

Node.js模块系统

Common.js(module.exports, require()) AMD CMD ES6(export, import, imports())

<http://www.runoob.com/nodejs/nodejs-module-system.html>

前言: ES6标准发布后, module成为标准, 标准的使用是以export指令导出接口, 以import引入模块, 但是在我们一贯的node模块中, 我们采用的是CommonJS规范, 使用require引入模块, 使用module.exports导出接口。

关键命令:

****exports** 导出模块 (exports, module.exports) 注意: 可以导出多个变量, exports是一个对象, 导出多个变量会在这个对象里面加入属性名和属性值得 如果后台在使用module.exports = 引用数据类型, 则引用改变了, 上面的失效了

****module.exports**导出模块

****require**引入模块 (node.js 默认后缀为 js)

Node.js 提供了 exports 和 require 两个对象, 其中 exports 是模块公开的接口, require 用于从外部获取一个模块的接口, 即所获取模块的 exports 对象。

1.定义: 模块是Node.js 应用程序的基本组成部分, 文件和模块是一一对应的。换言之, 一个 Node.js 文件就是一个模块, 这个文件可能是JavaScript 代码、JSON 或者编译过的C/C++ 扩展。

2.创建模块: 一个单独的JS文件

exports 导出模块

exports.a = 10;

module.exports = function() {};

3.引入模块

let 变量名 = require(模块的路径);

一般一个JS文件就是一个模块，一个JS文件，封装一个接口，给外面暴露出来使用

*** NODE的module遵循CommonJS规范，requirejs遵循AMD，seajs遵循CMD，虽各有不同，但总之还是希望保持较为统一的代码风格。

ES6模块

前言:

--->> ES6发布的module并没有直接采用CommonJS，甚至连require都没有采用，也就是说require仍然只是node的一个私有的全局方法，module.exports也只是node私有的一个全局变量属性，跟标准半毛钱关系都没有。

--->> ES6标准发布后，module成为标准，标准的使用是以export指令导出接口，以import引入模块，但是在我们一贯的node模块中，我们采用的是CommonJS规范，使用require引入模块，使用module.exports导出接口。

1. export导出模块接口

export 是一个对象，引用数据类型，给对象添加属性和方法，最后返回这个对象，所以import导入的时候，采用对象的结构复制去找里面的变量对象

export语法声明用于导出函数、对象、指定文件（或模块）的原始值。

-->> 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。下面是一个 JS 文件，里面使用export命令输出变量。

导出变量，函数，类，还可以使用as关键字起别名

```
// ==>> 导出方法一(对象依次加入属性和方法)
export let name = 'clh';
export let age = 23;
export let school = '石家庄建功科技';
```

```
// ==>> 导出方法二(推介, 在最后面看, 清晰)
```

```
let x = 10;
let y = 20;
let z = () => {
  let sum = 0;
  for (let i = 0; i < 10; i++) {
    sum += i;
  }
  return sum;
};
```

export {x, y, z}; (这其实就是export = {x, y, z}); 直接改变export的引用地址

```
import {name, age, school, x, y, z} from './a.js';
console.log(name, age, school);
let sum = z();
console.log(sum);
```

// ==>> 重命名了函数v1和v2的对外接口。重命名后, v2可以用不同的名字输出两次。

```
function v1() { ... }
function v2() { ... }
```

```
export {  
  v1 as streamV1, ==>> streamV1: V1  
  v2 as streamV2,  
  v2 as streamLatestVersion  
};
```

需要特别注意的是，`export`命令规定的是对外的接口，必须与模块内部的变量建立一一对应关系。

```
// 报错
```

```
export 1;
```

```
// 报错
```

```
var m = 1;
```

```
export m;
```

上面两种写法都会报错，因为没有提供对外的接口。第一种写法直接输出 1，第二种写法通过变量`m`，还是直接输出 1。`1`只是一个值，不是接口。正确的写法是下面这样

```
// 写法一
```

```
export var m = 1;
```

```
// 写法二
```

```
var m = 1;
```

```
export {m};
```

```
// 写法三
```

```
var n = 1;
```

```
export {n as m};
```

b.js

```
let fn = () => console.log('b-module-fn');

export {fn as clh1};
export {fn as clh2};
export {fn as clh3};
```

b.js

a.js

```
// ==> 导出方法一
export let name = 'clh';
export let age = 23;
export let school = '石家庄建功科技';

// ==> 导出方法二

let x = 10;
let y = 20;
let z = () => {
  let sum = 0;
  for (let i = 0; i < 10; i++) {
    sum += i;
  }
  return sum;
};

export {x, y, z};
```

deom1.html

```
<script type="module">
  import {name, age, school, x, y, z} from './a.js';
  import {clh1, clh2, clh3} from './b.js';

  console.log(name, age, school);
  let sum = z();
  console.log(sum);

  console.log(clh1());
</script>
```

2. import导入模块

*** 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。

说明：

上面代码的 `import` 命令，用于加载 `profile.js` 文件，并从中输入变量。 `import` 命令接受一对大括号，里面指定要从其他模块导入的变量

名。大括号里面的变量名，必须与被导入模块（`profile.js`）对外接口的名称相同。

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。

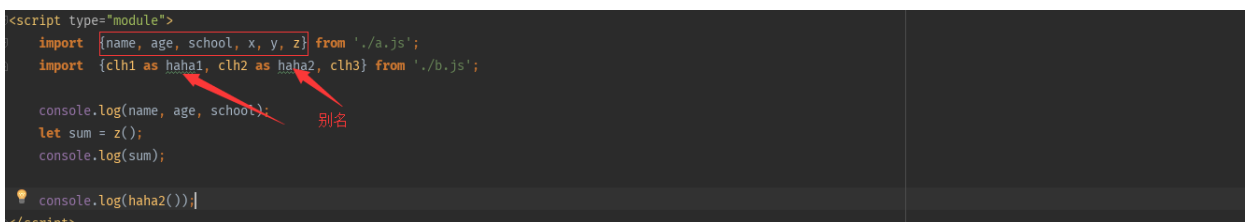
`import`的语法跟`require`不同，而且`import`必须放在文件的最开始，且前面不允许有其他逻辑代码，这和其他所有编程语言风格一致。

浏览器目前不支持这两个,采用Babel进行转换为ES5的

http://v.youku.com/v_show/id_XMTU5NDU4NzAxMg==.html (视频)

<https://babeljs.cn/docs/editors.html> Babel在线学习

`import`导入的变量名必须和 `export`导出的模块变量名保持一致，否则对象结构找不到，报错



```
<script type="module">
  import {name, age, school, x, y, z} from './a.js';
  import {clh1 as haha1, clh2 as haha2, clh3} from './b.js';

  console.log(name, age, school);
  let sum = z();
  console.log(sum);

  console.log(haha2());
</script>
```

细节知识:

1) 注意，`import`命令具有提升效果，会提升到整个模块的头部，首先执行。（在模块里面，在JS里面不会有这种效果）

2) **`import`导入的变量名必须和 `export`导出的模块变量名保持一致，否则对象结构找不到，报错**

3) 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。

4) 除了指定加载某个输出值，还可以使用整体加载，即用星号（`*`）指定一个对象，所有输出值都加载在这个对象上面。

5) export只能使用一次， 原因是：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。

6) export default导出的import导入不用加入{} 一个模块么

7) 在ES6模块中，无论你是否加入“use strict;”语句，默认情况下模块都是在严格模式下运行。

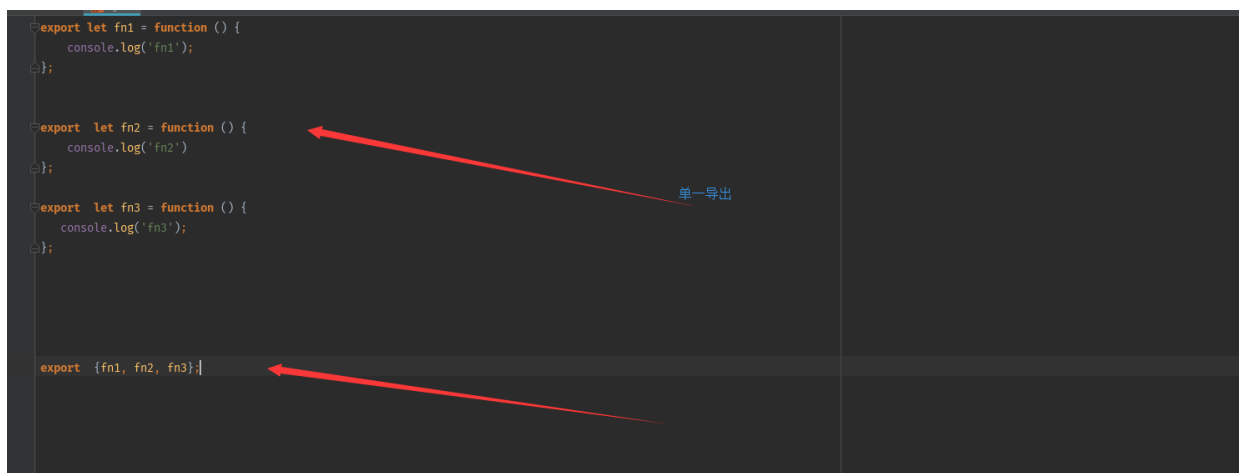
8) 在模块中你可以使用import和export关键字。

9) 你可以导出所有的最外层函数、类以及var、let或const声明的变量。类， 函数

10) ， 尤其需要注意this的限制。ES6 模块之中， 顶层的this指向undefined， 即不应该在顶层代码使用this。

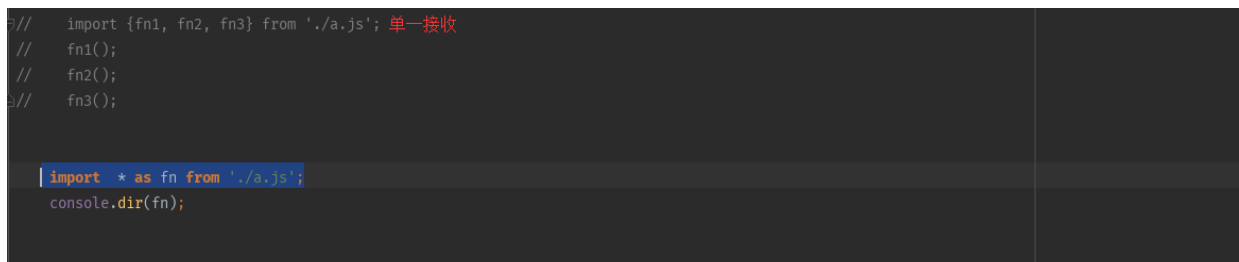
11) 在模块中， class和import有预解释一说， 但在 JS代码里面可没有哦（但是函数和类都是声明了， 不能直接执行函数和类）

单一导出和集体导出

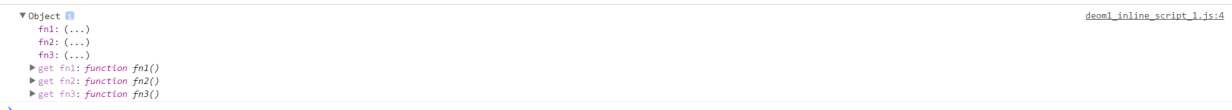


The diagram illustrates two types of exports in ES6 modules. On the left, three functions are defined and each is exported individually using the `export` keyword: `export let fn1 = function () { console.log('fn1'); };`, `export let fn2 = function () { console.log('fn2'); };`, and `export let fn3 = function () { console.log('fn3'); };`. A red arrow labeled "单一导出" (Single Export) points from this text to the first export statement. Below these, a collective export is shown: `export {fn1, fn2, fn3};`. A red arrow points from this statement to the text "单一导出和集体导出" (Single Export and Collective Export) at the top of the diagram.

单一变量对象接收和采用一个对象集体接收



The diagram illustrates two types of imports in ES6 modules. At the top, a single import is shown: `import {fn1, fn2, fn3} from './a.js';` followed by calls to `fn1();`, `fn2();`, and `fn3();`. A red arrow labeled "单一接收" (Single Receive) points from this text to the import statement. Below this, a collective import is shown: `import * as fn from './a.js';` followed by `console.dir(fn);`. A red arrow points from this statement to the text "单一变量对象接收和采用一个对象集体接收" (Single variable object receive and adopt one object collective receive) at the top of the diagram.



模块的整体加载(*)

原理：利用 `as` 关键字起个别名，（这时候这个别名是一个对象，但是这个对象是不可扩展的），导出的东西，都在这个别名上挂的

导入

```
import * as obj from './js/a.js';
console.log(typeof obj, obj);  'object' obj
obj.name = 'clh';    // ==>> Can't add property name,
object is not extensible
```

导出

```
export let a = function() { console.log('a')};
export class B {
    constructor(x = 1, y = 2) {
        Object.assign(this, {x, y});
    }
};
```

export default 命令

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。

为了给用户方便，让他们不用阅读文档就能加载模块，就要用到 `export default` 命令，为模块指定默认输出

1); `export default` 原理: (导出函数和类和对象)

`export default` 就是输出一个叫做 `default` 的变量或方法，然后系统允许你为它取任意名字。

模拟下：

`a.js`

```
let fn = () => console.log('fn');  
export {fn as default};
```

`deom1.html`

```
import {default as default} from './a.js' ==>>  
import fn from './a.js'
```

案例:

`export default val => console.log(val);` 匿名函数导出

`export default class A {` 类导出

`}`

`import _ from './a.js'`

`import $ from './a.js'`

// =====>> 案例

1) 导出对象

`export default {`

`a: 10,`

`b() {`

```
}  
}
```

2) 导出类

```
export default class {  
  
}
```

3) 导出函数

```
export default function() {  
  
}
```

对于这种默认的模式导出，我们直接使用import导入，直接指定一个变量去应用即可，不需要使用{}

```
import $ from './jQuery.js';
```

正是因为`export default`命令其实只是输出一个叫做`default`的变量，所以它后面不能跟变量声明语句。下面是错误的

```
export default var a = 10;
```

```
export default var b = function() {  
  
}
```

练习:

——对应接收:

```

export let fn = function () { // ==>单一导出
  console.log(123);
};

let a = 10;
let b = 20;
let fn1 = val => console.log(val);
export {a as a1, b as a2, fn1 as a3}; // ==>集体导出

export default val => console.log(val); // ==>导出一个匿名函数

```

```

<script type="module">
  import {fn} from './a.js';
  fn();

  import {a1, a2, a3} from './a.js';
  import {a1 as x, a2 as y, a3 as z} from './a.js';
  z();

  import fn from './a.js';
  fn('阿东');

```

模块的继承：

粗略地讲，当你通知JS引擎运行一个模块时，它一定会按照以下四个步骤执行下去：

1. 语法解析：阅读模块源代码，检查语法错误。
2. 加载：递归地加载所有被导入的模块。这也正是没被标准化的部分。
3. 连接：每遇到一个新加载的模块，为其创建作用域并将模块内声明的所有绑定填充到该作用域中，其中包括由其它模块导入的内容。
4. 如果你的代码中有 `import {cake} from "paleo"` 这样的语句，而此时“paleo”模块并没有导出任何“cake”，你就会触发一个错误。这实在是太糟糕了，你都快要运行模块中的代码了，都是cake惹的祸！
5. 运行时：最终，在每一个新加载的模块体内执行所有语句。此时，导入的过程就已经结束了，所以当执行到达有一行import声明的代码的时候.....什么都没发生！

面试题:

```
setTimeout(function () {  
  console.log(1);  
}, 0);
```

```
new Promise(function executor(resolve) {  
  console.log(2);  
  for (let i=0;i < 1000;i++) {  
    i = 9999 && resolve();  
  }  
  console.log(3);  
}).then(function () {  
  console.log(4);  
});
```

```
console.log(5); // ==>> 2 3 5 4 1
```

导出模块

```
export let a = function() {  
  alert(11)  
}  
  
export function fn() {  
  
}
```

```
export class BB {
```

```
        getX() {  
            alert('sb');  
        }  
    }  
}
```

```
export default function() {  
}
```

```
export default class {  
  
}
```

```
export {a as a1, b, c }
```