ThirdRock TECHKNO

Angular Async Pipe

Krunal Shah

MAR 06, 2020 | 3 MIN READ

What is the angular async pipe?

The async pipe in angular will subscribe to an Observable or Promise and returns the latest value it has emitted. When a new value is emitted, the async pipe marks the component to be checked for changes. When the component gets destroyed, the async pipe unsubscribes automatically to avoid potential memory leaks.

#Prerequisites

- Node installed on your machine
- NPM installed on your machine
- Basic Knowledge of JavaScript
- Installing Angular CLI : **npm install -g @angular/cli**

How can we use it in our Angular application?

We can use the async pipe in Angular application by including the CommonModule which exports all the basic Angular directives and pipes, such as NgIf, NgForOf, DecimalPipe, and so on. These are then re-exported by BrowserModule, which is

included automatically in the root AppModule, when you create a new app with the CLI new command.

Syntax of AsyncPipe

The below syntax shows how we can use the async pipe in our template(html) file.
{{obj_expression | async}}

Why should you use the async pipe?

There are many ways to subscribe to Observable or Promise. The default way, certainly without angular, is to subscribe to the observable manually and update a separate property with the value. Let's take an example. Let's create a component called without-async-pipe.component.ts. It's code looks like

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Observable, Observer, Subscription } from 'rxjs';

@Component({
  selector: 'without-asyc-pipe',
  template: `<div>{{data}}</div>`
})
export class WithoutAsyncPipeComponent implements OnInit, OnDestroy  {
  data: string;
  dataSubscription: Subscription;

  ngOnInit(){
    this.dataSubscription = this.getAPIData().subscribe((res) => {
      this.data = res;
    });
  }

  ngOnDestroy(){
    this.dataSubscription.unsubscribe();
  }

   getAPIData(): Observable<string> {
     return new Observable<string>((observer: Observer<string>) => {
        setTimeout(() => observer.next("This is a data From API Response"), 3000);
   });
  }
}
```

Now register this component in declarations[] of app.module.ts & to display this component, inject this component selector in app.component.html. So app.component.html code looks like

```
<without-asyc-pipe></without-asyc-pipe>
```

So why should you use the async pipe then? It turns out the code above is not all we need to do! Because we subscribed to the observable manually, we also need to manually unsubscribe. Otherwise, we risk a memory leak when the component is destroyed. That's why we added "this.dataSubscription.unsubscribe();" in ngOnDestroy() life cycle hook.

To avoid this manually doing unsubscribe in ngOnDestroy() life cycle hook, we should take advantage & use async pipe because it will automatically unsubscribe when component gets destroyed.

#Example 1: Using AsyncPipe with Observables

Let's create a component called time.component.ts which displays updated Date & Time every second with async pipe. It's code looks like

```
import { Component } from '@angular/core';
import { Observable, Observer } from 'rxjs';

@Component({
  selector: 'async-observable-pipe',
  template: '<div><code>observable|async</code>: Time: {{ time | async }}</div>'
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Observer<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

#Example 2: Using AsyncPipe with Promises

Let's create a component called promise.component.ts which displays async data after a few seconds with async pipe. It's code looks like

```
import { Component } from '@angular/core';

@Component({
  selector: 'async-pipe',
  template: `
 <div>
  <h4>AsyncPipe</h4>
  <p>{{ promise | async }}</p>
 </div>
  `
})
export class AsyncPipeComponent {
  public promise: Promise<string>;
  constructor() {
    this.promise = this.getPromise();
  }

  getPromise(): Promise<string> {
     return new Promise((resolve, reject) => {
       setTimeout(() => resolve("Promise complete!"), 3000);
     });
  }
}
```

Conclusion

Async Pipe is a very powerful feature. There is no need to unsubscribe manually in the component. Angular handles subscriptions of async pipes for us automatically using ngOnDestroy.

. . . .