Módulos en TS

Antonio Espín Herranz

Contenidos

- El ámbito global. Problemas
- El patrón módulo con funciones autoejecutables
  - Diferencia entre las closures y las funciones IIFE
- AMD y CommonJs
- Gestión de módulos con import / export

El ámbito global. Problemas

- Una variable definida fuera de cualquier bloque de código tiene alcance global.

- El alcance global permite que sean accesibles desde cualquier parte del programa.

- Problemas si tenemos ya ese nombre definido.
  - Se llaman colisiones y el compilador de TS las detecta.
  - Hay distintas estrategias para evilarlo.

El patrón módulo

- Se puede implementar mediante una **función autoejecutable** (IIFE: expresión de función invocada inmediatamente).
- Es una función que se invoca nada más definirla.
- Puede tener declaraciones privadas.
- El modulo se puede almacenar a través de una variable

Diferencia entre las closures y las funciones IIFE

Aspecto	Closure	IIFE
Ejecución	Se ejecuta cuando se llama explícitamente.	Se ejecuta inmediatamente al definirse.
Propósito	Mantener un estado o "recordar" variables del entorno.	Encapsular código y evitar la contaminación del espacio global.
Uso común	Crear funciones personalizadas o módulos con estado privado.	Inicializar variables o ejecutar código una sola vez.

Closures

- Definición: Una closure es una función que "recuerda" el entorno en el que fue creada, incluso después de que ese entorno haya dejado de existir.

- Propósito: Se utiliza para mantener un estado privado o para crear funciones personalizadas que dependen de un contexto específico.

```
function crearContador() {
    let contador = 0;
    return function() {
        contador++;
        return contador;
    };
}

const contador = crearContador();
console.log(contador()); // 1
console.log(contador()); // 2
```

Funciones IIFE

- Definición: Una IIFE es una función que se ejecuta inmediatamente después de ser definida, gracias a los paréntesis () al final de su declaración.

- Propósito: Se utiliza para encapsular código y evitar contaminar el espacio global, creando un ámbito privado temporal.

```
(function() {
    const mensaje = "Hola desde una IIFE";
    console.log(mensaje);
})();
```

- Se ejecuta inmediatamente.
- El valor de mensaje no es accesible desde fuera.

AMD y CommonJs

- El aumento de código JS impulsó la necesidad de tener código separado en varios fuentes.
- Las especificaciones AMD y CommonJs permiten crear y administrar módulos de forma eficiente.
- **Hoy en día se utilizan los ES Modules.**

Aspecto	AMD	CommonJS
Carga de módulos	Asíncrona	Síncrona
Entorno principal	Navegador	Servidor (Node.js)
Herramientas comunes	RequireJS	Node.js, Browserify

AMD

- AMD (Asynchronous Module Definition)

- Propósito: Diseñado para el desarrollo en el navegador, permite cargar módulos de forma asíncrona, lo que mejora el rendimiento al evitar bloqueos durante la carga de scripts.

- Sintaxis: Utiliza la función define para declarar módulos y sus dependencias.

Gestión de módulos con **import / export**

- A partir de **ECMAScript 2015** se estandarizó la carga de módulos.

- Se implementó la etiqueta:
  - **<script type="module">** para los navegadores.

- Se incluye las palabras:
  - **export**: permite exportar elementos.
  - **import**: permite importar elementos de un módulo.

- Cuando se incluyen una de estas dos palabras clave el archivo se convierte en un módulo.

- El estándar ECMAStript 2015 permite la carga asíncrona y síncrona de los módulos.

Gestión de módulos con **import / export**

- Los módulos deben importarse desde cualquier bloque de código.

- Se pueden importar dinámicamente con la función **import()**

- Un módulo puede importar varios otros y puede exportar varios elementos.
  - Se recomienda colocar las importaciones al principio del módulo.
  - La importación requiere una ruta relativa.

Ejemplo

salariedModule.ts > salaried

```ts
const salary = 20_000;

export const salaried = {
    getSalary: () => {
        return salary
    }
}
```

employeeModule.ts > employee

```ts
import { salaried } from "./salariedModule";

export const employee = {
    personList: ["Pedro", "Juan"],
    getSalary: () => {
        return salaried.getSalary()
    }
}
```

managerModule.ts > manager

```ts
import { salaried } from "./salariedModule";

const bonus = 40_000;

export const manager = {
    personList: ["Pedro", "Juan", "Luis"],
    getSalary: () => {
        return salaried.getSalary() + bonus
    }
}
```

main.ts

```ts
import { employee } from "./employeeModule";
import { manager } from "./managerModule";

console.log(employee.getSalary())
console.log(manager.getSalary())
console.log(manager.personList)
console.log(employee.personList)
```

export

- La palabra clave **export** permite exportar declaraciones para que sean accesibles desde fuera del módulo.

- Se puede utilizar antes de una declaración:
  - **export** elementoDeclarado

- O se va **exportando** a medida que se van haciendo las declaraciones.
  - export declaración1
  - export declaración2

- O se **agrupan** todas las **declaraciones** al final:
  - declaracion1
  - declaracion2

  - **export** {declaracion1, declaracion2 };

export

- Se pueden utilizar **alias** para cambiar los nombres de las declaraciones al exportar:
  - **export** {declaracion1 **as alias1**, declaracion2 **as alias2**}

- Dentro de módulo se puede definir una exportación predeterminada, pero no es una buena práctica.
  - export **default** declaracion

export

- Un modulo puede reexportar directamente las exportaciones de otro módulo.
  - export * from “path/otro_modulo”
  - export { elemento1, elemento2, … } from “path/otro_modulo”
  - export elemento1 as alias1 from “path/otro_modulo”

import

- La palabra clave **import**, permite cargar un módulo y utilizar los elementos que exporta.

- Cuando un módulo exporta varios elementos es posible elegir los que necesitamos y darles un **alias** si es necesario.

- import { declaracion1, declaracion2, ... } from “path/modulo”

- import { declaracion1 as alias1, declaracion2 as alias2, ... } from “path/modulo”

import

- Si un módulo exporta varios elementos se pueden importar varias declaraciones y agruparlos en un espacio de nombres.

- import * as alias from “path/otro_modulo”

- Ejemplo:
  - import * as util from “./util_modulo”

  - util.funcion1()
  - util.funcion2()