✓ 100 XP

Design considerations

5 minutes

You can use namespaces or modules for code organization, and both can contain code and declarations.

While namespaces are easy to use for simple implementations and do not depend on a module loader, modules offer some additional benefits that namespaces do not. Modules:

- Declare their dependencies.
- Provide better code reuse.
- Offer strong isolation.
- Hide the internal statements of the module definitions and show only the methods and parameters associated to the declared component.
- Provide better tooling support for bundling.
- Are recommended over namespaces for Node.js applications because modules are the default.
- Can resolve top-down JavaScript flow issues because a reference to an external method or class is instantiated only on method invocation.

And, starting with ECMAScript 2015, modules are native part of the language, and should be supported by all compliant engine implementations. So, for new projects, modules are recommended for code organization.

Note

It is not recommended to combine namespaces and modules in the same project.

The following table summarizes and compares modules and namespaces.

Module	Namespace
Use modules to organize code into separate files for logical grouping of functionality.	Use namespaces to organize code into separate files for logical grouping of

Module	Namespace
	functionality.
Modules execute in their local scope, not in the global scope.	Namespaces execute in their local scope, not in the global scope.
Modules are declarative; the relationships between modules are specified in terms of imports and exports at the file level.	Namespaces cannot declare their dependencies.
You define a module by using either the `export` or `import` keyword within a file. Any file containing a top-level import or export is considered a module.	You define a namespace by using the `namespace` keyword within a file. Namespace statements can be nested and span multiple files.
To expose individual module components outside the module, use the `export` keyword.	To expose individual namespace components outside of the namespace, use the `export` keyword.
To use a component from one module in another module, use the `import` keyword.	To use a component from one namespace in another TypeScript file, add a `reference` statement using the triple-slash (`///`) syntax.
To compile a module and all its dependent files, use the `tsc --module` command.	To compile TypeScript files containing namespaces and all their dependent files into individual JavaScript files, use the `tsc` command.
It is not possible to have multi-file modules compiled to a single module.	To compile all TypeScript files containing namespaces into a single JavaScript file, use the `tsc --outFile` command.
Modules import another module by using a module loader API. You specify the API when you compile the module.	Namespaces are loaded by specifying the **.js** file names (in order) using the `<script>` tag in the HTML page.
In modules, you can re-export the components either using their original name or rename it.	In namespaces, you cannot re-export components or rename them.

Next unit: Lab - Organize code with namespaces

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆