Combining Operators

Outline

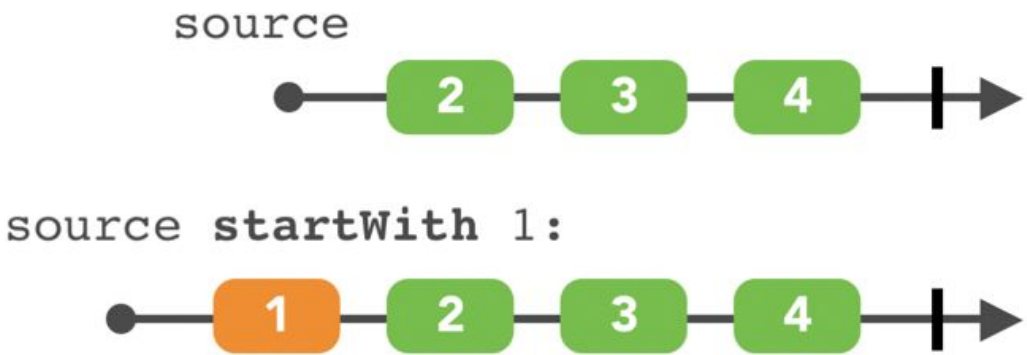
1. Prefixing and concatenating
2. Merging
3. Combining elements
4. Triggers
5. Combining element within a sequence

1. Prefixing and concatenating

1. startWith()
2. concat()
3. concatMap()

1.1 startWith()

- `startWith(_:)` allows to create an initial element for an Observable. This value must be the same type of other elements in Observable.



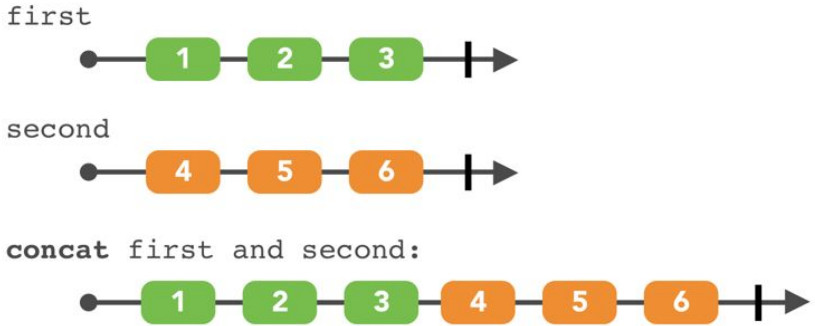
1.1. startWith()

- Example:

```
example(of: "startWith") {
    let disposeBag = DisposeBag()

    let observable = Observable.of(2, 3, 4, 5, 6, 7)
    observable.startWith(1)
        .subscribe { event in
            print(event)
        }
        .disposed(by: disposeBag)
}
/*
--- Example of: startWith ---
next(1)
next(2)
next(3)
next(4)
next(5)
next(6)
next(7)
completed
*/
```

1.2 concat()

- This operator concatenates Observables **sequentially** into a source Observable.
- The source Observable will complete only if all the child Observables complete.
- If one of the child Observables emit error, the source Observable will emit error immediately and terminate.



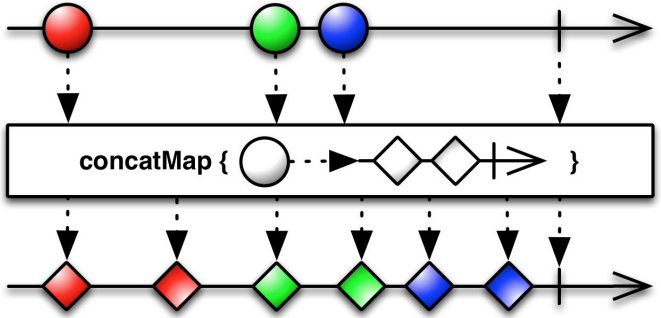
1.2. concat()

- Example:

```
example(of: "concat") {
    let disposeBag = DisposeBag()

    let one = Observable.of(1, 2, 3)
    let two = Observable.of(4, 5, 6)
    let three = Observable.of(7, 8, 9)

    let observable = Observable.concat([one, two, three])
    observable
        .subscribe { event in
            print(event)
        }
        .disposed(by: disposeBag)
}
/*
--- Example of: concat ---
next(1)
next(2)
next(3)
next(4)
next(5)
next(6)
next(7)
next(8)
next(9)
completed
*/
```

1.3 concatMap()

- This operator maps values to inner observable, subscribe and emit in order.
- The only difference between `flatMapLatest` and `concatMap` is it preserves the order of items.
- It has one flow, not good for asynchronous operations as waits for observable to finish all items in sequence.



1.3. concatMap()

- Example:

```
example(of: "concatMap") {
  // 1
  let sequences = [
    "German cities": Observable.of("Berlin", "Münich", "Frankfurt"),
    "Spanish cities": Observable.of("Madrid", "Barcelona", "Valencia")
  ]

  // 2
  let observable = Observable.of("German cities", "Spanish cities")
    .concatMap { country in sequences[country] ?? .empty() }

  // 3
  _ = observable.subscribe(onNext: { string in
      print(string)
    })
}
```

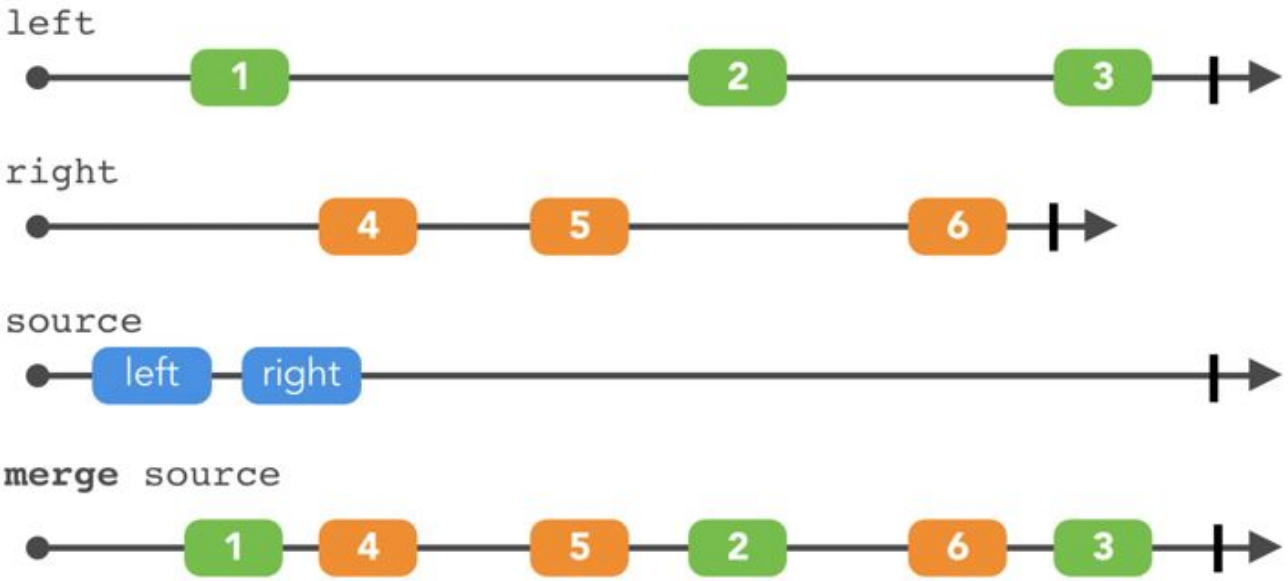
2. Merging

1. merge()

2.1. merge()

- merge(_:) operator allows to combine multiple Observables by merging their emissions into a source Observable.
- The source Observable will complete only if all inner Sequences and source Observable both complete.
- All inner Sequences operate independently.
- If any inner Sequence emits Error, the source Observable emits Error immediately and terminates.

2.1. merge()



2.1. merge()

- Example:

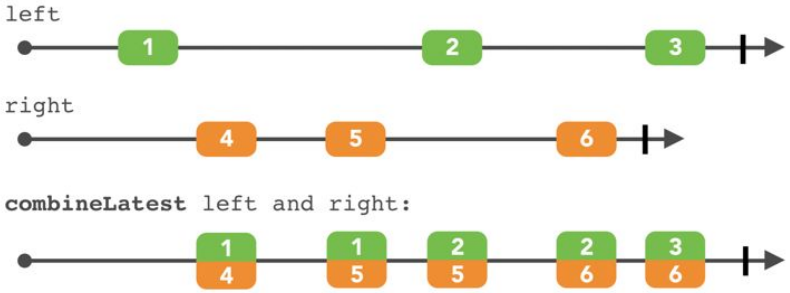
```
example(of: "merge") {
    let disposeBag = DisposeBag()

    let firstObservable = PublishSubject<String>()
    let secondObservable = PublishSubject<String>()

    Observable.merge(firstObservable, secondObservable)
        .subscribe { element in
            print(element)
        }
        .disposed(by: disposeBag)

    firstObservable.onNext("firstObservable 1")
    secondObservable.onNext("secondObservable 1")
    secondObservable.onNext("secondObservable 2")
    secondObservable.onNext("secondObservable 3")
    firstObservable.onNext("firstObservable 2")
    secondObservable.onNext("secondObservable 4")
}
/*
--- Example of: merge ---
next(firstObservable 1)
next(secondObservable 1)
next(secondObservable 2)
next(secondObservable 3)
next(firstObservable 2)
next(secondObservable 4)
*/
```

3. Combining elements

1. combineLatest(_:_:resultSelector:)
2. zip(_:_:resultSelector:)

3.1. combineLatest(_:_:resultSelector:)

- Every time one of the inner (combined) sequences emits a value, it calls a closure you provide. You receive the last value emitted by each of the inner sequences.
- Like the `map(_:)` operator, `combineLatest(_:_:resultSelector:)` creates an observable whose type is the closure return type.



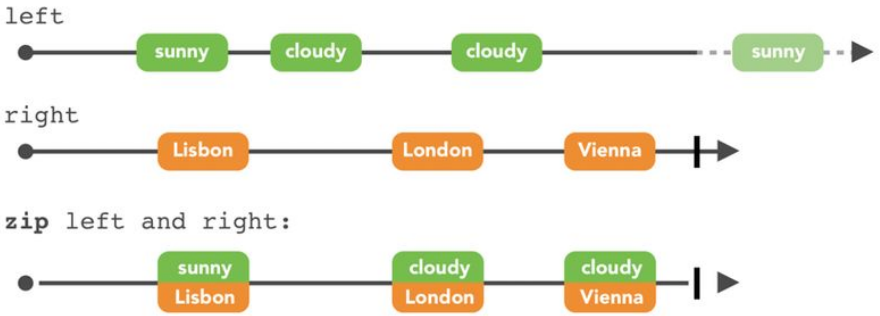
3.1. combineLatest(_:_:resultSelector:)

- Example:

```
example(of: "combineLatest") {
    let disposeBag = DisposeBag()

    let one = Observable.of(1, 2, 3)
    let two = Observable.of(4, 5, 6)

    let observable = Observable
        .combineLatest(one, two) { e1, e2 -> (Int, Int) in
            return (e1, e2)
    }
    observable
        .subscribe { event in
            print("event: \(event)")
        }
        .disposed(by: disposeBag)
}
/*
--- Example of: combineLatest ---
event: next((1, 4))
event: next((2, 4))
event: next((2, 5))
event: next((3, 5))
event: next((3, 6))
event: completed
*/
```

3.2. zip(_:_:resultSelector:)

- This operator combine elements that have the same index in the Observable into a new element and return to the subscribers.
- If one of the input Observables does not have an appropriate element at the index in other Observables, `zip(_:_:resultSelector:)` will not emit element.



3.2. zip(_:_:resultSelector:)

- Example:

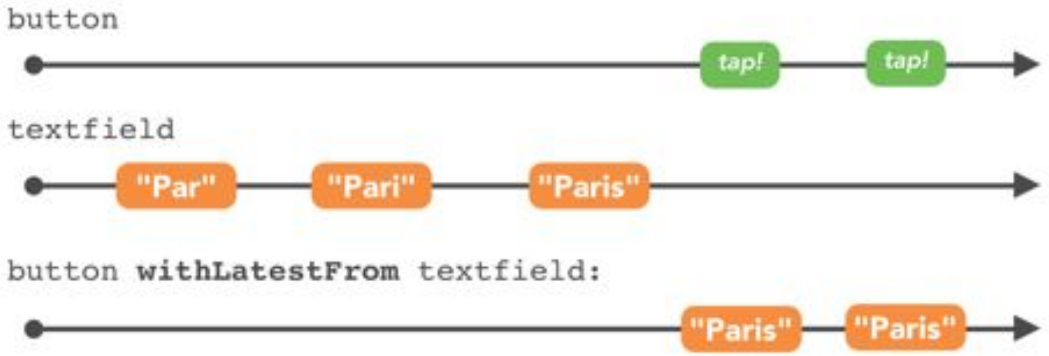
```
example(of: "zip") {
    let one = Observable.of(1, 2, 3, 7)
    let two = Observable.of(4, 5, 6)
    let three = Observable.of(7, 8)

    let observable = Observable
        .zip(one, two, three) { element1, element2, element3 -> (Int, Int, Int) in
            return (element1, element2, element3)
    }
    observable
        .subscribe { event in
            print("event: \(event)")
        }
        .disposed(by: disposeBag)
}
/*
--- Example of: zip ---
event: next((1, 4, 7))
event: next((2, 5, 8))
event: completed
*/
```

4. Triggers

1. withLatestFrom(_:)

4.1 withLatestFrom(_:)

- This operator receive an Observable as the input and transform a trigger into the latest event emitted from the input Observable.



4.1 withLatestFrom(_:)

- Example:

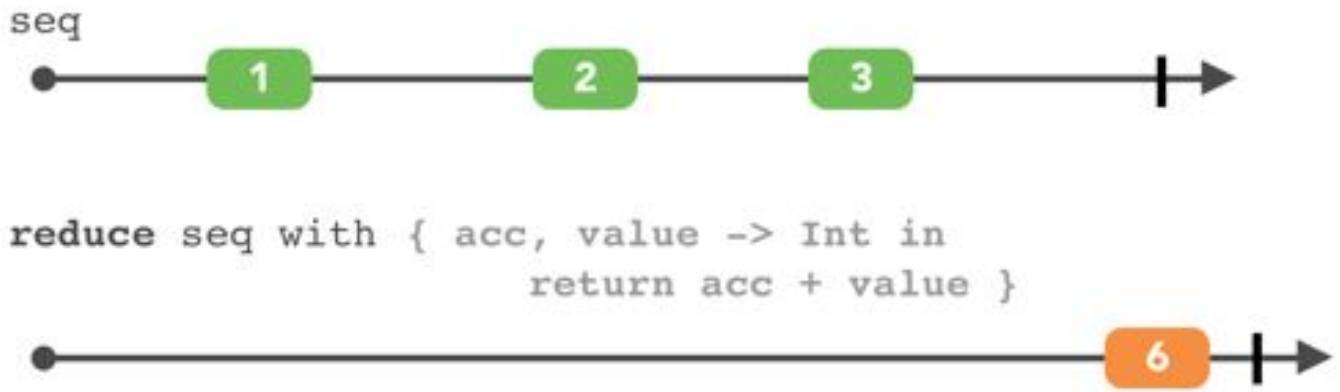
```
example(of: "withLatestFrom") {
    let disposeBag = DisposeBag()
    let button = PublishSubject<Void>()
    let textField = PublishSubject<String>()

    // 2
    button
        .withLatestFrom(textField)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)

    // 3
    textField.onNext("Par")
    textField.onNext("Pari")
    textField.onNext("Paris")
    button.onNext(())
    button.onNext(())
}
/*
--- Example of: zip ---
Paris
Paris
*/
```

5. Combining element within a sequence

1. reduce(_:_:)
2. scan(_:_:)

5.1 reduce(_:_:)

- This operator applies a function to each item emitted by an Observable, sequentially, and emit the final value.



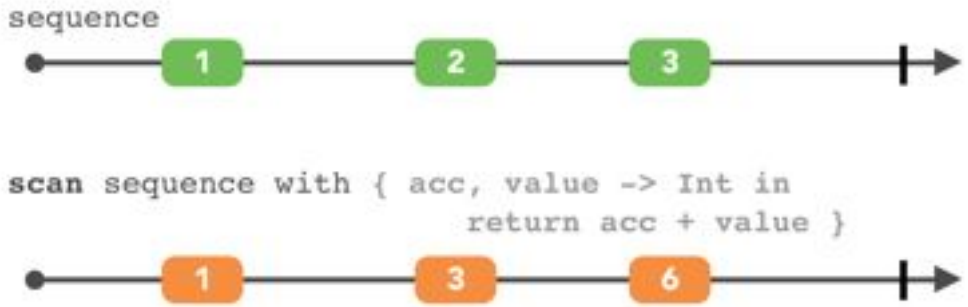
5.1 reduce(_:_:)

- Example:

```
example(of: "reduce") {
    let disposeBag = DisposeBag()

    Observable.of(1, 3, 5, 7, 9)
        .reduce(0, accumulator: +)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}
/*
--- Example of: reduce ---
25
*/
```

5.2 scan(_:_:)

- This operator applies a function to each item emitted by an Observable, sequentially, and emit each successive value.



5.2 scan(_:_:)

- Example:

```
example(of: "scan") {
    let disposeBag = DisposeBag()

    Observable.of(1, 3, 5, 7, 9)
        .scan(0, accumulator: +)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}
/*
--- Example of: scan ---
1
4
9
16
25
*/
```

Question & Answer?