## **1. webpack编译流程**

* 初始化参数：从配置文件和Shell语句中读取与合并参数，得出最终的参数；
* 开始编译：用上一步得到的参数初始化Compiler对象，加载所有配置的插件，执行对象的run方法开始执行编译； 确定入口：根据配置中的entry找出所有的入口文件
* 编译模块：从入口文件出发，调用所有配置的Loader对模块进行编译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理；
* 完成模块编译：在经过第4步使用Loader翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系
* 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个Chunk转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会
* 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统

在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用 Webpack 提供的 API 改变 Webpack 的运行结果



## **2.调试webpack**

* [cli](https://webpack.js.org/api/cli)

### **2.1 如何生成调试文件**

* 打开工程目录，点击调试按钮，再点击小齿轮的配置按钮系统就会生成launch.json配置文件
* 修改好了以后直接点击F5就可以启动调试

.vscode\launch.json

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "debug webpack",

"cwd":"${workspaceFolder}",

"program":"${workspaceFolder}/node\_modules/webpack-cli/bin/cli.js"

}

]

}

### **2.2 webpack.cmd**

webpack-source\node\_modules\.bin\webpack.cmd

%~dp0是批处理文件所在的盘符:+路径(%~dp0 C:\vipdata\vipproject\webpack-source\node\_modules.bin)

* SETLOCAL主要针对临时环境变量，不会影响到系统的变量环境设置，应与endlocal联用
* PATHEXT当在一个相同的目录结构下，有相同的多个主文件名，不同的文件后缀名时，系统会根据PATHEXT中的后缀名，选择其中顺序最靠前的那一个

@IF EXIST "%~dp0\node.exe" (//如果当前盘符的根目录下存在node.exe,用当前的node执行

"%~dp0\node.exe" "%~dp0\..\\_webpack@4.39.3@webpack\bin\webpack.js" %\*

) ELSE (//如果当前的盘符没有node.exe

@SETLOCAL

@SET PATHEXT=%PATHEXT:;.JS;=;%

node "%~dp0\..\\_webpack@4.39.3@webpack\bin\webpack.js" %\*

)

### **2.3 webpack.js**

* node\_modules\webpack\bin\webpack.js

**const** path = require("path");**const** pkgPath = require.resolve(`${installedClis[0].package}/package.json`);**const** pkg = require(pkgPath);require(path.resolve(

path.dirname(pkgPath),

pkg.bin[installedClis[0].binName]

));

**const** path = require("path");**const** pkgPath = require.resolve(`webpack-cli/package.json`);**const** pkg = require(pkgPath);require(path.resolve(path.dirname(pkgPath),pkg.bin['webpack-cli']));

npx webpack = node ./node\_modules/webpack-cli/bin/cli.js

### **2.4 cli.js**

**const** webpack = require("webpack");**const** webpackOptions = require("./webpack.config");**const** compiler = webpack(webpackOptions);

compiler.run((err, stats) => {

console.log(err);

console.log(stats.toJson({

entries:true,

chunks:true,

modules:true,

\_modules:true,

assets:true

}));

});

## **3.Stats 对象**

* 在 Webpack 的回调函数中会得到stats对象
* 这个对象实际来自于Compilation.getStats()，返回的是主要含有modules、chunks和assets三个属性值的对象。
* Stats对象本质上来自于[lib/Stats.js](https://github.com/webpack/webpack/blob/v4.39.3/lib/Stats.js)的类实例

| **字段** | **含义** |
| --- | --- |
| modules | 记录了所有解析后的模块 |
| chunks | 记录了所有chunk |
| assets | 记录了所有要生成的文件 |

npx webpack --profile --json > stats.json

{

"errors": [],// 错误字符串 (error string) 的数组

"warnings": [],//警告字符串 (warning string) 的数组

"version": "4.39.3",// 用来编译的 webpack 的版本

"hash": "3e945ec6b2c56d0b010e",//编译使用的 hash

"time": 66, // 编译耗时 (ms)

"builtAt": 1567225465347,//编译的时间

"publicPath": "",//资源访问路径

"outputPath": "C:\\vipdata\\vipproject\\webpack-source\\dist",//webpack输出目录

"assetsByChunkName": {//用作映射的 chunk 的名称

"lazy": "lazy.bundle.js",//chunk的名字叫lazy,lazy.bundle.js

"main": "bundle.js"//chunk的名字叫main,打包出来了bundle.js

},

"assets": [//asset 对象 (asset objects) 的数组

{

"name": "bundle.js",//文件名

"size": 9043,//大小

"chunks": [//包含的代码块

"main"

],

"chunkNames": [//包含的代码块名称

"main"

],

"emitted": true//是否要生成

},

{

"name": "lazy.bundle.js", // 输出的文件名

"size": 336,// 文件的大小

"chunks": [ // 这个 asset 包含的 chunk 的 id

"lazy"

],

"chunkNames": [// 这个 asset 包含的 chunk

"lazy"

],

"emitted": true // 表示这个 asset 是否会让它输出到 output 目录

}

],

"filteredAssets": 0,

"entrypoints": {

"main": {

"chunks": [

"main"

],

"assets": [

"bundle.js"

],

"children": {},

"childAssets": {}

}

},

"namedChunkGroups": {

"main": {

"chunks": [

"main"

],

"assets": [

"bundle.js"

],

"children": {},

"childAssets": {}

},

"lazy": {

"chunks": [

"lazy"

],

"assets": [

"lazy.bundle.js"

],

"children": {},

"childAssets": {}

}

},

"chunks": [ //chunk 对象 (chunk objects) 的数组

{

"id": "lazy", // 这个 chunk 的id

"rendered": true,// 表示这个 chunk 是否会参与进编译

"initial": false,

"entry": false,// 表示这个 chunk 是否包含 webpack 的运行时

"size": 24,//预估的模块大小

"names": [// 包含在这个 chunk 内的 chunk 的名字的数组

"lazy"

],

"files": [

"lazy.bundle.js"

],

"hash": "d08a8b502d30324f81e1",

"siblings": [],

"parents": [// 父 chunk 的 ids

"main"

],

"children": [],

"childrenByOrder": {},

"modules": [

{

"id": "./src/lazy.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\lazy.js",

"name": "./src/lazy.js",

"index": 2,

"index2": 2,

"size": 24,

"cacheable": true,

"built": true,

"optional": false,

"prefetched": false,

"chunks": [

"lazy"

],

"issuer": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"issuerId": "./src/index.js",

"issuerName": "./src/index.js",

"issuerPath": [

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"profile": {

"factory": 18,

"building": 14

}

}

],

"profile": {

"factory": 4,

"building": 2

},

"failed": false,

"errors": 0,

"warnings": 0,

"assets": [],

"reasons": [// 生成 assets 的原因

{

"moduleId": "./src/index.js",//模块的ID

"moduleIdentifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",//唯一标识

"module": "./src/index.js",//模块

"moduleName": "./src/index.js",//模块名称

"type": "import()",//类型

"userRequest": "./lazy",//用户请求方式

"loc": "2:0-46"//在父模块中的位置

}

],

"providedExports": null,

"optimizationBailout": [],

"depth": 1,

"source": "module.exports = 'lazy';"

}

],

"filteredModules": 0,

"origins": [

{

"moduleId": "./src/index.js",// 模块的ID

"module": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",// 模块的位置

"moduleIdentifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",// 模块的地址

"moduleName": "./src/index.js",//模块的相对地址

"loc": "2:0-46",

"request": "./lazy",

"reasons": [] // 具体是哪行生成了这个chunk

}

]

},

{

"id": "main",

"rendered": true,

"initial": true,

"entry": true,

"size": 162,

"names": [

"main"

],

"files": [

"bundle.js"

],

"hash": "263cadc0459e8470151b",

"siblings": [],

"parents": [],

"children": [// 自己引用哪些chunk

"lazy"

],

"childrenByOrder": {}, // 引用的顺序

"modules": [

{

"id": "./src/hello.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\hello.js",

"name": "./src/hello.js",

"index": 1,

"index2": 0,

"size": 25,

"cacheable": true,

"built": true,

"optional": false,

"prefetched": false,

"chunks": [

"main"

],

"issuer": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"issuerId": "./src/index.js",

"issuerName": "./src/index.js",

"issuerPath": [

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"profile": {

"factory": 18,

"building": 14

}

}

],

"profile": {

"factory": 4,

"building": 2

},

"failed": false,

"errors": 0,

"warnings": 0,

"assets": [],

"reasons": [

{

"moduleId": "./src/index.js",

"moduleIdentifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"module": "./src/index.js",

"moduleName": "./src/index.js",

"type": "cjs require",

"userRequest": "./hello",

"loc": "1:12-30"

}

],

"providedExports": null,

"optimizationBailout": [],

"depth": 1,

"source": "module.exports = 'hello';"

},

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"index": 0,

"index2": 1,

"size": 137,

"cacheable": true,

"built": true,

"optional": false,

"prefetched": false,

"chunks": [

"main"

],

"issuer": null,

"issuerId": null,

"issuerName": null,

"issuerPath": null,

"profile": {

"factory": 18,

"building": 14

},

"failed": false,

"errors": 0,

"warnings": 0,

"assets": [],

"reasons": [

{

"moduleId": null,

"moduleIdentifier": null,

"module": null,

"moduleName": null,

"type": "single entry",

"userRequest": "./src/index.js",

"loc": "main"

}

],

"providedExports": null,

"optimizationBailout": [],

"depth": 0,

"source": "let hello = require('./hello');\r\nimport(/\* webpackChunkName: \"lazy\" \*/'./lazy').then(result=>{\r\n console.log(hello,resut.default)\r\n});"

}

],

"filteredModules": 0,

"origins": [

{

"module": "",

"moduleIdentifier": "",

"moduleName": "",

"loc": "main",

"request": "./src/index.js",

"reasons": []

}

]

}

],

"modules": [// 模块对象 (module objects) 的数组

{

"id": "./src/hello.js",//模块ID

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\hello.js",//webpack内部使用的唯一的标识

"name": "./src/hello.js",// 实际文件的地址

"index": 1,//索引

"index2": 0,//索引

"size": 25,// 预估模块的大小 (byte)

"cacheable": true,// 表示这个模块是否会被缓存

"built": true,// 表示这个模块会参与 Loaders , 解析, 并被编译

"optional": false,// 每一个对这个模块的请求都会包裹在 `try... catch` 内

"prefetched": false,// 表示这个模块是否会被 prefetched

"chunks": [//此模块在哪个代码块内

"main"

],

"issuer": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",//使用者唯一标识

"issuerId": "./src/index.js",//使用者ID

"issuerName": "./src/index.js",//使用者名称

"issuerPath": [//使用者路径

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"profile": { //这个模块特有的编译时间数据(ms)

"factory": 18,// 解决依赖的时间

"building": 14 // 载入和解析的时间

}

}

],

"profile": {

"factory": 4,// 解决依赖的时间

"building": 2// 载入和解析的时间

},

"failed": false,//是否失败

"errors": 0,// 处理模块时错误的数量

"warnings": 0,// 处理模块时警告的数量

"assets": [],//在哪个资源内

"reasons": [

{

"moduleId": "./src/index.js",// 模块的 ID

"moduleIdentifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",// 模块的地址

"module": "./src/index.js",// 所基于模块的相对地址 context

"moduleName": "./src/index.js",

"type": "cjs require",// 使用的请求的种类 (require或import)

"userRequest": "./hello",// 用来 `import` 或者 `require` 的源字符串

"loc": "1:12-30" // 导致这个被加入依赖图标的代码行数

}

],

"providedExports": null,//提供的导出对象

"optimizationBailout": [],//失败时的优化

"depth": 1,//模块深度

"source": "module.exports = 'hello';"// 字符串化的输入

},

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"index": 0,

"index2": 1,

"size": 137,

"cacheable": true,

"built": true,

"optional": false,

"prefetched": false,

"chunks": [

"main"

],

"issuer": null,

"issuerId": null,

"issuerName": null,

"issuerPath": null,

"profile": {

"factory": 18,

"building": 14

},

"failed": false,

"errors": 0,

"warnings": 0,

"assets": [],

"reasons": [

{

"moduleId": null,

"moduleIdentifier": null,

"module": null,

"moduleName": null,

"type": "single entry",

"userRequest": "./src/index.js",

"loc": "main"

}

],

"providedExports": null,

"optimizationBailout": [],

"depth": 0,

"source": "let hello = require('./hello');\r\nimport(/\* webpackChunkName: \"lazy\" \*/'./lazy').then(result=>{\r\n console.log(hello,resut.default)\r\n});"

},

{

"id": "./src/lazy.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\lazy.js",

"name": "./src/lazy.js",

"index": 2,

"index2": 2,

"size": 24,

"cacheable": true,

"built": true,

"optional": false,

"prefetched": false,

"chunks": [

"lazy"

],

"issuer": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"issuerId": "./src/index.js",

"issuerName": "./src/index.js",

"issuerPath": [

{

"id": "./src/index.js",

"identifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"name": "./src/index.js",

"profile": {

"factory": 18,

"building": 14

}

}

],

"profile": {

"factory": 4,

"building": 2

},

"failed": false,

"errors": 0,

"warnings": 0,

"assets": [],

"reasons": [

{

"moduleId": "./src/index.js",

"moduleIdentifier": "C:\\vipdata\\vipproject\\webpack-source\\src\\index.js",

"module": "./src/index.js",

"moduleName": "./src/index.js",

"type": "import()",

"userRequest": "./lazy",

"loc": "2:0-46"

}

],

"providedExports": null,

"optimizationBailout": [],

"depth": 1,

"source": "module.exports = 'lazy';"

}

],

"filteredModules": 0,

"logging": {

"webpack.buildChunkGraph.visitModules": {

"entries": [],

"filteredEntries": 5,

"debug": false

}

},

"children": []

}

## **4. 主要工作流程**

* Webpack 的运行流程是一个串行的过程，从启动到结束会依次执行以下流程：
* 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数；
* 开始编译：用上一步得到的参数初始化 Compiler 对象，加载所有配置的插件，执行对象的run方法开始执行编译； 确定入口：根据配置中的 entry 找出所有的入口文件
* 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行编译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理；
* 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系；
* 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会；
* 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统。
* 在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用 Webpack 提供的 API 改变 Webpack 的运行结果。

