11 业务对象生命周期及其状态

11.4 Delete 业务对象

指业务对象被打上删除标志（IsDeleted = true），并不是指从内存中释放掉这个对象的存储空间，而是当它被提交到数据库的时候，相应的表记录会被删除。

11.4.1Phenix.Business.BusinessBase<T>提供自己删除自己的函数

如果业务对象是属于一个业务对象集合的，则应该由业务对象集合负责删除它，而不应该使用下述函数自己删除自己。

```
/// <summary>
/// Marks the object for deletion. The object will be deleted as part of the
/// next save operation.
/// </summary>
/// <remarks>
/// <para>
/// CSLA .NET supports both immediate and deferred deletion of objects. This
/// method is part of the support for deferred deletion, where an object
/// can be marked for deletion, but isn't actually deleted until the object
/// is saved to the database. This method is called by the UI developer to
/// mark the object for deletion.
/// </para><para>
/// To 'undelete' an object, use n-level undo as discussed in Chapters 2 and 3.
/// </para>
/// </remarks>
public virtual void Delete()
```

11.4.2Phenix.Business.BusinessListBase<T, TBusiness>提供删除业务对象项的函数

```
/// <summary>
/// 移除业务对象
/// </summary>
public void RemoveAt(int index)

/// <summary>
/// 移除业务对象
/// </summary>
public bool Remove(T item);

/// <summary>
/// 按照条件移除业务对象
/// </summary>
```

```
/// <param name="match">定义要移除的元素应满足的条件</param>
public int RemoveItems(Predicate<TBusiness> match)


/// <summary>
/// 移除业务对象
/// </summary>
/// <param name="isSelected">是被选择的</param>
public int RemoveItems(bool isSelected)
```

11.4.3禁止 Phenix.Business.BusinessListBase<T, TBusiness>内的业务对象自己删除自己

CSLA 对集合内业务对象的删除功能做了限制，以下摘录自 Csla.Core.BusinessBase：

```
/// <summary>
/// Marks the object for deletion. The object will be deleted as part of the
/// next save operation.
/// </summary>
/// <remarks>
/// <para>
/// CSLA .NET supports both immediate and deferred deletion of objects. This
/// method is part of the support for deferred deletion, where an object
/// can be marked for deletion, but isn't actually deleted until the object
/// is saved to the database. This method is called by the UI developer to
/// mark the object for deletion.
/// </para><para>
/// To 'undelete' an object, use n-level undo as discussed in Chapters 2 and 3.
/// </para>
/// </remarks>
public virtual void Delete()
{
  if (this.IsChild)
    throw new NotSupportedException(Resources.ChildDeleteException);

  MarkDeleted();
}
```

当写出以下代码的时候：

```
            if (DeliveryPlan != null && DeliveryPlan.IsDirty)
            {
                //将按实际拣货的计划货物,但在拣货货物中没有使用到的数据删除
```

```
                foreach (var planDetail in DeliveryPlan.DeliveryPlanDetails)
                {
                  foreach (var planGoods in planDetail.DeliveryPlanGoodsViews)
                    {
                        if (planGoods.GoodsPickingMode == GoodsPickingMode.ByPicking ||
planGoods.PlanCount == 0)
                        {
                            if (DeliveryPickingGoodsList.GetRecordCount(transaction,
                                DeliveryPickingGoods.DPG_DPG_IDProperty == planGoods.DPG_ID
                                & DeliveryPickingGoods.DPG_DJT_IDProperty != DJT_ID) == 0)
                            {
                                if (!DeliveryPickingGoodsList.Any(p => p.DPG_DPG_ID ==
planGoods.DPG_ID))
                                {
                                    planGoods.Delete();
                                }
                            }
                        }
                    }
                }
                DeliveryPlan.Save(transaction);
            }
```

程序运行时是会抛出异常的：

2012-10-29 09:19:24 Phenix.Business.BusinessListBase`2.Save[SHB.Component.Warehouse.DeliveryWork.Business.DeliveryJobTicketList] : Can not directly mark a child object for deletion - use its parent collection(System.NotSupportedException)

2012-10-29 09:21:26 Phenix.Business.BusinessListBase`2.xe7ebae0df474ce49[SHB.Component.Warehouse.DeliveryWork.Business.DeliveryJobTicketList.0] : Edit level mismatch in AcceptChanges(Csla.Core.UndoException)

正确的写法是：

```
            if (DeliveryPlan != null && DeliveryPlan.IsDirty)
            {
                //将按实际拣货的计划货物,但在拣货货物中没有使用到的数据删除
              foreach (var planDetail in DeliveryPlan.DeliveryPlanDetails)
              {
                for (int i = planDetail.DeliveryPlanGoodsViews.Count - 1; i >= 0; i--)
                {
                  var planGoods = planDetail.DeliveryPlanGoodsViews[i];
```

```
                    if (planGoods.GoodsPickingMode == GoodsPickingMode.ByPicking ||
planGoods.PlanCount == 0)
                    {
                      if (DeliveryPickingGoodsList.GetRecordCount(transaction,
                        DeliveryPickingGoods.DPG_DPG_IDProperty == planGoods.DPG_ID
                          & DeliveryPickingGoods.DPG_DJT_IDProperty != DJT_ID) == 0)
                      {
                        if (!DeliveryPickingGoodsList.Any(p => p.DPG_DPG_ID == planGoods.DPG_ID))
                        {
                          planDetail.DeliveryPlanGoodsViews.RemoveAt(i);
                        }
                      }
                    }
                  }
                }
                DeliveryPlan.Save(transaction);
              }
```

CSLA 为什么禁止在集合中的业务对象调用 Delete()函数，而强制必须由集合对象来删除自己？是考虑到在集合中的业务对象，如果通过调用 Delete()函数仅标记自己 IsSelfDeleted = true 而不从集合中剔除自己的话，类似集合的 Count 属性等接口会产生二义性；另一方面，在业务逻辑代码里，要是在集合中删除某些符合条件的业务对象，往往会循环访问集合，这样要求通过调用业务对象 Delete()函数自动来从集合中剔除自己的话（就是要求 Delete()函数做到“找到自己所属集合并从中剔除自己”），框架并不知道这个 Delete()函数是嵌在 foreach 语句中还是 for 语句中、for 语句是递增的还是递减的，而我们知道“foreach 语句不应用于更改集合内容，以避免产生不可预知的副作用。”、在递增或递减的 for 语句中更改集合内容效果是不一样的。鉴于上述这些理由，CSLA 强制开发人员写出在 for 语句中嵌 RemoveAt()的语句，这样既不会出错，代码也较为清晰规范(“精简代码”不是绝对的)。