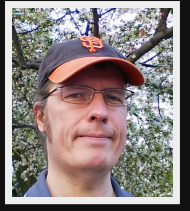
JavaScript Type Systems

Oliver Zeigermann / @DJCordhose

Online Version: http://bit.ly/1GNo1Bd

Oliver Zeigermann

- Developer, Architect, Consultant, and Coach
- Hamburg, Germany
- http://zeigermann.eu
- Working for *embarc Software Consulting GmbH*
- @DJCordhose
- Book: JavaScript für Java-Entwickler



Contents

Motivation: Why a type system in the first place?

- JavaScript is great to create usable solutions quickly
- However if your system
  - is large
  - is very important or even crucial for the success of your company
  - contains substantial amounts of business logic
  - lives for a long time
  - has a lot of contributors
- ... long term maintainability might be more interesting than short term velocity
- JavaScript is lacking a type system, which might impact this...
- even in the next version of JavaScript, there will be no support for type annotations

Does a type system make you more productive?

Or less?

- some poeple claim not using types is even unethical in this Strangeloop talk
  - (given you have an SLA)
- http://danluu.com/empirical-pl/ elaborates that there is little or no impact

In any case, type systems make code easier to maintain

- large code and long living code bases are always hard to maintain…
- … but type annotations
  - make code more readable
  - make code easier to analyse
  - allow for reliable refactoring
  - allow for better IDE support

Typed JavaScript in modern UI frameworks

- Google and Facebook have huge code bases
- They offer the most modern JavaScript UI frameworks
  - Angular 2.0
  - React
- both frameworks either use or heavily support type annotations in framework code
- Angular 2 build on TypeScript

The classic: JsDoc and Google Closure Compiler

Key Features

- Optional Static type information in comments
- Supports a superset of JSDoc
- Compiler uses type information for static checks and optimizations
- Main Page of Compilers
- Used in many Google products and standard way to write JavaScript at Google

Code Sample: Interface

```
/**
 *
 * @interface
 */
function HasName() { }

/**
 * @returns {string}
 */
HasName.prototype.getName = function() {};
```

Code Sample: Class

```
/**
 *
 * @param name {string}
 * @param age {number=}
 * @constructor
 * @implements {HasName}
 */
function Person(name, age) {
    this.name = name;
    this.age = age;
}

/**
 * @type {Person}
 */
var olli = new Person('Olli', 43);
```

Google Closure Compiler: Pros and Cons

- Pros
  - Improved human readability
  - Enhanced IDE-Support
  - Code runs without compilation step
  - Type information can also be used for enhanced minifying
- Cons
  - Growing technology stack
  - Looks sort of ugly
  - Dependency on Google technology (way back to pure JavaScript easy, though)
  - Extensions of JSDoc sometimes surprising (e.g. Interfaces or Varargs)

Mainstream: Microsoft's TypeScript

Key Features

- Developed by Microsoft, v1.0 released April, 2nd, 2014
- More or less ECMAScript 6 with optional declared types
- Compiles to human readable JavaScript
- TypeScript-Playground
- External declarations allow for type declarations for existing JavaScript code
- in release v20150126 even Google's Closure Compiler will support TypeScript's type annotations
- Version 1.5 will feature Annotations and type introspection
- IDE WebStorm 10 features native support

Code Sample

```
interface HasName {
    getName(): string;
}

class Person implements HasName {
    constructor(private name: string, private age?: number) {
    }
    getName() {
        return this.name;
    }
}

var olli: HasName = new Person('Olli', 43);
```

External Declarations

- Adds tooling and type checking to existing JavaScript libraries
- Declarations disappear when compiled
- TypeScript compiler comes with external declarations for core and DOM libraries (`lib.d.ts`)
- External declarations for many JavaScript libraries exists already

Excerpt from `lib.d.ts` #1

```
declare function parseFloat(string: string): number;

interface Function {
    apply(thisArg: any, ...argArray: any[]): any;
    call(thisArg: any, ...argArray: any[]): any;
    bind(thisArg: any, ...argArray: any[]): Function;
    prototype: any;
    length: number;
}
```

TypeScript: Pros and Cons

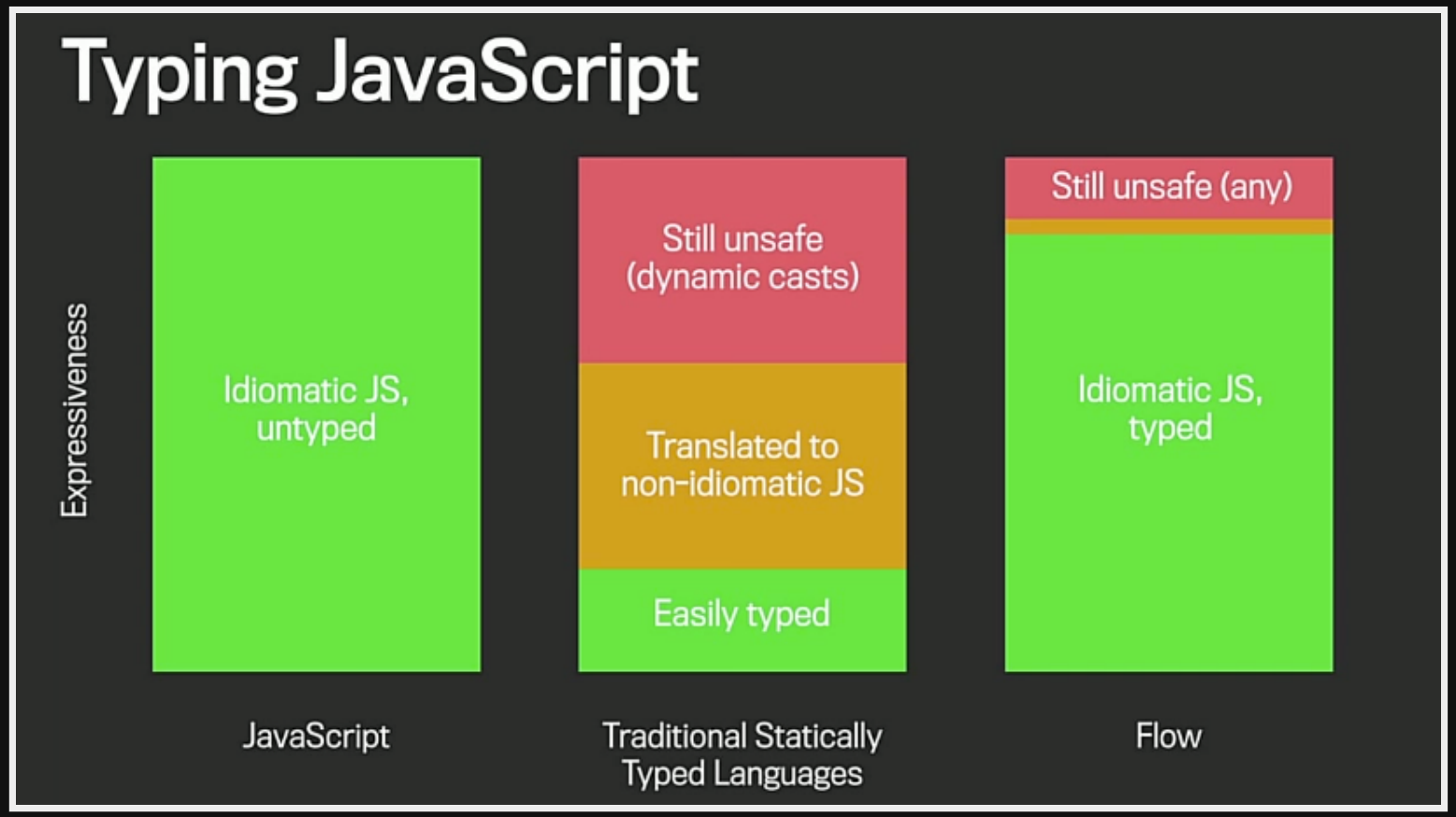
- Pros
  - Improved human readability
  - Looks nice
  - Enhanced IDE-Support
  - Typed IDE-Support even for JSON-Objects (via interfaces)
- Cons
  - Growing technology stack
  - Mandatory compilation step
  - Dependency on Microsoft technology (way back to pure JavaScript easy, though)
  - Optional types may impose too much responsibility on developer

Fancy Pancy: Facebook's Flow

Key Features

- http://flowtype.org/
- Done and used by Facebook
- Uses TypeScripts type annotations
- Can infer types even without any type annotations
- Also supports a larger set of type annotations (maybe types, intersection types, open methods, mixed type)
- Special support for Facebook React's JSX

Motivation: Make the Semantic Gap as small as possible



Code Sample

```
// type inference with zero type annotations:
//  x must be string or number, function returns number
function foo(x) {
    if (typeof(x) === 'string') {
        return x.length;
    } else {
        return x;
    }
}
// fails as there is no support for boolean
var res = foo("Hello") + foo(42) + foo(true);

// tupple types
var tup: [string, number, boolean, string] = ["1", 1, true, "positive"];
var b = tup[0] + tup[1].toFixed();
```

Flow: Pros and Cons

- Pros
  - Open Source
  - Builds on TypeScript
  - Very strong type inference
  - Richer type annotations, matching most JavaScript use cases
  - Forced TypeScript to add least add tupple types and unions
- Cons
  - Pretty new, not much adoption, yet
  - No integration in IDEs, yet
  - Written in OCaml (instead of JavaScript)
  - How much is Facebook committed to it?

Experiments: Google's SoundScript and AtScript

AtScript

- was based on Google's Tracuer ES6 to ES6 transpiler
- was the basis for Google's Angular 2.0 framework
- has been abandoned in favor of TypeScript 1.5 in March 2015

Strong mode and SoundScript

Key Features

- Work as a special modes in Google Chrome's V8 engine
- Based on ES6 and type annotations from TypeScript, Closure Compiler, and Flow
- There are two modes
  - "Strong Mode"
  - "SoundScript"

Strong Mode

- Special mode in Chrome
- `Chrome --js-flags="--strong-mode"`
- Traceur Compiler can also translate Strong Mode to ordinary JavaScript
- Next level of "strict mode"
- Aim is retire some of the really "bad parts" of JavaScript
- Still in an early stage, not fully implemented

Code Sample (Chrome Canary 43.0.2331.3)

```
"use strong";

// fails, as strong mode implies strict mode
doesNotExist = 1000;

// fails, var is no longer supported
var olli = {};

let olli = {};
// should fail, objects no longer work as maps
olli.age = 44;

let array = [1,2];
// should fail, as this is no longer an array
array[10] = 3;

// fails, only === allowed
100 == "100";
```

SoundScript

- Based on "Strong Mode"
- Pretty much like TypeScript
- Runs in the browser including type annotations
- Soundness has a much higher priority than in TypeScript
- static types gurantee they will not be incorrect at run time
- JavaScript engines can optimize more agressively
- Still in an early stage and lacks a PoC

The dark side: Compiling from completely other languages to JavaScript

Dart

https://www.dartlang.org/

- By Google (again)
- Tech lead former Chrome V8 tech lead
- Optional Types
- Compiles to JavaScript
- Meant to be a replacement for JavaScript (because JavaScrip is just too bad)
- Semantics very different from JavaScript as a design choice
- Integration of JavaScript a toughir (partly because of different semantics, partly because of different VM)
- Also runs natively in Dartium browser (derived from Chromium)
- Runs on server on special VM
- considered obsolete by some people

Java: GWT

http://www.gwtproject.org/

- Originally by Google (again)
- Now open project
- Compiles Java to JavaScript
- Uses Google Closure compiler to shrink complete application
- Semantics even more different from JavaScript as this is Java
- Integration of JavaScript a toughie (because of different semantics)
- Integrates well in Java environment (language, Eclipse IDE, Maven)
- Quote by Douglas Crockford: The reason why #GWT exists: a lot of people really like the idea of not having to learn anything!

Scala: Scala.js

http://www.scala-js.org/

- Compiles Scala to JavaScript
- Requires a lib at runtime (many MB in size)
- Uses Google Closure compiler to shrink complete application
- Semantics even much, much more different from JavaScript as this is Scala
- Integration of JavaScript a little easier as Scala allows for dynamically typed (untyped) objects
- Integrates well in Scala environment (language, sbt)

Haskell (sort of): PureScript.js

http://www.purescript.org/

- Typed and functional
- Based on Haskell infrastructure
- Compiles to JavaScript only
- Example code for Person class and show method

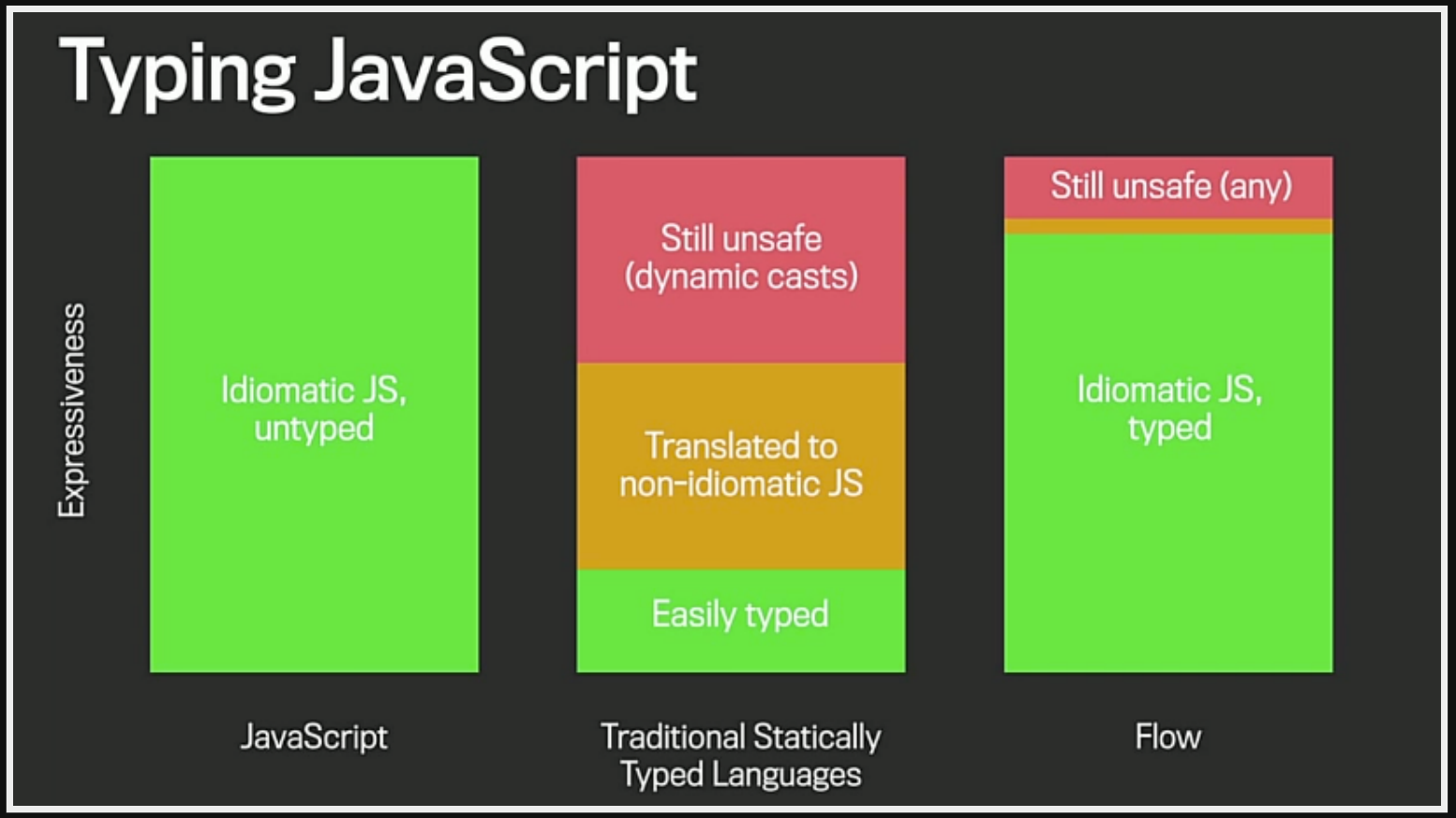
```
data Person = Person { name :: String, age :: Number }

showPerson :: Person -> String
showPerson (Person o) = o.name ++ ", aged " ++ o.age
```

- Example code for creation of a person

```
examplePerson :: Person
examplePerson = Person { name: "Bonnie", age: 26 }
```

- Compiled JavaScript is surprisingly readable

BUT: Remember and Consider the Semantic Gap



Summary

- If you have anything like an enterprise application, seriously consider using types
- Typed JavaScript is dominated by Microsoft's TypeScript annotation style
- Google's AtScript has been abandoned in favor of TypeScript
- Facebook's flow can infer types even without any type annotations
- Chrome's V8 engine features two new "strict" modes that may become interesting in the future (but are not relevant now)
- Compiling to JavaScript from languages that are far away semantically might be a bad idea
- When shit hits the fan, in any case you must be able to understand and debug JavaScript

Thank you!

Questions / Discussion

Oliver Zeigermann / @DJCordhose