




Typescript

 2025-01   45 min.

Tipado

Javascript es un lenguaje de programacion con tipado debil.

Tipado

"Tipado" hace referencia al posible requisito de un lenguaje de programación de declarar el tipo de una variable.

- Java tiene un tipado fuerte.
- JavaScript tiene un tipado débil.

Tipado

"Tipado" hace referencia al posible requisito de un lenguaje de programación de declarar el tipo de una variable.

- Java tiene un tipado fuerte.
- JavaScript tiene un tipado débil.
- **TypeScript tiene un tipado fuerte.**

Typescript

Typescript es un lenguaje de programacion que *transpila* a Javascript.

Ejemplo

```
1  function getUserAge(user, today) {  
2      return today - user.birthdate;  
3  }  
4  
5  // que es user? que es today? que es user.birthdate? que somos? que?
```

Ejemplo

Javascript

```
1  function getUserAge(user, today) {  
2      return today - user.birthdate;  
3  }
```

Typescript

```
1  function getUserAge(user: User, today: Date): number {  
2      return today - user.birthdate;  
3  }
```

Instalación

```
$ npm install typescript
```


Configuración de TypeScript

Se encuentra en el archivo `tsconfig.json` .

Ejemplo de Configuración

```
1  {
2    "compilerOptions": {
3      "module": "system",
4      "noImplicitAny": true,
5      "removeComments": true,
6      "preserveConstEnums": true,
7      "outFile": "../../built/local/tsc.js",
8      "sourceMap": true
9    },
10   "include": ["src/**/*"],
11   "exclude": ["node_modules", "**/*.spec.ts"]
12 }
```

Configuración de TypeScript

<https://www.typescriptlang.org/tsconfig>

Declaracion de una variable

Javascript

```
1 let variable1 = 1;  
2 const variable2 = "2";
```

Typescript

```
1 let variable1: number = 1;  
2 const variable2: string = "2";
```

Y los arrays?

Javascript

```
1 let variable1 = [1, 3, 5];
```

Typescript

```
1 // ?
```

Y los arrays?

Javascript

```
1 let variable1 = [1, 3, 5];
```

Typescript

```
1 let variable1: number[] = [1, 3, 5];
```

Y los objetos?

Javascript

```
1  let usuario = {  
2    name: "Anyelo",  
3    birthdate: "2002-10-05",  
4    semester: 7,  
5  };
```

Typescript

```
1  // ?
```

Y los objetos?

Javascript

```
1 let usuario = {  
2   name: "Anyelo",  
3   birthdate: "2002-10-05",  
4   semester: 7,  
5 };
```

Typescript

```
1 let usuario: {  
2   name: string,  
3   birthdate: Date,  
4   semester: number,  
5 } = {  
6   name: "Anyelo",  
7   birthdate: "2002-10-05",  
8   semester: 7,  
9 };
```


Y dos objetos?

```
1  let usuario1: {  
2    name: string,  
3    birthdate: Date,  
4    semester: number,  
5  } = {  
6    name: "Anyelo",  
7    birthdate: "2002-10-05",  
8    semester: 7,  
9  };  
10  
11 let usuario2: {  
12   name: string,  
13   birthdate: Date,  
14   semester: number,  
15 } = {  
16   name: "Emily",  
17   birthdate: "2002-11-06",  
18   semester: 5,  
19 };
```

Tipos

```
1  type User = {  
2    name: string,  
3    birthdate: Date,  
4    semester: number,  
5  };  
6  
7  let usuario1: User = {  
8    name: "Anyelo",  
9    birthdate: "2002-10-05",  
10   semester: 7,  
11  };  
12  
13  let usuario2: User = {  
14    name: "Emily",  
15    birthdate: "2002-11-06",  
16    semester: 5,  
17  };
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun Typescript)
2 let variable1: number;
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun typescript)
2 let variable1: number;
3
4 // segun TS, la variable es un numero,
5 // porque ahí lo dice
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun Typescript)
2 let variable1 = 1;
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun typescript)
2 let variable1 = 1;
3
4 // segun TS, la variable es numero,
5 // porque el valor dado a la variable es un numero
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun typescript)
2 let variable1;
```

Inferencia de tipos

```
1 // de que tipo es esta variable? (segun typescript)
2 let variable1;
3
4 // segun TS, es tipo "any"
```


« Any »

El tipo "any" significa cualquier cosa.

Se recomienda evitar utilizarlo.

Se recomienda no permitir que se infiera.

Declaracion de una función

```
1  function suma(a, b) {  
2      return a + b;  
3  }
```

```
1  function suma(a, b) {  
2      return a + b;  
3  }  
4  // cual es el tipo de a? b? a+b?
```

Declaracion de una función

```
1  function suma(a: number, b: number) {  
2      return a + b;  
3  }  
4  // cual es el tipo de a+b?
```

Declaracion de una función

```
1  function suma(a: number, b: number) {  
2      return a + b;  
3  }  
4  // cual es el tipo de a+b?  
5  // TS infiere que suma(a,b) retorna un numero
```

Declaracion de una función

```
1  function suma(a: number, b: number): number {  
2      return a + b;  
3  }
```

Declaracion de una función (arrow)

```
1  const suma = (a: number, b: number): number => {  
2    return a + b;  
3  };
```

Pero bueno volvamos a los tipos

Tipos

```
1  type User = {  
2    name: string,  
3    birthdate: Date,  
4    semester: number,  
5  };
```


Que pasa si...

Fecha de Nacimiento quizá no es una "Fecha"

```
1  type User = {  
2    name: string,  
3    birthdate: Date,  
4    semester: number,  
5  };
```

Puede ser "1990-07-15" .

Como permitimos que birthdate sea Date y string?

Que pasa si...

Fecha de Nacimiento quizá no es una "Fecha"

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5  };
```

El simbolo `|` nos permite decir que una variable o campo puede tener dos o mas tipos.

Que pasa si...

Queremos agregar telefono

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5  };
```

Pero no todos los usuarios van a tener telefono.

Como permitimos que `User` tenga telefono, pero que no sean todos?

Que pasa si...

Queremos agregar telefono

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6  };
```

El simbolo `?` nos permite decir que un campo es opcional.

Que pasa si...

Queremos usuarios administrador

```
type User = {  
  name: string,  
  birthdate: Date | string,  
  semester: number,  
  telephone?: number,  
};
```

Un administrador es como un usuario, pero tiene unos campos adicionales.

Que pasa si...

Queremos usuarios administrador

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6  };  
7  
8  type Admin = {  
9    name: string,  
10   birthdate: Date | string,  
11   semester: number,  
12   telephone?: number,  
13   employeeID: number,  
14   department: string,  
15  };
```

Que pasa si...

Queremos usuarios administrador

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6  };  
7  
8  type Admin = {  
9    name: string,  
10   birthdate: Date | string,  
11   semester: number,  
12   telephone?: number,  
13   employeeID: number,  
14   department: string,  
15  };
```

Pero entonces nos estamos repitiendo.

Que pasa si...

Queremos usuarios administrador

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6    employeeID?: number,  
7    department?: string,  
8  };
```

Podemos agregar los campos de Admin a User, y volverlos opcionales.

Que prefieren, este o este?

Tener dos tipos, y repetir los campos de usuario en administrador

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6  };  
7  
8  type Admin = {  
9    name: string,  
10   birthdate: Date | string,  
11   semester: number,  
12   telephone?: number,  
13   employeeID: number,  
14   department: string,  
15  };
```

Tener un tipo, y los campos de administrador son opcionales

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6    employeeID?: number,  
7    department?: string,  
8  };
```

Que pasa si...

Queremos usuarios administrador

```
1  type User = {  
2    name: string,  
3    birthdate: Date | string,  
4    semester: number,  
5    telephone?: number,  
6  };  
7  
8  type Admin = User & {  
9    employeeID: number,  
10   department: string,  
11  };
```

El simbolo `&` nos permite 'extender' un tipo en otro.

Type o Interface?

Son funcionalmente lo mismo.

Type o Interface?

```
1 interface Animal {  
2   name: string;  
3 }  
4  
5 interface Bear extends Animal {  
6   honey: boolean;  
7 }
```

```
1 type Animal = {  
2   name: string,  
3 };  
4  
5 type Bear = Animal & {  
6   honey: boolean,  
7 };
```

Type o Interface?

```
1 interface Window {  
2   title: string;  
3 }  
4 interface Window {  
5   ts: TypeScriptAPI;  
6 }
```

✓ Todo bien

```
1 type Window = {  
2   title: string,  
3 };  
4 type Window = {  
5   ts: TypeScriptAPI,  
6 };
```

⚠ Window ya existe

"Yo sé lo que es"

– Devs, 1997 - Hoy.

```
1  function func1(value: number): string | number {  
2      if (value < 5) {  
3          return "Error.";  
4      } else {  
5          return value;  
6      }  
7  }  
8  
9  console.log(func1(6) * 10);
```

"Yo sé lo que es"

– Devs, 1997 - Hoy.

```
1  function func1(value: number): string | number {  
2      if (value < 5) {  
3          return "Error.";  
4      } else {  
5          return value;  
6      }  
7  }  
8  
9  console.log(func1(6) * 10); // multiplicando una string?
```

"Yo sé lo que es"

– Devs, 1997 - Hoy.

```
1  function func1(value: number): string | number {  
2      if (value < 5) {  
3          return "Error.";  
4      } else {  
5          return value;  
6      }  
7  }  
8  
9  console.log(<number>func1(6) * 10)  
10 console.log(func1(6) as number * 10)
```


Tipos literales

```
1  function compara(a: number, b: number) {  
2    if (a === b) {  
3      return 0;  
4    } else if (a < b) {  
5      return -1;  
6    }  
7    return 1;  
8  }
```

Tipos literales

```
1  function compara(a: number, b: number): -1 | 0 | 1 {  
2      if (a === b) {  
3          return 0;  
4      } else if (a < b) {  
5          return -1;  
6      }  
7      return 1;  
8  }
```

Tipos literales

```
1  function compara(a: number, b: number): -1 | 0 | 1 {  
2    if (a === b) {  
3      return 0;  
4    } else if (a < b) {  
5      return -1;  
6    }  
7    return 1;  
8  }
```

Tambien se puede con strings.
Y con booleanos pero eso es como tonto.

Generics

```
1  function sum<X>(a: X, b: X): X {  
2      return a + b;  
3  }
```

Generics

```
1  function sum(a: number, b: number): number {  
2      return a + b;  
3  }  
4  // como permito que la funcion sum concatene  
5  // dos strings si es el caso?  
6  
7  console.log(sum(2, 3));  
8  console.log(sum("a", "b")); // Error
```

Generics

```
1  function sum<X>(a: X, b: X): X {  
2      return a + b;  
3  }  
4  // usando generics, podemos decir que ambos params  
5  // son un tipo, que tambien es el del resultado  
6  
7  console.log(sum<number>(2, 3));  
8  console.log(sum<string>("a", "b"));
```

Utility Types

- `Partial<Type>`
 - Todos los campos de `Type`, pero todos opcionales
- `Required<Type>`
 - Todos los campos de `Type`, pero todos obligatorios
- `Pick<Type, Keys> => Pick<User, "name" | "semester">`
 - Solo los campos `Keys` de `Type`
- `Omit<Type, Keys>`
 - Todos los campos de `Type`, excepto `Keys`



Saben Typescript!