反写插件示例代码

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No** | **日期** | **编制\修订** | **校对** | **批准** | **修改的章节号** |
| V1.0 | 20150415 | 符勇  张晋博 | 丁振华 | 赖碧云 | 初始版本 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

问题和意见

如果你对文档有任何意见、问题或想法，或者你的问题未在此文档中找到答案，请通过电子邮件联系我们

[jinbo\_zhang@kingdee.com](mailto:jinbo_zhang@kingdee.com)

目录

[反写插件示例代码 1](#_Toc417398248)

[反写插件概要说明 3](#_Toc417398249)

[反写插件的基类与注册 3](#_Toc417398250)

[示例 – 演示事件触发顺序 5](#_Toc417398251)

[公共属性 11](#_Toc417398252)

[BusinessInfo 11](#_Toc417398253)

[Context 11](#_Toc417398254)

[DynamicObjects 11](#_Toc417398255)

[LinkEntity 11](#_Toc417398256)

[OperateOption 11](#_Toc417398257)

[OperationNumber 11](#_Toc417398258)

[事件 11](#_Toc417398259)

[BeforeTrackBusinessFlow 11](#_Toc417398260)

[BeforeCreateArticulationRow 12](#_Toc417398261)

[BeforeWriteBack 13](#_Toc417398262)

[AfterCustomReadFields 14](#_Toc417398263)

[AfterCommitAmount 17](#_Toc417398264)

[BeforeCloseRow 19](#_Toc417398265)

[AfterCloseRow 22](#_Toc417398266)

[BeforeCheckHighLimit 24](#_Toc417398267)

[AfterCheckHighLimit 27](#_Toc417398268)

[BeforeSaveWriteBackData 28](#_Toc417398269)

[AfterSaveWriteBackData 31](#_Toc417398270)

[FinishWriteBack 31](#_Toc417398271)

# 反写插件概要说明

单据保存、审核时，会根据其业务流程追踪信息，搜索出全部上游单据，然后逐个执行反写规则，把数据反写到上游单据上。

单据反审核、删除时，会根据历史反写记录，回滚反写，把已反写数据从上游单据上撤销。

反写过程，大概分为如下几个步骤：

1. 根据单据的关联子表xxx\_LK记录，寻找到单据的直接源单；
2. 寻找到直接源单所在的业务流程实例；
3. 把本单，追加到业务流程实例中；
4. 搜索本单的全部反写规则；
5. 判断本单的各行，是否符合各个反写规则的反写条件、反写量，反写源单内码等；
6. 开始循环执行反写规则；
7. 批量读取本反写规则需要反写的全部源单；
8. 启动循环，开始逐行反写；
9. 反写数量到源单；
10. 判断源单是否达成关闭条件，进行关闭处理；
11. 判断源单反写后，是否超额；
12. 逐行循环，反写完毕；
13. 批量保存本次反写的源单；
14. 开始执行下一个反写规则，跳到步骤7；

保存、审核、反审核、删除、作废等操作的反写过程，可以通过插件进行干预，即反写插件。

特别说明：

**单据的反写插件是公用的，会覆盖本单全部反写规则的执行过程。**

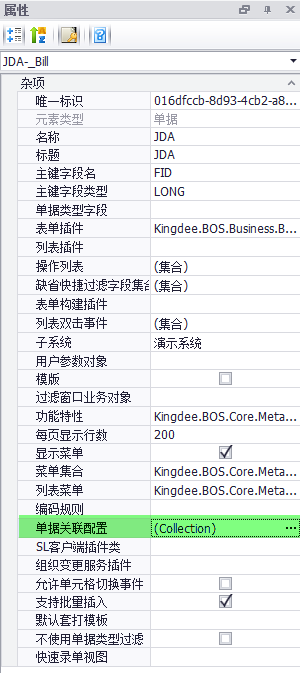
**因此，非常有必要，在反写插件中，对当前执行的反写规则进行判断，如果不是目标反写规则，则需要略过；**

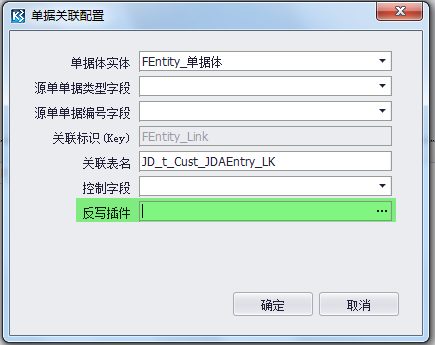
## 反写插件的基类与注册

反写插件基类：

Kingdee.BOS.Core.BusinessFlow.PlugIn.AbstractBusinessFlowServicePlugIn

反写插件，注册到单据关联配置属性：





反写插件需要部署到K/3 Cloud站点目录的Website\Bin子目录；

## 示例 – 演示事件触发顺序

下文展示了一个空的反写插件，仅仅用于输出每个事件的执行顺序，没有任何业务逻辑。

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.ComponentModel;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.BusinessFlow;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.BusinessFlow.PlugIn.Args;  namespace SamplePlugIn.WriteBackPlugIn  {  /// <summary>  /// 按顺序输出反写插件的事件  /// </summary>  [Description("按顺序输出反写插件的事件")]  public class WritebackEventSeq : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 记录业务流程跟踪信息之前，触发本事件；  /// 此时，还未分析本单的关联信息  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，强制要求不创建业务流程跟踪数据，不反写  /// </remarks>  public override void BeforeTrackBusinessFlow(BeforeTrackBusinessFlowEventArgs e)  {  base.BeforeTrackBusinessFlow(e);  this.PrintEventName("BeforeTrackBusinessFlow");  }  /// <summary>  /// 逐行循环，分析单据体每行的关联数据之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，强制要求不分析当前单据体行的关联关系，从而不创建此行的业务流程跟踪数据，不反写  /// </remarks>  public override void BeforeCreateArticulationRow(BeforeCreateArticulationRowEventArgs e)  {  base.BeforeCreateArticulationRow(e);  this.PrintEventName("BeforeCreateArticulationRow");  }  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 系统在循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 每个反写规则的执行，从本事件开始，到反写源单反写完毕，保存成功，为一个大循环；  /// 之后再继续下一个反写规则；  /// 因此，本事件之后的全部事件，会被循环执行；  /// 为避免重复处理，插件在事件处理之前，需要自行判断当前的反写规则标示。  /// 可以在此事件中，监测待执行的反写规则，设置一个本地标志量，在随后的事件中，直接略过事件处理代码；  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  base.BeforeWriteBack(e);  this.PrintEventName("BeforeWriteBack");  if (e.Rule.Id.EqualsIgnoreCase("MyRuleId"))  {  \_thisIsMyRule = true;  }  }  /// <summary>  /// 确认了反写源单需加载的字段之后，实际读取数据包之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，请求加载插件需要用到的源单字段  /// </remarks>  public override void AfterCustomReadFields(AfterCustomReadFieldsEventArgs e)  {  base.AfterCustomReadFields(e);  this.PrintEventName("AfterCustomReadFields");    if (this.\_thisIsMyRule == false) return;  e.AddFieldKey("MyFieldKey");  }  /// <summary>  /// 对源单行反写之后，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 本事件参数中，提供了源单数据包、源单行数据包、源单反写字段，以及本次反写量；  /// 可以在此事件中，根据本次反写量，做其他数据的同步  /// </remarks>  public override void AfterCommitAmount(AfterCommitAmountEventArgs e)  {  base.AfterCommitAmount(e);  this.PrintEventName("AfterCommitAmount");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 判断源单行关闭条件之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，取消对源单行的关闭处理  /// </remarks>  public override void BeforeCloseRow(BeforeCloseRowEventArgs e)  {  base.BeforeCloseRow(e);  this.PrintEventName("BeforeCloseRow");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 对源单行进行关闭处理之后，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，做其他数据行的同步关闭处理  /// </remarks>  public override void AfterCloseRow(AfterCloseRowEventArgs e)  {  base.AfterCloseRow(e);  this.PrintEventName("AfterCloseRow");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 判断反写结果是否超额之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，取消超额检查，允许超额  /// </remarks>  public override void BeforeCheckHighLimit(BeforeCheckHighLimitEventArgs e)  {  base.BeforeCheckHighLimit(e);  this.PrintEventName("BeforeCheckHighLimit");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 在超额检查完毕，给出超额警告之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，修正超额检查结果，或者补充超额检查提示  /// </remarks>  public override void AfterCheckHighLimit(AfterCheckHighLimitEventArgs e)  {  base.AfterCheckHighLimit(e);  this.PrintEventName("AfterCheckHighLimit");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 当前反写规则执行完毕，完成了对反写源单全部行的反写，保存源单数据之前，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对源单数据包进行修订  /// </remarks>  public override void BeforeSaveWriteBackData(BeforeSaveWriteBackDataEventArgs e)  {  base.BeforeSaveWriteBackData(e);  this.PrintEventName("BeforeSaveWriteBackData");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 当前反写规则执行完毕，源单数据成功保存之后，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，同步处理其他数据  /// </remarks>  public override void AfterSaveWriteBackData(AfterSaveWriteBackDataEventArgs e)  {  base.AfterSaveWriteBackData(e);  this.PrintEventName("AfterSaveWriteBackData");  if (this.\_thisIsMyRule == false) return;  }  /// <summary>  /// 全部反写规则执行完毕后，触发本事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，释放一些本地变量  /// </remarks>  public override void FinishWriteBack(FinishWriteBackEventArgs e)  {  base.FinishWriteBack(e);  this.PrintEventName("FinishWriteBack");  }    /// <summary>  /// 等待100ms之后，输出当前时间 + 事件名  /// </summary>  /// <param name="eventName"></param>  private void PrintEventName(string eventName)  {  System.Threading.Thread.Sleep(100);  string message = string.Format("{0} : {1}", DateTime.Now, eventName);  System.Diagnostics.Debug.Print(message);  }  }  } | |

特别关注：**示例代码中this.\_thisIsMyRule变量的运用；**

# 公共属性

## BusinessInfo

当前单据元数据信息

## Context

上下文

## DynamicObjects

当前选择单据数据信息

## LinkEntity

当前单据BOSIDE中配置的单据关联配置信息

## OperateOption

当前操作的额外选项信息，传递额外信息

## OperationNumber

当前操作，操作码

# 事件

## BeforeTrackBusinessFlow

**备注：**

反写插件的第一个事件，此时，还未分析本单的关联信息。

**触发时机：**

分析单据体的关联子表，提取关联关系（业务流程跟踪信息）之前，触发本事件；

**应用场景：**

插件取消记录业务流程跟踪信息，取消本单的全部反写

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeTrackBusinessFlowEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Cancel | bool | 是否取消本次反写 | Y：True，取消本次反写，取消后，不会分析关联关系，不建立业务流程跟踪信息，不支持联查，不反写上游单据，不关闭源单等 |

**插件示例：**

无

## BeforeCreateArticulationRow

**备注：**

构建每行单据体的关联信息数据包前，均触发本事件；

可以在此事件中，强制要求不分析当前单据体行的关联关系，从而不创建此行的业务流程跟踪数据，此行不反写。

**触发时机：**

构建下游单据体行的关联信息数据包之前

**应用场景：**

插件取消记录本行业务流程跟踪信息，取消本行反写

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeCreateArticulationRowEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Entity | Entity | 关联主实体的元数据信息 |  |
| ActiveRow | DynamicObject | 关联主实体的当前焦点行数据包 |  |
| Cancel | bool | 是否取消本行关联信息数据包的创建 | Y：True，取消本行关联数据包的创建，相当于本行无源单，不记录本行的业务流程路线和反写源单 |

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.Core.BD;  using Kingdee.BOS;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.Core.BusinessFlow.PlugIn.Args  namespace Kingdee.K3.SCM.App.DRP.ServicePlugIn.NeedApp  {  [Description("要货申请单的反写插件 V2.0正式版添加")]  public class NeedAppWriteBack : AbstractBusinessFlowServicePlugIn  {  public override void BeforeTrackBusinessFlow(BeforeTrackBusinessFlowEventArgs e)  {  //订货下达要货申请流程，不做业务流程跟踪处理  e.Cancel = true;  }  }  } | |

## BeforeWriteBack

**备注：**

单据全部反写规则，会在循环中，逐个执行。执行每个反写规则之前，均会触发本事件；

**触发时机：**

正式开始执行某条反写规则前

**应用场景：**

插件取消执行本条反写规则

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeWriteBackEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次采用的反写规则 |  |
| Cancel | bool | 是否取消本次反写 | Y：True，取消本次反写相关的所有操作；False，执行反写 |

**示例代码：**

无

## AfterCustomReadFields

**备注：**

每个反写规则前，需要整理出加载源单需要用到的字段。平台首先解析反写规则中相关控制与计算公式用到的源单字段，然后进入此插件事件，由插件补充需要加载的源单额外字段。

加载源单数据时，只加载字段清单中的字段。如果反写插件要用到的字段未加入到字段清单，则会引发中断，提示属性不存在。

**触发时机：**

正式开始执行某条反写规则前，平台加载完反写规则中使用到的相关源单字段后

**应用场景：**

插件干预在正式反写过程中需要用到的源单字段（只允许增加字段、不允许删除字段）

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterCustomReadFieldsEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次采用的反写规则 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | **入参定义** | **入参描述** | **关键属性** |
| AddFieldKey | string fieldKey | 字段元数据的唯一标示 | Y：添加字段方法 |

**需求背景：**

反写采购订单交货计划时，在取源单数据前，增加一些业务后续处理上、必要的字段

**解决方案：**

反写插件在AfterCustomReadFields事件里，调用e.AddFieldKey方法添加源单需要加载的字段

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.K3.Core.SCM;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  [Description("收料单、入库单共用反写订单交货计划的插件 V1.0正式版添加")]  public class PODeliveryPlanWriteBack : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 判断当前规则，是否为本插件需关注的规则  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  string srcFormId = e.SourceBusinessInfo.GetForm().Id  if (this.WriteBackDeliPlanCondition(srcFormId, this.OperationNumber))  {  \_thisIsMyRule = true;  }  }  public override void AfterCustomReadFields(AfterCustomReadFieldsEventArgs e)  {  if (\_thisIsMyRule == true)  {  e.AddFieldKey("FBillTypeID");  this.AddDeliveryPlanField(e);  }  }  private void AddDeliveryPlanField(AfterCustomReadFieldsEventArgs e)  {  e.AddFieldKey("FPlanDate");//要货日期（交货计划）  e.AddFieldKey("FDeliveryDate\_Plan");//计划交货日期（交货计划）  e.AddFieldKey("FPlanUnitId");//计划基本单位（交货计划）  e.AddFieldKey("FPlanQty");//计划数量（交货计划）  e.AddFieldKey("FDeliCommitQty");//已发数量（交货计划）  e.AddFieldKey("FDeliRemainQty");//剩余未发数量（交货计划）  e.AddFieldKey("FBasePlanQty");//计划数量（基本）（计划数量）  e.AddFieldKey("FBaseDeliCommitQty");//已发数量（基本）（交货计划）  e.AddFieldKey("FBaseDeliRemainQty");//剩余未发数量（基本）（交货计划）  e.AddFieldKey("FBaseUnitQty");//基本单位数量竟然没有(Entry)  e.AddFieldKey("FBASERECEIVEQTY");//累计收料数量(基本单位)  e.AddFieldKey("FBASEMRBQTY");//累计退料数量(基本单位)  }  private bool WriteBackDeliPlanCondition(string srcformId, string operNumber)  {  //判断源单是到采购订单  //因为现在是根据库存更新时点反写累计数量的，保存、审核根据库存更新时点配置只会执行一次//(无需取库存更新时点参数)  //反写分配交货计划明细行的过程是每次根据累计数量重新分配的，类似于恒等式  if (srcformId.EqualsIgnoreCase("PUR\_PurchaseOrder") &&  (  operNumber.EqualsIgnoreCase("AUDIT") || operNumber.EqualsIgnoreCase("UNAUDIT") ||  operNumber.EqualsIgnoreCase("SAVE") || operNumber.EqualsIgnoreCase("DELETE") ||  operNumber.EqualsIgnoreCase("CANCEL") || operNumber.EqualsIgnoreCase("UNCANCEL")))  {  return true;  }  else  {  return false;  }  }  }  } | |

## AfterCommitAmount

**备注：**

完成反写条目中的数值反写处理后，插件干预源单记录

**触发时机：**

对每个源单行完成反写之后，均触发本事件

**应用场景：**

完成源单数据包反写后，插件干预源单记录，这时可以填写额外字段，或修改已填写的字段值

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterCommitAmountEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 采用的反写规则 |  |
| SourceEntity | Entity | 上游单据反写的实体元数据信息 |  |
| SourceDataObject | DynamicObject | 上游单据整体数据包 |  |
| SourceActiveRow | DynamicObject | 上游单据实体焦点行的数据包 | Y：本次反写的源单数据 |
| OperationName | string | 当前操作的名称 |  |
| WriteBackRuleRow | object | 当前反写条目 |  |
| SourceCommitField | Field | 上游单据接收反写值的字段元数据信息 |  |
| WriteBackSourceRow | object | 反写条目的源单信息对象，包含上游单据标示、数据包、本次需要累加的值 |  |

**需求背景：**

发货通知单保存时，反写销售订单的关联发货通知数量（基本单位）

**实现方案：**

反写插件在AfterCommitAmount事件中，对源单此字段做特殊处理

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.BOS;  using Kingdee.BOS.Core.Const;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.BusinessEntity;  namespace Kingdee.K3.SCM.App.Sal.ServicePlugIn.DeliveryNotice  {  [Description("发货通知单的反写插件")]  public class WriteBackNew : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 系统在循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 判断当前规则是否为本插件关注的反写规则  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  if (e.Rule.Id.EqualsIgnoreCase("MyRuleId"))  {  \_thisIsMyRule = true;  }  }  public override void AfterCommitAmount(AfterCommitAmountEventArgs e)  {  if (this.\_thisIsMyRule == false) return;  if(e.OperationName.EqualsIgnoreCase("Save"))  {  string srcFormId = e.SourceBusinessInfo.GetForm().Id;  Decimal leftQty = ((Kingdee.BOS.BusinessEntity.BusinessFlow.WSRow<Kingdee.BOS.BusinessEntity.BusinessFlow.Id>)e.WriteBackSourceRow).Val;  //保存反写可出数量是同步处理设置 【关联发货通知数量（基本单位）】的值  if (srcFormId.EqualsIgnoreCase("SAL\_SaleOrder") && e.SourceCommitField.Key.EqualsIgnoreCase("FBaseCanOutQty"))  {  e.SourceActiveRow["BaseDeliJoinQty"] = Convert.ToDecimal(e.SourceActiveRow["BaseDeliJoinQty"]) + leftQty;  }  }  }  }  } | |

## BeforeCloseRow

**备注：**

达成关闭条件后，关闭上游源单行及单据的处理

**触发时机：**

每个源单行反写完毕后，都会判断是否符合行关闭条件，以关闭源单行。在每次判断源单行关闭条件之前，均会触发本事件。

**应用场景：**

可以在此事件中，取消对源单行的关闭处理

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeCloseRowEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次反写采用的反写规则 |  |
| SourceEntity | Entity | 上游单据反写实体的元数据信息 |  |
| SourceDataObject | DynamicObject | 上游单据整体数据包 |  |
| SourceActiveRow | DynamicObject | 上游单据实体焦点行数据包 |  |
| OperationName | string | 当前操作名称 |  |
| WriteBackRuleRow | object | 反写条目信息 |  |
| SourceCloseField | Field | 上游单据接收反写值的字段元数据信息 |  |
| Cancel | bool | 是否取消本次关闭操作 | Y：True，取消本次反写关闭操作；False，执行本次关闭操作 |

需求背景：

销售出库单反写销售发货通知单时，如销售发货通知单已出库关闭，取消销售出库单反写发货通知单时的关闭源单操作

实现方案：

反写插件，在BeforeCloseRow事件中，判断销售发货通知单源单单据头关闭状态字段的值，如是已关闭状态（枚举值为”B”），则取消本次反写关闭源单的操作

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.BOS;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.BusinessEntity;  namespace Kingdee.K3.SCM.App.Sal.ServicePlugIn.OutStock  {  [Description("销售出库单的反写插件 V1.0正式版添加")]  public class WriteBackNew : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 系统在循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 判断当前规则是否为本插件关注的反写规则  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  // 销售出库单反写发货通知单  if (e.Rule.Id.EqualsIgnoreCase("MyRuleId"))  {  \_thisIsMyRule = true;  }  }  public override void BeforeCloseRow(BeforeCloseRowEventArgs e)  {  //发货通知单出库已关闭，且 发货通知单已经关闭  if (\_thisIsMyRule && e.SourceDataObject["CLOSESTATUS"].ToString().EqualsIgnoreCase("B"))  {  e.Cancel = true;  }  }  }  } | |

## AfterCloseRow

**备注：**

关闭源单后，相关业务处理

**触发时机：**

每个源单行反写完毕后，都会判断是否符合行关闭条件，以关闭源单行。在处理源单行关闭逻辑之后，不管源单行是否符合关闭条件，均会触发本事件。

**应用场景：**

插件干预关闭源单后相关业务逻辑

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterCloseRowEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次反写采用的反写规则 |  |
| SourceEntity | Entity | 上游单据反写实体元数据信息 |  |
| SourceDataObject | DynamicObject | 上游单据整体数据包 | Y：支持修改 |
| SourceActiveRow | DynamicObject | 上游单据实体焦点行数据包 | Y：支持修改 |
| OperationName | string | 当前操作名称 |  |
| WriteBackRuleRow | object | 反写条目信息 |  |
| SourceCloseField | Field | 上游单据接收反写值的字段元数据信息 |  |
| EntryCloseFieldStatus | string | 填写到上游单据体行上的关闭状态值 |  |
| BillCloseFieldStatus | string | 填写到上游单整体数据包上的关闭状态值 |  |
| EntryOldCloseFieldStatus | string | 反写前，上游单据体行上的关闭状态值 |  |
| BillOldCloseFieldStatus | string | 反写前，上游整体数据上的关闭状态值 |  |

**需求背景**：

销售出库单反写销售订单，当完成源单行关闭后，根据源单行关闭状态修改源单行分录发货状态，当源单分录行基本出库数量为0时，修改源单行分录发货状态为另一个值

**实现方案**：

反写插件，在AfterCloseRow事件中，判断源单行分录关闭状态枚举值，修改源单行分录发货状态，根据源单行分录基本出库数量，修改行分录发货状态枚举值

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.BOS;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.BusinessEntity;  namespace Kingdee.K3.SCM.App.Sal.ServicePlugIn.OutStock  {  [Description("销售出库单的反写插件 V1.0正式版添加")]  public class WriteBackNew : AbstractBusinessFlowServicePlugIn  {  public override void AfterCloseRow(AfterCloseRowEventArgs e)  {  if (e.SourceBusinessInfo.GetForm().Id == "SAL\_SaleOrder")  {  if (e.EntryCloseFieldStatus.EqualsIgnoreCase("B"))  {  e.SourceActiveRow["DeliveryStatus"] = "C";  }  else if (Convert.ToDecimal(e.SourceActiveRow["BaseStockOutQty"]) == 0)  {  e.SourceActiveRow["DeliveryStatus"] = "A";  }  else  {  e.SourceActiveRow["DeliveryStatus"] = "B";  }  }  }  }  } | |

## BeforeCheckHighLimit

**备注：**

对源单进行超额检查前，判断当前检查是否执行，以及在执行前提下，一些重要属性的初始化（插件接口介绍中的关键属性）

**触发时机：**

每个源单行反写完毕后，进行超额检查之前，均会触发本事件

**应用场景：**

插件干预源单超额检查进行前，初始化重要属性，或者直接取消超额检查，允许超额

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeCheckHighLimitEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次反写采用的反写规则 |  |
| SourceEntity | Entity | 上游单据反写实体元数据信息 |  |
| SourceDataObject | DynamicObject | 上游单据整体数据包 |  |
| SourceActiveRow | DynamicObject | 上游单据实体焦点行数据包 |  |
| OperationName | string | 当前操作名称 |  |
| WriteBackRuleRow | object | 当前反写条目信息 |  |
| Cancel | bool | 是否取消超额检查 | Y： |
| MustCheck | bool | 是否必须检查 | Y： |
| OnlyCheckAtLastTime | bool | 是否最后一次反写才做超额检查 | Y：同一个分组的反写规则对源单的反写，仅在最后一次更新时，才进行超额检查；对于同一源单行，可能会被多个规则进行多次反写，并且这些反写规则的超额检查又相互影响，在此情况下，需指定这些规则的分组相同，这样，默认只需要在最后一次执行超额检查 |
| ExcessCheckFormula | string | 超额检查自定义前置条件 | Y：默认为空，以反写规则的校验条件为准，如果插件配置了此属性，会忽略反写规则的校验条件，以插件配置的这个校验条件做前置检查 |

**需求背景**：

采购订单反写采购申请，反写进行超额检查时，根据采购系统参数决定是否进行超额检查，如果允许超申请数量采购，则不进行超额检查，否则，执行超额检查

**实现方案**：

反写插件BeforeCheckHighLimit事件中，调用业务系统通用方法，查询采购系统参数，根据系统参数，是否进行本次超额检查

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.K3.Core.SCM;  using System.ComponentModel;  using Kingdee.BOS.Core.BusinessFlow.PlugIn.Args;  using Kingdee.BOS;  using Kingdee.BOS.App.Data;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.PurchaseOrder  {  [Description("采购订单反写插件")]  public class WriteBack : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 系统在循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 判断当前规则是否为本插件关注的反写规则  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  if (e.Rule.Id.EqualsIgnoreCase("MyRuleId"))  {  \_thisIsMyRule = true;  }  }  public override void BeforeCheckHighLimit(BeforeCheckHighLimitEventArgs e)  {  if (this.\_thisIsMyRule == true)  {  ICommonService service = ServiceHelper.GetService<ICommonService>();  DynamicObject reqData = e.SourceDataObject;  DynamicObjectCollection dyEntry =  reqData["ReqEntry"] as DynamicObjectCollection;  long purOrgId = 0;  if (dyEntry != null && dyEntry.Count > 0)  {  purOrgId = Convert.ToInt64(dyEntry[0]["PurchaseOrgId\_Id"]);  }  object allowPuroverReq = service.GetSystemProfile(this.Context, purOrgId, "PUR\_SystemParameter", "AllowPurOverReq", false);  if (Convert.ToBoolean(allowPuroverReq))  {  e.Cancel = true;  }  }  }  }  } | |

## AfterCheckHighLimit

**备注：**

源单超额检查后，插件干预是否抛出超额检查错误

**触发时机：**

每个源单行反写完毕后，超额检查之后，均会触发本事件

**应用场景：**

插件干预源单超额检查后，是否抛出超额检查错误，以及自定义超额检查错误提示语

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterCheckHighLimitEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| Rule | WriteBackRuleElement | 本次反写采用的反写规则 |  |
| SourceEntity | Entity | 上游单据反写实体的元数据信息 |  |
| SourceDataObject | DynamicObject | 上游单据整体数据包 |  |
| SourceActiveRow | DynamicObject | 上游单据实体焦点行数据包 |  |
| OperationName | string | 当前操作名称 |  |
| WriteBackRuleRow | object | 当前反写条目信息 |  |
| IsExcessHighLimit | bool | 是否超额提示 | Y：默认使用平台超额检查结果 |
| ErrorMessage | string | 自定义超额提示信息 | Y：编辑此属性，即忽略平台超额检查结果提示语 |

**示例代码：**

无

## BeforeSaveWriteBackData

**备注：**

当前反写规则的全部源单行，反写逻辑已处理完，准备进入保存源单数据

**触发时机：**

保存源单数据前

**应用场景：**

保存源单数据前，插件干预源单整体数据包，处理相关业务

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeSaveWriteBackDataEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| SourceDataEntitySet | ExtendedDataEntitySet | 上游单据数据包扁平化管理对象 | Y：源单整体数据包，以单据实体对象为管理维度 |

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.BusinessFlow;  using Kingdee.BOS.Core.BusinessFlow.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.BusinessEntity.BusinessFlow;  using System.ComponentModel;  namespace Kingdee.K3.FIN.AR.App.ServicePlugIn  {  /// <summary>  /// 应收单保存后反写插件  /// </summary>  /// <remarks>  /// 组织间结算清单（应收）下推应收单，应收单保存后需要重新计算明细页签的关联数量，  /// 汇总页签的关联金额两个字段  /// </remarks>  [Description("应收,应付单反写插件")]  public class ReceivableFlowPlugIn:AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 当前是否为本插件关注的反写规则?  /// </summary>  /// <remarks>  /// 从BeforeWriteBack事件开始，事件基于反写规则，循环触发；  /// 为避免重复反写，需要判断当前反写规则  /// </remarks>  private bool \_thisIsMyRule = false;    /// <summary>  /// 系统在循环中，逐一执行反写规则之前，触发本事件；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 判断当前规则是否为本插件关注的反写规则  /// </remarks>  public override void BeforeWriteBack(BeforeWriteBackEventArgs e)  {  if (e.Rule.Id.EqualsIgnoreCase("MyRuleId"))  {  \_thisIsMyRule = true;  }  }  public override void BeforeSaveWriteBackData(BeforeSaveWriteBackDataEventArgs e)  {  if (this.\_thisIsMyRule == false) return;  foreach (var item in \_dctSourceObjs)  {  this.WriteBackAmt();  }  }  // 反写应收关联金额  private void WriteBackAmt()  {  foreach (var item in \_dctSourceObjs)  {  DynamicObject srcDObj = item.Value;  DynamicObjectCollection sumEntityDObjCol = srcDObj[EntityKey\_SumEntity] as DynamicObjectCollection;  DynamicObjectCollection detailEntityDObjCol = srcDObj[EntityKey\_DetailEntity] as DynamicObjectCollection;  for (int i = 0; i < sumEntityDObjCol.Count(); i++)  {  DynamicObject sumDObj = sumEntityDObjCol[i] as DynamicObject;  decimal sum = 0;  DynamicObject sumTypeDObj = sumDObj[FieldPerty\_SumType] as DynamicObject;  long lSumTypeId = Convert.ToInt64(sumTypeDObj["ID"]);  //从明细当中汇总业务类型相同的关联应收单金额  sum = detailEntityDObjCol.Sum<DynamicObject>(o => GetReferAmt(o, lSumTypeId));  sumEntityDObjCol[i][FieldPerty\_SumAmount] = sum;  }  }  }  // 返回关联应收单金额  private Decimal GetReferAmt(DynamicObject detailEntityDObj, long lSumTypeId)  {  //业务类型  DynamicObject bizTypeDObj = detailEntityDObj[FieldPerty\_TransferBizType] as DynamicObject;  string transferDirect = detailEntityDObj["TransferDirect"].ToString().ToUpperInvariant();  long lBizTypeId = Convert.ToInt64(bizTypeDObj["ID"]);  //退货类型的要扣减，即为负数  if (lBizTypeId == lSumTypeId && transferDirect.Equals("RETURN"))  return -1 \* Convert.ToDecimal(detailEntityDObj[FieldPerty\_ReferAmount]);  else if (lBizTypeId == lSumTypeId)  return Convert.ToDecimal(detailEntityDObj[FieldPerty\_ReferAmount]);  return 0;  }  }  } | |

## AfterSaveWriteBackData

**备注：**

当前反写规则执行完，被反写的源单完成保存后，支持插件处理相关业务逻辑

**触发时机：**

源单数据保存后

**应用场景：**

保存源单数据后，插件处理其他相关业务逻辑

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterSaveWriteBackDataEventArgs.cs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceBusinessInfo | BusinessInfo | 上游单据元数据信息 |  |
| SourceDataEntitySet | ExtendedDataEntitySet | 上游单据数据包扁平化管理对象 | Y：源单整体数据包，以单据实体对象为管理维度 |

**插件示例：**

无

## FinishWriteBack

**备注：**

所有反写规则均已经执行完毕，反写插件的最后一个事件

**触发时机：**

所有反写逻辑处理完之后

**应用场景：**

反写逻辑处理完之后，插件处理相关业务逻辑

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS;  using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using Kingdee.BOS.Core.BusinessFlow.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  namespace Kingdee.K3.FIN.FA.App.ServicePlugIn.ConvertPlugIn  {  /// <summary>  /// 资产卡片反写规则通用抽象插件基类  /// </summary>  [Description("资产卡片反写规则通用抽象插件基类")]  public class AbsractCardWriteBackSourceBill : AbstractBusinessFlowServicePlugIn  {  /// <summary>  /// 资产卡片反写上流单据  /// </summary>  public override void FinishWriteBack(FinishWriteBackEventArgs e)  {  //反写盘盈盘亏单的所有EntryID  List<int> entryIds = new List<int>();  foreach (DynamicObject dyCard in this.DynamicObjects)  {  foreach (DynamicObject cardDetail in dyCard["CardDetail"] as DynamicObjectCollection)  {  foreach (DynamicObject cardDetailLink in cardDetail["FCardDetail\_Link"] as DynamicObjectCollection)  {  if (cardDetailLink["RuleId"].ToString() == "FA\_CR\_CountingSheet2Card")  {  entryIds.Add(Convert.ToInt32(cardDetailLink["SId"]));  }  }  }  }  //根据查询的盘盈分录，更新卡片其它会计政策的已建卡字段  if (entryIds.Count > 0)  {  WriteBackCountSheet(this.Context, entryIds);  }  }  /// <summary>  /// 更新盘盈盘亏单的已建卡状态  /// </summary>  private void WriteBackCountSheet(Context ctx, List<int> entryIds)  {  StringBuilder sb = new StringBuilder();  sb.AppendLine(" Merge Into T\_FA\_COUNTINGSHEETENTRY T ");  sb.AppendLine(" Using( ");  sb.AppendLine(" Select E.FID,FENTRYID,E.FASSETNUMBER,E.FALTERID,E.FOWNERORGID,E.FHASCREATEDCARD ");  sb.AppendLine(" from T\_FA\_COUNTINGSHEETENTRY E");  sb.AppendFormat(" inner join table(fn\_StrSplit(@FID, ',',1)) FS on E.FENTRYID=FS.FID ");  sb.AppendLine(" ) F on (F.FID=T.FID and F.FENTRYID<>T.FENTRYID and F.FALTERID=T.FALTERID And F.FOWNERORGID=T.FOWNERORGID and F.FASSETNUMBER=T.FASSETNUMBER) ");  sb.AppendLine(" WHEN MATCHED THEN UPDATE SET T.FHASCREATEDCARD=F.FHASCREATEDCARD ");  List<SqlParam> paramList = new List<SqlParam>();  paramList.Add(new SqlParam("@FID", KDDbType.udt\_inttable, entryIds.Distinct().ToArray()));  Kingdee.BOS.App.Data.DBUtils.Execute(Context, sb.ToString(), paramList);  }  }  } | |