React

React Overview

According to the React documentation, React is a “Javascript library for building user interfaces”[[1]](#footnote-1) As such, React’s approach is uniquely intended only to serve the needs of a view – only with those things with which a user interacts.

Unlike other solutions’ origins, Facebook’s stack has always included PHP. PHP has arguably excelled at applications which require heavy HTML templating because if it’s integrated approach to embedding logic within templates. As such, Facebook’s React also does not seek to maintain a strict separation of HTML templates and imperative logic.

React employs a special version of Javascript called JSX. JSX is used to provide syntactic sugar when representing/manipulating React’s internal version of the DOM – often referred to as Virtual DOM. Virtual DOM is a tree data structure, defined and available in Javascript, meant to mirror many aspects of the real DOM created every time a browser renders a document. React uses the virtual DOM to only make the minimal amount of changes to the real DOM, automatically making the changes in the least computationally expensive way for the browser.

JSX is a declarative, very HTML-like syntax, which must be compiled into native React API calls in order to modify the virtual DOM. React elements are Javascript objects which represent HTML elements. React can be used without using JSX, by directly calling the API methods which build the virtual DOM manually, but that often means losing one of the primary benefits of React, it’s declarative syntactic sugar.

React has a small API, making it generally easy to learn, and simple to use. For example, <h1>Hello</h1>, is a React JSX element. Looks like regular HTML, right? The same React element can be written without JSX, using vanilla Javascript with React.DOM.h1(null, ‘Hello’); JSX is so much like HTML, it takes less effort to read and write, and is compiled from the first, to the second example before running in the browser.

React’s unique approach revolves around creating reusable components that contain logic and templates, and are designed to be reusable and composable declaratively. Components represent whole pieces of your application, such as a NavBar, LoginForm, DataTable, or ProfileEditor.

Rendering

First, it’s important to understand that React allows you to build elements in memory. Similar to how this line of jQuery translates into a wrapped element in memory, React allows for this abstraction before you use React’s render method.

JQuery snippet:

var el = $(“<div class=”virtualElement”>I’m not real, only virtual!</div>”);

$(“body”).append(el);

Just as jQuery allows for creating the element, and then must append it to the DOM to make it real, virtual elements only exist in JS memory, so the render method is required to actually make a difference in the DOM:

React snippet:

React.render(<div class=”virtualElement”>I’m becoming Real!</div>, document.body)

Render takes to argument, first the virtual element or reference to a virtual element, then second a real DOM node in the DOM to modify. React uses it’s virtual DOM to minimize actual DOM changes, so the render method uses this abstraction to maximize efficient DOM manipulation.

Components

Components make up the building blocks on React. They are the abstraction that makes React unique in it’s combined JS/HTML/(CSS) style. Many oppose this mixing of logic, but coming from a PHP background, React’s goal is to keep components highly modularized, so that they embody encapsulation as a component.

React example:

var ReactButton = React.createClass({

render: function() {

return <button>I’m a button</button>

}

});

React.render(<ReactButton />, document.body);

The createClass method takes an object which implements the render method. The

ReactButton component is constructed, <ReactButton>, is then rendered to the body in the last line.

This just provides the structure for component encapsulation. You can add styles, and additional functionality to extend this component.

Props

Props are very similar to options or arguments to a component. They take advantage of the attribute concept in HTML to configure components, mixing in JS references.

var Image = React.createClass({

render: function() {

return (

<div className=’image’>

<img src={this.props.URL} />

<span>{this.props.imageText}</span>

</div>

);

}

});

React.render(<Image URL=’http://reddit.com/cats.jpg’ imageText=’This is a cat.’ />, document.body);

The react render function is passing two props to the Image component; URL and imageText. Inside the component’s render function, the URL prop uses a sort of templating {} to include the URL as plugged in from the bottom render function. The imageText prop is also used as plain text within the component’s span tag.

A component never changes its own props, those cannot be changed, they’re immutable. If you need mutable data that changes, you use state.

State

The state object holds data that will change internal to the component. Similar to how JS uses inner and outer function scopes. Outer scopes can keep information in memory to be used inside a nested inner function scope.

var Image = React.createClass({

toggleLiked: function() {

this.setState({

liked: !this.state.liked

});

},

getInitialState: function() {

return {

liked: false

};

}

render: function() {

var likeButtonClass = this.state.liked ? ‘active’ : ‘’;

return (

<div className=’image’>

<img src={this.props.URL} />

<div className=’bar’>

<button onClick={this.toggleLiked} className={buttonClass}> Like This Image </button>

<span>{this.props.imageText}</span>

</div>

</div>

);

}

});

React.render(<Image URL=’http://reddit.com/cats.jpg’ imageText=’This is a cat.’ />, document.body);

Adding state obviously makes things a bit more complex. The component has a new getInitialState function which is called by default when the component is initialized. The returned object sets the component’s initial state object.

The other function the component now has is called toggleLiked. This function uses setState on the component to toggle the liked value.

In the component’s render function, buttonClass is assigned active or nothing, depending on whether the liked state has been changed.

buttonClass is used as a class name in the React button, and the button has an onClick handler to set the toggleLinked function.

So this is what happens when the component is rendered in the DOM:

1. When the component’s button is clicked, toggleLiked is called
2. The liked state is changed
3. React re-renders the component in the virtual DOM
4. The new virtual DOM is compared with the previous virtual DOM
5. React diffs the changes, and updates only the items in the DOM which need updates.

So react changes the class name on the button as a result.

Composition

What’s the point of all this componentization? Well, components can be declaratively combined using composition. Components can be combined with other components into larger components. Each time <Image /> that is used in a render function creates a new instance of the component.

React Docs

React only seeks to do one thing well: the View. It’s up to you to add all the logic which will move data around your application. React is only responsible for rendering already available data which plugged into props on your components. This lends itself to very fast performance when paired with virtual DOM diffing:

<https://www.codementor.io/reactjs/tutorial/reactjs-vs-angular-js-performance-comparison-knockout>

This is a very brief overview. To find out more about React, check out the React documentation here: <http://facebook.github.io/react/docs/getting-started.html>

There’s also a great, quick visual overview at <http://egghead.io/lessons/react-react-in-7-minutes/>

1. http://facebook.github.io/react/ [↑](#footnote-ref-1)