

ECMAScript 6 入门

作者：阮一峰

授权：署名-非商用许可证



目录

- 0.前言
- 1.ECMAScript 6简介
- 2.let 和 const 命令
- 3.变量的解构赋值
- 4.字符串的扩展
- 5.正则的扩展
- 6.数值的扩展
- 7.函数的扩展
- 8.数组的扩展
- 9.对象的扩展
- 10.对象的新增方法
- 11.Symbol
- 12.Set 和 Map 数据结构
- 13.Proxy
- 14.Reflect
- 15.Promise 对象
- 16.Iterator 和 for...of 循环
- 17.Generator 函数的语法
- 18.Generator 函数的异步应用
- 19.async 函数
- 20.Class 的基本语法
- 21.Class 的继承
- 22.Decorator
- 23.Module 的语法
- 24.Module 的加载实现
- 25.编程风格
- 26.读懂规格
- 27.ArrayBuffer
- 28.最新提案
- 29.参考链接

其他

- 源码
- 修订历史
- 反馈意见

Module 的语法

- 1.概述
- 2.严格模式
- 3.export 命令
- 4.import 命令
- 5.模块的整体加载
- 6.export default 命令
- 7.export 与 import 的复合写法
- 8.模块的继承

1. 概述

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。

ES6 模块的设计思想是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonJS 和 AMD 模块，都只能在运行时确定这些东西。比如，CommonJS 模块就是对象，输入时必须查找对象属性。

```
// CommonJS模块
let { stat, exists, readFile } = require('fs');

// 等同于
let _fs = require('fs');
let stat = _fs.stat;
let exists = _fs.exists;
let readFile = _fs.readFile;
```

上面代码的实质是整体加载 `fs` 模块（即加载 `fs` 的所有方法），生成一个对象（`_fs`），然后再从这个对象上面读取 3 个方法。这种加载称为“运行时加载”，因为只有运行时才能得到这个对象，导致完全没办法在编译时做“静态优化”。

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。

```
// ES6模块
import { stat, exists, readFile } from 'fs';
```

上面代码的实质是从 `fs` 模块加载 3 个方法，其他方法不加载。这种加载称为“编译时加载”或者静态加载，即 ES6 可以在编译时就完成模块加载，效率要比 CommonJS 模块的加载方式高。当然，这也导致了没法引用 ES6 模块本身，因为它不是对象。

由于 ES6 模块是编译时加载，使得静态分析成为可能。有了它，就能进一步拓宽 JavaScript 的语法，比如引入宏（macro）和类型检验（type system）这些只能靠静态分析实现的功能。

除了静态加载带来的各种好处，ES6 模块还有以下好处。

- 不再需要 UMD 模块格式了，将来服务器和浏览器都会支持 ES6 模块格式。目前，通过各种工具库，其实已经做到了这一点。
- 将来浏览器的新 API 就能用模块格式提供，不再必须做成全局变量或者 `navigator` 对象的属性。
- 不再需要对象作为命名空间（比如 `Math` 对象），未来这些功能可以通过模块提供。

本章介绍 ES6 模块的语法，下一章介绍如何在浏览器和 Node 之中，加载 ES6 模块。

2. 严格模式

ES6 的模块自动采用严格模式，不管你有没有在模块头

严格模式主要有以下限制。

- 变量必须声明后再使用
- 函数的参数不能有同名属性，否则报错
- 不能使用 `with` 语句
- 不能对只读属性赋值，否则报错
- 不能使用前缀 `0` 表示八进制数，否则报错
- 不能删除不可删除的属性，否则报错
- 不能删除变量 `delete prop`，会报错，只能删除属性 `delete global[prop]`
- `eval` 不会在它的外层作用域引入变量
- `eval` 和 `arguments` 不能被重新赋值
- `arguments` 不会自动反映函数参数的变化
- 不能使用 `arguments.callee`
- 不能使用 `arguments.caller`
- 禁止 `this` 指向全局对象
- 不能使用 `fn.caller` 和 `fn.arguments` 获取函数调用的堆栈
- 增加了保留字（比如 `protected`、`static` 和 `interface`）

上面这些限制，模块都必须遵守。由于严格模式是 ES5 引入的，不属于 ES6，所以请参阅相关 ES5 书籍，本书不再详细介绍了。

其中，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`。

3. export 命令

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。

```
// profile.js
export var firstName = 'Michael';
export var lastName = 'Jackson';
export var year = 1958;
```

上面代码是 `profile.js` 文件，保存了用户信息。ES6 将其视为一个模块，里面用 `export` 命令对外部输出了三个变量。

`export` 的写法，除了像上面这样，还有另外一种。

```
// profile.js
var firstName = 'Michael';
var lastName = 'Jackson';
var year = 1958;

export {firstName, lastName, year};
```

上面代码在 `export` 命令后面，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。

`export` 命令除了输出变量，还可以输出函数或类（`class`）。

```
export function multiply(x, y) {
  return x * y;
```

```
};
```

上面代码对外输出一个函数 `multiply`。

通常情况下，`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。

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

export {
  v1 as streamV1,
  v2 as streamV2,
  v2 as streamLatestVersion
};
```

上面代码使用 `as` 关键字，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。

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};
```

上面三种写法都是正确的，规定了对外的接口 `m`。其他脚本可以通过这个接口，取到值 `1`。它们的实质是，在接口名与模块内部变量之间，建立了一一对应的关系。

同样的，`function` 和 `class` 的输出，也必须遵守这样的写法。

```
// 报错
function f() {}
export f;

// 正确
export function f() {};

// 正确
function f() {}
export {f};
```

另外，`export` 语句输出的接口，与其对应的值是动态绑定关系，即通过该接口，可以取到模块内部实时的值。

```
export var foo = 'bar';
setTimeout(() => foo = 'baz', 500);
```

上面代码输出变量 `foo`，值为 `bar`，500 毫秒之后变成 `baz`。

这一点与 `CommonJS` 规范完全不同。`CommonJS` 模块输出的是值的缓存，不存在动态更新，详见下文《`Module` 的加载实现》一节。

最后，`export` 命令可以出现在模块的任何位置，只要处于模块顶层就可以。如果处于块级作用域内，就会报错，下一节的 `import` 命令也是如此。这是因为处于条件代码块之中，就没法做静态优化了，违背了 `ES6` 模块的设计初衷。

```
function foo() {  
  export default 'bar' // SyntaxError  
}  
foo()
```

上面代码中，`export` 语句放在函数之中，结果报错。

4. import 命令

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。

```
// main.js  
import {firstName, lastName, year} from './profile.js';  
  
function setName(element) {  
  element.textContent = firstName + ' ' + lastName;  
}
```

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 { lastName as surname } from './profile.js';
```

`import` 命令输入的变量都是只读的，因为它的本质是输入接口。也就是说，不允许在加载模块的脚本里面，改写接口。

```
import {a} from './xxx.js'  
  
a = {}; // Syntax Error : 'a' is read-only;
```

上面代码中，脚本加载了变量 `a`，对其重新赋值就会报错，因为 `a` 是一个只读的接口。但是，如果 `a` 是一个对象，改写 `a` 的属性是允许的。

```
import {a} from './xxx.js'  
  
a.foo = 'hello'; // 合法操作
```

上面代码中，`a` 的属性可以成功改写，并且其他模块也可以读到改写后的值。不过，这种写法很难查错，建议凡是输入的变量，都当作完全只读，轻易不要改变它的属性。

`import` 后面的 `from` 指定模块文件的位置，可以是相对路径，也可以是绝对路径，`.js` 后缀可以省略。如果只是模块名，不带有路径，那么必须有配置文件，告诉 `JavaScript` 引擎该模块的位置。

```
import {myMethod} from 'util';
```

上面代码中，`util` 是模块文件名，由于不带有路径，必须通过配置，告诉引擎怎么取到这个模块。

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。

```
foo();

import { foo } from 'my_module';
```

上面的代码不会报错，因为 `import` 的执行早于 `foo` 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。

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 { 'f' + 'oo' } from 'my_module';

// 报错
let module = 'my_module';
import { foo } from module;

// 报错
if (x === 1) {
  import { foo } from 'module1';
} else {
  import { foo } from 'module2';
}
```

上面三种写法都会报错，因为它们用到了表达式、变量和 `if` 结构。在静态分析阶段，这些语法都是没法得到值的。

最后，`import` 语句会执行所加载的模块，因此可以有下面的写法。

```
import 'lodash';
```

上面代码仅仅执行 `lodash` 模块，但是不输入任何值。

如果多次重复执行同一句 `import` 语句，那么只会执行一次，而不会执行多次。

```
import 'lodash';
import 'lodash';
```

上面代码加载了两次 `lodash`，但是只会执行一次。

```
import { foo } from 'my_module';
import { bar } from 'my_module';

// 等同于
import { foo, bar } from 'my_module';
```

上面代码中，虽然 `foo` 和 `bar` 在两个语句中加载，但是它们对应的是同一个 `my_module` 实例。也就是说，`import` 语句是 `Singleton` 模式。

目前阶段，通过 `Babel` 转码，`CommonJS` 模块的 `require` 命令和 `ES6` 模块的 `import` 命令，可以写在同一个模块里面，但是最好不要这样做。因为 `import` 在静态解析阶段执行，所以它是一个模块之中最早执行的。下面的代码可能不会得到预期结果。

```
require('core-js/modules/es6.symbol');
require('core-js/modules/es6.promise');
import React from 'React';
```

5. 模块的整体加载

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。

下面是一个 `circle.js` 文件，它输出两个方法 `area` 和 `circumference`。

```
// circle.js

export function area(radius) {
  return Math.PI * radius * radius;
}

export function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

现在，加载这个模块。

```
// main.js

import { area, circumference } from './circle';

console.log('圆面积: ' + area(4));
console.log('圆周长: ' + circumference(14));
```

上面写法是逐一指定要加载的方法，整体加载的写法如下。

```
import * as circle from './circle';

console.log('圆面积: ' + circle.area(4));
console.log('圆周长: ' + circle.circumference(14));
```

注意，模块整体加载所在的那个对象（上例是 `circle`），应该是可以静态分析的，所以不允许运行时改变。下面的写法都是不允许的。

```
import * as circle from './circle';

// 下面两行都是不允许的
circle.foo = 'hello';
circle.area = function () {};
```

6. export default 命令

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。

为了给用户提供方便，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。

```
// export-default.js
export default function () {
  console.log('foo');
}
```

上面代码是一个模块文件 `export-default.js`，它的默认输出是一个函数。

其他模块加载该模块时，`import` 命令可以为该匿名函数

```
// import-default.js
import customName from './export-default';
customName(); // 'foo'
```

上面代码的 `import` 命令，可以用任意名称指向 `export-default.js` 输出的方法，这时就不需要知道原模块输出的函数名。需要注意的是，这时 `import` 命令后面，不使用大括号。

`export default` 命令用在非匿名函数前，也是可以的。

```
// export-default.js
export default function foo() {
  console.log('foo');
}

// 或者写成

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

export default foo;
```

上面代码中，`foo` 函数的函数名 `foo`，在模块外部是无效的。加载的时候，视同匿名函数加载。

下面比较一下默认输出和正常输出。

```
// 第一组
export default function crc32() { // 输出
  // ...
}

import crc32 from 'crc32'; // 输入

// 第二组
export function crc32() { // 输出
  // ...
};

import {crc32} from 'crc32'; // 输入
```

上面代码的两组写法，第一组是使用 `export default` 时，对应的 `import` 语句不需要使用大括号；第二组是不使用 `export default` 时，对应的 `import` 语句需要使用大括号。

`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，因为只可能唯一对应 `export default` 命令。

本质上，`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。所以，下面的写法是有效的。

```
// modules.js
function add(x, y) {
  return x * y;
}
export {add as default};
// 等同于
// export default add;

// app.js
import { default as foo } from 'modules';
// 等同于
// import foo from 'modules';
```


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。

```
// 正确
export var a = 1;

// 正确
var a = 1;
export default a;

// 错误
export default var a = 1;
```

上面代码中，`export default a` 的含义是将变量 `a` 的值赋给变量 `default`。所以，最后一种写法会报错。

同样地，因为 `export default` 命令的本质是将后面的值，赋给 `default` 变量，所以可以直接将一个值写在 `export default` 之后。

```
// 正确
export default 42;

// 报错
export 42;
```

上面代码中，后一句报错是因为没有指定对外的接口，而前一句指定对外接口为 `default`。

有了 `export default` 命令，输入模块时就非常直观了，以输入 `lodash` 模块为例。

```
import _ from 'lodash';
```

如果想在一条 `import` 语句中，同时输入默认方法和其他接口，可以写成下面这样。

```
import _, { each, forEach } from 'lodash';
```

对应上面代码的 `export` 语句如下。

```
export default function (obj) {
  // ...
}

export function each(obj, iterator, context) {
  // ...
}

export { each as forEach };
```

上面代码的最后一行的意思是，暴露出 `forEach` 接口，默认指向 `each` 接口，即 `forEach` 和 `each` 指向同一个方法。

`export default` 也可以用来输出类。

```
// MyClass.js
export default class { ... }

// main.js
import MyClass from 'MyClass';
let o = new MyClass();
```

如果在一个模块之中，先输入后输出同一个模块，`import` 语句可以与 `export` 语句写在一起。

```
export { foo, bar } from 'my_module';

// 可以简单理解为
import { foo, bar } from 'my_module';
export { foo, bar };
```

上面代码中，`export` 和 `import` 语句可以结合在一起，写成一行。但需要注意的是，写成一行以后，`foo` 和 `bar` 实际上并没有被导入当前模块，只是相当于对外转发了这两个接口，导致当前模块不能直接使用 `foo` 和 `bar`。

模块的接口改名和整体输出，也可以采用这种写法。

```
// 接口改名
export { foo as myFoo } from 'my_module';

// 整体输出
export * from 'my_module';
```

默认接口的写法如下。

```
export { default } from 'foo';
```

具名接口改为默认接口的写法如下。

```
export { es6 as default } from './someModule';

// 等同于
import { es6 } from './someModule';
export default es6;
```

同样地，默认接口也可以改名为具名接口。

```
export { default as es6 } from './someModule';
```

下面三种 `import` 语句，没有对应的复合写法。

```
import * as someIdentifier from "someModule";
import someIdentifier from "someModule";
import someIdentifier, { namedIdentifier } from "someModule";
```

为了做到形式的对称，现在有提案，提出补上这三种复合写法。

```
export * as someIdentifier from "someModule";
export someIdentifier from "someModule";
export someIdentifier, { namedIdentifier } from "someModule";
```

8. 模块的继承

模块之间也可以继承。

假设有一个 `circleplus` 模块，继承了 `circle` 模块。

```
// circleplus.js

export * from 'circle';
export var e = 2.71828182846;
export default function(x) {
  return Math.exp(x);
}
```

上面代码中的 `export *`，表示再输出 `circle` 模块的所有属性和方法。注意，`export *` 命令会忽略 `circle` 模块的 `default` 方法。然后，上面代码又输出了自定义的 `e` 变量和默认方法。

这时，也可以将 `circle` 的属性或方法，改名后再输出。

```
// circleplus.js

export { area as circleArea } from 'circle';
```

上面代码表示，只输出 `circle` 模块的 `area` 方法，且将其改名为 `circleArea`。

加载上面模块的写法如下。

```
// main.js

import * as math from 'circleplus';
import exp from 'circleplus';
console.log(exp(math.e));
```

上面代码中的 `import exp` 表示，将 `circleplus` 模块的默认方法加载为 `exp` 方法。

9. 跨模块常量

本书介绍 `const` 命令的时候说过，`const` 声明的常量只在当前代码块有效。如果想设置跨模块的常量（即跨多个文件），或者说一个值要被多个模块共享，可以采用下面的写法。

```
// constants.js 模块
export const A = 1;
export const B = 3;
export const C = 4;

// test1.js 模块
import * as constants from './constants';
console.log(constants.A); // 1
console.log(constants.B); // 3

// test2.js 模块
import {A, B} from './constants';
console.log(A); // 1
console.log(B); // 3
```

如果要使用的常量非常多，可以建一个专门的 `constants` 目录，将各种常量写在不同的文件里面，保存在该目录下。

```
// constants/db.js
export const db = {
  url: 'http://my.couchdbserver.local:5984',
  admin_username: 'admin',
  admin_password: 'admin password'
};
```

```
// constants/user.js
export const users = ['root', 'admin', 'staff', 'ceo', 'chief', 'moderator'];
```

然后，将这些文件输出的常量，合并到 `index.js` 里面。

```
// constants/index.js
export {db} from './db';
export {users} from './users';
```

使用的时候，直接加载 `index.js` 就可以了。

```
// script.js
import {db, users} from './constants/index';
```

10. import()

简介

前面介绍过，`import` 命令会被 JavaScript 引擎静态分析，先于模块内的其他语句执行（`import` 命令叫做“连接” `binding` 其实更合适）。所以，下面的代码会报错。

```
// 报错
if (x === 2) {
  import MyModual from './myModual';
}
```

上面代码中，引擎处理 `import` 语句是在编译时，这时不会去分析或执行 `if` 语句，所以 `import` 语句放在 `if` 代码块之中毫无意义，因此会报句法错误，而不是运行时错误。也就是说，`import` 和 `export` 命令只能在模块的顶层，不能在代码块之中（比如，在 `if` 代码块之中，或在函数之中）。

这样的设计，固然有利于编译器提高效率，但也导致无法在运行时加载模块。在语法上，条件加载就不可能实现。如果 `import` 命令要取代 Node 的 `require` 方法，这就形成了一个障碍。因为 `require` 是运行时加载模块，`import` 命令无法取代 `require` 的动态加载功能。

```
const path = './' + fileName;
const myModual = require(path);
```

上面的语句就是动态加载，`require` 到底加载哪一个模块，只有运行时才知道。`import` 命令做不到这一点。

因此，有一个提案，建议引入 `import()` 函数，完成动态加载。

```
import(specifier)
```

上面代码中，`import` 函数的参数 `specifier`，指定所要加载的模块的位置。`import` 命令能够接受什么参数，`import()` 函数就能接受什么参数，两者区别主要是后者为动态加载。

`import()` 返回一个 `Promise` 对象。下面是一个例子。

```
const main = document.querySelector('main');
```

```
import(`./section-modules/${someVariable}.js`)
  .then(module => {
    module.loadPageInto(main);
  })
  .catch(err => {
    main.textContent = err.message;
  });
```

`import()` 函数可以用在任何地方，不仅仅是模块，非模块的脚本也可以使用。它是运行时执行，也就是说，什么时候运行到这一句，就会加载指定的模块。另外，`import()` 函数与所加载的模块没有静态连接关系，这点也是与 `import` 语句不相同。`import()` 类似于 `Node` 的 `require` 方法，区别主要是前者是异步加载，后者是同步加载。

适用场合

下面是 `import()` 的一些适用场合。

（1）按需加载。

`import()` 可以在需要的时候，再加载某个模块。

```
button.addEventListener('click', event => {
  import('./dialogBox.js')
  .then(dialogBox => {
    dialogBox.open();
  })
  .catch(error => {
    /* Error handling */
  })
});
```

上面代码中，`import()` 方法放在 `click` 事件的监听函数之中，只有用户点击了按钮，才会加载这个模块。

（2）条件加载

`import()` 可以放在 `if` 代码块，根据不同的情况，加载不同的模块。

```
if (condition) {
  import('moduleA').then(...);
} else {
  import('moduleB').then(...);
}
```

上面代码中，如果满足条件，就加载模块 A，否则加载模块 B。

（3）动态的模块路径

`import()` 允许模块路径动态生成。

```
import(f())
  .then(...);
```

上面代码中，根据函数 `f` 的返回结果，加载不同的模块。

`import()` 加载模块成功以后，这个模块会作为一个对象，当作 `then` 方法的参数。因此，可以使用对象解构赋值的语法，获取输出接口。

```
import('./myModule.js')
  .then(({export1, export2}) => {
    // ...
  });
```

上面代码中，`export1` 和 `export2` 都是 `myModule.js` 的输出接口，可以解构获得。

如果模块有 `default` 输出接口，可以用参数直接获得。

```
import('./myModule.js')
  .then(myModule => {
    console.log(myModule.default);
  });
```

上面的代码也可以使用具名输入的形式。

```
import('./myModule.js')
  .then(({default: theDefault}) => {
    console.log(theDefault);
  });
```

如果想同时加载多个模块，可以采用下面的写法。

```
Promise.all([
  import('./module1.js'),
  import('./module2.js'),
  import('./module3.js'),
])
  .then([module1, module2, module3]) => {
    ...
  });
```

`import()` 也可以用在 `async` 函数之中。

```
async function main() {
  const myModule = await import('./myModule.js');
  const {export1, export2} = await import('./myModule.js');
  const [module1, module2, module3] =
    await Promise.all([
      import('./module1.js'),
      import('./module2.js'),
      import('./module3.js'),
    ]);
}
main();
```

留言

98 Comments

ECMAScript 6 入门

 Login ▾

 Recommend 18

 Tweet

 Share

Sort by Best ▾



Join the discussion...

[上一章](#)

[下一章](#)

**Don** • 2 years ago

@ruanyf@ruanyf

50 ^ | v • Reply • Share ›

**Jess Jia** • 3 years ago

我想问问为什么我返回require is not defined ? 求帮助。

43 ^ | v • Reply • Share ›

**sdfsd sdf** → Jess Jia • a year ago

请问你是在何种环境下出现这个错误呢？

^ | v • Reply • Share ›

**Willin Wang** • 3 years ago

```
if(xxx){
  require('a');
}
else{
  require('b');
}
```

这种以后该怎么写

2 ^ | v • Reply • Share ›

**yugasun** → Willin Wang • a year ago

我一般是这么实现的：

```
require(`${xxx ? 'a' : 'b'})`
```

这个利用了 **require** 的支持动态引入特性。

^ | v • Reply • Share ›

**cjy** → Willin Wang • 2 years ago

写一个中间模块 mid.js

```
import a from 'a.js'
import b from 'b.js'
```

```
export {
  a,
  b
}
```

^ | v • Reply • Share ›

**ruanyf** Mod → Willin Wang • 3 years ago

```
import {a, b} from 'someModule';
```

```
if (x) {
```

```
if (x) {  
  a()  
} else {  
  b()  
}  
^ | v • Reply • Share ›
```



Jon Chung → ruanyf • 3 years ago

这种方式并没有达到他想实现异步加载脚本的目的

^ | v • Reply • Share ›



Youxiang Lu → Jon Chung • 3 years ago

你那require也达不到异步加载模块的功能

首先在浏览器端，无论if 还是else 都是会把a,b模块都加载的

在nodejs端，就更不用说了，压根不存在异步加载模块的说法，模块加载是同步的。

你的意思是不是按需加载？

51 ^ | v • Reply • Share ›



常修 • 10 months ago

最后的import()是后来加的，真是非常棒的内容。看来下一章必看了。

1 ^ | v • Reply • Share ›



陈耀军 • 3 years ago

根据 <https://www.nczonline.net/b...>

我想，是不是用AMD比有Module更好一些？

1 ^ | v • Reply • Share ›



杨培阳 • a month ago

Thanks a lot!

^ | v • Reply • Share ›



ymqy • 3 months ago

求推荐 ES5 相关书籍

^ | v • Reply • Share ›



liyun • 6 months ago

阮老师，请问“使得编译时就能确定模块的依赖关系”中的编译指得是在浏览器中编译的时候，还是ES6编译成ES5的时候。我使用webpack+babel编译打包的时候，模块中没有用到的方法，仍然被打包到输出文件中了

```
// a.js
```

```
import { add } from './b';
```

```
//b.js
```

```
export function add(str) {
```

```
  console.log(str);
```

```
}
```

```
export function foo(str) {
```



```
console.log(str);  
}
```

```
// c.js  
console.log('c')
```

打包后的文件，没有包含 **c.js** 的内容，但是包含了 **b.js** 中的 **foo** 方法

^ | v • Reply • Share ›



ruanyf Mod ➔ liyun • 6 months ago

引擎编译的时候。

1 ^ | v • Reply • Share ›



Cyclone • 9 months ago

中文引号用反了：**import**命令叫做“连接”**binding** 其实更合适

^ | v • Reply • Share ›



ruanyf Mod ➔ Cyclone • 8 months ago

谢谢，已经修正。

^ | v • Reply • Share ›



秦佳东 • 9 months ago

跨模块常量 一节中：

```
1、export const db = {  
  admin_username: 'admin',  
  admin_password: 'admin password'  
};
```

2、export const A = 1;

方式1的问题是：**db**不能修改，但是属性能修改，这样属性就不是常量了。

方式2的问题是：由于比1少了一层，导致容易和其他文件中定义的重名。

我的需求就是想定义一些常量（**url**、配置信息、版本信息等），

并且进行分类（结构清晰，防止重名），

java中很容易做到，但是**js**中我目前还没有好的办法，不知道阮老师有什么好办法没？

^ | v • Reply • Share ›



ruanyf Mod ➔ 秦佳东 • 9 months ago

采用这种语法应该可以

```
import * as CONSTANT from './constant';
```

2018-04-10 15:01 GMT+08:00 Disqus <notifications@disqus.net>:

^ | v • Reply • Share ›



秦佳东 ➔ ruanyf • 9 months ago

这种写法可以，完全满足我的需求。

不过有个小疑问，我在两个**js**文件中各定义了一个名称相同的**const**，按照你说的这种方式引用，居然不冲突，也不会相互覆盖值，定义的这两个**const**作用域应该是一样的啊，感觉应该会覆盖，但是并没有

[上一章](#)

[下一章](#)



陈银 • 10 months ago



```
import * as circle from './circle';
```

// 下面两行都是不允许的

```
circle.foo = 'hello';
```

```
circle.area = function () {};
```

现在是不是可以这样写了？

^ | v • Reply • Share ›



ruanyf Mod ➔ 陈银 • 10 months ago

你用了 Babel 吧。

^ | v • Reply • Share ›



Jianwen Li • a year ago

每次读都有新收获，感谢阮老师。

^ | v • Reply • Share ›



suyuTech • a year ago

哈哈、谢谢阮老师、总感觉自己遇到的所有问题、检索之后总能看到您写的文章。获益匪浅！

^ | v • Reply • Share ›



依韵 • a year ago

```
```js
```

```
export let a = {};
```

```
```
```

```
```js
```

```
import {a} from 'xxx.js'
```

```
a = {}; // Syntax Error : 'a' is read-only
```

```
```
```

我在[\[http://exploringjs.com/es6/...\]](http://exploringjs.com/es6/...)(<http://exploringjs.com/es6/...> 中找到了 **** Imports are read-only views on exports **** 的说明，而在此文中没有涉及到，是否会补充说明一下。

^ | v • Reply • Share ›



Yawei Xue • a year ago

想问下 `import * as myModule from './myModule'` 和 `import 'myModule'` 有什么区别啊？

^ | v • Reply • Share ›

[上一章](#)

[下一章](#)



yugasun → Yawei Xue • a year ago

前者会暴露出一个 `myModule` 变量，供你后面的代码使用，后者则不是，只是引入了 `./muModule` 中的所有模块依赖而已。

^ | v • Reply • Share ›



邓曲英 → Yawei Xue • a year ago

@ruanyf 同问老师。。。MDN上解释的 `import 'modules'` 看不太懂

^ | v • Reply • Share ›



陈妙妙 • a year ago

请问老师有了局部作用域以后是不是不用立即执行函数了？

^ | v • Reply • Share ›



ruanyf Mod → 陈妙妙 • a year ago

理论上是这样的。

^ | v • Reply • Share ›



陈妙妙 → ruanyf • a year ago

我刚刚入门，有点不理解为什么现在还是有很多框架用的是立即执行函数？老师

^ | v • Reply • Share ›



ruanyf Mod → 陈妙妙 • a year ago

兼容考虑。另外，这样更直观

2017年12月8日 11:49, "Disqus" <notifications@disqus.net>写道:

"我刚刚入门，有点不理解为什么现在还是有很多框架用的是立即执行函数？老师"

[image: Disqus] <https: disqus.com="" home="" ?

utm_medium="email&utm_content=logo"> Settings <https: disqus.com="" home="" settings="" email="" ?utm_medium="email"> <https: disqus.com="" home="" account="" ?utm_medium="email">

A new comment was posted on ECMAScript 6 入门 <http: disq.us="" url?

impression="28c8558e-dbca-11e7-b7f6-

002590f0f6c8&thread=3905630927&forum=2979275&url=http%3A%2F3652114818%3APUykFfc0TDxDcjfmW-

tyrQzchGA&variant=active&experiment=digests&behavior=click&----- <https: disq.us="" url?impression="28c8558e-dbca-11e7-b7f6-

002590f0f6c8&thread=3905630927&forum=2979275&url=https%3A%2*陈妙妙* <https: disq.us="" url?impression="28c8558e-dbca-11e7-b7f6-

002590f0f6c8&thread=3905630927&forum=2979275&url=https%3A%2

[see more](#)

^ | v • Reply • Share ›



Tonedony Jine • a year ago

写的好好哦

^ | v • Reply • Share ›



Xu Dong • a year ago

阮老师，好像有一点问题：
严格模式主要有以下限制。
变量必须声明后再使用

```
'use strict'  
a = 1;  
console.log(a);  
var a;
```

这段代码貌似可以在严格模式下运行

^ | v • Reply • Share ›



ruanyf Mod ➔ Xu Dong • a year ago

下面这样会报错。

```
use strict'  
a = 1;  
console.log(a);  
^ | v • Reply • Share ›
```



Joe Zheng • 2 years ago

```
export {  
a: 1,  
}
```

为什么上面那样 **expxort** 不行呢，按照您的说法，这种写法有接口啊？

^ | v • Reply • Share ›



qq20004604 ➔ Joe Zheng • a year ago

显然是没有的。要么 **export default {}**，要么按照写法写。最简单的说法，**export**后面的大括号，并不是对象，更不是对象的简写，所以你理解错了

^ | v • Reply • Share ›



余皖林 • 2 years ago

阮老师，你好。这篇文章没有书上的**20.5 module**命令部分吗？书上说道：

moudle命令可以代替**import**语句，达到整体输入模块的作用。

```
// main.js  
module circle from './circle';
```

^ | v • Reply • Share ›



ruanyf Mod ➔ 余皖林 • 2 years ago

规格里面拿掉了这个语法，这一节就删掉了。

^ | v • Reply • Share ›



Bendan Xiao • 2 years ago



^ | v • Reply • Share ›



ruanyf Mod ➔ Bendan Xiao • 2 years ago

这里其实是 `import {db, users} from './constants/index.js'`; 后面的 `index.js` 可以省略的。

^ | v • Reply • Share ›



李鲁杨 ➔ ruanyf • a year ago

一般情况不是 `import {db, users} from './constants/index'` 吗？
如果在一个目录下的文件是 `index.js` 甚至可以直接省略文件名？
那如果对应目录下文件是 `constants/index-01.js` 那还可以写为
`import {db, users} from './constants'` 吗？

^ | v • Reply • Share ›



ruanyf Mod ➔ 李鲁杨 • a year ago

`index-01.js` 不行，只有 `index.js` 可以省略文件名。

2 ^ | v • Reply • Share ›



常修 • 2 years ago

前端苦、前端累、前端还不是人。。。。。

^ | v • Reply • Share ›



Simba Chen • 2 years ago

一个模块被不同的地方引用，其中的变量值是一个引用的话，就失控了，这个怎么怎么解决

^ | v • Reply • Share ›



Jianwen Li ➔ Simba Chen • a year ago

所以阮老师也说了，尽量不要修改从模块中引入的变量的值，把它当做常量来用。

^ | v • Reply • Share ›



hiluluke • 2 years ago

终于理解vue里面两个文件引同一个module能共享的原因了！！

^ | v • Reply • Share ›



路小盆 • 3 years ago

阮老师，最后一个SystemJS 中好像import中传入的路径必须要带.js后缀，否则会报错。

^ | v • Reply • Share ›





ruanyf · 3 years ago

谢谢提醒，已经加上。

^ | v · Reply · Share ›



gloomyson · 3 years ago

请问能否在浏览器中使用模块呢？

我使用的是**babel standalone**，在page里加上了<script src="babel.min.js"></script>

然后构建模块M.js:

```
export var M={
```

```
p:3
```

```
};
```

再在page中使用:

```
<script type="text/babel">
```

```
import {M} from './M';
```

```
console.log(M.p);
```

```
</script>
```

报错: Uncaught ReferenceError: require is not defined

请问如何解决？

^ | v · Reply · Share ›

[Load more comments](#)

ALSO ON ECMAScript 6 入门

Module 的加载实现

18 comments · 2 years ago

ruanyf — 你说的对，这里是有错，我已经把缓存这部分删掉了。import() 可以用 Babel 执行。``javascript//

...

对象和函数的扩展

90 comments · 5 years ago

yunnysunny — 可惜Null传导运算符现在还不支持

Reflect

10 comments · 2 years ago

ruanyf — *谢谢指出，已经改正。*

数组的扩展

71 comments · 5 years ago

liyun — 不好意思，之前是我理解错了浅拷贝和深拷贝的概念

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus](#) [Disqus' Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#)

