Typescript - Intro

Prof. Sergio Bonato

Parcialmente baseado em https://www.typescripttutorial.net

Superset of JavaScript

Strongly Typed

Modular

Tooling Support

Scalable Application Structure

ES6 Support

Rodando Typescript no Node.JS

- Como vimos, o Typescript é um superconjunto do Javascript criado e mantido pela Microsoft: tipo um Javascript com cara de C#
- O Node.JS roda Javascript, não Typescript; então, é necessário ***transpilar*** o código, isto é, transformar o código Typescript em Javascript

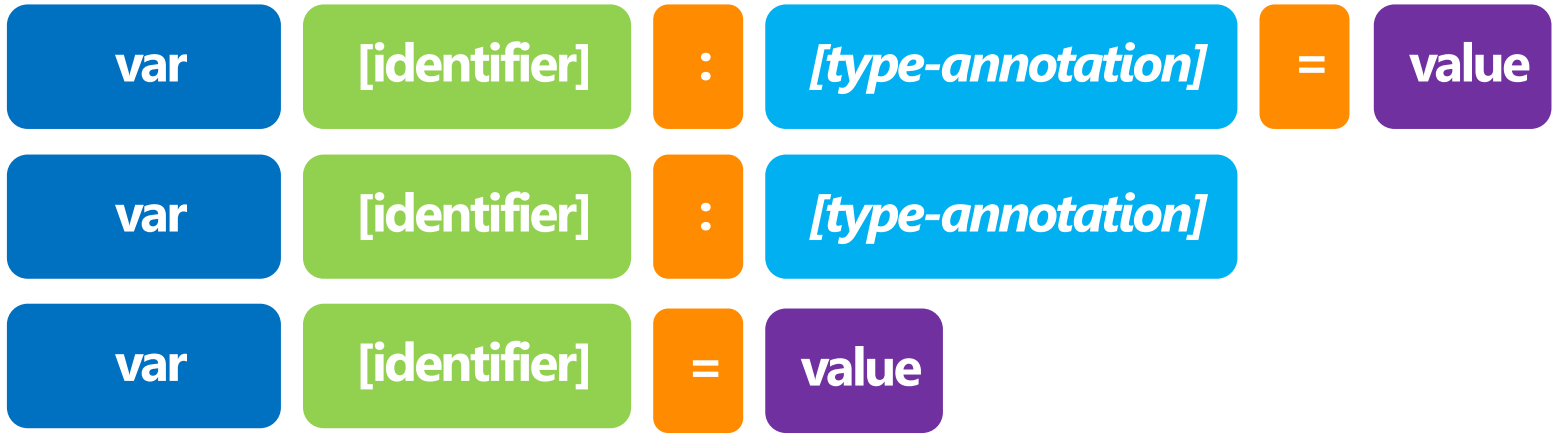
Para isso, instale na sua linha de comando:

```
> npm install –g typescript
> npm install –g ts-node
```

Para rodar um código typescript que esteja em um arquivo **index.ts**, digite na linha de comando

```
> ts-node index.ts
```

Obs: se quiser rodar pelo VS-Code siga o tutorial que está em
https://imasters.com.br/desenvolvimento/typescript-configurando-o-vs-code-para-escrever-rodar-e-debugar




```
var n: number;

var a;                  // no type -> Any

var s = "Max";          // Contextual typing -> string

n = 5;                  // valid because 5 is a number
a = 5;                  // valid because a is of type Any
a = "Hello";            // valid because a is of type Any

n = "Hello";            // compile time error because
                        // "Hello" is not a number
```

Tipos Básicos

Any

Tipos Primitivos

Number
Boolean
String

Tipagem Contextual

Determina o resultado do tipo de uma expressão automaticamente

```
var person = function (this: any, age: number) {
    this.age = age;

    this.growOld = function () {
        this.age++;
        console.log(this.age);
    }

    this.growOldL = () => {
        this.age++;
        console.log(this.age);
    }
}

var p = new (person as any)(1);
p.growOldL();
console.log(p.age);
```

Tipos Básicos

Função Lambda também conhecida como Arrow Function

- Elimina a necessidade de tipagem de funções
- Captura o significado lexicamente

```
function getAverage(a: number, b: number, c?:number){
    var total = a + b;
    var average = total / 2;
    if (c) {
        total = total + c;
        average = total / 3;
    }
    return average;
}

console.log(getAverage(3, 5));
console.log(getAverage(3, 5, 7));

function getAverages(...a: number[]):number {
    var average: number = 0;
    var total: number = 0;
    for(var i = 0; i < a.length; i++){
        total += a[i];
    }
    if (a.length > 0){
        average = total/a.length;
    }
    return average;
}

console.log(getAverages(3, 5));
console.log(getAverages(1,2,3,4,5,6,7,8,9,10));
```

Tipos Básicos

Funções

Parâmetros Opcionais
Parâmetros Default

Parâmetros Rest
Permite ao chamador especificar zero ou mais argumentos para o tipo especificado.

```
function getTotal(a: string, b:string, c:string): number;
function getTotal(a: number, b:number, c:number): number;
function getTotal(a: number, b:string, c:number): number;

function getTotal(a: any, b: any, c: any): number{
    var total = parseInt(a, 10) + parseInt(b, 10) + parseInt(c, 10);
    return total;
}

var result = getTotal(2, 2, 2);
console.log(result);
result = getTotal("3", "3", "3");
console.log(result);
result = getTotal(4, "4", 4);
console.log(result);
```

Tipos Básicos

Funções

Sobrecarga

```typescript
interface IStudent {
    id: number;
    name: string;
    onLeave?: boolean;
}
function printStudent(s: IStudent) {
}

// Describing function types

interface searchFunction {
    (source: string, subString: string):boolean
}

var searchFunctionImpl: searchFunction = function (s, ss)
{
    return true;
}
```

Tipos Básicos

Interfaces

Uma interface pode ser usada como um tipo abstrato que pode ser implementado por classes concretas, mas que também pode definir uma estrutura em um programa Typescript.

Interfaces também podem definir tipos função.

```typescript
class Student {
    private name: string;

    constructor(name: string, age: number) {
        this.name = name;
    }

    print() {
        console.log(this.name);
    }
}

class SpecialStudent extends Student {
    constructor (nome: string, age: number){
        super(nome, age);
    }

    print() {
        console.log("estudante especial");
    }
}

var estudante: Student = new Student("João", 23);
estudante.print();
var est_especial: SpecialStudent = new SpecialStudent("Maria", 18);
est_especial.print();
var est_especial2: Student = new SpecialStudent("José", 15);
est_especial2.print();
```

Tipos Básicos

Classes

Classes em Typescript viram pseudo classes em Javascript.

http://javascript.info/tutorial/pseudo-classical-pattern

Herança

Se faz com o uso de extends; note a chamada super

Sobreposição

Método com mesma assinatura

```
class Pessoa {
    private cpf: string;
    private nome: string;
    private sobrenome: string;
    readonly idade: number;

    constructor(cpf: string, nome: string, sobrenome: string, idade: number){
        this.cpf = cpf;
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.idade = idade;
    }

    public getNomeCompleto(): string {
        return `${this.nome} ${this.sobrenome}`
    }

    public getCpf(): string{
        return this.cpf;
    }

    protected setCpf(cpf: string){
        if (!cpf){
            throw new Error('CPF inválido');
        }
        this.cpf = cpf;
    }
}
```

Modificadores de Acesso

private – mesma classe

protected – classe e subclasses

public – acesso total

readonly – cria uma constante; só pode ser inicializada na declaração ou no construtor

Getters & Setters

Nota: por padrão, os atributos devem ser não nulos; caso você não os inicialize na sua declaração ou no construtor, ocorrerá um erro de compilação; para que isso não ocorra, ponha um ponto de exclamação (!) depois do nome do atributo; este ponto se chama *bang operator.*

```
class Employee {
    private static headcount: number = 0;

    constructor(
        private firsName: string,
        private lastName: string,
        private jobTitle: string){

        Employee.headcount++;
    }

    public static getHeadcount() {
        return Employee.headcount;
    }
}

let pedro = new Employee('Pedro', 'Silva', 'Desenvolvedor Front-End');
let maria = new Employee('Maria', 'Souza', 'Desenvolvedor Back-End');

console.log(Employee.getHeadcount); //imprime 2
```

Métodos e Propriedades Estáticas

Pertencem à classe.

As propriedades tem o mesmo valor para todas as instâncias.

Note que os atributos da classe estão sendo criados no construtor; é um modo alternativo; mas a ausência de modificador de acesso faz com que o parâmetro não seja visto como um atributo.

```
abstract class A {
  foo(): number { return this.bar(); }
  abstract bar(): number;
}

// error, Cannot create an instance of the abstract class 'A'
var a = new A();



class B extends A {
  bar() { return 1; }
}



var b = new b();  // success, all abstracts are defined
```

Tipos Básicos

Classes Abstratas

Similares a interfaces, mas com código implementado, permitindo criar uma classe base para outras.

```
// Internal Modules.

module Shipping {

    export interface Ship {
        name: string;
        tons: number;
    }

    export class NewShip implements Ship {
        name = "New Ship";
        tons = 500;
    }
}

// Splitting into multiple files.

/// <reference path="Shipping.ts" />
```

Tipos Básicos

Modulos

Fornecem vários modos de organizar o código em Typescript

Módulos internos
Módulos externos

```
// External Modules.

export class Ship {
    name = "New Ship";
    tons = 500;
}

// ---------------------------------

import Shipping = require('Ship')
var s = new Shipping.Ship();

// ---------------------------------
```

Tipos Básicos

Modulos

Fornecem vários modos de organizar o código em Typescript

Módulos internos
Módulos externos

```
class MyContainer<T> {

    private array: T[];

    constructor(array: T[]) {
        this.array = array;
    }

    add(item: T) {
        this.array.push(item);
    }
}

var strContainer = new MyContainer<number>([1]);
strContainer.add(2);
```

Generics

Permite criar componentes que trabalham com vários tipos diferentes.