Módulo 1 - Lectura: Introducción a Typescript

Angular y Nativescript están construidos en un lenguaje similar a JavaScript llamado Typescript. Typescript no es un lenguaje completamente nuevo, es un superconjunto de ES6. Si escribimos código ES6, es perfectamente código Typescript válido y compilable.

En general, muy pocos navegadores ejecutarán ES6 de manera inmediata, y mucho menos Typescript. Para resolver este problema tenemos transpilers (o algunas veces llamados transcompiler). El transpiler de Typescript toma nuestro código Typescript como entrada y genera un código ES5 que casi todos los navegadores comprenden.

Para convertir Typsacript a ES5 hay un solo transpiler escrito por el equipo Core de Typescript.
Sin embargo, si quisiéramos convertir el código ES6 (no Typescript) a ES5, hay dos principales transpilers de ES6 a ES5: traceur de Google y babel creado por la comunidad abierta de JavaScript.

Typescript es una colaboración oficial entre Microsoft y Google. Esa es una gran noticia porque, con dos pesos pesados de tecnología detrás, sabemos que se mantendrá durante mucho tiempo. Ambos grupos se comprometen a hacer avanzar la web y, como desarrolladores, ganamos por ello.

Una de las mejores cosas de los transpilers es que permiten que equipos relativamente pequeños hagan mejoras a un lenguaje sin requerir que todos en Internet actualicen su navegador.

¿Por qué deberíamos utilizar Typescript en absoluto? Porque hay algunas grandes características en Typescript que hacen que el desarrollo sea mucho mejor.

¿Qué conseguimos con Typescript?
Hay cinco grandes mejoras que Typescript trae sobre ES5:
• tipos
• clases
• decoradores
• importaciones
• utilidades de lenguaje (por ejemplo, desestructuración)
Vamos a tratar esto de uno en uno.

Los tipos
La mejora más importante de Typescript sobre ES6 es que da nombre al lenguaje, es la tipificación. Para algunas personas, la falta de verificación de tipos se considera uno de los beneficios de usar un lenguaje como JavaScript. Es posible que te muestres un poco escéptico respecto a la verificación de tipos, pero te recomiendo que le des una oportunidad.
Una de las grandes cosas de la verificación de tipos es que:
1. ayuda al escribir código porque puede prevenir errores en tiempo de compilación y

2. ayuda al leer el código porque aclara tus intenciones.
También vale la pena señalar que los tipos son opcionales en Typescript. Si queremos escribir algún código rápido o como prototipo de una característica, podemos omitir tipos y agregarlos gradualmente a medida que el código se vuelve más maduro.

Los tipos básicos de Typescript son los mismos que hemos estado usando implícitamente cuando escribimos código "normal" JavaScript: cadenas, números, booleanos, etc.

Hasta ES5, definiríamos variables con la palabra clave var, como var fullName ;.
La nueva sintaxis de Typescript es una evolución natural de ES5, todavía usamos var, pero ahora podemos opcionalmente proporcionar el tipo de variable junto con su nombre:
var fullName: string;

Al declarar funciones podemos usar tipos para argumentos y valores de retorno:
function saludar(nombre: string): string {
return "Hola" + nombre;
}

En el ejemplo anterior, estamos definiendo una nueva función llamada saludar que toma un argumento: nombre. La sintaxis nombre:string dice que esta función espera que el nombre sea una cadena. Nuestro código no compila si llamamos a esta función con otra cosa que no sea una cadena y eso es algo bueno porque, de lo contrario, introduciríamos un error.

Observa que la función saludar también tiene una nueva sintaxis después del paréntesis: string {. Los dos puntos indica que especificaremos el tipo de retorno para esta función, que en este caso es una cadena. Esto es útil porque

1. si accidentalmente devolvemos algo más que una cadena en nuestro código, el compilador nos dirá que cometimos un error y
2. cualquier otro desarrollador que quiera usar esta función sabe exactamente qué tipo de objeto obtendrán.

Entonces, ahora que sabemos cómo usar los tipos, ¿cómo podemos saber qué tipos están disponibles para usar? Vamos a mirar la lista de tipos incorporados y luego descubriremos cómo crear el nuestro.

Probemos con un REPL
Para jugar con los ejemplos de este capítulo, instalemos una pequeña utilidad agradable llamada TSUN:
$ npm instalar -g tsun
Ahora ejecute tsun.
Ese pequeño > que aparece, es el indicador que indica que TSUN está listo para recibir comandos.
En la mayoría de los ejemplos a continuación, puede copiar y pegar en este terminal y seguirlo.

Strings
Una cadena contiene texto y se declara usando el tipo de cadena:
var nombre: string = 'Roger Federer';

Number
Un número es cualquier tipo de valor numérico. En Typescript, todos los números se representan como punto flotante.
El tipo para los números es el number:
var edad:number = 36;

Booleano
El valor booleano mantiene verdadero o falso como el valor.
var casado: boolean = true;

Any
Any es el tipo predeterminado si omitimos escribir para una variable dada. Tener una variable de tipo cualquiera permite recibir cualquier tipo de valor:
var algo: any = 'como cadena';
algo = 1;
algo = [1, 2, 3];

Vacío - void
Usar void significa que no hay ningún tipo esperado. Esto suele ser en funciones sin valor de retorno:
function guardar(nombre: string): void {
this.nombre = nombre;
}

Las clases
En JavaScript, la programación orientada a objetos de ES5 se realizó utilizando objetos basados en prototipos.
Este modelo no usa clases, sino que se basa en prototipos.
La comunidad de JavaScript ha adoptado una serie de buenas prácticas para compensar la falta de de clases. Un buen resumen de esas buenas prácticas se puede encontrar en Mozilla Developer Network JavaScript Guide, y puede encontrar una buena visión general de la Introducción a la Orientación a Objetos

Sin embargo, en ES6 finalmente tenemos clases incorporadas en JavaScript.
Para definir una clase, usamos la nueva palabra clave de clase “class” y le damos a nuestra clase un nombre y un cuerpo:

```
class Vehiculo {
}
```

Las clases pueden tener propiedades, métodos y constructores.

Propiedades
Las propiedades definen los datos adjuntos a una instancia de una clase. Por ejemplo, una clase llamada Persona podría tener propiedades como nombre, apellido y edad.

Cada propiedad en una clase puede tener opcionalmente un tipo. Por ejemplo, podríamos decir que el nombre y apellido son cadenas y la propiedad edad es un número.

Métodos

Los métodos son funciones que se ejecutan en el contexto de un objeto. Para llamar a un método en un objeto, primero tenemos que tener una instancia de ese objeto.
Para crear una instancia de una clase, usamos la palabra clave new. Usamos new Persona() para crear una nueva instancia de la clase Persona, por ejemplo.

Si quisiéramos agregar una forma de saludar a una persona que usa la clase anterior, escribiríamos algo como:

```
class Persona {
nombre: string;
saludar () {
console.log ("Hola ", this.nombre);
}
}
```

Ten en cuenta que podemos acceder al nombre de esta Persona utilizando la palabra clave this y llamando a this.nombre.

Constructores

Un constructor es un método especial que se ejecuta cuando se crea una nueva instancia de la clase. Por lo general, el constructor es donde se realiza cualquier configuración inicial para nuevos objetos. Los métodos de constructor deben llamarse constructor. Opcionalmente pueden tomar parámetros, pero no puede devolver ningún valor, ya que se invocan cuando se crea una instancia de la clase (es decir, una instancia de dicha clase será retornada, por ende, no se puede devolver ningún otro valor).

Cuando una clase no tiene un constructor definido explícitamente, se creará automáticamente.

En Typescript solo puedes tener un constructor por clase.
Eso es una desviación de ES6 que permite que una clase tenga más de un constructor siempre y cuando tengan un número diferente de parámetros.
Los constructores pueden tomar parámetros cuando queremos parametrizar la creación de nuestra nueva instancia.

Herencia

Otro aspecto importante de la programación orientada a objetos es la herencia. La herencia es una forma de indicar que una clase recibe el comportamiento de una clase padre. Entonces podemos anular, modificar o aumentar esos comportamientos en la nueva clase.

Typescript es totalmente compatible con la herencia y, a diferencia de ES5, está integrado en el lenguaje central.
La herencia se logra a través de la palabra clave extends.

Utilidades

ES6, y por extensión, Typescript proporciona una serie de características de sintaxis que hacen que la programación sea muy agradable. Dos de las más importantes son:

• Sintaxis de la función de flecha ancha o *fat arrow function* o también llamadas *lambda functions*
• Strings multilínea.

Funciones de flecha ancha

Las funciones de flecha gorda => son una notación abreviada para funciones.
En ES5, siempre que queramos usar una función como argumento tenemos que usar la palabra clave function junto con llaves {}.
Sin embargo, con la sintaxis => podemos reescribirla así:
lineas.forEach ((linea) => console.log (linea));

Los paréntesis son opcionales cuando solo hay un parámetro.

Strings de múltiples líneas

En ES6 se introdujeron nuevas plantillas de cadenas. Las dos grandes características de las cadenas de plantillas o multilíneas son
1. Variables dentro de cadenas (sin ser forzado a concatenar con +) y
2. Strings multilínea

Esta función también se denomina "interpolación de cadenas". La idea es que puede colocar las variables directamente en sus strings. Así es cómo:
var saludo = `Hello $ {firstName} $ {lastName}`;

Ten en cuenta que para usar la interpolación de cadenas debes encerrar su cadena en backticks, y no usar ni comillas simples ‘ ni dobles “.

Terminando

Hay otras características en Typescript / ES6, tales como:

• Interfaces
• Genéricos
• Importación y exportación de módulos
• Decoradores
• Destructuraciones

No nos referiremos a estos conceptos en detalle, pero algunos serán vistos brevemente a medida que avancemos en diferentes tutoriales.

¡Éxitos!