

# 目次

---

- テーマ「なぜcreate-react-appを使うのか」
  - 2021/02/02,2021/02/04 react勉強会

## 前編：JavaScriptを取り巻く環境とJSX

- どこまでがJavaScript？
- JavaScriptを変換する仕組み
- モジュールを束ねる仕組み

## 後篇：コード実演+質問

1. 素のhtml+js
2. babelでjsを変換
3. html内にjsxを書く
4. webpackでモジュールをひとまとめに

# [1]どこまでがJavaScript？

---

## あれもJavaScript、これもJavaScript

- ウェブサイト/ウェブアプリ
- 高機能エディター
  - atom/vscode
- チャットツール
  - discord/slack
- スマートフォン向けのアプリ
  - Instagram
- ゲーム開発ソフトやデザインソフトの拡張スクリプト
  - Unity, Adobe Illustrator
- IoTデバイス
  - ラズパイ

## 実行環境が異なる

- ブラウザ
  - JavaScript/HTML/WebGL/WebAssemblyなどを実行
  - vscode/discord/slackは、実はブラウザを内部で保持
- Node.js
  - javascriptを実行できる環境(特にサーバーサイド)
- その他多数のjavascriptを解釈できるエンジン

## JavaScriptの最大の特徴

- 言語が日々進化し、日々定義されている
- 幅広く使われている
  - Web系、特にHTMLやブラウザとは切り離せない
- 動的型付け言語(型宣言はブラウザなどが勝手に行う)
  - 手軽に始められるが、その分バグに気づきにくい

## 要点

- javascriptは始めやすく、極めにくい
- 方言やバージョン差、環境差も激しく、バグも入りやすい

## 参考

- <https://jsprimer.net/basic/introduction/>
  - 基礎から応用まで、JavaScriptを学べる良書

## [2]変換する仕組みについて

---

### 歴史的経緯

- 多様な理由により、変換する仕組みが求められた
  - ブラウザにより実行できるJavaScriptの処理が異なる
  - JavaScriptはブラウザよりも成長が早い
  - 動的型付けは不具合が多い
  - 多様な拡張言語が生まれた
- 拡張言語→JavaScript、またはJavaScript同士の変換を行うように
  - 「トランスパイル」を行うトランスパイラ
  - トランスパイラの1つがBabel

### トランスパイル言語

#### JSX

- JavaScriptを拡張した言語
  - XMLに似た構文で記述できる
  - トランスパイルされる前提でコードの視認性を高める
- エラー防止とUIメンテナンスが簡単になる
- Reactでは利用を推奨している

#### TypeScript(重要度:低)

- JavaScriptを拡張した言語(今は触れない)
  - JavaScriptに強力な型補完を行ったようなもの
- 型補完により開発が効率化され、エラーも防止される
- TypeScript版JSXの「TSX」も存在する
- <https://typescript-jp.gitbook.io/deep-dive/recap>
  - JavaScriptとTypeScriptの関係をうまく示した図が分かりやすい

## JSXの使い方(p37-)

1. スcopeにはReactが必要
2. {}を使って式を埋め込む
3. JSX要素も式として扱う
  - 例えば、関数の戻り値をJSXを代入する
  - 例えば、変数にJSX要素を代入する
4. 属性の指定は原則HTMLと同じ。ただし一部の属性名はHTMLと異なる
  - class, className
  - for, htmlFor
  - onclick, onClick
  - tabIndex, tabIndex
5. HTML同様、入れ子にできる
6. 要素は必ず閉じる
  - `<要素名></要素名>`または`<要素名/>`
7. 必要に応じてReact.createElementも使える
  - `React.createElement("要素名",{属性名:"属性の値"},子要素やテキスト)`

### JSXをもっと知るには

- <https://ja.reactjs.org/docs/jsx-in-depth.html>
  - JSXを深く知る為のReact発展チュートリアル

## Babelについて

- トランパイラの1つ
  - JSXやTypeScriptもトランスパイル可能
- <https://babeljs.io/repl/>
  - 変換ツールを即座に試せるインタプリタ(READ-EVAL-PRINT LOOP)
- プラグインにより、オリジナルの変換も行うことができる

## [3]モジュールを束ねる仕組み

---

- 大量のモジュールをJavaScriptでは利用しており、束ねる必要がある

### モジュールの歴史的経緯(重要度:低)

#### 概要

- 元々モジュールを読み込む仕組みがjavascriptに用意されていなかった
- AMD/RequireJS
  - `define(***,***)`
  - 現在の主流から外れる
- CommonJS
  - `require(***)`
  - nodejsで用いられている
- ES2015
  - `import *** from ***`
  - 欠陥を補うべく、ES2015(ES6)から登場した
- HTML内では下記のような表記
  - `<script src="***"></script>`
  - モジュールごとに上記を記述する
- <https://tsuchikazu.net/javascript-module/>
  - あえてES2015登場以前の情報を引用
- <https://jsprimer.net/basic/module/>
  - ES2015のimportについてより詳しく説明

#### 要点

- 基本は`import *** from *** ;`でモジュールを読み込む
- 歴史的経緯から複数の書き方があり、注意が必要

## webpack

- モジュールをひとまとめに束ねる仕組みの1つ
  - 「モジュールバンドラー」
- webpack内部で、事前処理でトランスパイルもすることが多い
  - loaderを使い事前処理を読み込む(babel-loaderなど)

## モジュールを束ねるメリット

- ファイルの追加漏れの解消/いらないファイルを読み込まない
- 圧縮モードにすることで、ファイル圧縮も可能

## 現在主流の開発の流れ

- ローカルにnodejsを用意し、モジュールを読み込んで開発
- リリースなどしたい際、「ビルド」する
  - コードをトランスパイル(babelなど)
  - モジュールをバンドル(webpackなど)
  - 静的なJavaScriptを出力する
- 環境設定が非常に手間であり、create-react-appなどのツールやフレームワークを使うのが主流

## [前編要点]JavaScriptを取り巻く環境とJSX

---

- 言語が日々進化し、さらに手軽に始められる為、広く使われている
  - ただし、素のJavaScriptは扱いにくい側面もある
- 課題を補う為、トランスパイルやモジュールバンドルが行われる
  - これにより、コード不備も減り、高機能なサイトが作れるように
  - ただし、環境構築は非常に手間になってしまった
- よく使う内容は環境設定を行いやすくできるように
  - create-react-app

## 後篇：コード実演+質問

---

1. 素のhtml+js
2. babelでjsを変換
3. html内にjsxを書く
4. webpackでモジュールをひとまとめに