academeez

RXJS

4. Closing the Observable

Closing Observables

academeez

- One question you have to ask yourself… is my async code finite or infinite?
- Recall that there will be 3 ways of creating an Observable:
  - Creating yourself
  - Using RXJS creating observable operators
  - Using method from library
- When creating an Observable yourself, and it is finite, you will have to close the Observable
- When using a created Observable you need to know if that Observable will close, if it’s not closing you might want to detach the listeners.
- Not detaching listeners might cause leaks in your app!

Self closing Observable

academeez

- Self closing Observable represents finite data push, which means it will push data to the listeners finite amount of times and then close
- That Observalbe will self close from within the async method it is wrapping
- An Observable can close with **complete** or with **error**
- If you are creating the Observable yourself and it is finite, it is your responsibility to make sure to close it.
- If the Observable is given to you from operators or external library you can assume the creator closed the Observable when it’s finished.

Observable closing with success

academeez

- This Observable is closing successfully
- This is when you create the Observable, when you get the Observable from external source, if the observable is finite than the complete might be called by the creator
- Example: in angular you can call an Ajax request which will call the complete after server response

```
const helloObservable: Observable<string> = new Observable((observer) => {
  observer.next('Hello Listeners!');
  observer.complete()
});
```

Observable closing with error

academeez

- This Observable is closing with an error
- This is when you create the Observable, when you get the Observable from external source, if the observable can get an error than the creator called the error method
- Example: in angular you can call an Ajax request which might fail and call the error if there is no internet connection

```
const helloObservable: Observable<string> = new Observable((observer) => {
  observer.next('Hello Listeners!');
  observer.error(new Error('Something happened!'));
});
```

What happens to the listener?

- What happens to the listener if the Observable is closing?
- On the **subscribe** method the listener can add an error and complete callbacks which will be called according to how the Observable closed
- The listener will get the **Error** passed from the **Observable**

```
helloObservable.subscribe(
  (msg) => console.log(msg), // This is called on every next
  (error) => console.log(error.message), // this is called when close with error
  () => console.log('Observable completed'), // this is called when close with complete
);
```

Closing Observable by detaching listener

academeez

- Another way an Observable can close is by detaching the listener
- In this case it does not matter if the Observable is finite or infinite, the data stream will close
- It’s highly recommended to close every infinite Observable when it is not used. Otherwise you might get a leak!
- The **subscribe** method returns a **Subscription** object which we can use to detach a listener

Subscription

academeez

- **Subscription** object is returned from the **subscribe** method
- Using that object we can detach a listener
- A single Subscription can represent more that one listener observable connection, in that case you can detach multiple connections at once.
- Listeners complete will not be called

```
const sub: Subscription = new Subscription()
sub.add(helloObservable.subscribe()); // our subscription contains 1 connection
sub.add(helloObservable.subscribe()); // our subscription contains 2 connections

sub.unsubscribe(); // detach 2 connections
```

Memory leak warning!

academeez

- Remember to unsubscribe from infinite Observables, otherwise you might get a link
- Example in Angular

```
@Component({...})
export class MemoryLeakComponent extends OnInit, OnDestroy {
  private _sub = new Subscription();

  ngOnInit() {
    this._sub.add(
      interval(1000).
        subscribe(
          (num: number) => console.log(`counting: ${num}`)
        )
    );
  }

  ngOnDestroy() {
    this._sub.unsubscribe()
  }
}
```

Cleanup

- When you create the Observable yourself (not getting it from operators or external source), the Observable will sometimes need to do some cleanups to the items opened in the async method.

- You can return a function from the Observable async method that will be used when the Observable close for cleanup

```
const intervalObservable: Observable<string> = new Observable((observer) => {
  const intervalId = setInterval(() => {
    observer.next('Hello Listeners!');
  }, 1000);

  return () => {
    clearInterval(intervalId);
  }
});
```

Summary

academeez

- When we are dealing with Observables, we need to think about closing them
- We need to know if the Observable is finite or infinite?
- If it's finite and I'm creating the Observable I need to make sure to close it with **complete** or **error**
- If your Observable is infinite remember to detach the listeners when needed
- The Observable can return a function to clean the opened resource.

academeez

Thank You

Next Lesson: 5. Subjects