

webpack代码优化实践




1. Webpack原理
2. JS tree-shaking
3. CSS tree-shaking

第一部分

Webpack原理

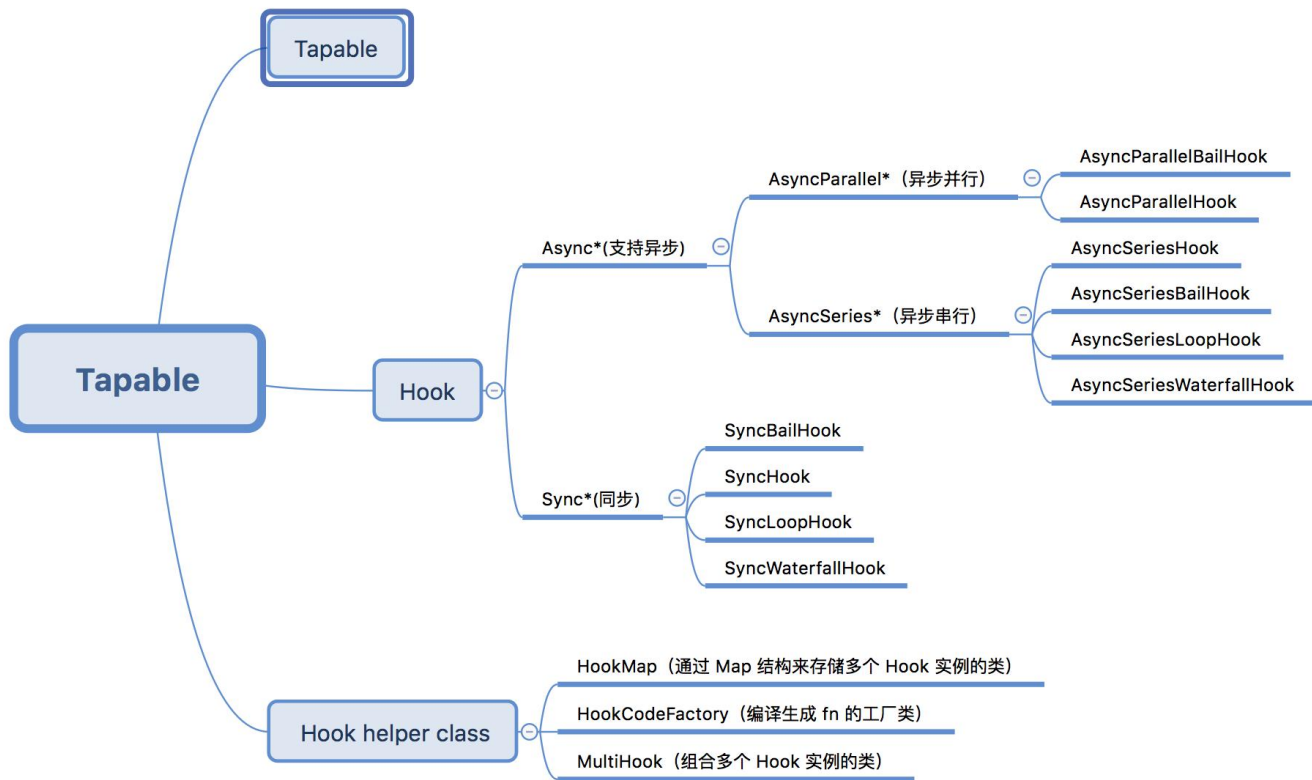
Webpack 核心模块 tapable

- Webpack有一句话Everything is a plugin。可见webpack是靠插件堆积起来的。而实现这个插件机制的就是Tapable
- Webpack 本质上是一种事件流的机制，它的工作流程就是将各个插件串联起来，而实现这一切的核心就是 tapable。tapable有点类似EventsBus库，核心原理也是发布订阅模式，Webpack 中最核心的，负责编译的 Compiler 和负责创建 bundles 的 Compilation 都是 tapable 构造函数的实例。
- 打开 Webpack 4.0 的源码中一定会看到下面这些以 Sync、Async 开头，以 Hook 结尾的方法，这些都是 tapable 核心库的类，为我们提供不同的事件流执行机制，我们称为“钩子”。
- webpack 整个编译过程中暴露出来大量的 Hook 供内部/外部插件使用，同时支持扩展各种插件，而内部处理的代码，也依赖于 Hook 和插件，这部分的功能就依赖于 Tapable。webpack 的整体执行过程，总的来看就是事件驱动的。从一个事件，走向下一个事件。扩展自 tapable 的对象内部会有很多钩子，它们贯穿了 webpack 构建的整个过程。



```
const {
  SyncHook,
  SyncBailHook,
  SyncWaterfallHook,
  SyncLoopHook,
  AsyncParallelHook,
  AsyncParallelBailHook,
  AsyncSeriesHook,
  AsyncSeriesBailHook,
  AsyncSeriesWaterfallHook
} = require('tapable')
```

下面的实现事件流机制的“钩子”大方向可以分为两个类别，“同步”和“异步”，“异步”又分为两个类别，“并行”和“串行”，而“同步”的钩子都是串行的。



➤ Sync 类型的钩子

- **SyncHook**：串行同步执行，不关心事件处理函数的返回值，在触发事件之后，会按照事件注册的先后顺序执行所有的事件处理函数。
- **SyncBailHook**：同样为串行同步执行，如果事件处理函数执行时有一个返回值不为空，则跳过剩下未执行的事件处理函数。
- **SyncWaterfallHook**：为串行同步执行，上一个事件处理函数的返回值作为参数传递给下一个事件处理函数，依次类推，正因如此，只有第一个事件处理函数的参数可以通过 call 传递，而 call 的返回值为最后一个事件处理函数的返回值。
- **SyncLoopHook**：为串行同步执行，事件处理函数返回 true 表示继续循环，即循环执行当前事件处理函数，返回 undefined 表示结束循环，SyncLoopHook 与 SyncBailHook 的循环不同，SyncBailHook 只决定是否继续向下执行后面的事件处理函数，而 SyncLoopHook 的循环是指循环执行每一个事件处理函数，直到返回 undefined 为止，才会继续向下执行其他事件处理函数，执行机制同理。

注意：在 Sync 类型“钩子”下执行的插件都是顺序执行的，只能使用 tab 注册。

➤ Async 类型的钩子

- AsyncParallelHook: AsyncParallelHook 为异步并行执行, 通过 tapAsync 注册的事件, 通过 callAsync 触发, 通过 tapPromise 注册的事件, 通过 promise 触发 (返回值可以调用 then 方法)。
- AsyncSeriesHook: 为异步串行执行, 与 AsyncParallelHook 相同, 通过 tapAsync 注册的事件, 通过 callAsync 触发, 通过 tapPromise 注册的事件, 通过 promise 触发, 可以调用 then 方法。

在上面 Async 异步类型的 “钩子中”, 我们只着重介绍了 “串行” 和 “并行” (AsyncParallelHook 和 AsyncSeriesHook) 以及回调和 Promise 的两种注册和触发事件的方式, 还有一些其他的具有一定特点的异步 “钩子” 我们并没有进行分析, 因为他们的机制与同步对应的 “钩子” 非常的相似。

注意: Async 类型可以使用 tap、tapAsync 和 tapPromise 注册不同类型的插件 “钩子”, 分别通过 call、callAsync 和 promise 方法调用, 我们下面会针对 AsyncParallelHook 和 AsyncSeriesHook 的 async 和 promise 两种方式分别介绍和模拟。

参考:

[webpack-tapable-2.0 的源码分析](#)
[脑壳疼的Webpack-tapable](#)

➤ 两个最重要的类 Compiler 与 Compilation

Compiler: 可以简单的理解为 Webpack 实例，它包含了当前 Webpack 中的所有配置信息，如 options, loaders, plugins 等信息，全局唯一，只在启动时完成初始化创建，随着生命周期逐一传递；

Compilation: 可以称为 编译实例。当监听到文件发生改变时，Webpack 会创建一个新的 Comilation 对象，开始一次新的编译。它包含了当前的输入资源，输出资源，变化的文件等，同时通过它提供的 api，可以监听每次编译过程中触发的事件钩子；

➤ webpack构建流程

➤ webpack是如何运行的？

1. webpack会解析所有模块，如果模块中有依赖其他文件，那就继续解析依赖的模块。直到文件没有依赖为止。
2. 解析结束后，webpack会把所有模块封装在一个函数里，并放入一个名为modules的数组里。
3. 将modules传入一个自执行函数中，自执行函数包含一个installedModules对象，已经执行的代码模块会保存在此对象中。
4. 最后自执行函数中加载函数（webpack__require）载入模块。

➤ 它如何与webapck插件关联呢?

DEMO: tapable-webpack文件夹

➤ plugin和loader的区别是什么?

➤ 写一个插件:

(1) 去除webpack打包生成js中多余的注释

(2) 扩展 HtmlwebpackPlugin 插入自定义的脚本

[写一个webpack内联代码插件](#)

➤ 写一个loader:

[撸一个webpack-loader&plugin](#)

第二部分

JS Tree-shaking

➤ DCE

DCE，即死码消除，编译器原理中，死码消除（Dead code elimination）是一种编译最优化技术，它的用途是移除对程序运行结果没有任何影响的代码。移除这类的代码有两种优点，不但可以减少程序的大小，还可以避免程序在运行中进行不相关的运算行为，减少它运行的时间。不会被运行到的代码（unreachable code）以及只会影响到无关程序运行结果的变量（Dead Variables），都是死码（Dead code）的范畴。

通过覆盖率测试就可以发现所有的永远都不会执行到的代码，常见的覆盖率测试工具AQtime, JaCoCo, coverage.py, ScriptCover, Istanbul和JSCover等。大部分工作原理都是经过“代码插桩”-“覆盖率信息收集”-“生成覆盖率报告”这三个步骤。

➤ 死码具有以下几个特征：

- 代码不会被执行，不可到达
- 代码执行的结果不会被用到
- 代码只会影响死变量（只写不读）

```
let foo = ()=>{  
  let x=1;  
  if(false){  
    console.log("无效代码");  
  }  
  
  function getUser(){  
    return 100;  
  }  
  
  let a=3;  
  return a;  
  
  let b=3;  
  return b;  
}  
  
foo();
```

➤ 什么是Tree-shaking

可以理解为通过工具"摇"我们的JS文件，将其中用不到的代码"摇"掉，是一个性能优化的范畴。具体来说，在 webpack 项目中，有一个入口文件，相当于一棵树的主干，入口文件有很多依赖的模块，相当于树枝。实际情况中，虽然依赖了某个模块，但其实只使用其中的某些功能。通过 tree-shaking，将没有使用的模块摇掉，这样来达到删除无用代码的目的。

➤ 支持Tree-shaking的工具

rollup , webpack , google closure compiler (java) , Parcel (不支持)

➤ Tree-shaking的原理

Tree-shaking的本质是消除无用的js代码。无用代码消除在广泛存在于传统的编程语言编译器中，编译器可以判断出某些代码根本不影响输出，然后消除这些代码，这个称之为DCE (dead code elimination) 。

Tree-shaking 是 DCE 的一种新的实现，Javascript同传统的编程语言不同的是，javascript绝大多数情况需要通过网络进行加载，然后执行，加载的文件大小越小，整体执行时间更短，所以去除无用代码以减少文件体积，对javascript来说更有意义。

Tree-shaking 和传统的 DCE的方法又不太一样，传统的DCE 消灭不可能执行的代码，而Tree-shaking 更关注于消除没有用到的代码。

传统编译型的语言中，都是由编译器将Dead Code从AST（抽象语法树）中删除，那javascript中是由谁做DCE呢？

首先肯定不是浏览器做DCE，因为当我们的代码送到浏览器，那还谈什么消除无法执行的代码来优化呢，所以肯定是送到浏览器之前的步骤进行优化。

其实也不是上面提到的三个工具，rollup，webpack，gcc做的，而是著名的代码压缩优化工具uglify，uglify完成了javascript的DCE。

为什么tree-shaking是最近几年流行起来了？而前端模块化概念已经有很多年历史了，其实tree-shaking的消除原理是依赖于ES6的模块特性。

ES6 module 特点：

- 只能作为模块顶层的语句出现
- import 的模块名只能是字符串常量
- import binding 是不可改变的

ES6模块依赖关系是确定的，和运行时的状态无关，可以进行可靠的静态分析，这就是tree-shaking的基础。

所谓静态分析就是不执行代码，从字面上对代码进行分析，ES6之前的模块化，比如我们可以动态require一个模块，只有执行后才知道引用的什么模块，这个就不能通过静态分析去做优化。

这是 ES6 modules 在设计时的一个重要考量，也是为什么没有直接采用 CommonJS，正是基于这个基础上，才使得 tree-shaking 成为可能，这也是为什么 rollup 和 webpack 都要用 ES6 module 语法才能 tree-shaking。

ES6 module 语法依赖关系是确定的，和运行时的状态无关，可以进行可靠的静态分析，然后进行消除！

```
import {post} from './util.js'    //tree-shaking有效
import menu from './menu.js'     //tree-shaking无效
```

- 只处理函数和顶层的import/export变量，不能把没用到的类的方法消除掉
- javascript动态语言的特性使得静态分析比较困难
- <https://github.com/rollup/rollup/issues/349>
- rollup不需要配置插件就可以进行tree-shaking，而webpack要实现tree-shaking必须依赖uglifyJs
- Tree-shaking对函数效果较好

➤ Tree shaking到底能做哪些事情?

Webpack Tree shaking从ES6顶层模块开始分析, 可以清除未使用的模块

会对多层调用的模块进行重构, 提取其中的代码, 简化函数的调用结构

Tree shaking不会清除IIFE(立即调用函数表达式)

Tree shaking对于IIFE的返回函数, 如果未使用会被清除

➤ Webpack Tree shaking做不到的事情

对引用的第三方库做不了代码消除

解决: 使用这个插件webpack-deep-scope-analysis-plugin解决

参考:

[解密webpack tree-shaking](#)

[你的Tree-Shaking并没什么卵用](#)

[体积减少80%! 释放webpack tree-shaking的真正潜力](#)

```
import _ from 'lodash';  
Asset      Size  
bundle.js  70.5 KiB
```

```
import { last } from 'lodash';  
Asset      Size  
bundle.js  70.5 KiB
```

//这种引用方式明显降低了打包后的大小

```
import last from 'lodash/last';  
Asset      Size  
bundle.js  1.14 KiB
```

➤ GCC

Google Closure Compiler是一个使JavaScript下载和运行更快的工具。它不是从源语言编译到机器代码，而是从JavaScript编译成更好的JavaScript。它解析您的JavaScript，分析它，删除死代码并重写并最小化剩下的内容。它还检查语法，变量引用和类型，并警告常见的JavaScript陷阱。

在线工具: <https://closure-compiler.appspot.com/home>

github : <https://github.com/google/closure-compiler>

GCC提供三种压缩模式:

- Whitespace only (只是简单的去除空格换行注释)
- Simple (对局部变量的变量名进行缩短, 和Uglify类似)
- Advanced (破坏级压缩, 对原有代码结构分析, 重写, 破坏, 代码量减小到了极致)

参考: [Google Closure Compiler高级压缩混淆Javascript代码](#)

Closure Compiler

Add a URL:

Add

Example: <http://www.example.com/bigfile.js>

Optimization: ☐ Whitespace only ☐ Simple ☒ Advanced

[Which optimization is right for my code?](#)

Formatting: ☐ Pretty print ☐ Print input delimiter

Compile

[Reset](#)

```
var foo = "1234";  
alert(window.foo);
```

```
let baz = () => {  
  console.log(foo);  
  post()  
  var l=100  
  var x = "我是无效变量1"  
  console.log(x)  
  function unused () {  
    return "我是无效函数"  
  }  
  return 1;  
}  
  
baz()
```

Compilation was a success!

Original Size: 189 bytes gzipped (225 bytes uncompressed)

Compiled Size: 95 bytes gzipped (96 bytes uncompressed)

Saved 49.74% off the gzipped size (57.33% without gzip)

The code may also be accessed at [default.js](#).

Compiled Code

Warnings

Errors

POST data

```
alert(window.a);console.log("1234");post();console.log("\u6211\u662f\u65e0\u6548\u53d8\u91cf1");
```

closure compiler (java) 是最好的, 但对代码具有侵入性, 且与我们日常的基于JS的开发流很难兼容。

➤ 如果要更好的使用Tree shaking,请满足:

- 使用ES2015(ES6)开发模块
- 避免使用IIFE
- 如果使用第三方的模块, 可以尝试直接从文件路径引用的方式使用, 或者使用es版
- 对第三方的库, 团队的维护的: 视情况加上`sideEffects`标记, 同时更改Babel配置来导出ES6模块

➤ 现状:

各种工具对tree-shaking的支持都不够完美! 目前只能通过约束代码规范来保证!

tree-shaking对web意义重大, 是一个极致优化的理想世界, 是前端进化的又一个终极理想。

理想是美好的, 但目前还处在发展阶段, 还比较困难, 有各个方面的, 甚至有目前看来无法解决的问题, 但还是应该相信新技术能带来更好的前端世界。

➤ 对组件库引用的优化

当我们使用组件库的时候，`import {Button} from 'element-ui'`，相对于`Vue.use(elementUI)`，已经是具有性能意识，是比较推荐的做法，但如果我们写成右边的形式，具体到文件的引用，打包之后的区别是非常大的，以`antd`为例，右边形式`bundle`体积减少约80%。

这个引用也属于有副作用，`webpack`不能把其他组件进行`tree-shaking`。既然工具本身是做不了，那我们可以做工具把左边代码自动改成右边代码这种形式。这个工具`antd`库本身也是提供的。

<https://ant.design/docs/react/use-with-create-react-app-cn>

<https://github.com/lin-xi/babel-plugin-import-fix>

原理：使用 `babel-plugin-import`，通过核心`babylon`将ES6代码转换成AST抽象语法树，然后插件遍历语法树找出类似 `import {Button} from 'antd'` 这样的语句，进行转换 `import {Button} from 'antd/lib/button'`，最后重新生成代码替换。

➤ tree-shaking 简单模拟

```
import { Button, Alert } from 'element-ui';
```

这样引用资源，Webpack 在打包的时候会找到 element-ui 并把里面所有的代码全部打包到出口文件，我们只使用了两个组件，全部打包不是我们所希望的，tree-shaking 是通过在 Webpack 中配置 babel-plugin-import 插件来实现的，它可以将解构的代码转换成下面的形式：

```
import Button from 'element-ui/lib/button';  
import Alert from 'element-ui/lib/Alert';
```

转化后会去 node_modules 中的 element-ui 模块找到 Button 和 Alert 两个组件对应的文件，并打包到出口文件中。

通过上面的转换可以看出，其实 tree-shaking 的实现原理是通过改变 AST 语法树的结构来实现的我们可以通过在线转换网站 <http://esprima.org/demo/parse.html> 将 JS 代码装换成 AST 语法树。

实现：<https://github.com/Deturium/babel-plugin-my-import>

第三部分

CSS tree-shaking

➤ CSS Tree-shaking

前面所说的tree-shaking都是针对js文件，通过静态分析，尽可能消除无用的代码，那对于css如何做tree-shaking?

➤ 对css进行tree-shaking 2种思路:

- 遍历所有的css文件中的selector选择器，然后去所有js代码中匹配，如果选择器没有在代码出现过，则认为该选择器是无用代码。

<https://github.com/lin-xi/webpack-css-treeshaking-plugin>

缺点：如果在js里面对dom的操作（例如对dom增加一个class样式等操作）

- 找到那些一定不会被用到的选择器，去掉这些即可

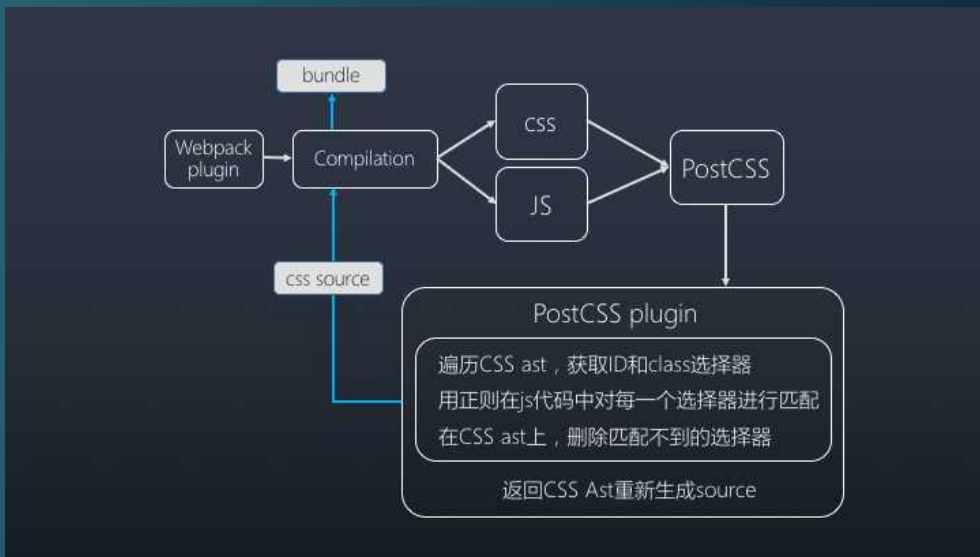
在html或者js被引用的话，那么html或者js代码里面一定会出现这个选择器的所有单词。如果没有出现或者没有全部出现的话，证明这个选择器是没有被用到的。

使用PurifyCSSPlugin，需要配合 extract-text-webpack-plugin 使用。**缺点：去除不彻底**

➤ PostCSS

PostCSS是一款使用插件去转换CSS的工具，有许多非常好用的插件，例如autoprefixer,cssnext以及CSS Modules。而上面列举出的这些特性，都是由对应的postcss插件去实现的。而使用PostCSS则需要与webpack或者parcel结合起来。

PostCSS 提供了一个解析器，它能够将 CSS 解析成AST抽象语法树。然后我们能写各种插件，对抽象语法树做处理，最终生成新的css文件，以达到对css进行精确修改的目的。



➤ PostCSS Tree-shaking 原理

- 插件监听webpack编译完成事件，webpack编译完成之后，从compilation中找出所有的css文件和js文件
- 将所有的css文件送至postCss处理，找出无用代码
- postCss 遍历，匹配，删除过程
- checkRule 处理每一个规则核心代码

➤ purifycss [坑太多]

- 必须要配合extract-text-webpack-plugin
- 消除未使用的CSS
- 比如我们在使用bootstrap的时候，他127KB但是实际上我们不需要那么多，要是使用了这个插件就只留下我们用到的样式，会明显减小css文件大小

➤ 使用: 用到了3个包 一个是 purifycss-webpack 一个是 purify-css 和 glob-all (路径匹配)

```
//用来抽离CSS单独打包
let extractTextPlugin = new ExtractTextPlugin({
  filename: "[name].min.css",
  allChunks: false
});
//引入插件tree-shaking, 必须要配合extract-text-webpack-plugin使用
let purifyCSS = new PurifyCSS({
  paths: glob.sync([
    // 要做CSS Tree Shaking的路径文件
    path.resolve(__dirname, ".*.html"),
    path.resolve(__dirname, ".*src/*.js")
  ])
});
```

```
import base from "./css/base.css";
import bootstrap from "./css/bootstrap.css";

var app = document.getElementById("app");
var div = document.createElement("div");
div.className = "box container";
app.appendChild(div);
```

```
Hash: f76a0835126aabec4060
Version: webpack 4.31.0
Time: 2795ms
Built at: 2019-05-19 15:13:40
```

	Asset	Size	Chunks		Chunk Names
app.bundle.js	1.13 KiB	0	[emitted]	app	app
app.min.css	1.66 KiB	0	[emitted]	app	app

```
Entrypoint app = app.bundle.js app.min.css
[0] ./src/app.js 219 bytes {0} [built]
```

➤ 编码规范的约束，插件才能更好的工作

CSS Tree-shaking 无效:

```
render(){  
  this.stateClass = 'state-' + this.state == 2 ? 'open' : 'close'  
  return <div class={this.stateClass}></div>  
}
```

CSS Tree-shaking 有效:

```
render(){  
  this.stateClass = this.state == 2 ? 'state-open' : 'state-close'  
  return <div class={this.stateClass}></div>  
}
```

注意：purifycss 功能现在还不完美，多数情况会误判，生产环境慎用！

THANK YOU