12 业务结构对象模型

12.7 业务数据的缺省值

New 业务对象，可以通过重载下述函数来初始化它的字段值：

```
/// <summary>
/// 当新增初始化之后
/// 本函数仅通过业务类工厂函数New()或将新增业务对象添加到业务集合时才被调用
/// </summary>
protected virtual void OnInitializeNew()
{
}
```

严格禁止将 New 业务对象的逻辑代码写在构造函数里，原因是不仅仅 New 业务对象时会调用到构造函数，当 Fetch 业务对象时也一样会被调用到，这将严重影响到 Fetch 效率，而且本该是和数据库业务表的记录内容保持一致的属性值也被这些代码改写了，造成前后台数据的不一致。从概念理解上，构造函数是用来操作对象的，而 New 业务对象才是逻辑意义上真正要操作业务对象的。

不过，也可以通过下述方法来填充上缺省值。

12.7.1缺省值的填充顺序

Phenixヾ为应用系统提供了几种缺省值注册机制，而且是按照下述顺序执行填充的，后者不会覆盖掉前者的填充值，只有当字段值为 null 才被填充：

1，通过 UserPrincipal.User.Identity.SetFieldDefaultValue()函数注册的缺省值；

2，通过 Phenix.Business.BusinessBase<T>的 RegisterProperty()函数注册的缺省值；

3，对应到数据库表字段的缺省值；

4，如果是日期字段，当通过上述方式得到的缺省值不为 null、且不是通过 function 得到时，则替换为当前时间；

5，通过 Phenix.Business.BusinessBase<T>的 SetDefaultValue()函数注册的缺省值；

12.7.2用缺省值重置业务数据

除了 New 业务对象在初始化的时候，其业务数据会按照上述顺序被自动填充缺省值外，还可以通过 Phenix.Business.BusinessBase<T>提供的下述函数重置业务数据：

```
/// <summary>
/// 填充字段值到缺省值
/// </summary>
```

```
    public void FillFieldValuesToDefault()

    /// <summary>
    /// 填充字段值到缺省值
    /// </summary>
    /// <param name="reset">重新设定</param>
    public virtual void FillFieldValuesToDefault(bool reset)
```

12.7.3通过 Phenix.Business.BusinessBase<T>的 RegisterProperty()函数注册

12.7.3.1 静态缺省值

```
    /// <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="defaultValue">缺省值</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Expression<Func<T, object>>
propertyLambdaExpression, P defaultValue)

    /// <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="defaultValue">缺省值</param>
    /// <param name="friendlyName">友好名</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Expression<Func<T, object>>
propertyLambdaExpression, P defaultValue, string friendlyName)
```

12.7.3.2 动态缺省值

```
    /// <summary>
    /// 注册属性信息
    /// </summary>
    /// <typeparam name="P">属性类</typeparam>
    /// <param name="info">属性信息</param>
    /// <param name="defaultValueFunc">缺省值函数</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Csla.PropertyInfo<P> info,
      Func<object> defaultValueFunc)

    /// <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<object> defaultValueFunc)

    /// <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>
    /// <param name="friendlyName">友好名</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Expression<Func<T, object>>
propertyLambdaExpression, Func<object> defaultValueFunc, string friendlyName)

    /// <summary>
    /// 注册属性信息
    /// </summary>
    /// <typeparam name="P">属性类</typeparam>
    /// <param name="info">属性信息</param>
    /// <param name="defaultValueFunc">缺省值函数</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Csla.PropertyInfo<P> info,
      Func<T, object> defaultValueFunc)

    /// <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)

    /// <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>
    /// <param name="friendlyName">友好名</param>
    /// <returns>属性信息</returns>
    protected static PropertyInfo<P> RegisterProperty<P>(Expression<Func<T, object>>
propertyLambdaExpression, Func<T, object> defaultValueFunc, string friendlyName)
```

设置了动态缺省值，可以在系统的运行期，每次提取的缺省值都允许不一样：

```
        /// <summary>
        /// 计划开始时间
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<DateTime?> OpenTimeProperty =
RegisterProperty<DateTime?>(c => c.OpenTime, () => DateTime.Today);
        [Phenix.Core.Mapping.Field(FriendlyName = "计划开始时间", Alias = "ETP_OPEN_TIME", TableName =
"WPL_ENTRYING_PLAN", ColumnName = "ETP_OPEN_TIME", NeedUpdate = true)]
        private DateTime? _openTime;
        /// <summary>
        /// 计划开始时间
        /// </summary>
        [System.ComponentModel.DisplayName("计划开始时间")]
        public DateTime? OpenTime
        {
            get { return GetProperty(OpenTimeProperty, _openTime); }
            set { SetProperty(OpenTimeProperty, ref _openTime, value); }
        }

        /// <summary>
        /// 计划结束时间
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<DateTime?> CloseTimeProperty =
RegisterProperty<DateTime?>(c => c.CloseTime, () => DateTime.Today.AddDays(7));
        [Phenix.Core.Mapping.Field(FriendlyName = "计划结束时间", Alias = "ETP_CLOSE_TIME", TableName =
"WPL_ENTRYING_PLAN", ColumnName = "ETP_CLOSE_TIME", NeedUpdate = true)]
        private DateTime? _closeTime;
        /// <summary>
        /// 计划结束时间
        /// </summary>
        [System.ComponentModel.DisplayName("计划结束时间")]
        public DateTime? CloseTime
        {
            get { return GetProperty(CloseTimeProperty, _closeTime); }
            set { SetProperty(CloseTimeProperty, ref _closeTime, value); }
        }
```

12.7.4通过 UserPrincipal.User.Identity.SetFieldDefaultValue()函数注册

```
    /// <summary>
    /// 设置正则表达式字段缺省值
    /// </summary>
    /// <param name="regexColumnName">正则表达式字段名</param>
    /// <param name="value">缺省值</param>
    public void SetFieldDefaultValue(string regexColumnName, object value)

    /// <summary>
    /// 删除正则表达式字段缺省值
    /// </summary>
    /// <param name="regexColumnName">正则表达式字段名</param>
    public void RemoveFieldDefaultValue(string regexColumnName)

    /// <summary>
    /// 获取正则表达式字段缺省值
    /// </summary>
    /// <param name="regexColumnName">正则表达式字段名</param>
    public object GetFieldDefaultValue(string regexColumnName)
```

12.7.5通过 Phenix.Business.BusinessBase<T>的 SetDefaultValue()函数注册

```
    /// <summary>
    /// 设置缺省值
    /// </summary>
    /// <param name="property">属性信息</param>
    /// <param name="value">值</param>
    /// <param name="allowReplace">如果为 true, 则当属性被赋值时允许赋值的内容覆盖本缺省值</param>
    public static void SetDefaultValue(Csla.Core.IPropertyInfo property, object value, bool allowReplace)

    /// <summary>
    /// 设置缺省值
    /// </summary>
    /// <param name="property">属性信息</param>
    /// <param name="valueFunc">值函数</param>
    public static void SetDefaultValue(Csla.Core.IPropertyInfo property, Func<T, object> valueFunc)
```

```
/// <summary>
/// 移除缺省值
/// </summary>
/// <param name="property">属性信息</param>
public static void RemoveDefaultValue(Csla.Core.IPropertyInfo property)
```

此类注册方式，可在不改动原业务类代码，或者说不改动原业务逻辑层 DLL 的前提下，注入（或替换、移除）扩展的业务逻辑。

比如下列代码，可自动为新增（AddNew()）业务对象的索引号（案例中的 ProcessLock.SubIndex 属性）赋值，计算逻辑为所属业务对象集合（Owner）拥有的业务对象数+1：

```
ProcessLock.SetDefaultValue(ProcessLock.SubIndexProperty, t => t.Owner.Count + 1);
ProcessLockList processLockList = ProcessLockList.Fetch();
ProcessLock processLock = processLockList.AddNew();
```

12.7.6Phenix.Business.BusinessBase<T>提供获取缺省值的函数

```
/// <summary>
/// 获取缺省值
/// </summary>
/// <param name="property">属性信息</param>
/// <param name="onlyDynamic">如果为 true, 则仅返回通过SetDefaultValue()设置的缺省值; 否则返回完整版</param>
public object GetDefaultValue(Csla.Core.IPropertyInfo property, bool onlyDynamic)
```