# 邂逅Webpack

王红元
coderwhy


实力IT教育

# 前端发展的几个阶段

- 无论是作为专业的开发者还是接触互联网的普通人，其实都能深刻的感知到Web前端的发展是非常快速的
  - 对于开发者来说我们会更加深有体会；
  - 从后端渲染的JSP、PHP，到前端原生JavaScript，再到jQuery开发，再到目前的三大框架Vue、React、Angular；
  - 开发方式也从原来的JavaScript的ES5语法，到ES6、7、8、9、10，到TypeScript，包括编写CSS的预处理器less、scss等；

## Web早期

互联网发展早期，前端只负责写静态页面，纯粹的展示功能，JavaScript的作用也只是进行一些表单的验证和增加特效。
当然为了动态在页面中填充一些数据，也出现了JSP、ASP、PHP这样的开发模式。

## Web近期

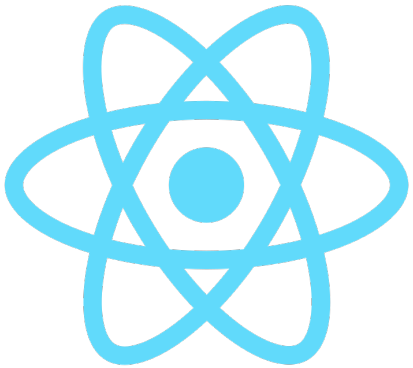
随着AJAX技术的诞生改变了前端的发展历史，使得开发者发现，前端的作用不仅仅是展示页面，可以管理数据以及和用户互动。
由于用户交互、数据交互的需求增多，也让jQuery 这样优秀的前端工具库大放异彩。

## 现代Web

而现代的Web开发事实上变得更加多样化和复杂化。开发的多样性包括我们需要开发PC Web页面、移动端Web页面、小程序、公众号、甚至包括App，都属于前端开发的范畴。
当然也包括一系列复杂性的问题。

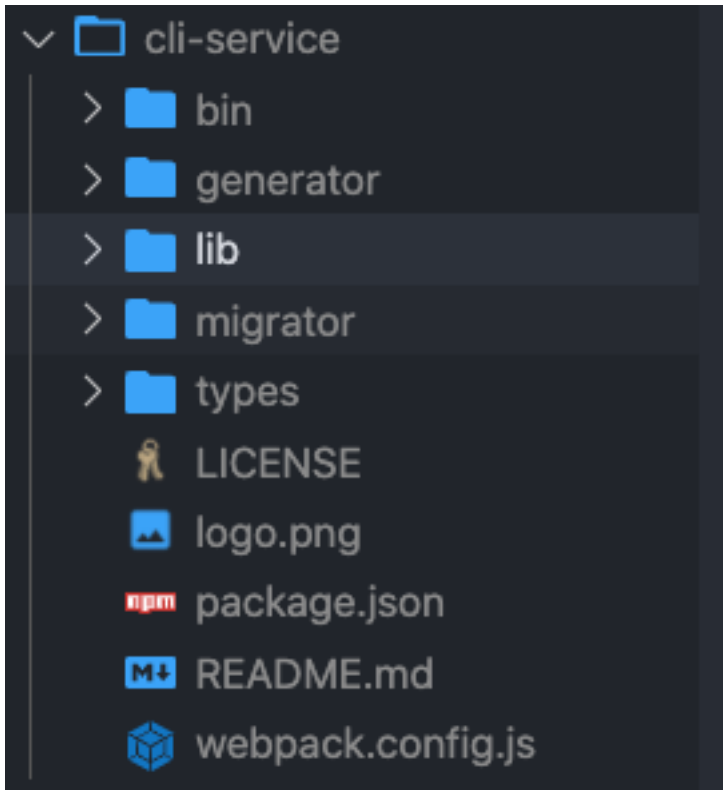
# 前端开发的复杂化

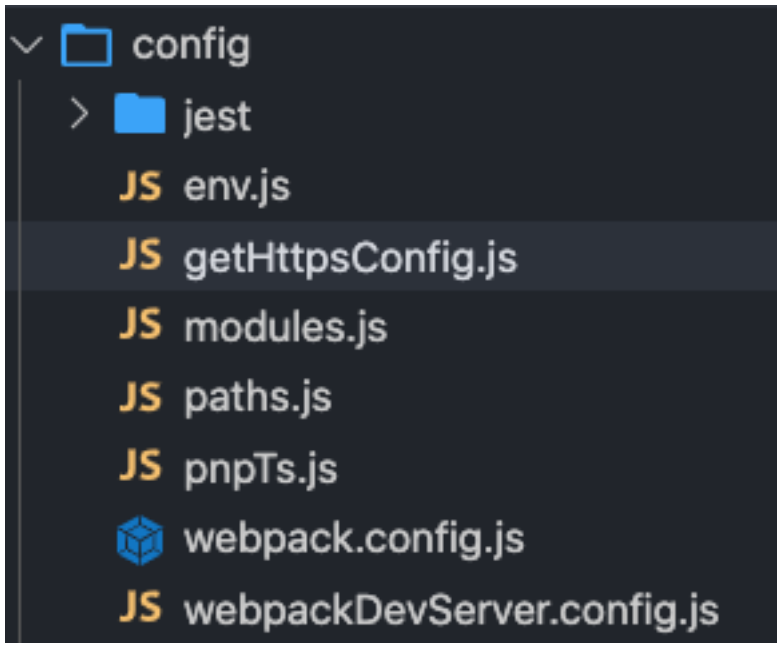
- 前端开发目前我们面临哪些复杂的问题呢？
  - 比如开发过程中我们需要通过模块化的方式来开发；
  - 比如也会使用一些高级的特性来加快我们的开发效率或者安全性，比如通过ES6+、TypeScript开发脚本逻辑，通过sass、less等方式来编写css样式代码；
  - 比如开发过程中，我们还希望试试的监听文件的变化来并且反映到浏览器上，提高开发的效率；
  - 比如开发完成后我们还需要将代码进行压缩、合并以及其他相关的优化；
  - 等等....

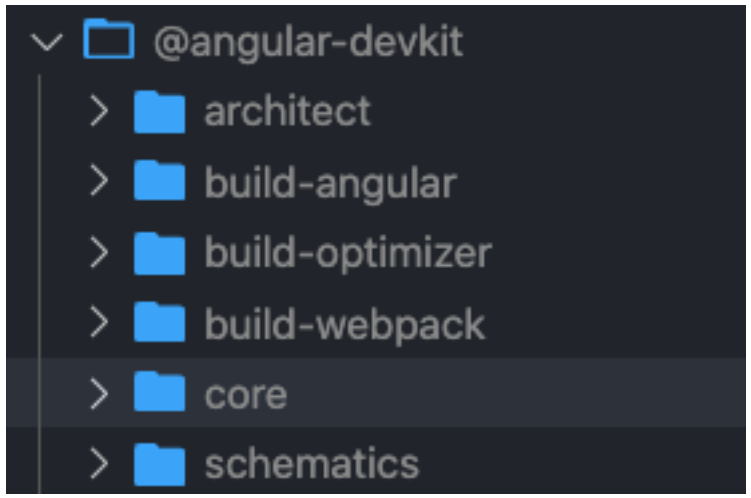




# 前端三个框架的脚手架

- 目前前端流行的三大框架：Vue、React、Angular
  - 但是事实上，这三大框架的创建过程我们都是借助于脚手架（CLI）的；
  - 事实上Vue-CLI、create-react-app、Angular-CLI都是基于webpack来帮助我们支持模块化、less、TypeScript、打包优化等的；



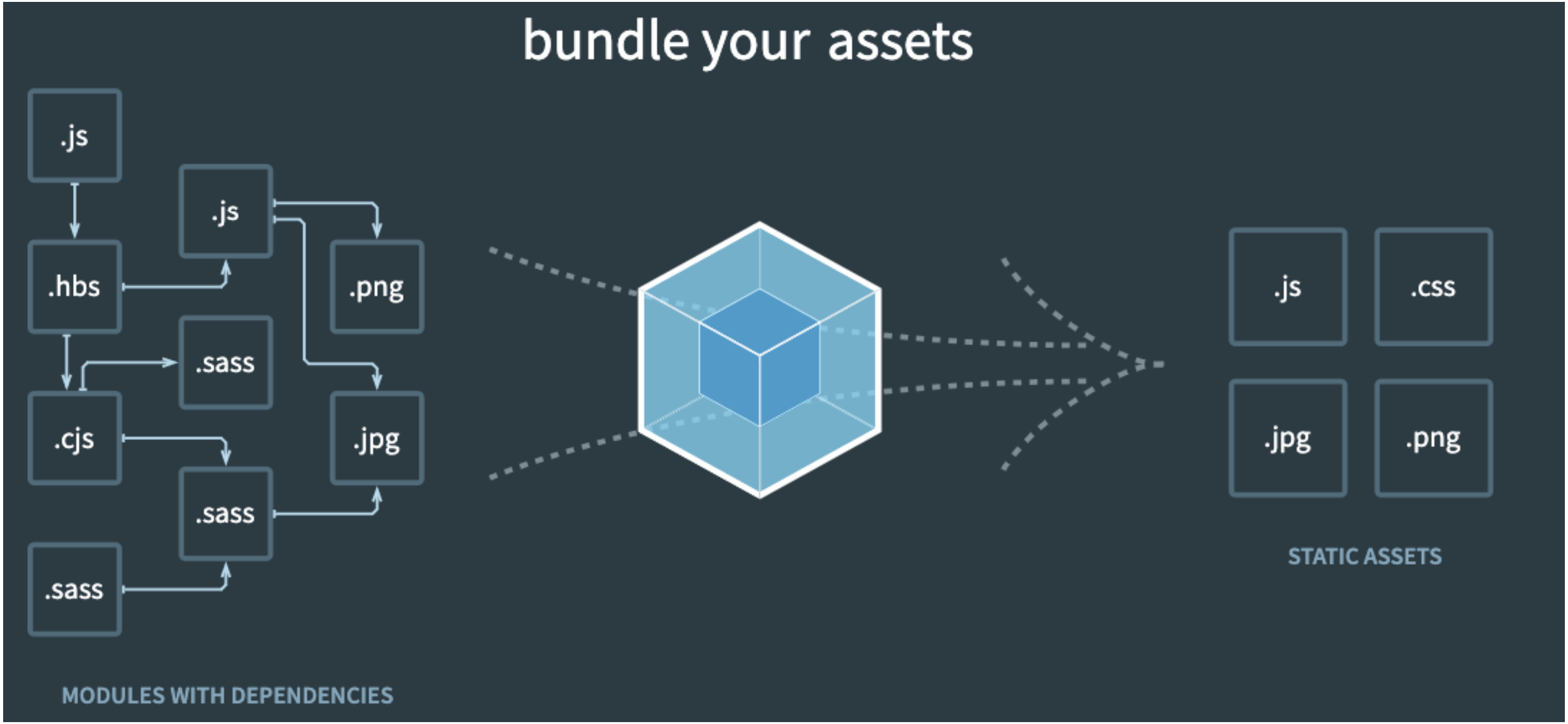

可见，我们日常工作根本是无法离开webpack的。

# Webpack是什么？

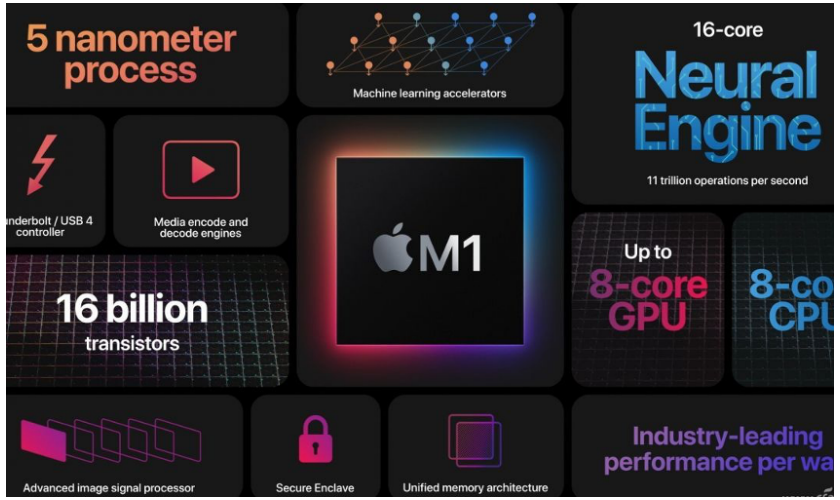
**webpack** is a *static module bundler* for modern JavaScript applications.

- webpack是一个静态的模块化打包工具，为现代的JavaScript应用程序；
- 我们来对上面的解释进行拆解：
  - 打包bundler：webpack可以将帮助我们进行打包，所以它是一个打包工具
  - 静态的static：这样表述的原因是我们最终可以将代码打包成最终的静态资源（部署到静态服务器）；
  - 模块化module：webpack默认支持各种模块化开发，ES Module、CommonJS、AMD等；
  - 现代的modern：我们前端说过，正是因为现代前端开发面临各种各样的问题，才催生了webpack的出现和发展；

# Webpack官网图片



# 工作中的webpack




日常工作来说，比如在开发vue、react、angular等项目的过程中我们需要一些特殊的配置：比如给某些目录结构起别名，让我们的项目支持sass、less等预处理器，希望在项目中手动的添加TypeScript的支持，都需要对webpack进行一些特殊的配置工作。

当然，除了日常工作之外，如果我们希望将在原有的脚手架上来定制一些自己的特殊配置提供性能：比如安装性能分析工具、使用gzip压缩代码、引用cdn的资源，公共代码抽取等等操作，甚至包括需要编写属于自己的loader和plugin。

对于想要在前端领域进阶成为高级前端开发工程师，甚至是架构师的前端开发者来说，webpack等构建工具是必须学习的，包括其中的一些高级特性和原理，都是要熟练掌握的。企业在招聘高级前端工程师或者架构师时，必然会对webpack和其他的构建工具有比较高的要求。

# webpack和vite

- 我们来提一个问题：webpack会被vite取代吗？
  - vite推出后确实引起了很多的反响，也有很多人看好vite的发展（包括我）；
- 但是目前vite取代webpack还有很长的路要走
  - 目前vue的项目支持使用vite，也支持使用webpack；
  - React、Angular的脚手架目前没有支持，暂时也没有转向vite的打算；
  - vite最终打包的过程，依然需要借助于rollup来完成；
- vite的核心思想并不是首创
  - 事实上，vite的很多思想和之前的snowpack是重合的，而且相对目前来说snowpack会更加成熟；
  - 当然，后续发展来看vite可能会超越snowpack；
- webpack的更新迭代
  - webpack在发展的过程中，也会不断改进自己，借鉴其他工具的一些优势和思想；
  - 在这么多年的发展中，无论是自身的优势还是生态都是非常强大的；

# 关于vite的思考

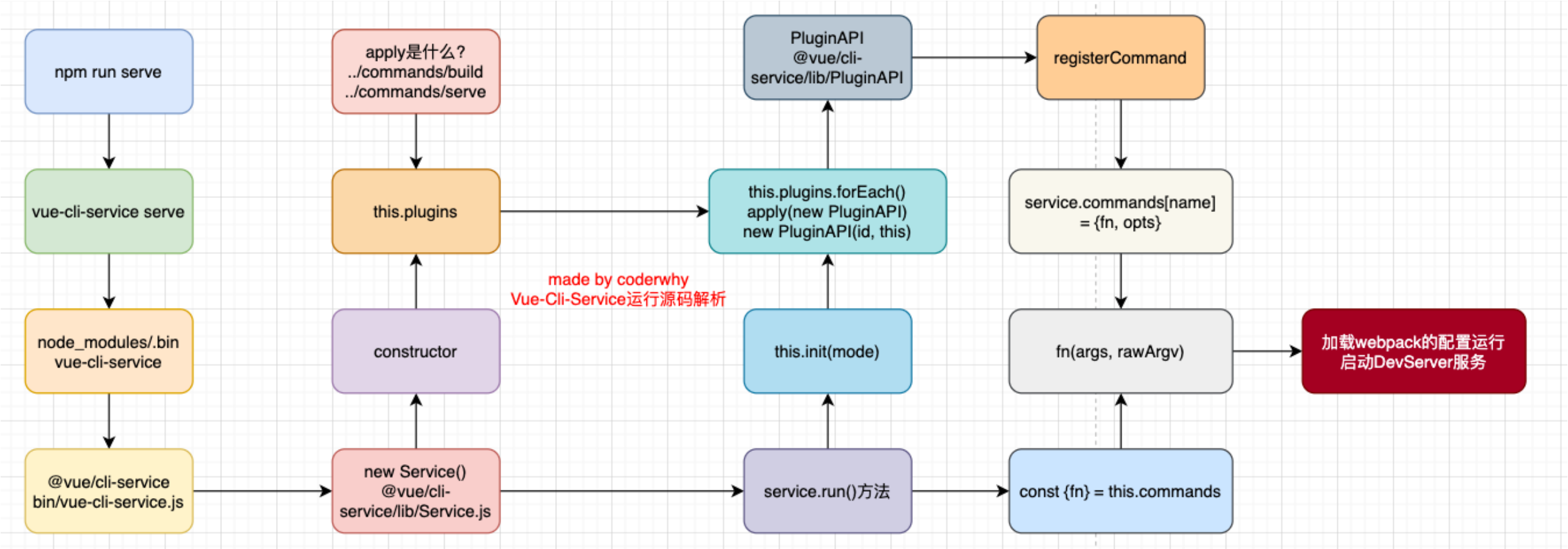
- **我的个人观点：**
- 学习任何的东西，重要的是学习核心思想：
  - 我们不能学会了JavaScript，有一天学习TypeScript、Java或者其他语言，所有的内容都是从零开始的；
  - 我们在掌握了react之后，再去学习vue、Angular框架，也应该是可以快速上手的；
- 任何工具的出现，都是更好的服务于我们开发：
  - 无论是vite的出现，还是以后新的工具的出现，**不要有任何排斥的思想**；
  - 我们要深刻的明白，工具都是为了更好的给我们提供服务；
  - 不可能出现了某个工具，让我们的开发效率变得更低，而这个工具却可以变得非常流行，这是不存在的；
- 在后续的学习中，我也会讲完webpack核心知识后带着大家一起来学习gulp、rollup、vite工具：
  - 还需要担心什么吗？

# 课程的内容

■ **课程核心内容：**

- webpack核心配置深入解析；
- webpack常用Loaders和Plugins深入学习；
- 自定义webpack中自己的Loader和Plugin；
- Babel各种用法以及polyfill、TypeScript的支持；
- ESLint的配置规则以及在VSCode、webpack中的使用；
- 各种性能优化方案：打包抽取分包、Tree Shaking、动态链接库、CDN、gzip压缩等等；
- webpack模块化原理解析、打包原理实现；
- 掌握其他流行构件工具：gulp、rollup、vite；

# vue-cli-service运行过程

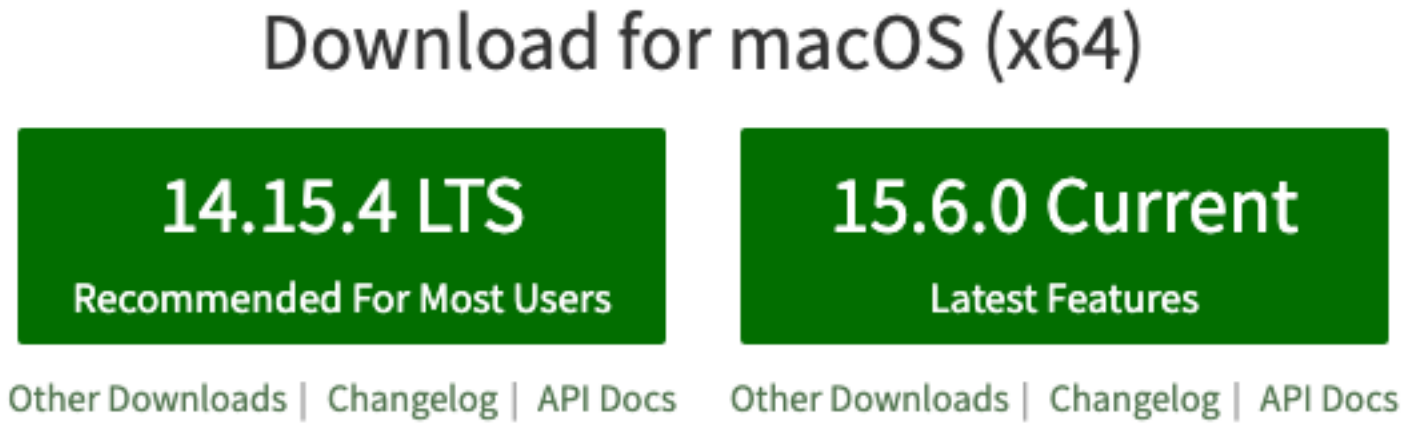


# Webpack的官方文档

- webpack的官方文档是https://webpack.js.org/
  - webpack的中文官方文档是https://webpack.docschina.org/
  - DOCUMENTATION：文档详情，也是我们最关注的
- 点击DOCUMENTATION来到文档页：
  - **API**：API，提供相关的接口，可以自定义编译的过程（比如自定义loader和Plugin可以参考该位置的API）
  - **BLOG**：博客，等同于上一个tab的BLOG，里面有一些博客文章；
  - **CONCEPTS**：概念，主要是介绍一些webpack的核心概念，比如入口、出口、Loaders、Plugins等等，但是这里并没有一些对它们解析的详细API；
  - **CONFIGURATION**：配置，webpack详细的配置选项，都可以在这里查询到，更多的时候是作为查询手册；
  - **GUIDES**：指南，更像是webpack提供给我们的教程，我们可以按照这个教程一步步去学习webpack的使用过程；
  - **LOADERS**：loaders，webpack的核心之一，常见的loader都可以在这里查询到用法，比如css-loader、babel-loader、less-loader等等；
  - **PLUGINS**：plugins，webpack的核心之一，常见的plugin都可以在这里查询到用法，比如BannerPlugin、CleanWebpackPlugin、MiniCssExtractPlugin等等；
  - **MIGRATE**：迁移，可以通过这里的教程将webpack4迁移到webpack5等；

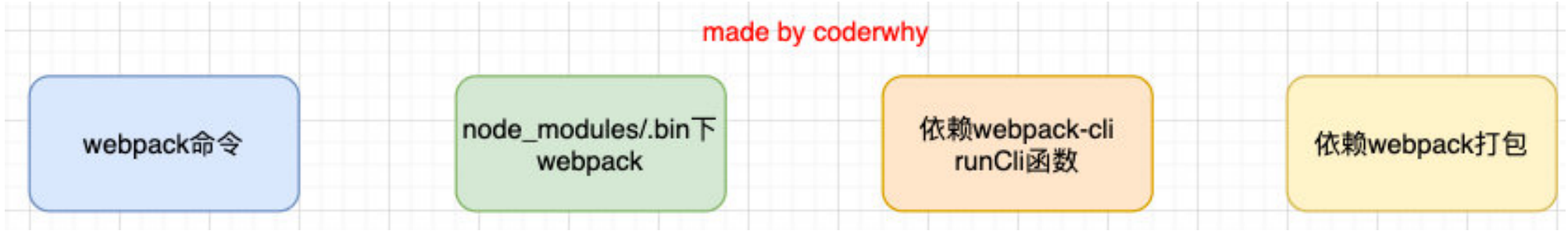
# Webpack的依赖

- Webpack的运行是依赖Node环境的，所以我们电脑上必须有Node环境
  - 所以我们需要先安装Node.js，并且同时会安装npm；
  - 我当前电脑上的node版本是v15.4.0，npm版本是7.0.15（你也可以使用nvm或者n来管理Node版本）；
  - Node官方网站：https://nodejs.org/



# Webpack的安装

- webpack的安装目前分为两个：webpack、webpack-cli
- 那么它们是什么关系呢？
  - 执行webpack命令，会执行node_modules下的.bin目录下的webpack；
  - webpack在执行时是依赖webpack-cli的，如果没有安装就会报错；
  - 而webpack-cli中代码执行时，才是真正利用webpack进行编译和打包的过程；
  - 所以在安装webpack时，我们需要同时安装webpack-cli（第三方的脚手架事实上是没有使用webpack-cli的，而是类似于自己的vue-service-cli的东西）




```
npm install webpack webpack-cli -g # 全局安装
npm install webpack webpack-cli -D # 局部安装
```

# 传统开发存在的问题

- 我们的代码存在什么问题呢？某些语法浏览器是不认识的（尤其在低版本浏览器上）
  - 1.使用了ES6的语法，比如const、箭头函数等语法；
  - 2.使用了ES6中的模块化语法；
  - 3.使用CommonJS的模块化语法；
  - 4.在通过script标签引入时，必须添加上 type="module" 属性；
- 显然，上面存在的问题，让我们在发布静态资源时，是不能直接发布的，因为运行在用户浏览器必然会存在各种各样的兼容性问题。
  - 我们需要通过某个工具对其进行打包，让其转换成浏览器可以直接识别的语法；

# webpack默认打包

- 我们可以通过webpack进行打包，之后运行打包之后的代码
  - 在目录下直接执行 webpack 命令

```
webpack
```

- 生成一个dist文件夹，里面存放一个main.js的文件，就是我们打包之后的文件：
  - 这个文件中的代码被压缩和丑化了；
  - 我们暂时不关心他是如何做到的，后续我讲webpack实现模块化原理时会再次讲到；
  - 另外我们发现代码中依然存在ES6的语法，比如箭头函数、const等，这是因为默认情况下webpack并不清楚我们打包后的文件是否需要转成ES5之前的语法，后续我们需要通过babel来进行转换和设置；
- 我们发现是可以正常进行打包的，但是有一个问题，webpack是如何确定我们的入口的呢？
  - 事实上，当我们运行webpack时，webpack会查找当前目录下的 src/index.js作为入口；
  - 所以，如果当前项目中没有存在src/index.js文件，那么会报错；

# Webpack配置文件

- 在通常情况下，webpack需要打包的项目是非常复杂的，并且我们需要一系列的配置来满足要求，默认配置必然是不可以的。
- 我们可以在根目录下创建一个webpack.config.js文件，来作为webpack的配置文件：

```
const path = require('path');

// 导出配置信息
module.exports = {
  entry: "./src/main.js",
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "./dist")
  }
}
```

继续执行webpack命令，依然可以正常打包

```
webpack
```

# 指定配置文件

- 但是如果我们的配置文件并不是webpack.config.js的名字，而是其他的名字呢？
  - 比如我们将webpack.config.js修改成了 wk.config.js；
  - 这个时候我们可以通过 --config 来指定对应的配置文件；

```
webpack --config wk.config.js
```

- 但是每次这样执行命令来对源码进行编译，会非常繁琐，所以我们可以在package.json中增加一个新的脚本：

```
{
  ▷ Debug
  "scripts": {
    "build": "webpack --config wk.config.js"
  },
  "devDependencies": {
    "webpack": "^5.14.0",
    "webpack-cli": "^4.3.1"
  }
}
```

之后我们执行 npm run build来打包即可。

# Webpack依赖图

- webpack到底是如何对我们的项目进行打包的呢？
  - 事实上webpack在处理应用程序时，它会根据命令或者配置文件找到入口文件；
  - 从入口开始，会生成一个 依赖关系图，这个依赖关系图会包含应用程序中所需的所有模块（比如.js文件、css文件、图片、字体等）；
  - 然后遍历图结构，打包一个个模块（根据文件的不同使用不同的loader来解析）；