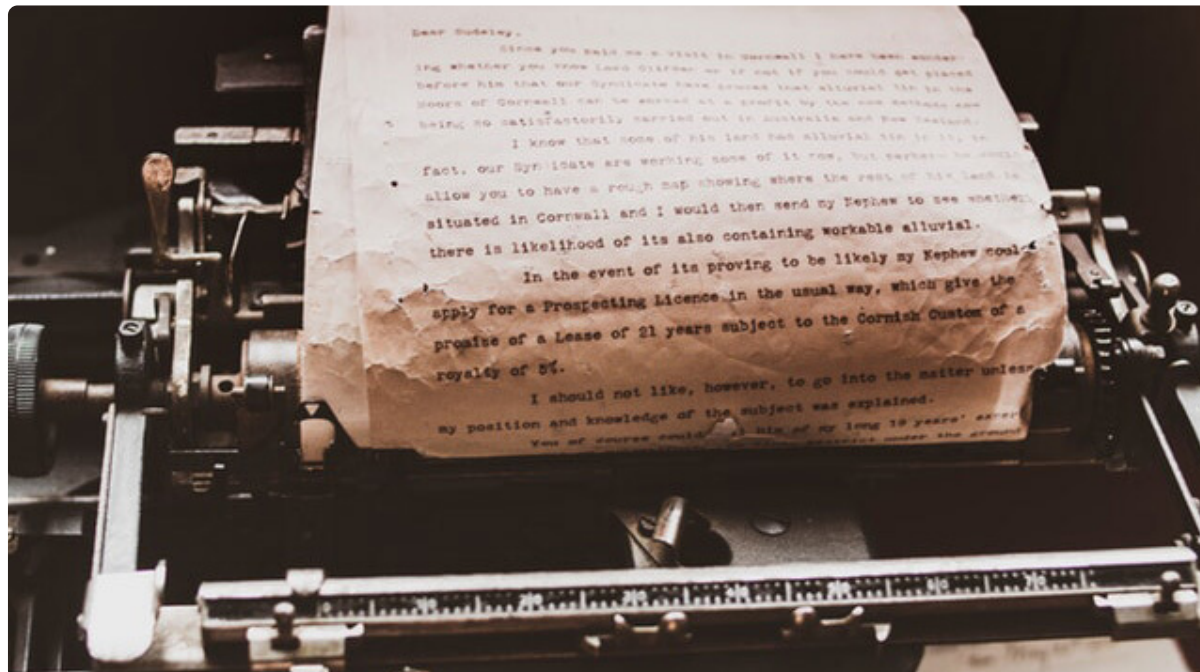


11 Webpack 中使用 lint 工具来保证代码风格和质量

更新时间：2019-06-24 09:26:10



“当你做成功一件事，千万不要等待着享受荣誉，应该再做那些需要的事。

—— 巴斯德”

代码不是写给机器看的，归根结底还是写给人看的！在团队的项目开发过程中，代码维护所占的时间比重往往大于新功能的开发。因此编写符合团队编码规范的代码是至关重要的，这样做不仅可以很大程度地避免基本语法错误，也保证了代码的可读性。

对于代码版本管理系统（SVN 和 GIT 或者其他），代码格式不一致带来的问题是严重的，在代码一致的情况下，因为格式不同，触发了版本管理系统标记为 diff，导致无法检查代码和校验。

本小节就简单介绍一下如何通过 Webpack 结合 ESLint 和 Stylelint 为代码库配置针对 JavaScript 与 CSS 的代码风格检查。

ESLint



ESLint 是国外大神 Nicholas C. Zakas（JavaScript「红宝书」作者）于 2013 年 6 月创建的开源项目。它的目标是提供一个插件化的 JavaScript 代码检测工具。

ESLint 是通过配置规则（Rules）来检测 JavaScript 语法规则的，目前 ESLint 的规则可以在[规则](#)页面查看。目前业内有很多针对团队代码风格制订的 ESLint 规则配置，然后生成一套自己的代码规范检查机制发布到开源社区供社区使用，业内比较著名的规范有：[Airbnb 的 JavaScript 代码规范](#)、[JavaScript Standard Style Guide](#)、[Google JavaScript 代码规范](#)，国内则有百度的[FECS](#)，这些代码规范都有对应的 ESLint 版本的配置规则。

Tips: 除了 ESLint, 前端业内还有同类型的代码检查工具 [JSLint](#) 和 [JSHint](#), 和它们相比, ESLint 对 ES6 语法支持更好, 这是我们选择使用 ESLint 的主要原因之一, 可以通过 ESLint 在团队内快速统一 ES6 的语法, 精简产品代码, 提高开发效率, 另外 ESLint 的扩展性很好, 能够很好的支持 JSX 语法的检测。

在项目中使用 ESLint, 需要先安装它的 CLI 工具: `npm install -D eslint`, 安装之后, 可以通过使用 `npx eslint` 直接运行, 在运行之前, 我们需要创建个 ESLint 的配置文件, 使用 `eslint --init` 命令创建 `.eslintrc.json`, 创建配置文件之前, 需要回答几个问题, 最后还会让选择代码风格, 这些问题只需要按照实际情况回答即可。下面是我回答的问题:

```
? How would you like to use ESLint? To check syntax, find problems, and enforce code style
? What type of modules does your project use? JavaScript modules (import/export)
? Which framework does your project use? None of these
? Where does your code run? (Press <space> to select, <a> to toggle all, <i> to invert selection) Browser
? How would you like to define a style for your project? Use a popular style guide
? Which style guide do you want to follow? Airbnb (https://github.com/airbnb/javascript)
? What format do you want your config file to be in? JSON
```

最后生成的 `.eslintrc.json` 内容如下:

```
{
  "env": {
    "browser": true,
    "es6": true
  },
  "extends": "airbnb-base",
  "globals": {
    "Atomics": "readonly",
    "SharedArrayBuffer": "readonly"
  },
  "parserOptions": {
    "ecmaVersion": 2018,
    "sourceType": "module"
  },
  "rules": {}
}
```

如果需要安装类似 `airbnb`、`google` 和 `standard` 的规则, 则可以首先查找规则对应的 ESLint 配置 NPM 模块, 然后安装它:

```
# airbnb
npm install --save-dev eslint-config-airbnb
# google
npm install --save-dev eslint-config-google
# standard
npm install --save-dev eslint-config-standard
```

然后在 `.eslintrc.json` 配置文件中修改 `extends` 到对应值:

```
{
  "extends": "google",
  "rules": {
    // Additional, per-project rules...
  }
}
```

一般 ESLint 的 rules 需要根据团队的规范来制定, 出了这点之外, 我个人推荐添加下面几点 rule:

```
{
  'rules': {
    // 禁止 console, 要用写 eslint disable
    'no-console': 2,
    // 禁止 debugger, 防止上线
    'no-debugger': 2,
    // 禁止 alert, 要用写 eslint disable
    'no-alert': 2,
    // 不用的 var, 要删除, 手动 tree shaking, 要洁癖
    'no-unused-vars': 2,
    // 没定义就用的就别用, 全局的要用 写 eslint global
    'no-undef': 2
  }
}
```

Tips: ESLint 的报错类型包括三种: **off**、**warn** 和 **error**, 分别对应着: 0、1、2, 所以上面的配置的 rule 实际为 **error** 级别的规则, 检测到了则直接报错误 (Error)。

这几个 rule 是为了防止线下 debug 代码上到线上去的, 曾经有段血泪史一不小心将 **alert** 和 **debugger** 上到线上去, 加上这几个 rule 就可以得到代码提示, 如果在代码中真的需要用到 **alert**, 可以使用 ESLint 的注释:

```
// eslint-disable-next-line
alert('我就是要用 alert'); // eslint-disable-line
```

Webpack 中使用 ESLint

首先需要安装 **eslint-loader**:

```
npm install -D eslint-loader
```

然后在 webpack.config.js 中 **module.rules** 添加如下代码:

```
{
  test: /\.js$/,
  loader: 'eslint-loader',
  enforce: 'pre',
  include: [path.resolve(__dirname, 'src')], // 指定检查的目录
  options: { // 这里的配置项参数将会被传递到 eslint 的 CLIEngine
    formatter: require('eslint-friendly-formatter') // 指定错误报告的格式规范
  }
}
```

1. ESLint 的配置单独做了一个 Webpack 的 **module.rule** 配置, 所以使用了 **enforce: 'pre'** 来调整了 loader 加载顺序, 保证先检测代码风格, 之后再做 Babel 转换等工作;
2. 也可以放到 Babel 放到一起, 不过要将 **eslint-loader** 放到 **babel-loader** 之前检测;
3. 这里为了让 ESLint 报错更加好看一些, 使用了 **eslint-formatter-friendly** 这个 ESLint **formatter**, 记得安装它: **npm i -D eslint-formatter-friendly**

新建一个 entry 文件, 内容如下:

```
import _ from 'lodash';
const a = 1;
const b = 2;
alert(b);
console.log(b);
```

执行 `npm run webpack` 之后，ESLint 报错内容如下：

```
Hash: b0a267ece39ac70855fc
Version: webpack 4.29.6
Time: 824ms
Built at: 2019-04-13 14:06:11
    Asset      Size  Chunks             Chunk Names
main.js    533 KiB       0  [emitted]  main
Entrypoint main = main.js
[./../node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 472 bytes {main} [built]
[./../node_modules/webpack/buildin/module.js] (webpack)/buildin/module.js 497 bytes {main} [built]
[./src/index.js] 76 bytes {main} [built] [1 error]
+ 1 hidden module

ERROR in ./src/index.js
Module Error (from ./node_modules/eslint-loader/index.js):

  X https://google.com/search?q=import%2Fnewline-after-import Expected 1 empty line after import statement not followed by another import
    src/index.js:1:1

  X https://eslint.org/docs/rules/no-unused-vars      '_' is defined but never used
    src/index.js:1:8

  X https://eslint.org/docs/rules/no-unused-vars      'a' is assigned a value but never used
    src/index.js:2:7

  X https://eslint.org/docs/rules/no-alert            Unexpected alert
    src/index.js:4:1

  X https://eslint.org/docs/rules/no-console          Unexpected console statement
    src/index.js:5:1

X 5 problems (5 errors, 0 warnings)

Errors:
   2 https://eslint.org/docs/rules/no-unused-vars
   1 https://eslint.org/docs/rules/no-console
   1 https://eslint.org/docs/rules/no-alert
   1 https://google.com/search?q=import%2Fnewline-after-import
```

有关 ESLint 的更多使用方式，请参考[中文官网内容](#)，这里不再做展开介绍。

Tips: TypeScript 也有自己版本的 ESLint，可以使用[tslint-loader](#)进行检测。

StyleLint



检测 CSS 语法使用[StyleLint](#)。StyleLint 和 ESLint 很像，它们都只是提供了工具与规则，如何配置这些规则完全取决于使用者，所以我们要根据需要自己引入或配置规则。StyleLint 的代码风格也有很多社区开源版本，官方推荐的代码风格有两个：

- [stylelint-config-recommended](#)
- [stylelint-config-standard](#)

要使用 StyleLint 需要先安装它：

```
npm install -D stylelint
```

Tips: 除了 StyleLint 本身之外，还可以安装[stylelint-order](#) 插件，该插件的作用是强制我们在写 CSS 的时候按照某个顺序来编写。例如先写定位，再写盒模型，再写内容区样式，最后写 CSS3 相关属性。这样可以极大的保证我们代码的可读性和风格统一。

StyleLint 的配置文件是 `.stylelintrc.json`，其中的写法跟 ESLint 的配置类似，都是包含 `extend` 和 `rules` 等内容，下面是一个示例：

```
{
  "extends": ["stylelint-config-standard", "stylelint-config-recess-order"],
  "rules": {
    "at-rule-no-unknown": [true, {"ignoreAtRules": ["mixin", "extend", "content"]}]]
  }
}
```

配置文件中单独配置 `at-rule-no-unknown` 是为了让 StyleLint 支持 SCSS 语法中的 `mixin`、`extend`、`content` 语法，更多详细的规则，可以查看[官方文档](#)。

Webpack 中使用 StyleLint

Webpack 中使用 StyleLint 是通过插件的方式来使用，这个插件的名字是 `stylelint-webpack-plugin`。

```
npm install -D stylelint-webpack-plugin
```

安装之后，按照插件的使用方式在 `webpack.config.js` 添加配置：

```
const StyleLintPlugin = require('stylelint-webpack-plugin');

module.exports = {
  // ...
  plugins: [new StyleLintPlugin(options)]
  // ...
};
```

默认 `StyleLint-webpack-plugin` 会查找项目中的 StyleLint 配置文件，根据配置文件的配置来检测 CSS 代码。

在 `stylelint-webpack-plugin` 插件中有两个跟 Webpack 编译相关的配置项：

- `emitErrors`：默认是 `true`，将遇见的错误信息发送给 webpack 的编辑器处理；
- `failOnError`：默认是 `false`，如果是 `true` 遇见 StyleLint 报错则终止 Webpack 编译。

小结

本小节主要介绍了 Webpack 中结合 ESLint 和 StyleLint 对 JavaScript 和 CSS 文件进行代码风格检测。结合 `eslint-loader` 和 `stylelint-webpack-plugin` 可以在 Webpack 编译的过程中检测出代码中不符合规范的部分，提醒开发者及时修改代码。Webpack 项目中配置 ESLint 和 StyleLint 可以提高项目的可维护性，保持团队代码风格统一。