3.IObservable 和 IObserver 的简单使用

IObservable 用来定义可观察的类，而 IObserver 用来定义观察者类。

观察者模式，我们最容易接触的应用场景就是 MVC。

在 MVC 中，Model 通常是在多个 Controller/View 之间是共享的。

也就是一个 Model 经常要对应多个 View。

当 Model 中的数据有更改时，View 中的数据显示就会跟着更新。

IObservable 和 IObserver，一个是形容可观察的类，一个是观察者。

理解起来很容易，可观察的类就是应对发布者，而观察者应对订阅者。

那么 Model 和 View 与 IObservable 和 IObserver 对应关系是如何呢?

Model 是发布者对应的是 IObservable

View 是订阅者对应的是 IObserver

我们先来定义一个 Model 类。数据比较简单，只管理一个数据 Level(等级)

先看下 IObservable 接口的定义:

```
public interface IObservable<T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

只要实现 Subseribe 方法就好了。

而 T 则是感兴趣的数据类型，可以是一个类、结构体或者一个值。Level 是 int 类型，所以 T 是 int 类型。

代码如下:

```
public class Model : IObservable<int>
{
    public Model()
    {
        mObservers = new List<IObserver<int>>();
    }

    private readonly List<IObserver<int>> mObservers;

    public IDisposable Subscribe(IObserver<int> observer)
    {
        if (!mObservers.Contains(observer))
            mObservers.Add(observer);
        return new Unsubscriber(mObservers, observer);
    }

    public void UpdateLevel(int loc)
    {
        foreach (var observer in mObservers)
        {
            if (loc < 0)
                observer.OnError(new LevelUnknownException());
            else
                observer.OnNext(loc);
        }
    }

    public void QuitGame()
    {
        foreach (var observer in mObservers.ToArray())
            if (mObservers.Contains(observer))
                observer.OnCompleted();

        mObservers.Clear();
    }
```

```
        private class Unsubscriber : IDisposable
        {
            private List<IObserver<int>> mObservers;
            private IObserver<int>       mObserver;

            public Unsubscriber(List<IObserver<int>> observers, IObserver<int>
observer)
            {
                this.mObservers = observers;
                this.mObserver = observer;
            }

            public void Dispose()
            {
                if (mObserver != null && mObservers.Contains(mObserver))
                    mObservers.Remove(mObserver);
            }
        }
    }
```

我们在看下，IObserver 的定义

```
public interface IObserver<T>
{
    void OnCompleted();

    void OnError(Exception error);

    void OnNext(T value);
}
```

这三个方法，全部都是被 IObservable 调用。当有数据更新时则调用 OnNext，这样实现 IObserver 并对 IObservable 订阅的类，则会接收到更新的数据。

当结束数据发送则 OnCompleted 被调用，当出现异常则 OnErrer 被调用。

接下来定义 View.cs

```
public class View : IObserver<int>
{
    private IDisposable unsubscriber;
    private string      instName;

    public View(string name)
    {
        this.instName = name;
    }

    public string Name
    {
        get { return this.instName; }
    }

    public void OnCompleted()
    {
        Debug.LogFormat("The Model has completed update data to {0}.",
this.Name);
    }

    public void OnError(Exception e)
    {
        Debug.LogFormat("{0}: The level cannot be determined.", this.Name);
    }

    public void OnNext(int value)
    {
        Debug.LogFormat("{1}: The current level is {0}", value, this.Name);
    }
}
```

测试代码如下:

```
/*************************************************************************
 * http://sikiedu.com liangxie
```

```
 ***********************************************************************/

using System;
using System.Collections.Generic;
using UnityEngine;

namespace UniRxLesson
{
    public class SimpleObservableExample : MonoBehaviour
    {
        private void Start()
        {
            var model = new Model();

            var loginView = new View("Login");
            var unsubsriber = model.Subscribe(loginView);

            var userInfoView = new View("UserInfo");
            model.Subscribe(userInfoView);

            model.UpdateLevel(1);

            unsubsriber.Dispose();

            model.UpdateLevel(2);
            model.UpdateLevel(-1);
            model.QuitGame();
        }
    }
}
```

输出结果为:

```
Login: The current level is 1
UserInfo: The current level is 1
UserInfo: The current level is 2
UserInfo: The level cannot be determined.
The Model has completed update data to UserInfo.
```

今天的内容就这些。