8 业务对象公共接口的数据验证

8.1 业务对象公共接口

面向对象的三个基本特征之一就是“封装”。业务对象自身拥有的数据只有通过public的属性、过程、事件等方式才能被外部对象操作。

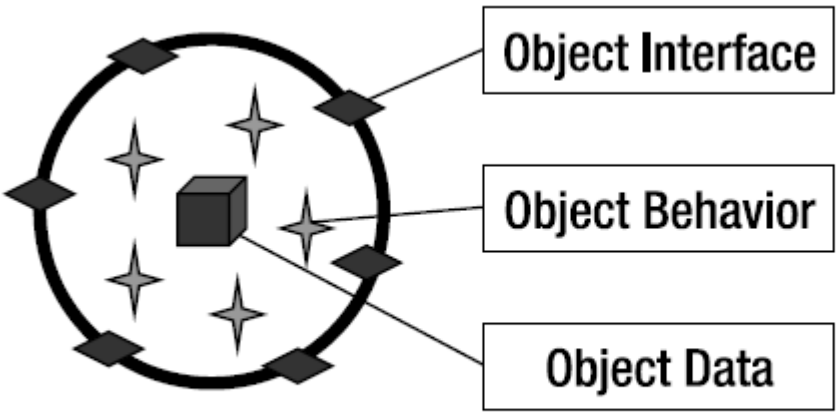

Figure 1-14. *A business object composed of state, implementation, and interface*

上图摘自 CSLA 作者的《Expert C# 2008 Business Objects》。

8.2 什么是数据验证

任何通过业务对象的公共接口交换的数据，其来源是多种多样且不可信的，都必须被业务对象验证，以保证其有效性（正确、完整），这样的应用系统才是强壮的。

操作数据的验证是业务对象的职责之一，与业务需求的数据约束条件一一对应。

8.3 数据验证的实现方法

数据验证是采取业务规则类的方式实现的，并通过业务基类 (BusinessBase<T>)提供的注册机制注入到业务类的业务规则库中的：

```
/// <summary>
/// 添加业务规则
/// </summary>
protected override void AddBusinessRules()
```

当业务对象属性被 set 时（内部代码调用了 SetProperty()函数），CSLA 会遍历这个业务规则库中继承自 Phenix.Business.Rules.EditValidationRule 的属性有效性规则对象，验证属性值的有效性。

业务对象本身数据完整性的验证，需构建继承自 Phenix.Business.Rules.ObjectRule 的对象级别规则类，并注入到业务规则库中，当提交业务对象时，这些规则对象会被调用。

业务规则类的构建及注入方法，请参考“03.Addin 工具使用方法”的“构建对象级别规则类”、“构建属性有效性规则类” 、“初始化公共业务规则类”等章节。

8.4 通用的数据验证规则

除了 CSLA 的业务规则注入机制， Phenix丶还额外提供了一套机制，结合了数据字典的有效性验证信息、微软 Entity Framwork 的 System.ComponentModel.DataAnnotations。它们之间的区别如下，请按需使用：

- 数据字典的有效性验证，被默认设定到业务类上；
- 规则类是在发现验证数据无效时抛出异常；而打标签或数据字典则可能会对数据进行再处理，比如“超长的字符串会被截断”之类的处理方式，以符合最接近于规则要求的值；
- 规则类的应用可满足应用系统的可扩展性需求（见下文“可扩展性”章节）；

具体说明如下：

8.4.1 限制必输

方法（按优先级从高到低排序）：

- 注册：Phenix.Business.Rules.Required；
- 配置表 PH_AssemblyClassProperty.AP_Required 非空值：0=false、1=true；
- 字段、属性标签：System.ComponentModel.DataAnnotations.RequiredAttribute；
- 字段标签：Phenix.Core.Mapping.FieldRuleAttribute 的 IsRequired = true；
- 数据库的数据字典：字段为 not null；

8.4.2 限制字符串最大、最小长度

方法（按优先级从高到低排序）：

- 注册：Phenix.Business.Rules.MaxLength、Phenix.Business.Rules.MinLength；
- 字段、属性标签：System.ComponentModel.DataAnnotations.StringLengthAttribute；
- 字段标签：Phenix.Core.Mapping.FieldRuleAttribute 的 MaximumLength、MinimumLength 属性值；
- 数据库的数据字典：字段长度；

8.4.3 限制大小范围

方法（按优先级从高到低排序）：

- 注册：Csla.Rules.CommonRules.MinValue<T>、Csla.Rules.CommonRules.MaxValue<T>；
- 字段、属性标签：System.ComponentModel.DataAnnotations.RangeAttribute；

8.4.4 以正则表达式验证数据格式

方法（按优先级从高到低排序）：

- 注册：Csla.Rules.CommonRules.RegExMatch；
- 字段、属性标签：System.ComponentModel.DataAnnotations.RegularExpressionAttribute；

8.4.5 比较值大小（范围）

方法：

- 注册：Phenix.Business.Rules.RangeCompareRule<T>；

用于对象两个关联属性的值大小比较，比如“计划开始时间必须早于计划结束时间”、“库存最大值要大于库存最小值”等等。

8.4.6 强制移除字符串所有前导空白字符和尾部空白字符

方法：

- 字段、属性标签：Phenix.Core.Mapping.FieldRuleAttribute 的 StringTrim = true；

当字段名称后缀为“CODE”时被强制设定本规则。

8.4.7 强制转换大小写

方法：

- 字段、属性标签：Phenix.Business.Rules.FieldRuleAttribute 的 StringUpperCase = true；

当字段名称后缀为“CODE”时被强制设定本规则。

8.4.8 自动将指定属性值转换成本属性值的拼音码

方法：

- 字段、属性标签：Phenix.Business.Rules.FieldRuleAttribute 的 PinyinCodeProperty = ″被转换的属性名″；

在业务类里如果存在后缀为“CODE”、“NAME”两种字段，则后缀为“NAME”的字段被强制设定本规则，而其 PinyinCodeProperty 参数值被设定为后缀为“CODE” 字段所关联属性的名称。这样，后缀为“CODE”的字段值将与后缀为“NAME”的字段值保持同步，它等于后缀为“NAME”的字段值的中文字符的拼音第一个字符及（可能）夹杂英文字母的顺序组合；但是，当“CODE”的字段值被赋值为其他值后就不受此规则约束，所以本规则不是强制约束。

8.4.9 强制移除 DateTime 值的时间部分

方法：

- 字段、属性标签：Phenix.Business.Rules.FieldRuleAttribute 的 DateNotContainTime = true；

当字段名称后缀为“DATE”时被强制设定本规则。

8.4.10强制限定 DateTime 值的起始范围（相对当前时间）

方法：

- 字段、属性标签：Phenix.Business.Rules.FieldRuleAttribute 的 DateTimeStartSpan ="相对当前时间"、DateTimeStartSpanOfDays = "相对当前天数"、DateTimeStartSpanOfHours = "相对当前小时数"、DateTimeStartSpanOfMinutes = "相对当前分钟数"]；

如果超出此起始范围，则强制更改为最接近的起始值。

8.4.11强制限定 DateTime 值的截止范围（相对当前时间）

方法：

- 字段、属性标签：Phenix.Business.Rules.FieldRuleAttribute 的 DateTimeStopSpan = "相对当前时间"、DateTimeStopSpanOfDays = "相对当前天数"、DateTimeStopSpanOfHours = "相对当前小时数"、DateTimeStopSpanOfMinutes = "相对当前分钟数"；

如果超出此截止范围，则强制更改为最接近的截止值。

8.5 可扩展性

与授权规则一样（见“05.业务对象公共接口的授权”中“可扩展性”章节）,数据验证规则也分基础规则和扩展规则。基础规则的算法是被固化的，其余的都可以实现可扩展性，以支持产品化软件的可扩展能力。

8.5.1 基础规则的可扩展性

基础规则的扩展方法，是从业务逻辑的算法中抽取出参数并将它重构为规则对象的可配置化属性。具体如何实现？请见“03.Addin 工具使用方法”的“可配置化对象”章节。

8.5.2 扩展规则的可插拔性

业务基类 Phenix.Business.BusinessBase<T>为数据验证规则的可扩展性提供了以下的静态事件：

```
/// <summary>
/// 业务规则注册中事件
/// 可配置化：当应用程序初始化时，可通过本事件来添加额外的业务规则库
/// </summary>
```

```
    public static event BusinessRuleRegisteringEventHandler BusinessRuleRegistering;
```

以“"是否允许执行"有效性规则”类为例，黄底代码部分是为方便动态注册业务规则而额外编写的：

```
  /// <summary>
  /// "是否允许执行"有效性规则
  /// </summary>
  public class ProcessLockAllowexecuteEditValidationRule : Phenix.Business.Rules.EditValidationRule
  {
    /// <summary>
    /// 初始化
    /// </summary>
    public ProcessLockAllowexecuteEditValidationRule()
      : base(ProcessLock.AllowexecuteProperty) { }

    #region Register

    /// <summary>
    /// 注册业务规则
    /// </summary>
    public static void Register()
    {
      ProcessLock.BusinessRuleRegistering += new
Phenix.Business.Core.BusinessRuleRegisteringEventHandler(ProcessLock_BusinessRuleRegistering);
    }

    private static void ProcessLock_BusinessRuleRegistering(Csla.Rules.BusinessRules businessRules)
    {
      businessRules.AddRule(new ProcessLockAllowexecuteEditValidationRule());
    }

    #endregion

    /// <summary>
    /// 执行
    /// </summary>
    protected override void Execute(Csla.Rules.RuleContext context)
    {
      var value = this.ReadProperty(context.Target, PrimaryProperty);
      if (value == null || string.IsNullOrWhiteSpace(value.ToString()))
      {
        var message = string.Format("{0}不允许为空！", PrimaryProperty.FriendlyName);
        context.Results.Add(new RuleResult(RuleName, PrimaryProperty, message) { Severity =
```

```
RuleSeverity.Error });
      }
    }
  }
```

这样，只要在应用程序初始化时，调用如下代码，就可以将上述规则添加进业务类的业务规则库中：

```
ProcessLockAllowexecuteEditValidationRule.Register();
```

那么，以下写死在业务类中的代码（黄底代码）就可删除掉了：

```
/// <summary>
/// 注册业务规则
/// </summary>
protected override void AddBusinessRules()
{
  BusinessRules.AddRule(new ProcessLockAllowexecuteEditValidationRule());
  base.AddBusinessRules();
}
```

8.5.3 具体应用

软件产品的业务逻辑组件具体形式可分为基础程序集和扩展程序集，业务类及其基础业务规则类被编译为一个基础程序集、而外围的扩展规则类被编译成各个独立的扩展程序集。将这些程序集发布给实施团队，在产品的实施阶段就可以按需组装成实际的应用系统，实施团队也可以自行在此基础上二次开发规则库，动态注册到应用系统里。

8.6 业务类上与数据验证规则相关的接口

8.6.1 Phenix.Business.BusinessBase<T>提供的功能

属性	说明	备注
IsValid	是否本业务对象及其从业务对象集合中数据具备有效性	
IsSelfValid	是否本业务对象数据具备有效性	

方法	说明	参数类型	参数	参数说明
IsValidProperty	是否有效属性	IPropertyInfo	property	属性信息
		bool	returns	

CheckRules	校验本业务对象及其从业务对象集合中数据是否有效	bool	onlyOldError	仅检查原有错误
		bool	onlySelfDirty	仅检查脏数据
		string	returns	错误信息
CheckSelfRules	校验本业务对象数据是否有效	bool	onlyOldError	仅检查原有错误
		bool	onlySelfDirty	仅检查脏数据
		bool	throwIfException	如果为 true，则当发现!IsSelfValid 时抛出 Csla.Rules.ValidationException 异常
		IDataErrorInfo	returns	错误信息
CheckSelfObjectRules	校验本业务对象数据是否有效（仅 ObjectRules）	bool	onlySelfDirty	仅检查脏数据
		bool	throwIfException	如果为 true，则当发现!IsSelfValid 时抛出 Csla.Rules.ValidationException 异常
		IDataErrorInfo	returns	错误信息
CheckRepeated	校验是否存在重复数据	List<IEntity>	returns	重复数据的对象

8.6.2 Phenix.Business.BusinessListBase<T, TBusiness>提供的功能

属性	说明	备注
IsValid	是否本集合内的业务对象及其从业务对象集合中数据具备有效性	
IsSelfValid	同 IsValid	

方法	说明	参数类型	参数	参数说明
FindInvalidItem	搜索无效对象	bool	onlyOldError	仅检查原有错误
		TBusiness	returns	无效对象
CheckRules	校验本集合内的业务对象及其从业务对象集合	bool	onlyOldError	仅检查原有错误

	中数据是否有效			
		bool	onlySelfDirty	仅检查脏数据
		string	returns	错误信息
CheckSelfRules	校验本集合内的业务对象数据是否有效	bool	onlyOldError	仅检查原有错误
		bool	onlySelfDirty	仅检查脏数据
		bool	throwIfException	如果为 true，则当发现在!IsSelfValid 时抛出 Csla.Rules.ValidationException 异常
		IDataErrorInfo	returns	错误信息
CheckSelfObjectRules	校验本业务对象数据是否有效（仅 ObjectRules）	bool	onlySelfDirty	仅检查脏数据
		bool	throwIfException	如果为 true，则当发现!IsSelfValid 时抛出 Csla.Rules.ValidationException 异常
		IDataErrorInfo	returns	错误信息
CheckRepeated	校验是否存在重复数据	List<IEntity>	returns	重复数据的对象

8.7 前提条件

上述验证方法是否能得到正确的结果，是建立在业务类是否按照以下的编码规范要求进行设计。

8.7.1 注册需纳入到数据验证规则的属性

```
        /// <summary>
        /// 名称
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<string> NameProperty =
RegisterProperty<string>(c => c.Name);
        [Phenix.Core.Mapping.FieldUniqueAttribute("I_SSF_NAME")]
        [Phenix.Core.Mapping.FieldRuleAttribute(StringOnImeMode = true, PinyinCodeProperty = "Code")]
        [Phenix.Core.Mapping.Field(FriendlyName = "名称", Alias = "SSF_NAME", TableName = "SYS_SHIFT",
ColumnName = "SSF_NAME", NeedUpdate = true, InLookUpColumn = true, InLookUpColumnDisplay = true)]
        private string _name;
        /// <summary>
```

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

8.7.2 注册需纳入到数据验证规则的对象

无编码规范要求。

8.8 数据验证的开关

8.8.1 静态开关

可设置 Phenix.Core.Mapping.FieldAttribute 的 InValidation 属性为 false（默认 true）关闭验证功能。

8.8.2 动态开关

业务类实现了 Csla.Core.ICheckRules 接口：

```
namespace Csla.Core
{
  /// <summary>
  /// Defines the common methods for any business object which exposes means
  /// to supress and check business rules.
  /// </summary>
  public interface ICheckRules
  {
    /// <summary>
    /// Sets value indicating no rule methods will be invoked.
    /// </summary>
    void SuppressRuleChecking();

    /// <summary>
    /// Resets value indicating all rule methods will be invoked.
    /// </summary>
    void ResumeRuleChecking();

    /// <summary>
    /// Invokes all rules for the business type.
    /// </summary>
```

```
    void CheckRules();

    /// <summary>
    /// Gets the broken rules collection
    /// </summary>
    /// <returns></returns>
    BrokenRulesCollection GetBrokenRules();
  }
}
```

SuppressRuleChecking()函数和 ResumeRuleChecking()函数是一对是否执行数据验证的开关。

在隐含执行数据验证的时候，比如业务对象属性被赋值、业务对象被保存的时候，可以事先调用 SuppressRuleChecking()函数来关闭数据验证，以达到临时忽略数据验证的目的，事后需及时调用 ResumeRuleChecking()函数来解除关闭状态。