Webpack

**概念**

本质上，*webpack* 是一个现代 JavaScript 应用程序的*静态模块打包器(module bundler)*。当 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，然后将所有这些模块打包成一个或多个 *bundle*。

它是[高度可配置的](https://doc.webpack-china.org/configuration)，但是，在开始前你需要先理解四个**核心概念**：

* 入口(entry)
* 输出(output)
* loader
* 插件(plugins)

## 入口(entry)

**入口起点(entry point)**指示 webpack 应该使用哪个模块，来作为构建其内部依赖图的开始。进入入口起点后，webpack 会找出有哪些模块和库是入口起点（直接和间接）依赖的。

每个依赖项随即被处理，最后输出到称之为 bundles 的文件中，我们将在下一章节详细讨论这个过程。

可以通过在 [webpack 配置](https://doc.webpack-china.org/configuration)中配置 entry 属性，来指定一个入口起点（或多个入口起点）。

接下来我们看一个 entry 配置的最简单例子：

**webpack.config.js**

module.exports = {

entry: './path/to/my/entry/file.js'

};

*根据应用程序的特定需求，可以以多种方式配置 entry 属性。从*[*入口起点*](https://doc.webpack-china.org/concepts/entry-points)*章节可以了解更多信息。*

### 多页面应用程序

**webpack.config.js**

const config = {

entry: {

pageOne: './src/pageOne/index.js',

pageTwo: './src/pageTwo/index.js',

pageThree: './src/pageThree/index.js'

}

};

**这是什么？**我们告诉 webpack 需要 3 个独立分离的依赖图（如上面的示例）。

**为什么？**在多页应用中，（译注：每当页面跳转时）服务器将为你获取一个新的 HTML 文档。页面重新加载新文档，并且资源被重新下载。然而，这给了我们特殊的机会去做很多事：

* 使用 CommonsChunkPlugin 为每个页面间的应用程序共享代码创建 bundle。由于入口起点增多，多页应用能够复用入口起点之间的大量代码/模块，从而可以极大地从这些技术中受益。

*根据经验：每个 HTML 文档只使用一个入口起点。*

## 出口(output)

**output** 属性告诉 webpack 在哪里输出它所创建的 bundles，以及如何命名这些文件。你可以通过在配置中指定一个 output 字段，来配置这些处理过程：

**webpack.config.js**

const path = require('path');

module.exports = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

}

};

在上面的例子中，我们通过 output.filename 和 output.path 属性，来告诉 webpack bundle 的名称，以及我们想要生成(emit)到哪里。

*你可能会发现术语****生成(emitted 或 emit)****贯穿了我们整个文档和*[*插件 API*](https://doc.webpack-china.org/api/plugins)*。它是“生产(produced)”或“释放(discharged)”的特殊术语。*

*output 属性还有*[*更多可配置的特性*](https://doc.webpack-china.org/configuration/output)*，如果你想要了解更多关于 output 属性的概念，你可以通过*[*阅读概念章节*](https://doc.webpack-china.org/concepts/output)*来了解更多。*

可以存在多个入口起点，但只指定一个输出配置。

## loader

loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只理解 JavaScript）。loader 可以将所有类型的文件转换为 webpack 能够处理的有效[模块](https://doc.webpack-china.org/concepts/modules)，然后你就可以利用 webpack 的打包能力，对它们进行处理。（解释器）

本质上，webpack loader 将所有类型的文件，转换为应用程序的依赖图可以直接引用的模块。

*注意，loader 能够 import 导入任何类型的模块（例如 .css 文件），这是 webpack 特有的功能，其他打包程序或任务执行器的可能并不支持。我们认为这种语言扩展是有很必要的，因为这可以使开发人员创建出更准确的依赖关系图。*

在更高层面，在 webpack 的配置中 **loader** 有两个目标。

1. 识别出应该被对应的 loader 进行转换的那些文件。(使用 test 属性)
2. 转换这些文件，从而使其能够被添加到依赖图中（并且最终添加到 bundle 中）(use 属性)

**webpack.config.js**

const path = require('path');

const config = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

},

module: {

rules: [

{ test: /\.txt$/, use: 'raw-loader' }

]

}

};

module.exports = config;

以上配置中，对一个单独的 module 对象定义了 rules 属性，里面包含两个必须属性：test 和 use。这告诉 webpack 编译器(compiler) 如下信息：

“嘿，webpack 编译器，当你碰到「在 require()/import 语句中被解析为 '.txt' 的路径」时，在你对它打包之前，先**使用** raw-loader 转换一下。”

*重要的是要记得，****在 webpack 配置中定义 loader 时，要定义在 module.rules 中，而不是 rules****。然而，在定义错误时 webpack 会给出严重的警告。为了使你受益于此，如果没有按照正确方式去做，webpack 会“给出严重的警告”*

loader 还有更多我们尚未提到的具体配置属性。

### 配置[Configuration]

[module.rules](https://doc.webpack-china.org/configuration/module/#module-rules) 允许你在 webpack 配置中指定多个 loader。 这是展示 loader 的一种简明方式，并且有助于使代码变得简洁。同时让你对各个 loader 有个全局概览：

module: {

rules: [

{

test: /\.css$/,

use: [

{ loader: ['style-loader'](https://doc.webpack-china.org/loaders/style-loader) },

{

loader: ['css-loader'](https://doc.webpack-china.org/loaders/css-loader),

options: {

modules: true

}

}

]

}

]

}

**插件(Plugins)**

插件是 wepback 的[支柱](https://github.com/webpack/tapable)功能。webpack 自身也是构建于，你在 webpack 配置中用到的**相同的插件系统**之上！

插件目的在于解决 [loader](https://doc.webpack-china.org/concepts/loaders) 无法实现的**其他事**。

**webpack.config.js**

const HtmlWebpackPlugin = require('html-webpack-plugin'); //通过 npm 安装

const webpack = require('webpack'); //访问内置的插件

const path = require('path');

const config = {

entry: './path/to/my/entry/file.js',

output: {

filename: 'my-first-webpack.bundle.js',

path: path.resolve(\_\_dirname, 'dist')

},

module: {

loaders: [

{

test: /\.(js|jsx)$/,

use: 'babel-loader'

}

]

},

plugins: [

new webpack.optimize.UglifyJsPlugin(),

new HtmlWebpackPlugin({template: './src/index.html'})

]

};

module.exports = config;