

CommonJS

JavaScript 是一个强大面向对象语言，它有很多快速高效的解释器。然而，JavaScript 标准定义的 API 是为了构建基于浏览器的应用程序。并没有制定一个用于更广泛的应用程序的标准库。CommonJS 规范的提出,主要是为了弥补当前 JavaScript 没有标准的缺陷。它的终极目标就是：提供一个类似 Python，Ruby 和 Java 语言的标准库,而不只是停留在小脚本程序的阶段。用 CommonJS API 编写出的应用，不仅可以利用 JavaScript 开发客户端应用，而且还可以编写以下应用。

- 服务器端的应用
- 命令行工具
- 桌面图形界面应用

**** CommonJS 是模块化的标准，Nodejs是模块化的实现 ****

模块化系统

1. 一个js文件 就是一个模块。
2. 文件内容 js代码、json、编译过c/c++扩展
3. 作用域：当前模块中的变量|函数只属于当前这个模块。

解决问题

1. 命名冲突

node中模块化

node中，模块分为两类

- 核心模块

核心模块部分在 Node 源代码的编译过程中，编译进了二进制执行文件。在 Node 进

程启动时，部分核心模块就被直接加载进内存中，所以这部分核心模块引入时，文件定位和 编译执行这两个步骤可以省略掉，并且在路径分析中优先判断，所以它的加载速度是最快的。：如：HTTP 模块、URL 模块、Fs 模块都是 nodejs 内置的核心模块。

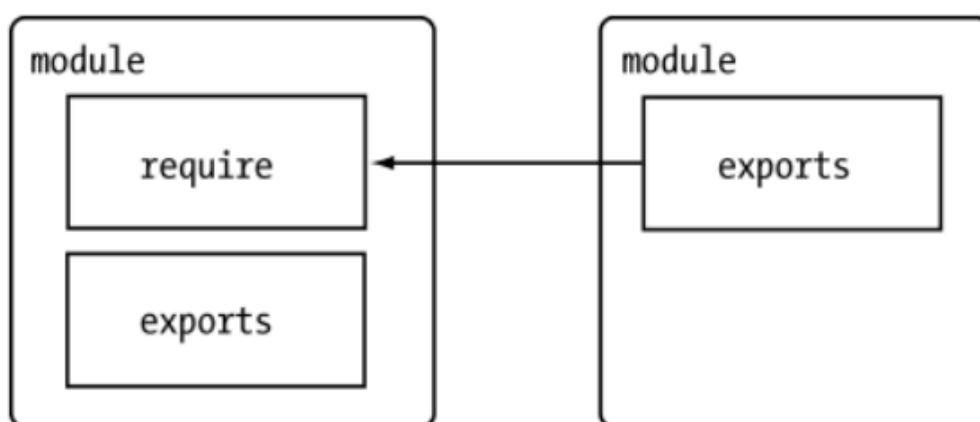
- 文件模块（自定义模块）

文件模块则是在运行时动态加载，需要完整的路径分析、文件定位、编译执行过程、速度相比核心模块稍微慢一些，但是用的非常多。这些模块需要我们自己定义。接下来我们看一下 nodejs

Commonjs中自定义包规定

- 我们可以把公共的功能 **抽离成一个单独js文件** 作为一个模块，默认情况下这个模块下的属性和方法外面是无法访问的，如果想在外部进行访问，就必须在模块内部通过 **** exports 或者 module.exports **** 暴露属性和方法

- 在需要使用这个模块的文件中，通过require的方式引入这个模块。这样就可以利用模块中的属性和方法。



创建模块

```
// 定义模块
let tools = {
  name: 'zhangsan',
  age: 18,
  sayHello: function() {
    console.log('sayhello');
  }
};
// 暴露1
exports.tools = tools;
// 暴露2
module.exports = tools; // 推荐
```



引入模块

```
let tools = require('./tools.js');
console.log(tools);
// name
tools.tools.name
tools.name
```

利用暴露1输出 { tools: { name: 'zhangsan', age: 18, sayHello: [Function: sayHello] } } 利用暴露2输出 { name: 'zhangsan', age: 18, sayHello: [Function: sayHello] }

引入文件规则

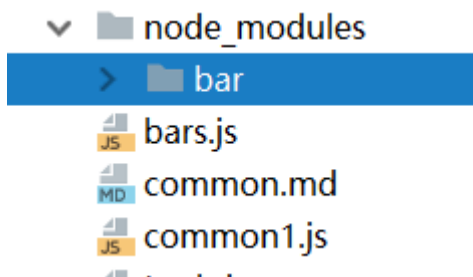
根目录下 (./tools.js) 后缀名可省

 common1.js
 tools.js

;

如果根目录没有，默认到 node_modules 文件夹找

```
// 引入node_modules文件夹下的bar.js
var bar = require('bar.js')
// 等同于
var bar = require('bar')
```



node_modules 文件夹下的 bar 下免得barjs

```
var bar = require('bar/bar');
```

npm init 生成 package.json

上述这种方式显得有些累赘，我们能否将上述文件利用一种简单的方式引入？

- npm init --yes
生成package.json文件