你知道 webpack 中的 chunk 么

Chunk 是什么？

chunk 表示一个文件，默认情况下 webpack 的输入是一个入口文件，输出也是一个文件，这个文件就是一个 chunk，chunkId 就是产出时给每个文件一个唯一标识 id，chunkhash 就是文件内容的 md5 值，name 就是在 entry 中指定的 key 值。

先来看一张图：

```
Version: webpack 4.29.6
Time: 86ms
Built at: 2019-03-28 10:55:12
                    Asset      Size  Chunks             Chunk Names
bundle80f3e7ed9a68d742fa7c  3.77 KiB  bundle  [emitted]  bundle
   fun80f3e7ed9a68d742fa7c  3.76 KiB     fun  [emitted]  fun
Entrypoint bundle = bundle80f3e7ed9a68d742fa7c
Entrypoint fun = fun80f3e7ed9a68d742fa7c
[./src/func.js] 0 bytes {fun} [built]
[./src/index.js] 0 bytes {bundle} [built]
```

这张图是简单打包的结果。一般运行结果和这个差不多可能文件更大一些，更多一些，但是也是这几部分构成的。

里面有两个字段，一个 chunks，另外一个 chunk names；要解释这两个字段是需要配置文件来进行解释的。

webpack.config.js

```
const path = require('path');
module.exports = {
    mode: 'development',
    entry: {
        bundle: './src/index.js',
        fun: './src/func.js'
    },
    output: {
        filename: '[name][hash]',
        path: path.resolve(__dirname, 'dist')
    }
}
```

在上述代码中，我们在 entry 中设置的 对象中的 key 值，也就是 bundle，以及 fun 就是我们给与入口文件的命名，一个 key 对应一个 chunk 文件。在输出的时候，在 output

中，[name]表示用原有的 chunk 名称[hash] 利用 md5 命名的目的用于浏览器的缓存机制。

这个 chunk 的名称的计算过程：

1. 先打开根目录下的 webpack.config.js
2. 找 entry（入口）属性的值
3. 进入到 bundle.js 里，如果依赖其他文件，再找到其他文件
4. 把 bundle.js 与依赖的其他文件合并成一个 js 文件
5. 在 webpack.config.js 里找到 output（出口）属性
6. 解析 output 里的 path 与 filename 属性的值，如果有 hash 在加上 hash 的值
7. 把第 4 步合并成的 js 文件放到 dist 文件夹里，并起个名字叫 bundle.js 或者（bundle.xxxx(此处为 hash 值）.js）

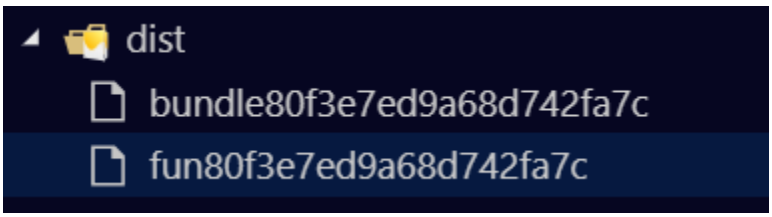
note:

1、当 entry 为数组的时候，webpack 会把数组里所有文件打包成一个 js 文件
2、当 entry 为对象的时候，webpack 会把对象里的文件分别打包成多个文件

这里在多说一些关于 webpack 中的 hash 的内容。

浏览器为了优化体验，会有缓存机制。如果浏览器判断当前资源没有更新，就不会去服务端下载，而是直接使用本地资源。在 webpack 的构建中，我们通常使用给文件添加后缀值来改名以及提取公共代码到不会改变的 lib 包中来解决新资源缓存问题。这个东西就是 hash 也就是我们所说的 md5

如上面配置，在我们的打包结果中就会出现



在上面，我们看到，我们配置的 hash 值，然后打包出的是一个长串字符串，这边的的长度我们可以指定，比如配置为

webpack.config.js

```
// ...
output: {
        filename: '[name][hash:5]',
        path: path.resolve(__dirname, 'dist')
```

```
    }
// ...
```

这样打包出来的就是 5 位的 hash 字符串，我们看到这边 bundle 模块打包出的和 fun 模块打包出的 hash 值是一样的，这个是什么原因呢？

对于 webpack 来说，它是一个打包编译的过程，也就是一个 compilation 的过程，这个标识符，标识的就是这个打包的过程。这样就很好解释了模块标识符的概念就是在相同编译打包过程中的模块所共有的标识符，也就是说同一过程产出的产物的 hash 值都是一样的，也就解释了上面的过程。

这里会出现一个问题

这个问题就是：如果都使用 hash 的话，所有文件的 hash 都是一样的，而且每次修改任何一个文件，所有文件名的 hash 值都将改变。所以一旦修改了任何一个文件，整个项目的文件缓存都将失效

我们本着网络优化的思想去做浏览器缓存，但是当我们服务器修改一个文件重新打包的时候。所有的文件名称都会发生变化，本应该缓存的内容也并没有缓存。这并不是我们想要的，那该怎么办？

既然 hash 的用法有这种缺陷，那是否有更好的办法，使只有被修改了的文件的文件名 hash 值修改呢？答案就是使用 chunkhash。

当我们把配置文件中的 hash 换成 chunkhash 的时候,like this

webpack.config.js

```
// ...
output: {
        filename: '[name][chunkhash:5]',
        path: path.resolve(__dirname, 'dist')
    }
// ...
```

会发现我们修改那个文件，那个文件就会重新打包

之前的打包结果：

```
                Asset             Size        Chunks
   Chunk Names
bundle4e992     3.78 KiB          bundle [emitted]                      bundle
         fun49599     3.78 KiB                    fun
```

```
[emitted]                                fun
Entrypoint bundle = bundle4e992
Entrypoint fun = fun49599
```

修改 func 的内容后的打包结果

```
                  Asset               Size           Chunks
           Chunk Names
bundle4e992      3.78
KiB            bundle     [emitted]                        bundle
funec2df                  3.78
KiB            fun              [emitted]                                fun
Entrypoint bundle = bundle4e992
Entrypoint fun = funec2df
```

发现只有我们修改的 chunk fun 发生变化，进行重新打包。这个也是符合预期的

对于 CSS 来说，JS 引入的 CSS 发生变化，也会重新 hash，这个时候无论 chunkhash，还是 hash 都不好用，那该怎么办呢？

使用 mini-css-tract-plugin 中定义的 contenthash 来进行打包，来解决以上问题。

webpack.config.js

```
// ...
{
    test: /\.css$/,
    use: [{
           loader: MiniCssTractPlugin.loader
           },
               'css-loader'


]
},
// ...
```

webpack.config.js

```
// ...
```

```
 plugins: [
        new MiniCssTractPlugin({
            filename: "[name].[contenthash:5].css"
        })
    ]

// ...
```