JSX の導入

以下の変数宣言を考えてみましょう：

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

このおかしなタグ構文は文字列でも HTML でもありません。
これは JSX と呼ばれる JavaScript の構文の拡張です。UI がどのような見た目かを記述するために、React とともに JSX を使用することを私たちはお勧めしています。JSX はテンプレート言語を連想させるでしょうが、JavaScript の機能を全て備えたものです。
JSX は React “要素” を生成します。次の章で React 要素を DOM に変換する方法について見ていきます。以下では、JSX を使い始めるのに必要な基礎を学ぶことができます。

JSXを使う理由

表示のためのロジックは、イベントへの応答や経時的な状態の変化、画面表示のためのデータを準備する方法といった、他の UI ロジックと本質的に結合したものであり、React はその事実を受け入れます。
マークアップとロジックを別々のファイルに書いて人為的に**技術**を分離するのではなく、React はマークアップとロジックを両方含む疎結合の「コンポーネント」という単位を用いて**関心**を分離します。
後のセクションでコンポーネントについては改めて詳しく紹介しますが、現時点で JavaScript にマークアップを書くことが気にくわない場合、こちらの議論で考えが改まるかもしれません。
React で JSX を使うことは必須ではありませんが、ほとんどの人は JavaScript コード中で UI を扱う際に、JSX を見た目に有用なものだと感じています。また、JSX があるために React は有用なエラーや警告をより多く表示することができます。
前置きはこのくらいにして、早速始めましょう。

JSX に式を埋め込む

以下の例では、`name` という変数を宣言して、それを中括弧に囲んで JSX 内で使用しています。

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

あらゆる有効な JavaScript の式を JSX 内で中括弧に囲んで使用できます。例えば、`2 + 2` や `user.firstName` や `formatName(user)` はすべて有効な JavaScript の式です。
以下の例では、`formatName(user)` という JavaScript 関数の結果を `<h1>` 要素内に埋め込んでいます。

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Try it on CodePen

読みやすさのため JSX を複数行に分けています。必須ではありませんが、複数行に分割する場合には、自動セミコロン挿入の落とし穴にはまらないように括弧で囲むことをおすすめします

JSX もまた式である

コンパイルの後、JSX の式は普通の JavaScript の関数呼び出しに変換され、JavaScript オブジェクトへと評価されます。
これは JSX を `if` 文や `for` ループの中で使用したり、変数に代入したり、引数として受け取ったり、関数から返したりすることができるということです。

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
```

```
  return <h1>Hello, Stranger.</h1>;
}
```

JSX で属性を指定する
文字列リテラルを属性として指定するために引用符を使用できます。

```
const element = <div tabIndex="0"></div>;
```

属性に JavaScript 式を埋め込むために中括弧を使用することもできます。

```
const element = <img src={user.avatarUrl}></img>;
```

属性に JavaScript 式を埋め込む時に中括弧を囲む引用符を書かないでください。（文字列の場合は）引用符、もしくは（式の場合は）中括弧のどちらか一方を使用するようにし、同じ属性に対して両方を使用するべきではありません。

> **警告:**
> JSX は HTML よりも JavaScript に近いものですので、React DOM は HTML の属性ではなくキャメルケース (camelCase) のプロパティ命名規則を使用します。
> JSX では例えば、`class` は `className` となり、`tabindex` は `tabIndex` となります。

JSX で子要素を指定する
タグが空の場合、XML のように `/>` でタグを閉じる事ができます：

```
const element = <img src={user.avatarUrl} />;
```

JSX のタグは子要素を持つことができます：

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX はインジェクション攻撃を防ぐ
JSX にユーザーの入力を埋め込むことは安全です：

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

デフォルトでは、React DOM は JSX に埋め込まれた値をレンダリングされる前にエスケープします。このため、自分のアプリケーションで明示的に書かれたものではないあらゆるコードは、注入できないことが保証されます。レンダーの前に全てが文字列に変換されます。これは XSS (cross-site-scripting) 攻撃の防止に役立ちます。

JSX はオブジェクトの表現である
Babel は JSX を `React.createElement()` の呼び出しへとコンパイルします。
以下の 2 つの例は等価です：

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

`React.createElement()` はバグの混入を防止するためにいくつかのチェックも行いますが、本質的には以下のようなオブジェクトを生成します：

```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

このようなオブジェクトは “React 要素” と呼ばれます。これらは画面に表示したいものの説明書きとして考えることができます。React はこれらのオブジェクトを読み取り、DOM を構築して最新に保ちます。
次の章では React 要素を DOM に変換することについて見ていきましょう。

> **ヒント:**
> 利用したいエディタで “Babel” 言語定義を使用して、ES6 と JSX が適切にハイライトされるようにしておくことをお勧めします。このウェブサイトはそれと互換性のある Oceanic Next カラースキームを使用しています。

このページを編集する