## **1. plugin**

插件向第三方开发者提供了 webpack 引擎中完整的能力。使用阶段式的构建回调，开发者可以引入它们自己的行为到 webpack 构建流程中。创建插件比创建 loader 更加高级，因为你将需要理解一些 webpack 底层的内部特性来做相应的钩子

### **1.1 为什么需要一个插件**

* webpack基础配置无法满足需求
* 插件几乎能够任意更改webpack编译结果
* webpack内部也是通过大量内部插件实现的

### **1.2 可以加载插件的常用对象**

| **对象** | **钩子** |
| --- | --- |
| [Compiler](https://github.com/webpack/webpack/blob/v4.39.3/lib/Compiler.js) | run,compile,compilation,make,emit,done |
| [Compilation](https://github.com/webpack/webpack/blob/v4.39.3/lib/Compilation.js) | buildModule,normalModuleLoader,succeedModule,finishModules,seal,optimize,after-seal |
| [Module Factory](https://github.com/webpack/webpack/blob/master/lib/ModuleFactory.js) | beforeResolver,afterResolver,module,parser |
| Module |  |
| [Parser](https://github.com/webpack/webpack/blob/master/lib/Parser.js)] | program,statement,call,expression |
| [Template](https://github.com/webpack/webpack/blob/master/lib/Template.js) | hash,bootstrap,localVars,render |

## **2. 创建插件**

webpack 插件由以下组成：

* 一个 JavaScript 命名函数。
* 在插件函数的 prototype 上定义一个 apply 方法。
* 指定一个绑定到 webpack 自身的事件钩子。
* 处理 webpack 内部实例的特定数据。
* 功能完成后调用 webpack 提供的回调。

## **3. Compiler 和 Compilation**

在插件开发中最重要的两个资源就是compiler和compilation对象。理解它们的角色是扩展webpack引擎重要的第一步。

compiler 对象代表了完整的 webpack 环境配置。这个对象在启动 webpack 时被一次性建立，并配置好所有可操作的设置，包括 options，loader 和 plugin。当在 webpack 环境中应用一个插件时，插件将收到此 compiler 对象的引用。可以使用它来访问 webpack 的主环境。

compilation 对象代表了一次资源版本构建。当运行 webpack 开发环境中间件时，每当检测到一个文件变化，就会创建一个新的 compilation，从而生成一组新的编译资源。一个 compilation 对象表现了当前的模块资源、编译生成资源、变化的文件、以及被跟踪依赖的状态信息。compilation 对象也提供了很多关键时机的回调，以供插件做自定义处理时选择使用。

## **4. 基本插件架构**

* 插件是由「具有 apply 方法的 prototype 对象」所实例化出来的
* 这个 apply 方法在安装插件时，会被 webpack compiler 调用一次
* apply 方法可以接收一个 webpack compiler 对象的引用，从而可以在回调函数中访问到 compiler 对象

### **4.1 使用插件代码**

* [使用插件][https://github.com/webpack/webpack/blob/master/lib/webpack.js#L60-L69](https://github.com/webpack/webpack/blob/master/lib/webpack.js" \l "L60-L69))

**if** (options.plugins && Array.isArray(options.plugins)) {

**for** (**const** plugin **of** options.plugins) {

plugin.apply(compiler);

}

}

### **4.2 Compiler插件**

* [done: new AsyncSeriesHook(["stats"])](https://github.com/webpack/webpack/blob/master/lib/Compiler.js" \l "L105)

#### **4.2.1 同步**

**class** **DonePlugin** {

**constructor**(options) {

**this**.options = options;

}

apply(compiler) {

compiler.hooks.done.tap('DonePlugin', (stats) => {

console.log('Hello ', **this**.options.name);

});

}

}module.exports = DonePlugin;

#### **4.2.2 异步**

**class** **DonePlugin** {

**constructor**(options) {

**this**.options = options;

}

apply(compiler) {

compiler.hooks.done.tapAsync('DonePlugin', (stats, callback) => {

console.log('Hello ', **this**.options.name);

callback();

});

}

}module.exports = DonePlugin;

### **4.3 使用插件**

* 要安装这个插件，只需要在你的 webpack 配置的 plugin 数组中添加一个实例

**const** DonePlugin=require('./plugins/DonePlugin');module.exports={

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

output: {

path: path.resolve('build'),

filename:'bundle.js'

},

plugins: [

**new** DonePlugin({name:'zfpx'})

]

}

### **4.4 触发钩子执行**

* [done](https://github.com/webpack/webpack/blob/master/lib/Compiler.js" \l "L360-L363)

if (this.hooks.shouldEmit.call(compilation) === false) {

const stats = new Stats(compilation);

stats.startTime = startTime;

stats.endTime = Date.now();+ this.hooks.done.callAsync(stats, err => {+ if (err) return finalCallback(err);+ return finalCallback(null, stats);+ });

return;

}

## **5. compilation插件**

* 使用 compiler 对象时，你可以绑定提供了编译 compilation 引用的回调函数，然后拿到每次新的 compilation 对象。这些 compilation 对象提供了一些钩子函数，来钩入到构建流程的很多步骤中

### **5.1 asset-plugin.js**

plugins\asset-plugin.js

**class** **AssetPlugin** {

**constructor**(options) {

**this**.options = options;

}

apply(compiler) {

compiler.hooks.compilation.tap('AssetPlugin', **function** (compilation) {

compilation.hooks.chunkAsset.tap('AssetPlugin', **function** (chunk, filename) {

console.log('filename=', filename);

});

});

}

}module.exports = AssetPlugin;

### **5.2 compilation.call**

* [Compiler.js](https://github.com/webpack/webpack/blob/master/lib/Compiler.js" \l "L853-L860)

newCompilation(params) {

**const** compilation = **this**.createCompilation();

**this**.hooks.compilation.call(compilation, params);

**return** compilation;

}

### **5.3 chunkAsset.call**

* [chunkAsset.call](https://github.com/webpack/webpack/blob/v4.39.3/lib/Compilation.js" \l "L2019)

chunk.files.push(file);+this.hooks.chunkAsset.call(chunk, file);

关于 compiler, compilation 的可用回调，和其它重要的对象的更多信息，请查看 [插件](https://webpack.docschina.org/api/compiler-hooks/) 文档。

## **6. 打包zip**

* [webpack-sources](https://www.npmjs.com/package/webpack-sources)

### **6.1 zip-plugin.js**

* plugins\zip-plugin.js

**const** { RawSource } = require("webpack-sources");**const** JSZip = require("jszip");**const** path = require("path");**class** **ZipPlugin** {

**constructor**(options) {

**this**.options = options;

}

apply(compiler) {

**let** that = **this**;

compiler.hooks.emit.tapAsync("ZipPlugin", (compilation, callback) => {

**var** zip = **new** JSZip();

**for** (**let** filename **in** compilation.assets) {

**const** source = compilation.assets[filename].source();

zip.file(filename, source);

}

zip.generateAsync({ type: "nodebuffer" }).then(content => {

compilation.assets[that.options.filename] = **new** RawSource(content);

callback();

});

});

}

}module.exports = ZipPlugin;

### **6.2 webpack.config.js**

webpack.config.js

plugins: [+ new zipPlugin({+ filename: 'assets.zip'+ })

]

## **7.自动外链**

### **7.1 使用外部类库**

* 手动指定 external
* 手动引入 script

能否检测代码中的import自动处理这个步骤?

### **7.2 思路**

* 解决import自动处理external和script的问题，需要怎么实现，该从哪方面开始考虑
* 依赖 当检测到有import该library时，将其设置为不打包类似exteral,并在指定模版中加入script,那么如何检测import？这里就用Parser
* external依赖 需要了解external是如何实现的，webpack的external是通过插件ExternalsPlugin实现的，ExternalsPlugin通过tap NormalModuleFactory 在每次创建Module的时候判断是否是ExternalModule
* webpack4加入了模块类型之后，Parser获取需要指定类型moduleType,一般使用javascript/auto即可

### **7.3 使用plugins**

plugins: [

**new** HtmlWebpackPlugin({

template: './src/index.html',

filename:'index.html'

}),

**new** AutoExternalPlugin({

jquery: {

expose: '$',

url: 'https://cdn.bootcss.com/jquery/3.1.0/jquery.js'

}

})

]

### **7.4 AutoExternalPlugin**

* [ExternalsPlugin.js](https://github.com/webpack/webpack/blob/0d4607c68e04a659fa58499e1332c97d5376368a/lib/ExternalsPlugin.js)
* [ExternalModuleFactoryPlugin](https://github.com/webpack/webpack/blob/eeafeee32ad5a1469e39ce66df671e3710332608/lib/ExternalModuleFactoryPlugin.js)
* [ExternalModule.js](https://github.com/webpack/webpack/blob/eeafeee32ad5a1469e39ce66df671e3710332608/lib/ExternalModule.js)
* [parser](https://github.com/zhufengnodejs/webpack-analysis/blob/master/node_modules/_webpack@4.20.2@webpack/lib/NormalModuleFactory.js" \l "L87)
* [factory](https://github.com/zhufengnodejs/webpack-analysis/blob/master/node_modules/_webpack@4.20.2@webpack/lib/NormalModuleFactory.js" \l "L66)
* [htmlWebpackPluginAlterAssetTags](https://github.com/jantimon/html-webpack-plugin/blob/v3.2.0/index.js" \l "L62)

**const** ExternalModule = require('webpack/lib/ExternalModule');**class** **AutoExternalPlugin** {

**constructor**(options) {

**this**.options = options;

**this**.externalModules = {};

}

apply(compiler) {

//1.在解析语法树的过程中查找那些需要外部引入的模块名称

compiler.hooks.normalModuleFactory.tap('AutoExternalPlugin', normalModuleFactory => {

normalModuleFactory.hooks.parser

.for('javascript/auto')

.tap('AutoExternalPlugin', parser => {

parser.hooks.import.tap('AutoExternalPlugin', (statement, source) => {

**if** (**this**.options[source])

**this**.externalModules[source] = true;

});

});

//2.在生产模块的过程中发现如果是外部模块则返回外部模块

normalModuleFactory.hooks.factory.tap('AutoExternalPlugin', factory => (data, callback) => {

**const** dependency = data.dependencies[0];

**let** value = dependency.request;//jquery

**if** (**this**.externalModules[value]) {

**let** varName = **this**.options[value].expose;

callback(null, **new** ExternalModule(varName, 'window'));

} **else** {

factory(data, callback);

}

});

});

compiler.hooks.compilation.tap('AutoExternalPlugin', compilation => {

//3.向body底部插入全局变量的脚本

compilation.hooks.htmlWebpackPluginAlterAssetTags.tapAsync('normalModuleFactory', (htmlPluginData, callback) => {

Object.keys(**this**.externalModules).forEach(name => {

**let** src = **this**.options[name].url;

htmlPluginData.body.unshift({

tagName: 'script',

closeTag: true,

attributes: { type: 'text/javascript', src }

});

});

callback(null, htmlPluginData);

});

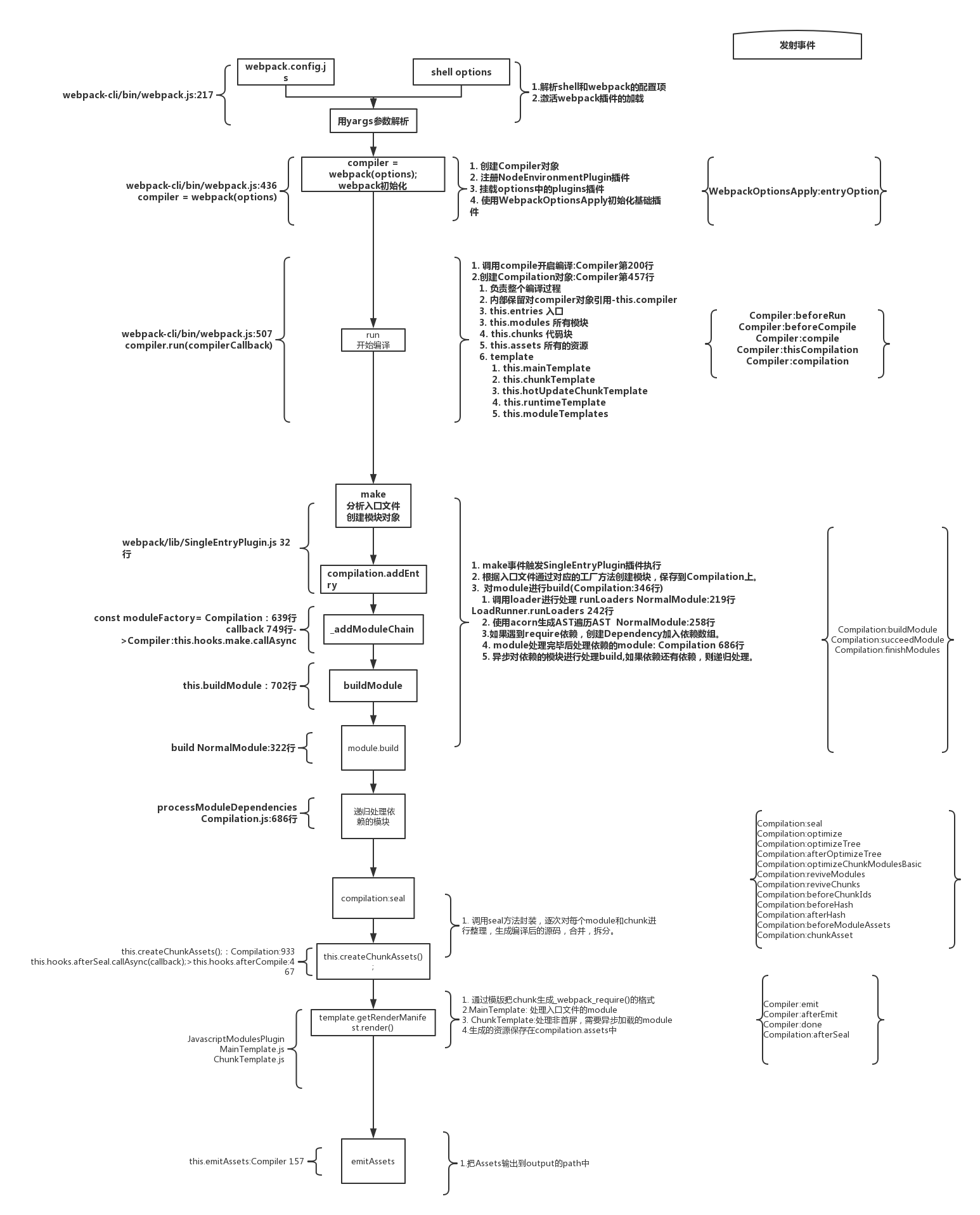
});

}

}module.exports = AutoExternalPlugin;

## **8. 参考**

* [Node.js SDK](https://developer.qiniu.com/kodo/sdk/1289/nodejs)
* [writing-a-plugin](https://webpack.js.org/contribute/writing-a-plugin/)
* [api/plugins](https://webpack.js.org/api/plugins/)



Powered by [idoc](https://github.com/jaywcjlove/idoc" \t "https://zhufeng-document.vercel.app/html/_blank). Dependence [Node.js](https://nodejs.org/) run.