CommonJS

前言

NodeJS中使用的模块规范。
根据CommonJS规范，一个单独的文件就是一个模块（module）。加载模块使用require方法，该方法读取一个文件并执行，最后返回文件内部的module.exports对象。
CommonJS采用同步加载模块的方式，只有加载完才能执行后面的操作，主要适用于服务端。

使用规则

1、exports与require使用的模块导入导出规则遵循的是CommonJS(AMD和CMD)规范, 一般在NodeJS(express,koa)中使用，是相对比较先出现的规范,也是目前大多数浏览器支持的模块导入导出规范。

module.exports 与 exports

1、一个文件即为一个module；
2、一个module中有很多字段，例如 `id` `path` `parent` `exports` 等等，可以直接在js文件输出 `module` 即可查看；
3、`exports` 表示这个module要导出的数据，`module.export` 默认指向一个空的对象内存；
4、变量 `exports` 默认指向 `module.exports` （引用传递）；
5、`module` 实际上导出的数据是在 `module.export` 这个变量中；

require

1、使用 `require` 即可导入另一模块中导出的数据；

exports与require之间的联系

无论模块 `exports` 的是什么数据类型，`module.exports` 与 `require` 指向的是同一块内存地址，任何一方改变都会影响另一方的数据（动态）

案例一：

```
// a.js文件
let aNum = 1;
let aArr = [1];
setTimeout(() => {
  aNum = 11; // 并不会影响module.exports.aNum的值
  aArr.push(11); // 会影响module.exports.aArr的值
  console.log('1s a模块改变数据aNum&aArr:');
  console.log(`aNum=${aNum},module.exports.aNum=${module.exports.aNum}`);
  console.log(`aArr=${aArr},module.exports.aArr=${module.exports.aArr}`);
}, 1000)
setTimeout(() => {
  console.log(`3s a模块接受数
据:aNum=${module.exports.aNum},aArr=${module.exports.aArr}`);
}, 3000)
// 以下写法等价于
// module.exports.aNum = aNum aNum属于基本类型 值拷贝
// module.exports.aArr = aArr aArr属于对象 值引用
module.exports = {
```

```
  aNum,
  aArr
};
console.log(`a模块已导出数
据:aNum=${module.exports.aNum},aArr=${module.exports.aArr}`);

// b.js文件
const a = require("./commonjs_a");
console.log(`b模块已接收数据:aNum=${a.aNum},aArr=${a.aArr}`);
setTimeout(() => {
  console.log(`2s b模块接受数据:aNum=${a.aNum},aArr=${a.aArr}`);
  a.aNum = 2;
  a.aArr = [2]
  console.log(`2s b模块改变数据:aNum=${a.aNum},aArr=${a.aArr}`);
}, 2000)


// 输出结果
a模块已导出数据:aNum=1,aArr=1
b模块已接收数据:aNum=1,aArr=1
1s a模块改变数据aNum&aArr:
aNum=11,module.exports.aNum=1
aArr=1,11,module.exports.aArr=1,11
2s b模块接受数据:aNum=1,aArr=1,11
2s b模块改变数据:aNum=2,aArr=2
3s a模块接受数据:aNum=2,aArr=2
```

ES6 Module

前言

使用规则

1、export与import是ES组织官方退出的模块化方案,一般在typescript和三大框架(Angular, Vue, React)中比较常见,但目前支持这套规范的客户端浏览器比较少,所以通常情况下代码都要经过Babel转换成目前浏览器能支持的,也就是exports和require那一套。

export 与 export default

1、无论导出数据是什么类型的，`export`导出的都是变量的引用绑定；

2、任何未显式导出的变量、函数或类都是模块私有的，无法从模块外部访问；

```
// es6_a.mjs
export let aNum = 1;
export let aArr = [1];
setTimeout(() => {
  aNum = 11;
  aArr.push(11);
  console.log('1s a模块改变数据aNum&aArr:');
  console.log(`aNum=${aNum},`);
  console.log(`aArr=${aArr},`);
}, 1000)
console.log(`a模块已导出数据:aNum=${aNum},aArr=${aArr}`);
```

```
// es6_b.mjs
// import时必须加{ }
import {aNum, aArr} from './es6_a.mjs';
console.log(`b模块已接收数据:aNum=${aNum},aArr=${aArr}`);
setTimeout(() => {
  console.log(`2s b模块接受数据:aNum=${aNum},aArr=${aArr}`);
}, 2000);

// 输出结果
a模块已导出数据:aNum=1,aArr=1
b模块已接收数据:aNum=1,aArr=1
1s a模块改变数据aNum&aArr:
aNum=11
aArr=1,11
2s b模块接受数据:aNum=11,aArr=1,11
```

2、`export default`有点特殊，导出的数据要根据数据类型来定，与CommonJS的`module.exports`有点类似

```
// es6_a.mjs
let aNum = 1;
let aArr = [1];
setTimeout(() => {
  aNum = 11;
  aArr.push(11);
  console.log('1s a模块改变数据aNum&aArr:');
  console.log(`aNum=${aNum}`);
  console.log(`aArr=${aArr}`);
}, 1000)
// 与module.exports类似
export default {
  aNum, // 数值类型 深拷贝
  aArr // 对象类型 浅拷贝
}
console.log(`a模块已导出数据:aNum=${aNum},aArr=${aArr}`);

// es6_b.mjs
import a from './es6_a.mjs';
console.log(`b模块已接收数据:aNum=${a.aNum},aArr=${a.aArr}`);
setTimeout(() => {
  console.log(`2s b模块接受数据:aNum=${a.aNum},aArr=${a.aArr}`);
}, 2000);

// 输出结果
a模块已导出数据:aNum=1,aArr=1
b模块已接收数据:aNum=1,aArr=1
1s a模块改变数据aNum&aArr:
aNum=11
aArr=1,11
2s b模块接受数据:aNum=1,aArr=1,11
```

3、* 在`export * from 'xxx'`后面加注释会报怪异的bug，导出的数据为`undefined`

import

1、导入`default`的不需要使用`{ }`，导入非默认模块则需要加`{ }`

2、一个文件可以同时导出默认的模块和非默认的模块，如下所示：

```
// es6_a.mjs
export let color = 'red';
export default function(num1, num2) {
  return num1 + num2;
}

// es6_b.mjs
// 默认值必须排在非默认值之前
import sum, { color } from  './es6_a.mjs';
console.log(sum(1,2));
console.log(color);

// 输出结果
3
red
```

3、可以使用`as`在导出和导入时重命名

```
// es6_a.mjs
function sum(num1, num2) {
  return num1 + num2;
}
export { sum as a };

// es6_b.mjs
import { a as aa } from './es6_a.mjs';
console.log(aa(1,2));

// 输出结果
3
```

两者差异

1. CommonJS 模块输出的是一个值的拷贝（根据数据类型分为深拷贝或浅拷贝），ES6 模块输出的是值的引用（任何数据类型）
CommonJS 模块输出的是值的拷贝，也就是说，一旦输出一个值，模块内部的变化就影响不到这个值。ES6 模块的运行机制与 CommonJS 不一样。JS 引擎对脚本静态分析的时候，遇到模块加载命令 import，就会生成一个只读引用。等到脚本真正执行时，再根据这个只读引用，到被加载的那个模块里面去取值。换句话说，ES6 的import有点像 Unix 系统的“符号连接”，原始值变了，import加载的值也会跟着变。因此，ES6 模块是动态引用，并且不会缓存值，模块里面的变量绑定其所在的模块。

2. CommonJS 模块是运行时加载，ES6 模块是编译时输出接口

3. ES6 Module中导入的数据为`const`，不允许重新赋值；而CommonJS中导入的数据允许重新赋值

参考1