JS JavaScript TUTORIAL

JavaScript Import

Summary: in this tutorial, you will learn how to use the JavaScript import keyword to import variables, functions, and classes from another module.

Introduction to JavaScript import keyword

ES6 modules allow you to structure JavaScript code into modules and share values (variables, functions, classes, etc.) between them.

To import values from a module, you use the `import` keyword. Also, you need to load the JavaScript source file as a module. In HTML , you can do it by specifying the `type="module"` in the script tag:

```
<script type="module" src="app.js"></script>
```

We'll take a simple example to illustrate how to use the `import` keyword.

Suppose we have a project with the following structure:

```
├── index.html
└── js
    ├── app.js
    └── greeting.js
```

In this project, the `index.html` loads the `app.js` as a module:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>JavaScript import</title>
    </head>
    <body>
        <script src="js/app.js" type="module"></script>
    </body>
</html>
```

The `greeting.js` module has a function `sayHi()` function that display a message. It exports the `sayHi()` function as a default export:

```
export default function sayHi() {
  alert('Hi');
}
```

The `app.js` will import the `sayHi()` method from the `greeting.js` module and execute it.

Import a default export

If a module uses a default export, you can import the default export from that module using the following syntax:

```
import name from 'module.js';
```

For example, you can import the `sayHi()` function from the `greeting.js` module to the `app.js` module as follows:

```
import sayHi from './greeting.js';

sayHi();
```

If you open the `index.html` in the web browser, you'll see an alert message.

Because the `sayHi()` function is a default export, you can assign it any names you want when you import it.

For example, you can import the `sayHi` function from the `greeting.js` module and set its name to `displayGreeting()` as follows:

```
import displayGreeting from './greeting.js';

displayGreeting();
```

Import named exports

Unlike importing a default export, you need to specify the exact name of the named exports when you import them into a module. In addition, you need to place the named exports inside a pair of curly braces.

Here's the syntax for importing named exports:

```
import { namedExport1, namedExport2} from 'module.js';
```

For example, we can modify the `greeting.js` module that contains two named exports:

```
export function sayHi() {
  alert('Hi');
}

export function sayBye() {
  alert('Bye');
}
```

To import the `sayHi()` and `sayBye()` functions to the `app.js` module, you use the following code:

```
import { sayHi, sayBye } from './greeting.js';
```

You can also call these functions:

```
import { sayHi, sayBye } from './greeting.js';

sayHi();
sayBye();
```

If you open the `index.html` in the web browser, you'll see two alerts created by these two functions.

Namespace import

A module namespace object is a static object that includes all exports from a module. It is a static object that JavaScript creates when evaluating the module.

To access the module namespace object, you use the following syntax:

```
import * as name from 'module.js';
```

In this syntax, the `name` is the module namespace object that includes all exports from the `module.js` module as the properties.

For example, if the `modules.js` has the `myVariable`, `myFunction`, and `myClass` named exports, you can access them via the `name` object like this:

```
name.myVariable
name.myFunction
name.myClass
```

To illustrate how the namespace import works, we can change the `app.js` to use the namespace import as follows:

```
import * as greeting from './greeting.js';

greeting.sayHi();
greeting.sayBye();
```

The following specifies the `greeting` as the module namespace object:

```
import * as greeting from './greeting.js';
```

Once we have the module namespace object, we can call the `sayHi()` and `sayBye()` functions:

```
greeting.sayHi();
greeting.sayBye();
```

If the importing module has a default export, you can access it via the `default` keyword:

```
name.default
```

For example, we can change the `sayHi` function to a default export in the `greeting.js` module:

```
export default function sayHi() {
  alert('Hi');
}

export function sayBye() {
  alert('Bye');
}
```

And import the exports from the `greeting.js` module into the `app.js` module using the namespace import:

```
import * as greeting from './greeting.js';

greeting.default(); // sayHi()
greeting.sayBye();
```

In this case, we can call the `sayHi()` funtion via the `default()` as shown above. Note that you cannot access the `sayHi()` function using its name like this:

```
greeting.sayHi();
```

If you do so, you'll get the following error:

```
Uncaught TypeError: greeting.sayHi is not a function
```

Renaming a named export

When you import a named export, you can assign it a new name. It is useful when you import the function with the same name from different modules:

```
import { name as name1 } from "module1.js";
import { name as name2 } from "module2.js";
```

For example, the following illustrates how to rename the `sayHi()` and `sayBye()` functions to `hi()` and `bye()` when importing to the `app.js` module:

```
import { sayHi as hi, sayBye as bye } from './greeting.js';


hi();
bye();
```

Summary

- Use JavaScript `import` keyword to import variables, functions, and classes into a module.