React実習

02. JSXとコンポーネント

株式会社ジードライブ

今回学ぶこと

- JSX構文
- コンポーネント
- Props
- スタイルの適用

JSX構文

JSXとは

- React要素を生成するためのJavaScriptの拡張構文

JSXの例

```
const element = <h1>Hello</h1>;
```

JSXでの記述
クォート記号で囲んでいない
(文字列ではない)点に注目

- 元々は以下のような構文が使われていたが、React要素の生成処理がコンパクトに記述できる点と、画面を構成するHTMLのイメージが持ちやすい点から、JSXの利用が主流になっている

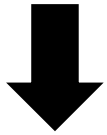
```
const element = React.createElement("h1", null, "Hello");
```

  - JSXはReact.createElementの糖衣構文

React要素について

- React要素は、JavaScriptのオブジェクト

```
const element = <h1 className="greeting">こんにちは</h1>;
console.log(element);
```

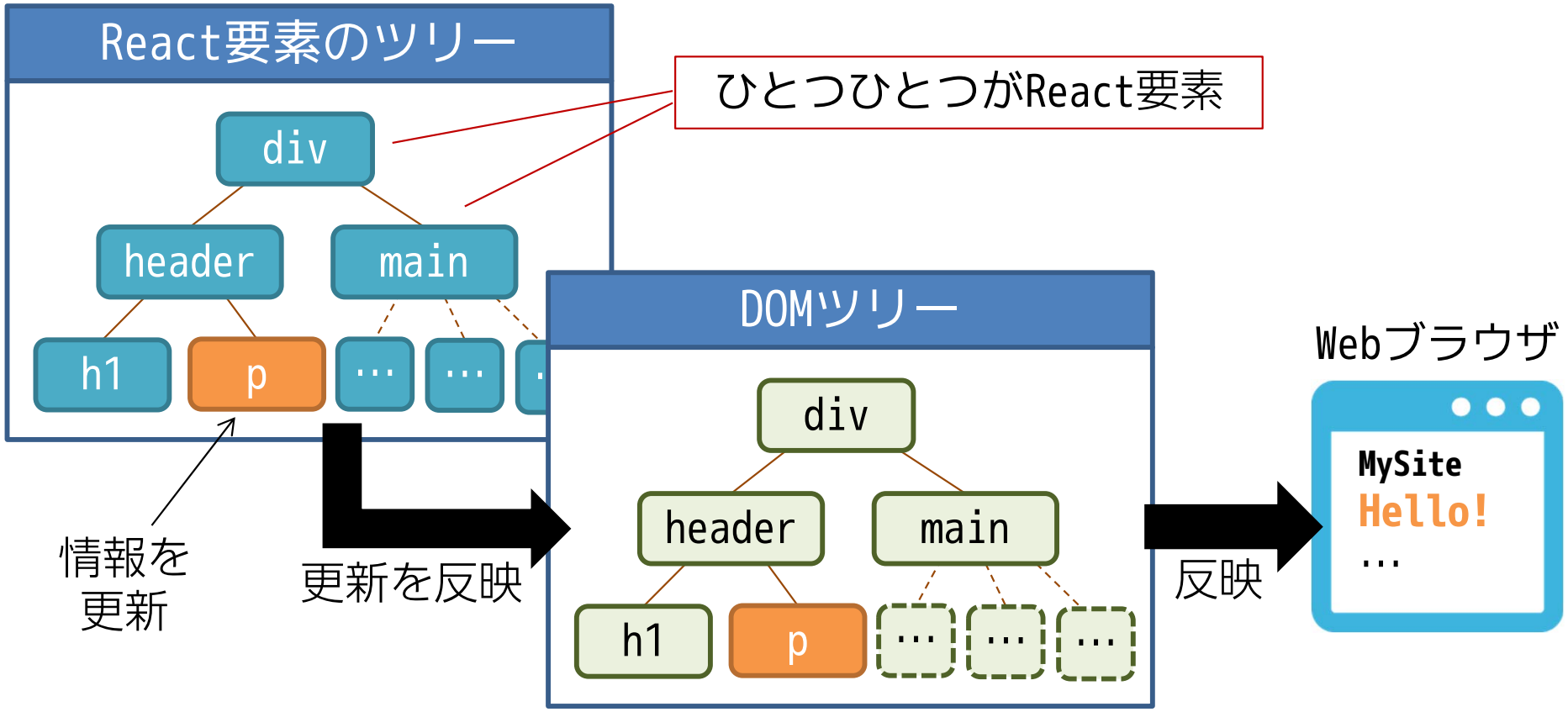


```
{
  type: "h1",
  props: {
     children: "こんにちは",
     className: "greeting"
  },
  ...
}
```

console.logで確認すると、
このような構造のオブジェクト
になっていることがわかる

React要素とDOM

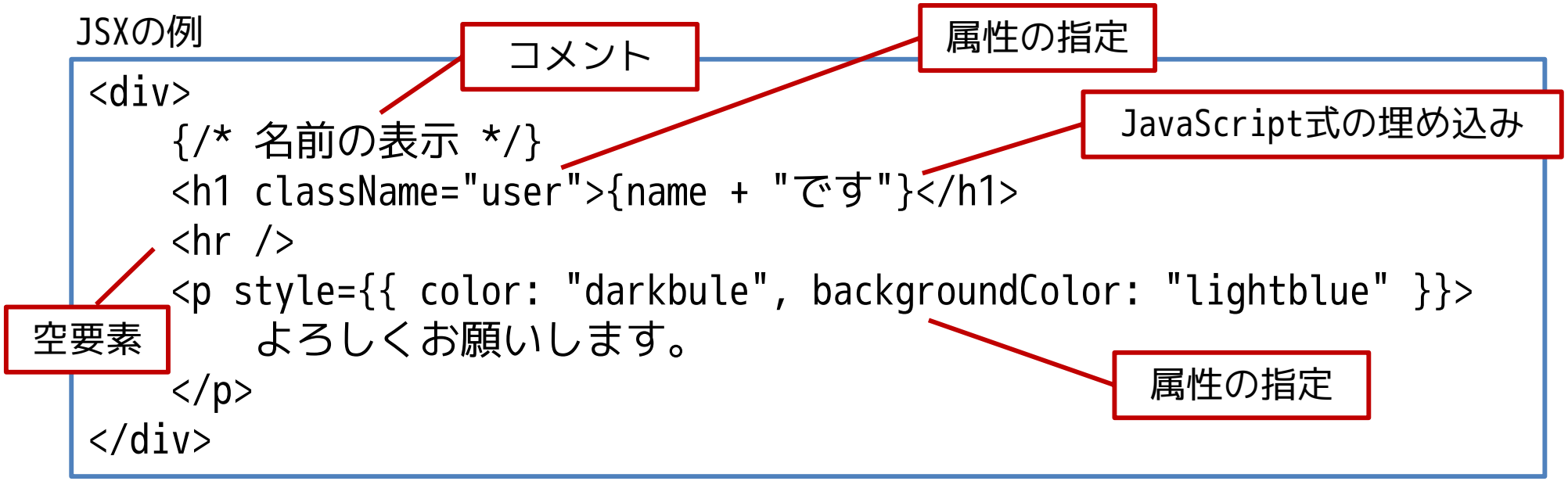
- React要素はツリー構造のデータとして管理される
  - React要素が更新されると、DOMツリーに反映される



JSXとHTML構文との違い

- JSXはHTMLの書き方に似ているが、以下の点が異なる
  - 空要素の書き方
  - JavaScript式の埋め込み
  - コメントの書き方
  - 属性の指定方法(一部)…など

JSXの例



空要素の書き方

- JSXはXMLがベースになっているため、空要素のタグは **/>** で閉じる必要がある

例: img要素

```
<img src="flower.png">      ⇐ NG
<img src="flower.png" />    ⇐ OK
```

例: input要素

```
<input type="text">      ⇐ NG
<input type="text" />    ⇐ OK
```

JavaScript式の埋め込み

- テンプレート内では、{ }を使って任意のJavaScript式を記述することができる

```
const name = "りんご";
const price = 100;
const photo = "apple.jpg";
```

例: JavaScript式の埋め込み

```
<p>商品名: {name}</p>
<p>税込み価格: {(price * 1.1).toFixed(0)}円</p>
<img src={`/images/${photo}`} alt={name} />
```

{ }を使って属性値を指定する場合、
属性値の前後にはクォート記号は付けない

コメントの書き方

- JSX内では、HTMLのコメント構文は使えないので、JavaScriptのブロックコメントを記述する必要がある

例：コメントの記述

```
<!-- 商品名の表示 -->     ⇐ NG
<p>商品名：{name}</p>

{/* 商品名の表示 */}      ⇐ OK
<p>商品名：{name}</p>
```

属性の指定

- classやforという属性名は、JavaScriptの予約語なので、代わりの属性名で代用する
  - class属性 ⇒ className属性
  - for属性 ⇒ htmlFor属性

例: className属性

```
<h1 className="title">Hello</h1>;
```

例: htmlFor属性

```
<label htmlFor="user">ユーザー名</label>
<input id="user" type="text" />
```

style属性の指定

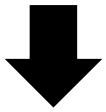
- style属性の指定にはオブジェクトリテラルを使う

例: style属性の指定

```
<p style={{ color: "#f00", backgroundColor: "#eee" }}>
    こんにちは
</p>
```

例: 変数を使ったstyle属性の指定

```
const textStyle =
        { color: "#f00", backgroundColor: "#eee" };
```



```
<p style={textStyle}>こんにちは</p>
```

children属性による子要素の指定

- children属性を使い、子要素を指定することができる

例：通常の子要素の指定

```
<div>
   <h1>こんにちは</h1>
</div>
```

```
const message = <h1>こんにちは</h1>;
```

例：children属性を使った子要素の指定

```
<div children={message}></div>
```

エスケープ処理

- JSX内で { } を使って文字列を出力する場合、文字列はエスケープ処理されて出力される

```
const message = "Hello<br>React!";
```

例: HTMLタグを含む文字列の出力

```
<h1>{message}</h1>
```

ブラウザでの表示

Hello
React!

エスケープ処理の無効化

- エスケープ処理を無効にして変数の値を出力したい場合は、以下の構文を使用する
  - 属性名からわかるように、安全が保障されないので、基本的には使用しない方がよい

```
const message = "Hello<br>React!";
```

例：HTMLタグを含む文字列の出力

```
<h1 dangerouslySetInnerHTML={{__html: message}}></h1>
```

ブラウザでの表示

Hello
React!

複数の要素

- JSX式は一つの親要素でまとめられている必要がある

```
const elm = <h1>Hello</h1><p>こんにちは</p>; ⇐ NG
const elm = <div><h1>Hello</h1><p>こんにちは</p></div>; ⇐ OK
const elm = <><h1>Hello</h1><p>こんにちは</p></>; ⇐ OK
```

<Fragment>～</Fragment>の省略記法
- フラグメントと呼ばれるもので、余計なタグを生成しない
- 後述のkey属性を設定する場合、この省略記法は使用できない
  ⇒ 省略記法を使わない場合、import {Fragment} from 'react'; の記述が必要

- JSX式を複数行で記述する場合は、( )で囲む

```
const elm = (<div>
    <h1>Hello</h1>
</div>);
```

分岐処理

- JSX内ではif文が使えないので、三項演算子を利用した分岐の記述がされる

```
<h1>MySite</h1>
{lang === "ja" ? <p>こんにちは!</p> : <p>Hello!</p>}
```

- &&が使われることもある
  - 左辺がfalseの場合、右辺を評価しない(短絡評価)

```
<h1>MySystem</h1>
{isAdmin && (<div>
              <h2>管理者情報</h2>
              <p>...</p>
            </div>
)}
```

falseの場合、管理者情報は出力されない

ループ処理

- 配列の要素をループで出力する場合、map()メソッドが利用される
  - ループ出力される要素にkey属性を設定する必要がある
  - key属性には、重複しないユニークな値が入る必要がある

```
const items = [{ id: 1, name: "りんご", price: 100 },
               { id: 2, name: "ぶどう", price: 700 },
               { id: 3, name: "みかん", price: 300 }];
```

例: オブジェクトの配列を出力

```
<div className="item-container">
        {items.map(item => (<Fragment key={item.id}>
            <h2>{item.name}</h2>
            <p>{(item.price * 1.1).toFixed(0)}円</p>
        </Fragment>))}
</div>
```

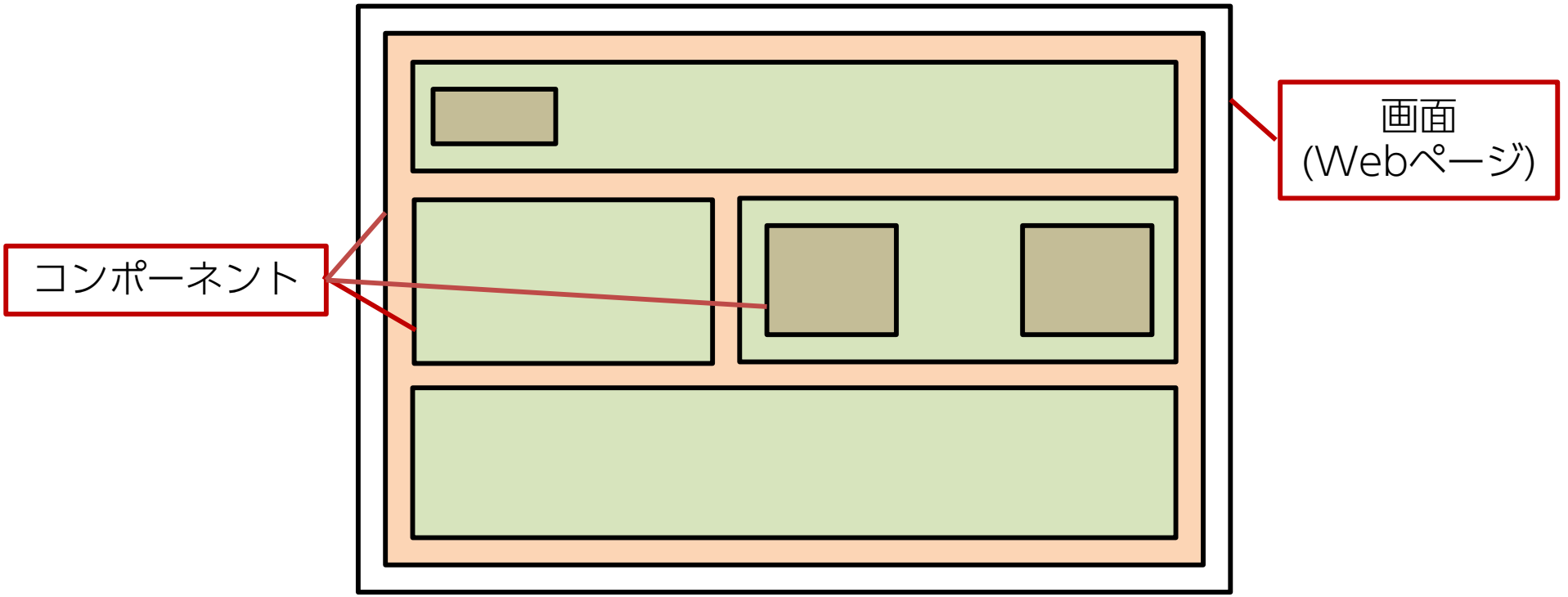
練習

- 練習02-1

コンポーネント

コンポーネントとは

- マークアップ(HTML)とロジックをまとめたReactアプリの構成部品
  - これらコンポーネントを画面内に配置することで、Reactアプリの画面(Webページ)を形成する



コンポーネントの作成

- コンポーネントファイルの拡張子は、.jsxを使用する
  - .jsでも問題ないが、.jsxの方がエディタのサポートを得やすい
  - ファイル名は任意だが、コンポーネント名をパスカルケースにしたものが採用されることが多い
  - １ファイルに複数のコンポーネントを定義することも可能だが、基本的にはコンポーネントごとにファイルを作成する
- コンポーネントには、以下の２種類の作成方法がある
  - クラスコンポーネント:
    JavaScriptのクラス構文を使い定義されるコンポーネント
  - 関数コンポーネント:
    JavaScriptの関数構文を使い定義されるコンポーネント

現在は関数コンポーネントを使う方法が主流

クラスコンポーネント

- クラスコンポーネントはReact.Componentクラスのサブクラスとして作成する
  - render()メソッドを実装する必要がある
    ⇒ メソッドの戻り値はReact要素(JSX式で記述する)

例： Greeting.jsx

コンポーネント名

import React from "react";

```
export default class Greeting extends React.Component {
  render() {
    return (<div>
        <h1>ごあいさつ</h1>
        <p>こんにちは, {this.props.name}さん!</p>
    </div>);
  }
}
```

コンポーネントを構成するHTMLをJSXで記述

propsは親コンポーネントからデータを受け取るための特別なプロパティ

関数コンポーネントの作成

- 関数コンポーネントは、React要素を返す関数として作成する

例： Greeting.jsx

親コンポーネントからデータを受け取るための引数を指定することができる

コンポーネント名

```
export default function Greeting(props) {
  return (<div>
      <h1>ごあいさつ</h1>
      <p>こんにちは, {props.name}さん!</p>
    </div>);
}
```

関数式で記述してもよい

```
const Greeting = (props) => {
  return <h1>こんにちは</h1>;
};
```

コンポーネントの読み込み

- コンポーネントはJSX式の中に埋め込むことができる
  - コンポーネント名をタグ名として記述する

例： AppコンポーネントにGreetingコンポーネントを読み込む

```
// Greetingコンポーネントを Greeting.jsxからインポート
import Greeting from "./components/Greeting";

export default function App() {
  return (<div>
      <h1>My Site</h1>
      <Greeting name="山田太郎" />
    </div>);
}
```

コンポーネント名

コンポーネント名

属性を使い、コンポーネントにデータを渡すことができる

ファイル名変更時の注意点

- ファイル名を変更後、コンポーネントのインポートでエラーが発生する場合は、一時的に別ファイル名に変更してから元に戻す

例: myComponent.jsx のファイル名を MyComponent.jsx に修正した
⇒ インポート文でエラーが発生

```
import MyComponent from "./components/MyComponent";
```

エラー

① ファイル名を tempMyComponent.jsx のように変更し、
⇒ インポート文も併せて修正する

```
import MyComponent from "./components/tempMyComponent";
```

② ファイル名を元の MyComponent.jsx に戻し、インポート文も元に戻す

```
import MyComponent from "./components/MyComponent";
```

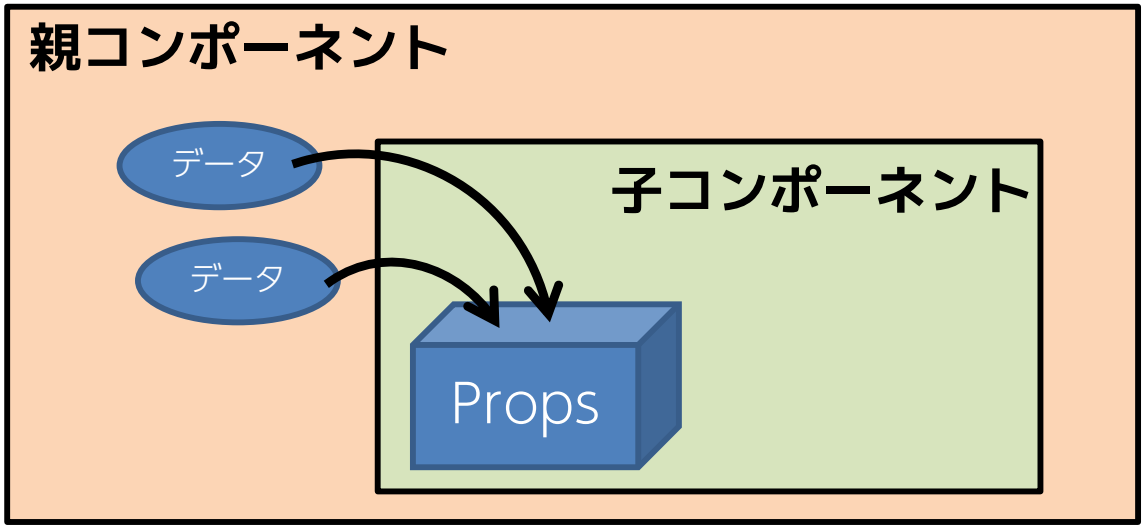
練習

- 練習02-2

Props

Props

- Propertiesの略で、コンポーネントのプロパティ(属性)という意味
- JSXでは、属性を使うことで、親コンポーネントから、子コンポーネントへデータを渡すことができ、そのデータが子コンポーネントのプロパティ(Props)となる
  - Propsは読み取り専用で、子コンポーネント内では変更できない



親コンポーネントの記述例

- 親コンポーネントからは、任意の属性を通じてデータを渡すことができる

例： AppコンポーネントからGreetingコンポーネントへデータを渡す

```
import Greeting from "./components/Greeting";

export default function App() {
  return (
    <div className="container">
      <Greeting name="山田太郎" lang={1} />
    </div>
  );
}
```

数値や真偽値など、文字列以外を渡す場合は、{ }で値を囲む

任意のプロパティ

子コンポーネントの記述例

- 関数コンポーネントの場合、親コンポーネントからのデータは引数を通じて受け取ることができる

例： 親コンポーネントから受け取ったデータの利用

```
export default function Greeting(props) {
  return (
    <div>
     <h1>{props.name}さん</h1>
     {props.lang === 1 ? <p>こんにちは!</p> : <p>Hello!</p>}
    </div>
  );
}
```

任意の引数名

親コンポーネントで指定した属性名

親コンポーネントで指定した属性名

childrenプロパティ

- 開始タグと終了タグに囲まれた要素は、childrenプロパティとして受け取ることができる

例: Cardコンポーネントの利用(親コンポーネントの記述)

```
<Card title="りんご">
      <img src="/images/apple.jpg" alt="りんご" />
      <p>青森県産</p>
</Card>
```

例: Cardコンポーネント

```
export default function Card(props) {
  return (<div className="card">
       <h1>{props.title}</h1>
       {props.children}
  </div>);
}
```

Propsの分割代入

- 関数コンポーネントの引数は分割代入の形式で記述することができる

例: Cardコンポーネントの利用(親コンポーネントの記述)

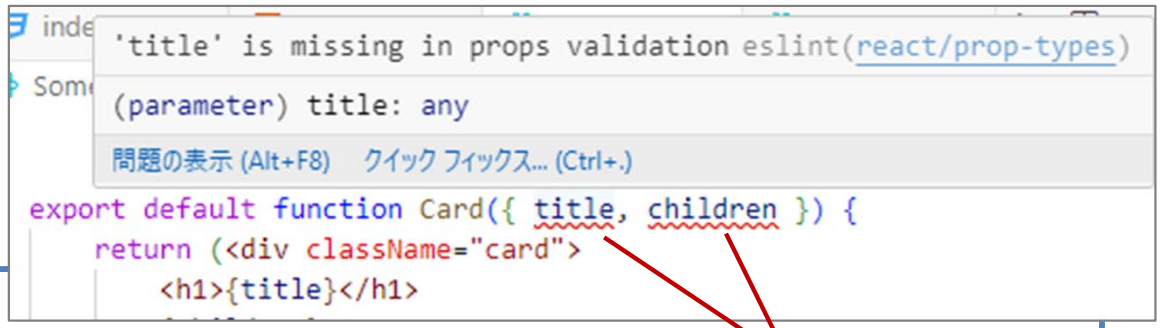
```
<Card title="りんご">
      <img src="/images/apple.jpg" alt="りんご" />
      <p>青森県産</p>
</Card>
```

例: Cardコンポーネント

```
export default function Card({ title, children }) {
  return (<div className="card">
       <h1>{title}</h1>
       {children}
  </div>);
}
```

Propsの警告

- Propsに関する警告を抑制するには、以下の設定を記述する



警告が出てくる

eslint.config.jsへの追記

```
    rules: {
      ...js.configs.recommended.rules,
      ...react.configs.recommended.rules,
      ...react.configs['jsx-runtime'].rules,
      ...reactHooks.configs.recommended.rules,
      'react/prop-types': 0,
      'react/jsx-no-target-blank': 'off'
      'react-refresh/only-export-compone
        'warn',
        { allowConstantExport: true },
    ],
```

react.configs.recommended.rules
よりも下に記す

viteの最新版では、この設定は不要

Propsのバケツリレー

- 親コンポーネントから孫コンポーネントにデータを渡すことも可能だが、データのバケツリレーが必要になる
  - 記述が煩雑になってしまう ⇒ 解決手段は、後の章で学習する

親

```
function Parent() {
  return (<div>
      <h1>Parent</h1>
      <Child dataToChild="Hello" />
    </div>);
}
```

子

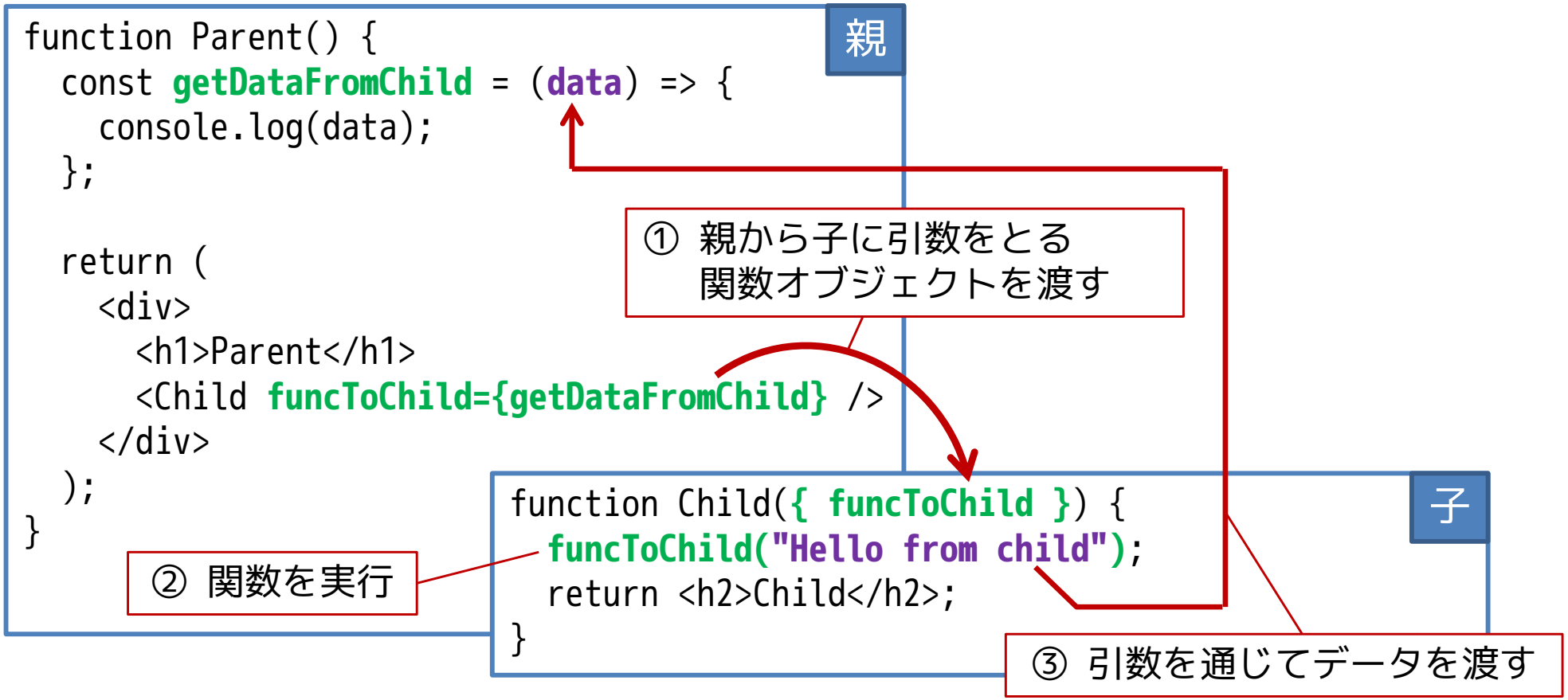
```
function Child({ dataToChild }) {
  return (<div>
      <h2>Child</h2>
      <p>Data from parent: {dataToChild} </p>
      <GrandChild dataToGrandChild={dataToChild} />
    </div>);
}
```

孫

```
function GrandChild({ dataToGrandChild }) {
  return (<div>
      <h3>GrandChild</h3>
      <p>Data from parent: {dataToGrandChild}</p>
    </div>);
}
```

子から親にデータを渡す

- 子から親コンポーネントにデータを渡す場合、関数オブジェクトを利用する必要がある
  - これも記述が煩雑になってしまう ⇒ 解決手段は、後の章で学習する



練習

- 練習02-3

スタイルの適用

スタイルの適用方法

- コンポーネントにスタイルを適用するには、以下のような方法がある
  - グローバルなCSS
  - CSSフレームワーク
  - コンポーネントライブラリ
  - CSS Modules
  - CSS-in-JS

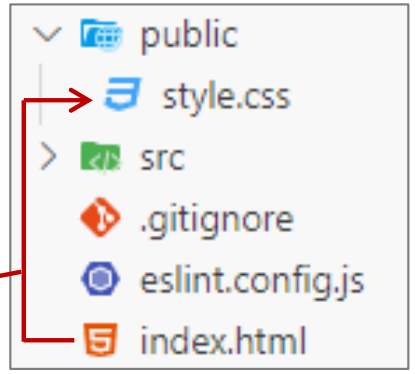
コンポーネントごとに独立したスタイリングができる
(他のコンポーネントに影響を与えない)

グローバルなCSSの適用方法

- 方法1：publicフォルダ内のCSSファイルをindex.htmlから参照する

```
<link href="/style.css" rel="stylesheet">
```

linkタグで読み込む

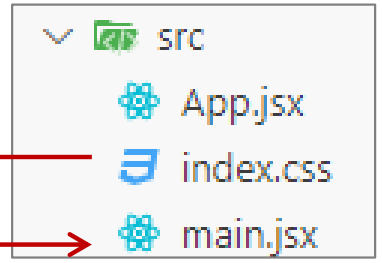


- 方法2：main.jsxからCSSファイルをインポートする
  - 各コンポーネントからCSSをインポートすることもできるが、読み込んだCSSは、他のコンポーネントにも影響を及ぼしてしまう

同階層にあるindex.cssのインポート

```
import "./index.css";
```

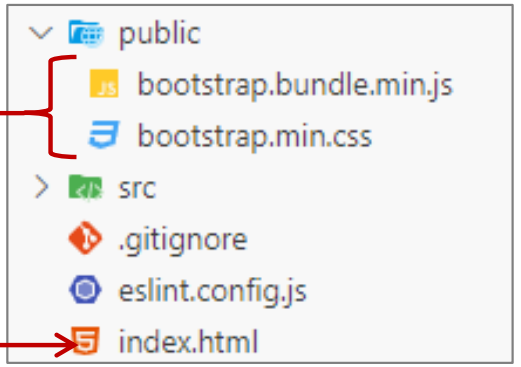
importする



CSSフレームワーク

- Reactプロジェクトにおいても、**Bootstrap**や**Tailwind**といったCSSフレームワークを利用することができる
  - Bootstrapは、これまでに学習してきたように、linkタグやscriptタグを使い、必要なCSSやJavaScriptファイルを読み込むことで利用可能

linkタグやscriptタグで読み込む



例：コンポーネントでのBootstrapの利用

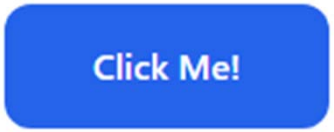
```
export default function App() {
  return (<div className="container">
      <h1 className="display-2">MySite</h1>
      <button className="btn btn-primary">Show Details</button>
    </div>);
}
```

Tailwind

- 決められたクラス名を指定することで、スタイルを適用する点はBootstrapと同じだが、ナビゲーションやボタンといった一連の定義済みのクラスは提供しない
  - 数多くのユーティリティクラスが用意されており、それらを組み合わせて目的とするスタイルを適用する
  - Tailwindの導入手順：
    https://tailwindcss.com/docs/guides/create-react-app

例: Tailwindによるボタンの作成

```
<button className="px-5 py-4 text-base font-medium text-center
text-white transition duration-500 ease-in-out transform bg-blue-
600 lg:px-10 rounded-xl hover:bg-blue-700 focus:outline-none
focus:ring-2 focus:ring-offset-2 focus:ring-blue-500">
      Click Me!
</button>
```



コンポーネントライブラリ

- ライブラリとして配布されているコンポーネントを利用することで、画面を作成することも可能
  - 多くの場合、色や大きさなどをカスタマイズすることができる
    ⇒ Propsとして設定する
- 以下のようなライブラリが存在する
  - NextUI
  - Chakra UI
  - Semantic UI
  - React Suite
  - **React Bootstrap**…など

React Bootstrap

- Bootstrapをコンポーネントとして利用することができるライブラリ

npmでインストールする必要がある

```
npm install react-bootstrap bootstrap
```

例：Buttonコンポーネントの利用

```
import Button from "react-bootstrap/Button";

export default function App() {
  return <Button as="a" variant="primary">Link</Button>;
}
```

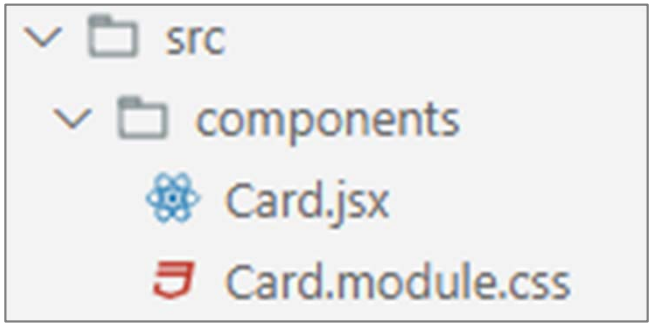
<a class="btn btn-primary>をコンポーネントとして記述できる

※ BootstrapのCSSの読み込みが必要

CSS Modules

- CSSファイルをJavaScriptのオブジェクトとして、読み込むことができる技術
  - HTMLとして出力される際に、ユニークなクラス名になるため、他のコンポーネントに影響しない
- CSSファイルの名称は、**コンポーネント名.module.css**にする
  - 一つのコンポーネントに一つのCSSモジュールファイルを対応させる

例：Cardコンポーネントと CSSモジュール



CSS Modules

- CSSの記述は、クラスセレクタを使って行う
- CSSをインポートする際に、任意の名前を付ける

例：Card.module.css

```
.card {
  border: 1px solid #000;
  border-radius: 5%;
}

.title {
  margin-bottom: 10px;
}
```

クラスセレクタ

例：Card.jsx

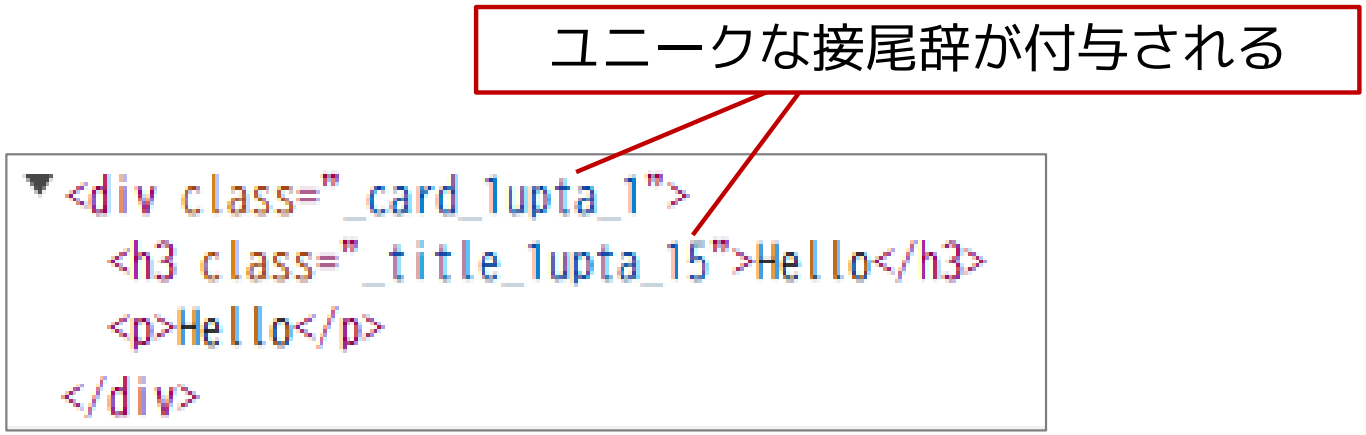
```
import classes from "./Card.module.css";

export default function Card(props) {
  return (
    <div className={classes.card}>
      <h3 className={classes.title}>
        {props.text}
      </h3>
      {props.children}
    </div>
  );
}
```

任意の名前

インポートしたモジュールのプロパティとして、クラス名が使われる

CSS Modules

- レンダリングされる際は、ユニークなクラス名になる



CSS-in-JS

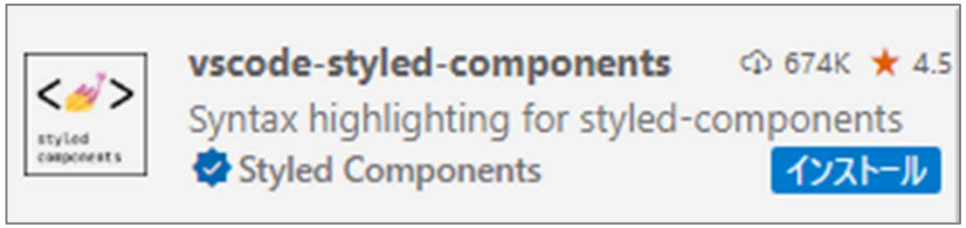
- CSS-in-JSは、CSSをJavaScriptのオブジェクトとして記述するしくみで、以下のようなライブラリが存在する
  - **styled-components**
  - Emotion
  - JSS
  - styled-jsx…など

styled-components

- styled-componentsを利用するには、パッケージのインストールが必要

```
npm install styled-components
```

- VS Codeの拡張機能を入れておくと、CSSの記述がしやすくなる



styled-components

例：Cardコンポーネント

```
import styled from "styled-components";

const StyledCard = styled.div`
  border: 1px solid #000;
  text-align: center;

  h3 {
    background: #000;
  }
`;

export default function Card({ text, children }) {
  return (
    <StyledCard>
      <h3>{text}</h3>
      {children}
    </StyledCard>
  );
}
```

styled-componentsを任意の名前でインポート

任意のHTMLタグ名

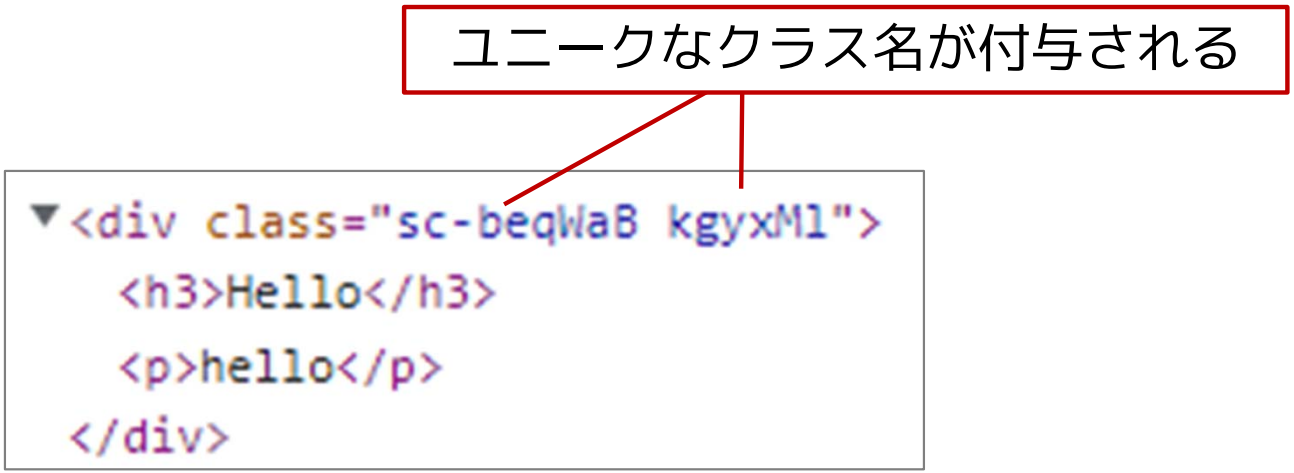
バッククオート(` `)内に、CSSを記述する
↓
タグ付きテンプレートリテラルという記法で関数が実行され、スタイルが適用されたコンポーネントを返す（この場合、divコンポーネント）

divに変換され、border, text-alignが適用される

backgroundが適用される

styled-components

- レンダリングされる際は、ユニークなクラス名になる



練習

- 練習02-4