THỢ XÂY LÀM CODER

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

Reactive Programming và UniRX trong Unity – P3 : ngoài lề về UniRX

Mục lục cho các bạn dễ tìm

Đến phần đào sâu

Làm cái thế nào mà UniRX có thể làm được các tác vụ cần đến hàm Update(), LateUpdate() … mà mình lại chẳng thấy cái hàm Update ở đâu hết ?

Đúng, như mình cũng có đề cập ở các phần trước rồi, nếu để ý hoặc chịu khó đào code, UniRX có tạo một object *MainThreadDispatcher*, đặt dontDestroyOnLoad, và chính nó sẽ đảm nhiệm việc chạy các *Update, FixUpdate, LateUpdate* .v.v.. Đây là một trong các khâu cầu nối với Unity, và cũng là để tạo ra một số các build-in observer mà nãy giờ các bạn đã đọc : *EveryUpdate, EveryLateUpdate* .v.v…

Và cơ bản thì, UniRX đã "chuyển" hầu hết các message thân thuộc của Unity thành các Observables để mình có thể nghịch (Enable, Destroy, ApplicationQuit, EndOfFrame ..v.v…)

Cái OnValueChanged tuyệt ghê luôn, trước giờ mình toàn phải làm Update hoặc chơi event mới được, giờ chỉ cần OnValueChanged phát là khỏe re luôn, quá đã !!

Thật ra thì, UniRX có một số custom class để thực hiện việc CreateObservable này, cái lõi cuối cùng, cũng là dùng *EveryUpdate* để mà check value của cái biến mà các bạn muốn theo dõi, nếu nó thay đổi so với lastValue thì sẽ OnNext một phát, thế thôi. Chứ đừng nghĩ là có "phép màu" gì đó khiến cho cái value đó tự động notify bạn khi giá trị nó thay đổi. Nói đi cũng nói lại, cứ yên tâm mà xài, không hao tốn bao nhiêu lắm đâu mà phải đắn đo.

Cái Button.OnClickAsObservable tuyệt ghê, biến một event của Button thành Observable

Cái này thì chắc mấy bạn cũng dễ phát hiện, thật ra là nó add thêm component vào cái Button đó, lắng nghe sự kiện onClick từ button rồi OnNext ra. Mình thấy có vài ông đọc cái vd của UniRX xong, xài liền, kiểu *Button.OnClickAsObservable().Subscribe(…)*, trong khi vốn sẵn Button nó có cái event rồi, tự dưng tốn thêm một cái component cho Button chỉ để nhận 1 event vốn đã có sẵn ?? Mình thì không thích cách dùng này lắm, nếu như đề bài thay đổi thành "người chơi nhấn nút 3 lần, cộng thêm này nọ … thì mới thực thi function ABC", tức là có sự kết hợp của nhiều stream, như vậy mới cần UniRX, mới cần đến tính năng chuyển UnityAction thành Observable.

Chà chà, khai báo ra một đống stream, cái nào cũng EveryUpdate, EveryNàyNọ, có phí tài nguyên quá không ta?

Yên tâm, các stream, chỉ mang tính chất khai báo và định nghĩa thôi, khi nào có một cái *Subscribe()* được gọi, chúng mới chính thức được "nạp" và "chạy", mới bắt đầu "ăn" tài nguyên é. Ví dụ dễ thấy là, mấy bạn hoàn toàn có thể define các stream bên ngoài các hàm, để không đó. Khi nào cần mới subscribe.

Có mấy chỗ, mình cứ Subscribe() ra xài thế, rồi nó sẽ "ngưng" lại lúc nào ta ? Cái Object bị destroy rồi thì chắc ba cái IDisposable đó cũng tự dừng lại hé ?

Đây cũng là phần lưu ý quan trọng khi sử dụng UniRX, về việc quản lý và giải phóng các IDisposable sau khi Subscribe. Phần này các bạn đọc ở session *Observable Lifecycle Management (https://github.com/neuecc/UniRx#observable-lifecycle-management)* ngay trên Github của tác giả nhé.

Các bạn thử xem lại các ví dụ của mình, có phải các bạn sẽ tạo một MonoBehavior, paste script vào, attach lên một GameObject và Play + testing. Thử không stop editor, mà xóa object đó đi, mọi function sau khi Subscribe() vẫn còn hiệu lực !! Đó là lý do chúng ta cần chú ý *Dispose()*. Như ví dụ dưới đây, việc in log khi user pressMouse sẽ ngưng lại khi object bị destroy :

```
1  void Start()
2  {
3      var mousePressStream = Observable.EveryUpdate()
4          .Where(_ => Input.GetMouseButtonDown(0));
5      var logger = mousePressStream.Subscribe(_ => Debug.Log("Mouse press"));
6      logger.AddTo(_disposables);
7  }
8
9
10 CompositeDisposable _disposables = new CompositeDisposable();
11 void OnDestroy()
12 {
13     _disposables.Clear();
14 }
```

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

UniRX cũng có hỗ trợ một vài Operator như *TakeUntilDestroy(), TakeUntilDisable()*, hoặc Operator hỗ trợ cho cái IDisposable là *AddTo(gameObject/Component)* …. Thật ra, cái lõi của nó là Add thêm Component vào gameObject đó, và dùng các Observable của các component đó để dispose(), mục này các bạn tự đào code xuống để hiểu nhé. Cá nhân mình thích manual handle trong *OnDisable()* hoặc *OnDestroy()* ngay trong script của mình, vì khi thêm các component phụ, sẽ dẫn đến một số side effect nếu như trên cùng một GameObject có nhiều component dùng chúng.

Drawback, Reactive Programming có vẻ như hoàn hảo nhỉ?

Theo như một vài bài blog mà mình có tham khảo qua, thì RP đã có từ khá sớm, nhưng vì một số thứ, vừa có cả lý do lịch sử, "programming fashion" và cả về mặt phần cứng nữa, nên RP chưa được lên ngôi. Mình khá tâm đắc bài viết của tác giả bài viết Thời trang lập trình, Declarative Programming (https://dangthaison91.wordpress.com/2015/10/24/declarative-programming/), các bạn tham khảo thêm nhé. Mặc dù phải xác nhận là RP sẽ cần nhiều bộ nhớ hơn so với cách lập trình thông thường, tuy nhiên vấn đề phần cứng thời nay đã khiến khoảng cách bộ nhớ đó không còn đáng kể lắm, và so với những lợi ích mà RP đem lại thì từng đó cũng chả là gì.

Gom góp

Hiểu cho tổng quát vấn đề một tí

Reactive Programming :

Một "fashion" trong lập trình (Programming Paradigm), với nhiều điểm cộng hơn so với Imperative Programming : code ngắn gọn, hạn chế những side effect do việc đặt và thay đổi giá trị biến quá nhiều .v.v..
Hiểu theo kiểu "thợ xây" thì nó là một quá trình :

- Định nghĩa các stream, hay là các Observables
- Áp dụng các Operator lên chúng.
- Khi đã đạt yêu cầu về output, mình Subscribe.

UniRX framework:

Được viết bởi Yoshifumi Kawai, người Nhật, dựa trên bộ framework RX của .NET để có thể tương thích và vận hành trên Unity3D. Tất nhiên anh ấy đã phải viết lại + viết thêm khá nhiều mới ra được bộ UniRX.

UniRX cung cấp:

- Hầu hết các Operator để có thể nghịch ngợm. Tuy không phải tất cả operator đều đã được UniRX "chuyển thể" từ Rx.NET qua, nhưng nhiêu đó cũng đã là quá nhiều, quá đầy đủ cho hầu hết các nhu cầu rồi.
- Kha khá build-in Observables, một số cho Network, cả một số MonoBehavior trigger để "chuyển" các message của Unity thành Observable. MicroCoroutine feature, rất lighweigh và fast.
- Chú ý việc Dispose sau khi đã subscribe, và chú ý khi dùng các Repeat operator. Đây là các chú ý quan trọng để khỏi bị leak memory hoặc infinite loop.

Bắt đầu thế nào :

Hiểu Reactive Programming trước, sau đó hiểu UniRX nó làm cái gì, nó "liên quan" thế nào đến Unity, cách nó biến những thứ của Unity (message, event) thành các Observable. Rồi từ đó sẽ tự hiểu mình vận dụng UniRX thế nào cho ổn.

RP là một phạm trù rộng, bạn có thể còn gặp và ứng dụng trong nhiều mảng khác, ngôn ngữ khác, không riêng gì C# trong Unity. Vì thế, cứ nắm và hiểu RP cho vững nhé.

P/S

Tài liệu trên mạng nói về món Reactive Programming trong Game developmenent, mình chưa thấy nhiều lắm. Tiếp cận món RP này không phải dễ, chỉ có luyện từ từ thôi. Mình cũng dùng RX được một thời gian, có thể phần 4 mình sẽ đề cập tới chủ đề *"Làm thế nào setup project theo kiểu RP này"*, xoay quanh các câu hỏi như :

- Dùng tư duy như thế nào để setup ra các stream dựa trên các input
- Cài đặt các Global-stream (những stream sẽ được dùng trong toàn game) như thế nào ?
- Sử dụng RP này trong một hệ thống Component-System như thế nào là hợp lý ?
- Những trường hợp ngoại lệ, không nên gó buộc vào RP và các chuẩn mực của RP.
- ..v.v….

Cuối cùng là một số link tham khảo cho các bạn ngâm cứu :

- Một bài viết về RP rất hay : https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (https://gist.github.com/staltz/868e7e9bc2a7b8c1f754)
- Một slide đơn giản nhưng cũng thú vị của một tác giả nào đó, nói về UniRX http://slides.com/sammegidov/unirx#/7 (http://slides.com/sammegidov/unirx#/7)

- Một Slide khác, chuyên về chủ đề so sánh Imperative programming và Reactive Programming : https://www.slideshare.net/dangthaison91/introduction-to-functional-reactive-programming-62527220 (https://www.slideshare.net/dangthaison91/introduction-to-functional-reactive-programming-62527220)
- Một bài viết cũng khá hay của một blogger người Việt https://dangthaison91.wordpress.com/2015/10/24/declarative-programming/ (https://dangthaison91.wordpress.com/2015/10/24/declarative-programming/)

Published by **ThoXayLamCoder**

View all posts by ThoXayLamCoder

7 thoughts on “Reactive Programming và UniRX trong Unity – P3 : ngoài lề về UniRX”

1. Pingback: Reactive Programming và UniRX trong Unity – P1 – Thợ xây làm Coder
2. Pingback: Reactive Programming và UniRX trong Unity – P2 – Thợ xây làm Coder
3. **sinsamset** says:
   SEPTEMBER 7, 2017 AT 12:12 PM
   hay quá đại ca :v

   REPLY
4. **Đức Anh Lê** says:
   NOVEMBER 28, 2017 AT 9:32 AM
   rất cần part 4 của anh (/^w^)/

   REPLY
5. **Hải** says:
   JULY 4, 2019 AT 8:39 AM
   Cần doc của nó a ạ, chứ vọc chay chẳng biết mấy hàm của nó có tác dụng gì nữa là áp dụng

   REPLY

   ThoXayLamCoder says:
   JULY 4, 2019 AT 12:35 PM
   Có chứ, mình có ghi rõ ở phần 2 ấy mà
   ” Ứng với các ví dụ về Operator, mình đính link đến page ReactiveX, có phần định nghĩa cho từng Operator.”
   VD với cái Filter mình để link ngay trong bài ấy mà, như này nè :
   http://reactivex.io/documentation/operators/filter.html
   😉

REPLY

6. **ngoctuandl** says:
   JULY 13, 2019 AT 5:07 AM
   Quá ngon!

   Cái RX này mà phối hợp với CoReaction thì tuyệt hảo.
   Để mình mix nó lại thử coi hiểu đúng những gì bạn viết không.

   REPLY