The Go Programming Language

Package rxgo

```
import "github.com/yilin0041/service-computing/rxgo"
```


Overview ▾

Package rxgo provides basic supporting to reactiveX of the Go.

Index ▾

Package files

filtering.go generators.go rxgo.go transforms.go utility.go

Variables

default buffer of channels

```
var BufferLen uint = 128
```

if user function throw EoFlow, the Observeable will stop and close it

```
var ErrEoFlow = errors.New("End of Flow!")
```

operator func error

```
var ErrFuncFlip = errors.New("Operator Func Error")
```

Subscribe paeameter error

```
var ErrFuncOnNext = errors.New("Subscribe paramteter needs func(x anytype) or
Observer or ObserverWithContext")
```

if user function throw SkipItem, the Observeable will skip current item

```
var ErrSkipItem = errors.New("Skip item!")
```

type **FlowableError**

Error that can flow to subscriber or user function which processes error as an input

```
type FlowableError struct {
```

```
    Err      error
    Elements interface{}
}
```

func (FlowableError) Error

```
func (e FlowableError) Error() string
```

type InnerObserver

Test Observer

```
type InnerObserver struct {
    // contains filtered or unexported fields
}
```

func (InnerObserver) OnCompleted

```
func (o InnerObserver) OnCompleted()
```

func (InnerObserver) OnError

```
func (o InnerObserver) OnError(e error)
```

func (InnerObserver) OnNext

```
func (o InnerObserver) OnNext(x interface{})
```

type Observable

An Observable is a 'collection of items that arrive over time'. Observables can be used to model asynchronous events. Observables can also be chained by operators to transformed, combined those items The Observable's operators, by default, run with a channel size of 128 elements except that the source (first) observable has no buffer

```
type Observable struct {
    Name string
    // contains filtered or unexported fields
}
```

func Empty

```
func Empty() *Observable
```

create an Observable that emits no items but terminates normally

func From

```
func From(items interface{}) *Observable
```

convert Slice, Channel, and Observable into Observables

func Generator

```
func Generator(sf sourceFunc) *Observable
```

func Just

```
func Just(items ...interface{}) *Observable
```

Just creates an Observable with the provided item(s).

func Never

```
func Never() *Observable
```

create an Observable that emits no items and does not terminate. It is important for combining with other Observables

func Range

```
func Range(start, end int) *Observable
```

Range creates an Observable that emits a particular range of sequential integers.

func Start

```
func Start(f interface{}) *Observable
```

creates an Observable with the provided item(s) producing by the function `func() (val anytype, end bool)`

func Throw

```
func Throw(e error) *Observable
```

create an Observable that emits no items and terminates with an error

func (*Observable) Debounce

```
func (parent *Observable) Debounce(_debounce time.Duration) (o *Observable)
```

Debounce : only emit an item from an Observable if a particular timespan has passed without it emitting another item

func (*Observable) Debug

```
func (o *Observable) Debug(debug bool) *Observable
```

set a innerMonitor for debug

func (*Observable) Distinct

```
func (parent *Observable) Distinct() (o *Observable)
```

Distinct :suppress duplicate items emitted by an Observable

func (*Observable) ElementAt

```
func (parent *Observable) ElementAt(index int) (o *Observable)
```

ElementAt :emit only item n emitted by an Observable

func (*Observable) Filter

```
func (parent *Observable) Filter(f interface{}) (o *Observable)
```

Filter `func(x anytype) bool` filters items in the original Observable and returns a new Observable with the filtered items.

func (*Observable) First

```
func (parent *Observable) First() (o *Observable)
```

First :emit only the first item, or the first item that meets a condition, from an Observable

func (*Observable) FlatMap

```
func (parent *Observable) FlatMap(f interface{}) (o *Observable)
```

FlatMap maps each item in Observable by the function with `func(x anytype) (o *Observable) ` and returns a new Observable with merged observables appling on each items.

func (*Observable) IgnoreElement

```
func (parent *Observable) IgnoreElement() (o *Observable)
```

IgnoreElement :do not emit any items from an Observable but mirror its termination notification

func (*Observable) Last

```
func (parent *Observable) Last() (o *Observable)
```

Last :emit only the last item emitted by an Observable

func (*Observable) Map

```
func (parent *Observable) Map(f interface{}) (o *Observable)
```

Map maps each item in Observable by the function with `func(x anytype) anytype` and returns a new Observable with applied items.

func (*Observable) ObserveOn

```
func (o *Observable) ObserveOn(t ThreadModel) *Observable
```

func (*Observable) Sample

```
func (parent *Observable) Sample(_sample time.Duration) (o *Observable)
```

Sample :emit the most recent item emitted by an Observable within periodic time intervals

func (*Observable) SetBufferLen

```
func (o *Observable) SetBufferLen(length uint) *Observable
```

func (*Observable) SetMonitor

```
func (o *Observable) SetMonitor(observer Observer) *Observable
```

set a observer to monite items in data stream

func (*Observable) Skip

```
func (parent *Observable) Skip(num int) (o *Observable)
```

Skip :suppress the first n items emitted by an Observable

func (*Observable) SkipLast

```
func (parent *Observable) SkipLast(num int) (o *Observable)
```

SkipLast :suppress the last n items emitted by an Observable

func (*Observable) Subscribe

```
func (o *Observable) Subscribe(ob interface{})
```

func (*Observable) SubscribeOn

```
func (o *Observable) SubscribeOn(t ThreadModel) *Observable
```

func (*Observable) Take

```
func (parent *Observable) Take(num int) (o *Observable)
```

Take :emit only the first n items emitted by an Observable

func (*Observable) TakeLast

```
func (parent *Observable) TakeLast(num int) (o *Observable)
```

TakeLast :emit only the last n items emitted by an Observable

func (*Observable) TransformOp

```
func (parent *Observable) TransformOp(tf transformFunc) (o *Observable)
```

type Observer

Observer subscribes to an Observable. Then that observer reacts to whatever item or sequence of items the Observable emits.

```
type Observer interface {
    OnNext(x interface{})
    OnError(error)
    OnCompleted()
}
```

type ObserverMonitor

Create observer quickly with function

```
type ObserverMonitor struct {
    Next              func(x interface{})
    Error             func(error)
    Completed         func()
```

```
    Context           func() context.Context // an observer context musit gived
when observables before connected
    AfterConnected    func()
    CancelObservables context.CancelFunc
}
```

func (ObserverMonitor) GetObserverContext

```
func (o ObserverMonitor) GetObserverContext() (c context.Context)
```

func (ObserverMonitor) OnCompleted

```
func (o ObserverMonitor) OnCompleted()
```

func (ObserverMonitor) OnConnected

```
func (o ObserverMonitor) OnConnected()
```

func (ObserverMonitor) OnError

```
func (o ObserverMonitor) OnError(e error)
```

func (ObserverMonitor) OnNext

```
func (o ObserverMonitor) OnNext(x interface{})
```

func (ObserverMonitor) Unsubscribe

```
func (o ObserverMonitor) Unsubscribe()
```

type ObserverWithContext

Make Observables Context and support unsubscribe operation

```
type ObserverWithContext interface {
    Observer
    GetObserverContext() context.Context // you must create a cancelable context
here when unsubscribe
    OnConnected()
    Unsubscribe()
}
```

type ThreadModel

```
type ThreadModel uint
```

```
const (
    ThreadingDefault   ThreadModel = iota // one observable served by one
goroutine
    ThreadingIO                           // each item served by one goroutine
    ThreadingComputing                    // each item served by one goroutine in
a limited group
)
```

Subdirectories

Name
..
test