ReactDOM

If you load React from a `<script>` tag, these top-level APIs are available on the `ReactDOM` global. If you use ES6 with npm, you can write `import ReactDOM from 'react-dom'`. If you use ES5 with npm, you can write `var ReactDOM = require('react-dom')`.

Overview

The `react-dom` package provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside of the React model if you need to. Most of your components should not need to use this module.

- `render()`
- `hydrate()`
- `unmountComponentAtNode()`
- `findDOMNode()`
- `createPortal()`

Browser Support

React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10.

Note
We don't support older browsers that don't support ES5 methods, but you may find that your apps do work in older browsers if polyfills such as es5-shim and es5-sham are included in the page. You're on your own if you choose to take this path.

Reference

`render()`

```
ReactDOM.render(element, container[, callback])
```

Render a React element into the DOM in the supplied `container` and return a reference to the component (or returns `null` for stateless components).

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

If the optional callback is provided, it will be executed after the component is rendered or updated.

> **Note:**
> `ReactDOM.render()` controls the contents of the container node you pass in. Any existing DOM elements inside are replaced when first called. Later calls use React's DOM diffing algorithm for efficient updates.
>
> `ReactDOM.render()` does not modify the container node (only modifies the children of the container). It may be possible to insert a component to an existing DOM node without overwriting the existing children.
>
> `ReactDOM.render()` currently returns a reference to the root `ReactComponent` instance. However, using this return value is legacy and should be avoided because future versions of React may render components asynchronously in some cases. If you need a reference to the root `ReactComponent` instance, the preferred solution is to attach a callback ref to the root element.
>
> Using `ReactDOM.render()` to hydrate a server-rendered container is deprecated and will be removed in React 17. Use `hydrate()` instead.

---

hydrate()

```
ReactDOM.hydrate(element, container[, callback])
```

Same as `render()`, but is used to hydrate a container whose HTML contents were rendered by `ReactDOMServer`. React will attempt to attach event listeners to the existing markup.

React expects that the rendered content is identical between the server and the client. It can patch up differences in text content, but you should treat mismatches as bugs and fix them. In development mode, React warns about mismatches during hydration. There are no guarantees that attribute differences will be patched up in case of mismatches. This is important for performance reasons because in most apps, mismatches are rare, and so validating all markup would be prohibitively expensive.

If a single element's attribute or text content is unavoidably different between the server and the client (for example, a timestamp), you may silence the warning by adding `suppressHydrationWarning={true}` to the element. It only works one level deep, and is intended to be an escape hatch. Don't overuse it. Unless it's text content, React still won't attempt to patch it up, so it may remain inconsistent until future updates.

If you intentionally need to render something different on the server and the client, you can do a two-pass rendering. Components that render something different on the client can read a state variable like `this.state.isClient`, which you can set to `true` in `componentDidMount()`. This way the initial render pass will render the same content as the server, avoiding mismatches, but an additional pass will happen synchronously right after hydration. Note that this approach will make your components slower because they have to render twice, so use it with caution.

Remember to be mindful of user experience on slow connections. The JavaScript code may load significantly later than the initial HTML render, so if you render something different in the client-only pass, the transition can be jarring. However, if executed well, it may be beneficial to render a "shell" of the application on the server, and only show some of the extra widgets on the client. To learn how to do this without getting the markup mismatch issues, refer to the explanation in the previous paragraph.

`unmountComponentAtNode()`

```
ReactDOM.unmountComponentAtNode(container)
```

Remove a mounted React component from the DOM and clean up its event handlers and state. If no component was mounted in the container, calling this function does nothing. Returns `true` if a component was unmounted and `false` if there was no component to unmount.

`findDOMNode()`

> **Note:**
> `findDOMNode` is an escape hatch used to access the underlying DOM node. In most cases, use of this escape hatch is discouraged because it pierces the component abstraction. It has been deprecated in `StrictMode`.

```
ReactDOM.findDOMNode(component)
```

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. **In most cases, you can attach a ref to the DOM node and avoid using `findDOMNode` at all.**

When a component renders to `null` or `false`, `findDOMNode` returns `null`. When a component renders to a string, `findDOMNode` returns a text DOM node containing that value. As of React 16, a component may return a fragment with multiple children, in which case `findDOMNode` will return the DOM node corresponding to the first non-empty child.

> **Note:**
> `findDOMNode` only works on mounted components (that is, components that have been placed in the DOM). If you try to call this on a component that has not been mounted yet (like calling `findDOMNode()` in `render()` on a component that has yet to be created) an exception will be thrown.
>
> `findDOMNode` cannot be used on function components.

`createPortal()`

```
ReactDOM.createPortal(child, container)
```

Creates a portal. Portals provide a way to render children into a DOM node that exists outside the hierarchy of the DOM component.

Edit this page