The component lifecycle

All living things are born. And then they die.

React components have similar lifecycles. They are born (mounted), updated, and eventually die (unmounted). By building our understanding of React, we can help all components have good lifecycles.

We'll look at the different steps to a component's lifecycle while going over the component API. Most of this logic lives in **ReactCompositeComponent** if you'd like to reference the source.

Part 1: The creation

constructor(props)

A component begins its life when it is instantiated. The constructor is called with the initial set of props. In this function, we perform any initialization logic and set the component's initial state.

For components that don't have state, the default constructor is often sufficient (and so you can skip writing one).

render()

React will render the component for the first time. React will process the return value and recursively instantiate and mount child components.

At the end of the render loop, React will update the DOM to match the virtual DOM. For newly instantiated components, this is when the DOM nodes are created.

componentDidMount()

When the instance is rendered into DOM for the first time, React will call `componentDidMount()` if it is defined. This is the first time you have access to the DOM and child refs.

Now the component instance is rendered and alive. It will live happily and update until it is unmounted.

Part 2: Updates

Preparing to update

There are four reasons why React will update a component.

- A parent (technically "owner") component is re-rendered.
- The component calls `this.setState`.
- The component calls `this.forceUpdate()`.
- The component is rendered again by `ReactDOM.render`.

When a parent component is re-rendered, React will either update or unmount the child instance. We'll take a deeper look at this process when we dive into the reconciliation algorithm.

componentWillReceiveProps(nextProps)

This is called when a parent component is re-rendered (or ReactDOM.render is called). This is where you update state that is derived from changes in props.

shouldComponentUpdate(nextProps, nextState)

Before continuing with the update, React will call this optional method to check whether it should continue with the update. By returning false here, the update loop for this instance ends early and all the remaining steps are skipped. This is a key hook to making your apps more performant; learn more in the article **Optimizing with shouldComponentUpdate**.

The update

Once React has the green light to do the update, React will perform the following steps.

componentWillUpdate(nextProps, nextState)

This method is called right before the update call to render.

render()

There is nothing different about this `render` call compared to the call for the initial `render`. This point is very important to React. Your `render` function should not differentiate whether this is the initial render or an update.

React takes `render`'s return value and compares it to the return value last time `render` was called and decides what updates to make, if any. This is the heart of the reconciliation process that we'll dive into next chapter. After reconciliation, React will recursively mount, update, and unmount child components as needed. Once the process resolves to base virtual DOM components, React updates the actual DOM.

componentDidUpdate(prevProps, prevState)

This method is called after React finishes the update and syncs the DOM.

Part 3: The unmount

Components are unmounted when the parent component is no longer rendered or the parent component performs an update that does not render this instance. `ReactDOM.unmountComponentAtNode` will also trigger an unmount.

componentWillUnmount()

This method is called right before React unmounts the component and does its cleanup. As with other operations, React recursively unmounts the children. Finally, React removes the nodes from the DOM and the component lifecycle is complete.

> **Why is there no componentDidUnmount?**
>
> Once the component is unmounted, the instance no longer exists. If the instance continued to execute instructions, it'd be equivalent of someone operating from the grave. Any logic you would want in `componentDidUnmount` should go in `componentWillUnmount` or managed by the owner component.

Summary

- Components are mounted, updated, and then unmounted.
- The component API allows you to tap into all stages of the lifecycle.
- The same `render` method is called on the initial mount and subsequent updates. Render should be a pure function that maps input (`this.props` and `this.state`) to output that is unaware whether the component has been rendered before.

For more details on the API, check out the **official docs**.

__ JUNE 28, 2016

READ ANOTHER

Why iOS developers should try React Native

Now is a great time to give React Native 5 minutes (or more honestly, several hours). Let's dig in why.

Browse articles

LIKED THIS?

Subscribe

Subscribe for a range of articles from React basics to advanced topics such as performance optimization and deep dives in the React source code.

Follow @buildwithreact