构建性能： 持久化缓存

构建效率方法：

- 不重复干： 缓存
- 一块干： 并行处理
- 快干：提高效率、高效编译器、编译路径
- 少干：减少步骤、缩减范围

在开发过程中，如果轻微的修改就需要运行全部的打包过程显然是一种巨大的浪费。

使用缓存技术，存储中间产物就成了最有效的优化手段。

- 将首次构建过程持久化到文件系统中
- 在下次执行是跳过解析、链接、编译过程

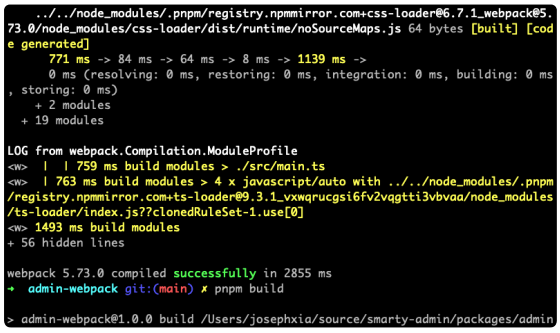
webpack 中启动缓存

webpack.config.js

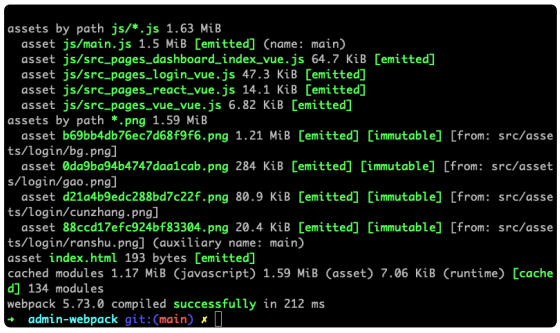
```JavaScript
  cache: {
    type: "filesystem",
    profile: true,
  },
```

对比

第 1 列



第 2 列



常用参数

- `cache.type`：缓存类型，支持 `'memory' | 'filesystem'`，需要设置为 `filesystem` 才能开启持久缓存；
- `cache.cacheDirectory`：缓存文件路径，默认为 `node_modules/.cache/webpack` ；
- `cache.buildDependencies`：额外的依赖文件，当这些文件内容发生变化时，缓存会完全失效而执行完整的编译构建，通常可设置为各种配置文件
- `cache.managedPaths`：受控目录，Webpack 构建时会跳过新旧代码哈希值与时间戳的对比，直接使用缓存副本，默认值为 `['./node_modules']`；
- `cache.profile`：是否输出缓存处理过程的详细日志，默认为 `false`；
- `cache.maxAge`：缓存失效时间，默认值为 `5184000000` 。

service/build.js

TypeScript

```
"use strict";

let base = require("./config/base");
const cache = require("./config/cache");
let config = base;
config.mode = "production";

module.exports = config;
```

service/dev.js

```JavaScript
"use strict";

let base = require("./config/base");
const cache = require("./config/cache");
let config = base;
config.mode = "development";

module.exports = config;
```

config/cache.js

```JavaScript
module.exports = (base) => {
  Object.assign(base, {
    cache: {
      type: "filesystem",
      profile: true,
    },
  });

  return base;
};
```

config/speedMeasure.js

```JavaScript
const SpeedMeasurePlugin = require("speed-measure-webpack-plugin");
const smp = new SpeedMeasurePlugin();
module.exports = (base) => {
  // 屏蔽VueLoader与speedMeasure不兼容的问题
  loader = base.plugins[0];
  base.plugins.shift();
```

```
  base = smp.wrap(base);
  base.plugins.unshift(loader);
  return base;
};
```

package.json

```JSON
"scripts": {
   "build": "webpack --config ./service/build.js",
   "dev": "webpack serve --progress --config ./service/dev.js"
 },
```