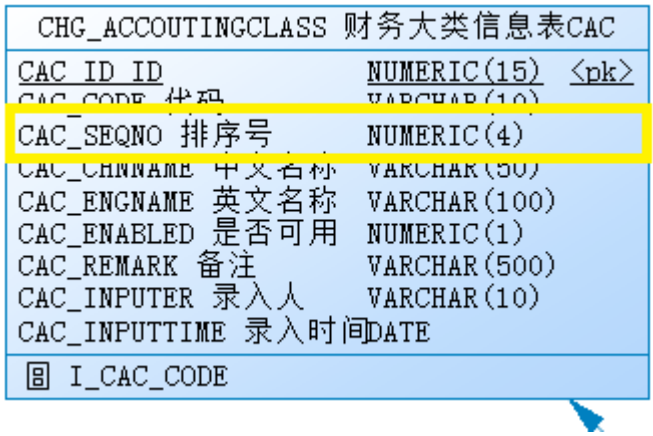
1 业务数据的缺省值-序号

1.1 参考资料

- “使用指南.12.业务结构对象模型”的“业务数据的缺省值”章节；

1.2 业务场景



当新增一个财务大类时，其排序号属性值应该缺省等于财务大类集合中所有排序号属性值的最大值加1。

1.3 实现方法

Phenix.Business.BusinessBase<T>的RegisterProperty()函数，提供了为业务类的字段动态赋缺省值的功能，其中一个重载函数是：

```
/// <summary>
/// 注册属性信息
/// </summary>
/// <typeparam name="P">属性类</typeparam>
/// <param name="propertyLambdaExpression">属性表达式</param>
/// <param name="defaultValueFunc">缺省值函数</param>
/// <returns>属性信息</returns>
protected static PropertyInfo<P> RegisterProperty<P>(Expression<Func<T, object>>
propertyLambdaExpression, Func<T, object> defaultValueFunc)
```

我们使用它，可实现业务场景所需功能，代码如下：

```
/// <summary>
/// 排序号
/// </summary>
```

```
        public static readonly Phenix.Business.PropertyInfo<int?> SeqnoProperty =
RegisterProperty<int?>(c => c.Seqno, (c) =>
            {
                if (c.Owner != null)
                  return c.Owner.Count == 0 ? 1 : ((IEnumerable<T>)c.Owner).Max(t => t.Seqno == null ? 0 :
t.Seqno.Value) + 1;
                return null;
            });
        [Phenix.Core.Mapping.Field(PropertyName = "Seqno", FriendlyName = "排序号", Alias = "CAC_SEQNO",
TableName = "CHG_ACCOUTINGCLASS", ColumnName = "CAC_SEQNO", NeedUpdate = true)]
        private int? _seqno;
        /// <summary>
        /// 排序号
        /// </summary>
        public int? Seqno
        {
            get { return GetProperty(SeqnoProperty, _seqno); }
            set { SetProperty(SeqnoProperty, ref _seqno, value); }
        }
```

当新增该业务对象及被添加到业务对象集合中时，上述黄底代码（缺省值计算函数）会被自动调用。我们可以通过测试代码，观察被调用时的一些上下文数据的变化情况：

```
    static void Main(string[] args)
    {
      //设为调试状态
      Phenix.Core.AppConfig.Debugging = true;
      //模拟登陆
      Phenix.Business.Security.UserPrincipal.User =
        Phenix.Business.Security.UserPrincipal.CreateAuthenticated(
          UserIdentity.AdminId, UserIdentity.AdminUserName, UserIdentity.AdminUserNumber,
          String.Empty, new long?(), new long?());
      Phenix.Services.Client.Library.Registration.RegisterEmbeddedWorker(false);


      AccoutingClassList accoutingClassList = AccoutingClassList.New();
      accoutingClassList.AddNew();
      AccoutingClass accoutingClass = AccoutingClass.New();
      accoutingClassList.Add(accoutingClass);
    }
```

当程序执行到上述黄底第二行代码，然后跳转到缺省值计算函数时：

```
61          /// <summary>
62          /// 排序号
63          /// </summary>
64          public static readonly Phenix.Business.PropertyInfo<int?> SeqnoProperty = RegisterProperty<int?>(c => c.Seqno, (c) =>
65            {
66              if (c.Owner != null)
67                return c.Owner.Count == 0 ? 1 : ((IEnumerable<T>)c.Owner).Max(t => t.Seqno == null ? 0 : t.Seqno.Value) + 1;
68              return null;
69            });
70          [Phenix.Core.Mapping.Field(PropertyName = "Seqno", FriendlyName = "排序号", Alias = "CAC_SEQNO", TableName = "CHG_ACCO
71          private int? _seqno;
72          /// <summary>
```

100 %

自动窗口

名称	值	类型
c	{393262519555116}	SHB.Component.Charge.Systemic.Business.AccoutingClass
c.Owner	Count = 0	Phenix.Business.IBusinessCollection {SHB.Component.Charge.Systemic.Business.AccoutingClassList}

执行到上述黄底第三行代码，然后跳转到缺省值计算函数时：

```
61          /// <summary>
62          /// 排序号
63          /// </summary>
64          public static readonly Phenix.Business.PropertyInfo<int?> SeqnoProperty = RegisterProperty<int?>(c => c.Seqno, (c) =>
65            {
66              if (c.Owner != null)
67                return c.Owner.Count == 0 ? 1 : ((IEnumerable<T>)c.Owner).Max(t => t.Seqno == null ? 0 : t.Seqno.Value) + 1;
68              return null;
69            });
70          [Phenix.Core.Mapping.Field(PropertyName = "Seqno", FriendlyName = "排序号", Alias = "CAC_SEQNO", TableName = "CHG_ACCOU
71          private int? _seqno;
72          /// <summary>
```

100 %

自动窗口

名称	值	类型
c	{393265697760116}	SHB.Component.Charge.Systemic.Business.AccoutingClass
c.Owner	null	Phenix.Business.IBusinessCollection

执行到上述黄底第四行代码，然后跳转到缺省值计算函数时：

```
61          /// <summary>
62          /// 排序号
63          /// </summary>
64          public static readonly Phenix.Business.PropertyInfo<int?> SeqnoProperty = RegisterProperty<int?>(c => c.Seqno, (c) =>
65            {
66              if (c.Owner != null)
67                return c.Owner.Count == 0 ? 1 : ((IEnumerable<T>)c.Owner).Max(t => t.Seqno == null ? 0 : t.Seqno.Value) + 1;
68              return null;
69            });
70          [Phenix.Core.Mapping.Field(PropertyName = "Seqno", FriendlyName = "排序号", Alias = "CAC_SEQNO", TableName = "CHG_ACCO
71          private int? _seqno;
72          /// <summary>
```

100 %

自动窗口

名称	值	类型
c	{393265697760116}	SHB.Component.Charge.Systemic.Business.AccoutingClass
c.Owner	Count = 1	Phenix.Business.IBusinessCollection {SHB.Component.Charge.Systemic.Business.AccoutingClassList}

我们会注意到，测试代码中，共生成了两个业务对象并添加进了业务对象集合中，而且采用了两种方法：

- 调用业务集合对象的 AddNew()函数；
- 先调用业务类的 New()工厂方法生成业务对象，然后调用业务集合对象的 Add()函数；

这两种方法，会执行不一样的流程来调用到缺省值计算函数：前一种方法，调用一次就达到目的；而后一种方法，将被调用两次，第一次是在 New()工厂方法时，虽然被调用了，但此时并未知其所属业

务集合，所以未达到初始化的目的，而第二次是在它被添加进业务对象集合时重新调用的，此时因 Owner 有了实际的值而得以实现字段值的初始化。所以，不管采取那种方法，缺省值计算函数都会被调用到，只要这些字段未被初始化过（即值为 null）。