# 02 什么是 Webpack

更新时间：2019-06-14 14:37:49


> 勤能补拙是良训，一分辛劳一分才。
>
> ——华罗庚

随着 Web 前端的不断发展，传统网页开发在逐渐往 Web 应用（Web Application，简称 WebAPP）的开发方式转变，页面开始变得越来越复杂，复杂的应用场景必然引起技术的进步，出现新的技术手段来解决现有问题。前端模块化和工程化的呼声越来越高，随着前些年大行其道的 Grunt、Gulp、FIS 等构建工具的发展，带动了一波前端工程化热。近几年，经过 React、Vue 库这些年的扩张，大型 Webapp 不再局限于手写 jQuery 操作 DOM，让大型 Webapp 有了全新的开发体验。在这个过程中，前端逐渐发展成了模块化和单页应用（single-page application，简称 SPA）为主的形式，在这种形态和 React、Vue 这些库的普及下，Webpack 越来越被更多成为主流构建工具。

## 模块化

说起 Webpack，大家都知道这是一个模块化构建工具，那么究竟什么是模块化呢？

> 模块化是指解决一个复杂问题时自顶向下逐层把系统划分成若干模块的过程，有多种属性，分别反映其内部特性。（百度百科）

模块化被越来越多的应用到我们的日常生活中，在我的印象中，小时候家电（比如收音机、电视）坏了都是拿到店里去找老师傅维修，现在家电都是模块化的，检测下哪里坏了直接换个新模块就可以了，由此可见，模块化不仅仅是个前端概念（相反，前端模块化也是这些年刚刚被得到重视的），我们生活场景中大量的充斥着模块化，让我们的生活效率更高。

前端模块化一般指得是 JavaScript 的模块，最常见的是 Nodejs 的 NPM 包，每个模块可能是最小甚至是最优的代码组合，也可能是解决某些问题有很多特定模块组成的大模块。如果没有模块化，可能大家编写代码当中遇见最多的就是复制（copy），当我们需要某个功能的代码时，自己由之前在哪个项目写过，那么我们就会 copy 过来，copy 多了，自然代码的可维护性就会下降。

模块化之后的代码，我们考虑更多的代码使用和维护成本的问题。所以有了很多模块化的规范：`CommonJS`、`AMD`和`ES6 Module`规范（另外还有`CMD`、`UMD`等）。

- CommonJS：是 Nodejs 广泛使用的一套模块化规范，是一种**同步**加载模块依赖的方式；
  - `require`：引入一个模块
  - `exports`：导出模块内容
  - `module`：模块本身
- AMD：是 js 模块加载库RequireJS提出并且完善的一套模块化规范，AMD 是一条**异步**加载模块依赖的方式；
  - id：模块的 id
  - dependencies：模块依赖
  - factory：模块的工厂函数，即模块的初始化操作函数
  - require：引入模块
- ES6 Module：ES6 推出的一套模块化规范。
  - import：引入模块依赖
  - export：模块导出

> Tips：除了上面三大主流规范，还有 CMD（国产库 SeaJS 提出来的一套模块规范）和 UMD（兼容 CommonJS 和 AMD 一套规范）。目前多数模块的封装，是既可以在 Node.js 环境又可以在 Web 环境运行，那么一般会采用 UMD 的规范，后面 Webpack 针对 lib 库的封装会有进一步介绍。

除了 JavaScript 的模块化，在 CSS 样式中也可以采用`@import`的方式来引入自己依赖的模块：

```
@import 'reset.css';
```

在一些 Less 或者 Sass 这些 CSS 预处理语言中`@import`还可以用来导入一些变量、函数和`mixin`的定义。

> Tips：大家可能也经常听到组件化这个名词，模块化一般指的是可以被抽象封装的最小/最优代码集合，模块化解决的是功能耦合问题；组件化则更像是模块化进一步封装，根据业务特点或者不同的场景封装出具有一定功能特性的独立整体；另外，前端提到组件化更多的是具有模板、样式和 js 交互的 UI 组件。

关于模块化开发相关的内容，在「Webpack 模块化开发」部分会继续详细介绍。

## 工程化

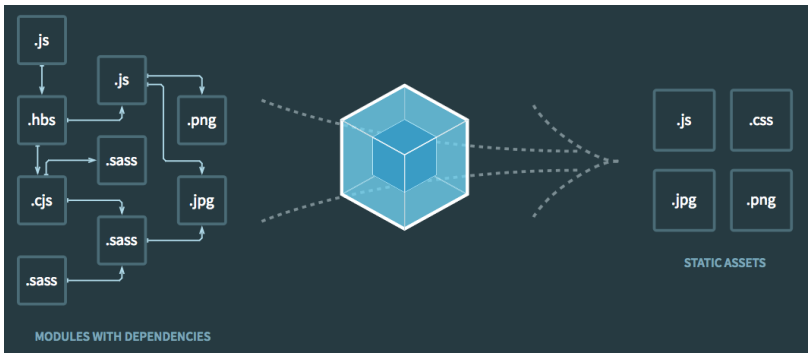
当我们开发的 Web 应用越来越复杂的时候，会发现我们面临的问题会逐渐增多：

1. 模块多了，依赖管理怎么做；
2. 页面复杂度提升之后，多页面、多系统、多状态怎么办；

3. 团队扩大之后，团队合作怎么做；
4. 怎么解决多人研发中的性能、代码风格等问题；
5. 权衡研发效率和产品迭代的问题。

这些问题就是软件工程需要解决的问题。工程化的问题需要运用工程化工具来解决，得益于 Nodejs 的发展，前端这些年在工程化上取得了不俗的成绩。前端工程化早期，是以 Grunt、Gulp 等构建工具为主的阶段，这个阶段解决的是重复任务的问题，它们将某些功能拆解成固定步骤的任务，然后编写工具来解决，比如：图片压缩、地址添加 hash、替换等，都是固定套路的重复工作。

而现阶段的 Webpack 则更像是从一套解决 JavaScript 模块化依赖打包开始，利用强大的插件机制，逐渐解决前端资源依赖管理问题，依附社区力量逐渐进化成一套前端工程化解决方案。




## 什么是 webpack


> 本质上，Webpack 是一个现代 JavaScript 应用程序的静态模块打包器（static module bundler）。在 Webpack 处理应用程序时，它会在内部创建一个依赖图（dependency graph），用于映射到项目需要的每个模块，然后将所有这些依赖生成到一个或多个 bundle。

## webpack 解决什么问题？

像 Grunt、Gulp 这类构建工具，打包的思路是：`遍历源文件`→`匹配规则`→`打包`，这个过程中**做不到按需加载**，即对于打包起来的资源，到底页面用不用，打包过程中是不关心的。

webpack 跟其他构建工具本质上不同之处在于：**webpack 是从入口文件开始，经过模块依赖加载、分析和打包三个流程完成项目的构建**。在加载、分析和打包的三个过程中，可以针对性的做一些解决方案，比如`code split`（拆分公共代码等）。

当然，Webpack 还可以轻松的解决传统构建工具解决的问题：

- 模块化打包，一切皆模块，JS 是模块，CSS 等也是模块；
- 语法糖转换：比如 ES6 转 ES5、TypeScript；
- 预处理器编译：比如 Less、Sass 等；
- 项目优化：比如压缩、CDN；
- 解决方案封装：通过强大的 Loader 和插件机制，可以完成解决方案的封装，比如 PWA；

- 流程对接：比如测试流程、语法检测等。

小结

本文从 Web 前端发展开始，介绍了模块化、工程化相关的概念和发展现状，最后介绍了 Webpack 的应用场景，以及为什么有了 Grunt、Gulp 这里传统构建工具，社区有产生 Webpack 这种构建工具。

> 本小节 Webpack 相关面试题：
>
> 1. Webpack 与 Grunt、Gulp 这类打包工具有什么不同？
> 2. 与 Webpack 类似的工具还有哪些？谈谈你为什么选择（或放弃）使用 Webpack？

课程的源代码下载　获取源代码

01 开篇词 | 使用 Webpack 实现前端工程化

03 Webpack 开发环境搭建