**Eslint 超简单入门教程**

**官网：** [**https://eslint.bootcss.com/docs/user-guide/getting-started**](https://eslint.bootcss.com/docs/user-guide/getting-started)

两个月之前在项目中就开始使用 Eslint ，当时直接 copy 别人的 .eslintrc.js 文件，感觉好复杂，一直没吃透，中间去把 Eslint 官方文档看了数遍，依然无果。今天刚好没事，回过头来想整理一下 Eslint 的使用，突然发现变得好简单。总结下在这过程中走得弯路。

* 先要知道 Lint 是什么，Eslint 又是什么；
* 因为想要的太多（想把 Eslint 官方文档里每个字都记住）就容易抓不住重点（目的是在项目中使用 Eslint，这仅仅是个工具，工具又怎么会搞的太难）。

**目标**

学习完本教程希望对下面这个 .eslintrc.js 文件能够做到心里有数。

// .eslintrc.js

module.exports = {

"extends": "airbnb",

"rules": {

"semi": [2, "never"],

"no-console": 0,

"comma-dangle": [2, "always-multiline"],

"max-len": 0,

"react/jsx-first-prop-new-line": 0,

"react/jsx-filename-extension": 0,

"space-before-function-paren": [2, "always"],

"no-unused-expressions": [0, {

"allowShortCircuit": true,

"allowTernary": true

}],

"arrow-body-style": [0, "never"],

"func-names": 0,

"prefer-const": 0,

"no-extend-native": 0,

"no-param-reassign": 0,

"no-restricted-syntax": 0,

"no-eval": 0,

"no-continue": 0,

"react/jsx-no-bind": 0,

"no-unused-vars": [2, { "ignoreRestSiblings": true }],

"no-underscore-dangle": 0,

"global-require": 0,

"import/no-unresolved": 0,

"import/extensions": 0,

"jsx-a11y/href-no-hash": 0,

"react/no-array-index-key": 0,

"react/require-default-props": 0,

"react/forbid-prop-types": 0,

"react/no-string-refs": 0,

"react/no-find-dom-node": 0,

"import/no-extraneous-dependencies": 0,

"import/prefer-default-export": 0,

"react/no-danger": 0,

"jsx-a11y/no-static-element-interactions": 0,

},

"parser": "babel-eslint",

"parserOptions": {

"sourceType": "module",

"ecmaVersion": 8,

"ecmaFeatures": {

"jsx": true,

"experimentalObjectRestSpread": true

}

},

"settings": {

"import/resolver": "node"

}

};

**Eslint 是什么**

**编码规范**

每个程序员都有自己的编码习惯，最常见的莫过于：

* 有的人写代码一行代码结尾必须加分号 ;，有的人觉得不加分号 ; 更好看；
* 有的人写代码一行代码不会超过 80 个字符，认为这样看起来简洁明了，有的人喜欢把所有逻辑都写在一行代码上，觉得别人看不懂的代码很牛逼；
* 有的人使用变量必然会先定义 var a = 10;，而粗心的人写变量可能没有定义过就直接使用 b = 10;；

**Lint 的含义**

如果你写自己的项目怎么折腾都没关系，但是在公司中老板希望每个人写出的代码都要符合一个统一的规则，这样别人看源码就能够看得懂，因为源码是符合统一的编码规范制定的。

那么问题来了，总不能每个人写的代码老板都要一行行代码去检查吧，这是一件很蠢的事情。凡是重复性的工作，都应该被制作成工具来节约成本。这个工具应该做两件事情：

* 提供编码规范；
* 提供自动检验代码的程序，并打印检验结果：告诉你哪一个文件哪一行代码不符合哪一条编码规范，方便你去修改代码。

Lint 因此而诞生。

**Eslint 的含义**

Lint 是检验代码格式工具的一个统称，具体的工具有 Jslint 、 Eslint 等等 ...........

我们可以形象地将 Lint 看成是电商行业，而电商行业具体表现有淘宝（Eslint）、京东（Jslint）等。

**使用 Eslint**

确保你的电脑安装了 node 和 npm 环境

**创建项目**

npm init 指令会在项目根目录下生成 package.json 文件。

$ d:

$ cd d:

$ mkdir test-eslint

$ cd test-eslint

$ npm init

**安装依赖包**

--save-dev 会把 eslint 安装到 package.json 文件中的 devDependencies 属性中，意思是只是开发阶段用到这个包，上线时就不需要这个包了。

$ npm install eslint --save-dev

**设置 package.json 文件**

打开 package.json 文件，修改 script 属性如下：

"scripts": {

"test": "react-scripts test --env=jsdom",

"lint": "eslint src",

"lint:create": "eslint --init"

}

* script 属性的意思是脚本，使用方法是在 cmd 窗口中输入 npm run 指令 的形式，如：npm run lint:create；
* "lint:create": "eslint --init" 这个脚本是为了生成 .eslintrc.js 文件，在介绍 Lint 的时候说到 Lint 应该提供编码规范，规范写在哪里，就写在这个文件，所以我们需要创建它；
* "lint": "eslint src" 在介绍 Lint 的时候也说到 Lint 应该提供自动校验代码的程序，这个脚本是让 Lint 自动检验 src 目录下所有的 .js 文件。

**创建 .eslint.js 文件**

$ npm run lint:create

> 20170811@0.1.0 lint:create D:\code\test\20170811

> eslint --init

? How would you like to configure ESLint? Answer questions about your style // 以问答的形式创建配置文件

? Are you using ECMAScript 6 features? Yes // 是否校验 Es6 语法

? Are you using ES6 modules? Yes // 是否校验 Es6 模块语法

? Where will your code run? Browser // 代码运行环境，Browser 指浏览器

? Do you use CommonJS? Yes // 是否校验 CommonJs 语法

? Do you use JSX? Yes // 是否校验 JSX 语法

? Do you use React? Yes // 是否校验 React 语法

? What style of indentation do you use? Tabs // 首行空白选择 Tab 键还是 Space

? What quotes do you use for strings? Double // 字符串使用单引号 'string' 还是双引号 "string"

? What line endings do you use? Windows // 操作系统

? Do you require semicolons? Yes // 每行代码结尾是否校验加分号 ;

? What format do you want your config file to be in? JavaScript // 以 .js 格式生成配置文件

Installing eslint-plugin-react@latest // 因为要校验 Reac 语法，所以这里需要下载一个 React 语法规则的包

创建完成后根目录下应该会出现 .eslintrc.js 文件

**创建 index.js 文件**

在根目录下创建 src/index.js 文件，内容如下，接下来就使用 Eslint 来检验这个 .js 文件是否符合编码规范。

const lint = 'eslint'

此时的目录结构应该为：

- test-eslint

+ .eslintrc.js

+ package.json

- src

+ index.js

**校验代码**

$ npm run lint

1:7 error 'lint' is assigned a value but never used no-unused-vars

1:14 error Strings must use doublequote quotes

1:22 error Missing semicolon semi

3 problems (3 errors, 0 warnings)

2 errors, 0 warnings potentially fixable with the `--fix` option.

这里报了三个错误，分别是：

* index.js 第1行第7个字符，报错编码规则为 no-unused-vars：变量 lint 只定义了，但是未使用；
* index.js 第1行第14个字符，报错编码规则为 quotes：编码规范字符串只能使用双引号，这里却使用了单引号；
* index.js 第1行第22个字符，报错编码规则为 semi：编码规范每行代码结尾必须加分号，这里没有加分号。

当我们熟悉了编码规范之后，只需进行响应的修改就可以使代码形成统一的风格。刚开始如果对编码规范具体某一条规则不了解的话，可以在 [eslint规则表](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Frules%2F" \t "_blank) 查看用法。（不建议去背规则表，而是用到什么查什么，把它当成字典来用，不那么累）

**设置 --fix 参数**

打开 package.json 文件，修改 script 属性如下：

"scripts": {

"test": "react-scripts test --env=jsdom",

"lint": "eslint src --fix",

"lint:create": "eslint --init"

}

说明：这里给 "lint": "eslint src --fix", 加上 --fix 参数，是 Eslint 提供的自动修复基础错误的功能。

此时运行 npm run lint 会看到少了两条报错信息，并不是说编码规范变了，而是 Eslint 自动修复了基础错误，打开 index.js 文件，可看到字符串自动变成了双引号，并且代码末尾也加上了分号。可惜的是 --fix 只能修复基础的不影响代码逻辑的错误，像 no-unused-vars 这种错误只能手动修改。

**配置文件讲解**

按照上述操作，会生成默认 .eslintrc.js 配置文件，内容如下：

// .eslintrc.js

module.exports = {

"env": {

"browser": true,

"commonjs": true,

"es6": true

},

"extends": "eslint:recommended",

"parserOptions": {

"ecmaFeatures": {

"experimentalObjectRestSpread": true,

"jsx": true

},

"sourceType": "module"

},

"plugins": [

"react"

],

"rules": {

"indent": [

"error",

"tab"

],

"linebreak-style": [

"error",

"windows"

],

"quotes": [

"error",

"double"

],

"semi": [

"error",

"always"

]

}

}

该文件导出一个对象，对象包含属性 env、extends、parserOptions、plugins、rules 五个属性，作为刚学习 Eslint 的新手，我们总是想要竭尽所能的详细了解每一个属性是什么，干嘛用的，以获取小小的安全感。

**env、parserOptions、plugins**

这三个放在一起将是因为你只需要知道它们是干嘛的：我的程序里要用到 ES6 、React 、JSX 语法，这几个属性就是让 Eslint 能够检验到这些语法的。其余的你不需要知道更多的哪怕一丢丢的东东了。

作者在学习之初在这块浪费了很多时间，官方文档看了很多遍，大多不能理解什么意思，后来想它既然提供这么一个自动生成配置文件的工具，并且是命令行交互的方式，我只需要告诉它我要用 ES6 、React 、JSX 语法，它会自动进行相关配置满足我的要求即可。

**extends**

前面一直说检验代码遵循编码规范，那到底是什么规范呢。

值为 "eslint:recommended" 的 extends 属性启用一系列核心规则，这些规则是经过前人验证的最佳实践（所谓最佳实践，就是大家伙都觉得应该遵循的编码规范），**想知道最佳实践具体有哪些编码规范**，可以在 [eslint规则表](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Frules%2F" \t "_blank) 中查看被标记为 √ 的规则项。

如果觉得官方提供的默认规则不好用，可以自定义规则配置文件，然后发布成 Npm 包，eslint-config-airbnb 就是别人自定义的编码规范，使用 npm 安装后，在我们自己的 .eslintrc.js 中进行配置 "extends": "airbnb"，eslint-config 这个前缀可以省略不写，这样我们就使用了 eslint-config-airbnb 中的规则，而不是官方的规则 "extends":"eslint:recommended" 了。关于如何创建自定义规则配置并共享可以参考： [如何自定义规则配置](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Fdeveloper-guide%2Fshareable-configs)

关于 "airbnb" 编码规范说两句，在接触到大多数开源项目中，大多数的作者都会使用 "airbnb" 编码规范而不是 官方的 "extends": "eslint:recommended" 编码规范。

如果我们觉得 eslint-config-airbnb 规则配置中个别规则并不符合当前项目的要求，可以直接在 .eslintrc.js 配置 rules 属性，优先级高于共享规则 airbnb。

**rules**

配置文件也生成了，我们怎么知道我们的系统会遵循什么规则呢？？

在前面的列子中，使用 npm run lint 校验出了三处错误，假如我们的项目中字符串就是要使用单引号而不是双引号，代码结尾就是要不加分号才好看，变量就是定义了可能不会使用，我们可以通过设置 rules 来定义我们自己的编码规范，即规则。

ESLint 附带有大量的规则，修改规则应遵循如下要求：

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

有的规则没有属性，只需控制是开启还是关闭，像这样："eqeqeq": "off"，有的规则有自己的属性，使用起来像这样："quotes": ["error", "double"]，具体有没有自带属性，可查看 [eslint规则表](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Frules%2F" \t "_blank)。

修改 .eslintrc.js 文件中的 rules 属性：

"rules": {

"indent": [

"error",

"tab"

],

"linebreak-style": [

"error",

"windows"

],

"quotes": [

"error",

"single" // 改成字符串必须由单引号括起来而不是双引号，'string'不报错，"string"报错

],

"semi": [

"error",

"never" // 改成代码结尾不再加分号，加了分号报错，不加分号不报错

],

"no-unused-vars": 0 // 0 相当于 off，表示关闭规则，相当于不再校验这条规则：变量定义了必须使用

}

此时再使用 npm run lint 进行代码校验，没有报错就说明校验通过，代码符合统一编码规范。

D:\code\test\20170811>npm run lint

> 20170811@0.1.0 lint D:\code\test\20170811

> eslint src

D:\code\test\20170811>

**可能存在的疑问**

刚接触 ESlint ，并不清楚有哪些规则怎么办，要去 [eslint规则表](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Frules%2F" \t "_blank) 一条条记忆吗？答案是 no。

个人认为 ESlint 的配置文件并不是一次性完成的，而是在项目过程中慢慢完善的。你可以放心大胆的先进行编码，然后使用 npm run lint 校验代码的编码规范，如果这时候报错，可以在报错信息中知道是哪一条编码规范报错了，你可能并不认识它们，此时去 [eslint规则表](https://links.jianshu.com/go?to=http%3A%2F%2Feslint.cn%2Fdocs%2Frules%2F" \t "_blank) 查询相应规则的使用方法，如：no-unused-vars，从而进一步确定项目中是否需要这条编码规范，如果需要，进行局部调整即可。

$ npm run lint

1:7 error 'lint' is assigned a value but never used no-unused-vars

1:14 error Strings must use doublequote quotes

1:22 error Missing semicolon semi

3 problems (3 errors, 0 warnings)

2 errors, 0 warnings potentially fixable with the `--fix` option.

**更新部分**

**全局变量配置。**

如使用 window 对象，默认情况下会报 no-undef 的错误，需要在 .eslintrc 中进行相应配置。

{

"rules": {

// ...

},

"globals": {

"window": true

}

}

**单行跳过 lint 校验**

在实际编码时，可能会出现以下的代码。

const apple = "apple";

const balana = "balana";

module.exports = {

fruit: balana ;

}

在最上面定义了两个变量，在底部的配置文件中只可能用到其中一个变量，另一个用不到的在 eslint 校验时就会报错 no-unused-vars 的错误，意思是变量定义了但是没有被用到。

其中一种解决方案是在 .eslintrc 文件中配置 rules no-unused-vars: 0，意思是项目中不检验变量定义未使用这条规则。强烈不建议这样做，这个规则十分有用，可以规避编写代码时遗漏的变量。

另一种解决方案就是使用行内注释跳过对 apple 和 balana 两个变量跳过 eslint 校验，只影响这两个变量，不影响外部。

const apple = "apple"; // eslint-disable-line

const balana = "balana"; // eslint-disable-line

module.exports = {

fruit: balana ;

}

此时使用 eslint 校验时就不会再报错了。

**常见规则含义解释**

* object-shorthand 设置该规则，表示[对象属性要简写](https://links.jianshu.com/go?to=http%3A%2F%2Fes6.ruanyifeng.com%2F%23docs%2Fobject%23%25E5%25B1%259E%25E6%2580%25A7%25E7%259A%2584%25E7%25AE%2580%25E6%25B4%2581%25E8%25A1%25A8%25E7%25A4%25BA%25E6%25B3%2595" \t "_blank)。

var foo = {x: x}; // 会报错

var bar = {a: function () {}}; // 会报错

var foo = {x}; // 不会报错

var bar = {a () {}}; // 不会报错

* prefer-arrow-callback 要求回调函数使用箭头函数

// 回调函数，函数的参数是个函数，这个参数函数就是回调函数

funciton bar () {} // 不是回调函数，不会报错

// setTimeout 的第一个参数就是回调函数，不用箭头函数会报错

setTimeout(() => {

// .......

}, 1000)

* no-param-reassign 禁止对函数的参数重新赋值

function bar ({ data = {} }) {

data.num = 12; // 会报错

}

/\*

虽然报错，又不想要将该校验关闭，如果代码中只有 data 这个属性有这种情形，

可以在 .eslintrc.js 中的 rules 属性中添加配置（可忽略 data 属性做此校验）：

'no-param-reassign': ['error', { 'props': true, 'ignorePropertyModificationsFor': ['data'] }],

'no-param-reassign': 0, // 关闭该属性校验

\*/

* no-trailing-spaces 禁止行尾空格
* no-shadow 禁止变量声明与外层作用域的变量同名

function sum (num) {

let num = 2; // 报错，因为 num 变量作为参数已经申明过了

}

**常用的几个规则**

"quotes": [1, "single"], # 单引号

"quote-props":[2, "as-needed"], # 双引号自动变单引号

"semi": [2, "never"], # 一行结尾不要写分号

"comma-dangle": [1,"always-multiline"] # 对象或数组多行写法时，最后一个值加逗号