9.MVP 实现

上堂课我们学习了 UniRx 中的 ReactiveProperty，今天呢，我们接着深入，我们来实现一个简洁的 MVP 模式框架。

在上一讲有简单介绍过，UniRx 对 UGUI 进行了增强。

UGUI 增强

UGUI 增强的原理很简单，就是对 UnityEvent 提供了 AsObservable 方法。

代码如下：

```
public Button mButton;

mButton.onClick.AsObservable().Subscribe(_ => Debug.Log("clicked"));
```

在此基础上，进一步对每个 UGUI 控件进行封装，从而可以像如下方式在 UGUI 中使用 UniRx。

```
public Toggle mToggle;
public InputField mInput;
public Text mText;
public Slider mSlider;

void Start()
{
    mToggle.OnValueChangedAsObservable().SubscribeToInteractable(mButton);

    mInput.OnValueChangedAsObservable()
        .Where(x => x != null)
        .SubscribeToText(mText);
```

```
    mSlider.OnValueChangedAsObservable()
        .SubscribeToText(mText,x => Math.Round(x,2).ToString());
}
```

这段内容理解起来很简单。

ReactiveProperty,ReactiveCollection

我们在实现 MVP 模式之前，先看下以下代码。

```
// Reactive Notification Model
public class Enemy
{
    public ReactiveProperty<long> CurrentHp { get; private set;}
    public ReactiveProperty<bool> IsDead {get;private set;}

    public Enemy(int initialHp)
    {
        // Declarative Property
        CurrentHp = new ReactiveProperty<long>(initialHp);
        IsDead = CurrentHp.Select(x => x <= 0).ToReactiveProperty();
    }
}

void Start()
{
    mButton.OnClickAsObservable().Subscribe(_ => enemy.CurrentHp.Value -= 99);

    enemy.CurrentHp.SubscribeToText(MyText);
    enemy.IsDead.Where(isDead => isDead)
                .Subscribe(_=>
                {
                    mButton.interactable = false;
                });

}
```

这段代码理解起来非常简单，Enemy 是一个数据类，我们可以理解成 Model。

而 下边的 Start 部分则是 Ctrl 的代码。它将 Hierarchy 中的 UI 控件 与 Model 绑定在了一起。当 Model 有改变则通知 UI 更新，当从 UI 接收到点击事件则对 Model 进行值的更改。这就是一个非常简单的 MVP 模式。

你可以用 UnityEvent.AsObservable 将 ReactiveProperties，ReactiveCollections 和 Observables 都组合起来。所有 UI 组件都提供了 XXXAsObservable

在 Unity 里，序列化是一个很重要的功能，如果不可序列化，则在编辑器上就看不到参数。而 ReactiveProperty 是泛型的，序列化起来比较麻烦。为了解决这个问题，UniRx 支持了可序列化的 ReactiveProperty 类型，比如 Int/LongReactivePropety、Float/DoubleReactiveProperty、StringReactiveProperty、BoolReactiveProperty,还有更多，请参见 InspectableReactiveProperty.cs。

以上都可以在 Inspector 中编辑。对于自定义的枚举 ReactiveProperty，写一个可检视的 ReactiveProperty[T] 也很容易。

如果你需要 [Multiline] 或者[Range] 添加到 ReactiveProperty 上，你可以使用 MultilineReactivePropertyAttribute 和 RangeReactivePropertyAttribute 替换 Multiline 和 Range。

这些 InspectableReactiveProperties 可以在 inspector 面板显示，并且当他们的值发生变化时发出通知，甚至在编辑器里变化也可以。

这个功能是实现在 InspectorDisplayDrawer。你可以通过继承这个类实现你自定义的 ReactiveProperties 在 Inspector 面板的绘制：

```
public enum Fruit
{
    Apple,Grape
}

[Serializable]
public class FruitReactiveProperty : ReactiveProperty<Fruit>
{
    public FruitReactiveProperty()
    {
    }
```

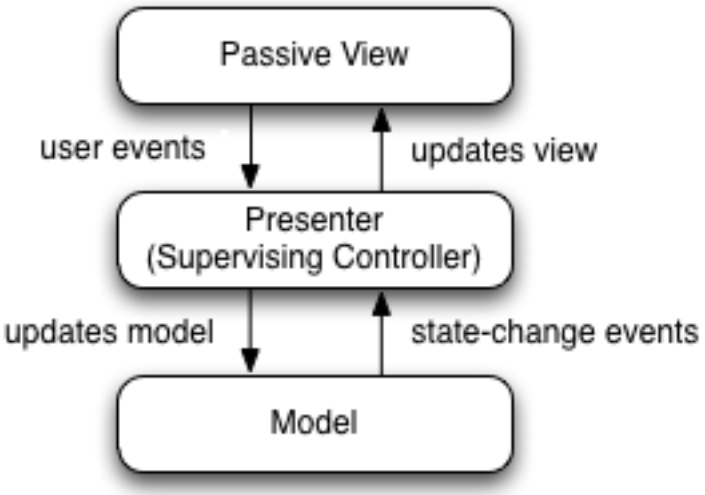
```
    public FruitReactiveProperty(Fruit initialValue) : base(initialValue)
    {
    }
}

[UnityEditor.CustomPropertyDrawer(typeof(FruitReactiveProperty))]
[UnityEditor.CustomPropertyDrawer(typeof(YourSpecializedReactiveProperty2))] //
// and others...
public class ExtendInspectorDisplayDrawer : InspectorDisplayDrawer
{
}
```

如果 ReactiveProperty 的值只在 Stream 中更新，你可以使用 ReadOnlyReactiveProperty 让这个属性只读。

MVP 设计模式 Model-View-（Reactive）Presenter Pattern

使用 UniRx 可以很容易地实现 MVP（MVRP）设计模式。

MVP 的结构图如下所示。



为什么应该用 MVP 模式而不是 MVVM 模式？Unity 没有提供 UI 绑定机制，创建一个绑定层过于复杂并且会对性能造成影响（使用反射）。尽管如此，视图还是需要更新。Presenters 层知道 View 组件并且能更新它们。

虽然没有真的绑定，但 Observables 可以通知订阅者，功能上也差不多。这种模式叫做 Reactive Presenter：

```
// Presenter for scene(canvas) root.
public class ReactivePresenter : MonoBehaviour
{
    // Presenter is aware of its View (binded in the inspector)
    public Button mButton;
    public Toggle mToggle;

    // State-Change-Events from Model by ReactiveProperty
    Enemy enemy = new Enemy(1000);

    void Start()
    {
        // Rx supplies user events from Views and Models in a reactive manner
        mButton.OnClickAsObservable().Subscribe(_=>enemy.CurrentHp.Value -=
99);
        mToggle.OnValueChangedAsObservable().SubscribeToInteractable(mButton);

        // Models notify Presenters via Rx,and Presenters update their views
        enemy.CurrentHp.SubscribeToText(MyText);
        enemy.IsDead.Where(isDead => isDead)
            .Subscribe(_=>
            {
                mToggle.interactable = mButton.interactable = false;
            });
    }
}

// The Mode. All property notify when their values change
public class Enemy
{
    public ReactiveProperty<long> CurrentHp { get;private set;}
    public ReactiveProperty<bool> IsDead { get;private set;}
```

```
    public Enemy(int initialHp)
    {
        // Declarative Property
        CurrentHp = new ReactiveProperty<long>(initialHp);
        IsDead = CurrentHp.Select(x => x <= 10).ToReactiveProperty();
    }
}
```

在 Unity 中，我们把 Scene 中的 GameObject 当做视图层，这些是在 Unity 的 Hierarchy 中定义的。

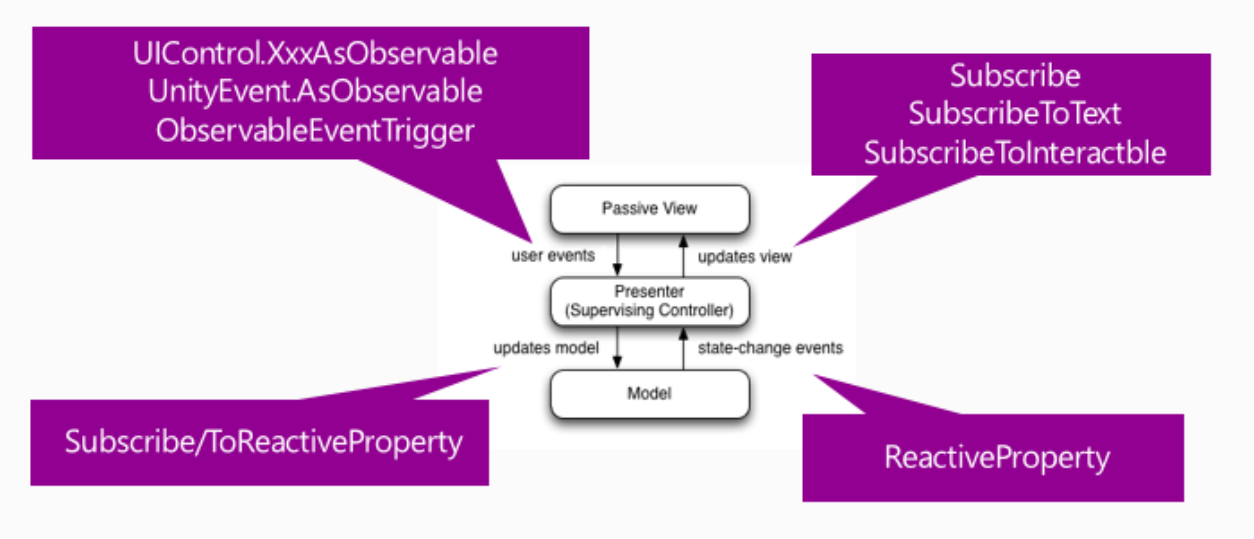
展示/控制层在 Unity 初始化时将视图层绑定。

SubscribeToText and SubscribeToInteractable 都是简洁的类似绑定的辅助函数。虽然这些工具很简单，但是非常实用。

在 Unity 中使用开发体验非常平滑，性能也很好，最重要的是让你的代码更简洁。

View -> ReactiveProperty -> Model -> RectiveProperty - View 完全用响应式的方式连接。UniRx 提供了所有的适配方法和类，不过其他的 MVVM (or MV*) 框架也可以使用。UniRx/ReactiveProperty 只是一个简单的工具包。

使用 UniRx 实现的 MVP 模式结构图如下：



今天的内容就这些。