Ref と DOM

Ref は render メソッドで作成された DOM ノードもしくは React の要素にアクセスする方法を提供します。
一般的な React のデータフローでは、props が、親コンポーネントがその子要素とやりとりする唯一の方法です。子要素を変更するには、新しい props でそれを再レンダーします。ただし、この一般的なデータフロー以外で、子要素を命令型のコードを使って変更する必要がある場合もあります。変更したい子要素が React コンポーネントのインスタンスのことも、DOM 要素のこともあるでしょう。どちらの場合でも、React は避難ハッチを提供します。

いつ Ref を使うか

Ref に適した使用例は以下の通りです。

- フォーカス、テキストの選択およびメディアの再生の管理
- アニメーションの発火
- サードパーティの DOM ライブラリとの統合

宣言的に行えるものには ref を使用しないでください。
例えば、`Dialog` コンポーネントに `open()` と `close()` メソッドを実装するかわりに、`isOpen` プロパティを渡してください。

Ref を使いすぎない

最初はアプリ内で「何かを起こす」ために ref を使いがちかもしれません。そんなときは、少し時間をかけて、コンポーネントの階層のどこで状態を保持すべきかについて、よりしっかりと考えてみてください。多くの場合、その状態を「保持する」ための適切な場所は階層のより上位にあることが明らかになるでしょう。具体例については state のリフトアップガイドを参照してください。

> **補足**
> 以下の例は React 16.3 で導入された `React.createRef()` API を使うように更新されました。以前のリリースの React を使用している場合は、代わりに callback ref を使用することをおすすめします。

Ref を作成する

Ref は `React.createRef()` を使用して作成され、`ref` 属性を用いて React 要素に紐付けられます。Ref は通常、コンポーネントの構築時にインスタンスプロパティに割り当てられるため、コンポーネントを通して参照が可能です。

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

Ref へのアクセス

ref が `render` メソッドの要素に渡されると、そのノードへの参照は ref の `current` 属性でアクセスできるようになります。

```
const node = this.myRef.current;
```

ref の値はノードの種類によって異なります。

- HTML 要素に対して `ref` 属性が使用されている場合、`React.createRef()` を使ってコンストラクタ内で作成された `ref` は、その `current` プロパティとして根底にある DOM 要素を受け取ります
- `ref` 属性がカスタムクラスコンポーネントで使用されるとき、`ref` オブジェクトはコンポーネントのマウントされたインスタンスを `current` として受け取ります
- **関数コンポーネント (function components) には `ref` 属性を使用してはいけません。**なぜなら、関数コンポーネントはインスタンスを持たないからです

以下の例ではその違いを示しています。

DOM 要素への Ref の追加

このコードでは DOM ノードへの参照を保持するために `ref` を使います。

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);
    // textInput DOM 要素を保持するための ref を作成します。
```

```
    this.textInput = React.createRef();
    this.focusTextInput = this.focusTextInput.bind(this);
  }

  focusTextInput() {
    // 生の DOM API を使用して明示的にテキストの入力にフォーカスします。
    // 補足：DOM ノードを取得するために "current" にアクセスしています。
    this.textInput.current.focus();
  }

  render() {
    // コンストラクタで作成した `textInput` に <input> ref を関連付けることを
    // React に伝えます。
    return (
      <div>
        <input
          type="text"
          ref={this.textInput} />

        <input
          type="button"
          value="Focus the text input"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}
```

コンポーネントがマウントされると React は `current` プロパティに DOM 要素を割り当て、マウントが解除されると `null` に戻します。`ref` の更新は `componentDidMount` または `componentDidUpdate` ライフサイクルメソッドの前に行われます。

クラスコンポーネントへの Ref の追加

マウント直後にクリックされることをシミュレーションするために上記の CustomTextInput をラップしたい場合は、ref を使用してカスタムインプットにアクセスし、その `focusTextInput` メソッドを手動で呼び出せます。

```
class AutoFocusTextInput extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }

  componentDidMount() {
    this.textInput.current.focusTextInput();
  }

  render() {
    return (
      <CustomTextInput ref={this.textInput} />
    );
  }
}
```

これは `CustomTextInput` がクラスとして宣言されている場合にのみ機能することに注意してください。

```
class CustomTextInput extends React.Component {
  // ...
}
```

Ref と関数コンポーネント

関数コンポーネントにはインスタンスがないため、**関数コンポーネントに `ref` 属性を使用することはできません。**

```
function MyFunctionComponent() {
  return <input />;
}

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }
  render() {
    // これは動き「ません」！
    return (
      <MyFunctionComponent ref={this.textInput} />
    );
  }
}
```

Ref が必要な場合は、ライフサイクルメソッドや state が必要なときと同じように、コンポーネントをクラスに変換しなければなりません。
ただし、DOM 要素またはクラスコンポーネントを参照している限り、**関数コンポーネント内で ref 属性を使用することはできます。**

```
function CustomTextInput(props) {
  // ref が参照できるように、textInput をここで宣言する必要があります。
  let textInput = React.createRef();

  function handleClick() {
    textInput.current.focus();
  }

  return (
    <div>
      <input
        type="text"
        ref={textInput} />

      <input
        type="button"
        value="Focus the text input"
        onClick={handleClick}
      />
    </div>
  );
}
```

DOM の Ref を親コンポーネントに公開する

まれに、親コンポーネントから子コンポーネントの DOM ノードにアクセスしたい場合があります。これは、コンポーネントのカプセル化を壊すため、一般的にはおすすめできませんが、フォーカスを発火させたり、子の DOM ノードのサイズや位置を計測するのに役立つことがあります。

子コンポーネントに ref を追加することはできますが、DOM ノードではなくコンポーネントインスタンスしか取得できないため、これは理想的な解決策ではありません。また、これは関数コンポーネントでは機能しません。

React 16.3 以降を使用している場合、これらの場合には ref forwarding を使用することをおすすめします。**Ref forwarding では、コンポーネントは任意の子コンポーネントの ref を自分のものとして公開することを選択できます。**Ref forwarding のドキュメントに、子の DOM ノードを親コンポーネントに公開する方法の詳細な例があります。

React 16.2 以下を使用している場合、または ref forwarding で提供される以上の柔軟性が必要な場合は、この代替手法を使用して ref を異なる名前の props として明示的に渡すことができます。

可能であれば DOM ノードを公開しないことをおすすめしますが、これは便利な避難ハッチになることもあります。留意すべき点として、この方法では子コンポーネントにコードを追加する必要があります。子コンポーネントの実装にまったく手を加えられない場合、最後の選択肢は `findDOMNode()` を使用することですが、おすすめできない上に、`StrictMode` では非推奨です。

コールバック Ref

React はまた「コールバック Ref」と呼ばれる、より細かい制御が可能な ref を設定するための別の方法をサポートします。

`createRef()` によって作成された `ref` 属性を渡す代わりに、関数を渡します。この関数は、引数として React コンポーネントのインスタンスまたは HTML DOM 要素を受け取ります。これを保持することで、他の場所からアクセスできます。

以下は、`ref` コールバックを用いて DOM ノードへの参照をインスタンスプロパティに格納する一般的な実装例です。

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);

    this.textInput = null;

    this.setTextInputRef = element => {
      this.textInput = element;
    };

    this.focusTextInput = () => {
      // 生の DOM API を使用して明示的にテキストの入力にフォーカスします。
      if (this.textInput) this.textInput.focus();
    };
  }

  componentDidMount() {
    // マウント時に入力をオートフォーカスします。
    this.focusTextInput();
  }

  render() {
    // インスタンスフィールド（例えば this.textInput）にテキスト入力の DOM 要素への
    // 参照を保存するために `ref` コールバックを使用してください。
    return (
      <div>
        <input
          type="text"
          ref={this.setTextInputRef}
        />
        <input
          type="button"
          value="Focus the text input"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}
```

コンポーネントがマウントされると React は DOM 要素とともに `ref` コールバックを呼び出し、マウントが解除されると `null` とともにコールバックを呼び出します。Ref は `componentDidMount` または `componentDidUpdate` が発火する前に最新のものであることが保証されています。

`React.createRef()` で作成されたオブジェクトの ref と同様に、コンポーネント間でコールバック ref を渡すことができます。

```
function CustomTextInput(props) {
  return (
    <div>
      <input ref={props.inputRef} />
    </div>
  );
}

class Parent extends React.Component {
  render() {
    return (
      <CustomTextInput
        inputRef={el => this.inputElement = el}
      />
    );
  }
}
```

上記の例では、`Parent` は ref コールバックを `inputRef` プロパティとして `CustomTextInput` に渡し、`CustomTextInput` は同じ関数を特別な `ref` 属性として `<input>` に渡します。その結果、`Parent` の `this.inputElement` は、`CustomTextInput` の `<input>` 要素に対応する DOM ノードに設定されます。

レガシー API：String Ref

以前に React を使用したことがある場合は、`ref` 属性が `"textInput"` のような文字列で、DOM ノードが `this.refs.textInput` としてアクセスされる古い API に慣れているかもしれません。String ref にはいくつかの問題があり、レガシーと見なされ、**将来のリリースのいずれかで削除される可能性が高い**ため、使用することをおすすめしません。

> **補足**
>
> Ref にアクセスするために `this.refs.textInput` を現在使用している場合は、代わりにコールバックパターンもしくは `createRef API` を使用することをおすすめします。

コールバック Ref の注意事項

`ref` コールバックがインライン関数として定義されている場合、更新中に 2 回呼び出されます。最初は `null`、次に DOM 要素で呼び出されます。これは、それぞれのレンダーで関数の新しいインスタンスが作成されるため、React は古い ref を削除し、新しい ref を設定する必要があるためです。`ref` コールバックをクラス内のバインドされたメソッドとして定義することでこれを回避できますが、ほとんどの場合は問題にならないはずです。

このページを編集する