Dynamic Routes

Examples

Dynamic Routing

Defining routes by using predefined paths is not always enough for complex applications. In Next.js you can add brackets to a page (`[param]`) to create a dynamic route (a.k.a. url slugs, pretty urls, and others).

Consider the following page `pages/post/[pid].js`:

```
import { useRouter } from 'next/router'

const Post = () => {
  const router = useRouter()
  const { pid } = router.query

  return <p>Post: {pid}</p>
}

export default Post
```

Any route like `/post/1`, `/post/abc`, etc. will be matched by `pages/post/[pid].js`. The matched path parameter will be sent as a query parameter to the page, and it will be merged with the other query parameters.

For example, the route `/post/abc` will have the following `query` object:

```
{ "pid": "abc" }
```

Similarly, the route `/post/abc?foo=bar` will have the following `query` object:

```
{ "foo": "bar", "pid": "abc" }
```

However, route parameters will override query parameters with the same name. For example, the route `/post/abc?pid=123` will have the following `query` object:

```
{ "pid": "abc" }
```

Multiple dynamic route segments work the same way. The page `pages/post/[pid]/[comment].js` will match the route `/post/abc/a-comment` and its `query` object will be:

```
{ "pid": "abc", "comment": "a-comment" }
```

Client-side navigations to dynamic routes are handled with `next/link`. If we wanted to have links to the routes used above it will look like this:

```
import Link from 'next/link'

function Home() {
  return (
    <ul>
      <li>
        <Link href="/post/abc">
          <a>Go to pages/post/[pid].js</a>
        </Link>
      </li>
      <li>
        <Link href="/post/abc?foo=bar">
          <a>Also goes to pages/post/[pid].js</a>
        </Link>
      </li>
      <li>
        <Link href="/post/abc/a-comment">
          <a>Go to pages/post/[pid]/[comment].js</a>
        </Link>
      </li>
    </ul>
  )
}

export default Home
```

Read our docs for Linking between pages to learn more.

Catch all routes

Examples

Catch All Routes

Dynamic routes can be extended to catch all paths by adding three dots (`...`) inside the brackets. For example:

- `pages/post/[...slug].js` matches `/post/a`, but also `/post/a/b`, `/post/a/b/c` and so on.

  Note: You can use names other than `slug`, such as: `[...param]`

Matched parameters will be sent as a query parameter (`slug` in the example) to the page, and it will always be an array, so, the path `/post/a` will have the following `query` object:

```
{ "slug": ["a"] }
```

And in the case of `/post/a/b`, and any other matching path, new parameters will be added to the array, like so:

```
{ "slug": ["a", "b"] }
```

Optional catch all routes

Catch all routes can be made optional by including the parameter in double brackets (`[[...slug]]`).

For example, `pages/post/[[...slug]].js` will match `/post`, `/post/a`, `/post/a/b`, and so on.

The main difference between catch all and optional catch all routes is that with optional, the route without the parameter is also matched (`/post` in the example above).

The `query` objects are as follows:

```
{ } // GET `/post` (empty object)
{ "slug": ["a"] } // `GET /post/a` (single-element array)
{ "slug": ["a", "b"] } // `GET /post/a/b` (multi-element array)
```

Caveats

- Predefined routes take precedence over dynamic routes, and dynamic routes over catch all routes. Take a look at the following examples:
  - `pages/post/create.js` - Will match `/post/create`
  - `pages/post/[pid].js` - Will match `/post/1`, `/post/abc`, etc. But not `/post/create`
  - `pages/post/[...slug].js` - Will match `/post/1/2`, `/post/a/b/c`, etc. But not `/post/create`, `/post/abc`
- Pages that are statically optimized by Automatic Static Optimization will be hydrated without their route parameters provided, i.e `query` will be an empty object (`{}`).

  After hydration, Next.js will trigger an update to your application to provide the route parameters in the `query` object.

Related

For more information on what to do next, we recommend the following sections:

next/link: Enable client-side transitions with next/link.

Routing: Learn more about routing in Next.js.