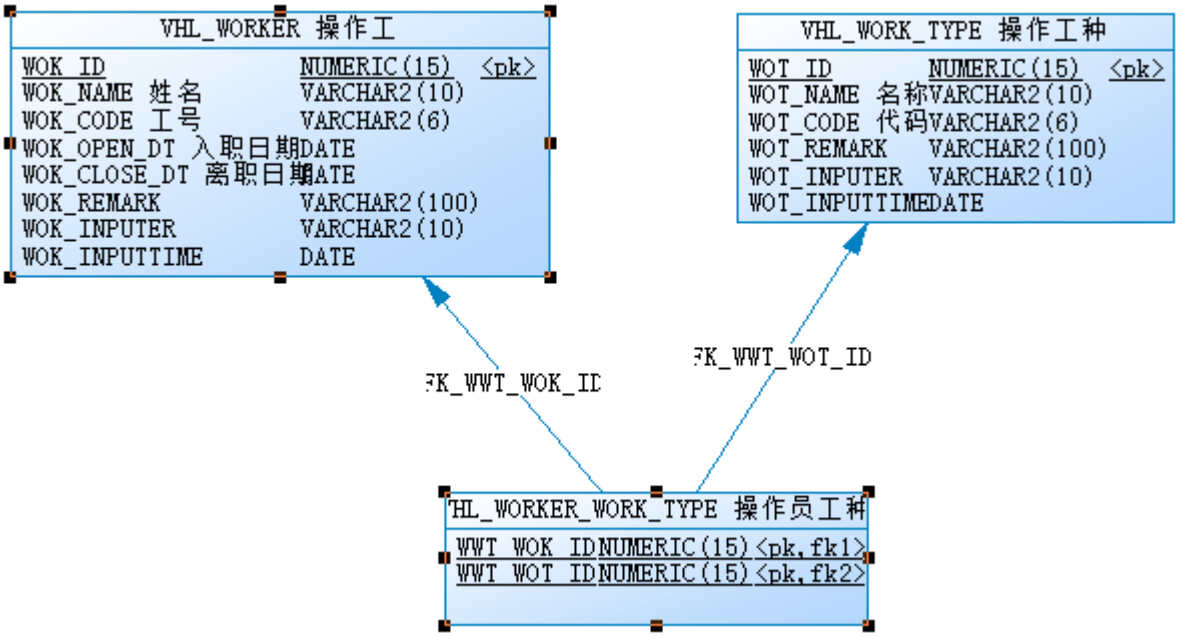
12 业务结构对象模型

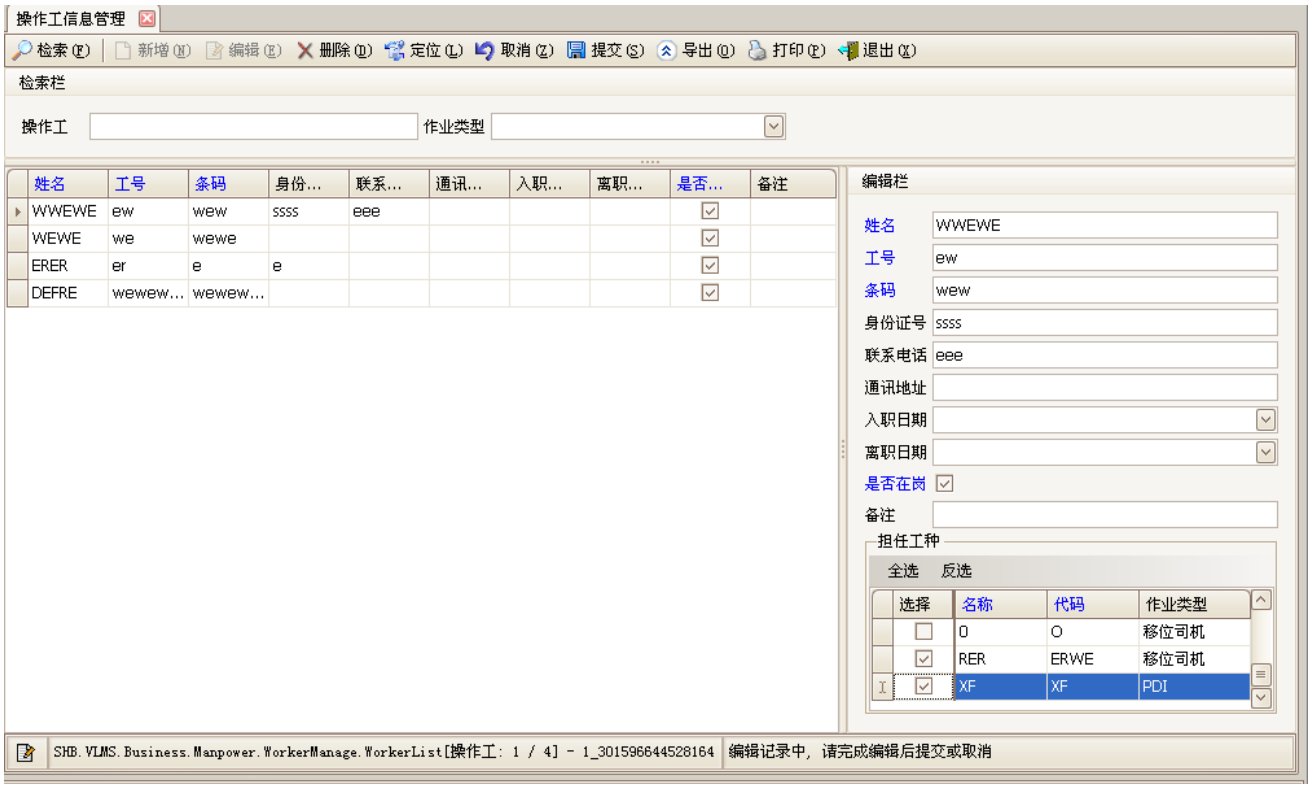
12.9 多对多表结构关系及其业务结构

12.9.1单纯联结表的多对多数据处理方法

多对多是较为复杂的数据结构，在关系数据库中为了表示多对多关系，必须创建第三个表，该表通常称为联结表，它将多对多关系划分为两个一对多关系：

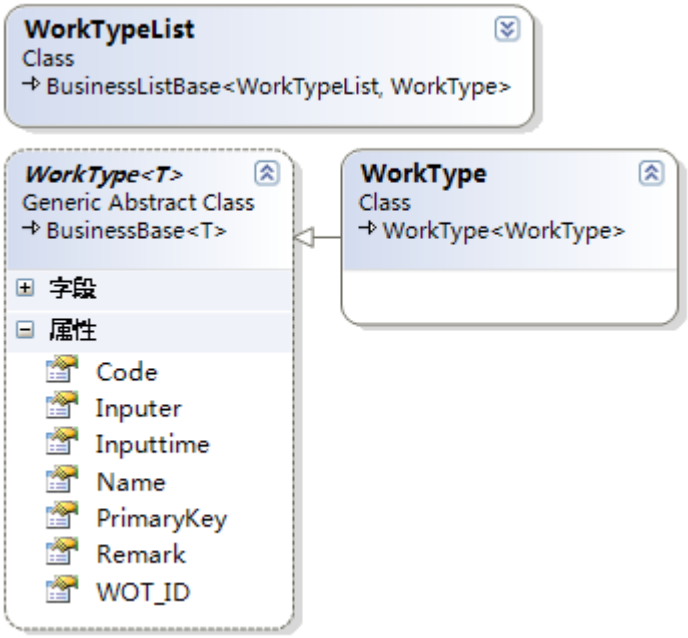


在具体的业务场景中，这两个主表必定有一个起着主导作用，并与联结表分别映射构建起一个主从业务结构；而从另一个主表映射出来的业务类及其集合类，主要是为了供界面层绑定为勾选清单。这样，似乎可以按照普通的主从业务结构来设计开发，但这仅仅完成了一半的工作，剩下的工作将是如何满足业务场景的界面交互设计需求：

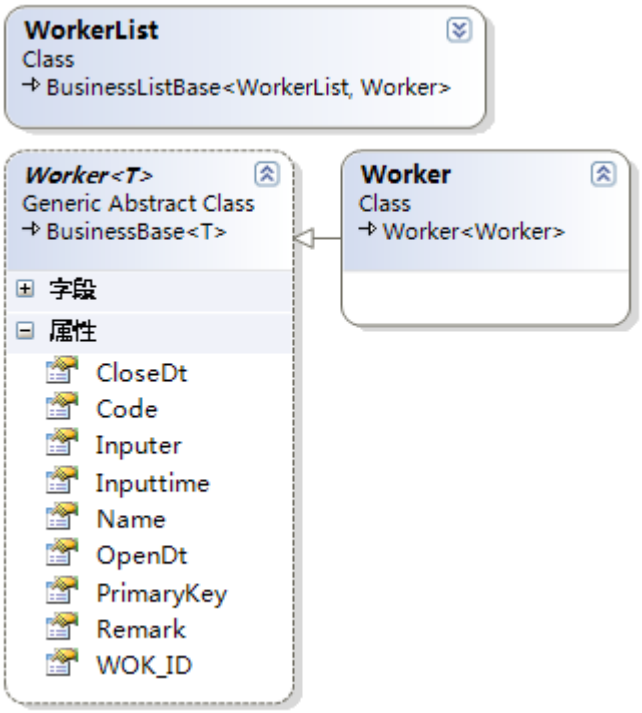


以上述界面设计需求为例：“操作工”所“担任的工种”，将从“工种”全集中挑选出来，也就是说，需要在“担任工种”栏中将所有可能的“工种”都罗列出来供勾选。

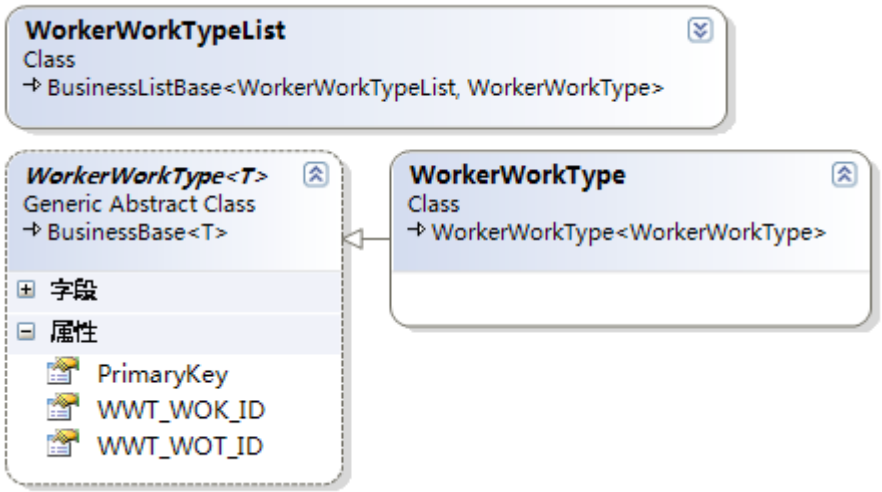
供挑选的“工种”全集绑定的是 WorkType 类（映射 VHL_WORKE_TYPE 表）及其集合类：



“操作工”绑定的是 WorkerList 类（映射 VHL_WORKER 表）及其集合类：



“操作工”所“担任的工种”绑定的是 WorkerWorkType 类（映射 VHL_WORKER_WORK_TYPE 表）及其集合类：



这样，当主业务集合对象 WorkerList 在初始化、移动游标的时候，必须用其从业务集合对象 WorkerWorkTypeList 来刷新供挑选的全集清单 WorkTypeList 中每个 WorkType 对象的 Selected 属性值（Selected = true 代表 WorkerWorkTypeList 中有其对应的 WorkerWorkType 对象（WWT_WOT_ID = WOT_ID））；而当其 Selected 属性值发生改变时，也必须同步到 WorkerWorkTypeList 上（增删 WorkerWorkType 对象）。。。那这些代码是不是即繁琐又无聊？甚至这些代码势必要写在了界面控制层上？

Phenix丶为此提供了简便的设计模式，在 Phenix.Business.BusinessListBase<T, TBusiness>里有如下的函数：

```
/// <summary>
/// 整理勾选项清单
/// 当本集合为A(其主业务对象集合)、B集合(source)的交叉关联集合时，可返回刷新过(与本集合项存在关联的项Selected都被置为true)的B集合，而当它发生变更时将即时反映到本集合
/// emptyIsAllSelected = false
/// </summary>
/// <param name="source">源业务集合</param>
public TSelectableList CollatingSelectableList<TSelectableList, TSelectable>(TSelectableList source)
  where TSelectableList : BusinessListBase<TSelectableList, TSelectable>
  where TSelectable : BusinessBase<TSelectable>
```

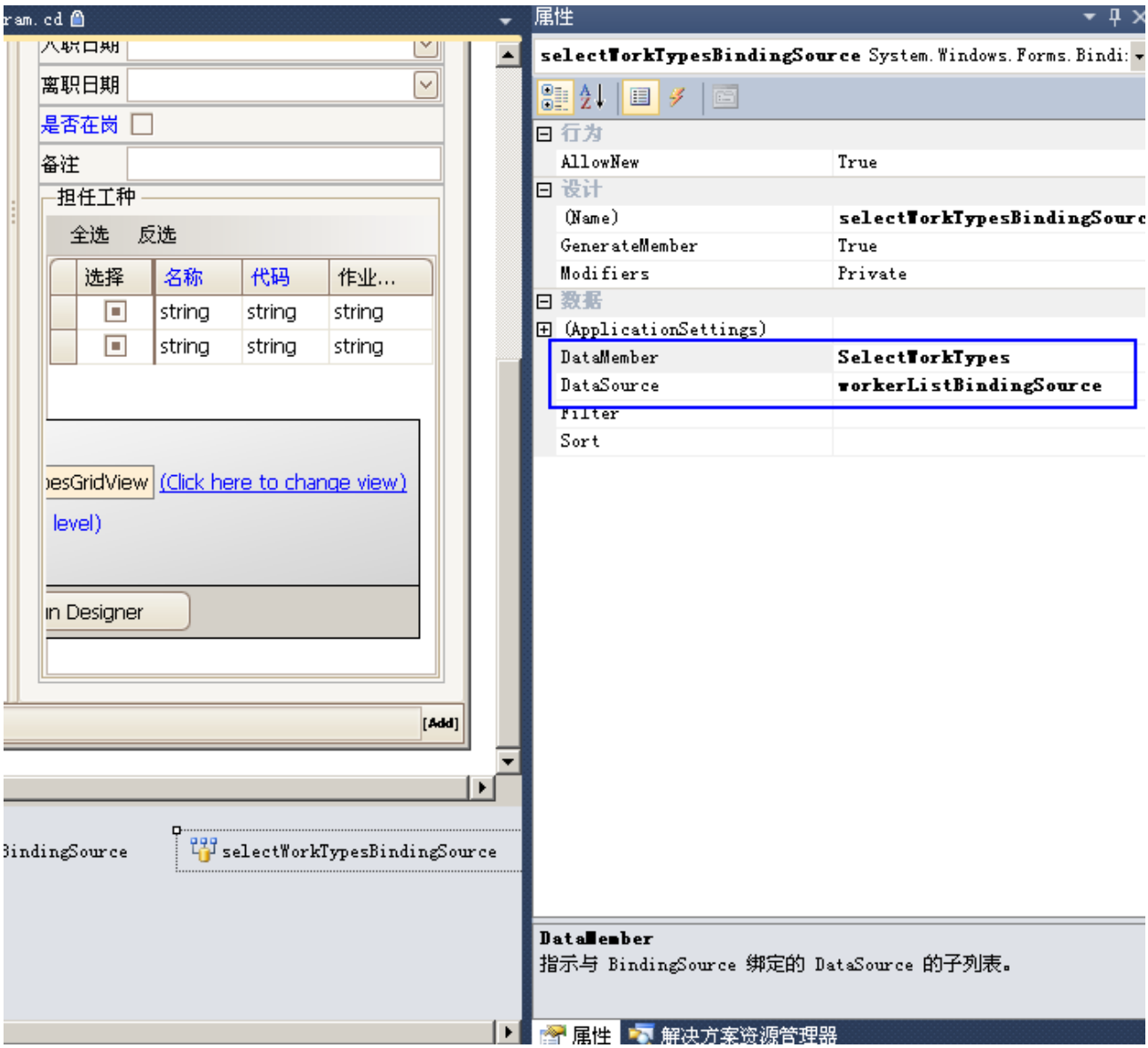
我们可以在 Worker 业务类中定义如下属性：

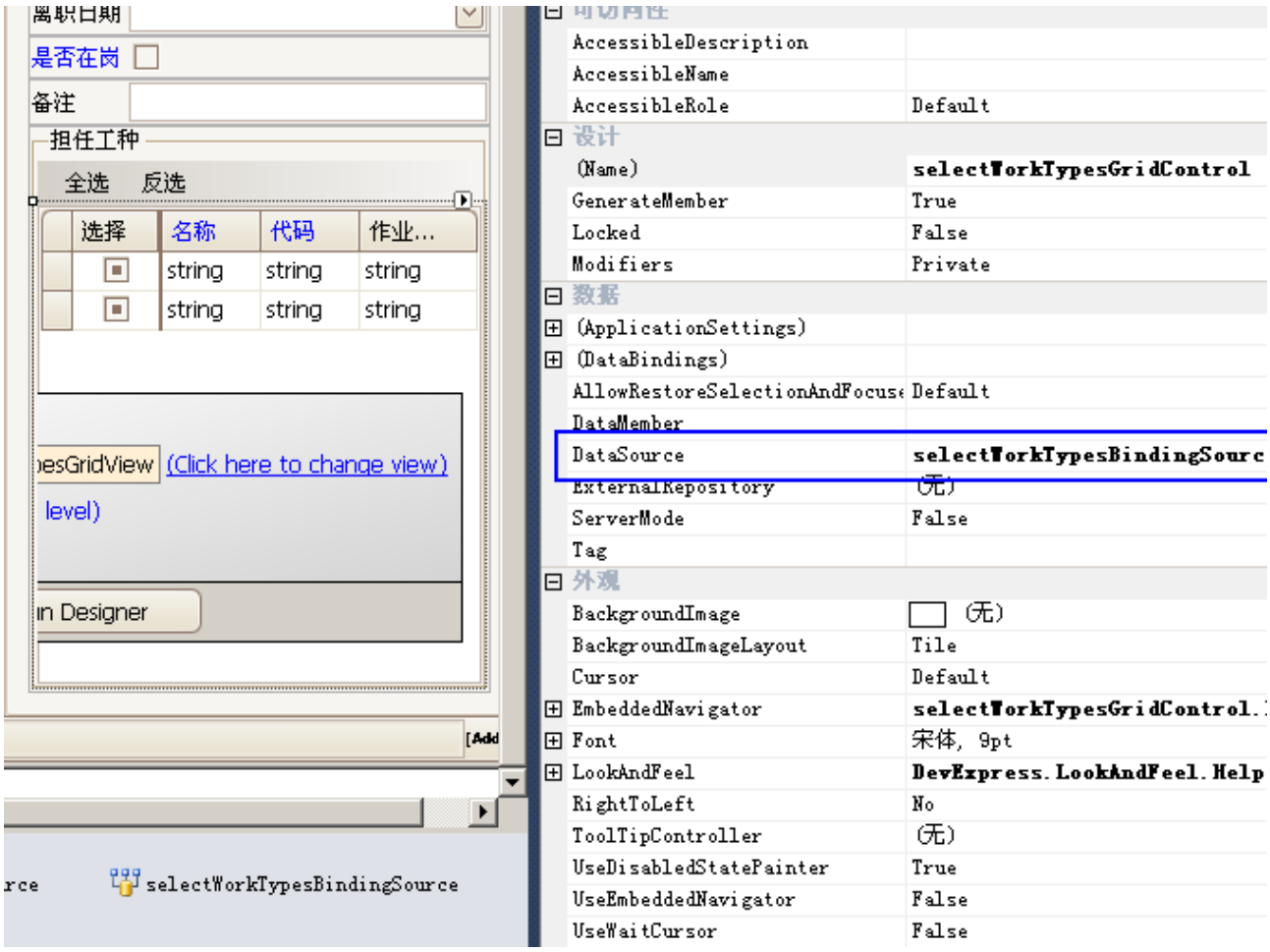
```
/// <summary>
/// 操作工
/// </summary>
[Serializable]
public class Worker : Worker<Worker>
{
  /// <summary>
  /// 供挑选的工种全集
  /// </summary>
  public WorkTypeList SelectWorkTypes
  {
    get
    {
      //担任的工种
      WorkerWorkTypeList workerWorkTypes = GetCompositionDetail<WorkerWorkTypeList, WorkerWorkType>();
      //供挑选的工种全集
      return workerWorkTypes.CollatingSelectableList<WorkTypeList, WorkType>(WorkTypeList.Fetch());
    }
  }
}
```

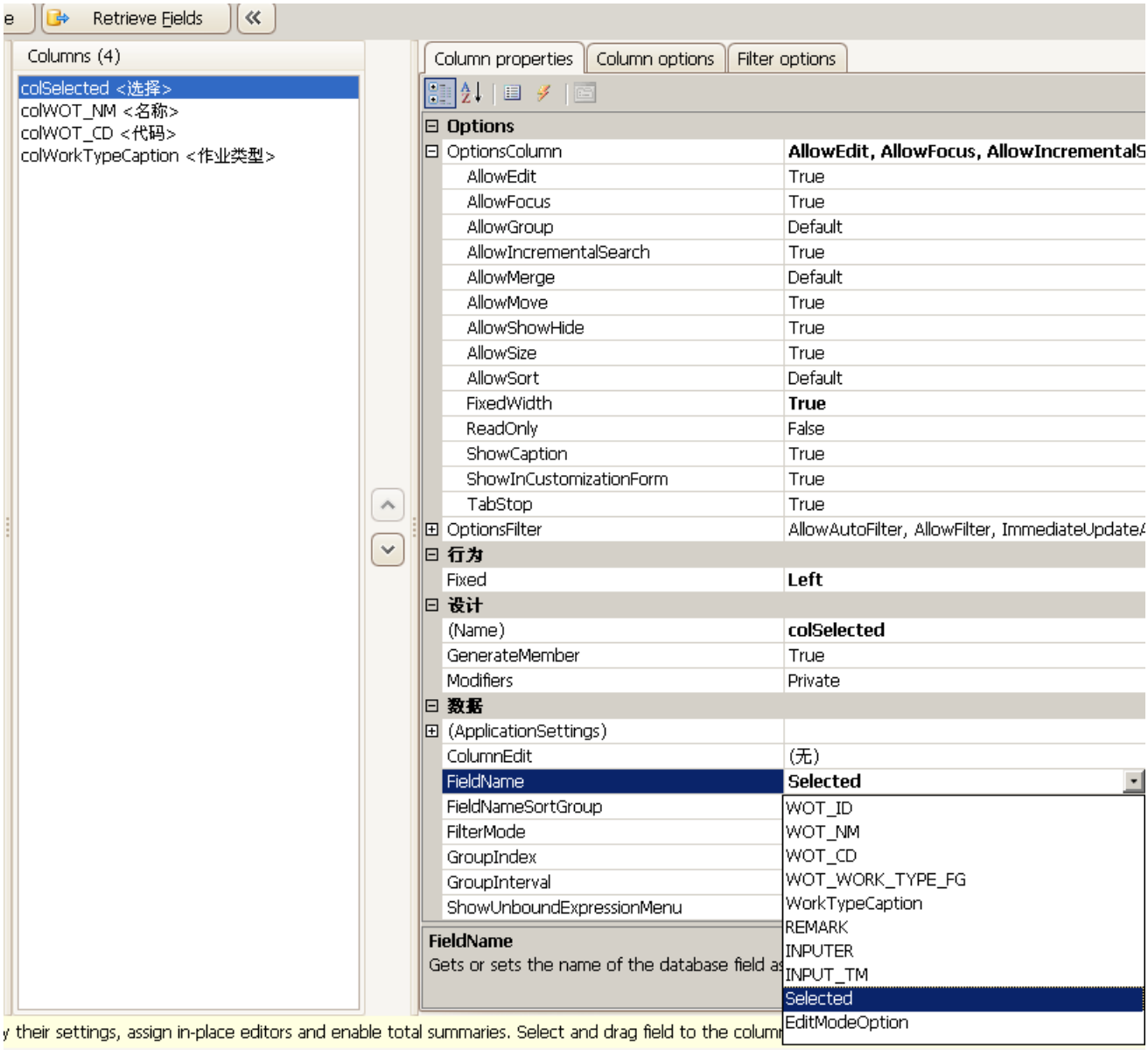
从业务集合对象 workerWorkTypes 被隐藏在了主业务对象 Worker 中，由 Phenix丶负责它们之间的数据同步以及持久化，而对外的接口仅是供挑选的 SelectWorkTypes 属性。

如此，上图界面设计的“担任工种”栏中清单的数据源，将绑定为 Worker 对象的 SelectWorkTypes

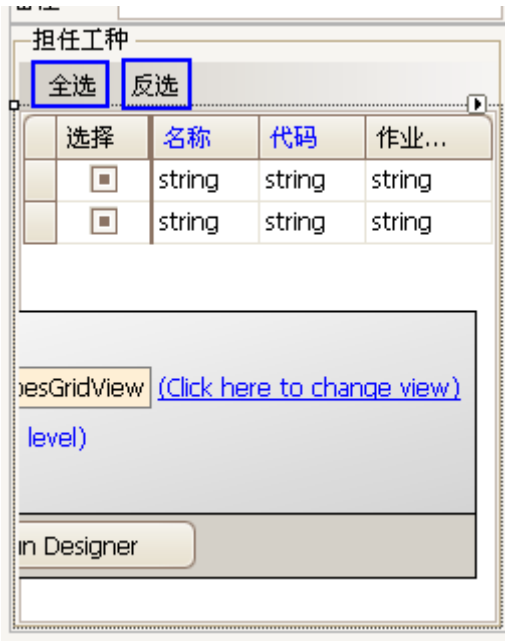
属性：


属性
selectWorkTypesBindingSource System.Windows.Forms.Bindi
行为
AllowNew True
设计
(Name) selectWorkTypesBindingSourc
GenerateMember True
Modifiers Private
数据
(ApplicationSettings)
DataMember SelectWorkTypes
DataSource workerListBindingSource
Filter
Sort
DataMember
指示与 BindingSource 绑定的 DataSource 的子列表。
属性
解决方案资源管理器
selectWorkTypesBindingSource
离职日期
是否在岗
备注
担任工种
全选
反选
选择
名称
代码
作业...
string
(Click here to change view)
level)
in Designer
[Add]


是否在岗
备注
担任工种
全选 反选
选择 名称 代码 作业...
string string string
string string string
(Click here to change view)
level)
in Designer
selectWorkTypesBindingSource
AccessibleDescription
AccessibleName
AccessibleRole Default
设计
(Name) selectWorkTypesGridControl
GenerateMember True
Locked False
Modifiers Private
数据
(ApplicationSettings)
(DataBindings)
AllowRestoreSelectionAndFocus Default
DataMember
DataSource selectWorkTypesBindingSourc
ExternalRepository (无)
ServerMode False
Tag
外观
BackgroundImage (无)
BackgroundImageLayout Tile
Cursor Default
EmbeddedNavigator selectWorkTypesGridControl.
Font 宋体, 9pt
LookAndFeel DevExpress.LookAndFeel.Help
RightToLeft No
ToolTipController (无)
UseDisabledStatePainter True
UseEmbeddedNavigator False
UseWaitCursor False



为了方便用户勾选，界面上提供两个功能按钮，全选和反选：



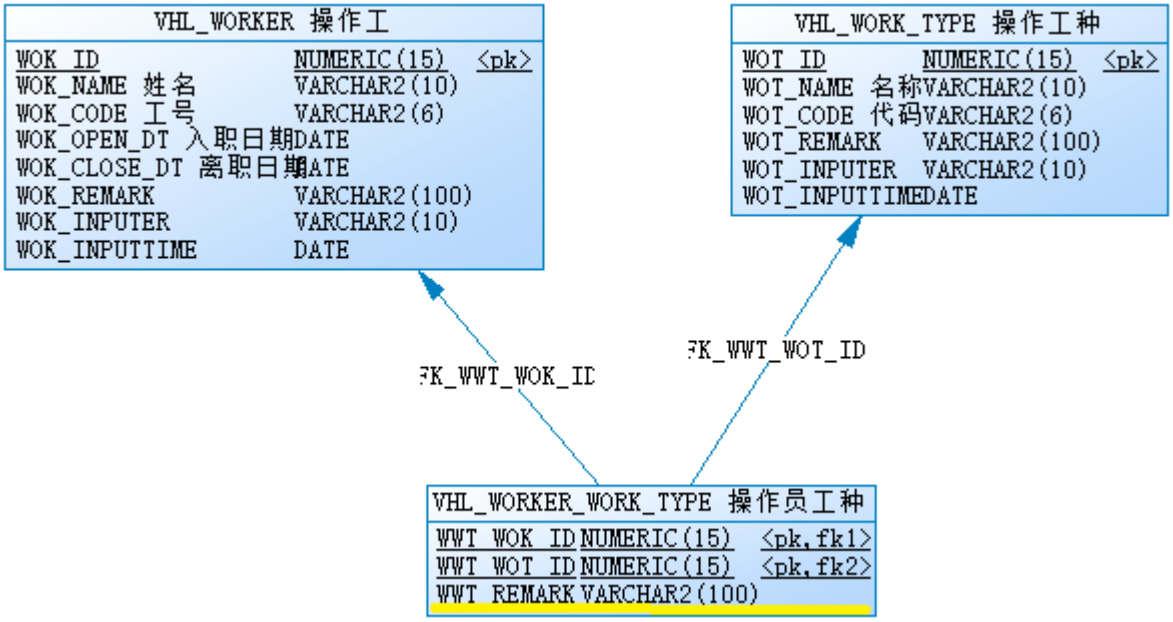
它们的触发事件如下（界面完整代码）：

```
public partial class WorkerManageForm : BaseForm
{
  public WorkerManageForm()
  {
    InitializeComponent();
  }

  #region 属性

  private WorkTypeList SelectWorkTypeList
  {
    get { return this.selectWorkTypesBindingSource.List as WorkTypeList; }
  }

  #endregion

  private void selectAllToolStripMenuItem_Click(object sender, System.EventArgs e)
  {
    SelectWorkTypeList.SelectAll();
  }

  private void InverseAllToolStripMenuItem_Click(object sender, System.EventArgs e)
  {
    SelectWorkTypeList.InverseAll();
  }
}
```

除此之外，我们无需在界面控制层、业务逻辑层上编写多余一行涉及维护（增删）从业务对象相关的控制代码。

12.9.2带属性联结表的多对多数据处理方法

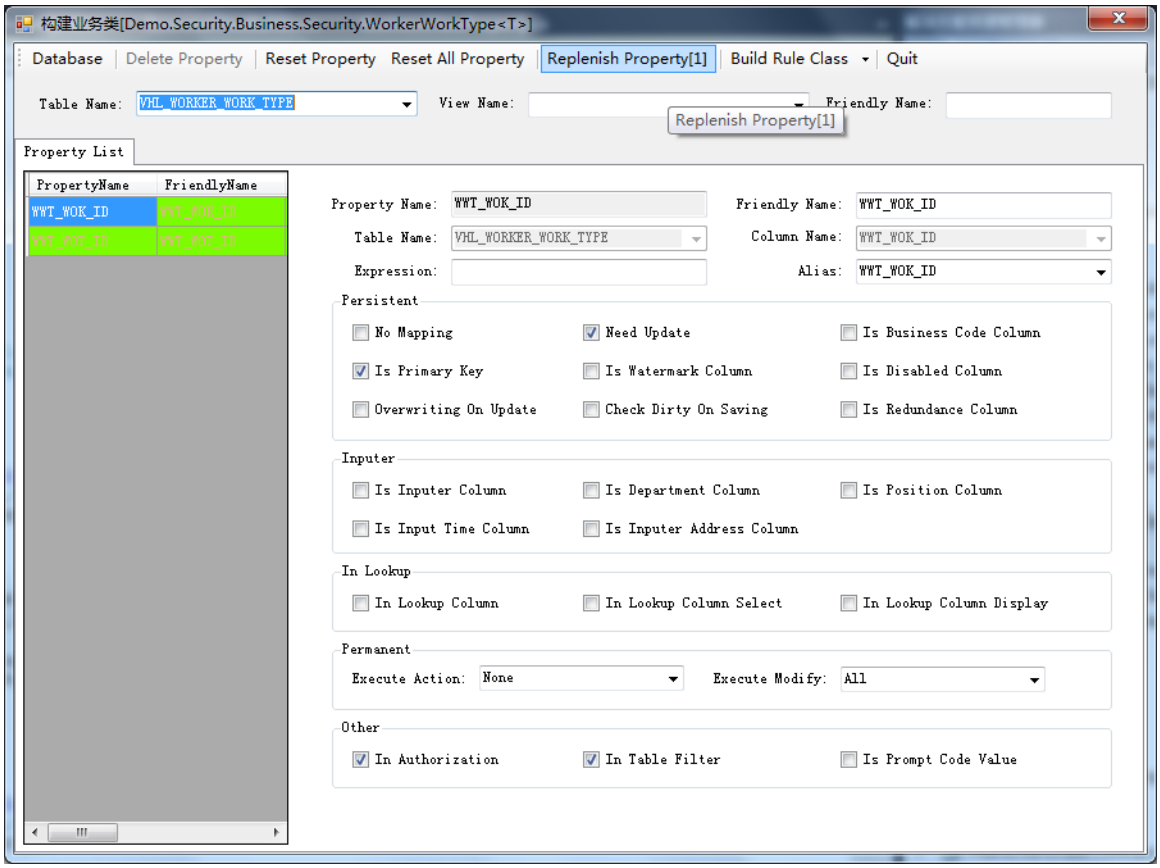
前文中所设计的联结表中仅包含了两主表的外键，如果联结表中还包含着需要编辑的字段，那如何处理？：

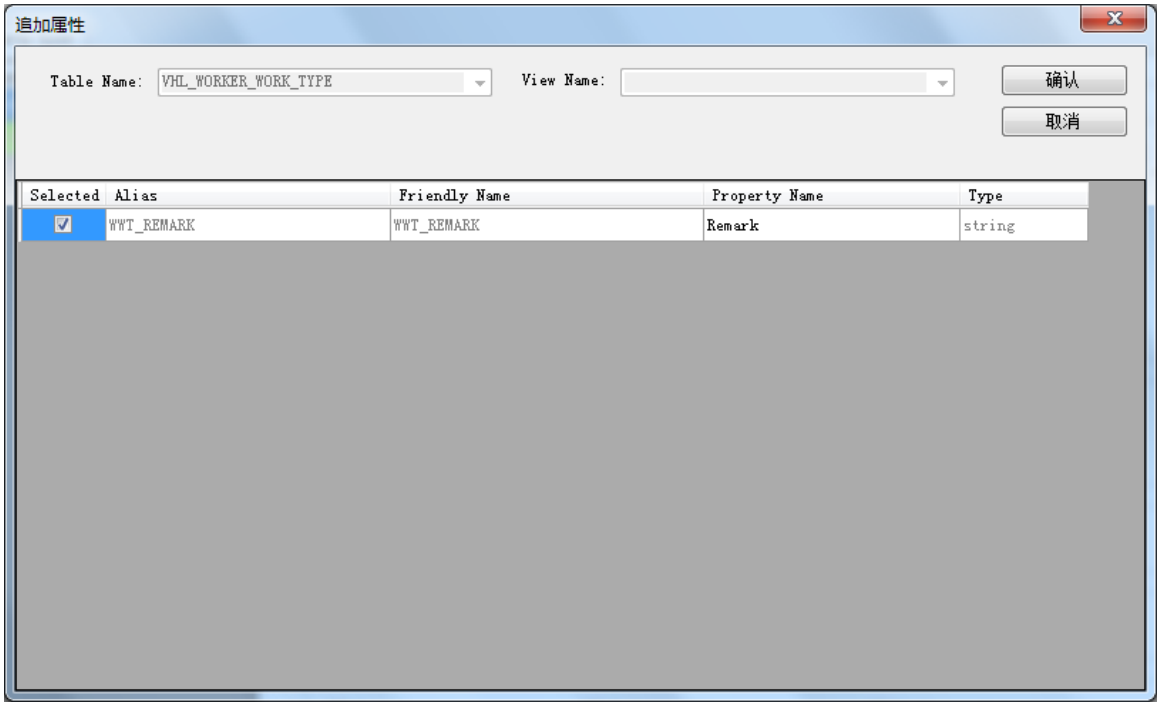


也就是说，在勾选业务对象的时候，还需要编辑被勾选业务对象上的业务数据。

下面我们一步步演示设计方法。

首先，为了这新增的字段，需要在 WorkerWorkType 类中补充上它的映射关系：

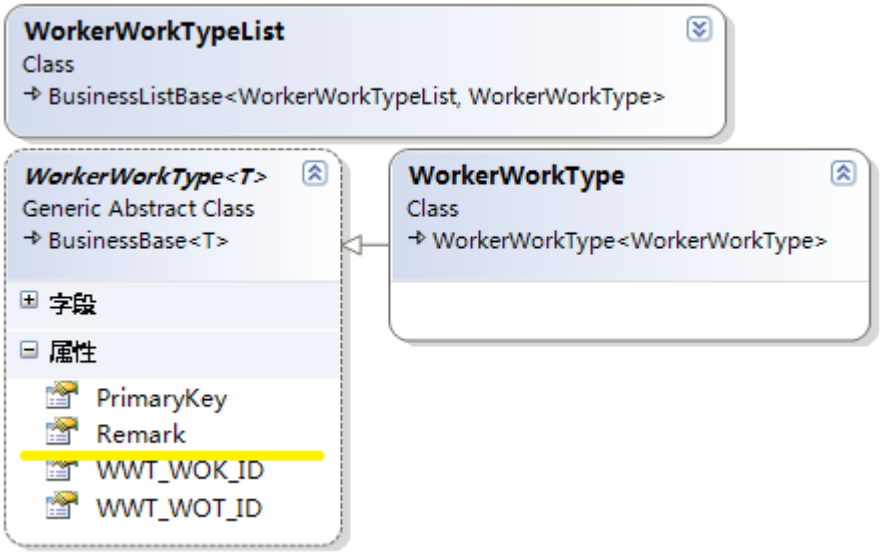





```
/// <summary>
/// 备注
/// </summary>
public static readonly Phenix.Business.PropertyInfo<string> RemarkProperty =
RegisterProperty<string>(c => c.Remark);
[Phenix.Core.Mapping.Field(FriendlyName = "备注", Alias = "WWT_REMARK", TableName =
"VHL_WORKER_WORK_TYPE", ColumnName = "WWT_REMARK", NeedUpdate = true)]
```

```
    private string _remark;
    /// <summary>
    /// 备注
    /// </summary>
    [System.ComponentModel.DisplayName("备注")]
    public string Remark
    {
      get { return GetProperty(RemarkProperty, _remark); }
      set { SetProperty(RemarkProperty, ref _remark, value); }
    }
```

重构之后，WorkerWorkType 类（映射 VHL_WORKER_WORK_TYPE 表）及其集合类：



然后，我们将这新增的映射关系拷贝到 WorkType 类中，因为界面上是直接操作 WorkType 对象的，而 Phenix 可以自动同步 WorkType 与 WorkerWorkType 之间相同映射关系的数据，所以，修改 WorkType 的 Remark 属性等于修改了 WorkerWorkType 的 Remark 属性：

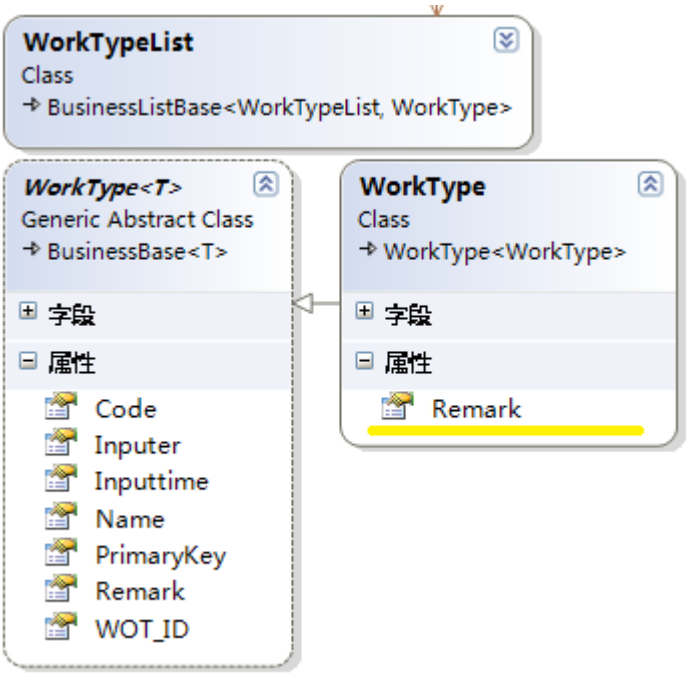
```
  /// <summary>
  /// 工种
  /// </summary>
  [Serializable]
  public class WorkType : WorkType<WorkType>
  {
    /// <summary>
    /// 备注
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> RemarkProperty =
RegisterProperty<string>(c => c.Remark);
    [Phenix.Core.Mapping.Field(FriendlyName = "备注", Alias = "WWT_REMARK", TableName =
```

```
"VHL_WORKER_WORK_TYPE", ColumnName = "WWT_REMARK", NoMapping = true, IsRedundanceColumn = true]
    private string _remark;
    /// <summary>
    /// 备注
    /// </summary>
    [System.ComponentModel.DisplayName("备注")]
    public string Remark
    {
      get { return GetProperty(RemarkProperty, _remark); }
      set { SetProperty(RemarkProperty, ref _remark, value); }
    }
  }
```

注意 Field 映射标签上要:

- 设置成不允许映射（NoMapping = true），因为它不应该被拼装到 Fetch 的 SQL 语句中。
- 设置成冗余字段（IsRedundanceColumn = true），以便实现字段值的自动同步。

重构之后，供挑选的 WorkType 类（映射 VHL_WORKE_TYPE 表）及其集合类:



Worker 业务类中原先定义的 SelectWorkTypes 属性无需重构，下面的代码仅是回顾一下如何调用 CollatingSelectableList()函数:

```
  /// <summary>
  /// 操作工
  /// </summary>
  [Serializable]
  public class Worker : Worker<Worker>
```

```
  {
    /// <summary>
    /// 供挑选的工种全集
    /// </summary>
    public WorkTypeList SelectWorkTypes
    {
      get
      {
        //担任的工种
        WorkerWorkTypeList workerWorkTypes = GetCompositionDetail<WorkerWorkTypeList,
WorkerWorkType>();
        //供挑选的工种全集
        return workerWorkTypes.CollatingSelectableList<WorkTypeList, WorkType>(WorkTypeList.Fetch());
      }
    }
  }
```

重构完成后，我们可以用下面的代码测试一下：

```
      WorkerList workerList = WorkerList.Fetch();
      foreach (Worker worker in workerList)
        foreach (WorkType workType in worker.SelectWorkTypes)
        {
          workType.Selected =! workType.Selected;
          if (workType.Selected)
            workType.Remark = "ok";
        }
      workerList.Save();
```