单据转换插件示例代码

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

[单据转换插件示例代码 1](#_Toc416794981)

[单据转换插件概要说明 4](#_Toc416794982)

[如何创建一个单据转换插件 4](#_Toc416794983)

[创建单据转换插件类库 4](#_Toc416794984)

[引用必要组件 5](#_Toc416794985)

[创建单据转换插件类 6](#_Toc416794986)

[如何启用单据转换插件 20](#_Toc416794987)

[公共属性 21](#_Toc416794988)

[Context 21](#_Toc416794989)

[语法 21](#_Toc416794990)

[OperationNumber 21](#_Toc416794991)

[语法 21](#_Toc416794992)

[OperationResult 21](#_Toc416794993)

[语法 21](#_Toc416794994)

[Option 22](#_Toc416794995)

[语法 22](#_Toc416794996)

[单据转换事件（插件基类的虚方法） 22](#_Toc416794997)

[下推事件 22](#_Toc416794998)

[OnInitVariable 23](#_Toc416794999)

[OnQueryBuilderParemeter 25](#_Toc416795000)

[OnInSelectedRow 27](#_Toc416795001)

[OnParseFilter 30](#_Toc416795002)

[OnBeforeGetSourceData 33](#_Toc416795003)

[OnGetSourceData 34](#_Toc416795004)

[OnBeforeGroupBy 38](#_Toc416795005)

[OnCreateTarget 41](#_Toc416795006)

[OnBeforeFieldMapping 42](#_Toc416795007)

[OnFieldMapping 43](#_Toc416795008)

[OnAfterFieldMapping 46](#_Toc416795009)

[OnCreateLink 48](#_Toc416795010)

[OnAfterCreateLink 49](#_Toc416795011)

[OnGetConvertBusinessService 50](#_Toc416795012)

[AfterConvert 50](#_Toc416795013)

[选单前事件 54](#_Toc416795014)

[OnInitVariable 54](#_Toc416795015)

[OnParseFilterOptions 54](#_Toc416795016)

[OnParseFilter 56](#_Toc416795017)

[选单后事件 56](#_Toc416795018)

[OnInitVariable 57](#_Toc416795019)

[OnQueryBuilderParemeter 57](#_Toc416795020)

[OnInSelectedRow 57](#_Toc416795021)

[OnBeforeGetSourceData 57](#_Toc416795022)

[OnGetDrawSourceData 58](#_Toc416795023)

[OnBeforeGroupBy 61](#_Toc416795024)

[OnCreateDrawTarget 61](#_Toc416795025)

[OnBeforeFieldMapping 62](#_Toc416795026)

[OnFieldMapping 62](#_Toc416795027)

[OnAfterFieldMapping 62](#_Toc416795028)

[OnCreateLink 63](#_Toc416795029)

[OnAfterCreateLink 63](#_Toc416795030)

[OnGetConvertBusinessService 63](#_Toc416795031)

[AfterConvert 63](#_Toc416795032)

[附录：列表插件提供的单据转换事件 63](#_Toc416795033)

[OnShowConvertOpForm 63](#_Toc416795034)

[OnGetConvertRule 68](#_Toc416795035)

# 单据转换插件概要说明

单据转换，是指把上游所选单据，按照转换规则，自动生成下游单据数据包的过程；不对下游单据进行保存、提交、审核等处理；

单据转换插件，能够介入到单据转换的各个关键时刻，对转换行为进行控制，从而调整所生成的下游数据包；

单据转换按照发起方不同，可分为下推、选单；

下推是指在上游单据列表，把所选单据，生成下游单据数据包，并展示出来；

选单是指在下游单据新增界面，弹出上游单据列表，选择源单返回，然后根据转换规则把源单数据填写到下游单据新增界面上；

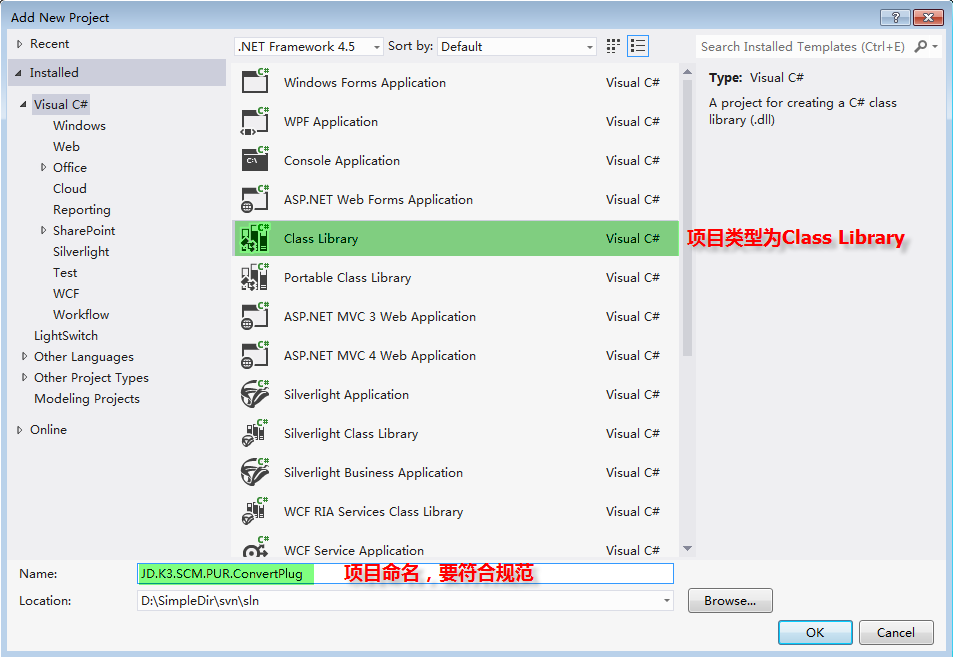
选单实际上分为两个独立的过程，一个是选单前过程，根据转换规则，生成源单数据筛选条件，传给源单列表。因此源单列表上显示的，都是允许下推的数据；另外一个就是选单过程，把用户选择返回的源单数据，迁移到目标单据上；

下推与选单，采用相同单据转换规则，进行数据迁移；也采用相同的单据转换插件，但触发的事件略有差异，编写单据转换插件时，需要兼顾这些差异；

## 如何创建一个单据转换插件

### 创建单据转换插件类库

使用VS 2010及以上版本的C#语言，创建一个Class Library项目：



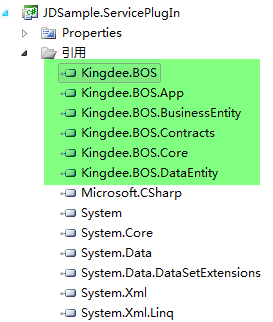
创建好单据转换插件类库工程后，可以把同一个子系统的所有单据转换插件类都放在这个类库中；

项目命名规范：[开发商].K3[.业务领域][.子系统].ConvertPlug；

### 引用必要组件

到K/3 Cloud安装目录的WebSite\Bin子目录，引用如下组件：

* Kingdee.BOS.dll
* Kingdee.BOS.App.dll
* Kingdee.BOS.BusinessEntity.dll
* Kingdee.BOS.Contracts.dll
* Kingdee.BOS.Core.dll
* Kingdee.BOS.DataEntity.dll



### 创建单据转换插件类

在前文所建的单据转换插件工程中，添加新类,派生自单据转换插件基类：Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.AbstractConvertPlugIn

下文列出了三个空的单据转换插件代码，分别演示下推、选单前、选单过程所触发的事件：

#### 示例 - 演示下推触发的事件

类PushEventSeq，仅重载了下推会执行的事件，并且按照各个事件的执行顺序进行重载：

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  namespace SamplePlugIn.ConvertPlugIn  {  /// <summary>  /// 按顺序输出单据转换-下推过程的插件事件  /// </summary>  [System.ComponentModel.Description("按顺序输出单据转换-下推过程的插件事件")]  public class PushEventSeq : AbstractConvertPlugIn  {  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 初始化变量之后触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 此方法执行时，源单、目标单元数据，转换规则均已加载；  /// 可以在此事件初始化自己的变量，或者调整转换规则  /// </remarks>  public override void OnInitVariable(InitVariableEventArgs e)  {  base.OnInitVariable(e);  this.PrintEventName("OnInitVariable");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 取源单数据参数(QueryBuilderParemeter)初始化完毕后触发，  /// 此时，已经扫描了转换规则上的映射关系、计算公式等，确定了需要加载的源单字段；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 默认情况下，仅会加载配置了映射关系、计算公式的源单字段；  /// 可以在此事件中，追加需要加载的源单字段；  /// </remarks>  public override void OnQueryBuilderParemeter(QueryBuilderParemeterEventArgs e)  {  base.OnQueryBuilderParemeter(e);  this.PrintEventName("OnQueryBuilderParemeter");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 把所选的源单内码，转换为取数条件后触发，  /// 此时，源单内码，被拼接成 FID IN (100000,100001) 条件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件，修改取数条件，完善内码筛选条件  /// </remarks>  public override void OnInSelectedRow(InSelectedRowEventArgs e)  {  base.OnInSelectedRow(e);  this.PrintEventName("OnInSelectedRow");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 解析完转换规则的选单条件策略之后触发，  /// 此时，已经把选单条件策略中设置的过滤条件，拼接成了一个SQL条件子句  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，修改取数条件，完善选单条件策略中设置的条件  /// </remarks>  public override void OnParseFilter(ParseFilterEventArgs e)  {  base.OnParseFilter(e);  this.PrintEventName("OnParseFilter");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 调用IQueryService.GetDynamicObjectCollection()函数之前触发，  /// 此时取源单参数已经解析完毕，构建出了取数SQL的各种子句，如Select、From、Where子句  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对取源单数据的SQL子句进行完善  /// </remarks>  public override void OnBeforeGetSourceData(BeforeGetSourceDataEventArgs e)  {  base.OnBeforeGetSourceData(e);  this.PrintEventName("OnBeforeGetSourceData");  }  // \*\*\*\*\*\*本事件仅适用于下推\*\*\*\*\*\*  /// <summary>  /// 读取了符合条件下推的源单数据之后触发，  /// 此时，已经拿到了源单完整数据，之前仅仅掌握源单内码；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件，修订源单数据  /// </remarks>  public override void OnGetSourceData(GetSourceDataEventArgs e)  {  base.OnGetSourceData(e);  this.PrintEventName("OnGetSourceData");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 读取源单数据之后，按照转换规则 - 分组策略，对源单数据进行分组之前触发，  /// 此时，已经解析完分组策略，后续将按照此进行单据分组、分录合并  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，修改单据分组、分录合并规则  /// </remarks>  public override void OnBeforeGroupBy(BeforeGroupByEventArgs e)  {  base.OnBeforeGroupBy(e);  this.PrintEventName("OnBeforeGroupBy");  }  // \*\*\*\*\*\*本事件仅适用于下推\*\*\*\*\*\*  /// <summary>  /// 根据单据分组规则，对源单分好组之后，根据分组数量，创建目标单据之后触发,  /// 此时，已经创建好目标单据数据包，并填写了各字段默认值  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，完善各个字段的默认值  /// </remarks>  public override void OnCreateTarget(CreateTargetEventArgs e)  {  base.OnCreateTarget(e);  this.PrintEventName("OnCreateTarget");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在开始按照转换规则 - 字段映射策略，进行数据迁移之前触发，  /// 此时，源单数据包、目标单数据包均已对应好，开始进行字段数据迁移  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 因为有很多行、很多字段需要逐个迁移，总的迁移次数会很多，  /// 默认情况下，不会触发每个字段的迁移事件。  /// 如果需要监控单个字段的迁移过程，可以在此事件中，开启字段的迁移事件；  /// </remarks>  public override void OnBeforeFieldMapping(BeforeFieldMappingEventArgs e)  {  base.OnBeforeFieldMapping(e);  this.PrintEventName("OnBeforeFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在计算完目标单据每行、每个字段值，准备填写到数据包之前，均会触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 此事件默认不会被触发  /// 插件在OnBeforeFieldMapping事件中设置e.FireFieldMappingEvent = true，可开启本事件  /// 可以在此事件中，监控、修正需要写到目标单据上的字段值；  /// </remarks>  public override void OnFieldMapping(FieldMappingEventArgs e)  {  base.OnFieldMapping(e);  this.PrintEventName("OnFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在根据转换规则 - 字段映射策略，迁移完字段值之后触发，  /// 此时，已经把源单数据包，都迁移到了目标单据数据包  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对目标单据数据包进行修订，或者补充其他单据体数据  /// </remarks>  public override void OnAfterFieldMapping(AfterFieldMappingEventArgs e)  {  base.OnAfterFieldMapping(e);  this.PrintEventName("OnAfterFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 字段迁移完毕，创建关联关系子表，在单据上记录关联关系前触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，设置e.Cancel = true，略过关联关系的创建；  /// 关联关系主要用于反写、联查，如无必要，请勿取消  /// </remarks>  public override void OnCreateLink(CreateLinkEventArgs e)  {  base.OnCreateLink(e);  this.PrintEventName("OnCreateLink");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 关联关系子表已经创建并填写完毕后触发，  /// 此时，已经在关联子表中，记录了源单与目标单之间的关系  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对关联关系子表内容进行调整，  /// </remarks>  public override void OnAfterCreateLink(CreateLinkEventArgs e)  {  base.OnAfterCreateLink(e);  this.PrintEventName("OnAfterCreateLink");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 数据从源单迁移完毕，开始执行转换规则 - 表单服务策略上配置的服务前触发，  /// 此时，已经创建好了服务执行列表，但并未实际执行服务  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，增删改服务列表  /// </remarks>  public override void OnGetConvertBusinessService(ConvertBusinessServiceEventArgs e)  {  base.OnGetConvertBusinessService(e);  this.PrintEventName("OnGetConvertBusinessService");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 单据从源单到目标单已经转换完毕，输出最后的目标单数据包之前触发,  /// 此时，目标单数据包已经构建完毕，不会再有变动了  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对目标单数据包进行最后的修订、补充  /// </remarks>  public override void AfterConvert(AfterConvertEventArgs e)  {  base.AfterConvert(e);  this.PrintEventName("AfterConvert");  }  /// <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);  }  }  } | |

#### 示例 - 演示选单前触发的事件

类BeforeDrawEventSeq，仅重载了选单前过程会执行的事件，并且按照各个事件的执行顺序进行重载：

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  namespace SamplePlugIn.ConvertPlugIn  {  /// <summary>  /// 输出单据转换-选单前过程的插件事件执行顺序  /// </summary>  [System.ComponentModel.Description("输出单据转换-选单前过程的插件事件执行顺序")]  public class BeforeDrawEventSeq : AbstractConvertPlugIn  {  // \*\*\*\*\*\*本事件同时适用于下推、选单前、选单\*\*\*\*\*\*  /// <summary>  /// 初始化变量之后触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 此方法执行时，源单、目标单元数据，转换规则均已加载；  /// 可以在此事件初始化自己的变量，或者调整转换规则  /// </remarks>  public override void OnInitVariable(InitVariableEventArgs e)  {  base.OnInitVariable(e);  this.PrintEventName("OnInitVariable");  }  // \*\*\*\*\*\*本事件仅适用于选单前\*\*\*\*\*\*  /// <summary>  /// 根据转换规则的配置，把目标单关键字段值，拼进过滤条件之后触发，  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 转换规则 - 字段映射策略中，可以勾选目标字段的过滤、仅追加选项，  /// 目标字段勾上此过滤选项后，选单时，会该字段值作为过滤条件，传递给源单列表，  /// 可以在此事件中，调整根据目标字段值生成的过滤条件  /// </remarks>  public override void OnParseFilterOptions(ParseFilterOptionsEventArgs e)  {  base.OnParseFilterOptions(e);  this.PrintEventName("OnParseFilterOptions");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单前\*\*\*\*\*\*  /// <summary>  /// 解析完转换规则的选单条件策略之后触发，  /// 此时，已经把选单条件策略中设置的过滤条件，拼接成了一个SQL条件子句  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，修改取数条件，完善选单条件策略中设置的条件  /// </remarks>  public override void OnParseFilter(ParseFilterEventArgs e)  {  base.OnParseFilter(e);  this.PrintEventName("OnParseFilter");  }  /// <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);  }  }  } | |

#### 示例 - 演示选单触发的事件

类DrawEventSeq，仅重载了选单过程会执行的事件，并且按照各个事件的执行顺序进行重载：

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  namespace SamplePlugIn.ConvertPlugIn  {  /// <summary>  /// 输出单据转换-选单前过程的插件事件执行顺序  /// </summary>  [System.ComponentModel.Description("输出单据转换-选单前过程的插件事件执行顺序")]  public class DrawEventSeq : AbstractConvertPlugIn  {  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 初始化变量之后触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 此方法执行时，源单、目标单元数据，转换规则均已加载；  /// 可以在此事件初始化自己的变量，或者调整转换规则  /// </remarks>  public override void OnInitVariable(InitVariableEventArgs e)  {  base.OnInitVariable(e);  this.PrintEventName("OnInitVariable");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 取源单数据参数(QueryBuilderParemeter)初始化完毕后触发，  /// 此时，已经扫描了转换规则上的映射关系、计算公式等，确定了需要加载的源单字段；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 默认情况下，仅会加载配置了映射关系、计算公式的源单字段；  /// 可以在此事件中，追加需要加载的源单字段；  /// </remarks>  public override void OnQueryBuilderParemeter(QueryBuilderParemeterEventArgs e)  {  base.OnQueryBuilderParemeter(e);  this.PrintEventName("OnQueryBuilderParemeter");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 把所选的源单内码，转换为取数条件后触发，  /// 此时，源单内码，被拼接成 FID IN (100000,100001) 条件  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件，修改取数条件，完善内码筛选条件  /// </remarks>  public override void OnInSelectedRow(InSelectedRowEventArgs e)  {  base.OnInSelectedRow(e);  this.PrintEventName("OnInSelectedRow");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 调用IQueryService.GetDynamicObjectCollection()函数之前触发，  /// 此时取源单参数已经解析完毕，构建出了取数SQL的各种子句，如Select、From、Where子句  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对取源单数据的SQL子句进行完善  /// </remarks>  public override void OnBeforeGetSourceData(BeforeGetSourceDataEventArgs e)  {  base.OnBeforeGetSourceData(e);  this.PrintEventName("OnBeforeGetSourceData");  }    // \*\*\*\*\*\*本事件仅适用于下推\*\*\*\*\*\*  /// <summary>  /// 读取了所选的源单数据之后触发，  /// 此时，已经拿到了源单完整数据，之前仅仅掌握源单内码；  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件，修订源单数据  /// </remarks>  public override void OnGetDrawSourceData(GetDrawSourceDataEventArgs e)  {  base.OnGetDrawSourceData(e);  this.PrintEventName("OnGetDrawSourceData");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 读取源单数据之后，按照转换规则 - 分组策略，对源单数据进行分组之前触发，  /// 此时，已经解析完分组策略，后续将按照此进行单据分组、分录合并  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，修改单据分组、分录合并规则  /// </remarks>  public override void OnBeforeGroupBy(BeforeGroupByEventArgs e)  {  base.OnBeforeGroupBy(e);  this.PrintEventName("OnBeforeGroupBy");  }  // \*\*\*\*\*\*本事件仅适用于下推\*\*\*\*\*\*  /// <summary>  /// 读取源单数据之后，对源单行进行合并，但还没有构建目标单据行之前触发,  /// 此时，源单行已经分好组，目标单据还是选单之前的数据包  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，调整源单分组结果  /// </remarks>  public override void OnCreateDrawTarget(CreateDrawTargetEventArgs e)  {  base.OnCreateDrawTarget(e);  this.PrintEventName("OnCreateDrawTarget");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在开始按照转换规则 - 字段映射策略，进行数据迁移之前触发，  /// 此时，源单数据包、目标单数据包均已对应好，开始进行字段数据迁移  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 因为有很多行、很多字段需要逐个迁移，总的迁移次数会很多，  /// 默认情况下，不会触发每个字段的迁移事件。  /// 如果需要监控单个字段的迁移过程，可以在此事件中，开启字段的迁移事件；  /// </remarks>  public override void OnBeforeFieldMapping(BeforeFieldMappingEventArgs e)  {  base.OnBeforeFieldMapping(e);  this.PrintEventName("OnBeforeFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在计算完目标单据每行、每个字段值，准备填写到数据包之前，均会触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 此事件默认不会被触发  /// 插件在OnBeforeFieldMapping事件中设置e.FireFieldMappingEvent = true，可开启本事件  /// 可以在此事件中，监控、修正需要写到目标单据上的字段值；  /// </remarks>  public override void OnFieldMapping(FieldMappingEventArgs e)  {  base.OnFieldMapping(e);  this.PrintEventName("OnFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 在根据转换规则 - 字段映射策略，迁移完字段值之后触发，  /// 此时，已经把源单数据包，都迁移到了目标单据数据包  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对目标单据数据包进行修订，或者补充其他单据体数据  /// </remarks>  public override void OnAfterFieldMapping(AfterFieldMappingEventArgs e)  {  base.OnAfterFieldMapping(e);  this.PrintEventName("OnAfterFieldMapping");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 字段迁移完毕，创建关联关系子表，在单据上记录关联关系前触发  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，设置e.Cancel = true，略过关联关系的创建；  /// 关联关系主要用于反写、联查，如无必要，请勿取消  /// </remarks>  public override void OnCreateLink(CreateLinkEventArgs e)  {  base.OnCreateLink(e);  this.PrintEventName("OnCreateLink");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 关联关系子表已经创建并填写完毕后触发，  /// 此时，已经在关联子表中，记录了源单与目标单之间的关系  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对关联关系子表内容进行调整，  /// </remarks>  public override void OnAfterCreateLink(CreateLinkEventArgs e)  {  base.OnAfterCreateLink(e);  this.PrintEventName("OnAfterCreateLink");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 数据从源单迁移完毕，开始执行转换规则 - 表单服务策略上配置的服务前触发，  /// 此时，已经创建好了服务执行列表，但并未实际执行服务  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，增删改服务列表  /// </remarks>  public override void OnGetConvertBusinessService(ConvertBusinessServiceEventArgs e)  {  base.OnGetConvertBusinessService(e);  this.PrintEventName("OnGetConvertBusinessService");  }  // \*\*\*\*\*\*本事件同时适用于下推、选单\*\*\*\*\*\*  /// <summary>  /// 单据从源单到目标单已经转换完毕，输出最后的目标单数据包之前触发,  /// 此时，目标单数据包已经构建完毕，不会再有变动了  /// </summary>  /// <param name="e"></param>  /// <remarks>  /// 可以在此事件中，对目标单数据包进行最后的修订、补充  /// </remarks>  public override void AfterConvert(AfterConvertEventArgs e)  {  base.AfterConvert(e);  this.PrintEventName("AfterConvert");  }  /// <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);  }  }  } | |

**特别说明：**

下推、选单前、选单过程，采用的插件是同一个。

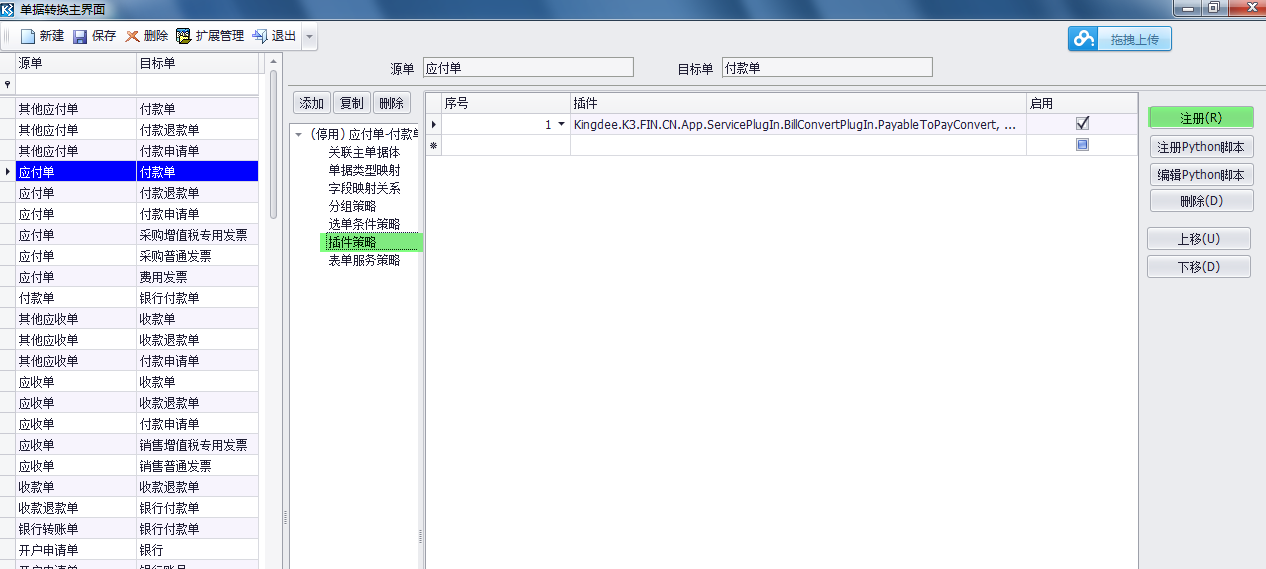
上文只是为了突出演示效果，为下推、选单、选单过程，分别写插件演示其所触发的事件；

从事件的覆盖度来看，下推过程触发的事件是最全面的，针对下推过程编写的插件，选单过程被自动覆盖；

## 如何启用单据转换插件

插件代码编写完毕后，编译组件，把生成的类库文件(\*.dll)复制到K/3 Cloud安装目录的WebSite\Bin子目录；

然后进入BOS设计器，打开单据转换主界面，找到转换规则，去掉启用选项，修改插件策略，注册新插件：



利用BOS设计器制作部署包时，需要在部署向导中，加入生成的转换插件类库文件(\*.dll)，以及自定义(或扩展)的单据转换规则。

# 公共属性

## Context

用户登录上下文，记录当前登录用户、数据连接等信息。

### 语法

**属性定义**

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

**备注**

调用各种服务函数时，需传入本对象作为参数。

## OperationNumber

操作编码标识：如选单操作、下推操作。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public string OperationNumber { get; } | |

**备注**

本属性只有两种可选值：Push、Draw；

## OperationResult

单据转换结果信息。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| public ConvertOperationResult OperationResult { get; } | |

**备注**

管理单据转换结果，包括提示信息、成功生成的目标单据数据包。

## Option

额外业务控制选项：整体业务流程的每个时机点都可拿到这些控制信息

### 语法

**属性定义**

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

**备注**

调用下推引擎时，额外传递的一些参数字典。如是否整单下推、是否需要校验下游单据新增权限等；

单据转换插件可以到此参数字典中，获取自定义参数。

# 单据转换事件（插件基类的虚方法）

## 下推事件

下推，是把所选源单，转换为目标单据数据包的过程，大致经过以下几个阶段：

1. 初始化变量；
2. 构建读取源单数据参数；
3. 根据单据转换规则上，字段映射关系、计算公式等，设置需要读取的源单字段；
4. 根据单据转换规则 – 选单条件策略，设置源单过滤条件；
5. 把所选源单内码，拼进源单过滤条件；
6. 构建一条取数SQL，读取源单数据； （把单据头、单据体、子单据体的字段合并在一起）
7. 按照单据转换规则 – 分组策略，对源单数据进行分组；
8. 根据分组，构建目标单空数据包；
9. 把源单数据，一一迁移到目标单数据包上；
10. 在目标单数据包中，记录与源单的关联关系，并自动进行单位换算；
11. 执行单据转换规则 – 表单服务策略，对目标单据进行后期数据修正、补充；
12. 返回已经生成的目标单据数据包；

整个下推的过程，单据转换插件基类上可重载的虚方法，按如下顺序被调用：

1. OnInitVariable
2. OnQueryBuilderParemeter
3. OnInSelectedRow
4. OnParseFilter
5. OnBeforeGetSourceData
6. OnGetSourceData
7. OnBeforeGroupBy
8. OnCreateTarget
9. OnBeforeFieldMapping
10. OnAfterFieldMapping
11. OnCreateLink
12. OnAfterCreateLink
13. OnGetConvertBusinessService
14. AfterConvert

插件重载这些方法，调整方法的参数，以实现对下推的过程、生成的目标单据数据包进行干预、修订；

下文将按照顺序介绍下推过程所触发的全部事件：

### OnInitVariable

初始化变量之后触发本事件；

此方法执行时，源单、目标单元数据，转换规则均已加载；可以在此事件初始化自己的变量，或者调整转换规则。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnInitVariable(InitVariableEventArgs e); | |

**备注：**

变量初始化完成，准备单据转换所需的变量，这个事件最早触发，插件可以在这个事件中准备所需的变量。

**触发时机：**

单据转换服务准备完源单和目标单的元数据后，还未进入单据转换的数据处理过程

**应用场景：**

为插件提供时机

1. 根据接口参数，初始化插件自身的信息

**插件接口介绍：**

|  |  |  |
| --- | --- | --- |
| **InitVariableEventArgs** | | |
| **属性** | **类型** | **描述** |
| Context | Context | 当前系统上下文全局信息 |
| Rule | ConvertRuleElement | 转换规则元数据信息 |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |
| TargetBusinessInfo | BusinessInfo | 目标单据元数据信息 |

#### 案例 – 结算清单下推应付单插件，初始化本地变量

**需求背景：**

正式进入单据转换过程前，初始化一些插件全局属性

**实现方案：**

单据转换插件中，在OnInitVariable事件里做插件全部变量初始化的工作

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core;  using Kingdee.K3.FIN.Core;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Util;  namespace Kingdee.K3.FIN.AP.App.ServicePlugIn.BillConvert  {  [Description("结算清单到应付单")]  public class OrgSettleToPayableConvert : StockToPayableConvert  {  /// 目标单FormID  private string targetFormID;  public override void OnInitVariable(InitVariableEventArgs e)  {  this.targetFormID = e.TargetBusinessInfo.GetForm().Id;  }  }  } | |

### OnQueryBuilderParemeter

取源单数据的参数对象(QueryBuilderParemeter)初始化完毕之后，触发本事件。

此时，已经扫描了转换规则上的映射关系、计算公式等，确定了需要加载的源单字段；

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnQueryBuilderParemeter(QueryBuilderParemeterEventArgs e); | |

**备注：**

默认情况下，仅会加载配置了映射关系、计算公式的源单字段；可以在此事件中，修改QueryBuilderParemeter，追加需要加载的源单字段；

**触发时机：**

获取单据转换字段映射中源单使用的字段和其对应的表格信息

获取源单数据之前，准备查询字段信息

**应用场景：**

为插件提供时机

1、解析源单取数的字段列

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **QueryBuilderParemeterEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| DicFieldAlias | Dictionary<string, string> | 基础资料属性字段key与字段别名关系字典，默认别名可能会超出自定义长度 |  |
| SelectItems | List<SelectorItemInfo> | 摘选的字段片段信息 | Y：源单字段列表 |

#### 案例 – 收货单下推退货单，需强制加载收货单的库存状态字段

**需求背景：**

获取源单数据表格信息中，添加库存状态等字段

**实现方案：**

OnQueryBuilderParemeter事件中，修改参数的SelectItems属性

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core;  using Kingdee.K3.Core.BD.ServiceArgs;  using Kingdee.K3.BD.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.BOS;  using Kingdee.K3.SCM.App.Utils;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  /// <summary>  /// 收货单-->退货单关联路线插件  /// </summary>  public class PurReceiveToMRBConvertService : AbstractConvertPlugIn  {  private const long stockStatusId = 10001;///库存状态 :待检  /// <summary>  /// 需要来料检验重新分配基本数量的分录ID集  /// </summary>  Dictionary<long, decimal> dicQty = new Dictionary<long, decimal>();  public override void OnQueryBuilderParemeter(QueryBuilderParemeterEventArgs e)  {  //添加库存状态字段  e.SelectItems.Add(new SelectorItemInfo("FStockStatusId"));  //添加紧急放行字段  e.SelectItems.Add(new SelectorItemInfo("FEmergencyRelease"));  //添加来料检验字段  e.SelectItems.Add(new SelectorItemInfo("FCheckInComing"));  //添加来判退数量字段  e.SelectItems.Add(new SelectorItemInfo("FRefuseBaseQty"));  ///退料调拨关联数量（基本单位 ）  e.SelectItems.Add(new SelectorItemInfo("FReturnJoinBaseQty"));  }  }  } | |

### OnInSelectedRow

把所选的源单内码，转换为取数条件之后，触发本事件。

此时，源单内码，已经被拼接成类似于”FID IN (100000,100001)” 的SQL条件子句。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnInSelectedRow(InSelectedRowEventArgs e); | |

**备注：**

通过干预源单取数的Where子句，达到过滤源单业务数据的作用

**触发时机：**

构建源单选择行的Id的IN语句

**应用场景：**

为插件提供时机

1、干预源单选中行Id的语句构造

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **InSelectedRowEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| SelectedRows | IEnumerable<ListSelectedRow> | 用户在列表上的选择对象 |  |
| PkKey | string | In子句的主键字段key |  |
| InSelectedRowsSQL | string | 根据选择的行产生的条件子句 | Y：拼接到后续取数Sql指令的Where条件中 |
| SqlParamPKValues | SqlParam | 内码集合对应的Sql变量，例如：e.SqlParamPKValues = new SqlParam("@PKValue", KDDbType.udt\_inttable, pkValueArray.Distinct().ToArray()); | Y：表变量参数 |
| JoinTable | ExtJoinTableDescription | 内码临时表描述对象，例如：e.JoinTable = new ExtJoinTableDescription()  {  TableName = "table(fn\_StrSplit(@PKValue,',',1))",  TableNameAs = "plugTmp",  FieldName = "FId",  ScourceKey = pkKey  }; | Y：表变量 |

#### 案例 – 简单生产领料下推退料单，追加条件：可退数量大于0

**需求背景：**

简单生产领料单下推简单退料单，其源单过滤条件增加实发数量减退料选单数量必须大于0

**实现方案：**

转换插件中，在OnInSelectedRow中，修改参数的InSelectedRowsSQL，携带其过滤条件，即(FActualQty - FSelPrcdReturnQty) > 0

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.QueryElement;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Util;  using Kingdee.K3.Core.MFG.EntityHelper;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.Core.SCM.STK.SP.Const;  using Kingdee.K3.SCM.App.ConvertPlugIn;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.SCM.Contracts.SP;  namespace Kingdee.K3.SCM.App.Stock.ServicePlugIn  {  /// <summary>  /// 物料清单或者简单领料单到简单退料单的单据转换插件  /// </summary>  public class BOMOrSpPick2SpReturnItemConvert : BaseBillConvertServicePlugIn  {  public override void OnInSelectedRow(InSelectedRowEventArgs e)  {  if (this.IsPush)  {  List<ListSelectedRow> SelectedRows = e.SelectedRows.ToList();  if (SelectedRows != null && SelectedRows.Count() > 0)  {  string strFilterSql = string.Empty;  //简单生产领料单  if (SelectedRows[0].FormID.Equals("SP\_PickMtrl"))  {  strFilterSql = " (FActualQty - FSelPrcdReturnQty) > 0 ";  e.InSelectedRowsSQL = e.InSelectedRowsSQL + (string.IsNullOrWhiteSpace(e.InSelectedRowsSQL) ? "" : " AND ") + strFilterSql;  }  }  }  }  }  } | |

### OnParseFilter

解析完转换规则的选单条件策略之后，触发本事件。

此时，已经把选单条件策略中设置的过滤条件，拼接成了一个SQL条件子句。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnParseFilter(ParseFilterEventArgs e); | |

**备注：**

通过干预单据转换中配置的过滤策略中的过滤条件，干预源单取数的Where子句，达到过滤业务数据的作用

**触发时机：**

解析过滤策略中配置的过滤条件时

**应用场景：**

为插件提供时机

1、干预单据转换的过滤策略中解析出的过滤条件

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **ParseFilterEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| FilterPolicySQL | string | 根据过滤策略解析的条件 | Y：拼接到后续源单取数Sql指令的Where条件中 |
| PlugFilterDesc | string | 插件添加的过滤条件说明 | Y：如果后续在取源单数据时，发现没取到符合条件的源单数据，通过此字符串，抛出取数失败的原因 |

#### 案例 – 采购订单入库，追加条件：收料组织不能为空

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Resource;  using Kingdee.K3.BD.Contracts;  using Kingdee.BOS.Contracts;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.Core.BD;  using Kingdee.K3.Core.BD.ServiceArgs;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS;  using Kingdee.K3.SCM.App.ConvertPlugIn;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.K3.SCM.App.Utils;  using Kingdee.BOS.App.Data;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  public class PurchaseOrderToInStockService : AbstractConvertPlugIn  {  string m\_BillTypeId = "";  bool m\_DeliveryByPlan = false;//参考交货计划执行  long stockOrgId = 0;  public override void OnParseFilter(ParseFilterEventArgs e)  {  string strFilter = string.Empty;  string strDesc = string.Empty;  switch (this.OperationNumber.ToUpperInvariant())  {  case "DRAW":  strFilter = string.Format(" (ISNULL(FRECEIVEORGID,0) = {0} Or ISNULL(FRECEIVEORGID,0) = 0)", stockOrgId);  break;  case "PUSH":  strFilter = " ISNULL(FRECEIVEORGID,0) >　0 ";  strDesc = "收料组织不能为空";  break;  default:  break;  }  if (string.IsNullOrWhiteSpace(e.FilterPolicySQL))  {  e.FilterPolicySQL = strFilter;  }  else  {  e.FilterPolicySQL += " AND " + strFilter;  }  if (string.IsNullOrWhiteSpace(e.PlugFilterDesc))  {  e.PlugFilterDesc = strDesc;  }  else  {  e.PlugFilterDesc += strDesc;  }  }  }  } | |

### OnBeforeGetSourceData

调用取数服务，读取源单数据之前，触发本事件。

此时取源单参数已经解析完毕，据此构建出了取数SQL的各种子句，如Select、From、Where子句。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnBeforeGetSourceData(BeforeGetSourceDataEventArgs e); | |

**备注：**

查找符合前述条件的源单数据之前，进入到数据库查询前，最后一次干预查询对象（综合前述查询条件的查询对象）

**触发时机：**

进入数据库查询满足条件的源单数据前

**应用场景：**

为插件提供时机

1、干预查询源单数据的过滤对象等过滤信息

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeGetSourceDataEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| BusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| QueryObject | QueryObject | 查询对象 | Y：查询对象，决定源单取数结果 |
| ParamList | List<SqlParam> | Sql参数对象 | Y：查询参数对象 |

**插件示例：**

无

### OnGetSourceData

执行取数SQL，获取到了源单数据之后，触发本事件。

此时，已经拿到了源单需要迁移到下游单据的完整数据。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnGetSourceData(GetSourceDataEventArgs e); | |

**备注：**

进入此插件时，已经通过前面步骤中准备的源单过滤条件，找出符合过滤条件的源单数据

**触发时机：**

根据过滤条件，找出符合条件的源单数据后

**应用场景：**

为插件提供时机

1、干预源单数据集

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **GetSourceDataEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| SourceData | DynamicObjectCollection | 列表上选择的源单数据集合 | Y：选单源单数据集合 |
| DicFieldAlias | Dictionary<string, string> | 基础资料字段key与字段别名的关系字典 |  |
| QueryBuilderParemeter | QueryBuilderParemeter | 获取源单数据的查询参数对象 |  |
| SqlParamPKValues | SqlParam | 内码参数对象 |  |

#### 案例 – 库存请检单生成库存状态转换单时，强制拆单

**需求背景：**

库存请检单下推库存转换单时，请检单3个字段数据拆分成3行数据

**实现方案：**

转换插件中，在OnGetSourceData事件中，对需要拆单的源单行，进行复制、拆分，新增拆分行。

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Util;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Orm.Metadata.DataEntity;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.App;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.SCM.App;  using Kingdee.BOS.Contracts;  using System.ComponentModel;  using Kingdee.BOS.Core.SqlBuilder;  using System.Data;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.K3.BD.Contracts;  using Kingdee.BOS;  namespace Kingdee.K3.SCM.App.Stock.ServicePlugIn  {  /// <summary>  /// 库存请检单检验生成库存状态转换单插件  /// </summary>  [Description("库存请检单检验生成库存状态转换单插件")]  public class StkAppInspectCheck2StockConvert : AbstractConvertPlugIn  {  /// <summary>  /// 下推时拆分请检单  /// </summary>  public override void OnGetSourceData(GetSourceDataEventArgs e)  {  e.SourceData = GetSplitSourceData(e.SourceBusinessInfo, e.SourceData);  }  /// <summary>  /// 重新构建源数据，实现3个字段数据变3行的逻辑  /// </summary>  private DynamicObjectCollection GetSplitSourceData(BusinessInfo billInfo,  DynamicObjectCollection sourceData)  {  DynamicObjectCollection newSourceData = new DynamicObjectCollection(sourceData.DynamicCollectionItemPropertyType);  List<object> unitIds = new List<object>();  foreach (var oneSourceData in sourceData)  {  long secUnitId = Convert.ToInt64(oneSourceData["FSecUnitId"]);  if (secUnitId > 0 && !unitIds.Any(p => Convert.ToInt64(p) == secUnitId))  {  unitIds.Add(secUnitId);  }  }  DynamicObject[] units = null;  IViewService service = BOS.App.ServiceHelper.GetService<IViewService>();  BaseDataField secUintFld = billInfo.GetField("FSecUnitId") as BaseDataField;  if (unitIds.Count > 0)  {  units = service.LoadFromCache(this.Context, unitIds.ToArray(),  secUintFld.RefFormDynamicObjectType);  }  foreach (var oneSourceData in sourceData)  {  string roundType = "";  int precision = 0;  long secUnitId = Convert.ToInt64(oneSourceData["FSecUnitId"]);  if (units != null && secUnitId > 0)  {  DynamicObject unit = units.FirstOrDefault(p => Convert.ToInt64(p["Id"]) == secUnitId);  if (unit != null)  {  roundType = unit["RoundType"] == null ? "" : unit["RoundType"].ToString();  precision = Convert.ToInt32(unit["Precision"]);  }  }    RoundMode roundMode = GetRoundMode(roundType);  decimal baseQty = Convert.ToDecimal(oneSourceData["FBaseQty"]);  decimal secBaseQty = Convert.ToDecimal(oneSourceData["FSecQty"]);  decimal secRate = secBaseQty / baseQty;  decimal desBaseQty = (Convert.ToDecimal(oneSourceData["FBaseQuaQty"]) - Convert.ToDecimal(oneSourceData["FBaseSelQuaTranQty"]));  decimal desSecQty = 0;  //合格数量-合格转换选单数量>0（源行）  if (desBaseQty > 0)  {  desSecQty = secRate == 0 ? 0 : MathUtil.Round(desBaseQty \* secRate, precision, roundMode);  //复制行  DynamicObject newOneSource = (DynamicObject)ObjectUtils.CreateCopy(oneSourceData);//深度拷贝  newOneSource["FBaseQuaQty"] = desBaseQty;  newOneSource["FSecQty"] = desSecQty;  newOneSource["FStockConvertType"] = "1";//合格  newSourceData.Add(newOneSource);//添加源行  }  //不合格数量-不合格转换选单数量>0（新创建行）  desBaseQty = (Convert.ToDecimal(oneSourceData["FBaseFailQty"]) - Convert.ToDecimal(oneSourceData["FBaseSelFailTranQty"]));  if (desBaseQty > 0)  {  desSecQty = secRate == 0 ? 0 : MathUtil.Round(desBaseQty \* secRate, precision, roundMode);  //复制行  DynamicObject newOneSource = (DynamicObject)ObjectUtils.CreateCopy(oneSourceData);//深度拷贝  newOneSource["FBaseQuaQty"] = desBaseQty;  newOneSource["FSecQty"] = desSecQty;  newOneSource["FStockConvertType"] = "2";//不合格  newSourceData.Add(newOneSource);//添加源行  }  //报废数量-报废转换选单数量>0（新创建行）  desBaseQty = (Convert.ToDecimal(oneSourceData["FBaseScrapQty"]) - Convert.ToDecimal(oneSourceData["FBaseSelScrapTranQty"]));  if (desBaseQty > 0)  {  desSecQty = secRate == 0 ? 0 : MathUtil.Round(desBaseQty \* secRate, precision, roundMode);  //复制行  DynamicObject newOneSource = (DynamicObject)ObjectUtils.CreateCopy(oneSourceData);//深度拷贝  newOneSource["FBaseQuaQty"] = desBaseQty;  newOneSource["FSecQty"] = desSecQty;  newOneSource["FStockConvertType"] = "3";//报废  newSourceData.Add(newOneSource);//添加源行  }  }  return newSourceData;  }  }  } | |

### OnBeforeGroupBy

读取源单数据之后，开始按照转换规则 - 分组策略，对源单数据进行分组之前，触发本事件。

此时，已经解析完分组策略，提取了分组、合并依赖的字段，但并未实际分组。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnBeforeGroupBy(BeforeGroupByEventArgs e); | |

**备注：**

可以在此事件中，修改单据分组、分录合并规则，从而调整控制源单数据分组。

**触发时机：**

找出符合过滤条件的源单数据后，准备进行源单数据分组之前

**应用场景：**

为插件提供时机

1、干预源单数据集合分组条件

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeGroupByEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| SourceData | DynamicObjectCollection | 源单数据集合 | Y |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| GroupByMode | GroupByMode | 分组方式 |  |
| HeadGroupKey | string | 单据头分组字段 | Y |
| EntryGroupKey | string | 单据体分组字段 | Y |
| SubEntryGroupKey | string | 子单据体分组字段 | Y |

#### 案例 – BOM展开到领料单，按供应组织强制分单

**需求背景：**

BOM展开结果推简单生产领料单，不同的供应组织，需要下推为不同的领料单。

**实现方案：**

转换插件中，在OnBeforeGroupBy事件里，配置分组字段，修改对应分组属性，HeadGroupKey

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using Kingdee.BOS.Core.Metadata.QueryElement;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Util;  using Kingdee.K3.Core.BD;  using Kingdee.K3.Core.BD.ServiceArgs;  using Kingdee.K3.Core.MFG.EntityHelper;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.Core.SCM.STK.SP.Const;  using Kingdee.K3.SCM.App.ConvertPlugIn;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.SCM.Contracts.SP;  using Enums = Kingdee.K3.Core.MFG.EnumConst.Enums;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.K3.SCM.App.Utils;  namespace Kingdee.K3.SCM.App.Stock.ServicePlugIn  {  /// <summary>  /// BOM展开结果至简单生产领料单-单据转换插件  /// </summary>  [Description("BOM展开结果至简单生产领料单-单据转换插件")]  public class BomExpand2SpPickMtrlItemConvert : BaseBillConvertServicePlugIn  {  internal class RegFieldInfo  {  public string KeyName;  public string ProperyName;  public Type ProperyType;  public string ValueName;  public object DefaultValue;  }  private List<RegFieldInfo> lstRegField = new List<RegFieldInfo>  {  new RegFieldInfo(){KeyName = "fprdorgid\_reg", ProperyName = "prdorgid\_reg", ProperyType = typeof(long), ValueName = "fprdorgid", DefaultValue = 0},  new RegFieldInfo(){KeyName = "fworkshopid\_reg", ProperyName = "workshopid\_reg", ProperyType = typeof(long), ValueName = "fworkshopid", DefaultValue = 0},  new RegFieldInfo(){KeyName = "fownertypeid\_reg", ProperyName = "ownertypeid\_reg", ProperyType = typeof(string), ValueName = "fownertypeid", DefaultValue = ""},  new RegFieldInfo(){KeyName = "fownerid\_reg", ProperyName = "ownerid\_reg", ProperyType = typeof(long), ValueName = "fownerid", DefaultValue = 0},    // 用于校验上拉或下推数据行的基础资料是否已经分配到目标单的主业务组织  new RegFieldInfo(){KeyName = "ftargetorgid\_reg", ProperyName = "targetorgid\_reg", ProperyType = typeof(long), ValueName = "fstockorgid", DefaultValue = 0},  };  /// <summary>  /// 分组操作  /// </summary>  public override void OnBeforeGroupBy(BeforeGroupByEventArgs e)  {  base.OnBeforeGroupBy(e);  if (this.IsPush && e.GroupByMode == GroupByMode.GroupByField)  {  e.HeadGroupKey = string.IsNullOrWhiteSpace(e.HeadGroupKey) ?  string.Format("{0},FSUPPLYORGID", string.Join(",", lstRegField.Select(s => s.KeyName))) :  string.Format("{0},{1},FSUPPLYORGID", e.HeadGroupKey, string.Join(",", lstRegField.Select(s => s.KeyName)));  }  }  }  } | |

### OnCreateTarget

对源单分组完毕，并根据分组结果，创建好了目标单据数据包之后，触发本事件。

此时，已经创建好目标单据数据包，并填写了各字段默认值。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnCreateTarget(CreateTargetEventArgs e); | |

**备注：**

可以在此事件中，完善各个字段的默认值

**触发时机：**

源单数据分组后，根据分组创建完目标实体之后

**应用场景：**

为插件提供时机

1、干预根据分组数据创建起来的目标单据实体

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **CreateTargetEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| GroupingData | IEnumerable<object> | 分组后的源单数据集合 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| TargetExtendedDataEntities | ExtendedDataEntitySet | 扩展的数据实体数据集合 | Y：目标单据扁平化实体数据集合；扁平化的数据实体集合，这个对象会把元模型中的所有实体信息数据拉平进行管理，每个实体数据信息中包含了源单分组数据信息，通过此属性，可以区分扁平化的数据实体集合中，哪些实体数据信息是一组 |

**插件示例：**

无

### OnBeforeFieldMapping

在开始按照转换规则 - 字段映射策略，进行数据迁移之前，触发本事件。

此时，源单数据包、目标单数据包均已对应好，可以进行字段数据迁移。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnBeforeFieldMapping(BeforeFieldMappingEventArgs e); | |

**备注：**

因为需要把源单数据，逐行、逐字段的迁移到目标单，字段迁移次数会很多，默认情况下，不会触发每个字段的迁移事件(OnFieldMapping)。

如果需要监控单个字段的迁移过程，可以在此事件中，开启字段的迁移事件(OnFieldMapping)；

**触发时机：**

目标单据字段隐射发生前

**应用场景：**

为插件提供时机

1、是否在目标单据字段完成隐射和赋值的时候，触发OnFieldMapping事件

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BeforeFieldMappingEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| FireFieldMappingEvent | bool | 是否触发填充字段值更新时间 | Y：决定是否触发后续的OnFieldMapping事件 |

**插件示例：**

无

### OnFieldMapping

基于源单数据包，计算完目标单据的字段值，准备填写到目标单据数据包之前，触发此事件。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnFieldMapping(FieldMappingEventArgs e); | |

**备注：**

此事件默认不会被触发。插件在OnBeforeFieldMapping事件中设置e.FireFieldMappingEvent = true之后，本事件被开启，会被触发。

可以在此事件中，监控、修正需要写到目标单据上的字段值；

**触发时机：**

目标单据目标字段赋值前

**应用场景：**

为插件提供时机

1. 干预目标字段的值
2. 前提是：在OnBeforeFieldMapping事件中，对FireFieldMappingEvent属性设置其属性值为true

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **FieldMappingEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| TargetField | Field | 目标字段元数据信息 |  |
| TargetExtendDataEntitySet | ExtendedDataEntitySet | 目标单据扩展数据包，包含了全木目标单据，全部实体，全部行对象数据 |  |
| ConvertSource | object | 源单数据，可能是行集合，也坑内是单行（DataRow）数据 |  |
| MapValue | object | 将要填写到目标字段上的数据 | Y：最终填写到目标字段上的值 |
| Cancel | bool | 是否取消内置的字段值填充处理 |  |

#### 案例 – 采购订单变更，需要根据选项确定价格取值来源

**需求背景：**

采购订单推采购订单变更单，当对变更单明细信息单据体的原单价和新单价赋值时，如源单含税，变更单原单价和新单价是源单含税单价，否则是源单单价

**实现方案：**

转换插件中，OnFieldMapping事件中，通过获取当前字段所在实体的源单数据，得到源单是否含税，取源单对应字段数据为变更单新老单价赋值

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Data;  using Kingdee.BOS;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Resource;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.JSON;  using Kingdee.BOS.Util;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Core;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  public class PurchaseOrderToChangeService : AbstractConvertPlugIn  {  public override void OnFieldMapping(FieldMappingEventArgs e)  {  if (e.TargetField.Key == "FOPrice" || e.TargetField.Key == "FNPrice")  {  List<DynamicObject> sourceDatas = e.ExtendedDataEntity["ConvertSource"] as List<DynamicObject>;  if(sourceDatas.Count>0)  {  DynamicObject sourceObj=sourceDatas[0];  if (Convert.ToBoolean(sourceObj["FIsIncludedTax"]))  {  e.MapValue = sourceObj["FTaxPrice"];  }  else  {  e.MapValue = sourceObj["FPrice"];  }  }  else  {  throw new Exception("没有找到原单数据！");  }  }  }  }  } | |

### OnAfterFieldMapping

根据转换规则 - 字段映射策略，迁移完字段值之后，触发此事件。

此时，已经把源单数据包，都迁移到了目标单据数据包。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnAfterFieldMapping(AfterFieldMappingEventArgs e); | |

**备注：**

此事件触发之前，已完成目标单据的每个实体中的目标字段的赋值操作。可以在此事件中处理没有设置字段映射关系的字段的赋值和计算。

**触发时机：**

目标单据全部目标字段完成赋值后

**应用场景：**

为插件提供时机

1、干预目标单据数据结果

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterFieldMappingEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| TargetExtendDataEntitySet | ExtendedDataEntitySet | 目标单据扩展数据包，包含了全木目标单据，全部实体，全部行对象数据 | Y：目标单据所有实体数据 |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |

#### 案例 – 采购订单变更，记录变更单来源为下推

**需求背景：**

采购订单下推变更单，变更单单据头是否下推的标示置为是

**实现方案：**

转换插件OnAfterFieldMapping事件里，从扁平化实体数据中，找到单据头实体数据，修改这些单据头实体对应字段（是否下推）的值为true

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Data;  using Kingdee.BOS;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Resource;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.JSON;  using Kingdee.BOS.Util;  using Kingdee.BOS.App.Data;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Core;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  public class PurchaseOrderToChangeService : AbstractConvertPlugIn  {  public override void OnAfterFieldMapping(AfterFieldMappingEventArgs e)  {  ExtendedDataEntity[] heads = e.TargetExtendDataEntitySet.FindByEntityKey("FBillHead");  if (heads == null)  {  return;  }  foreach (ExtendedDataEntity head in heads)  {  head.DataEntity["FIsConvert"] = true;  }  }  }  } | |

### OnCreateLink

字段迁移完毕，创建关联关系子表，在单据上记录关联关系前，触发本事件。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnCreateLink(CreateLinkEventArgs e); | |

**备注：**

可以在此事件中，设置e.Cancel = true，略过关联关系的创建；

特别说明：关联关系主要用于反写、联查，如无必要，请勿取消。

**触发时机：**

创建对应关系前

**应用场景：**

为插件提供时机

1、是否使用系统内置的方法创建源单、目标单的对应关系

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **CreateLinkEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| DicSelectField | Dictionary  <string, string> | 字段key与字段别名的关系字典 |  |
| TargetExtendedDataEntities | ExtendedDataEntitySet | 目标单单据扩展实体数据集合 |  |
| Cancel | bool | 是否终止系统内置处理 | Y：是否取消创建对应关系的系统内置方法 |

**插件示例：**

无

### OnAfterCreateLink

关联关系子表已经创建并填写完毕之后，触发此事件。

此时，已经在关联子表中，记录了源单与目标单之间的关系。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnAfterCreateLink(CreateLinkEventArgs e); | |

**备注：**

可以在此事件中，对关联关系子表内容进行调整。

**触发时机：**

创建对应关系后

**应用场景：**

为插件提供时机

1、处理一些业务逻辑，在此特定事件点上（已完成对应关系的创建）

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **CreateLinkEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| DicSelectField | Dictionary  <string, string> | 字段key与字段别名的关系字典 |  |
| TargetExtendedDataEntities | ExtendedDataEntitySet | 目标单单据扩展实体数据集合 |  |
| Cancel | bool | 是否终止系统内置处理 |  |

**插件示例：**

无

### OnGetConvertBusinessService

数据从源单迁移完毕，开始执行转换规则 - 表单服务策略上配置的服务之前，触发此事件。

此时，已经创建好了服务执行列表，但并未实际执行服务。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnBeforeGetSourceData(BeforeGetSourceDataEventArgs e); | |

**备注：**

此事件中，通知插件，准备开始在目标单上执行指定表单服务, 插件可以添加自定义业务服务，实现IPushFormBusinessService即可，注意要在FormBusinessService里进行描述，PushClassName的属性一定要有值

**触发时机：**

执行表单服务前

**应用场景：**

为插件提供时机

1、添加自定义服务

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **ConvertBusinessServiceEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| FormBusinessServices | List<FormBusinessService> | 表单服务集合 | Y：包含插件自定义的服务集合 |

**插件示例：**

无

### AfterConvert

单据从源单到目标单已经转换完毕，输出生成的目标单数据包之前，触发此事件。

此时，目标单数据包已经构建完毕，不会再有变动了。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void AfterConvert(AfterConvertEventArgs e); | |

**备注：**

可以在此事件中，对目标单数据包进行最后的修订、补充。

特别说明：如果在本事件中为基础资料字段赋值，必须同时填写基础资料的内码、数据包两个属性（BaseDataField.RefIdDynamicProperty, BaseDataField.DynamicProperty），否则显示的下游单据上，基础资料字段是空白的。

**触发时机：**

单据转换完成，获取转换结果前

**应用场景：**

为插件提供时机

1、干预转换结果

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **AfterConvertEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| Result | ExtendedDataEntitySet | 单据转换输出的目标单据结果 | Y：转换结果 |
| Configuration | DefaultConvertPolicyElement | 当前转换动作关联的策略配置信息 |  |

#### 案例 – 采购申请推采购订单，根据VMI选项修订供应商

**需求背景：**

如果是采购申请单推VMI采购订单，如果VMI采购订单供应商不为空，判断此供应商是否支持VMI业务，如不支持，修改VMI采购订单表头供应商字段，置为空

**实现方案：**

转换插件，在AfterConvert事件中，找到单据头实体对应数据的供应商字段，根据判断，决定当前VMI采购订单供应商字段是否赋值为空

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Orm.Metadata;  using Kingdee.BOS.Orm.Metadata.DataEntity;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.BusinessEntity.Organizations;  using Kingdee.BOS.Resource;  using Kingdee.K3.SCM.Contracts;  using Kingdee.K3.SCM.App.Utils;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using Kingdee.BOS.App.Data;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  /// <summary>  /// 采购申请单到采购订单的单据转换插件  /// </summary>  public class PurReqToOrderService : AbstractConvertPlugIn  {  /// <summary>  /// 采购申请单转换到VMI采购订单  /// </summary>  bool isToVMI = false;  public override void AfterConvert(AfterConvertEventArgs e)  {  if (this.isToVMI)  {  ExtendedDataEntity[] data = e.Result.FindByEntityKey("FBillHead");  if (data != null && data.Length > 0)  {  foreach (ExtendedDataEntity entity in data)  {  DynamicObject billObj = entity.DataEntity;  long supplierId = Convert.ToInt64(billObj["SupplierId\_Id"]);  if (supplierId != 0)  {  var supplierObj = dealSupplierVMI(supplierId);  if (supplierObj == null)  {  billObj["SupplierId\_Id"] = 0;  billObj["SupplierId"] = null;  }  }  }  }  }  this.isToVMI = false;  }  private object dealSupplierVMI(long suggestSupplierId)  {  string sql = " select \* from t\_BD\_Supplier a inner join t\_BD\_SupplierBusiness b on a.FSUPPLIERID = b.FSUPPLIERID where a.FSUPPLIERID= @FSUPPLIERID and b.FVMIBUSINESS = '1'; ";  SqlParam param = new SqlParam("FSUPPLIERID", KDDbType.Int64, suggestSupplierId);  using (var reader = DBUtils.ExecuteReader(this.Context, sql, param))  {  if (reader.Read())  {  return suggestSupplierId;  }  else  {  return null;  }  }  }  }  } | |

## 选单前事件

在下游单据新增界面，点击选单按钮，显示源单列表之前，会调用单据转换引擎，拼接出源单筛选条件。此过程为选单前过程，也会触发单据转换插件事件。

被触发的事件如下：

1. OnInitVariable
2. OnParseFilterOptions
3. OnParseFilter

### OnInitVariable

初始化变量之后触发本事件；

本事件是下推、选单公共事件，[前文](#_OnInitVariable)已有介绍。

### OnParseFilterOptions

根据转换规则的配置，把目标单关键字段值，拼进过滤条件之后，触发本事件。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnParseFilterOptions(ParseFilterOptionsEventArgs e); | |

**备注：**

转换规则 - 字段映射策略中，可以勾选目标字段的”过滤”、”仅追加”选项。目标字段勾上此过滤选项后，选单时，会把该字段值作为过滤条件，传递给源单列表。

可以在此事件中，调整根据目标字段值生成的过滤条件。

本事件仅在选单前触发。

**触发时机：**

选单列表弹出前，准备选单列表过滤条件

**应用场景：**

为插件提供时机

1、为选单列表查询对象QueryBuilderParemeter准备过滤条件

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **ParseFilterOptionsEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| TargetData | DynamicObject | 目标单单据当前数据包 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| FilterOptionsSQL | string | 根据过滤条件生成的Sql | Y：拼接到查询过滤对象QueryBuilderParemeter的Where条件中 |

#### 案例 – 退料单选退料申请，追加条件：需求组织为货主

**需求背景：**

退料类型为检验退料，货主是供应商，退料类型是库存退料，需求组织是货主

**实现方案：**

OnParseFilterOptions事件中，修改参数的FilterOptionSQL属性，实现过滤

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.K3.SCM.App.Utils;  using Kingdee.K3.SCM.Contracts;  using Kingdee.BOS.JSON;  namespace Kingdee.K3.SCM.App.Pur.ServicePlugIn  {  //退料申请-->退货单 选单路线插件  public class MRAppToMRBConvertService : AbstractConvertPlugIn  {  public override void OnParseFilterOptions(ParseFilterOptionsEventArgs e)  {  string filter = string.Empty;  if (Convert.ToString(e.TargetData["MRTYPE"]).Equals("A"))  {  filter = String.Format(" FSUPPLIERID = {0} ", Convert.ToInt64(e.TargetData["OwnerIdHead\_Id"]));  }  else  {  filter = String.Format(" FREQUIREORGID = {0} ", Convert.ToInt64(e.TargetData["OwnerIdHead\_Id"]));  }  if (String.IsNullOrEmpty(e.FilterOptionsSQL))  {  e.FilterOptionsSQL = filter;  }  else  {  e.FilterOptionsSQL += " AND " + filter;  }  }  }  } | |

### OnParseFilter

解析完转换规则的选单条件策略之后，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnParseFilter)已经介绍。

## 选单后事件

用户在下游单据新增界面，点击选单菜单，显示源单列表，返回所选源单之后，开始选单后处理。选单后处理过程非常类似于下推，可以参阅前文[下推事件](#_下推事件)。

选单过程，触发的事件，也非常类似于下推过程，但稍有差异。

选单处理，按顺序触发如下事件：

1. OnInitVariable
2. OnQueryBuilderParemeter
3. OnInSelectedRow
4. OnBeforeGetSourceData
5. OnGetDrawSourceData
6. OnBeforeGroupBy
7. OnCreateDrawTarget
8. OnBeforeFieldMapping
9. OnAfterFieldMapping
10. OnCreateLink
11. OnAfterCreateLink
12. OnGetConvertBusinessService
13. AfterConvert

### OnInitVariable

初始化变量之后触发本事件；

本事件为下推、选单公共事件，[前文](#_OnInitVariable)已经介绍。

### OnQueryBuilderParemeter

取源单数据的参数对象(QueryBuilderParemeter)初始化完毕之后，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnQueryBuilderParemeter)已经介绍。

### OnInSelectedRow

把所选的源单内码，转换为取数条件之后，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnInSelectedRow)已经介绍。

### OnBeforeGetSourceData

调用取数服务，读取源单数据之前，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnBeforeGetSourceData)已经介绍。

### OnGetDrawSourceData

读取了所选的源单数据之后触发，此时，已经拿到了源单完整数据；

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnGetDrawSourceData(GetDrawSourceDataEventArgs e); | |

**备注：**

选单时触发，获取上拉时的源单数据行，进入此事件时，已经获取到了源单数据集合

**触发时机：**

在选单列表选中数据返回后，OnBeforeGetSourceData事件后

**应用场景：**

为插件提供时机

1、干预选单源单数据集合

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **GetDrawSourceDataEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| SourceBusinessInfo | BusinessInfo | 源单单据元数据信息 |  |
| SourceData | DynamicObjectCollection | 列表上选择的源单数据集合 | Y：选单源单数据集合 |
| DicFieldAlias | Dictionary<string, string> | 基础资料字段key与字段别名的关系字典 |  |
| QueryBuilderParemeter | QueryBuilderParemeter | 获取源单数据的查询参数对象 |  |
| SqlParamPKValues | SqlParam | 内码参数对象 |  |

#### 案例 – 生产入库单选工序转移单，拆分合格数量、废品数量

**需求背景：**

生产入库单上拉工序转移单得到的工序转移单数据列表，如基本单位合格数量大于基本单位合格入库选单数量，将废品相关数量字段清零等进行相关源单数据修改

**实现方案：**

插件OnGetDrawSourceData事件中，对工序转移单满足上述条件的业务数据，重新赋值

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Enums;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn.Args;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Util;  using Kingdee.K3.Core.MFG;  using Kingdee.K3.Core.MFG.EnumConst;  using Kingdee.K3.Core.MFG.EntityHelper;  using Kingdee.K3.MFG.App;  using Kingdee.K3.MFG.App.ConvertPlugIn;  using Kingdee.K3.MFG.App.ServiceValidator;  using Kingdee.K3.MFG.Contracts;  using Kingdee.K3.MFG.Contracts.PRD;  namespace Kingdee.K3.MFG.SFC.App.BillConvertServicePlugIn  {  /// <summary>  /// 工序转移单至生产入库单\_单据转换插件  /// </summary>  [Description("工序转移单至生产入库单\_单据转换插件")]  public class OperTrans2InStockConvert : BaseBillConvertServicePlugIn  {  /// <summary>  /// 选单时，执行SQL,读取源单数据  /// </summary>  public override void OnGetDrawSourceData(GetDrawSourceDataEventArgs e)  {  e.SourceData = this.GetSplitSourceData(e.SourceData);  }  /// <summary>  /// 重新构建源数据  /// </summary>  private DynamicObjectCollection GetSplitSourceData(DynamicObjectCollection sourceData)  {  DynamicObjectCollection newSourceData = new DynamicObjectCollection(sourceData.DynamicCollectionItemPropertyType);  foreach (var oneSourceData in sourceData)  {  // 基本单位合格数量-基本单位合格入库选单数量>0  if (oneSourceData.GetDynamicObjectItemValue<decimal>("FQualifiedBaseQty")   * oneSourceData.GetDynamicObjectItemValue<decimal>("FQuaInStockSelBaseQty") > 0)   {  // 深度拷贝复制行  DynamicObject newOneSource = ObjectUtils.CreateCopy(oneSourceData) as DynamicObject;  // 废品相关数量清零  newOneSource["FUnqualifiedBaseQty"] = decimal.Zero;  newOneSource["FUnQuaInStockSelBaseQty"] = decimal.Zero;  // 入库类型=合格入库  newOneSource["FInStockType"] = Convert.ToInt32(Enums.Enu\_InStockType.KdConformity);  // 赋值分组标识  newOneSource["FGroupById"] = SequentialGuid.NewGuid().ToString();  // 添加新复制行  newSourceData.Add(newOneSource);  }  // 基本单位废品数量-基本单位废品入库选单数量>0  if (oneSourceData.GetDynamicObjectItemValue<decimal>("FUnqualifiedBaseQty")   * oneSourceData.GetDynamicObjectItemValue<decimal>("FUnQuaInStockSelBaseQty") > 0)   {  // 深度拷贝复制行  DynamicObject newOneSource = ObjectUtils.CreateCopy(oneSourceData) as DynamicObject;  // 废品相关数量清零  newOneSource["FQualifiedBaseQty"] = decimal.Zero;  newOneSource["FQuaInStockSelBaseQty"] = decimal.Zero;  // 入库类型=报废品入库  newOneSource["FInStockType"] = Convert.ToInt32(Enums.Enu\_InStockType.kdScrapConvormity);  // 赋值分组标识  newOneSource["FGroupById"] = SequentialGuid.NewGuid().ToString();  // 添加新复制行  newSourceData.Add(newOneSource);  }  }  return newSourceData;  }  }  } | |

### OnBeforeGroupBy

读取源单数据之后，开始按照转换规则 - 分组策略，对源单数据进行分组之前，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnBeforeGroupBy)已经介绍。

### OnCreateDrawTarget

对源单行进行分组、合并之后，但还没有构建目标单据行之前触发。此时，源单行已经分好组，但目标单据数据包还未改变，依然是选单之前的数据包。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnCreateDrawTarget(CreateDrawTargetEventArgs e); | |

**备注：**

本事件仅在选单过程触发。可以在此事件中，调整源单分组结果。

**触发时机：**

根据分组后的源单数据，初始化目标单据实体数据

**应用场景：**

为插件提供时机

1、干预目标单据实体数据

**插件接口介绍：**

|  |  |  |  |
| --- | --- | --- | --- |
| **CreateDrawTargetEventArgs** | | | |
| **属性** | **类型** | **描述** | **关键属性** |
| Context | Context | 当前系统上下文全局信息 |  |
| GroupingData | IEnumerable<object> | 分组后的源单数据集合 |  |
| TargetData | DynamicObject | 目标单单据数据包 |  |
| TargetBusinessInfo | BusinessInfo | 目标单单据元数据信息 |  |
| TargetExtendedDataEntities | ExtendedDataEntitySet | 扩展的数据实体数据集合 | Y：目标单据扁平化实体数据集合 |

**插件示例：**

无

### OnBeforeFieldMapping

在开始按照转换规则 - 字段映射策略，进行数据迁移之前，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnBeforeFieldMapping)已经介绍。

### OnFieldMapping

基于源单数据包，计算完目标单据的字段值，准备填写到目标单据数据包之前，触发此事件。

本事件为下推、选单公共事件，[前文](#_OnFieldMapping)已经介绍。

### OnAfterFieldMapping

根据转换规则 - 字段映射策略，迁移完字段值之后，触发此事件。

本事件为下推、选单公共事件，[前文](#_OnAfterFieldMapping)已经介绍。

### OnCreateLink

字段迁移完毕，创建关联关系子表，在单据上记录关联关系前，触发本事件。

本事件为下推、选单公共事件，[前文](#_OnCreateLink)已经介绍。

### OnAfterCreateLink

关联关系子表已经创建并填写完毕之后，触发此事件。

本事件为下推、选单公共事件，[前文](#_OnAfterCreateLink)已经介绍。

### OnGetConvertBusinessService

数据从源单迁移完毕，开始执行转换规则 - 表单服务策略上配置的服务之前，触发此事件。

本事件为下推、选单公共事件，[前文](#_OnGetConvertBusinessService)已经介绍。

### AfterConvert

单据从源单到目标单已经转换完毕，输出生成的目标单数据包之前，触发此事件。

本事件为下推、选单公共事件，[前文](#_AfterConvert)已经介绍。

# 附录：列表插件提供的单据转换事件

在单据列表插件上，也为单据转换处理，提供了两个事件，供用户调整目标单据、实现第三方下推、控制采用的转换规则等。

列表插件基类为：Kingdee.BOS.Core.List.PlugIn.AbstractListPlugIn；

下文列出列表上的两个下推事件：

### OnShowConvertOpForm

用户在单据列表上点击下推菜单，弹出下推界面前，触发本事件。

此时，已经准备好可选的下推目标单据。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnShowConvertOpForm(ShowConvertOpFormEventArgs e) | |

**备注：**

此事件虽然属于单据、列表插件，但应用在单据转换过程中，也是单据转换第一个插件。

插件可以修改本事件的参数，增删改允许下推的下游单据。

如果需要支持第三方下推，可以在此事件中，设置与本单对应，用于实际下推的第三方单据内码。

**触发时机：**

在源单单据上点“下推”或在目标单单据上点“选单”，弹出选择转换路线窗口前

**应用场景：**

为插件提供时机干预：

1. 目标单据范围（Bills）
2. 目标单单据替代关系（ReplaceRelations）
3. 源单单据信息（SelectedRows）
4. 目标单单据组织信息（OptionalTargetOrgIds）

**插件接口参数介绍：**

|  |  |  |
| --- | --- | --- |
| **ShowConvertOpFormEventArgs** | | |
| **属性** | **类型** | **描述** |
| Bills | object | 选单或下推时的目标单据集合,通过改变此集合，可改变业务流程走向 |
| SelectedRows | ListSelectedRow[] | 当前选择的单据集合 |
| ReplaceRelations | List<ReplaceRelation> | 源目标单对应关系，适用于第三方选单下推，如生产订单推领料单，其实是生产用料清单推领料单 |
| OptionalTargetOrgIds | Dictionary<string, List<long>> | 目标单据的可选组织列表 |
| ConvertOperation | FormOperationEnum | 单据转换操作类型枚举 |
| Cancel | bool | 是否取消当前操作 |

#### 案例 – 单体组织，要货申请单不允许下推直接调拨单

**需求背景1：**

要货申请单列表下推或下查时，如系统启用多组织，不可下推到直接调拨单，否则可以

**实现方案1：**

要货申请单列表插件中，在OnShowConvertOpForm事件，根据事件传入参数的ConvertOperation属性，判断当前操作是否是下推或下查操作，由当前系统上下文对象this.Context的IsMultiOrg属性判断系统是否启用多组织，对事件传入参数的Bills做过滤

**插件示例1：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Data;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.Resource;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core.Metadata.ConvertElement;  namespace Kingdee.K3.SCM.DRP.Business.PlugIn  {  /// <summary>  /// 要货申请单 单据列表插件  /// </summary>  public class DRPNeedApplicationList : AbstractListPlugIn  {  public override void OnShowConvertOpForm(ShowConvertOpFormEventArgs e)  {  if (e.ConvertOperation == FormOperationEnum.Push || e.ConvertOperation == FormOperationEnum.TrackDown)  {  var bills = e.Bills as List<ConvertBillElement>;    if (bills != null && bills.Count > 0)  {  if (this.Context.IsMultiOrg)  {  e.Bills = bills.Where(c => !c.FormID.Equals("STK\_TransferDirect", StringComparison.OrdinalIgnoreCase)).ToList();  }  else  {  e.Bills = bills.Where(c => c.FormID.Equals("STK\_TransferDirect", StringComparison.OrdinalIgnoreCase)).ToList();  }  }  }  }  }  } | |

#### 案例 - 生产补料单，实现第三方下推后联查

**需求背景2：**  
生产补料单的直接源单为用料清单，但是上查时，却需要查到间接源单 - 生产订单；并且上查投料单时，也可以查询到生产订单。

**实现方案2：**

补料单表单插件，在OnShowConvertOpForm事件中，根据当前操作类型，为Bills增加生产订单，并且配置上查投料单时，其实是从用料清单到生产订单的路线

**插件示例2：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Data;  using System.Linq;  using System.Text;  using Kingdee.BOS.Core.CommonFilter;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Permission;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.K3.Core.MFG;  using Kingdee.K3.Core.MFG.EntityHelper;  using Kingdee.K3.MFG.ServiceHelper;  using Kingdee.K3.Core.MFG.Utils;  using Kingdee.K3.Core.MFG.EnumConst;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using System.Collections.Generic;  using Kingdee.BOS.BusinessEntity.BillTrack;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.K3.MFG.ServiceHelper.ENG;  using Kingdee.K3.Core.MFG.EnumConst.PRD;  using Kingdee.BOS.Core.DependencyRules;  using Kingdee.K3.MFG.Common.BusinessEntity.PRD;  using Kingdee.K3.Core.BD.ServiceArgs;  using Kingdee.K3.BD.ServiceHelper;  using Kingdee.K3.Core.BD;  namespace Kingdee.K3.MFG.PRD.Business.PlugIn.Bill  {  /// <summary>  /// 生产补料单表单插件  /// </summary>  public partial class FeedMtrlEdit : BaseControlEdit  {  public override void OnShowConvertOpForm(ShowConvertOpFormEventArgs e)  {  switch (e.ConvertOperation)  {  case FormOperationEnum.TrackUp: //上查  List<ConvertBillElement> lstConvertBills = null;  if (e.Bills != null)  {  lstConvertBills = e.Bills as List<ConvertBillElement>;  }  if (lstConvertBills == null)  {  lstConvertBills = new List<ConvertBillElement>();  }  ConvertBillElement convertBillElement = new ConvertBillElement();  convertBillElement.FormID = "PRD\_MO";  lstConvertBills.Add(convertBillElement);  e.AddReplaceRelation("PRD\_PPBOM", "PRD\_MO");  e.Bills = lstConvertBills;  break;  }  }  }  } | |

### OnGetConvertRule

在用户确认需要下推的目标单、下推采用的转换规则之后，触发本事件。

此时，目标单据已经确定、转换规则也已经确定。

#### 语法

**虚方法定义**

|  |  |
| --- | --- |
| C# |  |
| public virtual void OnGetConvertRule(GetConvertRuleEventArgs e) | |

**备注：**

此事件虽然属于单据、列表插件，但应用在单据转换过程中。

可以在此事件中，更改用户所选的转换规则；

**触发时机：**

目标单据选择窗口弹出后，点击确定，正式进入单据转换前

**应用场景：**

为插件提供时机

1、修改当前转换操作的路线

**插件接口介绍：**

|  |  |  |
| --- | --- | --- |
| **GetConvertRuleEventArgs** | | |
| **属性** | **类型** | **描述** |
| SourceFormId | string | 源单单据唯一标示 |
| TargetFormId | string | 目标单单据唯一标示 |
| ConvertOperation | FormOperationEnum | 单据转换操作类型枚举（选单、下推） |
| Rules | object | 可选的转换规则集合 |
| Rule | object | 当前选择的转换规则 |
| DynamicFormShowParameter | DynamicFormShowParameter | 控制源单显示的参数包（仅上拉过程有效） |

#### 案例 – 生产领料单选BOP展开单据时，调整源单与转换规则

**需求背景：**

生产线生产领料单，当前操作是选单，并且源单单据是**BOP展开**单据时，当前转换规则修改为从**BOP正向展开**模型到当前单据，待显示的源单列表，也调整为**BOP正向展开**列表。

**实现方案**

通过当前参数的ConvertOperation属性判断是否选单操作，根据ConvertServiceHelper提供的GetConvertRules方法，获取源单是BOP正向展开模型到当前模型的转换规则，并赋值给当前参数的Rule属性

**插件示例：**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using Kingdee.BOS.Core.DependencyRules;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.Operation;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Enums;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using Kingdee.BOS.Core.Metadata.ConvertElement.ServiceArgs;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Orm;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using Kingdee.BOS.Util;  using Kingdee.K3.BD.ServiceHelper;  using Kingdee.K3.Core.BD;  using Kingdee.K3.Core.BD.ServiceArgs;  using Kingdee.K3.Core.MFG.EntityHelper;  using Kingdee.K3.Core.MFG.EnumConst;  using Kingdee.K3.Core.MFG.Utils;  using Kingdee.K3.MFG.REM.Common.BusinessEntity;  using Kingdee.K3.MFG.REM.Common.Core.EnumConst;  using Kingdee.K3.MFG.ServiceHelper.ENG;  using System;  using System.Collections.Generic;  using System.Linq;  using Kingdee.K3.Core.BD.Const;  using Kingdee.K3.MFG.ServiceHelper;  using Kingdee.K3.Core.MFG;  using Kingdee.BOS.Core.SqlBuilder;  namespace Kingdee.K3.MFG.REM.Business.PlugIn.Bill  {  /// <summary>  /// 生产线生产领料单表单插件  /// </summary>  public class PickMtrlEdit : BaseControlEdit  {  /// 选单/下推过程, 确定单据转换规则  public override void OnGetConvertRule(GetConvertRuleEventArgs e)  {  if (e.ConvertOperation == FormOperationEnum.Draw && string.Equals(e.SourceFormId, "ENG\_BopExpand"))  {  ConvertRuleElement rule = ConvertServiceHelper.GetConvertRules(this.Context, "ENG\_BopExpandBill", e.TargetFormId).FirstOrDefault(f => f.IsDefault);  if (rule != null)  {  DynamicFormShowParameter param = new DynamicFormShowParameter();  param.FormId = e.SourceFormId;  param.OpenStyle.ShowType = ShowType.NonModal;  param.CustomParams["ShowMode"] = ((int)BOSEnums.Enu\_ListType.SelBill).ToString();  param.ParentPageId = this.View.PageId;  e.DynamicFormShowParameter = param;  }  e.Rule = rule;  }  }  }  } | |