prerenderToNodeStream

`prerenderToNodeStream`은 Node.js Stream을 사용하여 React 트리를 정적 HTML 문자열로 렌더링합니다.

```
const {prelude, postponed} = await prerenderToNodeStream(reactNode, optio
```


중요합니다!

이 API는 Node.js 전용입니다. Deno나 최신 엣지 런타임처럼 Web Streams를 지원하는 환경에서는 `prerender` 를 사용해야 합니다.

레퍼런스

prerenderToNodeStream(reactNode, options?)

prerenderToNodeStream 을 호출해 앱을 정적 HTML로 렌더링합니다.

```
import { prerenderToNodeStream } from 'react-dom/static';

// 라우트 핸들러 문법은 사용하는 백엔드 프레임워크에 따라 다릅니다.
app.use('/', async (request, response) => {
  const { prelude } = await prerenderToNodeStream(<App />, {
    bootstrapScripts: ['/main.js'],
  });

  response.setHeader('Content-Type', 'text/plain');
  prelude.pipe(response);
});
```

클라이언트에서 hydrateRoot 를 호출해 서버에서 생성된 HTML을 인터랙티브하게 만듭니다.

아래에서 더 많은 예시를 확인하세요.

매개변수

- reactNode : HTML로 렌더링할 React 노드입니다. 예를 들어 <App /> 과 같은 JSX 노드가 해당됩니다. 전체 문서를 나타내야 하므로, App 컴포넌트는 <html> 태그를 렌더링해야 합니다.
- **optional** options : 정적 생성 옵션을 가진 객체입니다.
  - **optional** bootstrapScriptContent : 지정될 경우, 이 문자열이 인라인 <script> 태그에 삽입됩니다.
  - **optional** bootstrapScripts : 페이지에 출력할 <script> 태그의 문자열 URL 배열입니다. hydrateRoot 를 호출하는 <script> 를 포함하려면 이 옵션을 사용하세요. 클라이언트에서 React를 전혀 실행하지 않으려면 생략하세요.
  - **optional** bootstrapModules : bootstrapScripts 와 같지만, <script type="module"> 을 출력합니다.
  - **optional** identifierPrefix : useId 로 생성된 ID에 React가 사용하는 문자열 접두사입니다. 한 페이지에서 여러 개의 루트를 사용할 때 충돌을 피하는 데 유용합니다. hydrateRoot 에 전달한 접두사와 동일해야 합니다.

- **optional** `namespaceURI` : 스트림의 루트 namespace URI를 담은 문자열입니다. 기본값은 일반 HTML입니다. SVG의 경우 `'http://www.w3.org/2000/svg'` , MathML의 경우 `'http://www.w3.org/1998/Math/MathML'` 을 전달하세요.
- **optional** `onError` : 서버 오류가 발생할 때마다, 복구 가능 불가능 여부와 관계없이 호출되는 콜백입니다. 기본적으로 `console.error` 만 호출합니다. 충돌 보고를 기록하도록 재정의하는 경우에도 반드시 `console.error` 를 호출해야 합니다. 셸이 출력되기 전에 상태 코드를 설정하는 데에도 사용할 수 있습니다.
- **optional** `progressiveChunkSize` : 청크의 바이트 수입니다. 기본 휴리스틱에 대해 더 읽어보세요.
- **optional** `signal` : 프리렌더링을 중단하고 나머지를 클라이언트에서 렌더링할 수 있게 하는 중단 신호입니다.

반환값

`prerenderToNodeStream` returns a Promise:

- If rendering the is successful, the Promise will resolve to an object containing:
  - `prelude` : a Node.js Stream. of HTML. You can use this stream to send a response in chunks, or you can read the entire stream into a string.
  - `postponed` : a JSON-serializeable, opaque object that can be passed to `resumeToPipeableStream` if `prerenderToNodeStream` did not finish. Otherwise `null` indicating that the `prelude` contains all the content and no resume is necessary.
- If rendering fails, the Promise will be rejected. Use this to output a fallback shell.

주의 사항

프리렌더링 시 `nonce` 옵션은 사용할 수 없습니다. Nonce는 요청마다 고유해야 하며, CSP로 애플리케이션을 보호할 때 Nonce 값을 프리렌더링 결과에 포함하는 것은 부적절하고 안전하지 않습니다.

중요합니다!

`prerenderToNodeStream` 은 언제 사용해야 하나요?

정적 `prerenderToNodeStream` API는 정적 서버 사이드 생성(SSG)에 사용합니다. `renderToString`과 달리, `prerenderToNodeStream`은 모든 데이터가 로드될 때까지 기다린 후에 성공합니다. 이는 Suspense를 사용해 가져와야 하는 데이터를 포함해, 전체 페이지의 정적 HTML을 생성하는 데 적합합니다. 콘텐츠가 로드되는 동안 스트리밍하려면, `renderToReadableStream`과 같은 스트리밍 서버 사이드 렌더링(SSR) API를 사용하세요.

`prerenderToNodeStream` can be aborted and resumed later with `resumeToPipeableStream` to support partial pre-rendering.

사용법

React 트리를 정적 HTML 스트림으로 렌더링하기

`prerenderToNodeStream`를 호출해 React 트리를 정적 HTML로 렌더링하고, 이를 Node.js Stream에 출력합니다.

```
import { prerenderToNodeStream } from 'react-dom/static';

// 라우터 핸들러 문법은 사용하는 백엔드 프레임워크에 따라 다릅니다.
app.use('/', async (request, response) => {
  const { prelude } = await prerenderToNodeStream(<App />, {
    bootstrapScripts: ['/main.js'],
  });

  response.setHeader('Content-Type', 'text/plain');
  prelude.pipe(response);
});
```

루트 컴포넌트와 함께, 부트스트랩 `<script>` 경로 목록을 제공해야 합니다. 루트 컴포넌트는 **루트 `<html>` 태그를 포함한 전체 문서를 반환해야 합니다.**

예를 들어, 다음과 같을 수 있습니다.

```
export default function App() {
  return (
    <html>
      <head>
        <meta charSet="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="stylesheet" href="/styles.css"></link>
        <title>My app</title>
      </head>
      <body>
        <Router />
      </body>
    </html>
  );
}
```

React는 doctype과 부트스트랩 <script> 태그 를 결과 HTML 스트림에 삽입합니다.

```
<!DOCTYPE html>
<html>
  <!-- ... 컴포넌트에서 생성된 HTML ... -->
</html>
<script src="/main.js" async=""></script>
```

클라이언트에서 부트스트랩 스크립트는 hydrateRoot 를 호출해 document 전체를 hydrate해야 합니다.

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

hydrateRoot(document, <App />);
```

이는 정적 서버 생성 HTML에 이벤트 리스너를 연결해 인터랙티브하게 만듭니다.

자세히 살펴보기

빌드 출력에서 CSS 및 JS 에셋 경로 읽기

자세히 보기

React 트리를 정적 HTML 문자열로 렌더링하기

`prerenderToNodeStream`을 호출해 앱을 정적 HTML 문자열로 렌더링합니다.

```
import { prerenderToNodeStream } from 'react-dom/static';

async function renderToString() {
  const {prelude} = await prerenderToNodeStream(<App />, {
    bootstrapScripts: ['/main.js']
  });

  return new Promise((resolve, reject) => {
    let data = '';
    prelude.on('data', chunk => {
      data += chunk;
    });
    prelude.on('end', () => resolve(data));
    prelude.on('error', reject);
  });
}
```

이렇게 하면 React 컴포넌트의 초기 상호작용하지 않은 HTML 출력이 생성됩니다. 클라이언트에서는 `hydrateRoot`를 호출하여 서버에서 생성된 HTML을 *Hydrate*하고 상호작용하게 만들어야 합니다.

모든 데이터가 로드될 때까지 기다리기

`prerenderToNodeStream`는 모든 데이터가 로드될 때까지 기다린 뒤 정적 HTML 생성을 완료하고 Promise를 해결합니다. 예를 들어 표지 이미지, 친구와 사진이 포함된 사이드바, 게시물 목록을 표시하는 프로필 페이지를 생각해 보겠습니다.

```
function ProfilePage() {
  return (
    <ProfileLayout>
      <ProfileCover />
      <Sidebar>
        <Friends />
        <Photos />
      </Sidebar>
      <Suspense fallback={<PostsGlimmer />}>
        <Posts />
      </Suspense>
    </ProfileLayout>
  );
}
```

예를 들어 `<Posts />`가 데이터를 로드해야 하고, 이 과정에 시간이 걸린다고 가정해 보겠습니다. 이 경우, 이상적으로는 게시물 데이터가 모두 로드된 뒤 HTML에 포함되기를 원할 것입니다. 이를 위해 Suspense를 사용해 데이터 로드가 완료될 때까지 렌더링을 일시 중단할 수 있으며, `prerenderToNodeStream`는 해당 중단된 콘텐츠가 완료될 때까지 기다린 후 정적 HTML로 변환을 완료합니다.

중요합니다!

Suspense를 지원하는 데이터 소스만이 Suspense 컴포넌트를 활성화합니다. 여기에는 다음이 포함됩니다.

- Relay 혹은 Next.js 처럼 Suspense를 지원하는 프레임워크를 사용한 데이터 가져오기.
- `lazy`를 사용한 컴포넌트 코드의 지연로딩.
- `use`를 사용해 Promise의 값을 읽기.

Suspense는 Effect나 이벤트 핸들러 내부에서 데이터가 패칭될 때 이를 **감지하지 않습니다.**

위 예시의 `Posts` 컴포넌트에서 데이터를 로드하는 구체적인 방법은 사용하는 프레임워크에 따라 다릅니다. Suspense를 지원하는 프레임워크를 사용한다면, 해당 프레임워크의 데이터 패칭 문서에서 자세한 내용을 확인할 수 있습니다.

특정 프레임워크를 사용하지 않는 Suspense 지원 데이터 패칭은 아직 지원되지 않습니다. Suspense를 지원하는 데이터 소스를 구현하기 위한 요구 사항은 현재 불안정하고 문서화되어 있지 않습니다. 데이터 소스를 Suspense와 통합하기 위한 공식 API는 React의 향후 버전에서 제공될 예정입니다.

사전 렌더링 중단하기

타임아웃 이후 사전 렌더링을 "포기"하도록 강제할 수 있습니다.

```
async function renderToString() {
  const controller = new AbortController();
  setTimeout(() => {
    controller.abort()
  }, 10000);

  try {
    // Prelude에는 컨트롤러가 중단하기 전에
    // 사전렌더링된 모든 HTML이 포함됩니다.
    const {prelude} = await prerenderToNodeStream(<App />, {
      signal: controller.signal,
    });
    //...
```

불완전한 자식을 가진 모든 Suspense 경계는 폴백 상태로 Prelude에 포함됩니다.

This can be used for partial prerendering together with `resumeToPipeableStream` or `resumeAndPrerenderToNodeStream`.

문제 해결

전체 앱이 렌더링될 때까지 스트림이 시작되지 않았습니다.

`prerenderToNodeStream` 응답은 모든 Suspense 경계가 해결될 때까지 기다리는 것을 포함하여 전체 앱이 렌더링이 완료될 때까지 기다린 후 완료됩니다. 이 API는 정적 사이트 생성(SSG)을 위해 설계되었으며 콘텐츠가 로드되면서 더 많은 콘텐츠를 스트리밍하는 것을 지원하지 않습니다.

콘텐츠가 로드되면서 스트리밍하려면 `renderToPipeableStream`과 같은 스트리밍 서버 렌더링 API를 사용하세요.

이전
prerender

다음
resumeAndPrerender

감사의 말

약관