ES6 なしで React を使う

通常、React コンポーネントはプレーンな JavaScript クラスとして定義されます。

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

もしあなたがまだ ES6 を使っていないのであれば、代わりに `create-react-class` モジュールを使うことができます。

```
var createReactClass = require('create-react-class');
var Greeting = createReactClass({
  render: function() {
    return <h1>Hello, {this.props.name}</h1>;
  }
});
```

ES6 のクラス API は `createReactClass()` とよく似ていますが、いくつかの例外があります。

デフォルト props の宣言

関数や ES6 クラスでは、`defaultProps` はコンポーネント自体のプロパティとして定義されます。

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

Greeting.defaultProps = {
  name: 'Mary'
};
```

`createReactClass()` の場合、渡されるオブジェクト内の関数として `getDefaultProps()` を定義する必要があります。

```
var Greeting = createReactClass({
  getDefaultProps: function() {
    return {
      name: 'Mary'
    };
  },

  // ...

});
```

---

state の初期値の設定

ES6 クラスでは、コンストラクタで `this.state` へ代入することで state の初期値を定義できます。

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {count: props.initialCount};
  }
  // ...
}
```

`createReactClass()` の場合、state の初期値を返す `getInitialState` メソッドを別途用意しなければなりません。

```
var Counter = createReactClass({
  getInitialState: function() {
    return {count: this.props.initialCount};
  },
  // ...
});
```

自動バインド

ES6 クラスとして宣言された React コンポーネントでは、メソッドは通常の ES6 クラスと同様のセマンティクスに従います。つまり、`this` がそのインスタンスへ自動的にバインドされることはありません。コンストラクタで明示的に `.bind(this)` を利用する必要があります。

```
class SayHello extends React.Component {
  constructor(props) {
    super(props);
    this.state = {message: 'Hello!'};
    // This line is important!
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    alert(this.state.message);
  }

  render() {
    // Because `this.handleClick` is bound, we can use it as an event handler.
    return (
      <button onClick={this.handleClick}>
        Say hello
      </button>
    );
  }
}
```

`createReactClass()` の場合は全てのメソッドがバインドされるため、明示的なバインドは必要ありません。

```
var SayHello = createReactClass({
  getInitialState: function() {
    return {message: 'Hello!'};
  },

  handleClick: function() {
    alert(this.state.message);
  },

  render: function() {
    return (
      <button onClick={this.handleClick}>
        Say hello
      </button>
    );
  }
});
```

これはつまり、ES6 クラスで書くとイベントハンドラのための定型文が少し多くなってしまうということなのですが、一方では大きなアプリケーションの場合にわずかながらパフォーマンスが向上するという側面もあります。
この定型文的コードがあまりに醜く感じられる場合、Babel を使って**実験的**な Class Properties 構文提案を有効にするとよいかもしれません。

```
class SayHello extends React.Component {
  constructor(props) {
    super(props);
    this.state = {message: 'Hello!'};
  }
  // WARNING: this syntax is experimental!
  // Using an arrow here binds the method:
  handleClick = () => {
    alert(this.state.message);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Say hello
      </button>
    );
  }
}
```

上記の構文は**実験的**なものであり、構文が変わるかもしれないこと、あるいは言語に取り入れられないかもしれないことに留意してください。
安全にやりたい場合は他の選択肢もあります。

- コンストラクタでメソッドをバインドする。
- 例えば `onClick={(e) => this.handleClick(e)}` のような形でアロー関数を利用する。
- 引き続き `createReactClass` を利用する。

ミックスイン

> **補足：**
> ES6 はミックスインのサポートを含んでいません。従って、React を ES6 クラスで使う場合にミックスインのサポートはありません。
> **加えて、ミックスインを用いたコードによる多くの問題が見つかっており、新規コードで利用することは推奨されません。**
> この節は参考のためだけに存在します。

時には同じ機能が全く異なるコンポーネント間で共有されることがあります。これは横断的関心事 (cross-cutting concerns) と呼ばれることがあります。`createReactClass` であれば、横断的関心事のためにレガシーな `mixins` 機能を使うことができます。
よくある利用例のひとつは、一定時間ごとに自分自身を更新するコンポーネントです。`setInterval()` を使うのは簡単ですが、その場合はメモリ節約のため、コンポーネントが不要になった際にキャンセルすることが重要です。React はライフサイクルメソッドを提供しており、コンポーネントが生成、破棄されるときに知らせてくれます。次のようなシンプルなミックスインを作ってみましょう。このミックスインのメソッドは簡単な `setInterval()` 機能を提供し、コンポーネントが破棄されるタイミングで自動的にクリーンアップされます。

```
var SetIntervalMixin = {
  componentWillMount: function() {
    this.intervals = [];
  },
  setInterval: function() {
    this.intervals.push(setInterval.apply(null, arguments));
  },
  componentWillUnmount: function() {
    this.intervals.forEach(clearInterval);
  }
};

var createReactClass = require('create-react-class');

var TickTock = createReactClass({
  mixins: [SetIntervalMixin], // Use the mixin
  getInitialState: function() {
    return {seconds: 0};
  },
  componentDidMount: function() {
    this.setInterval(this.tick, 1000); // Call a method on the mixin
  },
  tick: function() {
    this.setState({seconds: this.state.seconds + 1});
  },
  render: function() {
    return (
      <p>
        React has been running for {this.state.seconds} seconds.
      </p>
    );
  }
});

ReactDOM.render(
  <TickTock />,
  document.getElementById('example')
);
```

コンポーネントが複数のミックスインを使っており、いくつかのミックスインが同じライフサイクルメソッドを定義している場合（例：コンポーネント破棄時に複数のミックスインがクリーンアップ処理をする）、全てのライフサイクルメソッドが呼ばれることが保証されています。ミックスインで定義されているメソッドはミックスインとして列挙された順に実行され、そのあとにコンポーネントのメソッドが呼ばれます。

このページを編集する