コンポジション vs 継承

React は強力なコンポジションモデルを備えており、コンポーネント間のコードの再利用には継承よりもコンポジションをお勧めしています。
この章では、React を始めて間もない開発者が継承に手を出した時に陥りがちないくつかの問題と、コンポジションによりその問題がどのように解決できるのかについて考えてみます。

子要素の出力 (Containment)

コンポーネントの中には事前には子要素を知らないものもあります。これは `Sidebar` や `Dialog` のような汎用的な “入れ物” をあらわすコンポーネントではよく使われています。
このようなコンポーネントでは特別な `children` という props を使い、以下のようにして受け取った子要素を出力することができます。

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

これにより他のコンポーネントから JSX をネストすることで任意の子要素を渡すことができます。

```
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}
```

Try it on CodePen

`<FancyBorder>` JSX タグの内側のあらゆる要素は `FancyBorder` に `children` という props として渡されます。`FancyBorder` は `<div>` の内側に `{props.children}` をレンダリングするので、渡された要素が出力されます。
あまり一般的ではありませんが、複数の箇所に子要素を追加したいケースも考えられます。そのようなケースでは以下のように `children` の props の代わりに独自の props を作成して渡すことができます。

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}

function App() {
  return (
    <SplitPane
      left={
        <Contacts />
      }
      right={
        <Chat />
      } />
  );
}
```

Try it on CodePen

`<Contacts />` や `<Chat />` のような React の要素はただのオブジェクトなので、他のあらゆるデータと同様に props として渡すことができます。このアプローチは他のライブラリで言うところの slot に似ていると感じるかもしれませんが、React のコンポーネントに props として渡せるものに制限はありません。

特化したコンポーネント (Specialization)

コンポーネントを他のコンポーネントの "特別なケース" として考えることがあります。例えば、`WelcomeDialog` は `Dialog` の特別なケースと言えるでしょう。

React ではこれもコンポジションで実現できます。汎用的なコンポーネントに props を渡して設定することで、より特化したコンポーネントを作成することができます。

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
    </FancyBorder>
  );
}

function WelcomeDialog() {
  return (
    <Dialog
      title="Welcome"
      message="Thank you for visiting our spacecraft!" />

  );
}
```

Try it on CodePen

コンポジションはクラスとして定義されたコンポーネントでも同じように動作します。

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
      {props.children}
    </FancyBorder>
  );
}

class SignUpDialog extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSignUp = this.handleSignUp.bind(this);
    this.state = {login: ''};
  }

  render() {
    return (
      <Dialog title="Mars Exploration Program"
              message="How should we refer to you?">
        <input value={this.state.login}
               onChange={this.handleChange} />

        <button onClick={this.handleSignUp}>
          Sign Me Up!
        </button>
      </Dialog>
    );
  }

  handleChange(e) {
    this.setState({login: e.target.value});
  }

  handleSignUp() {
    alert(`Welcome aboard, ${this.state.login}!`);
  }
}
```

Try it on CodePen

継承はどうするの？

Facebook では、何千というコンポーネントで React を使用していますが、コンポーネント継承による階層構造が推奨されるケースは全く見つかっていません。

props とコンポジションにより、コンポーネントの見た目と振る舞いを明示的かつ安全にカスタマイズするのに十分な柔軟性が得られます。コンポーネントはどのような props でも受け付けることができ、それはプリミティブ値でも、React 要素でも、あるいは関数であってもよい、ということに留意して下さい。

コンポーネント間で非 UI 機能を再利用したい場合は、それを別の JavaScript モジュールに抽出することをお勧めします。コンポーネントはその関数やオブジェクト、クラスなどを継承することなくインポートすることで使用することができるでしょう。

このページを編集する