Reference

Examples

Production Apps

- All of Instagram.com is built on React.
- Many components on Facebook.com, including the commenting interface, ads creation flows, and page insights.
- Khan Academy is using React for its question editor.

Sample Code

- We've included a step-by-step comment box tutorial.
- The React starter kit includes several examples which you can view online in our GitHub repository.
- reactapp is a simple app template to get you up-and-running quickly with React.
- React one-hour email goes step-by-step from a static HTML mock to an interactive email reader (written in just one hour!)
- Rendr + React app template demonstrates how to use React's server rendering capabilities.

API

React

`React` is the entry point to the React framework. If you're using one of the prebuilt packages

`React.DOM` provides all of the standard HTML tags needed to build a React app. You generally don't use it directly; instead, just include it as part of the `/** @jsx React.DOM */` docblock.

React.initializeTouchEvents

Code

```
initializeTouchEvents(boolean shouldUseTouch)
```

Configure React's event system to handle touch events on mobile devices.

React.createClass

Code

```
function createClass(object specification)
```

Creates a component given a specification. A component implements a `render` method which returns one single child. That child may have an arbitrarily deep child structure. One thing that makes components different than standard prototypal classes is that you don't need to call new on them. They are convenience wrappers that construct backing instances (via new) for you.

React.renderComponent

Code

```
ReactComponent renderComponent(ReactComponent container, DOMElement container)
```

Renders a React component into the DOM in the supplied `container`.

If the React component was previously rendered into `container`, this will perform an update on it and only mutate the DOM as necessary to reflect the latest React component.

React.unmountAndReleaseReactRootNode

Remove a mounted React component from the DOM and clean up its event handlers and state.

React.renderComponentToString

Code

```
renderComponentToString(ReactComponent component, function callback)
```

Render a component to its initial HTML. This should only be used on the server. React will call `callback` with an HTML string when the markup is ready. You can use this method to create static site generators, or you can generate HTML on the server and send it down to have a very fast initial page load. If you call `React.renderComponent()` on a node that already has this server-rendered markup, React will preserve it and only attach event handlers, allowing you to have a very performant first-load experience.

AbstractEvent

Your event handlers will be passed instances of `AbstractEvent`, a cross-browser wrapper around the browser's native event. It has the same interface as the browser's native event (such as `stopPropagation()` and `preventDefault()`) except they work exactly the same across all browsers.

If you find that you need the underlying browser event for some reason, simply use the `nativeEvent` attribute to get it.

ReactComponent

Component classses created by `createClass()` return instances of `ReactComponent` when called. Most of the time when you're using React you're either creating or consuming `ReactComponent`s.

getDOMNode

If this component has been mounted into the DOM, this returns the corresponding native browser DOM element. This method is useful for reading values out of the DOM, such as form field values and performing DOM measurements.

setProps

Code

```
setProps(object nextProps)
```

When you're integrating with an external JavaScript application you may want to signal a change to a React component rendered with `renderComponent()`. Simply call `setProps()` to change its properties and trigger a re-render.

> **Note:**
>
> This method can only be called on a root-level component. That is, it's only available on the component passed directly to `renderComponent()` and none of its children. If you're inclined to use `setProps()` on a child component, instead take advantage of reactive updates and pass the new prop to the child component when it's created in `render()`.

replaceProps

Code

```
replaceProps(object nextProps)
```

Like `setProps()` but deletes any pre-existing props that are not in nextProps.

transferPropsTo

Transfer properties from this component to a target component that have not already been set on the target component. This is usually used to pass down properties to the returned root component. `targetComponent`, now updated with some new props is returned as a convenience.

setState

Code

```
setState(object nextState[, function callback])
```

Merges nextState with the current state. This is the primary method you use to trigger UI updates from event handlers and server request callbacks. In addition, you can supply an optional callback function that is executed once `setState` is completed.

> **Note:**
>
> *NEVER* mutate `this.state` directly. As calling `setState()` afterwards may replace the mutation you made. Treat `this.state` as if it were immutable.
>
> Note:
>
> `setState()` does not immediately mutate `this.state` but creates a pending state transition. Accessing `this.state` after calling this method can potentially return the existing value.
>
> Note:
>
> There is no guarantee of synchronous operation of calls to `setState` and calls may be batched for performance gains.

replaceState

Like `setState()` but deletes any pre-existing state keys that are not in nextState.

forceUpdate()

Code

```
forceUpdate([function callback])
```

If your `render()` method reads from something other than `this.props` or `this.state` you'll need to tell React when it needs to re-run `render()`. Use `forceUpdate()` to cause React to automatically re-render. This will cause `render()` to be called on the component and all of its children but React will only update the DOM if the markup changes.

Normally you should try to avoid all uses of `forceUpdate()` and only read from `this.props` and `this.state` in `render()`. This makes your application much simpler and more efficient.

> **Note:**
>
> There is no guarantee of synchronous operation of calls to `forceUpdate` and calls may be batched for performance gains.

Lifecycle methods

Code

```
object getInitialState()
componentWillMount()
componentDidMount(DOMElement domNode)
componentWillReceiveProps(object nextProps)
boolean shouldComponentUpdate(object nextProps, object nextState)
componentWillUpdate(object nextProps, object nextState)
ReactComponent render()
componentDidUpdate(object prevProps, object prevState, DOMElement domNode)
```

See the working with the browser documentation for more details on these lifecycle methods.

DOM Differences

React has implemented a browser-independent events and DOM system for performance and cross-browser compatibility reasons. We took the opportunity to clean up a few rough edges in browser DOM implementations.

- All events (including submit) bubble correctly per the W3C spec
- All event objects conform to the W3C spec
- All DOM properties and attributes (including event handlers) should be camelCased to be consistent with standard JavaScript style. We intentionally break with the spec here, since the spec is inconsistent.
- The `style` attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM `style` JavaScript property, is more efficient, and prevents XSS security holes.
- `onChange` behaves as you would expect it to: whenever a form field is changed this event is fired rather than inconsistently on blur. We intentionally break from existing browser behavior because `onChange` is a misnomer for its behavior and React relies on this event to react to user input in real time.

← Prev

Add a comment...

Post to Facebook Posting as Stoyan Stefanov (Change) Comment

Facebook social plugin