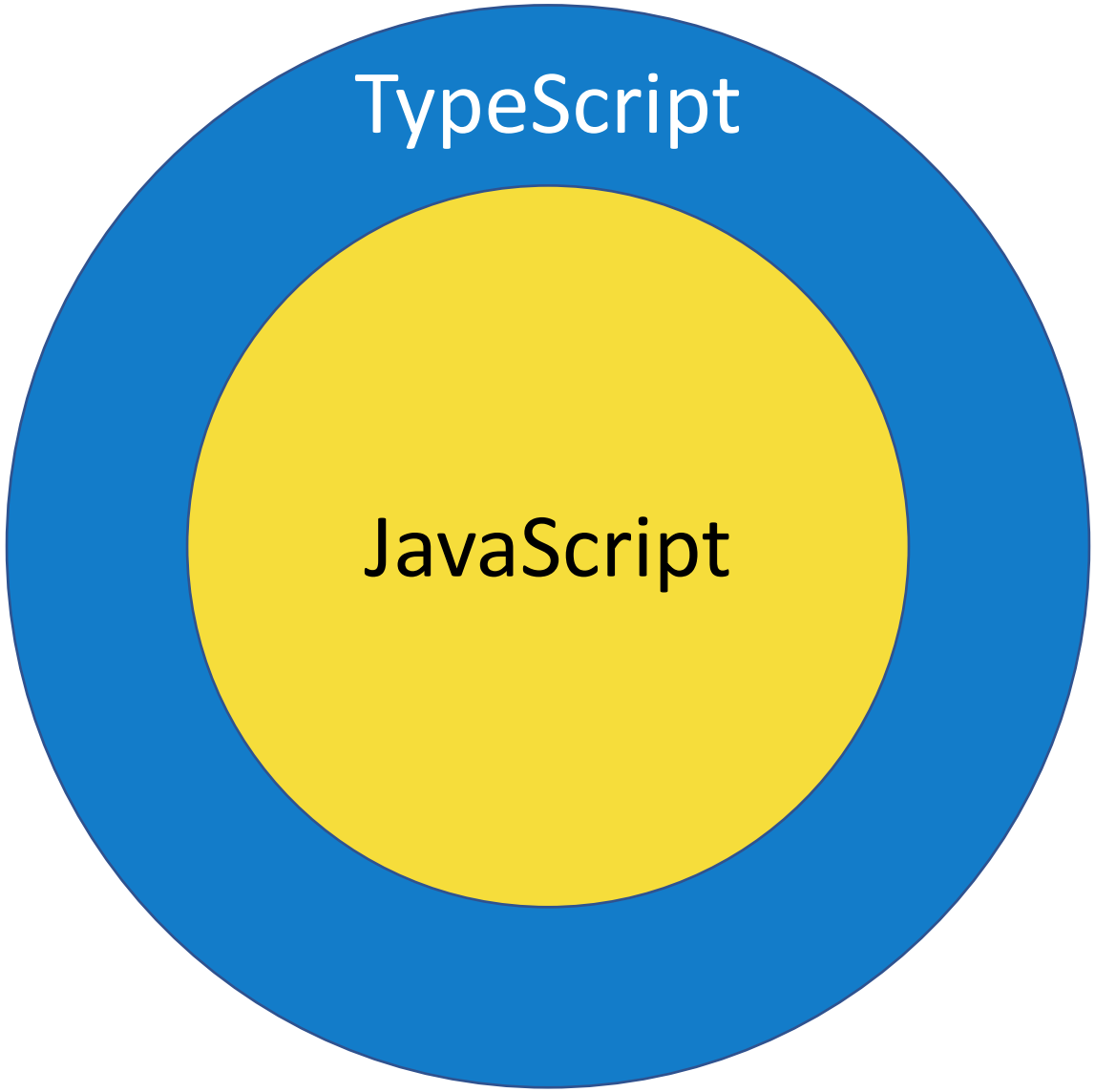
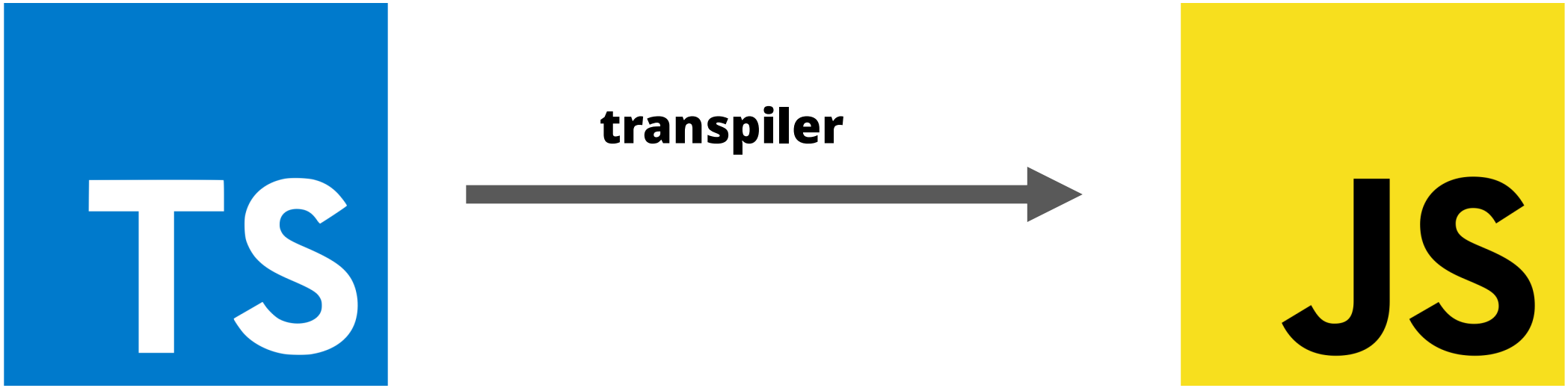
Typescript per sviluppatori c#

Un confronto tra il linguaggio che sta conquistando il web ed uno dei più usati linguaggi di programmazione orientati agli oggetti.

STEFANO VENA


TypeScript
JavaScript


TS
transpiler
JS

TypeScript

- Strongly typed
- Object-Oriented features
- Compile-time errors

Tools

- Visual Studio Code (https://code.visualstudio.com/)
- Webpack (https://webpack.js.org/)

Installazione e verifica

Tipi e Variabili

C#

```
1 bool userMan = true;
2 int userAge = 81;
3 float userAverage = 10.5f;
4 string userName = "Giuseppe Verdi";
```

TypeScript

```
1 let userMan: boolean = true;
2 var userAge: number = 81;
3 let userAverage: number = 10.5;
4 let userName: string = "Giuseppe Verdi";
```

Tipi inferiti (inferred types)

C# e Typescript

```
1 //inferred types
2 var foo = "bar";
3
4 //errore! foo è una stringa
5 foo = 1;
```

let e var

var

```
1 function varTest() {
2   var x = 31;
3   if (true) {
4     var x = 71; //è la stessa di sopra
5     console.log(x);  // 71
6   }
7   console.log(x);  // 71
8 }
```

let

```
1 function letTest() {
2   let x = 31;
3   if (true) {
4     let x = 71;  // different variable
5     console.log(x);  // 71
6   }
7   console.log(x);  // 31
8 }
```

String Interpolation

C#

```
1 string userFirstName = "Stefano";
2 string userLastName = "Vena";
3 string userFullName  = $"Hello {userFirstName} {userLastName}!";
```

TypeScript

```
1 let userFirstName = "Stefano";
2 let userLastName = "Vena";
3 let userFullName  = `Hello ${userFirstName} ${userLastName}!`;
```

Lambda expressions/ fat arrow functions

C#

```
/** Funzioni lambda */
List<int> numbersList = new List<int> {1,2,3,4,5,6,7,8};
IEnumerable<int> query = numbersList.Where(value => value > 2);

foreach (int value in query)
{
    Console.WriteLine(value);
}
```

TypeScript

```
 /** Funzioni lambda */
var numbersList : number[] = [1,2,3,4,5,6,7,8];
let query: number[] = numbersList.filter((value: number) => value > 2);

query.forEach(element => {
    console.info(element);
});
```

Funzioni base su stringhe

C#

```
/* metodi stringa  */
 String myEmail = "stefano.vena@gmail.com";
        myEmail.IndexOf("@");
        myEmail.Replace("@", " at ");
        myEmail.Substring(0, 5);
     if( myEmail.Length > 0 ) ...

 int intero = Int32.Parse("10");
```

TypeScript

```
//* metodi stringa  */
let myEmail:string = "stefano.vena@gmail.com";
    myEmail.indexOf("@");
    myEmail.replace("@", " at ");
    myEmail.substring(0, 5);
    if( myEmail.length > 0 ) ...
let intero: number = parseInt("10");
```

Classi

C#

```csharp
public class User {
  public string FirstName;
  public User(string firstName) {
    FirstName = firstName;
  }
  public void SayHello() {}
}

User myUser = new User("Stefano");
myUser.FirstName;
myUser.SayHello();
```

TypeScript

```typescript
class User {
  public firstName: string;
  public constructor(firstName: string) {
    this.firstName = firstName;
  }
  public sayHello(): void {}
}

let myUser: User = new User("Stefano");
myUser.firstName;
myUser.sayHello();
```

Classi, Interfacce, Classi astratte

C#

```
1 interface Movable {
2   void Move();
3 }
4
5 public class Vehicle: Movable {
6     public void Move() {}
7 }
8
9 public abstract class Test {}
```

TypeScript

```
1 interface Movable {
2   public Mve(): void;
3 }
4
5 class Vehicle implements Movable {
6     public move(): void {}
7 }
8
9 abstract class Test {}
```

Classi

C#

```
1 public class Editor: User {
2   public Editor(string firstName) {
3     base(firstName);
4   }
5   public override void SayHello() {
6     base.SayHello();
7   }
8 }
```

TypeScript

```
1 class Editor extends User {
2   public constructor(firstName: string) {
3     super(firstName);
4   }
5   public sayHello(): void {
6     super.sayHello();
7   }
8 }
```

Classi – getter e setter

C#

```
1 public class User {
2   private string name;
3   public string Name {
4     get {
5         return name;
6     }
7     set {
8       name = value;
9     }
10  }
11 }
12 User myUser = new User("Stefano");
13 myUser.Name;
14 myUser.Name = "Nuccio";
```

TypeScript

```
1 class User {
2   protected _name: string;
3
4   public get name(): string {
5     return this._name;
6   }
7
8   public set name(newName: string) {
9     this._name = newName;
10  }
11 }
12
13 let myUser: User = new User("Andrea");
14 myUser.name;
15 myUser.name = "Capomastro";
```

Classi – namespaces

C#

```cs
//member.cs
namespace Meetup
{
    public class Member {}
}

//esempio.cs
using Meetup;

Member andrea = new Member();
```

TypeScript

```ts
// memeber.ts
export class Member {}

// esempio.ts
import { Member } from "./member";

let stefano: Member = new Member();
```

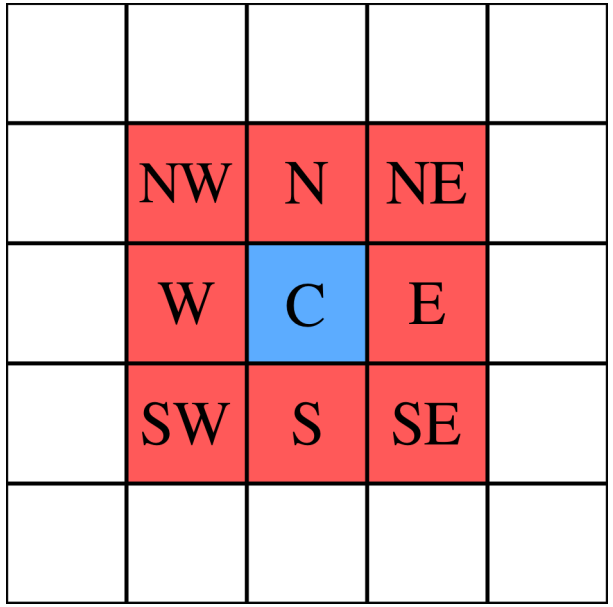
Conway’s game of Life in typescript

Comportamento complesso, regole semplici.

- Insieme di celle che possono vive o morte (assumono lo stato 0 o 1)

- Ogni cella ha 8 vicini (vicinato di Moore)

Ref. https://it.wikipedia.org/wiki/Gioco_della_vita



Comportamento complesso, regole semplici.

- Qualsiasi **cella viva con meno di due celle vive** adiacenti **muore**, come per effetto d'isolamento;
- Qualsiasi **cella viva** con **due o tre celle vive** adiacenti **sopravvive** alla generazione successiva;
- Qualsiasi **cella viva** con **più di tre celle vive** adiacenti **muore**, come per effetto di sovrappopolazione;
- Qualsiasi **cella morta** con esattamente **tre celle vive** adiacenti diventa una cella **viva**, come per effetto di riproduzione.

Implementazione – Interfaccia Cella

```
1 export interface ICell<T> {
2     status: T;
3     evolve(neighbours: ICell<T>[]): T;
4 }
```

Cella di Conway

```
export class ConwayLifeCell implements ICell<number> {
    //... alcune cose sono omesse ...///
    public evolve ( neighbours: ConwayLifeCell[] ) : number {
        //Devo levare la cella attuale dal conto
        var alives  = -this.status;
        neighbours.forEach( v => {
           if(v && v.alive){
              alives++;
           }
         });

        if( this.alive && alives < 2){
             return 0;
         }

         if( this.alive && alives > 3 ){
            return 0;
        }

        if( this.alive && (alives == 3 || alives == 2)  ){
            return 1;
        }

        if( !this.alive && alives == 3  ){
            return 1;
        }

        return this.status;
    }

}
```

Evoluzione

```
export class World<U, T extends ICell<U>> {

    private _layer: number = 0;
    private _space: T[][][] = [[], []];

    private _width: number;
    private _height: number;

    public radius: number = 1;
    /**
     * Dal momento che tutte le informazioni sui tipi vengo rimosse nella conversione in javascript
     dobbiamo passare le informazioni sul tipo nel costruttore.
     * con "private _cellType: new () => T" oltre a dichiarare il campo nella firma
     del meteodo lo aggiungiamo anche ai membri della classe.
    */
    public constructor(private _cellType: new () => T, width: number, height: number, radius: number = 1) {
        // [ ... ]
        for (var x: number = 0; x < this._width; x++) {
            this._space[0][x] = new Array(this._height);
            this._space[1][x] = new Array(this._height);
            for (var y: number = 0; y < this._height; y++) {
                this._space[0][x][y] = this.getNew();
                this._space[1][x][y] = this.getNew();
            }
        }
    }

    /** metodo per instanziare una nuova cella */
    private getNew(): T {
        return new this._cellType();
    }


    public evolve() {

        let neighbours: Array<T> = new Array((2 * this.radius + 1) * (2 * this.radius + 1));
        let next_layer = 1 - this._layer;

        //Elaboriamo lo stato di tutte le celle dello spazio.
        for (var x: number = 0; x < this._width; x++) {
            for (var y: number = 0; y < this._height; y++) {
                /** per ogni cella (x,y) calcoliamo i vicini */
                let position = 0;
                for (var i = x - this.radius; i <= x + this.radius; i++) {
                    for (var j = y - this.radius; j <= y + this.radius; j++) {
                        let a = (i + this._width) % this._width;
                        let b = (j + this._height) % this._height;
                        neighbours[position] = this._space[this._layer][a][b];
                        position++;
                    }
                }
                this._space[next_layer][x][y].status = this._space[this._layer][x][y].evolve(neighbours);
            }
        }

        this.swap();
    }

    //[ ... ]
}
```

Riferimenti

- https://github.com/wetfire2k/tslife

- https://www.typescriptlang.org/docs/home.html
- https://carbon.now.sh
- https://webpack.js.org/

Grazie