

# TypeScript

Sergio Álvarez Alonso  
Adrián Mínguez Graña  
Mario Morano Orviz  
Ángel Manuel Méndez Campal

Máster ingeniería web  
Universidad de Oviedo

Diciembre 2018

# Contexto y motivación del lenguaje I

- ▶ Lanzado primera vez en 2012 (versión 0.8)
- ▶ 2 años de desarrollo interno en Microsoft
- ▶ Limitaciones de js para desarrollos de gran escala
- ▶ “Adelanto” de lo que cabría esperar de ECMAScript2015
- ▶ Anders Hejlsberg (LA de C#) involucrado en el desarrollo
- ▶ Se transcompila a javascript
- ▶ Licencia Apache 2 (open source)
- ▶ Versión estable 3.1.6, 1 Noviembre



## Herramientas

Al principio sólo Visual Studio. Luego Eclipse, Emacs, Vim, Sublime, Webstorm, Atom y Visual Studio Code.

# Contexto y motivación del lenguaje II

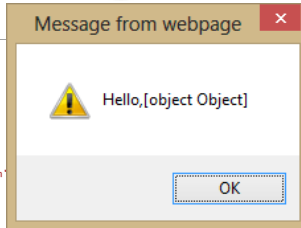
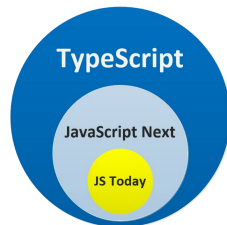
## Objetivos:

- ▶ Dotar js con un sistema opcional de tipos
- ▶ Proveer a los motores de js actuales de características planeadas para versiones futuras

Superset de javascript: .js → .ts y compilar devuelve js válido

```
1 function Greeter(greeting) {  
2   this.greeting = greeting;  
3 }  
4 Greeter.prototype.greet = function() {  
5   return "Hello," + this.greeting;  
6 }  
7 var greeter = new Greeter({message: "world"});  
8 var button = document.createElement('button');  
9 button.innerText = "Say Hello"  
10 button.onclick = function() {  
11   alert(greeter.greet())  
12 }  
13 document.body.appendChild(button)  
14
```

```
1 function Greeter(greeting) {  
2   this.greeting = greeting;  
3 }  
4 Greeter.prototype.greet = function () {  
5   return "Hello," + this.greeting;  
6 }  
7 var greeter = new Greeter({  
8   message: "world"  
9 });  
10 var button = document.createElement('button'  
11 button.innerText = "Say Hello";  
12 button.onclick = function () {  
13   alert(greeter.greet());  
14 }  
15 document.body.appendChild(button);  
16
```



# Contexto y motivación del lenguaje III

Añade anotación de tipos que se borra al transcopilar

Más información en tiempo de compilación

```
1 function Greeter(greeting : string) {
2   this.greeting = greeting;
3 }
4
5 Greeter.prototype.greet = function() {
6   return "Hello, " + this.greeting;
7 }
8
9 var greeter = new Greeter({message: "world"})
10 Supplied parameters do not match any signature of
11 call target: Could not apply type 'string' to
12 argument 1, which is of type '{ message: string; }'
13 (greeting: string) -> void
14   alert(greeter.greet())
15 }
16 document.body.appendChild(button)
```

```
1 function Greeter(greeting) {
2   this.greeting = greeting;
3 }
4
5 Greeter.prototype.greet = function () {
6   return "Hello, " + this.greeting;
7 };
8
9 var greeter = new Greeter({
10   message: "world"
11 });
12
13 var button = document.createElement('button');
14 button.innerText = "Say Hello";
15 button.onclick = function () {
16   alert(greeter.greet());
17 };
18
19 document.body.appendChild(button);
```

Declaración de clases y modularidad

```
1 class Greeter {
2   greeting : string;
3   constructor(message : string) {
4     this.greeting = message;
5   }
6   greet() {
7     return "Hello" + this.greeting;
8   }
9 }
10
11 var greeter = new Greeter("world");
12 var button = document.createElement('button');
13 button.innerText = "Say Hello";
14 button.onclick = function () {
15   alert(greeter.greet());
16 }
17 document.body.appendChild(button)
```

```
1 var Greeter = (function () {
2   function Greeter(message) {
3     this.greeting = message;
4   }
5   Greeter.prototype.greet = function () {
6     return "Hello" + this.greeting;
7   };
8   return Greeter;
9 })();
10
11 var greeter = new Greeter("world");
12 var button = document.createElement('button');
13 button.innerText = "Say Hello";
14 button.onclick = function () {
15   alert(greeter.greet());
16 };
17 document.body.appendChild(button);
```

IntelliSense

```
1 function Greeter(greeting : string) {
2   greeting.
3
4   charAt (pos: number) => string
5   (pos: number) => string
6
7   charCodeAt
8   concat
9   indexOf
10  lastIndexOf
11  length
12  localeCompare
13  match
```

Sintaxis de declaración de tipos de ECS6  
Produce código js preparado para hacer herencia por prototipos



Escalar sitios web a aplicaciones más grandes

# Principales características I

- ▶ Orientación a objetos
- ▶ Transcompilación a js
- ▶ Fuertemente tipado (estático) y comprobación de tipos en tiempo de ejecución
- ▶ Primitivos: number, boolean, string
- ▶ Tipo any (tipado débil, dinámico) similar EC6
- ▶ npm gestor de dependencias
- ▶ cross platform
- ▶ TSC: compilador escrito en ts
- ▶ Tipos enum, void. Inferencia de tipos

```
/** Funcion con tipo de retorno number. */  
funcion restar (n1: number, n2: number):  
    number {return n1 - n2;}  
  
/** Funcion sin tipo de retorno. Se infiere de los  
    parametros y el return */  
funcion restar () (n1: number, n2:number)  
    {return n1 - n2;}
```

# Principales características II

Posibilidad de definir tus propios "tipos" a partir de otros tipos primitivos.

```
type Contador = number | string;

funcion mostrarAsignaturas (asignatura: Contador): string
{
    return `Tengo ${asignatura} asignaturas.`;
}
```

Se define contador de tipo number o string. El parámetro de entrada de la función es de tipo contador por lo que puede ser uno de ambos tipos primitivos.

Soporta parámetros opcionales

```
function funcionConParametroOpcional(nombre: string,
    age?: number)
```

## Principales características III

Interfaces. Siguiendo la filosofía de ser un lenguaje orientado a objetos y basado en clases. Durante la transcompilación de código .ts a .js, las interfaces no generan código EC6. Simplemente se utilizan durante el desarrollo typescript a modo de "contratos".

```
interface Persona {  
    nombre: string;  
}  
  
let a: Persona = {  
    nombre: 'Pedro'  
}
```

# Conclusiones

- ▶ Buena curva de aprendizaje
- ▶ Es necesario saber javascript (misma sintaxis y semántica)
- ▶ Permite reusar js y sus librerías
- ▶ Produce código limpio y garantiza compatibilidad con el navegador
- ▶ Escalabilidad
- ▶ Open source
- ▶ Comunidad activa
- ▶ Menos código que en js. Ahorra tiempo
- ▶ Código más legible
- ▶ En pleno desarrollo



# Referencias

- ▶ <https://blogs.msdn.microsoft.com/somasegar/2012/10/01/typescript-javascript-development-at-application-scale/>
- ▶ TypeScript Deep Dive. Basarat Ali Syed.
- ▶ <https://fullstack-developer.academy/debugging-typescript-in-the-browser/>
- ▶ <https://www.typescriptlang.org/>
- ▶ <https://dzone.com/articles/what-is-typescript-and-why-use-it>