Subjects

The EventEmitter of RXJS

Subjects

academeez

- Similar to Observable, Subject also shouts to listeners
- You call **next**, **error**, **complete** on the **Subject** and not from inside a function like Observable
- **Subject** does not duplicate a data stream like observable it represents a single data stream for all the listeners
- Subject will maintain a list of listeners and will call them synchronously with each next

```
const helloSubject: Subject<string> = new Subject();

helloSubject.subscribe((msg) => console.log(msg));

helloSubject.next('hello');
```

Subject extends Observable

academeez

- **Subject** extends **Observable**
- **Subject** implements **SubscriptionLike**

```
export declare class Subject<T> extends Observable<T> implements SubscriptionLike
{
  // ...
}
```

Close a Subject

- You can close a Subject by calling it’s **error** or **complete**

```
// close a subject with an error
helloSubject.error(new Error('something happened'));

// close a subject with success
helloSubject.complete();
```

Unsubscribe from Subject

academeez

- Subject will manage a list of listeners
- You can detach all listeners
- You can detach a single listener

```
const sub: Subscription = helloSubject.subscribe();

// remove one listener
sub.unsubscribe();

// remove all listeners
helloSubject.unsubscribe();
```

Subject usage

- Subject is more usable for use cases where the data stream should be one
  - One source of data

From Observable to Subject

academeez

- There will be times when you have an Observable but want's the data as a single source.
- You can transform Observable for a single data source, which means it won't be duplicated for every listener

```
// the observable we want to transform
const obsToTransform = of([1,2,3]);

// The subject which will hold the single source of data
const transformerSubject = new Subject();

// the transformed observable
const transformedObs = obsToTransform.pipe(
  multicast(transformerSubject)
)
```

From Subject to Observable

academeez

- You can also transform a Subject to Observable.
- This is useful to supply the subject without the ability for the consumers to call next

```
// the next will be exposed
const helloSubject: Subject<string> = new Subject();

// the next is hidden
// can supply this to consumer
const obsForConsumers: Observable<string> = helloSubject.asObservable();
```

Types of Subjects

academeez

- RXJS ships with different kind of Subjects
- Each Subject has a certain behaviour regarding the data inside the subject.
- Let’s go over the different types of Subjects

BehaviourSubject

academeez

- The BehaviourSubject is initiated with an initial value
- The BehaviourSubject always saves the last value from next
- You can also access the last value
- Every Subscriber will be called first time synchronously with the last value

```
const helloSubject: BehaviorSubject<string> = new BehaviorSubject('hello');

helloSubject.subscribe(
  (greeting: string) => console.log(`Subscriber A ${greeting}`)
);

helloSubject.next('hi');

helloSubject.subscribe(
  (greeting: string) => console.log(`Subscriber B ${greeting}`)
);
```

ReplySubject

academeez

- The ReplySubject holds a history of values
- That history can be determined by the number of last values or by time

```
// will store up to 5 items
// will discard old values after 1 second
const helloSubject: ReplaySubject<string> = new ReplaySubject(5, 1000);

helloSubject.next('Hello!');
helloSubject.next('Hi!');
helloSubject.next('Hey!');

helloSubject.subscribe((greeting: string) => console.log(greeting));
```

AsyncSubject

academeez

- Subscribers will jump after this Subject call complete
- The Subscribers will jump with the last value

```
const helloSubject: AsyncSubject<string> = new AsyncSubject();

helloSubject.subscribe((greeting: string) => {
  console.log(greeting);
});

helloSubject.next('Hello!');
helloSubject.next('Hi!');
helloSubject.next('Hey!');
helloSubject.complete();
```

Summary

academeez

- With Subject we can use RXJS world with a single data stream and not a duplicated one for every listener
- The behaviour will be more similar to event emitter
- It allows us also to transform our observable to subject or transform our subject to observable
- RXJS ships with different types of Subject we can use for different scenarios

academeez

Thank You

Next Lesson: Operators