Namsara User Manual Script

www.pixysoft.net

Content

License

张辰
Reborn_zhang@hotmail.com

Introduction

Namsara 介绍

Namsara = .Net + Samsara (轮回)

是 ERP 中的数据流层，负责对输入数据进行各种逻辑处理，得到需要的输出数据，然后交给持久层进行数据库操作。

将 ERP 逻辑从代码中分离出来，以一套简单的 XML 规范进行自描述，提高重用性、可维护性、可扩展性。

需要与 Noebe 持久层紧密结合。

Namsara 逻辑介绍

概述：
ERP 的大部分处理针对数据库的数据，因此我设计了一套规范去约束这些处理，采用了数据流的思想，得到了数据流层框架 Namsara。

核心模块：
Namsara 的核心模块包括 Exchanger, Filter, Loader, Channel。通过这四个模块的综合运用，可以处理任意的 ERP 需求。

Exchanger：实现数据映射，包括简单的数据四则混合运算，字符串处理等。是数据流层的核心部分。例如订单生成到货单，通过配置 Exchanger,就可以实现。

Filter：数据的 bool 运算，其结果影响到 Channel 的数理（channel 下文介绍），产生分流。其作用类似程序语言的 IF 条件判断，满足不同需要的数据处理

Loader：加载外部数据，与持久层 Nemuria 紧密结合。目的在于引进新数据源到数据流。通过配置可以简单的向数据源添加需要的数据，满足数据流处理的要求，不需要写代码进行数据读取与加载。其结果根据加载是否成功产生分流。

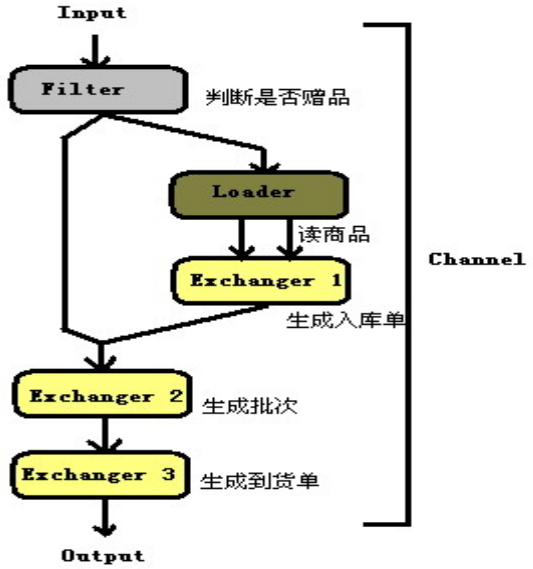
Channel：数据流层另外一个核心模块。他主要对以上三个模块进行链条式连接，形成类似流水线一样的数据处理。

模块小结：
简单的说，通过 Channel 对 Exchanger, Filter, Loader 进行装配，实现了数据流处理；Loader 加载了 Exchanger 需要的数据源，Filter 提供条件选择合适的 Exchanger, Loader 进行处理。

Namsara 的用例介绍

- 输入数据：界面输入的订货单信息：商品、数量
- Namsara 处理：通过输入数据，进行数据流的映射、过滤、装载得到入库单、批次、到货单等。
- 输出数据：入库单、批次、到货单等。

数据流图示例：



下面我就套用一个订单到货的例子进行文字分析上图：

Precondition:

- 有一张订货单，现在需要生成到货单，批次，入库单。

Requirement:

- 订货单中如果是赠品，则不生成入库单。
- 如果是非商品，则生成入库单。
- 所有商品生成到货单，批次

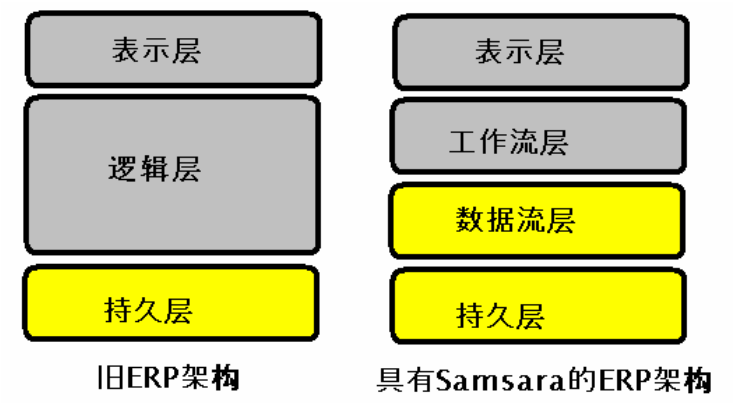
DataFlow:

- 赠品流：Filter 判断为赠品，进入 Exchanger2 生成批次
- 商品流：Filter 判断为商品，进入 Loader 读取数据库商品信息；进
- Exchanger1 生成入库单，进入 Exchanger2 生成批次
- 全部再进入 Exchanger3 生成到货单

上面体现了我设计的 Namsara 实际处理逻辑，即在 Filter 分流，在 Exchanger2 并流，然后再运行 Exchanger3。当然具体的程序逻辑下文再讲。

ERP 架构对比图

黄色部分代表可重用代码，不需要维护，扩展性强。
灰色部分代表不可重用代码，需要维护。



通过对比可以发现引入数据流层提高了项目开发的重用性。

架构分析

旧 ERP 架构逻辑层主要功能

- 取得界面的数据，并验证；进行数据的处理，得到需要的新数据（ERP 逻辑）；读取数据，交给界面显示

新架构分析

- 工作流层：取得界面的数据，并验证；读取数据，交给界面显示
- 数据流层：进行数据处理，得到新数据（ERP 逻辑）

通过对比可以发现，引入可数据流层，将 ERP 主要的逻辑分离出来，使代码集中于界面控制。

架构对比

旧 ERP 架构的问题

- 代码不可重用
- 增加一个模块就要增加一套代码
- 当发生数据表字段变更，字段数据变更的时候需要查找代码。
- 例如表字段改变
- 例如增加了个税率字段，需要查找数据来源，然后付值给它
- 当需要增加新逻辑的时候要搜索代码进行插入逻辑
- 在原有订单基础上增加一个批次生成，则需要加入代码，引入了错误概率
- 注意，以上的代码全部需要进行人工维护。

旧 ERP 架构商品 B 到货实例分析

一个到货操作：

```
internal void setIncomeForOrder(DataRow mainRow, DataTable commodityTable)
{
    InsertReceiveConfirmB(mainRow);
    InsertReceiveConfirmBCommodity(commodityTable, mainRow);
    InsertGroupStock(commodityTable, mainRow, GroupStockMapping);
    InsertWarehouseStock(commodityTable, mainRow, WarehouseStockMapping);
    InsertCommodityStockForAll(commodityTable, mainRow, CommodityStockForAllMapping);
    DataTable IntoStockTable = InsertIntoStock(mainRow, InstockMapping);
    InsertIntoStockCommodity(commodityTable, mainRow, IntoStockTable, InstockCommodityMapping);
}
```

一种一个方法的实现：

```
private void InsertCommodityStockForAll(Hashtable mapping, DataRow _newRow, DataRow _oldRow, DataRow _updateRow)
{
    myMapping.AutoMappingOneToOne(_oldRow, _updateRow);
    myMapping.MappingOneToOne(_newRow, _updateRow, mapping);
    float _oldAmount = float.Parse(_oldRow["ALLINTOSTOCKNUMBER"].ToString());
    float _oldPrice = float.Parse(_oldRow["ALLINTOSTOCKMONEY"].ToString());
    float _newAmount = float.Parse(_updateRow["LASTINTOSTOCKNUMBER"].ToString());
    float _newPrice = float.Parse(_updateRow["LASTINTOSTOCKMONEY"].ToString());
    float _resultAmount = _oldAmount + _newAmount;
    float _resultPrice = _oldPrice + _newPrice;
    _updateRow["ALLINTOSTOCKNUMBER"] = _resultAmount.ToString();
    _updateRow["ALLINTOSTOCKMONEY"] = _resultPrice.ToString();
    _updateRow["LASTINTOSTOCKTIME"] = this.getCurrentDate();
}
```

分析：

- 一个到货需要上面一套代码，与数据库字段紧密相关。此代码完全与其他逻辑代码混合在一起。并且没有完全包含了到货逻辑。
- 如果增加了模块？如果数据库字段变了？如果发现税率的计算不是这样？找代码，改代码。
- 有可能提高维护吗？不可能。库存的数据处理其实不简单！

使用数据流层架构分析

程序使用的代码：

```
private DataFlowOutput CreateWorkFlowByPass(DataRow mainRow, DataTable subTable, object para)
{
    DataFlowInput input = DataFlowManager.Instance.GetDataFlowInput();

    input.AddSource(mainRow);
    input.AddSource(subTable);
    input.AddSource("ITEMCODE", subTable, "ITEMCODE");
    input.AddSource("INWAREHOUSECODE", mainRow, "WAREHOUSECODE");
    input.AddSource("REALTIMEQTY", subTable, "QTY");
    input.AddSource("AFFIRMPERSON", para);

    input.AddSource("WORKMODULE", SYS_FUNCTIONTYPE.ToString("d"));
    input.AddSource("PRIMARYKEY", DatabaseUtility.GetPrimaryKeyValue(mainRow));
    input.AddSource("SUBMITPERSON", para);

    input.AddSource("BILLID", "HST_PURCHASEBILL", "PURCHASECODE");
    input.AddSource("AFFIRMQTY", subTable, "QTY");
    input.AddSource("SHOULDDELIEVERQTY", subTable, "QTY");
    input.AddSource("REALDELIEVERQTY", subTable, "QTY");

    DataFlowOutput output = DataFlowManager.Instance.Run(input,
        ChannelID.CHANNEL_HST_ORDERBILL_HST_ORDERBILLDETAIL_ITEMCODE_INWAREHOUSECODE_REALTIMEQTY_AFFIRMPERSON.ToString(),
        ChannelID.CHANNEL_STOCKWORKFLOW_SUBMITPERSON_WORKMODULE_PRIMARYKEY.ToString(),
        ChannelID.
_ITEMMOVELOG_RO_HST_STOCKWORKFLOW_INWAREHOUSECODE_OUTWAREHOUSECODE_ITEMCODE_SHOULDDELIEVERQTY_REALDELIEVERQTY_AFFIRMQTY_BILLID.ToString(),
        ChannelID.CHANNEL_HST_TMP_BRCPTDETAIL.ToString());

    return output;
}
```

使用界面写配置文件：

Identifier: EXCHANGER_HST_ITEMSTOCK_SUCC__HST_TMP_CITMRCPTBI　TableName: HST_ITEMSTOCK

Summary | Table | Object | NumericExpr | StringExpr

	Column	SrcTable	SrcColumn	Prefix	Postfix
▶	LASTMERCHANTCODE	HST_TMP_CITMRCPTBILL	MERCHANTCODE		
	LASTPURCHASETAXRATE	HST_TMP_CITMRCPTDETAIL	PURCHASETAXRATE		
	LASTPURCHASEPRICE	HST_TMP_CITMRCPTDETAIL	PURCHASEPRICEWITHTAX		
	LASTPURCHASEQTY	HST_TMP_CITMRCPTDETAIL	AFFIRMQTY		
	LASTPURCHASEAMT	HST_TMP_CITMRCPTDETAIL	PURCHASEPRICEWITHTAX		
*					

	Column	Expression	Prefix	Postfix
▶	TOTALPURCHASEQTY	[HST_ITEMSTOCK].[TOTALPURCHASEQTY]+[HST_ITEMSTOCK].[LASTPURCHASEQTY]		
	TOTALPURCHASEAMT	[HST_ITEMSTOCK].[TOTALPURCHASEAMT]+[HST_ITEMSTOCK].[LASTPURCHASEAMT]		
*				

可见引入了 Namsara，提高了代码的规范性，所有代码书写一致。通过修改配置文件提高了程序的扩展性。

性能分析

代码特点：

- 容易理解代码，维护代码：无论任何模块，所有具体数据流操作的模式一致，就是提供数据源，调用 Namsara，得到数据。
- 逻辑清晰：将数据流操作和其他逻辑完全分离
- 维护直观方便：使用 XML 进行直观的表述，不需要再次理解代码。

Worst Case Analysis

代码式编程与 XML 式编程

- Case:假设界面不变，数据库字段变动、表增加情况
- 代码式编程：代码式编程需要搜索代码空间，找到需要修改的部分，同时这种修改容易提高错误率，会影响到未知的代码部分。
- XML 编程：所有变动仅仅限制在 XML 部分，不需要修改原先任何代码！

代码式编程与 XML 式编程

- Case: 界面发生变化，界面引入了系统外的新数据。
- 代码式编程：需要修改界面代码，逻辑代码等。
- XML 编程：修改界面代码，逻辑代码。

可见，最坏情况下，XML 编程不会比代码编程差。

Summary

- 把需要经常变动的代码使用 XML 进行表述，增加可系统扩展性，易于维护。
- 持久层、数据流层的代码是不需要维护的。
- 难点：理解 XML 对数据流的描述规范，实际上就是理解配置文件的含义。

Quick Start

- 目的实现一个简单的映射
- 输入名为 HST_TMP_MOVEDETAIL 的表，表结构如下：

ID	编号	NUMBER(38)
MOVEBILLID	单据编号	CHAR(9)
ITEMCODE	商品编码	CHAR(7)
ITEMNAME	商品名称	VARCHAR2(50)
SHOULDDELIEVERQTY	应发数量	NUMBER(12,3)
REALDELIEVERQTY	实发数量	NUMBER(12,3)
AFFIRMQTY	确认数量	NUMBER(12,3)

- 处理前，表存在的数据如下：

ID	ITEMCODE	ITEMNAME	SHOULDDELIEVERQTY
1	0000001	联想电脑	12
2	0000002	神州电脑	11
3	0000003	代尔电脑	32

- 数据生成规则如下：
  - 填写 MOVEBILLID=0000555
  - REALDELIEVERQTY= SHOULDDELIEVERQTY-10
  - AFFIRMQTY=SHOULDDELIEVERQTY- REALDELIEVERQTY

- 编写配置文件

Namsara.Exchanger.config

```
<ArrayOfExchangeConfiguration>
  <ExchangeConfiguration>
  <Identifier>HST_TMP_MOVEDETAIL交换器</Identifier>
    <TableName>HST_TMP_MOVEDETAIL</TableName>
    <StringExprRelations>
      <StringExprRelation Column="MOVEBILLID" Expression="0000555" />
    </StringExprRelations>
    <NumericExprRelations>
    <NumericExprRelation Column="REALDELIEVERQTY" Expression="[HST_TMP_MOVEDETAIL].[SHOULDDELIEVERQTY]-10" />
<NumericExprRelation Column="AFFIRMQTY"
Expression="[HST_TMP_MOVEDETAIL].[SHOULDDELIEVERQTY]-[HST_TMP_MOVEDETAIL].[ REALDELIEVERQTY]" />
    </NumericExprRelations>
    <Memo />
  </ExchangeConfiguration>
```

```
</ArrayOfExchangeConfiguration>
```

Namsara.Channel.config

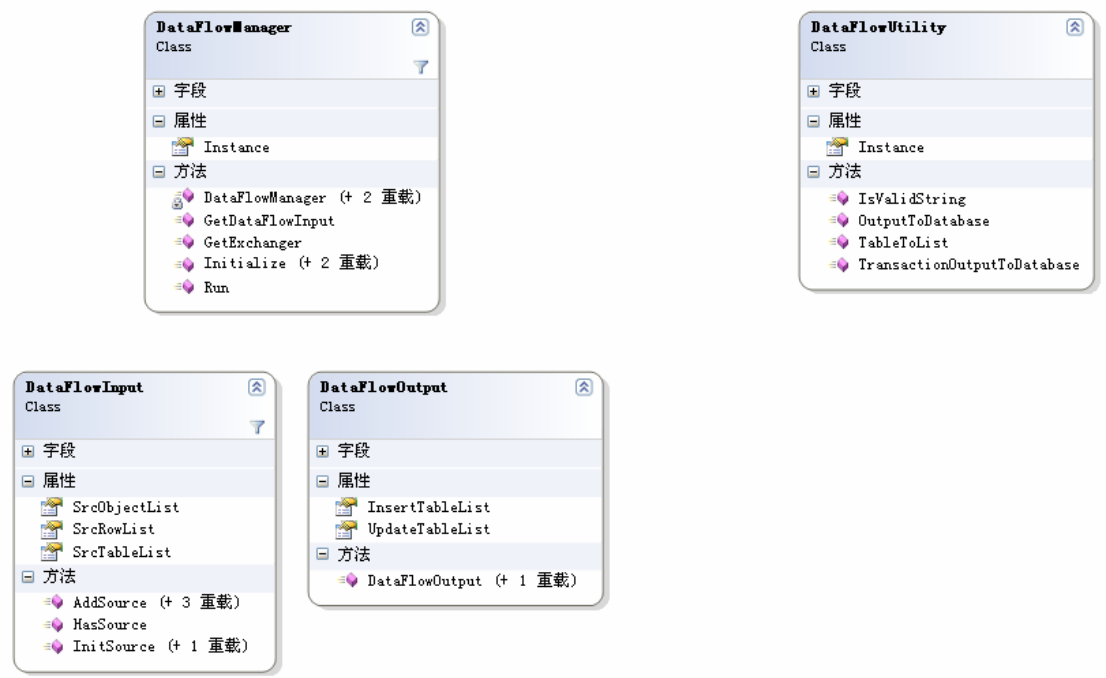
```
<ArrayOfChannelConfiguration>
  <ChannelConfiguration>
    <Identifier>HST_TMP_MOVEDETAIL通道</Identifier>
    <DirectPipelines>
      <DirectPipeline Identifier="1" ModuleID="HST_TMP_MOVEDETAIL交换器" />
    </DirectPipelines>
    <BoolPipelines/>
    <Memo />
    </ChannelConfiguration>
</ArrayOfChannelConfiguration>
```

- 将配置文件拷贝到：e:\temp
- 编写代码：

```
            DataTable table =
Pixysoft.Framework.Noebe.DatabaseManager.Instance.GetEntity("HST_TMP_MOVEDETAIL");

            DataRow row1 = table.NewRow();
            row1["ITEMCODE"] = "0000001";
            row1["ITEMNAME"] = "联想电脑";
            row1["SHOULDDELIEVERQTY"] = "12";
            table.Rows.Add(row1);

            DataRow row2 = table.NewRow();
            row2["ITEMCODE"] = "0000002";
            row2["ITEMNAME"] = "神州电脑";
            row2["SHOULDDELIEVERQTY"] = "11";
            table.Rows.Add(row2);

            DataRow row3 = table.NewRow();
            row3["ITEMCODE"] = "0000003";
            row3["ITEMNAME"] = "代尔电脑";
            row3["SHOULDDELIEVERQTY"] = "32";
            table.Rows.Add(row3);

            DataFlowManager.Initialize(@"e:\temp");
            DataFlowInput input = DataFlowManager.Instance.GetDataFlowInput();
            input.AddSource(table); //注意，这个table就是上文说的，里面包含了数据
            DataFlowOutput output = DataFlowManager.Instance.Run(input, "HST_TMP_MOVEDETAIL通道");
```

- 这样output就保存了处理后的数据

API



DataFlowManager

- 初始化 Namsara
- 获得 Input
- 运行

DataFlowInput

- 添加需要处理的数据源和需要依赖的数据源

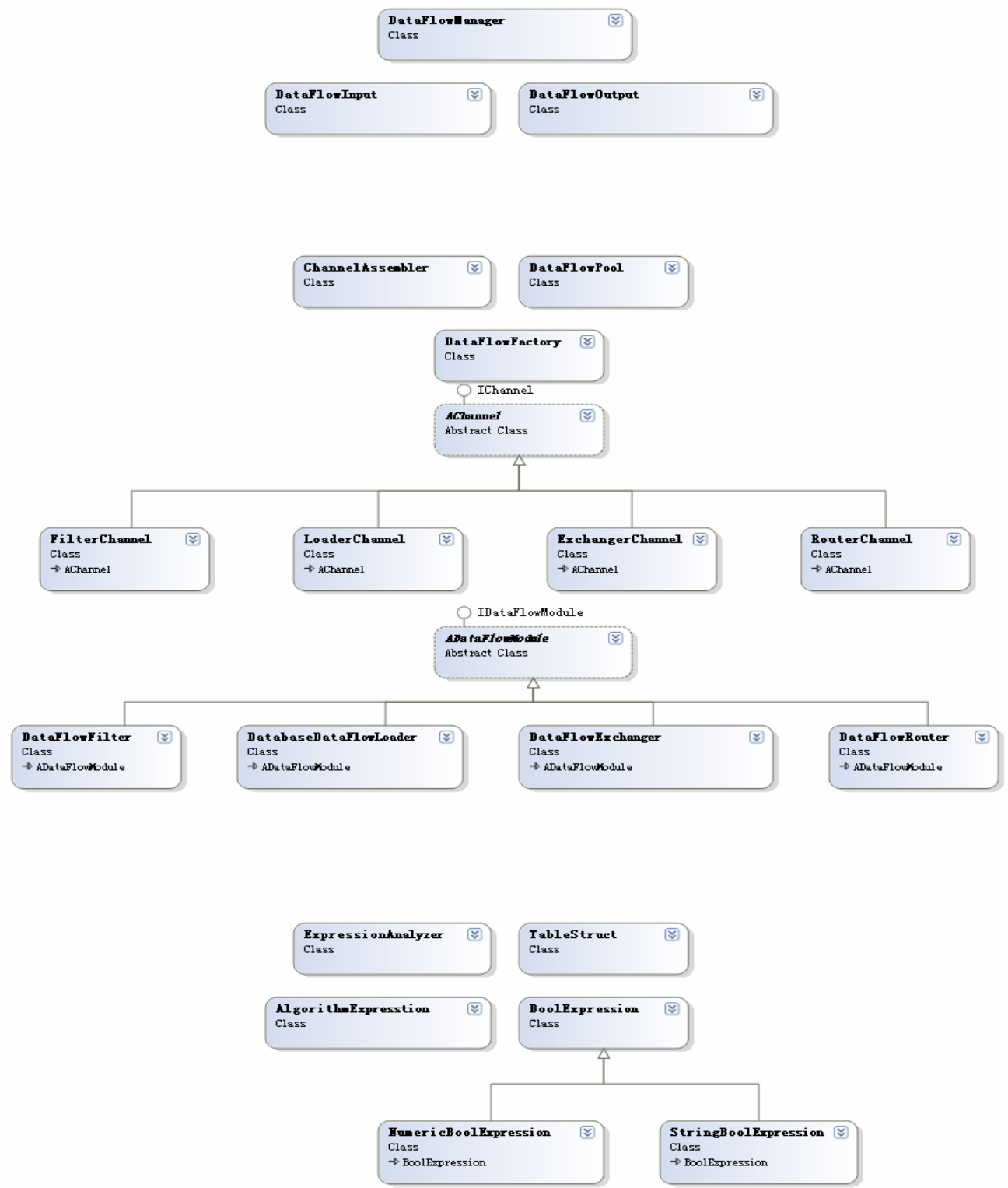
DataFlowOutput

- 输出处理结果，分为 Insert / Update 两种，一般 Loader 装载成功的都属于 Update，其他都是 Insert

DataFlowUtility

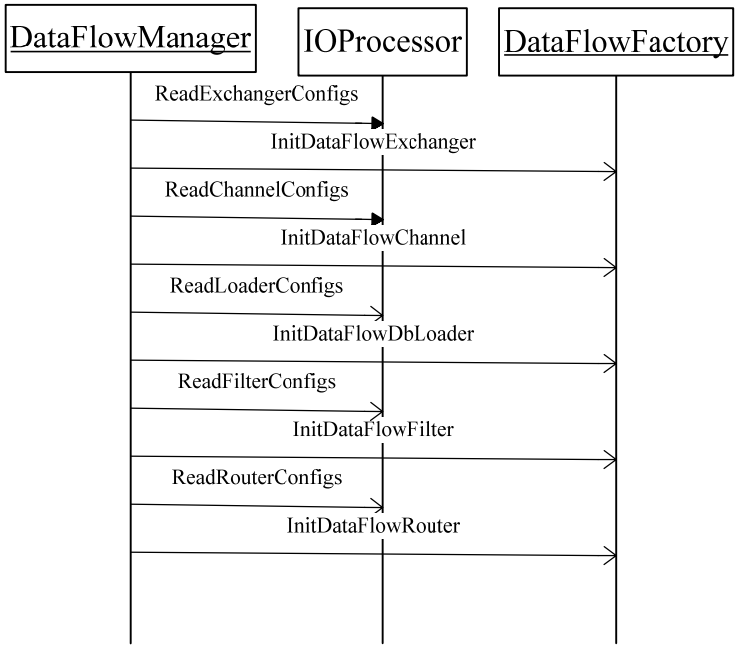
- 数据库的一些操作，自动识别 Output 的 insert/update，进行插数据库。

Architecture 架构

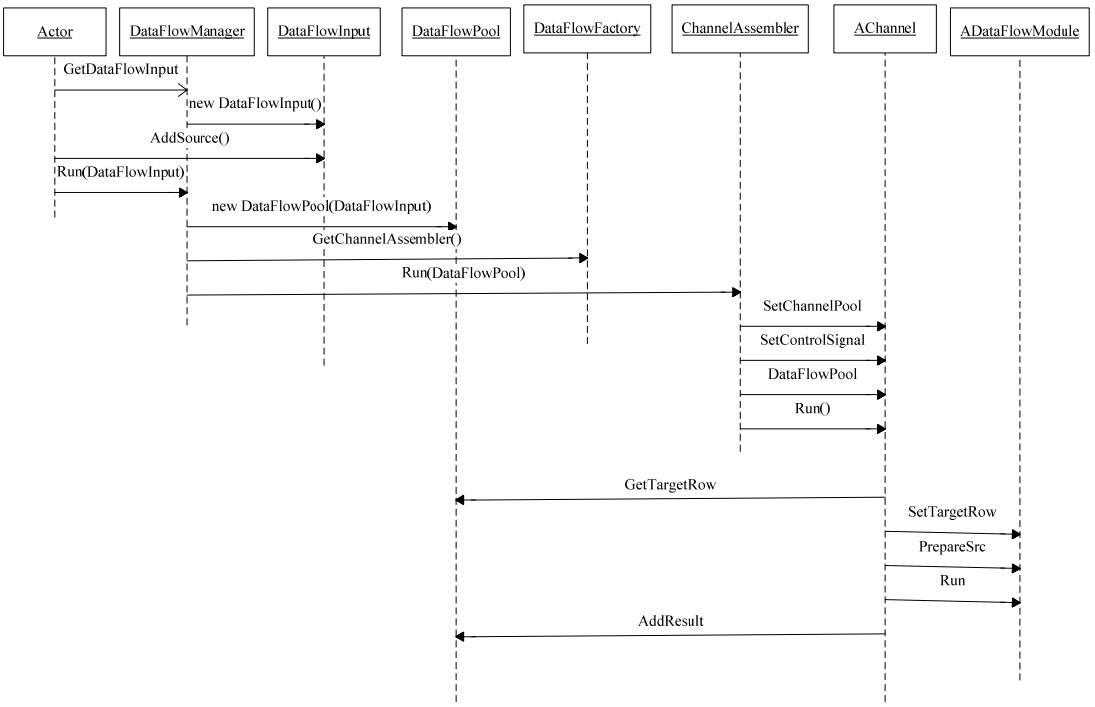


Sequence Diagrams

初始化



Run



Configurations 配置文件详解

关键词

Channel 通道、Pipeline 管道、Exchanger 交换器、Loader 装载器、Filter 过滤器、Router 路由器

一句话描述

为数据流层的配置文件，数据流层通过解析这些配置文件实现各种 ERP 数据操作需求。

UML

ChannelConfiguration

概述：

- 不同类型的管道首尾相接形成通道，构成数据流运算。
- 每个管道对应当前管道 id，后继 id，管道对应模块 id

文件名：

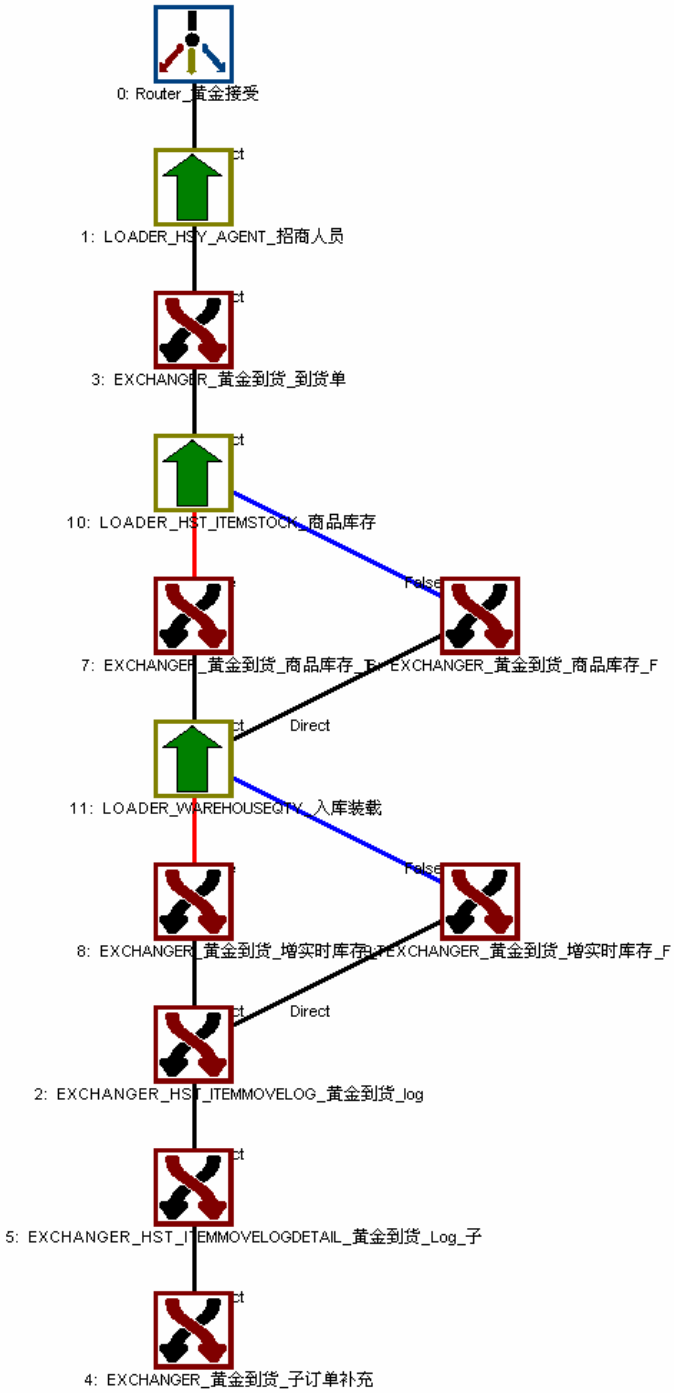
Namsaral.Channel.*.config

实例：

```
<ArrayOfChannelConfiguration>
  <ChannelConfiguration>
    <Identifier>CHANNEL_黄金到货_C</Identifier>
    <SourceID />
    <DirectPipelines>
      <DirectPipeline Identifier="0" Successor="1" ModuleID="Router_黄金接受" />
      <DirectPipeline Identifier="1" Successor="3" ModuleID="LOADER_HSY_AGENT_招商人员" />
      <DirectPipeline Identifier="3" Successor="10" ModuleID="EXCHANGER_黄金到货_到货单" />
      <DirectPipeline Identifier="7" Successor="11" ModuleID="EXCHANGER_黄金到货_商品库存_T" />
      <DirectPipeline Identifier="6" Successor="11" ModuleID="EXCHANGER_黄金到货_商品库存_F" />
      <DirectPipeline Identifier="8" Successor="2" ModuleID="EXCHANGER_黄金到货_增实时库存_T" />
      <DirectPipeline Identifier="9" Successor="2" ModuleID="EXCHANGER_黄金到货_增实时库存_F" />
      <DirectPipeline Identifier="2" Successor="5" ModuleID="EXCHANGER_HST_ITEMMOVELOG_黄金到货_log" />
      <DirectPipeline Identifier="5" Successor="4" ModuleID="EXCHANGER_HST_ITEMMOVELOGDETAIL_黄金到货_Log_子"
/>
      <DirectPipeline Identifier="4" ModuleID="EXCHANGER_黄金到货_子订单补充" />
    </DirectPipelines>
    <BoolPipelines>
      <BoolPipeline Identifier="10" RollToSuccId="7" RollToFailId="6" ModuleID="LOADER_HST_ITEMSTOCK_商品库存
" />
      <BoolPipeline Identifier="11" RollToSuccId="8" RollToFailId="9" ModuleID="LOADER_WAREHOUSEQTY_入库装载"
/>
    </BoolPipelines>
    <Memo />
    </ChannelConfiguration>
</ArrayOfChannelConfiguration>
```

实例解释：

- 上面构成了下面的数据流：
- 读取招商人员信息；填写黄金到货单；读取商品库存表；如果读取成功填写成功的黄金库存表否则填写读取失败的黄金库存表；读取库存；如果读取成功就修改库存否则创建库存；生成黄金移动记录主表；声称黄金移动字表；补充黄金订单信息



配置文件解释：

ChannelConfiguration

Identifier	全局唯一标识符
DirectPipelines	直流管道配置，在数据流里面直线流动
BoolPipelines	二分流管道配置，通过特定的模块运算得到的 bool 结果进行分流操作
Memo	备注

DirectPipeline

Identifier	当前管道 id

Successor	后继管道 id
ModuleID	管道对应的处理模块 id

BoolPipeline

Identifier	当前管道 id
RollToSuccId	布尔真的后继管道 id
RollToFailId	布尔假的后继管道 id
ModuleID	管道对应的处理模块 id；注：只有 Filter / Loader 支持 BoolPipeline

ExchangeConfiguration

概述：

- 交换器模块配置
- 实现表数据填充、运算

文件名：

Namsaral.Exchanger.*.config

实例：

```
<ArrayOfExchangeConfiguration>
  <ExchangeConfiguration>
<Identifier>EXCHANGER_HST_PURCHASEBILL__HST_TMP_BRCPTBILL_HST_TMP_BRCPTDETAIL_AFFIRMPERSON</Identifier>
    <TableName>HST_PURCHASEBILL</TableName>
    <Memo />
    <NumericExprRelations>
      <NumericExprRelation Column="PURCHASETAX"
Expression="[HST_PURCHASEBILL].[PURCHASEAMTWITHTAX]-SUM[HST_TMP_BRCPTDETAIL].[PURCHASEPRICEWITHOUTTAX]" />
    </NumericExprRelations>
    <StringExprRelations>
      <StringExprRelation Column="PURCHASEDATE" Expression="OBJECT.[SYS_DATETIME]" />
      <StringExprRelation Column="CHECKEDTAX" Expression="0" />
    </StringExprRelations>
  </ExchangeConfiguration>
</ArrayOfExchangeConfiguration>
```

实例解释：

- 对 HST_PURCHASEBIL 的 PURCHASETAX、PURCHASEDATE、CHECKEDTAX 进行填充
- PURCHASETAX 使用了四则混合运算映射
- PURCHASEDATE 使用了通用对象映射（重用设计，下文介绍）
- CHECKEDTAX 填 0

配置文件解释：

ChannelConfiguration

Identifier	全局唯一标识符

TableName	模块需要处理的表名
NumericExprRelation	四则运算映射
StringExprRelation	字符串映射
Memo	备注

NumericExprRelation、StringExprRelation

Column	需要处理的列名
Expression	具体的映射表达式

LoaderConfiguration

概述：

- 通过配置文件自动将数据库表装载到数据流层，而不需要写代码

文件名：

Namsaral.Loader.*.config

实例：

```
<ArrayOfLoaderConfiguration>
  <LoaderConfiguration>
    <Identifier>LOADER_WAREHOUSEQTY_出库装载</Identifier>
    <TableName>HST_WAREHOUSEQTY</TableName>
    <Memo />
    <SQL>SELECT * FROM HST_WAREHOUSEQTY WHERE WAREHOUSECODE=:WAREHOUSECODE AND ITEMCODE=:ITEMCODE</SQL>
    <Exported>true</Exported>
    <NumericExprRelations />
    <StringExprRelations>
      <StringExprRelation Name="WAREHOUSECODE" Expression="OBJECT.[OUTWAREHOUSECODE]" DataType="string"
IsNullable="false" />
      <StringExprRelation Name="ITEMCODE" Expression="OBJECT.[ITEMCODE]" DataType="string" IsNullable="false"
/>
    </StringExprRelations>
  </LoaderConfiguration>
  </ArrayOfLoaderConfiguration>
```

实例解释：

- 选择仓位编码 WAREHOUSECODE 和商品编码 ITEMCODE 下的库存表的数据。

配置文件解释：

LoaderConfiguration

Identifier	全局唯一标识符
TableName	模块需要装载的表名
SQL	SQL
Exported	装载的表是否输出给用户，如果不输出，则作为数据流层内部使用数据
NumericExprRelation	四则运算映射

StringExprRelation	字符串映射
Memo	备注

NumericExprRelation、StringExprRelation

Name	SQL 里面对应的参数名
Expression	具体的映射表达式
DataType	数据类型，包括： string char int long float double boolean byte object byte[] DateTime
IsNullable	是否允许为空，如果使用模式生成 SQL，则此选项很重要。模式生成下文介绍

FilterConfiguration

概述：

- 根据配置对数据进行 bool 运算，运算结果影响到数据流的流向。例如判断某表字段数值是否大于一个阈值。

文件名：

Namsaral.Filter.*.config

实例：

```
<ArrayOfFilterConfiguration>
  <FilterConfiguration>
    <Identifier>FILTER_HST_TMP_CITMRCPTDETAIL_C到货</Identifier>
    <Memo />
    <StringExpression>[HST_TMP_CITMRCPTDETAIL].[ISITEM]==T</StringExpression>
    </FilterConfiguration>
  </ArrayOfFilterConfiguration>
```

实例解释：

- 判断表 HST_TMP_CITMRCPTDETAI 的字段 ISITEM 是否等于 T，用于判断输入的是否为商品

配置文件解释：

FilterConfiguration

Identifier	全局唯一标识符

NumericExpression	布尔四则混合运算映射
StringExpression	布尔字符串映射
Memo	备注

RouterConfiguration

概述：

- 针对映射中的对象映射，具体赋予值。提高重用性
- 例如上文的 OBJECT.[ITEMCODE]这里可以具体实现指定来自什么表
- 通过对映射表达式里使用 OBJECT.[ID]以及配置 Router 实现通用模块，提高重用性。

文件名：

Namsaral.Router.*.config

实例：

```
<ArrayOfRouterConfiguration>
  <RouterConfiguration>
    <Identifier>Router_订单</Identifier>
    <Memo />
    <ObjMappings>
      <ObjMapping Identifier="CONTRACTCODE" TableName="HST_ORDERBILL" ColumnName="CONTRACTCODE" />
      <ObjMapping Identifier="PRIMARYKEY" TableName="HST_ORDERBILL" ColumnName="ORDERCODE" />
      <ObjMapping Identifier="INWAREHOUSECODE" TableName="HST_ORDERBILL" ColumnName="WAREHOUSECODE" />
      <ObjMapping Identifier="ITEMCODE" TableName="HST_ORDERBILLDETAIL" ColumnName="ITEMCODE" />
      <ObjMapping Identifier="WAREHOUSECODE" TableName="HST_ORDERBILL" ColumnName="WAREHOUSECODE" />
    </ObjMappings>
    <ValueMappings />
    </RouterConfiguration>
</ArrayOfRouterConfiguration>
```

实例解释：

- 具体指定了对象 CONTRACTCODE 等的数据源

配置文件解释：

FilterConfiguration

Identifier	全局唯一标识符
TableName	模块需要处理的表名
ObjMappings	具体指定对象的值来的数据源表
ValueMappings	具体指定对象的值是什么
Memo	备注

ObjMapping

Identifier	对象编号
TableName	数据源的表名

ColumnName	具体指定对象的值来自什么表
Prefix	映射后的数据的前缀（可以再补充数据）

ValueMapping

Identifier	对象编号
Value	直接指定对象的值

Module

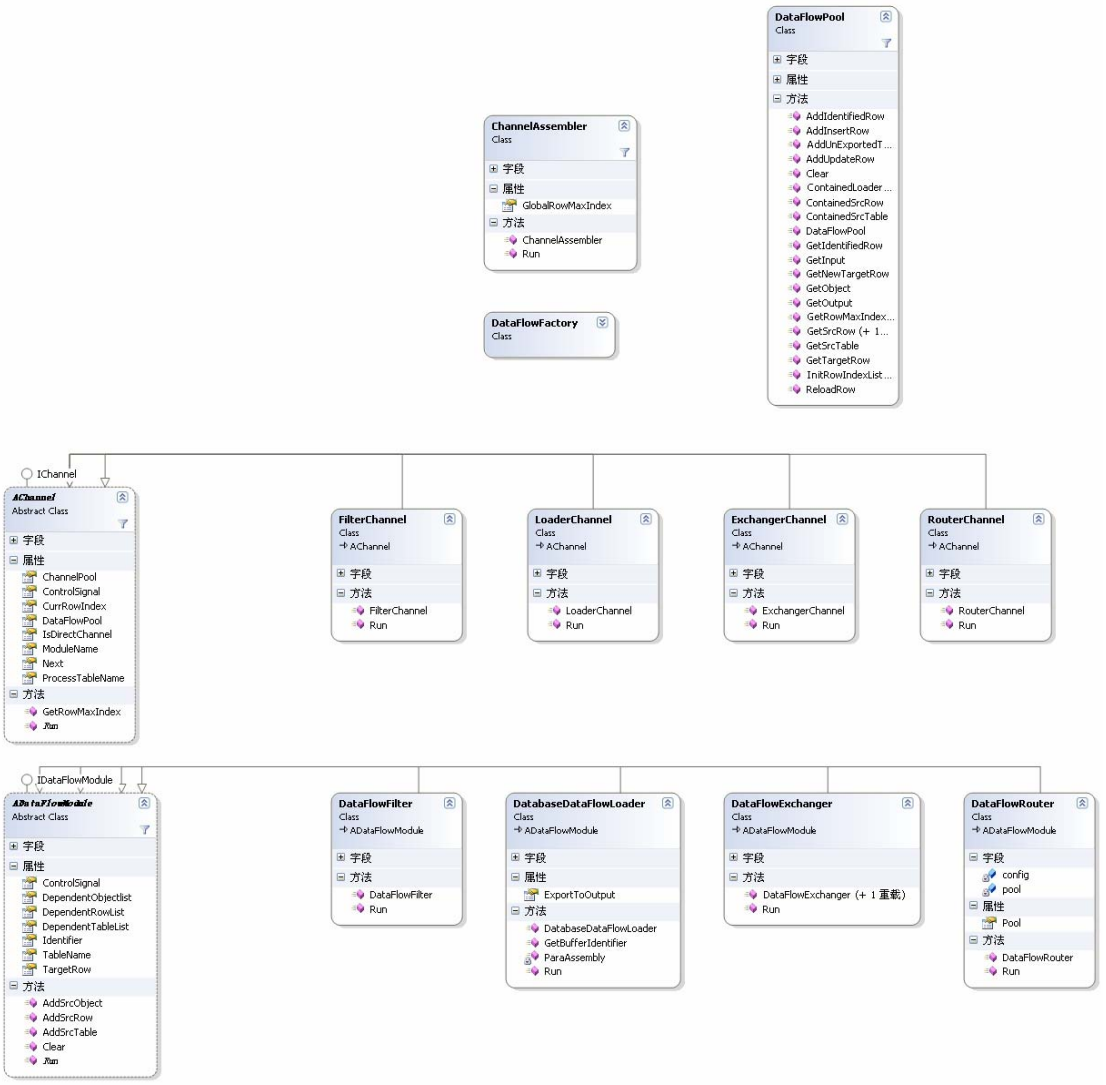
关键词

直流、二分流

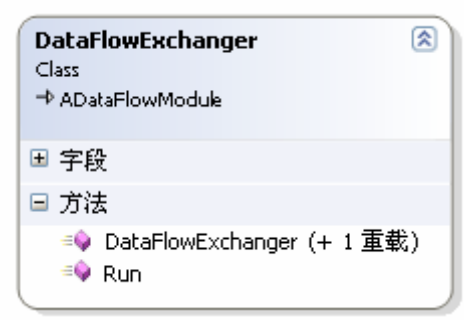
一句话概述

ChannelAssembler 将对应的 ExchangerChannel、LoaderChannel、RouterChannel、FilterChannel 进行链条式组织，实现数据流处理。每个 Channel 对应一个模块，此模块依赖配置文件进行驱动。

UML

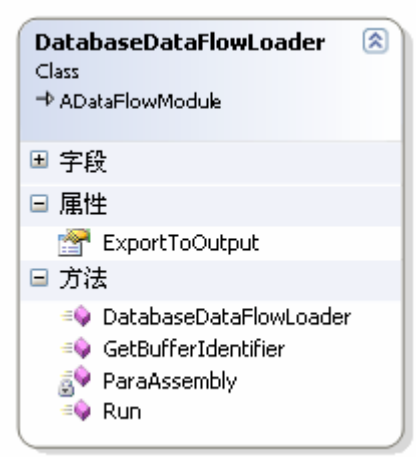


Exchanger



- 交换器，根据配置的映射进行表填充，得到需要的表数据。
- 只能配置为直流

Loader

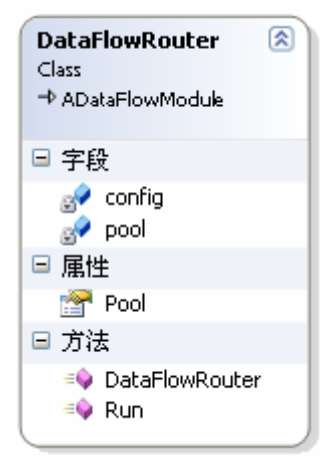


- 装载器，根据配置文件读取数据库得到需要的表，而不需要写代码。
- 根据装载结果分为装载失败、装载成功。其结果能够影响到数据流的流向。
- 可以直流、二分流
- 支持缓存操作

Filter

- 过滤器，根据配置文件的表达式计算 bool 值，影响流的方向
- 支持直流、二分流

Router



- 路由器，与模块配置文件中映射的对象表达式配合使用，制定对象表达式的数据源，实现模块的重用

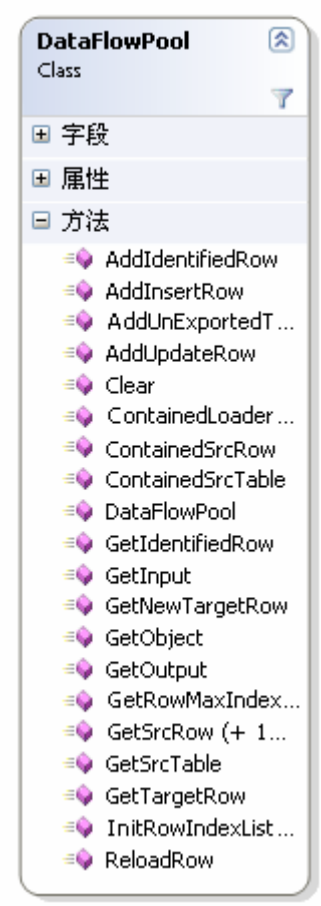
Channel

- 一共有四种 Channel，针对上面 4 种模块。每个 Channel 负责调用模块，得到需要的结果。

ChannelAssembler

Channel 装配器，负责将上面四种 Channel 根据配置文件进行链条式装配，实现数据流处理。

DataFlowPool



数据池，是数据流层的另外一个核心模块，负责向各模块提供数据源，同时保存各模块生成的数据。具体在数据池专栏中介绍。

Mappings

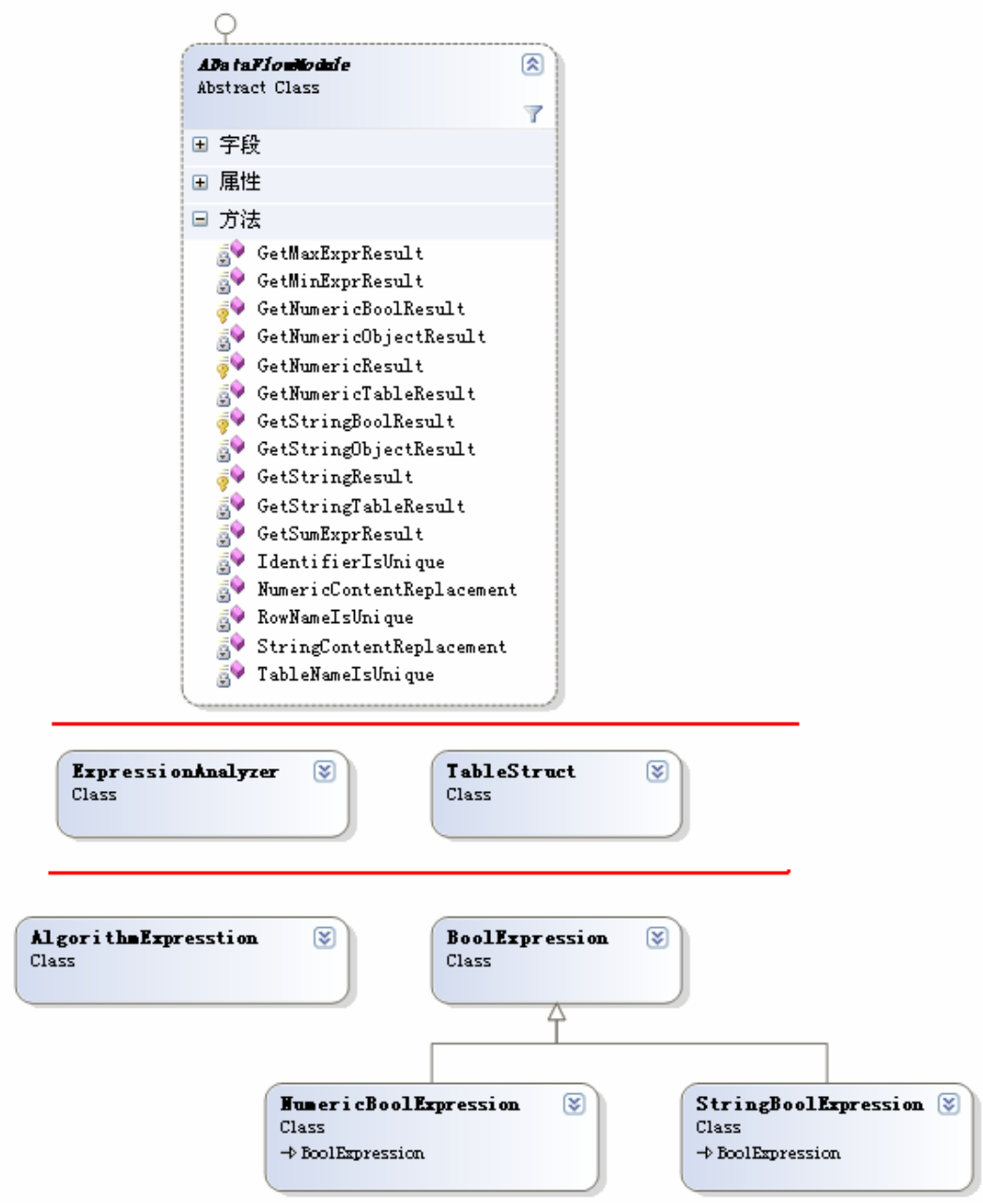
关键词

映射、表达式、对象表达式

一句话概述

整个数据流层通过不同的映射机制对需要的数据表进行填充(Exchanger)、对加载的 SQL 参数赋值(Loader)、计算过滤器的布尔值(Filter)。因此映射的能力直接影响到数据流层。

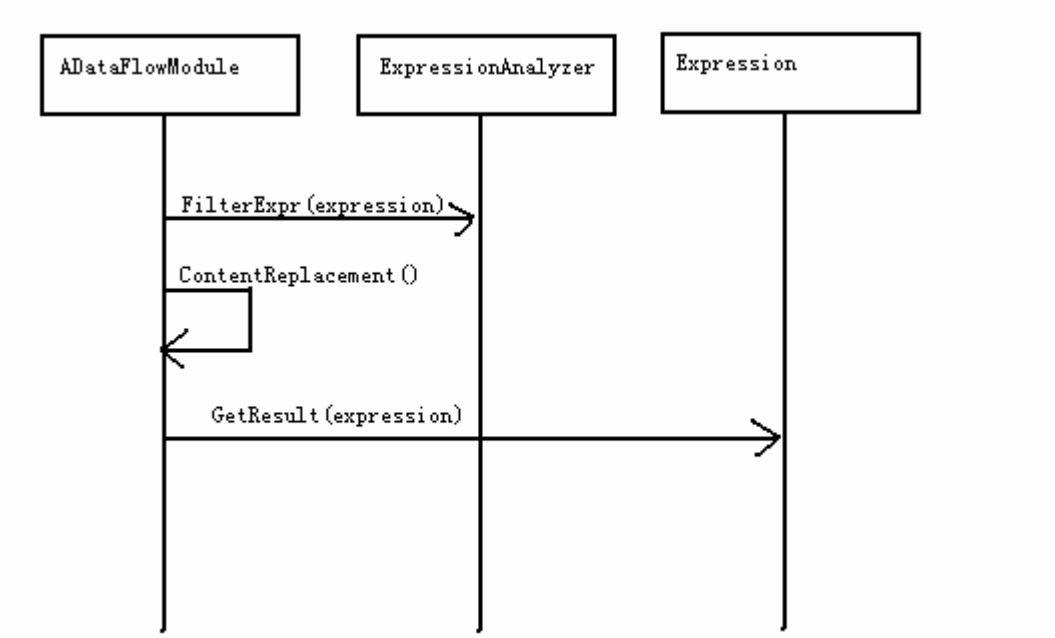
映射解析器 UML



说明：

- 每个模块（Exchanger 等）都继承 ADataFlowModule，其具体解析模块的映射表达式。
- 首先通过 ExpressionAnalyzer 分析表达式，提取出表达式模式
- 根据分析结果对表达式进行数据赋值，得到具体带数据的表达式。
- 把表达式交给计算器计算，得到结果。

时序图



映射

数据流层提供了多种映射，包括四则混合运算映射、字符串映射、布尔四则混合运算映射、布尔字符串映射。具体如下

各模块对映射的支持

配置模块名	支持映射类型
ChannelConfiguraion	无
ExchangeConfiguration	四则混合运算映射、字符串映射
LoaderConfiguration	四则混合运算映射、字符串映射
RouterConfiguration	布尔四则混合运算映射、布尔字符串映射
FilterConfiguration	无

映射实例

四则混合运算映射

```
[HST_PURCHASEBILL].[PURCHASEAMTWITHTAX]-SUM[HST_TMP_BRCPTDETAIL].[PURCHASEPRICEWITHOUTTAX]
```

- 对 HST_TMP_BRCPTDETAIL 的字段 PURCHASEPRICEWITHOUTTA 求合
- 表 PURCHASEBILL 的字段 PURCHASEAMTWITHTAX 减去上面的求和结果，得到最终结果

字符串映射

```
[HST_PURCHASEBILL].[PURCHASECODE]+[HST_TMP_BRCPTDETAIL].[SYSISITEM]
```

- 表 PURCHASEBILL 的字段 PURCHASECODE 与表 BRCPTDETAIL 的字段 SYSISITEM 字符串相加

布尔四则混合运算映射

```
a > b && (c > d || e > f)
```

- 根据上面的表达式计算 bool 值

布尔字符串映射

```
a == b || c == d
```

- 根据上面的表达式计算 bool 值

映射中支持的映射表达式

sum[TABLENAME].[COLUMNNAME]	对表字段求和
max[TABLENAME].[COLUMNNAME]	对表字段求最大值
min[TABLENAME].[COLUMNNAME]	对表字段求最小值
[TABLENAME].[COLUMNNAME]	得到表字段值
PK.[TABLENAME]	得到表主键值
OBJECT.[IDENTIFIER]	得到对象值（与 Router 配合用于通用模块设计）

映射中提供的默认对象表达式

OBJECT.[SYS_DATETIME]	系统时间
OBJECT.[SYS_NULL]	Null 值
OBJECT.[SYS_SERIAL]	表当前数据计数器（例如字段有 3 条数据，当前为第几条）

映射中的对象表达式

- 在映射中通过使用对象表达式和 Router 模块相结合，能够提高模块的重用性。
- 例如有移动单确认、配货单确认等多个模块，他们确认的数据流大致相同，因此只要设计出一套确认模块，然后里面使用对象表达式，最后在对应的通道配置（Channel）里面使用对应的 Router 指定对象表达式的数据源就实现了模块通用

Channel 核心思想

关键词

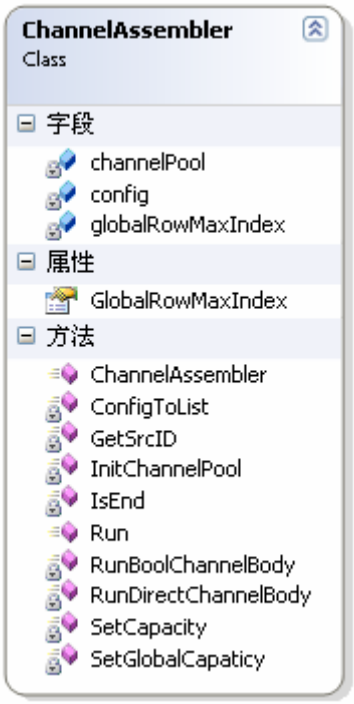
直流、二分流、单流、多流

- 直流、二分流是 Channel 的属性，在配置文件中实现对流的流向控制。
- 单流、多流是数据表数据量的标示，如果表只有一条数据则是单流、否则是多流。

一句话概述

ChannelAssembler 对各个 Channel 进行装配，每个 Channel 又对应一个 Module。大家构成数据流，达到任意逻辑处理。

UML



解析

- 在 ChannelAssembler 里面有个 Channel Pool；由于所有 Module 使用了全局 ID，初始化的时候，Assembler 读取 Channel 的配置文件，创建 Channel，并把对应的 Module 加载到 Channel 里面，最后保存在 ChannelPool 里。
- 运行的时候，首先找到数据流起点，默认是没有入度的节点作为起点，因此逻辑上允许有多个起点。且多条数据流之间不共用 Pool 的装载层，但是共用数据层。
- 运行的时候，得到当前 Channel 的 rowIndex，就是流标识，得到 Channel 的类型（直流、二分流），如果是直流则采用步步为营的思想前进；如果是二分流则通过判断结束点进行前进。
- 二分流结束点判断：单流转多流、多流转单流的时候认为是结束点。

高级话题

由于 Channel 通过判断结束点控制二分流前进，因此设计上要考虑当前模块是否单流、多流，否则设计二分流的时候会出错。

例如设计了 Filter，作为过滤器，如果 Filter 本身是多流，但是后继模块是单流，则设计错误；或者 Filter 是单流后继模块是单流也出错。

Pool 核心思想

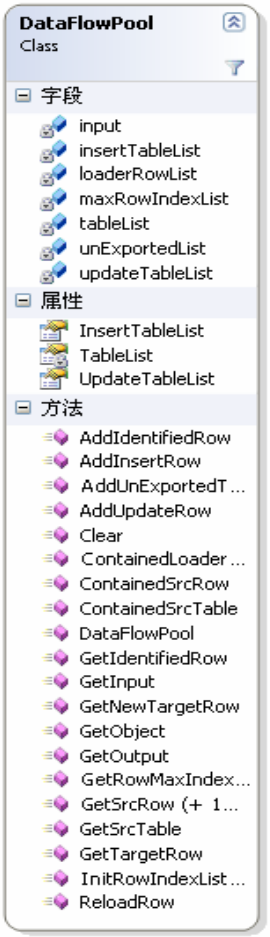
关键词

多流、单流、装载层、数据层

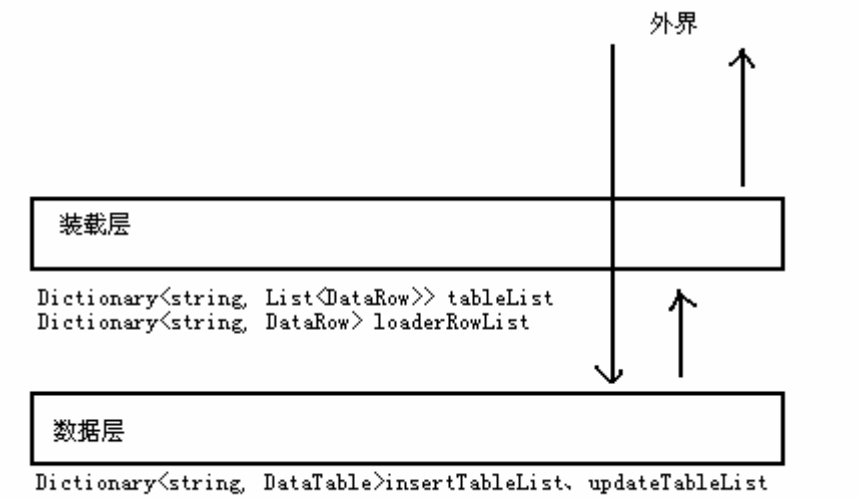
一句话描述

为模块提供正确的数据源、保存模块生成的数据

UML



数据结构



使用双层数据结构，包括

- 装载层：`Dictionary<string, List<DataRow>>` tableList、`Dictionary<string, DataRow>` loaderRowList
- 数据层：`Dictionary<string, DataTable>`insertTableList、updateTableList

外界输入的数据首先保存在数据层；然后被装载到装载层；最后调用的时候从装载层调用
数据层就像仓库，不实际参与数据流运算；但是需要输出，不允许有重复冗余数据；
装载层是数据结构中的核心，里面可能有重复数据，空数据，不输出。

由于数据源每张表可能有多条数据，也有单条数据，因此流运算过程中需要对数据进行标示，保证数据一致性这里是用了计数器。

- 计数器：`Dictionary<string, int>` maxRowIndexList

计数器记录每个表对应的容量，作为逻辑的参考。

装载层 loaderRowList

用于 LoaderModule 的缓存操作。如果之前做了 Loader，则此时再次 Load 仅仅把上次的数据装载！即从 loaderRowList 装载到 tableList

【问题】为什么出现相同的装载？

逻辑分析：在数据流中对同一张表在不同点进行了多次装载，存在顺序的差异。例如库存操作。

装载层 tableList

- 装载层，当前操作的对象通过 CurRowIndex 进行表示

流标识

- 整个数据流层使用 maxRowIndexList 进行全局流记录。
- 每次流操作都会有个 CurRowIndex 作为标识，代表当前所处理的流的编号。
- 流策略：对于只有一条数据的表，其数据参数所有涉及到的流操作（包括单流、多流）；对于多条数据的表，根据对应的流标示进行操作。
- 流策略实例：订单主从表，主表只有 1 条数据，子表有 n 条数据，现在要填充子表的外键为主表的主键，这样即主表一条数据参与了子表多条数据的流运算。

装载层装载逻辑

取得 Src 数据

- 如果数据来自 Input，则直接获取
- 如果来自 tableList，如果 tableList 里面只有 1 条数据，则默认为单流，始终默认返回
- 如果来自 TableList，并且里面保存了多条数据，则根据标示号返回数据
- 如果不存在则返回 null

取得 Target 数据

- 如果数据来自 Input，则直接获取
- 如果来自 tableList，如果 tableList 里面只有 1 条数据，则默认为单流，始终默认返回
- 如果来自 TableList，并且里面保存了多条数据，则根据标示号返回数据
- 如果不存在则新建一个

添加数据

- 只添加 Detach 的数据，因为他们是本层产生的。
- 需要添加到数据层和装载层
- 数据层：添加包括添加 Insert 类型的、添加 Update 类型的。Exchanger 只添加 Insert 的，Loader 两种都有
- 装载层：如果装载对象已经被装载过（可能之间做过了处理），则做最近替换原则处理

高级话题

- 数据池里面每张表只能存在一次，即不允许出现重复的表，因此数据流所有的操作如果针对相同的表名，那么所有的数据来自相同的地方。设计数据流的时候需要考虑这点，防止流出错。
- 数据池使用数据标示符对多流进行处理，因此在同一条流里面，如果有二义性操作要注意。
- 例如模块 A 对订单进行字段修改，然后模块 B 对原有订单进行克隆生成一张新订单，这种操作实际上是无法实现的，称为（自映射）。因为对同一张表进行了 2 种不同逻辑的操作。
- 又如，在同一条标识符的流里面，先对商品 A 进行减库存操作，然后对商品 B 进行增库存操作，这也是无法实现的，在同一条流中间对一张表不同标识符的字段进行处理。因为前者已经在当前流在装载层加载了库存数据，后者变成修改此数据。
- 上面情况唯一的例外是模块之间使用 Loader 进行分割，这样 Loader 会在装载层加载合适的数据。

Advanced Topics

可重用模块设计

- 在模块中使用对象表达式可以提高模块的重用性。
- 例如多个模块使用了 OJBECT.[]，然后在不同的 Channel 里面加载这些模块，最后在开始的时候加载一个 Router 具体声明模块中的对象表达式数据源，就能够实现重用。

设计中的流设计问题

- 由于 Pool 目前设计的局限，涉及模块之间逻辑关系的时候要小心。
- 如果连续几个模块对同一张表进行不同逻辑的操作，则无法实现，例如模块 A 对商品 A 库存增加；模块 B 对商品 B 库存减少，这样是无法实现的。
- 解决上面的问题方法是在模块 A\B 之间使用 Loader 进行加载

设计中的二分流设计问题

- 由于 ChannelAssembler 目前的设计局限，对于二分流设计要小心。
- 如果二分流为多流，则分流之后的模块必须也是多流，否则分流就没有意义，导致出错。

映射中的表达式

- 表达式中提供了一些函数运算，例如 MAX[].[]等。目前的版本中仅能够对单张表的单个字段进行函数运算，不能对四则混合运算结果求函数值。

字符串映射与四则混合运算映射

- 如果需要计算具体的数字，使用四则混合运算映射；否则通常使用字符串映射
- 这个区别在写配置文件的时候很容易搞错，即把四则运算的映射写在了字符串映射，导致赋值的时候字段类型报错。

Error Code List

一句话概述

数据流层发生错误的时候，会有错误代码提示，通过下面的查询可以找到错误的位置。

Configuration::01XXX 代码说明

Channel:
01101:配置主标识Identifier为空

01103:Channel缺少配置Pipeline
01104:DirectPipeLine缺少Identifier
01105:BooleanPipelineConfiguration缺少Identifier

Exchanger:
01201:配置主标识Identifier为空
01202:缺少处理的表名
01203:缺少关系配置
01206:NumericExprRelation重复
01207:StringExprRelation重复

Filter:
01301:配置主标识Identifier为空
01302:FilterConfiguration缺少Expression
01303:FilterConfiguration有多余的表达式

Loader:
01401:LoaderConfiguration缺少indentifier
01402:LoaderConfiguration缺少sql
01403:缺少ExprRelations
01404:numericExprRelations重复
01405:stringExprRelations重复
01406:LoaderExprRelation缺少name
01407:LoaderExprRelation缺少expression

Router:
01501: RouterConfiguration缺少Identifier
01502: RouterConfiguration缺少Mappings
01503: ValueMappingConfiguration缺少identifier

01504: ValueMappingConfiguration缺少value
01505: ObjectMappingConfiguration缺少identifier
01506: ObjectMappingConfiguration缺少tableName
01507: ObjectMappingConfiguration缺少columnName

NumericExprRelation:
01601: 缺少columnName
01602: 缺少expression

StringExprRelation:
01701: 缺少columnName
01702: 缺少expression

Module::02XXX 代码说明

ADataFlowModule:
02001:GetNumericResult计算错误，表达式错误（值非数字、值不全、表达式错误）
02002:GetStringResult计算错误
02003:GetNumericBoolResult
02004:GetStringBoolResult

02005:NumericContentReplacement/SumPair
02006:NumericContentReplacement/MaxPair
02007:NumericContentReplacement/MinPair
02008:NumericContentReplacement/TablePair
02009:NumericContentReplacement/ObjPair

02010:StringContentReplacement/SumPair
02011:StringContentReplacement/MaxPair
02012:StringContentReplacement/MinPair
02013:StringContentReplacement/TablePair
02014:StringContentReplacement/ObjPair

02015:GetSumExprResult/表不存在
02016:GetSumExprResult/表数据非数字
02017:GetMaxExprResult/表不存在
02018:GetMaxExprResult/表数据非数字
02019:GetMinExprResult/表不存在
02020:GetMinExprResult/表数据非数字
02019:GetMinExprResult/表不存在
02020:GetMinExprResult/表数据非数

Pool::03XXX 代码说明

DataFlowPool:

03101:初始化流量记录的时候出错.表重复

03102:运算取得数据源的时候,流量超出INPUT数据源范围

03103:运算取得数据对象的时候,流量超出INPUT数据源范围

03104:运算取得数据对象的时候,数据对象在产生流中不存在

03105:Load对数据流进行重载的时候,表对象为游离态,因此在数据源不存在

Utility::04XXX 代码说明

BoolExpression:

04101:GetBoolResult表达式计算错误

NumericBoolExpression:

04201:GetCompareResult的bool表达式出错!

04202:GetCompareResult的表达式值非bool型

StringBoolExpression:

04301:GetCompareResult 的 bool 表达式出错!

In Future

- DataFlowPool 和 ChannelAssembler 是设计的瓶颈。下一个版本将重构这两部分，实现完全的数据流。
- 增强 xml 描述的规范性
- 重构 Mapping 机制，增加新的功能，类似 Excel 里面的数据处理。