TypeScript

Mario Moraño Orviz

Adrián Mínguez Graña

Sergio Álvarez Alonso

Ángel Manuel Méndez Campal

Septiembre 2018

Resumen

Trabajo consistente en describir el lenguaje de programación Typescript y su relación en la asignatura, centrándose en al descripción sus principales características, el porqué de su creación y utilidad en el contexto de la web.

1. Contexto y motivación del lenguaje

Se trata de un lenguaje de programación que surge en el año 2012 de la mano de Microsoft, tras dos años de desarrollo interno de la compañía. La motivación de los ingenieros de Microsoft para desarrollar dicha tecnología consiste en la demanda de muchos desarrolladores web de contar con herramientas que ayudasen a escalar sus aplicaciones. El desarrollo web en javascript no era en ese año una experiencia agradable para el programador, y el escalado de sitios web hacia aplicaciones más grandes era una tarea dificultosa debido a la naturaleza del propio lenguaje. En conclusión, el uso de javascript en la web crecía exponencialmente y también la complejidad del software producido, sin que la herramienta contase con características que se ajustasen a la demanda de los programadores.

La solución por parte de Microsoft a dicho problema fue el desarrollo del lenguaje de programación Typescript, que permitía a los usuarios no tener que esperar a futuras versiones de ECMAScript para poder desarrollar con nuevas características de javascript, además de dotar al lenguaje con un sistema de tipado fuerte. En éste sentido se puede decir que el desarrollo de Typescript iba "por delante"de ECMAScript, y su punto fuerte es que al ser un superset de

javascript, todo el código javascript es de forma automática código typescript válido, gracias al compilador TSC.

```typescript
function Greeter(greeting) {
    this.greeting = greeting;
}
Greeter.prototype.greet = function() {
    return "Hello," + this.greeting;
}
var greeter = new Greeter({message: "world"});
var button = document.createElement('button');
button.innerText = "Say Hello"
button.onclick = function() {
    alert(greeter.greet())
}
document.body.appendChild(button)
```

```javascript
function Greeter(greeting) {
    this.greeting = greeting;
}
Greeter.prototype.greet = function () {
    return "Hello," + this.greeting;
};
var greeter = new Greeter({
    message: "world"
});
var button = document.createElement('button');
button.innerText = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};
document.body.appendChild(button);
```

Figura 1: Comparación ts vs js I.

Como se puede apreciar en la figura 1, el código generado al transcompilar typescript es prácticamente igual al código javascript, salvo algunas cuestiones semánticas como puntos y coma o espacios. Cabe destacar el que todo el codigo javascript es código typescript válido: sin más que renombrar el fichero .js a .ts y compilar con TSC se obtiene javascript válido.

Ocurre que el ejemplo código anterior contiene un error que sólo será detectado por javascript en tiempo de ejecución debido al carácter dinámico del lenguaje, y no durante el desarrollo. De este modo obtendremos el siguente pop-up al ejecutarlo con un navegador:

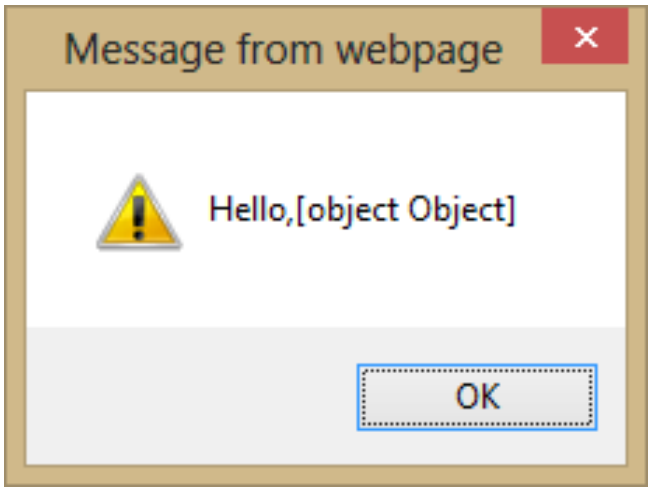

Figura 2: Error en t ejecución. JS dinámico

Con typescript se posibilita el añadir anotaciones de tipo a las variables definidas en el código del programa. Al introducir tipado estático la información que puede proveer el compilador y el abanico de herramientas de desarrollo disponible mejoran notablemente, proveyendo al usuario una experiencia mucho más completa que con javascript, como el hecho de disponer de mensajes

en el código y detección de errores en tiempo de compilacón. Además al compilar se eliminan todas las anotaciones "de tipos", con lo que el código resultante siempre será código javascript válido.

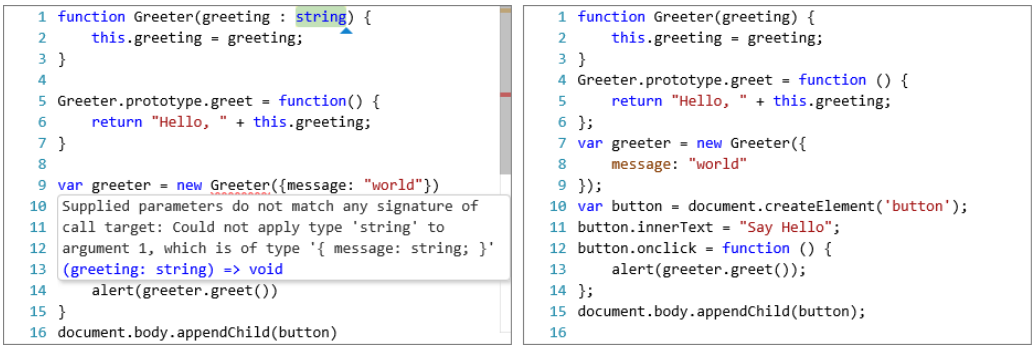

Figura 3: Errores inline gracias a TSC

Otra ca

2. Principales características

3. Ejemplo de aplicación

4. Motivación del lenguaje