JAVASCRIPT

ES6 Modules

Sreekanth M. E.

Freelance Trainer & Consultant

http://www.SreekanthME.com

In the past, modules were implemented via 3rd party libraries (like require.js).

ES6 is the first time that JavaScript has built-in modules.

ES6 modules are stored in files.

There is exactly one module per file and one file per module.

Things exported by one module can be imported by another module.

There are two ways of exporting things from a module.

- Multiple named exports
- Single default export

MULTIPLE NAMED EXPORTS

```
//------ lib.js ------
export function square(x) {
    return x * x;
}
export function cube(x) {
    return x * x * x;
}
```

```
//------ main1.js ------
import { square, cube } from ‘lib.js';
console.log(square(11)); // 121
console.log(cube(4)); // 64
```

```
//------ main2.js ------
import * as lib from ‘lib.js';
console.log(lib.square(11)); // 121
console.log(lib.cube(4)); // 64
```

MULTIPLE NAMED EXPORTS

Destructuring assignment syntax can be used to unpack things from the imported module.

```
import { square, cube } from 'lib';
```

MULTIPLE NAMED EXPORTS

The keyword as can be used to create an alias for the thing being imported.

When as is used with the wildcard *, the alias acts as a namespace for the imported module.

SINGLE DEFAULT EXPORT

```
//------ myFunc.js ------
export default function () {
    ...
}
//------ MyClass.js ------
export default class { ··· }
```

```
//------ main.js ------
import myFunc from 'myFunc.js’;
import MyClass from 'MyClass.js';

myFunc();
const inst = new MyClass();
```

SINGLE DEFAULT EXPORT

When importing a thing that was exported through a single default export, simply use a module name without any Destructuring Assignment syntax.

Modules are different from Scripts.

When a module is included in a HTML file, the script tag’s type attribute must be set to the value module.

Top-level variables are global in scripts while they are local in a module.

Value of this at top level is window in scripts while it is undefined in a module.

A module is executed once it is loaded.

So are scripts.

Relative or absolute paths can be used while importing.

The file extension .js can usually be omitted in the import statement.

Modules are singletons. Even if a module is imported multiple times, only a single instance of it exists.

Module imports are hoisted (internally moved to the beginning of the current scope).

Therefore, it doesn’t matter where import statement is written.

As of now, browsers don't support ES6 modules.

A poly-fill can be used to make modules work in browser. One such poly-fill is browser-es-module-loader which internally uses babel:

https://github.com/ModuleLoader/browser-es-module-loader

END OF CHAPTER

APPENDIX