10 BarManager 组件

所在程序集：Phenix.Windows

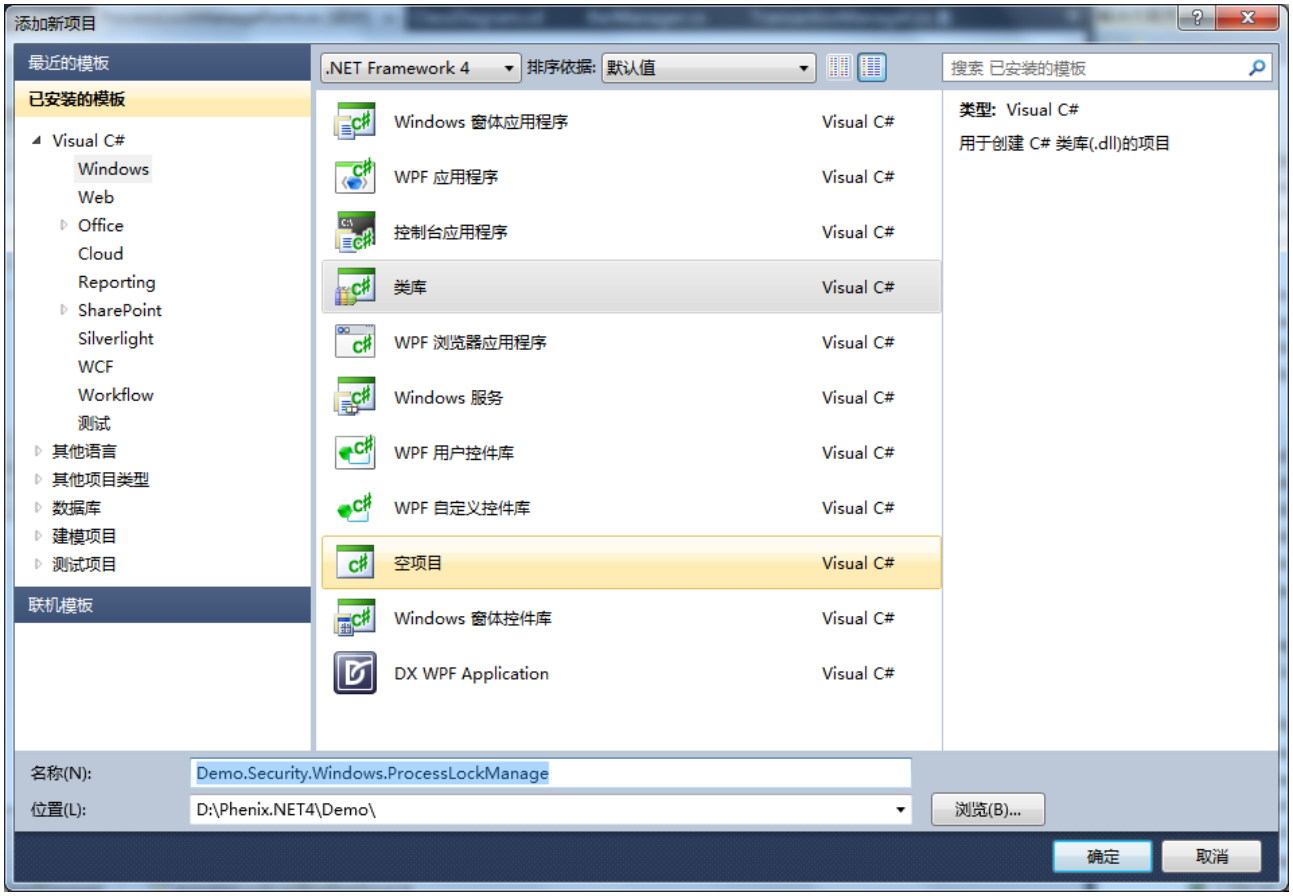
命名空间：Phenix.Windows

组件功能：继承自 DevExpress.XtraBars.BarManager，缺省构建了数据集的增删改查、打印、导入、导出等功能按钮；集成 EditValidation、ReadWriteAuthorization 组件，通过当前 BindingSource 以及当前用户的权限，控制 Tools Bar、Status Bar 的灰亮、显示内容及绑定数据控件的读写规则、验证规则等；Tools Bar 上的按钮除了 Help 外，都有缺省的处理过程。

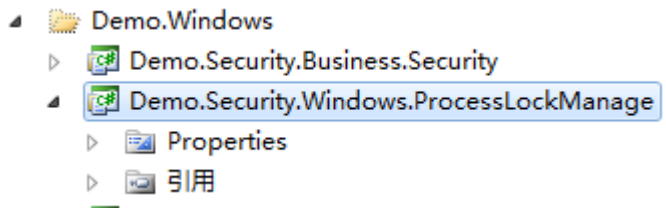
10.1 快速界面开发

有了 BarManager 组件，可以减轻不少界面的开发工作量，开发人员无需关心业务对象增删改的界面逻辑（功能按钮灰亮、数据控件的读写等控制），特别是单表和简单的主从表结构操作界面，基本上不用编写代码。

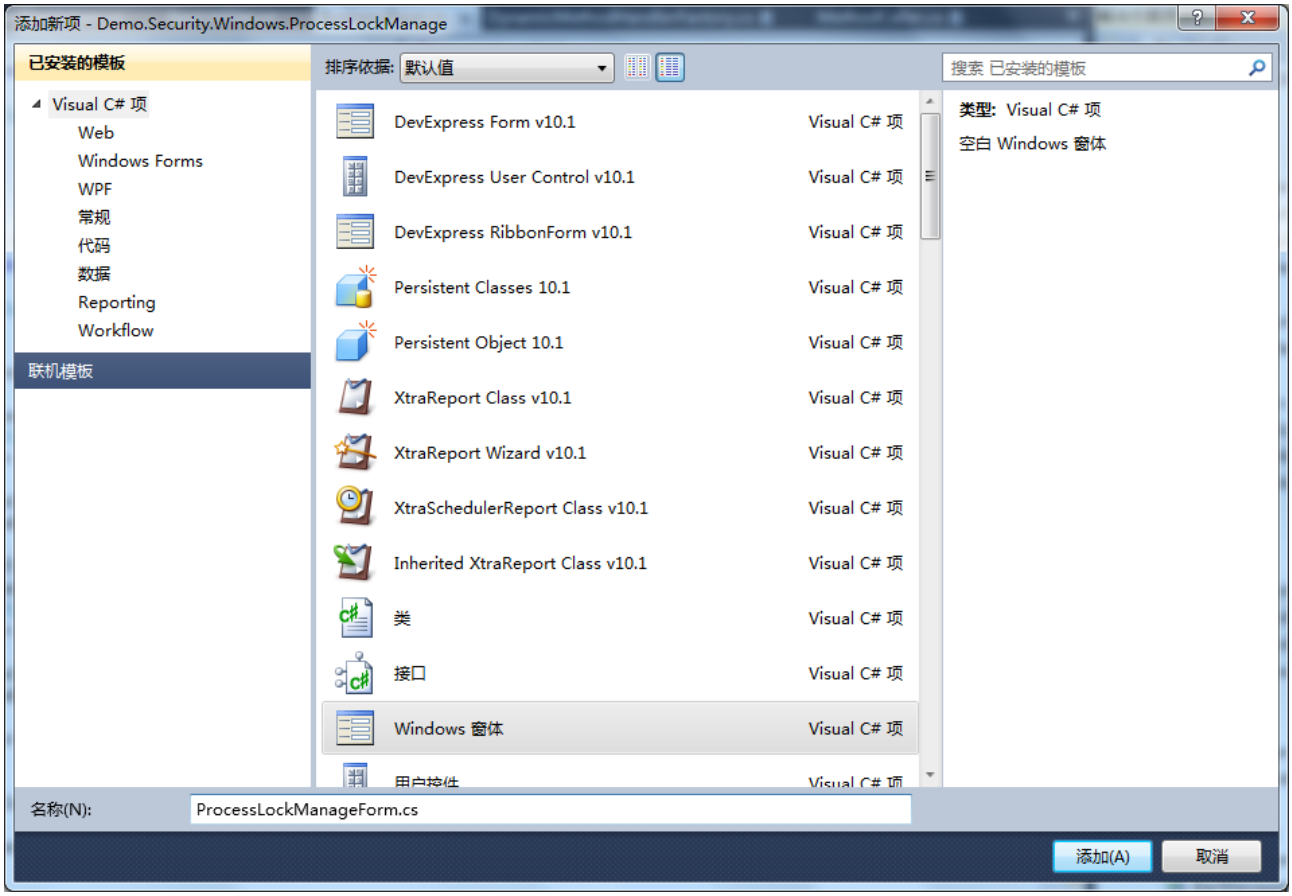
下面我们演示一个窗体插件的设计过程：



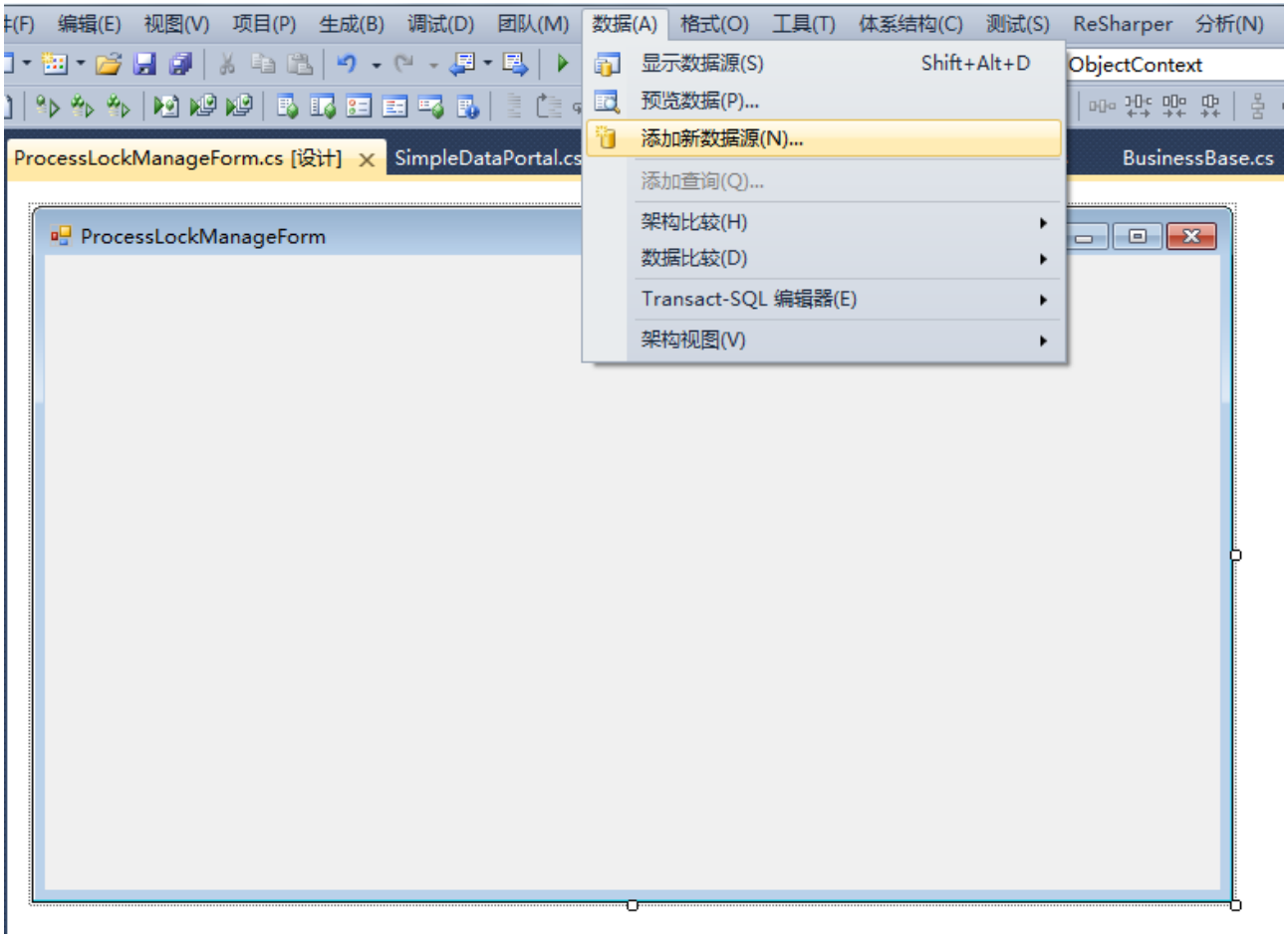
新建类库工程：

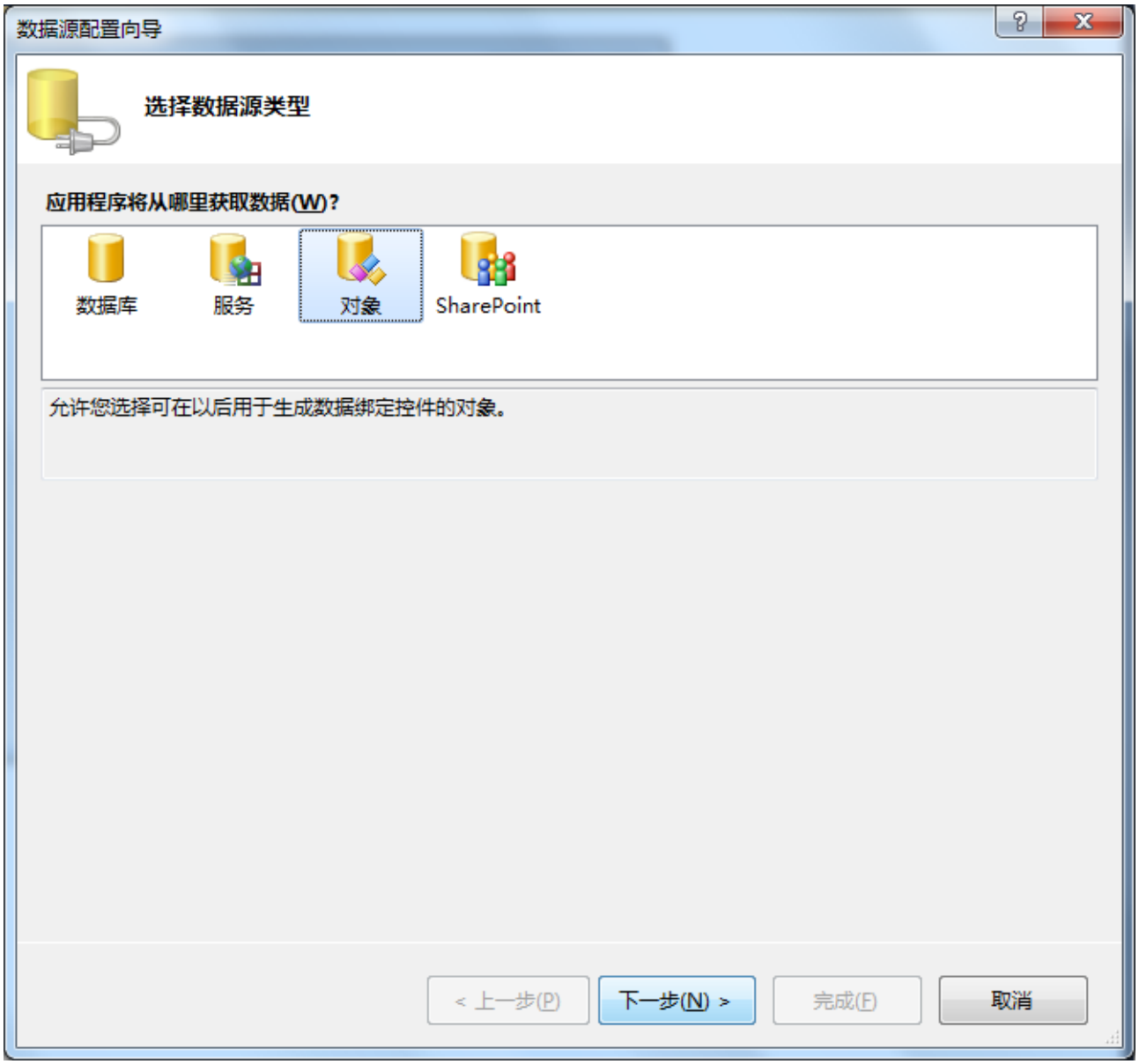

Demo.Windows
Demo.Security.Business.Security
Demo.Security.Windows.ProcessLockManage
Properties
引用

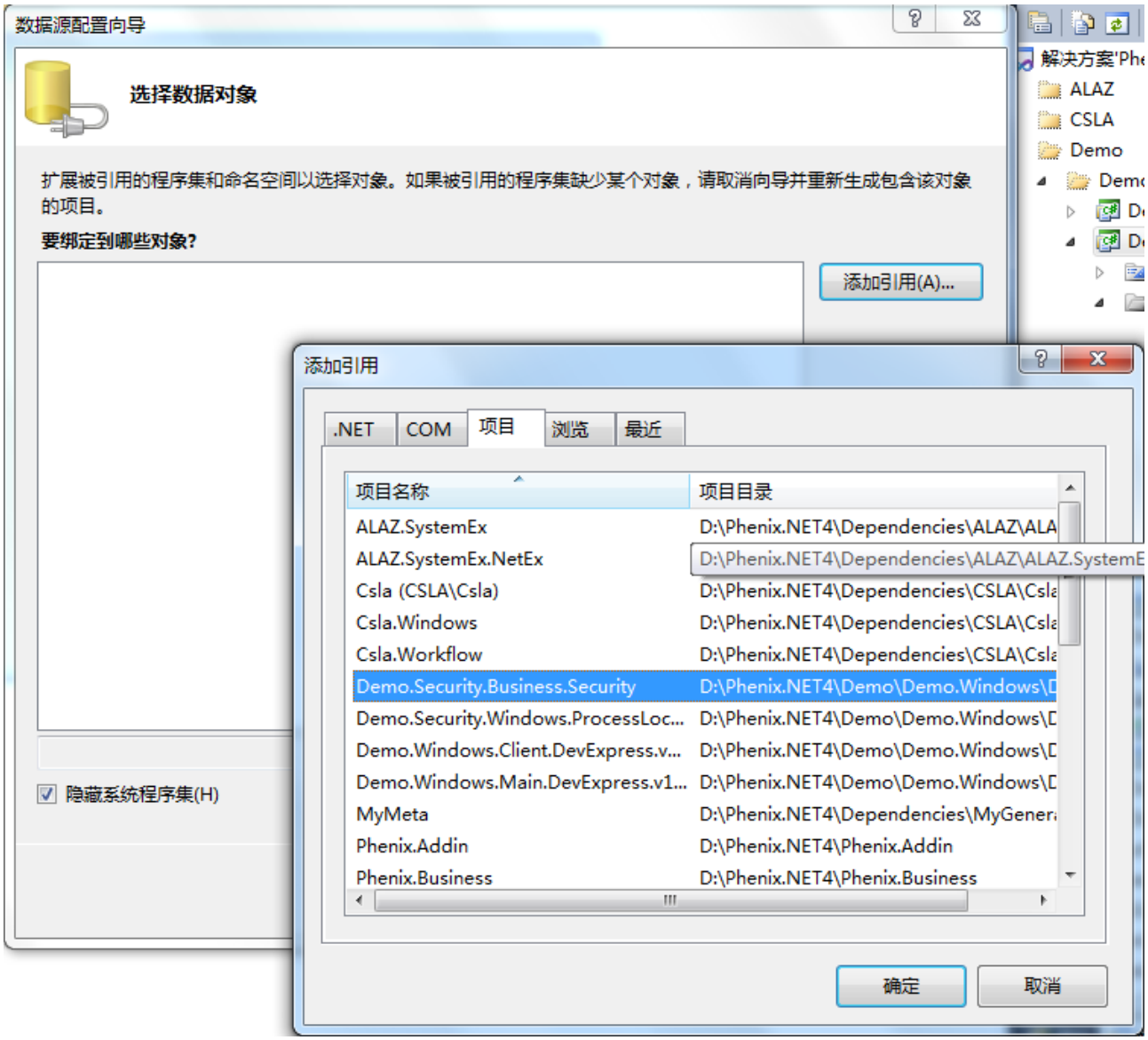
添加窗体：

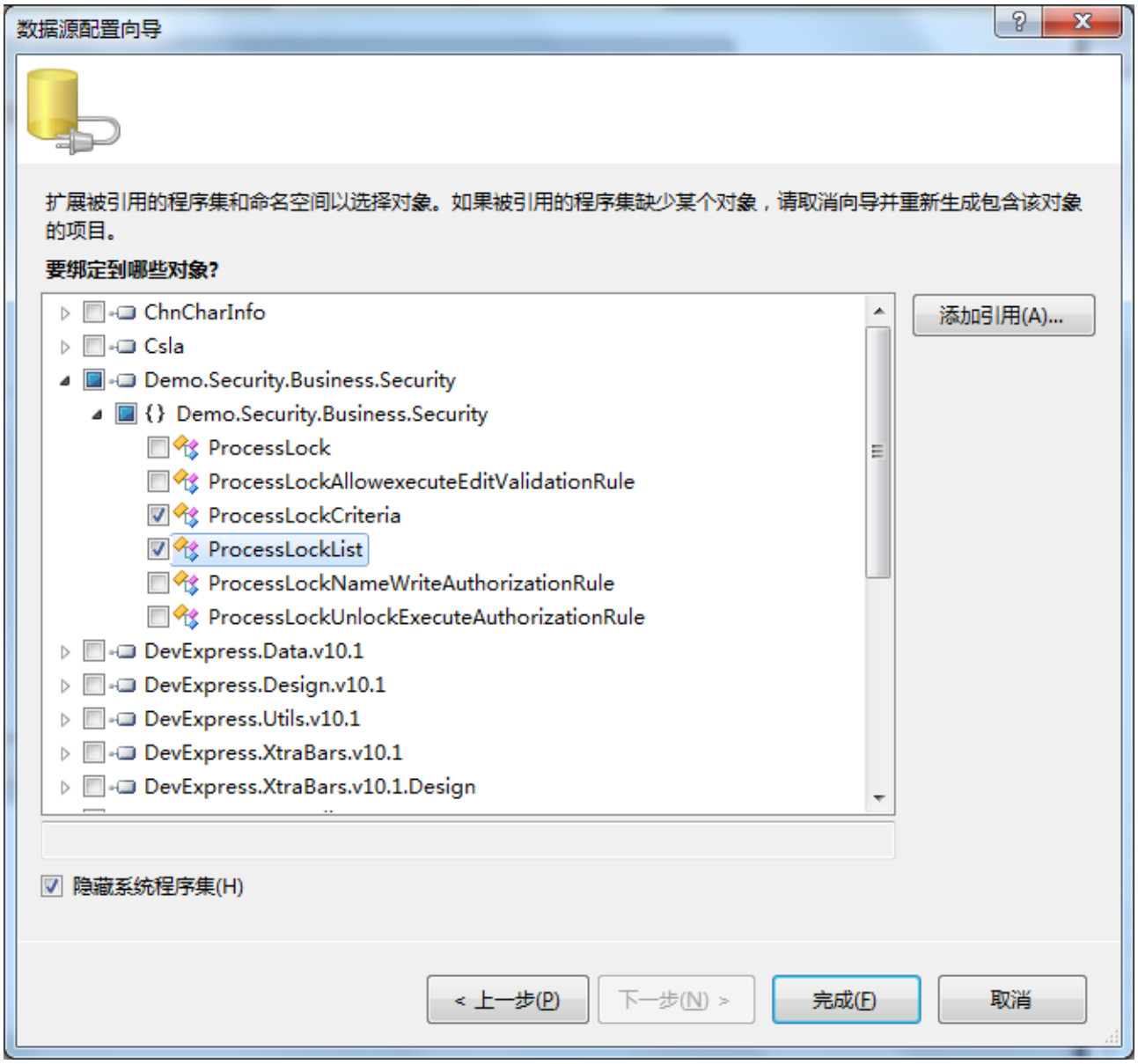

添加新项 - Demo.Security.Windows.ProcessLockManage
已安装的模板
排序依据: 默认值
搜索 已安装的模板
Visual C# 项
Web
Windows Forms
WPF
常规
代码
数据
Reporting
Workflow
联机模板
DevExpress Form v10.1 Visual C# 项
DevExpress User Control v10.1 Visual C# 项
DevExpress RibbonForm v10.1 Visual C# 项
Persistent Classes 10.1 Visual C# 项
Persistent Object 10.1 Visual C# 项
XtraReport Class v10.1 Visual C# 项
XtraReport Wizard v10.1 Visual C# 项
XtraSchedulerReport Class v10.1 Visual C# 项
Inherited XtraReport Class v10.1 Visual C# 项
类 Visual C# 项
接口 Visual C# 项
Windows 窗体 Visual C# 项
类型: Visual C# 项
空白 Windows 窗体
名称(N): ProcessLockManageForm.cs
添加(A)
取消

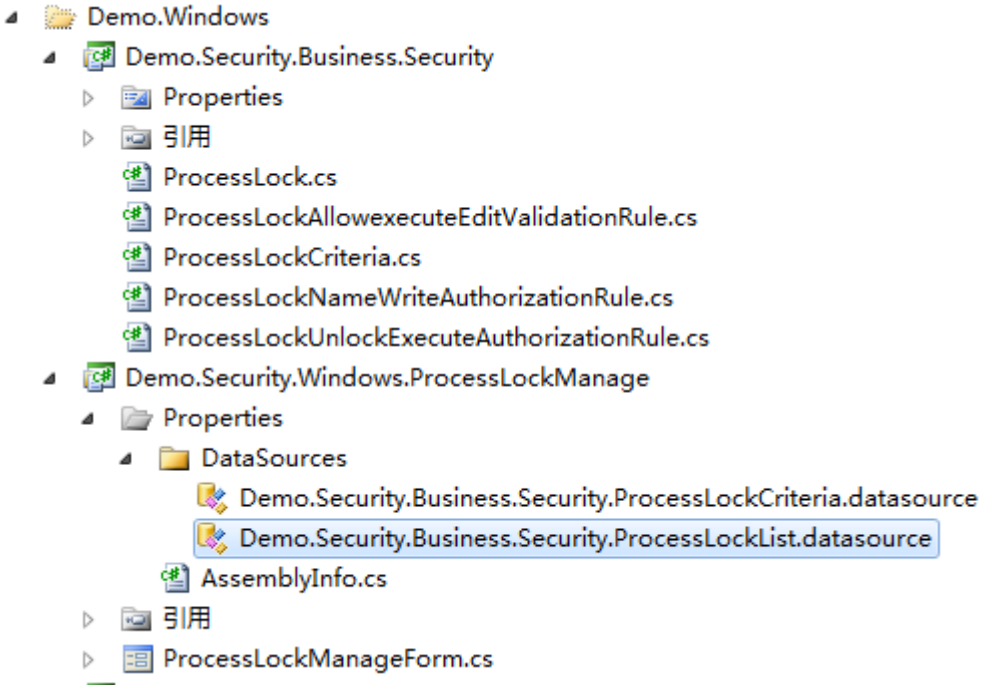
添加数据源：


编辑(E)
视图(V)
项目(P)
生成(B)
调试(D)
团队(M)
数据(A)
格式(O)
工具(T)
体系结构(C)
测试(S)
ReSharper
分析(N)
显示数据源(S)
Shift+Alt+D
预览数据(P)...
添加新数据源(N)...
添加查询(Q)...
架构比较(H)
数据比较(D)
Transact-SQL 编辑器(E)
架构视图(V)
ObjectContext
ProcessLockManageForm.cs [设计]
SimpleDataPortal.cs
BusinessBase.cs
ProcessLockManageForm

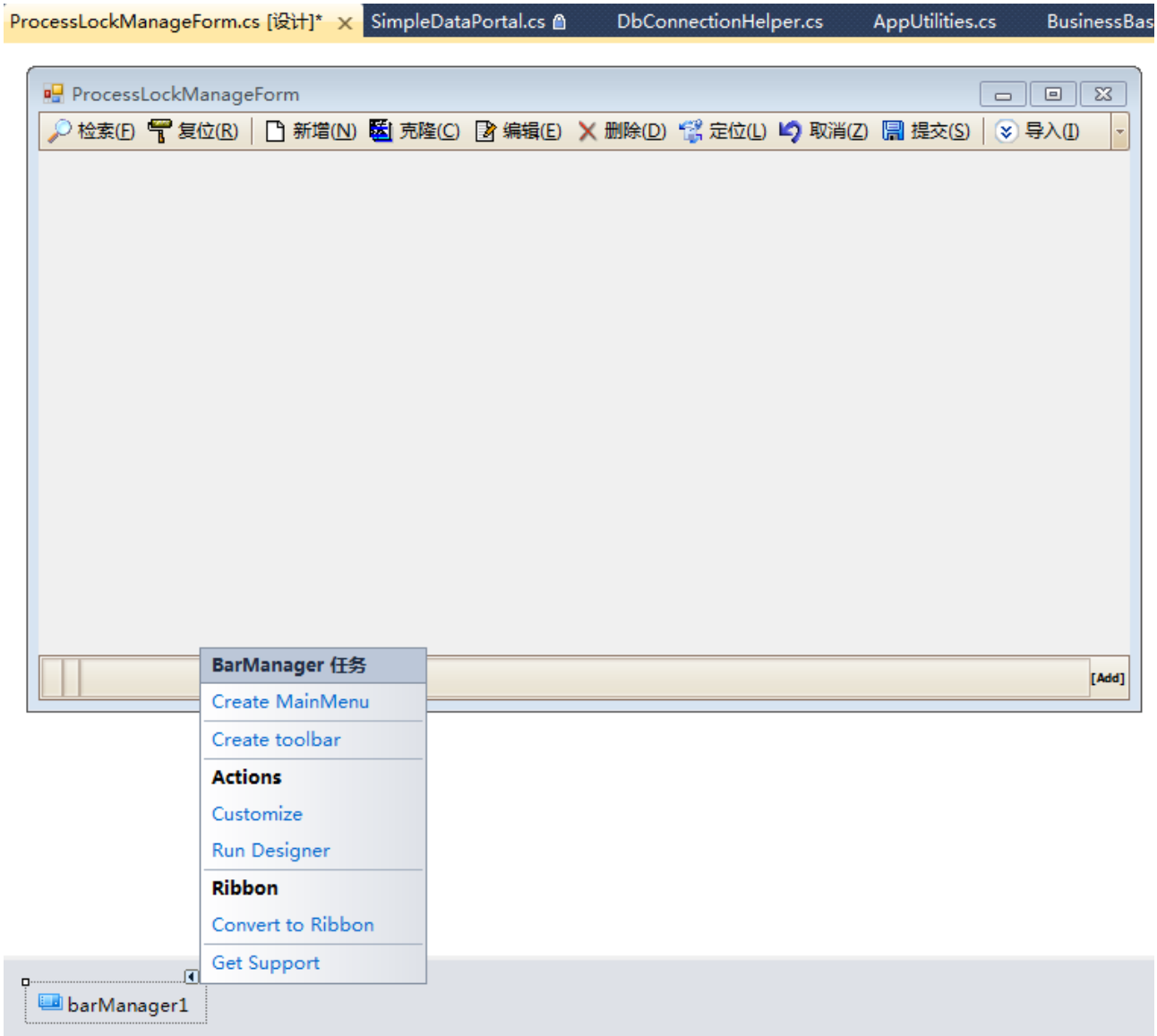

数据源配置向导
选择数据源类型
应用程序将从哪里获取数据(W)?
数据库
服务
对象
SharePoint
允许您选择可在以后用于生成数据绑定控件的对象。
< 上一步(P)
下一步(N) >
完成(F)
取消


数据源配置向导
选择数据对象
扩展被引用的程序集和命名空间以选择对象。如果被引用的程序集缺少某个对象，请取消向导并重新生成包含该对象的项目。
要绑定到哪些对象?
添加引用(A)...
隐藏系统程序集(H)
解决方案'Phe
ALAZ
CSLA
Demo
添加引用
.NET
COM
项目
浏览
最近
项目名称
项目目录
ALAZ.SystemEx
D:\Phenix.NET4\Dependencies\ALAZ\ALA
ALAZ.SystemEx.NetEx
D:\Phenix.NET4\Dependencies\ALAZ\ALAZ.SystemE
Csla (CSLA\Csla)
D:\Phenix.NET4\Dependencies\CSLA\Csla
Csla.Windows
D:\Phenix.NET4\Dependencies\CSLA\Csla
Csla.Workflow
D:\Phenix.NET4\Dependencies\CSLA\Csla
Demo.Security.Business.Security
D:\Phenix.NET4\Demo\Demo.Windows\D
Demo.Security.Windows.ProcessLoc...
D:\Phenix.NET4\Demo\Demo.Windows\D
Demo.Windows.Client.DevExpress.v...
D:\Phenix.NET4\Demo\Demo.Windows\D
Demo.Windows.Main.DevExpress.v1...
D:\Phenix.NET4\Demo\Demo.Windows\D
MyMeta
D:\Phenix.NET4\Dependencies\MyGener
Phenix.Addin
D:\Phenix.NET4\Phenix.Addin
Phenix.Business
D:\Phenix.NET4\Phenix.Business
确定
取消

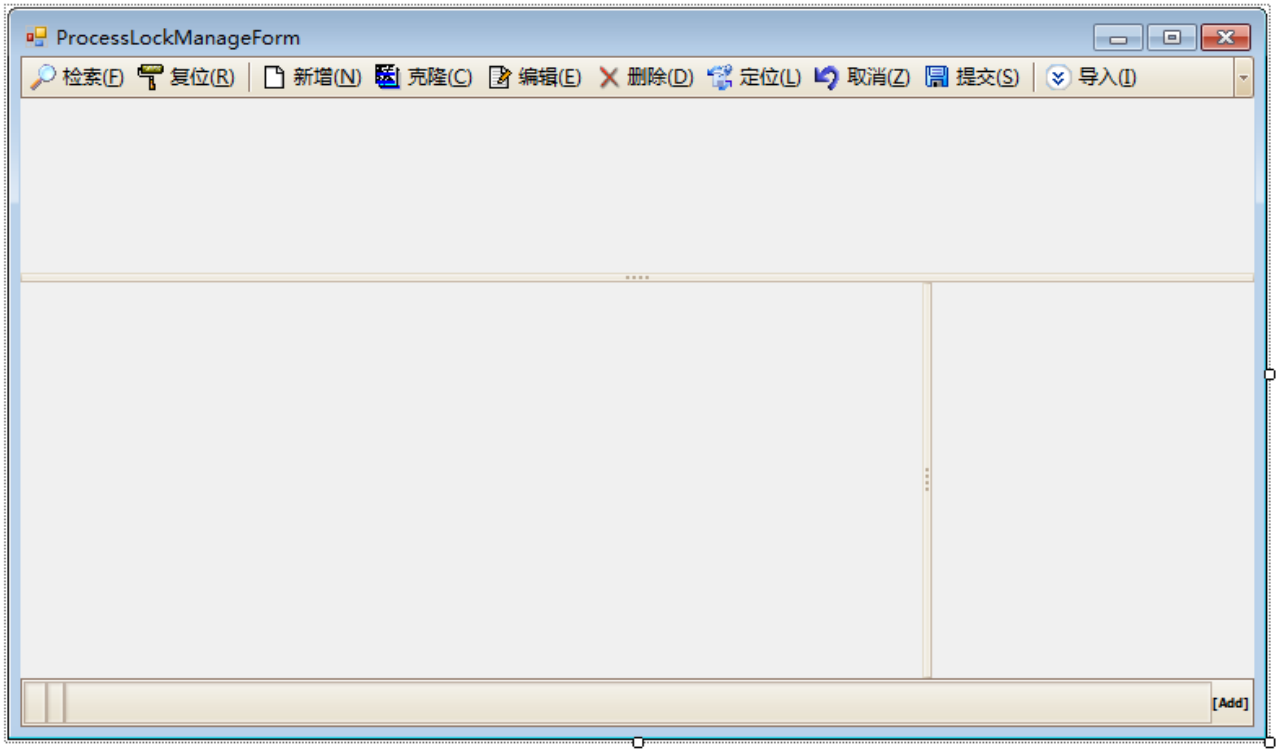

数据源配置向导
扩展被引用的程序集和命名空间以选择对象。如果被引用的程序集缺少某个对象，请取消向导并重新生成包含该对象的项目。
要绑定到哪些对象?
ChnCharInfo
Csla
Demo.Security.Business.Security
{} Demo.Security.Business.Security
ProcessLock
ProcessLockAllowexecuteEditValidationRule
ProcessLockCriteria
ProcessLockList
ProcessLockNameWriteAuthorizationRule
ProcessLockUnlockExecuteAuthorizationRule
DevExpress.Data.v10.1
DevExpress.Design.v10.1
DevExpress.Utils.v10.1
DevExpress.XtraBars.v10.1
DevExpress.XtraBars.v10.1.Design
添加引用(A)...
隐藏系统程序集(H)
< 上一步(P)
下一步(N) >
完成(F)
取消


Demo.Windows
Demo.Security.Business.Security
Properties
引用
ProcessLock.cs
ProcessLockAllowexecuteEditValidationRule.cs
ProcessLockCriteria.cs
ProcessLockNameWriteAuthorizationRule.cs
ProcessLockUnlockExecuteAuthorizationRule.cs
Demo.Security.Windows.ProcessLockManage
Properties
DataSources
Demo.Security.Business.Security.ProcessLockCriteria.datasource
Demo.Security.Business.Security.ProcessLockList.datasource
AssemblyInfo.cs
引用
ProcessLockManageForm.cs

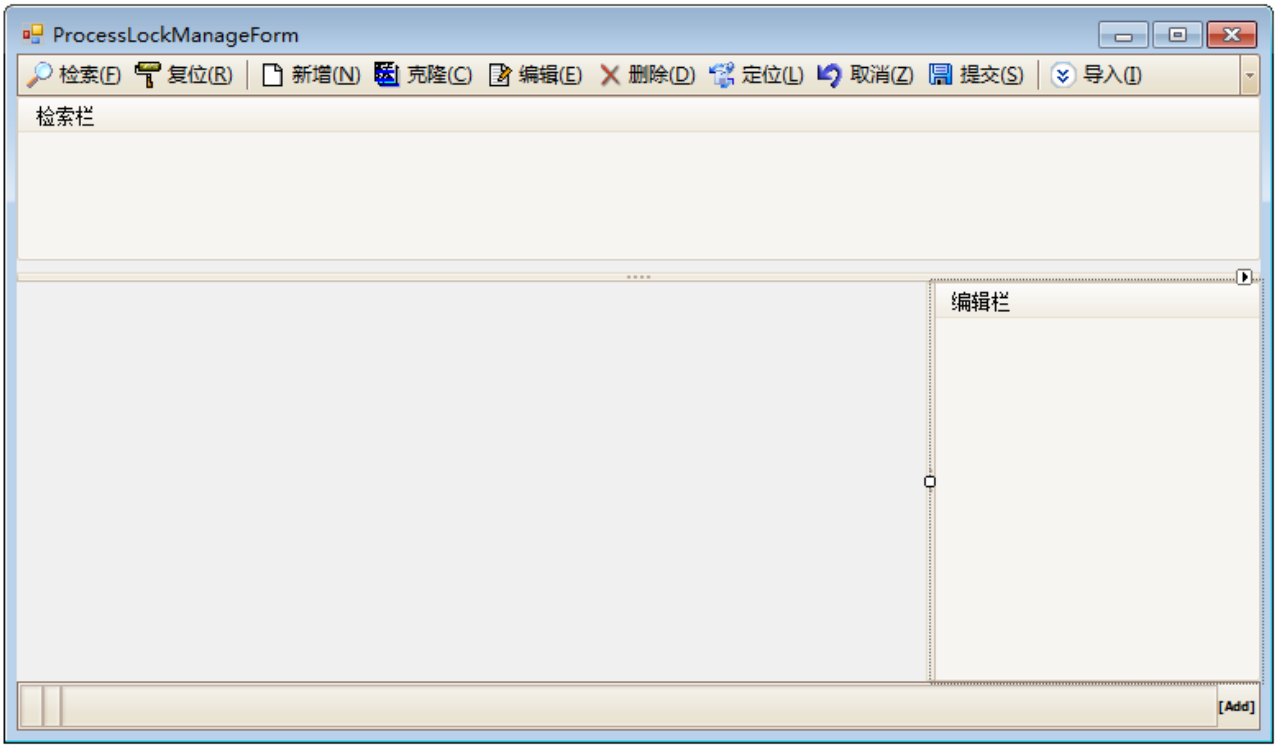
将 BarManager 组件拖到设计界面上：



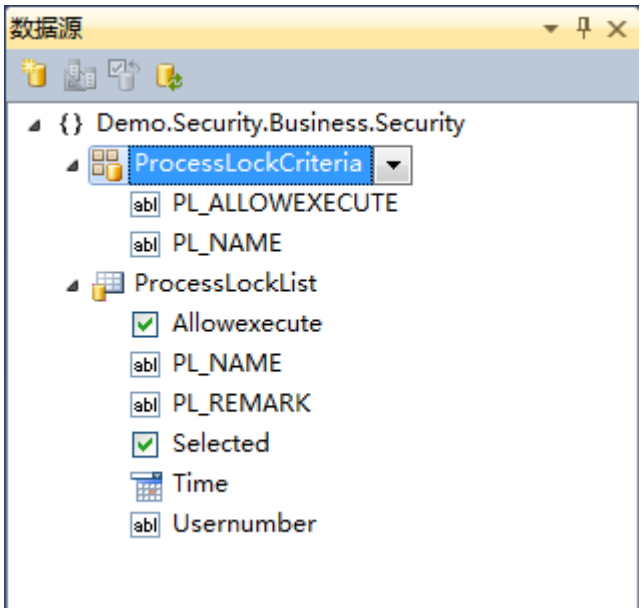
拖放隔离栏：

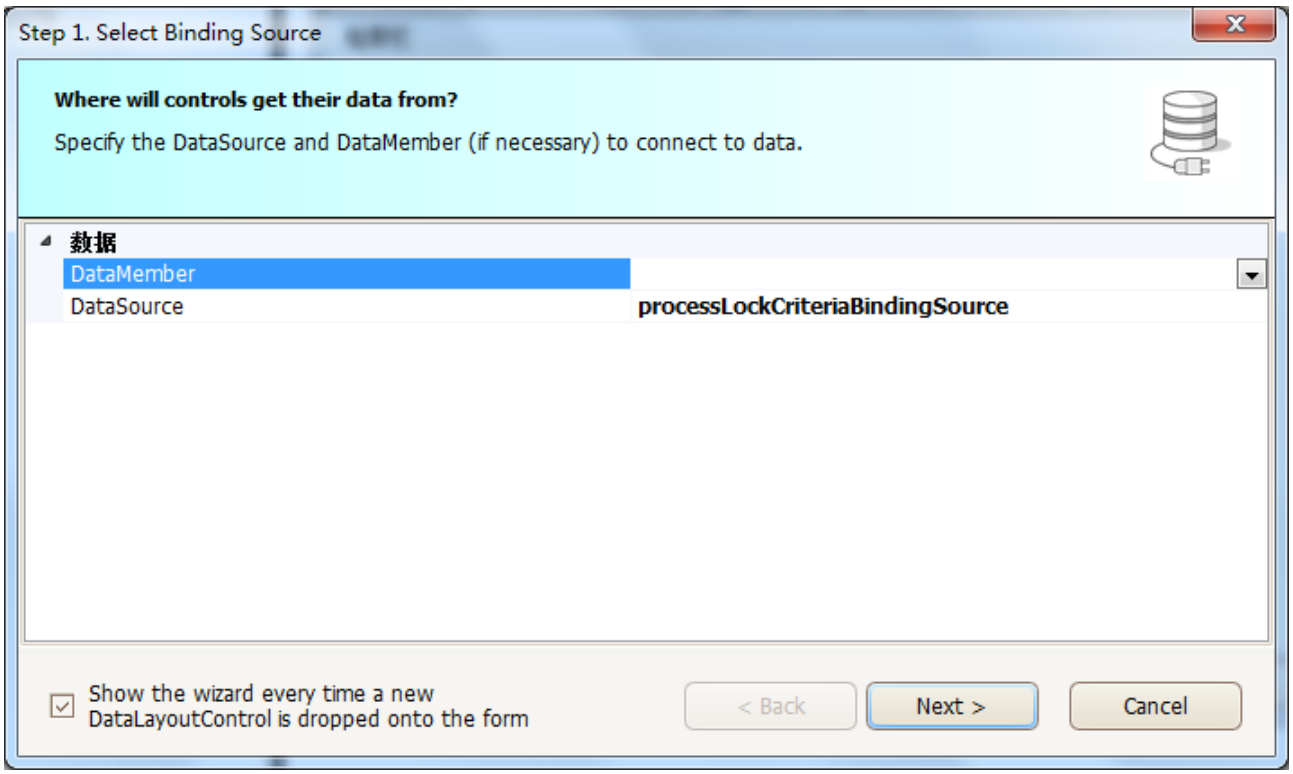


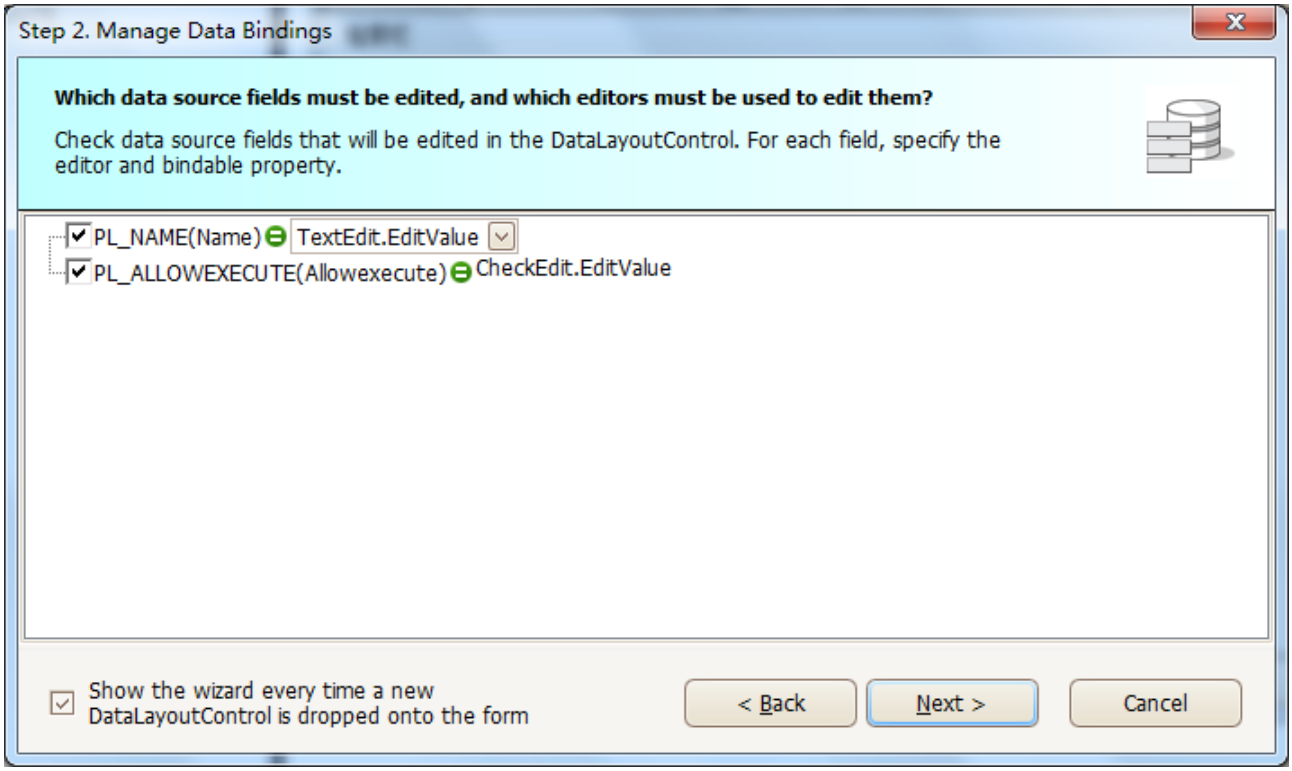
拖放检索框、编辑框：

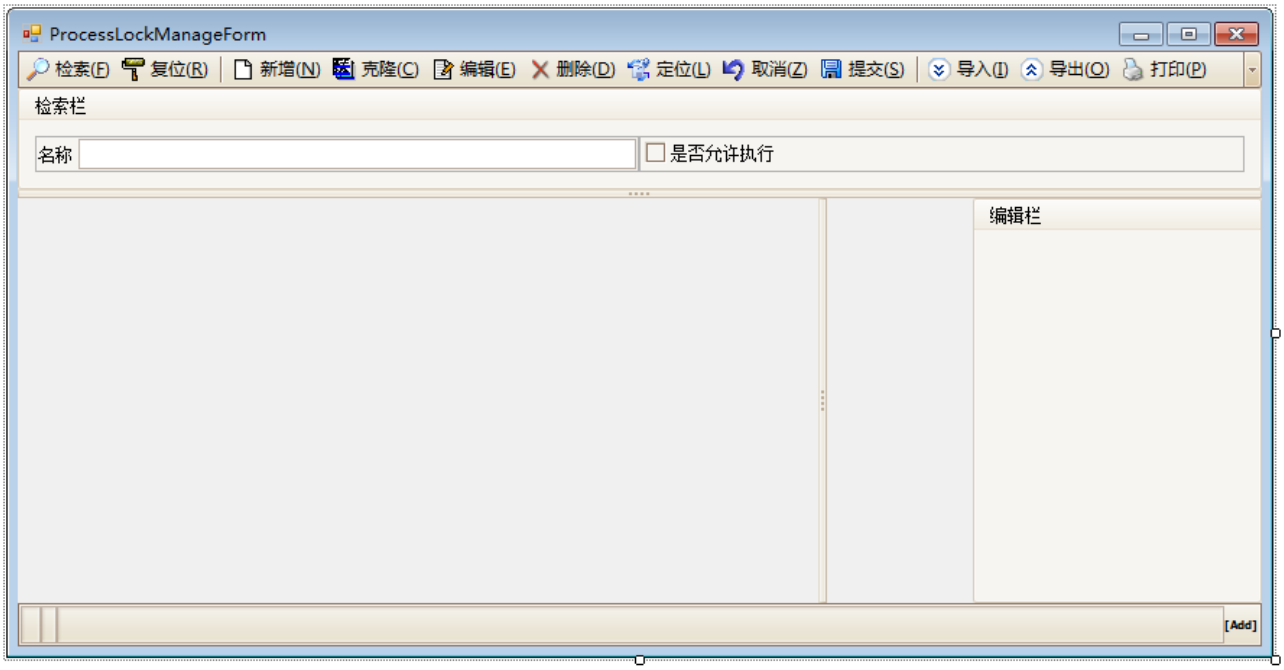


拖放检索数据源 ProcessLockCriteria：

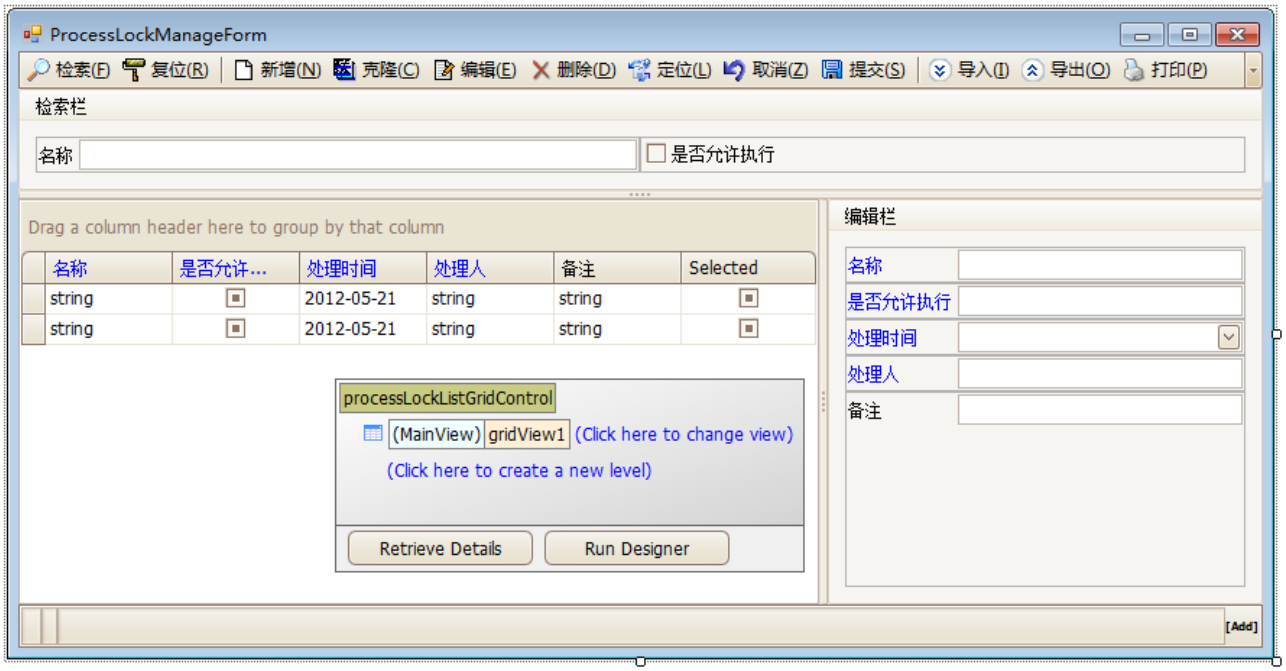

数据源
Demo.Security.Business.Security
ProcessLockCriteria
PL_ALLOWEXECUTE
PL_NAME
ProcessLockList
Allowexecute
PL_NAME
PL_REMARK
Selected
Time
Usernumber


Step 1. Select Binding Source
Where will controls get their data from?
Specify the DataSource and DataMember (if necessary) to connect to data.
数据
DataMember
DataSource
processLockCriteriaBindingSource
Show the wizard every time a new DataLayoutControl is dropped onto the form
< Back
Next >
Cancel

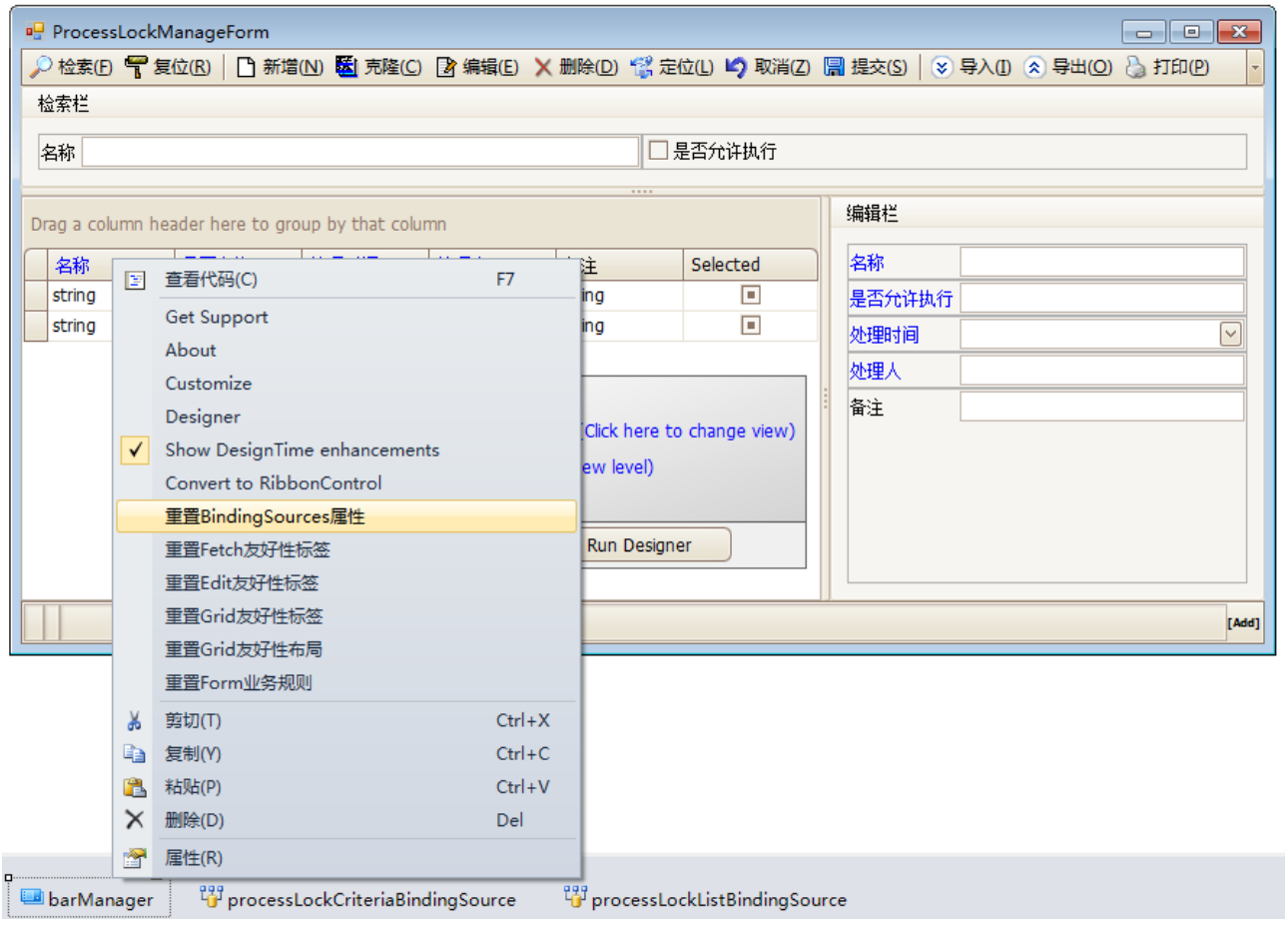




以相同的方式拖放编辑数据源 ProcessLock:

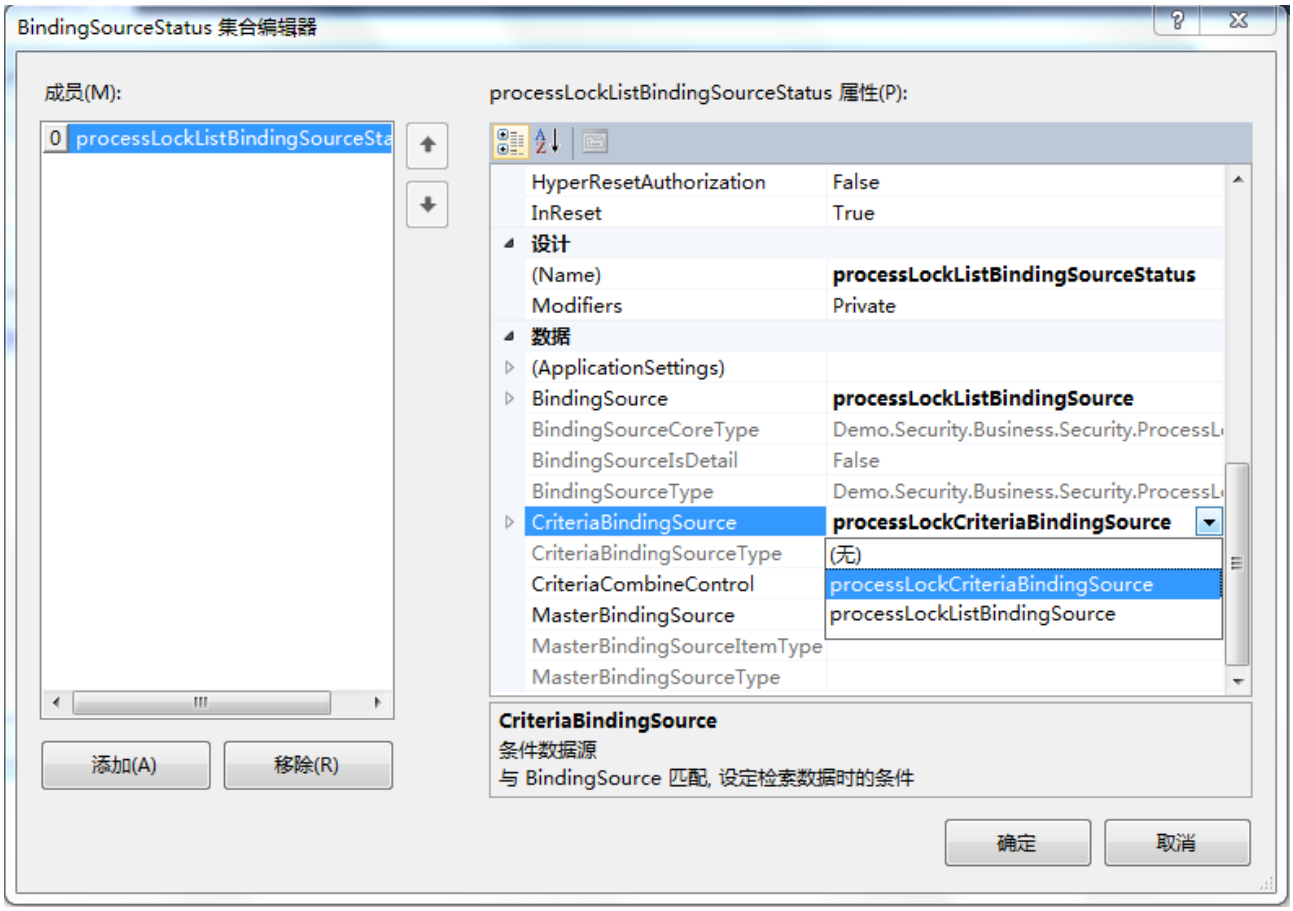


重置 BindingSources 属性：

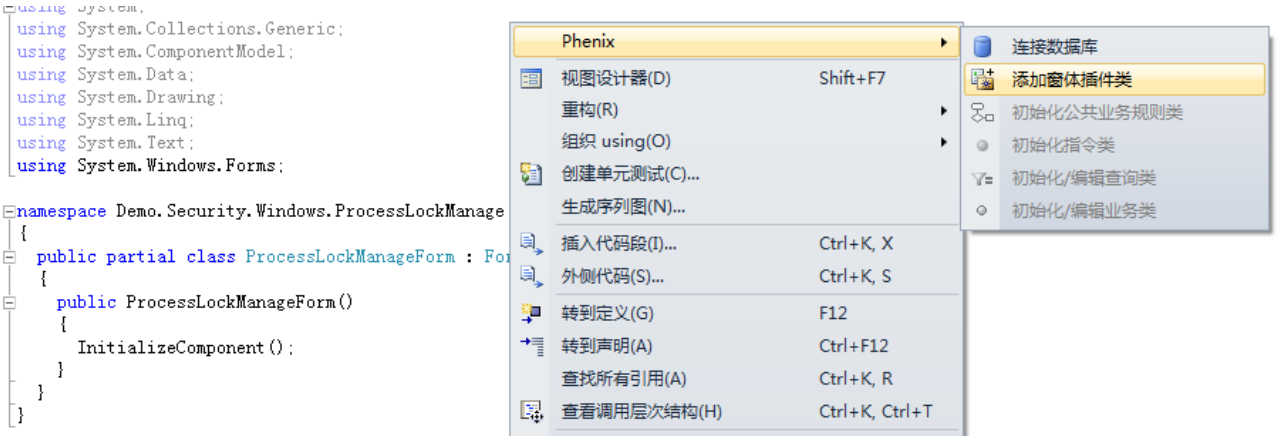


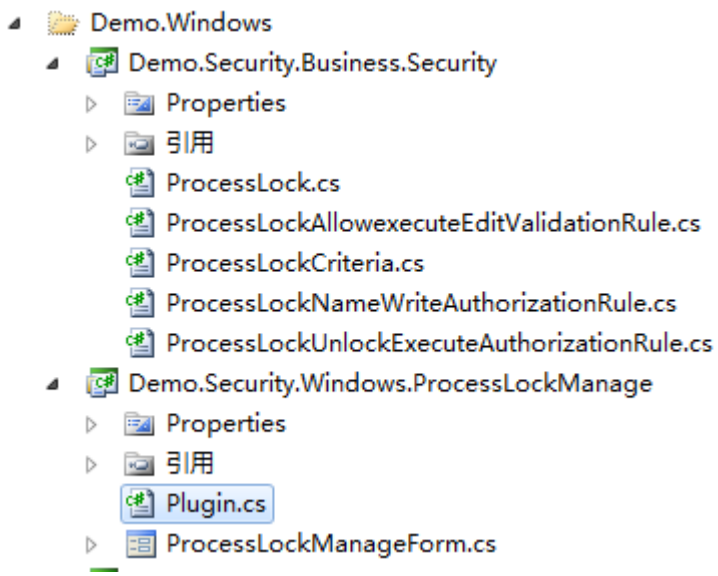
打开组件 BindingSources 属性的编辑器，可以看到组件已自动添加了针对 ProcessLock 数据源的

BindingSourceStatus，然后我们手工设置其检索数据源 ProcessLockCriteria：



最后，将本工程封装为窗体插件：



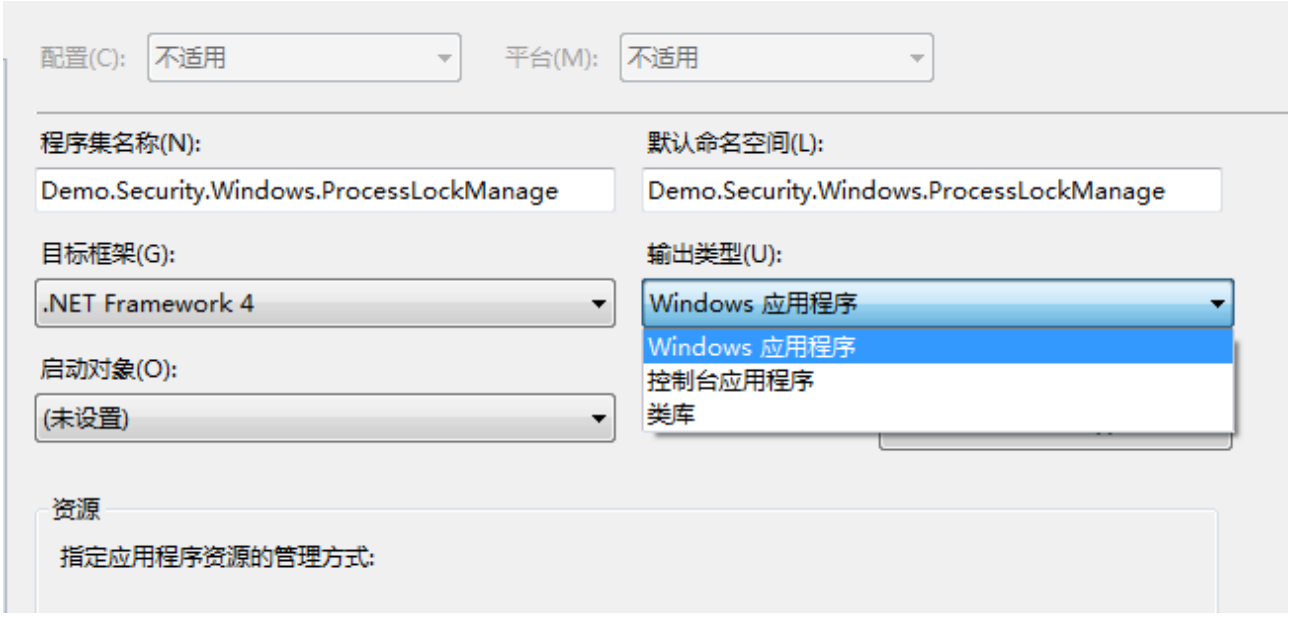


插件代码如下：

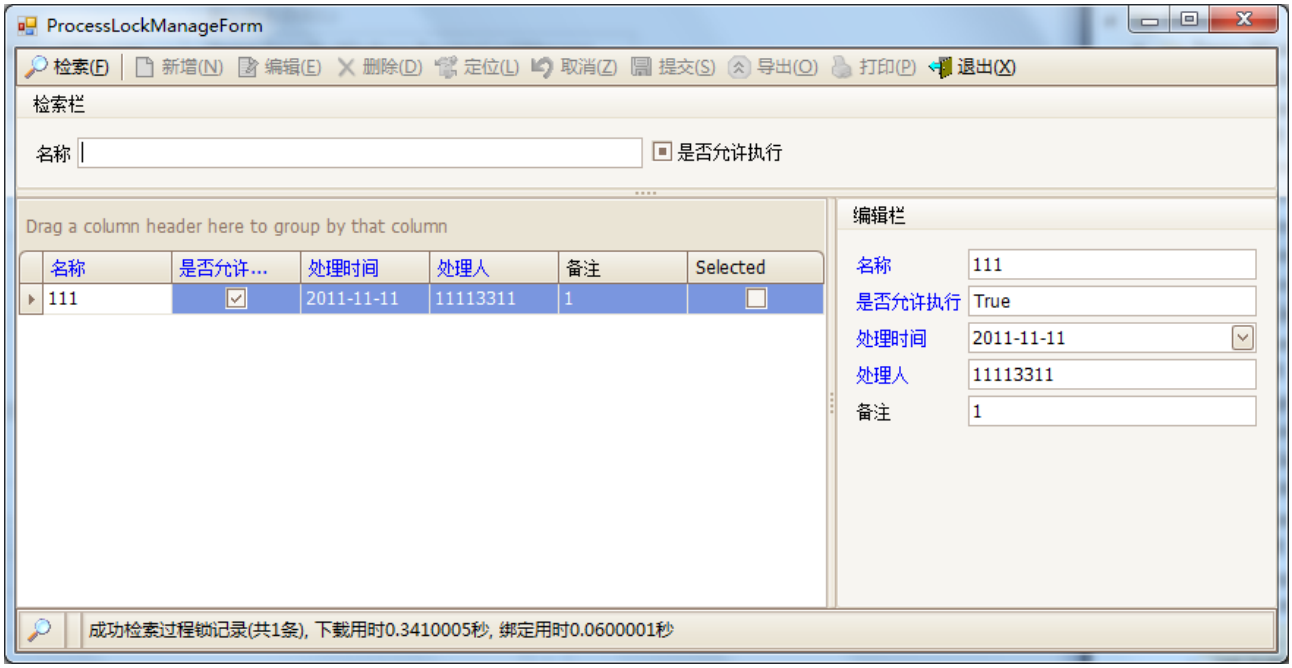
```
public class Plugin : PluginBase<Plugin>
{
  private BaseForm _mainForm;

  /// <summary>
  /// 分析消息
  /// 由 PluginHost 调用
  /// </summary>
  /// <param name="message">消息</param>
  /// <returns>按需返回</returns>
  public override object AnalyseMessage(object message)
  {
    BaseForm ownerForm = message as BaseForm;
    if (ownerForm != null && ownerForm.IsMdiContainer)
    {
      _mainForm = BaseForm.ExecuteMdi<ProcessLockManageForm>(ownerForm);
      return _mainForm;
    }
    if (_mainForm != null)
      return _mainForm.AnalyseMessage(message);
    return BaseForm.ExecuteDialog<ProcessLockManageForm>(message);
  }
}

/// <summary>
/// 可变更程序集输出类型以用于调试
/// </summary>
static class Program
```

```
{
  [STAThread]
  static void Main()
  {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    //设为调试状态
    Phenix.Core.AppConfig.Debugging = true;
    //模拟登陆
    Phenix.Business.Security.UserPrincipal.User =
      Phenix.Business.Security.UserPrincipal.CreateAuthenticated(
        UserIdentity.AdminId, UserIdentity.AdminUserName, UserIdentity.AdminUserNumber,
        String.Empty, new long?(), new long?());
    Phenix.Services.Client.Library.Registration.RegisterEmbeddedWorker(false);
    //模拟启动插件
    PluginHost.Default.SendSingletonMessage("Demo.Security.Windows.ProcessLockManage", null);
  }
}
```

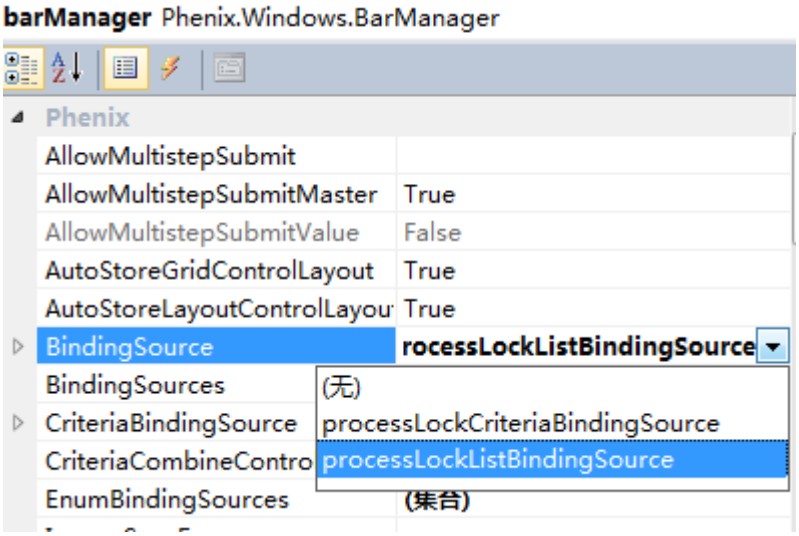
由于顺带生成了程序启动类代码，所以只要将工程输出类型改为“Windows 应用程序”就可以直接调试本插件（正式发布时，请改回“类库”重新编译；应用系统整体发布时，需将超级管理员 ADMIN 的密码修改掉）：



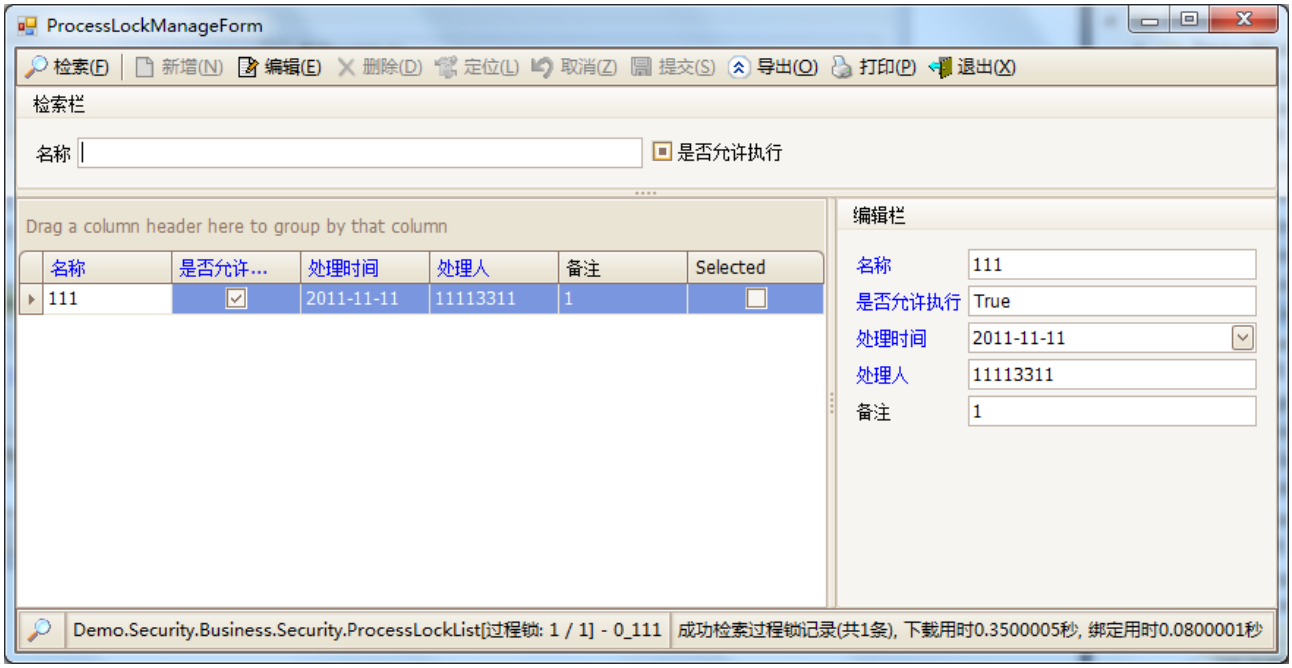
在 IDE 中启动调试后：



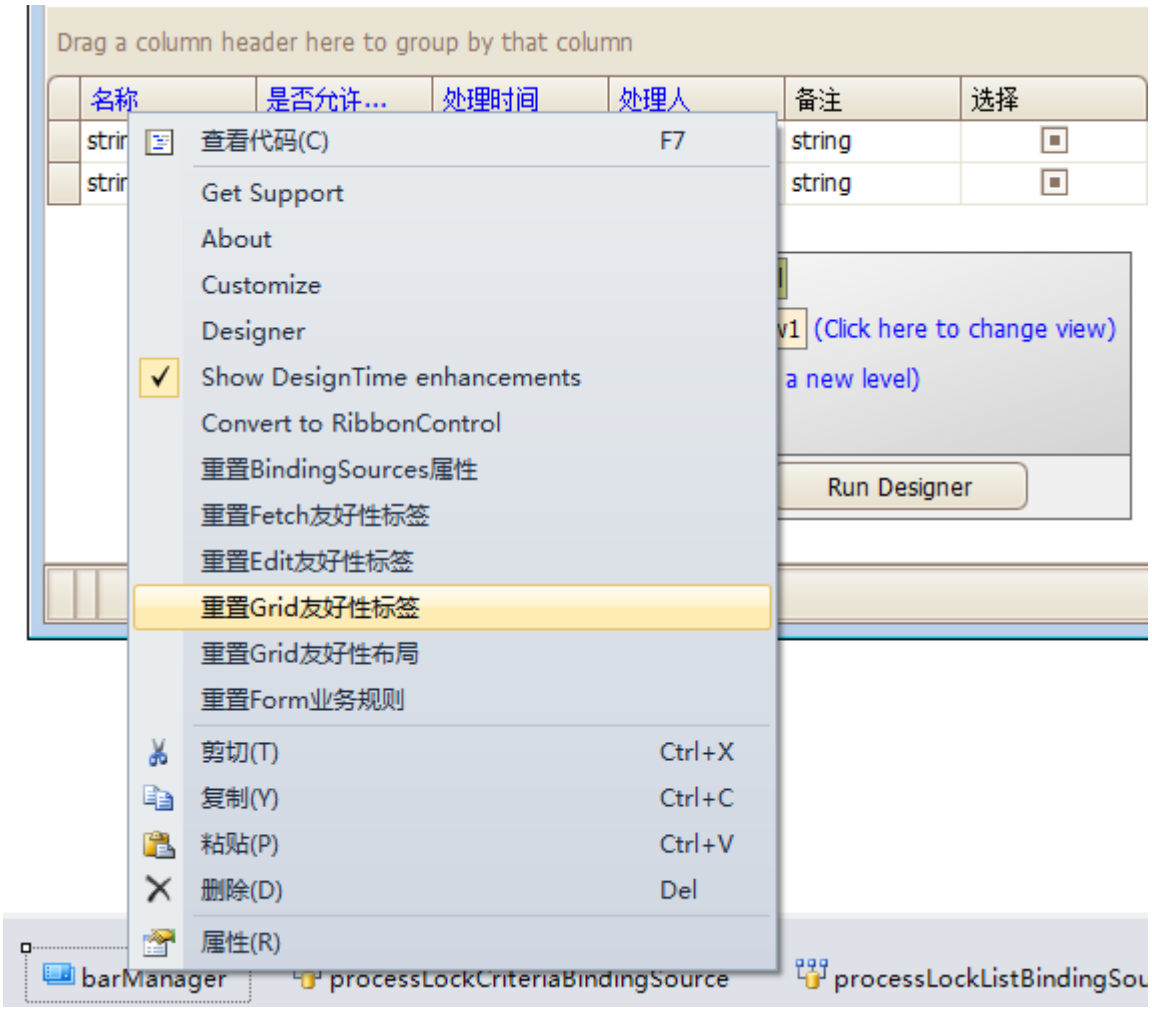
一个单表检索+编辑的功能界面就基本完成了。不过有个小缺陷，就是界面打开后检索完数据，编辑功能按钮仍然没法直接使用，这是未设置 BarManager 组件 BindingSource 的原因：



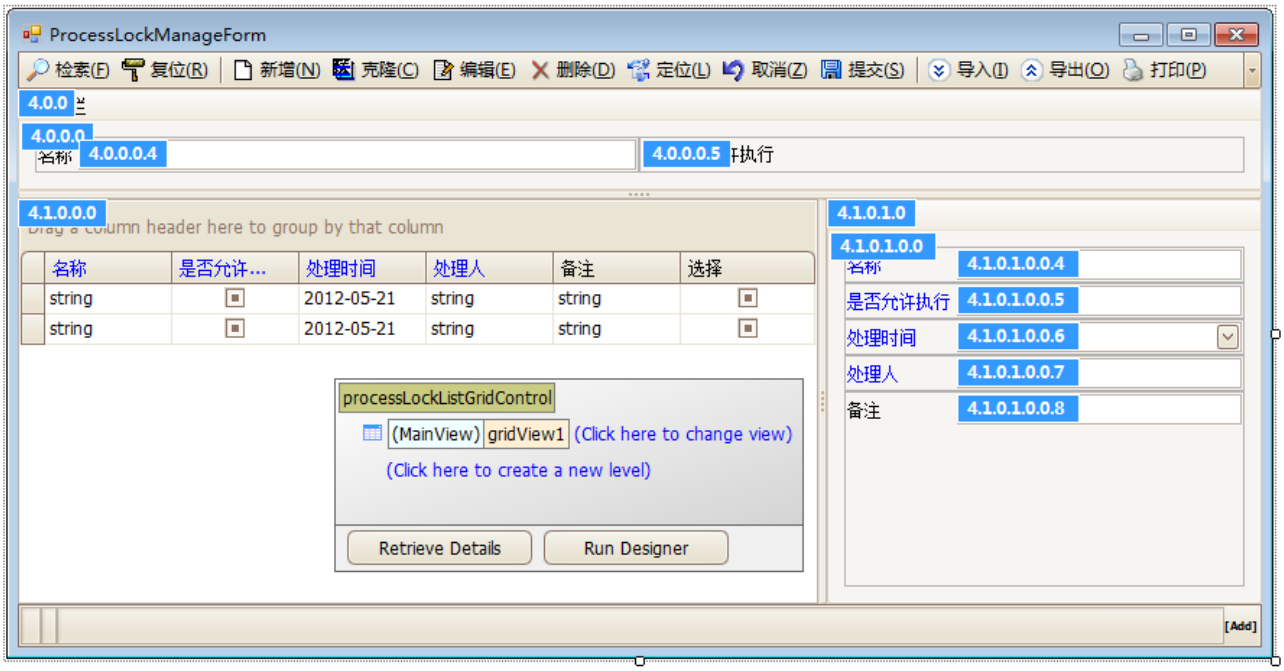
设置完成后再次启动调试，点取检索功能按钮后，编辑功能按钮就能用了：



但还发现有些标签未置换正确，这只要通过 BarManager 组件重置一下所有的友好性标签就可以了：



最后提请注意的是，需要检查一下界面控件的 tab 键顺序：

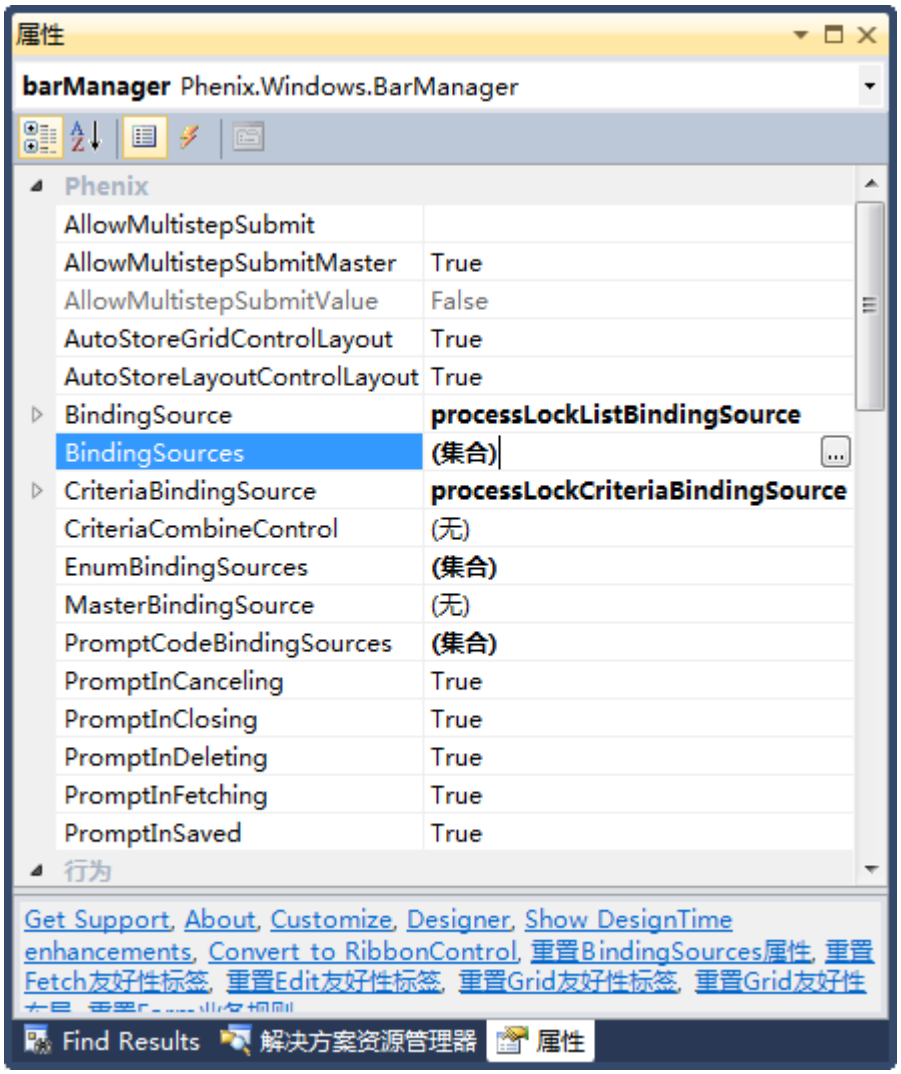


10.2 属性

下面介绍的是提供给 IDE 设计界面的可配置属性。

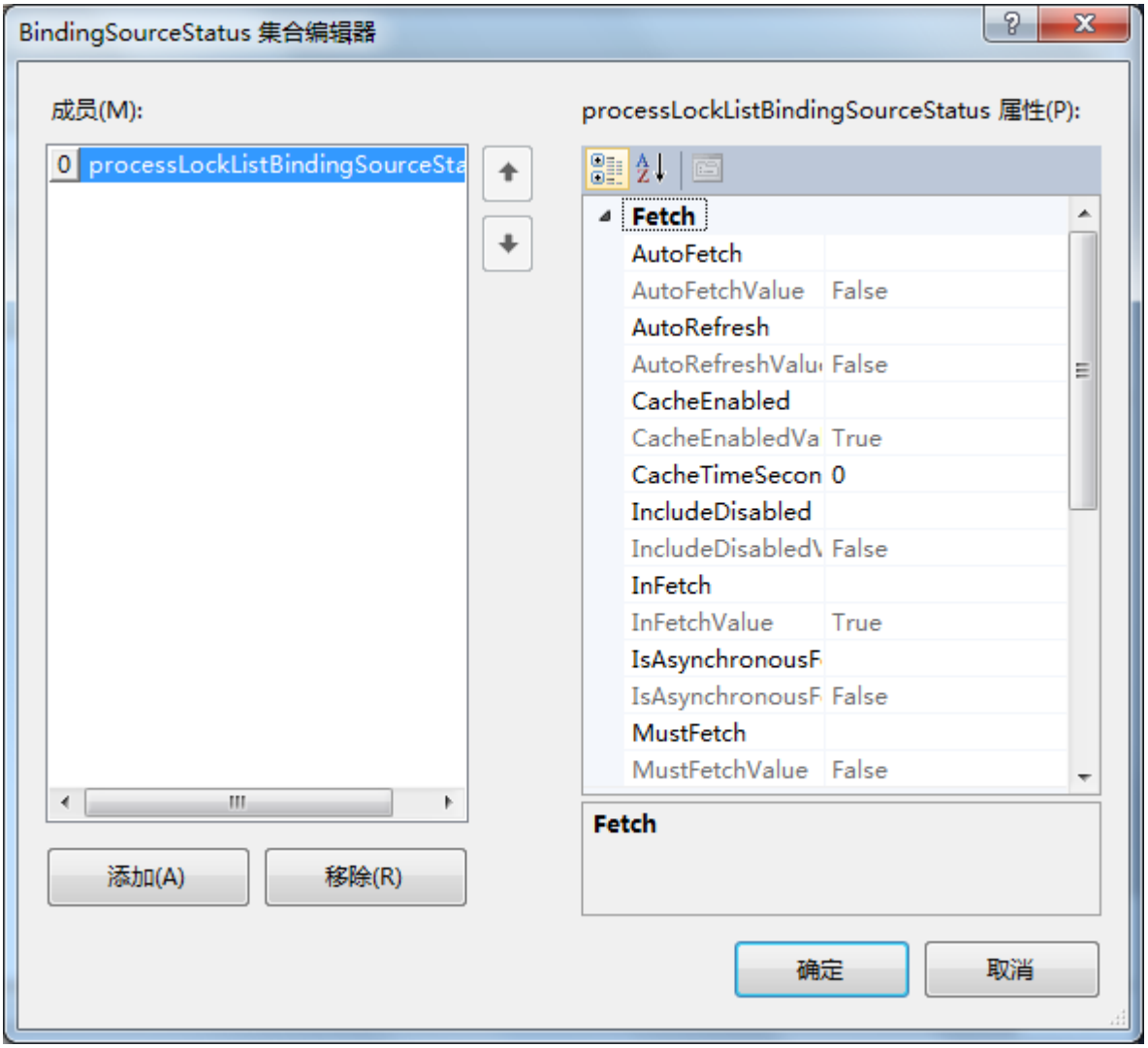
由于一些属性之间有复杂的关联性，与这些 XX 属性对应的 XXValue 只读属性为实际被应用的值，在设置这些属性时需留意。

10.2.1BindingSources 属性



添加进 BindingSources 属性的 BindingSource 将被 BarManager 组件管理上：Tools Bar 上的功能按钮、Status Bar 上的状态信息、被绑定的控件，将根据当前数据集的状态，实现缺省的界面逻辑。

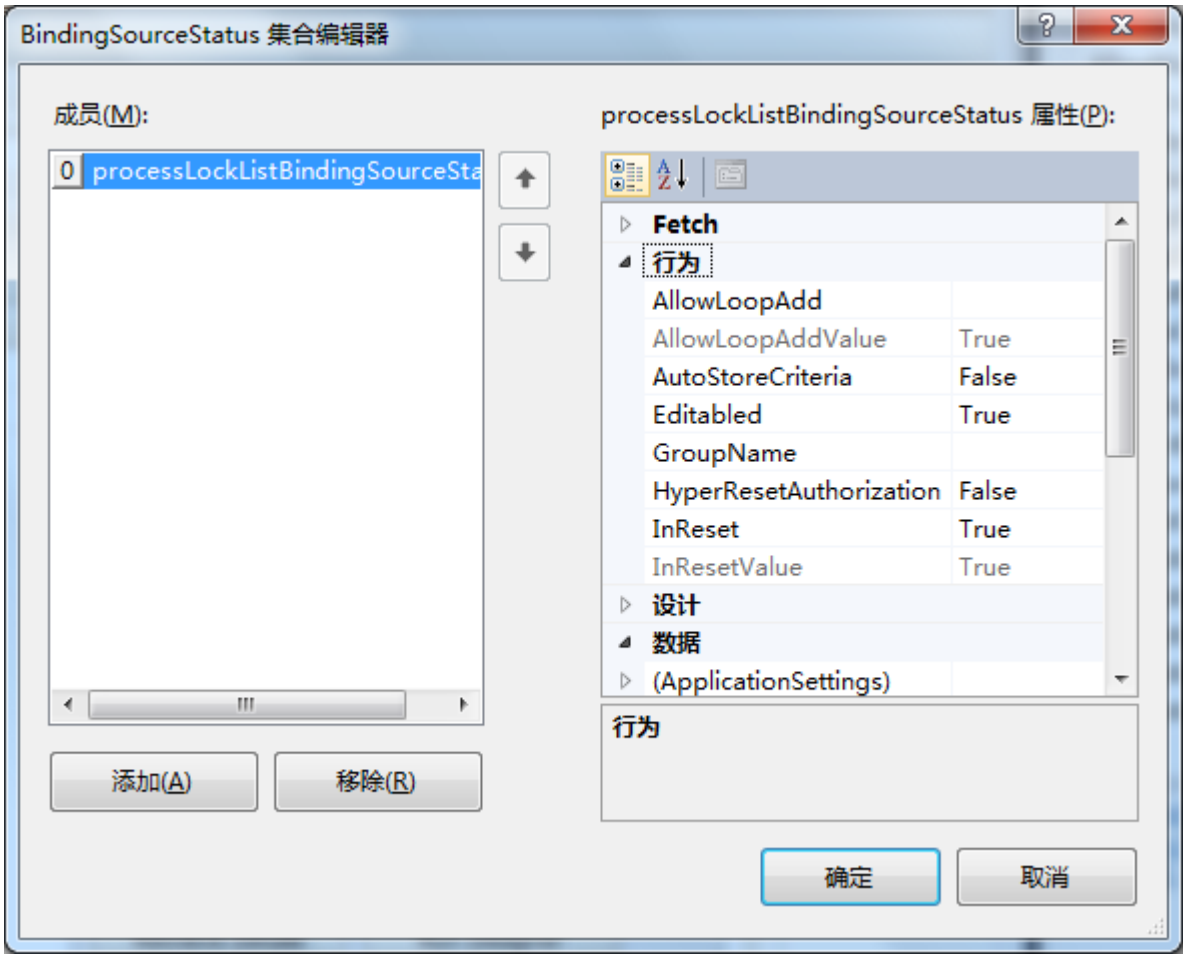
BindingSourceStatus 中与检索数据（Fetch）相关的属性：



属性	说明	备注
AutoFetch	是否自动 Fetch?	仅适用于主数据集；如设置为 true，界面 Shown 事件触发时本数据集会被自动 Fetch；
AutoRefresh	是否自动刷新?	仅适用于只读数据集&主数据集&异步 Fetch；如设置为 true，本进程内所有界面上的相关数据集内容发生修改且提交到数据库后，本数据集会被自动 Fetch 以保持内容一致；一般用于下拉选择清单数据集的内容与其编辑界面数据集内容的同步，比如本窗体内客户信息数据集被用于下拉选择，当开启客户信息编辑界面修改且提交客户信息后，本窗体内的客户信息数据集将会被自动 Fetch；
CacheEnabled	Fetch 时是否需要缓存对象?	仅适用于只读数据集&主数据集；如设置为 true，数据集会被缓存在服务端和客户端，以空间换性能；
IncludeDisabled	Fetch 时是否包含禁用记录?	缺省下 IncludeDisabled 等于 false；“禁用记录”是指业务对象

		里被打上 FieldAttribute.IsDisabledColumn = true 标记的字段值等于 CodingStandards.DefaultDisabledTrueValue 的记录；缺省下，字段名后缀为“_DISABLED”的是禁用标志字段，当该字段值为 1 时，当前记录即被标示为“禁用记录”；
InFetch	是否用于 Fetch 功能?	仅适用于主数据集；用于点击 Fetch 功能按钮时，本数据集是否需要被 Fetch，即响应点击 Fetch 事件；本属性与 MustFetch、GroupName 属性匹配使用；
IsAsynchronousFetch	是否异步 Fetch?	仅适用于只读数据集&主数据集；异步 Fetch 可提升系统响应能力，适用于静默加载数据而无需担心数据完整性的业务场景；
MustFetch	必须用于 Fetch 功能?	仅适用于主数据集；本属性支撑 InFetch 的使用；如设置为 true，则不管 GroupName 怎么设置，都响应点击 Fetch 事件；

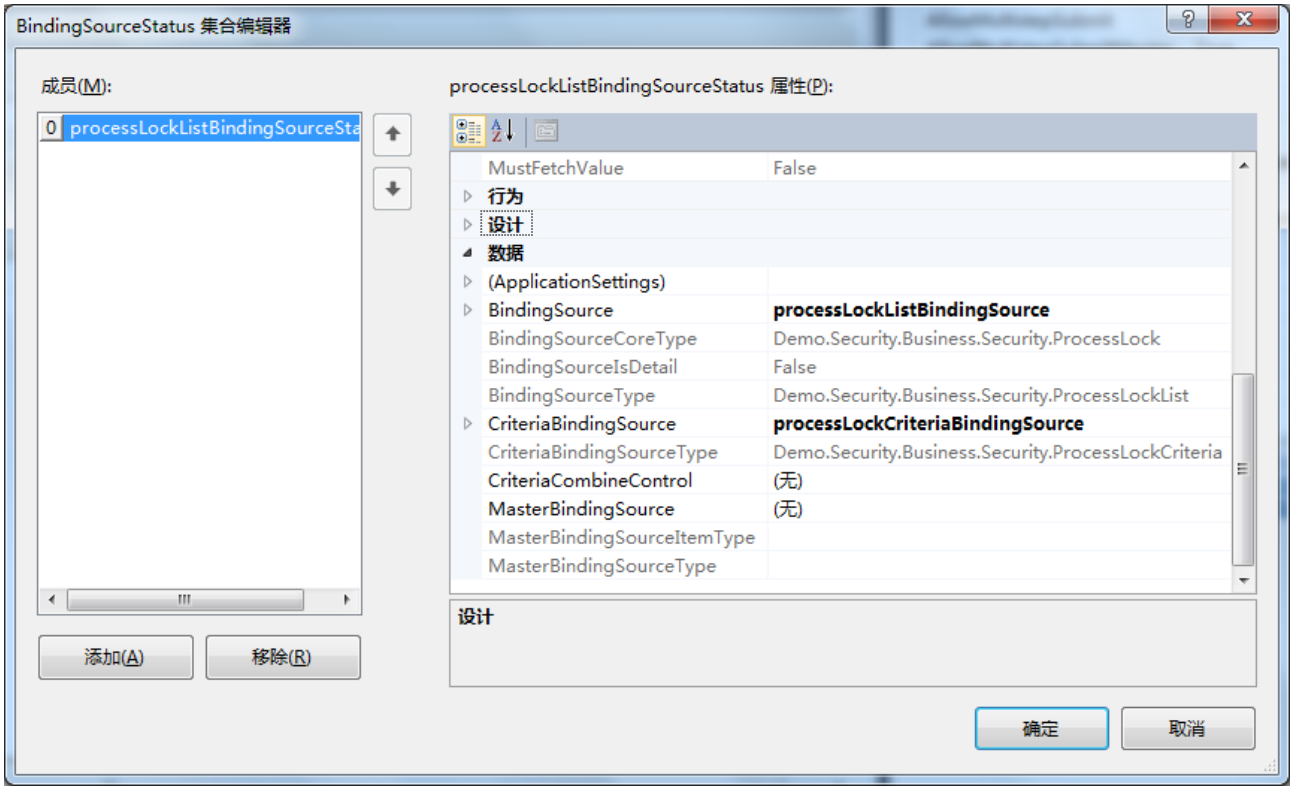
BindingSourceStatus 中“行为”大类的属性：



属性	说明	备注

AllowLoopAdd	是否允许循环添加?	仅适用于可编辑数据集；如设置为 true，当用户点击 Add 功能按钮后将处于循环添加状态；
AutoStoreCriteria	是否自动保存和恢复 Criterian 内容?	如设置为 true，当窗体关闭时，会自动保存 CriteriaBindingSource 内容并在下次窗体开启时恢复；这样，用户可以保留他最近一次的检索条件；
Editabled	是否可编辑的?	如设置为 true，将受到 BarManager 的编辑功能控制；本属性与 GroupName 属性匹配使用；IsReadOnly 的类始终是 false；
GroupName	分组名	用于区分存在处理多组数据的情形；当与 BarManager.GroupName 一致时本 BindingSourceStatus 可被应用；
HyperResetAuthorization	超灵敏重置授权?	如设置为 true，当 BindingSource 的 Current 对象属性值发生改变时也会重置一下相关控件的读写权限；适用于授权规则中用到某些对象属性值且它们会在本界面上被赋值的场景；除此之外应该保持 false，因为设置为 true 会影响到些微的界面性能；

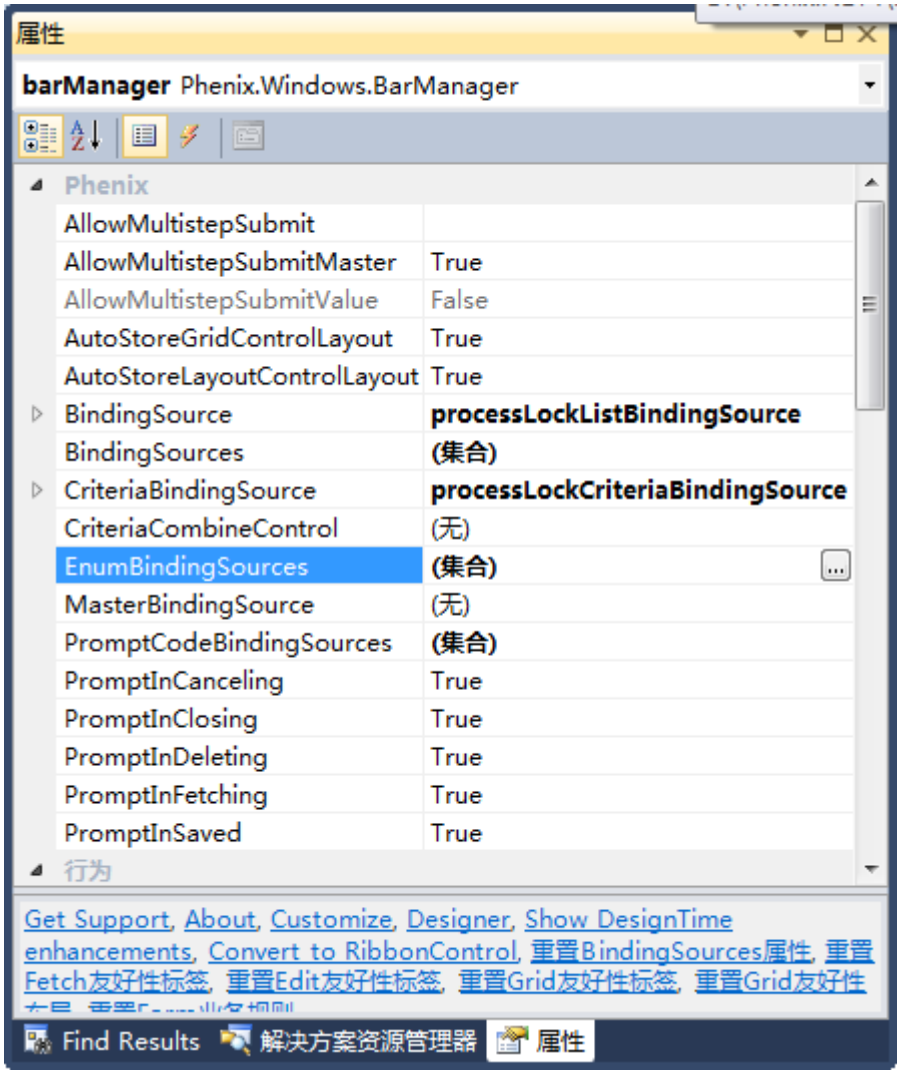
BindingSourceStatus 中“数据”大类的属性：



属性	说明	备注
BindingSource	数据源	

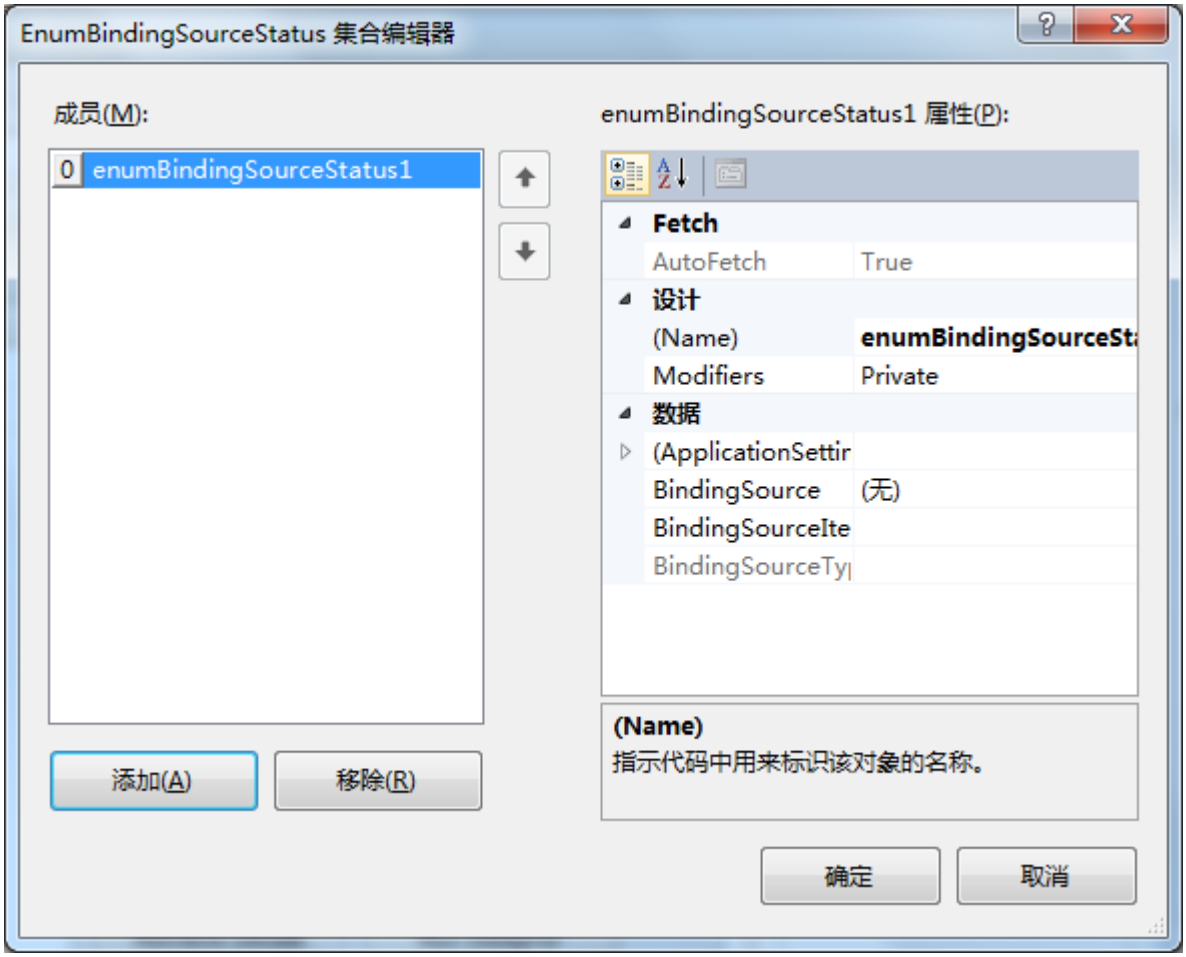
CriteriaBindingSource	条件数据源	设定检索 BindingSource 数据集时的条件；
CriteriaCombineControl	查询组合框控件	非空时有效；当 CriteriaBindingSource 为空时可使用本属性指定控件 CriteriaCombineControl.WorkingCriteriaExpression 作为检索 BindingSource 数据集时的条件；
MasterBindingSource	主数据源	非空时有效；当 BarManager.BindingSource 变更为本 BindingSource 时将使用本属性替换 BarManager.MasterBindingSource；

10.2.2EnumBindingSources 属性



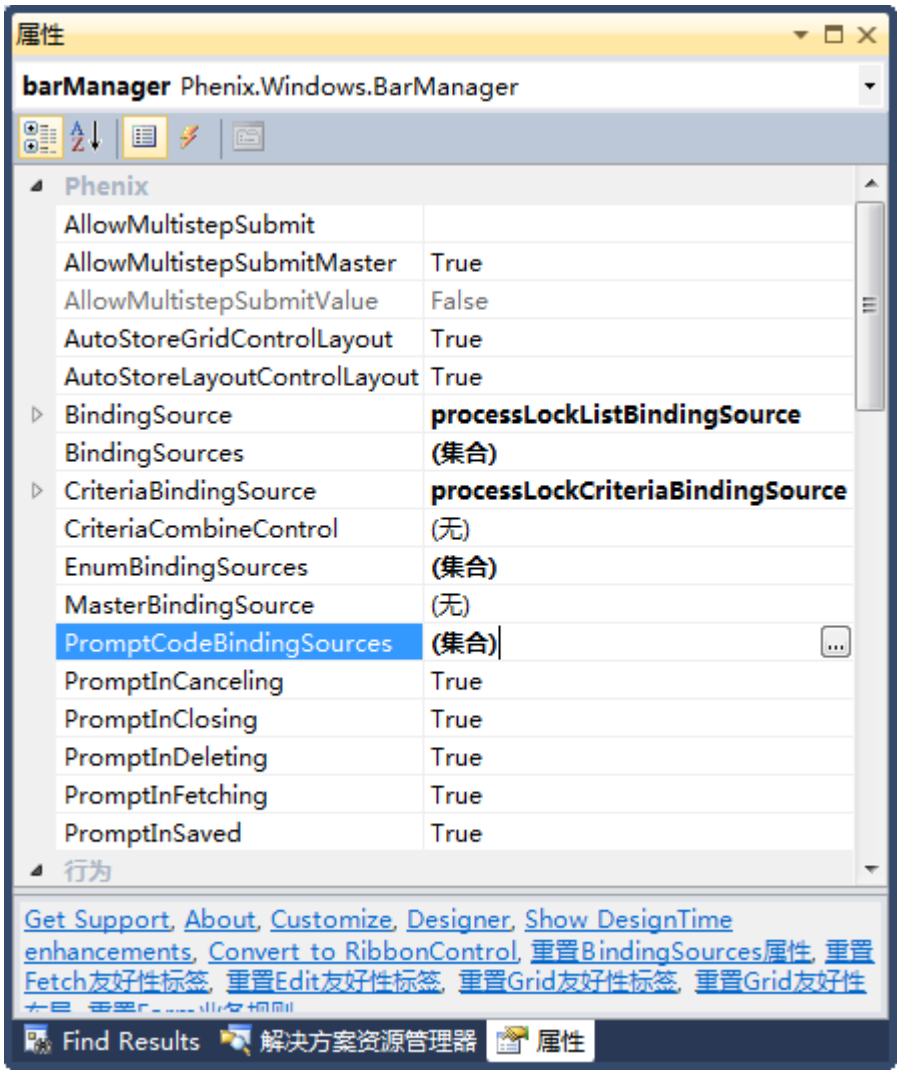
在界面设计上，枚举经常被用于下拉选择。所以，我们可以将枚举封装为普通的数据集合（承载枚举数据的队列载体为：Phenix.Core.Rule.EnumKeyCaptionCollection），能够提供与业务类一致的数据绑定设计方法，实现快速设计开发的目的。（见“03.Addin 工具使用方法”的“添加枚举标签”章节）

添加进 EnumBindingSources 的 BindingSource 将被 BarManager 组件管理上：负责为数据源自动填充 Phenix.Core.Rule.EnumKeyCaptionCollection 枚举数据集。



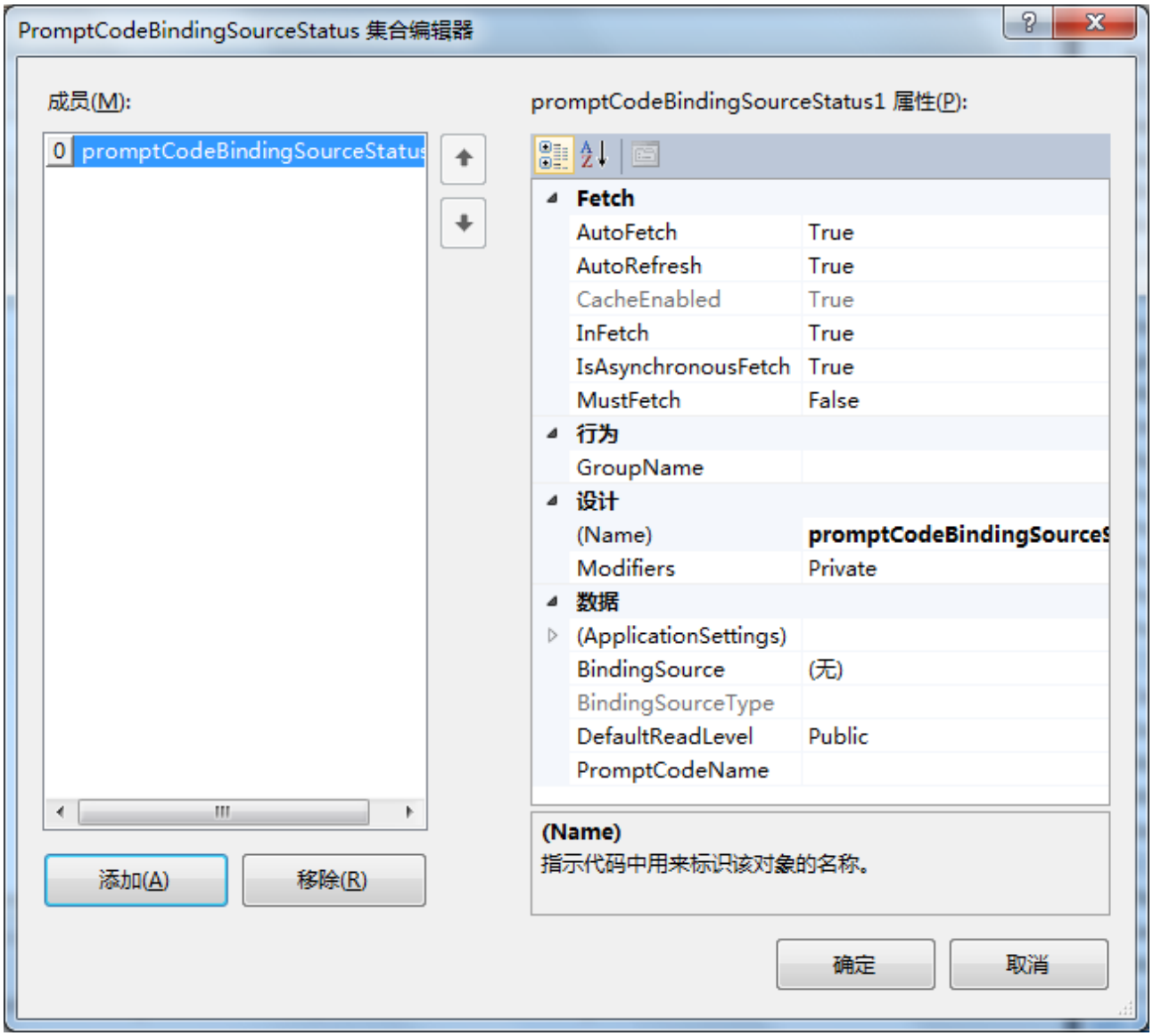
属性	说明	备注
BindingSource	数据源	DataSource 必须是 EnumKeyCaptionCollection 类；
BindingSourceItemType	数据源项的类型	设置为绑定的枚举类型；
AutoFetch	是否自动 Fetch?	在界面 Shown 时，本项 BindingSource 被自动 Fetch；

10.2.3PromptCodeBindingSources 属性



在界面设计上，除了枚举这种明确的下拉选择外，还有一种可由用户自行维护（增删改）的下拉选择清单，但这些清单的内容对应用系统无实质含义，仅协助方便用户录入而已，这就是提示码。提示码有唯一键标识、显示标签和值等属性组成，被封装在 Phenix.Core.Rule.PromptCodeKeyCaption，Phenix.Core.Rule.PromptCodeKeyCaptionCollection 为其队列载体。

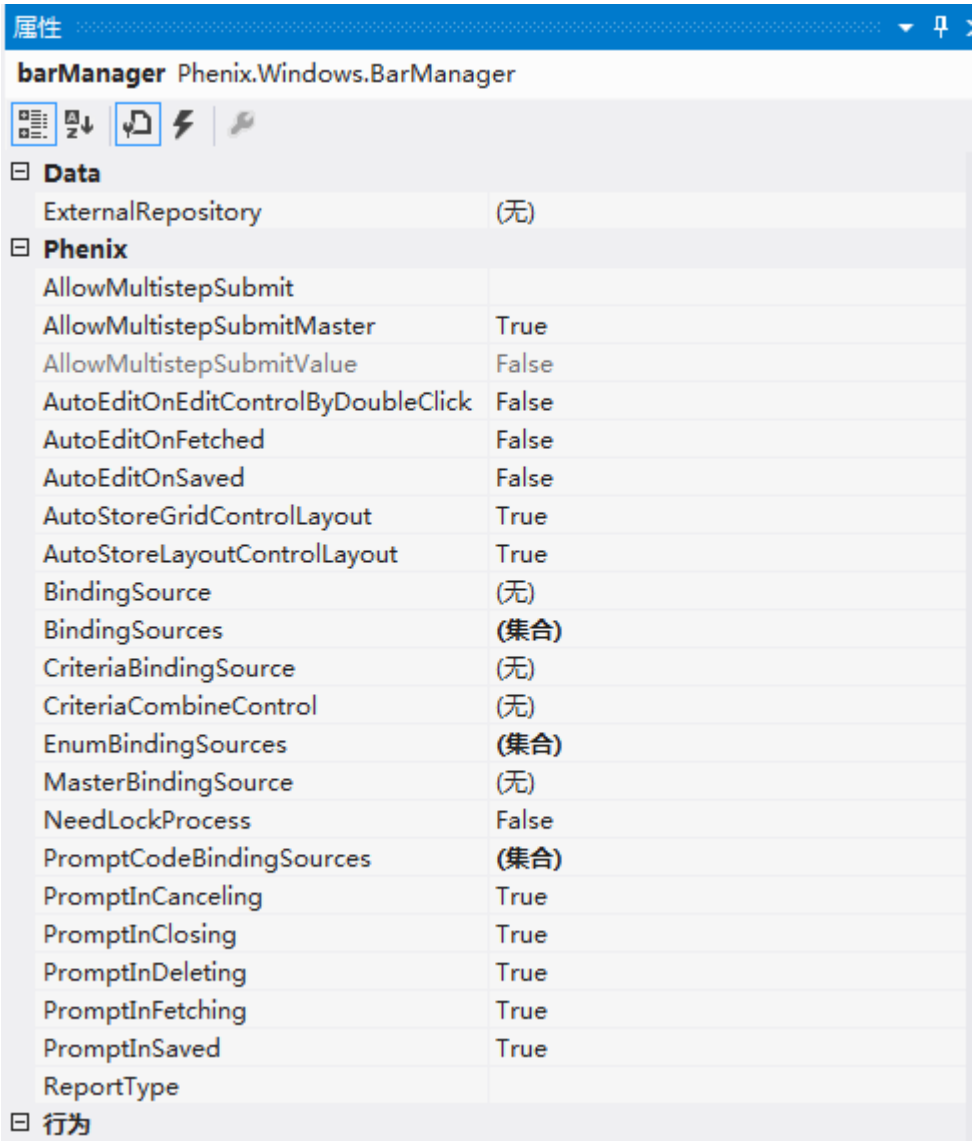
BarManager 组件对提示码（需添加进 PromptBindingSources）做了管理：负责为数据源自动填充 Phenix.Core.Rule.PromptCodeKeyCaptionCollection 提示码数据集。



属性	说明	备注
BindingSource	数据源	DataSource 必须是 PromptCodeKeyCaptionCollection 类；
PromptCodeName	提示码名	标识提示码数据集以下载和维护其内容；
DefaultReadLevel	缺省读取级别	当新增提示码时缺省定义的读取级别；为枚举值，选项见 Phenix.Core.Rule.ReadLevel；
GroupName	分组名	用于区分存在处理多组数据的情形；与 BarManager.GroupName 一致时本 PromptCodeBindingSourceStatus 可被应用；
AutoFetch	是否自动 Fetch?	在界面 Shown 时，本项 BindingSource 被自动 Fetch；
AutoRefresh	是否自动刷新?	如设置为 true，本进程内所有界面上的相关数据集内容发生修改且提交到数据库后，本数据集会被自动 Fetch 以保持内容一致；
CacheEnabled	Fetch 时是否需要缓存	

	对象?	
InFetch	是否用于 Fetch 功能?	用于点击 Fetch 功能按钮时，本数据集是否需要被 Fetch，即响应点击 Fetch 事件；本属性与 MustFetch、GroupName 属性匹配使用；
IsAsynchronousFetch	是否异步 Fetch?	异步 Fetch 可提升系统响应能力，适用于静默加载数据而无需担心数据完整性的业务场景；
MustFetch	必须用于 Fetch 功能?	本属性支撑 InFetch 的使用；如设置为 true，则不管 GroupName 怎么设置，都响应点击 Fetch 事件；

10.2.4其他属性



属性	说明	备注
AllowMultistepSubmit	允许多步提交	如设置为 true，允许连续处理

		BindingSources 的多条记录；
AllowMultistepSubmitMaster	允许多步提交主业务	AllowMultistepSubmit 等于 true 时有意义；如设置为 true，允许连续处理 MasterBindingSource 的多条记录；
AutoEditOnFetched	OnFetched 时是否自动进入编辑状态	点击 Fetch 功能按钮检索到数据后自动点击 Edit 功能按钮
AutoEditOnSaved	OnSaved 时是否自动进入编辑状态	点击 Save 功能按钮保存好数据后自动点击 Edit 功能按钮
AutoEditOnEditControlByDoubleClick	双击编辑控件时是否自动进入编辑状态	双击编辑控件后自动点击 Edit 功能按钮
AutoStoreGridControlLayout	是否自动保存和恢复 GridControl 布局	自动保存所属容器关闭前的 GridControl 布局并在下次开启时恢复；
AutoStoreLayoutControlLayout	是否自动保存和恢复 LayoutControl 布局	自动保存所属容器关闭前的 LayoutControl 布局并在下次开启时恢复；
BindingSource	数据源	当前操作的数据源；在切换焦点的时候，会根据当前控件绑定的数据源从 BindingSources 集合中检索出匹配的 BindingSource；如指定数据源，可在窗体 Shown 时正确初始化界面逻辑；
CriteriaBindingSource	条件数据源	当前操作数据源的检索条件；在切换焦点的时候，会根据当前控件绑定的数据源从 BindingSources 集合中检索出匹配的 CriteriaBindingSource；
CriteriaCombineControl	查询组合框控件	当前操作数据源的检索条件；在切换焦点的时候，会根据当前控件绑定的数据源从 BindingSources 集合中检索出匹配的 CriteriaCombineControl；
MasterBindingSource	主数据源	当前操作数据源的主数据源；在切换焦点的时候，会根据当前控件绑定的数据源从 BindingSources 集合中检索出匹配的

		MasterBindingSource；非空时以本数据源为 root 实现编辑、回滚、提交等数据操作；
PromptInFetching	在编辑状态下检索数据时提示当前处于编辑状态且已更改过数据	
PromptInDeleting	删除前提示确认删除操作	
PromptInCanceling	取消前提示确认取消操作	
PromptInSaved	保存后提示保存成功	
PromptInClosing	在编辑状态下关闭窗体时提示当前处于编辑状态且已更改过数据	
NeedLockProcess	需要过程锁独占窗体	一旦发现被他人占用将提示 ProcessLockException 信息
ReportType	输出报表的类型	可绑定继承自 DevExpress.XtraReports.UI.XtraReport 的报表组件，实现自定义报表的功能
ParallelFetch	是否并行 Fetch	默认为阻塞主线程下并行 Fetch 多个数据集的方法，本功能仅适用于主数据集且其 IsAsynchronousFetch=false

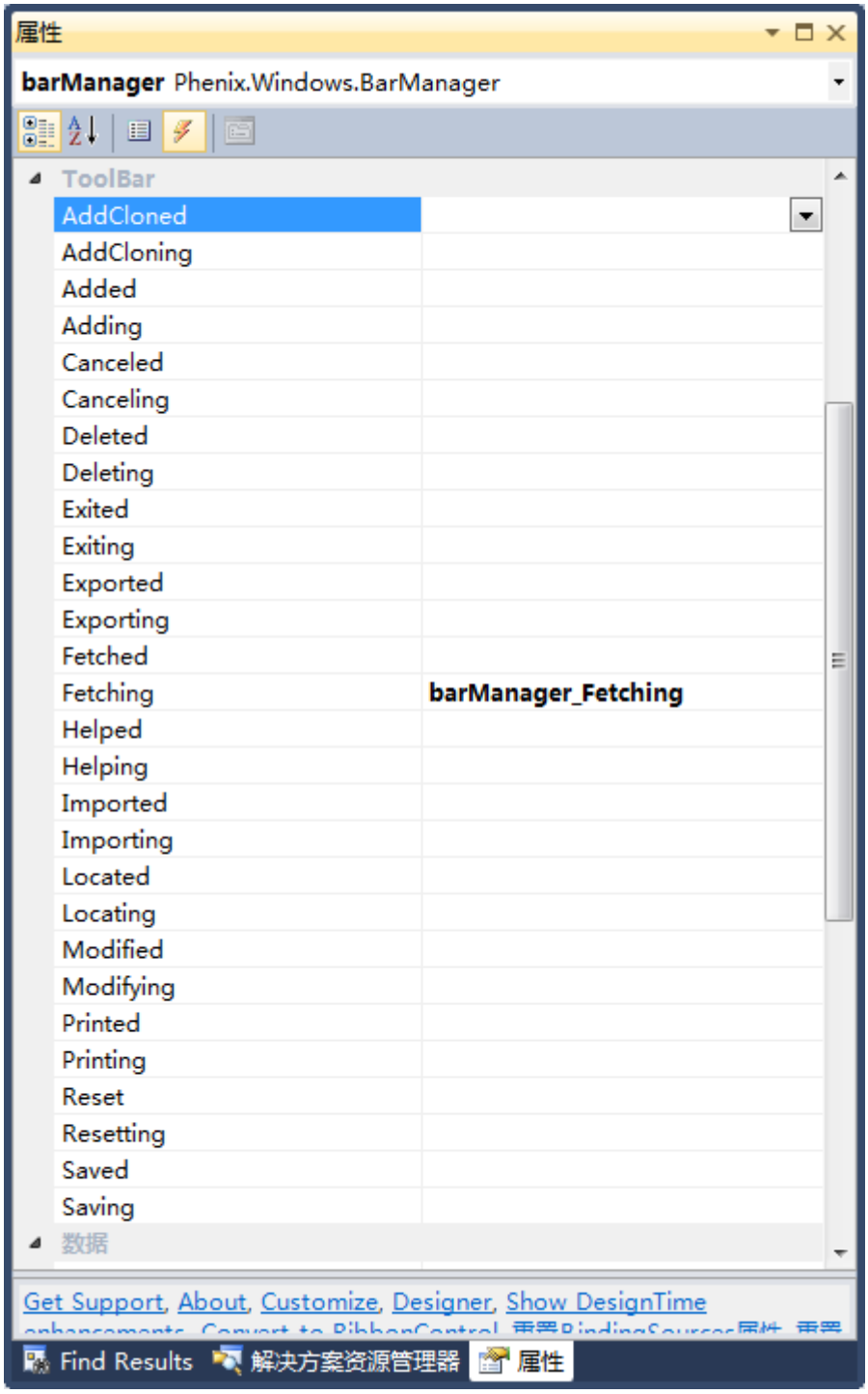
下列为辅助操作数据集的属性：

属性	说明	备注
CurrentBusinessList	当前业务对象集合	BindingSource.List as IBusinessCollection
CurrentBusiness	当前业务对象	BindingSource.Current as IBusinessObject
CurrentBusinessRoot	当前业务根对象	(MasterBindingSource ?? BindingSource)[.List/.Current] as IBusiness
EditMode	在编辑状态	OperateState == DataOperateState.Add || OperateState == DataOperateState.Modify || OperateState == DataOperateState.Delete
EditDirty	更改过数据	CurrentBusinessRoot.IsDirty

10.3 事件

10.3.1ToolBar 事件

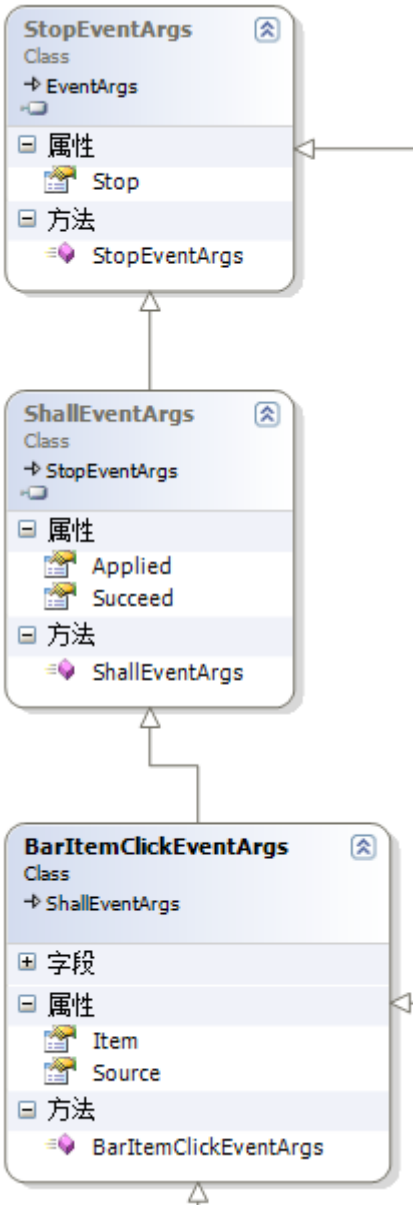
BarManager 组件 Tools Bar 上的功能按钮除了 Help 外都有缺省的功能实现，且自动处理了这些功能按钮的界面逻辑关系，所以不允许直接响应这些功能按钮的事件，而应该响应以下 BarManager 组件提供的事件，通过这些事件来干预组件提供的缺省执行：



这些事件按“事前/事后”与功能按钮的执行过程一一对应匹配。

10.3.1.1 一般干预方法

事件所传递的参数 BarItemClickEventArgs 都继承自 Phenix.Core.ShallEventArgs，因此可以通过参数传递的属性来干预各功能按钮的缺省执行：



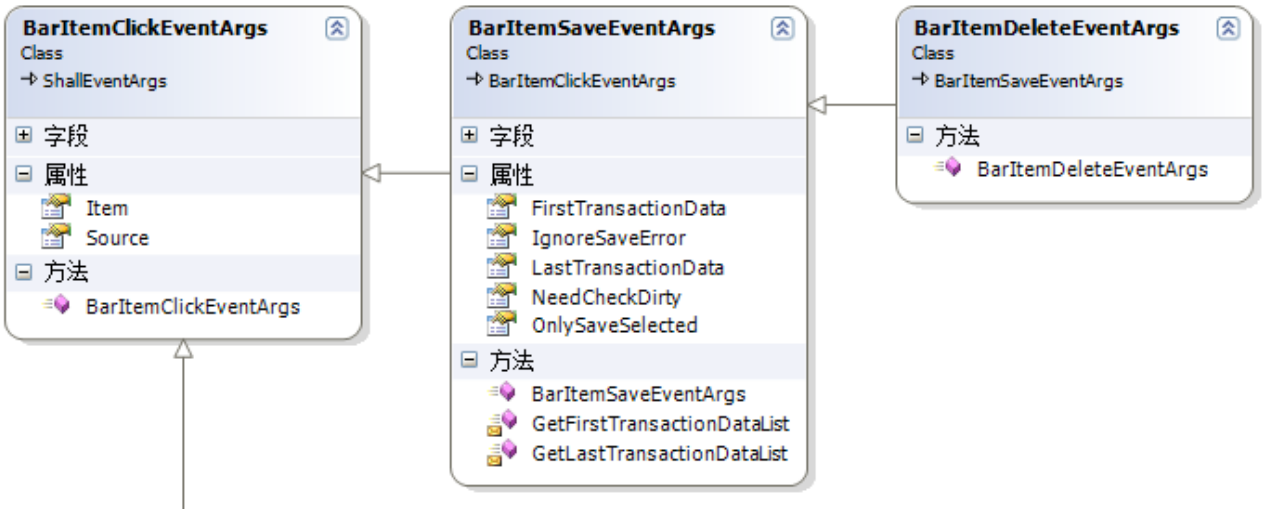
属性	说明	备注
Stop	是否终止	缺省为 false；依此判断是否需要后续的处理过程；
Applied	是否已应用	缺省为 false；依此判断是否需要实施常规的处理过程；
Succeed	是否已成功	缺省为 true；依此判断是否需要更改相应的成功标志等；
Item	BarItem	
Source	BindingSource	

以 BarManager 组件处理“退出”功能按钮点击事件为例，可以更加清晰地理解这些属性的作用：

```
/// <summary>
/// 点击退出按钮
/// </summary>
/// <param name="e">点击按钮事件数据</param>
public BarItemClickEventArgs ClickExitButton(ItemClickEventArgs e)
{
  ControlHelper.ResetFocus(Form);
  BarItemClickEventArgs args = new BarItemClickEventArgs(e != null ? e.Item : null, BindingSource);
  if (Exiting != null) ///触发“事前”事件
    Exiting(this, args);
  if (args.Stop) /// 如“终止”则直接返回
    return args;
  if (!args.Applied) /// 如未“应用”则执行缺省功能
  {
    Form form = Form as Form;
    if (form != null)
    {
      form.Close();
      args.Succeed = true; /// 申明已“成功”执行
    }
  }
  if (args.Succeed) /// 如已“成功”执行则触发“事前”事件
    if (Exited != null)
      Exited(this, args);
  return args;
}
```

10.3.1.2 BarItemSaveEventArgs、BarItemDeleteEventArgs

BarManager 组件在保存事件（Saving、Saved）、删除事件（Deleting、Deleted、DeleteCanceled）中提供了调用“提交数据”函数注入参数值的方法：



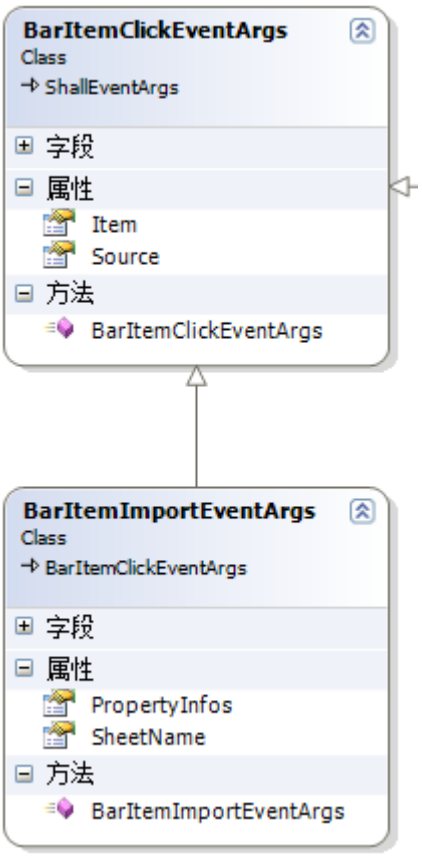
属性	说明	备注
NeedCheckDirty	校验数据库数据在下载到提交期间是否被更改过，一旦发现将报错：CheckDirtyException	缺省为 true；适用于操作 update 的乐观锁机制；
OnlySaveSelected	仅提交被选择（Selected = true）的业务对象	设置为 null 时，以 Source 上的 OnlySaveSelected 属性值为准；仅适用于业务集合对象的提交；
FirstTransactionData	与 Source 数据在一个事务里提交数据库，并提前于 Source 数据执行保存操作	本队列里的数据按队列依次提交；
LastTransactionData	与 Source 数据在一个事务里提交数据库，并滞后于 Source 数据执行保存操作	本队列里的数据按队列依次提交；

提交数据的方法是调用 BarManager 组件属性 CurrentBusinessRoot（当前业务根对象，类型 IBusiness）提供的接口函数：

```
    /// <summary>
    /// 保存
    /// </summary>
    /// <param name="needCheckDirty">校验数据库数据在下载到提交期间是否被更改过，一旦发现将报错：CheckDirtyException</param>
    /// <param name="onlySaveSelected">仅提交被选择的业务对象</param>
    /// <param name="firstTransactionData">参与事务处理前端的业务队列</param>
    /// <param name="lastTransactionData">参与事务处理末端的业务队列</param>
    /// <returns>成功提交的对象</returns>
    IBusiness Save(bool needCheckDirty, bool? onlySaveSelected, IBusiness[] firstTransactionData,
```

```
IBusiness[] lastTransactionData);
```

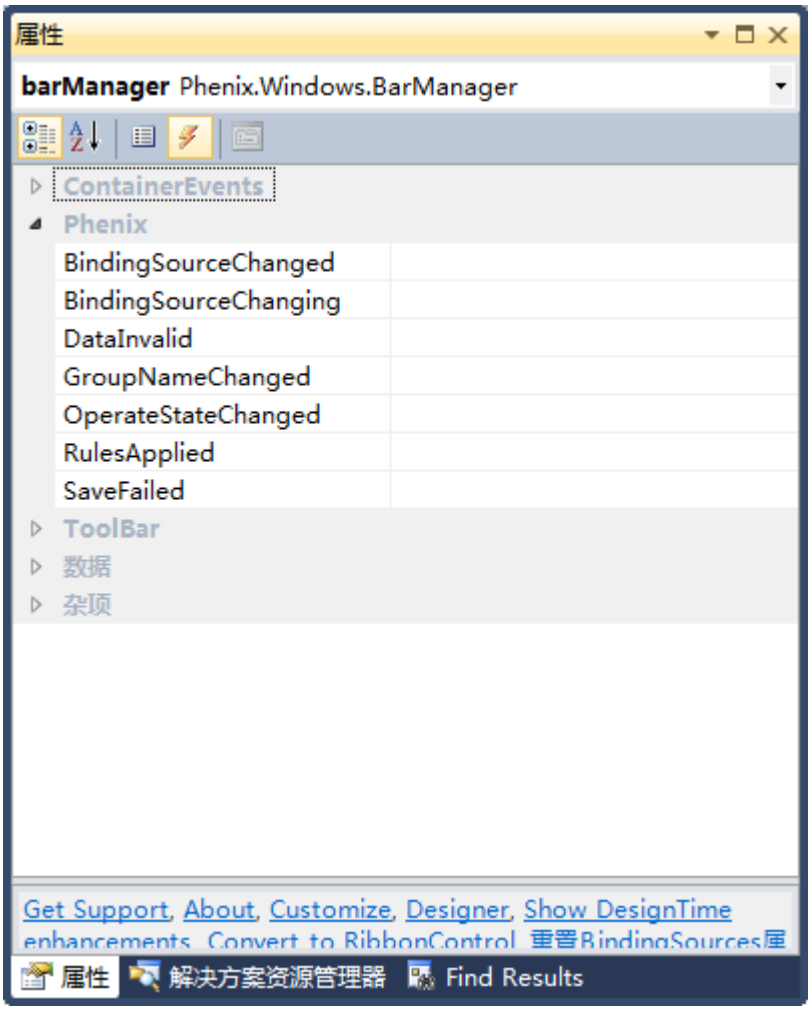
10.3.1.3 BarItemImportEventArgs



BarManager 组件在导入事件（Importing、Imported）中提供了调用“导入 Execl 表单数据”函数注入参数值的方法：

属性	说明	备注
SheetName	表单名	如为空则取 Execl 表单第一个 Sheet 内数据；
PropertyInfos	属性信息队列，顺序与表单 columnIndex 一致	如为空则按 Execl 表单列名与业务类属性名匹配条件进行数据填充；

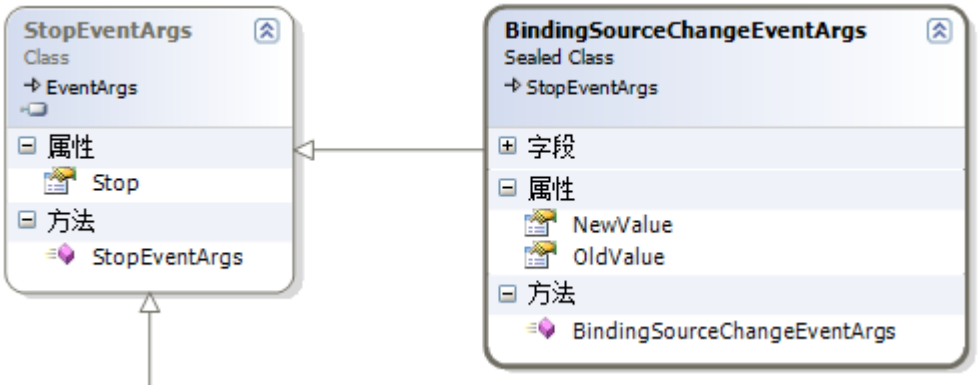
10.3.2其他事件



事件	说明	参数
BindingSourceChanging	属性 BindingSource 值发生更改时	BindingSourceChangeEventArgs
BindingSourceChanged	属性 BindingSource 值发生更改后	BindingSourceChangeEventArgs
DataInvalid	当数据源数据发生错误之后触发	DataInvalidEventArgs
GroupNameChanged	当属性 GroupName 值更改之后触发	EventArgs
OperateStateChanged	当属性 OperateState 值更改之后触发	EventArgs
RulesApplied	当规则被应用之后触发	EventArgs
SaveFailed	当保存数据源数据失败之后触发	ExceptionEventArgs

10.3.2.1 属性 BindingSource 值发生变更

属性 BindingSource 值会随着窗体上焦点在不同数据源的数据控件之间切换而变更，也可以直接被赋值，这时事件 BindingSourceChanging、BindingSourceChanged 都会被触发：



事件传递的参数 BindingSourceChangeEventArgs 继承自 Phenix.Core.ShallEventArgs，因此可以通过参数传递的属性来干预对属性 BindingSource 的赋值。

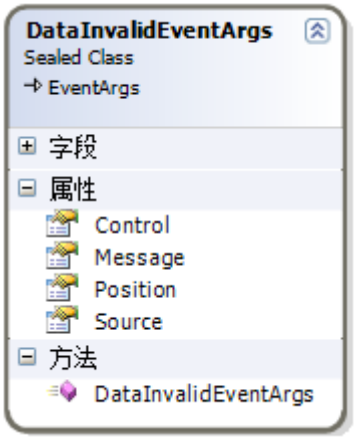
除此之外，可以通过此参数的两个属性（OldValue、NewValue）知道切换前后的 BindingSource 旧值和新值。

10.3.2.2 数据源数据发生错误

当窗体上的数据控件有录入动作，或者直接调用 BarManager 组件的下列函数，而验证出数据是失效的时候，都会触发 DataInvalid 事件：

```
/// <summary>
/// 校验控件值是否有效
/// </summary>
/// <param name="control">控件</param>
/// <returns>有效</returns>
public bool IsValid(Control control)

/// <summary>
/// 校验当前项是否有效
/// </summary>
/// <param name="source">数据源</param>
/// <param name="onlyOldError">仅检查原有错误</param>
/// <returns>有效</returns>
public bool IsValidCurrentItem(BindingSource source, bool onlyOldError)

/// <summary>
/// 定位无效项
/// </summary>
/// <param name="source">数据源</param>
/// <param name="onlyOldError">仅检查原有错误</param>
/// <returns>存在无效项</returns>
public bool LocateInvalidItem(BindingSource source, bool onlyOldError)
```

事件传递的参数 Phenix.Core.Windows.DataInvalidEventArgs：



我们可以通过事件传递的参数属性值，知道具体是哪个数据源（Source）的游标（Position）记录存在无效数据，且可以定位在哪个数据控件（Control）上，以及系统会抛出怎样的提示信息（Message）。

由于 Phenix丶在数据验证的处理上做到了最大可能的友好性，所以应用系统开发当中一般无需响应本事件，除非需要更加个性化的界面效果。

10.3.2.3 属性 GroupName 值发生变更

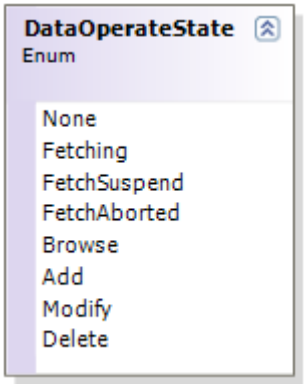
属性 GroupName 是用于当前窗体逻辑较为复杂，会处理到多组数据集且需要区分它们的场景。

属性 GroupName 值决定了 Fetch、Reset、Edit、Cancel、Save 功能需操作到哪些数据集。当本属性值非空，且在 BindingSources 清单中 BindingSourceStatus.GroupName 值与本属性值一致的 BindingSourceStatus.BindingSource，都属于被操作的范围。

属性 BindingSource 值发生变更时，BarManager 组件会根据其在 BindingSources 清单中 BindingSourceStatus.GroupName 值来更新属性 GroupName 值。属性 GroupName 值发生变更后，就会触发 GroupNameChanged 事件。

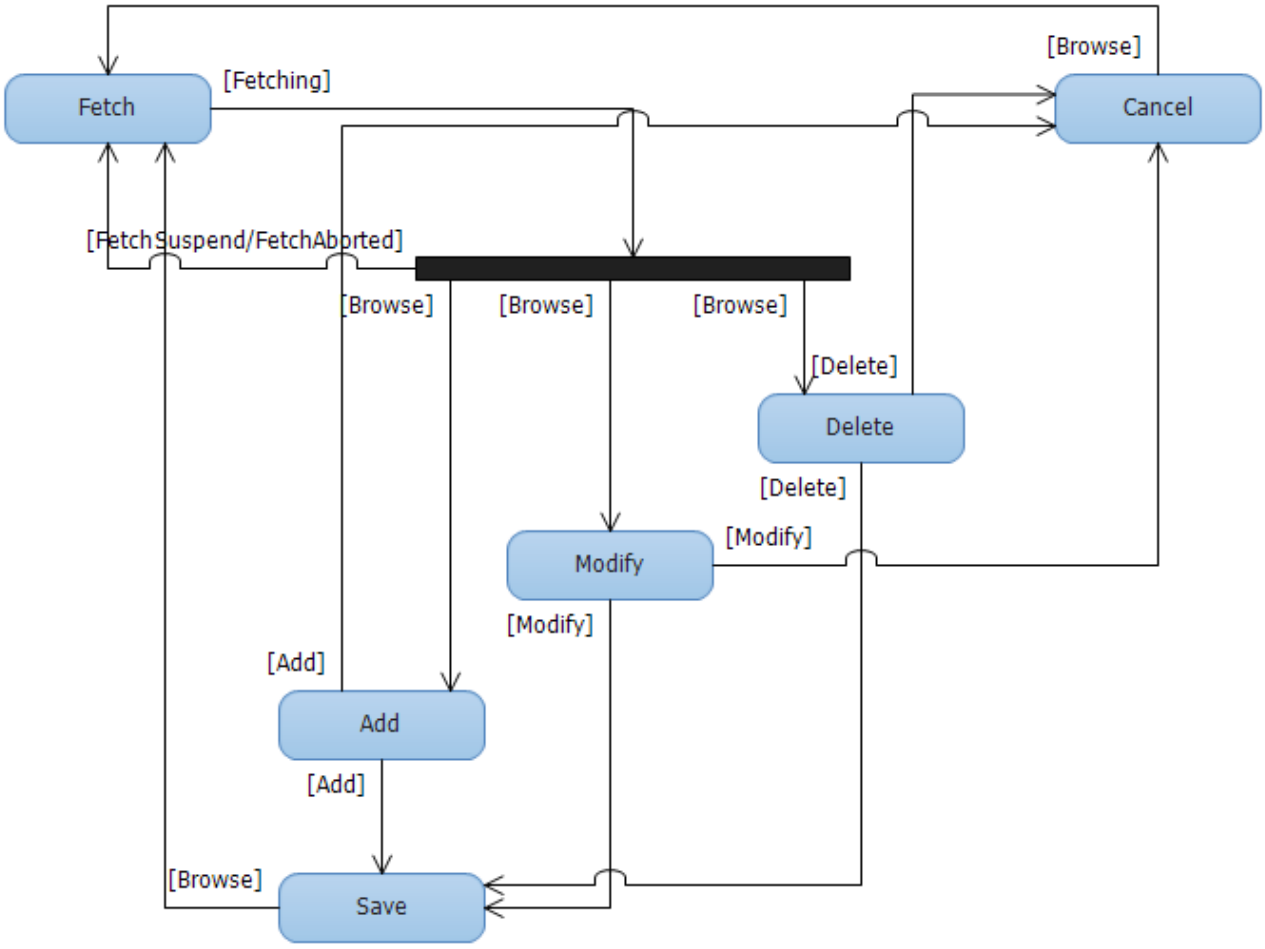
10.3.2.4 属性 OperateState 值发生变更

属性 OperateState 动态记录了当前数据集的操作状态，是个枚举：



属性 OperateState 值随着 BarManager 组件 Tools Bar 上的“增删改查”功能按钮被触发而变更，此时就会触发 OperateStateChanged 事件。

“增删改查”功能按钮与属性 OperateState 枚举值的互动关系如下：



BarManager 组件还提供了用编码方式触发功能按钮的函数，以达到与界面操作一样的效果：

```
/// <summary>
/// 点击检索按钮
/// </summary>
public BarItemClickEventArgs ClickFetchButton()

/// <summary>
/// 点击复位按钮
/// </summary>
public BarItemClickEventArgs ClickResetButton()

/// <summary>
```

```
    /// 点击增加按钮
    /// </summary>
    public BarItemClickEventArgs ClickAddButton()

    /// <summary>
    /// 点击克隆按钮
    /// </summary>
    public BarItemClickEventArgs ClickAddCloneButton()

    /// <summary>
    /// 点击编辑按钮
    /// </summary>
    public BarItemClickEventArgs ClickModifyButton()

    /// <summary>
    /// 点击删除按钮
    /// </summary>
    public BarItemDeleteEventArgs ClickDeleteButton()

    /// <summary>
    /// 点击定位按钮
    /// </summary>
    public BarItemClickEventArgs ClickLocateButton()

    /// <summary>
    /// 点击取消按钮
    /// </summary>
    public BarItemClickEventArgs ClickCancelButton()

    /// <summary>
    /// 点击保存按钮
    /// </summary>
    public BarItemSaveEventArgs ClickSaveButton()

    /// <summary>
    /// 点击导入按钮
    /// </summary>
    public BarItemImportEventArgs ClickImportButton()

    /// <summary>
    /// 点击导出按钮
    /// </summary>
```

```
public BarItemClickEventArgs ClickExportButton()

/// <summary>
/// 点击打印按钮
/// </summary>
public BarItemClickEventArgs ClickPrintButton()

/// <summary>
/// 点击帮助按钮
/// </summary>
public BarItemClickEventArgs ClickHelpButton()

/// <summary>
/// 点击退出按钮
/// </summary>
public BarItemClickEventArgs ClickExitButton()
```

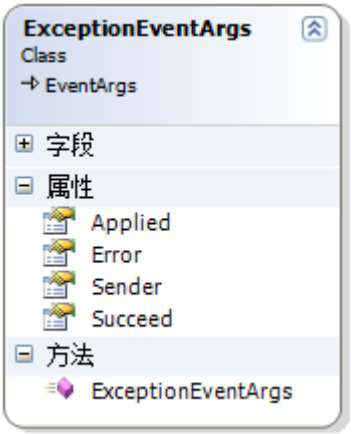
使用这些函数，需自行判断当前的界面逻辑，BarManager 组件是不会做验证的。

10.3.2.5 RulesApplied 事件

当前数据集的操作状态决定了界面逻辑，也就是 Tools Bar 上功能按钮的灰亮，每次刷新之后，都会触发 RulesApplied 事件。利用这个事件，我们可以实现更加个性化的界面效果。

10.3.2.6 SaveFailed 事件

SaveFailed 事件传递的参数是 Phenix.Core.ExceptionEventArgs：



我们可以通过事件传递的参数属性值，知道具体是哪个根业务对象（Sender）抛出的异常（Error）。如果在响应事件代码中将参数 e.Appliced = true，可以跳过 BarManager 组件提供的缺省异常处理方法，但仍维持编辑状态；如果在响应事件代码中将参数 e.Succeed = true，则代表你已经处理了异常且自行

提交了数据，组件将恢复到浏览状态。

下列代码是组件中提交数据的核心代码，可理解如何通过 SaveFailed 事件来干预数据提交失败时的处理过程：

```
private bool DoSave(bool needPrompt, bool needCheckDirty, bool? onlySaveSelected, IBusiness[]
firstTransactionData, IBusiness[] lastTransactionData)
    {
        while (true)
          try
          {
            using (DevExpress.Utils.WaitDialogForm waitDialog =
              new DevExpress.Utils.WaitDialogForm(String.Format(Properties.Resources.DataSaving,
CurrentBusinessRoot.Caption,
                AppConfig.Debugging || UserPrincipal.User == null || UserPrincipal.User.IsAdminRole ?
CurrentBusinessRoot.GetType().Name : null),
                Phenix.Core.Properties.Resources.PleaseWait))
            {
              try
              {
                CurrentBusinessRoot.Save(needCheckDirty, onlySaveSelected, firstTransactionData,
lastTransactionData);
              }
              catch (Exception ex)
              {
                ExceptionEventArgs e = new ExceptionEventArgs(CurrentBusinessRoot, ex);
                OnSaveFailed(e);
                if (!e.Succeed)
                  if (e.Applied)
                    return false;
                  else
                    throw;
              }
            }
            DoSaved();
            string hint = String.Format(Properties.Resources.DataSaveSucceed,
CurrentBusinessRoot.Caption);
            ShowHint(hint);
            if (needPrompt)
              MessageBox.Show(hint, Properties.Resources.DataSave, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            return true;
          }
          catch (CheckDirtyException ex)
          {
```

```
            string hint = String.Format(Properties.Resources.DataSaveForcibly,
              CurrentBusinessRoot.Caption,
              AppConfig.Debugging || UserPrincipal.User == null || UserPrincipal.User.IsAdminRole ?
CurrentBusinessRoot.GetType().FullName : null,
              AppUtilities.GetErrorHint(ex));
            if (MessageBox.Show(hint, Properties.Resources.DataSave, MessageBoxButtons.YesNo,
MessageBoxIcon.Question) != DialogResult.Yes)
              return false;
            needCheckDirty = false;
            continue;
          }
          catch (Exception ex)
          {
            string hint = String.Format(Properties.Resources.DataSaveAborted,
              CurrentBusinessRoot.Caption,
              AppConfig.Debugging || UserPrincipal.User == null || UserPrincipal.User.IsAdminRole ?
CurrentBusinessRoot.GetType().FullName : null,
              AppUtilities.GetErrorHint(ex, typeof(Csla.DataPortalException),
typeof(Csla.Reflection.CallMethodException)));
            ShowHint(hint);
            MessageBox.Show(hint, Properties.Resources.DataSave, MessageBoxButtons.OK,
MessageBoxIcon.Error);
            LocateInvalidItem(MasterBindingSource ?? BindingSource, false);
            return false;
          }
    }
```

10.4 其他有用的公共函数

前文已罗列了组件中部分有用的公共函数，除此之外，下列这些公共函数在设计复杂的界面逻辑时也可以派上用场。

10.4.1重置读写授权

```
    /// <summary>
    /// 重置读写授权
    /// </summary>
    /// <param name="readOnly">只读</param>
    /// <param name="sources">数据源队列</param>
    public void ResetControlAuthorizationRules(bool readOnly, params BindingSource[] sources)
```

10.4.2判断组件是否授权

```
/// <summary>
/// 是否拒绝执行
/// </summary>
public bool DenyExecute(Component component)
```

10.4.3重置功能按钮

```
/// <summary>
/// 应用ToolsBar规则
/// </summary>
public void ApplyToolsBarRules()
```

10.4.4重置状态栏

```
/// <summary>
/// 重置状态栏
/// </summary>
public void ResetRecordState()
```

10.4.5显示提示信息

```
/// <summary>
/// 显示提示信息
/// </summary>
/// <param name="text">文本</param>
public void InvokeShowHint(string text)
```