iUAP DI中元数据转换的行处理和SOAP解析

作者：魏剑龙

目录

[1. AE-DI转换插件开发简介 2](#_Toc427333145)

[1.1. 转换步骤的四个组件 2](#_Toc427333146)

[1.1.1. 元数据类 2](#_Toc427333147)

[1.1.2. 对话框类 3](#_Toc427333148)

[1.1.3. 步骤类 3](#_Toc427333149)

[1.1.4. 数据类 4](#_Toc427333150)

[2. AE-DI元数据输入转换中行的处理 4](#_Toc427333151)

[2.1. 代码走查 4](#_Toc427333152)

[3. SOAP简介 8](#_Toc427333153)

[3.1. 为什么需要SOAP，什么是SOAP 8](#_Toc427333154)

[4. SOAP消息的解析 9](#_Toc427333155)

# AE-DI转换插件开发简介

## 转换步骤的四个组件

这里有四个类构成了步骤/节点，每一个类都有其特定的目的及所扮演的角色。

\*\*\*\*\*\*Step:  步骤类实现了StepInteface接口，在转换运行时，它的实例将是数据实际处理的位置。每一个执行线程都表示一个此类的实例。

\*\*\*\*\*\*StepData:  数据类用来存储数据，当插件执行时，对于每个执行的线程都是唯一的。执行时里面存储的东西主要包括数据库连接、文件句柄、缓存等等其他东西。

\*\*\*\*\*\*StepMeta: 元数据类实现了StepMetaInterface接口。它的职责是保存和序列化特定步骤实例的配置，在我们这个例子中，它负责保存用户设置的步骤名称和输出字段的名称。

\*\*\*\*\*\*StepDialog：对话框类实现了该步骤与用户交互的界面，它显示一对话框，通过对话框用户可以自己的喜好设定步骤的操作。对话框类与元数据类关系非常紧密，因为元数据类可以追踪用户的设置。

### 元数据类

下面显示了元数据的几个关键的方法，注意元数据类里面用私有成员变量outputField 存储了下一个步骤的输出字段。

 // keep track of the step settings  
public String getOutputField()  
public void setOutputField(…)  
public void setDefault()

// serialize the step settings to and from xml  
public String getXML()  
public void loadXML(…)

// serialize the step settings to and from a kettle repository  
public void readRep(…)  
public void saveRep(…)

// provide information about how the step affects the field structure of processed rows  
public void getFields(…)

// perform extended validation checks for the step  
public void check(…)

// provide instances of the step， data and dialog classes to Kettle  
public StepInterface getStep(…)  
public StepDataInterface getStepData()  
public StepDialogInterface getDialog(…)

TemplateStepMeta元数据类其实还有很多方面，不过大多被他的父类BaseStepMeta给默认实现了，这些默认的实现足以使我们的元数据类工作良好。

### 对话框类

TemeplateStepDialog为步骤实现了对话框的设置，使用的是eclipse的swt框架，如果要开发比较复杂的对话框，你还必须熟悉大部分swt代码。在开发过程中，一个对话框对象拥有一个元数据对象，它记录了应该从哪里读取配置？应该把设置好的配置保存在哪里？ 它仅仅设置了输出字段的名称在我们这个模板步骤里面。一个继承自BaseStepDialog特定的对话框类必须提供open(…)方法，这个方法必须返 回这个步骤的名称（发生改变时）或NULL（对话框被取消时）。

### 步骤类

步骤类是实际的处理和转换工作的地方。因为大部分样本代码已经由父类BaseStep提供了，大多数插件仅仅关注下面几个特定的方法就行。

 // initialization and teardown  
public boolean init(…)  
public void dispose(。。)

// processing rows  
public void run()  
public boolean processRow(。。)

Init()方法在转换执行前被调用，转换必须在所有步骤初始化成功时才真正执行。我们这个模板步骤没有做任何事情，这里仅仅是拿出来让大家了解了解。

dispose()方法是在步骤执行完之后执行（非转换执行完哈），它完成资源的关闭，像文件句柄、缓存等等。

run()方法在实际处理记录集的时候调用。里面其实是个调用processRow()方法处理记录的小循环，当此步骤再没有数据处理或转换被停止时退出循环。

processRow()方法在处理单条记录的时候被调用。这个方法通常通过调用getRow()来获取需要处理的单条记录。 这个方法如果有需要将会被阻塞，例如当此步骤希望放慢脚步处理数据时。processRow()随后的流程将执行转换工作并调用putRow()方法将处 理过的记录放到它的下游步骤。

### 数据类

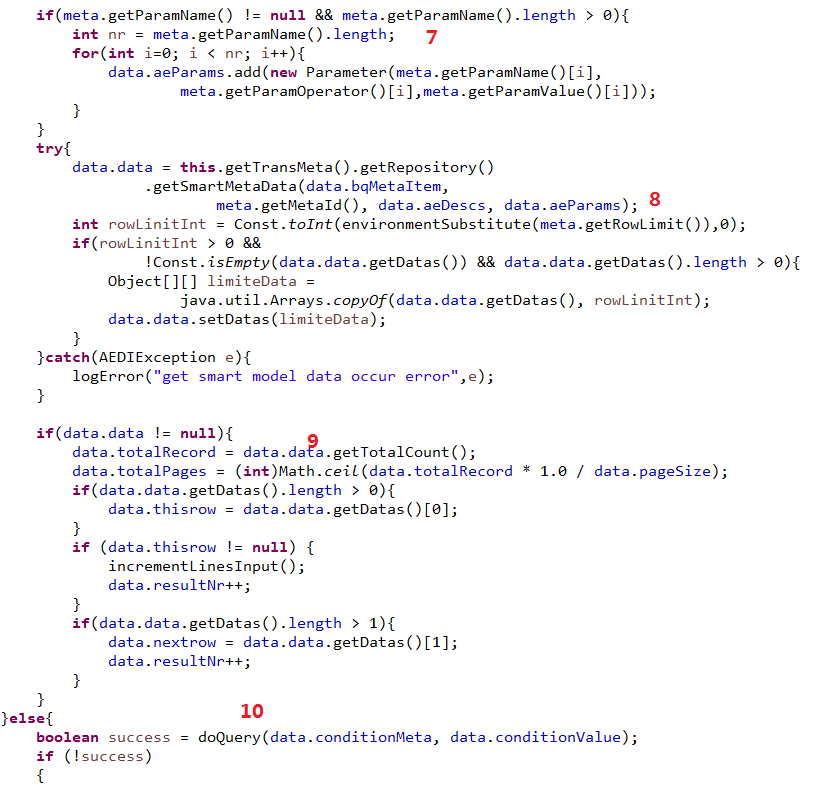
大多数步骤都需要临时的缓冲或者临时的数据。数据类就是这些数据合适的存放位置。每一个执行线程将得到其拥有的数据类实例，所以它能在独立的空间里面运行。TemplateStepData继承自BaseStepData，作为一个经验法则，不要将non-constant字段放置 BaseStepData类里面，如果你必须，请将它最好放置TemplateStepData数据类里面。

# AE-DI元数据输入转换中行的处理

## 代码走查



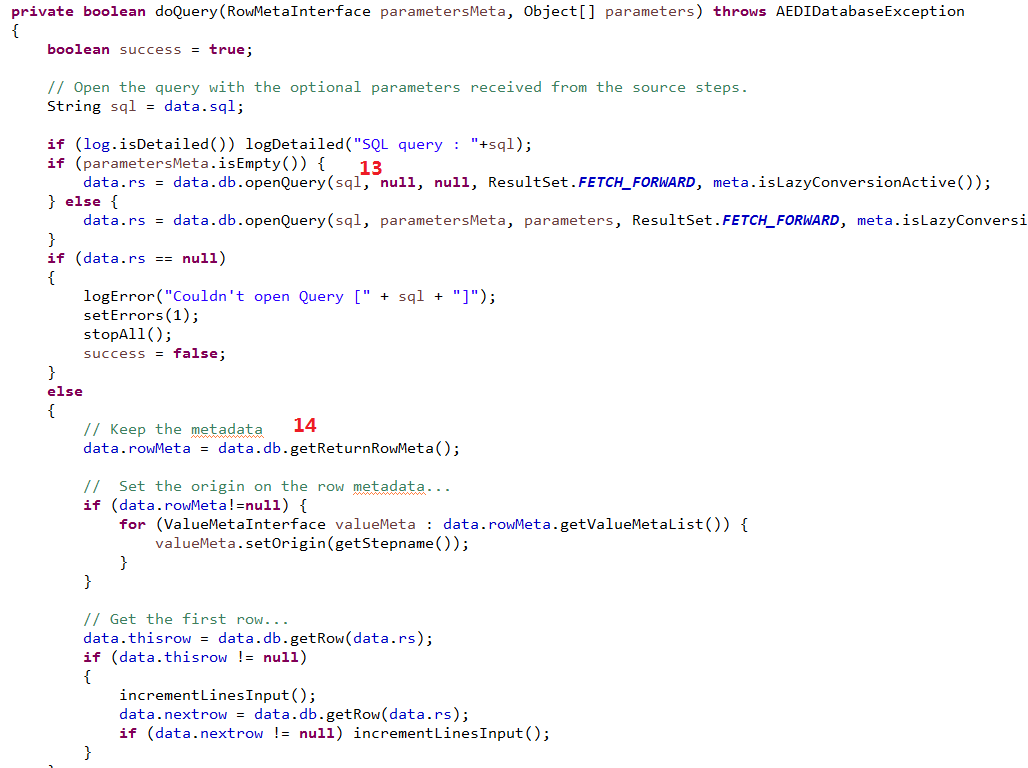
1. 判断是否是语义模型，是的话，然后设置分页描述器。
2. 添加选择的列描述器
3. 添加条件过滤描述器
4. 为了支持Between过滤器条件，在FilterItem中添加FILTER\_OPERATOR\_BETTWEEN常量，并且获取前台设置的两个value，设置到filter中，然后添加到过滤器描述器
5. 添加其他出bettween外，只需要一个值的过滤条件
6. 添加排序描述器



1. 判断是否有参数，有的话，添加语义模型参数
2. 为了在前台的记录限制其作用，在这里进行了限制，因为在meta中有限制条数的属性，就容易处理了
3. 根据总条数，进行分页
4. 如果不是语义模型，直接调用数据库查询



1. 是语义模型元数据的话，按照分页大小，如果当前结果集的数量小于单页的尺寸，就没有下一页，设置为null，否则，读取下一页放到data.nextrow中
2. 将一行结果推送到下一个步骤中



1. 根据前面设置的过滤条件，选择的列，和排序等描述器，对数据库进行查询，并返回结果
2. 获取行的元数据信息



1. 初始化的过程中，如果是语义模型，就获取需要的域field
2. 如果是数据仓库元数据，就通过repository的getAEMetaExecSql，调用元数据MetaWebService，发起SOAP请求，返回UAPDatabaseMeta
3. 将返回的值设置到data.conditionMeta中，这就涉及到构造输出列，主要有两种方法，如下。

//构造一个新的输出列。方法一  
RowMetaInterface rowMeta = new RowMeta();  
Object[] rowData = new Object[1];  
int valtype = ValueMeta.getType("String");  
ValueMetaInterface valueMeta = new ValueMeta( “FieldName1”, valtype); valueMeta .setLength(-1);  
rowMeta.addValueMeta(valueMeta);  
//构造一个新的输出列。方法二  
RowMetaInterface outputRowMeta = getInputRowMeta().clone();  
int valtype = ValueMeta.getType("String");  
ValueMetaInterface valueMeta = new ValueMeta( “FiledName1”, valtype);  
valueMeta.setLength(-1);  
outputRowMeta.addValueMeta(valueMeta);

# SOAP简介

## 为什么需要SOAP，什么是SOAP

为了使异构系统能够互连互通，打破壁垒，实现"基于WEB无缝集成"的目标，Web Service孕育而生。SOAP（Simple Object Access Protocol ）简单对象访问协议作为Web Service的传输协议，是一个基于XML的协议，它包括四个部分：SOAP封装(envelop)，封装定义了一个描述消息中的内容是什么，是谁发送的，谁应当接受并处理它以及如何处理它们的框架；SOAP编码规则（encoding rules），用于表示应用程序需要使用的数据类型的实例； SOAP RPC表示(RPC representation)，表示远程过程调用和应答的协定；SOAP绑定（binding），使用底层协议交换信息。SOAP的两个主要设计目标是简单性和可扩展性。这就意味着有一些传统消息系统或分布式对象系统中的某些性质将不是SOAP规范的一部分。下面分别是reqeust和response的SOAP消息，它包括header和body。

* Response:

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

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Header>

<ns0:Urc xmlns:ns0="http://ws.uap.nc/lang">

<ns0:datasource>design</ns0:datasource>

<ns0:userCode>#UAP#</ns0:userCode>

<ns0:langCode>simpchn</ns0:langCode>

</ns0:Urc></soap:Header>

<soap:Body>

<ns1:loadFieldsResponse

xmlns:ns1="http://meta.ae.pubitf.uap/IMetaWebService">

<ns1:return xmlns:ns0="http://webservice.meta.ae.pub.uap/MetaElement"

xmlns:ns2="http://webservice.meta.ae.pub.uap/MetaField">

<ns0:metaKind>3</ns0:metaKind>

<ns0:metaDisplayName>FIRST\_NAME</ns0:metaDisplayName>

<ns0:metaId>field^FIRST\_NAME</ns0:metaId>

<ns0:metaDescription/>

<ns0:businessId>FIRST\_NAME</ns0:businessId>

<ns0:metaType>field</ns0:metaType>

<ns0:metaCode>FIRST\_NAME</ns0:metaCode>

<ns2:scale>0</ns2:scale>

<ns2:dataType>12</ns2:dataType>

<ns2:precision>20</ns2:precision>

<ns2:primaryKey>false</ns2:primaryKey>

<ns2:fieldIndex>0</ns2:fieldIndex>

<ns2:nullable>false</ns2:nullable>

</ns1:return></ns1:loadFieldsResponse>

</soap:Body>

</soap:Envelope>

* Request：

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

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<loadFields

xmlns="http://meta.ae.pubitf.uap/IMetaWebService">

<string>SmartModel^1001ZZ1000000000GZVW</string>

</loadFields>

</soapenv:Body>

</soapenv:Envelope>

# SOAP消息的解析

AE-DI在元数据输入，输出，查询的转换中，要调用元数据的Web Service，进行SOAP消息的解析，就是要处理XML。对此采用DOM解析。DOM是用与平台和语言无关的方式表示XML文档的官方W3C标准。DOM是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找 特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而DOM被认为是基于树或基于对象的。DOM解析器把XML文档转化为一个包含其内容的树，并可以对树进行遍历。DOM是拉模型，在遍历文档时，会把感兴趣的部分从读取器中拉出，不需要引发事件，允许我们选择性地处理节点。这大大提高了灵活性，以及整体效率。DOM的基本对象有5个：Document，Node，NodeList，Element和Attr。

* **Document对象**

代表了整个XML的文档，所有其它的Node，都以一定的顺序包含在Document对象之内，排列成一个树形的结构，程序员可以通过遍历这颗树来得到XML文档的所有的内容，这也是对XML文档操作的起点。我们总是先通过解析XML源文件而得到一个Document对象，然后再来执行后续的操作。

* **Node对象**

是DOM结构中最为基本的对象，代表了文档树中的一个抽象的节点。在实际使用的时候，很少会真正的用到Node这个对象，而是用到诸如Element、Attr、Text等Node对象的子对象来操作文档。Node对象为这些对象提供了一个抽象的、公共的根。虽然在Node对象中定义了对其子节点进行存取的方法，但是有一些Node子对象，比如Text对象，它并不存在子节点，这一点是要注意的。

* **NodeList对象**

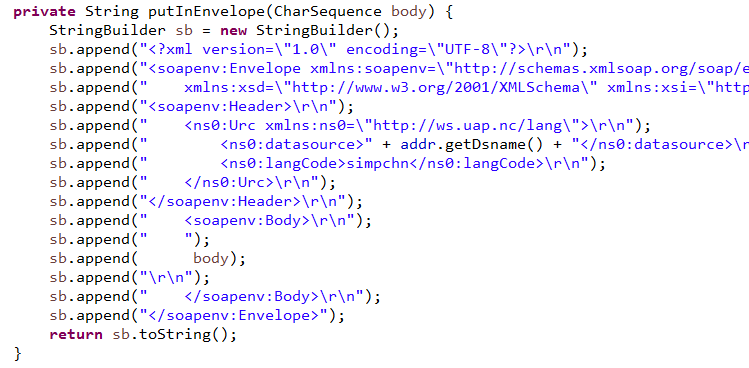
顾名思义，就是代表了一个包含了一个或者多个Node的列表。可以简单的把它看成一个Node的数组，我们可以通过方法来获得列表中的元素：

getLength()：返回列表的长度。

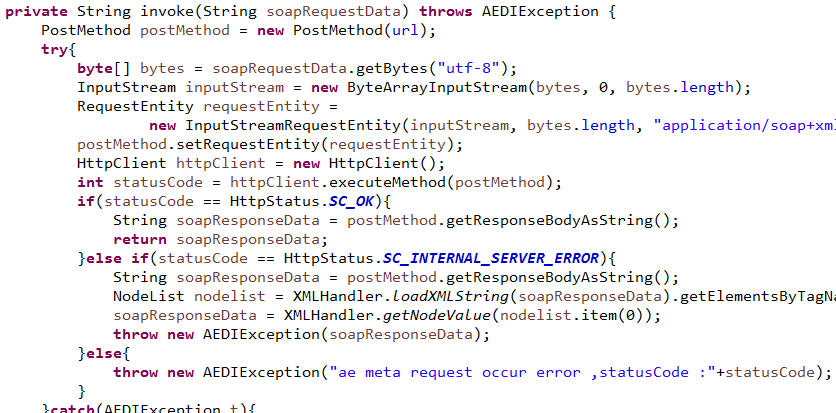
item(int)：返回指定位置的Node对象。

* **Element对象**

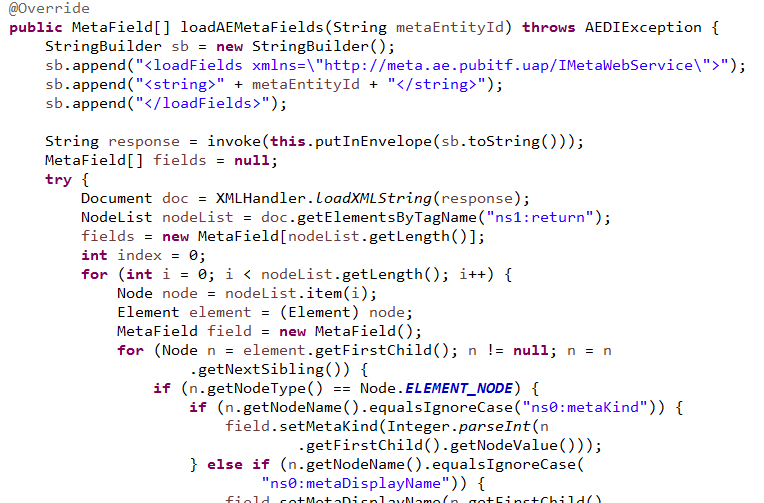
代表的是XML文档中的标签元素，继承于Node，亦是Node的最主要的子对象。在标签中可以包含有属性，因而Element对象中有存取其属性的方法，而任何Node中定义的方法，也可以用在Element对象上面。



该方法主要是封装SOAP请求，将头信息的数据源名字添加到header中，以<ns0:datasource>标签存在，该方法避免了重复的SOAP请求繁琐的字符串拼接，代码得到了重用。



该方法是使用HttpClient发起SOAP请求



该方法使用StringBuilder将要请求的方法信息，通过调用封装方法putInEnvelope，放回请求SOAP消息，调用invoke，通过HttpClient发起请求，返回SOAP响应消息，并使用jdk自带的DOM解析。其中，XMLHandler工具可以解析XML字符串，返回Document对象，自后定位到制定的标签，有点XPath的作用，然后解析。值得一提的是，DOM把空白和文本值当作了Node，所以解析到达以个标签的时候，要想获取它的值，还要调用getFirstChild方法，