11 业务对象生命周期及其状态

11.3 Fetch 业务对象

Fetch 出来的业务对象，与数据库中的相关表记录有一一对应的关系，在被持久化的时候，是以更新（update）、删除（delete）记录的方式被提交的。其特征是属性 IsNew = false。

11.3.1从数据库获取业务对象

11.3.1.1 Phenix.Business.BusinessBase<T>提供 Fetch 一个业务对象的函数

一般情况下，这些不同形式所传入的参数都是为 Fetch 方法提供提取表记录的主键值。如果提供的是非主键值，除非从业务逻辑上自己很清楚只能 Fetch 到一条记录，否则 Fetch 得到的业务对象将是不确定的。

```
/// <summary>
/// 按照指定主键值来获取对应的数据库记录构建业务对象
/// </summary>
/// <param name="primaryKeyValue">主键值</param>
public static T Fetch(long primaryKeyValue)

/// <summary>
/// 按照指定主键值来获取对应的数据库记录构建业务对象
/// </summary>
/// <param name="primaryKeyValue">主键值</param>
public static T Fetch(string primaryKeyValue)

/// <summary>
/// 按照指定主键/唯一键值来获取对应的数据库记录构建业务对象
/// </summary>
/// <param name="business">带主键/唯一键值的业务对象</param>
public static T Fetch(T business)

/// <summary>
/// 构建业务对象
/// 表中仅一条记录
/// 否则仅取表的第一条记录
/// </summary>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(params OrderByInfo[] orderByInfos)

/// <summary>
/// 构建业务对象
```

```
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criteria">条件对象</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(ICriteria criteria, params OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(Expression<Func<T, bool>> criteriaExpression, params OrderByInfo[]
orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(CriteriaExpression criteriaExpression, params OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="criterions">条件集</param>
    public static T Fetch(Criterions criterions)
```

11.3.1.2 Phenix.Business.BusinessListBase<T, TBusiness>提供 Fetch 一组业务对象集合的函数

```
    /// <summary>
    /// 构建业务对象集合
    /// </summary>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(params OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criteria">条件对象</param>
    /// <param name="cacheEnabled">是否需要缓存对象?</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(ICriteria criteria, bool cacheEnabled, bool lazyFetch, params OrderByInfo[]
orderByInfos)

    /// <summary>
    /// 构建业务对象集合
    /// </summary>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="cacheEnabled">可以缓存对象?</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(Expression<Func<TBusiness, bool>> criteriaExpression, bool cacheEnabled, bool
lazyFetch, params OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象集合
    /// </summary>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="cacheEnabled">可以缓存对象?</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(CriteriaExpression criteriaExpression, bool cacheEnabled, bool lazyFetch,
params OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象集合
    /// </summary>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public static T Fetch(Criterions criterions, bool lazyFetch)
```

11.3.1.3 Phenix.Business.BusinessBase<T>提供 Fetch 一个从业务对象的函数

一般情况下，这些不同形式所传入的参数都是为 Fetch 方法提供提取表记录的主键值。如果提供的是非主键值，除非从业务逻辑上自己很清楚只能 Fetch 到一条记录，否则 Fetch 得到的从业务对象将是不确定的。

通过 GetDetail()函数得到的从业务对象，都会被主业务对象缓存在本地，对它的编辑结果也会通过主业务对象的提交一起被持久化到数据库。

```
    /// <summary>
    /// 取从业务对象
    /// 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 GetDetail<TDetailBusiness>(Criterions criterions)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(ICriteria criteria, string groupName,
params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(ICriteria criteria, string groupName,
params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(CriteriaExpression criteriaExpression,
string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(CriteriaExpression criteriaExpression,
string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
   /// <param name="orderByInfos">数据排列顺序队列</param>
     public TDetailBusiness GetAggregationDetail<TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

除此之外，Phenix.Business.BusinessBase<T>还提供了带 DbConnection、DbTransaction 类型参数的 GetDetail()方法，以便于在服务端、与提交的主业务对象一个事务中处理数据：

```
    /// <summary>
    /// 取从业务对象
    /// 从业务对象与本业务对象是一对一的关系
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criterions">条件集</param>
    public TDetailBusiness GetDetail<TDetailBusiness>(DbConnection connection, Criterions criterions)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="connection">数据库连接</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbConnection connection, params
OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="connection">数据库连接</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbConnection connection, params
OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="connection">数据库连接</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbConnection connection, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="connection">数据库连接</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbConnection connection, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbConnection connection,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbConnection connection,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbConnection connection,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbConnection connection,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
```

```
orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象
    /// 从业务对象与本业务对象是一对一的关系
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criterions">条件集</param>
    public TDetailBusiness GetDetail<TDetailBusiness>(DbTransaction transaction, Criterions criterions)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="transaction">数据库事务</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbTransaction transaction, params
OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="transaction">数据库事务</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbTransaction transaction, params
OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="transaction">数据库事务</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbTransaction transaction, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(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="transaction">数据库事务</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbTransaction transaction, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbTransaction transaction,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbTransaction transaction,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(组合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetCompositionDetail<TDetailBusiness>(DbTransaction transaction,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象(聚合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetailBusiness GetAggregationDetail<TDetailBusiness>(DbTransaction transaction,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

11.3.1.4 Phenix.Business.BusinessBase<T>提供 Fetch 一组从业务对象集合的函数

通过 GetDetail()函数得到的从业务对象集合，都会被主业务对象缓存在本地，对它的编辑结果也会通过主业务对象的提交一起被持久化到数据库。

```
    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail GetDetail<TDetail, TDetailBusiness>(Criterions criterions, bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(ICriteria criteria, string groupName,
bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(ICriteria criteria, string groupName,
bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(CriteriaExpression criteriaExpression,
string groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(CriteriaExpression criteriaExpression,
string groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> criteriaExpression, string groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

这些函数主要用于在主业务对象内从业务对象属性值的获取，鉴于Phenix丷对这些对象做了缓存和自动处理，所以一般情况下无需自行再做缓存，直接return值即可：

```
    /// <summary>
    /// 程序集类信息
    /// </summary>
    public AssemblyClassInfoList AssemblyClassInfos
    {
      get { return GetCompositionDetail<AssemblyClassInfoList, AssemblyClassInfo>(); }
    }
```

可按照条件获取从业务对象：

```
        /// <summary>
        /// 未做起退租的箱信息
        /// </summary>
        public OutYardPlanContainerList OutYardPlanInnerContainers
        {
            get { return GetCompositionDetail<OutYardPlanContainerList,
OutYardPlanContainer>(OutYardPlanContainer.IsOutProperty == false); }
        }
        /// <summary>
        /// 已经起退租的箱信息
        /// </summary>
        public OutYardPlanContainerList OutYardPlanOutContainers
        {
            get { return GetCompositionDetail<OutYardPlanContainerList,
OutYardPlanContainer>(OutYardPlanContainer.IsOutProperty == true); }
        }
```

除此之外，Phenix.Business.BusinessBase<T>还提供了带 DbConnection、DbTransaction 类型参数的 GetDetail()方法，以便于在服务端、与提交的主业务对象一个事务中处理数据：

```
    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail GetDetail<TDetail, TDetailBusiness>(DbConnection connection, Criterions criterions,
bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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="connection">数据库连接</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbConnection connection, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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="connection">数据库连接</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbConnection connection, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbConnection connection,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbConnection connection,
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbConnection connection,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbConnection connection,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail GetDetail<TDetail, TDetailBusiness>(DbTransaction transaction, Criterions criterions,
bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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="transaction">数据库事务</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbTransaction transaction, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(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="transaction">数据库事务</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbTransaction transaction, ICriteria
criteria, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbTransaction transaction, 
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbTransaction transaction, 
CriteriaExpression criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(DbTransaction transaction, 
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[] 
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(DbTransaction transaction, 
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

11.3.1.5 Phenix.Business.BusinessListBase<T, TBusiness>提供 Fetch 一组从业务对象集合的函数

通过业务集合对象的 FetchDetail()函数得到其所有业务对象的从业务对象的集合（不会被缓存在本地），可与 BusinessBase 类中带(TDetail source)参数的 GetDetail()函数组合使用，将它们过滤进各主业务对象的 Detail 缓存中。这些 Detail 缓存的从业务集合对象，可以对它们进行编辑，并通过主业务对象的提交一起被持久化到数据库。

```
    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(Criterions criterions, bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// 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>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(ICriteria criteria, string groupName, bool
lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(CriteriaExpression criteriaExpression, string
groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness, bool>>
criteriaExpression, string groupName, bool lazyFetch, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbConnection connection, Criterions criterions,
bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbConnection connection, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbConnection connection, ICriteria criteria,
string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbConnection connection, CriteriaExpression
criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="connection">数据库连接</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbConnection connection,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criterions">条件集</param>
    /// <param name="lazyFetch">是否惰性Fetch(</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbTransaction transaction, Criterions
criterions, bool lazyFetch)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbTransaction transaction, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteria">从业务条件对象</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbTransaction transaction, ICriteria criteria,
string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbTransaction transaction, CriteriaExpression
criteriaExpression, string groupName, params OrderByInfo[] orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">从业务条件表达式</param>
    /// <param name="groupName">分组名</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public TDetail FetchDetail<TDetail, TDetailBusiness>(DbTransaction transaction,
Expression<Func<TDetailBusiness, bool>> criteriaExpression, string groupName, params OrderByInfo[]
orderByInfos)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

11.3.1.6 惰性加载

前文中 Fetch 或 GetDetail 业务集合对象的时候，都提供了惰性加载参数（lazyFetch = true），使用它，则只有当遍历这个业务对象集合的时候，业务数据才正式被加载到本地，从而提高了应用系统的响应能力。

11.3.1.6.1 惰性加载的状态

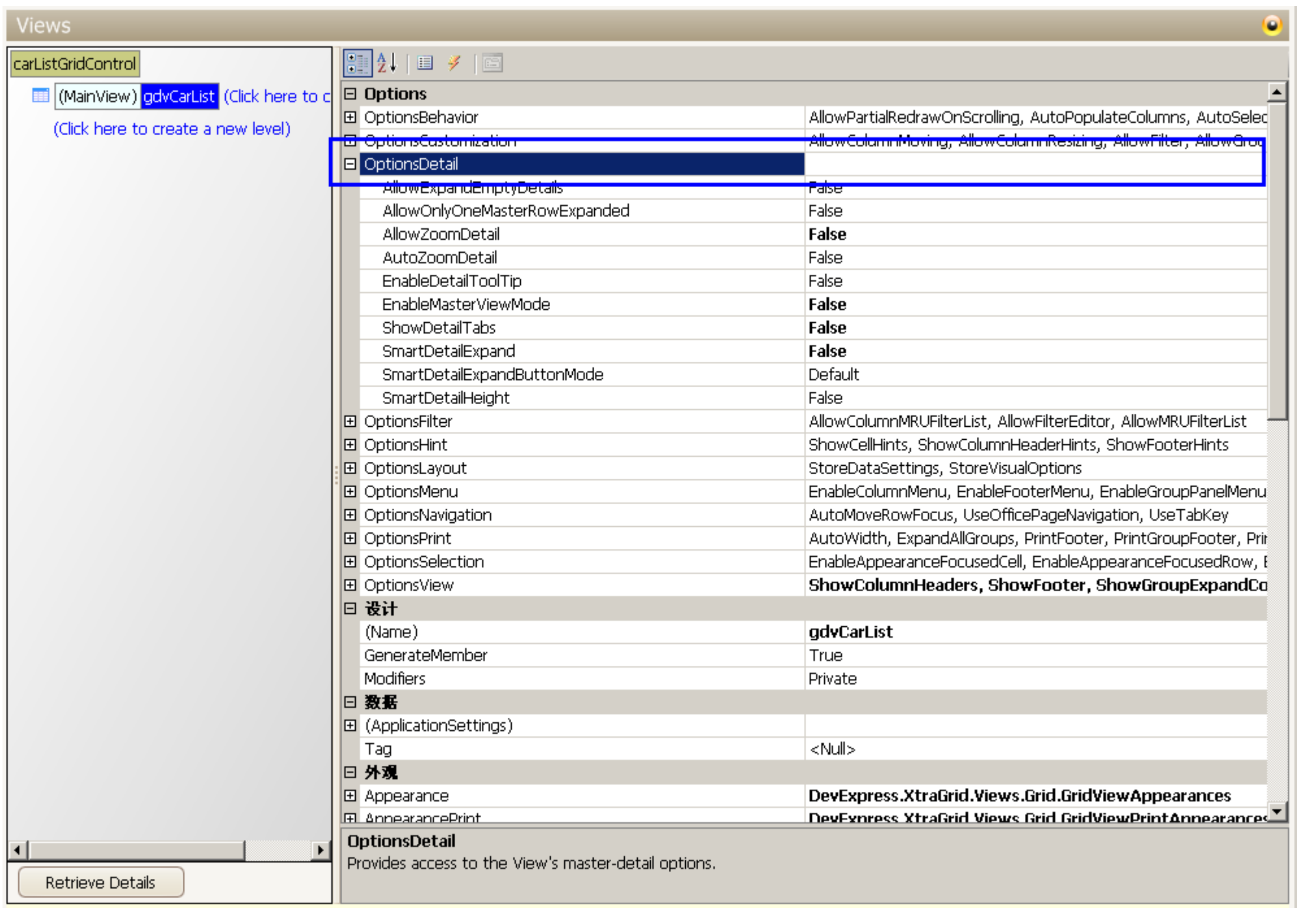
要知道业务对象集合当前是否处于惰性加载当中，可参考 Phenix.Business.BusinessListBase<T, TBusiness>属性：

属性	说明	备注
InLazyFetch	是否处于惰性 Fetch 中	Fetch 时不加载数据，仅当检索集合里的业务对象时才正式加载，加载完成后自动变为 false;

11.3.1.6.2 强制惰性加载从业务对象

在界面设计的时候，我们经常会将主业务集合对象绑定在Grid控件上，但由于从业务对象集合一般都设计成主业务对象上的一个属性，这样，如果不做特殊处理的话，从业务对象集合的属性会被界面控件遍历到，并触发它的 GetDetail()函数。主业务对象数量越多，从业务对象集合的属性被遍历到的次数也就越多，特别是业务结构嵌套得越深，性能将按照乘积倍数急剧下滑。

为了解决这类问题，在界面层设计上，我们可以考虑如下配置方法（针对 DevExpress.XtraGrid 控件）：


Views
carListGridControl
(MainView) gdvCarList (Click here to c
(Click here to create a new level)
Retrieve Details

⊟ Options	
⊞ OptionsBehavior	AllowPartialRedrawOnScrolling, AutoPopulateColumns, AutoSelec
⊞ OptionsCustomization	AllowColumnMoving, AllowColumnResizing, AllowFilter, AllowGrou
⊟ OptionsDetail	
AllowExpandEmptyDetails	False
AllowOnlyOneMasterRowExpanded	False
AllowZoomDetail	**False**
AutoZoomDetail	False
EnableDetailToolTip	False
EnableMasterViewMode	**False**
ShowDetailTabs	**False**
SmartDetailExpand	**False**
SmartDetailExpandButtonMode	Default
SmartDetailHeight	False
⊞ OptionsFilter	AllowColumnMRUFilterList, AllowFilterEditor, AllowMRUFilterList
⊞ OptionsHint	ShowCellHints, ShowColumnHeaderHints, ShowFooterHints
⊞ OptionsLayout	StoreDataSettings, StoreVisualOptions
⊞ OptionsMenu	EnableColumnMenu, EnableFooterMenu, EnableGroupPanelMenu
⊞ OptionsNavigation	AutoMoveRowFocus, UseOfficePageNavigation, UseTabKey
⊞ OptionsPrint	AutoWidth, ExpandAllGroups, PrintFooter, PrintGroupFooter, Pri
⊞ OptionsSelection	EnableAppearanceFocusedCell, EnableAppearanceFocusedRow, E
⊞ OptionsView	**ShowColumnHeaders, ShowFooter, ShowGroupExpandCo**
⊟ 设计	
(Name)	**gdvCarList**
GenerateMember	True
Modifiers	Private
⊟ 数据	
⊞ (ApplicationSettings)	
Tag	<Null>
⊟ 外观	
⊞ Appearance	**DevExpress.XtraGrid.Views.Grid.GridViewAppearances**
⊞ AppearancePrint	**DevExpress.XtraGrid.Views.Grid.GridViewPrintAppearances**

OptionsDetail
Provides access to the View's master-detail options.

如果不需要被绑定的话，也可以在从业务对象集合的属性上：

```
/// <summary>
/// 程序集类信息
/// </summary>
[System.ComponentModel.Browsable(false)]
[System.ComponentModel.DataAnnotations.Display(AutoGenerateField = false)]
public AssemblyClassInfoList AssemblyClassInfos
{
  get { return GetCompositionDetail<AssemblyClassInfoList, AssemblyClassInfo>(); }
}
```

但是，往往又需要被绑定，那么可以：

```
private bool DoPrint(BindingSource source)
{
  IBusinessCollection businessList = source.List as IBusinessCollection;
  bool itemLazyGetDetail = false;
  if (businessList != null)
  {
    itemLazyGetDetail = businessList.ItemLazyGetDetail;
    businessList.ItemLazyGetDetail = true;
  }
  try
  {
    GridControl gridControl = GridControlHelper.Find(Form, source);
    if (gridControl != null)
    {
      gridControl.ShowPrintPreview();
      return true;
    }
    return false;
  }
  finally
  {
    if (businessList != null)
      businessList.ItemLazyGetDetail = itemLazyGetDetail;
  }
}
```

Phenix.Business.BusinessListBase<T, TBusiness>提供了 ItemLazyGetDetail 属性，可以强制约束其业务对象采取惰性加载从业务对象（ItemLazyGetDetail = true）的方法：

属性	说明	备注
ItemLazyGetDetail	业务对象惰性 GetDetail	缺省为 false;

11.3.2从本地获取业务对象

如何高效合理地从服务端获取到数据是设计者必须考虑的问题。比如：对于业务结构复杂的数据集，是一次获取全部数据、还是分批获取数据，哪个更能提供高效的用户体验？

- 如果需要第一时间在客户端展现全部的业务结构数据，则应该一次获取全部数据；
- 如果在客户端的业务结构数据是层层展现的，则应该分批获取数据，用户体验或更好些；

当然，在需要下载海量业务数据的应用场景下，这两点建议就不适用了，而应该实现分页下载的机制。

除此之外，还有一种处理方式，就是类似于建设太空站的模式，先将零配件打包发送到太空，然后再在现场组装，这样传输成本是最低的，也减少了事前组装和传递过程中复杂业务结构的序列化/反序列化造成的消耗。

以下介绍的是打包下载数据后如何在本地组装、繁衍出新的、不同类型对象的方法。

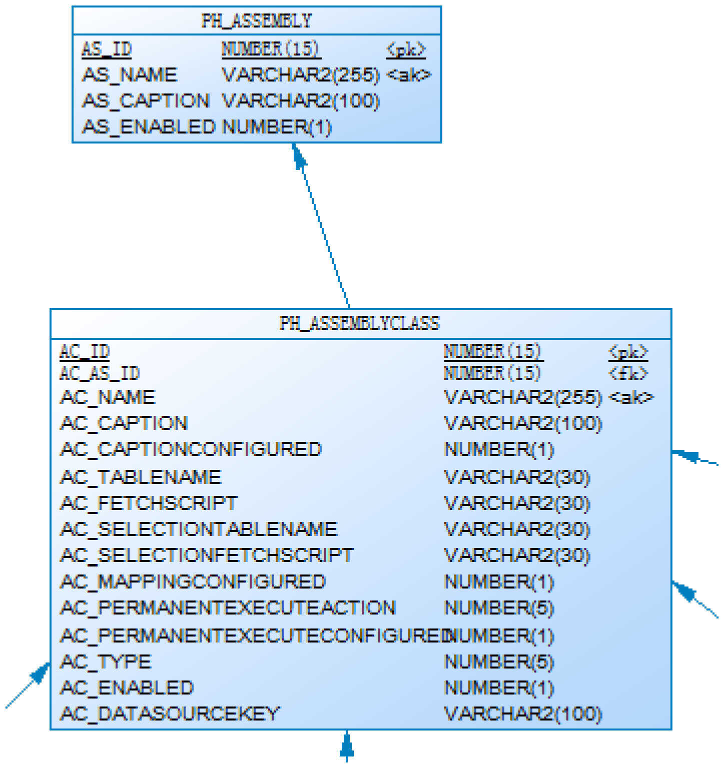
11.3.2.1 Phenix.Business.BusinessBase<T>提供从 source 业务对象中 Fetch 出另一种类的业务对象的函数

```
    /// <summary>
    /// 构建业务对象
    /// 按照数据源填充(指定属性或映射的表字段需一致)
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="propertyInfos">需匹配的属性信息，当为null、空队列时匹配全部属性, 映射的表字段(或主外键)需一致 </param>
    public static T Fetch(IBusinessObject source, IPropertyInfo[] propertyInfos)
```

以下案例是演示如何利用本功能，对一个映射视图的业务对象实现自动 Fetch 出主业务对象，并与这个视图对象（即从业务对象）一起提交更新到数据库的方法。

需注意的是，本案例并不是唯一的、也不是最优的处理多表关联数据的方法。

业务场景是，表“PH_ASSEMBLY”和表“ PH_ASSEMBLYCLASS”是主从结构，界面要求这两个表能展现在一个清单列表中，且需要象一张表一样同时进行编辑和提交。


PH_ASSEMBLY
AS_ID NUMBER(15) <pk>
AS_NAME VARCHAR2(255) <ak>
AS_CAPTION VARCHAR2(100)
AS_ENABLED NUMBER(1)
PH_ASSEMBLYCLASS
AC_ID NUMBER(15) <pk>
AC_AS_ID NUMBER(15) <fk>
AC_NAME VARCHAR2(255) <ak>
AC_CAPTION VARCHAR2(100)
AC_CAPTIONCONFIGURED NUMBER(1)
AC_TABLENAME VARCHAR2(30)
AC_FETCHSCRIPT VARCHAR2(30)
AC_SELECTIONTABLENAME VARCHAR2(30)
AC_SELECTIONFETCHSCRIPT VARCHAR2(30)
AC_MAPPINGCONFIGURED NUMBER(1)
AC_PERMANENTEXECUTEACTION NUMBER(5)
AC_PERMANENTEXECUTECONFIGURED NUMBER(1)
AC_TYPE NUMBER(5)
AC_ENABLED NUMBER(1)
AC_DATASOURCEKEY VARCHAR2(100)

11.3.2.1.1 映射视图的从业务类

首先，我们可以设计一个视图，将“PH_ASSEMBLYCLASS”、“PH_ASSEMBLY”两表关联：

```
  CREATE OR REPLACE VIEW PH_ASSEMBLYCLASSINFO AS
  SELECT AC_ID,
      AC_AS_ID,
      AC_NAME,
      AC_CAPTION,
      AC_FETCHSCRIPT,
      PH_ASSEMBLY.AS_NAME,
      PH_ASSEMBLY.AS_CAPTION
    FROM PH_ASSEMBLYCLASS
    JOIN PH_ASSEMBLY
    ON PH_ASSEMBLY.AS_ID = PH_ASSEMBLYCLASS.AC_AS_ID;
```

映射为业务类 AssemblyClassInfo：

```
  /// <summary>
  ///  程序集类信息
  /// </summary>
  [Serializable]
  public class AssemblyClassInfo : AssemblyClassInfo<AssemblyClassInfo>
  {
  }

  /// <summary>
  /// 程序集类信息清单
  /// </summary>
  [Serializable]
  public class AssemblyClassInfoList : Phenix.Business.BusinessListBase<AssemblyClassInfoList,
AssemblyClassInfo>
  {
  }

  /// <summary>
  /// 程序集类信息
  /// </summary>
  [Phenix.Core.Mapping.ClassAttribute("PH_ASSEMBLYCLASS", FetchScript = "PH_ASSEMBLYCLASSINFO",
FriendlyName = "程序集类信息"), System.SerializableAttribute(),
System.ComponentModel.DisplayNameAttribute("程序集类信息")]
  public abstract class AssemblyClassInfo<T> : Phenix.Business.BusinessBase<T> where T :
AssemblyClassInfo<T>
  {
    [System.ComponentModel.Browsable(false)]
    [System.ComponentModel.DataAnnotations.Display(AutoGenerateField = false)]
    public override string PrimaryKey
    {
      get { return String.Format("{0}", AC_ID); }
    }

    /// <summary>
    /// AC_ID
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<long?> AC_IDProperty = RegisterProperty<long?>(c
=> c.AC_ID);
    [Phenix.Core.Mapping.Field(FriendlyName = "AC_ID", TableName = "PH_ASSEMBLYCLASS", ColumnName =
"AC_ID", IsPrimaryKey = true, NeedUpdate = true)]
    private long? _AC_ID;
    /// <summary>
    /// AC_ID
    /// </summary>
    [System.ComponentModel.DisplayName("AC_ID")]
    public long? AC_ID
    {
      get { return GetProperty(AC_IDProperty, _AC_ID); }
      set { SetProperty(AC_IDProperty, ref _AC_ID, value); }
    }

    /// <summary>
    /// AC_AS_ID
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<long?> AC_AS_IDProperty =
RegisterProperty<long?>(c => c.AC_AS_ID);
    [Phenix.Core.Mapping.Field(FriendlyName = "AC_AS_ID", TableName = "PH_ASSEMBLYCLASS", ColumnName =
"AC_AS_ID", NeedUpdate = true)]
    private long? _AC_AS_ID;
    /// <summary>
    /// AC_AS_ID
    /// </summary>
    [System.ComponentModel.DisplayName("AC_AS_ID")]
    public long? AC_AS_ID
    {
      get { return GetProperty(AC_AS_IDProperty, _AC_AS_ID); }
      set { SetProperty(AC_AS_IDProperty, ref _AC_AS_ID, value); }
    }

    /// <summary>
    /// AC_NAME
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> NameProperty =
RegisterProperty<string>(c => c.Name);
    [Phenix.Core.Mapping.Field(FriendlyName = "AC_NAME", Alias = "AC_NAME", TableName =
"PH_ASSEMBLYCLASS", ColumnName = "AC_NAME", NeedUpdate = true, InLookUpColumn = true,
InLookUpColumnDisplay = true)]
```

```
    private string _name;
    /// <summary>
    /// AC_NAME
    /// </summary>
    [System.ComponentModel.DisplayName("AC_NAME")]
    public string Name
    {
      get { return GetProperty(NameProperty, _name); }
      set { SetProperty(NameProperty, ref _name, value); }
    }

    /// <summary>
    /// AC_CAPTION
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> CaptionProperty =
RegisterProperty<string>(c => c.Caption);
    [Phenix.Core.Mapping.Field(FriendlyName = "AC_CAPTION", Alias = "AC_CAPTION", TableName =
"PH_ASSEMBLYCLASS", ColumnName = "AC_CAPTION", NeedUpdate = true)]
    private string _caption;
    /// <summary>
    /// AC_CAPTION
    /// </summary>
    [System.ComponentModel.DisplayName("AC_CAPTION")]
    public string Caption
    {
      get { return GetProperty(CaptionProperty, _caption); }
      set { SetProperty(CaptionProperty, ref _caption, value); }
    }

    /// <summary>
    /// AC_FETCHSCRIPT
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> FetchscriptProperty =
RegisterProperty<string>(c => c.Fetchscript);
    [Phenix.Core.Mapping.Field(FriendlyName = "AC_FETCHSCRIPT", Alias = "AC_FETCHSCRIPT", TableName =
"PH_ASSEMBLYCLASS", ColumnName = "AC_FETCHSCRIPT", NeedUpdate = true)]
    private string _fetchscript;
    /// <summary>
    /// AC_FETCHSCRIPT
    /// </summary>
    [System.ComponentModel.DisplayName("AC_FETCHSCRIPT")]
    public string Fetchscript
    {
      get { return GetProperty(FetchscriptProperty, _fetchscript); }
      set { SetProperty(FetchscriptProperty, ref _fetchscript, value); }
    }
    /// <summary>
    /// AS_NAME
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> Name_Property =
RegisterProperty<string>(c => c.Name_);
    [Phenix.Core.Mapping.Field(FriendlyName = "AS_NAME", Alias = "AS_NAME", TableName = "PH_ASSEMBLY",
ColumnName = "AS_NAME")]
    private string _name_;
    /// <summary>
    /// AS_NAME
    /// </summary>
    [System.ComponentModel.DisplayName("AS_NAME")]
    public string Name_
    {
      get { return GetProperty(Name_Property, _name_); }
      set { SetProperty(Name_Property, ref _name_, value); }
    }

    /// <summary>
    /// AS_CAPTION
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> Caption_Property =
RegisterProperty<string>(c => c.Caption_);
    [Phenix.Core.Mapping.Field(FriendlyName = "AS_CAPTION", Alias = "AS_CAPTION", TableName =
"PH_ASSEMBLY", ColumnName = "AS_CAPTION")]
    private string _caption_;
    /// <summary>
    /// AS_CAPTION
    /// </summary>
    [System.ComponentModel.DisplayName("AS_CAPTION")]
    public string Caption_
    {
      get { return GetProperty(Caption_Property, _caption_); }
      set { SetProperty(Caption_Property, ref _caption_, value); }
    }
  }
}
```

11.3.2.1.2 映射表的主业务类

而业务类 AssemblyInfo，则从表“PH_ASSEMBLY”直接映射出来：

```
  /// <summary>
  ///  程序集
  /// </summary>
  [Serializable]
  public class AssemblyInfo : AssemblyInfo<AssemblyInfo>
  {
  }

  /// <summary>
  /// 程序集清单
  /// </summary>
  [Serializable]
  public class AssemblyInfoList : Phenix.Business.BusinessListBase<AssemblyInfoList, AssemblyInfo>
  {
  }

  /// <summary>
  /// 程序集
  /// </summary>
  [Phenix.Core.Mapping.ClassAttribute("PH_ASSEMBLY", FriendlyName = "程序集"),
System.SerializableAttribute(), System.ComponentModel.DisplayNameAttribute("程序集")]
  public abstract class AssemblyInfo<T> : Phenix.Business.BusinessBase<T> where T : AssemblyInfo<T>
  {
    /// <summary>
    /// AS_ID
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<long?> AS_IDProperty = RegisterProperty<long?>(c
=> c.AS_ID);
    [Phenix.Core.Mapping.Field(FriendlyName = "AS_ID", TableName = "PH_ASSEMBLY", ColumnName = "AS_ID",
IsPrimaryKey = true, NeedUpdate = true)]
    private long? _AS_ID;
    /// <summary>
    /// AS_ID
    /// </summary>
    [System.ComponentModel.DisplayName("AS_ID")]
    public long? AS_ID
    {
      get { return GetProperty(AS_IDProperty, _AS_ID); }
      internal set { SetProperty(AS_IDProperty, ref _AS_ID, value); }
    }

    [System.ComponentModel.Browsable(false)]
    [System.ComponentModel.DataAnnotations.Display(AutoGenerateField = false)]
    public override string PrimaryKey
    {
      get { return String.Format("{0}", AS_ID); }
    }

    /// <summary>
    /// AS_NAME
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> NameProperty =
RegisterProperty<string>(c => c.Name);
    [Phenix.Core.Mapping.Field(FriendlyName = "AS_NAME", Alias = "AS_NAME", TableName = "PH_ASSEMBLY",
ColumnName = "AS_NAME", NeedUpdate = true, InLookUpColumn = true, InLookUpColumnDisplay = true)]
    private string _name;
    /// <summary>
    /// AS_NAME
    /// </summary>
    [System.ComponentModel.DisplayName("AS_NAME")]
    public string Name
    {
      get { return GetProperty(NameProperty, _name); }
      set { SetProperty(NameProperty, ref _name, value); }
    }

    /// <summary>
    /// AS_CAPTION
    /// </summary>
    public static readonly Phenix.Business.PropertyInfo<string> CaptionProperty =
RegisterProperty<string>(c => c.Caption);
    [Phenix.Core.Mapping.Field(FriendlyName = "AS_CAPTION", Alias = "AS_CAPTION", TableName =
"PH_ASSEMBLY", ColumnName = "AS_CAPTION", NeedUpdate = true)]
    private string _caption;
    /// <summary>
    /// AS_CAPTION
    /// </summary>
    [System.ComponentModel.DisplayName("AS_CAPTION")]
    public string Caption
    {
      get { return GetProperty(CaptionProperty, _caption); }
      set { SetProperty(CaptionProperty, ref _caption, value); }
    }
  }
}
```

11.3.2.1.3 从从业务对象中 Fetch 主业务对象

AssemblyClassInfo 由于含有 AssemblyInfo 的信息，所以可以将它作为 Fetch 出 AssemblyInfo 业务对象的母体：

```
  /// <summary>
  ///  程序集类信息
  /// </summary>
```

```
 [Serializable]
 public class AssemblyClassInfo : AssemblyClassInfo<AssemblyClassInfo>
 {
   [NonSerialized]
   [Csla.NotUndoable]
   private AssemblyInfo _linkAssemblyInfo;
   /// <summary>
   /// 关联的程序集
   /// </summary>
   public AssemblyInfo LinkAssemblyInfo
   {
     get
     {
       if (_linkAssemblyInfo == null)
         _linkAssemblyInfo = AssemblyInfo.Fetch(this);
       return _linkAssemblyInfo;
     }
   }
 }
```

11.3.2.1.4 业务类中包含有业务类字段的处理方法

在业务类的编写过程中，注意字段是否需要：

- 被回滚机制管理？不需要的话（如果本身已定义为业务类的 Detail、Link 对象，则是必须的），应该打上 Csla.NotUndoableAttribute 标签；
- 跨物理域传递？不需要的话，应该打上 System.NonSerializedAttribute 标签；

否则，应用系统的性能会受到影响。

在本业务场景里，LinkAssemblyInfo 是需要和 AssemblyClassInfo 一起被编辑和提交的，所以这两个标签都不需要打。

11.3.2.1.5 同时提交附带的业务对象

由于业务类仅支持单表（由 Phenix.Core.Mapping.ClassAttribute.TableName 指定）提交，AssemblyClassInfo 提交的是“PH_ASSEMBLYCLASS”表，所以我们利用 LinkAssemblyInfo 来更新“PH_ASSEMBLY”表。方法是：AssemblyClassInfo 里那些与 AssemblyInfo 相关的属性值发生变更时都将被同步到属性LinkAssemblyInfo对象的相关属性上，在提交AssemblyClassInfo业务对象的时候，属性 LinkAssemblyInfo 对象会一起被传递到服务端，然后我们可以通过覆写 AssemblyClassInfo 对象的 OnUpdatingSelf 函数，将属性 LinkAssemblyInfo 对象的字段内容更新到主表上。

完整代码如下：

```
/// <summary>
///  程序集类信息
/// </summary>
[Serializable]
public class AssemblyClassInfo : AssemblyClassInfo<AssemblyClassInfo>
{
  //[NonSerialized]
  //[Csla.NotUndoable]
  private AssemblyInfo _linkAssemblyInfo;
  /// <summary>
  /// 关联的程序集
  /// </summary>
  public AssemblyInfo LinkAssemblyInfo
  {
    get
    {
      if (_linkAssemblyInfo == null)
        _linkAssemblyInfo = AssemblyInfo.Fetch(this);
      return _linkAssemblyInfo;
    }
  }

  /// <summary>
  /// AS_NAME
  /// </summary>
  public override string Name_
  {
    get { return base.Name_; }
    set
    {
      base.Name_ = value;
      LinkAssemblyInfo.Name = value;
    }
  }

  /// <summary>
  /// AS_CAPTION
  /// </summary>
  public override string Caption_
  {
    get { return base.Caption_; }
    set
    {
      base.Caption_ = value;
      LinkAssemblyInfo.Caption = value;
    }
    }

    /// <summary>
    /// 更新本对象集合之前
    /// 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="limitingCriteriaExpressions">限制保存的条件(not exists 条件语句)</param>
    protected override void OnUpdatingSelf(DbTransaction transaction, ref List<CriteriaExpression>
limitingCriteriaExpressions)
    {
      if (_linkAssemblyInfo != null && _linkAssemblyInfo.IsDirty)
        _linkAssemblyInfo.Save(transaction);
    }
  }
```

11.3.2.2 Phenix.Business.BusinessBase<T>提供从 source 业务对象集合中 Fetch 出从业务对象集合的函数

```
    /// <summary>
    /// 取从业务对象集合(组合关系)
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="source">数据源</param>
    /// <param name="groupName">分组名</param>
    public TDetail GetCompositionDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> expression, TDetail source, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>

    /// <summary>
    /// 取从业务对象集合(聚合关系)
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="source">数据源</param>
    /// <param name="groupName">分组名</param>
    public TDetail GetAggregationDetail<TDetail, TDetailBusiness>(Expression<Func<TDetailBusiness,
bool>> expression, TDetail source, string groupName)
      where TDetail : BusinessListBase<TDetail, TDetailBusiness>
      where TDetailBusiness : BusinessBase<TDetailBusiness>
```

通过GetDetail()函数得到的从业务对象集合，都会被主业务对象缓存在本地，对它的编辑结果也会通过主业务对象的提交一起被持久化到数据库。

由于 Fech 出来的业务对象和 source 业务对象集合中的业务对象是同一个对象，所以需注意编辑操

作过程中的层级关系。

举例如下：

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

  /// <summary>
  /// 类信息
  /// </summary>
  public AssemblyClassList AssemblyClasses
  {
    get { return GetCompositionDetail<AssemblyClassList, AssemblyClass>(AssemblyClassList.Fetch()); }
  }

  #endregion
}
```

11.3.2.3 Phenix.Business.BusinessListBase<T, TBusiness>提供从 source 业务对象集合中 Fetch 出新的业务对象集合的函数

```
  /// <summary>
  /// 构建业务对象集合
  /// </summary>
  /// <param name="source">数据源</param>
  public static T Fetch(IEnumerable<TBusiness> source)
```

如果 source 与返回的新业务对象集合是属于相同的类型（业务对象集合类），则 source 的检索条件信息 Criterions 也会被赋到新业务对象集合上。

由于 Fech 出来的业务对象和 source 业务对象集合中的业务对象是同一个对象，所以需注意编辑操作过程中的层级关系。

11.3.2.4 Phenix.Business.BusinessListBase<T, TBusiness>提供根据 masterBusiness 主业务对象过滤出它的从业务对象集合的函数

```
  /// <summary>
  /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(组合关系)
```

```
    /// cascadingSave = true
    /// cascadingDelete = true
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusiness">主业务对象</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public T CompositionFilter(Expression<Func<TBusiness, bool>> expression, IBusinessObject
masterBusiness, string groupName)

    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(聚合关系)
    /// cascadingSave = true
    /// cascadingDelete = false
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusiness">主业务对象</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public T AggregationFilter(Expression<Func<TBusiness, bool>> expression, IBusinessObject
masterBusiness, string groupName)
```

也提供了一组静态方法从IEnumerable<TBusiness> 类型的source中过滤出从业务对象集合:

```
    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusiness">主业务对象</param>
    /// <param name="groupName">分组名, null代表全部</param>
    /// <param name="cascadingSave">是否级联保存?</param>
    /// <param name="cascadingDelete">是否级联删除?</param>
    public static T Filter(IEnumerable<TBusiness> source, Expression<Func<TBusiness, bool>> expression,
IBusinessObject masterBusiness, string groupName, bool cascadingSave, bool cascadingDelete)

    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(组合关系)
    /// cascadingSave = true
    /// cascadingDelete = true
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusiness">主业务对象</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public static T CompositionFilter(IEnumerable<TBusiness> source, Expression<Func<TBusiness, bool>>
expression, IBusinessObject masterBusiness, string groupName)
```

```
    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(聚合关系)
    /// cascadingSave = true
    /// cascadingDelete = false
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusiness">主业务对象</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public static T AggregationFilter(IEnumerable<TBusiness> source, Expression<Func<TBusiness, bool>>
expression, IBusinessObject masterBusiness, string groupName)
```

举例如下：

```
  /// <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)
          result = AssemblyClassList.Fetch().CompositionFilter(Owner);
        return result;
      }
    }

    #endregion
  }
```

我们发现这个例子其实就是对前一个例子的改写而已，业务类的 GetDetail()函数封装了业务集合类 Filter()函数的功能，所得到的结果是一样的。

由于 Filter 出来的业务对象和 source 业务对象集合中的业务对象不是同一个对象（而是 Clone 出来），所以无需注意编辑操作过程中的层级关系。如需 Item 是同一个对象的话，建议从 source 里 Linq

出 AddRange 到新的业务对象集合里。

11.3.2.5 Phenix.Business.BusinessListBase<T, TBusiness>提供根据 masterBusinesses 主业务对象集合过滤出它的从业务对象集合的函数

```
    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(组合关系)
    /// cascadingSave = true
    /// cascadingDelete = true
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusinesses">主业务对象集合</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public IDictionary<IBusinessObject, T> CompositionFilter(Expression<Func<TBusiness, bool>>
expression, IBusinessCollection masterBusinesses, string groupName)

    /// <summary>
    /// 过滤(克隆)出符合条件的、主业务对象的主键值和业务对象的外键值相等的业务对象集合(聚合关系)
    /// cascadingSave = true
    /// cascadingDelete = false
    /// </summary>
    /// <param name="expression">条件表达式</param>
    /// <param name="masterBusinesses">主业务对象集合</param>
    /// <param name="groupName">分组名, null代表全部</param>
    public IDictionary<IBusinessObject, T> AggregationFilter(Expression<Func<TBusiness, bool>>
expression, IBusinessCollection masterBusinesses, string groupName)
```

举例如下：

```
  /// <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
  }
```

可以实现前两个方法一样的结果。

由于 Filter 出来的业务对象和 source 业务对象集合中的业务对象不是同一个对象（而是 Clone 出来），所以无需注意编辑操作过程中的层级关系。如需 Item 是同一个对象的话，建议从 source 里 Linq 出 AddRange 到新的业务对象集合里。

11.3.2.6 Phenix.Business.BusinessListBase<T, TBusiness>提供按条件表达式过滤出业务对象集合的函数

```
    /// <summary>
    /// 过滤(克隆)出符合条件的业务对象队列
    /// </summary>
    /// <param name="expression">条件表达式</param>
    public T Filter(Expression<Func<TBusiness, bool>> expression)

    /// <summary>
    /// 过滤(克隆)出符合条件的业务对象队列
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="expression">条件表达式</param>
    public static T Filter(IEnumerable<TBusiness> source, Expression<Func<TBusiness, bool>> expression)
```

由于 Filter 出来的业务对象和 source 业务对象集合中的业务对象不是同一个对象（而是 Clone 出来），所以无需注意编辑操作过程中的层级关系。如需 Item 是同一个对象的话，建议从 source 里 Linq 出 AddRange 到新的业务对象集合里。

11.3.2.7 Phenix.Business.BusinessListBase<T, TBusiness>提供的排序函数 Fetch 出新的业务对象集合的函数

```
    /// <summary>
    /// 排序(克隆)
    /// </summary>
    /// <param name="orderByInfo">数据排列顺序</param>
    /// <param name="comparer">比较方法</param>
    public T OrderBy<TKey>(OrderByInfo orderByInfo, IComparer<TKey> comparer)

    /// <summary>
    /// 排序(克隆)
    /// </summary>
    /// <param name="source">数据源</param>
    /// <param name="orderByInfo">数据排列顺序</param>
    /// <param name="comparer">比较方法</param>
    public static T OrderBy<TKey>(IEnumerable<TBusiness> source, OrderByInfo orderByInfo,
IComparer<TKey> comparer)
```

由于 Order 出来的业务对象和 source 业务对象集合中的业务对象不是同一个对象（而是 Clone 出来），所以无需注意编辑操作过程中的层级关系。如需 Item 是同一个对象的话，建议从 source 里 Linq 出 AddRange 到新的业务对象集合里。

11.3.2.8 Phenix.Business.BusinessListBase<T, TBusiness>提供按条件表达式过滤自身业务对象集合的函数

```
    /// <summary>
    /// 从队列中(非删除)过滤剔掉符合条件的业务对象
    /// </summary>
    /// <param name="expression">条件表达式</param>
        public IList<TBusiness> FilterSelf(Expression<Func<TBusiness, bool>> expression)
```

本函数一般用于已 Fetch 到本地的业务对象集合再次清理以缩小条件范围。符合条件表达式的 Item 会被剔出本业务对象集合，完成后函数返回被剔除出来的业务对象。这些被剔除出去的业务对象将不会被本业务对象集合管理上，函数执行前后如有对这些业务对象有操作过，需自行负责其生命周期。

11.3.3嵌入到现有事务中获取业务对象的方法，不可跨物理域

在事务内获取到的数据和事务外获取到的数据，有时候会不一样的，比如在事务内更改了的数据只能在本事务内看到。如果在事务内的逻辑处理过程中，使用了非本事务获取到的数据，往往会发生意想不到的隐含错误，很难调试、也很难被及时发现。因此，如无特殊应用场景要求，必须在这种逻辑处理代码中使用带有 DbTransaction 参数的 Fetch()函数。

11.3.3.1 Phenix.Business.BusinessBase<T>提供 Fetch 一个业务对象的函数

```
    /// <summary>
    /// 按照指定主键值来获取对应的数据库记录构建业务对象
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="primaryKeyValue">主键值</param>
    public static T Fetch<TPrimaryKeyValue>(DbTransaction transaction, TPrimaryKeyValue primaryKeyValue)
    /// <summary>
    /// 按照指定主键/唯一键值来获取对应的数据库记录构建业务对象
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="business">带主键/唯一键值的业务对象</param>
    public static T Fetch(DbTransaction transaction, T business)

    /// <summary>
    /// 构建业务对象
    /// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteria">条件对象</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(DbTransaction transaction, ICriteria criteria, params OrderByInfo[]
orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(DbTransaction transaction, Expression<Func<T, bool>> criteriaExpression, params
OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criteriaExpression">条件表达式</param>
    /// <param name="orderByInfos">数据排列顺序队列</param>
    public static T Fetch(DbTransaction transaction, CriteriaExpression criteriaExpression, params
OrderByInfo[] orderByInfos)

    /// <summary>
    /// 构建业务对象
    /// </summary>
    /// <param name="transaction">数据库事务</param>
    /// <param name="criterions">条件集</param>
    public static T Fetch(DbTransaction transaction, Criterions criterions)
```

11.3.3.2 Phenix.Business.BusinessListBase<T, TBusiness>提供 Fetch 一组业务对象的函数

```
/// <summary>
/// 构建业务对象集合
/// </summary>
/// <param name="transaction">数据库事务</param>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(DbTransaction transaction, params OrderByInfo[] orderByInfos)

/// <summary>
/// 构建业务对象集合
/// 条件类的字段映射关系请用Phenix.Core.Mapping.CriteriaFieldAttribute标注
/// </summary>
/// <param name="transaction">数据库事务</param>
/// <param name="criteria">条件对象</param>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(DbTransaction transaction, ICriteria criteria, params OrderByInfo[]
orderByInfos)

/// <summary>
/// 构建业务对象集合
/// </summary>
/// <param name="transaction">数据库事务</param>
/// <param name="criteriaExpression">条件表达式</param>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(DbTransaction transaction, Expression<Func<TBusiness, bool>>
criteriaExpression, params OrderByInfo[] orderByInfos)

/// <summary>
/// 构建业务对象集合
/// </summary>
/// <param name="transaction">数据库事务</param>
/// <param name="criteriaExpression">条件表达式</param>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(DbTransaction transaction, CriteriaExpression criteriaExpression, params
OrderByInfo[] orderByInfos)

/// <summary>
/// 构建业务对象集合
/// </summary>
/// <param name="transaction">数据库事务</param>
/// <param name="criterions">条件集</param>
public static T Fetch(DbTransaction transaction, Criterions criterions)
```