前端代码检查与代码规范

DM6 2019-12-04 15:05 5238

一、背景

本篇介绍的是如何做到在代码提交前，统一团队代码风格，检查代码质量，并修复一些低级错误。最终期待项目中的开发人员提交的代码都符合代码规范、风格统一。

二、组合技

Git Hook + lint-staged + Prettier + ESLint，先介绍最终实现，具体每个模块的作用和配置后面有各自的介绍。

2.1 实现步骤

- 准备好待提交的代码
- git add . 添加到暂存区
- 执行 git commit
- husky注册在git pre-commit的钩子调起 lint-staged
- lint-staged 取得所有被提交的文件依次执行写好的任务（ Prettier + ESLint）
- 如果有错误（没通过 ESLint 检查）则停止任务，等待下次commit，同时打印错误信息
- 成功提交

2.2 安装包


```
npm i --save-dev husky lint-staged prettier

npm i --save-dev eslint babel-eslint eslint-plugin-react
```

2.3.1 .eslintrc

复制代码

```
{
  "env": {
    "browser": true,
    "es6": true
  }
  "eslint": "recommended",
  "parserOptions": {
    "ecmaFeatures": {
      "jsx": true,
      "impliedStrict": true
    }
  },
  "plugin: react",
  "rules": {
    "quotes": [
      "error",
      "single"
    ]
  }
}
```

2.3.2 .prettier.js

复制代码

```
module.exports = {
  // 一行最多 100 字符
  printWidth: 100,
  // 使用 4 个空格缩进
  tabWidth: 4,
  // 不使用缩进符，而使用空格
  useTabs: false,
  // 行尾需要有分号
  semi: false,
  // 使用单引号
  singleQuote: true,
  // 对象的 key 仅在必要时用引号
  quoteProps: 'as-needed',
```

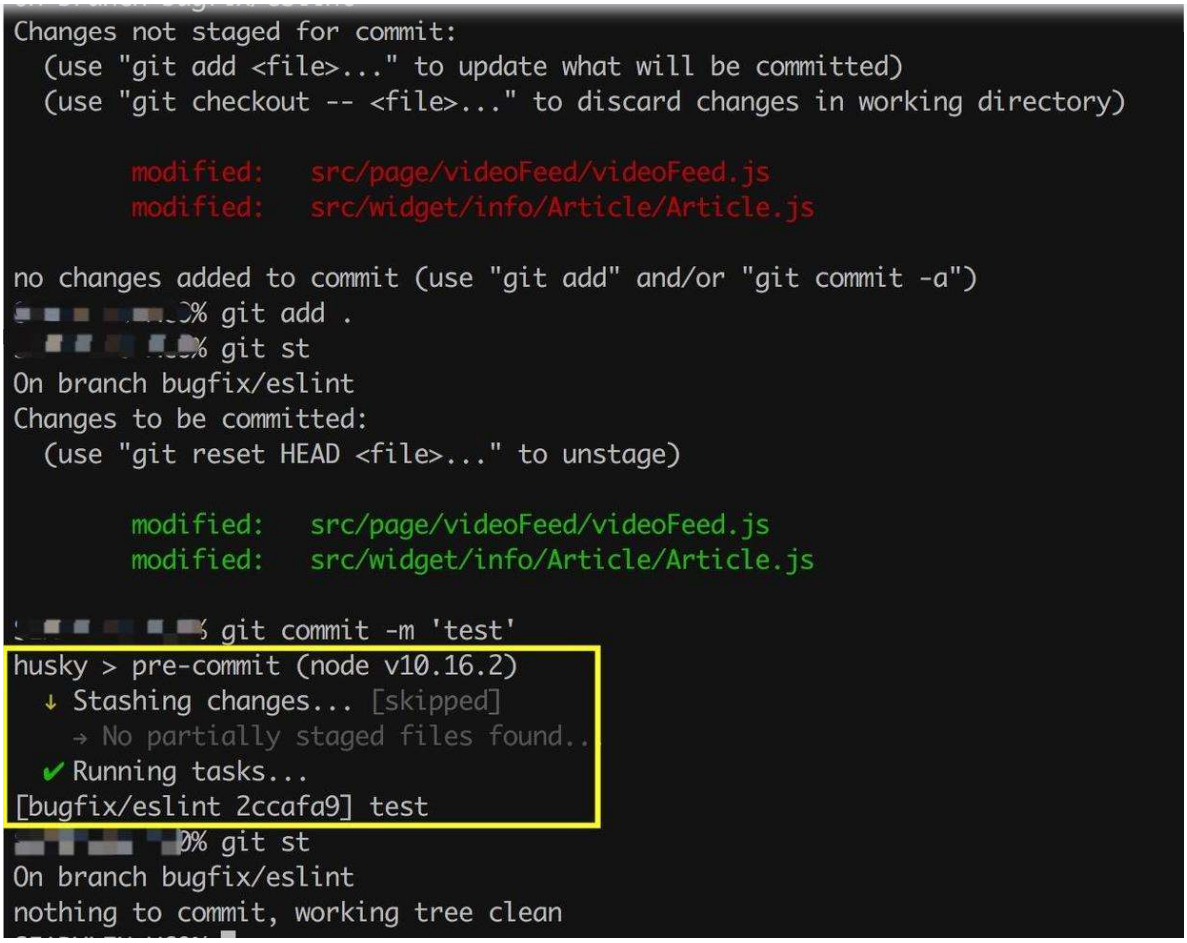
```
  // 大括号内的首尾需要空格
  bracketSpacing: true,
  // jsx 标签的反尖括号需要换行
  jsxBracketSameLine: false,
  // 箭头函数，只有一个参数的时候，也需要括号
  arrowParens: 'always',
  // 每个文件格式化的范围是文件的全部内容
  rangeStart: 0,
  rangeEnd: Infinity,
  // 不需要写文件开头的 @prettier
  requirePragma: false,
  // 不需要自动在文件开头插入 @prettier
  insertPragma: false,
  // 使用默认的折行标准
  proseWrap: 'preserve',
  // 换行符使用 lf
  endOfLine: 'lf'
}
```

2.3.3 package.json

复制代码

```
{
  ...
  "husky": {
    "hooks": {
      "pre-commit": "lint-staged"
    }
  },
  "lint-staged": {
    "src/**/*.{js,jsx}": [
      "prettier --write",
      "eslint --fix",
      "git add ."
    ]
  },
  ...
}
```

2.3.4 运行结果



上面的配置只是一种组合方式的举例，还可以自由组合为多种检查方式，这里就不一一列举了。后面是对各个模块的介绍，感兴趣可以继续阅读。

二、Prettier 的介绍和配置

2.1 Prettier 是什么?

- Prettier 作为强大的代码格式化工具，能够完全统一你和同事的代码风格 (这真的很重要...)
- 同时结合 ESLint 规则和自身配置的规则，进行代码自动格式化
- Prettier 只会做代码风格的统一，并不会检查代码规范，关于代码规范的检查应该交给 ESLint

2.2 Prettier 的必要性

- 简单的配置即可自动格式化并统一代

2.3 如何配置?

2.3.1 在编辑器中安装插件

- 目前开发使用的是 `vscode` 编辑器，在扩展中查找 `Prettier - Code formatter` 插件安装后即可使用。不同编辑器的插件
- 安装后打开需要格式化的文件，按照下面的方式可以格式化整个文件或者格式化选定的代码片段

复制代码

```
1. CMD + Shift + P -> Format Document
OR
1. Select the text you want to Prettify
2. CMD + Shift + P -> Format Code
```

[图片上传失败...(image-4acd90-1575447043750)]

2.3.2 使用配置文件

1. 创建规则文件：在根目录创建 `.prettierrc` 文件（支持 json 配置格式）或者 创建一个 `.prettierrc.js` 配置文件
2. 在 `package.json` 文件的 `scripts` 中添加对应的代码

复制代码

```
{
  ...
  "scripts": {
    "format": "prettier --write 'src/**/*.{js,jsx,css,less}'",
    ...
  },
  ...
}
```

3. 在控制台执行 `npm run format` 即可，置范围内的所有文件

3.1 ESLint 是什么?

ESLint 是一个代码检查工具，它能够被开发者灵活的配置，使其能够满足制定好的代码规范的要求，并且在编码过程中实时检测输入的代码，对于不符合代码规范的代码警告或报错。

3.2 ESLint 的必要性

- 编码阶段及时发现问题
- 保证代码遵循最佳实践
- 统一代码编写规范

3.3 如何使用

3.3.1 配置方法

- **配置注释**：使用JavaScript注释将配置信息直接嵌入到文件中
- **配置文件**：使用JavaScript，JSON或YAML文件指定配置信息

3.3.2 配置文件说明

1. **文件格式**

- **JavaScript**：使用 `.eslintrc.js` 并导出包含您的配置的对象。
- **YAML**：使用 `.eslintrc.yaml` 或 `.eslintrc.yml` 定义配置结构。
- **JSON**：用于 `.eslintrc.json` 定义配置结构。
- 使用 `.eslintrc` ，可以是 JSON 或 YAML。
- **package.json**： `eslintConfig` 在你的 `package.json` 文件中创建一个属性并在那里定义你的配置。

2. **环境**：

> 脚本设计运行的环境。每个环境都带有一组预定义的全局变量。

```
  "node": true
}
```

3. **全局变量**：

> 脚本在执行期间访问的其他全局变量。

复制代码

```
"globals": {
  "React": true,
  "MtaH5": true,
  "TencentGDT": true
}
```

4. **解析器选项**：

> ESLint允许你指定你想要支持的JavaScript语言选项。默认情况下，ESLint需要ECMAScript 5语法。您可以覆盖该设置，以使用解析器选项启用对其他ECMAScript版本以及JSX的支持。

复制代码

```
"parserOptions": {
  "ecmaVersion": 6,
  "sourceType": "module", // 设置为`"script"`（默认）或者`"module"`您的代码位于ECMAScript模块中。
  "ecmaFeatures": { // 一个对象，指示您想要使用哪些其他语言功能
    "jsx": true, // 启用JSX
    "impliedStrict": true // 启用全局严格模式
  }
},
```

5. **插件**

> 要在配置文件中配置插件，请使用 `plugins` 包含插件名称列表的密钥。该 `eslint-plugin-` 前缀可以从插件名称被省略。

```
  "eslint-plugin-react"
]
```

6. **扩展**

> 扩展就是直接使用别人已经写好的 lint 规则， `rules` 属性中配置的规则都是基于这个规则之上配置的

复制代码

```
"extends": [
  "standard"
  "eslint:recommended",
  "plugin:react/recommended"
],
```

推荐的扩展配置：

standardjs、airbnb、eslint-config-alloy

7. **规则**：

> ESLint 附带有大量的规则。你可以使用注释或配置文件修改你项目中要使用的规则。要改变一个规则设置，你必须将规则 ID 设置为下列值之一

- `"off"` 或 `0` ：关闭规则
- `"warn"` 或 `1` ：开启规则，使用警告级别的错误(不会导致程序退出)
- `"error"` 或 `2` ：开启规则，使用错误级别的错误(当被触发的时候，程序会退出)

复制代码

```
"rules": {
  "strict": 2,
}
```


3.3.3 如何检测

复制代码

```
{
 ...
  "scripts": {
    "lint": "eslint ./src"
    "lintFix": "eslint ./src --fix"
  },
  ...
}
```

在控制台运行如下命令：

复制代码

```
// 代码检查
npm run lint
// 代码检查并修复
npm run lintFix
```

四、Git Hook 的介绍和配置

4.1 Git 钩子

和其它版本控制系统一样，Git 能在特定的重要动作发生时触发自定义脚本。这里介绍的是提交工作流钩子

4.1.1 pre-commit

- 钩子在键入提交信息前运行
- 用于检查即将提交的快照
- 如果该钩子以非零值退出，Git 将放弃此次提交
- 可以用 `git commit --no-verify` 来绕过这个环节
- 可以利用该钩子，来检查代码风格是否一致

4.1.2 prepare-commit-msg

4.1.3 如何配置

将配置添加到 `package.json` 中，执行命令添加到 `scripts` 脚本中：

复制代码

```
{
  "scripts": {
    "lint": "eslint ./ --cache --ignore-pattern .gitignore",
    "precommit-msg": "echo 'Pre-commit checks...' && exit 0"
  },
  "pre-commit": [ "precommit-msg", "lint" ],
  "devDependencies": {
    "eslint": "^2.12.0",
    "pre-commit": "^1.1.3"
  }
}
```

4.2 Husky

在我们的 `package.json` 中配置 husky，并且在对应的 git hook 阶段来执行对应的命令。因此，不用繁琐的去配置 git hook 阶段的脚本文件了。

复制代码

```
{
  "husky": {
    "hooks": {
      "pre-commit": "npm test",
      "pre-push": "npm test"
    }
  }
}
```

五、lint-staged

lint-staged的作用是每次提交只检查本次 .j文件，其中 staged 是 Git 里面的概

```
// package.json
{
  ...
  "scripts": {
    "precommit": "lint-staged", // git commit 执行这个命令，这个命令在调起 lint-staged
  },
  "lint-staged": {   // lint-staged 配置
    "src/**/*.{js,jsx}": [
      "prettier --write",
      "eslint --fix",
      "git add"
    ]
  },
  ...
}
```

六、总结

上面整理了一些项目实践过程中的方式和方法。方法的使用可以提高项目的质量。有规范的代码，一定程度可以减少问题的发生。代码的可读性也有所提高。

标签： ESLint 前端

评论

全部评论 8

最新 最热

e015be70172... JY.1 3年前

[内容违规]

点赞 回复 ···

d05e90d0171... JY.1 3年前

[内容违规]

点赞 回复 ···

8c4d452016b... JY.1 3年前

好文顶

点赞 2 ···

8c4d452016b411e... 3年前

[内容违规]

点赞 回复 ···

微少 3年前

[内容违规]

点赞 回复 ···

27d61a00167... JY.1 3年前

[内容违规]

点赞 回复 ···

27d61a00167... JY.1 3年前

[内容违规]

点赞 1 ···

相关推荐

WEB 前端规范
Sandy485 6年前 1.1w 663 12

前端开发规范手册
F48VJ 7年前 6734 302 1

前端代码规范
fanyaohui 3年前 212 点赞 2

腾讯AlloyTeam：坚持既定代码规范！ （上）
CF14年老兵 3年前 2574 8 评论

前端-团队效率（二）代码规范
吴文周 3年前 1610 9 评论

前端项目规范
Seenroot 4年前 577 点赞 评论

很多大厂都在用的前端代码规范（来自凹凸实验室）
Tian13 1年前 478 1 评论

前端优秀的命名规范都是这样做的
情系半生e 2年前 734 1 评论

吐血整理的前端代码规范系列 -- Vue 代码规范
关索 3年前 1694 3 3

前端代码编写规范
chengliu0508 2年前 209 8 4

代码风格统一：利用husky, prettier统一[illegible]的代码

前端代码篇总结

_米花 3年前 4326 44 8

3大类15小类前端代码规范，让团队代码统一规范起来！

南极一块修炼千年的大... 2年前 1063 13 3

Vue 前端代码开发规范

Hisen 1年前 1968 15 评论