1-1课程导学

同学们，大家好！又和大家见面了。这次给大家带来的是一套,webpack4的课程。随着前端工程越来越复杂。单独建几个文件，来写业务代码。这样的方式已经无法保证，项目的可维护性了，所以我们就想，把不同的业务逻辑拆成模块，然后去，分开引入这些模块。每个模块自己做自己的事情。这样就可以保证项目的维护性，和，可扩展性了。但假如，你有几千个模块，你能在页面上引入几千个JS文件吗？所以当项目大到这个程度的时候，我们就嘚借助工具，来管理我们的这些模块了。

webpack就是这样的一种工具,帮助我们管理复杂项目的一个工具。你可以先，暂时这么理解。与webpack类似还有很多相似的工具。比如说gulp，grunt，browserify，但是这两年webpack的使用率直线上升。比如说VUE，React，Angular三大框架的脚手架工具，都开始使用webpack来做底层代码的构建。这得益于webpack可以提供，其他工具无法实现的一些特性。比如说Tree shaking，代码懒加载，codes spling也就是代码分割。当然这些内容呢，我们在后面都会细讲。所以现在来看，用webpack，做代码管理的工具，已经是越来越多前端开发者的共识了，如果你不会webpack，你可能觉得，我写业务代码也没有任何的问题啊，但是如果你学会了webpack，你会从更深一层，去思考前端工程化上面的内容。也能发现在代码的背后其实别有一番天地。

学习webpack，将极大的扩充你的前端开发视野，所以我非常推荐大家来学习一下webpack，webpack4相对于之前的版本，有了比较多的变化。它的速度更快了，在大型项目的构建过程种，webpack4甚至可以节约百分之九十以上的构建时间，同时webpack4内置了更多的默认配置，也变更了很多之前的API，这门课程我希望通过详细的讲解，带大家全面掌握webpack4，同时，这门课也不局限于webpack4这个版本，在课程里，我会详细给大家讲解webpack官方文档的查阅方式。这样即便，未来版本升级，也可以迅速的跟上节奏。

下面我来给大家介绍一下整个课程的章节安排。首先，第一章：就是我们现在的课程介绍，第二章，初识webpack，我们来讲解webpack是什么？它的产生背景以及它的基础使用。第三章，webpack核心概念。这里面我会给大家讲解清楚loader， plugin，sourceMap，HMR这样的一些基础概念。

第四章，webpack进阶。这一章，我们将讲解，诸如：TreeShaking，代码分割，懒加载，缓存，shimming这样的一些稍微复杂的概念。

第五章，webpack实战配置案例，这一章，我们结合之前的基础知识，给大家讲解如何打包一个库。如何打包P W A项目，如何在项目中配置typeScript，ESlint。如何打包多页应用？这样的一些实战常用配置项。

第六章， webpack底层原理及脚手架工具分析。

这章我将教大家，如何编写自己的loader和plugin。同时，我们还会通过自己不到一百行的代码，实现一个类似webpack的打包器。当然，应大家的建议。在最后，我还给大家分析了一波create-react-app和vue-cli 3.0，这样的脚手架工具中的webpack配置。

如果把课程的知识点打散，大家可以看到，这就是整个课程中，我们涉及到的所有知识点。内容虽然多，光背单词，好像对很多同学来说都有难度，但我相信大家，只要认真的跟下来。一定可以掌握这些知识点。

好，下面我们一起来看一下，学习这门课程我们之前，要有什么样的知识储备，我个人认为啊，webpack的课程应该是一门高级的前端课程，虽然我努力的提升讲解质量。降低大家的学习难度。但，要想学习这门课程，HTML，js，css这样的基础前端知识，大家还是必须要掌握的。

在此基础上了，你还需要对node.js有一点了解。至少你要使用过node中的一些核心模块。同时学习这门课程，还需要大家或多或少，对webpack有一点了解。如果在开发中，你从来没有用过这样的打包工具，或者压根就没听说过webpack，那么学习起来你可能，也会有一点吃力。好！假如你的前端基础没问题，或多或少的了解一点儿node.js。和webpack的内容，那么就可以过来和我一起学习这门课程了，

这套课程的讲授方式，我会本着下面这个思路，首先通过通俗易懂的例子，来给大家讲解基础知识，然后通过实战例子，带大家巩固基础知识，所有的代码呢，我都是全部逐行手写完成的，这样避免大家的思维跳跃，最后课程的难度循序渐进，由浅入深，我们先学使用，再讲原理，通过这门课程的学习，大家可以彻底学会webpack配置项的使用，同时也可以理解webpack的作用。以及它的底层原理。当然在公司里面，上手项目的打包过程配置，也是不会有问题的。同时你会拥有工程化的前端思维，步入前端高级工程师的行列，如果大家对学习前端很感兴趣，想进一步学习前端技能，拿到高薪，可以关注我，我会经常分享前端的一些内容，大家如果有兴趣，可以持续学习，进一步提升技能。好！那么接下来我就正式带大家开始这门课程的学习，大家加油！

2-1很久以前是如何编写前端代码的

这节课开始，我来给大家讲解webpack这门课程。对于从事前端开发工作的同学们来说，webpack大家肯定都听说过。我想，很多同学呢，对webpack也有一点点的了解。但是webpack究竟是什么？它产生的背景，以及它能解决的问题是什么呢？我想有一部分同学。就不是很清楚了，所以第一节课，我就带大家，来搞清楚，这个最基础的问题。Webpack究竟是什么？在很久很久以前，当我们写一个web网页的时候，一般是这样来做的。

首先创建一个session文件夹,在这个文件里，我们创建一个index.html文件，和index.js文件。大家可以跟着我一起来操作。…….在这里，我给它初始化一个HTML内容。然后呢，一般来说，我们的一个网页文件，会有对应的HTML标题。我们可以写一个“这是最原始的网页开发”。

body里面会有网页内同，比如说，

<p>这是我们的网页内容</p>

再加一个div标签。<div id = “root”></div>

OK，我们的一个网页不仅仅只有内容。还应该包含一些样式和js内容。样式呢我们可以在head里面写一些。 这里样式我就暂时先省略。那些业务逻辑呢，我们可以写在js里面。

在这里我们增加一个js标签 script。

<script src=”./index.js”></script>

好，src呢，让他引入一个当前目录下的index.js这样的一个文件。保存一下，然后我们运行一下这个网页。这是我们的网页内容。就输出出来了

好，接下来我们在js文件里面写一些业务逻辑。我就写一些非常简单的逻辑。首先我们，定义一个变量， var dom = document.getElementById(‘root’); 接下来，我们向这个DOM节点上添加一些内容。

首先我们创建一个元素 var header = document.createElement(‘div’); 然后header.innerText = ‘header’. 之后呢我们把这个header append这个dom元素里。好， 我们回到网页，看一下效果。我们看到这个header的div,就被挂载到页面上。好，一个网页通常会有头部header区域, setBar区域,以及 content内容区域，于是我们可以继续写。

var sidebar = document.createElement(‘div’);

sidebar.innerText = ‘sideBar’;

dom.append(sideBar)

同理：

var content = document.createElement(‘div’);

content.innerText = ‘content’;

dom.append(sideBar)

好！保存，我们看下效果。这样的话，我们页面的header, sidebar, content三个区域，就由js渲染出来。很久很久之前呢，大家知道，js能实现的逻辑相对而言是比较弱的。所以呢，可能，我们只是简简单单的写一些非常基本的js代码。随着前端技术的发展。Js能实现的代码内容，越来越多。所以呢，大家可以想一下，我们现在就不仅仅需要创建这个header, sidebar，content了。我们可能在里面，增加非常非常多的逻辑。于是我们就发现，如果，通过面向过程的方式，去写我们的代码，我们的代码就会变得非常的长，很多同学呢，可能对面向对象，本身就不太熟。其实，你现在写的代码，打开来看，如果是按照面向过程这种方式来写的话，所有的Js逻辑，都堆在一个文件里。这个文件会堆得越来越大，越来越大，最终变得，不可维护。

这个时候，就出现了一种新的编程方式，面向对象编程。那么，如果我们用面向对象的写法，来写这段代码，该怎么来写呢？

2-2 webpack 究竟是什么？

好，同学们！上一节，我们讲了面向过程，是如何开发的，接着上一节课，我们继续讲。我们说页面有三个部分，分别是header，sidebar，content。那我呢，就可以把这三个部分，按照三个对象来写。所以我呢，就创建三个文件，（……创建文件）大家来看，分别是header.js，sidebar.js，和content.js。

然后，我们来写header.js里面的代码，那，面向对象的方式，我们该怎么来写header.js的代码呢？我们先打开index.js，大家来看，index.js里面，写了这样一段代码。在header里，代码就要这样写。我们首先定义一个构造函数，function Header() {} 其实呢，它就是一个类，在构造函数里面呢，我们再去做，这个创建DOM的逻辑，我们把这块，都放在构造函数里。（…停顿）

function Header() {

var dom = document.getElementById(‘root’);

var header = document.createElement(‘div’);

header.innerText = ‘header’;

dom.append(header);

}

OK！

这样的话header.js，只负责header的逻辑，

同样的，我们再去写sidebar的逻辑，记住啊！首字母要大写，因为在创建一个构造函数，或者定义一个类的时候。类的首字母，一般都是大写的。

(写)

function SideBar() {

var dom = document.getElementById('root');

var sideBar = document.createElement('div');

sideBar.innerText = 'sideBar';

dom.append(sideBar);

}

这样sideBar.js只负责sideBar里面的逻辑。

好！继续写，content.js。我想大家，也知道该怎么写了，这里，再写一下 content.js。

function Content() {

var dom = document.getElementById('root');

var content = document.createElement('div');

content.innerText = 'content';

dom.append(content);

}

这样的话，我们就有了，三个类。在index.html里面。我们把这三个js，分别都引入进来。

<script src=’./header.js’></script>

<script src=’./sideBar.js’></script>

<script src=’./content.js’></script>

<script src=’./index.js’></script>

我们知道，真正的业务逻辑，都是在index.js里面执行的。那么index.js文件应该怎么写呢？

其实很简单，在执行index.js的时候，创建header， sidebar，Content就可以了。好！我们来写一下。

new Header();

new sideBar();

new Content();

保存一下！然后回到页面，刷新一下！我们来看一下，没有问题，页面又是正常的！这样的话，我们就通过面向对象的方式，改造了我们的代码。

使用面向对象，可以使我们的代码，更具有维护性。为什么呢？因为啊！

关于header部门，的逻辑，全部写在Header这个对象里。

关于sidebar部分，的逻辑，全部写在SideBar这个对象里。

关于content部分，的逻辑，全部写在content这个对象里。

这样的话，每个对象，它所承载的职责，是固定的。那么维护起来就方便的多。比如说以前，如果header有问题了，你需要到index.js里面，整篇的去查找，header到底哪一块出了问题。但现在，如果是，这么写代码。Header一旦出了问题，你只需要去打开header.js去查错，就可以了。所以，面向对象，有非常大的优势。

但是，如果我们把代码拆成这个样子，它就带来了新的问题，

首先：大家会发现，在一个index.html里面。引入了，多个js文件。这样的话，整个页面的加载速度，就会变慢，大家想一下，是不是这样的。以前，我们只需要，加载一个js文件就行了。现在加载了四个js文件。这就会导致页面，多了三个http请求。那页面的加载速度，当然会变慢。

第二点，当打开index.js的时候，会发现index.Js，需要创建Header(),SideBar()， Content()；但是，并不能，直接从这个代码中，看出Header这个类，它对应的文件是哪个？Sidebar对应的文件在哪里？如果你想知道，它对应的文件在哪里的话。你必须要，再回头来看，这个HTML文件。你才会发现原来header.js和index.html在同一个目录下。它是这样的，一个文件层级关系。所以第二点就是从代码中，你看不出文件之间的，位置的相互关系。

第三点：也很难，去查错。举个例子：假设，我们把这个content.js位置放错了，放在了index.js的下面了。

原来：

<script src=’./header.js’></script>

<script src=’./sideBar.js’></script>

<script src=’./content.js’></script>

<script src=’./index.js’></script>

现在：

<script src=’./header.js’></script>

<script src=’./sideBar.js’></script>

<script src=’./index.js’></script>

<script src=’./content.js’></script>

好！我们保存一下，再来打开我们的网页。刷新一下。打开控制台！看到报错了，说：Content is not defined。然后呢，我们点这里进去会发现。new Content()这句语法出了错误。但是，如果对一个小白来说，看到这个报错的时候，你是很难联想到，原来问题不是在new Content()这里。而是你引入js的顺序上面出了问题。

所以，如果我们，这么去写，面向对象编程的时候。另一个巨大的问题就是，代码变得不容易维护了。

那！为了解决这个问题，有人就想出了，这样的一个方式。不在html里面引入这么多的js。我们先把它给删掉，而是在index.js里面，这么引入header.js，sideBar.js，content.js。(这个时候要把index.html中，把引入的header.js，sideBar.js和content.js删掉)

import Header from ‘./header.js’;

import SideBar from ‘./sidebar.js’;

import Content from ‘./content.js’;

new Header();

new SideBar();

new Content();

我们把它称之为：

// ES Module模块的引入方式

好！接下来在header.js中，把Header这个类输出出去， export default Header;

在sideBar.js中，把sideBar这个类给输出出去。 export default SideBar;

同理,在content.js中，把Content.js输出出去。 export default Content;

好，大家来看。如果代码这么来写，而且能在网页中运行的话，是不是就能解决，我们之前的几个问题呢？

首先，在index.html里面，只引入了一个js文件。它的网页运行速度会比较快。

第二点，文件和文件之间的依赖关系，非常的明确。你可以看到index.js里面，引入了当前目录下的header.js，当前目录下的sideBar.js，当前目录下的content.js。他们的目录结构，在这儿一目了然。

第三点：如果你去颠倒顺序，绝对不会出现代码报错的问题。为什么？因为你在运行index.js的代码之前。在上面已经确保引入了，它所依赖的几个模块。这样的话，依赖颠倒，或者依赖顺序错误，造成代码运行错误，这样的问题了，就不存在了。

所以，如果我们能这样写代码，而且这样的代码，能在浏览器上运行，是不是很好啊！

保存一下！我们在浏览器上试一下。用浏览器打开index.html。刷新一下，打开控制台，看到，这里就直接报错了，Unexpected identifier；

点击进去看具体报什么错！我们看到import Header这段语句出了错。学习过Vue或者React的同学。肯定对这个语法非常的熟悉。大家会觉得，这个语法，不就是我们经常说的ES module吗？

以前我们在写react，vue的时候。这样的语法都能用啊。但是这里怎么就报错了呢？其实原因是这样的，在浏览器里，压根就不认识import这样的一个语句。浏览器根本就不知道你在做什么。所以，如果我们想这样去，实现我们的代码。该怎么做呢？这个时候webpack就登场了。虽然原生浏览器，不知道import这个语句是什么意思。但是webpack知道。webpack知道import是在，引入一个模块。它可以帮浏览器做一个翻译。告诉浏览器，你是在引入一个模块。通过webpack，把这一段代码做一次翻译。浏览器就能够，正确识别我们的代码，正常的运行我们的代码了。

所以这个时候，我们可以安装一下webpack，对当前的代码做一次翻译，首先怎么安装呢？

在我们的项目目录，session目录下，打开命令行工具，运行npm init生成一个，项目描述文件。如果大家想执行npm init。前提条件是，你已经安装了node.js，和npm 包管理工具。好，这里我们一路回车。输入yes 等这个命令执行完。

这个命令主要是，在我们项目目录下，生成一个package.json，这样的一个，项目初始化，描述文件，我们可以看一下。

在这里我们执行下，npm install webpack webpack-cli --save-dev回车。稍等一会，等它安装好。

好！这个时候我们看到webpack， webpack-cli都已经安装好了。接下来，我们让webpack对我们的这段代码做一个翻译。

怎么翻译呢？其实非常的简单。我们在命令行里，这样来写就可以了。npx webpack index.js点击回车。这个时候，webpack就帮我们把这段代码，做了一次翻译。

我们回到session目录下，大家有没有发现，目录下多了个dist文件夹，dist文件下，又多了个main.js文件。这就是webpack帮，翻译好的文件。

这个时候打开我们的index.html，我们把之前的src更改为，同级dist目录下的main.js文件。

<script src=’./dist/main.js’></script>

保存一下，我们再来打开我们的网页。刷新一下，大家来看，header， sideBar，content，是不是都出来了。

好！回过头来看一下。按照我们想要，编写代码的方式。来编写我们的代码。但是index.js这段代码，是在浏览器中无法执行的。好在！webpack有这样的一个翻译功能。它可以帮助我们把这段代码，翻译成浏览器能够运行的代码。这个代码被翻译，放在了dist目录下的main.js里面。

我们在index.html中，引入dist目录下的main.js文件。是不是就能正确的，把我们的代码，想要执行的结果，执行出来了啊。

好了，讲到这儿！大家是不是吃了一惊！噢~~，我懂了！webpack原来就是一个，js代码的翻译器啊！实际上，如果你按照这个定义去理解的话，是完全不准确的。那webpack的准确定义是什么呢？我们下节课，再给大家讲解。这节课呢，大家先有个，感官上的认识。Webpack好像是一个js的翻译器，就可以了。下一节课，我会给大家指出。Webpack明确的定义。这节课呢，就给大家讲解这么多，大家加油！！！

2-3 什么是模块打包工具？

同学们，大家好！我们接着上节课，继续给大家，讲解webpack，上节课讲完了呢，有的同学肯定会觉得，webpack就是一个js的翻译器。

其实呢，webpack称不上，是一个翻译器。我们打开google，百度搜索一下webpack，打开webpack的官方网站（这里，我们把页面放大一点）……

发现它的定义是： 一个现代JavaScript 应用程序的静态模块打包器，也叫module bundler，好，我们在笔记上，记一下。

webpack的核心定义是：

// webpack定义：**webpack** is a static module bundler

import Header from ‘./header.js’;

大家来看，这一行语法，import Header from ‘./header.js’; 首先我们来看，Header它是一个什么呢？实际上，Header就是一个模块。SideBar就是一个模块，Content也是一个模块。它们都是符合ES Module引入方式的一个模块。

webpack的作用，是能够把这些模块，打包到一起的，这样的一个工具。所以呢，叫做模块打包工具。大家这样去理解，是不是，就很容易理解了。

import实际上是在引入，一个模块。webapck能够把你引入的这些模块，打包到一起，生成一个最终的JS文件。所以它是模块打包工具，说到这儿，大家还要记住一点，就是。当你用import，引入这些类，或者引入一些组件代码的时候。实际上，我们把这样的代码，叫做一个模块。

所以看到import后面的内容，大家就应该想到，他是一个模块，webpack把这些模块，打包到一起去。

大家知道，在我们写JS代码的时候，不仅仅有ES Module这样的模块引入方式。实际上还有

// CommonJS的模块引入方式 （这是node里面，最常用的，一个模块引入方式）

还有诸如CMD, AMD这样的引入方式，

// CMD

// AMD

最终写成这样

// CommonJS的模块引入方式

// CMD

// AMD

webpack对这样，引入方式的模块。一样，可以正确的识别。好！我们举个例子，来说明一下。

我们把这种，import的引入方式，换成commonJS的模块，引入方式，来写。我们先把这块，给注释掉

// import Header from ‘./header.js’;

// import Sidebar from ‘./sidebar.js’;

// import Content from ‘./content.js’;

改成这种，commonJS模块引入的方式来写，我们暂时先不用const 定义变量，直接使用var来定义。

var Header = require(‘./header.js’);

var SideBar = require(‘./sidebar.js’);

var Content = require(‘./content.js’);

new Header();

new SideBar();

new Content();

那如果，你用这个commonJS做模块引入，这么写的话，那导出模块代码，相应的也要发生变化。

我们打开header.js，在header.js里，这么写！

将之前的export default Header 改为 module.exports = Header;

同理！ 再去改sideBar.js

export default SideBar 改为 module.exports = SideBar;

再去改content.js里面的输出方式

export default Content 改为 module.exports = Content;

这样的话，我们的每一个模块，通过CommonJS的这种模块导出方式，把它导出出去，

在index.js里面，我们又通过CommonJS的语法，把，这三个模块都引入进来。

webpack是一个模块打包工具，它能够识别出任何，模块引入的，这样的一个语法。所以呢，CommonJS这种语法，webpack也可以正常的帮我们去做翻译。然后，生成可以执行的代码，

好！我们可以试一下。首先把dist目录删除一下，然后再打开命令行工具！再次执行命令npx webpack index.js。用webpack对index.js的内容进行再次打包。回车！OK，我们可以看到，又多出来了一个dist目录。里面又是一个main.js的代码。

我们在浏览器中再次打开这个页面，大家可以看到，这个时候，页面依然可以正确的显示。这就是这节课，我给大家讲解的内容。实际上，在这节课里面，大家应该，很明确的知道，webpack实际上，不是一个js的翻译器，它是一个module bundler。也就是模块打包工具。最早的时候呢，webpack刚推出的时候。webpack是js的模块打包工具。

但是后来，随着webpack的发展。现在它所打包的模块，已经不仅仅，只能是js文件了，他还可以打包其他任何形式的模块文件，比如说：react和vue框架里面，经常可以见到。……

var style = require(‘./index.css’); (当前目录下的index.css文件)

再或者import style from ‘./index.less’；（当前目录下的index.less文件）

var Header = require(‘./header.js’);

var SideBar = require(‘./sidebar.js’);

var Content = require(‘./content.js’);

var style = require(‘./index.css’);

import style from ‘./index.less’;

大家可以看到，webpack发展到现在，它能够打包的模块，已经不仅仅是js文件，这样的模块了，它还能打包css这样的文件。它还能够打包png，jpg，这样的图片文件。它可以打包，任何你想打包的内容。所以webpack它准确的定义，就是一个模块打包工具。大家一定，记清楚。

怎么样？之前，你对webpack一无所知，听了这两节课，是不是，你已经知道webpack到底是什么啦！

那课程讲解完之后呢，这节课我再给大家，扩展一点知识，希望大家能够利用业余时间，可以把这两篇文章，看一下。我们打开webpack的官方网站……(<https://webpack.js.org/>)（可以把页面放大一点！好，我们把页面放大一点）

首先，我们点开“中文文档”也就是DOCUMENTATION，文档的这个位置。在这里面呢，大家，要去掌握两个点，分别是……“概念”目录下的，Modules模块。我们点开Modules，点开“什么是webpack模块”这里，在这里，详细给大家介绍了ES Module，CommonJs Module，AMD Module。这样的一些，模块化的概念。大家阅读这块相关的内容，可以对模块化的基本概念，有一个明确的认识。

然后，大家点开API 这个目录，在API这个目录下，会有MODULES这样的一个目录。在下面呢，会有一些，Module Methods，Module Variables [ˈverɪblz]也就是模块方法，模块变量，有兴趣的同学呢，大家也可以读一下这两块的内容，这两块，通过API 的形式，比较完整的给大家罗列了，所有的这种模块化，引入的语法，导出的语法，包括里面常用的一些变量。通过阅读这里，大家都可以比较好的掌握。当然呢，如果你对，这些内同，并不是很感兴趣的话。那你把我课上的内容，掌握好就可以了。课上的内容，是大家一定要会的内同。

课下的，这部分内容，是留给大家，自己作为提升使用。这节课就给大家讲解这么多。大家加油！

2-4 Webpack的正确安装方式

同学们，大家好！这节课我来给大家介绍下。Webpack的环境搭配。Webpack呢，是基于nodeJS开发的模块打包工具，所以呢，它本质上，是由node实现的。那我们首先呢，就要安装nodeJS这样的一个环境。大家跟我一起打开网页。我们呢，可以到百度里，搜索nodejs，进入nodeJS的官网。

大家可以看到，这里是一个10.13.0 LTS的版本，和11.2.0 的一个，最新版本。那我建议大家，来安装，这个10.13.0 LTS的版本就可以了。它呢是一个比较稳定的版本。当我们去安装这个nodeJS的时候，大家要尽量去安装，这个最新版本的nodeJS。因为啊，新版本的nodeJS会很大程度上，提高webpack的打包速度，如果大家仔细阅读过webpack的文档。

会发现，提升webpack打包速度里面，有两个非常重要的点。一个是你要保持，nodeJS的版本尽量的新。另一个呢就是保持webpack版本尽量的新。高版本的webpack，会利用node新版本中的一些特性。来提高它的打包速度。在js conference上，webpack的作者也提到了，在某些复杂的打包情况下。在最新的node的版本下，用webpack4打包。比老版本的nodeJS下，用webpack3打包，快了90%以上。所以呢，大家安装，node的时候，一定要安装这个最新的node版本。但是为什么我们不安装右侧的这个更新的版本呢。因为这个版本里面，不是一个稳定的版本。它是最新版没问题。但是它里面包含的一些尝试性的特性。有可能呢，存在一些小的BUG。所以为了稳定，同时呢。为了使它的版本比较新。我们用这个左侧的这个10.13.0的这个版本。如果大家呢，是windows的用户，点击一下，下载之后，下一步，下一步，安装就可以了。MAC的同学呢。也一样，点击下载，下一步，下一步，傻瓜式的就可以帮助大家，构建好，你的nodeJS这个环境。好！当你安装完成之后呢，你可以按照我下面的操作来验证一下。这里我就不演示，安装的这个过程了。因为它非常傻瓜，非常容易。好！假如，你安装好了nodeJS。打开你的终端，也就是你的控制台。在这里，你运行node –v回车，它就会打印出，你的node版本号。这里我的版本号，已经比较旧了，我是x.x.x这个版本。大家安装左侧最新的版本，打印出来就是10.13.0这个版本了。好！当然呢，这里我不升级也可以。后面的课程为了跟大家保持版本的统一呢，我会在课下，把它做一个升级。

接着呢，大家再运行一个命令，npm –v。如果呢，这里也正常输出了一个npm的版本号，那就说明，安装node的时候，连同它的npm工具，一样也安装好了。当这两个命令，能输出对应的版本号，就说明你node安装的没有任何问题。

我们就可以接下来继续的学习webpack了。首先呢，我们创建一个node的项目。在任意位置去建一个webpack-demo这样的文件夹。然后进入这个文件夹，运行npm init这个命令，去初始化我们的这个项目。这里可以填写一些基本信息。我们一路回车就可以了，最后呢，它会问你，配置文件是否OK？你回答yes，回车！好，初始化文件成功之后呢，我们用VScode打开我们的webpack-demo这个文件夹。

大家可以看到，其实npm init就是在我们的这个文件夹里，生成了一个package.json这样的一个文件。它描述我的node这个项目里面的一些信息。比如：这里可以看到我的包的名字，叫做：webpack-demo。版本号是1.0.0等。

在这里呢我们可以增加一些内容。比如我们在”description”:””,的下面增加一个”private”:true, // 意思是我们的这个项目是一个私有的项目,它不会被发布到这个npm线上仓库里面去。如果大家对npm不太了解。那你呢造着我的这个代码写上去就可以了。后面呢，大家自己去看一下node相关的内容。这块不了解，不影响我们后面课程的学习。你只要跟着写就可以了。然后呢，我们可以把这个，”main”:”index.js”配置项去掉。因为我们的这个项目啊，不会被外部引用。我们只是自己来用，没有必要，向外部暴漏一个js文件。所以这块呢我把”main”:”index.js”去掉。”script”标签里面的东西大家也可以先把它去掉。”script”: {},”license”:”ISC”如果大家想开源，那你可以写成”MIT”,”ISC”也是OK的。好这里，我们这样的话，package.json就配置好了。

接下来，项目初始化好了之后呢，我们去项目内安装webpack，我们进入到webpack-demo这个目录。打开命令行工具，输入npm install webpack webpack-cli –save-dev 你写可以不写—save-dev，直接写-D就可以了 –save-dev和-D是等价的， 回车。好，这个时候呢，我们在项目里面，webpack就已经安装成功了。我们打开项目目录来看一下，大家可以看到，当你安装好了webapck之后啊。项目目录下会多出node\_modules和package-lock.json。node\_modules这里面呢，就是webapck和它依赖的一些包都安装在这里。安装好了之后呢，同学们就会想，我在命令行里输入webpack –v试一下。大家会看到 webpack: command not found，是说webpack –v这个命令没有找到，是什么原因呢？因为当你输入webpack这个命令的时候，nodeJS会尝试到全局的这个模块目录中去找webpack，但是这个时候webpack，我们并没有安装在全局，而是在安装在项目内，所以就找不到webpack这个命令。 但是没关系，node提供了一个命令叫做npx,我们通过npx来运行webpack，在命令行中输入npx webpack –v打印出的版本是x.x.x。npx这个命令，会帮助我们，在当前项目的webpack-demo这个目录的node\_modules文件夹里，帮助我们去找webpack这个包。（我们找一下啊）所以通过npx，我们就可以在这个目录下，去运行存在于，这个目录下的webpack的这个安装包。所以这种方式，是把我们的webpack，安装在了项目内。然后呢，我们通过npx去运行webpack就可以了。这种方式呢，它的好处是什么呢？就是在不同项目当中，我们可以使用不同webpack的版本。这种webpack的安装方式，才是我推荐的方式。好这样的话，我就给大家讲解了。全局安装webpack和在项目中安装webpack的两种方式，OK，同样，我也给大家讲解了，如何通过这个版本号，去安装具体版本的这个webpack。好，我呢。接下来，把我的这个文件，都打包上传到，我们的git仓库里。供大家学习使用，大家来看啊！现在的项目目录下啊，都会有一个node\_modules这样的一个文件夹，但是，我呢。把它删除掉，然后把这个文件夹，上传到我的git仓库里，那么大家下载了代码之后，就是这个文件里面的内容，它里面不会有node\_modules这个目录。那如果你想向我一样，正常的运行webpack。