手把手实现一个ESLint的插件

通过记录一下手写实现一个eslint的插件和规则，可以让大家了解eslint的运行原理

> 🏆 目标是实现一个eslint的插件，插件可以禁止 `setTimeout` 的第二个参数是数字。避免走查的时候会被认为是魔法数字。

初始化插件项目

ESLint 官方为了方便开发者，提供了使用Yeoman脚手架的模板(generator-eslint🔗)

Yeoman可以理解为是个通用的脚手架工具，可以生成包含指定框架结构的工程化目录。

安装的过程.....

> 生成项目模版

```
# 生成ESLint插件的项目结构
yo eslint:plugin

# 初始的配置
? What is your name? OBKoro1
? What is the plugin ID? korolint   // 这个插件的ID是什么
? Type a short description of this plugin: XX公司的定制ESLint rule // 输入这个插件的
? Does this plugin contain custom ESLint rules? Yes // 这个插件包含自定义ESLint规则吗
? Does this plugin contain one or more processors? No // 这个插件包含一个或多个处理器
// 处理器用于处理js以外的文件 比如.vue文件
```

```
   create package.json
   create lib/index.js
   create README.md
```

这样我们就得到了一个文件夹结构和一些文件，然后我们需要创建规则。

> 创建具体规则的文件

```
# 生成规则模版文件
yo eslint:rule

# 相关配置
? What is your name? OBKoro1
? Where will this rule be published? (Use arrow keys) // 这个规则将在哪里发布?
❯ ESLint Core  // 官方核心规则 (目前有200多个规则)
  ESLint Plugin  // 选择ESLint插件
? What is the rule ID? settimeout-no-number  // 规则的ID
? Type a short description of this rule: setTimeout 第二个参数禁止是数字  // 输入该规
? Type a short example of the code that will fail:  占位  // 输入一个失败例子的代码
   create docs/rules/settimeout-no-number.md
   create lib/rules/settimeout-no-number.js
   create tests/lib/rules/settimeout-no-number.js
```

创建完成后，我们的项目大概会是这样的：

```
.
├─── README.md
├─── docs // 使用文档
│    └─── rules // 所有规则的文档
│         └─── settimeout-no-number.md // 具体规则文档
├─── lib // eslint 规则开发
│    ├─── index.js 引入+导出rules文件夹的规则
│    └─── rules // 此目录下可以构建多个规则
│         └─── settimeout-no-number.js // 规则细节
├─── package.json
└─── tests // 单元测试
     └─── lib
          └─── rules
               └─── settimeout-no-number.js // 测试该规则的文件
```

安装好对应的依赖后，我们就完成了开发一个ESLint插件的准备工作。在我们开始开发自己的规则之前，我们还需要了解一下AST和ESLint原理的相关知识，如果已经会了的同学可以跳过。

AST抽象语法树

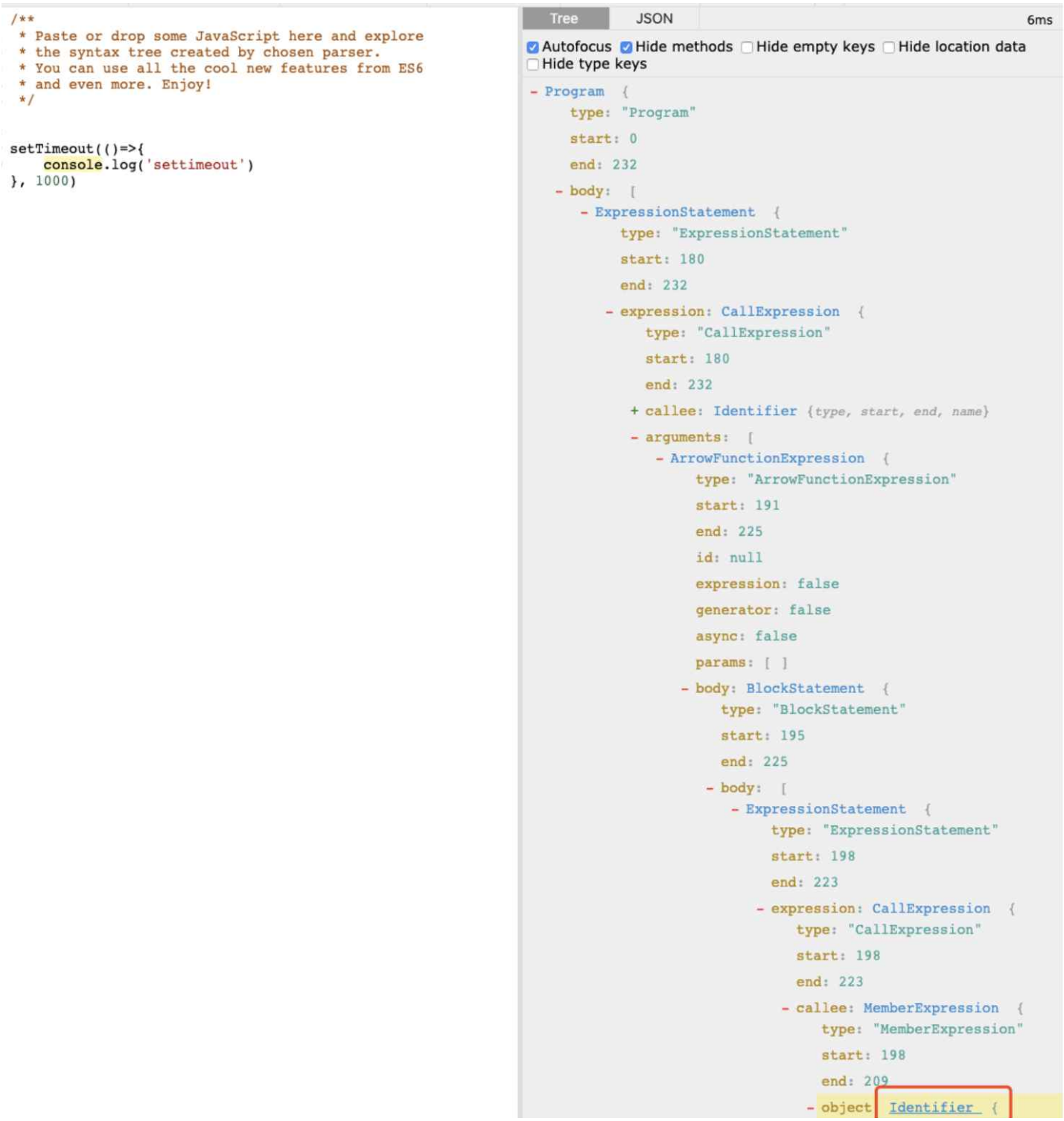
> 💡 AST是: `Abstract Syntax Tree` 的简称，中文叫做：抽象语法树。
>
> AST的作用是将代码抽象成树状的数据结构，方便后续的分析操作代码。

> 可以通过 这个工具网站来查看代码被解析成AST的结构

具体AST的知识就不展开描述了，有兴趣的可以去看我的另一篇文章： AST能做什么?

这里我们为了写一个插件需要了解的是AST的选择器（selectors）。

选择器的作用是通过选择器来选中特定的代码片段，然后对代码进行静态的分析和操作。



ESLint的运行原理

在开发规则前，我们还需要知道ESLint是怎样运行的，才能知道我们的插件怎么实现。ESLint的工作流程大概是酱紫的：

1. 将代码解析成AST。ESLint会通过它本身的Espree解析器将js的代码解析成AST。
2. 深度遍历AST。拿到解析完的AST，ESLint会**"从上至下" 再 "从下至上"**遍历每个节点两次，然后放到nodeQueue 队列。
3. 触发规则的回调。在遍历AST的过程中，每一条已经生效的规则会就会给节点的选择器添加事件绑定，触发规则的回调处理。
4. 具体的规则逻辑处理。通过规则的检查完返回的linting problems来对代码进行提示和修复等逻辑。

创建规则

当我们了解了ESLint的运行原理后，我们就可以着手开始自己的规则了。

规则模版

我们打开之前初始化生成的 `lib/rules/settimeout-no-number.js` 文件，删除不必要的规则后，内容大概是酱紫的：

```
module.exports = {
  meta: {
    docs: {
      description: "setTimeout 第二个参数禁止是数字",},
      fixable: null,  // 修复函数
      },
```

```
7         // rule 核心
8         create: function(context) {
9         // 公共变量和函数应该在此定义
10        return {
11        // 返回事件钩子
12        };}
13  };
```

create方法

> 监听选择器：在深度遍历的过程中，生效的每条规则都会对其中的某一个或多个选择器进行监听，每当匹配到选择器，监听该选择器的rule，都会触发对应的回调。

create方法会返回一个对象，对象的属性设为选择器，然后ESLint收集这些选择器，在遍历AST的时候就会监听这些选择器的回调。

```
1   // rule 核心
2  create: function(context) {
3      // 公共变量和函数应该在此定义
4      return {
5          // 返回事件钩子
6          Identifier: (node) => {
7              // node是选中的内容，是我们监听的部分，它的值参考AST
8          }
9      };
10 }
```

观察AST

创建一个ESLint的的规则首先需要观察AST，然后选中需要处理的代码的进行判断。

```
1  setTimeout(()=>{
2          console.log('settimeout')
3  }, 1000)
```

```
- expression: CallExpression  {
    type: "CallExpression"
    start: 180
    end: 232
  - callee: Identifier  {
      type: "Identifier"
      start: 180
      end: 190
      name: "setTimeout"
    }
  - arguments:  [
    + ArrowFunctionExpression {type, start, end, id, expression, ... +4}
    - Literal = $node {
        type: "Literal"
        start: 227
        end: 231
        value: 1000
        raw: "1000"
      }
    ]
  }
}
```

实现rule

```
// lib/rules/settimeout-no-number.js:
module.exports = {
    meta: {
        docs: {
            description: "setTimeout 第二个参数禁止是数字",
        },
        fixable: null,  // 修复函数
    },
    // rule 核心
    create: function (context) {
        // 公共变量和函数应该在此定义
        return {
            // 返回事件钩子
            'CallExpression': (node) => {
                if (node.callee.name !== 'setTimeout') return // 不是定时器即过滤
```

```
                const timeNode = node.arguments && node.arguments[1] // 获取第二个
                if (!timeNode) return // 没有第二个参数
                // 检测报错第二个参数是数字 报错
                if (timeNode.type === 'Literal' && typeof timeNode.value === 'num
                    context.report({
                        node,
                        message: 'setTimeout第二个参数禁止是数字'
                    })
                }
            }
        };
    }
};
```

context.report()：这个方法是用来通知ESLint这段代码是警告或错误的，用法如上。在这里查看 `context` 和 `context.report()` 的文档。

规则写完了，原理就是依据 `AST` 解析的结果，做针对性的检测，过滤出我们要选中的代码，然后对代码的值进行逻辑判断。

可能现在会有点懵逼，但是不要紧，我们来写一下测试用例，然后用 `debugger` 来看一下代码是怎么运行的。

测试用例

```
/**
 * @fileoverview setTimeout 第二个参数禁止是数字
 * tests/lib/rules/settimeout-no-number.js:
 */
"use strict";
var rule = require("../../../lib/rules/settimeout-no-number"), // 引入rule
    RuleTester = require("eslint").RuleTester;

var ruleTester = new RuleTester({
    parserOptions: {
        ecmaVersion: 7, // 默认支持语法为es5
    },
});
// 运行测试用例
ruleTester.run("settimeout-no-number", rule, {
    // 正确的测试用例
    valid: [
        {
            code: 'let someNumber = 1000; setTimeout(()=>{ console.log(11) },some
        },
```

```
        {
            code: 'setTimeout(()=>{ console.log(11) },someNumber)'
        }
    ],
    // 错误的测试用例
    invalid: [
        {
            code: 'setTimeout(()=>{ console.log(11) },1000)',
            errors: [{
                message: "setTimeout第二个参数禁止是数字", // 与rule抛出的错误保持一致
                type: "CallExpression" // rule监听的对应钩子
            }]
        }
    ]
});
```

这里推荐使用VSCode来调试node文件，可以观察rule是怎样运行的。

1. 在规则文件中打一些debugger
2. 点击开始，用调试的形式运行测试文件
3. 开始调试rule

集成到项目

当我们开发完规则测试通过后，可以通过npm包来发布，需要注意的是推荐使用ESLint官方的命名规范： `eslint-plugin-xxxx` 。

发布好插件后，我们就可以在项目中安装npm包来使用规则。

常规的引用方法：

```
// .eslintrc.js
module.exports = {
  plugins: [ 'xxxx' ],
  rules: {
    "xxxx/settimeout-no-number": "error"
 }
}
```

这样有个问题是当我们的规则比较多的时候，一条一条添加就太蠢了吧。我们可以继承插件的配置。

修改规则文件

```
'use strict';
```

```
var requireIndex = require('requireindex');
const output = {
  rules: requireIndex(__dirname + '/rules'), // 导出所有规则
  configs: {
    // 导出自定义规则 在项目中直接引用
    koroRule: {
      plugins: ['korolint'], // 引入插件
      rules: {
        // 开启规则
        'korolint/settimeout-no-number': 'error'
      }
    }
  }
};
module.exports = output;
```

使用extends来拓展插件的配置

```
// .eslintrc.js
module.exports = {
  extends: [ 'plugin:xxxx/xxxxRule' ] // 继承插件导出的配置
}
```

拓展

💡 遍历AST的方向？"从上至下" 再 "从下至上"两次？

💡 fix函数自动修复？