renderToString

주의하세요!

`renderToString`은 스트리밍이나 데이터 대기를 지원하지 않습니다. 대안을 참고하세요.

`renderToString`은 React 트리를 HTML 문자열로 렌더링합니다.

```
const html = renderToString(reactNode, options?)
```

레퍼런스

`renderToString(reactNode, options?)`

서버에서 `renderToString` 을 실행하면 앱을 HTML로 렌더링합니다.

```
import { renderToString } from 'react-dom/server';

const html = renderToString(<App />);
```

클라이언트에서 `hydrateRoot` 를 호출하면 서버에서 생성된 HTML을 상호작용하게 만듭니다.

아래 예시를 참고하세요.

매개변수

- `reactNode` : HTML로 렌더링할 React 노드입니다. 예를 들어 `<App />` 과 같은 JSX 노드입니다.
- **optional** `options` : 서버 렌더링을 위한 객체입니다.
  - **optional** `identifierPrefix` : `useId` 에 의해 생성된 ID에 대해 React가 사용하는 문자열 접두사입니다. 같은 페이지에서 여러 루트를 사용할 때 충돌을 피하기 위해 유용합니다. `hydrateRoot` 에 전달된 접두사와 동일해야 합니다.

반환값

HTML 문자열.

주의 사항

- `renderToString` 는 Suspense 지원에 한계가 있습니다. 컴포넌트가 중단된다면 `renderToString` 는 즉시 해당 폴백을 HTML로 보냅니다.
- `renderToString` 은 브라우저에서 동작하지만, 클라이언트 코드에서 사용하는 것은 권장하지 않습니다.

사용법

React 트리를 HTML 문자열로 렌더링하기

서버 응답과 함께 보낼 수 있는 HTML 문자열로 앱을 렌더링하려면 `renderToString` 을 호출하세요.

```
import { renderToString } from 'react-dom/server';

// 라우트 핸들러 구문은 백엔드 프레임워크에 따라 다릅니다
app.use('/', (request, response) => {
  const html = renderToString(<App />);
  response.send(html);
});
```

이는 React 컴포넌트의 초기 상호작용하지 않는 HTML 출력을 생성합니다. 클라이언트에서 서버에서 생성된 HTML을 *Hydrate*하여 상호작용할 수 있도록 `hydrateRoot` 를 실행해야 합니다.

주의하세요!

`renderToString` 은 스트리밍 또는 데이터 대기를 지원하지 않습니다. 대안을 참고하세요.

대안

서버에서 `renderToString` 을 스트리밍 렌더링으로 마이그레이션

`renderToString` 은 문자열을 즉시 반환하므로, 로딩 중인 콘텐츠를 스트리밍하는 것을 지원하지 않습니다.

가능하면 다음과 같은 완전한 기능을 갖춘 대안을 사용하는 것을 권장합니다.

- Node.js를 사용하는 경우 `renderToPipeableStream` 을 사용하세요.
- Deno와 최신 엣지 런타임에서 Web Stream을 사용하는 경우 `renderToReadableStream` 을 사용하세요.

서버 환경에서 스트림을 지원하지 않는 경우에도 `renderToString` 을 계속 사용할 수 있습니다.

서버에서 `renderToString`을 정적 프리렌더로 마이그레이션

`renderToString`은 문자열을 즉시 반환하므로, 정적 HTML 생성을 위해 데이터 로딩이 완료될 때까지 기다리는 것을 지원하지 않습니다.

가능하면 다음과 같은 완전한 기능을 갖춘 대안을 사용하는 것을 권장합니다.

- Node.js를 사용하는 경우 `prerenderToNodeStream`을 사용하세요.
- Deno와 최신 엣지 런타임에서 Web Streams을 사용하는 경우 `prerender`를 사용하세요.

정적 사이트 생성 환경에서 스트림을 지원하지 않는 경우에는 `renderToString`을 계속 사용할 수 있습니다.

클라이언트 코드에서 `renderToString` 제거하기

클라이언트에서 일부 컴포넌트를 HTML로 변환하기 위해 `renderToString`을 사용하기도 합니다.

```
// 🚩 불필요: 클라이언트에서 `renderToString` 사용하기.
import { renderToString } from 'react-dom/server';

const html = renderToString(<MyIcon />);
console.log(html); // 예를 들어, "<svg>...</svg>"
```

클라이언트에서 `react-dom/server`를 가져오면 불필요하게 번들 크기가 커지므로 피해야 합니다. 브라우저에서 일부 컴포넌트를 HTML로 렌더링해야 할 경우 `createRoot`를 사용하고 DOM에서 HTML을 읽으세요.

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

const div = document.createElement('div');
const root = createRoot(div);
flushSync(() => {
  root.render(<MyIcon />);
```

```
});
console.log(div.innerHTML); // 예를 들어, "<svg>...</svg>"
```

`flushSync` 호출은 `innerHTML` 속성을 읽기 전에 DOM을 업데이트하기 위해 필요합니다.

문제 해결

컴포넌트가 일시 중단되면 HTML에 항상 폴백을 포함합니다.

`renderToString`은 Suspense를 완벽하게 지원하지 않습니다.

일부 컴포넌트가 일시 중단Suspend되거나 (예를 들어, `lazy`와 함께 정의되거나 데이터를 가져올 때) `renderToString`은 콘텐츠가 해결될 때까지 기다리지 않습니다. `renderToString`는 그 위에 가장 가까운 `<Suspense>` 경계를 찾아 `fallback` 프로퍼티를 HTML에 렌더링합니다. 내용Content은 클라이언트 코드가 로드될 때까지 나타나지 않습니다.

이를 해결하려면 권장되는 스트리밍 솔루션 중 하나를 사용하면 좋습니다. 서버 사이드 렌더링의 경우, 서버에서 해결되는 대로 콘텐츠를 작은 단위chunk로 스트리밍할 수 있어 클라이언트 코드가 로드되기 전에 사용자가 페이지가 단계적으로 나타나는 것을 볼 수 있습니다. 정적 사이트 생성의 경우, 정적 HTML을 생성하기 전에 모든 콘텐츠가 해결될 때까지 기다릴 수 있습니다.

이전
renderToStaticMarkup

다음
resume

커뮤니티

더 보기