
U9-UAP 开发大全

(BE、BP、参照、列表及插件开发)

徐中荣

2010 年 5 月 31 日

U9-UAP开发大全	1
第一章 开发基础.....	7
一、 开发工具.....	7
1、 UBF开发工具	7
2、 应用组装工具.....	9
3、 客开部署.....	10
二、 客开引用.....	11
1、 UBF常用引用	12
2、 C#设置引用路径.....	12
三、 系统相关.....	13
四、 数据字典的使用.....	14
第二章 档案开发.....	14
一、 基础概念.....	14
1、 BE组件.....	14
2、 UI组件	17
3、 其他应用.....	22
二、 实体建模.....	27
1、 创建解决方案.....	27
2、 创建BE组件.....	28
3、 构造BE.....	33
三、 创建UI组件	34
1、 建立UI项目	34
2、 建立UIModel.....	34
3、 创建UIForm.....	36
4、 构造UI组件	39
四、 组件发布与装配.....	40
1、 发布服务组.....	40
2、 将组件DLL文件拷贝至U9 相应目录.....	40
3、 通过装配工具进行装配发布。	41
五、 运行使用.....	46
1、 设置菜单权限.....	46
2、 运用.....	47
六、 实现级次编码规则.....	47
1、 BE中应用版型.....	47
2、 应用特性.....	48
3、 BE代码.....	49
4、 UIModel设置.....	51
5、 UIForm设置.....	52
6、 UI代码	52
第三章 单据开发（初级）	57
一、 开发需求.....	57
二、 创建BE.....	57
1、 建立单据类型BE（OrderDocType）	57

2、	建立单据BE.....	58
三、	创建单据类型UI.....	59
1、	UIModel.....	59
2、	UIForm（单卡）.....	60
四、	创建单据UI.....	62
1、	UIModel.....	62
2、	UIForm.....	62
五、	构造、发布和装配.....	64
六、	其他应用.....	64
1、	单据自动编号.....	64
2、	给单据赋默认值.....	65
3、	支持单据个性化功能.....	66
4、	表体自动行号.....	66
5、	实体扩展字段.....	68
6、	单据查找.....	69
第四章	参照开发.....	70
一、	创建单据类型参照组件.....	70
1、	定义订货单类型参照组件.....	70
2、	调用单据类型参照.....	71
二、	Form引用.....	72
1、	订货单单据类型引用.....	72
2、	料品档案.....	73
3、	客户档案.....	74
4、	部门档案.....	74
5、	业务员档案.....	75
三、	条件参照（参照过滤）.....	76
1、	固定条件.....	76
2、	完整OQL条件.....	77
3、	命名参数.....	78
4、	动态条件.....	79
四、	多选参照.....	81
1、	UIModel.....	81
2、	UIForm设置.....	82
3、	定义料品参照.....	82
4、	添加代码.....	82
5、	绑定参照.....	84
五、	自定义参照.....	84
1、	UIModel设置.....	85
2、	UIForm绑定部门视图.....	86
3、	UIForm绑定业务员.....	86
4、	参照添加代码.....	86
5、	注册参照.....	89
六、	料品特殊参照.....	89
第五章	单据列表开发.....	91

一、	创建查询项目	91
二、	重建查询UI	92
三、	定义查询数据源	93
四、	定义查询方案	94
1、	定义查询条件	94
2、	定义查询栏目	94
3、	定义排序字段	96
五、	定义查询UI	96
1、	查看UIModel	96
2、	查看UIForm	97
3、	响应列表事件	97
六、	装配	99
1、	加载页面	99
2、	增加页面	100
3、	定义菜单	101
4、	组装发布	101
七、	调用列表	101
八、	多WebPart查询	102
第六章	单据开发（高级）	102
一、	后台组件编程	102
1、	实体对外结构的基础组成	102
2、	Session	102
3、	实体查询	105
4、	实体资源	107
5、	实体校验	108
6、	事件扩展方法	108
7、	U9 平台异常介绍	109
二、	前台组件编程	109
1、	关系图	109
2、	UI 代码写入时机	111
3、	UIModel 编程	112
4、	UIForm 编程	114
三、	公共API	121
1、	组件端的上下文（后台）	121
2、	UI 端的上下文（前台）	121
3、	获取系统参数	122
四、	UI 高级开发	124
1、	确认框	124
2、	状态栏提示	125
3、	显示错误信息	125
4、	显示图片	125
5、	使能关联	128
6、	UDC 可扩展枚举	128
7、	卡片翻页（换行）处理	129

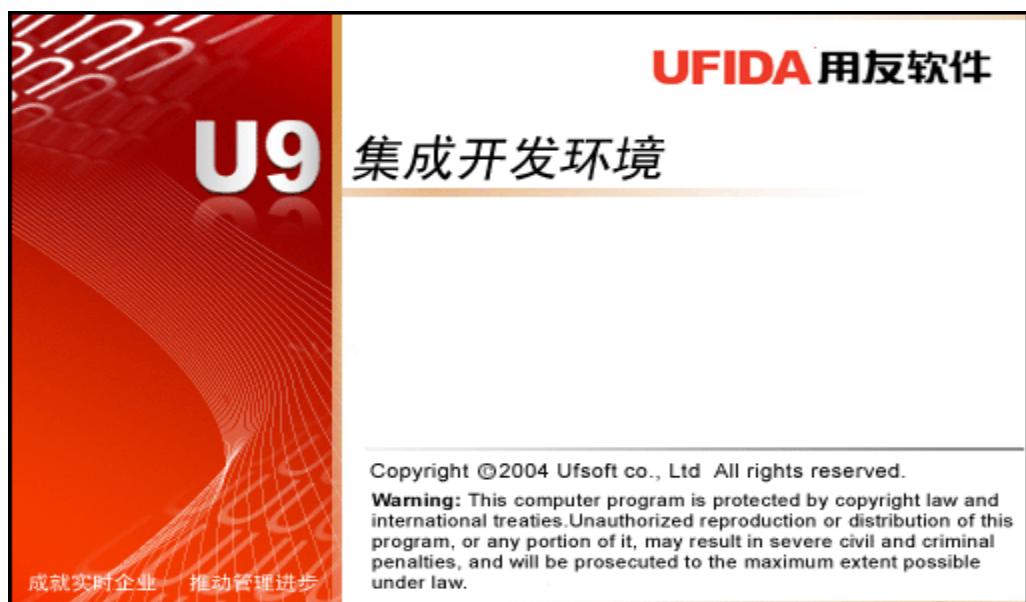
8、	水印效果.....	130
9、	自由项处理（略）	131
10、	数量金额处理.....	132
11、	计价单位.....	136
12、	UI调用BP.....	136
13、	卡片CallBack.....	137
14、	多行参照并自增行.....	142
15、	易用性（表头影响表体）	145
五、	其他应用.....	148
1、	直接运行SQL和存储过程.....	148
2、	弱类型实体.....	149
3、	异常处理.....	151
4、	实体锁.....	152
5、	实体缓存.....	153
6、	数据记录状态.....	154
7、	CallBack函数调用	154
8、	BP数据校验（共享冲突）	155
第七章	单据生单.....	156
一、	推式生单.....	156
1、	建立BP组件.....	156
2、	定义生单UI.....	161
二、	拉式生单.....	167
1、	总体框架.....	167
2、	建立SV组件.....	168
3、	定义生单UI.....	168
第八章	单据插件开发.....	169
一、	插件开发特点.....	169
二、	插件开发流程.....	173
三、	BE插件开发.....	174
1、	代码引用.....	174
2、	配置文件.....	179
3、	DLL部署	180
四、	BP/SV插件开发	180
1、	代码引用.....	180
2、	配置文件.....	181
3、	DLL部署	182
五、	UI插件开发	182
1、	代码引用.....	182
2、	配置文件.....	183
3、	DLL部署	183

第一章 开发基础

一、 开发工具

1、 UBF开发工具

用友集成开发环境: D:\UFIDA\UBF\U9. VOB. Product. UBF\UBFStudio\ UBFdevenv. exe



1) UBF启用选项

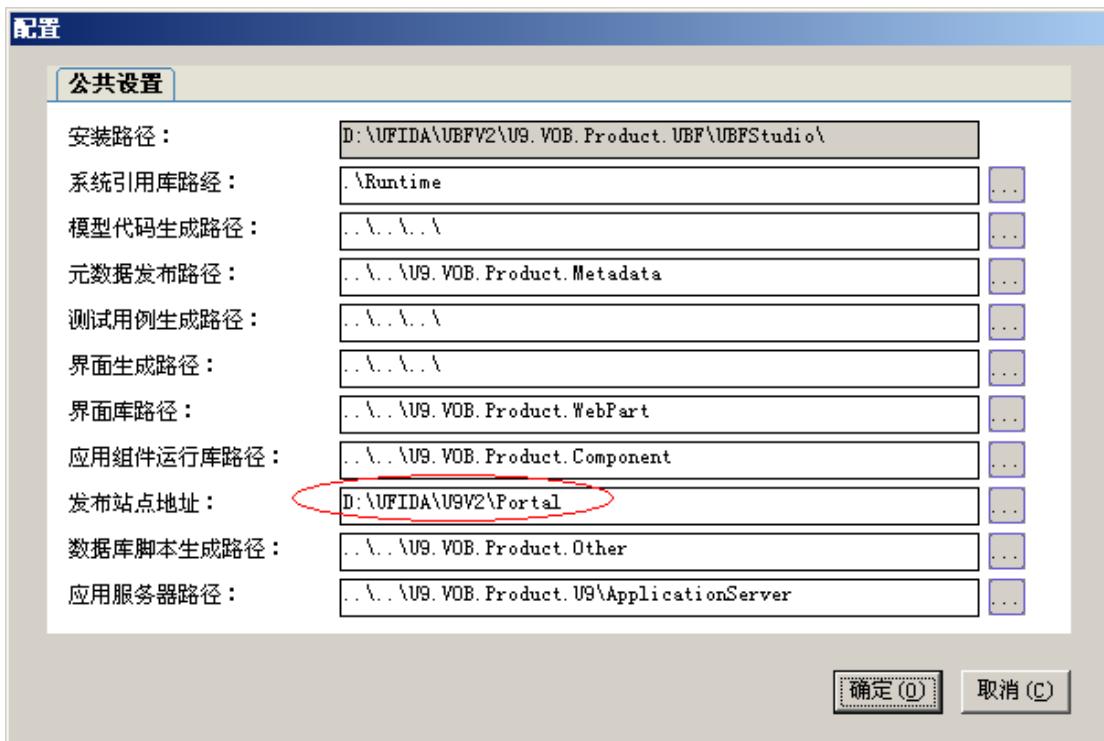
D:\UFIDA\UBF\U9. VOB. Product. UBF\UBFStudio\IDEConfig.xml

(false 为开发, true 为报表及打印元数据调整及开发)

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- 与 UFSofT.UBF.CodeGenerator.TemplateTransConfig.xml 配合使用, 得到最终的生成路径 -->
- <Options RuntimeForReportDesigner="true">
  <!-- 路径配置要求都是相对路径, 路径的基准是 UBFStudio 的路径 -->
  <!-- 属性里设置的是测试用例生成路径, 目前未用。 -->
- <Common TestTargetPath=".\\..\\..">
  <BasePath>..\\..\\</BasePath>
  <!-- 代码生成模型的路径, 目前未用。 -->
  <ModelTemplatePath>..\\</ModelTemplatePath>
```

2) 修改发布站点路径

发布路径: D:\UFIDA\U9V2\Portal, 以便构造的组件可执行批处理拷贝文件。



另一种配置修改:



2、应用组装工具

D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\BuildupDesigner.exe



- D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\BuildupApp.bdcfg 可添加新的应用节点。

1) 数据库连接配置

启用组装工具前，必须配置数据库连接串 connectstring 节点。配置文件：

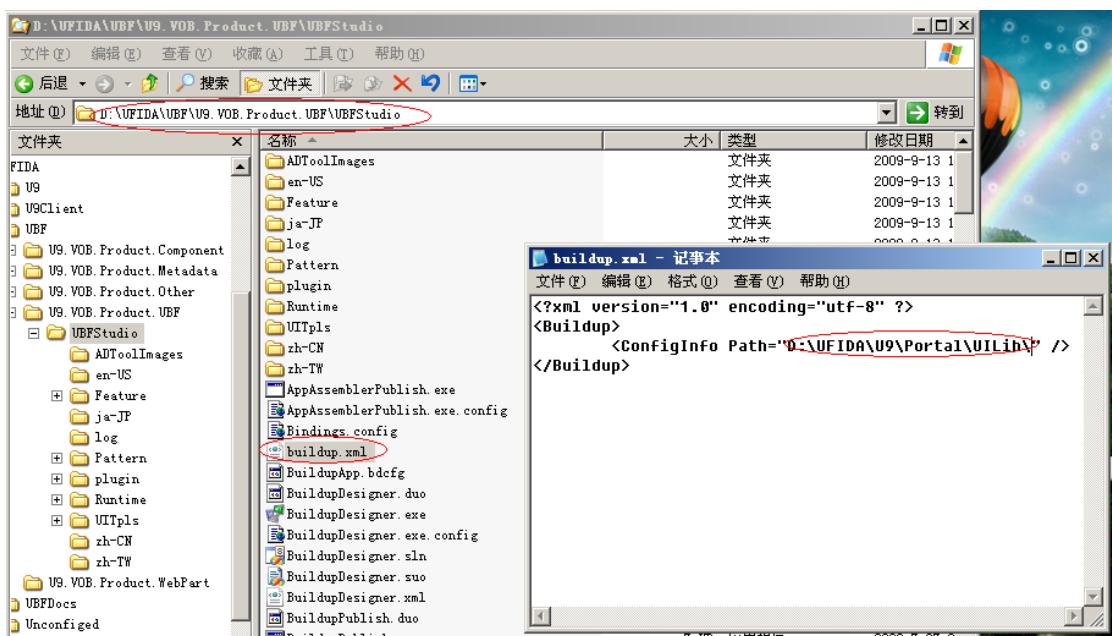
D:\UFIDA\UBF\UBFStudio\Runtime\environment.xml

```
<connectionString>packet size=4096;user id=sa;Connection Timeout=150;Max  
Pool size=1500;data source=pt02;persist security info=True;initial  
catalog=u9v1base;password=sa</connectionString>
```

构造部门给出的解决方案是：将 UBFV2 拷贝一份到其他地方，然后将 UBFV2 名字修改为 U9Product。

2) 修改组装工具配置参数

将 buildup.xml 中的 Path 路径修改为：D:\UFIDA\U9V2\Portal\UILib\



3、客开部署

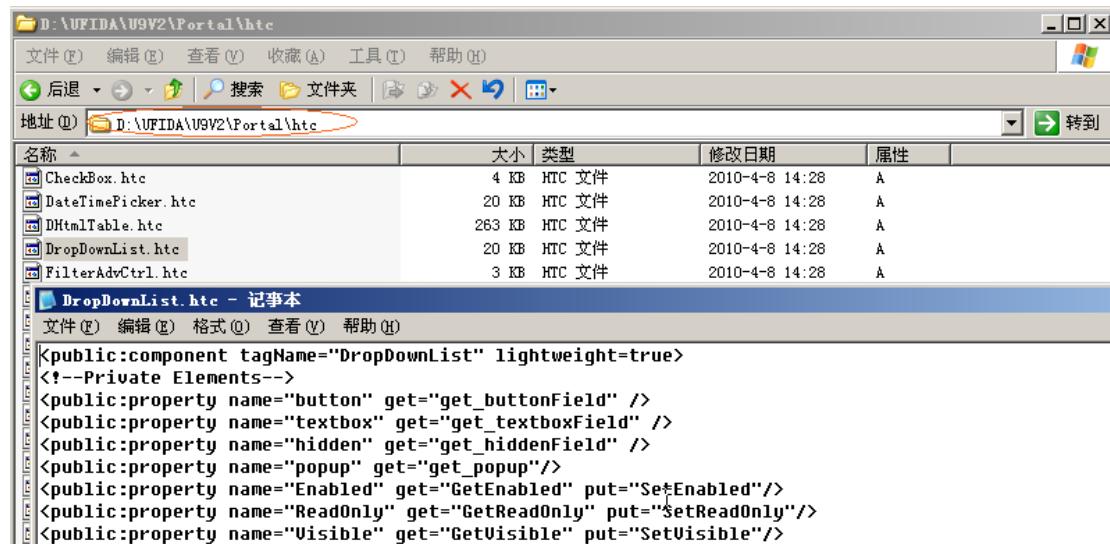
1) 部署文件位置

- 脚本：直接执行（最好有事务保护）
- Deploy 文件：拷贝到 Portal\ApplicationLib 和 Portal\ApplicationServer\Libs
- Agent 文件：拷贝到 Portal\ApplicationLib
- BE、BP 文件：拷贝到 Portal\ApplicationServer\Libs
- UI 文件：拷贝到 Portal\UILib
- DataCommand 文件：拷贝到报表服务器 Bin 目录

2) 元数据存放位置

D:\UFIDA\UBFV2\U9.VOB.Product.Metadata

3) 控件脚本文件



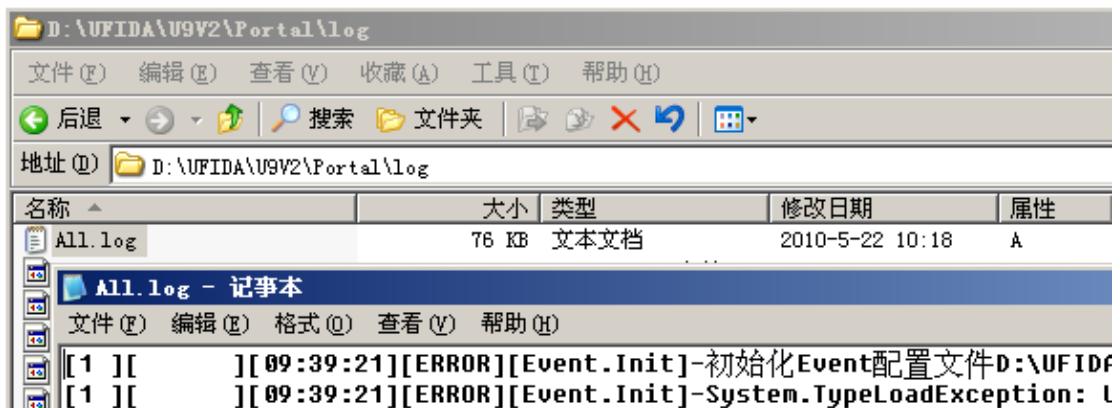
The screenshot shows a Windows File Explorer window with the path D:\UFIDA\U9V2\Portal\htc. The address bar is highlighted with a red oval. Below the address bar, a list of files is shown in a table:

名称	大小	类型	修改日期	属性
CheckBox.htm	4 KB	HTC 文件	2010-4-8 14:28	A
DateTimePicker.htm	20 KB	HTC 文件	2010-4-8 14:28	A
DHtmlTable.htm	263 KB	HTC 文件	2010-4-8 14:28	A
DropDownList.htm	20 KB	HTC 文件	2010-4-8 14:28	A
FilterAdvCtrl.htm	3 KB	HTC 文件	2010-4-8 14:28	A

The file 'DropDownList.htm' is selected and its contents are displayed in a Notepad window below the file list. The Notepad window title is 'DropDownList.htm - 记事本'. The code in the Notepad window is:

```
<public:component tagName="DropDownList" lightweight=true>
<!--Private Elements-->
<public:property name="button" get="get_buttonField" />
<public:property name="textbox" get="get_textboxField" />
<public:property name="hidden" get="get_hiddenField" />
<public:property name="popup" get="get_popup"/>
<public:property name="Enabled" get="GetEnabled" put="SetEnabled"/>
<public:property name="ReadOnly" get="GetReadOnly" put="SetReadOnly"/>
<public:property name="Visible" get="GetVisible" put="SetVisible"/>
```

4) Log日志文件存放位置



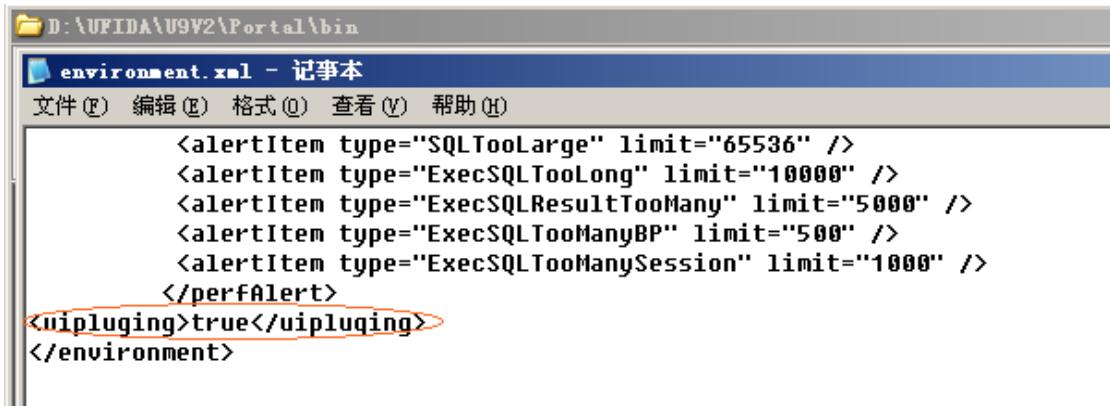
5) UI热插支持

支持 UI 的 DLL 文件重构后，不需重启 IIS 就可直接复制发布。

修改 D:\UFIDA\U9V2\Portal\bin\environment.xml

在</environment>节点前插入以下语句行.

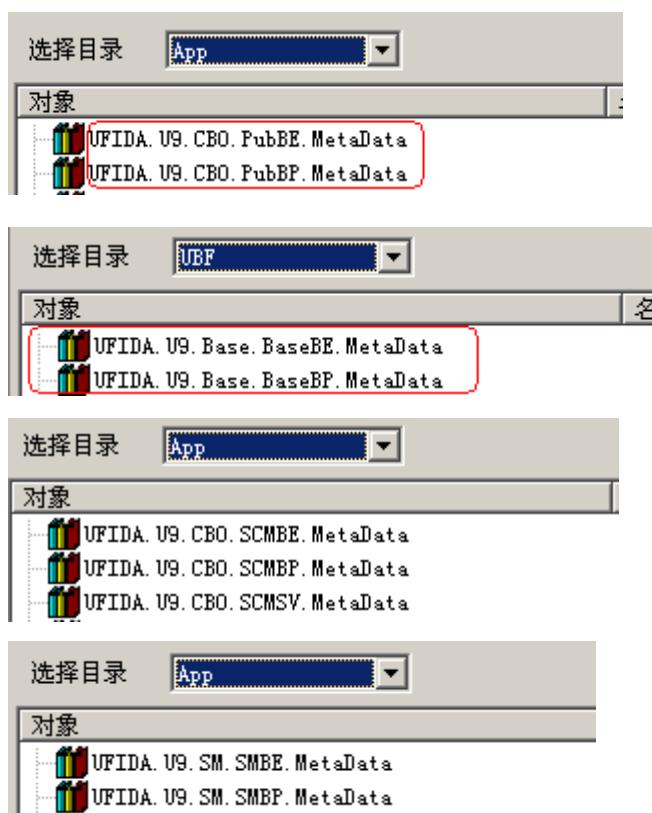
```
<uiplugging>true</uiplugging>
```



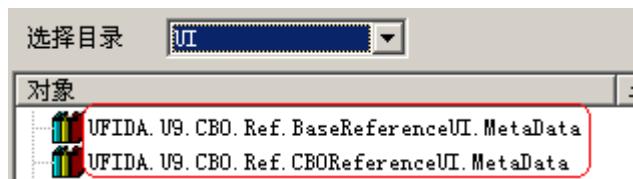
二、客开引用

1、 UBF常用引用

1) BE实体或BP服务引用

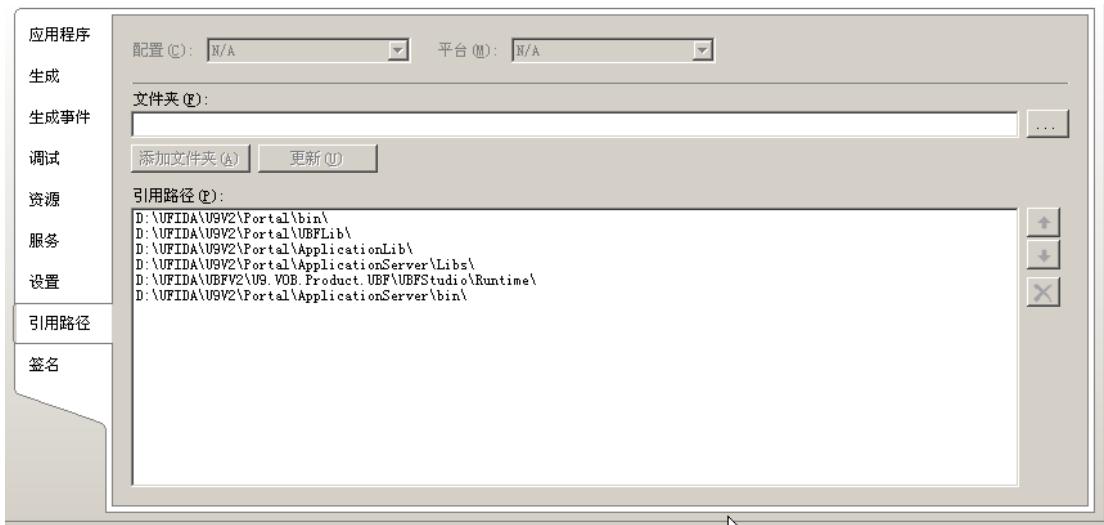


2) UI引用



2、 C#设置引用路径

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\



三、 系统相关

应用组装文件存放位置

D:\UFIDA\U9V2\Portal\ApplicationServer\Libs\SqlScript\

查看应用特性:

D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\Feature

查看应用模式:

D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\Pattern\Balance

系统日志

UBF 自身定义的一套 LOG 接口，基于 LOG4NET 框架包装而成。

- 日志输出地址: D:\UFIDA\U9V2\Portal\log

- 配置路径: Portal\UBFConfig\UFIDA.UBF.Log.Config

- Log 输出类型，从级别低到高依次为：

- Debug: 输出调试信息

- Info: 输出提示信息（该级别可包括警告、错误等信息）

- Warn: 输出警告信息

- Error: 输出错误信息

- Fatal: 输出严重错误信息

如配置文件中如果配置的为 Info，那 Info 及 Info 以上级别的都会输出。

- 系统日志的使用

```
ILogger logger = LoggerManager.GetLogger(typeof(OrderDoc));
if (logger.IsEnabled)
    logger.Info("Info级别的信息");
if (logger.isDebugEnabled)
    logger.Debug("Debug级别的信息");
```

使用 Is**Enabled 的方式主要用于，当 logger 的信息是经过运算或者有一定的处理消耗，那可以用该判断来避免不输出 Log 时，该些方法的无谓的消耗。

四、 数据字典的使用



The screenshot shows a Windows Internet Explorer window titled "实体查询 - Windows Internet Explorer". The address bar displays "http://127.0.0.1/classview/". The main content area is a table titled "实体查询" (Entity Query). The table has columns: 名称 (Name), 全称 (Full Name), 显示名称 (Display Name), and 描述 (Description). The data rows are:

名称	全称	显示名称	描述
AABizType	UFIDA::U9::AAI::AABizType	自动会计业务类型	自动会计类型
AAICreateErrorList	UFIDA::U9::CA::BaseData::AAICreateErrorList	交易分录生成失败日志	
AAIEnum	UFIDA::U9::InvTrans::Enums::AAIEnum	会计分录枚举	会计分录枚举

注：使用前需先在IIS中添加并配置数据连接串。

第二章 档案开发

创建基础档案 BE 实体。

字段类型：字符串、数字、枚举、日期、参照字段；

一、 基础概念

1、 BE组件

1) 什么是BE?

BE (Business Entity) 即我们通常所说的实体，指领域模型中的业务数据对象，如：订单头，客户，地址等。后台的可持久化数据对象，可以理解为数据库的一个表的内存对象类型，其本身也是与数据库表对应的。通过对对象关系映射(ORMAPPING)的持久化机制，就可以实现不用写 SQL 来进行对数据表的加载和保存。使开发的代码隔离了数据库。

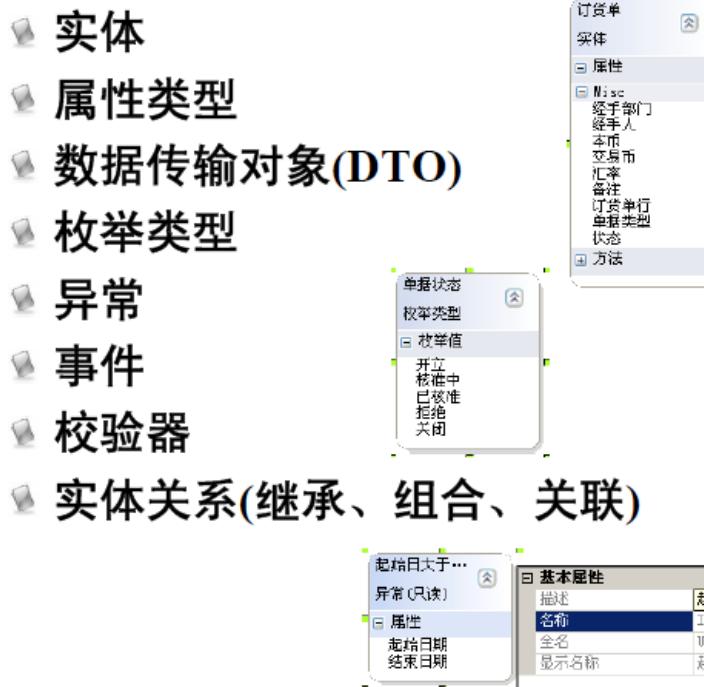
2) 什么是组件?

组件是 UBF 的模型文件组织单位，一个组件对应操作系统中一个实际的模型文件。组件同时也是 UBF 的运行期的执行单位。(如以组件为单位的资源缓存...)

3) 什么是BE组件?

UBF 的基本持久化单元。在实体组件中，只能有一个主实体。也就是其它该组件中的实体，必须要被该主实体直接或间接的组合上。

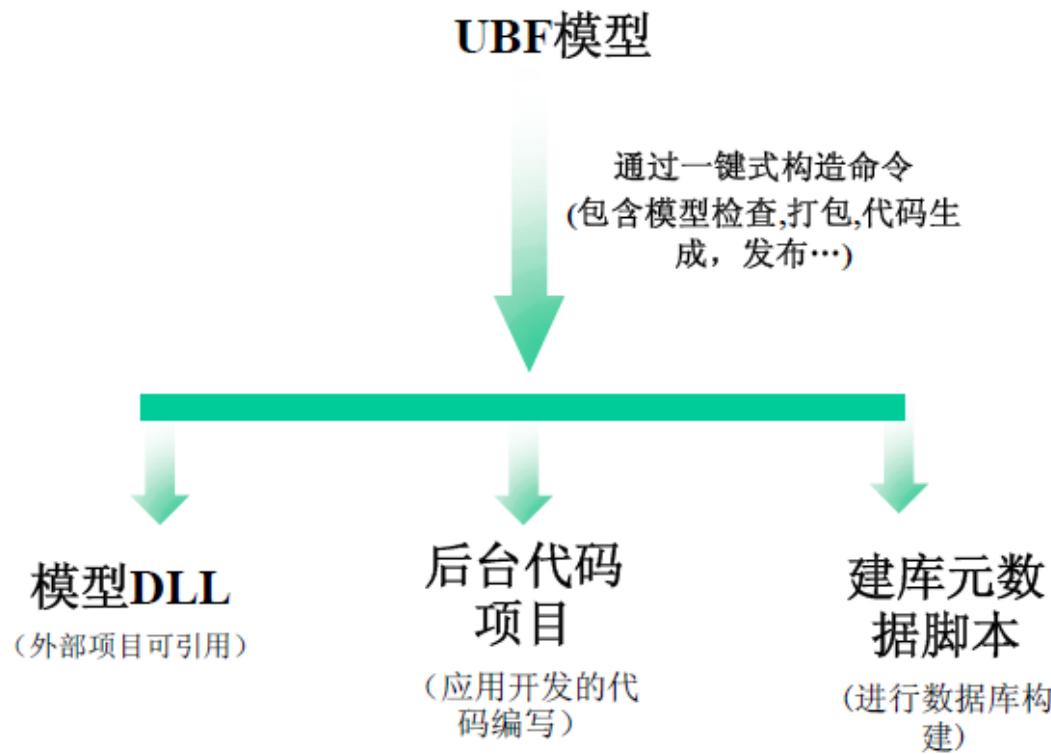
4) 常用模型元素



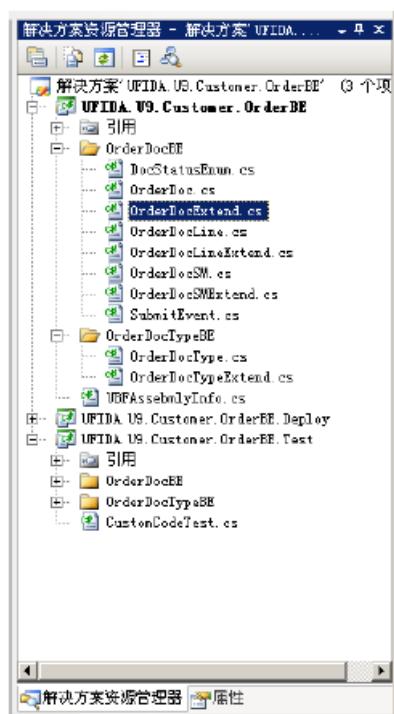
5) 实体设计器的主要功能

- 新建实体对象
- 定义实体字段属性
- 定义实体关系
- 级联删除及敏感字段设置
- 相似实体设计期的复用(特性介绍)
- 特殊实体的版型设置
- 定义审批流状态机模型
- 其他模型元素的定义
- 构造自动生成建库脚本、元数据脚本

6) 实体模型



7) 代码生成框架



实体代码解决方案三个项目

- * BE实现项目，编写应用代码
- * *.Deploy BE部署项目，用于对外部分服务组可见接口，不可改。
- * *.Test BE测试项目，可添加测试用例代码

每个实体组件会生成一个目录，组件内模型均生成在这目录下。

每个实体会生成两个代码文件

例：Order.cs和SaleOrderExtend.cs是一个类型的两个partial文件。只有Extend文件才是用于应用开发编写代码逻辑，通过这种方式来解决，模型变化导致代码变化和应用代码的尽量分离。

开发也可以自己添加代码文件或者DLL引用来加入到当前项目中。

8) BE代码

① 写代码的目的:

- 实体自身数据的完善，实体组件的数据的完善。
- 维持实体自身对象完整性的检查逻辑，——如订单头的客户数据没有是不正确的。
- 维持实体组件完整性的检查逻辑，——如仅有订单头，没有订单行是没有意义的。
- 实体及实体组件持久化时的应用逻辑。
- 实体的模型方法的实现。（对内对外接口）

② 在哪儿写代码?

根据具体需要在实体持久化各个时机（如下）插入自己的业务逻辑

- OnSetDefaultValue(一开始提交时，进行数据完善的事件)
- OnValidate(走完 OnSetDefaultValue 后，开始进行自身数据校验)
- OnInserting/ OnUpdating/ OnDeleting(校验后，数据库实际操作前)
- OnInserted/ OnUpdated/ OnDeleted(数据库实际更改后)

2、 UI组件

1) 什么是UI?

UI 即 User Interface (用户界面)。UI 设计则是指对软件的人机交互、操作逻辑、界面美观的整体设计。好的 UI 设计不仅让软件变得有个性有品味，还要让软件的操作变得舒适、简单、自由，充分体现软件的定位和特点

2) 什么是UIModel?

UIModel 是前台的数据模型，由 UIView、UILink、UIAction 等组成。

UIModel 是单据、档案卡片和列表界面展现与交互的数据模型。

3) 什么是UIView?

UIView 是界面容器的数据集合，对应于数据表（数据视图），由 UIField、UIFilter、UIParameter 等组成。

UIModel 支持业务组件对象持久化的完整事务处理，BE 实体之间的组合关系一定反映为 UIView 之间的主子关系

- UIView 的系统字段

参与持久化的 UIView 必须绑定一个 BE 实体；

UIView 必须有 2 个系统字段分别对应该 BE 实体的 ID、SysVersion；

而且 UIView 中只有与该实体绑定的字段才可以参与持久化；

- **UIView 的字段属性**

参与持久化的 UIView 字段的数据类型、数据格式、数据长度、缺省值等是与绑定的 BE 相对应属性是一致的；

文本型的数据字符串长度无须处理，超长部分在持久层自动截断；

UIView 字段的是否可空属性主要是控制 UI 输入操作和进行前端非空校验的；

如果是编辑数据，不可以进行非空项的减弱设置；

对于 BE 可空属性，可以进行非空属性的增强设置；

参与持久化字段的是否作为参照属性不要改变其缺省设置，否则会影响持久化操作；

- **描述性弹性域字段**

描述性弹性域包括普通段、公共段、私有段等类型，在设计 UIView 时，要把对应 BE 实体的描述性弹性域字段全部绑定，不可差缺；

- **系统字段的业务借用**

在 BE 实体中 CreateOn、CreateBy、ModifiedOn、ModifiedBy 等 4 个字段是系统预留的，一般不参与业务处理；

但是如果业务模型上有这些业务字段，可以借用这 4 个系统属性，并把 CreateOn、CreateBy、ModifiedOn、ModifiedBy 加入 UIView 字段中；

- **Link**

实体之间的组合关系一定要反映为 UIView 之间的主子关系；

在 UIModel 中以 UILink 自动建立 UIView 之间的连接；

如果不建立 Link 关系，持久层无法完成被组合实体的持久化事务操作；

- **UIView 分组**

UI 界面分组来源于 UIView 分组；界面分组可以有 3 层：页签、分组线、分组框，一般 1~2 层即可；

- **自定义字段**

UIView 自定义字段有两个作用：界面布局和信息资源；

自定义字段不参与 CRUD；但可以参与交互处理；自定义字段可以是目前 UIView 已经支持的任何数据类型；

界面业务操作的提示信息，除了公共信息外，全部来源于自定义字段的 DisplayName；

- **实体关联字段**

UIView 绑定 BE 实体，该 BE 实体的关联实体的属性可以作为 UIView 字段；

关联字段可以参与数据加载，但不参与 CUD；

- **Filter**

Filter 设置加载数据的过滤条件，仅仅在主 UIView 上进行设置；设置的条件一定要全部绑定参数后才能够加载数据；

4) 什么是UIAction？

Action 设计主要包括ToolBar 操作和额外功能操作；

系统缺省 Action 已经通过模板在模型生成代码时，实现一些默认ToolBar 按钮功能；

5) UI模型设计器之表单设计器

① 基本 UI 界面模式

- Card: 是单一对象的交互模式。



单据类型	<input type="text"/>	<input type="button" value=""/>	*	交易币	<input type="text"/>	<input type="button" value=""/>
单号	<input type="text"/>	<input type="button" value=""/>	*	汇率	<input type="text"/>	<input type="button" value=""/>
日期	<input type="text"/>	<input type="button" value=""/>	*	经手部门	<input type="text"/>	<input type="button" value=""/>
本币	<input type="text"/>	<input type="button" value=""/>		经手人	<input type="text"/>	<input type="button" value=""/>



基本	币种	外销	商务	物流	财务	
单据类型	<input type="text"/>	<input type="button" value=""/>	*	币种	<input type="text"/>	<input type="button" value=""/>
单号	自动编号	<input type="button" value=""/>	*	业务员	<input type="text"/>	<input type="button" value=""/>
客户	<input type="text"/>	<input type="button" value=""/>	*	部门	<input type="text"/>	<input type="button" value=""/>
交期	2009.05.26	<input type="button" value=""/>		来源类别	手工创建	<input type="button" value=""/>
日期	2009.05.26	<input type="button" value=""/>	*	来源单据	<input type="text"/>	<input type="button" value=""/>

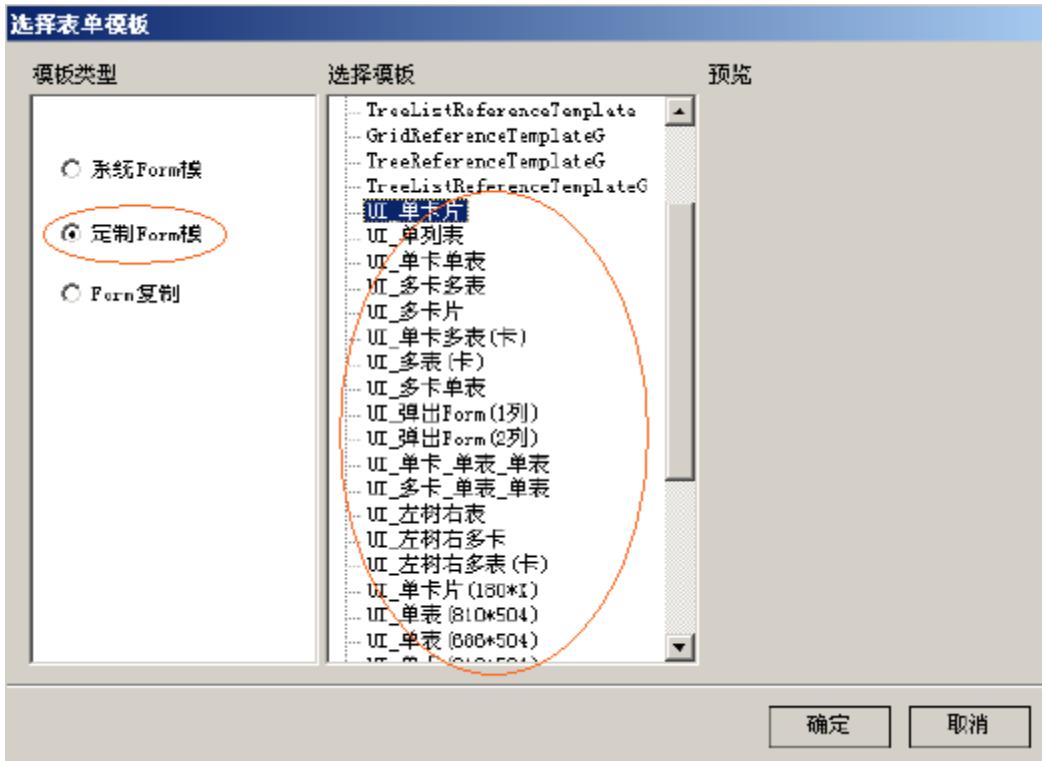
- List: 是集合对象的交互模式。

行号	客户	料品	料品名称	规格型号	数量	单位	含税单价	含税金额	币
10					0		0	0	

1/1 [首页] [上页] [下页] [末页] [清]

- Tree:

所有业务表单 UI 界面均是以是几种基本模式的组合，UI 基本界面模式构成各种业务表单，如档案、单据、列表。



② 基本操作模式 (Action)

保存、新增、放弃、翻页、流程、打印、输出等

所有表单的处理动作是这些标准操作的组合

工具栏 Action

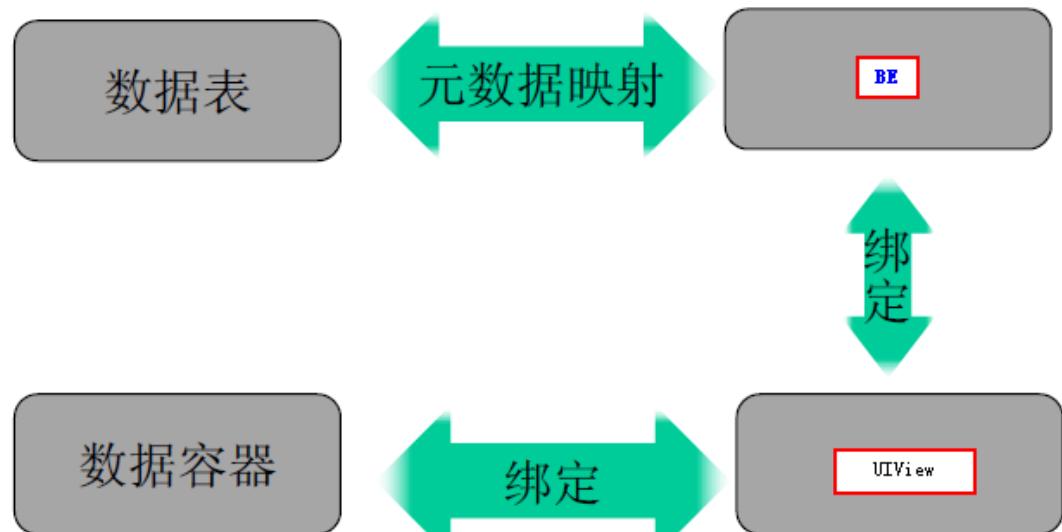
保存	放弃	新增	删除	复制	提交	审核	共享	查找	列表	首页	上页	下页	末页	附件	流程	输出	打印	设置默认值																																																																																																																	
<table border="1"> <tr> <td colspan="2">基本</td><td>币种</td><td>外销</td><td>商务</td><td>物流</td><td>财务</td><td colspan="12"></td> </tr> <tr> <td colspan="2">单据类型</td><td>单号</td><td>客户编号</td><td>业务员</td><td>部门</td><td>备注</td><td colspan="12"></td> </tr> <tr> <td colspan="2">单号</td><td>自动编号</td><td>01</td><td></td><td></td><td></td><td colspan="12"></td> </tr> <tr> <td colspan="2">客户</td><td>客户编号</td><td></td><td>部门</td><td></td><td></td><td colspan="12"></td> </tr> <tr> <td colspan="2">交期</td><td>2009.05.26</td><td></td><td>未清类别</td><td>手工创建</td><td></td><td colspan="12"></td> </tr> <tr> <td colspan="2">日期</td><td>2009.05.26</td><td></td><td>未清单据</td><td></td><td></td><td colspan="12"></td> </tr> </table>																		基本		币种	外销	商务	物流	财务													单据类型		单号	客户编号	业务员	部门	备注													单号		自动编号	01																客户		客户编号		部门															交期		2009.05.26		未清类别	手工创建														日期		2009.05.26		未清单据														
基本		币种	外销	商务	物流	财务																																																																																																																													
单据类型		单号	客户编号	业务员	部门	备注																																																																																																																													
单号		自动编号	01																																																																																																																																
客户		客户编号		部门																																																																																																																															
交期		2009.05.26		未清类别	手工创建																																																																																																																														
日期		2009.05.26		未清单据																																																																																																																															
<table border="1"> <thead> <tr> <th>行</th><th>料品</th><th>交易</th><th>发货</th><th>财务</th><th>其它</th><th colspan="12"></th> </tr> </thead> <tbody> <tr> <td>10</td><td></td><td></td><td></td><td></td><td></td><td>数量1</td><td>指售单位</td><td>计价数量</td><td>计价单位</td><td>交期</td><td colspan="12"></td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td>0</td><td></td><td>2009.05.26</td><td colspan="12"></td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td>0</td><td></td><td></td><td colspan="12"></td> </tr> </tbody> </table>																		行	料品	交易	发货	财务	其它													10						数量1	指售单位	计价数量	计价单位	交期																			0		0		2009.05.26																			0		0																																									
行	料品	交易	发货	财务	其它																																																																																																																														
10						数量1	指售单位	计价数量	计价单位	交期																																																																																																																									
						0		0		2009.05.26																																																																																																																									
						0		0																																																																																																																											
1/1 [首页] [上页] [下页] [末页] [共 00 页]																																																																																																																																			
快速录入 价格重算 预留 操作 后续处理 客户 料品 相关功能																																																																																																																																			

功能区 Action

6) 视图模型 (UIView) 和界面模型的关系

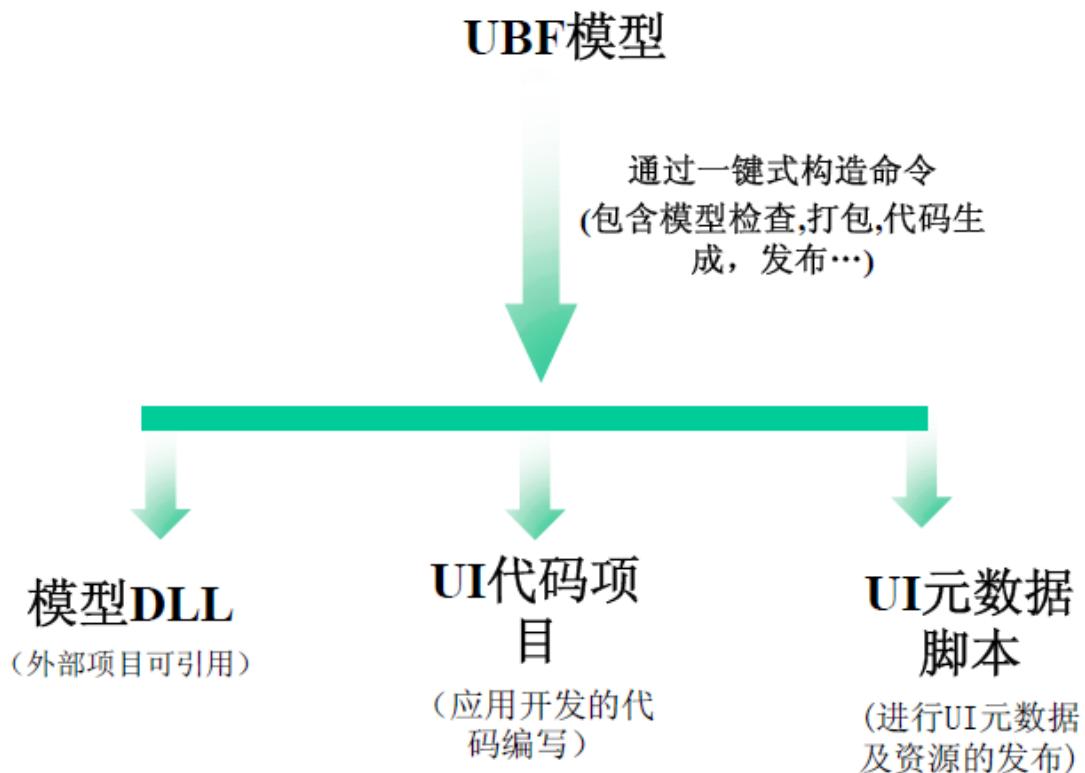
- 一张表单中的 Card, Grid (对应 List), Tree 又叫数据容器;

-
- 每个容器对应一个 **UIView**;

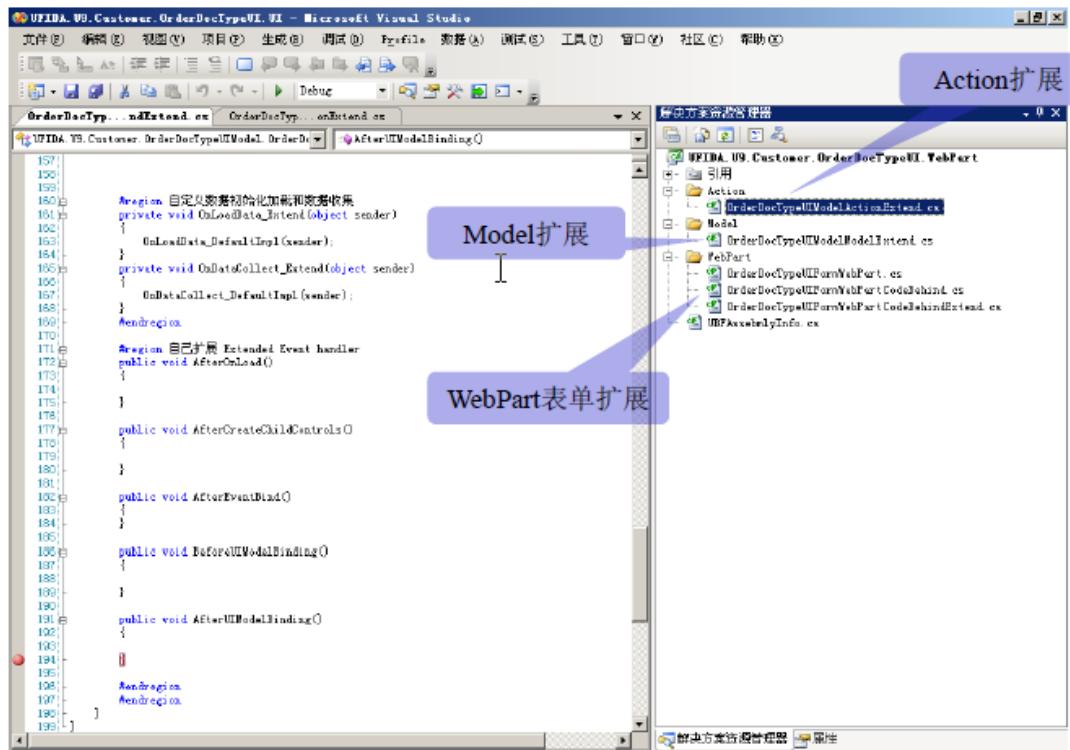


BE 组件—>UIModel→UI Form。

7) UI模型



8) UI代码生成框架



9) UI代码

① 写代码的目的:

- UI 数据加载前后的数据转换
- UI 操作对应后台的响应
- UI 字段状态控制
- UI 交互控制
- 自定义视图（字段）的数据加载

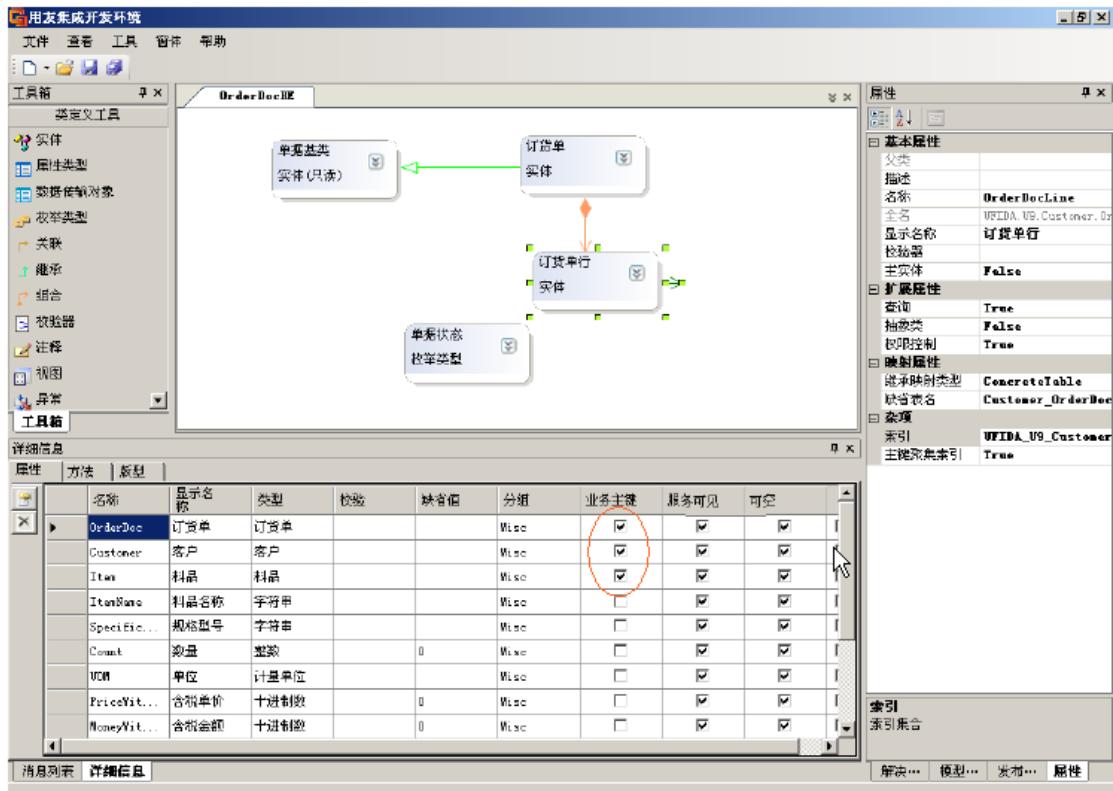
② 在哪儿写代码?

UI 端不能调 BE, 最多也只能调用 BP。

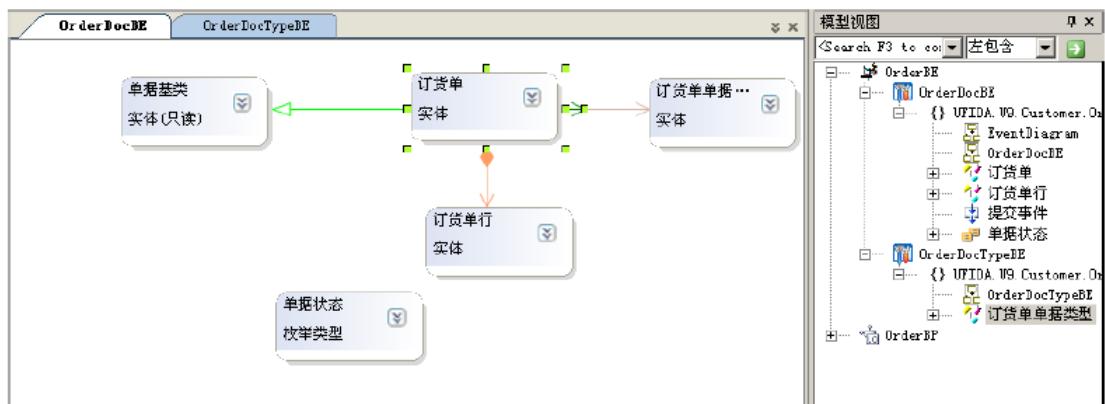
3、 其他应用

1) 实体业务主键

非聚集性唯一索引。数据表主键 ID。



2) 实体关系



- ① 对一个案例如何分析确定实体关系
 - 确定各实体
 - 划分实体组件
 - 确定组件主实体
 - 定义组件内实体的组合关系
- ② 实体级联删除



对级联规则为 NoAction 的情况，增加一个是否启用级联效验的标志，只有当这个标志为 true，才进行 NoAction 下的引用效验，而这个是否启用级联效验的标志，也只在规则为 NoAction 时有效，其他规则下都无效，即：

- 级联删除：是 bool 型，表示在删除时关联关系是不是要做及联处理；
- 级联删除规则：只有当及联删除标志为 true 时，这个规则才起作用，有 3 种情况，cascade, noaction, setnull；
- 是否启用及联效验：只有当级联删除为 true，及联删除规则为 noaction 时才有效，表示这个关联是否进行被引用不能删除的检查。

③ 实体级联修改（敏感字段设置）



例：订货单单据类型的 Code 设置为“一旦使用不可修改”的属性，那么，当修改这个属性时，如果发现有订货单引用了这个单据类型，则抛异常“订货单单据类型已被使用，不能修改”。

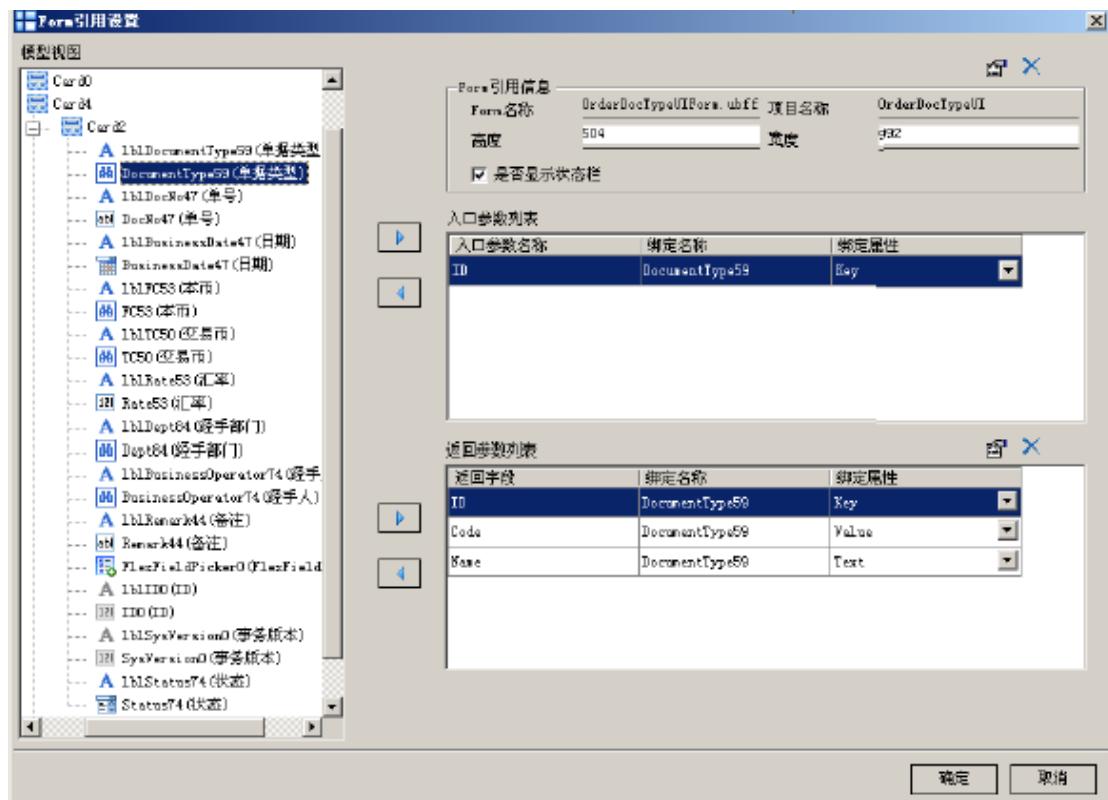
实体设计器敏感控制开关：

- 实体关系线的属性：检查修改
- 实体字段属性：一旦使用不可修改

场景号	级联删除	启用级联校验	级联删除规则	实现效果	异常提示信息
1	true	true	cascade		
2	true	true	noaction		
3	true	true	setnull		
4	false	true	不限		

3) Form引用

- ① 设置被引用 Form 的传入传出参数
通常情况：传入参数 **ID**，传出参数 **ID, Code, Name**
- ② 添加需要引用的 UI 元数据
- ③ 设置 label 的 Form 引用参数

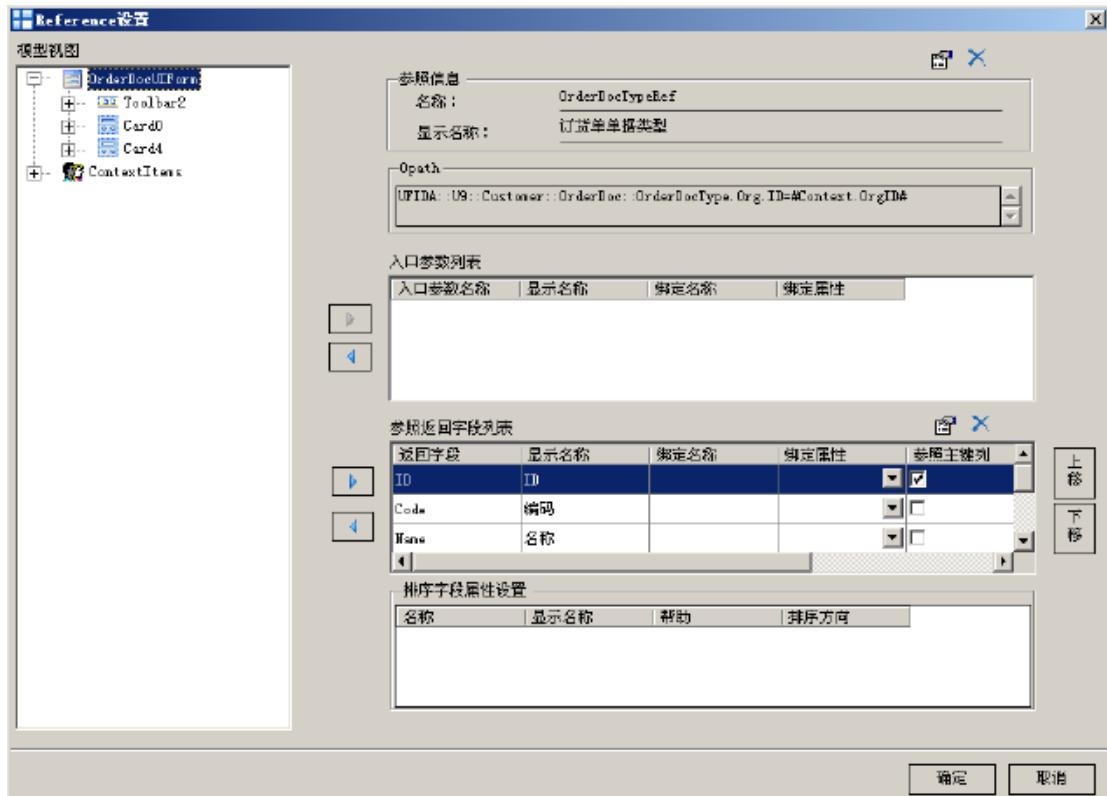


4) 参照绑定



- ① 将包含参照的项目加入项目引用
- ② 点击“参照信息”即可设置参照的参数绑定
- ③ 通常参照会返回该对象的 ID, Code, Name 值, 如果界面需要绑定可以设定参照返回值的绑定。

● **ID, Code, Name 值不需单独绑定。**

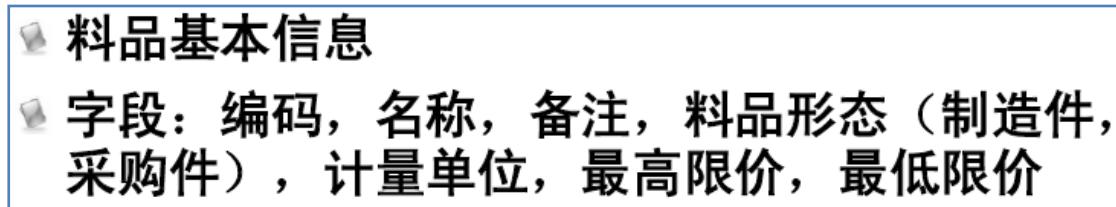


● **数值控件精度的绑定:** 需根据参照返回绑定精度、舍入值和舍入类型。详

见 UI 高级开发中的“数量金额处理”。

如：币种的参照，通常在设计参照时会将币种的精度，舍入值，舍入类型带到界面，绑定需要精度控制的数字控件对应的值即可完成精度的控制。

二、 实体建模

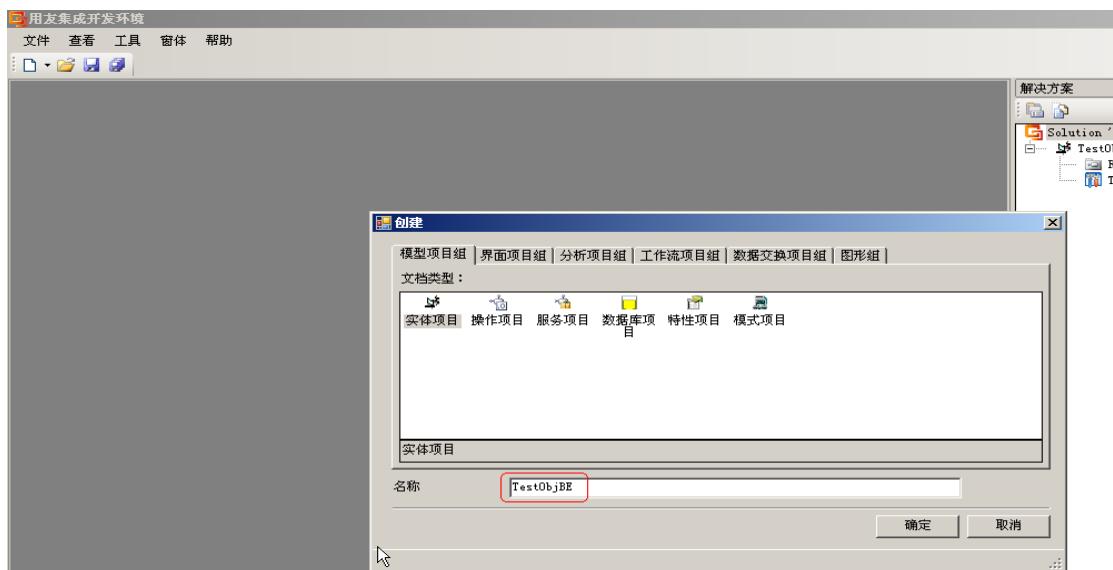


1、 创建解决方案



2、 创建BE组件

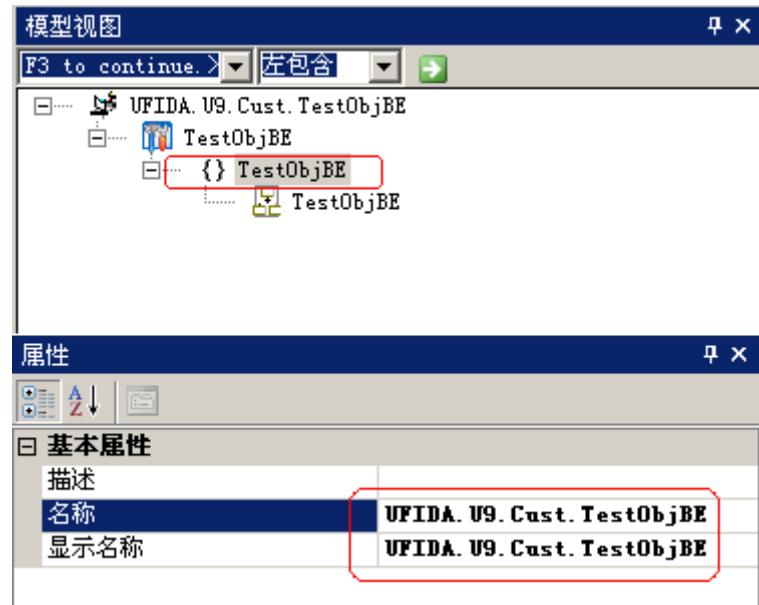
1) 创建BE实体项目



2) 修改BE组件的配件名及命名空间

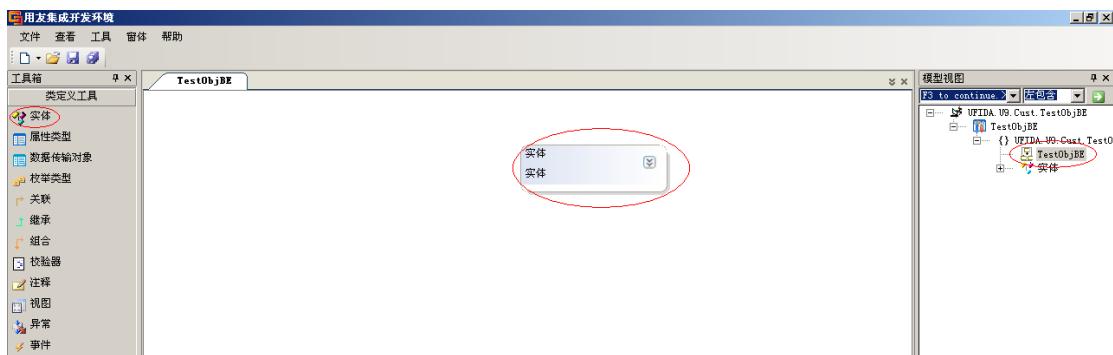


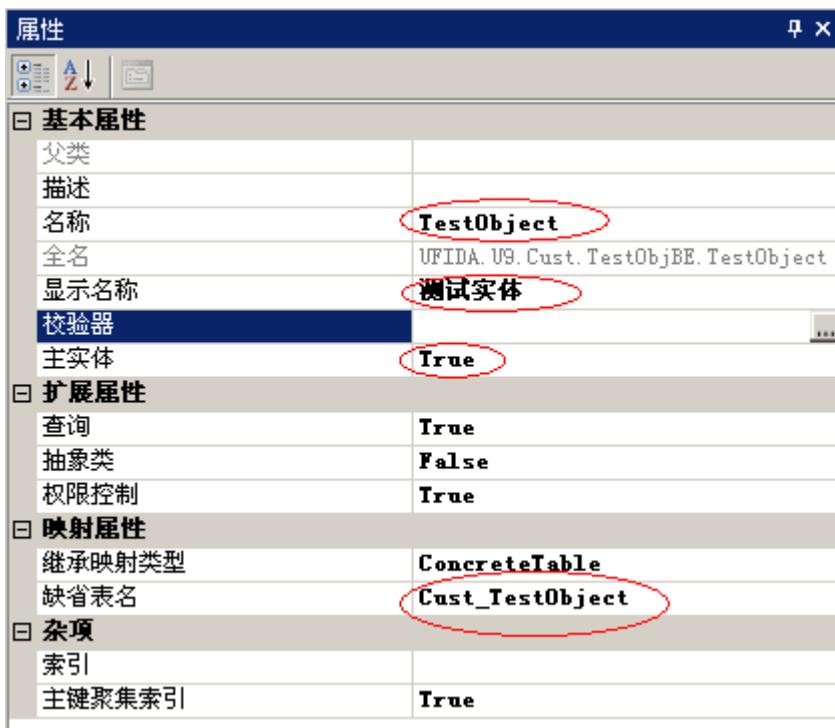
注：[名称]为编译后 Code 代码的文件夹名称，若为“XXX.XXX.XXX”格式，则省略最后一节作为文件名；[配件名]为 C# 工程的项目及 DLL 文件名。



注：[名称]为 C#语句中的命名空间，其命名规范均以“UFIDA.U9.Cust.XXX”，且首字母大写。

3) 添加实体并修改相关属性





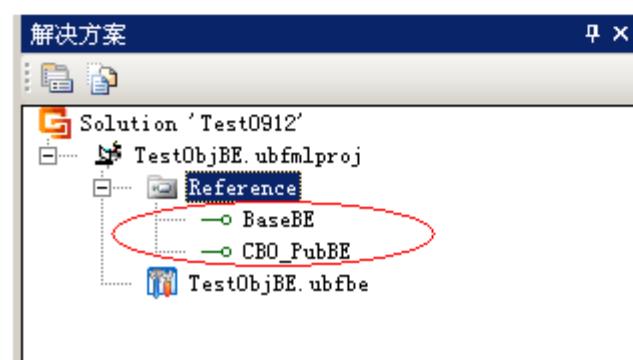
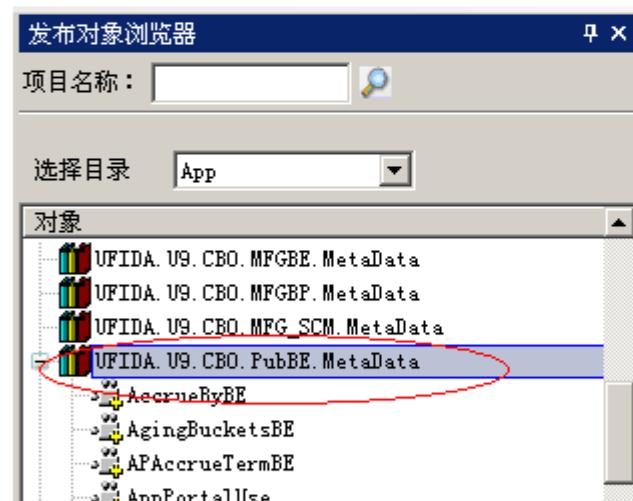
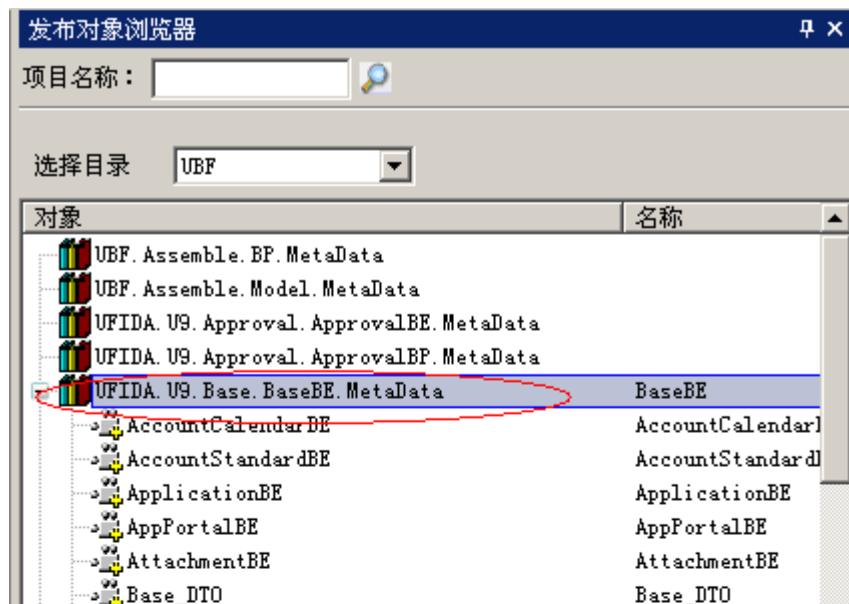
- 当存在多个实体时，其它实体必须将主实体属性设为 False；
- 缺省表名即数据库中表名；
- [全名]为命名空间.实体名称。

4) 建立实体字段

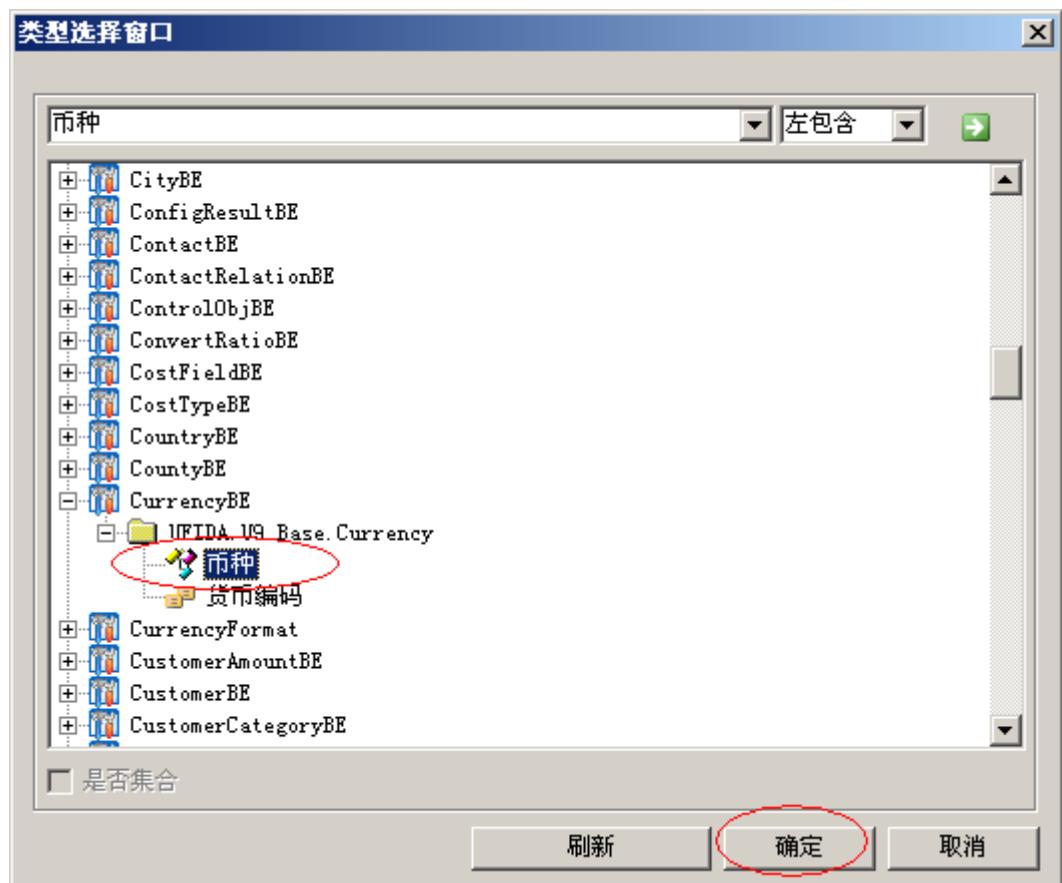
属性	显示名称	类型	枚举	缺省值	分组	业务主键	服务可见	可空	唯一
Code	编码	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Name	名称	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Remark	备注	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CurrencyType	币种	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Importance	重要程度	字符串		0	Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Price	单价	十进制数		0	Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
FileDate	日期	日期			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
*						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

5) 关联U9 基础档案引用

1) 添加 U9 基础对象引用



2) 设置字段类型，引用基础参照。

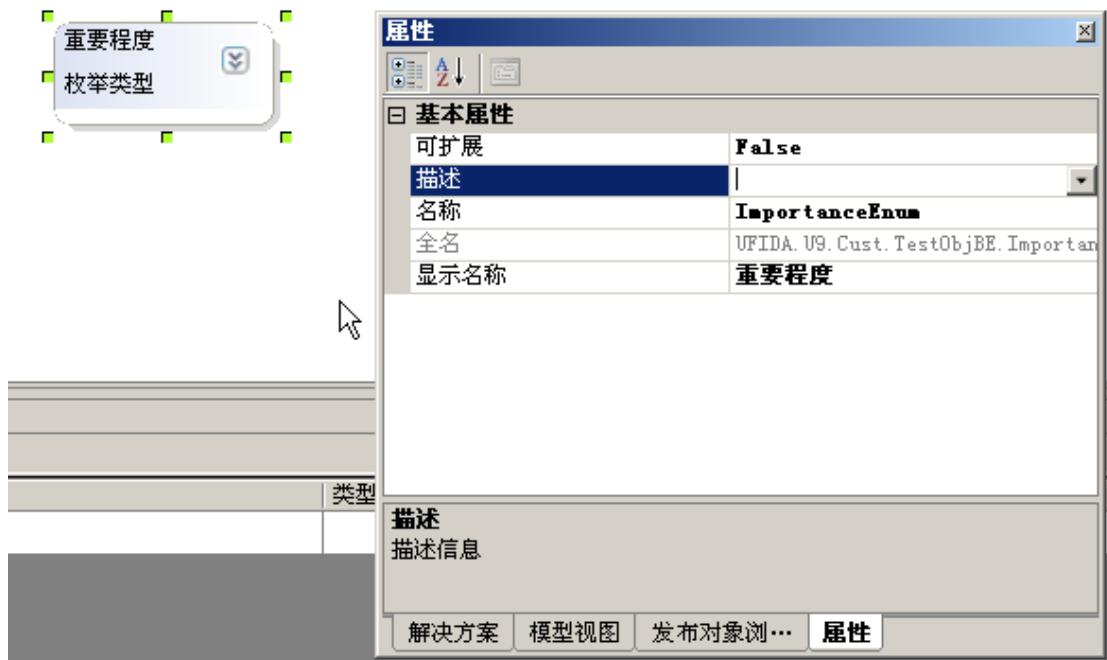


详细信息

属性 | 方法 |

	名称	显示名称	类型	校验	缺省值	分组	业务主键	服务可见	可空	
	Code	编码	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Name	名称	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Remark	备注	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
β	CurrencyType	币种	币种 <input checked="" type="checkbox"/>			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
*	Importance	重要程度	字符串			Misc	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

6) 设置枚举

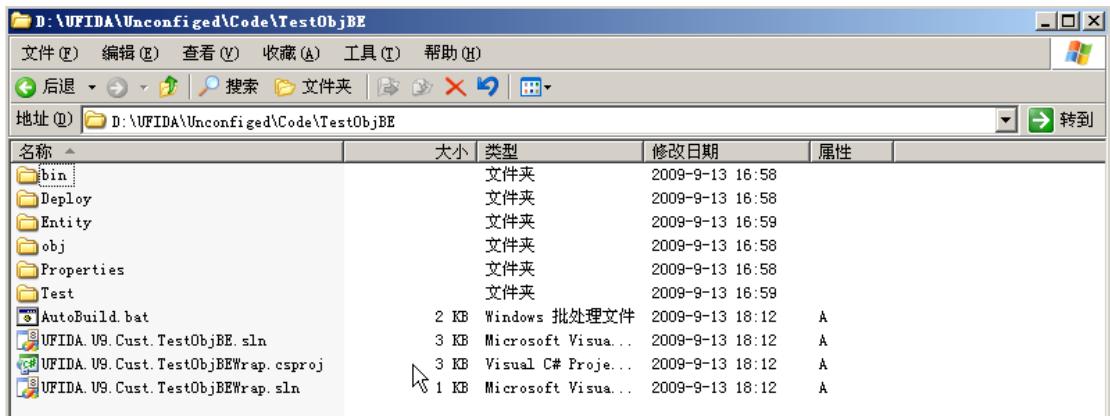


	名称	显示名称	类型	校验	缺省值
	Code	编码	字符串		
	Name	名称	字符串		
	Remark	备注	字符串		
	CurrencyType	币种	币种		
▶	Importance	重要程度	重要程度		
*					

3、构造BE

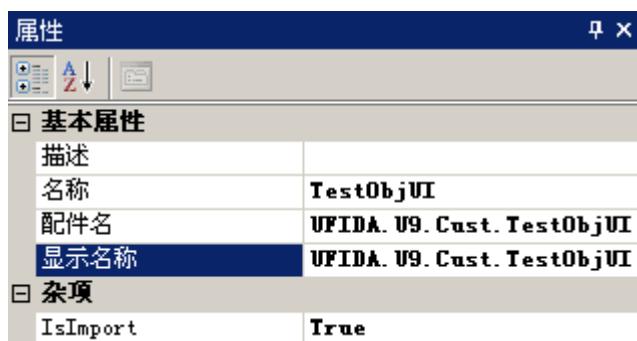
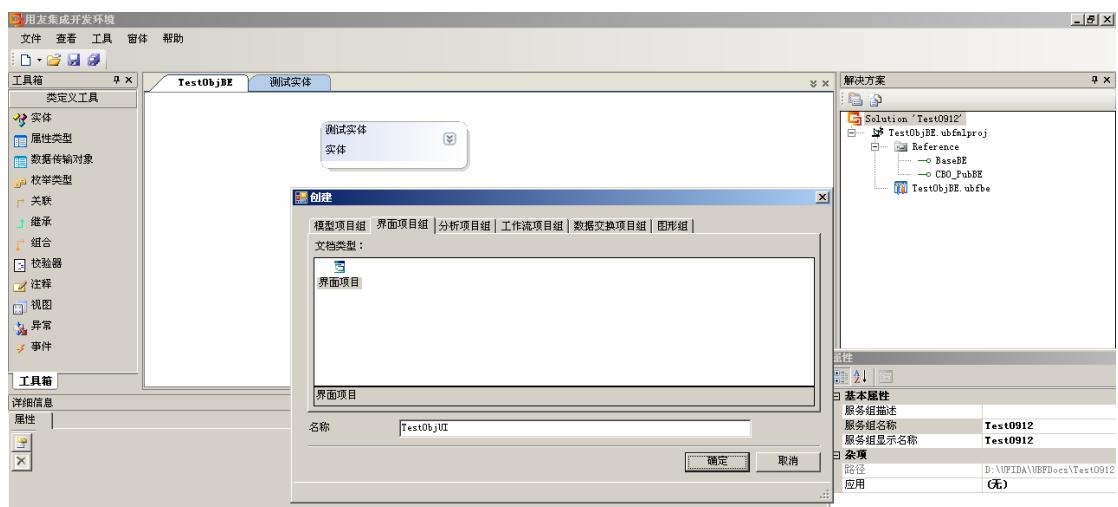


构造成功后，会相应生成代码文件。



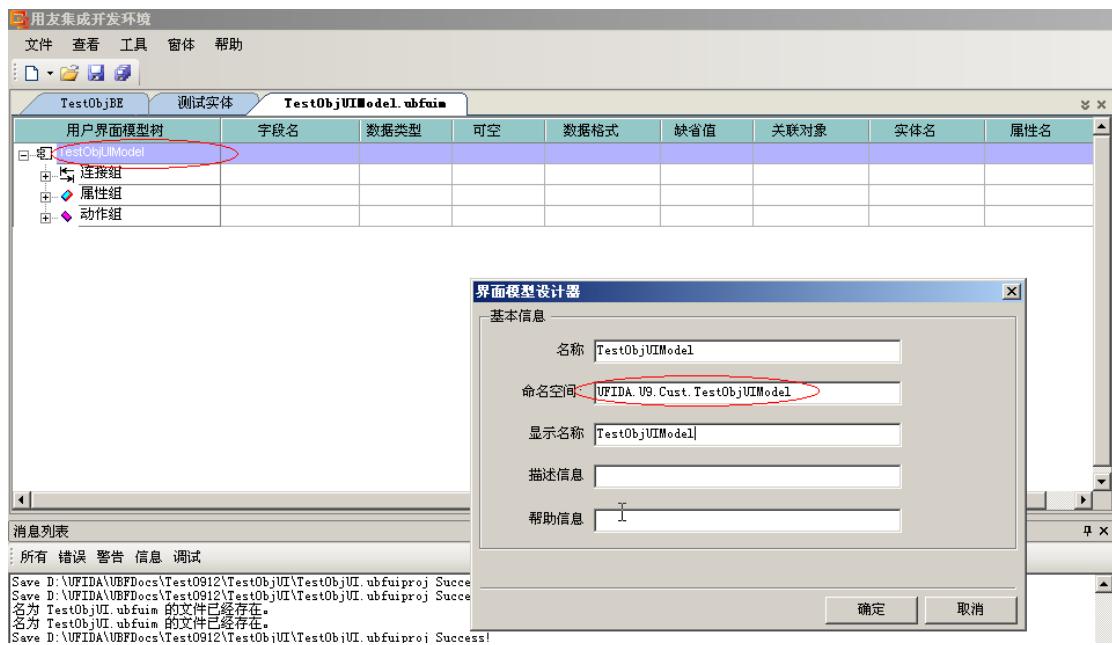
三、 创建UI组件

1、 建立UI项目



2、 建立UIModel

建议先删除 UI 项目下的原实体后再另建。



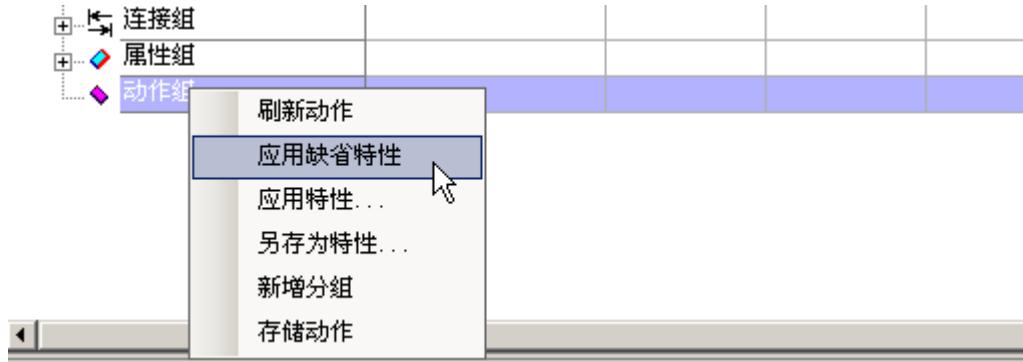
① 挂接实体



② 删除 Sys 节点下的无用字段，必需保留 SysVersion

用户界面模型树	字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
TestObjUIModel								
TestObject								
Key								
Sys								
Misc								
属性组								
DefaultFilter	缺省过滤条件							
连接组								
属性组								
动作组								

③ 添加动作，以备挂接菜单



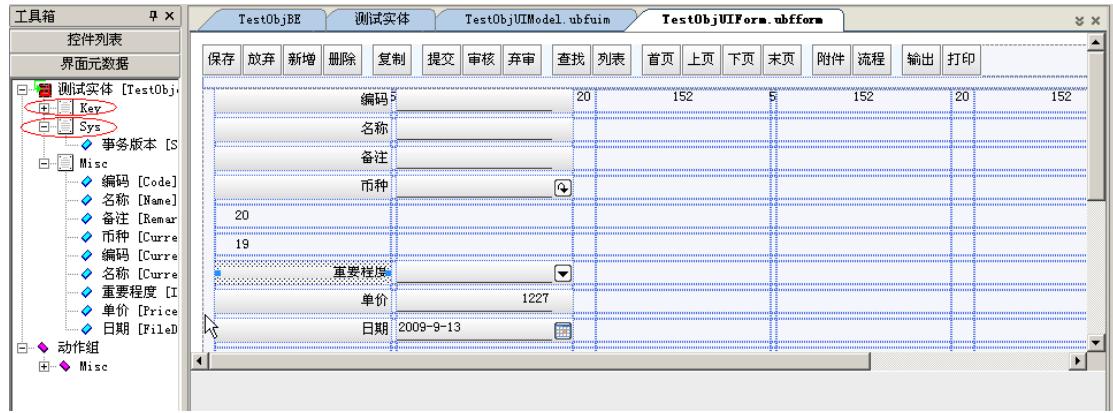
3、 创建UIForm

① 建立 UIForm

The screenshot shows the UFT interface for creating a UIForm:

- Solution Explorer:** Shows the solution 'Test0912' containing projects 'TestObjUIModel.ubfui' and 'TestObjUIModel.ubfuiproj'.
- Properties View:** Displays properties for the 'TestObjUIModel' entity, specifically for the 'Key' field under the 'Sys' group. The 'SysVersion' field is set to '长整数' (Long Integer) with a value of '19.0'.
- UIForm Wizard:**
 - 新建表单 (New Form):** Shows the '模版类型 (Template Type)' section with '定制Form模 (Custom Form Model)' selected.
 - 选择模板 (Select Template):** Shows a list of UI templates, including 'UI_单卡片' (UI_SingleCard) and 'UI_多卡片' (UI_MultiCard).
 - 预览 (Preview):** Shows a preview of the selected template.
 - 界面元数据 (Interface Metadata):** Shows 'TestObjUIModel'.
 - 确定 (OK) / 取消 (Cancel):** Buttons to confirm or cancel the selection.

② 绑定 ID: 选定表后, 双击 Key、Sys



③ 指定参照项目

- 添加参照对象引用

发布对象浏览器

项目名称： []

选择目录 UI

对象

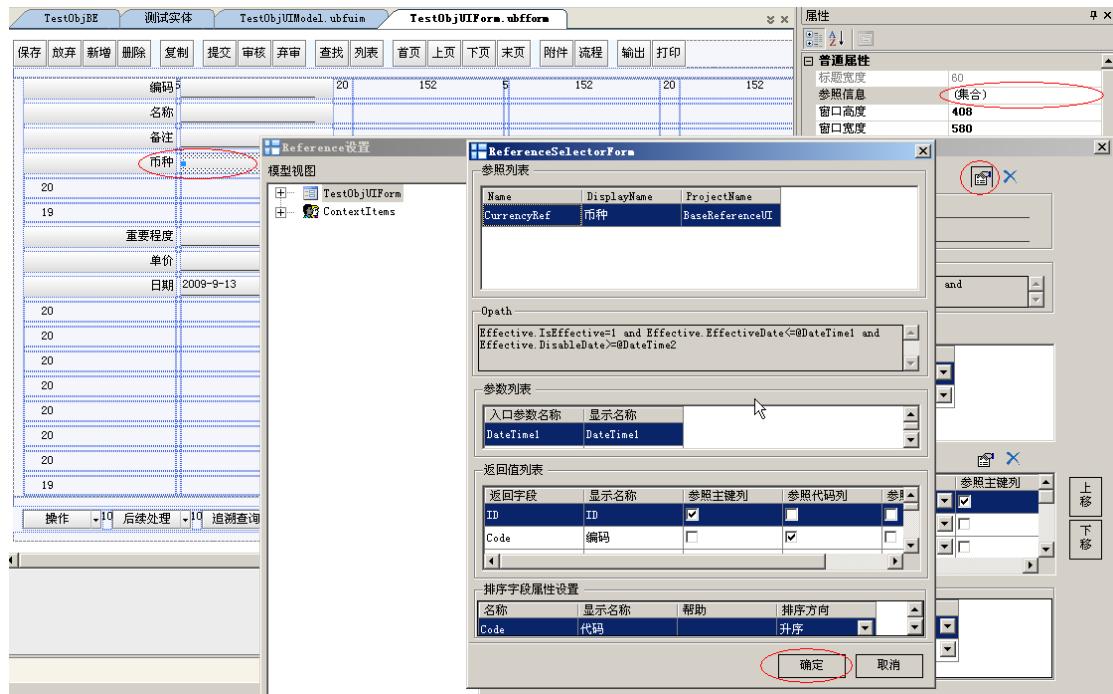
- UFIDA.U9.CBO.Ref.QCReferenceUI_MetaData
- UFIDA.U9.CBO.Ref.FIReferenceUI_MetaData
- UFIDA.U9.CBO.Ref.CBOReferenceUI_MetaData
- UFIDA.U9.CBO.Ref.BaseReferenceUI_MetaData**
- WorkCalendarRef.ubref
- ValueSetRef.ubref
- ValueSet4KeyFlexField.ubref

解决方案

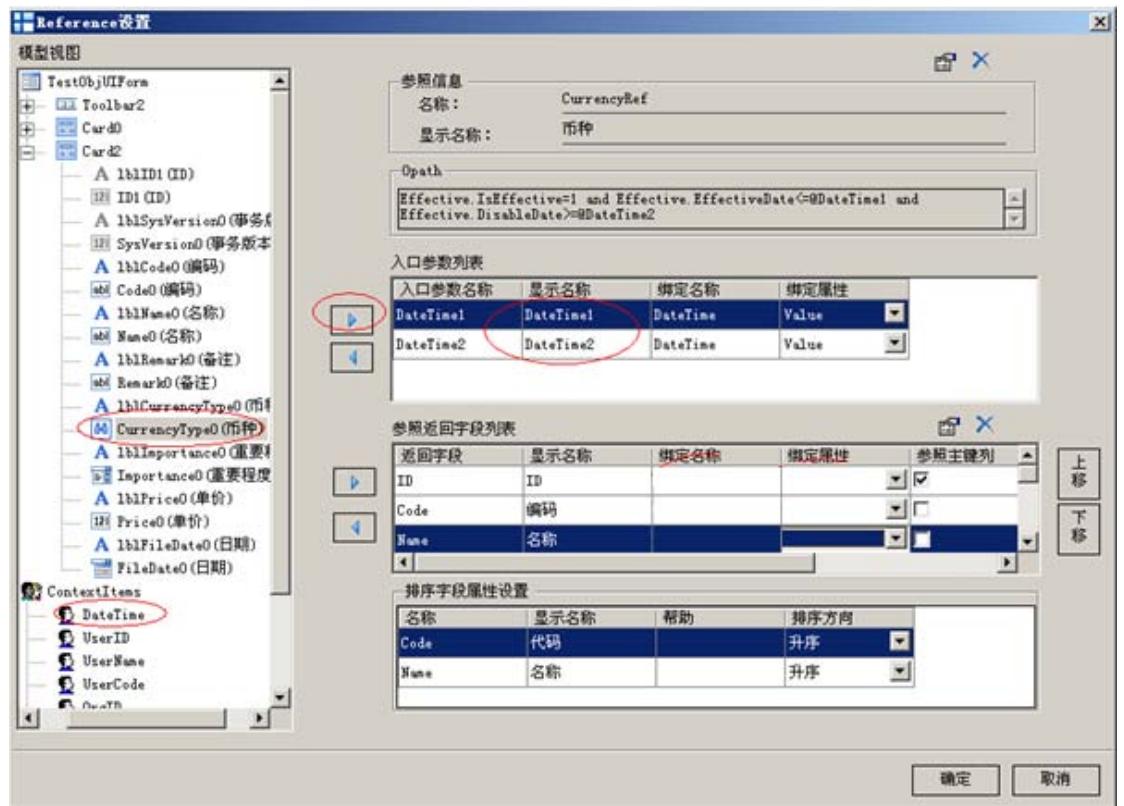
Solution 'Test0912'

- TestObjBE. ubfmlproj
 - Reference
 - BaseBE
 - CBO_PubBE
 - TestObjBE. ubfbe
- TestObjUIModel. ubfuiproj
 - Reference
 - TestObjBE
 - BaseReferenceUI**
 - CBOReferenceUI**
 - TestObjUIModel. ubfuim
 - TestObjUIForm. ubfform

- 查看参照条件 (opath 中)



- 绑定入口参数，返回字段 ID、Code、Name 可不绑定

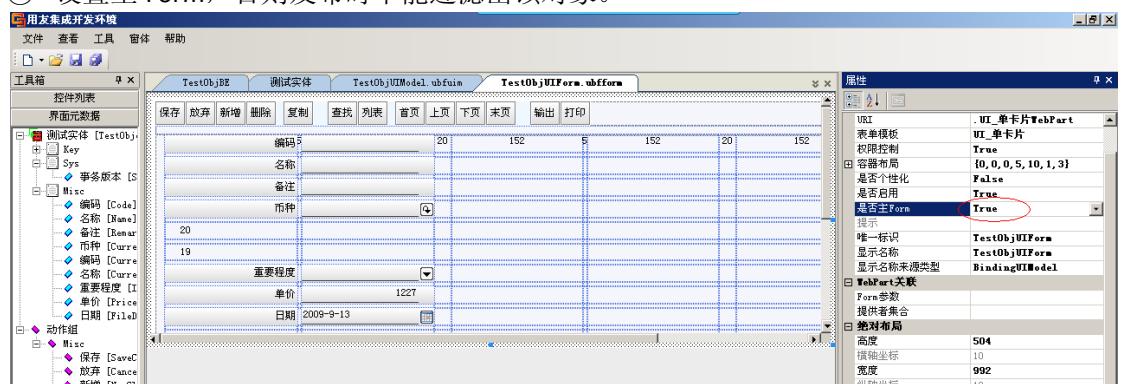


④ 设置 Form 引用

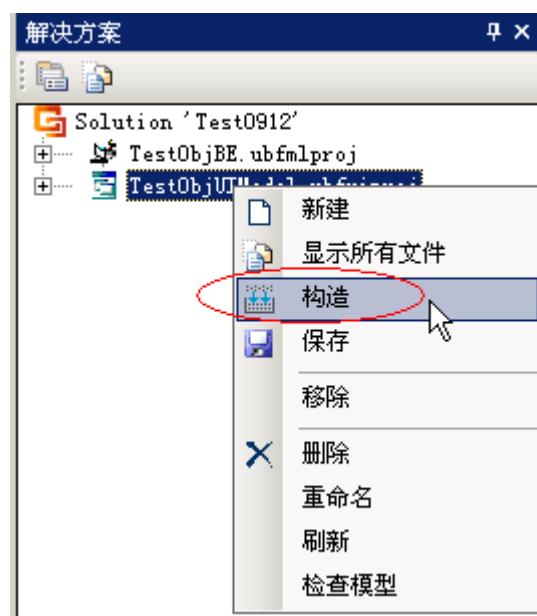
⑤ 启用按钮事件



⑥ 设置主 Form，否则发布时不能过滤出该对象。



4、构造UI组件

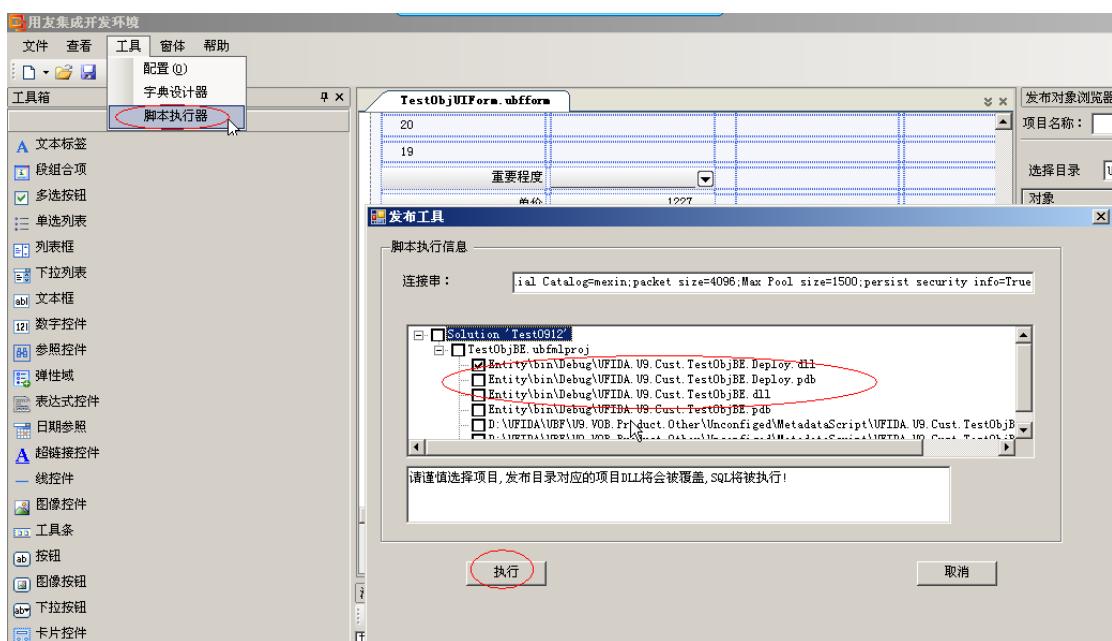


四、组件发布与装配

1、发布服务组



2、将组件DLL文件拷贝至U9 相应目录



也可到相应目录下执行 AutoBuild 批处理文件。

D:\UFIDA\Unconfiged\Code\TestObjBE

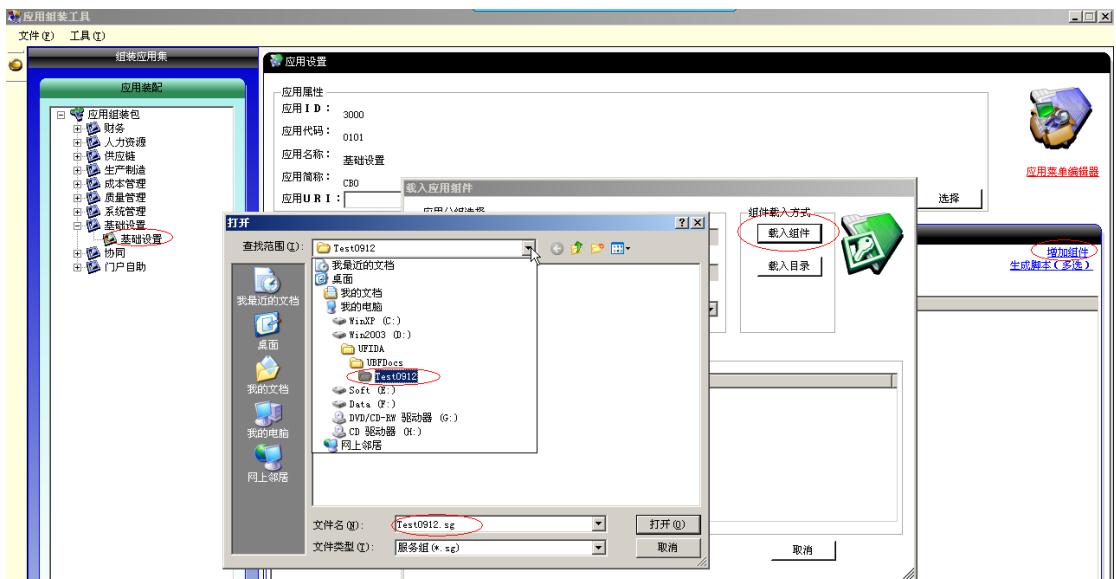
名称	大小	类型	修改日期	属性
bin		文件夹	2009-9-13 16:58	
Deploy		文件夹	2009-9-13 16:58	
Entity		文件夹	2009-9-13 16:59	
obj		文件夹	2009-9-13 16:58	
Properties		文件夹	2009-9-13 16:58	
Test		文件夹	2009-9-13 16:59	
AutoBuild.bat	2 KB	Windows 批处理文件	2009-9-13 20:50	A
UFIDA.U9.Cust.TestObjBE.sln	3 KB	Microsoft Visual Studio Solution File	2009-9-13 20:50	A
UFIDA.U9.Cust.TestObjBEWrap.csproj	3 KB	Visual C# Project File	2009-9-13 20:50	A
UFIDA.U9.Cust.TestObjBEWrap.sln	1 KB	Microsoft Visual Studio Solution File	2009-9-13 20:50	A

D:\UFIDA\Unconfiged\Code\TestObjUI

名称	大小	类型	修改日期	属性
Action		文件夹	2009-9-13 20:18	
bin		文件夹	2009-9-13 20:18	
Model		文件夹	2009-9-13 20:18	
obj		文件夹	2009-9-13 20:18	
Properties		文件夹	2009-9-13 20:17	
WebPart		文件夹	2009-9-13 20:18	
AutoBuild.bat	1 KB	Windows 批处理文件	2009-9-13 20:51	A
UFIDAAssemblyInfo.cs	1 KB	Visual C# Source File	2009-9-13 20:51	A
UFIDA.U9.Cust.TestObjUI.UI.sln	1 KB	Microsoft Visual Studio Solution File	2009-9-13 20:51	A
UFIDA.U9.Cust.TestObjUI.WebPart.csproj	2 KB	Visual C# Project File	2009-9-13 20:18	A
UFIDA.U9.Cust.TestObjUI.WebPart.target	12 KB	TARGET 文件	2009-9-13 20:51	A
UFIDA.U9.Cust.TestObjUIWrap.csproj	3 KB	Visual C# Project File	2009-9-13 20:51	A
UFIDA.U9.Cust.TestObjUIWrap.sln	1 KB	Microsoft Visual Studio Solution File	2009-9-13 20:51	A

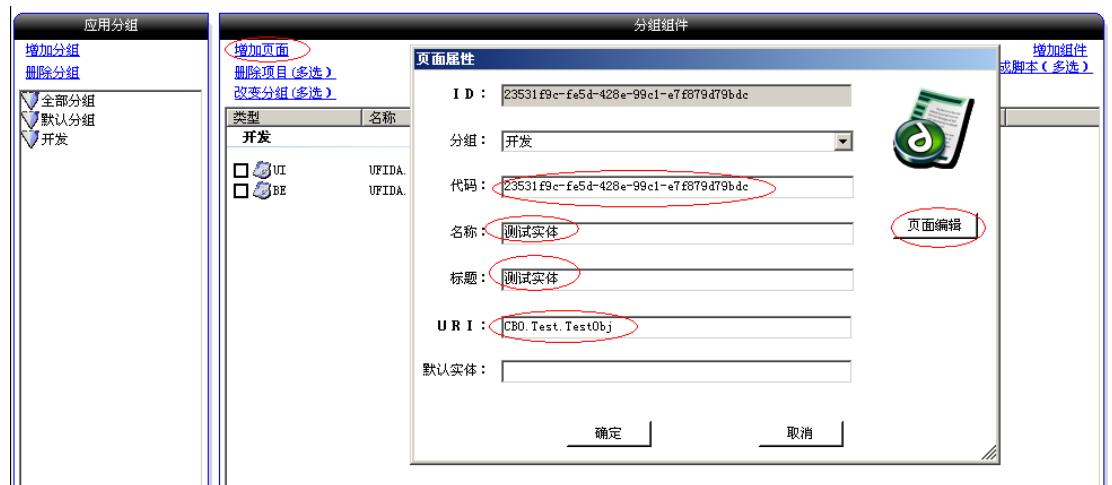
3、通过装配工具进行装配发布。

1) 加载开发的组件



2) 装配页面

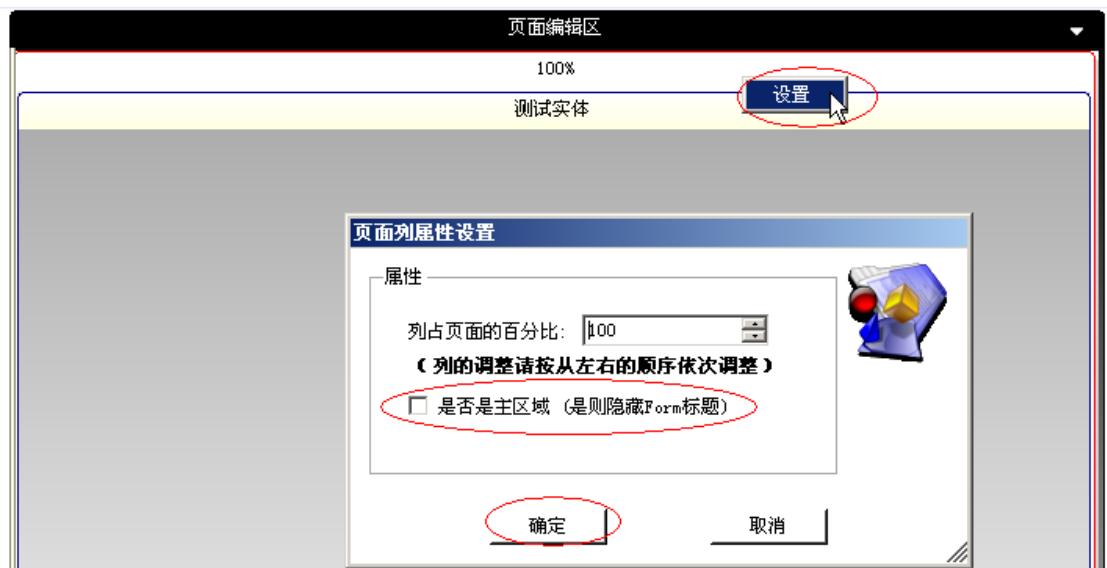
- 增加页面



- 设置页面属性



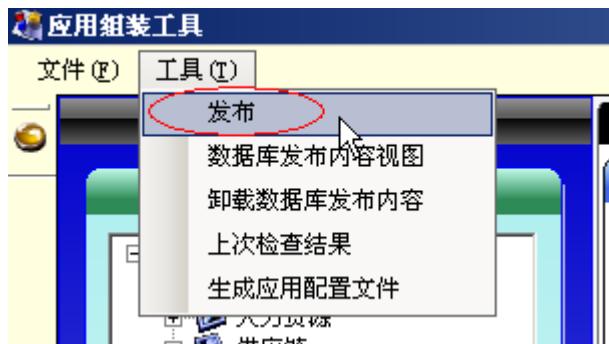
- 设置主区域

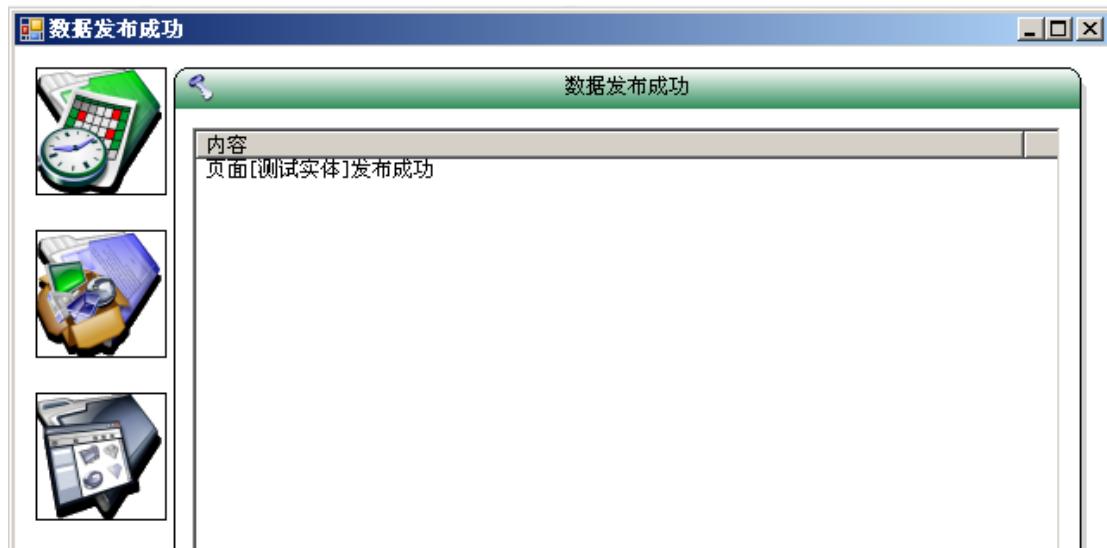


3) 装配菜单



4) 装配发布

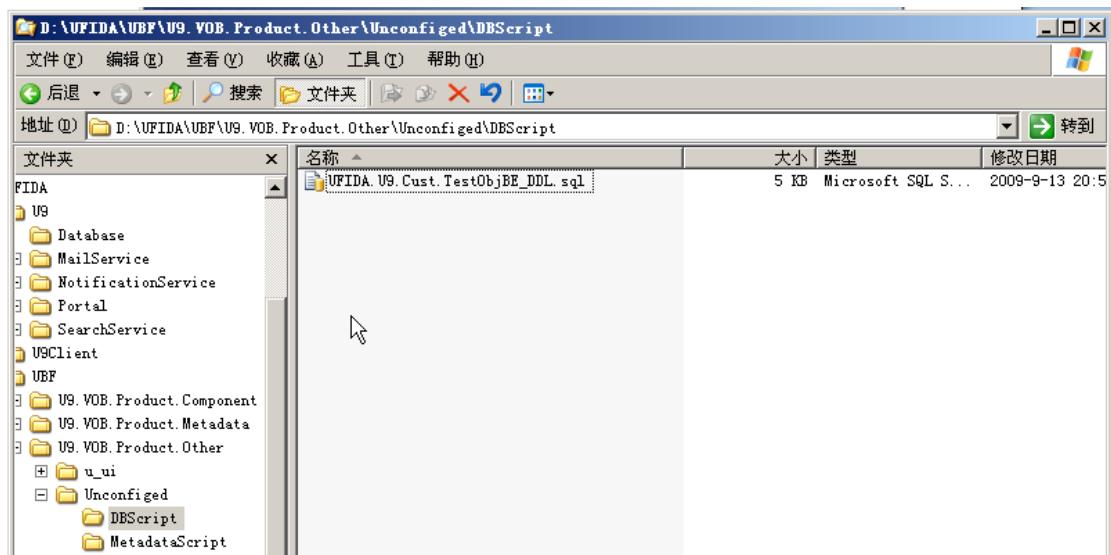




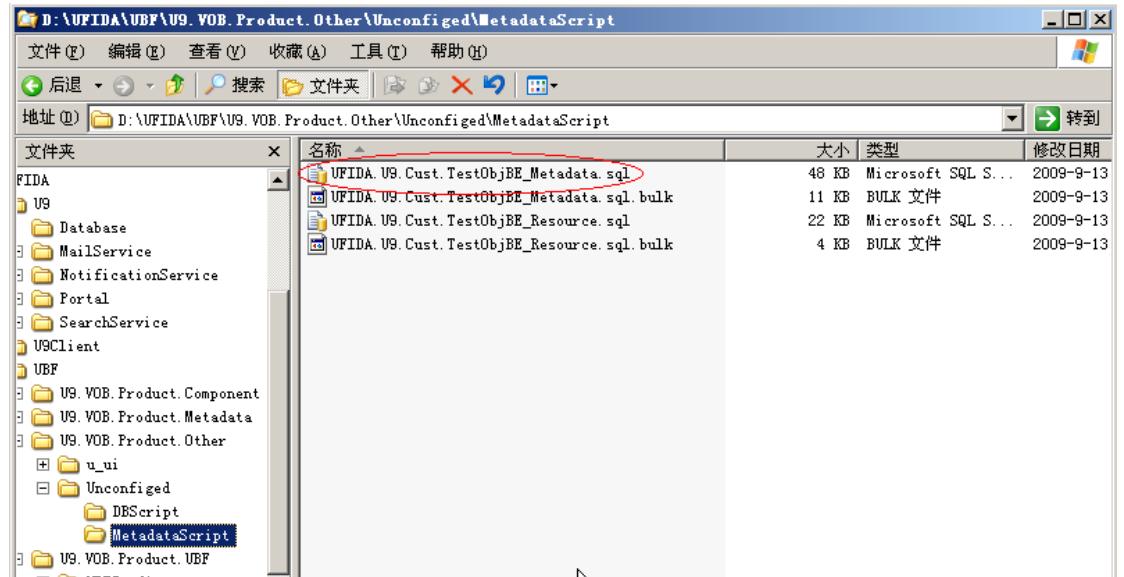
5) 执行数据库脚本

BE 组件有 3 个，UI 组件有 2 个。

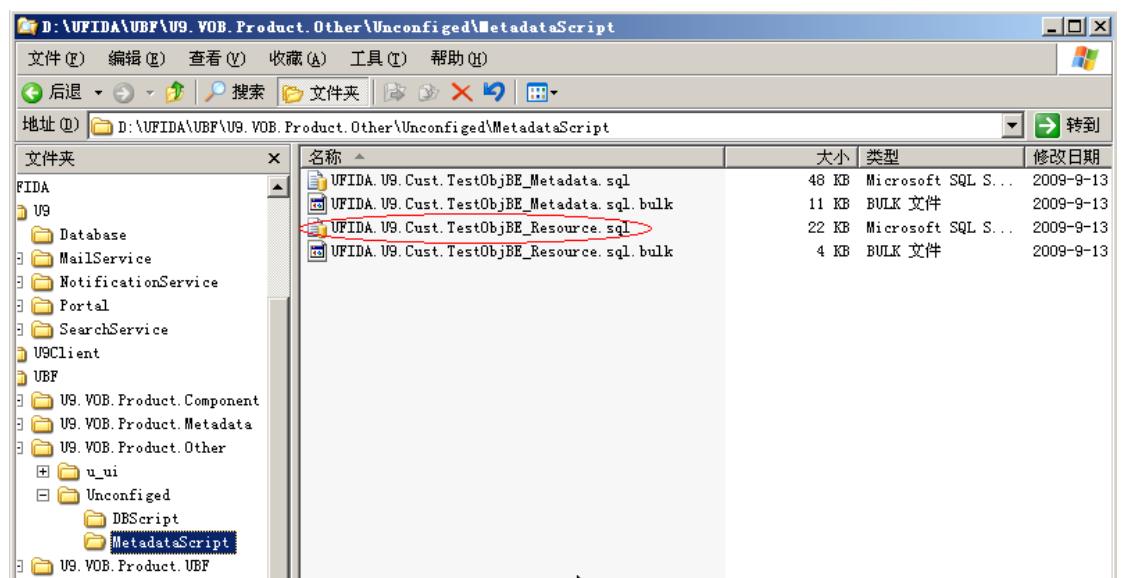
- 建库脚本：D:\UFIDA\UBF\U9.VOB.Product.Other\Unconfiged\DBScript



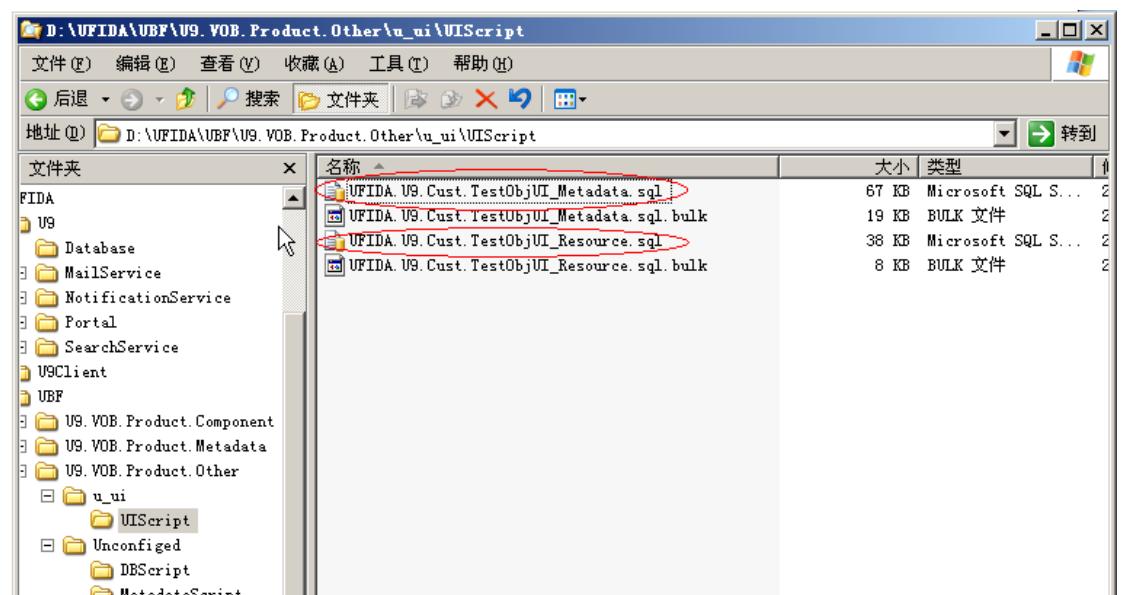
- 元数据脚本：



● 资源脚本：



UI 涉及的脚本：



● 枚举脚本：

若开发的单据中涉及枚举值的定义，则需执行 SQL 语句：

D:\UFIDA\U9\Portal\ApplicationServer\Libs\SqlScript\UBF\\\DataUpdate\MetadataInitNew.sql。

注意：该语句中 `insert into ubf_sys_extenumtype_tr` 的相关条件类型不匹配，需作修改才能执行成功。

如：`convert(nvarchar(100),a.[ID]) = c.resourcename`

```
delete from ubf_sys_extenumtype
delete from ubf_sys_extenumtype_trl
delete from ubf_sys_extenumvalue
delete from ubf_sys_extenumvalue_trl
GO -- 注: v2.1多一个isrefview字段
insert into ubf_sys_extenumtype select local_id as
id,'20060612','u9zhangheng','20060612','u9zhangheng',0,fullname as code,isextend,id as uid,0 as
isrefview from ubf_md_class where classtype=3
insert into ubf_sys_extenumtype_trl select a.local_id as id,'zh-CN',b.[displayname] as
[name],b.[Description] from ubf_md_class a left join ubf_RES_resourcevalue b on
convert(nvarchar(100),a.[ID]) = b.resourcename where classtype=3
insert into ubf_sys_extenumvalue select a.local_id as
id,'20060612','u9zhangheng','20060612','u9zhangheng',0,a.[Local_Class_ID] as
ExtEnumType ,a.[name] as code,a.defaultvalue as evalue,issystem,md_class_id as ExtEnumTypeUID
from ubf_md_attribute as a inner join ubf_md_class as b on a.md_class_id=b.id where classtype=3
insert into ubf_sys_extenumvalue_trl select a.local_id as id,'zh-CN',c.[displayname] as [name]
from ubf_md_attribute as a inner join ubf_md_class as b on a.md_class_id=b.id left join
ubf_RES_resourcevalue c on convert(nvarchar(100),a.[ID]) = c.resourcename where classtype=3
GO
```

--另一种方法

```
exec p_ReBuildExtEnum;
exec UpdateMetaDataTable;
```

五、运行使用

1、设置菜单权限



2、运用

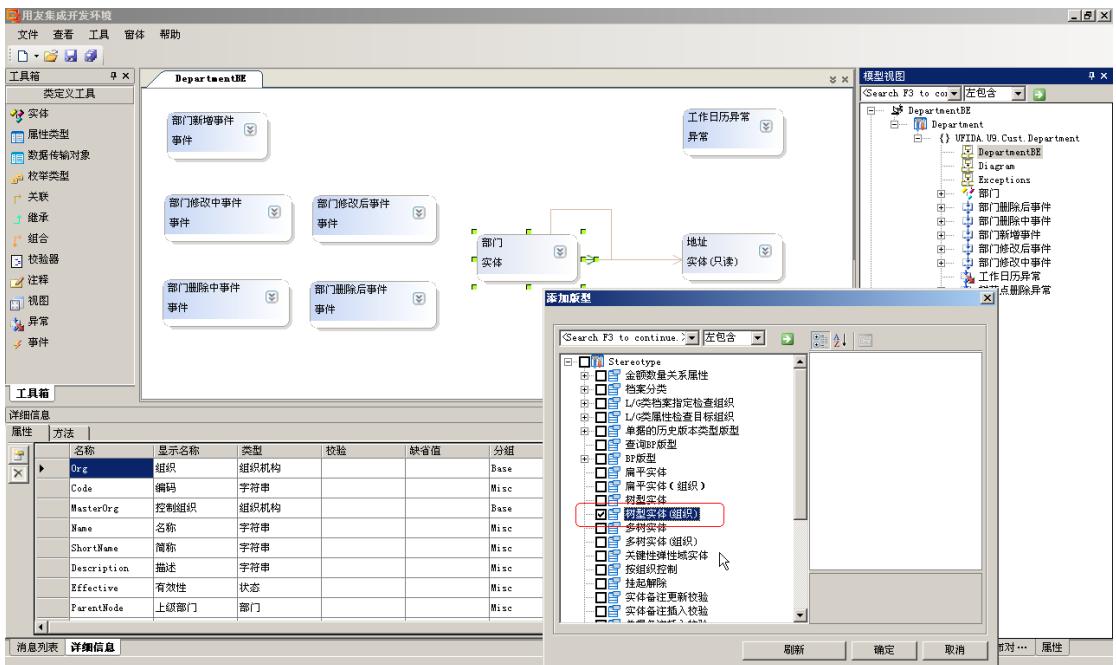


六、实现级次编码规则

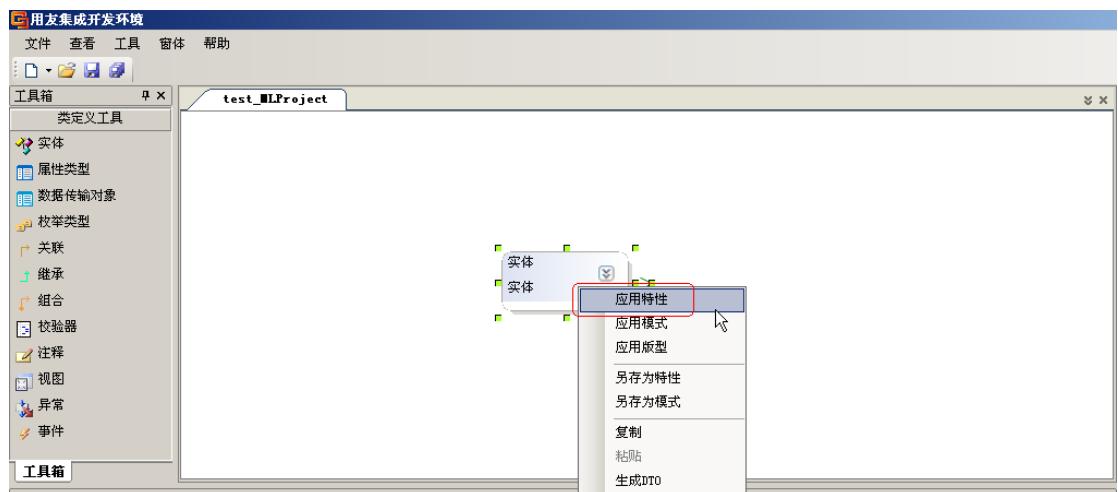
实现形如部门档案编码规则中有上下级关系的效果。

1、BE中应用版型

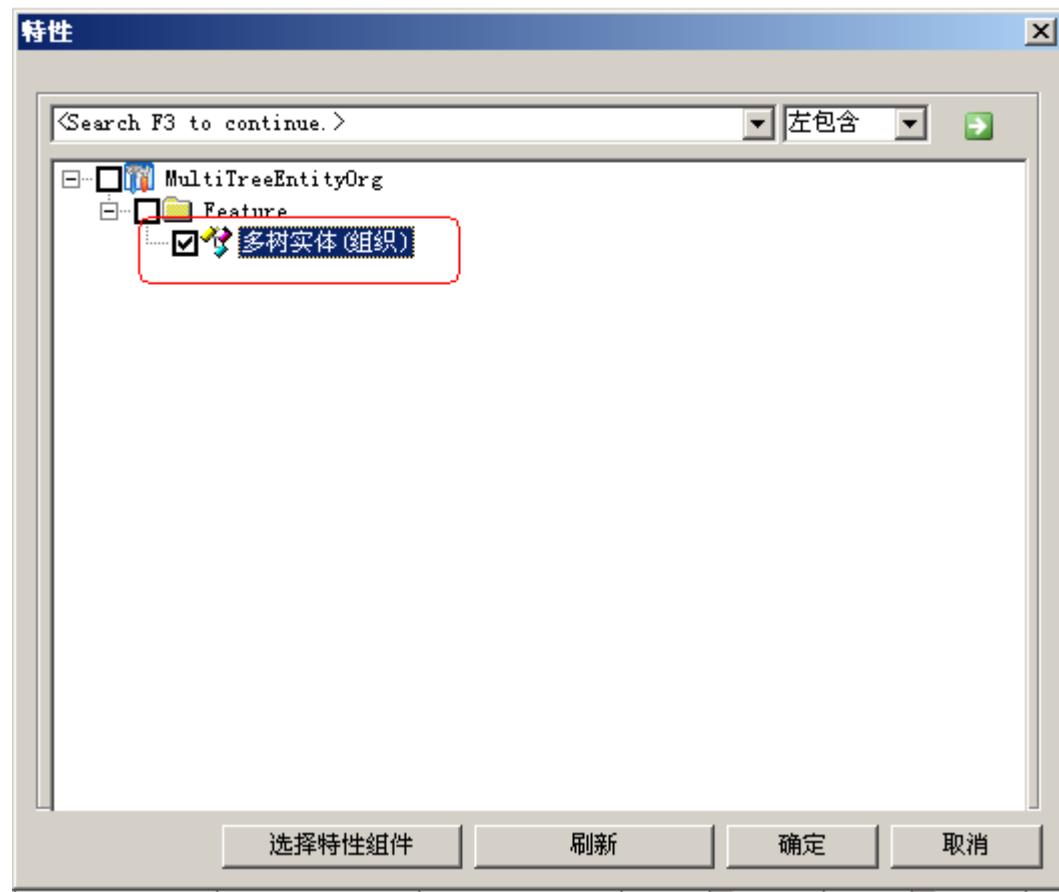




2、应用特性



选择 D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\Feature\MultiTreeEntityOrg\
MultiTreeEntityOrg.ubffeproj



修改有效期为有效性（数据类型为“状态”）。

The screenshot shows the 'Detailed Information' dialog box for the 'MultiTreeEntityOrg' entity. It has tabs for '属性' (Properties) and '方法' (Methods), with '属性' selected. On the left is a toolbar with icons for adding, deleting, and saving. The main area is a table of properties:

	名称	显示名称	类型	校验
	MasterOrg	控制组织	组织机构	
	Name	名称	字符串	
	ShortName	简称	字符串	
	Description	描述	字符串	
▶	Effective	有效性	状态	
	ParentNode	上级部门	部门	
	Level	级别	整数	
	IsLeaf	末级部门	布尔	

3、 BE代码

```
protected override void OnSetDefaultValue()
{
    this.TreeEntityOrg_SetDefaultValue();
    base.OnSetDefaultValue();
}

protected override void OnDeleting() {
```

```

        EntityDataQuery query = Finder.CreateDataQuery();
        query.Select(new string[] { "count(ID)" });
        query.Parameters.Add(new OqlParam(this.ID));
        if (Convert.ToInt32(query.FindValue("ParentNode = @ID")) > 0)
        {
            throw new UFIDA.U9.Cust.Department.ParentDeleteException(this.Code,
this.Name);
        }
        base.OnDeleting();
    }

private static Department CreateDefault_Extend()
{
    //TODO delete next code and add your custome code here
    //throw new NotImplementedException() ;
    UFIDA.U9.Cust.Department department = null;
    department =Department.Finder.FindByBusinessKey(Base.Context.LoginOrg, "001");
    if (department == null)
    {
        using (ISession session = Session.Open())
        {
            department = Create();
            department.Org = Context.LoginOrg;
            department.Code = "001";
            department.Name = "department";
            department.Effective.EffectiveDate = DateTime.Now.Date;
            session.Commit();
        }
    }
    return department;
}

private static Department CreateDefaultComponent_Extend()
{
    //TODO delete next code and add your custome code here
    //throw new NotImplementedException() ;
    UFIDA.U9.Cust.Department department = null;
    using (ISession session = Session.Open())
    {
        department = CreateDefault();
        session.Commit();
    }
    return department;
}

public UFIDA.U9.Base.WorkCalendar.WorkCalendar GetWorkCalendar()
{
    if (this.WorkCalendar != null)

```

```

    {
        return this.WorkCalendar;
    }

    if (this.ParentNode != null) // 有上级部门
    {
        return this.ParentNode.GetWorkCalendar();
    }

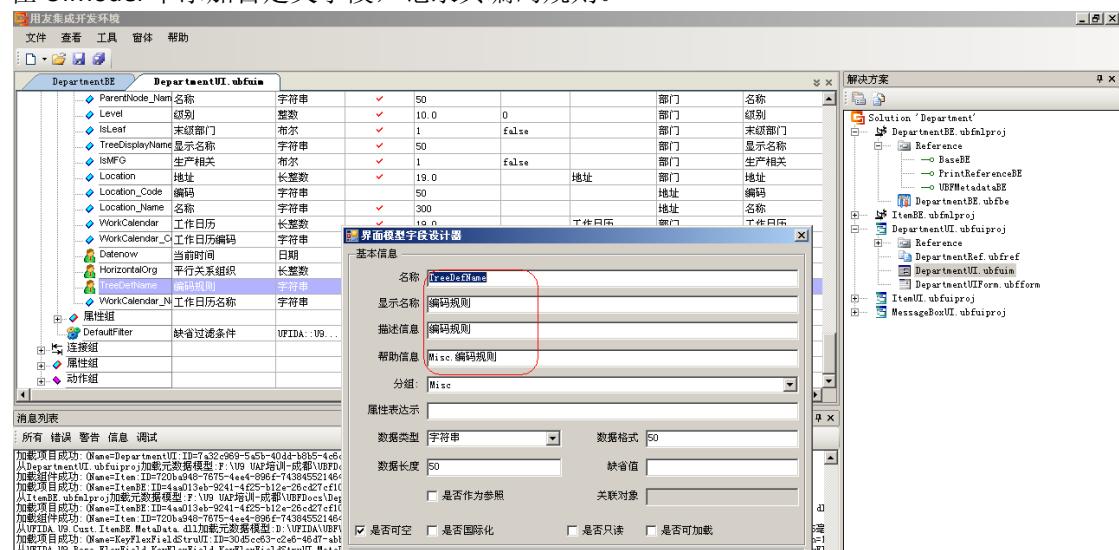
    return this.Org.WorkCalendar; // 无上级部门, 查找对应组织工作日历
}

// 在当前组织查找
public static UFIDA.U9.Cust.Department Department FindByCode(string code)
{
    return FindByCode(Base.Context.LoginOrg, code);
}

```

4、UIModel设置

在 UIModel 中添加自定义字段，记录其编码规则。



5、 UIForm设置



6、 UI代码

*UIModelActionExtend.cs:

```
private void SaveClick_Extend(object sender, UIEventArgs e)
{
    DepartmentRecord focusedRecord = this.CurrentModel.Department.FocusedRecord;
    if (focusedRecord != null)
    {
        if (!string.IsNullOrEmpty(focusedRecord.Name))
        {
            focusedRecord.Name = focusedRecord.Name.Trim();
        }
        if (!string.IsNullOrEmpty(focusedRecord.Code))
        {
            focusedRecord.Code = focusedRecord.Code.Trim();
        }
    }
    this.SaveClick_DefaultImpl(sender, e);
}

private void CopyClick_Extend(object sender, UIEventArgs e)
{
    this.CopyClick_DefaultImpl(sender, e);
```

```
DepartmentRecord focusedRecord = this.CurrentModel.Department.FocusedRecord;
if (focusedRecord != null)
{
    focusedRecord.Code = string.Empty;
    focusedRecord.MasterOrg = null;
}
}

private void OnLoadData_Extend(object sender, UIActionEventArgs e)
{
    string oql = this.CurrentModel.Department.CurrentFilter.OPath;
    if (NavigateManager.IsTitleLink(this.CurrentPart) &&
this.CurrentPart.NameValues["ID"] != null && this.CurrentPart.NameValues["ID"] != "")
    {
        this.CurrentModel.Department.CurrentFilter.OPath="ID=" +
this.CurrentPart.NameValues["ID"].ToString();
    }
    this.OnLoadData_DefaultImpl(sender, e);
    if (this.MainView.Records.Count == 0)
    {
        this.NewClick_Extend(sender, e);
    }
    this.CurrentModel.Department.CurrentFilter.OPath = oql;
    this.OnLoadData_DefaultImpl(sender, e);
}

*UIModelModelExtend.cs:
public override void AfterInitModel()
{
    this.Department.PageStrategy.PageSize=1;

this.Department.FieldEffective_EffectiveDate.DefaultValue=PlatformContext.Current.DateTime;
    this.Department.FieldOrg.DefaultValue=PlatformContext.Current.OrgID;

    // 返回当前组织默认的工作日历
    if (!string.IsNullOrEmpty(PlatformContext.Current.OrgID))
    {
        GetOrgWorkCalendarInfoProxy proxy = new GetOrgWorkCalendarInfoProxy();
        proxy.Org=long.Parse(PlatformContext.Current.OrgID);
        WorkCalendarInfoDTOData data = proxy.Do();
        if (data != null)
        {
            this.Department.FieldWorkCalendar.DefaultValue=data.ID;
            this.Department.FieldWorkCalendar_Code.DefaultValue=data.Code;
        }
    }
}
```

```
        this.Department.FieldWorkCalendar_Name.DefaultValue=data.Name;
    }
}
}

*UIFormWebPartCodeBehindExtend.cs:
public void BeforeUIModelBinding()
{
    this.BtnCancel.IsClientClose=true;
    // 返回部门的编码规则
    GetProfileValueProxy proxy = new GetProfileValueProxy();
    proxy.ProfileCode="DepartmentTD"; // proxy.set_ProfileName("DepartmentTD");
    PVDTOData data = proxy.Do();
    if (data != null)
    {
        DepartmentRecord focusedRecord = this.Model.Department.FocusedRecord;
        if (focusedRecord != null && !string.IsNullOrEmpty(data.ProfileValue))
        {
            focusedRecord.TreeDefName = data.ProfileValue;
        }
    }
    this.Code76.Focus(); // 编码字段
}

public void AfterUIModelBinding()
{
    this.SetToolbarState();
    this.SetButtonState();
    this._BuilderControl_ServerExcute_Trigger0();           // 关联控件
    this.FlexFieldPicker0.Enabled=IsDescFlexFieldDefined();
}

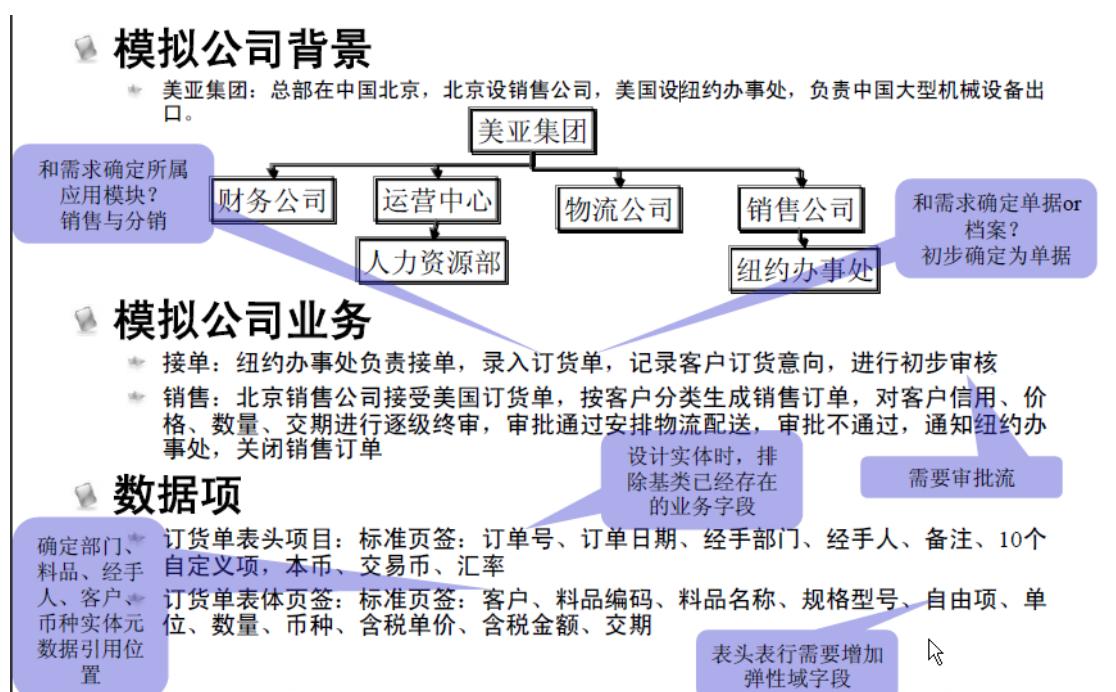
private void SetToolbarState()
{
    IUIRecord record = Action.MainView.FocusedRecord;
    if ((record != null) && (record.DataRecordState == DataRowState.Added))
    {
        ButtonManger.SetToolBarNewStatus(this.Toolbar2, 0);
    }
    else
    {
        ButtonManger.SetToolBarModifyStatus(this.Toolbar2, 0);
    }
}
```

```
if ((record["MasterOrg"] == null) && record["Org"].ToString() ==  
PDContext.Current.OrgID) // if ((record.get_Item("MasterOrg") == null) &&  
(record.get_Item("Org").ToString() == PDContext.get_Current().get_OrgID()))  
  
{  
    this.Toolbar2.Enabled=true;  
    this.Card2.Enabled=true;  
    this.DDBtnTaskLink.Enabled=true;  
}  
else  
{  
    this.Toolbar2.Enabled=false;  
    this.Card2.Enabled=false;  
    this.DDBtnTaskLink.Enabled=false;  
}  
}  
  
private void SetButtonState()  
{  
    if (NavigateManager.IsTitleLink(this) || NavigateManager.IsModelPopup(this))  
    {  
        this.BtnOk.Visible=true;  
        this.BtnCancel.Visible=true;  
        this.BtnList.Enabled=false;  
        //this.MenuR1.Enabled=false;  
    }  
    else  
    {  
        this.BtnOk.Visible = false;  
        this.BtnCancel.Visible=false;  
        this.BtnList.Enabled=true;  
    }  
}  
  
private bool IsDescFlexFieldDefined()  
{  
    UFIDA.U9.Base.FlexField.DescFlexField.Proxy. IsDescFlexFieldDefExistedProxy proxy  
= new UFIDA.U9.Base.FlexField.DescFlexField.Proxy. IsDescFlexFieldDefExistedProxy();  
    proxy.EntityFullName = this.Action.MainView.EntityFullName;  
    return proxy.Do();  
}  
  
#region 关联控件  
private void _BuilderControl_ServerExcute_Trigger0()
```

```
{  
    try  
    {  
        if (!bool.Parse(this.Effective_IsEffective92.Checked.ToString()))  
        {  
            // 直接给客户端控件赋值  
            this.Effective_EffectiveDate31.Enabled=false;  
            this.Effective_EffectiveDate31.ReadOnly=true;  
            this.Effective_DisableDate34.ReadOnly=true;  
            this.Effective_DisableDate34.Enabled=false;  
        }  
        if (bool.Parse(this.Effective_IsEffective92.ToString()))  
        {  
            this.Effective_EffectiveDate31.Enabled=true;  
            this.Effective_EffectiveDate31.ReadOnly=false;  
            this.Effective_DisableDate34.Enabled=true;  
            this.Effective_DisableDate34.ReadOnly=false;  
        }  
    }  
    catch (Exception)  
    {  
    }  
}  
#endregion
```

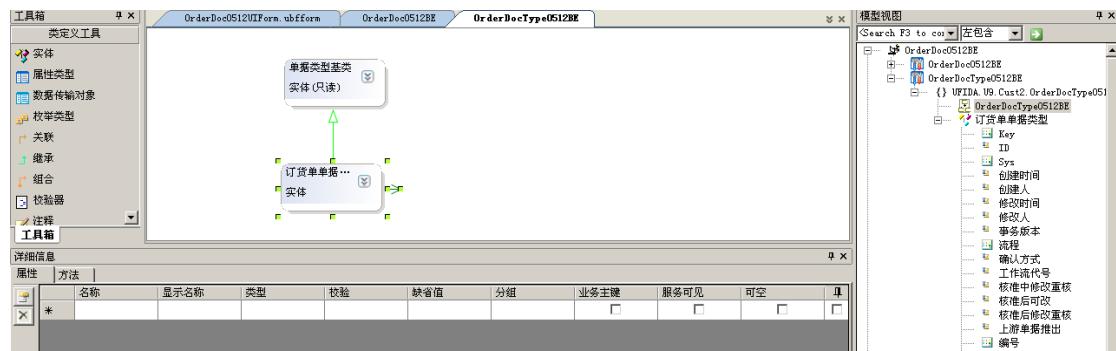
第三章 单据开发（初级）

一、 开发需求



二、 创建BE

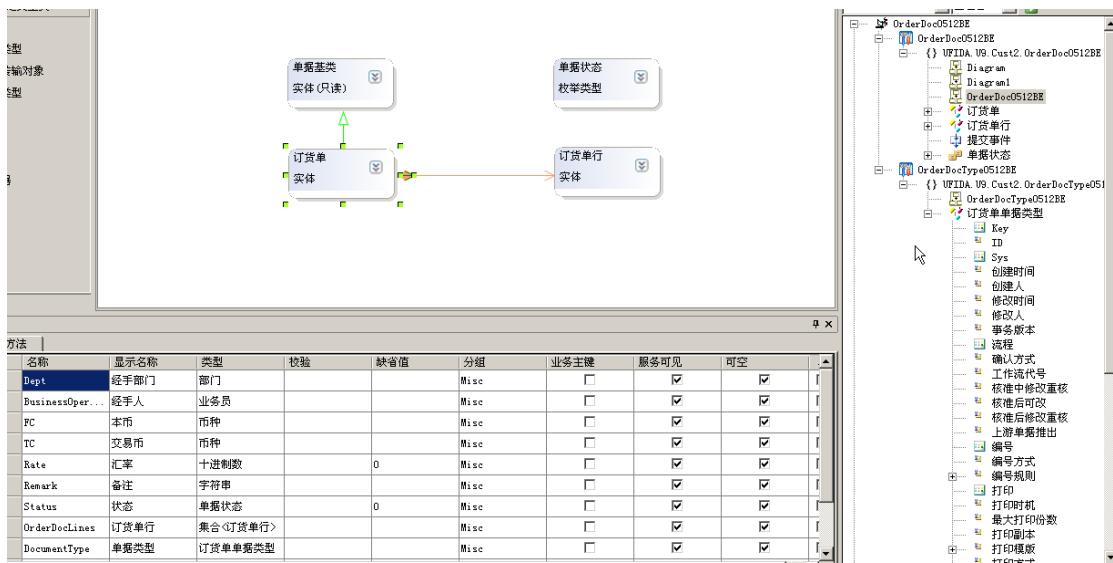
1、 建立单据类型BE（OrderDocType）



注：必须继承“单据类型基类”。即将 DocType 拖放至窗体上。

2、 建立单据BE

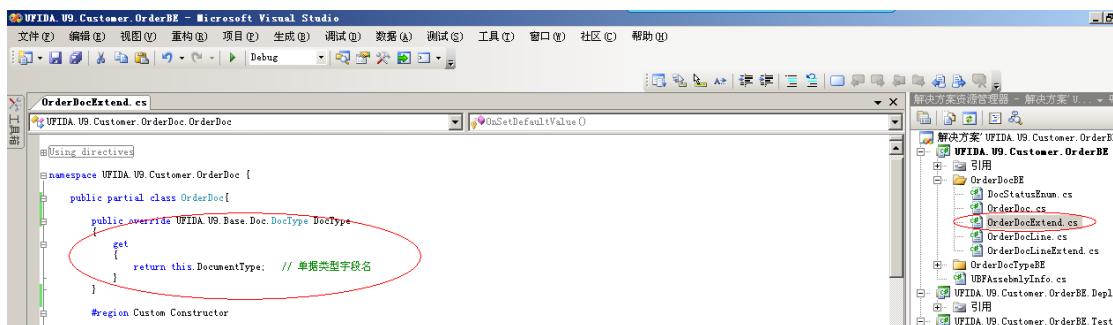
1) 订货单头 (OrderDoc)



注：必须继承单据基类 DocBE（在 UBF 下 ufida.u9.base.Doc.DocBE）。

U9 单据构造问题，BE 实体的表头语句后加入如下语句：OrderDocExtend.cs

```
public override UFIDA.U9.Base.Doc.DocType DocType
{
    get
    {
        return this.DocumentType; // 单据类型字段名
    }
}
```



2) 订货单体(OrderDocLine)



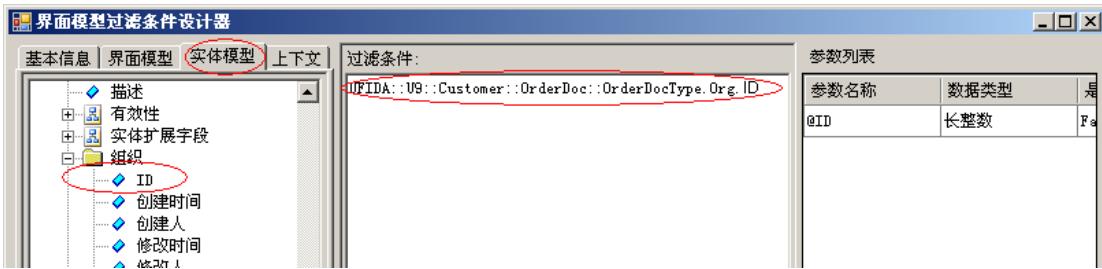
三、 创建单据类型UI

1、 UIModel

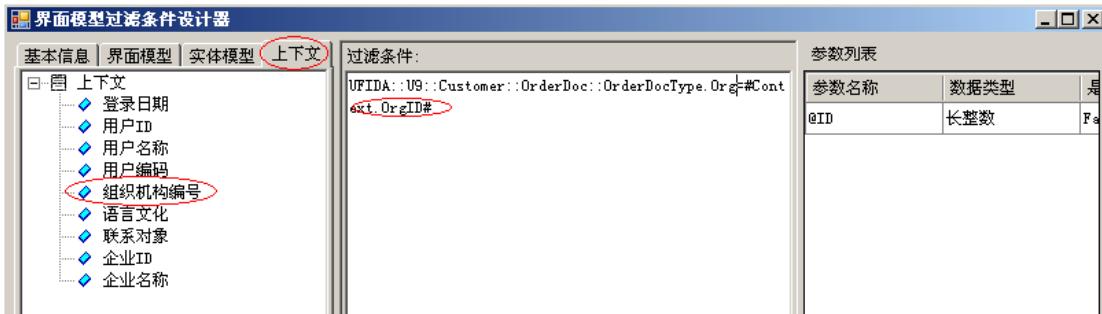
① 在单据类型的 UIModel 缺省条件中设置参数 ID（需右键增加）



② 设置过滤条件：通常按当前组织进行过滤。

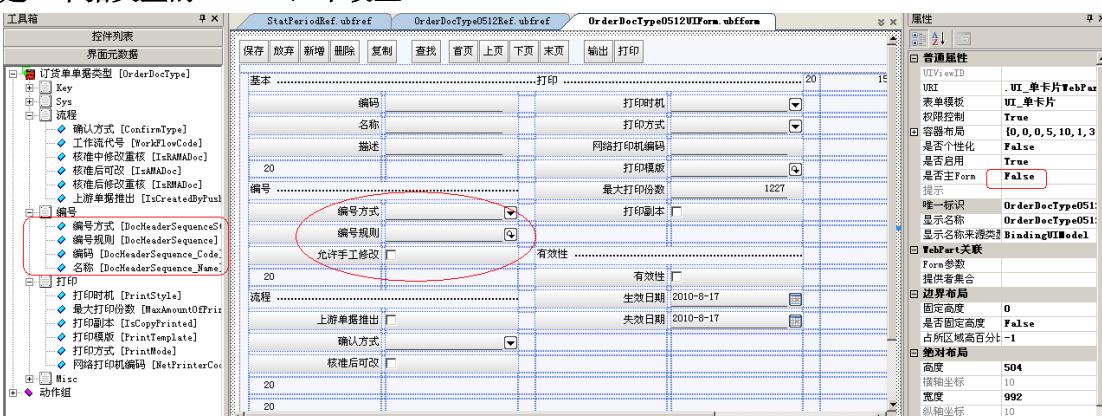


并将.ID 删掉，以避免 LEFT 关联，加快运行效率。



2、UIForm (单卡)

① 建立单据类型的 UIForm，不设主 Form。

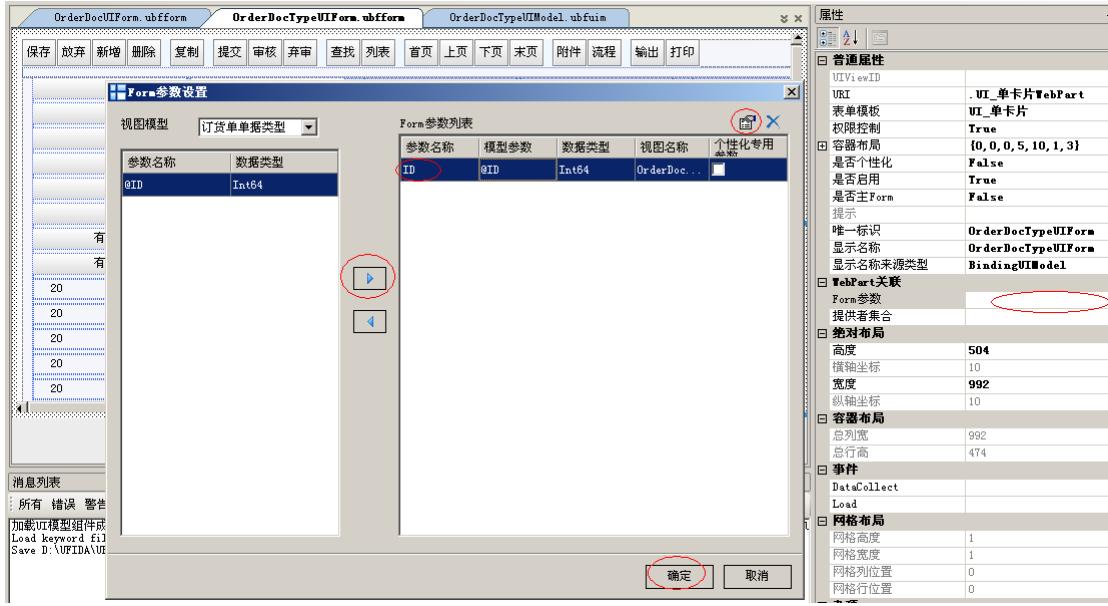


注：编号方式、编号规则等不在 Misc 分组中，而在编号分组中选择。

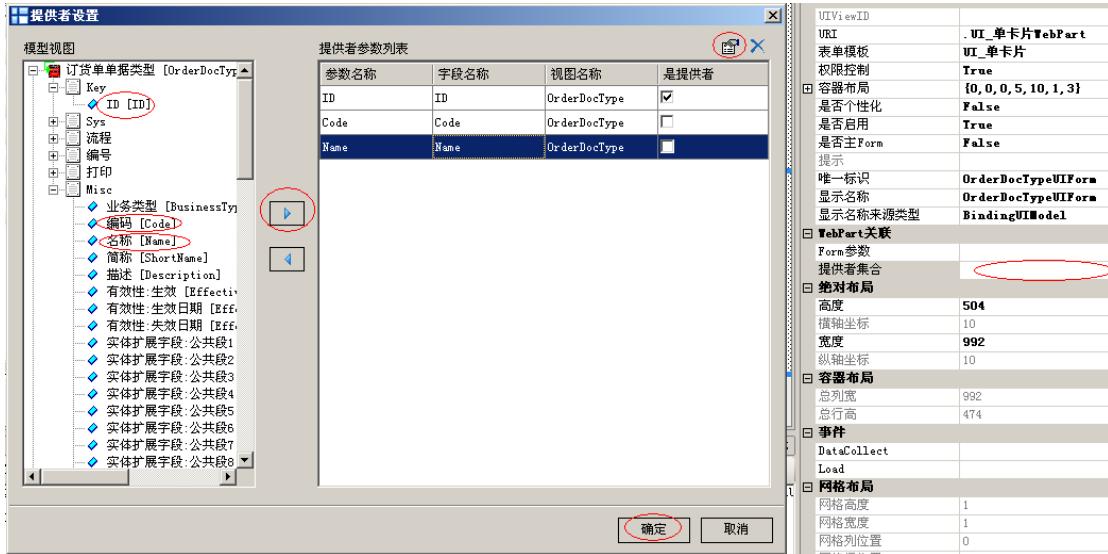
② 设置单据编号规则的 Form 引用。

需先引用 D:\UFIDA\UBFV2\U9.VOB.Product.Metadata\UI\下的
UFIDA.U9.Base.Sequence.SequenceUI.MetaData.dll。

③ 在单据类型参照的 UIForm 中定义传入参数。

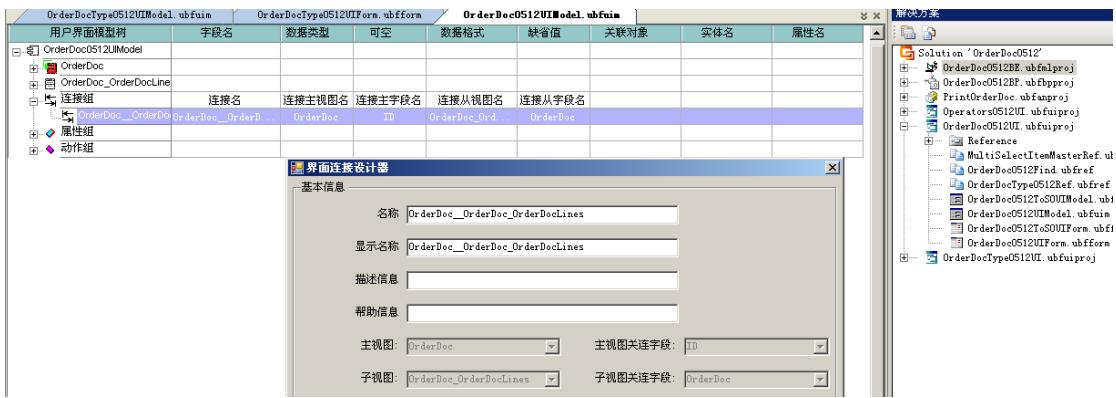


定义提供者集合:



四、 创建单据UI

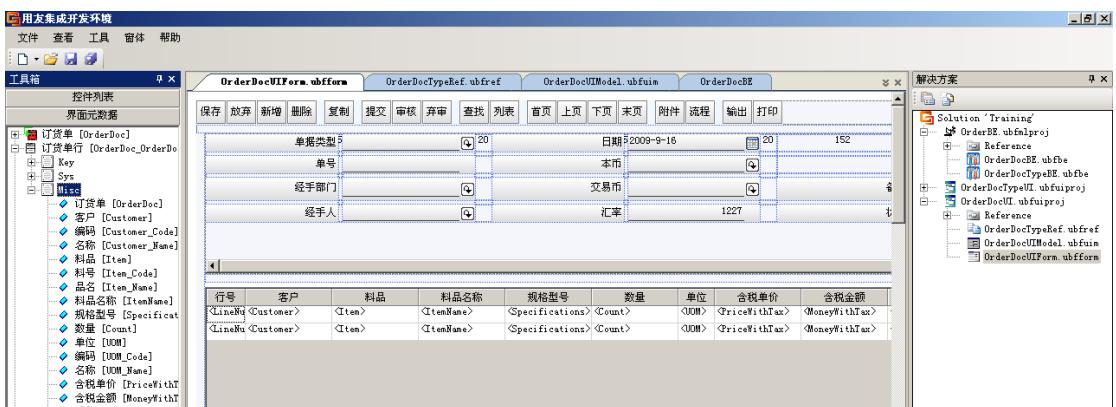
1、 UIModel



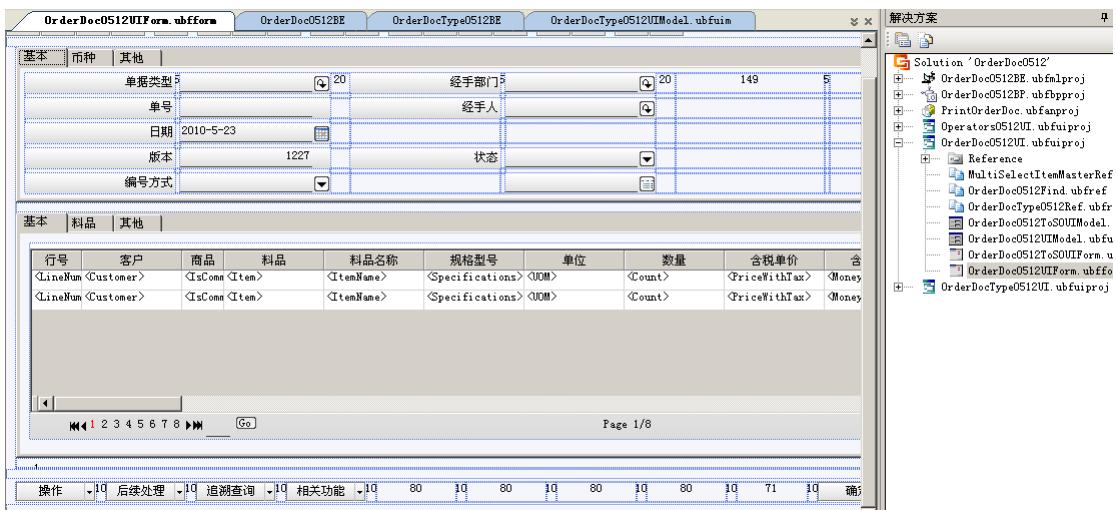
注：需先拖放主表，然后在主表中找对应子表，系统会自动建立关联，否则只能手工建立。

2、 UIForm

1) 单卡单表样式

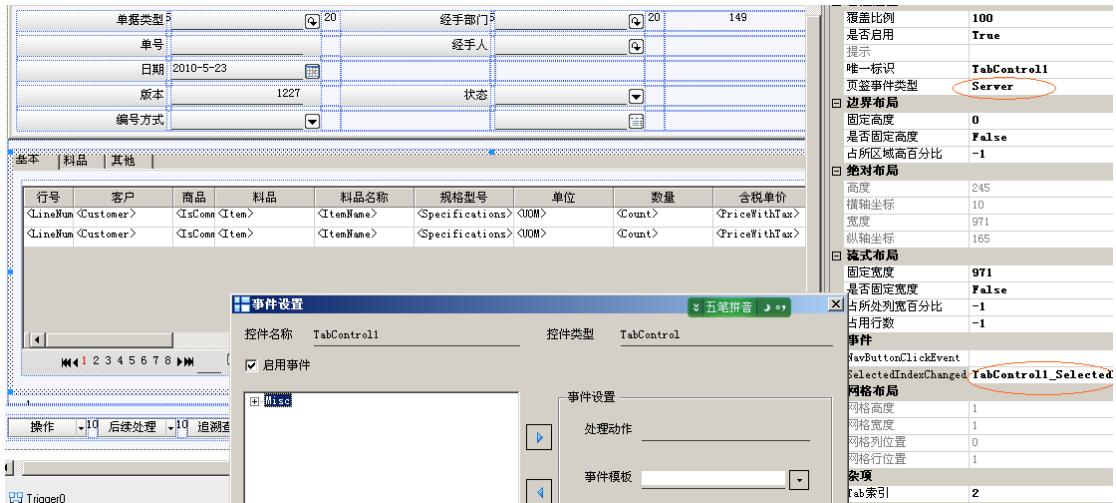


2) 多卡多表样式

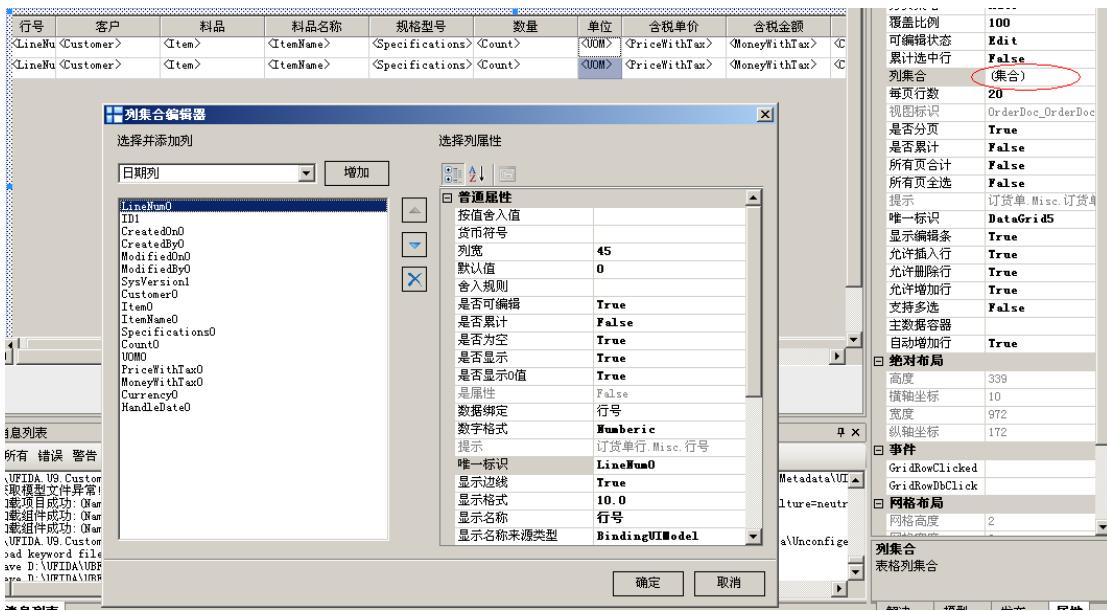


注：

- ① 多卡中，各页签均需绑定 ID，且需将页签事件类型设为 Server，并启用页事件。



- ② 若需修改表格列的宽度或设置参照等，均在列集合中进行。



其参照详见参照及列表开发部分。

五、 构造、发布和装配

同档案开发，略。

六、 其他应用

1、 单据自动编号

- ① 定义单据编码规则（编码规则需进行 Form 引用，
UFIDA.U9.Base.Sequence.SequenceUI.MetaData.dll）

- ② 定义单据类型中选择 SD013 编码规则。

设置单据编号规则:

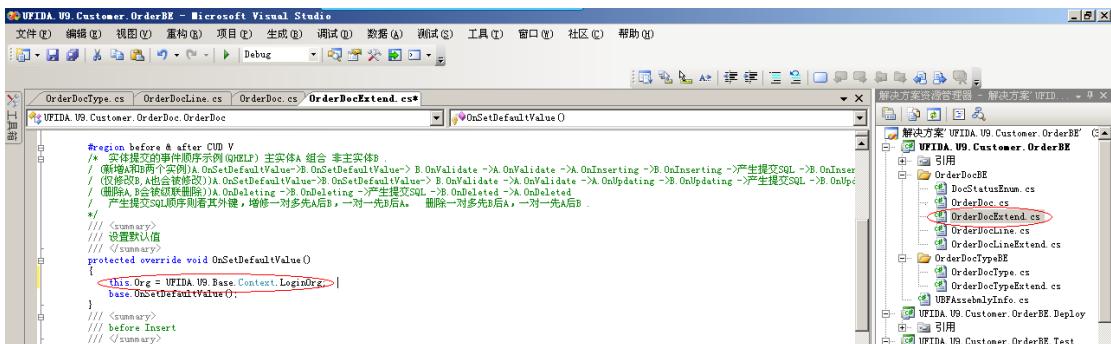
```
Protected override void OnSetDefaultValue() {
    Base.OnSetDefaultValue();
    ...
    this.GetDocNos();
}
```

2、给单据赋默认值

● BE 中赋默认值

需要在 BE 实体（单据和单据类型）的 Extend 上设置。如设置当前单据所属组织:

```
this.Org = UFIDA.U9.Base.Context.LoginOrg;
```



● UI 中赋默认值

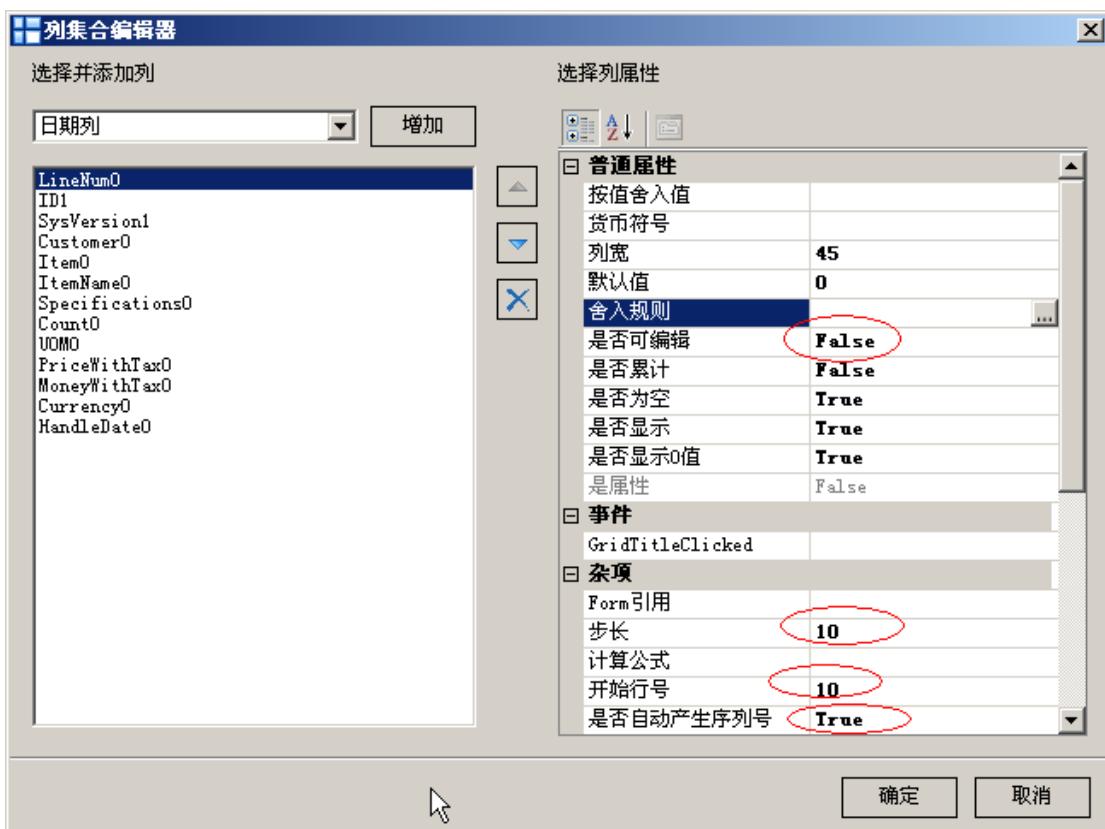
```
UFSoft.UBF.Util.Context.PlatformContext.Current.OrgID;
this.Model.CarHonner.FieldCarNo.DefaultValue=this.CurrentState["CarNo"]; // 赋车号默认值
this.DataGrid0.Columns["CarNo"].Enabled = false; // 设置列不允许编辑
```

3、 支持单据个性化功能

```
//引用 UFIDA.U9.UI.PDHelper.dll    /ubfstudio/runtime/
public void AfterCreateChildControls()
{
    UFIDA.U9.UI.PDHelper.PersonalizationHelper.SetPersonalizationEnable(this, true);
}
```

4、 表体自动行号

在 UIForm 中设置行号的列属性为自动生成序列号。



若要从系统参数中读取行号信息，其设置信息可通过以下 SQL 查询得到：

```
select Code, Name, Description, * from Base_Profile as a left join Base_Profile_Trl as b
on a.ID=b.ID where code='syslineno'
```

代码中调用 BP 得到参数信息，并将其添加到 *WebPartExtend.cs 的 AfterCreateChildControls 方法中，如：

```
private void InitGridLineNo()
{
    UFIDA.U9.Base.Profile.Proxy.GetProfileValueProxy bpObj = new GetProfileValueProxy();
    bpObj.ProfileCode = "Syslineno";
    UFIDA.U9.Base.Profile.PVTDOData pVTDOData = bpObj.Do();
    int LineNoStep = Convert.ToInt32(pVTDOData.ProfileValue); // 取得行号参数值
    this.Model.OrderDoc_OrderDocLines.FieldLineNum.DefaultValue = LineNoStep; // 修
```

改行号递增值，以备后续使用

```
//行GRID  
((UFSoft.UBF.UI.ControlModel.IAutoRowNo)this.DataGrid9.Columns[2]).Sequence = true;  
((UFSoft.UBF.UI.ControlModel.IAutoRowNo)this.DataGrid9.Columns[2]).SequenceStep = LineNoStep;  
((UFSoft.UBF.UI.ControlModel.IAutoRowNo)this.DataGrid9.Columns[2]).SequenceStart = LineNoStep;  
}
```

-
- 取得下一行号值 GetNewLineNo()

```
//int? lineNoStep = lineRecord.LineNum; // 该方法会导致行号翻倍  
//lineRecord.LineNum = GetNewLineNo(lineNoStep.HasValue ? lineNoStep.GetValueOrDefault() : 10);  
public int GetNewLineNo(int step)  
{  
    int maxLineNo = GetMaxLineNo();  
    maxLineNo +=  
Convert.ToInt32(this.Model.OrderDoc_OrderDocLines.FieldLineNum.DefaultValue);  
    return maxLineNo;  
}
```

```
public int GetMaxLineNo()  
{  
    int docLineNo = 0;  
    if (this.Model.OrderDoc_OrderDocLines.Records == null)  
    {  
        return 0;  
    }  
    foreach (OrderDoc_OrderDocLinesRecord record in  
this.Model.OrderDoc_OrderDocLines.Records)  
    {  
        if (record.LineNum > docLineNo)  
        {  
            docLineNo = (int) record.LineNum; // ?? 10;  
        }  
    }  
    return docLineNo;  
}
```

- 按递增值批改行号

```
internal void SetLinesAutoNum(P2PSendDocRecord record)  
{  
    P2PSendDoc_TransItemsRecord line;  
    //设置订单行默认值  
    IUIRecord[] lines = this.GetEffectiveChildRecords(record,  
this.CurrentModel.P2PSendDoc_TransItems);  
    //int sysLineNoStep = this.SysLineNoStep;
```

```
if (lines.Length == 0)
{
    return;
}

int sysLineNoStep = 10;
sysLineNoStep = this.GetSysLineNoByBp();

if (lines.Length == 1)
{
    line = lines[0] as P2PSendDoc_TransItemsRecord;
    line.LineNum = sysLineNoStep;
}
else
{
    //第1行的
    if (Convert.ToInt32(lines[0]["LineNum"].ToString().Trim()) <= 0)
    {
        lines[0]["LineNum"] = sysLineNoStep;
    }

    //大于1的行
    for (int i = 1; i < lines.Length; i++)
    {
        line = lines[i] as P2PSendDoc_TransItemsRecord;
        if (line.LineNum <= 0)
        {
            line.LineNum = (int)lines[i - 1]["LineNum"] + sysLineNoStep;
        }
    }
}
```

5、 实体扩展字段

1) 卡片处理

- ① 拖一个弹性域控件，并设置其类型为**Description**。
- ② 在AfterCreateChildControls()里面调用：

FlexFieldHelper.SetDescFlexField(this.FlexFieldPicker0, this.Model.Territory);
其中：`this.FlexFieldPicker0`为要设置的描述性弹性域控件，`this.Model.Territory`为描述

性弹性域所在的View，把红色的Territory换成你的View就可以了。

2) Grid处理

- ① 在Grid的末尾加一文本列，不需要绑定任何字段，可置其显示名称来源为Custom。
- ② 在AfterCreateChildControls()里面调用：

```
FlexFieldHelper.SetDescFlexField(this.DataGrid3, 2);
```

其中：this.DataGrid3 为要设置的描述性弹性域的 Grid，2 为描述性弹性域列（即上面添加的文本列）在 Grid 中的列索引。

3) 注意事项

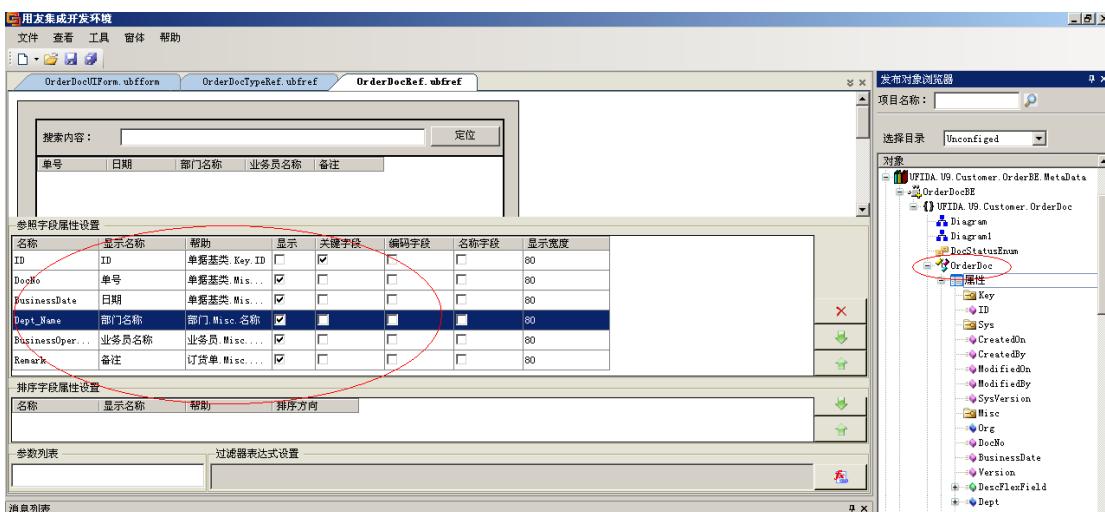
如果一个Form只有一处用到描述性弹性域，可以使用上面的方法，但是如果一个Form有多处用到描述性弹性域（包括卡片和列表），请一定用下面的方式，提高效率：

```
FlexFieldHelper.SetDescFlexField(DescFlexFieldParameter[] parameters);
```

其中parameters为整个Form上所有要设置的描述性弹性域的相关参数的数组（包括卡片和列表），new DescFlexFieldParameter的时候请使用你想要的构造函数构造实例。

6、单据查找

① 建立单据查询(OrderDocRef)



注：若要区别不同组织，需定义查询过滤器：Org=#Context.OrgID#。

② 代码实现单据查询

```
private void BtnFind_Click_Extend(object sender, EventArgs e)
{
    this.Action.NavigateAction.MovePageAt(null, long.Parse(wpFindID.Value));
    BtnFind_Click_DefaultImpl(sender, e);
}
```

```

private void BtnFind_Click_Extend(object sender, EventArgs e)
{
    //调用组件提供的默认实现，一键默认可能会调用相应的Action
    this.Action.NavigateAction.MovePageAt(null, long.Parse(wpFindID.Value));
    BtnFind_Click_DefaultImpl(sender, e);
}

public void AfterCreateChildControls()
{
    //取得提示信息资源，是否删除当前记录，需using UFIDA.U9.UI.PDHelper
    string message = PDResource.GetDeleteConfirmInfo();
    PDFormMessage.ShowDelConfirmDialog(this.Page, message, "", this.BtnDelete);

    //取得当前卡片参照的属性变量：FormID、Width、Height、Title;
    //传递隐藏域wpFindID的客户端ID; 注意：隐藏域wpFindID会记录参照选择的记录ID;
    //获取OQL
    //string paramWhere = "SHIP_OPATH=" + Action.GetOPathByShipType();
    string paramWhere = null;
    UFIDA.U9.UI.PDHelper.PDFormMessage.ShowConfirmDialog(this.Page,
        "0e6fe785-2624-47d6-bb5a-d1561ebc046a", "580", "408", Title, wpFindID.ClientID,
        this.BtnFind, paramWhere);
}

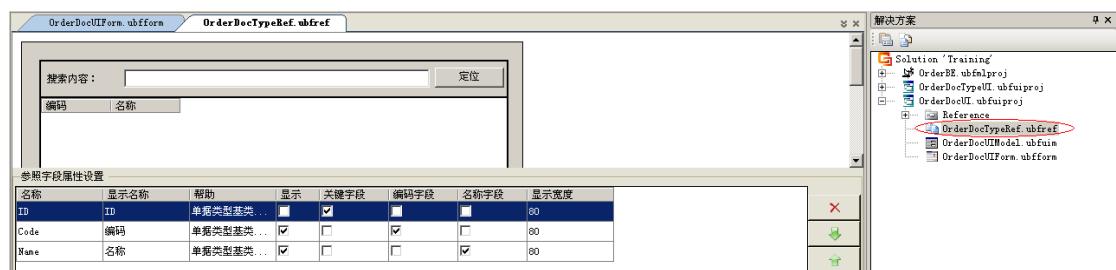
```

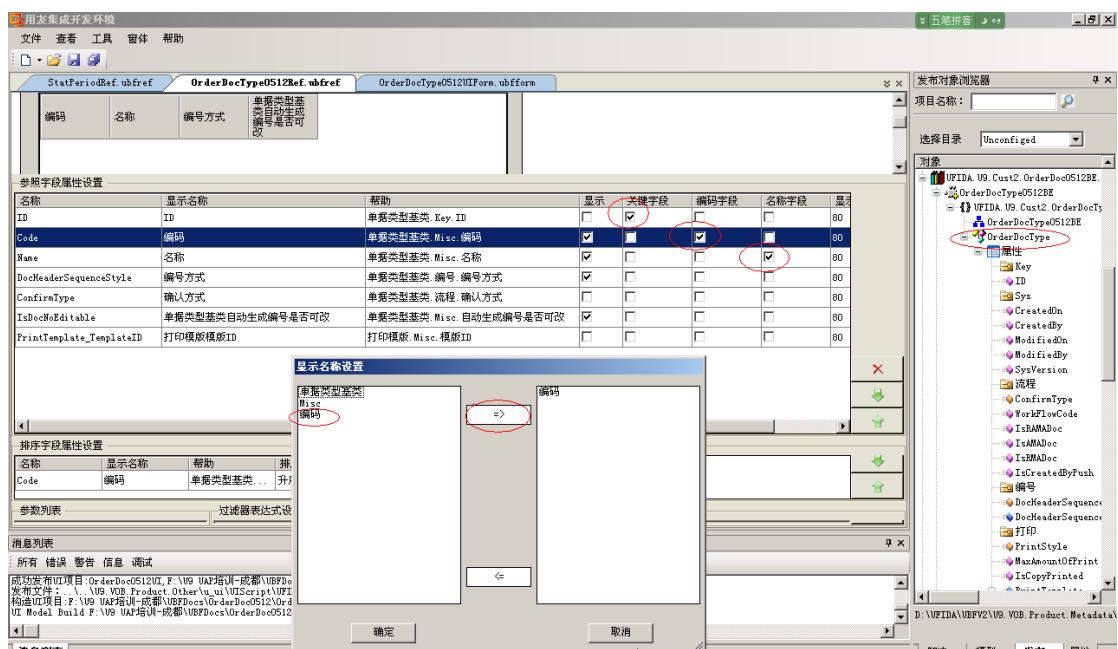
第四章 参照开发

一、 创建单据类型参照组件

1、 定义订货单类型参照组件

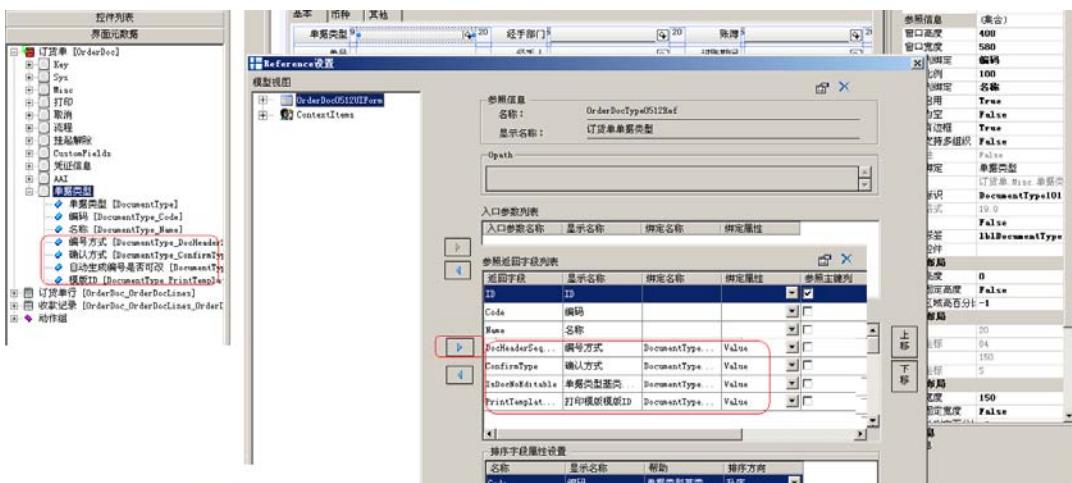
在订货单 UI 组件中新增参照组件，并从参照实体中引用（拖放到搜索窗下面即可），并将 **Code**、**Name** 字段设置为显示栏目，同时，将 **ID** 指定为关键字段，**Code** 指定为编码字段，**Name** 指定为名称字段。





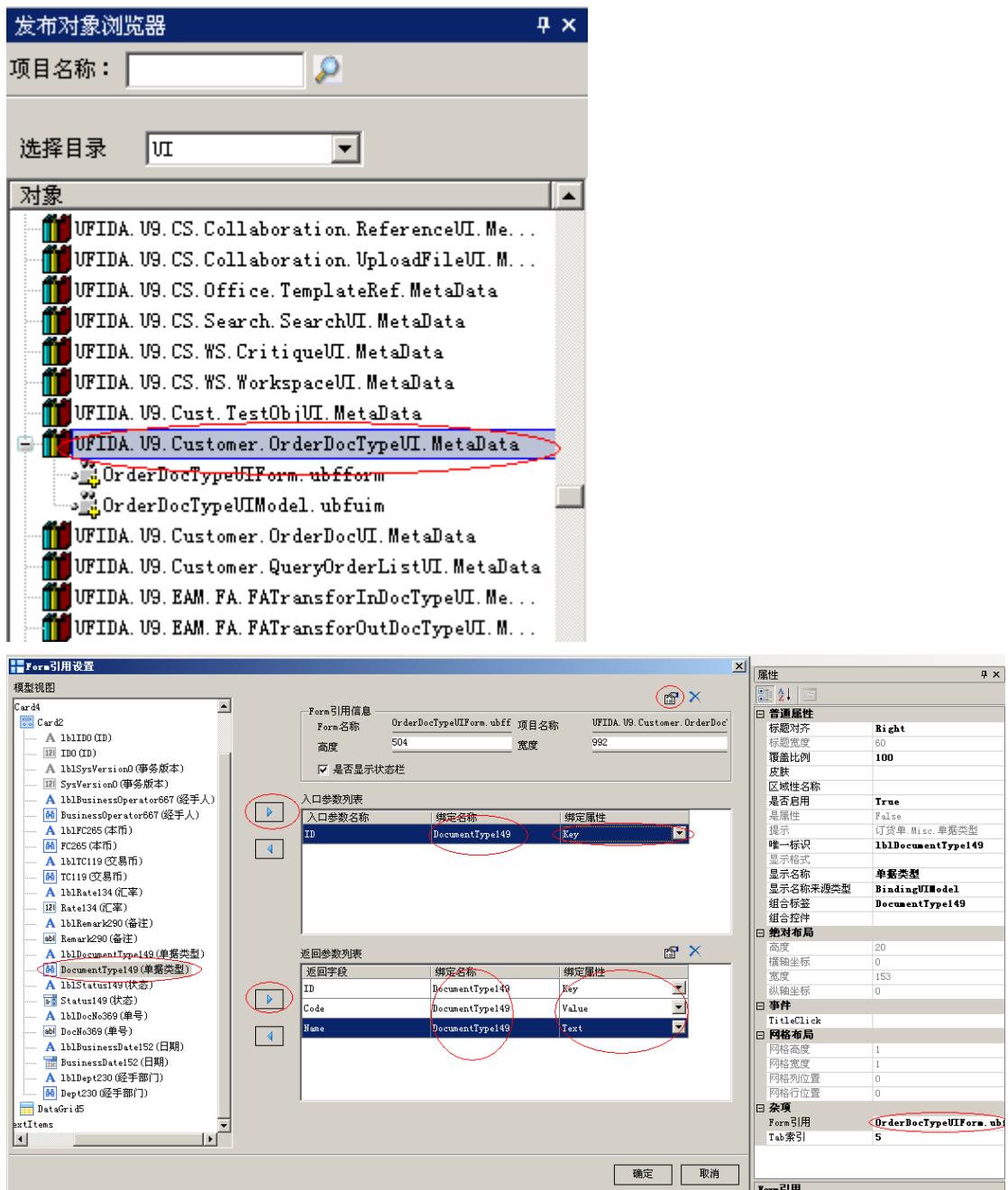
2、调用单据类型参照

在订货单 UIFrom 中设置单据类型参照，并绑定返回值，如编号方式、打印模板等

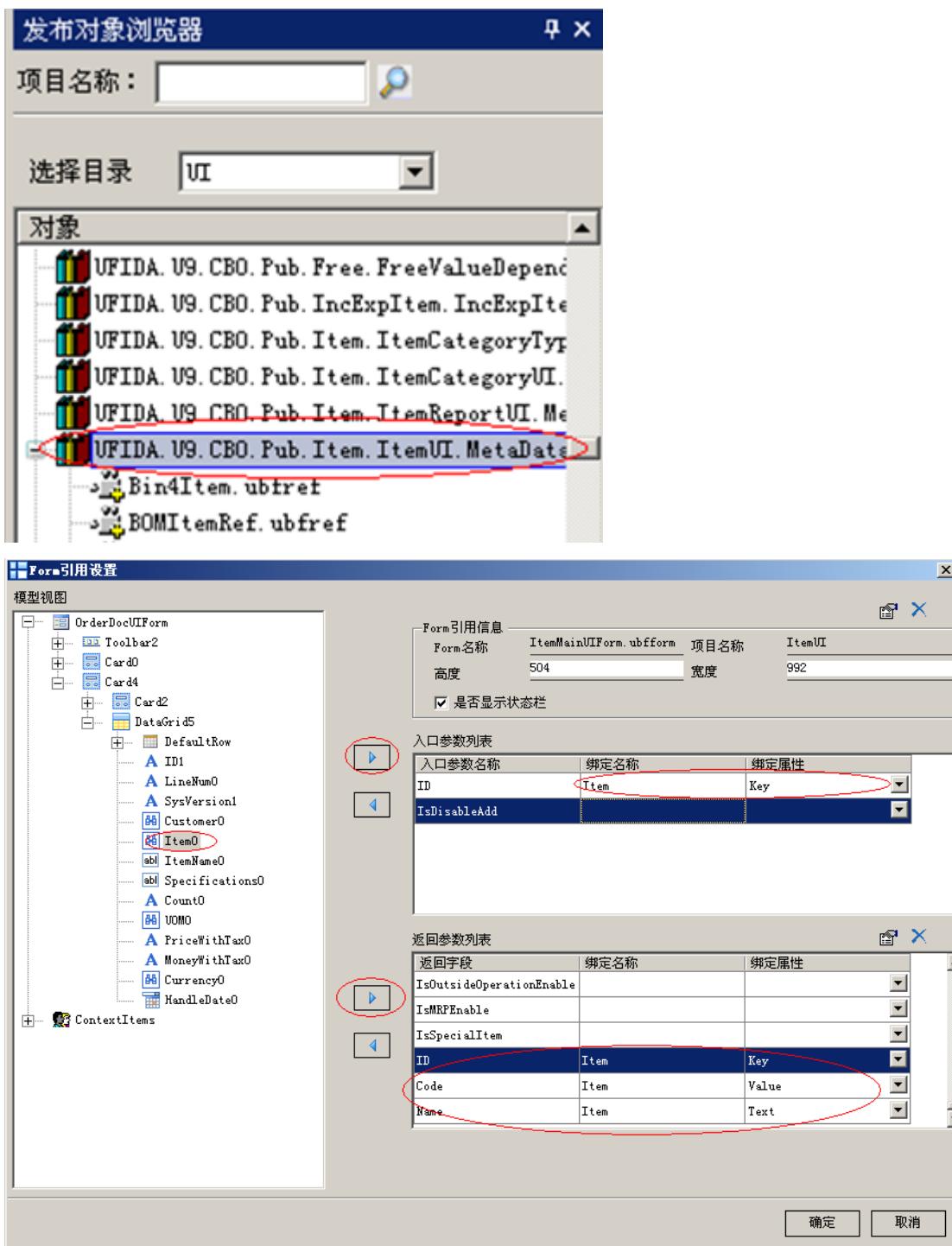


二、 Form引用

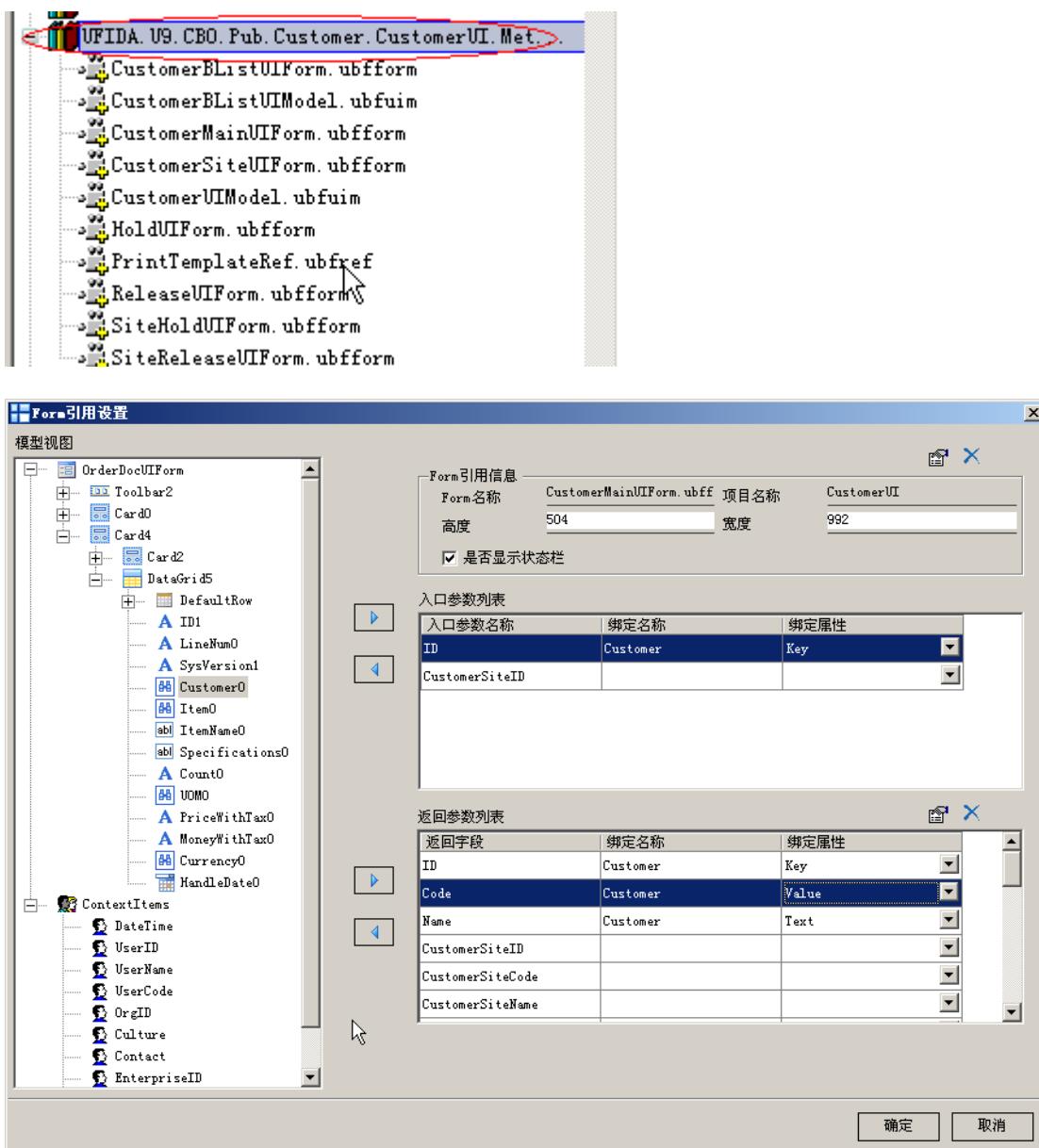
1、 订货单单据类型引用



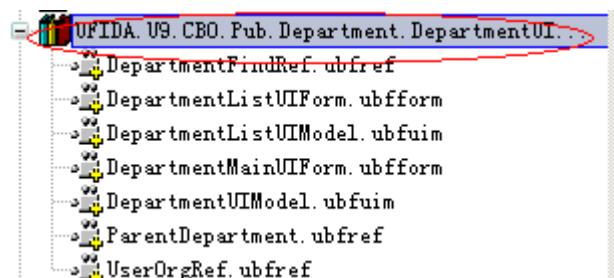
2、 料品档案

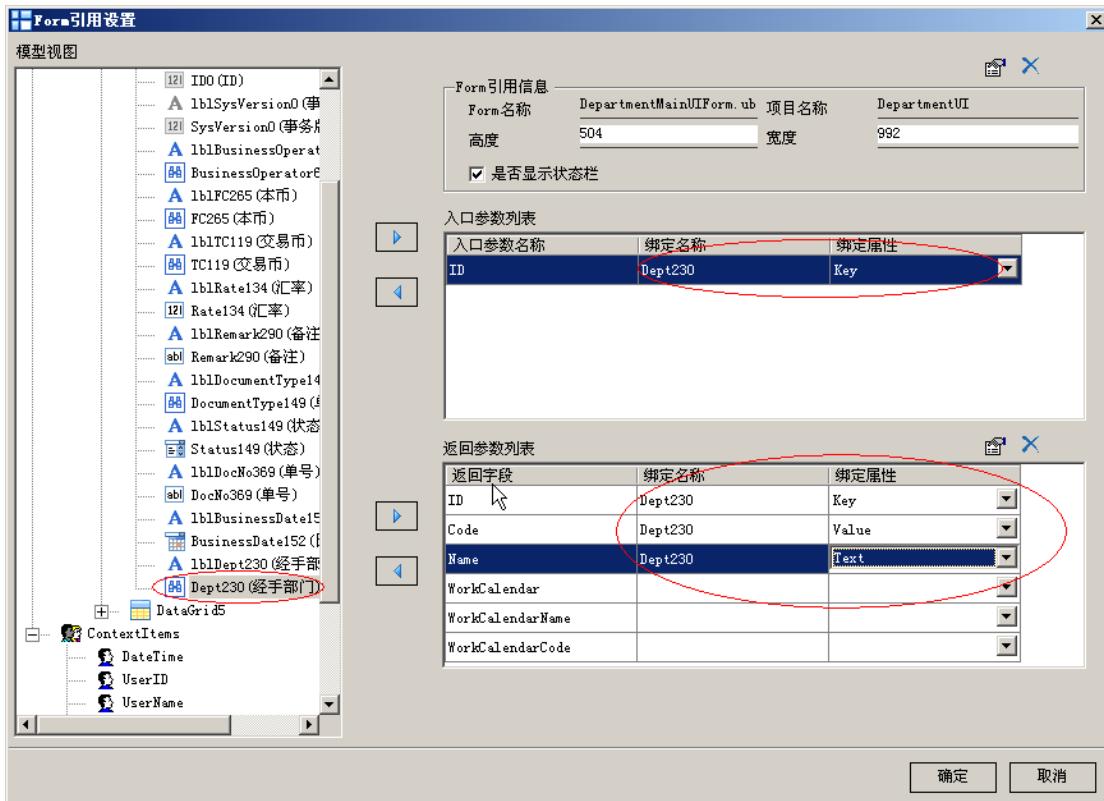


3、客户档案

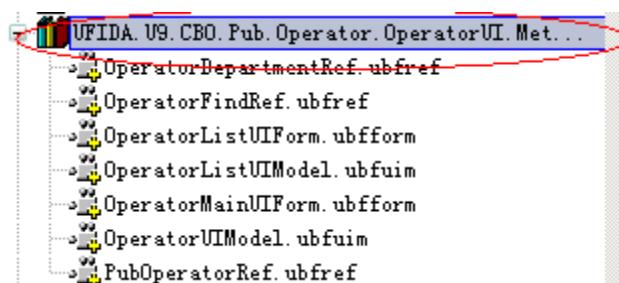


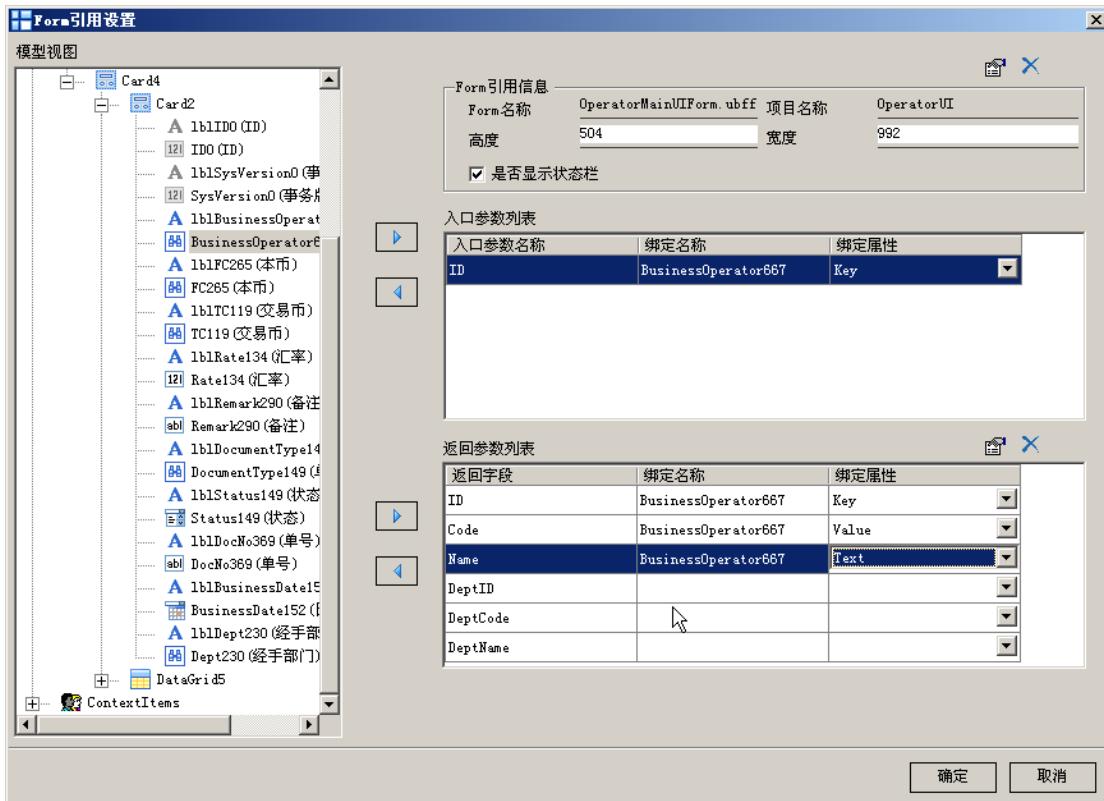
4、部门档案





5、业务员档案





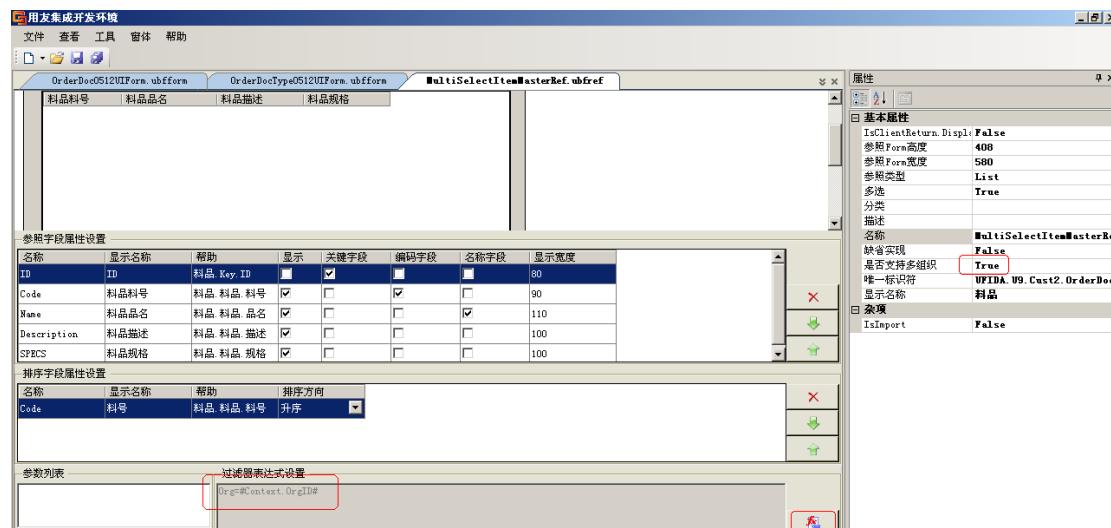
三、 条件参照（参照过滤）

1、 固定条件

若是固定条件,可在设置参照时设定即可,如下,设置按组织参煤断品.

Org=#Context.OrgID#

Effective.Iseffective=1 and '#Context.DateTime#' between Effective.EffectiveDate and Effective.DisableDate



2、完整OQL条件

```
// 单头业务员参照过滤条件，加入 AfterUIModelBinding () 中。  
private void SetHeadBusinessOperatorRefCustomFilter(OrderDocRecord headRecord)  
{  
    if (headRecord == null) return;  
    string strFilter = "1=1";  
    if (!this.IsNullID(headRecord.Dept))  
        strFilter = " Dept = " + headRecord.Dept;  
    this.BusinessOperator137.CustomInParams = BaseAction.Symbol_AddCustomFilter +  
    "= " + strFilter; // 该方法可传入完整条件OQL，若传参照窗中的参数值，可直接用“参数名=XXX”形  
    式传入。  
    // this.BusinessOperator137.CustomInParams="Dept=" + headRecord.Dept;  
}  
  
// Grid 表体中参照过滤条件带入  
((IUFFldReferenceColumn)this.DataGrid9.Columns["BusinessOperator"]).CustomInParams =  
BaseAction.Symbol_AddCustomFilter + "= " + strFilter;  
  
// 标准参照的 Action 中响应传入的参数值  
private void OnLoadData_Extend(object sender, UIEventArgs e)  
{  
    // BaseAction.Symbol_AddCustomFilter="UBF_CustomFilter"  
    string oql =  
this.CurrentModel.cRef.CurrentFilter.OPath;// .get_CurrentFilter().get_OPath();  
    if (NavigateManager.IsTitleLink(this.CurrentPart) &&  
(this.CurrentPart.NameValues["ID"] != null) && this.CurrentPart.NameValues["ID"] != "")  
    {  
        this.CurrentModel.cRef.CurrentFilter.OPath = "ID=" +  
this.CurrentPart.NameValues["ID"].ToString();  
    }  
  
    if (NavigateManager.IsReferencePopup(this.CurrentPart) &&  
(this.CurrentPart.NameValues[BaseAction.Symbol_AddCustomFilter] != null) &&  
this.CurrentPart.NameValues[BaseAction.Symbol_AddCustomFilter] != "")  
    {  
        this.CurrentModel.cRef.CurrentFilter.OPath =  
this.CurrentPart.NameValues[BaseAction.Symbol_AddCustomFilter].ToString();  
    }  
  
    this.OnLoadData_DefaultImpl(sender, e);  
  
    //if (this.MainView.Records.Count == 0)  
    //{

```

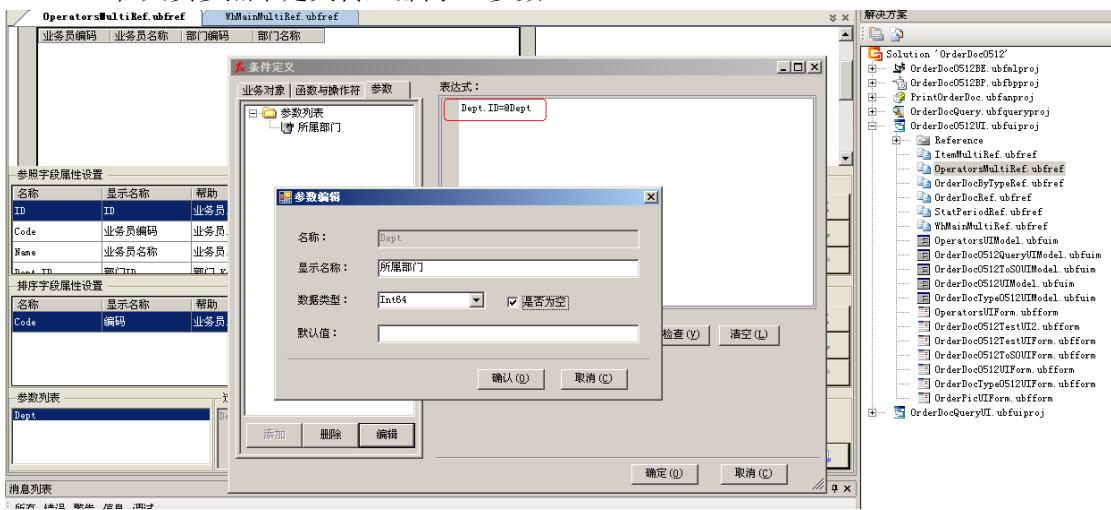
```

        //      this.NewClick_Extend(sender, e);
    //}
    this.CurrentModel.cRef.CurrentFilter.OPath=oql;
}

```

3、命名参数

- 在人员参照中定义传入部门 ID 参数。



- 主 Form 传入参数 [AfterUIModelBinding ()]

```

// 表头
this.BusinessOperator137.CustomInParams = "Dept=" + headRecord.Dept;
// Grid 表体
((IUFF1dReferenceColumn)this.DataGrid9.Columns["Operator"]).CustomInParams = strFilter;

```

- RefForm 中响应参数 [*RefActionExtend.cs]

```

private void OnLoadData_Extend(object sender, UIEventArgs e)
{
    If(this.CurrentPart.NameValues["Dept"]==null)
        throw new Exception("Dept部门参数未传入");
    else
        IUIParameter idParameter0=this.Model.cRef.DefaultFilter.Parameters["@Dept"];
        idParameter0.DataValue= this.CurrentPart.NameValues["Dept"].ToString();
}
this.OnLoadData_DefaultImpl(sender, e);
}

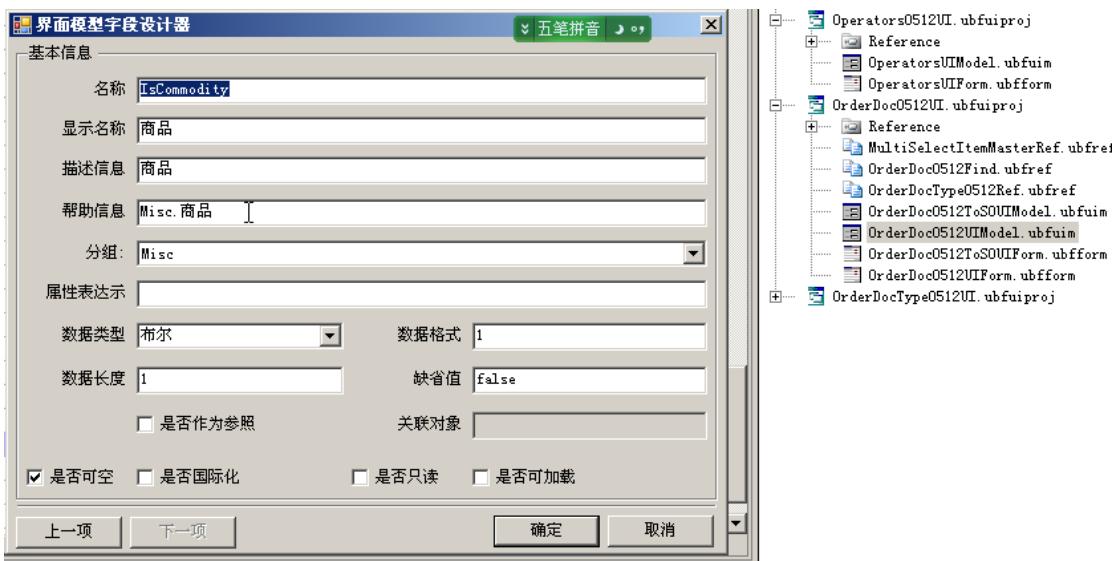
```

4、 动态条件



1) UIModel

在单行新增一布尔型字段“商品”，并设其缺省值为 false。



2) UIForm

将商品列设置出来。

基本 料品 其他						
行号	客户	商品	料品	料品名称	规格型号	单位
<LineNumber <Customer>	<IsComm <Item>	<ItemName>	<Specifications>	<UOM>		
<LineNumber <Customer>	<IsComm <Item>	<ItemName>	<Specifications>	<UOM>		

3) 添加代码

将 GridItemRef_DoBeforeCallback 方法添加到 AfterCreateChildControls () 中。

```
private void GridItemRef_DoBeforeCallback()
{
    AssociationControl assoCGrid = new AssociationControl();
    assoCGrid.SourceServerControl = this.DataGrid9;
    assoCGrid.SourceControl.EventName = "OnBeforeCellFocusEnter";
    ((UFWebClientGridAdapter)assoCGrid.SourceControl).FireEventCols.Add("Item");
//ItemInfo_ItemID
    ClientCallBackFrm callBackFrm = new ClientCallBackFrm();
    callBackFrm.DoCustomerAction += new
    ClientCallBackFrm.ActionCustomer(assoCGrid_DoBeforeItemAction);
    callBackFrm.ParameterControls.Add(this.DataGrid9);
    callBackFrm.Add(assoCGrid);
    this.Controls.Add(callBackFrm);
}

Object assoCGrid_DoBeforeItemAction(CustomerEventArgs args)
{
    ArrayList list =
(ArrayList)args.ArgsHash[UFWebClientGridAdapter.ALL_GRIDDATA_SelectedRows];
    if (list == null || list.Count == 0)
        return args;

    int curIndex = int.Parse(list[0].ToString());
    Hashtable hashT = (Hashtable)((ArrayList)
args.ArgsHash[this.DataGrid9.ClientID])[curIndex];
    object oIsCommodity = hashT["IsCommodity"];

    string strWhere = " 1=1 ";
    //if (isNull(oIsCommodity) || !bool.Parse(oIsCommodity.ToString()))
    //{
    //    this.CurrentState["ItemRefCondition"] = strWhere;
    //    return args;
    //}

    if (!isNull(oIsCommodity) && bool.Parse(oIsCommodity.ToString())) // 响应追加的
条件 ‘商品’
        strWhere = " ItemFormAttribute=" + (int)ItemTypeAttributeEnumData.Commodity; //商品
```

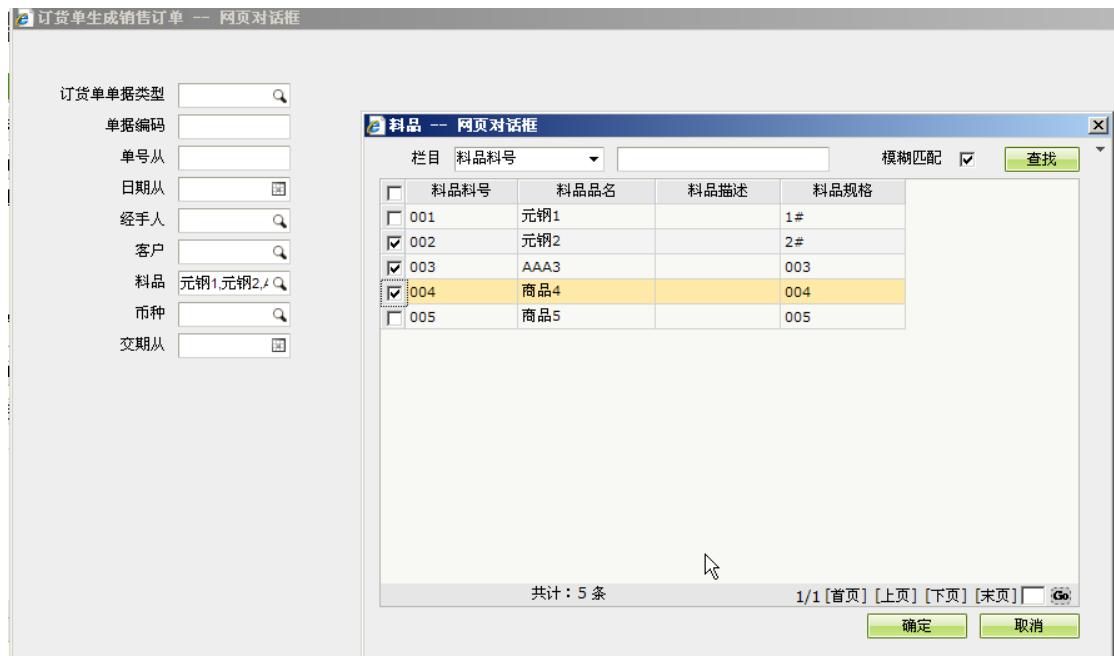
```

//物料参照模式2的自定义条件赋值
//this.CurrentState["ItemRefCondition"] = strWhere;
//return args;

//标准参照的自定义条件赋值
UF WebClientGridAdapter grid = new UF WebClientGridAdapter(this.DataGrid9);
grid.ResetColumnEditorAttribute("Item",
UF WebClientRefControlAdapter.Attributes_AddParam, new string[]
{ BaseAction.Symbol_AddCustomFilter, strWhere });
args.ArgsResult.Add(grid.ClientInstanceWithRefAddParam);
return args;
}

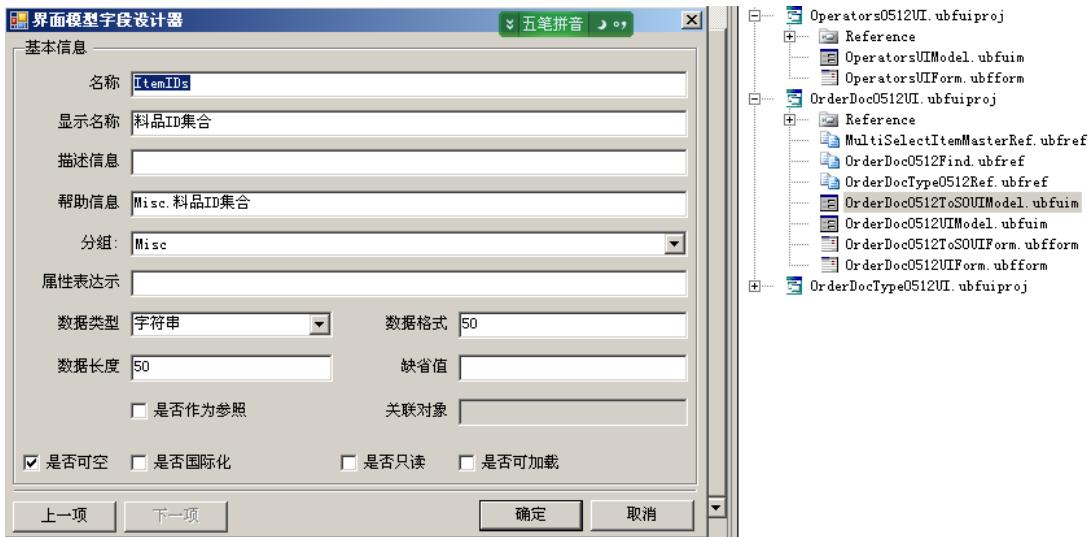
```

四、 多选参照



1、 UIModel

新增一料品 ID 集合，以存放参照返回的 ID，用逗号分隔。



2、 UIForm设置

将 ItemIDs 设置到 UIForm 中，并隐藏该字段。

3、 定义料品参照

设置多选=true。



4、 添加代码

```
private void DataGrid_GridRowDbClicked_Extend(object sender, GridEventArgs e)
{
    //调用模版提供的默认实现。--默认实现可能会调用相应的Action.
```

```
string returnXml = "";
IUIView view = this.Model.cRef;
string strID = "";
string strCode = "";
```

```

        string strName = "";
        Random Ra = new Random();
        string rodID = Ra.Next(200).ToString();
        GetReturnXML(view, ref strID, ref strCode, ref strName);
        returnXml += "<ReferenceReturn>";
        returnXml += "<ReturnField name=' ID' value=' " + rodID + " '/>";//ID集合
        returnXml += "<ReturnField name=' Code' value=' " + strCode + " '/>";//Code集合
        returnXml += "<ReturnField name=' Name' value=' " + strName + " '/>";//name集合
        returnXml += "<ReturnField name=' Description' value=' " + strID + " '/>";//id集合
        //G 类档案,多组织 ID 转换=====
        returnXml += string.Format("<ReturnField name=' {0}' value=' {1}' />",
UFIDA.U9.UI.PDHelper.Commands.ReferenceMultiOrgInitComplete.MultiRefIDFlag);

//=====end=====
returnXml += "</ReferenceReturn>";
if (string.IsNullOrEmpty(returnXml))
{
    string script1 = "<script language=\"javascript\">";
    script1 += "window.close();";
    script1 += "</script>";
    AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),
"ReferenceReturn", script1, false);
    return;
}
string xml = "<ReferenceReturns>";
xml += returnXml;
xml += "</ReferenceReturns>";
 XmlDocument doc = new XmlDocument();
doc.LoadXml(xml);
xml = doc.DocumentElement.OuterXml;
xml = UFIDA.U9.UI.Commands.CommandHelper.EscapeHTML(xml);//编码
string script = "<script language=\"javascript\">";
script += "window.returnValue = \" " + xml + " \" ;";
script += "window.close();";
script += "</script>";
AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),
"ReferenceReturn", script, false);
}

private void GetReturnXML(IUIView view, ref string strID, ref string strCode, ref string
strName)//,ref string strSPECS
{
    //从缓存中读取选中的记录
    System.Collections.Generic.IList<IUIRecord> SelRec =
UFSoft.UBF.UI.MD.Runtime.UIRuntimeHelper.Instance.GetSelectRecordFromCache(view);

```

```

if (SelRec != null && SelRec.Count > 0)
{
    foreach (cRefRecord record in SelRec)
    {
        strID += record["ID"] + ",";
        strCode += record["Code"] + ",";
        strName += record["Name"] + ",";
        // strSPECS += record["SPECS"] + ",";
    }
    strID = strID.Substring(0, strID.Length - 1);
    strCode = strCode.Substring(0, strCode.Length - 1);
    strName = strName.Substring(0, strName.Length - 1);
    //strSPECS = strSPECS.Substring(0, strSPECS.Length - 1);

}
}

```

5、 绑定参照

返回料品 ID 集合。



五、 自定义参照

如业务员参照同时显示所属部门。

网页对话框

	编码 *	名称
一公司(001)	CG	王采购

共计：1 条 1/1 [首页] [上页] [下页] [末页] [Go]

确定 取消

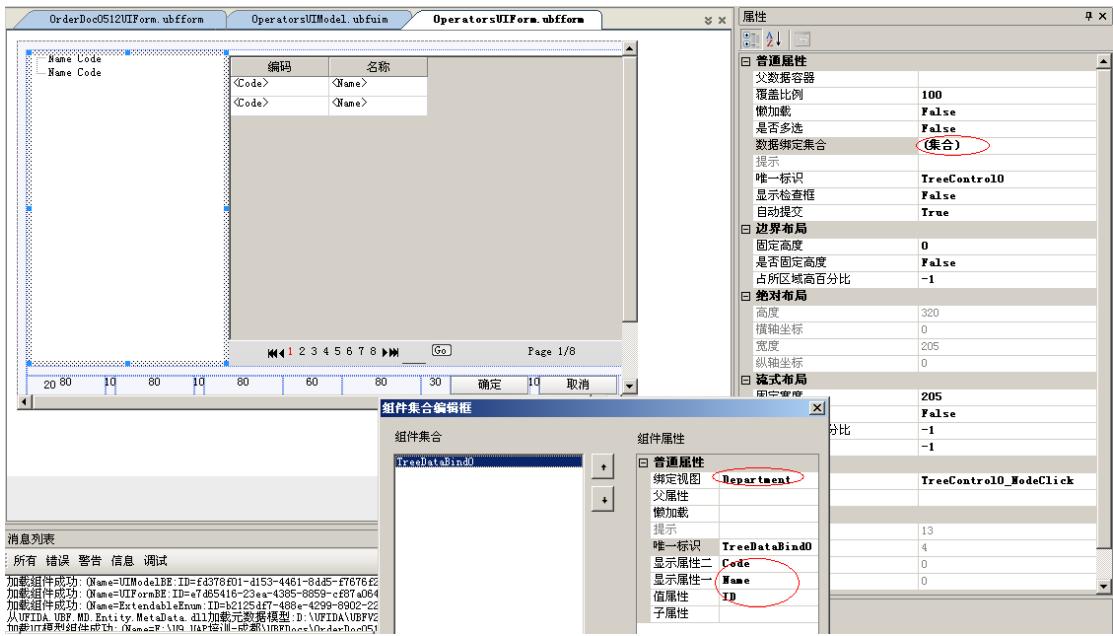
1、UIModel设置

将部门和业务员实体分别拖放到设计器中。注意业务员需将该实体下的部门拖入。

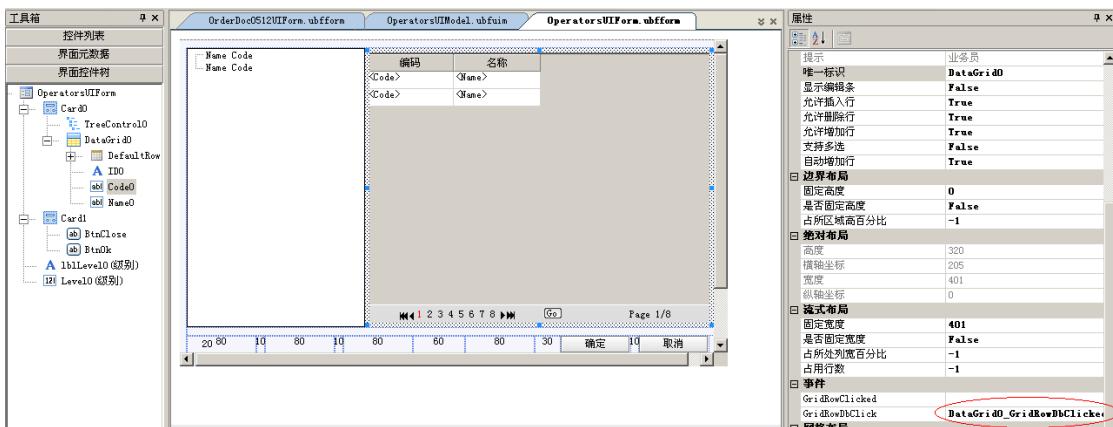
字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
Key	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
ID	长整数	可空	8.0		部门	ID	
Misc	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
Org	长整数	可空	19.0		组织机构	部门	组织
Code	字符串	可空	50			编码	
Name	字符串	必填	50			名称	名称
Level	整数	必填	10.0	0		级别	级别

字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
Key	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
ID	长整数	可空	8.0		业务员	ID	
Misc	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
Dept	部门	可空	19.0		部门	业务员	部门
Code	字符串	可空	50			编码	
Name	字符串	必填	50			名称	名称
属性组	属性名称	数据类型	可空	数据格式	缺省值		

2、UIForm绑定部门视图



3、UIForm绑定业务员



4、参照添加代码

1) 加载部门数据

```
private void OnLoadData_Extend(object sender)
{
    OnLoadData_DefaultImpl(sender);
    this.Model.Department.Clear();
}
```

```
this.Model.Department.CurrentFilter.OPath = " Org =" + PDContext.Current.OrgID;
this.Model.Department.CurrentFilter.OrderBy = " Code";
this.Action.CommonAction.Load(this.Model.Department);
//加载部门树
foreach (DepartmentRecord dept in this.Model.Department.Records)
{
    //RadTreeNode newNode = new RadTreeNode();
    //newNode.Value = dept.ID.ToString();
    //newNode.Text = dept.Name + "(" + dept.Code + ")";
    //this.TreeControl0.Nodes.Add(newNode);
    CreateNode(dept.ID.ToString(), dept.Name + "(" + dept.Code + ")", dept.Level.Value);

}
}

private void CreateNode(string Value, string Text, int Level)
{
    RadTreeNode newNode = new RadTreeNode();
    newNode.Value = Value;
    newNode.Text = Text;
    if (Level == 0) // 根节点
        this.TreeControl0.Nodes.Add(newNode);
    else if (Level == 1)
    {
        this.TreeControl0.Nodes[TreeControl0.Nodes.Count - 1].Nodes.Add(newNode);
    }
    else if (Level == 2)
    {
        RadTreeNode nd1 = this.TreeControl0.Nodes[TreeControl0.Nodes.Count - 1];
        nd1.Nodes[nd1.Nodes.Count - 1].Nodes.Add(newNode);
        //this.TreeControl0.Nodes[TreeControl0.Nodes.Count -
1].Nodes[TreeControl0.Nodes.Count - 1].Nodes.Add(newNode);
    }
}
```

2) 响应确定和双击事件

```
private void BtnOk_Click_Extend(object sender, EventArgs e)
{
    BtnOk_Click_DefaultImpl(sender, e);
    OperatorsRecord oper = this.Model.Operators.FocusedRecord;
    if (oper == null) return;
    string returnXml = "";
    returnXml += "<ReferenceReturn>";
```

```

        returnXml += "<ReturnField name=' ID' value=' " + oper.ID + "' />";
        returnXml += "<ReturnField name=' Code' value=' " + oper.Code + "' />";
        returnXml += "<ReturnField name=' Name' value=' " + oper.Name + "' />";
        returnXml += "</ReferenceReturn>";
        if (string.IsNullOrEmpty(returnXml))
        {
            string script1 = "<script language=\"javascript\">";
            script1 += "window.close();";
            script1 += "</script>";
            AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),
"ReferenceReturn", script1, false);
            return;
        }
        string xml = "<ReferenceReturns>";
        xml += returnXml;
        xml += "</ReferenceReturns>";
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(xml);
        xml = doc.DocumentElement.OuterXml;
        xml = UFIDA.U9.UI.Commands.CommandHelper.EscapeHTML(xml); //编码
        string script = "<script language=\"javascript\">";
        script += "window.returnValue = \" " + xml + " ;";
        script += "window.close();";
        script += "</script>";
        AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),
"ReferenceReturn", script, false);
    }

    private void DataGrid0_GridRowDbClicked_Extend(object sender, GridDBClickEventArgs e)
    {
        DataGrid0_GridRowDbClicked_DefaultImpl(sender, e);
        this.BtnOk_Click_Extend(sender, e);
    }

```

3) 响应部门选择变更事件

```

private void TreeControl0_NodeClick_Extend(object sender, RadTreeNodeEventArgs e)
{
    TreeControl0_NodeClick_DefaultImpl(sender, e);
    this.Model.Operators.Clear();
    this.Model.Operators.CurrentFilter.OPath = " Dept = " +
this.TreeControl0.SelectedNode.Value;
    this.Action.CommonAction.Load(this.Model.Operators);
}

```

4) 响应取消事件

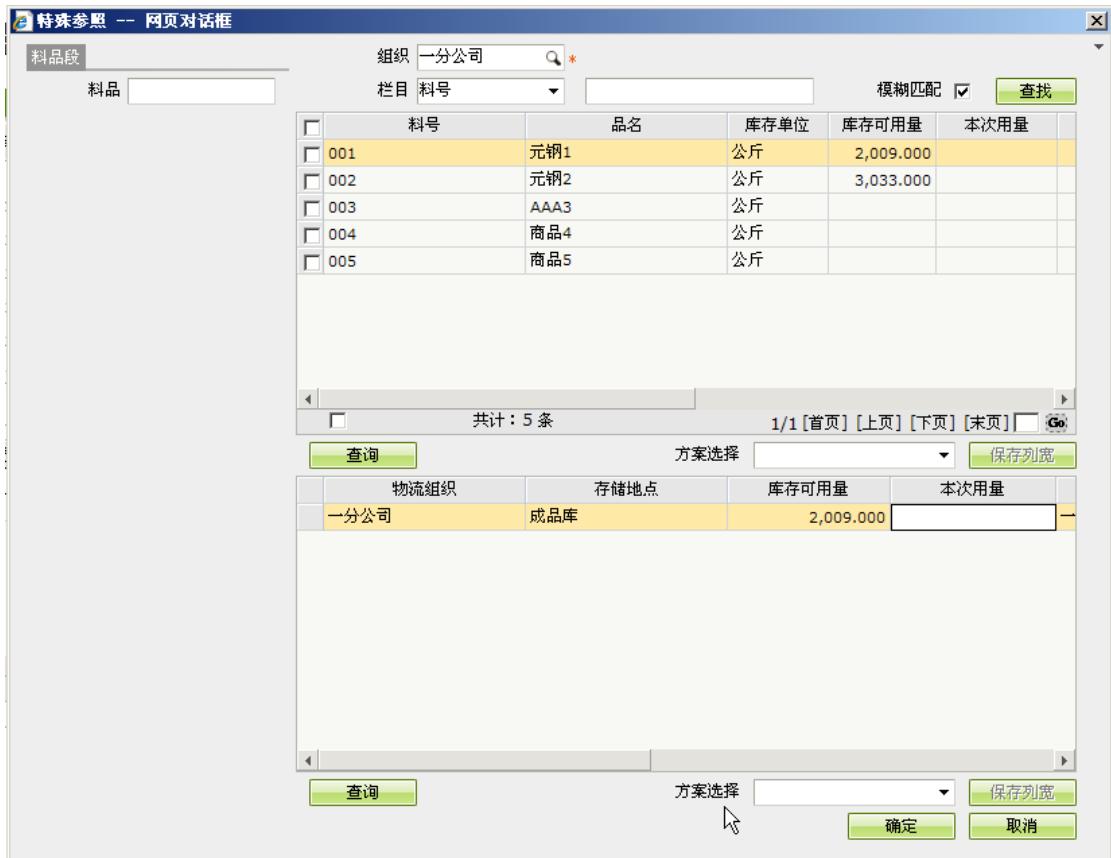
```
private void BtnClose_Click_Extend(object sender, EventArgs e)
{
    BtnClose_Click_DefaultImpl(sender, e);
    string script1 = "<script language=\"javascript\">";
    script1 += "window.close();";
    script1 += "</script>";
    AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),
"ReferenceReturn", script1, false);
}
```

5、注册参照

```
public void AfterCreateChildControls()
{
    //修改业务员参照RefType为参照模式2
    this.BusinessOperator137.RefType = "abccee64-22d3-41d7-a368-678c8500fbf7";
    this.BusinessOperator137.RefHeight = 375;
    this.BusinessOperator137.RefWidth = 630;
}
```

六、料品特殊参照

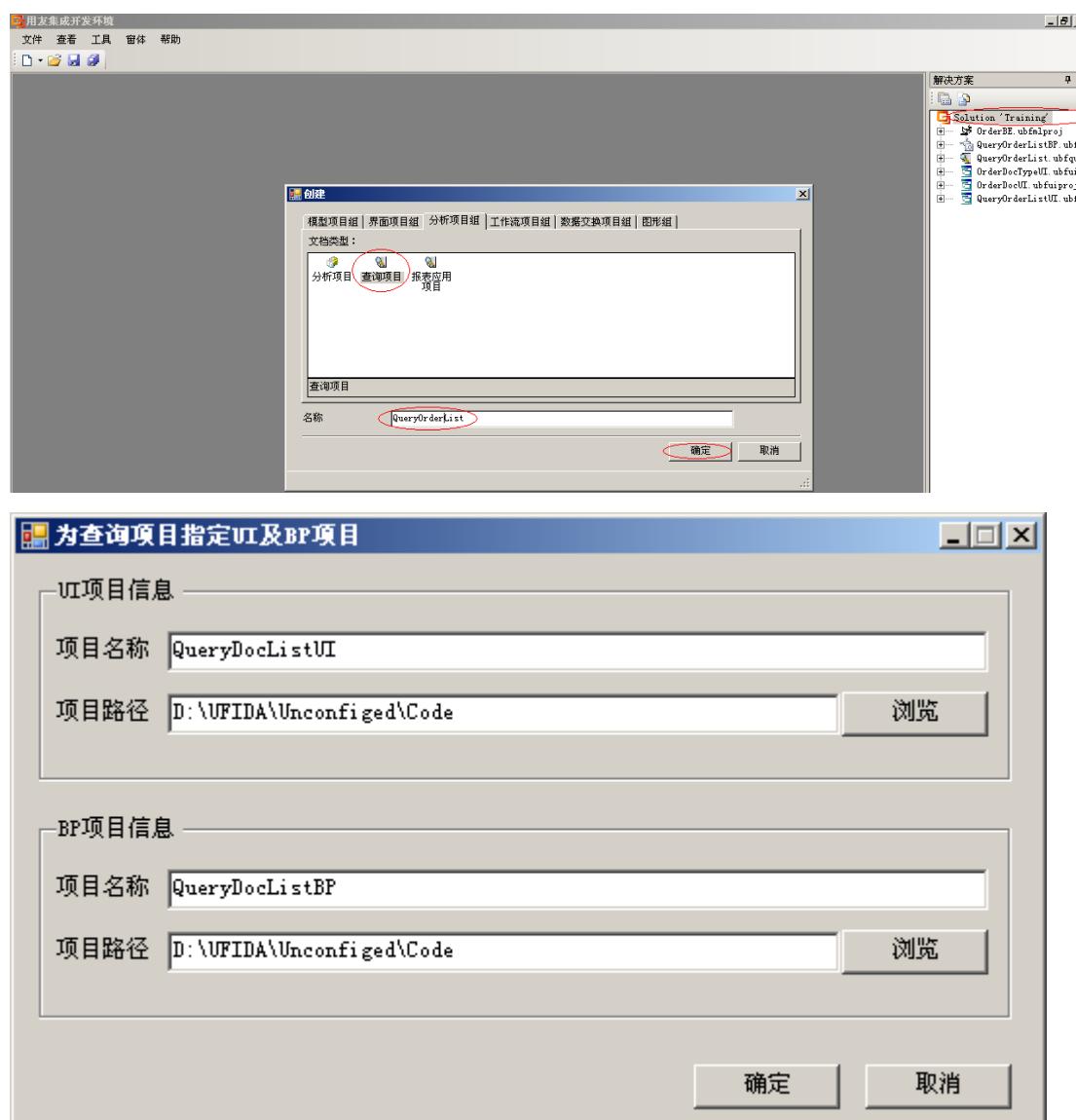
如杂发单上可同步显示料品的存储地点、可用量等。



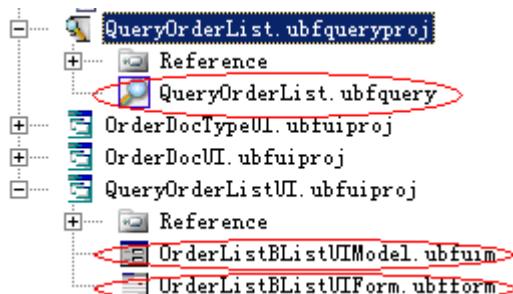
```
Using UFFSoft.UBF.UI.ControlModel;
public void AfterCreateChildControls()
{
    ((IUFFIdReferenceColumn)this.DataGrid9.Columns["Item"]).RefType =
"326218e2-d0a5-4c3e-92c2-19bc134bf5a5";
    ((IUFFIdReferenceColumn)this.DataGrid9.Columns["Item"]).RefHeight = 600;
    ((IUFFIdReferenceColumn)this.DataGrid9.Columns["Item"]).RefWidth = 800;
}
```

第五章 单据列表开发

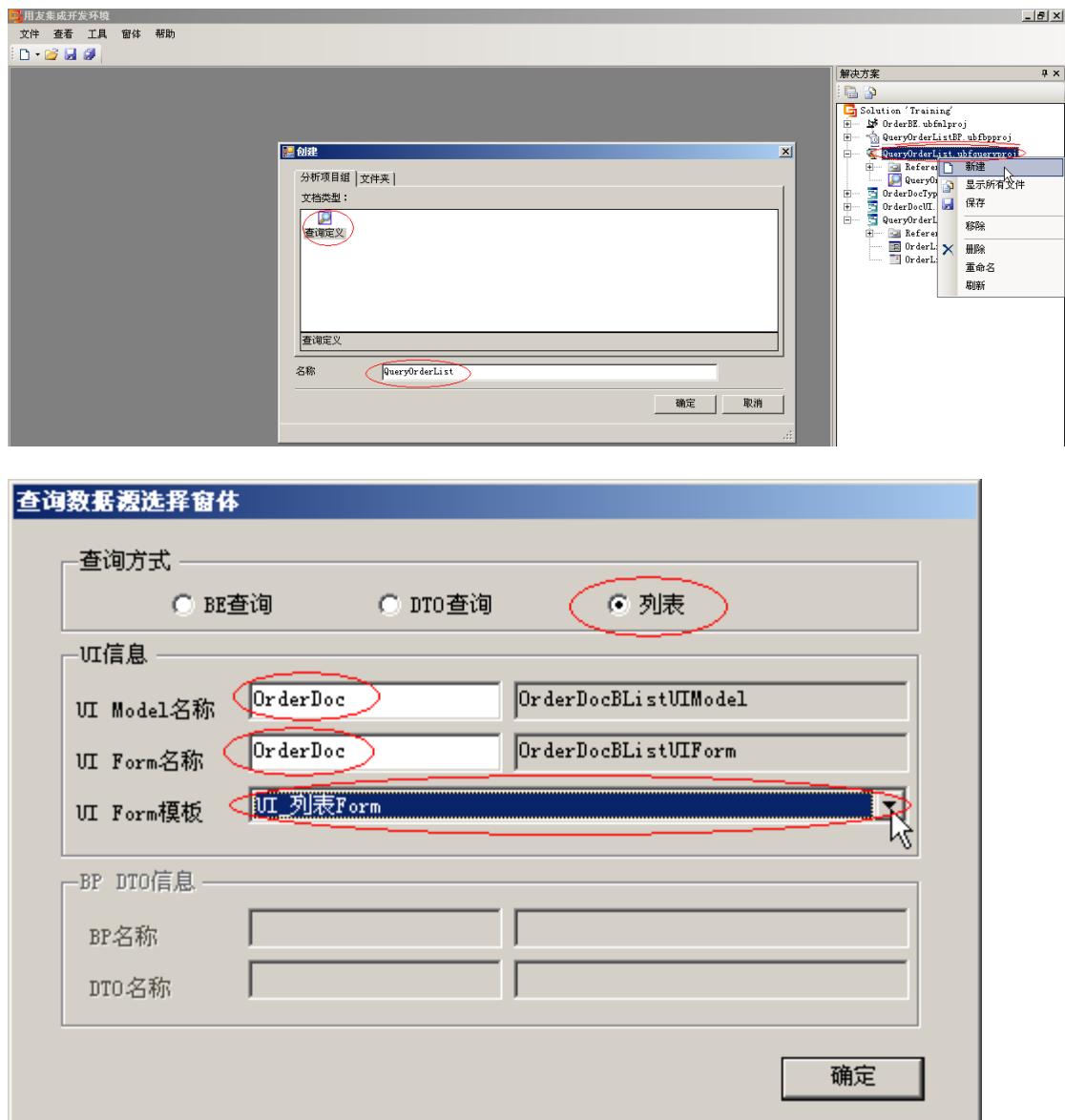
一、 创建查询项目



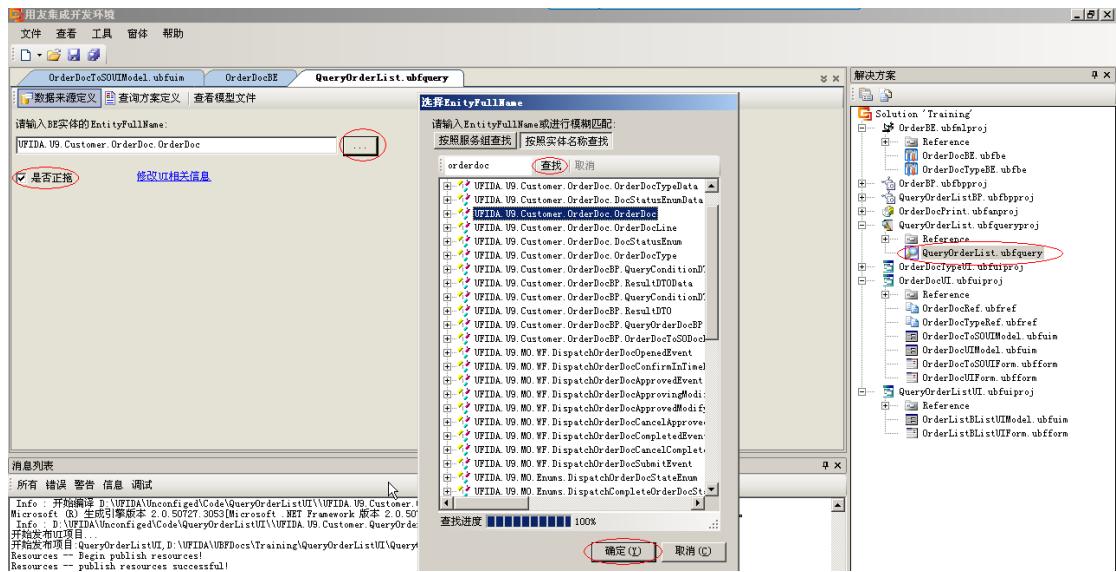
注：此次指定后，通常将其查询 UI 和查询 BP 删除后通过下一步重建。



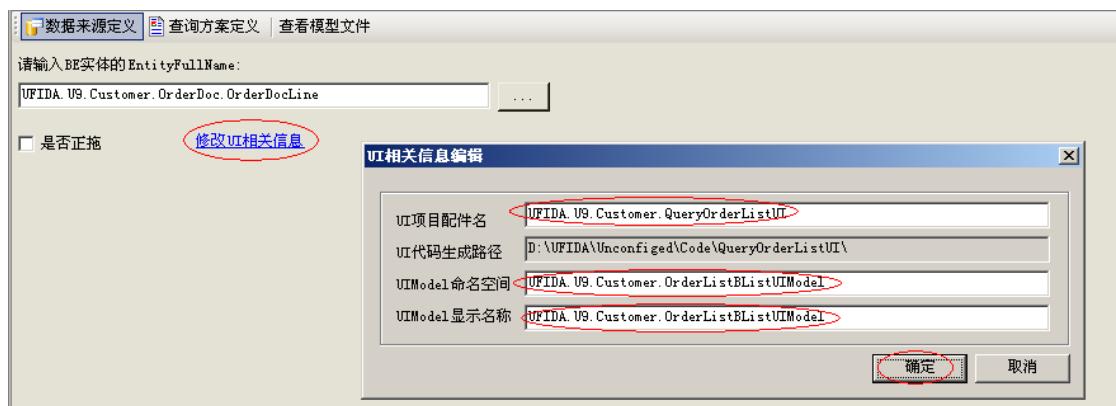
二、重建查询UI



三、 定义查询数据源



注：取表头数据则选择正拖，取单据表体则进行反拖，然后修改 UI 相关信息。



四、 定义查询方案

1、 定义查询条件

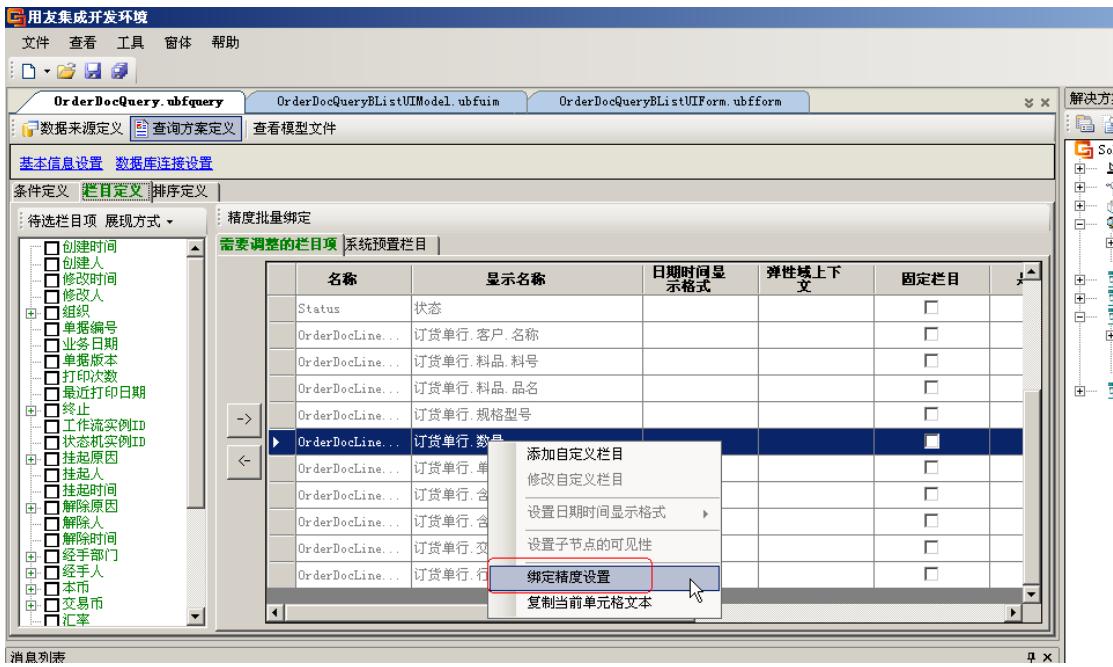


2、 定义查询栏目

可设置栏目是否可见、始终加载及是否合计等。



数字型字段栏目可在“需要调整的栏目项”页签中设置其小数位数，可通过静态或动态绑定实现。



如下：指定数量列固定为 2 位小数。



3、 定义排序字段

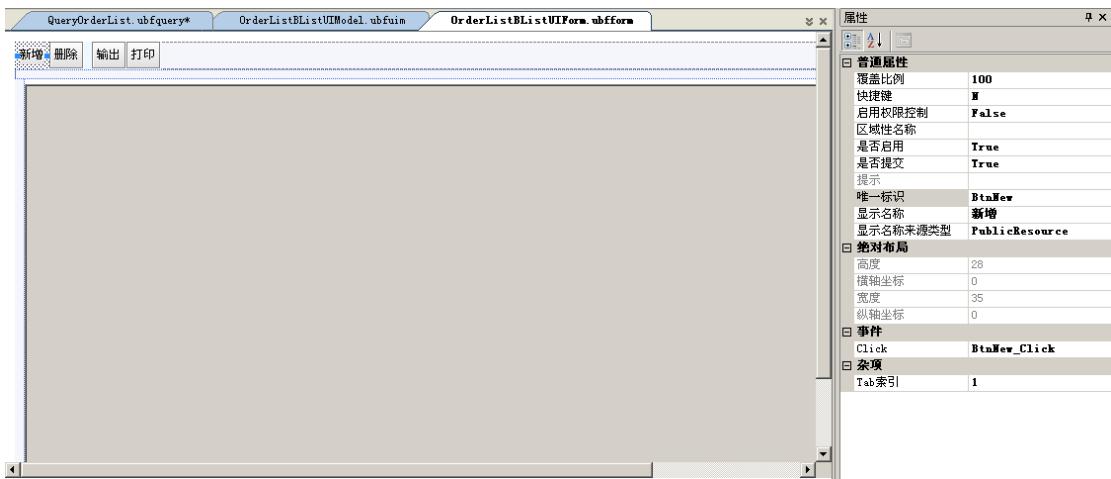


五、 定义查询UI

1、 查看UIModel



2、查看UIForm



3、响应列表事件

5) 响应列表新增事件

在 OnNew_Extend 事件中调用原订货单。

```
string PageID = "SD.Cust.OrderDoc";
this.CommonAction.CurrentPart.NavigatePage(PageID, null);
```



6) 响应列表双击事件

在 OnNew_Extend 事件，根据其单据 Key 显示出对应单据。

```
private void OnGridRowDoubleClick_Extend(object sender, UIActionEventArgs e) // 响应列表双击事件
{
    string DataID = this.CurrentModel.OrderDoc.FocusedRecord.ID.ToString(); // OrderDoc表头实体名
    string CardPageID = "SD.Cust.OrderDoc"; // 在这里CardPageID表示卡片的URI
    OnNavigateCard("Browse", DataID, CardPageID);
    this.OnGridRowDoubleClick_DefaultImpl(sender, e); // 调用模版定义的默认实现方法.如需扩展,请直接在此编程.
}

private void OnNavigateCard(string type, string dataID, string formID)
```

```

    {
        string FormID = formID;//AddFormID
        string DataID = dataID;
        if (dataID == String.Empty && type == "Browse")
        {
            return;
        }
        else
        {
            System.Collections.Specialized.NameValueCollection nameValCol = new
            System.Collections.Specialized.NameValueCollection();
            nameValCol.Add("PDPPageStatus", type);
            nameValCol.Add("ID", dataID);
            this.CommonAction.CurrentPart.NavigatePage(FormID, nameValCol);
        }
    }

```

UFIDA.U9.Customer.OrderListBLListUIModel.OrderListBLListUIModelAction.cs

注：列表只能提取表头数据（正拖）方式，若加载了表体数据，则只能取表体的 Key，导致单据显示错误。

7) 保存列表的列宽

```

private void btnSaveCurrentQueryCase_Click_Extend(object sender, EventArgs e)
{
    //调用模版提供的默认实现。--默认实现可能会调用相应的Action。
    //保存列宽的代码

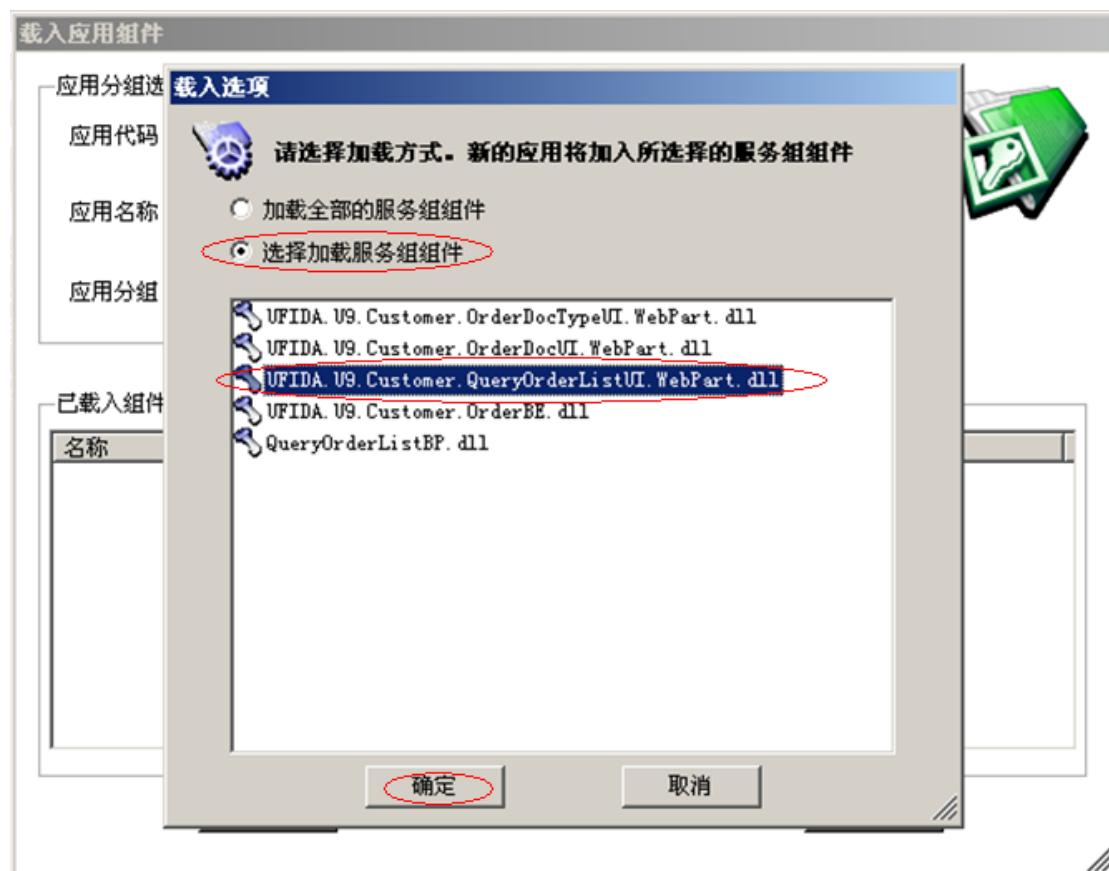
    UFIDA.UBF.Query.CommonService.QryService.SaveQueryCaseColumnsWidth(this.Action, sender, e);

    btnSaveCurrentQueryCase_Click_DefaultImpl(sender, e);
}

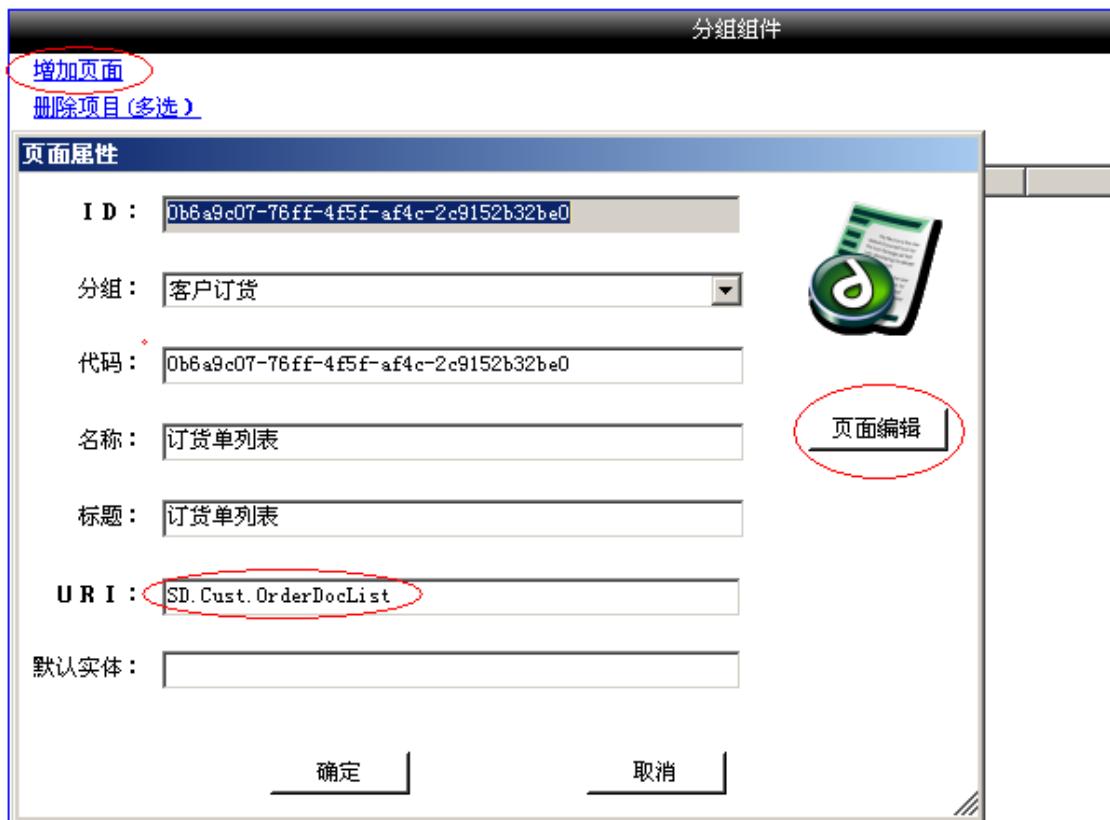
```

六、 装配

1、 加载页面



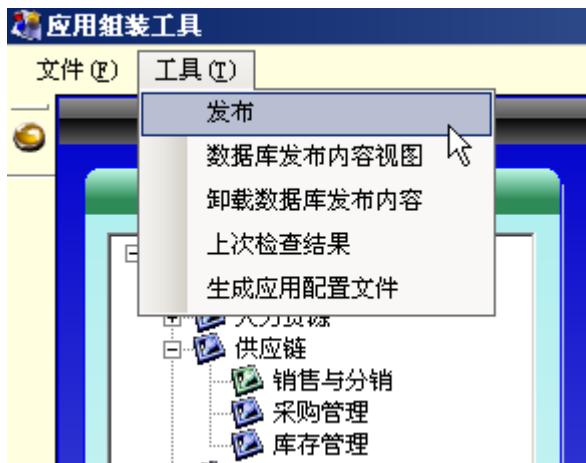
2、增加页面



3、 定义菜单



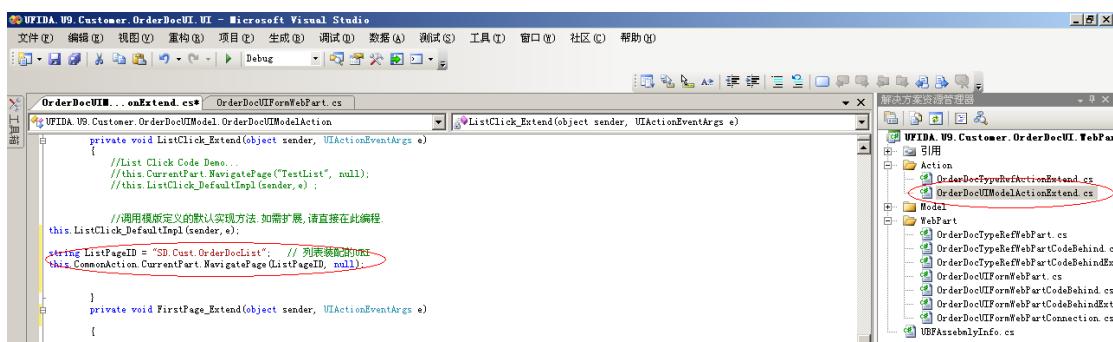
4、 组装发布



七、 调用列表

在相应 UIModel(OrderDocUI)的 Action 列表事件中调用列表 URI。

```
string PageID = "SD.Cust.OrderDoc";
this.CommonAction.CurrentPart.NavigatePage(PageID, null);
```



八、 多WebPart查询

第六章 单据开发（高级）

一、 后台组件编程

1、 实体对外结构的基础组成

- 实体的Key（强类型的EntityKey）
- 实体的查询类Finder
- 实体的强类型集合EntityList
- 实体的资源属性和强类型访问属性的辅助类EntityRes
- 实体的OriginalData
- 实体状态SysState，其枚举定义UFSoft.UBF.PL.Engine.ObjectState。

状态值	枚举值	说明
-1	NotExist	不存在，操作被取消
0	Unknown	未知
1	Unchanged	未变化
2	Inserted	被增加
4	Updated	被修改
8	Deleted	被删除
9	Operating	操作中
10	NeedLoad	需重新加载

2、 Session

需引用 D:\UFIDA\UBFV2\U9.VOB.Product.UBF\UBFStudio\Runtime\UFSoft.UBF.Business.dll。

1) Session的概念

- 在 UBF 中，即持久层的一个边界，主要用作批量提交，并标识这次批量提交的边界，不涉及到事务等概念。
- 当前 ISession 可通过 Session 的 Current 属性获得，每调用一次 Session 的 Open 方法，Current 属性都会被更新

2) Session的使用

```
using(ISSession session = Session.Open()) {  
    ...  
    session.Commit(); //提交更改，但只是Session范围内的修改更新  
}
```

3) Session嵌套

当发生session嵌套的情况时，每次提交都是真正提交（新建请求、修改请求、删除请求）

```
using(ISSession session1 = Session.Open()) {  
    ...  
    using(ISSession session2 = Session.Open()) {  
        ...  
        session2.Commit(); //提交更改，但只是Session2范围内的修改更新  
    }  
    session1.Commit(); //提交更改，只处理Session1的修改更新  
}
```

4) Session应用

① 实体新建

新建实体实体的操作分为主实体的新建和非主实体的新建，如果是主实体新建，用不带参数的Create方法，如果是非主实体的，用带上级实例参数的Create方法。

```
using(ISSession session = Session.Open()) //最好在 Session 中创建  
{  
    A_Com1c1 objA=A_Com1c1.Create();  
    objA.Code_A_Com1c1=1000;  
    objA.Name_A_Com1c1="objA";  
    B_Com1c1 objB=B_Com1c1.Create(objA) //B为A的子实体  
    objB.Code_B_Com1c1=1001;
```

```
    objB.Name_B_Com1c1="objB";
    session.Commit();
}
```

② 实体更新

```
using(ISession session = Session.Open()) {
Customer obj= Customer.Finder.FindByID(id);
if(obj!=null)
obj.Name= "new name";
session.Modify(obj);
...
...
session.Commit();
}
```

③ 实体删除

对于组合关系，如果删除主实体，那么，主实体下面的所有子实体，孙实体会被级联删除，对于非主实体的删除，可以用子集合的**RemoveAt**和**Remove (entity)**方法，这2个方法等价，例如

```
using(ISession session = Session.Open())
{
    A_Com1cNobjA=A_Com1cN.Finder.FindByID(aID);
    objA.B_Com1cN.RemoveAt(0);
    //或用下面的方式:
    B_Com1cNb=Com1cN.Finder.FindByID(bID);
    objA.B_Com1cN.Remove(b);
    session.Commit();
}
```

④ 实体复制

```
using (ISession session1 = Session.Open())
{
    CarInfoVerBE.CarInfoVer carinfoVer= CarInfoVerBE.CarInfoVer.Create();
    carinfo.CopyTo(carinfoVer, false);    // 不复制其他相关数据
    carinfoVer.DocVersion = GetDocVersion(carinfo.Code);
    session1.Commit();                  // 提交更改
}
```

```
private int GetDocVersion(string carCode) // 取得carCode车的历史版本号
{
    EntityDataQuery q = CarInfoBE.CarInfo.Finder.CreateDataQuery();
    int ver = Convert.ToInt32(q.FindValue("select count(code) from
UFIDA::U9::BUS::CarInfoVerBE::CarInfoVer where code=' " + carCode + "'") ?? "0");
    return ver + 1;
}
```

3、 实体查询

1) 实体的查询类Finder

- 返回强类型的对象
- 常用方法: Find, FindByID, FindAll
- 举例

```
OqlParamList pList= new OqlParamList();
string strOql="Code=@Code";
pList.Add(new OqlParam("Code","11"));
Customer.EntityListlist=Customer.Finder.FindAll(strOql, pList.ToArray());
```

2) EntityDataQuery

主要用于希望返回结果是IDataReader, DataSet, 单值的情况。

- 创建方式:
 - ③ 通过Entity对象:
EntityDataQuery q = Customer.Finder.CreateDataQuery();
 - ④ 通过EntityFullName:
EntityDataQuery q = new EntityDataQuery("UFIDA.U9.CB0.SCM.Customer.Customer");
EntityDataQuery q = new EntityDataQuery(Customer.EntityRes.BE_FullName);
- 返回DataSet: FindDataSet
 - ① 全OQL方式
DataSet ds=q.FindDataSet("select ID, Name, Code from UFIDA::U9::CB0::SCM::Customer::Customer");
 - ② 条件OQL方式
DataSet ds=q.FindDataSet("ID>1");

示例代码:

```
EntityDataQuery query = Finder.CreateDataQuery();
query.Select(new string[] { "count(ID)" });
query.get_Parameters().Add(new UFSOFT.UBF.PL.OqlParam(this.ID));
```

```
if (Convert.ToInt32(query.FindValue("ParentNode = @ID")) > 0)
{
    throw new ParentDeleteException(this.Code, this.Name);
}
```

3) EntityQuery

主要用于希望返回的结果是实体（弱类型）

以强类型的方式使用Customer.Finder，其对外接口和功能与EntityQuery类似，只是Customer.Finder返回强类型的对象，EntityQuery返回弱类型的对象。

- 创建方式：

- ① 通过Entity对象：

```
EntityQuery e q= Customer.Finder.CreateQuery();
```

- ② 通过EntityFullName：

```
EntityQuery q = new EntityQuery("UFIDA.U9.CBO.SCM.Customer.Customer");
EntityQuery q = new EntityQuery(Customer.EntityRes.BE_FullName);
```

- 返回实体：FindByID和Find

FindByID, 先从缓存加载对象，如果缓存没有，则从数据库加载对象；

```
Customer a = Customer.Finder.FindByID(a. ID);
```

```
Customer a = (Customer) eq.FindByID(a. ID);
```

Find方式，直接从数据库加载，然后查缓存，如果缓存有，用缓存对象替换；

```
Customer.Finder.Find("ID = @ID");
```

```
eq.Find("ID = @ID");
```

- 返回实体集合

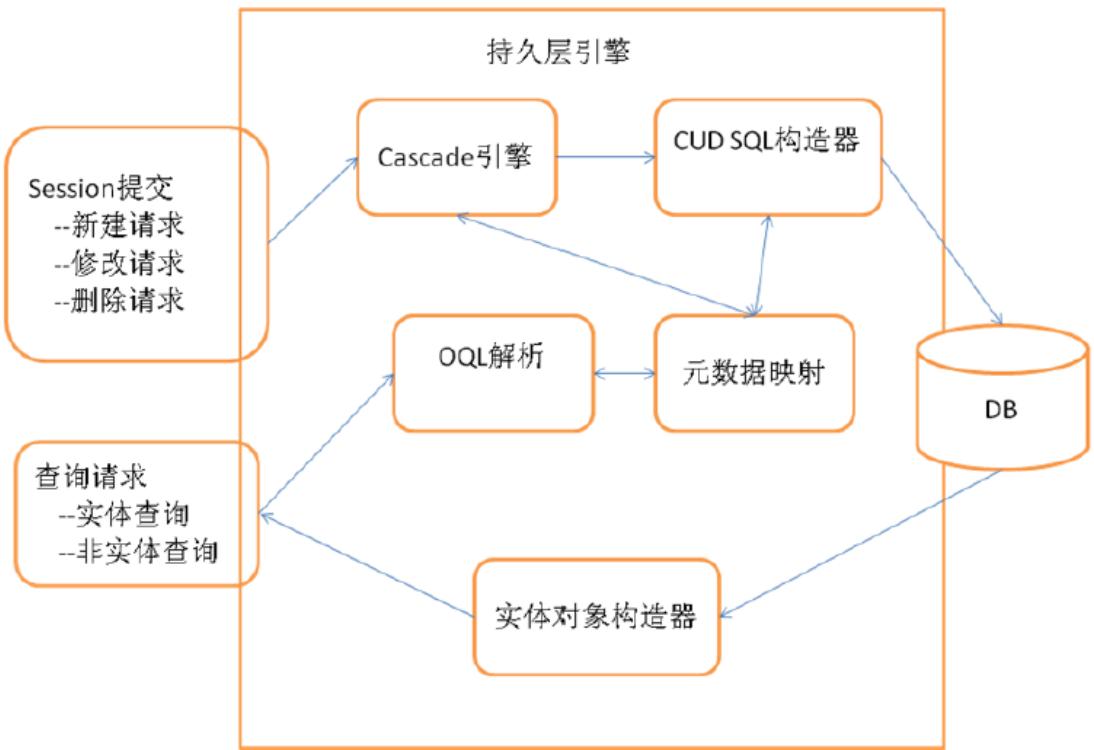
- 条件OQL方式：

```
q = new EntityQuery(Customer.EntityRes.BE_FullName);
```

```
q.FindAll("ID >1 ");
```

4) EntityViewQuery

返回临时表（常用于报表，不提供后台使用）



4、 实体资源

1) 实体资源的访问

```

//取实体ID的资源
string str= Customer.EntityRes.GetResource(Customer.EntityRes.ID);
//取实体全名的资源
str= Customer.EntityRes.GetResource(Customer.EntityRes.BE_Name);
//取枚举的资源
str= Customer.EntityRes.GetResource(Customer.EntityRes.Name);
str= Customer.EntityRes.GetResource(Customer.EntityRes.ExtEnum);
str= Customer.EntityRes.GetResource(Customer.EntityRes.ExtEnum_ExtNum1);
//取属性类型的资源
str= Customer.EntityRes.GetResource(Customer.EntityRes.PropertyType);
str= Customer.EntityRes.GetResource(Customer.EntityRes.PropertyType_PropertyType1);

```

2) 为什么要使用实体资源类?

- 强类型的方式访问属性名;
- 减少手动的误差
- 提高程序健壮性

- 举例

```
EntityDataQuery q = new EntityDataQuery(Customer.EntityRes.BE_FullName);
EntityDataQuery q = new EntityDataQuery("UFIDA.U9.CBO.SCM.Customer.Customer");
q.Select(Customer.EntityRes.Name);
q.Select("Name");
q.Parameters.Add(new UFSOFT.UBF.PL.OqlParam("1"));
object value = q.FindValue(Customer.EntityRes.Bargain+"=@"+
Customer.EntityRes.Bargain);
object value = q.FindValue("Bargain=@ Bargain");
```

5、 实体校验

1) 实体本身的业务校验

- 通过在实体Extend代码中的OnValidate方法中定义校验业务逻辑来实现

2) 平台统一校验

- 非空校验
- 引用对象的存在性校验（关联实体，枚举对象）
- 敏感字段校验

6、 事件扩展方法

1) 实体的事件在子类上会自动生成事件扩展方法

OnSetDefaultValue(), OnInserting(), OnInserted(), OnUpdating()

OnUpdated(), OnDeleting(), OnDeleted()

这些方法提供了编写实体业务逻辑的入口

2) 扩展方法的一些规则

实体的 **OnSetDefaultValue** 方法只关注对自身和下级的影响，如果有涉及到兄弟行或父实体的逻辑，需要移交到父实体的 **OnSetDefaultValue**

在后事件（**OnInserted/ OnUpdated/ OnDeleted**）中不允许在当前 **session** 范围内对实体本身的属性再进行修改，但可以调用服务修改（慎用）

3) 事件顺序

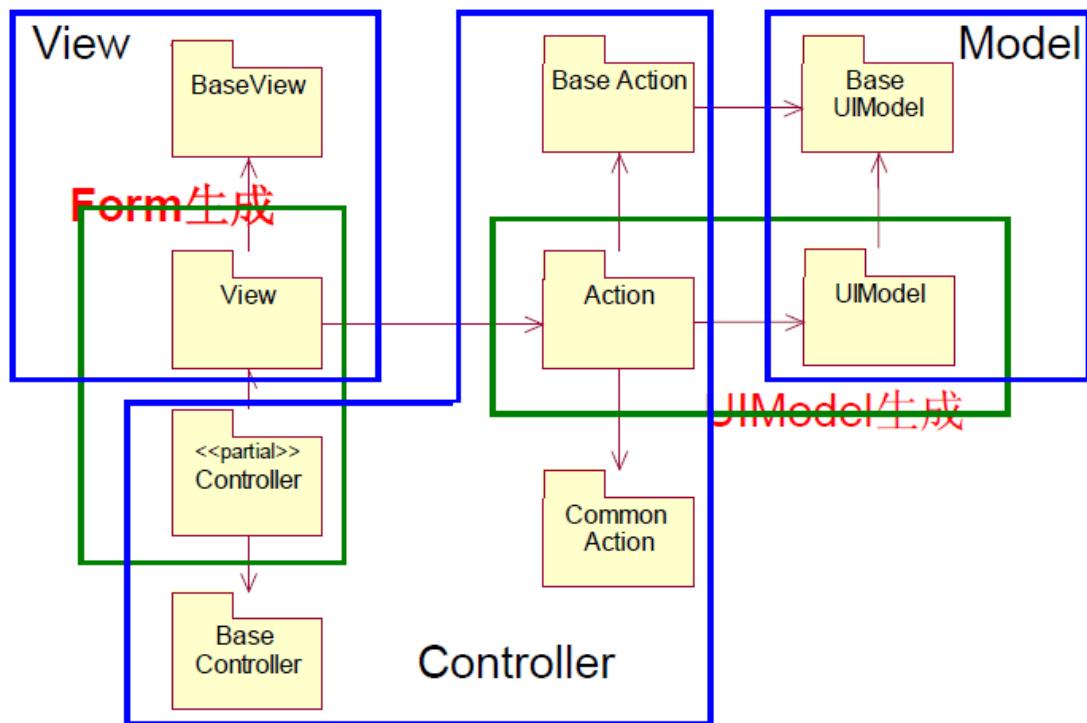
7、 U9 平台异常介绍

- 属性空效验异常：
`AttrNotNullableException`: 显示为 **XXX 不能为空, 请录入**;
例如：部门界面的[编码]字段不能为空，但是为空时保存了，则错误信息为‘**编码不能为空, 请录入**’
- 权限异常：
`AuthorityAddException`: 显示为**实体 XXX 未授权, 不能新增记录**;
`AuthorityDeleteException`: **实体 XXX 未授权, 不能删除记录**;
`AuthorityUpdateException`: **实体 XXX 未授权, 不能修改记录**;
例如：**实体收款单未授权, 不能新增记录**。
- 引用检查异常：
`CascadeDeleteException`: **当前 XXX 已被使用, 不能删除**;
`CascadeModifyException`: **当前 XXX 已被使用, 不能修改**;
例如：部门 a 已经被业务员甲引用，则删除部门 a 时，出错信息为‘**当前部门已被使用, 不能删除**’。
- 业务主键重复异常：
`SqlUniqueKeyException`: **XXX 已存在, 请重新录入**;
例如：部门档的主键是组织+编码，则出错信息为‘**组织+编码已存在, 请重新录入**’。
- 引用有效性异常：
`RefObjectInValidException`: **XXX 不存在, 请重新录入**;
例如：业务员甲引用了部门 a，但是在保存业务员甲时，部门 a 已经被删除了，这时出错信息是‘**部门不存在, 请重新录入**’。

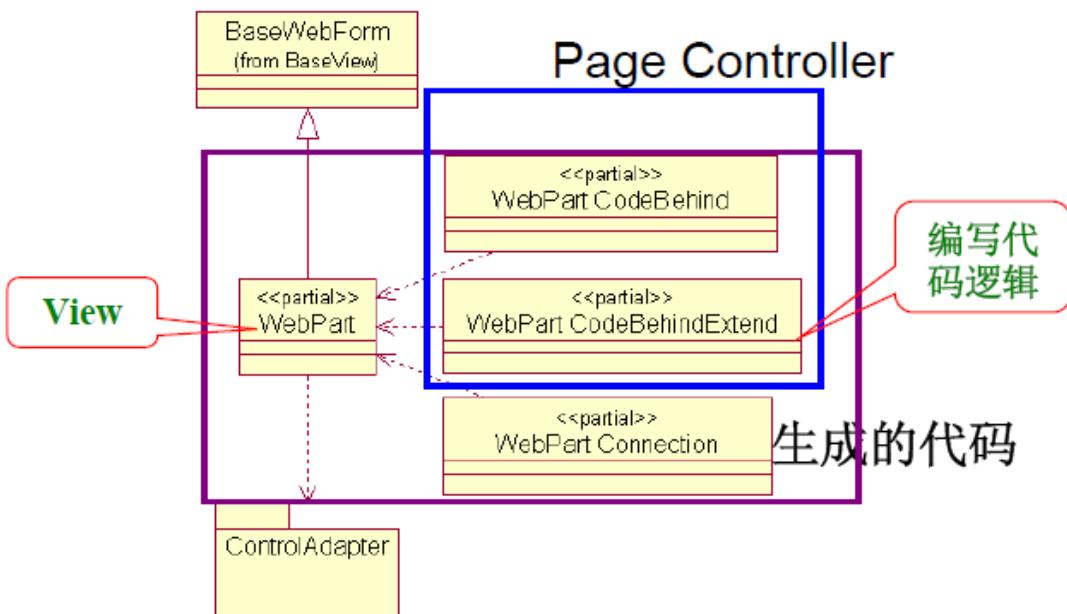
二、 前台组件编程

1、 关系图

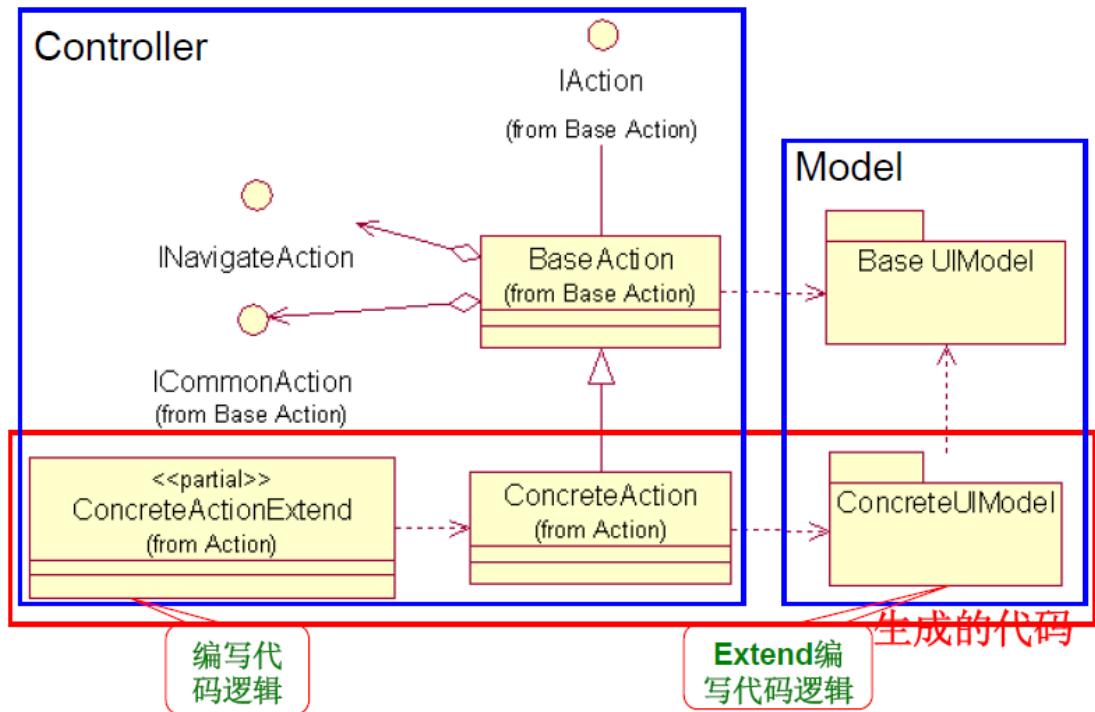
- **UI** 生成代码框架与 **MVC** 的关系



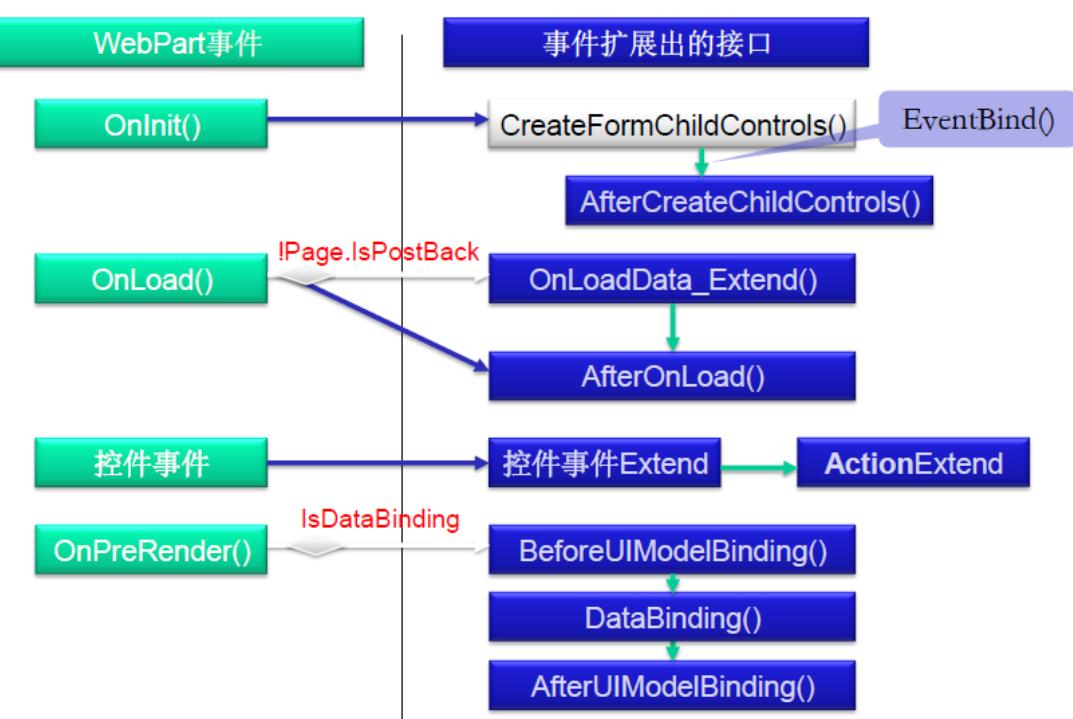
- **Form** 生成的代码框架结构



- **UIModel** 生成的代码框架结构



2、UI代码写入时机



UIForm扩展方法	可写的逻辑
AfterCreateChildControls()	创建控件树，创建关联控件
OnLoadData_Extend()	非PostBack状态下创建控件树，加载数据
AfterOnLoad()	创建控件树的最后机会，也可以加载数据，修改控件属性
事件Extend()	事件响应业务逻辑
BeforeUIModelBinding()	修改模型数据最后的机会，否则不会反映到控件上，修改控件属性
AfterUIModelBinding()	修改控件属性
AfterEventBind()	绑定事件
ActionExtend()	默认通用逻辑，可扩展
ModelExtend()	对Model中视图的修改，如默认值

3、 UIModel编程

1) 如何访问UIModel?

- 在Action基类中，有CurrentModel属性，可以在某个ActionExtend方法中这样访问 UIModel: this.CurrentModel;
- 在WebPart的CodeBehind中访问this.Model，得到强类型的UIModel；
- 通过UIModel，即可以以强类型或弱类型的方式访问得到UIView和Link集合；可以访问UIModel的属性、UIModel下UIView的属性、以及UIField属性；
- UIView是数据的具体对象与接口：OQIString、DefaultFilter、对应EntityFullName、ParentLink、Childlinks、ParentView、FocusedRecord、FocusedIndex、FocusRecord、Records等；
- UIView可以强名访问字段，也可以弱类型访问字段；

2) UIModel数据收集与界面绑定

- 对于Action操作，可以在设计期设置该Action操作是否要收集数据和绑定数据；
- BaseWebForm实现IPart接口，有DataCollect()、DataBanding()方法和IsDataBanding属性，负责收集数据、界面绑定；
- 可以在WebPart的CodeBehindExtend代码的相应方法中编程，调用this.DataCollect()、this.DataBanding()等方法，完成数据收集与绑定；

3) 操作UIModel

- **UIModel赋值、赋默认值和其他属性**

在*ModelExtend.cs文件中AfterInitModel()方法中设置字段的defaultValue

在AfterInitModel()中完成对UIModel的默认值的设置：

```
public override void AfterInitModel()
```

```
{
```

```
//默认值
```

```
this.Organization.FieldEffective_EffectiveDate.DefaultValue=
```

```
PlatformContext.Current.DateTime;
```

```
this.Organization.FieldIsLegacyOrg.DefaultValue= true;
```

```
}
```

也可以在Action方法中直接对UIModel的字段赋值、赋默认值；方式同上；

- 清空主Form的Card容器界面

在卡片界面做“新增”等操作时，需要增加一条空Record，这条空记录与界面完成绑定，达到清空界面的效果，例如，清空主Form的Card容器界面的处理：

```
int index = this.MainView.FocusedIndex;  
IUIRecordnewRecord= this.MainView.NewUIRecord();  
if(this.MainView.RecordCount== 0 || index < 0)  
    index = 0;  
this.MainView.Records.Insert(index, newRecord);  
this.MainView.FocusedRecord= newRecord;
```

- UIView与Record的处理：

```
//UIView从DataTable、UIView中复制  
this.MainView.FromDataTable(DataTabletable);  
this.MainView.FromUIView(UIViewview);  
//复制到DataTable  
this.MainView.ToDataTable(DataTabletable);  
//UIRecord与DataRow复制  
UIRecord.FromDataRow(DataRowdataRow);  
UIRecord.ToDataRow(DataRowdataRow);  
//获取焦点记录索引  
intindex = this.MainView.FocusedIndex;  
//获取焦点记录  
IUIRecordrecord = this.MainView.FocusedRecord;  
//设置焦点记录  
this.MainView.FocusedRecord= record;  
//获取被选择的记录集合  
IUIRecordCollectionrecords = this.MainView.SelectRecords;  
//获取\设置焦点字段  
this.MainView.FocusedRecord;{ 支持get、set}  
//设置记录删除标志  
UIRecord.Delete();
```

```
//移出记录
UIRecord.Remove();
//记录复制到UIRecord
UIRecord.CopyTo(IUIRecord);
//UIRecord与EntityData复制
UIRecord.FromEntityData(object entityData);
UIRecord.WriteToEntityData(object entityData);
//UIRecord与DataRow复制
UIRecord.FromDataRow(DataRowdataRow);
UIRecord.ToDataRow(DataRowdataRow);
//判断该记录是否为焦点、被选择的记录
UIRecord.IsFocused;
UIRecord.IsSelected;
```

4) 加载UIModel

修改视图中的过滤条件

- 直接修改Opah语句
修改条件语句值，增加Order By， Group By语句
- 修改过滤条件中的参数

4、 UIForm编程

1) UI控件

⑤ 容器控件

⑥ Grid控件

- 切换 Grid 可编辑状态

```
private void ChangeDataGridBrowse()
{
    Grid1.EditStatus=UFSoft.UBF.UI.ControlModel.EditStatus.Browse;
    Grid1.AllowAddRow=false;
    Grid1.AllowDeleteRow=false;
    Grid1.AllowInsertRow=false;
    Grid1.AutoEditModel=false;
}
```

⑦ Label控件

⑧ 文本框控件（含数字控件）

⑨ 日期控件

⑩ 参照控件

⑪弹性域控件

⑫枚举控件（含动态枚举）

⑬关联控件

⑭向导控件

2) 数据绑定控件的查找方法

IPart接口中的查找控件的方法定义如下：

- 通过名称得到抽象容器。
IUIViewBindingControlGetContainerByName(string name);
- 通过UIView名称得到其绑定的容器控件。
IUIViewBindingControlGetViewContainerByUIViewName(string viewName);
- 通过UIView绑定的容器，和UIField的名字，查找绑定UIField的控件
IUIFieldBindingControlGetUIFieldBindingControl(IUIViewBindingControlviewContainer, string fieldName);
- 在指定容器中查找控件
IUFCControlGetUFControlByName(IContainercontainer, string name);
 - ① 在**ActionExtend**中通过**IPart**的接口在指定容器中查找控件
 - ② 查找到指定控件后，然后修改属性即可。

3) 通用的CRUD的Action

可以在Action中调用通用CRUD Action，不过目前都是在Action模板写好了通用CRUD Action调用，除非有特殊需求，否则还是使用现成的通用CRUD Action调用模板；

4) 在状态条中显示信息

- 做法：

在UFSoft.UBF.UI.Engine.Runtime.dll位于UFSoft.UBF.UI.Engine命名空间的UIEngineHelper类下有静态方法：

```
/// <summary>
/// 在客户端的状态栏中显示指定的信息
/// </summary>
/// <paramname="page">用户控件页面</param>
/// <paramname="message">在客户端状态栏中要显示的信息</param>
public static void ShowWindowStatus(IPartform, string message)
```

- 示例：

在ActionExtend中，需要引用UFSoft.UBF.UI.Engine.Runtime.dll，代码如下：

```
UIEngineHelper.ShowWindowStatus(this.CurrentPart, Resource);
Resource 代表资源信息；
```

5) 在ActionExtend中调用专用BP Agent

- 做法：

在ActionExtend所在的项目中引用BPAgent的dll
请注意，按照我们的架构设计，在ActionExtend只能访问BPAgent

- 示例：

在ActionExtend所在的项目中引用BPAgent的dl

增加命名空间的引用，如

```
using UFSoft.UBF.Alert.Proxy;
```

写调用BP Agent的代码，如下

```
StartAlertBPProxybp= new StartAlertBPProxy();
```

```
bp.AlertKey=。 。 。
```

```
bp.Do();
```

6) 状态编程

在Action的基类中有接口取当前的State，并可以给CurrentState赋值和取值。

```
public UIState CurrentState{ get; }
```

示例：比如说Form1弹出Form2，如果传递比较大的数据时，如传Form1把自己的整个UIModel传递给Form2，可以使用CurrentState进行传递，做法如下：

- 在Form1弹出Form2的ActionExtend中，给CurrentState赋值：

```
this.CurrentState["MyUIModel"] = this.CurrentModel;
```

- 在Form2的某个ActionExtend中取出传递的UIModel：

```
IUIModel uiModel=this.CurrentState["MyUIModel"];  
if(uiModel!=null)  
{  
    .....  
}
```

这种方式最后要自己手工清除掉缓存的数据，比较麻烦。**推荐的做法：**在UIModel中添加一个属性，来缓存，系统会在适当的时机自动清除缓存。

7) 弹出Form

在*ActionExtend.cs 中实现，在 Form1 中弹出 Form2。

⑯ 页面共享UIModel

Form1 弹出 Form2，Form1 和 Form2 共享 UIModel，只要在 Form1 中调用如下代码就能弹出 Form2，并且把 Form1 当前的 UIModel 传过去。

```
This.CurrentPart.ShowModalDialog(PartID,"Form2","400","500",This.CurrentPart.TaskID.ToString());
```

⑯ 页面不共享UIModel

Form1 弹出 Form2，Form1 和 Form2 不共享 UIModel，只要在 Form1 中调用如下代码就能弹出 Form2，Form2 自会加载自己的 UIModel。

```
This.CurrentPart.ShowModalDialog(PartID,"Form2","400","500",null);
```

⑰ 参数传递（不共享UIModel）

Form2 有参数 ID，参数 ID 和这个 Form2 对应的 UIModel 的主 UIView 的过滤条件中的参数 ID 有映射关系，只要在 Form1 中调用如下代码就能弹出 Form2，并把参数 ID 传给 Form2，Form2 自会根据过滤条件加载自己的 UIModel。

- Form1 中代码，定义参数：

```
System.Collections.Specialized.NameValueCollection query = new NameValueCollection();  
query.Add("ID", "1002345");  
this.CurrentPart.ShowModalDialog(PartID,"Form2","400","500", null, query);
```

-
- Form2 中代码，响应参数：

➤ 弹出页面中未定义过滤条件时：

```
public void AfterOnLoad()
{
    this.Model.CarHonner.Clear(); // 删除原有数据
    this.Model.CarHonner.CurrentFilter.OPath = "ID=" + CurrentPart.NameValues["ID"].ToString() + "";
    this.Action.CommonAction.Load(this.Model.CarHonner); // 重新加载数据
}
```

➤ 若在弹出页面的 UIModel 中已定义窗体参数 @ItemID，并设置过滤条件 ID=@ItemID。

```
private void LoadCustData()
{
    long ItemID=long.Parse(CurrentPart.NameValues["ItemID"].ToString());
    IUIParameter idParameter=this.CurrentModel.ItemMaster.DefaultFilter.Parameters["@ItemID"];
    idParameter.DataValue=itemID;
    CommonAction.Load();
}
```

注：也可通过状态值实现不同窗体间参数传递，如：this.CurrentState["ID"] = 1002345。

⑯ 数据交换

- Form1 弹出 Form2，Form1 把数据传递给 Form2：

```
string id = (string) this.CurrentPart.NameValues["ID"];
string oldOpath=string.Empty;
if (string.IsNullOrEmpty(id))
{
    oldOpath = this.CurrentModel.ApprovalResult.DefaultFilter.OPath;
    this.CurrentModel.ApprovalResult.DefaultFilter.OPath = "";
}
this.OnLoadData_DefaultImpl(sender, e);
if (!string.IsNullOrEmpty(oldOpath))
{
    this.CurrentModel.ApprovalResult.DefaultFilter.OPath = oldOpath;
}
if (!string.IsNullOrEmpty(id))
{
    this.CurrentModel.ApprovalResult_Process_Action.FocusedRecord
    =(ApprovalResult_Process_ActionRecord)this.CurrentModel.ApprovalResult_Process_Action.Fin
    dRecordByFieldValue(this.CurrentModel.ApprovalResult_Process_Action.FieldID.Name, id);
    if(this.CurrentModel.ApprovalResult_Process_Action.FocusedRecord!=null)
    {
        this.CurrentModel.ApprovalResult_Process.FocusedRecord =
    }
```

```
(ApprovalResult_ProcessRecord)
this.CurrentModel.ApprovalResult_Process.FindRecordByFieldValue(this.CurrentModel.ApprovalResult_Process.FieldID.Name,this.CurrentModel.ApprovalResult_Process_Action.FocusedRecord.Process);
}
}
```

- Form2 把数据传递给 Form1，通过共享 UIModel，或者通过状态进行数据传递。

8) 页面导航（非封装模式）

- 导航页面和被导航页面共享 UIModel（传参）

比如说 Page1 导航到 Page2，Page1 和 Page2 共享 UIModel，只要在 Page1 中调用如下代码就能导航 Page2，并且把 Page1 当前的 UIModel 传过去。

```
NameValueCollection query=New NameValueCollection();
Query.Add("ParentTaskID",this.CurrentPart.TaskID.ToString());
This.CurrentPart.Navigate(PageURI,query);
```

- 导航页面和被导航页面不共享 UIModel（不传参）

说 Page1 导航到 Page2，Page1 和 Page2 不共享 UIModel，只要在 Page1 中调用如下代码就能导航 Page2，并且 Page1 自会加载自己的 UIModel。

```
this.CurrentPart.Navigate(PageURI, null);
```

9) 页面导航（封装模式）

使用UFIDA.U9.UI.PDHelper.NavigateManager类完成Form的弹出和页面切换导航与非封装的方式类似，省去了不必要的参数设置。

① 页面切换导航

- 完成无参数传递的页面导航：

```
NavigateManager.NavigatePage(IPartpart, string pageId)
```

- 完成有参数的页面导航：

```
NavigateManager.NavigatePage(IPartpart, string pageId, NaviteParamterparam)
```

首先，实例化导航参数

```
NaviteParamter paramter = new NaviteParamter();
paramter.SetPageNaivgate() ;//  
.....
```

```
paramter.AddPDPPageID(this.CurrentModel.Organization.FocusedRecord.ID.  
ToString());
```

然后，进行导航

```
NavigateManager.NavigatePage(this.CurrentPart, pageID, paramter);
```

② 弹出 Form (WebPart)

□ 弹出标准尺寸的无参数 WebPart (Form) :

```
NavigateManager. ShowModelWebpart(IPart part, partId);
```

□ 弹出标准尺寸的有参数 WebPart (Form) :

```
NaviteParamter paramter = new NaviteParamter();
```

```
paramter. SetModelPopup();
```

```
.....
```

```
NavigateManager. ShowModelWebpart(IPart part,, partId, paramter);
```

□ 弹出自定义尺寸的无参数 WebPart (Form) :

```
NavigateManager. ShowModelWebpart(IPart part, string partId, int width, int height)
```

□ 弹出自定义尺寸的有参数 WebPart (Form) :

```
NaviteParamter paramter = new NaviteParamter();
```

```
paramter. SetModelPopup();
```

```
.....
```

```
NavigateManager. ShowModelWebpart(IPart part, string partId, int width, int height, NaviteParamter paramter)
```

③ 导航参数设置：NaviteParamter

● 关键字名值对 ShowType:

PartShowType.ModalRef : 参照弹出

PartShowType.NavigateForm : 预留;

PartShowType.NavigatePage : 页面导航

PartShowType.Normal : 预留;

PartShowType.ShowModal : Post 模态弹出

PartShowType.TitleLink : TitleClick 模态弹出;

是记录WebPart 展现方式是“PopUp 弹出”还是“Navigation 页面切换”;

通过NavigateManager 的导航与弹出缺省设置，也可以New 导航参数实例进行设置，例如：

```
NaviteParamter paramter = new NaviteParamter();
```

```
paramter. SetPageNaivgate(); 设置为页面导航;
```

或则

```
paramter. SetModelPopup(); 设置为弹出窗口;
```

● 关键字名值对PDDataStatus: Edit、New、Browse

PDDataStatus 是记录数据状态是“Edit 编辑”还是“New 新增”、还是“Browse 浏览”;

根据NavigationStatus 和PDDataStatus 来决定按钮等状态;

通过New 导航参数实例进行设置，例如：

```
NaviteParamter paramter = new NaviteParamter();
```

```
paramter. SetNewStatus(); 设置为新增状态;
```

```
paramter. SetEditStatus(); 设置为编辑修改状态;
```

```
paramter. SetBrowseStatus (); 设置为浏览状态;
```

-
- 关键字名值对 PDPageID
PDPageID 是数据记录的 ID，根据它唯一查找定位数据；
通过 New 导航参数实例进行设置，例如：
NaviteParamter paramter = new NaviteParamter();
paramter.AddPDPageID(string id); 传递一条记录 ID;
paramter.AddPDPageIDs(ArrayList ids); 传递多条记录 ID;
通过 NavigateManager 的 IsMorePDDataID(IPart part) 方法判断是否有多条记录
通过 NavigateManager 的 IsSinglePDDataID(IPart part) 方法判断是否有一条记录

三、 公共API

1、 组件端的上下文（后台）

- ◆ 所属 DLL
UFIDA.U9.Base.BaseBE.dll
- ◆ 上下文实体全名
UFIDA.U9.Base.Context
- ◆ 上下文内容
 - //当前用户登录组织
UFIDA.U9.Base.Context.LoginOrg;
 - //当前的登录用户 ID
UFIDA.U9.Base.Context.LoginUserID; // userID = PlatformContext.Current.UserID;
 - //当前登录用户
UFIDA.U9.Base.Context.LoginUser;
 - //当前用户的登录时间
UFIDA.U9.Base.Context.LoginDateTime;

2、 UI端的上下文（前台）

- ◆ 引用 DLL
UFIDA.U9.UI.PDHelper.dll
- ◆ 上下文实体全名
UFIDA.U9.UI.PDHelper.PDContext
- ◆ 上下文内容
 - //当前登录用户 ID/Code/Name
UFIDA.U9.UI.PDHelper.PDContext.Current.UserID;
 - UFIDA.U9.UI.PDHelper.PDContext.CurrentUserCode;
 - UFIDA.U9.UI.PDHelper.PDContext.Current.UserName;
 - //当前登录组织参照(包含 ID/Code/Name)

```
UFIDA.U9.UI.PDHelper.PDContext.Current.OrgRef;
//当前用户登录的时间
UFIDA.U9.UI.PDHelper.PDContext.Current.LoginDateTime
```

3、 获取系统参数

1) 行号处理

- ◆ 系统提供公共 BP
- ◆ 引用 DLL
 - UFIDA.U9.Base.BaseBP.Agent.dll
 - UFIDA.U9.Base.BaseBP.Deploy.dll
- ◆ BP 全名
 - UFIDA.U9.Base.Profile.Proxy.GetProfileValueProxy
- ◆ 查询参数
 - select Code,Name,Description,* from Base_Profile,Base_Profile_Trl where Base_Profile.ID=Base_Profile_Trl.ID order by Base_Profile.Code
- ◆ 实现
 - ((IAutoRowNo)this.DataGrid5.Columns[1]).Sequence = true;
 - ((IAutoRowNo)this.DataGrid5.Columns[1]).SequenceStep = LineNoStep;
 - ((IAutoRowNo)this.DataGrid5.Columns[1]).SequenceStart = LineNoStep;
- 代码放置位置: *WebPartExtend.cs AfterCreateChildControls
- private UFIDA.U9.Base.Profile.PVDTOData GetBaseProfile(string profileCode){
 Base.Profile.Proxy.GetProfileValueProxy getProfile = new
 UFIDA.U9.Base.Profile.Proxy.GetProfileValueProxy();
 getProfile.ProfileCode = profileCode;
 getProfile.ProfileOrg = long.Parse(PDContext.Current.OrgID);
 getProfile.ProfileUser = long.Parse(PDContext.Current.UserID);
 UFIDA.U9.Base.Profile.PVDTOData obj = getProfile.Do();
 return obj;
}
//方法调用
// UFIDA.U9.CBO.SCM.Util.SCMPProfile. SYS_SysLineNo
UFIDA.U9.Base.Profile.PVDTOData pVTDOData = this.GetBaseProfile("SysLineNo");
int DocLineNoStep = Convert.ToInt32(pVTDOData.ProfileValue);

2) 获取当前组织的法人组织

- ◆ 所属 DLL: UFIDA.U9.CBO.SCMBP
- GetLegacyOrgFromOrg
- ◆ 调用示例

```
GetLegacyOrgFromOrgProxy bp = new GetLegacyOrgFromOrgProxy();
bp.Org = PDContext.Current.OrgID;
bp.App = UFIDA.U9.UI.BusinessHelper.ApplicationExtend.INV;
long LegacyOrgID = bp.Do();
```

3) 获取当前组织的本位币

■ 所属 DLL: **UFIDA.U9.CBO.SCMBP**

■ **GetDefaultSOBCurrencyFromOrg**

■ 调用示例:

```
SOBCurrencyDTOData SCDTO = new SOBCurrencyDTOData();
GetDefaultSOBCurrencyFromOrgProxy CostCurrency = new
GetDefaultSOBCurrencyFromOrgProxy();
CostCurrency.App = ApplicationExtend.INVKey;
CostCurrency.Org = PDContext.Current.OrgRef.ID;
SCDTO = CostCurrency.Do();
if (SCDTO != null)
{
    SobCurrency = SCDTO.Currency;
    SOBID = SCDTO.SOBI;
}
```

4) 获取当前账簿的会计期间

■ 所属 DLL: **UFIDA.U9.Base.BaseBP**

■ **GetSOBAccountPeriod**

■ 调用示例:

```
UFIDA.U9.Base.SOBI.Proxy.GetSOBAccountPeriodProxy getSOBAccountPeriodProxy = new
UFIDA.U9.Base.SOBI.Proxy.GetSOBAccountPeriodProxy();
getSOBAccountPeriodProxy.SOBI = PDContext.Current.PrimarySOBRef.ID;
getSOBAccountPeriodProxy.Date = PDContext.Current.DateTime;
SOBAccountingPeriodData data = getSOBAccountPeriodProxy.Do();
if (data != null)
{
    Record.SOBAccountPeriod = data.ID;
    Record.SOBAccountPeriod_Code = data.Year.ToString() + "-" + data.Code;
}
```

5) 数量计算相关

所属 DLL: UFIDA.U9.CBO.PubBP

- ❑ 批量获取计价单位主副单位转换率
GetItemUOMRatioV2
- ❑ 批量获取计价单位间转换率
GetUOMRatioV2
- ❑ 批量获取组内转换率
GetUOMConvertRatioDTOInClass
- ❑ 批量获取组间基准单位间转换率
GetUOMConvertRatioDTOBetweenClass

6) 获取币种间的汇率

所属 DLL: UFIDA.U9.Base.BaseBP

❑ **GetDateExchangeRateProxy**

```
int ExRateType =  
UFIDA.U9.CBO.SCM.Util.SCMPProfile.Inv_Inv_DefaultExchangeType_Org(Base.Context.LoginOrg.Key).Value;  
GetDateExchangeRateProxy getDateExchangeRateProxy = new GetDateExchangeRateProxy();  
getDateExchangeRateProxy.Date = Base.Context.LoginDate; ;  
getDateExchangeRateProxy.ExchangeRateType = ExRateType;  
getDateExchangeRateProxy.FromCurrency = docDTO.FC.ID;  
getDateExchangeRateProxy.ToCurrency = docDTO.TC.ID;  
decimal rate = getDateExchangeRateProxy.Do();
```

四、 UI高级开发

1、 确认框

需引用 D:\UFIDA\UBFV2\U9. V0B. Product. UBF\UBFStudio\Runtime\UFIDA. U9. UI. PDHelper. dll

代码放置位置: *WebPartExtend.cs 的 AfterCreateChildControls()

- ❑ 删除确认框
PDFFormMessage.ShowDelConfirmDialog(this.Page, PDResource.GetDeleteConfirmInfo(), "",
this.BtnDelete);
- ❑ 保存确认框
PDFFormMessage.ShowConfirmDialog(this.Page, "保存确认！", "确认窗口", this.BtnSave);
- ❑ 即时确认框
this.ShowAlertMessagesAtClient("AAAAA");

2、 状态栏提示

代码放置位置： *WebPartExtend.cs 的 AfterCreateChildControls()

◆ 提示自定义消息

```
PDFormMessage.ShowWindowStatus(this.Page, "当前单据已审核！");
```

或：

```
this.ShowWindowStatus("当前单据已审核！");
```

◆ 提示系统公共消息

如状态栏显示"完成数据加载,进入编辑状态",可在 Action 中设置。

```
CurrentPart.ShowWindowStatus(UFIDA.U9.UI.PDHelper.PDResource.InitUpEditSucessInfo);
```

3、 显示错误信息

在页面左上角显示感叹号信息供查看。

```
UFIDA.U9.UI.PDHelper.PDFormMessage.ShowAlertDialog(this, "错误XXX");
```

或：

```
try {
    this.Model.ClearErrorMessage(); // 清除原错误信息
    throw new Exception("错误XXX"); // 新的错误抛出
}
catch (Exception ex)
{
    IUIModel model = this.Model;
    this.Model.ErrorMessage.SetErrorMessage(ref model, ex);
}
```

4、 显示图片

1) 静态图片

用 UBF 的图片控件显示 Portal 目录下的图片文件。

```
this.Image0.ImageUrl = @"~\bmp\b" + docno.Substring(docno.Length - 1, 1) + ".BMP";
```

2) 动态图片

不用 UBF 自带的图片控件，而是通过在 UI 中加载子控件，实现加载数据库中保存的图片。

1) 定义BE实体字段

在 BE 中定义一个 Picture 字段(类型: 二进制流, 长度: 9999), 构造后在数据库中查询为 varbinary(max) 类型。

2) 图片弹出窗定义

```
*PicUIFormWebPartCodeBehindExtend.cs。
IUFRichImage itemPIC;
public void AfterCreateChildControls()
{
    itemPIC = new UFWebRichImageAdapter();
    this.itemPIC.Parent = this.Card0;
    CommonBuilder.GridLayoutPropBuilder(this.Card0, this.itemPIC, 150, 70, 0, 1, 3, 8,
"100"); // 加入到页面中 , 130, 70, 2, 8, 1, 3, "100"
    this.Card0.Controls.Add(this.itemPIC);
    UIControlBuilder.BuilderUIFieldBindingControl(this, this.itemPIC, "False",
"Picture", this.Model.OrderDoc, this.Model.OrderDoc.FieldPicture, "OrderDoc");
    ScriptManager.GetCurrent(this.Page).EnablePartialRendering = false;
    ID2.Visible = false;
}

public void AfterUIModelBinding()
{
    this.BtnClose.IsClientClose = true;
}

private void BtnOk_Click_Extend(object sender, EventArgs e) // 响应确定按钮
{
    CloseDialog(true);
    CurrentState["OrderPicRefresh"] = true;
}
```

3) 主Form加载子控件

在主 WebPartCodeBehindExtend.cs 的 AfterCreateChildControls 中加载子控件。

```
private IUFRichImage itemPIC;
public void AfterCreateChildControls()
{
    this.itemPIC = new UFSOFT.UBF.UI.WebControlAdapter.UFWebRichImageAdapter();
    this.itemPIC = new UFWebRichImageAdapter();
```

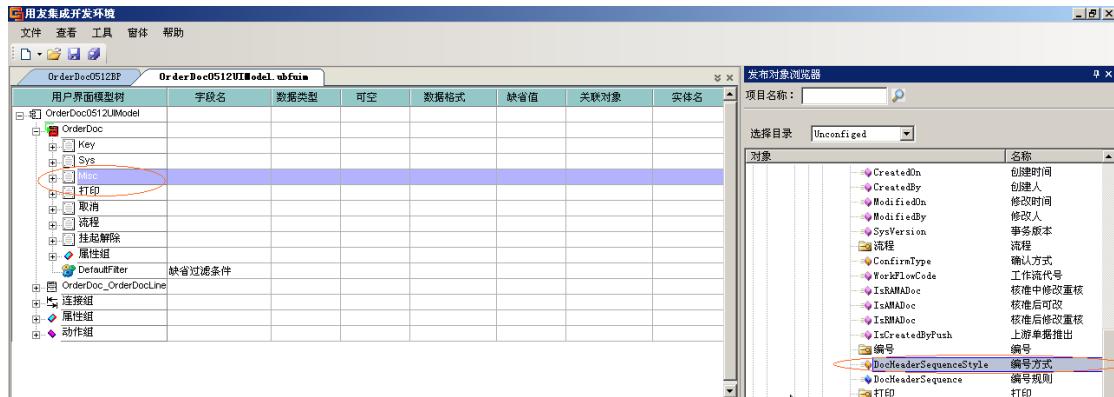
```
        this.itemPIC.ID = "MyPic";
        this.itemPIC.LabelID = this.Label249.ID;
        CommonBuilder.GridLayoutPropBuilder(this.TabPage0, this.itemPIC, 150, 70, 8,
1, 1, 4, "100");
        this.TabPage0.Controls.Add(this.itemPIC);
        UIControlBuilder.BuilderUIFieldBindingControl(this, this.itemPIC, "False",
"Picture", this.Model.OrderDoc, this.Model.OrderDoc.FieldPicture, "OrderDoc");
        this.itemPIC.ReadOnly = true;
        try
        {
            if (CurrentState.ContainsKey("OrderPicRefresh") &&
CurrentState["OrderPicRefresh"].ToString().ToLower() == "true")
            {
                UFRichImage img = this.itemPIC as UFRichImage; // itemPIC
                img.IsNeedReload=true; // .set_IsNeedReload(true);
                CurrentState.Remove("OrderPicRefresh");
            }
        }
        catch
        {
        }
    }

    // 刷新页面
    public void AfterOnLoad()
    {
        if (CurrentState.ContainsKey("OrderPicRefresh") &&
CurrentState["OrderPicRefresh"].ToString().ToLower() == "true")
        {
            this.Action.NavigateAction.MovePageAt(null,
this.Action.CurrentModel.OrderDoc.FocusedRecord.ID);
            CurrentState.Remove("OrderPicRefresh");
        }
    }

    // 响应弹出图片窗
    private void Label249_Click_Extend(object sender, EventArgs e)
    {
        Label249_Click_DefaultImpl(sender, e);
        this.ShowModalDialog("9442fade-f9f1-4419-b84e-653a9c5bf4cc", "图片上传", "315",
"275", this.TaskId.ToString());
    }
}
```

5、使能关联

拖放编号方式至 UIModel



拖放使能关联控制至 UI，并设置触发及响应控件等。



6、UDC可扩展枚举

组件端处理

BE 模型中创建枚举时将属性[可扩展]置为 True

界面端处理

代码放置位置： *WebPartExtend.cs 的 AfterCreateChildControls()

➤ 单头：

UDC.Refresh(this, this.lblImportantLevel147, this.ImportantLevel147);

➤ 单体： Grid 中的扩展枚举，可通过绑定列名或列序号实现。

UDC.Refresh(this.DataGrid5, this.ItemCode); // 将列的唯一标识末尾的序号去掉即可

或：

UDC.Refresh(this.DataGrid5, 3); // 列序号（从 0 开始计数）

7、卡片翻页（换行）处理



注册翻页事件

```
public void AfterEventBind()
{
    //注册页签控件的导航按钮 Click 事件
    this.TabControl0.NavButtonClickEvent
        += new NavButtonClickEventHandler(TabControl0_NavButtonClickEvent);
}

实现翻页逻辑
private void TabControl0_NavButtonClickEvent(object sender, NavButtonClickEventArgs e)
{
    try
    {
        this.OnDataCollect(this);
        NavButtonHelper.NavigateTabPage(sender, e); //设置当前焦点纪录
        NavButtonHelper.NavigateGrid(this, this.DataGrid5); //设置 DataGrid,PageIndex
        //其它处理
        this.TabControl0_SelectedIndexChanged_Extend(sender, new EventArgs());
    }
    catch (Exception ex)
    {
        if (UFSoft.UBF.Exceptions.ExceptionHelper.IsUnknownException(ex, true))
        {
            throw;
        }
        IUIModel model = this.Model;
        this.Model.ErrorMessage.SetErrorMessage(ref model, ex);
    }
}
```

控制翻页按钮

在 BeforeUIModelBinding 方法中调用如下代码

```
private void SetNavButtonIsShow()
{
    for (int i = 1; i < this.TabControl0.TabPages.Count; i++)
    {
        NavButtonHelper.SetNavButtonEnableStatus(this.TabControl0.TabPages[i]);
    }
}
```

8、水印效果



注册事件

```
public void AfterUIModelBinding ()
{
    this.SetWaterMark(); //水印效果
}
```

实现逻辑

```
private void SetWaterMark()
{
    OrderDocRecord headRecord = this.Model.OrderDoc.FocusedRecord;
    if (headRecord == null) return;

    if (headRecord.Status == (int)DocStatusEnumData.Opened)
    {
        UFWaterMark mark_Opened = new UFWaterMark();
        mark_Opened.TargetControl = this.Status143;
        mark_Opened.WaterMarkType = enumMarkType.Open;
    }

    if (headRecord.Status == (int)DocStatusEnumData.Approving)
    {
        UFWaterMark mark_Approving = new UFWaterMark();
```

```
mark_Approving.TargetControl = this.Status143;
mark_Approving.WaterMarkType = enumMarkType.Approving;
}

if (headRecord.Status == (int)DocStatusEnumData.Approved)
{
    UFWaterMark mark_Approved = new UFWaterMark();
    mark_Approved.TargetControl = this.Status143;
    mark_Approved.WaterMarkType = enumMarkType.Approved;
}

if (headRecord.Status == (int)DocStatusEnumData.Closed)
{
    UFWaterMark mark_Closed = new UFWaterMark();
    mark_Closed.TargetControl = this.Status143;
    mark_Closed.WaterMarkType = enumMarkType.Colse;
}
}
```

9、自由项处理（略）

料品的形态属性为【商品】的支持 10 个自由项。可设置自由项约束关系。

1) UIModel的设计

自定义参照字段

2) UIForm的设计

自由项参照的处理

3) 自由项状态的控制

4) 料品选定后自由项的处理

5) 自由项BP调用

6) 自由项过滤条件的处理

7) 自由项Label名称的处理

自由项相互约束的处理

10、数量金额处理

◆ 数字控件的关键属性

按值舍入值

舍入规则

小数位数（静态）

小数位数（动态）

货币符号

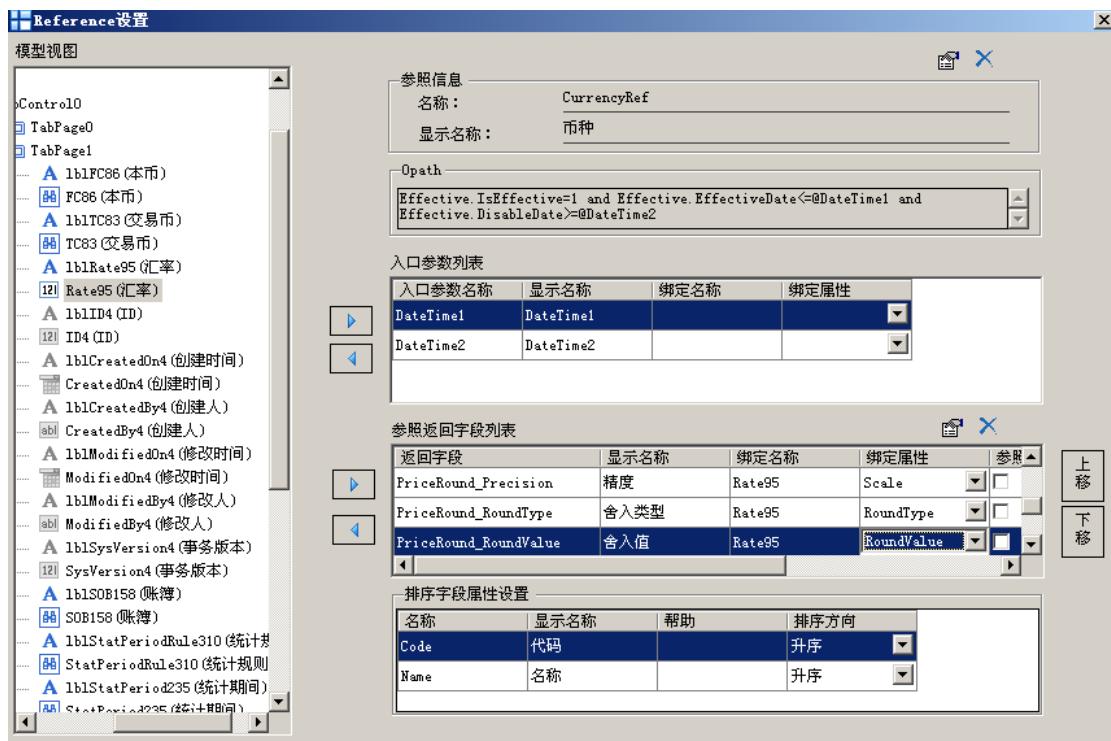
数字格式



普通属性	
按值舍入值	单价精度:舍入值
货币符号	货币符号
列宽	100
默认值	0
舍入规则	单价精度:舍入类型
是否累计	False
是否启用	True
是否为空	True
是否显示	True
是否显示0值	True
是属性	False
数据绑定	含税单价
数字格式	Currency
提示	订货单行.Misc.含税单价
唯一标识	PriceWithTax0
显示边线	True
显示格式	24.9
显示名称	含税单价
显示名称来源类型	BindingUIModel

1) 单头数字精度绑定

可直接根据币种或计量单位的返回值进行精度绑定。



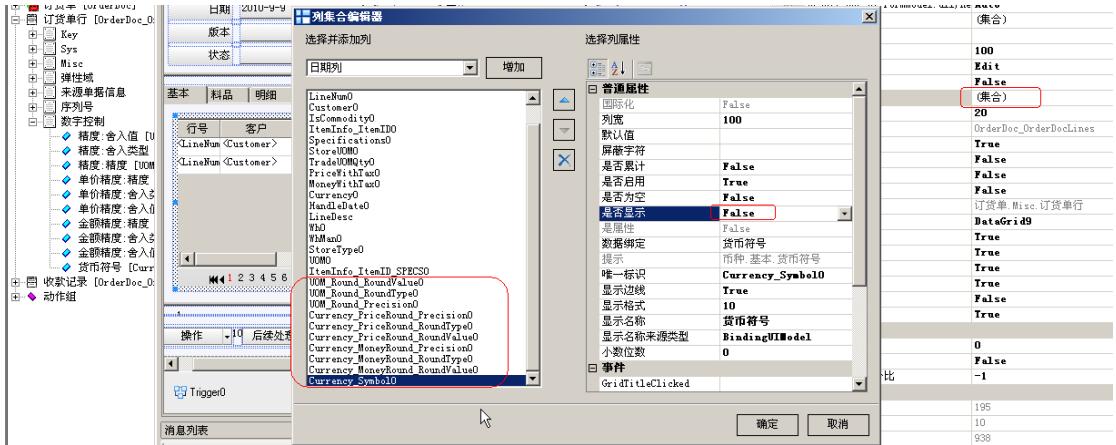
2) 单体数字精度绑定

第一步：将该 BE 下的计量单位、币种(如本币)的精度、舍入类型、舍入值等拖入 UIModel。

The screenshot shows the UIModel editor with two main panes:

- 左侧 UIModel 属性树**: 显示了 Project_Name, Task, Task_Code, Task_Name, IsCommodity, TradeUOMCity, ItemInfo_ItemID_SPECS 等项，以及数字控制、属性组、DefaultFilter 等。
- 右侧 属性映射**: 显示了 UOM 和 Currency 的属性映射。
 - UOM 属性**:
 - ID: ID
 - CreatedOn: 创建时间
 - CreatedBy: 创建人
 - ModifiedOn: 修改时间
 - ModifiedBy: 修改人
 - Round: 精度
 - Precision: 精度
 - RoundType: 舍入类型
 - RoundValue: 舍入值
 - Currency 属性**:
 - Key: Key
 - ID: ID
 - Sys: Sys
 - CreatedOn: 创建时间
 - CreatedBy: 创建人
 - ModifiedOn: 修改时间
 - ModifiedBy: 修改人
 - SysVersion: 事务版本
 - ISOCode: ISO 代码
 - Symbol: 货币符号
 - Code: 编码
 - Name: 名称

第二步：UIForm 中将精度等字段加入 Grid 中，并将其隐藏：



第三步：设置计量单位、币种参照返回精度等值。

Reference Settings

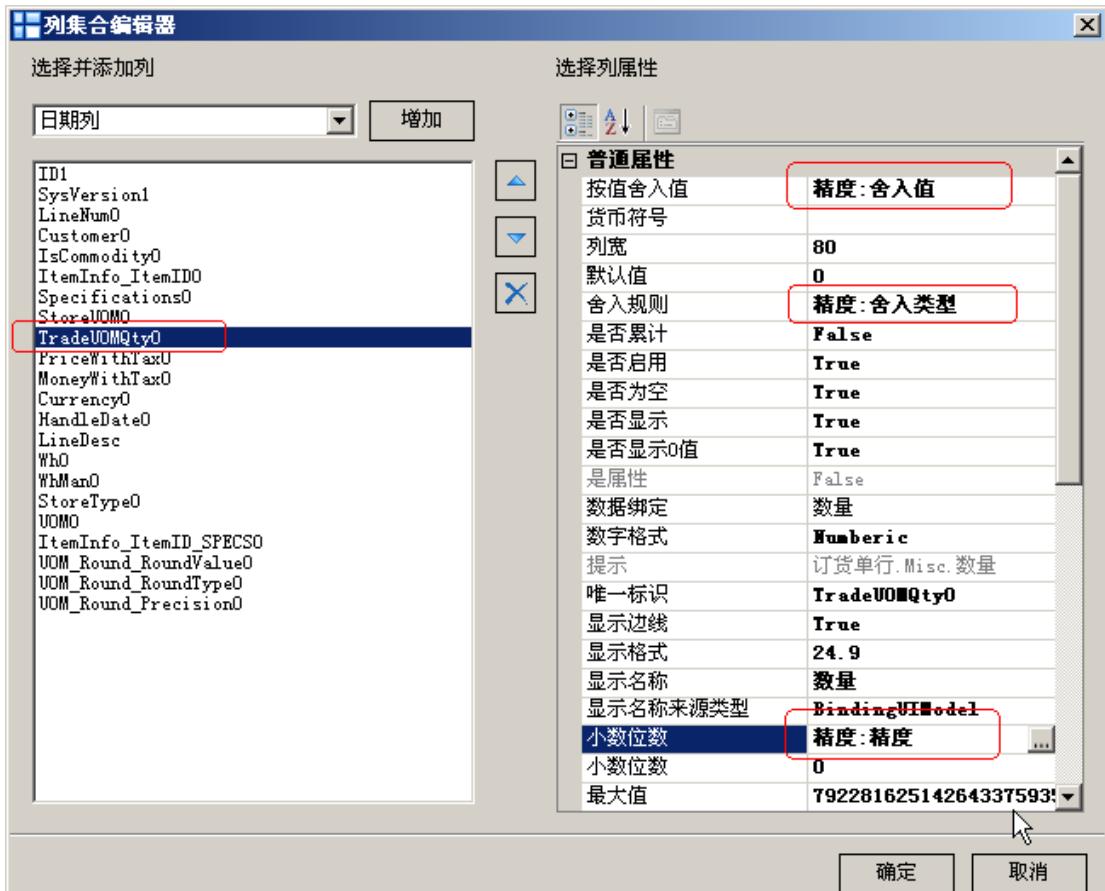
The screenshot shows the 'Reference Settings' dialog box. It includes sections for '参照信息' (Reference Information), '入口参数列表' (Input Parameter List), and '参照返回字段列表' (Reference Return Field List). A red box highlights the 'Round_Precision' field in the '参照返回字段列表' table, which is bound to the 'UOM_Round_Precision' property. Below this, there are two tables for '排序字段属性设置' (Sort Field Property Settings) and '返回字段' (Return Fields), both showing mappings between fields and their properties.

返回字段	显示名称	绑定名称	绑定属性
MoneyRound_Precision	精度	Currency_MoneyRound_Precision	Value
MoneyRound_RoundType	舍入类型	Currency_MoneyRound_RoundType	Value
MoneyRound_RoundValue	舍入值	Currency_MoneyRound_RoundValue	Value

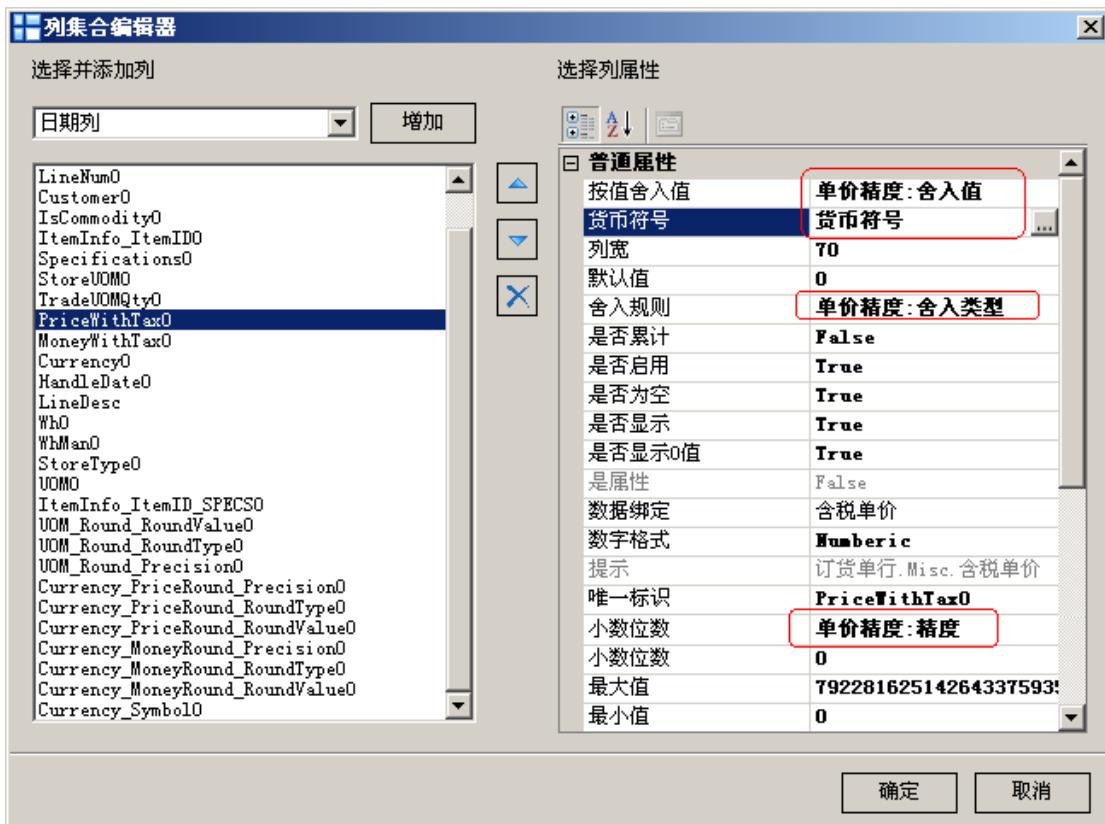
返回字段	显示名称	绑定名称	绑定属性
PriceRound_Precision	精度	Currency_PriceRound_Precision	Value
PriceRound_RoundType	舍入类型	Currency_PriceRound_RoundType	Value
PriceRound_RoundValue	舍入值	Currency_PriceRound_RoundValue	Value

第四步：分别绑定数字字段的精度、舍入值、舍入类型等。

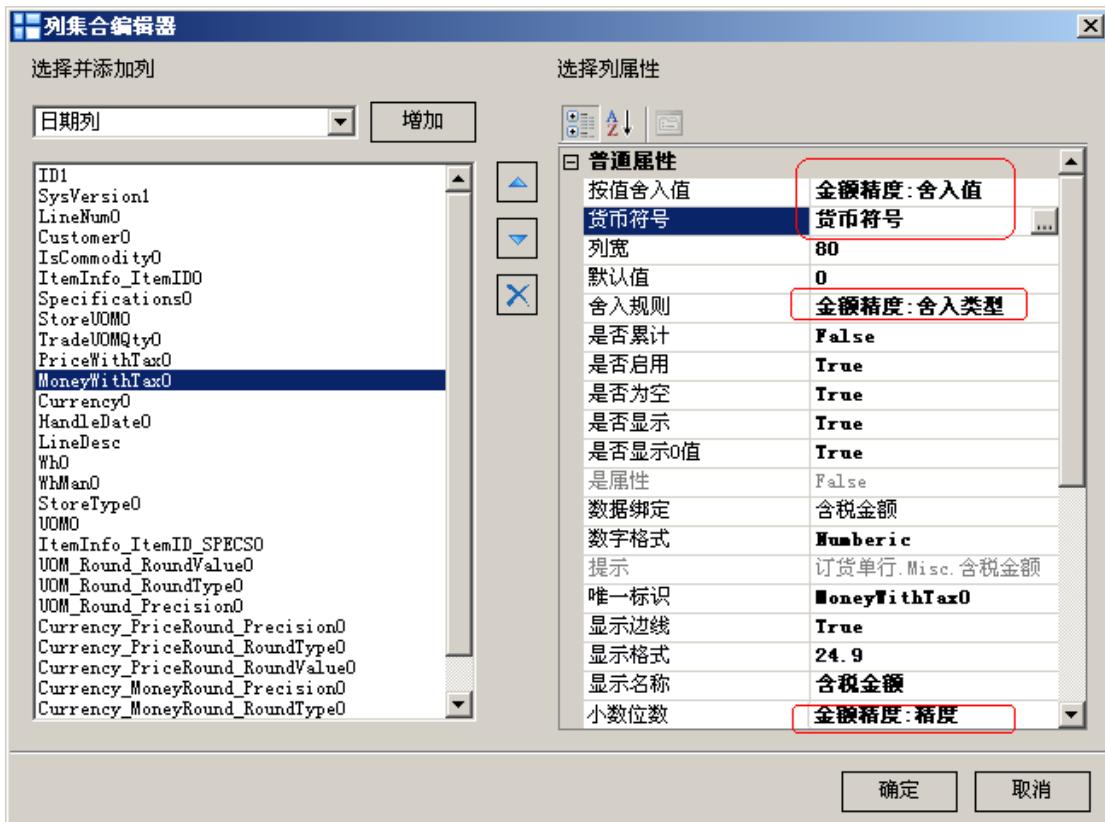
A、绑定数量字段精度：



B、绑定单价字段精度：



C、绑定金额字段精度：



11、计价单位

- 计价单位
- 计价单位副单位
- 主副单位的转换率
- 计价单位组
- 计价单位组基准单位
- 计价单位的组间转换率
- 料品的转换率策略
- 料品的特定转换率

12、UI调用BP

```
UFIDA.U9.Customer.OrderDocBP.Proxy.OrderDocApproveProxy tmpproxy = new OrderDocApproveProxy();
if (this.CurrentModel.OrderDoc.FocusedRecord != null)
{
    tmpproxy.DocKey = this.CurrentModel.OrderDoc.FocusedRecord.ID;
    tmpproxy.UISysversion = this.CurrentModel.OrderDoc.FocusedRecord.SysVersion.ToString();
```

```
tmpproxy.OperationType = 0;//提交  
tmpproxy.Do();  
}  
NavigateAction.Refresh(null); //刷新界面
```

13、卡片CallBack

1) 基础知识

① 概念

在客户端向服务器发起一个CallBack 请求，经过CallBack UI 框架处理后，从而使得开发人员能够在服务器端的CallBack 事件处理中进行服务器端编程，最后将服务器端代码的执行结果反映在客户端页面上。这样可以实现类似 ASP.NET 中 PostBack 一样的事件处理过程。

在 UI 开发过程中，CallBack 是在 AfterCreateChildControls() 时机定义的。即在页面生命周期的 Init 的最后，Init_Completed 之前。

- ❑ Ajax 技术的 U9 封装
 - ❑ 优点
 - 比 PostBack 性能高
 - 用户体验好
 - ❑ 缺点
 - 代码逻辑复杂
 - 代码不易维护
 - ❑ 常用事件
 - ✓ Grid 常用事件
 - 单元格数据改变 : OnCellDataChanged
 - 单元格获得焦点前: OnBeforeCellFocusEnter
 - ✓ 数字控件
 - ..\U9.VOB.Product.U9\Portal\htc\ Numeric.htm
 - 文件中节点 event name
 - ✓ 其他控件
- 在 D:\UFIDA\U9V2\Portal\htc 目录下可查看其事件。

② 典型的CallBack 编程

- 定义一个关联控件，指定关联控件的源控件、事件。
- 定义一个CallBack 框架，指定发起CallBack 时所需参数。
- 将刚定义的关联控件加入这个CallBack 框架中。
- 编写CallBack 服务器端处理函数。

其含义是当关联控件指定的的源控件发生特定的客户端事件后，便通过 U9 的CallBack

框架来向服务器端发起 CallBack 请求。服务器端便会执行 CallBack 处理函数，将服务器端（后台）所作的操作结果再通过 CallBack 框架回送并更新至客户端。这个回送更新的过程是 CallBack 自动完成的。

③ 典型样例代码

```
AssociationControl assoC = new AssociationControl(); //创建关联控件
assoC.SourceServerControl = this.RcvPrograme243; //指定关联控件的源控件
assoC.SourceControl.EventName = "onchange"; //指定源控件的触发事件
ClientCallBackFrm cF = new ClientCallBackFrm(); //创建一个CallBack框架
cF.ParameterControls.Add(this.BusinessDate659); //从UI携带参数的集合
cF.ParameterControls.Add(this.DocNo191); //从UI携带参数的集合
//定义后端CallBack处理函数，开发人员将在assoC_DoCustomerAction方法中写自己的业务逻辑
cF.DoCustomerAction += new ClientCallBackFrm.ActionCustomer(assoC_DoCustomerAction);
cF.Add(assoC); //将关联控件加到callback框架中
//开发人员自行编写自己的CallBack 处理函数
private object assoC_DoCustomerAction(CustomerEventArgs args)
{
    //根据CallBack 参数，借助控件适配器模型，编写相应的callback 处理逻辑
    .....
    return args; //将CallBack 计算结果返回为CallBack 框架，进而反映到客户端。
}
```

④ CallBack 编程中参数

大多数 CallBack 的时候，需要携带前台（IE 端）某些控件的客户端的值及状态至服务器端。CallBack 框架提供了这样一套参数携带的方法。CallBackFrm 的 Add() 方法可以将参数控件直接加入 CallBackFrm 中。

```
private void ChangeDate()
{
    AssociationControl assoC_3 = new AssociationControl();
    assoC_3.SourceServerControl = this.RecDate271;
    assoC_3.SourceControl.EventName = "onchange";
    ClientCallBackFrm cF = new ClientCallBackFrm();
    cF.DoCustomerAction += new ClientCallBackFrm.ActionCustomer(assoC_DoDate1Action);
    cF.ParameterControls.Add(this.PreBillLine1); //携带前台控件参数
    cF.ParameterControls.Add(this.PreBillLineN01); //携带前台控件参数
    cF.ParameterControls.Add(this.PreBillMoney1); //携带前台控件参数
    cF.ParameterControls.Add(this.RecMoney3); //携带前台控件参数
    cF.Add(assoC_3);
}
private object assoC_DoDate1Action (CustomerEventArgs args)
{
    //在CallBack 处理函数中取得CallBack 参数
}
```

```
long lPreBillLineID = Convert.ToInt64(args.ArgsHash[this.PreBillLine1.ClientID]);
int sPreBillLineNO = Convert.ToInt32(args.ArgsHash[this.PreBillLineNO1.ClientID]);
decimal dPreBillMoney = Convert.ToDecimal(args.ArgsHash[this.PreBillMoney1.ClientID]);
decimal dPreBillMoney = Convert.ToDecimal(args.ArgsHash[this.RecMoney3.ClientID]);
//以下处理自己的业务逻辑
.....
return args; //将Callback 结果回送并更新客户端
}
```

⑤ CallBack 处理函数

开发人员可以获取 **CallBack** 前台参数，编写自己业务处理逻辑，通过使用客户端控件适配器来以编写服务器端代码（C#代码）的方式来访问操作客户端控件。最后 **CallBack** 框架会将服务器端（后台）运算的结果通过 **CallBack** 框架更新至客户端（IE 端）

```
ClientCallBackFrm cF = new ClientCallBackFrm();
.....
cF.DoCustomerAction += new ClientCallBackFrm.ActionCustomer(assoC_DoCustomerAction);
.....
private object assoC_DoCustomerAction (CustomerEventArgs args)
{
    //根据CallBack参数，借助控件适配器模型，编写相应的callback处理逻辑
    //获取参数【资本额币种】客户端的值
    object oRegCKey = args.ArgsHash[this.RegisterCapitalCurrency0.ClientID];//获取参照Key
    object oRegCText = args.ArgsHash[this.RegisterCapitalCurrency0.ClientID + "_Value"];//获取参照Value
    object oRegCValue = args.ArgsHash[this.RegisterCapitalCurrency0.ClientID + "_Text"];//获取参照Text
    //获取参数【资本额】客户端的值
    object oRegCap = args.ArgsHash[this.RegisterCapital.ClientID];
    //使用【营业额币种】的客户端控件适配器，来访问和操作【营业额币种】的客户端控件实例
    UFWebClientRefControlAdapter turC = new UFWebClientRefControlAdapter(this.TurnoverCurrency);
    //设置【营业额币种】客户端控件实例的值
    turC.PKValue = oRegCKey.ToString();
    turC.CodeValue = oRegCText.ToString();
    turC.NameValue = oRegCValue.ToString();
    //将设置的值更新至客户端
    args.ArgsResult.Add(turC.ClientInstanceWithPK);
    args.ArgsResult.Add(turC.ClientInstanceWithName);
    args.ArgsResult.Add(turC.ClientInstanceWithCode);
    UFWebClientNumberAdapter tur = new UFWebClientNumberAdapter(this.Turnover);
    //设置【营业额】客户端控件实例的值
    tur.Value = oRegCap.ToString();
    //将后台结果更新至客户端
    args.ArgsResult.Add(tur.ClientInstanceWithValue);
    return args; //将返回整个CallBack计算结果返回至客户端。
}
```

2) 可实现的控制

Grid 控件对应的 Call 控件声明

```
UF WebClientGridAdapter grid = new UF WebClientGridAdapter(this.DataGrid5);
```

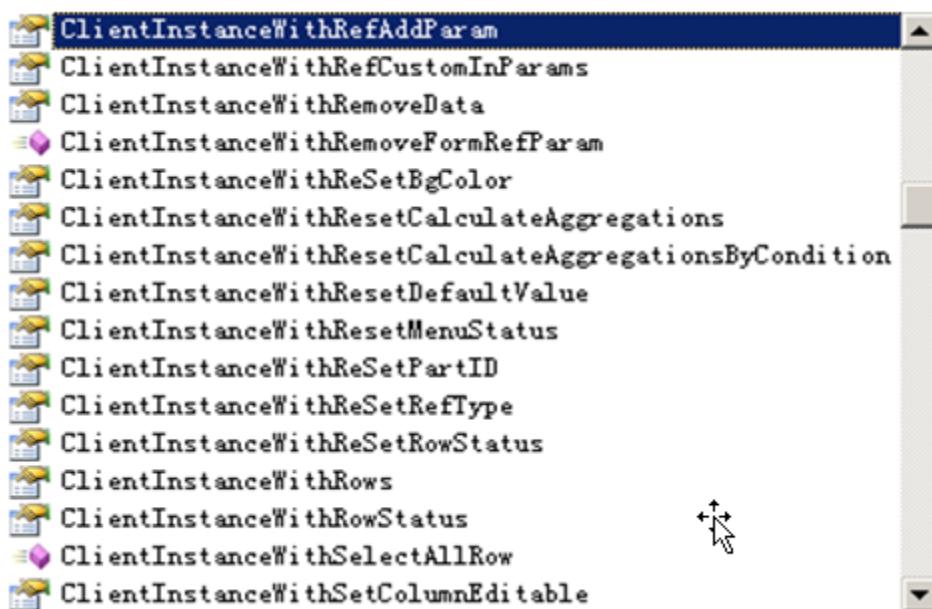
GridCell 赋值

```
args.ArgsResult.Add(grid.ClientInstanceWithValue);
```

GridCell 状态控制

```
args.ArgsResult.Add(grid.ClientInstanceWithColumnsVisbled);
```

其他控制



3) CallBack_Grid

事件注册位置

AfterCreateChildControls

事件声明

```
private void GridQtyCallBackEvents()
{
    AssociationControl assoCGrid = new AssociationControl();
    assoCGrid.SourceServerControl = this.DataGrid5;
    assoCGrid.SourceControl.EventName = "OnCellDataChanged";
    ((UF WebClientGridAdapter)assoCGrid.SourceControl).FireEventCols.Add("Count");

    ClientCallBackFrm DistributeQtyCallBackFrm = new ClientCallBackFrm();
    DistributeQtyCallBackFrm.ParameterControls.Add(this.DataGrid5);
    DistributeQtyCallBackFrm.DoCustomerAction += new
}
```

new

```

ClientCallBackFrm.ActionCustomer(GridQtyCallBackFrm_DoCustomerAction);

        DistributeQtyCallBackFrm.Add(assoCGrid);
        this.Controls.Add(DistributeQtyCallBackFrm);

    object GridQtyCallBackFrm_DoCustomerAction(CustomerActionEventArgs args)
    {

        OrderDoc_OrderDocLinesRecord           lineRecord      =
this.Model.OrderDoc_OrderDocLines.FocusedRecord;
        if (lineRecord == null) return args;

        ArrayList list = (ArrayList)args.ArgsHash[this.DataGrid5.ClientID];
        int               curIndex      =
Convert.ToInt32(args.ArgsHash[UF WebClientGridAdapter.FocusRow]);

        Hashtable hashT= (Hashtable)list[curIndex];
        UF WebClientGridAdapter grid = new UF WebClientGridAdapter(this.DataGrid5);
        object oQty = hashT["Count"];
        decimal Qty = decimal.Zero;
        if (!this.IsNotNullOrZero(oQty))

            Qty = Convert.ToDecimal(oQty);
        object oPrice = hashT["PriceWithTax"];
        decimal Price = decimal.Zero;
        if (!this.IsNotNullOrZero(oPrice))

            Price = Convert.ToDecimal(oPrice);
        //数量格式化
        int roundType = Convert.ToInt32(hashT["Currency_MoneyRound_RoundType"]);
        int roundValue = Convert.ToInt32(hashT["Currency_MoneyRound_RoundValue"]);
        int precision = Convert.ToInt32(hashT["Currency_MoneyRound_Precision"]);
        decimal Money = RoundEx(Qty * Price, roundType, roundValue, precision);
        grid.CellValue.Add(new Object[] { curIndex, "MoneyWithTax", new string[]
{ Money.ToString(), Money.ToString(), Money.ToString() } });

        args.ArgsResult.Add(grid.ClientInstanceWithValue);
        return args;
    }
}

```

4) CallBack_其他控件

CallBack 控件声明 (AfterCreateChildControls)

```

UF WebClientNumberAdapter           qty2      =
UF WebClientNumberAdapter(this.MoneyWithTax59);
格式
UF WebClient*Adapter qty2 = new UF WebClient*Adapter(this.MoneyWithTax59);
 CallBack 控件值返回
args.ArgsResult.Add(qty2.ClientInstanceWithValue);
return args;

```

格式

```
args.ArgsResult.Add([CallBack 控件].[返回值标志]);  
return args;
```

14、 多行参照并自增行

本示例以 `DataTable` 作为多页面参数传递，也可直接将 `UIModel` 作为参数传递。

1) 定义多行参照UIForm

在确定按钮或 `Grid` 双击事件中添加如下代码：

```
DataTable table = new DataTable("RefMultiItem");  
table.Columns.Add(new DataColumn("ID", typeof(long)));  
table.Columns.Add(new DataColumn("Code", typeof(string))); // System.Type.GetType("System.Int64"));  
table.Columns.Add(new DataColumn("Name", typeof(string)));  
table.Columns.Add(new DataColumn("SPECS", typeof(string)));  
foreach (IUIRecord rd in this.Model.cRef.SelectRecords)  
{  
    DataRow row = table.NewRow();  
    row[0] = rd["ID"].ToString();  
    row[1] = rd["Code"].ToString();  
    row[2] = rd["Name"].ToString();  
    row[3] = rd["SPECS"].ToString();  
    table.Rows.Add(row);  
}  
this.CurrentState["INV_ItemMasterMulitSelectIDS"] = table;  
  
// 返回参照选择值，以触发主 Form 上的列编辑事件。  
string returnXml = "";  
IUIView view = this.Model.cRef;  
string strID = "";  
string strCode = "";  
string strName = "";  
Random Ra = new Random();  
string rodID = Ra.Next(200).ToString();  
GetReturnXML(view, ref strID, ref strCode, ref strName);  
returnXml += "<ReferenceReturn>;  
returnXml += "<ReturnField name='ID' value=" + rodID + " />"; //  
returnXml += "<ReturnField name='Code' value=" + strCode + " />"; // Code 集合  
returnXml += "<ReturnField name='Name' value=" + strName + " />"; // Name 集合  
returnXml += "<ReturnField name='Description' value=" + strID + " />"; // id 集合  
// G 类档案，多组织 ID 转换=====
```

```
        returnXml += string.Format("<ReturnField name=' {0}' value='Description' />",  
UFIDA.U9.UI.PDHelper.Commands.ReferenceMultiOrgInitComplete.MultiRefIDFlag);  
        //=====end=====  
        returnXml += "</ReferenceReturn>";  
        if (string.IsNullOrEmpty(returnXml))  
        {  
            string script1 = "<script language=\"javascript\">";  
            script1 += "window.close();";  
            script1 += "</script>";  
            UFSoft.UBF.UI.AtlasHelper.RegisterStartupScript((Control)TopLevelContainer,  
this.GetType(), "ReferenceReturn", script1, false);  
            return;  
        }  
        string xml = "<ReferenceReturns>";  
        xml += returnXml;  
        xml += "</ReferenceReturns>";  
        XmlDocument doc = new XmlDocument();  
        doc.LoadXml(xml);  
        xml = doc.DocumentElement.OuterXml;  
        xml = UFIDA.U9.UI.Commands.CommandHelper.EscapeHTML(xml); //编码  
        string script = "<script language=\"javascript\">";  
        script += "window.returnValue = " + xml + ";";  
        script += "window.close();";  
        script += "</script>";  
        AtlasHelper.RegisterStartupScript((Control)TopLevelContainer, this.GetType(),  
"ReferenceReturn", script, false); // 需注意ReferenceReturn作为Key值必须保证唯一  
  
private void GetReturnXML(IUIView view, ref string strID, ref string strCode, ref string strName)  
{  
    //从缓存中读取选中的记录  
    System.Collections.Generic.IList<IUIRecord> SelRec =  
UFSoft.UBF.UI.MD.Runtime.UIRuntimeHelper.Instance.GetSelectRecordFromCache(view);  
    if (SelRec != null && SelRec.Count > 0)  
    {  
        foreach (cRefRecord record in SelRec)  
        {  
            strID += record["ID"] + ",";  
            strCode += record["Code"] + ",";  
            strName += record["Name"] + ",";  
            // strSPECS += record["SPECS"] + ",";  
        }  
        strID = strID.Substring(0, strID.Length - 1);  
        strCode = strCode.Substring(0, strCode.Length - 1);  
        strName = strName.Substring(0, strName.Length - 1);  
    }  
}
```

```
//strSPECS = strSPECS.Substring(0, strSPECS.Length - 1);  
}  
}  
  


## 2) 主Form添加PostBack


根据参照返回的多行自动追加行。



```
public void AfterCreateChildControls()
{
 this.MiscRcvLsGridPostBack(this.DataGrid9);
}

public void MiscRcvLsGridPostBack(IUFDataGrid SourceGrid)
{
 AssociationControl control = new AssociationControl();
 control.SourceServerControl=SourceGrid;
 control.SourceControl.EventName="OnCellDataChanged";
 ((IUFClientAssoGrid)control.SourceControl).FireEventCols.Add("Item");
 CodeBlock block = new CodeBlock();
 UFWebClientGridAdapter adapter = new UFWebClientGridAdapter(SourceGrid);
 adapter.IsPostBack=true;
 adapter.PostBackTag=SourceGrid.ID + " " + control.SourceControl.EventName;
 block.TargetControls.addControl(adapter);
 control.addBlock(block);
 (SourceGrid as UFGrid).GridCustomerPostBackEvent += new
 GridCustomerPostBackDelegate(this.grid_MiscRcvLsGridPostBack);
}

private void grid_MiscRcvLsGridPostBack(object sender, GridCustomerPostBackEventArgs e)
{
 try
 {
 if (e.PostTag.ToString().EndsWith("OnCellDataChanged"))
 {
 this.OnDataCollect(this);
 base.IsDataBinding=true;
 base.IsConsuming=false;
 if (e.ColIndex == 5) // Item所在列号
 {
 this.OnLineItemRefPostBack();
 }
 }
 }
 catch (Exception exception)
```


```

```
        {
            //if (ExceptionHelper.IsUnknownException(exception, true))
            //{
            //    throw;
            //}
            //IUIModel model = this.Model;
            //this.Model.ErrorMessage.SetErrorMessage(ref model, exception);
        }
    }

private void OnLineItemRefPostBack()
{
    OrderDoc_OrderDocLinesRecord focusedRecord = this.Model.OrderDoc_OrderDocLines.FocusedRecord;
    if (focusedRecord != null)
    {
        if (this.CurrentState["INV_ItemMasterMulitSelectIDS"] != null)
        {
            DataTable dt = CurrentState["INV_ItemMasterMulitSelectIDS"] as DataTable;
            bool bAdd = false;
            foreach (DataRow row in dt.Rows)
            {
                if (bAdd)
                    focusedRecord = this.Model.OrderDoc_OrderDocLines.AddNewUIRecord();
                else
                    bAdd = true;

                focusedRecord.Item = Convert.ToInt64(row["ID"]);
                focusedRecord.Item_Code = row["Code"].ToString();
                focusedRecord.Item_Name = row["Name"].ToString();
                focusedRecord.Specifications = row["SPECS"].ToString();
                focusedRecord.SetParentRecord(this.Model.OrderDoc.FocusedRecord);
                focusedRecord.LineNum = GetNewLineNo(); // 函数见自动行号代码
            }
        }
    }
}
```

15、易用性（表头影响表体）

1) 添加表头自定义字段

在表头 **UIModel** 中添加仓库、库管员等字段，这些字段可通过对象浏览器中该 **BE** 的单体中拖入，然后新建一个 **CustomFields** 分组。

The screenshot shows the 'User Interface Model' (UIModel) for the 'OrderDoc0512' document type. The 'CustomFields' section contains fields for 'Wh' (仓库), 'Wh_Code' (编码), 'Wh_Name' (名称), 'WhMan' (库管员), and 'WhMan_Code' (编码). The 'Wh' field is highlighted with a red box.

2) 新增动作

The screenshot shows the 'Action Designer' dialog for the 'Misc' category. An action named 'OnWhManChange' is selected and its properties are displayed: Name: OnWhManChange, Display Name: OnWhManChange, and Code Template: Wh92.

3) 绑定事件

在 UIForm 中绑定参照及 TextChanged 事件。

The screenshot shows the 'Event Settings' dialog for the 'Wh92' control. The 'Event Settings' section shows the event 'OnWhChange [OnWhManChange]' is mapped to the 'OnWhChange' action. The 'Properties' panel on the right shows the 'TextChanged' event is bound to the 'Wh92' control.

4) 代码实现

通过单头上的存储地点和库管员修改事件，触发修改 model 中的字段默认值。

```
private void Wh92_TextChanged_Extend(object sender, EventArgs e)
{
    Wh92_TextChanged_DefaultImpl(sender, e);
    OrderDocRecord focusedRecord = this.Model.OrderDoc.FocusedRecord;
    if (focusedRecord != null)
    {
        long? wh = focusedRecord.Wh;
        this.headWhChangedPubDeal(focusedRecord, null, wh.HasValue ?
            wh.GetValueOrDefault() : -1L);
    }
}

private void headWhChangedPubDeal(OrderDocRecord headRecord, WarehouseRecord WhRecord,
long WhID)
{
    if (!this.isNullID(WhID))
    {
        if (WhRecord == null)
        {
            OrderDoc0512QueryUIModelModel model = new OrderDoc0512QueryUIModelModel();
            model.Warehouse.CurrentFilter.OPath = "ID=" + WhID.ToString();
            model.Warehouse.Clear();
            this.Action.CommonAction.Load(new IUIView[] { model.Warehouse });
            WhRecord = model.Warehouse.FocusedRecord;
        }
        if (this.isNullID(headRecord.WhMan))
        {
            headRecord.WhMan = WhRecord.Manager;
            headRecord.WhMan_Code = WhRecord.Manager_Code;
            headRecord.WhMan_Name = WhRecord.Manager_Name;
            this.Model.OrderDoc_OrderDocLines.FieldWhMan.DefaultValue = WhRecord.Manager;
            this.Model.OrderDoc_OrderDocLines.FieldWhMan_Code.DefaultValue = WhRecord.Manager_Code;
            this.Model.OrderDoc_OrderDocLines.FieldWhMan_Name.DefaultValue = WhRecord.Manager_Name;
        }
        headRecord.Wh = new long?(WhRecord.ID);
        headRecord.Wh_Code = WhRecord.Code;
        headRecord.Wh_Name = WhRecord.Name;
        this.Model.OrderDoc_OrderDocLines.FieldWh.DefaultValue = WhRecord.ID;
        this.Model.OrderDoc_OrderDocLines.FieldWh_Code.DefaultValue = WhRecord.Code;
        this.Model.OrderDoc_OrderDocLines.FieldWh_Name.DefaultValue = WhRecord.Name;
    }
}
```

```
    }

    private void WhMan148_TextChanged_Extend(object sender, EventArgs e)
    {
        WhMan148_TextChanged_DefaultImpl(sender, e);
        OrderDocRecord focusedRecord = this.Model.OrderDoc.FocusedRecord;
        this.Model.OrderDoc_OrderDocLines.FieldWhMan.DefaultValue=focusedRecord.WhMan;

        this.Model.OrderDoc_OrderDocLines.FieldWhMan_Code.DefaultValue=focusedRecord.WhMan_Code;
        this.Model.OrderDoc_OrderDocLines.FieldWhMan_Name.DefaultValue=focusedRecord.WhMan_Name;
        if (((focusedRecord != null) && !this.IsNullID(focusedRecord.WhMan)) &&
        this.IsNullID(focusedRecord.Wh))
        {
            OrderDoc0512QueryUIModelModel model = new OrderDoc0512QueryUIModelModel();
            model.Warehouse.CurrentFilter.QPath="Manager =" +
        focusedRecord.WhMan.ToString();
            model.Warehouse.Clear();
            this.Action.CommonAction.Load(new IUIView[] { model.Warehouse });
            if (model.Warehouse.Records.Count == 1)
            {
                WarehouseRecord whRecord = model.Warehouse.FocusedRecord;
                this.headWhChangedPubDeal(focusedRecord, whRecord, whRecord.ID);
            }
        }
    }
}
```

五、 其他应用

1、 直接运行SQL和存储过程

1) 引用UFSoft.UBF.Util.DataAccess.dll

2) DataAccessor类

方法 RunSQL: 直接运行 SQL

方法 RunSP: 直接运行存储过程

3) 示例（RunSQL）

```
StringBuilderExecuteSql= new StringBuilder(256);
ExecuteSql.Append(" DECLARE @StartSNbigint; ");
ExecuteSql.Append(" DECLARE @EmpCountint; ");
ExecuteSql.Append(" Set @EmpCount= " + this.EmpPayrolls.Count* 10 + ";");
ExecuteSql.Append(" EXEC [dbo].[AllocSerials] @EmpCountOUTPUT, @StartSNOUTPUT ;");
ExecuteSql.Append(InsertSql.ToString());//其它 Insert 语句
ExecuteSql.Append(UpdatePayrollDocSQL.ToString());//其他 Update 语句
ExecuteSql.Append(UpdatePayrollSupplySQL.ToString());
//进行提交 SQL
DataAccessor.RunSQL(UFSoft.UBF.Sys.Database.DatabaseManager.GetCurrentConnection(),
ExecuteSql.ToString(), new DataParamList(), true);
```

4) 示例（RunSP）

```
DataParamListlst;
lst= new DataParamList();
lst.Add(DataParamFactory.CreateInput("PayrollDoc",
                                         this.ID.ToString()),
System.Data.DbType.StringFixedLength);
lst.Add(DataParamFactory.CreateInput("PayrollCalculate",
                                         this.PayrollCaculate.ID.ToString()),
System.Data.DbType.StringFixedLength);
DataAccessor.RunSP(DatabaseManager.GetCurrentConnection(), "PA_ReWritepayrollresult", lst);
```

2、弱类型实体

1) 弱类型实体新建

```
///<summary>
///根据名称建实体
///</summary>
[Test]
public void TestCrtEntity()
{
    using (ISession=Session.Open())
    {
        string fullname="UFIDA.U9.CBO.SCM.Customer.Customer";
        //新建
        Entity entity=Entity.Create(fullname, null);
        entity.SetValue("Name", "1234567");
```

```
//提交
s.Commit();
}
}
```

2) 弱类型实体删除

```
///<summary>
///根据名称删除实体
///</summary>
[Test]
public void TestDelEntity()
{
    using (ISession s = Session.Open())
    {
        string fullname = "UFIDA.U9.CBO.SCM.Customer.Customer";
        //查询
        EntityQuery query = new EntityQuery(fullname);
        Entity entity = query.FindByID((long)111);
        //删除
        s.Remove(entity);
        //提交
        s.Commit();
    }
}
```

3) 弱类型实体查询修改

```
///<summary>
///查询
///</summary>
[Test]
public void FindAndMod()
{
    using (ISession s = Session.Open())
    {
        string fullname = "UFIDA.U9.CBO.SCM.Customer.Customer";
        //查询
        EntityQuery query = new EntityQuery(fullname);
        Entity entity = query.FindByID((long)111);
        //修改
    }
}
```

```
entity.SetValue("Name", "forTest");
s.Modify(entity);
//提交
s.Commit();
}
}
```

3、 异常处理

在一些特殊的业务情况下，可能不够详细，如果需要扩展平台的异常信息，可以在实体的**Extend**的文件中，重载基类方法**DealException (Exception e)**，然后捕获自己期待的异常，封装为规范的异常显示

示例：

```
public override void DealException (Exception e)
{
    if (e is RefObjectInValidException)
    {
        RefObjectInValidException re = (RefObjectInValidException)e;
        throw new ApplicationExceptionBase ("实体[" + re.EntityFullName + "] 的属性[" +
            re AttrName + "] 引用对象不合法，值为[" + this.Values [re AttrName] + "]");
    }
    base.DealException (e);
}
```

1) 业务异常绑定到实体

需要给异常赋值EntityID和EntityFullName，如：

```
XXXException.EntityID= EntityInstanceID;
XXXException.EntityFullName= EntityFullName;
```

2) 业务异常绑定到属性

需要给异常赋EntityID, EntityFullName, AttributeName, 如：

```
XXXException.EntityID= EntityInstanceID;
XXXException.EntityFullName= EntityFullName
XXXException.AttributeName=AttributeName
```

异常绑定到实体：XX.EntityID=EntityInstanceID;

```
XX.EntityFullName = EntityFullName
```

异常绑定到属性：

```
XX.AttributeName = "AA";
```

3) UI插件中使用示例

```
public override void BeforeEventProcess(IPart Part, string eventName, object sender, EventArgs args, out bool executeDefault)
{
    base.BeforeEventProcess(Part, eventName, sender, args, out executeDefault);
    IUIView view = Part.ViewContainers[1].UIView;
    ShipPlanByBatchPart.Model.ClearErrorMessage(); // 清除原有错误提示
    if (view.SelectRecords.Count != 0)
    {
        // 检查可用量，当可用量不足时，抛出错误
        IUIModel model = ShipPlanByBatchPart.Model;
        Exception ex = new Exception();
        foreach (IUIRecord rd in view.SelectRecords)
        {
            if (GetSL(rd["SPBB_CurrentPlanQty1"]) > GetSL(rd["ky1"])) {
                ShipPlanByBatchPart.Model.ErrorMessage.SetErrorMessage(ref model, ex);
                throw new Exception("料品可用量不足，请重新选择或修改本次出货计划数量！");
            }
        }
    }
}
```

4、 实体锁

1) 引用UFSoft.UBF.Business.dll

2) 主要接口

- UFSoft.UBF.Business.Lock.EntityLockManager 提供实体的加锁、解锁、以及查询接口 API。
- LockForWrite(entity.Key) // 加写锁
- LockForRead(entity.Key) // 加读锁
- Unlock(eLock) // 解锁
- QueryLockOwners(entity.Key) // 查询特定实体的加锁信息

-
- QueryLocks()//查询当前用户的加锁信息

3) 使用样例

```
using(ISession session = Session.Open())
{
    //获得实体entity
    .....
    //对实体entity加写锁
    EntityLock eLock=EntityLockManager.LockForWrite(entity.Key);
    if(eLock!=null)
    {
        //如果加锁成功
        .....
        session.Commit();
        EntityLockManager.Unlock(eLock); // 对实体entity解锁
    }
}
```

加锁操作与解锁操作一定要成对的进行。或者使用 **using** 语句。

5、 实体缓存

1) 实体缓存简介

为提高性能，避免频繁的访问数据库，后台引擎提供了实体缓存的机制

2) 典型的特殊场景

开发在服务中嵌入部分存储过程以致修改了实体的数据，而这个数据在该服务的后续场景还要访问到被存储过程更新的新值，这是很特殊的情况，需要将 2 层缓存的数据全部清除，强制下次访问直接从数据库加载新值。

3) 清除缓存中的实体

可以通过下面的方式访问**2**层缓存，引用UFSoft.UBF.PL.dll：

线程级数据缓存是：UFSof t.UBF.PL.Tool.ConfigParm.EntityCache；

服务级引用缓存是：**UFSof t.UBF.PL.Engine.StateManager.EntityStateCache**；

两者返回的对象都是**ICache**类型，**ICache**有下面**2**个对外方法可以清除缓存：

```
///<summary>
///从Cache中删除指定的对象
///</summary>
///<paramname="key"></param>
void Remove(object key);
///<summary>
///清除所有的Cache内容
///</summary>
void Flush();
```

注意

- 清缓存时这2级缓存都要清除。
- void Remove (object key) , 参数 key 是实体的 EntityKey

4) UI中清除缓存

```
private void ClearCache()
{
    UFSofT.UBF.UI.MD.Runtime.UIRuntimeHelper.Instance.ClearCache(CurrentModel.ItemMaster);
    UFSofT.UBF.UI.MD.Runtime.UIRuntimeHelper.Instance.ClearCache(CurrentModel.ItemMaster_MrpInfo);
}
```

6、 数据记录状态

- BE 中，通过 SysState 值进行判断。
if (so.SysState == ObjectState.Deleted) {...}
- UIForm 中，数据状态有 Add/Delete/Change/Insert/Unchange。
// DataRowState DataRecordState { get; set; }
this.Model.P2PSendDoc_TransFees.Records[0].DataRecordState;
this.Model.CarInfo.FocusedRecord.DataRecordState
- 在 UIModelAction 中，可通过 CurrentPart.PageStatus 获得页编辑状态
如 UFSofT.UBF.UI.IView.PartStateType.Insert 为新增状态。

7、 CallBack函数调用

1) CallBackVSPostBack

页面无刷新，PostBack 是回发到服务端进行处理，页面要刷新，在 AfterCreateChildControls 中注册 CallBack 函数。

2) 示例

在AfterCreateChildControls中注册CallBack函数，函数定义如下

```
private void SchemaCallBack()
{
    AssociationControlObjectTypeASC= new AssociationControl();//交互关联控件实例
    objectTypeASC.SourceServerControl= this.QCSchemaHead172;//触发源控件
    objectTypeASC.SourceControl.EventName= “onchange”,触发事件
    ClientCallBackFrmCBF = new ClientCallBackFrm();
    CBF.DoCustomerAction+= new
    ClientCallBackFrm.ActionCustomer( CBF_DoCustomerAction_SetRef);//定义服务器端的处理方法
    CBF.ParameterControls.Add( this.QCSchemaHead172 );//添加传送到服务器端的控件值
    CBF.Add( objectTypeASC );
}

object CBF_DoCustomerAction_SetRef(CustomerEventArgsargs)
{
    UF WebClientTextBoxAdapterSrcRef= new UF WebClientTextBoxAdapter( this.Version0 );
    UFIDA.U9.CBO.QCBP.Proxy.GetSchemaVersionProxy tmpproxy= new
    UFIDA.U9.CBO.QCBP.Proxy.GetSchemaVersionProxy();
    tmpproxy.QCSchemaID= long.Parse( args.ArgsHash[this.QCSchemaHead172.ClientID].ToString() );//获
    取客户端控件值
    SrcRef.Value= tmpproxy.Do();//赋值
    args.ArgsResult.Add( SrcRef.ClientInstanceWithValue );//生成前台JS脚本更新控件值
    return args;
}
```

8、 BP数据校验（共享冲突）

可在 BP 设计时加入以下代码进行数据版本校验，检查是否存在其他人修改单据问题。

```
AllowanceMaterial allowanceMaterialDoc = bpObj.DocKey.GetEntity();
if (allowanceMaterialDoc == null)
{
    throw new DataDeletedException(AllowanceMaterial.EntityRes.GetResource(
        AllowanceMaterial.EntityRes.BE_FullName));
}

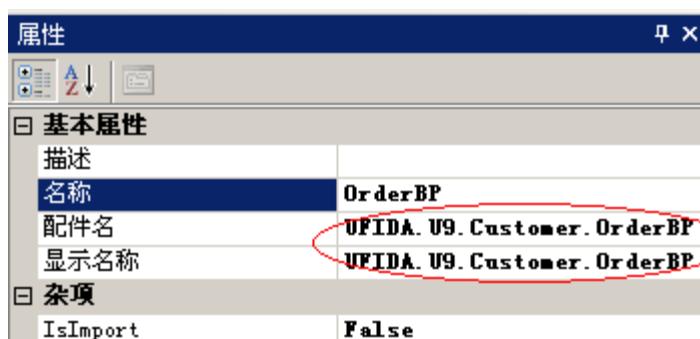
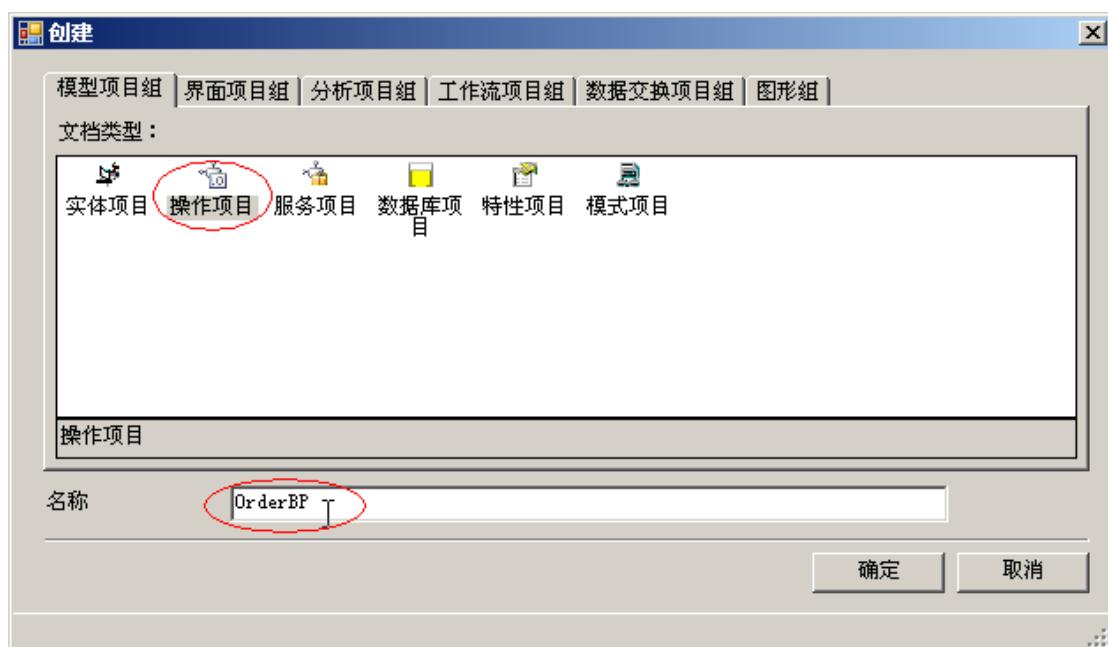
if ((bpObj.UISysversion.Trim() != string.Empty) &&
    (allowanceMaterialDoc.SysVersion.ToString() != bpObj.UISysversion.Trim()))
{
    throw new DataModifiedException(AllowanceMaterial.EntityRes.GetResource(
        AllowanceMaterial.EntityRes.BE_FullName));
}
```

第七章 单据生单

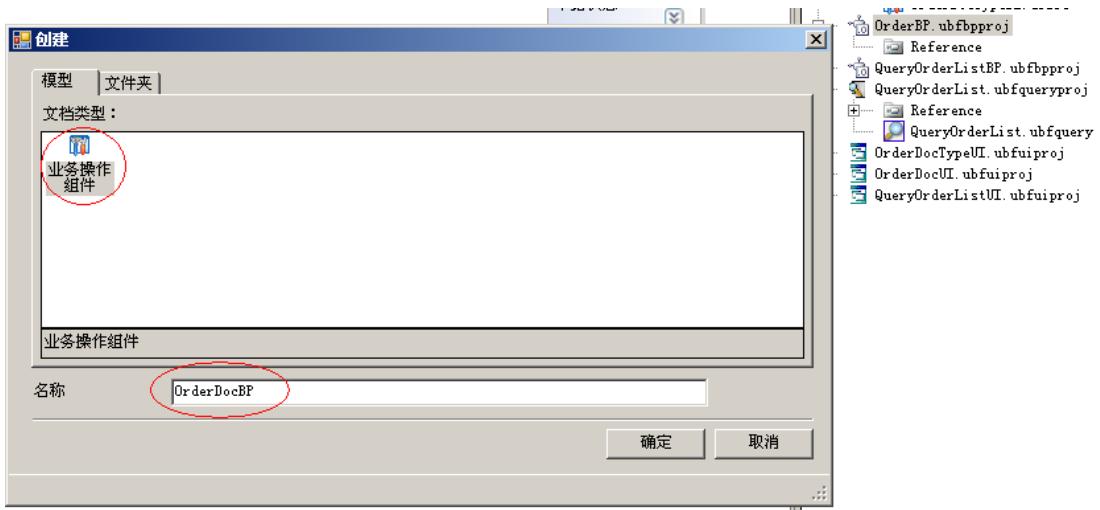
一、 推式生单

1、 建立BP组件

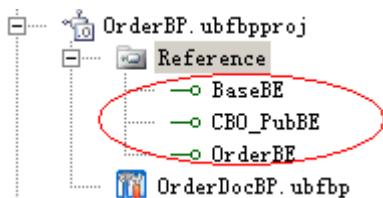
1) 建立BP项目



2) 建立业务操作组件



● 添加组件引用



● 定义查询条件 DTO

ApproveBP OrderDocBP

数据已经被其它人修改异常 订货单生成销售订单

属性

基本属性

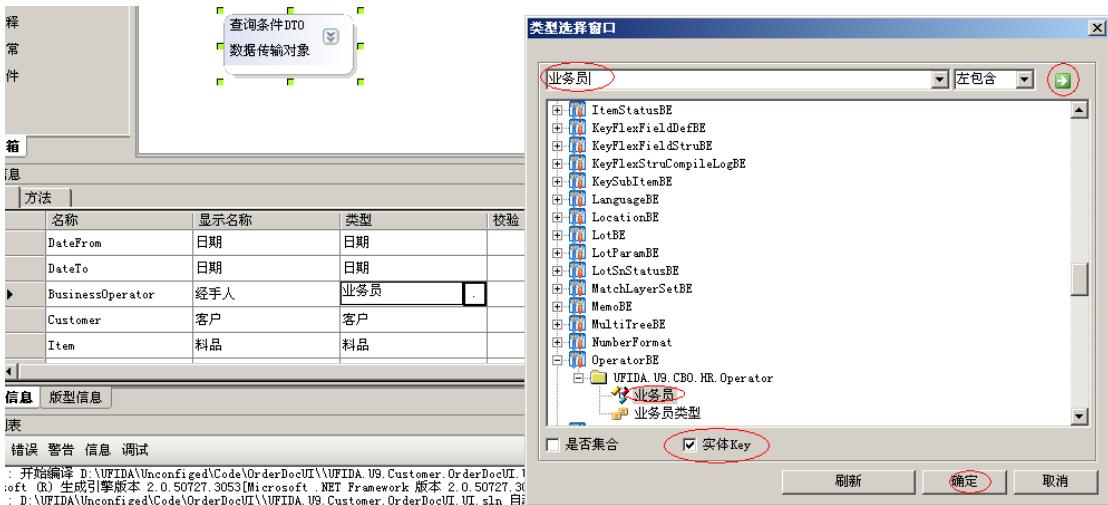
名称: QueryConditionDTO
显示名称: 查询条件 DTO

杂项

是否查询DTO: True

详细信息

名称	显示名称	类型	校验	缺省值	分组	业务主题	主键
DocType	单据类型	订货单单据类型			Misc	<input type="checkbox"/>	<input type="checkbox"/>
DocNoFrom	单号	字符串			Misc	<input type="checkbox"/>	<input type="checkbox"/>
DocNoTo	单号	字符串			Misc	<input type="checkbox"/>	<input type="checkbox"/>
DateFrom	日期	日期			Misc	<input type="checkbox"/>	<input type="checkbox"/>
DateTo	日期	日期			Misc	<input type="checkbox"/>	<input type="checkbox"/>
BusinessOperator	经手人	业务员			Misc	<input type="checkbox"/>	<input type="checkbox"/>
Customer	客户	客户			Misc	<input type="checkbox"/>	<input type="checkbox"/>
Item	料品	料品			Misc	<input type="checkbox"/>	<input type="checkbox"/>
Currency	币种	币种			Misc	<input type="checkbox"/>	<input type="checkbox"/>
HandleDateFrom	交期	日期			Misc	<input type="checkbox"/>	<input type="checkbox"/>
HandleDateTo	交期	日期			Misc	<input type="checkbox"/>	<input type="checkbox"/>
*						<input type="checkbox"/>	<input type="checkbox"/>

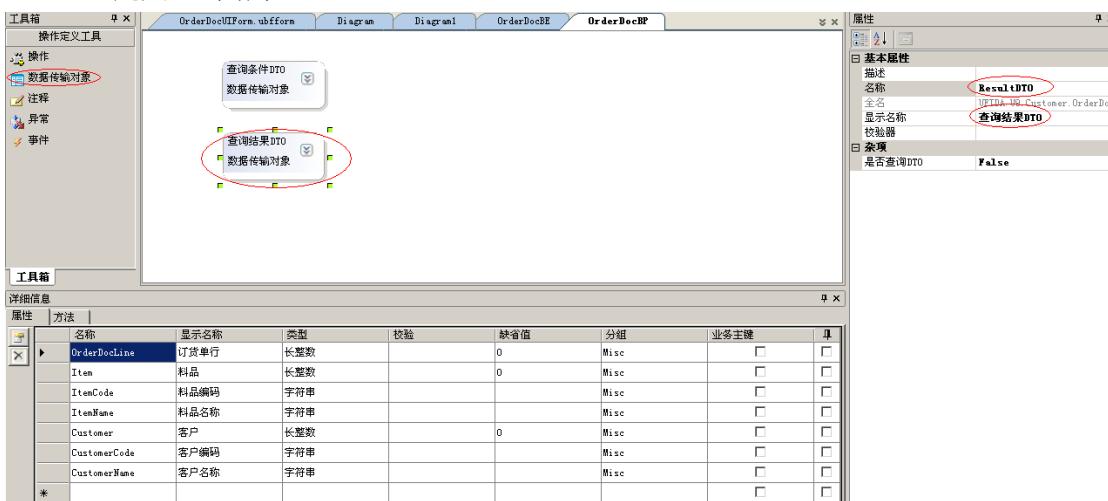


注：参照类型需打上实体 Key 标志，否则 C# 添加代码编译时会报以下错误。

无法将类型“long”隐式转换为“UFTIDA.U9.CBO.HR.Operator.OperatorsData”

BP 内涉及实体的均是实体 KEY，不允许用 ID；若需传入多参数，则需设计 DTO。

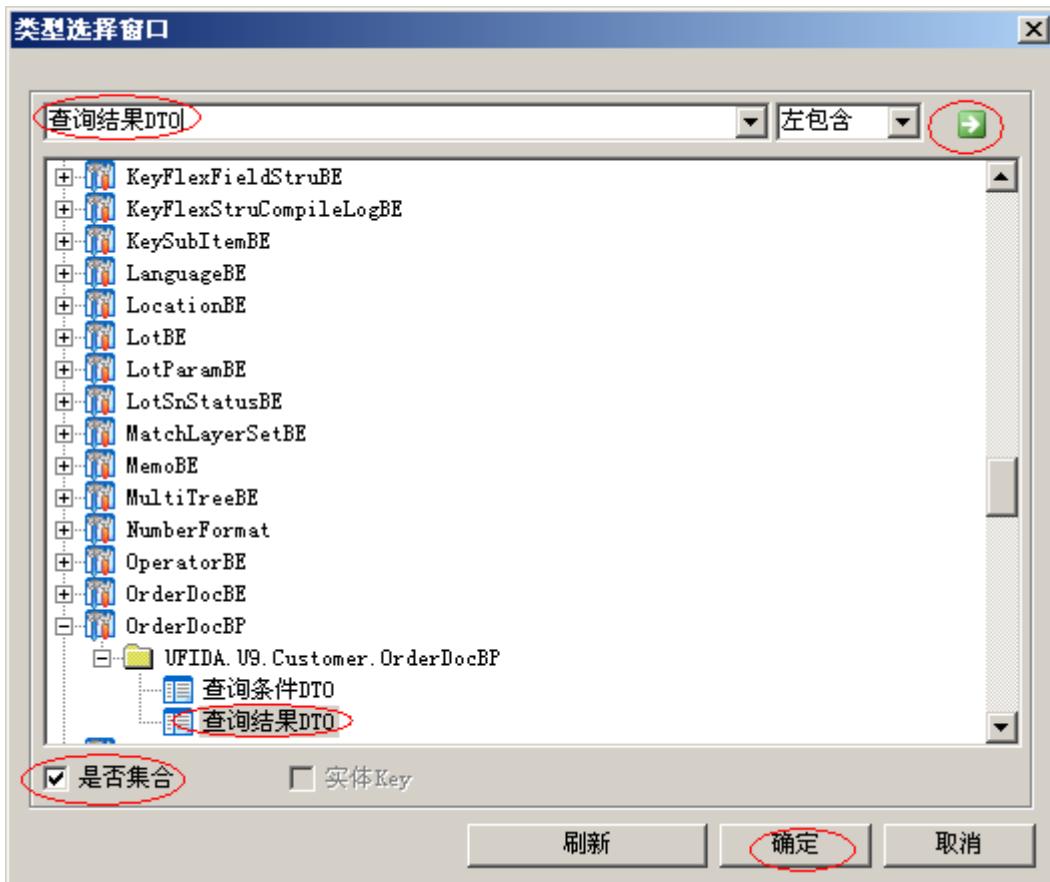
● 定义查询结果 DTO



● 定义查询订货单操作



注：必须先指定返回类型，且指定为集合。



● 定义订货单生成操作



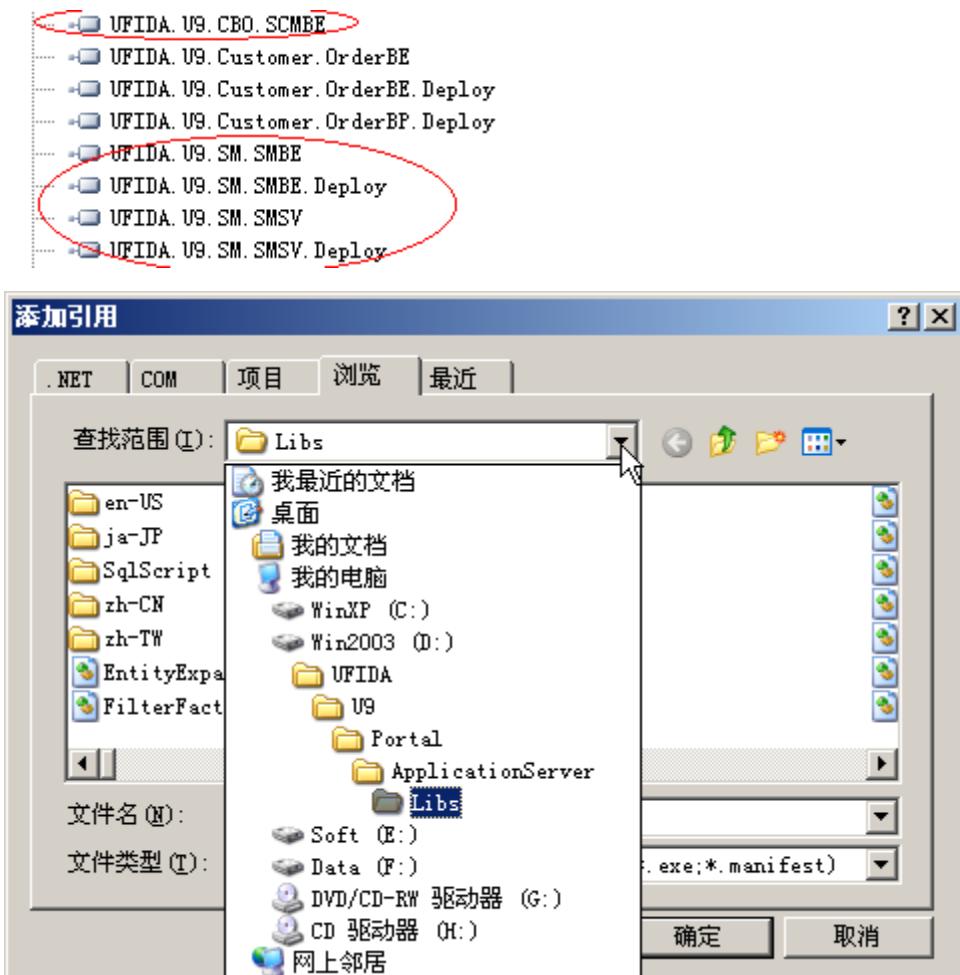


3) 构造BP组件

4) 编写BP代码

① 引用:

```
using UFIDA.U9.Customer.OrderDoc;
using System.Collections;
using UFIDA.U9.SM.SO;
using UFSOFT.UBF.Business; // session
using UFIDA.U9.CBO.SCM.Enums;
using UFIDA.U9.Base.Organization;
using UFIDA.U9.CBO.SCM.Item;
using UFIDA.U9.CBO.FI.RecPayTerm;
using UFIDA.U9.CBO.FI.ConfirmTerm;
using UFIDA.U9.CBO.SCM.ShipmentBy;
```

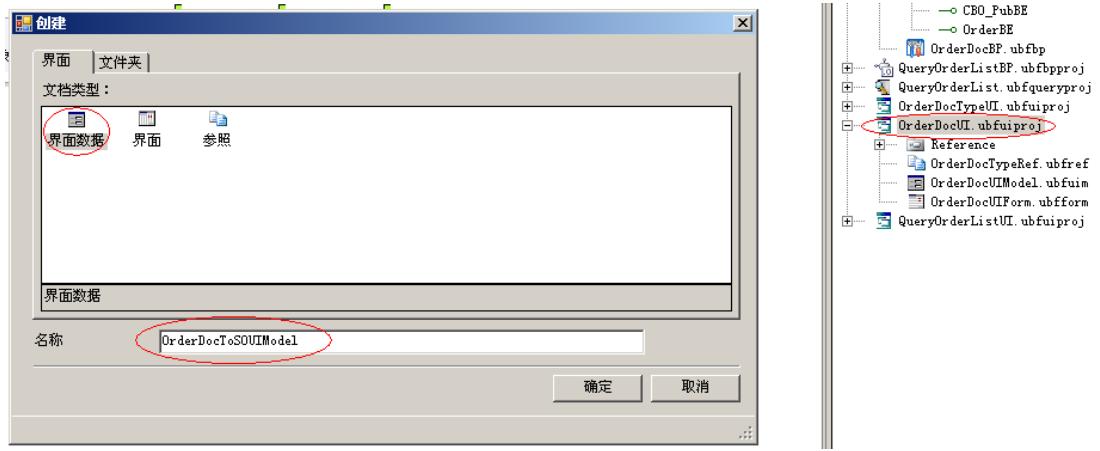


5) 装配BP组件

2、 定义生单UI

1) 定义生单UIModel

① 创建 UIModel

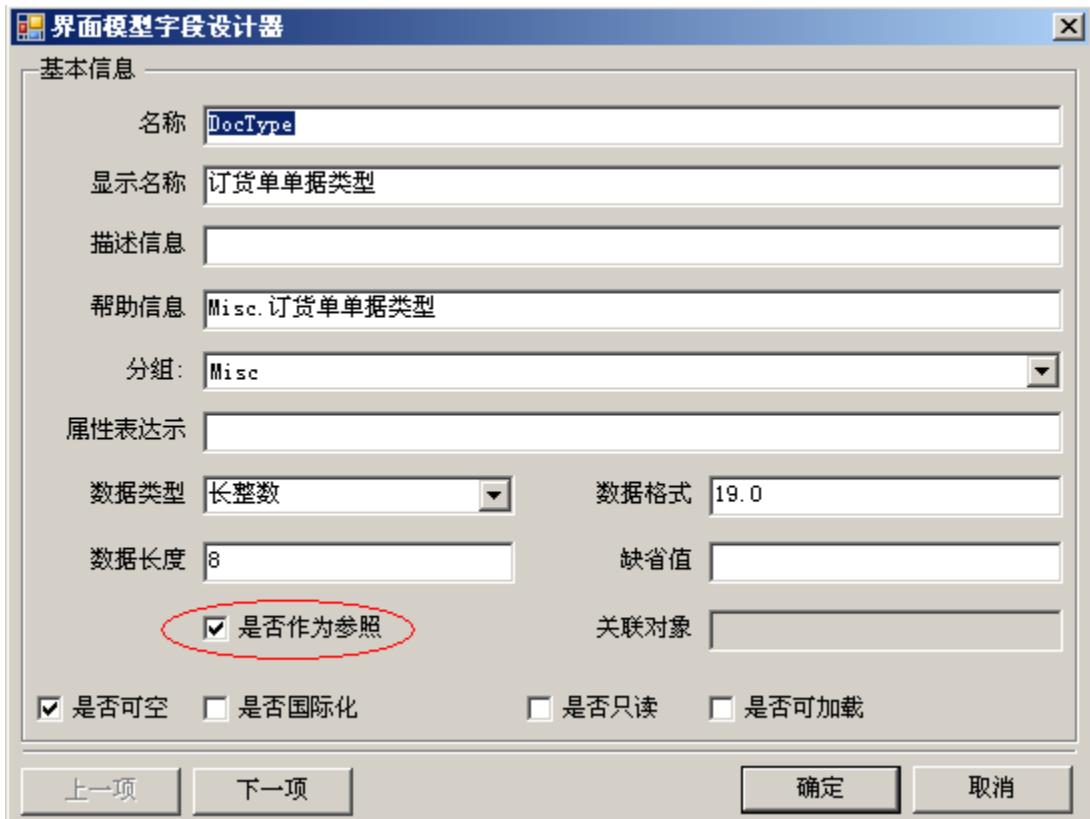


② 定义条件视图

The screenshot shows the 'OrderDocToSOUIModel.ufuiim' interface model editor. On the left, the 'User Interface Model Tree' pane shows a node 'ConditionView' under 'OrderDocToSOUIModel'. A context menu is open over this node, with the 'New View' option highlighted by a red circle. A sub-dialog titled 'Interface Model View Designer' is open, showing fields for 'Name' (set to 'ConditionView') and 'Display Name' (also set to 'ConditionView'). The 'OK' button is circled in red. Below the tree view, a detailed table lists fields for the 'ConditionView' model, including columns for 'Field Name', 'Data Type', 'Nullable', 'Format', 'Default Value', 'Relationship Object', 'Entity Name', and 'Properties'.

字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
ID	长整数	可空	19.0				
Misc	单据编码	可空	50				
DocType	订货单单据类型	可空	19.0				
DocType_Code	单据编码	可空	50				
DocType_Name	单据名称	可空	50				
DocNoFrom	单号从	可空	50				
DocNoTo	到	可空	50				
DateFrom	单据日期从	日期	8				
DateTo	到	日期	8				
BusinessOperator	经手人	长整数	19.0		业务员		
BusinessOperator_Code	经手人编码	字符串	50				
BusinessOperator_Name	经手人名称	字符串	50				
Customer	客户	长整数	19.0		客户		
Customer_Code	客户编码	字符串	50				
Customer_Name	客户名称	字符串	50				
Item	料品	长整数	19.0		料品		
Item_Code	料品编码	字符串	50				
Item_Name	料品名称	字符串	50				
Currency	币种	长整数	19.0		币种		
Currency_Code	币种编码	字符串	50				
Currency_Name	币种名称	字符串	50				
HandleDateFrom	交期从	日期	8				
HandleDateTo	到	日期	8				
DefaultFilter	缺省过滤条件						

注：若该字段是参照 I D，则可设置是否作为参照，且必须同步设置参照 Code 和 Name 字段，并且需找到相应实体，拖放到参照行自动形成关联对象。



字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
ID	长整数	✓	19.0				
DocType	订货单单据类型	长整数	✓	19.0			
DocType_Code	单据编码	字符串	✓	50			
DocType_Name	单据名称	字符串	✓	50			
DocNoFrom	单号从	字符串	✓	50			
DocNoTo	到	字符串	✓	50			
DateFrom	单据日期从	日期	✓	8	2009-09-17		
DateTo	到	日期	✓	8	2009-09-17	业务员	
BusinessOperate	经手人	长整数	✓	19.0			
BusinessOperateCode	经手人编码	字符串	✓	50			
BusinessOperateName	经手人名称	字符串	✓	50			
Customer	客户	长整数	✓	19.0		客户	
Customer_Code	客户编码	字符串	✓	50			
Customer_Name	客户名称	字符串	✓	50			
Item	料品	长整数	✓	19.0			料品
Item_Code	料品编码	字符串	✓	50			
Item_Name	料品名称	字符串	✓	50			
Currency	币种	长整数	✓	19.0			币种
Currency_Code	币种编码	字符串	✓	50			
Currency_Name	币种名称	字符串	✓	50			
HandleDateFrom	交易从	日期	✓	8	2009-09-17		
HandleDateTo	到	日期	✓	8	2009-09-17		

③ 定义查询结果视图

字段名	数据类型	可空	数据格式	缺省值	关联对象	实体名	属性名
ID	长整数	✓					
Customer	客户	长整数	✓	19.0		客户	
Customer_Code	客户编码	字符串	✓	50			
Customer_Name	客户名称	字符串	✓	50			
Item	料品	长整数	✓	19.0		料品	
Item_Code	料品编码	字符串	✓	50			
Item_Name	料品名称	字符串	✓	50			

④ 定义按钮事件:

系统事件需指定代码模板, 如: .\UITpls\WebPart\RefCodeBehindCommon.tpl

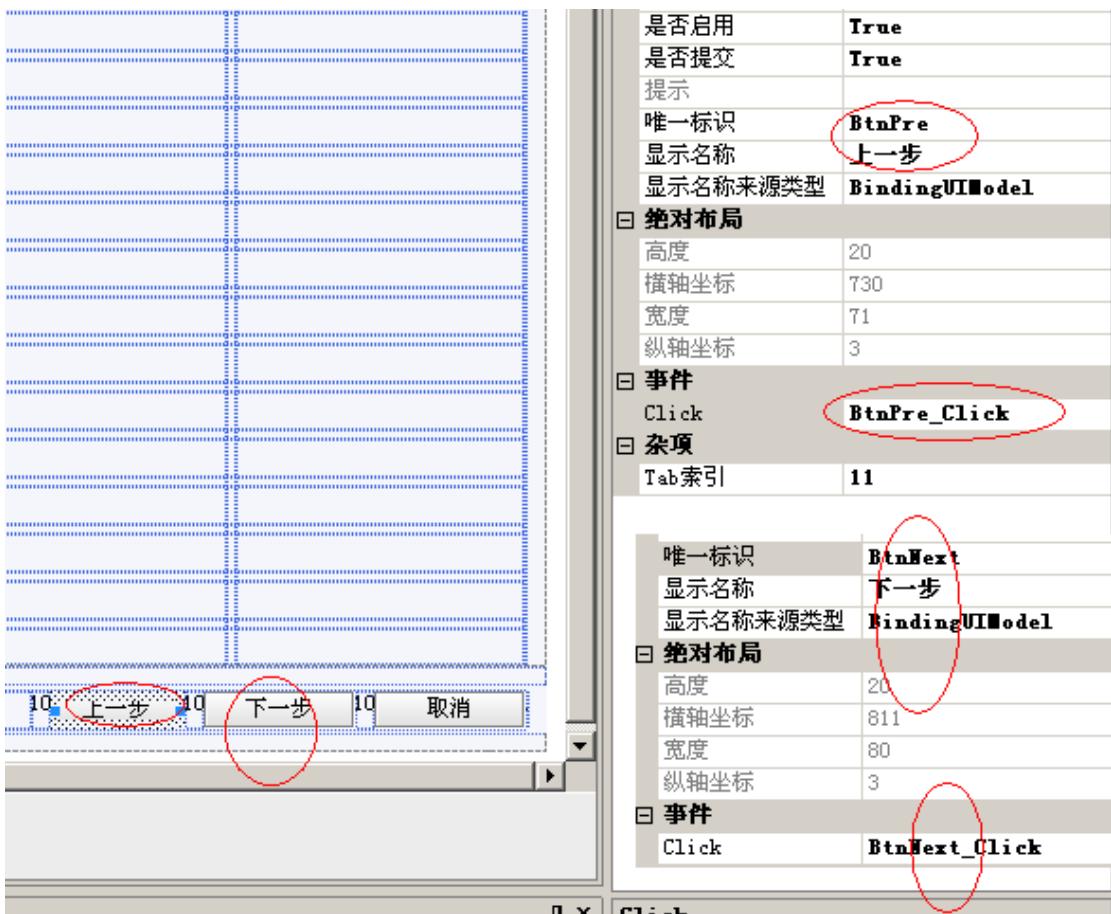


2) 定义UIForm

- 定义 View

属性	值
UIViewID	.UI_单卡片WebPart
URI	.UI_单卡片
表单模板	UI_单卡片
权限控制	True
容器布局	{0,0,0,5,10,1,3}
是否个性化	False
是否启用	True
是否主Form	False
提示	OrderDocToSOUIModel
唯一标识	OrderDocToSOUIModel
显示名称	订货单转销售订单
显示名称来源类型	Custom

- 定义事件



3) 实现生单

- ① 添加文件引用, D:\UFIDA\U9\Portal\ApplicationLib\或 D:\UFIDA\U9\Portal\UBFLib 目录。

```

... -> UFIDA.U9.Base.BaseBE.Deploy
... -> UFIDA.U9.CBO.PubBE.Deploy
... -> UFIDA.U9.Customer.OrderBE.Deploy
... -> UFIDA.U9.Customer.OrderBP.Agent
... -> UFIDA.U9.Customer.OrderBP.Deploy

```

- ② 添加代码引用:

```

using System.Collections.Generic;
using UFIDA.U9.Customer.OrderDocBP;

```

- ③ 编写生单代码:

```

private void BtnNext_Click_Extend(object sender, EventArgs e)
{
    //调用模版提供的默认实现--默认实现可能回调用相应的Action

    BtnNext_Click_DefaultImpl(sender, e);

    if (this.WizardControl1.SelectedIndex == 0)
    {
        //查询订单单行
        if (this.Model.ConditionView.FocusedRecord != null)
        {
            UFGDA.U9.Customer.OrderDocBP.QueryConditionDUDatas ConditionDUT = new UFGDA.U9.Customer.OrderDocBP.QueryConditionDUDatas();
            ConditionDUT.BusinessOperator = this.Model.ConditionView.FocusedRecord.BusinessOperator ?? 0;
            ConditionDUT.Currency = this.Model.ConditionView.FocusedRecord.Currency ?? 0;
            ConditionDUT.Customer = this.Model.ConditionView.FocusedRecord.Customer ?? 0;
            ConditionDUT.DateFrom = this.Model.ConditionView.FocusedRecord.DateFrom ?? DateTime.MinValue;
            ConditionDUT.DateTo = this.Model.ConditionView.FocusedRecord.DateTo ?? DateTime.MinValue;
            ConditionDUT.DocFrom = this.Model.ConditionView.FocusedRecord.DocFrom;
            ConditionDUT.DocType = this.Model.ConditionView.FocusedRecord.DocType ?? 0;
            ConditionDUT.HandleDateFrom = this.Model.ConditionView.FocusedRecord.HandleDateFrom ?? DateTime.MinValue;
            ConditionDUT.HandleDateTo = this.Model.ConditionView.FocusedRecord.HandleDateTo ?? DateTime.MinValue;
            ConditionDUT.QueryOrderDocBP proxy = new UFGDA.U9.Customer.OrderDocBP.Proxy.QueryOrderDocBPProxy();
            proxy.Condition = ConditionDUT;
            List<GenulUTDDatas> list = proxy.Do();
            if (list != null)
            {
                this.Model.ResultView.Clear();
                foreach (GenulUTDDatas dtdatas in list)
                {
                    ResultViewRecord resultRecord = this.Model.ResultView.NewUIRecord();
                    resultRecord.ID = dtdatas.OrderNoLine;
                    resultRecord.CustomerCode = dtdatas.CustomerCode;
                    resultRecord.CustomerName = dtdatas.CustomerName;
                    resultRecord.Item = dtdatas.Item;
                    resultRecord.Item_Code = dtdatas.ItemCode;
                    resultRecord.Item_Name = dtdatas.ItemName;
                    this.Model.ResultView.Records.Add(resultRecord);
                }
            }
        }
    }
}

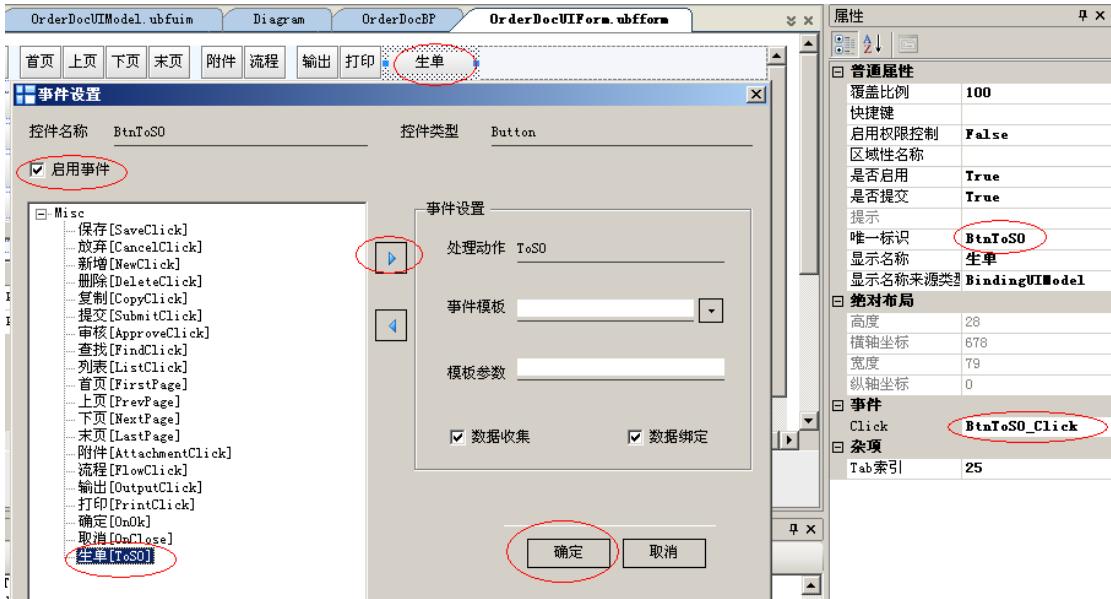
```

④ UI 中新增按扭调用生单。

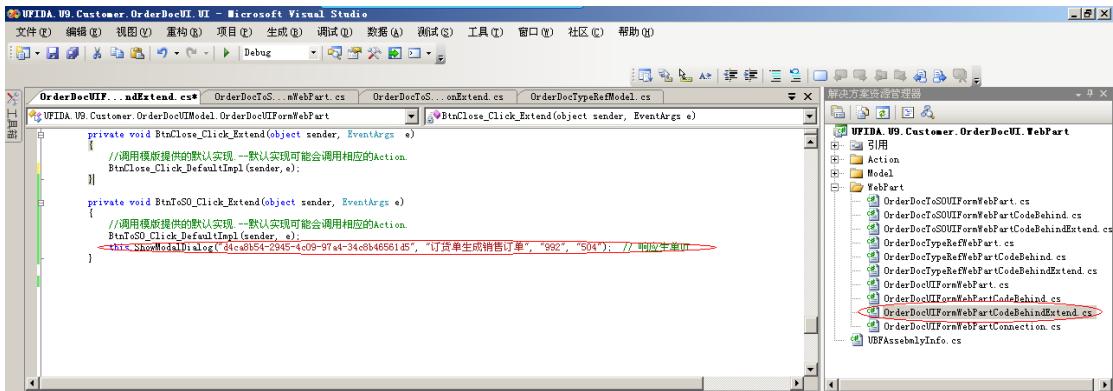
- UIModel 中新增事件



- 新增生单按钮，并指定生单事件



● 生单事件中挂接窗体引用



```

private void BtnClose_Click_Extend(object sender, EventArgs e)
{
    //调用模版提供的默认实现. --默认实现可能会调用相应的Action.
    BtnClose_Click_DefaultImpl(sender, e);
}

private void BtnToSO_Click_Extend(object sender, EventArgs e)
{
    //调用模版提供的默认实现. --默认实现可能会调用相应的Action.
    BtnToSO_Click_DefaultImpl(sender, e);
    this.ShowModalDialog("d4ca8b54-2945-4c09-97a4-34c8b46561d5", "订货单生成销售订单", "992", "504"); // 响应生单UI
}

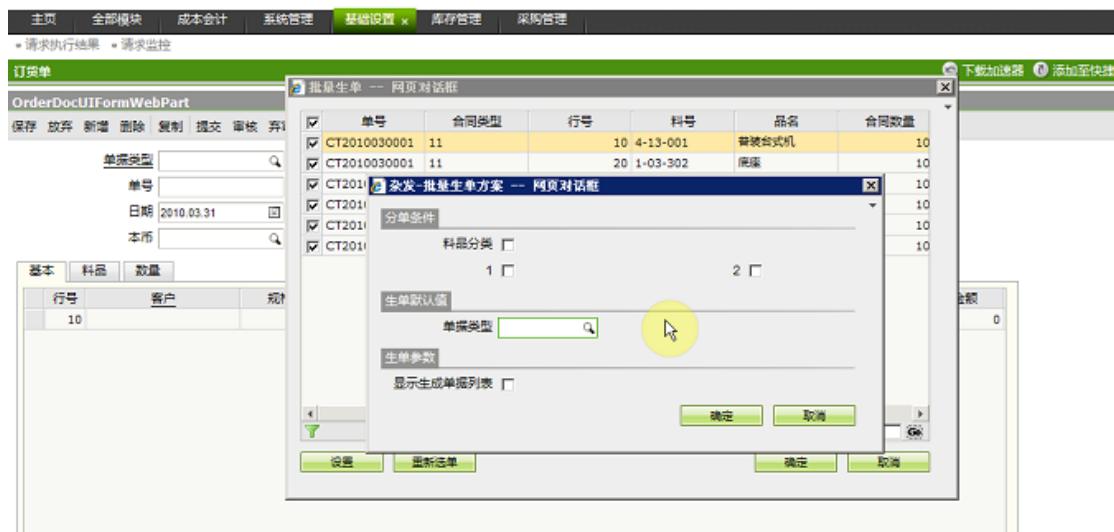
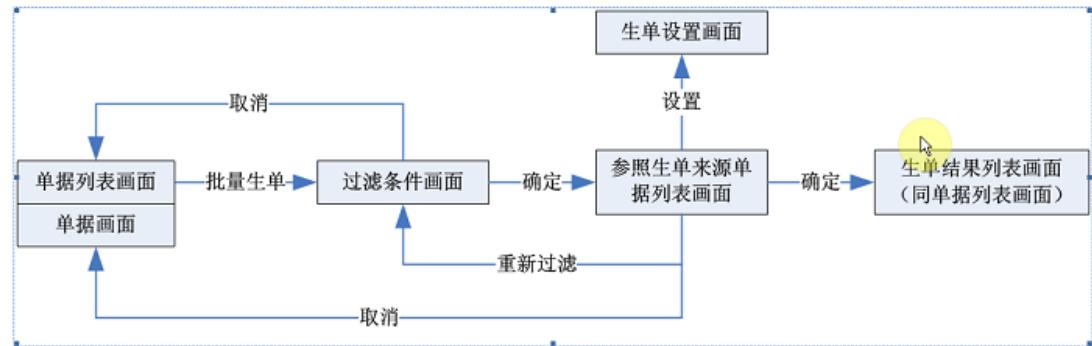
```

二、 拉式生单

1、 总体框架

根据合同拉式生成订货单。

生单方案的处理、生单 UI 处理、系统 API 的调用，生单时弹性域的处理

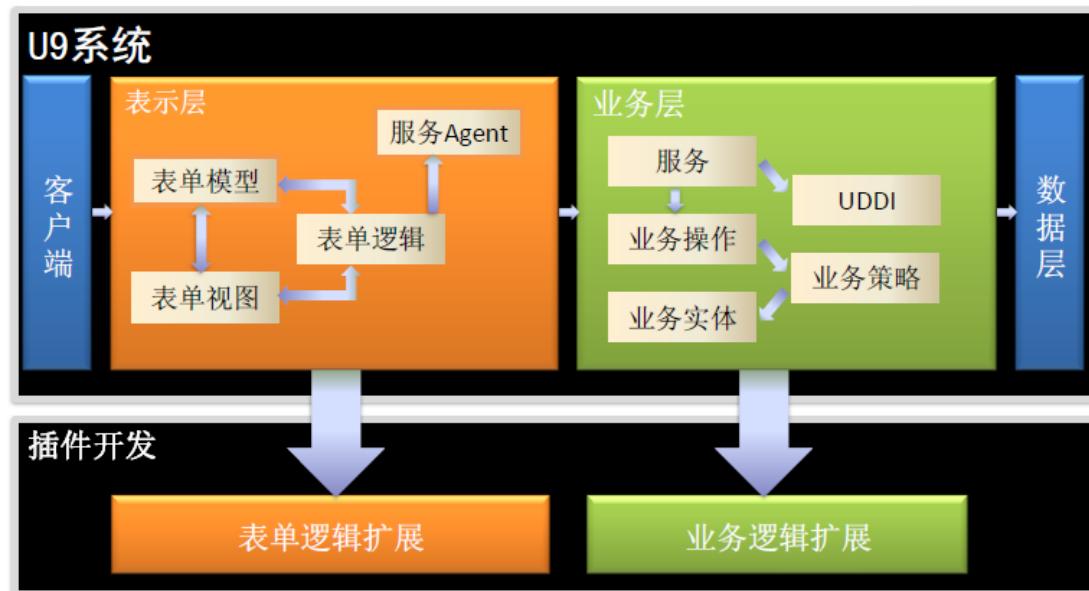


2、 建立SV组件

3、 定义生单UI

第八章 单据插件开发

一、 插件开发特点



1) 业务逻辑扩展

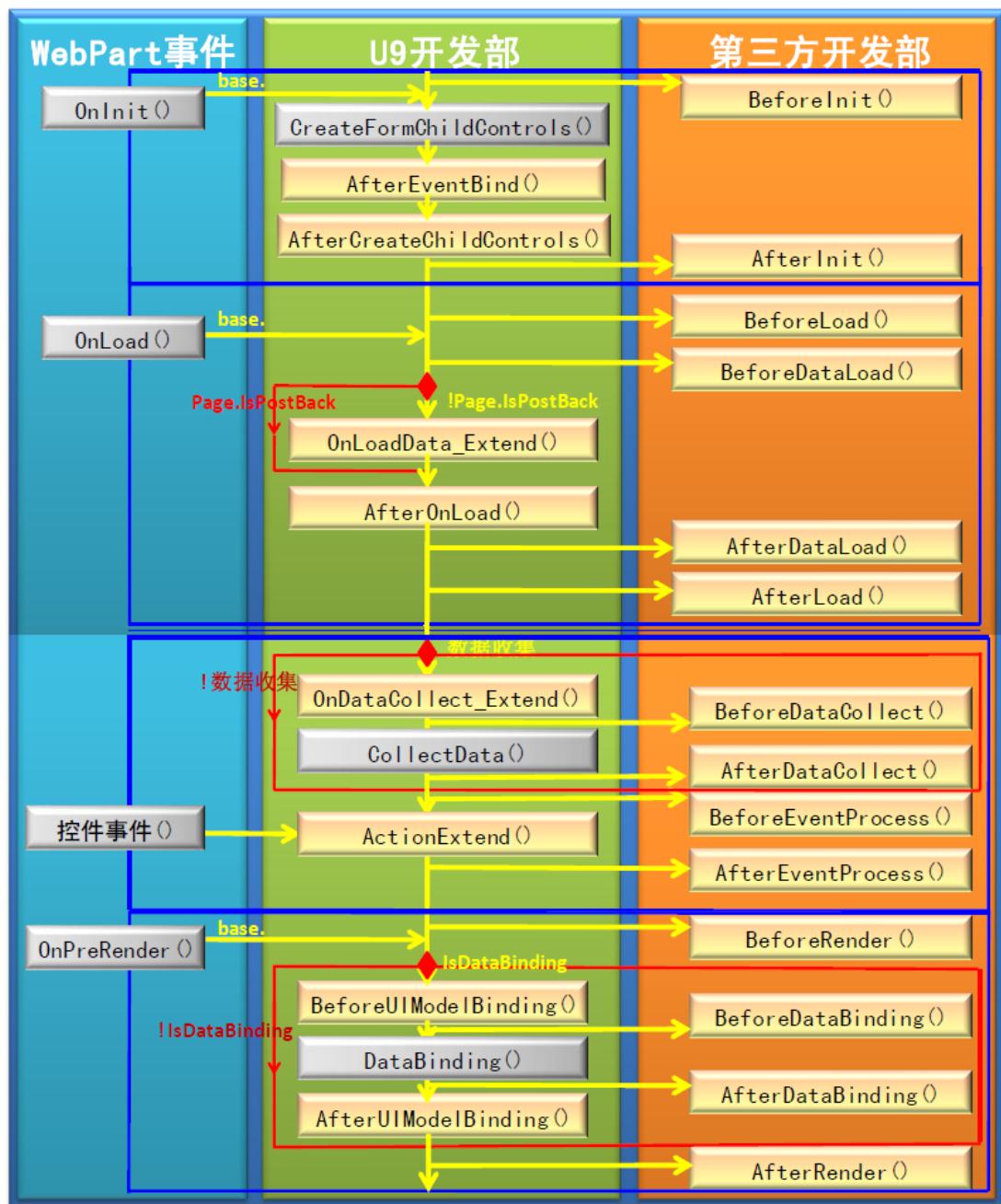




2) 表单逻辑扩展

表单逻辑扩展



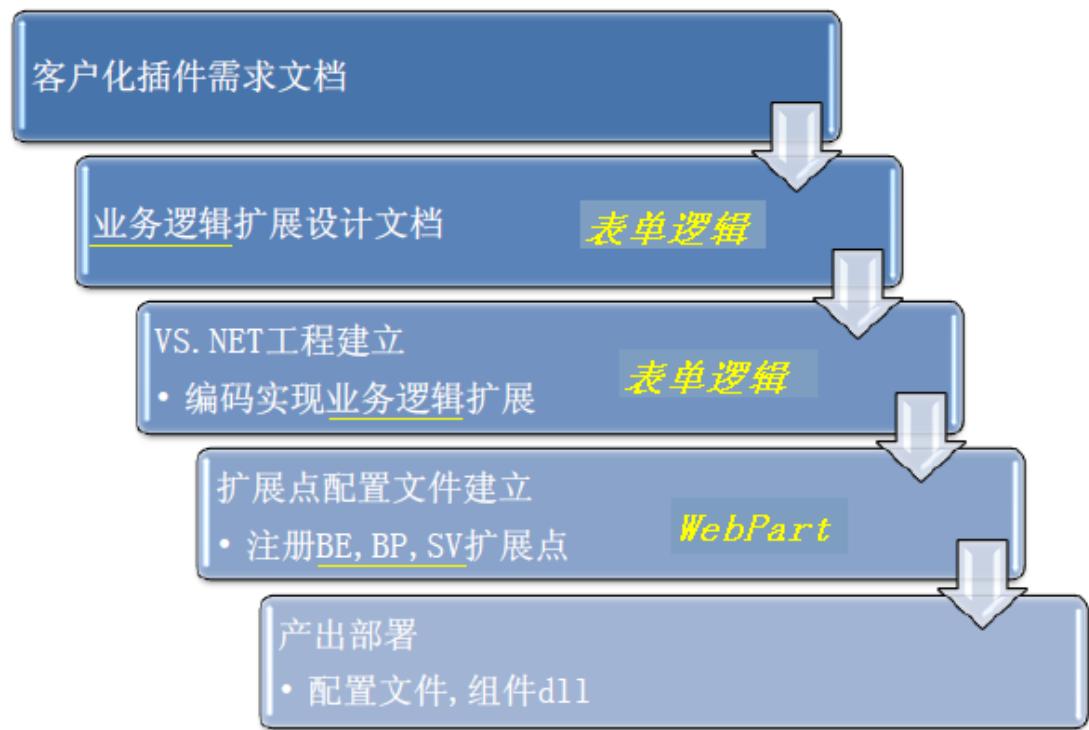


可写的逻辑	
AfterCreateChildControls()	创建控件树， 创建关联控件
OnLoadData_Extend()	非PostBack状态下创建控件树， 加载数据
AfterOnLoad()	创建控件树的最后机会， 也可以加载数据， 修改控件属性
ActionExtend()	针对每个事件的操作
BeforeUIModelBinding()	修改模型数据最后的机会，不然不会反映 到控件上，修改控件属性
AfterUIModelBinding()	修改控件属性

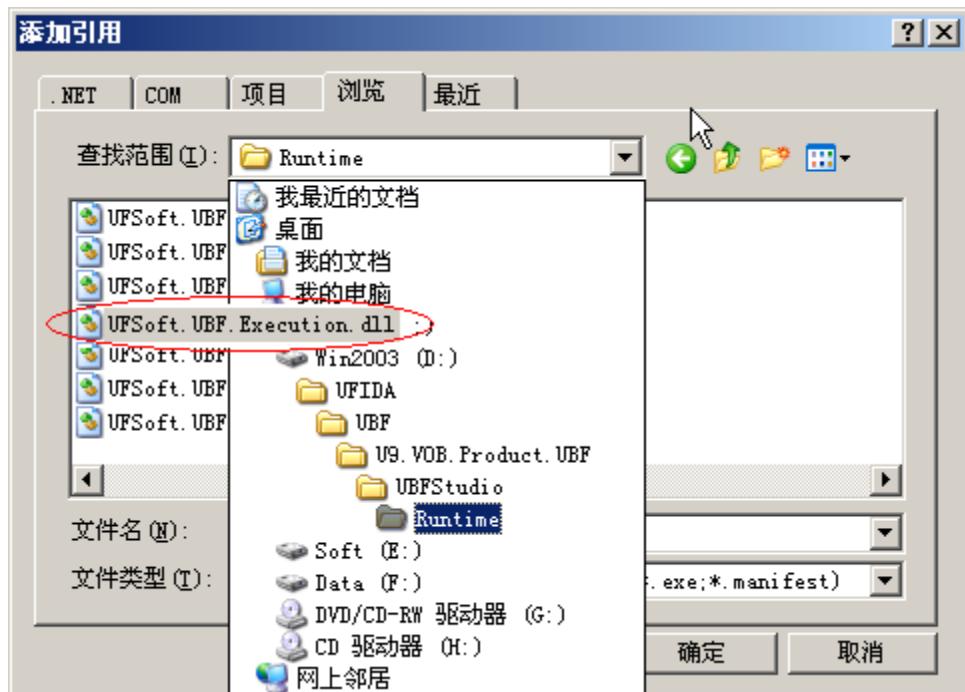
可写的逻辑	
Before/AfterInit(IPart Part, EventArgs args)	前： 初始化后 后： 创建控件树， 创建关联控件
BeforeLoad(IPart Part, EventArgs args)	Form加载前
BeforeDataLoad(IPart Part, out bool executeDefault) /AfterDataLoad(IPart Part)	数据加载前后
AfterLoad(IPart Part, EventArgs args)	Form加载后
BeforeAfterDataCollect(IPart Part, out bool executeDefault) /AfterDataCollect(IPart Part)	数据收集前后的处理
BeforeAfterEventProcess(IPart Part, string eventName, object sender, EventArgs args, out bool executeDefault) /AfterEventProcess(IPart Part, string eventName, object sender, EventArgs args)	针对每个事件的操作前后的处理
BeforeRender(IPart Part, EventArgs args)	修改模型和控件属性
BeforeDataBinding(IPart Part, out bool executeDefault) /AfterDataBinding(IPart Part)	前： 修改模型， 修改控件属性 后： 修改控件属性
AfterRender(IPart Part, EventArgs args)	修改控件属性

Part: 原Form对象， executeDefault: 是否执行系统默认操作， sender: 触发控件

二、 插件开发流程



D:\ufida\ubf\u9.vob.product.ubf\ubfstudio\runtime\UFSOFT.UBF.Business.dll
UFSOFT.UBF.Execution.dll
UFSOFT.UBF.PL.dll
UBF.System.dll



三、 BE插件开发

1、 代码引用

- 添加引用：UFSoft.UBF.Execution.dll
- 继承 UFSoft.UBF.Eventing.IEventSubscriber。
如：

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UFSoft.UBF.Util.Expression.Tool;
using UFSoft.UBF.Business;
using UFIDA.U9.CC.CreditObjectPolicy;
using UFIDA.U9.SM.SO;
using UFIDA.U9.CC.CreditParameter;
using UFIDA.U9.CBO.SCM.Customer;
using UFIDA.U9.Base.Doc;
using UFSoft.UBF.PL.Engine;
using UFIDA.U9.CC.Enum;
using UFIDA.U9.CBO.DTOs;

namespace UFIDA.U9.Cust.MX.SD.AppPlugIn
{
    [UFSoft.UBF.Eventing.Configuration.Failfast]
    class S0ModifySubscriber : UFSoft.UBF.Eventing.IEventSubscriber
    {
        public void Notify(params object[] args)
        {
            #region 从事件参数中取得当前业务实体
            //从事件参数中取得当前业务实体
            if (args == null || args.Length == 0 || !(args[0] is UFSoft.UBF.Business.EntityEvent))
            {
                return;
            }
            UFSoft.UBF.Business.BusinessEntity.EntityKey key =
((UFSoft.UBF.Business.EntityEvent)args[0]).EntityKey;
            if (key == null)
            {
                return;
            }
        }
    }
}
```

```

        }

        SM.SO.SO so = key.GetEntity() as SM.SO.SO;

        if (so == null)
        {
            return;
        }

        #endregion

        #region 信用控制

        if (so.BusinessType != BusinessTypeEnum.SD_StandardSale || so.ConfirmAccording == null) return;
        using (Session Session = Session.Open())
        {

            Dictionary<long, Dictionary<long, decimal>> dicMoney;
            dicMoney = GetSOLimitMoney(so);

            if (dicMoney != null && dicMoney.Count > 0)
            {

                foreach (long siteid in dicMoney.Keys)
                {

                    CustomerSite site = CustomerSite.Finder.FindByID(siteid);
                    ObjectCreditLevel ocl =
                    ObjectCreditLevel.Finder.Find("UFIDA::U9::CC::CreditObjectPolicy::ObjectCreditLevel.ControlPoint = " +
                    ControlPointEnum.All.Value.ToString() +
                    " and
                    UFIDA::U9::CC::CreditObjectPolicy::ObjectCreditLevel.ObjectCreditPolicy.CreditObject.CustomerSite.Code = '" +
                    site.Code +
                    "' and
                    UFIDA::U9::CC::CreditObjectPolicy::ObjectCreditLevel.ObjectCreditPolicy.CreditObject.Customer.Code = '" +
                    site.Customer.Code +
                    "' and
                    UFIDA::U9::CC::CreditObjectPolicy::ObjectCreditLevel.ObjectCreditPolicy.CreditObject.ObjectType = '" +
                    ObjectTypeEnum.CustomerSite.Value.ToString() +
                    "' and
                    UFIDA::U9::CC::CreditObjectPolicy::ObjectCreditLevel.CreditLevel.Organization = " + so.AccountOrg.ID.ToString());

                    if (ocl != null)
                    {

                        foreach (long currID in dicMoney[siteid].Keys)
                        {

                            foreach (ObjectCreditLevelCurrency oclc in ocl.ObjectCreditLevelCurrency)
                            {

                                //这里用的额币
                                if (oclc.CreditLimitCurrency.ID == currID)
                                {

                                    oclc.CreditContent.CreditLimit += dicMoney[siteid][currID];
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```
        }

    }

}

}

Session.Commit();

}

#endregion

}

}

Dictionary<long, Dictionary<long, decimal>> GetSOLimitMoney(SO so)

{

Dictionary<long, Dictionary<long, decimal>> result = new Dictionary<long, Dictionary<long, decimal>>();

SOLine.EntityList soLineList;

if (so.SysState == ObjectState.Deleted)

{

//删除状态下要用ID查

soLineList = SOLine.Finder.FindAll("so=" + so.ID.ToString());

}

else

{

soLineList = so.SOLines;

}

foreach (SOLine line in soLineList)

{

//是否占用额度

//if (!line.IsEngrossCreditLimit) continue;

//空对象时进行下一循环

if (line.CreditObj == null || line.CreditObj.CustomerSite == null)

{

continue;

}

decimal rate = 0;

try

{

//段值不为数字时不处理该Site额度

rate = decimal.Parse(line.CreditObj.CustomerSite.DescFlexField.PrivateDescSeg1.Trim());

}

catch

{

continue;

}

if (rate < 0 || rate > 1) continue;
```

```

        decimal money = 0;
        if (line.SysState == ObjectState.Inserted)
        {
            money = line.TotalMoneyTC * so.TCToACEExchRate * (1 - rate);
        }
        else if (line.SysState == ObjectState.Updated)
        {
            if (so.IsHolded == false && line.TotalMoneyTC != line.OriginalData.TotalMoneyTC)
            {
                //line.TCToCCEExchRate汇率都为零是否未计算? 这里暂时取so.TCToACEExchRate汇率
                money = (line.TotalMoneyTC - line.OriginalData.TotalMoneyTC) * so.TCToACEExchRate * (1
                - rate);
            }
            else if (so.IsHolded == true && so.OriginalData.IsHolded == false)
            {
                money = -line.TotalMoneyTC * so.TCToACEExchRate * (1 - rate);
            }
            else if (so.IsHolded == false && so.OriginalData.IsHolded == true)
            {
                money = line.TotalMoneyTC * so.TCToACEExchRate * (1 - rate);
            }
            else if (line.Status == SODocStatusEnum.CloseShort && line.OriginalData.Status ==
SODocStatusEnum.Approved)
            {
                //数量和累计出货计划数量取值是否正确
                if (so.IsPriceIncludeTax)
                {
                    money = -(line.TotalMoneyTC - line.SOLineSumInfo.SumShipPlanQtyPU *
line.FinallyPriceTC) * so.TCToACEExchRate * (1 - rate);
                }
                else
                {
                    money = -(line.TotalMoneyTC - line.SOLineSumInfo.SumShipPlanQtyPU *
line.FinallyPriceTC * (1 + line.TaxRate)) * so.TCToACEExchRate * (1 - rate);
                }
            }
            else if (line.Status == SODocStatusEnum.Approved && line.OriginalData.Status ==
SODocStatusEnum.CloseShort || so.Cancel.Canceled == true && so.OriginalData.Cancel.Canceled == false)
            {
                if (so.IsPriceIncludeTax)
                {
                    money = -(line.TotalMoneyTC - line.SOLineSumInfo.SumShipPlanQtyPU *
line.FinallyPriceTC) * so.TCToACEExchRate * (1 - rate);
                }
            }
        }
    }
}

```


2、配置文件

③ 扩展点配置文件

- 名称.sub.xml，部署到 Portal\Bin 下。
- 配置节点：

```
<pub-sub>
    <subscription event="UFIDA.U9.SM.SO.SO.Inserting">
        <subscriber type="UFIDA.U9.Cust.MX.SD.AppPlugin.SOModifySubscriber,UFIDA.U9.Cust.MX.SD.AppPlugin.dll" />
    </subscription>
</pub-sub>
```

注：event 为要扩展的 BE 事件，如：Inserting(插入前)、Inserted(插入后)、Updating(更新前)、Updated(更新后)、Deleted(删除后)。Validate

type 为插件类名和插件程序集。如：

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UFSOFT.UBF.Util.Expression.Tool;
using UFSOFT.UBF.Business;
using UFIDA.U9.CC.CreditObjectPolicy;
using UFIDA.U9.SM.SO;
using UFIDA.U9.CC.CreditParameter;
using UFIDA.U9.CBO.SCM.Customer;
using UFIDA.U9.Base.Doc;
using UFSOFT.UBF.PL.Engine;
using UFIDA.U9.CC.Enum;
using UFIDA.U9.CBO.DTOs;

namespace UFIDA.U9.Cust.MX.SD.AppPlugin
{
    [UFSOFT.UBF.Eventing.Configuration.Failfast]
    class SOModifySubscriber : UFSOFT.UBF.Eventing.IEventSubscriber
    {
        public void Notify(params object[] args)
        {
            #region 从事件参数中取得当前业务实体
            //从事件参数中取得当前业务实体
            if (args == null || args.Length == 0 || !(args[0] is UFSOFT.UBF.Business.EntityEvent))
            {
                return;
            }
            UFSOFT.UBF.Business.BusinessEntity.EntityKey key = ((UFSOFT.UBF.Business.EntityEvent)args[0]).EntityKey;
            if (key == null)
            {
                return;
            }
        }
    }
}
```

④

⑤ ②③④

插件式开发技术

BE扩展类继承和配置文件

● 继承的扩展类

- 添加引用UFSoft.UBF.Execution.dll
- 继承UFSoft.UBF.Eventing.IEventSubscriber

● 插件dll部署

部署到Portal\ApplicationServer\Libs

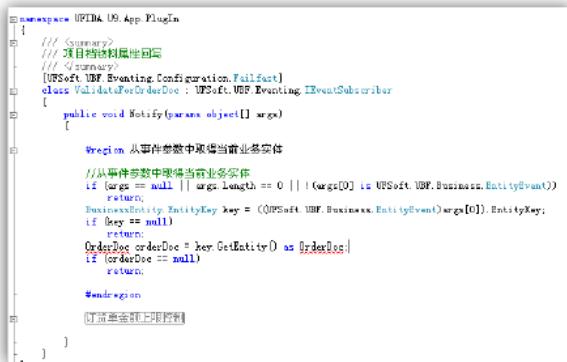
● 扩展点配置文件

- 文件：名称.sub.xml，部署到Portal\Bin\
- 配置节：

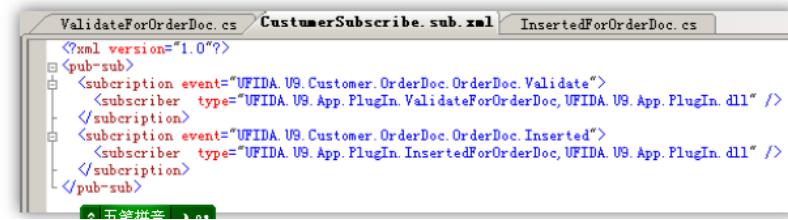
```
<subscription event="UFIDA.U9.Customer.OrderDoc.OrderDoc.Validate">
    <subscriber type="UFIDA.U9.App.Plugin.ValidateForOrderDoc,UFIDA.U9.App.Plugin.dll" />
</subscription>
```

说明：event：要扩展的BE事件，如Inserting, Inserted

type：插件类，插件程序集



```
namespace UFIDA.U9.App.Plugin
{
    // ...
    // ...
    [UFSoft.UBF.Eventing.Configuration.Failfast]
    class ValidateForOrderDoc : UFSoft.UBF.Eventing.IEventSubscriber
    {
        public void Notify(object[] args)
        {
            #region 从事件参数中根据当前业务读取
            //从事件参数中根据当前业务读取
            if (args == null || args.Length == 0 || !(args[0] is UFSoft.UBF.Business.EntityEvent))
                return;
            BusinessEntity.EntityKey key = (UFSoft.UBF.Business.EntityEvent)args[0].EntityKey;
            if (key == null)
                return;
            OrderDoc orderDoc = key.GetEntity() as OrderDoc;
            if (orderDoc == null)
                return;
            #endregion
            //...
        }
    }
}
```



```
<?xml version="1.0"?>
<pub-sub>
    <subscription event="UFIDA.U9.Customer.OrderDoc.OrderDoc.Validate">
        <subscriber type="UFIDA.U9.App.Plugin.ValidateForOrderDoc,UFIDA.U9.App.Plugin.dll" />
    </subscription>
    <subscription event="UFIDA.U9.Customer.OrderDoc.OrderDoc.Inserted">
        <subscriber type="UFIDA.U9.App.Plugin.InsertedForOrderDoc,UFIDA.U9.App.Plugin.dll" />
    </subscription>
</pub-sub>
```

3、 DLL部署

将 DLL 部署到 Portal\ApplicationServer\Libs。

四、 BP/SV插件开发

1、 代码引用

- 添加引用：UFSoft.UBF.AopFrame.dll
- 继承 UFSoft.UBF.Service.BPSVExtendBase。

```
using System;
using System.Collections.Generic;
using System.Linq;
using UFIDA.U9.SM.SO;

namespace UFIDA.U9.Cust.MX.SDRecheck.AppPlugIn
{
    class RecheckSO : UFSoft.UBF.Service.BPSVExtendBase
    {
        public override void AfterDo(object bp, ref object result) { }

        public override void BeforeDo(object bp)
```

```
{  
  
    ReCheckCreditSRV bpObj = (ReCheckCreditSRV)bp;  
    if (bpObj.SoList == null || bpObj.SoList.Count == 0)  
        return;  
    using (UFSoft.UBF.Business.ISession session = UFSoft.UBF.Business.Session.Open())  
    {  
        foreach (SOInfo info in bpObj.SoList)  
        {  
            SO so = SO.Finder.FindByID(info.SOID);  
            so.HoldReason = UFIDA.U9.Base.HoldRelease.HoldReason.Finder.Find("Code='SD01'");  
            so.ReleaseByCredit();  
        }  
        session.Commit();  
    }  
}  
}  
}
```

2、配置文件

⑥ 扩展点配置文件

- 文件：Portal\UBFConfig\UFIDA.UBF.BPSVExtend.config。
- 配置节点：

```
<BPSVExtend>  
    <Extend on="UFIDA.U9.SM.SO.ReCheckCreditSRV"  
            extendType="UFIDA.U9.Cust.MX.SDRecheck.AppPlugIn.RecheckSO,UFIDA.U9.Cust.MX.SDRecheck.AppPlugIn.dll"/>  
</BPSVExtend>
```

注：on 为要扩展的 BP/SV 事件

extendType 为插件类名和插件程序集。如：

```
using System.Collections.Generic;
using System.Linq;
using UFIDA.U9.SM.SO;

namespace UFIDA.U9.Cust.MX.SDRecheck.AppPlugIn
{
    class RecheckSO : UFSOFT.UBF.Service.BPSVExtendBase
    {
        public override void AfterDo(object bp, ref object result) { }

        public override void BeforeDo(object bp)
        {

            ReCheckCreditSRV bpObj = (ReCheckCreditSRV)bp;

            if (bpObj.SoList == null || bpObj.SoList.Count == 0)

                return;

            using (UFSOFT.UBF.Business.ISession session = UFSOFT.UBF.Business.Session.Open())
            {

                foreach (SOInfo info in bpObj.SoList)
                {
                    SO so = SO.Finder.FindByID(info.SOID);
                    so.HoldReason = UFIDA.U9.Base.HoldRelease.HoldReason.Finder.Find("Code='SD01'");
                    so.ReleaseByCredit();
                }

                session.Commit();
            }
        }
    }
}
```

3、 DLL部署

将 DLL 部署到 Portal\ApplicationServer\Libs。

五、 UI插件开发

1、 代码引用

- 添加引用：

```
// c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
// D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UBF.System.dll
D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.Controls.Interface.dll
// D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.FormProcess.dll
// D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.IView.dll
// D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.Engine.Runtime.dll
// D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.MD.Runtime.dll
D:\UFIDA\UBFV201\U9. VOB. Product. UBF\UBFStudio\Runtime\UFSoft.UBF.UI.WebControls.BaseComponent.dll
// 继承类 UFSoft.UBF.UI.Custom.ExtendedPartBase
using UFSoft.UBF.UI.ControlModel;
using UFSoft.UBF.UI.WebControlAdapter;
using UFSoft.UBF.UI.MD.Runtime;
using System.Collections.Specialized;
using System.Web.UI.WebControls;
using UFSoft.UBF.UI.IView;
```

-
- 继承 UFSOFT.UBF.UI.Custom.ExtendedPartBase。

2、 配置文件

⑦ 扩展点配置文件

- 文件 Portal\WebPartExtend.config。
- 配置节点：

```
<WebPartExtend>
    <ExtendedPart>
        parentPartFullName="UFIDA.U9.SCM.SM.SOUIModel.StandardSOMainUIFormWebPart"
        extendedPartFullName="UFIDA.U9.Cust.MX.SD.UIPlugIn.OrderDocProcess"
        extendedPartAssemblyName="UFIDA.U9.Cust.MX.SD.UIPlugIn.dll">
    </ExtendedPart>
</WebPartExtend>
```

注： parentPartFullName 为要扩展的原 Form 全名

extendedPartFullName 插件类全名

extendedPartAssemblyName 插件程序集

```
using UFIDA.U9.CBO.SCM.Customer;
using UFIDA.U9.CBO.SCM.Item;
using UFIDA.U9.Base.FlexField.DescFlexField;
using UFIDA.U9.CBO.SCM.ProjectTask;
using UFIDA.U9.Cust.MXBPF.Proxy;
using UFIDA.U9.Cust.MXBPF;
using UFIDA.U9.UI.PDHelper;
using UFIDA.U9.CBO.DTOs;
using UFIDA.U9.Cust.MX.BOM.FormulaBP.Proxy;
using UFIDA.U9.Base.Currency.Proxy;
using UFIDA.U9.Base.Currency;

namespace UFIDA.U9.Cust.MX.SD.UIPlugIn
{
    public class OrderDocProcess : UFSOFT.UBF.UI.Custom.ExtendedPartBase
    {
        //UI扩展的代码
        private IPart SOPart;
        private long? _soItemID = null;

        public override void BeforeDataCollect(UFSOFT.UBF.UI.IView.IPart Part, out bool executeDefault)
        {
            IUIView soLineView = Part.Model.Views["SO_SO_LINES"];
            if (soLineView != null && soLineView.FocusedRecord != null)
            {
                _soItemID = soLineView.FocusedRecord["ItemInfo_ItemID"] as long?;
            }
            base.BeforeDataCollect(Part, out executeDefault);
        }
    }
}
```

3、 DLL部署

将 DLL 部署到 Portal\UILib。