COURSEWARE

JSX

Contents

Overview

Java Syntax Extension (JSX) is a powerful variant of JavaScript used with React.

Separation of technologies

Traditionally, web pages are built using a combination of three different languages:

- **HTML** - HyperText Markup Language - which organises the page's structure;
- **CSS** - Cascading Style Sheets - which dictate the design of the page;
- **JavaScript** - which designates the functionality of a page.

When used in conjunction, the three filetypes for a Web page would look something like `index.html`, `index.js` and `index.css`.

By separating each technology into a separate file, web developers would be able to clearly work on each aspect of a page individually.

Separation of Concern

React uses a slightly different approach to this setup.

Rather than separate web pages into their constituent technologies, React separates pages into *concerns*, which can be encapsulated into individual **components**.

One of the ways in which React does this is by combining HTML and JavaScript together, by using **JSX**.

JSX Tags

JSX is clever because it allows **HTML** inside of it by using a near-identical `<tag>` system:

```
const element = <h1>This is an HTML element stored inside a JS constant</h1>
```

This works for child elements as well:

```
const element = (
    <div>
        <h1>Hello there.</h1>
        <h2>General Kenobi.<h2>
    </div>
)
```

Embedding Expressions

JSX allows you to embed JavaScript expressions, which can then be used to display JavaScript elements inside HTML.

For instance, the example below declares a variable and pings it across to the HTML inside JSX, which we then pass to React's `render()` function:

```
const name = "Nick Johnson"
const element = <h1>Hello, {name}</h1>

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

The {curly braces} can contain any valid JavaScript expression, allowing us to embed more complex things into the HTML elements we define.

Here, we'll embed the result from calling a function in JavaScript into an `<h1>` element:

Embedding Function Results

```
const formatName = (someName) => {
    return someName.forename + " " + someName.surname;
}

const somename = {
    forename: "Nick",
    surname: "Johnson"
};

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

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

Embedding Inside Conditionals/Iterators

On compilation, JSX expressions directly turn into JavaScript function calls.

As such, JSX directly evaluates into JavaScript objects.

This means that you can use JSX pretty much anywhere inside a React application, including for `for`-loops, `if`-statements, and as a function return:

```
const getGreeting = (someName) => {
    if (someName){
        return <h1>Hello, {formatName(someName)}!</h1>;
    } else {
        return <h1>Hello, whoever you are.</h1>;
    }
}
```

Embedding Inside Attributes

WIth JSX, it's possible to embed expressions into attributes, including string literals:

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

```
const element = <img src={someUser.avatarURL}/>
```

Note that if you're embedding an expression using {curly braces}, you can't use "quote marks" in conjunction with them:

```
const element = <img src="{doNotDoThis}"/>
```

The value of JSX

As well as being a hybrid between JavaScript and HTML, other features have been baked into JSX.

Security

One of the cooler things about JSX is that you can embed user input into it securely.

Let's say some kind of malicious script has been injected into a `response`:

```
const title = response.someInjectedScript;
const element = <h1>{title}</h1>;
```

By default, the React DOM *escapes* any values embedded in JSX before rendering them by converting them to strings.

This means that it is impossible to refer to anything that hasn't already been explicitly declared within your application.

As a result, any malicious code is neutralised *before* the page is rendered.

Object Representation

In JSX, everything is an object. When used in React, its helper agent *Babel* compiles JSX to `React.createElement()` calls.

As such, the following two code snippets are identical:

```
const element = (
  <h1 className="greeting">
    Hello there.
  </h1>
);
```

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

Tutorial

There is no tutorial for this module.

Exercises

Inside the React application you made in the Introduction module, implement some JSX which renders your name and address on the screen:

▶ Click for solution