条件付きレンダー

React ではあなたの必要なふるまいをカプセル化した独立したコンポーネントを作ることができます。そして、あなたのアプリケーションの状態に応じて、その一部だけを描画することが可能です。
React における条件付きレンダーは JavaScript における条件分岐と同じように動作します。`if` もしくは条件演算子のような JavaScript 演算子を使用して現在の状態を表す要素を作成すれば、React はそれに一致するように UI を更新します。
以下の 2 つのコンポーネントを考えてみましょう：

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

ユーザがログインしているかどうかによって、これらのコンポーネントの一方だけを表示する `Greeting` コンポーネントを作成しましょう：

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

Try it on CodePen
この例では `isLoggedIn` プロパティの値によって異なる挨拶メッセージを表示します。

要素変数
要素を保持しておくために変数を使うことができます。これは、出力の他の部分を変えずにコンポーネントの一部を条件付きでレンダーしたい時に役立ちます。
ログアウトとログインボタンを表す以下の 2 つの新しいコンポーネントを考えましょう：

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

以下の例では、`LoginControl` というステート付きコンポーネントを作成します。
`LoginControl` は現在の state によって `<LoginButton />` もしくは `<LogoutButton />` の一方をレンダーします。加えて、前の例の `<Greeting />` もレンダーします：

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
```

```
    this.setState({isLoggedIn: false});
  }

  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button;

    if (isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}

ReactDOM.render(
  <LoginControl />,
  document.getElementById('root')
);
```

Try it on CodePen

変数を宣言して `if` 文を使用することはコンポーネントを条件的にレンダーするなかなか良い方法ではありますが、より短い構文を使いたくなる時もあります。以下で述べるように、JSX でインラインで条件を記述する方法がいくつか存在します。

論理 && 演算子によるインライン If

中括弧で囲むことで、JSX に任意の式を埋め込むことができます。これには JavaScript の論理 && 演算子も含まれます。これは条件に応じて要素を含めたいというときに便利です。

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}

const messages = ['React', 'Re: React', 'Re:Re: React'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

Try it on CodePen

これが動作するのは、JavaScript では `true && expression` は必ず `expression` と評価され、`false && expression` は必ず `false` と評価されるからです。
従って、条件部分が `true` であれば、`&&` の後に書かれた要素が出力に現れます。もし `false` であれば、React はそれを無視して飛ばします。

条件演算子によるインライン If-Else

条件的に要素をレンダリングするもうひとつの方法は JavaScript の `condition ? true : false` 条件演算子を利用することです。
以下の例では条件演算子を用いて、条件に応じてテキストの小さなブロックをレンダリングします。

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

より大きな式にも適用することができますが、何が起こっているのか分かりづらくはなります：

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn ? (
        <LogoutButton onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton onClick={this.handleLoginClick} />
```

```
      )}
    </div>
  );
}
```

普通の JavaScript を書くときと同様、あなたとチームが読みやすいと思えるものに合わせて、適切なスタイルを選択してください。条件が複雑になりすぎたら、コンポーネントを抽出するべきタイミングかもしれない、ということにも留意してください。

コンポーネントのレンダーを防ぐ

稀なケースですが、他のコンポーネントによってレンダーされているにも関わらず、コンポーネントが自分のことを隠したい、ということがあるかもしれません。その場合はレンダー出力の代わりに `null` を返すようにしてください。

以下の例では、`<WarningBanner />` バナーは `warn` と呼ばれるプロパティの値に応じてレンダーされます。そのプロパティの値が `false` なら、コンポーネントはレンダリングされません：

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}

class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = {showWarning: true};
    this.handleToggleClick = this.handleToggleClick.bind(this);
  }

  handleToggleClick() {
    this.setState(state => ({
      showWarning: !state.showWarning
    }));
  }

  render() {
    return (
      <div>
        <WarningBanner warn={this.state.showWarning} />
        <button onClick={this.handleToggleClick}>
          {this.state.showWarning ? 'Hide' : 'Show'}
        </button>
      </div>
    );
  }
}

ReactDOM.render(
  <Page />,
  document.getElementById('root')
);
```

Try it on CodePen

コンポーネントの `render` メソッドから `null` を返してもコンポーネントのライフサイクルメソッドの発火には影響しません。例えば `componentDidMount` は変わらず呼び出されます。

このページを編集する