- 性能优化网络篇
  - webpack
    - 优化瓶颈
      - 构建速度
      - 打包大小
    - 构建速度优化：
      - 1.不要让loader做太多
        - include 或 exclude 来帮我们避免不必要的转译
        - noParse: /jquery|lodash/ 可以用于配置哪些模块文件的内容不需要进行解析。对于一些不需要解析依赖（即无依赖） 的第三方大型类库等，可以通过这个字段来配置，以提高整体的构建速度
          - 使用 noParse 进行忽略的模块文件中不能使用 import、require、define 等导入机制
        - 开启缓存将编译结果缓存到文件系统loader: 'babel-loader?cacheDirectory=true'
      - 配置 resolve.
        - mainFields（慎用 除非你知道你用的所有第三方模块的入口文件描述）
          - // 只采用 main 字段作为入口文件描述字段，以减少搜索步骤
mainFields: ['main'],
        - 1.resolve
          - resolve: {
// 使用绝对路径指明第三方模块存放的位置，以减少搜索步骤
modules: [path.resolve(__dirname, 'node_modules')]}
        - extensions
          - 尽可能较少默认后缀 频繁使用的放前面 等
        - alias
          - 使用 alias 把导入 react 的语句换成直接使用单独完整的 react.min.js 文件，
// 减少耗时的递归解析操作
alias: {
'react': path.resolve(__dirname, './node_modules/react/dist/react.min.js'),
      - 2.处理第三方库
        - Externals
          - 不够聪明，引发重复打包问题
          - CommonsChunkPlugin 每次构建时重新构建一次vendor
          - 动态链接库：dll(见generator-easy-vue)
      - 3.Happypack充分利用CPU的多核，将loader等由单进程改为多进程
        - // 手动创建进程池
const happyThreadPool = HappyPack.ThreadPool({ size: os.cpus().length })
      - webpack-parallel-uglify-plugin
        - 开启多个子进程 UglifyJS
      - 1.可视化分析工具webpack-bundle-analyzer
      - 拆分资源动态链接库 dll
        - const DllPlugin = require('webpack/lib/DllPlugin');
        - require('webpack/lib/DllReferencePlugin');
      - tree shake 删除用于代码
        - 基于 import/export 语法，Tree-Shaking 可以在编译的过程中获悉哪些模块并没有真正被使用，这些没用的代码，在最后打包的时候会被去除。(如果你采用 ES5 中的模块化，例如 module.export={...}则不会生效)
        - 第三方采用的commonJS是无效的
        - 修改.babelrc 让其保留 ES6 模块化语句 "modules": false
        - tree shake（webpack2原生支持）结合压缩插件
          - UglifyJsPlugin(weboack3)
          - webpack4生产环境自动开启压缩可以配置optimization.minimize 与 optimization.minimizer
      - css tree shake? purecss
      - cope hoisting作用域提升
        - Webpack3 new webpack.optimize.ModuleConcatenationPlugin()
        - Webpack4 mode生产环境自动启用
        - 前提和tree shake一样 要ES2015 的模块语法写的
        - 原理：将所有模块的代码按照引用顺序放在一个函数作用域里，然后适当的重命名一些变量以防止变量名冲突 分析出模块之间的依赖关系，尽可能的把打散的模块合并到一个函数中去，但前提是不能造成代码冗余。 因此只有那些被引用了一次的模块才能被合并。
      - image-webpack-loader 压缩图片
      - PWA:workbox-webpack-plugin
      - sideEffects https://github.com/webpack/webpack/issues/6571
        - package.json 文件中已经有 sideEffects: false 这个声明了，当某个模块的 package.json 文件中有了这个声明之后，webpack 会认为这个模块没有任何副作用，只是单纯用来对外暴露模块使用，那么在打包的时候就会做一些额外的处理import { default as forEach } from 'lodash-es/forEach'
import { default as includes } from 'lodash-es/includes'
    - 构建结果体积优化
      - Webpack编译优化之 后编译 https://github.com/DDFE/DDFE-blog/issues/23
      - 按需加载/Code Splitting && Long-term caching
        - output
          - // 指定 chunkFilename
chunkFilename: '[name].[chunkhash:5].chunk.js',
          - - 以 ./show.js 为入口新生成一个 Chunk；
- 当代码执行到 import 所在语句时才会去加载由 Chunk 对应生成的文件。
- import 返回一个 Promise，当文件加载成功时可以在 Promise 的 then 方法中获取到 show.js 导出的内容。
- /* webpackChunkName: "show" */ 为动态生成的文件取名字， 默认[id].js,当然为了正确输出文件名字需要配置出口
- babel-plugin-syntax-dynamic-import
        - 最佳实践
          - 1.为 Vendor 单独打包（Vendor 指第三方的库或者公共的基础组件，因为 Vendor 的变化比较少，单独打包利于缓存）
2.为 Manifest （Webpack 的 Runtime 代码）单独打包
3.为不同入口的公共业务代码打包（同理，也是为了缓存和加载速度）
4.为异步加载的代码打一个公共的包
            - webpack3
            - webpack4
          - 本质: 在正确的时机去触发相应的回调
      - HRM 模块热替换
        - new webpack.HotModuleReplacementPlugin(), //当启动时带上 `new webpack.NamedModulesPlugin(), // HMR shows correct file names in console on update.
      - prepack(优化代码运行时的效率 目前技术不成熟，慎用)
        - 在保持运行结果一致的情况下，改变源代码的运行逻辑，输出性能更高的 JavaScript 代码
        - 实质上就是一个部分求值器，编译代码时提前将计算结果放到编译后的代码中，而不是在代码运行时才去求值。通过在编译阶段预先执行了源码得到执行结果
        - https://prepack.io/getting-started.html
        - 原理
          - 通过 Babel 把 JavaScript 源码解析成抽象语法树（AST），以方便更细粒度地分析源码；
          - Prepack 实现了一个 JavaScript 解释器，用于执行源码。借助这个解释器 Prepack 才能掌握源码具体是如何执行的，并把执行过程中的结果返回到输出中。
        - prepack-webpack-plugin
only works with webpack 4.x
      - webpack-dashboard 优化输出界面
      - Gzip原理
        - compression-webpack-plugin
        - request headers： accept-encoding:gzip
        - 构建过程中去做一部分服务器的工作，为服务器分压
        - 原理 一个文本文件中找出一些重复出现的字符串、临时替换它们，从而使整个文件变小