**EBook Gratis**

# APRENDIZAJE system.reactive

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

#system.rea

ctive

# Tabla de contenido


Acerca de ... 1

Capítulo 1: Empezando con system.reactive ... 2

Observaciones ... 2

Examples ... 2

Usando Rx en tu proyecto ... 2

Filtrado de los valores de un observable ... 2

Seleccionando un nuevo valor para cada valor en un observable ... 2

Suscripción / cancelación de la suscripción a un observable (IDisposable) ... 2

Suscribiéndose a un observable (CancelaciónToken) ... 2

Envolviendo un método asíncrono como un observable ... 3

Compartiendo una sola suscripción (Publicar) ... 3

Compartiendo una sola suscripción (Publicar + RefCount) ... 3

Instalación o configuración ... 3

Estrangulamiento de un arroyo ... 4

Ignorando valores repetidos ... 5

Obtener una agregación en ejecución ... 5

Creditos ... 7

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: system-reactive

It is an unofficial and free system.reactive ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official system.reactive.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con system.reactive

## Observaciones

Esta sección proporciona una descripción general de qué es system.reactive y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de system.reactive, y vincular a los temas relacionados. Dado que la Documentación para system.reactive es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Examples

### Usando Rx en tu proyecto

Instale el paquete NuGet `System.Reactive` , luego agregue esta declaración usando para acceder a los métodos de extensión Rx:

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

### Filtrado de los valores de un observable.

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

### Seleccionando un nuevo valor para cada valor en un observable

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

### Suscripción / cancelación de la suscripción a un observable (IDisposable)

Suscripción devuelve un `IDisposable` :

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

Cuando esté listo para darse de baja, simplemente elimine la suscripción:

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

### Suscribiéndose a un observable (CancelaciónToken)

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

## Envolviendo un método asíncrono como un observable.

Dado un método `async` como este:

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

Envuélvalo como una `IObservable<string>` como esto:

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

## Compartiendo una sola suscripción (Publicar)

Dado un `IObservable<Offer>` de `offers` de comerciantes para comprar o vender algún tipo de artículo a un precio fijo, podemos emparejar pares de compradores y vendedores de la siguiente manera:

```
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));
```

El problema con esto es que cada suscripción a las `trades` se suscribirá a las `offers` dos veces. Podemos hacer que los `sellers` y `buyers` compartan una única suscripción a las `offers` utilizando `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));
});
```

## Compartiendo una sola suscripción (Publicar + RefCount)

Este código se suscribirá a los `emails` observables dos veces:

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

Para compartir una única suscripción a los `emails` , podemos utilizar `Publish` y `RefCount` en `RefCount` lugar:

```
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"));
```

## Instalación o configuración

Las extensiones reactivas se publican tanto en NuGet como en MyGet .

Instalarlos y usarlos es, por lo tanto, lo mismo que cualquier otro paquete NuGet:

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

NB nombres de paquetes cambiados entre v2 y v3. Ver el archivo README en Github para más información.

> Rompiendo cambios
>
> Los paquetes de NuGet han cambiado su nombre de paquete en el movimiento de v2.xx a> v3.0.0
>
> Rx-Main ahora es System.Reactive Rx-Core ahora es System.Reactive.Core Rx-Interfaces es ahora System.Reactive.Interfaces Rx-Linq ahora es System.Reactive.Linq Rx-PlatformServices es ahora System.Reactive.PlatformServices Rx- Pruebas ahora es Microsoft.Reactive.Testing

## Estrangulamiento de un arroyo

Supongamos que necesita implementar un cuadro de búsqueda automático, pero la operación de búsqueda es algo costosa, como enviar una solicitud web o acceder a una base de datos. Es posible que desee limitar la cantidad de búsqueda que se realiza.

Por ejemplo, el usuario está escribiendo "C # Extensiones reactivas" en el cuadro de búsqueda:

```
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 */ };
    });
}
```

Ahora, no queremos realizar la búsqueda cada vez que el usuario presiona una tecla. En su lugar,

se hará cuando el usuario deje de escribir durante más de medio segundo:

```
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));
```

Salida:

```
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
```

## Ignorando valores repetidos

Hay dos operadores para filtrar duplicados:

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

También puedes pasar un predicado:

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

## Obtener una agregación en ejecución

Supongamos que tiene un observable para el que le encantaría mantener la cuenta. Podría ser el `IObservable<StockTick>` y desea mantener la cuenta del volumen de comercio promedio. Puedes usar `Scan` para eso.

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

Ahora puede simplemente suscribirse a su volumen de comercio que se actualiza en vivo al recibir cada nuevo Tick.

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

Lea Empezando con system.reactive en línea: https://riptutorial.com/es/system-reactive/topic/1325/empezando-con-system-reactive

# Creditos

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