NEXT.JS

> Menu

App Router > ... > Components > <Link>

<Link>

`<Link>` is a React component that extends the HTML `<a>` element to provide prefetching and client-side navigation between routes. It is the primary way to navigate between routes in Next.js.

app/page.tsx TypeScript

```tsx
import Link from 'next/link'

export default function Page() {
  return <Link href="/dashboard">Dashboard</Link>
}
```

Props

Here's a summary of the props available for the Link Component:

Prop	Example	Type	Required
`href`	`href="/dashboard"`	String or Object	Yes
`replace`	`replace={false}`	Boolean	-
`scroll`	`scroll={false}`	Boolean	-

Prop	Example	Type	Required
`prefetch`	`prefetch={false}`	Boolean or null	-

Good to know: `<a>` tag attributes such as `className` or `target="_blank"` can be added to `<Link>` as props and will be passed to the underlying `<a>` element.

`href` (required)

The path or URL to navigate to.

```
<Link href="/dashboard">Dashboard</Link>
```

`href` can also accept an object, for example:

```
// Navigate to /about?name=test
<Link
  href={{
    pathname: '/about',
    query: { name: 'test' },
  }}
>
  About
</Link>
```

`replace`

Defaults to `false`. When `true`, `next/link` will replace the current history state instead of adding a new URL into the browser's history ↗ stack.

app/page.tsx TypeScript

```
import Link from 'next/link'

export default function Page() {
  return (
```

```
      <Link href="/dashboard" replace>
        Dashboard
      </Link>
    )
  }
```

`scroll`

Defaults to `true`. The default behavior of `<Link>` **is to scroll to the top of a new route or to maintain the scroll position for backwards and forwards navigation.** When `false`, `next/link` will *not* scroll to the top of the page after a navigation.

app/page.tsx TypeScript

```tsx
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/dashboard" scroll={false}>
      Dashboard
    </Link>
  )
}
```

Good to know:

- Next.js will scroll to the Page if it is not visible in the viewport upon navigation.

`prefetch`

Prefetching happens when a `<Link />` component enters the user's viewport (initially or through scroll). Next.js prefetches and loads the linked route (denoted by the `href`) and its data in the background to improve the performance of client-side navigations. Prefetching is only enabled in production.

- `null` **(default)**: Prefetch behavior depends on whether the route is static or dynamic. For static routes, the full route will be prefetched (including all its data). For dynamic routes, the partial route down to the nearest segment with a `loading.js` boundary will be prefetched.

- `true`: The full route will be prefetched for both static and dynamic routes.
- `false`: Prefetching will never happen both on entering the viewport and on hover.

app/page.tsx TypeScript

```tsx
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/dashboard" prefetch={false}>
      Dashboard
    </Link>
  )
}
```

Examples

Linking to Dynamic Routes

For dynamic routes, it can be handy to use template literals to create the link's path.

For example, you can generate a list of links to the dynamic route `app/blog/[slug]/page.js`:

app/blog/page.js

```jsx
import Link from 'next/link'

function Page({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blog/${post.slug}`}>{post.title}</Link>
        </li>
      ))}
    </ul>
  )
}
```

If the child is a custom component that wraps an `<a>` tag

If the child of `Link` is a custom component that wraps an `<a>` tag, you must add `passHref` to `Link`. This is necessary if you're using libraries like styled-components↗. Without this, the `<a>` tag will not have the `href` attribute, which hurts your site's accessibility and might affect SEO. If you're using ESLint, there is a built-in rule `next/link-passhref` to ensure correct usage of `passHref`.

```jsx
import Link from 'next/link'
import styled from 'styled-components'

// This creates a custom component that wraps an <a> tag
const RedLink = styled.a`
  color: red;
`

function NavLink({ href, name }) {
  return (
    <Link href={href} passHref legacyBehavior>
      <RedLink>{name}</RedLink>
    </Link>
  )
}

export default NavLink
```

- If you're using emotion↗'s JSX pragma feature (`@jsx jsx`), you must use `passHref` even if you use an `<a>` tag directly.
- The component should support `onClick` property to trigger navigation correctly

If the child is a functional component

If the child of `Link` is a functional component, in addition to using `passHref` and `legacyBehavior`, you must wrap the component in `React.forwardRef`↗:

```
import Link from 'next/link'

// `onClick`, `href`, and `ref` need to be passed to the DOM element
// for proper handling
const MyButton = React.forwardRef(({ onClick, href }, ref) => {
  return (
    <a href={href} onClick={onClick} ref={ref}>
      Click Me
    </a>
  )
})

function Home() {
  return (
    <Link href="/about" passHref legacyBehavior>
      <MyButton />
    </Link>
  )
}

export default Home
```

Replace the URL instead of push

The default behavior of the `Link` component is to `push` a new URL into the `history` stack. You can use the `replace` prop to prevent adding a new entry, as in the following example:

```
<Link href="/about" replace>
  About us
</Link>
```

Disable scrolling to the top of the page

The default behavior of `Link` is to scroll to the top of the page. When there is a hash defined it will scroll to the specific id, like a normal `<a>` tag. To prevent scrolling to the top / hash `scroll={false}` can be added to `Link`:

```
<Link href="/#hashid" scroll={false}>
  Disables scrolling to the top
</Link>
```

Middleware

It's common to use Middleware for authentication or other purposes that involve rewriting the user to a different page. In order for the `<Link />` component to properly prefetch links with rewrites via Middleware, you need to tell Next.js both the URL to display and the URL to prefetch. This is required to avoid un-necessary fetches to middleware to know the correct route to prefetch.

For example, if you want to serve a `/dashboard` route that has authenticated and visitor views, you may add something similar to the following in your Middleware to redirect the user to the correct page:

middleware.js

```
export function middleware(req) {
  const nextUrl = req.nextUrl
  if (nextUrl.pathname === '/dashboard') {
    if (req.cookies.authToken) {
      return NextResponse.rewrite(new URL('/auth/dashboard', req.url))
    } else {
      return NextResponse.rewrite(new URL('/public/dashboard', req.url))
    }
  }
}
```

In this case, you would want to use the following code in your `<Link />` component:

```tsx
import Link from 'next/link'
import useIsAuthed from './hooks/useIsAuthed'

export default function Page() {
  const isAuthed = useIsAuthed()
  const path = isAuthed ? '/auth/dashboard' : '/public/dashboard'
  return (
    <Link as="/dashboard" href={path}>
      Dashboard
    </Link>
  )
}
```

Version History

Version	Changes
`v13.0.0`	No longer requires a child `<a>` tag. A codemod is provided to automatically update your codebase.
`v10.0.0`	`href` props pointing to a dynamic route are automatically resolved and no longer require an `as` prop.
`v8.0.0`	Improved prefetching performance.
`v1.0.0`	`next/link` introduced.

Previous
<Image>

Next
<Script>