ステートフックの利用法

フック (hook) は React 16.8 で追加された新機能です。state などの React の機能を、クラスを書かずに使えるようになります。
以前のページで以下の例を挙げてフックの紹介を行いました：

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

このコードをクラスによる等価版と比較しながらフックについて学び始めましょう。

クラスによる同等の例

以前 React でクラスを使っていたなら、このコードに馴染みがあるでしょう。

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

state は `{ count: 0 }` から始まり、ユーザがボタンをクリックした時に `this.setState()` を呼ぶことで `state.count` をインクリメントします。このページの全体にわたってこのクラスからの例が出てきます。

> **補足**
> もっと現実的な例ではなくカウンタを使っているのはなぜか気になるかもしれません。フックについての第一歩の説明ですので API にフォーカスするためです。

フックと関数コンポーネント

念のため、React の関数コンポーネントとはこのようなものです：

```
const Example = (props) => {
  // You can use Hooks here!
  return <div />;
}
```

あるいはこのようなものです：

```
function Example(props) {
  // You can use Hooks here!
```

```
  return <div />;
}
```

これらのことを「ステートレスコンポーネント (stateless component)」だと理解していたかもしれません。これらの内部で React の state を利用できるようにしていますので、「関数コンポーネント (function component)」という名前を利用します。
フックはクラスの内部では動作**しません**。クラスを書く代わりに使うものです。

フックとは何か？

この新しい例では、React から `useState` をインポートするところから始めましょう。

```
import React, { useState } from 'react';

function Example() {
  // ...
}
```

Hook とは何？ フックとは React の機能に「接続 (hook into)」するための特別な関数です。例えば `useState` によって React の state の機能を関数コンポーネントに追加できます。他のフックについても後で学びます。
フックをいつ使うべき？ 関数コンポーネントを書いていて state が必要だと気付いた場合、これまではコンポーネントをクラスに変換する必要がありました。今後は、既に書いた関数コンポーネントの中でフックを使うことができます。早速やってみましょう！

> **補足：**
> コンポーネント内のどこでフックが使えてどこで使えないかに関する特別なルールがあります。これについてはフックのルールで学びます。

---

state 変数の宣言

クラスでは、コンストラクタ内で `this.state` を `{ count: 0 }` にセットするという方法で、state である `count` を `0` へと初期化します。

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

関数コンポーネントには `this` は存在しませんので、`this.state` を読み書きすることはできません。その代わりに、コンポーネントの内部で直接 `useState` フックを使いましょう。

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

`useState` を呼ぶと何が起きるの？ これにより『state 変数』が宣言されます。我々の例では変数を `count` と名付けましたが、`banana` でも何でも好きな名前にすることができます。これが関数コールの間で値を『保持』しておくための方法です — `useState` は、クラスにおいて `this.state` が提供するのと全く同じ機能を実現するための新しい方法です。通常、関数が終了すると変数は『消えて』しまいますが、state 変数は React によって保持されます。
引数として `useState` に何を渡すのか？ `useState()` フックの唯一の引数は state の初期値です。クラスの場合とは異なり、state はオブジェクトである必要はありません。数字や文字列が欲しいだけであればそれらを保持することができます。我々の例ではユーザがクリックした回数に対応する数字が欲しいだけですので、`0` を state 変数の初期値として渡します。（もし違う値を保持したい場合は `useState()` を 2 回呼ぶことになります）
`useState` は何を返すのか？ 現在の state と、それを更新するための関数とを、ペアにして返します。これが `const [count, setCount] = useState()` という書き方をした理由です。これはクラスにおける `this.state.count` と `this.setState` に似ていますが、それをペアとして手に入れる点が異なります。もしもここで使った構文になじみがない場合は、このページの下部で改めて説明します。
`useState` が何をやるのかが分かったので、最初の例の意味がより分かりやすくなりました：

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

`count` という名前の state 変数を宣言しており、それを `0` にセットします。再レンダー間で React はその変数の現在値を記憶しており、最新の値を関数に渡します。現在の `count` の値を更新したい場合は、`setCount` を呼ぶことができます。

> **補足**
> どうして `createState` ではなく `useState` という名前なのか気になるでしょうか？

state が「作成」されるのはコンポーネントの初回レンダー時だけですので、 `createState` という名前はあまり正確ではありません。次回以降のレンダー時には、 `useState` からは既存の state の現在値を受け取ります。毎回作成していたのではそもそも「状態」になりませんね。また、フックの名前が**常に** `use` から始まることには理由があります。フックのルールで改めて説明します。

---

state の読み出し

クラス内で現在のカウント値を表示したい場合、 `this.state.count` を読み出します：

```
<p>You clicked {this.state.count} times</p>
```

関数内では、直接 `count` を使うことができます：

```
<p>You clicked {count} times</p>
```

---

state の更新

クラスでは、 `this.setState()` を呼ぶことで `count` ステートを更新します：

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>
  Click me
</button>
```

関数内では、すでに `setCount` と `count` を変数として受け取っていますので、 `this` は必要ありません：

```
<button onClick={() => setCount(count + 1)}>
  Click me
</button>
```

まとめ

では **これまで学んだことを 1 行ずつまとめて**、理解を確認しましょう。

```
 1:  import React, { useState } from 'react';
 2:
 3:  function Example() {
 4:    const [count, setCount] = useState(0);
 5:
 6:    return (
 7:      <div>
 8:        <p>You clicked {count} times</p>
 9:        <button onClick={() => setCount(count + 1)}>
10:          Click me
11:        </button>
12:      </div>
13:    );
14:  }
```

- **1 行目：** `useState` フックを React からインポートします。これが関数コンポーネント内でローカル state を使えるようにするためのものです。
- **4 行目：** `Example` コンポーネント内で `useState` フックを呼び出すことで新しい state 変数を宣言します。2 つの値のペアが返されるので、それらに名前を与えます。ボタンのクリック回数を保持するための変数ですので `count` と名付けましょう。`useState` 唯一の引数として `0` を渡すことで、変数をゼロへと初期化します。返り値の 2 つ目はそれ自体が関数です。これにより `count` を更新するので、 `setCount` とう名前にします。
- **9 行目：** ユーザがクリックした時に、新しい値で `setCount` を呼びます。React は `Example` コンポーネントを再レンダーし、その際には新たな `count` の値を渡します。

最初は飲み込むのに時間がかかるかもしれません。急がないようにしましょう！ 途中で分からなくなった場合は上記のコードを最初から最後まで読み直してください。一旦クラスで state がどう動くのかを「忘れて」新鮮な目でこのコードを見るようにすれば、必ず理解できるようになるはずです。

ヒント：この角カッコの意味は？

state 変数を宣言するときのこの角カッコに気付かれたでしょうか：

```
const [count, setCount] = useState(0);
```

左辺に書かれている名前は、React の API の一部ではありません。state 変数には好きな名前を付けることができます：

```
const [fruit, setFruit] = useState('banana');
```

この JavaScript の構文は "分割代入 (destructuring)" と呼ばれています。これが意味するところは、`fruit` と `setFruit` という名前の 2 つの変数を作って、`useState` から返される値のうち 1 つ目を `fruit` に、2 つ目を `setFruit` に代入する、ということです。これは以下のコードと等価です：

```
var fruitStateVariable = useState('banana'); // Returns a pair
var fruit = fruitStateVariable[0]; // First item in a pair
var setFruit = fruitStateVariable[1]; // Second item in a pair
```

`useState` で state 変数を宣言する際、ペア、つまり 2 つの要素を含んだ配列が返されます。1 つ目の要素は state の現在の値、2 つ目の要素はそれを更新するための関数です。これらには特定の意味があるので、アクセスするのに `[0]` や `[1]` を使うのではちょっと分かりづらくなります。だから代わりに分割代入を使うというわけです。

> **補足**
> React に対して `this` のようなものを一切渡していないので、どのようにコンポーネントと `useState` の呼び出しの対応を知るのか不思議に思うかもしれません。FAQ セクションで、この質問およびその他の疑問についてお答えしています。

ヒント：複数の state 変数を使う
state 変数の宣言を `[something, setSomething]` のペアの形で行うのが便利であるもうひとつの理由は、state 変数を複数使いたくなった場合にそれらに**異なる**名前をつけることができるからです：

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

上記の例では、ローカル変数として `age`、`fruit`、`todos` があり、それぞれを個別に更新することができます。

```
  function handleOrangeClick() {
    // Similar to this.setState({ fruit: 'orange' })
    setFruit('orange');
  }
```

たくさんの state 変数を使う**必要は**ありません。state 変数はオブジェクトや配列も何ら問題なく保持できますので、関連する値をいっしょにまとめておくこともできます。しかしクラスでの `this.setState` とは異なり、state 変数の更新は、マージではなく必ず古い値を**置換**します。
独立した state 変数を分割する際の推奨事項については FAQ で詳しく述べています。

次のステップ

このページでは React によって提供されるフックのうちのひとつである `useState` について学びました。これからは「ステートフック」という名前でも呼ぶことにします。ステートフックによって、React の歴史上はじめて、React の関数コンポーネントにローカル state を加えることができます！
またフックが一体何なのかについても少々学びました。フックとは関数コンポーネント内から React の機能に「接続する (hook into)」ための関数のことです。フックの名前は必ず `use` から始まり、他にもまだ説明していない様々なフックがあります。
それでは次のフックである `useEffect` について学びましょう。これらは関数におけるライフサイクルメソッドと似ており、コンポーネント内で副作用を実行することができるようにするものです。

このページを編集する