Tree Shaking以及其他优化

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"
],
```

```
{
  test: /\.css/i,
  // style-lodader -> development
  use: [
    isProduction ? MiniCssExtractPlugin.loader: "style-loader",
    "css-loader"],
  sideEffects: true
},
```

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）；

什么是HTTP压缩？

- HTTP压缩是一种内置在 服务器 和 客户端 之间的，以改进传输速度和带宽利用率的方式；
- HTTP压缩的流程什么呢？
  - 第一步：HTTP数据在服务器发送前就已经被压缩了；（可以在webpack中完成）
  - 第二步：兼容的浏览器在向服务器发送请求时，会告知服务器自己支持哪些压缩格式；

```
GET /encrypted-area HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
```

  - 第三步：服务器在浏览器支持的压缩格式下，直接返回对应的压缩后的文件，并且在响应头中告知浏览器；

```
HTTP/1.1 200 OK
Date: Tue, 27 Feb 2018 06:03:16 GMT
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
```

目前的压缩格式

- 目前的压缩格式非常的多：
  - compress – UNIX的“compress”程序的方法（历史性原因，不推荐大多数应用使用，应该使用gzip或deflate）；
  - deflate – 基于deflate算法（定义于RFC 1951）的压缩，使用zlib数据格式封装；
  - gzip – GNU zip格式（定义于RFC 1952），是目前使用比较广泛的压缩算法；
  - br – 一种新的开源压缩算法，专为HTTP内容的编码而设计；

Webpack对文件压缩

- webpack中相当于是实现了HTTP压缩的第一步操作，我们可以使用CompressionPlugin。
- 第一步，安装CompressionPlugin：

```
npm install compression-webpack-plugin -D
```

- 第二步，使用CompressionPlugin即可：

```
new CompressionPlugin({
  test: /\.(css|js)$/, // 匹配哪些文件需要压缩
  // threshold: 500, // 设置文件从多大开始压缩
  minRatio: 0.7, // 至少的压缩比例
  algorithm: "gzip", // 采用的压缩算法
  // include:
  // exclude:
}),
```

HTML文件中代码的压缩

- 我们之前使用了**HtmlWebpackPlugin**插件来生成HTML的模板，事实上它还有一些其他的配置：
- inject：设置打包的资源插入的位置
  - true、 false 、body、head
- cache：设置为true，只有当文件改变时，才会生成新的文件（默认值也是true）
- minify：默认会使用一个插件html-minifier-terser

```
minify: isProduction ? {
  removeComments: true, // 移除注释
  collapseWhitespace: false, // 折叠空格
  removeRedundantAttributes: false, // 移除多余的属性 type=text
  useShortDoctype: true, // 比如我们的模板是html4，那么会转成html5的文档
  removeEmptyAttributes: true, // 移除空的属性 id=""
  removeStyleLinkTypeAttributes: true, // 比如link中的 type="text/css"
  keepClosingSlash: true, // 是否保持单元素的尾部/
  minifyCSS: false, // 是否压缩css
  minifyJS: {
    mangle: {
      toplevel: true
    }
  },
}: false
```

InlineChunkHtmlPlugin

- 另外有一个插件，可以辅助将一些chunk出来的模块，内联到html中：
  - 比如runtime的代码，代码量不大，但是是必须加载的；
  - 那么我们可以直接内联到html中；
- 这个插件是在react-dev-utils中实现的，所以我们可以安装一下它：

```
npm install react-dev-utils -D
```

- 在production的plugins中进行配置：

```
const InlineChunkHtmlPlugin = require('react-dev-utils/InlineChunkHtmlPlugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // ...
  plugins: [
    // ...
    new InlineChunkHtmlPlugin(HtmlWebpackPlugin, [/runtime.+\.js/])
  ]
}
```

封装Library

- webpack可以帮助我们打包自己的库文件，比如我们需要打包一个coderwhy_utils的一个库。

```
// lib/format.js
export function dateFormat() {
  return "2020-10-20"
}

// lib/math.js
export function sum(num1, num2) {
  return num1 + num2;
}
export function mul(num1, num2) {
  return num1 * num2;
}

// index.js
import * as math from "./lib/math";
import * as format from "./lib/format";
export {
  math,
  format
}
```

打包Library

- 配置webpack.config.js文件

```
const path = require('path');

module.exports = {
  mode: "development",
  entry: "./index.js",
  output: {
    path: path.resolve(__dirname, "./build"),
    filename: "why_util.js",
    // AMD/CommonJS/浏览器
    libraryTarget: "umd",
    library: "whyUtil",
    globalObject: "this"
  }
}
```

```
(function webpackUniversalModuleDefinition(root, factory) {
  // 支持commonjs2,也就node中的CommonJS
  if(typeof exports === 'object' && typeof module === 'object')
    module.exports = factory();
    // AND
  else if(typeof define === 'function' && define.amd)
    define([], factory);
    // Pure CommonJS
  else if(typeof exports === 'object')
    exports["whyUtil"] = factory();
    // 浏览器的支持
  else
    root["whyUtil"] = factory();
})(this, function() {
```