[使用webpack 2](#_Toc21917)

[webpack配置文件 2](#_Toc30489)

[概念 2](#_Toc27227)

[入口起点(entry points) 2](#_Toc20959)

[上下文(context) 2](#_Toc15910)

[devtool 2](#_Toc7370)

[输出(output) 2](#_Toc12962)

[模式(mode) 3](#_Toc18238)

[loader 3](#_Toc1200)

[插件(plugins) 3](#_Toc8571)

[解析(resolve) 3](#_Toc14054)

[Webpack2学习记录 3](#_Toc12720)

[安装webpack，打包 4](#_Toc21734)

[数据更新 4](#_Toc8825)

[webpack-dev-server开发环境打包器 4](#_Toc32336)

[生产环境和开发环境 4](#_Toc7813)

[添加模块 5](#_Toc24165)

[生产环境下压缩代码 5](#_Toc19001)

[全局挂在插件(jQuery、VUE) 5](#_Toc7569)

# 使用webpack

打包命令，webpack --config webpack.config.js

## webpack配置文件

<https://www.webpackjs.com/concepts/>

### 概念

webpack处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，而后将这些模块大包围一个或多个bundle。

四个核心概念：入口entry、输出output、loader、插件plugins。

入口entry，指定构建依赖图的起始模块；默认值：./src

出口output，指定输出bundles的位置，及bundles文件命名；默认值：./dist

loader，用于对模块的源代码进行转换；必须定义在module.rules中，必需属性：test、use。

插件plugins，增强功能；使用时，require()导入，而后添加到plugins数组(new)。

模式mode，production、development；可启用相应的webpack内置优化。

### 入口起点(entry points)

简写语法：entry: string|Array<string>

对象语法：entry: {[entryChunkName: string]: string|Array<string>}

可使用CommonsChunkPlugin插件，从对象语法中通过entryChunkName提取引用。

### 上下文(context)

语法：context: string

基础目录，绝对路径，用于从配置中解析入口起点(entry point)和 loader

### devtool

控制是否生成，以及如何生成 source map。

### 输出(output)

只有一个输出配置；必需属性：文件名filename、绝对路径path

多入口起点时，使用占位符，如[name]，确保每个文件有唯一的名称。

chunkFileName，非入口(non-entry) chunk 文件的名称。

publicPath，指定在浏览器中所引用的此输出目录对应的公开 URL

### 模式(mode)

语法：mode: production/development

告知webpack使用相应模式的内置优化；可省。

### loader

用于对模块的源代码进行转换。

loader 支持链式传递。能够对资源使用流水线(pipeline)。一组链式的 loader 将按照相反的顺序执行。loader 链中的第一个 loader 返回值给下一个 loader。在最后一个 loader，返回 webpack 所预期的 JavaScript。

使用方式：

配置，配置webpack.config.js；推荐

module.rules: [{test: /\.css$/, user: [{loader: ‘style-loader’},

{loader:’css-loader, options {modules: true}}]

内联，在import语句中使用

import Styles from 'style-loader!css-loader?modules!./styles.css';

命令行中指定

webpack --module-bind jade-loader --module-bind 'css=style-loader!css-loader'

**属性**

test，条件；

enforce，’pre’/’post’；前置/后置，不指定此属性为普通loader

exclude、include，排除、包含

use，应用于模块的 UseEntries 列表；通过options为loader指定条件。

### 插件(plugins)

增强功能；使用时，require()导入，而后添加到plugins数组(new)。

### 解析(resolve)

帮助找到模块的绝对路径。

语法： resolve: { }

**属性**

extensions，自动解析确定的扩展。默认值为：[".js", ".json"]

alias，import 或 require 的模块路径别名，简化路径。数组[]

modules，解析模块时应该搜索的路径。数组[]

## Webpack2学习记录

webpack核心哲学思想：一切都是模块；将模块打包为bundle.js文件，通过特性去切割为小片段。

### 安装webpack，打包

npm install webpack -save -dev

webpack index.js bundle.js，打包

更新package.json的scripts部分，通过npm调用webpack打包。

...

“scripts”: {

“build”: “webpack index.js bundle.js”,

...

}

### 数据更新

npm install rimraf -save -dev，rimraf插件，用于删除文件

rimraf dist，删除当前目录下dist目录

rimraf dist && webpack ./src/index.js ./src/dist/bundle.js --watch，相对路径，监视

### webpack-dev-server开发环境打包器

webpack的加载器有两个重要接口：

webpack cli，默认接口，和webpack一起安装；用于生产环境。

webpack-dev-server，通过cli和配置文件webpack.config.js控制打包动作。

npm install webpack-dev-server

webpack-dev-server，打包，文件路径等在配置文件中。

**自动刷新**

webpack-dev-server -iframe，默认；页面嵌套在一个iframe下，代码改动后，iframe重新加载。

webpack-dev-server -inline -hot，先尝试更新模块，而后可能尝试刷新页面。

还可通过Nodejs API的方式实现，<https://blog.csdn.net/alanfancy/article/details/64446367>

### 生产环境和开发环境

在package.json的scripts中，设置环境：

“build”: “rimraf dist && set NODE\_ENV=production && webpack”

“dev”: “set NODE\_ENV=development && webpack-dev-server -inline -hot”

“dev”: “set NODE\_ENV=development && node dev-server.js”，API方式实现热加载

获取当前环境：

process.env.NODE\_ENV；production、development

app.get(‘env’)；production、development

获取环境后，可根据环境设置不同的webpack.config.js配置。

### 添加模块

在index.js文件中，引入其他js文件模块。

配置devtool，[https://webpack.js.org/configuration/devtool/#devtool](https://webpack.js.org/configuration/devtool/" \l "devtool)

### 生产环境下压缩代码

plugins = process.env.NODE\_ENV===”production”?[

new webpack.optimize.UglifyJsPlugin() //代码压缩，参数默认

]:[

new webpack.HotModuleReplacementPlugin() //全局开启代码热替换

];

### 全局挂在插件(jQuery、VUE)

npm install jquery -save -dev，安装jquery插件

导入、使用插件：

import $ from “jquery”;

$(“app”).html(“hello world”);

将jquery加载为全局模块，配置webpack.config.js：

externals: {

‘jquery’: ‘jQuery’ //类似pom中的dependency

}