Masterarbeit

Statische Typsysteme in JavaScript

Entwicklung eines Transpilers von Flow nach TypeScript

TBA

Zur Erlangung des akademisches Grades eines
Master of Science

angefertigt von Jonathan Gruber (68341)

Fakultät Informatik, Mathematik und Naturwissenschaften
Studiengang: Master Informatik (16INM/VZ)

Erstprüfer	Prof. Dr. rer. nat. habil. Frank
Zweitprüfer	M. Sc. Michael Lückgen

ausgegeben am 11. Juni 2019
abgegeben am TBA 2019

Inhaltsverzeichnis

1 Motivation

1.1 JavaScripts Typsystem

Historische Entwicklung usw blabla. Warum ist das Typsystem eher nicht so gut?

1.2 Sinnhaftigkeit statischer Typsysteme

Überleitung: genau deswegen brauchen wir statische Typsysteme. Da hätten wir zwei…

1.3 Zielsetzung der Arbeit

2 Grundlagen

... und diese werden nun sogleich näher beschrieben:

2.1 Statische Typsysteme in JavaScript

2.1.1 Flow

Flow beschreiben (und zwar mit entsprechender Fachsprache)

Basistypen

Hilfstypen

Deklarationen

2.1.2 TypeScript

TS beschreiben (und zwar mit entsprechender Fachsprache)

2.2 Transpiler

Was macht eigentlich so ein Transpiler? Hier Theorie (AST etc.)

2.3 Babel

3 Ziel- und Anforderungsanalyse

3.1 Beschreibung der Ausgangslage

Was geht so bei TeamShirts?

3.2 Ziele der angestrebten Migration zu TypeScript

3.3 Evaluation bestehender Transpilierungs-Ansätze

Erläutern, dass es da schon eine Handvoll Ansätze auf GitHub gab, aber die alle nicht einsatzbereit waren.

3.4 Anforderungen an den Transpiler

Was soll der angestrebte Transpiler können?

1. semantisch äquivalente Übersetzung des Codes
2. korrekte Übersetzung aller Flow-Typen möglichst ohne Verlust von Typinformation
3. Behandlung von Spezialfällen
4. Anpassung der Typimporte

5. exakte Beibehaltung der Formatierung und Kommentare der Codebase

4 Umsetzung des Transpilers

4.1 Software-Architektur

4.2 Entwicklungsprozess

Irgendwo sollte wohl geschrieben werden, dass TypeScript, TDD usw. verwendet wurde, um das Plugin zu bauen…

4.3 Implementierung als Babel-Plugin

4.3.1 Transpilierung der Basistypen

4.3.2 Transpilierung der Hilfstypen

4.3.3 Transpilierung der Deklarationen

4.3.4 Formatierung des Ausgabequelltexts

Prettier, synchronisieren der Leerzeilen und Kommentare beschreiben usw.

4.3.5 Weitere Optimierungen

Mapping der Importe (verschiedene Typnamen in Flow und TS), Umwandlung der Decorators usw.

4.4 Kommandozeilenprogramm

5 Ergebnisse der Migration

5.1 TODO: Wie gut hat das generell geklappt?

Aufwand manueller Nacharbeit

5.2 Semantische Äquivalenz des Ausgabeprogramms

5.3 Erkennung neuer Typfehler

5.4 Verfügbarkeit und Qualität externer Typdefinitionen

5.5 Performance der Typüberprüfungen

5.6 Vergleich des Transpilers mit konkurrierenden Ansätzen

6 Zusammenfassung

Online-Quellen

[1] Microsoft Corporation. *TypeScript - JavaScript that scales*. 2018. URL: https://www.typescriptlang.org/ (besucht am 22. 11. 2018) (siehe S. VI).

[2] Stack Exchange Inc. *Stack Overflow Annual Developer Survey*. 2018. URL: https://insights.stackoverflow.com/survey/2018 (besucht am 21. 11. 2018) (siehe S. VIII).

[3] Facebook Inc. *Flow: A Static Type Checker For JavaScript*. 2018. URL: https://flow.org/ (besucht am 22. 11. 2018) (siehe S. VI).

[4] Google LLC. *Angular*. 2018. URL: https://angular.io/ (besucht am 22. 11. 2018) (siehe S. VIII).

[5] Sebastian McKenzie und other contributors. *Babel - The compiler for next generation JavaScript*. 2018. URL: https://babeljs.io/ (besucht am 22. 11. 2018) (siehe S. VI).

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcode-Listings

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst habe. Ich versichere, dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahren gewesen ist.

Leipzig, den 30. Juni 2019

_____

Jonathan Gruber

A TBA

A.1 Expose

A.1.1 Statische Typsysteme in JavaScript

Derzeit existieren zwei Ansätze, um statische Typsicherheit für JavaScript-Code zu erzielen:

1. **Flow** [3] ist ein von Facebook entwickeltes Werkzeug, um den Quelltext mittels Typannotationen hinsichtlich der Korrektheit der so umgesetzten Typisierung zu überprüfen. Die Annotationen und eigenen Typdefinitionen werden in den Quelltext eingefügt und müssen vor dessen Auslieferung durch einen Transpilierungsprozess entfernt werden, sodass wieder standardkonformer JavaScript-Code entsteht. Dieser Schritt kann beispielsweise durch *Babel* [5] realisiert werden.
2. **TypeScript** [1] ist eine von Microsoft entwickelte, vollständige Programmiersprache, welche eine Obermenge von JavaScript darstellt. Ein stark ausgeprägtes Typsystem ist elementarer Bestandteil der Sprache. Im Gegensatz zu Flow wird TypeScript-Code durch einen Compiler in JavaScript übersetzt, welcher auch auch die statische Typprüfung vollzieht. Diese kann natürlich auch simultan zur Arbeit des Programmierers innerhalb der integrierten Entwicklungsumgebung ablaufen.

```
// Beispiel in Flow:
// @flow
type Vector = {
  +x: number,                          // + bedeutet "schreibgeschützt"
  +y: number,
}

function vectorLength<T: Vector>(vec: Vector): number {
  return Math.sqrt(Math.pow(vec.x, 2) * Math.pow(vec.y, 2));
}

const vec: Vector = { x: 2, y: 5 };
vec.x = 3;                             // Fehler: Attribut x ist schreibgeschützt

vectorLength({ x: 2, y: 5 });          // In Ordnung
vectorLength({ x: 2, y: 'string' });   // Fehler: Attribut y ist nicht vom Typ `number`
vectorLength({ y: 5 });                // Fehler: Attribut x fehlt
vectorLength({ x: 2, z: 5 });          // Fehler: Attribut z ist nicht Teil des Typs


// ---------------------------------------------------------------------------------------

// Analoges Beispiel in TypeScript:
type Vector = {
  readonly x: number,
  readonly y: number,
}

function vectorLength<T extends Vector>(vec: Vector): number {
  return Math.sqrt(Math.pow(vec.x, 2) * Math.pow(vec.y, 2));
}

const vec: Vector = { x: 2, y: 5 };
vec.x = 3;                             // Fehler: Attribut x ist schreibgeschützt

vectorLength({ x: 2, y: 5 });          // Korrekt
// Rest analog...
```

Quellcode 1: Vergleich der zwei Ansätze für statische Typisierung von JavaScript mit Flow (oben) und TypeScript (unten).

Wie Quelltext 1 verdeutlicht, haben Flow und TypeScript eine ähnliche, wenn auch in manchen Fällen nicht völlig identische Syntax:

- Vgl. „`+x: number`“ vs. „`readonly x: number`“
- Vgl. „`<T: Vector>`“ vs. „`<T extends Vector>`“

Innerhalb des eCommerce-Unternehmen **TeamShirts**[1] gibt es strategische Überlegungen, die bestehenden Frontend-Projekte, die im Moment allesamt auf React und Flow basieren, nach TypeScript zu migrieren. Grund hierfür ist die Annahme, dass TypeScript langfristig zukunftsträchtiger und besser geeignet als Flow sein könnte. Es gibt einige Argumente, die für TypeScript sprechen:

- **Performance** — Während derzeit noch keine verlässlichen Zahlen vorliegen, um diese These stichhaltig zu untermauern, wird TypeScript von vielen Entwicklern im Vergleich zu Flow als schneller hinsichtlich der Typprüfung beschrieben.
- **Vorgefertigte Typisierungen** — Jede Software basiert im Normalfall auf externen Bibliotheken. Sowohl für TypeScript als auch für Flow gibt es von der Gemeinschaft verwaltete Projekte[2], welche die Typisierung dieser Abhängigkeiten bereit stellen. Hierdurch wird beispielsweise die Autovervollständigung in integrierten Entwicklungsumgebungen erheblich verbessert. Jedoch bietet TypeScript dabei eine deutlich vollständigere Unterstützung als Flow[3].
- **Kontinuität / Stabilität** – Es besteht Unsicherheit darüber, ob Facebook und Microsoft auf lange Sicht daran interessiert bleiben werden Flow bzw. TypeScript weiterzuentwickeln und zu warten. Da TypeScript als eigenständige Programmiersprache in vielen Bereichen eingesetzt wird und stetig an Popularität gewinnt [2], erscheint es jedoch unwahrscheinlich, dass Microsoft dieses Produkt in naher Zukunft einstellen wird. Angular [4] ist beispielsweise ein großes, erfolgreiches Framework, welches in TypeScript geschrieben ist und die Verwendung der Sprache fördert.
- **Community / Dokumentation** – Auch die Größe und Dynamik der „*Community*“ ist ein relevanter Aspekt, da eine große Nutzerzahl bei Open-Source-Projekten einen hohen Grad an Aktivität bedeutet. Offene Fragen und Probleme können durch Online-Recherche meistens schnell beantwortet werden, da das Problem oft bereits behandelt wurde. Ein einfacher, aber wichtiger Aspekt ist darüber hinaus die Aktualität und Qualität der Dokumentation.

[1] Ein Tochterunternehmen der Spreadshirt AG. Kunden haben die Möglichkeit Textilien und andere geeignete Gegenstände wie Taschen, Mützen, Tassen etc. mit eigenen und vorgefertigten Bildmotiven zu bedrucken. Hierfür können die Produkte in einer Webanwendung selbstständig gestaltet und im Anschluss bestellt werden.

[2] Flow Typed für Flow und Definetely Typed für Typescript.

[3] Vgl. GitHub-Projekte in Fußnote 2.

Ziel der angestrebten Masterarbeit ist damit zunächst eine Analyse hinsichtlich der Vor- und Nachteile sowie der Machbarkeit einer Migration von Flow nach TypeScript. Der eigentliche Kern der Arbeit wäre anschließend die prototypische Entwicklung eines Werkzeugs, welches die bestehende Codebasis[4] von TeamShirts automatisch in äquivalenten TypeScript-Programmtext transformiert. Dabei muss die Typisierung aufgrund der unterschiedlichen Syntax von Flow und TypeScript z. T. angepasst werden. Konzeptionell würde dies durch Einlesen (*Parsing*) des Quelltexts, anschließender Manipulation des zugehörigen abstrakten Syntaxbaums und schließlich der Ausgabe als TypeScript-Code realisiert werden. Ein funktionsfähiger Prototyp könnte darüber hinaus auch für weitere Unternehmen, die vor derselben Problematik stehen, von Vorteil sein.

[4] Circa 150.000 Zeilen JavaScript-Code.