**http://www.cnblogs.com/diligenceday/p/5503777.html**

**[ES6新特性：使用export和import实现模块化](http://www.cnblogs.com/diligenceday/p/5503777.html)**

**阅读目录**

* [ES6的模块化的基本规则或特点：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label0)
* [下面列出几种import和export的基本语法：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label1)
* [ES6导入的模块都是属于引用：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label2)
* [循环依赖的问题：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label3)
* [浏览器兼容：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label4)
* [参考：](http://www.cnblogs.com/diligenceday/p/5503777.html#_label5)

　　在ES6前， 前端就使用RequireJS或者seaJS实现模块化， requireJS是基于AMD规范的模块化库，  而像seaJS是基于CMD规范的模块化库，  两者都是为了为了推广前端模块化的工具， 更多有关AMD和CMD的区别， 后面参考给了几个链接；

 　　现在ES6自带了模块化， 也是JS第一次支持module， 在很久以后 ，我们可以直接作用**import**和**export**在浏览器中导入和导出各个模块了， 一个js文件代表一个js模块；

　　现代浏览器对模块(module)支持程度不同， 目前都是使用babelJS， 或者Traceur把ES6代码转化为兼容ES5版本的js代码;

[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**ES6的模块化的基本规则或特点：**

　　ES6的模块化的基本规则或特点， 欢迎补充：

　　　　1：每一个模块只加载一次， 每一个JS只执行一次， 如果下次再去加载同目录下同文件，直接从内存中读取。 一个模块就是一个单例，或者说就是一个对象；

　　　　2：每一个模块内声明的变量都是局部变量， 不会污染全局作用域；

　　　　3：模块内部的变量或者函数可以通过export导出；

　　　　4：一个模块可以导入别的模块

运行下面代码

[复制代码](javascript:void(0);)

//lib.js

//导出常量

export const sqrt = Math.sqrt;

//导出函数

export function square(x) {

return x \* x;

}

//导出函数

export function diag(x, y) {

return sqrt(square(x) + square(y));

}

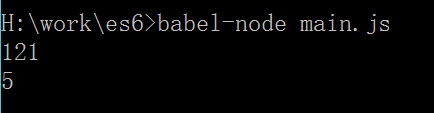
//main.js

import { square, diag } from './lib';

console.log(square(11)); // 121

console.log(diag(4, 3)); // 5

[复制代码](javascript:void(0);)



[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**下面列出几种import和export的基本语法：**

　　第一种导出的方式：

　　在lib.js文件中， 使用 **export{接口}** 导出接口， 大括号中的接口名字为上面定义的变量， **import**和**export**是对应的；

运行下面代码

[复制代码](javascript:void(0);)

//lib.js 文件

let bar = "stringBar";

let foo = "stringFoo";

let fn0 = function() {

console.log("fn0");

};

let fn1 = function() {

console.log("fn1");

};

export{ bar , foo, fn0, fn1}

//main.js文件

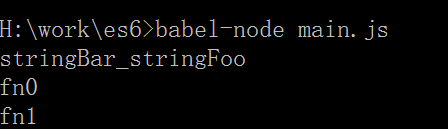
import {bar,foo, fn0, fn1} from "./lib";

console.log(bar+"\_"+foo);

fn0();

fn1();

[复制代码](javascript:void(0);)



　　第二种导出的方式：

 　　在export接口的时候， 我们可以使用 XX as YY， 把导出的接口名字改了， 比如： **closureFn as sayingFn**， 把这些接口名字改成不看文档就知道干什么的：

运行下面代码

[复制代码](javascript:void(0);)

//lib.js文件

let fn0 = function() {

console.log("fn0");

};

let obj0 = {}

export { fn0 as foo, obj0 as bar};

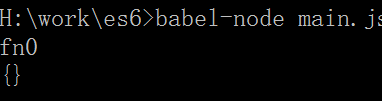
//main.js文件

import {foo, bar} from "./lib";

foo();

console.log(bar);

[复制代码](javascript:void(0);)



　　第三种导出的方式：

　　这种方式是直接在export的地方定义导出的函数，或者变量：

运行下面代码

[复制代码](javascript:void(0);)

//lib.js文件

export let foo = ()=> {console.log("fnFoo") ;return "foo"},bar = "stringBar";

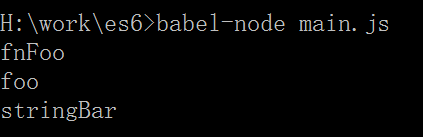
//main.js文件

import {foo, bar} from "./lib";

console.log(foo());

console.log(bar);

[复制代码](javascript:void(0);)



　　第四种导出的方式：

　　这种导出的方式不需要知道变量的名字， 相当于是匿名的， 直接把开发的接口给export；  
　　如果一个js模块文件就只有一个功能， 那么就可以使用export default导出；

运行下面代码

[复制代码](javascript:void(0);)

//lib.js

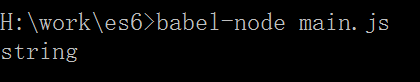
export default "string";

//main.js

import defaultString from "./lib";

console.log(defaultString);

[复制代码](javascript:void(0);)



　　第五种导出方式：

　　export也能默认导出函数， 在import的时候， 名字随便写， 因为每一个模块的默认接口就一个：

运行下面代码

[复制代码](javascript:void(0);)

//lib.js

let fn = () => "string";

export {fn as default};

//main.js

import defaultFn from "./lib";

console.log(defaultFn());

[复制代码](javascript:void(0);)

　　第六种导出方式：

　　使用通配符\*  ,重新导出其他模块的接口 (其实就是转载文章， 然后不注明出处啦);

运行下面代码

[复制代码](javascript:void(0);)

//lib.js

export \* from "./other";

//如果只想导出部分接口， 只要把接口名字列出来

//export {foo,fnFoo} from "./other";

//other.js

export let foo = "stringFoo", fnFoo = function() {console.log("fnFoo")};

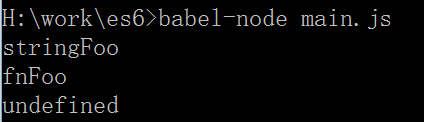
//main.js

import {foo, fnFoo} from "./lib";

console.log(foo);

console.log(fnFoo());

[复制代码](javascript:void(0);)



　　其他：ES6的import和export提供相当多导入以及导出的语法；

　　在**import**的时候可以使用**通配符\***导入外部的模块：

运行下面代码

import \* as obj from "./lib";

console.log(obj);

[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**ES6导入的模块都是属于引用：**

　　每一个导入的js模块都是活的， 每一次访问该模块的变量或者函数都是最新的， 这个是**原生ES6模块** 与**AMD**和**CMD**的区别之一,以下代码修改自[http://exploringjs.com/es6/ch\_modules.html#\_imports-are-read-only-views-on-exports](http://exploringjs.com/es6/ch_modules.html" \l "_imports-are-read-only-views-on-exports" \t "_blank)

运行下面代码

[复制代码](javascript:void(0);)

//lib.js

export let counter = 3;

export function incCounter() {

counter++;

}

export function setCounter(value) {

counter = value;

}

//main.js

import { counter, incCounter ,setCounter} from './lib';

// The imported value `counter` is live

console.log(counter); // 3

incCounter();

console.log(counter); // 4

setCounter(0);

console.log(counter); // 0

[复制代码](javascript:void(0);)

　　在main.js中， counter一直指向lib.js中的局部变量counter， 按照JS的尿性， 像**数字**或者**字符串类型**或者**布尔值**的原始值要被复制， 而不是赋址；

[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**循环依赖的问题：**

　　NodeJS的循环依赖是这么处理的：[打开；](https://nodejs.org/api/modules.html" \l "modules_cycles" \t "_blank)

　　循环依赖是JS模块化带来的问题， 在浏览器端， 使用RequireJS测试模块化， 比如有一个文件file0.js依赖于file1.js， 而file1.js又依赖于file0.js， 那么file0.js和file1.js到底谁先执行？

运行下面代码

[复制代码](javascript:void(0);)

//index.html

<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8"/>

</head>

<body>

<script data-main="cyclic" src="//cdn.bootcss.com/require.js/2.2.0/require.min.js"></script>

<script>

//cyclic.js

require(["file0"], function(file0) {

console.log(file0)

})

//file0.js

define(["file1"], function(file1) {

console.log(file1)

return {

file0 : "file0"

}

})

//file1.js

define(["file0"], function(file0) {

console.log(file0);

return {

file1 : "file1"

}

})

</script>

</body>

</html>

[复制代码](javascript:void(0);)

　　在控制台的依次输出为：

运行下面代码

undefined

Object { file1: "file1" }

Object { file0: "file0" }

　　在执行file1.js的时候file0.js还没执行完， 所以输出了undefined， 这种输出结果和NodeJS输出的情况是一样的；

　　然后我又使用了司徒大神的**mass-framework**框架试了一下， 司徒大神的框架直接提示我： **"模块与之前的某些模块存在循环依赖"**， 这样还比较好点， requireJS对于**循环依赖**是直接执行循环依赖的模块， 会导致在开发的时候给自己挖坑....；

　　接下来我又在**babel-node**下进行测试：**下面是几个测试，可以无视：**

　　我使用ES6的模块试一试， 只要每一个模块被引用， 无论模块是否执行完毕， 该模块的export已经被导出了， 如果导出的是函数：

运行下面代码

[复制代码](javascript:void(0);)

//cyclic.js

import fn0 from "./file0";

fn0();

//file0.js

import fn1 from "./file1";

fn1();

console.log("file0.js runs");

export default function() {console.log("file0 export runs")}

//file1.js

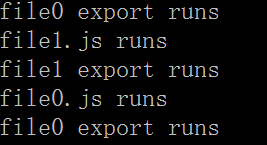
import fn0 from "./file0";

fn0();

console.log("file1.js runs");

export default function() {console.log("file1 export runs")}

[复制代码](javascript:void(0);)



　　如果导出的是字符串：

运行下面代码

[复制代码](javascript:void(0);)

//cyclic.js

import str from "./file0";

console.log(str);

//file0.js

import str1 from "./file1";

console.log(str1)

console.log("file0.js runs");

export default "str0";

//file1.js

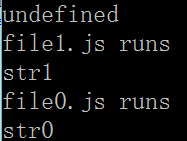
import str0 from "./file0";

console.log(str0)

console.log("file1.js runs");

export default "str1";

[复制代码](javascript:void(0);)



**如果导出的是对象：**

　　那么第一行会先输出一个初始值**{},**在最后等待**file0.js**和**file1.js**执行完毕以后， 才输出**file0.js**导出的对象；

**如果是数组：**

　　那么第一行会输出一个被静态分析过的初始值**undefined**,在最后等待**file0.js**和**file1.js**执行完毕以后， 才输出**file0.js**导出的对象；

**如果是布尔值：**

　　那么第一行会输出一个被静态分析过的初始值**undefined**,在最后等待**file0.js**和**file1.js**执行完毕以后， 才输出**file0.js**导出的布尔值；

　　为什么会这样呢？ 我好像在这边找到了答案：<http://exploringjs.com/es6/ch_modules.html#_modules> ，ES6的import和export被提前到js的最顶层， 在函数或者对象，或者基本值被导出去的时候提前被静态分析过，参考：<http://www.ecma-international.org/ecma-262/6.0/#sec-parsemodule> , <http://www.ecma-international.org/ecma-262/6.0/#sec-toplevelmoduleevaluationjob>

　　结论：用ES6的export导出数据接口的时候， 最好统一用函数， 避免在循环依赖的时候， 因为JS会把不同类型的对象静态解析成不同的初始值;

1

1

[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**浏览器兼容：**

　　　　chrome浏览器目前不支持import，和export；

　　　　火狐的支持也有限， 比chrome好；

　　　　我都用babel；

[回到顶部](http://www.cnblogs.com/diligenceday/p/5503777.html#_labelTop)

**参考：**

　　　　ecma-262：[http://www.ecma-international.org/ecma-262/6.0/#sec-imports](http://www.ecma-international.org/ecma-262/6.0/" \l "sec-imports" \t "_blank)

　　　　import：[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import" \t "_blank)

　　　　export：[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export" \t "_blank)

　　　　babelJS：[http://babeljs.io/](http://babeljs.io/" \t "_blank)

　　　　Exploring ES6(Dr.Axel Rauschmayer)：[http://exploringjs.com/es6/ch\_modules.html](http://exploringjs.com/es6/ch_modules.html" \t "_blank)

　　　　SeaJS和RequireJS的异同：[https://github.com/seajs/seajs/issues/277](https://github.com/seajs/seajs/issues/277" \t "_blank)

　　　　SeaJS与RequireJS最大的区别：<https://www.douban.com/note/283566440/>

　　　　阮老师es6#module：<http://es6.ruanyifeng.com/#docs/module>

　　　　AMD 和 CMD 的区别有哪些：<https://www.zhihu.com/question/20351507/answer/14859415>

作者： [**NONO**](http://www.cnblogs.com/diligenceday/)  
出处：<http://www.cnblogs.com/diligenceday/>   
QQ：287101329   
微信：18101055830