规范化标准

为什么要有规范化标准

软件开发需要多人协同

不同开发者具有不同的编码习惯和喜好

不同的喜好增加项目维护成本

每个项目或者团队需要明确统一的标准

哪里需要规范化标准

代码、文档、甚至是提交日志

开发过程中人为编写的成果图

代码标准化规范最为重要

实施规范化的方法

编码前人为的标准约定

通过工具实现Lint

常见的规范化实现方式

ESLint 工具使用

定制ESLint校验规则

ESLint对TypeScript的支持

ESLint结合自动化工具或者Webpack

基于ESLint的衍生工具

StyleLint工具的使用

一、ESLint

1. ESLint介绍

最为主流的JavaScript Lint工具，检测JS代码质量

ESLint很容易统一开发者的编码风格

ESLint可以帮助开发者提升编码能力

2. ESLInt安装

初始化项目: yarn init -y

安装ESLint模块为开发依赖: yarn add eslint --dev

通过CLI命令验证安装结果：yarn eslint -v

3. ESLint 检查步骤

编写“问题”代码

使用eslint执行检测 : 执行yarn eslint 01-prepare.js，执行自动修复yarn eslint 01-prepare.js --fix

完成eslint使用配置 : yarn eslint --init

4. ESLint配置文件解析

module.exports = {

env: {

// 运行的环境，决定了有哪些默认全局变量

browser: true,

es2020: true

},

// eslint 继承的共享配置

extends: [

'standard'

],

// 设置语法解析器，控制是否允许使用某个版本的语法

parserOptions: {

ecmaVersion: 11

},

// 控制某个校验规则的开启和关闭

rules: {

'no-alert': 'error'

},

// 添加自定义的全局变量

globals: {

"$": 'readonly',

}

}

1234567891011121314151617181920212223

5. ESLint配置注释

将配置写在代码的注释中，然后再对代码进行校验

const str1 = "${name} is coder" // eslint-disable-line no-template-curly-in-string

console.log(str1)

123

6. ESLint 结合自动化工具

集成之后，ESLint一定会工作

与项目统一，管理更加方便

7. ESLint结合Webpack

eslint通过loader形式校验JavaScript代码

前置工作：

git clone 仓库

安装对应模块

安装eslint模块

安装eslint-loader模块

初始化.eslintrc.js配置文件

后续配置：

reles: {

'react/jsx-uses-react': 2

},

plugins: [

'react'

]

123456

8 现代化项目集成ESLint

9 ESLint检查TypeScript

二、StyleLint

1. StyleLint使用介绍

提供默认的代码检查规则

提供CLI工具，快速调用

通过插件支持Sass、Less、PostCSS

支持Gulp或者Webpack集成

npm install stylelint --dev

npx stylelint ./index.css

npm install stylelint-config-sass-guidelines

.stylelintrc.js

module.exports = {

extends: [

"stylelint-config-standard",

"stylelint-config-sass-guidelines"

]

}

123456

运行：npx stylelint ./index.css

三、Prettier 的使用

近两年流行的前端代码通用型格式化工具，几乎可以完成各种代码的格式化。

yarn add prettier --dev安装prettier到当前项目

yarn prettier style.css将格式化的结果输出到命令行

yarn prettier style.css --write 将格式化的结果覆盖原文件

yarn prettier . --write对当前整个项目进行格式化

四、Git Hooks 介绍

代码提交至仓库之前为执行lint工作

Git Hook也称为Git钩子，每个钩子都对应一个任务

通过shell脚本可以编写钩子任务出发时要具体执行的操作

在一个Git仓库中，进入.git/hooks目录，然后看到很多sample文件，执行cp pre-commit.sample pre-commit,拷贝了一份pre-commit文件出来，把里面的内容先去掉，就写一句简单的echo看看Git钩子的效果(第一行是可执行文件必须要有的固定语法，不可以删除)

#!/bin/sh

echo "git hooks"

12

然后回到仓库根目录，执行git add .,git commit -m"xx"

就可以看到输出了git hooks，说明pre-commit这个钩子已经生效了

五、ESLint结合Git Hooks

很多前端开发者并不擅长使用shell，Husky可以实现Git Hooks的使用需求

在已有了eslint的Git项目中，安装husky，实现在Git commit的时候，进行lint

yarn add husky --dev

package.json

{

"name": "GitHooks",

"version": "1.0.0",

"main": "index.js",

"author": "jiailing <517486222@qq.com>",

"license": "MIT",

"scripts": {

"test": "eslint ./index.js"

},

"devDependencies": {

"eslint": "^7.3.1",

"eslint-config-standard": "^14.1.1",

"eslint-plugin-import": "^2.21.2",

"eslint-plugin-node": "^11.1.0",

"eslint-plugin-promise": "^4.2.1",

"eslint-plugin-standard": "^4.0.1",

"husky": "^4.2.5"

},

"husky": {

"hooks": {

"pre-commit": "yarn test"

}

}

}

12345678910111213141516171819202122232425

然后执行

echo node\_modules > .gitignore

git add .

git commit -m "husky"

可以看到我们的index.js的代码报错被输出到控制台了，并且Git commit失败了。

说明husky已经完成了在代码提交前的lint工作。不过husky并不能对代码进行格式化，此时可以使用lint-stage

yarn add lint-staged --dev

package.json

{

"name": "GitHooks",

"version": "1.0.0",

"main": "index.js",

"author": "jiailing <517486222@qq.com>",

"license": "MIT",

"scripts": {

"test": "eslint ./index.js",

"precommit": "lint-staged"

},

"devDependencies": {

"eslint": "^7.3.1",

"eslint-config-standard": "^14.1.1",

"eslint-plugin-import": "^2.21.2",

"eslint-plugin-node": "^11.1.0",

"eslint-plugin-promise": "^4.2.1",

"eslint-plugin-standard": "^4.0.1",

"husky": "^4.2.5",

"lint-staged": "^10.2.11"

},

"husky": {

"hooks": {

"pre-commit": "yarn precommit"

}

},

"lint-staged": {

"\*.js": [

"eslint",

"git add"

]

}

}