0.export和import

export和import在es5不支持，所以直接下面的代码直接在浏览器不能运行

0.1 export

模块是独立的文件，该文件内部的所有的变量外部都无法获取。如果希望获取某个变量，必须通过export输出

```
// profile.js
export let firstName = 'Michael';
export let lastName = 'Jackson';
export let year = 1958;
```

或者用更好的方式：用大括号指定要输出的一组变量

```
// profile.js
let firstName = 'Michael';
let lastName = 'Jackson';
let year = 1958;

export {firstName, lastName, year};
```

除了输出变量，还可以输出函数或者类（class），

```
export function multiply(x, y) {
  return x * y;
};
```

还可以批量输出，同样是要包含在大括号里，也可以用as重命名：

```
function v1() { ... }
function v2() { ... }

export {
  v1 as streamV1,
  v2 as streamV2,
  v2 as streamLatestVersion
};
```

export 命令规定的是对外接口，必须与模块内部变量建立一一对应的关系

main.js　import profile.js文件

```
//profile.js
// 写法一
export let m = 1;

// 写法二
let m = 1;
```

```
export {m};

// 写法三
let n = 1;
export {n as m};


// 报错
export 1;

// 报错
let m = 1;
export m;
```

0.2 import

export定义了模块的对外接口后，其他JS文件就可以通过import来加载这个模块，

```
import {firstName, lastName, year} from './profile';  //使用相对路径或者绝对路径

function setName(element) {
  element.textContent = firstName + ' ' + lastName;
}
```

import命令接受一对大括号，里面指定要从其他模块导入的变量名，**必须与被导入模块（`profile.js`）对外接口的名称相同。**

如果想重新给导入的变量一个名字，可以用as关键字，

```
import { lastName as surname } from './profile'
```

import后的from 可以指定需要导入模块的路径名，可以是绝对路径，也可以是相对路径， .js路径可以省略，如果只有模块名，不带有路径，需要有配置文件指定。

注意，`import`命令具有**提升效果**，会提升到整个模块的头部，首先执行。（是在编译阶段执行的）

module的整体加载

除了指定加载某个输出值，还可以用（*）指定一个对象，所有的变量都会加载在这个对象上。

```
// circle.js。输出两个函数

export function area(radius) {
  return Math.PI * radius * radius;
}

export function circumference(radius) {
  return 2 * Math.PI * radius;
}


// main.js 加载在个模块

import { area, circumference } from './circle';
```

```
console.log('圆面积：' + area(4));
console.log('圆周长：' + circumference(14));

//上面写法是逐一指定要加载的方法，整体加载的写法如下。
import * as circle from './circle';

console.log('圆面积：' + circle.area(4));
console.log('圆周长：' + circle.circumference(14));
```

注意，模块整体加载所在的那个对象（上例是`circle`），应该是可以**静态分析的**，所以不允许运行时改变。

```
import * as circle from './circle';

// 下面两行都是不允许的
circle.foo = 'hello';
circle.area = function () {};
```

export default

之前的例子中，使用import导入时，都需要知道模块中所要加载的变量名或函数名，用户可能不想阅读源码，只想直接使用接口，就可以用export default命令，为模块指定输出

```
// export-default.js
export default function () {
  console.log('foo');
}
```

其他模块加载该模块时，`import`命令可以为该匿名函数指定任意名字。

```
// import-default.js
import customName from './export-default';
customName(); // 'foo'
```

export default也可以用于非匿名函数前。

下面比较一下默认输出和正常输出。

```
// 第一组
export default function crc32() { // 输出
  // ...
}

import crc32 from 'crc32'; // 输入


// 第二组
export function crc32() { // 输出
  // ...
};

import {crc32} from 'crc32'; // 输入
```

在一个文件或模块中，export、import可以有多个，export default仅有一个，通过export方式导出，在导入时要加{ }，export default则不需要

1.掌握cli方式搭建项目、安装依赖

1.1 vue-cli 是vue.js的脚手架，用于自动生成vue.js模板工程的。

- 脚手架工具简单讲就是自动将项目需要的环境、依赖等信息都配置好

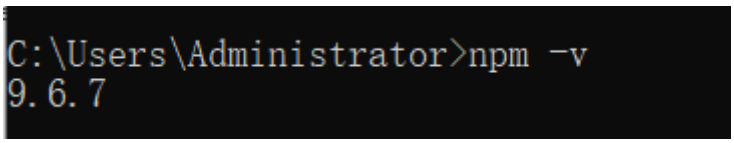
1.2 安装node.js

node下载地址：

```
https://nodejs.org/zh-cn/download
```

Vue CLI 4.x 需要 Node.js v8.9 或更高版本

安装完成输入命令：npm -v



1.3 设置node.js prefix（全局）和cache（缓存）路径（根据自己的电脑磁盘空间大小选择盘符，路径不能用中文或者特殊字符）

设置全局模块存放路径(路径自定义，保证磁盘有足够的空间)

```
npm config set prefix "D:\develop_tool\nodejs\node_global"
```

设置缓存文件夹(路径自定义，保证磁盘有足够的空间)

```
npm config set cache "D:\develop_tool\nodejs\node_cache"
```

设置成功后，之后用命令npm install XXX -g安装以后模块就在D:\develop_tool\nodejs\node_global里

1.4 设置环境变量（非常重要）

说明：设置环境变量可以使得住任意目录下都可以使用cnpm、vue等命令，而不需要输入全路径。

编辑系统变量PATH中添加（**路径：根据步骤3设置prefix的路径进行配置**）

```
D:\develop_tool\nodejs\node_global
```

添加系统变量NODE_PATH,输入**路径：根据步骤3设置prefix的路径进行配置后面加上node_modules**

```
D:\develop_tool\nodejs\node_global\node_modules
```

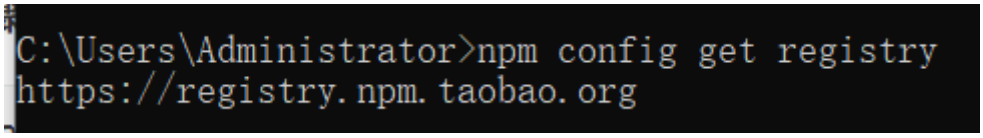
1.5、设置npm镜像为淘宝镜像

输入命令设置淘宝镜像

```
npm config set registry https://registry.npm.taobao.org
```

查看镜像设置是否成功

```
npm config get registry
```

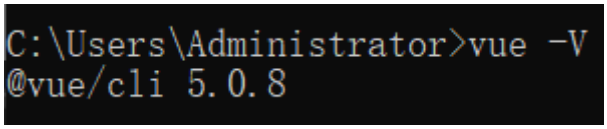


1.6 安装vue命令行工具，即vue-cli 脚手架

```
npm install -g @vue/cli
```

检查vue-cli脚手架版本,如果安装成功可以看到vue-cli脚手架的版本信息

`vue -V`或者`vue --version`

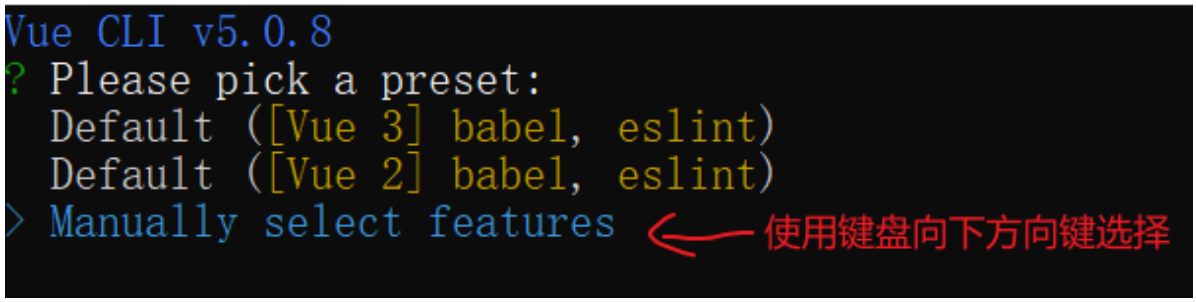


1.7 使用vue-cli来构建vue项目

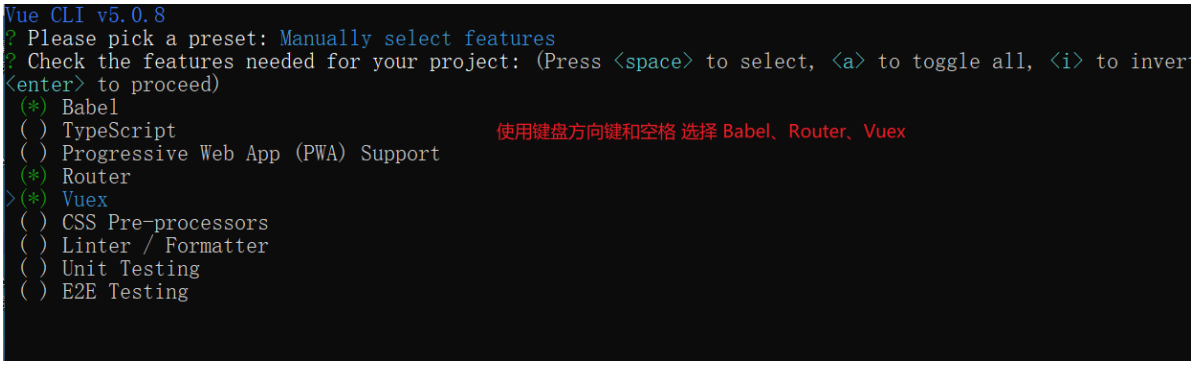
在执行目录下执行命令创建vue模板项目

```
vue create 项目名
```

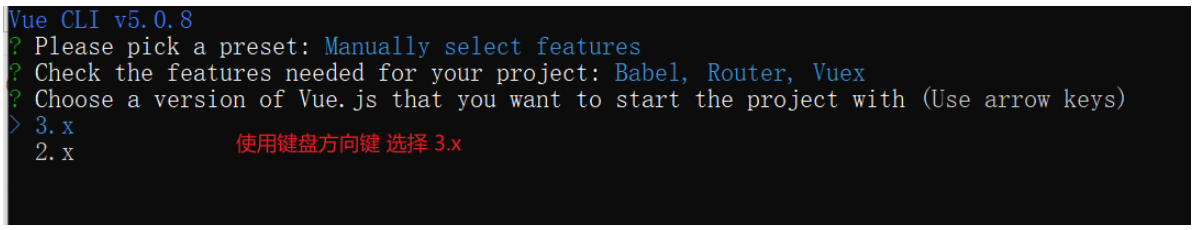
回车



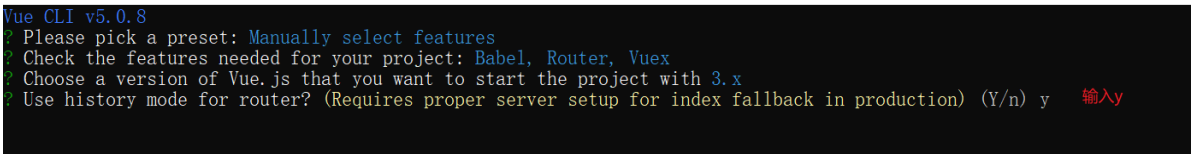
回车



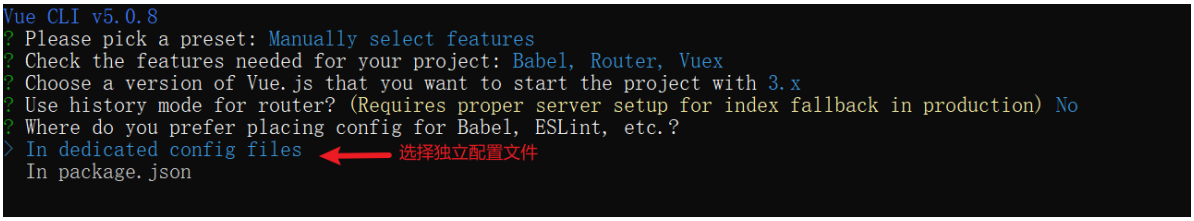
回车



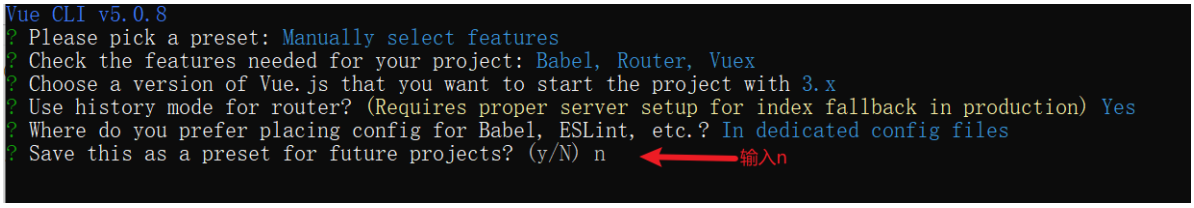
回车



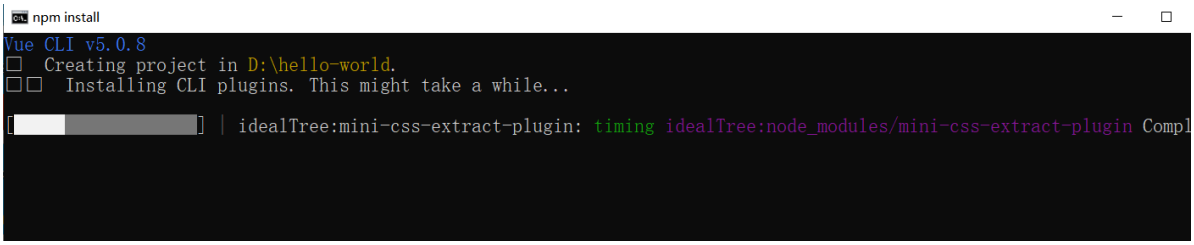
回车



回车



回车，等待下载依赖的js库，需要等一会



创建项目成功

```
Vue CLI v5.0.8
□  Creating project in D:\hello-world.
□□  Installing CLI plugins. This might take a while...


added 844 packages in 3m
   Invoking generators...
   Installing additional dependencies...


added 17 packages in 2m
□  Running completion hooks...

   Generating README.md...

   Successfully created project hello-world.
   Get started with the following commands:

 $ cd hello-world
 $ npm run serve
```

1.8 目录切换到项目目录下

如：cd D:\hello-world

执行 `npm run serve`

出现如下就说明启动成功了

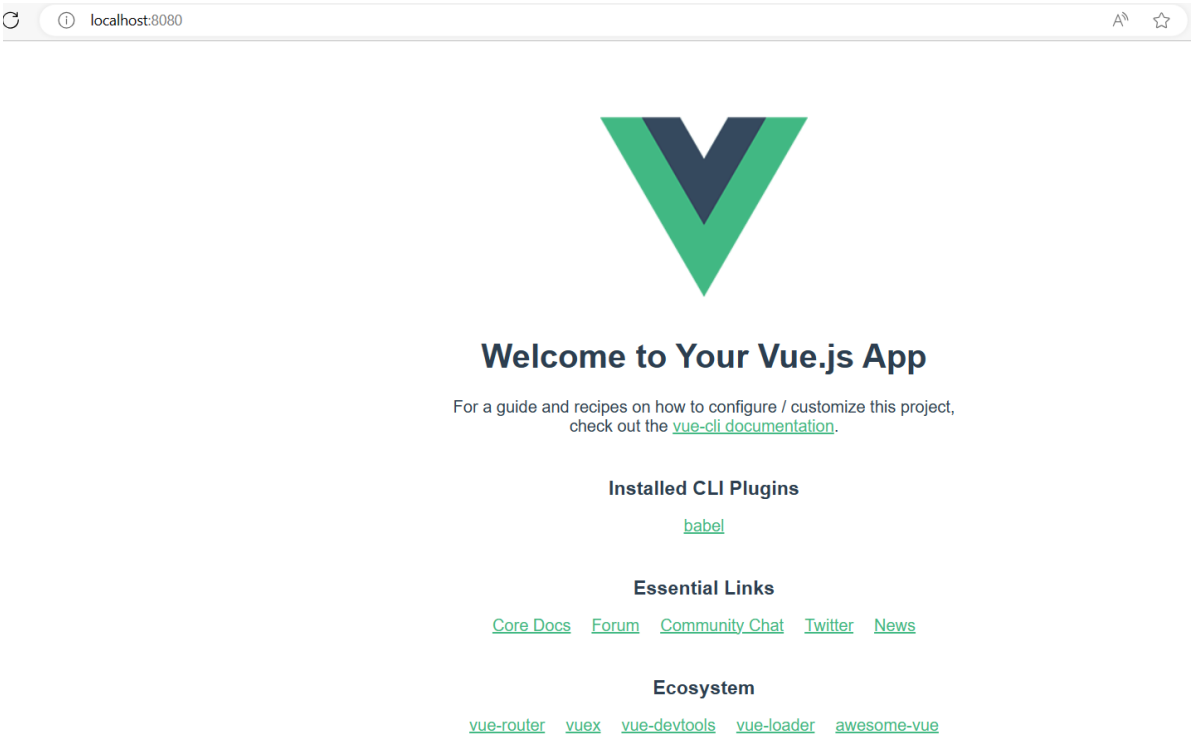
```
DONE  Compiled successfully in 2937ms


App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.1.103:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

项目运行成功后，使用浏览器打开对应页面如下：

http://localhost:8080/



注：这里是默认服务启动的是本地的8080端口，所以请确保你的8080端口不被别的程序所占用。

1.9 项目目录

- node_modules目录：存放项目依赖的第三方库。
- public目录：存放不需要经过webpack处理的静态文件，如index.html、favicon.ico等。
- src目录：存放项目的源代码。
  - assets目录：存放项目的静态资源文件，如图片、字体等。
  - components目录：存放公共组件，可复用的组件。
  - router目录：存放路由相关的文件，包括路由配置和路由守卫等。
  - store目录：存放状态管理相关的文件，包括状态管理的配置和状态管理模块等。
  - views目录：存放视图组件，即页面级别的组件。
  - App.vue文件：根组件，所有的组件都将嵌套在该组件之内。
  - main.js文件：入口文件，创建Vue实例并挂载到DOM节点上。
- babel.config.js文件：Babel编译器的配置文件。
- package.json文件：存放项目的基本信息、依赖和脚本等。
- README.md文件：项目的说明文档。
- vue.config.js文件：配置文件

1.10 加载过程

- 项目启动过程解析

- 启动项目后：变相的通过配置文件加载main.js
- 构建实例：main.js构建了Vue全局实例对象，并声明了App.vue组件。
- 导入组件：App.vue中使用路由插座显示默认路径的视图HellowWorld.vue。
- HellowWorld.vue经过渲染后挂接在当前网页的DOM树上

2.npm、webpack和node关系

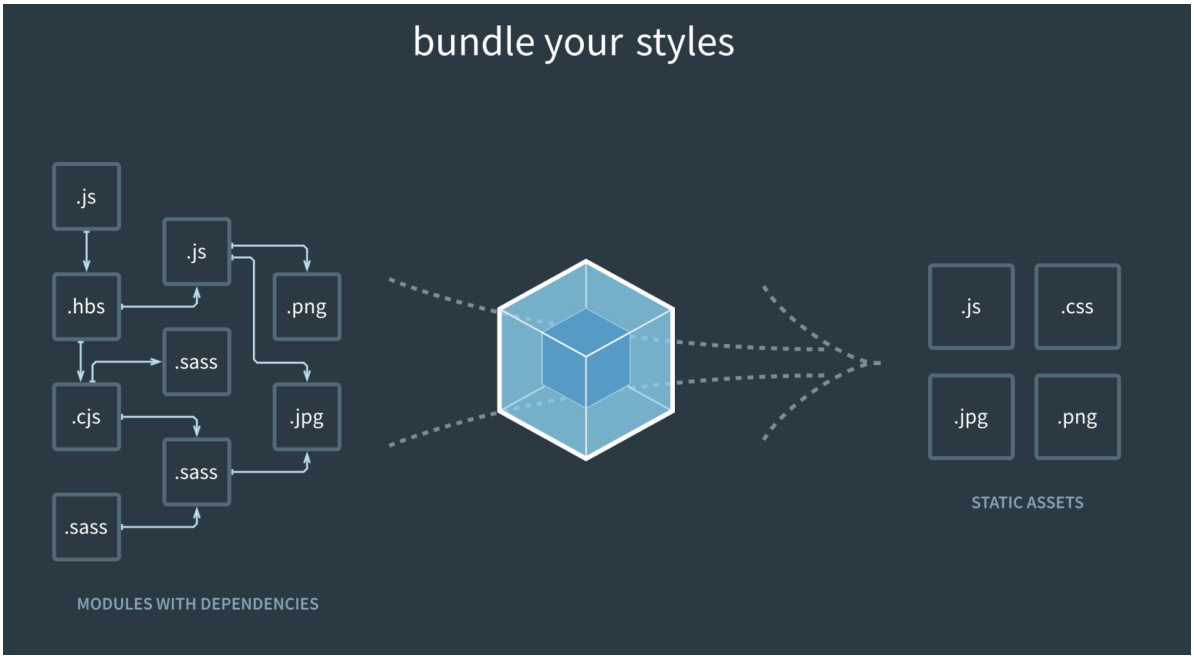
2.1 什么是webpack?

Webpack 是一个前端资源加载/打包工具。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。即**WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。**

2.2 webpack的核心作用

模块化开发中，我们会编写大量模块，如果不打包就上线，那么页面加载或交互时，将会发起大量请求。为了性能优化，需要使用webpack这样的打包器对模块进行打包整合，以减少请求数。就像简单的vue项目，所有组件最终都将被打包到一个app.js中。
相较于无差别打包依赖模块的传统打包器，webpack的核心优势在于它从入口文件出发，递归构建依赖关系图。通过这样的依赖梳理，webpack打包出的bundle不会包含重复或未使用的模块，实现了按需打包，极大的减少了冗余。

如图：



webpack是一个工具，这个工具可以帮你处理好各个包/模块之间的依赖关系（modules with dependencies），并将这些复杂依赖关系的静态文件打包成一个或很少的静态文件，提供给浏览器访问使用；除此之外，webpack因为可以提高兼容性，可以将一些浏览器尚不支持的新特性转换为可以支持格式，进而减少由新特性带来的浏览器的兼容性问题

2.3 什么是npm

介绍了webpack，我们可能会疑问，我的JS，CSS，HTML文件分开写，挺好的呀，为什么要使用webpack工具，进行复杂的各项配置。在传统前端开发模式下，我们确实是按照JS/CSS/HTML文件分开写的模式就可以，但是随着前端的发展，社区的壮大，各种前端的库和框架层出不穷，我们项目中可能会使用很多额外的库，如何有效管理这些引入的库文件是一个大问题，而且我们知道基于在HTML中使用`<script>`引入的方式，有两个弊端，一个是会重复引入，二是当库文件数量很多时管理成为一个大难题。
面对这样的局面，为了简化开发的复杂度，前端社区涌现了很多实践方法。模块化就是其中一项成功实践，而npm就是这样在社区（node）中产生的

npm 由三个独立的部分组成：

- 网站
- 注册表（registry）
- 命令行工具 (CLI)

网站 是开发者查找包（package）、设置参数以及管理 npm 使用体验的主要途径。
注册表 是一个巨大的数据库，保存了每个包（package）的信息。
CLI 通过命令行或终端运行。开发者通过 CLI 与 npm 打交道。

一般来说提起npm有两个含义，一个是说npm官方网站，一个就是说npm包管理工具。npm社区或官网是一个巨大的Node生态系统，社区成员可以随意发布和安装npm生态中的包，也就是不用在重复造轮子了，别人造好了，**你直接安装到你的项目中就可以使用，但是因为前面说了，当包引入数量很多时管理就成为了一个问题，这个就是npm为开发者行了方便之处，npm已经为你做好了依赖和版本的控制，也就是说使用npm可以让你从繁杂的依赖安装和版本冲突中解脱出来，进而关注你的业务而不是库的管理。**

而**webpack就是将你从npm中安装的包打包成更小的浏览器可读的静态资源，这里需要注意的是，webpack只是一个前端的打包工具，打包的是静态资源，和后台没有关系，虽然webpack依赖于node环境**

2.4 什么是node?

实node和nodejs两个概念没有太大差别，唯一的区别就是，人们说起node的时候语境更多的是再说node环境，而说nodejs时更多的是在说node是一门可以提供后端能力的技术。本质上来说，node就是nodejs，nodejs就是node，简单的说 Node.js 就是运行在服务端的 JavaScript。

2.5 webpack npm node之间关系?

- webpack是npm生态中的一个模块，我们可以通过全局安装webpack来使用webpack对项目进行打包；
- webpack的运行依赖于node的环境，没有node是不能打包的，但是webpack打包后的项目本身只是前端静态资源和后台没有关系，也就是说不依赖与于node，只要有后台能力的都可以部署项目
- npm是于Node社区中产生的，是nodejs的官方包管理工具，当你下载安装好node的时候，npm cli 就自动安装好了
- 正是因为npm的包管理，使得项目可以模块化的开发，而模块化的开发带来的这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别，而手动处理又是非常繁琐的，这就是webpack工具存在的意义