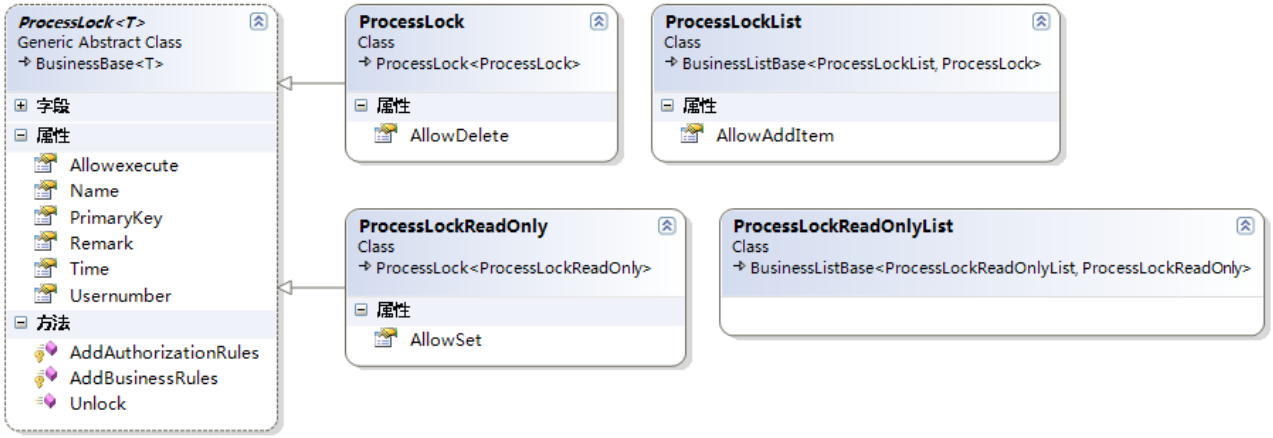
12 业务结构对象模型

12.3 业务结构和方法

Phenix丶也是采取类似CSLA的树状业务类的结构，并对这个业务结构的整个生命周期进行管控，但实现方法区别于CSLA：比如未从物理（而是从逻辑）代码上区分只读对象和可编辑对象，目的是最大限度地精简代码，并能共享相同的业务结构和方法；又比如提供给开发人员更加丰富的搭建业务结构的手段和方法，以最大可能满足现实中应用场景的开发需求。

12.3.1业务类家族

通过Phenix丶Addin工具的“初始化/编辑业务类”功能，我们可以自动构建出虚拟业务类及其可编辑业务类和只读业务类，包括它们的集合类。



因为有了类的继承性，我们可以创建出类似上述案例中这样一个业务类的家族。

在这个业务类家族（“树”）里，只有“叶端”的业务类，才在应用场景中使用，而这棵“树”的“根”类、“节点”类，都应该设计成虚拟的泛型类。在这些泛型类中，为叶端的业务类提供基本的业务逻辑功能。

12.3.2只读的业务对象

Phenix丶未封装 CSLA 的 ReadOnlyBase、ReadOnlyListBase 基类，而代之以在业务类上打Phenix.Core.Mapping. ReadOnlyAttribute 标签的方法实现相类似的功能。

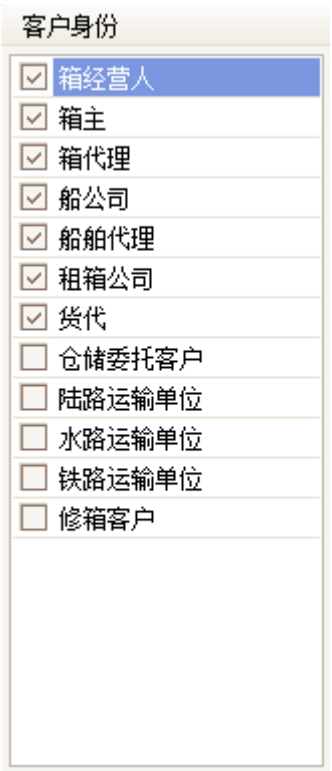
```
[System.Serializable]
[Phenix.Core.Mapping.ReadOnly]
public class UserReadOnly : User<UserReadOnly>
{
  private UserReadOnly()
  {
    //禁止添加代码
```

```
    }
  }

  /// <summary>
  /// 清单
  /// </summary>
  [System.Serializable]
  public class UserReadOnlyList : Phenix.Business.BusinessListBase<UserReadOnlyList, UserReadOnly>
  {
    private UserReadOnlyList()
    {
      //禁止添加代码
    }
  }
```

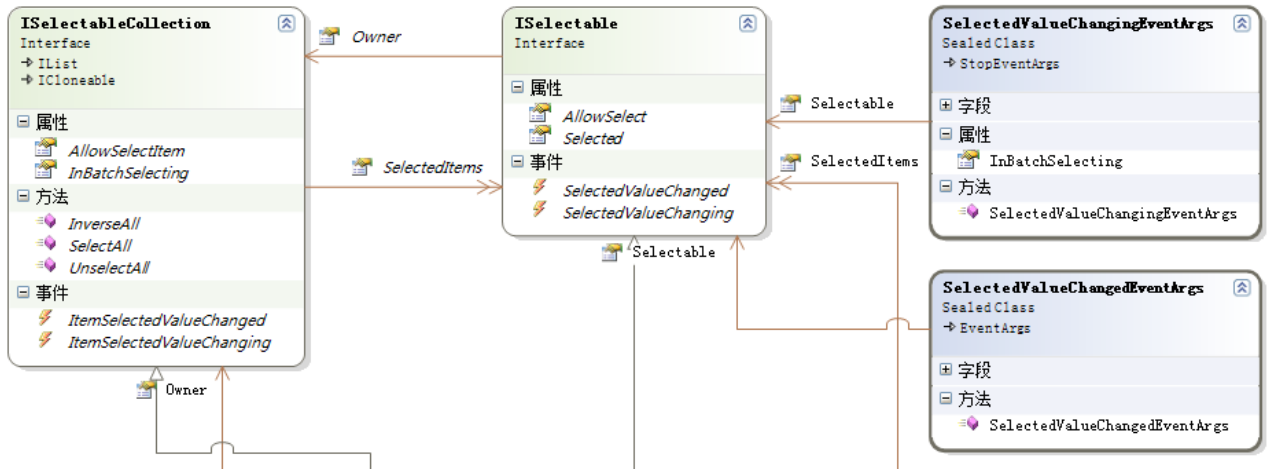
不过，虽然禁止了直接增删改业务对象、（通过 set{}方式）操作属性值，但并不影响我们通过其他方式间接操作业务对象的字段值，进而将这些更改提交到数据库中。

12.3.3可被勾选的业务对象

许多业务场景中，交互界面上需要提供对业务清单进行勾选的功能，然后对这些被勾选的业务对象做进一步操作：



Phenix丫为此提供了业务对象的可被勾选能力：



Phenix.Business.BusinessListBase<T, TBusiness>实现了 ISelectableCollection 接口，而 Phenix.Business.BusinessBase<T>则实现了 ISelectable 接口。

下面分别罗列这些接口：

12.3.3.1 Phenix.Business.BusinessListBase<T, TBusiness>提供的应用接口

属性	说明	备注
SelectedItems	被勾选的对象队列	当业务对象的 Selected 属性值发生改变时自动维护本队列；
AllowSelectItem	是否允许被勾选	

事件	说明	备注
ItemSelectedValueChanging	Selected 属性被更改前	
ItemSelectedValueChanged	Selected 属性被更改后	

并提供了方便勾选的方法：

```
/// <summary>
/// 勾选所有
/// </summary>
void SelectAll();

/// <summary>
/// 取消所有勾选
/// </summary>
void UnselectAll();
```

```
/// <summary>
/// 反选所有
/// </summary>
void InverseAll();
```

12.3.3.2 Phenix.Business.BusinessBase<T>提供的应用接口

属性	说明	备注
Owner	所属对象集合	
Selected	是否被勾选	缺省为 false；用于标记本对象；
AllowSelect	是否允许被勾选	

事件	说明	备注
SelectedValueChanging	Selected 属性值被更改前	
SelectedValueChanged	Selected 属性值被更改后	

12.3.4以 Root 业务对象搭建的业务结构

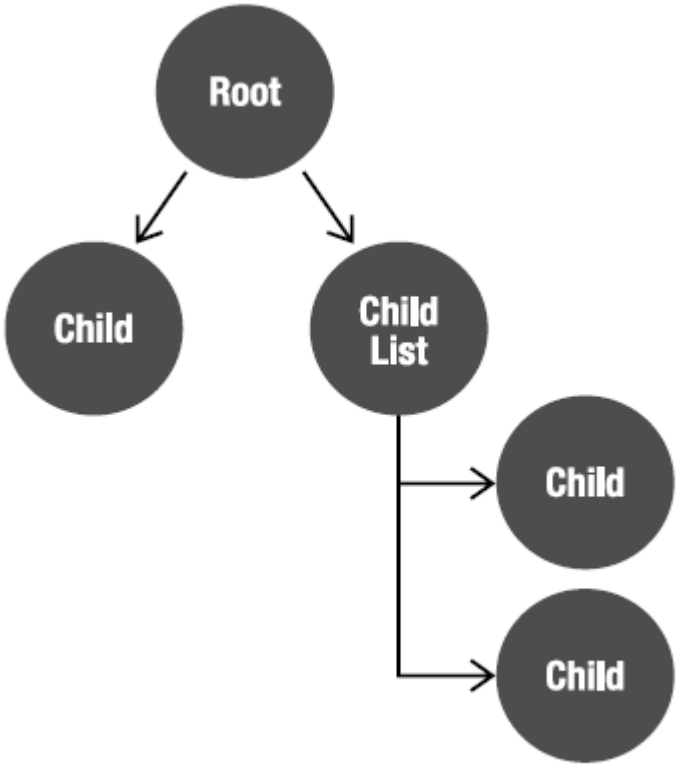
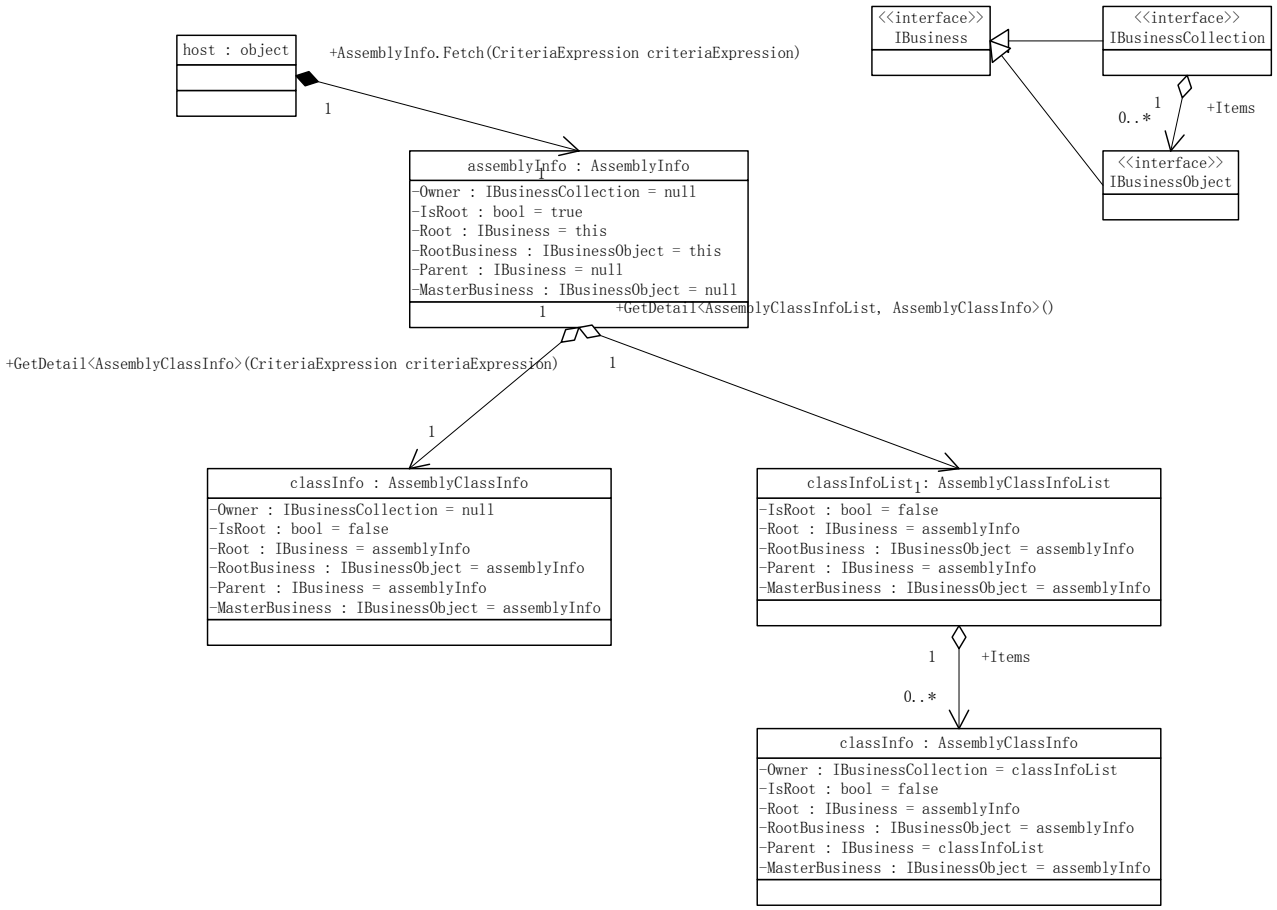


Figure 4-2. *Object graph illustrating containment*

上图摘自 CSLA 作者的《Expert C# 2008 Business Objects》。

Phenixヾ实现的方法如下图所示：



12.3.5以 Root 业务集合对象搭建的业务结构

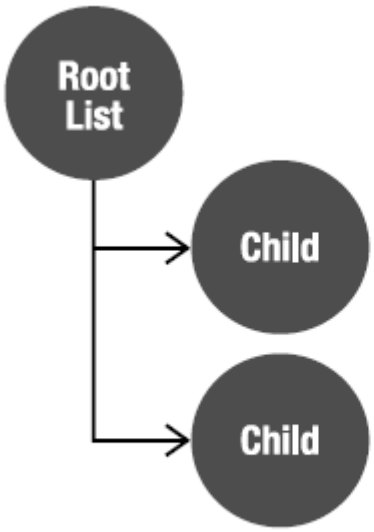
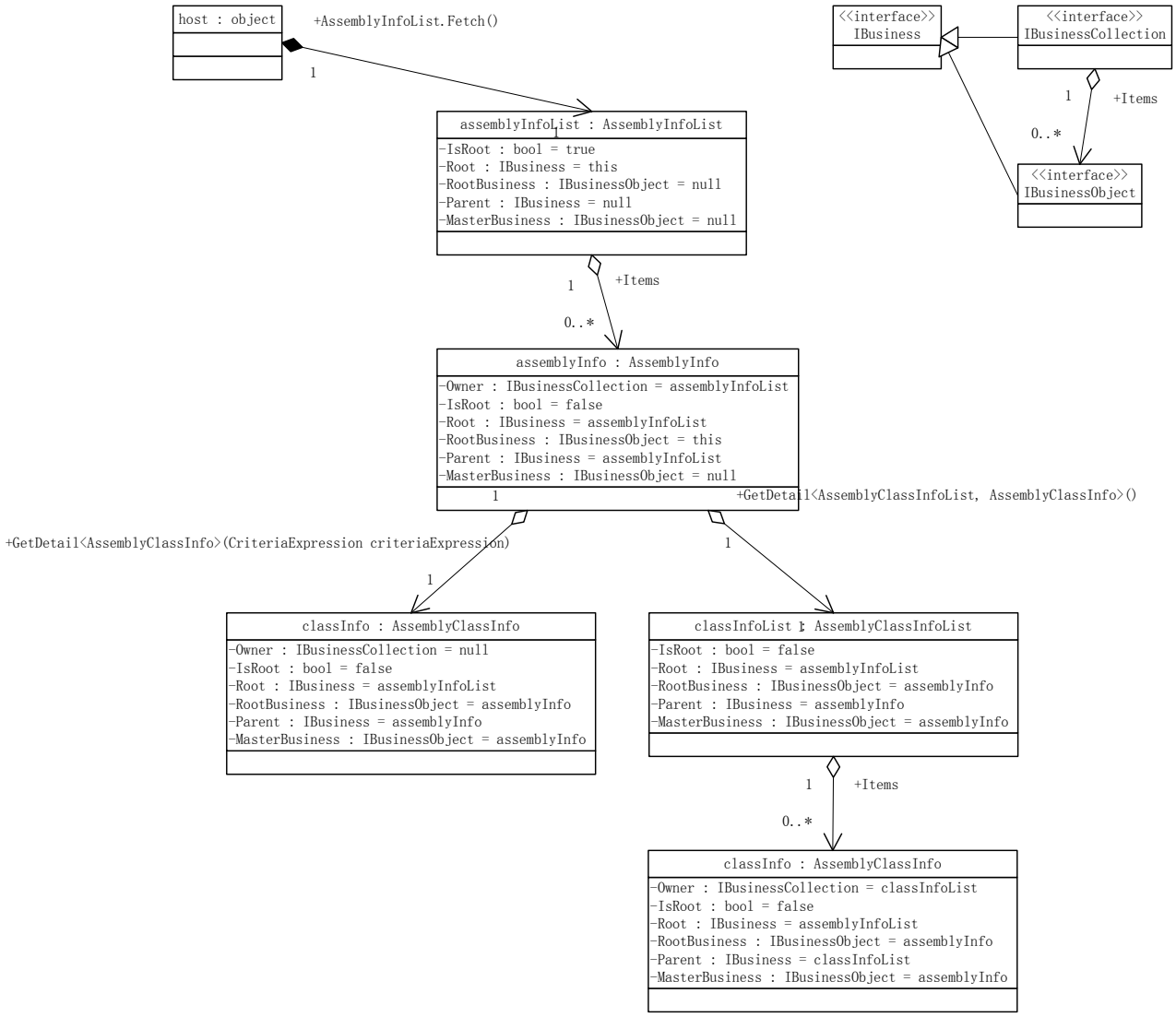


Figure 4-3. *Object graph with a root list object*

上图摘自 CSLA 作者的《Expert C# 2008 Business Objects》。

Phenixヾ实现的方法如下图所示：



12.3.6业务结构的组合关系和聚合关系

以上两种业务结构，虽 Root 入口对象不一样，但在业务类的 GetDetail 定义上其实是没有区别的。代码参考如下：

```
/// <summary>
/// 类信息
/// </summary>
[Serializable]
public class AssemblyClass : AssemblyClass<AssemblyClass>
{

  /// <summary>
  /// 类属性
```

```
    /// </summary>
    public AssemblyClassPropertyList AssemblyClassProperties
    {
      get { return GetCompositionDetail<AssemblyClassPropertyList, AssemblyClassProperty>(); }
    }

    /// <summary>
    /// 类方法
    /// </summary>
    public AssemblyClassMethodList AssemblyClassMethods
    {
      get { return GetCompositionDetail<AssemblyClassMethodList, AssemblyClassMethod>(); }
     }
   }
```

示例中的 GetCompositionDetail()函数，强调了属于组合关系的主从结构。组合关系也可以使用 GetDetail()函数来隐式定义，效果是一样的。

与 GetCompositionDetail()函数对应的是 GetAggregationDetail()函数，强调了聚合关系的主从结构。聚合关系对应到表结构设计，一般做法是松关联，即去掉了物理外键关系（本文定义为“虚”外键字段）。此时，应该在从业务类的“虚”外键字段上显式申明 FieldLinkAttribute 标签。聚合关系不同于组合关系的级联删除效果，Phenix丶只负责自动实现 Unlink 效果，也就是将从业务类的“虚”外键字段值置为 null。

需注意的是，虽然申明了 GetDetail 定义，但在运行中如果没有调用到它们的话，Phenix丶是不清楚有这层组合或聚合关系的。如果希望自动删除（级联删除/Unlink）子表的记录，应该显式声明 ClassDetailAttribute 标签。

12.3.7无表结构关系的主从业务结构

在业务逻辑上两个业务类存在着主从关系，但它们在表结构上却没有一点关系（有时候是故意去掉了物理外键关系，采取松关联的设计方法），不能通过上述的 GetDetail()函数直接获取到从业务对象，此时可参考下述代码进行设计：

```
        public PortContainerRangeRuleViewList PortContainerRangeRuleViews
        {
            get
            {
                var result = FindDetail(typeof(PortContainerRangeRuleViewList).FullName);
                if (result == null)
                {
                   result = PortContainerRangeRuleViewList.Fetch(p => p.CPC_CTN_OPERATOR_CTI_ID ==
```

```
this.CPC_CTN_OPERATOR_CTI_ID && p.Ctnno == this.Ctnno && p.CtnnoArrayLast == this.CtnnoArrayLast &&
p.ContainerSize == this.ContainerSize && p.ContainerType == this.ContainerType && p.EffectiveDate ==
this.EffectiveDate && p.ExpiryDate == this.ExpiryDate);
                    SetAggregationDetail(typeof(PortContainerRangeRuleViewList).FullName, result);
                }
                return result as PortContainerRangeRuleViewList;
            }
        }
```

此案例中 PortContainerRangeRuleViews 属性没有自己的字段，是由 Fetch 到的从业务集合对象(通过 SetDetail()函数）交给主业务对象的缓存池统一管理。get 代码里，首先要到缓存中（通过 FindDetail()函数）查找，找不到才去 Fetch。

此案例要求始终维持一个从业务集合对象，不管查询条件如何变化，所以操作缓冲池（FindDetail()函数、SetDetail()函数）的时候必须提供相同的 key 值（推荐使用从业务类的类名全称）。

通过 SetDetail()函数，可以将 Fetch 到的从业务对象统一存放在从业务对象缓存池中。这样，当主业务对象处理 BeginEdit、Save 等操作的时候，这些从业务对象都会跟着一起联动，并且在界面绑定、数据交互上，也和普通从业务对象的处理无任何区别。

12.3.8从业务对象缓冲池的使用

上述案例中，用到了以下两个 Phenix.Business.BusinessBase<T>的函数：

```
/// <summary>
/// 检索从业务对象
/// </summary>
/// <param name="key">比对键值</param>
public IBusiness FindDetail(string key)

/// <summary>
/// 设置从业务对象(组合关系)
/// </summary>
/// <param name="key">比对键值</param>
/// <param name="detail">从业务对象</param>
public void SetCompositionDetail(string key, IBusiness detail)

/// <summary>
/// 设置从业务对象(聚合关系)
/// </summary>
/// <param name="key">比对键值</param>
/// <param name="detail">从业务对象</param>
public void SetAggregationDetail(string key, IBusiness detail)
```

这两个函数的配合使用，除了适用于上述案例外，也可以用于不同查询条件下仅需维持一个缓存子（集合）对象的应用场景：

```
    /// <summary>
    /// 费用计算主表
    /// </summary>
    [Serializable]
    public class AccountBook : AccountBook<AccountBook>
    {
      。。。


      /// <summary>
      /// 费用收入明细
      /// </summary>
      public virtual AccountbookDetailList AccountBookIncomeDetails
      {
        get
        {
          CriteriaExpression expression =
            AccountbookDetail.ACD_ACB_IDProperty == AccountBook.ACB_IDProperty &
            AccountbookDetail.IncomeTypeProperty == IncomeType.Income;
          if (this.DetailBusinessIds != null && this.DetailBusinessIds.Count > 0)
            expression = expression &
AccountbookDetail.BusinessidProperty.In(this.DetailBusinessIds.ToArray());

          AccountbookDetailList result = FindDetail("AccountBookIncomeDetails") as
AccountbookDetailList;
          if (result == null ||
            result.Criterions.CriteriaExpression != expression)
          {
            result = AccountbookDetailList.Fetch(expression);
            SetCompositionDetail("AccountBookIncomeDetails", result);
          }
          return result;
        }
      }
    }
```

如果需要，不同的查询条件可以对应各自的一个缓存子（集合）对象：

```
      /// <summary>
      /// 费用收入明细
```

```
        /// </summary>
        public virtual AccountbookDetailList AccountBookIncomeDetails
        {
            get
            {
                CriteriaExpression expression = AccountbookDetail.IncomeTypeProperty ==
IncomeType.Income;
                if (this.DetailBusinessIds != null && this.DetailBusinessIds.Count > 0)
                {
                    expression = expression &
AccountbookDetail.BusinessidProperty.In(this.DetailBusinessIds.ToArray());
                }
                return GetCompositionDetail<AccountbookDetailList, AccountbookDetail>(expression);
            }
        }
```

另外，FindDetail()函数，还能用于在首次尝试获取子（集合）对象的过程中做一些特殊的处理，比如：

```
    /// <summary>
    /// 出仓拣货货物
    /// </summary>
    [Phenix.Core.Mapping.ClassAttribute("WGW_DELIVERY_PICKING_GOODS", FetchScript =
"WGW_DELIVERY_PICKING_GOODS_V", FriendlyName = "出仓拣货货物"), System.SerializableAttribute(),
System.ComponentModel.DisplayNameAttribute("出仓拣货货物")]
    public class DeliveryPickingGoods : DeliveryPickingGoods<DeliveryPickingGoods>
    {
        。。。


        /// <summary>
        /// 出仓拣货存储单元(数据变动时,DeliveryPickingGoods的实际件数,实际体积,实际重量和实际计量数
会随之累加)
        /// </summary>
        public DeliveryPickingLocationList DeliveryPickingLocations
        {
            get
            {
                var detail = FindAggregationDetail<DeliveryPickingLocationList,
DeliveryPickingLocation>();
                if (detail == null)
                {
                    detail = GetAggregationDetail<DeliveryPickingLocationList,
DeliveryPickingLocation>();
```

```
                    detail.ChildChanged += new
EventHandler<Csla.Core.ChildChangedEventArgs>(DoChildChanged);
                }
                return detail;
            }
        }

        private void DoChildChanged(object sender, Csla.Core.ChildChangedEventArgs e)
        {
            if (e.PropertyChangedArgs != null && e.ChildObject is DeliveryPickingLocation)
            {
                if (e.PropertyChangedArgs.PropertyName ==
DeliveryPickingLocation.ActualCountProperty.Name
                    || e.PropertyChangedArgs.PropertyName ==
DeliveryPickingLocation.ActualVolumeProperty.Name
                    || e.PropertyChangedArgs.PropertyName ==
DeliveryPickingLocation.ActualWeightProperty.Name
                    || e.PropertyChangedArgs.PropertyName ==
DeliveryPickingLocation.ActualGaugeCountProperty.Name)
                {
                    ActualCount = DeliveryPickingLocations.Sum(p => p.ActualCount);
                    ActualVolume = DeliveryPickingLocations.Sum(p => p.ActualVolume);
                    ActualWeight = DeliveryPickingLocations.Sum(p => p.ActualWeight);
                    ActualGaugeCount = DeliveryPickingLocations.Sum(p => p.ActualGaugeCount);
                }
            }
        }
    }
```

除了这两个函数，Phenix.Business.BusinessBase<T>还提供了其他方式，下文一一罗列。

12.3.8.1 检索从业务对象

```
    /// <summary>
    /// 检索从业务对象
    /// 从业务对象与本业务对象是一对一的关系
    /// </summary>
    /// <param name="criterions">条件集</param>
    public TDetailBusiness FindDetail<TDetailBusiness>(Criterions criterions)
       where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>

    /// <summary>
```

```
    /// 检索从业务对象(组合关系)
    /// 从业务对象与本业务对象是一对一的关系
    /// </summary>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>()
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// 从业务对象与本业务对象是一对一的关系
    /// </summary>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>()
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// </summary>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// </summary>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(ICriteria criteria)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(ICriteria criteria)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
```

```
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(ICriteria criteria, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(ICriteria criteria, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(CriteriaExpression
criteriaExpression)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(CriteriaExpression
criteriaExpression)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(CriteriaExpression
criteriaExpression, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(CriteriaExpression
```

```
criteriaExpression, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindCompositionDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetailBusiness FindAggregationDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合
    /// </summary>
    /// <param name="criterions">条件集</param>
    public TDetail FindDetail<TDetail, TDetailBusiness>(Criterions criterions)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
```

```
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>()
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>()
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="groupName">分组名</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="groupName">分组名</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(ICriteria criteria)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(ICriteria criteria)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
```

```
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(ICriteria criteria, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(ICriteria criteria, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(CriteriaExpression
criteriaExpression)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(CriteriaExpression
criteriaExpression)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(CriteriaExpression
criteriaExpression, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
```

```
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(CriteriaExpression
criteriaExpression, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindCompositionDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 检索从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    public TDetail FindAggregationDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

12.3.8.2 设置从业务对象

```
/// <summary>
/// 设置从业务对象(组合关系)
/// 从业务对象与本业务对象是一对一的关系
/// </summary>
/// <param name="detail">从业务对象</param>
public void SetCompositionDetail<TDetailBusiness>(TDetailBusiness detail)
  where TDetailBusiness : BusinessBase<TDetailBusiness>

/// <summary>
/// 设置从业务对象(聚合关系)
/// 从业务对象与本业务对象是一对一的关系
/// </summary>
/// <param name="detail">从业务对象</param>
public void SetAggregationDetail<TDetailBusiness>(TDetailBusiness detail)
  where TDetailBusiness : BusinessBase<TDetailBusiness>

/// <summary>
/// 设置从业务对象集合(组合关系)
/// </summary>
/// <param name="detail">从业务对象集合</param>
public void SetCompositionDetail<TDetail, TDetailBusiness>(TDetail detail)
  where TDetail : BusinessListBase<TDetail, TDetailBusiness>
  where TDetailBusiness : BusinessBase<TDetailBusiness>

/// <summary>
/// 设置从业务对象集合(聚合关系)
/// </summary>
/// <param name="detail">从业务对象集合</param>
public void SetAggregationDetail<TDetail, TDetailBusiness>(TDetail detail)
  where TDetail : BusinessListBase<TDetail, TDetailBusiness>
  where TDetailBusiness : BusinessBase<TDetailBusiness>
```

12.3.8.3 清除从业务对象的本地缓存

```
/// <summary>
/// 清除从业务对象Cache
/// </summary>
protected void ClearDetailCache()
```

12.3.9干预业务数据的获取

在构建上述业务结构时，都是通过 Feth 业务对象来实现的（见“11.业务对象生命周期及其状态”的“Fetch 业务对象”章节）。

Phenix.Business.BusinessBase<T>和 Phenix.Business.BusinessListBase<T, TBusiness>都提供了调用 Fetch()函数跨物理域时干预业务数据获取过程的 virtual 函数，以“On”为函数名的前缀，我们可按需在这些函数里嵌入自己的业务逻辑代码。

```
    /// <summary>
    /// 构建本业务对象之前
    /// 在运行持久层的程序域里被调用
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="command">DbCommand</param>
    /// <param name="criterions">条件集</param>
    protected virtual void OnFetchingSelf(DbConnection connection, DbCommand command, Criterions
criterions)

    /// <summary>
    /// 构建本业务对象之后
    /// 在运行持久层的程序域里被调用
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criterions">条件集</param>
    protected virtual void OnFetchedSelf(DbConnection connection, Criterions criterions)
```

或者在事务中：

```
    /// <summary>
    /// 构建本业务对象之前
    /// 在运行持久层的程序域里被调用
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="command">DbCommand</param>
    /// <param name="criterions">条件集</param>
    protected virtual void OnFetchingSelf(DbTransaction transaction, DbCommand command, Criterions
criterions)
    {
    }

    /// <summary>
    /// 构建本业务对象之后
    /// 在运行持久层的程序域里被调用
```

```
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criterions">条件集</param>
    protected virtual void OnFetchedSelf(DbTransaction transaction, Criterions criterions)
    {
    }
```

如果要在本地物理域中拦截 Fetch，可用下述的挂接事件和重载方法来实现：

```
    /// <summary>
    /// 构建业务对象前事件
    /// </summary>
    protected static event Action<object> Fetching;

   /// <summary>
    /// 构建本业务对象之后
    /// </summary>
    protected virtual void OnFetchedSelf(object criteria)
```

12.3.10 与 BindingSource 界面组件的联动

Phenix丶在 WinForm 界面的数据操作上，依靠 BindingSource 组件来协同界面的交互，实现对数据的增删改处理。

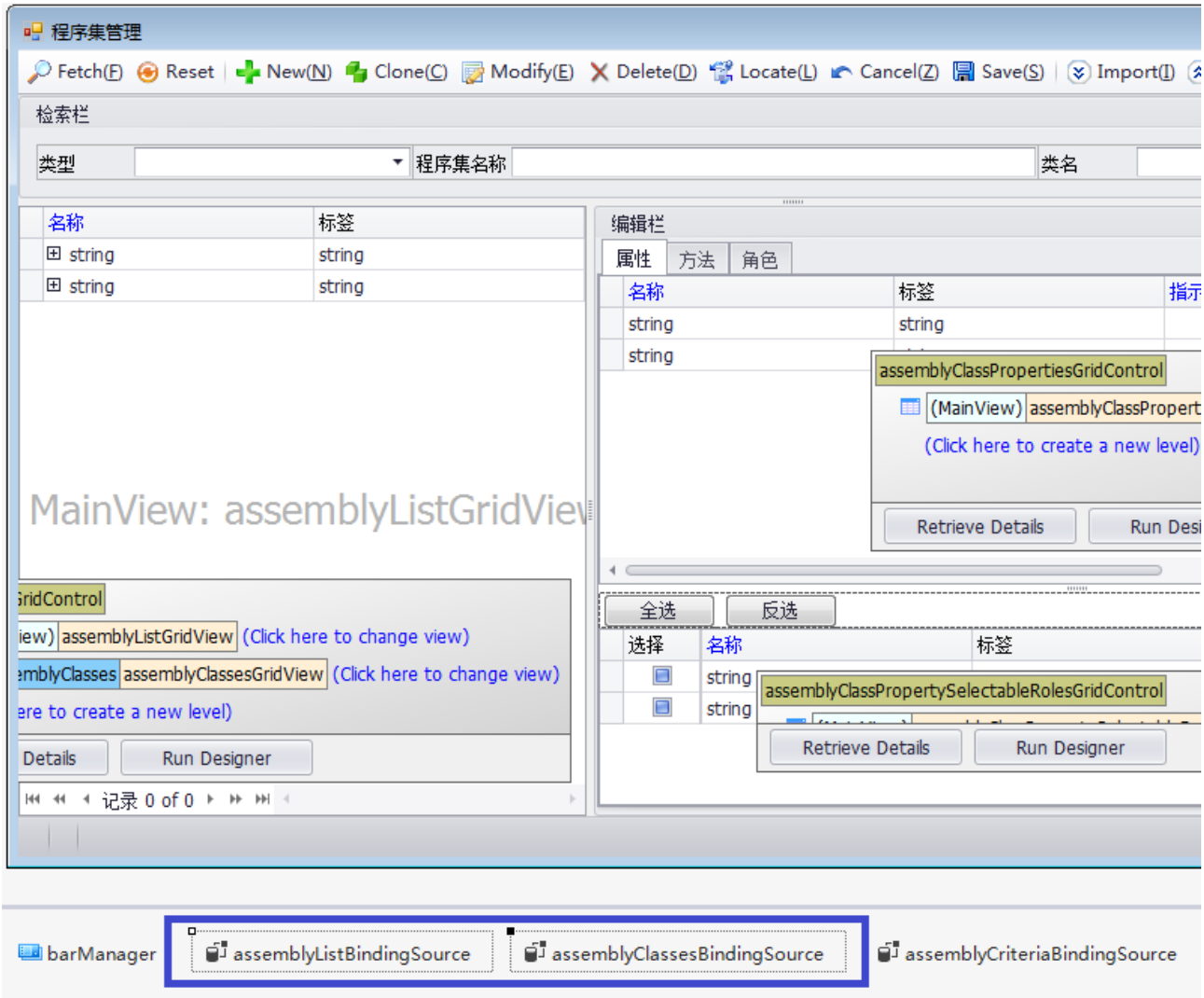
BindingSource 组件的使用方法，请参考《Windows Foms2.0 数据绑定—.NET 智能客户端数据应用设计》一书。重点学习第 4 章，深度学习在第 9 章。

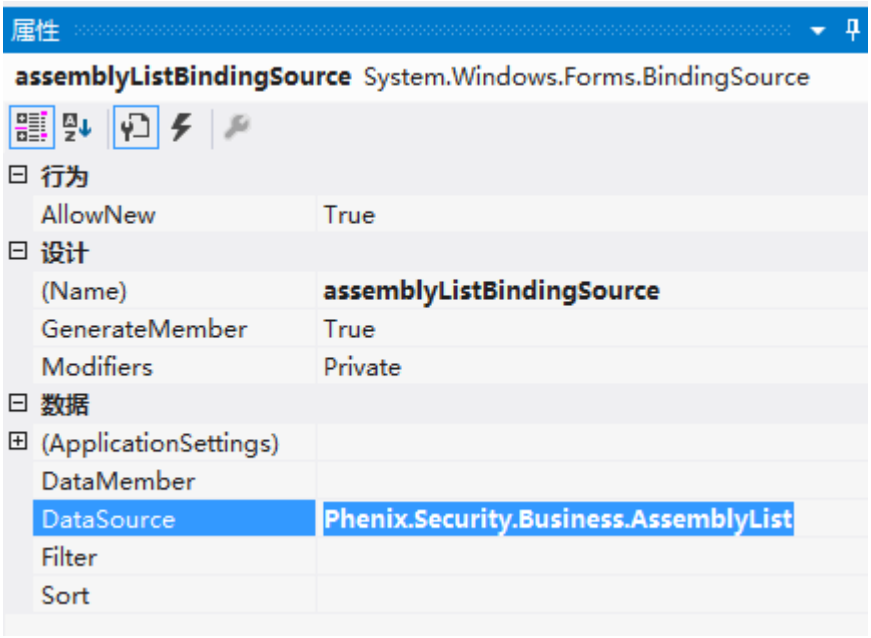
有关主从结构的绑定，请参见第 4.3 章节。以 Phenix.Security.Business 工程的 Assembly 和 AssemblyClass 业务结构为例：

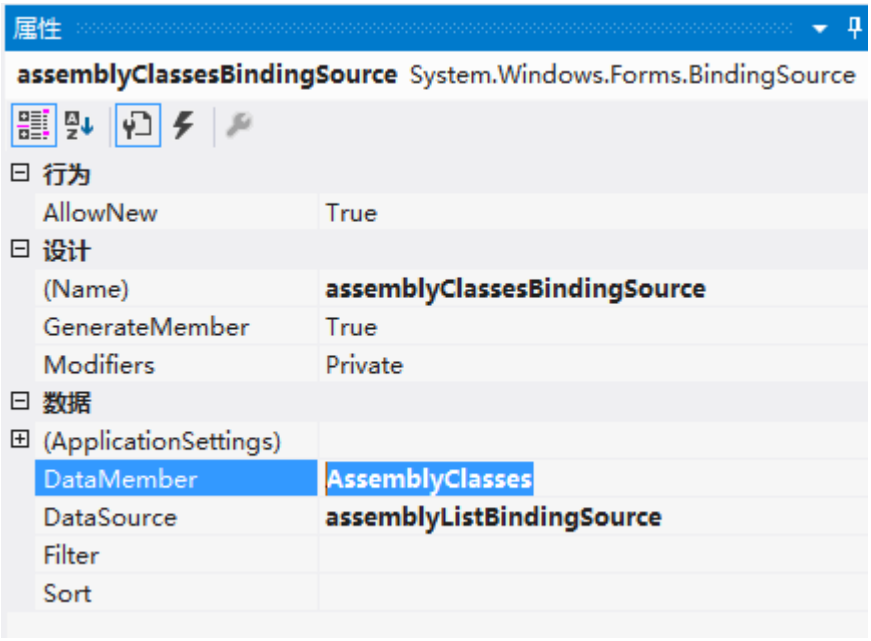
```
  /// <summary>
  /// 程序集
  /// </summary>
  [Serializable]
  public class Assembly : Assembly<Assembly>
  {
    #region 属性

    /// <summary>
    /// 类信息
    /// </summary>
    public AssemblyClassList AssemblyClasses
    {
```

```
      get
      {
        AssemblyClassList result = FindCompositionDetail<AssemblyClassList, AssemblyClass>();
        if (result == null)
          AssemblyClassList.Fetch().CompositionFilter(Owner).TryGetValue(this, out result);
        return result;
      }
    }

    #endregion
  }
```

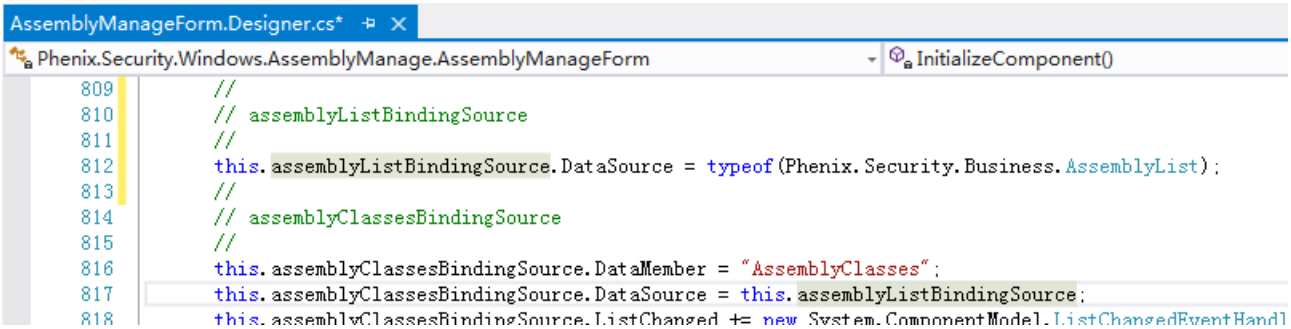
Phenix.Security.Windows.AssemblyManage 工程主界面的 BindingSource 设置如下：





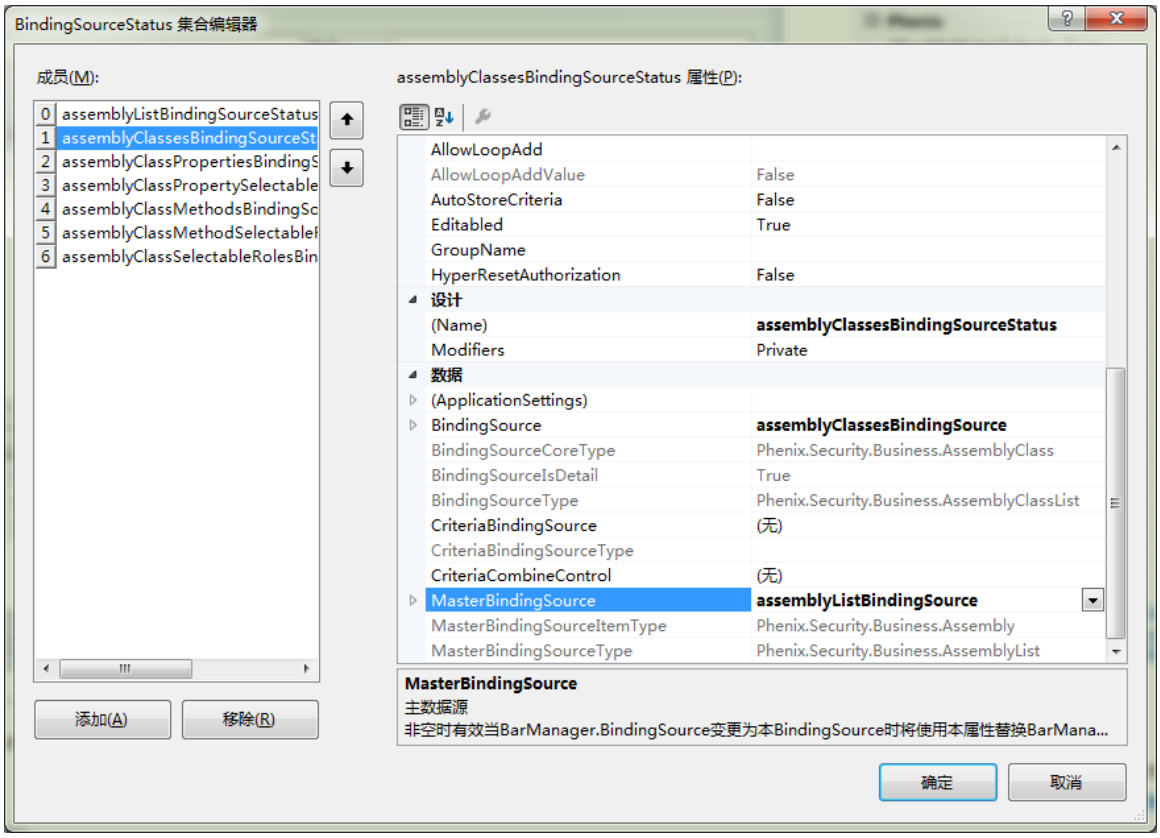


切换到 designer 文件中，代码如下：



一旦设计好各个 BindingSource 的结构关系，如果在界面上使用到了 BarManager 组件，应该在这个

组件上点击其“Reset BindingSources”菜单功能，重置其 BindingSources 属性：



我们可以通过浏览 BindingSources 属性的清单内容，检查各个 BindingSource 的结构关系是否已设计正确。

BarManager 组件可以自动实现界面数据的 CRUD 和持久化，无需自行编写相关代码，具体请见“10.BarManager 组件”章节。