为什么要用 eslint

JavaScript 是一个动态的弱类型语言，在开发中比较容易出错。由于没有编译程序，为了定位 JavaScript 代码错误通常需要在执行过程中不断调试，ESLint 可以在编码的过程中静态的分析发现问题而不是在执行的过程中。

ESLint 相对于其它 linter 代码查检工具有更好的扩展性，基于插件化机制可以很好的定制和添加规则，对 JSX 的支持也非常友好。

ESLint 是一个插件化的 javascript 代码检测工具，既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，这样可以根据团队协作的需要制定一套 ESLint 配置，应用到实际项目中，从而实现辅助编码规范的执行，有效控制项目代码的质量，提前发现代码书写问题，保证团队代码风格统一性及可读性。

代码规范制定

业界使用比较多的有 airbnb 和 google 代码规范，这两个规范对代码书写风格和错误检查都非常严格，直接拿过来当作团队内代码规范不一定完全适合，太过严格的检查对编码过程也有一定的束缚和限制，在此我们的选择是在 eslint:recommended 的基础上适当添加了一些错误查检的代码书写规范。

eslint-config-info

ESLint 对所有的规则进行了分类，包括：

- JavaScript 代码中可能的错误或逻辑错误相关
- 最佳实践的，帮助你避免一些问题
- 变量声明有关
- 关于Node.js 或 在浏览器中使用CommonJS
- 关于风格指南的，而且是非常主观的
- 与 ES6 有关的

详细规则参见：https://eslint.org/docs/rules/

✔	🔧	no-extra-boolean-cast	disallow unnecessary boolean casts
	🔧	no-extra-parens	disallow unnecessary parentheses
✔	🔧	no-extra-semi	disallow unnecessary semicolons
✔		no-func-assign	disallow reassigning `function` declarations
✔		no-inner-declarations	disallow variable or `function` declarations in nested blocks
✔		no-invalid-regexp	disallow invalid regular expression strings in `RegExp` constructors
✔		no-irregular-whitespace	disallow irregular whitespace outside of strings and comments
✔		no-obj-calls	disallow calling global object properties as functions
		no-prototype-builtins	disallow calling some `Object.prototype` methods directly on objects
✔	🔧	no-regex-spaces	disallow multiple spaces in regular expressions
✔		no-sparse-arrays	disallow sparse arrays

其中 打勾 的是 `eslint:recommended` 开启的规则，有 小扳手 的是可以通过添加命令行参数 `--fix` 参数自动修复的。

在 `eslint:recommended` 的基础上，添加了环境依赖及 JavaScript 语言选项，启用对 `ECMAScript 2017` 版本和 `JSX` 的支持。

```
{
  ...
  {
    env: {
    browser: true,      // 浏览器全局变量
    node: true,         // node 环境全局变量
    commonjs: true,     // commonjs 规范全局变量
    es6: true           // es6 新增全局变更或类型
  },

  parserOptions: {              // 解析器相关配置
    ecmaVersion: 2018,          // 按照 ECMAScript2018 版本解析语法
    sourceType: 'module',       // 支持 ECMAScript 模块
    ecmaFeatures: {
      jsx: true,                // 支持 react JSX 语法检测
      modules: true
    }
  },
  ...
}
```

在 `eslint:recommended` 上推荐规则上，基于一些常见的错误及代码风格统一性做了相应的补充和覆盖，如下：

```
{
  ...
  rules: {
```

```
      // 强制 “for” 循环中更新子句的计数器朝着正确的方向移动
      'for-direction': 'error',
      // 禁止使用 console
      'no-console': 'off',
      // 强制数组方法的回调函数中有 return 语句
      'array-callback-return': 'error',
      // 强制把变量的使用限制在其定义的作用域范围内
      'block-scoped-var': 'error',
      // 指定程序中允许的最大环路复杂度
      'complexity': ['error', { max: 10 }],
      // 强制所有控制语句使用一致的括号风格
      'curly': [
        'error',
        'multi-line',
        'consistent'
      ],
      // 强制在点号之前和之后一致的换行
      'dot-location': [
        'error',
        'property'
      ],
      // 要求使用 === 和 !==, null 除外
      'eqeqeq': [
        'error',
        'always',
        {
          null: 'ignore'
        }
      ],
      // 要求 for-in 循环中有一个 if 语句
      'guard-for-in': 'error',
      ...
    }
    ...
  }
```

更多详细详细补充规则，http://eslint.cn/docs/user-guide/configuring

如何使用

安装

```
npm install --save-dev eslint-config-info
```

配置文件

在项目根目录中新建文件 `.eslintrc.json`, 支持 `json`, `js`, `yaml` 格式配置，选其一即可。

普通 es6 项目

```
//eslintrc.json
{
  "root": true,
  "extends": "info",
  "rules": {
    //这里可根据项目实际需求，添加和覆盖相应的规则
    // ...
  }
}
```

react 项目

配置 `"extends": "info/react"` 即可，无则中默认添加了 `eslint-plugin-react` 检测插件，配置如下：

```
//eslintrc.json
{
  "root": true,
  "extends": "info/react",
  "rules": {
    //这里可根据项目实际需求，添加和覆盖相应的规则
    // ...
  }
}
```

`eslint-plugin-react` 插件默认开启了 `plugin:react/recommended` 推荐配置规则，其中包括 `react` 和 `JSX` 的语法检测。

更多关于 `eslint-plugin-react` 的 `plugin:react/recommended` https://github.com/yannickcr/eslint-plugin-react

ts 项目

如果是 `ts` 项目，推荐使用 `tslint` 工具配合使用，这里也针对 `ts` 项目单独抽离了一份公共的 `tslint` 配置文件。

关于 `tslint` 使用，请参考 https://github.com/yannickcr/eslint-plugin-react

如果是 `ts` 项目也想使用 `eslint` 工具，可以通过更改 `eslint` 的编译器为 `typescript-eslint-parser` 和添加 `tslint-plugin-typescript` 插件，参考配置如下：

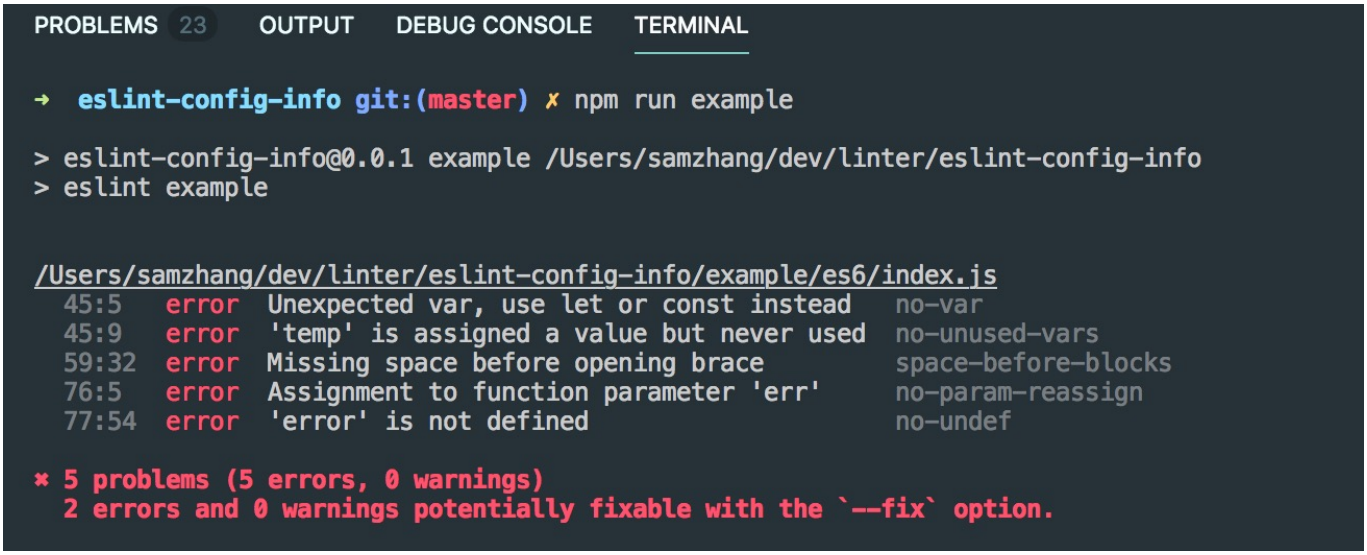
```
{
  extends: 'info',
  parser: 'typescript-eslint-parser',
  plugins: ['typescript'],
  rules: {
    // Require that interface names be prefixed with I
```

```
    'typescript/interface-name-prefix': ['error', 'always']
  }
}
```

执行脚本

```
// 在 package.json script 中添加
"lint": "node_modules/.bin/eslint src/**.js"

// 如果期望自动修复
"lint": "node_modules/.bin/eslint src/**.js --fix"
```

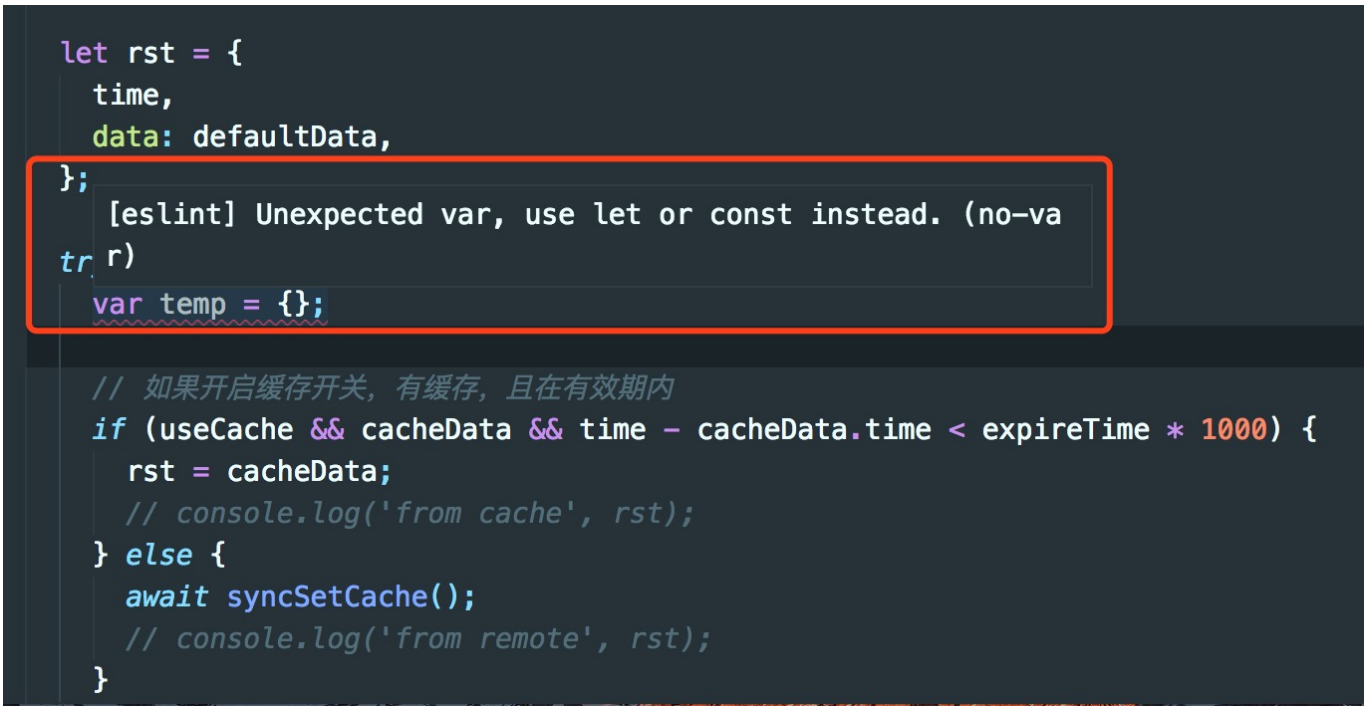


临时屏蔽规则

```
//eslint-disable-next-line no-var
var eslint = 'eslint';
```

配合编辑器高亮提示

vscode, webstorm 都有相应的插件，当项目配置了相应的 .tslintrc 配置文件时，会即可高亮提示，非常方便。



几种常见错误检测举例

```
// no-cond-assign 禁止条件表达式中出现赋值操作符
// Check the user's job title
if (user.jobTitle = "manager") {
  // user.jobTitle is now incorrect
}

// no-constant-condition 禁止在条件中使用常量表达式
if (false) {
  doSomethingUnfinished();
}

// no-unmodified-loop-condition 禁用一成不变的循环条件
while (node) {
  doSomething(node);
}
node = other;

for (var j = 0; j < items.length; ++i) {
  doSomething(items[j]);
}

while (node !== root) {
  doSomething(node);
}

// radix 强制在parseInt()使用基数参数
var num = parseInt("071");   // 57

// no-return-await 禁止没必要的 return await
```

```
async function foo() {
  return await bar();
}

// eqeqeq 要求使用 === 和 !==
if (x == 42) { }
if ("" == text) { }
if (obj.getStuff() != undefined) { }
```