React 用語集

シングルページアプリケーション
シングルページアプリケーション (single-page application) は、単一の HTML ページでアプリケーションの実行に必要なすべてのアセット（JavaScript や CSS など）をロードするようなアプリケーションです。初回ページもしくはそれ以降のページでのユーザとのやりとりにおいて、サーバーとの往復が不要、すなわちページのリロードが発生しません。
React でシングルページアプリケーションを構築することができますが、そうすることは必須ではありません。React は、既存のウェブサイトの小さな一部分を拡張してよりインタラクティブにするために使用することもできます。React で記述されたコードは、PHP などによりサーバーでレンダリングされたマークアップや、他のクライアントサイドのライブラリと問題なく共存できます。実際に、Facebook では React はそのように利用されています。

ES6、ES2015、ES2016 など
これらの頭字語は全て、ECMAScript 言語の標準仕様の最近のバージョンのことを指しており、JavaScript 言語とははこれらの仕様に対する実装です。ES6 バージョン（ES2015 とも呼ばれます）はそれ以前のバージョンから、多くのものが追加されています：アロー関数、クラス、テンプレートリテラル、`let` および `const` ステートメントなどです。特定のバージョンの詳細についてはこちらで学ぶことができます。

コンパイラ
JavaScript コンパイラは JavaScript コードを受け取って変換し、別のフォーマットの JavaScript コードを返します。最も一般的なユースケースは ES6 構文を受け取り、古いブラウザーが解釈できる構文に変換することです。Babel は React とともに利用されることが最も多いコンパイラです。

バンドラ
バンドラは別々のモジュールとして記述された（しばしば数百個になる）JavaScript および CSS のコードを受け取り、ブラウザに最適化された数個のファイルに結合します。Webpack や Browserify を含む、いくつかのバンドラが React アプリケーションで一般的に利用されています。

パッケージマネージャ
パッケージ マネージャーは、プロジェクト内の依存関係を管理するためのツールです。npm および Yarn の 2 つのパッケージマネージャが React アプリケーションで一般的に利用されています。どちらも同じ npm パッケージレジストリのクライアントです。

CDN
CDN は Content Delivery Network の略です。CDN はキャッシュされた静的なコンテンツをネットワーク化されたサーバから世界中に配信します。

JSX
JSX は JavaScript の拡張構文です。テンプレート言語に似ていますが、完全に JavaScript だけで動作します。JSX は、“React 要素” と呼ばれるプレーンな JavaScript オブジェクトを返す、`React.createElement()` のコールにコンパイルされます。JSX の基本的な導入部分を学ぶにはこちらのドキュメントを参照してください。また、さらに JSX について詳細に学ぶにはこちらを参照してください。
React DOM は HTML の属性名ではなく、キャメルケースの命名規則を使用します。例えば、`tabindex` は、JSX では `tabIndex` となります。`class` も `className` と記述されますが、これは `class` が JavaScript において予約語であるためです：

```
const name = 'Clementine';
ReactDOM.render(
  <h1 className="hello">My name is {name}!</h1>,
  document.getElementById('root')
);
```

要素
React 要素は React アプリケーションを構成するブロックです。要素を、より広く知られている概念である “コンポーネント” と混同する人もいるかもしれません。要素はあなたが画面上に表示したいものの説明書きとなるものです。React 要素はイミュータブルです。

```
const element = <h1>Hello, world</h1>;
```

通常、要素は直接使用されるものではなく、コンポーネントから返されるものです。

コンポーネント

React のコンポーネントとは、ページに表示される React 要素を返す、小さく再利用可能なコードのことです。もっともシンプルな形の React コンポーネントは、React 要素を返すプレーンな JavaScript 関数です：

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

コンポーネントは ES6 クラスであることもあります：

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

コンポーネントは機能的に異なる部品に分割でき、他のコンポーネントの中で使用することができます。コンポーネントは別のコンポーネント、配列、文字列および数値を返すことができます。役に立つ経験則として、UI の一部（Button、Panel、Avatar など）が複数回使われている場合、または UI の一部が独立できるほど複雑（App、FeedStory、Comment など）な場合、それらは再利用可能なコンポーネントの有力な候補であるといえます。コンポーネント名は常に大文字で始める必要があります（`<wrapper/>` ではなく `<Wrapper/>` とすること）。コンポーネントのレンダーについての詳細はこのドキュメントを参照してください。

`props`

`props` は、React コンポーネントへの入力のことです。親コンポーネントから子コンポーネントへと順番に渡されるようなデータです。
`props` は読み取り専用です。どのような形であれ変更されるべきではありません：

```
// 間違った例
props.number = 42;
```

ある値を、ユーザーの入力やネットワークのレスポンスに応じて変更する必要がある場合は、代わりに state を使用してください。

`props.children`

`props.children` は全てのコンポーネントで使用可能です。これには、コンポーネントの開始タグと終了タグの間の全てのコンテンツが含まれています。例えば：

```
<Welcome>Hello world!</Welcome>
```

文字列 `Hello world!` は `Welcome` コンポーネントの `props.children` で利用できます：

```
function Welcome(props) {
  return <p>{props.children}</p>;
}
```

クラスとして定義されたコンポーネントでは、`this.props.children` を使用してください：

```
class Welcome extends React.Component {
  render() {
    return <p>{this.props.children}</p>;
  }
}
```

`state`

あるコンポーネントが時間とともに変化するデータと関連付けられている場合は、`state` が必要です。例えば、`Checkbox` というコンポーネントはその state に `isChecked` が必要となるかもしれません。また `NewsFeed` というコンポーネントは `fetchedPosts` を state に入れて管理したいかもしれません。
`state` と `props` の最も重要な違いは、`props` は親コンポーネントから渡されますが、`state` はコンポーネント自身によって管理されるということです。コンポーネントは自身の `props` を変更できませんが、`state` を変更することはできます。変更するには、`this.setState()` を呼び出す必要があります。クラスとして定義されたコンポーネントだけが state を持つことができます。
経時的に変化するそれぞれのデータについて、それを state として「所有する」コンポーネントは 1 つだけであるべきです。2 つの異なるコンポーネントの state を同期しようとしないでください。代わりに、それらの直近の共通祖先コンポーネントに state をリフトアップして、両方に props として渡してください。

ライフサイクルメソッド

ライフサイクルメソッドは、コンポーネントの様々なフェーズにおいて実行される特別な関数です。コンポーネントが作成されて DOM に挿入（マウント）された時、コンポーネントが更新された時、コンポーネントが DOM からアンマウント（つまり削除）された時のそれぞれで、利用可能なメソッドがあります。

制御されたコンポーネント vs. 非制御コンポーネント

React では、フォームの入力を扱うのに 2 つの異なるアプローチがあります。
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** (controlled component) と呼ばれます。ユーザーが制御されたコンポーネントにデータを入力すると、変更イベントハンドラがトリガーされ、コードが入力が有効であるか（更新された値で再レンダリングすることで）決定します。再レンダリングしなければフォーム要素は変更されないままとなります。
非制御コンポーネント (uncontrolled component) は React の管理外にあるフォーム要素と同様に動作します。ユーザーがフォームフィールド（入力ボックス、ドロップダウンなど）にデータを入力した場合、更新された情報が反映され、その際に React は何もする必要がありません。しかし、このことはフィールドに特定の値を設定できないということでもあります。
ほとんどの場合では、制御されたコンポーネントを使用するべきでしょう。

---

key

“key” は特別な文字列の属性で、要素の配列を作成する際に含めておく必要があります。key は React がどの要素が変更、追加もしくは削除されたかを識別するのに役立ちます。key は配列内の要素に安定した一意性を与えるよう設定されるべきです。
key は同じ配列内の兄弟要素間で一意としなければなりません。アプリケーション全体、単一のコンポーネントに渡ってすべて一意である必要はありません。
`Math.random()` のようなものを key として設定しないでください。key には再レンダリングをまたいだ「安定した一意性」を持たせることで、要素の追加、削除および並べ替えがあった時に React が識別できることが重要です。理想的には key は `post.id` のように、データから得られる一意で安定した識別子に対応するべきです。

---

ref

React は任意のコンポーネントに追加できる特別な属性をサポートしています。 `ref` 属性は、`React.createRef()` 関数、コールバック関数、あるいは（古い API では）文字列によって生成されるオブジェクトです。 `ref` 属性がコールバック関数の場合、その関数は引数として（要素の種類によって）DOM 要素またはクラスインスタンスを受け取ります。これによって、DOM 要素またはコンポーネントのインスタンスへと直接アクセスできます。
ref は消極的に利用してください。アプリケーション内で何かを実行するために ref を頻繁に使用している場合、トップダウンのデータフローに慣れ親しむことを検討してください。

イベント

React 要素でイベントを扱う場合には、構文上の違いがあります：

- React のイベントハンドラは小文字ではなく、キャメルケースで名前が付けられます。
- JSX ではイベントハンドラとして文字列ではなく関数を渡します。

リコンシリエーション（更新検出処理）

コンポーネントの props か state が変更された場合、React は新しく返された要素を以前にレンダーされた要素と比較することで、本物の DOM の更新が必要かを判断します。それらが等しくない場合、React は DOM を更新します。このプロセスを “リコンシリエーション (reconciliation)” と呼びます。

このページを編集する