8th February 2020

Dynamic imports

Export and import statements that we covered in previous chapters are called “static”. The syntax is very simple and strict.

First, we can’t dynamically generate any parameters of `import`.

The module path must be a primitive string, can’t be a function call. This won’t work:

```
import ... from getModuleName(); // Error, only from "string" is allowed
```

Second, we can’t import conditionally or at run-time:

```
if(...) {
  import ...; // Error, not allowed!
}

{
  import ...; // Error, we can't put import in any block
}
```

That’s because `import`/`export` aim to provide a backbone for the code structure. That’s a good thing, as code structure can be analyzed, modules can be gathered and bundled into one file by special tools, unused exports can be removed (“tree-shaken”). That’s possible only because the structure of imports/exports is simple and fixed.

But how can we import a module dynamically, on-demand?

The import() expression

The `import(module)` expression loads the module and returns a promise that resolves into a module object that contains all its exports. It can be called from any place in the code.

We can use it dynamically in any place of the code, for instance:

```
let modulePath = prompt("Which module to load?");

import(modulePath)
  .then(obj => <module object>)
  .catch(err => <loading error, e.g. if no such module>)
```

Or, we could use `let module = await import(modulePath)` if inside an async function.

For instance, if we have the following module `say.js`:

```
// 📁 say.js
export function hi() {
  alert(`Hello`);
}

export function bye() {
  alert(`Bye`);
}
```

…Then dynamic import can be like this:

```
let {hi, bye} = await import('./say.js');

hi();
bye();
```

Or, if `say.js` has the default export:

```
// 📁 say.js
export default function() {
  alert("Module loaded (export default)!");
}
```

…Then, in order to access it, we can use `default` property of the module object:

```
let obj = await import('./say.js');
let say = obj.default;
// or, in one line: let {default: say} = await import('./say.js');

say();
```

Here's the full example:

Result say.js index.html

```
<!doctype html>
<script>
  async function load() {
    let say = await import('./say.js');
    say.hi(); // Hello!
    say.bye(); // Bye!
    say.default(); // Module loaded (export default)!
  }
</script>
<button onclick="load()">Click me</button>
```

Please note:

Dynamic imports work in regular scripts, they don't require `script type="module"` .

Please note:

Although `import()` looks like a function call, it's a special syntax that just happens to use parentheses (similar to `super()`).

So we can't copy `import` to a variable or use `call/apply` with it. It's not a function.

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)