ポータル

ポータル (portal) は、親コンポーネントの DOM 階層外にある DOM ノードに対して子コンポーネントをレンダーするための公式の仕組みを提供します。

```
ReactDOM.createPortal(child, container)
```

第 1 引数 (`child`) は React の子要素としてレンダー可能なもの、例えば、要素、文字列、フラグメントなどです。第 2 引数 (`container`) は DOM 要素を指定します。

使い方

通常、コンポーネントの `render` メソッドから要素を返すと、最も近い親ノードの子として DOM にマウントされます。

```
render() {
  // React は新しい div 要素をマウントし、子をその中に描画します
  return (
    <div>
      {this.props.children}
    </div>
  );
}
```

しかし、時に子要素を DOM 上の異なる位置に挿入したほうが便利なことがあります。

```
render() {
  // React は新しい div をつくり*ません*。子要素は `domNode` に対して描画されます。
  // `domNode` は DOM ノードであれば何でも良く、 DOM 構造内のどこにあるかは問いません。
  return ReactDOM.createPortal(
    this.props.children,
    domNode
  );
}
```

ポータルの典型的なユースケースとは、親要素が `overflow: hidden` や `z-index` のスタイルを持っていても、子要素がコンテナを「飛び出して」見える必要があるものです。例えば、ダイアログ、ホバーカード、ツールチップがそれに当たります。

> **補足**
> ポータルを利用する際は、キーボードのフォーカスの管理を行うことが重要になるので、忘れずに行ってください。
> モーダルダイアログについては WAI-ARIA モーダルの推奨実装方法に従い、誰もが利用できるという状態を確保してください。

Try it on CodePen

ポータルを介したイベントのバブリング

ポータルは DOM ツリーのどこにでも存在できますが、他のあらゆる点では通常の React の子要素と変わらずに振る舞います。コンテクスト (context) のような機能は、たとえ子要素がポータルであろうと全く同じように動きます。というのも、**DOM ツリー**上の位置にかかわらず、ポータルは依然として **React のツリー**内にいるからです。
これにはイベントのバブリングも含まれます。ポータルの内部で発火したイベントは **React のツリー**内の祖先へと伝播します。たとえそれが **DOM ツリー**上では祖先でなくともです。次のような HTML 構造があったとして、

```
<html>
  <body>
    <div id="app-root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

`#app-root` 内にある `Parent` コンポーネントは、`#modal-root` 内のコンポーネントから伝播したイベントが捕捉されなかった場合に、それを捕捉できます。

```
// この 2 つのコンテナは DOM 上の兄弟要素とします
const appRoot = document.getElementById('app-root');
const modalRoot = document.getElementById('modal-root');

class Modal extends React.Component {
```

```
  constructor(props) {
    super(props);
    this.el = document.createElement('div');
  }

  componentDidMount() {
    // ポータルの要素が DOM ツリーに挿入されるのは、
    // Modal の子要素がマウントされた後になります。
    // つまり、子要素は一旦どこにも結びつかない
    // DOM ノードへとマウントされるということです。
    // もし子コンポーネントがマウント後すぐに DOM ツリーに結びついてほしい ―
    // たとえば DOM ノードの大きさを測りたい、子孫要素で `autoFocus` を使いたいなど
    // ― 場合は、 Modal に状態を持たせて Modal が
    // DOM ツリーに挿入されているときだけ子要素をレンダーするようにします。
    modalRoot.appendChild(this.el);
  }

  componentWillUnmount() {
    modalRoot.removeChild(this.el);
  }

  render() {
    return ReactDOM.createPortal(
      this.props.children,
      this.el,
    );
  }
}

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {clicks: 0};
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    // これは Child 内のボタンがクリックされた際に発火し、
    // Parent の state を更新します。
    // たとえそのボタンが DOM 上では直系の子孫でなかったとしてもです。
    this.setState(state => ({
      clicks: state.clicks + 1
    }));
  }

  render() {
    return (
      <div onClick={this.handleClick}>
        <p>Number of clicks: {this.state.clicks}</p>
        <p>
          Open up the browser DevTools
          to observe that the button
          is not a child of the div
          with the onClick handler.
        </p>
        <Modal>
          <Child />
        </Modal>
      </div>
    );
  }
}

function Child() {
  // クリックするとイベントが親に伝播します。
  // なぜならここには `onClick` 属性が定義されてないからです。
  return (
    <div className="modal">
      <button>Click</button>
    </div>
  );
}

ReactDOM.render(<Parent />, appRoot);
```

Try it on CodePen

ポータルから伝播したイベントが親コンポーネントで捕捉できるということは、ポータルに本質的に依存することのない、より柔軟な抽象化が可能であるということを示しています。たとえば `<Modal />` の実装がポータルを使っているかに関係なく、 `<Modal />` コンポーネントをレンダーしてそこから来るイベントを捕捉することができます。

このページを編集する