Working With the Browser

React provides powerful abstractions that free you from touching the DOM directly in most cases, but sometimes you simply need to access the underlying API, perhaps to work with a third-party library or existing code.

The Mock DOM

React is so fast because it never talks to the DOM directly. React maintains a fast in-memory representation of the DOM. `render()` methods return a *description* of the DOM, and React can diff this description with the in-memory representation to compute the fastest way to update the browser.

Additionally, React implements a full synthetic event system such that all event objects are guaranteed to conform to the W3C spec despite browser quirks, and everything bubbles consistently and in a performant way cross-browser. You can even use some HTML5 events in IE8!

Most of the time you should stay within React's "faked browser" world since it's more performant and easier to reason about. However, sometimes you simply need to access the underlying API, perhaps to work with a third-party library like a jQuery plugin. React provides escape hatches for you to use the underlying DOM API directly.

Refs and getDOMNode()

To interact with the browser, you'll need a reference to a DOM node. Every mounted React component has a `getDOMNode()` function which you can call to get a reference to it.

> **Note:**

> mounted yet (like calling `getDOMNode()` in `render()` on a component that has yet to be created) an exception will be thrown.

In order to get a reference to a React component, you can either use `this` to get the current React component, or you can use refs to refer to a component you own. They work like this:

Code

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

var MyComponent = React.createClass({
  handleClick: function() {
    // Explicitly focus the text input using the raw DOM API.
    this.refs.myTextInput.getDOMNode().focus();
  },
  render: function() {
    // The ref attribute adds a reference to the component to
    // this.refs when the component is mounted.
    return (
      <div>
        <input type="text" ref="myTextInput" />
        <input
          type="button"
          value="Focus the text input"
          onClick={this.handleClick}
        />
      </div>
    );
  }
});

React.renderComponent(
  <MyComponent />,
```

More About Refs

To learn more about refs, including ways to use them effectively, see our more about refs documentation.

Component Lifecycle

Components have three main parts of their lifecycle:

- **Mounting:** A component is being inserted into the DOM.
- **Updating:** A component is being re-rendered to determine if the DOM should be updated.
- **Unmounting:** A component is being removed from the DOM.

React provides lifecycle methods that you can specify to hook into this process. We provide **will** methods, which are called right before something happens, and **did** methods which are called right after something happens.

Mounting

- `getInitialState(): object` is invoked before a component is mounted. Stateful components should implement this and return the initial state data.
- `componentWillMount()` is invoked immediately before mounting occurs.
- `componentDidMount(DOMElement rootNode)` is invoked immediately after mounting occurs. Initialization that requires DOM nodes should go here.

Updating

- `componentWillReceiveProps(object nextProps)` is invoked when a mounted component receives new props. This method should be used to compare `this.props`

when a component decides whether any changes warrant an update to the DOM. Implement this as an optimization to compare `this.props` with `nextProps` and `this.state` with `nextState` and return false if React should skip updating.

- `componentWillUpdate(object nextProps, object nextState)` is invoked immediately before updating occurs. You cannot call `this.setState()` here.
- `componentDidUpdate(object prevProps, object prevState, DOMElement rootNode)` is invoked immediately after updating occurs.

Unmounting

- `componentWillUnmount()` is invoked immediately before a component is unmounted and destroyed. Cleanup should go here.

Mounted Methods

Mounted composite components also support the following methods:

- `getDOMNode(): DOMElement` can be invoked on any mounted component in order to obtain a reference to its rendered DOM node.
- `forceUpdate()` can be invoked on any mounted component when you know that some deeper aspect of the component's state has changed without using `this.setState()`.

> **Note:**
>
> The `DOMElement rootNode` argument of `componentDidMount()` and `componentDidUpdate()` is a convenience. The same node can be obtained by calling `this.getDOMNode()`.

Browser Support and Polyfills

At Facebook, we support older browsers, including IE8. We've had polyfills in place for a long

example, instead of seeing `+new Date()`, we can just write `Date.now()`. Since the open source React is the same as what we use internally, we've carried over this philosophy of using forward thinking JS.

In addition to that philosophy, we've also taken the stance that we, as authors of a JS library, should not be shipping polyfills as a part of our library. If every library did this, there's a good chance you'd be sending down the same polyfill multiple times, which could be a sizable chunk of dead code. If your product needs to support older browsers, chances are you're already using something like es5-shim.

Polyfills Needed to Support Older Browsers

- `Array.isArray`
- `Array.prototype.forEach`
- `Array.prototype.indexOf`
- `Function.prototype.bind`
- `Date.now`
- `Array.prototype.some` (also in `es5-shim.js`)

All of these can be polyfilled using `es5-shim.js` from https://github.com/kriskowal/es5-shim.

- `console.*` - Only needed when not using the minified build. If you need to polyfill this, try https://github.com/paulmillr/console-polyfill.
- `Object.create` - Provided in `es5-sham.js` @ https://github.com/kriskowal/es5-shim.

← Prev Next →

Post to Facebook

Posting as Stoyan Stefanov (Change) Comment

Facebook social plugin