既存のウェブサイトに React を追加する

まずは必要なぶんだけ使ってみましょう。

React は段階的に導入することができるように最初からデザインされています。つまり**最小限の部分で React を利用することも、あるいは大規模に React を利用することも可能です。**既存のページにちょっとしたインタラクティブ性をもたせたいだけでも構いません。React コンポーネントを使えばお手の物です。

多くのウェブサイトはシングルページアプリケーションではありませんし、そうする必要もありません。まずは**たった数行のコード**から、あなたのウェブサイトに React を取り入れてみましょう。**ビルドツールは必要ありません。**そこから徐々に React の使用範囲を広げていくのもいいですし、あるいは少しの動的なウィジェットだけにとどめておくのもいいでしょう。

1分で React を導入する

このセクションでは、既存の HTML ページに React コンポーネントを導入する方法を説明します。以下の部分では自分のウェブサイトを利用して進めてもいいですし、練習用に空の HTML ファイルを用意するのもいいでしょう。

複雑なツール類や事前にインストールしておかなければいけないものはありません。**インターネットへの接続さえあれば、1 分間でこのセクションを終わらせることができます。**

オプション：お手本をダウンロードする (2KB ZIP 圧縮)

ステップ 1：HTML に DOM コンテナを追加する

まずは編集したい HTML ファイルを開きましょう。React で描画したい箇所を決めて、空の `<div>` 要素を追加しましょう。例えばこんな感じです。

```
<!-- ... 既存の HTML ... -->

<div id="like_button_container"></div>

<!-- ... 既存の HTML ... -->
```

ここでは `<div>` 要素にユニークな `id` 属性を指定しています。こうしておけば、後から JavaScript のコードでこの `<div>` 要素を探し出し、この中に React コンポーネントを表示できます。

> **ヒント**
>
> 「コンテナ」としての `<div>` 要素は `<body>` タグの中であれば**どこにでも**置くことができます。また空の `<div>` はひとつのページにひとつだけでも、あるいは必要なだけたくさんあっても大丈夫です。`<div>` 要素は空のことが多いですが、それはたとえ `<div>` の中に他の要素があったとしても、React が結局その中身を置き換えてしまうからです。

ステップ 2：script タグを追加する

次に、同じ HTML ファイルの `</body>` タグの直前に、3 つの `<script>` タグを追加しましょう。

```
  <!-- ... other HTML ... -->

  <!-- Load React. -->
  <!-- Note: when deploying, replace "development.js" with "production.min.js". -->
  <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

  <!-- Load our React component. -->
  <script src="like_button.js"></script>

</body>
```

最初のふたつのタグは React を読み込んでおり、最後のタグはこれから書くコンポーネントのコードを読み込んでいます。

ステップ 3：React コンポーネントを作成する

`like_button.js` という名前の新しいファイルを作成し、HTML ファイルのすぐ隣に置きましょう。

サンプルコードを開いて、自分のファイルにコピーアンドペーストしてください。

> **ヒント**
>
> このコードは `LikeButton` という React コンポーネントを定義しています。まだわからなくても心配しなくて大丈夫です。こういった React の構成要素については、チュートリアル と Hello World のページで後ほど見ていくことにして、まずはサンプルコードを画面に表示させてみましょう！

サンプルコードの末尾に次の 2 行を追加してみましょう。

```
// ... コピーアンドペーストしたサンプルコード ...

const domContainer = document.querySelector('#like_button_container');
ReactDOM.render(e(LikeButton), domContainer);
```

この 2 行のコードは、ステップ 1 で追加した空の `<div>` 要素を見つけてきて、その中に React コンポーネントの「いいね」ボタンを表示します。

これだけです！
ステップ 4 はありません。**これであなたは自分のウェブサイトにはじめての React コンポーネントを導入できました。**
React の導入についてもっと知るには、次のセクションも見てみてください。
完成したソースコードをみる
完成したソースコードをダウンロードする (2KB ZIP 圧縮)

ヒント：コンポーネントを再利用する
React コンポーネントを HTML ページの一箇所だけではなくいろいろな箇所で使いたくなることがあるかもしれません。そこで「いいね」ボタンを 3 回繰り返し表示し、さらにそこにちょっとしたデータを渡すプログラムを用意しました。
ソースコードをみる
ソースコードをダウンロードする (2KB ZIP 圧縮)

> **補足**
> このようなやり方は、主に React を利用する DOM コンテナがページ内でお互いに干渉していない場合において便利な手段です。React 単体のコードとしては、コンポーネントを組み合わせる やり方のほうが手軽です。

ヒント：本番環境用に JavaScript を圧縮する
ウェブサイトを本番環境にデプロイするにあたって、圧縮していない JavaScript はページの速度を著しく落としてしまうということに配慮してください。
自分のスクリプトの圧縮が完了していて、デプロイ後の HTML が `production.min.js` で終わる React スクリプトを読み込んでいることが検証できていれば、**あなたのウェブサイトは本番環境にリリースする準備ができています。**

```
<script src="https://unpkg.com/react@16/umd/react.production.min.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js" crossorigin></script>
```

自分のスクリプトを圧縮することがまだできていないのであれば、例えばこんなやり方があります。

オプション：React で JSX を使う

今までの例では、ブラウザにもともと備わっている機能のみ使ってきました。React コンポーネントを表示するために次のような JavaScript の関数を呼び出していたのはそのためです。

```
const e = React.createElement;

// Display a "Like" <button>
return e(
  'button',
  { onClick: () => this.setState({ liked: true }) },
  'Like'
);
```

ただし、React においては JSX を利用することもできます。

```
// Display a "Like" <button>
return (
  <button onClick={() => this.setState({ liked: true })}>
    Like
  </button>
);
```

これらふたつのスニペットはまったく同じ内容です。**JSX の使用は完全にオプションです**が、React はもちろん他のライブラリで UI を記述する際にも、JSX は多くの人に支持されています。
このコンバーター上で JSX を使って遊んでみてください。

JSX を手軽に試してみる
手っ取り早く JSX を自分のプロジェクトで試してみるには、次の `<script>` タグを追加してみてください。

```
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

この状態で、任意の `<script>` タグに `type="text/babel"` 属性を持たせることで、その `<script>` タグの中では JSX が使えるようになります。サンプル用 HTML ファイルをダウンロードして遊んでみてください。
この方法は学習やシンプルなデモの作成にはいいですが、これをそのまま使うとウェブサイトは重くなってしまい、**本番環境には向きません**。次のレベルに進む準備ができたら、先ほど追加した `<script>` タグと `type="text/babel"` 属性は削除してしまいましょう。そして次のセクションに進み、JSX プリプロセッサを設定して `<script>` タグを自動変換するようにしましょう。

JSX をプロジェクトに追加する

JSX をプロジェクトに追加するためには、バンドルツールや開発用サーバーといった複雑なツールは必要ありません。つまるところ、JSX を追加することは **CSS プリプロセッサを追加することにとてもよく似ています**。唯一必要となるのは、コンピューターに Node.js がインストールされていることだけです。
ターミナルを開き、プロジェクトのディレクトリに移動した上で、次のふたつのコマンドを実行してください。

1. **ステップ 1：** `npm init -y` (うまくいかなければこうやってみてください)
2. **ステップ 2：** `npm install babel-cli@6 babel-preset-react-app@3`

> **ヒント**
> ここでは **JSX プリプロセッサをインストールするためだけに npm を使っています**。それ以外の用途では必要ありません。React のソースコードもアプリケーションコードも引き続き `<script>` タグの中にそのまま書くことができます。

お疲れ様です！ これで**本番環境用の JSX の設定**をプロジェクトに追加することができました。

JSX プリプロセッサを実行する

`src` というディレクトリを作成したうえで、次のコマンドをターミナルから実行してみてください。

```
npx babel --watch src --out-dir . --presets react-app/prod
```

> **補足**
> `npx` はタイプミスではありません。npm 5.2 以上で利用可能なパッケージ実行ツールです。
> 万が一 “You have mistakenly installed the `babel` package” というエラーが表示されたのであれば、JSX をプロジェクトに追加するのステップがうまく実行できていなかったのかもしれません。今いるディレクトリと同じディレクトリで改めて実行してみてください。

このコマンドは JSX を継続的に監視するため、実行が完了するのを待つ必要はありません。
このお手本の JSX コードを参考に `src/like_button.js` というファイルを作成すると、先ほど起動したコマンドがブラウザでの実行に適した `like_button.js` に変換してくれます。JSX ファイルを編集したら、自動的に再変換してくれます。
さらにこの変換コマンドのおかげで、古いブラウザの互換性を気にすることなく、クラス構文といったモダンな JavaScript の構文を使うこともできるようになります。このツールは Babel というもので、もっと詳しく知りたければ公式ドキュメントをご覧になってみてください。
ビルドツールの便利さを体感して、もっとたくさんのことをツールに任せたいと思っていただけたなら、次のセクションではさらにいくつかの人気で扱いやすいツールチェーンを紹介しています。そうでもない場合は… `<script>` タグだけでも十分な機能を果たせます！

このページを編集する