THỢ XÂY LÀM CODER

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

Reactive Programming và UniRX trong Unity – P1 : vài khái niệm cơ bản

Mục lục cho các bạn dễ tìm

Học từ ví dụ

Món này tiếp cận theo hướng học bằng ví dụ có vẻ dễ hơn, cho nên mình bắt đầu từ một đề bài và một mẫu code đơn giản thế này. Đề bài : *mỗi lần người chơi click trên màn hình thì in ra tọa độ vị trí click*. Ui dào, dễ ẹc.

```
1  void Update()
2  {
3      if (Input.GetMouseButtonDown(0))
4      {
5          Debug.Log("MousePress");
6      }
7  }
```

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

Và rồi, có một kiểu lập trình, người ta gọi nó là ReactiveProgramming, và có một thư viện hỗ trợ kiểu lập trình đó trong Unity là UniRX, và người ta gõ lại thế này :

```
1  void Start()
2  {
3      Observable.EveryUpdate()
4          .Where(_ => Input.GetMouseButtonDown(0))
5          .Subscribe(_ => Debug.Log("MouseClick"));
```

```
6 }
```

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

Zòi, có thể sẽ có vài ý nghĩ thế này "*sax, cũng chả ngắn hơn được bao nhiêu*". Oker, đề bài kế tiếp, cũng là một đề bài HelloWorld cho mấy bạn mới nghịch UniRX : viết hàm detect double click, với mô tả : *khi người chơi click lên màn hình đủ 2 click, khoảng thời gian giữa 2 click không quá 0.25s (hoặc một số được config)*. Rồi, mời các bác dùng hàm Update, với biến count gì gì đó, thêm biết tính thời gian _time gì gì … Còn với RP thì người ta gõ vài dòng thế này :

```
1 var clickStream = Observable.EveryUpdate()
2       .Where(_ => Input.GetMouseButtonDown(0));
3
4 clickStream.Buffer(clickStream.Throttle(TimeSpan.FromMilliseconds(250)))
5       .Where(xs => xs.Count >= 2)
6       .Subscribe(xs => Debug.Log("DoubleClick Detected! Count:" + xs.Count));
```

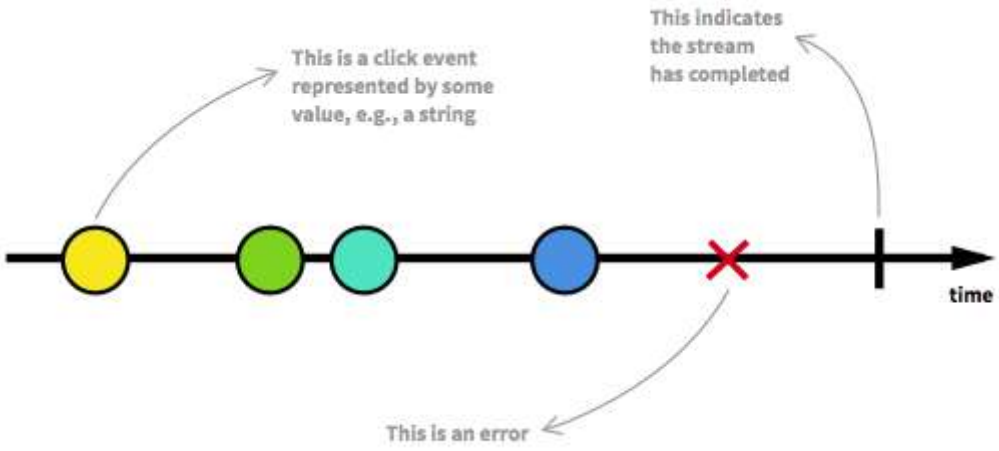
view raw UniRX_doubleClick.cs hosted with ❤ by **GitHub**

Từ câu chuyện ví dụ thực tế, đến việc hiểu Reactive Programming (RP)

Nói đến ReactiveProgramming thì sẽ cần đụng đến các khái niệm *Functional programming*, *Declarative programming*. Mấy món này thì hơi trừu tượng, và cũng khá nhiều các bài viết trên mạng nói về mấy chủ đề này rồi. Trong seri này, mình sẽ ko đi lại phần khái niệm cho mấy Programming Paradigm này, chủ yếu mình chỉ trình bày một số ví dụ áp dụng trong thực tế, rồi các bạn tự rút ra định nghĩa cho bản thân hé.

Phần này, khuyến khích các bạn tự đọc hiểu, tham khảo bài viết tuyệt vời này (https://gist.github.com/staltz/868e7e9bc2a7b8c1f754), tác giả viết rất chi tiết và rất dễ hiểu. Sau đó các bạn nên đọc thêm bài viết ở Wiki (https://en.wikipedia.org/wiki/Reactive_extensions), từ ngữ nó hơi .. học thuật một tí. Cá nhân mình sau khi tìm hiểu và ứng dụng, thì thấy các khái niệm quan trọng gồm :

Stream (Observable)



Viết cho đủ sẽ phải là Asynchronize Data Stream, mà thôi, stream là đủ hiểu rồi. Các bạn đọc định nghĩa đầy đủ của Stream (hay Observable) ở link này (http://reactivex.io/documentation/observable.html). Ở đây mình gõ nháp nháp ra 2 mẫu code (cứ đọc đã, chưa cần hiểu)

```
1  //--------------------------------
2  //---------C# Action -------------
3  //--------------------------------
4
5  public event Action<Vector2> OnMouseClickHandler;
6  void Start()
7  {
8      OnMouseClickHandler += HandleMouseClick;
9  }
10
11 void HandleMouseClick(Vector2 mousePos)
12 {
13     Debug.Log("MouseClick !!!");
14 }
15
16 //--------------------------------
17 //---------- Stream - Observable---
18 //--------------------------------
19
20 /// Assume we get this Observale from .."somewhere", not define like this
21 IObservable<Vector2> mouseClickStream;
22 void Start()
23 {
24     mouseClickStream.Subscribe(mousePos => HandleMouseClick(mousePos));
25 }
26
27 void HandleMouseClick(Vector2 mousePos)
```

```
28  {
29      Debug.Log("MouseClick !!!");
30  }
```

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

Cái khúc code đầu tiên, là viết theo "kiểu" C# event. Mình đang cần bắt sự kiện người chơi click chuột để xử lý, và cứ cho là ở đâu đó người ta setup sẵn cái event *OnMouseClickHandler* rồi, mình chỉ có register ra rồi xử lý trong hàm *HandleMouseClick*. Cái OnMouseClickHandler có para là Vector2, tọa độ click chuột. Zòi, rút ra mấy điểm sau :

- Cái event OnMouseClick đó được invoke ở đâu, bao nhiêu lần, mình không cần biết
- Chỉ biết là mỗi lần invoke, nó sẽ đi kèm 1 tham số Vector2
- Việc của mình chỉ có register để nhận event, và xử lý trong hàm HandleMouseClick, với đầu vào là Vector2 nhận đc.

Thì Stream nó cũng na ná như thế, như ở trên, mình có gõ nháp cái *mouseClickStream* đấy, đại loại cũng là :

- Nếu C# event gọi là invoke, thì cái stream này họ dùng từ ***emit***, (phát ra, bắn ra, chưởng ra .v.v.).
- Cái C# event nó đi kèm para là Vector2, thì stream có thể hiểu là nó emit ra các ***items***, như vd trên là các Vector2. Tất nhiên item có thể là int, float… struct, class … và thậm chí có thể là một stream khác nữa.
- C# event có thao tác AddListener, thì với Stream người ta gọi cái thao tác "*đăng kí để nhận item đc emit ra*" là ***Subscribe***. Có thể hiểu, chúng ta là các subscribers, chúng ta ngồi theo dõi, chầu chực các stream, khi nào chúng phun (emit) ra item, thì mình "chụp" lại để xử lý. Và trong RP, những đối tượng có thể "theo dõi" được (streams that can be observed), họ dùng một cái tên hay hơn là ***Observables***. Và các bạn sẽ gặp cái từ Observable này nhiều hơn, oker :v

Úi dào, giống y chang thế thì phát minh ra chi cho cực zậy hà trời ???
Hé hé, đó chính là lý do vì sao mình cần biết đến khái niệm tiếp theo :

Operator

Trích nguyên văn định nghĩa Operator nhé "*An operator is a function that takes one observable (the source) as its first argument and returns another Observable (the destination, or outer observable). Then for every item that the source observable emits, it will apply a function to that item, and then emit it on the destination Observable. It can even emit another Observable on the destination observable. This is called an inner observable.*" Buồn buồn đọc thêm bài phân loại operator ở đây (http://reactivex.io/documentation/operators.html) nhé.

Nôm na Operator giúp mình làm những việc kiểu thế này :

- Ê cái *mouseClickStream* kia, giờ tao méo thích nhận Vector2 nữa, chỉ thích nhận cái giá trị X của cái Vector2 đó thôi.
- Tao muốn chỉ nhận được những vector2 mà nó nằm ở nữa trái màn hình thôi (x<screenW/2), đám còn lại drop, ko xài.
- Nhận mỗi lần một cái mất công quá, tao muốn gom lại, được 3 cái click rồi emit ra thành một List<vector2> cho tao.

- .v.v…v…

Và người ta phân loại các operator theo category thế này :

- **Creating** : tạo ra một Observable mới từ observable ban đầu.
- **Transforming** : thay đổi item được emit từ một observable, có thể là áp một function lên, chia một item thành nhiều item khác .v.v..
- **Filtering** : có thể hiểu là "lọc", gom lại, hoặc loại bỏ một số item được emit.
- **Combining** : kết hợp các Observable lại để tạo ra một observable mới.

Cơ bản là có mấy category đó, ở phần 2 (https://thoxaylamcoder.wordpress.com/2017/05/05/reactive-programming-va-unirx-trong-unity-p2/) mình sẽ có ví dụ mẫu cho mấy category này.

Chaining Operators

Cái này mình nói thêm chút thôi, để các bạn dễ hiểu khi đọc các ví dụ, sample trên mạng. *Phần lớn các Operator, khi đã áp dụng lên một Observable, bạn sẽ nhận lại một Observable mới, chứ không hề thay đổi Observable cũ*. Đây cũng là một tư tưởng lớn trong RP : *không thay đổi data, chỉ chuyển nó qua dạng khác* !!. Và vì kết quả trả về luôn là một Observable mới, nên người ta hay viết nối liền các operator (Chaining Operators), các bạn sẽ hay gặp các câu lệnh nối một đống operator, đến cuối cùng mới Subscribe() ra, trông thế này :

```
1 IObsrvable<Vector2> mouseClickStream;
2 mouseClickStream.Buffer(...).Where(...).SkipWhile(...).Take(...).Subscribe(...);
```

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

Quay lại câu chuyện DoubleClick

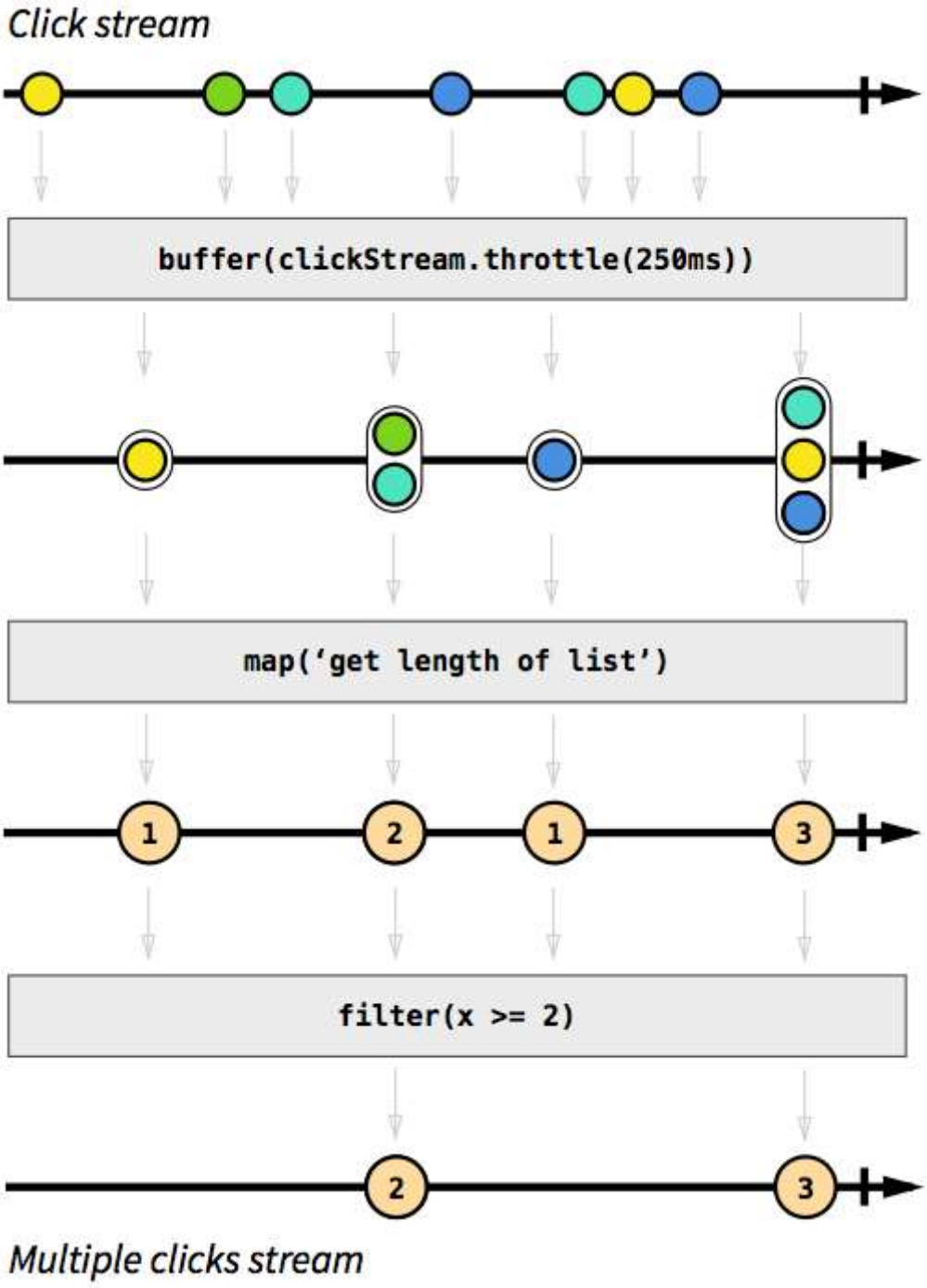
Imperative Programming

Để giải quyết nó, theo cách nghĩ và cách làm thông thường mà chúng ta vẫn áp dụng từ xưa giờ (nó được gọi là Imperative programming), trông đại khái thế này:
– vì yêu cầu click phải 2 lần mới được tính là DoubleClick nên phải có một biến lưu số lượng click (clickCount), và một câu if ở đâu đó, kiểm tra clickCount >= 2
– khoảng thời gian giữa 2 click không được quá 0.25 giây, vậy phải có một biến đếm thời gian (countTime), và nếu countTime > 0.25 thì phải reset clickCount
.v.v…

Nói chung nó là một chuỗi xoay vòng giữa việc đặt thêm các biến để LƯU TRẠNG THÁI (state) chương trình, cộng thêm một cơ số các hàm để kiểm tra và thay đổi giá trị của các biến đó, tạo nên flow của chương trình.

Reactive Programming

Với RP thì người ta tư duy khác :



1. Ta có một "input" sơ khai là một ClickStream, sẽ emit những lần click của user. Công việc tiếp theo là "nghịch" cái stream này, cái đích cuối cùng là phải ra được một stream, mà nó emit những lần user click>2 và trong khoảng 0.25s
2. Đầu tiên là dùng operator *Buffer (http://reactivex.io/documentation/operators/buffer.html)* để gom những click trong các đoạn có chiều dài 0.25s. Để có cái input cho Buffer thì cần sự tham gia của operator

Throttle (http://reactivex.io/documentation/operators/debounce.html) (hay có chỗ là Debounce) để xử cái việc tạo ra các khoảng 0.25s. Đến đây, ta được một stream sẽ emit các item là các List<clicks>.

3. Tiếp theo là transform nó, thành một stream sẽ emit độ dài các list từ stream kế trên. Nói rõ ra là, ứng với mỗi item của stream kề trc (List<click>), nó emit cái length của list. Tới đây sẽ dùng operator *Map*, hay trong UniRX là *Select (http://reactivex.io/documentation/operators/map.html)*.
4. Cuối cùng là "lọc" phát nữa, chỉ lấy những item có giá trị > 2, cái filter ở trong hình trên, trong UniRX sẽ là operator *Where (http://reactivex.io/documentation/operators/filter.html)*. Và thế là được stream cuối cùng, thõa mãn yêu cầu, Subscribe ra, làm gì thì tùy, ở trên là in cái Log ra Console thôi.

Nhìn chung, *RP làm việc xoay quanh các data stream, nó không có thêm biến để lưu trạng thái chương trình, chỉ áp dụng các function lên data, để chuyển đổi nó thành dạng khác, chứ không hề thay đổi data (y = f(x), chớ ko có mà x += z …).*

Thoy, mở màn dài quá sợ mấy bạn buồn ngủ, qua phần 2 (https://thoxaylamcoder.wordpress.com/2017/05/05/reactive-programming-va-unirx-trong-unity-p2/) mình sẽ làm vài cái demo áp dụng cho sinh động.

Published by **ThoXayLamCoder**

View all posts by ThoXayLamCoder

5 thoughts on “Reactive Programming và UniRX trong Unity – P1 : vài khái niệm cơ bản”

1. Pingback: Reactive Programming và UniRX trong Unity – P2 – Thợ xây làm Coder
2. Pingback: Reactive Programming và UniRX trong Unity – P3 – Thợ xây làm Coder
3. **Thành Công** says:
   MAY 7, 2017 AT 4:46 PM
   hi’ hi’

   REPLY
4. **Cuong** says:
   MAY 10, 2017 AT 3:45 AM
   Nguyên lý hoạt động gần giống với ReactiveSystem của Entitas nhỉ.

   REPLY
   ThoXayLamCoder says:
   MAY 10, 2017 AT 5:36 AM
   À, mình lấy ví dụ cho dễ hiểu thôi. Cái này nó là một Programming paradigm, một “phong cách” trong lập trình. Cái Entitas là một framework ECS, 2 cái này thật ra không liên quan gì nhiều lắm, mặc dù nghe tên hoặc phân tích ra thì có vài cái nghe giống giống nhau
   😉

   REPLY