

API 참고서 > 클라이언트 API >

# createRoot

createRoot로 브라우저 DOM 노드 안에 React 컴포넌트를 표시하는 루트를 생성할 수 있습니다.

```
const root = createRoot(domNode, options?)
```

- [레퍼런스](#)

- [createRoot\(domNode, options?\)](#)
- [root.render\(reactNode\)](#)
- [root.unmount\(\)](#)

- [사용법](#)

- 온전히 React만으로 작성된 앱 렌더링하기
- React로 부분적으로 작성된 페이지 렌더링하기
- 루트 컴포넌트 업데이트하기
- 프로덕션 환경에서 오류 로깅하기

- [문제 해결](#)

- 루트를 생성했는데 아무것도 표시되지 않습니다
- 오류 발생: “root.render에 두 번째 인수를 전달했습니다”
- “대상 컨테이너가 DOM 엘리먼트가 아닙니다”라는 오류가 발생합니다.
- “함수가 React 자식으로 유효하지 않습니다” 오류가 발생합니다.
- 서버에서 렌더링된 HTML이 처음부터 다시 생성됩니다

---

## 레퍼런스

### [createRoot\(domNode, options?\)](#)

`createRoot` 를 호출하면 브라우저 DOM 엘리먼트 안에 콘텐츠를 표시할 수 있는 React 루트를 생성합니다.

```
import { createRoot } from 'react-dom/client';

const domNode = document.getElementById('root');
const root = createRoot(domNode);
```

React는 `domNode` 에 대한 루트를 생성하고 그 안에 있는 DOM을 관리합니다. 루트를 생성한 후에는 `root.render` 를 호출해 그 안에 React 컴포넌트를 표시해야 합니다.

```
root.render(<App />);
```

온전히 React만으로 작성된 앱에서는 일반적으로 루트 컴포넌트에 대한 `createRoot` 호출이 하나만 있습니다. 페이지의 일부에 React를 “뿌려서” 사용하는 페이지의 경우에는 루트를 필요로 하는 만큼 작성할 수 있습니다.

[아래 예시를 참고하세요.](#)

## 매개변수

- `domNode` : [DOM 엘리먼트](#). React는 DOM 엘리먼트에 대한 루트를 생성하고 렌더링된 React 콘텐츠를 표시하는 `render` 와 같은 함수를 루트에서 호출할 수 있도록 합니다.
- **optional** `options` : React 루트에 대한 옵션을 가진 객체입니다.
  - **optional** `onCaughtError` : React가 Error Boundary에서 오류를 잡을 때 호출되는 콜백. Error Boundary에서 잡은 `error` 와 `componentStack` 을 포함하는 `errorInfo` 객체와 함께 호출됩니다.
  - **optional** `onUncaughtError` : 오류가 Error Boundary에 의해 잡히지 않을 때 호출되는 콜백. 오류가 발생한 `error` 와 `componentStack` 을 포함하는 `errorInfo` 객체와 함께 호출됩니다.
  - **optional** `onRecoverableError` : React가 오류로부터 자동으로 복구될 때 호출되는 콜백. React가 던지는 `error` 와 `componentStack` 을 포함하는 `errorInfo` 객체와 함께 호출됩니다. 복구 가능한 오류는 원본 오류 원인을 `error.cause` 로 포함할 수 있습니다.

- **optional identifierPrefix**: React가 `useId`에 의해 생성된 ID에 사용하는 문자열 접두사. 같은 페이지에서 여러개의 루트를 사용할 때 충돌을 피하는 데 유용합니다.

## 반환값

`createRoot`는 `render` 와 `unmount` 두 가지 메서드를 포함한 객체를 반환합니다.

## 주의 사항

- 앱이 서버에서 렌더링 되는 경우 `createRoot()`는 사용할 수 없습니다. 대신 `hydrateRoot()`를 사용하세요.
- 앱에서 `createRoot` 호출이 단 한번만 있을 가능성이 높습니다. 프레임워크를 사용하는 경우 프레임워크가 이 호출을 대신 수행할 수도 있습니다.
- 컴포넌트의 자식이 아닌 DOM 트리의 다른 부분(예: 모달 또는 툴팁)에 JSX 조각을 렌더링하려는 경우, `createRoot` 대신 `createPortal`을 사용하세요.

## root.render/reactNode

`root.render`를 호출하여 `JSX` 조각("React 노드")을 React 루트의 브라우저 DOM 노드에 표시합니다.

```
root.render(<App />);
```

React는 `root`에 `<App />`을 표시하고 그 안에 있는 DOM을 관리합니다.

아래 예시를 참고하세요.

## 매개변수

- `reactNode`: 표시하려는 `React` 노드. 일반적으로 `<App />`과 같은 `JSX` 조각이 되지만, `createElement()`로 작성한 React 엘리먼트, 문자열, 숫자, `null`, `undefined` 등을 전달할 수도 있습니다.

## 반환값

`root.render`는 `undefined`를 반환합니다.

## 주의 사항

- `root.render` 를 처음 호출하면 React는 React 컴포넌트를 렌더링하기 전에 React 루트 내부의 모든 기존 HTML 콘텐츠를 지웁니다.
- 서버에서 또는 빌드 중에 React에 의해 생성된 HTML이 루트의 DOM 노드에 포함된 경우, 대신 이벤트 핸들러를 기존 HTML에 첨부하는 `hydrateRoot()` 를 사용하세요.
- 동일한 루트에서 `render` 를 두 번 이상 호출하면, React는 필요에 따라 DOM을 업데이트하여 사용자가 전달한 최신 JSX를 반영합니다. React는 이전에 렌더링 된 트리와 “비교”해서 재사용할 수 있는 부분과 다시 만들어야 하는 부분을 결정합니다. 동일한 루트에서 `render` 를 다시 호출하는 것은 루트 컴포넌트에서 `set` 함수를 호출하는 것과 비슷합니다. React는 불필요한 DOM 업데이트를 피합니다.
- 렌더링이 시작된 이후에는 동기적으로 실행되지만, `root.render(...)` 자체는 비동기적입니다. 즉, `root.render()` 이후에 작성된 코드가 해당 렌더링의 `useLayoutEffect` 나 `useEffect` 보다 먼저 실행될 수 있습니다. 일반적인 상황에서는 이러한 동작도 문제 없이 잘 작동하며, 대부분 수정이 필요하지 않습니다. 다만, Effect의 실행 순서가 중요한 경우에는 `flushSync` 로 `root.render(...)` 호출을 감싸면 초기 렌더링이 완전히 동기적으로 실행되도록 보장할 수 있습니다.

```
const root = createRoot(document.getElementById('root'));
root.render(<App />);
// ▶ HTML에는 아직 렌더링된 <App /> 내용이 포함되지 않습니다.
console.log(document.body.innerHTML);
```

## `root.unmount()`

`root.unmount` 를 호출하면 React 루트 내부에서 렌더링된 트리를 삭제합니다.

```
root.unmount();
```

완전히 React만으로 작성된 앱에는 일반적으로 `root.unmount` 에 대한 호출이 없습니다.

이 함수는 주로 React 루트의 DOM 노드(또는 그 조상 노드)가 다른 코드에 의해 DOM에서 제거될 수 있는 경우에 유용합니다. 예를 들어 DOM에서 비활성 탭을 제거하는 jQuery 탭 패널을 상상해 보세요. 탭이 제거되면 그 안에 있는 모든 것(내부의 React 루트를 포함)이 DOM에서 제거됩니다. 이 경우 `root.unmount` 를 호출하여 제거된 루트의 콘텐츠 관리를 “중지”하도록 React에 지시해야 합니다. 그렇지 않으면 제거된 루트 내부의 컴포넌트는 구독과 같은 전역 리소스를 정리하고 확보하는 법을 모르는 채로 있게 됩니다.

`root.unmount` 를 호출하면 루트에 있는 모든 컴포넌트가 마운트 해제되고, 트리상의 이벤트 핸들러나 State가 제거되며, 루트 DOM 노드에서 React가 “분리”됩니다.

## 매개변수

`root.unmount` 는 매개변수를 받지 않습니다.

## 반환값

`root.unmount` returns `undefined`.

## 주의 사항

- `root.unmount` 를 호출하면 트리의 모든 컴포넌트가 마운트 해제되고 루트 DOM 노드에서 React가 “분리”됩니다.
- `root.unmount` 를 한 번 호출한 후에는 같은 루트에서 `root.render` 를 다시 호출할 수 없습니다. 마운트 해제된 루트에서 `root.render` 를 호출하려고 하면 “마운트 해제된 루트를 업데이트할 수 없습니다.”Cannot update an unmounted root 오류가 발생합니다. 그러나 해당 노드의 이전 루트가 마운트 해제된 후 동일한 DOM 노드에 새로운 루트를 만들 수는 있습니다.

## 사용법

### 온전히 React만으로 작성된 앱 렌더링하기

앱이 온전히 React만으로 작성된 경우, 전체 앱에 대해 단일 루트를 생성하세요.

```
import { createRoot } from 'react-dom/client';
const root = createRoot(document.getElementById('root'));
```

```
root.render(<App />);
```

일반적으로 이 코드는 시작할 때 한 번만 실행하면 됩니다.

1. HTML에 정의된 브라우저 DOM 노드를 찾으세요.

2. 앱 내부에 React 컴포넌트를 표시하세요.

[index.js](#) [index.html](#) [App.js](#)

↺ 새로고침 X Clear ⌂ 포크

```
import { createRoot } from 'react-dom/client';
import App from './App.js';
import './styles.css';

const root = createRoot(document.getElementById('root'));
root.render(<App />);
```

앱이 온전히 React만으로 작성된 경우, 추가적으로 루트를 더 만들거나 `root.render`를 다시 호출할 필요가 없습니다.

이 시점부터 React는 전체 앱의 DOM을 관리합니다. 컴포넌트를 더 추가하려면 App 컴포넌트 안에 중첩시키세요. UI 업데이트는 각 컴포넌트의 State를 통해 수행할 수 있습니다. 모달이나 툴팁과 같은 추가 콘텐츠를 DOM 노드 외부에 표시해야 하는 경우 Portal로 렌더링하세요.

## ▣ 중요합니다!

HTML이 비어있으면, 앱의 자바스크립트 코드가 로드되고 실행될 때까지 사용자에게 빈 페이지가 표시됩니다.

```
<div id="root"></div>
```

This can feel very slow! To solve this, you can generate the initial HTML from your components [on the server or during the build](#). Then your visitors can read text, see images, and click links before any of the JavaScript code loads. We recommend [using a framework](#) that does this optimization out of the box. Depending on when it runs, this is called *server-side rendering (SSR)* or *static site generation (SSG)*.

## ❗ 주의하세요!

서버 측 렌더링이나 정적 사이트 생성을 사용하는 앱은 `createRoot` 대신 [hydrateRoot](#)를 호출해야 합니다. 그러면 React는 DOM 노드를 파괴하고 다시 생성하는 대신 HTML으로부터 *Hydrate*(재사용)합니다.

## React로 부분적으로 작성된 페이지 렌더링하기

페이지가 [React만으로 작성되지 않은 경우](#), React가 관리하는 각 최상위 UI에 대한 루트를 생성하기 위해 `createRoot`를 여러 번 호출할 수 있습니다. 루트마다 `root.render`를 호출함으로써

각각 다른 콘텐츠를 표시할 수 있습니다.

다음 예시에서는 서로 다른 두 개의 React 컴포넌트를 `index.html` 파일에 정의된 두 개의 DOM 노드에 렌더링합니다.

[index.js](#) [index.html](#) [Components.js](#)

↪ 새로고침 X Clear ☰ 포크

```
import './styles.css';
import { createRoot } from 'react-dom/client';
import { Comments, Navigation } from './Components.js';

const navDomNode = document.getElementById('navigation');
const navRoot = createRoot(navDomNode);
navRoot.render(<Navigation />);

const commentDomNode = document.getElementById('comments');
const commentRoot = createRoot(commentDomNode);
commentRoot.render(<Comments />);
```

`document.createElement()`를 사용하여 새 DOM 노드를 생성하고 문서에 수동으로 추가할 수도 있습니다.

```
const domNode = document.createElement('div');
const root = createRoot(domNode);
root.render(<Comment />);
document.body.appendChild(domNode); // You can add it anywhere in the document
```

DOM 노드에서 React 트리를 제거하고 이 트리가 사용하는 모든 리소스를 정리하려면 `root.unmount` 를 호출하세요.

```
root.unmount();
```

이 기능은 React 컴포넌트가 다른 프레임워크로 작성된 앱 내부에 있는 경우에 주로 유용합니다.

## 루트 컴포넌트 업데이트하기

같은 루트에서 `render` 를 두 번 이상 호출할 수도 있습니다. 컴포넌트 트리 구조가 이전 렌더링과 일치하는 한, React는 [기존 State를 유지](#)합니다. 다음 예시에서 입력창에 어떻게 타이핑하든 관계없이, 매 초 반복되는 `render` 호출로 인한 업데이트가 아무런 문제를 일으키지 않음을 주목하세요.

[index.js](#) [App.js](#)

↪ 새로고침 × Clear ⌂ 포크

```
import { createRoot } from 'react-dom/client';
import './styles.css';
import App from './App.js';

const root = createRoot(document.getElementById('root'));

let i = 0;
setInterval(() => {
  root.render(<App counter={i} />);
  i++;
}, 1000);
```

`render` 를 여러 번 호출하는 경우는 흔하지 않습니다. 일반적으로는, 컴포넌트가 [State를 업데이트합니다.](#)

## 프로덕션 환경에서 오류 로깅하기

React는 기본적으로 모든 오류를 콘솔에 출력합니다. 사용자 정의 오류 보고 기능을 구현하기 위해서 `onUncaughtError`, `onCaughtError`, `onRecoverableError` 와 같은 에러 핸들러 루트 옵션을 제공할 수 있습니다.

```
import { createRoot } from "react-dom/client";
import { reportCaughtError } from "./reportError";

const container = document.getElementById("root");
const root = createRoot(container, {
  onCaughtError: (error, errorInfo) => {
    if (error.message !== "Known error") {
      reportCaughtError({
        error,
        componentStack: errorInfo.componentStack,
      });
    }
  },
});
```

onCaughtError 옵션은 다음 두 개의 인자를 받는 함수입니다.

1. 발생한 error 객체.
2. 오류의 componentStack 정보를 포함한 errorInfo 객체.

onUncaughtError 와 onRecoverableError 를 함께 사용하면, 사용자 정의 오류 보고 시스템을 구현할 수 있습니다.

[index.js](#) [reportError.js](#) [App.js](#)

↺ 새로고침 X Clear ☒ 포크

```
import { createRoot } from "react-dom/client";
import App from "./App.js";
import {
  onCaughtErrorProd,
  onRecoverableErrorProd,
  onUncaughtErrorProd,
} from "./reportError";

const container = document.getElementById("root");
const root = createRoot(container, {
  // 개발 환경에서는 이 옵션들을 제거하고
  // React의 기본 핸들러를 사용하거나 직접 오버레이를 구현하는 것을 권장합니다.
```

▼ 자세히 보기

# 문제 해결

## 루트를 생성했는데 아무것도 표시되지 않습니다

실제로 앱을 루트에 렌더링하는 것을 잊지 않았는지 확인하세요.

```
import { createRoot } from 'react-dom/client';
import App from './App.js';

const root = createRoot(document.getElementById('root'));
root.render(<App />);
```

root.render(...) 명령 없이는 아무것도 표시되지 않습니다.

## 오류 발생: “root.render에 두 번째 인수를 전달했습니다”

흔히 하는 실수는 createRoot의 옵션을 root.render(...)에 전달하는 것입니다.

Console

- ✖ Warning: You passed a second argument to root.render(...) but it only accepts one argument.

해결 방법: 루트 옵션은 root.render(...) 가 아니라 createRoot(...)에 전달하세요.

```
// 🔴 잘못된 방법: root.render는 하나의 인자만 받습니다.
root.render(App, {onUncaughtError});

// ✅ 올바른 방법: 옵션은 createRoot에 전달합니다.
const root = createRoot(container, {onUncaughtError});
root.render(<App />);
```

# ”대상 컨테이너가 DOM 엘리먼트가 아닙니다”라는 오류가 발생합니다.

이 오류는 `createRoot`에 전달한 값이 DOM 노드가 아니라는 뜻입니다.

무슨 상황인지 잘 모르겠다면, 해당 값을 콘솔에 출력해서 확인해 보세요.

```
const domNode = document.getElementById('root');
console.log(domNode); // ???
const root = createRoot(domNode);
root.render(<App />);
```

예를 들어 `domNode`가 `null`이면 `getElementById`가 `null`을 반환했음을 의미합니다. 이는 호출 시점에 문서에 지정된 ID를 가진 노드가 없는 경우에 발생합니다. 여기에는 몇 가지 이유가 있을 수 있습니다.

1. 찾고자 하는 ID가 HTML 파일에서 사용한 ID와 다를 수 있습니다. 오타가 있는지 확인하세요!
2. 번들의 `<script>` 태그는 HTML에서 그보다 뒤에 있는 DOM 노드를 “인식할” 수 없습니다.

또 다른 일반적인 사례는 `createRoot(domNode)` 대신 `createRoot(<App />)`으로 작성했을 경우입니다.

# ”함수가 React 자식으로 유효하지 않습니다” 오류가 발생합니다.

이 오류는 `root.render`에 전달하는 것이 React 컴포넌트가 아님을 의미합니다.

이 오류는 `<Component />` 대신 `Component`로 `root.render`를 호출할 때 발생할 수 있습니다.

```
// 🔴 잘못된 방법: App은 컴포넌트가 아니라 함수입니다.
root.render(App);
```

```
// ✅ 올바른 방법: <App />은 컴포넌트입니다.
root.render(<App />);
```

또는 함수를 호출한 결과 대신 `root.render`에 함수 자체를 전달했을 때도 발생할 수 있습니다.

```
// 🔴 잘못된 방법: createApp은 컴포넌트가 아니라 함수입니다.  
root.render(createApp);  
  
// ✅ 올바른 방법: createApp을 호출하여 컴포넌트를 반환합니다.  
root.render(createApp());
```

## 서버에서 렌더링된 HTML이 처음부터 다시 생성됩니다

앱이 서버에서 렌더링되고 React의 초기 HTML을 포함하는 경우에, 루트를 생성해서 `root.render`를 호출하면, 모든 HTML이 삭제되고 모든 DOM 노드가 처음부터 다시 생성되는 것을 볼 수 있습니다. 이렇게 하면 속도가 느려지고, 포커스와 스크롤 위치가 재설정되며, 그 밖의 다른 사용자 입력들이 손실될 수 있습니다.

서버에서 렌더링된 앱은 `createRoot` 대신 `hydrateRoot`를 사용해야 합니다.

```
import { hydrateRoot } from 'react-dom/client';  
import App from './App.js';  
  
hydrateRoot(  
  document.getElementById('root'),  
  <App />  
);
```

API가 다르다는 점에 유의하세요. 특히, 일반적으로는 `root.render`를 아예 호출하지 않습니다.

이전

[클라이언트 API](#)

다음

[hydrateRoot](#)

## React 학습하기

[빠르게 시작하기](#)

[설치하기](#)

[UI 표현하기](#)

[상호작용성 더하기](#)

[State 관리하기](#)

[탈출구](#)

## API 참고서

[React APIs](#)

[React DOM APIs](#)

## 커뮤니티

[행동 강령](#)

[팀 소개](#)

[문서 기여자](#)

[감사의 말](#)

## 더 보기

[블로그](#)

[React Native](#)

[개인 정보 보호](#)

[약관](#)

