

# 数据绑定概述

---

北京理工大学计算机学院  
金旭亮

# 什么是数据绑定

---

# 数据绑定引例

什么是数据绑定？-纯手工实现

客户姓名: 刘克平

邮政编码: 100084

客户地址: 海淀区学院路37号北京航空航天大学收发室

装入数据 刷新显示 保存用户修改 直接修改对象 显示对象当前值

```
{
  "ClientID": 1,
  "ClientName": "刘克平",
  "Address": "海淀区学院路37号北京航空航天大学收发室",
  "PostCode": "100084",
  "Telephone": "",
  "Email": ""
}
```

什么是数据绑定？- 使用数据绑定机制

客户姓名: 刘克平

邮政编码: 100084

客户地址: 海淀区学院路37号北京航空航天大学收发室

装入数据 直接修改对象 显示对象当前值

```
{
  "ClientID": 1,
  "ClientName": "刘克平",
  "Address": "海淀区学院路37号北京航空航天大学收发室",
  "PostCode": "100084",
  "Telephone": "",
  "Email": ""
}
```

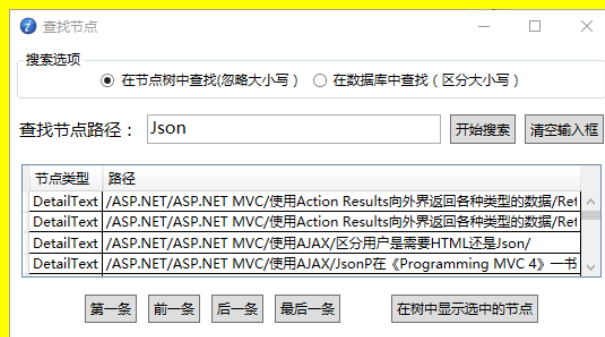
对比两个窗体可以发现：

使用数据绑定不仅在使用上更为简单，还可以精简代码，并且随着程序功能的增加，其优势更为明显。

# 什么是数据绑定？

## 数据绑定

使用数据填充用  
户界面



收集修改过的数  
据

将数据获取到内  
存中

永久保存数据

数据存储

# 数据绑定中的“源”与“目标”



数据源  
(DataSource)



数据绑定



UI界面控件

# 实际开发中应用数据绑定

MainWindow

Customers: Derek Puckett

First Name: Puckett Last Name: Derek Email: derek.puckett@outdate.net

ID	CustomerId	OrderDate	OrderStatusId	ItemsTotal
1	7462c7c8-e24c-484a-8993-013f1c479615	3/12/2013 5:00:00 PM	2	31.9500
2	7462c7c8-e24c-484a-8993-013f1c479615	4/15/2013 1:00:00 AM	2	31.6500
3	7462c7c8-e24c-484a-8993-013f1c479615	5/5/2013 11:00:00 PM	3	66.8000
4	7462c7c8-e24c-484a-8993-013f1c479615	5/25/2013 8:10:00 PM	2	15.1000

Order ID: 1

Order Date: 3/12/2013 5:00:00 PM

Order Total: 31.9500

Order Status:

Read



Update  
Create  
Delete

数据源对象

“数据绑定”技术  
负责这块

Database

Web  
Service

File

“数据存取技术”  
负责这块

## 学习与应用数据绑定技术的好处

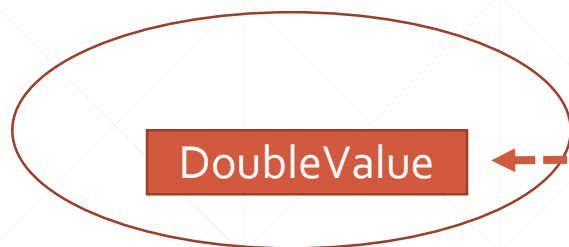
- 数据绑定机制降低了应用程序各组件之间的关联，使它们可以各自地独立变化。
- 大幅度地减少代码。
- 知识迁移，举一反三。

# 探索Binding对象

---



# Binding对象的使用示例



数据对象

The screenshot shows a window titled 'Binding对象探秘'. It contains a text input field labeled 'DoubleValue值' with the value '0.84'. Below it are two buttons: '设定绑定数据' (highlighted with a blue border) and '更改绑定对象属性值'. To the right, under '数据源对象', there is a checkbox labeled '实现INotifyPropertyChanged接口' which is checked. At the bottom right is a '清空' button. A large text area displays event logs: '在Format事件中 : Value:0.839215116034828, DesiredType:System.String' and '在BindingComplete事件中 : BindingCompleteContext:ControlUpdate, BindingCompleteState:Success, Cancel:False, ErrorText:, Exception:无'. A dashed black line connects the 'DoubleValue' label in the diagram to the 'DoubleValue值' input field.

Binding对象实际上“代表了一个数据绑定控件的某个属性与一个数据对象的相应属性之间的绑定关系”

# Binding对象的关键属性

```
public Binding(string propertyName, object dataSource, string dataMember);
```

1

2

3

1 参与绑定的数据绑定控件的属性，比如TextBox控件的Text属性

2 要绑定的数据源，可以是单个对象，也可以是对象集合

3 用于在数据源中定位数据对象的被绑定属性的“**导航路径 (navigation path)**”，比如“数据对象的某个属性名”

# 控件的DataBindings属性

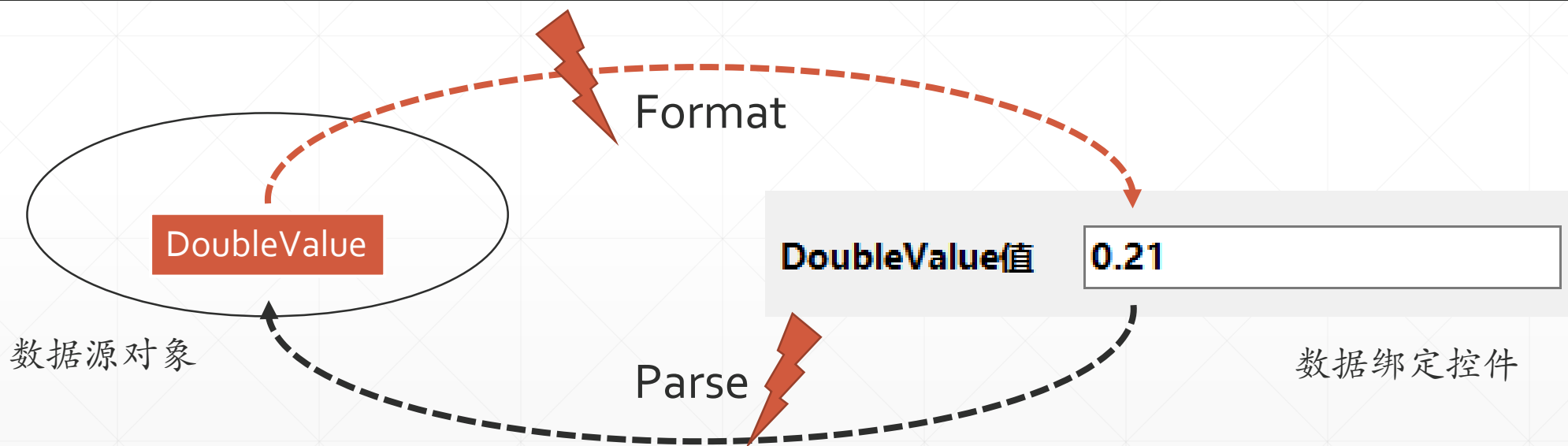
支持绑定的Windows Forms控件，有一个**DataBindings**集合，里面可以包容多个Binding对象。



```
Binding doubleBind = new Binding("Text", DataObject, "DoubleValue");  
.....  
txtDouble.DataBindings.Add(doubleBind);
```

# Binding对象的职责

Binding对象实际上完成的是从数据源中“**抽取**”数据传给数据绑定控件，以及将数据绑定控件中“**取回**”数据保存到数据源的这两个相互对应的功能。

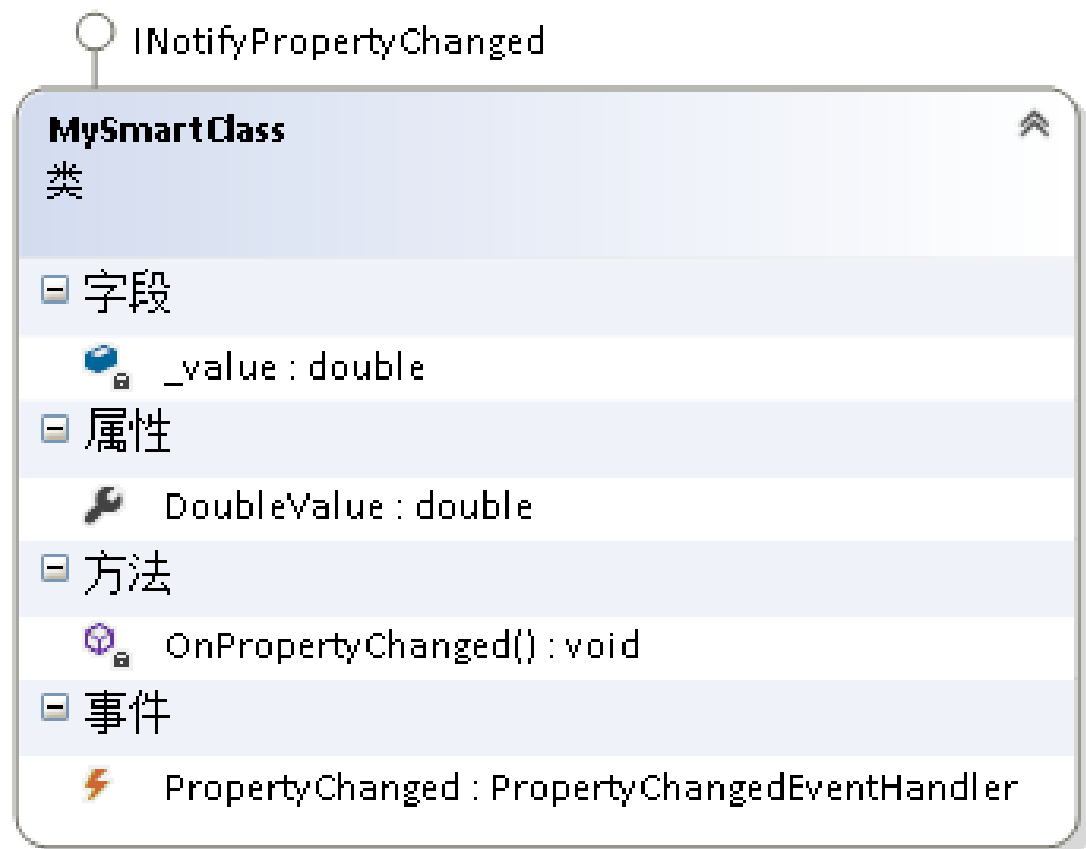


# 以特定的格式显示数据

```
doubleBind = new Binding("Text", DataObject, "DoubleValue");  
  
//指定按两位小数格式化Double数值  
doubleBind.FormattingEnabled = true;  
doubleBind.FormatString = "N2";  
  
txtDouble.DataBindings.Add(doubleBind);
```

[https://msdn.microsoft.com/en-us/library/26etazsy\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/26etazsy(v=vs.110).aspx)  
在这里可以找到各种用于数据格式化的字符串。

# 对象属性变更通知



如果数据对象实现了 **INotifyPropertyChanged** 接口，那么，将数据对象的属性值变动时，与它所绑定的 UI 控件会自动地更新

如果数据对象没有实现 **INotifyPropertyChanged** 接口，那么，将数据对象的属性值变动后，必须调用相应 Binding 对象的 **ReadValue()** 方法显式刷新 UI 控件的显示。

# 如果数据无效.....

如果用户输入了无效的数据，则可以在BindingComplete事件的e.Exception中知道出现了错误，并可以进一步地处理：

0.67m是一个无效的数字

DoubleValue值: 0.67m

数据源对象

☒ 实现INotifyPropertyChanged接口

设定绑定数据 更改绑定对象属性值

在Parse事件中: Value:0.67m, DesiredType:System.Double

在BindingComplete事件中: BindingCompleteContext:DataSourceUpdate, BindingCompleteState:Exception, Cancel:True, ErrorText:输入字符串的格式不正确。 , Exception:System.FormatException

清空

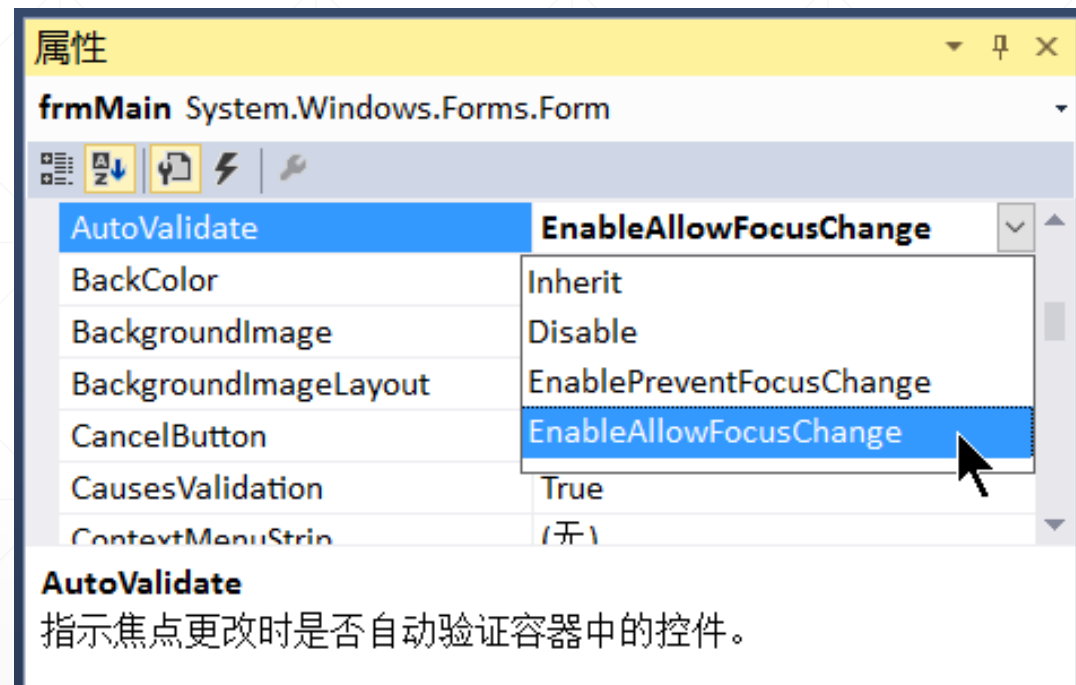
BindingComplete事件中检测和处理这种错误

# 数据无效时的焦点移动问题

Windows Forms中，拥有“**焦点 (focus)**”的控件接收键盘和鼠标输入。

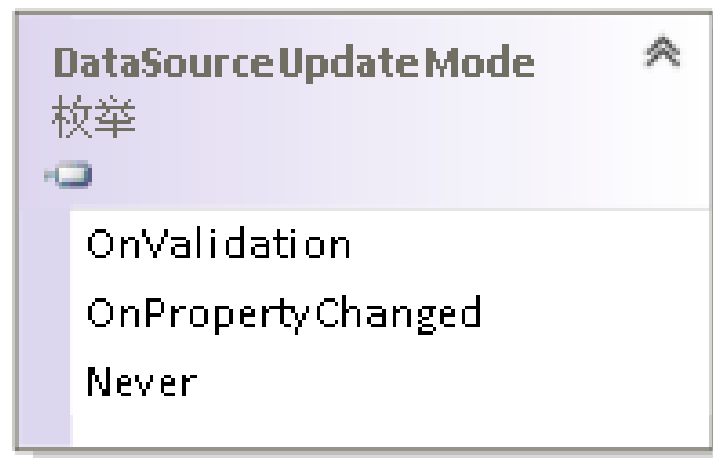
默认情况下，如果用户在数据绑定控件中输入了无效的数据，不更正这些数据，他将无法移动焦点，甚至不能关闭窗口体。

设置窗体的**AutoValidate**属性，可以调整这一行为特性。





# 数据和界面更新的时机选择



```
doubleBind = new Binding("Text", DataObject, "DoubleValue");  
.....  
doubleBind.ControlUpdateMode = ControlUpdateMode.OnPropertyChanged;  
doubleBind.DataSourceUpdateMode = DataSourceUpdateMode.OnPropertyChanged;  
.....  
txtDouble.DataBindings.Add(doubleBind);
```

# 可绑定数据源

---

# 数据源的职责

1

数据源向数据绑定控件提供要显示的数据，并且它也负责暂存被用户修改的数据。

2

数据源本身 **不负责** 将数据保存回数据库等底层数据存储介质。

# Windows Forms控件可绑定数据源类型

普通对象的属性

必须是属性

实现了IList接口的集合

诸如List<T>之类的集合，.NET基类库中的大部分集合满足这个要求

DataSet、DataTable、  
DataView

ADO.NET技术中的核心类型，本课程不介绍

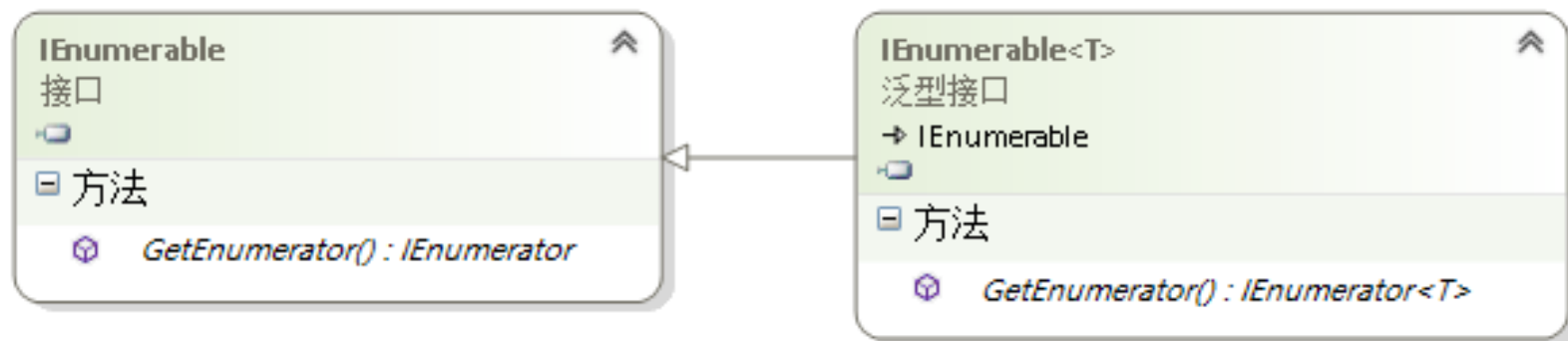
BindingSource

一个功能强大易用的软件组件，本课程将重点介绍

# 集合类型的绑定数据源

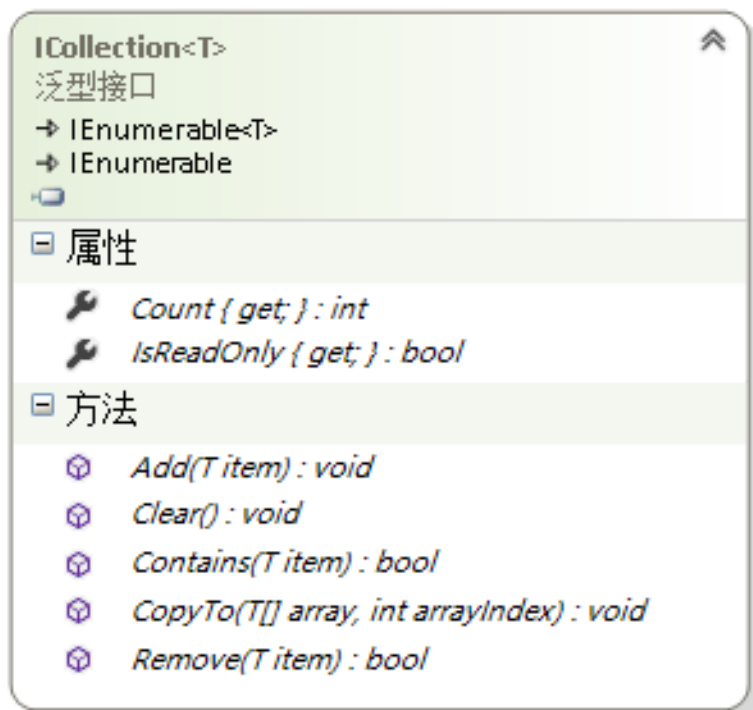
有多种类型的集合可以当作绑定数据源，它的能力主要通过相关的接口来表达。

# IEnumerable接口



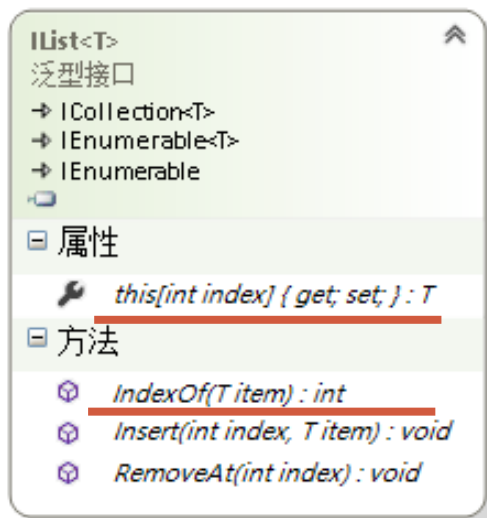
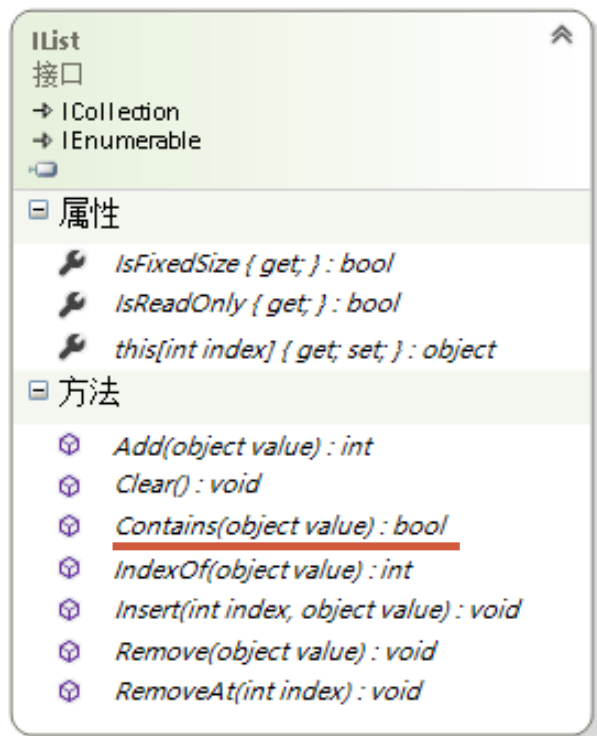
实现了**IEnumerable**接口的对象支持 **foreach** 遍历。

# ICollection接口



**ICollection**接口是定义一个集合的基础接口，提供了Add、Clear、Remove等方法，支持向集合中添加和删除数据。

# ICollection接口



实现了**IList**接口的对象可以看成是一个“可索引”、“可修改”、“有顺序”、“可查找”的数据集合。

**IList**是实现数据绑定的关键接口。诸如DataGridView之类数据绑定控件，只能绑定到实现了**IList**接口的对象上。

在.NET基类库中，有大量的集合类型实现了**IList**接口，比如所有的数组都实现了**IList**接口，因此，可以被用作绑定数据源。



# IBindingList接口

## IBindingList

### 接口

- IList
- ICollection
- IEnumerable

### 属性

- AllowEdit { get; } : bool
- AllowNew { get; } : bool
- AllowRemove { get; } : bool
- IsSorted { get; } : bool
- SortDirection { get; } : ListSortDirection
- SortProperty { get; } : PropertyDescriptor
- SupportsChangeNotification { get; } : bool
- SupportsSearching { get; } : bool
- SupportsSorting { get; } : bool

### 方法

- AddIndex(PropertyDescriptor property) : void
- AddNew() : object
- ApplySort(PropertyDescriptor property, ListSortDirection direction) : void
- Find(PropertyDescriptor property, object key) : int
- RemoveIndex(PropertyDescriptor property) : void
- RemoveSort() : void

### 事件

- ListChanged : ListChangedEventHandler

**IBindingList** 定义了一个 **ListChanged** 事件，此事件在向集合中添加和移除元素时触发，在数据绑定控件可以利用此事件及时刷新显示。

.NET 基类库中有一个 **BindingList<T>** 实现了此接口，它也是数据绑定应用程序中最常用的数据集合类型。

# IList和IBindingList示例

ID	info
1	A1
2	A2
3	A3
4	A4
*	

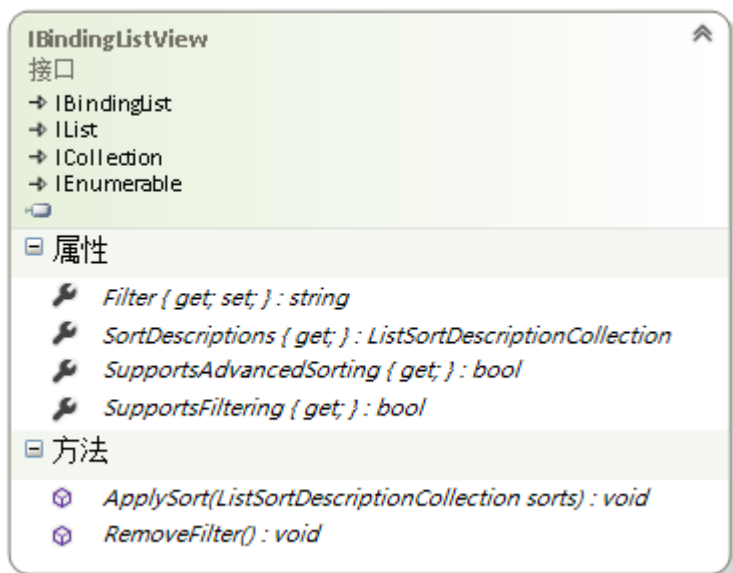
☒ 绑定到BindingList<MyClass>    ☐ 绑定到List<MyClass>

**新加对象**    删除对象

当DataGridView绑定到List<T>时，如果向此集合中添加了新对象，会发现表格不会刷新。

示例程序UseIBindingList

# IBindingListView接口



IBindingList有一个派生的接口**IBindingListView**，此接口在IBindingList的基础上，添加了支持多属性**排序**和**过滤**的相关属性和方法。它可以说是.NET基类库中数据处理功能最丰富的接口。

基类库中**DataView**和**BindingSource**都实现了此接口。

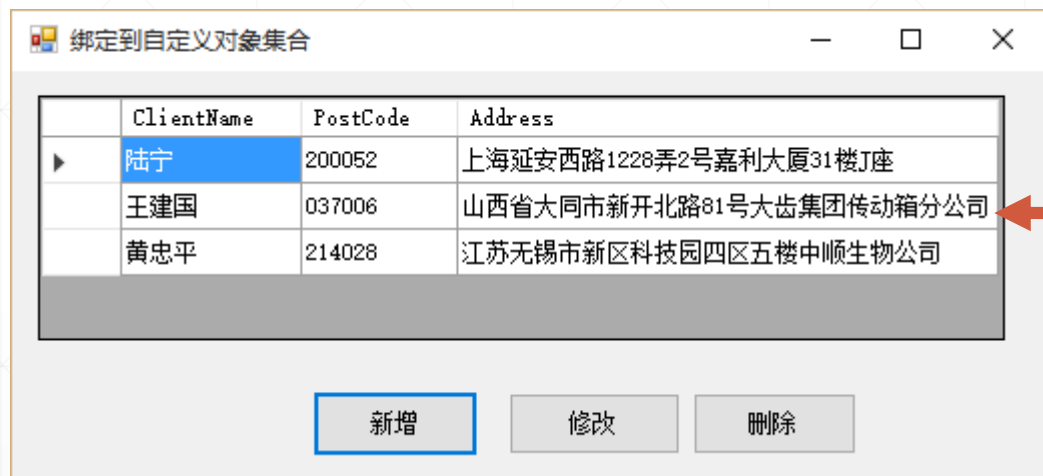
# 绑定到对象集合示例

---

本讲示例：BindToObjectCollection

# 使用DataGridView绑定数据集合

只要实现了 **ICollection<T>** 接口，这样的集合就可以被DataGridView所绑定。



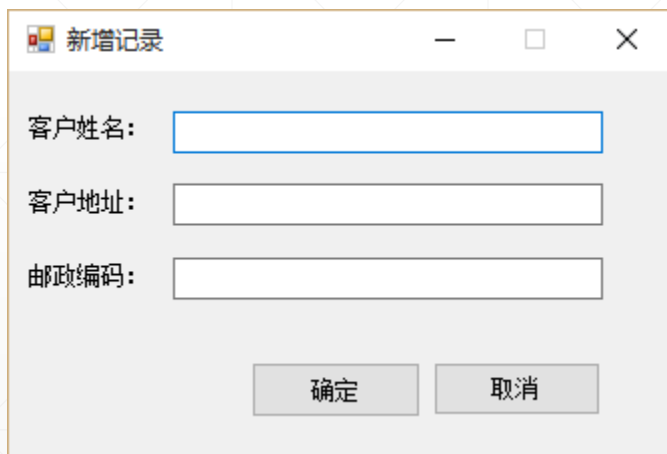
只读显示时，可以直接绑定List<T>。

```
List<OrderClient> clients = new List<OrderClient>();  
dataGridView1.DataSource = clients; // 绑定显示
```

将List<T>换成 **BindingList<T>**，则当对象集合中对象数目改变时，表格可以实时地更踪集合中对象数目的增减。

# 窗体重用

“新增”与“修改”记录窗体通常高度一致，开发中经常使用“**重用同一个窗体 + 特定标记**”的方式实现。



新增记录

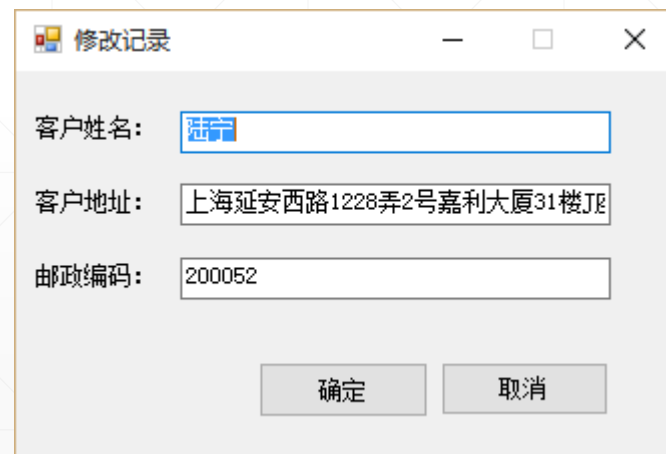
客户姓名:

客户地址:

邮政编码:

确定 取消

IsModify = false



修改记录

客户姓名:

客户地址:

邮政编码:

确定 取消

IsModify = true

主窗体传入的标记，用于区分是“修改”还是“新增”。

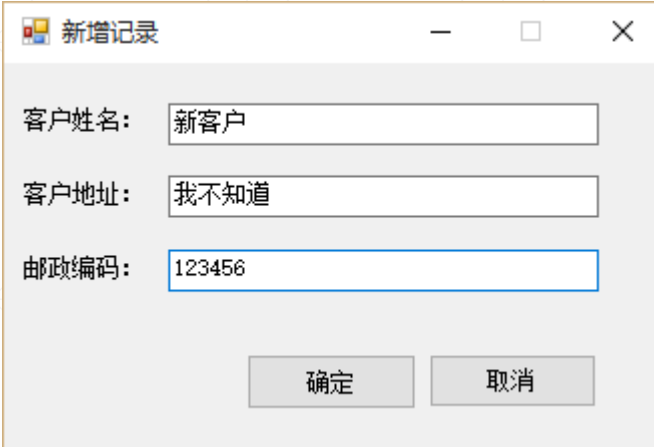
```
2 references
public frmAddOrModify(OrderClient client, bool IsModify)
{
    InitializeComponent();
    _client = client;

    //设定控件的数据绑定
    txtClientName.DataBindings.Add("Text", client, "ClientName");
    txtAddress.DataBindings.Add("Text", client, "Address");
    txtPostCode.DataBindings.Add("Text", client, "PostCode");

    if (!IsModify)
    {
        this.Text = "新增记录";
    }
    else
    {
        this.Text = "修改记录";
    }
}
```

依据标记值对界面进行必要的调整

在开发“新增记录”时，有两种方案：

A screenshot of a Windows-style dialog box titled "新增记录" (Add Record). It contains three text input fields: "客户姓名:" (Customer Name) with the value "新客户" (New Customer), "客户地址:" (Customer Address) with the value "我不知道" (I don't know), and "邮政编码:" (Postal Code) with the value "123456". At the bottom right, there are two buttons: "确定" (OK) and "取消" (Cancel).

新增记录

客户姓名: 新客户

客户地址: 我不知道

邮政编码: 123456

确定 取消

1

用户点击“确定”按钮关闭新增记录窗体时，临时new一个新的数据对象，设置其属性值，保存到本窗体的公有属性，然后主窗体就可以取出这个新加对象加入到数据源集合中。

2

主窗体事先new好一个数据对象，然后将其“注入”到新增记录窗体，用户点击“确定”，则将其加入到数据集合中。



# 使用数据绑定自动更新数据对象

```
public frmAddOrModify(OrderClient client, bool IsModify)
{
    InitializeComponent();

    //将注入对象的引用保存于私有字段中
    _client = client;

    //设定控件的数据绑定
    txtClientName.DataBindings.Add("Text", client, "ClientName");
    txtAddress.DataBindings.Add("Text", client, "Address");
    txtPostCode.DataBindings.Add("Text", client, "PostCode");
}
```

外部注入的数据对象

让文本框控件直接绑定到注入的数据对象上，这样一来，我们将无需手工编写代码依据控件当前值更新对象的对应属性值

```
private OrderClient _client;

2个引用
public OrderClient OrderClientObj
{
    get
    {
        return _client;
    }
}
```

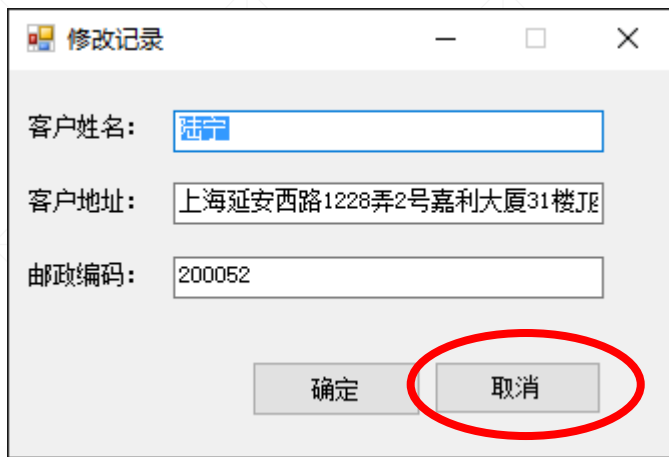
使用只读公有属性，向外界提供数据对象的引用。

# 示例程序中“新增记录”功能的实现

1个引用

```
private void btnAdd_Click(object sender, EventArgs e)
{
    //主窗体new一个新对象，注入到“新建记录”窗体中
    frmAddOrModify frm = new frmAddOrModify(new OrderClient(), false);
    if (frm.ShowDialog() == DialogResult.OK)
    {
        //从“新建记录”窗体的公有属性中取出属性已更新的对象，并将其加入到数据源集合中
        clients.Add(frm.OrderClientObj);
    }
}
```

# 数据绑定机制给“修改记录”带来的麻烦



修改记录

客户姓名: 赵子

客户地址: 上海延安西路1228弄2号嘉利大厦31楼J1E

邮政编码: 200052

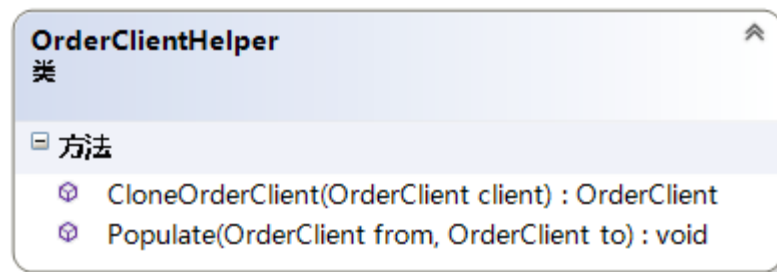
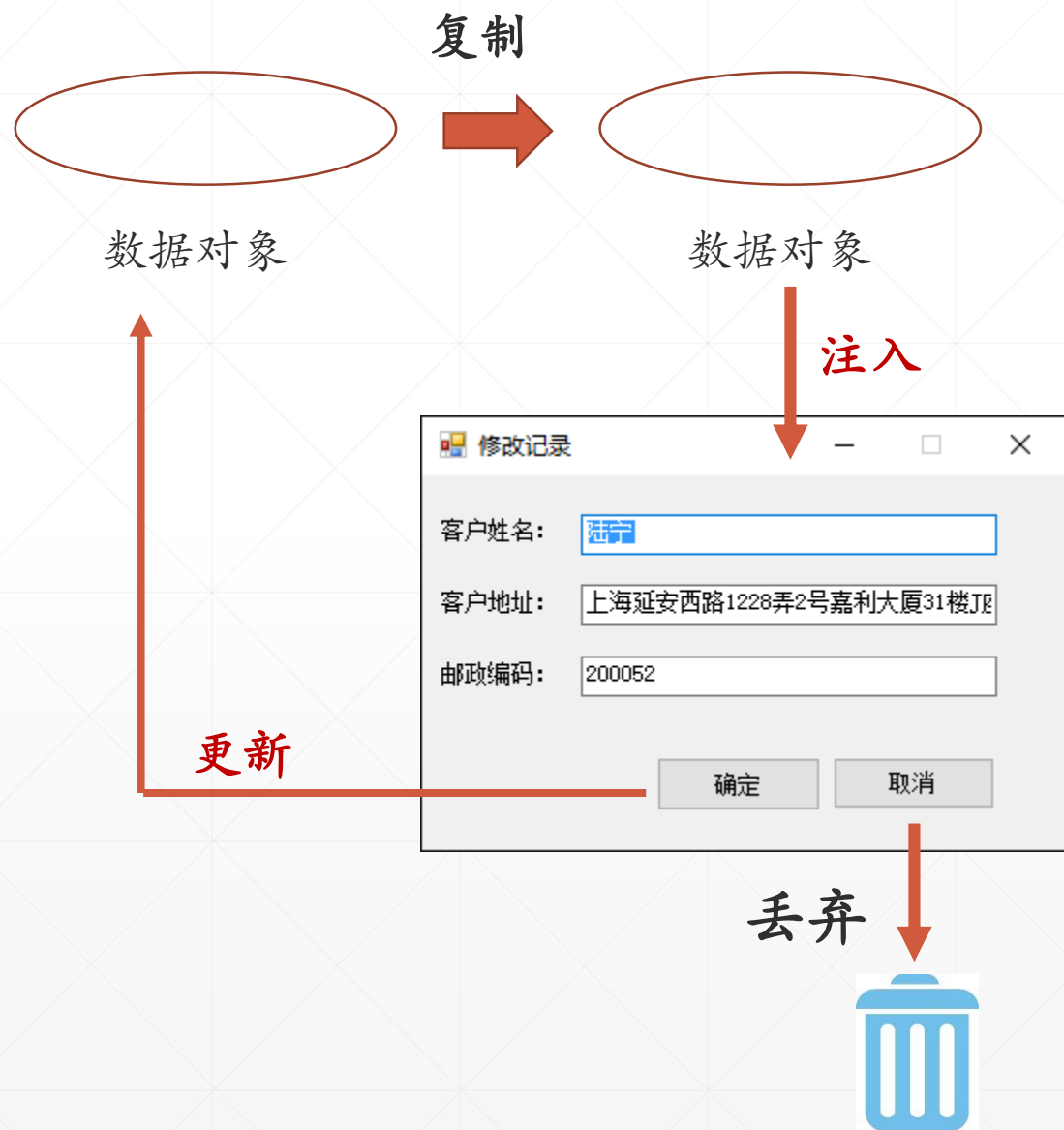
确定 取消

修改记录时，应该向窗体中注入一个“**现有**”的数据对象，然后使用数据绑定机制自动更新其属性。

麻烦之处在于，我们**允许用户取消修改**！因此，如果用户在取消之前在文本框中输入了一些东西，那么，即使他点击了“取消”按钮，由于数据绑定机制会自动更新底层数据对象，数据对象的相应属性已经更新！

有其他办法吗？

# 解决方案



CloneOrderClient() 方法:

克隆一个新OrderClient对象

Populate() 方法:

使用from对象的属性值更新to对象的相应属性值

# “修改记录”功能的最终实现

```
private void btnModify_Click(object sender, EventArgs e)
{
    int index = dataGridView1.CurrentRow.Index;
    if (index == -1)
        return;
    //创建一个待编辑数据对象的副本
    var modifyObject = OrderClientHelper.CloneOrderClient(clients[index]);
    //注入到“修改记录”窗体中
    frmAddOrModify frm = new frmAddOrModify(modifyObject, true);
    if (frm.ShowDialog() == DialogResult.OK)
    {
        //更新原始数据对象属性值
        OrderClientHelper.Populate(frm.OrderClientObj, clients[index]);
        if (frm.OrderClientObj is INotifyPropertyChanged == false)
        {
            //如果OrderClient没有实现INotifyPropertyChanged接口，
            //而DataGridView又是直接绑定到BindingList集合的
            //当对象被修改时，只能通过重新绑定实现刷新。
            dataGridView1.DataSource = null;
            dataGridView1.DataSource = clients;
        }
    }
}
```

当修改记录时，将需要修改的数据对象复制一份，再注入到窗体中。

更新原始数据对象

注意一下界面刷新的方法

# 小结

如果让数据绑定控件直接绑定到一个对象集合，那么，此对象集合最好满足以下条件以简化开发：

1

数据对象中所有的需要参与绑定的字段都必须封装为“属性”。

2

数据对象的类型最好实现**INotifyPropertyChanged**接口。

3

选择**BindingList<T>**保存自定义类型对象，并把它作为绑定数据源。

# 常用的数据绑定控件

---

# 支持数据绑定的Windows Forms控件

只要控件实现了 **IBindableComponent** 接口，它就支持数据绑定。

```
public interface IBindableComponent : IComponent, IDisposable
{
    BindingContext BindingContext { get; set; }
    ControlBindingsCollection DataBindings { get; }
}
```

由于Windows Form控件的基类Control实现了此接口，因此，我们可以这么说：**基类库中所提供的所有标准Windows Form控件都是数据绑定控件。**



# 数据绑定控件的分类

## 简单数据绑定控件

数据绑定控件中的相应属性和数据对象的相应属性是一一对应的，如TextBox、Label

## 复杂数据绑定控件

可以绑定到一个对象集合，同时显示数据源中的多个数据对象，如DataGridView

# 简单数据绑定控件实现数据绑定

1

```
//创建绑定对象
Binding ClientNameBind = new Binding(
    "Text",           //要绑定的数据绑定控件的名字
    client,           //绑定数据源，一个OrderClient对象
    "ClientName"     //导航路径，指定要绑定的数据对象属性名
);
```

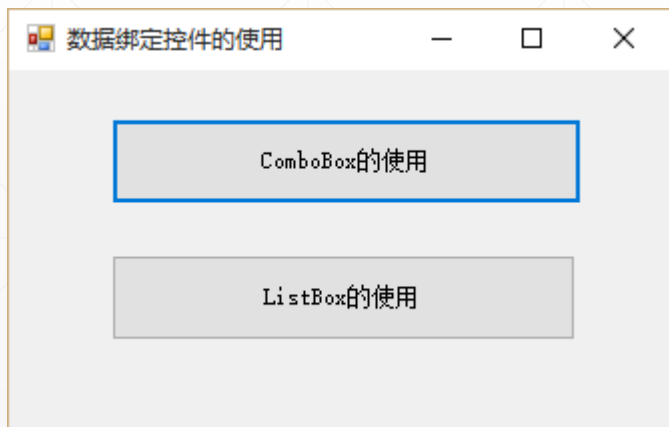
2

```
//追加到TextBox控件的DataBinding集合中
textBox1.DataBindings.Add(ClientNameBind);
```

# 复杂控件的数据绑定

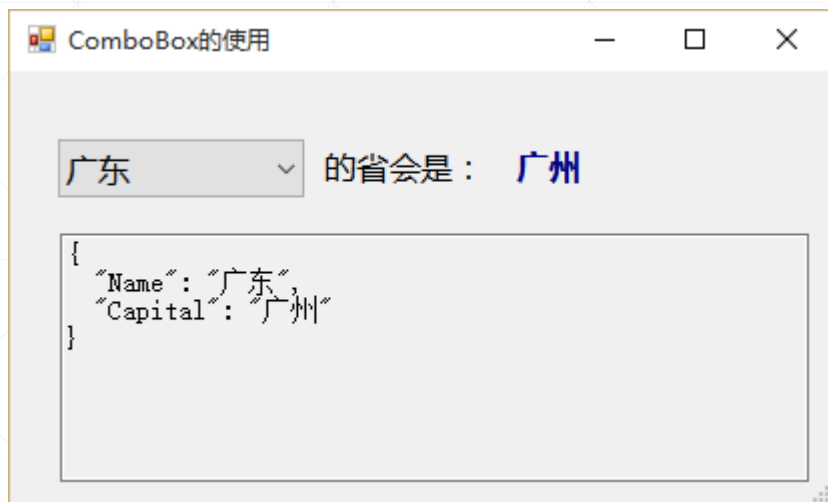
ComboBox、ListBox和DataGridView是最常见的三种复杂数据绑定控件，其中，ComboBox与ListBox的用法高度类似，而DataGridView则要复杂得多。

目前只介绍ComboBox与ListBox的用法，DataGridView放到后面再介绍。



示例：UseDataBoundControl

# ComboBox的使用



```
private void ComboBoxBinding()
{
    comboBox1.DisplayMember = "Name";
    comboBox1.ValueMember = "Capital";
    comboBox1.DataSource = ProvinceRepository.GetProvinces();
}
```

① **DataSource**: 指定数据源

② **DisplayMember**: 指定显示数据源单个对象中的哪个属性

③ **ValueMember**: 指定当前选中项所对应的SelectedValue值取自数据源对象的哪个属性

④ **SelectedItem**: 引用当前选中项所对应的数据源对象

# ListBox

The screenshot shows a Windows Form application titled "ListBox使用示例". It features a button "提取数据" at the top left. Below it is a ListBox containing a list of names: 刘克平, 刘丽娟, 阮淑婷, 黄璐璐, 黄全 (highlighted in blue), 刘定平, 周长江, 黄耀文, 程训华, 王江霞, 代连瑜, 王洪蓝, 陆瑛珍, and 黄宇珍. To the right of the ListBox are three input fields: "邮编:" with value "519070", "地址:" with value "珠海前山兰埔路178号香洲区人民医院外科", and "ClientID:" with value "5". At the bottom is a code editor showing the following JSON:

```
{  
  "ClientID": 5,  
  "ClientName": "黄全",  
  "Address": "珠海前山兰埔路178号香洲区人民医院外科",  
  "PostCode": "519070",  
  "Telephone": "",  
  "Email": ""  
}
```

ListBox与ComboBox的使用方法基本一致。

```
private void SetupBind()
{
    lstClients.DisplayMember = "ClientName";
    lstClients.ValueMember = "ClientID";
    lstClients.DataSource = clients;

    txtAddress.DataBindings.Add("Text", clients, "Address");
    txtPostCode.DataBindings.Add("Text", clients, "PostCode");
}
```

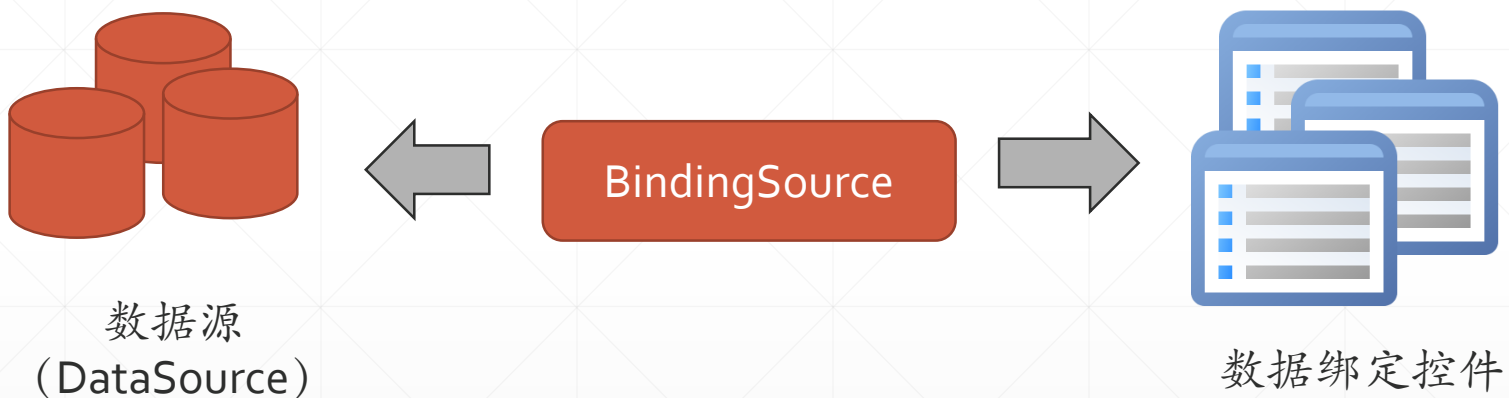
当ListBox与TextBox等混用，并且绑定到同一个数据集合时，在ListBox中选择不同的项，各个数据绑定控件所呈现的数据会自动同步。

# BindingSource组件开发实例

---

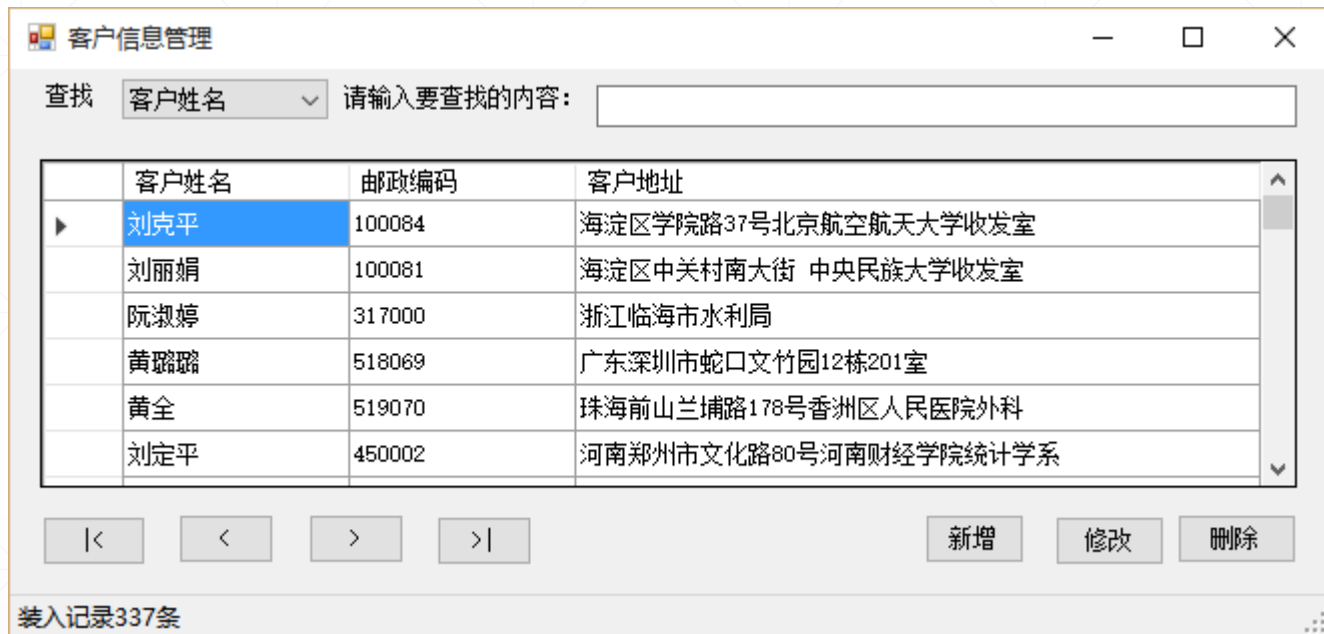
# BindingSource组件的地位

BindingSource组件可以看成是绑定控件和相关联的数据源之间的代理。



在引入BindingSource组件之后，开发Windows Form数据绑定应用程序时，不再推荐将数据绑定控件直接绑定到对象集合，而应将对象集合赋值给BindingSource组件的DataSource属性，再让数据绑定控件绑定到BindingSource。

# BindingSource示例



客户姓名	邮政编码	客户地址
刘克平	100084	海淀区学院路37号北京航空航天大学收发室
刘丽娟	100081	海淀区中关村南大街 中央民族大学收发室
阮淑婷	317000	浙江临海市水利局
黄璐璐	518069	广东深圳市蛇口文竹园12栋201室
黄全	519070	珠海前山兰埔路178号香洲区人民医院外科
刘定平	450002	河南郑州市文化路80号河南财经学院统计学系

示例：UseBindingSource

此示例展示了典型桌面数据库应用程序的编程技巧

1

在记录集中移动

2

新增/修改/删除记录

3

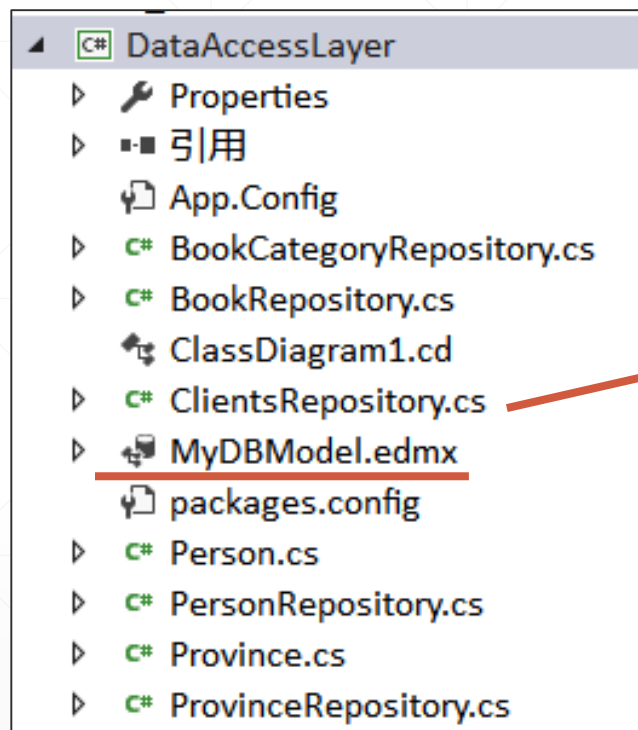
查找与过滤

4

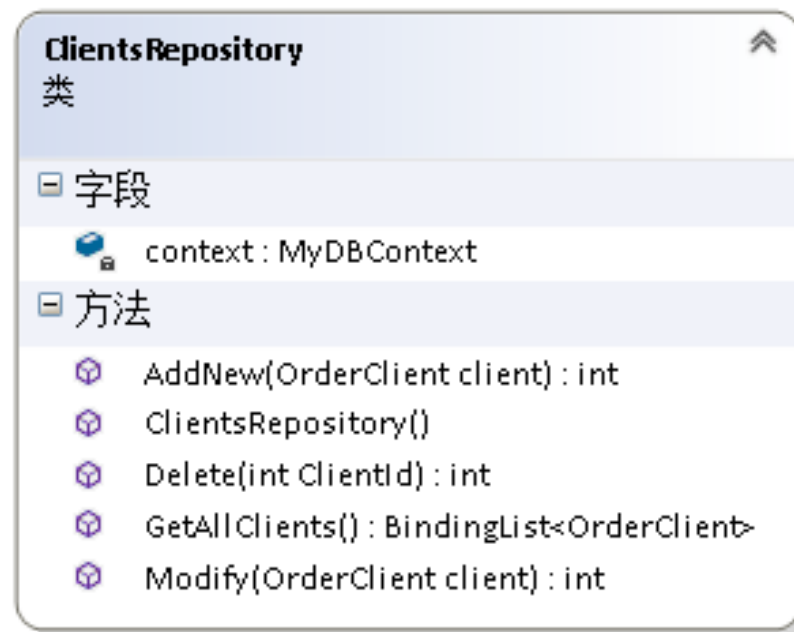
DataGridView的使用技巧



# 示例分析



使用DataAccessLayer程序集封装Entity Framework，实现数据存取功能



使用ClientsRepository封装对OrderClient表的CRUD功能

# 基于BindingSource的数据绑定代码模板

```
//从数据库中提取数据，放到BindingList<T>集合中
clients = repo.GetAllClients();

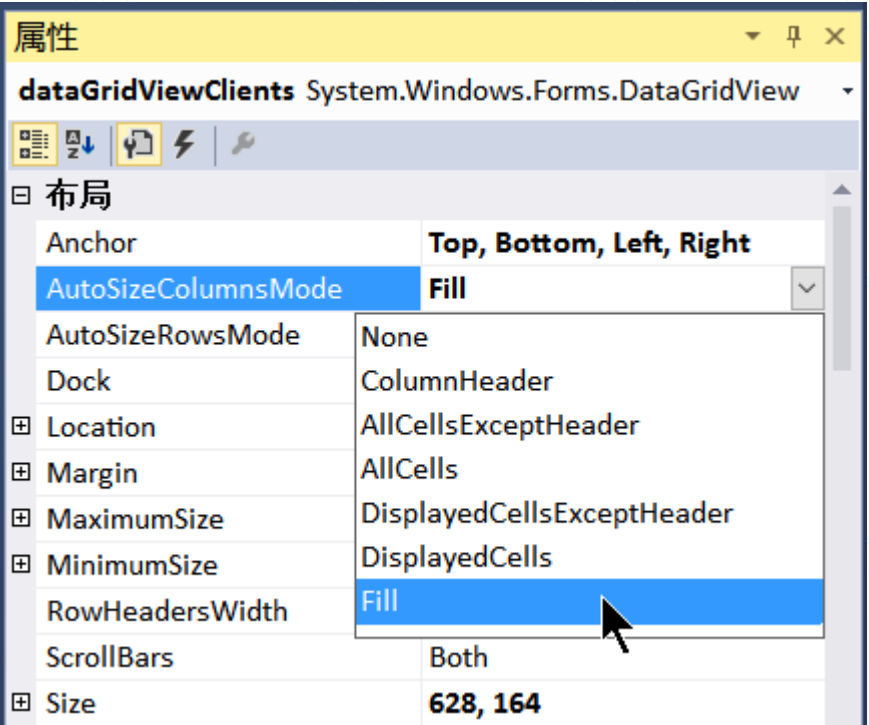
//将其关联上BindingSource组件
bindingSourceClients.DataSource = clients;

//设定DataGridView的数据源引用BindingSource
dataGridViewClients.DataSource = bindingSourceClients;
```

# 将DataGridView设定为“浏览状态”

属性名	说明
ReadOnly	为true时，不允许用户修改单元格
AllowUserToAddRows	为false时，不允许用户新加行
AllowUserToDeleteRows	为false时，不允许用户删除行

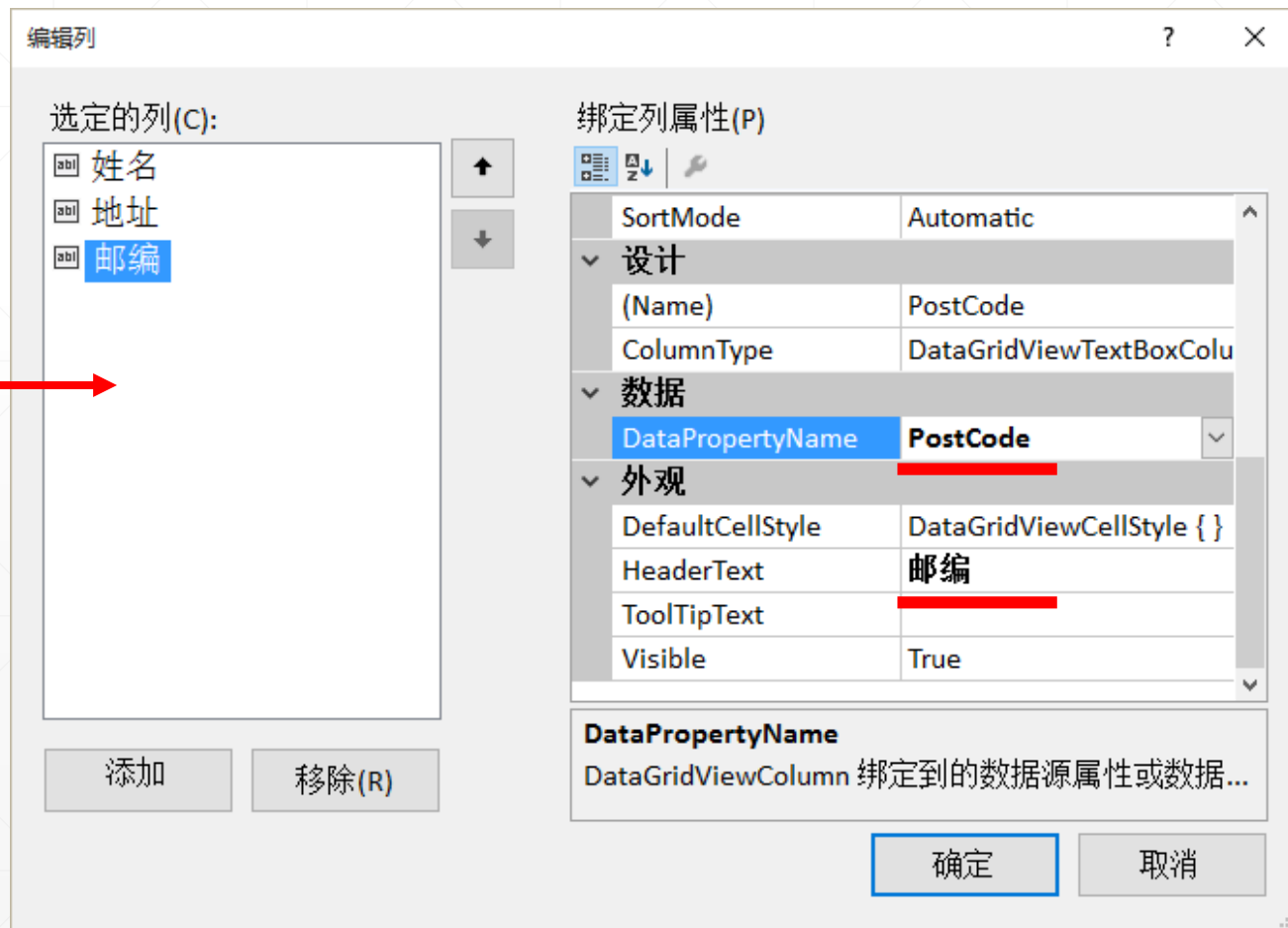
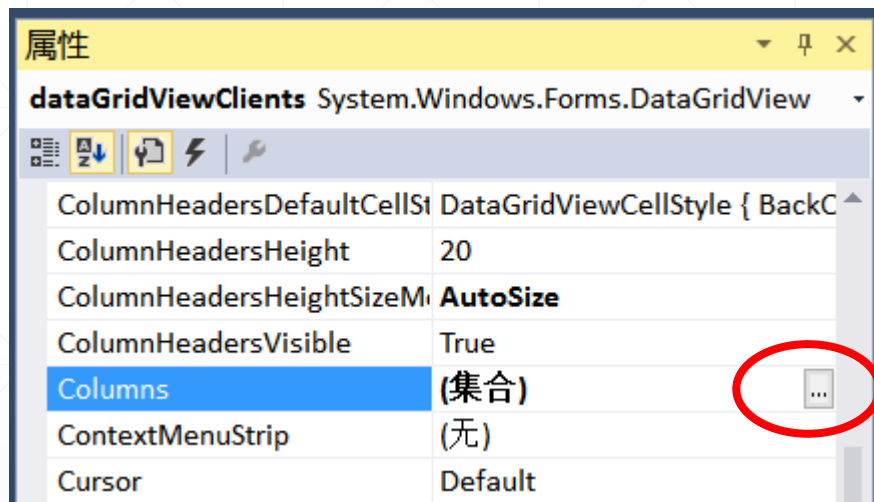
# 设置列宽自动调整



值	说明
AllCells	依据本列所有单元格要显示的最长数据而定
AllCellsExceptHeader	依据本列所有单元格（除了列标题）要显示的最长数据而定
DisplayedCells	依据当前用户可以看到的本列所有单元格要显示的最长数据而定
DisplayedCellsExceptHeader	同上，只是不考虑列标题的长度
ColumnHeader	依据本列标题的长度而定
Fill	让所有单元格自动调整以充满整个显示区域，每个列的最终宽度由其FillWeight属性决定。
None	关闭列宽度自动调用功能。

# 设置中文列名

使用代码将DataGridView的**AutoGenerateColumns**属性设置为**false**，取消其自动生成列的功能，然后设置Columns属性.....



# 数据导航的实现

调用BindingSource的MoveNext()  
等方法实现

客户信息管理

查找 客户姓名 请输入要查找的内容:

客户姓名	邮政编码	客户地址
		关村南大街 中央民族大学收发室
		市水利局
		市蛇口文竹园12栋201室
		兰埔路178号香洲区人民医院外科
刘定平	450002	河南郑州市文化路80号河南财经学院统计学系
周长江	121000	辽宁省锦州凌河区解放路五段12号市司法局

|< < > >|

新增 修改 删除

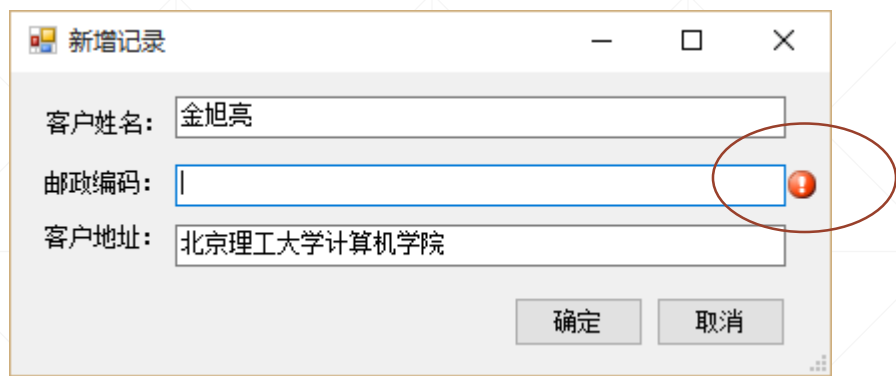
第7条/共337条

响应BindingSource的PositionChanged事件

# BindingSource的各种事件

事件名	说明
AddingNew	新加对象时触发
CurrentChanged	“当前”绑定的对象有变动（比如引用另一个数据对象或DataSource）时触发
CurrentItemChanged	在Current属性值更改后触发，如果数据对象实现了INotifyPropertyChanged接口，则数据对象的属性更改后，也会触发
ListChanged	数据源集合中数据个数有变化（比如新增或删除），或者引用了新的DataSource时触发
PositionChanged	在Position属性更改后触发
BindingComplete	在所有数据绑定控件均获取了值之后触发

# 使用ErrorProvider组件显示出错信息

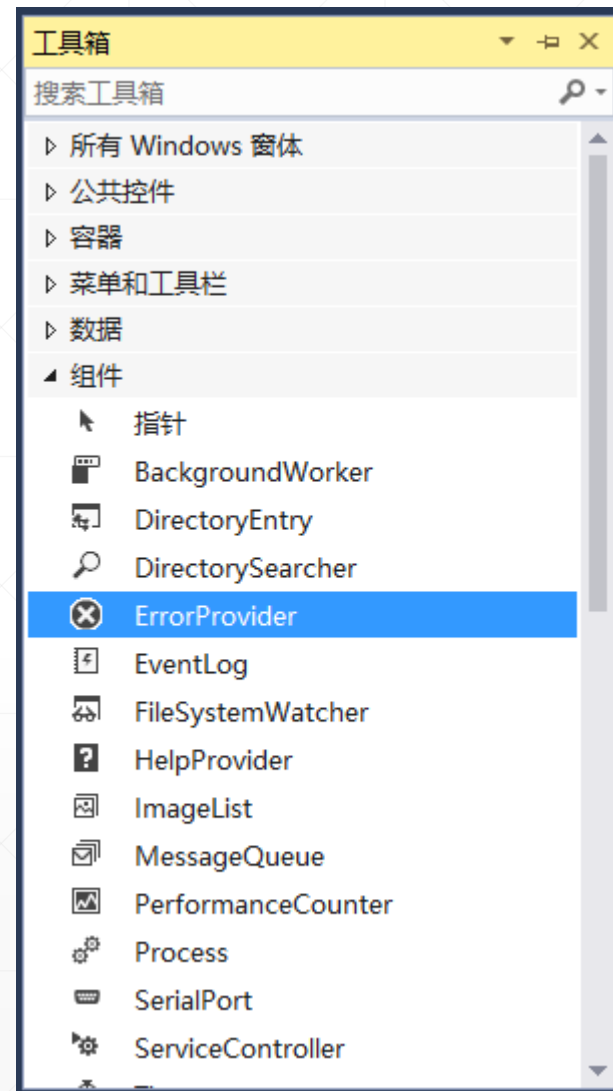


//显示出错的信息

```
ErrorProvider组件名.SetError(相关联的控件, "出错信息");
```

//取消出错信息的显示

```
ErrorProvider组件名.SetError(相关联的控件,null);
```





# 数据验证

属性

txtClientName System.Windows.Forms.TextBox

使用控件的Validating事件完成验证数据的功能

Validating

txtClientName\_Validating

Validating  
在控件验证时发生。

```
private void txtPostCode_Validating(object sender, CancelEventArgs e)
{
    //如果用户没有按“取消”按钮，则进行数据验证
    if (CanClose == false)
    {
        //使用正则表达式验证邮编
        Regex rgx = new Regex(@"^\d{6}$");
        if (rgx.Match(txtPostCode.Text).Success == false)
        {
            //显示出错信息
            errProviderForPostCode.SetError(txtPostCode, "邮编为6位数字");
            e.Cancel = true; //未通过验证
        }
        else
        {
            //通过数据验证，取消出错信息的显示
            errProviderForPostCode.SetError(txtPostCode, null);
        }
    }
}
```

# 通知所有控件验证数据

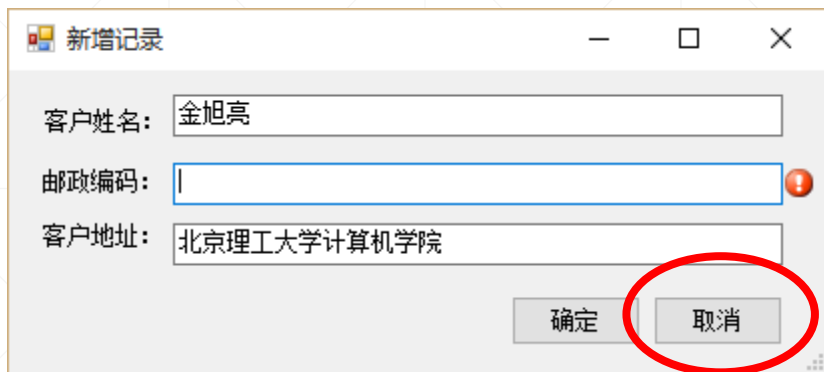
当用户点击“确定”按钮时，必须有一个方法能自动地触发所有控件的 Validating 事件，为此，可以调用窗体的 **ValidateChildren()** 方法，如果有任一控件的验证失败，此方法返回 false。

```
private void btnOK_Click(object sender, EventArgs e)
{
    if (this.ValidateChildren() == false)
    {
        MessageBox.Show("你输入了无效的数据,请更正.....");
        return;
    }
    CanClose = true;
    //保存到数据库中
    ClientsRepository repo = new ClientsRepository();
    OrderClient client = _bindingSource.Current as OrderClient;
    try
    {
        if (_isModify)
        {
            repo.Modify(client);
        }
        else
        {
            repo.AddNew(client);
        }
        _bindingSource.EndEdit();

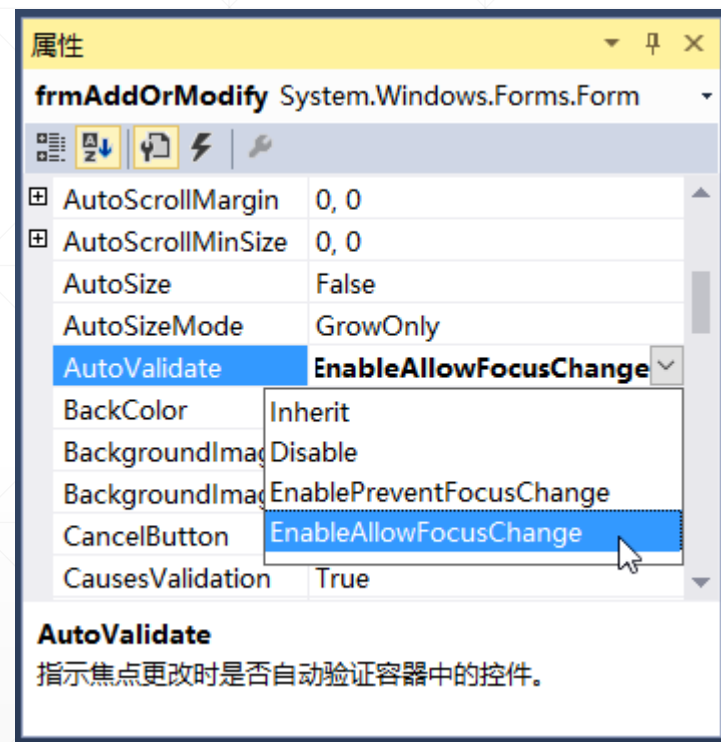
        Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# “数据验证”带来的麻烦

当用户点击“取消”按钮时，不管通不通过数据验证，都应该允许用户关闭窗口体：



但在默认设置下，如果绑定控件中有无效数据，将造成“取消”按钮无法被点击！用户只有输入有效的数据之后，他才能点击“取消”按钮关闭窗口体。



设置窗体的 **AutoValidate** 属性取 “**EnableAllowFocusChange**” 值，允许验证失败的控件放弃焦点，这样才可以让用户点击“取消”按钮关闭窗口体。

# 数据的过滤

客户信息管理

查找 客户姓名 请输入要查找的内容: 王

	客户姓名	邮政编码	客户地址
	王蓝蓝	200240	上海市闵行区鹤庆路560弄77号502室
▶	王建国	037006	山西省大同市新开北路81号大齿集团传动箱分公司
	王家俊	024000	内蒙古赤峰市赤峰人民广播电台（赤峰市红山区钢铁西杰）
	王勤	350000	福建福州市尚宾路尚宾花园1号607
	王江霞	100081	北京市海淀区8125信箱船舶经济中心
	王蓝蓝	200240	上海市闵行区鹤庆路560弄77号502室

|< < > >| 新增 修改 删除

第3条/共39条

由于DataGridView绑定的数据已经在内存中，以BindingList<T>的形式存在，因此，可以直接使用LINQ to Object对这个集合进行过滤，然后重新绑定数据源即可。

```
private void FilterData()
{
    if (clients == null)
    {
        return;
    }
    //获取用户输入
    string userInput = txtUserInput.Text.Trim();
    BindingList<OrderClient> result = null;
    switch (cboFindWhat.SelectedIndex)
    {
        case 0:
            //按ClientName过滤数据
            result = new BindingList<OrderClient>(
                clients.Where(client => client.ClientName.Contains(userinput)).ToList());

            break;
        case 1:
            //按Address过滤数据
            result = new BindingList<OrderClient>(
                clients.Where(client => client.Address.Contains(userinput)).ToList());

            break;
        case 2:
            //按PostCode过滤数据
            result = new BindingList<OrderClient>(
                clients.Where(client => client.PostCode.Contains(userinput)).ToList());

            break;
        default:
            result = clients;
            break;
    }
    bindingSourceClients.DataSource = result;
}
```

使用“**Where标准扩展方法+Lambda表达式**”实现过滤，你也可以直接使用LINQ查询完成这个工作。

重新绑定以刷新显示

# 分页浏览数据功能的实现

数据的分页浏览

	ClientID	ClientName	AddressStr	PostCode
▶	692	陆宁	上海延安西路1228弄2号嘉利大厦31楼J座	200052
	693	王建国	山西省大同市新开北路81号大齿集团传动箱分公司	037006
	694	张栩	北京市西城区黄寺大街23号阳光丽景2-4-602	100011
	695	叶玮池	福建厦门市枋湖西二路5-7号厦门锦江电子有限公司	361000
	696	郭楠	辽宁鞍山市铁东区东解放路53-44号	114005
	697	黄忠平	江苏无锡市新区科技园四区五楼中顺生物公司	214028
	698	陈莉	广州市华景新城华景路粤生街74号405室	510630
	699	王家俊	内蒙古赤峰市赤峰人民广播电台（赤峰市红山区...	024000
	700	王勤	福建福州市尚宾路尚宾花园1号607	350000
	701	刘克平	海淀区学院路37号北京航空航天大学收发室	100083

< >

第一页 前一页 下一页 最后一页 跳到: 31 页

第3页/共31页

示例：BrowseByPage

# 要点：

使用“标准集合扩展方法”中的 **Skip()**、**Take()** 方法实现

```
/// <summary>
/// 显示指定页的记录
/// </summary>
/// <param name="PageIndex"></param>
6 个引用
private void ShowPage(int PageIndex)
{
    if (PageIndex <= 0 || PageIndex > PageCount)
        return;

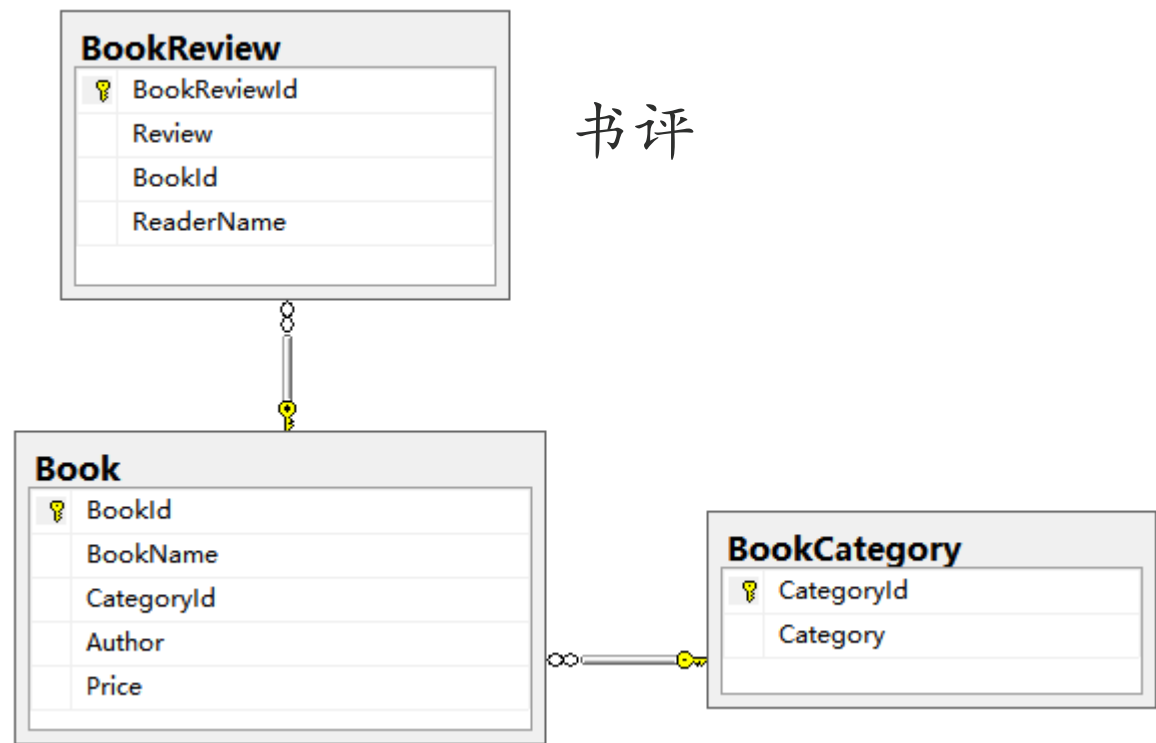
    var result = new BindingList<OrderClient>(clients.Skip((PageIndex - 1) * PageSize)
        .Take(PageSize).ToList());

    bindingSource1.DataSource = result;

    lblInfo.Text = string.Format("第{0}页/共{1}页", CurrentPage, PageCount);
}
```

# 一对多关联的数据

某种类别包容多本书，每本书可以有多个书评。





# 一对多数据的浏览

列出书所属的类别，选择一个种类，  
可以看到这个类别下的所有书

绑定到同一个BindingSource，  
以实现同步

本种类下所包容  
的书清单

读者	书评
读者350	非常漂亮的一个版本。很棒
读者376	慕名已久，已经看过一部分了，感觉刚开始读时会比较晦涩，...
读者909	书体外观精美，内容丰富，情节极富吸引力，一晚看完大半本...
读者221	书很精致，这个价钱可以
读者342	得了国际大奖的书，很值得看

响应BindingSource组件  
的CurrentChange事件，  
动态绑定书评集合。

# 学习指南

数据绑定是一个比较庞杂的技术领域，首先要理解它是干什么的，之后就是弄清楚它的脾气，接着主动地在开发中应用它们，才能真正地用好它们。

本部分提供了大量的示例，为进一步学习提供了扎实的基础，请务必阅读这些代码，并且整理好，开发备用。

# 挑战极限

BookAndBookReview示例展示了一对多数据的显示。  
你能实现一对多关联数据的新增、删除、修改和查询吗？

自己编写一个示例掌握这些功能的开发技巧。