ref のフォワーディング

ref のフォワーディングはあるコンポーネントを通じてその子コンポーネントのひとつに ref を自動的に渡すテクニックです。これは基本的にはアプリケーション内のほとんどのコンポーネントで必要ありません。しかし、コンポーネントの種類によっては、特に再利用可能なコンポーネントライブラリにおいては、便利なものとなるかもしれません。一般的なシナリオについて以下で述べます。

DOM コンポーネントに ref をフォワーディングする

ネイティブの `button` DOM 要素をレンダーする `FancyButton` というコンポーネントを考えてみましょう：

```
function FancyButton(props) {
  return (
    <button className="FancyButton">
      {props.children}
    </button>
  );
}
```

React コンポーネントは、レンダーの結果も含め、実装の詳細を隠蔽します。`FancyButton` を使用する他のコンポーネントは内側の `button` DOM 要素に対する ref を取得する **必要は通常ありません**。これは、互いのコンポーネントの DOM 構造に過剰に依存することを防ぐので、良いことです。
そういったカプセル化は `FeedStory` や `Comment` のようなアプリケーションレベルのコンポーネントでは望ましいことではありますが、`FancyButton` や `MyTextInput` といった非常に多くのところで再利用可能な "末梢の" コンポーネントでは不便である可能性があります。このようなコンポーネントは、アプリケーションのいたるところで通常の DOM である `button` や `input` と同様に扱われる傾向にあり、フォーカス、要素の選択、アニメーションをこなすにはそれらの DOM にアクセスすることが避けられないかもしれません。
ref のフォワーディングはオプトインの機能であり、それにより、コンポーネントが ref を受け取って、それをさらに下層の子に渡せる（つまり、ref を "フォワーディング" できる）ようになります。
下の例では、`FancyButton` は渡された `ref` を取得して、それをレンダーする `button` DOM にフォワーディングするために、`React.forwardRef` を使っています。

```
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="FancyButton">
    {props.children}
  </button>
));

// You can now get a ref directly to the DOM button:
const ref = React.createRef();
<FancyButton ref={ref}>Click me!</FancyButton>;
```

このように、`FancyButton` を使ったコンポーネントは下層の `button` DOM ノードの ref を取得することができ、必要であれば `button` DOM を直接使うかのように、DOM にアクセスすることができます。
上の例で、何が起こっているかを順々に説明します。

1. `React.createRef` を呼び、React ref をつくり、それを `ref` 変数に代入します。
2. `ref` を `<FancyButton ref={ref}>` に JSX の属性として指定することで渡します。
3. React は `ref` を、`forwardRef` 内の関数 `(props, ref) => ...` の 2 番目の引数として渡します。
4. この引数として受け取った `ref` を `<button ref={ref}>` に JSX の属性として指定することで渡します。
5. この ref が紐付けられると、`ref.current` は `<button>` DOM ノードのことを指すようになります。

> **補足**
> 2 番目の引数 `ref` は `React.forwardRef` の呼び出しを使ってコンポーネントを定義したときにだけ存在します。通常の関数またはクラスコンポーネントは `ref` 引数を受け取らず、ref は props からも利用できません。
> ref のフォワーディング先は DOM コンポーネントだけにとどまりません。クラスコンポーネントインスタンスに対しても ref をフォワーディングできます。

コンポーネントライブラリのメンテナ向けの補足

コンポーネントライブラリの中で、`forawrdRef` を使い始めた場合、破壊的変更として扱い、新しいメジャーバージョンをリリースすべきです。ライブラリが外から見て今までと違う挙動（例えば、どの値が ref に代入されるかや、どの型がエクスポートされるのか）をする可能性があり、古い挙動に依存しているアプリケーションや他のライブラリを壊す可能性があるからです。
`React.forwardRef` が存在する場合だけ、条件的に `React.forwardRef` を適用することも同じ理由で推奨されません：そのような実装は、React そのものを更新したとき、ライブラリがどのように振る舞うかを変えてしまい、ユーザのアプリケーションを破壊する可能性があるからです。

高階コンポーネントにおける ref のフォワーディング

このテクニックは 高階コンポーネント（HOC としても知られています）においても特に便利です。コンポーネントの props をコンソールにログ出力する HOC を例として考えてみましょう。

```
function logProps(WrappedComponent) {
  class LogProps extends React.Component {
    componentDidUpdate(prevProps) {
      console.log('old props:', prevProps);
      console.log('new props:', this.props);
    }

    render() {
      return <WrappedComponent {...this.props} />;
    }
  }

  return LogProps;
}
```

“logProps” HOC はすべての `props` をラップするコンポーネントに渡すので、レンダーされる出力は同じになるでしょう。例えば、“fancy button” コンポーネントに渡されるすべての props をログとして記録するために、この HOC を使用することができます。

```
class FancyButton extends React.Component {
  focus() {
    // ...
  }

  // ...
}

// Rather than exporting FancyButton, we export LogProps.
// It will render a FancyButton though.
export default logProps(FancyButton);
```

上の例でひとつ注意があります：ref は渡されません。`ref` は props のひとつではないからです。`key` と同様に ref は React では props とは違う扱いになります。HOC に対する ref を追加した場合、ラップされたコンポーネントではなく、一番外側のコンテナコンポーネントを参照します。
これは `FancyButton` コンポーネントに紐付けられることを意図した ref が、実際には `LogProps` コンポーネントに紐付けられることを意味します。

```
import FancyButton from './FancyButton';

const ref = React.createRef();

// The FancyButton component we imported is the LogProps HOC.
// Even though the rendered output will be the same,
// Our ref will point to LogProps instead of the inner FancyButton component!
// This means we can't call e.g. ref.current.focus()
<FancyButton
  label="Click Me"
  handleClick={handleClick}
  ref={ref}
/>;
```

幸いにも、`React.forwardRef` API を使って、内側の `FancyButton` コンポーネントに対して ref を明示的に渡すことができます。`React.forwardRef` は render 関数を受け取り、その関数は `props` と `ref` を引数として取り、React ノードを返します。例えば、

```
function logProps(Component) {
  class LogProps extends React.Component {
    componentDidUpdate(prevProps) {
      console.log('old props:', prevProps);
      console.log('new props:', this.props);
    }

    render() {
      const {forwardedRef, ...rest} = this.props;

      // Assign the custom prop "forwardedRef" as a ref
      return <Component ref={forwardedRef} {...rest} />;
    }
  }

  // Note the second param "ref" provided by React.forwardRef.
  // We can pass it along to LogProps as a regular prop, e.g. "forwardedRef"
  // And it can then be attached to the Component.
  return React.forwardRef((props, ref) => {
    return <LogProps {...props} forwardedRef={ref} />;
  });
}
```

DevTools でのカスタム名表示

`React.forwardRef` は render 関数を受け取ります。React DevTools は ref をフォワーディングしているコンポーネントとして何を表示すべきかを決定するために、この関数を使います。例えば、次のコンポーネントは *"ForwardRef"* として DevTools に表示されます。

```
const WrappedComponent = React.forwardRef((props, ref) => {
  return <LogProps {...props} forwardedRef={ref} />;
});
```

render 関数に名前をつけると、DevTools はその名前を含めるようになります（例：*"ForwardRef(myFunction)"*）：

```
const WrappedComponent = React.forwardRef(
  function myFunction(props, ref) {
    return <LogProps {...props} forwardedRef={ref} />;
  }
);
```

ラップしているコンポーネントを含めるために、render 関数の `displayName` を設定することもできます：

```
function logProps(Component) {
  class LogProps extends React.Component {
    // ...
  }

  function forwardRef(props, ref) {
    return <LogProps {...props} forwardedRef={ref} />;
  }

  // Give this component a more helpful display name in DevTools.
  // e.g. "ForwardRef(logProps(MyComponent))"
  const name = Component.displayName || Component.name;
  forwardRef.displayName = `logProps(${name})`;

  return React.forwardRef(forwardRef);
}
```

このページを編集する