ES6 PPT1606202: ES6 模块 module

Thursday, November 5, 2015 10:15 AM

ES6 模块 Module：

1. 使用 export 关键字定义/导出 ES6 模块：export var / export function / export {,} / export {as,,as,,}

```
// myES6Export.js
export var elByVar = '我是变量声明式导出的元素'; // 直接导出声明的变量
export function fnByFunction() { console.log('我是函数声明式导出的函数元素'); }; // 直接导出声明的函数
var el1ByBrackets = '我是第一个通过花括号导出的元素', el2ByBrackets = {text: '我是第二个通过花括号导出的元素'};
function fnByBrackets() { console.log('我是通过花括号导出的函数元素'); }
export {el1ByBrackets, el2ByBrackets, fnByBrackets}; // 导出花括号内的所有元素
export {fnByBrackets as refToFnByBrackets}; // 重命名花括号内导出的元素
export '该元素导出方式有错'; // 报错，不能直接导出值
var elDirect = {text: '该元素导出方式也有错'};
export elDirect; // 报错，不能直接导出表达式值
(function() { export var elInFunc = {text: '该元素导出方式依然有错'}; })(); // 报错，export 必须在顶层作用域下调用
```

2. 使用 import 关键字加载/引入 ES6 模块：import / import {,} / import {as,,as,,} / import * as

```
// myES6Import.js
import 'noExportModule.js'; // 仅执行 ES6 模块代码而不引入任何元素
import { elByVar } from 'myES6Export.js'; // 引入一个元素
import { fnByFunction, el1ByBrackets, el2ByBrackets, fnByBrackets, refToFnByBrackets } from 'myES6Export.js'; // 引入多个元素
console.log(fnByBrackets === refToFnByBrackets); // true -> 证明重命名后导出的是同一个元素
import { el2ByBrackets as myEl2ByBrackets } from 'myES6Export.js'; // 引入一个元素并将其重命名使用
console.log(el2ByBrackets === myEl2ByBrackets); // true -> 证明重命名后引入的是同一个元素
import * as myModuleObj from 'myES6Export.js'; // 将所有元素一起打包作为属性输出到指定的对象下
console.log(myModuleObj.elByVar, myModuleObj.el1ByBrackets); // 我是变量声明式导出的元素, 我是第一个通过花括号导出的元素
```

3. 默认 ES6 模块元素：
   a. 本质上，ES6 默认模块元素就是导出一个名为 default 的元素，然后允许你引入时为它取任意名字
   b. 导出 ES6 模块默认元素： export default myExpression / export {myExpression as default}

```
// myES6ExportDefault.js
export default function myFunc1() { // 正确：使用 default 关键字导出默认元素，但因为后面还有其他默认元素导出，该元素最终未被导出
 console.log('我是模块默认导出元素，但因为稍后还导出了一些默认元素所以最终我不会被导出');
}
export default var myVarError = '该元素导出方式不正确'; // 报错：不能通过变量声明式导出默认元素
var myVarOK = '该元素导出方式正确';
export default myVarOK; // 正确：default 关键字后直接相接表达式，但并非最后获取结果的元素，因此最终未被导出
export { myVarOK as default }; // 正确：导出一个名为 default 的元素，但并非最后获取结果的元素，因此最终未被导出
var myFuncExport = function fnByVar() {
 console.log('我是模块最终导出的默认元素');
};
export default myFuncExport; // 正确：这是最后一个获取结果并输出的默认元素，因此被成功导出
export default function myFunc2() { // 正确：由于提升执行而非最后获取结果的元素，因此最终未被导出
 console.log('我由于提升定义执行所以最终默认导出的不是我');
}
export function nonDefaultFunc() { console.log('我不是默认导出的元素'); }
```

   c. 引入 ES6 模块默认元素： import myElement / import {default as myElement}

```
// myES6ImportDefault.js
import myDefaultFunc from './myES6ExportDefault.js'; // 引入默认元素，import 关键字后面不使用花括号
import { default as refToMyDefaultFunc } from './myES6ExportDefault.js'; // 引入默认元素，import 关键字后面使用花括号和 default as
import { nonDefaultFunc } from './myES6ExportDefault.js';
myDefaultFunc(); // 我是模块默认导出的元素
console.log(myDefaultFunc === refToMyDefaultFunc); // true -> 证明重命名后引入的仍是同一个默认元素
nonDefaultFunc(); // 我不是默认导出的
```

4. 导出引入的 ES6 模块

```
// myES6DirectExport.js
export { fnByFunction as fnExport } from './myES6Export.js'; // 将 fnByFunction 重命名为 fnExport 导出
export * from './myES6Export.js'; // 导出从 myES6Export.js 引入的除默认元素的所有元素
```

```
export default fnByFunction from './myES6Export.js'; // 将 fnByFunction 作为默认元素导出
```

5. 浏览器中引用 ES6 模块语法

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

6. ES6 模块的循环引用

```
// myInnerExport.js 该模块不被程序主脚本直接引用
import { myMain } from './myMainExport.js';
console.log('我最先被显示'); // "我最先被显示", -> 此行为第1行输出
console.log(myMain); // undefined, -> 此行为第2行输出
export let myInner = '我是 myInnerExport 模块输出的元素';
// myMainExport.js 该模块将被程序主脚本直接引用
import { myInner } from './myInnerExport.js';
console.log('我随后被显示'); // "我随后被显示", 此行为第3行输出
console.log(myInner); // "我是 myInnerExport 模块输出的元素", -> 此行为第4行输出
export let myMain = '我是 myMainExport 模块输出的元素';
```

7. 不同于 CommonJS 模块输出一个元素值的拷贝，ES6 模块输出的是元素的引用
   a. 在 CommonJS 模块输出后其元素值不会随模块内后续运算再次更新

```
// myCommonJSExport.js
console.log('commonJS 模块导出');
var counter = 3;
module.exports = {
  counter: counter,
  incCounter: function() { counter++ },
};
// myCommonJSImport.js
console.log('commonJS 模块尚未加载');
var mod = require('./myCommonJSExport.js'); // 执行 myCommonJSExport.js 代码并将输出的元素值缓存于 mod 指向的引用
console.log(mod.counter);  // 3
mod.incCounter();
console.log(mod.counter); // 3
var mod2 = require('./myCommonJSExport.js'); // 不再次执行 myCommonJSExport.js 代码，而直接从早前执行结果中获取元素缓存值
console.log(mod === mod2); // true
```

   b. 引入 ES6 模块命令有提升作用，模块输出后其元素值仍然可能随模块内后续运算再次更新

```
// myESExport.js
console.log('ES6 模块导出');
export let counter = 3;
export function incCounter() {
  counter++;
}
// myESImport.js
console.log('引入 ES6 模块已经被提升到此行前并执行');
import { counter, incCounter } from './myESExport.js'; // 此行被提升至首行，执行 myESExport.js 代码并从结果中获取输出元素的引用
console.log(counter); // 3
incCounter();
console.log(counter); // 4
import { incCounter as incCounter2 } from './myESExport.js'; // 不再次执行 myESExport.js 代码，而直接从早前执行结果中获取同一输出元素的引用
console.log(incCounter === incCounter2); // true
```