

RxJS Workshop



Agenda

- Lecture
- Exercises
- Exercise Review

What Is RxJS?

A functional reactive programming JavaScript/TypeScript library for creating and manipulating observable data streams.

What Is An Observable?

An observable is a representation of a stream, or discrete emission of data that can arrive over time.

Example:

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

// grab button reference
const button = document.getElementById('button');

// create an observable of button clicks
const buttonClick$ = fromEvent(button, 'click');
```

Subscription

A subscription is an object that is created when you "subscribe" to an observable event stream that enables you to "unsubscribe" from the stream.

Subscription

Example:

```
// ...import rxjs & set up button click observable

// using a function
const subscription = buttonClick$.subscribe(console.log);

// using an object
const subscription = buttonClick$.subscribe({
  // on successful emissions
  next: console.log,
  // on errors
  error: console.log,
  // called once on completion
  complete: () => console.log('complete!')
});
```

[Demo](#)

Subscription

Example:

```
// ...import rxjs & create observable

// addEventListener called
const subscription_1 = observable$.subscribe(event => console.log(event));

// addEventListener called again!
const subscription_2 = observable$.subscribe(event => console.log(event));

// clean up with unsubscribe
subscription_1.unsubscribe();
subscription_2.unsubscribe();
```


Pipe

A function that enables you to defined a sequence of operators for data to pass through.

Operators

Helper functions that allow you to manipulate data from an observable stream and returns an observable with the manipulated data.

map Example:

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';
/*
 * 'of' allows you to deliver values in a sequence
 * In this case, it will emit 1,2,3,4,5 in order.
 */
of(1, 2, 3, 4, 5).pipe(
  // add 1 to each emitted value
  map(value => value + 1)
)
// output: 2, 3, 4, 5, 6
.subscribe(console.log);
```

Operators

filter Example:

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';

of(1, 2, 3, 4, 5).pipe(
  // remove values less than 2
  filter(value => value >= 2)
)
// output: 2, 3, 4, 5, 6
.subscribe(console.log);
```


Creation

Operators that enable the creation of an observable stream.

of

Example:

```
import { of } from 'rxjs';  
//emits a sequence of values of any type  
const source$ = of({ name: 'Brian' }, [1, 2, 3], function hello() {  
  return 'Hello';  
});  
//output: {name: 'Brian'}, [1,2,3], function hello() { return 'Hello' }  
source$.subscribe(console.log);
```

from

Examples:

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

//emit array as a sequence of values
const arraySource$ = from([1, 2, 3, 4, 5]);

//output: 1,2,3,4,5
arraySource$.subscribe(console.log);
```

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

//emit result of promise
const promise$ = from(new Promise(resolve => resolve('Hello World!')));
//output: 'Hello World'
promise$.subscribe(console.log);
```

from

Example:

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

const map = new Map();
map.set(1, 'Hi');
map.set(2, 'Bye');

// emits values using iterator
const mapSource$ = from(map);

//output: [1, 'Hi'], [2, 'Bye']
mapSource$.subscribe(console.log);
```


from

Example:

```
import { from } from 'rxjs';  
  
//emit string as a sequence  
const stringSource$ = from('Hello World');  
  
stringSource$.subscribe(console.log);
```

interval

Example:

```
import { interval } from 'rxjs';  
  
//emit value in sequence every 1 second  
const interval$ = interval(1000);  
  
//output: 0,1,2,3,4,5....  
interval$.subscribe(console.log);
```

timer

Example:

```
import { timer } from 'rxjs';  
  
//emit 0 after 1 second then complete, since no second argument is supplied  
const timer$ = timer(1000);  
  
//output: 0  
timer$.subscribe(console.log);
```

timer

Example:

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

/*
  timer takes a second argument, which is how often to emit subsequent values.
  in this case we will emit first value after 1 second and subsequent values every 2 seconds after
*/
const timer$ = timer(1000, 2000);

//output: 0,1,2,3,4,5.....
const subscribe = timer$.subscribe(console.log);
```

Combination

Operators that enable the merging data from multiple observable streams.

combineLatest

Example:

```
import { timer, combineLatest } from 'rxjs';

// timerOne emits first value at 1s, then once every 4s, emits at 1s, 5s, 9s...
const timerOne$ = timer(1000, 4000);
// timerTwo emits first value at 2s, then once every 4s, emits at 2s, 6s, 10s...
const timerTwo$ = timer(2000, 4000);
// timerThree emits first value at 3s, then once every 4s, emits at 3s, 7s, 11s...
const timerThree$ = timer(3000, 4000);

// continued on next slide
```

combineLatest

Example (continued):

```
// when one timer emits, emit the latest values from each timer as an array
combineLatest(timerOne$, timerTwo$, timerThree$).subscribe(
  ([timerValOne, timerValTwo, timerValThree]) => {
    /*
     Example:
     timerThree first tick: 'Timer One Latest: 0, Timer Two Latest: 0, Timer Three Latest: 0
     timerOne second tick: 'Timer One Latest: 1, Timer Two Latest: 0, Timer Three Latest: 0
     timerTwo second tick: 'Timer One Latest: 1, Timer Two Latest: 1, Timer Three Latest: 0
    */
    console.log(
      `Timer One Latest: ${timerValOne}, Timer Two Latest: ${timerValTwo}, Timer Three Latest: ${timerValThree}`
    );
  }
);
```

[Demo](#)

combineLatest With Projection Function

Example:

```
import { timer, combineLatest } from 'rxjs';

const timerOne$ = timer(1000, 4000);
const timerTwo$ = timer(2000, 4000);
const timerThree$ = timer(3000, 4000);

combineLatest(
  timerOne$,
  timerTwo$,
  timerThree$,
  // combineLatest also takes an optional projection function
  (one, two, three) => `Timer One (Proj) Latest: ${one}, Timer Two (Proj) Latest: ${two}, Timer Three (Proj) Latest: ${three}`
).subscribe(console.log);
```


merge

Example:

```
import { interval, merge } from 'rxjs';
import { mapTo } from 'rxjs/operators';

// emit every 5 seconds
const first$ = interval(5000).pipe(mapTo('FIRST!'));
// emit every 4 seconds
const second$ = interval(4000).pipe(mapTo('SECOND!'));
// emit every 3 seconds
const third$ = interval(3000).pipe(mapTo('THIRD'));
// emit every 2 second
const fourth$ = interval(2000).pipe(mapTo('FOURTH'));

// output: "FOURTH", "THIRD", "SECOND!", "FOURTH", "FIRST!", "THIRD", "FOURTH"
const merged$ = merge(first$, second$, third$, fourth$).subscribe(console.log);
```

[Demo](#)

concat

Example

```
import { of, concat } from 'rxjs';

concat(
  of(1, 2, 3),
  // subscribed after first completes
  of(4, 5, 6),
  // subscribed after second completes
  of(7, 8, 9)
)
// log: 1, 2, 3, 4, 5, 6, 7, 8, 9
.subscribe(console.log);
```

concat Cadveat

Example:

```
import { interval, of, concat } from 'rxjs';  
  
// when source never completes, any subsequent observables never run  
concat(interval(1000), of('This', 'Never', 'Runs'))  
  // log: 1,2,3,4.....  
  .subscribe(console.log);
```

startWith

Example:

```
import { startWith } from 'rxjs/operators';
import { of } from 'rxjs';

//emit (1,2,3)
const source$ = of(1, 2, 3);

//start with 0
const startWithZero$ = source$.pipe(startWith(0));

//output: 0,1,2,3
startWithZero$.subscribe(console.log);
```

withLatestFrom

Example:

```
import { withLatestFrom } from 'rxjs/operators';
import { interval } from 'rxjs';

const firstSource$ = interval(5000); //emit every 5s
const secondSource$ = interval(1000); //emit every 1s
const withLatestFrom$ = firstSource$.pipe(withLatestFrom(
  secondSource$,
  (first, second) => `First Source (5s): ${first} Second Source (1s): ${second}`
));
/*
  output:
  "First Source (5s): 0 Second Source (1s): 4"
  "First Source (5s): 1 Second Source (1s): 9"
  ...
*/
withLatestFrom$.subscribe(console.log);
```

[Demo](#)

Filtering

Operators that enable passing or rejecting data from an observable stream.

filter

Example:

```
import { from } from 'rxjs';
import { filter } from 'rxjs/operators';

// emit (1,2,3,4,5)
const source = from([1, 2, 3, 4, 5]);
// filter out non-even numbers
const evenNumbers$ = source.pipe(filter(num => num % 2 === 0));

// output: "Even number: 2", "Even number: 4"
evenNumbers$.subscribe(evenNumber => console.log(`Even number: ${evenNumber}`));
```

filter

Example:

```
import { from } from 'rxjs';
import { filter } from 'rxjs/operators';

// emit ({name: 'Joe', age: 31}, {name: 'Bob', age:25})
const people$ = from([
  { name: 'Joe', age: 31 },
  { name: 'Bob', age: 25 }
]);

// filter out people with age under 30
const peopleOver30$ = people$.pipe(filter(person => person.age >= 30));

// output: "Over 30: Joe"
peopleOver30$.subscribe(val => console.log(`Over 30: ${val.name}`));
```


take

Example:

```
import { of } from 'rxjs';
import { take } from 'rxjs/operators';

// emit 1,2,3,4,5
const source$ = of(1, 2, 3, 4, 5);
// take the first emitted value then complete
const takeOne$ = source$.pipe(take(1));

// output: 1
takeOne$.subscribe(val => console.log(val));
```

take

Example:

```
import { interval } from 'rxjs';
import { take } from 'rxjs/operators';

// emit value every 1s
const source$ = interval(1000);

// take the first 5 emitted values
const takeFive$ = source$.pipe(take(5));

// output: 0,1,2,3,4
takeFive$.subscribe(console.log);
```

takeUntil

Example:

```
import { interval, timer } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

// emit value every 1s
const source$ = interval(1000);
// after 5 seconds, emit value
const timer$ = timer(5000);

// when timer emits after 5s, complete source
const takeUntilAfterFiveSecs$ = source.pipe(takeUntil(timer$));

// output: 0,1,2,3
takeUntilAfterFiveSecs$.subscribe(console.log);
```

Transformations

Operators that modify data from an observable stream.

map

Example:

```
// RxJS v6+
import { from } from 'rxjs';
import { map } from 'rxjs/operators';

// emit (1,2,3,4,5)
const source$ = from([1, 2, 3, 4, 5]);

// add 10 to each value
const addTen$ = source$.pipe(map(val => val + 10));

// output: 11,12,13,14,15
addTen$.subscribe(console.log);
```

map

Example:

```
import { from } from 'rxjs';
import { map } from 'rxjs/operators';

// emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50})
const people$ = from([
  { name: 'Joe', age: 30 },
  { name: 'Frank', age: 20 },
  { name: 'Ryan', age: 50 }
]);

// grab each persons name, could also use pluck for this scenario
const names$ = people$.pipe(map(({ name }) => name));

// output: "Joe","Frank","Ryan"
names$.subscribe(val => console.log(val));
```

pluck

Example:

```
import { from } from 'rxjs';
import { pluck } from 'rxjs/operators';

// emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50})
const people$ = from([
  { name: 'Joe', age: 30 },
  { name: 'Frank', age: 20 },
  { name: 'Ryan', age: 50 }
]);

// grab each persons name, could also use pluck for this scenario
const names$ = people$.pipe(map(pluck('name')));

// output: "Joe","Frank","Ryan"
names$.subscribe(val => console.log(val));
```

mapTo

Example

```
import { fromEvent } from 'rxjs';
import { map } from 'rxjs/operators';

const click$ = fromEvent(document, 'click');

click$
  .pipe(map(() => 'You clicked!'))
  // 'You clicked!', 'You clicked!'
  .subscribe(console.log);

// equivalent to above
click$
  .pipe(mapTo('You clicked!'))
  // 'You clicked!', 'You clicked!'
  .subscribe(console.log);
```


mergeMap

Example

```
import { of } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

const getPromise = () => new Promise(resolve => resolve('This is a promise.));

// emit 'Hello'
const source$ = of();

// map to promise and emit result
source$
  .pipe(mergeMap(getPromise))
  // output: 'This is a promise'
  .subscribe(console.log);
```

mergeMap

Example

```
import { interval } from 'rxjs';
import { ajax } from 'rxjs/ajax';
import { map, mergeMap } from 'rxjs/operators';

const API_URL = 'https://jsonplaceholder.typicode.com/todos';
const getToDo = (id: number) => ajax.getJSON(`${API_URL}/${id}`);
const incrementByOne = (value: number) => value + 1;

interval(1000)
  .pipe(map(incrementByOne), mergeMap(getToDo))
  // output: { userId: 1, id: 1, ...}
  .subscribe(console.log);
```

mergeMap Cadveat

Example

```
import { of } from 'rxjs';
import { concatMap, delay, mergeMap } from 'rxjs/operators';

//emit delay value
const delayBy$ = of(2000, 1000);

delayBy$
  .pipe(
    mergeMap((delayBy) => of(`Delayed by: ${delayBy}ms`).pipe(delay(delayBy)))
    //output: With concatMap: Delayed by: 1000ms, With concatMap: Delayed by: 2000ms
  )
  .subscribe(console.log);
);
```

[Demo](#)

concatMap

Example

```
import { of } from 'rxjs';
import { concatMap, delay } from 'rxjs/operators';

//emit delay value
const delayBy$ = of(2000, 1000);

delayBy$
  .pipe(
    concatMap((delayBy) => of(`Delayed by: ${delayBy}ms`).pipe(delay(delayBy)))
    //output: With concatMap: Delayed by: 2000ms, With concatMap: Delayed by: 1000ms
  )
  .subscribe(console.log);
```

[Demo](#)

switchMap

Example

```
import { fromEvent } from 'rxjs';
import { ajax } from 'rxjs/ajax';
import { map, switchMap, delay } from 'rxjs/operators';

// grab button reference
const button = document.querySelector('button');

const API_URL = 'https://jsonplaceholder.typicode.com/todos';
const getToDo = (id: number) =>
  ajax.getJSON(`${API_URL}/${id}`).pipe(delay(id * 10));
const getRandomToDoId = () => Math.ceil(Math.random() * 200);

fromEvent(button, 'click')
  .pipe(map(getRandomToDoId), switchMap(getToDo))
  .subscribe(console.log);
```

[Demo](#)

scan

```
import { of } from 'rxjs';
import { scan } from 'rxjs/operators';

const source$ = of(1, 2, 3);
// basic scan example, sum over time
const sum$ = source$.pipe(scan((acc, curr) => acc + curr));

// output: 1,3,6
sum.subscribe(console.log);
```


