Curso de Desarrollo Web con Angular

Artículo

Guía de TypeScript

Uriel Ramírez 23 PlatziRank Feb. 15, 2017

¿Qué es TypeScript?

TypeScript es un superset hecho por Microsoft el cuál tiene una sintaxis muy intuitiva y que nos recuerda a otros lenguajes orientados a objetos. Agrega funcionalidades que extiende lo que haría JavaScript por si sólo, tal cómo Types y Decorators - los cuales veremos más adelante-.

¿Entonces mi navegador ejecutará TypeScript en lugar de JavaScript?

R. No, TypeScript cuenta con un 'transpiler' que convierte todo ese código a JavaScript e inclusive puede elegir entre ES2015 ó ES2016.

Instalando TypeScript

Para poder usar TypeScript necesitas tener instalado previamente Node.js, y después en nuestra terminal, ejecutar el comando:

```
npm install -g typescript
```

Y usando tu editor de texto favorito, puedes ir creando tus archivos de TypeScript recordando que la extensión es .ts

Datos básicos en TypeScript

imprimir en consola

Para imprimir datos en consola no es diferente a JavaScript:

```
console.log("Mi mensaje");
```

variables y valores primitivos

Las variables tienen la palabra reservada *var*, pero el tipo de dato se especifica usando *:typo*

Tenemos 3 tipos de datos primitivos: *string*, *number* y *boolean*.

Ejemplo

```
var full_nombre:string = "jorge cano";
var age:number = 27;
var developer:boolean = true;
```

Arreglos

Los arreglos se declaran con la palabra Array<*Tipo*> en el que el tipo son los valores primitivos que vimos anteriormente. Recuerda que solo pueden existir en el Array valores del tipo antes especificado.

Ejemplo

```
var skills:Array<String> = ['JavaScript','TypeScript','Angular'];
```

También podemos declarlos de la forma:

```
var numberArray:number[] = [123,123,1213,1231];
```

Enumerables

Los Enumerables son un objeto que enumera los elementos dentro de él. Se usa principalmente para definir roles o configuraciones y evitar el uso de constantes. Tienen la palabra reservada *enum* y el nombre es recomendable que se encuentre todo en mayúscula.

Ejemplo

```
enum ROLE {Employee, Manager, Admin, Developer }
```

Entonces podemos asignarle un valor de Role a cualquier variable

```
var role:Role = Role.Employee
```

Funciones

Las funciones son bloques de código que esperamos ejecutar de manera recurrente. Tienen la palabra reservada *function* y tenemos múltiples formas de declararlas según las necesidades, por ejemplo:

a) Función que no retorne nada y que no reciba parámetros:

```
function hello():void {
}
```

b) Función que no retorne nada y que reciba un parámetro del tipo string:

```
function setName(name:string):void{
}
```

c) Función que retorne un dato tipo string y que reciba 2 parámetros del tipo string:

```
function setName(name:string, surName:string ):string {
    return "string";
}
```

Es importante de observar que seguimos necesitando indicar qué *tipo de dato* es que el vamos a recibir y/o regresar.

Scope de una variable y la palabra "this"

Las variables que se encuentran dentro de una función no pueden ser accedidas desde otras partes fuera de ella. Por ejemplo:

```
// Teniendo la función

function setName(name:string ):string {
    var variableInterna:string = "Uriel";
    return "Hola" + name;
}

// No se puede acceder a la variable Interna
console.log("Hola" + variableInterna );
```

Sin embargo, podemos acceder a las variables globales que se encuentren fuera de una función desde cualquier parte, inclusive desde en interior de una función.

```
// Teniendo la misma función pero con la variable fuera de ella

 var variableExterna:string = "Uriel";

function setName(name:string ):string {
    // Accedemos a la variable externa usando "this"
    this.variableExterna = name;
    return "Hola" + name;
}

// Desde aquí podemos igualmente acceder a la variable

console.log("Hola" + variableExterna );
```

Pasando de .ts a .js

Al instalar TypeScript, se agregan una serie de herramientas que nos ayudan a 'transpilar' nuestro código. Solo necesitas ejecutar el comando

```
tsc "NombreDelArchivo".ts
```

Y al terminar, en el mismo directorio, encontrarás el .js tranformado con el mismo nombre de tu archivo de TypeScript, y ya listo para usar para cualquier navegador.

TypeScript CLI

TypeScript CLI es la herramienta que nos va a ayudar a 'transpilar' nuestros archivos de TypeScript a JavaScript. Podemos hacer entre otras cosas: Transpilaciones globales, de un solo archivo, elegir la versión de ECMAScript que queremos, etc. Recordando la instalación:

```
npm install -g typescript
```

Y para pasar de .ts a .js sólo necesitas ejecutar el comando

```
tsc "NombreDelArchivo".ts
```

Y al terminar, en el mismo directorio, encontrarás el .js tranformado con el mismo nombre de tu archivo de TypeScript, y ya listo para usar para cualquier navegador.

Objetos en TypeScript

Un objeto es una abstracción de algo que existe en el mundo "real", el cuál llevamos todas sus características a código. Estos objetos se crean por algo llamado "clases". En typescript

tenemos la palabra reservada *class*.

Ejemplo:

```
class Persona {

  first_name:string;
  last_name:string;

  constructor(_first_name?:string, _last_name?:string) {
    this.first_name = "_first_name";
    this.last = "_last_name";

  }
}
```

Es importante decir que una clase siempre tiene un constructor, y ese constructor es lo primero que se va a llamar por defecto cuando hagas una instancia de esa clase. Si tu no lo implementas, por defecto es el constructor más sencillo:

```
constructor () {

}
```

Sin embargo, podemos tener parámetros obligatorios u opcionales, los opcionales que se distinguen por tener un "?".

```
constructor(_first_name?:string, _last_name?:string)
    this.first_name = "_first_name";
    this.last = "_last_name";

  }
```

Para hacer una instancia de la clase, podemos crear una nueva variable o constante e indicar la clase mediante la palabra *new*:

```
let personaUno = new Persona();
let personaDos = new Persona("Jorge");
let personaTres = new Persona("Jorge", "Cano");
```

¿Qué pasa si los parámetros de mi constructor no son opcionales?

R. En el caso de que tengamos parámetros obligatorios, no se podrá crear una instancia de la clase sin mandar esos datos en el constructor:

```
//Teniendo en cuenta:

constructor(_first_name:string, _last_name:string)
    this.first_name = "_first_name";
    this.last = "_last_name";

  }

// Esto ya no se podrá ser una instancia de esa clase

let personaDos = new Persona("Jorge");

//Esta será una instancia de esa clase

let personaTres = new Persona("Jorge", "Cano");
```

Interpolation

Interpolation es la forma en que mezclamos variables justo con los strings. Acompañado de comillas francesas, podemos acceder al valor de una variable de forma directa desde ese string.

Ejemplo:

```
var a:string = "Uriel";

var b = `Saludos a ti  ${this.a},`;

console.log(b);`
```

Otra de las ventajas de usar las comillas francesas es que respeta cada espacio y cada salto de linea que insertamos, tal como un template:

```
getSaludo():string{
        let emojis = '(⌐■_■)';
        return `Saludos
            ${this.last_name}, ${this.first_name}
            Le enviamos un saludo desde la consola!
            ${emojis}
        `;
    }
```

Interfaces

A diferencia de las Clases, las interfaces no se instancian mediante constructores, si no mediante un objeto JSON. Tienen la palabra reservada *interface*.

Ejemplo:

```
interface MyPersona{
    first_name:string;
    last_name:string;
    twitter_account?:string;
}
```

De igual manera podemos tener campos obligatorios y opcionales.

Para hacer uso de una interfaz, incluimos en el tipo de dato el nombre de la interfaz y le asignamos un objeto:

```
let personaUno:MyPersona = {
    first_name : 'Jorge',
    last_name : 'Cano',
    twitter_account: '@jorgeucano'
}
```

Shapes

Los Shapes son muy parecidos en implementación a las clases y siguen siendo objetos. La única diferencia es que no dependemos de los constructores para asignar valores a las variables internas de las clases, si no que asignamos directamente el valor después de la creación de la instancia:

```
class Person{
    first_name:string;
    last_name:string;
    twitter_user:string;

    constructor(){
        this.first_name = "Jor";
        this.last_name = "Ca";
        this.twitter_user = "@jorgeucano";
    }

    setLastName(last_name : string){
        this.last_name = last_name;
    }
```

```
}

var myPerson = new Person();
myPerson.first_name = "Jorge";
myPerson.setLastName("Cano");
console.log(myPerson);
```

Mientras las variables dentro de la clase sean públicas, podemos hacer uso de los shapes, de otro modo, se harían por getters y setters.

Decorators

Los decorators son algo que se usan mucho en Angular pero que tambien que tambien tenemos en TypeScript. Los decorators nos permiten agregar una funcionalidad extra a algo, como por ejemplo a métodos o miembros de una clase.

La estructura más simple de un decorator es:

```
function color(value: string) {   // Así definimos el decorator fabric
    return function (target) { // Este es el decorator, prácticamente regresamos un una función con la funcionalidad, se pide el objetivo

  // Aquí es donde modificamos el objetivo con el valor pedido desde el decorator
    }
```

Particularmente, los decorators de método necesitan de 3 parámetros en su definición:

- **target:**: es método el cuál vamos a aplicar el decorator
- **key:**: es nombre del método el cuál vamos a aplicar el decorator
- **value**: es indefinido en otros casos.

_Un ejemplo: _

```
class Greeter {

    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }

    @enumerable(false)
    greet() {
         return "Hey, " + this.greeting;
        // return
```

```
    }
}

function enumerable(value: boolean) {
    return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
        descriptor.enumerable  = value;
    };
}

let gree = new Greeter("Soy el mensaje");

console.log(gree.greet());
```

Lo que va a hacer nuestros decorator @enumerable(false) es verificar si el tipo de dato no es enum. De serlo, no podrá implementar el método greet()

Por último, conTypeScript podemos tener un archivo de configuración para aprovechar todas las ventajas al momento de hacer la transpilación, entre muchas cosas tenemos:

```
{
    "compilerOptions": {
        "target": "ES5",        // Definir en que versión de JS nos gustaría convertir el
.ts
        "experimentalDecorators": true,  //  Activar funciones experimentales, en este
caso decorators
        "module": "system",   // Usar el modulo system
        "noImplicitAny": true, // Habilitar el uso de implicitos Any
        "removeComments": true, // Remover todos los comentarios en nuestros archivos
        "preserveConstEnums": true, // Manejar los enumerables como constantes, esto es
muy recomendado
        "outFile": "./tsc.js", // Donde vamos a tener el archivo resultante
        "sourceMap": true // Para ver un log de la transpilación
    },
    "include": [
        "*"       // Ver que archivos vamos a compilar, usar * indica que serán todos
    ],
    "exclude": [
        "node_modules"   // Agregar los archivos que no queremos incluir en la
transpilación
        "**/*.spec.ts"
    ]
}
```

Y ya guardado este archivo como **tsconfig.json** podemos ejecutar en la terminal el comando **tsc** para realizar los cambios.

Durante todo el concepto, tuvimos diferentes archivos .ts, y notarás que con la ayuda de la configuración, resulto un solo archivo .js que contiene todos los ejercicios ahora juntos.

¿Qué otras cosas haz hecho con TypeScript?, déjalo en los comentarios

B I U | Insertar código | Enlace | Imagen

Deja tu comentario

Suma tu comentario

Top Nuevas

2 **wp_wilsonperez** 10 Puntos — 3 meses

Diferencia entre LET y VAR.
La diferencia es el alcance de las variables. let permite declarar variables limitando su alcance al bloque, declaración, o expresión donde se está usando y var define una variable global o local en una función sin importar el ámbito del bloque.

Responder

1 **enzzoperez** 0 Puntos — 2 meses

Hola, que tal?..Me perdi un poco en el tema de los decoradores.

```
Lo que va a hacer nuestros decorator @enumerable(false) es verificar si el tipo de dato no es enum.
Deserlo, nopodrá implementarelmétodogreet()
```

Porque el metodo greet() del ejemplo no podria implementarse si el parametro que le pasas a enumerable() es enum?, la relacion entre la funcion greet() y enumerable() es lo que no me queda claro... Lo que yo entendi es que en el ejemplo lo que se hace es solo pasarle un parametro a la funcion enumerable()

0 **ThespianArtist** — 2 meses

Lo que hace ese decorator es como un tipo una validacion .

Si es false, quiere decir que va a ejecutarse si el valor no es enumerable, si es true lo que va a hacer es ejecutarse.

Un anotacion nos permite que una funcion o clase se comporte mediante ciertos parametros iniciales. Mas adelante cuando veas el primer componente en Angular notaras que usa del decorator @component.

Responder

1 **eleanatorresortiz** 0 Puntos — 2 meses

Hay una mejora considerable en este curso con respecto a la primera versión, Jorge Cano es mucho mejor instructor. Hasta ahora voy súper bien 😊

Responder

codetechie 2 Puntos — 2 meses

1

la configuración del archivo tsconfig.json me recuerda a los makefiles, me gusta el concepto

Responder

andresnator 28 Puntos — 2 meses

0

Me gusta que se realice un resumen, gracias

Responder

Luzadri 0 Puntos — 3 meses

0

la verdad es la primera vez que veo Typescript. ¿Esto es indispensable para trabajar en Angular?

ThespianArtist — 3 meses

0

No es necesario, puedes usar JavaScript normal o inclusive Dart:

http://stackoverflow.com/questions/30689094/is-it-possible-to-use-es5-javascript-with-angular-2-instead-of-typescript

Con Typescript no lo veas como otro lenguaje, es una versión con más cosas de ECMAScript 6. 😊

Responder

kami_lml 0 Puntos — 3 meses

0

Yo también tengo esa duda... por qué Angular usa TS, por qué React usa JSX (aunque es opcional), por qué no simplemente usan JS ambos y ya?

sergiodxa — 3 meses

4

En el caso de React este internamente es JS, JSX es solo para facilitar el crear componentes ya que sin JSX para crear un div harías esto:

```
React.DOM.div({ id: 'my-div' }, 'hello world');
```

Mientras que con JSX hacemos:

```
<div id="my-div">hello world</div>
```

Ver más...

Responder

jancarlo 11 Puntos — 3 meses

0

Es la primera vez que codeo usando TypeScript.

Actualmente uso ECMAScript 6 para codear con Nodejs, ¿Por qué en Angular se decide por TS? ¿El proyecto MEAN será totalmetne en TS? 😟

ThespianArtist — 3 meses

2

Solo Angular usa Typescript. pero no lo veas como otro lenguaje, es una versión con más cosas de ECMAScript 6.

Responder