# VSCode

## 杂项设置

### 中文界面配置

首先安装中文插件：Chinese (Simplified) Language Pack for Visual Studio Code

右下角弹出是否重启vs，点击“yes”

有些机器重启后如果界面没有变化，则 点击 左边栏Manage -> Command Paletet...【Ctrl+Shift+p】

在搜索框中输入“configure display language”，回车

打开locale.json文件，修改文件下的属性 "locale":"zh-cn"

|  |
| --- |
| {  // 定义 VS Code 的显示语言。  // 请参阅 https://go.microsoft.com/fwlink/?LinkId=761051，了解支持的语言列表。  "locale":"zh-cn" // 更改将在重新启动 VS Code 之后生效。  } |

### 插件安装

为了便于开发，建议安装如下插件：

Chinese(Simplified) Language

Live Server

Node.js Modules Intellisense

Vetur

VueHelper

### 更改编辑区的背景颜色

在C:\Program Files\Microsoft VS Code\resources\app\extensions\theme-defaults\themes目录下找到dark\_defaults的文件，以管理员身份打开，然后修改其中的editor.background属性为#292929，然后在VSCode中选择颜色主题为Dark(Visual Studio)。

### 创建Vue代码模板

文件=>首选项=>用户代码片段=>新建全局代码片段/或文件夹代码片段：vue-html.code-snippets

注意：制作代码片段的时候，字符串中如果包含文件中复制过来的“Tab”键的空格，要换成“空格键”的空格。

|  |
| --- |
| {      "vue htm": {          "scope": "html",          "prefix": "vuehtml",          "body": [              "<!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>",              " <div id=\"app\">",              "",              " </div>",              "</body>",              " <script src=\"vue.min.js\"></script>",              " <script>",              " new Vue({",              " el: '#app',",              " data: {",              " $1",              " }",              " })",              " </script>",              "</html>",          ],          "description": "my vue template in html"      }  } |

### 格式化vue文件时，js代码会自动加上冒号和分号

在设置中把"vetur.format.defaultFormatter.js": "prettier",

改为 "vetur.format.defaultFormatter.js": "vscode-typescript"

## VSCode快捷键

### 基础编辑 Basic editing

|  |  |
| --- | --- |
| **基础编辑 Basic editing快捷键** | |
| Ctrl+X  剪切行（空选定） Cut line (empty selection) | Ctrl+C  复制行（空选定）Copy line (empty selection) |
| Alt+ ↑ / ↓  向上/向下移动行 Move line up/down | Shift+Alt + ↓ / ↑  向上/向下复制行 Copy line up/down |
| Ctrl+Shift+K  删除行 Delete line | Ctrl+Enter  在下面插入行 Insert line below |
| Ctrl+Shift+Enter  在上面插入行 Insert line above | Ctrl+Shift+\  跳到匹配的括号 Jump to matching bracket |
| Ctrl+] / [  缩进/缩进行 Indent/outdent line | Home  转到行首 Go to beginning of line |
| Ctrl+Home  转到文件开头 Go to beginning of file | End  转到行尾 Go to end of line |
| Ctrl+End  转到文件末尾 Go to end of file | Ctrl+↑ / ↓  向上/向下滚动行 Scroll line up/down |
| Alt+PgUp / PgDown  向上/向下滚动页面 Scroll page up/down | Ctrl+Shift+[  折叠（折叠）区域 Fold (collapse) region |
| Ctrl+Shift+]  展开（未折叠）区域 Unfold (uncollapse) region | Ctrl+K Ctrl+[  折叠（未折叠）所有子区域 Fold (collapse) all subregions |
| Ctrl+K Ctrl+]  展开（未折叠）所有子区域 Unfold (uncollapse) all subregions | Ctrl+K Ctrl+0  折叠（折叠）所有区域 Fold (collapse) all regions |
| Ctrl+K Ctrl+J  展开（未折叠）所有区域 Unfold (uncollapse) all regions | Ctrl+K Ctrl+C  添加行注释 Add line comment |
| Ctrl+K Ctrl+U  删除行注释 Remove line comment | Ctrl+/  切换行注释 Toggle line comment |
| Shift+Alt+A  切换块注释 Toggle block comment | Alt+Z  切换换行 Toggle word wrap |

### 导航Navigation

|  |  |
| --- | --- |
| **导航Navigation快捷键** | |
| Ctrl + T  显示所有符号 Show all Symbols | Ctrl + G  转到行... Go to Line... |
| Ctrl + P  转到文件... Go to File... | Ctrl + Shift + O  转到符号... Go to Symbol... |
| Ctrl + Shift + M  显示问题面板 Show Problems panel | F8  转到下一个错误或警告 Go to next error or warning |
| Shift + F8  转到上一个错误或警告 Go to previous error or warning | Ctrl + Shift + Tab  导航编辑器组历史记录 Navigate editor group history |
| Alt + ←/→  返回/前进 Go back / forward | Ctrl + M  切换选项卡移动焦点 Toggle Tab moves focus |

### 搜索和替换 Search and replace

|  |  |
| --- | --- |
| **搜索和替换 Search and replace快捷键** | |
| Ctrl + F  查找 Find | Ctrl + H  替换 Replace |
| F3 / Shift + F3  查找下一个/上一个 Find next/previous | Alt + Enter  选择查找匹配的所有出现 Select all occurences of Find match |
| Ctrl + D  将选择添加到下一个查找匹配 Add selection to next Find match | Ctrl + K Ctrl + D  将最后一个选择移至下一个查找匹配项 Move last selection to next Find match |
| Alt + C / R / W  切换区分大小写/正则表达式/整个词 Toggle case-sensitive / regex / whole word |  |

### 多光标和选择 Multi-cursor and selection

|  |  |
| --- | --- |
| **多光标和选择 Multi-cursor and selection快捷键** | |
| Alt +单击 插入光标 Insert cursor | Ctrl + Alt +↑/↓ 在上/下插入光标 Insert cursor above / below |
| Ctrl + U 撤消上一个光标操作 Undo last cursor operation | Shift + Alt + I 在选定的每一行的末尾插入光标 Insert cursor at end of each line selected |
| Ctrl + I 选择当前行 Select current line | Ctrl + Shift + L 选择当前选择的所有出现 Select all occurrences of current selection |
| Ctrl + F2 选择当前字的所有出现 Select all occurrences of current word | Shift + Alt + → 展开选择 Expand selection |
| Shift + Alt + ← 缩小选择 Shrink selection | Shift + Alt + （拖动鼠标） 列（框）选择 Column (box) selection |
| Ctrl + Shift + Alt +（箭头键） 列（框）选择 Column (box) selection | Ctrl + Shift + Alt + PgUp / PgDown 列（框）选择页上/下 Column (box) selection page up/down |

### 丰富的语言编辑 Rich languages editing

|  |  |
| --- | --- |
| **丰富的语言编辑 Rich languages editing快捷键** | |
| Ctrl + 空格  触发建议 Trigger suggestion | Ctrl + Shift + Space  触发器参数提示 Trigger parameter hints |
| Tab Emmet  展开缩写 Emmet expand abbreviation | Shift + Alt + F  格式化文档 Format document |
| Ctrl + K Ctrl + F  格式选定区域 Format selection | F12  转到定义 Go to Definition |
| Alt + F12  Peek定义 Peek Definition | Ctrl + K F12  打开定义到边 Open Definition to the side |
| Ctrl + .  快速解决 Quick Fix | Shift + F12  显示引用 Show References |
| F2  重命名符号 Rename Symbol | Ctrl + Shift + . /，  替换为下一个/上一个值 Replace with next/previous value |
| Ctrl + K Ctrl + X  修剪尾随空格 Trim trailing whitespace | Ctrl + K M  更改文件语言 Change file language |

### 编辑器管理 Editor management

|  |  |
| --- | --- |
| **编辑器管理 Editor management快捷键** | |
| Ctrl+F4, Ctrl+W  关闭编辑器 Close editor | Ctrl+K F  关闭文件夹 Close folder |
| Ctrl+\  拆分编辑器 Split editor | Ctrl+ 1 / 2 / 3  聚焦到第1，第2或第3编辑器组 Focus into 1st, 2nd or 3rd editor group |
| Ctrl+K Ctrl+ ←/→  聚焦到上一个/下一个编辑器组 Focus into previous/next editor group | Ctrl+Shift+PgUp / PgDown  向左/向右移动编辑器 Move editor left/right |
| Ctrl+K ← / →  移动活动编辑器组 Move active editor group |  |

### 文件管理 File management

|  |  |
| --- | --- |
| **文件管理 File management快捷键** | |
| Ctrl+N  新文件 New File | Ctrl+O  打开文件... Open File... |
| Ctrl+S  保存 Save | Ctrl+Shift+S  另存为... Save As... |
| Ctrl+K S  全部保存 Save All | Ctrl+F4  关闭 Close |
| Ctrl+K Ctrl+W  关闭所有 Close All | Ctrl+Shift+T  重新打开关闭的编辑器 Reopen closed editor |
| Ctrl+K  输入保持打开 Enter Keep Open | Ctrl+Tab  打开下一个 Open next |
| Ctrl+Shift+Tab  打开上一个 Open previous | Ctrl+K P  复制活动文件的路径 Copy path of active file |
| Ctrl+K R  显示资源管理器中的活动文件 Reveal active file in Explorer | Ctrl+K O  显示新窗口/实例中的活动文件 Show active file in new window/instance |

### 显示Display

|  |  |
| --- | --- |
| **显示Display快捷键** | |
| F11  切换全屏 Toggle full screen | Shift+Alt+1  切换编辑器布局 Toggle editor layout |
| Ctrl+ = / -  放大/缩小 Zoom in/out | Ctrl+B  切换侧栏可见性 Toggle Sidebar visibility |
| Ctrl+Shift+E  显示浏览器/切换焦点 Show Explorer / Toggle focus | Ctrl+Shift+F  显示搜索 Show Search |
| Ctrl+Shift+G  显示Git Show Git | Ctrl+Shift+D  显示调试 Show Debug |
| Ctrl+Shift+X  显示扩展 Show Extensions | Ctrl+Shift+H  替换文件 Replace in files |
| Ctrl+Shift+J  切换搜索详细信息 Toggle Search details | Ctrl+Shift+C  打开新命令提示符/终端 Open new command prompt/terminal |
| Ctrl+Shift+U  显示输出面板 Show Output panel | Ctrl+Shift+V  切换Markdown预览 Toggle Markdown preview |
| Ctrl+K V  从旁边打开Markdown预览 Open Markdown preview to the side |  |

### 调试Debug

|  |  |
| --- | --- |
| **调试Debug快捷键** | |
| F9  切换断点 Toggle breakpoint | F5  开始/继续 Start/Continue |
| Shift+F5  停止 Stop | F11 / Shift+F11  下一步/上一步 Step into/out |
| F10  跳过 Step over | Ctrl+K Ctrl+I  显示悬停 Show hover |

### 集成终端 Integrated terminal

|  |  |
| --- | --- |
| **集成终端 Integrated terminal快捷键** | |
| Ctrl+`  显示集成终端 Show integrated terminal | Ctrl+Shift+`  创建新终端 Create new terminal |
| Ctrl+Shift+C  复制选定 Copy selection | Ctrl+Shift+V  粘贴到活动端子 Paste into active terminal |
| Ctrl+↑ / ↓  向上/向下滚动 Scroll up/down | Shift+PgUp / PgDown  向上/向下滚动页面 Scroll page up/down |
| Ctrl+Home / End  滚动到顶部/底部 Scroll to top/bottom |  |

### 其它Other

|  |  |
| --- | --- |
| **其它Other快捷键** | |
| Ctrl + Shift + P，F1  显示命令面板 Show Command Palette | Ctrl + P  快速打开 Quick Open |
| Ctrl + Shift + N  新窗口/实例 New window/instance | Ctrl + Shift + W  关闭窗口/实例 Close window/instance |

## VSCode插件

### ESLint

JavaScript 是一个动态的弱类型语言，在开发中比较容易出错。因为没有编译程序，为了寻找 JavaScript 代码错误通常需要在执行过程中不断调适。

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。让程序员在编码的过程中发现问题而不是在执行的过程中。

ESLint 内置了一些规则，也可以在使用过程中自定义规则。

本项目的语法规则包括：两个字符缩进，必须使用单引号，不能使用双引号，语句后不可以写分号，代码段之间必须有一个空行等。

打开 config/index.js，配置开启语法检查：useEslint: true。

**ESLint配置**

|  |
| --- |
| "files.autoSave": "off",  "eslint.validate": [  "javascript",  "javascriptreact",  "vue-html",  {  "language": "vue",  "autoFix": true  }  ],  "eslint.run": "onSave",  "eslint.autoFixOnSave": true |

# Node.js

Node.js 就是运行在服务端的JavaScript，是一个事件驱动I/O服务端JavaScript环境，基于Google的V8引擎，V8引擎执行Javascript的速度非常快，性能非常好。

在DOS系统中输入node -v可以查看当前安装的Node.js的版本。

## 快速入门

创建nodeJsTest.js文件，然后输入console.log('Hello Node.js')，在VSCode中使用快捷键Ctrl + Shift + y打开命令行终端，进入到该文件所在的目录，输入node nodeJsTest.js后回车，就会在控制台上打印出Hello Node.js。

浏览器的内核包括两部分核心：DOM渲染引擎和js解析器（js引擎），js文件运行在浏览器内核中的js引擎内部，Node.js是脱离浏览器环境运行的JavaScript程序，基于V8 引擎。

## 服务器端应用开发

创建test.js文件并编写如下内容：

|  |
| --- |
| const http = require('http')  http.createServer(function (request, response) {  // 发送 HTTP 头部  // HTTP 状态值: 200 : OK  // 内容类型: text/plain  response.writeHead(200, {'Content-Type': 'text/plain'})  // 发送响应数据 "Hello World"  response.end('Hello Server')  }).listen(8888)  // 终端打印如下信息  console.log('Server running at http://127.0.0.1:8888/') |

然后在终端中运行该文件

|  |
| --- |
| PS D:\WebProject\test\nodejs> node test.js  Server running at http://127.0.0.1:8888/ |

服务器启动成功后，在浏览器中输入：http://localhost:8888/ 查看webserver成功运行，并输出html页面。如果要停止服务器，在终端中使用快捷键Ctrl + C。

# NPM

NPM全称Node Package Manager，是Node.js包管理工具，是全球最大的模块生态系统，里面所有的模块都是开源免费的；也是Node.js的包管理工具，相当于前端的Maven 。

#在命令提示符输入 npm -v 可查看当前npm版本

npm -v

## 使用npm管理项目

#建立一个空文件夹，在命令提示符进入该文件夹 执行命令初始化

|  |
| --- |
| npm init |

#按照提示输入相关信息，如果是用默认值则直接回车即可。

#name: 项目名称

#version: 项目版本号

#description: 项目描述

#keywords: {Array}关键词，便于用户搜索到我们的项目

#最后会生成package.json文件，这个是包的配置文件，相当于maven的pom.xml

#我们之后也可以根据需要进行修改。

#如果想直接生成 package.json 文件，那么可以使用命令

|  |
| --- |
| npm init -y |

## 修改npm镜像

NPM官方的管理的包都是从 http://npmjs.com下载的，但是这个网站在国内速度很慢。

这里推荐使用淘宝 NPM 镜像 http://npm.taobao.org/ ，淘宝 NPM 镜像是一个完整 npmjs.com 镜像，同步频率目前为 10分钟一次，以保证尽量与官方服务同步。

#经过下面的配置，以后所有的 npm install 都会经过淘宝的镜像地址下载

|  |
| --- |
| npm config set registry https://registry.npm.taobao.org |

#查看npm配置信息

|  |
| --- |
| npm config list |

修改仓库安装位置

|  |
| --- |
| PS D:\WebProject\test\npm> npm config set prefix "D:\\software\\aMavenofNPM"  PS D:\WebProject\test\npm> npm config list  ; cli configs  metrics-registry = "https://registry.npm.taobao.org/"  scope = ""  user-agent = "npm/6.4.1 node/v10.14.2 win32 x64"  ; userconfig C:\Users\xmm\.npmrc  prefix = "D:\\software\\aMavenofNPM"  registry = "https://registry.npm.taobao.org/"  ; builtin config undefined  ; node bin location = D:\software\Nodejs\node.exe  ; cwd = D:\WebProject\test\npm  ; HOME = C:\Users\xmm  ; "npm config ls -l" to show all defaults. |

## 安装依赖包

通过npm install命令来安装依赖包的最新版

|  |
| --- |
| #使用 npm install 安装依赖包的最新版，  #模块安装的位置：项目目录\node\_modules  #安装会自动在项目目录下添加 package-lock.json文件，这个文件帮助锁定安装包的版本  #同时package.json 文件中，依赖包会被添加到dependencies节点下，类似maven中的  <dependencies>  npm install jquery  #如果安装时想指定特定的版本  npm install jquery@2.1.x  #devDependencies节点：开发时的依赖包，项目打包到生产环境的时候不包含的依赖  #使用 -D参数将依赖添加到devDependencies节点  npm install --save-dev eslint  #或  npm install -D eslint  #全局安装  #Node.js全局安装的npm包和工具的位置：用户目录\AppData\Roaming\npm\node\_modules  #一些命令行工具常使用全局安装的方式  npm install -g webpack  #npm管理的项目在备份和传输的时候一般不携带node\_modules文件夹  npm install #根据package.json中的配置下载依赖，初始化项目 |

## 其它命令

|  |
| --- |
| #更新包（更新到最新版本）  npm update 包名  #全局更新  npm update -g 包名  #卸载包  npm uninstall 包名  #全局卸载  npm uninstall -g 包名 |

# Babel

ES6的某些高级语法在浏览器环境甚至是Node.js环境中无法执行。Babel是一个广泛使用的转码器，可以将ES6代码转为ES5代码，从而在现有环境执行执行。这意味着，你可以现在就用 ES6 编写程序，而不用担心现有环境是否支持。

## Babel的安装

### 安装命令行转码工具

Babel提供babel-cli工具，用于命令行转码。它的安装命令如下，安装完成后还需要将babel安装目录添加到系统环境变量中，babel安装目录是npm指定的仓库目录：

npm install --global babel-cli

#查看是否安装成功

babel --version

### 安装转码器

在项目中安装：npm install --save-dev babel-preset-es2015

## Babel入门使用

1、创建项目目录，并在目录下使用npm命令初始化项目：npm init -y

2、创建文件test.js，并编写ES6代码

|  |
| --- |
| // 转码前,定义数据  let input = [1, 2, 3]  // 将数组的每个元素 +1  input = input.map(item => item + 1)  console.log(input) |

3、在项目的根目录下配置babel的配置文件.babelrc，该文件名就是.babelrc，不可改变。这个配置文件用来设置转码规则和插件，基本格式如下：

|  |
| --- |
| {  "presets": ["es2015"],  "plugins": []  } |

presets字段设定转码规则，将es2015规则加入到.babelrc配置文件中。

4、安装转码器

5、转码

|  |
| --- |
| # 转码结果写入一个文件  mkdir dist1  # --out-file 或 -o 参数指定输出文件  babel src/example.js --out-file dist1/compiled.js  # 或者  babel src/example.js -o dist1/compiled.js  # 整个目录转码  mkdir dist2  # --out-dir 或 -d 参数指定输出目录  babel src --out-dir dist2  # 或者  babel src -d dist2 |

## 自定义脚本

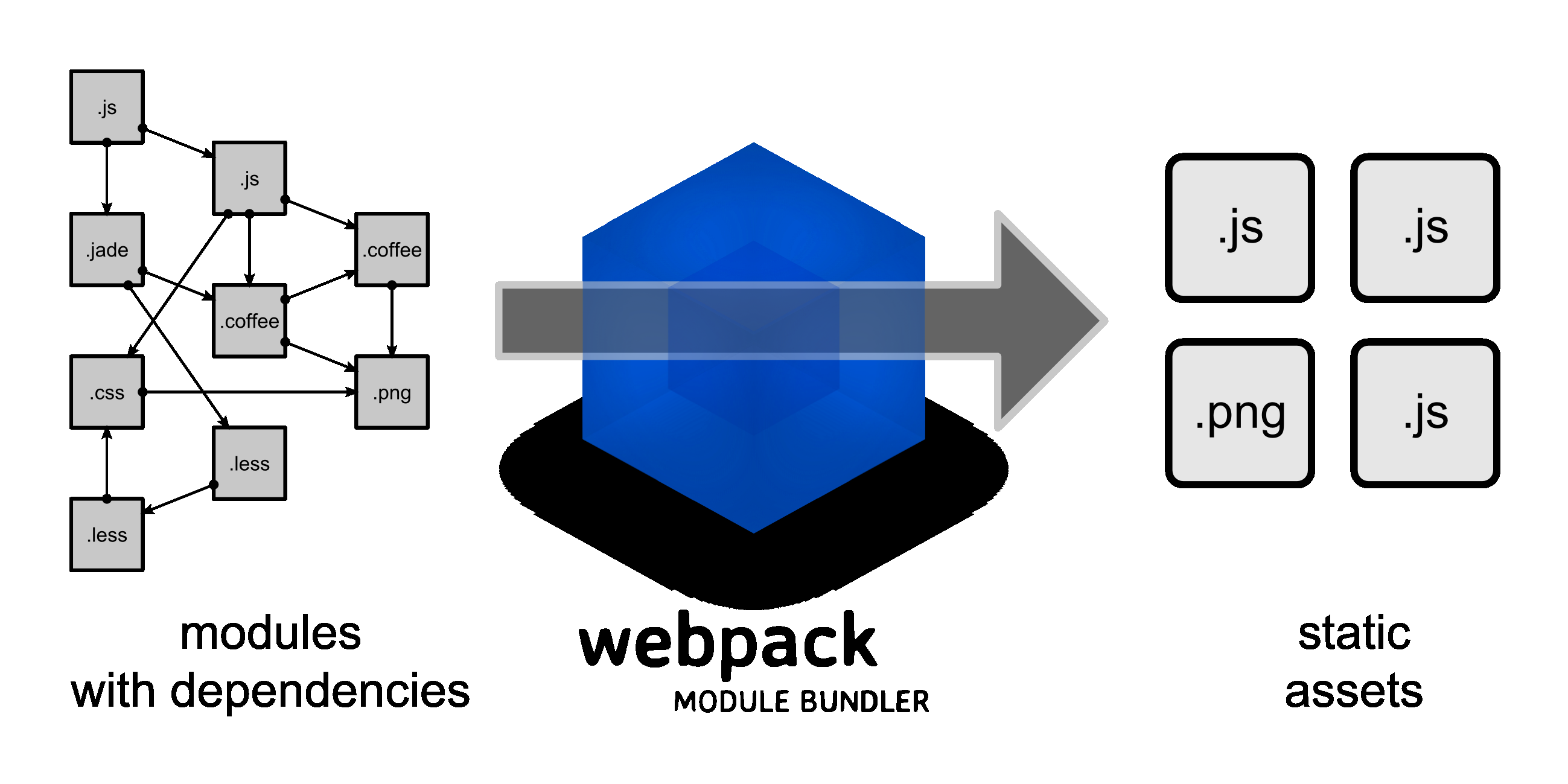
通过修改package.json配置文件来自定义脚本。

|  |
| --- |
| {  ...  "scripts": {  ...  "build": "babel .\\src\\ -d .\\dist\\ "  },  ...  } |

转码时执行：npm run build命令进行转码。

# 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。Webpack 可以将多种静态资源 js、css、less 转换成一个静态文件，减少了页面的请求。



webpack安装命令如下：npm install -g webpack webpack-cli

安装后查看版本号：webpack -v

## 打包JavaScript文件

1、创建需要打包的js文件，创建webpack目录，进入webpack目录，执行命令。

npm init -y

2、在webpack目录下创建src目录，然后创建目标js文件。

common.js

|  |
| --- |
| exports.info = function (str) {  document.write(str);  } |

util.js

|  |
| --- |
| exports.add = function (a, b) {  return a + b;  } |

main.js

|  |
| --- |
| const common = require('./common');  const utils = require('./util');  common.info('Hello world!' + utils.add(100, 200)); |

3、webpack目录下创建配置文件webpack.config.js，以下配置的意思是：读取当前项目目录下src文件夹中的main.js（入口文件）内容，分析资源依赖，把相关的js文件打包，打包后的文件放入当前目录的dist文件夹下，打包后的js文件名为bundle.js。

|  |
| --- |
| const path = require("path"); //Node.js内置模块  module.exports = {  entry: './src/main.js', //配置入口文件  output: {  //输出路径，\_\_dirname：当前文件所在路径  path: path.resolve(\_\_dirname, './dist'),  filename: 'bundle.js' //输出文件  }  } |

4、打包，打包的方式有两种

第一种：执行如下命令：

webpack --mode=development

#执行后查看bundle.js 里面包含了上面两个js文件的内容并进行了代码压缩

第二种：配置项目的npm运行命令，修改package.json文件，再执行npm run dev命令进行打包：

"scripts": {

//...,

"dev": "webpack --mode=development"

}

5、测试：

webpack目录下创建index.html并引用bundle.js，

|  |
| --- |
| <body>  <script src="dist/bundle.js"></script>  </body> |

## 打包CSS文件

1、安装style-loader和 css-loader

Webpack 本身只能处理 JavaScript 模块，如果要处理其他类型的文件，就需要使用 loader 进行转换。

Loader 可以理解为是模块和资源的转换器。需要安装相关Loader插件，css-loader 是将 css 装载到 javascript；style-loader 是让 javascript 认识css。

npm install --save-dev style-loader css-loader

2、修改webpack.config.js

|  |
| --- |
| const path = require("path"); //Node.js内置模块  module.exports = {  entry: './src/main.js', //配置入口文件  output: {  //输出路径，\_\_dirname：当前文件所在路径  path: path.resolve(\_\_dirname, './dist'),  filename: 'bundle.js' //输出文件  },  module: {  rules: [  {  test: /\.css$/, //打包规则应用到以css结尾的文件上  use: ['style-loader', 'css-loader']  }  ]  }  } |

3、在src文件夹创建style.css

|  |
| --- |
| body{  background:pink;  } |

4、修改main.js

|  |
| --- |
| require("./style.css"); |

5、运行编译命令：npm run dev，然后在浏览器中查看index.html。

# Easy Mock

## 一、什么是Easy Mock

Easy Mock是一个可视化，并且能快速生成模拟数据的服务。以项目管理的方式组织Mock List，能帮助我们更好的管理Mock数据，不怕丢失。

### 1、解决的问题

开发时，后端还没完成数据输出，前端只好写静态模拟数据。

### 2、常见的Mock方式

* 将Mock数据直接写在代码里
* 利用JS拦截请求
* 利用Charles、Fiddler等代理工具拦截请求
* Mock平台。如Easy Mock、RAP等

### 3、使用Easy Mock的优势

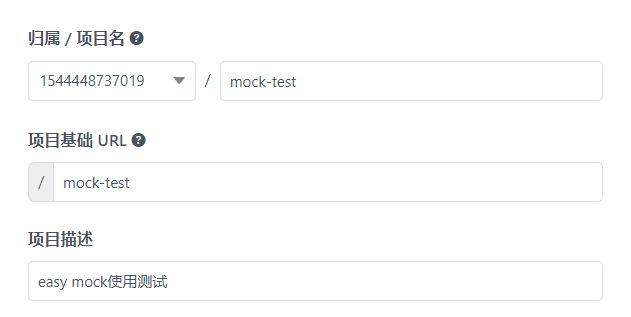
* 前后端分离：让前端工程师独立于后端进行开发。
* 增加单元测试的真实性：通过随机数据，模拟各种场景。
* 开发无侵入：不需要修改既有代码，就可以拦截Ajax请求，返回模拟的响应数据。
* 用法简单：符合直觉的接口。
* 数据类型丰富：支持生成随机的文本、数字、布尔值、日期、邮箱、链接、图片、颜色等。
* 方便扩展：支持支持扩展更多数据类型，支持自定义函数和正则。
* 在已有接口文档的情况下，我们可以直接按照接口文档来开发，将相应的字段写好，在接口完成之后，只需要改变url地址即可。

## 二、使用Easy Mock

### 1、注册Easy Mock账号

<https://www.easy-mock.com/>

### 2、创建项目



### 3、创建列表接口



### 4、编辑json数据

将接口 <http://localhost:8101/admin/edu/teacher> 返回的json数据复制过来

### 5、预览



### 6、测试接口

将easy mock接口地址替换到axios.html文件中

axios.get('https://www.easy-mock.com/mock/5c72aae79f7c4d72ba29a8f5/mock-test/admin/edu/teacher')

### 7、浏览器中运行

## 三、Mock.js占位符语法

模拟的json数据人工编辑很繁琐，我们可以使用Mock.js语法动态生成json数据

**文档：**

<https://www.easy-mock.com/docs>

<https://github.com/nuysoft/Mock/wiki>

**示例：**

<http://mockjs.com/examples.html>

{

 "success": true,

 "code": 20000,

 "message": "成功",

 "data": {

   "items|5-10": [{

     "id": "@id",

     "name": "@cname",

     "intro": "@csentence",

     "career": "@cparagraph",

     "level|1": [1, 2],

     "avatar": "@image(200x200)",

     "sort|1-100": 1,

     "deleted|1": true,

     "gmtCreate": "@datetime",

     "gmtModified": "@datetime"

  }]

}

}