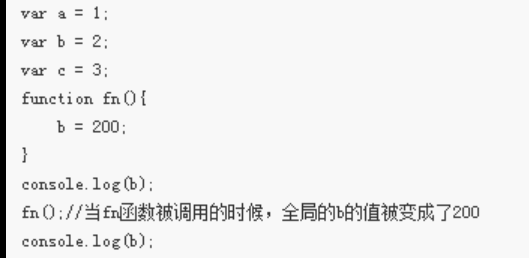
# webpack的作用

我们之前写js代码遇到的问题

1. 全局变量污染



1. 文件依赖过于复杂





为了解决这个问题，我们学习了基于amd规范的require.js

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<script src="./require.js"></script>

<script>

require(['./index.js'])//加载模块

</script>

</body>

</html>

//index.js

define(['./a','./b'],function(a,b){

console.log(a);

console.log(b);

});

//a.js

define(function(){

return {

name:'a模块',

fn:function(){

console.log(222);

}

}

});

define(function(){

return {

name:'b模块',

fn:function(){

console.log(444);

}

}

});



关于模块化，虽然我们只学过require.js(其实angular.js当中也是学过的)，我们一定要学会总结规律，所有的模块化库或者框架其实主要由以下几个部分组成的：

**如何把模块引入到html文件当中**

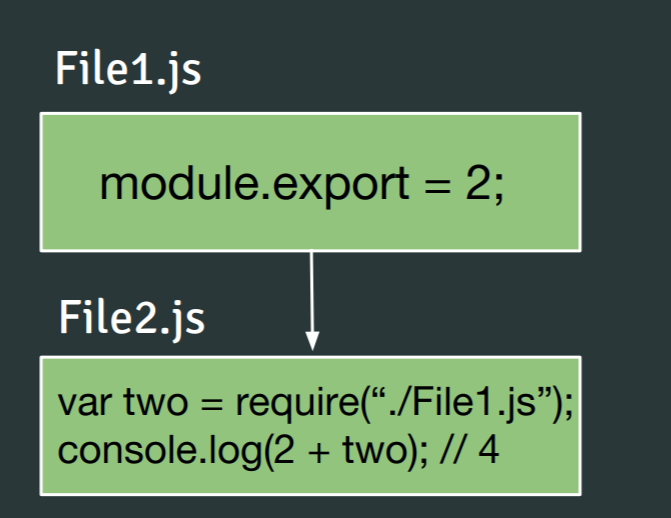
**如何创建一个模块**

**当前模块如何导出（或者说暴露）出来属性和方法**

**但是，我们在写的时候，发现很恶心，因为我们必须要引入require.js,同时，我们发现我们为了实现模块化，不得不多写出来很多的require.js代码。**

**同时，我们的代码在上线的时候，浏览器不得不多加载一个require.js的库，感觉十分的浪费。**

**但其实我们在学习Node.js的时候，也是学习过一套模块化的规范：**



我们不禁会思考，假如我们的前端也能像我们的Node.js这样：

* 不需要额外引入类似require.js这样的库
* 一个文件就是一个模块
* 通过require引入第三方模块
* 通过module.exports或者exports来导出属性和方法

那肯定是一件美好的事情(可以实现写一套代码可以同时在前后端都可以运行)，但是前端是不支持commonJs规范的，所以，这时候webpack就出现了，它可以实现：

我们写的时候按照Node.js的方式来写代码，但是写好之后，我们可以通过webpack进行翻译。

同时，有的同学在学习了es6的语法之后，也是很喜欢es6的模块化的规范，也想用，webpack本身不能翻译es6的模块化，但是，可以借助babel进行翻译。

总结：webpack让我们在写js代码的时候可以不需要引入require.js之类的库，可以使用Node.js,es6中的规范。

**so far so good**

**我们接着来看我们在写样式、图片、字体等静态资源的时候遇到的问题：**

1. **经常为了引入一小块功能，我们要引入一整个js库，但是这个js库好多代码用不上**
2. **引入一整个css文件，但是大部分样式代码用不上**
3. **后期某块功能模块删除了，但是相对应的css样式并没有被删除，这样十分浪费性能**
4. **写的样式互相冲突，总是想着早期能搞一个命名空间或作用域就好了**

**我们大胆的想，我们能不能把css,图片等静态资源也当作模块，这样，用到了我们就引入，用不上我们就不引用，如果能实现这样，我们就不会因为某块功能不需要了，样式等文件还一直没有被删除。**

**所以，总结一下，我们其实需要的是：**

1. **一切皆模块**

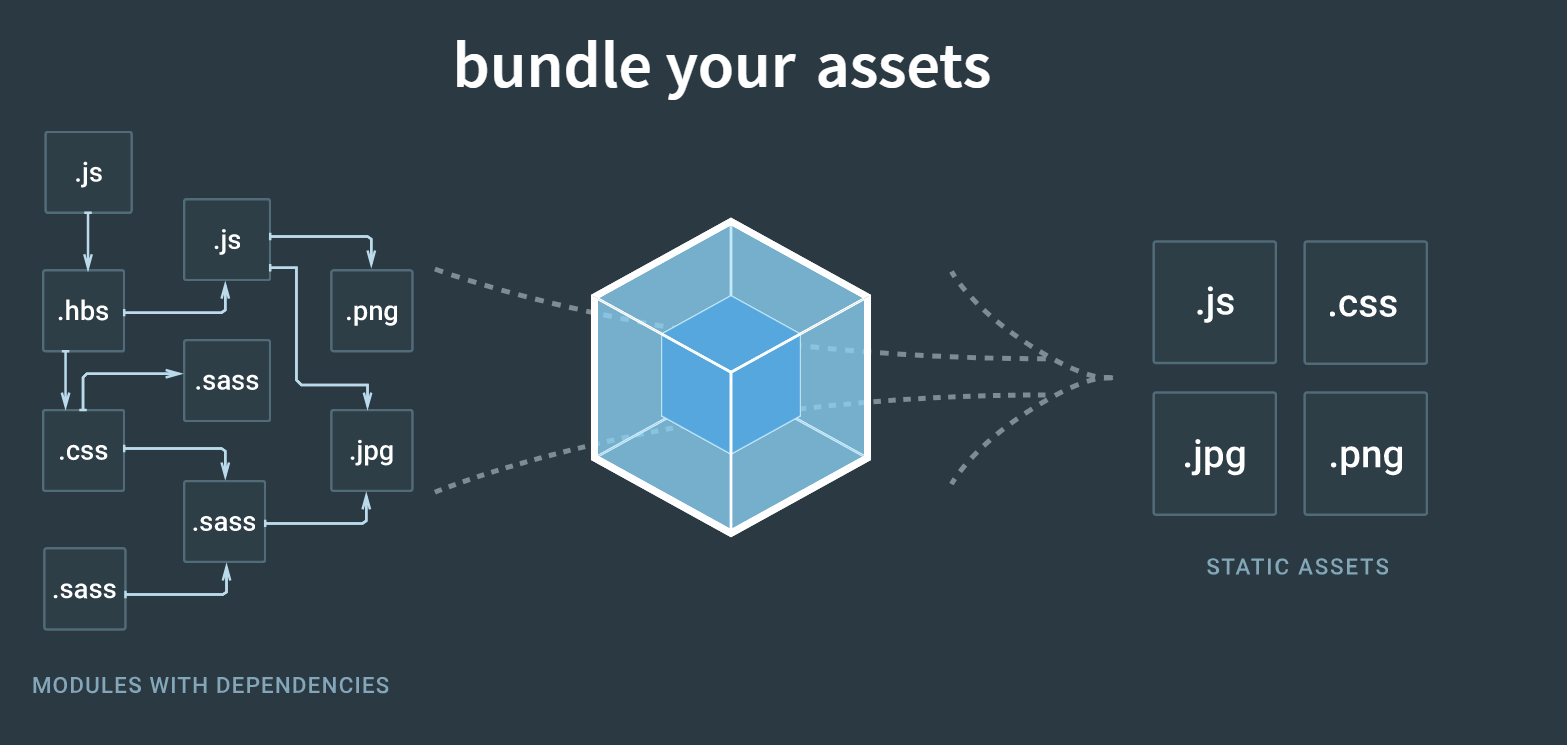
**我们将js,css等切割成一个一个的模块，实现复用和方便管理维护**

1. **按需加载和懒加载**

**同时，我们还有一些需求：**

1. **写代码用的是less,sass,上线的时候要翻译成css**
2. **实现热加载， 也就是边写代码，浏览器边刷新同步**

**所有的这一切，webpack早已为我们解决：**



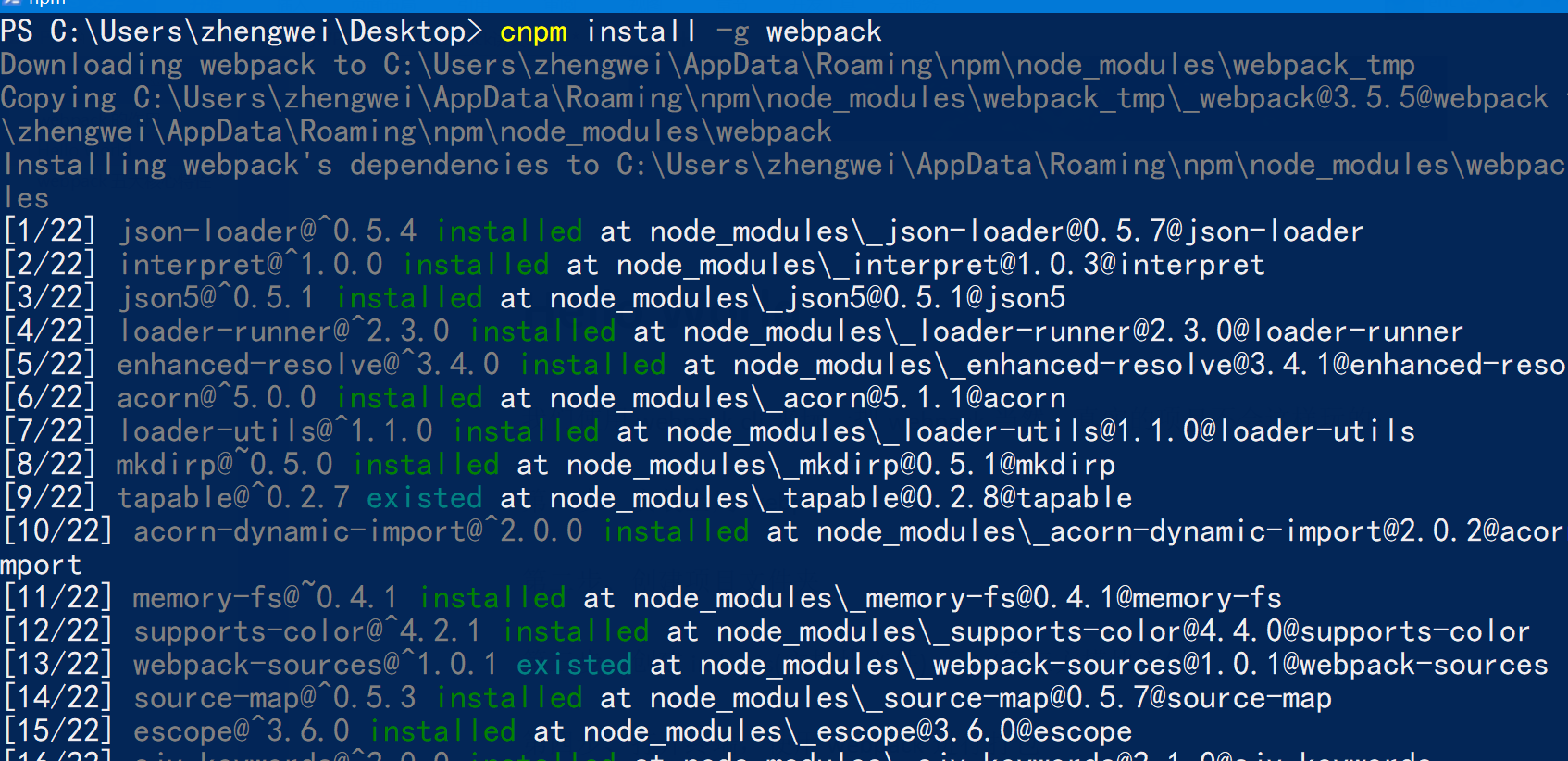
可以理解成：webpack是一个模块打包器。



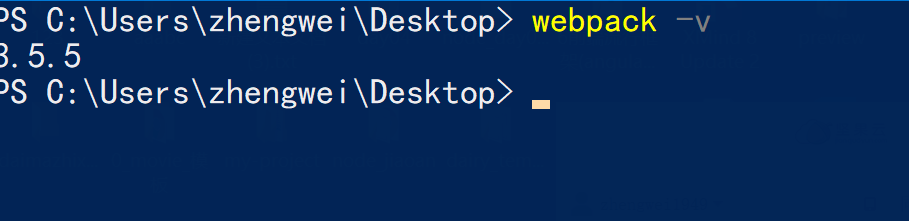
# Hello World

我们使用webpack cli来玩一把webpack,当然，真实的项目不会这样玩的：

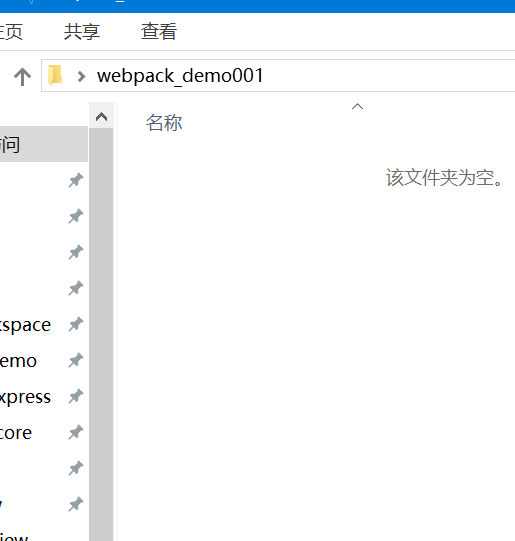
第一步、安装全局webpack



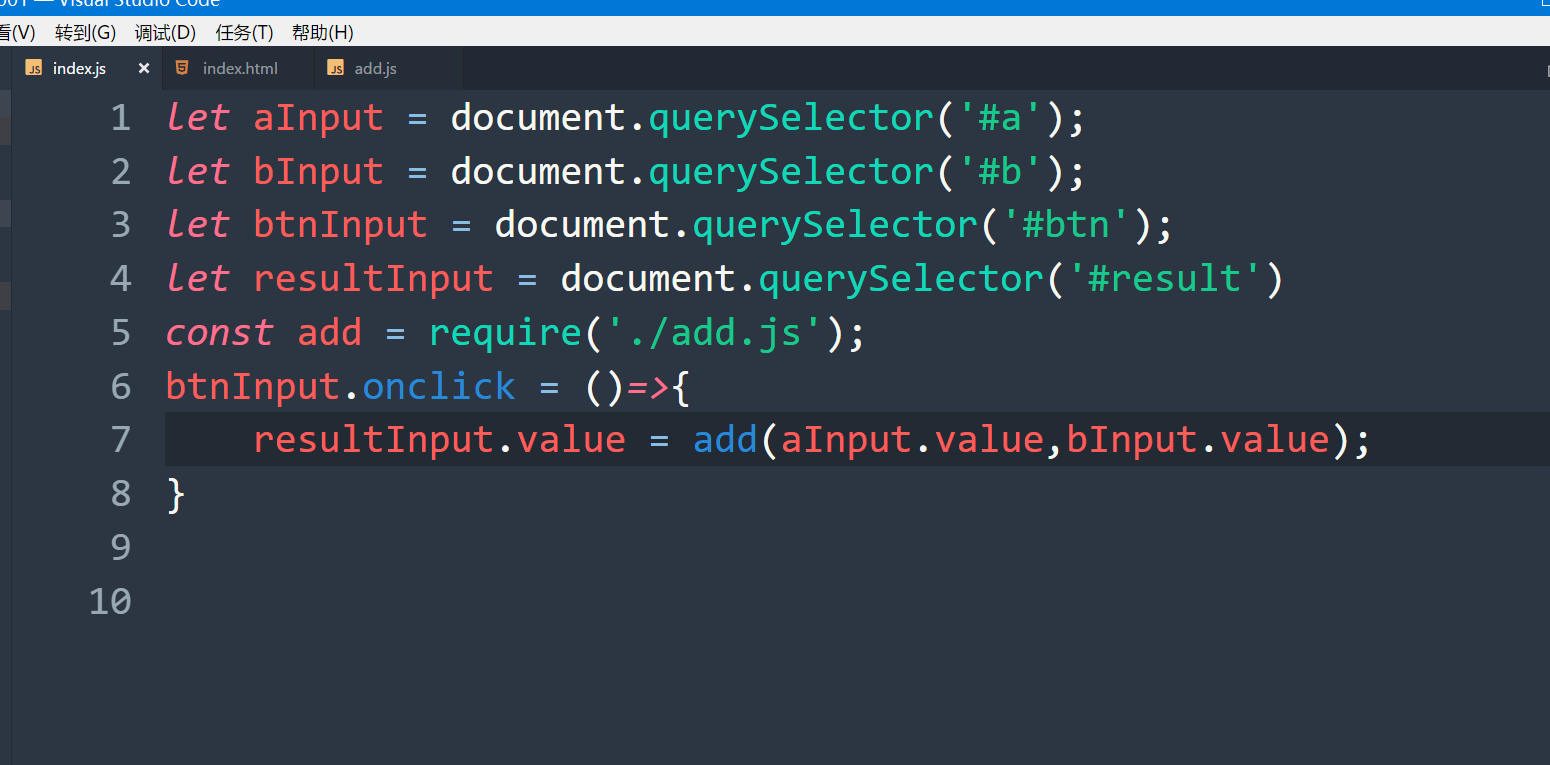
第二步、测试webpack是否安装成功

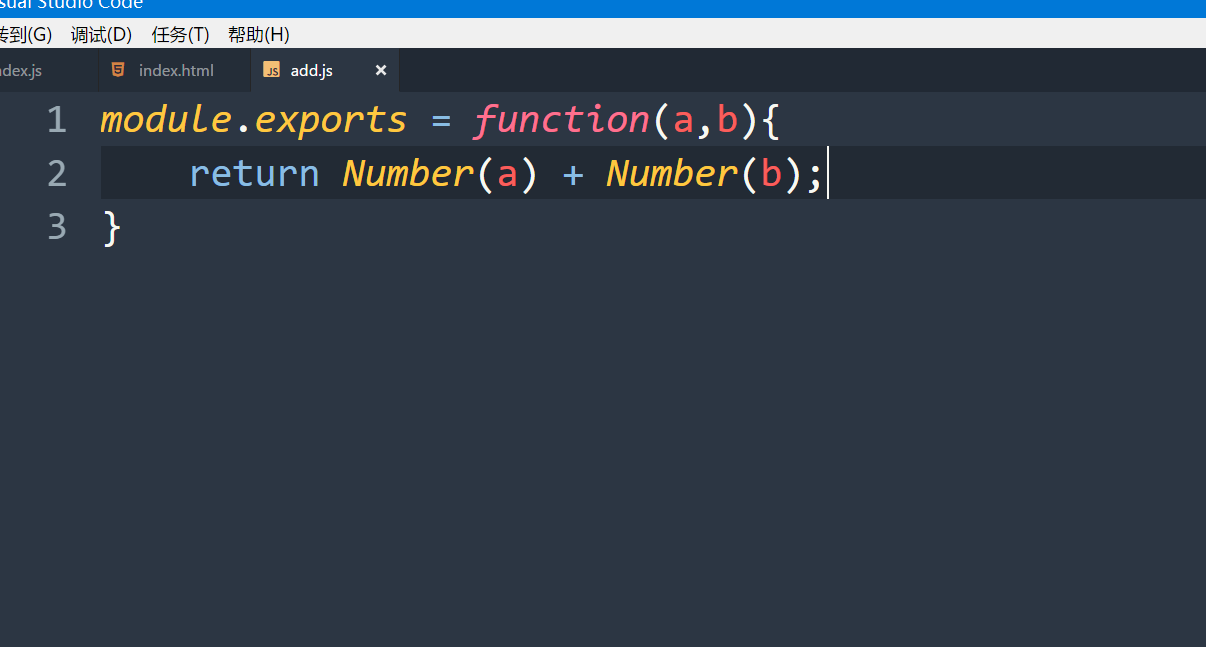


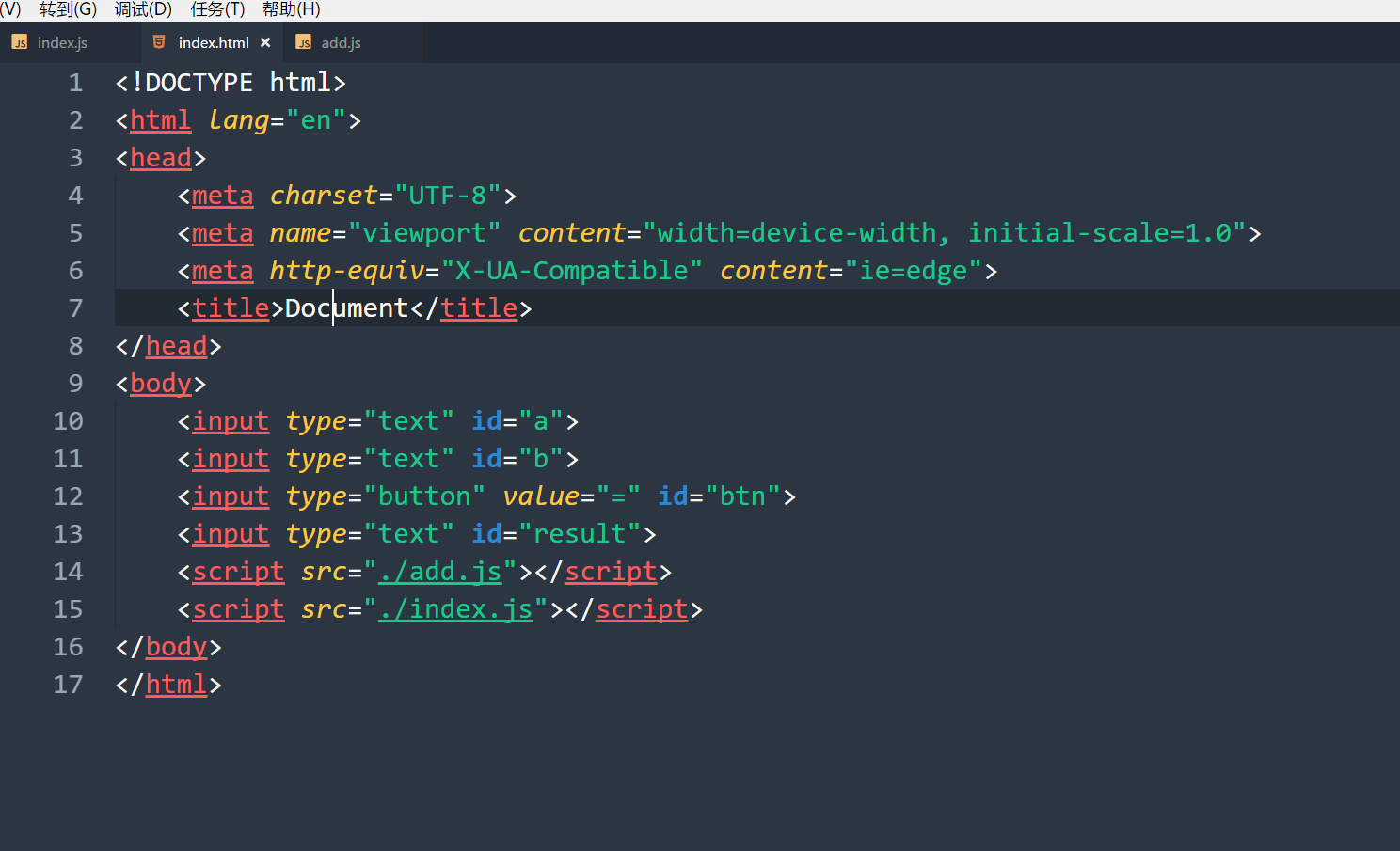
第三步、创建项目文件夹



第四步、创建index.js(主模块文件)、add.js(第三方模块文件),index.html

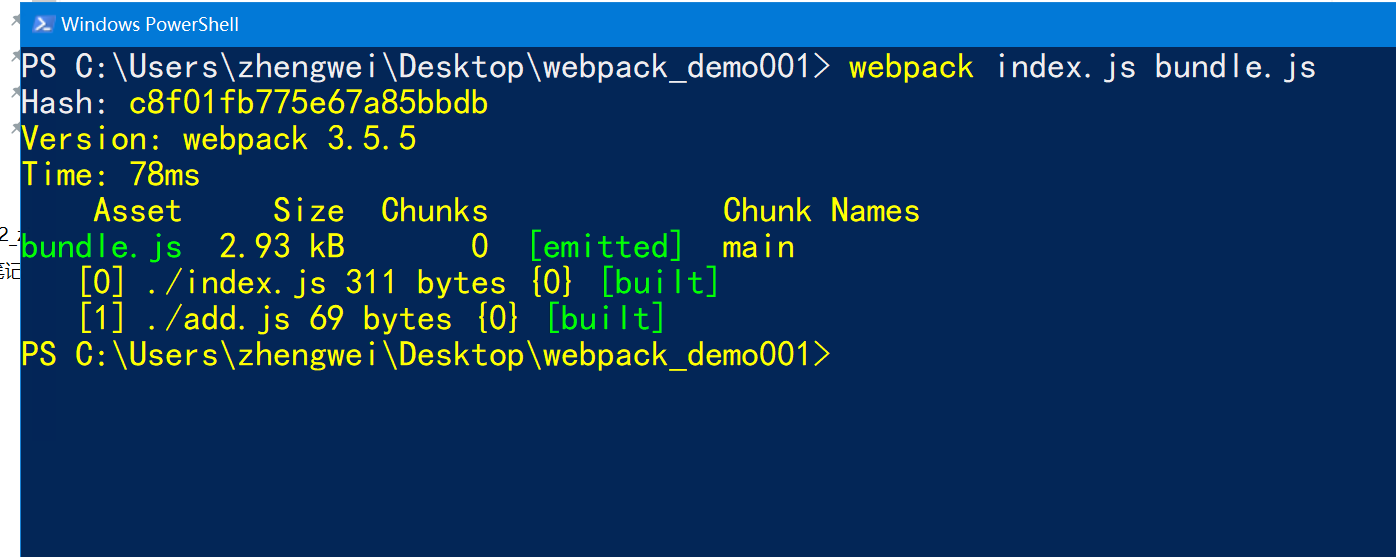






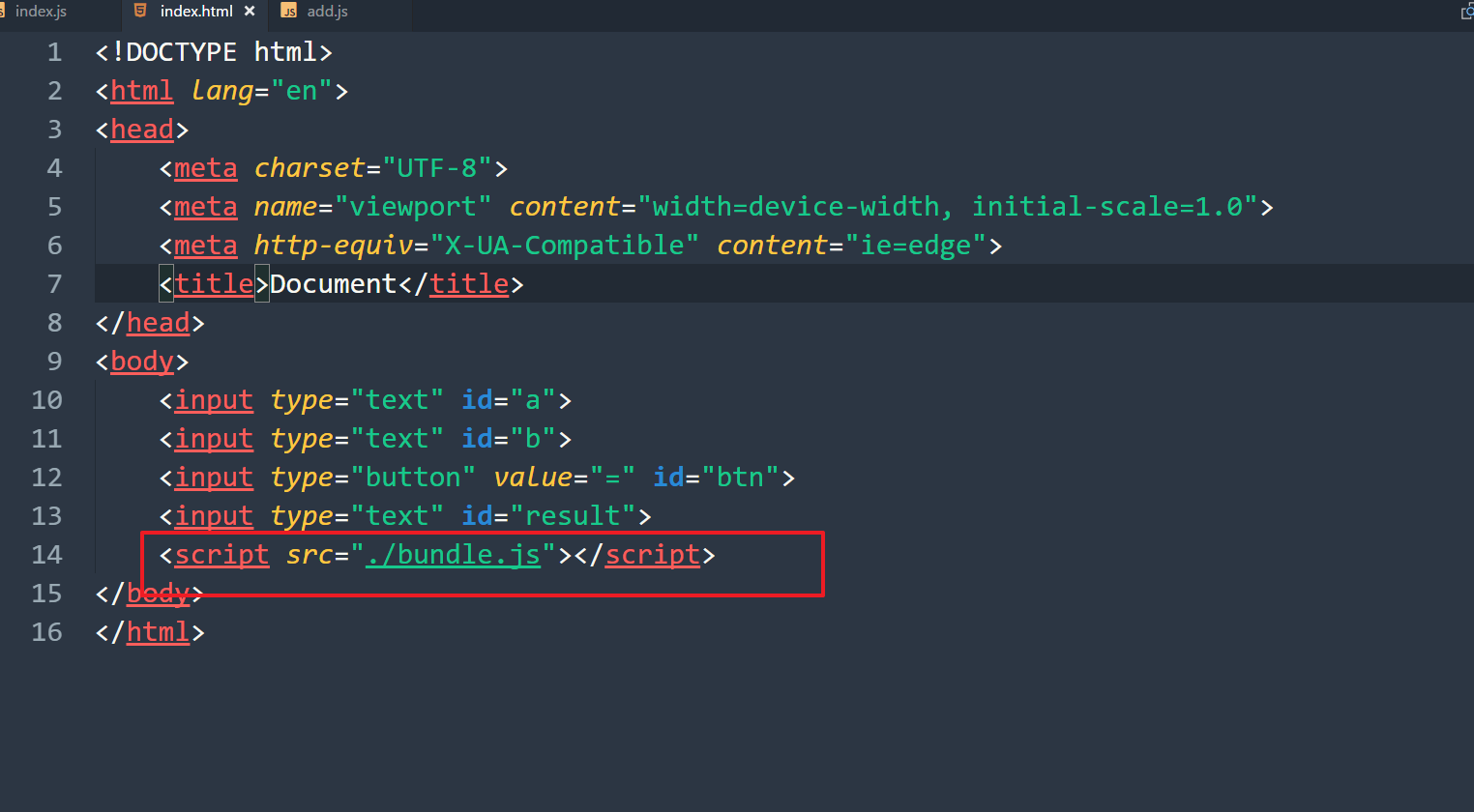
运行代码会报错，因为前端不支持commonJs规范

第五步、打开终端，使用webpack进行打包



webpack index.js bundle.js的意思是把index.js以及所有在index.js中加载的打包成一个名字叫bundle.js

第六步、修改index.html文件



第七步、测试index.html，可以成功执行代码

# webpack cli的命令详解

|  |  |
| --- | --- |
| --progress | 显示打包进度条信息 |
| --watch|简写成-w | 监视文件的变化，如果有变化重新打包 |
| -p | 压缩代码 |
| --devtool |简写成 -d | 提供sourcemap信息() |
| --colors | 输出结果带有彩色，比如，会用红色显示耗时长的步骤 |
| --profile | 输出性能数据 |
| --display-modules | 默认情况下node\_modules下的模块会被隐藏，加上这个参数可以显示这些被隐藏的模块 |
| --display-error-details | 方便出错时能查阅更详尽的信息，从而更好的定位问题 |
| --config | 指定配置文件 |
| --context | 指定要处理基于哪个文件夹 |
| --json | 以json的形式打印 |

# 学习注意事项

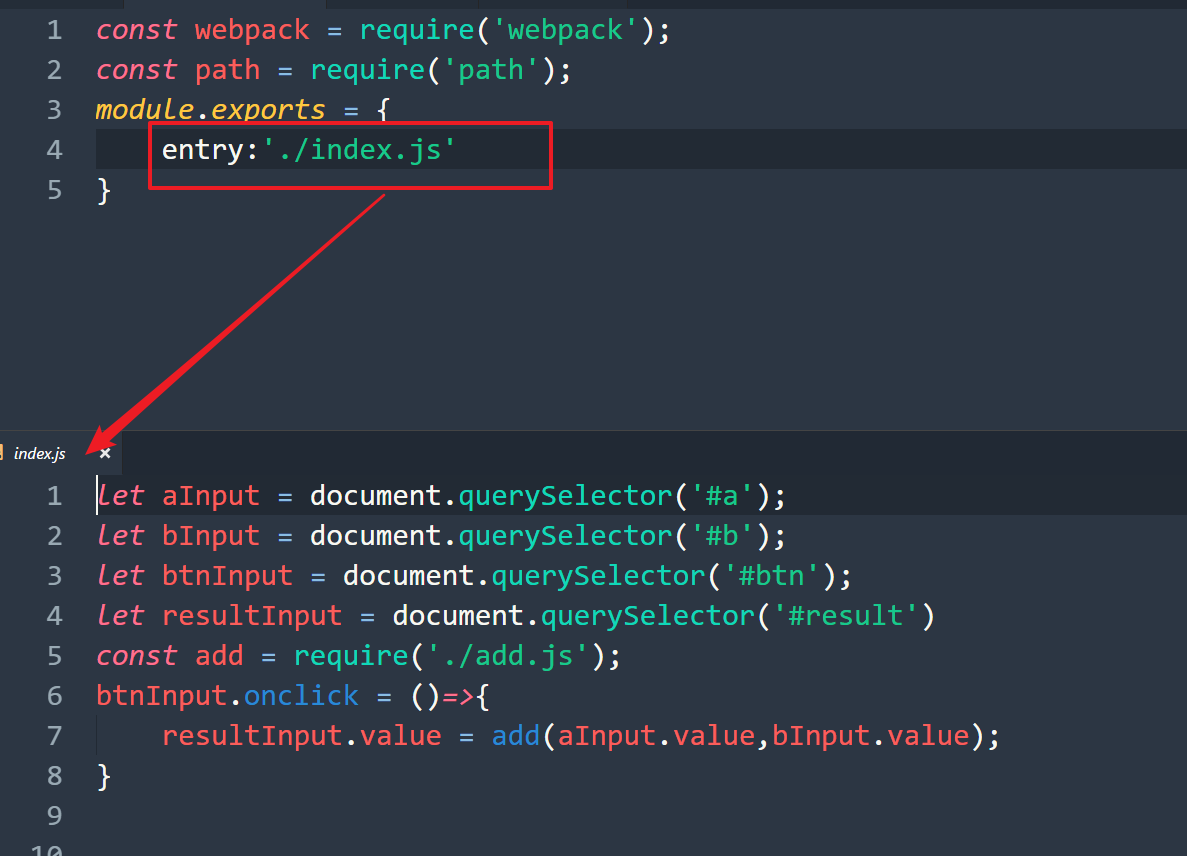
1. webpack现在3个版本了，在用的时候一定要搞明白自己用的哪一个版本
2. 尽量通过本地的方式+npm scripts的方式来运行webpack,如果装全局容易出现版本冲突用不了的问题
3. 插件和loader有些可能只能在特定的版本才可以用
4. 一开始先把我这个文档研究明白，再去看一些详细的配置参数的含义，否则会陷进去出不来
5. 尽量用cnpm

当然，我们如果用命令行来玩webpack肯定不现实，因为命令行有的时候太长了，写起来很麻烦，我们一般的还是基于webpack配置文件来玩webpack

# webpack五大核心特性

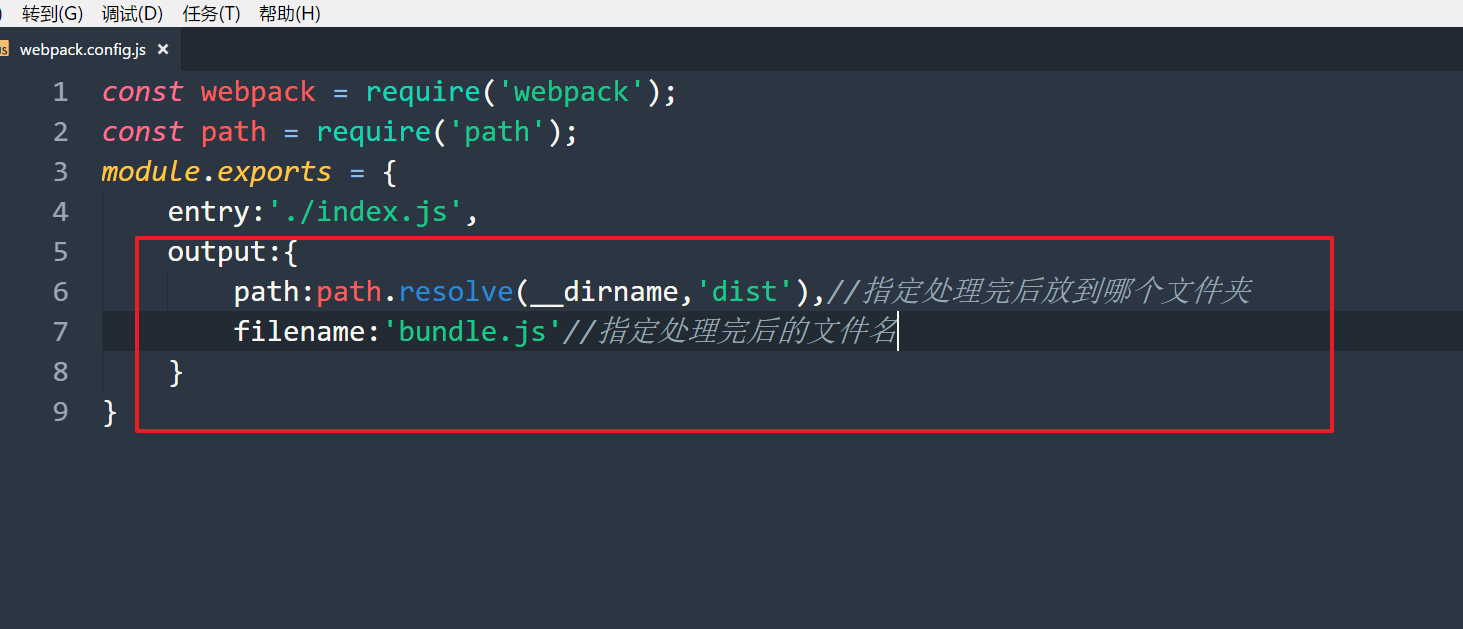
* entry 入口文件
* output 出口
* module/loaders 加载器
* plugin 插件
* devServer 开发辅助静态资源服务器

# entry



理解：从entry中指定的js文件作为入口进行解析代码，每一个通过import(es6)或require(Node.js)引入的依赖都会被打包到最终的构建结果当中。它会接着搜索那些依赖，以及那些依赖的依赖，直到所有的依赖全部被加载进来。

# output



理解：所有的依赖的文件全部打包进来

# module/loaders

记住一句话：webpack只认识javascript



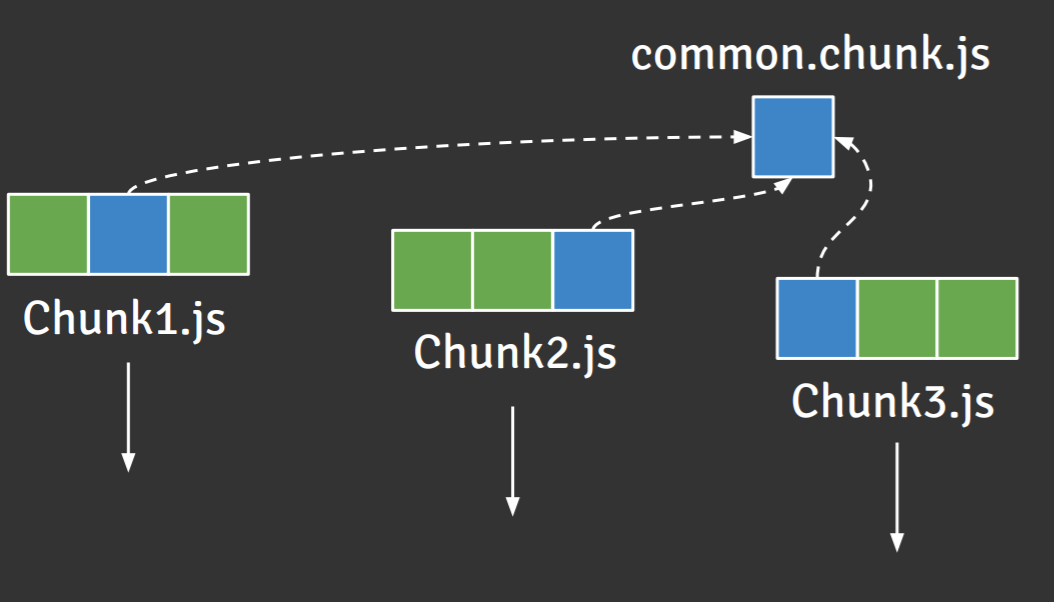
如何理解loader加载器：

1. 可以理解成一个转换器 比如像字符串"abc"，我们通过toUpperCase()转换成大写
2. 中文通过翻译软件翻译成英文, ---> less通过less转换器(less loader)转换成css,sass通过sass转换器(node sass)转换成css

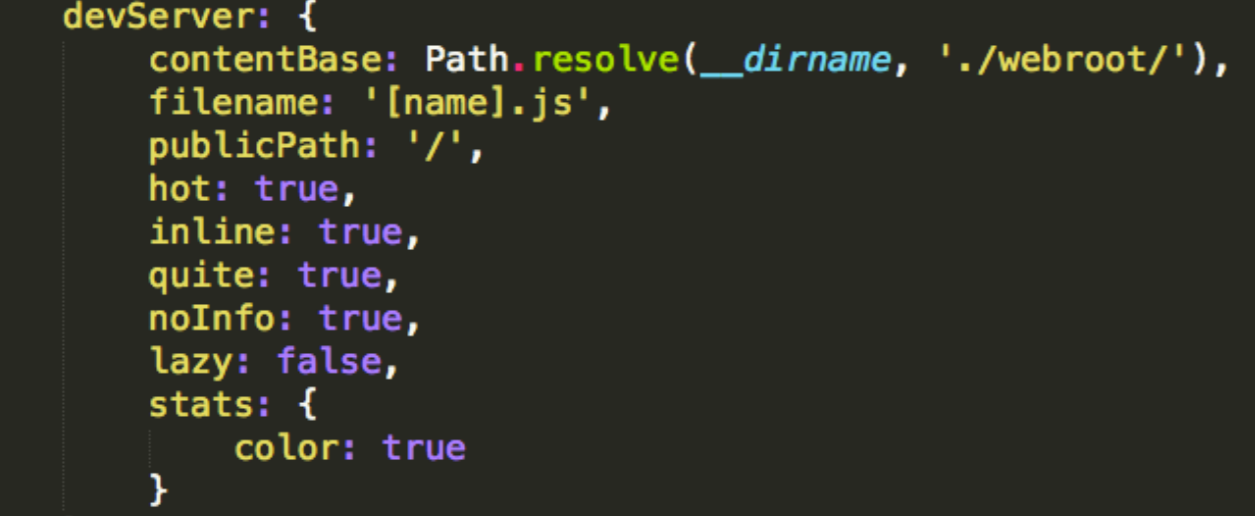
# plugin

一切loader不能做的事情由plugin负责





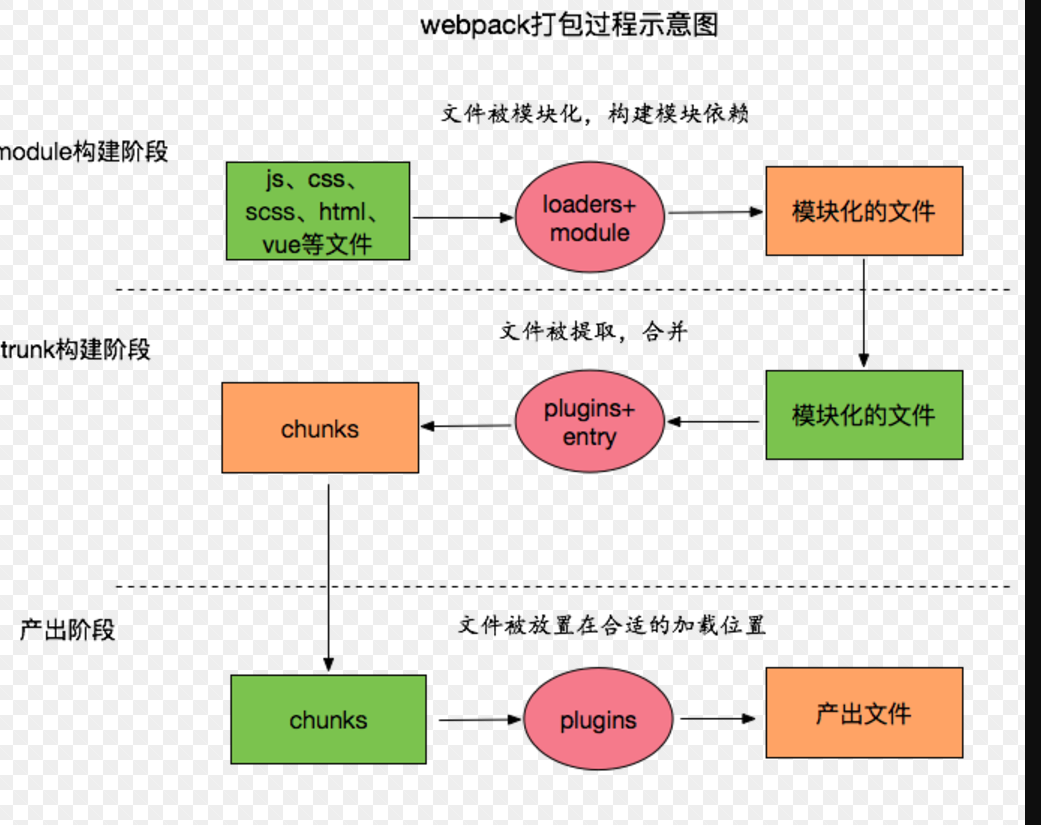
# devServer



“inline”选项为整个页面提供了“Live reloading”功能。“hot”选项提供了“模块热重载”功能，它会尝试仅仅更新组件被改变的部分（而不是整个页面）。如果我们把这两个选项都写上，那么当文件被改动时，webpack-dev-server会先尝试HMR，如果这不管用，它就会重新加载整个页面。

使用browser-sync来测试一下

# webpack打包理解

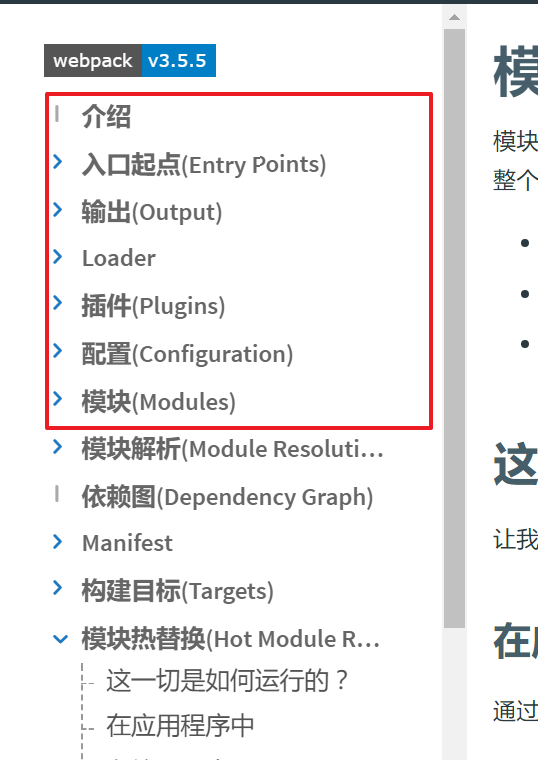


* 如果修改的是css文件，则会局部刷新，如果修改的是客户端的js文件，则浏览器整页会刷新，如果修改的是后台代码，则后台服务器会重启，再刷新前端整页
* hash:特定于整个构建过程的
* chunkhash：特定于每个文件内容的
* webpack-dev-server本质上是一个静态资源服务器
* webpack-dev-middleware和webpack-hot-middleware的静态资源服务只用于开发环境。到了线上环境，应该使用express.static()。
* 1. 不会刷新浏览器 $ webpack-dev-server //2. 刷新浏览器 $ webpack-dev-server --inline //3. 重新加载改变的部分，HRM失败则刷新页面 $ webpack-dev-server --inline --hot



# 要看的资料

<https://doc.webpack-china.org/concepts>



（看这些就可以了）

<https://doc.webpack-china.org/guides/hot-module-replacement>

webpack思维模式

1. webpack是运行在node.js环境当中，因此其中可以使用一些node提供的特殊变量和语法，比如\_\_dirname

2、用node.js的方式写写代码，但希望代码能够在浏览器端运行

webpack常见的占位符

[name] 代表打包后文件的名称，在entry或代码中（之后会看到）确定

[id]:webpack给块分配的内部chunk id,如果你没有隐藏，你能在打包后的命令行中看到

[hash]每次构建过程中，生成的唯一的hash值

[chunkhash]依赖于打包生成文件内容的hash值，内容不变，值不变

[ext]:资源扩展名，如js,jsx,png



<https://doc.webpack-china.org/loaders>

<https://doc.webpack-china.org/plugins>

<https://github.com/zhengwei1949/webapp_jiangyi/blob/master/%E5%8D%8A%E5%B0%8F%E6%97%B6%E5%AD%A6%E4%BC%9Awebpack.md>

<https://github.com/zhengwei1949/webapp_jiangyi/blob/master/webpack_peizhi.md>

<https://github.com/zhengwei1949/webpack_study>