ESLintの独自ルール作成にチャレンジした話

株式会社NoSchool meijin

目次

独自ルールを作ろうと思ったきっかけ

作って適用する手順

独自ルールの作りかた

テストの書きかた

補足① TypeScript対応

補足② 運用手段の候補

余談 勉強会の主催

まとめ

自己紹介

ニックネームは「名人」

Twitter: **@Meijin_garden**

Webエンジニア6年目

株式会社NoSchool CTO

- オンライン家庭教師マナリンク -> https://manalink.jp

好きな分野はWebフロントエンド

- 最近は趣味でWYSIWYGエディタを作ってます

趣味は将棋（アマチュア二段くらい）

独自ルールを作ろうと思ったきっかけ

ESLintのルールって字句解析の知識とかないと作るの大変そうやけど、みんな知識がある上で作っているの？

業務上で、「今後はこのライブラリは使わないで欲しい」とか「この便利関数作ったからみんな使ってね」といったローカルルールを、レビューではなくLintで縛れたらいいな〜と思った

作って適用する手順

今回は以下の手順で独自ルールを動作させることができた。

独自ルールを記述したJavaScript(or TS)ファイルを作成して、`rules`ディレクトリに置く

npm scriptsで`"lint": "eslint --rulesdir rules"`のように、rulesdirを指定する

※`--rulesdir`オプションはDeprecatedなので、他の方法が有力（後述）

https://eslint.org/docs/user-guide/command-line-interface

独自ルールの作りかた

本家のドキュメント

「Working with Rules」

https://eslint.org/docs/developer-guide/working-with-rules

英語かつものすごく長く、（個人的には）理解に時間がかかるので、日本語の文献や巷のライブラリのコードを読みつつ進めるのが良いと思う。ある程度理解すると、公式ドキュメントが割と手にとるようにわかる（当社比）。

以後、これを分かっていれば公式ドキュメントの言っていることがなんとなくわかると思う知識をザッと書いていきます。

各ルールの構造

`meta` と `create` から構成される大きなオブジェクトをexportするJS（or TS）ファイルが実態

```
module.exports = {
    meta: {
        type: "suggestion",
        // ...
    },

    create(context) {
        return {
            VariableDeclaration(node) {
                // ...
            }
        };

    }
};
```

`meta`キーについて

その名の通りルールのメタ情報をオブジェクトで表現する。

ESLint v8から必要となるプロパティが追加されたみたいなので、昔の日本語記事など一部参考にならないものがあるかも。要注意。

https://eslint.org/docs/user-guide/migrating-to-8.0.0#rules-require-metahassuggestions-to-provide-suggestions

```
module.exports = {
    meta: {
        hasSuggestions: true
    },
    create(context) {
        // your rule
    }
};
```

`create`キーについて

createは `context` を受け取る関数で、ASTのノードの種類に対応してLintルールを実装する

- **プログラム中のこの種類のノードに対しては、このチェックをする**！という考えで実装する
- なのでLintに関係ないノードについては実装しなくてよい

ここで必要な知識は2つ！

- ASTのノードの種類ってなに？ ≒ ASTとはなにか
- どうやってLintルールを実装するか

ASTとはなにか

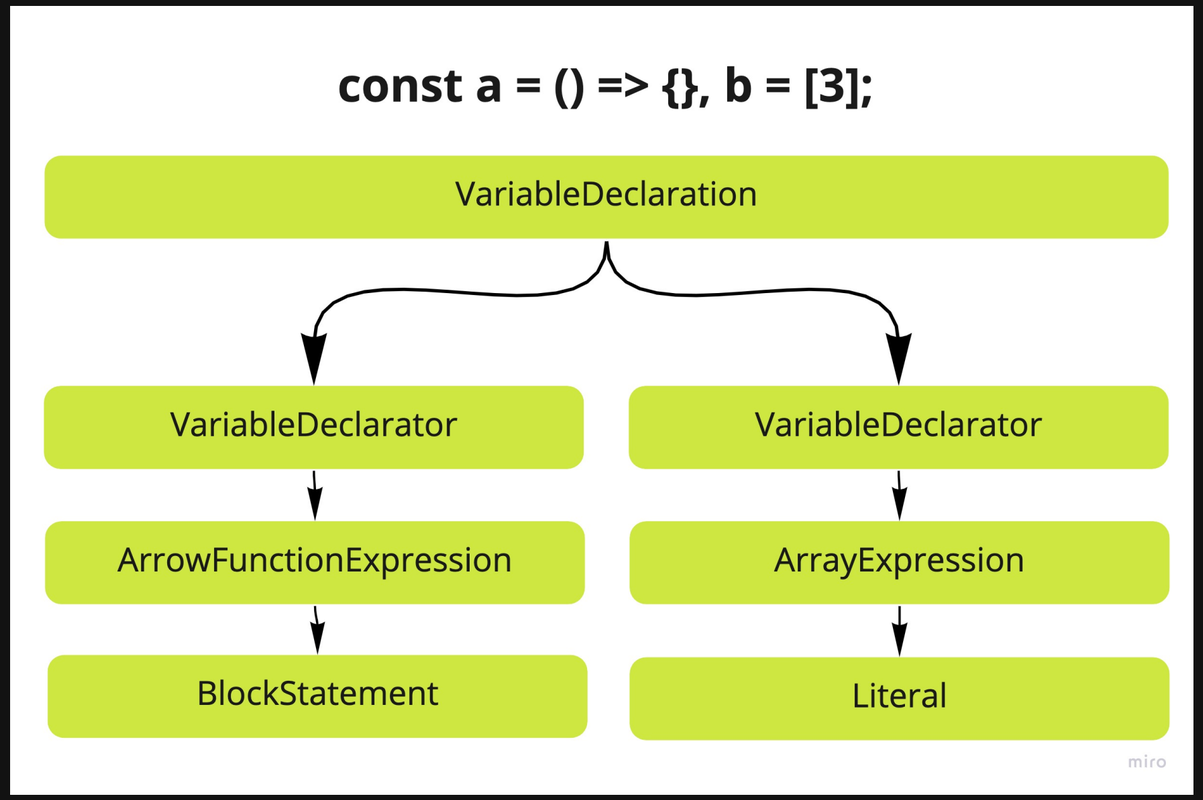
(広義には)文字列で表現されたプログラムから演算子や変数など、文法的に意味のある情報のみを取り出して、木構造で表現したもの

> プログラムは、【プログラム全体->複数のクラス->複数の関数->複数の変数宣言や代入など】といったように木構造で表現できる

プログラムを木構造で表現すると、プログラムから扱いやすくなるので、ESLintなどの「**プログラムを意味的に解釈して何らかの動作をするもの**」を実装しやすくなる

ASTを覗いてみる

https://astexplorer.net/ で簡単な式のAST表現を見てみると、文字列のコードが巨大なJSONとして表現されることがわかる。雑に図解すると以下の感じ



どうやってLintルールを実装するか

引数で渡される `context` に、エラーや警告をレポートする `report()` 関数や、nodeから変数名を取得できる `getDeclaredVariables()` 関数などが入っており、簡単なルールなら十分実装できる

```
// 変数名が _ から始まっていたらエラーになる独自ルールの例
create(context) {
    return {
        VariableDeclaration(node) {
            context.getDeclaredVariables(node).forEach(variable => {
                const name = variable.name;
                if (name.charAt(0) === "_") {
                    context.report({
                        node,
                        messageId: "unexpected",
                        data: { name }
                    });
                }
            });
        }
    };
}
```

`VariableDeclaration` が変数宣言ノードの種類名.

`context.getDeclaredVariables` で変数名を全取得.

`context.report` でLint結果をレポート.

リファレンス

`context` の仕様

https://eslint.org/docs/developer-guide/working-with-rules#the-context-object

`VariableDeclaration` などのASTのノードの種類名

覚える必要はなく、たとえば以下の文献を参照するとよさそう

`ESTree` https://github.com/estree/estree/blob/master/es5.md

`@types/eslint` https://github.com/DefinitelyTyped/DefinitelyTyped/blob/master/types/eslint/index.d.ts#L438

テストの書きかた

`RuleTester` を使ってテストを実装.

`valid / invalid` キーで正常系と異常系を書く.

`RuleTester` のコンストラクタでオプションも色々設定できるらしい.

```
const rule = require("../no-underscore-prefix"),
    RuleTester = require("eslint").RuleTester;
const ruleTester = new RuleTester();
ruleTester.run("no-underscore-prefix", rule, {
    valid: [
        "'hoge'",
        // ...
        { code: "const obj = { _name: 'hoge' };", env: { es6: true } },
    ],
    invalid: [
        {
            code: "var _hoge = 'hoge';",
            errors: [{ messageId: "unexpected", data: { name: "_hoge" }, type: "VariableDeclaration",
        },
        // ...
        { code: "let x,_y = 'hoge';", env: { es6: true }, errors: [{ messageId: "unexpected", data: {
        // ...
    ]
});
```

補足① TypeScript対応

`@types/eslint` を使う

以下の記事が詳しい（厳密にはESLint7時代の記事なので少しだけ注意が必要）

https://blog.sa2taka.com/post/custom-eslint-rule-with-typescript

ESLint自体はtsを読み込んでくれないので、コンパイルフェーズを自前で組むことに注意

補足② 運用手段

冒頭で示した `--rulesdir` オプションはDeprecated

あくまで外部プラグインとして、ローカルフォルダを読み込めるプラグインとしてのアプローチがいくつかある

eslint-plugin-rulesdir

https://github.com/not-an-aardvark/eslint-plugin-rulesdir

`eslintrc.js` を使っている前提だが最も最新か？

https://github.com/cletusw/eslint-plugin-local-rules

使い方の癖があるように見えるが、任意のeslintrcの形式で対応できそう

余談 勉強会の主催

今回ESLintの独自ルールを調べるにあたって、勉強会を企画してみました

https://connpass.com/event/232064/

2021/12/13に開催し、共催の RyoKawamata さんと2時間作業してなんとか形にできた（ありがとうございました）

勉強会駆動開発（Meetup Driven Development, MDD？）よさそう

勉強会企画したい方、一緒に主催できるので声かけてくださいw

まとめ

ESLintの独自ルールを書くために難しい知識は必要ない

ドキュメントは丁寧だが、概念をつかむまで少々大変なのと、`@types/eslint`などを見たほうが理解が早く進むところがあるので、手を動かして学ぶのが一番早そう

勉強会駆動開発おすすめ

告知

Twitterフォローしてね

@Meijin_garden

だいたいWeb開発の話か将棋の話してます

絶賛エンジニア採用中です！

オンライン家庭教師のスタートアップです！

2020年リリースで、これから**塾や家庭教師や通信教育を置き換える教育スタイル**を広げていきます

少しでも興味があればカジュアル面談歓迎なので、DMやMeetyでお声がけください！！

ご清聴ありがとうございました