# nodejs

Node 是一个基于 [Chrome V8](https://blog.csdn.net/qq_30638831/article/details/90552912) 的 Javascript 运行平台

### Node.js 与浏览器的区别

构建的应用程序不同

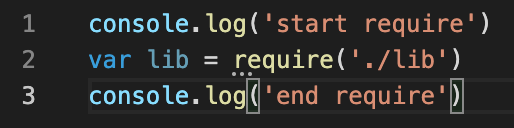
生态系统不同：浏览器具有一系列web API，而node会有文件访问系统等API

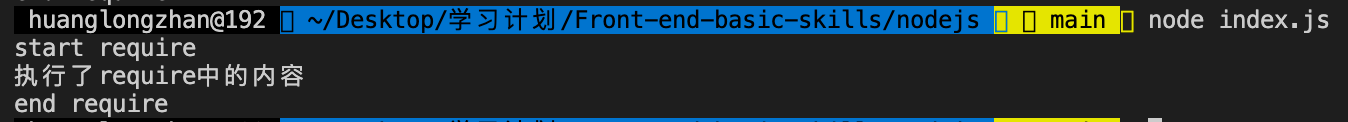
nodejs可以自主控制运行环境，而不必像浏览器一样考虑用户的各种浏览器问题。

## 1、常用的特有变量

console.log(\_\_filename); //当前文件目录+文件名  
console.log(\_\_dirname); //当前文件目录  
  
console.log(process) //进程对象，记载了运行环境的信息

## 2、node支持commonJS规范





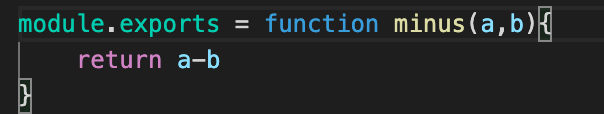
①require会引入对应模块并立即执行模块中的内容

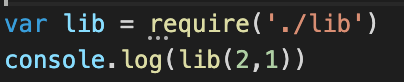
②每个模块如lib.js会注入一个exports变量，默认是{}，可以被require到，且只是一个引用地址

//lib.js  
exports.name = 'laomao'  
console.log('执行了require中的内容')

//index.js  
console.log('start require')  
var lib = require('./lib')  
console.log('end require')  
  
console.log(lib) //{ name: 'laomao' }

③更改外部require到的值，使用module.exports





此时lib是一个函数

### 3、nodejs的非阻塞I/O特性

阻塞io和非阻塞io的区别就在于系统接收输入再到输出期间，能不能接收其他输入。

理解非阻塞io

①确定一个input/output的系统

②思考在io过程中，能不能进行其他io

### 4、回调函数遇到的问题

①回调地狱：臃肿

②异步流程控制：并发的异步难以控制,Promise.all解决

### 5、HTTP。express & koa

### 6、rpc调用

Remote Procedure Call 远程过程调用

#### rpc与ajax的相同点

①都是两个计算机之间的通信

* ajax是浏览器与服务器通信
* rpc是两个服务器之间的通信

②需要约定好数据格式

#### rpc与ajax的不同点

①rpc不一定使用DNS作为寻址服务

rpc一般是在内网范围内的调用，使用特有的寻址服务器来寻址

②rpc在应用层协议一般不使用http协议

rpc一般使用二进制协议，具有更小的数据包体积和更快的编解码速率的优点

③基于tcp或udp

### 7、net建立多路复用的rpc通道

### 8、Node.js 事件触发器

nodejs提供了events模块，events模块提供了EventEmitter 类

const EventEmitter = require('events')  
const eventEmitter = new EventEmitter()  
  
eventEmitter.on('start', () => {  
 console.log('开始')  
 setTimeout(()=>{  
 eventEmitter.emit('start')  
 },100)  
})  
setTimeout(()=>{  
 eventEmitter.emit('start')  
},2000)  
  
  
eventEmitter.once('start',()=>{  
 console.log('start once')  
})

### 9、Node.js 路径模块

path.sep（作为路径段分隔符，在 Windows 上是 \，在 Linux/macOS 上是 /`）和 path.delimiter（作为路径定界符，在 Windows 上是 ;，在 Linux/macOS 上是 :）。

const path = require('path')  
console.log(path.sep) /  
console.log(path.delimiter) ：

##### path.basename()

返回路径的最后一部分。 第二个参数可以过滤掉文件的扩展名

const path = require('path')  
console.log(path.basename('/test/something')) //something  
console.log(path.basename('/demo/demo.txt')) //demo.txt  
console.log(path.basename('/test/something.wdnmd', '.wdnmd')) //something

##### path.dirname()

返回路径的部分

require('path').dirname('/test/something') // /test  
require('path').dirname('/test/something/file.txt') // /test/something

##### path.extname()

返回文件扩展名

require('path').extname('/test/something') // ''  
require('path').extname('/test/something/file.txt') // '.txt'

##### path.join()

连接路径的两个或多个部分

const name = 'joe'  
require('path').join('/', 'users', name, 'notes.txt') //'/users/joe/notes.txt'

##### path.parse()

解析路径为

root: 根路径。

dir: 从根路径开始的文件夹路径。

base: 文件名 + 扩展名

name: 文件名

ext: 文件扩展名

require('path').parse('/users/test.txt')  
  
结果是  
{  
 root: '/',  
 dir: '/users',  
 base: 'test.txt',  
 ext: '.txt',  
 name: 'test'  
}

##### path.resolve()

官网给的例子有:

path.resolve('/foo/bar', './baz');  
// 返回: '/foo/bar/baz'

path.resolve('/foo/bar', '/tmp/file/');  
// 返回: '/tmp/file'

path.resolve('wwwroot', 'static\_files/png/', '../gif/image.gif');  
// 如果当前工作目录为 /home/myself/node，  
// 则返回 '/home/myself/node/wwwroot/static\_files/gif/image.gif'

此时的目录结构：

F:/foo/bar/baz  
F:/bar/baz  
F:/tmp/file  
F:/home/myself/node/wwwroot/static\_files/gif/image.gif  
F:/home/myself/node/wwwroot/static\_files/png

解释：path.resolve('wwwroot', 'static\_files/png/', '../gif/image.gif');

相当于在当前工作目录（ /home/myself/node）执行

①cd wwwroot

②cd static\_files/png/

③cd ../gif/image.gif

最终结果就是'/home/myself/node/wwwroot/static\_files/gif/image.gif'

### npx的优点

##### 轻松地运行本地【node\_modules】命令

开发者想执行命令如webpack，通常将webpack包安装在全局下，以使它们处于路径中且可被立即地执行。这导致无法安装同一命令的不同版本。

而运行 npx commandname 会自动地在项目的 node\_modules 文件夹中找到命令的正确引用，而无需知道确切的路径，也不需要在全局和用户路径中安装软件包。

##### 无需先安装命令即可运行命令

* npx create-react-app my-react-app

上面代码运行时，npx 将create-react-app下载到一个临时目录，使用以后再删除。