AngularJS with Typescript

by Vlad Costel Ungureanu
for “Learn Stuff” and “CGM Romania”

What is Typescript ?

- ✓ A type superset of JavaScript
- ✓ It compiles to pure JavaScript code
- ✓ It enforces the usage of classes and type for compile time type safety
- ✓ It allows for the usage of inheritance, interfaces and generics as any other OOP language
- ✓ It is trending, specially with it being used as default in AngularJS 2

Simple example

The following typescript code defines a class:

```
module Test {
    export class MyClass{
     constructor() {}
    }
}
```

The resulting JS looks like this:

```
var Test;
(function (Test) {
    var MyClass = (function () {
        function MyClass() {
        }
        return MyClass;
    }());
    Test.MyClass = MyClass;
})(Test || (Test = {}));
```

Short Overview - Types

- ✓ Basic types:
  - ✓void
  - ✓any
  - ✓string
  - ✓number
  - ✓boolean
  - ✓array
  - ✓tuple
  - ✓null
  - ✓undefined
  - ✓never

Short Overview - Variables

✓ Working with variables:

```
// any
let notSure: any = 4;
// primitive
let someValue: string = "this is a string";
// enum
enum Color {Red, Green, Blue}
let c: Color = Color.Green;
// tuple/map
let x: [string, number];
// array
let list: number[] = [1, 2, 3];
// array with generics
let list: Array<number> = [1, 2, 3];
```

LEARN stuff IO

Short Overview - Variables

```
var Test;
(function (Test) {
    var MyClass = (function () {
        function MyClass() {
            // any
            this.notSure = 4;
            // primitive
            this.someValue = "this is a string";
            // array
            this.list = [1, 2, 3];
            // array with generics
            this.array = [1, 2, 3];
        }
        return MyClass;
    }());
    Test.MyClass = MyClass;
})(Test || (Test = {}));
```

Short Overview – Classes and Fuctions

The following TS code defines a class;

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() { return "Hello, " + this.greeting; }
}
let greeter = new Greeter("world");
```

The resulted JS code looks like this:

```
var Greeter = (function () {
    // constructor
    function Greeter(message) { this.greeting = message; }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
```

Short Overview – Classes and Functions

✓ Classes are the building blocks of OOP development

✓The follow the basic definition and functionality of a class

✓Methods declared in a class will be part of that classes prototype

✓Members can have access modifiers

✓Classes can extend other classes and implement interfaces

Short Overview - Interfaces

The following typescript code describes an interface:

```
interface LabelledValue {
   label: string;
   // optional – may or may not exist in the object that is of this interface type
  color? : string;
  // readonly – cannot be changed in objects that is of interface type
  readonly size: number;
  // functions
  // draw (originX:number, originY: number): void;}
function printLabel(labelledObj: LabelledValue) {
   console.log(labelledObj.label);}
export class MyObj implements LabelledValue{
    size= 10; label= "Size 10 Object“; };
printLabel(new MyObj());
```

The resulting JS looks like this:

```
function printLabel(labelledObj) { console.log(labelledObj.label); }
var MyObj = (function () {
   function MyObj() { this.size = 10;this.label = "Size 10 Object"; }
   return MyObj;
}());
exports.MyObj = MyObj; printLabel(new MyObj());
```

Short Overview - Interfaces

✓ Interfaces in TS are contacts for existing properties of callable objects

✓In JS we do not differentiate between methods and variables when accessing them

✓In TS we enforce the existing of specific fields not just methods, like interfaces in other languages

Short Overview – Type Inference

✓ For primitive types type inference is direct

```
let x = 3;
```

✓For array the best common type will be used

```
let x = [0, 1, null];
```

✓ Type inference to the closest type

```
window.onmousedown = function(mouseEvent: any) {
        console.log(mouseEvent.buton); //<- Now, no error is given
};
```

Short Overview – Fat Arrows

✓ TS methods will always require the use of “this” in order to determine the context in which to search for variables

✓ This means that call-backs or methods passed and called in other execution context will loose the initial reference and will use the callers context as “this”

✓This makes all references of the initial context un-usable

✓Using fat arrows is the equivalent of correctly using “bind”

Short Overview – Fat Arrows

```
varsmartPhones = [
        { name:'iphone', price:649 },
        { name:'Galaxy S6', price:576 },
        { name:'Galaxy Note 5', price:489 }
];
console.log(smartPhones.map(
        smartPhone=>smartPhone.price
)); // [649, 576, 489]
export class FatClass {
 public aClassValue: string;
 constructor(){ }
 public aClassmethod(): void { }
 myMethod = (anyParameter: any): boolean => {
        this.aClassValue = "test";    this.aClassmethod();
        console.log(anyParameter);   return true;}
}
let inst = new FatClass();
inst.myMethod("test 2");
```

AngularJS Configuration

✓ Since Typescript is typed we need to specify what the angular type definitions are

```
/// <reference path="node_modules/@types/angular/index.d.ts"/>
/// <reference path="node_modules/@types/angular-ui-router/index.d.ts"/>
```

✓ Everything in JS is a callable object, but in TS we need to create classes that return the necessary callable objects and their properties

```
class Configuration {
   constructor(private $stateProvider: ng.ui.IStateProvider, private $urlRouterProvider:
        ng.ui.IUrlRouterProvider, private $httpProvider : ng.IHttpProvider) {
      this.init();
   }
   private init(): void {
}
```

✓Classes also need to be used

```
var app = angular.module('antype', ['ui.router', 'toastr'])
   .config(
($stateProvider: ng.ui.IStateProvider,
$urlRouterProvider: ng.ui.IUrlRouterProvider,
$httpProvider: ng.IHttpProvider) => {
      return new Configuration($stateProvider, $urlRouterProvider, $httpProvider)
});
```

AngularJS Routes

```
private init(): void {
    this.$stateProvider.state("main", Configuration.defaultState());
    this.$stateProvider.state("login", Configuration.loginState());
    this.$urlRouterProvider.otherwise('/main'); }
    private static defaultState(): ng.ui.IState {
            return { url: "/main", templateUrl: "/app/main/main.html“ } }
   private static dashboardState(): ng.ui.IState {
    return {
       url: "/dashboard",
       templateUrl: "/app/dashboard/dashboard.html",
       controller: "DashboardController",
       controllerAs: "dashboardController" } }
    private static loginState(): ng.ui.IState {
    return {
       url: "/login",
       templateUrl: "/app/login/login.html",
       controller: "LoginController",
       controllerAs: "loginController"} }
```

AngularJS Interceptors

```
module Shared {
 export class HttpInterceptor {
      private $q;
      public preloaderService: any;
      public $injector: ng.auto.IInjectorService;
      static $inject = ['$injector', '$q'];
      constructor($injector: ng.auto.IInjectorService, $q) {
         this.$q = $q;
         this.preloaderService = $injector.get("PreloaderService");
         this.$injector = $injector;}
      public static instance($injector, $q: ng.IQService) {
         HttpInterceptor.instance.$inject = ["$injector", "$q"];
         return new HttpInterceptor($injector, $q);
      }
      public responseError = (rejection) => {
         this.preloaderService.removePreloader();
         return this.$q.reject(rejection);
      };
```

AngularJS Interceptors

```
    public response = (response) => {
        if (this.$q.when(response)) {
            this.preloaderService.removePreloader();
            return this.$q.when(response);}
        return response || this.$q.when(response);
    }
    public request = (config) => {
        this.preloaderService.startPreloader();
        return config;};
    public requestError = (rejection) => {
        this.preloaderService.removePreloader();
        return this.$q.reject(rejection);};
  }
}
```

An the config:

```
 private init(): void {
    this.$httpProvider.interceptors.push(Shared.HttpInterceptor.instance);
  }
```

AngularJS Controllers

```
module Login {
   export interface ILoginScope  extends ng.IScope {
   }
   export class LoginController{
      static $inject = ["$scope", "LoginService"];
      constructor(protected $scope: Login.ILoginScope, protected LoginService: LoginService){
        // do nothing
      }
      login = (user: any): void => {
        this.LoginService.verifyLogin(user);
      }
   }
   app.controller('LoginController',  Login.LoginController);
}
```

AngularJS Services

```
module Login {
  export class LoginService {
    static $inject = ['LoginValidator', 'AuthenticationService', 'QueryService', 'ErrorHandlingService',
        '$state', 'NotificationService'];
    constructor(protected LoginValidator: LoginValidator, protected AuthenticationService:
        Shared.AuthenticationService,
        protected QueryService: Shared.QueryService, protected ErrorHandlingService:
        Shared.ErrorHandlingService,protected $state:any,
        protected NotificationService: Shared.NotificationService){
    }
    verifyLogin = (user: any) => {
       lat validationResult = this.LoginValidator.validateLogin(user);
       if (validationResult == true) {
         this.QueryService.get('login/'+user.username).
         then((response) => this.processLogin(response.data)).
         catch((error) =>this.handleError(error));
       }
    }
```

AngularJS Services

```
public processLogin(data:any): void{
      if (data.login === 'accepted') {
        this.AuthenticationService.setLoggedIn(true);
        this.$state.go('dashboard');
      } else {
        this.AuthenticationService.setLoggedIn(false);
        this.NotificationService.error("Username and password do not match any existing account!", "
Error");
      }
    }
    public handleError(error): void {
      this.ErrorHandlingService.handle(error);
    }
  }
  app.service('LoginService',  Login.LoginService);
}
```

AngularJS Directive

```
/// <reference path="../../../node_modules/@types/angular/index.d.ts"/>
module Commons {
   export interface MyDirective extends ng.IDirective {
   }
   export class LoginFormDirective implements MyDirective {
      static $inject =[];
      constructor() {
        // do nothing
      }
      restrict = 'E';
      scope = { 'loginmethod': '=loginmethod' };
      templateUrl = '/app/common/loginFormDirective/login-form.html';
      static instance(): MyDirective{
         return new LoginFormDirective();
      }
   }
   app.directive('loginform', LoginFormDirective.instance);
}
```

AngularJS $http

```
module Shared {
  export class QueryService {
     static $inject = ["$http", "ConfigService", "NotificationService"];
     constructor(protected $http: ng.IHttpService, protected  ConfigService: ConfigService,
       protected NotificationService: Shared.NotificationService){
     }
     public get(urlPath:string): ng.IPromise<any> {
        let result: ng.IPromise<any> = this.$http.get( this.ConfigService.customPath(urlPath),
        this.ConfigService.restGetConfiguration());
       return result;
     }
     public post(urlPath: string, body: any): ng.IPromise<any> {
        let result: ng.IPromise<any> = this.$http.post(this.ConfigService.customPath(urlPath),
        body, this.ConfigService.restConfiguration());
       return result;
     }
```

AngularJS $http

```
public put(urlPath: string, body: any): ng.IPromise<any> {
        let result: ng.IPromise<any> = this.$http.put(this.ConfigService.customPath(urlPath),
        body, this.ConfigService.restConfiguration());
      return result;
    }
    public delete(urlPath: string): ng.IPromise<any> {
        let result: ng.IPromise<any> = this.$http.delete(this.ConfigService.customPath(urlPath),
        this.ConfigService.restConfiguration());
      return result;
    }
  }
  app.service('QueryService',  Shared.QueryService);
}
```

Configure Typescript

- ✓ We need to install node
- ✓ Install typescript

```
npm install --save-dev typescript
npm install --save-dev @types/angular
npm install --save-dev @types/angular-ui-router
```

- ✓ Create the tsconfig.json file

```
{
  "compilerOptions": {
    "module": "system",
    "target": "es6",
    "noImplicitAny": false,
    "removeComments": true,
    "preserveConstEnums": true
  },
  "files": [ "app.ts" ],
  "include": [ "app/**/*" ],
  "exclude": [ "node_modules", "**/*.spec.ts"]
}
```

- ✓ Run the compiler

```
node_modules/.bin/tsc
```

Configure Grunt

✓ We need to have node installed

✓ Install grunt:

```
npm install --save-dev grunt
npm install --save-dev grunt-contrib-concat
npm install --save-dev grunt-contrib-cssmin
npm install --save-dev grunt-contrib-watch
npm install --save-dev grunt-sass
npm install --save-dev grunt-ts
```

Configure Grunt

✓ We create the grunt config file Gruntfile.js:

```
module.exports = function(grunt) {
        grunt.initConfig({
        ts: {
            default : { tsconfig: true }
          },
        sass: { options: { },
                dist: { files: {  'css/modules/base.css': 'css/modules/_base.scss‘  } }
        },
        concat: {
             options: {  separator: '\n/*next file*/\n\n‘ },
             dist: {  src: ['app/**/*.js'],  dest: 'build/built.js‘ }
           },
        cssmin: {
                build: { src: 'build/style.css', dest: 'build/main.min.css‘ }
        },
```

LEARN stuff IO

Configure Grunt

```
    watch: {
        sass: {
            files: ['css/**/*.{scss,sass}','css/modules/**/*.{scss,sass}'],
            tasks: ['css']
        },
        ts: {
          files:  [ '**/*.ts' ],
          tasks:  [ 'ts', 'concat' ],
          options: { spawn: false }
        },
        all: {
          files: ['**/*.html'],
          options: { livereload: true }
        }
      }
    }
);
```

Configure Grunt

```
    // Load up tasks
    grunt.loadNpmTasks('grunt-sass');
    grunt.loadNpmTasks('grunt-contrib-cssmin');
    grunt.loadNpmTasks('grunt-contrib-concat');
    grunt.loadNpmTasks('grunt-contrib-watch');
    grunt.loadNpmTasks('grunt-ts');

    // Default task and other task aliases
    grunt.registerTask('css', ['sass:dist', 'cssmin']);
    grunt.registerTask('js', ['concat']);
    grunt.registerTask('default', ['ts','css', 'js', 'watch']);

// end of config file
};
```

Configure Grunt

✓Run grunt:

```
// default task
grunt
// only typescript tasks
grunt ts
// only css tasks
grunt css
```

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com