フォーム

自然な HTML のフォーム要素は内部に何らかの状態を持っていますので、フォーム要素は React において他の DOM 要素とちょっと異なる動作をします。例えば、このプレーン HTML によるフォームは 1 つの名前を受け付けます：

```
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

このフォームは、ユーザーがフォームを送信した際に新しいページに移動する、という、HTML フォームとしてのデフォルトの動作をします。React でこの振る舞いが必要なら、そのまま動きます。しかし大抵のケースでは、フォームの送信に応答してユーザーがフォームに入力したデータにアクセスするような JavaScript 関数があった方が便利です。これを実現する標準的な方法は、“制御された (controlled) コンポーネント” と呼ばれるテクニックを使うことです。

制御されたコンポーネント

HTML では `<input>`、`<textarea>`、そして `<select>` のようなフォーム要素は通常、自身で状態を保持しており、ユーザーの入力に基づいてそれを更新します。React では、変更されうる状態は通常はコンポーネントの state プロパティに保持され、`setState()` 関数でのみ更新されます。
React の state を “信頼できる唯一の情報源 (single source of truth)” とすることで、上述の 2 つの状態を結合させることができます。そうすることで、フォームをレンダーしている React コンポーネントが、後続するユーザー入力でフォームで起きることも制御できるようになります。このような方法で React によって値が制御される入力フォーム要素は「制御されたコンポーネント」と呼ばれます。
例えば、前述のフォームの例において、フォーム送信時に名前をログに残すようにしたい場合、フォームを制御されたコンポーネントとして書くことができます：

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

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

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Try it on CodePen

フォーム要素の `value` 属性が設定されているので、表示される値は常に `this.state.value` となり、React の state が信頼できる情報源となります。`handleChange` はキーストロークごとに実行されて React の state を更新するので、表示される値はユーザーがタイプするたびに更新されます。
制御されたコンポーネントを使うと、すべての state の変更には紐付けられたハンドラー関数が存在することになります。これによりユーザー入力の改変や検証が簡単になります。例えば、名前が全て大文字で書かれるように強制したいなら、`handleChange` を次のように書くことができます：

```
handleChange(event) {
  this.setState({value: event.target.value.toUpperCase()});
}
```

textarea タグ

HTML では、<textarea> 要素はテキストを子要素として定義します。

```
<textarea>
  Hello there, this is some text in a text area
</textarea>
```

React では、<textarea> は代わりに value 属性を使用します。こうすることで、<textarea> を使用するフォームは単一行の入力フォームと非常に似た書き方ができるようになります：

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

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

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Essay:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

this.state.value がコンストラクタで初期化されているので、テキストエリアには始めからテキストが入っていることに注意してください。

select タグ

HTML では、<select> はドロップダウンリストを作成します。例えばこの HTML は味についてのドロップダウンリストを作成しています：

```
<select>
  <option value="grapefruit">Grapefruit</option>
  <option value="lime">Lime</option>
  <option selected value="coconut">Coconut</option>
  <option value="mango">Mango</option>
</select>
```

selected 属性があるため Coconut オプションが最初に選択されていることに注意してください。この selected 属性の代わりに React は value 属性を親の select タグで使用します。一箇所で更新すればよいだけなので、制御されたコンポーネントを使う場合にはこちらがより便利です。 例えば：

```
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

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

  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
```

```
        <label>
          Pick your favorite flavor:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Try it on CodePen

全体的に見て `<input type="text">`、`<textarea>`、そして `<select>` が非常に似た動作をするようになっています。これらはすべて、制御されたコンポーネントを実装する時に使うことができる `value` 属性を受け取ります。

補足

`value` 属性に配列を渡すことで、`select` タグ内の複数のオプションを選択することができます：

```
<select multiple={true} value={['B', 'C']}>
```

---

file input タグ

HTML では、`<input type="file">` によってユーザにデバイス内の 1 つ以上のファイルを選ばせて、それをサーバにアップロードしたり File API を使って JavaScript で操作したりすることができます。

```
<input type="file" />
```

この値は読み取り専用ですので、これは**非制御**コンポーネントになります。このドキュメントの後の方で、他の非制御コンポーネントと併せて説明しています。

複数の入力の処理

複数の制御された `input` 要素を処理する必要がある場合、それぞれの入力要素に `name` 属性を追加すれば、ハンドラー関数に `event.target.name` に基づいて処理を選択させるようにできます。例えば：

```
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };

    this.handleInputChange = this.handleInputChange.bind(this);
  }

  handleInputChange(event) {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;

    this.setState({
      [name]: value
    });
  }

  render() {
    return (
      <form>
        <label>
          Is going:
          <input
            name="isGoing"
            type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleInputChange} />
        </label>
        <br />
        <label>
          Number of guests:
          <input
            name="numberOfGuests"
            type="number"
            value={this.state.numberOfGuests}
```

```
            onChange={this.handleInputChange} />
        </label>
      </form>
    );
  }
}
```

Try it on CodePen

渡された入力名に対応する state のキーを更新するのに用いた ES6 の computed property name 構文の使い方に注意してください：

```
this.setState({
  [name]: value
});
```

これは以下の ES5 のコードと同等です：

```
var partialState = {};
partialState[name] = value;
this.setState(partialState);
```

また、`setState()` は自動的に部分的な state を現在の state にマージするので、変更された部分のみで呼び出せば大丈夫です。

制御された入力における null 値

制御されたコンポーネントで value プロパティに値を指定することで、変更させたくない場合にユーザーが値を変更できないようになります。もしも `value` を指定したのに入力フィールドが依然変更可能であるという場合は、`value` を誤って `undefined` もしくは `null` に設定してしまったのかもしれません。
以下のコードでこれを示しています。（入力フィールドは最初はロックされていますが、短い遅延の後に編集可能になります）

```
ReactDOM.render(<input value="hi" />, mountNode);

setTimeout(function() {
  ReactDOM.render(<input value={null} />, mountNode);
}, 1000);
```

制御されたコンポーネントの代替手段

制御されたコンポーネントは、あらゆる種類のデータの変更に対してイベントハンドラを書き、あらゆる入力状態を React コンポーネントに通してやる必要があるため、時としてうんざりすることがあります。このことは既存のコードベースを React に変換する場合や、React アプリケーションを非 React のライブラリと統合する場合に、特に問題化します。これらの状況においては、入力フォームを実装する代替手段である非制御コンポーネントを検討してみてください。

本格的なソリューション

入力値のバリデーション、訪問済みフィールドの追跡やフォーム送信を含む完全なソリューションをお探しの場合は、Formik が人気のある選択肢のひとつです。しかしながらこれは制御されたコンポーネントや state の管理と同じ原理で作成されていますので、これらについて学ぶことを無視しないようにしましょう。

このページを編集する