

# ES6 Module

2015年8月20日 23:23:32

滚屏阅读

 阅读 15       评论 0

## ES6 Module

相对比起Python,PHP等其他语言, JavaScript 在设计上似乎缺失了模块管理的部分, 不过好消息是这些问题很快就会得到解决, 在下一代JavaScript ECMASCIPT 6中, 这个问题被很好的解决了。

## ES6 Module 特性

Module 特性是在14年的7月被完全确认的, 这就意味着到目前为止, 语法相对比较稳定。

Module 包含了2个主要的部分, 模块的定义 `export` 和模块的引入 `import`, 如果是写过 Python 的同学对这个部分的概念会很容易理解, 因为语法真的很像、很像。

### 一、Export

`export` 顾名思义, 就是定义模块的输出, 基础语法很简单:

```
export {export Name}
```

但根据输出的参数不同, 它的表现形式也有所区别。

```
export var a = '123';

export function name(arguments){ //... }

export {nameA, nameB}

export default function name(arguments){ //.. }

export *
```

#### 1. 直接输出

在前两行中, 我们对外输出了一个变量a, 一个方法name, 你可能已经留意到了, 我们的方法和变量的声明都是直接写在的 `export`的后面, 那么你会想如果`export`关键字后面直接填变量名会怎么样呢?

尝试如下:

```
// BAD EXAMPLE !!

var a = '123';

function name(arguments) { //... }

export a;

// or
// export name;
```

执行之后会发现，这种在export后面直接写变量名的方式是会触发Error的，这种情况下我们只能吧定义写在export的后面。那么问题来了，如果所有的定义都写在export后面的话，相信处女做的同学们可能会坐不住了，别急，我们有第二种的方法。

## 2. 输出变量

这种情况应该比第一种方法要常见，很多时候我们会在输出之前定义好方法或者变量，如果需要输出可以加一个花括号，同样拿上面的例子来说明：

```
var a = '123';

function name(arguments) { //... }

export {a, name};

// or
// export {a}
// export {name}
```

这样就能愉快的输出变量了，不过和 return 略有区别的是，我们在export之后还能继续输出。

也就是说下面的两条语句的表现完全相同：

### 单行模式

```
export {a, name}
```

### 多行模式

```
export {a}
export {name}
```

## 3. default

default模式比较特殊，也比较有用，先看例子吧：

### a.js

```
var $ = function() {}

$.prototype.show = function(){ //.. }

export default $
```

### b.js

```
import jQuery from 'a.js'
```

由于还没有说到import所以暂时忽略b.js，这里的export default \$是指该文件返回\$方法/类，import调用的时候可以自定义导入的名称。

简单来说，这种情况多用于封装大型类库，比如引入jQuery,backbone等之类的类库的时候，非常方便。

4.\*

据说可以导出所有的变量和方法，但是没有实际测试过，后续接触到之后在详细解说。

## 二、Import

介绍完export之后import应该就不用再细说概念了，其实就是导入的意思，其基本方法也异常简单：

```
import Name from Module
```

同样它也有几种表现形式：

```
import CustomerName from 'module';
import {nameA,nameB} from 'module'
import CustomerName,{nameA} from 'module'
import {name as alias} from 'module'
import 'module'
```

### 1. 导入default

正如export时候说到的default，在export中通过default定义的输出，我们可以简单的通过自定义方法名来导入默认模块。

例子参考export的default的例子，就不多说了。

### 2. 导出独立变量/方法

通过花括号，我们可以导入在export中声明的多个或一个变量/方法

例如：

a.js

```
var a = 1;
var b = 2;
var c = function(){ return 3 }

export {a,b,c}
```

**b.js**

```
// 一次导入一个变量/方法
import {a} from 'a.js'

// 一次导入多个变量/方法
import {b,c} from 'a.js'

console.log(a,b,c());
```

**3. default 和 变量/方法 导入混用**

实际上再使用过程中，import非常灵活，我们甚至可以混合导入一些东西，例如：

**a.js**

```
var a = 1;
var b = 2;
var c = function(){ return 3 };
var d = function(){ return 4 };
export {a,b,c};
export default d;
```

**b.js**

```
// default 混合导入
import funD,{a,b,c} from 'a.js'
console.log(funD(), a, b, c())
```

**4. 别名导入**

除了default导入可以自定义名称外，我们可以通过as来声明别名

```
import {a as vara,c as func} from 'a.js'
```

Follow me: 新浪微博 腾讯微博 Google+ RSS

**5. 执行导入**

很奇怪的一个方法，他会执行导入的方法中的js，但不会返回任何值。

**a.js**

```
var a = 1;
console.log(a)
export {a}
```

**b.js**

```
import 'a.js'
// 会执行a.js 在控制台输出 1

console.log(a)
// 返回 undefined 因为， import moudle 不会返回任何值，并且执行的环境也是在独立的scope
```

## 结尾

总的来说 ES6 的模块管理让我们看到了很多的希望，也给我们带了更多的可能，不知道它会不会让AMD，CMD慢慢的走向坟墓。至于大家问的比较多的问题，究竟什么时候可以使用 ES6，答案是现在就可以用，我们有很多编译工具，可以平稳的把ES6代码降级为兼容的ES5代码，后面有空我会单独来聊聊ES6的在项目中使用的方法的，今天就先说到这里吧。

本文地址<http://www.zhuwenlong.com/blog/55d5f0f48cf91c7157121a0d>

原创文章，转载请注明出处[朱文龙的自留地](#)

Mofei

13761509829@163.com

<http://www.zhuwenlong.com>

Verification code

Let's say some thing

Post Comment

---

首页    博客    实验    留言    小伙伴们