培训教材手册

-客户化开发

拉单开发

用友网络科技股份有限公司
2018 年

目录

本文档的业务是从下游单据向上游单据进行拉单，就是点击下游的新增菜单下的引用上游单据 TR14，根据转单规则生成一张下游单据，但是并没有保存到数据库，用户在输入一下业务信息后，保存即可。

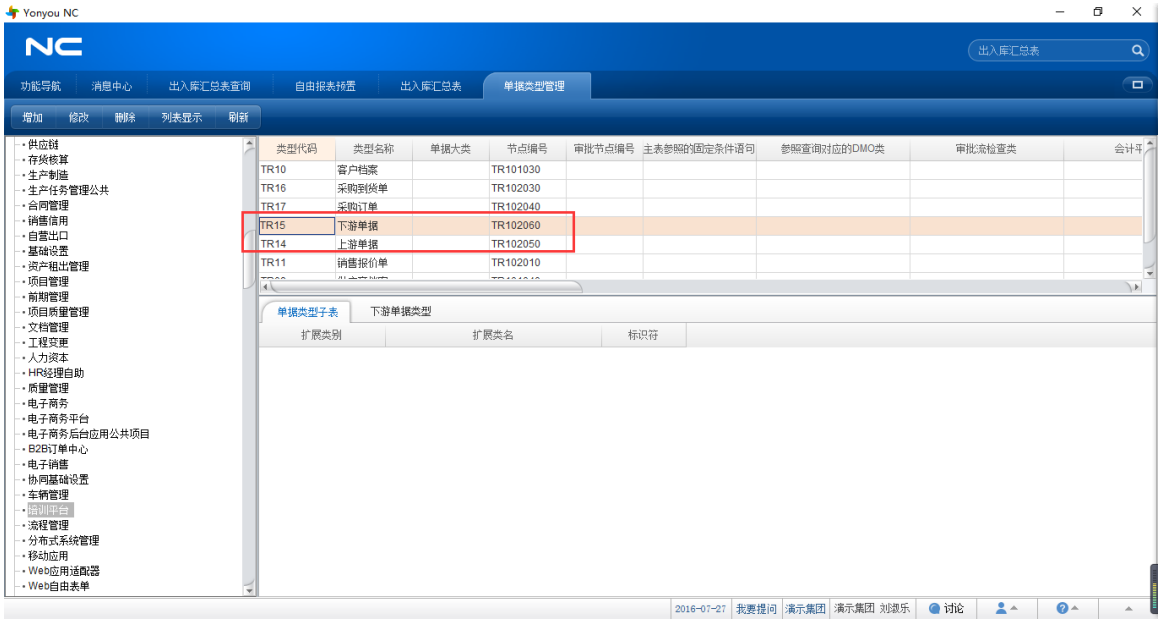
在讲解拉单需要的配置时，用到下面一些单据信息，请牢记，本文档讲解所用到的信息如下：

- *单据：上游单据，下游单据；*
- *单据类型：TR14，TR15，*

一. 准备工作

业务流拉单是从下游单据向上游单据拉单，所以需要两个单据。

本文档用到的单据类型是 TR14，TR15，其中 TR14 是上游单据类型，TR15 是下游单据类型

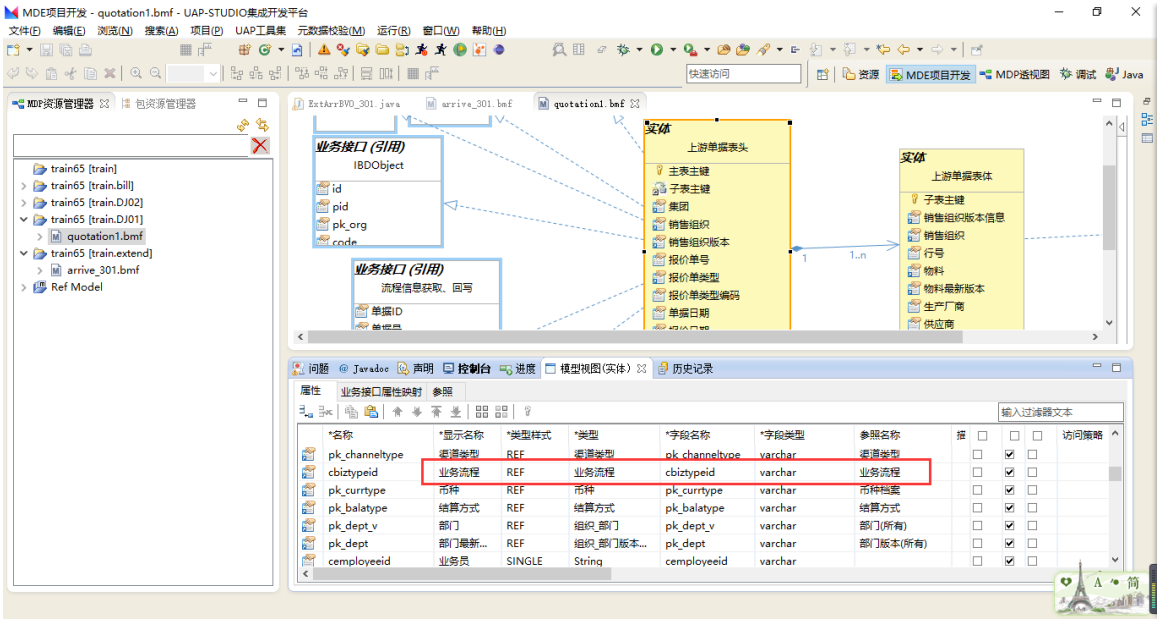


二. 元数据设计

1 上游单据元数据设计

通过 uap mdp 打开元数据设计视图，打开上游单据的元数据设计视图，需要做如下工作：

1. 上游单据需要实现流程 3 大接口
2. 并且上游单据一定要有业务流程字段，参照为【业务流程】并且映射到 IFlowBizItf 接口的【业务类型】，因为是从上游单据向下游单据推单，所以需要在上游单据首选选择好业务流程，所以需要此字段。

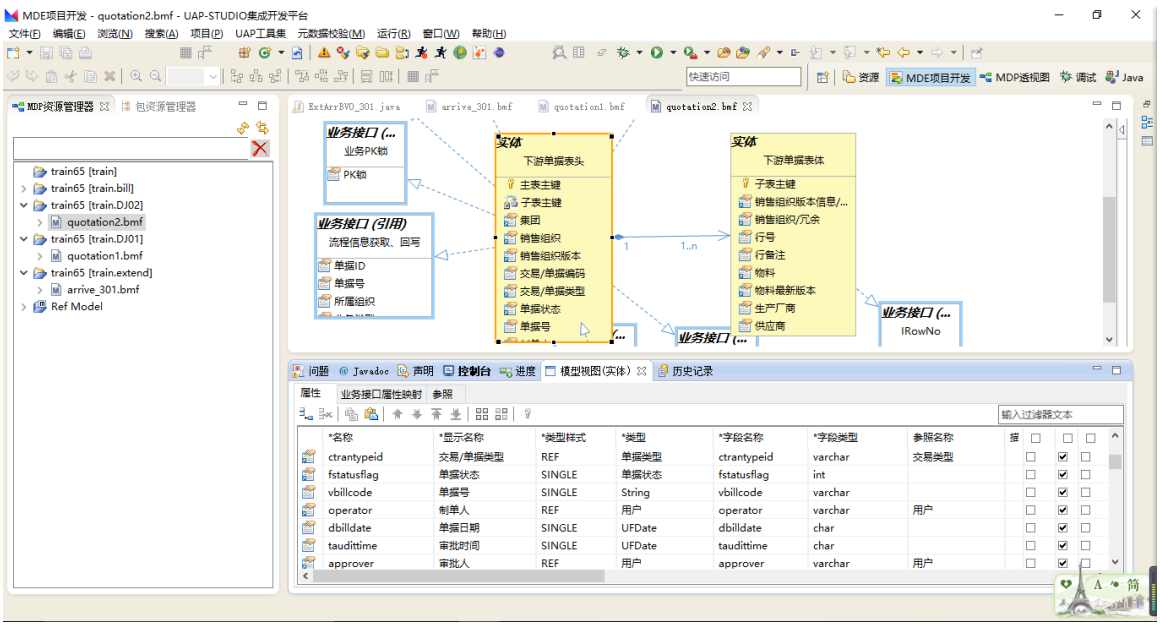


2 下游单据元数据设计

通过 uap mdp 打开元数据设计视图，打开下游单据的元数据设计视图，需要做如下工作：

1. 下游单据需要实现流程 3 大接口

2. 下游单据不一定要有业务流程字段。



三. 定义单据转换规则

通过[动态建模平台]->[流程管理]->[交易类型管理]来打开单据转换规则，定义上下游之间的转单规则，首先添加一条转单规则，来源单据类型选择上游单据类型 TR14，目的单据类型是下游单据类型 TR15，如下图所示，然后在设置规则，如果推单的时候已经设置，那么不用在设置了，使用推单时设计的单据转换规则即可。

通过[动态建模平台]->[流程管理]->[单据转换规则]设置

添加完记录后，点击设置规则来，添加规则。如下图所示，步骤如下

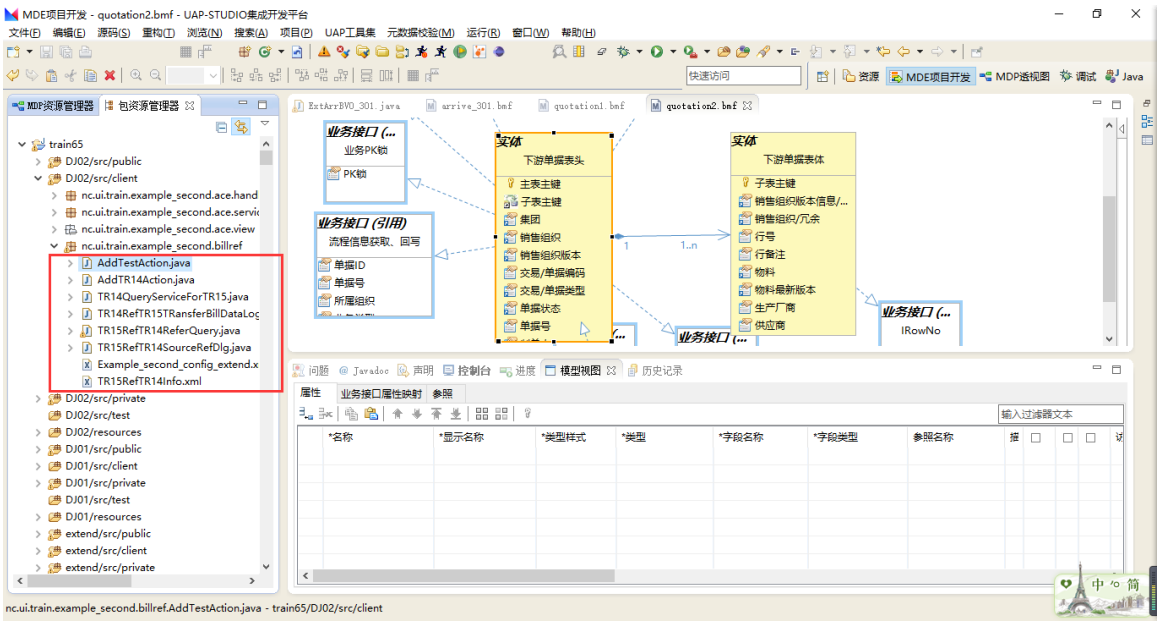
1. 首先添加主表信息，如下图所示，一般必填项是集团，组织，单据状态默认为-1,此外就是你的元数据所设置的哪些必填项了。

2. 添加子表的信息，一般是集团，组织即可

四. 按钮类编写

1 添加上游单据动作类

如下图所示需要创建过 AddTR14Action 类，步骤如下：



1. 创建包 billref，如上图

2. 创建 AddTR14Action，代码如下，***注意红色字体，“TR15”:表示下游单据类型编码，AggSecondBillHVO：表示下游单据元数据，如果给本文档不一样请修改。***

```
3.  package nc.ui.train.example_second.billref;
4.
5.
6.  import java.awt.event.ActionEvent;
7.
8.  import nc.itf.uap.pf.busiflow.PfButtonClickContext;
9.  import nc.ui.pub.pf.PfUtilClient;
10. import nc.ui.pubapp.uif2app.actions.AbstractReferenceAction;
11. import nc.ui.pubapp.uif2app.funcnode.trantype.TrantypeFuncUtils;
12. import nc.ui.pubapp.uif2app.view.BillForm;
13. import nc.ui.uif2.UIState;
14. import nc.ui.uif2.model.AbstractAppModel;
15. import nc.vo.example.entity.AggSecondBillHVO;
16. import nc.vo.jcom.lang.StringUtil;
17.
18. public class AddTR14Action     extends AbstractReferenceAction {
19.
20.         private static final long serialVersionUID = -4417976703049420324L;
21.
```

```
22.     private BillForm editor;
23.
24.     private AbstractAppModel model;
25.
26.     @Override
27.     public void doAction(ActionEvent e) throws Exception {
28.
29.         PfUtilClient.childButtonClickedNew(createPfButtonClickContext());
30.         if (PfUtilClient.isCloseOK()) {
31.             AggSecondBillHVO[] vos = (AggSecondBillHVO[])
    PfUtilClient.getRetVos();
32.             // 显示到转单界面上
33.             this.getTransferViewProcessor().processBillTransfer(vos);
34.         }
35.     }
36.
37.     private PfButtonClickContext createPfButtonClickContext() {
38.         PfButtonClickContext context = new PfButtonClickContext();
39.         context.setParent(this.getModel().getContext().getEntranceUI());
40.         context.setSrcBillType(this.getSourceBillType());
41.         context.setPk_group(this.getModel().getContext().getPk_group());
42.         context.setUserId(this.getModel().getContext().getPk_loginUser());
```

```
43.             // 如果该节点是由交易类型发布的，那么这个参数应该传交易类型，否
    则传单据类型
44.             String vtrantype = TrantypeFuncUtils.getTrantype(this.getModel()
45.                     .getContext());
46.             if (StringUtil.isEmptyWithTrim(vtrantype)) {
47.                 context.setCurrBilltype(" TR15");
48.             } else {
49.                 context.setCurrBilltype(vtrantype);
50.             }
51.             context.setUserObj(null);
52.             context.setSrcBillId(null);
53.             context.setBusiTypes(this.getBusitypes());
54.             // 上面的参数在原来调用的方法中都有涉及，只不过封成了一个整结
    构，下面两个参数是新加的参数
55.             // 上游的交易类型集合
56.             context.setTransTypes(this.getTranstypes());
57.             // 标志在交换根据目的交易类型分组时，查找目的交易类型的依据，有
    三个可设置值：1（根据接口定义）、
58.             // 2（根据流程配置）、-1（不根据交易类型分组）
59.             context.setClassifyMode(PfButtonClickContext.ClassifyByItfdef);
60.             return context;
61.         }
```

```
62.
63.     public BillForm getEditor() {
64.         return this.editor;
65.     }
66.
67.     public AbstractAppModel getModel() {
68.         return this.model;
69.     }
70.
71.     public void setEditor(BillForm editor) {
72.         this.editor = editor;
73.     }
74.
75.     public void setModel(AbstractAppModel model) {
76.         this.model = model;
77.         model.addAppEventListener(this);
78.     }
79.
80.     @Override
81.     protected boolean isActionEnable() {
82.         return this.model.getUiState() == UIState.NOT_EDIT;
83.     }
```

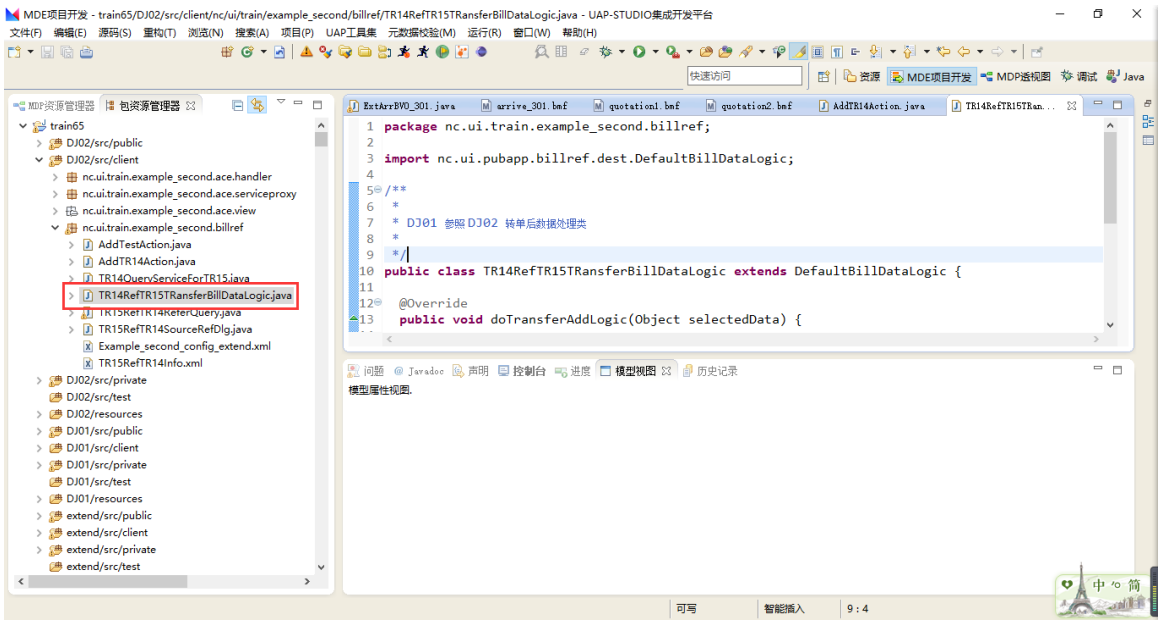
```
84.

85.

86. }
```

2 添加单据逻辑类

如下图所示，添加单据逻辑类 TR14RefTR15TRansferBillDataLogic，步骤如下：



1. 在上一步的 billref 中添加此类，代码源码如下：

```
2. package nc.ui.train.example_second.billref;
3.
4. import nc.ui.pubapp.billref.dest.DefaultBillDataLogic;
5.
6. /**
7.  *
8.  * DJ01 参照 DJ02 转单后数据处理类
9.  *
10. */
11.public class TR14RefTR15TRansferBillDataLogic extends
   DefaultBillDataLogic {
```

```
12.
13.  @Override
14.  public void doTransferAddLogic(Object selectedData) {
15.
16.    // 把数据设置在界面上
17.    super.doTransferAddLogic(selectedData);
18.
19.//    BillCardPanel cardPanel =
   this.getBillForm().getBillCardPanel();
20.//    IKeyValue keyValue = new CardKeyValue(cardPanel);
21.//
22.//    // 1.表头合计
23.//    HeadTotalCalculateRule totalrule = new
   HeadTotalCalculateRule(keyValue);
24.//    totalrule.calculateHeadTotal();
25.  }
26.
27.}
28.
```

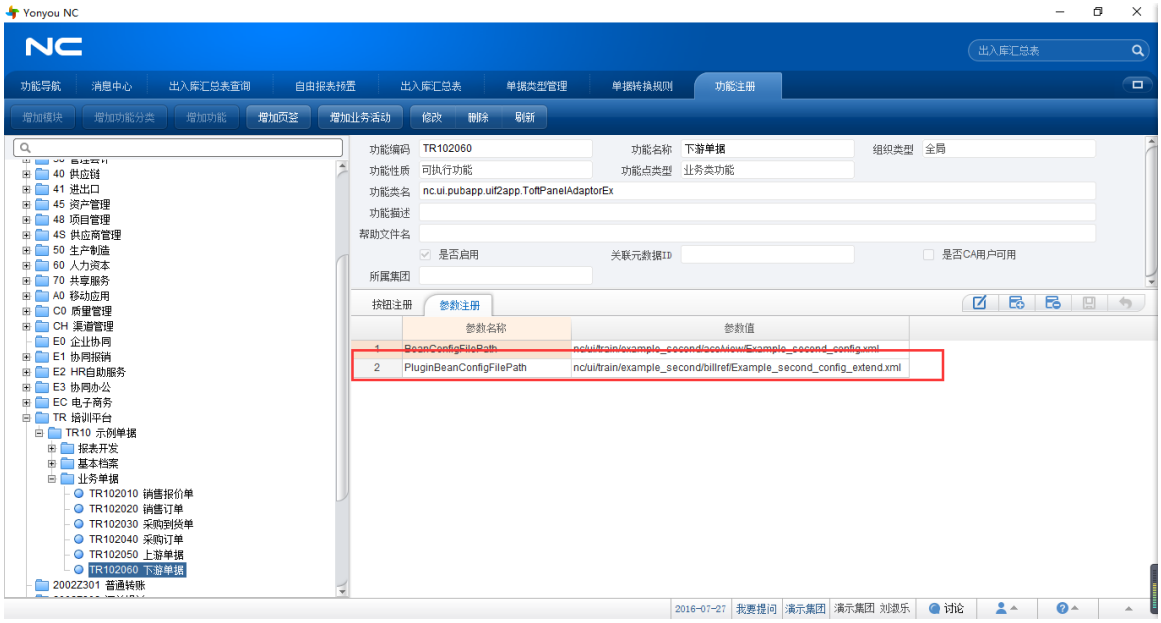
3 修改下游单据 xml 配置文件

下游单据 xml 配置文件如下，注意修改如下：

1. 这里 xml 的配置文件“TR14”表示上游单据类型，“TR15”表示下游单据类型

2. 把原来向导生成的列表按钮配置中的“addAction”后面新增“*addMenuGroup*”

3、下游单据功能注册添加配置文件，如下图所示:




```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <bean id="remoteCallCombinatorCaller"
class="nc.ui.uif2.editor.UIF2RemoteCallCombinatorCaller">
        <property name="remoteCallers">
            <list>
                <ref bean="pfAddInfoLoader" />
            </list>
        </property>
    </bean>

    <!-- 按钮组 -->
    <bean id="addMenuGroup"
class="nc.ui.pubapp.uif2app.actions.AddMenuAction" init-method="initUI">
        <!-- 当前单据的单据类型 -->
        <property name="billType" value="TR15" />
        <!-- 当前单据的所有上游单据的一个全集 -->
        <property name="actions">
            <list>
                <ref bean="addAction" />
                <ref bean="separatorAction" />
                <ref bean="addTR14Action" />
                <ref bean="separatorAction" />
                <ref bean="AddTestAction" />
```

```
            </list>
        </property>
        <property name="model" ref="bmModel" />
        <property name="pfAddInfoLoader" ref="pfAddInfoLoader"></property>
    </bean>

    <!-- 测试 按钮 -->
    <bean id="AddTestAction"
class="nc.ui.train.example_second.billref.AddTestAction">
        <property name="model" ref="bmModel" />
        <property name="editor" ref="billForm" />
        <property name="code" value="addTest" />
    </bean>

    <!-- 参照TR14 -->
    <bean id="addTR14Action"
class="nc.ui.train.example_second.billref.AddTR14Action">
        <!-- 来源单据类型编码 -->
        <property name="sourceBillType" value="TR14" />
        <!-- 来源单据类型名称 -->
        <property name="sourceBillName" value="参照 上游单据" />

        <!-- 是否流程单据，如果是流程单据，可以删除 -->
        <property name="flowBillType" value="false" />
        <property name="model" ref="bmModel" />
        <property name="editor" ref="billForm" />
        <property name="transferViewProcessor" ref="transferProcessorforTR14"
/>
    </bean>

    <bean name="transferProcessorforTR14"
class="nc.ui.pubapp.billref.dest.TransferViewProcessor">
        <property name="list" ref="billListView" />
        <property name="actionContainer" ref="container" />
        <property name="cardActionContainer" ref="actionsOfCard" />
        <property name="transferLogic" ref="transferLogicforTR14" />
        <property name="billForm" ref="billForm" />
        <property name="cancelAction" ref="cancelAction" />
        <property name="saveAction" ref="saveScriptAction" />
    </bean>

    <bean name="transferLogicforTR14"
class="nc.ui.train.example_second.billref.TR14RefTR15TRansferBillDataLogic">
        <property name="billForm" ref="billForm" />
```

```
    </bean>

    <!-- 新增按钮处理 -->
    <bean id="pfAddInfoLoader"
class="nc.ui.pubapp.uif2app.actions.PfAddInfoLoader">
        <property name="billType" value="TR15" />
        <property name="model" ref="bmModel" />
    </bean>
```

五. 拉单结果呈现

点击新增菜单下的“参照 上游单据”，如下图所示出来了

通过拉单生成的下游单据如下图所示：

六. 分单、合单

在推拉单的业务中常常涉及到两个概念分单、合单，下面详细讲解下如何进行分单、合单开发。

分单：如果上游单据为一张单据，但是下游生成的时候需要根据一定的规则进行数据的差分分为多个单据。同样可以根据表体的数据生成不同的下游单据。

合单：在上游单据推式生成下游单据或者下游拉式生成下游单据的时候，下游单据的表体数据有可能来源不同的单据的表体数据，这时就需要根据一定的规则合并为下游单据的一条或者几条数据，如按物料进行合单，也就是相同的物料的合并为一个单据并且数量进行合并。

上面都讲到了一定规则，就是分单规则，并且衍生出来分单依据也就是具体的规则。分单依据有两种方式：a、元数据属性 b、分单函数。

1、元数据属性-分单注册依据步骤

打开【分单注册依据】节点，在左侧选择上游单据，点击修改，在属性页签的右边点击增加选择需要分单的属性添加即可。

在单据转化规则的分单规则页签即可看到分单分单条件，通过勾选 “选择”选项确定是否进行分单。

如果有些业务比较复杂，单纯的根据元数据属性不能分单，则可以通过注册函数进行分单，如下图所示：

添加步骤同元数据

2、分单函数-分单注册依据步骤

如果有些业务比较复杂，单纯的根据元数据属性不能分单，则可以通过注册函数进行分单。

比较重要的信息下面几个属性，举例如

返回类型：ARRAYLIST （每个表体数据按照规则生成的分单规则 key 或者根据表头信息返回 key)

函数类名称：nc.extend.SplitBillRule

函数方法名称：splitBill

函数参数名称：nc.vo.pub.AggregatedValueObject:01

注意：分单的返回类型如果为 ARRAYLIST ，则根据表体行返回每行对应的 key,也就是根据行进行分单，如果返回的是 string 则为表头返回的 key 即按表头进行分单。

在【分单注册依据】添加分单函数，步骤同元数据。

单据转化规则的分单规则页签即可看到分单函数，通过勾选 “选择”选项确定是否进行分单。

综合上述：分单、合单使用同一个分单规则，即按照这些维度分割上游单据的数据。