

Eslint

意义?



by eric wen

eslint规范的是什么

ESLint 在减少低级错误方面非常有用，它可以通过静态代码分析捕获许多开发者容易忽略的问题。以下是一些具体的例子：

1. 捕获未使用的变量

问题：

```
function calculateSum(a, b) {
  const result = a + b;
  return a + b; // 这里重复计算了，`result` 定义后未被使用
}
```

ESLint 提示：

- 规则：no-unused-vars
- 错误信息：`result` is defined but never used (no-unused-vars)

修复后的代码：

```
function calculateSum(a, b) {
  return a + b;
}
```

收益：

- 避免定义无用的变量，减少代码冗余。

2. 防止意外的全局变量声明

问题：

```
function printMessage() {
  message = "Hello, World!"; // 忘记使用 `let` 或 `const`
  console.log(message);
}
```

ESLint 提示：

- 规则：no-undef
- 错误信息：`message` is not defined (no-undef)

修复后的代码：

```
function printMessage() {
  const message = "Hello, World!";
  console.log(message);
}
```

收益：

- 避免污染全局作用域，防止潜在的命名冲突和难以调试的错误。

3. 捕获多余的代码

问题：

```
function checkValue(value) {
  if (value === true) {
    return true;
  } else {
    return true; // `else` 分支是多余的
  }
}
```

ESLint 提示：

- 规则：no-unreachable
- 错误信息：Unnecessary `else` after `return` (no-else-return)

修复后的代码：

```
function checkValue(value) {
  if (value === true) {
    return true;
  }
  return true;
}
```

收益：

- 简化代码逻辑，提高可读性。

4. 防止意外的赋值操作

问题：

```
function isEqual(a, b) {
  if (a = b) { // 本意是 `===`，但不小心写成了 `=`
    return true;
  }
  return false;
}
```

ESLint 提示：

- 规则：no-cond-assign
- 错误信息：Expected a conditional expression and instead saw an assignment (no-cond-assign)

修复后的代码：

```
function isEqual(a, b) {
  if (a === b) {
    return true;
  }
  return false;
}
```

收益：

- 避免因意外赋值导致的逻辑错误。

5. 捕获未处理的异常

问题：

```
try {
  throw new Error("Something went wrong!");
} catch (error) {
  // 忘记处理错误
}
```

ESLint 提示：

- 规则：no-unused-vars（针对未使用的 error 变量）
- 错误信息：`error` is defined but never used (no-unused-vars)

修复后的代码：

```
try {
  throw new Error("Something went wrong!");
} catch (error) {
  console.error(error.message); // 处理错误
}
```

收益：

- 确保异常被正确处理，避免隐藏的错误。

6. 防止意外的类型转换

问题：

```
if ("5" == 5) { // 使用 `==` 导致隐式类型转换
  console.log("They are equal!");
}
```

ESLint 提示：

- 规则：eqeqeq
- 错误信息：Expected `===` and instead saw `==` (eqeqeq)

修复后的代码：

```
if ("5" === 5) { // 明确比较类型和值
  console.log("They are equal!");
}
```

收益：

- 避免隐式类型转换带来的不可预测行为。

7. 防止无限循环

问题：

```
for (let i = 0; i < 10; i--) { // 本意是 `i++`，但写错了
  console.log(i);
}
```

ESLint 提示：

- 规则：`no-unmodified-loop-condition`

理解eslint的意义

好处：

- 1、风格一致
- 2、避免低级错误，养成良好习惯
- 3、规范代码，方便维护

坏处：

- 1、影响紧急项目效率
- 2、矫枉过正
- 3、对业务逻辑纠错意义甚微

项目eslint配置

1、新建.env文件，写入DISABLE_ESLINT_PLUGIN=false

2、打开编辑器对eslint的提示：

- 新建.eslintrc.json文件，配置规范

```
"extends": [  
  "eslint:recommended",  
  "plugin:react/recommended" // 如果是React项目，请包含此行  
]
```

- 安装eslint插件

eslint的常见规范

```
// 缩进设置为2个空格

"indent": ["error", 2],

// 强制使用单引号

"quotes": ["error", "single"],

// 强制在语句末尾使用分号

"semi": ["error", "always"],

// 禁止不必要的分号

"no-extra-semi": "error",

// 要求箭头函数的箭头两边有空格 (fixable)

"arrow-spacing": ["error", { "before": true, "after": true }],

// 不允许定义未使用的变量

"no-unused-vars": ["warn", { "vars": "all", "args": "after-used" }],

// 不允许在变量定义之前使用它们

"no-use-before-define": ["error", { "functions": true, "classes": true, "variables": true }],

// 强制所有控制语句使用大括号

"curly": ["error", "all"],


// 不允许使用var，推荐使用let和const

"no-var": "error",

// 如果变量一旦被赋值后不再改变，则要求使用const声明该变量


"prefer-const": ["error", { "destructuring": "any", "ignoreReadBeforeAssign": true }]
```

更多：




Rules Reference

A pluggable and configurable linter tool for identifying and reporting on patterns in JavaScript. Maintain your code quality with ease.



Rules Reference – ESLint – Pluggable JavaScript Linter

A pluggable and configurable linter tool for identifying and reporting on patterns in JavaScript. Maintain your code quality with ease.



常见规范合集

常见的规范合集

- 1、eslint:recommended，这是 ESLint 自带的一个推荐规则集，专注于捕获潜在的错误和最佳实践
- 2、Airbnb，对于 React 应用开发来说是一个很好的选择。

```
"extends": [  
  "airbnb"  
],
```

- 3、StandardJS，适合那些希望快速开始而不愿花费时间在配置上的项目。

自定义规则

```
"rules": {  
  // 缩进设置为2个空格  
  "indent": ["warn", 2],  
  
  // 强制使用单引号  
  "quotes": ["warn", "single"],  
  
  // 强制在语句末尾使用分号  
  "semi": ["warn", "always"],  
  
  // 禁止不必要的分号  
  "no-extra-semi": "warn",  
  
  // 要求或禁止对象字面量中冒号周围的空格 (fixable)  
  "key-spacing": ["warn", { "beforeColon": false, "afterColon": true }],  
  
  // 要求箭头函数的箭头两边有空格 (fixable)  
  "arrow-spacing": ["warn", { "before": true, "after": true }],  
  
  // 不允许定义未使用的变量  
  "no-unused-vars": ["warn", { "vars": "all", "args": "after-used" }],  
  
  // 不允许在变量定义之前使用它们  
  "no-use-before-define": ["warn", { "functions": true, "classes": true, "variables": true }],  
  
  // 强制所有控制语句使用大括号  
  "curly": ["warn", "all"],  
  
  // 在对象字面量的花括号内强制使用空格 (fixable)  
  "object-curly-spacing": ["warn", "always"],  
  
  // 在数组字面量的方括号内强制不使用空格 (fixable)  
  "array-bracket-spacing": ["warn", "never"],  
  
  // 不允许使用var，推荐使用let和const  
  "no-var": "warn",  
  
  // 如果变量一旦被赋值后不再改变，则要求使用const声明该变量  
  "prefer-const": ["warn", { "destructuring": "any", "ignoreReadBeforeAssign": true }],  
  "react/prop-types": 0,  
  "no-debugger": "off"  
},
```