Script Optimization

Application Scripts

To load a third-party script for all routes, import `next/script` and include the script directly in your custom `_app`:

```
import Script from 'next/script';

export default function MyApp({ Component, pageProps }) {
  return (
    <>
      <Component {...pageProps} />
      <Script src="https://example.com/script.js" />
    </>
  );
}
```

This script will load and execute when *any* route in your application is accessed. Next.js will ensure the script will **only load once**, even if a user navigates between multiple pages.

Recommendation: We recommend only including third-party scripts in specific pages or layouts in order to minimize any unnecessary impact to performance.

Strategy

Although the default behavior of `next/script` allows you load third-party scripts in any page or layout, you can fine-tune its loading behavior by using the `strategy` property:

- `beforeInteractive`: Load the script before any Next.js code and before any page hydration occurs.
- `afterInteractive`: (**default**) Load the script early but after some hydration on the page occurs.
- `lazyOnload`: Load the script later during browser idle time.
- `worker`: (experimental) Load the script in a web worker.

Refer to the `next/script` API reference documentation to learn more about each strategy and their use cases.

Offloading Scripts To A Web Worker (Experimental)

Warning: The `worker` strategy is not yet stable and does not yet work with the `app` directory. Use with caution.

Scripts that use the `worker` strategy are offloaded and executed in a web worker with Partytown↗. This can improve the performance of your site by dedicating the main thread to the rest of your application code.

This strategy is still experimental and can only be used if the `nextScriptWorkers` flag is enabled in `next.config.js`:

next.config.js

```js
module.exports = {
  experimental: {
    nextScriptWorkers: true,
  },
};
```

Then, run `next` (normally `npm run dev` or `yarn dev`) and Next.js will guide you through the installation of the required packages to finish the setup:

Terminal

```
npm run dev
```

You'll see instructions like these: Please install Partytown by running `npm install @builder.io/partytown`

Once setup is complete, defining `strategy="worker"` will automatically instantiate Partytown in your application and offload the script to a web worker.

pages/home.tsx

```tsx
import Script from 'next/script';

export default function Home() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="worker" />
    </>
  );
}
```

There are a number of trade-offs that need to be considered when loading a third-party script in a web worker. Please see Partytown's tradeoffs ↗ documentation for more information.

Inline Scripts

Inline scripts, or scripts not loaded from an external file, are also supported by the Script component. They can be written by placing the JavaScript within curly braces:

```
<Script id="show-banner">
  {`document.getElementById('banner').classList.remove('hidden')`}
</Script>
```

Or by using the `dangerouslySetInnerHTML` property:

```
<Script
  id="show-banner"
  dangerouslySetInnerHTML={{
    __html: `document.getElementById('banner').classList.remove('hidden')`,
  }}
/>
```

Warning: An `id` property must be assigned for inline scripts in order for Next.js to track and optimize the script.

Executing Additional Code

Event handlers can be used with the Script component to execute additional code after a certain event occurs:

- `onLoad`: Execute code after the script has finished loading.
- `onReady`: Execute code after the script has finished loading and every time the component is mounted.
- `onError`: Execute code if the script fails to load.

These handlers will only work when `next/script` is imported and used inside of a Client Component where `"use client"` is defined as the first line of code:

pages/index.tsx

```
import Script from 'next/script';
```

```

export default function Page() {
  return (
    <>
      <Script
        src="https://example.com/script.js"
        onLoad={() => {
          console.log('Script has loaded');
        }}
      />
    </>
  );
}
```

Refer to the `next/script` API reference to learn more about each event handler and view examples.

Additional Attributes

There are many DOM attributes that can be assigned to a `<script>` element that are not used by the Script component, like `nonce` ↗ or custom data attributes ↗. Including any additional attributes will automatically forward it to the final, optimized `<script>` element that is included in the HTML.

pages/index.tsx

```tsx
import Script from 'next/script';

export default function Page() {
  return (
    <>
      <Script
        src="https://example.com/script.js"
        id="example-script"
        nonce="XUENAJFW"
        data-test="script"
      />
    </>
  );
}
```