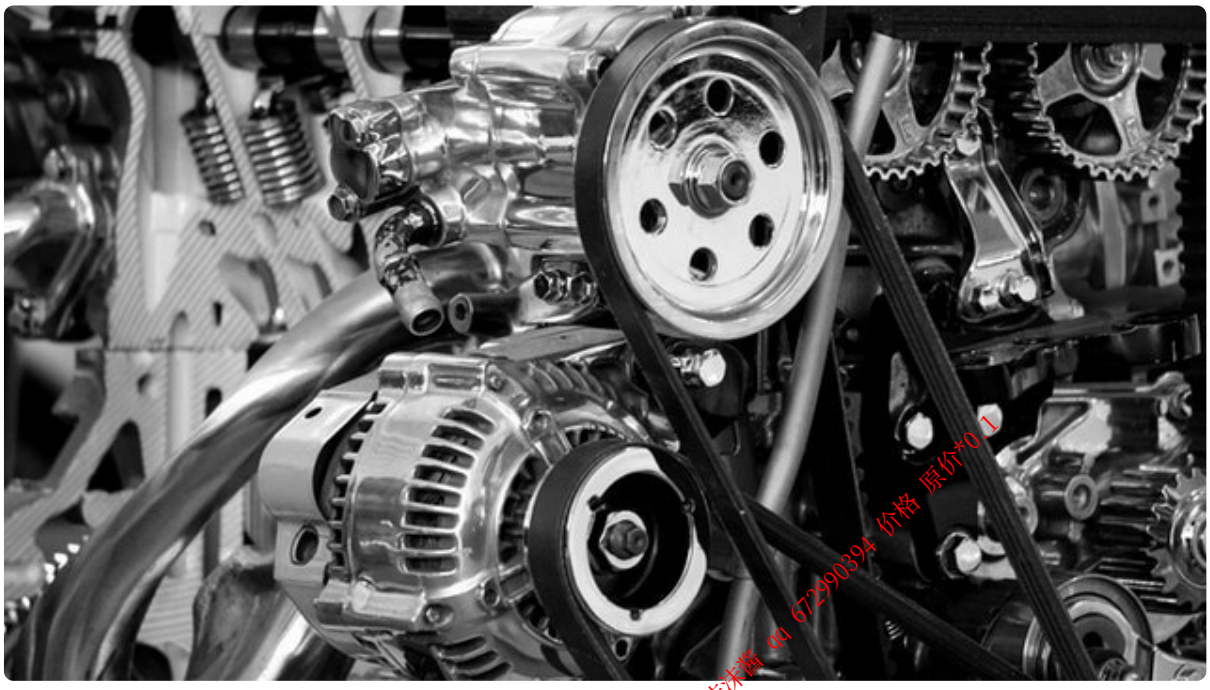
10 Webpack 中样式相关的配置

更新时间：2019-06-24 09:25:58



> 人的影响短暂而微弱，书的影响则广泛而深远。
>
> ——普希金

Webpack 中一切皆模块，CSS 也可以在 JavaScript 中被直接引用，但是 CSS 的语法 JavaScript 是不能解析的，所以下面代码会报错：

```
import css from './css/index.css';
console.log(css);
```

```
Hash: 4abd5de9fbc3a688eb38
Version: webpack 4.29.6
Time: 76ms
Built at: 2019-03-23 16:16:20
 Asset      Size  Chunks             Chunk Names
css.js  4.35 KiB    main  [emitted]  main
Entrypoint main = css.js
[./app.js] 147 bytes {main} [built]
[./css/index.css] 168 bytes {main} [built] [failed] [1 error]

ERROR in ./css/index.css 1:5
Module parse failed: Unexpected token (1:5)
You may need an appropriate loader to handle this file type.
> body {
|     background: magenta;
| }
 @ ./app.js 4:12-38
```

这时候就需要添加 Webpack 的 `loader` 来处理 CSS 了。

css-loader

首先添加 css-loader：

```
npm install --save-dev css-loader
# or
npm i -D css-loader
```

然后给`webpack.config.js`添加`rule`：

```
{
    module: {
        rules: [
            {
                test: /\.css$/,
                use: ['css-loader']
            }
        ];
    }
}
```

这时候修改`app.js`添加下面代码：

```
import css from './css/index.css';
console.log(css, css.toString());
```

效果如下：

这时候 CSS 会被转成字符串， JS 就可以直接使用。

除了上面直接在`webpack.config.js`中添加`rule`，还可以在 JavaScript 中直接使用下面的方式引入：

```
import css from 'css-loader!./css/index.css';
console.log(css);
```

上面代码中`import css from 'css-loader!./css/index.css'`是 webpack loader 的内联写法。

style-loader

有了 css-loader 可以识别 CSS 语法了，下面就需要 style-loader 出场了。简单来说，style-loader 是将 css-loader 打包好的 CSS 代码以`<style>`标签的形式插入到 HTML 文件中，所以`style-loader`是和`css-loader`成对出现的，并且`style-loader`是在`css-loader`之后。首先安装`style-loader`：

```
npm install --save-dev style-loader
# or
npm i -D css-loader
```

mini-css-extract-plugin

CSS 作为`<style>`标签放到 HTML 内还是不够的，我们还需要将 CSS 以`<link>`的方式通过 URL 的方式引入进来，这时候就需要使用mini-css-extract-plugin这个插件了，首先安装它：

```
npm install --save-dev mini-css-extract-plugin
```

`mini-css-extract-plugin`这个使用的时候需要分别配置 loader 和 plugin，loader 需要放在`css-loader`之后代替`style-loader`：

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
module.exports = {
    plugins: [
        // 添加 plugin
        new MiniCssExtractPlugin({
            filename: '[name].css',
            chunkFilename: '[id].css'
        })
    ],
    module: {
        rules: [
            {
                test: /\.css$/,
                // 添加 loader
                use: [MiniCssExtractPlugin.loader, 'css-loader']
            }
        ]
    }
};
```

CSS Modules

CSS Modules 指的是所有的 CSS 类名及其动画名都只是局部作用域的 CSS 文件。CSS Modules 既不是官方标准，也不是浏览器的特性，而是在构建过程中对 CSS 类名选择器限定作用域的一种方式，如我们的广告样式、某个 UI 通用弹层 SDK 这类样式，都需要避免自己的命名跟宿主环境的样式冲突或者避免被 AdBlock 这类广告拦截器拦截掉。CSS Modules 主要解决的问题有：

1. 解决 CSS 类都是全局的，容易造成全局污染（样式冲突）；
2. JS 和 CSS 共享类名；
3. 可以方便的编写出更加健壮和扩展方便的 CSS。

这类 CSS 模块化的解决方案很早之前前端社区就有一些讨论和方案，比如最早的通过 CSS 命名约定的BEM、OOCSS等，再到 React 中使用的用 JavaScript 来写 CSS 规则的 CSS in JS 方案，再到通过编译工具来帮助 JavaScript 可以使用 CSS 的 CSS Modules 方案。

下面来看下 CSS Modules 究竟是什么，我们来看下代码表现，首先创建一个`app.css`文件，内容如下：

```
/* app.css */
.element {
    background-color: blue;
    color: white;
    font-size: 24px;
}
```

我们知道了，在 JS 中可以直接`import`一个 CSS 文件：

```
// app.js
import styles from './app.css';
```

那么 CSS Modules 中，JS 可以直接使用 CSS 的类名作为对象值，例如下面代码：

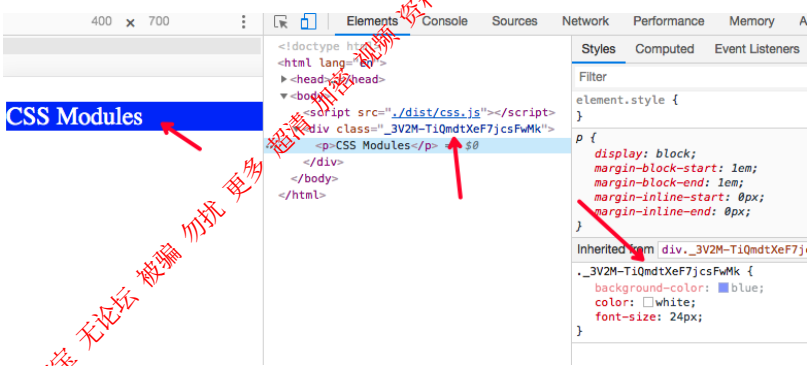
```
// app.js
import styles from './app.css';

let element = `
  <div class="${styles.element}">
    <p>CSS Modules</p>
  </div>
`;
document.write(element);
```

在 css-loader 增加`modules`的选项，说明打开 CSS Modules 支持。

```
module.exports = {
    //...
    module: {
        rules: [
            {
                test: /\.css$/,
                use: [
                    'style-loader',
                    {
                        loader: 'css-loader',
                        options: {
                            modules: true
                        }
                    }
                ]
            }
        ]
    }
};
```

执行 WebPack 打包，最终效果如下：



跟 CSS Modules 相关的配置还有很多，具体可以在`css-loader`对应的文档找到。

CSS 预处理器

由于 CSS 标准自诞生以来，一直致力于在表现力层面的发展，相对基本语法和核心机制并没有实质性的变化，所以产生了好多 CSS 的预处理器。预处理器补足了 CSS 的一些语法上的缺陷，支持变量、运算、函数、作用域、继承、嵌套写法等，使用 CSS 预处理器可以大大的提升开发效率和体验，同时能够更好的做好模块化开发。

> Tips：CSS 核心语法直到近些年才有大的发展，比如自定义属性（custom properties，又称为变量 variables）、嵌套写法，但是已经远远的落后于 CSS 预处理器的发展。

常见的 CSS 预处理器有：Less，Sass 及其语法变种 Scss和Stylus。

使用预处理器 loader

下面以 Less 预处理器为例，介绍 CSS 预处理器的用法。首先安装对应的 loader：`less-loader`：

```
npm i -D less-loader
# or
npm install less-loader --save-dev
```

然后修改`webpack.config.js`：

```
// webpack.config.js
module.exports = {
    //   ...
    module: {
        rules: [
            {
                test: /\.less$/,
                loader: 'less-loader' // 将 Less 编译为 CSS
            }
        ]
    }
};
```

`less-loader`只是将 Less 语法编译成 CSS，后续还需要使用`css-loader`和`style-loader`处理才可以，所以一般来说需要配合使用：

```
// webpack.config.js
module.exports = {
    //   ...
    module: {
        rules: [
            {
                test: /\.less$/,
                use: [
                    'style-loader',
                    {
                        loader: 'css-loader',
                        options: {
                            modules: true
                        }
                    },
                    'less-loader' // 将 Less 编译为 CSS
                ]
            }
        ]
    }
};
```


> Tips：注意一些预处理语言需要安装对应的解析器，例如 sass-loader，需要同时安装 node-sass：`npm install sass-loader node-sass --save-dev`

PostCSS：CSS 后处理器

好吧，可能是 CSS 实在是太弱了，有了预处理器之后，又出现了后处理器 PostCSS（另外也有称 PostCSS 也为另外一种 CSS 预处理器的），不过 PostCSS 的出现的确解决了很多问题，让我们写 CSS 更加轻松，类似不同浏览器前缀的写法，只需要使用引入一个名字叫Autoprefixer的 PostCSS 插件就可以使用标准的语法，在构建的过程中，PostCSS 会根据适配的浏览器使用 Autoprefixer 插件自动添加不同浏览器的适配。

```
/*没有前缀的写法*/
.flex {
    display: flex;
}

/*经过 postcss autoprefixer 处理后*/
.flex {
    display: -webkit-box;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
}
```

> Tips：PostCSS 是一个使用 JavaScript 插件来转换 CSS 的工具，PostCSS 核心是将 CSS 解析成 AST，然后通过各种插件做各种转换，最终生成处理后的新 CSS，跟 Babel 在功能和实现上都类似，这里就不再详细讲解实现原理了。在语法转换上还有一个开源项目cssnext，使用最新的 CSS 标准来写 CSS，通过 cssnext 可以转换成对应的 CSS 版本。

postcss-loader

使用 PostCSS 需要安装postcss-loader，然后按照 loader 顺序，在 css-loader 之前（注意 loader 顺序：从右到左，从后到前）加上 postcss-loader：

```
// webpack.config.js
module.exports = {
    //   ...
    module: {
        resule: [
            {
                test: /\.css$/,
                use: [
                    'style-loader',
                    {
                        loader: 'css-loader',
                        options: {
                            modules: true,
                            importLoader: 1
                        }
                    },
                    'postcss-loader'
                ]
            }
        ]
    }
};
```


如果有 CSS 预处理语言，则配置写法：

```
// webpack.config.js
module.exports = {
    //   ...
    module: {
        rules: [
            {
                test: /\.less$/,
                use: [
                    'style-loader',
                    {
                        loader: 'css-loader',
                        options: {
                            modules: true,
                            importLoader: 2
                        }
                    },
                    'less-loader',
                    'postcss-loader'
                ]
            }
        ]
    }
};
```

PostCSS 配置

通过 PostCSS 的强大插件系统，不仅可以处理 CSS 语法，还可以处理 CSS 预处理器的语法，实现的功能也有很多，包括添加前缀、最新语法转义、压缩等，甚至可以扩展 CSS 的语言特性。配置了 postcss-loader 之后，WebPack 就可以使用 PostCSS 来处理 CSS了。但是 PostCSS 本身只不过是将 CSS 解析成 AST，真正起作用的还需要依赖其强大的插件系统。

所以，PostCSS 配置其实主要是配置其使用哪些插件，PostCSS 的配置写法有以下三种方式：

1. 通过配置文件`postcss.config.js`，一般放置在项目的根目录下；
2. 通过 loader 的配置项`options`；
3. 直接在 package.json 中添加个`postcss`属性。

postcss.config.js

`postcss.config.js`完全是按 Node.js 模块写法来写，使用什么插件就引入什么插件依赖：

```
// postcss.config.js
const autoprefixer = require('autoprefixer');
module.exports = {
    plugins: [autoprefixer(['IE 10'])]
};
```

loader 配置项 options

在`webpack.config.js`中，直接配置了`postcss-loader`之后，然后通过`loader`的`options`可以配置 postcss 的参数。

```
// 引入postcss 插件
const autoprefixer = require('autoprefixer');
module.exports = {
    module: {
        rules: [
            {
                test: /\.css$/,
                use: [
                    'style-loader',
                    'css-loader',
                    {
                        loader: 'postcss-loader',
                        options: {
                            // 通过 plugins 选项
                            plugins: [autoprefixer(['IE 10'])]
                        }
                    }
                ]
            }
        ]
    }
};
```

package.json 中添加个 `postcss` 属性

最后一种方式是在 `package.json` 文件中添加 `postcss` 属性，**这种方式受限于 json 的语法，可扩展性较弱，一般不推荐！**

```
{
    "postcss": {
        "plugins": {
            "autoprefixer": "IE 10"
        }
    }
}
```

下面介绍几个项目中可能用到的 postcss 插件，带大家学习下 postcss 插件的基本用法。

Autoprefixer

`Autoprefixer` 这个插件前面内容已经简单提到过，就是给 css 补齐各种浏览器私有的前缀，例如 `-webkit`、`-moz`、`-ms` 等，当然还会处理各种兼容性问题，比如 flex 语法，不能简单添加 `-webkit` 就解决，还需要处理成 `-webkit-box` 这类老版本的标准。

Autoprefixer 还支持各种 IDE 插件，可以在 IDE 中直接转换对应的 css 文件（不推荐这样用，多人合作项目跟进 IDE 配置不同，转换的文件也会存在差异）。

Autoprefixer 的主要参数就是 browserslist，即需要代码支持的浏览器列表，这部分内容在 babel 章节已经介绍过了。其他相关的参数说明可以在文档中找到：https://github.com/postcss/autoprefixer#options。

postcss-preset-env

postcss-preset-env是跟 babel 的 preset-env 类似的功能，通过它可以安心的使用最新的 CSS 语法来写样式，不用关心浏览器兼容性，浏览器兼容的问题交给了 postcss-preset-env 和 WebPack，在打包构建的时候，会根据不同的配置输出对应支持的 CSS 文件。postcss-preset-env 支持的 CSS 标准，完全可以媲美 CSS 预处理器的功能，所以如果对 cssnext 新的标准比较熟悉，可以直接用新标准来写样式，这样等到浏览器支持新标准之后可以无缝切换到 cssnext 语法，那么可以直接抛弃 CSS 预处理器，直接使用 cssnext 语法来写样式，通过 WebPack 和 postcss-preset-env 来构建。

PreCSS

如果我们厌倦 cssnext 的变量定义方式，想使用 Sass 的语法，而又不想引入 Sass 这个 CSS 预处理器，PreCSS就是你的选择。使用 PreCSS，可以写类 Sass 和 cssnext 语法的 CSS，详细可以参考它的文档。

cssnano

cssnano是一个强大的 PostCss 插件，在 CSS 压缩优化中会经常被用到，它有别于常规的 CSS 压缩工具只是去除空格注释，还支持根据 CSS 语法解析结果**智能**压缩代码，比如合并一些类写法：

```
/*未经 cssnano 处理之前的 css*/
.a {
    background: red;
    color: yellow;
}
.b {
    font-size: bolder;
    background: red;
}
.c {
    color: yellow;
    text-align: center;
    font-size: bolder;
}
.d {
    display: flex;
    text-align: center;
}
```

经过 cssnano 处理之后的 CSS 文件，会合并压缩一些类，缩短一些常见的值，例如颜色值等：

```
/*经 cssnano 处理之后的 css*/
.a {
    color: #ff0;
}
.a,
.b {
    background: red;
}
.b,
.c {
    font-size: bolder;
}
.c {
    color: #ff0;
}
.c,
.d {
    text-align: center;
}
.d {
    display: flex;
}
```


cssnano 的配置会在 WebPack 优化章节继续详细介绍。

解惑：理解 css-loader 的 importLoaders 参数

在 css-loader 的文档中，有个比较引起疑惑的参数项：`importLoaders`，这个参数用于配置 **css-loader 作用于** `@import` **的资源之前**有多少个 loader。给出的示例代码如下：

```
{
  test: /\.css$/,
  use: [
    'style-loader',
    {
      loader: 'css-loader',
      options: {
        importLoaders: 2 // 0 => 默认，没有 loader; 1 => postcss-loader; 2 => postcss-loader, sass-loader
      }
    },
    'postcss-loader',
    'sass-loader'
  ]
}
```

通过示例并不能看出来配置`importLoaders`是否对项目打包有什么差异，下面通过实例代码来看下加上`importLoaders`和没添加有什么区别。首先我们创建两个文件：`style.css`和`body.css`，`style.css`中通过`@import 'body.css';`引入`body.css`：

```
/* style.css */
@import 'body.css';
body {
    /*background: yellow;*/
    font-size: 20px;
}
div {
    display: flex;
}
/* body.css */
.body-import {
    /* body import */
    display: flex;
}
```

在这两个文件中，分别添加了两个特殊属性的 CSS 值：`display: flex;`，目的是使用`autoprefixer`对其进行处理，如果 postcss-loader 都起作用，则`display: flex;`都会被处理添加对应的浏览器前缀，如果`importLoaders`设置不同，则根据文档输出的 CSS 会有差异，具体的差异就是我们需要理解的地方。

第二步，创建 entry 文件`import-loader.js`和 WebPack 配置文件`webpack.config.importLoader.js`，在 WebPack 配置文件中，一个 css-loader 没有使用`importLoaders`，一个使用了`importLoaders=1`，内容如下（为了方便查看 CSS 的差异，这里使用了mini-css-extract-plugin 直接打包出两个 CSS 文件）：

```
// entry: import-loader.js
import styles from './css/style.css';
console.log(styles);

// webpack.config.importLoader.js
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
module.exports = [
    {
        entry: './import-loader.js',
        mode: 'development',
        module: {
            rules: [
                {
                    test: /\.css$/,
                    use: [MiniCssExtractPlugin.loader, 'css-loader', 'postcss-loader']
                }
            ]
        },
        plugins: [
            // 添加 plugin
            new MiniCssExtractPlugin({
                filename: 'no-import-loaders.css'
            })
        ]
    },
    {
        entry: './import-loader.js',
        mode: 'development',
        module: {
            rules: [
                {
                    test: /\.css$/,
                    use: [
                        MiniCssExtractPlugin.loader,
                        {
                            loader: 'css-loader',
                            options: {
                                importLoaders: 1
                            }
                        },
                        'postcss-loader'
                    ]
                }
            ]
        },
        plugins: [
            // 添加 plugin
            new MiniCssExtractPlugin({
                filename: 'with-import-loaders.css'
            })
        ]
    }
];
```


第三步创建 PostCSS 配置文件，添加`autoprefixer`，增加一个 IE10 浏览器的配置：

```
// postcss.config.js
const autoprefixer = require('autoprefixer');
module.exports = {
    plugins: [autoprefixer(['IE 10'])]
};
```

都准备完毕了，下面执行命令`webpack --config webpack.importLoader.js`，打包后的文件内容如下：

```
/* no-import-loaders.css */
.body-import {
    /* body import */
    display: flex;
}

body {
    /*background: yellow;*/
    font-size: 20px;
}

div {
    display: -ms-flexbox;
    display: flex;
}
```

```
/* with-import-loaders.css */
.body-import {
    /* body import */
    display: -ms-flexbox;
    display: flex;
}

body {
    /*background: yellow;*/
    font-size: 20px;
}

div {
    display: -ms-flexbox;
    display: flex;
}
```


通过观察打包出来的两个 CSS 文件发现，使用`@import 'body.css'`引入了`body.css`文件之后，`body.css`的 CSS 因为配置了不同的`importLoaders`所以表现不一样：

- 未使用`importLoaders`：被`styles.css`引入的`body.css`内的`display: flex;`未添加了前缀，说明 postcss 没有作用到`@import`引入的文件中；
- 使用了`importLoaders=1`：被`styles.css`引入的`body.css`内的`display: flex;`也被添加了前缀，说明 postcss 作用到了被`@import`引入的文件中。

> Tips：除了设置 css-loader 的`importLoaders`，如果使用 PostCSS 则可以使用它的插件：postcss-import 同样可以处理`@import`引入的 CSS 代码：

```
const autoprefixer = require('autoprefixer');
const postcssImport = require('postcss-import');
module.exports = {
    plugins: [postcssImport(), autoprefixer(['IE 10'])]
};
```

小结

本小节主要介绍了 Webpack 中 CSS 相关的配置，主要内容包含：CSS Webpack 配置、CSS 预处理器配置和 PostCSS 配置。CSS 配置相对来说比较复杂，如果我们使用 CSS 的预处理器来编写代码，首先需要配置对应的预处理器 loader，将扩展的语法转成 CSS 代码，然后在配合 css-loader 和 style-loader。在生产环境推荐使用 mini-css-extract-plugin 将 CSS 内容导出到 CSS 文件来供页面单独引入。PostCSS 是一个强大的 CSS 后处理器，我们通过 PostCSS 的强大插件可以实现 CSS 前缀的自动添加（autoprefixer），还可以更加智能的实现 CSS 的压缩（cssnano）等功能。

本小节 Webpack 相关面试题：

1. 怎么使用 PostCSS 来处理 CSS；
2. 你会在 Webpack 中使用 CSS module 吗？
3. Webpack 的 style-loader 和 css-loader 有什么区别？

09 Webpack 中使用 TypeScript 开发项目

11 Webpack 中使用 lint 工具来保证代码风格和质量