prerender

`prerender` 는 Web Stream을 사용하여 React 트리를 정적 HTML 문자열로 렌더링합니다.

```
const {prelude, postponed} = await prerender(reactNode, options?)
```


중요합니다!

이 API는 Web Stream에 의존합니다. Node.js에서는 `prerenderToNodeStream` 을 대신 사용하세요.

레퍼런스

prerender(reactNode, options?)

prerender 를 호출하여 앱을 정적 HTML로 렌더링합니다.

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

async function handler(request, response) {
  const {prelude} = await prerender(<App />, {
    bootstrapScripts: ['/main.js']
  });
  return new Response(prelude, {
    headers: { 'content-type': 'text/html' },
  });
}
```

클라이언트에서 hydrateRoot 를 호출하여 서버에서 생성된 HTML을 상호작용할 수 있도록 만듭니다.

아래에서 더 많은 예시를 확인하세요.

매개변수

- reactNode : HTML로 렌더링하려는 React 노드. 예를 들어 <App /> 과 같은 JSX 엘리먼트입니다. 전체 문서를 나타낼 것으로 예상되므로 App 컴포넌트는 <html> 태그를 렌더링해야 합니다.
- **optional** options : 정적 생성 옵션을 가진 객체입니다.
  - **optional** bootstrapScriptContent : 지정될 경우, 해당 문자열은 <script> 태그에 인라인 형식으로 추가됩니다.
  - **optional** bootstrapScripts : 페이지에 표시할 <script> 태그에 대한 문자열 URL 배열입니다. hydrateRoot 를 호출하는 <script> 를 포함하려면 이것을 사용하세요. 클라이언트에서 React를 전혀 실행하지 않으려면 생략하세요.
  - **optional** bootstrapModules : bootstrapScripts 와 유사하지만 대신 <script type="module"> 을 추가합니다.
  - **optional** identifierPrefix : React가 useId 에 의해 생성된 ID를 사용하는 문자열 접두사입니다. 같은 페이지에서 여러 루트를 사용할 때 충돌을 피하는 데 유용합니다. hydrateRoot 에 전달된 것과 동일한 접두사여야 합니다.

- **optional** `namespaceURI` : 스트림의 루트 namespace URI를 가진 문자열입니다. 기본값은 일반 HTML입니다. SVG의 경우 `'http://www.w3.org/2000/svg'` 를, MathML의 경우 `'http://www.w3.org/1998/Math/MathML'` 을 전달합니다.
- **optional** `onError` : 복구 가능 또는 불가능에 관계없이 서버 오류가 발생할 때마다 호출되는 콜백입니다. 기본적으로 `console.error` 만 호출합니다. 이 함수를 재정의하여 크래시 리포트를 로깅하는 경우 `console.error` 를 계속 호출해야 합니다. 또한 셸이 출력되기 전에 상태 코드를 설정하는 데 사용할 수도 있습니다.
- **optional** `progressiveChunkSize` : 청크의 바이트 수입니다. 기본 휴리스틱에 대해 더 읽어보기.
- **optional** `signal` : 사전 렌더링을 중단하고 나머지를 클라이언트에서 렌더링하기 위한 중단 신호Abort Signal를 설정합니다.

반환값

`prerender` returns a Promise:

- If rendering the is successful, the Promise will resolve to an object containing:
  - `prelude` : a Web 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 `resume` if `prerender` 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

`nonce` 는 사전 렌더링할 때 사용할 수 없는 옵션입니다. Nonce는 요청마다 고유해야 하며, CSP로 애플리케이션을 보호하기 위해 Nonce를 사용한다면 Nonce 값을 사전 렌더링 자체에 포함하는 것은 부적절하고 안전하지 않습니다.

중요합니다!

`prerender` 를 언제 사용해야 하나요?

정적 `prerender` API는 정적 사이트 생성SSG, Static Site Generation에 사용됩니다. `renderToString`과 달리 `prerender`는 해결되기 전에 모든 데이터가 로드될 때까지 대기합니다. 이는 Suspense를 사용하여 가져와야 하는 데이터를 포함하여 전체 페이지에 대한 정적 HTML을 생성하는 데 적합합니다. 콘텐츠가 로드되면서 스트리밍하려면 `renderToReadableStream`과 같은 스트리밍 서버 사이드 렌더링(SSR) API를 사용하세요.

`prerender` can be aborted and later either continued with `resumeAndPrerender` or resumed with `resume` to support partial pre-rendering.

사용법

React 트리를 정적 HTML 스트림으로 렌더링하기

`prerender`를 호출해 Readable Web Stream을 통해 React 트리를 정적 HTML로 렌더링합니다.

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

async function handler(request) {
  const {prelude} = await prerender( <App /> , {
    bootstrapScripts: ['/main.js']
  });
  return new Response(prelude, {
    headers: { 'content-type': 'text/html' },
  });
}
```

루트 컴포넌트와 함께 부트스트랩 `<script>` 경로 목록을 제공해야 합니다. 루트 컴포넌트는 **루트 `<html>` 태그를 포함하여 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>
  <!-- ... HTML from your components ... -->
</html>
<script src="/main.js" async=""></script>
```

클라이언트에서 부트스트랩 스크립트는 hydrateRoot 를 호출하여 전체 document 를 Hydrate 해야 합니다.

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

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

이렇게 하면 정적 서버 생성 HTML에 이벤트 리스너가 연결되어 상호작용하게 만들어집니다.

자세히 살펴보기

빌드 출력에서 CSS 및 JS 에셋 경로 읽기

자세히 보기

React 트리를 정적 HTML 문자열로 렌더링하기

`prerender` 를 호출하여 앱을 정적 HTML 문자열로 렌더링합니다.

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

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

  const reader = prelude.getReader();
  let content = '';
  while (true) {
    const {done, value} = await reader.read();
    if (done) {
      return content;
    }
    content += Buffer.from(value).toString('utf8');
  }
}
```

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

모든 데이터 로드 대기

`prerender` 는 정적 HTML 생성을 완료하고 해결되기 전에 모든 데이터가 로드될 때까지 대기합니다. 예를 들어, 표지, 친구와 사진이 있는 사이드바, 게시물 목록을 보여주는 프로필 페이지를 생각해 보세요.

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

`<Posts />` 가 데이터를 로드해야 하는데 시간이 걸린다고 가정해 보겠습니다. 이상적으로는 게시물이 완료될 때까지 기다려서 HTML에 포함하고 싶을 것입니다. 이를 위해 Suspense를 사용하여 데이터를 일시 중단할 수 있으며, `prerender` 는 일시 중단된 콘텐츠가 완료될 때까지 기다린 후 정적 HTML로 해결됩니다.

중요합니다!

Suspense를 지원하는 데이터 소스만 Suspense 컴포넌트를 활성화합니다. 이는 다음과 같습니다.

- Relay와 Next.js 같은 Suspense가 가능한 프레임워크를 사용한 데이터 가져오기.
- `lazy` 를 사용한 지연 로딩 컴포넌트.
- `use` 를 사용한 Promise 값 읽기.

Suspense는 Effect나 이벤트 핸들러 내부에서 데이터를 가져올 경우, **이를 감지하지 못합니다.**.

`Posts` 컴포넌트에서 데이터를 불러오는 정확한 방법은 프레임워크에 따라 다릅니다. Suspense를 지원하는 프레임워크를 사용하는 경우, 데이터를 가져오는 자세한 방법은 해당 프레임워크 문서에서 찾을 수 있습니다.

독자적인 프레임워크를 사용하지 않는 Suspense 지원 데이터 가져오기는 아직 지원되지 않습니다. Suspense를 지원하는 데이터 소스를 구현하기 위한 요구 사항은 불안정하고 문서화되지 않았습니다. 데이터 소스를 Suspense와 통합하기 위한 공식 API는 React의 향후 버전에서 출시할 예정입니다.

사전 렌더링 중단

타임아웃 후 사전 렌더링을 "포기"하도록 강제할 수 있습니다.

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

  try {
    // prelude에는 컨트롤러가 중단되기 전에
    // 사전 렌더링된 모든 HTML이 포함됩니다.
    const {prelude} = await prerender(<App />, {
      signal: controller.signal,
    });
    //...
```

불완전한 자식을 가진 모든 Suspense 경계는 폴백 상태로 prelude에 포함됩니다.

This can be used for partial prerendering together with `resume` or `resumeAndPrerender`.

문제 해결

전체 앱이 렌더링될 때까지 스트림이 시작되지 않습니다

`prerender` 응답은 모든 Suspense 경계가 해결될 때까지 기다리는 것을 포함하여 전체 앱이 렌더링을 완료할 때까지 기다린 후 해결됩니다. 이는 사전에 정적 사이트 생성(SSG)을 위해 설계되었으며 콘텐츠가 로드되면서 더 많은 콘텐츠를 스트리밍하는 것을 지원하지 않습니다.

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

이전
정적 API

다음
prerenderToNodeStream