**利用ESLint检查代码质量**

 发布于 2 年前  作者 [leizongmin](http://cnodejs.org/user/leizongmin)  39443 次浏览  来自 分享

原文发表于作者的个人博客：<http://morning.work/page/maintainable-nodejs/getting-started-with-eslint.html>

其实很早的时候就想尝试ESLint了，但是很多次都是玩了一下就觉得这东西巨复杂，**一执行检查就是满屏的error，简直是不堪入目，遂放弃**。直到某天终于下定决心深入看了文档，才发现其实挺简单的，只是当时没有看到合适入门教程而已。我相信很多人也有着跟我一样的经历，所以希望将自己的踩坑心得记录下来，让后来者更轻易地掌握ESLint的使用，因为它确实是个好东西。

JavaScript是一门神奇的动态语言，它在带给我们编程的灵活性的同时也悄悄埋下了一些地雷。除了基本的语法错误能在程序一启动的时候被检测到之外，很多隐含的错误都是在运行的时候才突然地蹦出来。除非你的程序有着100%的测试覆盖率，否则说不定哪天就会因为一个xxx is undefined而导致程序崩溃，而为了避免这样的错误可能你只需要在提交代码的时候用工具静态分析一下，仅此而已。

ESLint是一个插件化的javascript代码检测工具，它可以用于检查常见的JavaScript代码错误，也可以进行代码风格检查，这样我们就可以根据自己的喜好指定一套ESLint配置，然后应用到所编写的项目上，从而实现**辅助编码规范的执行，有效控制项目代码的质量**。

## 手把手入门

在开始使用ESLint之前，我们需要通过NPM来安装它：

$ npm install -g eslint

我从Gist上找到了自己几年前写的一个小函数，将其保存为文件merge.js：

function merge () {

var ret = {};

for (var i in arguments) {

var m = arguments[i];

for (var j in m) ret[j] = m[j];

}

return ret;

}

console.log(merge({a: 123}, {b: 456}));

然后执行node merge.js确保它是可以正确运行的（输出结果为{ a: 123, b: 456 }）。

接着我们执行以下命令来使用ESLint检查：

$ eslint merge.js

可以看到，没有任何输出结果。这是因为我们没有指定任何的配置，除非这个文件是有语法错误，否则应该是不会有任何提示的。现在我们先使用内置的eslint:recommended配置，它包含了一系列核心规则，能报告一些常见的问题。

首先新建ESLint配置文件.eslintrc.js：

module.exports = {

extends: 'eslint:recommended',

};

重新执行eslint merge.js可以看到输出了2个错误：

/example/merge.js

10:1 error Unexpected console statement no-console

10:1 error 'console' is not defined no-undef

✖ 2 problem (2 error, 0 warnings)

这两条提示信息还是足够我们搞清楚是怎么回事的：

* **Unexpected console statement no-console** - 不能使用console
* **‘console’ is not defined no-undef** - console变量未定义，不能使用未定义的变量

针对第1条提示，我们可以禁用no-console规则。将配置文件.eslintrc.js改为这样：

module.exports = {

extends: 'eslint:recommended',

rules: {

'no-console': 'off',

},

};

说明：配置规则写在rules对象里面，key表示规则名称，value表示规则的配置，具体说明见下文。

重新执行检查还是提示no-undef：

/example/merge.js

10:1 error 'console' is not defined no-undef

✖ 1 problem (1 error, 0 warnings)

这是因为JavaScript有很多种运行环境，比如常见的有浏览器和Node.js，另外还有很多软件系统使用JavaScript作为其脚本引擎，比如PostgreSQL就支持使用JavaScript来编写存储引擎，而这些运行环境可能并不存在console这个对象。另外在浏览器环境下会有window对象，而Node.js下没有；在Node.js下会有process对象，而浏览器环境下没有。

所以在配置文件中我们还需要指定程序的目标环境：

module.exports = {

extends: 'eslint:recommended',

env: {

node: true,

},

rules: {

'no-console': 'off',

},

};

再重新执行检查时，已经没有任何提示输出了，说明merge.js已经完全通过了检查。

### 配置文件

ESLint还可以在项目的package.json文件中指定配置，直接将上文中的module.exports的值写到eslintConfig里面即可：

{

"name": "my-package",

"version": "0.0.1",

"eslintConfig": {

"extends": "eslint:recommended",

"env": {

"node": true

},

"rules": {

"no-console": "off"

}

}

}

另外还可以在执行eslint命令时通过命令行参数来指定，详细文档可以参考这里：[Configuring ESLint - 配置](http://eslint.cn/docs/user-guide/configuring)

### 规则

每条规则有3个等级：off、warn和error。off表示禁用这条规则，warn表示仅给出警告，并不会导致检查不通过，而error则会导致检查不通过。

有些规则还带有可选的参数，比如comma-dangle可以写成[ "error", "always-multiline" ]；no-multi-spaces可以写成[ "error", { exceptions: { "ImportDeclaration": true }}]。

规则的详细说明文档可以参考这里：[Rules - 规则](http://eslint.cn/docs/rules/)

## 使用共享的配置文件

上文我们以eslint:recommended为基础配置，然后在此之上修改no-console这条规则。而在大多数时候，我们可能会根据自己个人或团队的习惯，定制更多的规则，比如限定缩进是2个空格和使用单引号的字符串等。而如果每一个项目都要这样写到.eslintrc.js文件上，管理起来会比较麻烦。

我们可以将定义好规则的.eslintrc.js文件存储到一个公共的位置，比如public-eslintrc.js：

module.exports = {

extends: 'eslint:recommended',

env: {

node: true,

},

rules: {

'no-console': 'off',

'indent': [ 'error', 2 ],

'quotes': [ 'error', 'single' ],

},

};

然后将原来的.eslintrc.js文件改成这样：

module.exports = {

extends: './public-eslintrc.js',

};

为了验证这样的修改是否生效，将merge.js中的var ret = {};这一行前面多加一个空格，再执行ESLint检查：

/example/merge.js

2:4 error Expected indentation of 2 space characters but found 3 indent

✖ 1 problem (1 error, 0 warnings)

这时候提示的是缩进只能为2个空格，而文件的第2行却发现了3个空格，说明公共配置文件public-eslintrc.js已经生效了。

我们还可以使用已经发布到NPM上的ESLint配置，这些配置的模块名一般以eslint-config-为前缀，比如我在学习ESLint时自己编写的一个配置名为eslint-config-lei。要使用这个配置，先执行以下命令安装它：

$ npm install -g eslint-config-lei

**注意：用于我们的eslint命令是全局安装的，所有用到的eslint-config-\*模块也必须全局安装，否则将无法正确载入。这是一个已知的Bug，参考这里：**[**Error: Cannot read config package for shareable config using global eslint #4822**](https://github.com/eslint/eslint/issues/4822#issuecomment-167600953)

然后将.eslintrc.js文件改成这样：

module.exports = {

extends: 'lei',

};

再执行ESLint检查，可以看到输出如下的提示：

/example/merge.js

1:15 warning Unexpected space before function parentheses space-before-function-paren

2:3 error Unexpected var, use let or const instead no-var

3:8 error Unexpected var, use let or const instead no-var

4:5 error Unexpected var, use let or const instead no-var

5:10 error Unexpected var, use let or const instead no-var

10:19 warning A space is required after '{' object-curly-spacing

10:26 warning A space is required before '}' object-curly-spacing

10:29 warning A space is required after '{' object-curly-spacing

10:36 warning A space is required before '}' object-curly-spacing

✖ 9 problems (4 errors, 5 warnings)

ESLint配置文件中的extends还可以用来指定各种来源的配置引用，具体说明可以参考以下链接：

* [Using a shareable configuration package - 使用共享的模块](http://eslint.cn/docs/user-guide/configuring#using-a-shareable-configuration-package)
* [Using the configuration from a plugin - 使用插件](http://eslint.cn/docs/user-guide/configuring#using-the-configuration-from-a-plugin)
* [Using a configuration file - 使用配置文件](http://eslint.cn/docs/user-guide/configuring#using-a-configuration-file)
* [Using “eslint:all” - 使用"eslint:all"](http://eslint.cn/docs/user-guide/configuring#using-eslintall)

## 代码格式化

在[ESLint规则列表](http://eslint.cn/docs/rules/)页面，我们发现有些规则的旁边会带有一个**橙色扳手图标**，表示在执行eslint命令时指定--fix参数可以**自动修复**该问题。

接着上文使用eslint-config-lei配置的检查，我们尝试在执行检查时添加--fix参数：

$ eslint merge.js --fix

执行完毕，没有发现任何提示。再打开merge.js文件发现已经变成了这样：

function merge() {

const ret = {};

for (const i in arguments) {

const m = arguments[i];

for (const j in m) ret[j] = m[j];

}

return ret;

}

console.log(merge({ a: 123 }, { b: 456 }));

主要的变化有以下三部分：

* 声明函数时，函数名与参数列表的空格不见了：merge ()修改为merge()
* var声明的变量变成了const声明：var ret = {}修改为const ret = {}
* 对象的内容与花括号之间增加了空格：{a: 123}修改为{ a: 123 }

我们可以利用这个特性来自动格式化项目代码，这样就可以保证代码书写风格的统一。

## 发布自己的配置

前文关于「共享的配置文件」一小节已经提到，可以在extends中指定一个文件名，或者一个eslint-config-开头的模块名。为了便于共享，一般推荐将其发布成一个NPM模块。

其原理就是在载入模块时输出原来.eslintrc.js的数据。比如我们可以创建一个模块eslint-config-my用于测试。

新建文件eslint-config-my/index.js：

module.exports = {

extends: 'eslint:recommended',

env: {

node: true,

es6: true,

},

rules: {

'no-console': 'off',

'indent': [ 'error', 2 ],

'quotes': [ 'error', 'single' ],

},

};

再新建文件eslint-config-my/package.json：

{

"name": "eslint-config-my",

"version": "0.0.1",

"main": "index.js"

}

为了能让eslint正确载入这个模块，我们需要执行npm link将这个模块链接到本地全局位置：

$ npm link eslint-config-my

然后将文件.eslintrc.js改成这样：

module.exports = {

extends: 'my',

};

说明：在extends中，eslint-config-my可简写为my。

在执行eslint merge.js检查，可看到没有任何错误提示信息，说明eslint已经成功载入了eslint-config-my的配置。如果我们使用npm publish将其发布到NPM上，那么其他人通过npm install eslint-config-my即可使用我们共享的这个配置。

另外可以参考我自己写的一个ESLint配置模块：[eslint-config-lei](https://github.com/leizongmin/eslint-config-lei)

关于共享ESLint配置的详细文档可参考：[Shareable Configs - 可共享的配置](http://eslint.cn/docs/developer-guide/shareable-configs)

## 例外情况

尽管我们在编码时怀着**严格遵守规则**的美好愿景，而**凡事总有例外**。定立ESLint规则的初衷是为了避免自己犯错，但是我们也要避免不顾实际情况而将其搞得太过于形式化，本末倒置。

ESLint提供了多种临时禁用规则的方式，比如我们可以通过一条eslint-disable-next-line备注来使得下一行可以跳过检查：

// eslint-disable-next-line

var a = 123;

var b = 456;

在上面的示例代码中，var a = 123不会受到检查，而var b = 456则右恢复检查，在上文我们使用的eslint-config-lei的配置规则下，由于不允许使用var声明变量，则var b这一行会报告一个error。

我们还可以通过成对的eslint-enable和eslint-disable备注来禁用对某一段代码的检查，但是稍不小心少写了一个eslint-disable就可能会导致后面所有文件的检查都被禁用，所以我并不推荐使用。

详细使用方法可以参考文档：[Disabling Rules with Inline Comments - 使用行内注释禁用规则](http://eslint.cn/docs/user-guide/configuring#disabling-rules-with-inline-comments)

## 总结

刚开始接触ESLint时觉得太难，是因为过太过于**迷信权威**。比如Airbnb公司的JavaScript风格，在GitHub上受到了很大的好评，其实我自己也非常认可这样的编码风格。但每个团队都会根据自己的的**实际情况**来**定制**不同的东西，我们并**不能随便照搬**过来。所以当使用eslint-config-airbnb这个配置进行ESLint检查时，满屏都是error和warning，从而觉得这东西根本没啥卵用。

另外我也犯了「大忌」：直接使用eslint-config-airbnb这种某个公司高度定制化的配置，而不是eslint:recommended这样保守的。而且是直接用来检查整个项目好几十个JS文件，可想而知那是怎样的画面（本文最后版本的merge.js文件使用airbnb的配置，总共12行的代码就提示了8个问题：✖ 8 problems (7 errors, 1 warning)）。

本文的目的是让读者以一个比较低的姿态开始接触ESLint，先学会简单地配置规则，如果要更深入地定制自己的规则，建议阅读「相关链接」中的ESLint文档。

## 相关链接

* [ESLint使用入门](https://csspod.com/getting-started-with-eslint/)
* [ESLinit中文版文档](http://eslint.cn/)