
webpack

目录

前端打包发展历史

为什么需要构建工具?

转换 ES6 语法

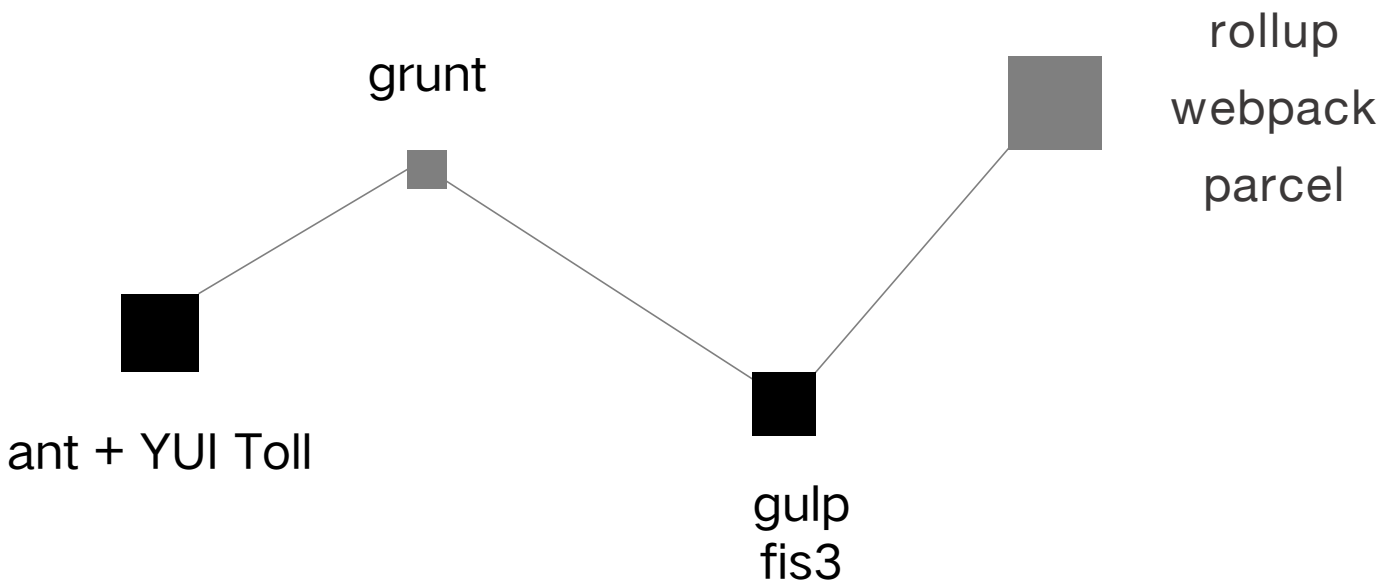
转换 JSX

CSS 前缀补全/预处理器

压缩混淆

ES6 module 主流浏览器支持情况

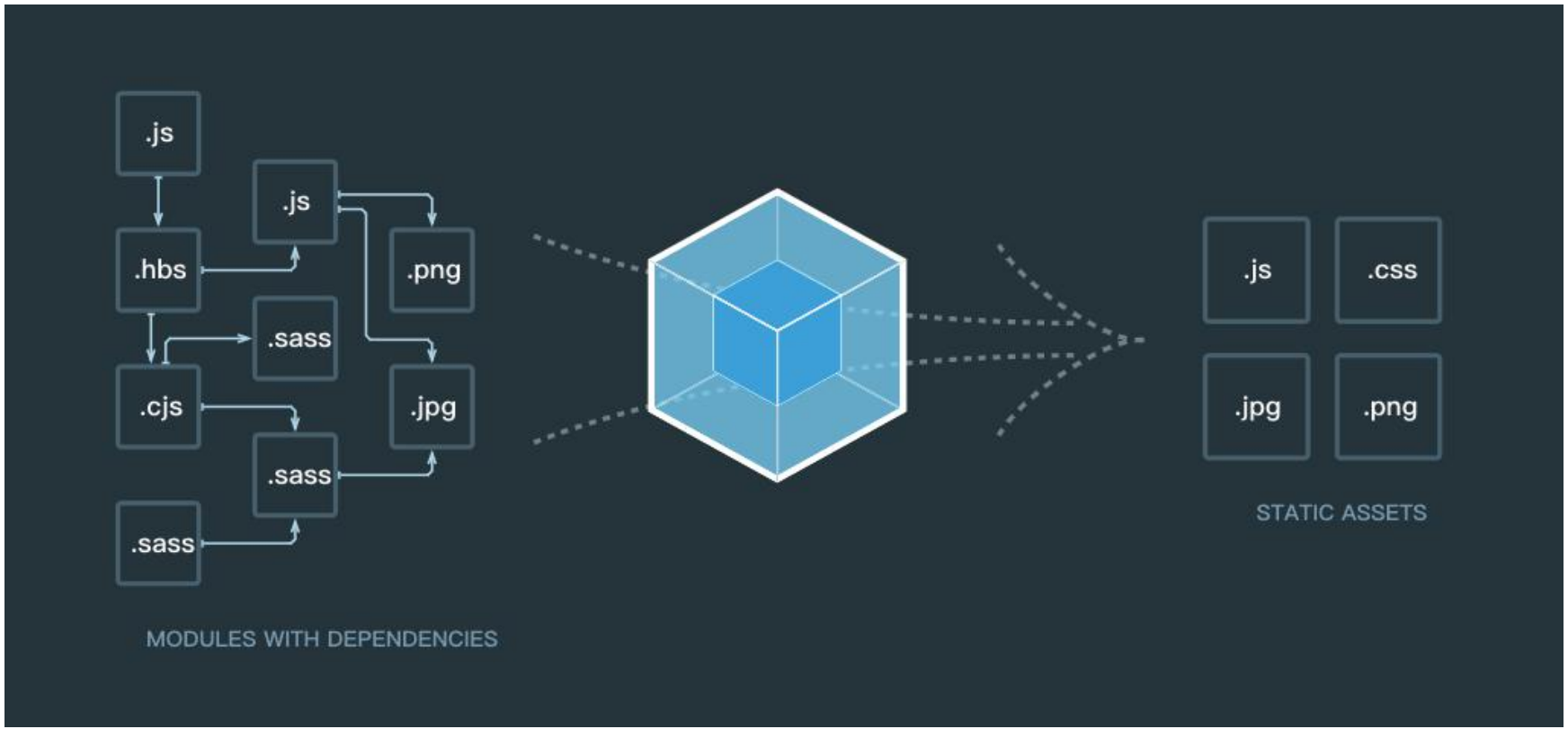
前端构建演变之路



为什么选择 webpack?

	webpack	grunt	glup
定义	Module bundler	Task runner	Task runner
语言	JavaScript	Node.js	Node.js
发布时间	2012.3	2012.6	2013.7
GitHub stars	54732	12017	31798
周下载量	3,385,392	478,876	816,228

webpack干了什么?



webpack 用法

配置文件名称

webpack 默认配置文件：webpack.config.js

可以通过 webpack --config 指定配置文件

webpack 配置组成

```
module.ex
  entry: './
  output: '
  mode: 'p
  module:
    rules:
      { tes
    ]
  },
  plugins:
    new H
      temp
    })
  ]
```

名称	功能
CommonsChunkPlugin	将chunks相同的模块提取成公共js
CleanWebpackPlugin	清理构建目录
ExtractTextWebpackPlugin	将css提取成独立的文件
CopyWebpackPlugin	将文件或文件夹拷贝到构建目录
HtmlWebpackPlugin	创建html文件承载输出的bundle
UglifyjsWebpackPlugin	压缩混淆js
ZipWebpackPlugin	将打包结果生成一个zip包

通过 npm script 运行 webpack

```
{
  "name": "hello-webpack",
  "version": "1.0.0",
  "description": "Hello webpack",
  "main": "index.js",
  "scripts": {
    "build": "webpack"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

通过 npm run build 运行构建

原理：模块局部安装会在 node_modules/.bin 目录创建软链接

webpack 源码分析

从webpack命令行开始分析

通过 npm scripts 运行 webpack

- 开发环境： npm run dev
- 生产环境：npm run build

通过 webpack 直接运行

- webpack entry.js build.js

分析 webpack 的入口文件：webpack.js

```
process.exitCode = 0;                                    //1. 正常执行返回
const runCommand = (command, args) =>{...};    //2. 运行某个命令
const isInstalled = packageName =>{...};              //3. 判断某个包是否安装 const
CLIs =[...];              //4. webpack 可用的 CLI: webpack-cli 和 webpack-command

const installedClis = CLIs.filter(cli => cli.installed);  //5. 判断是否两个 CII 是否安装了
if (installedClis.length === 0){...}else if          //6. 根据安装数量进行处理
(installedClis.length === 1){...}else{...}.
```

启动后的结果

webpack 最终找到 webpack-cli (webpack-command) 这个 npm 包，
并且 执行 CLI

webpack cli 所做的事

引入 yargs，对命令行进行定制

分析命令行参数，对各个参数进行转换，组成编译配置项

引用webpack，根据配置项进行编译和构建

webpack 的本质

Webpack可以将其理解是一种基于事件流的编程范例，一系列的插件运行。

核心对象 Compiler 继承 Tapable
class Compiler extends Tapable { // ... }

核心对象 Compilation 继承 Tapable
class Compilation extends Tapable { // ... }

Tapable 是什么

Tapable 是一个类似于 Node.js 的 EventEmitter 的库, 主要是控制钩子函数的发布 与订阅,控制着 webpack 的插件系统。

Tapable库暴露了很多 Hook（钩子）类，为插件提供挂载的钩子

```
const {
  SyncHook,                    //同步钩子
  SyncBailHook,                //同步熔断钩子
  SyncWaterfallHook,           //同步流水钩子
  SyncLoopHook,                //同步循环钩子
  AsyncParallelHook,           //异步并发钩子
  AsyncParallelBailHook,       //异步并发熔断钩子
  AsyncSeriesHook,             //异步串行钩子
  AsyncSeriesBailHook,         //异步串行熔断钩子
  AsyncSeriesWaterfallHook     //异步串行流水钩子
} = require("tapable");
```

Tapable 的使用-钩子的绑定与执行

Tabpack 提供了同步&异步绑定钩子的方法，并且他们都有绑定事件和执行事件对 应的方法。

Async*	Sync*
绑定： tapAsync/tapPromise/tap	绑定： tap
执行： callAsync/promise	执行： call

Tapable 的使用-hook 基本用法示例

```
const hook1 = new SyncHook(["arg1", "arg2", "arg3"]);

//绑定事件到webapck事件流
hook1.tap('hook1', (arg1, arg2, arg3) => console.log(arg1, arg2, arg3))

//执行绑定的事件
hook1.call(1,2,3)
```

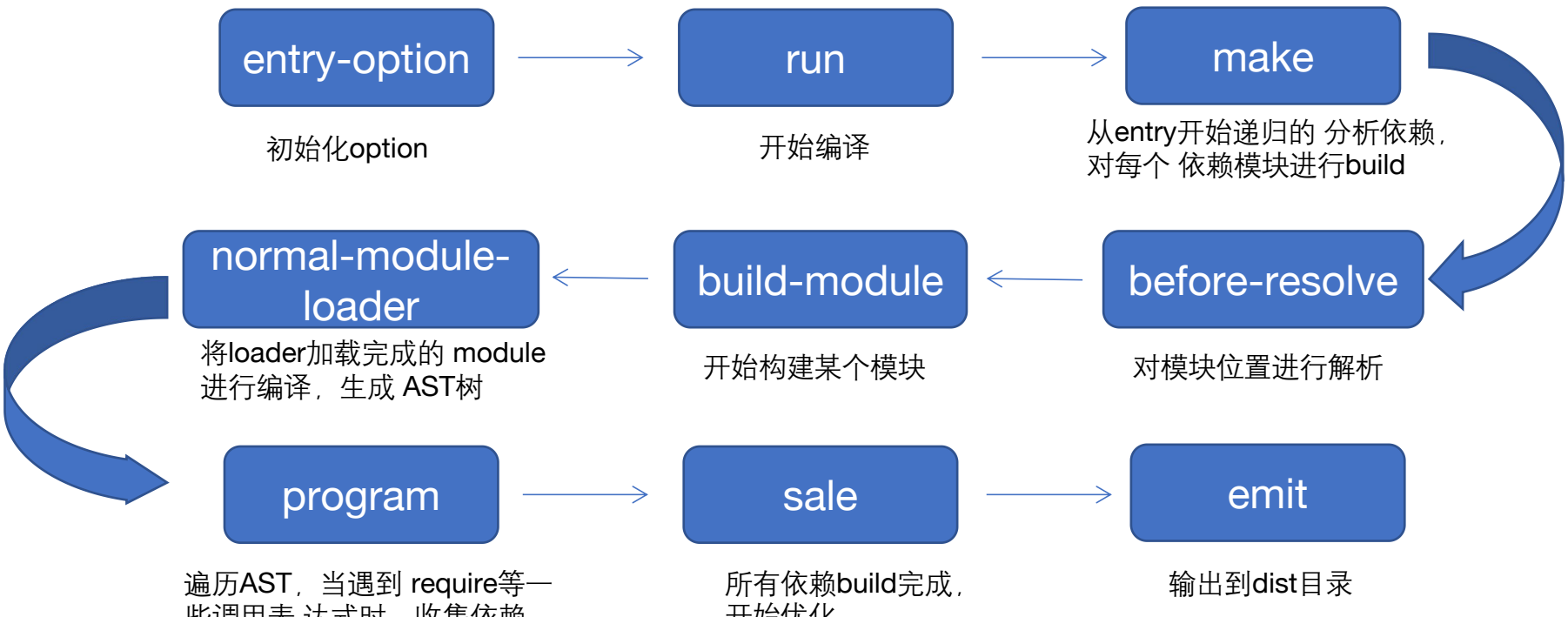
Tapable 是如何和 webpack 联系起来的?

```
let compiler;
if (Array.isArray(options)) {
  compiler = new MultiCompiler(options.map(options => webpack(options)));
} else if (typeof options === "object") {
  options = new WebpackOptionsDefaulter().process(options);

  compiler = new Compiler(options.context);
  compiler.options = options;
  new NodeEnvironmentPlugin().apply(compiler);
  if (options.plugins && Array.isArray(options.plugins)) {
    for (const plugin of options.plugins) {
      if (typeof plugin === "function") {
        plugin.call(compiler, compiler);
      } else {
        plugin.apply(compiler);
      }
    }
  }
  compiler.hooks.environment.call();
  compiler.hooks.afterEnvironment.call();
  compiler.options = new WebpackOptionsApply().process(options, compiler);
} else {
  throw new Error("Invalid argument: options");
}
```

webpack的打包流程

webpack的编译是按照下面的钩子调用顺序执行



感谢观看