

2.Unity 生命周期 与 Trigger

对于 Unity 的 Observable 增强，我们在第一章就接触过了。

Observable.EveryUpdate() 就是支持的 Unity 的 API。

单单 Update 就是支持非常多细分类型的 Update 事件捕获。

```
Observable.EveryFixedUpdate().Subscribe(_ => {});  
Observable.EveryEndOfFrame().Subscribe(_ => {});  
Observable.EveryLateUpdate().Subscribe(_ => {});  
Observable.EveryAfterUpdate().Subscribe(_ => {});
```

除了 Update 还支持其他的事件，比如 ApplicationPause, Quit 等。

```
Observable.EveryApplicationPause().Subscribe(paused => {});  
Observable.EveryApplicationFocus().Subscribe(focused => {});  
Observable.EveryApplicationQuit().Subscribe(_ => {}):
```

学习了以上这些，就不用再去创建一个单例类去实现一个诸如“应用程序退出事件监听”这种逻辑了。

命名几行代码就可以搞定的事情，何必再去创建一个类去搞定？

Trigger 简介

Observable.EveryUpdate() 这个 API 有的时候在某个脚本中实现，需要绑定 MonoBehaviour 的生命周期（主要是 OnDestroy），当然有的时候是全局的，而且永远不会被销毁的。

需要绑定 MonoBehaviour 生命周期的 EveryUpdate。只需要一个 AddTo 就可以进行绑定了。非常简单，代码如下。

```
Observable.EveryUpdate()  
    .Subscribe(_ => {})  
    .AddTo(this);
```

但其实有更简洁的实现:

```
this.UpdateAsObservable()  
    .Subscribe(_ => {});
```

这种类型的 Observable 是什么呢?

答案是: Trigger, 即触发器。

字如其意, 很容易理解。

Trigger 类型的关键字

触发器, 字如其意, 是当某个事件发生时, 则会将该事件发送到 Subscribe 函数中, 而这个触发器, 本身是一个功能脚本, 这个脚本挂在 GameObject 上, 来监听 GameObject 的某个事件发生, 事件发生则会回调给注册它的 Subscribe 中。

触发器的操作和其他的事件源 (Observable) 是一样的, 都支持 Where、First、Merge 等操作符。

Trigger 类型的 Observable 和我们之前讲的所有的 Observable 在表现上有一点点不一样:

1. Trigger 大部分都是都是 XXXAsObservable 命名形式的。
2. 在使用 Trigger 的 GameObject 上都会挂上对应的 Observable XXXTrigger.cs 的脚本。

Trigger 在此之前我们是接触过的。

AddTo() 这个 API 其实是封装了一种 Trigger: ObservableDestroyTrigger。

顾名思义，就是当 GameObject 销毁时获取事件的一个触发器。

一般的 Trigger 都会配合 MonoBehaviour 一起使用。

比如 ObservableDestroyTrigger 的使用代码如下：

```
this.OnDestroyAsObservable()  
    .Subscribe(_ => {});
```

除了 Destroy 还有非常多的 Trigger。

比如各种细分类型的 Update：

```
this.FixedUpdateAsObservable().Subscribe(_ => {});  
this.LateUpdateAsObservable().Subscribe(_ => {});  
this.UpdateAsObservable().Subscribe(_ => {});
```

还有各种碰撞的 Trigger：

```
this.OnCollisionEnterAsObservable(collision => {});  
this.OnCollisionExitAsObservable(collision => {});  
this.OnCollisionStayAsObservable(collision => {});  
// 同样 2D 的也支持  
this.OnCollision2DEnterAsObservable(collision2D => {});  
this.OnCollision2DExitAsObservable(collision2D => {});  
this.OnCollision2DStayAsObservable(collision2D => {});
```

一些脚本的参数监听：

```
this.OnEnableAsObservable().Subscribe(_ => {});  
this.OnDisableAsObservable().Subscribe(_ => {});
```

除了 `MonoBehaviour`，`Trigger` 也支持了其他组件类型，比如 `RectTransform`、`Transform`、`UIBehaviour` 等等。这里不再赘述。

详情可以查看 `ObservableTriggerExtensions.cs` 和 `ObservableTriggerExtensions.Component.cs` 中的 API。

今天的内容就这些。