createPortal

`createPortal`을 사용하면 일부 자식을 DOM의 다른 부분으로 렌더링할 수 있습니다.

```
<div>
  <SomeComponent />
  {createPortal(children, domNode, key?)}
</div>
```

레퍼런스

`createPortal(children, domNode, key?)`

Portal을 생성하려면 `createPortal`을 호출하여 일부 JSX와 렌더링할 DOM 노드를 전달합니다.

```
import { createPortal } from 'react-dom';

// ...
```

```
<div>
  <p>This child is placed in the parent div.</p>
  {createPortal(
    <p>This child is placed in the document body.</p>,
    document.body
  )}
</div>
```

아래 예시를 참고하세요.

Portal은 DOM 노드의 물리적 배치만 변경합니다. 이외의 모든 측면에서, Portal로 렌더링된 JSX는 이를 렌더링하는 React 컴포넌트의 자식 노드처럼 동작합니다. 예를 들어, 자식은 부모 트리가 제공하는 Context에 접근할 수 있으며, 이벤트는 React 트리를 따라 자식에서 부모로 전파됩니다.

매개변수

- `children` : JSX의 일부(`<div />` 또는 `<SomeComponent />`), `<Fragment>` (`<>...</>`), 문자열이나 숫자 또는 이들의 배열과 같이 React로 렌더링할 수 있는 모든 것입니다.
- `domNode` : `document.getElementById()` 가 반환하는 것과 같은 일부 DOM 노드. 노드가 이미 존재해야 합니다. 업데이트 중에 다른 DOM 노드를 전달하면 Portal 콘텐츠가 다시 생성됩니다.
- **optional** `key` : A unique string or number to be used as the portal's key.

반환값

`createPortal` 은 JSX를 포함하거나 React 컴포넌트를 반환할 수 있는 React 노드를 반환합니다. React가 렌더링 출력에서 이를 발견하면, 제공된 `children` 을 제공된 `domNode` 안에 배치합니다.

주의 사항

- Portal의 이벤트는 DOM 트리가 아닌 React 트리를 따라 전파됩니다. 예를 들어, Portal 내부를 클릭했을 때 포털이 `<div onClick>` 으로 감싸져 있으면 해당 `onClick` 핸들러가 실행됩니다. 이로 인해 문제가 발생하면 Portal 내부에서 이벤트 전파를 중지하거나 Portal 자체를 React 트리에서 위로 이동하세요.

사용법

DOM의 다른 부분으로 렌더링하기

*Portal*을 사용하면 컴포넌트가 일부 자식을 DOM의 다른 위치로 렌더링할 수 있습니다. 이를 통해 컴포넌트의 일부가 어떤 컨테이너에 있든 그 컨테이너에서 "탈출"할 수 있습니다. 예를 들어, 컴포넌트는 페이지의 나머지 부분 위쪽과 바깥에 표시되는 모달 대화상자나 툴팁을 보여줄 수 있습니다.

Portal을 생성하려면 일부 JSX 와 함께 `createPortal` 의 결과를 렌더링하고 DOM 노드가 있어야 할 위치 를 지정합니다.

```
import { createPortal } from 'react-dom';

function MyComponent() {
  return (
    <div style={{ border: '2px solid black' }}>
      <p>This child is placed in the parent div.</p>
      {createPortal(
        <p>This child is placed in the document body.</p>,
        document.body
      )}
    </div>
  );
}
```

React는 사용자가 전달한 JSX 에 대한 DOM 노드를 사용자가 제공한 DOM 노드 안에 넣습니다.

Portal이 없다면 두 번째 `<p>` 는 상위 `<div>` 안에 배치되지만, Portal은 이를 `document.body` 안으로 "순간이동"시킵니다.

App.js

다운로드 새로고침 Clear 포크

```
import { createPortal } from 'react-dom';

export default function MyComponent() {
  return (
```

```
        <div style={{ border: '2px solid black' }}>
          <p>This child is placed in the parent div.</p>
          {createPortal(
            <p>This child is placed in the document body.</p>,
            document.body
          )}
        </div>
      );
```

두 번째 단락이 테두리가 있는 부모 `<div>` 외부에 시각적으로 어떻게 나타나는지 주목하세요. 개발자 도구로 DOM 구조를 검사하면 두 번째 `<p>` 가 `<body>` 안에 직접 배치된 것을 확인할 수 있습니다.

```
<body>
  <div id="root">
    ...
      <div style="border: 2px solid black">
        <p>This child is placed inside the parent div.</p>
      </div>
    ...
  </div>
  <p>This child is placed in the document body.</p>
</body>
```

Portal은 DOM 노드의 물리적 배치만 변경합니다. 이외의 모든 측면에서, Portal로 렌더링된 JSX는 이를 렌더링하는 React 컴포넌트의 자식 노드처럼 동작합니다. 예를 들어, 자식은 부모 트리가 제공하는 Context에 접근할 수 있으며, 이벤트는 React 트리를 따라 자식에서 부모로 전파됩니다.

Portal이 있는 모달 대화 상자 렌더링하기

대화 상자를 불러오는 컴포넌트가 `overflow: hidden` 또는 대화 상자에 영향을 주는 다른 스타일이 있는 컨테이너 안에 있더라도 Portal을 사용하여 페이지의 나머지 부분 위에 떠 있는 모달 대화 상자를 만들 수 있습니다.

이 예시에서 두 컨테이너에는 모달 대화 상자에 영향을 주는 스타일이 있지만, Portal에 렌더링된 스타일은 영향을 받지 않는데, 그 이유는 DOM에서 모달이 상위 JSX 요소에 포함되지 않기 때문입니다.

App.js NoPortalExample.js PortalExample.js Mod 새로고침 Clear 포크

```
import NoPortalExample from './NoPortalExample';
import PortalExample from './PortalExample';

export default function App() {
  return (
    <>
      <div className="clipping-container">
        <NoPortalExample  />
      </div>
      <div className="clipping-container">
        <PortalExample />
      </div>
```

주의하세요!

Portal을 사용할 때 앱의 접근성accessibility, a11y이 준수되는지 확인하는 것이 중요합니다. 예를 들어 사용자가 Portal 안팎으로 자연스럽게 초점을 이동할 수 있도록 키보드 포커스를 관리해야 할 수 있습니다.

모달을 만들 때는 WAI-ARIA 모달 제작 관행을 따르세요. 커뮤니티 패키지를 사용하는 경우 해당 패키지가 접근 가능한지, 이 가이드라인을 따르고 있는지 확인하세요.

React 컴포넌트를 React가 아닌 서버 마크업으로 렌더링하기

Portal은 React 루트가 React로 빌드되지 않은 정적 또는 서버 렌더링 페이지의 일부일 때 유용할 수 있습니다. 예를 들어, 페이지가 Rails와 같은 서버 프레임워크로 빌드된 경우 사이드바와 같은 정적 영역 내에 인터랙티브 영역을 만들 수 있습니다. 여러 개의 개별 React 루트를 사용하는 것과 비교하여, Portal을 사용하면 앱의 일부가 DOM의 다른 부분에 렌더링 되더라도 공유 상태를 가진 단일 React 트리로 취급할 수 있습니다.

index.js index.html App.js 새로고침 Clear 포크

```
import { createPortal } from 'react-dom';

const sidebarContentEl = document.getElementById('sidebar-content');

export default function App() {
  return (
    <>
      <MainContent />
```

```
{createPortal(
  <SidebarContent />,
  sidebarContentEl
```

자세히 보기

React 컴포넌트를 React가 아닌 DOM 노드로 렌더링하기

Portal을 사용해 React 외부에서 관리되는 DOM 노드의 콘텐츠를 관리할 수도 있습니다. 예를 들어, React가 아닌 맵 위젯과 통합하고 팝업 안에 React 콘텐츠를 렌더링하고 싶다고 가정해 봅시다. 이렇게 하려면 렌더링할 DOM 노드를 저장할 `popupContainer` 상태 변수를 선언하세요.

```
const [popupContainer, setPopupContainer] = useState(null);
```

서드파티 위젯을 만들 때 위젯이 반환하는 DOM 노드를 저장하여 렌더링할 수 있도록 합니다.

```
useEffect(() => {
  if (mapRef.current === null) {
    const map = createMapWidget(containerRef.current);
```

```
    mapRef.current = map;
    const popupDiv = addPopupToMapWidget(map);
    setPopupContainer(popupDiv);
  }
}, []);
```

이렇게 하면 `createPortal` 을 사용하여 React 콘텐츠가 사용 가능해지면 `popupContainer` 로 렌더링할 수 있습니다.

```
return (
  <div style={{ width: 250, height: 250 }} ref={containerRef}>
    {popupContainer !== null && createPortal(
      <p>Hello from React!</p>,
      popupContainer
    )}
  </div>
);
```

다음은 실행할 수 있는 전체 예시입니다.

App.js map-widget.js 새로고침 Clear 포크

```
import { useRef, useEffect, useState } from 'react';
import { createPortal } from 'react-dom';
import { createMapWidget, addPopupToMapWidget } from './map-widget.js';

export default function Map() {
  const containerRef = useRef(null);
  const mapRef = useRef(null);
  const [popupContainer, setPopupContainer] = useState(null);

  useEffect(() => {
    if (mapRef.current === null) {
      const map = createMapWidget(containerRef.current);
```

자세히 보기

이전
API

다음
flushSync

감사의 말

약관