How do I access Query Parameters in Angular?

Posted At : February 19, 2019 9:00 AM | Posted By : Jeffry Houser
Related Categories: Professional, Angular

When creating web applications, it is common to add data to a URL that you need to access. For example, one might build an app with a list view that shows a lot of things, such as superheros. Then they have a detail view which shows an expanded detail of a single superhero, such as aquaman.

This post has nothing to do with superheroes. It is all about how to access that item in the URL from an Angular app.

The Setup

I created a default angular application with routing enabled. I created one route, named view1:

```
const routes: Routes = [
  { path: 'view1/:value', component: View1Component },
  { path: '**', redirectTo: 'view1/default' },
];
```

The route has a single query param, named value. That is what we'll try to access. The route directs to a new component, which you can create with the Angular CLI:

```
ng generate component view1
```

Make sure to replace the main app.component.html with the router outlet:

```
<router-outlet></router-outlet>
```

One more thing to do. Open up the view1.component.ts file. We'll need to inject the ActivatedRoute into the component:

```
constructor(private activatedRoute: ActivatedRoute) { }
```

You'll also need to import this:

```
import {ActivatedRoute} from '@angular/router';
```

You can find the full source here. Let's look at what it takes to access the query param.

Option 1: Params

I set up all these examples in the ngOnInit() method of the View1Component. This is the first:

```
this.activatedRoute.params.subscribe((params) => {
      console.log(params.value);
    });
```

The ActivatedRoute has a params Observable. We can subscribe to it, and when it resolves we can access the values. The params object sent to the subscribe function is just an object:

It is a collection of name value pairs and we can access our value using params.value.

Option 2: paramMap

The second option is to access the paramMap. This is also done in the ngOnInit() method against the activatedRoute:

```
this.activatedRoute.paramMap.subscribe((params ) => {
      console.log(params['params'].value);
      console.log(params.get('value'));
    });
```

paramMap is an observable that resolves to a paramMap. Instead of a generic object, it uses the JavaScript map:

Using a map is a more modern, or ES6, way to handle things instead of using a generic object. The map is better suited for a collection of key value pairs, because it provides an API for accessing

values instead of something easily changed.

Option 3: Router Snapshot

The final option I'll show today is to access the snapshot of the activated route. The snapshot mirrors a lot of similar properties to the ActivatedRoute. The main difference is that the snapshot is the route's values at a certain moment in time, whereas the activatedRoute is not. That is why the params, or paramMap, are Observables in the ActivatedRoute, but actual values in the snapshot.

Access the params on the value like this:

```
console.log(this.activatedRoute.snapshot.params.value);
```

You'll see something like this:

- You can also get at the paramMap, value:

```
console.log(this.activatedRoute.snapshot.paramMap.get('value'));
```

You'll see something like this:

Overall, this is similar to using the Observables, except without the Observable part. When accessing the activatedRoute in the ngOnInit, the snapshot should be identical. But, if you need to access something from an unrelated route, or are in the process of switching routes, that is why you'd access the snapshot.

How do you check if a URL variable exists?

When I envisioned this post, I thought I'd write about how to tell if a URL variable exists, but I realized the meat of the post was about finding and accessing the URL variables. If you need to check to see if a value exists, I recommend using the paramMap. For a value that does exist:

```
if (params.has('value')) {
        console.log('it has value');
      } else {
        console.log('No value');
      }
```

And for a value that won't exist in our demo:

```
if (params.has('noValue')) {
        console.log('it has a novalue');
      } else {
        console.log('No noValue exists');
      }
```

We use the has() function the params map to check for the existence of the key.

Final Thoughts

Don't forget to grab the code behind this post for runnable samples.

I'm not sure why there are so many ways to get at this data; my preference is to use the paramMap. What about you?