NODE

- node官网
- node中文网

- 什么是 node
  - Node.js® is a JavaScript runtime built on Chrome’s V8 JavaScript engine.
  - Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

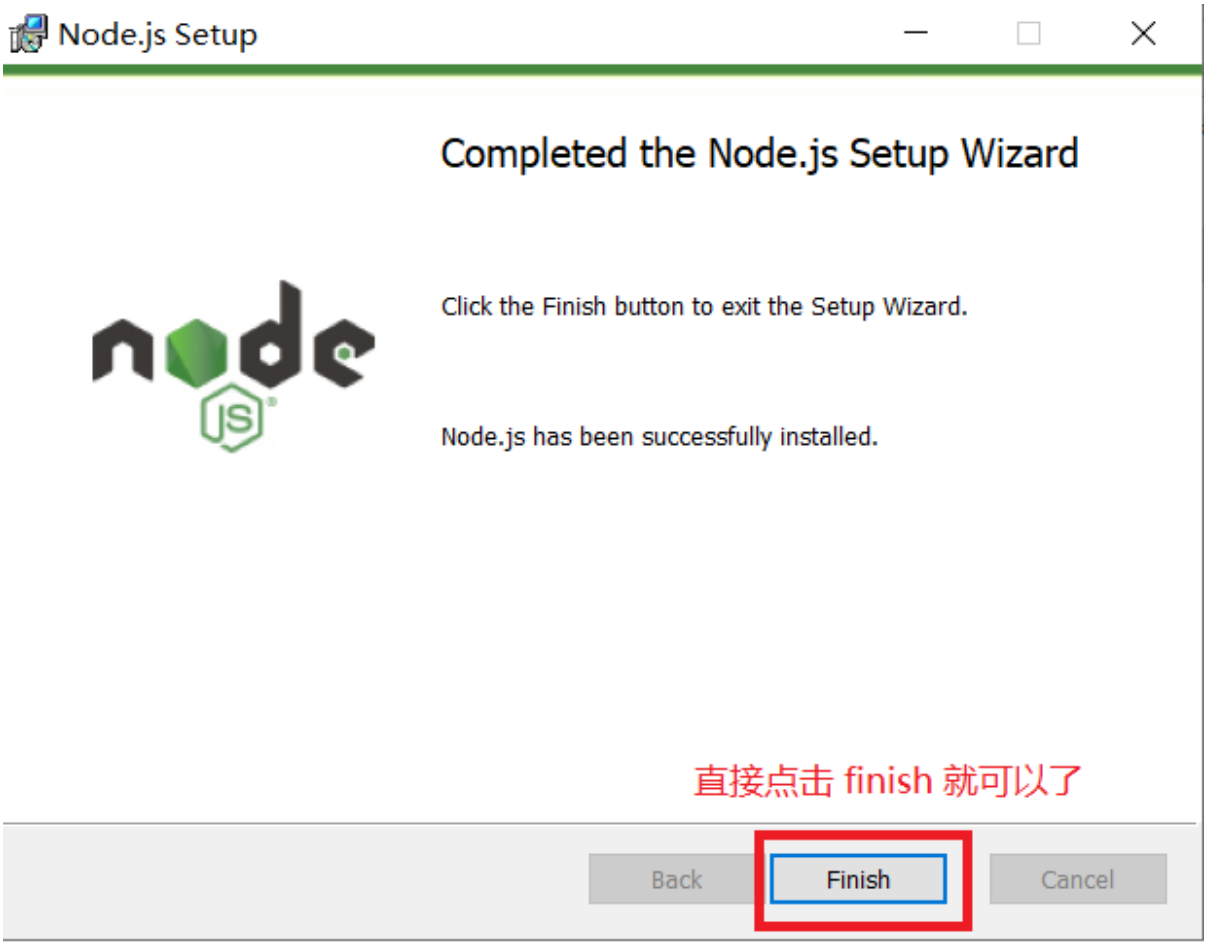
- 这个是 node 官网的解释
  - 其实 node 就是用 javascript 语言写 **后端**
  - 也就是说，我们的 javascript 有了 node 以后，不光是一个前端语言，也是一个后端语言
- 前端 javascript
  - 三大核心
    - ECMAScript
    - DOM
    - BOM
  - 操作内容
    - 浏览器
    - 解决兼容问题
- 后端 javascript （node）
  - 核心
    - ECMAScript
  - 操作内容
    - 后端代码
    - 数据库
- 也就是说，node 我们不需要解决兼容问题，不需要 DOM 和 BOM，只关注业务逻辑就可以了

下载 node 安装包

- 我们的电脑是没有自带 node 环境的
- 需要我们手动下载一个 node 安装包，安装 node 环境
- 有了 node 环境以后，我们就可以运行 node 了
- 下载方式
  - 直接到 node官网 或者 node中文网

- 直接到 node 官网 或者 node 中文网

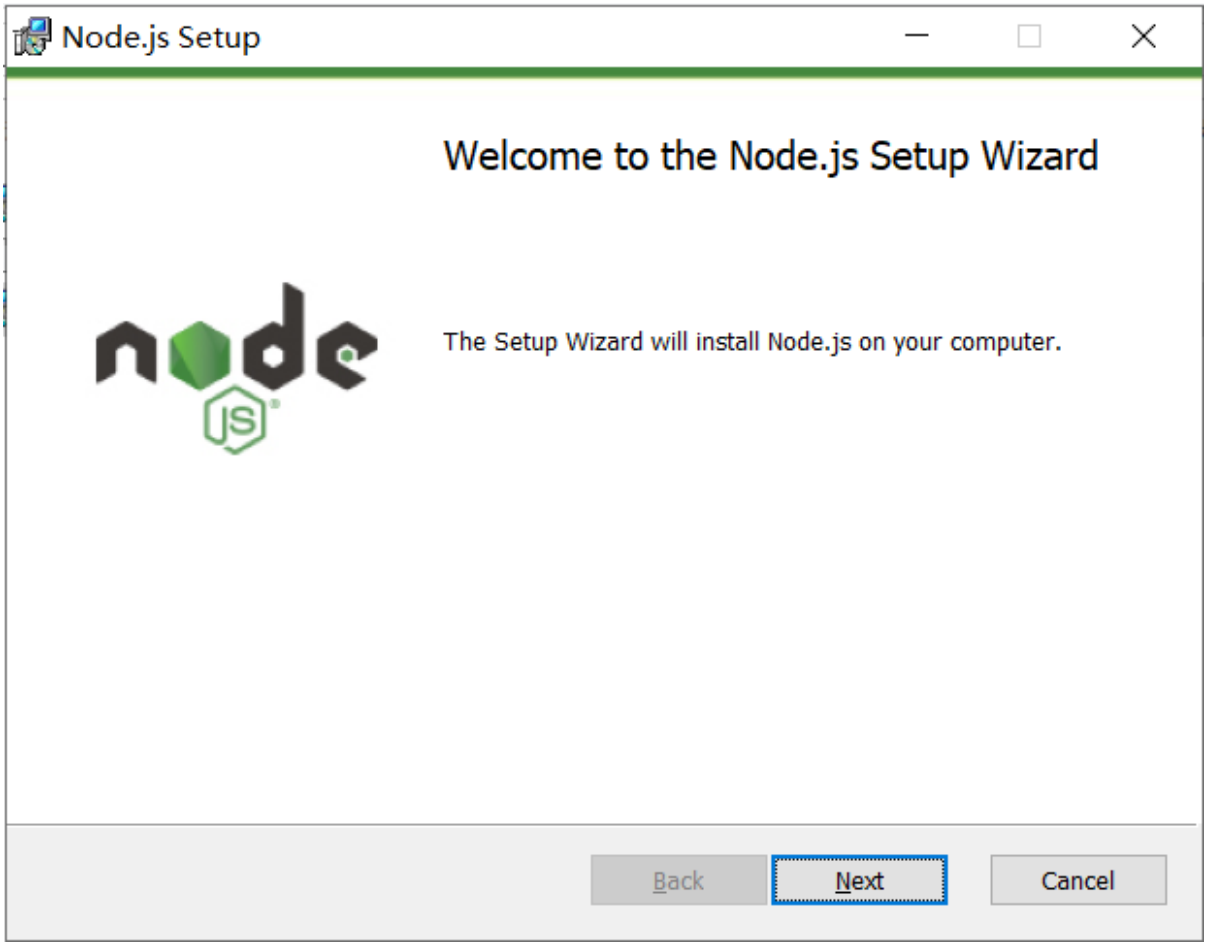
- 点击下载就可以

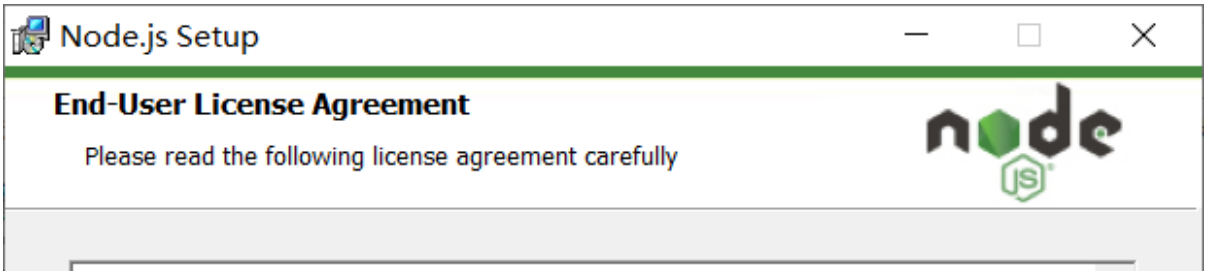


- 注意： **在 node 中文网下载的时候，选择安装包，不要选择 二进制文件**
  - 因为 二进制文件 是一个简单版，我们需要自己配置 **环境变量** 才可以使用

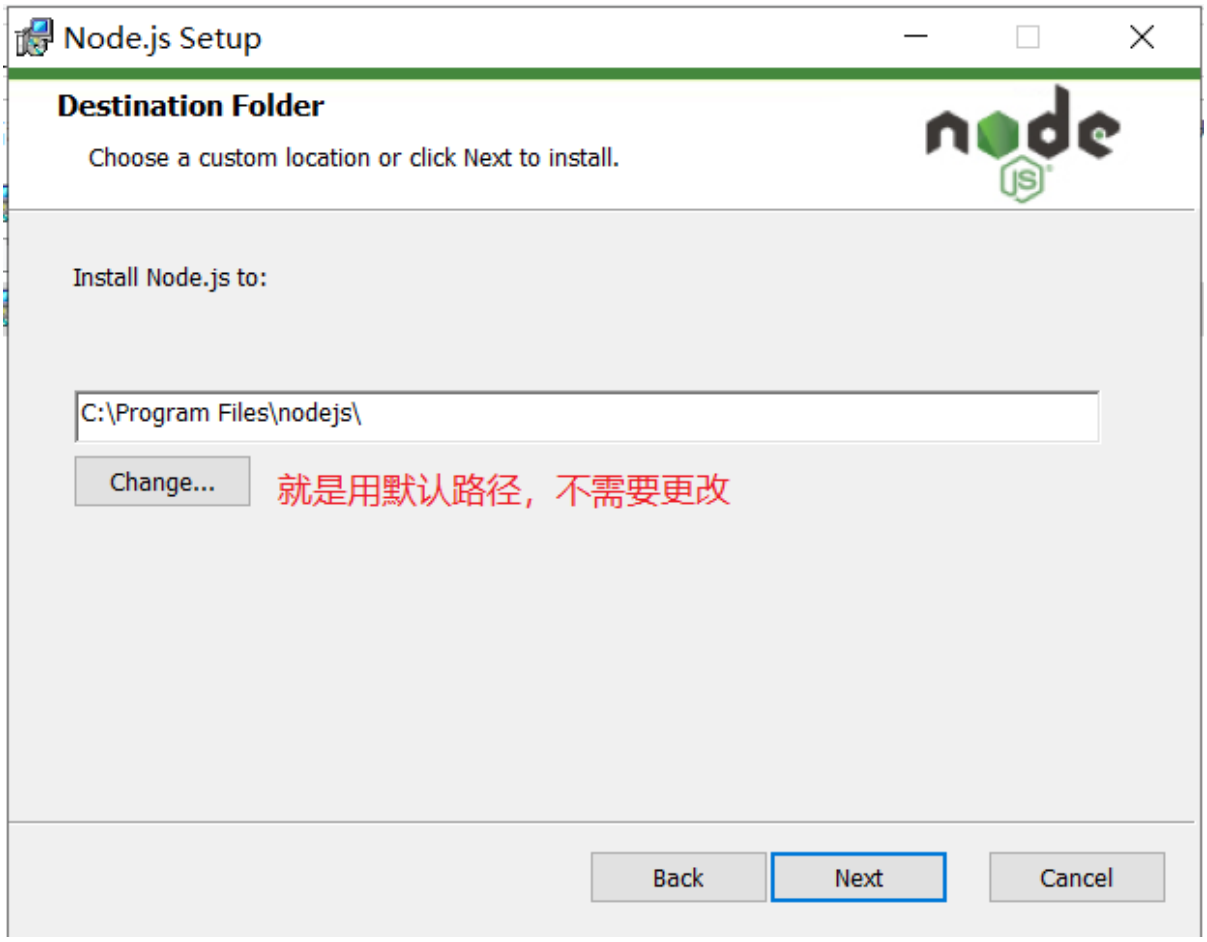
安装 node 环境

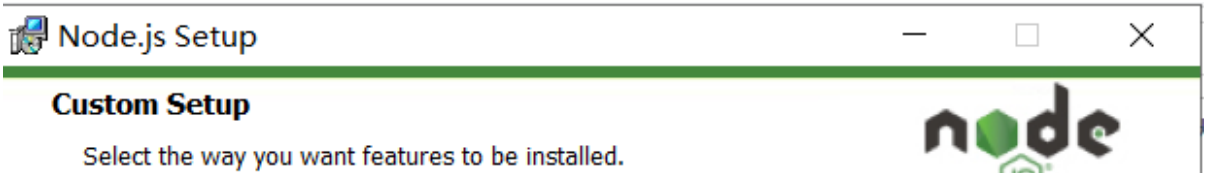
- 下载好以后，我们直接把下载好的文件双击运行就行
- 找到 `node-v10.16.2-x64.msi` 对应的文件

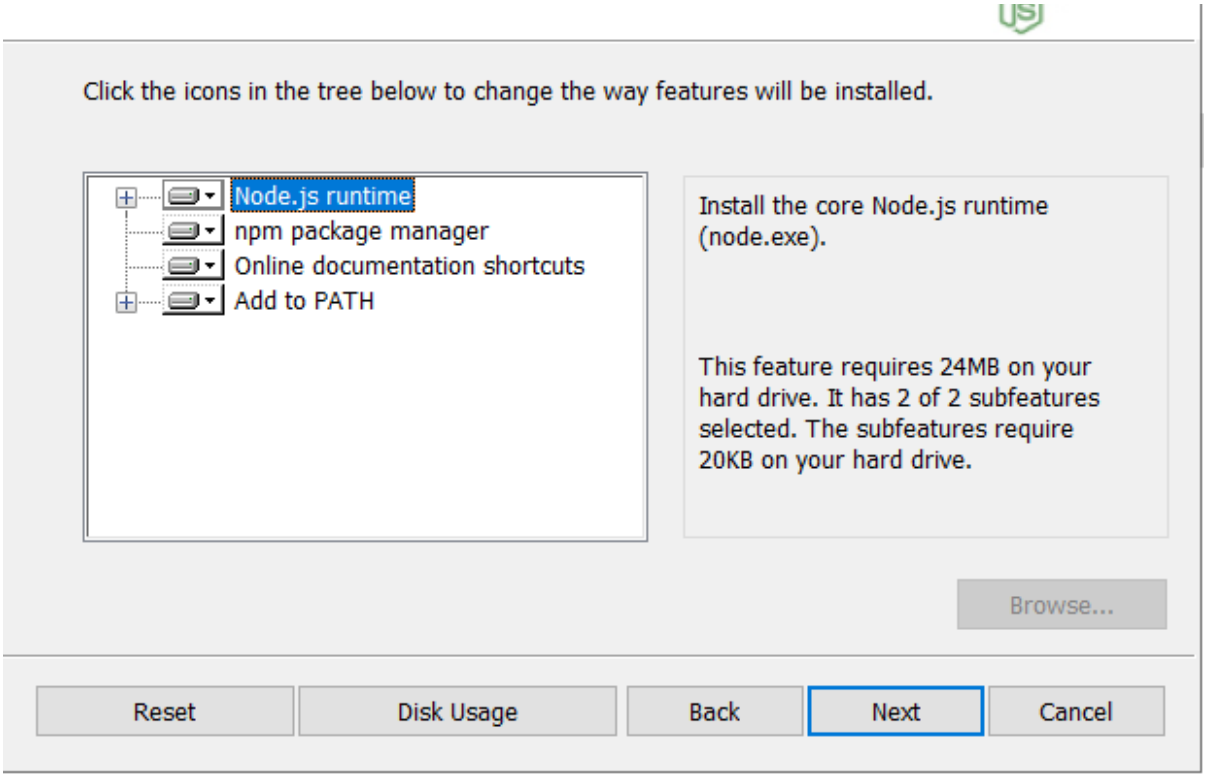


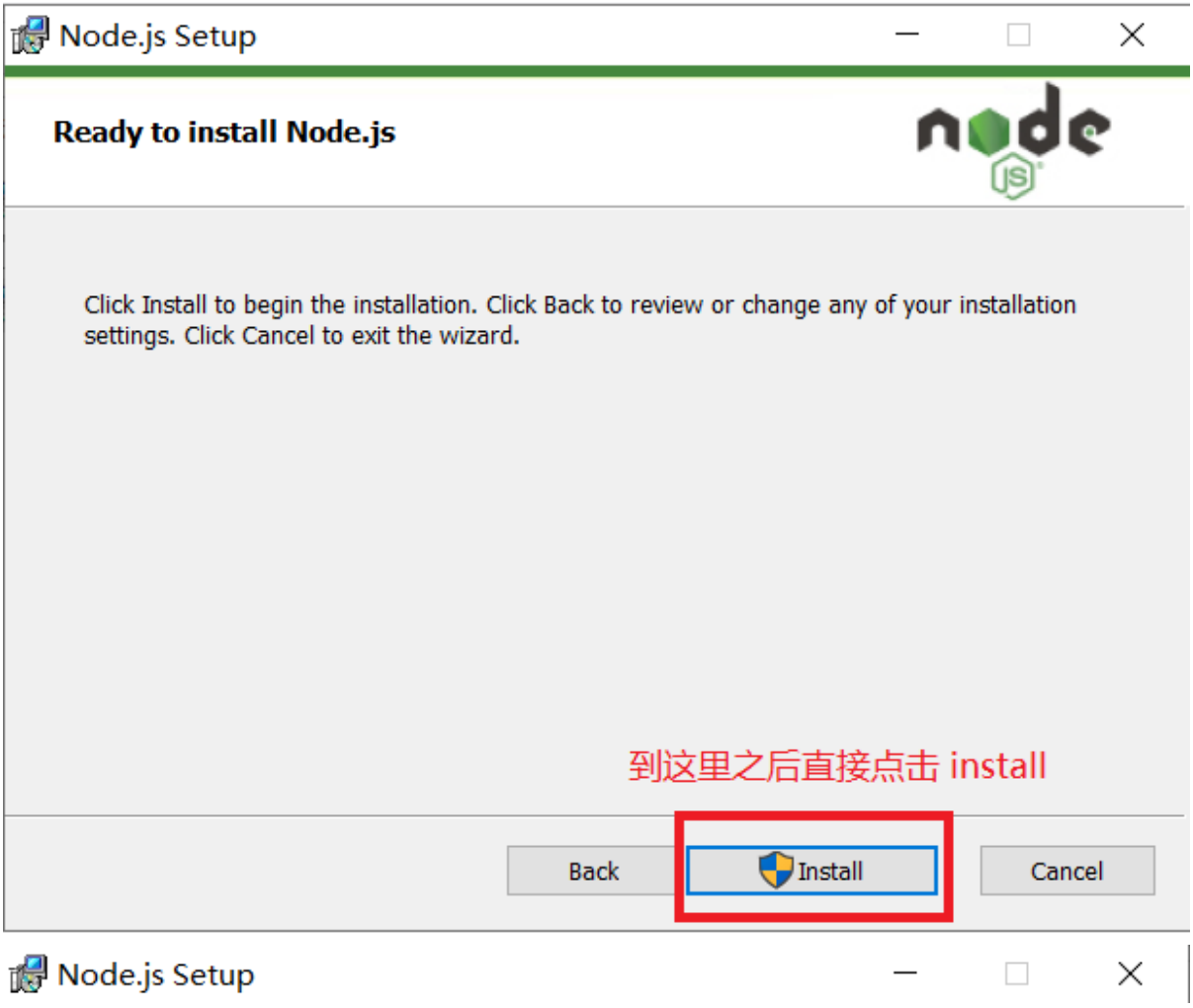


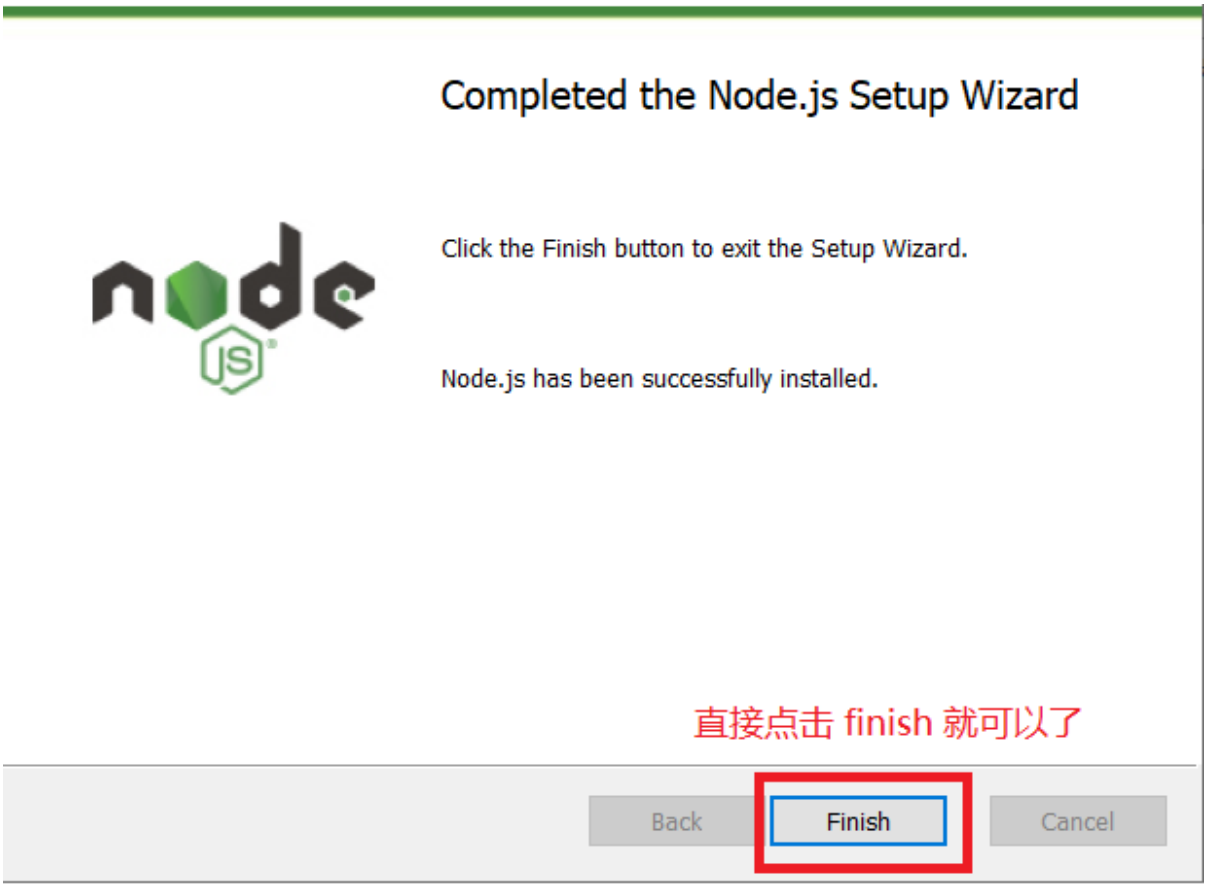

Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,
I accept the terms in the License Agreement
要勾选上
Print
Back
Next
Cancel


Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
就是用默认路径，不需要更改
Back
Next
Cancel


Node.js Setup
Custom Setup
Select the way you want features to be installed.


Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 24MB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 20KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel


Node.js Setup
Ready to install Node.js
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
到这里之后直接点击 install
Back
Install
Cancel
Node.js Setup

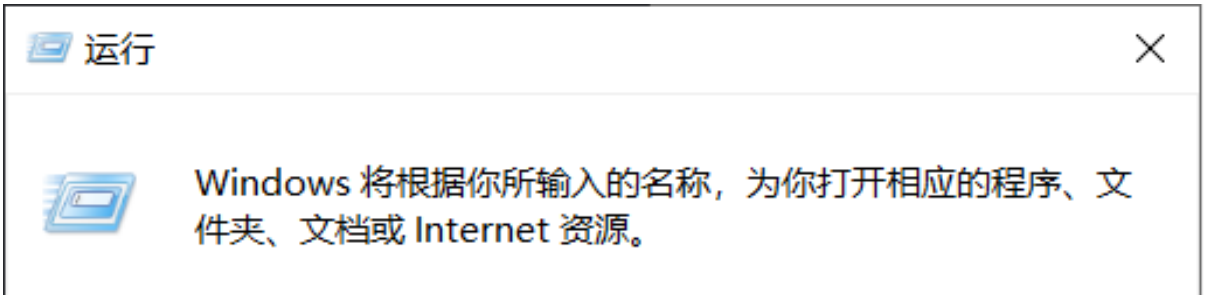


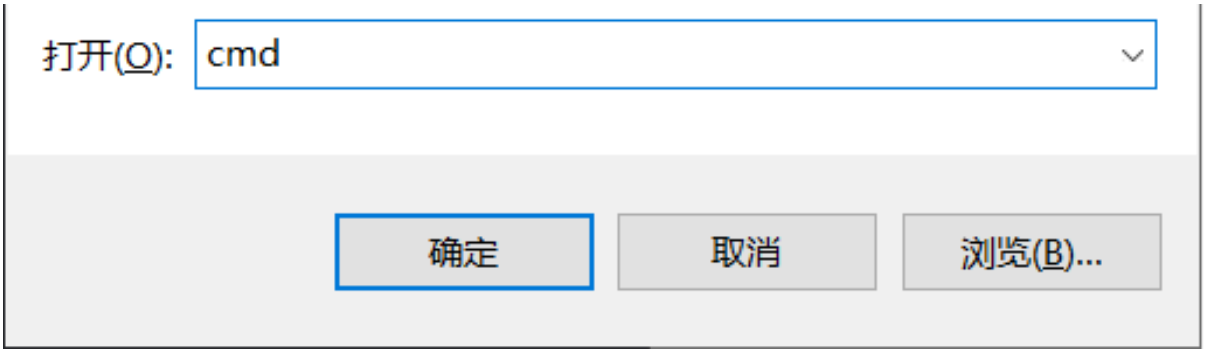
- 这个时候 node 就安装完毕了

检测安装环境

- 检测安装是否成功
- 我们打开运行窗口 （win + r）
    - 就是我们键盘下面那个 windows 的窗口键 + r 键

- 写入 cmd 然后按下回车，来到我们的命令行

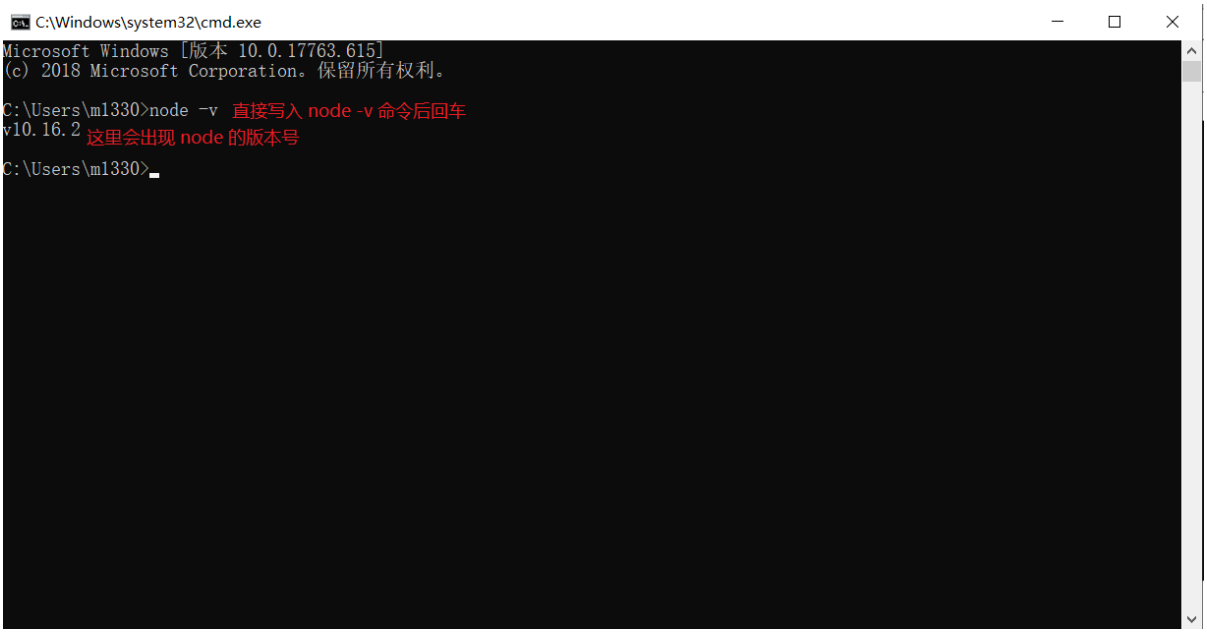




- 然后再命令行写入一个指令

  1. $ node -v

- 然后按下回车，会得到一个 node 的版本号
- 能看到版本号表示 node 环境安装成功



- 至此，我们的 node 环境就有了
- 我们就可以再电脑里面运行我们的 node 了

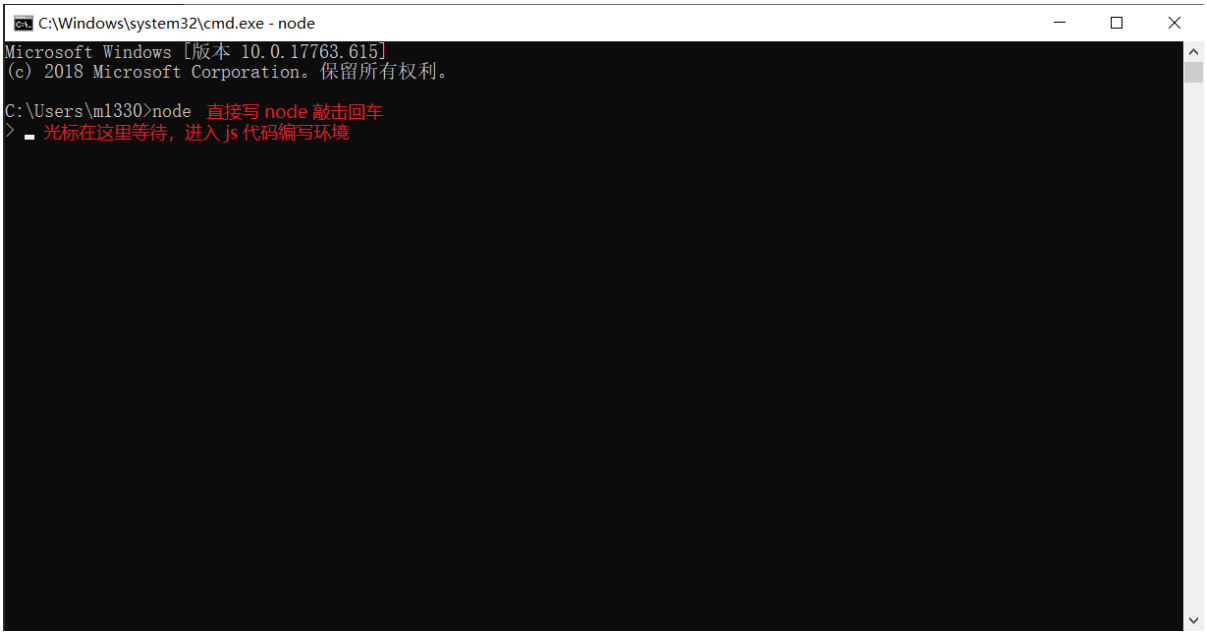
node 初体验

- 到现在，我们的 node 环境已经安装完毕了
- 接下来我们就体验一下 node
- 我们的 node 到底是一个什么东西
  - 就是直接在 终端（命令行） 中运行 `js` 代码

- 就是直接在 终端（命令行）一个运行 `js` 代码
- 也可以用 `.js` 文件写一堆 `js` 代码
- 然后不需要浏览器，直接让我们写的 `js` 代码运行在我们自己电脑的终端上

直接在终端中书写 js 代码

- 打开命令行
- 书写指令
  1. $ node
- 直接按下回车，会看到 光标在闪烁，我们就进入了 node 代码编写环境
- 直接书写代码就可以了

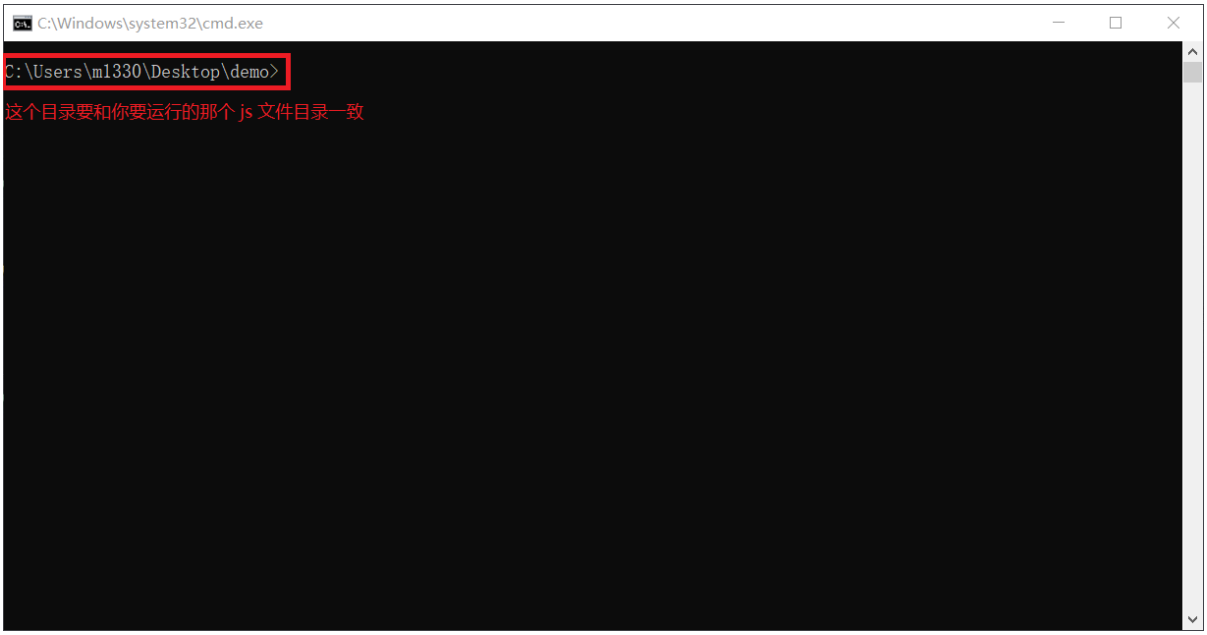


在命令行运行一个 js 文件

- 先新建一个文件夹
- 在里面写一个 js 文件
  - 我这里默认在 `桌面/demo文件夹/idnex.js`
- 在文件里面写一些 js 代码

```
// index.js
console.log('hello node')
```

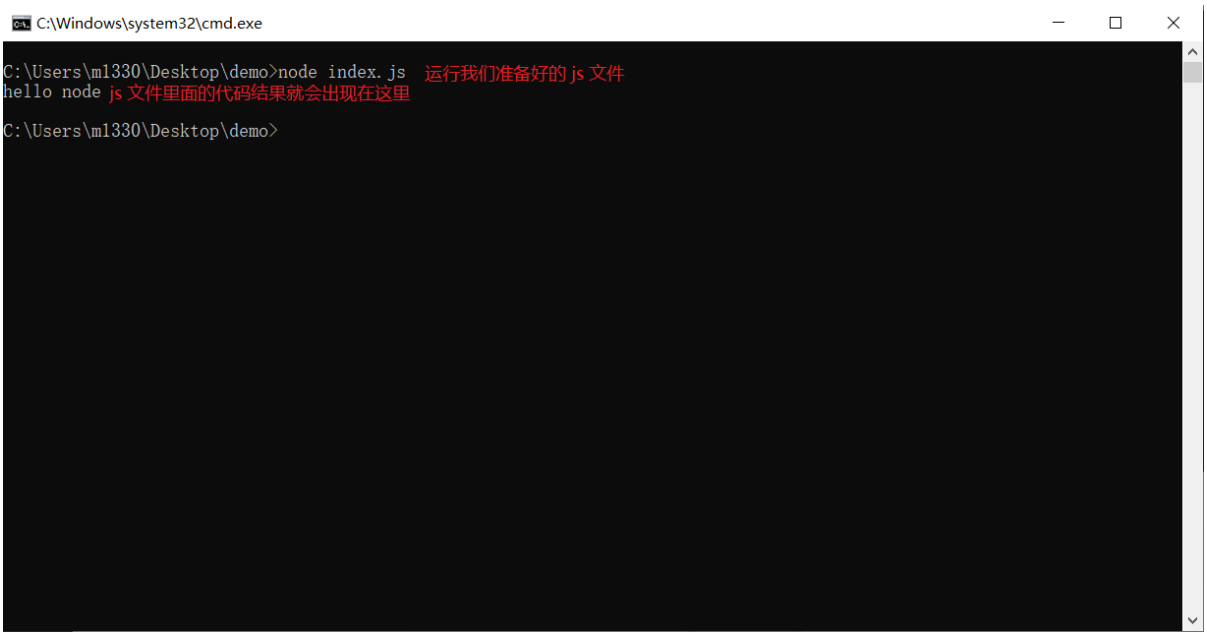
- 打开命令行，**要让命令行的路径和你存放这个要执行的 js 文件的目录一致**



- 切换好以后，我们直接使用指令来运行我们准备好的 js 文件

```
1.  $ node index.js
```

- 然后就会在命令行把我们刚才写的 js 文件运行了
- 就会在控制台输出 `hello node`



- 现在我们就已经运行了一段 js 代码在命令行了

- 这也就解释了一下最开始官网说的那句话
  - Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。
    - 我们的 node 安装完毕以后，就在命令行提供了一个基于 Chrome V8 引擎的运行环境
    - 在这个环境中运行 javascript 代码
    - 这个就是 node.js

常用的 LINUX 操作

- 什么是 LINUX 操作
- 其实就是在命令行使用指令来操作我们的电脑
- 因为我们的 node 就是在命令行运行 js
- 所以我们要了解一些常用的命令行指令

目录操作

- 目录操作就是操作我们的命令行路径

i. 查看当前目录下所有文件

```
$ dir
```

ii. 以树状结构展示当前目录下的所有文件及子目录下的所有文件

```
$ tree
```

iii. 进入当前目录下的某一个目录

```
$ cd 文件夹名称
```

iv. 返回上一级目录

```
$ cd ..
```

v. 切换盘符

```
$ 盘符:
$ d:
```

文件操作

- 文件操作就是通过指令创建文件或者文件夹

i. 创建文件夹

```
# 表示在当前目录下创建一个叫做 test 的文件夹
$ md test
```

ii. 移除文件夹

```
# 表示移除当前文件夹下的 test 文件夹
$ rd test
```

iii. 复制文件夹

```
# 表示复制一份 test 文件夹起名为 test2
$ xcopy test test2
```

iv. 创建文件

```
# 表示在当前目录下创建一个叫做 index.js 的文件
$ type nul> index.js
```

v. 拷贝一份文件

```
# 表示复制一份 index.js 文件起名为 ceshi.js
$ copy index.js ceshi.js
```

vi. 向文本中写入内容

```
# 表示向 index.js 中写入一段文本 console.log('hello world')
$ echo console.log("hello world") > index.js
```

vii. 查看文件内的文本内容

```
# 表示查看 index.js 文件中的文本内容是什么
$ type index.js
```

viii. 给文件或者目录重命名

```
# 表示把 index.js 更名为 abc.js
$ ren index.js abc.js
```

ix. 删除文件

```
# 表示把当前目录下的 index.js 删除
$ del index.js
```

x. 移动文件或文件夹

```
# 表示把当前目录下的 index.js 文件移动到当前目录下的 a 文件夹下
$ move index.js a
```

其他指令

- 做一些其他事情的时候使用的

i. 清屏

```
# 表示把当前屏幕的所有内容都清除
$ cls
```

ii. 查看当前电脑 IP 信息

```
# 表示查看当前电脑的 IP 信息
$ ipconfig
```

iii. 测试某一个链接地址的网速

```
# 表示查看访问 百度 网站的速度
$ ping www.baidu.com
```

iv. 查看电脑信息

```
# 表示查看当前电脑的信息
$ systeminfo
```

NODE 的导入导出

- `node` 的开发是模块化开发
- 每一个 js 文件都是一个独立的模块
- 都有自己独立的作用域
- 我们可以通过 导入导出 的方式把多个 js 文件合并在一起

导入

- 在 node 里面，我们使用 `require` 来导入一个文件

```
// 我是 index.js 文件
```

```
require('./a.js')

console.log('我是 index.js 文件')
```

- 当我在命令行运行 `index.js` 文件的时候
  - 首先会把 `a.js` 文件运行一遍
  - 然后再继续执行我自己文件内部的代码
- 也可以再导入的时候接受另一个文件导出的内容

```
// a 接受到的内容就是 a.js 这个文件导出的内容
// 如果 a.js 文件中什么都没有导出，那么接受到的就是一个 空对象
const a = require('./a.js')
```

导出

- 我们在写一个 js 文件的时候，可以向外导出一些内容
- 将来在这个文件被导入的时候，就可以接受到一些内容

```
// 我是 a.js

// 每一个 js 文件都会有一个对象叫做 module
// 在 module 里面有一个成员，叫做 exports
// 每一个 js 文件会默认把 module.exports 导出
// 也就是说，我们向 module.exports 中添加什么内容
// 那么就会导出什么内容

module.exports.name = 'Jack'
module.exports.age = 18
```

- 将来这个文件被导入的时候，接受到的内容就是一个对象，里面有两个成员

```
// 我是 index.js

const a = require('./a.js')

console.log(a) // { name: 'Jack', age: 18 }
```

模块化

- 在 node 的开发过程中
- 我们是把每一个功能独立做成一个模块
- 然后在使用 导入导出 的方式把他们关联在一起
  - 利于维护
  - 准确定位
- 我们一般把模块分为三种
  i. 内置模块 （node 天生就带有的模块）
  ii. 自定义模块 （我们自己写的文件）
  iii. 第三方模块 （从网上下载的别人写好的模块）

NODE 常用的内置模块

- 刚才是我们自己写的模块
- 现在我们来聊聊常见的内置模块

FS 模块

- `fs` 是 node 的一个内置模块
- 专门用来操作文件的
- 使用的时候直接导入就可以使用了

```
const fs = require('fs')

// 接下来就可以使用 fs 这个变量去操作文件了
```

异步读取文件内容

- 异步的读取某一个文件内的内容

```
const fs = require('fs')

// 因为是异步读取，所以要在回调函数里面获取结果
fs.readFile('./text.txt', 'utf8', function (err, data) {
 // err 表示读取的时候出现的错误
 // data 表示读取到的内容，如果出现错误，那么是 data 是没有内容的
})
```

同步读取文件内容

- 同步读取某一个文件的内容

```
const fs = require('fs')

// 因为是同步读取，所以直接以返回值的形式接收读取的内容就可以
const res = fs.readFileSync('./text.txt', 'utf8')
// 同步读取的时候，如果出错会直接在控制台报错，并中断程序继续执行
// 如果没有错误，res 就会得到文件中的内容
```

异步写入文件

- 异步的向某一个文件中写入内容

```
const fs = require('fs')

// 写入内容的时候，一般不会出现错误
// 因为如果没有这个文件的话，会创建一个这个文件在向里面写入内容
// 所以回调函数一般没什么用处，只不过是在写入文件结束后做些事情而已
// 虽然没有用处，但是必须要写
fs.writeFile('./text.txt', '我是要写入的内容', function () {
 console.log('写入完成')
})
```

同步写入文件

- 同步的向某一个文件内写入内容

```
const fs = require('fs')

// 因为是写入文件
// 没有返回值，因为一般都会写入成功
fs.writeFileSync('./text.txt', '我是要写入的内容')
```

HTTP 模块

- 因为 node 是一个服务端语言
- 所以 node 一定也可以开启一个服务器，开启一个服务
- `http` 这个模块就是专门用来开启服务，并且接受请求，返回响应的

- `http` 也是一个内置模块，直接导入使用就行

```
const http = require('http')

// 接下来就可以使用 http 这个模块去开启服务了
```

创建一个服务

- 要开启先要创建一个服务

```
const http = require('http')

// 创建一个服务
// 这个服务默认监听 http 协议
// 这个服务默认监听 localhost 域名
// 返回值就是这个服务
const server = http.createServer(function (request, response) {
 // 前端发来的每一个请求都会触发这个函数
 // request 包含着所有的请求信息
 // response 是所有的响应信息
})
```

监听一个端口

- 确定这个服务监听哪一个端口

```
const http = require('http')

// 创建一个服务
const server = http.createServer(function (request, response) {
 // 前端发来的每一个请求都会触发这个函数
})

server.listen(8080, function () {
 // 这个函数会在服务开启成功以后执行
 console.log('listening on port 8080')
})
```

给出一个响应

- 简单给出一个响应

```
const http = require('http')

// 创建一个服务
const server = http.createServer(function (request, response) {
 // 前端发来的每一个请求都会触发这个函数
 // 接受到请求以后给出一个响应
 response.end('hello world')
})

server.listen(8080, function () {
 // 这个函数会在服务开启成功以后执行
 console.log('lintening on port 8080')
})
```

- 此时，打开浏览器
- 地址栏输入 `localhost:8080`
- 浏览器就会响应文字 `hello world`

NPM

- 在我们安装 `node` 的环境的时候，会自动帮我们一起安装一个 `npm` 环境
- 就好像我们安装一些软件的时候，会自动在帮我们安装一些什么 `xxx软件管家` / `xxx游戏` 之类的东西
- 但是 `npm` 不是垃圾软件，而是一个我们超级实用的工具

检测是否安装

- 和检测 node 一样
- 在命令行输入指令

```
$ npm -v
```

- 能够得到一个版本号就可以了

了解 npm

- 什么是 npm 呢
- 我们可以把他想象成一个大超市，一个装着所有我们需要用到的 `插件` / `库` / `框架` 的超市
- 我们要下载一个 `jQuery-validation` 插件
  - 我们可以选择去官网进行下载
  - 可以选择去 GitHub 上查找并下载
  - 也可以选择直接在命令行用 npm 下载
- 我们要下载一个 `bootstrap`
  - 我们可以选择去官网进行下载
  - 可以选择去 GitHub 上查找并下载
  - 也可以选择直接在命令行用 npm 下载
- 也就是说，npm 包含着我们所有的第三方的东西
- 我们需要的时候，只要打开终端，就可以使用指令来帮我们下载
  - 再也不需要去官网找了
- 而且，npm 不光可以在后端使用，也可以在前端使用
- **npm 只不过是一个依赖于 node 环境的大型的包管理器**

使用 npm

- 我们想使用 npm 只要打开命令行就可以了
- 作为一个 **包管理器**
- 可以帮我们下载一些 插件 库 框架 之类的东西供我们使用

下载包

- 打开命令行
- 输入下载的指令

```
# 表示使用 npm 这个工具下载一个 jquery
$ npm install jquery
```

- 下载完毕以后，就会在当前目录下多出一个文件夹
  - 叫做 `node_modules`
  - 在这个目录下就会有一个文件夹叫做 `jquery`
  - 就是我们需要的东西了
- npm 的下载默认是下载最新版本的包
- 我们也可以在下载的时候指定一下我要下载哪一个版本

```
# 表示使用 npm 这个工具下载一个 3.3.7 版本的 jquery
$ npm install bootstrap@3.3.7
```

删除包

- 在删除包的时候，我们可以直接去 `node_modules` 文件夹中找到对应的包的文件夹删除掉
- 但是这样做并不好，我们还是应该使用命令行的指令来删除包

```
# 表示我要删除 jquery 这个包
$ npm uninstall jquery
```

- 这样，这个包就会被卸载了

管理项目

- 我们的每一个项目都有可能需要依赖很多的包（有插件/库/框架）
- npm 会帮助我们记录，我们当前这个项目所使用的包
- 但是前提是，你要告诉 npm 说： “你来帮我管理整个文件夹”
- 我们依旧是使用指令在命令行来告诉 npm

```
# 表示告诉 npm 你来帮我们管理整个文件夹（也就是我的整个项目）
$ npm init
```

npm 清除缓存

- 有的时候，有些包下载到一半，因为各种原因失败了（比如突然没有网了）
- 那么这个下载了一半的包 **有可能** 会被缓存下来
- 那么以后你再次下载的时候，就都是失败的状态
- 那么我们就要清除掉缓存以后，在重新下载

```
# 表示清除 npm 的缓存
$ npm cache clear -f
```

NRM

- 我们的 `npm` 虽然好用

- 我们的 `npm` 虽然好用
- 但是有一个缺点
  - 就是，他虽然在帮我们下载东西
  - 但是他的下载地址是在国外
  - 也就是说，每次使用 npm 下载的时候，都是去国外的服务器上进行下载
  - 那么就会有很多不稳定的因素
  - 而且相对时间比较长
- `nrm` 就是一个用来切换 `npm` 下载地址的工具（切换镜像源工具）

安装 NRM

- `nrm` 如果想使用，那么需要我们自己安装一下
- 因为是我们的工具，所以使用 `npm` 就可以安装
- 依旧是使用指令的方式来进行安装
- 只不过这里要把这个 `nrm` 安装成一个全局的依赖，而不再是项目内部的依赖了
  - 全局依赖，一个电脑安装一次，就一直可以使用
- 我们使用指令安装一个全局 `nrm`

```
# 表示安装一个全局 nrm
$ npm install --global nrm
```

检测安装

- 安装完毕之后，我们检测一下是否安装成功
- 和检测 node npm 的时候一样
- 在命令行使用指令查看一下版本号

```
$ nrm --version
```

- 能出现版本号，表示安装成功

使用 nrm

- nrm 里面存着好几个镜像源地址
- 我们要挑一个比较快的使用

检测镜像源地址

- 我们直接在命令行使用指令来查看所有镜像源地址的网速

```
# 表示查看 nrm 镜像源地址网速
$ nrm test
```

切换镜像源

- 我们检测完毕以后，就直到哪个比较快了
- 我们就使用指令切换一下镜像源地址就好了

```
# 表示切换到 taobao 镜像源地址
$ nrm use taobao
```