

Sequence Level Operations



Overview

- ➔ **Utility Sequences**
- ➔ **Skipping around values**
- ➔ **Concatenation**
- ➔ **Handling exceptions**
- ➔ **Zipping sequences together**
- ➔ **Grouping**

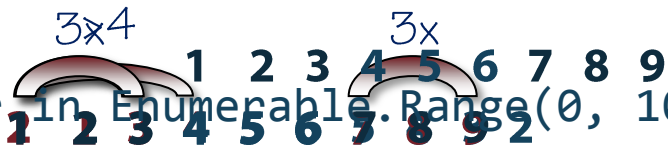
Utility Sequences

- ➔ Empty
- ➔ Return
- ➔ Repeat

Skip

➔ Skips OnNext

- ❑ fix number of times
- ❑ based on value
- ❑ at end of sequence
- ❑ waits for another sequence


from number in Enumerable.Range(0, 10) select number
sequence.Subscribe(value => Console.WriteLine(value))

Concat

- ➔ **Combines observables in order**
 - values must be same type
- ➔ **Subscriptions run in order**

Subscribe



Concat

`IObservable<string>`

`IObservable<string>`

`IObservable<string>`

Exceptions

➔ Catch

- produces IObservable

➔ OnErrorResumeNext

➔ Throw

- produces empty IObservable



Zip

- ➔ Zipper, not compressor
- ➔ Combines values from two observables
 - produces new observable
 - first input to finish completes processing

Don't block the sequence

➔ IObservable vs. ~IObservable

- ~IObservable can block the sequence

```
IObservable<int>  
sequence.Count().Subscribe(Console.WriteLine)  
  
Console.WriteLine(sequence.Count().First())  
int
```


GroupBy

➔ Produces observable sequence of groups

- handy for statistical calculations

New York, NY
Cleveland, OH
Newark, NJ
Youngstown, OH
Mawah, NJ
Albany, NY
Columbus, OH

New York, NY
Albany, NY

Cleveland, OH
Youngstown, OH
Columbus, OH

Newark, NJ
Mawah, NJ

Summary

- ➔ Use Empty, Repeat, and Return for simple sequences
- ➔ SkipLast may introduce delay
- ➔ Catch processes exceptions
- ➔ OnErrorResumeNext can be used like Concat
- ➔ Throw exceptions as you normally would
- ➔ Zip to process two sequences side-by-side
- ➔ GroupBy processes sequence as stream

References

- **Thiotimeline**
 - <http://en.wikipedia.org/wiki/Thiotimeline>
- **PowerGREP**
 - <http://www.powergrep.com>