https://blog.csdn.net/inlet511/article/details/46822907?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task

UnityAction本质上是delegate，且有数个泛型版本（参数最多是4个）,一个UnityAction可以添加多个函数（多播委托）

UnityEvent本质上是继承自UnityEventBase的类，它的AddListener()方法能够注册UnityAction，RemoveListener能够取消注册UnityAction，还有Invoke()方法能够一次性调用所有注册了的UnityAction。UnityEvent也有数个泛型版本（参数最多也是4个），但要注意的一点是，UnityAction的所有带参数的泛型版本都是抽象类（abstract），所以如果要使用的话，需要自己声明一个类继承之，然后再实例化该类才可以使用。

代码范例：

使用UnityEvent 和 UnityAction：

```
using UnityEngine;
using System.Collections;
using UnityEngine.Events;

public class UnityActionAndEvent : MonoBehaviour {

    public UnityAction action;
    public UnityEvent myEvent = new UnityEvent();

    void Start()
    {
        action = new UnityAction(MyFunction);
        action+=MyFunction2;
        myEvent.AddListener(action);
    }

    void Update()
```

```
    {
        if(Input.GetKeyDown(KeyCode.P))
        {
            myEvent.Invoke();
        }
    }

    public void MyFunction()
    {
        print ("Hello: ");
    }

    public void MyFunction2()
    {
        print ("Hello2: ");
    }

}
```

使用带参数的UnityEvent

```
using UnityEngine;
using System.Collections;
using UnityEngine.Events;

//因为UnityEvent<T0>是抽象类，所以需要声明一个类来继承它
public class MyEvent:UnityEvent<int>{}

public class UnityActionWithParameter : MonoBehaviour {

    public MyEvent myEvent = new MyEvent();
    public UnityAction<int> action;

    void Start () {
        action= new UnityAction<int>(MyFunction);
```

```
        action += MyFunction2;
        myEvent.AddListener(action);
    }

    void Update () {
        if(Input.GetKeyDown(KeyCode.A))
        {
            myEvent.Invoke(10);
        }
    }

    public void MyFunction(int i)
    {
        print (i);
    }
    public void MyFunction2(int i)
    {
        print(i*2);
    }
}
```

据这篇帖子10楼所说，Unity中通过面板中添加的Listener和通过脚本添加的Listener实际上是两种不同类型的Listener：

在脚本中通过AddListener()添加的是一个0个参数的delegate(UnityAction)回调。是不可序列化的，在Inspector中是无法看到的。这种Listener是常规Listener。
在Inspector中添加的则是永久性的Listener（persistent listener)。他们需要指定GameObject、方法以及方法需要的参数。他们是序列化的，用脚本是无法访问到的。
另外在脚本中使用lamda表达式来添加listener是非常方便的，可以添加任意多个参数的函数。
下面是脚本范例：

```
using UnityEngine;
using System.Collections;
using UnityEngine.EventSystems;
```

```
using UnityEngine.UI;

public class EventAndLamda : MonoBehaviour {



    void Start () {
        //lamda方式可以添加包含任意参数的函数，非常方便
        GetComponent<Button>().onClick.AddListener(()=>{
            //此处其实可以直接写Myfuction(.....)，因为是同一个脚本里的函数
            //这样写是为了说明调用其他组件中的函数也是可以的。如果有其他组件的引用，可以直接写：
            //someReference.theMethod(arguments);
            this.GetComponent<EventAndLamda>().MyFunction(10,20.0f,new Vector3(1,1,1));
        });
    }

    public void MyFunction(int i, float f, Vector3 v)
    {
        print (i.ToString()+"\n" + f.ToString() + "\n" + v.ToString());
    }
}
```