

12 业务结构对象模型

12.10 枚举在业务结构中的使用方法

在业务结构中经常会用到枚举（见“03.Addin 工具使用方法”的“为什么用枚举”章节），为此，Phenix 提炼出一些通用的方法以达到快速开发的目的，也避免了八仙过海各显神通式的开发行为，方便应用系统的维护。

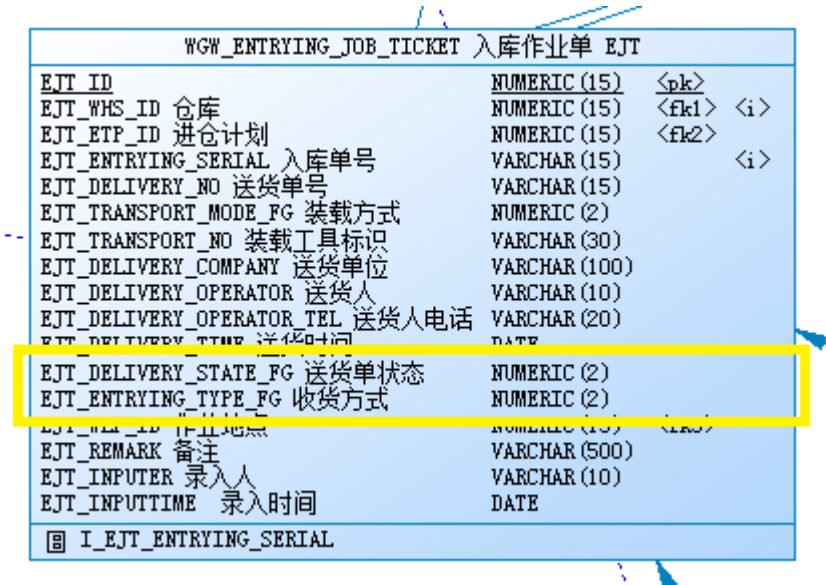
12.10.1 规范化约定

既然业务对象上用到了枚举类型的属性，那如何持久化它们？有两种持久化方法，一种是定义对应的字符串型表字段，存放枚举值的名称；一种是定义对应的整型表字段，存放枚举值的 Flag。权衡下来，我们选用了后一种方法，是因为整型的处理效率最高，而且不会因为枚举值名称的改变造成与表字段值对应不上发生的系统软故障问题。因为枚举的 Flag 值是无逻辑含义的，只要做到在重构枚举定义的时候，不改变那些旧的枚举值 Flag 即可。

为了达到快速开发、减轻编程负担的目的，我们在设计表结构的时候，需要遵守 Phenix 在 Phenix.Core.Mapping.CodingStandards 类中约定的表字段命名规则：

属性	说明	备注
DefaultEnumTagColumnName	缺省“枚举标志”字段名	缺省值：字段名后缀为“_FG”；

并且，表字段类型必须定义为 NUMERIC(2)（注：如果定义为 NUMERIC(1)，则会被自动映射为布尔型业务类字段和属性）：



WGW_ENTRYING_JOB_TICKET 入库作业单 EJT		
EJT_ID	NUMERIC (15)	<pk>
EJT_WHS_ID 仓库	NUMERIC (15)	<fk1> <i>
EJT_ETP_ID 进仓计划	NUMERIC (15)	<fk2>
EJT_ENTRYING_SERIAL 入库单号	VARCHAR (15)	<i>
EJT_DELIVERY_NO 送货单号	VARCHAR (15)	
EJT_TRANSPORT_MODE_FG 装载方式	NUMERIC (2)	
EJT_TRANSPORT_NO 装载工具标识	VARCHAR (30)	
EJT_DELIVERY_COMPANY 送货单位	VARCHAR (100)	
EJT_DELIVERY_OPERATOR 送货人	VARCHAR (10)	
EJT_DELIVERY_OPERATOR_TEL 送货人电话	VARCHAR (20)	
EJT_DELIVERY_TIME 送货时间	DATE	
EJT_DELIVERY_STATE_FG 送货单状态	NUMERIC (2)	
EJT_ENTRYING_TYPE_FG 收货方式	NUMERIC (2)	
EJT_REMARK 备注	VARCHAR (500)	
EJT_INPUTER 录入人	VARCHAR (10)	
EJT_INPUTTIME 录入时间	DATE	
I_EJT_ENTRYING_SERIAL		

这样，我们通过 Phenix 的 Addin 工具在初始化/编辑业务类时（见“03.Addin 工具使用方法”的“初

始化/编辑业务类”章节)，才能自动生成出正确的映射关系，业务类上字段和属性才会有对应的枚举类型定义（枚举类型名，为表字段名剔除前、后缀且转换为 PascalCasing 型的字符串）：

```

    /// <summary>
    /// 送货单状态
    /// </summary>

    public static readonly Phenix.Business.PropertyInfo<DeliveryState?> DeliveryStateProperty =
RegisterProperty<DeliveryState?>(c => c.DeliveryState, EntryingWork.Rule.DeliveryState.Arrived);
        [Phenix.Core.Mapping.Field(FriendlyName = "送货单状态", Alias = "EJT_DELIVERY_STATE_FG",
TableName = "WGW_ENTRYING_JOB_TICKET", ColumnName = "EJT_DELIVERY_STATE_FG", NeedUpdate = true,
CheckDirtyOnSaving = true)]
    private DeliveryState? _deliveryState;
    /// <summary>
    /// 送货单状态
    /// </summary>
    [System.ComponentModel.DisplayName("送货单状态")]
    public DeliveryState? DeliveryState
    {
        get { return GetProperty(DeliveryStateProperty, _deliveryState); }
        set { SetProperty(DeliveryStateProperty, ref _deliveryState, value); }
    }
    /// <summary>
    /// 送货单状态标签
    /// </summary>
    [System.ComponentModel.DisplayName("送货单状态标签")]
    public string DeliveryStateCaption
    {
        get { return Phenix.Core.Rule.EnumKeyCaption.GetCaption(_deliveryState); }
    }

    /// <summary>
    /// 收货方式
    /// </summary>

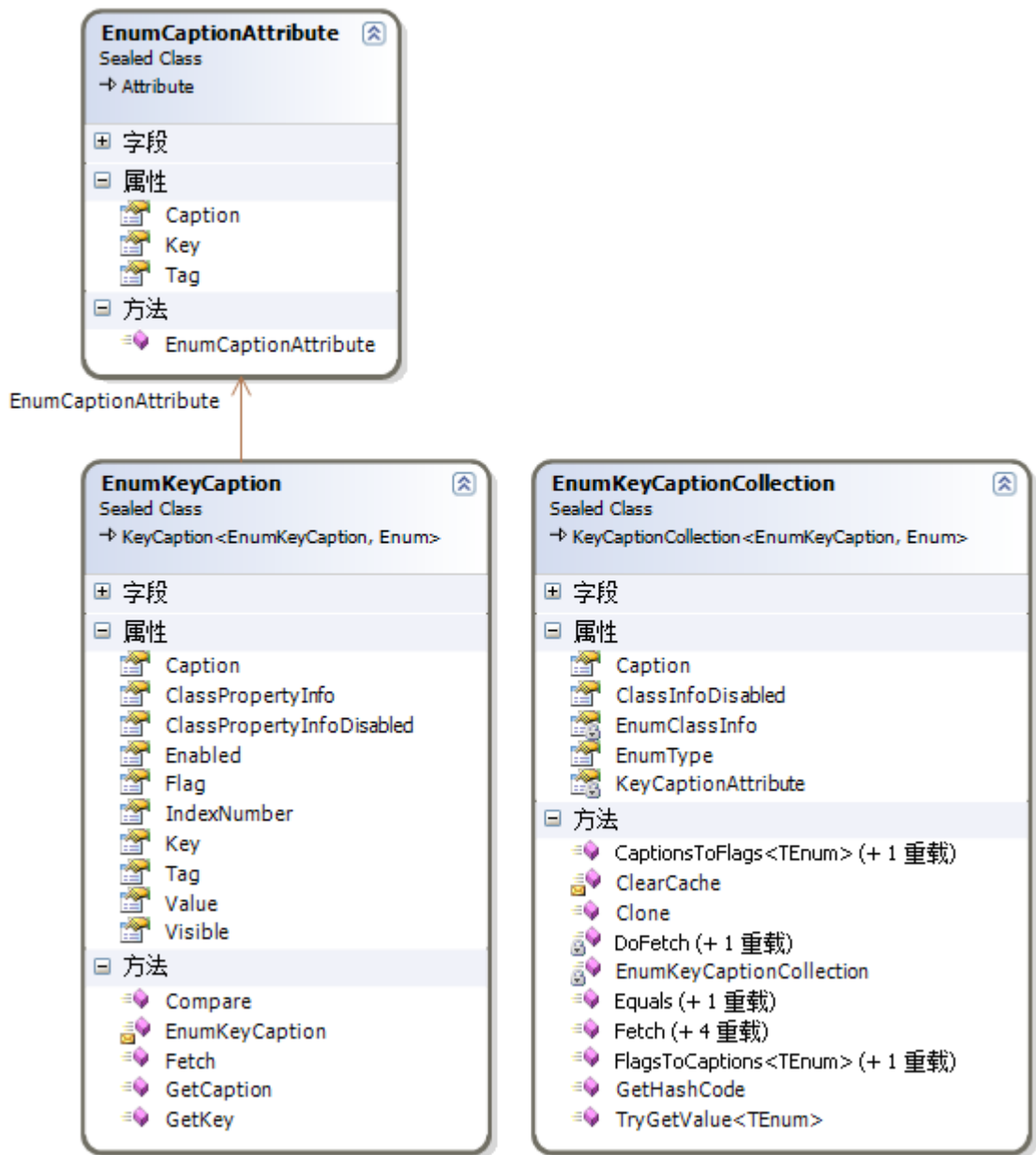
    public static readonly Phenix.Business.PropertyInfo<EntryingType?> EntryingTypeProperty =
RegisterProperty<EntryingType?>(c => c.EntryingType);
        [Phenix.Core.Mapping.Field(FriendlyName = "收货方式", Alias = "EJT_ENTRYING_TYPE_FG", TableName
= "WGW_ENTRYING_JOB_TICKET", ColumnName = "EJT_ENTRYING_TYPE_FG", NeedUpdate = true)]
    private EntryingType? _entryingType;
    /// <summary>
    /// 收货方式
    /// </summary>
    [System.ComponentModel.DisplayName("收货方式")]
    public EntryingType? EntryingType
    {
        get { return GetProperty(EntryingTypeProperty, _entryingType); }
    }

```

```
        set { SetProperty(EntryingTypeProperty, ref _entryingType, value); }
    }
    /// <summary>
    /// 收货方式标签
    /// </summary>
    [System.ComponentModel.DisplayName("收货方式标签")]
    public string EntryingTypeCaption
    {
        get { return Phenix.Core.Rule.EnumKeyCaption.GetCaption(_entryingType); }
    }
}
```

有了这些约定，我们在Fetch和Save业务对象的时候，这些枚举类型字段的持久化，才会是自动完成的，无需我们编写一行代码。

12.10.2 枚举的包装方法



12.10.2.1 枚举标签 EnumCaptionAttribute

枚举标签的应用目的请见“03. Addin 工具使用方法”的“为什么要为枚举打上标签”章节。枚举标签可通过 Addin 工具为枚举自动打上（见“03. Addin 工具使用方法”的“如何为枚举打上标签”章节）。

属性	说明	备注
Key	键	为 null 时采用枚举 Value.ToString("d")；用于界面下拉控件的输入选择；
Caption	标签	为 null 时采用枚举 Value.ToString()；用于界面下拉控件的 DisplayMember；

Tag	标记	留给开发者使用；比如用于分组筛选等；
-----	----	--------------------

使用案例如下：

```

/// <summary>
/// 数据读取级别
/// </summary>
[KeyCaption(FriendlyName = "数据读取级别"), Serializable]
public enum ReadLevel
{
    /// <summary>
    /// 公共
    /// </summary>
    [EnumCaption("公共")]
    Public,

    /// <summary>
    /// 私有
    /// </summary>
    [EnumCaption("私有")]
    Private,

    /// <summary>
    /// 同部门
    /// </summary>
    [EnumCaption("同部门")]
    Department,

    /// <summary>
    /// 同岗位
    /// </summary>
    [EnumCaption("同岗位")]
    Position,

    /// <summary>
    /// 同部门同岗位
    /// </summary>
    [EnumCaption("同部门同岗位")]
    DepartmentPosition
}

```

12.10.2.2 枚举包装类 EnumKeyCaption

一旦为枚举打上了标签，则可以被包装为 EnumKeyCaption 类，使得对枚举的操作和普通的业务类没多大区别。

属性	说明	备注
ValueType	枚举类型	
Value	枚举值	
Key	键	为 null 时采用枚举 Value.ToString("d"); 用于界面下拉控件的输入选择;
Caption	标签	为 null 时采用枚举 Value.ToString(); 用于界面下拉控件的 DisplayMember;
Flag	标记	(int)Convert.ChangeType(Value, typeof(int)) ; 与 EnumKeyCaptionCollection 配合使用支持界面多选控件;
Tag	标记	留给开发者使用; 比如用于分组筛选等;

EnumKeyCaption 类提供了 Fetch 方法:

```
/// <summary>
/// 根据枚举值构建填充
/// </summary>
public static EnumKeyCaption Fetch(Enum value)
```

使用案例如下:

```
EnumKeyCaption readLevelPublicEnumKeyCaption = EnumKeyCaption.Fetch(ReadLevel.Public);
```

12.10.2.3 枚举包装集合类 EnumKeyCaptionCollection

EnumKeyCaption 类的集合类。

属性	说明	备注
EnumType	枚举类型	
Caption	标签	在枚举上 KeyCaptionAttribute 标签的 FriendlyName;

EnumKeyCaptionCollection 类提供了 Fetch 方法:

```
/// <summary>
/// 根据枚举类型定义构建填充
/// </summary>
/// <param name="enumType">枚举类型</param>
public static EnumKeyCaptionCollection Fetch(Type enumType)
```

```

/// <summary>
/// 根据枚举类型定义构建填充
/// </summary>
/// <param name="enumType">枚举类型</param>
/// <param name="match">用于定义要搜索的元素应满足的条件</param>
public static EnumKeyCaptionCollection Fetch(Type enumType, Predicate<EnumKeyCaption> match)

/// <summary>
/// 根据枚举类型定义构建填充
/// </summary>
public static EnumKeyCaptionCollection Fetch<TEnum>()

/// <summary>
/// 根据枚举类型定义构建填充
/// </summary>
/// <param name="match">用于定义要搜索的元素应满足的条件</param>
public static EnumKeyCaptionCollection Fetch<TEnum>(Predicate<EnumKeyCaption> match)

/// <summary>
/// 根据枚举类型定义构建填充
/// </summary>
/// <param name="match">用于定义要搜索的元素应满足的条件</param>
public static EnumKeyCaptionCollection Fetch<TEnum>(Predicate<TEnum> match)

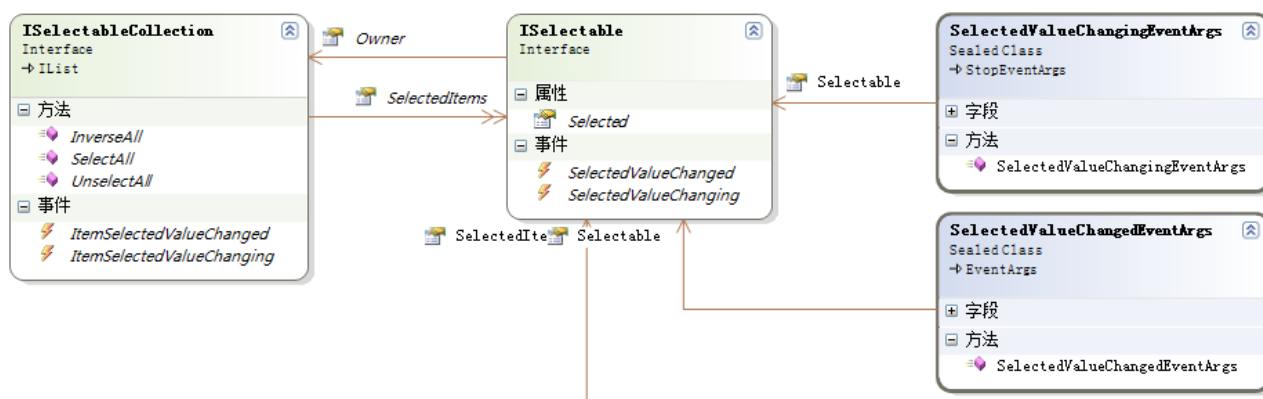
```

使用案例如下：

```
this.readLevelEnumKeyCaptionBindingSource.DataSource =
EnumKeyCaptionCollection.Fetch<ReadLevel>();
```

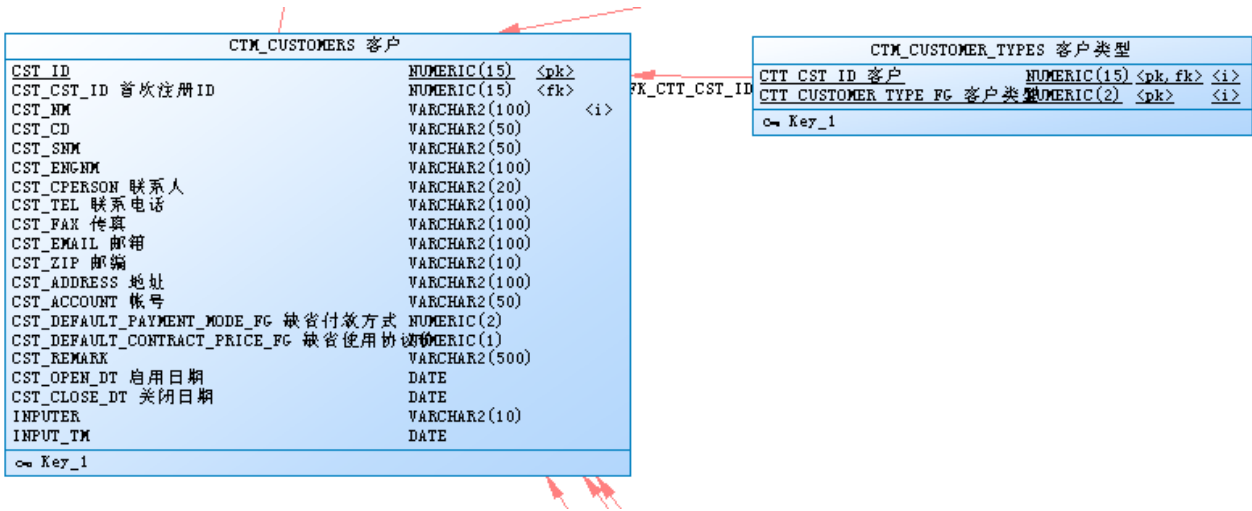
12.10.2.4 可被勾选的枚举

EnumKeyCaptionCollection 实现了 ISelectableCollection 接口，而 EnumKeyCaption 类实现了 ISelectable 接口，它们具备支持勾选操作的能力。



12.10.3 枚举与外键组合为主键的数据处理方法

在一些主从结构中，子表的主键由外键和枚举组成：



这种结构和前文中多对多数据关系无实质区别，完全可以把枚举全集队列看成是另一个主表，这在界面设计中也可以看出它们之间的雷同：

操作信息: 客户管理

检索: 新增 编辑 删除 定位 取消 提交 导出 打印 退出

注册帐户 | 帐户改名 | 查看改名记录 | 编辑持票人 | 编辑经营品牌 | 编辑客户联系人

检索栏

客户名或代码: [] 当前 历史 将来 全部

全部 船代 进口货代 出口货代 船公司 收货单位 送货单位 付款单位 PDI客户 VPC客户 货主

名称	代码	短名	英文名	联系人	联系电话	传真	邮箱	邮编	地址	帐号	付款方式
测试改名...	CSGMDEC		en	link	1tel		e	2121221	ee	ac	支票
SS	SSS										支票

中文名称: 测试改名第二次 英文名: en

代码: CSGMDEC 短名: [] 联系人: link 联系电话: 1tel

邮箱: e 传真: [] 邮编: 2121221

地址: ee

帐号: ac

付款方式: 支票 使用协议 启用日期: 2010-08-05

备注: mo

客户类型: 全选 反选

选择	名称
<input type="checkbox"/>	船代
<input checked="" type="checkbox"/>	进口货代
<input checked="" type="checkbox"/>	出口货代
<input type="checkbox"/>	船公司
<input type="checkbox"/>	收货单位

SHB.VLMS.Business.Customer.CustomerInfo.CustomerList[客户: 1 / 2] - 1_303818745592803 编辑记录中, 请完成编辑后提交或取消

为此，Phenix 在 `Phenix.Business.BusinessListBase<T, TBusiness>` 中提供了如下函数：

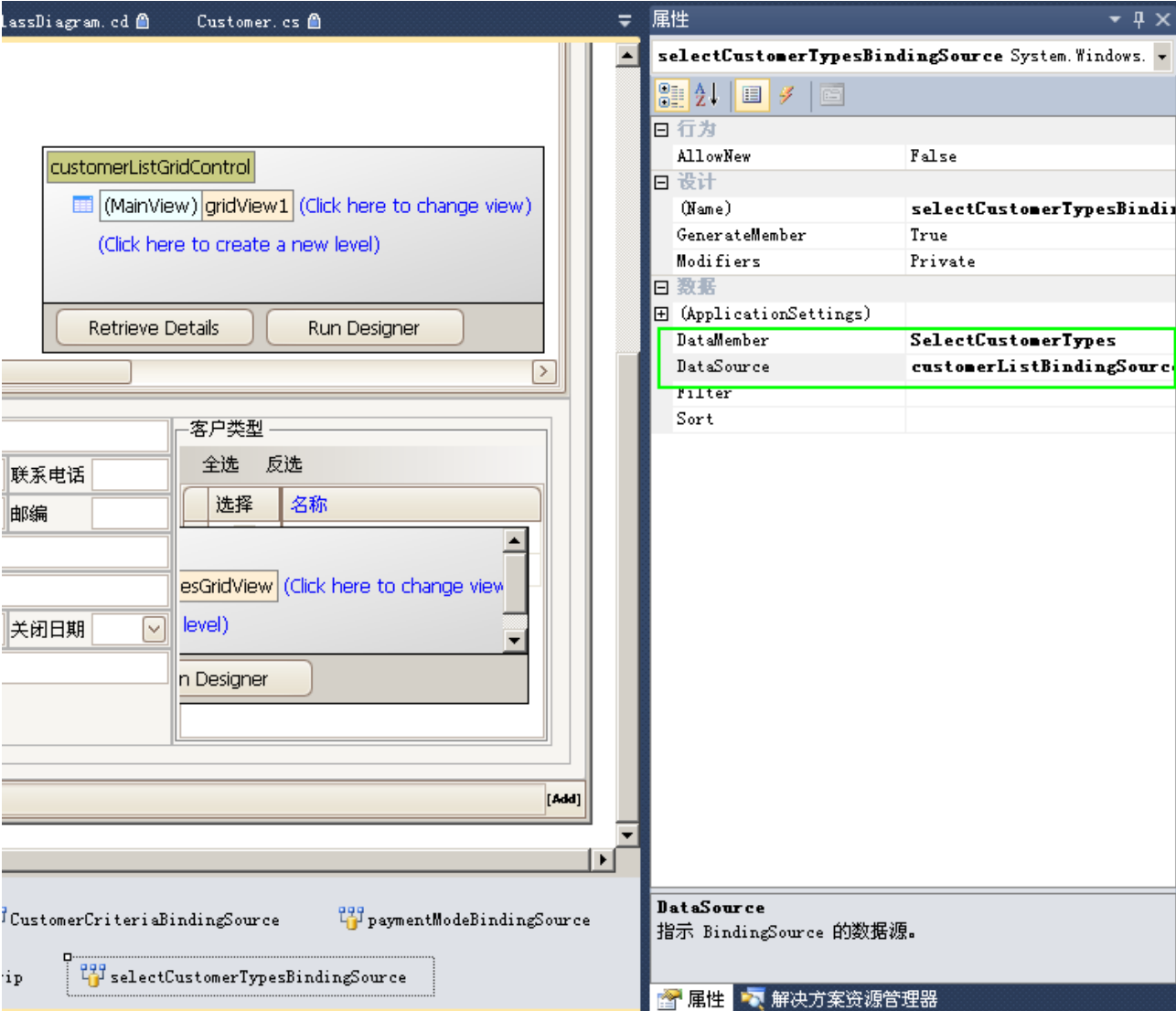
```
/// <summary>
/// 整理勾选项清单
/// 当本集合为A(其主业务对象集合)、B集合(source)的交叉关联集合时，可返回刷新过(与本集合项存在关联
/// 的项Selected都被置为true)的B集合，而当它发生变更时将即时反映到本集合
/// </summary>
/// <param name="source">源枚举集合</param>
public T CollatingSelectableList(EnumKeyCaptionCollection source)
```

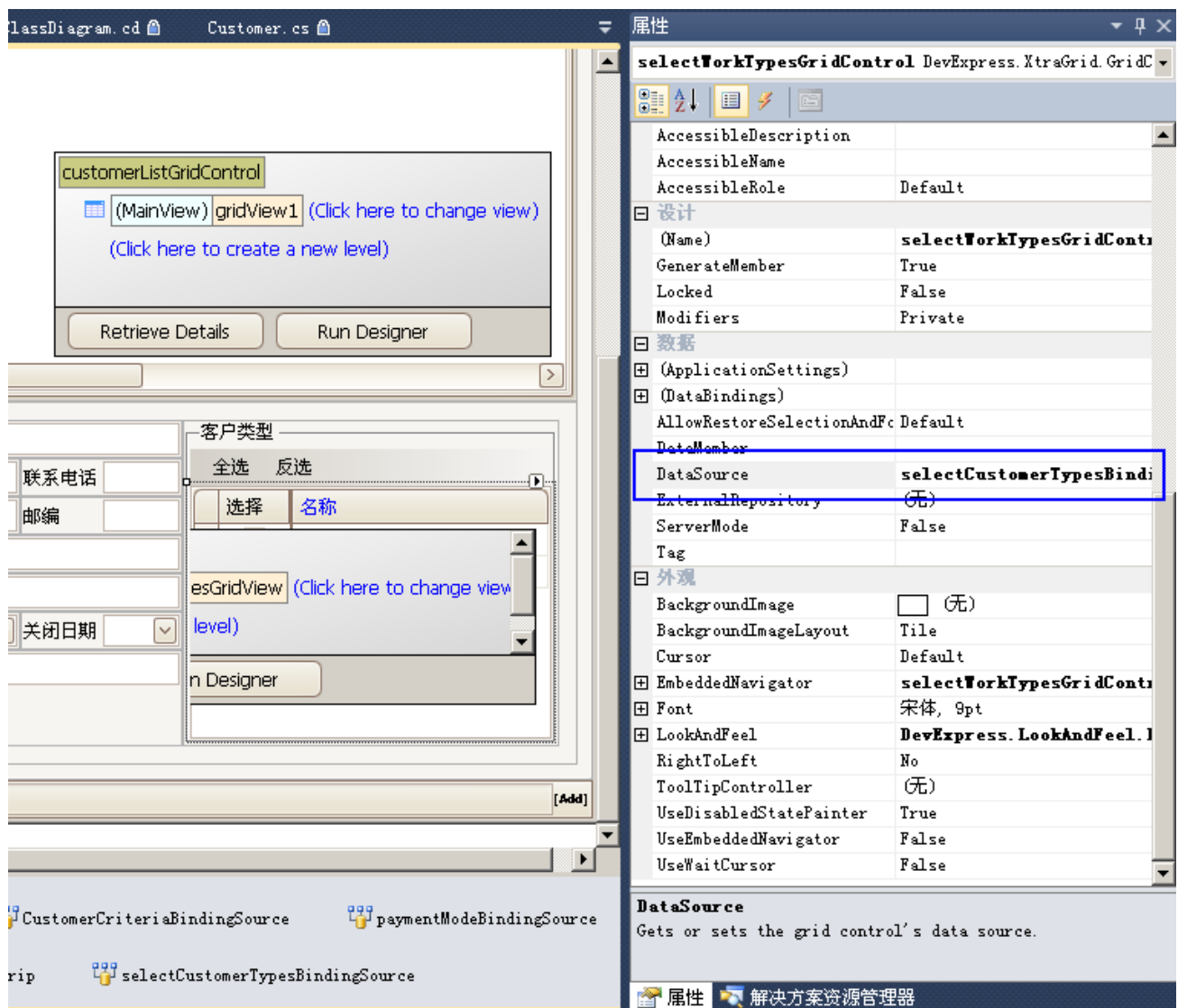
使用方法如下：

```
public CustomerTypeList SelectCustomerTypes
{
    get
    {
        //具备的客户类型
        CustomerTypeList customerTypes = GetCompositionDetail<CustomerTypeList, CustomerType>();
        //供挑选的客户类型全集
        return customerTypes.CollatingSelectableList(EnumKeyCaptionCollection.Fetch<CustomerType>());
    }
}
```

从业务集合对象 `customerTypes` 被隐藏在了主业务对象 `Customer` 中，由 Phenix 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 `SelectCustomerTypes` 属性。

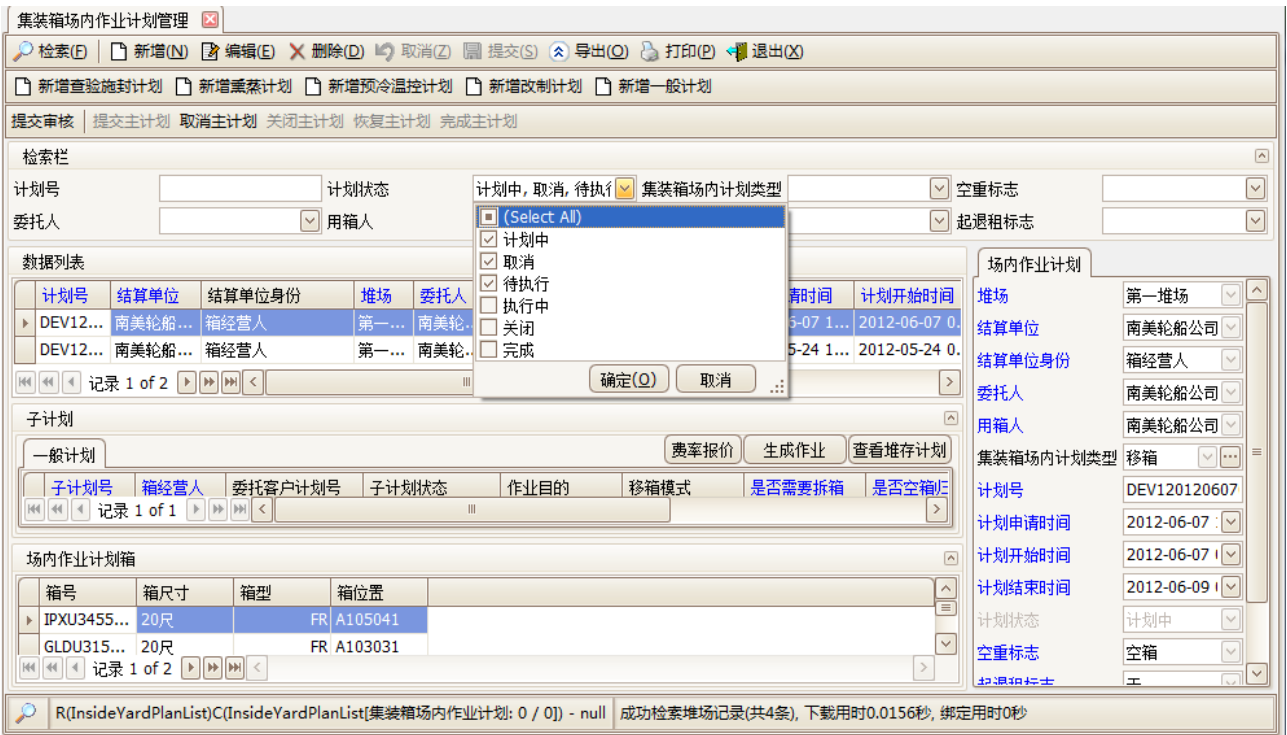
如此，上图界面设计的“客户类型”栏中清单的数据源，将绑定为 `Customer` 对象的 `SelectCustomerTypes` 属性：





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。

12.10.4 支持界面控件 CheckedComboBoxEdit

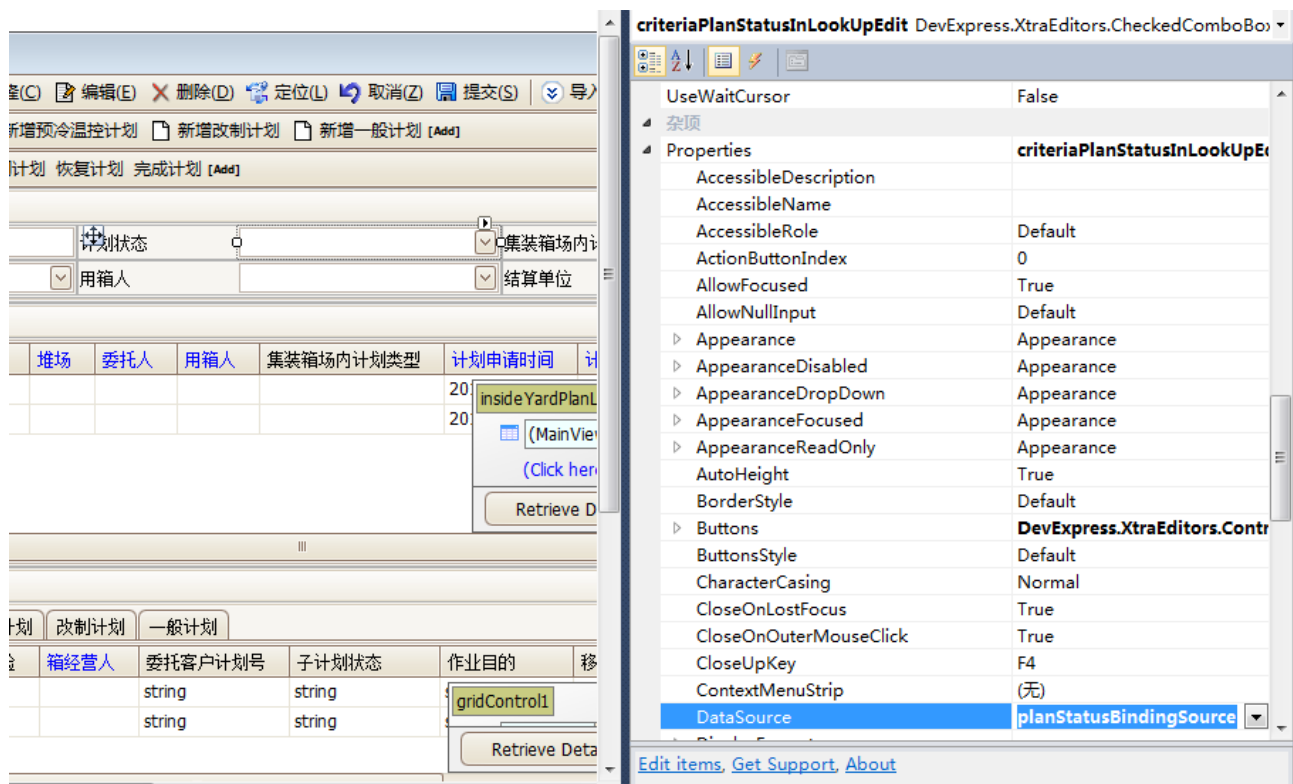


如上图业务场景中，要求检索栏中的“计划状态”条件是可以多选得到的，这就要用到 CheckedComboBoxEdit 控件（DEV 控件包，其他产品也有类似控件，用法类似），此时无法将这个控件直接绑定到普通的查询条件属性上：

```
/// <summary>
/// 计划状态
/// </summary>
[Phenix.Core.Mapping.CriteriaField(Operate = Phenix.Core.Mapping.CriteriaOperate.Equal,
Logical = Phenix.Core.Mapping.CriteriaLogical.And, FriendlyName = "计划状态", ColumnName =
"CYP_PLAN_STATUS_FG")]
private PlanStatus? _planStatus;
/// <summary>
/// 计划状态
/// </summary>
[System.ComponentModel.DisplayName("计划状态")]
public PlanStatus? PlanStatus
{
    get { return _planStatus; }
    set { _planStatus = value; }
}
```

那么，如何不用写界面控制层代码而又能实现多选效果呢？

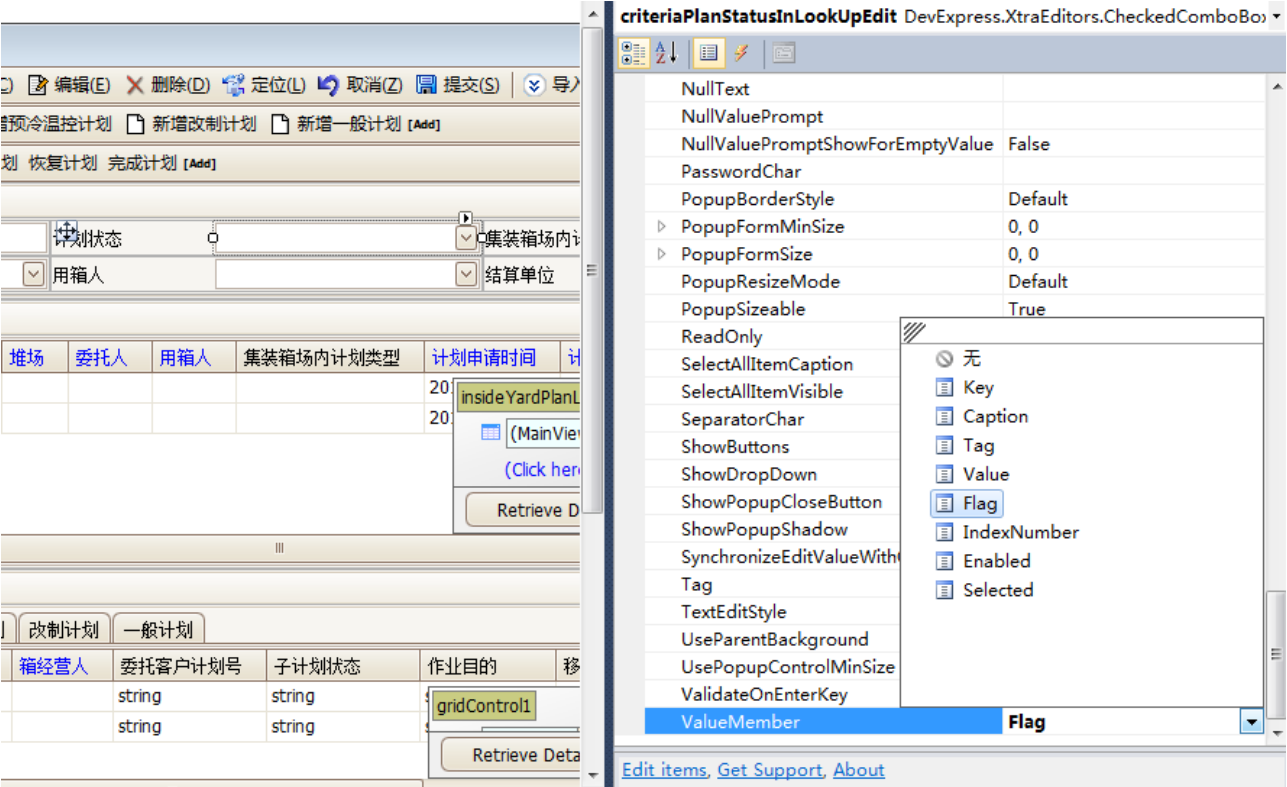
首先，我们把 CheckedComboBoxEdit 控件的下拉勾选框配置完成，这个配置方法是和 LookupEdit 等控件的配置方法差不多的，都是绑定到 EnumKeyCaptionCollection 数据源的 BindingSource 上：



然后在界面初始化的时候 Fetch 到这个 BindingSource 上：

```
planStatusBindingSource.DataSource = EnumKeyCaptionCollection.Fetch< PlanStatus >();
```

注意，在绑定下拉数据源的时候，ValueMember 应该是 EnumKeyCaption.Flag 属性：



也就是枚举的 Flag（整型）值。

但是，CheckedComboBoxEdit 控件绑定的查询类属性必须是字符串的，且这个字符串的内容是用“，”分隔的枚举 Flag.ToString()。

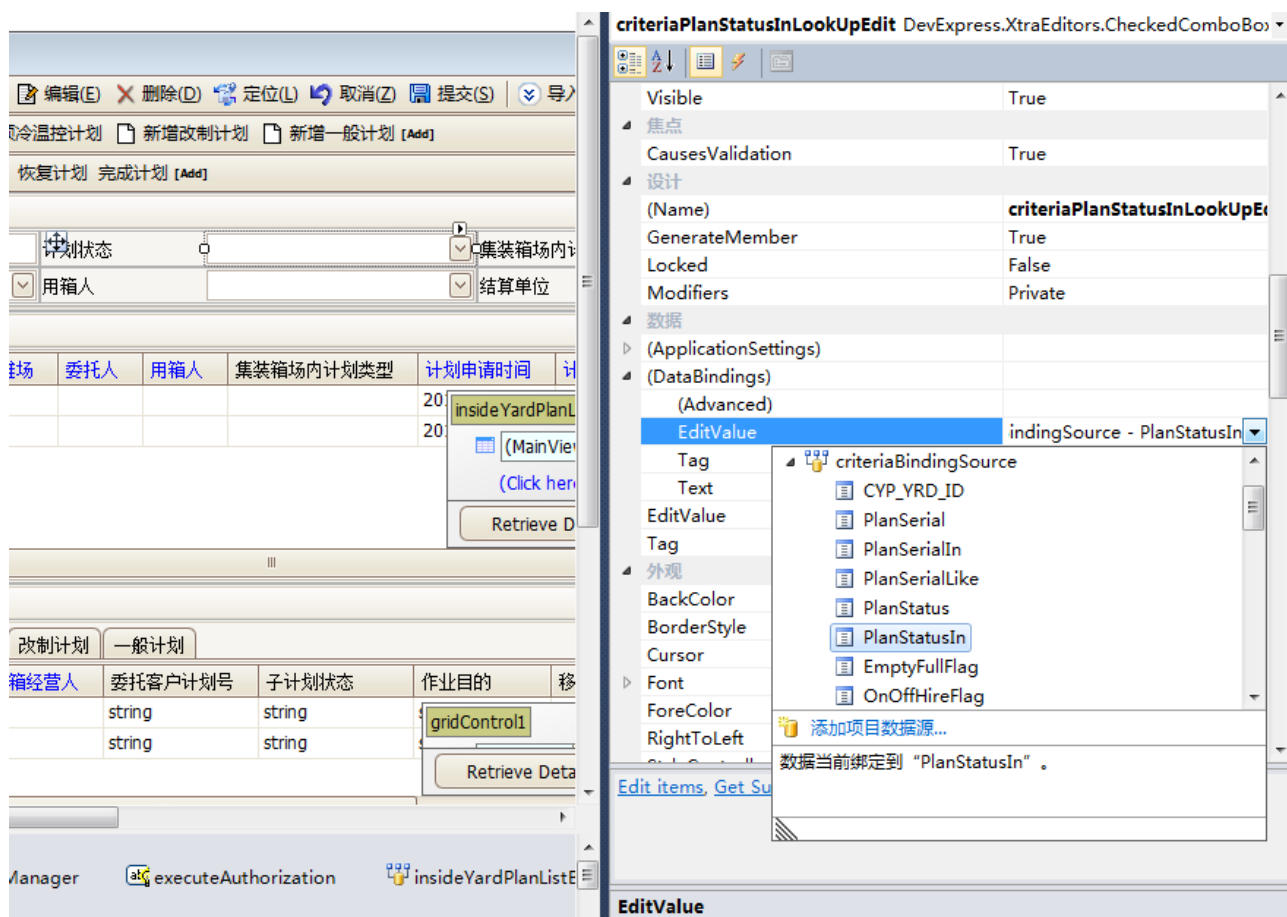
比如下列枚举，当字符串值为“0,1,2”时，逻辑值应该是：Planning | Cancel | WaitExecute:

```
[KeyCaption(FriendlyName = "计划状态")]
[Serializable]
public enum PlanStatus
{
    [EnumCaption("计划中")]
    Planning,
    [EnumCaption("取消")]
    Cancel,
    [EnumCaption("待执行")]
    WaitExecute,
    [EnumCaption("执行中")]
    Executing,
    [EnumCaption("关闭")]
    Closed,
    [EnumCaption("完成")]
    Completed,
}
```

要配合 CheckedComboBoxEdit 控件实现自动绑定，我们可以在查询类中添加下述类似的属性：

```
[CriteriaField(Operate = CriteriaOperate.In, Logical = CriterialLogical.And, FriendlyName = "计划状态", ColumnName = "CYP_PLAN_STATUS_FG")]
private PlanStatus[] _planStatusIn;
/// <summary>
/// 计划状态
/// </summary>
public string PlanStatusIn
{
    get { return Phenix.Core.Code.Converter.EnumArrayToFlags<PlanStatus>(_planStatusIn); }
    set { _planStatusIn = Phenix.Core.Code.Converter.FlagsToEnumArray<PlanStatus>(value); }
}
```

需提请注意，字段类型是枚举数组，而属性类型则是 string，这是为了能被绑定到 CheckedComboBoxEdit 控件上：



这样，当检索数据时，提交到数据库的 SQL 语句将类似于：

```
<Command Text="select
CYP_ID, CYP_YRD_ID, CYP_CONSIGNOR_CTI_ID, CYP_CTN_USER_CTI_ID, CYP_CYT_ID, CYP_PLAN_SERIAL, CYP_APPLY_TIME, CYP_
```

```
OPEN_TIME, CYP_CLOSE_TIME, CYP_PLAN_STATUS_FG, CYP_EMPTY_FULL_FLAG_FG, CYP_ON_OFF_HIRE_FLAG_FG, CYP_REMARK, CYP
_INPUTER, CYP_INPUTTIME, CYP_PLAN_CHECK_STATUS_FG, CYP_CHARGER_CTI_ID, CYP_CUSTOMER_IDENTITY_FG      from
CWP_INSIDE_YARD_PLAN where ( ( CWP_INSIDE_YARD_PLAN.CYP_PLAN_STATUS_FG in (0,1,2)) )" />
```

属性的 get、set 代码中将枚举数组类型和 string 类型做了互换，用到了 Phenix.Core.Code 类所提供的转换函数，汇总说明如下：

```
/// <summary>
/// 将枚举数组替换成标记组合
/// </summary>
/// <param name="enums">枚举数组</param>
public static string EnumArrayToFlags(params Enum[] enums)

/// <summary>
/// 将标记组合替换成枚举数组
/// </summary>
/// <param name="flags">标记组合</param>
public static TEnum[] FlagsToEnumArray<TEnum>(string flags)
```