Objetos TS

Antonio Espín Herranz

Contenidos

- Métodos de los Arrays

- Enlaces:
  - Date
  - String
  - Number
  - Bigint

- Tuple
- Object
- Function

Métodos de los Arrays

Método	Descripción
`push()`	Añade uno o más elementos al final del array y devuelve la nueva longitud.
`pop()`	Elimina el último elemento del array y lo devuelve.
`unshift()`	Añade uno o más elementos al principio del array y devuelve la nueva longitud del array.
`shift()`	Elimina el primer elemento del array y lo devuelve.
`concat()`	Combina el array actual con otro u otros y devuelve un nuevo array. El array original permanece sin cambios.
`join(separator)`	Convierte los elementos del array en una cadena y la devuelve. Puede especificar un separador para los elementos.

Métodos de los Arrays II

`slice(start, end)`	Crea una copia plana del array formada por los elementos entre los índices especificados `start` (inclusive) y `end` (exclusive). El array original permanece sin cambios.
`splice(start, deleteCount, element1, element2, ...)`	Inserta nuevos elementos en la posición especificada y/o elimina elementos del array.
`forEach(callback)`	Ejecuta una función proporcionada para cada elemento del array.
`map(callback)`	Genera un nuevo array aplicando una función a cada elemento del array.
`filter(callback)`	Crea un nuevo array que contiene todos los elementos, para los que la función proporcionada devuelve `true`.

Métodos Date, String, Number, BigInt

- Date: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
- String: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
- Number: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number
- BigInt: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt

Tuple

- // Declaración de tublas
- let x: [string, number];

- // Inicialización correcta
- x = ["hello", 10]; // OK

- // Inicialización incorrecta
- x = [10, "hello"]; // Error

Tuple

- console.log(x[0].substr(1)); // OK
- console.log(x[1].substr(1)); // Error, Un tipo 'number' no tiene la función 'substr’

Object

- Se pueden:
  - Definir clases (más adelante)
  - Definir en formato json (indicando los tipos).
  - Desestructurar en variables.

- Definición:

```
const persona:
{nombre: string, edad: number, ciudad: string, profesion: string} =
 {
   nombre: 'Juan',
   edad: 30,
   ciudad: 'Madrid',
   profesion: 'Ingeniero'
}
```

Ejemplo sin tipos

```
    const persona2 = {
    nombre: 'Juan',
    edad: 30,
    ciudad: 'Madrid',
    profesion: 'Ingeniero'
}

    // Desestructuración de un objeto JSON
const {nombre, edad, ciudad, profesion} = persona

    console.log(`Nombre: ${nombre}`)
console.log(`Edad: ${edad}`)
console.log(`Ciudad: ${ciudad}`)
console.log(`Profesión: ${profesion}`)
```

Function

- En TypeScript, el tipo Function es un tipo general que se utiliza para representar cualquier función.

- **Uso básico de la función:**

```
let miFuncion: Function;
miFuncion = () => console.log("Hola"); // Aceptado
miFuncion = (a: number, b: number) => a + b; // También aceptado
```

Function

- **Funciones con tipos específicos:**
- // Función con parámetros y tipo de retorno definidos

```
const suma: (a: number, b: number) => number = (a, b) => a + b;
```

- **Declarar funciones con tipo reutilizables:**
- // Definición usando un tipo

```
type Operacion = (a: number, b: number) => number;

const multiplicar: Operacion = (a, b) => a * b;
```

Function

- **Cuando no hay retorno utilizar void:**

```
const mostrarMensaje: (mensaje: string) => void = (mensaje) => {
  console.log(mensaje);
};
```

- **Parámetros opcionales y predeterminados:**
- **// Parámetro opcional**

```
const saludar: (nombre?: string) => void = (nombre) => {
  console.log(`Hola, ${nombre || "invitado"}`);
};
```

- **// Parámetro con valor por defecto**

```
const potencia: (base: number, exponente?: number) => number = (base, exponente = 2) => {
  return base ** exponente;
};
```

Function

- **Funciones que no terminan** (**never**)

- El tipo never se usa para funciones que nunca devuelven (por ejemplo, lanzan errores o entran en bucles infinitos):

```
const lanzarError: (mensaje: string) => never = (mensaje) => {
  throw new Error(mensaje);
};
```

Function

- Parámetro ...rest
- El parámetro rest en TypeScript permite que una función acepte un **número indefinido de argumentos** como un **array**.
- Es útil cuando no se sabe de antemano cuántos valores serán pasados a la función.

```
function sumarNumeros(...numeros: number[]): number {
  return numeros.reduce((acumulador, numero) => acumulador + numero, 0);
}

// Usar la función
const resultado = sumarNumeros(1, 2, 3, 4, 5);
console.log(resultado); // 15
```

Function

- Otro ejemplo:

```
function describirGrupo(grupo: string, ...miembros: string[]): string {
  return `El grupo ${grupo} tiene los siguientes miembros: ${miembros.join(", ")}`;
}

// Usar la función
const descripcion = describirGrupo("Desarrolladores", "Antonio", "María", "Luis");
console.log(descripcion);
// "El grupo Desarrolladores tiene los siguientes miembros: Antonio, María, Luis"
```