

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

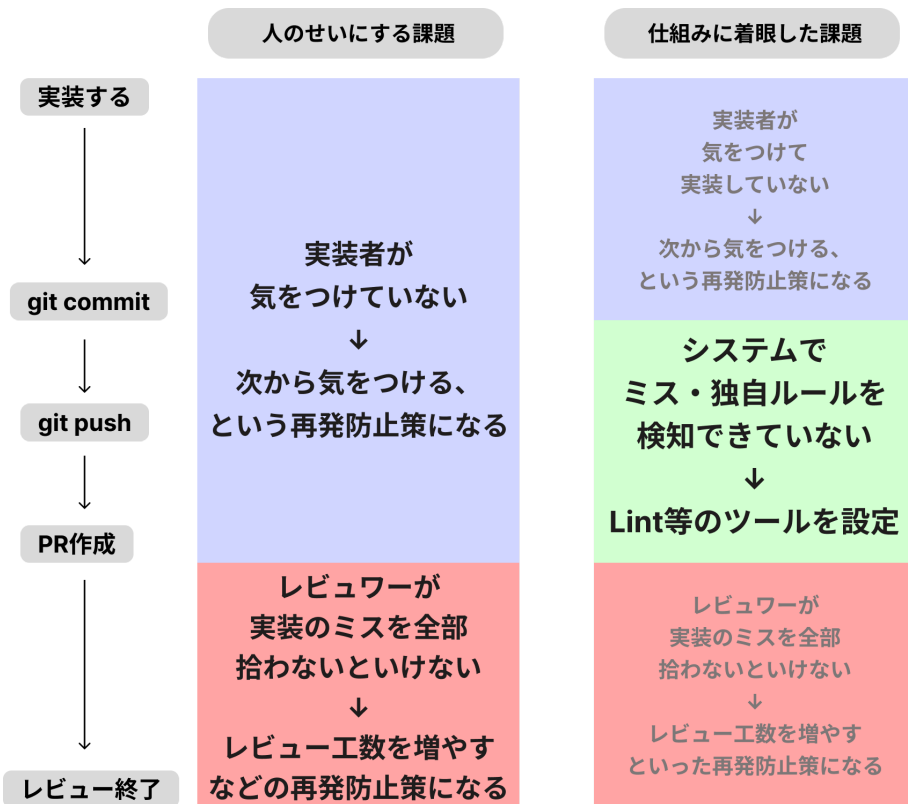
～あるところに、うっかりデバッグ用のconsole.logを含んで提出されたPull Requestに怒る人がいました～



console.logをそのままにして
Pull Request出すなんて
ありえないやろ！（怒）



課題を「人」の問題と「仕組み」の問題に切り分ける



仕組みで防ぐ

- git commit時
 - lint-staged/husky で変更ファイルに対してLintを実行する
 - またはIDEでできるように設定ファイルを配布するなどもあり
- Pull Request時
 - CIでLintを実行する
 - reviewdogやeslint-changed-filesを使えば、変更ファイルのみを対象にLintを実行できる

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

knowledge-work / eslint-plugin-strict-dependencies

Q

Type to search

>

+

<>

Code

Issues

1

Pull requests

Actions

Projects

Security

Insights

eslint-plugin-strict-dependencies

Public

Edit Pins

Watch 3

Fork 9

Starred 194

main

1 branch

8 tags

Go to file

Add file

<> Code

About

renovate[bot] chore(deps): lock file maintenance (#33)

6c8229b 5 days ago

65 commits

.github	chore(deps): update actions/checkout digest to f43a0e5 (#32)	last week
__tests__	feat: importされる変数・型名等で判定できるimportedMembersオブ...	2 weeks ago
strict-dependencies	feat: importされる変数・型名等で判定できるimportedMembersオブ...	2 weeks ago
.gitignore	chore: add yarn.lock and enable actions cache	2 years ago
LICENSE	Initial commit	2 years ago
README.md	feat: importされる変数・型名等で判定できるimportedMembersオブ...	2 weeks ago
index.js	add strict-dependencies rule	2 years ago
jest.config.js	feat: add resolveRelativeImport option	2 years ago
package.json	chore(deps): update dependency jest to v29.6.4 (#31)	last week
yarn.lock	chore(deps): lock file maintenance (#33)	5 days ago

README.md

ESLint plugin to define custom module dependency rules.

www.npmjs.com/package/eslint-plugin-strict-dependencies

eslinteslint-plugin

Readme

MIT license

Activity

194 stars

3 watching

9 forks

Report repository

Releases 8

v1.2.1

Latest

2 weeks ago

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``からしか呼べない
 - ``src/components/features``は``src/pages``からしか呼べない
 - ``src/components/ui``は``src/components/page``、``src/components/features``からしか呼べない
- 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``からしか呼べない
 - ``@sentry/react``は``src/libs/sentry.ts``からしか呼べない
 - ``react-icons``は``src/components/ui/icons``からしか呼べない

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

- 前述の通り`import A from B`でいうところのBを指定するため、「`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  
},
```

どうするか

`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機能追加したい！

イメージ

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

なんか既存の実装を理解したら、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。

ノリで理解するESLintプラグイン実装

※ちゃんとした解説はすでにググったらたくさんあるので

機能追加するときは（一旦）ここだけ見る

```
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`というのは

- import文のこと
- 例: ``import A from B``

`ImportDeclaration: checkImport`と指定することで

- 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

`ImportDeclaration`って予約語なの？

- 予約語です（適当に決めたら動きません）
- 他にも`ExportSpecifier`とか`FunctionDeclaration`とか色々ある
- ESLint実行時には、JavaScriptのソースコードをASTという形式に変換するが、そのときソースコード中の各パーツ（ノードという）に当てられるType
- なので、これらの名前はJavaScriptのASTで規定されている
- ESLintって要は「JavaScript内にこういう種類のノードがあったらこういうルールに沿っているかチェックしてね」の集合なので、作りたいルールに応じてASTのノードのTypeを調べて関数を定義していく感じ

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
- ソースコードはそのままの文字列で、意味をもったまとまりに分解できないので、決められた形式のオブジェクトに変換することで、後からLintしたりフォーマットしたりできる
- ASTは以下のようなツールで見れる
 - <https://astexplorer.net/>
 - <https://ts-ast-viewer.com/>
- ちなみにAbstractがあるということはConcreteもあります
 - ASTは不要な空白やカッコなどは除いて、意味のあるノードだけをまとめる
 - Concrete Syntax Treeは空白やカッコも含めて全てのノードをまとめる考え方（らしい）
 - 参考：<https://github.com/estree/estree/issues/41>
 - ASTのほうが一般的なはずだが、たとえばPrettierは空白も扱うのでASTではなくCSTなのか？とか疑問が出ました。詳しい方いたら教えてほしいです！

例

```
someFunction();
```

```
{
  "type": "Program",
  "start": 0,
  "end": 15,
  "body": [
    {
      "type": "ExpressionStatement",
      "start": 0,
      "end": 15,
      "expression": {
        "type": "CallExpression",
        "start": 0,
        "end": 14,
        "callee": {
          "type": "Identifier",
          "start": 0,
          "end": 12,
          "name": "someFunction"
        },
        "arguments": [],
        "optional": false
      }
    }
  ]
}
```

```
"body":[
  {
    "type": "ImportDeclaration",
    "start": 0,
    "end": 38,
    "specifiers": [
      {
        "type": "ImportSpecifier",
        "start": 8,
        "end": 24,
        "imported": {
          "type": "Identifier",
          "start": 8,
          "end": 16,
          "name": "useState"
        },
        "local": {
          "type": "Identifier",
          "start": 20,
          "end": 24,
          "name": "hook"
        }
      }
    ],
    "source": {
      "type": "Literal",
      "start": 31,
      "end": 38,
      "value": "react",
      "raw": "'react'"
    }
  }
]
```

クイズ

このASTの元のプログラムはなんでしょう？

正解

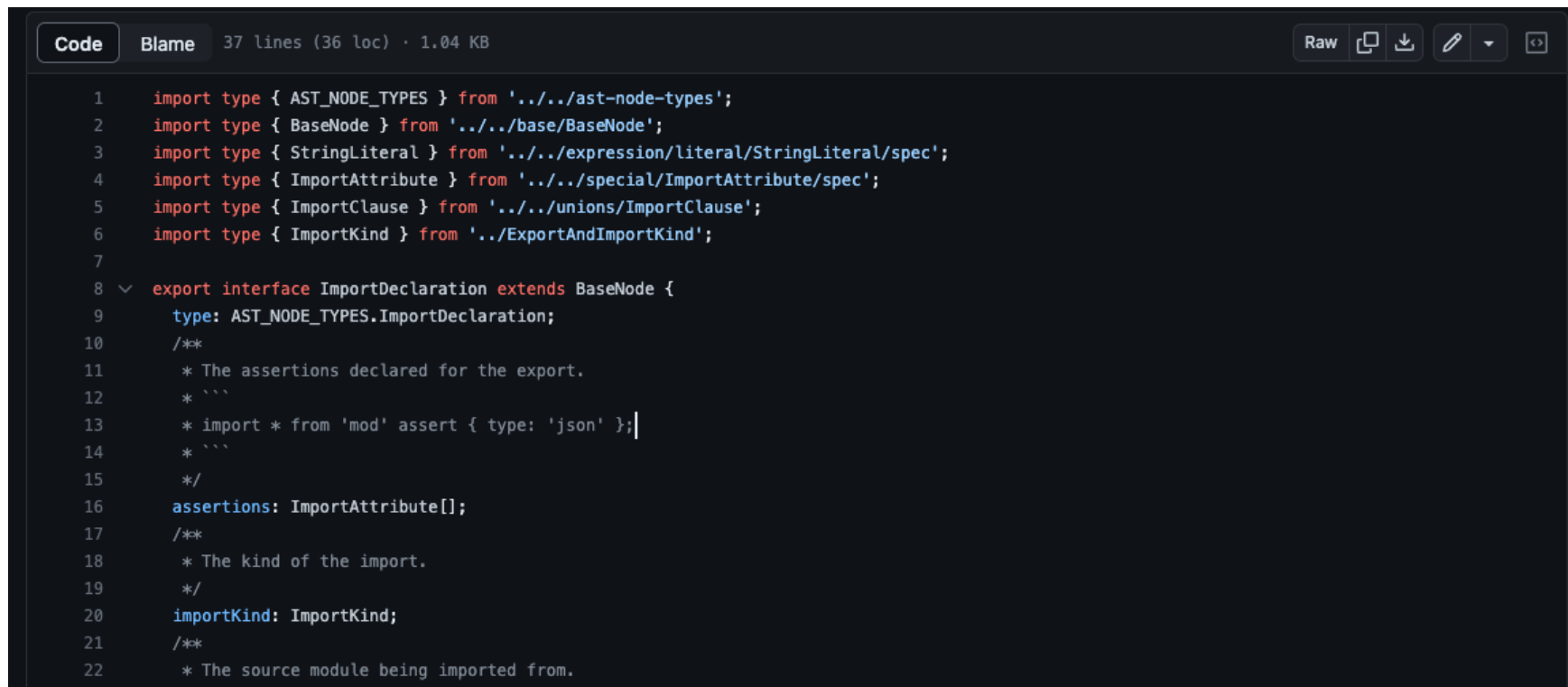
```
`import {useState as hook} from 'react'`
```

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``型って何やねん。という話ですが、前述のAST Explorerなどで見てもいいし、[typescript-eslint](#)に型情報が記載されているのでそちらを見る

参考

<https://github.com/typescript-eslint/typescript-eslint/blob/d2973cca391eeb763d72c5ef2f49cdafe4ffc216/packages/ast-spec/src/declaration/ImportDeclaration/spec.ts>

A screenshot of a code editor with a dark theme. The editor shows a file named 'spec.ts' with 37 lines of code. The code is TypeScript, defining types and an interface for ImportDeclaration. The interface extends BaseNode and includes properties for type, assertions, importKind, and source. The code is syntax-highlighted with various colors for keywords, types, and comments. The editor interface includes tabs for 'Code' and 'Blame', and a status bar at the bottom showing '37 lines (36 loc) · 1.04 KB'. On the right side, there are icons for 'Raw', 'Copy', 'Download', 'Edit', and a dropdown menu.

```
1  import type { AST_NODE_TYPES } from '../../ast-node-types';
2  import type { BaseNode } from '../../base/BaseNode';
3  import type { StringLiteral } from '../../expression/literal/StringLiteral/spec';
4  import type { ImportAttribute } from '../../special/ImportAttribute/spec';
5  import type { ImportClause } from '../../unions/ImportClause';
6  import type { ImportKind } from '../ExportAndImportKind';
7
8  export interface ImportDeclaration extends BaseNode {
9    type: AST_NODE_TYPES.ImportDeclaration;
10   /**
11    * The assertions declared for the export.
12    * ```
13    * import * from 'mod' assert { type: 'json' };|
14    * ```
15    */
16   assertions: ImportAttribute[];
17   /**
18    * The kind of the import.
19    */
20   importKind: ImportKind;
21   /**
22    * The source module being imported from.
```

既存実装

- `context.getFilename()` で対象のファイル名が取得できる
- `node.source.value` でImportDeclarationのImport先の名前が取れる

```
function checkImport(node) {
  const fileFullPath = context.getFilename()
  const relativeFilePath = normalize(path.relative(process.cwd(), fileFullPath))
  const importPath = resolveImportPath(node.source.value, resolveRelativeImport ? relativeFilePath : null, pathIndexMap)

  dependencies
    .filter((dependency) => isMatch(importPath, dependency.module))
    .forEach((dependency) => {
      const isAllowed =
        // 参照元が許可されている
        dependency.allowReferenceFrom.some((allowPath) =>
          isMatch(relativeFilePath, allowPath),
        ) || // または同一モジュール間の参照が許可されている場合
        (dependency.allowSameModule && isMatch(relativeFilePath, dependency.module))

      if (!isAllowed) {
        context.report(node, `import '${importPath}' is not allowed from ${relativeFilePath}.`)
      }
    })
}
```

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
- `node.specifiers`にImport対象が入っていて、それぞれこんな形状だが、Default Importは`imported.name`が無いなど罫があるので存在確認が必要

```
const importedModules = node.specifiers.filter(spec => 'imported' in spec).map(spec => spec.imported.name)
```

```
{
  "type": "ImportSpecifier",
  //
  "imported": {
    "type": "Identifier",
    //
    "name": "useState"
  },
  "local": {
    "type": "Identifier",
    //
    "name": "useState"
  }
},
```

テストコード

- 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
- デグレがないと確認できたら、追加機能分のテストを実装
- 前述したImportDeclarationが複数パターンある件にハマらないため、モックデータの拡充もついでにした

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

- 2023年6月30日16時頃：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18時頃：なんとなく動くやつができる
- 7月2日12時頃：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🎉

まとめ

まとめ

- ESLintのルール設定にこだわると嬉しいこと
 - 人の問題と仕組みの問題に切り分けられる
- 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
 - テストコードを書いて、デグレがないことを確認する

宣传

「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

マナリンク 数学, 大学受験, 英検 ログイン はじめての方へ

プロの先生を選べる オンライン家庭教師

大学生のアルバイトの先生は一切いません。
社会人のプロの先生が500名以上在籍中です。

オンライン家庭教師とは

Zoomなどのオンラインツールを使用して
1対1の個人指導を行うオンライン型の家庭教師です。

画面カメラを用いてお子さまの手元を映し、先生は常にどこでつまづいているかを確認しながら進めるため、**対面に劣らない授業**

LINEで相談 電話で相談

北島センセー オンライン家庭教師/小論文・作... @ikuko... · 5月27日 ...

マナリンクのアプリで、
ひそかに「熱い」のが、
「学習管理」ツール。

日割りで宿題が出せて、
生徒も完了したら投稿するだけなので
簡単に管理できる。

システム屋の自己満足ではなく
生徒や講師の視点に立っているから
こんなシステムが作れるんだな〜。

#マナリンクの好きな所

8 5 1,507

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】**
 - 「仕組み化」を重視しており、より本質的な開発に注力できるように日々課題の発見・解決策の調査や実装をしています
 - 毎月最大3営業日程度「仕組み化」に関する工数を使います
 - ESLint、PHPStan、PHPCS、GitHub Actions、新しいSaaSの導入、その他Dev系のライブラリの導入やアプデなどを行います
- 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しましょう！