CommonJs和ES6 module的区别是什么呢?

关注问题 写回答

JavaScript ECMAScript 2015

关注者 201 被浏览 199,310

CommonJs和ES6 module的区别是什么呢?

现在nodejs还是用的CommonJs规范，有点搞不懂他们的区别是什么？显示全部

关注问题 写回答 邀请回答 好问题 15 添加评论 分享 ···

查看全部 17 个回答

黑马程序员前端

分享前端资源、就业项目，免费答疑有问必答~不信可以私信我呀❤

+ 关注

7 人赞同了该回答

ES6 module和CommonJS到底有什么区别?

“ES6 module是编译时加载，输出的是接口，CommonJS运行时加载，加载的是一个对象”，这里的“编译时”是什么意思？和运行时有什么区别？“接口”又是什么意思？

“ES6 模块输出的是值的引用，CommonJS 模块输出的是一个值的拷贝”，那么“值的引用”和“值的拷贝”对于开发者又有什么区别呢?

下面通过一些示例详细说明ES6 module和CommonJS的区别。

编译时导出接口 VS 运行时导出对象

CommonJS 模块是运行时加载，因为 CommonJS 加载的是一个对象（即module.exports属性），该对象只有在脚本运行完才会生成。

ES6 模块是它的对外接口只是一种静态定义，在代码静态解析阶段就会生成。

这里的“编译时”，指的是js代码在运行之前的编译过程，我们熟悉的变量提升就发生在编译阶段，但是由于编译过程是引擎的行为，开发者没法在编译阶段做任何操作，所以不容易直观地理解“编译时导出接口和运行时导出对象”这个区别。

不过我们在循环引用这个场景就可以轻松地理解两者的差异。

来看下面CommonJS的代码

```
// index.js
const {log} = require('./lib.js');
module.exports = {
    name: 'index'
};
log();

// lib.js
const {name} = require('./index.js');
module.exports = {
    log: function () {
        console.log(name);
    }
};
```

▲ 赞同 7 ▼ 1 条评论 分享 收藏 喜欢 ···

CommonJs和ES6 module的区别是什么呢?

这里index模块和lib相互依赖。

我们分析一下代码执行过程，首先const {log} = require('./lib.js');导入lib.js模块，这时候开始加载lib模块，lib会先导入index，const {name} = require('./index.js');但这个时候index还没有定义name，所以lib中这里的name是undefined，然后lib导出log方法。

接下来index导出name，然后执行log，由于lib中的name是undefined，因此最终结果是打印undefined。

整个过程模块都是运行时加载，代码依次执行，所以很容易分析出执行结果。

而ES6 module有所不同，接下来看一个es6 module的例子。代码内容和上面一样，只是把模块规范从CommonJS换成es6 module。

```
// index.mjs
import {log} from './lib.mjs';
export const name = 'index';
log();

// lib.mjs
import {name} from './index.mjs';
export const log = () => {
    console.log(name);
};
```

首先import {log} from './lib.mjs';导入lib模块，注意这个时候虽然没有执行export const name = 'index';，index模块还没有导出name的值，但是index模块已经编译完成，lib已经可以获取到name的引用，只是还没有值。这非常像代码编译阶段的变量提升。

然后加载lib模块，import {name} from './index.mjs';这句导入了index模块的name，（这时候获取到的是name这个引用，但因为还没有值，因此如果马上打印name console.log(name)会报错。）接下来lib导出log方法。

然后index模块执行export const name = 'index';导出name，其值为"index"。

最后执行log方法log();因为name已经赋值，所以lib中name的引用可以正常访问到值"index"，所以最终结果是打印"index"。

综上所述，es6 module虽然模块未初始化好时候就被lib导入，但因为获取的是导出的接口（接口编译阶段就已经输出了），等初始化好之后就能使用了。

引用 VS 值拷贝

ES6 module导入的模块不会缓存值，它是一个引用，这个在上面的例子中已经讨论过。

CommonJS会缓存值，这个很好理解，因为js中普通变量是值的拷贝，其实就是把模块中的值赋给一个新的变量。

看下CommonJS的一个例子：

```
// index.js
const {name} = require('./lib.js'); // 等价于const lib = require('./lib'); const {name}
setTimeout(() => {
    console.log(name); // 'Sam'
}, 1000);
```


关于作者

黑马程序员前端
分享前端资源、就业项目，...

回答 819
文章 243
关注者 11,532

关注 发私信

CommonJs和ES6 module的区别是什么呢?

```
const lib = {
    name: 'Sam'
};
module.exports = lib;
setTimeout(() => {
    lib.name = 'Bob';
}, 500);
```

index模块中导入lib的nameconst {name} = require('./lib.js');其实就是把lib中的name赋给index里面一个name变量。后面lib中name的变化不会影响到index中的name变量。

而ES6中类似的引用语法，导入的则是引用

```
// index.mjs
import {name} from './lib.mjs';
setTimeout(() => {
    console.log(name); // 'Bob'
}, 1000);

// lib.mjs
export let name = 'Sam';
setTimeout(() => {
    name = 'Bob';
}, 500);
```

这里index模块中的name是lib导出的name的引用，因此lib中name变化会同步到index中。

当然这并不意味着ES6 module可以做到比CommonJS更多的事情，因为如果希望在CommonJS中获取到变化，也可以直接访问lib.name。

```
// index.js
const lib = require('./lib.js');
setTimeout(() => {
    console.log(lib.name); // 'Bob'
}, 1000);
```

所以这个特性的区别只是需要我们在实现模块时候注意一下，避免预期之外的情况。

其实在上面循环引用的例子中，也能看到CommonJS拷贝值和ES6 module引用的区别，CommonJS因为是拷贝值，所以导入模块时候如果还没初始化好，就是undefined，而ES6 module是引用，所以初始化好之后就可以用了。

静态 VS 动态

ES6 module静态语法和CommonJS的动态语法是很重要的区别，

CommonJS的动态性体现在两个方面

可以根据条件判断是否加载模块

```
if (condition) {
    require('./lib');
}
```

王成 创建

前端 0 人关注
hexuehui12 创建

工作 0 人关注
走走停停 创建

相关问题

Commonjs 和 Es Module 到底有什么区别? 0 个回答

ES6 与 CommonJS 中是如何对模块进行处理的? 0 个回答

ES6 module 和 CommonJS 的区别是什么? 0 个回答

为什么 CommonJS 和 ES Module 不能共存? 0 个回答

前端 ES6 与 CommonJS 中是如何对模块进行处理的? 0 个回答

CommonJs和ES6 module的区别是什么呢?

```
require(`./${template}/index.js`);
```

这种动态性导致依赖关系不好分析，打包工具在静态代码分析阶段不容易知道模块是否需要被加载、模块的哪些部分需要被加载，哪些不会被用到。

相应地，ES6 module的静态性体现在

1. import必须在顶层
2. import的模块参数只能是字符串，不能是变量

所以打包工具能够静态分析出依赖关系，并确定知道哪些模块需要被加载、模块的哪些部分被用到。

所以ES6 module静态语法支持打包tree-shaking，而CommonJS不行。

只读 VS 可变

CommonJS导入的模块和普通变量没有区别，ES6 module导入的模块则不同，import导入的模块是只读的。

```
// demo-commonjs.js
let path = require('path');
path = 1;

// demo-esm.js
import path from 'path';
path = 1; // Error: Cannot assign to 'path' because it is an import
```

异步 VS 同步

ES6 module支持异步加载，浏览器中会用到该特性，而Commonjs是不支持异步的，因为服务器端不需要异步加载。所以CommonJS不可替代ES6 module，ES6 module可以替代CommonJS。

总结

ES6 module和CommonJS的区别主要有5点

1. ES6 module是编译时导出接口，CommonJS是运行时导出对象。
2. ES6 module输出的值的引用，CommonJS输出的是一个值的拷贝。
3. ES6 module语法是静态的，CommonJS语法是动态的。
4. ES6 module导入模块的是只读的引用，CommonJS导入的是可变的，是一个普通的变量。
5. ES6 module支持异步，CommonJS不支持异步。

ES6 module作为新的规范，可以替代之前的AMD、CMD、CommonJS作为浏览器和服务端的通用模块方案。NodeJS在13.2.0版本后已经开始完全支持ES6 module，ES6 module在未来也会实际的模块化标准。

发布于 2022-05-27 06:02 · IP 属地北京

更多回答

CommonJs和ES6 module的区别是什么呢?

可以看以下这篇文章，这篇文章深入解释了为什么这两种模块难以互相兼容。在解释这个问题的过程中，作者讲解了这两种模块具体的生命周期：何时加载？何时编译？何时执行？中间如何进行优化避免加载、编译或执行不必要的代码？

展开阅读全文

作者是 Redfin 的 Principal Engineer。他不仅仅讲解了具体的标准是怎么样的，还提及了一些标准制定过程中的趣事。非常推荐大家花时间去阅读。

▲ 赞同 203 ▼ 5 条评论 分享 收藏 喜欢 ···

王玉略
写过点代码读过点书https://wangyulue.com

阮一峰在 ES6 入门 中提到 ES6 模块与 CommonJS 模块有一些重大的差异:

- CommonJS 模块输出的是一个值的拷贝，ES6 模块输出的是值的引用。
- CommonJS 模块是运行时加载，ES6 模块是编译时输出接口。

再细读上面阮老师提到的差异，会产生诸多疑问:

- 为什么 CommonJS 模块输出的是一个值的拷贝？其具体细节是什么样子的?
- 什么叫 运行时加载 ？
- 什么叫 编译时输出接口 ？
- 为什么 ES6 模块输出的是值的引用?

于是就有了这篇文章，力求把 **ESM 模块** 和 **CommonJS 模块** 讨论清楚。

CommonJS 产生的历史背景

展开阅读全文

CommonJS 由 Mozilla 工程师 Kevin Dangoor 于 2009 年 1 月创立，最初命名为ServerJS。

▲ 赞同 389 ▼ 9 条评论 分享 收藏 喜欢 ···

查看全部 17 个回答