# 数据绑定

## 实现具有通知外界自身属性的值发生变化能力的对象

class Student

{

    public string Name { get; set; }

}

上述的Student，在修改Name后，其他对象无法感知其Name的值发生了变化。

发布订阅模式，当对象内部发生了某个事件，可以发布出去，订阅者就可以感知并响应。如果使用发布订阅模式来实现具有通知外界自身属性的值发生变化能力的对象，则

发布者 – 属性发生变化的对象

订阅者 – 任何对象

发布事件 – 某某对象的某某属性的值发生了变化

事件参数 – 发布者对象引用，值发生了变化的属性的名称

发布时机 – Setter访问器

按照.NET的标准事件模型，可以有如下实现：

// 事件参数

public class PropertyChangedEventArgs : EventArgs

{

    public PropertyChangedEventArgs(string propertyName)

    {

        PropertyName = propertyName;

    }

    // 值发生变化的属性的标识符

    public string? PropertyName { get; }

}

// 定义事件

public delegate void PropertyChangedEventHandler(object? sender, PropertyChangedEventArgs e);

public class Student

{

    private string \_name;

    public string Name

    {

        get => \_name;

        set

        {

            \_name = value;

            OnPropertyChanged(nameof(Name));

        }

    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)

    {

        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

    }

    static void Main()

    {

        Student s = new Student();

        s.PropertyChanged += S\_PropertyChanged;

    }

    private static void S\_PropertyChanged(object? sender, PropertyChangedEventArgs e)

    {

        Type type = sender.GetType();

        object newValue = type.GetProperty(e.PropertyName).GetValue(sender);

        Console.WriteLine(newValue);

    }

}

.NET已经内置定义了事件类和事件参数类，并以INotifyPropertyChanged接口行程约定。

public class Student : INotifyPropertyChanged

{

    private string \_name;

    public string Name

    {

        get => \_name;

        set

        {

            \_name = value;

            OnPropertyChanged(nameof(Name));

        }

    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)

    {

        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

    }

    static void Main()

    {

        Student s = new Student();

        s.PropertyChanged += S\_PropertyChanged;

    }

    private static void S\_PropertyChanged(object? sender, PropertyChangedEventArgs e)

    {

        Type type = sender.GetType();

        object newValue = type.GetProperty(e.PropertyName).GetValue(sender);

        Console.WriteLine(newValue);

    }

}

## 数据源无法刷新依赖对象的原因