QUICK START

GUIDES

Displaying Data

The most basic thing you can do with a UI is display some data. React makes it easy to display data and automatically keeps the interface up-to-date when the data changes.

Getting Started

Let's look at a really simple example. Create a `hello-react.html` file with the following code:

Code

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello React</title>
    <script src="http://fb.me/react-0.4.1.min.js"></script>
    <script src="http://fb.me/JSXTransformer-0.4.1.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script type="text/jsx">

      // ** Your code goes here! **

    </script>
  </body>
</html>
```

For the rest of the documentation, we'll just focus on the JavaScript code and assume it's inserted into a template like the one above. Replace the placeholder comment above with the

```
/** @jsx React.DOM */

var HelloWorld = React.createClass({
  render: function() {
    return (
      <p>
        Hello, <input type="text" placeholder="Your name here" />!
        It is {this.props.date.toTimeString()}
      </p>
    );
  }
});

setInterval(function() {
  React.renderComponent(
    <HelloWorld date={new Date()} />,
    document.getElementById('example')
  );
}, 500);
```

Reactive Updates

Open `hello-react.html` in a web browser and type your name into the text field. Notice that React is only changing the time string in the UI — any input you put in the text field remains, even though you haven't written any code to manage this behavior. React figures it out for you and does the right thing.

The way we are able to figure this out is that React does not manipulate the DOM unless it needs to. It uses a fast, internal mock DOM to perform diffs and computes the most efficient DOM mutation for you.

The inputs to this component are called `props` — short for "properties". They're passed as

Components are Just Like Functions

React components are very simple. You can think of them as simple function that take in `props` and `state` (discussed later) and render HTML. Because they're so simple, it makes them very easy to reason about.

> **Note:**
>
> One limitation: React components can only render a single root node. If you want to return multiple nodes they *must* be wrapped in a single root.

JSX Syntax

We strongly believe that components are the right way to separate concerns rather than "templates" and "display logic." We think that markup and the code that generates it are intimately tied together. Additionally, display logic is often very complex and using template languages to express it becomes cumbersome.

We've found that the best solution for this problem is to generate markup directly from the JavaScript code such that you can use all of the expressive power of a real programming language to build UIs. In order to make this easier, we've added a very simple, optional HTML-like syntax for the function calls that generate markup called JSX.

JSX lets you write JavaScript function calls with HTML syntax. To generate a link in React using pure JavaScript you'd write: `React.DOM.a({href: 'http://facebook.github.io/react/'}, 'Hello React!')`. With JSX this becomes `<a href="http://facebook.github.io/react/">Hello React!</a>`. We've found this has made building React apps easier and designers tend to prefer the syntax, but everyone has their own workflow, so JSX is not required to use React.

JSX is similar to HTML, but not exactly the same. See JSX gotchas for some key differences.

The easiest way to get started with JSX is to use the in-browser `JSXTransformer`. We strongly recommend that you don't use this in production. You can precompile your code using our command-line react-tools package.

← Prev

Next →

Add a comment...

□ Post to Facebook

Posting as Stoyan Stefanov (Change) Comment

Facebook social plugin

A Facebook & Instagram collaboration.