**DataCleaner中文文档**

目录

[第一章 DataCleaner简介 1](#_Toc8825)

[1. 背景和概念 1](#_Toc6680)

[1.1 什么是数据质量（DQ）？ 1](#_Toc31512)

[1.2 什么是数据分析？ 1](#_Toc12083)

[1.3 什么是数据争用？ 2](#_Toc16764)

[1.4 什么是数据存储？ 3](#_Toc8960)

[1.5 什么是数据监控？ 3](#_Toc2251)

[1.6 什么是主数据管理（MDM）？ 4](#_Toc32280)

[2. DataCleaner桌面版入门 5](#_Toc6421)

[2.1 安装客户端 5](#_Toc19107)

[2.2 连接数据库(或存储文件) 6](#_Toc5334)

[2.3 向job中添加组件 8](#_Toc9817)

[2.4 连接组件 9](#_Toc15207)

[2.5 执行job 12](#_Toc4521)

[2.6 保存和打开job 12](#_Toc14352)

[2.7 job示例模板 13](#_Toc19717)

[2.8 将清洗后的数据写入文件 15](#_Toc10909)

[第二章 组件分析 16](#_Toc23968)

[3. Transform 16](#_Toc15778)

[3.1 JavaScript 转换器 16](#_Toc23852)

[3.2 调用子分析job 18](#_Toc32720)

[3.3 Apply classifier & Apply regression 19](#_Toc5196)

[3.4 Equals 过滤器 20](#_Toc8461)

[3.5 Max rows 过滤器 20](#_Toc7643)

[3.6 Not null 过滤器 20](#_Toc31019)

[3.7 Union转换器 21](#_Toc318)

[4. Improve 22](#_Toc11631)

[4.1 近义词查找 22](#_Toc25678)

[4.2 表查询 23](#_Toc13133)

[5. Analyze 24](#_Toc14760)

[5.1 布尔分析器 24](#_Toc8706)

[5.2 完整性分析器 25](#_Toc13922)

[5.3 字符集分布 26](#_Toc4590)

[5.4 日期间隔分析器 26](#_Toc29296)

[5.5 日期/时间分析器 26](#_Toc32741)

[5.6 数字分析器 26](#_Toc29001)

[5.7 模式工具 27](#_Toc7110)

[5.8 参考数据匹配器 29](#_Toc31875)

[5.9 参照完整性 29](#_Toc25887)

[5.10 字符串分析器 30](#_Toc18561)

[5.11 主键检查 30](#_Toc19882)

[5.12 值分布 30](#_Toc23404)

[5.13 值匹配 31](#_Toc6369)

[5.14 工作日分布 31](#_Toc12551)

[5.15 机器学习 32](#_Toc10489)

[6. Write 33](#_Toc10789)

[6.1 创建CSV文件 33](#_Toc11531)

[6.2 创建Excel表格 33](#_Toc3272)

[6.3 创建临时表 33](#_Toc310)

[6.4 插入表格 34](#_Toc16941)

[6.5 更新表格 35](#_Toc22770)

[第三章 数据参考 36](#_Toc10840)

[7. Dictionaries 36](#_Toc19702)

[7.1字典 36](#_Toc22746)

[8. Synonyms (aka. Synonym catalogs) 37](#_Toc29024)

[8.1 文本文件同义词目录 37](#_Toc27878)

[8.2 数据存储同义词目录 37](#_Toc32253)

[9. String patterns 37](#_Toc1777)

[9.1字符串模式 37](#_Toc8669)

[第四章 相关配置 38](#_Toc12400)

[10. Configuration file 38](#_Toc9380)

[10.1 XML模式 38](#_Toc28276)

[10.2 数据存储 38](#_Toc19245)

[10.3 参考数据 45](#_Toc15923)

[10.4 任务执行器 47](#_Toc25214)

[10.5 存储提供商 48](#_Toc29371)

[11. Analysis job files 49](#_Toc11034)

[11.1 xml模式 49](#_Toc25958)

[11.2 Source section 50](#_Toc28730)

[12. Logging 51](#_Toc14022)

[12.1 日志配置文件 51](#_Toc12660)

[12.2 默认日志配置 52](#_Toc3111)

[12.3 修改日志级别 53](#_Toc15051)

[12.4 选择性日志输出 53](#_Toc21970)

[13. Database drivers 54](#_Toc9813)

[13.1 在DataCleaner桌面中安装数据库驱动程序 54](#_Toc19638)

[第五章 调用 DataCleaner jobs 56](#_Toc23495)

[14.  Command-line interface 56](#_Toc2602)

[14.1 可执行程序 56](#_Toc1449)

[14.2 使用场景 56](#_Toc29409)

[14.3 执行分析job 57](#_Toc28941)

[14.4 列出数据存储内容和可用组件 58](#_Toc13888)

[14.5 参数化job 60](#_Toc13294)

[14.6 动态重写配置元素 61](#_Toc28095)

[15. Apache Hadoop and Spark interface 62](#_Toc6059)

[15.1 Hadoop部署概述 62](#_Toc1020)

[15.2 设置Spark和DataCleaner环境 63](#_Toc28729)

[15.3 使用Spark启动DataCleaner作业 65](#_Toc2465)

[15.4 在DataCleaner桌面端使用Hadoop 66](#_Toc3583)

[15.5 Hadoop接口的限制 68](#_Toc3668)

[第六章 第三方集成 69](#_Toc10106)

[16. Pentaho integration 69](#_Toc1606)

[第七章 开发者指南 71](#_Toc16668)

[17. Architecture 71](#_Toc18717)

[18. Executing jobs through code 73](#_Toc3328)

[19.  Developer resources 82](#_Toc23886)

[20. Extension packaging 90](#_Toc18798)

[21. Embedding DataCleaner 91](#_Toc6945)

# **DataCleaner简介**

## 背景和概念

### 什么是数据质量（DQ）？

数据质量（Data Quality简称DQ）是一个概念和一个业务术语，涵盖了用于特定目的的数据的质量。DQ术语通常用于商业决策中使用的数据质量，但也可能指研究、活动、流程等中使用的数据质量。

处理数据质量通常因项目而异，正如数据质量的问题有很大的不同。数据质量问题的例子包括：

* 数据的完整性
* 数据的正确性
* 数据冗余
* 数据统一/标准化

数据质量分析（DQA）是检查特定过程或组织的数据质量的（人工）过程。DQA包括技术和非技术要素。例如，要做好DQA，您可能需要与用户、业务人员、合作伙伴组织以及客户交谈。这需要评估DQA的目标应该是什么。

从技术角度来看，DQA的主要任务是数据分析活动，它将帮助您发现和度量数据中的当前状态。

### 什么是数据分析？

数据分析（Data profiling）是调查数据存储以创建其“profile”的活动。有了数据存储的概要文件，您就可以更好地实际使用和改进它。

分析的方式通常取决于您是否已经对数据的质量有了一些想法，或者您是否对手头的数据存储熟不熟练。无论哪种方式，我们都建议采用探索性的方法，因为尽管您认为只需要查找一定数量的问题，但根据我们的经验（以及DataCleaner的许多特性背后的推理），检查数据中您认为正确的项目细节同样重要！一般来说，在你的分析中加入更多的数据可能会有意想不到的结果，更节省你的时间！

### 什么是数据争用？

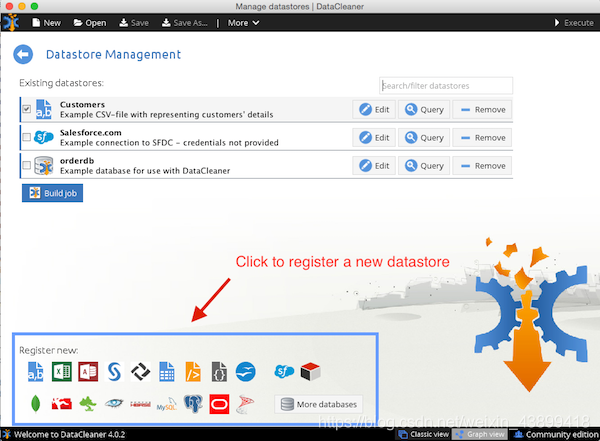
从维基百科，我们得到了一个很好的“数据争论”的介绍性解释：

数据挖掘或数据争用是一种松散地将数据从一种“原始”形式手动转换或映射到另一种格式的过程，这种格式允许在半自动化工具的帮助下更方便地使用数据。这可能包括进一步挖掘、数据可视化、数据聚合、训练统计模型以及许多其他潜在用途。

可以看出，为了分析数据，对数据进行更改通常是很重要的。这就是为什么DataCleaner捆绑了许多转换和改进功能，这些功能不仅允许用户分析数据，还允许用户将数据重塑为合适的形式。

### 什么是数据存储？

数据存储(datastore)是存储数据的地方。企业数据通常存在于关系数据库中，但这一规则有许多例外。为了理解不同的数据源，例如数据库、电子表格、XML文件甚至标准业务应用程序，我们使用了数据存储这一总称。DataCleaner能够从非常广泛的数据存储中检索数据。此外，DataCleaner还可以更新大多数数据存储的数据。可以在UI中或通过配置文件创建数据存储。您可以从任何类型的源创建数据存储，例如：CSV、Excel、Oracle数据库、MySQL等。



**复合数据存储**

复合数据存储(Composite datastore)包含多个数据存储。复合数据存储的主要优点是，它允许您在同一个job中分析和处理来自多个源的数据。

### 什么是数据监控？

我们认为数据分析是一种理想的探索活动。数据监控通常不是！您在分析时经常执行的度量需要不断地检查，以便随着时间的推移强制执行您的改进。这就是数据监控的典型意义。

数据监控解决方案由不同情况而定。您可以设置自己每天晚上运行的大量计划作业。您可以围绕它构建警报，当某个特定度量超出允许的阈值时向您发送电子邮件，或者在某些情况下，您可以尝试通过应用在输入时验证数据的首次权利（FTR）原则来完全排除该问题。例如，在数据注册表单时。

### 什么是主数据管理（MDM）？

主数据管理（MDM）是一个非常宽泛的术语，它以多种方式具体化。对于本文档的范围来说，它更多的是作为数据质量的上下文，而不是我们实际以DataCleaner本身为目标的活动。

MDM的总体目标是管理组织的重要数据。“主数据”指的是“单一版本的事实”，即不是某一特定系统的数据，而是某一公司的所有客户数据或产品数据。通常这些数据分散在多个数据存储中，因此MDM的一个重要部分是将数据统一到单个模型中的过程。

显然，MDM中要处理的另一个非常重要的问题是数据质量。如果您只是从组织中的所有系统收集“所有客户数据”，您很可能会看到许多数据质量问题。会有很多重复条目，填充客户数据的方式会有差异，会有不同的标识符，甚至定义“什么是客户？”的粒度级别也会不同。在MDM的上下文中，DataCleaner可以充当引擎，将多个数据存储中的数据清理、转换和统一到主数据的单个视图中。

## **DataCleaner桌面版入门**

### **安装客户端**

以下是安装DataCleaner的系统要求：

* 电脑一台
* jdk1.7及以上
* 专业版的DataCleaner软件许可证文件。如果您要求免费试用或在线购买DataCleaner，此文件将发送到您的电子邮件地址。

使用安装程序启动安装过程。安装程序是一个可执行的JAR文件，只要双击它，它在大多数系统上都是可执行的。就是启动一个jar文件。

如果双击安装程序时安装程序未启动，请打开命令提示符并输入：  
java -jar DataCleaner-[edition]-[version]-install.jar

**故障诊断**  
安装过程出现问题请检查以下几点：

* 在Windows系统上，如果您没有计算机的管理权限，我们建议您在自定义目录中安装DataCleaner，而不是在C盘的系统文件中。
* 在某些Windows系统上，您可能会收到警告“There is no script engine for file extension ‘.js’ ”。当.js文件（JavaScript）与编辑器而不是Windows内置脚本引擎相关联时，就会发生这种情况。要解决此问题，请参阅以下帮助链接：

answers.microsoft.com

https://www.winhelponline.com/

* 如果在查找或选择软件许可证文件时遇到问题，可以跳过安装程序中的步骤，方法是将许可证文件手动复制到以下文件夹：“~/.datacleaner”(其中 ~ 这个是用户的主文件夹）。请注意，在Windows计算机上，Windows资源管理器禁止创建以点（.）开头的目录，但可以使用命令：mkdir .datacleaner
* 如果堆内存不足，可以调整“DataCleaner.l4j”文件中分配的内存量，如果从控制台运行，也可以调整“DataCleaner console.l4j”文件中分配的内存量。为了增加Java堆内存，可以使用以下参数：

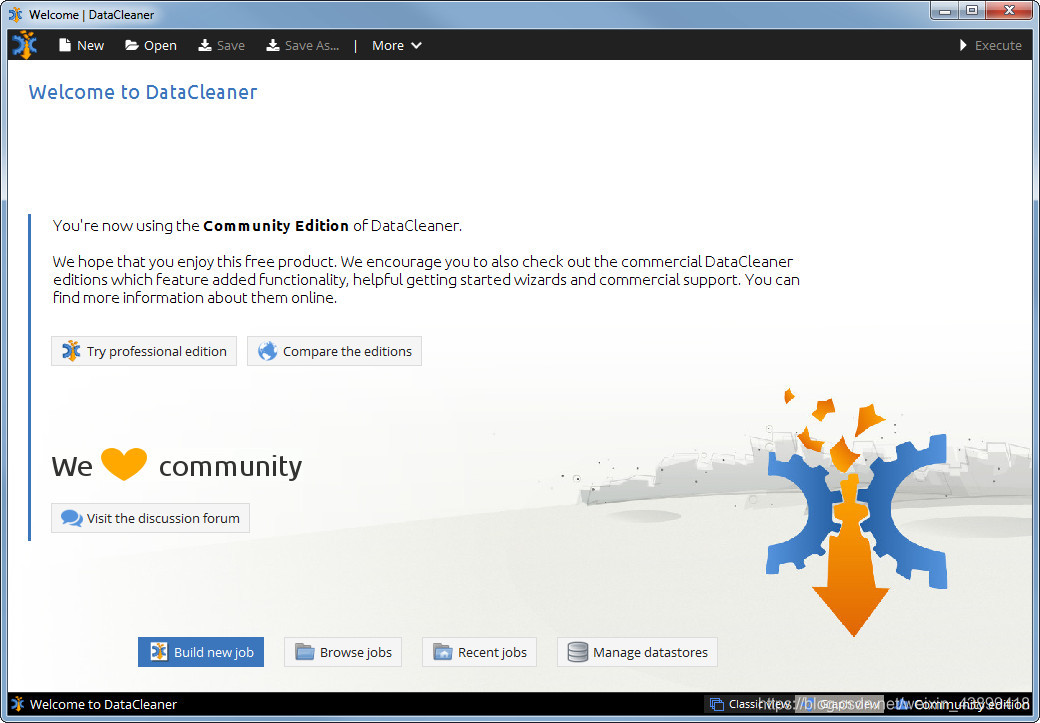
-Xms-设置最小堆大小

-Xmx-设置最大堆大小

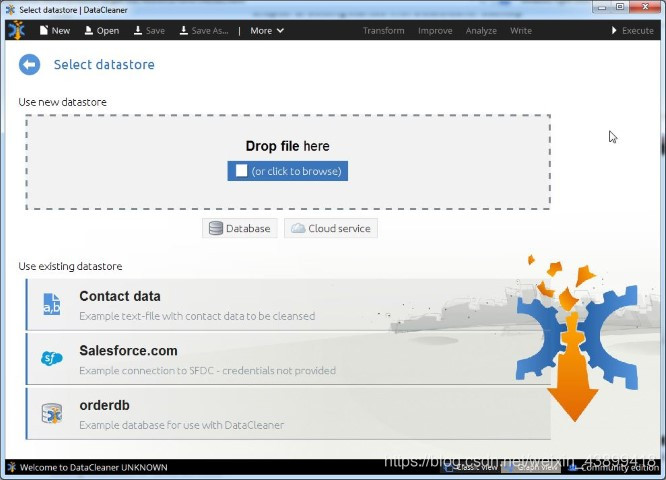
例如，-Xmx4g 指将堆的最大大小设置为4 GB。

### **连接数据库(或存储文件)**

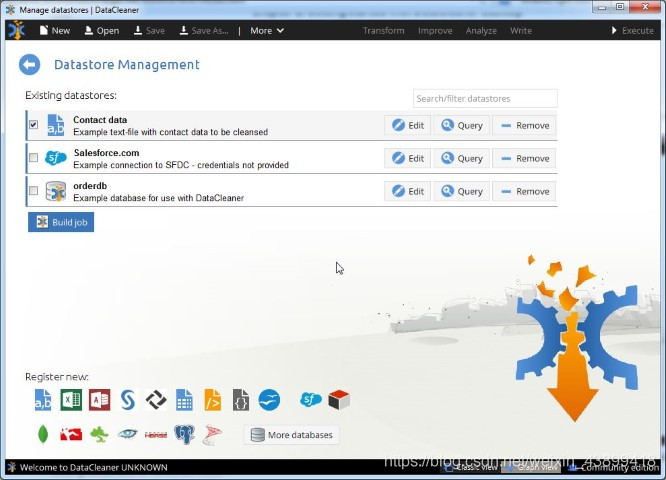
下面是启动DataCleaner（桌面社区版）时显示的初始屏幕截图。通过单击屏幕底部的按钮，可以在“build new job”或“Manage datastores”屏幕中添加新的数据存储。



可以使用左上角New按钮，添加文件数据存储（datastore）。下面是一些按钮，可以添加数据库或云服务。



如果文件是使用拖放区添加的，则将保留其格式。“Manage datastores”按钮管理者着各种各样的数据存储记录。



这个“数据存储管理Datastore management”屏幕-除了查看和编辑现有数据存储之外-有一个选项可以根据其类型添加新的数据存储。在屏幕底部选择一个适合您的数据存储类型的图标。

注册（“创建”）自己的数据存储后，便可以直接从以往存储列表中选择它，然后单击“Build job”开始使用它！

！提示  
您还可以通过配置文件来配置数据存储(conf.xml格式)，这既有优点也有缺点。  
有关更多信息，请阅读相关配置那一章。

### **向job中添加组件**

有几种不同类型的组件可以添加到job中：

* Analyzers (分析器)，这是最重要的组件。实际上，job要运行成功至少得有一个分析器(Analyze)（如果执行作业时没有添加分析器，DataCleaner将建议添加一个基本分析器，将输出保存到文件中）。分析器是一个组件，它检查它接收到的数据并生成结果或报告。大多数的数据分析都是分析器来创建的。
* Transformers(转换器)，是用于在分析数据之前修改数据的组件。有时有必要提取一个值的一部分或将两个值组合起来，以便正确地了解特定的数据。在其他场景中，转换器可用于执行参考数据查找或其他相似的任务，并将操作结果放入job中的数据流中。转换器的结果是一组输出列。这些数据列列的工作方式与job中的常规数据列完全相同，只是它们在被具体化之前在流中一个比较前面的步骤。
* Filters (过滤器) 是在job中分割处理流的组件。一个过滤器将有许多可能的结果，根据过滤器的结果，一个特定的数据行可能由不同的子流处理。过滤器通常只是用来忽略数据分析中的某些行，例如空值或超出某范围的值。

这些组件都将在job图中显示为一个节点。双击组件（图形节点）将显示其配置对话框。

使用job区域左侧的各种菜单下各种细节的组件都可添加到你的job中。菜单在左侧的组件库中可用，或者拖拽到画布中的空白区域。有关特殊的transformers 和 filters的更多信息，请参阅Transform章节。

Analyzers 是使用“Analyze”菜单（在大多数情况下）添加到作业中的，但也可以使用“Write”菜单将输出保存到数据存储中。有关特定分析仪的更多信息，请参阅Analyze章节。

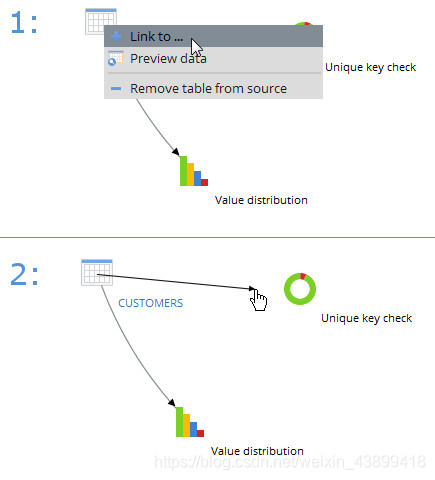
### **连接组件**

****连接组件****

简单地添加一个转换器(transformer )或过滤器(filter)实际上并不会改变你的工作本身！这是因为只有将这些组件连接在一起，它们才能发挥作用。

****转换器输出****

要连接转换器(transformer)，只需在图中的组件之间绘制一个箭头。可以通过右键单击第一个组件并从关联菜单中选择"Link to…“来开始绘制。进入绘图模式的另一种方法是选择元件，然后按Shift键连接元件。（箭头连接组件，根据自己的数据需求配置目标元件）



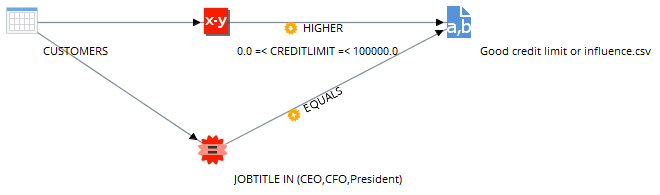
****过滤条件****

**要连接一个过滤器(filter)，您需要对它的任何一个结果设置依赖关系。所有组件在其配置对话框的右上角都有一个用于选择筛选结果的按钮。单击此按钮可选择要依赖的筛选器结果。**

**如果你有多个过滤器(filter)，你可以简单地通过拥有独立过滤器的相关结果来链接这些过滤器。这将要求满足过滤链中的所有过滤器要求，以便将记录传递给组件（和逻辑）。**

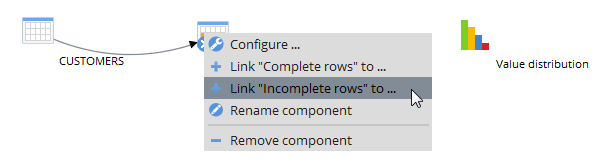
**666**

使用"Link to…”，还可以将多个过滤器(filter)连接到一种菱形的组件上。在这种情况下，若满足任何过滤器要求，记录将被传递给组件（或逻辑）。

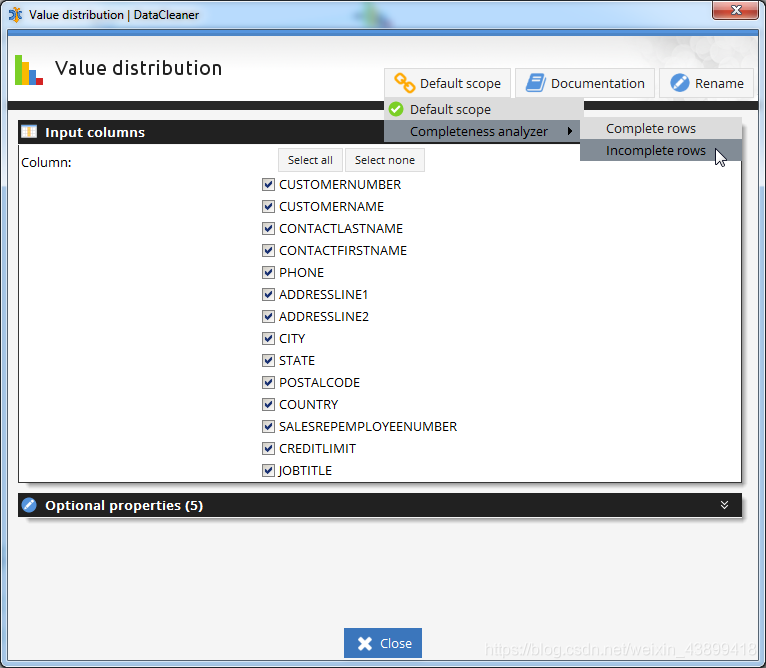


****输出数据流****

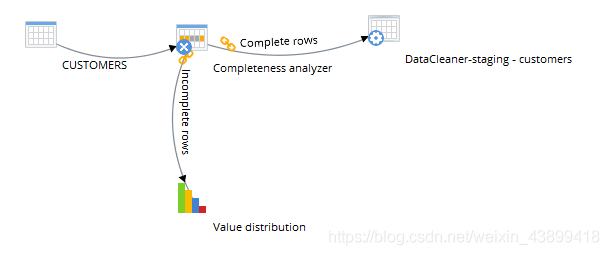
**"Link to…“选项将"main flow”(主流)中的组件连接在一起。然而，一些组件能够产生额外的输出数据流。例如，Completeness分析器的主要功能是在作业结果窗口中生成记录完整性摘要。此外，它还生成两个输出数据流-“Complete records"和"Incomplete records”。输出数据流的行为类似于源表，尽管这样的表是由组件动态创建的。这使得能够进一步处理这种输出。产生输出数据流的组件在右击菜单中有额外的"Link to…“位置，用于将输出与后续组件连接起来。**

****

双击一个组件会出现一个配置对话框，可以用来选择该组件的数据输入列，而不是用"Link to…” 菜单选项连接组件。在对话框的右上角，可以选择组件的范围。在作用域之间切换使我们可以从“主流”（默认作用域）或输出数据流中选择数据输入列。



使用输出数据流的示例job：



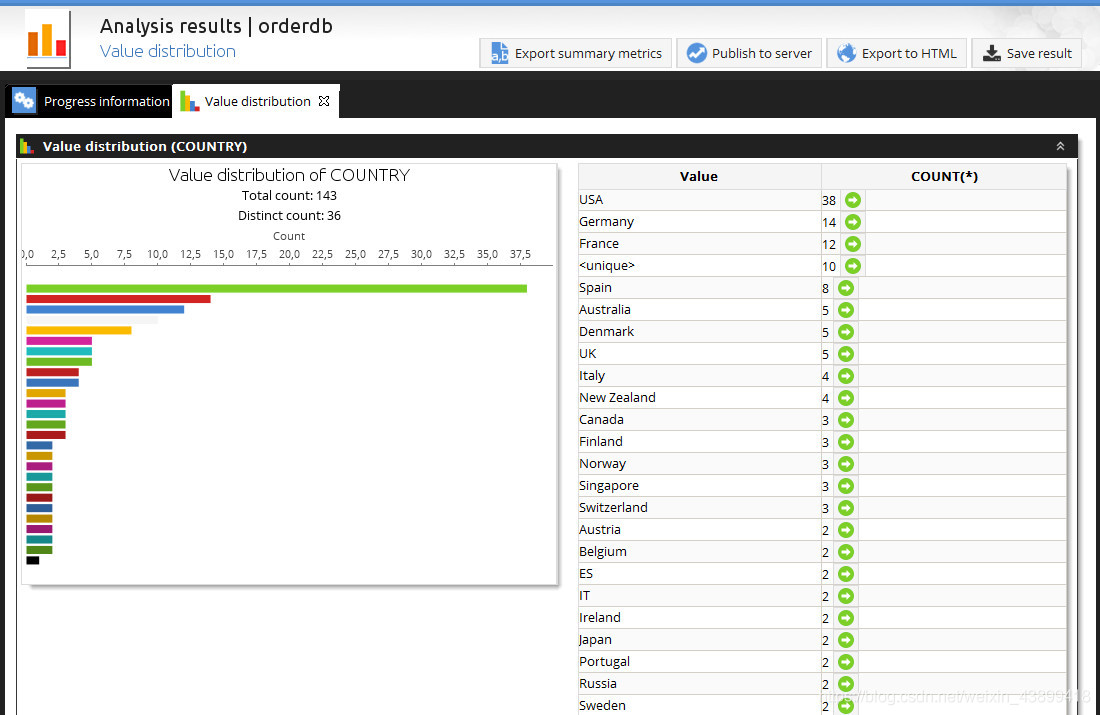
提示！！！  
画布显示消息（在屏幕底部），其中包含需要在其他窗口中执行的下一步操作的说明，以构建有效的job。

### **执行job**

当一个job已经生成时，你可以执行它。要检查作业是否已正确配置并准备好执行，请检查“job building”窗口底部的状态栏。（job白布的下面会有提示，可点击右上角按钮执行job了）要执行作业，只需单击窗口右上角的"Execute"按钮。这将打开结果窗口，其中包含：

* “Progress information”选项卡，其中包含执行作业时的有用信息和进度指示。
* 生成result/report每个组件类型的附加选项卡。例如，“Value distribution(值分布)”，如果这样的组件被添加到作业中。

以下是值分布分析器(Value distribution Analyzer)结果窗口的示例：



### **保存和打开job**

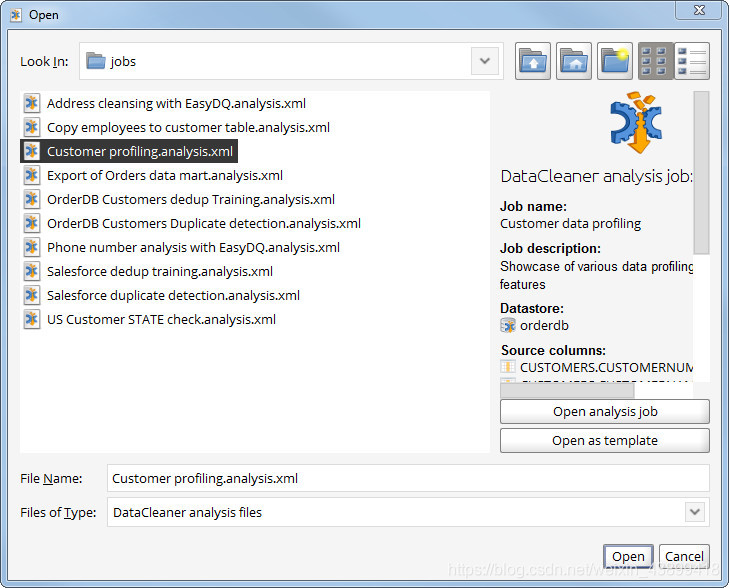
您可以保存job以便以后再次使用。保存job很简单：只需单击窗口顶部面板中的“Save”按钮。

分析数据相关的job以“.analysis.xml”的后缀名文件保存。这些文件是可使用任何XML编辑器读取和编辑。

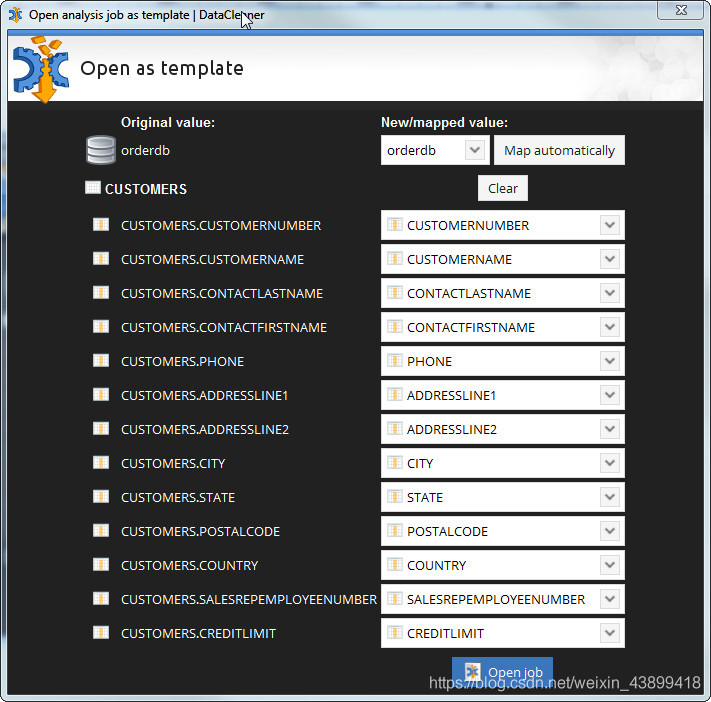
打开job可以使用“Open”菜单项完成。打开job将还原一个作业生成窗口，从中可以编辑和运行job。

### **job示例模板**

DataCleaner包含一个功能，您可以将作业重用于多个数据存储或同一数据存储中的多个列。我们称此功能为“template jobs模板作业”。当打开job时，会显示一个文件选择器。选择作业文件时，将出现一个面板，其中包含有关作业的一些信息以及可用操作：



如果单击“‘Open as template’”按钮，将显示一个对话框，您可以在其中将作业的原始列映射到一组新列：



首先需要指定要使用的数据存储。在左侧可以看到原始数据存储的名称，但job不限于仅使用此数据存储。从列表中选择一个数据存储，下面列的字段将变为活动字段。然后需要映射各个列。如果有两个具有相同列名的数据存储，可以单击“Map automatically”（自动映射）按钮，它们将被自动分配。否则，需要从新数据存储的可用列映射列。

最后，您的job可能包含’Job-level variables(job级别变量)’。这些是您可能还需要填写的job的可配置属性。完成这2-3个步骤后，单击"Open job"按钮，DataCleaner就可以在一组新列上执行job了！

### **2.8 将清洗后的数据写入文件**

尽管DataCleaner的关注焦点是分析数据，但在这种分析过程中，您经常会发现自己通过在数据上使用转换器和过滤器来改进数据。在这种情况下，显然您需要导出经过改进/清理的数据，以便在分析以外的其他情况下使用这些数据。

有关写入清除数据的更多信息，请参阅 Writer 章节。

# **组件分析**

## **Transform**

### **JavaScript 转换器**

JavaScript转换器(JavaScript transformer)允许用户定义自己的脚本，该脚本可以执行相当复杂的操作，比如调节、循环。它也可以用来表达小型企业的规则。

对于本文档，JavaScript的完整引用超出了范围。但是我们将展示一些例子，更重要的是讨论可用的变量及其类型。

JavaScript转换器返回一个字符串。输入的脚本应提供此字符串作为脚本的最后一行。这就是模板脚本如下所示的原因（因此您可以实现eval（）函数）：

1. function eval() {
2. return \"hello \" + values[0];
3. }
4. eval();

变量(Variables):

|  |  |
| --- | --- |
| **变量(Variable)** | **说明(Description)** |
| values | 行数据中所有值的数组(由“Columns”属性映射)。  使用“values”可以获取第一个和第三个值，例如：  var first = values[0];  var third = values[2];  注意JavaScript数组索引是基于0的。也可以通过列名获取，而不是索引，如下所示：  var idValue = values[“id”]; |
| column\_name\* | 保留了变量名的任何列名也可作为变量直接使用到脚本的作用域中是同样有效的。 例如，如果有两个列，FIRST\_NAME 和 LAST\_NAME，则可以轻松地将它们连接起来： var fullname = FIRST\_NAME + " " + LAST\_NAME; |
| out | 对系统控制台的“输出”流的引用。 如果在控制台可见的情况下运行DataCleaner，则可以将消息打印到控制台，如下所示： out.println("Value: " + values[0]); |
| log | 对日志模块的引用。可以配置日志记录，日志消息存储在文件中，这比简单地使用“out”更灵活。下面是如何编写一些严重程度不同的日志消息：  log.debug(“This is a DEBUG message, it will probably be disregarded”);  log.info(“This is a INFO message, it will probably be written to the logs”);  log.warn(“This is a WARN message, it will most likely be written to the logs”);  log.error(“This is a ERROR message, it will almost certainly be written to the logs”); |

数据类型(Data types)：

|  |  |
| --- | --- |
| **数据类型(Data types)** | **说明(Description)** |
| STRING | 字符串值表示为JavaScript字符串，这意味着它们（除其他外）具有如下方法：  var str = values[0];  // get the length of a string  var len = str.length();  // uppercase variant of a string  var up = str.toUpperCase();  // lowercase variant of a string  var lw = str.toLowerCase();  更多信息，我们建议使用W3 schools JavaScript字符串资料。 |
| NUMBER | Numbers 被视为常规的JavaScript Numbers ，  这意味着它们（除其他外）具有如下方法和运算符：  var num = values[0];  // format with 2 decimals  var formattedNumber = num.toFixed(2);  // add, subtract, multiply or divide  var m = (4 + num \* 2 - 1) / 2; |
| DATE | 日期值有点不同被视为Java日期，但留给您的接口几乎与常规JavaScript日期相同。以下是典型方法的总结：  var d = values[0];  var year = d.getYear();  var month = d.getMonth();  var date = d.getDate();  var hour = d.getHour();  var minutes = d.getMinutes();  var seconds = d.getSeconds();  // milliseconds since 1970-01-01  var timestamp = d.getTime();  更完整的资料，请查阅Java日期类。 |
| BOOLEAN | 布尔值（真/假）true/false |

### **调用子分析job**

通过这种转换，可以组成一个嵌入另一个保存的作业的功能的作业。

所调用的作业将从其分析部分剥离，这意味着只保留转换。这样，您就可以将复杂的转换流拆分为更小的部分—每个部分都表示为一个作业文件。

要配置转换，请从当前作业的数据源中选择需要通过管道传输到子作业的列。所选列的数量必须与子作业中定义的列的数量相同。系统会自动监测，否则无法进行转换。

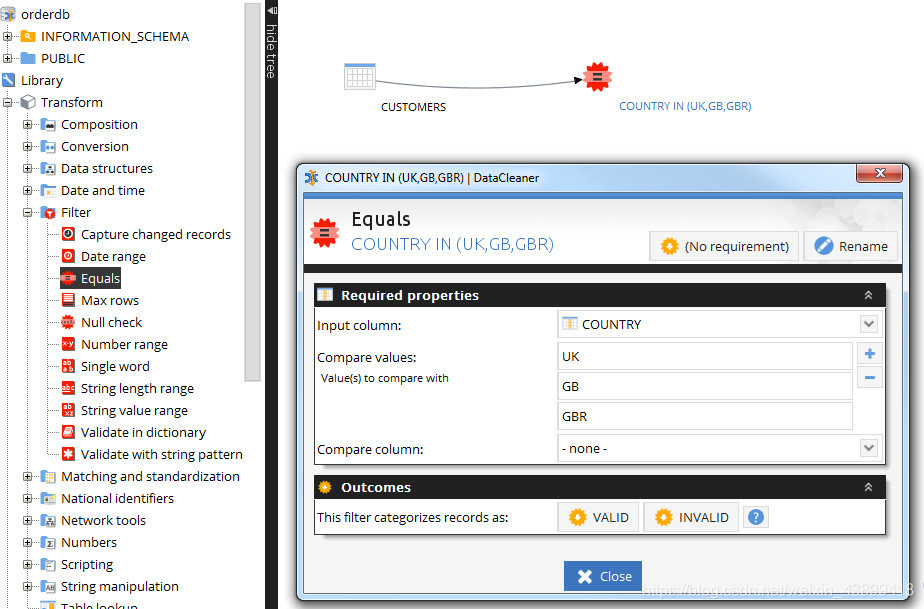
### **Apply classifier & Apply regression**

Apply classifier和Apply regression transformers 允许DataCleaner用户将经过训练的机器学习模型应用到传入的记录上。有关如何训练此类模型的详细信息，请参见机器学习分析器部分。  
两个转换器共享这些要配置的属性：

|  |  |
| --- | --- |
| **Property** | **Description** |
| Features | 选择用于特征提取的列。列的大小和顺序必须与训练模型时使用的列相匹配。 |
| Model file | 该文件包含在训练期间持久化的模型。 |

### **Equals 过滤器**

Equals过滤器提供了一种基于白名单/有效值列表的简单过滤条件的方法。只需输入一个为给出的列合适的值列表，然后就可以将流映射到过滤器的有效结果上。下面是一个Equals过滤器配置的示例，其中检查有效的性别值。



使用加号/减号按钮可以增大或缩小值列表。  
如果将Equals过滤器作为流中的第一个组件放置，那么它将以修改原始查询的方式进行优化。这意味着，如果您只想对job中使用的数据进行采样，那么它也是一个合适的过滤器。

### **Max rows 过滤器**

最大行数(Max rows)过滤器用于限制在作业流中进一步传递的记录数。  
如果将最大行数过滤器作为流中的第一个组件放置，那么它将以修改原始查询的方式进行优化。这意味着，如果您只想对作业中使用的数据进行采样，那么它也是一个合适的过滤器。

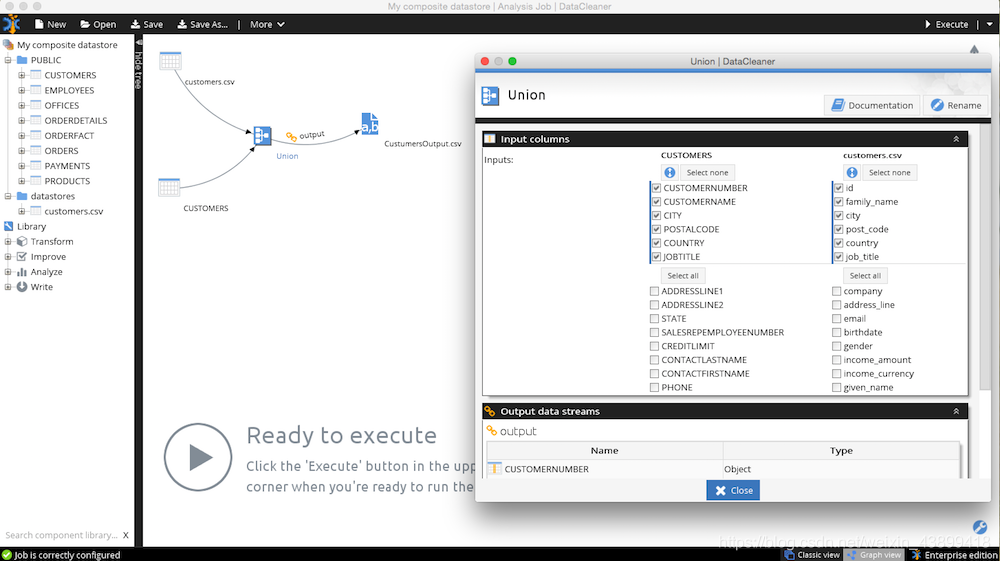
### **Not null 过滤器**

非空(Not null)过滤器是一个简单的过滤器，可用于从流中排除null值。此外，您还可以选择是否接受空字符串（" "）。如果将非空过滤器作为流中的第一个组件放置，那么它将以修改原始查询的方式进行优化。这意味着，如果您只想对作业中使用的数据进行采样，那么它也是一个合适的过滤器。

### **Union转换器**

联合转换器(Union transformer)允许您将多个流合并为一个流。提供相当于表的并集。使用它来融合来自不同源表的数据流。您可以定义新字段，这些字段的值表示其中一个输入流中可用的任何内容。

下面是一个带有联合转换和复合数据存储的作业示例。复合数据存储包含多个数据源，每个数据源都包含客户信息。本例中的Union transformer旨在合并这些数据源的表，以便单个数据流可以与来自所有源的记录一起使用。

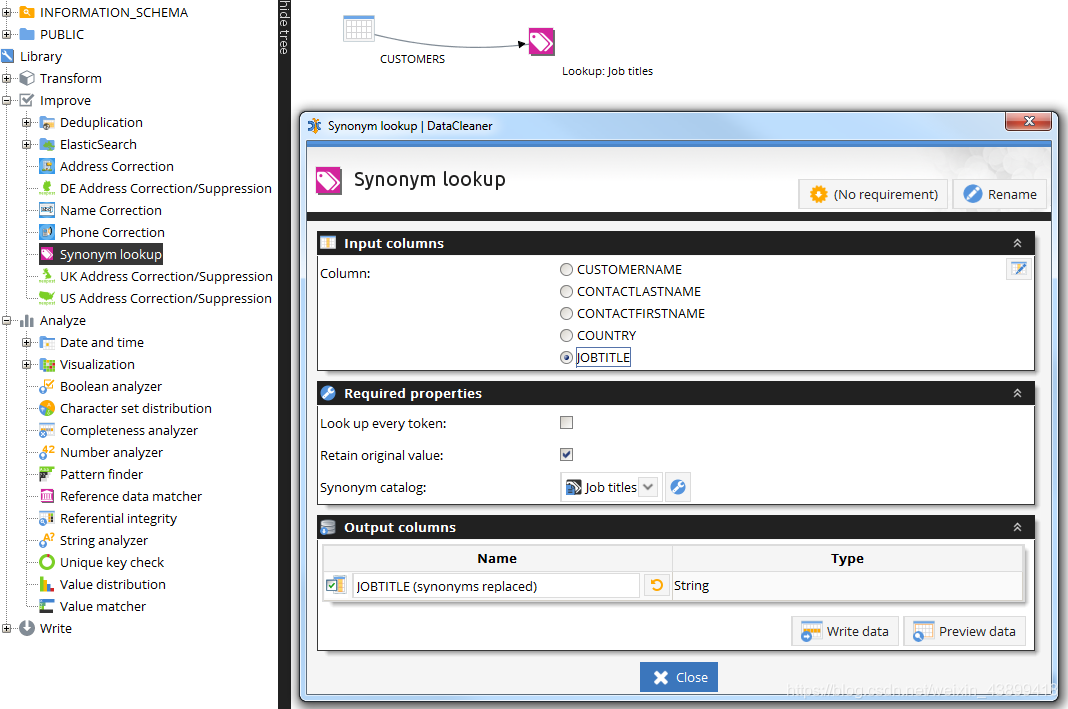


Union转换的配置是通过排列数据源的列来完成的。在所示的示例中，‘CUSTOMERNUMBER’ 和 ‘id’ 被合并到一个新字段中。类似地，‘CUSTOMERNAME’ 和 ‘family\_name’ 组合在一起，以此类推。

## **Improve**

### **[4.1 近义词查找](https://blog.csdn.net/weixin_43899418/article/details/113538844)**

近义词查找转换(Synonym lookup)是DataCleaner标准化和清理数据能力的关键部分。使用此组件，您可以在近义词目录中查找值，如果发现它是近义词，则将其替换为其主项。  
下面是近义词查找的配置面板的屏幕截图：



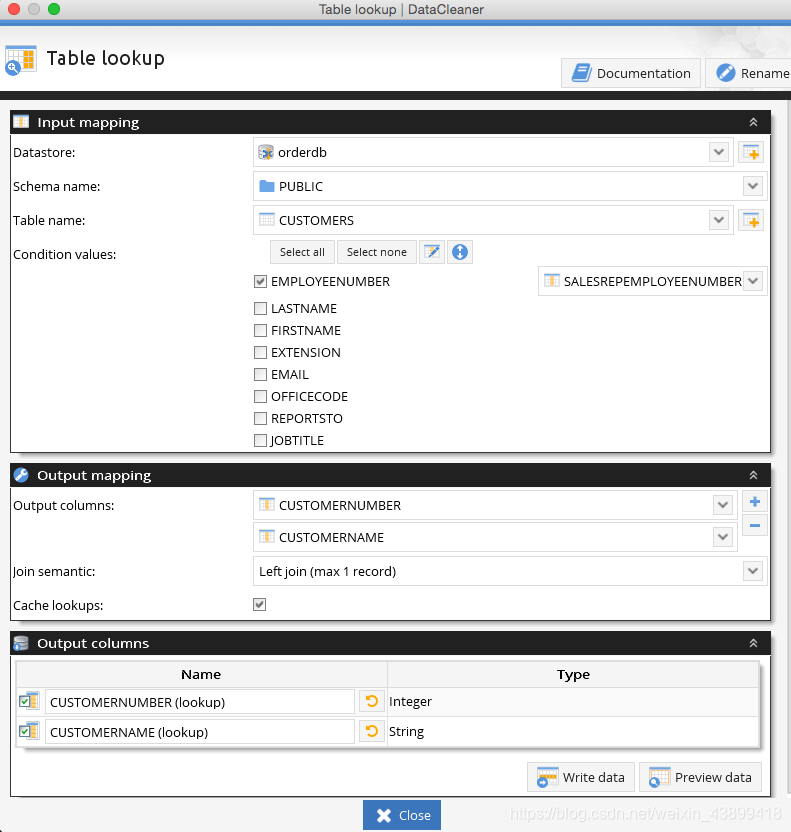
近义词查找的配置很简单：

* 选择要应用于查找函数的列。
* 使用 ‘Retain original value’ 选项确定是否应保留不匹配的值（非近义词），或者如果不匹配，是否应返回null值。
* 选择用于查找的近义词目录。

如果近义词目录包含某个特定列的所有允许值，则最好取消选中 ‘Retain original value’ 复选框，然后对生成的输出列执行简单的非空检查。如果发现空值，则是因为列中存在近义词目录无法标准化的值。

### **[4.2 表查询](https://blog.csdn.net/weixin_43899418/article/details/113541761)**

表查询转换器允许您在不同的表中查找值。任何数量的列都可以用于映射（查找条件）和输出（查找结果）。  
表查询转换器的配置屏幕如下所示：



要进行映射，需要选择目标数据存储、模式和表名。一旦选中，您将能够选择在查找值时用于条件设置的列。

表查询的语义接近于左连接查询的语义。如果找不到查找值，则返回null。但是，如果发现多条记录与条件匹配，则只返回第一条记录。

请注意，表查询将使用缓存来查找值，以避免在目标表中查询每个传入值。

## **Analyze**

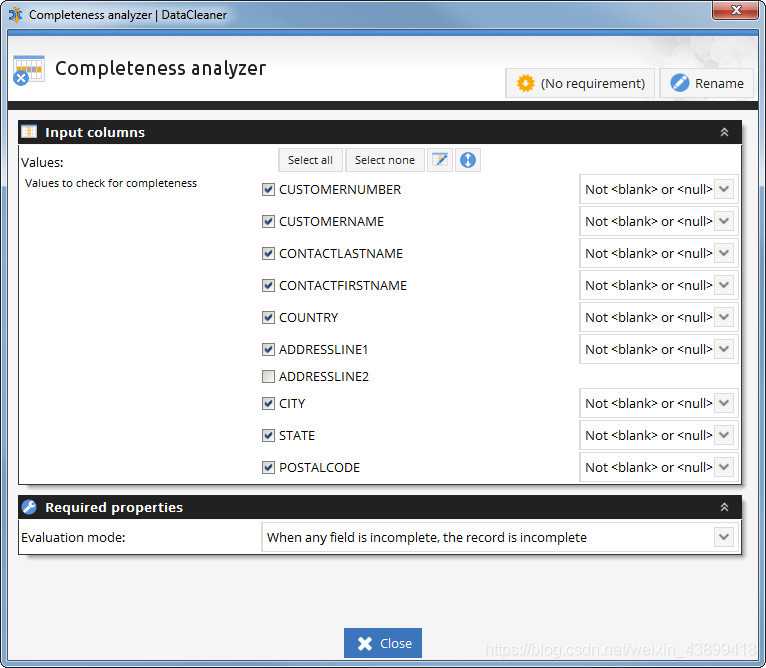
### **布尔分析器**

布尔分析器(Boolean analyzer)是针对布尔值(Boolean)的分析器。对于单个布尔值的数据列，它非常简单：它将显示列中 true/false 值的分布还有空值的分布。对于几个列，它还将显示值的组合和组合的频率。组合矩阵使布尔分析器成为一个方便的分析器，用于匹配转换器(transformer)和其他产生布尔值的转换器的组合。

除了输入列之外，布尔分析器没有任何配置参数。

### **完整性分析器**

完整性分析器(Completeness analyzer)提供了一种非常简单的方法来检查您的记录中的所有必填字段是否都已填写。把它想象成跨多个字段的一个大的非空检查。结合监控应用程序，该分析器可以轻松跟踪哪些记录需要附加信息。

以下是完整性分析器配置面板的屏幕截图：

完整性分析器的配置属性包括：

|  |  |
| --- | --- |
| **属性(Property)** | **说明(Description)** |
| Values | 选择想要被分析的数据列。对于每个选定的列，您可以选择分析器是简单的执行空值检查或null值检查。 |
| Evaluation mode | 这决定了完整性检查运行的模式。在这里，您可以配置如果所选值中的任何一个为 null/空值时，分析器是否应将其标记为“incomplete”，或者 在将记录计为不完整之前，是否所有值都需要为 null/空值。 |

### **字符集分布**

字符集分布(Character set distribution)分析器主要是根据字符集相似性来检查和绘制文本字符，如拉丁语、希伯来语、西里尔语、中文等。  
这样的分析有助于深入了解数据的国际方面。你能阅读和理解你所有的数据吗？它能在你的非国际化系统中工作吗？

### **日期间隔分析器**

日期间隔分析器(Date gap analyzer)用于识别记录的时间序列中的间隔。例如，如果您有记录开始日期和结束日期的员工登记系统，则此分析器非常有用。它将允许您识别数据中是否存在意外的间隙。

### **日期/时间分析器**

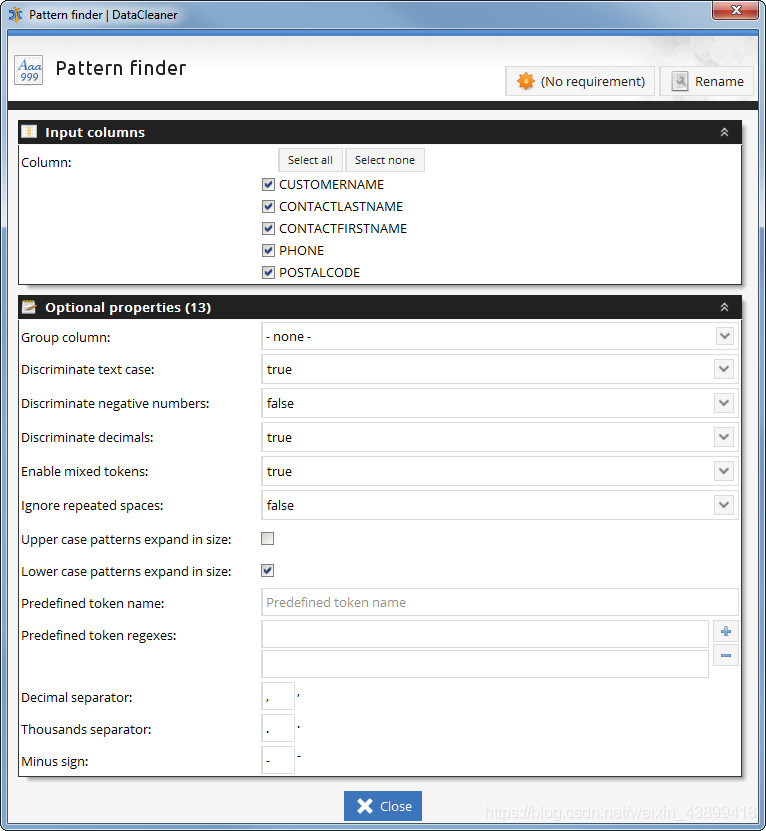
日期/时间分析器(Date/time analyzer)为时间数据列类型（如日期DATE、时间TIME 和时间戳TIMESTAMP ）提供通用的分析度量。

### **数字分析器**

数字分析器(Number analyzer)为数字列类型提供通用分析度量。

### **模式工具**

模式工具(Pattern Finder)是DataCleaner中比较先进但也非常流行的分析器之一。以下是Pattern finder配置面板的屏幕截图：



从截图中我们可以看到模式工具具有以下配置属性：

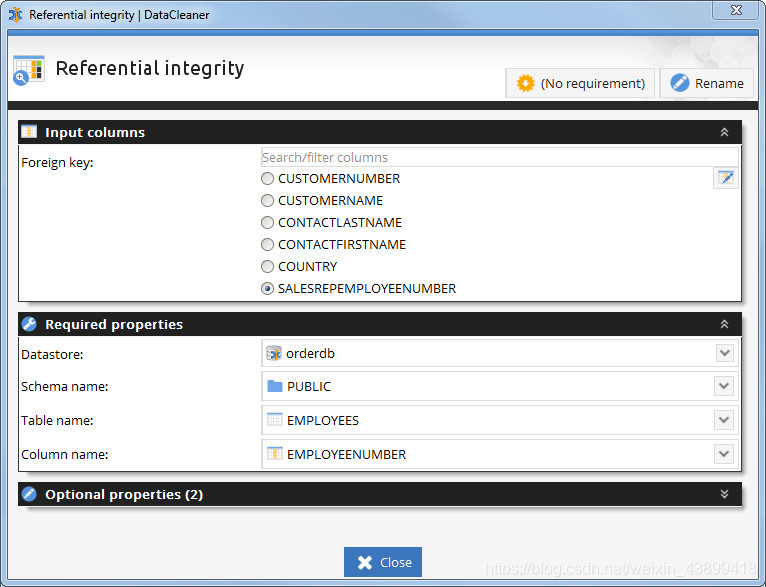
|  |  |
| --- | --- |
| **属性(Property)** | **说明(Description)** |
| 组列(Group column) | 允许您定义模式组列，通过模式组列，您可以将已标识的模式分离到单独的buckets/groups中。例如，假设您想检查客户的电话号码是否一致。如果您的客户是国际客户，则应按国家/地区列进行分组，以确保识别的电话模式与来自不同国家/地区的电话模式是否匹配。 |
| 区分文本大小写(Discriminate text case) | 定义是否根据文本大小写进行区分（即视为不同的模式部分）。如果为true，“DataCleaner” 和"datacleaner" 将被视为不同模式的实例，如果为false，它们将在同一模式中匹配 |
| 区分负数 | 解析数字时，此属性定义是否应区分负数和正数。 |
| 区分小数 | 解析数字时，此属性定义是否应将十进制数字与整数区分开来。 |
| 启用混合令牌(Enable mixed tokens) | 定义是否将同时包含字母和数字的标记分类为"mixed"，或者作为两个单独的标记。混合标记用问号(’?’)表示符号。这是更重要的配置属性之一。例如，如果启用了混合令牌（默认），那么所有这些值都将与相同的模式匹配：foo123、123foo、foobar123、foo123bar。如果未启用混合标记，则只有foo123和foobar123将匹配（因为123foo和foo123bar表示字母和数字标记的不同组合）。 |
| 忽略重复空格 | 定义是否基于空白量进行区分。 |
| 大写模式扩展大小(Upper case patterns expand in size) | 定义大写标记是否自动“扩展”大小。可扩展性指的是，如果一个候选者拥有相同类型的令牌，但大小不同，那么找到的模式是否会包含匹配项。大写字符的默认配置为false（即ABC与ABCD不匹配）。 |
| 小写模式扩展大小(Lower case patterns expand in size) | 定义小写标记是否自动“扩展”大小。与大写可扩展性一样，此属性指的是如果候选具有相同类型的标记，但大小不同，则找到的模式是否包含匹配项。小写字符的默认配置为true（即“abc”与“abc”不匹配）。两个"expandability"配置属性中的默认值意味着，例如，名称模式识别是有意义的：‘James’ 和 ‘John’ 都属于同一个模式（“aaaaaa”），而“McDonald”属于不同的模式（“aaaaaa”）。 |
| 预定义token名称(Predefined token name) | 预定义tokens使定义一个token成为可能，以便使用固定的值列表或正则表达式来查找和分类。通常，如果这些值包含要手动为其定义匹配类别的其他部分，则使用此选项。用’Predefined token name’属性定义此类类别的名称。 |
| 预定义标记正则表达式(Predefined token regexes) | 定义一些字符串值和/或正则表达式，用于将值与（预定义的）标记类别相匹配。 |
| 小数分隔符 | 分析数字时使用的十进制分隔符 |
| 千位分隔符(Thousand separator) | 千位分隔符，用于分析数字 |
| 减号标志(Minus sign) | 在分析数字时使用的减号字符 |

### **参考数据匹配器**

参考数据匹配器(Reference data matcher)分析器提供了一种简单的方法，可以将多个列与多个字典和/或多个字符串模式进行匹配。结果是所有列和所有匹配资源的匹配信息矩阵。

### **参照完整性**

使用参照完整性(Referential integrity)分析器，您可以检查记录之间的关键关系是否完好无损。分析器将处理单个表内、多个表之间甚至不同数据存储的表之间的关系。以下是参照完整性配置面板的屏幕截图：



对关系中具有外键的表应用分析器，并将其配置为对包含所有有效键的表进行检查。

|  |  |
| --- | --- |
| **属性(Property)** | **说明(Description)** |
| 缓存查找(Cache lookups) | 分析器是否应该通过缓存以前的查找结果来加速参照完整性检查。性能如何最终取决于要检查的键的重复次数。如果所有外键值或多或少都是唯一的，则必须将其关闭。但是，如果外键中有相当数量的重复（例如，订单行引用相同的产品或客户），则会加快查找速度。 |
| 忽略空值 | 定义是否应忽略“null”值或是否应将其视为完整性问题。当忽略时，所有外键值为空的记录都将被分析器丢弃 |

### **字符串分析器**

字符串分析器(String analyzer)为字符串列类型提供通用分析度量。字符串分析器特别关注的是单词、字符、特殊符号、音调符号和其他度量的数量，这些对于理解数据中出现的字符串值的类型至关重要。

### **主键检查**

主键检查分析器(Unique key check)提供了一种简单的方法来验证keys/IDs是否唯一。主键检查分析器的属性包括：

|  |  |
| --- | --- |
| **属性(Property)** | **说明(Description)** |
| 列(Column) | 选取此分析器应执行唯一性的列的检查。 |
| 缓冲区大小(Buffer size) | 缓冲区表示用于排序和比较key的内部资源。拥有一个大的缓冲区可以使分析器运行得更快，占用更少的磁盘资源，但这是以占用内存为代价的。如果作业尚未占用大量内存，建议将缓冲区大小增加到1M。 |

### **值分布**

值分布（Value distribution 通常也称为“频率分析”）允许您识别特定列的所有值。此外，您还可以调查哪些行属于特定值。

以下是值分布分析器的配置属性：

|  |  |
| --- | --- |
| **属性(Property)** | **说明(Description)** |
| 组列(Group column) | 允许您定义用于对结果进行分组的列。使用组列，可以将已标识的值分布划分为单独的buckets/groups。想象一下，例如，你想检查邮政编码和城市名称是否对应，或者如果你只是想分割你的值分布，例如国家或性别或。。。 |
| 记录唯一值(Record unique values) | 默认情况下，所有唯一值都将包含在值分布的结果中。如果分析的列包含许多唯一值（例如，如果它是唯一键），这可能会导致内存问题。如果对实际的唯一值不感兴趣，则取消选中此复选框仅计算（但不保存）唯一值。 |
| 前几个最常出现的值 | 如果分析只应显示“前5个最常见值”，则使用可选数字。如果提供了此属性，则分析结果将仅包含top/bottom 最常用值。 |
| 后几个最常出现的值 | 如果分析只应显示如“后5个最频繁值”，则使用可选数字。如果提供了此属性，则分析结果将仅包含top/bottom 最常用值。 |

### **值匹配**

值匹配器(Value matcher)的工作原理与值分布非常相似，只是它需要一个期望值列表，其他所有内容都放入一组 ‘unexpected values’ 中。这种值差异意味着两件事：

* 你得到了一个内置的验证机制。您可能只希望 “gender” 列的值为“M”和“F”，出现其他所有值在某种意义上都是无效的。
* 这种划分使得监视数据质量监视web应用程序中的特定值变得更容易。
* 这个分析器对于大型数据集的伸缩性更好，因为分组是确定性的，因此可以在批处理运行中进行准备。

### **工作日分布**

工作日分布(Weekday distribution）为日期列提供了频率分析，您可以很容易地确定日期字段表示哪些工作日。

### **机器学习**

DataCleaner提供了一组用于训练或测试机器学习模型的分析器。其思想是，可以使用记录来构建 (“train”) 一个数学模型，该模型基于记录的其他属性来描述任何给定属性/列的值。例如，您可以根据给定价格的历史数据和价格的相关因素（如大小、位置、条件和时间）对房价进行分类或预测。

支持两种用例：

* 分类，即确定一类记录的行为。例如，您可能希望根据客户的属性对最适合特定客户的产品进行分类。
* 回归，它决定了一个数字尺度上的连续值。例如，你可能想根据房子的特点来预测房子的价格。

DataCleaner内置了对以下类型机器学习模型的支持：

* Random Forests
* Support Vector Machines
* Neural Networks

在训练模型时，您提供了表示真实观察的数据。这些记录必须包含您将尝试使用回归进行分类或预测的属性的值。影响预测或分类的属性称为特征。特征必须是数字，以便为模型提供有意义的数学输入。因此，有时必须从原始值中提取特征，而不是按原样应用。例如，在文本分析中，当您试图确定一段文本的语言或性质时，通常需要从文本中提取n-grams。

DataCleaner提供以下特征提取策略（将列作为输入添加到training组件时选择）：

* Direct (0.0 to 1.0), Takes numerical values as-is.(按原样取数值。)
* Scaled (Min-Max), Scales numerical values from the minimum to the maximum value observed.(缩放（最小-最大），将观察到的数值从最小值缩放到最大值。)
* Vector (One-Hot Encoding), Generates a feature for every distinct value encountered. The values of the feature will be either 0 or 1 to indicate whether or not the record has that particular value.(向量（一个热编码），为遇到的每个不同值生成一个特征。特性的值将是0或1，以指示记录是否具有该特定值。)
* Vector (2-gram), Generates a feature for every 2-gram observed in the text.(向量（2-gram），为文本中观察到的每2-gram生成一个特征。)
* Vector (3-gram), Generates a feature for every 3-gram observed in the text.(向量（3-gram），为文本中观察到的每3-gram生成一个特征。)
* Vector (4-gram), Generates a feature for every 4-gram observed in the text.(向量（4-gram），为文本中观察到的每4-gram生成一个特征。)
* Vector (5-gram), Generates a feature for every 5-gram observed in the text.(向量（5-gram），为文本中观察到的每5-gram生成一个特征。)

## **Write**

### **[6.1 创建CSV文件](https://blog.csdn.net/weixin_43899418/article/details/113644347)**

将数据写入以逗号分隔的值文件。CSV文件是与其他系统进行互操作以及将数据加载到数据库中的常用选择。

### **[6.2 创建Excel表格](https://blog.csdn.net/weixin_43899418/article/details/113644565)**

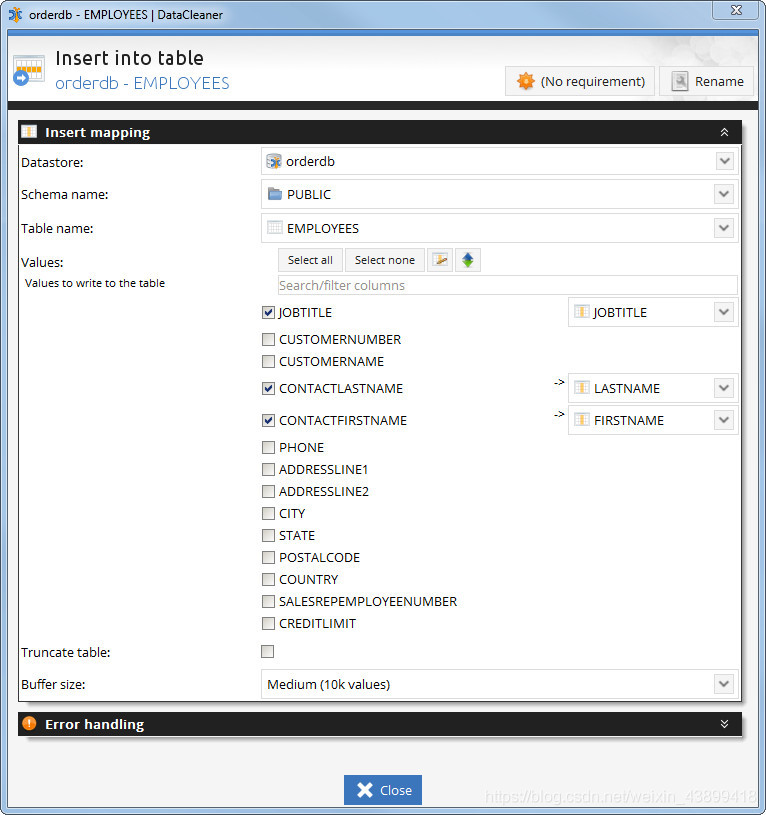
将数据集写入Excel电子表格。这种方法的一个优点是，一个文件可以包含多个工作表，并且在microsoft excel中很容易导航。缺点是对于非常大的数据集，它的性能较差。

### **[6.3 创建临时表](https://blog.csdn.net/weixin_43899418/article/details/113644718)**

将数据写入DataCleaner管理的嵌入式关系数据库。此选项主要用于暂存数据以供进一步分析。使用此功能的优点是它保留了列类型信息，可以处理大量数据，并且可以将多个数据集写入同一个数据存储。缺点是数据不容易被第三方应用程序读取（除非再次导出）。

### **[6.4 插入表格](https://blog.csdn.net/weixin_43899418/article/details/113644872)**

使用此编写器，您可以将数据插入现有数据存储的表中。如果您已经准备好了一个表布局，或者您想附加到一个数据库表中，那么这个写入选项就是适合您的。（可选）您可以用 “Insert into table”组件在插入数据之前删除表内容。这将删除表中的所有现有记录，对初始负载情况非常有用。



当前目标表可以来自以下任何数据存储类型：

* CSV文件。在这种情况下，数据将附加到文件中。
* Excel电子表格。在这种情况下，数据将附加到文件中。
* 关系数据库。在这种情况下，数据将使用INSERT语句插入到表中。
* MongoDB数据库。在这种情况下，数据将插入到MongoDB集合中。
* CouchDB数据库。在这种情况下，数据将被插入CouchDB数据库。
* Salesforce.com网站. 在这种情况下，数据将上传/插入到Salesforce.com网站使用SOQL web服务。
* ElasticSearch index。在这种情况下，数据将被索引到ElasticSearch中。

### **[6.5 更新表格](https://blog.csdn.net/weixin_43899418/article/details/113645392)**

Update table 编写器的工作方式与 ’ Insert into table ’ 编写器类似，只是它发出Update语句而不是Insert语句。这显然意味着它有一个附加属性，用于指定更新的条件（WHERE部分）。

# **数据参考**

## **Dictionaries**

### **[7.1字典](https://blog.csdn.net/weixin_43899418/article/details/113653428)**

字典是参考数据列表，用于对照某些黑名单或白名单对值进行验证或分类。字典通常是可枚举的和有限的，而字符串模式是动态的，每次都进行计算。

词典示例如下：

* 产品类型字典，如“珠宝”、“男装”、“运动装”等。
* 性别符号的字典，比如“M”、“F”或者“未知”。
* 年龄组名称词典（如婴儿、儿童、年轻人、成熟人、老年人）
* 两本男性和女性名字词典（以确定人的性别）

## **Synonyms (aka. Synonym catalogs)**

### **[8.1 文本文件同义词目录](https://blog.csdn.net/weixin_43899418/article/details/113680167)**

文本文件同义词目录是执行同义词替换的最简单也是最快的方法。只需创建一个文本文件，其中包含格式为的内容，其中主术语后面是逗号分隔的同义词列表，如下所示：

M,Male,Man,Guy,Boy  
F,Female,Woman,Girl

在上面的例子中，大多数典型的性别标记将被替换为“M”或“F”

### **[8.2 数据存储同义词目录](https://blog.csdn.net/weixin_43899418/article/details/113680747)**

如果同义词位于数据库或其他类型的数据存储中，则还可以基于此创建同义词目录。数据存储同义词目录允许您指定一个主术语列和多个同义词列。同义词目录将通过搜索/查询数据存储查找同义词匹配项。

## **String patterns**

### **[9.1字符串模式](https://blog.csdn.net/weixin_43899418/article/details/113680990)**

字符串模式为它们可能符合或不符合的字符串值定义一个“模板”。  
DataCleaner目前支持两种流行的字符串格式：

* 正则表达式是计算机科学中流行的一种通用字符串模式匹配语言。正则表达式确实需要一点时间来学习，但是一旦被利用，它就非常强大。解释正则表达式的语法绝对超出了DataCleaner文档的范围。如果您正在寻找有关Java正则表达式的资源，我们建议您学习Java正则表达式课程。
* 简单的字符串模式，使用与模式工具分析器相同的语法。例如像这样的模式aaaa@aaaa.aaa，可以用来匹配典型的电子邮件地址。

# **相关配置**

## **Configuration file**

### **XML模式**

配置文件(conf.xml格式)是属于XML命名空间"http://eobjects.org/analyzerbeans/configuration/1.0"的XML文件。

对于喜欢使用支持XML模式的编辑器来编辑XML文件的读者，您可以在此处找到此命名空间的XML模式： https://github.com/datacleaner/DataCleaner/blob/master/engine/xml-config/src/main/resources/configuration.xsd .

### **数据存储**

可以在元素标签 < datastore-catalog > 下的配置文件中配置数据存储。下面几节将详细介绍特定类型的数据存储。

**数据库（JDBC）连接**

MySQL

1. <jdbc-datastore name="MySQL datastore">
2. <url>jdbc:mysql://hostname:3306/database?defaultFetchSize=-2147483648</url>
3. <driver>com.mysql.jdbc.Driver</driver>
4. <username>username</username>
5. <password>password</password>
6. <multiple-connections>true</multiple-connections>
7. </jdbc-datastore>

Oracle

1. <jdbc-datastore name="Oracle datastore">
2. <url>jdbc:oracle:thin:@hostname:1521:sid</url>
3. <driver>oracle.jdbc.OracleDriver</driver>
4. <username>username</username>
5. <password>password</password>
6. <multiple-connections>true</multiple-connections>
7. </jdbc-datastore>

Microsoft SQL Server

1. <jdbc-datastore name="MS SQL Server datastore">
2. <url>jdbc:jtds:sqlserver://hostname/database;useUnicode=true;characterEncoding=UTF-8</url>
3. <driver>net.sourceforge.jtds.jdbc.Driver</driver>
4. <username>username</username>
5. <password>password</password>
6. <multiple-connections>true</multiple-connections>
7. </jdbc-datastore>

但是，如果要使用基于实例名称的连接，则必须运行SQL Server Browser服务，然后可以包含实例参数：下面是连接到SQLEXPRESS实例的示例：

1. <url>jdbc:jtds:sqlserver://hostname/database;instance=SQLEXPRESS;useUnicode=true;characterEncoding=UTF-8</url>

**CSV文件**

这是一个CSV文件数据存储的例子

1. <csv-datastore name="my\_csv\_file">
2. <filename>/path/to/file.csv</filename>
3. <quote-char>"</quote-char>
4. <separator-char>;</separator-char>
5. <encoding>UTF-8</encoding>
6. <fail-on-inconsistencies>true</fail-on-inconsistencies>
7. <header-line-number>1</header-line-number>
8. </csv-datastore>

**固定宽度值文件**

具有固定宽度值的文件可以通过两种方式进行注册—要么对所有列使用单个固定宽度大小，要么使用单个值宽度。  
下面是一个为所有列指定固定宽度的示例：

1. <fixed-width-datastore name="FIXED-WIDTH-ALL-COLUMNS">
2. <filename>/path/to/the/file.txt</filename>
3. <width-specification>
4. <fixed-value-width>20</fixed-value-width>
5. </width-specification>
6. <encoding>UTF-8</encoding>
7. <header-line-number>1</header-line-number>
8. <fail-on-inconsistencies>true</fail-on-inconsistencies>
9. <skip-ebcdic-header>false</skip-ebcdic-header>
10. <eol-present>true</eol-present>
11. </fixed-width-datastore>

下面是一个单独（2列）宽度规格的示例：

1. <fixed-width-datastore name="FIXED-WIDTH-2-COLUMNS">
2. <filename>/path/to/the/file.txt</filename>
3. <width-specification>
4. <value-width>20</value-width>
5. <value-width>30</value-width>
6. </width-specification>
7. <encoding>UTF-8</encoding>
8. <header-line-number>1</header-line-number>
9. <fail-on-inconsistencies>true</fail-on-inconsistencies>
10. <skip-ebcdic-header>false</skip-ebcdic-header>
11. <eol-present>true</eol-present>
12. </fixed-width-datastore>

下面是一个EBCDIC文件的示例:

1. <fixed-width-datastore name="FIXED-WIDTH-EBCDIC">
2. <filename>/path/to/the/file.ebc</filename>
3. <width-specification>
4. <value-width>2</value-width>
5. <value-width>10</value-width>
6. </width-specification>
7. <encoding>IBM01148</encoding>
8. <header-line-number>0</header-line-number>
9. <fail-on-inconsistencies>true</fail-on-inconsistencies>
10. <skip-ebcdic-header>true</skip-ebcdic-header>
11. <eol-present>false</eol-present>
12. </fixed-width-datastore>

**Excel表格 数据存储**

这是一个Excel表格数据存储的示例:

1. <excel-datastore name="my\_excel\_spreadsheet">
2. <filename>/path/to/file.xls</filename>
3. </excel-datastore>

**XML文件数据存储**

定义XML数据存储既可以用简单的（自动映射的）方法，也可以用高级的（性能和内存效率更高的）方法。简单的方法就是用文件名定义xml数据存储，如下所示：

1. <xml-datastore name="my\_xml\_datastore">
2. <filename>/path/to/file.xml</filename>
3. </xml-datastore>

当文件大小较小且层次结构不太复杂时，这种XML数据存储就可以工作。它的缺点是，它试图自动检测一个适合表示XML内容的表结构（这是一个树结构，而不是真正的表）。为了解决这个问题，还可以定义自己的表结构，在其中指定构成行的XPath和行中的值。举个例子：

1. <xml-datastore name="my\_xml\_datastore">
2. <filename>/path/to/file.xml</filename>
3. <table-def>
4. <rowXpath>/greetings/greeting</rowXpath>
5. <valueXpath>/greetings/greeting/how</valueXpath>
6. <valueXpath>/greetings/greeting/what</valueXpath>
7. </table-def>
8. </xml-datastore>

数据存储定义了一个表，其中每个记录都定义为与 XPath “/greetings/greeting” 匹配的元素。该表有两列，由“how”和“what”元素表示，它们是行路径的子元素。有关定义XML数据存储的表模型的XPath表达式的更多详细信息，请参阅MetaModel的教程（MetaModel是用于在DataCleaner中读取数据的数据访问库）。

**ElasticSearch 数据存储**  
这是ElasticSearch索引数据存储的示例:

1. <elasticsearch-datastore name="my\_elastic\_search\_index">
2. <hostname>localhost</hostname>
3. <port>9300</port>
4. <cluster-name>my\_es\_cluster</cluster-name>
5. <index-name>my\_index</index-name>
6. </elasticsearch-datastore>

**MongoDB 数据存储**  
这是一个完全指定的MongoDB数据存储的示例，示例表结构基于两个集合。

1. <mongodb-datastore name="my\_mongodb\_datastore">
2. <hostname>localhost</hostname>
3. <port>27017</port>
4. <database-name>my\_database</database-name>
5. <username>user</username>
6. <password>pass</password>
7. <table-def>
8. <collection>company\_collection</collection>
9. <property>
10. <name>company\_name</name>
11. <type>VARCHAR</type>
12. </property>
13. <property>
14. <name>customer</name>
15. <type>BOOLEAN</type>
16. </property>
17. <property>
18. <name>num\_employees</name>
19. <type>INTEGER</type>
20. </property>
21. <property>
22. <name>address\_details</name>
23. <type>MAP</type>
24. </property>
25. </table-def>
26. <table-def>
27. <collection>person\_collection</collection>
28. <property>
29. <name>person\_name</name>
30. <type>VARCHAR</type>
31. </property>
32. <property>
33. <name>birthdate</name>
34. <type>DATE</type>
35. </property>
36. <property>
37. <name>emails</name>
38. <type>LIST</type>
39. </property>
40. </table-def>
41. </mongodb-datastore>

**CouchDB 数据存储**  
这是一个完全指定的CouchDB数据存储的示例，示例表结构基于两个CouchDB数据库。

1. <couchdb-datastore name="my\_couchdb\_datastore">
2. <hostname>localhost</hostname>
3. <port>5984</port>
4. <username>user</username>
5. <password>pass</password>
6. <ssl>true</ssl>
7. <table-def>
8. <database>company\_collection</database>
9. <field>
10. <name>company\_name</name>
11. <type>VARCHAR</type>
12. </field>
13. <field>
14. <name>customer</name>
15. <type>BOOLEAN</type>
16. </field>
17. <field>
18. <name>num\_employees</name>
19. <type>INTEGER</type>
20. </field>
21. <field>
22. <name>address\_details</name>
23. <type>MAP</type>
24. </field>
25. </table-def>
26. <table-def>
27. <database>person\_collection</database>
28. <field>
29. <name>person\_name</name>
30. <type>VARCHAR</type>
31. </field>
32. <field>
33. <name>birthdate</name>
34. <type>DATE</type>
35. </field>
36. <field>
37. <name>emails</name>
38. <type>LIST</type>
39. </field>
40. </table-def>
41. </couchdb-datastore>

**Composite datastore(复合数据存储)**

这是一个复合数据存储的示例。它包含来自另外两个数据存储的数据：数据存储1和数据存储2。

1. <composite-datastore name="my composite">
2. <datastore-name>Datastore 1</datastore-name>
3. <datastore-name>Datastore 2</datastore-name>
4. </composite-datastore>

### **参考数据**

参考数据项（字典、同义词目录和字符串模式）在配置文件中 < reference-data-catalog > 标签中定义。以下是一些例子：

**字典**

字典存储在reference data部分的 < dictionaries > 标签中。可以添加三种类型的词典。  
Datastore dictionaries

1. <reference-data-catalog>
2. <dictionaries>
3. ...
4. <datastore-dictionary name="Lastnames" description="My datastore based dictionary">
5. <datastore-name>orderdb</datastore-name>
6. <column-path>EMPLOYEES.LASTNAME</column-path>
7. </datastore-dictionary>
8. ...
9. </dictionaries>
10. </reference-data-catalog>

Text file dictionaries

1. <reference-data-catalog>
2. <dictionaries>
3. ...
4. <text-file-dictionary name="Firstnames" description="My file based dictionary">
5. <filename>/path/to/first.txt</filename>
6. <encoding>UTF-8</encoding>
7. </text-file-dictionary>
8. ...
9. </dictionaries>
10. </reference-data-catalog>

Value list dictionaries

1. <reference-data-catalog>
2. <dictionaries>
3. <value-list-dictionary name="Greetings" description="My simple value list">
4. <value>hello</value>
5. <value>hi</value>
6. <value>greetings</value>
7. <value>godday</value>
8. </value-list-dictionary>
9. </dictionaries>
10. </reference-data-catalog>

**同义词目录**  
同义词目录(Synonym catalogs)存储在reference data部分的 < synonym-catalogs > 元素中。可以添加两种类型的词典。

Text file synonym catalogs

1. <reference-data-catalog>
2. <synonym-catalogs>
3. ...
4. <text-file-synonym-catalog name="textfile\_syn" description="My text file synonyms">
5. <filename>/path/to/synonyms.txt</filename>
6. <encoding>UTF-8</encoding>
7. <case-sensitive>false</case-sensitive>
8. </text-file-synonym-catalog>
9. ...
10. </synonym-catalogs>
11. </reference-data-catalog>

Datastore synonym catalogs

1. <reference-data-catalog>
2. <synonym-catalogs>
3. ...
4. <datastore-synonym-catalog name="datastore\_syn" description="My datastore synonyms">
5. <datastore-name>orderdb</datastore-name>
6. <master-term-column-path>CUSTOMERS.CUSTOMERNAME</master-term-column-path>
7. <synonym-column-path>CUSTOMERS.CUSTOMERNUMBER</synonym-column-path>
8. <synonym-column-path>CUSTOMERS.PHONE</synonym-column-path>
9. </datastore-synonym-catalog>
10. ...
11. </synonym-catalogs>
12. </reference-data-catalog>

**字符串模式**  
  
字典存储在reference data部分的 < string-patterns > 标签中。可以添加两种类型的字符串模式。

正则表达式（regex）字符串模式

1. <reference-data-catalog>
2. <string-patterns>
3. ...
4. <regex-pattern name="regex danish email" description="Danish email addresses">
5. <expression>[a-z]+@[a-z]+\.dk</expression>
6. <match-entire-string>true</match-entire-string>
7. </regex-pattern>
8. ...
9. </string-patterns>
10. </reference-data-catalog>

简单的字符串模式

1. <reference-data-catalog>
2. <string-patterns>
3. ...
4. <simple-pattern name="simple email" description="Simple email pattern">
5. <expression>aaaa@aaaaa.aa</expression>
6. </simple-pattern>
7. ...
8. </string-patterns>
9. </reference-data-catalog>

### **任务执行器**

任务执行器(Task runner)定义DataCleaner的引擎将如何执行分析作业的任务。通常不应编辑此标签内容。但是，这里有两个选项：

1. <multithreaded-taskrunner max-threads="30" />

定义具有30个可用线程的线程池的多线程任务运行程序。请注意，尽管30看起来是一个很高的数字，但是太小的线程池可能会导致问题，因为有些任务会安排额外的任务，因此当线程数非常低时，会有死锁的风险。

1. <singlethreaded-taskrunner />

定义单线程任务运行程序。在传统硬件或操作系统上，这种设置会更好，但它不会利用现代体系结构的多线程功能。

### **存储提供商**

存储提供程序用于存储执行分析作业时使用的临时数据。有两种类型的存储：大的（单个）值集合和“带注释的行”，即已采样或标记有用户感兴趣的特定类别的行。为了解释存储提供程序配置，让我们看一下默认元素：

1. <storage-provider>
2. <combined>
3. <collections-storage>
4. <berkeley-db/>
5. </collections-storage>
6. <row-annotation-storage>
7. <in-memory max-rows-threshold="1000" max-sets-threshold="200"/>
8. </row-annotation-storage>
9. </combined>
10. </storage-provider>

元素定义了一个组合存储策略。

集合使用berkeley db存储，berkeley db是Oracle的一个嵌入式数据库。这是推荐的收藏策略。

行注释存储在内存中。最多200组中有1000行的阈值。这意味着，如果使用同一类别注释了1000多条记录，则不会保存其他记录（因此用户无法查看）。此外，这意味着最多只能保存200个样本集。进一步的注释将不被采样，但度量仍然被计数。大多数用户场景不需要超过1000条带注释的记录进行检查，但如果确实有必要，可以采用不同的策略：

**对带注释的行使用MongoDB**

如果您有一个本地MongoDB实例，可以将其用作带注释行的存储。配置如下：

1. <row-annotation-storage>
2. <custom-storage-provider class-name="org.datacleaner.storage.MongoDbStorageProvider"/>
3. </row-annotation-storage>

MongoDB存储提供商解决方案显示了非常好的性能指标，但确实增加了安装的复杂性，这就是为什么它仍然被认为是实验性的，并且只针对聪明的用户。

## **Analysis job files**

### **[11.1 xml模式](https://blog.csdn.net/weixin_43899418/article/details/113988126)**

分析作业文件(Analysis job files)以与XML命名空间相关的XML格式编写”http://eobjects.org/analyzerbeans/job/1.0".

对于喜欢使用支持XML模式的编辑器来编辑XML文件的精通XML的读者，您可以在此处找到此命名空间的XML模式：https://github.com/datacleaner/datacleaner/blob/master/engine/xml-config/src/main/resources/job.xsd.

请继续阅读本章中有关作业文件格式各个部分的说明。

### **[11.2 Source section](https://blog.csdn.net/weixin_43899418/article/details/113988364)**

作业文件格式的源部分可能是手动编辑或查看的最有趣的部分。下面是一个示例源代码部分：

1. <source>
2. <data-context ref="orderdb" />
3. <columns>
4. <column path="PUBLIC.EMPLOYEES.EMPLOYEENUMBER" id="col\_0" type="INTEGER" />
5. <column path="PUBLIC.EMPLOYEES.LASTNAME" id="col\_1" type="VARCHAR" />
6. <column path="PUBLIC.EMPLOYEES.FIRSTNAME" id="col\_2" type="VARCHAR" />
7. <column path="PUBLIC.EMPLOYEES.EXTENSION" id="col\_3" type="VARCHAR" />
8. <column path="PUBLIC.EMPLOYEES.EMAIL" id="col\_4" type="VARCHAR" />
9. <column path="PUBLIC.EMPLOYEES.OFFICECODE" id="col\_5" type="VARCHAR" />
10. <column path="PUBLIC.EMPLOYEES.REPORTSTO" id="col\_6" type="INTEGER" />
11. <column path="PUBLIC.EMPLOYEES.JOBTITLE" id="col\_7" type="VARCHAR" />
12. </columns>
13. <variables>
14. <variable id="employee type" value="Sales Rep" />
15. </variables>
16. </source>

从这个来源部分，我们可以得出以下有趣的事实：

* 该作业正在使用名为“orderdb”的数据存储。此数据存储是如何配置的，作业不知道，因为它是在配置中定义的。该作业可能用于多个类似的数据存储，只要它们的名称是“orderdb”。
* 定义的列构成了作业将触发的源查询的基。为每一列分配一个人工ID，并提供关于其数据类型的提示。这些信息用于分离列或用新定义替换列。这意味着，如果您花了很长时间构建完美的作业，但希望将其应用于其他列，则可以在此处“只是”更改列定义并保留原始列ID。
* 在这个源代码部分，我们还看到了一些变量。这是一个可选的小节，并不常见。变量是可以在运行时用新值替换的属性值。有关更多信息和示例，请参见“可参数化作业”一章。

## **Logging**

### **[12.1 日志配置文件](https://blog.csdn.net/weixin_43899418/article/details/114062243)**

在DataCleaner 中登录是基于Apache的开源日志框架Log4j。使用Log4j，您可以在非常详细的级别配置日志记录，同时保持集中式配置。

在DataCleaner中配置日志记录有三种方法：

* 默认日志记录配置。这不需要更改DataCleaner的标准分发版。日志文件将在log/datacleaner.log 文件中。
* 指定您自己的XML日志配置。这要求您在datacleaner的根目录中放置一个名为log4j.XML的文件。
* 指定您自己的属性文件日志配置。这要求您将一个名为log4j.properties的文件放在DataCleaner的根目录中。
* 执行DataCleaner日志记录的自定义配置的推荐方法是使用XML格式。在下面的部分中，我们将使用示例来解释这种方法。有关Log4j配置的更多详细文档，请参阅Log4j网站。

### **[12.2 默认日志配置](https://blog.csdn.net/weixin_43899418/article/details/114063003)**

以下是XML格式的默认日志配置列表：

1. <?xml version="1.0"
2. encoding="UTF-8" ?>
3. <!DOCTYPE log4j:configuration SYSTEM
4. "log4j.dtd">
5. <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
6. <appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
7. <param name="Target" value="System.out"/>
8. <layout class="org.apache.log4j.PatternLayout">
9. <param name="ConversionPattern" value="%-5p %d{HH:mm:ss} %c{1} - %m%n"/>
10. </layout>
11. <filter class="org.apache.log4j.varia.LevelRangeFilter">
12. <param name="levelMin" value="WARN"/>
13. </filter>
14. </appender>
15. <appender name="fileAppender" class="org.apache.log4j.DailyRollingFileAppender">
16. <param name="File" value="${user.home}/.datacleaner/log/datacleaner.log"/>
17. <param name="DatePattern" value="'.'yyyy-MM-dd'.log'"/>
18. <layout class="org.apache.log4j.PatternLayout">
19. <param name="ConversionPattern" value="%-5p %d{HH:mm:ss.SSS} %c{1} - %m%n"/>
20. </layout>
21. </appender>
22. <logger name="org.apache.metamodel">
23. <level value="info"/>
24. </logger>
25. <logger name="org.datacleaner">
26. <level value="info"/>
27. </logger>
28. <root>
29. <priority value="warn"/>
30. <appender-ref ref="consoleAppender"/>
31. <appender-ref ref="fileAppender"/>
32. </root>
33. </log4j:configuration>

### **[12.3 修改日志级别](https://blog.csdn.net/weixin_43899418/article/details/114064461)**

这些是DataCleaner和Log4j中可用的日志级别，按优先级排序（最高优先级优先）：

* error
* warn
* info
* debug
* trace

通常不使用最底层的两个日志级别（调试和跟踪），除非开发人员必须调查意外情况。

可以全局或分层方式修改日志记录级别：

* 如果更改 < priority > 标签的value属性，则更改记录消息的全局阈值。
* 如果更改 < logger > 标签的级别，则会更改与特定记录器层次结构相关的日志优先级日志消息。

### **[12.4 选择性日志输出](https://blog.csdn.net/weixin_43899418/article/details/114064815)**

日志消息被打印到输出，通常是文件或控制台。在配置文件中，这是在< appender >标签中配置的。下面是一些您可以使用的可选附加器的示例。有关更多示例和文档，请参阅Log4j网站。

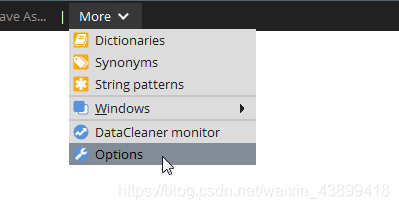
登录PostgreSQL数据库：

1. <appender name="jdbcAppender" class="org.apache.log4j.jdbc.JDBCAppender">
2. <param name="URL" value="jdbc:postgresql:db"/>
3. <param name="Driver" value="org.postgresql.Driver"/>
4. <param name="User" value="user"/>
5. <param name="Password" value="password"/>
6. <layout class="org.apache.log4j.PatternLayout">
7. <param name="ConversionPattern"
8. value="INSERT INTO log4j (log\_date,log\_level,log\_location,log\_message) VALUES ('%d{yyyy-MM-dd}','%p','%C;%L','%m')"/>
9. </layout>
10. </appender>

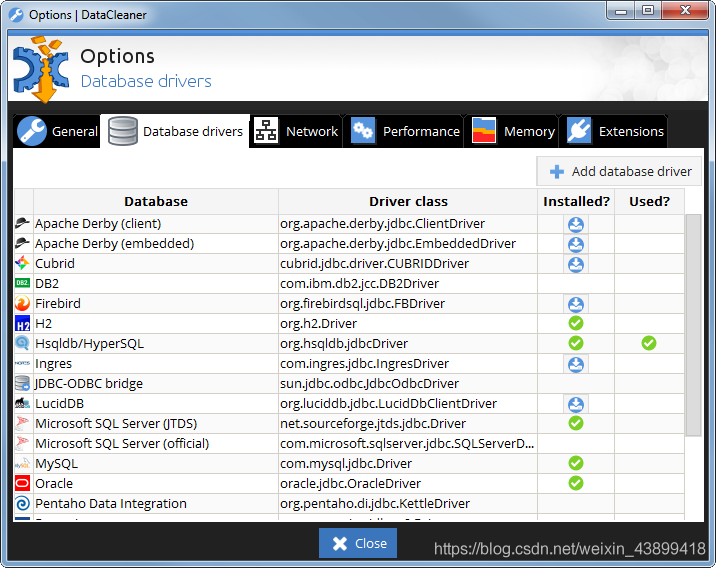
## **Database drivers**

### **[13.1 在DataCleaner桌面中安装数据库驱动程序](https://blog.csdn.net/weixin_43899418/article/details/114065085)**

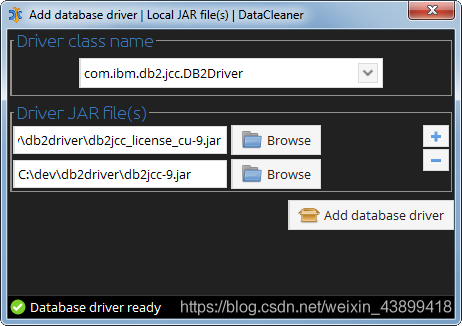
在DataCleaner desktop中安装数据库驱动程序是在应用程序本身运行时完成的。首先，找到顶部的more点击“Options ”菜单：



然后选择’Database drivers’选项卡。在此选项卡中，您将看到当前安装的数据库驱动程序列表（熟知的）：



如果单击’Add database driver’ 按钮，然后选择’Local JAR file(s)'选项，您将看到以下对话框：



在此对话框中，选择驱动程序类名和包含数据库驱动程序的文件。如果您不知道这是什么，请参阅数据库供应商关于JDBC数据库驱动程序的文档。  
在上面的示例中，我们看到了ibmdb2驱动程序，它涉及到安装两个文件，因为还必须包括软件许可证。

# **调用 DataCleaner jobs**

## **Command-line interface**

### **[14.1 可执行程序](https://blog.csdn.net/weixin_43899418/article/details/114082452)**

根据您的DataCleaner发行版，您将拥有以下CLI可执行文件之一：

* datacleaner-console.exe , 这是仅限Windows的可执行文件。
* datacleaner.cmd , 这是在Windows中启动DataCleaner的脚本。
* datacleaner.sh 这是在类Unix系统（如Linux和macos）中启动DataCleaner的脚本。
* 如果您在JavaWebStart模式下运行DataCleaner，那么就没有命令行界面！

### **[14.2 使用场景](https://blog.csdn.net/weixin_43899418/article/details/114082940)**

DataCleaner CLI的使用场景有：

* 执行分析job
* 列出已注册的数据存储
* 列出数据存储中的架构
* 列出架构中的数据表
* 列出数据表中的列
* 列出可用的分析仪、转换器或过滤器

通过使用-usage参数调用可执行文件，可以了解这些场景是如何实现的：

1. > datacleaner-console.exe -usage
2. -conf (-configuration, --configuration-file) FILE
3. : XML file describing the configuration of DataCleaner
4. -ds (-datastore, --datastore-name) VAL
5. : Name of datastore when printing a list of schemas, tables or columns
6. -job (--job-file) FILE
7. : An analysis job XML file to execute
8. -list [ANALYZERS | TRANSFORMERS | FILTERS | DATASTORES | SCHEMAS | TABLES | COLUMNS]
9. : Used to print a list of various elements available in the configuration
10. -s (-schema, --schema-name) VAL
11. : Name of schema when printing a list of tables or columns
12. -t (-table, --table-name) VAL
13. : Name of table when printing a list of columns

### **[14.3 执行分析job](https://blog.csdn.net/weixin_43899418/article/details/114083197)**

下面是如何执行分析作业-我们将使用捆绑的示例作业"employees.analysis.xml":

1. > datacleaner-console.exe -job examples/employees.analysis.xml
2. SUCCESS!
3. ...
4. RESULT:
5. Value distribution for column: REPORTSTO
6. Top values:
7. - 1102: 6
8. - 1143: 6
9. - 1088: 5
10. Null count: 0
11. Unique values: 0
12. RESULT:
13. Match count Sample
14. Aaaaaaa 22 William
15. Aaaa Aaa 1 Foon Yue
16. RESULT:
17. Match count Sample
18. aaaaaaaaaa 23 jfirrelli
19. RESULT:
20. Match count Sample
21. Aaaaa Aaa 17 Sales Rep
22. AA Aaaaaaaaa 2 VP Marketing
23. Aaaa Aaaaaaa (AAAA) 1 Sale Manager (EMEA)
24. Aaaaa Aaaaaaa (AA) 1 Sales Manager (NA)
25. Aaaaa Aaaaaaa (AAAAA, AAAA) 1 Sales Manager (JAPAN, APAC)
26. Aaaaaaaaa 1 President
27. ...

从清单中可以看到，分析结果将直接打印到命令行输出。如果要将结果保存到文件中，只需使用操作系统内置的功能将命令行输出管道化到文件中，通常使用’>'运算符。

通过在调用命令行界面时传递-ds参数，可以覆盖作业使用的数据存储：

1. > datacleaner-console.exe -job examples/employees.analysis.xml -ds orderdb

### **[14.4 列出数据存储内容和可用组件](https://blog.csdn.net/weixin_43899418/article/details/114083556)**

命令行界面允许列出数据存储内容和可用组件。如果需要的话，其预期用途是帮助手工编辑分析文件。通过使用-list参数，您可以获得数据存储的元数据以及允许您手动编写分析文件的DataCleaner组件。

如果您查看-usage命令的输出，那么列出数据存储的内容是非常简单的。下面是使用示例数据库“orderdb”的几个示例：

1. > datacleaner-console.exe -list datastores
2. Datastores:
3. -----------
4. Country codes
5. orderdb
6. > datacleaner-console.exe -list tables -ds orderdb
7. Tables:
8. -------
9. CUSTOMERS
10. CUSTOMER\_W\_TER
11. DEPARTMENT\_MANAGERS
12. DIM\_TIME
13. EMPLOYEES
14. OFFICES
15. ORDERDETAILS
16. ORDERFACT
17. ORDERS
18. PAYMENTS
19. PRODUCTS
20. QUADRANT\_ACTUALS
21. TRIAL\_BALANCE
22. > datacleaner-console.exe -list columns -ds orderdb -table employees
23. Columns:
24. --------
25. EMPLOYEENUMBER
26. LASTNAME
27. FIRSTNAME
28. EXTENSION
29. EMAIL
30. OFFICECODE
31. REPORTSTO
32. JOBTITLE

列出DataCleaner的组件是通过将-list参数设置为以下三种组件类型之一完成的：ANALYZER、TRANSFORMER或FILTER：

1. > datacleaner-console.exe -list analyzers
2. ...
3. name: Matching analyzer
4. - Consumes multiple input columns (type: UNDEFINED)
5. - Property: name=Dictionaries, type=Dictionary, required=false
6. - Property: name=String patterns, type=StringPattern, required=false
7. name: Pattern finder
8. - Consumes 2 named inputs
9. Input column: Column (type: STRING)
10. Input column: Group column (type: STRING)
11. - Property: name=Discriminate text case, type=Boolean, required=false
12. - Property: name=Discriminate negative numbers, type=Boolean, required=false
13. - Property: name=Discriminate decimals, type=Boolean, required=false
14. - Property: name=Enable mixed tokens, type=Boolean, required=false
15. - Property: name=Ignore repeated spaces, type=Boolean, required=false
16. - Property: name=Upper case patterns expand in size, type=boolean, required=false
17. - Property: name=Lower case patterns expand in size, type=boolean, required=false
18. - Property: name=Predefined token name, type=String, required=false
19. - Property: name=Predefined token regexes, type=String, required=false
20. - Property: name=Decimal separator, type=Character, required=false
21. - Property: name=Thousands separator, type=Character, required=false
22. - Property: name=Minus sign, type=Character, required=false
23. ...
24. > datacleaner-console.exe -list transformers
25. ...
26. name: Tokenizer
27. - Consumes a single input column (type: STRING)
28. - Property: name=Delimiters, type=char, required=true
29. - Property: name=Number of tokens, type=Integer, required=true
30. - Output type is: STRING
31. name: Whitespace trimmer
32. - Consumes multiple input columns (type: STRING)
33. - Property: name=Trim left, type=boolean, required=true
34. - Property: name=Trim right, type=boolean, required=true
35. - Property: name=Trim multiple to single space, type=boolean, required=true
36. - Output type is: STRING

### **[14.5 参数化job](https://blog.csdn.net/weixin_43899418/article/details/114083968)**

如果要使作业的一部分可参数化/变量化，则可以这样做。目前，这是一项仅通过编辑支持的功能。分析.xml文件，因为DataCleaner图形用户界面在保存作业时不存储作业变量。

在作业的源代码部分，可以添加变量，这些变量是将在整个作业中引用的键/值对。每个变量都可以有一个默认值，在未指定变量值的情况下将使用该值。下面是一个简单的例子：

1. ...
2. <source>
3. <data-context ref="my\_datastore" />
4. <columns>
5. <column path="column1" id="col\_1" />
6. <column path="column2" id="col\_2" />
7. </columns>
8. <variables>
9. <variable id="filename" value="/output/dc\_output.csv" />
10. <variable id="separator" value="," />
11. </variables>
12. </source>
13. ...

在这个例子中，我们定义了两个变量：filename和separator。具体的属性值，我们可以参考以下内容：

1. ...
2. <analyzer>
3. <descriptor ref="Write to CSV file"/>
4. <properties>
5. <property name="File" ref="filename" />
6. <property name="Quote char" value="&quot;" />
7. <property name="Separator char" ref="separator" />
8. </properties>
9. <input ref="col\_1" />
10. <input ref="col\_2" />
11. </analyzer>
12. ...

现在文件的属性值和Write to CSV文件中的Separator char属性都可以参数化了。要使用新的变量值执行作业，请使用命令行中的-var参数，如下所示：

1. DataCleaner-console.exe -job my\_job.analysis.xml -var filename=/output/my\_file.csv -var separator=;

### **[14.6 动态重写配置元素](https://blog.csdn.net/weixin_43899418/article/details/114085196)**

由于DataCleaner2.5版本，因此可以从命令行动态重写配置文件中的元素。这是一个功能，在您希望调用相同作业但配置细节略有不同的场景中非常有用。  
例如，您可能希望重用要在几个类似的CSV文件或类似的数据库环境上执行的相同作业。假设您有一个CSV数据存储，其定义如下：

1. </datastore-catalog>
2. <csv-datastore name="My csv file">
3. <filename>/path/to/file.csv</filename>
4. </csv-datastore>
5. </datastore-catalog>

要动态重写文件名，必须在命令行上使用“-D”参数指定属性路径（数据存储目录，然后是数据存储名称，然后是属性名称）。此外，任何空格或破折号都将被删除，随后的字符将被大写。最后它看起来像“camelCase”字符串，如下所示：

1. DataCleaner-console.exe ... -DdatastoreCatalog.myCsvFile.filename=anotherfile.csv

此机制可用于数据存储目录和引用数据目录中的任何配置属性。

## **Apache Hadoop and Spark interface**

### **Hadoop部署概述**

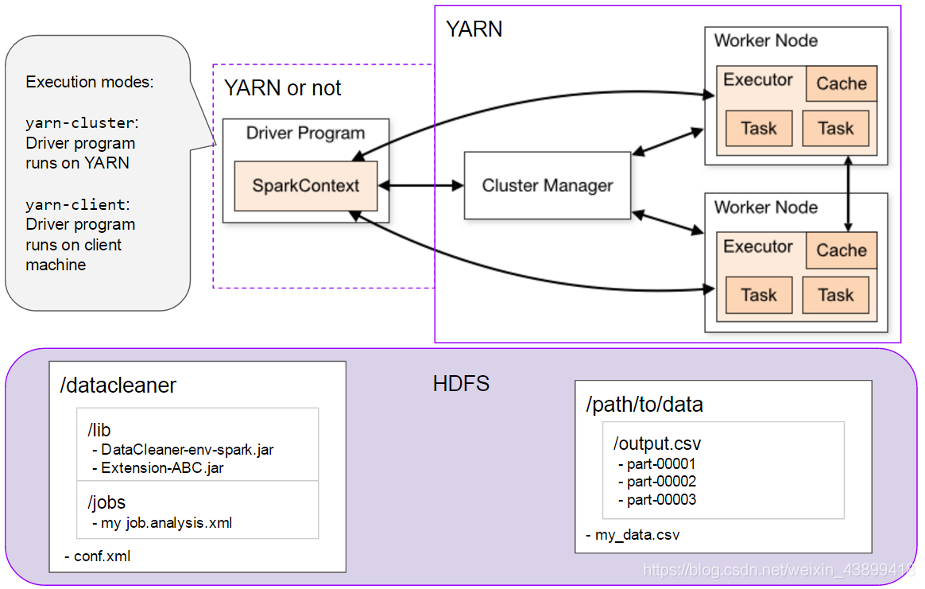
Apache Hadoop是一个分布式系统，有许多关键组件，其中有几个很重要：

* YARN，通常被称为Hadoop的“操作系统”。YARN是一个管理实体，它为运行特定的作业或任务分配资源。
* HDFS，这是Hadoop分布式文件系统。这是数据所在的位置，也是共享可执行文件的位置，以便可以在集群中的许多节点上获取分布式进程。
* Namenode是集群中的一个专用节点，负责处理HDFS和将数据分发给其他节点，即所谓的datanode。

此外，DataCleaner Hadoop是使用apachespark构建的，apachespark是一个与Hadoop以及其他集群技术一起工作的数据处理框架。Apache Spark的几个重要概念对于DataCleaner在Hadoop上的部署非常有用：

* 集群管理器，它是与集群协商的组件，例如Hadoop/YARN。从apachespark的角度来看，YARN是一个集群管理器。
* 驱动程序，它是指导集群管理器并告诉它做什么的程序。在apachespark for hadoop中，您有两种选择：作为外部进程运行驱动程序（“yarn-client”），或者作为yarn本身的进程运行驱动程序（“yarn-cluster”）。
* Executor，是Spark集群中执行作业分区（块）的节点。

在下图的顶部，您可以看到Hadoop/YARN和apachespark，以及它们是如何组件化的。



在图像的下半部分，您可以看到HDFS上DataCleaner的目录结构。如您所见，使用了通常的配置和作业文件，但放在HDFS上。HDFS上放置了一个特殊的JAR文件，作为apachespark执行器的可执行文件。

### **设置Spark和DataCleaner环境**

为了工作，Apache Spark需要环境变量HADOOP\_CONF\_DIR 或 YARN\_CONF\_DIR中的任何一个到包含Hadoop/Yarn配置文件（如core-site.xml , yarn-site.xml 等.）的目录中。

**将配置文件上传到HDFS**

Hadoop上的DataCleaner需要一个常规的DataCleaner配置文件(conf.xml格式). 最好将其上传到hadoop分布式文件系统（HDFS）。我们建议将此文件放入路劲 /datacleaner/conf.xml 下. 配置文件(conf.xml格式)的简单示例使用基于HDFS文件或目录的CSV数据存储：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <configuration xmlns="http://eobjects.org/analyzerbeans/configuration/1.0">
3. <datastore-catalog>
4. <csv-datastore name="mydata">
5. <filename>hdfs://bigdatavm:9000/path/to/data.txt</filename>
6. <multiline-values>false</multiline-values>
7. </csv-datastore>
8. </datastore-catalog>
9. </configuration>

注意这里用scheme、hostname和port指定的文件名：

1. <filename>hdfs://bigdatavm:9000/path/to/data.txt</filename>

这里指的是Hadoop Namenode的主机名和端口。  
也可以更含蓄地指定它，而不使用用户名和端口：

1. <filename>hdfs:///path/to/data.txt</filename>

甚至不使用scheme

1. <filename>/path/to/data.txt</filename>

**将可执行文件上传到HDFS**

在DataCleaner的安装中，您可以找到 ‘DataCleaner-spark.jar’ 文件.

这个jar文件包含在Hadoop上使用Apache Spark运行DataCleaner所需的核心内容。它还包含DataCleaner的标准组件。将这个jar文件上传到 /datacleaner/lib 文件夹中的HDFS。将您的DataCleaner许可证文件上传到 /datacleaner/hi\_datacleaner.lic.上传您需要的任何扩展jar文件（例如Groovy-DataCleaner.jar)到同一个文件夹。

**将作业文件上传到HDFS**

上传要运行的DataCleaner作业（一个DataCleaner .analysis.xml 作业文件）到HDFS。我们建议将此文件放入/datacleaner/jobs/myjob.xml 等路径中. 这些作业可以使用DataCleaner桌面UI构建，但是要确保它们也能很好地映射到HDFS上的配置文件。

基于上述数据存储的作业文件示例：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <job xmlns="http://eobjects.org/analyzerbeans/job/1.0">
3. <source>
4. <data-context ref="mydata" />
5. <columns>
6. <column id="col\_country" path="country" />
7. <column id="col\_company" path="company" />
8. </columns>
9. </source>
10. <analysis>
11. <analyzer>
12. <descriptor ref="Create CSV file"/>
13. <properties>
14. <property name="File" value="hdfs:///path/to/output.csv"/>
15. <property name="Separator char" value="&#44;"/>
16. <property name="Quote char" value="&quot;"/>
17. <property name="Escape char" value="\"/>
18. <property name="Include header" value="true"/>
19. <property name="Encoding" value="UTF-8"/>
20. <property name="Fields" value="[COUNTRY,CUSTOMERNUMBER]"/>
21. <property name="Overwrite file if exists" value="true"/>
22. </properties>
23. <input ref="col\_country" name="Columns"/>
24. <input ref="col\_company" name="Columns"/>
25. </analyzer>
26. </analysis>
27. </job>

此特定作业非常简单-它只是将数据从A复制到B。有关作业文件内容的说明：

* 作业引用的是mydata，mydata是配置文件中定义的CSV数据存储的名称。
* "File"属性中使用了另一个HDFS文件引用。文件名格式与配置文件中的格式相同。

如果您的桌面应用程序可以访问namenode，那么您可以在桌面应用程序中生成此作业，保存它并在spark上运行它。这个作业没有什么特别的地方可以让它在spark上运行，只是涉及的文件引用可以从hadoop节点解析。

### **使用Spark启动DataCleaner作业**

转到Spark安装路径以运行作业。使用以下命令行模板：

1. bin/spark-submit --class org.datacleaner.spark.Main --master yarn-cluster /path/to/DataCleaner-spark.jar
2. /path/to/conf.xml /path/to/job\_file.analysis.xml ([/path/to/custom\_properties.properties])

一种方便的组织方法是使用如下shell脚本，其中每个单独的参数都可以逐行编辑：

1. #!/bin/sh
2. SPARK\_HOME=/path/to/apache-spark
3. SPARK\_MASTER=yarn-cluster
4. DC\_PRIMARY\_JAR=/path/to/DataCleaner-spark.jar
5. DC\_EXTENSION\_JARS=/path/to/extension1.jar,/path/to/extension2.jar
6. DC\_CONF\_FILE=hdfs:///path/to/conf.xml
7. DC\_JOB\_FILE=hdfs:///path/to/job\_file.analysis.xml
8. DC\_PROPS=hdfs:///path/to/custom\_properties.properties
9. DC\_COMMAND="$SPARK\_HOME/bin/spark-submit"
10. DC\_COMMAND="$DC\_COMMAND --class org.datacleaner.spark.Main"
11. DC\_COMMAND="$DC\_COMMAND --master $SPARK\_MASTER"
12. echo "Using DataCleaner executable: $DC\_PRIMARY\_JAR"
13. if [ "$DC\_EXTENSION\_JARS" != "" ]; then
14. echo "Adding extensions: $DC\_EXTENSION\_JARS"
15. DC\_COMMAND="$DC\_COMMAND --jars $DC\_EXTENSION\_JARS"
16. fi
17. DC\_COMMAND="$DC\_COMMAND $DC\_PRIMARY\_JAR $DC\_CONF\_FILE $DC\_JOB\_FILE $DC\_PROPS"
18. echo "Submitting DataCleaner job $DC\_JOB\_FILE to Spark $SPARK\_MASTER"
19. $DC\_COMMAND

该示例清楚地表明，调用作业还需要几个参数。我们来看看：

* SPARK\_MASTER 表示驱动程序的位置，请参阅Hadoop部署概述一节。
* DC\_EXTENSION\_JARS 允许您向DataCleaner添加附加的JAR文件和扩展名。
* DC\_PROPS 也许是最重要的。它允许您添加一个.properties 文件，该文件可用于多种用途：

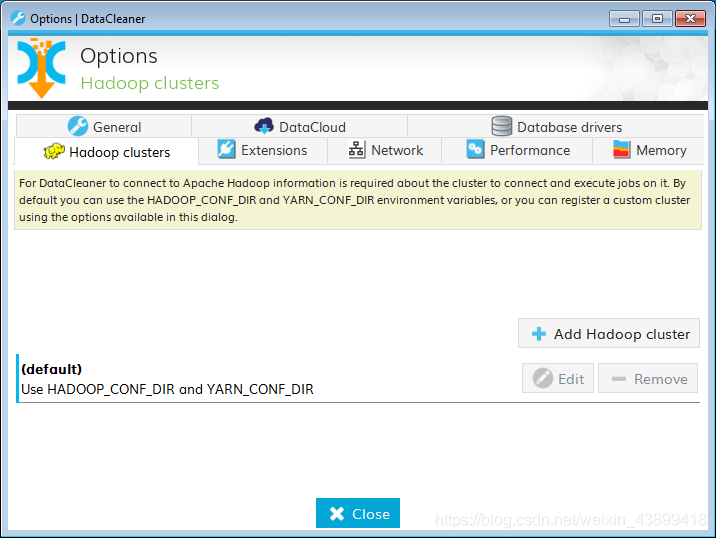
1. 特殊属性 datacleaner.result.hdfs.path 它允许您指定存储分析结果文件(.analysis.result.dat)的文件名（在HDFS上） 。它默认为/datacleaner/results/[job name]-[timestamp].analysis.result.dat
2. 特殊属性datacleaner.result.hdfs.enabled 可以是“true”（默认值）或“false”。将此属性设置为false将完全禁用DataCleaner作业的结果收集，这将显著提高性能，但不会收集或写入任何分析器结果。因此，这只适用于ETL风格的作业，作业的目的是从其他数据存储或文件中创建/插入/更新/删除。
3. 动态重写配置元素的属性。
4. 设置作业变量/参数的属性。

### **在DataCleaner桌面端使用Hadoop**

在DataCleaner 桌面端中，您可以处理位于HDFS上的CSV数据存储。

**配置Hadoop集群**

为了能够在Hadoop集群上从DataCleaner桌面端执行作业，您有许多配置选项，这些选项在选项对话框的Hadoop集群选项卡中进行管理。



* 默认

默认情况下，DataCleaner使用 HADOOP\_CONF\_DIR 和 YARN\_CONF\_DIR 环境变量来确定Hadoop/Yarn 配置文件（如core-site.xml and yarn-site.xml）.

* 使用配置目录

通过单击Add Hadoop cluster按钮，然后选择Using configuration directory，您可以通过添加包含Hadoop/Yarn配置文件的位置来注册其他Hadoop集群。

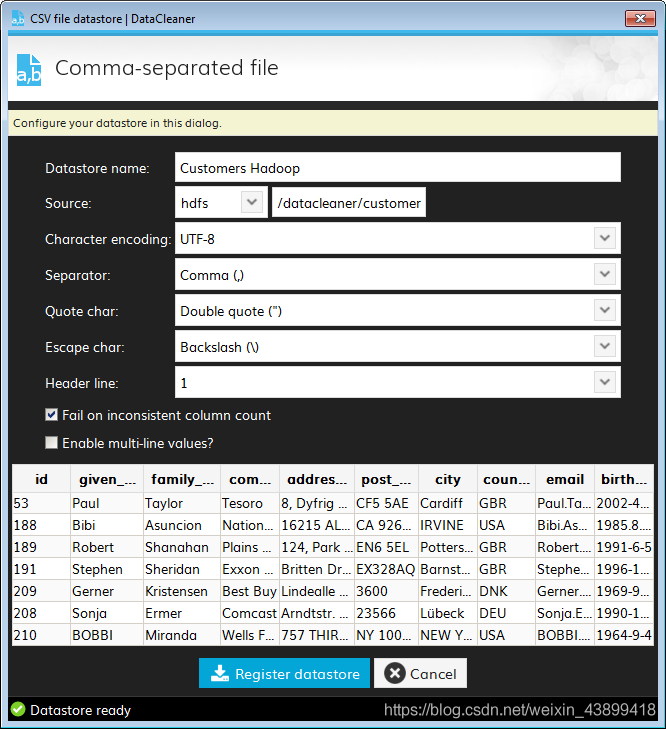
* 使用直接namenode连接

通过单击Add Hadoop cluster 按钮，然后选择Using direct namenode 连接，您可以使用其他Hadoop集群的文件系统URI（例如 hdfs://bigdatavm:9000/).

如果您添加了其他Hadoop集群，那么在HDFS上选择文件时，它首先会打开一个对话框，您可以在其中选择要从哪个Hadoop客户机中选择文件。

**HDFS上的CSV数据存储**

注册CSV数据存储时，您可以选择 “hdfs” 作为CSV源的方案。在路径字段中，您可以填写绝对路径，包括方案，例如 hdfs://bigdatavm:9000/datacleaner/customers.csv 或者HDFS上文件的相对路径，例如 /datacleaner/customers.csv。请注意，只有在设置了 HADOOP\_CONF\_DIR 或者 YARN\_CONF\_DIR 环境变量时，相对路径才起作用（请参阅设置Spark和DataCleaner环境）。



### **Hadoop接口的限制**

虽然DataCleaner的Hadoop接口允许在Hadoop平台上分布式运行DataCleaner作业，但有一些限制：

* 数据存储支持

目前，我们支持来自HDFS的一组有限的源数据存储。CSV文件是这里的主要来源。我们要求HDFS上的文件是UTF8编码的，并且只出现单行值。

* 不可分配组件

一些组件本质上是不可分配的。如果您的作业依赖于这些，DataCleaner将求助于在单Spark执行器上执行作业，这可能会对性能产生重大影响。

* 没有Namenode的Hadoop发行版

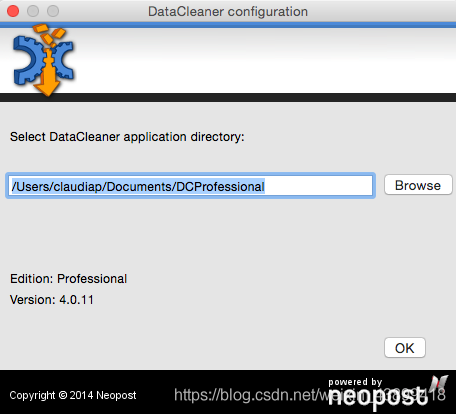
一些Hadoop发行版（比如MapR）已经用其他东西代替了Namenode的概念。这基本上是好的，但这确实意味着 username+port 的文件路径显然不起作用。

# **第三方集成**

## **Pentaho integration**

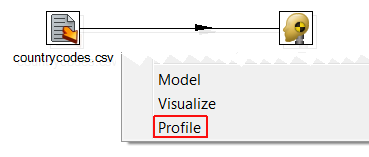
**[16.1 在Pentaho数据集成中配置DataCleaner](https://blog.csdn.net/weixin_43899418/article/details/114141644)**

为了在Pentaho中使用DataCleaner插件，需要在本地安装DataCleaner。DataCleaner安装可以是社区版或专业版。可通过 “Tools” 设置配置。此外，在设置DataCleaner配置之后，可以从同一菜单启动DataCleaner，而不依赖Pentaho中的上下文。



**[16.2 启动DataCleaner以分析Pentaho数据集成步骤](https://blog.csdn.net/weixin_43899418/article/details/114252376)**

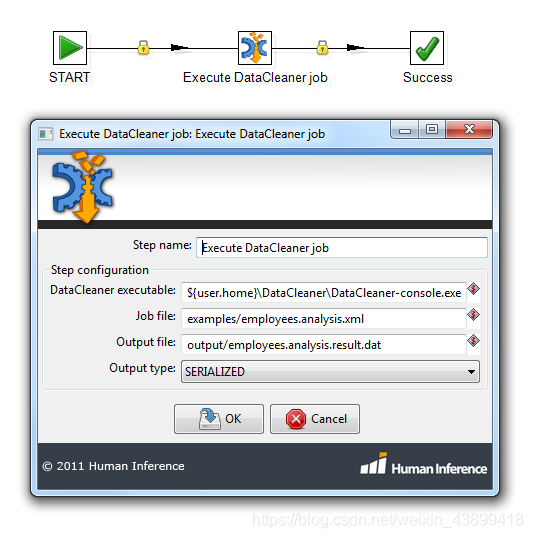
在Pentaho数据集成中，您可以通过右键单击转换的任何步骤来启动DataCleaner。这将启动DataCleaner，并预加载转换数据，准备进行分析。



此功能要求安装用于Pentaho数据集成的数据分析插件。关于这一点的说明和进一步的文档保存在Pentaho的wiki页面：[使用DataCleaner进行catter数据分析](http://wiki.pentaho.com/display/EAI/Kettle+Data+Profiling+with+DataCleaner)。 **[16.3 在Pentaho数据集成中运行DataCleaner作业](https://blog.csdn.net/weixin_43899418/article/details/114252769)**

Pentaho数据集成作业条目。如果您想将DataCleaner调度并集成到一个环境中，在这个环境中您可以迭代文件夹中的文件等，那么您可以使用Pentaho Data Integration（PDI），它是一个包含调度器的开源ETL工具。

构造一个PDI “job”（即不是“transformation”）并添加DataCleaner作业条目。可在 ‘Utility’ 子菜单中找到该条目。配置对话框如下所示：



最棘手的部分是填写可执行文件和作业文件名。请注意，所有配置选项都可以包含PDI变量，就像上面截图中的${user.home}。这是有用的。

# **开发者指南**

## **Architecture**

**[17.1 数据存取](https://blog.csdn.net/weixin_43899418/article/details/114253685)**

在DataCleaner中，所有数据源都称为“数据存储”。这个概念既包括本地读取/解析的源，也包括 ‘connected to’ 的源，例如数据库和应用程序。一些数据存储也可以写入，例如关系数据库。DataCleaner使用 Apache MetaModel framework 进行数据访问。从DataCleaner的角度来看，Apache元模型提供了许多特性：

* 与不同数据存储交互的常用方式。
* 一种编程查询语法，它抽象出特定于数据库的SQL方言，也可用于非面向SQL的数据存储（文件等）。
* 开箱即用的连接到许多源，例如CSV文件、关系数据库、Excel电子表格等等。
* 使用相同的通用模型对新源进行建模的框架。

DataCleaner数据存储模型还可以扩展，您可以自己实现新的数据存储，以便将DataCleaner连接到遗留系统、应用程序接口等。有关更多信息，请参阅 开发人员资源 一章。

**[17.2 处理框架](https://blog.csdn.net/weixin_43899418/article/details/114257612)**

DataCleaner处理数据的方式与大多数类似（ETL-like）工具略有不同。首先是应用多线程的方式，其次是DataCleaner有时可以在执行时优化图形的方式。

***多线程(Multithreading)*** ：DataCleaner中的多线程策略使该工具具有最小的阻塞和缓冲量以及最大的并行性和潜在的分布性。大多数类似ETL的工具都采用线程策略，其中作业中的每个组件都有自己的线程管理以及输入和输出缓冲区。在DataCleaner中，线程管理是这样做的：每个记录都是并行处理的—每个工作单元在一次传递中逐步完成整个作业图。这有很多有趣的特点：

* 组件之间有高度的自动“负载平衡”——在作业中最慢的组件周围没有限制和瓶颈。
* 系统适合于高度分布式处理，因为状态性是例外而不是规则。
* 在一项工作的组成部分之间，以缓冲区形式存在的浪费较少。
* 这种方法的一个缺点是无法保证处理记录的顺序。在数据分析和分析领域，这是很少需要的，如果需要的话，还有一些技术上的解决方法可以应用。

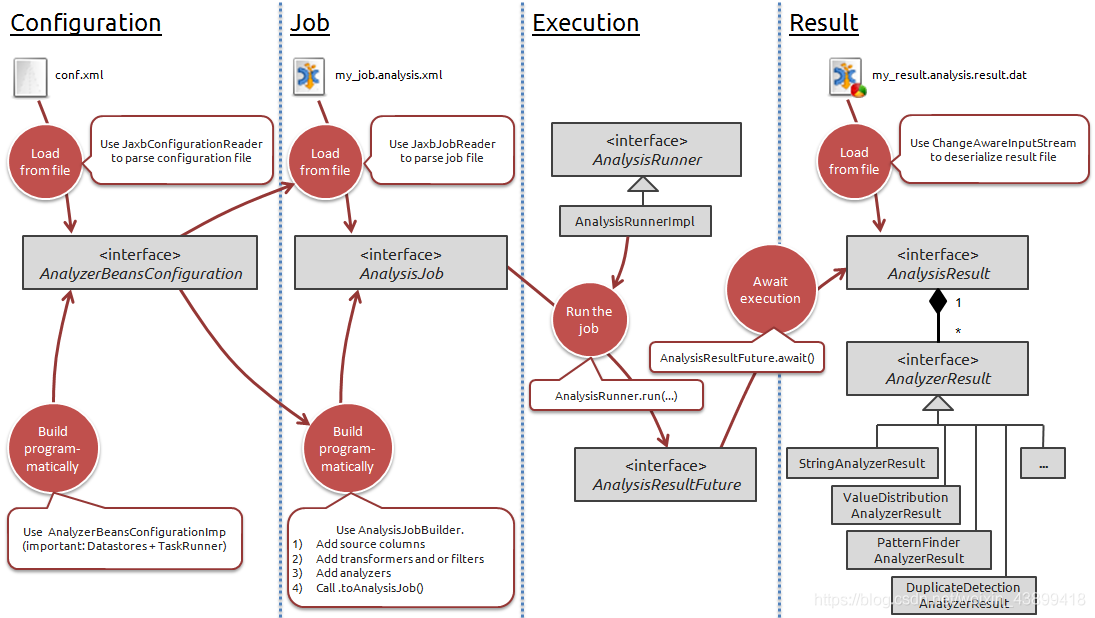
***图形优化(Graph optimization)*** ：虽然作业图（请参见 将组件连接在一起 章节）可能显示特定的以下顺序，但引擎可能在运行时对其进行某些优化。一些组件可能提供优化策略，包括更改源查询，以便更改已处理记录的数量（或内容）。显然，这是使用组件的副作用，该组件只会应用于不会影响作业中其他组件的情况。这个原理有时也被称为“下推优化”。

例如“Null check”过滤器：如果对源列应用Null检查，并且所有其他组件都需要Null或NOT-Null结果（显式或隐式），那么“Null check”过滤器可能会向源查询添加谓词，以过滤掉所有不相关的记录。有关这一原则的更多信息，请阅读Kasper Sørensen的博客文章“Push down query optimization in DataCleaner”。

## **Executing jobs through code**

**[18.1 步骤和选项概述](https://blog.csdn.net/weixin_43899418/article/details/114262002)**

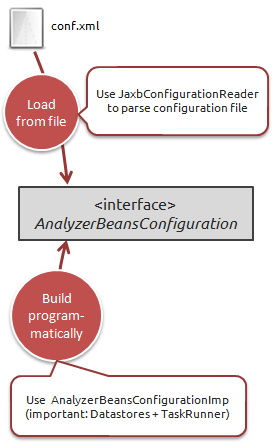
这个故事有两个变体-你想要什么样的配置选项？您希望以编程方式构建作业，还是将其作为一个.analysis.xml文件放在磁盘上的某个位置？您将对结果进行任何处理，还是作业本身包含所有必要的逻辑。  
下图描述了各种步骤和选项。在以下部分中，我们将逐一介绍图中的4个步骤/列：



**[18.2 第一步：配置](https://blog.csdn.net/weixin_43899418/article/details/114262124)**

DataCleaner的配置在类DataCleanerConfiguration（以前是“AnalyzerBeansConfiguration”类）中表示。您需要DataCleanerConfiguration作为大多数后续操作的先决条件。

获取DataCleanerConfiguration实例最简单、可能也是最方便的方法是从一个文件中加载（通常名为conf.xml格式（有关此文件格式的详细信息，请参阅配置文件一章）。要加载文件，请使用JaxbConfigurationReader类，如下所示：



1. InputStream inputStream = new FileInputStream("conf.xml");
2. JaxbConfigurationReader configurationReader = new JaxbConfigurationReader();
3. DataCleanerConfiguration configuration = configurationReader.read(inputStream);

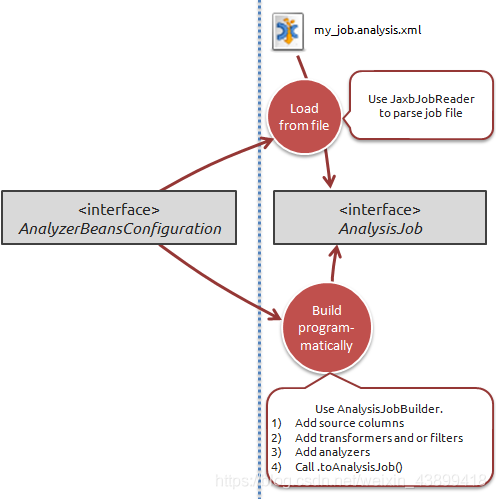
或者，您可以通过代码以编程方式构建配置。这通常比较麻烦，但在某些情况下，如果要动态构建配置或类似的配置，这也非常有用。  
下面是一个示例，我们使用2个示例数据存储和10个线程的线程池配置DataCleaner：

1. Datastore datastore1 = new CsvDatastore("my CSV file", "some\_data.csv");
2. boolean multipleConnections = true
3. Datastore datastore2 = new JdbcDatastore("my database",
4. "jdbc:vendor://localhost/database", "com.database.Driver",
5. "username", "password", multipleConnections);
6. DataCleanerConfigurationImpl configuration = new DataCleanerConfigurationImpl();
7. configuration = configuration.replace(new MultiThreadedTaskRunner(10));
8. configuration = configuration.replace(new DatastoreCatalogImpl(datastore1, datastore2));

无论采用哪种方式，我们现在都有一个变量名为“configuration”的DataCleanerConfiguration。然后我们可以继续定义要运行的作业。

**[18.3 第二步：作业](https://blog.csdn.net/weixin_43899418/article/details/114262296)**

与配置一样，我们可以选择从文件加载要运行的作业，或者以编程方式构建它。让我们从简单地从文件加载作业开始。我们需要使用JaxbJobReader类：

****

1. InputStream inputStream = new FileInputStream("my\_job.analysis.xml");
2. JaxbJobReader jobReader = new JaxbJobReader(configuration);
3. AnalysisJob analysisJob = jobReader.read(inputStream);

请注意，这是 ‘vanilla’ 的情况。您还可以使用JaxbJobReader读取有关作业的元数据，甚至可以 ‘as a template’ 读取作业，这使得可以用某些替换来实例化作业。有关如何在DataCleaner的桌面应用程序中使用此功能的示例，请参阅模板作业部分。

生成作业的另一种方法是以编程方式构建它。这是一个非常复杂的过程，根据你想做什么工作，这个过程会有很大的不同。但是API的设计使它尽可能简单。

提供了一些API，列出了一些重要的类：

AnalysisJobBuilder : 表示正在生成的可变作业。此生成器对象包含作业的源列，以及使用源列（或有时转换列）的所有组件。

TransformerComponentBuilder , FilterComponentBuilder , and AnalyzerComponentBuilder : 表示正在生成的作业的可变组件。它们可以有配置属性、过滤器要求、输入和输出列。

让我们看一个以编程方式构建作业的示例。为了确保我们不会错过重要的见解，我们将使用过滤器(filters)、转换器(transformers )和分析器(analyzers)使其成为一项相当重要的工作。该工作将包括：

* 这里有数据存储 ‘my database’ 中的三个源列：Name, Age and Company\_name。
* ‘Company\_name’ 为空的所有记录都将插入名为 ‘my CSV file’ 的数据存储中。在CSV文件中，这些列称为 ‘fullname’ 和 ‘age\_years’。
* ‘Company\_name’ 不为空的所有记录将 1）在数据库的另一个表中查找其工作地址，2）名称和工作地址将传递给 ‘Pattern finder’ 分析器。

1. Datastore myDatabase = configuration.getDatastoreCatalog().getDatastore("my database");
2. Datastore myCsvFile = configuration.getDatastoreCatalog().getDatastore("my CSV file");
3. AnalysisJobBuilder builder = new AnalysisJobBuilder(configuration);
4. builder.setDatastore(myDatabase);
5. builder.addSourceColumns("public.persons.Name","public.persons.Age","public.persons.Company\_name")
6. InputColumn<?> nameColumn = builder.getSourceColumnByName("Name");
7. InputColumn<?> ageColumn = builder.getSourceColumnByName("Age");
8. InputColumn<?> companyColumn = builder.getSourceColumnByName("Company\_name");
9. // add a filter to check for null 'company'
10. FilterComponentBuilder<NullCheckFilter> nullCheckBuilder = builder.addFilter(NullCheckFilter.class);
11. nullCheckBuilder.addInputColumn(companyColumn);
12. // add a InsertIntoTable analyzer to write the records without a company to the csv file
13. AnalyzerComponentBuilder<InsertIntoTableAnalyzer> insertBuilder = builder.addAnalyzer(InsertIntoTableAnalyzer.class);
14. insertBuilder.addInputColumns(nameColumn, ageColumn);
15. insertBuilder.setConfiguredProperty("Datastore", myCsvFile);
16. insertBuilder.setConfiguredProperty("Columns", new String[] {"fullname","age\_years"});
17. insertBuilder.setRequirement(nullCheckBuilder.getOutcome(NullCheckFilter.Category.NULL));
18. // add a lookup for the company working address
19. TransformerComponentBuilder<TableLookupTransformer> lookupBuilder =
20. builder.addTransformer(TableLookupTransformer.class);
21. lookupBuilder.addInputColumn(companyColumn);
22. lookupBuilder.setConfiguredProperty("Datastore", myDatabase);
23. lookupBuilder.setConfiguredProperty("Schema name", "public");
24. lookupBuilder.setConfiguredProperty("Table name", "companies");
25. lookupBuilder.setConfiguredProperty("Condition columns", new String[] {"name"});
26. lookupBuilder.setConfiguredProperty("Output columns", new String[] {"address"});
27. lookupBuilder.setRequirement(nullCheckBuilder.getOutcome(NullCheckFilter.Category.NOT\_NULL));
28. // reference the 'working address' column and give it a proper name
29. MutableInputColumn<?> addressColumn = lookupBuilder.getOutputColumns().get(0);
30. addressColumn.setName("Working address");
31. // add the Pattern finder analyzer
32. PatternFinder patternFinder = jobBuilder.addAnalyzer(PatternFinder.class);
33. patternFinder.addInputColumns(nameColumn, addressColumn);
34. // validate and produce to AnalysisJob
35. AnalysisJob analysisJob = jobBuilder.toAnalysisJob();

本例中需要注意的事项：

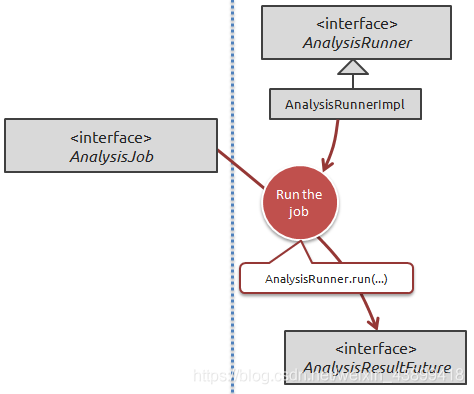
* 注意如何在后续组件上使用 .setRequirement(…) 方法设置过滤器要求。
* ‘Pattern finder’ 分析器上没有设置任何明确的筛选器要求。这不是必需的，因为它依赖于转换后的输入列(‘Working address’)，而输入列本身就有这个要求。DataCleaner将自动找出可传递的需求。
* 其中一个神奇的地方是如何正确设置组件的属性。我们可以看到我们调用了 .setConfiguredProperty(String,Object) ，但看不到如何找出作为参数传递的内容。有两个正确的方法来解决这个问题。。。

1. 您可以使用DataCleaner的命令行列出特定类型的所有组件，例如：
2. > DataCleaner-console.exe -list ANALYZERS
3. ...
4. name: Insert into table
5. - Consumes 2 named inputs
6. Input columns: Additional error log values (type: Object)
7. Input columns: Values (type: Object)
8. - Property: name=Column names, type=String, required=true
9. - Property: name=Datastore, type=UpdateableDatastore, required=true
10. ...
11. 您可以简单地在IDE中打开component类来检查它的@Configured属性。例如，如果我们看一下InsertIntoTableAnalyzer.java文件：
12. ...
13. @Inject
14. @Configured
15. @Description("Names of columns in the target table.")
16. @ColumnProperty
17. String[] columnNames;
18. @Inject
19. @Configured
20. @Description("Datastore to write to")
21. UpdateableDatastore datastore;
22. ...

从这些字段中，我们可以推断出将有两个配置的属性，‘Column names’ 和 ‘Datastore’。无论采用哪种方法，我们现在都有一个变量名为 ‘analysisJob’ 的 分析作业。然后我们可以继续实际执行作业。

**[18.4 第三步：执行](https://blog.csdn.net/weixin_43899418/article/details/114264558)**

执行作业是最简单的步骤之一，但显然除了 ‘vanilla’ 场景之外，还有其他可用的选项。运行作业的简单场景是使用普通AnalysisRunnerImpl类，如下所示：

****.

1. AnalysisRunner runner = new AnalysisRunnerImpl(configuration);
2. AnalysisResultFuture resultFuture = runner.run(analysisJob);

这将返回AnalysisResultFuture，在大多数情况下，它表示仍在运行的作业。您的应用程序可以继续在后台执行其他工作，也可以通过调用 .await() 来决定阻止。下面是一个处理结果的典型示例：

1. // block until the job has finished
2. resultFuture.await();
3. if (resultFuture.isSuccessful()) {
4. // do something with the successful result
5. handleResult(resultFuture);
6. } else {
7. List<Throwable> errors = resultFuture.getErrors();
8. for (Throable error : errors) {
9. logger.error("An error occurred while executing job", error);
10. }
11. // usually the first error that occurred is the culprit, so we'll throw that one
12. throw errors.get(0);
13. }

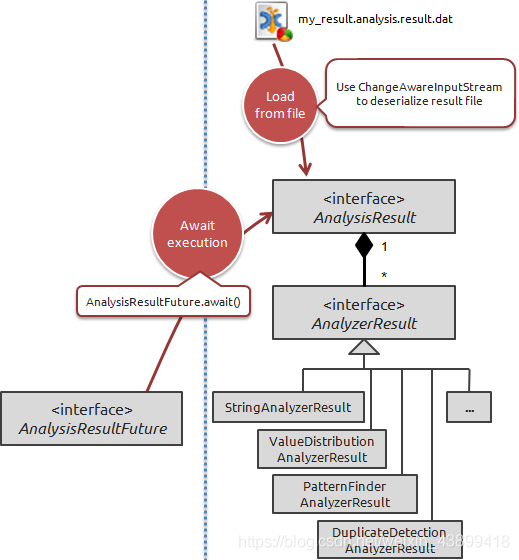
您可能会问在执行DataCleaner作业时会发生什么类型的错误？答案是它可以是很多东西，例如：

* 与源数据库或资源的连接可能会以某种方式失败。
* 作业中的某个组件可能引发意外异常。
* 其中一个组件可能会抛出异常，因为它的配置不完整或无效（尽管在大多数情况下，在构建AnalysisJob实例时会检测到这种情况）。
* 如果您正在将数据写入另一个数据存储，那么也可能由于依赖于数据存储的任何原因而失败。
* 如果你的工作是做一些愚蠢的事情，比如十亿个唯一id的值分布，那么你的内存就会用完。
* 现在假设您的作业已成功执行。现在我们来看看如何发布处理结果，以及如何将它们保存/加载到文件中。

**[18.5 第四步：结果](https://blog.csdn.net/weixin_43899418/article/details/114264777)**

很好，现在我们有了一个AnalysisResultFuture，我们已经确定它是成功的。我们能用它做什么？

job的每个分析器的结果都可以通过AnalysisResultFuture 实现的 ‘AnalysisResult’ 接口获得。请注意，analyzer结果类型彼此非常不同。例如，‘Insert into table’ 分析器生成 WriteDataResult ，而 ‘Pattern finder’ 生成 PatternFinderResult。让我们看看如何从中提取信息：



1. // demonstrate the the result
2. // future implements the AnalysisResult interface, which is sufficient
3. // for all the followin operations
4. AnalysisResult analysisResult = resultFuture;
5. List<AnalyzerResult> results = analysisResult.getResults();
6. for (AnalyzerResult result : results) {
7. if (result instanceof WriteDataResult) {
8. WriteDataResult writeDataResult = (WriteDataResult)result;
9. System.out.println("Inserted " + writeDataResult.getWrittenRowCount() + " records");
10. }
11. if (result instanceof PatternFinderResult) {
12. PatternFinderResult patternFinderResult = (PatternFinderResult)result;
13. int matches = patternFinderResult.getMatchCount("Aaaaa Aaaaa")
14. int total = patternFinderResult.getTotalCount();
15. System.out.println("There where " + matches + " matches out of " + total + " for our standard pattern.");
16. }
17. }

如您所见，如何处理结果在很大程度上取决于生成的结果类型。

对于结果的一般处理，包括可能出现的所有可能的结果扩展，DataCleaner使用了一个渲染器框架，该框架根据类型和优先级选择一个结果渲染器。如果您需要此类通用功能，请查看RendererBean、RendererFactory、Renderer和RenderingFormat类。

一个共同的要求是坚持下去。我们建议通过Java的序列化来实现这一点，因为分析结果是多态的，并且它的结构可能依赖于扩展。您还可以设计一个更“结构化”的持久性方案，但要注意，它需要相当多的稳定性，因为您在工作中添加了哪些分析器。

让我们看看如何使用Java序列化。但不幸的是AnalysisResultFuture不可序列化！但是，有一个类与“AnalysisResultFuture” 共享接口 “AnalysisResult”，该类是可序列化的，即“SimpleAnalysisResult”。让我们看看如何使用它，并将结果序列化为一个 .analysis.result.dat 文件（DataCleaner可以读取）：

1. // make the result serializeable
2. AnalysisResult analysisResult = resultFuture;
3. analysisResult = new SimpleAnalysisResult(analysisResult.getResultMap());
4. ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("my\_result.analysis.result.dat"));
5. oos.writeObject(analysisResult);
6. oos.close();

例如，现在让我们通过反序列化来加载文件。为此，我们需要使用 ChangeAwareObjectInputStream类，它确保对象的向后兼容反序列化：

1. ObjectInputStream ois = new ChangeAwareObjectInputStream(new FileInputStream("my\_result.analysis.result.dat"));
2. AnalysisResult analysisResult = (AnalysisResult) ois.readObject();

现在，结果已恢复，您可以进一步使用它。

## **Developer resources**

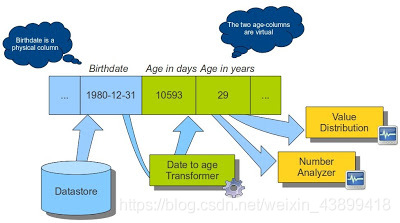
**[19.1 扩展开发教程](https://blog.csdn.net/weixin_43899418/article/details/114276799)**

对于那些从事开发扩展的人来说，有很多有用的资源。(插件/附加组件）到DataCleaner。为了帮助你，这里有一个有用的拓展开发。如果您认为此开发还不够，请告知我们：

* 教程1：开发转换器transformer
* 教程2：开发分析器analyzer
* 教程3：实现自定义数据存储
* [Javadoc: DataCleaner](http://eobjects.org/datacleaner/apidocs/current/)
* [Javadoc: MetaModel](http://metamodel.apache.org/apidocs/5.3.3/)

*****教程1：开发转换器transformer*****

**在这篇博客文章中，我将演示DataCleaner的Java Api来创建转换器，即用于基于数据集的现有值进行转换/可变/标记化/生成新值的组件。您将需要Java编程技能来学习本教程。我发现解释这个过程最简单的方法就是运行一个例子。所以这里是我的例子：我想把人的出生日期（表示为Date 字段）转换为年龄字段（表示为number 字段）。场景描述如下：**

**********

**转换后，我将能够独立处理年龄字段，例如，通过数字分析、值分布或应用一些依赖于年龄的业务规则。构建转换器transformer 的要求如下：**

* **该类必须要实现Transformer接口**
* **该类上面必须用 (javax.inject)@Named注解。该注解接受一个参数：转换器的可读名称。我们将这样注释：@Named（“Date to age”）**
* **为了从传入字段中读取数据，我们需要注入一个 InputColumn<E> 实例（或者一个数组），其中<E>是传入字段的数据类型。为了注入，我们使用@Configured注释。在我们的示例中，这转换为：@Configured InputColumn<Date> dateColumn；**

**在这些步骤之后，我们的代码将如下所示：**

1. @Named("Date to age")//就是转换器的名字
2. public class DateToAgeTransformer implements Transformer {
3. @Configured
4. InputColumn<Date> dateColumn;//从数据表中传入的字段数据
5. @Override
6. public OutputColumns getOutputColumns() {
7. // TODO
8. return null;
9. }
10. @Override
11. public Object[] transform(InputRow inputRow) {
12. // TODO
13. return null;
14. }
15. }

**如我们所见，Transformer接口定义了两种方法，我们需要实现它们。**

* *****getOutputColumns():***** 框架调用此方法以确定转换器将生成哪些虚拟列。在我们的例子中，它非常简单：transformer为age创建虚拟列（以天为单位和以年为单位，只是为了使它更灵活）。因此，方法主体应该是：

1. return new OutputColumns(Integer.class, "Age in days", "Age in years");

* *****transform(InputRow):***** 对于每一行要转换的值，都将调用此方法。方法的返回类型是表示行的新值的对象数组。返回数组的索引应与输出列匹配，即索引0表示“以天为单位的年限”，索引1表示“以年为单位的年限”。让我们看一下方法实现：

1. Integer[] result = new Integer[2];
2. Date date = inputRow.getValue(dateColumn);
3. if (date != null) {
4. long diffMillis = today.getTime() - date.getTime();
5. int diffDays = (int) (diffMillis / (1000 \* 60 \* 60 \* 24));
6. result[0] = diffDays;
7. // use Joda time to easily calculate the diff in years
8. int diffYears = Years.yearsBetween(new DateTime(date), new DateTime(today)).getYears();
9. result[1] = diffYears;
10. }
11. return result;

**当然，在编写本教程的过程中，我并没有在没有嵌入代码的情况下完成所有的工作，这样您就可以实际使用它了。这里提供了 ”Date to age” 转换器的代码，这里还有一个单元测试，可以用来演示如何对转换器进行单元测试。我希望你们中的一些人参与开发转换器，让我知道结果如何。在我的下一篇博客文章中，我将解释如何构建分析器，这是为DataCleaner开发组件时显而易见的下一步。这里还有其他一些转换器的好例子：**

* **转换为日期转换器，它将尝试将任何值转换为日期。这对于我刚刚在本教程中解释的转换器来说可能很有用。换句话说：如果要转换的出生日期存储在基于字符串的字段中，那么这两个转换器可能需要链接。**
* **标记器转换器，因为它具有基于用户配置的灵活数量的输出列。请注意该例子中用到的的 @Configured Integer numTokens变量。**

*****教程2：开发分析器analyzer*****

**上文已经写了如何开发一个转换器。现在是时候来看看如何开发一个分析器了，它是一个用来消耗数据并将其转换成一个可供人阅读的结果的组件，希望它是有用的。Java API的Javadocs位于这里。AnalyzerBeans 中已经有很多不同的分析器，当您决定开发自己的分析器时，不妨看看这些分析器：**

* **对于典型的度量，有数字分析器(Number analyzer)和字符串分析器(String analyzer)之类的分析器。这些分析器计算这些数据类型的标准化度量。**
* **有一个值分布分析器(Value distribution)很有趣，因为它使用一个后备数据库（使用 @Provided注释）来计算唯一值，如果值超过了可用内存量。**
* **日期间隔分析器(Date gap analyzer)也是一个很好的例子，因为它命名了输入列，用于构建从日期到日期的时间线。**
* **模式查找器(PatternFinder)分析器，你可以在我以前的一篇博文中读到更多。**

**让我们从一个简单的例子开始。假设您想构建一个非常简单的分析器，它使用基于日期date 或时间time 的值，并根据星期几确定值的分布（即，如何将值的分布分组在星期一、星期二、星期三等）。虽然这是一个相当幼稚的分析器示例，但它也能很好地工作作为一个示例。我们将从构建分析器(analyzer)的要求开始：**

* **您需要定义一个实现 Analyzer<R>的类。泛型 ‘R’ 参数定义了分析器的结果类型。我们可以重用内置的结果类型，也可以编写自己的结果类型。**
* **该类需要用@AnalyzerBean注解。此注解接受一个参数：分析器的显示名称。**
* **您需要使用@Configured注释注入一个或多个InputColumn<E>，以便使用传入的数据。<E>参数定义感兴趣的数据类型，它还用于确定分析器支持哪些类型的数据类型。在本例中，我们将使用Date作为InputColumn类型，因为我们希望分析器使用日期值。**

**因此我们的类是按照上述要求创建的：**

1. @AnalyzerBean("Average date analyzer")
2. public class AverageDateAnalyzer implements Analyzer<CrosstabResult> {
3. @Configured
4. InputColumn<Date> dateColumn;
5. public void run(InputRow row, int distinctCount) { ... }
6. public CrosstabResult getResult() { ... }
7. }

**请注意，我们使用的是内置的结果类型CrosstabResult，它表示由维度交叉表组成的结果。我们可以使用其他内置的结果类型，也可以创建自己的结果类型，唯一的要求是它得实现AnalyzerResult接口。**

**分析器的其余部分应该是“熟知的Java”，但当然也要使用AnalyzerBeans中提供的API。这些事我以前都解释过，但我会再解释一遍。所以现在来考虑如何实现具体的分析器逻辑。我们将使用常规 map 来保存分布值。我们将把工作日的数字映射到这个 map 上并计数。但我们需要为正在分析的每一列保留一个计数，因此它将是一个嵌套map：**

1. private Map<InputColumn<Date>, Map<Integer, Integer>> distributionMap;

**要初始化映射，我们需要先注入InputColumn。相反，我们可以用在方法上添加 @Initialize 注解，这将使AnalyzerBeans在bean被正确初始化时调用该方法。**

1. @Initialize
2. public void init() {
3. distributionMap = new HashMap<InputColumn<Date>, Map<Integer, Integer>>();
4. for (InputColumn<Date> col : dateColumns) {
5. Map<Integer, Integer> countMap = new HashMap<Integer, Integer>(7);
6. for (int i = Calendar.SUNDAY; i <= Calendar.SATURDAY; i++) {
7. // put a count of 0 for each day of the week
8. countMap.put(i, 0);
9. }
10. distributionMap.put(col, countMap);
11. }
12. }

**现在 map 已经初始化，我们可以继续实现run（…）方法：**

1. @Override
2. public void run(InputRow row, int distinctCount) {
3. for (InputColumn<Date> col : dateColumns) {
4. Date value = row.getValue(col);
5. if (value != null) {
6. Calendar c = Calendar.getInstance();
7. c.setTime(value);
8. int dayOfWeek = c.get(Calendar.DAY\_OF\_WEEK);
9. Map<Integer, Integer> countMap = distributionMap.get(col);
10. int count = countMap.get(dayOfWeek);
11. count += distinctCount;
12. countMap.put(dayOfWeek, count);
13. }
14. }
15. }

**这应该相当于“Java日常使用”。如果您是一名经验丰富的Java开发人员，那么您唯一应该了解的是使用InputColumns作为限定符从InputRow提取值的方法：**

1. Date value = row.getValue(col);

请注意，value变量具有日期Date 类型。AnalyzerBeans API 在很大程度上利用了类型安全性。由于注入的InputColumn被定义为日期列，这意味着我们可以安全大胆地假设传入行中的值一定是日期类型。此外，日期列将用于验证AnalyzerBeans job 的配置，如果用户尝试使用非日期列配置该特定Analyzer，则会向用户发送早期错误消息。现在开始创建结果。如前所述，我们将为此使用交叉表结果CrosstabResult 。交叉表结果是一种非常动态的结果类型，可以用于许多目的。它类似于DataCleaners结果矩阵，但增加了一些特性。下面是我们如何该分析器的结果交叉表：

1. @Override
2. public CrosstabResult getResult() {
3. CrosstabDimension columnDimension = new CrosstabDimension("Column");
4. CrosstabDimension weekdayDimension = new CrosstabDimension("Weekday");
5. weekdayDimension.addCategory("Sunday").addCategory("Monday")
6. .addCategory("Tuesday").addCategory("Wednesday").addCategory("Thursday")
7. .addCategory("Friday").addCategory("Saturday");
8. Crosstab crosstab = new Crosstab(Integer.class, columnDimension, weekdayDimension);
9. for (InputColumn col : dateColumns) {
10. columnDimension.addCategory(col.getName());
11. CrosstabNavigator nav = crosstab.where(columnDimension, col.getName());
12. Map countMap = distributionMap.get(col);
13. nav.where(weekdayDimension, "Sunday").put(countMap.get(Calendar.SUNDAY));
14. nav.where(weekdayDimension, "Monday").put(countMap.get(Calendar.MONDAY));
15. nav.where(weekdayDimension, "Tuesday").put(countMap.get(Calendar.TUESDAY));
16. nav.where(weekdayDimension, "Wednesday").put(countMap.get(Calendar.WEDNESDAY));
17. nav.where(weekdayDimension, "Thursday").put(countMap.get(Calendar.THURSDAY));
18. nav.where(weekdayDimension, "Friday").put(countMap.get(Calendar.FRIDAY));
19. nav.where(weekdayDimension, "Saturday").put(countMap.get(Calendar.SATURDAY));
20. }
21. return new CrosstabResult(getClass(), crosstab);
22. }

现在我们结束了。你可以看看这里的最终结果。当我用三列中的小数据样本运行此分析器时，结果如下所示：

1. Order date Shipment date Delivery date
2. Sunday 0 0 0
3. Monday 2 0 1
4. Tuesday 0 2 1
5. Wednesday 0 0 0
6. Thursday 1 0 0
7. Friday 1 1 2
8. Saturday 0 1 0

*****教程3：实现自定义数据存储*****

DataCleaner的超级用户、合作伙伴和开发人员经常问我一个问题：如何在DataCleaner中为我的system/file-format XYZ构建自定义数据存储？最近，我处理这个问题是为了在即将到来的与Pentaho-Kettle的集成中使用，对于一个拥有自己开发的数据库代理系统的人工接口客户，就在今天，当它在DataCleaner论坛上被询问时。在这篇博文中，我将指导您完成这个过程，这需要一些基本的Java编程技能，但如果这一点已经到位的话，它并不十分复杂。

首先，我应该说（对于那些喜欢“只看代码”的人来说），在DataCleaner的源代码已经有了一个简单示例( sample extension)。看一看这个 org.eobjects.datacleaner.sample.SampleDatastore 类。一旦您阅读、理解并编译了Java代码，您所需要做的就是在DataCleaner中的conf.xml文件 注册数据存储 （在< datastore-catalog >便签中）：

1. <custom-datastore class-name="org.eobjects.datacleaner.sample.SampleDatastore">
2. <property name="Name" value="My datastore" />
3. </custom-datastore>

首先，DataCleaner中的数据存储需要实现 Datastore 接口。但是我建议使用名为UsageAwareDatastore的抽象实现，而不是直接实现接口。这个抽象实现处理对数据存储的并发访问，重用现有连接等等。在扩展UsageAwareDatastore类时，您仍然需要提供的主要是 createDatastoreConnection() 方法，该方法在请求新连接时被调用。让我们看看新建一个数据存储实现是什么样子的：

1. public class ExampleDatastore extends UsageAwareDatastore<DataContext> {
2. private static final long serialVersionUID = 1L;
3. public ExampleDatastore() {
4. super("My datastore");
5. }
6. @Override
7. protected UsageAwareDatastoreConnection createDatastoreConnection() {
8. // TODO Auto-generated method stub
9. return null;
10. }
11. @Override
12. public PerformanceCharacteristics getPerformanceCharacteristics() {
13. // TODO Auto-generated method stub
14. return null;
15. }
16. }

注意，我已经创建了一个无参数构造函数。这对于自定义数据存储是必需的，因为数据存储将由DataCleaner实例化。稍后我们将重点讨论如何调整名称(“My datastore”)。首先，我们来看看两个未实现的方法：

* **createDatastoreConnection() : c用于创建新连接。DataCleaner构建在元模型框架之上，用于数据访问。您需要返回一个新的DatastoreConnectionImpl(…)。该类接受一个重要参数，即MetaModel DataContext implementation。通常在给定的配置情况下，已经有了一个可以使用的DataContext，例如JdbcDataContext、CsvDataContext、ExcelDataContext、MongoDbDatacontext或其他什么。**
* **getPerformanceCharacteristics(): DataCleaner在执行作业时使用该方法来确定查询计划。通常只返回一个新的PerformanceCharacteristics(false);，阅读javadoc了解更多信息**

到目前为止，您应该能够实现一个定制的数据存储，这有望满足您的基本需求。但是也许你想用不同的文件、不同的主机名等重用datastore类，换句话说：也许你想让你的用户定义datastore的某些属性。

拯救方法是@Configured注解，它是DataCleaner中广泛使用的注解。它允许您在类中注释应该由用户配置的字段。字段的类型可以是字符串、整数、文件等。让我们看看如何公开特有连接的属性：

1. public class ExampleDatastore extends UsageAwareDatastore<DataContext> {
2. // ...
3. @Configured
4. String datastoreName;
5. @Configured
6. String hostname;
7. @Configured
8. Integer port;
9. @Configured
10. String systemId;
11. // ...
12. }

以及通常如何使用它们来实现方法：

1. public class ExampleDatastore extends UsageAwareDatastore<DataContext> {
2. // ...
3. @Override
4. public String getName() {
5. return datastoreName;
6. }
7. @Override
8. protected UsageAwareDatastoreConnection createDatastoreConnection() {
9. DataContext dataContext = createDataContext(hostname, port, systemId);
10. return new DatastoreConnectionImpl(dataContext, this);
11. }
12. }

如果我想使用上面的参数配置配出一个数据存储，我可以在我的conf.xml格式文件如下：

1. <custom-datastore class-name="foo.bar.ExampleDatastore">
2. <property name="Datastore name" value="My datastore" />
3. <property name="Hostname" value="localhost" />
4. <property name="Port" value="1234" />
5. <property name="System id" value="foobar" />
6. </custom-datastore>

请注意，属性的名称是通过反转Java使用的camelCase表示法推断出来的，这样 “datastoreName” 就变成了 “Datastore name” ，以此类推。或者，您可以在 @Configured 注解中提供显式名称。

我希望这个介绍教程对你有意义。我再次敦促您查看示例Sample DataCleaner extension，它还包括构建设置（基于Maven）和自定义元模型DataContext实现。 **[19.2 Building DataCleaner](https://blog.csdn.net/weixin_43899418/article/details/114289488)**

从GitHub获取DataCleaner的源代码：

1. > git clone https://github.com/datacleaner/DataCleaner.git DataCleaner

Build the projects:

1. > cd DataCleaner
2. > mvn clean install

Run DataCleaner

1. > cd desktop/ui/target
2. > java -jar DataCleaner-desktop-ui-[version].jar

## **Extension packaging**

**[20.1 组件上的注解](https://blog.csdn.net/weixin_43899418/article/details/114289664)**

DataCleaner中扩展发现背后的主要原则是带注解的类。大部分组件都应具有以下注解之一：

* **@java.inject.Named – 对于实现Transformer、Filter或Analyzer接口的类。**
* **@org.datacleaner.api.RendererBean – 用于实现 Renderer 接口的类。**

有关用法的详细信息，请参阅接口的javadoc文档。

**[20.2 单个jar文件](https://blog.csdn.net/weixin_43899418/article/details/114289826)**

扩展名必须由单个JAR文件组成。如果您的依赖项不是DataCleaner发行版提供的库，那么您需要将这些依赖项打包到您自己的JAR文件中。如果您在构建中使用Maven，Maven程序集插件可以使用POM中的以下代码片段轻松提供此功能：

1. <build>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-assembly-plugin</artifactId>
6. <version>2.2.1</version>
7. <configuration>
8. <descriptorRefs>
9. <descriptorRef>jar-with-dependencies</descriptorRef>
10. </descriptorRefs>
11. </configuration>
12. </plugin>
13. </plugins>
14. </build>

**[20.3 扩展元数据XML](https://blog.csdn.net/weixin_43899418/article/details/114289926)**

为了改善体验，您可以选择在一个XML文件中包含关于扩展名的元数据metadata ，并将其捆绑在JAR文件中。  
扩展元数据文件的名称必须是 datacleaner-extension.xml 并放置在JAR文件的根目录中。以下是文件示例：

1. <extension xmlns="http://eobjects.org/datacleaner/extension/1.0">
2. <name>My extension</name>
3. <package>path.to.extension</package>
4. <description>This is an example extension. I should put a short description here.</description>
5. <icon>path/to/extension/ExtensionIcon.png</icon>
6. <author>John Doe</author>
7. <url>https://datacleaner.org/extensions</url>
8. <version>1.0</version>
9. </extension>

此元数据的附加值是DataCleaner可以向用户公开此信息，还可以使用它来管理扩展名的更新等。但是，元数据文件是完全可选的。 **[20.4 组件图标](https://blog.csdn.net/weixin_43899418/article/details/114290042)**

如果您想为您的组件添加一个自定义图标（例如，一个转换器或分析器），您需要将图标放置为一个PNG图像，其名称与组件的完全分类类名相同。

例如：如果组件类名为"com.company.ext.MyAnalyzer"，则此组件的图标应位于 “/com/company/ext/MyAnalyzer.png” 在扩展JAR文件中。

类似地，如果绑定自己的ComponentCategory实现（定义DataCleaner中的菜单组），则可以通过添加一个PNG文件来定义这些图标，该文件具有与ComponentCategory类名相对应的完全分类的文件名。

## **Embedding DataCleaner**

**[21 嵌入DataCleaner](https://blog.csdn.net/weixin_43899418/article/details/114290137)**

可以将DataCleaner嵌入到其他Java应用程序中。这允许一种简单的方法来添加数据质量分析（DQA）和数据分析功能，作为对正在构建的应用程序的补充。嵌入DataCleaner最简单的方法就是执行DataCleaner的主可执行文件所做的操作—用默认参数实例化引导类：

1. BootstrapOptions bootstrapOptions = new DefaultBootstrapOptions(args);
2. Bootstrap bootstrap = new Bootstrap(bootstrapOptions);
3. bootstrap.run();

要进一步自定义，请添加自己的BootstrapOptions类实现。嵌入DataCleaner的主要场景是以所谓的 “single datastore mode” 运行应用程序。这可以通过实现BootstrapOptions并为getSingleDatastore()方法提供非空值来实现。