操作插件示例代码

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No** | **日期** | **编制\修订** | **校对** | **批准** | **修改的章节号** |
| V1.0 | 20150420 | 丁振华 | 张晋博 | 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

目录

[操作插件示例代码 1](#_Toc416187604)

[操作插件概要说明 3](#_Toc416187605)

[如何创建一个操作插件 3](#_Toc416187606)

[操作插件概览 5](#_Toc416187607)

[如何启用操作插件 10](#_Toc416187608)

[公共属性 13](#_Toc416187609)

[BusinessInfo 13](#_Toc416187610)

[语法 13](#_Toc416187611)

[案例 – 单据已经生成凭证则禁止反审核 13](#_Toc416187612)

[Context 15](#_Toc416187613)

[语法 15](#_Toc416187614)

[案例 – 采购调价表审核时自动生效 16](#_Toc416187615)

[FormOperation 19](#_Toc416187616)

[语法 19](#_Toc416187617)

[案例 - 销售订单保存不做信用检查 19](#_Toc416187618)

[OperationResult 21](#_Toc416187619)

[语法 21](#_Toc416187620)

[案例 – 检查到负库存，提醒用户确认 21](#_Toc416187621)

[Option 23](#_Toc416187622)

[语法 23](#_Toc416187623)

[案例 – 寄售结算单只在列表上审核时，需检查价格 24](#_Toc416187624)

[基类虚方法，可捕获的事件 25](#_Toc416187625)

[OnPrepareOperationServiceOption 25](#_Toc416187626)

[语法 25](#_Toc416187627)

[案例 – 初始化库存保存 26](#_Toc416187628)

[OnPreparePropertys 32](#_Toc416187629)

[语法 32](#_Toc416187630)

[案例 – 采购价目表审核时，同步更新供应商价格 33](#_Toc416187631)

[OnAddValidators 35](#_Toc416187632)

[语法 35](#_Toc416187633)

[案例 – 单据反审核时，生成凭证校验 36](#_Toc416187634)

[BeforeExecuteOperationTransaction 40](#_Toc416187635)

[语法 41](#_Toc416187636)

[案例 – 销售价目表保存前更新数量区间价格 41](#_Toc416187637)

[BeginOperationTransaction 44](#_Toc416187638)

[语法 45](#_Toc416187639)

[案例 – 反审核采购退货单，删除关联的内部交易单据 45](#_Toc416187640)

[EndOperationTransaction 48](#_Toc416187641)

[语法 48](#_Toc416187642)

[案例 – 采购折扣表审核时，同步更新供应商折扣 49](#_Toc416187643)

[AfterExecuteOperationTransaction 51](#_Toc416187644)

[语法 51](#_Toc416187645)

[案例 – 消耗汇总审核，提示同步生成的物权转移单编号 52](#_Toc416187646)

# 操作插件概要说明

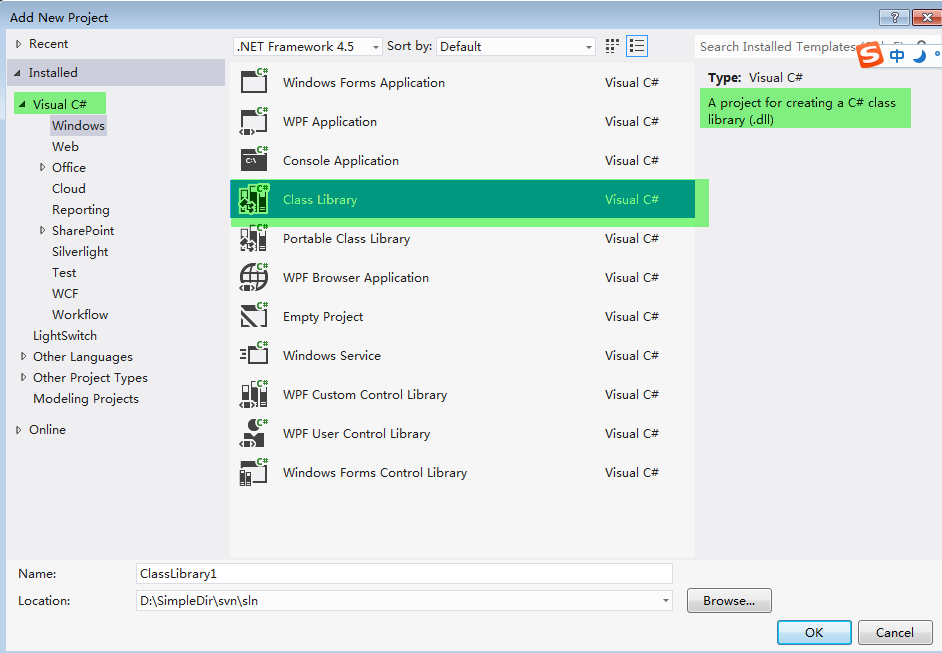
操作插件，用于对操作的执行过程进行干预、控制，以实现自定义扩展功能；

如可以为审核操作，附加插件，在审核前，进行数据检查，只有符合条件的数据才允许审核；审核后，进行自动下推，并与审核操作保持在一个事务，如果自动下推失败，则审核回滚，确保数据的完整性。

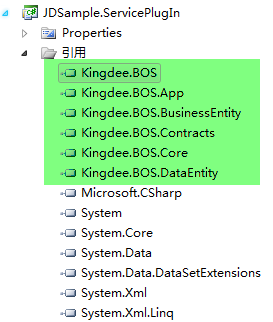
特别说明：请谨慎使用单据维护界面插件的BeforeDoOperate事件、AfterDoOperate事件，进行操作前后处理。单据维护界面的这两个事件，与操作本身并不是同一事务，如果在AfterDoOperate进行同步数据处理，如果处理失败，则无法回滚数据，从而造成数据不一致。

## 如何创建一个操作插件

* **创建类库工程，包含插件类定义**



* **引用如下类库（类库文件在K/3 Cloud站点目录\Bin\）**



必选的类库文件包括：

Kingdee.BOS.dll;

Kingdee.BOS.App.dll;

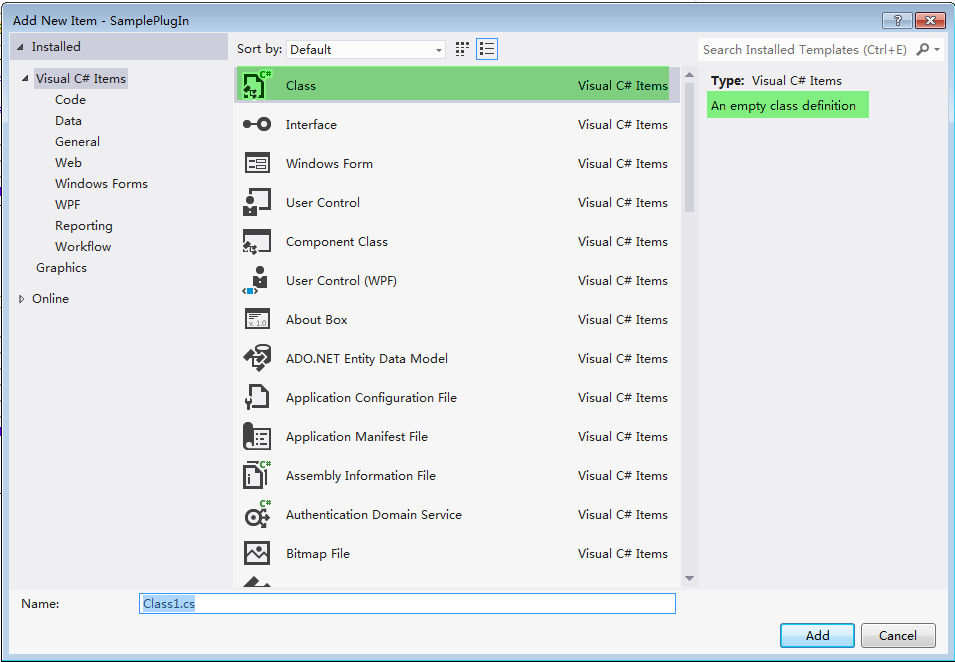
Kingdee.BOS.BusinessEntity.dll;

Kingdee.BOS.Contracts.dll;

Kingdee.BOS.Core.dll;

Kingdee.BOS.DataEntity.dll;

* **在类库中添加一个插件类**



## 操作插件概览

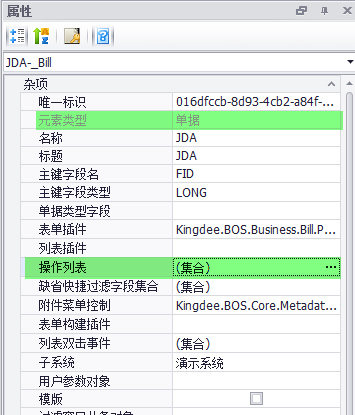
如下是一个典型的操作插件定义代码：

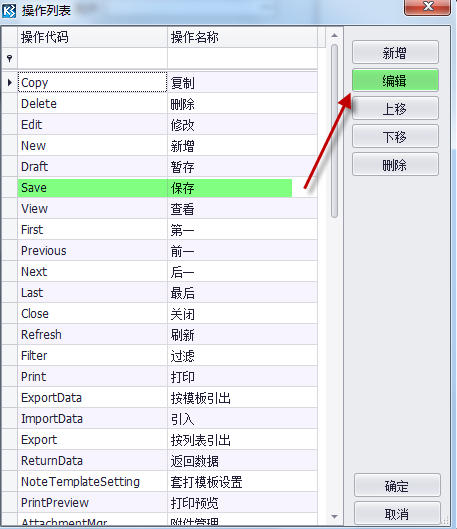
|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  namespace SamplePlugIn.ServicePlugIn  {  /// <summary>  /// 操作服务端插件示例  /// </summary>  /// <remarks>  /// 1. 示例如何强制要求加载某字段  /// 2. 示例如何注入校验器，校验器可以通用，为避免在各操作中重复编码，剥离出来，然后各操作注入剥离出的校验器  /// 3. 说明一些关键事件的用途  /// </remarks>  [System.ComponentModel.Description("操作服务端插件示例")]  public class OperationPlugSample : AbstractOperationServicePlugIn  {  /// <summary>  /// 操作初始化时，确认操作执行参数  /// </summary>  /// <param name="e"></param>  public override void OnPrepareOperationServiceOption(  OnPrepareOperationServiceEventArgs e)  {  base.OnPrepareOperationServiceOption(e);  }  /// <summary>  /// 数据加载前，确保需要的属性被加载  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 在列表上执行操作时，单据的字段并没有被完全加载。  /// 如果操作插件用到了未被加载的字段，一定会中断；  /// 本事件允许插件，强制要求加载某些字段，避免中断  /// </remarks>  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  // 如下代码行，指定字段xxxxx的Key，强制要求加载字段  e.FieldKeys.Add("xxxxx");  }  /// <summary>  /// 操作执行前，加载操作校验器  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 在系统开始执行校验前，插件可以追加自己的校验器进入操作校验器集合  /// </remarks>  public override void OnAddValidators(AddValidatorsEventArgs e)  {  // 如下代码行，示例加入自定义的校验器  // 自定义的校验器，必须从基类 Kingdee.BOS.Core.Validation.AbstractValidator 派生  e.Validators.Add(new OperationSampleValidator());  }  #region 接下来按照事件的执行事件顺序，由前向后逐一介绍各事件  /// <summary>  /// 操作执行前，事务开始前事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 1. 此事件在操作校验之后、操作实现代码之前执行  /// 2. 此事件在操作事务之前，即此事件中的数据库处理，不受操作的事务保护  /// 3. 通常此事件，也可以用来进行数据校验  /// </remarks>  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  // TODO: ....  // 如下代码，示意校验不通过时，终止操作的执行  e.CancelMessage = "测试终止操作执行";  e.Cancel = true;  }  /// <summary>  /// 操作执行前，事务开始后事件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 1. 此事件在操作校验之后  /// 2. 此事件在操作事务开始之后  /// 3. 此事件在操作执行代码之前  /// 4. 此事件中的数据库处理，受操作的事务保护  /// 5. 通常此事件，可以用来做数据准备，在操作之前，提前写数据到库  /// </remarks>  public override void BeginOperationTransaction(BeginOperationTransactionArgs e)  {  // TODO: 进行数据准备，甚至写预处理数据到数据库  }  /// <summary>  /// 操作执行后，事务结束前  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 1. 此事件在操作执行代码之后，操作的内部逻辑已经执行完毕  /// 2. 此事件在操作事务提交之前  /// 3. 此事件中的数据库处理，受操作的事务保护  /// 4. 通常此事件，可以用来做同步数据，如同步生成其他单据，而且需要受事务保护  /// </remarks>  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  // TODO：进行同步数据处理  }  /// <summary>  /// 操作执行后，事务结束后  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 1. 此事件在操作执行后，操作的内部逻辑已经执行完毕；  /// 2. 此事件在操作事务提交之后；  /// 3. 此事件中的数据库处理，不受操作的事务保护  /// 4. 通常此事件，也可以做同步数据，但是此同步数据的成功与否，不需影响操作  /// </remarks>  public override void AfterExecuteOperationTransaction(  AfterExecuteOperationTransaction e)  {  // TODO：对操作结果进行修订，使提示更加人性化  }  #endregion  }  /// <summary>  /// 操作校验器（示例）  /// </summary>  [System.ComponentModel.Description("操作校验器（示例）")]  class OperationSampleValidator : Kingdee.BOS.Core.Validation.AbstractValidator  {  #region 重载函数  /// <summary>  /// 校验主实体，以此实体数据包进行循环，逐行校验  /// </summary>  public override string EntityKey  {  get  {  // 根据实际校验要求，返回单据头或者单据体Key  return "FBillHead";  }  }  /// <summary>  /// 执行校验，把校验结果注入到validateContext中  /// </summary>  /// <param name="dataEntities"></param>  /// <param name="validateContext"></param>  /// <param name="ctx"></param>  public override void Validate(  Kingdee.BOS.Core.ExtendedDataEntity[] dataEntities,  Kingdee.BOS.Core.Validation.ValidateContext validateContext,  Kingdee.BOS.Context ctx)  {  foreach (var dataEntity in dataEntities)  {  // TODO: 逐个数据包执行校验代码    // 如下代码，示意如何注入校验提示，后续操作，会自动避开校验没通过的数据包  validateContext.AddError(dataEntity.DataEntity,  new Kingdee.BOS.Core.Validation.ValidationErrorInfo(  "出错字段.Key", // 出错的字段Key，可以空  Convert.ToString(dataEntity.DataEntity[0]), // 数据包内码，必填，后续操作会据此内码避开此数据包  dataEntity.DataEntityIndex, // 出错的数据包在全部数据包中的顺序  dataEntity.RowIndex, // 出错的数据行在全部数据行中的顺序，如果校验基于单据头，此为0  "E1", // 错误编码，可以任意设定一个字符，主要用于追查错误来源  "错误的详细提示信息", // 错误的详细提示信息  "错误摘要", // 错误的简明提示信息  Kingdee.BOS.Core.Validation.ErrorLevel.Error)); // 错误级别：警告、错误...  }  }  #endregion 重载函数  }  } | |

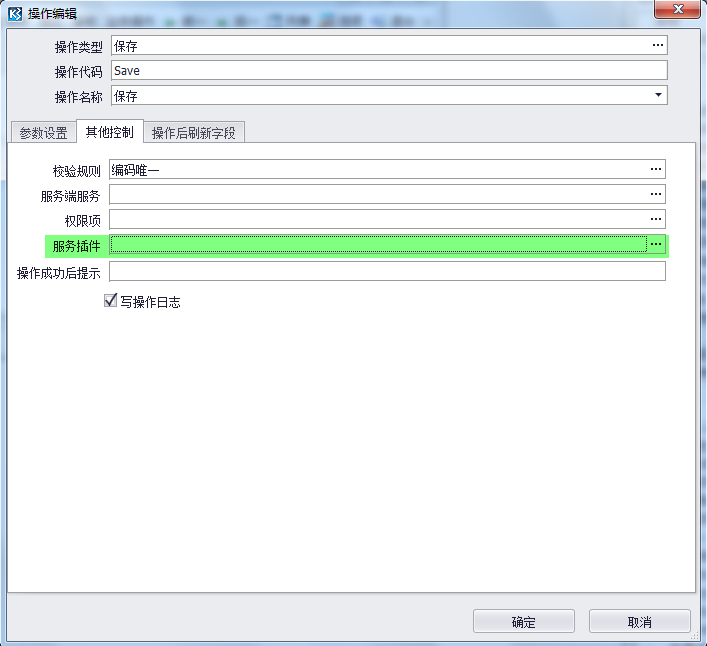
## 如何启用操作插件

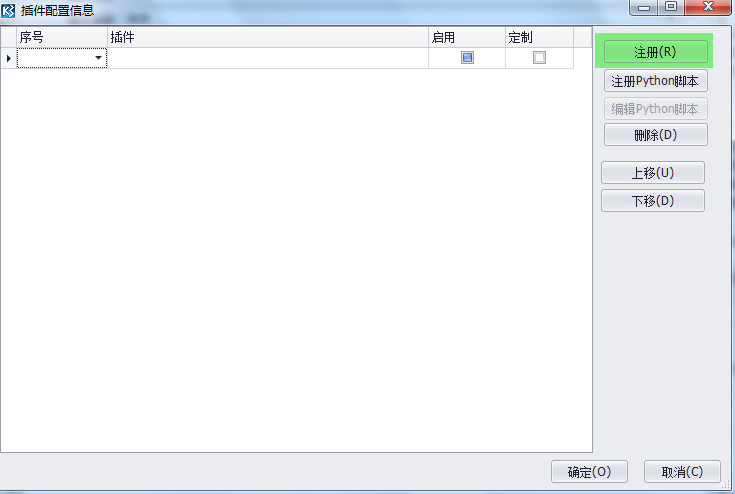
插件编写完毕，类库编译通过后，把类库文件复制到K/3 Cloud站点目录的\Bin子目录；

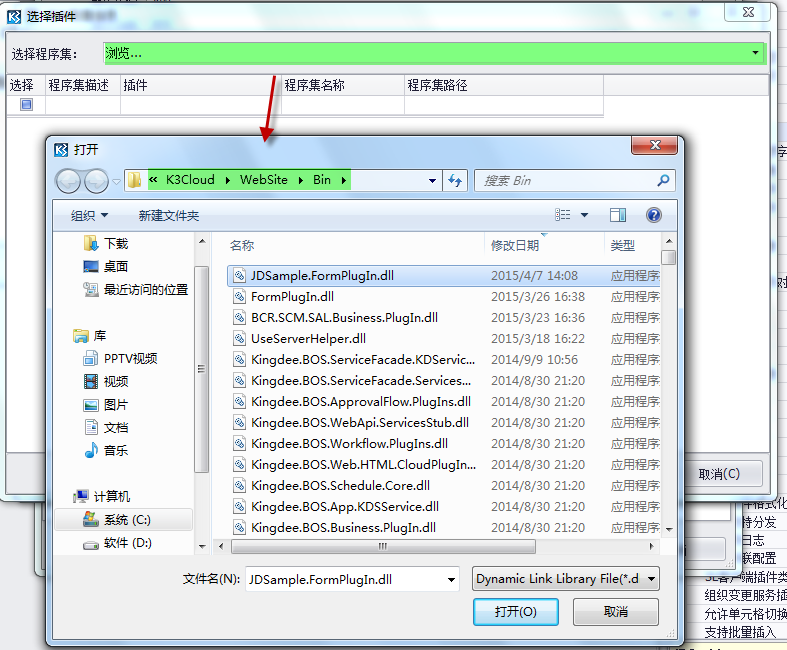
然后进入BOS设计器，打开单据操作配置界面，配置服务插件：











# 公共属性

## BusinessInfo

单据元数据对象。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public BusinessInfo BusinessInfo { get; private set; } | |

**备注**

可以通过此属性对象，获取单据的各种属性配置，以及所含单据体、字段信息。

对于一些公用类的操作插件，不能预先知道将要被应用到那种单据、那种操作上，此时从BusinessInfo对象中获取必要的信息就很重要。

### 案例 – 单据已经生成凭证则禁止反审核

**代码来源**

Kingdee.K3.SCM.App.ServicePlugIn.VoucherControl.UnAuditVoucherControl

**需求背景**

供应链单据，已经生成凭证，且凭证进行了更改、审核，则禁止反审核单据；如果生成的凭证没有变化，则同步删除凭证；

**实现方案**

提供一个供应链反审核通用插件，为全部单据服务，BusinessInfo中的单据FormId属性，作为最关键的参数传递。

在操作前，调用一个通用的校验器，判断单据是否已经生成凭证，该校验器要求传入当前单据的FormId，此信息需要到BusinessInfo属性中获取；

在标准反审核操作前(事务内)，调用一个通用服务，同步删除凭证，此服务要求传入单据FormId以及单据Fid。

**关键字**

单据FormId

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Validation;  using Kingdee.K3.Core.FIN;  using Kingdee.BOS.Core;  using Kingdee.K3.SCM.App.Validator;  using Kingdee.K3.BD.Contracts.FIN;  using Kingdee.BOS.Orm.DataEntity;  namespace Kingdee.K3.SCM.App.ServicePlugIn.VoucherControl  {  [Description("反审核插件，处理和凭证有关")]  public class UnAuditVoucherControl: AbstractOperationServicePlugIn  {  public override void OnAddValidators(AddValidatorsEventArgs e)  {  //添加凭证检查校验器：如果已经生成凭证且凭证已经更改或审核则单据不允许反审核  HasVoucherValidator voucherValidator = new HasVoucherValidator();  voucherValidator.EntityKey = "FBillHead";  voucherValidator.BillFormId = this.**BusinessInfo**.GetForm().Id;  e.Validators.Add(voucherValidator);  }  public override void BeginOperationTransaction(BeginOperationTransactionArgs e)  {  if (e.DataEntitys.Count() < 1)  {  return;  }  string formId = this.BusinessInfo.GetForm().Id;  List<long> lstBillID = new List<long>();  List<long> billIdList = new List<long>();  List<DynamicObject> dataEntitys = new List<DynamicObject>();  foreach (var item in e.DataEntitys)  {  lstBillID.Add(Convert.ToInt64(item["Id"]));  }  //以下是处理 如果生成的凭证未做任何处理则反审核时删除凭证。  var service = BD.Contracts.ServiceFactory.GetLocalService<IFINServiceForCommon>();  IEnumerable<BizBillVoucherInfor> lstBizBillInfor = service.GetBizBillVoucherInfor(this.Context, formId, lstBillID);  DynamicObject obj;  foreach (var bizBillInfor in lstBizBillInfor)  {  if (bizBillInfor.CanUnAudit)  {  long pkId = bizBillInfor.BillPKID;  billIdList.Add(pkId);  obj = (from p in e.DataEntitys  where Convert.ToInt64(p["Id"]) == pkId  select p).FirstOrDefault();  dataEntitys.Add(obj);  }  }  if (billIdList.Count > 0)  {  service.DeleteBizBillBizVoucher(this.Context, formId, billIdList);  }  e.DataEntitys = dataEntitys.ToArray();  }  }  } | |

## Context

上下文对象；

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public Context Context { get; private set; } | |

**备注**

上下文对象，包含了数据库连接信息，在调用各种读取数据库的接口函数时，需用本对象作为参数。

### 案例 – 采购调价表审核时自动生效

**代码来源**

Kingdee.K3.SCM.App.Pur.ServicePlugIn.PAT.Audit

**需求背景**

采购调价表执行审核操作时，需要同步执行生效处理；

**实现方案**

在处理操作前，先明确要求加载采购调价表的某些关键字段，后面处理数据时，需要用到。

在操作前（事务内），先调用采购调价表的生效操作，自动生效，如果失败，则抛出错误。

如果采购调价表生效成功，则继续执行审核操作。

在此过程中，有几处需要用到Context属性，如加载采购调价表元数据、调用采购调价表的生效操作、记录错误日志等；

**关键字**

加载元数据 状态转换 写日志

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.App;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.Metadata.DataEntity;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Core.Log;  using Kingdee.BOS;  using System.Data;  using Kingdee.BOS.App.Data;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.ServiceHelper;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.PAT  {  public class Audit : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {    e.FieldKeys.Add("FDate");  e.FieldKeys.Add("FPurchaseOrgId");  e.FieldKeys.Add("FBillNo");  e.FieldKeys.Add("FPaReason");  e.FieldKeys.Add("FName");  e.FieldKeys.Add("FAfterPrice");  e.FieldKeys.Add("FUpPrice");  e.FieldKeys.Add("FDownPrice");  e.FieldKeys.Add("FEffectiveDate");  e.FieldKeys.Add("FExpiryDate");  e.FieldKeys.Add("FMaterialId");  e.FieldKeys.Add("FMaterialTypeId");  e.FieldKeys.Add("FPriceListId");  e.FieldKeys.Add("FBillHead");  e.FieldKeys.Add("FIsPriceListPush");  e.FieldKeys.Add("FToQty");  e.FieldKeys.Add("FPATIdentity");  base.OnPreparePropertys(e);  }  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  ISaveService service = ServiceHelper.GetService<ISaveService>();  ISetStatusService setStatusService = Kingdee.BOS.Contracts.ServiceFactory.GetSetStatusService(this.**Context**);  IMetaDataService metaService = ServiceHelper.GetService<IMetaDataService>();  ILogService logService = ServiceHelper.GetService<ILogService>();  FormMetadata meta = (FormMetadata)metaService.Load(this.**Context**, SCMFormIdConst.PUR\_PAT);  BusinessInfo info = meta.BusinessInfo;  List<KeyValuePair<object, object>> kvp = new List<KeyValuePair<object, object>>();  foreach (DynamicObject item in e.DataEntitys)  {  long fid = Convert.ToInt64(item["Id"]);  DynamicObject dyOrg = item["PurchaseOrgId"] as DynamicObject;  long orgId = dyOrg == null ? 0 : Convert.ToInt64(dyOrg["Id"]);  bool isEffectWithoutClick = Convert.ToBoolean(SystemParameterServiceHelper.GetParamter(this.**Context**, orgId, 0, SCMFormIdConst.PUR\_PurchaseParameter, "AdjustEffectWithoutClick"));  if (isEffectWithoutClick)  {  kvp.Add(new KeyValuePair<object, object>(fid, ""));  }  }  if (kvp.Count > 0)  {  IOperationResult result = setStatusService.SetBillStatus(  this.**Context**, info, kvp, null, "Effectitve");  if (!result.IsSuccess)  {  StringBuilder sb = new StringBuilder();  foreach (ValidationErrorInfo errorInfo in result.ValidationErrors)  {  sb.Append(errorInfo.Message);  sb.Append("\r\n");  LogObject logObj = new LogObject()  {  Description = errorInfo.Message,  Environment = OperatingEnvironment.BizOperate,  OperateName = "调价表生效失败",  ObjectTypeId = K3.Core.SCM.SCMFormIdConst.PUR\_PriceCategory  };  logService.WriteLog(this.Context, logObj);  }  throw new KDException("","调价表生效失败。原因如下：调价表的" + sb);  }  }  }  }  } | |

## FormOperation

当前执行的操作。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public FormOperation FormOperation { get; private set; } | |

**备注**

操作插件，可以基于此属性，判断出正在执行的操作类型，从而做出相应的处理。

### 案例 - 销售订单保存不做信用检查

**代码来源**

Kingdee.K3.SCM.App.Credit.Core.AppBusinessService.UpdateCreditAmount

**需求背景**

销售订单提交、关闭时，需要进行信用检查，更新客户信用额度；

**实现方案**

定制一个通用的信用检查、额度更新服务，供销售订单及其子类单据调用，可能会被配置到保存、提交、关闭、审核等各种操作上。

需要用到FormOperation属性，判断当前操作类型，如果被配置保存操作上，则强制忽略检查。

此示例代码，来自于服务端服务(AbstractAppBusinessService)，但FormOperation用法相同。

**关键字**

操作类型

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.K3.Core.SCM;  namespace Kingdee.K3.SCM.App.Credit.Core.AppBusinessService  {  public class UpdateCreditAmount : AbstractAppBusinessService  {  public override void PreparePropertys(List<string> fieldKeys)  {  base.PreparePropertys(fieldKeys);  // 无关逻辑，代码略去  }  public override void DoActionBatch(AppBusinessServiceArgs e)  {  //初始化变量 及有效性检查  if (!InitialVariable(e)) return;  // 无关逻辑，代码略去  base.DoActionBatch(e);  }  /// <summary>  /// 初始化变量 当发现不需要再执行服务时直接return false  /// </summary>  /// <param name="e"></param>  private bool InitialVariable(AppBusinessServiceArgs e)  {  // 无关逻辑，代码略去    return ValidateData(e);  }  private bool ValidateData(AppBusinessServiceArgs e)  {  //保存且不是变更调用 不做信用检查和更新  if (e.**FormOperation**.Id.EqualsIgnoreCase(OperationNumber.Save))  {  return false;  }  // 无关逻辑，代码略去  return true;  }  }  } | |

## OperationResult

操作执行结果对象。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public IOperationResult OperationResult { get; private set; } | |

**备注**

本属性对象，在操作开始执行前，即会被构建，执行过程中，会不断收集执行结果，比如校验执行完毕后，有那些数据校验失败，基于什么原因校验失败，都会被收集进此操作结果；

最终那些数据执行成功，操作是否成功，也会被记录于此。

系统操作引擎，会自动更新本对象属性值，如果插件更改了本对象的属性值，可能会被覆盖。

### 案例 – 检查到负库存，提醒用户确认

**代码来源**

Kingdee.K3.SCM.App.Core.AppBusinessService.CheckInvMinusService

**需求背景**

出库单据保存、审核时，需要检查库存，如果出现了负库存，需要提醒用户确认，如果用户选择取消，则放弃保存、审核；

**实现方案**

定制通用的库存检查服务，供各种出库单调用。

如果检查出了负库存，则向操作结果，添加交互提示信息，以供用户确认。

此示例代码，来自于服务端服务(AbstractAppBusinessService)，但OperationResult用法相同。

**关键字**

交互提示

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| private void CheckMinusInv(AppBusinessServiceArgs e)  {  string tranId = "";  this.Option.TryGetVariableValue<string>("OperationId", out tranId);  if (string.IsNullOrWhiteSpace(tranId))  {  //throw new KDException("","未知事务ID，该服务需要配置在库存更新服务后，不能被独立调用！");  //为了减少负库存检查空跑的概率，更新库存服务里到有实际更新SQL时才设置该Id，如果抛错会造成误判  //也就是说如果该操作没有执行任何库存更新的SQL，那么这个Id就为空，也就没必要检查负库存了  return;  }    CommonService comService = new CommonService();  bool usePLNReserve = comService.IsUsePLNReserve(e.Context);  StringBuilder sBuilder = new StringBuilder();  List<string> sList = new List<string>();  DataTable dataTb = GetMinusInvDataSet(e.Context, tranId, usePLNReserve, e.BusinessInfo, e.FormBusinessService.PreCondition);  if (dataTb == null)  {  return;  }  //拼接错误消息并返回错误类型  DynamicObject errObj = GetMinusCheckErrInfo(e.Context, dataTb.Rows.ToArray<DataRow>(), true, usePLNReserve);  //构造操作结果  if (errObj != null)  {  BuildOperationResult(e.Context, errObj);  }  //打包处理弱预留提示信息  string sql = "SELECT COUNT(1) FROM T\_PLN\_RESERVEWARNING WHERE FTRANID = @TranId ";  List<SqlParam> paras = new List<SqlParam>();  paras.Add(new SqlParam("@TranId", DbType.AnsiString, tranId));  if (DBUtils.ExecuteScalar<int>(e.Context, sql, 0, paras.ToArray()) > 0)  {  if (this.**OperationResult** != null)  {  this.**OperationResult**.CustomMessageFormId = "PLN\_ReserveWarning";  this.OperationResult.CustomMessageFormParameter.CustomParams.Add("TranId", tranId);  this.**OperationResult**.CustomMessageFormParameter.FormId = "PLN\_ReserveWarning";  this.OperationResult.IsShowMessage = true;  }  }  } | |

## Option

操作附加参数对象，相当于一个定制的参数字典。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public OperateOption Option { get; private set; } | |

**备注**

调用操作处理，可以把一些特别的参数，封装到Option对象中，传入到操作引擎。

操作插件可以访问此属性对象，拿到传入的特别参数；

### 案例 – 寄售结算单只在列表上审核时，需检查价格

**代码来源**

Kingdee.K3.SCM.App.Sal.ServicePlugIn.ConsignSettle.Audit

**需求背景**

寄售结算单，在单据查看界面进行审核时，已经检查了价格，在审核操作时，无需再此检查；但是，在列表上进行审核时，则需要在审核操作中，特别检查价格。

**实现方案**

在单据查看界面，检查完价格之后，调用审核操作之前，把一个标志变量IsFromBill，放到Option中。

审核插件执行时，判断Option中的IsFromBill标志变量值，如果为true，则不检查价格；

**关键字**

操作附加参数传递 操作定制参数传递

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  namespace Kingdee.K3.SCM.App.Sal.ServicePlugIn.ConsignSettle  {  public class Audit : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  e.FieldKeys.Add("FPrice");  e.FieldKeys.Add("FTaxPrice");  e.FieldKeys.Add("FPriceUnitId");  e.FieldKeys.Add("FTaxNetPrice");  e.FieldKeys.Add("FLimitDownPrice");  e.FieldKeys.Add("FIsIncludedTax");  e.FieldKeys.Add("FMaterialId");  e.FieldKeys.Add("FUnitId");  e.FieldKeys.Add("FSettleCurrId");  e.FieldKeys.Add("FIsFree");  e.FieldKeys.Add("FPriceCoefficient");  }  public override void OnAddValidators(AddValidatorsEventArgs e)  {  bool isFromBill = this.**Option**.GetVariableValue<bool>("IsFromBill", false);  if ( !isFromBill)  {  Kingdee.K3.SCM.App.Sal.ServicePlugIn.SaleOrder.SaleLimitPriceCtrlValidator  salePrice = new Kingdee.K3.SCM.App.Sal.ServicePlugIn.SaleOrder.SaleLimitPriceCtrlValidator();  salePrice.AlwaysValidate = true;  salePrice.EntityKey = "FBillHead";  salePrice.PriceCtrlTime = "AUDIT";  salePrice.BillFormID = "SAL\_ConsignmentSettle";  salePrice.OrderEntryName = "SAL\_CONSIGSETTLEENTRY";  salePrice.OrderFinancialEntryName = "SaleCconsignSettleFinance";  e.Validators.Add(salePrice);  }  }  }  } | |

# 基类虚方法，可捕获的事件

操作插件基类，提供了多个按顺序执行的虚方法，插件按需要重载这些方法，即可适当的实际，对操作的执行进行干预；

本文将按照各虚方法的执行顺序进行介绍。

## OnPrepareOperationServiceOption

初始化操作执行参数时触发。通常在此事件中，设置操作执行选项；

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnPrepareOperationServiceOption(OnPrepareOperationServiceEventArgs e) | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| Context Context | 上下文 |
| bool SurportBatchTransaction | 整批数据放在一个事务中处理，性能比较好；  默认为true，如果设置为false，则会为每条数据单独开启一个事务，单条数据操作失败，不影响其他数据； |
| bool SupportTransaction | 操作处理时，开启一个事务进行数据保护；  默认为true，如果设置为false，则不开启事务； |
| bool AllowSetOperationResult | 是否允许平台自动添加操作成功的消息；  默认为true。业务上有非事务型的操作，需要设置此参数为false |
| bool RollbackWhenValidationError | 设置为true时，操作校验，发现其中有一条单据不符合条件，即会直接退出操作，整批单据均不处理；  默认为false，操作校验，发现有不符合条件的单据时，不直接退出，但会过滤掉不符合条件的单据，继续其余符合条件的单据。  通过设置此选项为true，可以实现整批单据同进同出 |

**备注**

通过设置本方法参数对象的一些属性值，控制操作执行策略；

### 案例 – 初始化库存保存

**代码来源**

Kingdee.K3.SCM.App.Stock.ServicePlugIn.InvInitBillService.Save

**需求背景**

初始化库存单据，部分检查无法在保存前进行，需要在保存结束后，数据已经入库，执行检查。

如果某条数据保存检查失败，需要撤销本条数据的保存，但是之前已经保存成功的数据，则不受影响。

**实现方案**

在操作执行前，设置操作执行参数，要求单据逐条保存，逐条开启事务。

在事务结束时，检索数据库中是否产生了重复库存条目，如果存在，说明本单据与历史保存的初始化库存单据有数据冲突，本单据不允许保存，需提示用户；

**关键字**

逐条执行操作 操作结束后检查 给出交互提示

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.Interaction;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Orm.Metadata.DataEntity;  using Kingdee.BOS.Resource;  using Kingdee.BOS.Util;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.Core.SCM.STK;  using Kingdee.K3.SCM.App.Validator;  using Kingdee.K3.SCM.Contracts;  using System;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  namespace Kingdee.K3.SCM.App.Stock.ServicePlugIn.InvInitBillService  {  /// <summary>  /// 初始库存单据保存操作服务端插件  /// </summary>  public class Save : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  // 无关代码，略过  }  /// <summary>  /// 设置选项  /// </summary>  /// <param name="e"></param>  public override void OnPrepareOperationServiceOption(  OnPrepareOperationServiceEventArgs e)  {  base.OnPrepareOperationServiceOption(e);  //为了在EndOperationTransaction里检查数据抛出交互异常的时候  //只回滚当前单据的事务，这里设置为不支持批量事务，这样BOS会  //循环为每一张单据创建事务调用操作  e.SupportTransaction = true;  e.SurportBatchTransaction = false;  }  public override void OnAddValidators(AddValidatorsEventArgs e)  {  // 无关代码，略过  }  /// <summary>  /// 初始库存单据保存后校验数据重复性  /// </summary>  /// <param name="e"></param>  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  if (e.DataEntitys.Count<DynamicObject>() > 0)  {  //检查数据重复性  CheckDuplicateData(e);  }  }  /// <summary>  /// 检查重复数据  /// </summary>  /// <param name="e"></param>  private void CheckDuplicateData(EndOperationTransactionArgs e)  {  //先获得单据头关键字段完全相同的单据列表  StringBuilder builder = new StringBuilder();  string billIds = "";  foreach (DynamicObject obj in e.DataEntitys)  {  builder.AppendFormat(",{0}", (obj["Id"] == null ? "0" : obj["Id"]));  }  //限定获得的单据内码的范围，只取这次保存的，历史数据的忽略  billIds = builder.ToString(1, builder.Length - 1);  string sql = string.Format(@"SELECT t1.FID,t4.FENTRYID  FROM T\_STK\_INVINIT t1  INNER JOIN  (SELECT COUNT(1) AS FCOUNT, t2.FSTOCKID, t2.FKEEPERID, t2.FOWNERID, t2.FSTOCKORGID,  t2.FOWNERTYPEID, t2.FKEEPERTYPEID,t3.FSTOCKSTATUSID,t3.FPROJECTNO,t3.FMTONO,t3.FBOMID,  t3.FProduceDate,t3.FSECUNITID,t3.FExpiryDate,t3.FUNITID,  t3.FMATERIALID,t3.FStockLocId,t3.FLOT,t3.FAUXPROPID  FROM T\_STK\_INVINIT t2 INNER JOIN T\_STK\_INVINITDETAIL T3 ON t2.FID = t3.FID  GROUP BY t2.FSTOCKID, t2.FKEEPERID, t2.FOWNERID, t2.FSTOCKORGID,  t2.FOWNERTYPEID, t2.FKEEPERTYPEID,t3.FSTOCKSTATUSID,t3.FPROJECTNO,t3.FMTONO,t3.FBOMID,  t3.FProduceDate,t3.FSECUNITID,t3.FExpiryDate,t3.FUNITID,  t3.FMATERIALID,t3.FStockLocId,t3.FLOT,t3.FAUXPROPID  HAVING COUNT(1) > 1 ) ta ON t1.FSTOCKID = ta.FSTOCKID AND t1.FKEEPERID = ta.FKEEPERID  AND t1.FOWNERID = ta.FOWNERID AND t1.FSTOCKORGID = ta.FSTOCKORGID  AND t1.FOWNERTYPEID = ta.FOWNERTYPEID AND t1.FKEEPERTYPEID = ta.FKEEPERTYPEID  INNER JOIN T\_STK\_INVINITDETAIL t4 ON t1.FID = t4.FID AND ta.FSTOCKSTATUSID = t4.FSTOCKSTATUSID  AND ta.FPROJECTNO = t4.FPROJECTNO AND ta.FMTONO = t4.FMTONO AND ta.FBOMID = t4.FBOMID  AND (ta.FProduceDate = t4.FProduceDate OR (ta.FProduceDate IS NULL AND t4.FProduceDate IS NULL))  AND (ta.FExpiryDate = t4.FExpiryDate OR (ta.FExpiryDate IS NULL AND t4.FExpiryDate IS NULL))  AND ta.FUNITID = t4.FUNITID AND ta.FSECUNITID = t4.FSECUNITID  AND ta.FMATERIALID = t4. FMATERIALID AND ta.FStockLocId = t4.FStockLocId AND ta.FLOT = t4.FLOT  AND ta.FAUXPROPID = t4.FAUXPROPID  WHERE t1.FID IN ({0}) ", billIds);  Dictionary<long, long> idInfos = new Dictionary<long, long>();  using (IDataReader dr = DBUtils.ExecuteReader(this.Context, sql))  {  while (dr.Read())  {  idInfos[Convert.ToInt64(dr["FENTRYID"])] = Convert.ToInt64(dr["FID"]);  }  dr.Close();  }  if (idInfos.Count < 1)  {  return;  }  \_ownerTypeField = this.BusinessInfo.GetField("FOwnerTypeIdHead") as ItemClassTypeField;  \_keeperTypeField = this.BusinessInfo.GetField("FKeeperTypeId") as ItemClassTypeField;  \_auxPropertyField = this.BusinessInfo.GetField("FAuxPropId") as RelatedFlexGroupField;  \_locPlaceField = this.BusinessInfo.GetField("FStockLocId") as RelatedFlexGroupField;  /\*用k3显示器显示错误信息\*/  string formTitle = "系统中已存在相同的初始库存数据或者单据中存在重复的明细数据分录,校验不通过";  string title = "库存组织~|~仓库~|~仓位~|~物料~|~辅助属性~|~批号~|~生产日期~|~有效期至~|~库存状态~|~BOM版~|~货主类型~|~货主~|~保管者类型~|~保管者";  K3DisplayerModel model = K3DisplayerModel.Create(this.Context, title);  model.Option.SetVariableValue(K3DisplayerModel.CST\_FormTitle, formTitle);  model.OKButton.Visible = false;  model.CancelButton.Caption = new BOS.LocaleValue(Kingdee.BOS.Resource.ResManager.LoadKDString("确定", "004004030005788", Kingdee.BOS.Resource.SubSystemType.SCM), this.Context.UserLocale.LCID);  foreach (DynamicObject dyObj in e.DataEntitys)  {  DynamicObjectCollection dyEntrys = dyObj["InvInitDetail"] as DynamicObjectCollection;  foreach (DynamicObject dyEntry in dyEntrys)  {  long id = 0;  idInfos.TryGetValue(Convert.ToInt64(dyEntry["Id"]), out id);  if (id == Convert.ToInt64(dyObj["Id"]))  {  //获取错误消息  List<string> errMsgList = GetFullDataDupErrMsg(dyObj, dyEntry);  model.AddMessage(string.Join("~|~", errMsgList));  }  }  }  KDInteractionException ie = new KDInteractionException(this.Option, "StockInvInitBillDuplicateDataCheckError");  ie.InteractionContext.InteractionFormId = FormIdConst.BOS\_K3Displayer;  ie.InteractionContext.K3DisplayerModel = model;  ie.InteractionContext.IsInteractive = false;  ie.InteractionContext.SimpleMessage = formTitle;    throw ie;  }  /// <summary>  /// 获取当前单据单据行的错误信息  /// </summary>  /// <param name="dyObj"></param>  /// <param name="dyEntry"></param>  /// <returns></returns>  private List<string> GetFullDataDupErrMsg(DynamicObject dyObj, DynamicObject dyEntry)  {  DynamicObject objFieldValue = null;  List<string> msgStr = new List<string>();  objFieldValue = dyObj["StockOrgID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  objFieldValue = dyObj["StockID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  objFieldValue = dyEntry["StockLocId"] as DynamicObject;  string locName = \_locPlaceField.GetDisplayName(objFieldValue);  msgStr.Add(locName == null ? "" : locName);  objFieldValue = dyEntry["MaterialID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  objFieldValue = dyEntry["AuxPropId"] as DynamicObject;  string auxProperyName = \_auxPropertyField.GetDisplayName(objFieldValue);  msgStr.Add(auxProperyName == null ? "" : auxProperyName);  msgStr.Add(dyEntry["Lot\_Text"] == null ? "" : dyEntry["Lot\_Text"].ToString());  msgStr.Add(dyEntry["ProduceDate"].IsNullOrEmptyOrWhiteSpace() ? "" : Convert.ToDateTime(dyEntry["ProduceDate"]).ToShortDateString());  msgStr.Add(dyEntry["ExpiryDate"].IsNullOrEmptyOrWhiteSpace() ? "" : Convert.ToDateTime(dyEntry["ExpiryDate"]).ToShortDateString());  objFieldValue = dyEntry["StockStatusID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  objFieldValue = dyEntry["BOMID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  string ownerTypeName = \_ownerTypeField.GetEnumItemName(dyObj["OwnerTypeIdHead"].ToString());  msgStr.Add(ownerTypeName);  objFieldValue = dyObj["OwnerIdHead"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  string keeperTypeName = \_keeperTypeField.GetEnumItemName(dyObj["KeeperTypeID"].ToString());  msgStr.Add(keeperTypeName);  objFieldValue = dyObj["KeeperID"] as DynamicObject;  msgStr.Add(objFieldValue == null ? "" : objFieldValue["Name"].ToString());  return msgStr;  }  } | |

## OnPreparePropertys

在操作初始化之后，加载待处理的单据数据之前触发。

用于指定在加载单据数据时，需加载的字段；

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnPreparePropertys(PreparePropertysEventArgs e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| IList<string> FieldKeys | 需加载的字段Key列表；  需加载的字段，把其Key添加到此集合 |
|  |  |

**备注**

单据列表，通常仅会加载单据的部分字段，在列表上执行单据操作，操作服务端，仅能够接收到单据的内码，而没有实际数据。

但操作处理，会进行数据校验、字段更新，均需要用到内码之外的其他字段值，因此会要求操作引擎加载单据的数据包。

操作引擎采用按需加载的方式，自动扫描校验条件中需要用到的字段，加载这些字段的数据，其他无关的字段，均不会加载。

操作插件中，需要用到的字段，操作引擎不会扫描到，因此，需要插件自行明确声明需要加载那些字段。

如果插件需要用到的字段，没有在本事件中申明加载，到数据包取字段值时，会触发”属性不存在”的错误。

向参数对象e.FieldList集合中，添加字段Key，要求加载字段值；

### 案例 – 采购价目表审核时，同步更新供应商价格

**代码来源**

Kingdee.K3.SCM.App.Pur.ServicePlugIn.PriceList.Audit

**需求背景**

采购价目表审核时，需要把采购价目表对应的供应商价格，更新到供应商基础资料的属性上。

**实现方案**

在审核操作前，要求加载采购价目表上的供应商、默认价格等字段值。

在操作事务中，审核完毕后，同步更新采购价目表上各供应商的默认价格。

**关键字**

属性不存在 使用字段值 加载字段值

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Orm.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Resource;  using Kingdee.BOS.Core;  using Kingdee.BOS.App.Data;  using Kingdee.BOS;  using System.Data;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.PriceList  {  /// <summary>  /// 采购价目表服务端审核操作  /// </summary>  public class Audit : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  e.FieldKeys.Add("FSupplierID");  e.FieldKeys.Add("FSupplierLocId");  e.FieldKeys.Add("FDefPriceListId");//是否默认价目表  e.FieldKeys.Add("FcreateorgId");  e.FieldKeys.Add("FuseorgId");  }  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  long supplierId = 0;  long priceListId = 0;  bool defPriceListId = false;  List<string> sqlArray = new List<string>();  foreach (DynamicObject entity in e.DataEntitys)  {  if (entity!=null)  {  supplierId = Convert.ToInt64(entity["supplierId\_Id"]);  priceListId = Convert.ToInt64(entity["Id"]);  defPriceListId = Convert.ToBoolean(entity["DefPriceListId"]);  if (priceListId>0 && supplierId>0 && defPriceListId)  {  string sql = string.Format(@" UPDATE T\_BD\_SUPPLIERBUSINESS AS P  SET FPRICELISTID= {0}  WHERE P.FSupplierid={1} ",priceListId.ToString(),supplierId.ToString());  sqlArray.Add(sql);  }  }  }  if (sqlArray.Count > 0)  {  DBUtils.ExecuteBatch(this.Context, sqlArray, 100);  }  }  }  } | |

## OnAddValidators

在单据数据加载后，数据校验前触发；

用于注册自定义的操作校验器，增加校验；或者移除预置的校验器，避开校验；

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnAddValidators(AddValidatorsEventArgs e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| DynamicObject[] DataEntities | 待校验的单据数据包 |
| List<Kingdee.BOS.Core.Validation.AbstractValidator> Validators | 校验器集合 |

**备注**

操作引擎会在一个事务中，进行数据的更新，如果更新成功则全部成功，如果某条数据操作失败，则全部失败。

实际业务中，需要把操作失败的数据，单独隔离出来，其他数据，顺利操作成功。为实现此需求，需要在操作事务之前，把不允许操作的数据检查出来，并剔除出数据集合，剩下符合条件，允许操作的数据，丢进事务，进行操作。

本事件，允许向操作中，添加自定义的校验规则，对将要操作的数据进行校验。

校验规则（AbstractServiceValidator）检查出不符合规则的数据后，通过通用的方法，通知操作剔除此数据。

### 案例 – 单据反审核时，生成凭证校验

**代码来源**

Kingdee.K3.SCM.App.ServicePlugIn.VoucherControl.UnAuditVoucherControl

Kingdee.K3.SCM.App.Validator.HasVoucherValidator

**需求背景**

业务单据已经生成凭证后，不允许反审核。

**实现方案**

建立一个通用的检查规则 – 单据是否生成凭证检查，在各个插件中引用此检查。

是否生成凭证检查规则，发现单据已经生成凭证之后，添加一个校验错误到操作错误列表，通知操作剔除此条数据不允许操作。

**关键字**

操作数据校验 校验规则 生成凭证检查 剔除校验不通过数据

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Validation;  using Kingdee.K3.Core.FIN;  using Kingdee.BOS.Core;  using Kingdee.K3.SCM.App.Validator;  using Kingdee.K3.BD.Contracts.FIN;  using Kingdee.BOS.Orm.DataEntity;  namespace Kingdee.K3.SCM.App.ServicePlugIn.VoucherControl  {  [Description("反审核插件，处理和凭证有关")]  public class UnAuditVoucherControl: AbstractOperationServicePlugIn  {  public override void OnAddValidators(AddValidatorsEventArgs e)  {  //添加凭证检查校验器：  // 如果已经生成凭证，且凭证已经更改，或审核  // 则单据不允许反审核  HasVoucherValidator voucherValidator = new HasVoucherValidator();  voucherValidator.EntityKey = "FBillHead";  voucherValidator.BillFormId = this.BusinessInfo.GetForm().Id;  e.Validators.Add(voucherValidator);  }  public override void BeginOperationTransaction(BeginOperationTransactionArgs e)  {  if (e.DataEntitys.Count() < 1)  {  return;  }  string formId = this.BusinessInfo.GetForm().Id;  List<long> lstBillID = new List<long>();  List<long> billIdList = new List<long>();  List<DynamicObject> dataEntitys = new List<DynamicObject>();  foreach (var item in e.DataEntitys)  {  lstBillID.Add(Convert.ToInt64(item["Id"]));  }  //以下是处理 如果生成的凭证未做任何处理则反审核时删除凭证。  var service = BD.Contracts.ServiceFactory.GetLocalService<IFINServiceForCommon>();  IEnumerable<BizBillVoucherInfor> lstBizBillInfor = service.GetBizBillVoucherInfor(this.Context, formId, lstBillID);  DynamicObject obj;  foreach (var bizBillInfor in lstBizBillInfor)  {  if (bizBillInfor.CanUnAudit)  {  long pkId = bizBillInfor.BillPKID;  billIdList.Add(pkId);  obj = (from p in e.DataEntitys  where Convert.ToInt64(p["Id"]) == pkId  select p).FirstOrDefault();  dataEntitys.Add(obj);  }  }  if (billIdList.Count > 0)  {  service.DeleteBizBillBizVoucher(this.Context, formId, billIdList);  }  e.DataEntitys = dataEntitys.ToArray();  }  }  } | |

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Enums;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.Core.FIN;  using Kingdee.K3.BD.Contracts.FIN;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  namespace Kingdee.K3.SCM.App.Validator  {  /// <summary>  /// 检查单据是否生成了凭证，如果生成了凭证则不允许反审核  /// </summary>  public class HasVoucherValidator : AbstractServiceValidator  {  /// <summary>  /// 当前单据的formid  /// </summary>  public string BillFormId { get; set; }  /// <summary>  /// 校验  /// </summary>  /// <param name="dataEntities"></param>  /// <param name="validateContext"></param>  /// <param name="ctx"></param>  public override void Validate(ExtendedDataEntity[] dataEntities,  ValidateContext validateContext, Context ctx)  {  if (dataEntities.IsNullOrEmpty() || dataEntities.Length == 0)  {  return;  }  base.Validate(dataEntities, validateContext, ctx);  List<long> lstBillID = new List<long>();  foreach (var item in dataEntities)  {  lstBillID.Add(Convert.ToInt64(item["Id"]));  }  var service = BD.Contracts.ServiceFactory.GetLocalService<IFINServiceForCommon>();  IEnumerable<BizBillVoucherInfor> lstBizBillInfor = service.GetBizBillVoucherInfor(  this.Context, this.BillFormId, lstBillID);  foreach (var bizBillInfor in lstBizBillInfor)  {  ////单据已经生成凭证，则凭证已经操作过则不可以反审核  if (!bizBillInfor.CanUnAudit)  {  long pkId = bizBillInfor.BillPKID;  ExtendedDataEntity item = GetDataEntity(dataEntities, pkId);  string msg = string.Format("单据“{0}”已生成凭证，不允许反审核！",  item.BillNo.ToString());  var errInfo = new ValidationErrorInfo(  item.BillNo,  item.DataEntity["Id"].ToString(),  item.DataEntityIndex,  item.RowIndex,  "SCMValid019",  msg,  " ",  BOS.Core.Validation.ErrorLevel.Error);  validateContext.AddError(item.DataEntity, errInfo);  continue;  }  }  }  private ExtendedDataEntity GetDataEntity(  ExtendedDataEntity[] dataEntities, long pkId)  {  foreach (ExtendedDataEntity item in dataEntities)  {  if (pkId == Convert.ToInt64(item["Id"]))  {  return item;  }  }  return new ExtendedDataEntity();  }  }  } | |

## BeforeExecuteOperationTransaction

操作执行前（事务外）事件，在数据检查完毕，正式开始执行操作前触发。

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void BeforeExecuteOperationTransaction(BeforeExecuteOperationTransaction e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| bool Cancel | 是否取消操作 |
| string CancelMessage | 操作取消原因 |
| DynamicObject[] DataEntitys | 待处理的单据数据包 |
| IEnumerable<Kingdee.BOS.Core.ExtendedDataEntity> SelectedRows | 通过操作校验的单据数据包 |

**备注**

此事件在事务外，不受事务保护。

通常将一些与数据更新无关的前期数据准备，放在本事件中进行，以减少事务占用时间。

### 案例 – 销售价目表保存前更新数量区间价格

**代码来源**

Kingdee.K3.SCM.App.Sal.ServicePlugIn.SalPrice.Save

**需求背景**

销售价目表，需要按照数量区间进行报价，用户录入的数量区间可能有重叠，需要在保存前进行修正。

**实现方案**

在保存操作插件中，捕获操作前事件，修正单据数据包中的数量区间价格。

修改后的单据数据包，在随后的保存操作中，统一提交到数据库。

**关键字**

保存前 操作前 事务外 修正数据

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Util;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS;  using Kingdee.BOS.Core.Metadata.ConvertElement;  namespace Kingdee.K3.SCM.App.Sal.ServicePlugIn.SalPrice  {  /// <summary>  /// 销售价目表服务端插件---保存校验  /// </summary>  public class Save : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  e.FieldKeys.Add("FMaterialId");//物料  e.FieldKeys.Add("FAuxPropId");//辅助属性  e.FieldKeys.Add("FCreateOrgId");//销售组织  e.FieldKeys.Add("FPriceObject");//价目对象  base.OnPreparePropertys(e);  }  public override void OnAddValidators(AddValidatorsEventArgs e)  {  SalPriceValidator valid = new SalPriceValidator();  valid.AlwaysValidate = true;  valid.EntityKey = "FBillHead";  e.Validators.Add(valid);  }  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  foreach (ExtendedDataEntity extended in e.SelectedRows)  {  DynamicObject dy = extended.DataEntity;  DynamicObjectCollection custs = dy["SAL\_APPLYCUSTOMER"] as DynamicObjectCollection;  DynamicObjectCollection salers = dy["SAL\_APPLYSALESMAN"] as DynamicObjectCollection;  if (custs == null || custs.Count() <= 0)  {  dy["LimitCustomer"] = "";  }  if (salers == null || salers.Count() <= 0)  {  dy["LimitSalesMan"] = "";  }  // 价格明细单据体  DynamicObjectCollection docPriceEntity = dy["SAL\_PRICELISTENTRY"] as DynamicObjectCollection;  DynamicObject doPriceEntityItem = null;  long lMaterialId = 0; // 物料Id  decimal deFromQty = 0M; // 数量从  decimal deToQty = 0M; // 数量至  int iAllCount = 0;  DynamicObject doPriceEntityItemCompare = null;  long lMaterialIdCompare = 0; // 物料Id（待比较）  decimal deFromQtyCompare = 0M; // 数量从（待比较）  decimal deToQtyCompare = 0M; // 数量至（待比较）  iAllCount = docPriceEntity.Count;  for (int i = 0; i < iAllCount; i++)  {  doPriceEntityItem = docPriceEntity[i];  lMaterialId = Convert.ToInt64(doPriceEntityItem["MaterialId\_Id"]); // 物料Id  deFromQty = Convert.ToDecimal(doPriceEntityItem["FromQty"]); // 数量从  deToQty = Convert.ToDecimal(doPriceEntityItem["ToQty"]); // 数量至  for (int j = i + 1; j < iAllCount; j++)  {  doPriceEntityItemCompare = docPriceEntity[j];  lMaterialIdCompare = Convert.ToInt64(doPriceEntityItemCompare["MaterialId\_Id"]); // 物料Id（待比较）  //优化，物料类别如果不相同，直接下一行，避免下面几个转换浪费性能  if (lMaterialId != lMaterialIdCompare)  continue;  deFromQtyCompare = Convert.ToDecimal(doPriceEntityItemCompare["FromQty"]); // 数量从（待比较）  deToQtyCompare = Convert.ToDecimal(doPriceEntityItemCompare["ToQty"]); // 数量至（待比较）  if (deToQtyCompare == 0)  {  deToQtyCompare = 999999999; // 如果至为0时，表示无穷大，这里默认设置为999999999  }  // 重新计算设置数量从的值  if (lMaterialId == lMaterialIdCompare && deToQty > deFromQtyCompare && deToQty < deToQtyCompare)  {  doPriceEntityItemCompare["FromQty"] = deToQty;  }  if (lMaterialId == lMaterialIdCompare && deToQtyCompare > deFromQty && deToQtyCompare < deToQty)  {  doPriceEntityItem["FromQty"] = deToQtyCompare;  }  }  }  }  }  }  } | |

## BeginOperationTransaction

操作执行前（事务内）事件，在数据检查完毕，启动事务之后触发。

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void BeginOperationTransaction(BeginOperationTransactionArgs e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| bool CancelFormService | 是否取消本操作关联的表单服务 |
| bool CancelOperation | 是否取消平台对本操作的内置处理 |
| DynamicObject[] DataEntitys | 本次处理的单据数据包 |

**备注**

此事件在事务内，在此事件对数据库进行更新，受事务保护。

此事件适用于需要在操作前，基于原始数据，对数据库进行更新，并与操作保持在一个事务中的需求。

例如需要在单据删除前，同步进行关联数据的删除。如果此功能，放在删除操作之后，则原始单据已经被删除了，就无法无法删除与原始单据关联的数据；

### 案例 – 反审核采购退货单，删除关联的内部交易单据

**代码来源**

Kingdee.K3.SCM.App.Sal.ServicePlugIn.SalPrice.Save

**需求背景**

销售价目表，需要按照数量区间进行报价，用户录入的数量区间可能有重叠，需要在保存前进行修正。

**实现方案**

在反审核操作插件中，反审核执行前，事务开启后，删除关联的内部交易单据，如果删除失败（数据库执行异常中断），则不会执行反审核操作。

如果删除成功，则执行反审核操作。

如果单据反审核操作失败，则回滚对内部交易单据的删除，恢复原始数据。

**关键字**

反审核前 操作前 事务内 同步删除数据 事务回滚

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Contracts;  using System.Data;  using Kingdee.BOS.App.Data;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.SCM.App.Service;  using Kingdee.K3.Core.MFG.PRD.BackFlush;  using Kingdee.K3.SCM.App.Validator;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Util;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS;  using Kingdee.K3.BD.App.Core.Validator.SCM;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.PurchaseReturn  {  public class UnAudit : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  base.OnPreparePropertys(e);  e.FieldKeys.Add("FBillTypeId");  }  /\*删除内部交易单据，配到反审核服务上\*/  public override void BeginOperationTransaction(BeginOperationTransactionArgs e)  {  DeleteBFReturnMtrlData(e.DataEntitys.ToList());  }  /// <summary>  /// 设置选项  /// </summary>  /// <param name="e"></param>  public override void OnPrepareOperationServiceOption(  OnPrepareOperationServiceEventArgs e)  {  base.OnPrepareOperationServiceOption(e);  //为了在BeginOperationTransaction里检查数据抛出异常时  //只回滚当前单据的事务，这里设置为不支持批量事务，这样BOS会  //循环为每一张单据创建事务调用操作  e.SupportTransaction = true;  e.SurportBatchTransaction = false;  }  /// <summary>  /// 删除委外倒冲退料单  /// </summary>  /// <param name="dataEntitys"></param>  private List<DynamicObject> DeleteBFReturnMtrlData(List<DynamicObject> dataEntitys)  {  List<DynamicObject> newEntitys = new List<DynamicObject>();  foreach (DynamicObject item in dataEntitys)  {  int index = dataEntitys.ToList().IndexOf(item);  string billTypeId = Convert.ToString(item["BillTypeID\_Id"]);  if (billTypeId.EqualsIgnoreCase(SCMBillTypeId.WWTLD))  {  List<object> pkIds = new List<object>();  object pkId = item["Id"];  pkIds.Add(pkId);  IOperationResult result = MFGService.DeleteBFReturnMtrlData(this.Context, pkIds, SCMFormIdConst.PUR\_MRB);  if (!result.IsSuccess)  {  string returnMtrlBillNo = result.CustomMessageFormId;  string billNo = item["BillNo"].ToString();  string strPkId = pkId.ToString();  string message = string.Format("对应的委外倒冲退料单{1}无法删除，退料单{0}不能反审核！", billNo, returnMtrlBillNo);  throw new KDBusinessException("", message); ;  }  }  newEntitys.Add(item);  }  return newEntitys;  }  public override void OnAddValidators(AddValidatorsEventArgs e)  {  //检查单据是否允许反审核，无关业务逻辑，略过  }  }  } | |

## EndOperationTransaction

操作执行后（事务内）事件，在操作处理完毕，未提交事务前触发。

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void EndOperationTransaction(EndOperationTransactionArgs e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| DynamicObject[] DataEntitys | 本次操作的单据数据包 |
|  |  |

**备注**

此事件在事务内，对数据库进行的更新，受事务保护。

此事件适合于在操作执行完毕，单据数据已经更改，需要基于更改后的数据，进行其他关联数据的处理。

例如需要在单据审核完毕，库存更新成功后，检查产生的负库存条目。

### 案例 – 采购折扣表审核时，同步更新供应商折扣

**代码来源**

Kingdee.K3.SCM.App.Pur.ServicePlugIn.DiscountList.Audit

**需求背景**

采购折扣表审核时，需要把采购折扣表上的供应商折扣，更新到供应商基础资料的属性上。

**实现方案**

在采购折扣表审核完成之后，事务未提交之前，生成更新供应商折扣的SQL语句，批量执行。

**关键字**

审核操作后 事务前 同步更新 事务回滚

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Orm.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Resource;  using Kingdee.BOS.Core;  using Kingdee.BOS.App.Data;  using Kingdee.BOS;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.DiscountList  {  /// <summary>  /// 采购折扣表服务端审核操作  /// </summary>  public class Audit : AbstractOperationServicePlugIn  {  public override void OnPreparePropertys(PreparePropertysEventArgs e)  {  e.FieldKeys.Add("FSupplierID");  e.FieldKeys.Add("FSupplierLocid");  e.FieldKeys.Add("FDefDiscountListId");//是否折扣表  e.FieldKeys.Add("FcreateorgId");  e.FieldKeys.Add("FuseorgId");  }  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  long supplierId = 0;  long discountListId = 0;  bool defDiscountListId = false;  List<string> sqlArray = new List<string>();  foreach (DynamicObject entity in e.DataEntitys)  {  if (entity != null)  {  supplierId = Convert.ToInt64(entity["supplierId\_Id"]);  discountListId = Convert.ToInt64(entity["Id"]);  defDiscountListId = Convert.ToBoolean(entity["DefDiscountListId"]);    if (discountListId > 0 && supplierId > 0 && defDiscountListId)  {  string sql = string.Format(@"UPDATE T\_BD\_SUPPLIERBUSINESS AS P  SET FDISCOUNTLISTID= {0}  WHERE P.FSupplierid={1} ", discountListId.ToString(), supplierId.ToString());  sqlArray.Add(sql);  }  }  }  if (sqlArray.Count > 0)  {  DBUtils.ExecuteBatch(this.Context, sqlArray, 100);  }  }  public override void AfterExecuteOperationTransaction(  AfterExecuteOperationTransaction e)  {  // 无关业务逻辑，代码略过  }  }  } | |

## AfterExecuteOperationTransaction

操作执行后（事务外）事件，在操作处理完毕，事务提交完毕后触发。

### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void AfterExecuteOperationTransaction(AfterExecuteOperationTransaction e); | |

**参数说明**

参数e的属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| DynamicObject[] DataEntitys | 本次操作，事务处理成功的单据数据包 |
| IEnumerable<Kingdee.BOS.Core.ExtendedDataEntity> SelectedRows | 通过操作校验的单据数据包 |
|  |  |

**备注**

通常用于在操作完毕后，进行后续的处理，但因为操作的事务已经提交，不能够撤销操作。

如操作完毕后，修饰操作提示信息，使之更加人性化。

### 案例 – 消耗汇总审核，提示同步生成的物权转移单编号

**代码来源**

Kingdee.K3.SCM.App.Pur.ServicePlugIn.VMIConsumeSum.Audit

**需求背景**

消耗汇总审核时，需同步生成物权单，并在提示信息中，包含生成的物权单编号，以便用户追查。

**实现方案**

在EndOperationTransaction事件（事务中）同步生成物权转移单，并记住生成的物权单编号。

在AfterExecuteOperationTransaction事件中，把物权单编号，拼接到提示信息中。

**关键字**

审核操作后 事务后 修饰提示信息 不适合做数据更新

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Util;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Core.DynamicForm;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.Core.SCM.Args;  using System.ComponentModel;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn.VMIConsumeSum  {  /// <summary>  /// 消耗汇总 审核操作服务端插件  /// </summary>  [Description("消耗汇总 审核服务插件")]  public class Audit : AbstractOperationServicePlugIn  {  //物权转移单编号  string tmpBillNo = string.Empty;  /// <summary>  /// 完成审核操作自动生成物权转移单  /// </summary>  /// <param name="e"></param>  public override void EndOperationTransaction(EndOperationTransactionArgs e)  {  if (e.DataEntitys != null && e.DataEntitys.Count<DynamicObject>() > 0)  {  ListSelectedRowCollection fxselectRows = new ListSelectedRowCollection();  foreach (DynamicObject item in e.DataEntitys)  {  string primaryKeyValue = item["Id"].ToString();  ListSelectedRow tempRow = new ListSelectedRow(primaryKeyValue,  string.Empty, 0, K3.Core.SCM.SCMFormIdConst.PUR\_VMIConsumeSum);  fxselectRows.Add(tempRow);  }  //审核时自动创建物权转移单  if (fxselectRows.Count > 0)  {  DynamicObject[] result = CreatePropertyConvertByVMIConsumeSum(  this.Context, fxselectRows, SCMBillTypeId.WQZYD);  if (this.OperationResult.IsSuccess && result.Length > 0)  {  tmpBillNo = result.FirstOrDefault()["BillNo"] as string;  }  }  }  }  public override void AfterExecuteOperationTransaction(  AfterExecuteOperationTransaction e)  {  base.AfterExecuteOperationTransaction(e);  OperateResultCollection oResult = this.OperationResult.OperateResult;  if (this.OperationResult.IsSuccess && oResult.Count > 0)  {  oResult[0].Message += string.Format("并已经自动生成物权转移单【{0}】。" tmpBillNo);  }  }  /// <summary>  /// 审核时自动创建 物权转移单  /// </summary>  /// <param name="ctx"></param>  /// <param name="selectRows"></param>  /// <returns></returns>  private DynamicObject[] CreatePropertyConvertByVMIConsumeSum(  Context ctx, ListSelectedRowCollection selectRows, string billTypeId)  {  IPurchaseService service = ServiceHelper.GetService<IPurchaseService>();  ConvertArgs args = new ConvertArgs();  args.SrcFormId = SCMFormIdConst.PUR\_VMIConsumeSum;  args.DestFormId = SCMFormIdConst.PUR\_PropertyConvert;  args.DestDocumentStatus = "P"; //保存不成功就暂存  args.DestBillTypeId = billTypeId;  return service.AutoCreateBillByConvertRule(ctx, args, selectRows);  }  }  } | |