TypeScript - Modules

In this section, we will learn about Modules in TypeScript.

The TypeScript code we write is in the global scope by default. If we have multiple files in a project, the variables, functions, etc. written in one file are accessible in all the other files.

For example, consider the following TypeScript files: file1.ts and file2.ts

file1.ts

 Copy

var greeting : string = "Hello World!";

file2.ts

 Copy

console.log(greeting); //Prints Hello World!

greeting = "Hello TypeScript"; // allowed

The above variable greeting, declared in file1.ts is accessible in file2.ts as well. Not only it is accessible but also it is open to modifications. Anybody can easily override variables declared in the global scope without even knowing they are doing so! This is a dangerous space as it can lead to conflicts/errors in the code.

TypeScript provides modules and namespaces in order to prevent the default global scope of the code and also to organize and maintain a large code base.

Modules are a way to create a local scope in the file. So, all variables, classes, functions, etc. that are declared in a module are not accessible outside the module. A module can be created using the keyword export and a module can be used in another module using the keyword import.

In TypeScript, files containing a top-level export or import are considered modules. For example, we can make the above files as modules as below.

file1.ts

 Copy

export var greeting : string = "Hello World!";

file2.ts

 Copy

console.log(greeting); //Error: cannot find 'greeting'

greeting = "Hello TypeScript";

In file1.ts, we used the keyword export before the variable. Now, accessing a variable in file2.ts will give an error. This is because greeting is no longer in the global scope. In order to access greeting in file2.ts, we must import the file1 module into file2 using the import keyword.

Let's learn export and import in detail.

Export

A module can be defined in a separate .ts file which can contain functions, variables, interfaces and classes. Use the prefix export with all the definitions you want to include in a module and want to access from other modules.

Employee.ts

 Copy

export let age : number = 20;

export class Employee {

empCode: number;

empName: string;

constructor(name: string, code: number) {

this.empName = name;

this.empCode = code;

}

displayEmployee() {

console.log ("Employee Code: " + this.empCode + ", Employee Name: " + this.empName );

}

}

let companyName:string = "XYZ";

In the above example, Employee.ts is a module which contains two variables and a class definition. The age variable and the Employee class are prefixed with the export keyword, whereas companyName variable is not. Thus, Employee.ts is a module which exports the age variable and the Employee class to be used in other modules by importing the Employee module using the import keyword. The companyName variable cannot be accessed outside this Employee module, as it is not exported.

Import

A module can be used in another module using an import statement.

Syntax:

Import { export name } from "file path without extension"

Let's see different ways of importing a module export.

Importing a Single export from a Module:

We exported a variable and a class in the Employee.ts. However, we can only import the export module which we are going to use. The following code only imports the Employee class from Employee.ts into another module in the EmployeeProcessor.ts file.

EmployeeProcessor.ts

 Copy

import { Employee } from "./Employee";

let empObj = new Employee("Steve Jobs", 1);

empObj.displayEmployee(); //Output: Employee Code: 1, Employee Name: Steve Jobs

Importing the Entire Module into a Variable

You can import all the exports in a module as shown below.

EmployeeProcessor.ts

 Copy

import \* as Emp from "./Employee"

console.log(Emp.age); // 20

let empObj = new Emp.Employee("Bill Gates" , 2);

empObj.displayEmployee(); //Output: Employee Code: 2, Employee Name: Bill Gates

In the above example, we import all the exports in Employee module in a single variable called Emp. So, we don't need to write an export statement for each individual module. In the above example, it will import age and Employee class into the Emp variable and can be accessed using Emp.age and Emp.Employee.

Renaming an Export from a Module:

You can change the name of an export as shown below.

EmployeeProcessor.ts

 Copy

import { Employee as Associate } from "./Employee"

let obj = new Associate("James Bond" , 3);

obj.displayEmployee();//Output: Employee Code: 3, Employee Name: James Bond

In the above example, the name of Employee export class is changed to Associate using { employee as Associate }. This is useful in assigning a more meaningful name to an export, as per your need which increases the readability.