**EBook Gratuito**

# APPRENDIMENTO system.reactive

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#system.rea

ctive

# Sommario

# Di

# Capitolo 1: Iniziare con system.reactive

## Osservazioni

Questa sezione fornisce una panoramica di cosa sia system.reactive e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare eventuali soggetti di grandi dimensioni all'interno di system.reactive e collegarsi agli argomenti correlati. Poiché la Documentazione per system.reactive è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

## Examples

### Usando Rx nel tuo progetto

Installa il pacchetto NuGet `System.Reactive` , quindi aggiungi questa istruzione using per accedere ai metodi di estensione Rx:

```
using System.Reactive.Linq;
```

### Filtrando i valori di un osservabile

```
emails.Where(email => email.From == "John")
```

### Selezione di un nuovo valore per ogni valore in un osservabile

```
emails.Select(email => email.Body)
```

### Iscrizione / annullamento dell'iscrizione a un osservabile (IDisposable)

L'abbonamento restituisce un `IDisposable` :

```
IDisposable subscription = emails.Subscribe(email =>
    Console.WriteLine("Email from {0} to {1}", email.From, email.To));
```

Quando sei pronto per annullare l'iscrizione, devi semplicemente disporre dell'abbonamento:

```
subscription.Dispose();
```

### Sottoscrizione ad un osservabile (CancellationToken)

```
emails.Subscribe(email =>
    Console.WriteLine("Email from {0} to {1}", email.From, email.To),
    cancellationToken);
```

## Avvolgere un metodo asincrono come osservabile

Dato un metodo `async` come questo:

```
Task<string> GetNameAsync(CancellationToken cancellationToken)
```

`IObservable<string>` come una `IObservable<string>` questo modo:

```
Observable.FromAsync(cancellationToken => GetNameAsync(cancellationToken))
```

## Condivisione di un singolo abbonamento (Pubblica)

Data una `IObservable<Offer>` di `offers` da parte dei commercianti di acquistare o vendere alcuni tipi di articoli ad un prezzo fisso, possiamo abbinare coppie di compratori e venditori come segue:

```
var sellers = offers.Where(offer => offer.IsSell).Select(offer => offer.Merchant);
var buyers = offers.Where(offer => offer.IsBuy).Select(offer => offer.Merchant);
var trades = Observable.Zip(sellers, buyers, (seller, buyer) => new Trade(seller, buyer));
```

Il problema con questo è che ogni abbonamento alle `trades` si iscriverà alle `offers` due volte. Possiamo fare in modo che `sellers` e `buyers` condividano un solo abbonamento alle `offers` utilizzando `Publish` :

```
var trades = offers.Publish(_offers =>
{
    var sellers = _offers.Where(offer => offer.IsSell).Select(offer => offer.User);
    var buyers = _offers.Where(offer => offer.IsBuy).Select(offer => offer.User);
    return Observable.Zip(sellers, buyers, (seller, buyer) => new Trade(seller, buyer));
});
```

## Condivisione di una singola sottoscrizione (Publish + RefCount)

Questo codice si iscriverà alle e- `emails` osservabili due volte:

```
emails.Where(email => email.From == "John").Subscribe(email => Console.WriteLine("A"));
emails.Where(email => email.From == "Mary").Subscribe(email => Console.WriteLine("B"));
```

Per condividere una singola iscrizione alle `emails` , possiamo usare invece `Publish` e `RefCount` :

```
var _emails = emails.Publish().RefCount();
_emails.Where(email => email.From == "John").Subscribe(email => Console.WriteLine("A"));
_emails.Where(email => email.From == "Mary").Subscribe(email => Console.WriteLine("B"));
```

## Installazione o configurazione

Le estensioni reattive sono pubblicate sia su NuGet che su MyGet .

L'installazione e il loro utilizzo è quindi uguale a qualsiasi altro pacchetto NuGet:

```
Install-Package System.Reactive
```

NB: i nomi dei pacchetti sono cambiati tra la v2 e la v3. Vedi il README su Github per maggiori informazioni

> Ultime modifiche
>
> I pacchetti NuGet hanno cambiato la denominazione del pacchetto nel passaggio da v2.xx a> v3.0.0
>
> Rx-Main è ora System.Reactive Rx-Core è ora System.Reactive.Core Rx-Interfaces è ora System.Reactive.Interfaces Rx-Linq è ora System.Reactive.Linq Rx-PlatformServices è ora System.Reactive.PlatformServices Rx- Il test è ora Microsoft.Reactive.Testing

## Limitazione di un flusso

Supponiamo che sia necessario implementare una casella di ricerca automatica, ma l'operazione di ricerca è alquanto costosa, ad esempio l'invio di una richiesta Web o il rilevamento di un database. Si consiglia di limitare la quantità di ricerca eseguita.

Ad esempio, l'utente sta digitando "C # Reactive Extensions" nella casella di ricerca:

```
IObservable<string> TypingSearchText()
{
    return Observable.Create<string>(o =>
    {
        const string SearchText = "C# Reactive Extensions";

        var builder = new StringBuilder();
        foreach (var c in SearchText)
        {
            builder.Append(c);

            // notify that the search text has been changed
            o.OnNext(builder.ToString());

            // pause between each character to simulate actual typing
            Thread.Sleep(125);

            // spent some time to think about the next word to type
            if (c == ' ')
                Thread.Sleep(1000);
        }

        o.OnCompleted();

        return () => { /* nothing to dispose here */ };
    });
}
```

Ora, non vogliamo eseguire la ricerca ogni volta che l'utente preme un tasto. Invece, sarà fatto ogni volta che l'utente smette di digitare più di mezzo secondo:

```
TypingSearchText()
    // print the changes
    .Do(x => Console.WriteLine("Typing: " + x))

    // ignore changes that happens within 500ms of each other
    .Throttle(TimeSpan.FromMilliseconds(500))

    // some costly operation
    .Subscribe(x => Console.WriteLine("Searching: " + x));
```

Produzione :

```
Typing: C
Typing: C#
Typing: C#
Searching: C#
Typing: C# R
Typing: C# Re
...
Typing: C# Reactive
Typing: C# Reactive
Searching: C# Reactive
Typing: C# Reactive E
Typing: C# Reactive Ex
...
Typing: C# Reactive Extension
Typing: C# Reactive Extensions
Searching: C# Reactive Extensions
```

## Ignorare i valori ripetuti

Esistono due operatori per il filtro dei duplicati:

```
emails.Distinct(); // Never see the same value twice
emails.DistinctUntilChanged(); // Never see the same value twice in a row
```

Puoi anche passare un predicato:

```
emails.DistinctUntilChanged(x => x.Length); // Never see the same length email twice in a row
```

## Ottieni una aggregazione in esecuzione

Supponiamo di avere un osservabile caldo per il quale ti piacerebbe tenerne conto. Potrebbe essere il `IObservable<StockTick>` e si desidera mantenere il conteggio del volume medio degli scambi. Puoi usare `Scan` per questo.

```
var tradeVolume = stockTicks.Select(e => e.Price)
    .Scan(0.0m, (aggregated, newtick) => aggregated + newtick)
    .Select((aggregated, index) => aggregated / (index + 1))
```

Ora puoi semplicemente iscriverti al tuo volume di scambi che viene aggiornato in tempo reale al ricevimento di ogni nuovo Tick.

```
var subscription = tradeVolume.Subscribe(vol => Console.WriteLine("New trade volume is {0}",
vol);
```

Leggi Iniziare con system.reactive online: https://riptutorial.com/it/system-reactive/topic/1325/iniziare-con-system-reactive

# Titoli di coda

| S. No | Capitoli | Contributors |
|---|---|---|
| 1 | Iniziare con system.reactive | Benjol, Community, Dorus, fahadash, Michael Richardson, Paul D'Ambra, sdgfsdh, Shlomo, Timothy Shields, Xiaoy312 |