

Reactive Programming dependencies :

```
14 > dependencies {
15     implementation("io.projectreactor:reactor-core:3.4.0")
16
17     //log - logback
18     implementation("ch.qos.logback:logback-classic:1.2.3")
19
20     //reactor-debug-agent
21     implementation('io.projectreactor:reactor-tools:3.4.0')
22
23     //testing
24     testImplementation("io.projectreactor:reactor-test:3.4.0")
25     testImplementation('org.junit.jupiter:junit-jupiter:5.5.1')
26     testImplementation("org.mockito:mockito-core:3.2.4")
27     testImplementation("org.mockito:mockito-junit-jupiter:3.2.4")
28
29 }
```

The reactive library here is reactive-core. Spring webflux uses behind the screen for us. reactor-test to test non-blocking code.

```
7 > public class FluxAndMonoGeneratorService {
8
9     public Flux<String> namesFlux() {
10
11         return Flux.fromIterable(List.of("alex", "ben", "chloe")); // db or a remote service call
12     }
13
14
15 >     public static void main(String[] args) {
16
17         FluxAndMonoGeneratorService fluxAndMonoGeneratorService = new FluxAndMonoGeneratorService();
18
19         fluxAndMonoGeneratorService.namesFlux()
20             .subscribe(name -> {
21                 System.out.println("Name is : " + name);
22             });
23
24     }
25 }
26
```

```
public Mono<String> nameMono(){
    return Mono.just("alex");
}
```

```

    fluxAndMonoGeneratorService.nameMono()
        .subscribe(name->{
            System.out.println("Mono name is : " + name);
        });
    }
}

```

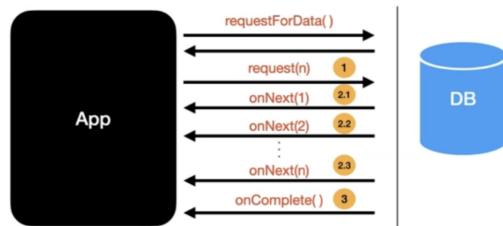
Until subscribe nothing will happen

```

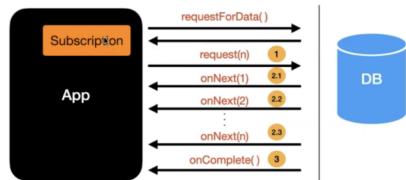
-----> Task :FluxAndMonoGeneratorService.main()
04:33:46.755 [main] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
Name is : alex
Name is : ben
Name is : chloe
Mono name is : alex!

```

Reactive Streams Events



Reactive Streams Events



For the second call (onSubscribe) in the above diagram it will create a subscription object in the subscriber.

```

public Flux<String> namesFlux() {

    return Flux.fromIterable(List.of("alex", "ben", "chloe"))
        .log(); // db or a remote service call
}

public Mono<String> nameMono(){

    return Mono.just("alex");
}

```

Add log method for both namesFlux and nameMono to see data flow between subscriber and publisher

```

reactive-732 ms 05:08:41.373 [main] INFO reactor.FluxIterable.1 - | onNext(alex)
Name is : alex
05:08:41.386 [main] INFO reactor.FluxIterable.1 - | onNext(ben)
Name is : ben
05:08:41.386 [main] INFO reactor.FluxIterable.1 - | onNext(chloe)
Name is : chloe
05:08:41.387 [main] INFO reactor.FluxIterable.1 - | onComplete()
05:08:41.422 [main] INFO reactor.MonoJust.2 - | onSubscribe([Synchronous Fuseable] Operators.ScalarSubscription)
05:08:41.422 [main] INFO reactor.MonoJust.2 - | request(unbounded)
05:08:41.422 [main] INFO reactor.MonoJust.2 - | onNext(alex)
Mono name is : alex
05:08:41.422 [main] INFO reactor.MonoJust.2 - | onComplete()

```

Flow of data for flux and mono

```

public class FluxAndMonoGeneratorServiceTest {

    FluxAndMonoGeneratorService fluxAndMonoGeneratorService =
        new FluxAndMonoGeneratorService();

    @Test
    void namesFlux() {
        //given
        //when
        var namesFlux = fluxAndMonoGeneratorService.namesFlux();

        //then
        StepVerifier.create(namesFlux)
            .expectNext("alex", "ben", "chloe")
            .verifyComplete();
    }
}

port reactor.core.publisher.Flux;
port reactor.core.publisher.Mono;
port java.util.List;

public class FluxAndMonoGeneratorService {

    public Flux<String> namesFlux() {
        return Flux.fromIterable(List.of("alex", "ben", "chloe"))
            .log(); // db or a remote service call
    }

    public Mono<String> nameMono(){
        return Mono.just("alex");
    }
}

public static void main(String[] args) {
    FluxAndMonoGeneratorService fluxAndMonoGeneratorService = ne
}

```

Use StepVerifier from reactive test to test non blocking code.

```
//then
StepVerifier.create(namesFlux)
    //expectNext("alex", "ben", "chloe")
    .expectNextCount(3)
    .verifyComplete();
}
```

```
//then
StepVerifier.create(namesFlux) StepVerifier.FirstStep<String>
    //expectNext("alex", "ben", "chloe")
    //expectNextCount(3)
    .expectNext("alex") StepVerifier.Step<String>
    .expectNextCount(2)
    .verifyComplete();
```

Combination of both expectNext and expectNextCount.

The create function will take care of subscribe method on publisher

Transforming Data Using Operators in Project Reactor

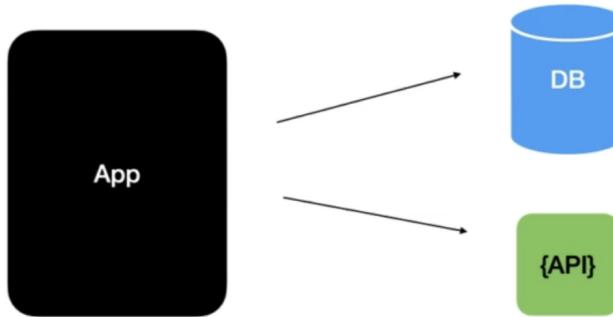
Why Transform Data ?

- It is pretty common for apps to transform data from its original form

`Flux.just("alex", "ben", "chloe")`  `Flux.just("ALEX", "BEN", "CHLOE")`

`Flux.just("alex", "ben", "chloe")`  `Flux.just("ALEX", "CHLOE")`

These are simple data transformations but in reality it is different when we subscribe data from DB and other API



map() Operator

- Used to transform the element from one form to another in a Reactive Stream
- Similar to the map() operator in Streams API

```
20
21     public Flux<String> namesFlux_map() {
22
23         return Flux.fromIterable(List.of("alex", "ben", "chloe"))
24             .map(String::toUpperCase)
25             // .map(s -> s.toUpperCase())
26             .log(); // db or a remote service call
27
28 }
```

Line number 24 and 25

```
27 @Test
28 void namesFlux_map() {
29     //given
30
31     //when
32     var namesFlux = fluxAndMonoGeneratorService.namesFlux_map();
33
34     //then
35     StepVerifier.create(namesFlux)
36         .expectNext("ALEX", "BEN", "CHLOE")
37         .verifyComplete();
38 }
```

```
Results 189 ms
1.learnreacti 189 ms
namesFlux_n 189 ms
04:39:43.262 [Test worker] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
04:39:43.288 [Test worker] INFO reactor.Flux.MapFuseable.1 - | onSubscribe([Fuseable] FluxMapFuseable)
04:39:43.292 [Test worker] INFO reactor.Flux.MapFuseable.1 - | request(unbounded)
04:39:43.292 [Test worker] INFO reactor.Flux.MapFuseable.1 - | onNext(ALEX)
04:39:43.292 [Test worker] INFO reactor.Flux.MapFuseable.1 - | onNext(BEN)
04:39:43.293 [Test worker] INFO reactor.Flux.MapFuseable.1 - | onNext(CHLOE)
04:39:43.293 [Test worker] INFO reactor.Flux.MapFuseable.1 - | onComplete()
```

Reactive Streams are Immutable

```
public Flux<String> namesFlux_immutability() {
    var namesFlux = Flux.fromIterable(List.of("alex", "ben", "chloe"));
    namesFlux.map(String::toUpperCase);
    return namesFlux;
}
```

Reactive streams are immutable and they will not change the value of original data.

```
39
40
41 @Test
42 void namesFlux_immutability() {
43     //given
44     //when
45     var namesFlux = fluxAndMonoGeneratorService.namesFlux_immutability();
46
47     //then
48     StepVerifier.create(namesFlux)
49         .expectNext("ALEX", "BEN", "CHLOE")
50         .verifyComplete();
51 }
52 }
```

```

48     Test
49     void namesFlux_immutability() {
50         //given
51
52         //when
53         var namesFlux = fluxAndMonoGeneratorService.namesFlux_immutability();
54
55         //then
56         StepVerifier.create(namesFlux)
57             .expectNext("ALEX", "BEN", "CHLOE")
58             .verifyComplete();
59
60     }
61
62     public Flux<String> namesFlux_map() {
63
64         return Flux.fromIterable(List.of("alex", "ben", "chloe"))
65             .map(String::toUpperCase)
66             .map(s -> s.toUpperCase())
67             .log(); // db or a remote se
68
69     }
70
71     public Flux<String> namesFlux_immutability() {
72
73         var namesFlux = Flux.fromIterable(List.of("alex", "ben", "chloe"));
74
75         namesFlux.map(String::toUpperCase);
76
77     }
78
79     public Mono<String> namesFlux_immutability() {
80
81         return Mono.just("alex");
82     }
83
84 }

```

Run: FluxAndMonoGeneratorServiceTest.namesFlux_immutability
Tests failed: 1 of 1 test - 176 ms
com.learnreactive.TestResults 176 ms
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test FAILED
05:00:05.294 [Test worker] DEBUG reactor.util.Loggers\$LoggerFactory - Using Slf4j logging framework

expectation "expectNext(ALEX)" failed (expected value: ALEX; actual value: alex)
java.lang.AssertionError: expectation "expectNext(ALEX)" failed (expected value: ALEX; actual value: alex)
at reactor.test.MessageFormatter.assertError(MessageFormatter.java:115)

Test failed because of immutability. This will pass if we change expected data to lower case.

```

//then
StepVerifier.create(namesFlux)
    .expectNext("alex", "ben", "chloe")
    .verifyComplete();
}

```

filter() Operator

- Used to filter elements in a Reactive Stream
- Similar to the filter() operator in Streams API

```

@Test
void namesFlux_map() {
    //given
    int stringLength = 3;
    //when
    var namesFlux = fluxAndMonoGeneratorService.namesFlux_map(stringLength);
    //then
    StepVerifier.create(namesFlux)
        .expectNext("ALEX", "BEN", "CHLOE")
        .expectNext("ALEX", "CHLOE")
        .verifyComplete();
}

```

```

public Flux<String> namesFlux_map(int stringLength) {
    //filter the string whose length is greater than 3
    return Flux.fromIterable(List.of("alex", "ben", "chloe"))
        .map(String::toUpperCase)
        //.map(s -> s.toUpperCase())
        .filter(s->s.length() > stringLength)
        .log(); // db or a remote service call
}

public Flux<String> namesFlux_immutability() {
    var namesFlux = Flux.fromIterable(List.of("alex", "ben", "chloe"));
    namesFlux.map(String::toUpperCase);
}

```

Right side actual code and left side test case.

Some changes to filtered values



```
36 //then
37 StepVerifier.create(namesFlux)
38     //.expectNext("ALEX", "BEN", "CHLOE")
39     .expectNext("4-ALEX", "5-CHLOE")
40     .verifyComplete();
41 }
42
43 @Test
44 void namesFlux_immutability() {
45     //given
46
47     //when
```

```
10  * @Test
11  void namesFlux_immutability() {
12
13      Flux<String> namesFlux_map(int stringLength) {
14          //filter the string whose length is greater than 3
15          return Flux.fromIterable(List.of("alex", "ben", "chloe"))
16              .map(String::toUpperCase)
17              //.map(s -> s.toUpperCase())
18              .filter(s->s.length() > stringLength)
19              .map(s-> s.length() + "-" + s)//4-ALEX, 5-CHLOE
20              .log(); // db or a remote service call
21
22      }
23
24      // ALEX ->A, L, E, X, C, H, L , O, E
25      .flatMap(s-> splitString(s)) // A,L,E,X,C,H,L,O,E
26      .log(); // db or a remote service call
27
28  }
```

Left is test case and right is actual code

flatMap()

- Transforms one source element to a Flux of 1 to N elements
"ALEX" -> Flux.just("A", "L", "E", "X") 
- Use it when the transformation returns a Reactive Type (Flux or Mono)
- Returns a Flux<Type>

```
public Flux<String> namesFlux_flatmap(int stringLength) {
    //filter the string whose length is greater than 3
    return Flux.fromIterable(List.of("alex", "ben", "chloe"))
        .map(String::toUpperCase)
        //.map(s -> s.toUpperCase())
        .filter(s->s.length() > stringLength)
        // ALEX,CHLOE ->A, L, E, X, C, H, L , O, E
        .flatMap(s-> splitString(s)) // A,L,E,X,C,H,L,O,E
        .log(); // db or a remote service call
}

//ALEX -> Flux(A,L,E,X)
public Flux<String> splitString(String name){
    var charArray = name.split( regex: "" );
    return Flux.fromArray(charArray);
}
```

```

61     //when
62     var namesFlux = fluxAndMonoGeneratorService.namesFlux.flatMap(s->
63         .log(); // db
64     }
65     //then
66     StepVerifier.create(namesFlux)
67         .expectNext("A","L","E","X","C","H","L","O","E")
68         .verifyComplete();
69     }
70 }

public Flux<String> splitString(String name){
    var charArray = name.split(regex);
    var delay = new Random().nextInt(bound: 1000);
    return Flux.fromArray(charArray)
        .delayElements(Duration.ofMillis(delay));
}

public static void main(St

```

Run: FluxAndMonoGeneratorServiceTest.names... Tests passed: 1 of 1 test - 200 ms

Test Results 200 ms

- com.learnreactive 200 ms
 - namesFlux_f 200 ms

```

09:10:41.769 [Test worker] INFO reactor.Flux.FlatMap$1 - onSubscribe(FluxFlatMap.FlatMapMain)
09:10:41.773 [Test worker] INFO reactor.Flux.FlatMap$1 - request(unbounded)
09:10:41.774 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(A)
09:10:41.774 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(L)
09:10:41.774 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(E)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(X)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(C)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(H)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(L)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(O)
09:10:41.775 [Test worker] INFO reactor.Flux.FlatMap$1 - onNext(E)
09:10:41.776 [Test worker] INFO reactor.Flux.FlatMap$1 - onComplete()

```

Testcase for flatmap.

Asynchronous operations using flatmap.

```

void namesFlux_flatmap_async() {
    //given
    int stringLength = 3;

    //when
    var namesFlux = fluxAndMonoGeneratorService.namesFlux_flatmap_async();

    //then
    StepVerifier.create(namesFlux)
        .expectNext("A","L","E","X","C","H","L","O","E")
        .expectNextCount(9)
        .verifyComplete();
}
}

public Flux<String> namesFlux_flatmap_async(int stringLength) {
    //filter the string whose length is greater than 3
    return Flux.fromIterable(List.of("alex", "ben", "chloe"))
        .map(string::toUpperCase)
        .map(s->s.toUpperCase())
        .filter(s->s.length() > stringLength)
        // ALEX,CHLOE ->A, L, E, X, C, H, L, O, E
        .flatMap(s-> splitString_withDelay(s)) // A,L,E,X,C,H,L,O,E
        .log(); // db or a remote service call
}

public Flux<String> splitString_withDelay(String name){
    var charArray = name.split(regex);
    var delay = new Random().nextInt(bound: 1000);
    return Flux.fromArray(charArray)
        .delayElements(Duration.ofMillis(delay));
}


```

Test Results 1s 399ms

- com.learnreactive 1s 399ms
 - namesFlux_flatmap_async 1s 399ms

```

15:09:14.097 [Test worker] INFO reactor.Flux.FlatMap$1 - request(unbounded)
15:09:14.182 [parallel-2] INFO reactor.Flux.FlatMap$1 - onNext(A)
15:09:14.238 [parallel-3] INFO reactor.Flux.FlatMap$1 - onNext(L)
15:09:14.272 [parallel-4] INFO reactor.Flux.FlatMap$1 - onNext(E)
15:09:14.315 [parallel-5] INFO reactor.Flux.FlatMap$1 - onNext(X)
15:09:14.368 [parallel-6] INFO reactor.Flux.FlatMap$1 - onNext(C)
15:09:14.414 [parallel-1] INFO reactor.Flux.FlatMap$1 - onNext(H)
15:09:14.491 [parallel-7] INFO reactor.Flux.FlatMap$1 - onNext(L)
15:09:14.496 [parallel-8] INFO reactor.Flux.FlatMap$1 - onNext(O)
15:09:15.244 [parallel-9] INFO reactor.Flux.FlatMap$1 - onNext(E)
15:09:15.248 [parallel-9] INFO reactor.Flux.FlatMap$1 - onComplete()

```

Here order is not important, the outcome should match. See first chloe and followed by alex. Flatmap can't be used incase of order is important.

map()

- One to One Transformation
- Does the simple transformation from **T** to **V**
- Used for simple synchronous transformations
- Does not support transformations that returns Publisher

flatmap()

- One to N Transformations
- Does more than just transformation. Subscribes to Flux or Mono that's part of the transformation and then flattens it and sends it downstream
- Used for asynchronous transformations
- Use it with transformations that returns Publisher

Last point publisher means flux or mono

concatMap()

- Works similar to **flatMap()**
- Only difference is that **concatMap()** preserves the ordering sequence of the Reactive Streams.

Order sequencing maintained here unlink flatmap during asynchronous call.

The screenshot shows a Java code editor with two parts: a test case and its implementation.

Test Case:

```
@Test
void namesFlux_concatmap() {
    //given
    int stringLength = 3;

    //when
    var namesFlux = fluxAndMonoGeneratorService.namesFlux_conc

    //then
    StepVerifier.create(namesFlux)
        .expectNext("A", "L", "E", "X", "C", "H", "L", "O", "E")
        .expectNextCount(9)
        .verifyComplete();
}
```

Implementation:

```
59 } .log(); // db or a remote service call
60 }
61 }
62 public Flux<String> namesFlux_concatmap(int stringLength) {
63     //filter the string whose length is greater than 3
64     return Flux.fromIterable(List.of("alex", "ben", "chloe"))
65         .map(String::toUpperCase)
66         // .map(s -> s.toUpperCase())
67         .filter(s->s.length()> stringLength)
68         // ALEX,CHLOE -> A, L, E, X, C, H, L , O, E
69         .concatMap(s-> splitString_withDelay(s)) // A,L,E,X,C,H,
70         .log(); // db or a remote service call
71 }
72
73 //ALEX -> Flux(A,L,E,X)
74 public Flux<String> splitString(String name){
75     var charArray = name.split(regex: "");
76     return Flux.fromArray(charArray);
}
```

```

Tests passed: 1 of 1 test - 3 s 504 ms
04:42:05.621 [Test worker] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
04:42:05.653 [Test worker] INFO reactor.Flux.ConcatMap.1 - onSubscribe(FluxConcatMap.ConcatMapImmediate)
04:42:05.656 [Test worker] INFO reactor.Flux.ConcatMap.1 - onSubscribe(Unbounded)
04:42:05.849 [parallel-1] INFO reactor.Flux.ConcatMap.1 - onNext(A)
04:42:06.085 [parallel-2] INFO reactor.Flux.ConcatMap.1 - onNext(L)
04:42:06.159 [parallel-3] INFO reactor.Flux.ConcatMap.1 - onNext(E)
04:42:06.313 [parallel-4] INFO reactor.Flux.ConcatMap.1 - onNext(X)
04:42:06.845 [parallel-5] INFO reactor.Flux.ConcatMap.1 - onNext(C)
04:42:07.378 [parallel-6] INFO reactor.Flux.ConcatMap.1 - onNext(H)
04:42:07.911 [parallel-7] INFO reactor.Flux.ConcatMap.1 - onNext(I)
04:42:08.445 [parallel-8] INFO reactor.Flux.ConcatMap.1 - onNext(O)
04:42:08.975 [parallel-9] INFO reactor.Flux.ConcatMap.1 - onNext(E)
04:42:08.978 [parallel-9] INFO reactor.Flux.ConcatMap.1 - onComplete()

```

Order issue fixed

Flatmap is faster where concatmap is slower

flatMap in Mono

- Use it when the transformation returns a Mono

```

private Mono<List<String>> splitStringMono(String s) {
    var charArray = s.split("");
    return Mono.just(List.of(charArray))
        .delayElement(Duration.ofSeconds(1));
}

```

- Returns a Mono<T>
- Use flatMap if the transformation involves making a REST API call or any kind of functionality that can be done asynchronously

The above slide and below example are very important

```

public Mono<List<String>> namesMono_flatMap(int stirngLength) {
    return Mono.just("alex")
        .map(String::toUpperCase)
        .filter(s-> s.length() > stirngLength)
        .flatMap(this::splitStringMono); //Mono<List of A, L, E X>
}

private Mono<List<String>> splitStringMono(String s) {
    var charArray = s.split("");
    var charList = List.of(charArray); //ALEX -> A, L, E, X
    return Mono.just(charList);
}

```

Command + shift + t will create a test case

```

    @Test
    void namesMono_flatMap() {
        //given
        int stringLength = 3;

        //when
        var value = fluxAndMonoGeneratorService.namesMono_flatMap(stringLength);

        //then
        StepVerifier.create(value)
            .expectNext(List.of("A", "L", "E", "X"))
            .verifyComplete();
    }
}

44     .filter(s-> s.length() > stirngLength);
45
46     }
47
48     public Mono<List<String>> namesMono_flatMap(int stirngLength) {
49         return Mono.just("alex")
50             .map(String::toUpperCase)
51             .filter(s-> s.length() > stirngLength)
52             .flatMap(this::splitStringMono)
53             .log(); //Mono<List of A, L, E X>
54
55     }
56
57     private Mono<List<String>> splitStringMono(String s) {
58         var charArray = s.split(regex: "");
59         var charList = List.of(charArray); //ALEX -> A, L, E, X
60         return Mono.just(charList);
61     }

```

flatMapMany() in Mono

- Works very similar to flatMap()

```

private Flux<String> splitString_withDelay(String name) {
    var delay = new Random().nextInt(1000);
    var charArray = name.split("");
    return Flux.fromArray(charArray)
        .delayElements(Duration.ofMillis(delay));
}

```

FlatmapMany will help to return flux when we apply on mono

```

    //when
    var value = fluxAndMonoGeneratorService.namesMono_flatMapMany(stringLength);

    //then
    StepVerifier.create(value)
        .expectNext("A", "L", "E", "X")
        .verifyComplete();
}

53     .log(); //Mono<List of A, L, E X>
54
55     }
56
57     public Flux<String> namesMono_flatMapMany(int stirngLength) {
58         return Mono.just("alex")
59             .map(String::toUpperCase)
60             .filter(s-> s.length() > stirngLength)
61             .flatMapMany(this::splitString)
62             .log(); //Mono<List of A, L, E X>
63
64     }

```

Run: FluxAndMonoGeneratorServiceTest.names...
 Tests passed: 1 of 1 test – 216 ms

```

> Task :test
15:51:39.560 [Test worker] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
15:51:39.660 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onSubscribe(MonoFlatMapMany.FlatMapManyMain)
15:51:39.663 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - request(unbounded)
15:51:39.664 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onNext(A)
15:51:39.664 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onNext(L)
15:51:39.664 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onNext(E)
15:51:39.665 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onNext(X)
15:51:39.665 [Test worker] INFO reactor.Flux.MonoFlatMapMany.1 - onComplete()
BUILD SUCCESSFUL in 1s
4 actionable tasks: 3 executed, 1 up-to-date
3:51:39 PM: Task execution finished ':test --tests "com.learnreactiveprogramming.service.FluxAndMonoGeneratorServiceTest.namesMono_flatM'

```

transform()

- Used to transform from one type to another
- Accepts **Function Functional Interface**
 - Function Functional Interface got released as part of Java 8
 - Input - Publisher (Flux or Mono)
 - Output - Publisher (Flux or Mono)

```
public Flux<String> namesFlux_transform(int stringLength) {
    //filter the string whose length is greater than 3

    Function<Flux<String>,Flux<String>> filterMap = name -> name.map(String::toUpperCase)
        .filter(s->s.length() > stringLength);

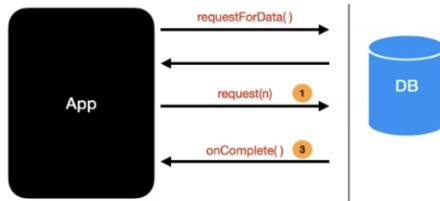
    return Flux.fromIterable(List.of("alex", "ben", "chloe"))
        .transform(filterMap)
        .flatMap(s-> splitString(s)) // A,L,E,X,C,H,L,O,E
        .log(); // db or a remote service call
    }
```

```
99
100
101 > d namesFlux_transform() {
102     //given
103     int stringLength = 3;
104
105     //when
106     var namesFlux = fluxAndMonoGeneratorService.namesFlux_transform(st
107
108     //then
109     StepVerifier.create(namesFlux)
110         .expectNext("A", "L", "E", "X", "C", "H", "L", "O", "E")
111         .verifyComplete();
112
113 }
```

defaultIfEmpty() & switchIfEmpty()

defaultIfEmpty() & switchIfEmpty()

- Its not mandatory for a data source to emit data all the time



- We can use the defaultIfEmpty() or switchIfEmpty() operator to provide default values

The screenshot shows an IDE interface with two main sections: code editor and test results.

Code Editor:

```
//when
var namesFlux = fluxAndMonoGeneratorService.namesFlux_transform(st
//then
StepVerifier.create(namesFlux)
    .expectNext("A", "L", "E", "X", "C", "H", "L", "O", "E")
    .verifyComplete();
}

//filter the string whose length is greater than 3
return Flux.fromIterable(List.of("alex", "ben", "chloe"))
    .map(String::toUpperCase)
    .map(s -> s.toUpperCase())
    .filter(s->s.length() > stringLength)
    // ALEX,CHLOE ->A, L, E, X, C, H, L , O, E
    .concatMap(s-> splitString_withDelay(s)) // A,L,E,X,C,H,L
    .log(); // db or a remote service call

public Flux<String> namesFlux_transform(int stringLength) {
    //filter the string whose length is greater than 3

    Function<Flux<String>,Flux<String>> filterMap = name -> name.map(
        .filter(s->s.length() > stringLength);
```

Test Results:

Tests failed: 1 of 1 test – 178 ms

com.learnreactive 178 ms

34:57:12.336 [Test worker] DEBUG reactor.util.LoggerFactory - Using Slf4j logging framework
34:57:12.365 [Test worker] INFO reactor.Flux.FlatMap.1 - onSubscribe(FluxFlatMap.FlatMapMain)
34:57:12.368 [Test worker] INFO reactor.Flux.FlatMap.1 - request(unbounded)
34:57:12.368 [Test worker] INFO reactor.Flux.FlatMap.1 - onComplete()

expectation "expectNext(A)" failed (expected: onNext(A); actual: onComplete())

Just pass a filter string with string length 6 and we will get an empty response. See the error message (last line)

```

120
121
122
123
124
125
126
127
128
129

    var namesFlux = fluxAndMonoGeneratorService.namesFlux_transforms(s)
    //then
    StepVerifier.create(namesFlux)
        //.expectNext("A","L","E","X","C","H","L","O","E")
        .expectNext("default")
        .verifyComplete();
    }

}

Run: FluxAndMonoGeneratorServiceTest.names...
Test Results 179 ms > Task :compileJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
04:58:55.788 [Test worker] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
04:58:55.816 [Test worker] INFO reactor.Flux.DefaultIfEmpty.1 - onSubscribe([Fuseable] FluxDefaultIfEmpty.DefaultIfEmptySubscriber)
04:58:55.819 [Test worker] INFO reactor.Flux.DefaultIfEmpty.1 - request(unbounded)
04:58:55.819 [Test worker] INFO reactor.Flux.DefaultIfEmpty.1 - onNext(default)
04:58:55.820 [Test worker] INFO reactor.Flux.DefaultIfEmpty.1 - onComplete()
BUILD SUCCESSFUL in 1s
4 actionable tasks: 3 executed, 1 up-to-date
4:58:55 AM: Task execution finished ':test --tests "com.learnreactiveprogramming.service.FluxAndMonoGeneratorServiceTest.namesFlux_trans...
```

Notice line number 84

```

130
131
132
133
134
135
136
137
138
139
140
141
142
143
144

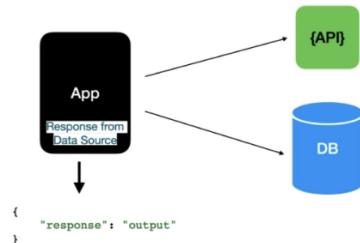
    void namesFlux_transform_ifEmpty() {
        //given
        int stringLength = 6;
        //when
        var namesFlux = fluxAndMonoGeneratorService.namesFlux_transform_si
        //then
        StepVerifier.create(namesFlux)
            //.expectNext("A","L","E","X","C","H","L","O","E")
            .expectNext("D","E","F","A","U","L","T")
            .verifyComplete();
    }
}

Run: FluxAndMonoGeneratorServiceTest.names...
Test Results 180 ms > Task :compileJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
05:02:38.009 [TEST WORKER] DEBUG reactor.util.Loggers$LoggerFactory - USING SLF4J logging framework
05:02:38.012 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onSubscribe(FluxSwitchIfEmpty.SwitchIfEmptySubscriber)
05:02:38.012 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - request(unbounded)
05:02:38.017 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(D)
05:02:38.017 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(E)
05:02:38.017 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(F)
05:02:38.018 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(A)
05:02:38.018 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(U)
05:02:38.018 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(L)
05:02:38.018 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onNext(T)
05:02:38.018 [Test worker] INFO reactor.Flux.SwitchIfEmpty.1 - onComplete()
```

switchIfEmpty will produce publisher
defaultIfEmpty will produce the default type of its source type.

Combining Flux & Mono

Why combining Flux and Mono ?

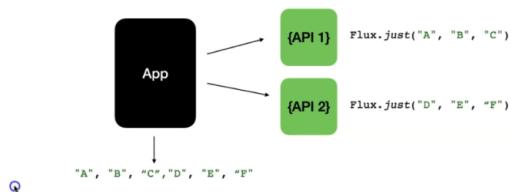


Most of the micro services work in this fashion only.

concat()
&
concatWith()

concat() & concatWith()

- Used to combine two reactive streams in to one



concat() & concatWith()

- Used to combine two reactive streams in to one
- Concatenation of Reactive Streams happens in a sequence
 - First one is subscribed first and completes
 - Second one is subscribed after that and then completes
- concat()** - static method in Flux
- concatWith()** - instance method in Flux and Mono
- Both of these operators works similarly

```
37     //then
38     StepVerifier.create(namesFlux)
39         //expectNext("A","L","E","X","C","H","L","O","E")
40         .expectNext("D","E","F","A","U","L","T")
41         .verifyComplete();
42     }
43
44     @Test
45     void explore_concat() {
46         //given
47
48         //when
49         var concatFlux = fluxAndMonoGeneratorService.explore_concat();
50
51         //then
52         StepVerifier.create(concatFlux)
53             .expectNext("A","B","C","D","E","F")
54             .verifyComplete();
55     }
56 }
```

```
183     }
184
185     public Flux<String> explore_concat(){
186
187         var abcFlux = Flux.just("A","B","C");
188
189         var defFlux = Flux.just("D","E","F");
190
191         return Flux.concat(abcFlux,defFlux);
192     }
193
194     //ALEX -> Flux(A,L,E,X)
195     public Flux<String> splitString(String name){
196         var charArray = name.split( regex: "" );
197         return Flux.fromArray(charArray);
198     }
199
200 }
```

```
114
115     public Flux<String> explore_concatwith(){
116
117         var abcFlux = Flux.just("A","B","C");
118
119         var defFlux = Flux.just("D","E","F");
120
121         return abcFlux.concatWith(defFlux).log();
122     }
123
124
125     public Flux<String> explore_concatwith_mono(){
126
127         var aMono = Mono.just("A");
128         var bMono = Mono.just("B");
129
130         return aMono.concatWith(bMono).log(); // A, B
131
132     }
133 }
```

Line number 115 to 130 concatwith examples

merge() and mergeWith() are used to combine two publishers in to one.

merge()

- merge() operator takes in two arguments

```
// "A", "D", "B", "E", "C", "F" ←
// Flux is subscribed early
public Flux<String> explore_merge() {

    var abcFlux = Flux.just("A", "B", "C")
        .delayElements(Duration.ofMillis(100)) 1
    ;
    var defFlux = Flux.just("D", "E", "F")
        .delayElements(Duration.ofMillis(125)) 2
    ;
    return Flux.merge(abcFlux,defFlux).log(); 3

}
```

Merge will happen interleaved fashion

merge() & mergeWith()

- Both the publishers are subscribed at the same time
 - Publishers are subscribed eagerly and the merge happens in an interleaved fashion
 - concat() subscribes to the Publishers in a sequence
- **merge()** - static method in Flux
- **mergeWith()** - instance method in Flux and Mono
- Both of these operators work similarly

```

reactive-programming-using-reactor src main java com learnreactiveprogramming service FluxAndMonoGeneratorService explore_merge
162     //when
163     var value = fluxAndMonoGeneratorService.explore_merge();
164
165     //then
166     StepVerifier.create(value)
167         .expectNext("A", "D", "B", "E", "C", "F")
168         .verifyComplete();
169
170 }
171
133     }
134
135     public Flux<String> explore_merge(){
136
137         var abcFlux = Flux.just("A", "B", "C")
138             .delayElements(Duration.ofMillis(100)); //A,B
139
140         var defFlux = Flux.just("D", "E", "F")
141             .delayElements(Duration.ofMillis(125)); //D,E
142
143         return Flux.merge(abcFlux, defFlux).log();
144
145     }
146

```

Test Results: 601ms
com.learnreactiveprogramming.FluxAndMonoGeneratorServiceTest.explore...
> Task :test
05:07:33.202 [Test worker] DEBUG reactor.util.Loggers\$LoggerFactory - Using Slf4j logging framework
05:07:33.235 [Test worker] INFO reactor.Flux.Merge.1 - onSubscribe(FluxFlatMap.FlatMapMain)
05:07:33.238 [Test worker] INFO reactor.Flux.Merge.1 - request(unbounded)
05:07:33.376 [parallel-1] INFO reactor.Flux.Merge.1 - onNext(A)
05:07:33.399 [parallel-2] INFO reactor.Flux.Merge.1 - onNext(D)
05:07:33.477 [parallel-3] INFO reactor.Flux.Merge.1 - onNext(B)
05:07:33.525 [parallel-4] INFO reactor.Flux.Merge.1 - onNext(E)
05:07:33.579 [parallel-5] INFO reactor.Flux.Merge.1 - onNext(C)
05:07:33.654 [parallel-6] INFO reactor.Flux.Merge.1 - onNext(F)
05:07:33.655 [parallel-6] INFO reactor.Flux.Merge.1 - onComplete()

Merge will happen at 100 , 125, 200, 250 sequence

If we change the value we see vice versa value

```

reactive-programming-using-reactor src main java com learnreactiveprogramming service FluxAndMonoGeneratorService explore_merge
162     //when
163     var value = fluxAndMonoGeneratorService.explore_merge();
164
165     //then
166     StepVerifier.create(value)
167         .expectNext("A", "D", "B", "E", "C", "F")
168         .verifyComplete();
169
170 }
171
133     }
134
135     public Flux<String> explore_merge(){
136
137         var abcFlux = Flux.just("A", "B", "C")
138             .delayElements(Duration.ofMillis(100)); //A,B
139
140         var defFlux = Flux.just("D", "E", "F")
141             .delayElements(Duration.ofMillis(90)); //D,E
142
143         return Flux.merge(abcFlux, defFlux).log();
144
145     }
146

```

Test Results: 277ms
com.learnreactiveprogramming.FluxAndMonoGeneratorServiceTest.explore...
> Task :test
15:07:51.014 [Test worker] INFO reactor.Flux.Merge.1 - request(unbounded)
15:07:51.139 [parallel-2] INFO reactor.Flux.Merge.1 - onNext(D)
15:07:51.143 [parallel-2] INFO reactor.Flux.Merge.1 - cancel()

expectation "expectNext(A)" failed (expected value: A; actual value: D)
java.lang.AssertionError: expectation "expectNext(A)" failed (expected value: A; actual value: D)
at reactor.test.MessageFormatter.assertError(MessageFormatter.java:115)
at reactor.test.MessageFormatter.failPrefix(MessageFormatter.java:104)
at reactor.test.MessageFormatter.fail(MessageFormatter.java:73)
at reactor.test.MessageFormatter.failOptional(MessageFormatter.java:88)

Def flux executed first because of less delay time. That is why test got failed

```
147     public Flux<String> explore_mergeWith(){  
148  
149         var abcFlux = Flux.just("A","B","C")  
150             .delayElements(Duration.ofMillis(100)); //A,B  
151  
152         var defFlux = Flux.just("D","E","F")  
153             .delayElements(Duration.ofMillis(125));//D,E  
154  
155         //return Flux.merge(abcFlux,defFlux).log();  
156         return abcFlux.mergeWith(defFlux).log();  
157     }
```

S (Ctrl+S for Win/Linux)

Event Log

mergeWith flux return example

```
160     public Flux<String> explore_mergeWith_mono(){  
161  
162         var aMono = Mono.just("A"); //A  
163  
164         var bMono = Mono.just("B"); //B  
165  
166         return aMono.mergeWith(bMono).log(); // A, B | I  
167  
168     }  
169 
```

Merge with mono return example

mergeSequential()

- Used to combine two Publishers (Flux) in to one
- Static method in Flux
- Both the publishers are subscribed at the same time
 - Publishers are subscribed eagerly
 - Even though the publishers are subscribed eagerly the merge happens in a sequence

```

168     .verifyComplete();
169 }
170
171 @Test
172 void explore_mergeSequential() {
173     //given
174
175     //when
176     var value = fluxAndMonoGeneratorService.explore_merg
177
178     //then
179     StepVerifier.create(value)
180         .expectNext("A", "B", "C", "D", "E", "F")
181         .verifyComplete();
182 }
183
166
167
168 }
169
170 public Flux<String> explore_mergeSequential(){
171
172     var abcFlux = Flux.just("A", "B", "C")
173         .delayElements(Duration.ofMillis(100)); //A,B
174
175     var defFlux = Flux.just("D", "E", "F")
176         .delayElements(Duration.ofMillis(125)); //D,E
177
178     return Flux.mergeSequential(abcFlux, defFlux).log();
179
180
181
182
183
//MIX -> Flux(A, B, C, D, E, F)

```

zip()

- Zips two publishers together in this example

```

// AD, BE, CF
public Flux<String> explore_zip() {
    var abcFlux = Flux.just("A", "B", "C");
    var defFlux = Flux.just("D", "E", "F");
    return Flux.zip(abcFlux, defFlux, (first, second) -> first + second );
}

```

The diagram illustrates the `zip` operation. It shows two separate streams: `abcFlux` and `defFlux`. Stream `abcFlux` contains elements "A", "B", and "C". Stream `defFlux` contains elements "D", "E", and "F". The `zip` operation combines these streams into a single stream where each element is the concatenation of corresponding elements from both streams. Therefore, the resulting stream contains the elements "AD", "BE", and "CF".

Third parameter is combinator lambda.

zip() & zipWith()

- zip()
 - Static method that's part of the Flux
 - Can be used to merge up-to 2 to 8 Publishers (Flux or Mono) in to one
- zipWith()
 - This is an instance method that's part of the Flux and Mono
 - Used to merge two Publishers in to one
- Publishers are subscribed eagerly
- Waits for all the Publishers involved in the transformation to emit one element
 - Continues until one publisher sends an OnComplete event

zip()

- Zips four publishers together in this example

```
// AD14, BE25, CF36
public Flux<String> explore_zip_1() {
    var abcFlux = Flux.just("A", "B", "C");
    var defFlux = Flux.just("D", "E", "F");
    var flux3 = Flux.just("1", "2", "3");
    var flux4 = Flux.just("4", "5", "6");

    return Flux.zip(abcFlux, defFlux, flux3, flux4)
        .map(t4 -> t4.getT1()+t4.getT2()+t4.getT3()+t4.getT4());
}
```

AD14 BE25 CF36

Tuple is returned before the map.

```
32
33     public Flux<String> explore_zip(){
34
35         var abcFlux = Flux.just("A","B","C");
36
37         var defFlux = Flux.just("D","E","F");
38
39         return Flux.zip(abcFlux,defFlux,(first, second)-> first+second); //AD, BE, CF
40
41     }
```

```

183
184
185 > void explore_zip() {
186     //given
187
188     //when
189     var value = fluxAndMonoGeneratorService.explore_zip();
190
191     //then
192     StepVerifier.create(value)
193         .expectNext("AD", "BE", "CF")
194         .verifyComplete();
195
196 }

```

...
182 `String> explore_zip(){
183
184 flux = Flux.just("A","B","C");
185
186 flux = Flux.just("D","E","F");
187
188 flux.zip(abcFlux,defFlux,(first, second)-> first+second); //AD, BE, CF
189
190
191
192 ux(A,L,E,X)
193 `String> splitString(String name){
194 @
195 Array = name.split(regex: "");
196 flux.fromArray(charArray);
197
198

Tuple return type

```

196
197
198 > void explore_zip_1() {
199     //given
200
201     //when
202     var value = fluxAndMonoGeneratorService.explore_zip_1();
203
204     //then
205     StepVerifier.create(value)
206         .expectNext("AD14", "BE25", "CF36")
207         .verifyComplete();
208
209 }

```

191
192 `}
193
194 public Flux<String> explore_zip_1(){
195
196 var abcFlux = Flux.just("A","B","C");
197
198 var defFlux = Flux.just("D","E","F");
199
200 var _123Flux = Flux.just("1","2","3");
201 var _456Flux = Flux.just("4","5","6");
202
203
204 return Flux.zip(abcFlux,defFlux,_123Flux,_456Flux)
205 .map(t4 -> t4.getT1()+t4.getT2()+t4.getT3()+t4.getT4())
206 .log(); //AD14, BE25, CF36

```
288
289     public Flux<String> explore_zipWith(){
290
291         var abcFlux = Flux.just("A","B","C");
292
293         var defFlux = Flux.just("D","E","F");
294
295         return abcFlux.zipWith(defFlux, (first, second)-> first+second)
296             .log(); //AD, BE, CF
297
298     }
299
300
301     public Mono<String> explore_mergeZipWith_mono(){
302
303         var aMono = Mono.just("A"); //A
304
305         var bMono = Mono.just("B"); //B
306
307         return aMono.zipWith(bMono)
308             .map(t2-> t2.getT1()+t2.getT2()) //AB
309             .log(); // A, B
310
311     }
312
```