Kettle Notes

`notes for blog`

Kettle简介

http://kettle.pentaho.com/
http://wiki.pentaho.com/
http://infocenter.pentaho.com
Pentaho Data Integration(Kettle)是一款开源的 ETL(Extract Transformation Load) 工具，用来完成数据的抽取，清洗、转换和加载等数据处理方面的工作。
竞争产品：Talend，Informatica

ETL

- 抽取(Extract)：一般抽取过程需要连接到不同的数据源，以便为随后的步骤提供数据。这一部分看上去简单而琐碎，实际上它是 ETL 解决方案的成功实施的一个主要障碍。
- 转换(Transform)：任何对数据的处理过程都是转换。这些处理过程通常包括(但不限于)下面一些操作：
    - 移动数据
    - 根据规则验证数据
    - 数据内容和数据结构的修改
    - 将多个数据源的数据集成
    - 根据处理后的数据计算派生值和聚集值
- 加载(Load)：将数据加载到目标系统的所有操作。

概念扩展：ELT，EII(Enterprise information Integration)/Data Federation

Kettle基本使用

子程序：
Spoon.bat： 图形界面方式启动作业和转换设计器。
Pan.bat： 命令行方式执行转换。
Kitchen.bat： 命令行方式执行作业。
Carte.bat： 启动web服务，用于Kettle的远程运行或集群运行。
Encr.bat： 密码加密

Kettle 的 Spoon 设计器用来设计转换(Transformation) 和作业(Job)。
转换主要是针对数据的各种处理，一个转换里可以包含多个步骤(Step)。
作业是比转换更高一级的处理流程，一个作业里包括多个作业项(Job Entry)，一个作业项代表了一项工作，转换也是一个作业项。

保存作业
用户通过 Spoon创建的转换、作业、数据库连接等可以保存在资源库和XML文件中。

- 转换文件以 ktr 为扩展名，作业文件以 kjb 为扩展名
- 资源库可以是各种常见的数据库。可以在 Spoon 中自动创建资源库，资源库默认用户名和密码是 admin/admin。

Kettle资源库

元数据

元数据的通用概念：“描述性数据”或“数据的数据”
ETL 的元数据：描述 ETL 要执行的任务
在Kettle里元数据的存储方式：

- 资源库
- XML文件
  .ktr 转换文件的XML的根节点必须是 < transformation>
  .kjb 作业XML的根节点是< job>

资源库

- 数据库资源库
    - 把 Kettle 的元数据串行化到数据库中，如R_TRANSFORMATION 表保存了Kettle 转换的名称、描述等属性。
    - 在Spoon 里创建和升级数据库资源库
- 文件资源库：
  在文件的基础上的封装，实现了org.pentaho.di.repository.Repository 接口。是Kettle 4.0 以后版本里增加的资源库类型。

不使用资源库： 直接保存为ktr或kjb文件。

资源库操作：新建，导入，导出

数据库资源库的缺点：

- 不能存储转换或作业的多个版本。
- 严重依赖于数据库的锁机制来防止工作丢失。

- 没有考虑到团队开发，开发人员不能锁住某个作业自己开发。

文件资源库的缺点：

- 对象(如转换、作业、数据库连接等对象)之间的关联关系难以处理，所以删除、重命名等操作会比较麻烦。
- 没有版本历史。
- 难以进行团队开发。

不使用资源库：使用svn进行文件版本控制。

资源库管理
ETL 开发的几个阶段：开发、测试、确认、发布
各阶段对应的资源库： 开发资源库、测试(确认)资源库、发布资源
各阶段推进

1. 从开发资源库到测试资源库：注意命名规则。 由一个人统一发布，避免冲突。两种移植方法：断开重连、导出/导入。
2. 从测试(确认)资源库到发布资源库：导出/导入

不使用资源库：SVN 版本控制，测试打tag，发布branch

参数化
为什么要参数化： 在资源库之间移植作业时，因为各个阶段的环境不一样，在作业里使用的数据库连接等元数据不能硬编码。
参数化的几个方法：

- kettle.properties，文件位于 java 的user.home目录下
- 自定义properties文件，通过属性文件输入步骤读取。
- 使用参数表
  参数表的结构
  Environment：Dev/Test
  parameter_name：host_name
  parameter_value：localhost
  valid_from
  valid_to

Kettle运行方式
Spoon
本地：在本地执行
远程：在远程服务器执行，需要远程服务器执行 Carte 。

- Carte 是内嵌 Jetty 的 http server。
- Carte执行命令`carte localhost 8080`
  集群：在集群上执行，需要转换里的某个步骤事先设置为集
  群方式运行。

  Pan
  参数格式 /参数名：值 或 -参数名=值

  1. 执行test.ktr 文件 日志保存在D：\log.txt 中， 默认日志级别是Basic
     `Pan /file: D: \AppProjects\nxkh\test.ktr /logfile: D: \log.txt`
  2. 执行test.ktr 文件 日志保存在D：\log.txt 中， 日志级别是Rowlevel
     `Pan /file: D: \AppProjects\nxkh\test.ktr /logfile: D: \log.txt /level:  Rowlevel`
  3. 导出一个 job 文件，以及该 job 文件依赖的转换及其他资源
     `kitchen /file: c: /job1.kjb /export: c: /a.zip`
  4. 直接执行一个导出的 zip 文件
     `Kitchen.bat /file: "zip: file: ///c: /a.zip!job1.kjb"`

日志级别说明：
Error： 只记录错误信息
Nothing： 不记录任何信息，执行效率最高
Minimal： 记录最少的信息
Basic： 记录基本信息
Detailed： 记录详细信息
Debug： 记录调试信息
Rowlevel： 转换过程中的每一行都记录下来，日志最详细，执行效率最低

文件日志

- 命令行的 /logfile 参数，将日志导出到指定的文件中。
- Linux 管道符将屏幕输出转成日志。
- 默认的日志文件保存在 java.io.tmpdir 目录下，文件名类似spoon_xxx.log
- 为便于调试，Spoon 里的有日志窗口，内容和日志文件相同。

内存中的日志太多，可能会引起 OutOfMemory 的错误

- Spoon 运行时设置日志缓存大小

Spoon的“选项”对话框里设置 “日志窗口的最大行数” 、“内存中保留日志时长” 、“日志视图的最大行数” 。
- kettle.properties 文件里设置 KETTLE_MAX_LOG_SIZE_IN_LINE 变量，KETTLE_MAX_LOG_TIMEOUT_IN_MINUTES变量

转换有四个日志表：

- 转换日志表
- 步骤日志表
- 性能日志表
- 日志通道日志表

作业有三个日志表：

- 作业日志表
- 作业项日志表
- 日志通道日志表

基本组件

输入步骤

- 生成记录/自定义常量
- 获取系统信息
    - 转换开始时间
    - 关键时间点信息
    - 最多十个命令行参数
    - 主机名/ip/进程号/虚拟机内存等
    - 上一个作业的完成情况参数

- 表输入
    - 变量？ 和 ${var}
    - 数据类型

java.sql.Types	带符号	Kettle type	length	precision	最大值
CHAR VARCHAR LONGVARCHAR		TYPE_STRING	Rm 获取		
CLOB		TYPE_STRING	9999999		
BIGINT	是	TYPE_INTEGER	15	0	9,223,372,036 ,854,775,807
BIGINT	否	TYPE_NUMBER	16	0	18,446,744,07 3,709,551,615
INTEGER		TYPE_INTEGER	9	0	2,147,483,647
SMALLINT		TYPE_INTEGER	4	0	32,767
TINYINT		TYPE_INTEGER	2	0	127

java.sql.Types	Kettle type	length	precision	最大值
DECIMAL DOUBLE FLOAT REAL NUMERIC	TYPE_NUMBER TYPE_BIGNUMBER	Rm 获取	Rm 获取	
DATE TIME TIMESTAMP	TYPE_DATE	-	-	
BOOLEAN BIT	TYPE_BOOLEAN	-	-	
BINARY BLOB VARBINARY LONGVARBINARY	TYPE_BINARY TYPE_STRING	Rm 获取	Rm 获取	

- 文本文件输入
- XML 文件输入
    - DOM方式
      使用简单，但是不能使用大XML文件，占用内存
    - 流方式
      可处理大的XML 文件，更灵活高效，但是使用较复杂

- Json输入
- 其他输入步骤

输出步骤

数据库表：

- 表输出
- 更新，删除，插入/更新
- 批量加载
- 数据同步
    - 常用同步方法： 时间戳，比较、触发器、日志
    - 基于比较的同步方式：根据一个flag字段执行相应的插入/更新/删除操作

文件：

- SQL 文件输出
- 文本文件输出
- XML 输出
- Excel Output/Excel Writer
  Excel输出有65535的限制，2007没有该限制。

其他：

- 报表输出
  事先定义好报表模板

转换步骤

- 增加新的列
    1. 增加常量列
    2. 增加序列列
    3. 增加分组序列列
    4. 增加校验列
    5. 增加XML 列
    6. 计算器

- 字符串处理
    - 拆分
        1. 按位置拆分（剪切字符串）
        2. 按标志字符一列拆分成多列（拆分字段）
        3. 按标志字符一列拆分成多行

    - 合并

1. 多列合并为一列（concat fields）
2. 多行合并为一行（分组）

- 替换
  1. 值映射
  2. 使用正则表达式（字符串替换）

- 其他转换
  1. 使用常量替换一个字段的值（将字段值设置为常量）
  2. 使用一个字段替换另一个字段的值（将字段值设置为其他字段）
  3. 字符串操作
     - Escape XML、Escape SQL、 Escape HTML
     - 保留/移去字符串里的数字
     - 移去字符串里的特殊字符
     - 补充指定长度的字符

- 行列变换
  多行的窄表：规范，便于扩展
  多列的宽表：便于查询
  - 行转列（反正规化 Denormaliser）
    把多行的窄表转成多列的宽表。
    前提：需要事先按照分组字段排序
  - 列转行（正规化 Normaliser ）
    把多列的宽表转成多行的窄表。

- 排序/排重/字段选择
  - 排序，快排算法
  - 排重
    1. Unique Row： 需要事先排序
    2. Unique Row(Hash)： 不排序，速度快，占内存

- 其他转换步骤
  - 闭合距离：计算树状结构表中父子节点的距离，提高Mondrian OLAP 分析的性能。
  - XSL转换
  - 数值范围

流程步骤

- 运行阶段
  - ETL元数据注入

- 过滤数据
  - Switch/Case，一到多，支持日期、数值、字符串类型比较
  - 过滤记录，一到二，支持日期、数值、字符串类型比较，自定义可嵌套的表达式
  - 根据java表达式过滤记录，一到二

- 处理不确定的行数
  - 检测空流，有则不通过，没有则生成一个空行
  - 识别流的最后一行
  - 阻塞数据，除了最后一条，其它的数据行都不能过去

- 多来源数据行合并（要求列名、列数、列类型相同）
  - 空操作，多个来源，以自然顺序合并
  - 追加流，只能两个来源，指定顺序合并
  - 数据流优先级排序，多个来源，指定顺序合并

- 终点
  - 空操作，垃圾箱
  - 终止，如果有数据到此步骤，转换会被强行停止，并报错误
  - 复制记录到结果，暂时保留在内存里的数据，供以后的转换使用
  - 设置变量，把字段值设置为变量，变量供以后转换使用

- 其他
  - 阻塞数据直到步骤都完成，注意依赖死锁问题
  - 克隆行
  - 延迟行

- 单线程

应用步骤

- NULL值处理
  - 替换NULL值
  - 设置值为NULL

- 启动其他程序

        - 启动一个进程，本地程序，使用Java的Runtime
        - 运行ssh命令，本地和远程都是linux，提前设置ssh-keygen

- 日志
    - 写日志
    - 发送信息至系统日志（apache syslog）

- 文件处理
    - 改变文件编码
    - 处理文件（移动、删除、复制等）

- 邮件

查询步骤

- 流查询
    - 流查询
        1. 字典数据完全加载到内存后，在内存中查询，速度快，占内存。
        2. 只支持“等于”的查询
        3. 如果匹配上多条，只保留最后一条。
        4. 如果没有匹配上，新增的字段值为NULL。
        5. 如果字典key和要查询的value都是integer，可以选中Key and value are exactly one integer field，节省内存
        6. Use sorted list：当比较的字符串比较长的，使用 hash方式，可以节省内存。

    - 模糊匹配
        1. 只支持单列的查询
        2. 匹配相似度最大的字符串
        3. 自定义匹配的取值范围
        4. 支持的模糊匹配算法： Jaro， Jaro Winkler， Levenshtein等

- 数据库查询
    - 数据库查询
        1. 只返回一行
        2. 如果有多行结果：只返回第一行或失败
        3. 对数据流里的每条记录都要做一次数据库查询，效率低
        4. 数据库查询(加载所有数据到缓存) = 表输入+流查询

    - 数据库连接
        1. 可以自定义参数位置
        2. 参数也可以输出

    - 调用存储过程
        1. Oralce 存储过程需要带参数
        2. 支持in参数， out参数

- Web查询
    - HTTP客户端，使用 GET 的方式提交请求，获得返回的页面内容。可以从前面步骤获得 URL，参数名，参数值。
    - HTTP Post，使用 POST 方式提交请求，获得返回的页面内容。Request entity field：保存文件名，可以提交文件
    - Web服务查询，通过 WebService 获取数据
    - Rest客户端，通过Restful 获取数据

连接步骤

- 记录集连接
    - 记录集连接，两个记录集做左连接、右连接，内连接、外连接
    - 记录关联（笛卡尔输出），两个记录集做笛卡尔乘积，如果发现速度慢，调整main step

- 排序合并
    - 排序合并，多个排好序的数据流，再排序。用于：
        1. 多份sort copy后的排序。
        2. 集群的master 节点，将多个slave 节点的排好序的数据再排序。

    - 数据比较，通过标识字段，比较两个数据源数据的变化情况。 标识字段的四种状态：new， identical， deleted， updated

- XML连接
    1. 用来构造自定义XML字符串
    2. 通过XML Path连接两个XML格式数据

映射步骤

- 映射输入接口
- 映射输出接口
- 映射（子转换）

集群

Kettle 集群是一个分布式的运行环境，由一个主节点和多个子节点构成。主节点调度在子节点上处理不同的数据行，子节点把处理后的结果再提交到主节点。

1. 在转换里， 需要在集群上运行的步骤上右键选择“集群” 。
2. 主节点和子节点上启动carte服务。

注意：

1. 一般只有大量消耗内存/CPU/IO等资源的步骤使用集群方式运行，如排序、表输出步骤。
2. 因为数据本身要在主从节点中传递，所以集群本身也会消耗 IO资源。

参数和变量

- 参数（命名参数，位置参数）
- 变量

数据仓库

数据清洗

- 数据剖析
    - 数据剖析（ Data Profile）： 分析并统计出原数据的类型、取值范围、长度、最大最小值等。
    - 数据剖析是 ETL 工作的第一步，在 ETL 的需求阶段就要开始。
    - 数据剖析的结果，是将来数据检验、更正步骤的依据。
    - Kettle 里使用的数据剖析工具是DataCleaner

- 数据检验
    - 用来检验数据的格式，NULL值，取值范围等是否符合要求。把符合规则的数据和不符合规则的数据区分开，并定义错误代码和描述，便于后续处理
    - 实践1：最好把每个检验条件放在不同的检验里，即使是对同一个字段的检验。如一个规则检验个字段是否为NULL，另一个规则验证同一个字段的取值范围
    - 实践2：数据检验的转换可以放在作业流程的开始部分， 如果发现不符合规则数据，作业退出

统计步骤

- 数据采样
    1. 从N个元素中随机的抽取k个元素，其中N无法确定
    2. 每个元素被选中的可能性都是相等的
    3. 使用 R 算法， (Vitter 1985)
    4. 设置相同的种子，采样的数据也相同

- 前后行查询
    1. 可以查询到同一个字段，前N行或后N行数据
    2. 用于行间数据的计算和统计

- 单变量统计
    1. 个数、最大值、最小值、平均值、中位数。
    2. 标准差、百分比（插值）

分区

- 镜像分区
- 取模分区
- 分区数据写入多个库：需要数据库先建立集群。
- 分区数据写入多个文件：使用内部变量， ${Internal.Step.Partition.ID}，作为文件名的一部分。

脚本

- 使用 var 声明变量，使用”; ” 结束语句。如var myVar;
- 在脚本里可以直接使用 java 的类，要使用类的全名。如
  不兼容模式： myVar = new java.lang.String(“pentahochina.com”);
  兼容模式： myVar = new Packages.java.lang.String (“pentahochina.com”);
  此时myVar是一个java.lang.String对象，可以直接使用 String里的方法。如
  myVar.replaceAll(”china”，””);
  myVar.replaceFirst(”.”，””);
- 获取字段的值：
  不兼容模式：直接使用字段名， myVar = FieldName;
  兼容模式：根据字段类型的不同， 使用不同的方法：
  myVar = FieldName.getString();
  myVar = FieldName.getNumber();
- 给字段赋值：
  不兼容模式：直接使用字段名，如 FieldName = myVar;
  兼容模式：使用 FieldName.setValue(myVar);
- 将变量输出为字段：
  使用“自动获取变量”按钮，手工设置字段类型
- 调试

```
Alert();
```

实践

资源库vs文件

资源库方式优点：

1. 可以共享数据库连接等资源
2. 便于管理（如使用SQL语句统一修改某个参数）

资源库方式缺点：

1. 开源版本不能多人同时访问资源库中的文件。
2. 没有版本控制 （企业版有版本控制）
3. 部署时需要有数据库作为资源库。

文件方式优点：

1. 可以使用 svn 做版本控制。
2. 支持多人并行开发。

文件方式缺点：

1. 开发团队成员之间不能共享数据库连接对象。
   使用变量自己维护。
2. 不利于查询，统计，管理。
3. 大量转换文件时， xml 解析效率低。

多人开发流程 – 资源库方式

1. 创建开发环境资源库。根据业务设置资源库目录，分配给开发人员。
2. 在资源库中创建数据库连接，定义必要的文件/目录等参数名，把参数名给开发人员统一使用， 开发人员根据情况， 设置参数值。
3. 开发人员设计作业后，把作业保存到自己的资源库目录下。
4. 开发完成后，创建测试资源库，把开发资源库的转换导入到测试资源库（全部/按目录），测试人员使用测试资源库测试。开发人员修改bug，并提交到资源库，再导出到测试资源库。
5. 实施人员在现场创建生产资源库，导入测试资源库的作业，修改作业里的参数值，调度运行作业。

多人开发流程 – 文件方式

1. 服务器上创建 SVN 版本控制系统，按照业务流程/阶段等建立目录结构，分配给开发人员。
2. 定义转换中涉及到的数据库/文件目录参数名， 定义转换中涉及到的数据库/文件目录参数名， 如 ${db_host_name}，以及数据库连接的名称。开发人员根据情况设置参数值。
3. 开发人员设计作业后，把作业提交到自己的目录下。
4. 开发完成后，测试人员从 SVN 下载作业，设置参数值，进行测试。开发人员修改相应转换并提交。版本稳定后，所有的ETL作业发布，打标签。
5. 实施人员从 svn 下载打标签的作业文件。把作业文件拷贝到用户服务器上，设置作业里的参数值，调度运行作业。

日志

资源库日志 vs 文件日志

1. 文件日志比较灵活， 如果发生错误，用户把日志文件传给分析人员。
2. 数据库日志可以记录更多的内容，结构化。便于查询和管理。
3. 上线后，处于防火墙或安全等原因，如果运维人员不能直接访问日志数据库，还是要使用文件日志。

命名

作业/转换/步骤的命名规则，有可能出现两种极端的情况：
• 一种是把命名规则设计的过于烦琐。
• 一种是根本没有命名规则。

命名示例： tr_stg_sap_customer， tr_stg_crm_customer， jb_clean_d_customer
作业和转换建议的命名规则：

1. tr_${阶段名}_${表名}/${业务名} 和 jb_${阶段名}_${表名}/${业务名}
2. 资源库不同目录下的转换和文件名不建议使用相同的名字。

转换中使用到的列的命名规则：

- 参与运算的临时列，不输出的列： 使用 t_ 或 tmp_ 为前缀。
- 输出的列：参照数据库的设计规范。

调度

- 尽量避免使用 Kettle 作业开始节点的调度。 Spoon 或kitchen 进程长期驻留内容，容易产生OOM 错误。
- 可以使用操作系统调度，通过操作系统调度直接执行Kitchen 命令行。 Kitchen 执行完会退出，不会长期驻留内存。
- 可以使用第三方调度机制，传递给第三方程序命令行。

监控

监控方法：

1. 开源版：作业在远程的 carte 上运行，通过下面 URL 监控 http://ip:8080/kettle/status 查看运行状态。
2. Pentaho 企业版： Kettle 企业版有服务器提供了调度监控等功能。

错误处理

错误通知：在主作业流里，每个作业项当执行失败时都发送邮件。
错误处理：

1. 发生错误后，一般使用人工处理，修改错误，重新跑作业。
2. 也可在作业里做循环判断，一般用于检测某个条件，如文件是否存在等，或设定循环次数。

注意第一种方法，上次失败已经提交的数据不能再回滚，所以作业在设计时不要依赖状态，保证在发生错误后，作业可以随时重跑。

安全和审计

数据安全

Kettle 里所有保存的数据库密码都可以使用密文保存。
kettle.properties 里的密码也可以手工保存为加密的形式。
命令行下执行`encr -kettle 123`
得到 123 对应的密码是 2be98afc86aa7f2e4cb79ce10bec3fd89
直接在 kettle.properties 文件里设置
`Password = Encrypted 2be98afc86aa7f2e4cb79ce10bec3fd89`
Kettle没有数据源安全性和数据混淆的功能。通过二次开发可以支持。

影响分析

影响分析：原数据库的元数据（表名、列名）发生变化时，对作业和目标数据库的影响。

- Kettle 开源版提供了有限的影响分析功能， Spoon 里的数据库影响分析。
- 企业版有完整的影响分析， 用户可在开源版基础上二次开发。

血统分析

血统分析：分析目标数据库/数据仓库的数据来源，和经过了哪些转换。

- Kettle 开源版没有血统分析功能。
- Kettle 企业版提供血统分析功能，用户可在开源版基础上二次开发。

使用batch id：通过 batch id 把所有一次加载的表都关联起来，知道那些数据是通过哪次ETL过程生成的。

注释和文档

每个转换都要有注释，注释应该包括的内容：

1. 转换的基本作用
2. 上线后如果要修改转换，要在注释里注明每次修改的时间，修改目的，修改人（企业版本有版本控制，这些信息不用写在注释里）
3. 其他运行转换要注意的问题

文档：

1. 应该把写文档的过程和 ETL 开发过程结合起来。如果两个流程分离，那么文档总有一天会过时。
2. 使用Kettle的自动文档功能输出文档。https://code.google.com/p/kettle-cookbook/

转换性能调优

普通开发机器，如果没有网络查询类步骤（ http 等），Kettle正常的速度应该在 3000 ~ 20000 条/秒。如果速度在 2000条/秒以下，就可能需要调优。

容易产生性能问题的步骤：

- 查询类：
  数据库查询，数据库连接， http get/post， webservice，插入/更新，
- 运算类：格式转换，复杂计算
  Javascript，计算器
- 排序类
  排序，合并连接，分组

调优的关键： Rowset

1. Rowset 是两个步骤之间的缓存。
2. 性能调优需要找出性能瓶颈步骤。
3. 如何通过观察 rowset，找到性能瓶颈。
4. 其它调优工具。
5. 如何提高瓶颈步骤的性能：
   - 合适的数据库索引：数据库查询类
   - 增加复制数：查询类
   - 加大缓存：排序

    - 集群：查询类，运算类，排序
    - 更换其他实现方式： 如 javascript使用Java类或插件

6. 注意日志级别（ Rowlevel 日志的性能会严重下降，是 Basic的 1/10）
7. 注意死锁问题： 1. 数据库死锁 （读写同一个表） 2.转换本身死锁

作业

作业：由多个作业项组成，用来执行一系列的工作，类似于脚本
作业的执行顺序：
串行：深度优先遍历
并行：各个子树同时执行

作业的执行状态：成功，失败，无条件
用户自定义控制表

1. 控制作业执行条件
2. 让一个作业项前面的作业项都执行成功后，再执行后面的作业项

数据库查询步骤vs流查询步骤

数据库查询步骤和流查询步骤的区别

- 流查询步骤只能进行等值查询，数据库查询步骤可以进行非等值查询
- 流查询在查询之前把数据都加载到内存里，数据库查询是否加载可以选择。
- 进行等值查询时，数据库查询步骤如果选中了全部缓存，性能和流查询类似
- 进行等值查询时，数据库查询步骤如果没选中全部缓存，性能较低
- 进行非等值查询时，数据库查询即使选中了全部缓存，性能也较低，没有索引

作业循环

计数循环：
有固定次数的循环，类似 for 循环
方法一：使用字段，通过“拷贝结果”和“读取结果”传递参数。
方法二：使用转换作业项的参数
方法三：使用变量，每次变量加1

条件循环：
不固定次数的循环，类似 while 循环
方法：使用作业里的条件类作业项，设置变量，通过变量判断。

作业技巧

作业设计技巧：

1. 使用控制表，进行调度。
2. 执行前面传来的每一行，作为命令行参数。
3. 正确和错误都发送邮件。
4. 心跳检测。

执行 SQL 语句的步骤：

- 没有参数的执行 SQL 步骤，在转换初始化阶段执行。所以可以用来执行一些建表语句。
- 有参数的执行 SQL 步骤，和其他步骤一样。

数据仓库

基本概念

数据仓库在20世纪90年代开始发展，Bill Inmon 和 Ralph Kimball 分别阐述了他们对数据仓库的观点。总的观点： 创建一种数据环境，以便支持数据分析和报表的应用。
两种观点的最大区别就是是否需要一个企业级的数据仓库（Enterprise Data Warehouse EDW）。 Inmon 认为需要一个EDW，Kimball 认为不需要。 EDW 从本质上就是一个大的数据仓库，包括了从企业各个数据库集成过来的所有的历史数据。 EDW 不能由终端用户直接访问。
两种观点的相似之处。 Inmon 和 Kimball 都同意使用数据集市。数据集市就是面向终端用户的数据库。

多维模型

维度数据和事实数据：

- 事实数据：用户关心的业务数据，如销售数量，库存数量，销售金额
- 维度数据：用来描述业务数据的数据，如日期，产品名称，地区，渠道。

多维模型：

- 星型模型：一个维度保存在一个表里。
- 雪花模型：类似E-R 模型，一个维度保存在多个表里。
- 混合模型：有的维度是雪花，有的维度星型
- Data Vault 模型：介于雪花和星型之间

代理键和业务键

- 业务键：在原系统中使用，业务实体的标识。单列或多列：如身份证号，姓名，自增列，uuid
- 代理键（技术键）：在数据仓库中使用，连接维度表和事实表。单列整数值

代理键的生成方式：“增加序列”步骤，数据库自增列

为什么使用代理键

- 因为要保存维度的修改历史（SCD2），业务键不能唯一标识一条记录。
- 节省空间 1 Byte * 100,000,000 = 100M

加载维度表

缓慢变化维(SCD)

- 类型1 的缓慢变化维： SCD 1，直接更新
- 类型2 的缓慢变化维： SCD 2，不更新，插入新行（更新历史记录的有效期）
- 类型3 的缓慢变化维： SCD 3，插入新列

SCD2
业务主键：原系统的业务主键也保留在维度表中。
代理键：维度表的主键。
有效开始期：维度记录的开始日期。
有效截止日期： 维度记录的截止日期。
版本号：区分同一个业务主键的不同记录

使用不同的步骤来加载维度表

- SCD 1：插入/更新步骤
- SCD 2：维度查询/更新步骤
- SCD 3：执行SQL步骤

其它类型维度表

生成维：可以事前准备好的维度，以后也不改变，如日期维度，区域维度，人口（性别、年龄段）
杂项维：不好归类的，不能提前确定值的小维度（付款方式，订单状态...)
使用“联合查询/更新”步骤加载杂项维
类似于插入更新，但有如下区别：

- 返回代理键
- 不区分查询键和返回字段

加载事实表

加载事实表

- 使用流查询或数据库查询步骤
- 使用各种批量加载步骤： MySQL支持双向批量
- 作业项里的批量加载步骤和转换里的批量加载步骤的区别
  作业：基于文件的方式（生成中间文件）
  转换：基于数据流的方式： API， STDIN
- 外键的使用问题
  不建议使用外键约束，可以使用外键，但加载时删除/禁用。

维度查询更新步骤

- 两种模式
  更新模式：用于维度更新
  查询模式：用于查询代理键
- 两种模式如何选择：是否更新维度
- 最小年份、最大年份：用于第一条记录
- 除了比较关键字，还要比较指定的流字段
- 操作类型：
  更新，插入，穿透
  列一级，不同的列可以有操作类型

Kettle 配置文件

- ${user.home}/.kettle /repositories.xml：
  该文件保存了用户设置的所有资源库信息，包括资源库名称，资源库需要的数据库连接参数等。该文件中定义的资源库将显示在 spoon 启动后出现的选择资源库下拉列表中，注意该文件的编码是 UTF-8， 资源库的名称尽量使用英文。
- ${user.home}/.kettle/kettle.properties
  该文件保存了转换或作业中需要的变量，spoon 启动后会自动加载该文件里定义的变量。
- ${user.home}/.kettle/shared.xml
  该文件里保存了共享对象，共享对象可以是Database connections， Steps， Slave servers， Partition schemas， Cluster schemas。对象共享实质上就是将对象序列化的过程，spoon 启动时，会加载 shared.xml 文件中定义的所有对象。

- lib\kettle-engine.jar\kettle-jobs.xml
  该文件中定义了Spoon启动时需要加载的作业项
- lib\kettle-engine.jar\kettle-partition-plugins.xml
  该文件中定义了Spoon启动时需要加载的分区插件
- lib\kettle-engine.jar\kettle-plugins.xml
  该文件中定义了Spoon启动时步骤和作业项插件的加载路径。
- lib\kettle-engine.jar\kettle-steps.xml
  该文件中定义了Spoon 启动时需要加载的转换步骤。