**OBSERVABLE**

In ReactiveX un osservatore si iscrive a un Observable . Quindi quell'osservatore reagisce a qualunque elemento o sequenza di elementi emetta l'Osservabile . Questo modello facilita le operazioni simultanee perché non ha bisogno di bloccarsi in attesa che l'Observable emetta oggetti, ma crea invece una sentinella sotto forma di osservatore pronto a reagire in modo appropriato in qualunque momento futuro l'Observable lo faccia.

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Immagine che contiene schermata, testo, diagramma, linea

Descrizione generata automaticamente

**Sfondo**

In molte attività di programmazione software, ti aspetti più o meno che le istruzioni che scrivi vengano eseguite e completate in modo incrementale, una alla volta, nell'ordine in cui le hai scritte. Ma in ReactiveX molte istruzioni possono essere eseguite in parallelo e i loro risultati vengono successivamente catturati, in ordine arbitrario, da “osservatori”. Invece di *chiamare* un metodo, si definisce un meccanismo per recuperare e trasformare i dati, sotto forma di un “osservabile”, e quindi si *iscrive* un osservatore ad esso, a quel punto il meccanismo precedentemente definito entra in azione con l’osservatore che fa da sentinella. per catturare e rispondere alle sue emissioni ogni volta che sono pronte.

Un vantaggio di questo approccio è che quando si hanno una serie di attività che non dipendono l'una dall'altra, è possibile avviarle tutte contemporaneamente anziché attendere che ciascuna finisca prima di iniziare quella successiva: in questo modo, l'intero il pacchetto di attività richiede per il completamento solo lo stesso tempo dell'attività più lunga nel pacchetto.

Esistono molti termini usati per descrivere questo modello di programmazione e progettazione asincrona. Questo documento utilizzerà i seguenti termini: Un *osservatore si iscrive* a un *Observable* . Un Observable *emette elementi* o invia *notifiche* ai suoi osservatori chiamando i metodi degli osservatori.

In altri documenti e altri contesti, ciò che chiamiamo “osservatore” è talvolta chiamato “abbonato”, “osservatore” o “reattore”. Questo modello in generale viene spesso definito [“modello del reattore”](http://en.wikipedia.org/wiki/Reactor_pattern) .

# Istituzione degli osservatori

In una chiamata di metodo ordinaria, ovvero *non* il tipo di chiamate parallele asincrone tipiche di ReactiveX, il flusso è qualcosa del genere:

1. Chiama un metodo.
2. Memorizza il valore restituito da quel metodo in una variabile.
3. Usa quella variabile e il suo nuovo valore per fare qualcosa di utile.

Oppure qualcosa del genere:

# 

Nel modello asincrono il flusso è più simile a questo:

1. Definire un metodo che faccia qualcosa di utile con il valore restituito dalla chiamata asincrona; questo metodo fa parte *dell'osservatore* .
2. Definire la chiamata asincrona stessa come *Observable* .
3. Associa l'osservatore a quell'Observable iscrivendolo *(* questo avvia anche le azioni dell'Observable).
4. Vai avanti con i tuoi affari; ogni volta che la chiamata ritorna, il metodo dell'osservatore inizierà a operare sul suo valore o sui suoi valori restituiti: gli *elementi* emessi dall'Observable.

Che assomiglia a questo:

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

## **onNext, onCompleted e onError**

[Il Subscribemetodo](https://reactivex.io/documentation/operators/subscribe.html) è come connettere un osservatore a un osservabile. Il tuo osservatore implementa alcuni sottoinsiemi dei seguenti metodi:

**onNext**

Un Observable chiama questo metodo ogni volta che l'Observable emette un elemento. Questo metodo prende come parametro l'item emesso dall'Observable.

**onError**

Un Observable chiama questo metodo per indicare che non è riuscito a generare i dati previsti o ha riscontrato qualche altro errore. Non effettuerà ulteriori chiamate a onNexto onCompleted. Il onErrormetodo prende come parametro un'indicazione di cosa ha causato l'errore.

**onCompleted**

Un Observable chiama questo metodo dopo averlo chiamato onNextper l'ultima volta, se non ha riscontrato errori.

Secondo i termini del [contratto Observable](https://reactivex.io/documentation/contract.html) , può chiamare onNextzero o più volte, quindi può far seguire a tali chiamate una chiamata a uno onCompletedo onErrorma non a entrambi, che sarà la sua ultima chiamata. Per convenzione, in questo documento, le chiamate a onNextsono solitamente chiamate “emissioni” di elementi, mentre le chiamate a onCompletedo onErrorsono chiamate “notifiche”.

Un esempio di chiamata più completo subscribeè simile al seguente:

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

## **Cancellazione, cancellami**

In alcune implementazioni di ReactiveX, esiste un'interfaccia di osservazione specializzata, Subscriberche implementa un unsubscribemetodo. È possibile chiamare questo metodo per indicare che l'abbonato non è più interessato a nessuno degli osservabili a cui è attualmente iscritto. Tali Osservabili possono quindi (se non hanno altri osservatori interessati) scegliere di smettere di generare nuovi elementi da emettere.

I risultati di questa cancellazione si riverseranno attraverso la catena di operatori che si applica all'Observable a cui l'osservatore si è iscritto, e questo farà sì che ogni anello della catena smetta di emettere elementi. Tuttavia, non è garantito che ciò accada immediatamente ed è possibile che un Osservabile generi e tenti di emettere oggetti per un po' anche se non rimangono osservatori per osservare queste emissioni.

## **Alcune note sulle convenzioni di denominazione**

Ogni implementazione specifica della lingua di ReactiveX ha le sue peculiarità di denominazione. Non esiste uno standard di denominazione canonico, sebbene esistano molti punti in comune tra le implementazioni.

Inoltre, alcuni di questi nomi hanno implicazioni diverse in altri contesti, o sembrano scomodi nell'idioma di un particolare linguaggio implementativo.

Ad esempio c'è il modello di denominazione (ad esempio , , ). In alcuni contesti tali nomi indicherebbero metodi mediante i quali vengono registrati i gestori di eventi . In ReactiveX, tuttavia, nominano i gestori di eventi stessi. on*Event*onNextonCompletedonError

# Osservabili “caldi” e “freddi”.

Quando un osservabile inizia a emettere la sua sequenza di elementi? Dipende dall'Osservabile. Un osservabile “caldo” può iniziare a emettere elementi non appena viene creato, e quindi qualsiasi osservatore che successivamente si iscrive a quell’osservabile può iniziare a osservare la sequenza da qualche parte nel mezzo. Un osservabile “freddo”, d'altra parte, attende finché un osservatore non si iscrive ad esso prima di iniziare a emettere elementi, e quindi a tale osservatore è garantito di vedere l'intera sequenza dall'inizio.

In alcune implementazioni di ReactiveX, esiste anche qualcosa chiamato osservabile “connettibile”. Un tale osservabile non inizia a emettere elementi finché non viene chiamato il suo metodo [Connect](https://reactivex.io/documentation/operators/connect.html) , indipendentemente dal fatto che qualche osservatore si sia iscritto o meno.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Composizione tramite operatori osservabili**

Gli osservabili e gli osservatori sono solo l'inizio di ReactiveX. Di per sé non sarebbero altro che una leggera estensione del modello di osservatore standard, più adatto a gestire una sequenza di eventi piuttosto che un singolo callback.

Il vero potere arriva con le “estensioni reattive” (da qui “ReactiveX”) – operatori che ti consentono di trasformare, combinare, manipolare e lavorare con le sequenze di elementi emessi da Observables.

Questi operatori Rx consentono di comporre sequenze asincrone insieme in modo dichiarativo con tutti i vantaggi in termini di efficienza dei callback ma senza gli svantaggi della nidificazione dei gestori di callback tipicamente associati ai sistemi asincroni.

Questa documentazione raggruppa le informazioni sui [vari operatori](https://reactivex.io/documentation/operators.html#alphabetical) ed esempi del loro utilizzo nelle seguenti pagine:

[**Creazione di osservabili**](https://reactivex.io/documentation/operators.html#creating)

Create, Defer, Empty/ Never/ Throw​​, From, Interval, Just, Range, Repeat, Start, eTimer

[**Trasformazione di elementi osservabili**](https://reactivex.io/documentation/operators.html#transforming)

Buffer, FlatMap, GroupBy, Map, Scan, e Window

[**Filtraggio degli osservabili**](https://reactivex.io/documentation/operators.html#filtering)

Debounce, Distinct, ElementAt, Filter, First, IgnoreElements, Last, Sample, Skip, SkipLast, Take, eTakeLast

[**Combinazione di osservabili**](https://reactivex.io/documentation/operators.html#combining)

And/ Then/ When, CombineLatest, Join, Merge, StartWith, Switch, eZip

[**Operatori di gestione degli errori**](https://reactivex.io/documentation/operators.html#error)

CatchERetry

[**Operatori di utilità**](https://reactivex.io/documentation/operators.html#utility)

Delay, Do, Materialize/ Dematerialize, ObserveOn, Serialize, Subscribe, SubscribeOn, TimeInterval, Timeout, Timestamp, eUsing

[**Operatori condizionali e booleani**](https://reactivex.io/documentation/operators.html#conditional)

All, Amb, Contains, DefaultIfEmpty, SequenceEqual, SkipUntil, SkipWhile, TakeUntil, eTakeWhile

[**Operatori matematici e aggregati**](https://reactivex.io/documentation/operators.html#mathematical)

Average, Concat, Count, Max, Min, Reduce, eSum

[**Conversione di osservabili**](https://reactivex.io/documentation/operators.html#conversion)

To

[**Operatori osservabili collegabili**](https://reactivex.io/documentation/operators.html#connectable)

Connect, Publish, RefCount, eReplay

[**Operatori di contropressione**](https://reactivex.io/documentation/operators/backpressure.html)

una varietà di operatori che applicano particolari politiche di controllo del flusso

Queste pagine includono informazioni su alcuni operatori che non fanno parte del nucleo di ReactiveX ma sono implementati in una o più implementazioni specifiche della lingua e/o moduli opzionali.

## **Operatori di concatenamento**

La maggior parte degli operatori opera su un Observable e restituisce un Observable. Ciò consente di applicare questi operatori uno dopo l'altro, in una catena. Ogni operatore della catena modifica l'Observable che risulta dall'operazione dell'operatore precedente.

Esistono altri pattern, come il Builder Pattern, in cui una varietà di metodi di una particolare classe operano su un elemento di quella stessa classe modificando quell'oggetto attraverso l'operazione del metodo. Questi modelli consentono inoltre di concatenare i metodi in modo simile. Ma mentre nel Builder Pattern, l'ordine in cui i metodi appaiono nella catena di solito non ha importanza, con gli operatori Observable l' ordine è importante .

Una catena di operatori Observable non operano indipendentemente sull'Observable originale che origina la catena, ma operano a turno , operando ciascuno sull'Observable generato dall'operatore immediatamente precedente nella catena.

**SUBJECT**

Un soggetto è una sorta di bridge o proxy disponibile in alcune implementazioni di ReactiveX che funge sia da osservatore che da osservabile. Poiché è un osservatore, può iscriversi a uno o più osservabili e, poiché è un osservabile, può passare attraverso gli elementi che osserva riemettendoli e può anche emettere nuovi elementi.

Poiché un soggetto si iscrive a un osservabile, farà sì che quell'osservabile inizi a emettere oggetti (se quell'osservabile è "freddo", cioè se attende una sottoscrizione prima di iniziare a emettere oggetti). Ciò può avere l'effetto di rendere il Soggetto risultante una variante osservabile “calda” dell'Osservabile “freddo” originale.

## **Varietà di soggetti**

Ne esistono quattro varietà Subjectprogettate per casi d'uso particolari. Non tutti questi sono disponibili in tutte le implementazioni e alcune implementazioni utilizzano altre convenzioni di denominazione (ad esempio, in RxScala, quello che viene chiamato "PublishSubject" qui è noto semplicemente come "Subject"):

### **AsyncSubject**

**Immagine che contiene schermata, linea, diagramma, Diagramma

Descrizione generata automaticamente**

An AsyncSubjectemette l'ultimo valore (e solo l'ultimo valore) emesso dalla sorgente Observable e solo dopo il completamento di tale source Observable. (Se l'Observable di origine non emette alcun valore, viene AsyncSubjectcompletato anche senza emettere alcun valore.)

**Immagine che contiene schermata, linea, diagramma, Diagramma

Descrizione generata automaticamente**

Emetterà lo stesso valore finale anche a tutti gli osservatori successivi. Tuttavia, se l'Observable di origine termina con un errore, AsyncSubjectnon emetterà alcun elemento, ma trasmetterà semplicemente la notifica di errore dall'Observable di origine.

**BehaviorSubject**

**Immagine che contiene schermata, diagramma, linea

Descrizione generata automaticamente**

Quando un osservatore si iscrive a un BehaviorSubject, inizia emettendo l'elemento più recentemente emesso dalla sorgente Observable (o un valore seed/predefinito se nessuno è stato ancora emesso) e poi continua a emettere qualsiasi altro elemento emesso successivamente dalla sorgente Observable(s) ).

**Immagine che contiene schermata, linea, diagramma, Diagramma

Descrizione generata automaticamente**

Tuttavia, se l'Observable di origine termina con un errore, BehaviorSubject non invierà alcun elemento agli osservatori successivi, ma passerà semplicemente la notifica di errore dall'Observable di origine.

**PublishSubject**

**Immagine che contiene schermata, diagramma, linea, Diagramma

Descrizione generata automaticamente**

PublishSubject emette a un osservatore solo quegli elementi emessi dagli osservabili sorgente successivamente al momento della sottoscrizione.

Tieni presente che a PublishSubject può iniziare a emettere elementi immediatamente dopo la creazione (a meno che tu non abbia adottato misure per impedirlo), e quindi c'è il rischio che uno o più elementi possano andare persi tra il momento in cui il Soggetto viene creato e l'osservatore si iscrive ad esso. Se devi garantire la consegna di tutti gli elementi dall'Observable di origine, dovrai formare quell'Observable in modo [Create](https://reactivex.io/documentation/operators/create.html)da poter reintrodurre manualmente il comportamento "a freddo" dell'Observable (controllando che tutti gli osservatori si siano iscritti prima di iniziare a emettere elementi) oppure passa all'utilizzo di a ReplaySubject.

Immagine che contiene schermata, linea, diagramma, Diagramma

Descrizione generata automaticamente

Se l'Observable di origine termina con un errore, PublishSubjectnon invierà alcun elemento agli osservatori successivi, ma trasmetterà semplicemente la notifica di errore dall'Observable di origine.

### **ReplaySubject**

Immagine che contiene schermata, diagramma

Descrizione generata automaticamente

ReplaySubject emette a qualsiasi osservatore tutti gli elementi emessi dagli osservabili sorgente, indipendentemente da quando l'osservatore si iscrive.

Esistono anche versioni ReplaySubject che eliminano i vecchi elementi una volta che il buffer di replay minaccia di crescere oltre una certa dimensione o quando è trascorso un periodo di tempo specificato da quando gli elementi sono stati originariamente emessi.

Se usi a ReplaySubject come osservatore, fai attenzione a non chiamare il suo onNext metodo (o altri onmetodi) da più thread, poiché ciò potrebbe portare a chiamate coincidenti (non sequenziali), che violano [il contratto Observable](https://reactivex.io/documentation/contract.html) e creano un'ambiguità nel risultato A seconda di quale elemento o notifica deve essere riprodotto per primo.

**OPERATORS**

# introduzione

Ogni implementazione specifica della lingua di ReactiveX implementa una serie di operatori. Sebbene vi sia molta sovrapposizione tra le implementazioni, esistono anche alcuni operatori implementati solo in determinate implementazioni. Inoltre, ogni implementazione tende a nominare i propri operatori in modo che assomiglino a quelli di metodi simili già familiari in altri contesti di quel linguaggio.

## **Operatori di concatenamento**

La maggior parte degli operatori opera su un Observable e restituisce un Observable. Ciò consente di applicare questi operatori uno dopo l'altro, in una catena. Ogni operatore della catena modifica l'Observable che risulta dall'operazione dell'operatore precedente.

Esistono altri pattern, come il Builder Pattern, in cui una varietà di metodi di una particolare classe operano su un elemento di quella stessa classe modificando quell'oggetto attraverso l'operazione del metodo. Questi modelli consentono inoltre di concatenare i metodi in modo simile. Ma mentre nel Builder Pattern, l'ordine in cui i metodi appaiono nella catena di solito non ha importanza, con gli operatori Observable l' ordine è importante .

Una catena di operatori Observable non operano indipendentemente sull'Observable originale che origina la catena, ma operano a turno , operando ciascuno sull'Observable generato dall'operatore immediatamente precedente nella catena.

## **Gli operatori di ReactiveX**

Questa pagina elenca innanzitutto quelli che potrebbero essere considerati gli operatori "principali" in ReactiveX e collegamenti a pagine che contengono informazioni più approfondite su come funzionano questi operatori e su come particolari versioni ReactiveX specifiche della lingua hanno implementato questi operatori.

Successivamente c'è un "albero decisionale" che può aiutarti a scegliere l'operatore più appropriato al tuo caso d'uso.

Infine, c'è un elenco alfabetico della maggior parte degli operatori disponibili nelle numerose implementazioni specifiche del linguaggio di ReactiveX. Questi si collegano alla pagina che documenta l'operatore principale che assomiglia di più all'operatore specifico della lingua (quindi, ad esempio, l'operatore "SelectMany" di Rx.NET si collega alla documentazione dell'operatore FlatMap ReactiveX, di cui "SelectMany" è l' operatore implementazione Rx.NET).

Se desideri implementare il tuo operatore, consulta [Implementazione dei tuoi operatori](https://reactivex.io/documentation/implement-operator.html) .

# Operatori per categoria

## **Creazione di osservabili**

Operatori che originano nuovi Osservabili.

* [**Create**](https://reactivex.io/documentation/operators/create.html)- crea un osservabile da zero chiamando i metodi dell'osservatore a livello di codice
* [**Defer**](https://reactivex.io/documentation/operators/defer.html)- non creare l'osservabile finché l'osservatore non si iscrive e creare un nuovo osservabile per ciascun osservatore
* [**Empty/ Never/Throw**](https://reactivex.io/documentation/operators/empty-never-throw.html) — crea Osservabili che hanno un comportamento molto preciso e limitato
* [**From**](https://reactivex.io/documentation/operators/from.html)- convertire qualche altro oggetto o struttura dati in un osservabile
* [**Interval**](https://reactivex.io/documentation/operators/interval.html)— creare un osservabile che emette una sequenza di numeri interi spaziati da un particolare intervallo di tempo
* [**Just**](https://reactivex.io/documentation/operators/just.html)— convertire un oggetto o un insieme di oggetti in un osservabile che emette quello o quegli oggetti
* [**Range**](https://reactivex.io/documentation/operators/range.html)- crea un osservabile che emette un intervallo di numeri interi sequenziali
* [**Repeat**](https://reactivex.io/documentation/operators/repeat.html)- creare un osservabile che emette ripetutamente un particolare elemento o una sequenza di elementi
* [**Start**](https://reactivex.io/documentation/operators/start.html)— crea un osservabile che emette il valore restituito da una funzione
* [**Timer**](https://reactivex.io/documentation/operators/timer.html)- creare un osservabile che emette un singolo elemento dopo un determinato ritardo

## **Trasformazione degli osservabili**

Operatori che trasformano gli elementi emessi da un osservabile.

* [**Buffer**](https://reactivex.io/documentation/operators/buffer.html)— raccogliere periodicamente oggetti da un osservabile in pacchetti ed emettere questi pacchetti anziché emetterli uno alla volta
* [**FlatMap**](https://reactivex.io/documentation/operators/flatmap.html)— trasformare gli elementi emessi da un osservabile in osservabili, quindi appiattire le emissioni di questi in un unico osservabile
* [**GroupBy**](https://reactivex.io/documentation/operators/groupby.html)— dividere un osservabile in un insieme di osservabili che emettono ciascuno un diverso gruppo di elementi dall'osservabile originale, organizzati per chiave
* [**Map**](https://reactivex.io/documentation/operators/map.html)— trasformare gli elementi emessi da un osservabile applicando una funzione a ciascun elemento
* [**Scan**](https://reactivex.io/documentation/operators/scan.html)— applica una funzione a ciascun elemento emesso da un osservabile, in sequenza, ed emette ogni valore successivo
* [**Window**](https://reactivex.io/documentation/operators/window.html)— suddividere periodicamente gli elementi da un Observable in finestre Observable ed emettere queste finestre anziché emettere gli elementi uno alla volta

## **Filtraggio degli osservabili**

Operatori che emettono selettivamente elementi da una fonte osservabile.

* [**Debounce**](https://reactivex.io/documentation/operators/debounce.html)- emette un oggetto da un osservabile solo se è trascorso un particolare periodo di tempo senza che esso abbia emesso un altro oggetto
* [**Distinct**](https://reactivex.io/documentation/operators/distinct.html)- sopprime gli elementi duplicati emessi da un osservabile
* [**ElementAt**](https://reactivex.io/documentation/operators/elementat.html)— emette solo l'elemento *n* emesso da un osservabile
* [**Filter**](https://reactivex.io/documentation/operators/filter.html)— emette solo quegli elementi da un osservabile che superano un test predicativo
* [**First**](https://reactivex.io/documentation/operators/first.html)— emette solo il primo elemento, o il primo elemento che soddisfa una condizione, da un osservabile
* [**IgnoreElements**](https://reactivex.io/documentation/operators/ignoreelements.html)— non emettono elementi da un oggetto osservabile ma rispecchiano la sua notifica di cessazione
* [**Last**](https://reactivex.io/documentation/operators/last.html)— emettono solo l'ultimo elemento emesso da un osservabile
* [**Sample**](https://reactivex.io/documentation/operators/sample.html)— emettere l'elemento più recente emesso da un osservabile entro intervalli di tempo periodici
* [**Skip**](https://reactivex.io/documentation/operators/skip.html)— sopprime i primi *n* elementi emessi da un osservabile
* [**SkipLast**](https://reactivex.io/documentation/operators/skiplast.html)— sopprime gli ultimi *n* elementi emessi da un osservabile
* [**Take**](https://reactivex.io/documentation/operators/take.html)— emettono solo i primi *n* elementi emessi da un osservabile
* [**TakeLast**](https://reactivex.io/documentation/operators/takelast.html)— emettono solo gli ultimi *n* elementi emessi da un osservabile

## **Combinazione di osservabili**

Operatori che lavorano con più osservabili sorgente per creare un singolo osservabile

* [**And/ Then/When**](https://reactivex.io/documentation/operators/and-then-when.html) — combina insiemi di elementi emessi da due o più osservabili tramitePatternePlanintermediari
* [**CombineLatest**](https://reactivex.io/documentation/operators/combinelatest.html)— quando un elemento viene emesso da uno dei due osservabili, combinare l'ultimo elemento emesso da ciascun osservabile tramite una funzione specifica ed emettere elementi in base ai risultati di questa funzione
* [**Join**](https://reactivex.io/documentation/operators/join.html)— combinare elementi emessi da due osservabili ogni volta che un elemento di un osservabile viene emesso durante una finestra temporale definita in base a un elemento emesso dall'altro osservabile
* [**Merge**](https://reactivex.io/documentation/operators/merge.html)— combinare più osservabili in uno solo unendo le loro emissioni
* [**StartWith**](https://reactivex.io/documentation/operators/startwith.html)— emettere una sequenza specifica di elementi prima di iniziare a emettere gli elementi dalla sorgente Observable
* [**Switch**](https://reactivex.io/documentation/operators/switch.html)— convertire un osservabile che emette osservabili in un singolo osservabile che emette gli elementi emessi dall'emesso più recentemente di tali osservabili
* [**Zip**](https://reactivex.io/documentation/operators/zip.html)— combinare insieme le emissioni di più osservabili tramite una funzione specifica ed emettere singoli elementi per ciascuna combinazione in base ai risultati di questa funzione

## **Operatori di gestione degli errori**

Operatori che aiutano a recuperare dalle notifiche di errore da un Observable

* [**Catch**](https://reactivex.io/documentation/operators/catch.html)— recuperare da una onErrornotifica continuando la sequenza senza errori
* [**Retry**](https://reactivex.io/documentation/operators/retry.html)- se una fonte Observable invia una onErrornotifica, iscriviti nuovamente ad essa nella speranza che venga completata senza errori

## **Operatori di utilità osservabili**

Una cassetta degli attrezzi di operatori utili per lavorare con gli osservabili

* [**Delay**](https://reactivex.io/documentation/operators/delay.html)— spostare le emissioni da un osservabile in avanti nel tempo di una determinata quantità
* [**Do**](https://reactivex.io/documentation/operators/do.html)- registrare un'azione per intraprendere una varietà di eventi osservabili del ciclo di vita
* [**Materialize/Dematerialize**](https://reactivex.io/documentation/operators/materialize-dematerialize.html) - rappresenta sia gli elementi emessi che le notifiche inviate come elementi emessi oppure inverte questo processo
* [**ObserveOn**](https://reactivex.io/documentation/operators/observeon.html)— specificare lo scheduler su cui un osservatore osserverà questo Observable
* [**Serialize**](https://reactivex.io/documentation/operators/serialize.html)- forzare un osservabile a effettuare chiamate serializzate e a comportarsi bene
* [**Subscribe**](https://reactivex.io/documentation/operators/subscribe.html)— operare sulle emissioni e sulle notifiche di un Osservabile
* [**SubscribeOn**](https://reactivex.io/documentation/operators/subscribeon.html)— specifica lo scheduler che un Observable dovrebbe utilizzare quando è iscritto
* [**TimeInterval**](https://reactivex.io/documentation/operators/timeinterval.html)— convertire un osservabile che emette elementi in uno che emette indicazioni sulla quantità di tempo trascorso tra tali emissioni
* [**Timeout**](https://reactivex.io/documentation/operators/timeout.html)— rispecchia la fonte Observable, ma emette una notifica di errore se trascorre un particolare periodo di tempo senza che vengano emessi elementi
* [**Timestamp**](https://reactivex.io/documentation/operators/timestamp.html)— allegare un timestamp a ciascun elemento emesso da un osservabile
* [**Using**](https://reactivex.io/documentation/operators/using.html)- creare una risorsa usa e getta che abbia la stessa durata di vita dell'Osservabile

## **Operatori condizionali e booleani**

Operatori che valutano uno o più Observable o elementi emessi dagli Observable

* [**All**](https://reactivex.io/documentation/operators/all.html)— determinare se tutti gli elementi emessi da un osservabile soddisfano alcuni criteri
* [**Amb**](https://reactivex.io/documentation/operators/amb.html)— dati due o più osservabili sorgente, emettere tutti gli elementi solo dal primo di questi osservabili per emettere un elemento
* [**Contains**](https://reactivex.io/documentation/operators/contains.html)— determinare se un oggetto osservabile emette o meno un particolare elemento
* [**DefaultIfEmpty**](https://reactivex.io/documentation/operators/defaultifempty.html)— emette elementi dall'Observable di origine o un elemento predefinito se l'Observable di origine non emette nulla
* [**SequenceEqual**](https://reactivex.io/documentation/operators/sequenceequal.html)— determinare se due Osservabili emettono la stessa sequenza di elementi
* [**SkipUntil**](https://reactivex.io/documentation/operators/skipuntil.html)— scarta gli oggetti emessi da un osservabile finché un secondo osservabile non emette un oggetto
* [**SkipWhile**](https://reactivex.io/documentation/operators/skipwhile.html)— scarta gli elementi emessi da un osservabile finché una condizione specificata non diventa falsa
* [**TakeUntil**](https://reactivex.io/documentation/operators/takeuntil.html)— scartare gli elementi emessi da un osservabile dopo che un secondo osservabile emette un elemento o termina
* [**TakeWhile**](https://reactivex.io/documentation/operators/takewhile.html)— scarta gli elementi emessi da un osservabile dopo che una condizione specificata diventa falsa

## **Operatori matematici e aggregati**

Operatori che operano sull'intera sequenza di elementi emessi da un Osservabile

* [**Average**](https://reactivex.io/documentation/operators/average.html)— calcola la media dei numeri emessi da un osservabile ed emette questa media
* [**Concat**](https://reactivex.io/documentation/operators/concat.html)— emettere le emissioni di due o più osservabili senza interlacciarli
* [**Count**](https://reactivex.io/documentation/operators/count.html)— conta il numero di elementi emessi dalla sorgente Observable ed emette solo questo valore
* [**Max**](https://reactivex.io/documentation/operators/max.html)- determinare ed emettere l'oggetto dal valore massimo emesso da un osservabile
* [**Min**](https://reactivex.io/documentation/operators/min.html)- determinare ed emettere l'elemento di valore minimo emesso da un osservabile
* [**Reduce**](https://reactivex.io/documentation/operators/reduce.html)— applicare una funzione a ciascun elemento emesso da un osservabile, in sequenza, ed emettere il valore finale
* [**Sum**](https://reactivex.io/documentation/operators/sum.html)— calcolare la somma dei numeri emessi da un osservabile ed emettere questa somma

## **Operatori di contropressione**

* [**operatori di contropressione**](https://reactivex.io/documentation/operators/backpressure.html) : strategie per far fronte agli osservabili che producono oggetti più rapidamente di quanto i loro osservatori li consumano

## **Operatori osservabili collegabili**

Osservabili specializzati che hanno dinamiche di sottoscrizione controllate in modo più preciso

* [**Connect**](https://reactivex.io/documentation/operators/connect.html)- istruire un osservabile collegabile a iniziare a emettere elementi ai suoi abbonati
* [**Publish**](https://reactivex.io/documentation/operators/publish.html)— convertire un osservabile ordinario in un osservabile collegabile
* [**RefCount**](https://reactivex.io/documentation/operators/refcount.html)- fare in modo che un osservabile collegabile si comporti come un osservabile ordinario
* [**Replay**](https://reactivex.io/documentation/operators/replay.html)— garantire che tutti gli osservatori vedano la stessa sequenza di elementi emessi, anche se si iscrivono dopo che l'osservabile ha iniziato a emettere elementi

## **Operatori per convertire osservabili**

* [**To**](https://reactivex.io/documentation/operators/to.html)- convertire un osservabile in un altro oggetto o struttura dati

# Un albero decisionale di operatori osservabili

Questo albero può aiutarti a trovare l'operatore ReactiveX Observable che stai cercando.

**Voglio creare un nuovo osservabile**

che emette un particolare oggetto

[Just](https://reactivex.io/documentation/operators/just.html)

restituito da una funzione chiamata al momento della sottoscrizione

[Start](https://reactivex.io/documentation/operators/start.html)

che è stato restituito da un Action, Callable, Runnable, o qualcosa del genere, chiamato al momento dell'iscrizione

[From](https://reactivex.io/documentation/operators/from.html)

dopo un ritardo specificato

[Timer](https://reactivex.io/documentation/operators/timer.html)

che estrae le sue emissioni da un particolare Array, Iterable, o qualcosa del genere

[From](https://reactivex.io/documentation/operators/from.html)

recuperandolo da un futuro

[Start](https://reactivex.io/documentation/operators/start.html)

che trae la sua sequenza da un Futuro

[From](https://reactivex.io/documentation/operators/from.html)

che emette ripetutamente una sequenza di elementi

[Repeat](https://reactivex.io/documentation/operators/repeat.html)

da zero, con logica personalizzata

[Create](https://reactivex.io/documentation/operators/create.html)

per ogni osservatore che si iscrive

[Defer](https://reactivex.io/documentation/operators/defer.html)

che emette una sequenza di numeri interi

[Range](https://reactivex.io/documentation/operators/range.html)

a particolari intervalli di tempo

[Interval](https://reactivex.io/documentation/operators/interval.html)

dopo un ritardo specificato

[Timer](https://reactivex.io/documentation/operators/timer.html)

che si completa senza emettere elementi

[Empty](https://reactivex.io/documentation/operators/empty-never-throw.html)

questo non fa assolutamente nulla

[Never](https://reactivex.io/documentation/operators/empty-never-throw.html)

**Voglio creare un osservabile combinando altri osservabili**

ed emettere tutti gli elementi da tutti gli osservabili in qualunque ordine vengano ricevuti

[Merge](https://reactivex.io/documentation/operators/merge.html)

ed emettere tutti gli elementi da tutti gli osservabili, un osservabile alla volta

[Concat](https://reactivex.io/documentation/operators/concat.html)

combinando gli elementi di due o più osservabili in sequenza per ottenere nuovi elementi da emettere

ogni volta che ciascuno degli osservabili ha emesso un nuovo elemento

[[Zip](https://reactivex.io/documentation/operators/zip.html)](https://reactivex.io/documentation/operators/zip.html)

ogni volta che uno qualsiasi degli osservabili ha emesso un nuovo elemento

[[CombineLatest](https://reactivex.io/documentation/operators/combinelatest.html)](https://reactivex.io/documentation/operators/combinelatest.html)

ogni volta che un elemento viene emesso da un osservabile in una finestra definita da un elemento emesso da un altro

[Join](https://reactivex.io/documentation/operators/join.html)

per mezzo di Patterne Planintermediari

[[And/Then/When](https://reactivex.io/documentation/operators/and-then-when.html)](https://reactivex.io/documentation/operators/and-then-when.html)

ed emettere gli elementi solo da quelli emessi più recentemente tra quegli osservabili

[Switch](https://reactivex.io/documentation/operators/switch.html)

**Voglio emettere gli elementi da un osservabile dopo averli trasformati**

uno alla volta con una funzione

[[Map](https://reactivex.io/documentation/operators/map.html)](https://reactivex.io/documentation/operators/map.html)

emettendo tutti gli elementi emessi dai corrispondenti osservabili

[[FlatMap](https://reactivex.io/documentation/operators/flatmap.html)](https://reactivex.io/documentation/operators/flatmap.html)

un Observable alla volta, nell'ordine in cui vengono emessi

[ConcatMap](https://reactivex.io/documentation/operators/flatmap.html)

sulla base di tutti gli elementi che li hanno preceduti

[Scan](https://reactivex.io/documentation/operators/scan.html)

allegando loro un timestamp

[Timestamp](https://reactivex.io/documentation/operators/timestamp.html)

in un indicatore della quantità di tempo trascorso prima dell'emissione dell'elemento

[[TimeInterval](https://reactivex.io/documentation/operators/timeinterval.html)](https://reactivex.io/documentation/operators/timeinterval.html)

**Voglio spostare in avanti nel tempo gli elementi emessi da un osservabile prima di riemetterli**

[Delay](https://reactivex.io/documentation/operators/delay.html)

**Voglio trasformare elementi e notifiche da un osservabile in elementi e inviarli nuovamente**

avvolgendoli in Notification oggetti

[Materialize](https://reactivex.io/documentation/operators/materialize-dematerialize.html)

con cui posso poi scartare di nuovo

[Dematerialize](https://reactivex.io/documentation/operators/materialize-dematerialize.html)

**Voglio ignorare tutti gli elementi emessi da un osservabile e trasmettere solo la notifica di completamento/errore**

[[IgnoreElements](https://reactivex.io/documentation/operators/ignoreelements.html)](https://reactivex.io/documentation/operators/ignoreelements.html)

**Voglio rispecchiare un osservabile ma prefisso gli elementi nella sua sequenza**

[[StartWith](https://reactivex.io/documentation/operators/startwith.html)](https://reactivex.io/documentation/operators/startwith.html)

solo se la sua sequenza è vuota

[DefaultIfEmpty](https://reactivex.io/documentation/operators/defaultifempty.html)

**Voglio raccogliere elementi da un osservabile e reimmetterli come buffer di elementi**

[Buffer](https://reactivex.io/documentation/operators/buffer.html)

contenente solo gli ultimi elementi emessi

[TakeLastBuffer](https://reactivex.io/documentation/operators/takelast.html)

**Voglio dividere un osservabile in più osservabili**

[Window](https://reactivex.io/documentation/operators/window.html)

in modo che elementi simili finiscano sullo stesso osservabile

[[GroupBy](https://reactivex.io/documentation/operators/groupby.html)](https://reactivex.io/documentation/operators/groupby.html)

**Voglio recuperare un particolare elemento emesso da un osservabile:**

l'ultimo elemento emesso prima del completamento

[Last](https://reactivex.io/documentation/operators/last.html)

l'unico oggetto che ha emesso

[Single](https://reactivex.io/documentation/operators/first.html)

il primo elemento emesso

[First](https://reactivex.io/documentation/operators/first.html)

**Voglio riemettere solo determinati elementi da un osservabile**

filtrando quelli che non corrispondono a qualche predicato

[Filter](https://reactivex.io/documentation/operators/filter.html)

cioè solo il primo elemento

[First](https://reactivex.io/documentation/operators/first.html)

cioè, solo i primi elementi s

[Take](https://reactivex.io/documentation/operators/take.html)

cioè solo l'ultimo elemento

[Last](https://reactivex.io/documentation/operators/last.html)

cioè solo l'articolo *n*

[ElementAt](https://reactivex.io/documentation/operators/elementat.html)

ovvero solo gli elementi successivi ai primi

cioè dopo i primi *n* elementi

[Skip](https://reactivex.io/documentation/operators/skip.html)

cioè finché uno di questi elementi non corrisponde a un predicato

[[SkipWhile](https://reactivex.io/documentation/operators/skipwhile.html)](https://reactivex.io/documentation/operators/skipwhile.html)

cioè dopo un periodo di tempo iniziale

[Skip](https://reactivex.io/documentation/operators/skip.html)

cioè, dopo che un secondo Observable emette un oggetto

[[SkipUntil](https://reactivex.io/documentation/operators/skipuntil.html)](https://reactivex.io/documentation/operators/skipuntil.html)

ovvero quegli elementi tranne gli ultimi elementi

cioè, tranne gli ultimi *n* elementi

[SkipLast](https://reactivex.io/documentation/operators/skiplast.html)

cioè finché uno di questi elementi non corrisponde a un predicato

[TakeWhile](https://reactivex.io/documentation/operators/takewhile.html)

cioè, ad eccezione degli elementi emessi durante un periodo di tempo prima del completamento della sorgente

[SkipLast](https://reactivex.io/documentation/operators/skiplast.html)

cioè, ad eccezione degli elementi emessi dopo che un secondo osservabile emette un elemento

[TakeUntil](https://reactivex.io/documentation/operators/takeuntil.html)

campionando periodicamente l'osservabile

[Sample](https://reactivex.io/documentation/operators/sample.html)

emettendo solo elementi che non sono seguiti da altri elementi entro una certa durata

[Debounce](https://reactivex.io/documentation/operators/debounce.html)

sopprimendo gli elementi che sono duplicati di elementi già emessi

[Distinct](https://reactivex.io/documentation/operators/distinct.html)

se seguono immediatamente l'elemento di cui sono duplicati

[[DistinctUntilChanged](https://reactivex.io/documentation/operators/distinct.html)](https://reactivex.io/documentation/operators/distinct.html)

ritardando il mio abbonamento per un po' di tempo dopo che ha iniziato a emettere articoli

[DelaySubscription](https://reactivex.io/documentation/operators/delay.html)

**Voglio riemettere elementi da un osservabile solo a condizione che sia stato il primo di una raccolta di osservabili a emettere un elemento**

[Amb](https://reactivex.io/documentation/operators/amb.html)

**Voglio valutare l'intera sequenza di elementi emessi da un osservabile**

ed emette un singolo booleano che indica se tutti gli elementi superano qualche test

[All](https://reactivex.io/documentation/operators/all.html)

ed emette un singolo booleano che indica se l'osservabile ha emesso qualche elemento (che supera qualche test)

[Contains](https://reactivex.io/documentation/operators/contains.html)

ed emette un singolo booleano che indica se l'Observable non ha emesso alcun elemento

[IsEmpty](https://reactivex.io/documentation/operators/contains.html)

ed emettono un singolo booleano che indica se la sequenza è identica a quella emessa da un secondo Observable

[SequenceEqual](https://reactivex.io/documentation/operators/sequenceequal.html)

ed emettono la media di tutti i loro valori

[Average](https://reactivex.io/documentation/operators/average.html)

ed emettono la somma di tutti i loro valori

[Sum](https://reactivex.io/documentation/operators/sum.html)

ed emette un numero che indica quanti elementi c'erano nella sequenza

[Count](https://reactivex.io/documentation/operators/count.html)

ed emettere l'oggetto con il valore massimo

[Max](https://reactivex.io/documentation/operators/max.html)

ed emettere l'elemento con il valore minimo

[Min](https://reactivex.io/documentation/operators/min.html)

applicando una funzione di aggregazione a ciascun elemento a turno ed emettendo il risultato

[Scan](https://reactivex.io/documentation/operators/scan.html)

**Voglio convertire l'intera sequenza di elementi emessi da un osservabile in qualche altra struttura dati**

[To](https://reactivex.io/documentation/operators/to.html)

**Voglio che un operatore operi su un particolare**[**Scheduler**](https://reactivex.io/scheduler.html)

[SubscribeOn](https://reactivex.io/documentation/operators/subscribeon.html)

quando avvisa gli osservatori

[ObserveOn](https://reactivex.io/documentation/operators/observeon.html)

**Voglio che un osservabile invochi un'azione particolare quando si verificano determinati eventi**

[Do](https://reactivex.io/documentation/operators/do.html)

**Voglio un osservabile che notificherà agli osservatori un errore**

[Throw](https://reactivex.io/documentation/operators/empty-never-throw.html)

se trascorre un periodo di tempo specificato senza che venga emesso un elemento

[[Timeout](https://reactivex.io/documentation/operators/timeout.html)](https://reactivex.io/documentation/operators/timeout.html)

**Voglio che un osservabile si riprenda con grazia**

da un timeout passando a un Observable di backup

[[Timeout](https://reactivex.io/documentation/operators/timeout.html)](https://reactivex.io/documentation/operators/timeout.html)

da una notifica di errore a monte

[Catch](https://reactivex.io/documentation/operators/catch.html)

tentando di iscriversi nuovamente all'Observable upstream

[Retry](https://reactivex.io/documentation/operators/retry.html)

**Voglio creare una risorsa che abbia la stessa durata di vita dell'Osservabile**

[Using](https://reactivex.io/documentation/operators/using.html)

**Voglio iscrivermi a un Observable e ricevere un Futureche si blocca fino al completamento dell'Observable**

[Start](https://reactivex.io/documentation/operators/start.html)

**Voglio un osservabile che non inizi a emettere elementi agli abbonati finché non viene richiesto**

[Publish](https://reactivex.io/documentation/operators/publish.html)

e quindi emette solo l'ultimo elemento nella sua sequenza

[PublishLast](https://reactivex.io/documentation/operators/publish.html)

e poi emette la sequenza completa, anche a chi si iscrive a sequenza iniziata

[Replay](https://reactivex.io/documentation/operators/replay.html)

ma voglio che scompaia una volta che tutti i suoi abbonati annullano l'iscrizione

[[RefCount](https://reactivex.io/documentation/operators/refcount.html)](https://reactivex.io/documentation/operators/refcount.html)

e poi voglio chiedergli di iniziare

[Connect](https://reactivex.io/documentation/operators/connect.html)

#### **Guarda anche**

* [[Which Operator do I use?](http://xgrommx.github.io/rx-book/content/which_operator_do_i_use/index.html)](http://xgrommx.github.io/rx-book/content/which_operator_do_i_use/index.html)di Dennis Stoyanov (un albero decisionale simile, specifico per gli operatori RxJS)

**An Alphabetical List of Observable Operators**

Canonical, core operator names are in **boldface**. Other entries represent language-specific variants of these operators or specialty operators outside of the main ReactiveX core set of operators.

* [Aggregate](https://reactivex.io/documentation/operators/reduce.html)
* [**All**](https://reactivex.io/documentation/operators/all.html)
* [**Amb**](https://reactivex.io/documentation/operators/amb.html)
* [ambArray](https://reactivex.io/documentation/operators/amb.html)
* [ambWith](https://reactivex.io/documentation/operators/amb.html)
* [and\_](https://reactivex.io/documentation/operators/and-then-when.html)
* [**And**](https://reactivex.io/documentation/operators/and-then-when.html)
* [Any](https://reactivex.io/documentation/operators/contains.html)
* [apply](https://reactivex.io/documentation/operators/create.html)
* [as\_blocking](https://reactivex.io/documentation/operators/to.html)
* [asObservable](https://reactivex.io/documentation/operators/from.html)
* [AssertEqual](https://reactivex.io/documentation/operators/sequenceequal.html)
* [asyncAction](https://reactivex.io/documentation/operators/start.html)
* [asyncFunc](https://reactivex.io/documentation/operators/start.html)
* [**Average**](https://reactivex.io/documentation/operators/average.html)
* [averageDouble](https://reactivex.io/documentation/operators/average.html)
* [averageFloat](https://reactivex.io/documentation/operators/average.html)
* [averageInteger](https://reactivex.io/documentation/operators/average.html)
* [averageLong](https://reactivex.io/documentation/operators/average.html)
* [blocking](https://reactivex.io/documentation/operators/to.html)
* [blockingFirst](https://reactivex.io/documentation/operators/first.html)
* [blockingForEach](https://reactivex.io/documentation/operators/subscribe.html)
* [blockingIterable](https://reactivex.io/documentation/operators/to.html)
* [blockingLast](https://reactivex.io/documentation/operators/last.html)
* [blockingLatest](https://reactivex.io/documentation/operators/to.html)
* [blockingMostRecent](https://reactivex.io/documentation/operators/to.html)
* [blockingNext](https://reactivex.io/documentation/operators/to.html)
* [blockingSingle](https://reactivex.io/documentation/operators/first.html)
* [blockingSubscribe](https://reactivex.io/documentation/operators/subscribe.html)
* [**Buffer**](https://reactivex.io/documentation/operators/buffer.html)
* [bufferWithCount](https://reactivex.io/documentation/operators/buffer.html)
* [bufferWithTime](https://reactivex.io/documentation/operators/buffer.html)
* [bufferWithTimeOrCount](https://reactivex.io/documentation/operators/buffer.html)
* [byLine](https://reactivex.io/documentation/operators/map.html)
* [cache](https://reactivex.io/documentation/operators/replay.html)
* [cacheWithInitialCapacity](https://reactivex.io/documentation/operators/replay.html)
* [case](https://reactivex.io/documentation/operators/defer.html)
* [Cast](https://reactivex.io/documentation/operators/map.html)
* [**Catch**](https://reactivex.io/documentation/operators/catch.html)
* [catchError](https://reactivex.io/documentation/operators/catch.html)
* [catchException](https://reactivex.io/documentation/operators/catch.html)
* [collect](https://reactivex.io/documentation/operators/reduce.html)
* [collect](https://reactivex.io/documentation/operators/filter.html) (RxScala version of **Filter**)
* [collectInto](https://reactivex.io/documentation/operators/reduce.html)
* [**CombineLatest**](https://reactivex.io/documentation/operators/combinelatest.html)
* [combineLatestDelayError](https://reactivex.io/documentation/operators/combinelatest.html)
* [combineLatestWith](https://reactivex.io/documentation/operators/combinelatest.html)
* [**Concat**](https://reactivex.io/documentation/operators/concat.html)
* [concat\_all](https://reactivex.io/documentation/operators/flatmap.html)
* [concatAll](https://reactivex.io/documentation/operators/concat.html)
* [concatArray](https://reactivex.io/documentation/operators/concat.html)
* [concatArrayDelayError](https://reactivex.io/documentation/operators/concat.html)
* [concatArrayEager](https://reactivex.io/documentation/operators/concat.html)
* [concatDelayError](https://reactivex.io/documentation/operators/concat.html)
* [concatEager](https://reactivex.io/documentation/operators/concat.html)
* [concatMap](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapDelayError](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapEager](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapEagerDelayError](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapIterable](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapObserver](https://reactivex.io/documentation/operators/flatmap.html)
* [concatMapTo](https://reactivex.io/documentation/operators/flatmap.html)
* [concatWith](https://reactivex.io/documentation/operators/concat.html)
* [**Connect**](https://reactivex.io/documentation/operators/connect.html)
* [connect\_forever](https://reactivex.io/documentation/operators/connect.html)
* [cons](https://reactivex.io/documentation/operators/startwith.html)
* [**Contains**](https://reactivex.io/documentation/operators/contains.html)
* [controlled](https://reactivex.io/documentation/operators/backpressure.html)
* [**Count**](https://reactivex.io/documentation/operators/count.html)
* [countLong](https://reactivex.io/documentation/operators/count.html)
* [**Create**](https://reactivex.io/documentation/operators/create.html)
* [cycle](https://reactivex.io/documentation/operators/repeat.html)
* [**Debounce**](https://reactivex.io/documentation/operators/debounce.html)
* [decode](https://reactivex.io/documentation/operators/from.html)
* [**DefaultIfEmpty**](https://reactivex.io/documentation/operators/defaultifempty.html)
* [**Defer**](https://reactivex.io/documentation/operators/defer.html)
* [deferFuture](https://reactivex.io/documentation/operators/start.html)
* [**Delay**](https://reactivex.io/documentation/operators/delay.html)
* [delaySubscription](https://reactivex.io/documentation/operators/delay.html)
* [delayWithSelector](https://reactivex.io/documentation/operators/delay.html)
* [**Dematerialize**](https://reactivex.io/documentation/operators/materialize-dematerialize.html)
* [**Distinct**](https://reactivex.io/documentation/operators/distinct.html)
* [distinctKey](https://reactivex.io/documentation/operators/distinct.html)
* [distinctUntilChanged](https://reactivex.io/documentation/operators/distinct.html)
* [distinctUntilKeyChanged](https://reactivex.io/documentation/operators/distinct.html)
* [**Do**](https://reactivex.io/documentation/operators/do.html)
* [doAction](https://reactivex.io/documentation/operators/do.html)
* [doAfterTerminate](https://reactivex.io/documentation/operators/do.html)
* [doOnComplete](https://reactivex.io/documentation/operators/do.html)
* [doOnCompleted](https://reactivex.io/documentation/operators/do.html)
* [doOnDispose](https://reactivex.io/documentation/operators/do.html)
* [doOnEach](https://reactivex.io/documentation/operators/do.html)
* [doOnError](https://reactivex.io/documentation/operators/do.html)
* [doOnLifecycle](https://reactivex.io/documentation/operators/do.html)
* [doOnNext](https://reactivex.io/documentation/operators/do.html)
* [doOnRequest](https://reactivex.io/documentation/operators/do.html)
* [doOnSubscribe](https://reactivex.io/documentation/operators/do.html)
* [doOnTerminate](https://reactivex.io/documentation/operators/do.html)
* [doOnUnsubscribe](https://reactivex.io/documentation/operators/do.html)
* [doseq](https://reactivex.io/documentation/operators/subscribe.html)
* [doWhile](https://reactivex.io/documentation/operators/repeat.html)
* [drop](https://reactivex.io/documentation/operators/skip.html)
* [dropRight](https://reactivex.io/documentation/operators/skiplast.html)
* [dropUntil](https://reactivex.io/documentation/operators/skipuntil.html)
* [dropWhile](https://reactivex.io/documentation/operators/skipwhile.html)
* [**ElementAt**](https://reactivex.io/documentation/operators/elementat.html)
* [ElementAtOrDefault](https://reactivex.io/documentation/operators/elementat.html)
* [**Empty**](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [emptyObservable](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [empty?](https://reactivex.io/documentation/operators/contains.html)
* [encode](https://reactivex.io/documentation/operators/map.html)
* [ensures](https://reactivex.io/documentation/operators/do.html)
* [error](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [every](https://reactivex.io/documentation/operators/all.html)
* [exclusive](https://reactivex.io/documentation/operators/switch.html)
* [exists](https://reactivex.io/documentation/operators/contains.html)
* [expand](https://reactivex.io/documentation/operators/flatmap.html)
* [failWith](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [**Filter**](https://reactivex.io/documentation/operators/filter.html)
* [filterNot](https://reactivex.io/documentation/operators/filter.html)
* [Finally](https://reactivex.io/documentation/operators/do.html)
* [finallyAction](https://reactivex.io/documentation/operators/do.html)
* [finallyDo](https://reactivex.io/documentation/operators/do.html)
* [find](https://reactivex.io/documentation/operators/contains.html)
* [findIndex](https://reactivex.io/documentation/operators/contains.html)
* [**First**](https://reactivex.io/documentation/operators/first.html)
* [firstElement](https://reactivex.io/documentation/operators/first.html)
* [FirstOrDefault](https://reactivex.io/documentation/operators/first.html)
* [firstOrElse](https://reactivex.io/documentation/operators/first.html)
* [**FlatMap**](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapFirst](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapIterable](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapIterableWith](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapLatest](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapObserver](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapWith](https://reactivex.io/documentation/operators/flatmap.html)
* [flatMapWithMaxConcurrent](https://reactivex.io/documentation/operators/flatmap.html)
* [flat\_map\_with\_index](https://reactivex.io/documentation/operators/flatmap.html)
* [flatten](https://reactivex.io/documentation/operators/merge.html)
* [flattenDelayError](https://reactivex.io/documentation/operators/merge.html)
* [foldl](https://reactivex.io/documentation/operators/reduce.html)
* [foldLeft](https://reactivex.io/documentation/operators/reduce.html)
* [for](https://reactivex.io/documentation/operators/flatmap.html)
* [forall](https://reactivex.io/documentation/operators/all.html)
* [ForEach](https://reactivex.io/documentation/operators/subscribe.html)
* [forEachFuture](https://reactivex.io/documentation/operators/start.html)
* [forEachWhile](https://reactivex.io/documentation/operators/subscribe.html)
* [forIn](https://reactivex.io/documentation/operators/flatmap.html)
* [forkJoin](https://reactivex.io/documentation/operators/zip.html)
* [**From**](https://reactivex.io/documentation/operators/from.html)
* [fromAction](https://reactivex.io/documentation/operators/from.html)
* [fromArray](https://reactivex.io/documentation/operators/from.html)
* [FromAsyncPattern](https://reactivex.io/documentation/operators/from.html)
* [fromCallable](https://reactivex.io/documentation/operators/from.html)
* [fromCallback](https://reactivex.io/documentation/operators/from.html)
* [FromEvent](https://reactivex.io/documentation/operators/from.html)
* [FromEventPattern](https://reactivex.io/documentation/operators/from.html)
* [fromFunc0](https://reactivex.io/documentation/operators/from.html)
* [fromFuture](https://reactivex.io/documentation/operators/from.html)
* [fromIterable](https://reactivex.io/documentation/operators/from.html)
* [fromIterator](https://reactivex.io/documentation/operators/from.html)
* [from\_list](https://reactivex.io/documentation/operators/from.html)
* [fromNodeCallback](https://reactivex.io/documentation/operators/from.html)
* [fromPromise](https://reactivex.io/documentation/operators/from.html)
* [fromPublisher](https://reactivex.io/documentation/operators/from.html)
* [fromRunnable](https://reactivex.io/documentation/operators/from.html)
* [Generate](https://reactivex.io/documentation/operators/create.html)
* [generateWithAbsoluteTime](https://reactivex.io/documentation/operators/create.html)
* [generateWithRelativeTime](https://reactivex.io/documentation/operators/create.html)
* [generator](https://reactivex.io/documentation/operators/create.html)
* [GetEnumerator](https://reactivex.io/documentation/operators/to.html)
* [getIterator](https://reactivex.io/documentation/operators/to.html)
* [**GroupBy**](https://reactivex.io/documentation/operators/groupby.html)
* [GroupByUntil](https://reactivex.io/documentation/operators/groupby.html)
* [GroupJoin](https://reactivex.io/documentation/operators/join.html)
* [head](https://reactivex.io/documentation/operators/first.html)
* [headOption](https://reactivex.io/documentation/operators/first.html)
* [headOrElse](https://reactivex.io/documentation/operators/first.html)
* [if](https://reactivex.io/documentation/operators/defer.html)
* [ifThen](https://reactivex.io/documentation/operators/defer.html)
* [**IgnoreElements**](https://reactivex.io/documentation/operators/ignoreelements.html)
* [indexOf](https://reactivex.io/documentation/operators/contains.html)
* [interleave](https://reactivex.io/documentation/operators/merge.html)
* [interpose](https://reactivex.io/documentation/operators/to.html)
* [**Interval**](https://reactivex.io/documentation/operators/interval.html)
* [intervalRange](https://reactivex.io/documentation/operators/range.html)
* [into](https://reactivex.io/documentation/operators/reduce.html)
* [isEmpty](https://reactivex.io/documentation/operators/contains.html)
* [items](https://reactivex.io/documentation/operators/just.html)
* [**Join**](https://reactivex.io/documentation/operators/join.html)
* [join](https://reactivex.io/documentation/operators/sum.html) (string)
* [jortSort](https://reactivex.io/documentation/operators/all.html)
* [jortSortUntil](https://reactivex.io/documentation/operators/all.html)
* [**Just**](https://reactivex.io/documentation/operators/just.html)
* [keep](https://reactivex.io/documentation/operators/map.html)
* [keep-indexed](https://reactivex.io/documentation/operators/map.html)
* [**Last**](https://reactivex.io/documentation/operators/last.html)
* [lastElement](https://reactivex.io/documentation/operators/last.html)
* [lastOption](https://reactivex.io/documentation/operators/last.html)
* [LastOrDefault](https://reactivex.io/documentation/operators/last.html)
* [lastOrElse](https://reactivex.io/documentation/operators/last.html)
* [Latest](https://reactivex.io/documentation/operators/first.html)
* [latest](https://reactivex.io/documentation/operators/switch.html) (Rx.rb version of **Switch**)
* [length](https://reactivex.io/documentation/operators/count.html)
* [let](https://reactivex.io/documentation/operators/publish.html)
* [letBind](https://reactivex.io/documentation/operators/publish.html)
* [lift](https://reactivex.io/documentation/implement-operator.html)
* [limit](https://reactivex.io/documentation/operators/take.html)
* [LongCount](https://reactivex.io/documentation/operators/count.html)
* [ManySelect](https://reactivex.io/documentation/operators/flatmap.html)
* [**Map**](https://reactivex.io/documentation/operators/map.html)
* [map](https://reactivex.io/documentation/operators/zip.html) (RxClojure version of **Zip**)
* [MapCat](https://reactivex.io/documentation/operators/flatmap.html)
* [mapCat](https://reactivex.io/documentation/operators/zip.html) (RxClojure version of **Zip**)
* [map-indexed](https://reactivex.io/documentation/operators/map.html)
* [mapTo](https://reactivex.io/documentation/operators/map.html)
* [mapWithIndex](https://reactivex.io/documentation/operators/map.html)
* [**Materialize**](https://reactivex.io/documentation/operators/materialize-dematerialize.html)
* [**Max**](https://reactivex.io/documentation/operators/max.html)
* [MaxBy](https://reactivex.io/documentation/operators/max.html)
* [**Merge**](https://reactivex.io/documentation/operators/merge.html)
* [mergeAll](https://reactivex.io/documentation/operators/merge.html)
* [mergeArray](https://reactivex.io/documentation/operators/merge.html)
* [mergeArrayDelayError](https://reactivex.io/documentation/operators/merge.html)
* [merge\_concurrent](https://reactivex.io/documentation/operators/merge.html)
* [mergeDelayError](https://reactivex.io/documentation/operators/merge.html)
* [mergeObservable](https://reactivex.io/documentation/operators/merge.html)
* [mergeWith](https://reactivex.io/documentation/operators/merge.html)
* [**Min**](https://reactivex.io/documentation/operators/min.html)
* [MinBy](https://reactivex.io/documentation/operators/min.html)
* [MostRecent](https://reactivex.io/documentation/operators/first.html)
* [Multicast](https://reactivex.io/documentation/operators/publish.html)
* [multicastWithSelector](https://reactivex.io/documentation/operators/publish.html)
* [nest](https://reactivex.io/documentation/operators/to.html)
* [**Never**](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [Next](https://reactivex.io/documentation/operators/takelast.html)
* [Next](https://reactivex.io/documentation/operators/first.html) (BlockingObservable version)
* [none](https://reactivex.io/documentation/operators/contains.html)
* [nonEmpty](https://reactivex.io/documentation/operators/contains.html)
* [nth](https://reactivex.io/documentation/operators/elementat.html)
* [**ObserveOn**](https://reactivex.io/documentation/operators/observeon.html)
* [ObserveOnDispatcher](https://reactivex.io/documentation/operators/observeon.html)
* [observeSingleOn](https://reactivex.io/documentation/operators/observeon.html)
* [of](https://reactivex.io/documentation/operators/from.html)
* [of\_array](https://reactivex.io/documentation/operators/from.html)
* [ofArrayChanges](https://reactivex.io/documentation/operators/from.html)
* [of\_enumerable](https://reactivex.io/documentation/operators/from.html)
* [of\_enumerator](https://reactivex.io/documentation/operators/from.html)
* [ofObjectChanges](https://reactivex.io/documentation/operators/from.html)
* [OfType](https://reactivex.io/documentation/operators/filter.html)
* [ofWithScheduler](https://reactivex.io/documentation/operators/from.html)
* [onBackpressureBlock](https://reactivex.io/documentation/operators/backpressure.html)
* [onBackpressureBuffer](https://reactivex.io/documentation/operators/backpressure.html)
* [onBackpressureDrop](https://reactivex.io/documentation/operators/backpressure.html)
* [OnErrorResumeNext](https://reactivex.io/documentation/operators/catch.html)
* [onErrorReturn](https://reactivex.io/documentation/operators/catch.html)
* [onErrorReturnItem](https://reactivex.io/documentation/operators/catch.html)
* [onExceptionResumeNext](https://reactivex.io/documentation/operators/catch.html)
* [onTerminateDetach](https://reactivex.io/documentation/operators/do.html)
* [orElse](https://reactivex.io/documentation/operators/defaultifempty.html)
* [pairs](https://reactivex.io/documentation/operators/from.html)
* [pairwise](https://reactivex.io/documentation/operators/buffer.html)
* [partition](https://reactivex.io/documentation/operators/groupby.html)
* [partition-all](https://reactivex.io/documentation/operators/window.html)
* [pausable](https://reactivex.io/documentation/operators/backpressure.html)
* [pausableBuffered](https://reactivex.io/documentation/operators/backpressure.html)
* [pluck](https://reactivex.io/documentation/operators/map.html)
* [product](https://reactivex.io/documentation/operators/sum.html)
* [**Publish**](https://reactivex.io/documentation/operators/publish.html)
* [PublishLast](https://reactivex.io/documentation/operators/publish.html)
* [publish\_synchronized](https://reactivex.io/documentation/operators/replay.html)
* [publishValue](https://reactivex.io/documentation/operators/publish.html)
* [raise\_error](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [**Range**](https://reactivex.io/documentation/operators/range.html)
* [**Reduce**](https://reactivex.io/documentation/operators/reduce.html)
* [reduceWith](https://reactivex.io/documentation/operators/reduce.html)
* [reductions](https://reactivex.io/documentation/operators/scan.html)
* [**RefCount**](https://reactivex.io/documentation/operators/refcount.html)
* [**Repeat**](https://reactivex.io/documentation/operators/repeat.html)
* [repeat\_infinitely](https://reactivex.io/documentation/operators/repeat.html)
* [repeatUntil](https://reactivex.io/documentation/operators/repeat.html)
* [repeatWhen](https://reactivex.io/documentation/operators/repeat.html)
* [**Replay**](https://reactivex.io/documentation/operators/replay.html)
* [rescue\_error](https://reactivex.io/documentation/operators/catch.html)
* [rest](https://reactivex.io/documentation/operators/first.html)
* [**Retry**](https://reactivex.io/documentation/operators/retry.html)
* [retry\_infinitely](https://reactivex.io/documentation/operators/retry.html)
* [retryUntil](https://reactivex.io/documentation/operators/retry.html)
* [retryWhen](https://reactivex.io/documentation/operators/retry.html)
* [Return](https://reactivex.io/documentation/operators/just.html)
* [returnElement](https://reactivex.io/documentation/operators/just.html)
* [returnValue](https://reactivex.io/documentation/operators/just.html)
* [runAsync](https://reactivex.io/documentation/operators/from.html)
* [safeSubscribe](https://reactivex.io/documentation/operators/subscribe.html)
* [**Sample**](https://reactivex.io/documentation/operators/sample.html)
* [**Scan**](https://reactivex.io/documentation/operators/scan.html)
* [scanWith](https://reactivex.io/documentation/operators/scan.html)
* [scope](https://reactivex.io/documentation/operators/using.html)
* [Select](https://reactivex.io/documentation/operators/map.html) (alternate name of **Map**)
* [select](https://reactivex.io/documentation/operators/filter.html) (alternate name of **Filter**)
* [selectConcat](https://reactivex.io/documentation/operators/flatmap.html)
* [selectConcatObserver](https://reactivex.io/documentation/operators/flatmap.html)
* [SelectMany](https://reactivex.io/documentation/operators/flatmap.html)
* [selectManyObserver](https://reactivex.io/documentation/operators/flatmap.html)
* [select\_switch](https://reactivex.io/documentation/operators/switch.html)
* [selectSwitch](https://reactivex.io/documentation/operators/flatmap.html)
* [selectSwitchFirst](https://reactivex.io/documentation/operators/flatmap.html)
* [selectWithMaxConcurrent](https://reactivex.io/documentation/operators/flatmap.html)
* [select\_with\_index](https://reactivex.io/documentation/operators/filter.html)
* [seq](https://reactivex.io/documentation/operators/from.html)
* [**SequenceEqual**](https://reactivex.io/documentation/operators/sequenceequal.html)
* [sequence\_eql?](https://reactivex.io/documentation/operators/sequenceequal.html)
* [SequenceEqualWith](https://reactivex.io/documentation/operators/sequenceequal.html)
* [**Serialize**](https://reactivex.io/documentation/operators/serialize.html)
* [share](https://reactivex.io/documentation/operators/refcount.html)
* [shareReplay](https://reactivex.io/documentation/operators/replay.html)
* [shareValue](https://reactivex.io/documentation/operators/refcount.html)
* [Single](https://reactivex.io/documentation/operators/first.html)
* [singleElement](https://reactivex.io/documentation/operators/first.html)
* [SingleOrDefault](https://reactivex.io/documentation/operators/first.html)
* [singleOption](https://reactivex.io/documentation/operators/first.html)
* [singleOrElse](https://reactivex.io/documentation/operators/first.html)
* [size](https://reactivex.io/documentation/operators/count.html)
* [**Skip**](https://reactivex.io/documentation/operators/skip.html)
* [**SkipLast**](https://reactivex.io/documentation/operators/skiplast.html)
* [skipLastWithTime](https://reactivex.io/documentation/operators/skiplast.html)
* [**SkipUntil**](https://reactivex.io/documentation/operators/skipuntil.html)
* [skipUntilWithTime](https://reactivex.io/documentation/operators/skipuntil.html)
* [**SkipWhile**](https://reactivex.io/documentation/operators/skipwhile.html)
* [skipWhileWithIndex](https://reactivex.io/documentation/operators/skipwhile.html)
* [skip\_with\_time](https://reactivex.io/documentation/operators/skip.html)
* [slice](https://reactivex.io/documentation/operators/filter.html)
* [sliding](https://reactivex.io/documentation/operators/window.html)
* [slidingBuffer](https://reactivex.io/documentation/operators/buffer.html)
* [some](https://reactivex.io/documentation/operators/contains.html)
* [sort](https://reactivex.io/documentation/operators/to.html)
* [sorted](https://reactivex.io/documentation/operators/to.html)
* [sort-by](https://reactivex.io/documentation/operators/to.html)
* [sorted-list-by](https://reactivex.io/documentation/operators/to.html)
* [split](https://reactivex.io/documentation/operators/flatmap.html)
* [split-with](https://reactivex.io/documentation/operators/groupby.html)
* [**Start**](https://reactivex.io/documentation/operators/start.html)
* [startAsync](https://reactivex.io/documentation/operators/start.html)
* [startFuture](https://reactivex.io/documentation/operators/start.html)
* [**StartWith**](https://reactivex.io/documentation/operators/startwith.html)
* [startWithArray](https://reactivex.io/documentation/operators/startwith.html)
* [stringConcat](https://reactivex.io/documentation/operators/sum.html)
* [stopAndWait](https://reactivex.io/documentation/operators/backpressure.html)
* [subscribe](https://reactivex.io/documentation/operators/subscribe.html)
* [subscribeActual](https://reactivex.io/documentation/operators/subscribe.html)
* [**SubscribeOn**](https://reactivex.io/documentation/operators/subscribeon.html)
* [SubscribeOnDispatcher](https://reactivex.io/documentation/operators/subscribeon.html)
* [subscribeOnCompleted](https://reactivex.io/documentation/operators/subscribe.html)
* [subscribeOnError](https://reactivex.io/documentation/operators/subscribe.html)
* [subscribeOnNext](https://reactivex.io/documentation/operators/subscribe.html)
* [subscribeWith](https://reactivex.io/documentation/operators/subscribe.html)
* [**Sum**](https://reactivex.io/documentation/operators/sum.html)
* [sumDouble](https://reactivex.io/documentation/operators/sum.html)
* [sumFloat](https://reactivex.io/documentation/operators/sum.html)
* [sumInteger](https://reactivex.io/documentation/operators/sum.html)
* [sumLong](https://reactivex.io/documentation/operators/sum.html)
* [**Switch**](https://reactivex.io/documentation/operators/switch.html)
* [switchCase](https://reactivex.io/documentation/operators/defer.html)
* [switchIfEmpty](https://reactivex.io/documentation/operators/defaultifempty.html)
* [switchLatest](https://reactivex.io/documentation/operators/switch.html)
* [switchMap](https://reactivex.io/documentation/operators/flatmap.html)
* [switchMapDelayError](https://reactivex.io/documentation/operators/flatmap.html)
* [switchOnNext](https://reactivex.io/documentation/operators/switch.html)
* [switchOnNextDelayError](https://reactivex.io/documentation/operators/switch.html)
* [Synchronize](https://reactivex.io/documentation/operators/serialize.html)
* [**Take**](https://reactivex.io/documentation/operators/take.html)
* [take\_with\_time](https://reactivex.io/documentation/operators/take.html)
* [takeFirst](https://reactivex.io/documentation/operators/first.html)
* [**TakeLast**](https://reactivex.io/documentation/operators/takelast.html)
* [takeLastBuffer](https://reactivex.io/documentation/operators/takelast.html)
* [takeLastBufferWithTime](https://reactivex.io/documentation/operators/takelast.html)
* [takeLastWithTime](https://reactivex.io/documentation/operators/takelast.html)
* [takeRight](https://reactivex.io/documentation/operators/last.html) (see also: [**TakeLast**](https://reactivex.io/documentation/operators/takelast.html))
* [**TakeUntil**](https://reactivex.io/documentation/operators/takeuntil.html)
* [takeUntilWithTime](https://reactivex.io/documentation/operators/takeuntil.html)
* [**TakeWhile**](https://reactivex.io/documentation/operators/takewhile.html)
* [takeWhileWithIndex](https://reactivex.io/documentation/operators/takewhile.html)
* [tail](https://reactivex.io/documentation/operators/takelast.html)
* [tap](https://reactivex.io/documentation/operators/do.html)
* [tapOnCompleted](https://reactivex.io/documentation/operators/do.html)
* [tapOnError](https://reactivex.io/documentation/operators/do.html)
* [tapOnNext](https://reactivex.io/documentation/operators/do.html)
* [**Then**](https://reactivex.io/documentation/operators/and-then-when.html)
* [thenDo](https://reactivex.io/documentation/operators/and-then-when.html)
* [Throttle](https://reactivex.io/documentation/operators/debounce.html)
* [throttleFirst](https://reactivex.io/documentation/operators/sample.html)
* [throttleLast](https://reactivex.io/documentation/operators/sample.html)
* [throttleWithSelector](https://reactivex.io/documentation/operators/debounce.html)
* [throttleWithTimeout](https://reactivex.io/documentation/operators/debounce.html)
* [**Throw**](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [throwError](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [throwException](https://reactivex.io/documentation/operators/empty-never-throw.html)
* [**TimeInterval**](https://reactivex.io/documentation/operators/timeinterval.html)
* [**Timeout**](https://reactivex.io/documentation/operators/timeout.html)
* [timeoutWithSelector](https://reactivex.io/documentation/operators/timeout.html)
* [**Timer**](https://reactivex.io/documentation/operators/timer.html)
* [**Timestamp**](https://reactivex.io/documentation/operators/timestamp.html)
* [**To**](https://reactivex.io/documentation/operators/to.html)
* [to\_a](https://reactivex.io/documentation/operators/to.html)
* [ToArray](https://reactivex.io/documentation/operators/to.html)
* [ToAsync](https://reactivex.io/documentation/operators/start.html)
* [toBlocking](https://reactivex.io/documentation/operators/to.html)
* [toBuffer](https://reactivex.io/documentation/operators/to.html)
* [to\_dict](https://reactivex.io/documentation/operators/to.html)
* [ToDictionary](https://reactivex.io/documentation/operators/to.html)
* [ToEnumerable](https://reactivex.io/documentation/operators/to.html)
* [ToEvent](https://reactivex.io/documentation/operators/to.html)
* [ToEventPattern](https://reactivex.io/documentation/operators/to.html)
* [ToFlowable](https://reactivex.io/documentation/operators/to.html)
* [ToFuture](https://reactivex.io/documentation/operators/to.html)
* [to\_h](https://reactivex.io/documentation/operators/to.html)
* [toIndexedSeq](https://reactivex.io/documentation/operators/to.html)
* [toIterable](https://reactivex.io/documentation/operators/to.html)
* [toIterator](https://reactivex.io/documentation/operators/to.html)
* [ToList](https://reactivex.io/documentation/operators/to.html)
* [ToLookup](https://reactivex.io/documentation/operators/to.html)
* [toMap](https://reactivex.io/documentation/operators/to.html)
* [toMultiMap](https://reactivex.io/documentation/operators/to.html)
* [ToObservable](https://reactivex.io/documentation/operators/from.html)
* [toSet](https://reactivex.io/documentation/operators/to.html)
* [toSortedList](https://reactivex.io/documentation/operators/to.html)
* [toStream](https://reactivex.io/documentation/operators/to.html)
* [ToTask](https://reactivex.io/documentation/operators/to.html)
* [toTraversable](https://reactivex.io/documentation/operators/to.html)
* [toVector](https://reactivex.io/documentation/operators/to.html)
* [tumbling](https://reactivex.io/documentation/operators/window.html)
* [tumblingBuffer](https://reactivex.io/documentation/operators/buffer.html)
* [unsafeCreate](https://reactivex.io/documentation/operators/create.html)
* [unsubscribeOn](https://reactivex.io/documentation/operators/subscribeon.html)
* [**Using**](https://reactivex.io/documentation/operators/using.html)
* [**When**](https://reactivex.io/documentation/operators/and-then-when.html)
* [Where](https://reactivex.io/documentation/operators/filter.html)
* [while](https://reactivex.io/documentation/operators/repeat.html)
* [whileDo](https://reactivex.io/documentation/operators/repeat.html)
* [**Window**](https://reactivex.io/documentation/operators/window.html)
* [windowWithCount](https://reactivex.io/documentation/operators/window.html)
* [windowWithTime](https://reactivex.io/documentation/operators/window.html)
* [windowWithTimeOrCount](https://reactivex.io/documentation/operators/window.html)
* [windowed](https://reactivex.io/documentation/operators/backpressure.html)
* [withFilter](https://reactivex.io/documentation/operators/filter.html)
* [withLatestFrom](https://reactivex.io/documentation/operators/combinelatest.html)
* [**Zip**](https://reactivex.io/documentation/operators/zip.html)
* [zipArray](https://reactivex.io/documentation/operators/zip.html)
* [zipIterable](https://reactivex.io/documentation/operators/zip.html)
* [zipWith](https://reactivex.io/documentation/operators/zip.html)
* [zipWithIndex](https://reactivex.io/documentation/operators/zip.html)
* [++](https://reactivex.io/documentation/operators/concat.html)
* [+:](https://reactivex.io/documentation/operators/startwith.html)
* [:+](https://reactivex.io/documentation/operators/just.html)

**SINGLE**

RxJava (e i suoi derivati ​​come RxGroovy e RxScala) ha sviluppato una variante [Observable](https://reactivex.io/documentation/observable.html) chiamata "Single".

Un Single è qualcosa di simile a un Observable, ma invece di emettere una serie di valori, da nessuno a un numero infinito, emette sempre un valore o una notifica di errore.

Per questo motivo, invece di iscriverti a un Single con i tre metodi che usi per rispondere alle notifiche di un Observable ( onNext , onError e onCompleted ), usi solo due metodi per iscriverti:

**onSuccess**

a Single passa questo metodo all'unico elemento emesso da Single

**onError**

un Single passa con questo metodo il Throwable che ha impedito al Single di emettere un oggetto

A Single chiamerà solo uno di questi metodi e lo chiamerà solo una volta. Chiamando uno dei due metodi, il Single termina e l'abbonamento ad esso termina.

**SCHEDULER**

Se vuoi introdurre il multithreading nella tua cascata di operatori osservabili, puoi farlo istruendo quegli operatori (o particolari osservabili) a operare su particolari scheduler .

Alcuni operatori ReactiveX Observable hanno varianti che accettano uno Scheduler come parametro. Questi istruiscono l'operatore a svolgere parte o tutto il suo lavoro su un particolare Scheduler.

Per impostazione predefinita, un Observable e la catena di operatori ad esso applicati faranno il loro lavoro e avviseranno i suoi osservatori, sullo stesso thread su cui viene Subscribechiamato il suo metodo. L' operatore SubscribeOn modifica questo comportamento specificando uno Scheduler diverso su cui dovrebbe operare l'Observable. L' operatore ObserveOn specifica uno Scheduler diverso che l'Observable utilizzerà per inviare notifiche ai suoi osservatori.

Come mostrato in questa illustrazione, l' operatore SubscribeOn indica su quale thread l'Observable inizierà a operare, indipendentemente dal punto della catena di operatori in cui viene chiamato l'operatore. ObserveOn , d'altra parte, influisce sul thread che l'Observable utilizzerà di seguito dove appare l'operatore. Per questo motivo, è possibile chiamare ObserveOn più volte in vari punti durante la catena di operatori Observable per modificare su quali thread operano alcuni di tali operatori.

**Immagine che contiene schermata, testo, Carattere, Policromia

Descrizione generata automaticamente**