OBSERVABLES & RXJS

🕮 OBSERVABLES

Observables provide support for passing messages between publishers and subscribers in your application. Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.

🕮 OBSERVABLES

```
import { Observable } from 'rxjs';

const observable = new Observable(subscriber => {
  subscriber.next(1);
  subscriber.next(2);
  subscriber.next(3);
  setTimeout(() => {
    subscriber.next(4);
    subscriber.complete();
  }, 1000);
});

observable.subscribe({
  next(x) { console.log('got value ' + x); },
  error(err) { console.error('something wrong occurred: ' + err); },
  complete() { console.log('done'); }
});
console.log('just after subscribe');
```

```
got value 1
got value 2
got value 3
just after subscribe
got value 4
done
```

TBC

🕮 OBSERVER

- next
- error
- complete

🕮 RXJS

Creating Observables

- of(…items)
- from(iterable)
- Interval(number)

🕮 OBSERVABLES

```
1. // Create simple observable that emits three values
2. const myObservable = of(1, 2, 3);
3.
4. // Create observer object
5. const myObserver = {
6.   next: x => console.log('Observer got a next value: ' + x),
7.   error: err => console.error('Observer got an error: ' + err),
8.   complete: () => console.log('Observer got a complete notification'),
9. };
10.
11. // Execute with the observer object
12. myObservable.subscribe(myObserver);
```

TBC

🕮 RXJS

RxJS provides an implementation of the Observable type, which is needed until the type becomes part of the language and until browsers support it. The library also provides utility functions for creating and working with observables.

🕮 RXJS OPERATORS

- filter
- map
- pipe
- …

TBC

🕮 RXJS ERROR HANDLING OPERATORS

- catchError
- retry

🕮 OBSERVABLES IN ANGULAR

- Event emitter
- Http
- Async pipe
- Reactive forms

TBC

🕮 OBSERVER VS PROMISE

- Observables are declarative; computation does not start until subscription. Promises execute immediately on creation.
- Observables provide many values. Promises provide one.
- Observables differentiate between chaining and subscription. Promises only have .then() clauses.
- Observables subscribe() is responsible for handling errors. Promises push errors to the child promises.

TBC

🕮 SUBJECT

- Observables can have multiple subscribers, but each of them will get unique values. Values won’t be shared
- With subject we can have multiple subscribers with multiple copies of data.
- Subject is both: Observable and Observer
- Methods
  - subscribe()
  - next()
  - complete()

🕮 SUBJECT VARIATIONS

- Subject (If you subscribe later, you have to wait to get new value)
- BehaviorSubject (Always stores last emitted value in value property)
- ReplaySubject (Can store old values and distribute to new subscribers)
- AsyncSubject (Only emits value when completed)

TBC

🕮 ASYNC PIPE AND UNSUBSCRIBING

- Async pipe
  - subscribes to observable and gets value
  - unsubscribes automatically on component destroy or even before
- If you subscribe in component
  - always unsubscribe it in onDestroy() hook
  - use .pipe(until($destroy)).subscribe()

TBC

HERE GOES!

TBC