



全4回で学ぶReact.js入門講座 2回目

2025/6/17

Kazuma SEKIGUCHI

前回のアジェンダ

- Reactとは何か
- Reactの環境設定
- JSX記法の使い方
- コンポーネントの使い方
- イベントの扱い方

今回のアジェンダ

- CSSModuleによるCSS適用の方法
- styled componentsの使い方
 - 外部のライブラリ (Emotion)の使い方
- Propsの扱い方
- State (useState) の扱い方

CSSを適用する

- 最終的にはindex.htmlに読み込まれてレンダリングされるため、index.htmlにCSSファイルを読み込ませておけば適用はされる
- Reactはコンポーネントを作成して、組み合わせて1つの仕組みを作ることに意味がある
 - それぞれのコンポーネントだけで完結するようなCSSが使えた方が利便性が高い

ReactでCSSを使う (Style属性)

- Reactでもタグにstyle属性を付けてCSSを適用することは可能
 - 効率が悪いので、全体的に使うのは不適當ではあるが、1カ所だけ変更したい場合などは有効
- style属性の値はオブジェクトとして記述する必要あり
 - JSのオブジェクトになるので {} を記述し、更にオブジェクトとして {} を記述するため、{{}}となる
 - 複数の値を指定したいときはカンマで区切る
 - ハイフンが入るCSSのプロパティはハイフンを取り除いて大文字にする
 - 値はダブルクォーテーションで括る

```
<p style={{ color:"#f90"}}>オレンジ色の文字です</p>
```

```
<p style={{ fontSize:"20px",color:"#09f"}}>青い色の大きな文字です</p>
```

ReactでCSSを使う（オブジェクトの事前定義）

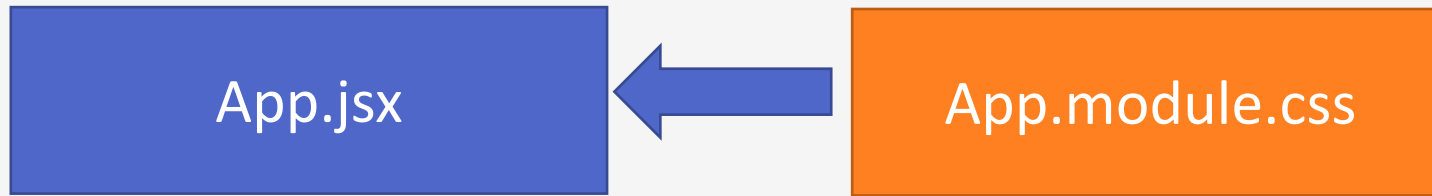
- 事前にオブジェクトを作成しておいてそれを指定することも可能
 - 複数指定することが可能

```
const stylevalue = {  
  fontSize: "40px",  
  color: "#09f"  
};  
<p style={stylevalue}>青い色の大きな文字です</p>
```

ReactでのCSS (CSS Modules)

- 正直style属性を使ってCSSスタイルを適用するのは煩雑
 - 普通のCSS記述が利用できないので、記述量が増える
- 普通のCSSやsassで記述して適用することが可能
 - CSS Modulesを使う
- 通常、コンポーネントごとにCSSファイルを作成して適用する
 - jsxファイルとそれに対応するCSSファイル (sassファイル) を作成する
- ファイル名はjsxの拡張子を除いて、.module.css (sassの場合は.module.scssとなる) とする

ReactでのCSS (CSS Modules)



- jsxファイルで利用したいCSSファイルをimportして利用する
 - importの後は自由な名前を記述可能
 - 利用するときはclassName属性で記述したclassを指定する

```
import classes from "./App.module.css";
export const App = () =>{
  return (
    <div className={classes.container}>
      <p className={classes.textstyle}>スタイルの適用された文字</p>
    </div>
  );
};
```

A code snippet demonstrating the use of CSS Modules in a React component. The code imports a CSS module named 'classes' from './App.module.css'. It then defines a functional component 'App' that returns a JSX element. The JSX element consists of a 'div' with a 'className' attribute set to 'classes.container', containing a 'p' element with a 'className' attribute set to 'classes.textstyle'. A blue arrow points from the 'classes' variable in the import statement to the 'classes' property access in the JSX attributes, highlighting how the imported classes are used.

ReactでのCSS (CSS Modules)

- そのままCSSとして記述することが可能
 - 読み出して使う場合、使い方が変わるがさほど違和感はないはず
- タグに対してスタイルを記述したときは特に指定無しで適用される
 - classなどに適用するときだけ記述が必要
- 別のコンポーネントに適用するCSSで同一のクラス名が存在しても維持される
 - Reactで適当なプレフィックスを付与して一意性を保つため、上書きされない
 - 上書きされないが、CSS適用の優先順位は影響されるため、絶対性はない

ReactでのCSS (styled components)

- CSS Modulesは特に追加機能無しで利用可能
 - styled componentsは追加のライブラリー
 - 別途導入することで利用可能となる
- styled components
 - スタイルを定義したタグを生成して、それを利用することでスタイルを適用する方法
 - JSファイル内に記述するため、見通しが良くなる
 - タグを独自に作成するため、間違いに気付きやすい

ReactでのCSS (styled components)

- 外部ライブラリーであるため、導入が必要



```
C:\Users\user\desktop\react>npm install styled-components
```

- npmを利用してインストールを行う

```
import styled from "styled-components";  
const StyledDiv = styled.div`  
  color:#f90;  
`;  
export const App = () =>{  
  return (  
    <StyledDiv>スタイルの適用された文字</StyledDiv>  
  );  
};
```

新しいdivタグを作成しているため、styled.divとする
styled.pなども可能

ReactでのCSS(styled components)

- styled componentsのタグ名はHTMLに存在しないものを使う必要がある
 - HTMLタグは小文字だけなので、大文字を含めると被ることを防ぐことができる
 - 先頭を大文字Sにして区別することが多い
- SCSS記法を利用できる

ReactでのCSS(Emotion)

- Emotionは追加のライブラリ
 - 別途導入することで利用可能
 - 高速なスタイル生成と適用が可能
- Emotionの特徴
 - スタイルを定義したタグを生成し、スタイルを適用する方法
 - JSファイル内に記述するため、見通しが良くなる
 - cssプロパティやClassNameを利用可能で、柔軟なスタイリングが可能
 - TypeScriptとの優れた統合
 - パフォーマンスとバンドルサイズに優れる
 - styled-componentsよりも高速で軽量

ReactでのCSS (Emotion)

- 外部ライブラリーであるため、導入が必要



```
C:¥Users¥user¥desktop¥react>npm install @emotion/react @emotion/styled
```

- npmを利用してインストールを行う

reactでemotionを使う時は、
@emotion/reactとstyledをスコープ
付きでインストールする

```
/** @jsxImportSource @emotion/react */  
import styled from '@emotion/styled';  
const StyledDiv = styled.div`  
  color: #f90;  
`;  
export const App = () => {  
  return (  
    <StyledDiv>スタイルの適用された文字</StyledDiv>  
  );  
};
```

Emotionを使うときに記述
cssやstyledを使うときにEmotionの
設定で処理すると指示するもの

ReactでのCSS (Emotion)

- classNameプロパティを利用してスタイル適用が可能

```
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';
const buttonStyle = css`
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  &:hover { background-color: #2980b9; }
`;
export const App = () => {
  return (
    <button css={buttonStyle}>Click Me</button>
  );
};
```


ReactでのCSS (Emotion)

- 複数のスタイルを適用する

```
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';
const baseStyle = css`
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
`;
const primaryStyle = css`
  background-color: #3498db;
  color: white;
  &:hover {
    background-color: #2980b9;
  }
`;
const secondaryStyle = css`
  background-color: #95a5a6;
  color: white;
  &:hover {
    background-color: #7f8c8d;
  }
`;
const App = ({ primary }) => {
  return (
    <button className={[baseStyle, primary ? primaryStyle :
secondaryStyle]}>
      Click Me
    </button>
  );
};
export default App;
```

例:App.jsx

primaryの値を
受け取る

primaryの値でスタイルを切り変える

```
import ReactDOMClient from "react-dom/client";
import App from './App';

export const ParentComponent = () => {
  return (
    <div>
      {/* Primary ボタン */}
      <App primary={true} />

      {/* Secondary ボタン */}
      <App primary={false} />
    </div>
  );
};
```

例:Parent.jsx

```
import ReactDOMClient from "react-dom/client";
import { ParentComponent } from './Parent.jsx';

const container = document.getElementById('root');
const root = createRoot(container);
root.render(<ParentComponent />);
```

例:main.jsx

ReactでのCSS(Emotion)

- Emotionのタグ名はstyled-componentsと同様HTMLに存在しないものを使う必要がある
 - HTMLタグは小文字だけなので、大文字を含めると被ることを防ぐことができる
- cssプロパティやClassNameも利用可能
- SCSS記法やオブジェクトスタイルを利用可能
- TypeScriptとの優れた統合

Props

- コンポーネントに対して、外部から値を与えてコンポーネントの動きを変化させるための仕組み
 - 全てのパターンのコンポーネントを作成しなくて済む
 - 通常は親から子のコンポーネントへ値を渡すときに利用する
 - 子から親への値を渡す場合は、PropsではなくuseStateを使うことが多い
- コンポーネントを呼び出して、それを表示するときに属性として値を与える
 - 呼び出されるコンポーネントでその値を受け取るようにすることで、データを橋渡しすることが可能

```
import { Text } from "../components/Text";
export const App = () =>{
  return (
    <Text color="blue" message="text.jsxから表示しています"></Text>
  );};
```

Props

- 受け取る側は (props) として受け取りをする
 - 使う時は、props.属性として受け取ったデータを使うことが可能

```
export const Text = (props)=>{
  const textStyle = {
    color:props.color,
    fontSize:"#20px"
  }
  return (
    <p style={textStyle}>{props.message}</p>
  );
}
```

Children

- コンポーネントのタグ（みたいなもの）の間に記述したものは、`props.children`として受け取ることが可能
 - 間に書かれているものがHTMLであった場合、そのHTML文章がそのまま渡る

App.jsx

```
import { Text } from "../components/Text";
export const App = () =>{
  return (
    <Text color="blue">text.jsxから表示
    しています</Text>
  );};
```

Text.jsx

```
export const Text = (props)=>{
  const textStyle = {
    color:props.color,
    fontSize:"20px"
  }
  return (
    <p
    style={textStyle}>{props.children}</p>
  );
}
```

Propsの省略

- Propsは分割投入が可能

プロパティ名が同一になるので
更に省略することが可能

```
export const Text = (props)=>{  
  const { color , children } = props;  
  const textStyle = {  
    color:color,  
    fontSize:"20px"  
  }  
}
```

```
export const Text = (props)=>{  
  const { color , children } = props;  
  const textStyle = {  
    color,  
    fontSize:"20px"  
  }  
}
```

- 最初から分割して受け取ることも可能

```
export const Text = ({ color , children })=>{  
  const textStyle = {  
    color:color,  
    fontSize:"20px"  
  }  
}
```

State(useState)

- Reactの場合、画面に表示するデータ、可変の状態を全てStateとして管理する
 - コンポーネントの状態を示す値
 - 状態を管理して、イベント実行時などに値を変更することでアプリケーションを実現する
- useState関数を利用して管理を行う
 - importする必要がある

```
import { useState } from "react";
```


useStateの利用

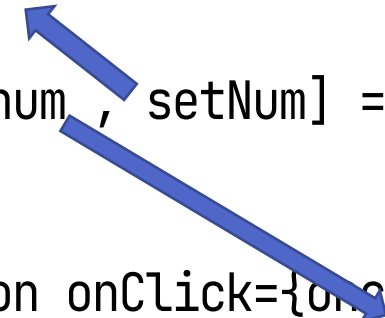
- useStateは配列の形で1つめにStateの変数、2つ目にその変数を更新するための関数を設定する

```
const [num , setNum] = useState();
```

- 変数名を付けて、暗黙的に更新側はset変数名とすることが多い
 - 変数の初期値はundefinedになるので、useState()の引数に値を指定すれば、初期化される

useStateの利用

```
import { useState } from "react";
export const App = ()=>{
  const onClickButton = ()=>{
    setNum(num + 1);
  }
  const [num, setNum] = useState(0);
  return (
    <>
      <button onClick={onClickButton}>クリック</button>
      <p>クリックした回数:{num}</p>
    </>
  );
}
```

A blue arrow points from the 'setNum' property in the 'useState(0)' call to the 'setNum' function call inside the 'onClickButton' function. Another blue arrow points from the 'onClickButton' function call to the 'onClick={onClickButton}' prop of the button element in the JSX.

- ボタンをクリックする度に
onClickButtonが動く
 - setNum() でnum値を増やしている
 - 動的に変更した値を表示

useState内の関数で更新

```
const onClickButton = ()=>{  
    setNum((prev) => prev + 1);  
}
```

- setNum内で関数を定義してしまう
 - 関数の引数に前のstate値が入ってくる
 - prevには実行される前の値が入ってくる
 - prevに1をプラスして戻す

前回

- 参加された方はフォルダーが残っていればそのまま実践可能です
- 参加されていない方は以下の手順をおこなってください
 1. デスクトップ等に「reactapp」というフォルダーを作成
 2. ターミナルを起動
 - Windowsならスタートボタンからターミナル
 - macならFinder→アプリケーション→ユーティリティからターミナル
 3. `cd`を入力し、半角スペースを入れて、作成したフォルダーをドラッグアンドドロップする
 4. `npm create vite@latest app -- --template react`を入力して実行
 5. インストールが終わると、必要なコマンドが出てくるはず
 - `cd app`
 - `npm install`
 - `npm run dev`の順番でコマンドを実行するURLが表示されるので、ブラウザでURLにアクセスすればOK

手順

- App.module.cssを作成
 - CSSを記述
 - App.jsxでimportを使いApp.module.cssを読み込む
 - classNameを利用してクラスを適用する
 - main.jsxを書き換え
 - 表示を確認
- styled componentsをインストール
 - App2.jsxを記述
 - import styled from "styled-components";として読み込み
 - 以下を記述し、main.jsxを書き換え

```
const StyledDiv = styled.div`
  color:#0F9;
`;
export const Text = () =>{
  return (
    <StyledDiv>スタイルの適用された文字</StyledDiv>
  );
};
```

ターミナルなどできちんと作業している
フォルダーに移動しておく
cdコマンドで移動が可能

reactを実行して確認するときはcdコマンドで移動した上で、
> npm run dev
で実行が可能

インストール前にはnpm startを停止する
必要があるので、ctrl + cキー(macなら
Control + cキー)で停止させること

手順

• Emotionをインストール

- App3.jsxを記述
 - main.jsxを書き換え
- App4.jsxを記述
 - main.jsxを書き換え
- App5.jsxを記述
 - Parent.jsxを記述
 - main.jsxを書き換え

```
/** @jsxImportSource @emotion/react */
import styled from '@emotion/styled';

const StyledDiv = styled.div`
  color: #09F;
`;
export const Text = () => {
  return (
    <StyledDiv>スタイルの適用された文字</StyledDiv>
  );
};
```

App3.jsx

```
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';
const buttonStyle = css`
  background-color: #f90;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  &:hover {
    background-color: #f00;
  }
`;
export const App = () => {
  return (
    <button css={buttonStyle}>Click Me</button>
  );
};
```

App4.jsx

```
import ReactDOMClient from "react-dom/client";
import { App } from "../App5";

export const ParentComponent = () => {
  return (
    <div>
      {/* Primary ボタン */}
      <App primary={true} />

      {/* Secondary ボタン */}
      <App primary={false} />
    </div>
  );
};
```

Parent.jsx

```
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';
const baseStyle = css`
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
`;
const primaryStyle = css`
  background-color: #00f;
  color: white;
  &:hover {
    background-color: #f00;
  }
`;
const secondaryStyle = css`
  background-color: #09f;
  color: white;
  &:hover {
    background-color: #f90;
  }
`;
export const App = ({ primary }) => {
  return (
    <button css={[baseStyle, primary ?
      primaryStyle : secondaryStyle]}>
      Click Me
    </button>
  );
};
```

App5.jsx

手順

- Text.jsxをcomponent フォルダー内に作成
- App6.jsxを作成
- main.jsxを書き換え
- App.module.cssは styled componentでも Emotionでも代用可能

```
import { useState } from 'react'
import classes from './App.module.css'
import { Text } from './components/Text'
export const App = ()=>{
  const onClickButton = ()=>{
    setNum((prev) => prev + 1);
  }
  const [num , setNum] = useState(0);
  return (
    <>
      <h1
        className={classes.titletext}>Reactでの表
        示</h1>
      <Text color="blue" message="text.jsx
        から表示しています"></Text>
      <button onClick={onClickButton}
        className={classes.btnLrg}>クリック
        </button>
      <p>クリックした回数:{num}</p>
    </>
  );
}
```

App6.jsx

```
export const Text = (props)=>{
  const textStyle = {
    color:props.color,
    fontSize:"#20px"
  }
  return (
    <p
      style={textStyle}>{props.message}</p>
    );
}
```

Text.jsx

ありがとうございました。
また次回。