TypeScript

ではじめるSAFEコーディング

jsCafe vol.10 / 2013-07-07

Agenda

- TypeScript
- Many IDE and Editor
- WebStorm And TypeScript
- Summary

自己紹介

{

Sakuya Sugo

@Sakunyo

好きなこと

[Vim, PhpStorm, NodeJS, Github, bitbucket,
nginx (Reverse Proxy), Jenkins, Oreilly]

}

TypeScript

アプリにサーバーサイドに、開発が複雑化

- Github
- Pull Request
- Jenkins, Travis CI
- CI (継続的インテグレーション)
- Grunt
- JavaScript Test Framework
  (QUnit, Jasmine, Sinon, js-test-driver, testem, karma)

TypeScript

Microsoftがオープンソースとして公開
altJS の 型付け特化のコンパイラ
2012年に登場し現在* v0.9 *2013-07-07
型の宣言(declaration)
module, export, interface, class, extents
など

TypeScript

- TypeScript

  http://www.typescriptlang.org/

  https://npmjs.org/package/typescript

- WEB DB PRESS Vol.75

  http://gihyo.jp/magazine/wdpress/archive/2013/vol75

CoffeeScript がしっくりこない

altJSのなかで人気がある CoffeeScript

CoffeeScript の文法は融通がきく反面

人によって書き方違いが多く、

コーディングスタイルは保ちにくい

コンパイルされたコードは読み辛い傾向

CoffeeScript という"言語" (*Ruby Like*)

Many IDE and Editor

IDE / Editor

- Visual Studio 2012
  + TypeScript for Visual Studio 2012
  http://www.microsoft.com/en-us/download/details.aspx?id=34790
- JetBrains
  Windows, OSX, Linux のマルチプラットフォーム
  TypeScript は 現在* v0.8 対応 *2013-07-07
- Sublime Text, Vim, Emacs
  TypeScript support for Sublime Text
  TypeScript support for Emacs
  TypeScript support for Vim
  http://blogs.msdn.com/b/interoperability/archive/2012/10/01/sublime-text-vi-emacs-typescript-enabled.aspx
- Grunt
  k-maru/grunt-typescript
  https://github.com/k-maru/grunt-typescript

npm install -g typescript

```
# Compile
tsc hello.ts

# Options
-c,    --comments    tsファイルのコメントを出力する
-d,    --declaration  tsファイルを元に hello.d.ts を出力する
-w,    --watch
--out    FILE|DIRECTORY
--sourcemap          chrome などで使える SourceMapする
...
```

WebStorm and TypeScript

型に特化とは言いつつも
紹介しきれないほど機能がありますので
今回はフロントエンドで
手軽に使える機能を重点に紹介します。

Type

```
/**
 * Type は大きく分けて、Primitive TypeとObject Type
 *
 * Primitive Types
 * [string, number, boolean, void, null, undefined, Enum ...]
 *
 * var 宣言した値などは暗黙的に型が決まる
 * 他の型の値を代入できない
 */
var varString = "String"; // 暗黙的に型を宣言
// varString = 123; // Error

var varString2:string = "String2"; // 型を明示
```

Type

```
/**
 * Array
 * 型を束縛しない any
 */
var varArray  = [ 1, 2, 3 ]; // 暗黙的に number[]
var varArray2 = [ 1, "2",
  { x:"string" }, function () {} ]; // 暗黙的に {}[]

var varArray3:any[] = [ 1, "2", function () {} ]; // 明示 any[]

/**
 * Object
 */
var varObject = { x:"x", y:123 };
```

Function

```
/**
 * Function は Arguments と Return にTypeを指定できる
 * Argus のオプションは v?:string
 * Default Valueを設定する場合は v:string = “str” と ？を省略できる
 */
function hello (v:string = ""):string {
  return "hello" + v;
}
console.log(hello());
console.log(hello("string"));

function myCallback (callback:() => any) {
  callback();
}
myCallback(function(){ /* ... */ });
```

Class

```
/**
 * class, extends
 * 出力される JavaScript コードは Classy なコード
 * [private, public] [static]
 */
class MyClass {
  constructor ( public x:number, private _x:string ) { }
  public myMethod ():number { return 123; }
}
var myclass = new MyClass(10, "private value");

console.log( myclass.myMethod() );
console.log( myclass.x );

// private を指定した場合には外側からアクセスするとError
// console.log( myclass._x ); // Error
```

Interface

```
/**
 * Interface
 * オブジェクトの持っている特徴を抽象化したもの
 * クラス以外にも適用できる
 * 慣例的に IMyInterface のように I を接頭辞にする
 */
interface IFruit {
  name:    string;  // フルーツ名
  isRipe: boolean; // フルーツは食べごろか？
  color?: string;  // フルーツの色 *オプション
}

class Banana implements IFruit {
  /* ... */
}
```

Module

```
/**
 * Module は Internal と External の 2 種
 * Internal はネームスペースを提供
 * External は CommonJS や RequireJS に対応する
 */
// Internal Module
module MyModule {
  class dntExport { // Export されず Module内でのみ有効
    constructor() { return this; }
  }
  export class MyExportClass { // Module名のプロパティにExportされる
    constructor() { return this; }
  }
}
```

Using jQuery 1

```
/**
 * DOM へのアクセスはビルトインされている
 * declaration ファイルによってアクセス可能になっている
 */
//  Return HTMLCanvasElement;
function getCanvasElement () {
  return document.createElement("canvas");
}

//  Return CanvasRenderingContext2D;
function getCanvasElementContext () {
  var canvas = getCanvasElement();
  return canvas.getContext("2d");
}
```

Using jQuery 2

```
// jQuery などのライブラリを使う場合 は declaration を読み込む
var $divs = jQuery(".div"), i:number, iz:number;

console.log(jQuery);
console.log($divs.length);

for (i = 0, iz = $divs.length; i < iz; i++) {
  console.log($divs[i]);
  $($divs[i]).text("string");
}

// 主要なライブラリの宣言ファイルが集まる borisyankov/DefinitelyTyped
// https://github.com/borisyankov/DefinitelyTyped
```

Summary

Summary

導入メリット

- CoffeeScript と違い、JavaScript の記述に近く学習コストが低く導入しやすい
- Visual Studio 2012, WebStorm, PhpStorm などIDEではより開発が進めやすくなる
- NodeJS があれば大丈夫
- CoffeeScript の出力した 生Script のように 出力されたファイルが読み辛いこともない

Summary

デメリット

- IDEで使えたほうが効率がはるかに良い
- コードが冗長になりやすい
- JavaScript Framework や、自作したコード資産を導入するには
  型宣言に any を用いたり、declare などを用意したりと手間がかかる
- 元のコードが CoffeeScript なんだけど ... 諦める
- CoffeeScript などとは違い便利な構文などはないので
  JavaScript の構文のノウハウが必要になる

参考に

- TypeScript ドキュメント
  http://typescript.codeplex.com/
- borisyankov/DefinitelyTyped
  https://github.com/borisyankov/DefinitelyTyped
- leafgarland/typescript-vim
  https://github.com/leafgarland/typescript-vim
- k-maru/grunt-typescript
  https://github.com/k-maru/grunt-typescript