DLL_Tree Shaking

认识DLL库

- DLL是什么呢？
  - DLL全程是动态链接库（Dynamic Link Library），是为软件在Windows中实现共享函数库的一种实现方式；
  - 那么webpack中也有内置DLL的功能，它指的是我们可以将可以共享，并且不经常改变的代码，抽取成一个共享的库；
  - 这个库在之后编译的过程中，会被引入到其他项目的代码中；
- DLL库的使用分为两步:
  - 第一步：打包一个DLL库；
  - 第二步：项目中引入DLL库
- 注意：在升级到webpack4之后，React和Vue脚手架都移除了DLL库（下面的vue作者的回复）

Build

- `dll` option will be removed. Webpack 4 should provide good enough perf and the cost of maintaining DLL mode inside Vue CLI is no longer justified.

打包一个DLL库

- 如何打包一个DLLPlugin？（创建一个新的项目）
  - webpack帮助我们内置了一个DllPlugin可以帮助我们打包一个DLL的库文件；

```
module.exports = {
  mode: 'production',
  entry: {
    react: ["react", "react-dom"]
  },
  output: {
    path: path.resolve(__dirname, "./dll"),
    filename: "dll_[name].js",
    library: 'dll_[name]'
  },
  plugins: [
    new webpack.DllPlugin({
      name: 'dll_[name]',
      path: path.resolve(__dirname, "./dll/[name].manifest.json")
    })
  ]
}
```

使用打包的DLL库

- 如果我们在我们的代码中使用了react、react-dom，我们有配置splitChunks的情况下，他们会进行分包，打包到一个独立的chunk中。
  - 但是现在我们有了dll_react，不再需要单独去打包它们，可以直接去引用dll_react即可：
  - 第一步：通过DllReferencePlugin插件告知要使用的DLL库；
  - 第二步：通过AddAssetHtmlPlugin插件，将我们打包的DLL库引入到Html模块中；

```
new webpack.DllReferencePlugin({
  context: resolveApp("./"),
  manifest: resolveApp("./dll/react.manifest.json")
}),
new AddAssetHtmlPlugin({
  outputPath: "./auto",
  filepath: resolveApp("./dll/dll_react.js")
})
```

Terser介绍和安装

- 什么是Terser呢？
  - Terser是一个JavaScript的解释（Parser）、Mangler（绞肉机）/Compressor（压缩机）的工具集；
  - 早期我们会使用 uglify-js来压缩、丑化我们的JavaScript代码，但是目前已经不再维护，并且不支持ES6+的语法；
  - Terser是从 uglify-es fork 过来的，并且保留它原来的大部分API以及适配 uglify-es和uglify-js@3等；
- 也就是说，Terser可以帮助我们压缩、丑化我们的代码，让我们的bundle变得更小。
- 因为Terser是一个独立的工具，所以它可以单独安装：

```
# 全局安装
npm install terser -g

# 局部安装
npm install terser
```

命令行使用Terser

- 我们可以在命令行中使用Terser：

```
terser [input files] [options]

# 举例说明
terser js/file1.js -o foo.min.js -c -m
```

- 我们这里来讲解几个Compress option和Mangle option：
  - 因为他们的配置非常多，我们不可能一个个解析，更多的查看文档即可；
  - https://github.com/terser/terser#compress-options
  - https://github.com/terser/terser#mangle-options

Compress和Mangle的options

- Compress option：
  - arrows：class或者object中的函数，转换成箭头函数；
  - arguments：将函数中使用 arguments[index]转成对应的形参名称；
  - dead_code：移除不可达的代码（tree shaking）；
  - 其他属性可以查看文档；
- Mangle option
  - toplevel：默认值是false，顶层作用域中的变量名称，进行丑化（转换）；
  - keep_classnames：默认值是false，是否保持依赖的类名称；
  - keep_fnames：默认值是false，是否保持原来的函数名称；
  - 其他属性可以查看文档；

```
npx terser ./src/abc.js -o abc.min.js -c
arrows,arguments=true,dead_code -m
toplevel=true,keep_classnames=true,keep_fnames=true
```

Terser在webpack中配置

- 真实开发中，我们不需要手动的通过terser来处理我们的代码，我们可以直接通过webpack来处理：
  - 在webpack中有一个minimizer属性，在production模式下，默认就是使用TerserPlugin来处理我们的代码的；
  - 如果我们对默认的配置不满意，也可以自己来创建TerserPlugin的实例，并且覆盖相关的配置；
- 首先，我们需要打开minimize，让其对我们的代码进行压缩（默认production模式下已经打开了）
- 其次，我们可以在minimizer创建一个TerserPlugin：
  - extractComments：默认值为true，表示会将注释抽取到一个单独的文件中；
    - ✓ 在开发中，我们不希望保留这个注释时，可以设置为false；
  - parallel：使用多进程并发运行提高构建的速度，默认值是true，并发运行的默认数量： os.cpus().length - 1；
    - ✓ 我们也可以设置自己的个数，但是使用默认值即可；
  - terserOptions：设置我们的terser相关的配置
    - ✓ compress：设置压缩相关的选项；
    - ✓ mangle：设置丑化相关的选项，可以直接设置为true；
    - ✓ toplevel：底层变量是否进行转换；
    - ✓ keep_classnames：保留类的名称；
    - ✓ keep_fnames：保留函数的名称；

CSS的压缩

- 另一个代码的压缩是CSS：
  - CSS压缩通常是去除无用的空格等，因为很难去修改选择器、属性的名称、值等；
  - CSS的压缩我们可以使用另外一个插件：css-minimizer-webpack-plugin；
  - css-minimizer-webpack-plugin是使用cssnano工具来优化、压缩CSS（也可以单独使用）；
- 第一步，安装 css-minimizer-webpack-plugin：

```
npm install css-minimizer-webpack-plugin -D
```

- 第二步，在optimization.minimizer中配置

```
minimizer: [
  new TerserPlugin({…
  }),
  new CssMinimizerPlugin({
    parallel: true
  })
],
```

什么是Tree Shaking

- 什么是Tree Shaking呢？
  - Tree Shaking是一个术语，在计算机中表示消除死代码（dead_code）；
  - 最早的想法起源于LISP，用于消除未调用的代码（纯函数无副作用，可以放心的消除，这也是为什么要求我们在进行函数式编程时，尽量使用纯函数的原因之一）；
  - 后来Tree Shaking也被应用于其他的语言，比如JavaScript、Dart；
- JavaScript的Tree Shaking：
  - 对JavaScript进行Tree Shaking是源自打包工具rollup（后面我们也会讲的构建工具）；
  - 这是因为Tree Shaking依赖于ES Module的静态语法分析（不执行任何的代码，可以明确知道模块的依赖关系）；
  - webpack2正式内置支持了ES2015模块，和检测未使用模块的能力；
  - 在webpack4正式扩展了这个能力，并且通过 package.json的 sideEffects属性作为标记，告知webpack在编译时，哪里文件可以安全的删除掉；
  - webpack5中，也提供了对部分CommonJS的tree shaking的支持；
    - ✓ https://github.com/webpack/changelog-v5#commonjs-tree-shaking

webpack实现Tree Shaking

- 事实上webpack实现Tree Shaking采用了两种不同的方案：
  - usedExports：通过标记某些函数是否被使用，之后通过Terser来进行优化的；
  - sideEffects：跳过整个模块/文件，直接查看该文件是否有副作用；

Clarifying tree shaking and `sideEffects`

The `sideEffects` and `usedExports` (more known as tree shaking) optimizations are two different things.

usedExports

- 将mode设置为development模式：
  - 为了可以看到 usedExports带来的效果，我们需要设置为 development 模式
  - 因为在 production 模式下，webpack默认的一些优化会带来很大额影响。
- 设置usedExports为true和false对比打包后的代码：
  - 在usedExports设置为true时，会有一段注释：unused harmony export mul；
  - 这段注释的意义是什么呢？告知Terser在优化时，可以删除掉这段代码；
- 这个时候，我们讲 minimize设置true：
  - usedExports设置为false时，mul函数没有被移除掉；
  - usedExports设置为true时，mul函数有被移除掉；
- 所以，usedExports实现Tree Shaking是结合Terser来完成的。

sideEffects

- sideEffects用于告知webpack compiler哪些模块时有副作用的：
  - 副作用的意思是这里面的代码有执行一些特殊的任务，不能仅仅通过export来判断这段代码的意义；
  - 副作用的问题，在讲React的纯函数时是有讲过的；
- 在package.json中设置sideEffects的值：
  - 如果我们将sideEffects设置为false，就是告知webpack可以安全的删除未用到的exports；
  - 如果有一些我们希望保留，可以设置为数组；
- 比如我们有一个format.js、style.css文件：
  - 该文件在导入时没有使用任何的变量来接受；
  - 那么打包后的文件，不会保留format.js、style.css相关的任何代码；

```
"sideEffects": [
  "./src/util/format.js",
  "*.css"
],
```

Webpack中tree shaking的设置

- 所以，如何在项目中对JavaScript的代码进行TreeShaking呢（生成环境）？
  - 在optimization中配置usedExports为true，来帮助Terser进行优化；
  - 在package.json中配置sideEffects，直接对模块进行优化；

CSS实现Tree Shaking

- 上面我们学习的都是关于JavaScript的Tree Shaking，那么CSS是否也可以进行Tree Shaking操作呢？
  - CSS的Tree Shaking需要借助于一些其他的插件；
  - 在早期的时候，我们会使用PurifyCss插件来完成CSS的tree shaking，但是目前该库已经不再维护了（最新更新也是在4年前了）；
  - 目前我们可以使用另外一个库来完成CSS的Tree Shaking：PurgeCSS，也是一个帮助我们删除未使用的CSS的工具；
- 安装PurgeCss的webpack插件：

```
npm install purgecss-webpack-plugin -D
```

配置PurgeCss

- 配置这个插件（生成环境）：
  - paths：表示要检测哪些目录下的内容需要被分析，这里我们可以使用glob；
  - 默认情况下，Purgecss会将我们的html标签的样式移除掉，如果我们希望保留，可以添加一个safelist的属性；

```
new PurgecssPlugin({
  paths: glob.sync(`${resolveApp("./src")}/**/*`, {nodir: true}),
  safelist: function() {
    return {
      standard: ["html"]
    }
  }
})
```

- purgecss也可以对less文件进行处理（所以它是对打包后的css进行tree shaking操作）；

Scope Hoisting

- 什么是Scope Hoisting呢？
  - Scope Hoisting从webpack3开始增加的一个新功能；
  - 功能是对作用域进行提升，并且让webpack打包后的代码更小、运行更快；
- 默认情况下webpack打包会有很多的函数作用域，包括一些（比如最外层的）IIFE：
  - 无论是从最开始的代码运行，还是加载一个模块，都需要执行一系列的函数；
  - Scope Hoisting可以将函数合并到一个模块中来运行；
- 使用Scope Hoisting非常的简单，webpack已经内置了对应的模块：
  - 在production模式下，默认这个模块就会启用；
  - 在development模式下，我们需要自己来打开该模块；

```
new webpack.optimize.ModuleConcatenationPlugin(),
```