resume

`resume` streams a pre-rendered React tree to a Readable Web Stream.

```
const stream = await resume(reactNode, postponedState, options?)
```


중요합니다!

This API depends on Web Streams. For Node.js, use `resumeToNodeStream` instead.

Reference

`resume(node, postponedState, options?)`

Call `resume` to resume rendering a pre-rendered React tree as HTML into a Readable Web Stream.

```
import { resume } from 'react-dom/server';
import {getPostponedState} from './storage';

async function handler(request, writable) {
  const postponed = await getPostponedState(request);
  const resumeStream = await resume(<App />, postponed);
  return resumeStream.pipeTo(writable)
}
```

See more examples below.

Parameters

- `reactNode`: The React node you called `prerender` with. For example, a JSX element like `<App />`. It is expected to represent the entire document, so the `App` component should render the `<html>` tag.
- `postponedState`: The opaque `postpone` object returned from a prerender API, loaded from wherever you stored it (e.g. redis, a file, or S3).
- **optional** `options`: An object with streaming options.
  - **optional** `nonce`: A `nonce` string to allow scripts for `script-src` Content-Security-Policy.
  - **optional** `signal`: An abort signal that lets you abort server rendering and render the rest on the client.
  - **optional** `onError`: A callback that fires whenever there is a server error, whether recoverable or not. By default, this only calls `console.error`. If you override it to log crash reports, make sure that you still call `console.error`.

Returns

`resume` returns a Promise:

- If `resume` successfully produced a shell, that Promise will resolve to a Readable Web Stream. that can be piped to a Writable Web Stream..
- If an error happens in the shell, the Promise will reject with that error.

The returned stream has an additional property:

- `allReady` : A Promise that resolves when all rendering is complete. You can `await stream.allReady` before returning a response for crawlers and static generation. If you do that, you won't get any progressive loading. The stream will contain the final HTML.

Caveats

- `resume` does not accept options for `bootstrapScripts` , `bootstrapScriptContent` , or `bootstrapModules` . Instead, you need to pass these options to the `prerender` call that generates the `postponedState` . You can also inject bootstrap content into the writable stream manually.
- `resume` does not accept `identifierPrefix` since the prefix needs to be the same in both `prerender` and `resume` .
- Since `nonce` cannot be provided to prerender, you should only provide `nonce` to `resume` if you're not providing scripts to prerender.
- `resume` re-renders from the root until it finds a component that was not fully pre-rendered. Only fully prerendered Components (the Component and its children finished prerendering) are skipped entirely.

Usage

Resuming a prerender

index.js index.html demo-helpers.js 새로고침 Clear 포크

```
import {
  flushReadableStreamToFrame,
  getUser,
  Postponed,
  sleep,
} from "./demo-helpers";
import { StrictMode, Suspense, use, useEffect } from "react";
import { prerender } from "react-dom/static";
import { resume } from "react-dom/server";
import { hydrateRoot } from "react-dom/client";

function Header() {
```

자세히 보기

Further reading

Resuming behaves like `renderToReadableStream`. For more examples, check out the usage section of `renderToReadableStream`.
The usage section of `prerender` includes examples of how to use `prerender` specifically.

이전
renderToString

다음
resumeToPipeableStream

커뮤니티

더 보기