やる夫で学ぶ React、Reduxだお‥

知識ゼロから環境構築するお。
Redux-toolkitまで学ぶお。

2021年1月版

まだReduxで消耗してんの？ Redux-Toolkitで楽するお…

おおおお　redux-toolkitやて～　wwwwwww
キターーー(* v*)ーーー　明日からはフロントエンド担当だお･･･
Typescript･･･　ちょwww　吹いたwwwww　フロント
！おめでとう👏👏👏👏　俺も知りたいす！
javascript･･････　お勉強は、楽しいお･･･

やる夫で学ぶ「react-redux」

— redux-toolkit で、簡単・完璧理解だお··· —

気分はもう　著

技術書典 10（2021 年夏）新刊
2021 年 6 月 25 日　ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、Ⓡ、Ⓒ などのマークは省略しています。

はじめに

このたびは、「やる夫で学ぶ-Redux- Redux Toolkit は簡単だお･･･」を、手にしていただき誠にありがとうございます。

私は 15 年近く Visual Studio 上で C#を使って UI のあるプログラムを書いていました。昨今は、同一コードでマルチプラットフォーム上で動作するアプリケーションに向かっています。
今では、Microsoft .NET がマルチプラットフォームで動作するようになり、UI なども充実し始めています。
あるとき、あるプロジェクトでは、「Windows、Mac 上にて同一 UI で動作」が要求されました。そのために選択したのが、ブラウザ上で動作する Web アプリケーションです。
さらに、Web アプリケーションではローカルファイルにアクセスできないため、Electron を使用することで、Web アプリケーションをマルチプラットフォームで動作するアプリケーションにしました。

当時は、Web アプリケーションのフレームワークとして Angular と React が候補に挙がりましたが、React の方が学習コストが低いと言われていたため、React を学びました。

本書は、私が React や Redux を使い始めたときに「こんな本があれば良かったのに･･･」を目指して書きました。React,Redux の初学者から中級の方のお役に立てれば幸いです。

そして、私が一番欲しかった「常に最新の情報」を掲載していきたいと思っています。

React、Redux の解説書やチュートリアルは、書籍・インターネット上にたくさんあります。しかし、つぎつぎと新機能が追加され本家以外の情報は、あっという間に古くなってしまいます。この書籍は、掲載情報を最新にするために電子書籍のみとし、古くなった情報は随時更新していきたいと思っています。

お気付き点がございましたら、Guthub 上へお寄せください。

本書は、フロントエンド開発で使用されている
* react * redux(redux-toolkit)

を習得するために、開発環境の作成 (つまりゼロ) から始め、チュートリアルの定番の ToDo リストを作成します。

すでに開発環境を整えている方は、第 1 章を飛ばしてもかまいません。

また、作成する ToDo アプリケーションは GutHub に公開していますが、こちらも最新の情報に更新していくつもりです。GutHub では、章毎に別ブランチにしてありますので、写経がメンドウな方は章に対応したブランチを使ってください。

git、GitHub の使い方については、本書では取り扱いません。
本書は、チュートリアルによくある ToDo リストを

1. Redux なしで作成
2. Redux を導入
3. Redux Toolkit を導入

と、書き換えていくことで Redux を用いた「状態管理 (アプリケーション全体でのデータ)」を理解してもらえるようになっています。

create-react-app を使用して、ゼロからの手順を解説しています。写経がメンドウな人は、GitHub にあるコードを使ってください。

必要なものは、すべて無料でそろえることができる以下のものです。これらが何なのか、そして、インストール方法は第１章にて解説しています。　* Node.js　* yarn（npm を使われる方は不要）　* Microsoft Visual Studio code　* Google Chrome

それでは、始めていきましょう。

目次

はじめに i

第 1 章 ゼロから始める (開発環境構築) 1

1.1 Node.js 2

1.1.1 Node.js について 2

1.1.2 Node.js のインストールの前に 8

1.1.3 nvm 9

1.2 Microsoft Visual Studio Code + 拡張機能 15

1.2.1 VSCode のインストール 15

1.2.2 VSCode の拡張機能 16

1.2.3 Eslint 17

1.2.4 Prettier 17

1.3 Google Chrome + 拡張機能 19

1.3.1 Google Chrome のインストール 19

1.4 第 1 章のまとめ 24

第 2 章 スタートプロジェクトの作成 25

2.1 create-react-app コマンド 25

2.2 アプリケーションを実行 28

2.3 create-react-app で作成された中身 30

2.4 eslint、prettier 32

2.4.1 eslint、prettier のインストール 33

2.5 eslint、prettier の指摘を修正 39

2.6 第 2 章のまとめ 46

第 3 章 Todo リストの作成 (React のみ) 47

3.1 React とは？ 47

3.2 表示するデータの型を決める 48

3.3 データ表示画面 49

3.4 React hooks を使用して、データの追加・編集・削除 50

第 1 章

ゼロから始める (開発環境構築)

▲図 1.1: やる夫が、突然の移動を命じられたお！

本章では、React、Redux の開発環境を作成します。

「なぜ、これらが必要なのか？」

は、とりあえず置いといて

- Node.js
- Microsoft Visual Studio Code + 拡張機能
- Google Chrome + 拡張機能

をインストールしてください。これらは、すべて無償で提供されています。

なお、これらの準備が整っている方は、本章を読み飛ばしていただいてもかまいません。

1.1 Node.js

♡| 1.1.1 Node.js について

Node.js とは？

「Node.js」は、通常ブラウザ上で実行される JavaScript をサーバや PC 上で実行できるようする「**JavaScript 実行環境**」です。たとえば、Windows に Python をインストールすると「python.exe」を使い python ファイルを Windows 上で実行できます。

▲図 1.2: python を実行

同じように、**Node.js** をインストールすると、「node.exe」を使い JavaScript ファイルを実行できます。例として、Node.js のドキュメント (Guides)「How do I start with Node.js after I installed it?」[*1]にある以下のスクリプトを実行してみましょう。

[*1] https://nodejs.org/en/docs/guides/getting-started-guide/

▼ リスト 1.1:

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
  });

  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
  });
```

このスクリプトを「app.js」のファイル名で保存し、実行します。

▼ リスト 1.2: app.js を実行

```
> node app.jp
Server running at http://127.0.0.1:3000/
```

ブラウザを開き、http://127.0.0.1:3000 へアクセスすると、ブラウザに実行結果が表示されます。

▲ 図 1.3: app.js の実行

通常の HTML に埋め込まれた JavaScript をブラウザから実行すると、OS 上の機能を使用する (たとえば、ファイルの書込み・読み込み) ことなどは制限されますが、Node.js で実行すると OS の機能も使用できます。

詳しくは、本家の「Node.js とは[*2]」を参照してください。

[*2] https://nodejs.org/ja/about/

Node.js について

▲図 1.4: Node.js をインストールするお！

▲図 1.5: Node.js のバージョンには注意

では、Node.js の本家トップページ:`https://nodejs.org/ja/` へアクセスします。

▲図 1.6: Node.js トップページ

ここでダウンロード可能なのは、「14.17.0 LTS(Long Term Support) 推奨版」と「16.3.0 最新版」[*3]の 2 つがあります。

LTS 版、最新版は以下のロードマップにより更新されます。

[*3] 2021/06/10 現在

リリース

GitHub 上で編集

Node.js のメジャーバージョンは 6 ヶ月間 *現行* リリースの状態になり、ライブラリの作者はそれらのサポートを追加する時間を与えられます。 6 ヶ月後、奇数番号のリリース (9, 11 など) はサポートされなくなり、偶数番号のリリース (10, 12 など) は *アクティブ LTS* ステータスに移行し、一般的に使用できるようになります。 *LTS* のリリースステータスは「長期サポート」であり、基本的に重要なバグは 30 ヶ月の間修正されることが保証されています。 プロダクションアプリケーションでは、*アクティブ LTS* または *メンテナンス LTS* リリースのみを使用してください。

リリース	ステータス	コードネーム	初回リリース	アクティブ LTS 開始	メンテナンス LTS 開始	サポート終了
v12	メンテナンス LTS	Erbium	2019-04-23	2019-10-21	2020-11-30	2022-04-30
v14	アクティブ LTS	Fermium	2020-04-21	2020-10-27	2021-10-19	2023-04-30
v16	現行		2021-04-20	2021-10-26	2022-10-18	2024-04-30
v17	次期		2021-10-19		2022-04-01	2022-06-01
v18	次期		2022-04-19	2022-10-25	2023-10-18	2025-04-30

日程は変更される場合があります。

▲図 1.7: Node.js ロードマップ

Node.js の Releases:https://nodejs.org/ja/about/releases/ にあるように、Node.js は、各年の 4 月、10 月にリリースされ、* Current * Active * Maintenance のフェーズを経ますが、メジャーバージョン番号が偶数のものだけが、Active 期間を経て長期サポートされます。

上記トップページにある Node.js 14 は、2023/4/30 までの長期サポートとなります。実際のプロジェクトで使用する場合は、よほどの理由がない限りは最新の LTS 版を使用します。

♡| 1.1.2　Node.js のインストールの前に

Node.js は、ロードマップにより定期的にバージョンアップされます。又、Node.js 自体の不具合の修正などでマイナーバージョンアップも行われます。

プロジェクト開発中のマイナーバージョンアップでも検証が必要になりますが、メジャーバージョンアップの場合はさらに大きな検証が必要になります。場合によっては、ソースコードの大幅な改良をしなければならなくなります。

それを避けるためにも、プロジェクト毎に Node.js のバージョンは固定して開発します。

通常は、OS にインストールできる Node.js のバージョンはひとつですが、長期にわたるサポートや新規プロジェクト開発のためには、複数の Node.js のバージョンを切り替えて使用できるしくみを用意しましょう。

私が使用しているのは、nvm(node version manager):https://github.com/nvm-sh/nvm です。いろいろなバージョンの Node.js を、簡単にインストール・アンインストール・切替ができます。

♡| 1.1.3　nvm

nvm(node version manager) を使えば、複数バージョンの Node.js を 1 台の PC にインストールし、バージョンを切り替えることが簡単にできます。

Hub 上の nvm は、Shellscript(sh, dash, zsh, bash) 上で動作するため、Linux(UNIX 系)、macOS にインストールできます。

Windows 版:https://github.com/coreybutler/nvm-windows は、別な方が Hub 上で公開されています。コマンドが本家と少し違いますが、複数バージョンのインストール・バージョンの切り替えなど機能は問題ありません。

nvm のインストール

macOS

お使いの Terminal から以下のコマンドを実行してください。

▼ リスト 1.3: nvm のインストール

```
  >curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

インストール完了後には、.zshrc へ以下を追加してください。

▼ .zshrc へ追加

```
  export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || pr›
›intf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

Windows

nvm-windows のリリースページ:https://github.com/coreybutler/nvm-windows/releases/tag/1.1.8 より、最新版をダウンロードしインストールしてください。

nvm の使い方

macOS ではターミナルを起動し、Windows ではコマンドプロンプト、または、Windows Terminal を起動してください。

nvm と nvm-windows では、コマンドが少し違いますが、「nvm --help」を入力することで使用できるコマンドが表示されます。

▼ Mac OSX

```
> nvm --help

Node Version Manager (v0.37.2)
←中略
Example:
  nvm install 8.0.0                     Install a specific version number
```

```
  nvm use 8.0                          Use the latest available 8.0.x release
  nvm run 6.10.3 app.js                Run app.js using node 6.10.3
  nvm exec 4.8.3 node app.js           Run `node app.js` with the PATH pointing t›
›o node 4.8.3
  nvm alias default 8.1.0              Set default node version on a shell
  nvm alias default node               Always default to the latest available nod›
›e version on a shell

 ←中略
 Note:
  to remove, delete, or uninstall nvm - just remove the `$NVM_DIR` folder (usually›
› `~/.nvm`)
```

もし、32bit 版 Windows をお使いの場合には、インストールの際に 32bit 版を指定する「32」をコマンドの最後につけてください。

▼ windows

```
PS C:\Users\inabakazuya> nvm --help

Running version 1.1.7.

Usage:

  nvm arch                     : Show if node is running in 32 or 64 bit mode.
  nvm install <version> [arch] : The version can be a Node.js version or "latest" ›
›for the latest stable version.
                                 Optionally specify whether to install the 32 or 6›
›4 bit version (defaults to system arch).
                                 Set [arch] to "all" to install 32 AND 64 bit vers›
›ions.
                                 Add --insecure to the end of this command to bypa›
›ss SSL validation of the remote download server.
  nvm list [available]         : List the Node.js installations. Type "available" ›
›at the end to see what can be installed. Aliased as ls.
 ←中略
  nvm uninstall <version>      : The version must be a specific version.
  nvm use [version] [arch]     : Switch to use the specified version. Optionally s›
›pecify 32/64bit architecture.
                                 nvm use <arch> will continue using the selected v›
›ersion, but switch to 32/64 bit mode.
  nvm root [path]              : Set the directory where nvm should store differen›
›t versions of Node.js.
                                 If <path> is not set, the current root will be di›
›splayed.
  nvm version                  : Displays the current running version of nvm for W›
›indows. Aliased as v.
```

インストール可能な Node.js を表示

まずは、インストール可能な Node.js のバージョンを表示してみます。macOS の場合には、古いバージョンから最新バージョンまでが表示されます。

▼ Mac OSX

```
>nvm ls-remote
←古いバージョンから全て表示されるので中略
v14.13.1
v14.14.0
v14.15.0   (LTS: Fermium)
v14.15.1   (LTS: Fermium)
v14.15.2   (Latest LTS: Fermium)
 v15.0.0
 v15.0.1
 v15.1.0
 v15.2.0
 v15.2.1
 v15.3.0
 v15.4.0
```

一方、Windows の場合には、表形式で表示されます。新しいバージョンが上に表示されます。

▼ Windows

```
PS C:\Users\inabakazuya> nvm list available

|   CURRENT    |     LTS      |  OLD STABLE  | OLD UNSTABLE |
|--------------|--------------|--------------|--------------|
|    15.4.0    |   14.15.2    |   0.12.18    |   0.11.16    |
|    15.3.0    |   14.15.1    |   0.12.17    |   0.11.15    |
|    15.2.1    |   14.15.0    |   0.12.16    |   0.11.14    |
|    15.2.0    |   12.20.0    |   0.12.15    |   0.11.13    |
|    15.1.0    |   12.19.1    |   0.12.14    |   0.11.12    |
|    15.0.1    |   12.19.0    |   0.12.13    |   0.11.11    |
|    15.0.0    |   12.18.4    |   0.12.12    |   0.11.10    |
|   14.14.0    |   12.18.3    |   0.12.11    |    0.11.9    |
|   14.13.1    |   12.18.2    |   0.12.10    |    0.11.8    |
|   14.13.0    |   12.18.1    |    0.12.9    |    0.11.7    |
|   14.12.0    |   12.18.0    |    0.12.8    |    0.11.6    |
|   14.11.0    |   12.17.0    |    0.12.7    |    0.11.5    |
|   14.10.1    |   12.16.3    |    0.12.6    |    0.11.4    |
|   14.10.0    |   12.16.2    |    0.12.5    |    0.11.3    |
|    14.9.0    |   12.16.1    |    0.12.4    |    0.11.2    |
|    14.8.0    |   12.16.0    |    0.12.3    |    0.11.1    |
|    14.7.0    |   12.15.0    |    0.12.2    |    0.11.0    |
|    14.6.0    |   12.14.1    |    0.12.1    |    0.9.12    |
|    14.5.0    |   12.14.0    |    0.12.0    |    0.9.11    |
|    14.4.0    |   12.13.1    |   0.10.48    |    0.9.10    |

This is a partial list. For a complete list, visit https://nodejs.org/download/rel›
›ease
```

Node.js 最新 LTS 版をインストール

それでは、最新の LTS 版をインストールします。インストールは、Mac、Windows とも、「nvm install xx.yy.zz(インストールするバージョン番号)」でインストールできます。

▼ Mac OSX

```
> nvm install v14.15.2
Downloading and installing node v14.15.2...
Downloading https://nodejs.org/dist/v14.15.2/node-v14.15.2-darwin-x64.tar.xz...
##################################################################################›
›############################## 100.0%
Computing checksum with shasum -a 256
Checksums matched!
Now using node v14.15.2 (npm v6.14.9)
```

▼ Window

```
PS C:\Users\inabakazuya> nvm install v14.15.2
Downloading Node.js version 14.15.2 (64-bit)...
Complete
Creating C:\Users\inabakazuya\AppData\Roaming\nvm\temp

Downloading npm version 6.14.9... Complete
Installing npm v6.14.9...

Installation complete. If you want to use this version, type

nvm use 14.15.2
```

インストールされている Node.js のバージョンの確認

インストールされている Node.js のバージョンは、「nvm ls」で表示させ確認できます。

▼ Mac OSX

```
$ > nvm ls
          v8.17.0
         v12.19.0
         v14.15.0
  ->     v14.15.2
           system
  default -> v12.18.3
  iojs -> N/A (default)
  unstable -> N/A (default)
  node -> stable (-> v14.15.2) (default)
  stable -> 14.15 (-> v14.15.2) (default)
  lts/* -> lts/fermium (-> v14.15.2)
  lts/argon -> v4.9.1 (-> N/A)
  lts/boron -> v6.17.1 (-> N/A)
  lts/carbon -> v8.17.0
  lts/dubnium -> v10.23.0 (-> N/A)
  lts/erbium -> v12.20.0 (-> N/A)
  lts/fermium -> v14.15.2
```

▼ Windows

```
PS C:\Users\inabakazuya> nvm ls

    14.15.2
    14.15.1
    12.13.1
  * 8.16.1 (Currently using 64-bit executable)
```

使用する Node.js のバージョン切り替え

先ほどの「インストールされている Node.js のバージョン表示」で、現在使われている

Node.js のバージョンも表示されています。使用する Node.js のバージョンを変更する場合には、「nvm use xx.yy.xx(使用するバージョン番号)」で切り替えます。

▼Node.js のバージョン確認と切替

```
$ > node -v
v14.15.2
[~]
$ > nvm use v12.18.3
Now using node v12.18.3 (npm v6.14.6)
[~]
$ > node -v
v12.18.3
```

以上で、複数のバージョンの Node.js を切り替えて使える環境が構築できました。

1.2 Microsoft Visual Studio Code + 拡張機能

Microsoft 社が無料で提供している「テキストエディタ」です。Electron:https://www.electronjs.org/ をベースにしたオープンソースで開発されています。

Electron は、GitHub 社が「Atom(テキストエディタ)」を開発するために構築したフレームワークで、HTML・CSS・JavaScript を使用して、Windows、Mac、Linux のマルチプラットフォームで動作するアプリケーションを開発できます。

Visual Studio Code(以後は、VSCode と表記します。) は、コードを記述する「テキストエディタ」として非常に優秀ですが、拡張機能 (Google Chrome や Firefox などのブラウザ同様に拡張機能が多くの開発者により公開されています。) を追加することで JavaScript 以外の言語 (C#、python など) でも使えます。

デバッグなども行えるため、Web 開発では、事実上の標準と言っても良いでしょう。有料では、Jetbrains 社:https://www.jetbrains.com/ の Webstorm がありますが、今回は、無償の VSCode を使用します。

♡| 1.2.1　VSCode のインストール

VSCode のインストールは
本家サイト https://code.visualstudio.com/
から、ダウンロード後インストールしてください。
または、以下の方法でもインストールできます。

Windows

パッケージマネージャー「Chocolatey」をお使いの方は、

▼ Chocolatey

```
choco install vscode
```

にて、インストールできます。
パッケージマネージャー「winget」をお使いの方は、

▼ winget

```
winget install -e --id Microsoft.VisualStudioCode
```

にて、インストールできます。

macOS

パッケージマネージャーに「brew」をお使いの方は、

▼ Homebrew

```
$ > brew update
$ > brew cask install visual-studio-code
```

にて、インストールできます。

♡| 1.2.2　VSCode の拡張機能

VSCode は、プラグイン形式で拡張機能を追加できます。
React、Redux を使用したプロジェクトでは、以下をインストールすると便利です。
VSCode を起動し、左ツールバーの拡張機能アイコンをクリックします。

▲図 1.8: VSCode の拡張機能

ここの検索窓に拡張機能の名前、キーワードを入力して検索します。

Debugger for Chrome

デバッグの際に、PC にインストールされている Chrome を自動で起動してくれます。Chrome の DevTools は、非常に強力です。また、Chrome にも React、Redux 用の拡張機能を追加すると更に便利になります。

▲図 1.9: Beautify

♡| 1.2.3　Eslint

コード記法の間違いを指摘・修正してくれます。

▲図 1.10: Beautify

♡| 1.2.4　Prettier

Eslint と同じように、コード記法の間違いの指摘・修正やコードフォーマットを行います。Eslint を合わせて使うと最強です。

▲図 1.11: Beautify

1.3 Google Chrome + 拡張機能

ご存じ Google 社が提供するブラウザです。
PC へインストールされていない方は、

♡| 1.3.1 Google Chrome のインストール

Google Chrome:https://www.google.com/intl/ja/chrome/

▲図 1.12: Google Chrome

===　Google Chrome の拡張機能こちらも、VSCode と同様に拡張機能を追加することで、さらに便利に使うことができます。

React、Redux の開発では、以下の拡張機能は必須と言っても良いほどです。

拡張機能のインストールは、Chrome Web store で検索してください。

Chrome Web store:https://chrome.google.com/webstore/category/extensions?hl=ja

▲図 1.13: Chrome Web store

React Developer Tools

React を使用して作成したページは、最終的にはページ出力用 JavaScript に変換され、ブラウザで表示されるときには HTML として出力されます。この拡張機能を使うと、Google Chrome の DevTools に Components タブが作成され、Props、State を確認できます。

React Developer Tools

chrome ウェブストア
ホーム > 拡張機能 > React Developer Tools
React Developer Tools
提供元: Facebook
1,299
デベロッパー ツール
ユーザー数: 2,000,000+ 人
Chrome から削除します
概要
レビュー
サポート
関連アイテム
todos
Elements
Sources
Console
Components
Profiler
Performance
Network
Search (text or /regex/)
App
Header
TodoTextInput
MainSection
TodoList
TodoItem key="0"
Footer
Link
Link
Link
TodoTextInput
props
newTodo:
onSave: onSave()
placeholder: What needs to be done?
hooks
State: Try React DevTools
rendered by
Header
App

▲図 1.14: React DevTools で App を表示

Redux DevTools

のちほど、Redux の章であらためて説明しますが、「タイムトラベルデバッグ (実行されたアクションをさかのぼる)」が簡単にできます。また、実行されたアクション、変更された State が「新」「旧」とあり、どの部分が変更されたのかも確かめるのも簡単です。

Redxu DevTools

chrome ウェブストア
ホーム > 拡張機能 > Redux DevTools
Redux DevTools
提供元: remotedevio
Chrome から削除します
531 | デベロッパー ツール | ユーザー数: 1,000,000+ 人
概要
レビュー
サポート
関連アイテム

1.4 第 1 章のまとめ

React、Redux の開発環境は、できましたでしょうか？

- nvm
- node
- VSCode + 拡張機能
- Google Chrome + 拡張機能

のインストールを完了してください。

第 2 章
スタートプロジェクトの作成

本章では、「create-react-app」と言う、react アプリケーションのひな型がコマンドひとつで作成できる優れものを使用して、スタート用のアプリケーションを作成し、ブラウザで表示するまでを行います。

また、作成するプロジェクトは、TypesScript を使用します。コード記法の指摘・修正を行えるよう「eslint」、「prettier」の設定も行います。

2.1 create-react-app コマンド

React アプリケーションをゼロから作成するためには、

* 「node プロジェクト」に必要な package.json を作成* react など必要なライブラリのインストール* 作成したアプリケーションが、古いブラウザでも実行できるようにコードを変換 (Babel 使用) * 出力するファイルをまとめる (バンドルする - webpack 使用)

など、react ライブラリのインストール以外にも、Babel や webpack をインストールして設定ファイルを作成しなくてはなりません。

また、使用するライブラリによっては、Babel のプラグインのインストールや設定など、アプリケーションのコードを書き始める前の作業がたいへんです。

しかし、「そんなメンドウなことは、やってられない。」と誰しもが思ったか、すぐにでもコードを書き始めることのできるスタート用アプリケーションが、react 開発元の Facebook から提供されています。

さらに、そのスタート用アプリケーションは、コマンド一発でインストールできます。

ターミナルを起動し、プロジェクトフォルダを作成するフォルダへ移動します。

▼create-react-app でスタート用アプリケーション作成

```
$ > npx create react-app プロジェクト名 --template typescript
```

で、「プロジェクト名」のフォルダが作成され、スグにでも開発に取りかかれます。

```
Success! Created yourproject at yourproject_path
Inside that directory, you can run several commands:

  yarn start
    Starts the development server.

  yarn build
    Bundles the app into static files for production.

  yarn test
    Starts the test runner.

  yarn eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

  cd yourproject
  yarn start

Happy hacking!
```

で、プロジェクト作成が完了します。

コンソールには、Facebook が関わっているノード・パッケージマネージャーの「yarn」を使ったコマンドが表示されています。

yarn start
: 開発用サーバの開始。

yarn build
: 製品用に静的はファイルにアプリケーションをまとめる。

yarn test
: テストランナーの開始。

yarn eject
: ツール (create-react-app) を取り除き、依存関係、設定ファイル、スクリプトを app ディレクトリにコピーする。

と、あります。

yarn は、pnp(プラグ＆プレイ-依存関係 (node_modules フォルダ以下にインストールされるパッケージ) を仮想化してロードする機能) を導入した v2 で大きく変わっています。今では v3 もリリースされています。

PnP なしでも yarn v3 を使うこともできるようですが、私は npm(ノード・パッケージマネージャー) を使っています。

github

ここまでの作業は、GitHub にあります。<!-- textlint-disable -->

▼ GitHub から

```
    $ > git clone -b 00_create-react-app https://github.com/yaruo-rea›
›ct-redux/yaruo-blog.git
```

<!-- textlit-enable -->

2.2 アプリケーションを実行

アプリケーションが作成できましたので、実行してみます。

ターミナルに表示されているように、プロジェクトフォルダへ移動し、スタート用のコマンドを入力します。

```
$ > cd プロジェクト名
$ > npm start
```

すると、webpack に同梱されている開発用の web server が起動し、デフォルトでは、port:3000 でアプリケーションへアクセスできます。

```
Compiled successfully!

You can now view yourproject in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://pcのローカルIPアドレス:3000

Note that the development build is not optimized.
To create a production build, use yarn build.
```

Google Chrome が起動し、http://localhost:3000 へアクセスし以下のページが表示されます。

▲図 2.1: create-react-app の画面

このページが表示されれば成功です。

2.3 create-react-app で作成された中身

create-react-app で作成された中身は、以下となります (使用するテンプレートにより作成されるファイル・フォルダは異なる)。

▼ package.json

```
.
├── node_modules
├── README.md
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── App.css
│   ├── App.test.tsx
│   ├── App.tsx
│   ├── index.css
│   ├── index.tsx
│   ├── logo.svg
│   ├── react-app-env.d.ts
│   ├── reportWebVitals.ts
│   └── setupTests.ts
├── tsconfig.json
└── yarn.lock
```

package.json ファイルは、Node.js を使用するプロジェクトの設計図にあたるものです。Node.js を使うプロジェクトを開始する場合には、プロジェクトフォルダで「npm init」を行うと対話形式で「package.json」を作成しますが、create-react-app コマンドを使用すると、package.json も以下のように作成されます。

▼ package.json

```
{
  "name": "yaruo-blog",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "@types/jest": "^26.0.15",
```

```
      "@types/node": "^12.0.0",
      "@types/react": "^17.0.0",
      "@types/react-dom": "^17.0.0",
      "react": "^17.0.2",
      "react-dom": "^17.0.2",
      "react-scripts": "4.0.3",
      "typescript": "^4.1.2",
      "web-vitals": "^1.0.1"
    },
    "scripts": {
      "start": "react-scripts start",
      "build": "react-scripts build",
      "test": "react-scripts test",
      "eject": "react-scripts eject"
    },
    "eslintConfig": {
      "extends": [
        "react-app",
        "react-app/jest"
      ]
    },
    "browserslist": {
      "production": [
        ">0.2%",
        "not dead",
        "not op_mini all"
      ],
      "development": [
        "last 1 chrome version",
        "last 1 firefox version",
        "last 1 safari version"
      ]
    }
  }
```

package.json 内にある「scripts」にあるものがコマンドになります。react-scripts は、npm スクリプトを連続、または、並列に実行してくれるものです。

package.json の「dependencies」には、実行に必要でインストール済みの npm パッケージが記載されています。必要な npm パッケージをインストールすると、ここに自動的に追記されます。
また、開発時のみ必要なパッケージ (build したときには組み込まれない) は、「devDependencies」に追加されます。

2.4 eslint、prettier

「lint」は、C 言語用のコンパイラよりも詳細で厳密なチェックを行うプログラムです。コンパイル前にコードをチェックするために使われます。

それが、いつしかコードをチェック・解析することを「lint」、lint を行うプログラムを linter と呼ぶようになったそうです。

JavaScript(ECMAScript) 用の linter が、「eslint」になります。もちろん、Java、HTML、Python などにも linter があります。

「eslint」は、JavaScript で指定されたルールと違うコードの書き方をしている部分を指摘してくれます。その指定されたルールとは、たいていの場合には JavaScript に詳しい人達が決めたもので、良く使われるものは、かの有名な AirBnB の開発チームのものです。

もちろん、ルールは改変・追加もできます。

チェックしてくれるのは、たとえば、

- const で宣言している変数への代入
- 未定義の変数やモジュールの使用
- 分割代入の使用を推奨

などがありますが、何をチェックし指摘するのかは、チーム毎、プロジェクト毎に自由に決めることができます。

「prettier」は、コードを整形 (インデント、改行など) してくれるツールです。実は、eslint でもコード整形はできるのですが、コード整形は prettier の方が優れいます。

そのために、

- コードチェックは、eslint
- コード整形は、prettier

と、得意なものに任せます。

この便利な機能こそが、第 1 章で紹介した

- Eslint
- Prettier

になります。

♡| 2.4.1　eslint、prettier のインストール

eslint のパッケージ追加と設定

create-react-app を使用して作成したスタートアッププロジェクトには、eslint は導入済みですので設定し直し、必要な関連パッケージをインストールします。

ターミナルに以下のように「eslint --init」と初期化コマンドを入力します。

▼eslint の初期化

```
$ npx eslint --init
```

「?」が行頭にある質問と選択枝が表示されますので、カーソルキーで選択枝を選びエンターキーで次ぎの質問に移ります。

▼eslint の質問に答える

```
? How would you like to use ESLint? …
❯ To check syntax only                    ←選択したものに　> が表示される
  To check syntax and find problems
  To check syntax, find problems, and enforce code style
```

最後の質問に答えると必要なパッケージをインストールするか尋ねられますので「Yes」と答えてます。

▼eslint への答え

```
❯ How would you like to use ESLint? · problems
❯ What type of modules does your project use? · esm
❯ Which framework does your project use? · react
❯ Does your project use TypeScript? · No / Yes        ← Yes を選択
❯ Where does your code run? · browser
❯ What format do you want your config file to be in? · JavaScript
Local ESLint installation not found.
The config that you've selected requires the following dependencies:

eslint-plugin-react@latest @typescript-eslint/eslint-plugin@latest @typescript-e›
›slint/parser@latest eslint@latest
❯ Would you like to install them now with npm? · No / Yes
```

▼package.json に eslint 関連のパッケージがインストールされました。

```
"devDependencies": {
  "@typescript-eslint/eslint-plugin": "^5.4.0",
  "@typescript-eslint/parser": "^5.4.0",
  "eslint": "^8.2.0",
  "eslint-plugin-react": "^7.27.0"
}
```

また、eslint の設定ファイル「.eslintrc.js」が作成されています。

▼ .eslint.js

```
module.exports = {
    "env": {
        "browser": true,
        "es2021": true
    },
    "extends": [
      "eslint:recommended"
      "plugin:react/recommended",
      "plugin:@typescript-eslint/recommended",
    ],
    "parser": "@typescript-eslint/parser",
    "parserOptions": {
        "ecmaFeatures": {
            "jsx": true
        },
        "ecmaVersion": 12,
        "sourceType": "module"
    },
    "plugins": [
        "react",
        "@typescript-eslint"
    ],
    "rules": {
    }
};
```

設定ファイル「.eslintrc.js」で、どのようなルールが適用されるのかを確認します。適用されるルールが、「current_rules.txt」に書き出されます。

▼ eslint 設定で適用されるルール

```
$ npx eslint --print-config .eslintrc.js > current_rules.txt
```

eslint で使用するルールは一般的なものをベースにしたいので、あの airbnb のルールをインストールします。

▼ airbnb のルーツのインストール

```
$  npx install-peerdeps --dev eslint-config-airbnb
   install-peerdeps v3.0.3
   It seems as if you are using Yarn.
   Would you like to use Yarn for the installation? (y/n) n← yarn を使っているのか
聞かれるので、no である「n」を入力
```

airbnb のルールをインストールしたので、設定ファイルに追加します。

▼ .eslintrc.js へ airbnb ルールを適用

```
  "extends": [
      "eslint:recommended",
      "plugin:react/recommended",
      "airbnb",        ← airbnb のルール
      "airbnb/hooks", ← airbnb の React hooks のルール
      "plugin:@typescript-eslint/recommended",
  ],
```

再度、ルールを出力すると適用されるルールがずいぶん増えているのが分かります。

次に、TypeScript もチェックできるようにルールを追加します。「plugin:」の下 3 行を追加しました。

▼ .eslintrc.js の extends 部分

```
  "extends": [
      "eslint:recommended",
      "plugin:react/recommended",
      "airbnb",
      "airbnb/hooks",
      "plugin:@typescript-eslint/recommended",
      "plugin:@typescript-eslint/recommended-requiring-type-checking",
      "plugin:import/recommended",
      "plugin:import/typescript",
  ],
```

TypeScript 用ルールを追加しましたので、「parserOptions」を以下のように変更する。

▼ .eslintrc.js の parserOptions 部分

```
  "parserOptions": {
    "ecmaFeatures": {
        "jsx": true
    },
    "ecmaVersion": 12,
    "sourceType": "module",
    "tsconfigRootDir": __dirname,
    "project": ["./tsconfig.json"],
  },
```

これでルールの適用は完了しましたが、都合の悪いルールには設定ファイルでルールの上書をします。

▼ .eslintrc.js の rules へ追加

```
"rules": {
    "import/extensions": [
        "error",
        {
          js: "never",
          jsx: "never",
          ts: "never",
          tsx: "never",
        },
      ],
      "react/jsx-filename-extension": [
        "error",
        {
          extensions: [".jsx", ".tsx"],
        },
      ],
      "react/react-in-jsx-scope": "off",
      "no-void": [
        "error",
        {
          allowAsStatement: true,
        },
      ],
}
```

Prettier のインストールと設定

ここからは、Prettier のインストールと設定する。

▼ Prettier のインストール

```
$ npm install -D prettier eslint-config-prettier
```

インストールが完了すると、package.json に追加されます。

▼ package.json

```
"devDependencies": {
  "@typescript-eslint/eslint-plugin": "^5.4.0",
  "@typescript-eslint/parser": "^5.4.0",
  "eslint": "^8.2.0",
  "eslint-config-airbnb": "^19.0.0",
  "eslint-config-prettier": "^8.3.0",
  "eslint-plugin-import": "^2.25.3",
  "eslint-plugin-jsx-a11y": "^6.5.1",
  "eslint-plugin-react": "^7.27.0",
  "eslint-plugin-react-hooks": "^4.3.0",
  "prettier": "^2.4.1"
```

```
}
```

Pretter のチェックを「.eslintrc.js」へ追加します。

▼ .eslintrc.js

```
"extends": [
    "plugin:react/recommended",
    "airbnb",
    "airbnb/hooks",
    "plugin:@typescript-eslint/recommended",
    "plugin:@typescript-eslint/recommended-requiring-type-checking",
    "plugin:import/recommended",
    "plugin:import/typescript",
    "prettier",      ← prettier を追加
],
```

pritter の設定ファイル「.prettierrc」を追加します。設定可能なオプションは、Prettier オプション[*1]で確認できます。

ほぼすべてがデフォルトでも良いのですが、create-react-app がシングルクオートなので設定します。

▼ .prettierrc

```
{
  "singleQuote": true
}
```

eslint と prettier が衝突すると検出・修正ループに入りますので、チェックします。

▼ eslint、prettier の衝突検出

```
$ npx eslint-config-prettier 'src/**/*.{js,jsx,ts,tsx}'
  No rules that are unnecessary or conflict with Prettier were found.
```

無事に衝突なしとなりました。

package.json にスクリプトコマンドを追加します。

▼ package.json

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
```

[*1] https://prettier.io/docs/en/options.html

```
    "lint": "eslint 'src/**/*.{js,jsx,ts,tsx}'",              ← lint:チェック
    "fix": "npm run format && npm run lint:fix",              ← fix:整形してチェック
して自動修復
    "format": "prettier --write 'src/**/*.{js,jsx,ts,tsx}'", ← format:整形
    "lint:fix": "eslint --fix 'src/**/*.{js,jsx,ts,tsx}'",    ← lint:fix チェックし
て修復
    "eject": "react-scripts eject"
  },
```

Eslint、Prettier の設定が完了しましたので、src フォルダにある「App.tsx」を開いてみると、ルールから外れるものは指摘されています。

▲図 2.2: Eslint、Prettier に怒られてます

2.5　eslint、prettier の指摘を修正

ESlint、Prettier は指摘するだけではなく、修正案の提示・修正 (できるものだけですが...) までしてくれます。

VSCode に Prettier 拡張機能を追加してあれば、以下のように、VSCode 側で設定すると、ファイルを保存する度に自動で修正をいれることもできます。

私は、修正を自分のタイミングで行いたいので VSCode 側の設定は行っていません。

もし、VSCode 側の設定をする場合には、VSCode で [File]->[Preferences]->[Settings] にて、以下の各項目を検索して設定するか、settings.json へ追加するか、このプロジェクト用には、プロジェクトフォルダ直下に「.vscode」フォルダを作成し、「settings.json」ファイルへ書き込みます。

ユーザー設定ファイルの内容が、この設定で上書きされます。

▼ VSCode の設定

```
"editor.formatOnSave": true,
"[JavaScript]": {
  "editor.formatOnSave": false
},
"[JavaScriptreact]": {
  "editor.formatOnSave": false
},
"[typescript]": {
  "editor.formatOnSave": false
},
"[typescriptreact]": {
  "editor.formatOnSave": false
},
"editor.codeActionsOnSave": {
    "source.fixAll": true,
    "source.fixAll.eslint": false
},
"prettier.disableLanguages": ["JavaScript", "JavaScriptreact", "typescript", "type›
›scriptreact"],
```

VSCode 上で、

- 赤波線で指摘されている
- 問題タブに表示されている

ものを修正します。

まずは、.eslintrc.js 自体に問題があるようです。

赤波線の上にマウスポンタを置くと eslint のコード、この場合は「no-use-before-define」が表示されます。、さらに、「コマンドキー (Windows では、ctrl)　＋ ピリオド」を押すと、修正方法が提示されます。

.eslintrc.js ファイルでの指摘は、以下の通り「es6 モジュールの書き方へ移行しろ！」と怒られています。以下のように、.eslintrc.js を変更します。

▼ .eslintrc.js

```
const config = {
    env: {
        browser: true,
        es2021: true
    },
    extends: [
        "plugin:react/recommended",
        "airbnb",
        "airbnb/hooks",
        "plugin:@typescript-eslint/recommended",
        "plugin:@typescript-eslint/recommended-requiring-type-checking",
        "plugin:import/recommended",
        "plugin:import/typescript",
        "prettier",
    ],
    parser: "@typescript-eslint/parser",
    parserOptions: {
        ecmaFeatures: {
            jsx: true
        },
        ecmaVersion: 12,
        sourceType: "module",
        tsconfigRootDir: __dirname,
        project: ["./tsconfig.json"],
    },
    plugins: [
        "react",
        "@typescript-eslint",
    ],
    rules: {
        "import/extensions": [
        "error",
        {
            js: "never",
            jsx: "never",
            ts: "never",
            tsx: "never",
        },
        ],
        "react/jsx-filename-extension": [
```

```
        "error",
        {
            extensions: [".jsx", ".tsx"],
        },
        ],
        "react/react-in-jsx-scope": "off",
        "no-void": [
        "error",
        {
            allowAsStatement: true,
        },
        ],
    }
};

module.exports = config;
```

このように修正して保存すると、次の指摘がきます。

> Parsing error: "parserOptions.project" has been set for @typescript-eslint/parser. The file does not match your project config: .eslintrc.js. The file must be included in at least one of the projects provided.

これは、ファイルが「どこからも import されていない」場合に表示されるエラーです。「.eslintrc.js」は、ESLint の設定ファイルですので、どこからもインポートされていません。

解消法は、「npx elsint --init」でファイルを作成した際に json 形式、または、yaml(yml) 形式を選択するかまたは、.eslintrc.js ファイル自体をチェックの対象から除外します。

今回は、JavaScript 形式で作成したのでチェック除外のための、「.eslintignore」ファイルをプロジェクトフォルダ直下に作成し、lint.js や config.js のパターンが含まれるファイル、パッケージがインストールされる node_modules フォルダなどを除外するように指定します。

▼ .eslintignore

```
build/
public/
**/node_modules/
*.config.js
.*lintrc.js
```

これで、.eslintrc.js については怒られなくなりました。

次に、App.tsx ファイルを修正します。

▲図 2.3: App.tsx の修正

筆者が VSCode を日本語化していないのは、エラーメッセージでググる場合を考えてのことです。英語でのエラーメッセージの方が的確なページをみつけやすいと考えています。

では、指摘されている点を修正していきます。

「react/function-component-definition」は、関数コンポーネントに一貫した関数タイプを適用しなさいと怒られています。

関数をアロー関数に直し、関数型の宣言も追加します。

▼ App.tsx

```
// React17からは、JSXでReactのインポートが不要になりましたので、以下の行を削除します。
import React from 'react';
```

▼ App.tsx

```
import { VFC } from 'react';
import logo from './logo.svg';
import './App.css';

const App: VFC = () => (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <p>
        Edit <code>src/App.tsx</code> and save to reload.
      </p>
      <a
        className="App-link"
        href="https://reactjs.org"
        target="_blank"
        rel="noopener noreferrer"
      >
        Learn React
      </a>
    </header>
  </div>
);

export default App;
```

最後に「reportWebVitals.ts」を修正します。

これは、Quick Fix を選択すると、「Add void operator to ignore」と出ますので、それを選択します。

▼ reportWebVitals.ts

```
import { ReportHandler } from 'web-vitals';

const reportWebVitals = (onPerfEntry?: ReportHandler) => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    void import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB })
=> {
```

▲図 2.4: reportWebVitals.ts の修正

```
    getCLS(onPerfEntry);
    getFID(onPerfEntry);
    getFCP(onPerfEntry);
    getLCP(onPerfEntry);
    getTTFB(onPerfEntry);
  });
```

```
  }
};

export default reportWebVitals;
```

これで現時点での指摘はすべて修正できました。

▲図 2.5: すべての問題の修正完了

2.6 第 2 章のまとめ

React を使用したアプリケーションは、スタートアップ用のアプリケーションがコマンド一発でインストールできます。

より良いコーディングをするためにも、ESlint、Prettier を導入しましょう。

ここまでの内容は、GitHub 上で、以下のコマンドでクローンできます。<!-- textlint-disable -->

▼ GitHub

```
$ > git clone -b 01_eslint_prettier https://github.com/tmkkz/yaruo.
›git
```

<!-- textlint-enable -->

第 3 章

Todo リストの作成 (React のみ)

この章では、第 2 章で作成したスタートアップ用のアプリケーションを魔改造し、Redux:https://redux.js.org/tutorials/essentials/part-1-overview-concepts のチュートリアルにある Todo リストを作成します。

していきます。

3.1 React とは？

3.2 表示するデータの型を決める

3.3 データ表示画面

3.4 React hooks を使用して、データの追加・編集・削除

やる夫で学ぶ「react-redux」

redux-toolkit で、簡単・完璧理解だお･･･

2021 年 6 月 25 日　ver 1.0

著　者　気分はもう

© 2020 気分はもう

(powered by Re:VIEW Starter)

少しは分かったのか？

ヒャヒャヒャ！
何とかなるお・・・

ほんとうは、非同期でのデータの取得や
UI関連の説明もしたかったのじゃが
紙面の関係で次回にする。

―― エッ！

なんてこった・・・！

まだ、まだ学ぶことが
沢山あるのかお！

―― というわけで
ここまで読んでくれて
ありがとうございますお！