem Nutzer ausgeführt werden kann. Der Parameter "-h" listet alle möglichen Parameter auf:

```
$ npm run start -- -h
Usage: node index.js

      --endpointFilename=[ARG]  path to the endpoints file, leave
                                blank to map ids to local ports
  -g, --graphFilename=[ARG]     path to the graph file defining
                                the network node topology
      --id=[ARG]                ID of this endpoint
  -c, --count=[ARG]             number of receives until a rumor
                                will be believed
  -h, --help                    Display this help
```

So kann zum Beispiel ein Netzwerkknoten mit der ID 5 wie folgt gestartet werden:

```
npm run start -- -g config/graph.dot --id 5 --c 2
```

Dieser Knoten ist dann auf localhost:4004 erreichbar. Der Knoten geht von einer Netzwerktopologie wie in der graphviz-Datei config/graph.dot definiert aus und glaubt ein Gerücht genau dann, wenn er die Nachricht von mindestens 2 Knoten erhalten hat. Wird keine endpoint-Datei angegeben (wie in diesem Beispiel), so werden die Endpunkte als Lokal angenommen und der Netzwerkknoten mit der kleinsten ID erhält den Port 4000. Für jeden weiteren Knoten wird die Portnummer entsprechend hochgezählt. Die minimale Portnummer (default: 4000) kann über die Konstante MIN_PORT im EndpointManager angepasst werden.

Starten des Init-Tools

Das Init-Tool kann dazu verwendet werden, um Kontrollnachrichten an einzelne Netzwerkknoten zu senden. Auch diese Tool kann über einen Dialog interaktiv verwendet weden, um verschiedene Kontrollnachrichten abzusetzen.

```
$ npm run init
Enter command: ?
init: Initialize distribution of a rumor.
stop: Stop one rumor node
stop all: Stop all rumor nodes
exit: Exit this program
?: Display this help
```

Wie in dem vorhergehenden Code-Ausschnitt gezeigt, listet das Kommando "?" eine Übersicht über alle vorhandenen Kommandos auf.

Alternativ kann auch ein Kommando als Parameter übergeben werden, was dafür sorgt, dass dieses ausgeführt wird und sich das Programm anschließend direkt beendet. Dies kann hilfreich sein, um Kontrollnachrichten aus einem Skript automatisiert abzusetzen. Auch hier kann die Verwendung über den Parameter "-h" angezeigt werden:

```
$ npm run init -- -h
Usage: node init.js

  -c, --cmd=[ARG]    Command: "init" | "stop" | "stop all"
      --host=[ARG]   host
      --port=[ARG]   port
  -r, --rumor=[ARG]  the rumor which should be sent
  -h, --help         Display this help
```

Generieren eines Graphen

Eine zufällige Netzwerktopologie kann mithilfe des Tools graphgen, das über das Skript "graphgen" ausgeführt werden kann, erstellt werden. Auch diese Anwendung kann ohne Parameter in einem interaktiven Modus gestartet werden, indem die notwendigen Informationen über einen Dialog erfragt werden. Überlicherweise werden jedoch die benötigten Informationen direkt als Parameter übergeben. Eine Hilfe wird auch hier mit dem Parameter "-h" angefordert.

```
$ npm run graphgen -- -h
Usage: node graphgen.js

  -n                    Number of nodes
  -m                    Number of edges
  -f, --filename=[ARG]  Filename
  -h, --help            Display this help
```

Über den Parameter "-f" bzw. "–filename" (Filename) wird der Pfad angegeben, an dem der erzeugte Graph gespeichert werden soll.

Shellskripte

Zum einfachen Ausführen von Versuchen und als nützliche Hilfe während des Entwicklungsprozesses wurden zusätzliche Shellskripte erstellt. Diese befinden sich im Verzeichnis "scripts". Ausgaben der Skripte erfolgen entweder auf die Standardausgabe oder in spezielle Log-Dateien, welche unter "scripts/logs" abgelegt werden. Benötigt ein Skript spezielle Eingabeparameter, so werden Hinweise zur Verwendung ausgegeben, falls keine Parameter übergeben wurden.

Skript: kill-all.sh

Das Skript "kill-all.sh" sendet das Signal "-SIGKILL" an alle derzeit laufenden node-Prozesse. Es kann dazu verwendet werden, falls sich die Netzwerkknoten in einem ungewolltem Zustand befinden und sich nicht mehr auf "normalem" Weg terminieren lassen.

> Hinweis: Das Skript ist mit besonderer Vorsicht zu verwenden, da andere Node-Prozesse, die auf diesem System laufen, ebenfalls terminiert werden.

Skript: start.sh

Das Skript "start.sh" generiert einen zufälligen Graphen, der als Netzwerktopologie verwendet wird und startet für jeden Knoten des Graphen einen Netzwerkknoten-Prozess mit der entsprechenden ID. Sobald alle Prozesse gestartet wurden, wird automatisch die Verbreitung eines Gerüchtes initialisiert und anschließend die Prozesse beendet. Sobald alle Prozesse beendet wurden, beendet sich das Skript ebenso.

Hinweis: Da das Programm in der aktuellen Version die Terminierung der Ausbreitung von Gerüchten nicht feststellen kann, wird nach dem Initialisieren eine feste Zeit gewartet, bis die Prozesse beendet werden.

```
$ ./scripts/start.sh
./scripts/start.sh n m c graphFilename rumor
```

Der Parameter "n" entspricht der Anzahl an Knoten, "m" bestimmt die Anzahl an Kanten, "c" entspricht der Anzahl an *eingehenden* Gerüchten, bis das Gerücht *geglaubt* wird. Der vierte Parameter "graphFilename" gibt den Dateinamen der Datei an, in der der erzeugte Graph gespeichert wird und "rumor" bestimmt das zu sendende Gerücht.

Skript: startTestSeries.sh

Das Skript "startTestSeries.sh" führt eine gesamte Testreihe durch und legt das Ergebnis in einer neuen Datei unter "scripts/results" ab.

Aufbau

Das Projekt wurde nach einem stark modularisierten Konzept entwickelt. Im Folgenden werden die Verzeichnisstruktur, die einzelnen Komponenten sowie das verwendete Nachrichtenprotokoll beschrieben.

Verzeichnisstruktur

Die Source-Dateien befinden sich alle im Verzeichnis "src" im Hauptordner des Projekts. Prinzipiell kann eine gesamte Node-Anwendung in einer einzelnen JavaScript-Datei entwickelt werden. Dies ist jedoch aufgrund fehlender Übersicht nicht empfehlenswert. Daher wurde das Projekt in verschiedene Komponenten unterteilt, welche in einzelnen Dateien entwickelt wurden. Ähnliche Module wurden dabei in Unterverzeichnissen gruppiert:

- /lib: enthält wiederverwendbare Module, die anwendungsübergreifend verwendet werden können
- /parser: enthält Komponenten zum Einlesen von Dateien bestimmter Formate
- /tools: enthält zusätzliche Skripte, die als eigenständige Node-Anwendungen gestartet werden können

Die Datei "index.js" ist der Einstiegspunkt der Anwendung. Darin werden die einzelnen Komponenten miteinander verknüpft und der Netzwerkknoten gestartet.

Komponenten

Das folgende UML-Klassendiagramm zeigt einen Überblick über die einzelnen Komponenten der Anwendung.

Die Klasse Server ist eine einfache Implementierung eines TCP-Servers, der bei eingehenden Nachrichten eine vorher definierte Funktion aufruft. Ein Aufruf der Methode "listen(callback)" startet den Server und registriert die übergebene Funktion als Callback, welches bei eingehenden Nachrichten aufgerufen wird. Die Methode "stop()" beendet den Server.

In der Basisklasse "ServerLogic" wird das Behandeln eingehender Nachrichten definiert. Sie kennt die verbundenen Endpunkte – also die Nachbarknoten – und besitzt eine Referenz auf den gestarteten Server. Verbunden werden beide Kompomenten, indem dem Konstruktor der ServerLogic die Referenz auf den Server übergeben wird. Anschließend wird die Methode "onReceiveData" der ServerLogic der "listen"-Methode des Servers als Callback übergeben. Auf diese Weise wird die Methode "onReceiveDate" bei jeder eingehenden Nachricht ausgeführt.

Handelt es sich bei einer eingehenden Nachricht um eine Kontrollnachricht, so wird diese von der ServerLogic direkt behandelt. Alternativ wird die *abstrakte* Methode "_runAlgorithm" aufgerufen, welche von der

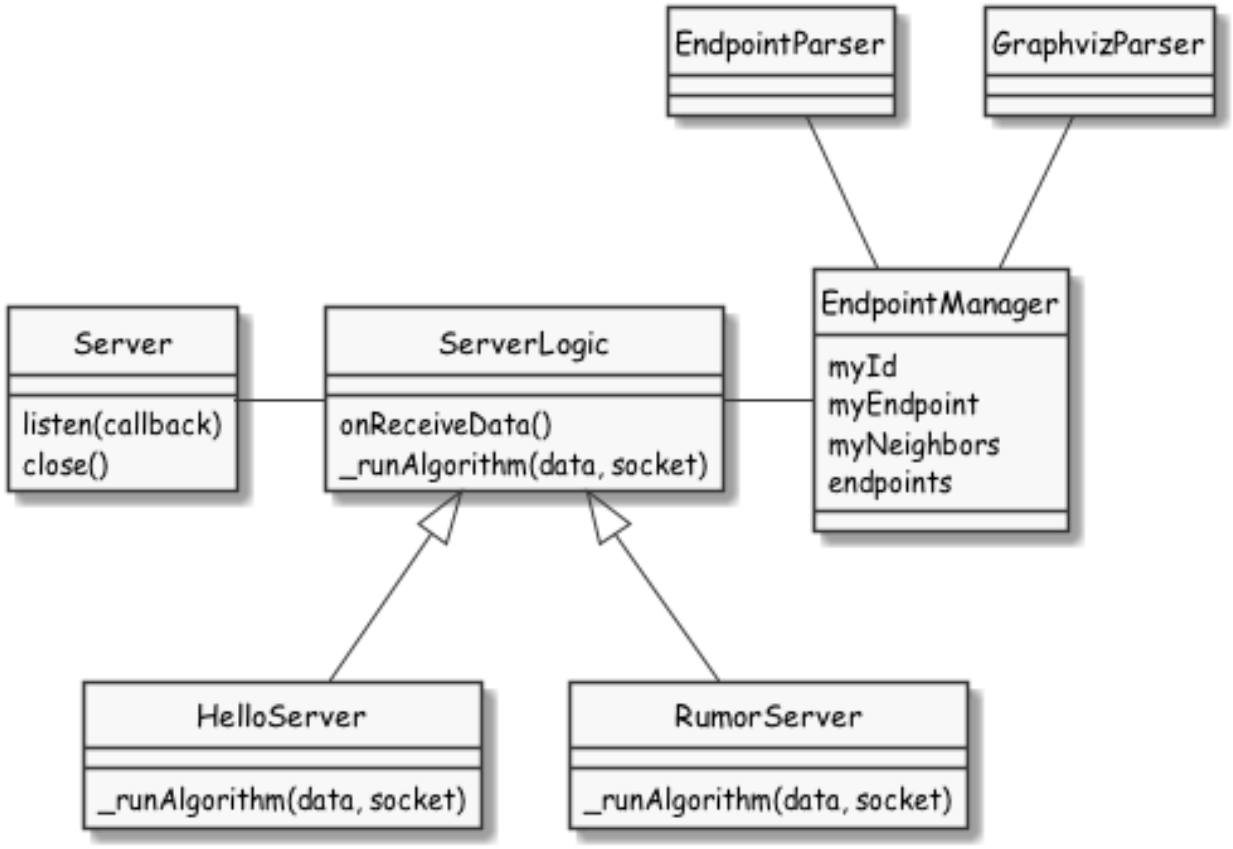

Figure 1: Komponenten der Anwendung

konkreten Implementierung “HelloServer” (bei Aufgabenteil a) bzw. “RumorServer” (späterer Aufgabenteil) überschrieben wird.

> Hinweis: In JavaScript gibt es eigentlich keine abstrakten Klassen bzw. Methoden. Daher wird die Methode “_runAlgorithm" in der Basisklasse einfach mit einem leeren Rumpf definiert.
>
> Speziallfall - Knoten als ein Prozess: In der Aufgabenstellung war gefordert, dass der Knoten als ein Prozess und somit die Bearbeitung einer eingehenden Nachricht als eine atomare Aktion ausgeführt werden soll. Um dies in node zu realisieren, wurde eine Semaphore verwendet, die alle anderen eingehenden Nachrichten blockiert, während eine Nachricht behandelt wird.

Promise-Wrapper

Ein bekanntes Problem bei Anwendungen in JavaScript ist der Umgang mit verschachtelten Callbacks, also Funktionsaufrufe, die als Parameter eine Callback-Funktion übergeben, in der ein weiterer Funktionsaufruf mit einer weiteren Callback- Funktion ausgeführt wird. Dies wird häufig auch als sogenannte *callback hell* bezeichnet. Durch die Verwendung des async-await Sprachfeatures von JavaScript lassen sich asynchrone Ausführungen wesentlich übersichtlicher gestalten.

Dieses Feature kann auf alle Funktionen angewendet werden, die statt der Verwendung eines Callbacks ein *Promise* zurück geben. Aus diesem Grund wurden für Standardfunktionen, wie zum Beispiel das Einlesen/Schreiben einer Datei oder das Lesen von der Kommandozeile, eine Wrapper-Funktion geschrieben, welche ein Promise zurückgibt, welches durch ein Callback aufgelöst wird.

Diese Wrapper befinden sich im Verzeichnis “lib”.

Protokoll

Zum Austausch der Nachrichten zwischen den einzelnen Netzwerkknoten wird das JSON-Format verwendet. Eine Nachricht hat dabei folgende Komponenten:

- type: Typ der Nachricht (rumor oder control)
- msg: Inhalt der Nachricht
- from: ID des Absenderknotens (optional)

Handelt es sich um eine Kontrollnachricht, so enthält das Messagefeld die Aktion (“STOP” zum Beenden des Knotens bzw. “STOP ALL” zum Senden der “STOP ALL” Nachricht an alle Nachbarn und zum anschließenden Beenden des Knotens).

Alle Nachrichten werden über TCP übertragen. Auch wenn dies – aufrund der TCP-Übertragung – nicht explizit notwendig wäre, werden alle empfangenen Nachrichten nach dem Erhalt mit einem leeren JSON-Objekt bestätigt.

Beispiel einer Nachricht

```
{"msg":"g.2","from":22,"type":"rumor"}
```

Beispiel einer Kontrollnachricht

```
{"msg":"STOP","type":"control"}
```

Tests

Die dynamische Typisierung von JavaScript birgt die Gefahr von häufigen Laufzeitfehlern. Aus diesem Grund ist es gerade bei der Verwendung einer solchen Programmiersprache besonders wichtig, die Komponenten der Anwendung mit automatisierten Tests auf korrekte Funktionalität zu überprüfen. Außerdem helfen gerade Unit-Tests bei der Entwicklung von einzelnen Komponenten, da man sich durch diese von der korrekten Funktionalität der Komponente überzeugen kann, da diese dann gezielt ohne den Kontext der gesamten Anwendung ausgeführt werden kann.

Automatisierte Tests

Einige Hilfsfunktionen sowie kleinere Module der Anwendung, wie zum Beispiel der Parser für die Endpoint- bzw. graphviz-Datei oder der Kantengenerator, wurden mit Unit-Tests versehen. Die Tests befinden sich im Verzeichnis "tests" des Projektordners.

Die Tests können mithilfe des NPM-Skripts "test" ausgeführt werden:

```
$ npm run test
```

Experimente

Zur Durchführung der Experimente wurden die oben beschriebenen Skripte "start.sh" sowie "startTestSeries.sh" verwendet. Die gesamte Versuchsreihe wird dabei in der Datei config/testSeries definiert. Jede Zeile dieser Datei entspricht einem Versuch und definiert die notwendigen Parameter. Das Shell-Skript "startTestSeries.sh" liest diese Datei aus und startet die einzelnen Versuche.

Beschreibung

Es wurden insgesamt 21 Versuche durchgeführt, wobei jeweils drei Versuche mit den gleichen Parametern in Folge ausgeführt wurden. Anschließend wurde genau einer der drei Parameter verändert.

Auswertung

In der unten stehenden Tabelle kann das Ergebnis der Versuchsreihe eingesehen werden. Auffällig ist, dass selbst bei gleichen Parametern die Anzahl an Knoten, die das Gerücht glauben, teilweise stark variiert (siehe zum Beispiel Versuche 10, 11 und 12). Dies lässt sich jedoch dadurch erkären, dass bei jeder Durchführung eines Versuchs ein neuer zufälliger Graph erstellt wird.

Nr.	Nodes	Edges	BelieveCount c	Rumor	InitNode	Believers	Percentage
1	10	15	2	a.1	1	7	70%
2	10	15	2	a.2	1	8	80%
3	10	15	2	a.3	1	7	70%
4	10	15	3	b.1	1	3	30%
5	10	15	3	b.2	1	4	40%
6	10	15	3	b.3	1	3	30%
7	10	15	6	c.1	1	2	20%
8	10	15	6	c.2	1	0	0%
9	10	15	6	c.3	1	1	10%
10	10	20	3	d.1	1	5	50%
11	10	20	3	d.2	1	8	80%

Nr.	Nodes	Edges	BelieveCount c	Rumor	InitNode	Believers	Percentage
12	10	20	3	d.3	1	4	40%
13	50	51	5	e.1	1	5	10%
14	50	51	5	e.2	1	3	60%
15	50	51	5	e.3	1	6	12%
16	50	90	5	f.1	1	15	30%
17	50	90	5	f.2	1	16	32%
18	50	90	5	f.3	1	14	28%
19	100	190	5	g.1	1	14	14%
20	100	190	5	g.2	1	13	13%
21	100	190	5	g.3	1	12	12%