自定义比较排序器

1 Comparer 非泛型比较器接口 直接比较两个object 可以比较两个不同类型的对象

```
using System;
using System.Runtime.InteropServices;

namespace System.Collections
{
    [ComVisible(true)]
    public interface IComparer
    {
        int Compare(object x, object y);
    }
}
```

2 实现IComparer<T> 泛型接口

```
using System;

namespace System.Collections.Generic
{
    public interface IComparer<T>
    {
        int Compare(T x, T y);
    }
}
```

游戏中的好友栏显示排序功能：

1 在线好友显示在离线好友之上

2 在线状态相同时，友好度高的显示在友好度低的好友之上

3 在线状态和友好度都相同时 等级高的好友显示在等级低的好友之上

```
//好友状态信息结构
public class FriendState
{
    public bool online;//是否在线
    public int level;//等级
    public float friendliness;//友好度

    public FriendState(bool online,int level,float friendliness)
    {
        this.online = online;
        this.level = level;
        this.friendliness = friendliness;
    }
}

public class FriendComparer:IComparer<FriendState>
{
    //排序原则:  左右两个对象进行比较 x,y
    //当返回-1时 相等于x索引值减少 排在前面
    //当返回0时 相当于x索引值不变 排序相同
    //当返回1时 相当于x索引值增加 排在后面
    public int Compare(FriendState x,FriendState y)
    {
        if (x.level > y.level)
            return -1;
        if (x.level < y.level)
            return 1;
        if (x.online && !y.online)
            return -1;
        if (!x.online && y.online)
            return 1;
        if (x.friendliness > y.friendliness)
            return -1;
        if (x.friendliness < y.friendliness)
            return 1;

        return 0;
    }
}

public class Text : MonoBehaviour {

```

```
    List<FriendState> friends = new List<FriendState>();
    void Start () {

        friends.Add(new FriendState(false, 10, 100));
        friends.Add(new FriendState(true, 20, 100));
        friends.Add(new FriendState(true, 30, 200));
        friends.Add(new FriendState(true, 30,300));
        friends.Add(new FriendState(true, 40, 100));
        friends.Add(new FriendState(false, 20, 300));

        friends.Sort(new FriendComparer());

        foreach (FriendState state in friends)
        {
            Debug.Log(state.online + ":" + state.level + ":" +
state.friendliness);
        }
    }
}
```

练习： 商品排序

{

折扣

enum 商品类型（VIP商品 普通商品）

int 商品剩余数量

}

排序原则:

1 有剩余数量的商品 显示在无剩余数量商品之上

2 折扣力度更大的商品 显示在折扣力度小的商品之上

3 vip商品显示在普通商品之上

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

```
4
5  public enum GoodsType
6  {
7      vip,
8      normal
9  }
10
11 public class Goods
12 {
13     public float disCount;//折扣
14     public int count;//数量
15     public GoodsType goodsType;//商品类型
16
17     public Goods(float disCount,int count,GoodsType goodsType)
18     {
19         this.disCount = disCount;
20         this.count = count;
21         this.goodsType = goodsType;
22     }
23 }
24
25 public class GoodsComparer : IComparer<Goods>
26 {
27     public int Compare(Goods x,Goods y)
28     {
29         if (x.count > 0 && y.count <= 0)
30             return -1;
31         if (x.count <= 0 && y.count > 0)
32             return 1;
33
34         if (x.disCount < y.disCount)
35             return -1;
36         if (x.disCount > y.disCount)
37             return 1;
38
39         if (x.goodsType < y.goodsType)
40             return -1;
41         if (x.goodsType > y.goodsType)
42             return 1;
43
44         return 0;
45     }
```

```
}

public class Text : MonoBehaviour {

    List<Goods> goods = new List<Goods>();
    void Start () {

        goods.Add(new Goods(1f, 5, GoodsType.vip));
        goods.Add(new Goods(0.3f, 3, GoodsType.vip));
        goods.Add(new Goods(0.5f, 5, GoodsType.vip));
        goods.Add(new Goods(0.4f,0, GoodsType.normal));
        goods.Add(new Goods(1f, 5, GoodsType.vip));
        goods.Add(new Goods(0.7f, 0, GoodsType.normal));
        goods.Add(new Goods(1f, 5, GoodsType.normal));

        goods.Sort(new GoodsComparer());

        foreach (Goods g in goods)
        {
            Debug.Log("剩余数量: " + g.count + " 折扣:" + g.disCount +
" Type:" + g.goodsType);
        }

    }

}

```

委托 delegate

委托是一种数据类型

可以把方法当做是一种变量进行存储(C函数指针)

委托的类型决定存储的方法类型

一般情况下 实际开发中用于实现回调函数

战争类游戏：

游戏可能包含多个兵种：骑兵 步兵 弓箭手 炮兵等等

玩家可以选择不同的战术

1 第一套方案 弓箭手进行远程攻击 骑兵原地待命

2 第二套方案 骑兵冲锋 弓箭手原地待命

当玩家的选择发生改变时 需要告诉通知很多不同兵种的对象

这里使用委托实现更为方便

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

//定义一个没有返回值的无参的委托
public delegate void myDelegate1();
//定义一个没有返回值的有一个string参数的委托
public delegate void myDelegate2(string data);

public class Player : MonoBehaviour {

    public myDelegate1 click1;
    public myDelegate2 click2;
    void OnGUI()
    {
        if (GUILayout.Button("第一套作战方案"))
        {
            if (click1 != null)
                click1();
        }
        if (GUILayout.Button("第二套作战方案"))
        {
            if (click2 != null)
                click2("第二套");
        }
    }
}

```

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class Bowman : MonoBehaviour {

    public Player player;
    void Start () {
        //订阅 设置委托所指向方法

        //使用=给委托赋值 不能同时存储多个回调
        //player.click1 = Idle;

        //需要存储多个回调使用+=给委托赋值
        player.click1 += Idle;
        player.click2 += Attack;
    }

    void Idle()
    {
        Debug.Log("弓箭手待命");
    }
    void Attack(string text)
    {
        Debug.Log("弓箭手攻击"+text);
    }
}

```

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

public class Cavalry : MonoBehaviour {

    public Player player;
    void Start () {
        player.click1 += cavalryAttack;
        player.click2 += cavalryIdle;
    }

    void cavalryAttack()
```

```
    {
        Debug.Log("骑兵冲锋");
    }
    void cavalryIdle(string text)
    {
        Debug.Log("骑兵原地待命"+text);
    }
}

```

在Unity中使用委托需要注意的一点

当物体在被设置为非激活时 委托依然会调用存储的回调方法

如果希望物体不激活时 不调用回调 那么订阅和取消订阅 应该在OnEnable和OnDisable里实现

```
public class Cavalry : MonoBehaviour {

    public Player player;
    void OnEnable () {
        //订阅
        player.click1 += cavalryAttack;
        player.click2 += cavalryIdle;
    }

    void OnDisable()
    {
        //取消订阅
        player.click1 -= cavalryAttack;
        player.click2 -= cavalryIdle;
    }

    void cavalryAttack()
    {
        Debug.Log("骑兵冲锋");
    }
    void cavalryIdle(string text)
    {
        Debug.Log("骑兵原地待命"+text);
    }
```

```
}
```

Event 事件

把委托看成是一个类型 事件可以理解为是某个委托类型的对象

在一个类中定义了一个事件 那么个事件的触发就必须在事件定义的类中调用 不能再类外部被主动调用

事件对外部来说 只能通过使用+= -=来注册方法和移除注册 不能通过=赋值

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

//定义一个没有返回值的无参的委托
public delegate void myDelegate1();
//定义一个没有返回值的有一个string参数的委托
public delegate void myDelegate2(string data);

public class Player : MonoBehaviour {

    public event myDelegate1 click1;
    public event myDelegate2 click2;
    void OnGUI()
    {
        if (GUILayout.Button("第一套作战方案"))
        {
            if (click1 != null)
                click1();
        }
        if (GUILayout.Button("第二套作战方案"))
        {
            if (click2 != null)
                click2("第二套");
        }
    }
}
```

```
public class Bowman : MonoBehaviour {

    public Player player;
    void Start () {

        //报错 提示只能出现在+= -=左边
        player.click1 = Idle;

        //需要存储多个回调使用+=给委托赋值
        player.click1 += Idle;
        player.click2 += Attack;

        //报错 外部不能调用非本类定义的事件
        player.click1();

    }

    void Idle()
    {
        Debug.Log("弓箭手待命");
    }
    void Attack(string text)
    {
        Debug.Log("弓箭手攻击"+text);
    }
}
```

委托事件在NGUI里的实际应用

在NGUI中 一个按钮被按下时 一般通过OnClick函数回调通知开发者

OnClick等其他所有的事件方法 都是通过委托实现的

UIButton 在Notify里设置对应的回调 实际就是给委托添加订阅

当UIEventListener的对应委托被调用时 notify设置的函数也会被调用

```
public class UIEventListener : MonoBehaviour
{
    public delegate void VoidDelegate (GameObject go);
    public delegate void BoolDelegate (GameObject go, bool state);
    public delegate void FloatDelegate (GameObject go, float delta);
    public delegate void VectorDelegate (GameObject go, Vector2 delta);
    public delegate void ObjectDelegate (GameObject go, GameObject obj);
    public delegate void KeyCodeDelegate (GameObject go, KeyCode key);

    public object parameter;

    public VoidDelegate onSubmit;
    public VoidDelegate onClick;
    public VoidDelegate onDoubleClick;
    public BoolDelegate onHover;
    public BoolDelegate onPress;
    public BoolDelegate onSelect;
    public FloatDelegate onScroll;
    public VoidDelegate onDragStart;
    public VectorDelegate onDrag;
    public VoidDelegate onDragOver;
    public VoidDelegate onDragOut;
    public VoidDelegate onDragEnd;
    public ObjectDelegate onDrop;
    public KeyCodeDelegate onKey;
    public BoolDelegate onTooltip;
```

Lambda表达式 匿名函数

高效的类似于函数式编程的表达式

Lambda表达式减少了开发过程中需要编写的代码量 一般用于委托或者事件

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Cavalry : MonoBehaviour {
6
7      public Player player;
8      void OnEnable () {
9          //lambda表达式第一种写法
10         player.click1 += delegate
```

```
                        {
                            Debug.Log(100);
                        };

        player.click2 += delegate(string x)
                         {
                             Debug.Log(x);
                          };


        //第二种lambda表达式写法
        player.click1 += ()=>{Debug.Log(100);};
        player.click2 += (string x) =>
                        {
                            Debug.Log(x);
                        };

    }


}

```