Grundlegender Aufbau des Javascript Universums

v8	Javascript Compiler (z.B. in Node.js und Chrome genutzt)
nvm	Node Versions Manager (Verwaltung parallel installierter Node.js Versionen)
node.js	Umgebung für Javascript (Compilieren, Ausführen, Debuggen, Installation von Werkzeugen, ...), stellt zusätzlich eine eigene Standardbibliothek bereit welche in anderen Umgebungen z.B. Browser nicht verfügbar ist.
npm	Node Package Manager (Verwaltung der zu installierenden Werkzeuge und Pakete)
Javascript	Hochsprache für Clients die im Browser laufen, standardisiert über ECMAScript
Typescript	Javascript Spracherweiterung. Jedes Javascript stellt gültiges Typescript dar. Typescript wird compiliert nach ECMAScript
Angular	Javascript Framework
Browser APIs	Konstrukte im Browser welche auf Javascript aufbauen
Third party APIs	APIs anderer Webseiten wie Facebook und Twitter
JavaScript libraries	Module von Drittanbietern z.B. jQuery oder React

Grundlegender Aufbau des Javascript Universums (cont)

JavaScript frameworks	Frameworks zum Schreiben von Apps z.B. Angular oder Ember im Unterschied zur Lib geht hier die Kontrolle vom Entwickler auf das Framework über. Das Framework ruft Entwicklerkode auf, nicht umgekehrt.

Quellen:
* https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

nvm Grundlagen

https://github.com/coreybutler/nvm-windows	Window Installation mittels Installer
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash	Linux Installation mittels curl + bash
nvm install <nodeVersion>	Installiert die gewünschte Node Version z.B. nvm install 6
nvm use <nodeVersion>	Aktiviert die angegebene Version und deaktiviert alle anderen
nvm ls	Zeigt alle aktuell installierten Node Versionen an

Node.js Grundlagen

Node.js ist eine Entwicklungs- und Laufzeitumgebung für Javascript Es wird ähnlich wie das Java SDK mit einer Standardbibliothek ausgeliefert deren Module in der Node API Dokumentation beschrieben sind. Diese Bibliothek steht nur zur Entwicklungszeit oder bei der Ausführung auf dem Server (auf welchem Node.js läuft) zur Verfügung. Zur Laufzeit im Browser steht diese Bibliothek nicht zur Verfügung.

By **Huluvu424242**
(FunThomas424242)

cheatography.com/funthomas424242/
github.com/Huluvu424242

Published 10th March, 2019.
Last updated 5th July, 2020.

Node.js Grundlagen (cont)

Node.js nutzt zum Compilieren der Javascript Dateien den Compiler V8. Dieser compiliert direkt in Maschinenkode (also früher z.B. x86 oder MIPS).
Im Chrome Browser kommt der V8 Compiler ebenfalls zum Einsatz.
Node.js wird stets in 2 Releases bereitgestellt (April und Oktober Release). Ein Release trägt dabei im Namen den Zusatz LTS (Long time support). Dieses eignet sich für den produktiven Einsatz. Für dieses Release werden ab Veröffentlichung ca. 30 Monate Bugfixes und sonstige Unterstützung angeboten. Das andere Release wird nur 6 Monate supported.
Grundsätzlich läuft Entwicklerkode in Node.js nur Singlethreaded ab aber Node.js selbst ist Multithreaded. Dem Entwickler steht halt nur ein Thread zur Verfügung. Um auf etwas zu warten oder zu pausieren stehen Konstrukte wie: setTimeout und setIntervall zur Verfügung. Der Trick dabei ist, dass dies Konstrukte nicht blockieren.

Grundlegende Befehle

```
# Node.js Version heraus  finden
>node -v
# Node.js im intera ktiven Mode starten
>node
# Node.js zur Ausführung einer Datei starten
>node app.js
```

AngularJS Bindungsarten

?	wird angehängt um optionale Parameter zu definnieren z.B. '@?'
'='	2 Wege Binding (Properties im Scope)
@	fixe Zeichenkette (String), Ausdrücke wie ('ID' | translate) funktionieren nicht
&	callback Funktion
<	1 Wegebindung in Komponenten

Spezifische Angular Probleme

Lokalisierung (l10n) und i18n

Die Lokalisierung von Angular Anwendungen entsprechend den Angular Vorgaben erfolgt durch separat zu bauende Anwendungen für jede Sprache eine (nebst neuer Compilierung und Bündelung). Die Texte werden dabei ausgelagert in separaten Dateien gehalten. Die Zuordnung der Texte zur Verwendung im Programm erfolgt über IDs. Sollen im HTML bestimmte Attribute mit lokalisierten Texten beladen werden so sind nach einer vordefinierten Syntax die Schlüssel dort anzugeben. Weiterhin muss bekanntgegeben werden welches Attribut ein i18n Attribut ist. Dazu wird das Attribut wiederholt und der Prefix i18n- vorangestellt. Handelt es sich bei dem Attribut leider um einen Input Parameter (@Input) welcher durch [] ausgedrückt wird, so funktioniert diese Ersetzung nicht. Hier hilft nur das Auflösen der [] und die Verwendung der Canonischen Form. Dadurch funktioniert der i18n Mechanismus als wäre der Attributwert ein Text und außerdem wird der Wert an die Komponente als Input Wert reingereicht.. Problemösung siehe hier: https://stackoverflow.com/questions/43202600/how-to-use-angular2-i18n-x-to-localize-properties-constant-value?answertab=votes#tab-top

Node Module - ausgewählte Übersicht

Testframeworks
~, stellen oft einen Testrunner und eine Assertion Library zur Verfügung.
* Jasmine ältestes Framework, aus der Browserumgebung kommend nun durch Node auch im Serverumfeld gelandet. Beinhaltet Testrunner und Assertions.
* Mocha das zweit älteste Framework, von Node.js her kommend lässt es sich inzwischen auch im Browser verwenden (https://medium.com/dailyjs/running-mocha-tests-as-native-es6-modules-in-a--browser-882373f2ecb0).
* Ava, ein recht junges Framework welches Wert auf konsequente asynchrone Testausführung legt. In Spezialfällen (Singleton etc.) schwieriger handhabbar.
Assertion Frameworks
* Assert Modul: Wird direkt von Node.js bereitgestellt
* assertthat: Bietet ein Fluent API

Node Module - ausgewählte Übersicht (cont)

* should.js Schöne Schreibweise aber in Spezialfällen bei denen das actual undefined ist, ist der Einsatz schwierig.
* expect.js Schick

Quellen im Netz

https://www.thenativeweb.io/learning/techlounge-nodejs	über 20 Videos zu Node.js

Die C't Serie der Techlounge stellte die initiale Grundlage für dieses Cheat Sheet dar.

Online IDEs

runkit	Node.js Spielwiese
Gitpod	IDE für Github Projekte
Plunker	IDE für Javascript
Stackblitz	IDE für Javascript
Play with Docker	Spielwiese für Docker Container

Für eine aktuelle IDE Liste schauen Sie bitte unter: https://www.cheatography.com/funthomas424242/cheat-sheets/plattform-bergreifend-arbeiten/

npm Grundlagen

npm out	zeigt outdated Abhängigkeiten (upgradeable)
npm list	Listet alle installierten Pakete auf
npm update	Aktualisierung der installierten Pakete
sudo npm install <paketname> -g	Globale Installation eines Paketes Systemweit so das es als Kommando ausgeführt werden kann

npm Grundlagen (cont)

npm install <paketname> --save-dev	Projektinterne Installation eines Paketes im Unterordner node_modules

npm ist der Node Paketmanager

package.json Grundlagen

Versionierung

Details unter: https://flaviocopes.com/package-json/#package-versions

^: Die angegebene Version mit Minoränderungen aber keine Majoränderungen
~: Die angegebene Version mit Patches
***** : Alle Versionen
>: Alle Versionen größer als die angegebene
>=: Alle Versionen größer oder gleich der angegebenen
<=: Alle Versionen kleiner oder gleich der angegebenen
X.Y.Z: Exakte Versionsangabe und nur diese Version wird benutzt
latest: Stets die aktuellste Version ist zu laden.
<: Alle Versionen kleiner der angegebenen

Node.js - Http Server starten

```
const http = require('http');
const server = http.c rea teS erv er( (req, res) =>
{
    res.wr ite Hea d(200, {
        'Co nte nt- Type': 'text/ html',
   });
    res.wr ite ('Hallo Httpl');
    res.end();
});
server.li ste n(3000, () => {
    con sol e.l og( 'Server lauscht auf Port
3000');
});
```

Node.js Grundlagen - Module

Module

Die weltweite Registry für npm Module befindet sich unter: npmjs.com

Wer hier Module ablegen will benötigt ein Login und muss sich registrieren.

```
# Ein eigenes Node.js Modul auf npmjs.com veröff -
ent lichen
>npm login
>npm publish
# Ein eigenes Node.js Modul von npmjs.com zurück -
ziehen
# nur 24 h lang möglich - danach geht es nur noch
über den Support
# lessons learned aus dem left-pad disaster
>npm unpublish [<@ sco pe> /]< pkg >[@ <ve rsi on>]
# Das Grundg erüst für ein eigenes Node.js Modul
interaktiv erstellen
>npm init
```

* Modul- (oder Projekt) **Metadaten** werden in der Datei `packag -e.json` verwaltet (wie bei maven die pom.xml).

Hier wird der Name und die Version des Moduls angegeben, seine Abhängigkeiten, seine Abhängigkeiten zur Entwicklungszeit und es gibt einen Script Abschnitt in dem Aliase für die Ausführung von Befehlen hinterlegt werden können z.B. um bestimmte Build Schritte anzustoßen wie `npm run test` um die Unittests auszuführen.

* Es existiert weiterhin ein Verzeichnis `node_m odules` in dem alle Abhängigkeiten des Modules abgelegt werden (auch die transitiven).

* **Ladereihenfolge:** Ein zu ladendes Modul wird von Node.js wie folgt gesucht; 1, Integriertes Modul (Module der Node.js Standardlib) 2. installierte Module (durch npm install im node_modules Verzeichnis abgelegt) 3. eigenes, entwickeltes Modul. - Diese Reihenfolge soll sicherstellen, dass kein Schadkode als transitive Abhängigkeit eingeschleußt werden kann.

* **Versionierung** Node.js Module verwenden eine semantische Versionierung.

Fragen - offene Punkte - FAQ

Hier lädt node.js das http Modul und gibt eine Referenz darauf zurück

http = require('http');

Frage: Ist require nur in node.js definiert oder auch in Javascript also auch im Browser und ist es zustätzlich auch in angular oder typescript nutzbar?

Antwort: require() is not part of the standard JavaScript API. But in Node.js, it's a built-in function with a special purpose: to load modules. (Quelle: https://stackoverflow.com/questions/9901082/what-is-this-javascript-require)

Fallstricke beim Testen

Falsche Kombination Runner - Assert

Die Testrunner sorgen dafür, dass die Tests aus den Testdateien gefunden und ausgeführt werden. Die Assertion Frameworks prüfen Testergebnisse und werfen bei nicht zutreffenden Bedingungen eine Exception. Diese Exception muss der Testrunner erwarten, sonst bleibt der Test grün. Es ist also wichtig, dass das genutzte Assertion Framework eine zum Testrunner passende Exception wirft. Das ist in der Regel so, in Sonderfällen muss man aber nachlesen.

Asynchrone Tests

Als Asynchrone Tests bezeichnen wir hier Tests welche mindestens eine Zeile enthalten in der eine Promise entsteht, das Kommando also sofort zurückkehrt, das Ergebnis aber noch nicht vorliegt sondern nur die Promise als Versicherung, dass es mal irgendwann ein Ergebnis geben wird.

Solche Tests werden in der Regel mit async gekennzeichnet und es wird in aller Regel auch ein callback (done) hineingegeben, welches man selbst nach dem letzten Assert aufrufen muss. Ruft man das callback nicht auf zum Beispiel weil man den Test gar nicht als async gekennzeichnet hat, betrachtet der Testrunner den Testfall nach dem letzten Assert als beendet auch wenn noch einige Promises gar nicht aufgelöst wurden. Er kennzeichnet den Test grün und wenn dann später noch eine Promise resolved wird - ändert dies das Testergebnis nicht mehr.. Daher ist es sinnvoll pauschal alle Tests in der async Variante zu implementieren und idealerweise eine solche Abarbeitung zu erzwingen. Bei Mocha geht dies mit der Option --async-only