# webpack

## 概念

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，然后将这些模块打包成一个或多个bundle。

### 核心概念

#### 入口(entry)

入口起点(entry point)指示webpack应该使用哪个模块来构建其内部依赖图的开始。进入入口文件后，webpack会找出哪些模块和库是入口起点依赖的。

每个依赖项随机被处理，最后输出称之为bundles的文件中。

module.exports = {  
 entry: ‘server.js’

}

用法： entry: string | Array<string>

#### 出口（output）

output属性告诉webpack 在那里你输出它所创建的bundles，以及如何命名这些文件，默认值为./dist。基本上， 整个应用程序结构都会被编译到指定的输出路径的文件中。

即使存在多个入口起点，但只指定一个输出配置。

const path = require(‘path’)

module.exports = {

entry: ‘./file.js’,

output: {

path: path.resolve(\_\_dirname, ‘./dist’), // 目标输出目录path的绝对路径

filename: ‘xxx.js’, // 用于输出文件的文件名

}

}

#### 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。

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。

loader的两个目标：

1. test属性，用于标识出应该被对应的loader进行转换的某个或某些文件。
2. use属性，标识进行转换时，应该使用哪个loader。

const path = require(‘path’)

module.exports = {

entry: ‘./file.js’,

output: {

path: path.resolve(\_\_dirname, ‘dist’),

filename: ‘xxx.js’,

}，

module: {

rules: [

{ test: /\.txt$/, use: ‘raw-loader’ }

]

}

}

当碰到在require()或import语句中被解析为’.txt’的路径时， 打包之前，先使用‘raw-loader’转换一下。

loader用于对源代码进行转换，可以在import或者加载模块时预处理文件。

不同语言转换 TypeScript -> JavaScript

内联图像转换为data URL

在JavaScript文件中import css文件

首先要安装对应的loader

npm install –save-dev css-loader

npm install –save-dev ts-loader

然后在webpack配置文件中配置

module.exports = {

module: {

rules: [

{ test: /\.css$/, use: ‘css-loader’ },

{ test: /\.ts$/, use: ‘ts-loader’ }

]

}

}

#### 插件(plugins)

插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量。

想要使用一个插件只需require()它，然后把它添加到plugins数组中。

在于解决loader无法解决的其他事。

const path = require(‘path’)

const htmlWebpackPlugin = require(‘html-webpack-plugin’) // 通过npm安装

const webpack = require(‘webpack’)

module.exports = {

entry: ‘./file.js’,

output: {

path: path.resolve(\_\_dirname, ‘dist’),

filename: ‘xxx.js’,

}，

module: {

rules: [

{ test: /\.txt$/, use: ‘raw-loader’ }

]

},

plugins: [

new htmlWebpackPlugin({ template: ‘./index.html’ })

]

}

#### 模式

module.exports = {

mode: ‘devel opment’, // ‘production’

}

development 将process.env.NODE\_ENV的值设为development

production将process.env.NODE\_ENV的值设为production