【拓展】webpack常用优化手段

写在前面

在第五、六章中，我们完成了原生 js 项目到 webpack5 的模块化框架升级和 ZBestPC 项目进阶升级的学习。在这两章内容中，我们接触到了 webpack 工程化应用的高级技巧。接下来，是我们的加餐环节：了解 webpack 的常用优化手段。在开始阅读之前，我希望大家能先思考一个问 我们进行优化打包的目的是什么?

webpack打包优化方向

- **打包速度**：优化打包速度，主要是提升了我们的开发效率，更快的打包构建过程，将让你保持一颗愉悦的心
- **打包体积**：优化打包体积，主要是提升产品的使用体验，降低服务器资源成本，更快的页面加载，将让产品显得更加“丝滑”，同时也可以让 更快

webpack打包速度优化

webpack 进行打包速度优化有七种常用手段

1. 优化 loader 搜索范围

对于 loader 来说，影响打包效率首当其冲必属 Babel 了。因为 Babel 会将代码转为字符串生成 AST，然后对 AST 继续进行转变最后再生成新 码，项目越大，转换代码越多，效率就越低。优化正则匹配、使用 include 和 exclude 指定需要处理的文件,忽略不需要处理的文件

```
rules: [{
   // 优化正则匹配
   test: /\.js$/,
   // 指定需要处理的目录
   include: path.resolve(__dirname, 'src')
   // 理论上只有include就够了，但是某些情况需要排除文件的时候可以用这个，排除不需要处理文件
   // exclude: []
 }]
```

2. 多进程/多线程

受限于 node 是单线程运行的，所以 webpack 在打包的过程中也是单线程的，特别是在执行 loader 的时候，长时间编译的任务很多，这样就会 等待的情况。我们可以使用一些方法将 loader 的同步执行转换为并行，这样就能充分利用系统资源来提高打包速度了

```
{
    test: /\.js?$/,
    exclude: /node_modules/,
    use: [
    {
      loader: "thread-loader",
      options: {
        workers: 3 // 进程 3 个
      }
    },
      {
        loader: "babel-loader",
        options: {
          presets: ["@babel/preset-env"],
          plugins: ["@babel/plugin-transform-runtime"]
        }
      }
    ]
 },
```