コンポーネントの state

`setState` は何をしているのですか？

`setState()` はコンポーネントの `state` オブジェクト更新をスケジュールします。state が更新されると、コンポーネントはそれに再レンダーで応じます。

`state` と `props` の違いは何ですか？

`props`（"properties" を短くしたもの）と `state` は、両方ともプレーンな JavaScript のオブジェクトです。どちらもレンダー結果に影響を及ぼす情報を持ってはいますが、ある重要な一点が異なっています。つまり、`props` は（関数引数のように）コンポーネント**へ**渡されるのに対し、`state` は（関数内で宣言された変数のように）コンポーネント**の内部**で制御されます。
`props` と `state` のどちらをいつ使うべきかについて、こちらでより詳しく読むことができます。

- Props vs State
- ReactJS: Props vs. State

`setState` が誤った値を返すのはなぜですか？

React では、`this.props` と `this.state` のいずれも、**レンダーされた**もの、つまりスクリーン上の値を表しています。
`setState` 呼び出しは非同期です。呼び出し直後から `this.state` が新しい値を反映することを期待しないでください。もし現在の state に基づいた値を計算する必要がある場合は、オブジェクトの代わりに更新関数を渡してください。（詳しくは以下を参照）
このコード例は期待した通りには**動きません**。

```
incrementCount() {
  // 補足：これは意図通りに*動きません*
  this.setState({count: this.state.count + 1});
}

handleSomething() {
  // `this.state.count` は 0 から始まるとします。
  this.incrementCount();
  this.incrementCount();
  this.incrementCount();
  // React がコンポーネントを再レンダーしても、`this.state.count` は意図通りの 3 ではなく 1 になります。

  // これは、上記の `incrementCount()` 関数は `this.state.count` の値を読むのですが、
  // しかしコンポーネントが再レンダーされるまで React が `this.state.count` を更新しないためです。
  // そして `incrementCount()` は値が 0 のままの `this.state.count` を毎回読み、そして 1 をセットしてしまいます。

  // 対処法は下で説明していますよ！
}
```

この問題を解決するには以下を見てください。

どうやって現在の state に依存する値を更新したらいいですか？

`setState` へオブジェクトを渡す代わりに関数を渡してください。その関数は常に最新の状態の state を使って呼ばれることが保証されています。（次項参照）

`setState` へオブジェクトを渡すのと関数を渡すのとでは何が違いますか？

更新関数を渡すと、その関数内で現在の state の値へアクセスできるようになります。`setState` 呼び出しはバッチ処理されるため、更新処理を連結して、それぞれの更新が競合せずに順序だって動作することが保証されます。

```
incrementCount() {
  this.setState((state) => {
    // 重要：更新には `this.state` ではなく `state` を使います。
    return {count: state.count + 1}
  });
}

handleSomething() {
  // `this.state.count` は 0 から始まるとします。
  this.incrementCount();
  this.incrementCount();
  this.incrementCount();

  // ここで `this.state.count` を読んでもまだ 0 のままです。
  // しかし React がコンポーネントを再レンダーするとき、値は 3 になります。
}
```

setState についてもっと学ぶ

いつ `setState` は非同期になりますか？

現在、`setState` はイベントハンドラの内側では非同期です。

例えばクリックイベントの間に `Parent` と `Child` の両方が `setState` を呼ぶとき、非同期処理のおかげで `Child` が 2 度レンダーされないことが保証されます。その代わりに React はブラウザイベントの最後に state の更新を「フラッシュ (flush)」します。これにより大規模アプリのパフォーマンスが大幅に向上します。

これは実装の詳細ですので、この仕組みに直接依存しないようにしてください。将来のバージョンにおいて、React はより多くの場合にバッチ更新するようになります。

どうして React は `this.state` を同期的に更新しないのですか？

前項で説明したように、全てのコンポーネントがそのイベントハンドラ内で `setState()` を呼ぶまで、React は再レンダー前に意図的に「待つ」ようになっています。これにより不必要な再レンダーが防がれ、パフォーマンスが向上します。

とはいえ、React がどうして再レンダーなしに `this.state` を即時更新しないのか、まだ疑問に思っているかもしれません。

これには主に 2 つの理由があります。

- 同期的更新が `props` と `state` の間の一貫性を破壊し、非常にデバッグが難しい問題を引き起こしうるため。
- 同期的更新が、我々が取り組んでいる新機能のいくつかを実装不可能にしうるため。

この GitHub コメントは特定の例について詳しく解説しています。

Redux や MobX のような state 管理ライブラリを使うべきでしょうか？

時には必要かもしれません。

まずは他のライブラリを追加する前に React を理解することをお勧めします。React だけでも非常に複雑なアプリケーションを作り上げることができます。

このページを編集する