RouterLink, Navigate & NavigateByUrl to Navigate Routes

3 Comments / 5 minutes of reading / February 1, 2024

← Query Parameters | Angular Tutorial | Pass Data to Route →

The Angular router allows us to navigate between the routes using either RouterLink directive or imperatively using router.navigate or router.navigatebyUrl method of the router service.

In this tutorial, we look at these methods more closely.

Table of Contents

Navigating between Angular routes

You can navigate between routes in Angular 2 in two ways

1. Using RouterLink Directive
2. Via Code

RouterLink directive

We looked at how to navigate using the RouterLink directive in the previous tutorials.

The RouterLink is a directive, which you can use to bind any clickable HTML element to a Route. When the user clicks on the HTML element the router will navigate to the associated Route.

For Example

```
<li><a [routerLink]="['product']">Product</a></li>
```

Will map to URL “/product” and renders the associated ProductComponent

Navigating Using Code

You can also navigate imperatively by using the code. This is done using the router service, which provides navigate and navigatebyUrl methods via which you can perform route changes.

router.navigate

Use this method, if you want to Navigate to a route using the link parameters array. The first argument to the navigate method is link parameters array, which is similar to what we provide while defining the routerlink directive

Navigate Method always uses the absolute path unless you provide a starting point.

navigate.navigateByUrl

Use this method if you want to navigate to a URL by using the absolute path. The first argument is a string containing the complete URL.

NavigateByUrl Method always uses the absolute path

To use both these methods, we need to inject router service into our component as shown below

```

constructor(private _router:Router){
}

```

And then invoke

```

this._router.navigate(['product']

```

Or

```

this._router.navigateByUrl('product')
```

```
3
```

To navigate to the desired route.

Link Parameters array

LINK Parameters array is an array of strings, which you must specify as argument to either to routerlink directive or navigate method for navigation to work

We need to specify the path of the route and route parameters that go into the route URL.

The following example resolves to the URL path '/product/detail/1'

```
1
2 <li><a [routerLink]="['product/detail/1']">Product 1 Overview</a></li>
3
```

or

```
1
2 this._router.navigate(['product/detail/1'])
3
```

Relative and Absolute Paths in Routes

The Angular routes resemble directory-like tree structures.

Hence, We can use directory-like syntaxes like add / (root node), ./(current node), or ../(Parent node) in the link parameters array

The First segment of the link parameters array can be prepended with “/“, “./“, or “../“

If the First segment of the route starts with “/“, then the path is considered to be an Absolute path

If the First segment begins with “./” or it does not begin with a slash, then the path is considered to be the relative path.

And if the First segment begins with “. ./“, then the path is relative to the parent route. (one level up)

router.navigate method and relative path

As mentioned earlier navigate method always uses the absolute path. To make Navigate method work with a relative path, we must let know the router where are we in the route tree.

This is done by setting the relativeTo Property to the ActivatedRoute as shown below

```
1
2 this._router.navigate(['detail'], { queryParams: { pageNum: this.pageNum + 1 }, relativeTo
3
```

RouterLink directive and relative path

If you were using a RouterLink to navigate instead of the Router service, you’d use the same link parameters array, but you wouldn’t provide the object with the relativeTo property. The ActivatedRoute is implicit in a RouterLink directive.

Absolute Path Vs Relative Path Which one to Use?

It is recommended to use the Relative path. Using an absolute path breaks our code if the parent URL structure changes. The relative path will not change even if the parent path changes

To go to the parent route

```
<li><a [routerLink]="['../']">Back</a></li>
```

To go to the Sibling route

```
<li><a [routerLink]="['../<sibling>']">Goto sibling</a></li>
```

To go to the child route

```
<li><a [routerLink]="['<Child>']">Goto Child</a></li>
```

NavigationExtras

We can provide the extra options to both router.navigate() or router.navigatebyURL() method.

relativeTo: ActivatedRoute

Enables relative navigation from the current ActivatedRoute. This is applicable only to router.navigate() method.

Example:

The following Navigates to the Detail route from the child route

```
this.router.navigate(['../Detail'], { relativeTo: this.activatedRoute });
```

queryParams: Params

Sets query parameters to the URL. You can refer to the tutorial on How to pass query parameters to the Angular route

Example:

The following code constructs the URL as “/product?page=2”.

```
this.router.navigate(['/products'], { queryParams: { page: 1 } });
```

fragment: string

Sets the hash fragment for the URL.

Example:

The following code constructs the URL as “/home#top”

```
this.router.navigate(['/home'], { fragment: 'top' });
```

preserveQueryParams: boolean

Passes the query parameters of the current route to the next route

Example:

If you are on the route “Product?Page=2”, then clicking on the following will pass the query parameters to the “view” route as “view?Page=2”

```
this.router.navigate(['/view'], { preserveQueryParams: true });
```

queryParamsHandling: QueryParamsHandling

The query parameters of the current route are merged with that of the new route if you set queryParamsHandling=”merge”.

```
this.router.navigate(['/view'], { queryParams: { page: 2 },preserveQueryParams: true, que
```

preserveFragment: boolean

Passes the fragment of the current route to the next navigation. Similar to the preserveQueryParams

```
this.router.navigate(['/view'], { preserveFragment: true });
```

skipLocationChange: boolean

You can change the route, without changing the URL in the browser. This Navigates to

a new URL without pushing a new state into history.

Example:

```
this.router.navigate(['/view'], { skipLocationChange: true });
```

replaceUrl: boolean

The current route is removed from the browser history while navigating to the new route. It replaces the current state in history with the new state.

Example:

```
this.router.navigate(['/view'], { replaceUrl: true });
```

RouterLink

You can provide the extra options to the RouterLink directive, similar to the NavigationExtras. The following options are supported

QueryParams: Params

```
<a [routerLink]="['product']" [queryParams]="{ page:2}" }>Page 2</a>
```

preserveQueryParams:boolean

```
```

```
<a [routerLink]="['product']" { preserveQueryParams: "true" }">Page 2</a>
```

queryParamsHandling : QueryParamsHandling

```
<a [routerLink]="['product']" { queryParams: { page: 2 }, queryParamsHandling: "merge" }
```

Fragment: string

```
<a [routerLink]="['product']" { fragment: 'top' }">Page 2</a>
```

PreserveFragment: boolean

```
<a [routerLink]="['product']" { preserveFragment: true }">Page 2</a>
```

SkipLocationChange: boolean

```
<a [routerLink]="['product']" { skipLocationChange: true">Page 2</a>
```

ReplaceUrl: boolean

```
<a [routerLink]="['product']" { replaceUrl: true">Page 2</a>
```