10-第十章 ECMAScript6 Module 模块

`ECMAscript 6`

一、概念

历史上，JavaScript 一直没有模块（`module`）体系，无法将一个大程序拆分成互相依赖的小文件，再用简单的方法拼装起来。其他语言都有这项功能，比如 Ruby 的require、Python 的import，甚至就连 CSS 都有`@import`，但是 JavaScript 任何这方面的支持都没有，这对开发大型的、复杂的项目形成了巨大障碍。

在 ES6 之前，社区制定了一些模块加载方案，最主要的有`CommonJS`和 AMD 两种。前者用于服务器，后者用于浏览器。ES6 在语言标准的层面上，实现了模块功能，而且实现得相当简单，完全可以取代`CommonJS`和`AMD`规范，成为`浏览器`和`服务器`通用的模块解决方案。

---

node的`CommonJS`模块`module.exports`等于`exports`

app.js文件 exports导出模块

```
exports.hello= function hello(val){
    console.log(val);
    console.log(exports);
}
exports.world= function world(val){
    console.log(val);
    console.log(exports);
}
exports.ending= function ending(val){
    console.log(val);

}
```

index.js文件导入app.js模块

```
var expor = require('./app.js');
function index(){
    console.log('index');
}
index();
expor.hello('hello');
expor.world('world');
expor.ending('ending');
```

等同于

```
let {hello,world,ending} = require('./app.js');
hello('hello');
world('world');
ending('ending');
```

以上代码需在node环境中，
ES6 模块不是对象，而是通过`export`命令显式指定`输出`的代码，再通过`import`命令`输入`。

二、export 导出模块变量

> 模块功能主要由两个命令构成：`export`和`import`。export命令用于规定模块的对外接口，import命令用于输入其他模块提供的功能。

一个模块就是一个独立的文件。该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用export关键字输出该变量

---

(1) 直接导出

app.js模块`export`导出

```
export var a = 'hello world~~';//变量导出
  export function hello(val){//函数导出
    console.log(val)
  }

```

index.js 模块 `import {...} from` 导入

```
import {a, hello }  from './app.js';
console.log(a);//hello world
hello('hello');//hello
```

注意：以下情况 `报错`

```
 var a = 'hello';
  function hello(val){
    console.log(val);
  }

  export a; //Error: Module parse failed
  export hello;//Error: Module parse failed
  export 1;//Error: Module parse failed
```

---

(2) 对象导出

```
var a = 'hello world';
   function hello(val){
    console.log(val);
   }

   export {a,hello};
```

使用大括号指定所要输出的一组变量。它与前一种写法（直接放置在var语句前）是等价的，但是应该 `优先考虑{}` 使用这种写法。因为这样就可以在脚本尾部，一眼看清楚输出了哪些变量

---

(3) as 重命名

> export输出的变量就是本来的名字，但是可以使用as关键字重命名。

app.js模块 导出

```
function foo(){ console.log('hello'); }
function bar(){ console.log('world'); }
function baz(){ console.log('ending');}
export {foo as v1,bar as v2, baz as v3,foo as v4};
```

上面代码使用as关键字，重命名了函数foo、bar、baz的对外接口。foo重命名了`2`次

---

index.js模块 导入

```
import {v1,v2,v3,v4 }  from './app.js';
v1();//hello
v2();//world
v3();//ending
v4();//hello
```

v1 和v4是一致的

---

export命令可以出现在模块`顶层作用域`的任何位置。如果处于局部作用域内，就会报错，下一节的`import`命令也是如此

```
function foo(){ console.log('hello'); }

  function fn(){
   export {foo as v1}; // Error: Module parse failed

  }
```

三、import 入模块

> 使用export命令定义了模块的对外接口以后，其他 JS 文件就可以通过 `import` 命令加载这个模块。

(1) import {...} from url

```
1.      import {v1,v2,v3,v4 }  from './app.js';
2.      v1();//hello
3.      v2();//world
4.      v3();//ending
5.      v4();//hello
```

接受的名称必须和port导出的名称一致

---

(2) as 变量重命名

> 如果想为输入的变量重新取一个名字，import命令要使用as关键字，将输入的变量重命名

```
1. import {v1 as hello,v2 as  world }  from './app.js';
2. hello();//hello
3. world();//world
```

---

(3) import 变量提升

下面的代码不会报错，因为import的执行早于hello、world的调用。这种行为的本质是，import命令是`编译阶段`执行的，在代码运行之前。

```
1. hello();//hello
2. world();//world
3.
4. import {v1 as hello,v2 as  world }  from './app.js';
```

由于import是静态执行，所以`不能使用表达式和变量`，这些只有在运行时才能得到结果的语法结构。

```
// 报错
import {'v'+1}  from './app.js';//Error: Module parse failed
//无效
var str = 'hello';
import {str}  from './app.js';
hello();//ReferenceError

```

三、* 模块的整体加载

```
import * as modu from './app.js';
console.log(modu);//Object {__esModule: true}
console.log( modu.a );//hello
modu.hello('my good');//my good
```

四、export default 默认输出

> 从前面的例子可以看出，使用import命令的时候，用户需要知道所要加载的变量名或函数名，否则无法加载。但是，用户肯定希望快速上手，未必愿意阅读文档，去了解模块有哪些属性和方法。

就要用到export default命令，为模块指定默认输出。

app.js 模块

```
export default function hello(val){
        console.log(val);

    }
```

index.js模块

```
import hello from './app.js';
hello('index');//index
```

其他模块加载该模块时，import命令可以为该匿名函数指定`任意名字`。

```
export default function hello(val){
        console.log(val);

    }
 export default function world(val){
      console.log(val);

  }
```

export default命令用于指定模块的默认输出。显然，一个模块只能有一个默认输出，因此`export default命令只能使用一次`。所以，import命令后面才不用加大括号，因为只可能对应一个方法。

```
import def from './app.js';
//console.log(hello);//Object {}
def.hello('hello world');//hello world
```

导出对象

```
    var  a = 'a';
    function hello(val){ console.log(val) }
    export default {a,hello};
```

```
import def from './app.js';
//console.log(hello);//Object {}
def.hello('hello world');//hello world

```

不能采用`{}`对象接收 ，下面写法无效

```
import {a,hello}from './app.js';
```

其他情况

```
1.
2.      // export default var a = 'ok';//报错
3.      // var b = 'ok';
4.      // export default b;//ok
5.      // export default 'hello';//ok
6.      // export default function fn(){};//ok
7.      // function fn(){}
8.      // export default fn;//ok
```

五、export 与export default 同时使用

> 同时输入默认方法和其他接口，可以写成下面这样。
> app.js模块

```
1.  export var a = 'a';
2.      function fn(){
3.          console.log('ok')
4.      }
5.      export default {a,fn};
```

index.js模块

```
1.  import defaut,{a}from './app.js';
2.      console.log(a);
3.      console.log(defaut);
```

`*` 接收 默认和其他接口

```
1.  import * as duf from './app.js';
2.
3.      console.log(duf.default);//{a: "a"}
4.       console.log(duf.a);//a
```