K/3 Cloud BOS 插件开发百问百答

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No** | **日期** | **编制\修订** | **校对** | **批准** | **修改的章节号** |
| V1.0 | 20150428 | 张晋博 | 丁振华 | 赖碧云 | 初始版本，共39个问题 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

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

[K/3 Cloud BOS 插件开发百问百答 1](#_Toc418150550)

[1. 入门篇 4](#_Toc418150551)

[1.1. 如何创建并启用各类插件 4](#_Toc418150552)

[1.1.1. 如何创建并启用动态表单插件 4](#_Toc418150553)

[1.1.2. 如何创建并启用单据编辑插件 8](#_Toc418150554)

[1.1.3. 如何创建并启用基础资料编辑插件 9](#_Toc418150555)

[1.1.4. 如何创建并启用列表插件 10](#_Toc418150556)

[1.1.5. 如何创建并启用操作插件 11](#_Toc418150557)

[1.1.6. 如何创建并启用账表取数插件 15](#_Toc418150558)

[1.1.7. 如何创建并启用单据转换插件 18](#_Toc418150559)

[1.1.8. 如何创建并启用反写插件 18](#_Toc418150560)

[1.2. 为何插件不起作用，如何排查 20](#_Toc418150561)

[2. 进阶篇 21](#_Toc418150562)

[2.1. 插件代码，如何自行调用单据操作服务 21](#_Toc418150563)

[2.1.1. 如何加载(Load)单据 21](#_Toc418150564)

[2.1.2. 如何调用保存(Save)操作 25](#_Toc418150565)

[2.1.3. 如何调用提交(Submit)操作 28](#_Toc418150566)

[2.1.4. 如何调用审核/反审核(Audit/UnAudit)操作 31](#_Toc418150567)

[2.1.5. 如何调用禁用(Disabled)操作 34](#_Toc418150568)

[2.1.6. 如何删除(Delete)单据 37](#_Toc418150569)

[2.2. 调用单据操作后，如何判断结果，并处理交互提示 40](#_Toc418150570)

[2.3. 插件给字段赋值后，为什么不执行字段”值更新属性”上配置的表单服务 44](#_Toc418150571)

[2.4. 在动态表单插件中，如何修改界面数据 47](#_Toc418150572)

[2.4.1. 如何新增单据体行 47](#_Toc418150573)

[2.4.2. 如何删除单据体行 49](#_Toc418150574)

[2.4.3. 如何给普通字段赋值 50](#_Toc418150575)

[2.4.4. 如何给基础资料字段赋值 51](#_Toc418150576)

[2.4.5. 如何动态设置下拉列表字段的可选项 52](#_Toc418150577)

[2.5. 字段标识(Key)、字段名(FieldName)、属性名(PropertyName)的差别 53](#_Toc418150578)

[2.6. 在动态表单中，如何实现文件下载 54](#_Toc418150579)

[2.7. 如何支持用户把本地文件上传到服务器 58](#_Toc418150580)

[2.8. 以实例说明如何为动态表单取数、填充 59](#_Toc418150581)

[2.9. 如何为预置账表，添加字段 60](#_Toc418150582)

[2.10. 如何获取基础资料的属性值 61](#_Toc418150583)

[2.11. 保存前，如何为新建的单据数据包、单据体行，填写主键 62](#_Toc418150584)

[2.12. 在动态表单插件中，如何显示指定界面 63](#_Toc418150585)

[2.13. 如何为收款单，往来单位类型，添加新的基础资料 65](#_Toc418150586)

[2.14. 如何创建、调试自定义执行计划 67](#_Toc418150587)

[2.15. 如何通过按钮实现F8选择返回效果 68](#_Toc418150588)

[2.16. 如何在操作插件中，自定义操作校验器并启用 69](#_Toc418150589)

[2.17. 如何实现超链接，打开外部URL地址 74](#_Toc418150590)

[2.18. 如何获取基础资料字段的类别(FormId) 75](#_Toc418150591)

[3. 高级篇 76](#_Toc418150592)

[3.1. 如何给单据体表格，增加自定义过滤行 76](#_Toc418150593)

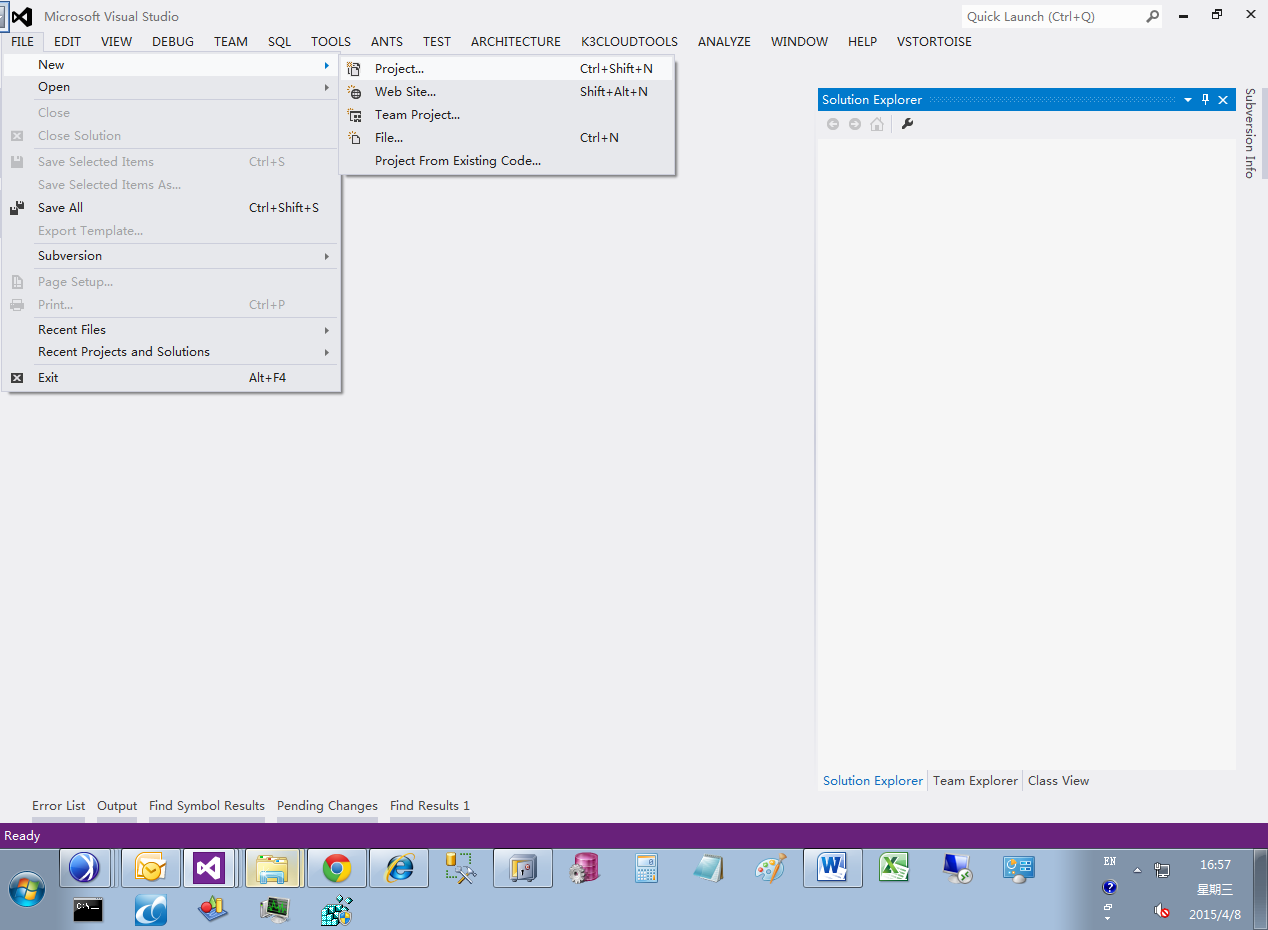
# 入门篇

## 如何创建并启用各类插件

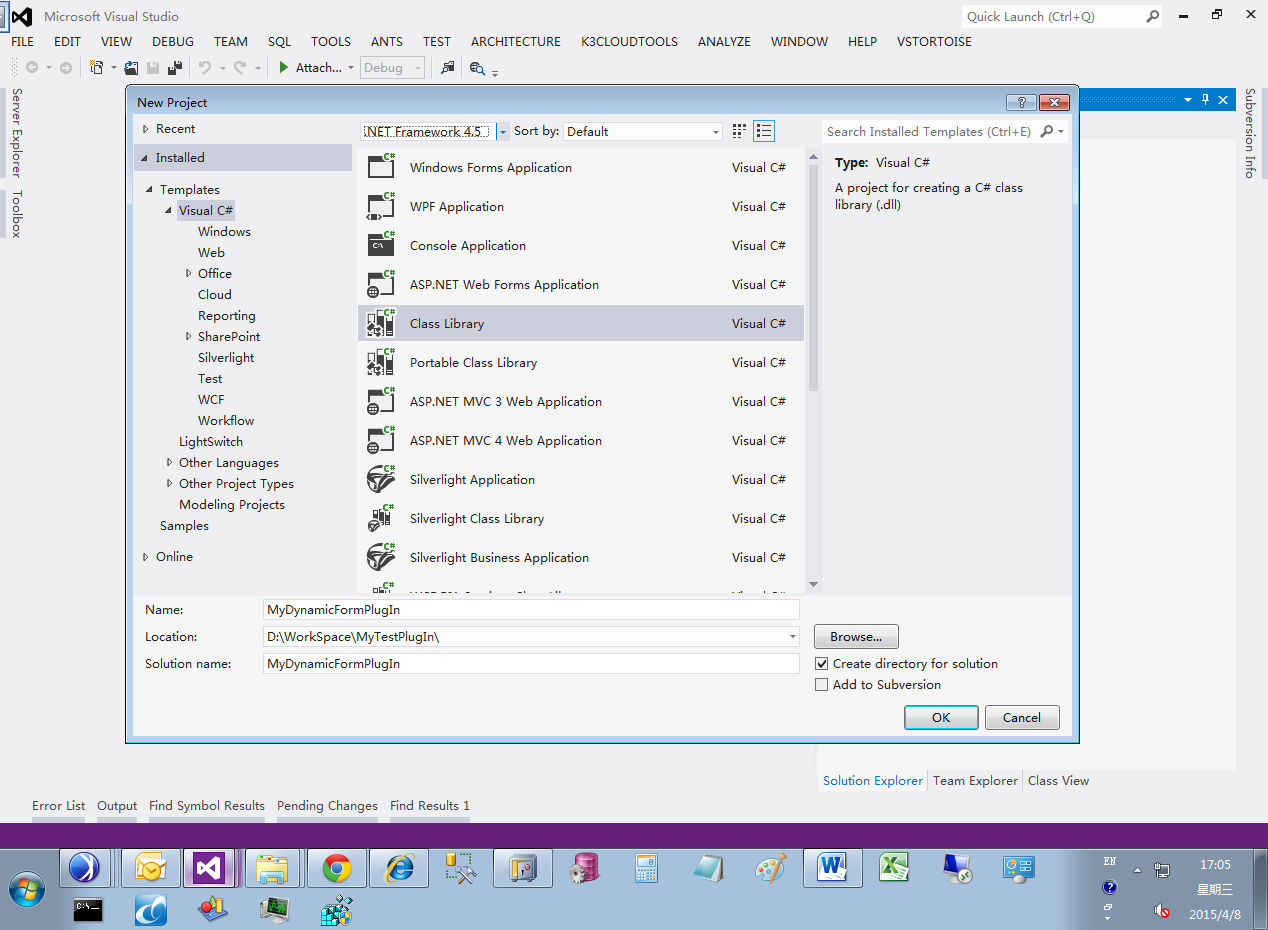
### 如何创建并启用动态表单插件

**创建插件工程**

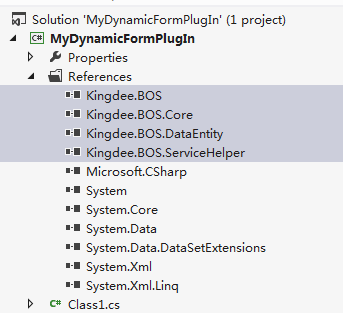
* 动态表单插件，运行在K/3 Cloud的Web服务层，需要建立一个独立的工程管理动态表单插件类；
* 动态表单插件工程，建议命名格式为[开发商][.K3][.业务领域][.子系统].Business.PlugIn；如JD.K3.SCM.PUR.Business.PlugIn；
* 创建工程过程：Visual Studio → New → Project



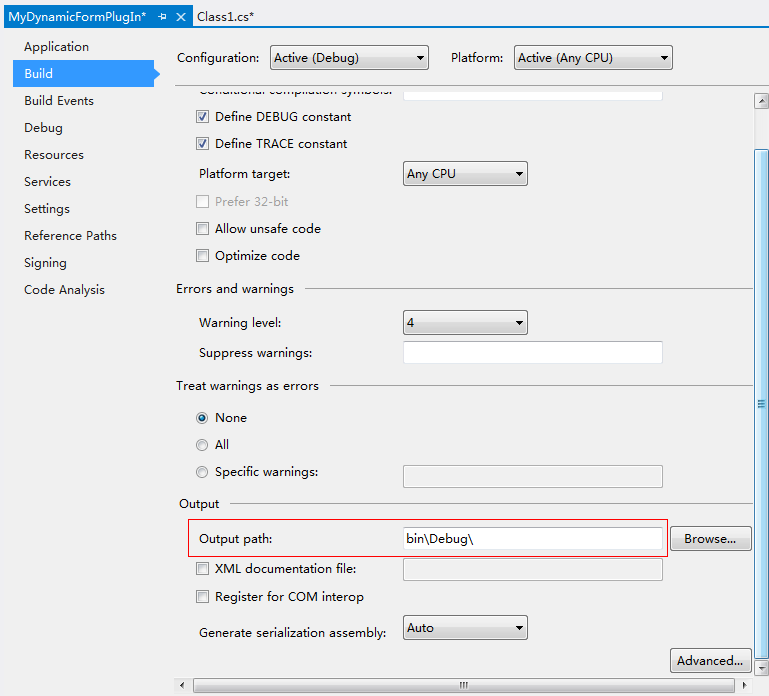
* Visual C# → Class Library → 选择工程文件存放位置 → OK



* 工程References右键，添加工程引用，所添加组件都来自Cloud网站的website\bin路径



* 修改插件工程输出路径为Cloud网站路径website\bin：选中插件工程 → 右键 → Property（属性） → Build页签 → Output Path（输出路径）

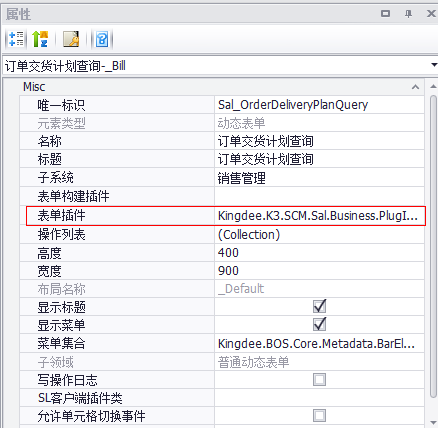


* 修改当前类（Class1）的继承对象为AbstractDynamicFormPlugIn
* 类文件引用基类（AbstractDynamicFormPlugIn）所在命名空间

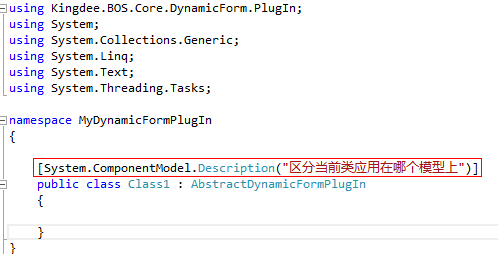
|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class1 : AbstractDynamicFormPlugIn  {  }  } | |

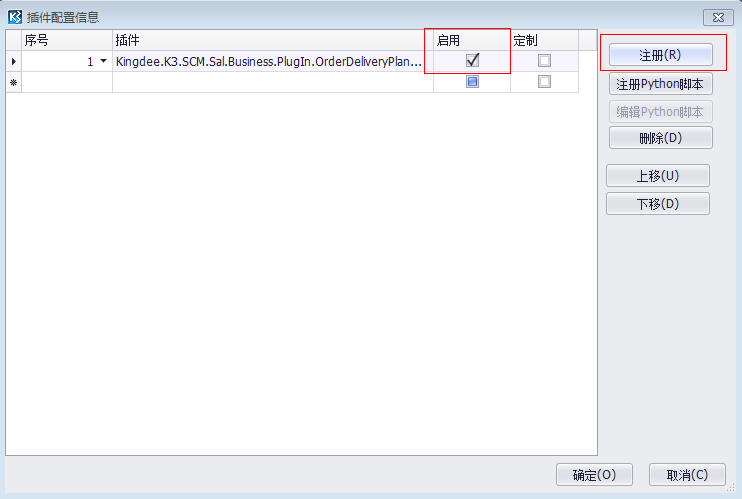
**注册并启用插件**

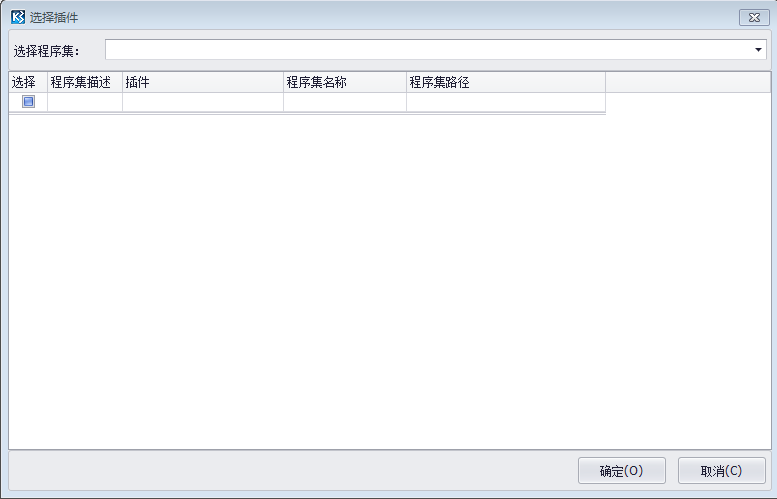
* 插件编写完毕，编译工程，输出类库\*.dll文件到WebSite\Bin目录；
* 打开BOSIDE，定位到动态表单模型整体属性，配置表单插件属性



* 点开表单插件属性，注册编译生成的插件工程组件，从website\bin路径下找到插件工程组件，选择插件工程组件下的插件类，一般在插件类上使用微软元数据描述，在注册插件时区分不同的类，选择注册插件后，注意勾选“启用”复选框







### 如何创建并启用单据编辑插件

**创建单据编辑插件**

* 单据编辑插件，运行在K/3 Cloud的Web服务层，重用动态表单插件工程；
* 创建插件类的过程，与创建动态表单插件几乎一样，唯一区别是单据编辑插件继承的基类是AbstractBillPlugIn
* 插件类引用基类所在命名空间

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class2 : AbstractBillPlugIn  {  //TODO  }  } | |

**启用单据编辑插件**

* 与启用动态表单插件一样

### 如何创建并启用基础资料编辑插件

**如何创建基础资料编辑插件**

* 基础资料编辑插件，运行在K/3 Cloud的Web服务层，重用动态表单插件工程；
* 与创建单据编辑插件一样，唯一区别是基础资料编辑插件继承的基类是AbstractBasePlugIn，目前AbstractBasePlugIn与AbstractBillPlugIn完全一样，但不论从模型分类角度或从未来发展趋势看，还是请大家严格遵守插件体系，即：从AbstractBasePlugIn继承

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Base.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class3 : AbstractBasePlugIn  {  //TODO  }  } | |

**如何启用基础资料编辑插件**

* 与启用动态表单插件一样

### 如何创建并启用列表插件

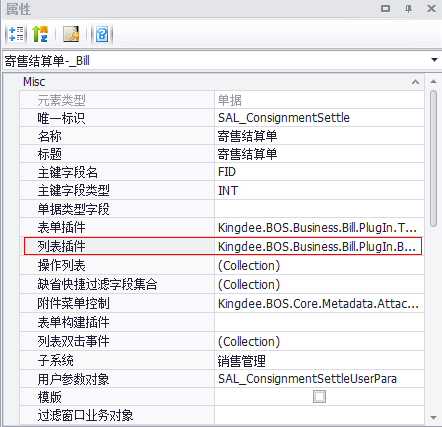
**如何创建列表插件**

* 列表插件，运行在K/3 Cloud的Web服务层，重用动态表单插件工程；
* 与创建表单插件一样，唯一区别是列表插件从AbstractListPlugIn继承出来

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.List.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class4 : AbstractListPlugIn  {  //TODO  }  } | |

**如何启用列表插件**

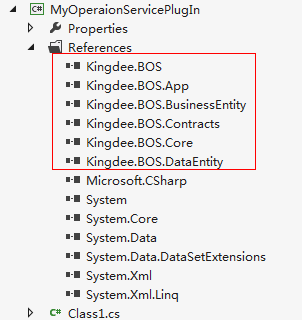
* 与启用动态表单插件一样，唯一区别是注册列表插件的位置是模型的列表插件属性



### 如何创建并启用操作插件

**如何创建操作插件**

* 操作插件，运行在K/3 Cloud的App服务层，需要单独创建一个工程管理操作插件类；
* 操作插件工程，命名规范：[开发商][.K3][.业务领域][.子系统].App.ServicePlugIn，如JD.K3.SCM.PUR.App.ServicePlugIn
* 操作插件工程类型也是Class library，编译输出的路径同样是Website\bin；
* 操作插件工程类型，因为运行层次不一样，与动态表单插件工程引用的类库\*dll稍有不同：

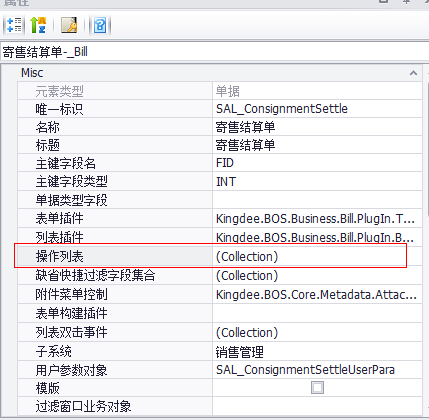


* 新建操作插件类的过程，与创建动态表单插件类似，区别在于操作插件继承的基类是AbstractOperationServicePlugIn

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个操作上")]  public class Class1 : AbstractOperationServicePlugIn  {  }  } | |

**如何启用操作插件**

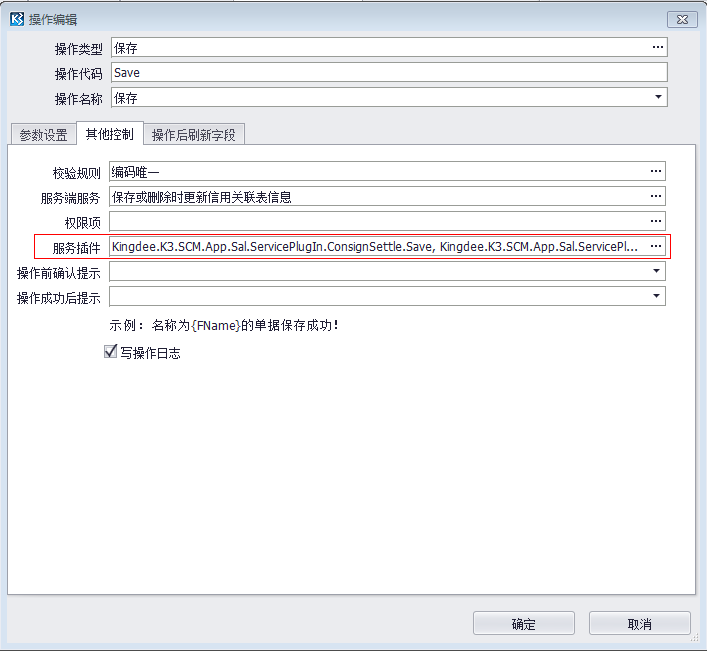
* 打开BOSIDE，点开模型操作列表



* 例如当前操作插件是应用在保存操作上，选择保存操作，点编辑



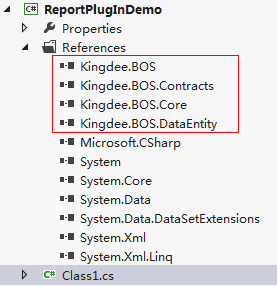
* 其他控制页签，服务插件



### 如何创建并启用账表取数插件

**如何创建账表取数插件**

* 账表取数插件，运行在K/3 Cloud的App服务层，需单独创建一个工程(Class library)；
* 账表取数插件工程，命名规范：[开发商][.K3][.业务领域][.子系统].App.ReportPlugIn，如JD.K3.SCM.PUR.App.ReportPlugIn；
* 插件工程引用组件（包括但不限于此，也可参考操作插件工程引用的组件）：

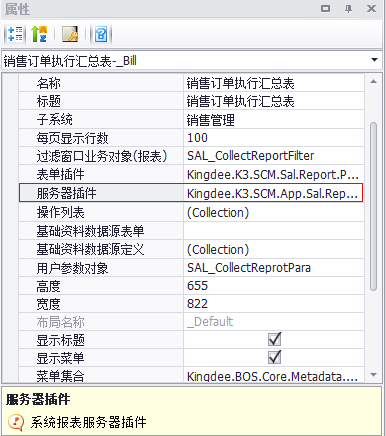


* 与创建动态表单插件类似，区别在于帐表服务器插件继承自SysReportBaseService
* 插件类文件引用基类所在命名空间

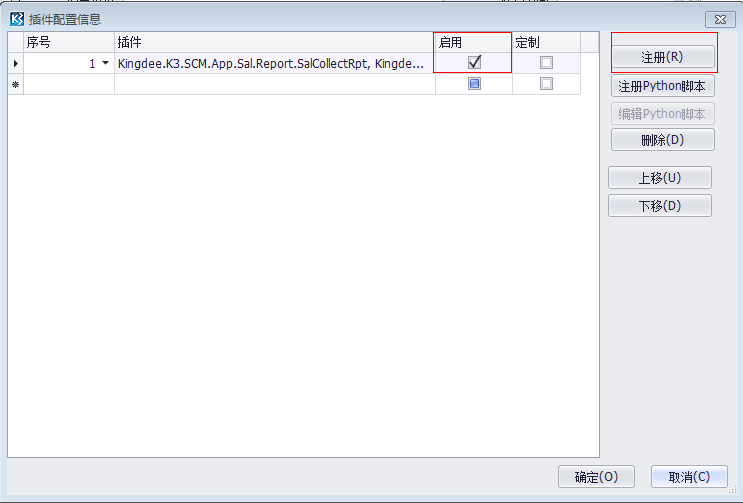
|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts.Report;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ReportPlugInDemo  {  public class Class1 : SysReportBaseService  {  //TODO  }  } | |

**如何启用账表服务器插件**

* 进入BOS IDE，打开账表模型，设置服务器插件属性



* 注册编译好的账表服务器插件，并点击启用复选框



### 如何创建并启用单据转换插件

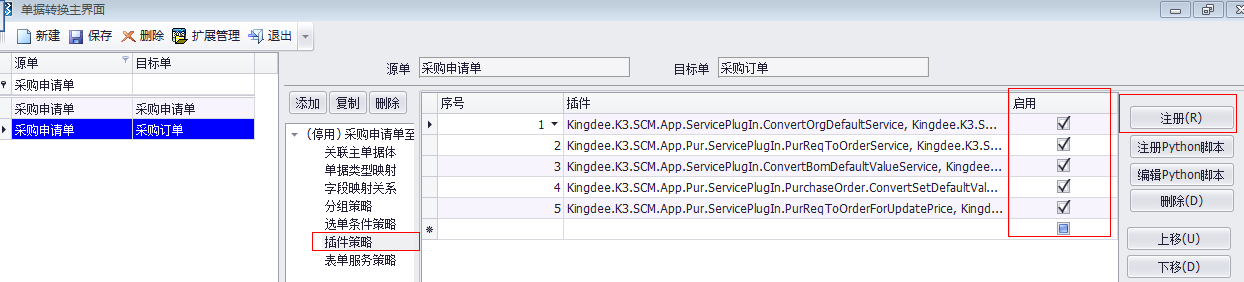
**如何创建单据转换插件**

* 单据转换插件，运行在K/3 Cloud的App服务层，重用操作插件工程；
* 创建插件类，与创建动态表单插件类类似，继承基类是AbstractConvertPlugIn
* 插件类文件引用基类所在命名空间

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Metadata.ConvertElement.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BusinessFlowPlugIn  {  public class Class1 : AbstractConvertPlugIn  {  //TODO  }  } | |

**如何启用单据转换插件**

* 打开BOSIDE，打开单据转换设计界面，插件策略，注册编译生成的单据转换插件，并启用



### 如何创建并启用反写插件

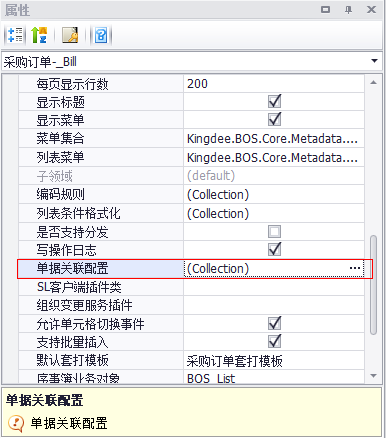
**如何创建反写插件**

* 反写插件，运行在K/3 Cloud的App服务层，重用操作插件工程；
* 创建插件类与创建动态表单插件类似，继承基类AbstractBusinessFlowServicePlugIn
* 插件文件引用基类所在命名空间

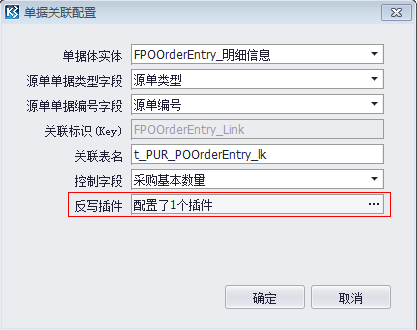
|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.BusinessFlow.PlugIn;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BusinessFlowPlugIn  {  public class Class2 : AbstractBusinessFlowServicePlugIn  {  //TODO  }  } | |

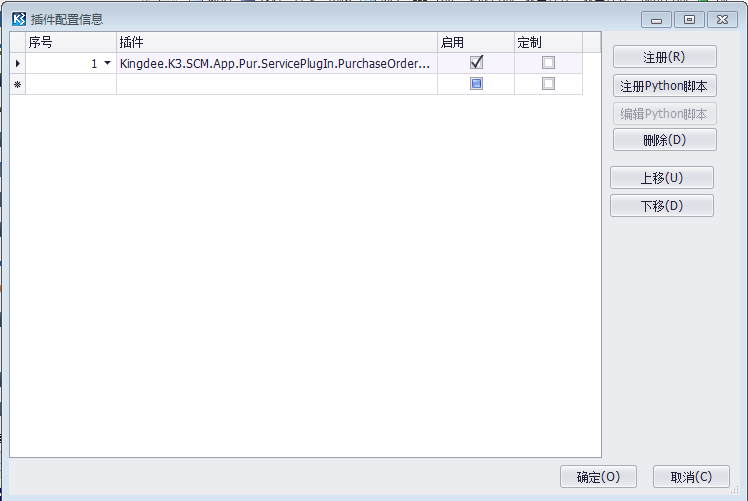
**如何启用反写规则插件**

* 打开BOSIDE，打开业务流程的下游单据模型（如采购申请单下推采购订单，则反写插件挂在采购订单上），选择单据关联配置属性



* 反写插件，注册编译生成的反写插件工程组件对应的反写类，并启用





## 为何插件不起作用，如何排查

插件不起作用，按照如下步骤检查：

* 插件工程编译后，是否输出到Cloud网站路径website\bin
* 插件类是否注册到了当前表单上，并成功保存？注册插件时，要找website\bin路径下的插件工程组件
* 编译生成dll到指定网站目录后，是否重启过站点的IIS？必须重启站点IIS，新组件才会被启用；
* 调试代码，附加w3wp.exe程序后，插件代码是否能够命中？如果不能命中，说明运行中的组件，与代码不匹配，需要重新编译并重启IIS；

# 进阶篇

## 插件代码，如何自行调用单据操作服务

### 如何加载(Load)单据

#### 运行在Web层的插件，如何加载单据？

* 运行在Web层的插件，包括动态表单插件、单据编辑插件、基础资料编辑插件、列表插件等，控制着交互界面；
* 实现思路
  + 调用Kingdee.BOS.ServiceHelper.dll的BusinessDataServiceHelper类提供的Load方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll组件
  + 类文件引用BusinessDataServiceHelper类所在的命名空间Kingdee.BOS.ServiceHelper
* BusinessDataServiceHelper.Load方法介绍

BusinessDataServiceHelper提供了4个Load方法，分别介绍如下

* + public static DynamicObject[] Load(Context ctx, BusinessInfo businessInfo, List<SelectorItemInfo> selector, OQLFilter ofilter).

构建过滤条件OQLFilter，检索某些字段片段信息（List<SelectorItemInfo>）的目标数据集合

* + public static DynamicObject[] Load(Context ctx, DynamicObjectType type, QueryBuilderParemeter queryParemeter)

构建过滤条件QueryBuilderParemeter，检索满足条件的目标数据集合

* + public static DynamicObject[] Load(Context ctx, Object[] pkArray, DynamicObjectType type)

指定目标数据id集合

根据数据id集合，动态对象类型，检索满足条件的目标数据集合

* + public static DynamicObject[] Load(Context ctx, string formid, List<SelectorItemInfo> selector, OQLFilter ofilter)

指定单据模型唯一标示，感兴趣的字段片段信息（也就是说，只取部分字段出来，不会把这个模型的所有字段数据都取出来），构建过滤条件OQLFilter

* 方法参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统上下文信息 |
| businessInfo | BusinessInfo | 待检索数据的模型元数据信息 |
| selector | List<SelectorItemInfo> | 字段片段信息 |
| ofilter | OQLFilter | 快速过滤条件对象 |
| type | DynamicObjectType | 动态实体类型 |
| queryParemeter | QueryBuilderParemeter | 查询过滤对象 |
| pkArray | Object[] | 目标实体数据id集合 |
| formid | string | 待查询模型唯一标示 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class7 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //加载id为10001的物料数据  //获取物料元数据  FormMetadata materialMetada =  MetaDataServiceHelper.Load(this.Context, "BD\_Material") as FormMetadata;  //构建过滤条件  QueryBuilderParemeter queryParameter = new QueryBuilderParemeter();  queryParameter.BusinessInfo = materialMetada.BusinessInfo;  queryParameter.FilterClauseWihtKey = "FMaterailId = 10001";  //构建快捷过滤条件  OQLFilter filter = new OQLFilter();  filter.Add(new OQLFilterHeadEntityItem() { FilterString = "FMaterialId = 10001" });  //构建关心的字段片段信息  List<SelectorItemInfo> lstSelectorItemInfos = new List<SelectorItemInfo>();  //物料属性  lstSelectorItemInfos.Add(new SelectorItemInfo("FErpClsId"));  //使用组织  lstSelectorItemInfos.Add(new SelectorItemInfo("FUseOrgId"));  //方法1：  DynamicObject[] objs = BusinessDataServiceHelper.Load(  this.Context,  materialMetada.BusinessInfo.GetDynamicObjectType(),  queryParameter);  //方法2：  objs = BusinessDataServiceHelper.Load(  this.Context,  new object[] { 10001 },  materialMetada.BusinessInfo.GetDynamicObjectType());  //方法3：根据过滤条件获取满足条件的目标数据的部分字段信息  objs = BusinessDataServiceHelper.Load(  this.Context,  materialMetada.BusinessInfo,  lstSelectorItemInfos,  filter);  //方法4：根据过滤条件获取满足条件的目标数据的部分字段信息  objs = BusinessDataServiceHelper.Load(  this.Context,  "BD\_Material",  lstSelectorItemInfos,  filter);  }  }  } | |

#### 运行在App服务层的插件，如何加载单据？

* 运行在App层的插件，包括操作插件、单据转换插件、反写插件、账表取数插件等，此类插件掌控着某种特定服务的执行过程，与交互界面无关；
* 实现思路
  + 通过Kingdee.BOS.App.dll的ServiceHelper类获取ViewService服务，调用此服务提供的Load方法
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* IViewService接口介绍
* 同web层插件方法介绍
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Orm.DataEntity;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class6 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(BeforeExecuteOperationTransaction e)  {  //加载id为10001的物料数据  //获取ViewService  IViewService viewService =  Kingdee.BOS.App.ServiceHelper.GetService<IViewService>();  //获取元数据服务  IMetaDataService metadataService =  Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //获取物料元数据  FormMetadata materialMetada =  metadataService.Load(this.Context, "BD\_Material") as FormMetadata;  //构建过滤条件  QueryBuilderParemeter queryParameter = new QueryBuilderParemeter();  queryParameter.BusinessInfo = materialMetada.BusinessInfo;  queryParameter.FilterClauseWihtKey = "FMaterailId = 10001";  //构建快捷过滤条件  OQLFilter filter = new OQLFilter();  filter.Add(new OQLFilterHeadEntityItem() { FilterString = "FMaterialId = 10001" });  //构建关心的字段片段信息  List<SelectorItemInfo> lstSelectorItemInfos = new List<SelectorItemInfo>();  //物料属性  lstSelectorItemInfos.Add(new SelectorItemInfo("FErpClsId"));  //使用组织  lstSelectorItemInfos.Add(new SelectorItemInfo("FUseOrgId"));  //方法1：  DynamicObject[] objs = viewService.Load(  this.Context,  materialMetada.BusinessInfo.GetDynamicObjectType(),  queryParameter);  //方法2：  objs = viewService.Load(  this.Context,  new object[] { 10001 },  materialMetada.BusinessInfo.GetDynamicObjectType());  //方法3：  objs = viewService.Load(  this.Context,  materialMetada.BusinessInfo,  lstSelectorItemInfos,  filter);  //方法4：  objs = viewService.Load(  this.Context,  "BD\_Material",  lstSelectorItemInfos,  filter);  }  }  } | |

### 如何调用保存(Save)操作

#### Web层插件，如何调用保存(Save)操作

* 实现思路
  + 调用BusinessDataServiceHelper类提供的Save方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll组件
  + 类文件引用BusinessDataServiceHelper类所在的命名空间Kingdee.BOS.ServiceHelper
* BusinessDataServiceHelper.Save方法介绍

BusinessDataServiceHelper提供4个Save方法，看上去差不多，其实本质有区别，可以将这4个方法区分为2类方法

* 直接调用Orm把数据保存到数据库
  + public static DynamicObject Save(Context ctx, DynamicObject dataObject)
  + public static DynamicObject[] Save(Context ctx, DynamicObject[] dataObject)
* 模拟保存操作服务端完整过程，调用填充主键、生成单据编号，执行校验器，调用服务端服务与服务端插件，最后调用Orm把数据保存到数据库
  + public static DynamicObject[] Save(Context ctx, BusinessInfo businessInfo, DynamicObject[] dataObject, string operationNumber)
  + public static IOperationResult Save(Context ctx, BusinessInfo businessInfo, DynamicObject dataObject, OperateOption option = null, string operationNumber = "")
* Save方法参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统全局信息，这里主要使用全局信息中的数据库连接信息 |
| dataObject | DynamicObject | 待保存的数据对象 |
| businessInfo | BusinessInfo | 待保存数据对象的元数据信息 |
| option | OperateOption | 自定义额外参数对象，服务端服务和服务端插件都能持有此对象 |
| operationNumber | string | 当前操作的操作码 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class2 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //直接调用Orm保存，把数据保存到数据库  BusinessDataServiceHelper.Save(this.Context, this.View.Model.DataObject);  //执行保存服务端服务和保存操作的服务端插件  BusinessDataServiceHelper.Save(this.Context, this.View.BusinessInfo, this.View.Model.DataObject);  }  }  } | |

#### App层插件，如何调用保存(Save)操作

* 实现思路
  + 通过Kingdee.BOS.App.dll组件提供的ServiceHelper类，调用GetService泛型方法，获取SaveService服务，调用SaveService提供的Save方法
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* ISaveService.Save接口介绍

同Web层插件介绍，分为两类方法

详见Web层插件、BusinessDataServiceHelper.Save介绍

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个操作上")]  public class Class1 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //例如：保存id为10001的物料  //获取元数据服务  IMetaDataService metadataService = Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //获取保存服务  ISaveService saveService = Kingdee.BOS.App.ServiceHelper.GetService<ISaveService>();  //获取加载数据服务  IViewService viewService = Kingdee.BOS.App.ServiceHelper.GetService<IViewService>();  //获取物料元数据  FormMetadata materialMetadata = metadataService.Load(this.Context,"BD\_Material") as FormMetadata;  //获取id为10001的物料数据  DynamicObject[] objs = viewService.Load(  this.Context,  new object[] { 10001 },  materialMetadata.BusinessInfo.GetDynamicObjectType());  //保存id为10001的物料  saveService.Save(this.Context, objs);  saveService.Save(this.Context, materialMetadata.BusinessInfo, objs);  }  }  } | |

### 如何调用提交(Submit)操作

#### Web层插件，如何调用提交(Submit)操作

* 实现思路
  + 调用Kingdee.BOS.ServiceHelper.dll组件的BusinessDataServiceHelper类提供的Submit方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll
  + 插件类引用BusinessDataServiceHelper类所在的命名空间
* BusinessDataServiceHelper.Submit方法介绍

BusinessDataServiceHelper类提供了1个Submit方法，调用此方法模拟提交操作的服务端服务全过程，即修改单据状态、执行提交服务端服务，执行提交服务端插件，最后执行Orm保存数据到数据库

* Submit方法参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统上下文全局信息，此处主要使用数据库链接信息 |
| businessInfo | BusinessInfo | 待执行提交服务的数据元数据描述信息 |
| Ids | object[] | 待执行提交服务的数据主键值数组 |
| operationNumber | string | 提交操作编码 |
| option | OperateOption | 业务控制额外参数集合，提交服务端服务和服务端插件都能持有此参数 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Base.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class3 : AbstractBasePlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //例如：提交当前单据数据  BusinessDataServiceHelper.Submit(  this.Context,  this.View.BusinessInfo,  new object[] { this.View.Model.DataObject["Id"] },  "Submit");  }  }  } | |

#### App层插件，如何调用提交(Submit)操作

* 实现思路
  + 调用Kingdee.BOS.App.dll组件提供的ServiceHelper类所提供的GetService泛型方法，获取提交服务，调用提交服务的Submit方法
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* ISubmit.Submit接口介绍
  + 同web层插件介绍
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class2 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //例如：提交id为10001的物料数据  //获取提交服务  ISubmitService submitService = Kingdee.BOS.App.ServiceHelper.GetService<ISubmitService>();  //获取元数据服务  IMetaDataService metadataService = Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //获取物料元数据  FormMetadata materialMetadata = metadataService.Load(this.Context, "BD\_Material") as FormMetadata;  //执行提交服务  submitService.Submit(  this.Context,  materialMetadata.BusinessInfo,  new object[] { 10001 },  "Submit");  }  }  } | |

### 如何调用审核/反审核(Audit/UnAudit)操作

#### Web层插件，如何调用审核/反审核(Audit/UnAudit)操作

* 实现思路
  + 调用Kingdee.BOS.ServiceHelper.dll组件提供的BusinessDataServiceHelper类提供的SetBillStatus方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll组件
  + 插件类文件引用BusinessDataServiceHelper所在的命名空间
* BusinessDataServiceHelper.SetBillStatus方法介绍

SetBillStatus方法与SetStatus方法看上去类似，其实完全不一样，其参数列表也完全不一样，调用单据审核操作的服务端要使用SetBillStatus方法，SetStatus方法后续介绍

* SetBillStatus方法参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统上下文全局信息 |
| businessInfo | BusinessInfo | 待审核\反审核数据的元数据描述信息 |
| pkEntryIds | List<KeyValuePair<object, object>> | 待审核\反审核数据的id集合，此场景key和value都是id的值 |
| paras | List<object> | 审核\反审核时附加的参数，包含两个值，第一个，同意与否，0：驳回；1：通过；2：反审核；第二个，审核意见，此参数默认可以为空 |
| operationNumber | string | 审核\反审核操作的操作码 |
| option | OperateOption | 额外业务选项，审核\反审核服务端服务和服务端插件都持有此对象 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("区分当前类应用在哪个模型上")]  public class Class4 : AbstractListPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  List<KeyValuePair<object,object>> lstKeyValuePairs = new List<KeyValuePair<object,object>>();  KeyValuePair<object, object> keyValuePair = new KeyValuePair<object, object>(10001, "");  lstKeyValuePairs.Add(keyValuePair);  //审核id为10001的当前单据  IOperationResult auditResult = BusinessDataServiceHelper.SetBillStatus(  this.Context,  this.View.BusinessInfo,  lstKeyValuePairs,  null,  "Audit");  //反审核id为10001的当前单据  IOperationResult unAuditResult = BusinessDataServiceHelper.SetBillStatus(  this.Context,  this.View.BusinessInfo,  lstKeyValuePairs,  null,  "UnAudit");  }  }  } | |

#### App层插件，如何调用审核/反审核(Audit/UnAudit)操作

* 实现思路
  + 调用Kingdee.BOS.App.dll组件提供的ServiceHelper类提供的GetService泛型方法，获取改变单据状态服务，调用此服务提供的SetBillStatus方法
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* 接口介绍
* 同Web层插件介绍
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class3 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //审核\反审核id为10001的物料数据  //获取单据状态转换服务  ISetStatusService setStatusService = Kingdee.BOS.App.ServiceHelper.GetService<ISetStatusService>();  //获取元数据服务  IMetaDataService metaDataService = Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //构建id为10001的数据集合  List<KeyValuePair<object, object>> lstKeyValuePairs = new List<KeyValuePair<object, object>>();  KeyValuePair<object, object> keyValuePair = new KeyValuePair<object, object>(10001, "");  lstKeyValuePairs.Add(keyValuePair);  //获取物料元数据  FormMetadata materialMetaData = metaDataService.Load(this.Context, "BD\_Material") as FormMetadata;    //审核  IOperationResult auditResult = setStatusService.SetBillStatus(  this.Context,  materialMetaData.BusinessInfo,  lstKeyValuePairs,  null,  "Audit");  //反审核  IOperationResult unAuditResult = setStatusService.SetBillStatus(  this.Context,  materialMetaData.BusinessInfo,  lstKeyValuePairs,  null,  "UnAudit");  }  }  } | |

### 如何调用禁用(Disabled)操作

#### Web层插件，如何调用禁用(Disabled)操作

* 实现思路
  + 调用Kingdee.BOS.ServiceHelper.dll组件提供的BusinessDataServiceHelper类所提供的SetState方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll组件
  + 插件类文件引用BusinessDataServiceHelper所在的命名空间
* BusinessDataServiceHelper.SetState方法介绍
* SetState方法直接操作数据库，执行update语句，按方法参数构建一条例如update xxx set xxx =’ xxx’ where xxx = xxx的指令
* SetState参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统上下文全局信息 |
| tableName | string | 待执行update指令的表名 |
| fieldName | string | 待禁用的状态字段的字段名 |
| fieldValue | string | 待禁用的状态字段的目标值 |
| pkFieldName | string | 当前表的唯一标识字段名 |
| pkFieldValues | object[] | 当前表的唯一标示字段值集合 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class5 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //禁用物料id为10001和10002的数据  BusinessDataServiceHelper.SetState(  this.Context,  "T\_BD\_Material",  "FForbidStatus",  "B",  "FMaterialdId",  new object[] { 10001, 10002 });  }  }  } | |

#### App层插件，如何调用禁用(Disabled)操作

* 实现思路
  + 调用Kingdee.BOS.App.dll组件提供的ServiceHelper类提供的GetService泛型方法，访问BusinessDataService服务的SetState方法
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* 接口介绍
  + 同Web层插件介绍
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class4 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //获取数据服务  IBusinessDataService businessDataService =  Kingdee.BOS.App.ServiceHelper.GetService<IBusinessDataService>();  businessDataService.SetState(  this.Context,  "T\_BD\_Material",  "FForbidStatus",  "B",  "FMaterialId",  new object[] { 10001, 10002 });  }  }  } | |

### 如何删除(Delete)单据

#### Web层插件，如何删除(Delete)单据

* 实现思路
  + 调用Kingdee.BOS.ServiceHelper.dll组件的BusinessDataServiceHelper类提供的Delete方法
* 插件准备
  + 插件工程引用Kingdee.BOS.ServiceHelper.dll组件
  + 插件类文件引用BusinessDataServiceHelper所在的命名空间
* 方法介绍

BusinessDataServiceHelper提供两个Delete方法，其区别在于

* 参数不带元数据描述信息的，直接执行数据库删除操作，把数据删除
  + public static void Delete(Context ctx, object[] Ids, DynamicObjectType dynamicObjectType)
* 参数带元数据描述信息的，执行删除操作的服务端完整服务，如执行删除操作的校验器、删除操作的服务端服务、删除操作的服务端插件
  + public static IOperationResult Delete(Context ctx, BusinessInfo businessInfo, object[] Ids, OperateOption option = null, string operationNumber = "")
* 参数介绍

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ctx | Context | 系统上下文全局信息 |
| Ids | object[] | 待删除数据的id集合 |
| dynamicObjectType | DynamicObjectType | 待删除数据的动态实体类型 |
| businessInfo | BusinessInfo | 待删除数据的元数据描述信息 |
| option | OperateOption | 业务控制额外选项，删除的服务端服务和插件都能持有此对象 |
| operationNumber | string | 删除操作的操作码 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class6 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //删除id为10001和10002的物料  //获取物料元数据  FormMetadata materialMetadata =  MetaDataServiceHelper.Load(this.Context,"BD\_Material") as FormMetadata;  // 模拟删除服务端操作完整过程  BusinessDataServiceHelper.Delete(  this.Context,  materialMetadata.BusinessInfo,  new object[] { 10001, 10002 });  }  }  } | |

#### App层插件，如何删除单据

* 实现思路
  + 调用Kingdee.BOS.App.dll组件的ServiceHelper类提供的GetService泛型方法获取删除服务
* 插件准备
  + 插件工程引用Kingdee.BOS.App.dll、Kingdee.BOS.Contracts.dll、Kingdee.BOS.DataEntity.dll
  + 类文件引用相关命名空间，详见示例代码
* 接口介绍
  + 同Web层插件介绍
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class5 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //获取删除服务  IDeleteService deleteService =  Kingdee.BOS.App.ServiceHelper.GetService<IDeleteService>();  //获取元数据服务  IMetaDataService metaDataService =  Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //获取物料元数据  FormMetadata materialMetadata =  metaDataService.Load(this.Context,"BD\_Material") as FormMetadata;  //执行删除服务的完整过程（推荐方式）  deleteService.Delete(  this.Context,  materialMetadata.BusinessInfo,  new object[] { 10001, 10002 });  // 直接执行数据删除 (风险极高，仅适用于未定义单据，只是利用ORM模型存取数据的情况）  // 调用此服务会跳过删除校验，仅删除ORM模型内含表格的数据，其他相关数据不会处理，如果有上下游单据及反写，不会撤销关联及回滚反写，从而造成数据错误！  deleteService.Delete(  this.Context,  new object[] { 10001, 10002 },  materialMetadata.BusinessInfo.GetDynamicObjectType());  }  }  } | |

## 调用单据操作后，如何判断结果，并处理交互提示

* 操作返回结果接口定义：

|  |  |
| --- | --- |
| C# |  |
| public interface IOperationResult : IInteractionResult | |

* IInteractionResult接口定义：

|  |  |
| --- | --- |
| C# |  |
| public interface IInteractionResult  {  // 交互表单显示标题  string FormTitle { get; set; }  // 交互上下文信息  InteractionContext InteractionContext { get; set; }  // 本次交互的发起人  string Sponsor { get; set; }  } | |

* 交互上下文信息定义：

|  |  |
| --- | --- |
| C# |  |
| public class InteractionContext  {  public InteractionContext();  // 当前表单模型信息  public BusinessInfo BusinessInfo { get; }  // 数据上下文  public Context Context { get; set; }  // 交互服务操作的实体集合  public ExtendedDataEntity[] DataEntities { get; set; }  // 自定义交互表单的初始化参数:InteractionFormId属性有值时，本属性才生效  public DynamicFormShowParameter FormShowParameter { get; set; }  // 交互表单标识：默认为BOS\_K3Displayer  public string InteractionFormId { get; set; }  // 是否交互（发现有领域把此当成单纯的自定义信息提示用，所以加此标识来加以区分） 默认是交互类型异常，如作它用，请在此标识  public bool IsInteractive { get; set; }  // K3显示器模型  public K3DisplayerModel K3DisplayerModel { get; set; }  // 其它控制选项  public OperateOption Option { get; set; }  // 简单交互信息(批量自定义消息时，此属性建议赋值，以便平台在批量操作时作简单提示)  public string SimpleMessage { get; set; }  // 默认值是false 是否支持移动审批交互处理，移动端目前仅能支持到是否的交互处理。 如果是简单的是/否交互处理，可以把这个值设置为true  public bool SupportMobile { get; set; }  } | |

* 如何判断是否存在交互提示：
* 操作返回结果如果其交互上下文不为null，说明需要进行交互
* 交互实现基本信息包含在交互信息上下文中

* 如何判断是否存在校验错误或警告信息
* 如果返回结果中的IsSuccess为false，说明执行当前操作失败，分析操作返回结果的ValidationErrors是否不为空，包含校验错误信息
* 示例代码

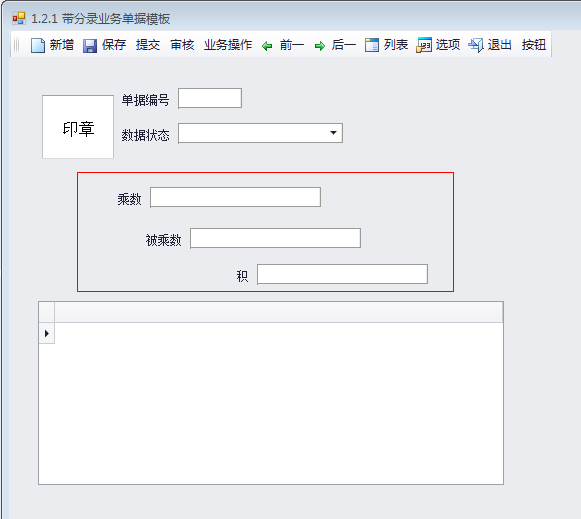
|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Interaction;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class14 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //保存id为10001的其他出库单  IOperationResult result =  new OperationResult();  //获取其他出库单元数据  FormMetadata metadata =  MetaDataServiceHelper.Load(this.Context,"STK\_MisDelivery") as FormMetadata;  //加载id为10001的其他出库单  DynamicObject[] objs =  BusinessDataServiceHelper.Load(  this.Context,  new object[]{10001},  metadata.BusinessInfo.GetDynamicObjectType());  //调用保存操作，用result临时变量持有操作返回结果对象  result =  BusinessDataServiceHelper.Save(this.Context, metadata.BusinessInfo, objs);  //分析操作结果  if (result.InteractionContext != null)  {  if (!string.IsNullOrWhiteSpace(result.InteractionContext.InteractionFormId))  {  this.DoComplextInteraction(result);  return;  }  else if (!string.IsNullOrWhiteSpace(result.InteractionContext.SimpleMessage))  {  this.DoSimpleInteraction(result);  return;  }  }  else  {  //保存不成功，并且要求显示信息  if (!result.IsSuccess && result.IsShowMessage)  {  //循环读取result.ValidationErrors信息，抛出来  }  }  }  private void DoComplextInteraction(IOperationResult result)  {  if (result.InteractionContext != null  && result.InteractionContext.InteractionFormId.Equals("BOS\_K3Displayer"))  {//调用K3显示器处理交互  this.View.ShowK3Displayer(result.InteractionContext.K3DisplayerModel,  (o) =>  {  if (o != null && (o.ReturnData is K3DisplayerModel) && (o.ReturnData as K3DisplayerModel).IsOK)  {  //重新执行保存，设置Option忽略交互信息  //this.Option.SetInteractionFlag(result.InteractionContext.Option.GetInteractionFlag());  }  });  }  else  {//调用业务定制Form进行交互  if (result.InteractionContext.FormShowParameter == null)  {  result.InteractionContext.FormShowParameter = new DynamicFormShowParameter();  result.InteractionContext.FormShowParameter.OpenStyle.ShowType = ShowType.Modal;  }  result.InteractionContext.FormShowParameter.FormId = result.InteractionContext.InteractionFormId;  result.InteractionContext.FormShowParameter.ParentPageId = this.View.PageId;  this.View.ShowForm(result.InteractionContext.FormShowParameter,  (o) =>  {  if (o != null && (o is InteractionFormResult) && (o as InteractionFormResult).IsOK)  {  //重新执行保存，设置Option忽略交互信息  }  });  }  }  private void DoSimpleInteraction(IInteractionResult result)  {  if (result == null) return;  this.View.ShowMessage(result.InteractionContext.SimpleMessage,  MessageBoxOptions.YesNo,  (o) =>  {  InteractionFormResult formResult = new InteractionFormResult(o == MessageBoxResult.Yes, o);  if (o == MessageBoxResult.Yes)  {  //重新执行保存，设置Option忽略交互信息  }  },  result.FormTitle);  }  }  } | |

## 插件给字段赋值后，为什么不执行字段”值更新属性”上配置的表单服务

**背景分析**

设计一张单据，单据头有3个整数字段，例如分别为乘数、被乘数、积，各字段BOSIDE配置关键属性如下：

* 乘数
  + 缺省值：2
  + 勾选及时触发值更新
  + 配置字段值更新，前提条件是乘数和被乘数都不为null，积=乘数\*被乘数
* 被乘数
  + 缺省值：2



设计一个菜单，点击时，插件为乘数SetValue为10，插件代码如下

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("class8")]  public class Class8 : AbstractBillPlugIn  {  public override void BarItemClick(BarItemClickEventArgs e)  {  if (e.BarItemKey == "tbButton")  {  this.View.Model.SetValue("FInteger", 10);  }  }  }  } | |

**现象描述**

新增单据时，由于默认值，乘数和被乘数都是2，这时，点新增的菜单，乘数变成10，但是没有触发在乘数字段上设计的值更新事件，也就是说积的值没有变化；此时，在单据界面上修改乘数的值为3，鼠标从乘数字段移开，积变成6，也就是说，界面修改乘数的值触发了乘数字段的值更新事件

**刨根问底**

* 插件写SetValue为乘数字段赋值，不会触发字段值更新服务，也就是说SetValue与调用字段值更新服务是相互解耦的两个服务，并不是紧耦合的关系，所以插件对目标字段SetValue之后，不会触发此字段上设计的字段值更新
* 在界面上修改乘数字段，由于乘数字段勾选了即时触发值更新，所以前端控件会及时触发updatevalue方法（通过httpwatch可看出），在服务端响应updatevalue的一系列方法中，除了为此字段执行setvalue的操作，同时也调用了此字段上设计的值更新服务

**坚持到底**

插件在SetValue之后还是希望能够调用字段值更新服务，效果就像是在单据页面上编辑此字段，鼠标移开，完成SetValue后，调用InvokeFieldUpdateService方法

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("class8")]  public class Class8 : AbstractBillPlugIn  {  public override void BarItemClick(BarItemClickEventArgs e)  {  if (e.BarItemKey == "tbButton")  {  this.View.Model.SetValue("FInteger", 10);  //调用乘数字段的值更新服务  //由于乘数字段在单据头，这里的行号传0  this.View.InvokeFieldUpdateService("FInteger", 0);  }  }  }  } | |

## 在动态表单插件中，如何修改界面数据

### 如何新增单据体行

* 实现思路
* 调用Model的CreateNewEntryRow方法
* CreateNewEntryRow介绍
* Model提供了5个重载方法，分别介绍
* public void CreateNewEntryRow(DynamicObject parentRow, SubEntryEntity entity, int row)

在父单据体对应的子单据体上新增一行

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| parentRow | DynamicObject | 父单据体行实体数据包 |
| entity | SubEntryEntity | 子单据体实体元数据描述信息 |
| row | int | 新创建的子单据体行，在此父分录对应的子单据体中对应的是哪一行，如果传-1，默认新增在子单据体的最后一行 |

* public void CreateNewEntryRow(Entity entity)

在指定单据体的最后一行，新增一行分录

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| entity | Entity | 指定需要创建一行分录的单据体的元数据描述信息 |

* public void CreateNewEntryRow(Entity entity, int row)

在指定单据体的指定行，新增一行分录

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| entity | Entity | 指定新增行的单据体的元数据描述信息 |
| row | int | 指定新增的单据体行是当前单据体的第几行 |

* public virtual void CreateNewEntryRow(Entity entity, int row, DynamicObject rowDataEntity)

在指定单据体的指定行，新增一行分录

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| entity | Entity | 指定新增行的单据体的元数据描述信息 |
| row | int | 新增到哪一行 |
| rowDataEntity | DynamicObject | 待新增行的单据体实体数据包 |

* public void CreateNewEntryRow(string key)

在指定单据体唯一标示的单据体上，新增一行分录，默认新增到最后一行

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| key | string | 待新增单据体行的单据体唯一标示 |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Orm.DataEntity;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class9 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //entity：单据体唯一标识  //subentity：子单据体唯一标识  //获取单据体元数据描述  Entity entity = this.View.BusinessInfo.GetEntity("entity");  //获取子单据体元数据描述  SubEntryEntity subentity = this.View.BusinessInfo.GetEntity("subentity") as SubEntryEntity;  //为单据体新增一行分录  this.View.Model.CreateNewEntryRow("entity");  this.View.Model.CreateNewEntryRow(entity, -1);  this.View.Model.CreateNewEntryRow(  entity,  -1,  this.View.Model.DataObject["entity"] as DynamicObject);  //为子单据体新增一行  this.View.Model.CreateNewEntryRow(  this.View.Model.DataObject["entity"] as DynamicObject,  subentity,  -1);  }  }  } | |

### 如何删除单据体行

* 实现思路
* 调用model的DeleteEntryRow方法
* DeleteEntryRow介绍
* 传递待删除行的单据体唯一标示和要删除行的行号
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class10 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //entity：单据体唯一标识  //删除单据体第一行分录  this.View.Model.DeleteEntryRow("entity", 1);  }  }  } | |

### 如何给普通字段赋值

普通字段，在单据的数据包中，只是用一个属性(DynamicProperty)存储，赋值比较简单：

* 赋值方法列举
* 调用model的SetValue方法
* 使用字段元数据的动态属性赋值
* 直接为DynamicObject对象属性赋值
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.FieldElement;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("class11")]  public class Class11 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //为数量字段赋值10，数量字段的唯一标示是FQty，其动态绑定属性是Qty  //调用SetValue赋值（推荐）  this.View.Model.SetValue("FQty", 10);  //调用动态实体属性赋值（仅供参考）  Field field = this.View.BusinessInfo.GetField("FQty");  field.DynamicProperty.SetValue(this.View.Model.DataObject, 10);  //使用动态实体数据属性，直接赋值（仅供参考，极不推荐）  this.View.Model.DataObject["Qty"] = 10;  }  }  } | |

* 特别说明
* 不同字段类型，赋值方法、接受的值类型有差别
* 使用字段元数据的动态属性赋值与直接使用数据模型的属性赋值，其效果是一样的，这种赋值方式直接修改模型数据包数据，不触发字段的值改变事件，**极不推荐**；
* 使用this.View.Model.SetValue方法赋值，除了修改数据包数据，还触发一系列插件事件，如beforeupdatevalue，datachanged，触发实体服务规则等，**推荐**使用；

### 如何给基础资料字段赋值

基础资料字段，在单据数据包中，会有两个属性（分别为DynamicProperty, RefIDDynamicProperty）存储，直接操作数据包比较复杂。另外，基础资料值是一个DynamicObject类型，获取基础资料值也有些复杂。

* 实现思路
* 使用基础资料id值，调用Model的SetValue方法
* 使用基础资料完整数据包，调用Model的SetValue方法
* 使用动态属性赋值
* 使用DynamicObject直接访问属性赋值
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class12 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //为基础资料字段赋值  //唯一标示：FBase；绑定动态属性：Base  //获取id为10060的物料完整数据包  DynamicObject[] objs = BusinessDataServiceHelper.Load(  this.Context,  new object[] { 10060 },  (MetaDataServiceHelper.Load(this.Context, "BD\_Material") as FormMetadata).  BusinessInfo.GetDynamicObjectType());  //获取基础资料字段元数据描述信息  BaseDataField baseDataField = this.View.BusinessInfo.GetField("FBase") as BaseDataField;  //方法1：使用物料id10060调用SetValue（推荐）  this.View.Model.SetValue("FBase", 10060);  //方法2：使用物料完整数据包调用SetValue  this.View.Model.SetValue("FBase", objs[0]);  //方法3：使用基础资料字段动态属性赋值  baseDataField.RefIDDynamicProperty.SetValue(this.View.Model.DataObject, 10060);  baseDataField.DynamicProperty.SetValue(this.View.Model.DataObject, objs[0]);  //方法4：使用DynamicObject直接访问属性方式赋值  this.View.Model.DataObject["Base\_Id"] = 10060;  this.View.Model.DataObject["Base"] = objs[0];  }  }  } | |

* 区别简述
* 方法1与方法2，效果一样，都会为基础资料字段赋值，并且触发一系列插件事件，如beforeupdatevalue等，**推荐**采用
* 方法3与方法4，效果基本一样，区别非常小，基本可以认为是一样的，与前述1、2方法的区别是不触发运行时一系列插件事件，此方式复杂且不执行字段值改变事件，**极不推荐**
* 从上述示例代码可看出，基础资料字段赋值比较特殊，除了要为其完整数据包赋值外，还要为其id属性赋值，两种SetValue的方法，BOS运行引擎都会为其完整数据包和id属性分别赋值

### 如何动态设置下拉列表字段的可选项

* 实现思路
* 获取下拉列表控件
* 为下拉列表控制赋值
* 调用Model的SetValue方法为模型赋初始值
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.BOS.Core.Metadata;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class13 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //构建下拉列表控件动态数据  List<EnumItem> lstEnumItems = new List<EnumItem>();  EnumItem item0 = new EnumItem();  item0.Value = "B";  item0.Caption = new LocaleValue("北京");  lstEnumItems.Add(item0);  EnumItem item1 = new EnumItem();  item1.Value = "S";  item1.Caption = new LocaleValue("上海");  lstEnumItems.Add(item1);  EnumItem item2 = new EnumItem();  item2.Value = "G";  item2.Caption = new LocaleValue("广州");  lstEnumItems.Add(item2);  //获取下拉列表控件  ComboFieldEditor comFieldEditor =  this.View.GetControl<ComboFieldEditor>("FComboField");  //为下拉列表控件赋值  comFieldEditor.SetComboItems(lstEnumItems);  this.View.Model.SetValue("FComboField", "S");  }  }  } | |

## 字段标识(Key)、字段名(FieldName)、属性名(PropertyName)的差别

* 字段名 (FieldName)
* 数据库物理字段名，不容易和其他两个属性概念混淆，只有在sql指令的地方才用得上
* 属性名 (PropertyName)
* K/3 Cloud BOS设计的动态表单（或单据、基础资料），在运行时，会自动根据动态表单的元数据模型，建立一个ORM模型，并为每个单据体、字段，在ORM模型中，创建一个属性（基础资料字段等特殊字段，会创建两个属性），并以字段的属性名(PropertyName)命名。如果需要到数据包中，直接读取字段值，需传入字段的属性名(PropertyName)；
* 字段标识 (Key)
* 字段的标识(Key)，在当前模型内唯一，不允许重复，除DynamicObject[“xxx”]这种方式访问属性值时使用字段属性名，其他运行时引擎提供的方法都使用字段标识 (Key)，例如this.View.Model.SetValue(“YYY”,100);YYY就是字段标识 (Key)

## 在动态表单中，如何实现文件下载

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Kingdee.BOS.Util;  using Kingdee.BOS.JSON;  using Kingdee.BOS;  using Kingdee.BOS.Core.Const;  using System.Web;  using System.IO;  using Kingdee.BOS.Core.DynamicForm;  namespace MyDynamicFormPlugIn  {  public class Class15 : AbstractListPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  string pk = "";//某个附件在cloud数据库中存储的唯一标示  OQLFilter filter = OQLFilter.CreateHeadEntityFilter(string.Format("FID={0}", pk));  //获取某附件在Cloud数据库下存储的信息对象  DynamicObject dyn = BusinessDataServiceHelper.Load(  this.ListView.Context,  "BOS\_Attachment",  null,  filter).FirstOrDefault();  if (dyn != null)  {  //文件服务器下载  if (dyn.DynamicObjectType.Properties.ContainsKey("FileId") && dyn["FileId"].IsNullOrEmptyOrWhiteSpace() == false)  {  DownloadFileFromFileServer(dyn);  return;  }  //云盘下载  if (dyn.DynamicObjectType.Properties.ContainsKey("FileStorage")  && dyn["FileStorage"].IsNullOrEmptyOrWhiteSpace() == false  && dyn["FileStorage"].Equals("2"))  {  DownloadFileFromKingdeeYun(dyn);  return;  }  //数据库下载  DownloadFileFromDatabase(dyn);  }  }  private void DownloadFileFromFileServer(DynamicObject dyn)  {  JSONArray array1 = new JSONArray();  JSONObject obj1 = new JSONObject();  string url = GetFileServerUrl();  if (url.IsNullOrEmptyOrWhiteSpace())  {  throw new KDBusinessException("", "该文件需要从文件服务器下载，但是系统没有提供文件服务器配置信息。");  }  if (!url.EndsWith("/"))  {  url += "/";  }  string fileurl = string.Format("{0}download.aspx?fileId={1}&dbId={2}&t={3}", url, dyn["FileId"].ToString(), this.Context.DBId, DateTime.Now.Ticks);  obj1.Put("url", HttpUtility.UrlEncode(fileurl));  obj1.Put("title", "文件下载-");  obj1.Put("desc", "请点击打开附件：");  obj1.Put("urltitle", dyn["AttachmentName"].ToString());  array1.Add(obj1);  this.View.AddAction(JSAction.openUrlWindow, array1);  }  private void DownloadFileFromKingdeeYun(DynamicObject dyn)  {  JSONArray array1 = new JSONArray();  JSONObject obj1 = new JSONObject();  string url = GetFileServerUrl();  if (url.IsNullOrEmptyOrWhiteSpace())  {  throw new KDBusinessException("", "该文件需要从文件服务器下载，但是系统没有提供文件服务器配置信息。");  }  if (!url.EndsWith("/"))  {  url += "/";  }  string fileurl = string.Format("{0}download.aspx?fileId={1}&dbId={2}&t={3}", url, dyn["FileId"].ToString(), this.Context.DBId, DateTime.Now.Ticks);  obj1.Put("url", HttpUtility.UrlEncode(fileurl));  obj1.Put("title", "文件下载-");  obj1.Put("desc","请点击打开附件：");  obj1.Put("urltitle", dyn["AttachmentName"].ToString());  array1.Add(obj1);  this.View.AddAction(JSAction.openUrlWindow, array1);  }  private void DownloadFileFromDatabase(DynamicObject dyn)  {  string path = HttpContext.Current.Server.MapPath(KeyConst.TEMPFILEPATH);  //不存在则创建文件目录  if (!Directory.Exists(path))  {  Directory.CreateDirectory(path);  }  string fileName = Path.Combine(path, dyn["AttachmentName"].ToString());  if (File.Exists(fileName))  {  File.Delete(fileName);  }  using (FileStream fs = new FileStream(fileName, FileMode.CreateNew, FileAccess.Write, FileShare.Read))  {  BinaryWriter bw = new BinaryWriter(fs);  bw.Write((Byte[])dyn["Attachment"]);  bw.Close();  fs.Close();  }  string fileUrl = GetWebAppRootUrl() + "/" + KeyConst.TEMPFILEPATH + "/" + HttpUtility.UrlEncode(dyn["AttachmentName"].ToString());  DynamicFormShowParameter showPara = new DynamicFormShowParameter();  showPara.FormId = "BOS\_FileDownLoad";  showPara.CustomParams.Add("url", fileUrl);  showPara.CustomParams.Add("linktext", dyn["AttachmentName"].ToString());  this.View.ShowForm(showPara);  }  private string GetFileServerUrl()  {  DynamicObject[] objs = BusinessDataServiceHelper.LoadFromCache(this.View.Context, "BOS\_FileServerAdr", null, null);  if (objs != null && objs.Count() > 0)  {  foreach (DynamicObject obj in objs)  {  if (obj["FK3CLOUDHOST"].ToString().EqualsIgnoreCase(HttpContext.Current.Request.Url.Host))  {  string httpType = obj["FSCHEME"] as string;  string serverIp = obj["FFILESERVERHOST"] as string;  string port = obj["FPORT"] as string;  string vName = obj["FVIRDIR"] as string;  string fileServerUrl = string.Format("{3}://{0}{1}/{2}/",  serverIp, string.IsNullOrWhiteSpace(port) ? "" : ":" + port, vName, httpType);  return fileServerUrl;  }  }  }  return string.Empty;  }  private string GetWebAppRootUrl()  {  Uri siteuri;  if (Uri.TryCreate(HttpContext.Current.Request.ApplicationPath, UriKind.Absolute, out siteuri))  {  return siteuri.AbsoluteUri;  }  return "";  }  }  } | |

## 如何支持用户把本地文件上传到服务器

* 实现思路
* BOS设计器，表单界面添加【文件上传】控件，运行时通过点击控件来选择本地文件，将上传到服务器默认的临时目录WebSite\FileUpLoadServices\UploadFiles
* 插件捕获CustomEvents事件，在里面获取到上传到服务器的文件路径信息，然后做后续操作。
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| public override void CustomEvents(CustomEventsArgs e)  {  //事件是对应文件上传控件触发  if (e.Key.ToUpper() == "FFileUpdate".ToUpper())  {  //是文件变化触发的事件  if (e.EventName.ToUpper() == "FILECHANGED")  {  JSONObject jSONObject = KDObjectConverter.DeserializeObject<JSONObject>(e.EventArgs);  if (jSONObject != null)  {  JSONArray jSONArray = new JSONArray(jSONObject["NewValue"].ToString());  if (jSONArray.Count > 0)  {  //获取上传的文件名  string text = (jSONArray[0] as Dictionary<string, object>)["ServerFileName"].ToString();  //文件上传到服务端的临时目录  string directory = "FileUpLoadServices\\UploadFiles";  //文件的完整路径  string fileFullPath = PathUtils.GetPhysicalPath(directory, text);  }  }  }  }  base.CustomEvents(e);  } | |

## 以实例说明如何为动态表单取数、填充

* 背景描述
* 动态表单上设计两个文本字段，用于显示物料的编码和名称
* 取id为10001和10002的物料的编码和名称
* 把这些物料数据填充到此单据体中
* 实现思路
* 调用BusinessDataServiceHelper.Load方法记载id为10001和10002的物料字段片段信息
* 清空当前动态表单中的单据体中的数据
* 循环获取的物料数据
  + 循环中，创建分录行
  + 根据行号，获取新建行的模型
  + 使用DynamicObject直接访问属性方式为每个字段赋值
* 最后调用UpdateView把单据体的数据发送到前端控件
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class17 : AbstractDynamicFormPlugIn  {  public override void AfterBindData(EventArgs e)  {  //获取id为10001和10002的物料相关信息，包含编码、名称  List<SelectorItemInfo> lstSelectFields = new List<SelectorItemInfo>();  lstSelectFields.Add(new SelectorItemInfo("FNumber"));  lstSelectFields.Add(new SelectorItemInfo("FName"));  DynamicObject[] objs = BusinessDataServiceHelper.Load(  this.Context,  "BD\_Material",  lstSelectFields,  OQLFilter.CreateHeadEntityFilter("FMaterialId = 10001 or FMaterialId = 10002"));  if (objs != null && objs.Count() > 0)  {  //清除分录数据包  this.View.Model.DeleteEntryData("FEntity");  //获取单据体实体元数据描述信息  Entity entity = this.View.BusinessInfo.GetEntity("FEntity");  int i = 0;  foreach (DynamicObject item in objs)  {  this.View.Model.CreateNewEntryRow("FEntity");  DynamicObjectCollection collecions =  this.View.Model.GetEntityDataObject(entity);  DynamicObject obj = collecions[i];  obj["Number"] = item["Number"];  obj["Name"] = item["Name"];  i++;  }  this.View.UpdateView("FEntity");  }  }  }  } | |

## 如何为预置账表，添加字段

* 论坛资源
* <http://club.kisdee.com/forum.php?mod=viewthread&tid=707332>
* <http://club.kisdee.com/forum.php?mod=viewthread&tid=699480>

## 如何获取基础资料的属性值

* 实现思路
* 基础资料字段唯一标识为FBase，绑定实体属性为Base
* BOSIDE，设计此基础资料字段时，配置需要获取的引用属性
* 通过基础资料id属性值，加载此基础资料数据包，从数据包中获取目标属性的值
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class18 : AbstractBillPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //物料基础资料字段在单据头  DynamicObject obj =  this.View.Model.GetValue("FBase") as DynamicObject;  //xxx属性在基础资料字段上做过引用，直接在基础资料数据包中获取  string xxxStr = Convert.ToString(obj["xxx"]);  //xxx属性没有引用过，需要到数据库读取  List<SelectorItemInfo> lstSelectorInfos = new List<SelectorItemInfo>();  lstSelectorInfos.Add(new SelectorItemInfo("Fxxx"));  DynamicObject[] objs =  BusinessDataServiceHelper.Load(  this.Context,  "BD\_Material",  lstSelectorInfos,  OQLFilter.CreateHeadEntityFilter(string.Format("FMaterailId = {0}", obj["Id"].ToString())));  if (objs != null && objs.Count() > 0)  {  xxxStr = Convert.ToString(objs[0]["xxx"]);  }  }  }  } | |

## 保存前，如何为新建的单据数据包、单据体行，填写主键

* 实现思路
  + 插件利用ORM模型创建的数据包，保存前，需要主动设置主键
* 调用SequenceReader服务， AutoSetPrimaryKey方法，为数据包设置主键
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.App.Core;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class7 : AbstractOperationServicePlugIn  {  public override void BeforeExecuteOperationTransaction(  BeforeExecuteOperationTransaction e)  {  //新增一条物料  //获取元数据服务  IMetaDataService metadataService =  Kingdee.BOS.App.ServiceHelper.GetService<IMetaDataService>();  //获取物料元数据  FormMetadata materialMetadata =  metadataService.Load(this.Context,"BD\_Material") as FormMetadata;  //利用物料的ORM模型，创建一个空的物料数据包  DynamicObject obj = new DynamicObject(materialMetadata.BusinessInfo.GetDynamicObjectType());  //为物料数据包各字段赋值  //逻辑无关，pass  //此时数据包中主键的值都没被填上  SequenceReader reader = new SequenceReader(this.Context);  //为主键赋值  reader.AutoSetPrimaryKey(  new DynamicObject[] { obj },  materialMetadata.BusinessInfo.GetDynamicObjectType());  }  }  } | |

## 在动态表单插件中，如何显示指定界面

* 实现思路
* View调用ShowForm方法
* DynamicFormShowParameter关键属性
  + FormId（要显示哪个类型的模型，此属性必填）
  + PKey（如果要显示具体某张单据，通过此属性填写目标单据id）
  + CustomParams & CustomComplexParams（个性化参数）
  + OpenStyle（打开风格属性内有停靠对象、打开风格等属性，应用于动态表单在其某个面板上打开模型）
* ShowForm方法可以处理回调函数，即在子页面关闭时，在父页面插件中触发此回调函数，此函数定义如下：

|  |  |
| --- | --- |
| C# |  |
| void ShowForm(DynamicFormShowParameter param, Action<FormResult> action); | |

* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class19 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  //显示id为10001的物料  BillShowParameter showParam = new BillShowParameter();  showParam.FormId = "BD\_Materail";  showParam.PKey = "10001";  showParam.ParentPageId = this.View.PageId;  //当前待显示的物料数据，会嵌入到当前页面的一个面板中显示出来  showParam.OpenStyle.ShowType = ShowType.InContainer;  showParam.OpenStyle.TagetKey = "FPanel";  this.View.ShowForm(showParam);  //传递特殊属性到目标模型  //目标模型可以在其表单插件中的OnInitial事件中获取这些参数  showParam.CustomParams.Add("fkey", "fvalue");  showParam.CustomComplexParams.Add("parentView", this.View);  this.View.ShowForm(showParam);  //显示id为10001的物料，关闭此物料界面时，处理一些回调逻辑  this.View.ShowForm(showParam,  (result) =>  {  object returndata = result.ReturnData;  //对返回的数据做业务处理  }  );  }  }  } | |

## 如何为收款单，往来单位类型，添加新的基础资料

* 实现思路
* 收款单的往来单位类型是多类别基础资料列表字段
* BOSIDE配置此字段的“可选基础资料”属性，添加新的基础资料
* 修改此字段对应的“视图名称”属性所对应的数据库设计，把新增的基础资料添加到此视图设计中
* 示例代码

|  |  |
| --- | --- |
| SQL |  |
| CREATE MATERIALIZED VIEW V\_FIN\_CONTACTTYPE  USING NO INDEX  REFRESH FORCE ON COMMIT  AS  SELECT FSUPPLIERID fitemid  ,'BD\_Supplier' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_BD\_SUPPLIER  UNION ALL  SELECT FCUSTID fitemid  ,'BD\_Customer' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_BD\_CUSTOMER  UNION ALL  SELECT FDEPTID fitemid  ,'BD\_Department' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_BD\_DEPARTMENT  UNION ALL  SELECT FID fitemid  ,'BD\_Empinfo' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_HR\_EMPINFO  UNION ALL  SELECT FID fitemid  ,'FIN\_OTHERS' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_FIN\_OTHERS  UNION ALL  SELECT FORGID fitemid  ,'ORG\_Organizations' fformid  ,FNUMBER fnumber  ,FORGID fmasterid  ,FORGID fcreateorgid  ,FORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_ORG\_ORGANIZATIONS  UNION ALL  SELECT FBANKID fitemid  ,'BD\_BANK' fformid  ,FNUMBER fnumber  ,FMASTERID fmasterid  ,FCREATEORGID fcreateorgid  ,FUSEORGID fuseorgid  ,FDOCUMENTSTATUS fdocumentstatus  ,FFORBIDSTATUS fforbidstatus  ,FCREATORID fcreatorid  ,FMODIFIERID fmodifierid  ,FCREATEDATE fcreatedate  ,FMODIFYDATE fmodifydate  ,rowid RID  FROM T\_BD\_BANK; | |

## 如何创建、调试自定义执行计划

* 创建
* 创建插件工程类，实现IScheduleService接口
* IScheduleService接口位于Kingdee.BOS.Contract.dll组件
* 插件工程编译输出路径是cloud网站路径（website\bin）
* 运行时配置执行计划
* 调试
* 附加进程（K3CloudJobProcess.exe）启动调试
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS;  using Kingdee.BOS.Contracts;  using Kingdee.BOS.Core;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  public class Class8 : IScheduleService  {  public void Run(Context ctx, Schedule schedule)  {  //TODO  }  }  } | |

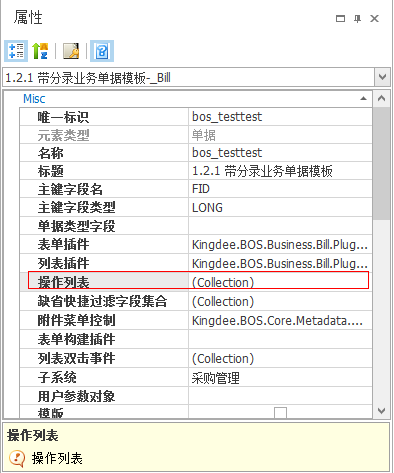
## 如何通过按钮实现F8选择返回效果

* 实现思路
* 捕获按钮Click事件，调用ShowForm，显示列表
* 通过设置ShowForm的回调函数，处理列表返回
* 设计案例
* 例如：设计一个文本控件，勾选其BOSIDE属性“显示编辑按钮”
* 运行时点此文本字段的编辑按钮，触发BeforeF7Select事件
* 在此事件中展示物料F8列表，选中某行记录把物料名称写到当前文本字段中
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.List;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("cls21")]  public class Class21 : AbstractBillPlugIn  {  public override void BeforeF7Select(BeforeF7SelectEventArgs e)  {  if (e.FieldKey.Equals("FText"))  {  //准备物料F8列表打开参数  ListShowParameter lstParam = new ListShowParameter();  lstParam.FormId = "BD\_MATERIAL";  lstParam.ListType = (int)Kingdee.BOS.Core.Enums.BOSEnums.Enu\_ListType.BaseList;  //不允许多选  lstParam.MultiSelect = false;  this.View.ShowForm(lstParam,  (result) =>  {  ListSelectedRowCollection selRows = result.ReturnData as ListSelectedRowCollection;  if (selRows != null && selRows.Count > 0)  {  this.View.Model.SetValue("FText", selRows[0].Name);  }  }  );  }  }  }  } | |

## 如何在操作插件中，自定义操作校验器并启用

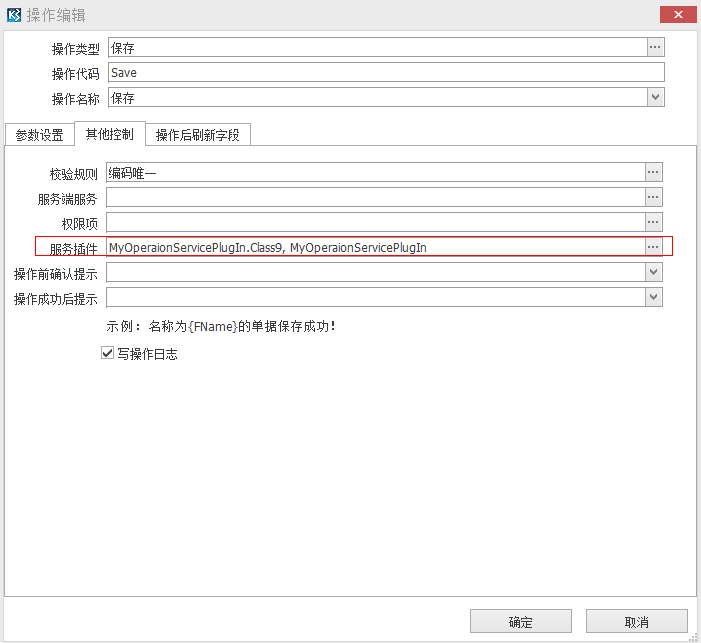
* 实现思路
* 创建自定义校验器类
  + 新增Class
  + 继承自AbstractValidator，类文件引用其命名空间Kingdee.BOS.Core.Validation
  + 此类实现Validate方法、EntityKey属性
* 创建操作插件类
  + 新增Class
  + 继承自AbstractOperationServicePlugIn，类文件引用其命名空间Kingdee.BOS.Core.DynamicForm.PlugIn
  + 此类实现OnAddValidators方法，创建自定义校验器，并初始化其EntityKey属性，添加到e.Validators
* 把操作插件注册到操作中
* 打开BOSIDE模型，操作列表



* 编辑保存操作



* 其他控制页签，配置服务插件



* 示例代码—自定义校验器

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Validation;  using Kingdee.BOS.Orm.DataEntity;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn.MyValidator  {  public class IntegerValidator : AbstractValidator  {  public override void Validate(  ExtendedDataEntity[] dataEntities,  ValidateContext validateContext,  Kingdee.BOS.Context ctx)  {  foreach (ExtendedDataEntity item in dataEntities)  {  DynamicObject curDynamicObject = item.DataEntity;  if (curDynamicObject["FInteger"] != null)  {  long intValue = (long)curDynamicObject["FInteger"];  if (intValue <= 100)  {  validateContext.AddError(curDynamicObject,  new ValidationErrorInfo(  "FInteger",  curDynamicObject["Id"].ToString(),  item.DataEntityIndex,  item.RowIndex,  "E33",  "数量字段必须大于100",  "数量字段"));  }  }  }  }  }  } | |

* 示例代码—操作服务插件

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using MyOperaionServicePlugIn.MyValidator;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyOperaionServicePlugIn  {  [System.ComponentModel.Description("cls9")]  public class Class9 : AbstractOperationServicePlugIn  {  public override void OnAddValidators(AddValidatorsEventArgs e)  {  IntegerValidator validator = new IntegerValidator();  //校验实体对象  //校验器依赖的数据在单据头，所以EntityKey使用单据头  validator.EntityKey = "FBillHead";  e.Validators.Add(validator);  }  }  } | |

## 如何实现超链接，打开外部URL地址

* 实现思路
* 方法一：使用链接控件，配置其url属性
* 方法二：纯用插件实现
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.JSON;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class23 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  JSONObject webobj = new JSONObject();  webobj["source"] = @"http://www.kingdee.com";  webobj["height"] = 545;  webobj["width"] = 810;  webobj["isweb"] = false; //是否新弹出一个浏览器窗口（or选项卡）打开网页地址  webobj["title"] = "金蝶官网";  this.View.AddAction("ShowKDWebbrowseForm", webobj);  this.View.SendDynamicFormAction(this.View);  }  }  } | |

## 如何获取基础资料字段的类别(FormId)

* 实现思路
* 获取基础资料字段元数据信息
* 通过基础资料元数据获取基础资料模型唯一标示
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.FieldElement;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  public class Class24 : AbstractDynamicFormPlugIn  {  public override void ButtonClick(ButtonClickEventArgs e)  {  string fromId = "";  //获取基础资料字段的元数据描述信息  BaseDataField baseDataField =  this.View.BillBusinessInfo.GetField("FBase") as BaseDataField;  //通过基础资料字段元数据获取其查找对象  LookUpObject baseDataFieldLookUpField = baseDataField.LookUpObject;    if (baseDataFieldLookUpField != null)  {  //通过查找对象获取其对应基础资料模型的唯一标示  fromId = baseDataFieldLookUpField.FormId;  }  }  }  } | |

# 高级篇

## 如何给单据体表格，增加自定义过滤行

* 实现思路
* 使用表格控件模型的SetFilterString方法，设置表格过滤条件
* 表格控件过滤条件使用的是字符串比较
* 示例代码

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MyDynamicFormPlugIn  {  [System.ComponentModel.Description("cls16")]  public class Class16 : AbstractBillPlugIn  {  public override void BarItemClick(BarItemClickEventArgs e)  {  if (e.BarItemKey == "tbButton")  {  EntryGrid grid = this.View.GetControl<EntryGrid>("FEntity");  grid.SetFilterString("FInteger > 2");  }  }  }  } | |