Portals

Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

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

The first argument (`child`) is any renderable React child, such as an element, string, or fragment. The second argument (`container`) is a DOM element.

Usage

Normally, when you return an element from a component's render method, it's mounted into the DOM as a child of the nearest parent node:

```
render() {
  // React mounts a new div and renders the children into it
  return (
    <div>
      {this.props.children}
    </div>
  );
}
```

However, sometimes it's useful to insert a child into a different location in the DOM:

```
render() {
  // React does *not* create a new div. It renders the children into `domNode`.
  // `domNode` is any valid DOM node, regardless of its location in the DOM.
  return ReactDOM.createPortal(
    this.props.children,
    domNode
  );
}
```

A typical use case for portals is when a parent component has an `overflow: hidden` or `z-index` style, but you need the child to visually "break out" of its container. For example, dialogs, hovercards, and tooltips.

> **Note:**
> When working with portals, remember that managing keyboard focus becomes very important.
>
> For modal dialogs, ensure that everyone can interact with them by following the WAI-ARIA Modal Authoring Practices.

Try it on CodePen

Event Bubbling Through Portals

Even though a portal can be anywhere in the DOM tree, it behaves like a normal React child in every other way. Features like context work exactly the same regardless of whether the child is a portal, as the portal still exists in the *React tree* regardless of position in the *DOM tree*.

This includes event bubbling. An event fired from inside a portal will propagate to ancestors in the containing *React tree*, even if those elements are not ancestors in the *DOM tree*. Assuming the following HTML structure:

```
<html>
  <body>
    <div id="app-root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

A `Parent` component in `#app-root` would be able to catch an uncaught, bubbling event from the sibling node `#modal-root`.

```
// These two containers are siblings in the DOM
const appRoot = document.getElementById('app-root');
```

```
const modalRoot = document.getElementById('modal-root');


class Modal extends React.Component {
  constructor(props) {
    super(props);
    this.el = document.createElement('div');
  }

  componentDidMount() {
    // The portal element is inserted in the DOM tree after
    // the Modal's children are mounted, meaning that children
    // will be mounted on a detached DOM node. If a child
    // component requires to be attached to the DOM tree
    // immediately when mounted, for example to measure a
    // DOM node, or uses 'autoFocus' in a descendant, add
    // state to Modal and only render the children when Modal
    // is inserted in the DOM tree.
    modalRoot.appendChild(this.el);
  }

  componentWillUnmount() {
    modalRoot.removeChild(this.el);
  }

  render() {
    return ReactDOM.createPortal(
      this.props.children,
      this.el
    );
  }
}

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {clicks: 0};
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    // This will fire when the button in Child is clicked,
    // updating Parent's state, even though button
    // is not direct descendant in the DOM.
    this.setState(state => ({
      clicks: state.clicks + 1
    }));
  }

  render() {
    return (
```

```
        <div onClick={this.handleClick}>

          <p>Number of clicks: {this.state.clicks}</p>
          <p>
            Open up the browser DevTools
            to observe that the button
            is not a child of the div
            with the onClick handler.
          </p>
          <Modal>
            <Child />
          </Modal>
        </div>
      );
    }
  }

  function Child() {
    // The click event on this button will bubble up to parent,
    // because there is no 'onClick' attribute defined
    return (
      <div className="modal">
        <button>Click</button>
      </div>
    );
  }

  ReactDOM.render(<Parent />, appRoot);
```

Try it on CodePen

Catching an event bubbling up from a portal in a parent component allows the development of more flexible abstractions that are not inherently reliant on portals. For example, if you render a `<Modal />` component, the parent can capture its events regardless of whether it's implemented using portals.

Is this page useful?  Edit this page