LIF'Tech – Marseille

Angular

TypeScript

TypeScript

Bertrand Estellon / Didier Villevalois

Aix*Marseille université
Initiative d'excellence

Pourquoi TypeScript ?

JavaScript peu adapté pour les applications

- langage de script
- pas de constructions structurantes (classes, modules, ...)
- pas de typage statique

« Application-scale JavaScript »

TypeScript = JavaScript avec types, classes et modules optionnels

Lancé en 2012 par Microsoft

- 2014, version 1.0
- 2017, version 2.3

TypeScript

TypeScript, sur-ensemble de JavaScript qui compile vers JavaScript

TypeScript démare de JavaScript

- tout code JavaScript est du code TypeScript
- toute librairie JavaScript fonctionne avec TypeScript

TypeScript finit en JavaScript

- compile vers du code JavaScript idiomatique
- produit du code portable (browsers, hôtes, OS)
  - cible EcmaScript3 (ou bien ES5 ou ES6)

Compilateur, librairies : tout en Open Source

System de typage

```
void, boolean, number, string, symbol, object

null, undefined,

string[]

[string, number, bool]

(number, string) => boolean

{ isClosed: boolean; close(): void  }

number | string

Saucer & Flying

any
```

System de typage – curiosités

```
never
```

```
"a" | "b" | "c"
```

```
{ [index: number]: string  }
```

- les types indexés ont un accesseur `[]`
- les types indexés par des chaînes ont des accesseurs `[]` et `.`

```
name?: type
```

- avec détection des valeurs nulles

```
function isObservable(x: any): x is Observable {
  return x.next
}
```

Démonstration

System de typage

Une représentation statique du system de typage de JavaScript

Inference de type avec typage structurel

- en pratique, peu d'annotations nécessaires

Fichiers de déclaration pour les librairies externes

- typage statique pour DOM, JQuery, NodeJS, ...

Pas de preuve de sûreté du typage... 

- indique une intention et non pas des garanties

Classes, Interfaces et Héritage

Classes

- suit le standard ECMAScript 6
- attributs, méthodes (avec paramêtres à valeur par défaut)
- constructeurs (avec attributs automatiques)
- membres privés et membres statiques
- surcharge de méthodes (plusieurs définitions, une implémentation)
- paramêtres de type (avec paramêtres à valeur par défaut)
- produisent du code JavaScript idiomatique

Interfaces

- ouvertes
- ne produisent pas de code JavaScript

Héritage, similaire à Java

Modules

Modules et export

- suit le standard ECMAScript 6

```
module My.Module {
  export function myFunction() { /* ... */ }
}
```

Modules ouverts

- plusieurs fichiers peuvent contribuer au même module

Produisent du code JavaScript idiomatique

- cible les standards (de-facto) CommonJS ou AMD

Clotûres

Fonctions anonymes comme en JavaScript

```
myDiv.addEventListener(function (event) { /* ... */ })
```

- variable `this` liée manuellement

Fonctions « flèches » (aka. lambdas)

```
myDiv.addEventListener(event => { /* ... */ })
```

- moins verbeuses
- variable `this` liée à l'unité léxicalle englobante

Décorateurs

Similaire à python

`@expression`

- sur une déclaration de classe, méthode, accesseur, propriété, ou paramêtre
- l'expression doit s'évaluer en une fonction
- appliquée à l'exécution sur la cible
- reçoit aussi des méta-informations sur la cible

```
function myAnnotation(...someArguments) {
  return function (target,
                   propertyKey: string,
                   descriptor: PropertyDescriptor) {
    // ...
  }
}
```

Curiosités

`Async` et `Await`

Itérateurs et générateurs

- aussi bien synchrones que asynchrones

Déstructuration de tableaux et d'objets

```
let [a, , b] = [1, 2, 3]
let [first, ...rest] = myArray
let newArray = [...rest, first, ...rest]

let { x, y } = myPoint;
```

Interpolation de chaîne de charactères

Boîte à outils

Construction

- compilateur écrit en TypeScript
- transpile et fabrique des fichiers .js et .d.ts
- open-source

Environnements

- typage et completion
- même dans les templates Angular

Certaines librairies directement écrites en TypeScript

- Angular, Ionic, RxJs 5, ...