

12 使用 Webpack 管理项目中的静态资源

更新时间：2019-06-24 09:26:25



“

低头要有勇气，抬头要有底气。

——韩寒

”

前端项目离不开各种静态资源，静态资源指前端中常用的图片、富媒体（Video、Audio 等）、字体文件等。Webpack 中静态资源也是可以作为模块直接使用的，本小节将介绍下 Webpack 中对静态资源的管理。

图片引入方式

图片是前端项目必不可少的静态资源，在日常开发中，我们可能会在下面三种情况使用图片：

1. HTML 中通过 `` 标签等方式引入；
2. CSS 中通过 `src` 等方式引入；
3. JavaScript 中使用图片的 URL 或者内容（比如 Canvas 等）。

最笨最直接的方式就是直接写死线上的地址，例如在页面中，我们引入 `` 如下：

```

```

上面地址的 `http://s.bxstatic.com` 是一个 CDN 静态域名，后面是我们完整的路径，这样我们上线的时候地址就可以直接看了，我们线下开发的时候可以提前将静态资源打包好上传到线上。这样操作想想就很费劲，而且 CDN 每次静态资源更新都要需要刷新缓存，如果我们使用 MD5 命名图片的时候就麻烦了。

我们在 Webpack 中，则可以使用 loader 的方式完成图片的引入。例如在 CSS 文件中，直接相对路径使用背景图片：

```
.bg-img {
  background: url(../foo/bar.png) no-repeat;
}
```

在 HTML 中也可以直接使用相对路径:

```

```

还记得我们之前学过的 `resolve.alias` 方式创建一个目录的 `alias` 引用, 这种方式不仅仅可以在 `JavaScript` 中使用, 在 `HTML` 和 `CSS` 中也可以使用的:

```
// webpack.config.js
module.exports = {
  resolve: {
    alias: {
      '@assets': path.resolve('./src/assets')
    }
  }
};
```

```

```

使用 **loader** 来加载图片资源

怎么使用图片我们了解了, 但是怎么让 `Webpack` 识别图片, 并且能够打包输出呢? 这时候就需要借助 **loader** 了, 这里有两个 **loader** 可以使用: `file-loader`和`url-loader`。

`file-loader` 和 `url-loader` 是经常在一些 `Webpack` 配置中看到的两个 **loader**, 并且两个 **loader** 在一定应用场景上是可以相互替代的, 但是对于两者的区别, 很少有人能够说得清楚, 下面介绍下两者的区别。

- **file-loader**: 能够根据配置项复制使用到的资源 (不局限于图片) 到构建之后的文件夹, 并且能够更改对应的链接;
- **url-loader**: 包含 `file-loader` 的全部功能, 并且能够根据配置将符合配置的文件转换成 `Base64` 方式引入, 将小体积的图片 `Base64` 引入项目可以减少 `http` 请求, 也是一个前端常用的优化方式。

下面以 `url-loader` 为例说明下 `Webpack` 中使用方法。首先是安装对应的 **loader**: `npm install -D url-loader`。

下面我们创建一个项目, 目录结构如下:

```
├─ package.json      # npm package.json
├─ src
│  └─ assets          # 静态资源
│     └─ img          # 图片资源
│        └─ large.png  # 大图图片 超过1M
│        └─ small-02.png # 小图图片
│        └─ small-03.png
│        └─ small-04.png
│        └─ small-05.png
│        └─ small.png  # 小图图片
├─ index.css          # css 文件
├─ index.html         # html
├─ index.js           # js
└─ webpack.config.js  # webpack 配置
```

首先我们在 `index.css` 中引入 `small.png`,

```
.bg-img {
  background: url(./assets/img/small.png) no-repeat;
}
```

然后在 `index.js` 中引入了 `index.css` 和 `large.png`：

```
import img from './assets/img/large.png';
import style from './index.css';
console.log(img, style);
```

最后我们在 `index.html` 中通过 `` 引入 `large.png`：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Image</title>
  </head>
  <body>
    
  </body>
</html>
```

这时候我们修改 `webpack.config.js`：

```
// webpack.config.js
const HTMLPlugin = require('html-webpack-plugin');

module.exports = {
  mode: 'development',
  entry: './src/index.js',
  devtool: false,
  module: {
    rules: [
      {
        test: /\.html$/,
        use: ['html-loader']
      },
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader']
      },
      {
        test: /\.(png|svg|jpg|gif)/,
        use: {
          loader: 'url-loader'
        }
      }
    ]
  },
  plugins: [
    new HTMLPlugin({
      template: './src/index.html'
    })
  ]
};
```

执行 `webpack` 之后的 log:

```
Hash: a9d76b931b5cf9ebf6b3
Version: webpack 4.29.6
Time: 1254ms
Built at: 2019-04-13 12:24:28
```

| Asset | Size | Chunks | Chunk Names |
|------------|----------|--------|----------------|
| index.html | 1.52 MiB | | [emitted] |
| main.js | 1.54 MiB | main | [emitted] main |

```
Entrypoint main = main.js
```

这时候发现，打包出来的文件都比较大，通过查看内容发现，我们的图片被 Base64 处理了，然后直接引入了：

```

/***/ " ./src/assets/img/large.png":
/*! *****! * \
!*** ./src/assets/img/large.png ***!
\******/
/*! no static exports found */
/***/ (function(module, exports) {

module.exports = "data:image/png;base64,
ivBORw0KGGooAAAANSUUhEUGad6AAAAQQCAYAAAD2sqSNAABAGdBtUEAA
eQKZriKA7AugauZm4V/+77f/9/tbGd/fvr99sf8AZswyAmJk+av07ENPX
+5/ped7z5yp91F6PP9PzfMY/9z97Nvf61py8gBi5f67Ss82s7+3r97evP
+WQBvX77EcxdltgySBoir7WWZzJ/ped7Tl/mz/ix/LT5qr8ef6awD8gB8
+rPBVXRp3tsXK923sm9QP99HVSrbz5RtLPu/zVVxs/4sf5VeLQXW09fjz
+V340cXhr2s2jPn6v0UXs9/kzP8yZ+lKtCFxcIAA4ZeJw+njzM6bjKr7d
+DIW+i2hnnP+sIPZhuj/Nn/PO/py/w9f67SR+31+DM9z0fix3zjfLf3y
+jpCZ0WHOOmm4ZrbL2vl/KtEMG2Nq/xb0nWNzu3ZV17gPX+u0kft9fgz
+BBP-B/d+dc+w69HwY3AP-3P4w+EEZ+*TL/Abh-Vvd-6O/Vf52w+u+

```

这是因为 `url-loader` 本身优先是将资源 `Base64` 引入的。虽然图片 `Base64` 可以减少 `http` 请求，但是对于 `1M+` 这么大的图片都 `Base64` 处理，范围增加了 `CSS`、`JavaScript` 等文件的大小，而且将这么大的 `Base64` 反解成可以使用的图片渲染出来，时间消耗也是很大的。

所以这时候需要使用 `url-loader` 的 `limit` 选项来控制不超过一定限制的图片才使用 `Base64`:

```
{
  test: /\.png|svg|jpg|gif$/,
  use: {
    loader: 'url-loader',
    options: {
      limit: 3*1024 // 3k
    }
  }
}
```

这时候再执行 `webpack`，发现多打出一个 `ad19429dc9b3ac2745c760bb1f460892.png` 的图片，这张图片就是 `large.png` 的图片，因为超过了 `limit=3*1024` 显示所以没有被处理成 `Base64`。

```

Hash: 3415569d8d8ed734d02b
Version: webpack 4.29.6
Time: 523ms
Built at: 2019-04-13 12:34:24

```

| Asset | Size | Chunks | Chunk Names |
|--------------------------------------|-----------|----------------|-------------|
| ad19429dc9b3ac2745c760bb1f460892.png | 1.14 MiB | [emitted] | |
| index.html | 350 bytes | [emitted] | |
| main.js | 27.8 KiB | main [emitted] | main |

```

Entrypoint main = main.js

```

继续查看 `index.html` 和 `main.js`（`index.js` 打包出来的文件），发现我们使用 `large.png` 的地址都被 Webpack 自动替换成了新的路径 `ad19429dc9b3ac2745c760bb1f460892.png`。

配置 CDN 域名

一般静态资源上线的时候都会放到 CDN，假设我们的 CDN 域名和路径为：`http://bd.bxstatic.com/img/`，这时候只需要修改 `output.publicPath` 即可：

```

module.exports = {
  //...
  output: {
    publicPath: 'http://bd.bxstatic.com/img/'
  }
  //...
};

```

修改后执行 `webpack` 打包后的结果如下：

```

<body>
  
  <script src="http://bd.bxstatic.com/img/main.js"></script>
</body>

```

说明 Webpack 为我们自动替换了路径，并且加上了 CDN 域名。

HTML 和 CSS 中使用 alias

前面提到过，除了使用相对路径的方式引入静态资源，还可以使用别名（`alias`）的方式，`url-loader` 也会给我们处理这种情况的引用。

修改 `index.html` 和 `index.css`：

```



```

```

.bg-img {
  background: url(@assets/img/small.png) no-repeat;
}

```

然后修改 `webpack.config.js` 增加 `resolve.alias`：

```

module.exports = {
  //...
  resolve: {
    alias: {
      '@assets': path.resolve(__dirname, './src/assets')
    }
  }
  //...
};

```

这时候执行 `webpack` 发现报错了：

```
ERROR in ./src/index.css (/webpack-tutorial/node_modules/css-loader/dist/cjs.js!./src/index.css)
Module not found: Error: Can't resolve './@assets/img/small.png' in '/webpack-tutorial/packages/chapter-02/static/src'
  @ ./src/index.css (/webpack-tutorial/node_modules/css-loader/dist/cjs.js!./src/index.css) 4:41-75
  @ ./src/index.css
  @ ./src/index.js

ERROR in   Error: Child compilation failed:
  Module not found: Error: Can't resolve './@assets/img/large.png' in '/webpack-tutorial/packages/chapter-02/static/src':
  Error: Can't resolve './@assets/img/large.png' in '/webpack-tutorial/packages/chapter-02/static/src'
```

这是因为在 `HTML` 和 `CSS` 使用 `alias` 必须要前面添加 `~`，即：

```

```

```
.bg-img {
  background: url(~@assets/img/small.png) no-repeat;
}
```

修改完后，直接执行 `webpack` 既可以看到正确的结果了。

Tips: `HTML` 中使用 `` 引入图片等静态资源的时候，需要添加 `html-loader` 配置，不然也不会处理静态资源的路径问题。

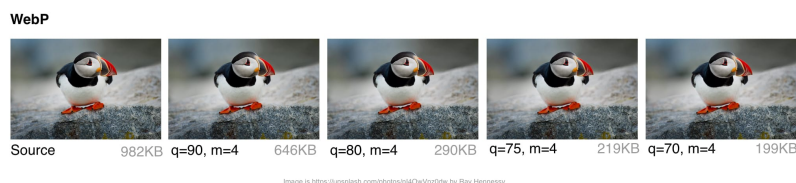
`svg-url-loader` 的工作原理类似于 `url-loader`，除了它利用 `URL encoding` 而不是 `Base64` 对文件编码。对于 `SVG` 图片这是有效的，因为 `SVG` 文件恰好是纯文本，这种编码规模效应更加明显，使用方法如下：

```
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.svg$/,
        loader: 'svg-url-loader',
        options: {
          // 小于 10kB(10240字节) 的内联文件
          limit: 10 * 1024,
          // 移除 url 中的引号
          // (在大多数情况下它们都不是必要的)
          noquotes: true
        }
      }
    ]
  }
};
```

Tips: `svg-url-loader` 拥有改善 `IE` 浏览器支持的选项，但是在其他浏览器中更糟糕。如果你需要兼容 `IE` 浏览器，设置 `iesafe: true` 选项。

图片优化

图片体积是个经常诟病的问题，一个页面中，完全一样内容的图片，在肉眼可见的范围内并不一定有差异但是体积却相差甚大，例如下面的图片：



所以图片优化也是我们在前端项目中经常做的事情，在 Webpack 中可以借助来对使用到的图片进行优化。它支持 JPG、PNG、GIF 和 SVG 格式的图片，因此我们在碰到所有这些类型的图片都会使用它。

```
# 安装
npm install image-webpack-loader --save-dev
```

这个 loader 不能将图片嵌入应用，所以它必须和 url-loader 以及 svg-url-loader 一起使用。为了避免同时将它复制粘贴到两个规则中（一个针对 JPG/PNG/GIF 图片，另一个针对 SVG），我们使用 `enforce: 'pre'` 作为单独的规则涵盖在这个 loader：

```
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.?(jpe?g|png|gif|svg)$/i,
        loader: 'image-webpack-loader',
        // 这会应用该 loader，在其它之前
        enforce: 'pre'
      }
    ]
  }
};
```

通过 `enforce: 'pre'` 我们提高了 image-webpack-loader 的优先级，保证在 url-loader 和 svg-url-loader 之前就完成了图片的优化。

另外默认的配置就已经适用于日常开发图片的压缩优化需求了，但是如果你想更进一步去配置它，参考[插件选项](#)。要选择指定选项，请查看国外牛人写的一个[图像优化指南](#)。

CSS Sprite 雪碧图

CSS 使用小图标图片的时候，我们经常做的优化项目是将小图标的图片合并成雪碧图（CSS Sprite），雪碧图的好处是将页面用到的小图片合并到一张大图中，然后使用 `background-position` 重新定位，这样节省了 HTTP 的请求数。

在 Webpack 中我们可以借助 PostCSS 来给图片做雪碧图，经过简单的配置之后，生成雪碧图就是全自动的过程了。下面来看看怎么操作。

首先安装 postcss-sprites

```
npm install postcss-sprites -D
# 如果没有安装 postcss-loader 那么也安装它
npm install postcss-loader -D
```

Tips: postcss-sprites 安装需要安装phantomjs可能需要正确上网。

然后修改 PostCSS 的 `postcss.config.js`，增加插件的调用：

```
// postcss.config.js
const postcssSprites = require('postcss-sprites');

module.exports = {
  plugins: [
    postcssSprites({
      // 在这里制定了从哪里加载的图片被主动使用css sprite
      // 可以约定好一个目录名称规范，防止全部图片都被处理
      spritePath: './src/assets/img/'
    })
  ]
};
```

然后修改 `webpack.config.js` 在 `css-loader` 之前配置上 `postcss-loader`（注意 loader 加载顺序，从后往前）：

```
//webpack.config.js
// rules

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

好了，下面我们的 CSS 中使用了 `spritePath: './src/assets/img/'` 路径的图片就会被处理了，例如下面的文件：

```
.bg-img02 {
  background: url(./assets/img/small-02.png) no-repeat;
}

.bg-img03 {
  background: url(./assets/img/small-03.png) no-repeat;
}

.bg-img04 {
  background: url(./assets/img/small-04.png) no-repeat;
}

.bg-img05 {
  background: url(./assets/img/small-05.png) no-repeat;
}
```

经过打包之后，输出 log 如下，可见生成了一个新的图片文件 `99b0de3534d3e852ea4ce83b15cbad60.png`：


```
Hash: b0d0eda7ebcc6850c457
Version: webpack 4.29.6
Time: 1743ms
Built at: 2019-04-13 13:29:39

```

| Asset | Size | Chunks | Chunk Names |
|--------------------------------------|-----------|----------------|-------------|
| 99b0de3534d3e852ea4ce83b15cbad60.png | 25 KiB | [emitted] | |
| ad19429dc9b3ac2745c760bb1f460892.png | 1.14 MiB | [emitted] | |
| index.html | 350 bytes | [emitted] | |
| main.js | 28.6 KiB | main [emitted] | main |

```
Entrypoint main = main.js
```

打开 `99b0de3534d3e852ea4ce83b15cbad60.png` 文件，我们看到图片被合并到了一起：



在打开打包之后的 `CSS` 文件，发现内容被主动替换成了 `CSS Sprite` 写法，并且设置了正确的 `background-position` 和 `background-size` 了：

```
.bg-img02 {
  background-image: url(99b0de3534d3e852ea4ce83b15cbad60.png);
  background-position: 0px 0px;
  background-size: 320px 320px;
}

.bg-img03 {
  background-image: url(99b0de3534d3e852ea4ce83b15cbad60.png);
  background-position: -160px 0px;
  background-size: 320px 320px;
}

.bg-img04 {
  background-image: url(99b0de3534d3e852ea4ce83b15cbad60.png);
  background-position: 0px -160px;
  background-size: 320px 320px;
}

.bg-img05 {
  background-image: url(99b0de3534d3e852ea4ce83b15cbad60.png);
  background-position: -160px -160px;
  background-size: 320px 320px;
}
```

其他资源处理

字体、富媒体

对于字体、富媒体等静态资源，可以直接使用 `url-loader` 或者 `file-loader` 进行配置即可，不需要额外的操作，具体配置内容如下：

```
{
  // 文件解析
  test: /\.?(eot|woff|ttf|woff2|appcache|mp4|pdf)(\?|$)/,
  loader: 'file-loader',
  query: {
    // 这么多文件，ext不同，所以需要使用[ext]
    name: 'assets/[name].[hash:7].[ext]'
  }
},
```

Tips: 如果不需要 Base64，那么可以直接使用 `file-loader`，需要的话就是用 `url-loader`，还需要注意，如果将正则（`test`）放在一起，那么需要使用 `[ext]` 配置输出的文件名。

数据

如果我们项目需要加载的类似 JSON、CSV、TSV 和 XML 等数据，那么我们需要单独给它们配置相应的 loader。对 JSON 的支持实际上是内置的，类似于 Node.js，这意味着 `import Data from './data.json'` 导入数据默认情况将起作用。要导入 CSV，TSV 和 XML，可以使用 `csv-loader` 和 `xml-loader`。

首先是安装它们的 loader: `npm i -D xml-loader csv-loader`，然后增加文件 loader 配置如下：

```
{
  test: /\.?(csv|tsv)$/,
  use: [
    'csv-loader'
  ]
},
{
  test: /\.xml$/,
  use: [
    'xml-loader'
  ]
}
```

现在，您可以导入这四种类型的数据中的任何一种（JSON，CSV，TSV，XML），并且导入它的 `Data` 变量将包含已解析的 JSON 以便于使用。

小结

本小节主要介绍 Webpack 中的图片、字体、富媒体、数据等多种静态资源的管理方式。页面经常用到的图片是页面的重点，Webpack 提供了很多插件和 loader 对图片进行压缩、合并（CSS Sprite）。Webpack 还会使用 `url-loader` 等插件，将较小的资源通过 Base64 的方式引入。

当项目足够大了之后，配置太多的静态资源处理流程也会影响 Webpack 的打包速度，想突破压缩和合并这类前端常见优化，我们可以通过让视觉人员提供最优图片格式等方式来人工解决。当然如果项目组一直没有优化的意识，担心一不小心上到线上一个很大的图片，那么使用 Webpack 来兜底也是个很不错的方案。

本小节 Webpack 相关面试题：

1. Webpack 中怎么给静态资源添加上 CDN 域名？
2. url-loader 和 file-loader 有什么区别？

