代码分片

code splitting
Webpack特有的技术，可以按照特定形式把代码拆分，实现按需加载
代码分片可以有效降低首次加载资源的大小

通过入口划分代码

每个入口 entry 都将生成一个对应的资源文件
通常入口产生的资源不会经常更新，可以利用客户端缓存

CommonChunkPlugin

Webpack4之前的自带插件
可以将多个chunk中公共部分提取出来

- 开发过程减少了重复模块打包，提升开发速度
- 减小整体资源体积
- 合理分片后的代码可以更有效利用客户端缓存

Webpack.config.js配置需要更改
因为要使用内部的CommonChunkPlugin，所以需要导入webpack。然后在plugin中配置

- name 公共chunk名字
- filename 提取后资源的文件名

```
const webpack = require('webpack')
module.exports = {
    entry:{
        foo:"./foo.js",
        bar:"./bar.js"
    },
    output:{
        filename:'[name].js'
    },
    plugins:[
        new webpack.optimize.CommonsChunkPlugin({
            name:'commons',
            filename:'common.js'
        })
    ]
}
```

提取vendor

CommonChunkPlugin 多入口提取公共模块
单入口也能用，只需要为公共模块创建一个入口即可

设置提取范围

chunks配置项可以规定 从哪些入口提取公共模块

设置提取规则

CommonChunkPlugin默认规则
一个模块被两个入口chunk所只用，就会被提取出来

但是有些模块 虽然被重复引用，但是可能经常修改，如果跟其他第三方库提取到一起，反而不利于缓存
因为chunk的hash经常因为模块改动而改动

可以通过minChunk 来设置提取的规则

- 数字 被引用的入口>=n时，才会被提取
- Infinity
  - 只让Webpack提取特定模块，让提取可控
  - 生成一个没有任何模块，只包含webpack初始化环境的文件，manifest
- 函数
  更细颗粒度地控制公共模块，只有true才会被提取

hash与长效缓存

运行时 runtime 中有模块id，因为模块id变动，会导致chunk地hash变化，从而影响缓存

解决方案：把运行时代码单独提取出来
plugins中最后加上manifest

```
const webpack = require('webpack');
module.exports = {
    entry: {
        app: './app.js',
        vendor: ['react'],
    },
    output: {
        filename: '[name].js',
    },
    plugins: [
        new webpack.optimize.CommonsChunkPlugin({
            name: 'vendor',
        }),
        new webpack.optimize.CommonsChunkPlugin({
            name: 'manifest',
        })
    ],
};
```

manifest必须写在最后，不然Webpack无法正常提取模块
并且页面需要最先引用manifes，来初始化webpack环境

CommonChunkPlugin地不足

- 一个CommonChunkPlugin只能提取一个vendor
  多个vendor需要配置多个插件，增加很多重复代码
- manifest会让浏览器多加载一个资源，不利于页面加载速度
- 会破坏原有Chunk中模块的依赖关系，难以进行更多优化
- 异步chunk，无法按照预进行工作

optimization.SplitChunks

简称SplitChunks， Webpack4 自带
与CommonChunkPlugin的不同

- mode
- chunks

声明式

只需要设置 提取条件（提取模式，提取体积等）
默认提取条件

- 提取后的chunk可被共享或来自node_modules目录
- 提取后js chunk > 30kb (压缩和gzip之前)，CSS chunk > 50kb
  小资源其实没啥提取的必要
- 按序加载 并行请求最大值 <= 5
  提取规则只在并行请求不多的时候生效
  请求需要建立连接
- 首次加载，并行请求资源最大值 <= 3

默认异步提取

配置

```
splitChunks: {
    chunks: "async",
    minSize: {
      javascript: 30000,
      style: 50000,
    },
    maxSize: 0,
    minChunks: 1,
    maxAsyncRequests: 5,
    maxInitialRequests: 3,
    automaticNameDelimiter: '~',
    name: true,
    cacheGroups: {
        vendors: {
            test: /[\\/]node_modules[\\/]/,
            priority: -10,
        },
        default: {
            minChunks: 2,
            priority: -20,
            reuseExistingChunk: true,
        },
    },
},
```

- 匹配模式
  配置SplitChunk工作模式
  async（默认）, initial, all
- 匹配条件
  minSize，minChunks，maxAsyncRequests，maxInitialRequest 等
- 命名
  name默认为true，能够根据cacheGroups和作用范围自动生成新的chunk
- cacheGroups
  分离chunks是的规则
  - vendor 提取所有node_modules中符合条件的模块
  - default 多次引用的模块

资源异步加载

模块数量过多，资源体积过大，可以把一些暂时用不到的模块延迟加载。
页面初次渲染用户下载的资源尽可能小，后续模块等到恰当的时机再去触发加载。
按需加载

import()

Webpack 2种异步加载方式

- require.ensure Webpack 1
- import() Webpack 2 开始并推荐使用

与ES6 Module中import语法不同
import函数加载的模块及其依赖会被异步加载，并且返回Promise

ES6 Module 要求import出现在代码顶层作用域，但是webpack import可以出现任何时候，减小首屏加载资源的压力

异步chunk的设置

异步资源名称默认全是id，没有意义，需要通过配置
output.chunkFilename来指定异步chunk文件名

小结

Webpack代码分片的几种方式

- 合理规划入口
- CommonChunkPlugin
- SplitChunk
- 资源异步加载

减小资源体积，更好的利用缓存