カスタムフックの作成

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。
自分独自のフックを作成することで、コンポーネントからロジックを抽出して再利用可能な関数を作ることが可能です。
以下のコンポーネントは副作用フックの使い方について学んだ際に見たチャットアプリのコンポーネントであり、フレンドがオンラインかオフラインかを示すメッセージを表示します。

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

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

さて、このチャットアプリには連絡先リストもあって、そこではオンラインのユーザを緑色で表示したいとしましょう。新しい `FriendListItem` コンポーネントに似たようなロジックをコピーペーストしても構いませんが、それは理想的ではないでしょう：

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

function FriendListItem(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

代わりに、このロジックを `FriendStatus` と `FriendListeItem` 間で共有したいと思います。
これまで React では、ステートを有するロジックをコンポーネント間で共有するための人気の手法が 2 つありました。レンダープロップ と 高階コンポーネントです。ツリーに新しいコンポーネントを加える必要なしに、フックが同じ問題をどのように解決するのかを見ていきましょう。

カスタムフックの抽出

2 つの JavaScript の関数間でロジックを共有したい場合、それを別の関数に抽出します。コンポーネントもフックも関数ですので、同じ方法が使えます！
カスタムフックとは、名前が "use" で始まり、ほかのフックを呼び出せる JavaScript の関数のことです。例えば、以下の `useFriendStatus` が我々の最初のカスタムフックの例です：

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
```

```
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

新しいことは何もありません。ロジックは上記のコンポーネントからコピーしてきただけです。コンポーネントの時と同様に、他のフックを呼ぶときはカスタムフックのトップレベルで無条件に呼び出していることを確認してください。
React のコンポーネントと違い、カスタムフックは特定のシグネチャを持つ必要はありません。何を引数として受け取り、そして（必要なら）何を返すのか、といったことは自分で決めることができます。別の言い方をすると、普通の関数と同じだということです。一目でフックのルールが適用されるものだと分かるようにするために、名前は `use` で始めるべきです。
この `useFriendStatus` の目的はフレンドのステータスを購読するというものです。ですので `friendID` を引数として持ち、そのフレンドがオンラインかどうかを返します。

```
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  // ...

  return isOnline;
}
```

ではこのカスタムフックの使い方を見ていきましょう。

カスタムフックを使う

そもそもの我々の目的は `FriendStatus` と `FriendListItem` コンポーネントでの重複したロジックを取り除くことでした。どちらのコンポーネントもフレンドがオンラインかどうかを知りたいのです。
既にロジックを `useFriendStatus` フックへと抽出したので、それを**ただ単に使えばいい**のです：

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

```
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

このコードは元のコードと同等？ はい、全く同じように動作します。注意深く見れば、ふるまいに何の変更も加えていないということが分かります。やったのは、共通のコードを別の関数に抽出したということだけです。**カスタムフックは React の機能というよりは、フックの設計から自然と導かれる慣習のようなものです。**
カスタムフックは “use” という名前で始めるべき？ ぜひそうしてください。この規約はとても重要です。この規約がなければ、ある関数が内部でフックを呼んでいるかどうかを知る方法がなくなり、フックのルールの違反を自動でチェックすることができなくなります。
同じフックを使うコンポーネントは state を共有する？ いいえ。カスタムフックは **state を使うロジック**（データの購読を登録したり現在の値を覚えておいたり）を共有するためのものですが、カスタムフックを使う場所ごとで、内部の state や副作用は完全に分離しています。
どのようにしてカスタムフックは独立したステートを得るのか？ それぞれのフックの**呼び出し**が独立した state を得ます。`useFriendStatus` を直接呼びだしていますので、React から見れば我々のコンポーネントが `useState` や `useEffect` を呼んだ場合と変わりません。すでにここやここで学んだ通り、`useState` や `useEffect` はひとつのコンポーネント内で複数回呼ぶことができ、それらは完全に独立しています。

ヒント：フック間で情報を受け渡す

フックは関数ですので、フック間で情報を受け渡すことができます。
これを例示するため、我々のチャットの例で、別のコンポーネントを使うことにしましょう。これはチャットの受信者を選ぶ画面であり、現在選択中のフレンドがオンラインかどうかを表示します。

```
const friendList = [
  { id: 1, name: 'Phoebe' },
  { id: 2, name: 'Rachel' },
  { id: 3, name: 'Ross' },
];

function ChatRecipientPicker() {
  const [recipientID, setRecipientID] = useState(1);
  const isRecipientOnline = useFriendStatus(recipientID);
```

```
  return (
    <>
      <Circle color={isRecipientOnline ? 'green' : 'red'} />
      <select
        value={recipientID}
        onChange={e => setRecipientID(Number(e.target.value))}
      >
        {friendList.map(friend => (
          <option key={friend.id} value={friend.id}>
            {friend.name}
          </option>
        ))}
      </select>
    </>
  );
}
```

現在選択中のフレンド ID を `recipientID` という state 変数に保持し、`<select>` ピッカー内で別のフレンドが選択されるごとにそれを更新します。
`useState` フックは `recipientID` という state 変数の最新の値を返しますので、それを `useFriendStatus` カスタムフックに引数として渡すことができます。

```
  const [recipientID, setRecipientID] = useState(1);
  const isRecipientOnline = useFriendStatus(recipientID);
```

これにより**現在選択中の**フレンドがオンラインかどうかが分かります。別のフレンドを選択して `recipientID` 変数が更新された場合、`useFriendStatus` フックはこれまで選択されていたフレンドを購読解除して、新しく選択されたフレンドのステータスを購読開始します。

---

useYourImagination()

カスタムフックにより、これまでの React コンポーネントでは不可能であった、ロジック共有に関する柔軟性が得られます。フォーム操作、アニメーション、宣言的データ購読、タイマー、さらには我々が思いついたことのない多様なユースケースに対するカスタムフックを記述することが可能です。何より、作ったカスタムフックは React の組み込み機能と同じくらい簡単に使えるようになるのです。
あまり焦って抽象化を加えないようにしましょう。関数コンポーネントがやれることが増えたので、平均的な関数コンポーネントはこれまでより長いものになるでしょう。それは普通のことですので、いますぐカスタムフックに分割しないと**いけない**とは考えないでください。一方で、カスタムフックをどこで使えば複雑なロジックをシンプルなインターフェースに置き換えたり、ごちゃっとしたコンポーネントを整理したりできるのか、考え始めることをお勧めします。
一例として、その場しのぎで多くのローカル state が含まれるようになった複雑なコンポーネントをお持ちかもしれません。`useState` を使っても更新ロジックの集中化が簡単になるわけではありませんので、それを Redux のリデューサ (reducer) で書きたくなることがあるでしょう：

```
function todosReducer(state, action) {
  switch (action.type) {
    case 'add':
      return [...state, {
        text: action.text,
        completed: false
      }];
    // ... other actions ...
    default:
      return state;
  }
}
```

リデューサは単独でのテストが非常にやりやすく、複雑な更新ロジックを表現する場合でもスケールします。必要に応じて後でより小さなリデューサに分割することも可能です。しかし、React のローカル state による手軽さの方が好ましい場合もあるでしょうし、他のライブラリをインストールしたくない場合もあるでしょう。
そこで、`useReducer` というフックを書いて、コンポーネントの**ローカル** state をリデューサで管理できるとしたらどうでしょうか？ 簡略版では以下のようなものになるでしょう：

```
function useReducer(reducer, initialState) {
  const [state, setState] = useState(initialState);

  function dispatch(action) {
    const nextState = reducer(state, action);
    setState(nextState);
  }

  return [state, dispatch];
}
```

これをコンポーネント内で使うことができ、リデューサを活用してステート管理ができるようになります：

```
function Todos() {
  const [todos, dispatch] = useReducer(todosReducer, []);

  function handleAddClick(text) {
    dispatch({ type: 'add', text });
  }

  // ...
}
```

ローカルステートをリデューサで管理したいという要求はとてもよくあるので、React にその機能を含めてあります。Hooks API reference のページで他のビルトインフックと共に解説しています。

このページを編集する