THỢ XÂY LÀM CODER

MAY 5, 2017SEPTEMBER 6, 2020 THOXAYLAMCODER TIPS & TRICKS, UNCATEGORIZED

Reactive Programming và UniRX trong Unity – P2 : một số ví dụ

Mục lục cho các bạn dễ tìm

Nói về UniRX

Do lập trình viên người Nhật Yoshifumi Kawai, re-implement lại từ bộ .NET Reactive Extensions, để giúp chúng ta thoải mái code RP trên C# Unity, ngoài ra còn có thêm vài Utilities này nọ dùng riêng trong Unity nữa. Mấy bạn có clone hay download từ GitHub (https://github.com/neuecc/UniRx)của ông ấy thì Star một cái nhé, họ làm không công vì cộng đồng mà 😉

Thế UniRX nó gồm cái gì trong đó ? Trích nguyên văn lời tác giả "*UniRx is Core Library (Port of Rx) + Platform Adaptor (MainThreadScheduler / FromCoroutine / etc) + Framework (ObservableTriggers / ReactiveProoperty / etc)*".

Core Library (port of RX)

UniRX có hầu hết các Operators (http://reactivex.io/documentation/operators.html) từ bộ .NET Reactive Extensions. Vẫn còn thiếu một số operator, nhưng mà yên tâm, với các operator đã có là quá dư thừa cho hầu hết các yêu cầu (theo như thực tế sử dụng, và đánh giá từ một số blog mà mình có đọc qua).

Platform Adaptor

Có thể hiểu là "cầu nối", đại khái là UniRX viết thêm một số thứ (custom classes và một số MonoBehavior) để tận dụng / chuyển một số features trong engine Unity, thành các Observables cơ bản (có thể tạm hiểu là các build-in Observables của UniRX) cho mình xài. Như thông qua cái *MainThreadScheduler*, RX cung cấp được các Observable như *EveryUpdate, EveryEndOfFrame, Timer, Interval* … mấy cái này dùng nhiều lắm. Thêm cả mấy cái Coroutine nữa, nhất là cái *MicroCoroutine*, khá nổi về mặt performance. (tự đọc trong github của tác giả nghe mấy chú).

Framework

Mình cũng vừa ghi ở trên đó, sau khi tác giả viết một cơ số các class … adaptor bên dưới, sẽ tòi ra được vài API cho mình xài, và công việc của mình là học cách dùng các API này. Nói cho đúng là UniRX tòi ra một cơ số các Observables để mình nghịch ngợm cùng Operators (Nếu ai đọc qua Github của tác giả sẽ thấy thêm khái niệm *ReactiveProperty*, thật ra cũng là Observable á mà).

Một vài sample cơ bản

Nhấn mạnh là các bạn nên đọc qua các bài viết mình có đề cập hé, phần sau đây chủ yếu mình nêu các ví dụ đơn giản, ứng dụng trong thực tế, nên mình không có giải thích hay cắt nghĩa lại các khái niệm operator, transform, filter, combine này kia hế :v. Ứng với các ví dụ về Operator, mình đính link đến page ReactiveX, có phần định nghĩa cho từng Operator.

Xài thử cái "*build-in*" Interval của UniRX

Ví dụ đầu siêu đơn giản, làm hàm đếm tik tak, sau mỗi 1s thì phát ra âm thanh tik tik.

```
1  void Start()
2  {
3      Observable.Interval(TimeSpan.FromSeconds(1f))
4          .Subscrivbe(_ => PlayTikSound());
5  }
6
7  void PlayTikSound()
8  {
9  }
```

view raw UniRX_sample_Interval.cs hosted with ❤ by **GitHub**

Quá dễ, khỏi cần giải thích rườm rà hế.

Where (http://reactivex.io/documentation/operators/filter.html) (Filter)

Cái này thì mấy bạn thấy ngay trong cái ví dụ về mouseClick ở phần 1 (https://thoxaylamcoder.wordpress.com/2017/05/05/reactive-programming-va-unirx-trong-unity-p1/) rồi đấy.

```
1 var mousePressStream = Observable.EveryUpdate()
2     .Where(_ => Input.GetMouseButtonDown(0))
3     .Subscribe(_ => Debug.Log("Mouse Press"));
```

view raw UniRX_sample_Where.cs hosted with ❤ by **GitHub**

Diễn tả cho có văn vẻ tí thì thế này :

– *Observable.EveryUpdate()* : đây là một trong các build-in Observable của UniRX, stream này emit một giá trị *long* mỗi frame của chương trình, giá trị *long* này là số framecount từ khi UniRX đc init.

– áp cái operator Where lên stream trên, mình thu về một stream mới, chỉ emit giá trị *long* mỗi khi Input.Get…=true. À, cái *operator Where nó không thay đổi kiểu dữ liệu của item*, nên ở đây vẫn là *long*.

– được stream "ưng ý" rồi thì Subscribe ra, ở đây chỉ là in Log, nên ko dùng giá trị *long* được emit từ stream.

Thêm một đề bài bonus : cái này thì nghe quen, mình hay dùng làm ví dụ trong các post trước -> Làm tính năng Jump, với điều kiện người chơi phải đang ở trên mặt đất thì bấm Space mới cho nhảy. Tự viết hé.

Select (http://reactivex.io/documentation/operators/map.html) (Map)

Đề bài : in ra tọa độ mousePress.

```
1 var mousePressPosStream = Observable.EveryUpdate()
2     .Where(_ => Input.GetMouseButtonDown(0))
3     .Select(_ => Input.mousePosition)
4     .Subscribe(pos => Debug.Log("MousePressPos : " + pos));
```

view raw UniRX_sample_Select.cs hosted with ❤ by **GitHub**

Ở đây, sau khi dùng operator Select, ta thu về một stream emit ra các vector3 (Input.mousePosition), chỗ Subscribe sẽ in ra các position này. Operator này có tên gọi hay dùng khác là *Map*, hay có thể hiểu, ứng với mỗi item của stream này, mình áp một function lên để thu về một item khác. Như demo trên, item *long* đc select lại thành *Vector3* đó.

Bonus một đề bài mình có nói ở phần 1 (https://thoxaylamcoder.wordpress.com/2017/05/05/reactive-programming-va-unirx-trong-unity-p1/) : chỉ in ra giá trị X của tọa độ mousePress của những press ở nữa phải màn hình.

```
1 var mousePressPosStream = Observable.EveryUpdate()
2     .Where(_ => Input.GetMouseButtonDown(0))
```

```
3       .Select(_ => Input.mousePosition);
4
5   var filterMousePosStream = mousePressPosStream
6       .Where(mousePos => mousePos.x > Screen.width * 0.5f)//on halfRight screen only
7       .Select(mousePos => mousePos.x)//take the xPos
8       .Subscribe(xPos => Debug.Log("MousePress on HalfRight screen : " + xPos));
```

view raw UniRX_sample_Select_02.cs hosted with ❤ by **GitHub**

Scan (http://reactivex.io/documentation/operators/scan.html)

Cái này cho đề bài đơn giản thôi : in ra số lần click mouse. (tức in ra 1,2,3…. khi người dùng press mouse). Cái này giống kiểu HelloWorld cho cái operator Scan này.

```
1   var mouseCountStream = Observable.EveryUpdate()
2       .Where(_ => Input.GetMouseButtonDown(0))// mousePress stream
3       .Select(_ => 1)//each mousePress, we emit "1" value
4       .Scan((previous, current) => previous + current)//counting
5       .Subscribe(x => Debug.Log("Press count : " + x));
```

view raw UniRX_sample_Scan.cs hosted with ❤ by **GitHub**

Từ mousePressStream, chúng ta chuyển thành stream sẽ emit giá trị **"1"** mỗi lần pressMouse, rồi dùng Scan để được một stream, sẽ emit các giá trị theo quy luật cộng dồn lên.

Hoặc nếu muốn đề bài có vẻ game game hơn thì thế này : player đi trong rừng, nhặt các bịch tiền, giá trị các túi tiền khác nhau, bạn có một stream, emit số tiền trong từng bịch. Yêu cầu : hiển thị số tiền cộng dồn của player lên màn hình.

SelectMany (http://reactivex.io/documentation/operators/flatmap.html) (FlatMap)

Nguyên văn định nghĩa nó thế này : "*transform the items emitted by an Observable into Observables, then flatten the emissions from those into a single Observable*", tạm dịch "*chuyển hóa item của một Observable thành một cơ số Observable khác, sau đó san phẳng cái đống đó vào một Observable duy nhất*". ?%#@?$%@& !!! Hé hé.

Hiểu cái này thì … cũng hơi lằn nhằn, mấy bạn xem cái video này (https://channel9.msdn.com/Blogs/J.Van.Gogh/Reactive-Extensions-API-in-depth-SelectMany), mình thì lấy một vd thế này, cái này thì hay gặp nè : *hero bắn súng tiểu liên, khi nhấp chuột xuống thì bắn đầu xả đạn, mỗi 0.1s thì bắn một viên*, cho đơn giản thì tạm cho unlimit ammo luôn hé.

```
1   void Start()
2   {
3       var mouseDownStream = Observable.EveryUpdate().Where(_ => Input.GetMouseButtonDown
```

```
4       var mouseUpStream = Observable.EveryUpdate().Where(_ => Input.GetMouseButtonUp(0));
5       var intervalStream = Observable.Interval(TimeSpan.FromSeconds(0.1f));
6
7       var shootStream = mouseDownStream
8           // flat the mouseDown with the 0.1s interval
9           .SelectMany(_ => intervalStream)
10          .TakeUntil(mouseUpStream)// stop shooting when mouseUp emit
11          .RepeatSafe()// repeat the process
12          .Subscribe(_ => Shoot());
13  }
14
15  void Shoot()
16  {}
```

view raw UniRX_sample_SelectMany.cs hosted with ❤ by **GitHub**

Với đề bài trên, mình cần một stream, bắt đầu emit khi mouseDown, kết thúc emit khi mouseUp, và nó sẽ emit item với interval = 0.1s.

- Declare mouseUp/Down stream, 2 stream này sẽ được dùng để "kẹp đầu đuôi". Hai stream này item kiểu *long*, mình sẽ ko dùng giá trị *long* đó, chỉ dùng stream này để … lấy thời điểm emit, làm "đầu đuôi"
- Cái intervalStream sẽ được dùng làm tham số cho operator SelectMany phía sau.
- Dùng *SelectMany* lên mouseDownStream, ta sẽ được một stream mới (tạm gọi là *shootStream* hế), nó lấy item đầu tiên khi mouseDownStream emit, và tiếp tục emit liên tục (*FlatMap*), với interval như khai báo ở trên. Đây là lí do intervalStream dùng như input cho cái SelectMany.
- Dùng *TakeUntil* để "ngưng" cái shootStream lại, khi cái mouseUp stream này emit. Đến đây thì coi như shootStream đã ok, đã có thể Subscribe ra và gọi hàm *Shoot()*.
- Nếu dừng ở *TakeUntil*, cái shootStream sẽ stop luôn, và lần kế tiếp mouseDown stream emit, sẽ chẳng có hiện tượng gì cả. Dùng Operator *Repeat (http://reactivex.io/documentation/operators/repeat.html)* sẽ giúp lặp lại chu trình trên. Cái *RepeatSafe* operator thuộc về UniRX, "*allows to stop when a target GameObject has been destroyed*", mấy bạn tự đọc thêm ở GitHub (https://github.com/neuecc/UniRx#observable-lifecycle-management) của UniRX hé.

First (http://reactivex.io/documentation/operators/first.html)

Cái này thì chắc cũng dễ đoán rồi hé, emit item đầu tiên của stream, mấy bạn tự đọc trong link mình đính để hiểu nhé. Ở đây mình nói thêm, operator này nếu dùng thêm *Func boolean* làm tham số thì hữu dụng hơn. Như cái ví dụ sau đây, sẽ in ra lần đầu tiên user click ở bên phải màn hình:

```
1   var mousePosStream = Observable.EveryUpdate()
2       .Where(_ => Input.GetMouseButtonDown(0))
3       .Select(_ => Input.mousePosition.x);
4
```

```
5  var halfRightScreenStream = mousePosStream
6      .First(x => x > Screen.width * 0.5f)
7      .Subscribe(_ => Log.Info("First click on half right screen !!"));
```

view raw UniRX_sample_First.cs hosted with ❤ by **GitHub**

Trong thực tế project thì mấy cái kiểu "lấy cái đầu tiên" này hay gặp trong Tutorial, Achievement. Gợi ý thế, tùy các bạn thích implement hay áp dụng thế nào thấy hợp lý là được.

Gom góp

Ví dụ về operator thì nhiều vô số kể, trước mắt mình demo vài cái hay dùng, mai mốt có thời gian mình update thêm. Nhấn mạnh là các bạn tự đọc thêm ngay trang Github của UniRX hé, nhiều operator lẫn Ultilities hay ho do lắm. Phần 2 mình tạm kết ở đây, qua Phần 3 (https://thoxaylamcoder.wordpress.com/2017/05/05/reactive-programming-va-unirx-trong-unity-p3/) sẽ nói vài cái sâu hơn chút về UniRX.

Published by **ThoXayLamCoder**

View all posts by ThoXayLamCoder

2 thoughts on “Reactive Programming và UniRX trong Unity – P2 : một số ví dụ”

1. Pingback: Reactive Programming và UniRX trong Unity – P1 – Thợ xây làm Coder
2. Pingback: Reactive Programming và UniRX trong Unity – P3 – Thợ xây làm Coder