初めてESLintプラグインにコントリビュートした話

～ESLintルール作成のすゝめ～

@meijin_garden / 株式会社NoSchool CTO

お話する内容

1. ESLintのルール設定にこだわる意義
2. 関わっているプロダクトで見つけた課題
3. OSSにコントリビュートした内容

対象読者

- ESLintを使ったことはあるけど、使う意義があまりわかっていない方
- ESLintのルールを自前で実装するイメージが湧いていない方
- OSSコントリビュートのハードルが高いと思っている方

自己紹介

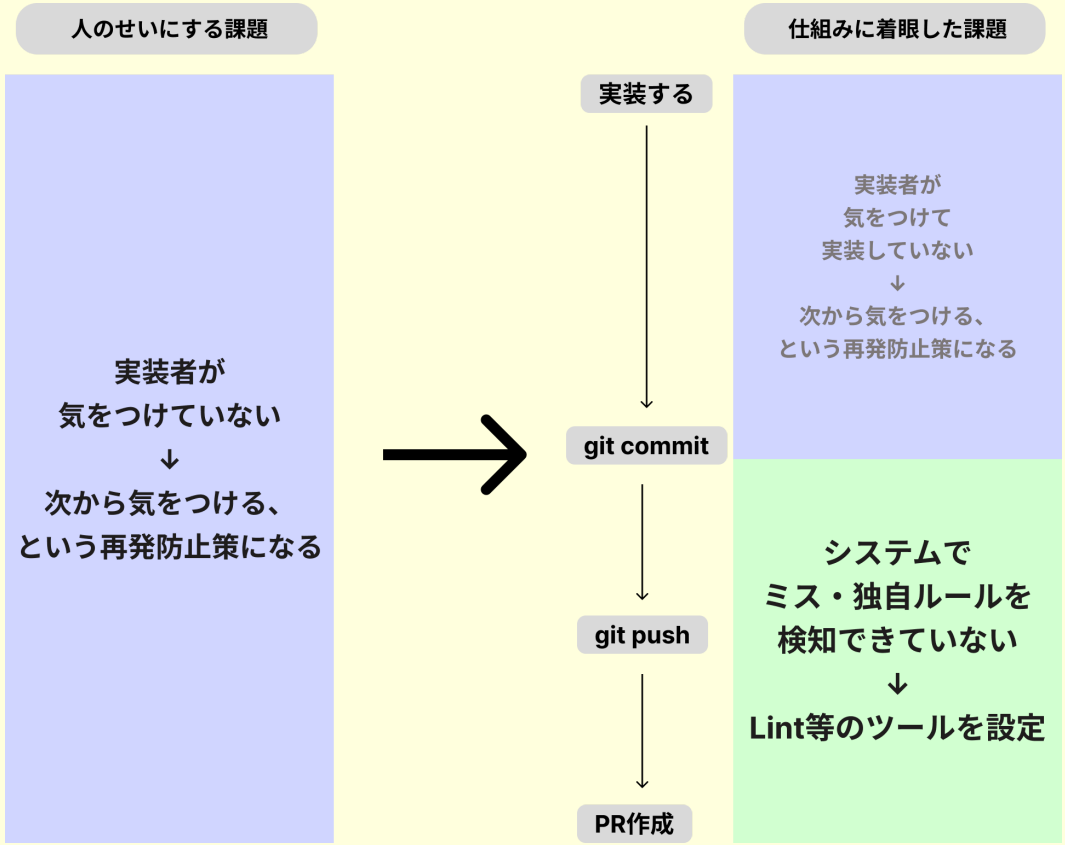
- 名人
  - Twitter(X): 名人｜マナリンクCTO
  - Zenn: https://zenn.dev/texmeijin
- 株式会社NoSchool CTO
  - オンライン家庭教師マナリンク(https://manalink.jp/)
- 個人開発
  - テストメーカー(https://test-maker.app/)
- 好きな言語はTypeScript、好きなHTTPヘッダーはContent-Disposition
- 趣味
  - 将棋☗、カメラ📸、ラム酒🥃、個人開発💻、筋トレ💪、高校野球観戦⚾

ESLintのルール設定にこだわると嬉しいこと

簡単な例え話

～あるところに、うっかりデバッグ用のconsole.logを含んで提出されたPull Requestに怒る人がいました～

課題を「人」の問題と「仕組み」の問題に切り分け



仕組みで防げることは仕組みで防ぐ

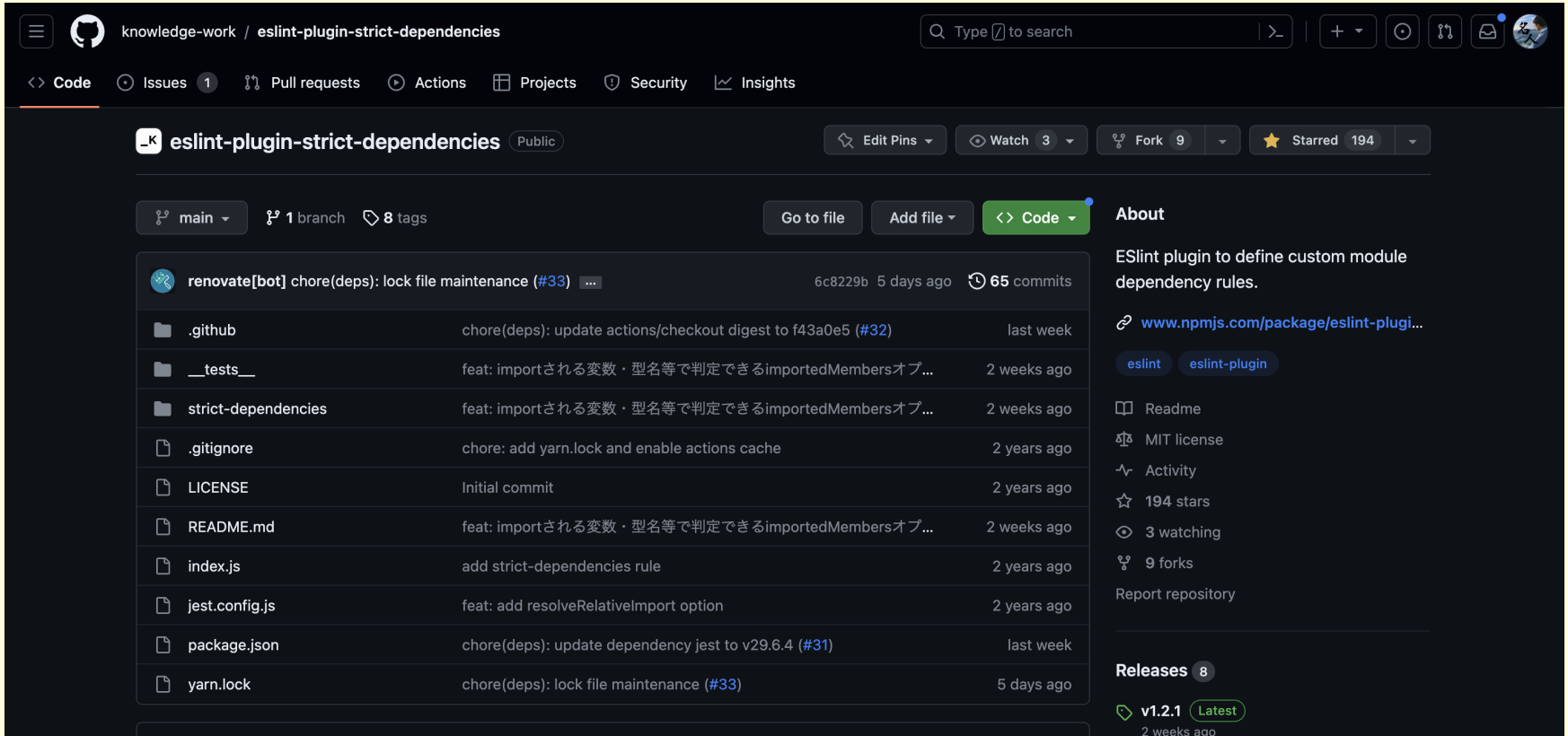
- 注意する、とか気をつける、といった属人的な方針をネクストアクションにするのは最後の手段にする
  - レビュワーが頑張る、も同じ
- 人間は（自分も含め）誰でもミスをする、忘れてしまう可能性がある
- IDE、git hooks、CIなどを使ってチェックを自動化する
- 意外とできることは多い

ESLintでできること

- console.logの入れっぱなしなど**イージーミスを防ぐ**
  - https://eslint.org/docs/latest/rules/no-console
- いわゆる**書き方の好み**の問題をPrj内で統一する
  - https://github.com/jsx-eslint/eslint-plugin-react/blob/master/docs/rules/jsx-boolean-value.md
- 見やすくするが、**手作業**するかしないかが人によって分かれるやつ
  - https://github.com/import-js/eslint-plugin-import/blob/main/docs/rules/order.md
- 社内で決めた**アーキテクチャを統一**する
  - https://www.npmjs.com/package/eslint-plugin-strict-dependencies
    - **これが本スライドで話す主題です**

コントリビュートしたESLintプラグイン

eslint-plugin-strict-dependencies



どんなプラグイン（だった）か

- あるモジュールから別のモジュールをimportできる・できないのルールを規定する
  - `.eslintrc.js`に設定を書き、破られていたらエラー扱いとする
  - husky/lint-stagedやCIで強制できる
- 利用例1：プロダクトで決めたアーキテクチャの徹底
  - `src/components/page`は`src/pages`からのみ
  - `src/components/features`は`src/pages`からのみ呼べる
  - `src/components/ui`は`src/components/page`、`src/components/features`からのみ呼べる
- 利用例2：外部ライブラリに対する腐敗防止層利用の徹底
  - MUIのコンポーネントは`src/components/ui`からのみ呼べる
  - `@sentry/react`は`src/libs/sentry.ts`からのみ呼べる
  - `react-icons`は`src/components/ui/icons`からのみ呼べる

設定例（※旧eslintrc形式）

```
module.exports = {
  plugins: ['strict-dependencies'],
  rules: {
    'strict-dependencies/strict-dependencies': [
      'error',
      [
        {
          "module": "src/components/ui",
          "allowReferenceFrom": ["src/components/page"],
          "allowSameModule": true
        },
        {
          "module": "next/router",
          "allowReferenceFrom": ["src/libs/router.ts"],
          "allowSameModule": false
        },
      ]
    ],
  },
}
```

利用シーンと、そこで見つけた課題

背景

- 弊チームでもさっそくアーキテクチャの徹底と、外部ライブラリの利用制限で用いた
- あるとき、メンバーが`react.Suspense`のラッパーを作ってくれた
- なので、`Suspense`を直接呼ぶのではなく作ったラッパーを使うように徹底したい

気がついたこと

- 「`react`の中の`Suspense`のみ利用範囲を制限したい」ケースには対応できない
- これまで通り設定すると、`react`からのimportが全部NGになってしまう

```
      {
        "module": "react",
        "allowReferenceFrom": ["src/libs/suspense.ts"],
        "allowSameModule": false
      },
```

ではどうするか

importするメンバも指定できる機能を追加しよう！

`import A from B`に対して「**BからAを**importしているとき」というより細かな条件を指定できるように

イメージ

```
{
  "module": "react",
  "targetMembers": ["Suspense"],
  "allowReferenceFrom": ["src/libs/suspense.ts"],
  "allowSameModule": false
},
```

なんか実装できそう

既存の実装を理解したら、

【import文において、import対象のモジュール名を取得する方法と、対象ファイル名を取得する方法】

がわかるはずなので、もう少し応用してimportするメンバー名を取得する方法を考えればよさそう。

機能追加するときは（一旦）ここだけ見る

https://github.com/knowledge-work/eslint-plugin-strict-dependencies/blob/9e4064539a5b571efa7e8ea4c9f84a2f7f1c0926/strict-dependencies/index.js#L19

```
module.exports = {
  meta: {
    // meta情報なので機能理解にあたってはスルー
  },
  create: (context) => {
    // ここに色々書いてあるのも一旦スルー

    // ここでreturnされたものがプラグインの動作を決めるのでまずはここで全体理解
    return {
      ImportDeclaration: checkImport,
    }
  },
}
```

ざっくり解説

```
  return {
    ImportDeclaration: checkImport,
  }
```

`ImportDeclaration`とは

- AST(後述)におけるimport文のこと
- 例：`import A from B`

`ImportDeclaration: checkImport`と指定すると

- ESLintプログラムがimport文を見つけたら、`checkImport`関数を実行するようになる
- 個人的には脳内で「`onImportDeclarationAppeared: checkImport`」といった風に読み替えて読んでいて、**イベントハンドラをプラグインを通して登録している**と考えるとしっくりきています

AST(Abstract Syntax Tree)とは

- こちらで規定されている：https://github.com/estree/estree
  - ※JavaScriptに限らずどの言語にもある一般的な概念
- ASTは以下のようなツールで見れる
  - https://astexplorer.net/
  - https://ts-ast-viewer.com/
- 従って、import文のことを`ImportDeclaration`と呼ぶのは予約語です

覚えておくこと

- 全体的に
  - よほどのことがない限り、ASTについて丸暗記したり徹底理解する必要はない
  - 個人的には「**まあ、プログラムをプログラムが解析したり変換するなら、プログラムはただの文字列なので、プログラムが操作可能な形式に変換しないとダメやんな〜**」くらいに思っておく
- `ImportDeclaration: checkImport`における`checkImport`関数について
  - 前述の通り、import文が見つかったときにそのimport文に対して実行する関数
  - 第1引数にASTでパースされた`ImportDeclaration`型のオブジェクトが渡される
  - `ImportDeclaration`型の詳細はAST Explorerなどで見たりtypescript-eslintを見て把握する

`ImportDeclaration`型

https://github.com/typescript-eslint/typescript-eslint/blob/6ed0ca43b1fea58522f1135e224ddc3fe788b40c/packages/ast-spec/src/unions/ImportClause.ts#L5

```
import type { ImportDefaultSpecifier } from '../special/ImportDefaultSpecifier/spec';
import type { ImportNamespaceSpecifier } from '../special/ImportNamespaceSpecifier/spec';
import type { ImportSpecifier } from '../special/ImportSpecifier/spec';

export type ImportClause =
  | ImportDefaultSpecifier
  | ImportNamespaceSpecifier
  | ImportSpecifier;
```

```
export interface ImportSpecifier extends BaseNode {
  type: AST_NODE_TYPES.ImportSpecifier;
  local: Identifier;
  imported: Identifier;
  importKind: ImportKind;
}
```

実装方針

- 前述の知識から、今回の目的の一つである「import対象のメンバー名を取得する」方法は `node.specifiers`を使う

```
    // ここのnodeはImportDeclaration型
    function checkImport(node) {
      // 〜中略〜
      // specifiersにはImportDefaultSpecifier/ImportNamespaceSpecifier/ImportSpecifier型があり、ImportSpecifierの場合のみimportedが存在する
      const importedModules = node.specifiers.filter(spec => 'imported' in spec).map(spec => spec.imported.name)
```

テストコードと動作確認

- 本プラグインはありがたいことにテストコードが用意されていたので、手元にCloneして実装した後にデグレがないか実行

ローカルでの動作確認

- 方法は複数あると思うが、`yarn`や`npm`はローカルにCloneしたモジュールをinstallすることもできるので、手元で改修後のプラグインをinstallして自社プロダクトにて動作確認した
  - e.g. `yarn add -D ../../../hoge/eslint-plugin-strict-dependencies`

Pull Request提出〜マージまで

- **6月30日**：弊社メンバーからSuspenseラッパー実装の発案があり、それに伴ってプラグインへの機能追加を思いつく
- **6月30日**：なんとなく動くやつができる
- **7月2日**：テストコードを書き、動作確認もできたのでPRを提出
  - https://github.com/knowledge-work/eslint-plugin-strict-dependencies/pull/12
  - 和製OSSなので日本語で書けたのがありがたい
- **8月18日**：なんだかんだあってPull Requestをマージしていただけた🎉

※今回ESLintプラグインへのコントリビュートは初めてでしたが、ESLintプラグインの作り方自体は昨年から知ってはいました。なので機能追加したいときにすぐに動けたと思います。今すぐ解決したいIssueがなくても、ESLintプラグインの作り方をざっくり知っておくといつか使えるかもしれません

まとめ

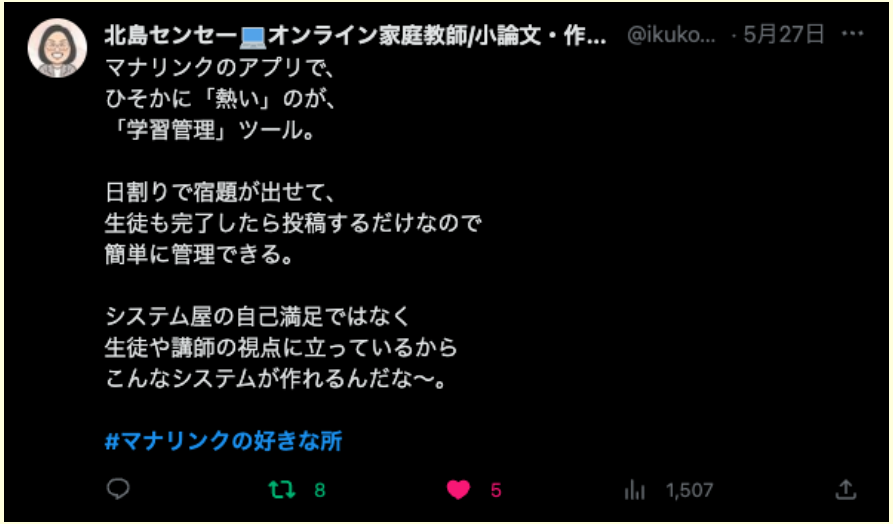
まとめ

- ESLintのルール設定にこだわると嬉しいこと
  - プログラミングで起きる問題は、人の問題と仕組みの問題に切り分けられる
  - 仕組みの問題のうち、いくつかはESLintで解決できる
- ESLintプラグインを作る/機能追加するときは
  - ASTの知識は必要だが、丸暗記する必要はない
  - 既存の実装を読んで、どういうノードがあるか、どういうノードを取得すればいいかを理解する
  - ESLintプラグインでできることを知っておくと、いつかタイミングが来たときに役に立つ

宣伝

「マナリンク」について

- オンライン家庭教師マナリンク(https://manalink.jp/)
- コロナ禍から増え始めた新しい教育の仕事である「オンライン家庭教師」を広めるスタートアップ
- 先生と保護者様のマッチングサイトと、指導開始後の宿題や指導料金の管理等のツールを提供しています



弊社の開発チームについて

- メンバー構成
  - 全4名（CTO、フルスタック2名、React Nativeエンジニア1名）
- 【仕組みを憎んで人を憎まず】
  - 毎月**最大3営業日程度**「仕組み化・自動化」に関する工数を使います
  - 実績の一例
    - ローカル環境の色々なデータの自動生成・破棄コマンドの作成
    - PHPStanの導入と設定
    - SQLのSlow Query検知やN+1の自動テスト時の検知
    - Mock Service Workerの導入とテストコードへの統合
    - renovateによるライブラリバージョンアップの自動化
- 勉強会
  - 1年以上、週1〜2回の社内勉強会を続けています（※業務時間内）
  - https://zenn.dev/manalink_dev/articles/manalink-study-meetup-history-front-and-network

募集内容

- 開発メンバーを随時募集しているのですが、いきなり面接等は敷居が高いと思うので
- 以下募集しています！
  - 弊社の社内勉強会にゲスト参加✏️
    - 平日15時〜15時半頃
    - 平日夜
  - 弊社メンバーとレンタルジムを借りて合同筋トレ💪
  - 弊社メンバーと秋葉原の国内最大級のボルダリング上で壁登り🧗
  - 普通にカジュアル面談（オンライン30min）
  - バーにお酒🥃を飲みに行く（私はラム酒がおすすめなのでラム酒デビューしたい方布教させて）

ご清聴ありがとうございました

この後の懇親会でぜひお話しましょう！