



Informatica® PowerCenter  
10.4.0

# 转换指南

## Informatica PowerCenter 转换指南

10.4.0

2019 年 12 月

© 版权所有 Informatica LLC 1999, 2020

本软件和文档仅根据包含使用与披露限制的单独许可协议提供。未事先征得 Informatica LLC 同意，不得以任何形式、通过任何手段（电子、影印、录制或其他手段）复制或传播本文档的任何部分。

Informatica、Informatica 标志和 PowerCenter 是 Informatica LLC 在美国和世界其他许多司法管辖区的商标或注册商标。欲获得 Informatica 商标的最新列表，请访问 <https://www.informatica.com/trademarks.html>。其他公司和产品名称可能是其各自所有者的商业名称或商标。

本软件和/或文档的某些部分受第三方版权制约，包括但不限于：版权所有 DataDirect Technologies。保留所有权利。版权所有 (C) Sun Microsystems。保留所有权利。版权所有 (C) RSA Security Inc. 保留所有权利。版权所有 (C) Ordinal Technology Corp. 保留所有权利。版权所有 (C) Aandacht c.v. 保留所有权利。版权所有 Genivia, Inc. 保留所有权利。版权所有 Isomorphic Software。保留所有权利。版权所有 (C) Meta Integration Technology, Inc. 保留所有权利。版权所有 (C) Intalio。保留所有权利。版权所有 (C) Oracle。保留所有权利。版权所有 (C) Adobe Systems Incorporated。保留所有权利。版权所有 (C) DataArt, Inc. 保留所有权利。版权所有 (C) ComponentSource。保留所有权利。版权所有 (C) Microsoft Corporation。保留所有权利。版权所有 (C) Rogue Wave Software, Inc. 保留所有权利。版权所有 (C) Teradata Corporation。保留所有权利。版权所有 (C) Yahoo! Inc. 保留所有权利。版权所有 (C) Glyph & Cog, LLC。保留所有权利。版权所有 (C) Thinkmap, Inc. 保留所有权利。版权所有 (C) Clearpace Software Limited。保留所有权利。版权所有 (C) Information Builders, Inc. 保留所有权利。版权所有 (C) OSS Nokalva, Inc. 保留所有权利。版权所有 Edifecs, Inc. 保留所有权利。版权所有 Cleo Communications, Inc. 保留所有权利。版权所有 (C) International Organization for Standardization 1986。保留所有权利。版权所有 (C) ej-technologies GmbH。保留所有权利。版权所有 (C) Jaspersoft Corporation。保留所有权利。版权所有 (C) International Business Machines Corporation。保留所有权利。版权所有 (C) yWorks GmbH。保留所有权利。版权所有 (C) Lucent Technologies。保留所有权利。版权所有 (C) University of Toronto。保留所有权利。版权所有 (C) Daniel Veillard。保留所有权利。版权所有 (C) Unicode, Inc. 版权所有 IBM Corp. 保留所有权利。版权所有 (C) MicroQuill Software Publishing, Inc. 保留所有权利。版权所有 (C) PassMark Software Pty Ltd. 保留所有权利。版权所有 (C) LogiXML, Inc. 保留所有权利。版权所有 (C) 2003-2010 Lorenzi Davide。保留所有权利。版权所有 (C) Red Hat, Inc. 保留所有权利。版权所有 (C) The Board of Trustees of the Leland Stanford Junior University。保留所有权利。版权所有 (C) EMC Corporation。保留所有权利。版权所有 (C) Flexera Software。保留所有权利。版权所有 (C) Jinfonet Software。保留所有权利。版权所有 (C) Apple Inc. 保留所有权利。版权所有 (C) Telerik Inc. 保留所有权利。版权所有 (C) BEA Systems。保留所有权利。版权所有 (C) PDFlib GmbH。保留所有权利。版权所有 (C) Orientation in Objects GmbH。保留所有权利。版权所有 (C) Tanuki Software, Ltd. 保留所有权利。版权所有 (C) Ricebridge。保留所有权利。版权所有 (C) Sencha, Inc. 保留所有权利。版权所有 (C) Scalable Systems, Inc. 保留所有权利。版权所有 (C) jQWidgets。保留所有权利。版权所有 (C) Tableau Software, Inc. 保留所有权利。版权所有 (C) MaxMind, Inc. 保留所有权利。版权所有 (C) TMate Software s.r.o. 保留所有权利。版权所有 (C) MapR Technologies Inc. 保留所有权利。版权所有 (C) Amazon Corporate LLC。保留所有权利。版权所有 (C) Highsoft。保留所有权利。版权所有 (C) Python Software Foundation。保留所有权利。版权所有 (C) BeOpen.com。保留所有权利。版权所有 (C) CNRI。保留所有权利。

本产品包括由 Apache Software Foundation (<http://www.apache.org/>) 开发的软件 and/或在不同 Apache 许可证版本（以下简称“许可证”）下许可的其他软件。您可从 <http://www.apache.org/licenses/> 获取这些许可证的副本。除非适用法律要求或者有相应书面协议，否则依据这些“许可证”分发的软件以“原样”提供，不附带任何明示或暗示的担保或条件。请参阅“许可证”中规定的具体语言管理权限和限制。

本产品包括由 Mozilla (<http://www.mozilla.org/>) 开发的软件、由 JBoss Group, LLC 开发的软件（版权所有 JBoss Group, LLC 保留所有权利）、由 Bruno Lowagie 和 Paulo Soares 开发的软件（版权所有 (C) 1999-2006 Bruno Lowagie 和 Paulo Soares）以及在 <http://www.gnu.org/licenses/lgpl.html> 网站上的不同版本 GNU Lesser General 公共许可协议下许可的软件。这些材料由 Informatica 按“原样”免费提供，不附带任何明示或暗示的担保，包括但不限于适销性和特定用途适用性的暗示担保。

本产品包括 ACE(TM) 和 TAO(TM) 软件，这些软件版权归 Douglas C. Schmidt 及其在华盛顿大学、加利福尼亚大学欧芬分校以及范德堡大学的研发团队所有（版权所有 (C) 1993-2006，保留所有权利）。

本产品包括由 OpenSSL Project 开发并在 OpenSSL Toolkit（版权所有 OpenSSL Project。保留所有权利）中使用的软件，该软件的再分发受 <http://www.openssl.org> 和 <http://www.openssl.org/source/license.html> 上规定条款之制约。

本产品包括 Curl 软件，版权所有 1996-2013, Daniel Stenberg <[daniel@haxx.se](mailto:daniel@haxx.se)>。保留所有权利。有关该软件的权限和限制受 <http://curl.haxx.se/docs/copyright.html> 上规定条款之制约。允许出于任何目的以免费或收费形式使用、复制、修改和分发该软件，但前提是所有副本均应注明上述版权声明以及本许可声明。

本产品包括由 MetaStuff, Ltd. 开发的软件，版权所有 2001-2005 ((C)) MetaStuff, Ltd. 保留所有权利。有关该软件的权限和限制受 <http://www.dom4j.org/license.html> 上规定条款之制约。

本产品包括由 Per Bothner 开发的软件，版权所有 (C) 1996-2006 Per Bothner。保留所有权利。<http://www.gnu.org/software/kawa/Software-License.html> 上的许可证中规定了您使用这些材料的权利。

本产品包括 OSSP UUID 软件，版权所有 (C) 2002 Ralf S. Engelschall，版权所有 (C) 2002 OSSP Project，版权所有 (C) 2002 Cable & Wireless Deutschland。有关该软件的权限和限制受 <http://www.opensource.org/licenses/mit-license.php> 上规定条款之制约。

本产品包括由 Boost (<http://www.boost.org/>) 开发的软件或在 Boost 软件许可证下许可的软件。有关该软件的权限和限制受 [http://www.boost.org/LICENSE\\_1\\_0.txt](http://www.boost.org/LICENSE_1_0.txt) 上规定条款之制约。

本产品包括由 University of Cambridge 开发的软件，版权所有 (C) 1997-2007 University of Cambridge。有关该软件的权限和限制受 <http://www.pcre.org/license.txt> 上规定条款之制约。

本产品包括由 The Eclipse Foundation 开发的软件，版权所有 (C) 2007 The Eclipse Foundation。保留所有权利。有关该软件的权限和限制受 <http://www.eclipse.org/org/documents/epl-v10.php> 和 <http://www.eclipse.org/org/documents/edl-v10.php> 上规定条款之制约。

本产品包括在 <http://www.tcl.tk/software/tcltk/license.html>、<http://www.bosrup.com/web/overlib/?License>、<http://www.stlport.org/doc/license.html>、<http://asm.ow2.org/license.html>、<http://www.cryptix.org/LICENSE.TXT>、<http://hsqldb.org/web/hsqldbLicense.html>、<http://httpunit.sourceforge.net/doc/license.html>、<http://jung.sourceforge.net/license.txt>、[http://www.gzip.org/zlib/zlib\\_license.html](http://www.gzip.org/zlib/zlib_license.html)、<http://www.openldap.org/software/release/liblicense.html>、<http://www.libssh2.org>、<http://slf4j.org/license.html>、<http://www.sente.ch/software/OpenSourceLicense.html>、<http://fusesource.com/downloads/license-agreements/fuse-message-broker-v-5-3-license-agreement>、<http://antlr.org/license.html>、<http://aopalliance.sourceforge.net/>、<http://www.bouncycastle.org/licence.html>、<http://www.jgraph.com/jgraphdownload.html>、<http://www.jcraft.com/jsch/LICENSE.txt>、[http://jotm.objectweb.org/bsd\\_license.html](http://jotm.objectweb.org/bsd_license.html)、<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>、<http://www.slf4j.org/license.html>、<http://nanoxml.sourceforge.net/orig/copyright.html>、<http://www.json.org/license.html>、<http://forge.ow2.org/projects/javaservice/>、<http://www.postgresql.org/about/license.html>、<http://www.sqlite.org/copyright.html>、<http://www.tcl.tk/software/tcltk/license.html>、<http://www.jaxen.org/faq.html>、<http://www.jdom.org/docs/faq.html>、<http://www.slf4j.org/license.html>、<http://www.iodbc.org/dataspace/iodbc/wiki/ODBC/License>、<http://www.keplerproject.org/md5/license.html>、<http://www.toedter.com/en/jcalendar/license.html>、<http://www.edankert.com/bounce/index.html>、<http://www.net-snmp.org/about/license.html>、<http://www.openmdx.org/#FAQ>、[http://www.php.net/license/3\\_01.txt](http://www.php.net/license/3_01.txt)、<http://srp.stanford.edu/license.txt>、<http://www.schneier.com/blowfish.html>、<http://www.jmock.org/license.html>、<http://xsom.java.net>、<http://benalman.com/about/license/>、<https://github.com/CreateJS/EaselJS/blob/master/src/easeljs/display/Bitmap.js>、<http://www.h2database.com/html/license.html#summary>、<http://jsoncpp.sourceforge.net/LICENSE>、<http://jdbc.postgresql.org/license.html>、<http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto>、<https://github.com/rantav/hector/blob/master/LICENSE>、<http://web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html>、<http://jibx.sourceforge.net/jibx-license.html>、<https://github.com/lyokato/libgeohash/blob/master/LICENSE>、<https://github.com/hjiang/jsonxx/blob/master/LICENSE>、<https://code.google.com/p/lz4/>、<https://github.com/jedisct1/libsodium/blob/master/LICENSE>、<http://one-jar.sourceforge.net/index.php?page=documents&file=license>、<https://github.com/EsotericSoftware/kryo/blob/master/license.txt>、<http://www.scala-lang.org/license.html>、<https://github.com/tinkerpop/blueprints/blob/master/LICENSE.txt>、<http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html>、<https://aws.amazon.com/asl/>、<https://github.com/twbs/bootstrap/blob/master/LICENSE> 和 <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt> 下许可的软件。

本产品包括在 Academic 免费许可证 (<http://www.opensource.org/licenses/afl-3.0.php>)、通用开发和分发许可证 (<http://www.opensource.org/licenses/cddl1.php>)、通用公共许可证 (<http://www.opensource.org/licenses/cpl1.0.php>)、Sun Binary Code 许可协议补充许可条款、BSD 许可证 (<http://www.opensource.org/licenses/bsd-license.php>)、新 BSD 许可证 (<http://opensource.org/licenses/BSD-3-Clause>)、MIT 许可证 (<http://www.opensource.org/licenses/mit-license.php>)、Artistic 许可证 (<http://www.opensource.org/licenses/artistic-license-1.0>) 以及原始开发者公共许可证版本 1.0 (<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>) 下许可的软件。

本产品包括由 Joe Walnes 和 XStream Committers 开发的软件，版权所有 (C) 2003-2006 Joe Walnes, 2006-2007 XStream Committers。保留所有权利。有关该软件的权限和限制受 <http://xstream.codehaus.org/license.html> 上规定条款之制约。本产品包括由 Indiana University Extreme! Lab 开发的软件。有关详细信息，请访问 <http://www.extreme.indiana.edu/>。

本产品包括软件版权所有 (c) 2013 Frank Balluffi 和 Markus Moeller。保留所有权利。有关此软件的权限和限制受 MIT 许可证上规定条款之制约。

请参阅位于以下位置的专利：<https://www.informatica.com/legal/patents.html>。

免责声明：Informatica LLC 以“原样”提供本文档，不附带任何明示或暗示的担保，包括但不限于非侵权、适销性或特定用途适用性的暗示担保。Informatica LLC 不保证本软件和文档中没有错误。本软件或文档中提供的信息可能包括技术上的不准确性或排字错误。本软件和文档中包含的信息随时可能更改，恕不另行通知。

#### 声明

本 Informatica 产品（以下称“软件”）包括由 Progress Software Corporation 的运营公司 DataDirect Technologies（以下称“DataDirect”）提供的某些驱动程序（以下称“DataDirect 驱动程序”），受以下条款和条件制约：

1. DataDirect 驱动程序以“原样”提供，不附带任何明示或暗示的担保，包括但不限于适销性、特定用途适用性以及非侵权的暗示担保。
2. 在任何情况下，DataDirect 或其第三方供应商均不对最终用户客户承担因使用 ODBC 驱动程序而引起的任何直接、间接、偶发、特殊、继发或其他损害赔偿的责任，无论是否已提前告知该种损害的可能性。这些限制适用于所有诉因，包括但不限于违反合同、违反担保、过失、严格责任、虚假陈述以及其他侵权行为。

本文档中的信息如有更改，恕不另行通知。如发现本文档中有什么问题，请通过以下电子邮件地址向我们报告：[infa\\_documentation@informatica.com](mailto:infa_documentation@informatica.com)。

Informatica 产品根据对应协议的条款和条件进行担保。INFORMATICA 按“原样”提供本文档中的信息，无任何明示或暗示的担保，包括但不限于任何适销性和特定用途适用性担保，也没有任何非侵权担保或条件。

发布日期: 2020-02-24

# 目录

前言 .....	22
Informatica 资源 .....	22
Informatica Network .....	22
Informatica 知识库 .....	22
Informatica 文档 .....	22
Informatica 产品可用性矩阵 .....	23
Informatica Velocity .....	23
Informatica Marketplace .....	23
Informatica 全球客户支持部门 .....	23
第 1 章：使用转换 .....	24
转换概览 .....	24
主动转换 .....	24
被动转换 .....	25
未连接的转换 .....	25
本地转换和非本地转换 .....	25
转换说明 .....	25
创建转换 .....	28
配置转换 .....	28
重命名转换 .....	28
转换端口 .....	29
创建端口 .....	29
配置端口 .....	29
链接端口 .....	29
多组转换 .....	29
使用表达式 .....	30
使用表达式编辑器 .....	31
计算表达式 .....	32
计算表达式 .....	33
计算表达式限制 .....	33
局部变量 .....	34
临时存储数据和简化复杂表达式 .....	34
存储行中的值 .....	35
捕获来自存储过程的值 .....	35
配置变量端口的准则 .....	35
端口的默认值 .....	36
用户定义的默认值 .....	37
用户定义的默认输入值 .....	38
用户定义的默认输出值 .....	40
默认值的常规规则 .....	41

默认值验证. . . . .	42
配置转换中的跟踪级别. . . . .	42
可重用转换. . . . .	43
实例和继承的更改. . . . .	43
表达式中的映射变量. . . . .	43
创建可重用转换. . . . .	43
提升不可重用转换. . . . .	44
创建可重用转换的不可重用实例. . . . .	44
将可重用转换添加到映射中. . . . .	44
修改可重用转换. . . . .	44
<b>第 2 章： 汇总器转换. . . . .</b>	<b>46</b>
汇总器转换概览. . . . .	46
汇总器转换组件. . . . .	46
配置汇总器转换属性. . . . .	47
配置汇总器转换端口. . . . .	47
配置汇总缓存. . . . .	48
汇总表达式. . . . .	48
汇总函数. . . . .	48
嵌套汇总函数. . . . .	49
条件子句. . . . .	49
非汇总函数. . . . .	49
汇总函数中的空值. . . . .	49
分组依据端口. . . . .	49
非汇总表达式. . . . .	50
默认值. . . . .	50
使用已排序输入. . . . .	51
已排序输入条件. . . . .	51
数据排序. . . . .	51
创建汇总器转换. . . . .	52
汇总器转换提示. . . . .	53
汇总器转换故障排除. . . . .	53
<b>第 3 章： 自定义转换. . . . .</b>	<b>54</b>
自定义转换概览. . . . .	54
使用基于自定义转换构建的转换. . . . .	54
代码页兼容性. . . . .	55
分发自定义转换过程. . . . .	55
创建自定义转换. . . . .	55
自定义转换的规则和准则. . . . .	56
自定义转换组件. . . . .	56
使用组和端口. . . . .	56
创建组和端口. . . . .	57

编辑组和端口	57
定义端口关系	57
使用端口属性	58
编辑端口属性值	58
自定义转换属性	58
设置更新策略	60
使用线程特定的过程代码	60
使用事务控制	60
转换范围	60
生成事务	61
使用事务边界	61
阻止输入数据	61
将过程代码写入块数据	62
将自定义转换配置为阻止转换	62
使用自定义转换验证映射	62
使用过程属性	63
创建自定义转换过程	63
步骤 1。创建自定义转换	63
步骤 2。生成 C 文件	65
步骤 3。使用转换逻辑填充代码	66
步骤 4.构建模块	71
步骤 5。创建映射	72
步骤 6。在工作流中运行会话	73
<b>第 4 章：自定义转换函数</b>	<b>74</b>
自定义转换函数概览	74
使用句柄	74
函数引用	75
使用行	78
基于行和基于数组的数据访问模式的规则和准则	79
生成的函数	79
初始化函数	79
通知函数	81
取消初始化函数	82
API 函数	84
设置数据访问模式函数	85
导航函数	85
属性函数	87
重新绑定数据类型函数	94
数据处理函数（基于行的模式）	96
设置传递端口函数	98
输出通知函数	98
数据边界输出通知函数	99

错误函数. . . . .	99
会话日志消息函数. . . . .	99
递增错误计数函数. . . . .	100
已终止函数. . . . .	101
编块函数. . . . .	101
指针函数. . . . .	102
更改字符串模式函数. . . . .	103
设置数据代码页函数. . . . .	103
行策略函数（基于行的模式）. . . . .	104
更改默认行策略函数. . . . .	105
基于数组的 API 函数. . . . .	106
最大行数函数. . . . .	106
行数函数. . . . .	107
行是否有效函数. . . . .	108
数据处理函数（基于数组的模式）. . . . .	108
行策略函数（基于数组的模式）. . . . .	110
设置输入错误行函数. . . . .	111
 第 5 章：数据屏蔽转换. . . . .	113
数据屏蔽转换. . . . .	113
屏蔽属性. . . . .	114
区域设置. . . . .	114
屏蔽类型. . . . .	114
可重复的输出. . . . .	115
种子. . . . .	115
映射参数. . . . .	115
关联 O/P. . . . .	115
键屏蔽. . . . .	116
屏蔽字符串值. . . . .	116
屏蔽数值. . . . .	117
屏蔽日期时间值. . . . .	117
置换屏蔽. . . . .	117
字典. . . . .	118
存储表. . . . .	118
置换屏蔽属性. . . . .	119
关系字典. . . . .	119
连接要求. . . . .	120
置换屏蔽的规则和准则. . . . .	120
相关屏蔽. . . . .	120
相关屏蔽示例. . . . .	120
可重复的相关屏蔽. . . . .	121
随机屏蔽. . . . .	122
屏蔽数值. . . . .	122

屏蔽字符串值. . . . .	122
屏蔽日期值. . . . .	122
应用屏蔽规则. . . . .	123
屏蔽格式. . . . .	123
源字符串字符. . . . .	124
结果字符串替换字符. . . . .	125
范围. . . . .	125
模糊. . . . .	125
表达式屏蔽. . . . .	126
可重复表达式屏蔽. . . . .	127
表达式屏蔽的规则和准则. . . . .	128
特殊屏蔽格式. . . . .	128
社会保障号屏蔽. . . . .	128
社会保障号格式. . . . .	129
区号要求. . . . .	129
可重复社会保障号屏蔽. . . . .	129
信用卡号屏蔽. . . . .	129
电话号码屏蔽. . . . .	130
电子邮件地址屏蔽. . . . .	130
高级电子邮件屏蔽. . . . .	130
社会保险号屏蔽. . . . .	131
SIN 起始数字. . . . .	132
可重复的 SIN 编号. . . . .	132
IP 地址屏蔽. . . . .	132
URL 地址屏蔽. . . . .	132
默认值文件. . . . .	132
数据屏蔽转换会话属性. . . . .	133
数据屏蔽转换的规则和准则. . . . .	134
<b>第 6 章：数据屏蔽示例. . . . .</b>	<b>135</b>
名称和地址查找文件. . . . .	135
通过查找转换置换数据. . . . .	135
通过表达式转换屏蔽数据. . . . .	138
<b>第 7 章：表达式转换. . . . .</b>	<b>141</b>
表达式转换概览. . . . .	141
表达式转换组件. . . . .	141
配置端口. . . . .	142
计算值. . . . .	142
创建表达式转换. . . . .	142
<b>第 8 章：外部过程转换. . . . .</b>	<b>144</b>
外部过程转换概览. . . . .	144



代码页兼容性. . . . .	144
外部过程和外部过程转换. . . . .	145
外部过程转换属性. . . . .	145
COM 与 Informatica 外部过程对比. . . . .	145
BankSoft 示例. . . . .	145
配置外部过程转换属性. . . . .	145
开发 COM 过程. . . . .	147
COM 过程创建步骤. . . . .	147
COM 外部过程服务器类型. . . . .	148
使用 Visual C++ 开发 COM 过程. . . . .	148
使用 Visual Basic 开发 COM 过程. . . . .	153
开发 Informatica 外部过程. . . . .	154
步骤 1。创建外部过程转换. . . . .	155
步骤 2。生成 C++ 文件. . . . .	157
步骤 3。使用实施填充方法存根. . . . .	158
步骤 4。构建模块. . . . .	159
步骤 5。创建映射. . . . .	160
步骤 6。运行会话. . . . .	160
分发外部过程. . . . .	161
分发 COM 过程. . . . .	161
分发 Informatica 模块. . . . .	162
开发说明. . . . .	162
COM 数据类型. . . . .	162
行级过程. . . . .	163
过程返回值. . . . .	163
过程调用中的异常. . . . .	164
过程内存管理. . . . .	164
已有 C/C++ 库或 VB 函数的包装类. . . . .	164
生成错误和跟踪消息. . . . .	164
未连接的外部过程转换. . . . .	166
初始化 COM 和 Informatica 模块. . . . .	166
TX 中分发和使用的其他文件. . . . .	167
初始化属性中的服务进程变量. . . . .	168
外部过程接口. . . . .	168
分派函数. . . . .	168
外部过程函数. . . . .	169
属性访问函数. . . . .	169
参数访问函数. . . . .	169
代码页访问函数. . . . .	172
转换名访问函数. . . . .	172
过程访问函数. . . . .	172
分区相关函数. . . . .	172

跟踪级别函数. . . . .	173
<b>第 9 章：筛选器转换. . . . .</b>	<b>174</b>
筛选器转换概览. . . . .	174
筛选转换组件. . . . .	175
配置筛选器转换端口. . . . .	175
筛选条件. . . . .	175
筛选具有空值的行. . . . .	175
创建筛选器转换的步骤. . . . .	176
筛选器转换提示. . . . .	176
<b>第 10 章：HTTP 转换. . . . .</b>	<b>177</b>
HTTP 转换概览. . . . .	177
身份验证. . . . .	178
连接至 HTTP 服务器. . . . .	178
创建 HTTP 转换. . . . .	178
配置“属性”选项卡. . . . .	178
配置“HTTP”选项卡. . . . .	179
选择方法. . . . .	180
配置组和端口. . . . .	180
配置 URL. . . . .	183
示例. . . . .	185
GET 示例. . . . .	185
POST 示例. . . . .	186
SIMPLE POST 示例. . . . .	187
SIMPLE PATCH 示例. . . . .	188
SIMPLE PUT 示例. . . . .	189
SIMPLE DELETE 示例. . . . .	190
<b>第 11 章：Identity Resolution 转换. . . . .</b>	<b>192</b>
Identity Resolution 转换概览. . . . .	192
创建和配置转换. . . . .	192
搜索服务器连接. . . . .	192
系统和搜索配置. . . . .	193
视图选择. . . . .	194
Identity Resolution 转换选项卡. . . . .	194
组和端口. . . . .	195
输入组和端口. . . . .	195
输出组和端口. . . . .	195
<b>第 12 章：Java 转换. . . . .</b>	<b>197</b>
Java 转换概览. . . . .	197
定义 Java 转换的步骤. . . . .	198

主动和被动 Java 转换. . . . .	198
数据类型转换. . . . .	198
使用“Java 代码”选项卡. . . . .	200
配置端口. . . . .	200
创建组和端口. . . . .	200
设置默认端口值. . . . .	201
配置 Java 转换属性. . . . .	201
使用事务控制. . . . .	203
设置更新策略. . . . .	203
开发 Java 代码. . . . .	204
创建 Java 代码段. . . . .	204
导入 Java 包. . . . .	205
定义帮助程序代码. . . . .	205
“在输入行”选项卡. . . . .	206
“在数据末尾”选项卡. . . . .	207
“在接收事务中”选项卡. . . . .	207
使用 Java 代码解析平面文件. . . . .	207
配置 Java 转换设置. . . . .	208
配置类路径. . . . .	208
启用高精度. . . . .	209
处理子秒. . . . .	209
编译 Java 转换. . . . .	210
修复编译错误. . . . .	210
定位编译错误来源. . . . .	210
标识编译错误来源. . . . .	211
 第 13 章：Java 转换 API 引用. . . . .	 212
Java 转换 API 方法概览. . . . .	212
提交. . . . .	213
defineJExpression. . . . .	214
failSession. . . . .	214
generateRow. . . . .	215
getInRowType. . . . .	215
getMetadata. . . . .	216
incrementErrorCount. . . . .	217
invokeJExpression. . . . .	217
isNull. . . . .	218
logError. . . . .	219
logInfo. . . . .	219
resetNotification. . . . .	220
rollBack. . . . .	220
setNull. . . . .	221
setOutRowType. . . . .	221

storeMetadata. . . . .	222
<b>第 14 章：Java 表达式. . . . .</b>	<b>224</b>
Java 表达式概览. . . . .	224
表达式函数类型. . . . .	224
使用定义表达式定义函数对话框来定义表达式. . . . .	225
步骤 1。配置函数. . . . .	225
步骤 2。创建并验证表达式. . . . .	226
步骤 3。生成表达式的 Java 代码. . . . .	226
使用定义表达式定义函数对话框创建表达式并生成 Java 代码. . . . .	226
Java 表达式模板. . . . .	226
使用简单接口. . . . .	227
invokeJExpression. . . . .	227
简单接口示例. . . . .	228
使用高级接口. . . . .	228
通过高级接口调用表达式. . . . .	228
使用高级接口的规则和准则. . . . .	229
EDatatype 类. . . . .	229
JExprParamMetadata 类. . . . .	229
defineJExpression. . . . .	230
JExpression 类. . . . .	231
高级接口示例. . . . .	231
JExpression 类 API 引用. . . . .	232
getBytes. . . . .	232
getDouble. . . . .	232
getInt. . . . .	233
getLong. . . . .	233
getResultDataType. . . . .	233
getResultMetadata. . . . .	233
getStringBuffer. . . . .	234
invoke. . . . .	234
isResultNull. . . . .	234
<b>第 15 章：Java 转换示例. . . . .</b>	<b>235</b>
Java 转换示例概览. . . . .	235
步骤 1。导入映射. . . . .	236
步骤 2。创建转换和配置端口. . . . .	236
步骤 3。输入 Java 代码. . . . .	237
“导入包”选项卡. . . . .	237
“帮助程序代码”选项卡. . . . .	237
“在输入行”选项卡. . . . .	238
步骤 4。编译 Java 代码. . . . .	239
步骤 5。创建会话和工作流. . . . .	240

示例数据. . . . .	240
<b>第 16 章： 联接器转换. . . . .</b>	<b>242</b>
联接器转换概览. . . . .	242
使用联接器转换. . . . .	242
联接器转换属性. . . . .	243
定义联接条件. . . . .	244
定义联接类型. . . . .	244
普通联接. . . . .	245
主外部联接. . . . .	245
详细外部联接. . . . .	246
完整外部联接. . . . .	246
使用已排序输入. . . . .	247
配置排序顺序. . . . .	247
将转换添加到映射. . . . .	247
配置联接器转换. . . . .	248
定义联接条件. . . . .	248
联接单个源中的数据. . . . .	249
联接同一管道的两个分支. . . . .	249
联接同一源的两个实例. . . . .	249
有关联接单个源中的数据的准则. . . . .	250
阻止源管道. . . . .	250
未排序联接器转换. . . . .	250
已排序联接器转换. . . . .	250
使用事务. . . . .	251
保留单个管道的事务边界. . . . .	251
在详细管道中保留事务边界. . . . .	252
删除两个管道的事务边界. . . . .	252
创建联接器转换. . . . .	252
联接器转换提示. . . . .	253
<b>第 17 章： 查找转换. . . . .</b>	<b>254</b>
查找转换概览. . . . .	254
查找源类型. . . . .	255
关系查找. . . . .	255
平面文件查找. . . . .	255
管道查找. . . . .	256
已连接和未连接的查找. . . . .	257
已连接的查找. . . . .	258
未连接的查找. . . . .	259
查找组件. . . . .	259
查找源. . . . .	259
查找端口. . . . .	260

查找属性. . . . .	260
查找条件. . . . .	260
查找属性. . . . .	261
在会话中配置查找属性. . . . .	264
查找查询. . . . .	265
默认查找查询. . . . .	266
替代查找查询. . . . .	266
未缓存查找的 SQL 替代. . . . .	268
查找源筛选器. . . . .	268
查找条件. . . . .	269
未缓存或静态缓存. . . . .	269
动态缓存. . . . .	270
处理多项匹配. . . . .	270
查找缓存. . . . .	270
返回多个行. . . . .	271
返回多个行的规则和准则. . . . .	271
配置未连接的查找转换. . . . .	271
步骤 1。添加输入端口. . . . .	272
步骤 2。添加查找条件. . . . .	272
步骤 3。指定返回值. . . . .	272
步骤 4。通过表达式调用查找. . . . .	273
数据库死锁弹性. . . . .	273
创建查找转换. . . . .	274
创建可重用管道查找转换. . . . .	275
创建不可重用管道查找转换. . . . .	275
查找转换提示. . . . .	275
 第 18 章：查找缓存. . . . .	 277
查找缓存概览. . . . .	277
缓存比较. . . . .	278
构建连接的查找缓存. . . . .	279
顺序缓存. . . . .	279
并发缓存. . . . .	279
使用持久性查找缓存. . . . .	280
使用非持久性缓存. . . . .	280
使用持久性缓存. . . . .	280
重建查找缓存. . . . .	280
使用未缓存的查找或静态缓存. . . . .	281
共享查找缓存. . . . .	281
共享未命名查找缓存. . . . .	282
共享命名的查找缓存. . . . .	283
查找缓存的提示. . . . .	286

第 19 章：动态查找缓存.....	287
动态查找缓存概览.....	287
动态查找属性.....	288
NewLookupRows.....	288
关联表达式.....	289
空值.....	290
在比较中忽略端口.....	291
SQL 替代.....	291
查找转换值.....	291
初始缓存值.....	292
输入值.....	292
查找值.....	292
输出值.....	293
动态查找缓存更新.....	294
插入 Else 更新.....	294
更新 Else 插入.....	295
具有动态查找的映射.....	295
配置上游更新策略转换.....	295
配置下游转换.....	296
使用动态查找缓存配置会话.....	297
条件动态缓存更新.....	297
会话处理.....	297
配置条件动态缓存查找.....	297
表达式结果的动态缓存更新.....	298
空表达式值.....	298
会话处理.....	298
配置动态缓存更新的表达式.....	298
同步缓存与查找源.....	299
NewLookupRow.....	299
配置动态缓存同步.....	300
动态查找缓存示例.....	300
动态查找缓存的规则和准则.....	300
第 20 章：规范器转换.....	302
规范器转换概览.....	302
规范器转换要素.....	303
“端口”选项卡.....	303
“属性”选项卡.....	304
“规范器”选项卡.....	304
规范器转换生成的键.....	305
存储生成的键值.....	306
更改生成的键值.....	306

VSAM 规范器转换. . . . .	306
“VSAM 规范器端口” 选项卡. . . . .	308
“VSAM 规范器” 选项卡. . . . .	308
创建 VSAM 规范器转换的步骤. . . . .	309
管道规范器转换. . . . .	310
管道规范器 “端口” 选项卡. . . . .	311
管道 “规范器” 选项卡. . . . .	312
创建管道规范器转换的步骤. . . . .	313
在映射中使用规范器转换. . . . .	313
生成键值. . . . .	315
规范器转换故障排除. . . . .	316
 第 21 章： 等级转换. . . . .	 318
等级转换概览. . . . .	318
为字符串值评级. . . . .	319
等级缓存. . . . .	319
等级转换属性. . . . .	319
等级转换中的端口. . . . .	319
等级索引. . . . .	320
定义组. . . . .	320
创建等级转换. . . . .	321
 第 22 章： 路由器转换. . . . .	 323
路由器转换概览. . . . .	323
使用组. . . . .	324
输入组. . . . .	324
输出组. . . . .	324
使用组筛选条件. . . . .	325
添加组. . . . .	326
使用端口. . . . .	326
在映射中连接路由器转换. . . . .	327
创建路由器转换. . . . .	327
 第 23 章： 序列生成器转换. . . . .	 328
序列生成器转换概览. . . . .	328
序列生成器端口. . . . .	329
传递端口. . . . .	329
NEXTVAL 端口. . . . .	329
CURRVAL. . . . .	332
序列生成器转换属性. . . . .	334
起始值. . . . .	335
增量. . . . .	335
结束值. . . . .	336



增量值. . . . .	336
在值范围内循环. . . . .	336
当前值. . . . .	336
缓存值数. . . . .	337
不可重用的序列生成器. . . . .	337
可重用序列生成器. . . . .	337
重置. . . . .	338
保持行顺序. . . . .	338
序列数据对象. . . . .	338
创建序列数据对象. . . . .	338
创建序列生成器转换. . . . .	340
创建序列生成器转换. . . . .	341
常见问题. . . . .	342
非本地环境下的序列生成器转换. . . . .	342
Blaze 引擎上的序列生成器转换. . . . .	343
Spark 引擎上的序列生成器转换. . . . .	343
<b>第 24 章：排序器转换. . . . .</b>	<b>344</b>
排序器转换概览. . . . .	344
数据排序. . . . .	344
排序器转换属性. . . . .	345
排序器缓存大小. . . . .	345
区分大小写. . . . .	346
工作目录. . . . .	346
相异输出行. . . . .	346
跟踪级别. . . . .	346
空值视为低值. . . . .	346
转换范围. . . . .	347
创建排序器转换. . . . .	347
<b>第 25 章：源限定符转换. . . . .</b>	<b>348</b>
源限定符转换概览. . . . .	348
转换数据类型. . . . .	349
目标加载顺序. . . . .	349
日期时间值. . . . .	349
参数和变量. . . . .	349
源限定符转换属性. . . . .	350
默认查询. . . . .	351
查看默认查询. . . . .	351
替代默认查询. . . . .	351
联接源数据. . . . .	352
默认联接. . . . .	352
自定义联接. . . . .	353

异构联接. . . . .	353
创建键关系. . . . .	353
添加 SQL 查询. . . . .	354
输入用户定义的联接. . . . .	355
外部联接支持. . . . .	355
Informatica 联接语法. . . . .	355
创建外部联接. . . . .	360
常见数据库语法限制. . . . .	361
输入源筛选器. . . . .	361
使用排序端口. . . . .	362
选择相异. . . . .	363
在会话中替代为选择相异. . . . .	363
添加会话前和会话后 SQL 命令. . . . .	363
创建源限定符转换. . . . .	364
手动创建源限定符转换. . . . .	364
配置源限定符转换选项. . . . .	364
源限定符转换故障排除. . . . .	365
 第 26 章: SQL 转换. . . . .	 366
SQL 转换概览. . . . .	366
脚本模式. . . . .	367
示例. . . . .	367
脚本模式的规则和准则. . . . .	367
查询模式. . . . .	368
使用静态 SQL 查询. . . . .	368
使用动态 SQL 查询. . . . .	370
传递端口配置. . . . .	371
被动模式配置. . . . .	372
查询模式的规则和准则. . . . .	372
连接到数据库. . . . .	372
使用静态数据库连接. . . . .	373
传递逻辑数据库连接. . . . .	373
传递完整连接信息. . . . .	373
数据库连接的规则和准则. . . . .	375
会话处理. . . . .	375
事务控制. . . . .	376
高可用性. . . . .	376
SQL 查询日志. . . . .	378
输入行至输出行基数. . . . .	378
查询语句处理. . . . .	378
受影响的行数. . . . .	378
最大输出行计数. . . . .	379
了解错误行. . . . .	379

出现 SQL 错误时继续 . . . . .	381
SQL 转换属性 . . . . .	381
“属性”选项卡 . . . . .	381
“SQL 设置”选项卡 . . . . .	383
“SQL 端口”选项卡 . . . . .	383
SQL 语句 . . . . .	384
创建 SQL 转换 . . . . .	385
<b>第 27 章：在映射中使用 SQL 转换 . . . . .</b>	<b>386</b>
SQL 转换示例概览 . . . . .	386
动态更新示例 . . . . .	386
定义源文件 . . . . .	387
创建目标定义 . . . . .	388
创建数据库表 . . . . .	388
配置表达式转换 . . . . .	388
定义 SQL 转换 . . . . .	389
配置会话属性 . . . . .	391
目标数据结果 . . . . .	391
动态连接示例 . . . . .	391
定义源文件 . . . . .	392
创建目标定义 . . . . .	392
创建数据库表 . . . . .	392
创建数据库连接 . . . . .	393
配置表达式转换 . . . . .	393
定义 SQL 转换 . . . . .	394
配置会话属性 . . . . .	395
目标数据结果 . . . . .	395
<b>第 28 章：存储过程转换 . . . . .</b>	<b>396</b>
存储过程转换概览 . . . . .	396
输入和输出数据 . . . . .	397
已连接和未连接 . . . . .	397
指定何时运行存储过程 . . . . .	398
在映射中使用存储过程 . . . . .	399
编写存储过程 . . . . .	399
存储过程示例 . . . . .	400
创建存储过程转换 . . . . .	401
导入存储过程 . . . . .	402
手动创建存储过程转换 . . . . .	402
设置存储过程的选项 . . . . .	403
更改存储过程 . . . . .	404
配置连接转换 . . . . .	405
配置未连接转换 . . . . .	405

从表达式调用存储过程. . . . .	405
调用会话前或会话后存储的过程. . . . .	407
错误处理. . . . .	408
会话前错误. . . . .	408
会话后错误. . . . .	409
会话错误. . . . .	409
受支持的数据库. . . . .	409
SQL 声明. . . . .	409
参数类型. . . . .	409
映射中的输入/输出端口. . . . .	409
受支持的返回值的类型. . . . .	410
表达式规则. . . . .	410
有关存储过程转换的提示. . . . .	410
排除存储过程转换的故障. . . . .	411
 第 29 章：事务控制转换. . . . .	 412
事务控制转换概览. . . . .	412
事务控制转换属性. . . . .	412
“属性”选项卡. . . . .	413
示例. . . . .	413
在映射中使用事务控制转换. . . . .	414
包含多个目标的事务控制映射示例. . . . .	415
映射准则和验证. . . . .	416
创建事务控制转换. . . . .	416
 第 30 章：联合转换. . . . .	 417
联合转换概览. . . . .	417
联合转换的规则和准则. . . . .	417
联合转换组件. . . . .	417
使用组和端口. . . . .	418
创建联合转换. . . . .	418
在映射中使用联合转换. . . . .	418
 第 31 章：非结构化数据转换. . . . .	 420
非结构化数据转换概览. . . . .	420
配置非结构化数据选件. . . . .	421
配置 Data Transformation 存储库目录. . . . .	421
Data Transformation 服务类型. . . . .	422
非结构化数据转换组件. . . . .	422
“属性”选项卡. . . . .	422
“UDT 设置”选项卡. . . . .	423
查看状态跟踪消息. . . . .	424
非结构化数据转换端口. . . . .	425

输入和输出类型. . . . .	425
附加非结构化数据转换端口. . . . .	426
从 Data Transformation 服务中创建端口. . . . .	426
非结构化数据转换服务名称. . . . .	427
关系层次结构. . . . .	427
导出层次结构架构. . . . .	427
映射. . . . .	428
解析关系表的 Word 文档. . . . .	428
通过 XML 创建 Excel 工作表. . . . .	428
分割 XML 文件输出. . . . .	429
非结构化数据映射的规则和准则. . . . .	429
创建非结构化数据转换. . . . .	430
 第 32 章：更新策略转换. . . . .	 431
更新策略转换概览. . . . .	431
设置更新策略. . . . .	431
在映射中标记行. . . . .	432
转发拒绝的行. . . . .	432
更新策略表达式. . . . .	432
汇总器和更新策略转换. . . . .	433
查找和更新策略转换. . . . .	433
为会话设置更新策略. . . . .	433
为所有行指定一项操作. . . . .	433
为各个目标表指定操作. . . . .	434
更新策略清单. . . . .	435
 第 33 章：XML 转换. . . . .	 436
XML 源限定符转换. . . . .	436
XML 解析器转换. . . . .	436
XML 生成器转换. . . . .	437
 索引. . . . .	 438

# 前言

要学习 Informatica 转换的配置、指南、用法以及运行时行为，请阅读《PowerCenter® 转换指南》。转换是存储库对象，代表了集成服务对数据执行的操作。

请参考本指南执行生成、更改或传输数据的数据操作。根据计划运行映射的位置和方式，查看对每种转换的支持。数据会通过转换端口传递至您在映射或 Maplet 中链接的目标和其他转换。您可以在活动转换中使用表达式编辑器输入表达式。转换是一组用于定义一行传入或传出数据的端口，可以具有多个输入和输出组。

## Informatica 资源

Informatica 通过 Informatica Network 和其他在线门户为您提供一系列产品资源。使用这些资源，可以充分利用 Informatica 产品和解决方案，并向其他 Informatica 用户和主题专家学习。

### Informatica Network

在 Informatica Network 中可以获得许多资源，包括 Informatica 知识库和 Informatica 全球客户支持。要进入 Informatica Network，请访问 <https://network.informatica.com>。

作为 Informatica Network 成员，您可以选择以下服务：

- 在知识库中搜索产品资源。
- 查看产品可用性信息。
- 创建并检查您的支持案例。
- 查找当地的 Informatica 用户组网络并与您的伙伴进行协作。

### Informatica 知识库

使用 Informatica 知识库可查找产品资源，例如操作方法文章、最佳实践、视频教程以及常见问题的答案。

要搜索知识库，请访问 <https://search.informatica.com>。如果您对知识库有任何疑问、意见或建议，请与 Informatica 知识库团队联系，电子邮件地址为 [KB\\_Feedback@informatica.com](mailto:KB_Feedback@informatica.com)。

### Informatica 文档

使用 Informatica 文档门户可浏览大量当前与最近产品版本的文档库。要浏览文档门户，请访问 <https://docs.informatica.com>。

如果您对产品文档有任何疑问、意见或建议，请与 Informatica 文档团队联系，电子邮件地址为 [infa\\_documentation@informatica.com](mailto:infa_documentation@informatica.com)。

## Informatica 产品可用性矩阵

产品可用性矩阵 (PAM) 指明了产品版本支持的操作系统版本、数据库以及数据源和目标类型。您可以在以下网址中浏览 Informatica PAM:

<https://network.informatica.com/community/informatica-network/product-availability-matrices>。

## Informatica Velocity

Informatica Velocity 是由 Informatica 专业服务根据数百个数据管理项目的实际经验所开发出来的，其中汇集了大量使用技巧和最佳实践。Informatica Velocity 代表了 Informatica 顾问的集体知识，这些顾问与世界各地的组织合作，共同计划、开发、部署和维护成功的数据管理解决方案。

您可以在以下网址中找到 Informatica Velocity 资源：<http://velocity.informatica.com>。如果您对 Informatica Velocity 有任何疑问、意见或建议，请通过 [ips@informatica.com](mailto:ips@informatica.com) 与 Informatica 专业服务联系。

## Informatica Marketplace

Informatica Marketplace 是一个论坛，该论坛中提供的解决方案可扩展和增强您的 Informatica 实施。利用 Informatica 开发人员和合作伙伴在 Marketplace 中提供的数以百计的解决方案，可提高您的工作效率并加快项目实施时间。您可以在以下网址中找到 Informatica Marketplace：<https://marketplace.informatica.com>。

## Informatica 全球客户支持部门

您可以通过电话或 Informatica Network 与全球支持中心联系。

要查找您当地的 Informatica 全球客户支持部门电话号码，请访问 Informatica 网站，链接为：<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>。

要在 Informatica Network 上查找在线支持资源，请访问 <https://network.informatica.com>，然后选择 eSupport 选项。

# 第 1 章

## 使用转换

本章包括以下主题：

- [转换概览, 24](#)
- [创建转换, 28](#)
- [配置转换, 28](#)
- [转换端口, 29](#)
- [多组转换, 29](#)
- [使用表达式, 30](#)
- [局部变量, 34](#)
- [端口的默认值, 36](#)
- [配置转换中的跟踪级别, 42](#)
- [可重用转换, 43](#)

## 转换概览

转换是一种用于生成、修改或传递数据的存储库对象。Designer 提供了一组执行特定功能的转换。例如，汇总器转换执行数据组的计算。

映射中的转换代表集成服务对数据执行的操作。数据通过您在映射或 Mapplet 中链接的转换端口传递。

转换可以是以下类型之一：

- 主动或被动
- 已连接或未连接
- 本地或非本地

## 主动转换

主动转换可执行以下任何操作：

- **更改传递转换的行数。**例如，筛选器转换是主动转换，因为其会删除不满足筛选条件的行。所有多组转换都是主动转换，因为它们可能会更改传递转换的行数。
- **更改事务边界。**例如，事务控制转换是主动转换，因为其基于为每一行计算的表达式定义提交或回滚事务。
- **更改行类型。**例如，更新策略转换是主动转换，因为其会标记要插入、删除、更新或拒绝的行。



Designer 不允许将多个主动转换或者一个主动转换和一个被动转换连接到同一个下游转换或转换输入组，因为集成服务可能无法将由主动转换传递的多个行连接起来。例如，映射中的一个分支包含标记要删除的行的更新策略转换。另一个分支包含标记要插入的行的更新策略转换。如果将这些转换连接到一个转换输入组，则集成服务无法将行的删除和插入操作组合在一起。

序列生成器转换不适用于该规则。Designer 确实允许将一个序列生成器转换和一个主动转换连接到同一个下游转换或转换输入组。序列生成器转换不接收数据，而是生成唯一的数值。因此，集成服务不会遇到将由序列生成器转换和主动转换传递的多个行连接起来的问题。

## 被动转换

被动转换不更改通过转换传递的行数，并保持事务边界和行类型。

如果上游分支中的所有转换都是被动转换，可以将多个转换连接到同一个下游转换或同一个转换输入组。源自该分支的转换可以是主动转换也可以是被动转换。

## 未连接的转换

转换可以连接到数据流，也可以取消转换的连接。未连接转换不连接到映射中的其他转换。未连接转换在其他转换内调用，并向该转换返回一个值。

## 本地转换和非本地转换

本地转换是 Designer 提供的一组转换。非本地转换是您使用自定义转换创建的转换。Designer 还提供某些非本地转换，如 Java、SQL 和联合转换。应用于自定义转换的规则也应用于使用自定义转换构建的非本地转换。

## 转换说明

下表提供了各种转换的简要说明：

转换	类型	说明
汇总器	<ul style="list-style-type: none"><li>- 活动</li><li>- 已连接</li><li>- 本地</li></ul>	执行聚合计算。
应用程序源限定符	<ul style="list-style-type: none"><li>- 活动</li><li>- 已连接</li><li>- 非本地</li></ul>	代表集成服务在运行会话时从应用程序（如 ERP 源）中读取的行。
自定义	<ul style="list-style-type: none"><li>- 主动或被动</li><li>- 已连接</li><li>- 非本地</li></ul>	在共享库或 DLL 中调用某一过程。
数据屏蔽	<ul style="list-style-type: none"><li>- 被动</li><li>- 已连接</li><li>- 非本地</li></ul>	将敏感生产数据替换为非生产环境中的实际测试数据。
表达式	<ul style="list-style-type: none"><li>- 被动</li><li>- 已连接</li><li>- 本地</li></ul>	计算值。

转换	类型	说明
外部过程	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接或未连接</li> <li>- 本地</li> </ul>	在共享库或 Windows 的 COM 层中调用某一过程。
筛选器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	筛选数据。
HTTP	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接</li> <li>- 非本地</li> </ul>	连接到 HTTP 服务器以读取或更新数据。
输入	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	定义 Mapplet 输入行。可在 Mapplet Designer 中使用。
Java	<ul style="list-style-type: none"> <li>- 主动或被动</li> <li>- 已连接</li> <li>- 非本地</li> </ul>	执行使用 Java 编码的用户逻辑。用户逻辑的字节代码存储在存储库中。
联接器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	联接来自不同数据库或平面文件系统的数据。
查找	<ul style="list-style-type: none"> <li>- 主动或被动</li> <li>- 已连接或未连接</li> <li>- 本地</li> </ul>	从平面文件、关系表、视图或同义词中查找并返回数据。
规范化程序	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	COBOL 源的源限定符。还可以在管道中使用，以规范化数据关系源或平面文件源中的数据。
输出	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	定义 Mapplet 输出行。可在 Mapplet Designer 中使用。
等级	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	将记录数限制到范围的最高或最低值。
路由器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	根据组条件将数据路由到多个转换。
序列生成器	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	生成主键。
排序器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	根据排序键对数据进行排序。

转换	类型	说明
源限定符	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	代表集成服务在运行会话时从关系源或平面文件源中读取的行。
SQL	<ul style="list-style-type: none"> <li>- 主动或被动</li> <li>- 已连接</li> <li>- 非本地</li> </ul>	对数据库执行 SQL 查询。
存储过程	<ul style="list-style-type: none"> <li>- 被动</li> <li>- 已连接或未连接</li> <li>- 本地</li> </ul>	调用存储过程。
事务控制	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	定义提交事务和回滚事务。
联合	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 非本地</li> </ul>	合并来自不同数据库或平面文件系统的数据。
非结构化数据	<ul style="list-style-type: none"> <li>- 主动或被动</li> <li>- 已连接</li> <li>- 非本地</li> </ul>	以非结构化和半结构化格式转换数据。
更新策略	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	确定是否插入、删除、更新或拒绝行。
XML 生成器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	从一个或多个输入端口读取数据，然后通过一个输出端口输出 XML。
XML 解析器	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	从一个输入端口读取 XML，然后向一个或多个输出端口输出数据。
XML 源限定符	<ul style="list-style-type: none"> <li>- 活动</li> <li>- 已连接</li> <li>- 本地</li> </ul>	代表集成服务在运行会话时从 XML 源中读取的行。

在构建映射时，添加转换并对将转换配置为根据业务目的处理数据。完成以下任务以将转换合并到映射中：

1. **创建转换。**在 Mapping Designer 中作为映射的组成部分、在 Mapplet Designer 中作为 Mapplet 的组成部分，或在 Transformation Developer 中作为可重用转换的组成部分，创建转换。
2. **配置转换。**每种类型的转换都有唯一一组可以配置的选项。
3. **将转换链接到其他转换和目标定义。**将一个端口拖动到另一个端口，以在映射或 Mapplet 中链接这两个端口。

# 创建转换

可以使用以下 Designer 工具创建转换：

- **Mapping Designer**。创建将源连接到目标的转换。一个映射中的转换不能用于其他映射，除非将它们配置为可重用。
- **Transformation Developer**。创建用于多个映射的单个转换（称为可重用转换）。
- **Mapplet Designer**。创建和配置用于多个映射的一组转换（称为 Mapplet）。

在 Mapping Designer、Transformation Developer 和 Mapplet Designer 中可按照相同的过程创建转换。

要创建转换，请执行以下操作：

1. 打开相应的 Designer 工具。
2. 在 Mapping Designer 中，打开或创建一个映射。在 Mapplet Designer 中，打开或创建一个 Mapplet。
3. 单击“转换”>“创建”，然后选择要创建的转换类型。
4. 拖动到映射中希望放置转换的部分。

此时，该新转换将显示在工作区中。接下来，需要通过在其中添加任何新端口及设置其他属性来配置该转换。

# 配置转换

创建转换后可对其进行配置。每个转换均包含以下常见选项卡：

- **转换**。命名转换或添加说明。
- **端口**。添加和配置端口。
- **属性**。配置对转换唯一的属性。

某些转换可能包含其他选项卡，如“条件”选项卡，您可以在其中输入连接器或规范器转换中的条件。

配置转换时，可完成以下任务：

- **添加端口**。定义移入和移出转换的数据的列。
- **添加组**。在某些转换中，定义指示一行数据进入或离开转换的输入或输出组。
- **输入表达式**。在某些转换中输入转换数据的 SQL 类表达式。
- **定义局部变量**。在某些转换中定义临时存储数据的局部变量。
- **替代默认值**。配置端口的默认值以处理输入空值和输出转换错误。
- **输入跟踪级别**。选择集成服务在会话日志中写入的关于转换的详细信息量。

# 重命名转换

要重命名转换，请单击“重命名”按钮并为转换输入描述性名称，然后单击“确定”。

# 转换端口

创建转换后，可定义端口。创建端口并定义端口属性。

创建某些转换时，不必手动创建所有端口。例如，可以创建查找转换并引用查找表。如果查看转换端口，可以看到转换针对您引用的表中的每列都有一个输出端口。无需定义这些端口。

## 创建端口

创建某些转换时，不必手动创建所有端口。例如，可以创建查找转换并引用查找表。如果查看转换端口，可以看到转换针对您引用的表中的每列都有一个输出端口。无需定义这些端口。

请按照以下方法创建端口：

- **从其他转换中拖动端口。**从其他转换中拖动端口时，Designer 将创建一个具有相同属性的端口，并链接起两个端口。单击“布局”>“复制列”以便复制端口。
- **在“端口”选项卡上单击“添加”按钮。**Designer 将创建一个可以配置的空端口。

## 配置端口

定义转换端口时，可定义端口属性。端口属性包括端口名称、数据类型、端口类型和默认值。

配置以下端口属性：

- **端口名称。**端口的名称。命名端口时，请遵循以下约定：
  - 以单字节或双字节字母或者单字节或双字节的下划线 ( \_ ) 开头。
  - 端口名称可以包含以下任意单字节或双字节字符：字母、数字、下划线 ( \_ )、\$、# 或 @。
- **数据类型、精度和小数位数。**如果您计划输入表达式或条件，请验证数据类型是否与表达式的返回值匹配。
- **端口类型。**转换可包含输入、输出、输入/输出和变量端口类型的组合。
- **默认值。**为包含空值或输出转换错误的端口分配一个默认值。可以替代某些端口中的默认值。
- **说明。**端口的说明。
- **其他属性。**有些转换具有特定于该转换的属性，例如表达式或按属性组合。

## 链接端口

在映射中添加和配置转换后，将其链接到目标和其他转换。通过端口链接映射对象。数据通过以下端口传入和传出映射：

- **输入端口。**接收数据。
- **输出端口。**传递数据。
- **输入/输出端口。**接收数据并将其原封不动地传递出去。

要链接端口，请在不同的映射对象之间拖动端口。Designer 会验证链接并且仅在链接符合验证要求时创建链接。

# 多组转换

转换可以包含多个输入和输出组。组是指定义一行传入或传出数据的端口组。

组类似于关系源或目标定义中的表。大多数转换都有一个输入组和一个输出组。不过，有些包含多个输入组、多个输出组或两者。组是一行数据进入或离开转换的表示形式。

所有多组转换都是主动转换。不能将多个主动转换或一个主动和一个被动转换连接到同一个下游转换或转换输入组。

有些多输入组转换要求集成服务在等待来自其他输入组的行时对输入组中的数据编块。编块转换指对传入数据进行编块的多输入组转换。以下转换为编块转换：

- 启用“输入可编块”属性的自定义转换
- 为未排序的输入配置的联接器转换

在保存或验证映射时，有些包含主动或编块转换的映射可能无效。

# 使用表达式

可以在某些转换中使用表达式编辑器来输入表达式。使用以下函数创建表达式：

- **转换语言函数。**类似于 SQL 的函数，用于处理通用表达式。
- **用户定义的函数。**您在 PowerCenter 中根据转换语言函数创建的函数。
- **自定义函数。**使用自定义函数 API 创建的函数。

在使用来自输入端口或输入/输出端口的数据值的端口中输入表达式。例如，您有一个使用输入端口 IN\_SALARY 的转换，该端口包含所有员工的薪酬数据。以后可以在映射中使用 IN\_SALARY 列中的值，以及您通过此转换计算的总薪酬和平均薪酬。因此，Designer 要求您为每个计算的值创建一个单独的输出口。

下表列出了可在其中输入表达式的转换：

转换	表达式	返回值
汇总器	基于通过转换传递的所有数据执行聚合计算。或者，可以为聚合计算中的记录指定筛选器，以排除某些类型的记录。例如，可以使用该转换得出分公司所有员工的总人数和平均薪酬。	端口的聚合计算结果。
数据屏蔽	根据某一行的输入或输出端口的值执行计算。表达式是在数据屏蔽转换中屏蔽生产数据的一种方法。	使用输入或输出端口的行级计算的结果。
表达式	基于单个行内的值执行计算。例如，根据特殊物品的价格和数量，您可以计算订单中该行物品的总采购价格。	端口的行级计算结果。
筛选器	指定用于筛选通过该转换传递的行的条件。例如，如果要具有未结余额的客户的客户数据写入到 BAD_DEBT 表中，可以使用筛选器转换筛选客户数据。	TRUE 或 FALSE（取决于行是否满足指定条件）。只有返回 TRUE 的行才会传递此转换。该转换将此值应用到通过其传递的每个行。
等级	设置包含在某一等级中的行的条件。例如，可以设定就职于公司的前 10 位销售人员的等级。	端口的条件或计算结果。

转换	表达式	返回值
路由器	根据组表达式将数据路由到多个转换。例如，使用该转换比较处于三个不同薪酬水平的员工的薪酬。这可以通过在路由器转换中创建三个组来实现。例如，为各个薪酬范围创建一个组表达式。	TRUE 或 FALSE（取决于行是否满足指定的组表达式）。只有返回 TRUE 的行才会传递此转换中每个用户定义的组。返回 FALSE 的行通过默认组传递。
更新策略	将行标记为更新、插入、删除或拒绝。如果要根据应用的部分条件控制对目标的更新，可使用该转换。例如，当邮寄地址更改时，可以使用更新策略转换将所有客户行标记为要进行更新；或将不再就职于公司的人员的所有员工行标记为要拒绝。	更新、插入、删除或拒绝的数字代码。该转换将此值应用到通过其传递的每个行。
事务控制	指定用于确定集成服务执行操作（提交、回滚或无事务更改）的条件。如果想要根据传递转换的某一行或行集合来控制提交事务和回滚事务，则可使用此转换。例如，使用此转换根据订单输入日期提交一组行。	以下内置变量之一，具体取决于某一行是否满足指定条件： <ul style="list-style-type: none"> <li>- TC_CONTINUE_TRANSACTION</li> <li>- TC_COMMIT_BEFORE</li> <li>- TC_COMMIT_AFTER</li> <li>- TC_ROLLBACK_BEFORE</li> <li>- TC_ROLLBACK_AFTER</li> </ul> 集成服务根据返回值执行操作。

## 使用表达式编辑器

使用表达式编辑器构建类似 SQL 的语句。虽然可以手动输入表达式，但请使用点击式方法。从点击式界面选择函数、端口、变量和运算符，以便在构建表达式时最大程度地减少错误。在表达式中可以包含的最大字符数为 32,767。

可以计算在表达式转换的表达式编辑器中配置的表达式。

测试某个表达式时，可以输入示例数据，然后计算该表达式。在 Transformation Designer 中计算可重用转换中的表达式。在 Mapping Designer 中计算不可重用转换中的表达式。当您端口数据类型设置为二进制或日期/时间时，某些函数不支持计算表达式。

## 将端口名称输入表达式

对于已连接的转换，如果在表达式中使用端口名称，则当您更改转换中的端口名称时，Designer 会更新该表达式。例如，写入确定 Date\_Promised 与 Date\_Delivered 这两个日期之差的有效表达式。然后，如果您将 Date\_Promised 端口名称更改为 Due\_Date，Designer 会将表达式中的 Date\_Promised 端口名称更改为 Due\_Date。

**注意：**可以将名称 Due\_Date 传播到依赖于映射中的此端口的其他不可重用转换中。

## 添加注释

可以在表达式中添加注释，以提供有关表达式的描述性信息或指定用于访问有关表达式的业务文档的有效 URL。

可以通过以下方式添加注释：

- 要在表达式中添加注释，请使用 -- 或 // 注释指示符。
- 要在对话框中添加注释，请单击“注释”按钮。

## 验证表达式

使用“验证”按钮验证表达式。如果您不验证表达式，Designer 会在您关闭表达式编辑器的时候对其进行验证。如果表达式无效，Designer 会显示一条警告。可以保存无效的表达式，或者对其进行修改。不能针对包含无效表达式的映射运行会话。

## 表达式编辑器显示

表达式编辑器可以不同颜色显示语法表达式，以提高可读性。如果您在系统中安装了最新的 Rich Edit 控件 riched20.dll，则表达式编辑器将以蓝色显示表达式函数、以灰色显示注释并以绿色显示加引号的字符串。

可以调整表达式编辑器的大小。通过从边界拖动可以展开对话框。Designer 会将对话框的新大小保存为一个客户端设置。

## 将表达式添加到端口

在数据屏蔽转换中，可以将表达式添加到输入端口中。对于所有其他转换，可将表达式添加到输出端口中。

要向端口添加表达式，请完成以下步骤：

1. 在转换中，选择端口并打开表达式编辑器。
2. 输入表达式。  
使用“函数”和“端口”选项卡以及运算符键。
3. 在表达式中添加注释。  
使用注释指示符 `--` 或 `//`。
4. 验证表达式。  
使用“验证”按钮验证表达式。

## 在参数文件中定义表达式字符串

集成服务在解析表达式后可展开映射参数和变量。如果您有一个表达式要频繁更改，可以在参数文件中定义该表达式，这样在该表达式更改时则不必更新使用它的映射。

要在参数文件中定义表达式字符串，可以创建映射参数或变量来存储该表达式字符串，并将该参数或变量设置为参数文件中的表达式字符串。对于您创建的参数或变量，必须将 `IsExprVar` 设置为 `true`。当 `IsExprVar` 为 `true` 时，集成服务将在解析表达式之前展开该参数或变量。

例如，要在参数文件中定义表达式 `IIF(color= 'red' ,5)`，请执行以下步骤：

1. 在使用该表达式的映射中，创建映射参数 `$$Exp`。将 `IsExprVar` 设置为 `true`，并将“数据类型”设置为“字符串”。
2. 在表达式编辑器中，将该表达式设置为该映射参数的名称，如下所示：  
`$$Exp`
3. 配置会话或工作流以使用参数文件。
4. 在该参数文件中，将 `$$Exp` 的值设置为该表达式字符串，如下所示：  
`$$Exp=IIF(color= 'red' ,5)`

## 计算表达式

可以计算在表达式转换的表达式编辑器中配置的表达式。测试某个表达式时，可以输入示例数据，然后计算该表达式。

如果编辑了表达式条件，则必须在测试面板中选择“刷新”，然后才能计算测试数据。如果没有输入有效的表达式，则测试面板将无法填充端口。指定系统定义的参数时，输入的值不是运行时值。



下面的图像显示了如何在表达式转换的表达式编辑器中计算示例表达式：



**注意:** 不能计算日期/时间或二进制数据类型的表达式。

### 示例

在将所有结果加载到目标之前，您需要根据每个客户收到的订单总数来计算促销优惠。您可以通过在表达式转换中定义表达式来创建一个映射，以便计算优惠。在运算符运行映射之前，您可以计算表达式以验证结果。

## 计算表达式

可以在表达式转换的表达式编辑器中测试和验证表达式。

1. 在表达式转换的表达式编辑器中，输入表达式。
2. 要验证表达式，请单击“验证”。
3. 若要在右窗格中的“测试表达式”部分反映表达式条件的最新更改，请单击“刷新”。
4. 在右窗格中，输入表达式中使用的输入端口的示例值。
5. 要计算表达式，请单击“计算”。

## 计算表达式限制

计算表达式时，会存在某些限制。

计算表达式时，请考虑下列限制：

### 转换限制

您无法在以下转换中计算表达式：

- 汇总器
- 数据屏蔽
- 筛选器
- 等级
- 路由器
- 存储过程
- 事务控制
- 更新策略

数据类型限制

如果输入端口类型或返回函数包含二进制或日期/时间数据类型，则无法计算表达式。

端口限制

无法计算使用查找或存储过程端口的表达式。

# 局部变量

在汇总器、表达式和等级转换中使用局部变量可提高性能。可以在表达式中引用变量或使用它们临时存储数据。

可以使用变量完成以下任务：

- 临时存储数据。
- 简化复杂表达式。
- 存储来自以前行的值。
- 捕获来自存储过程的多个返回值。
- 比较值。
- 存储未连接的查找转换的结果。

## 临时存储数据和简化复杂表达式

在同一转换中输入多个相关表达式时，变量可提高性能。可以将组件定义为变量，而不必在转换中多次解析和验证相同的表达式组件。

例如，如果汇总器转换在计算总和和平均值以前使用相同的筛选器条件，则可以将此条件定义为变量，然后在两项汇总计算中重用该条件。

可以简化复杂表达式。如果汇总器在多个表达式中的计算相同，则可以通过创建变量来存储计算结果而提高性能。

例如，您可以创建以下表达式来查找相同数据的平均薪资和总薪资。

```
AVG( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )  
SUM( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )
```

无需为两种计算输入相同的参数，可以为此计算中的每个条件创建一个变量端口，然后将表达式改为使用变量。

下表显示了如何使用变量简化复杂表达式和临时存储数据：

端口	值
V_CONDITION1	JOB_STATUS = ‘Full-time’
V_CONDITION2	OFFICE_ID = 1000
AVG_SALARY	AVG(SALARY, (V_CONDITION1 AND V_CONDITION2) )
SUM_SALARY	SUM(SALARY, (V_CONDITION1 AND V_CONDITION2) )

# 存储行中的值

您可以在转换中配置变量，以存储源行的数据。在转换表达式中可以使用变量。

例如，源文件包含以下行：

```
California
California
California
Hawaii
Hawaii
New Mexico
New Mexico
New Mexico
```

每行包含一个省/自治区/直辖市。您需要统计行数并对每个省/自治区/直辖市返回行数：

```
California,3
Hawaii      ,2
New Mexico,3
```

您可以配置一个汇总器转换来按省/自治区/直辖市组合源行，并统计每组中的行数。在汇总器转换中配置变量，以存储行数。定义另一个变量来存储上一行的省/自治区/直辖市名称。

汇总器转换具有以下端口：

端口	端口类型	表达式	说明
省/自治区/直辖市	传递	n/a	省/自治区/直辖市的名称。源行按省/自治区/直辖市名称进行组合。汇总器转换针对每个省/自治区/直辖市返回一行。
State_Count	变量	IIF (PREVIOUS_STATE = STATE, STATE_COUNT +1, 1)	当前省/自治区/直辖市的行计数。在当前“省/自治区/直辖市”列的值与 Previous_State 列相同时，集成服务将对 State_Count 递增。否则，则将 State_Count 重置为 1。
Previous_State	变量	省/自治区/直辖市	上一行中“省/自治区/直辖市”列的值 当集成服务处理行时，会将“省/自治区/直辖市”值移到 Previous_State 中。
State_Counter	输出	State_Count	汇总器转换针对省/自治区/直辖市处理的行数。集成服务对于每个省/自治区/直辖市返回一次 State_Counter。

# 捕获来自存储过程的值

变量提供一种捕获来自存储过程的多列返回值的方式。

# 配置变量端口的准则

在转换中配置变量端口时，应考虑以下因素：

- **端口顺序。**集成服务按依赖关系计算端口。转换中端口的顺序必须与计算顺序匹配。输入端口、变量端口、输出端口。
- **数据类型。**所选的数据类型反映您输入的表达式返回值。
- **变量初始化。**集成服务设置变量端口的初始值，在变量端口中可创建计数器。

## 端口顺序

集成服务首先计算输入端口。接下来，集成服务计算变量端口，最后是输出端口。

集成服务按照以下顺序计算端口：

1. **输入端口。**集成服务首先计算所有输入端口，因为它们不依赖于任何其他端口。因此，可以按任何顺序创建输入端口。集成服务不会对输入端口排序，因为输入端口不引用其他端口。
2. **变量端口。**变量端口可引用输入端口和变量端口，但不引用输出端口。由于变量端口可引用输入端口，所以集成服务要在输入端口之后计算变量端口。变量可引用其他变量，所以变量端口的显示顺序与集成服务计算各个变量的顺序相同。

例如，如果要计算建筑物的原始价值，然后调整进行折旧，则可以创建原始价值计算作为变量端口。此变量端口需要出现在调整折旧的端口之前。

3. **输出端口。**集成服务最后计算输出端口，因为输出端口可引用输入端口和变量端口。输出端口的显示顺序无关紧要，因为输出端口无法引用其他输出端口。务必使输出端口显示在端口列表的最底部。

## 数据类型

将端口配置为变量时，可以在其中输入任意表达式或条件。为此端口选择的数据类型反映您输入的表达式返回值。如果通过变量端口指定条件，则任何数值数据类型将针对 TRUE（非零）和 FALSE（零）返回值。

## 变量初始化

集成服务不会将变量的初始值设置为空。

集成服务遵循以下准则设置变量初始值：

- 数值端口为零
- 字符串端口为空字符串
- 日期/时间端口为 01/01/0001

因此，使用变量作为计数器需要初始值。例如，可以使用以下表达式创建数值变量：

`VAR1 + 1`

此表达式将统计 VAR1 端口中的行数。如果将变量初始值设置为空，该表达式将始终计算为空。因此，应将初始值设置为零。

# 端口的默认值

所有转换均使用默认值，以确定集成服务如何处理输入空值和输出转换错误。

输入、输出和输入/输出端口都有系统默认值，有时可使用用户定义的默认值替代它们。默认值在不同类型的端口中有不同的功能：

- **输入端口。**空输入端口的系统默认值为空。默认值在转换中看起来为空白。如果输入值为空，则集成服务将其保留为空。
- **输出端口。**输出转换错误的系统默认值为 ERROR。默认值在转换中看起来为 ERROR（“转换错误”）。如果出现转换错误，则集成服务将跳过该行。集成服务将在日志文件中记入 ERROR 函数跳过的所有输入行。

以下错误为转换错误：

- 数据转换错误，例如将数字传递给日期函数。

- 表达式计算错误，例如除以零。
- 调用 ERROR 函数。
- **传递端口。**空输入的系统默认值与输入端口相同 — 空。系统默认值在转换中看起来为空白。输出转换错误的默认值与输出端口相同。转换中不显示输出转换错误的默认值。

**注意:** Java 转换会根据 Java 转换端口类型将 PowerCenter® 数据类型转换为 Java 数据类型。空输入的默认值视 Java 数据类型有所不同。

下表显示了已连接转换中端口的默认值：

端口类型	默认值	集成服务行为	支持用户定义的默认值
输入、传递	空值	集成服务以空值形式传递所有输入空值。	输入、输入/输出
输出、传递	ERROR	对于输出端口转换错误，集成服务将调用 ERROR 函数。集成服务将跳过错误行，并在日志文件中写入输入数据和错误消息。	输出

变量端口不支持默认值。集成服务将根据数据类型初始化变量端口。

在集成服务遇到空输入值和输出转换错误时，可以替代部分默认值来更改其行为。

## 用户定义的默认值

对于已连接转换内支持的输入、传递和输出端口，可以使用用户定义的默认值替代系统默认值。

要替代端口的系统默认值，请遵循以下规则和准则：

- **输入端口。**如果不希望集成服务将空值视为 NULL，可以为输入端口输入用户定义的默认值。如果 NULL 传递至输入端口，集成服务则使用默认值来替换 NULL。
- **输出端口。**如果不希望集成服务跳过行或希望集成服务在日志中对跳过的行写入特定消息，可以为输出端口输入用户定义的默认值。如果在输出端口中定义默认值，则在输出端口发生转换错误时，集成服务会将行替换为默认值。
- **传递端口。**如果不希望集成服务将空值视为 NULL，可以为传递端口输入用户定义的默认值。在传递端口中，不能针对输出转换错误输入用户定义的默认值。

**注意:** 对于未连接的转换，集成服务将忽略用户定义的默认值。例如，如果通过表达式调用查找或存储过程转换，集成服务将忽略任何用户定义的默认值并应用系统默认值。

使用以下选项输入用户定义的默认值：

- **常量值。**使用任意常量（数值或文本），包括 NULL。
- **常量表达式。**可以在转换函数中使用常量参数。
- **ERROR。**生成转换错误。在会话日志或行错误日志中写入行和消息。
- **ERROR。**生成转换错误。在映射日志或行错误日志中写入行和消息。
- **ABORT。**中止会话。
- **ABORT。**中止映射。

## 常量值

可以输入任意常量值作为默认值。常量值必须与端口数据类型匹配。

例如，数值端口的默认值必须是数值型常量。部分常量值包括：

```
0
9999
NULL
'Unknown Value'
'Null input data'
```

## 常量表达式

常量表达式是指使用转换函数（除汇总函数外）写入常量表达式的任何表达式。常量表达式中不能使用来自输入、输入/输出或变量端口的值。

部分有效的常量表达式包括：

```
500 * 1.75
TO_DATE('January 1, 1998, 12:05 AM', 'MONTH DD, YYYY, HH:MI AM')
ERROR ('Null not allowed')
ABORT ('Null not allowed')
SESSSTARTTIME
```

在表达式内不能使用来自端口的值，因为集成服务在初始化会话时会为整个映射分配默认值。

在表达式内不能使用来自端口的值，因为集成服务在初始化映射时会为整个映射分配默认值。

以下示例使用了来自端口的值，所以是无效的：

```
AVG(IN_SALARY)
IN_PRICE * IN_QUANTITY
:LKP(LKP_DATES, DATE_SHIPPED)
```

**注意：** 不能从默认值表达式调用存储过程或查找表。

## ERROR 和 ABORT 函数

对于输入和输出端口默认值及输入/输出端口的输入值，可使用 ERROR 和 ABORT 函数。当集成服务遇到 ERROR 函数时，它将跳过该行。当它遇到 ABORT 函数时，将中止会话。

对于输入和输出端口默认值及输入/输出端口的输入值，可使用 ERROR 和 ABORT 函数。当集成服务遇到 ERROR 函数时，它将跳过该行。当它遇到 ABORT 函数时，将中止映射。

## 用户定义的默认输入值

如果不希望集成服务将空值视为空，可以输入用户定义的默认输入值。

要替代空值，请完成以下任务之一：

- 将空值替换为常量值或常量表达式。
- 使用 ERROR 函数跳过空值。
- 使用 ABORT 函数中止映射。
- 使用 ABORT 函数中止会话。

下表汇总了集成服务如何处理输入和输入/输出端口的空值输入：

默认值	默认值类型	说明
空（显示空白）	系统	集成服务传递空值。
常量或常量表达式	用户定义	集成服务将空值替换为常量值或常量表达式。
ERROR	用户定义	集成服务将此情况视为转换错误： <ul style="list-style-type: none"> <li>- 转换错误计数增加 1。</li> <li>- 跳过该行，并向日志文件或行错误日志写入错误消息。</li> </ul> 集成服务不会将这些行写入拒绝文件。
中止	用户定义	当集成服务遇到空输入值时，会话将中止。集成服务不会增加错误计数或将这些行写入拒绝文件。 当集成服务遇到空输入值时，映射将中止。集成服务不会增加错误计数或将这些行写入拒绝文件。

### 替换空值

使用常量值或表达式以指定值替代端口的空值。

例如，如果输入字符串端口称为 DEPT\_NAME，您希望将空值替换为字符串“UNKNOWN DEPT”，则可以将默认值设置为“UNKNOWN DEPT”。集成服务会将“UNKNOWN DEPT”传递到转换内的表达式或变量或数据流中的下一个转换，具体视转换而定。

例如，集成服务可将端口中的所有空值替换为字符串“UNKNOWN DEPT”。

DEPT_NAME	REPLACED VALUE
Housewares	Housewares
NULL	UNKNOWN DEPT
Produce	Produce

### 跳过空记录

当不希望将空值传递到转换中时，可使用 ERROR 函数作为默认值。例如，当 DEPT\_NAME 的值为空时，您可能希望跳过某行。可以使用以下表达式作为默认值：

```
ERROR('Error. DEPT is NULL')
```

使用 ERROR 函数作为默认值时，集成服务将跳过具有空值的该行。集成服务会将 ERROR 函数跳过的所有行写入日志文件，而不会将这些行写入拒绝文件。

DEPT_NAME	RETURN VALUE
Housewares	Housewares
NULL	'Error. DEPT is NULL' (Row is skipped)
Produce	Produce

以下日志显示了集成服务跳过具有空值的行的位置：

```
TE_11019 Port [DEPT_NAME]: Default value is: ERROR(<<Transformation Error>> [error]: Error. DEPT is NULL
... error('Error. DEPT is NULL')
).
CMN_1053 EXPTRANS: : ERROR: NULL input column DEPT_NAME: Current Input data:
CMN_1053 Input row from SRCTRANS: Rowdata: ( RowType=4 Src Rowid=2 Targ Rowid=2
DEPT_ID (DEPT_ID:Int.): "2"
DEPT_NAME (DEPT_NAME:Char.25.): "NULL"
MANAGER_ID (MANAGER_ID:Int.): "1"
)
```

## 中止会话

当集成服务遇到空输入值时，可使用 ABORT 函数中止会话。

## 用户定义的默认输出值

可以创建用户定义的默认值来替代输出端口的系统默认值。

如果不希望集成服务跳过错误行或希望集成服务在日志中对跳过的行写入特定消息，可以为输出端口输入用户定义的默认值。当集成服务遇到输出转换错误时，可以输入默认值以完成以下函数：

- 使用常量值或常量表达式替换错误。集成服务不会跳过该行。
- 使用 ABORT 函数中止会话。
- 使用 ABORT 函数中止映射。
- 在日志中针对转换错误写入特定消息。

不能为输入/输出端口输入用户定义的默认输出值。

下表汇总了集成服务如何处理输出端口转换错误及转换中的默认值：

默认值	默认值类型	说明
转换错误	系统	出现转换错误时，如果您未替代默认值，集成服务将执行以下任务： <ul style="list-style-type: none"><li>- 转换错误计数增加 1。</li><li>- 跳过该行，并将错误和输入行写入会话日志文件或行错误日志，具体视会话配置而定。</li></ul> 集成服务不会将该行写入拒绝文件。
常量或常量表达式	用户定义	集成服务将使用默认值替换错误。 集成服务不会增加错误计数或向日志中写入消息。
中止	用户定义	会话中止，并且集成服务向会话日志中写入消息。 映射中止，并且集成服务向日志中写入消息。 集成服务不会增加错误计数或将这些行写入拒绝文件。

## 替换错误

如果在出现转换错误时不希望集成服务跳过行，可使用常量或常量表达式作为输出端口的默认值。

例如，如果有某个数值输出端口名为 NET\_SALARY，您希望在出现转换错误时使用常量值“9999”，可将默认值 9999 分配给 NET\_SALARY 端口。如果在计算 NET\_SALARY 的值时出现任何转换错误（例如除以零），集成服务将使用默认值 9999。

## 中止会话

如果您不允许出现转换错误，可使用 ABORT 函数作为输出端口中的默认值。

## 在会话日志或行错误日志中写入消息

如果想要集成服务在会话日志中写入特定消息和跳过的行，则可以在输出端口中输入用户定义的默认值。系统默认值为 ERROR（“transformation error”），集成服务会在会话日志中写入消息“transformation error”以及跳过的行。如果要写入其他消息，可以替换“转换错误”。



启用行错误日志记录时，集成服务会将错误消息写入到错误日志而非会话日志中，并且集成服务不会记录事务控制转换回滚或提交错误。如果除了将行写入到行错误日志之外还要写入到会话日志，则可以启用详细数据跟踪。

## 输出端口表达式中的 ERROR 函数

如果输入使用 ERROR 函数的表达式，则输出端口的用户定义的默认值可替代表达式中的 ERROR 函数。

例如，输入以下表达式以指示集成服务在遇到错误时使用值 “Negative Sale”：

```
IIF( TOTAL_SALES>0, TOTAL_SALES, ERROR ('Negative Sale'))
```

以下示例显示了用户定义的默认值可如何替代表达式中的 ERROR 函数：

- **常量值或表达式。**常量值或表达式将替代输出端口表达式中的 ERROR 函数。

例如，如果您输入 “0” 作为默认值，集成服务将替代输出端口表达式中的 ERROR 函数。遇到错误时，它将传递值 0。它不会跳过该行或在日志中写入 “Negative Sale”。

- **ABORT。**ABORT 函数将替代输出端口表达式中的 ERROR 函数。

如果使用 ABORT 函数作为默认值，出现转换错误时集成服务将中止会话。ABORT 函数将替代输出端口表达式中的 ERROR 函数。

- **错误。**If you use the ERROR function as the default value, the Integration Service includes the following information in the log:

- 来自默认值的错误消息
- 输出端口表达式中 ERROR 函数中指示的错误消息
- 跳过的行

例如，可以使用以下 ERROR 函数替代默认值：

```
ERROR('No default value')
```

集成服务将跳过该行，并在日志中包括错误消息。

```
TE_7007 Transformation Evaluation Error; current row skipped...
TE_7007 [<<Transformation Error>> [error]: Negative Sale
... error('Negative Sale')
]
Sun Sep 20 13:57:28 1998
TE_11019 Port [OUT_SALES]: Default value is: ERROR(<<Transformation Error>> [error]: No default value
... error('No default value')
```

## 默认值的常规规则

创建默认值时，请遵循以下规则和准则：

- 默认值必须是空值、常量值、常量表达式、ERROR 函数或 ABORT 函数。
- 对于输入/输出端口，集成服务将使用默认值处理空输入值。输入/输出端口的输出默认值始终为 ERROR（“转换错误”）。
- 变量端口不使用默认值。
- 在汇总器和等级转换中，可以按端口将默认值分配到组中。
- 并不是所有转换中的所有端口类型都允许用户定义的默认值。如果端口不允许用户定义的默认值，将禁用默认值字段。
- 并不是所有转换都允许用户定义的默认值。
- 如果转换未连接到映射数据流，集成服务将忽略用户定义的默认值。
- 如果任何输入端口未连接，将假定其值为空，集成服务将对该输入端口使用默认值。
- 如果输入端口默认值包含 ABORT 函数且输入值为空，集成服务将立即停止会话。使用 ABORT 函数作为默认值可限制空输入值。输入端口中的第一个空值将停止会话。

- 如果输出端口默认值包含 ABORT 函数且该端口发生任何转换错误，会话将立即停止。使用 ABORT 函数作为默认值可对转换错误强制执行严格规则。此端口的第一个转换错误将停止会话。
- 如果输入端口默认值包含 ABORT 函数且输入值为空，集成服务将立即停止映射。使用 ABORT 函数作为默认值可限制空输入值。输入端口中的第一个空值将停止映射
- 如果输出端口默认值包含 ABORT 函数且该端口发生任何转换错误，映射将立即停止。使用 ABORT 函数作为默认值可对转换错误强制执行严格规则。此端口的第一个转换错误将停止映射
- ABORT 函数、常量值和常量表达式将替代在输出端口表达式中配置的 ERROR 函数。

## 默认值验证

输入默认值时可以对其进行验证。Designer 包括“验证”按钮，可确保默认值有效。系统将会显示一条消息，指示默认值是否有效。

保存映射时，Designer 也会验证默认值。如果输入的默认值无效，Designer 会将该映射标记为无效。

输入默认值时，Developer tool 工具将对其进行验证。

保存映射时，Developer tool 将验证默认值。如果输入的默认值无效，Developer tool 会将该映射标记为无效。

## 配置转换中的跟踪级别

配置转换时，可以设置集成服务写入会话日志中的详细信息量。

下表介绍了会话日志跟踪级别：

跟踪级别	说明
普通	集成服务记录初始化信息和状态信息、遇到的错误以及由于转换行错误而跳过的行。汇总会话结果（但不是在单个行的级别）。
简洁	集成服务记录初始化信息、错误消息和已拒绝数据的通知。
详细初始化	除了普通跟踪以外，集成服务还记录其他初始化详细信息、使用的索引名称和数据文件，以及详细的转换统计信息。
详细数据	除了详细初始化跟踪以外，集成服务还记录传递到映射的每一行。还要记下集成服务截断字符串数据的位置以符合列的精度，并提供详细的转换统计信息。 启用行错误日志记录时，允许集成服务将错误写入会话日志和错误日志。 将跟踪级别配置为详细数据时，集成服务将在处理转换时写入块中所有行的行数据。

默认情况下，每个转换的跟踪级别都为“普通”。仅在需要对行为方式不符合预期的转换进行调试时，才会将跟踪级别更改为“详细”设置。要稍微提高性能，也可以将跟踪级别设置为“简洁”，它将在运行包含转换的工作流时向会话日志中写入最少的详细信息。

配置会话时，可以使用会话中适用于所有转换的单一跟踪级别替代各个转换的跟踪级别。

# 可重用转换

映射可以包含可重用和不可重用的转换。不可重用转换存在于单个映射内。可重用转换可以用于多个映射中。

例如，可以创建一个表达式转换，用于计算在加拿大的销售增值税，这在分析在该国的经营成本时非常有用。您可以创建可重用转换，而不是每次执行同样的工作。需要将该转换合并到映射中时，可以将该转换的一个实例添加到映射中。随后，如果更改转换的定义，则其所有实例都将继承更改。

Designer 将每个可重用转换作为独立于使用该转换的任何映射的元数据进行存储。如果查看导航器中某一文件夹的内容，可以看到该文件夹中所有可重用转换的列表。

每个可重用转换都属于可在 Designer 中使用的转换的类别。例如，可以创建一个可重用汇总器转换，以便在多个映射中执行相同的汇总计算；或者创建一个可重用存储过程转换，以便在多个映射中调用相同的存储过程。

可以将大多数转换创建为不可重用或可重用。不过，只能将外部过程转换创建为可重用转换。

在向映射添加可重用转换的实例时，必须注意对该转换所做的更改不得使映射无效，也不得生成意外数据。

## 实例和继承的更改

将可重用转换添加到映射后，您就添加了转换的实例。转换的定义仍存在于映射外部，而转换的实例则显示在映射内。

由于可重用转换的实例是指向该转换的指针，所以在 Transformation Developer 中更改转换时，其实例将反映这些更改。您无需在每个使用相同转换的映射中都更新该转换，而是可以更新一次可重用转换，该转换的所有实例都将继承此更改。请注意，实例不会继承属性设置的更改，只能继承对转换端口、表达式和名称的修改。

## 表达式中的映射变量

映射参数和变量用于可重用转换表达式。Designer 在验证参数或变量时，会将其视为整数数据类型。在 Maplet 或映射中使用转换时，Designer 将再次验证表达式。如果 Maplet 或映射中不存在该映射参数或变量，Designer 将记录错误。

## 创建可重用转换

您可以使用以下方法创建可重用转换：

- **在 Transformation Developer 中进行设计。**在 Transformation Developer 中，您可以构建新的可重用转换。
- **从 Mapping Designer 升级非可重用转换。**在将转换添加到映射后，您可将其升级为可重用转换的状态。然后，在映射中设计的转换会变为在存储库中其他地方维护的可重用转换的实例。

将转换升级为可重用状态后无法再将其降级。但是，您可以为它创建一个非可重用实例。

**注意：**序列生成器转换在 Maplet 中必须是可重用的。您无法将 Maplet 中的可重用序列生成器转换降级为非可重用转换。

要创建可重用转换：

1. 在 Designer 中，切换到 Transformation Developer。
2. 单击“转换”工具栏上与您要创建的转换类型相对应的按钮。
3. 在工作簿内拖动以创建转换。
4. 双击转换标题栏以打开显示其属性的对话框。
5. 单击“重命名”按钮并为转换输入描述性名称，然后单击“确定”。
6. 单击“端口”选项卡，然后为该转换添加您需要的所有输入和输出端口。

7. 设置转换的其他属性，然后单击“确定”。

这些属性会因您所创建的转换不同而异。例如，如果您创建了表达式转换，则需要为一个或多个转换输出端口输入表达式。如果创建存储过程转换，则需要将存储过程标识为调用。

## 提升不可重用转换

用于创建可重用转换的其他技术可提升映射中的现有转换。通过在“编辑转换”对话框中选中“设为可重用”选项，指导 Designer 提升转换并在映射中创建转换的实例。

要提升不可重用转换，请执行以下操作：

1. 在 Designer 中，打开映射并双击要提升的转换的标题栏。
2. 选择“设为可重用”选项。
3. 系统提示您是否确定要提升转换时，单击“是”。
4. 单击“确定”返回到映射。

现在，查看您正在处理的文件夹中的可重用转换列表时，新提升的转换会显示在此列表中。

## 创建可重用转换的不可重用实例

在映射中可以使用可重用转换的不可重用实例。必须将可重用转换设为在同一个文件夹中不可重用。如果要在其他文件夹中创建可重用转换的不可重用实例，需要首先在源文件夹中创建转换的不可重用实例，然后将该实例复制到目标文件夹。

创建可重用转换的不可重用实例：

1. 在 Designer 中打开一个映射。
2. 在导航器中，选择现有的转换，并将该转换拖至映射工作区。释放转换前按住 Ctrl 键。

状态栏将显示以下消息：

Make a non-reusable copy of this transformation and add it to this mapping.

3. 释放转换。

Designer 现已创建现有可重用转换的不可重用实例。

## 将可重用转换添加到映射中

创建可重用转换后，可将其添加到映射中。

要添加可重用转换：

1. 在 Designer 中，切换到 Mapping Designer。
2. 打开或创建映射。
3. 在存储库对象列表中，向下钻取直到在文件夹的“转换”部分中找到想要的可重用转换。
4. 将此转换从导航器拖到映射中。

此时将显示该可重用转换的副本（或实例）。

5. 将此新转换链接到其他转换或目标定义。

## 修改可重用转换

对通过 Transformation Developer 输入的可重用转换的更改会立即反映在该转换的所有实例中。尽管此功能是保存工作和实施标准的有力方式，但在修改可重用转换时存在使映射失效的风险。

要查看可能受您对转换所做更改影响的映射、Mapplet 或快捷方式，请在工作区或导航器中选择转换，右键单击并选择“查看相关性”。

如果对可重用转换进行了以下任意更改，则使用该转换的实例的映射可能失效：

- 删除转换中的一个或多个端口时，会将该实例与通过映射的部分或全部数据流断开连接。
- 更改端口数据类型时，无法将来自该端口的数据映射到使用不兼容数据类型的其他端口。
- 更改端口名称时，引用该端口的表达式将不再有效。
- 在可重用转换中输入无效的表达式时，使用该转换的映射将不再有效。集成服务无法基于无效映射运行会话。

## 还原为原始可重用转换

如果在映射中更改了可重用转换的属性，可以通过单击“还原”按钮还原为原始可重用转换属性。

## 第 2 章

# 汇总器转换

本章包括以下主题：

- [汇总器转换概览, 46](#)
- [汇总器转换组件, 46](#)
- [配置汇总缓存, 48](#)
- [汇总表达式, 48](#)
- [分组依据端口, 49](#)
- [使用已排序输入, 51](#)
- [创建汇总器转换, 52](#)
- [汇总器转换提示, 53](#)
- [汇总器转换故障排除, 53](#)

## 汇总器转换概览

汇总器转换可执行汇总计算，如求平均值和求和。集成服务读取和存储汇总缓存中的数据组和行数据时将执行汇总计算。汇总器转换属于主动转换。

汇总器转换与表达式转换的区别在于使用汇总器转换对组执行计算。表达式转换可逐行执行计算。

使用转换语言创建汇总表达式时，可以使用条件子句来筛选行，因此比 SQL 语言更具灵活性。

创建包含汇总器转换的会话后，可以启用“增量汇总”会话选项。集成服务执行增量汇总时，会通过映射传递源数据，并使用历史缓存数据以增量方式执行汇总计算。

## 汇总器转换组件

汇总器是一种更改管道中行数的主动转换。汇总器转换包含以下组件和选项：

- **汇总缓存。** 集成服务将数据存储在汇总缓存中，直至完成汇总计算。集成服务将组值存储在索引缓存中，将行数据存储在数据缓存中。
- **汇总表达式。** 在输出端口中输入表达式。表达式可包含非汇总表达式和条件子句。
- **分组依据端口。** 指示组的创建方式。可以为组配置输入、输入/输出、输出或变量端口。分组数据时，除非另行指定，否则汇总器转换将输出每个组的最后一行。

- **已排序输入。** 选择此选项可改善会话性能。要使用已排序输入，必须将按分组依据端口以升序或降序顺序排序的数据传递至汇总器转换。

可以在“属性”和“端口”选项卡上配置汇总器转换组件和选项。

## 配置汇总器转换属性

在“属性”选项卡上修改汇总器转换属性。

配置以下选项：

汇总器设置	说明
缓存目录	集成服务创建索引和数据缓存文件的本地目录。默认情况下，集成服务使用 Workflow Manager 中输入的用于进程变量 \$PMCacheDir 的目录。如果您输入新目录，确保该目录存在，且具有可用于汇总缓存的充足磁盘空间。 如果已启用增量汇总，集成服务会在每次运行会话时创建文件备份。缓存目录必须具有可用于两组文件的充足磁盘空间。
跟踪级别	此转换的会话日志中显示的详细信息量。
已排序输入	指示输入数据已按组预先排序。仅在映射将已排序数据传递到汇总器转换时选择此选项。
汇总器数据缓存大小	转换的数据缓存大小。默认缓存大小为 2,000,000 字节。如果配置的会话缓存总大小为 2 GB (2,147,483,648 字节) 或更大，则必须在 64 位集成服务上运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。
汇总器索引缓存大小	转换的索引缓存大小。默认缓存大小为 1,000,000 字节。如果配置的会话缓存总大小为 2 GB (2,147,483,648 字节) 或更大，则必须在 64 位集成服务上运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。
转换范围	指定集成服务如何将转换逻辑应用至传入数据： <ul style="list-style-type: none"> <li>- 事务。将转换逻辑应用至事务中的所有行。当一行数据取决于同一事务中的所有行，但不取决于其他事务中的行时，请选择“事务”。</li> <li>- 全部输入。将转换逻辑应用至所有传入数据。选择“全部输入”时，PowerCenter 将删除传入的事务边界。当一行数据取决于源中的所有行时，请选择“全部输入”。</li> </ul>

## 配置汇总器转换端口

要配置汇总器转换中的端口，请完成以下任务：

- 使用端口中的条件子句或非汇总函数在任何输出端口中输入表达式。
- 创建多个汇总输出端口。
- 将任何输入、输入/输出、输出或变量端口配置为分组依据端口。
- 仅将必要的输入/输出端口连接至后续转换，可减小数据缓存大小，从而改善性能。
- 使用本地变量的变量端口。
- 输入表达式时创建与其他转换的连接。

## 配置汇总缓存

运行使用汇总器转换的会话时，集成服务会在内存中创建索引和数据缓存以处理转换。如果集成服务需要更多空间，则会将溢出值存储在缓存文件中。

可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。

**注意：**集成服务使用内存来处理具有已排序端口的汇总器转换。集成服务不使用高速缓存。因此，无需为使用已排序端口的汇总器转换配置高速缓存。

## 汇总表达式

Designer 仅允许在汇总器转换中使用汇总表达式。汇总表达式可包含条件子句和非汇总函数。该表达式还可以包含一个包含在其他汇总函数中的汇总函数，例如：

```
MAX( COUNT( ITEM ) )
```

汇总表达式的结果因转换中的分组依据端口而异。例如，集成服务在计算未定义分组依据端口的以下汇总表达式时，会查找已售项目的总数量：

```
SUM( QUANTITY )
```

但是，如果使用同一表达式，并按 ITEM 端口分组，集成服务会按项目返回已售项目的总数量。

可以在任何输出端口中创建汇总表达式，并在转换中使用多个汇总端口。

### 相关主题：

- [“使用表达式” 页面上 30](#)

## 汇总函数

可在汇总器转换中使用以下汇总函数。可以将一个汇总函数嵌套在另一汇总函数中。

转换语言包括以下汇总函数：

- AVG
- COUNT
- FIRST
- LAST
- MAX
- MEDIAN
- MIN
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

上述任一函数均必须在汇总器转换中的表达式中使用。



# 嵌套汇总函数

可以在汇总器转换的不同输出端口中包含多个单层函数或多个嵌套函数。但是，不能在汇总器转换中同时包含单层和嵌套函数。因此，如果汇总器转换的任何输出端口中包含单层函数，则不能在该转换的任何其他端口中使用嵌套函数。将单层函数和嵌套函数包含在同一汇总器转换中时，Designer 工具会将映射或 Mapplet 标记为无效。如果需要同时创建单层函数和嵌套式函数，请另行创建汇总器转换。

# 条件子句

在汇总表达式中使用条件子句可减少汇总中使用的行数。条件子句可以是计算结果为 TRUE 或 FALSE 的任何子句。

例如，使用以下表达式计算超过其季度配额的员工的佣金总额：

```
SUM( COMMISSION, COMMISSION > QUOTA )
```

# 非汇总函数

您还可以在汇总表达式中使用非汇总函数。

以下表达式可返回每个项目的最大已售项目数量（按项目分组）。如果没有售出项目，表达式将返回 0。

```
IIF( MAX( QUANTITY ) > 0, MAX( QUANTITY ), 0))
```

# 汇总函数中的空值

配置集成服务时，可选择集成服务处理汇总函数中空值的方式。可以选择将汇总函数中的空值作为空值或零处理。默认情况下，集成服务将汇总函数中空值作为空值处理。

# 分组依据端口

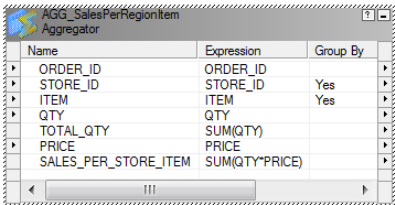
通过汇总器转换可以定义要进行汇总的组，无需对所有输入数据执行汇总。例如，可以查找按区域分组的销售总额，不必查找公司销售总额。

要为汇总表达式定义组，请在汇总器转换中选择相应的输入、输入/输出、输出和变量端口。可以选择多个分组依据端口来为每个唯一的组合创建新组。然后，集成服务会为每个组执行定义的汇总。

如果对值进行分组，集成服务会为每个组生成一行。如果不对值进行分组，集成服务会针对所有输入行返回一行。集成服务通常返回每个组的最后一行（或接收到的最后一行），并在行中显示汇总的结果。但是，如果指定要返回的特定行（例如，通过使用 FIRST 函数），集成服务将返回指定的行。

在汇总器转换中选择多个分组依据端口时，集成服务将通过端口顺序来确定分组顺序。由于组顺序可能会影响结果，因此请对分组依据端口进行排序以确保得到相应的分组。例如，依次按 ITEM\_ID 和 QUANTITY 分组的结果可能不同于依次按 QUANTITY 和 ITEM\_ID 分组的结果，因为数量的数值不一定唯一。

以下汇总器转换依次按 STORE\_ID 和 ITEM 分组：



如果通过该汇总器转换发送以下数据：

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
101	'AAA'	2	2.45
201	'battery'	1	1.99
201	'battery'	4	1.59
301	'battery'	1	2.45

集成服务将对以下唯一的组执行汇总计算：

STORE_ID	ITEM
101	'battery'
101	'AAA'
201	'battery'
301	'battery'

然后，集成服务将传递接收到的最后一行以及汇总结果，如下所示：

STORE_ID	ITEM	TOTAL_QTY	SALES_PER_STORE_ITEM
101	'AAA'	2	4.90
101	'battery'	6	17.34
201	'battery'	5	8.35
301	'battery'	1	2.45

## 非汇总表达式

可在分组依据端口中使用非汇总表达式来修改或替换组。例如，如果要在分组之前替换“AAA battery”，可以使用以下表达式创建名为 CORRECTED\_ITEM 的新分组依据输出端口：

```
IIF( ITEM = 'AAA battery', battery, ITEM )
```

## 默认值

可为组中的每个端口定义默认值以替换空输入值。此操作允许集成服务在汇总中包含空项组。

相关主题：

- [“端口的默认值” 页面上 36](#)

# 使用已排序输入

可以通过使用已排序输入选项提高汇总器转换的性能。使用已排序输入时，集成服务假定所有数据已按组排序，并在读取某个组的行时执行汇总计算。如有需要，数据集成服务会将组信息存储在内存中。要使用已排序输入选项，必须将已排序数据传递至汇总器转换。配置具有多个分区的会话时，可以通过已排序端口提高性能。

如果不使用已排序输入，则集成服务将在读取时执行汇总计算。由于数据未排序，因此集成服务将存储每个组的数据，直至读取整个源，以确保所有汇总计算准确无误。

例如，如果选择已排序输入选项，则汇总器转换将具有 STORE\_ID 和 ITEM 分组依据端口。在向汇总器传递以下数据后，集成服务将在发现新组“201/电池”后对“101/电池”组中的三行执行汇总。

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
201	'battery'	4	1.59
201	'battery'	1	1.99

如果使用已排序输入，但没有预先对数据进行正确排序，则会产生意外结果。

## 已排序输入条件

如果以下任一条件为 true，请勿使用已排序输入：

- 汇总表达式使用嵌套的汇总函数。
- 会话使用增量汇总。

如果使用已排序输入且未对数据进行正确排序，则会话会失败。

## 数据排序

要使用已排序输入，可以通过汇总器传递已排序数据。

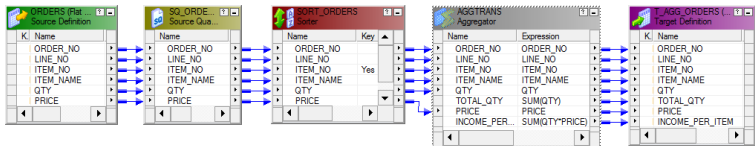
必须通过以下方式对数据排序：

- 按汇总器组依据端口、按照这些端口在汇总器转换中显示的顺序。
- 使用为会话配置的相同排序顺序。如果数据没有按照会话排序顺序以严格的升序或降序排列，则集成服务将使会话失败。例如，如果您将会话配置为使用法语排序顺序，则必须使用法语排序顺序对传递到汇总器转换中的数据排序。

对于关系源和文件源，在将映射中的数据传递到汇总器转换之前，使用排序器转换对这些数据排序。如果没有转换更改已排序数据的顺序，则可以在汇总器之前将排序器转换放置在映射中任意位置。汇总器转换中“分组依据”列的顺序必须与其在排序器转换中显示的顺序相同。

如果会话使用关系源，则还可以使用源限定符转换中的“已排序端口数”选项对源数据库中分组列排序。分组列在汇总器转换和源限定符转换中的顺序必须相同。

以下映射显示了一个排序器转换，该排序器转换配置为按 ITEM\_NO 对源数据进行升序顺序排序：



排序器转换对数据进行如下排序：

ITEM_NO	ITEM_NAME	QTY	PRICE
345	Soup	4	2.95
345	Soup	1	2.95
345	Soup	2	3.25
546	Cereal	1	4.49
546	Cereal	2	5.25

使用已排序输入后，汇总器转换将返回以下结果：

ITEM_NO	ITEM_NAME	TOTAL_QTY	INCOME_PER_ITEM
345	Soup	7	21.25
546	Cereal	3	14.99

## 创建汇总器转换

要在映射中使用汇总器转换，请在映射中添加汇总器转换。然后，使用汇总表达式和分组依据端口配置转换。

创建汇总器转换：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择汇总器转换。
2. 输入汇总器的名称，并单击“创建”。然后单击“完成”。  
Designer 现已创建汇总器转换。
3. 将端口拖至汇总器转换。  
Designer 现已为您添加的每个端口创建输入/输出端口。
4. 双击转换的标题栏，可打开“编辑转换”对话框。
5. 选择“端口”选项卡。
6. 对汇总器用于创建组的每个列单击“分组”选项。  
或者，输入默认值来替换空组。
7. 单击“添加”可添加表达式端口。

表达式端口必须为输出端口。通过清除“输入 (I)”将端口设为输出端口。

8. 或者，为特定端口添加默认值。

如果目标数据库不处理空值，且某些端口可能包含空值，请指定默认值。

9. 在“属性”选项卡上配置属性。

## 汇总器转换提示

使用已排序输入以减少汇总缓存的使用。

已排序输入可减少会话期间缓存的数据量并提高会话性能。请将此选项与排序器转换配合使用，以将已排序数据传递到汇总器转换。

汇总器转换提供的输出可能未排序。

要对汇总器转换的输出排序，请使用排序器转换。

限制连接的输入/输出或输出端口。

限制连接的输入/输出端口或输出端口的数量可减少汇总器转换在数据缓存中存储的数据量。

在汇总数据之前对其进行筛选。

如果您在映射中使用筛选器转换，请在执行汇总器转换之前执行该转换，以减少不必要的汇总。

## 汇总器转换故障排除

我选择了已排序输入，但是工作流花费的时间与之前相同。

出现以下任一情况时，您无法使用已排序输入：

- 汇总表达式包含嵌套汇总函数。
- 会话使用增量汇总。
- 源数据为数据驱动。

出现其中任意情况时，集成服务将如未使用已排序输入一样处理该转换。

使用汇总器转换的会话会降低性能。

在工作流过程中，集成服务可以分页到磁盘。您可以通过增加转换属性中的索引和数据缓存大小来提高会话性能。

我在汇总器转换中输入了替代缓存目录，但集成服务将会话增量汇总文件保存到其他位置。

您可以在会话级别替代转换缓存目录。集成服务会在会话日志中记录缓存目录。还可以检查替代缓存目录的会话属性。

## 第 3 章

# 自定义转换

本章包括以下主题：

- [自定义转换概览, 54](#)
- [创建自定义转换, 55](#)
- [使用组和端口, 56](#)
- [使用端口属性, 58](#)
- [自定义转换属性, 58](#)
- [使用事务控制, 60](#)
- [阻止输入数据, 61](#)
- [使用过程属性, 63](#)
- [创建自定义转换过程, 63](#)

## 自定义转换概览

自定义转换可与您在 Designer 接口外部创建的过程结合使用，以扩展 PowerCenter 功能。您可以创建自定义转换，并将其与使用自定义转换函数开发的过程绑定。自定义转换可以是主动转换，也可以是被动转换。

使用自定义转换可创建排序和汇总等这类需要在输出任何输出行之前处理所有输入行的转换应用。在此过程中，自定义转换中的输入和输出函数会单独运算，这一点与外部过程转换不同。

集成服务使用输入函数将输入数据传递至过程。输出函数是过程代码中必须输入的独立函数，可将输出数据传递至集成服务。相对而言，在外部过程转换中，外部过程函数可同时执行输入和输出，且其参数包含所有转换端口。

您还可以使用自定义转换创建需要多个输入组、多个输出组，或多个输入和输出组的转换。组是一行数据进入或离开转换的表示形式。例如，可以创建具有一个输入组和多个输出组，并可解析 XML 数据的自定义转换。或者，可以创建具有两个输入组和一个输出组，并可将两个输入数据流合并至一个输出数据流的自定义转换。

## 使用基于自定义转换构建的转换

可以使用自定义转换构建转换。可以使用自定义转换构建某些 PowerCenter 转换。随附 Informatica 产品提供的以下转换是原生转换，无法使用自定义转换构建：

- 汇总器转换
- 表达式转换
- 外部过程转换

- 筛选器转换
- 联接器转换
- 查找转换
- 规范器转换
- 等级转换
- 路由器转换
- 序列生成器转换
- 排序器转换
- 源限定符转换
- 存储过程转换
- 事务控制转换
- 更新策略转换

所有其他转换均可使用自定义转换构建。适用于自定义转换的规则，如阻止规则，也适用于使用自定义转换构建的转换。例如，在映射中连接自定义转换时，必须验证并确保数据能从目标加载顺序组中的所有源流动到目标，而不会使集成服务阻止所有源。同样，也必须为使用自定义转换构建的转换验证并确保这一点。

## 代码页兼容性

当集成服务在 ASCII 模式下运行时，可将数据以 ASCII 格式传递至自定义转换过程。当集成服务在 Unicode 模式下运行时，可将数据以 UCS-2 格式传递至该过程。

使用自定义转换过程代码中的 `INFA_CTChangeStringMode()` 和 `INFA_CTSetDataCodePageID()` 函数请求其他格式或其他代码页中的数据。

具体使用哪些函数视集成服务的数据移动模式而定：

- **ASCII 模式。** 使用 `INFA_CTChangeStringMode()` 函数请求 UCS-2 格式的数据。使用此函数时，该过程只能将 ASCII 字符以 UCS-2 格式传递至集成服务。当集成服务在 ASCII 模式下运行时，不能使用 `INFA_CTSetDataCodePageID()` 函数更改代码页。
- **Unicode 模式。** 使用 `INFA_CTChangeStringMode()` 函数请求 MBCS（多字节字符集）格式的数据。该过程请求 MBCS 格式的数据时，集成服务将传递集成服务代码页中的数据。使用 `INFA_CTSetDataCodePageID()` 函数请求不同于集成服务代码页的其他代码页中的数据。`INFA_CTSetDataCodePageID()` 函数中指定的代码页必须与集成服务代码页双向兼容。

**注意：**您还可以使用 `INFA_CTRebindInputDataType()` 函数更改自定义转换中特定端口的格式。

## 分发自定义转换过程

可以将自定义转换从一个存储库复制到另一个存储库。在存储库之间复制自定义转换时，必须验证目标存储库使用的集成服务计算机是否包含自定义转换过程。

## 创建自定义转换

可以在 Transformation Developer 中创建可重用自定义转换，并将转换实例添加到映射中。可以在 Mapping Designer 或 Mapplet Designer 中创建不可重用自定义转换。

每个自定义转换均可指定模块和过程名称。可以基于包含该过程的现有共享库或 DLL 创建自定义转换，也可以创建自定义转换作为创建该过程的基础。创建自定义转换以结合使用现有共享库或 DLL 时，确保定义的模块和过程名称正确无误。

创建自定义转换作为创建该过程的基础时，请选择转换并生成代码。Designer 将使用转换属性生成过程代码。对于共享同一模块名称的所有转换，Designer 会将代码生成到一个目录中。

Designer 将生成以下文件：

- **m\_<module\_name>.c**. 定义模块。此文件包含初始化函数 `m_<module_name>_moduleInit()`，利用此函数可编写要让集成服务在加载模块时运行的代码。同样，此文件包含取消初始化函数 `m_<module_name>_moduleDeinit()`，利用此函数可编写要让集成服务在卸载模块之前运行的代码。
- **p\_<procedure\_name>.c**. 定义模块中的过程。此文件包含实施过程逻辑（如数据清理或合并数据）的代码。
- **makefile.aix**、**makefile.aix64**、**makefile.hpparisc64**、**makefile.linux**、**makefile.sol** 和 **makefile.sol64**. UNIX 平台（zLinux 除外）的 makefile 文件。64 位 AIX 平台使用 `makefile.aix64`，64 位 Solaris 平台使用 `makefile.sol64`。

**注意：**对于 zLinux，需要更新 `makefile.linux`。在 `FLAGS` 部分添加 `-m64`。例如，`FLAGS=-Wall -fPIC -DUNIX -m64`。

## 自定义转换的规则和准则

创建自定义转换时，请使用以下规则和准则：

- 自定义转换是连接的转换。无法在表达式中引用自定义转换。
- 可以在一个模块中包含多个过程。例如，可以在同一模块中包含一个 XML 写入器过程和一个 XML 解析器过程。
- 如果要写入过程代码以处理多个自定义转换实例，则可以将一个共享库或 DLL 绑定到多个自定义转换实例。
- 写入过程代码时，必须确保其不违反基本映射规则。
- 自定义转换会发送和接收高精度小数作为高精度小数。
- 在自定义转换过程中使用多线程代码。

## 自定义转换组件

配置自定义转换时，请定义以下组件：

- **“转换”选项卡**。可以在“转换”选项卡上重命名转换，并添加描述。
- **“端口”选项卡**。可以在自定义转换中添加和编辑端口与组。您也可以定义输出端口依赖的输入端口。
- **“端口属性定义”选项卡**。可以为自定义转换端口创建用户定义的端口属性。
- **“属性”选项卡**。可以定义转换属性，例如，模块和函数标识符、事务属性以及运行时位置。
- **“初始化属性”选项卡**。可以定义外部过程在运行时（如初始化过程中）使用的属性。
- **“元数据扩展”选项卡**。可以创建用于定义过程在运行时（如初始化过程中）所使用的属性的元数据扩展。

## 使用组和端口

自定义转换同时包含输入组和输出组。它还可以包含输入端口、输出端口和输入/输出端口。可以在自定义转换的“端口”选项卡上创建及编辑组和端口。还可以在“端口”选项卡上定义输入端口与输出端口之间的关系。



## 创建组和端口

在自定义转换中可以创建多个输入组和多个输出组。必须至少创建一个输入组和一个输出组。要创建输入组，请单击“创建输入组”图标。要创建输出组，请单击“创建输出组”图标。您可以通过在组表头中键入来更改现有组名称。创建被动自定义转换时，只能创建一个输入组和一个输出组。

创建端口时，Designer 会将该端口添加到当前所选行或组的下方。端口可归属于显示在其正上方的输入组和输出组。显示在输入组下方的输入/输出端口也属于输出组。显示在输出组下方的输入/输出端口也属于输入组。

共享端口的组称为耦合组。类型相反的相邻组可共享端口。一个组可以属于多个耦合组。例如，在[“步骤 1。创建自定义转换” 页面上 63](#) 的图中，InputGroup1 和 OutputGroup1 为共享 ORDER\_ID1 的耦合组

如果该转换包含“端口属性定义”选项卡，则可以编辑每个端口的属性。

## 编辑组和端口

编辑自定义转换中的端口和组时，请遵循以下规则和准则：

- 您可以通过在组表头中键入来更改组名称。
- 对于端口和组名称，只能输入 ASCII 字符。
- 创建组后，则不能更改组类型。如果需要更改组类型，请删除该组并添加一个新组。
- 删除组时，Designer 将删除该组中同一类型的所有端口。不过，所有输入/输出端口仍保留在转换中，归属于它们上方的组，并视乎删除的组类型更改为输入端口或输出端口。例如，一个输出组包含输出端口和输入/输出端口。您删除了输出组。Designer 将删除输出端口。由此，使输入/输出端口更改为输入端口。这些输入端口归属于表头直接位于它们上方的输入组。
- 要上下移动组，请选择组表头并单击“上移端口”或“下移端口”按钮。该组表头上下方的端口保持不变，但它们归属的组可能会更改。
- 要创建输入/输出端口，转换必须包含输入组和输出组。

## 定义端口关系

默认情况下，自定义转换中的输出端口依赖于所有输入端口。不过，您可以在自定义转换中定义输入和输出端口之间的关系。这样做时，可以查看包含自定义转换的映射中的链接路径，并且可以看到输出端口所依赖的输入端口。另外，也可以查看包含自定义转换的映射中目标端口的源列相关性。

要定义自定义转换中端口之间的关系，请创建端口相关性。端口相关性是指输出或输入/输出端口与一个或多个输入或输入/输出端口之间的关系。创建端口相关性时，请基于代码中的过程逻辑创建。

要创建端口相关性，请单击“端口”选项卡中的“自定义转换”并选择“端口相关性”。

例如，创建一个解析 XML 数据的外部过程。创建一个自定义转换，其中包含一个输入组（包含一个输入端口）和多个输出组（包含多个输出端口）。根据外部过程逻辑，所有输出端口均依赖于输入端口。在自定义转换中通过创建每个输出端口的端口相关性可以定义此关系。定义各个端口相关性，以便输出端口依赖于一个输入端口。

要创建端口相关性，请执行以下操作：

1. 在“端口”选项卡上，单击“自定义转换”并选择“端口相关性”。
2. 在“输出端口相关性”对话框中，在“输出端口”字段中选择输出或输入/输出端口。
3. 在“输入端口”窗格，选择该输出端口或输入/输出端口所依赖的输入或输入/输出端口。
4. 单击“添加”。
5. 重复步骤 3 至 4，以便在端口相关性中加入更多输入或输入/输出端口。
6. 要创建其他端口相关性，请重复步骤 2 至 5。
7. 单击“确定”。

# 使用端口属性

端口包含多项属性，如数据类型和精度。在创建自定义转换时，可以创建用户定义的端口属性。用户定义的端口属性可以应用于自定义转换中的所有端口。

例如，可以创建一个外部过程，以解析 XML 数据。可以创建一项名为“XML 路径”的端口属性，在该端口属性中，可以定义元素在 XML 层次结构中的位置。

可以在自定义转换的“端口属性定义”选项卡上创建端口属性并分配默认值。可以在“端口”选项卡上为每个端口定义特定的端口属性值。

在创建端口属性时，可以定义以下属性：

- **名称。**端口属性的名称。
- **Datatype.**端口属性值的数据类型。可以选择“布尔型”、“数值”或“字符串型”。
- **值。**端口属性的默认值。此属性是可选的。在此处输入值后，该值将应用于自定义转换中的所有端口。可以在“端口”选项卡上替代每个端口的端口属性值。

可以为每个自定义转换定义端口属性。不能将端口属性从一个自定义转换复制到另一个。

## 编辑端口属性值

创建端口属性后，可以编辑转换中每个端口的端口属性值。要编辑端口属性值，请单击“端口”选项卡中的“自定义转换”并选择“编辑端口属性”。

通过单击“打开”按钮可以更改特定端口的端口属性值。此时将打开“编辑端口属性默认值”对话框。或者，可以通过直接在“值”列键入而输入新值。

通过从“选择组”字段选择一个组，可以筛选“编辑端口级属性”对话框中列出的端口。

# 自定义转换属性

自定义转换属性同时适用于过程和转换。在自定义转换的“属性”选项卡上配置自定义转换属性。

下表描述了自定义转换属性：

选项	说明
语言	过程代码使用的语言。创建自定义转换时定义语言。如果需要更改语言，请创建新自定义转换。
模块标识符	模块名称。适用于使用 C 或 C++ 开发的自定义转换过程。 只能在此字段中输入 ASCII 字符。不能输入多字节字符。 此属性是包含过程的 DLL 或共享库的基础名称。生成外部过程代码时，Designer 将使用此名称创建 C 文件。
函数标识符	模块中的过程的名称。适用于使用 C 开发的自定义转换过程。 只能在此字段中输入 ASCII 字符。不能输入多字节字符。 输入过程代码时，Designer 将使用此名称创建 C 文件。
类名称	自定义转换过程的类名。适用于使用 C++ 或 Java 开发的自定义转换过程。 只能在此字段中输入 ASCII 字符。不能输入多字节字符。

选项	说明
运行时位置	<p>包含 DLL 或共享库的位置。默认为 \$PMExtProcDir。输入相对于运行自定义转换会话的集成服务节点的路径。</p> <p>如果此属性为空，集成服务使用在集成服务节点上定义的环境变量查找 DLL 或共享库。</p> <p>您必须将所有 DLL 或共享库复制到运行时位置或在集成服务节点上定义的环境变量。如果不能找到 DLL、共享库或引用文件，集成服务无法加载过程。</p>
跟踪级别	此转换的会话日志中显示的详细信息量。默认值为“普通”。
是否可分区	<p>指示是否可以在使用此转换的管道中创建多个分区：</p> <ul style="list-style-type: none"> <li>- 否。无法对转换进行分区。转换和同一渠道中的其他转换只有一个分区。</li> <li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。当自定义转换的不同分区必须共享内存中的对象时，请选择“本地”。</li> <li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li> </ul> <p>默认值为“否”。</p>
必须阻止输入	指示与转换关联的过程是否必须能够阻止传入的数据。默认为“已启用”。
是否活动	<p>指示此转换是主动转换还是被动转换。</p> <p>不能在创建自定义转换后更改此属性。如果需要更改此属性，请创建新自定义转换，并选择正确的属性值。</p>
更新策略转换	指示此转换是否为输出行定义更新策略。默认为“已禁用”。可以为活动自定义转换启用此属性。
转换范围	<p>指示集成服务如何将转换逻辑应用至传入数据：</p> <ul style="list-style-type: none"> <li>- 行</li> <li>- 事务</li> <li>- 全部输入</li> </ul> <p>当转换为被动转换时，此属性始终为“行”。当转换为主动转换时，此属性默认为“全部输入”。</p>
生成事务	<p>指示此转换是否可以生成事务。自定义转换生成事务时，会为所有输出组生成事务。</p> <p>默认为“已禁用”。只能为活动自定义转换启用此属性。</p>
输出是可重复的	<p>指示输出数据的顺序在会话运行之间是否一致。</p> <ul style="list-style-type: none"> <li>- 从不。会话运行之间的输出数据顺序不一致。主动转换默认选择此选项。</li> <li>- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。被动转换默认选择此选项。</li> <li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li> </ul>
要求每个分区一个线程	指示集成服务在处理分区时是否为过程的每个分区使用一个线程。启用此选项后，过程代码可使用线程特定的操作。默认为“已启用”。
输出具有确定性	指示转换在多次会话运行时是否生成一致的输出数据。启用此属性可对使用此转换的会话执行恢复操作。

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

## 设置更新策略

可以使用活动自定义转换为处于以下级别的映射设置更新策略：

- **在过程中。**可以编写外部过程代码，为输出行设置更新策略。外部过程可以标记各行执行插入、更新、删除或拒绝操作。
- **在映射内。**可以在映射中使用自定义转换，以标记各行执行插入、更新、删除或拒绝操作。为自定义转换选择“更新策略转换”属性。
- **在会话内。**配置会话以将源行视为数据驱动。

如果没有配置自定义转换来定义更新策略，或者没有将会话配置为数据驱动，则集成服务不会使用外部过程代码来标记输出行。相反，当自定义转换处于活动状态时，集成服务会将输出行标记为插入。当自定义转换处于被动状态时，集成服务将保留行的类型。例如，当被标记为更新的行进入被动自定义转换时，集成服务将保留该行的类型，并将该行输出为更新。

## 使用线程特定的过程代码

自定义转换过程可以包括特定于线程的操作。特定于线程的操作是根据处理过程的线程执行操作的代码。

可以配置自定义转换，以便集成服务使用一个线程为使用“要求每个分区一个线程”属性的每个分区处理自定义转换。

将自定义转换配置为使用一个线程处理每个分区时，集成服务会为每个分区使用同一线程调用以下函数：

- `p_<proc_name>_partitionInit()`
- `p_<proc_name>_partitionDeinit()`
- `p_<proc_name>_inputRowNotification()`
- `p_<proc_name>_dataBdryRowNotification()`
- `p_<proc_name>_eofNotification()`

可以在这些函数中包含特定于线程的操作，因为集成服务会使用同一线程为每个分区处理这些函数。例如，可以将线程附加和分离到 Java 虚拟机。

**注意：**将自定义转换配置为使用一个线程处理每个分区时，Workflow Manager 会根据映射配置添加分区点。

## 使用事务控制

可以使用以下转换属性为自定义转换定义事务控制：

- **转换范围。**确定集成服务如何将转换逻辑应用到传入数据。
- **生成事务。**指示过程是否会生成事务行并将这些行输出到输出组。

### 转换范围

可以配置集成服务如何将转换逻辑应用到传入数据。可以选择以下值之一：

- **行。**一次将转换逻辑应用至一行数据。当过程结果取决于单行数据时选择“行”。例如，可以在过程解析包含 XML 文件的行时选择“行”。
- **事务。**将转换逻辑应用至事务中的所有行。当过程结果取决于同一事务中的所有行（但不取决于其他事务中的行）时选择“事务”。选择“事务”时，必须将所有输入组连接至同一事务控制点。例如，可以在外部过程对单一事务中的数据执行汇总计算时选择“事务”。

- **全部输入。**将转换逻辑应用至所有传入数据。选择“全部输入”时，集成服务将删除事务边界。当过程结果取决于源中的所有数据行时选择“全部输入”。例如，可以在外部过程对所有传入数据执行汇总计算时或在对所有传入数据进行排序时选择“所有输入”。

# 生成事务

您可以将外部过程代码写入输出事务，例如提交和回滚行。外部过程输出提交和回滚行时，将配置自定义转换以生成事务。选择“生成事务”转换属性。您可以为主动自定义转换启用此属性。

外部过程输出提交或回滚行时，会为所有输出组输出或回滚该行。

配置转换以生成事务时，集成服务会将自定义转换视为事务控制转换进行处理。映射中适用于事务控制转换的大多数规则还适用于自定义转换。例如，在配置自定义转换以生成事务时，无法连接包含转换的管道或管道分支。

使用配置为生成事务的自定义转换编辑或创建会话时，配置该会话用于用户定义的提交。

# 使用事务边界

集成服务将在进入和离开自定义转换时，根据映射配置和自定义转换属性来处理事务边界。

下表介绍了集成服务如何处理自定义转换的事务边界：

转换范围	已启用生成事务	已禁用生成事务
行	集成服务将删除传入事务边界，并且不会调用数据边界通知函数。 它将根据所有输出组的过程逻辑输出事务行。	如果所有输入组的传入数据都来自同一事务控制点，则集成服务将保留传入事务边界，并在所有输出组输出这些传入事务边界。但它不会调用数据边界通知函数。 如果输入组的传入数据来自不同的事务控制点，则集成服务将删除传入事务边界。它不会调用数据边界通知函数。集成服务将在一个开放式事务中输出所有行。
事务	集成服务将保留传入事务边界，并调用数据边界通知函数。 但是，它将根据所有输出组的过程逻辑输出事务行。	集成服务将保留传入事务边界，并调用数据边界通知函数。 它将在所有输出组输出事务行。
全部输入	集成服务将删除传入事务边界，并且不会调用数据边界通知函数。集成服务将根据所有输出组的过程逻辑输出事务行。	集成服务将删除传入事务边界，并且不会调用数据边界通知函数。它将在一个开放式事务中输出所有行。

# 阻止输入数据

默认情况下，集成服务可同时读取目标加载顺序组中的源。但是，可以编写外部过程代码，以阻止某些输入组上的输入数据。阻止是挂起数据流以使其进入多输入组转换的某个输入组中。

要使用自定义转换阻止输入数据，必须编写用于阻止和取消阻止数据的过程代码。您还必须在自定义转换的“属性”选项卡上启用阻止。

## 将过程代码写入块数据

可以编写相应过程，用于阻止和接受传入数据。要阻止传入数据，请使用 `INFA_CTBlockInputFlow()` 函数。要接受传入数据，请使用 `INFA_CTUnblockInputFlow()` 函数。

如果外部过程需要从输入组中交替读取，您可能需要阻止输入数据。如果没有阻止功能，您需要编写过程代码以缓冲传入数据。您可以阻止而非缓存输入数据，这样通常可以提高会话性能。

例如，您需要创建包含两个输入组的外部过程。该外部过程从第一个输入组中读取行，然后从第二个输入组中读取行。如果使用阻止功能，您可以编写外部过程代码，以便在它处理一个输入组中的数据时阻止其他输入组中的数据。当您编写外部过程代码以阻止数据时，可以提高性能，因为该过程不需要将源数据复制到缓冲区。但是，您编写的外部过程可能会分配缓冲区，并将一个输入组中的数据复制到该缓冲区，直到它准备好处理数据。将源数据复制到缓冲区会降低性能。

### 相关主题：

- [“编块函数”页面上 101](#)

## 将自定义转换配置为阻止转换

创建自定义转换时，Designer 将默认启用“输入必须阻止”转换属性。此属性会在您保存或验证映射时影响数据流验证。启用此属性后，自定义转换将变成阻止转换。清除此属性后，自定义转换将不再是阻止转换。

当外部过程代码必须能够阻止输入数据时，请将自定义转换配置为阻止转换。

当满足以下条件之一时，可以将自定义转换配置为非阻止转换：

- 过程代码不包含阻止函数。
- 过程代码包含两种算法，其中一种算法使用阻止，另一种算法将源数据（而非阻止数据）复制到过程分配的缓冲区。代码将检查集成服务是否允许自定义转换阻止数据。可以阻止数据时，过程将使用具有阻止函数的算法，不能阻止数据时则使用另一种算法。创建在多个映射配置中使用的自定义转换时可能要执行此操作。

**注意：**当过程阻止数据，而您将自定义转换配置为非阻止转换时，集成服务会导致会话失败。

## 使用自定义转换验证映射

在映射中包含自定义转换时，Designer 和集成服务都会验证该映射。Designer 会在您保存或验证时验证该映射，集成服务会在您运行会话时验证该映射。

### 设计时验证

保存或验证映射时，Designer 执行数据流验证。Designer 执行该操作时，会验证数据是否可从目标加载顺序组中的所有源流到不含阻止所有源的 z 阻止转换的目标。具有阻止转换的某些映射无效。

### 运行时验证

运行会话时，集成服务会根据运行时的过程代码验证映射。当集成服务执行该操作时，会跟踪是否允许自定义转换阻止数据：

- **将自定义转换配置为阻止转换。**集成服务始终允许自定义转换阻止数据。
- **将自定义转换配置为非阻止转换。**集成服务允许自定义转换阻止数据，具体取决于映射配置。如果集成服务可以在自定义转换中阻止数据，同时不会在目标加载顺序组中阻止所有源，则会允许自定义转换阻止数据。

可以写入过程代码，以检查集成服务是否允许自定义转换阻止数据。使用 `INFA_CT_getInternalProperty()` 函数访问 `INFA_CT_TRANS_MAY_BLOCK_DATA` 属性 ID。如果自定义转换可以阻止数据，则集成服务返回 `TRUE`；如果自定义转换无法阻止数据，则集成服务返回 `FALSE`。



# 使用过程属性

当集成服务运行过程时（例如初始化期间），可以在过程可以使用的自定义转换中定义属性名称和值对。可以在自定义转换的以下选项卡上创建用户定义的属性：

- **元数据扩展。**可以指定属性名称、数据类型、精度和值。使用元数据扩展将信息传递到过程。
- **初始化属性。**可以指定属性名称和值。

虽然在自定义转换中的两个选项卡上都可以定义属性，但“元数据扩展”选项卡可用于提供属性的更多详细信息。使用元数据扩展将属性传递到过程。

例如，创建可在转换数据后对数据进行排序的自定义转换外部过程。您可能会创建名为 Sort\_Ascending 的布尔元数据扩展。在映射中使用自定义转换时，可以为元数据扩展选择 True 或 False，具体取决于想要过程如何对数据排序。

在自定义转换中定义属性时，使用获取所有属性名称函数（例如 INFA\_CTGetAllPropertyNamesM()）来访问“初始化属性和元数据扩展”选项卡中定义的所有属性的名称。使用获取外部属性函数（例如 INFA\_CT\_getExternalPropertyM()）来访问您指定的属性 ID 的属性名称和值。

**注意：**在定义具有相同名称的元数据扩展和初始化属性时，这些属性函数只返回元数据扩展的信息。

## 创建自定义转换过程

可以创建在 32 位或 64 位集成服务计算机上运行的自定义转换过程。创建自定义转换过程时，可以参考以下步骤：

1. 在 Transformation Developer 中创建可重用自定义转换。或者，在 Mapplet Designer 或 Mapping Designer 中创建不可重用自定义转换。
2. 生成过程模板代码。  
生成过程代码时，Designer 将使用自定义转换中的信息创建 C 源代码文件和生成文件。
3. 修改 C 文件来添加过程逻辑。
4. 使用 C/C++ 编译器编译源代码文件，将其链接至 DLL 或共享库，然后复制到集成服务计算机上。
5. 使用自定义转换创建映射。
6. 在工作流中运行会话。

本部分将通过示例演示此过程。使用本部分中的步骤创建一个包含两个输入组和一个输出组的自定义转换。自定义转换过程将验证该自定义转换是否使用两个输入组和一个输出组。它还将验证所有组中的端口数量是否相等，以及所有组的端口数据类型是否相同。该过程将利用每个输入组中的数据行，并将所有行输出到输出组中。

### 步骤 1. 创建自定义转换

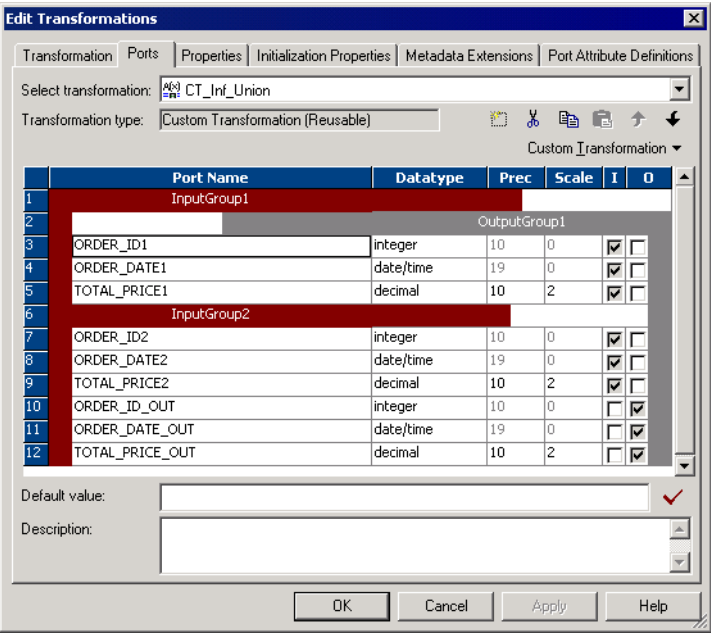
要创建自定义转换：

1. 在 Transformation Developer 中，单击“转换”>“创建”。
2. 在“创建转换”对话框中，选择“自定义转换”，输入转换名称，然后单击“创建”。  
在“联合”示例中，输入 CT\_Inf\_Union 作为转换名称。
3. 在“活动”或“被动”对话框中，以被动转换或活动转换的方式创建转换，然后单击“确定”。  
在“联合”示例中，选择“活动”。
4. 单击“完成”以关闭“创建转换”对话框。

5. 打开该转换，然后单击“端口”选项卡。创建组和端口。

如有必要，可在以后对组和端口进行编辑。

下图显示了一个联合转换的示例，其中包含两个组：



在“联合”示例中，创建组 InputGroup1 和 InputGroup2。为 InputGroup1 创建以下端口：

端口名称	数据类型	精度	小数位数
ORDER_ID1	整型	10	0
ORDER_DATE1	日期/时间	19	0
TOTAL_PRICE1	小数	10	2

为 InputGroup2 创建以下端口：

端口名称	数据类型	精度	小数位数	输入/输出
ORDER_ID2	整型	10	0	输入
ORDER_DATE2	日期/时间	19	0	输入
TOTAL_PRICE2	小数	10	2	输入
ORDER_ID_OUT	整型	10	0	输出
ORDER_DATE_OUT	日期/时间	19	0	输出
TOTAL_PRICE_OUT	小数	10	2	输出



- 选择“属性”选项卡，然后输入一个模块和函数标识符，以及运行时位置。编辑其他转换属性特性，如“跟踪级别”、“是否可区分”、“必须阻止输入”、“是否活动”、“更新策略转换”、“转换范围”和“生成事务”值/复选框。

在“联合”示例中，配置以下属性：

属性名称	值
模块标识符	UnionDemo
函数标识符	联合
运行时位置	\$PMExtProcDir
跟踪级别	普通
是否可区分	否
必须阻止输入	否
是否活动	是
更新策略转换	否
转换范围	全部输入
生成事务	否

- 单击“元数据扩展”选项卡以输入元数据扩展，如外部过程进行初始化可能需要的属性。

在“联合”示例中，请勿创建元数据扩展。

- 如有必要，请单击“端口属性定义”选项卡，以创建端口属性。

在“联合”示例中，请勿创建端口属性。

在创建调用该过程的自定义转换之后，下一步是生成 C 文件。

## 步骤 2。生成 C 文件

在创建自定义转换后，生成源代码文件。Designer 会使用小写字母创建文件名。

要为自定义转换过程生成代码，请执行以下操作：

- 在 Transformation Developer 中，选择转换并单击“转换”>“生成代码”。
- 选择您刚创建的过程。Designer 会列出这些过程，名称为 <module\_name>.<procedure\_name>。  
在“联合”示例中，选择 UnionDemo.Union。
- 指定要生成文件所在的目录，然后单击“生成”。

在“联合”示例中，选择 <client\_installation\_directory>/TX。

Designer 会在您指定的目录中创建一个子目录 <module\_name>。在“联合”示例中，Designer 会创建 <client\_installation\_directory>/TX/UnionDemo。还会创建以下文件：

- m\_UnionDemo.c
- m\_UnionDemo.h

- p\_Union.c
- p\_Union.h
- makefile.aix (32 位)、makefile.aix64 (64 位)、makefile.hp (32 位)、makefile.hp64 (64 位)、makefile.hpparisc64、makefile.linux (32 位) 和 makefile.sol (32 位)。

### 步骤 3。使用转换逻辑填充代码

必须对过程 C 文件进行编码。或者，也可以对模块 C 文件进行编码。在联合示例中，仅填写过程 C 文件。不需要填写模块 C 文件。

要对过程 C 文件进行编码，请执行以下操作：

1. 为过程打开 p\_<procedure\_name>.c。  
在联合示例中，打开 p\_Union.c。
2. 为过程输入 C 代码。
3. 保存修改的文件。

在联合示例中，使用以下代码：

```

/*****
 * Custom Transformation p_union Procedure File
 *
 * This file contains code that functions that will be called by the main
 * server executable.
 *
 * for more information on these files,
 * see $(INFA_HOME)/ExtProc/include/Readme.txt
 *****/

/*
 * INFORMATICA 'UNION DEMO' developed using the API for custom
 * transformations.
 *
 * File Name: p_Union.c
 *
 * An example of a custom transformation ('Union') using PowerCenter
 *
 * The purpose of the 'Union' transformation is to combine pipelines with the
 * same row definition into one pipeline (i.e. union of multiple pipelines).
 * [ Note that it does not correspond to the mathematical definition of union
 * since it does not eliminate duplicate rows.]
 *
 * This example union transformation allows N input pipelines ( each
 * corresponding to an input group) to be combined into one pipeline.
 *
 * To use this transformation in a mapping, the following attributes must be
 * true:
 * a. The transformation must have >= 2 input groups and only one output group.
 * b. In the Properties tab set the following properties:
 *    i. Module Identifier: UnionDemo
 *    ii. Function Identifier: Union
 *    iii. Inputs May Block: Unchecked
 *    iv. Is Active: Checked
 *    v. Update Strategy Transformation: Unchecked *
 *    vi. Transformation Scope: All
 *    vii. Generate Transaction: Unchecked *
 *
 * * This version of the union transformation does not provide code for
 * changing the update strategy or for generating transactions.
 * c. The input groups and the output group must have the same number of ports
 * and the same datatypes. This is verified in the initialization of the
 * module and the session is failed if this is not true.
 * d. The transformation can be used in multiple number of times in a Target
 * Load Order Group and can also be contained within multiple partitions.

```

```

*
*/
/*****
Includes
*****/

include <stdlib.h>
#include "p_union.h"

/*****
Forward Declarations
*****/
INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition);

/*****
Functions
*****/

/*****
Function: p_union_procInit

Description: Initialization for the procedure. Returns INFA_SUCCESS if
procedure initialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
initialization time. It will be called after the moduleInit function.
*****/

INFA_STATUS p_union_procInit( INFA_CT_PROCEDURE_HANDLE procedure)
{
    const INFA_CT_TRANSFORMATION_HANDLE* transformation = NULL;
    const INFA_CT_PARTITION_HANDLE* partition = NULL;
    size_t nTransformations = 0, nPartitions = 0, i = 0;

    /* Log a message indicating beginning of the procedure initialization */
    INFA_CTLogMessageM( eESL_LOG,
        "union_demo: Procedure initialization started ..." );

    INFA_CTChangeStringMode( procedure, eASM_MBCS );

    /* Get the transformation handles */
    transformation = INFA_CTGetChildrenHandles( procedure,
        &nTransformations,
        TRANSFORMATIONTYPE);

    /* For each transformation verify that the 0th partition has the correct
    * properties. This does not need to be done for all partitions since rest
    * of the partitions have the same information */
    for (i = 0; i < nTransformations; i++)
    {
        /* Get the partition handle */
        partition = INFA_CTGetChildrenHandles(transformation[i],
            &nPartitions, PARTITIONTYPE );

        if (validateProperties(partition) != INFA_SUCCESS)
        {
            INFA_CTLogMessageM( eESL_ERROR,
                "union_demo: Failed to validate attributes of "
                "the transformation");
            return INFA_FAILURE;
        }
    }

    INFA_CTLogMessageM( eESL_LOG,
        "union_demo: Procedure initialization completed." );

    return INFA_SUCCESS;
}

```

```

}

/*****
Function: p_union_procDeinit

Description: Deinitialization for the procedure. Returns INFA_SUCCESS if
procedure deinitialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
deinitialization time. It will be called before the moduleDeinit
function.
*****/

INFA_STATUS p_union_procDeinit( INFA_CT_PROCEDURE_HANDLE procedure,
                               INFA_STATUS sessionStatus )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_partitionInit

Description: Initialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/

INFA_STATUS p_union_partitionInit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_partitionDeinit

Description: Deinitialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/

INFA_STATUS p_union_partitionDeinit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_inputRowNotification

Description: Notification that a row needs to be processed for an input
group in a transformation for the given partition. Returns INFA_ROWSUCCESS
if the input row was processed successfully, INFA_ROWFAILURE if the input
row was not processed successfully and INFA_FATALERROR if the input row
causes the session to fail.

Input: partition - the handle for the partition for the given row
       group - the handle for the input group for the given row
Output: None
*****/

```

Remarks: This function is probably where the meat of your code will go,  
as it is called for every row that gets sent into your transformation.  
\*\*\*\*\*/

```
INFA_ROWSTATUS p_union_inputRowNotification( INFA_CT_PARTITION_HANDLE partition,
                                             INFA_CT_INPUTGROUP_HANDLE inputGroup )
```

```
{
    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    const INFA_CT_INPUTPORT_HANDLE* inputGroupPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumInputPorts = 0, nNumOutputGroups = 0,
           nNumPortsInOutputGroup = 0, i = 0;

    /* Get the output group port handles */
    outputGroups = INFA_CTGetChildrenHandles(partition,
                                             &nNumOutputGroups,
                                             OUTPUTGROUPTYPE);

    outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                                  &nNumPortsInOutputGroup,
                                                  OUTPUTPORTTYPE);

    /* Get the input groups port handles */
    inputGroupPorts = INFA_CTGetChildrenHandles(inputGroup,
                                                  &nNumInputPorts,
                                                  INPUTPORTTYPE);

    /* For the union transformation, on receiving a row of input, we need to
     * output that row on the output group. */
    for (i = 0; i < nNumInputPorts; i++)
    {
        INFA_CTSetData(outputGroupPorts[i],
                       INFA_CTGetDataVoid(inputGroupPorts[i]));

        INFA_CTSetIndicator(outputGroupPorts[i],
                           INFA_CTGetIndicator(inputGroupPorts[i]) );

        INFA_CTSetLength(outputGroupPorts[i],
                         INFA_CTGetLength(inputGroupPorts[i]) );
    }

    /* We know there is only one output group for each partition */
    return INFA_CTOutputNotification(outputGroups[0]);
}
```

```
/******
Function: p_union_eofNotification
```

Description: Notification that the last row for an input group has already  
been seen. Return INFA\_FAILURE if the session should fail as a result of  
seeing this notification, INFA\_SUCCESS otherwise.

Input: partition - the handle for the partition for the notification  
group - the handle for the input group for the notification

Output: None

```
*****/
```

```
INFA_STATUS p_union_eofNotification( INFA_CT_PARTITION_HANDLE partition,
                                     INFA_CT_INPUTGROUP_HANDLE group)
{
    INFA_CTLogMessageM( eESL_LOG,
                       "union_demo: An input group received an EOF notification");

    return INFA_SUCCESS;
}
```

```
/******
Function: p_union_dataBdryNotification
```

Description: Notification that a transaction has ended. The data boundary type can either be commit or rollback.  
Return INFA\_FAILURE if the session should fail as a result of seeing this notification, INFA\_SUCCESS otherwise.

Input: partition - the handle for the partition for the notification  
transactionType - commit or rollback

Output: None

\*\*\*\*\*/

```
INFA_STATUS p_union_dataBdryNotification ( INFA_CT_PARTITION_HANDLE partition,
                                           INFA_CT_DATABDRY_TYPE transactionType)
```

```
{
    /* Do nothing */
    return INFA_SUCCESS;
}
```

/\* Helper functions \*/

\*\*\*\*\*/

Function: validateProperties

Description: Validate that the transformation has all properties expected by a union transformation, such as at least one input group, and only one output group. Return INFA\_FAILURE if the session should fail since the transformation was invalid, INFA\_SUCCESS otherwise.

Input: partition - the handle for the partition

Output: None

\*\*\*\*\*/

```
INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition)
```

```
{
    const INFA_CT_INPUTGROUP_HANDLE* inputGroups = NULL;
    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    size_t nNumInputGroups = 0, nNumOutputGroups = 0;
    const INFA_CT_INPUTPORT_HANDLE** allInputGroupsPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumPortsInOutputGroup = 0;
    size_t i = 0, nTempNumInputPorts = 0;

    /* Get the input and output group handles */
    inputGroups = INFA_CTGetChildrenHandles(partition[0],
                                           &nNumInputGroups,
                                           INPUTGROUPTYPE);

    outputGroups = INFA_CTGetChildrenHandles(partition[0],
                                           &nNumOutputGroups,
                                           OUTPUTGROUPTYPE);

    /* 1. Number of input groups must be >= 2 and number of output groups must
     * be equal to one. */
    if (nNumInputGroups < 1 || nNumOutputGroups != 1)
    {
        INFA_CTLogMessageM( eESL_ERROR,
                           "UnionDemo: There must be at least two input groups "
                           "and only one output group");
        return INFA_FAILURE;
    }

    /* 2. Verify that the same number of ports are in each group (including
     * output group). */
    outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                           &nNumPortsInOutputGroup,
                                           OUTPUTPORTTYPE);

    /* Allocate an array for all input groups ports */
    allInputGroupsPorts = malloc(sizeof(INFA_CT_INPUTPORT_HANDLE*) *
                                nNumInputGroups);
}
```

```

for (i = 0; i < nNumInputGroups; i++)
{
    allInputGroupsPorts[i] = INFA_CTGetChildrenHandles(inputGroups[i],
                                                         &nTempNumInputPorts,
                                                         INPUTPORTTYPE);

    if ( nNumPortsInOutputGroup != nTempNumInputPorts)
    {
        INFA_CTLogMessageM( eESL_ERROR,
                            "UnionDemo: The number of ports in all input and "
                            "the output group must be the same.");
        return INFA_FAILURE;
    }
}

free(allInputGroupsPorts);

/* 3. Datatypes of ports in input group 1 must match data types of all other
 * groups.
 * TODO:*/

return INFA_SUCCESS;
}

```

## 步骤 4.构建模块

可在 Windows 或 UNIX 平台上构建模块。

下表列出了在构建模块时每个平台的库文件名称：

平台	模块文件名称
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

### 在 Windows 上构建模块

在 Windows 上，使用 Microsoft Visual C++ 来构建模块。

要在 Windows 上构建模块：

1. 启动 Visual C++。
2. 单击“文件”>“新建”。
3. 在“新建”对话框中，单击“项目”选项卡，选择“Win32 动态链接库”选项。
4. 输入其位置。  
在 Union 示例中，输入 <client\_installation\_directory>/TX/UnionDemo。
5. 输入项目的名称。  
您必须将为自定义转换指定的模块名称用作项目名称。在 Union 示例中，输入 UnionDemo。
6. 单击“确定”。  
此时，Visual C++ 创建一个向导来帮助您定义项目组件。
7. 在向导中，选择一个空 DLL 项目，并单击“完成”。在“新项目信息”对话框中单击“确定”。

此时，Visual C++ 在您指定的目录中创建项目文件。

- 8. 单击“项目”>“添加至项目”>“文件”。
- 9. 向上导航一个目录级别。此目录包含您创建的过程文件。选择所有 .c 文件，并单击“确定”。

在 Union 示例中，添加以下文件：

- m\_UnionDemo.c
- p\_Union.c

- 10. 单击“项目”>“设置”。
- 11. 单击 C/C++ 选项卡，从“类别”字段中选择“预处理器”。
- 12. 在“附加包括目录”字段中，输入以下路径，并单击“确定”：  
    .; <PowerCenter\_install\_dir>\extproc\include\ct
- 13. 单击“构建”>“构建 <module\_name>.dll”或按 F7 以构建项目。

此时，Visual C++ 将创建该 DLL，并将其放在项目目录下的调试或发行目录中。

### 在 UNIX 上构建模块

在 UNIX 上，使用任何 C 编译器构建模块。

要在 UNIX 上构建模块，请执行以下操作：

- 1. 将 Designer 生成的所有 C 文件和生成文件复制到 UNIX 计算机。  
**注意:** 如果在集成服务计算机以外的计算机上构建共享库，则还必须将以下目录中的文件复制到构建计算机：  
    <PowerCenter\_install\_dir>\ExtProc\include\ct  
在联合示例中，复制 <client\_installation\_directory>/TX/UnionDemo 中的所有文件。
- 2. 将环境变量 INFA\_HOME 设置为 集成服务安装目录。  
**注意:** 如果您为 INFA\_HOME 环境变量指定了不正确的目录路径，则 集成服务无法启动。
- 3. 输入下表中的命令以创建项目：

UNIX 版本	命令
AIX (32 位)	make -f makefile.aix
AIX (64 位)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

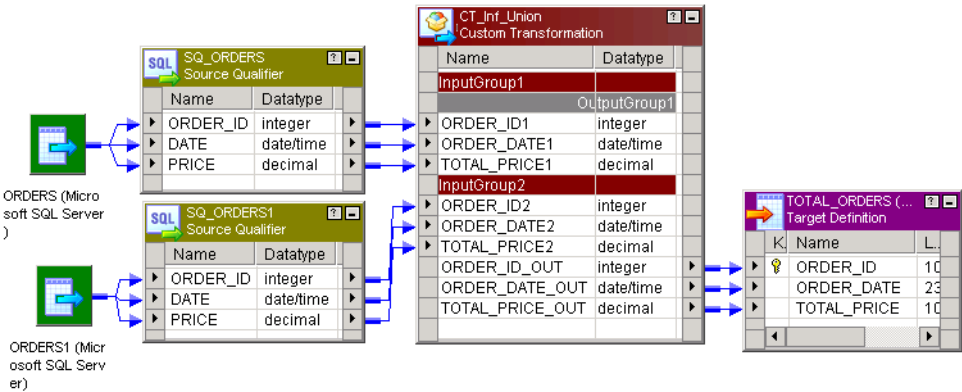
### 步骤 5。创建映射

在 Mapping Designer 中，创建使用自定义转换的映射。

在此映射中，两个具有相同端口和数据类型的源连接到自定义转换中的两个输入组。自定义转换从两个源中获取行，并通过其中一个输出组输出所有行。该输出组具有与输入组相同的端口和数据类型。



在“联合”示例中，创建一个映射，与下图中的某一映射相似：



## 步骤 6。在工作流中运行会话

在运行会话时，集成服务会在您于自定义转换中指定的运行时位置查找共享库或 DLL。

要在工作流中运行会话，请执行以下操作：

1. 在 Workflow Manager 中，创建一个工作流。
2. 在工作流中为此映射创建一个会话。
3. 将共享库或 DLL 复制到运行时位置目录。
4. 运行包含会话的工作流。

当集成服务加载绑定到某个过程的自定义转换时，其会加载 DLL 或共享库并调用您定义的过程。

## 第 4 章

# 自定义转换函数

本章包括以下主题：

- [自定义转换函数概览, 74](#)
- [函数引用, 75](#)
- [使用行, 78](#)
- [生成的函数, 79](#)
- [API 函数, 84](#)
- [基于数组的 API 函数, 106](#)

## 自定义转换函数概览

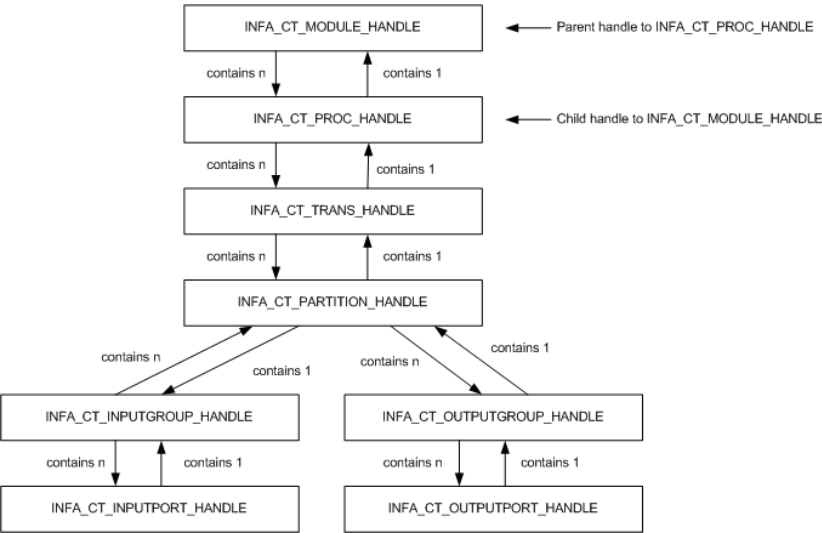
自定义转换与您在 Designer 之外创建的过程一起操作可扩展 PowerCenter 功能。使用自定义转换函数，您可以在与自定义转换关联的过程中创建转换逻辑。PowerCenter 提供两组函数，即生成的函数和 API 函数。集成服务使用生成的函数与过程进行交互。在创建自定义转换和生成源代码文件时，Designer 可在这些文件中加入生成的函数。在过程代码中使用 API 函数可创建转换逻辑。

在写入过程代码时，可以将其配置为接收集成服务中的行块或一次接收一行。当过程接收和处理行块时，可以提高性能。

### 使用句柄

大多数函数都与句柄（如 INFA\_CT\_PARTITION\_HANDLE）相关联。这些函数的第一个参数是函数影响的句柄。自定义转换句柄互相有层次结构关系。父句柄与其子句柄有 1: $n$  关系。

下图显示了自定义转换句柄：



下表介绍了自定义转换句柄：

句柄名称	说明
INFA_CT_MODULE_HANDLE	代表共享库或 DLL。外部过程只能访问其自己共享库或 DLL 中的模块句柄。它不能访问任何其他共享库或 DLL 中的模块句柄。
INFA_CT_PROC_HANDLE	代表共享库或 DLL 内的特定过程。 当需要写入函数以影响多个自定义转换引用的某一过程时，可以使用此句柄。
INFA_CT_TRANS_HANDLE	代表会话中的特定自定义转换实例。
INFA_CT_PARTITION_HANDLE	代表特定自定义转换实例中的特定分区。
INFA_CT_INPUTGROUP_HANDLE	代表分区中的输入组。
INFA_CT_INPUTPORT_HANDLE	代表分区中输入组的输入端口。
INFA_CT_OUTPUTGROUP_HANDLE	代表分区中的输出组。
INFA_CT_OUTPUTPORT_HANDLE	代表分区中输出组的输出端口。

## 函数引用

自定义转换函数包括生成的函数和 API 函数。

下表列出了自定义转换生成的函数：

函数	说明
m_<module_name>_moduleInit()	模块初始化函数。
p_<proc_name>_procInit()	过程初始化函数。
p_<proc_name>_partitionInit()	分区初始化函数。
p_<proc_name>_inputRowNotification()	输入行通知函数。
p_<proc_name>_dataBdryNotification()	数据边界通知函数。
p_<proc_name>_eofNotification()	文件末尾通知函数。
p_<proc_name>_partitionDeinit()	分区取消初始化函数。
p_<proc_name>_procedureDeinit()	过程取消初始化函数。
m_<module_name>_moduleDeinit()	模块取消初始化函数。

下表列出了自定义转换 API 函数：

函数	说明
INFA_CTSetDataAccessMode()	设置数据访问模式函数。
INFA_CTGetAncestorHandle()	获取 Ancestor 句柄函数。
INFA_CTGetChildrenHandles()	获取子句柄函数。
INFA_CTGetInputPortHandle()	获取输入端口句柄函数。
INFA_CTGetOutputPortHandle()	获取输出端口句柄函数。
INFA_CTGetInternalProperty<datatype>()	获取内部属性函数。
INFA_CTGetAllPropertyNamesM()	在 MBCS 模式下获取所有属性名称函数。
INFA_CTGetAllPropertyNamesU()	在 Unicode 模式下获取所有属性名称函数。
INFA_CTGetExternalProperty<datatype>M()	在 MBCS 模式下获取外部属性函数。
INFA_CTGetExternalProperty<datatype>U()	在 Unicode 模式下获取外部属性函数。
INFA_CTRebindInputDataType()	重新绑定输入端口数据类型函数。
INFA_CTRebindOutputDataType()	重新绑定输出端口数据类型函数。
INFA_CTGetData<datatype>()	获取数据函数。
INFA_CTSetData()	设置数据函数。
INFA_CTGetIndicator()	获取指示器函数。

函数	说明
INFA_CTSetIndicator()	设置指示器函数。
INFA_CTGetLength()	获取长度函数。
INFA_CTSetLength()	设置长度函数。
INFA_CTSetPassThruPort()	设置传递端口函数。
INFA_CTOutputNotification()	输出通知函数。
INFA_CTDataBdryOutputNotification()	数据边界输出通知函数。
INFA_CTGetErrorMsgU()	在 Unicode 模式下获取错误消息函数。
INFA_CTGetErrorMsgM()	在 MBCS 模式下获取错误消息函数。
INFA_CTLogMessageU()	在 Unicode 模式下记录会话日志中的消息函数。
INFA_CTLogMessageM()	在 MBCS 模式下记录会话日志的消息函数。
INFA_CTIncrementErrorCount()	递增错误计数函数。
INFA_CTIsterminateRequested()	是否已请求终止函数。
INFA_CTBlockInputFlow()	编块输入组函数。
INFA_CTUnblockInputFlow()	取消编块输入组函数。
INFA_CTSetUserDefinedPtr()	设置用户定义的指针函数。
INFA_CTGetUserDefinedPtr()	获取用户定义的指针函数。
INFA_CTChangeStringMode()	更改字符串模式函数。
INFA_CTSetDataCodePageID()	设置数据代码页 ID 函数。
INFA_CTGetRowStrategy()	获取行策略函数。
INFA_CTSetRowStrategy()	设置行策略函数。
INFA_CTChangeDefaultRowStrategy()	更改转换的默认行策略。

下表列出了自定义转换基于数组的函数：

函数	说明
INFA_CTAGetInputRowMax()	获取最大输入行数函数。
INFA_CTAGetOutputRowMax()	获取最大输出行数函数。
INFA_CTASetOutputRowMax()	设置最大输出行数函数。

函数	说明
INFA_CTAGetNumRows()	获取行数函数。
INFA_CTASetNumRows()	设置行数函数。
INFA_CTAIsRowValid()	行是否有效函数。
INFA_CTAGetData<datatype>()	获取数据函数。
INFA_CTAGetIndicator()	获取指示器函数。
INFA_CTASetData()	设置数据函数。
INFA_CTAGetRowStrategy()	获取行策略函数。
INFA_CTASetRowStrategy()	设置行策略函数。
INFA_CTASetInputErrorRowM()	设置 MBCS 的输入错误行函数。
INFA_CTASetInputErrorRowU()	设置 Unicode 的输入错误行函数。

# 使用行

集成服务可以将单个行传递到自定义转换过程或数组中的行块。您可以编写过程代码以指定过程接收一行还是行块。过程接收行块时，可以提高性能：

- 可以降低集成服务和过程执行的函数调用数量。集成服务调用输入行通知函数的次数变少，过程调用输出通知函数的次数也变少。
- 可以增加数据的内存访问区域空间。
- 可以编写过程代码以针对数据块而非每行数据执行算法。

默认情况下，该过程每次接收一行数据。要接收行块，必须包括 INFA\_CTSetDataAccessMode() 函数，以便将数据访问模式更改为基于数组。当数据访问模式为基于数组时，必须使用基于数组的数据处理和行策略函数，才能访问和输出数据。当数据访问模式为基于行时，必须使用基于行的数据处理和行策略函数，才能访问和输出数据。

所有基于数组的函数均使用前缀 INFA\_CTA。所有其他函数均使用前缀 INFA\_CT。

请使用以下步骤编写过程代码，以访问行块：

- 在过程初始化期间调用 INFA\_CTSetDataAccessMode()，以将数据访问模式更改为基于数组。
- 在创建被动自定义转换时，还可以在过程初始化期间调用 INFA\_CTSetPassThruPort()，以传递数据用于输入/输出端口。  
当行块到达自定义转换过程时，集成服务将为每个数据块调用 p\_<proc\_name>\_inputRowNotification()。在此函数内部执行其余步骤。
- 在输入行通知函数中使用输入组句柄调用 INFA\_CTAGetNumRows()，以查找当前块中的行数。
- 使用输入端口句柄调用某一 INFA\_CTAGetData<datatype>() 函数，以便为块中的特定行获取数据。
- 调用 INFA\_CTASetData，以便在块中输出行。

6. 在调用 `INFA_CTOutputNotification()` 之前，先调用 `INFA_CTASetNumRows()`，以通知集成服务该过程在块中输出的行数。
7. 调用 `INFA_CTOutputNotification()`。

## 基于行和基于数组的数据访问模式的规则和准则

在编写过程代码以使用基于行或基于数组的数据访问模式时，请使用以下规则和准则：

- 在基于行的模式下，可以在输入行通知函数中返回 `INFA_ROWERROR`，以指示该函数在输入数据行时遇到了错误。集成服务增加内部错误计数。
- 在基于数组的模式下，请勿在输入行通知函数中返回 `INFA_ROWERROR`。集成服务会将其视为致命错误。如果需要指示块中的某行具有错误，请调用 `INFA_CTASetInputErrorRowM()` 或 `INFA_CTASetInputErrorRowU()` 函数。
- 在基于行的模式下，集成服务仅会将有效的行传递到过程中。
- 在基于阵列的模式下，输入块可以包含无效的行，如被删除的行、被筛选掉的行，或错误行。请调用 `INFA_CTAIsRowValid()` 来确定块中的某一行是否有效。
- 在基于阵列的模式下，请勿为被动自定义转换调用 `INFA_CTASetNumRows()`。可以为活动自定义转换调用此函数。
- 在基于数组的模式下，请调用一次 `INFA_CTOutputNotification()`。
- 在基于数组的模式下，只能为被动自定义转换调用 `INFA_CTSetPassThruPort()`。
- 在基于数组的模式下，对于被动自定义转换，必须在一个输入块中输出所有行，包括任何错误行。

## 生成的函数

使用 Designer 生成过程代码时，Designer 将在 `m_<module_name>.c` 和 `p_<procedure_name>.c` 文件中加入一组函数，称为生成的函数。集成服务使用生成的函数与过程进行交互。运行会话时，集成服务将按以下顺序为映射中的每个目标加载顺序组调用这些生成的函数：

1. 初始化函数
2. 通知函数
3. 取消初始化函数

### 初始化函数

集成服务首先调用初始化函数。使用初始化函数可写入您希望集成服务在将数据传递到自定义转换前运行的进程。在初始化函数中写入代码可降低处理开销，因为集成服务对于模块、过程或分区仅运行这些进程一次。

Designer 将生成以下初始化函数：

- `m_<module_name>_moduleInit()`
- `p_<proc_name>_procInit()`
- `p_<proc_name>_partitionInit()`

### 模块初始化函数

集成服务在会话初始化期间及运行会话前任务之前调用 `m_<module_name>_moduleInit()` 函数。它将在所有其他函数之前为某个模块调用此函数一次。

如果您希望集成服务在加载该模块时运行特定的进程，必须将其加入到此函数中。例如，您可以写入代码来创建此模块内的过程访问的全局结构。

请使用以下语法：

```
INFA_STATUS m_<module_name>_moduleInit(INFA_CT_MODULE_HANDLE module);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
模块	INFA_CT_MODULE_HANDLE	输入	模块句柄。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

过程初始化函数

集成服务在会话初始化期间、运行会话前任务前及运行模块初始化函数后调用 p\_<proc\_name>\_procInit() 函数。集成服务对于模块中的每个过程调用此函数一次。

当您希望集成服务对于某个特定过程运行进程时，可在此函数中写入代码。也可以在过程初始化函数中输入一些 API 函数，例如导航和属性函数。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_procInit(INFA_CT_PROCEDURE_HANDLE procedure);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
过程	INFA_CT_PROCEDURE_HANDLE	输入	过程句柄。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

分区初始化函数

集成服务在将数据传递到自定义转换前调用 p\_<proc\_name>\_partitionInit() 函数。集成服务将针对自定义转换实例的每个分区调用此函数一次。

如果您希望集成服务在通过自定义转换的分区传递数据之前运行特定的进程，必须将其加入到此函数中。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_partitionInit(INFA_CT_PARTITION_HANDLE transformation);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
转换	INFA_CT_PARTITION_HANDLE	输入	分区句柄。



返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

**注意:** 当自定义转换对于每个分区需要一个线程时，可以在分区初始化函数中加入线程特定的操作。

## 通知函数

将数据行传递到自定义转换时，集成服务将调用通知函数。

Designer 将生成以下通知函数：

- p\_<proc\_name>\_inputRowNotification()
- p\_<proc\_name>\_dataBdryRowNotification()
- p\_<proc\_name>\_eofNotification()

**注意:** 当自定义转换对于每个分区需要一个线程时，可以在通知函数中加入线程特定的操作。

## 输入行通知函数

将一行或行块传递到自定义转换时，集成服务将调用 p\_<proc\_name>\_inputRowNotification() 函数。它会记录通过输入组句柄和分区句柄接收数据的输入组和分区。

请使用以下语法：

```
INFA_ROWSTATUS p_<proc_name>_inputRowNotification(INFA_CT_PARTITION_HANDLE Partition,  
INFA_CT_INPUTGROUP_HANDLE group);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
分区	INFA_CT_PARTITION_HANDLE	输入	分区句柄。
组	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。

返回值的数据类型为 INFA\_ROWSTATUS。将以下值用于返回值：

- **INFA\_ROWSUCCESS**。指示成功处理数据行的函数。
- **INFA\_ROWERROR**。指示遇到数据行错误的函数。集成服务增加内部错误计数。当数据访问模式为行时仅返回此值。  
如果输入行通知函数在基于数组的模式下返回 INFA\_ROWERROR，集成服务则将其视为致命错误。如果需要指示块中的某行具有错误，请调用 INFA\_CTASetInputErrorRowM() 或 INFA\_CTASetInputErrorRowU() 函数。
- **INFA\_FATALERROR**。指示遇到数据行或数据块致命错误的函数。集成服务处理会话失败。

## 数据边界通知函数

将提交或回滚行传递到分区时，集成服务将调用 p\_<proc\_name>\_dataBdryNotification() 函数。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_dataBdryNotification(INFA_CT_PARTITION_HANDLE transformation,  
INFA_CTDataBdryType dataBoundaryType);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
转换	INFA_CT_PARTITION_HANDLE	输入	分区句柄。
dataBoundaryType	INFA_CTDataBdryType	输入	对于 dataBoundaryType 参数，集成服务将使用以下值之一： - eBT_COMMIT - eBT_ROLLBACK

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

### 文件末尾通知函数

将最后一行传递到输入组中的分区后，集成服务将调用 p\_<proc\_name>\_eofNotification() 函数。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_eofNotification(INFA_CT_PARTITION_HANDLE transformation,
INFA_CT_INPUTGROUP_HANDLE group);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
转换	INFA_CT_PARTITION_HANDLE	输入	分区句柄。
组	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

### 取消初始化函数

集成服务在处理自定义转换的数据后将调用取消初始化函数。使用取消初始化函数可写入您希望集成服务在将所有数据行传递到自定义转换后运行的进程。

Designer 将生成以下取消初始化函数：

- p\_<proc\_name>\_partitionDeinit()
- p\_<proc\_name>\_procDeinit()
- m\_<module\_name>\_moduleDeinit()

**注意：**当自定义转换对于每个分区需要一个线程时，可以在初始化和取消初始化函数中加入线程特定的操作。

### 分区取消初始化函数

集成服务在调用 p\_<proc\_name>\_eofNotification() 或 p\_<proc\_name>\_abortNotification() 函数后将调用 p\_<proc\_name>\_partitionDeinit() 函数。集成服务将针对自定义转换的每个分区调用此函数一次。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_partitionDeinit(INFA_CT_PARTITION_HANDLE partition);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
分区	INFA_CT_PARTITION_HANDLE	输入	分区句柄。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

**注意：**当自定义转换对于每个分区需要一个线程时，可以在分区取消初始化函数中加入线程特定的操作。

过程取消初始化函数

对于映射中使用此过程的每个自定义转换实例的所有分区，集成服务将在调用 p\_<proc\_name>\_partitionDeinit() 后调用 p\_<proc\_name>\_procDeinit() 函数。

请使用以下语法：

```
INFA_STATUS p_<proc_name>_procDeinit(INFA_CT_PROCEDURE_HANDLE procedure, INFA_STATUS sessionStatus);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
过程	INFA_CT_PROCEDURE_HANDLE	输入	过程句柄。
sessionStatus	INFA_STATUS	输入	对于 sessionStatus 参数，集成服务将使用以下值之一： - INFA_SUCCESS.指示成功的会话。 - INFA_FAILURE.指示失败的会话。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

模块取消初始化函数

集成服务在运行会话后任务后调用 m\_<module\_name>\_moduleDeinit() 函数。它继所有其他函数之后对某个模块调用此函数一次。

请使用以下语法：

```
INFA_STATUS m_<module_name>_moduleDeinit(INFA_CT_MODULE_HANDLE module, INFA_STATUS sessionStatus);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
模块	INFA_CT_MODULE_HANDLE	输入	模块句柄。
sessionStatus	INFA_STATUS	输入	对于 sessionStatus 参数，集成服务将使用以下值之一： - INFA_SUCCESS.指示成功的会话。 - INFA_FAILURE.指示失败的会话。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。当该函数返回 INFA\_FAILURE 时，集成服务会话将失败。

## API 函数

PowerCenter 提供一组 API 函数，可用于创建转换逻辑。Designer 在生成源代码文件时，将加入源代码中的生成函数。将 API 函数添加到代码中以实施转换逻辑。该过程使用 API 函数与集成服务进行交互。您必须在过程 C 文件中对 API 函数进行编码。或者，也可以对模块 C 文件进行编码。

Informatica 提供以下 API 函数组：

- 设置数据访问模式
- 导航
- 属性
- 重新绑定数据类型
- 数据处理（基于行的模式）
- 设置传递端口
- 输出通知
- 数据边界输出通知
- 错误
- 会话日志消息
- 递增错误计数
- 已终止
- 编块
- 指针
- 更改字符串模式
- 设置数据代码页
- 行策略（基于行的模式）
- 更改默认行策略

Informatica 还提供基于数组的 API 函数。

## 设置数据访问模式函数

默认情况下，集成服务一次向自定义转换过程传递一行数据。但是，INFA\_CTSetDataAccessMode() 函数可用于将数据访问模式更改为基于数组的模式。将数据访问模式设置为基于数组的模式时，集成服务会在数组中以块的形式向过程传递多行。

将数据访问模式设置为基于数组的模式时，必须使用基于数组的版本的数据处理函数和行策略函数。如果使用基于行的数据处理或行策略函数并切换到基于数组的模式，则将得到非预期的结果。例如，DLL 或共享库可能会崩溃。

只能在过程初始化函数中使用该函数。

如果在过程代码中不使用该函数，则数据访问模式为基于行的模式。但是，如果您希望数据访问模式为基于行的模式，请包含该函数并将访问模式设置为基于行的模式。

请使用以下语法：

```
INFA_STATUS INFA_CTSetDataAccessMode( INFA_CT_PROCEDURE_HANDLE procedure, INFA_CT_DATA_ACCESS_MODE mode );
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
过程	INFA_CT_PROCEDURE_HANDLE	输入	过程名称。
模式	INFA_CT_DATA_ACCESS_MODE	输入	数据访问模式。 将以下值用于模式参数： - eDA_ROW - eDA_ARRAY

## 导航函数

当您希望过程在句柄层次结构中导航时，可使用导航函数。

PowerCenter 提供以下导航函数：

- INFA\_CTGetAncestorHandle()
- INFA\_CTGetChildrenHandles()
- INFA\_CTGetInputPortHandle()
- INFA\_CTGetOutputPortHandle()

### 获取 Ancestor 句柄函数

当希望过程访问指定句柄的父句柄时，可使用 INFA\_CTGetAncestorHandle() 函数。

请使用以下语法：

```
INFA_CT_HANDLE INFA_CTGetAncestorHandle(INFA_CT_HANDLE handle, INFA_CTHandleType returnHandleType);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_HANDLE	输入	句柄名称。
returnHandleType	INFA_CTHandleType	输入	返回句柄类型。 对于 returnHandleType 参数，可使用以下值： <ul style="list-style-type: none"><li>- PROCEDURETYPE</li><li>- TRANSFORMATIONTYPE</li><li>- PARTITIONTYPE</li><li>- INPUTGROUPTYPE</li><li>- OUTPUTGROUPTYPE</li><li>- INPUTPORTTYPE</li><li>- OUTPUTPORTTYPE</li></ul>

handle 参数指定您希望过程访问其父项的句柄。如果您在该函数中指定了有效的句柄，集成服务将返回 INFA\_CT\_HANDLE。否则将返回空值。

要避免编译错误，必须对过程编码以将句柄名称设置为返回值。

例如，可以输入以下代码：

```
INFA_CT_MODULE_HANDLE module = INFA_CTGetAncestorHandle(procedureHandle, INFA_CT_HandleType);
```

获取子句柄函数

当希望过程访问指定句柄的子句柄时，可使用 INFA\_CTGetChildrenHandles() 函数。

请使用以下语法：

```
INFA_CT_HANDLE* INFA_CTGetChildrenHandles(INFA_CT_HANDLE handle, size_t* pnChildrenHandles, INFA_CTHandleType returnHandleType);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_HANDLE	输入	句柄名称。
pnChildrenHandles	size_t*	输出	集成服务将返回子句柄的数组。 pnChildrenHandles 参数指示该数组中的子句柄数。
returnHandleType	INFA_CTHandleType	输入	对于 returnHandleType 参数，可使用以下值： <ul style="list-style-type: none"><li>- PROCEDURETYPE</li><li>- TRANSFORMATIONTYPE</li><li>- PARTITIONTYPE</li><li>- INPUTGROUPTYPE</li><li>- OUTPUTGROUPTYPE</li><li>- INPUTPORTTYPE</li><li>- OUTPUTPORTTYPE</li></ul>

handle 参数指定您希望过程访问其子项的句柄。如果您在该函数中指定了有效的句柄，集成服务将返回 INFA\_CT\_HANDLE\*。否则将返回空值。

要避免编译错误，必须对过程编码以将句柄名称设置为返回的值。

例如，可以输入以下代码：

```
INFA_CT_PARTITION_HANDLE partition = INFA_CTGetChildrenHandles(procedureHandle, pnChildrenHandles, INFA_CT_PARTITION_HANDLE_TYPE);
```

获取端口句柄函数

集成服务可将 INFA\_CT\_INPUTPORT\_HANDLE 与输入和输入/输出端口相关联，将 INFA\_CT\_OUTPUTPORT\_HANDLE 与输出和输入/输出端口相关联。

PowerCenter 提供以下获取端口句柄函数：

- **INFA\_CTGetInputPortHandle()**。当过程知道输入/输出端口的输出端口句柄并需要输入端口句柄时，可使用此函数。

请使用以下语法：

```
INFA_CTINFA_CT_INPUTPORT_HANDLE INFA_CTGetInputPortHandle(INFA_CT_OUTPUTPORT_HANDLE outputPortHandle);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
outputPortHandle	INFA_CT_OUTPUTPORT_HANDLE	输入	输出端口句柄。

- **INFA\_CTGetOutputPortHandle()**。当过程知道输入/输出端口的输入端口句柄并需要输出端口句柄时，可使用此函数。

请使用以下语法：

```
INFA_CT_OUTPUTPORT_HANDLE INFA_CTGetOutputPortHandle(INFA_CT_INPUTPORT_HANDLE inputPortHandle);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputPortHandle	INFA_CT_INPUTPORT_HANDLE	输入	输入端口句柄。

将获取端口句柄函数用于输入或输出端口时，集成服务将返回空值。

属性函数

当希望过程访问自定义转换属性时，可使用属性函数。属性函数可访问自定义转换的以下选项卡中的属性：

- 端口
- 属性
- 初始化属性
- 元数据扩展
- 端口属性定义

使用初始化函数中的下列属性函数：

- `INFA_CTGetInternalProperty<datatype>()`
- `INFA_CTGetAllPropertyNamesM()`
- `INFA_CTGetAllPropertyNamesU()`
- `INFA_CTGetExternalProperty<datatype>M()`
- `INFA_CTGetExternalProperty<datatype>U()`

## 获取内部属性函数

PowerCenter 提供的函数可访问自定义转换“端口”选项卡上指定的端口属性以及为“属性”选项卡上的特性指定的属性。

集成服务将每个端口和属性特性与属性 ID 关联起来。您在过程中必须指定属性 ID，才能访问为这些属性指定的值。对于句柄参数，指定句柄层次结构中的某个句柄名称。如果句柄名称无效，集成服务会话将失败。

当希望过程访问属性时，可使用以下函数：

- **`INFA_CTGetInternalPropertyStringM()`**.在 MBCS 模式下访问指定属性 ID 的字符串类型的值。

请使用以下语法：

```
INFA_STATUS INFA_CTGetInternalPropertyStringM( INFA_CT_HANDLE handle, size_t propId, const char**
psPropValue );
```

- **`INFA_CTGetInternalPropertyStringU()`**.在 Unicode 模式下访问指定属性 ID 的字符串类型的值。

请使用以下语法：

```
INFA_STATUS INFA_CTGetInternalPropertyStringU( INFA_CT_HANDLE handle, size_t propId, const
INFA_UNICHAR** psPropValue );
```

- **`INFA_CTGetInternalPropertyInt32()`**.访问指定属性 ID 的整数类型的值。

请使用以下语法：

```
INFA_STATUS INFA_CTGetInternalPropertyInt32( INFA_CT_HANDLE handle, size_t propId, INFA_INT32*
pnPropValue );
```

- **`INFA_CTGetInternalPropertyBool()`**.访问指定属性 ID 的布尔类型的值。

请使用以下语法：

```
INFA_STATUS INFA_CTGetInternalPropertyBool( INFA_CT_HANDLE handle, size_t propId, INFA_BOOLEAN*
pbPropValue );
```

- **`INFA_CTGetInternalPropertyINFA_PTR()`**.访问指向指定属性 ID 的值的指针。

请使用以下语法：

```
INFA_STATUS INFA_CTGetInternalPropertyINFA_PTR( INFA_CT_HANDLE handle, size_t propId, INFA_PTR*
pvPropValue );
```

返回值的数据类型为 `INFA_STATUS`。对于返回值使用 `INFA_SUCCESS` 和 `INFA_FAILURE`。

## 端口和属性特性属性 ID

下表列出了用于自定义转换的端口和属性特性的属性 ID。每个表都列出了自定义函数句柄以及使用属性函数中的句柄可访问的属性 ID。



下表列出了 INFA\_CT\_MODULE\_HANDLE 属性 ID：

句柄属性 ID	数据类型	说明
INFA_CT_MODULE_NAME	字符串	指定模块名称。
INFA_CT_SESSION_INFA_VERSION	字符串	指定 Informatica 版本。
INFA_CT_SESSION_CODE_PAGE	整型	指定集成服务代码页。
INFA_CT_SESSION_DATA_MOVEMENT_MODE	整型	指定数据移动模式。集成服务将返回以下值之一： - eASM_MBCS - eASM_UNICODE
INFA_CT_SESSION_VALIDATE_CODEPAGE	布尔型	指定集成服务是否强制进行代码页验证。
INFA_CT_SESSION_PROD_INSTALL_DIR	字符串	指定集成服务安装目录。
INFA_CT_SESSION_HIGH_PRECISION_MODE	布尔型	指定是否为会话配置了高精度。
INFA_CT_MODULE_RUNTIME_DIR	字符串	指定 DLL 或共享库的运行时目录。
INFA_CT_SESSION_IS_UPD_STR_ALLOWED	布尔型	指定转换中是否选择了“更新策略转换”属性。
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	整型	指定自定义转换是否按每个会话中运行的相同顺序生成数据。集成服务将返回以下值之一： - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	布尔型	指定自定义转换是否导致了致命错误。集成服务将返回以下值之一： - INFA_TRUE - INFA_FALSE

下表列出了 INFA\_CT\_PROC\_HANDLE 属性 ID：

句柄属性 ID	数据类型	说明
INFA_CT_PROCEDURE_NAME	字符串	指定自定义转换过程名称。

下表列出了 INFA\_CT\_TRANS\_HANDLE 属性 ID：

句柄属性 ID	数据类型	说明
INFA_CT_TRANS_INSTANCE_NAME	字符串	指定自定义转换实例名称。
INFA_CT_TRANS_TRACE_LEVEL	整型	指定跟踪级别。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- eTRACE_TERSE</li> <li>- eTRACE_NORMAL</li> <li>- eTRACE_VERBOSE_INIT</li> <li>- eTRACE_VERBOSE_DATA</li> </ul>
INFA_CT_TRANS_MAY_BLOCK_DATA	布尔型	指定集成服务是否允许过程阻止当前会话中的输入数据。
INFA_CT_TRANS_MUST_BLOCK_DATA	布尔型	指定是否选择了“必须阻止输入”自定义转换属性。
INFA_CT_TRANS_ISACTIVE	布尔型	指定自定义转换是主动转换还是被动转换。
INFA_CT_TRANS_ISPARTITIONABLE	布尔型	指定是否可以对使用此自定义转换的会话分区。
INFA_CT_TRANS_IS_UPDATE_STRATEGY	布尔型	指定自定义转换的行为是否与更新策略转换相同。
INFA_CT_TRANS_DEFAULT_UPDATE_STRATEGY	整型	指定默认更新策略。 <ul style="list-style-type: none"> <li>- eDUS_INSERT</li> <li>- eDUS_UPDATE</li> <li>- eDUS_DELETE</li> <li>- eDUS_REJECT</li> <li>- eDUS_PASSTHROUGH</li> </ul>
INFA_CT_TRANS_NUM_PARTITIONS	整型	指定使用此自定义转换的会话中的分区数。
INFA_CT_TRANS_DATA_CODEPAGE	整型	指定集成服务从中传递数据到自定义转换的代码页。如果希望自定义转换访问其他代码页中的数据，可使用设置数据代码页函数。
INFA_CT_TRANS_TRANSFORM_SCOPE	整型	指定自定义转换中的转换范围。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- eTS_ROW</li> <li>- eTS_TRANSACTION</li> <li>- eTS_ALLINPUT</li> </ul>
INFA_CT_TRANS_GENERATE_TRANSACTION	布尔型	指定是否启用了“生成事务”属性。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- INFA_TRUE</li> <li>- INFA_FALSE</li> </ul>

句柄属性 ID	数据类型	说明
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	整型	指定自定义转换是否按每个会话中运行的相同顺序生成数据。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- eOUTREPEAT_NEVER = 1</li> <li>- eOUTREPEAT_ALWAYS = 2</li> <li>- eOUTREPEAT_BASED_ON_INPUT_ORDER = 3</li> </ul>
INFA_CT_TRANS_FATAL_ERROR	布尔型	指定自定义转换是否导致了致命错误。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- INFA_TRUE</li> <li>- INFA_FALSE</li> </ul>

下表列出了 INFA\_CT\_INPUT\_GROUP\_HANDLE 和 INFA\_CT\_OUTPUT\_GROUP\_HANDLE 属性 ID：

句柄属性 ID	数据类型	说明
INFA_CT_GROUP_NAME	字符串	指定组名称。
INFA_CT_GROUP_NUM_PORTS	整型	指定组中的端口数。
INFA_CT_GROUP_ISCONNECTED	布尔型	指定组中的所有端口是否均已连接到另一个转换。
INFA_CT_PORT_NAME	字符串	指定端口名称。
INFA_CT_PORT_CDATATYPE	整型	指定端口数据类型。集成服务将返回以下值之一： <ul style="list-style-type: none"> <li>- eINFA_CTYPE_SHORT</li> <li>- eINFA_CTYPE_INT32</li> <li>- eINFA_CTYPE_CHAR</li> <li>- eINFA_CTYPE_RAW</li> <li>- eINFA_CTYPE_UNICHAR</li> <li>- eINFA_CTYPE_TIME</li> <li>- eINFA_CTYPE_FLOAT</li> <li>- eINFA_CTYPE_DOUBLE</li> <li>- eINFA_CTYPE_DECIMAL18_FIXED</li> <li>- eINFA_CTYPE_DECIMAL28_FIXED</li> <li>- eINFA_CTYPE_INFA_CT_DATETIME</li> </ul>
INFA_CT_PORT_PRECISION	整型	指定端口精度。
INFA_CT_PORT_SCALE	整型	指定端口小数位数（如果适用）。
INFA_CT_PORT_IS_MAPPED	布尔型	指定该端口是否链接到了映射中的其他转换。
INFA_CT_PORT_STORAGE_SIZE	整型	指定端口的内部数据存储大小。存储大小因端口的数据类型而异。
INFA_CT_PORT_BOUND_DATATYPE	整型	指定端口数据类型。如果要重新绑定端口并指定默认值以外的数据类型，可使用此函数替代 INFA_CT_PORT_CDATATYPE。

下表列出了 INFA\_CT\_INPUTPORT\_HANDLE 和 INFA\_CT\_OUTPUT\_HANDLE 属性 ID：

句柄属性 ID	数据类型	说明
INFA_CT_PORT_NAME	字符串	指定端口名称。
INFA_CT_PORT_CDATATYPE	整型	指定端口数据类型。集成服务将返回以下值之一： <ul style="list-style-type: none"><li>- eINFA_CTYPE_SHORT</li><li>- eINFA_CTYPE_INT32</li><li>- eINFA_CTYPE_CHAR</li><li>- eINFA_CTYPE_RAW</li><li>- eINFA_CTYPE_UNICHAR</li><li>- eINFA_CTYPE_TIME</li><li>- eINFA_CTYPE_FLOAT</li><li>- eINFA_CTYPE_DOUBLE</li><li>- eINFA_CTYPE_DECIMAL18_FIXED</li><li>- eINFA_CTYPE_DECIMAL28_FIXED</li><li>- eINFA_CTYPE_INFA_CTDATETIME</li></ul>
INFA_CT_PORT_PRECISION	整型	指定端口精度。
INFA_CT_PORT_SCALE	整型	指定端口小数位数（如果适用）。
INFA_CT_PORT_IS_MAPPED	布尔型	指定该端口是否链接到了映射中的其他转换。
INFA_CT_PORT_STORAGESIZE	整型	指定端口的内部数据存储大小。存储大小因端口的数据类型而异。
INFA_CT_PORT_BOUNDDATATYPE	整型	指定端口数据类型。如果要重新绑定端口并指定默认值以外的数据类型，可使用此函数替代 INFA_CT_PORT_CDATATYPE。

### 获取所有外部属性名称（MBCS 或 Unicode）

PowerCenter 提供两个函数，可用来访问自定义转换“元数据扩展”、“初始化属性”和“端口属性定义”选项卡上定义的属性名称。

当希望过程访问属性名称时，可使用以下函数：

- **INFA\_CTGetAllPropertyNamesM()**。访问 MBCS 中的属性名称。

请使用以下语法：

```
INFA_STATUS INFA_CTGetAllPropertyNamesM(INFA_CT_HANDLE handle, const char*const** paPropertyNames,
size_t* pnProperties);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_HANDLE	输入	指定句柄名称。
paPropertyNames	const char*const**	输出	指定属性名称。集成服务将返回 MBCS 模式下的属性名称数组。
pnProperties	size_t*	输出	指示数组中的属性数。

- **INFA\_CTGetAllPropertyNamesU()**.在 Unicode 模式下访问属性名称。

请使用以下语法：

```
INFA_STATUS INFA_CTGetAllPropertyNamesU(INFA_CT_HANDLE handle, const INFA_UNICHAR*const**
pasPropertyNames, size_t* pnProperties);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_HANDLE	输入	指定句柄名称。
paPropertyNames	const INFA_UNICHAR*const**	输出	指定属性名称。集成服务将返回 Unicode 模式下的属性名称数组。
pnProperties	size_t*	输出	指示数组中的属性数。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

获取外部属性（MBCS 或 Unicode）

PowerCenter 提供用来访问自定义转换“元数据扩展”、“初始化属性”和“端口属性定义”选项卡上定义的属性值的函数。

如果希望过程访问这些值，则必须在函数中指定属性名称。使用 INFA\_CTGetAllPropertyNamesM() 或 INFA\_CTGetAllPropertyNamesU() 函数可访问属性名称。对于句柄参数，指定句柄层次结构中的某个句柄名称。如果句柄名称无效，集成服务会话将失败。

**注意:** 如果使用与元数据扩展相同的名称定义初始化属性，集成服务将返回元数据扩展值。

当希望过程访问属性值时，可使用以下函数：

- **INFA\_CTGetExternalProperty<datatype>M()**.在 MBCS 模式下访问属性值。

下表显示了语法：

语法	属性数据类型
INFA_STATUS INFA_CTGetExternalPropertyStringM(INFA_CT_HANDLE handle, const char* sPropName, const char** psPropValue);	字符串
INFA_STATUS INFA_CTGetExternalPropertyINT32M(INFA_CT_HANDLE handle, const char* sPropName, INFA_INT32* pnPropValue);	整型
INFA_STATUS INFA_CTGetExternalPropertyBoolM(INFA_CT_HANDLE handle, const char* sPropName, INFA_BOOLEAN* pbPropValue);	布尔型

- **INFA\_CTGetExternalProperty<datatype>U()**.在 Unicode 模式下访问属性值。

下表显示了语法：

语法	属性数据类型
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_UNICHAR** psPropValue);	字符串
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_INT32* pnPropValue);	整型
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_BOOLEAN* pbPropValue);	布尔型

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 重新绑定数据类型函数

可以为端口重新绑定 PowerCenter 默认数据类型以外的数据类型。如果希望过程访问默认数据类型以外数据类型的数据，可使用重新绑定数据类型函数。必须为端口重新绑定兼容的数据类型。

只能在初始化函数中使用这些函数。

在为输出或输入/输出端口重新绑定数据类型时，请遵循以下规则：

- 必须使用数据处理函数设置数据和该端口的指示器。在基于行的模式下使用 INFA\_CTSetData() 和 INFA\_CTSetIndicator() 函数，并在基于数组的模式下使用 INFA\_CTASetData() 函数。
- 不要对输出端口调用 INFA\_CTSetPassThruPort() 函数。

下表列出了兼容的数据类型：

默认数据类型	兼容的数据类型
字符	Unichar
Unichar	字符

默认数据类型	兼容的数据类型
日期	INFA_DATETIME 请使用以下语法：  <pre>struct INFA_DATETIME {     int nYear;     int nMonth;     int nDay;     int nHour;     int nMinute;     int nSecond;     int nNanoSecond; }</pre>
Dec18	Char、Unichar
Dec28	Char、Unichar

PowerCenter 提供以下重新绑定数据类型函数：

- **INFA\_CTRebindInputDataType()**。重新绑定输入端口。请使用以下语法：  

```
INFA_STATUS INFA_CTRebindInputDataType(INFA_CT_INPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);
```
  - **INFA\_CTRebindOutputDataType()**。重新绑定输出端口。请使用以下语法：  

```
INFA_STATUS INFA_CTRebindOutputDataType(INFA_CT_OUTPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);
```
- 下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
portHandle	INFA_CT_OUTPUTPORT_HANDLE	输入	输出端口句柄。
数据类型	INFA_CDATATYPE	输入	<p>为端口重新绑定的数据类型。对于数据类型参数，可使用以下值：</p> <ul style="list-style-type: none"> <li>- eINFA_CTYPE_SHORT</li> <li>- eINFA_CTYPE_INT32</li> <li>- eINFA_CTYPE_CHAR</li> <li>- eINFA_CTYPE_RAW</li> <li>- eINFA_CTYPE_UNICHAR</li> <li>- eINFA_CTYPE_TIME</li> <li>- eINFA_CTYPE_FLOAT</li> <li>- eINFA_CTYPE_DOUBLE</li> <li>- eINFA_CTYPE_DECIMAL18_FIXED</li> <li>- eINFA_CTYPE_DECIMAL28_FIXED</li> <li>- eINFA_CTYPE_INFA_CTDATETIME</li> </ul>

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 数据处理函数（基于行的模式）

在集成服务调用输入行通知函数时，它会通知过程其可访问数据行或数据块。但要获取输入端口中的数据、进行修改及设置输出端口中的数据，则必须在输入行通知函数中使用数据处理函数。如果是基于行的数据访问模式，请使用基于行的数据处理函数。

加入 INFA\_CTGetData<datatype>() 函数可获取输入端口中的数据，加入 INFA\_CTSetData() 函数可设置输出端口中的数据。如果希望过程在获取数据前确认端口是否拥有空值或空字符串，可加入 INFA\_CTGetIndicator() 或 INFA\_CTGetLength() 函数。

PowerCenter 提供以下数据处理函数：

- INFA\_CTGetData<datatype>()
- INFA\_CTSetData()
- INFA\_CTGetIndicator()
- INFA\_CTSetIndicator()
- INFA\_CTGetLength()
- INFA\_CTSetLength()

### 获取数据函数（基于行的模式）

使用 INFA\_CTGetData<datatype>() 函数可检索该函数指定的端口的数据。

您必须根据希望过程访问的端口的数据类型修改函数名称。

下表列出了 INFA\_CTGetData<datatype>() 函数语法和返回值的数据类型：

语法	返回值数据类型
<code>void* INFA_CTGetDataVoid(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	指向返回值的 数据空型指针
<code>char* INFA_CTGetDataStringM(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	字符串 (MBCS)
<code>IUNICHAR* INFA_CTGetDataStringU(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	字符串 (Unicode)
<code>INFA_INT32 INFA_CTGetDataINT32(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	整型
<code>double INFA_CTGetDataDouble(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	双精度
<code>INFA_CT_RAWDATE INFA_CTGetDataDate(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	原始日期
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18( INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (精度 18)
<code>INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28( INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (精度 28)
<code>INFA_CT_DATETIME INFA_CTGetDataDateTime(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	日期时间



## 设置数据函数（基于行的模式）

如果希望过程将值传递到输出端口，请使用 INFA\_CTSetData() 函数。

请使用以下语法：

```
INFA_STATUS INFA_CTSetData(INFA_CT_OUTPUTPORT_HANDLE dataHandle, void* data);
```

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

**注意:** 如果针对输入/输出端口使用 INFA\_CTSetPassThruPort() 函数，请勿将设置数据或指示器用于该端口。

## 指示器函数（基于行的模式）

当希望过程获取输入端口的指示器或设置输出端口的指示器时，可使用指示器函数。端口指示器指示数据有效、是空值还是被截断。

PowerCenter 提供以下指示器函数：

- **INFA\_CTGetIndicator()**。获取输入端口的指示器。请使用以下语法：

```
INFA_INDICATOR INFA_CTGetIndicator(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

返回值的数据类型为 INFA\_INDICATOR。对于 INFA\_INDICATOR，可使用以下值：

- **INFA\_DATA\_VALID**.指示数据有效。
- **INFA\_NULL\_DATA**.指示空值。
- **INFA\_DATA\_TRUNCATED**.指示数据被截断。

- **INFA\_CTSetIndicator()**。设置输出端口的指示器。请使用以下语法：

```
INFA_STATUS INFA_CTSetIndicator(INFA_CT_OUTPUTPORT_HANDLE dataHandle, INFA_INDICATOR indicator);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
dataHandle	INFA_CT_OUTPUTPORT_HANDLE	输入	输出端口句柄。
指示器	INFA_INDICATOR	输入	输出端口的指示器值。使用以下值之一： <ul style="list-style-type: none"><li>- INFA_DATA_VALID。指示数据有效。</li><li>- INFA_NULL_DATA。指示空值。</li><li>- INFA_DATA_TRUNCATED。指示数据被截断。</li></ul>

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

**注意:** 如果在输入/输出端口上使用 INFA\_CTSetPassThruPort() 函数，请勿为该端口设置数据或指示器。

## 长度函数

当希望过程访问字符串或二进制输入端口的长度或设置二进制或字符串输出端口的长度时，可使用长度函数。

使用以下长度函数：

- **INFA\_CTGetLength()**。此函数仅用于字符串和二进制端口。集成服务将以字符数形式返回长度（包括尾随空格）。请使用以下语法：

```
INFA_UINT32 INFA_CTGetLength(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

返回值的数据类型为 INFA\_UINT32。返回值使用介于零和 2GB 之间的值。

- **INFA\_CTSetLength()**。如果自定义转换包含二进制或字符串输出端口，则必须使用此函数设置数据长度（包括尾随空格）。确保您为字符串和二进制端口设置的长度不高于该端口的精度。如果设置的长度高于该端口精度，则会获得异常结果。例如，会话可能失败。

请使用以下语法：

```
INFA_STATUS INFA_CTSetLength(INFA_CT_OUTPUTPORT_HANDLE dataHandle, IUINT32 length);
```

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 设置传递端口函数

如果希望集成服务将数据从输入端口传递到输出端口而不修改数据，请使用 INFA\_CTSetPassThruPort() 函数。当使用 INFA\_CTSetPassThruPort() 函数时，集成服务会在调用输入行通知函数时将数据传递到输出端口。

在使用设置传递端口函数时考虑以下规则和准则：

- 仅在初始化函数中使用该函数。
- 如果过程包括该函数，请不要包括 INFA\_CTSetData()、INFA\_CTSetLength、INFA\_CTSetIndicator() 或 INFA\_CTASetData() 函数来将数据传递到输出端口。
- 在基于行的模式中，只有在转换范围为“行”时才能包括该函数。当转换范围为“事务”或“全部输入”时，该函数会返回 INFA\_FAILURE。
- 在基于行的模式中，当使用该函数为指定的输入行输出多行时，每个输出行都包含从输入端口传递的数据。
- 在基于数组的模式中，只能对被动自定义转换使用该函数。

必须验证数据类型、精度和小数位数对输入和输出端口是否是相同的。如果数据类型、精度或小数位数对您在 INFA\_CTSetPassThruPort() 函数中指定的输入和输出端口是不同的，则集成服务会失败。

请使用以下语法：

```
INFA_STATUS INFA_CTSetPassThruPort(INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_CT_INPUTPORT_HANDLE inputport)
```

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 输出通知函数

当希望过程输出行到集成服务时，可使用 INFA\_CTOutputNotification() 函数。只能为活动自定义转换加入此函数。对于被动自定义转换，过程会在输入行通知函数返回值时输出行到集成服务。如果过程为被动自定义转换调用此函数，集成服务将忽略该函数。

**注意：**当转换范围为“行”时，只能在输入行通知函数中加入此函数。如果将其加入到其他函数中，则会返回失败。

请使用以下语法：

```
INFA_ROWSTATUS INFA_CTOutputNotification(INFA_CT_OUTPUTGROUP_HANDLE group);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
组	INFA_CT_OUTPUT_GROUP_HANDLE	输入	输出组句柄。

返回值的数据类型为 INFA\_ROWSTATUS。将以下值用于返回值：

- **INFA\_ROWSUCCESS**。指示成功处理数据行的函数。

- **INFA\_ROWERROR。**指示遇到数据行错误的函数。集成服务增加内部错误计数。
- **INFA\_FATALERROR。**指示遇到数据行致命错误的函数。集成服务处理会话失败。

**注意:** 在过程代码调用 INFA\_CTOutputNotification() 函数时，必须确认输出端口句柄中的所有指针均指向有效数据。如果某个指针未指向有效数据，集成服务可能会异常关闭。

## 数据边界输出通知函数

当希望过程输出提交或回滚事务时，可加入 INFA\_CTDataBdryOutputNotification() 函数。

使用此函数时，必须选择此自定义转换的“生成事务”属性。如果未选择此属性，集成服务会话将失败。

请使用以下语法：

```
INFA_STATUS INFA_CTDataBdryOutputNotification(INFA_CT_PARTITION_HANDLE handle, INFA_CTDataBdryType dataBoundaryType);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_PARTITION_HANDLE	输入	句柄名称。
dataBoundaryType	INFA_CTDataBdryType	输入	事务类型。 对于 dataBoundaryType 参数，可使用以下值： - eBT_COMMIT - eBT_ROLLBACK

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 错误函数

使用错误函数可访问过程错误。集成服务返回最新的错误。

PowerCenter 提供以下错误函数：

- **INFA\_CTGetErrorMsgM()。**在 MBCS 模式下获取错误消息。请使用以下语法：  

```
const char* INFA_CTGetErrorMsgM();
```
- **INFA\_CTGetErrorMsgU()。**在 Unicode 模式下获取错误消息。请使用以下语法：  

```
const IUNICHAR* INFA_CTGetErrorMsgU();
```

## 会话日志消息函数

如果想要过程以 Unicode 或 MBCS 格式在会话日志中记录消息，请使用会话日志消息函数。

PowerCenter 提供以下会话日志消息函数：

- **INFA\_CTLogMessageU()。**以 Unicode 格式记录消息。

请使用以下语法：

```
void INFA_CTLogMessageU(INFA_CT_ErrorSeverityLevel errorseverityLevel, INFA_UNICHAR* msg)
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	输入	您想要集成服务在会话日志中写入的错误消息的严重级别。将以下值用于 errorSeverityLevel 参数： - eESL_LOG - eESL_DEBUG - eESL_ERROR
msg	INFA_UNICHAR*	输入	输入用引号引起的 Unicode 消息文本。

- **INFA\_CTLogMessageM()**。以 MBCS 格式记录消息。

请使用以下语法：

```
void INFA_CTLogMessageM(INFA_CT_ErrorSeverityLevel errorSeverityLevel, char* msg)
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	输入	您想要集成服务在会话日志中写入的错误消息的严重级别。将以下值用于 errorSeverityLevel 参数： - eESL_LOG - eESL_DEBUG - eESL_ERROR
msg	char*	输入	输入用引号引起的 MBCS 消息文本。

## 递增错误计数函数

当希望增加会话的错误计数时，可使用 INFA\_CTIncrementErrorCount() 函数。

请使用以下语法：

```
INFA_STATUS INFA_CTIncrementErrorCount(INFA_CT_PARTITION_HANDLE transformation, size_t nErrors, INFA_STATUS* pStatus);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
转换	INFA_CT_PARTITION_HANDLE	输入	分区句柄。
nErrors	size_t	输入	集成服务按 nErrors 增加指定转换实例的错误计数。
pStatus	INFA_STATUS*	输入	当错误计数超出错误阈值且会话失败时，集成服务对于 pStatus 参数将使用 INFA_FAILURE。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 已终止函数

当希望过程检查 PowerCenter 客户端是否已请求集成服务停止会话时，可使用 INFA\_CTIsTerminated() 函数。如果过程包括耗时的进程，可调用此函数。

请使用以下语法：

```
INFA_CTTerminateType INFA_CTIsTerminated(INFA_CT_PARTITION_HANDLE handle);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
句柄	INFA_CT_PARTITION_HANDLE	输入	分区句柄。

返回值的数据类型为 INFA\_CTTerminateType。集成服务将返回以下值之一：

- **eTT\_NOTTERMINATED**. 指示 PowerCenter 客户端尚未请求停止会话。
- **eTT\_ABORTED**. 指示集成服务已中止会话。
- **eTT\_STOPPED**. 指示集成服务会话已失败。

## 编块函数

当自定义转换包含多个输入组时，可以写入代码对输入组中的传入数据编块。

使用编块函数时，请遵循以下规则：

- 最多可对  $n-1$  个输入组编块。
- 不可以对已经编块的输入组编块。
- 如果接收的输入组数据源与其他输入组相同，则不能对该输入组编块。
- 不能对已经取消编块的输入组取消编块。

PowerCenter 提供以下编块函数：

- **INFA\_CTBlockInputFlow()**。允许过程对输入组编块。

请使用以下语法：

```
INFA_STATUS INFA_CTBlockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

- **INFA\_CTUnblockInputFlow()**。允许过程对输入组取消编块。

请使用以下语法：

```
INFA_STATUS INFA_CTUnblockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
组	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

### 验证编块

在过程代码中使用 INFA\_CTBlockInputFlow() 和 INFA\_CTUnblockInputFlow() 函数时，请验证过程会检查集成服务是否允许自定义转换阻止传入数据。要执行该操作，请使用 INFA\_CTGetInternalPropertyBool() 函数检查 INFA\_CT\_TRANS\_MAY\_BLOCK\_DATA propID 的值。

如果 INFA\_CT\_TRANS\_MAY\_BLOCK\_DATA propID 的值为 FALSE，则过程应不使用阻止函数，或应返回致命错误并停止该会话。

在集成服务不允许自定义转换阻止数据时，如果过程代码使用阻止函数，则集成服务可能会使会话失败。

## 指针函数

当希望集成服务创建和访问指向对象或结构的指针时，可使用指针函数。

PowerCenter 提供以下指针函数：

- **INFA\_CTGetUserDefinedPtr()**。运行时允许过程访问对象或结构。

请使用以下语法：

```
void* INFA_CTGetUserDefinedPtr(INFA_CT_HANDLE handle)
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
句柄	INFA_CT_HANDLE	输入	句柄名称。

- **INFA\_CTSetUserDefinedPtr()**。允许过程将对象或结构与集成服务提供的任何句柄关联。要降低处理开销，请在初始化函数中加入此函数。

请使用以下语法：

```
void INFA_CTSetUserDefinedPtr(INFA_CT_HANDLE handle, void* pPtr)
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
句柄	INFA_CT_HANDLE	输入	句柄名称。
pPtr	空型*	输入	用户指针。

必须用有效的句柄替代 INFA\_CT\_HANDLE。

## 更改字符串模式函数

当集成服务在 Unicode 模式下运行时，默认情况下会以 UCS-2 格式将数据传递到过程中。当它在 ASCII 模式下运行时，默认情况下会以 ASCII 格式传递数据。如果希望更改过程的默认字符串模式，可使用 INFA\_CTChangeStringMode() 函数。将默认字符串模式更改为 MBCS 后，集成服务将传递集成服务代码页中的数据。如果希望更改代码页，可使用 INFA\_CTSetDataCodePageID() 函数。

如果过程包括 INFA\_CTChangeStringMode() 函数，集成服务将更改使用此特定过程的每个自定义转换中所有端口的字符串模式。

请在初始化函数中使用更改字符串模式函数。

请使用以下语法：

```
INFA_STATUS INFA_CTChangeStringMode(INFA_CT_PROCEDURE_HANDLE procedure, INFA_CTStringMode stringMode);
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
过程	INFA_CT_PROCEDURE_HANDLE	输入	过程句柄名称。
stringMode	INFA_CTStringMode	输入	指定您希望集成服务使用的字符串模式。对于 stringMode 参数，可使用以下值： - eASM_UNICODE。当集成服务在 ASCII 模式下运行且您希望过程在 Unicode 模式下访问数据时，可使用此函数。 - eASM_MBCS。当集成服务在 Unicode 模式下运行且您希望过程在 MBCS 模式下访问数据时，可使用此函数。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 设置数据代码页函数

当您想要集成服务将数据传递到集成服务代码页以外的某个代码页中的自定义转换时，请使用 INFA\_CTSetDataCodePageID()。

在过程初始化函数中使用设置数据代码页函数。

请使用以下语法：

```
INFA_STATUS INFA_CTSetDataCodePageID(INFA_CT_TRANSFORMATION_HANDLE transformation, int dataCodePageID);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
transformation： 转换	INFA_CT_TRANSFORMATION_HANDLE	输入	转换句柄名称。
dataCodePageID	int	输入	指定您想要集成服务在其中传递数据的代码页。 对于 dataCodePageID 参数的有效值，请参阅《管理员指南》中的“代码页”。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

## 行策略函数（基于行的模式）

借助行策略函数，可以访问每一行的更新策略，并对这些更新策略进行配置。

PowerCenter 提供以下行策略函数：

- **INFA\_CTGetRowStrategy()**。允许过程为某一行获取更新策略。

请使用以下语法：

```
INFA_STATUS INFA_CTGetRowStrategy(INFA_CT_INPUTGROUP_HANDLE group, INFA_CTUpdateStrategy  
updateStrategy);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
组	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。
updateStrategy	INFA_CT_UPDATESTRATEGY	输入	输入端口的更新策略。集成服务使用以下值： - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

- **INFA\_CTSetRowStrategy()**。为每一行设置更新策略。这将替代 INFA\_CTChangeDefaultRowStrategy 函数。

请使用以下语法：

```
INFA_STATUS INFA_CTSetRowStrategy(INFA_CT_OUTPUTGROUP_HANDLE group, INFA_CT_UPDATESTRATEGY  
updateStrategy);
```



下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
组	INFA_CT_OUTPUTGROUP_HANDLE	输入	输出组句柄。
updateStrategy	INFA_CT_UPDATESTRATEGY	输入	要为输出端口设置的更新策略。 使用以下值之一： - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

### 更改默认行策略函数

默认情况下，当转换范围为“行”时，自定义转换的行策略为传递。当转换范围为“事务”或“所有输入”时，默认情况下行策略的值与“将源行视为”会话属性相同。

例如，在映射中有一个更新策略转换，之后是一个转换范围为“行”的自定义转换。更新策略转换可标记要更新、插入或删除的行。当集成服务将行传递到自定义转换时，自定义转换将保留该标记，因为行策略是传递。

不过，可以使用 PowerCenter 更改自定义转换的行策略。使用 INFA\_CTChangeDefaultRowStrategy() 函数可在转换级别更改默认行策略。例如，将自定义转换的默认行策略更改为插入时，集成服务将标记通过此转换传递进行插入的所有行。

**注意：**如果会话未处于数据驱动的模式之下，集成服务将返回 INFA\_FAILURE。

请使用以下语法：

```
INFA_STATUS INFA_CTChangeDefaultRowStrategy(INFA_CT_TRANSFORMATION_HANDLE transformation,
INFA_CT_DefaultUpdateStrategy defaultUpdateStrategy);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
转换	INFA_CT_TRANSFORMATION_HANDLE	输入	转换句柄。
defaultUpdateStrategy	INFA_CT_DefaultUpdateStrategy	输入	指定您希望集成服务用于自定义转换的行策略。 - eDUS_PASSTHROUGH。标记要传递的行。 - eDUS_INSERT。标记要插入的行。 - eDUS_UPDATE。标记要更新的行。 - eDUS_DELETE。标记要删除的行。

返回值的数据类型为 INFA\_STATUS。对于返回值使用 INFA\_SUCCESS 和 INFA\_FAILURE。

# 基于数组的 API 函数

基于数组的函数是指您在将数据访问模式更改为基于数组时使用的 API 函数。

Informatica 提供以下基于数组的 API 函数组：

- 最大行数
- 行数
- 行是否有效
- 数据处理（基于数组的模式）
- 行策略
- 设置输入错误行

## 最大行数函数

默认情况下，集成服务允许输入块和输出块中的最大行数。不过，可以更改输出块中允许的最大行数。

使用 `INFA_CTAGetInputNumRowsMax()` 和 `INFA_CTAGetOutputNumRowsMax()` 函数可确定输入和输出块中的最大行数。如果过程需要缓冲区，使用这些函数返回的值可确定缓冲区大小。

使用 `INFA_CTASetOutputRowMax()` 函数可设置输出块中的最大行数。如果希望过程使用更大或更小的缓冲区，可以使用此函数。

只能在初始化函数中调用这些函数。

PowerCenter 提供以下函数来确定和设置块中的行数：

- **`INFA_CTAGetInputNumRowsMax()`**.使用此函数可确定输入块中允许的最大行数。

请使用以下语法：

```
IINT32 INFA_CTAGetInputRowMax( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。

- **`INFA_CTAGetOutputNumRowsMax()`**.使用此函数可确定输出块中允许的最大行数。

请使用以下语法：

```
IINT32 INFA_CTAGetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup );
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	输入	输出组句柄。

- **`INFA_CTASetOutputRowMax()`**.使用此函数可设置输出块中允许的最大行数。

请使用以下语法：

```
INFA_STATUS INFA_CTASetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRowMax );
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	输入	输出组句柄。
nRowMax	INFA_INT32	输入	您希望输出块中允许的最大行数。 必须输入一个正数。使用非正数（包括零）时，该函数将返回致命错误。

行数函数

使用行数函数可针对指定的输入或输出组，确定输入块中的行数或设置输出块中的行数。

PowerCenter 提供以下行数函数：

- **INFA\_CTAGetNumRows()**.可以确定输入块中的行数。

请使用以下语法：

```
INFA_INT32 INFA_CTAGetNumRows( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。

- **INFA\_CTASetNumRows()**.可以设置输出块中的行数。在调用输出通知函数之前调用此函数。

请使用以下语法：

```
void INFA_CTASetNumRows( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRows );
```

下表介绍了此函数的参数：

参数	数据类型	输入/ 输出	说明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	输入	输出端口句柄。
nRows	INFA_INT32	输入	您希望在输出块中定义的行数。 必须输入一个正数。如果指定的是非正数，集成服务的输出通知函数将失败。

## 行是否有效函数

块中的某些行可能会被删除、筛选或为错误行。使用 INFA\_CTIsRowValid() 函数可确定块中的行是否有效。如果行有效，此函数将返回 INFA\_TRUE。

请使用以下语法：

```
INFA_BOOLEAN INFA_CTIsRowValid( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。

## 数据处理函数（基于数组的模式）

当集成服务调用 p\_<proc\_name>\_inputRowNotification() 函数时，它会通知过程其可访问数据行或数据块。但要在基于数组的模式下获取输入端口中的数据、进行修改及设置输出端口中的数据，则必须在输入行通知函数中使用基于数组的数据处理函数。

加入 INFA\_CTGetData<datatype>() 函数可获取输入端口中的数据，加入 INFA\_CTSetData() 函数可设置输出端口中的数据。如果希望过程在获取数据前确认端口是否拥有空值或空字符串，可加入 INFA\_CTGetIndicator() 函数。

PowerCenter 为基于数组的数据访问模式提供以下数据处理函数：

- INFA\_CTGetData<datatype>()
- INFA\_CTGetIndicator()
- INFA\_CTSetData()

### 获取数据函数（基于数组的模式）

使用 INFA\_CTGetData<datatype>() 函数可检索该函数指定的端口的数据。您必须根据希望过程访问的端口的数据类型修改函数名称。集成服务将在基于数组的获取数据函数中传递数据长度。

下表列出了 INFA\_CTGetData<datatype>() 函数语法和返回值的数据类型：

语法	返回值数据类型
<pre>void* INFA_CTGetDataVoid( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</pre>	指向返回值的数据空型指针
<pre>char* INFA_CTGetDataStringM( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</pre>	字符串 (MBCS)
<pre>IUNICHAR* INFA_CTGetDataStringU( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</pre>	字符串 (Unicode)

语法	返回值数据类型
<code>INFA_INT32 INFA_CTGetDataINT32( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	整型
<code>double INFA_CTGetDataDouble( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	双精度
<code>INFA_CT_RAWDATETIME INFA_CTGetDataRawDate( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	原始日期
<code>INFA_CT_DATETIME INFA_CTGetDataDateTime( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	日期时间
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB（精度 18）
<code>INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB（精度 28）

## 获取指示器函数（基于数组的模式）

当希望过程确认输入端口是否包含空值时，可使用获取指示器函数。

请使用以下语法：

```
INFA_INDICATOR INFA_CTGetIndicator( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow );
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputport	INFA_CT_INPUTPORT_HANDLE	输入	输入端口句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。

返回值的类型为 INFA\_INDICATOR。对于 INFA\_INDICATOR，可使用以下值：

- **INFA\_DATA\_VALID**。指示数据有效。
- **INFA\_NULL\_DATA**。指示空值。
- **INFA\_DATA\_TRUNCATED**。指示数据被截断。

## 设置数据函数（基于数组的模式）

如果希望过程将值传递到输出端口，请使用设置数据函数。可以为指定的输出端口设置数据、数据长度（如果适用），以及指示器。请勿使用单独的函数为输出端口设置长度或指示器。

请使用以下语法：

```
void INFA_CTSetData( INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_INT32 iRow, void* pData, INFA_UINT32 nLength, INFA_INDICATOR indicator);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
outputport	INFA_CT_OUTPUTPORT_HANDLE	输入	输出端口句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。 必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。
pData	空型*	输入	指向数据的指针。
nLength	INFA_UINT32	输入	端口的长度。仅适用于字符串和二进制端口。 必须验证函数传递的数据长度是否正确。如果函数传递了不同长度，则输出通知函数将为此端口返回失败。 验证并确保为字符串和二进制端口设置的长度不大于端口的精度。如果设置的长度高于该端口精度，则会获得异常结果。例如，会话可能失败。
指示器	INFA_INDICATOR	输入	输出端口的指示器值。使用以下值之一： - INFA_DATA_VALID。指示数据有效。 - INFA_NULL_DATA。指示空值。 - INFA_DATA_TRUNCATED。指示数据被截断。

## 行策略函数（基于数组的模式）

通过基于数组的行策略函数，您可以访问并配置块中每行的更新策略。

PowerCenter 提供以下行策略函数：

- **INFA\_CTAGetRowStrategy()**.允许过程获取块中行的更新策略。

请使用以下语法：

```
INFA_CT_UPDATESTRATEGY INFA_CTAGetRowStrategy( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。 必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。

- **INFA\_CTASetRowStrategy()**.为块中的行设置更新策略。

请使用以下语法：

```
void INFA_CTASetRowStrategy( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 iRow,
INFA_CT_UPDATESTRATEGY updateStrategy );
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	输入	输出组句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。 必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。
updateStrategy	INFA_CT_UPDATESTRATEGY	输入	端口的更新策略。使用以下值之一： - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

设置输入错误行函数

如果使用基于数组的访问模式，则将无法在输入行通知函数中返回 INFA\_ROWERROR。而是使用设置输入错误行函数通知集成服务，特定输入行存在错误。

PowerCenter 将提供以下处于基于数组的模式下的设置输入行函数：

- **INFA\_CTASetInputErrorRowM()**.可以通知集成服务，输入块中的某一行存在错误，并向会话日志输出一条 MBCS 错误消息。

请使用以下语法：

```
INFA_STATUS INFA_CTASetInputErrorRowM( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t
nErrors, INFA_MBCSCHAR* sErrMsg );
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputGroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。
iRow	INFA_INT32	输入	块中行的索引号。该索引基于零创建。 必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。

参数	数据类型	输入/输出	说明
nErrors	size_t	输入	可以使用此参数来指定此输入行发生的错误数量。
sErrMsg	INFA_MBCSCHAR*	输入	<p>MBCS 字符串，其中包含您希望该函数输出的错误消息。必须输入以空值终止的字符串。</p> <p>此参数是可选的。在包括此参数后，即使您启用了行错误日志记录，集成服务也会在会话日志中打印该消息。</p>

- **INFA\_CTASetInputErrorRowU()**.可以通知集成服务，输入块中的某一行存在错误，并向会话日志输出一条 Unicode 错误消息。

请使用以下语法：

```
INFA_STATUS INFA_CTASetInputErrorRowU( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_UNICHAR* sErrMsg );
```

下表介绍了此函数的参数：

参数	数据类型	输入/输出	说明
inputGroup	INFA_CT_INPUTGROUP_HANDLE	输入	输入组句柄。
iRow	INFA_INT32	输入	<p>块中行的索引号。该索引基于零创建。</p> <p>必须确保过程仅传递数据块中存在的索引号。如果传递无效的值，集成服务将会异常关闭。</p>
nErrors	size_t	输入	可以使用此参数来指定此输出行发生的错误数量。
sErrMsg	INFA_UNICHAR*	输入	<p>Unicode 字符串，其中包含您希望该函数输出的错误消息。必须输入以空值终止的字符串。</p> <p>此参数是可选的。在包括此参数后，即使您启用了行错误日志记录，集成服务也会在会话日志中打印该消息。</p>



## 第 5 章

# 数据屏蔽转换

本章包括以下主题：

- [数据屏蔽转换, 113](#)
- [屏蔽属性, 114](#)
- [键屏蔽, 116](#)
- [置换屏蔽, 117](#)
- [相关屏蔽, 120](#)
- [随机屏蔽, 122](#)
- [应用屏蔽规则, 123](#)
- [表达式屏蔽, 126](#)
- [特殊屏蔽格式, 128](#)
- [社会保障号屏蔽, 128](#)
- [信用卡号屏蔽, 129](#)
- [电话号码屏蔽, 130](#)
- [电子邮件地址屏蔽, 130](#)
- [社会保险号屏蔽, 131](#)
- [IP 地址屏蔽, 132](#)
- [URL 地址屏蔽, 132](#)
- [默认值文件, 132](#)
- [数据屏蔽转换会话属性, 133](#)
- [数据屏蔽转换的规则和准则, 134](#)

## 数据屏蔽转换

使用数据屏蔽转换将敏感的生产数据更改为非生产环境的现实测试数据。数据屏蔽转换将基于为每一列配置的屏蔽规则来修改源数据。

可针对软件开发、测试、培训和数据挖掘创建屏蔽的数据。可以维护屏蔽数据中的数据关系，并维护数据库表之间的引用完整性。数据屏蔽转换是被动转换。

数据屏蔽转换根据为端口配置的源数据类型和屏蔽类型提供屏蔽规则。对于字符串，可以限制字符串中要替换的字符以及要在屏蔽中应用的字符。对于数值和日期，可以为屏蔽的数据提供一个数字范围。可以根据原始数字的固定差额或百分比差额配置一个范围。集成服务会根据通过屏蔽规则配置的区域设置来替换字符。

可以对数据屏蔽转换应用以下类型的屏蔽：

- **键屏蔽。**为相同的源数据、屏蔽规则和种子值生成确定性的结果。
- **置换屏蔽。**将数据列替换为字典中相似却无关的数据。
- **相关屏蔽。**根据其他源列的值替换某个源列的值。
- **随机选择屏蔽。**将一系列数据的值替换为同一列中的其他值。
- **随机屏蔽。**为相同的源数据和屏蔽规则生成不可重复的随机结果。
- **表达式屏蔽。**将表达式应用到端口以更改数据或创建数据。
- **特殊屏蔽格式。**将特殊屏蔽格式应用到常见类型的敏感数据。可以屏蔽社会保障号、社会保险号、信用卡号、电话号码、URL 地址、电子邮件地址或 IP 地址。

要使用数据屏蔽转换，您需要相应的许可。

相关主题：

- [“端口的默认值” 页面上 36](#)
- [“应用屏蔽规则” 页面上 123](#)
- [“默认值文件” 页面上 132](#)

## 屏蔽属性

在“属性”选项卡上定义输入端口并为每个端口配置屏蔽属性。您可以选择的屏蔽类型基于端口数据类型。选择屏蔽类型时，Designer 会显示屏蔽类型的屏蔽规则。

## 区域设置

区域设置可确定数据中字符的语言和地区。从列表中选择区域设置。数据屏蔽转换可通过所选区域设置中的字符屏蔽字符数据。源数据必须包含与所选区域设置兼容的字符。

如果区域设置不在列表中，请选择具有相似或匹配的代码页的区域设置。不能为区域设置选择 Unicode。

## 屏蔽类型

屏蔽类型是应用到所选列的数据屏蔽类型。选择以下屏蔽类型之一：

- **键屏蔽。**为相同的源数据、屏蔽规则和种子值生成确定性的结果。
- **置换屏蔽。**将数据列替换为字典中相似却无关的数据。
- **相关屏蔽。**根据其他源列的值替换某个源列的值。
- **随机屏蔽。**为相同的源数据和屏蔽规则生成随机结果。
- **表达式屏蔽。**将表达式应用到端口以更改数据或创建数据。
- **特殊屏蔽格式。**将特殊屏蔽格式应用到常见类型的敏感数据。可以屏蔽社会保障号、社会保险号、信用卡号、电话号码、URL 地址、电子邮件地址或 IP 地址。
- **置换。**将数据列替换为字典中相似却无关的数据。
- **无屏蔽。**数据屏蔽转换不会更改源数据。

默认设置为“无屏蔽”。

## 可重复的输出

可重复的输出是数据屏蔽转换返回的一组连续的值。

可重复输出会返回确定性的值。例如，为一个名字列配置可重复输出。每次在工作流中包含数据屏蔽转换时，该转换都会返回相同的屏蔽值。

可以为所有数据屏蔽类型配置可重复屏蔽。要配置可重复屏蔽，请单击**可重复的输出**并选择**种子值**。

## 种子

种子值是生成屏蔽值的起点。

数据屏蔽转换将创建一个默认种子值，它是一个介于 1 与 1,000 之间的随机数。可以输入不同的种子值，或者应用映射参数值。将同一种子值应用于某一列，可在不同源数据中返回相同的屏蔽数据值。例如，如果您的四个表中都有相同的 Cust\_ID 列，并且您想要所有这些列都输出相同的屏蔽值，请将全部四列设置为同一种子值。

## 映射参数

可以使用映射参数定义种子值。为您要添加到转换的每个种子值创建映射参数。映射参数值是介于 1 和 1000 之间的数字。

为某个列配置数据屏蔽时，为种子选择“映射参数”。Designer 会显示一个映射参数列表。从列表中选择一个映射参数。在运行会话之前，可以在会话的参数文件中更改映射参数值。

在创建数据屏蔽转换之前创建映射参数。如果选择参数化种子值且映射没有映射参数，则会显示错误。如果选择的端口具有引用已删除映射参数的屏蔽规则，则 Designer 会为该端口生成新的随机种子值。种子值不是映射参数。系统会显示消息指明映射参数已删除，并且 Designer 会创建新的种子值。

集成服务会在以下情况下应用默认种子值：

- 为列选择了映射参数选项，但是会话没有参数文件。
- 删除映射参数。
- 映射参数种子值并非介于 1 和 1,000 之间。

集成服务将应用默认值文件中的屏蔽值。可以编辑默认值文件以更改默认值。

默认值文件是位于以下位置的 XML 文件：

```
<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

种子的名称值对为

```
default_seed = "500".
```

如果默认值文件中的种子值并非介于 1 和 1,000 之间，则集成服务会为种子分配值 725，并在会话日志中写入一条消息。

### 相关主题：

- [“默认值文件” 页面上 132](#)

## 关联 O/P

关联的 O/P 是输入端口的关联输出端口。数据屏蔽转换会为每个输入端口创建输出端口。命名约定为 out\_<port name>。关联的输出端口为只读端口。

# 键屏蔽

每当源值和种子值相同时，配置了键屏蔽的列将返回确定性的屏蔽数据。数据屏蔽转换将为该列返回唯一值。

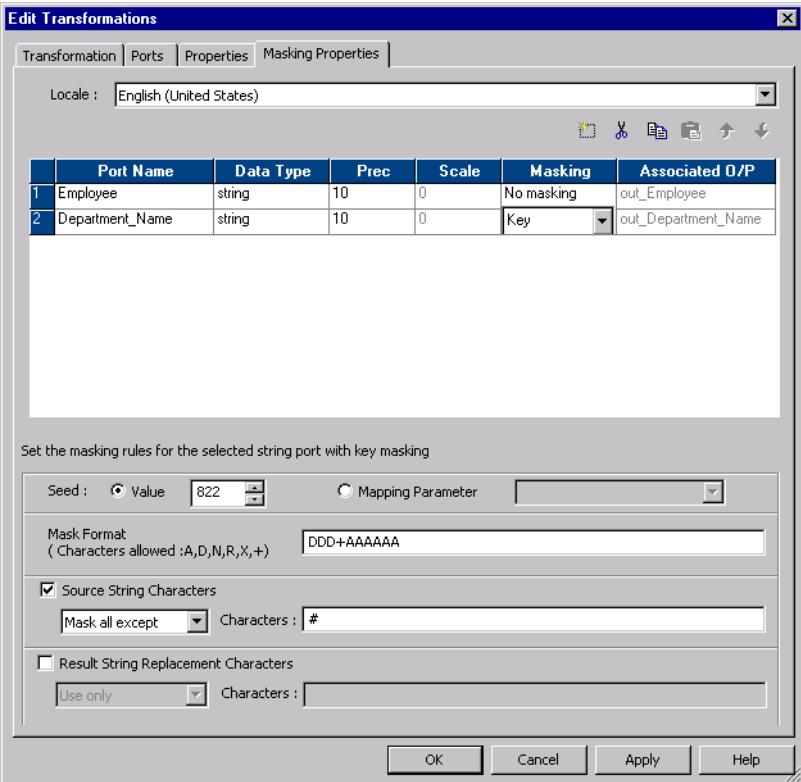
为列配置键屏蔽时，数据屏蔽转换将为该列创建种子值。可以更改种子值，以便在不同的数据屏蔽转换之间生成可重复的数据。例如，配置键屏蔽以强制执行引用完整性。使用相同的种子值屏蔽表中的主键和另一表中的外键值。

可以定义屏蔽规则以影响数据屏蔽转换所返回数据的格式。使用键屏蔽来屏蔽字符串值和数值。

## 屏蔽字符串值

可以为字符串配置键屏蔽，以便生成可重复输出。配置屏蔽格式，以便定义对输出字符串中每个字符的限制和定义屏蔽格式。配置源字符串字符和结果字符串替换字符，以用来定义要屏蔽的源字符和屏蔽它们所使用的字符。

下图显示了字符串数据类型的键屏蔽属性：



- 可以为键屏蔽字符串值配置以下屏蔽规则：
- **种子。**应用种子值，以便为列生成确定性屏蔽数据。选择以下选项之一：
    - **值。**接受默认种子值或输入一个 1 到 1,000 之间的数字。
    - **映射参数。**使用映射参数可定义种子值。Designer 将显示您为映射创建的映射参数列表。从该列表中选择映射参数以用作种子值。
  - **屏蔽格式。**定义置换输入数据中的每个字符的字符类型。可以将每个字符限制为字母、数字或字母数字字符类型。

- **源字符串字符。**定义源字符串中要屏蔽的字符。例如，每当输入数据中出现数字符号 (#) 字符时，屏蔽该字符。当“源字符串字符”为空时，数据屏蔽转换将屏蔽所有输入字符。如果源字符串的字符数小于结果字符串的字符数，则数据屏蔽转换不会始终返回唯一数据。
- **结果字符串字符。**用“结果字符串字符”中定义的字符替换目标字符串中的字符。例如，输入以下字符可将每个屏蔽配置为全部包含大写字母字符：

ABCDEFGHIJKLMNOPQRSTUVWXYZ

## 屏蔽数值

为数值源数据配置键屏蔽以生成确定性的输出。为列配置数值键屏蔽时，需要为该列分配随机种子值。数据屏蔽转换屏蔽源数据时，将应用需要使用种子的屏蔽算法。

如果不同的列中出现相同的源值，可以更改列的种子值以生成可重复的结果。例如，要维护两个表之间的主键-外键关系，请在每个数据屏蔽转换中，为主键列输入与外键列相同的种子值。数据屏蔽转换将为相同的数值生成确定性的结果。表之间的引用完整性得到维护。

## 屏蔽日期时间值

如果可以为日期时间值配置键屏蔽，数据屏蔽转换将需要使用随机数作为种子。可以更改种子以匹配其他列的种子值，以便返回列之间可重复的日期时间值。

数据屏蔽转换可以使用键屏蔽方法屏蔽介于 1753 和 2400 之间的日期。如果源年份是闰年，则数据屏蔽转换返回的年份也是闰年。如果源月份包含 31 天，则数据屏蔽转换返回的月份也是 31 天。如果源月份是二月，则数据屏蔽转换将返回二月。

数据屏蔽转换将始终生成有效日期。

### 相关主题：

- [“日期范围”页面上 125](#)

## 置换屏蔽

置换屏蔽将一系列数据替换为相似但无关的数据。使用置换屏蔽将生产数据替换为真实的测试数据。配置置换屏蔽时，请定义包含置换值的字典。

数据屏蔽转换会对您配置的字典执行查找。数据屏蔽转换将源数据替换为字典中的数据。字典文件可以包含字符串数据、日期时间值、整数以及浮点数。按以下格式输入日期时间值：

mm/dd/yyyy

可将数据置换为可重复或不可重复值。选择可重复值时，数据屏蔽转换将为相同的源数据和种子值生成确定性的结果。必须配置种子值才能将数据置换为确定性的结果。集成服务为可重复屏蔽维护一份源值和已屏蔽值的存储表。

可将多列数据置换为同一字典行中的已屏蔽值。为一个输入列配置置换屏蔽。为从同一字典行接收已屏蔽数据的其他列配置相关数据屏蔽。

## 字典

字典是指包含文件中每行的置换数据的平面文件或关系表。数据屏蔽转换将生成一个数字以检索字典行。数据屏蔽转换会生成一个哈希键以进行可重复的置换屏蔽，或者生成一个随机数字以进行不可重复的屏蔽。可以配置附加查找条件。

可以配置一个字典以屏蔽数据屏蔽转换中的多个端口。

以下示例显示了一个平面文件字典，其中包含名字和性别：

```
SNO,GENDER,FIRSTNAME
1,M,Adam
2,M,Adeel
3,M,Adil
4,F,Alice
5,F,Alison
```

在此字典中，行中的第一个字段是序列号，第二个字段是性别。您可以将性别添加为查找条件。集成服务会使用哈希键从字典中检索行，并查找性别与源数据中的性别匹配的行。

创建字典时，请遵循以下规则和准则：

- 平面文件字典的第一行必须具有列标签，才能标识每个记录中的字段。这些字段以逗号分隔。如果第一行不包含列标签，则集成服务会使用第一行中的字段值作为列名称。
- 平面文件字典必须位于 \$PMLookupFileDir 查找文件目录中。默认情况下，此目录位于以下位置：  
<PowerCenter\_Installation\_Directory>\server\infa\_shared\LkpFiles
- 如果在 Windows 上创建了一个平面文件字典，并将其复制到 UNIX 计算机中，请验证该文件格式是否适用于 UNIX。例如，Windows 和 UNIX 使用不同的字符作为行尾标记。
- 如果为多个端口配置了置换屏蔽，则所有关系字典必须位于同一个数据库架构中。
- 平面文件字典的换行缓冲区长度必须少于或等于 600 个字符。
- 不能更改会话属性中的字典类型或置换字典名称。

## 存储表

数据屏蔽转换为会话间的可重置置换维护存储表。存储表行包含源列和屏蔽的值对。当数据屏蔽转换每次使用可重置置换屏蔽某个值时，将根据字典名称、区域设置、列名称、输入值和种子搜索存储表。如果找到一个行，将从存储表返回该屏蔽值。如果数据屏蔽转换未找到行，将从字典接收一个行以及一个哈希键。

存储表中的字典名称格式对于平面文件字典和关系字典是不同的。平面文件字典名称通过文件名进行标识。关系字典名称使用以下语法：

```
<Connection object>_<dictionary table name>
```

Informatica 提供可运行以创建关系存储表的脚本。这些脚本位于以下位置：

```
<PowerCenter Client installation directory>\client\bin\Extensions\DataMasking
```

该目录包含 Sybase、Microsoft SQL Server、IBM DB2 和 Oracle 数据库的脚本。每个脚本都命名为 Substitution\_<数据库类型>。如果您配置 SQL 语句和主键约束，就能在一个不同的数据库中创建表。

当存储中具有未加密的数据并使用了相同的种子值和字典来加密相同的列时，需要为置换屏蔽加密存储表。

### 加密用于置换屏蔽的存储表

可以使用转换语言编码函数加密存储表。如果已启用存储加密，则需要加密存储表。

- 将 IDM\_SUBSTITUTION\_STORAGE 存储表作为源来创建一个映射。
- 创建数据屏蔽转换。
- 对输入值端口和屏蔽值端口应用置换屏蔽技术。

4. 对 INPUTVALUE 端口使用以下表达式：  
`Enc_Base64(AES_Encrypt(INPUTVALUE, Key))`
5. 对 MASKEDVALUE 端口使用以下表达式：  
`Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))`
6. 将这些端口链接到目标。

## 置换屏蔽属性

可以为置换屏蔽配置以下屏蔽规则：

- **可重复的输出。**返回会话之间的确定性结果。数据屏蔽会转换将屏蔽值存储在存储表中。
- **种子。**应用种子值，以便为列生成确定性屏蔽数据。选择以下选项之一：
  - **值。**接受默认种子值或输入一个 1 到 1,000 之间的数字。
  - **映射参数。**使用映射参数可定义种子值。Designer 将显示您为映射创建的映射参数列表。从该列表中选择映射参数以用作种子值。
  - **唯一输出。**强制数据屏蔽转换为唯一的输入值创建唯一的输出值。两个不同的输入值在屏蔽后不会产生相同的输出值。字典必须具有足够的唯一行来支持唯一输出。  
如果禁用唯一输出，则数据屏蔽转换可能不会为输入值生成唯一的屏蔽输出值。字典包含的行可能会较少。
- **字典信息。**配置包含替代数据值的平面文件或关系表。
  - **关系表。**如果字典在数据库表中，则选择关系表。单击“选择表”以配置数据库和表。
  - **平面文件。**如果字典位于以逗号分隔的平面文件中，则选择“平面文件”。单击“选择平面文件”以浏览并选择文件。
  - **字典名称。**显示选定的平面文件或关系表名称。
  - **输出列。**选择要返回到数据屏蔽转换的列。
- **查找条件。**配置查找条件以进一步限定要用于置换屏蔽的字典行。查找条件类似于 SQL 查询中的 WHERE 子句。配置查找条件时，可以将源中某个列的值与字典中某个列的值进行比较。  
例如，您要屏蔽名字。源数据和字典中都有名字列和性别列。您可以添加一个条件，使每个女性名字都用字典中的一个女性名字来替换。查找条件会将源中的性别与字典中的性别进行比较。
  - **输入端口。**要在查找中使用的源数据列。
  - **字典列。**要与输入端口进行比较的字典列。

## 关系字典

当您选择关系表作为字典时，请配置包含该字典的数据库表。定义此表时，Designer 会连接至表并显示表中的列。

要选择数据库，在数据屏蔽转换的“屏蔽属性”选项卡上单击“选择表”。

在“选择表”对话框中配置以下字段：

- **ODBC 数据源。**连接至包含字典的数据库的 ODBC 数据源。
- **用户名。**用于连接至数据库的数据库用户名。该用户名具有查看表格的数据库权限。
- **所有者名称。**如果用户名不适字典表的所有者，输入表所有者。
- **密码。**数据库用户名的密码。
- **搜索指定的表。**可选。限制显示在对话框中的表数。

当您单击“连接”时，Designer 会连接至数据源并显示表。滚动表列表，并选择要用于字典的表。该表的名称此时将显示在“屏蔽属性”选项卡的“字典名称”字段中。

## 连接要求

您必须为存储表和关系字典配置数据库连接，才能对它们使用置换屏蔽。

在 Workflow Manager 中配置数据库连接。在会话属性的“映射”选项卡上，选择 IDM\_Dictionary 连接和 IDM\_Storage 连接的关系连接对象。

## 置换屏蔽的规则和准则

对于置换屏蔽，请使用以下规则和准则：

- 如果唯一可重复置换屏蔽的存储表不存在，会话将失败。
- 如果字典不包含行，数据屏蔽转换将返回错误消息。
- 数据屏蔽转换找到存储表中包含区域设置、字典和种子的输入值时，它将检索屏蔽值，即使行不再位于字典中。
- 如果删除连接对象或修改字典，请截断存储表。否则，可能会产生意外结果。
- 如果字典中的值数量小于源数据中的唯一值数量，数据屏蔽转换将无法使用唯一可重复值来屏蔽数据。数据屏蔽转换将返回错误消息。

## 相关屏蔽

相关屏蔽会将源数据的多个列置换为相同字典行中的数据。

数据屏蔽转换对多个列执行置换屏蔽时，屏蔽的数据可能包含不切实际的字段组合。可以配置相关屏蔽，以便置换同一字典行中多个输入列的数据。屏蔽的数据将接收到有效组合，例如“New York, New York”或“Chicago, Illinois”。

配置相关屏蔽时，请先配置进行置换屏蔽的输入列。配置要与该置换列相关的其他输入列。例如，选择邮政编码列进行置换屏蔽，然后选择要与该邮政编码列相关的城市列和省/自治区/直辖市列。相关屏蔽可确保所置换的城市和省/自治区/直辖市对于所置换的邮政编码值有效。

**注意：**如果没有先配置进行置换屏蔽的列，将无法配置相关屏蔽列。

配置相关屏蔽列时，请配置以下屏蔽规则：

相关列

要配置为置换屏蔽的输入列的名称。数据屏蔽转换将使用适用于该列的屏蔽规则从字典中检索置换数据。配置为置换屏蔽的列将成为从字典中检索屏蔽数据的键列。

输出列

字典列的名称，其中包含配置为相关屏蔽的列的值。

## 相关屏蔽示例

数据屏蔽字典中可能包含具有以下值的地址行：

```
SNO, STREET, CITY, STATE, ZIP, COUNTRY
1,32 Apple Lane, Chicago, IL, 61523, US
2,776 Ash Street, Dallas, TX, 75240, US
3,2229 Big Square, Atleeville, TN, 38057, US
4,6698 Cowboy Street, Houston, TX, 77001, US
```

需要从“地址”字典中屏蔽包含城市、省/自治区/直辖市和邮政编码的有效组合的源数据。



为 ZIP 端口配置置换屏蔽。

为 ZIP 端口输入以下屏蔽规则：

规则	值
字典文件类型	平面文件或关系表
字典名称	Address.dic
输出列	邮政编码

为“城市”端口配置相关屏蔽。

为“城市”端口输入以下屏蔽规则：

规则	值
相关列	邮政编码
输出列	城市

为“省/自治区/直辖市”端口配置相关屏蔽。

为“省/自治区/直辖市”端口输入以下屏蔽规则：

规则	值
相关列	邮政编码
输出列	省/自治区/直辖市

当数据屏蔽转换屏蔽邮政编码时，将为字典行中的邮政编码返回正确的城市和省/自治区/直辖市。

## 可重复的相关屏蔽

配置多个输入字段以从同一字典行接收屏蔽数据时，需要确定哪一字段配置用于置换屏蔽。通过可重复的屏蔽，选择唯一标识源数据的列。数据屏蔽转换使用键字段来确定用于可重复屏蔽的存储中是否已存在行。

**注意：**源数据具有主键时，可以为置换屏蔽配置主键列。为相关屏蔽配置另一个列。

例如，如果屏蔽员工数据，则可为置换屏蔽配置 EmployeeID，并为相关屏蔽配置 FirstName 和 LastName。

配置可重复的屏蔽时，以下行会收到不同的屏蔽数据：

EmployeeID	FirstName	LastName
111	John	Jones
222	Radhika	Jones

如果选择 LastName 作为可重复相关屏蔽的键字段，则姓氏为 Jones 的每个行都会收到相同的屏蔽数据。

# 随机屏蔽

随机屏蔽将生成不确定性的随机屏蔽数据。当不同的行中出现相同的源值时，数据屏蔽转换将返回不同的值。可以定义屏蔽规则以影响数据屏蔽转换所返回数据的格式。使用随机屏蔽方法屏蔽数值、字符串值和日期值。

## 屏蔽数值

屏蔽数值数据时，可以配置列的输出值范围。数据屏蔽转换将根据端口精度返回一个介于范围上限和下限之间的值。要定义范围，请基于原始源值的差额配置最小和最大范围值或配置模糊范围。

可以为数值数据配置以下屏蔽参数：

### 范围

定义输出值的范围。数据屏蔽转换将返回介于最小值和最大值之间的数值数据。

### 模糊范围

定义一个落在源数据的固定差额或百分比差额内的输出值范围。数据屏蔽转换将返回接近源数据值的数值数据。可以配置范围和模糊范围。

## 屏蔽字符串值

配置随机屏蔽可为字符串列生成随机输出。要配置对输出字符串中每个字符的限制，请配置屏蔽格式。配置筛选字符以定义要屏蔽的源字符和屏蔽所用的字符。

可以为字符串端口应用以下屏蔽规则：

### 范围

配置字符串的最小长度和最大长度。数据屏蔽转换将返回一个介于最小字符串长度和最大字符串长度之间的随机字符串。

### 屏蔽格式

定义用于置换输入数据中每个字符的字符类型。可以将每个字符限制为字母、数字或字母数字字符类型。

### 源字符串字符

定义源字符串中要屏蔽的字符。例如，每当输入数据中出现数字符号 (#) 字符时，屏蔽该字符。当“源字符串字符”为空时，数据屏蔽转换将屏蔽所有输入字符。

### 结果字符串替换字符

用“结果字符串字符”中定义的字符置换目标字符串中的字符。例如，输入以下字符可将每个屏蔽配置为包含大写字母字符 A - Z：

ABCDEFGHIJKLMNOPQRSTUVWXYZ

## 屏蔽日期值

要使用随机屏蔽来屏蔽日期值，请配置输出日期的范围或选择一个差额。配置差额时，请选择要模糊的日期部分。选择年、月、日、小时、分钟或秒。数据屏蔽转换将返回配置范围内的一个日期。

屏蔽日期时间值时，可以配置以下屏蔽规则：

### 范围

为选定的日期时间值设置要返回的最小值和最大值。

模糊

基于要应用于日期单位的差额来屏蔽日期。数据屏蔽转换将返回差额范围内的一个日期。可以模糊年、月、日、小时、分钟或秒。选择要应用的低差额和高差额。

应用屏蔽规则

根据源数据类型应用屏蔽规则。在“屏蔽属性”选项卡上单击列属性时，Designer 会根据端口的数据类型显示屏蔽规则。

下表介绍了可以根据屏蔽类型和源数据类型配置的屏蔽规则：

屏蔽类型	源数据类型	屏蔽规则	说明
随机和键	字符串	屏蔽格式	屏蔽时将输出字符串中的每个字符限制为字母、数字或字母数字字符。
随机和键	字符串	源字符串字符	要屏蔽的源字符集或要从屏蔽中排除的源字符集。
随机和键	字符串	结果字符串替换字符	要在屏蔽中包含或排除的字符集。
随机	数值 字符串 日期/时间	范围	输出值的范围。 <ul style="list-style-type: none"><li>- 数值。数据屏蔽转换将返回介于最小值和最大值之间的数值数据。</li><li>- 字符串。返回介于最小字符串长度和最大字符串长度之间的随机字符串。</li><li>- 日期/时间。返回最小和最大日期时间范围内的日期和时间。</li></ul>
随机	数值 日期/时间	模糊	使用源数据固定差额或百分比差额的输出值范围。数据屏蔽转换将返回接近源数据值的数据。日期时间列要求使用固定差额。

屏蔽格式

配置屏蔽格式，将输出列中的每个字符限制为字母、数字或字母数字字符。使用以下字符定义屏蔽格式：

A, D, N, X, +, R

**注意:** 屏蔽格式包含大写字符。输入小写屏蔽字符时，数据屏蔽转换会将该字符转换为大写。

下表介绍了屏蔽格式字符：

Character	说明
A	字母字符。例如，ASCII 字符 a 到 z 以及 A 到 Z。
D	数字 0 到 9。对于数字 0 到 9 之外的其他字符，数据屏蔽转换将返回 “X”。
N	字母数字字符。例如，ASCII 字符 a 到 z、A 到 Z 以及 0-9。
X	任何字符。例如，字母数字或符号。

Character	说明
+	无屏蔽。
R	剩余字符。R 指定字符串中的剩余字符可以是任何字符类型。R 必须显示为屏蔽中的最后一个字符。

例如，某部门名称使用以下格式：

```
nnn-<department_name>
```

可以配置屏蔽，强制前三个字符为数字、部门名称为字母，并且在输出中保留短划线。配置以下屏蔽格式：

```
DDD+AAAAAAAAAAAAAAA
```

数据屏蔽转换会将前三个字符替换为数字字符。不会替换第四个字符。数据屏蔽转换会将剩余字符替换为字母字符。

如果未定义屏蔽格式，则数据屏蔽转换会将每个源字符替换为任意字符。如果屏蔽格式比输入字符串长，则数据屏蔽转换将忽略屏蔽格式中的多余字符。如果屏蔽格式比源字符串短，则数据屏蔽转换将屏蔽格式为 R 的剩余字符。

**注意：**无法使用范围选项配置屏蔽格式。

# 源字符串字符

源字符串字符是选择要对其进行屏蔽或不对其进行屏蔽的源字符。源字符串中字符的位置不会造成影响。源字符区分大小写。

可以配置任意数量的字符。当字符为空时，数据屏蔽转换将替换列中所有的源字符。

为源字符串字符选择以下选项之一：

## 仅屏蔽

数据屏蔽转换对源中配置为源字符串字符的字符进行屏蔽。例如，如果输入字符 A、B 和 c，当该字符出现在源数据中时，数据屏蔽转换会将 A、B 或 c 替换为不同的字符。不是 A、B 或 c 的源字符将不会发生更改。屏蔽区分大小写。

## 除了以下内容，全部屏蔽

除了出现在源字符串中的源字符串字符，屏蔽所有字符。例如，如果输入筛选器源字符“-”并选择“除了以下内容，全部屏蔽”，当该字符出现在源数据中时，数据屏蔽转换不会替换“-”字符。其余源字符将发生更改。

# 源字符串示例

源文件包含一个名为“相关项”的列。“相关项”列包含多个用逗号分隔的名称。需要屏蔽“相关项”列，并在测试数据中使用逗号以分隔名称。

对于“相关项”列，请选择“源字符串字符”。选择“不屏蔽”，然后输入“,”作为要跳过的源字符。请勿输入引号。

数据屏蔽转换会替换掉源字符串中除逗号之外的所有字符。

## 结果字符串替换字符

结果字符串替换字符是您选择用于作为屏蔽数据中置换字符的字符。配置结果字符串替换字符时，数据屏蔽转换将使用结果字符串替换字符替换源字符串中的字符。为了避免为不同的输入值生成相同的输出，请配置多种置换字符，或者只屏蔽一些源字符。字符串中每个字符的位置不会造成影响。

为结果字符串替换字符选择以下选项之一：

仅使用

仅使用您定义为结果字符串替换字符的字符屏蔽源。例如，如果输入字符 A、B 和 c，数据屏蔽转换将使用 A、B 和 c 替换源列中的每个字符。单词“horse”将替换为“BAcBA”。

除了以下内容，全部使用

使用除您定义为结果字符串替换字符的字符外的任意字符屏蔽源。例如，如果您输入 A、B 和 c 结果字符串替换字符，屏蔽数据将永远不会出现字符 A、B 或 c。

## 结果字符串替换字符示例

要将相关列中的所有逗号替换为分号，请完成以下任务：

1. 将逗号配置为源字符串字符，然后选择“仅屏蔽”。  
如果相关列中出现逗号，则数据屏蔽转换将仅屏蔽逗号。
2. 将分号配置为结果字符串替换字符，然后选择“仅使用”。  
数据屏蔽转换会将相关列中的每个逗号替换为分号。

## 范围

为数字、日期或字符串数据定义范围。如果为数值或日期值定义了范围，则数据屏蔽转换将使用一个介于最小值和最大值之间的值来屏蔽源数据。为字符串配置范围时，请配置字符串长度范围。

### 字符串范围

配置随机字符串屏蔽时，数据屏蔽转换将生成与源字符串长度不同的字符串。或者，可以配置字符串宽度的下限和上限。为宽度上限或下限输入的值必须是正整数。每个宽度都必须小于或等于端口精度。

### 数值范围

为数值列设置最小值和最大值。最大值必须小于或等于端口精度。默认范围介于一到端口精度长度之间。

### 日期范围

为日期时间值设置最小值和最大值。最小值字段和最大值字段包含默认的最小日期和最大日期。默认的日期时间格式为 MM/DD/YYYY HH24:MI:SS。最大的日期时间必须晚于最小的日期时间。

## 模糊

模糊将创建在源数据值的固定差额或百分比差额范围内的输出值。配置模糊可返回接近原始值的随机值。可以模糊数值和日期值。

### 模糊数值

选择固定差额或百分比差额以模糊数值源值。低模糊值是小于源值的差额。高模糊值是大于源值的差额。低值和高值都必须大于或等于零。数据屏蔽转换返回屏蔽的数据时，数值数据将落在您定义的范围内。

下表介绍了当输入源值为 66 时模糊范围值的屏蔽结果：

模糊类型	低	高	结果
固定	0	10	介于 66 与 76 之间
固定	10	0	介于 56 与 66 之间
固定	10	10	介于 56 与 76 之间
百分比	0	50	介于 66 与 99 之间
百分比	50	0	介于 33 与 66 之间
百分比	50	50	介于 33 与 99 之间

### 模糊日期值

通过配置模糊，将日期屏蔽为与源日期不同的日期。选择要应用差额的日期单位。可以选择年、月、日或小时。输入上限和下限以定义高于或低于源日期中的单位的差额。数据屏蔽转换将应用该差额并返回在该差额内的日期。

例如，要将屏蔽的日期限制为源日期两年内的日期，请选择年作为单位。输入 2 作为上限和下限。如果源日期是 02/02/2006，则数据屏蔽转换将返回一个介于 02/02/2004 和 02/02/2008 之间的日期。

默认情况下，模糊单位是年。

## 表达式屏蔽

表达式屏蔽会对端口应用表达式以更改数据或创建新数据。在配置表达式屏蔽时，请在“表达式编辑器”中创建一个表达式。选择输入和输出端口、函数、变量和运算符以构建表达式。

可以连接多个端口中的数据以便为其他端口创建值。例如，需要创建登录名。源具有名字列和姓氏列。屏蔽查找文件中的名字和姓氏。在数据屏蔽转换中，创建另一个名为“Login”的端口。对于 Login 端口，配置一个表达式以连接名字的首字母和姓氏：

`SUBSTR(FIRSTNM,1,1)||LASTNM`

为端口配置表达式屏蔽时，默认情况下端口名称会显示为表达式。要访问表达式编辑器，请单击“打开”按钮。表达式编辑器会显示没有为表达式屏蔽配置的输入端口和输出端口。无法将表达式的输出用作其他表达式的输入。如果将输出端口名称手动添加到表达式中，可能会获得意外的结果。

从点击式界面选择函数、端口、变量和运算符，以便在构建表达式时最大程度地减少错误。

创建表达式时，请验证表达式是否返回与端口数据类型匹配的值。如果表达式端口的数据类型是数值而表达式为其他数据类型，则数据屏蔽转换将返回零。如果表达式端口的数据类型是 String 而表达式为其他数据类型，则数据屏蔽转换将返回空值。

## 可重复表达式屏蔽

源列出现在多个表中，且需要使用相同值屏蔽每个表中的该列时，请配置可重复表达式屏蔽。

配置可重复表达式屏蔽时，数据屏蔽转换会将表达式结果保存到存储表中。如果该列出现在其他源表中，数据屏蔽转换将返回存储表中的屏蔽值，而不是表达式中的屏蔽值。

### 字典名称

配置可重复的表达式屏蔽时，必须输入字典名称。字典名称是允许多个数据屏蔽转换从相同的源值生成相同屏蔽值的键。请在每个数据屏蔽转换中定义相同的字典名称。字典名称可以是任意文本。

### 存储表

存储表包含会话之间可重复表达式屏蔽的结果。存储表行包含源列和屏蔽的值对。表达式屏蔽的存储表是来自置换屏蔽存储表的单独表。

当数据屏蔽转换每次使用可重复表达式屏蔽某个值时，将根据字典名称、区域设置、列名称和输入值搜索存储表。如果在存储表中找到了行，将从存储表中返回屏蔽的值。如果数据屏蔽转换未找到行，将根据列的表达式生成屏蔽值。

当存储中包含未加密数据并使用相同字典名称作为密钥时，需要加密表达式屏蔽的存储表。

### 加密存储表以用于表达式屏蔽

可以使用转换语言编码函数加密存储表。如果已启用存储加密，则需要加密存储表。

1. 将 IDM\_EXPRESSION\_STORAGE 存储表作为源来创建一个映射。
2. 创建数据屏蔽转换。
3. 在屏蔽值端口上应用表达式屏蔽技术。
4. 对 MASKEDVALUE 端口使用以下表达式：  
`Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))`
5. 将这些端口链接到目标。

### 示例

例如，Employees 表中包含以下列：

```
FirstName
LastName
LoginID
```

在数据屏蔽转换中，使用组合了 FirstName 和 LastName 的表达式屏蔽 LoginID。配置可重复的表达式屏蔽。输入一个字典名称，作为可重复屏蔽的键。

Computer\_Users 表包含一个 LoginID 列，但不包含 FirstName 或 LastName 列：

```
Dept
LoginID
Password
```

要屏蔽 Computer\_Users 中与 Employees 中相同的 LoginID，请为 LoginID 列配置表达式屏蔽。启用可重复屏蔽并输入为 Employees 表中 LoginID 所定义的字典名称。集成服务将从存储表中检索 LoginID 值。

请创建一个当集成服务在存储表中找不到 LoginID 行时要使用的默认表达式。Computer\_Users 表没有 FirstName 或 LastName 列，因此表达式将创建一个意义较小的 LoginID 列。

## 存储表脚本

Informatica 提供可运行以创建存储表的脚本。这些脚本位于以下位置：

```
<PowerCenter installation directory>\client\bin\Extensions\DataMasking
```

该目录包含 Sybase、Microsoft SQL Server、IBM DB2 和 Oracle 数据库的脚本。每个脚本都命名为 Expression\_<数据库类型>。

## 表达式屏蔽的规则和准则

对于表达式屏蔽，请使用以下规则和准则：

- 无法将表达式的输出用作其他表达式的输入。如果将输出端口名称手动添加到表达式中，可能会获得意外的结果。
- 使用点击方法来构建表达式。从点击式界面选择函数、端口、变量和运算符，以便在构建表达式时最大程度地减少错误。
- 如果已为可重复的屏蔽配置数据屏蔽转换，并且不存在存储表，则集成服务会将源数据替换为默认值。

相关主题：

- [“使用表达式编辑器” 页面上 31](#)

## 特殊屏蔽格式

以下掩码类型保留原始数据的格式：

- 社会保障号
- 信用卡号
- 电话号码
- URL 地址
- 电子邮件地址
- IP 地址
- 社会保险号

数据屏蔽转换使用真实格式的屏蔽值替换敏感数据。例如，屏蔽 SSN 时，数据屏蔽转换将返回格式正确但无效的 SSN。

当源数据格式或数据类型对于某个屏蔽无效时，集成服务将对数据应用默认屏蔽。集成服务将应用默认值文件中的屏蔽值。可以编辑默认值文件以更改默认值。

相关主题：

- [“默认值文件” 页面上 132](#)

## 社会保障号屏蔽



数据屏蔽转换将基于社会保障局的最新高层组织列表生成无效的社会保障号。高层组织列表包含社会保障局已签发的有效编号。数据屏蔽转换从以下位置访问最新的高层组织列表：

<Installation Directory>\infa\_shared\SrcFiles\highgroup.txt

数据屏蔽转换生成高层组织列表中不存在的 SSN 编号。社会保障局每个月更新一次高层组织列表。从以下位置下载最新版本的列表：

<http://www.socialsecurity.gov/employer/ssns/highgroup.txt>

## 社会保障号格式

数据屏蔽转换接受任何包含九位数字的 SSN 格式。可使用任意字符集分隔这些数字。例如，数据屏蔽转换接受以下格式： +=54-\*9944\$#789-,\*()”。

## 区号要求

数据屏蔽转换会返回一个社会保障号，该号码因与源格式相同而无效。SSN 的前三位数字用于定义区号。数据屏蔽转换不会屏蔽区号。它会屏蔽组号和序列号。源 SSN 必须包含一个有效的区号。数据屏蔽转换将在“高层组织列表”上查找区号，并确定一系列未使用的编号，该转换可将这些编号作为屏蔽数据来应用。如果 SSN 无效，则数据屏蔽转换不会屏蔽源数据。

## 可重复社会保障号屏蔽

可为社会保障号配置可重复屏蔽。要配置社会保障号的可重复屏蔽，请单击“可重复输出”并选择“种子值”或“映射参数”。

选择“种子值”时，Designer 会将随机数指定为种子。要在不同的源数据中产生相同的社会保障号，请更改每个数据屏蔽转换中的种子值以匹配其他转换中社会保障号的种子值。如果在映射中定义了数据屏蔽转换，则可以配置种子值的映射参数。

数据屏蔽转换将返回具有可重复屏蔽的确定性社会保障号。数据屏蔽转换无法返回所有唯一社会保障号，因为它无法返回社会保障局已签发的有效社会保障号。

## 信用卡号屏蔽

信用卡屏蔽会应用内置的屏蔽格式来掩藏信用卡号。可以输入多个信用卡号格式。有时，可以替换信用卡发卡行。

PowerCenter 集成服务在屏蔽有效的信用卡号时，逻辑上会生成一个有效的信用卡号。源信用卡号的长度必须介于 13 到 19 位数字之间。根据信用卡行业规则，输入信用卡号的校验和必须有效。

源信用卡号可以包含数字、空格和连字符。如果信用卡的字符数不正确，或者长度错误，则集成服务会将错误写入会话日志。如果源数据无效，PowerCenter 集成服务则会应用默认的信用卡号屏蔽。

可以配置信用卡屏蔽以保留或替换信用卡号的前六位数字。这些数字一起标识信用卡发卡行。如果替换信用卡发卡行，则可以指定另一个信用卡发卡行。可以指定以下信用卡发卡行：

- American Express
- Discover
- JCB
- MasterCard

- Visa
- 任意

如果选择“任意”，数据屏蔽转换会返回所有信用卡发卡行的组合。如果屏蔽信用卡发卡行，数据屏蔽转换会返回不唯一的值。

例如，CUSTOMER 表具有以下信用卡号：

```
2131 0000 0000 0008
5500 0000 0000 0004
6334 0000 0000 0004
```

如果选择单个信用卡发卡行，则信用卡号会共享除最后一位数之外的所有内容。为了生成有效的信用卡号，数据屏蔽转换会将每个信用卡的最后一位数设置为相同的值。

## 电话号码屏蔽

数据屏蔽转换可屏蔽电话号码，但不更改原始电话号码的格式。例如，数据屏蔽转换可以将电话号码 (408)382 0658 屏蔽为 (607)256 3106。

源数据可包含数字、空格、连字符和括号。集成服务不会屏蔽字母或特殊字符。

数据屏蔽转换可以屏蔽字符串、整型和长整型数据。

## 电子邮件地址屏蔽

使用数据屏蔽转换屏蔽包含字符串值的电子邮件地址。数据屏蔽转换可以使用随机 ASCII 字符屏蔽电子邮件地址，也可以将电子邮件地址替换为真实电子邮件地址。

您可以对电子邮件地址应用以下类型的屏蔽：

标准电子邮件屏蔽

数据屏蔽转换屏蔽电子邮件地址时会返回随机 ASCII 字符。例如，数据屏蔽转换可将 Georgesmith@yahoo.com 屏蔽为 KtrlupQAPyk@vdSKh.BIC。默认为标准。

高级电子邮件屏蔽

数据屏蔽转换会使用从转换输出端口或字典列中派生的其他真实电子邮件地址屏蔽电子邮件地址。

## 高级电子邮件屏蔽

通过高级电子邮件屏蔽，可以使用其他真实电子邮件地址来屏蔽电子邮件地址。数据屏蔽转换会从字典列或转换输出端口创建电子邮件地址。

可以从映射输出端口创建电子邮件地址的本地部分。或者，也可以从关系表或平面文件列创建电子邮件地址的本地部分。

数据屏蔽转换可从域字典中的一个常量值或一个随机值创建电子邮件地址的域名。

可以根据以下选项创建高级电子邮件屏蔽：

### 基于映射的电子邮件地址

可以根据数据屏蔽转换输出端口创建电子邮件地址。为名字和姓氏列选择转换输出端口。根据为名字和姓氏长度指定的值，数据屏蔽转换会屏蔽名字、姓氏或这两者。

### 基于字典的电子邮件地址

可以根据字典中的列创建电子邮件地址。字典可以是关系表或平面文件。

为名字和姓氏选择字典列。根据为名字和姓氏长度指定的值，数据屏蔽转换会屏蔽名字、姓氏或这两者。还可以配置一个表达式来屏蔽电子邮件地址。如果配置表达式，则数据屏蔽转换不会根据名字或姓氏列的长度进行屏蔽。

## 高级电子邮件地址屏蔽类型的配置参数

配置高级电子邮件地址屏蔽时指定配置参数。

您可以指定以下配置参数：

### 分隔符

可以选择分隔符来分隔电子邮件地址中的名和姓，例如点、连字符或下划线。 如果不想分隔电子邮件地址中的名和姓，请将分隔符保留为空白。

### FirstName 列

选择数据屏蔽转换输出端口或字典列以屏蔽电子邮件地址中的名。

### LastName 列

选择数据屏蔽转换输出端口或字典列以屏蔽电子邮件地址中的姓。

### FirstName 或 LastName 列的长度

限制名和姓列的屏蔽字符长度。 例如，名的输入数据为 Timothy，姓的输入数据为 Smith。选择 5 作为名的长度。选择 1 作为姓列的长度，并使用点作为分隔符。数据屏蔽转换将生成以下电子邮件地址：

```
timot.s@<domain_name>
```

### DomainName

您可以使用常量值作为域名，例如 gmail.com。或者，可以指定包含一组域名的另一个字典文件。域字典可以是平面文件或关系表。

## 高级电子邮件地址屏蔽类型的表达式

在选择字典列创建电子邮件地址时，可以使用表达式函数。

要配置高级电子邮件屏蔽类型的表达式，请从字典中选择列以构成电子邮件地址。然后，可以使用字典列、转换端口或同时使用字典列和转换端口配置表达式。

在表达式编辑器中，将按以下语法列出字典端口：

```
<输入字符串映射端口>_Dictionary_<字典列名称>
```

## 社会保险号屏蔽

数据屏蔽转换可屏蔽九位数的社会保险号。可使用任意字符集分隔这些数字。

如果该号码不包含分隔符，则屏蔽后的号码也不包含分隔符。否则屏蔽的号码将采用以下格式：

XXX-XXX-XXX

## SIN 起始数字

您可以定义屏蔽 SIN 的第一个数字。

启用**起始数字**，然后输入数字。数据屏蔽转换会从您输入的数字开始创建屏蔽 SIN 数字。

## 可重复的 SIN 编号

可以将数据屏蔽转换配置为返回可重复的 SIN 值。为可重复的 SIN 屏蔽配置端口时，数据屏蔽转换将在每次源 SIN 值和种子值相同时返回确定性的屏蔽数据。

要返回可重复的 SIN 编号，请启用**可重复的值**并输入种子号。数据屏蔽转换将返回每个 SIN 的唯一值。

## IP 地址屏蔽

数据屏蔽转换通过将 IP 地址拆分成 4 个数字（用句点分隔）来将其屏蔽成其他 IP 地址。第一个数字表示网络。数据屏蔽转换可屏蔽网络范围内的网络号。

数据屏蔽转换会将 A 类 IP 地址屏蔽为 A 类 IP 地址，将 10.x.x.x 地址屏蔽为 10.x.x.x 地址。数据屏蔽转换不会屏蔽类和专用网络地址。例如，数据屏蔽转换可以将 11.12.23.34 屏蔽为 75.32.42.52，将 10.23.24.32 屏蔽为 10.61.74.84。

**注意：**屏蔽多个 IP 地址时，数据屏蔽转换可以返回非唯一值，因为它不会屏蔽类或专用网络的 IP 地址。

## URL 地址屏蔽

数据屏蔽转换通过搜索 “://” 字符串并解析其右侧的子字符串来解析 URL。源 URL 必须包含 “://” 字符串。源 URL 可以包含数字和字母字符。

数据屏蔽转换不会屏蔽 URL 协议。例如，如果 URL 为 http://www.yahoo.com，数据屏蔽转换将返回 http://MgL.aHjCa.VsD/。数据屏蔽转换会生成无效 URL。

**注意：**数据屏蔽转换总是针对 URL 返回 ASCII 字符。

## 默认值文件

源数据格式或数据类型对于屏蔽无效时，集成服务会对数据应用默认屏蔽。集成服务将应用默认值文件中的屏蔽值。可以编辑默认值文件以更改默认值。

默认值文件是位于以下位置的 XML 文件：

```
<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

defaultValue.xml 文件包含以下名称-值对：

```
<?xml version="1.0" standalone="yes" ?>
<defaultValue
  default_char = "X"
  default_digit = "9"
  default_date = "11/11/1111 00:00:00"
  default_email = "abc@xyz.com"
  default_ip = "99.99.9.999"
  default_url = "http://www.xyz.com"
  default_phone = "999999999"
  default_ssn = "999-99-9999"
  default_cc = "9999 9999 9999 9999"
  default_seed = "500"
/>
```

## 数据屏蔽转换会话属性

可以配置数据屏蔽转换会话属性以提高性能。

配置以下会话属性：

### 缓存大小

主内存中字典缓存的大小。增加内存大小以提高性能。对于 100,000 条记录，建议的最小大小为 32 MB。默认值为 8 MB。

### 缓存目录

字典缓存的位置。必须具有该目录的写入权限。默认为 \$PMCacheDir。

### 共享存储表

在数据屏蔽转换实例之间启用存储表共享。当两个数据屏蔽转换实例为数据库连接、种子值和区域设置使用相同的字典列时，可启用共享存储表。当同一数据屏蔽转换中的两个端口为连接、种子和区域设置使用相同的字典列时，也可以启用共享存储表。当数据屏蔽转换或端口未共享字典列时，将禁用共享存储表。默认为“已禁用”。

### 存储提交时间间隔

一次提交到存储表中的行数。增加该值可提高性能。未配置共享存储表时，请配置提交时间间隔。默认值为 100,000。

### 对存储进行加密

对存储表（例如 IDM\_SUBSTITUTION\_STORAGE 和 IDM\_EXPRESSION\_STORAGE）进行加密。在启用加密存储属性之前，请验证是否已对存储表中的数据进行加密。如果不希望对存储表进行加密，请清除此选项。默认为“已禁用”。

### 存储加密键

数据屏蔽转换将基于存储加密键对存储进行加密。为同一数据屏蔽转换实例的所有会话运行使用相同的加密键。

# 数据屏蔽转换的规则和准则

配置数据屏蔽转换时，请使用以下规则和准则：

- 数据屏蔽转换不屏蔽空值。如果源数据包含空值，则数据屏蔽转换会返回空值。要替换空值，请为输入端口添加允许使用用户定义的默认值的上游转换。
- 源数据格式或数据类型对于屏蔽无效时，集成服务会对数据应用默认屏蔽。集成服务将应用默认值文件中的屏蔽值。
- 数据屏蔽转换会返回与源的格式和区域代码相同的无效社会保障号。如果社会保障局在某个地区签发的社会保障号已超过半数，则数据屏蔽转换可能无法返回具有键屏蔽的唯一无效社会保障号。

## 第 6 章

# 数据屏蔽示例

本章包括以下主题：

- [名称和地址查找文件, 135](#)
- [通过查找转换置换数据, 135](#)
- [通过表达式转换屏蔽数据, 138](#)

## 名称和地址查找文件

数据屏蔽安装包括多个平面文件，其中包含用于屏蔽数据的姓名、地址和公司名称。可以配置查找转换以从这些文件中检索随机名字、姓氏和地址。

安装数据屏蔽服务器组件时，安装程序会在 server\infa\_shared\LkpFiles 文件夹中放置以下文件：

- **Address.dic.**地址的平面文件。每个记录都具有序列号、街道、城市、州/省/自治区、邮政编码和国家。
- **Company\_names.dic.**公司名称的平面文件。每个记录都具有序列号和公司名称。
- **Firstnames.dic.**名字的平面文件。每条记录都具有序列号、名字和性别代码。
- **Surnames.dic.**姓氏的平面文件。每条记录都具有序列号和姓氏。

## 通过查找转换置换数据

可以替代具有类似但不相关数据的数据列。替代是一种使用逼真数据屏蔽敏感数据的有效方式。

以下示例显示了如何配置多个查找转换以检索测试数据并为源数据替代该测试数据。创建数据屏蔽映射以屏蔽 CUSTOMERS\_PROD 表中的敏感字段。

该示例包括以下屏蔽类型：

- 查找表中的名称和地址替代
- 键屏蔽
- 模糊
- 特殊屏蔽格式

**注意：**本示例是 M\_CUSTOMERS\_MASKING.xml 映射，可以从 client\samples 文件夹中导入到存储库。

名为 Customers\_Prod 的客户数据库表包含敏感数据。想要在测试方案中使用客户数据，但又想要维护安全。屏蔽每个列中的数据并将测试数据写入名为 Customers\_Test 的目标表。

Customers\_Prod 表包含以下列：

列	数据类型
CustID	整型
FullName	字符串
地址	字符串
电话	字符串
传真	字符串
CreateDate	日期
电子邮件	字符串
SSN	字符串
CreditCard	字符串

可以创建在字典文件中查找替代值的映射。数据屏蔽转换使用字典文件中的值屏蔽客户数据。这些文件包括名字文件、姓氏文件和地址文件。

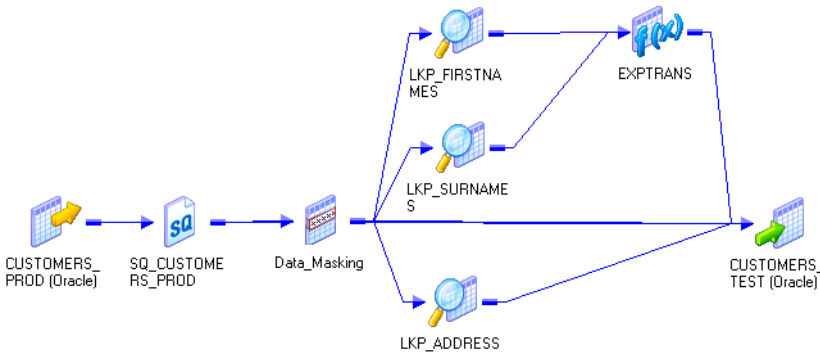
下表列出了位于 server\infa\_shared\LkpFiles 文件夹中的文件：

文件	记录数	字段	说明
Firstnames.dic	21,000	SNO、性别、名字	按字母顺序排列的名字列表。序列号范围为 1 到 21,000。性别指示名字是男性还是女性。
Surnames.dic	81,000	SNO、名字	按字母顺序排列的姓氏列表。序列号范围为 1 到 81,000。
Address.dic	13,000	SNO、街道、城市、省/自治区/直辖市、邮政编码、国家/地区	完整地址列表。序列号范围为 1 到 13,000。

**注意:** Informatica 在 Firstnames.dic 文件中包括性别列，以便您可以按性别创建单独的查找源文件。如果需要 使用女性名屏蔽男性名和女性名，则可以在查找条件中使用性别列。



下图显示了可以导入的映射：



映射具有以下转换以及源和目标：

- **源限定符。**将客户数据传递到数据屏蔽转换。将 CustID 列传递到转换中的多个端口：
  - **CustID。**客户编号。
  - **Randid1。**名字查找的随机编号生成器。
  - **Randid2。**姓氏查找的随机编号生成器。
  - **Randid3。**地址查找的随机编号生成器。
- **数据屏蔽转换。**创建随机编号以查找替换名字、姓氏和地址。为电话号码、传真、电子邮件地址和信用卡号应用特殊屏蔽格式。数据屏蔽转换会屏蔽以下列：

输入端口	屏蔽类型	屏蔽规则	说明	输出目标
CustID	键	种子 = 934	CustID 是主键列，必须使用可重复的确定性的随机编号屏蔽。	Customers_Test
Randid1	随机	范围 最小值 = 0 最大值 = 21000	LKUP_Firstnames 转换中名字查找的随机编号。	LKUP_Firstnames
Randid2	随机	范围 最小值 = 0 最大值 = 13000	LKUP_Surnames 转换中姓氏查找的随机编号。	LKUP_Surnames
Randid3	随机	范围 最小值 = 0 最大值 = 81000	LKUP_Address 转换中地址查找的随机编号。	LKUP_Address
电话	电话	-	电话号码与源电话号码具有相同的格式。	Customers_Test
传真	电话	-	电话号码与源电话号码具有相同的格式。	Customers_Test

输入端口	屏蔽类型	屏蔽规则	说明	输出目标
CreatedDate	随机	模糊 单位 = 年 下限 = 1 上限 = 1	随机日期，在源年份的一年以内。	Customers_Test
电子邮件	电子邮件地址	-	电子邮件地址与原始地址具有相同的格式。	Customers_Test
SSN	SSN	-	SSN 不在 highgroup.txt 文件中。	Customers_Test
CreditCard	信用卡	-	信用卡与源信用卡具有相同的前六位数，且具有有效的校验和。	Customers_Test

- **LKUP\_Firstnames。**对 Firstnames.dic 执行平面文件查找。转换使用等于随机编号 Randid1 的序列号检索记录。查找条件为：  
SNO = out\_RANDID1  
LKUP\_Firstnames 转换会将屏蔽的名字传递到 Exptrans 表达式转换。
  - **LKUP\_Surnames。**对 Surnames.dic 文件执行平面文件查找。使用等于 Randid2 的序列号检索记录。LKUP\_Firstnames 转换会将屏蔽的姓氏传递到 Exptrans 表达式转换。
  - **Exptrans。**合并名字和姓氏并返回全名。表达式转换会将全名传递到 Customers\_Test 目标。  
合并名字和姓氏的表达式为：  
FIRSTNAME || ' ' || SURNAME
  - **LKUP\_Address。**对 Address.dic 文件执行平面文件查找。它将使用等于 Randid3 的序列号检索地址记录。查找转换会将地址中的列传递到目标。
- 可以在测试环境中使用 Customer\_Test 表。

## 通过表达式转换屏蔽数据

将表达式转换与数据屏蔽转换配合使用，以在屏蔽一个列之后保留两个列之间的关系。

例如，在屏蔽包含保险政策开始日期和结束日期的帐户信息时，要在每条屏蔽记录中保留政策持续时间。使用数据屏蔽转换屏蔽除结束日期之外的所有数据。使用表达式转换计算政策持续时间，并将政策持续时间添加到屏蔽的开始日期。

此示例包括以下类型的屏蔽：

- 键
- 日期模糊
- 数字模糊
- 屏蔽格式化

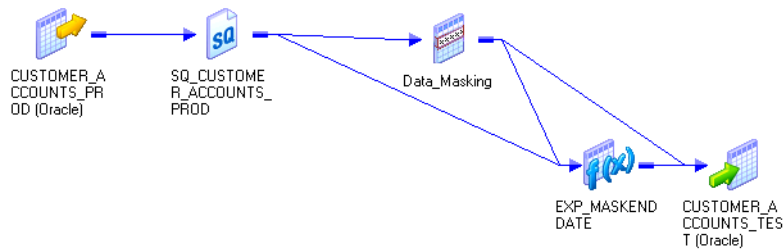
**注意：**此示例是 M\_CUSTOMER\_ACCOUNTS\_MASKING.xml 映射，可从 client\samples 文件夹导入到存储库。

名为 Customers\_Prod 的客户数据库表包含敏感数据。屏蔽每个列中的数据并将测试数据写入名为 Customers\_Test 的目标表。

屏蔽以下 Customer\_Accounts\_Prod 列：

列	数据类型
AcctID	字符串
CustID	整型
Balance	双精度
StartDate	日期时间
EndDate	日期时间

下图显示了可以导入的映射：



该映射具有以下转换以及源和目标：

- **源限定符。** 将 AcctID、CustID、Balance 和 Start\_Date 传递到数据屏蔽转换。它将 Start\_Date 和 End\_Date 列传递到表达式转换。
- **数据屏蔽转换。** 屏蔽除 End\_Date 之外的所有列。数据屏蔽转换会将屏蔽的列传递给目标。它会将政策开始日期、结束日期和屏蔽的开始日期传递到表达式转换。

数据屏蔽转换会屏蔽以下列：

输入端口	屏蔽类型	屏蔽规则	说明	输出目标
AcctID	随机	屏蔽格式 AA+DDDDD 结果字符串替换字 符 ABCDEFGHIJKLMNOPQRSTUVWXYZ NOPQRSTUVWXYZ	前两个字符是大写字母。第三个字符是短划线，不会被屏蔽。后五个字符是数字。	Customer_Account_Test 目标
CustID	键	种子 = 934	种子为 934。CustID 屏蔽是确定性的。	Customer_Account_Test 目标

输入端口	屏蔽类型	屏蔽规则	说明	输出目标
Balance	随机	模糊 百分比 下限 = 10 上限 = 10	屏蔽的余额在源余额的 10% 以内。	Customer_Account_Test 目标
Start_Date	随机	模糊 单位 = 年 下限 = 2 上限 = 2	屏蔽的 start_date 在源 日期的两年内。	Customer_Account_Test 目标 Exp_MaskEndDatetransf ormation

- **表达式转换。**计算屏蔽的结束日期。它会计算开始日期与结束日期之间的时间。它会将时间添加到屏蔽的开始日期以确定屏蔽的结束日期。

要生成屏蔽结束日期的表达式为：

```
DIFF = DATE_DIFF(END_DATE,START_DATE,'DD')  
out_END_DATE = ADD_TO_DATE(out_START_DATE,'DD',DIFF)
```

该表达式转换将 out\_END\_DATE 传递到目标。

## 第 7 章

# 表达式转换

本章包括以下主题：

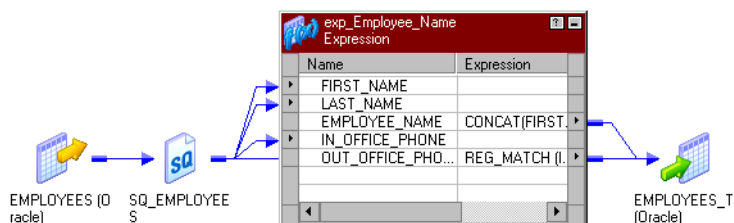
- [表达式转换概览, 141](#)
- [表达式转换组件, 141](#)
- [配置端口, 142](#)
- [创建表达式转换, 142](#)

## 表达式转换概览

使用表达式转换可计算单个行中的值。例如，您可能需要调整员工薪酬、连接名字和姓氏，或将字符串转换为数字。将结果传递给目标或其他转换之前，还可以使用表达式转换测试条件语句。表达式转换是一种被动转换。

使用表达式转换可执行非汇总计算。要执行涉及多行的计算（例如求和或平均值），请使用汇总器转换。

下图显示了一个包含表达式转换的简单映射，用于串联 EMPLOYEES 表中员工的名字和姓氏：



可以计算在表达式编辑器中配置的表达式。

## 表达式转换组件

可以在 Transformation Developer 或 Mapping Designer 中创建表达式转换。

表达式转换包含以下选项卡：

- **转换。**输入转换的名称和说明。表达式转换的命名约定是 `EXP_TransformationName`。也可以使转换可重用。

- **端口。**创建和配置端口。
- **属性。**配置跟踪级别，以确定会话日志文件中报告的事务详细信息量。
- **元数据扩展。**指定扩展名称、数据类型、精度和值。也可以创建可重用元数据扩展。

## 配置端口

您可以在“端口”选项卡上创建和修改端口。

在“端口”选项卡上配置以下组件：

- **端口名称。**端口的名称。
- **数据类型、精度和小数位数。**配置每个端口的数据类型并设置精度和小数位数。
- **Port type.**端口可以是输入、输出、输入/输出或变量类型。输入端口接收数据，输出端口传递数据。输入/输出端口原样传递数据。变量端口临时存储数据，并可在行间存储值。
- **表达式。**使用表达式编辑器输入表达式。表达式使用转换语言（包括类似于 SQL 的函数）执行计算。
- **默认值和说明。**设置端口的默认值并添加说明。

## 计算值

要使用表达式转换计算单个行的值，必须包括以下端口：

- **输入或输入/输出端口。**提供计算中使用的值。例如，如果您需要计算订单的总价格，可创建两个输入或输入/输出端口。一个端口提供单价，另一个提供订购数量。
- **输出端口。**提供表达式的返回值。您输入的表达式将成为输出端口的配置选项。还可以为每个端口配置默认值。

通过为每个输出端口创建表达式，可以在单个表达式转换中输入多个表达式。例如，您可能要计算每个员工薪酬中不同类型的代扣缴税款，例如当地所得税、联邦所得税、社会保险和医疗保险。由于所有这些计算都涉及员工薪酬、代扣缴类别，并可能需要使用相应的税率，因此可以创建输入/输出端口用于薪酬和代扣缴类别，并创建单独的输出端口用于每种计算。

## 创建表达式转换

按照以下步骤创建表达式转换：

1. 在 Mapping Designer 中，打开一个映射。
2. 单击“转换”>“创建”。选择表达式转换。
3. 输入名称并单击“完成”。
4. 从源限定符或其他转换中选择并拖动端口，以添加到表达式转换中。  
您也可以打开该转换，手动创建端口。
5. 双击标题栏，然后单击“端口”选项卡。在该转换内可以创建输出和变量端口。
6. 分配端口数据类型、精度和小数位数，以与表达式返回值匹配。
7. 在输出或变量端口的“表达式”部分，打开表达式编辑器。

8. 输入表达式。
9. 单击“验证”以验证表达式语法。
10. 若要在右窗格中的“测试表达式”部分反映表达式条件的最新更改，请单击“刷新”。
11. 在右窗格中，输入表达式中使用的输入端口的示例值。
12. 要计算表达式，请单击“计算”。
13. 单击“确定”。
14. 在“转换”选项卡上创建可重用转换。  
**注意：**使该转换可重用之后，则无法复制源限定符或其他转换中的端口。您 *可以* 在该转换内手动创建端口。
15. 在“属性”选项卡上配置跟踪级别。
16. 在“元数据扩展”选项卡上添加元数据扩展。
17. 单击“确定”。
18. 将输出端口连接到下游转换或目标。

## 第 8 章

# 外部过程转换

本章包括以下主题：

- [外部过程转换概览, 144](#)
- [配置外部过程转换属性, 145](#)
- [开发 COM 过程, 147](#)
- [开发 Informatica 外部过程, 154](#)
- [分发外部过程, 161](#)
- [开发说明, 162](#)
- [初始化属性中的服务进程变量, 168](#)
- [外部过程接口, 168](#)

## 外部过程转换概览

外部过程转换可与您在 Designer 接口外部创建的过程结合使用，以扩展 PowerCenter 功能。

标准转换具有一系列选项，即便如此也会遇到想要扩展 PowerCenter 功能的情况。例如，表达式转换和筛选器转换等标准转换范围可能不会提供您所需的功能。如果您是一名经验丰富的编程人员，您可能想在动态链接库 (DLL) 或 UNIX 共享库中开发复杂函数，而不是在映射中创建必要的表达式转换。

要获得这种扩展功能，可使用 PowerCenter 中内置的 Transformation Exchange (TX) 动态调用接口。使用 TX 可以创建 Informatica 外部过程转换，并将其与开发的外部过程绑定。可以将外部过程转换与两类外部过程绑定：

- COM 外部过程（仅在 Windows 上可用）
- Informatica 外部过程（在 Windows、AIX、Linux 和 Solaris 上可用）

只有经验丰富的 C、C++ 或 Visual Basic 编程人员才有能力使用 TX。

在外部过程中使用多线程代码。

## 代码页兼容性

当集成服务在 ASCII 模式下运行时，外部过程可以处理 7 位 ASCII 格式的数据。当集成服务在 Unicode 模式下运行时，外部过程可以处理与集成服务代码页双向兼容的数据。

如果外部过程 DLL 或共享库包含多字节字符，请将集成服务配置为在 Unicode 模式下运行。外部过程必须使用与集成服务相同的代码页来解释来自集成服务的输入字符串，以及创建包含多字节字符的输出字符串。

如果外部过程 DLL 或共享库仅包含 ASCII 字符，请将集成服务配置为在 ASCII 或 Unicode 模式下运行。



# 外部过程和外部过程转换

TX 包含两个组件：*外部过程*和*外部过程转换*。

*外部过程*独立于集成服务。它包括用户编写的用于定义转换的 C、C++ 或 Visual Basic 代码。此代码被编译并链接至 DLL 或共享库，集成服务运行时将加载此代码。外部过程“绑定”至外部过程转换。

*外部过程转换*在 Designer 中创建。它是驻留在 Informatica 存储库的对象，具有多种用途：

- 1. 它包含描述以下外部过程的元数据。正是通过这些元数据，集成服务才获悉外部过程的“签名”（参数的数量和类型、返回值的类型，如果有的话）。
- 2. 它支持在映射中引用外部过程。通过在映射中添加外部过程转换实例，您可以调用与该转换绑定的外部过程。

**注意：**可以创建已连接或未连接的外部过程。

- 3. 开发 Informatica 外部过程时，外部过程转换可提供生成 Informatica 外部过程存根所需的信息。

## 外部过程转换属性

在 Transformation Developer 中创建可重用外部过程转换，并将转换实例添加到映射中。不能在 Mapping Designer 或 Mapplet Designer 中创建外部过程转换。

外部过程转换可为每个输入行返回一个输出行，或不返回输出行。

在外部过程转换的“属性”选项卡上，只能在“模块/编程标识符”和“过程名称”字段中输入 ASCII 字符。不能在这些字段中输入多字节字符。在外部过程转换的“端口”选项卡上，只能输入 ASCII 字符作为端口名称。不能输入多字节字符作为外部过程转换端口名称。

## COM 与 Informatica 外部过程对比

下表描述了 COM 和 Informatica 外部过程之间的区别：

	COM	Informatica
技术	使用 COM 技术	使用 Informatica 专有技术
操作系统	仅在 Windows 上运行	在支持集成服务的所有平台上运行：Windows、AIX、HP、Linux、Solaris
语言	C、C++、VC++、VB、Perl、VJ++	仅限 C++

## BankSoft 示例

以下部分使用称为 BankSoft 的示例来说明如何开发 COM 和 Informatica 过程。BankSoft 示例使用财务函数 FV 来说明如何开发和调用外部过程。FV 过程会根据常规付款和固定利率计算投资的未来值。

## 配置外部过程转换属性

在“属性”选项卡上配置转换属性。

下表描述了外部过程转换属性：

属性	说明
类型	<p>外部过程的类型。使用以下类型：</p> <ul style="list-style-type: none"> <li>- COM</li> <li>- Informatica</li> </ul> <p>默认值为 Informatica。</p>
模块/编程标识符	<p>模块是包括外部过程的 DLL（在 Windows 上）或共享对象（在 UNIX 上）的基名。它决定了操作系统上 DLL 或共享对象的名称。</p> <p>只能输入 ASCII 字符。</p> <p>编程标识符或 ProgID 是类的逻辑名。在 Designer 中可通过 ProgID 引用 COM 类。在内部类由数值 CLSID 标识。例如：</p> <p><b>{33B17632-1D9F-11D1-8790-0000C044ACF9}</b></p> <p>ProgID 的标准格式为 <i>Project.Class[.Version]</i>。</p> <p>只能输入 ASCII 字符。</p>
过程名称	<p>外部过程的名称。只能输入 ASCII 字符。</p>
运行时位置	<p>包含 DLL 或共享库的位置。输入相对于运行外部过程会话的集成服务节点的路径。如果您输入 \$PMExtProcDir，集成服务会在进程变量 \$PMExtProcDir 指定的目录中查找库。</p> <p>如果此属性为空，集成服务将使用集成服务节点上定义的环境变量来查找 DLL 或共享库。</p> <p>可以按硬编码方式将路径处理为运行时位置。由于路径仅特定于一个计算机，因此，不推荐这种方式。</p> <p>您必须将所有 DLL 或共享库复制到运行时位置或在集成服务节点上定义的环境变量。如果不能找到 DLL、共享库或引用文件，集成服务无法加载过程。</p> <p>默认为 \$PMExtProcDir。</p>
跟踪级别	<p>会话日志文件中报告的事务详细信息量。使用以下跟踪级别：</p> <ul style="list-style-type: none"> <li>- 简洁</li> <li>- 普通</li> <li>- 详细初始化</li> <li>- 详细数据</li> </ul> <p>默认值为“普通”。</p>
是否可分区	<p>指示是否可以在使用此转换的管道中创建多个分区。使用以下值：</p> <ul style="list-style-type: none"> <li>- 否。无法对转换进行分区。转换和同一渠道中的其他转换只有一个分区。</li> <li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。当 BAPI/RFC 转换的不同分区必须共享内存中的对象时，请选择“本地”。</li> <li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li> </ul> <p>默认值为“否”。</p>

属性	说明
输出是可重复的	<p>指示转换是否在会话运行之间以相同的顺序生成行。当输出是可重复的确定性输出时，集成服务可以从上次检查点恢复会话。使用以下值：</p> <ul style="list-style-type: none"> <li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li> <li>- 基于输入顺序。当所有输入组的输入数据顺序在会话运行之间一致时，转换可在会话运行之间生成可重复数据。如果所有输入组中的输入数据均未排序，输出数据也不会排序。</li> <li>- 从不。会话运行之间的输出数据顺序不一致。如果转换不生成可重复数据，则无法将恢复配置为从上次检查点继续。</li> </ul> <p>默认值为“基于输入顺序”。</p>
输出具有确定性	<p>指示转换在多次会话运行时是否生成一致的输出数据。必须启用此属性以对使用此转换的会话执行恢复。</p> <p>默认为“已禁用”。</p>

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

下表描述了集成服务在各种平台上查找 DLL 或共享对象作为运行时位置使用的环境变量：

表 1. 环境变量

操作系统	环境变量
Windows	PATH
AIX	LIBPATH
HPUX	SHLIB_PATH
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH

## 开发 COM 过程

可以使用 Microsoft Visual C++ 或 Visual Basic 开发 COM 外部过程。下面的内容介绍了使用 Visual C++ 如何创建 COM 外部过程以及使用 Visual Basic 如何创建 COM 外部过程。

### COM 过程创建步骤

要创建 COM 外部过程，请完成以下步骤：

1. 使用 Microsoft Visual C++ 或 Visual Basic 创建一个项目。
2. 定义一个带有 *IDispatch* 接口的类。
3. 将方法添加到该接口。该方法是从集成服务内部调用的外部过程。
4. 编译该类并将其链接到动态链接库。

5. 在本地 Windows 注册表中注册该类。
6. 在 Transformation Developer 中导入 COM 过程。
7. 使用 COM 过程创建一个映射。
8. 使用该映射创建一个会话。

## COM 外部过程服务器类型

集成服务仅支持 *服务器类型* 为 *动态链接库* 的进程内 COM 服务器。此举的目的是改善性能。处理大量数据时效率更高，这时可以在同一进程中处理数据，而不是将其转发至同一计算机或远程计算机上的独立进程。

## 使用 Visual C++ 开发 COM 过程

C++ 开发人员可以使用 Visual C++ 版本 5.0 或更高版本开发 COM 过程。第一项任务是创建项目。

相关主题：

- [“分发外部过程” 页面上 161](#)
- [“已有 C/C++ 库或 VB 函数的包装类” 页面上 164](#)

### 步骤 1. 创建 ATL COM AppWizard 项目

1. 启动 Visual C++，然后单击“文件”>“新建”。
2. 在显示的对话框中，选择“项目”选项卡。
3. 输入项目名称和位置。  
在 BankSoft 示例中，输入 *COM\_VC\_Banksoft* 作为项目名称，*c:\COM\_VC\_Banksoft* 作为目录。
4. 在项目列表框中，选择 *ATL COM AppWizard* 选项，然后单击“确定”。  
此时将显示一个向导，用于在 Visual C++ 中创建 COM 项目。
5. 将“服务器类型”设置为“动态链接库”，选择 *支持 MFC* 选项，然后单击“完成”。  
此时将显示该向导的最终页面。
6. 单击“确定”以返回 Visual C++。
7. 为新项目添加类。
8. 在该向导的下一页面上，单击“确定”按钮。Developer Studio 将创建基本项目文件。

### 步骤 2. 在项目中添加 ATL 对象

1. 在“工作区”窗口中，选择“类查看”选项卡，右键单击树项 *COM\_VC\_BankSoft.BSoftFin* 类，然后从显示的本地菜单中选择“新建 ATL 对象”。
2. 突出显示左侧列表框中的“对象”项，然后从对象类型列表中选择 *简单对象*。
3. 单击“下一步”。
4. 在“短名称”字段中，输入要创建的类的短名称。  
在 BankSoft 示例中，使用名称 *BSoftFin*，因为您要为虚构公司 BankSoft 开发一项财务功能。当您键入到“短名称”字段时，向导会在其他字段中填入建议的名称。
5. 输入该类的编程标识符。  
在 BankSoft 示例中，将“ProgID”（编程标识符）字段更改为 *COM\_VC\_BankSoft.BSoftFin*。

编程标识符 (ProgID) 是类的有意义的标识名称。在内部类由数值 CLSID 标识。例如：

```
{33B17632-1D9F-11D1-8790-0000C044ACF9}
```

ProgID 的标准格式为 *Project.Class[.Version]*。在 Designer 中可通过 ProgID 引用 COM 类。

6. 选择“属性”选项卡，并将线程模型设置为 *自由*，将接口设置为 *双*，并将汇总设置设置为 *否*。

7. 单击“确定”。

有了已具有基本类定义之后，可以为其添加方法。

### 步骤 3。在类中添加所需方法

1. 返回到“工作区”窗口的“类查看”选项卡。

2. 展开树视图。

对于 BankSoft 示例，展开 *COM\_VC\_BankSoft*。

3. 右键单击新添加的类。

在 BankSoft 示例中，右键单击 *IBSoftFin* 树项。

4. 单击“添加方法”菜单项并输入方法的名称。

在 BankSoft 示例中，输入 *FV*。

5. 在“参数”字段中，输入方法的签名。

对于 FV，输入以下内容：

```
[in] double Rate,  
[in] long nPeriods,  
[in] double Payment,  
[in] double PresentValue,  
[in] long PaymentType,  
[out, retval] double* FV
```

该签名使用 Microsoft 接口说明语言 (MIDL) 的术语表示。有关 MIDL 的完整说明，请参阅 MIDL 语言参考。  
请注意：

- **[in]** 指示该参数是输入参数。
- **[out]** 指示该参数是输出参数。
- **[out, retval]** 指示该参数是方法的返回值。

而且，所有 [out] 参数都通过引用传递。在 BankSoft 示例中，参数 FV 是双精度型。

6. 单击“确定”。

Developer Studio 会向项目中添加您添加的方法的存根。

### 步骤 4。使用实施填充方法存根

1. 在 BankSoft 示例中，返回到“工作区”窗口的“类查看”选项卡，并展开 *COM\_VC\_BankSoft* 类项。

2. 展开 *CBSofFin* 项。

3. 在上面的项下展开 *IBSoftFin* 项。

4. 右键单击 *FV* 项，然后选择“转到定义”。

5. 将光标放在编辑窗口中 TODO 注释之后的行上，并添加以下代码：

```
double v = pow((1 + Rate), nPeriods);  
*FV = -(  
    (PresentValue * v) +  
    (Payment * (1 + (Rate * PaymentType)))) * ((v - 1) / Rate)  
);
```

由于引用 pow 函数，因此您必须在文件开头的所有其他包含语句后添加以下预处理器语句：

```
#include <math.h>
```

最后一步是构建 DLL。当您进行构建时，使用 Windows 注册表注册 COM 过程。

## 步骤 5。构建项目

1. 拉下“构建”菜单。
2. 选择“全部重建”。

在 Developer Studio 构建项目时，会生成以下输出：

```
-----Configuration: COM_VC_BankSoft - Win32 Debug-----  
  
Performing MIDL step  
Microsoft (R) MIDL Compiler Version 3.01.75  
Copyright (c) Microsoft Corp 1991-1997. All rights reserved.  
Processing .\COM_VC_BankSoft.idl  
COM_VC_BankSoft.idl  
Processing C:\msdev\VC\INCLUDE\oidl.idl  
oidl.idl  
Processing C:\msdev\VC\INCLUDE\objidl.idl  
objidl.idl  
Processing C:\msdev\VC\INCLUDE\unknwn.idl  
unknwn.idl  
Processing C:\msdev\VC\INCLUDE\wtypes.idl  
wtypes.idl  
Processing C:\msdev\VC\INCLUDE\ocidl.idl  
ocidl.idl  
Processing C:\msdev\VC\INCLUDE\oleidl.idl  
oleidl.idl  
Compiling resources...  
Compiling...  
StdAfx.cpp  
Compiling...  
COM_VC_BankSoft.cpp  
BSoftFin.cpp  
Generating Code...  
Linking...  
Creating library Debug/COM_VC_BankSoft.lib and object Debug/COM_VC_BankSoft.exp  
Registering ActiveX Control...
```

```
RegSvr32: DllRegisterServer in .\Debug\COM_VC_BankSoft.dll succeeded.  
COM_VC_BankSoft.dll - 0 error(s), 0 warning(s)
```

请注意，Visual C++ 会编译项目中的文件，将其链接到称为 COM\_VC\_BankSoft.DLL 的动态链接库 (DLL)，然后在本地注册表中注册 COM (ActiveX) 类 COM\_VC\_BankSoft.BSoftFin。

注册组件后，该主机上运行的集成服务便可访问该组件。

## 步骤 6. 在存储库中注册 COM 过程

1. 打开 Transformation Developer。
2. 单击“转换”>“导入外部过程”。
- 此时将显示“导入外部 COM 方法”对话框。
3. 单击“浏览”按钮。
4. 选择您创建的 COM DLL 并单击“确定”。
- 在 Banksoft 示例中，选择 *COM\_VC\_Banksoft.DLL*。
5. 在“选择方法”树视图下，展开类节点（在本示例中为 BSoftFin）。
6. 展开方法。
7. 选择需要的方法（在本示例中为 FV）并按“确定”。
- Designer 会创建外部过程转换。
8. 打开外部过程转换，然后选择“属性”选项卡。
- 输入“模块/编程标识符”和“过程名称”字段。
9. 单击“端口”选项卡。
10. 输入端口名称。
11. 单击“确定”。

## 步骤 7. 为映射创建源和目标

可以使用以下 SQL 语句创建源表，并用示例数据填充此表：

```
create table FVInputs(  
    Rate float,  
    nPeriods int,  
    Payment float,  
    PresentValue float,  
    PaymentType int  
)  
  
insert into FVInputs values (.005,10,-200.00,-500.00,1)  
insert into FVInputs values (.01,12,-1000.00,0.00,0)  
insert into FVInputs values (.11/12,35,-2000.00,0.00,1)  
insert into FVInputs values (.005,12,-100.00,-1000.00,1)
```

可以使用以下 SQL 语句创建目标表：

```
create table FVOutputs(  
    FVin_ext_proc float,  
)
```

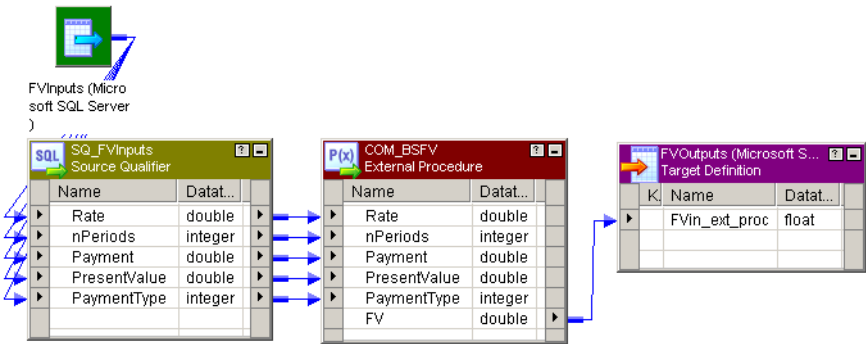
可以使用 Source Analyzer 和 Target Designer 将 FVInputs 和 FVOutputs 导入到您创建 COM\_BSFV 转换的文件夹中。

### 步骤 8。创建测试外部过程转换的映射

现在创建一个映射，用于测试外部过程转换：

1. 在 Mapping Designer 中，创建名为 Test\_BSFV 的映射。
2. 将源表 FVInputs 拖放到该映射中。
3. 将目标表 FVOutputs 拖放到该映射中。
4. 将转换 COM\_BSFV 拖放到该映射中。
5. 在适当情况下，将源限定符转换端口连接到外部过程转换端口。
6. 将外部过程转换中的 FV 端口连接到 FVIn\_ext\_proc 目标列。
7. 验证并保存该映射。

下图显示了完整映射：



### 步骤 9。启动集成服务

请启动集成服务。必须在注册 COM 组件的同一台主机上启动该服务。

### 步骤 10。运行工作流测试映射

当集成服务在工作流中运行会话时，它将执行以下函数：

- 使用 COM 运行时设备加载 DLL，并创建类的实例。
- 使用 COM IDispatch 接口调用曾为每一行定义、用于传递映射的外部过程。

**注意：**可在一个项目中定义多个类，每个类可以包含多种方法。其中每种方法均可作为外部过程调用。

要运行工作流以测试映射：

1. 在 Workflow Manager 中，通过 Test\_BSFV 映射创建会话 s\_Test\_BSFV。
2. 创建包含会话 s\_Test\_BSFV 的工作流。
3. 运行工作流。集成服务将搜索注册表，查找 COM\_VC\_BankSoft.BSoftFin 类的项。此项包含信息，允许集成服务确定包含该类的 DLL 的位置。集成服务将加载该 DLL，创建该类的实例，然后为源表中的每一行调用 FV 函数。

当工作流完成时，FVOutputs 表应该包含以下结果：

**FVIn\_ext\_proc**

2581.403374



FVIn\_ext\_proc

12682.503013

82846.246372

2301.401830

## 使用 Visual Basic 开发 COM 过程

Microsoft Visual Basic 提供了创建 COM 过程的不同开发环境。尽管 Basic 语言具有不同的语法和约定，但开发过程提供的大纲种类与在 Visual C++ 中开发 COM 过程相同。

### 相关主题：

- [“分发外部过程” 页面上 161](#)
- [“已有 C/C++ 库或 VB 函数的包装类” 页面上 164](#)

### 步骤 1。创建具有单个类的 Visual Basic 项目

1. 启动 Visual Basic 并单击“文件”>“新建项目”。
2. 在显示的对话框中，选择 ActiveX DLL 作为项目类型并单击“确定”。  
Visual Basic 会创建一个名为 *Project1* 的新项目。  
如果不显示“项目”窗口，请键入 Ctrl+R，或单击“查看”>“项目浏览器”。  
如果不显示“属性”窗口，请按 F4 或单击“查看”>“属性”。
3. 在新项目的“项目浏览器”窗口中，右键单击该项目并从显示的菜单中选择“Project1 属性”。
4. 输入新项目的名称。  
在“项目”窗口中，选择 *Project1* 并在“属性”窗口中将名称更改为 *COM\_VB\_BankSoft*。

### 步骤 2。更改项目和类的名称

1. 在项目浏览器内部，选择“项目 - Project1”项，这应该是树控制项中的根目录项。项目属性显示在“属性”窗口中。
2. 在“属性”窗口中选择“字母顺序”选项卡，并将“名称”属性更改为 COM\_VB\_BankSoft。这会将项目浏览器中的根目录项重命名为 COM\_VB\_BankSoft (COM\_VB\_BankSoft)。
3. 在项目浏览器中展开 COM\_VB\_BankSoft (COM\_VB\_BankSoft) 项。
4. 展开“类模块”项。
5. 选择“Class1 (Class1)”项。类属性显示在“属性”窗口中。
6. 在“属性”窗口中选择“字母顺序”选项卡，并将“名称”属性更改为 BSoftFin。

通过更改项目和类的名称，指定您所创建类的编程标识符为“COM\_VB\_BankSoft.BSoftFin”。使用 ProgID 在 Designer 内部引用该类。

### 步骤 3。在类中添加方法

将指针放置在“代码”窗口内，然后输入以下文本：

```
Public Function FV( _  
    Rate As Double, _
```

```

nPeriods As Long, _
Payment As Double, _
PresentValue As Double, _
PaymentType As Long _
) As Double
Dim v As Double

v = (1 + Rate) ^ nPeriods

FV = -( _
    (PresentValue * v) + _
    (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate) _
)
End Function

```

该 Visual Basic FV 函数执行的操作与 [“使用 Visual Basic 开发 COM 过程” 页面上 153](#) 中的 C++ FV 函数相同。

## 步骤 4。构建项目

要构建项目，请执行以下操作：

1. 从“文件”菜单中，选择“设置 COM\_VB\_BankSoft.DLL”。对话框会提示您提供文件位置。
2. 输入文件位置并单击“确定”。

Visual Basic 会编译源代码并在您指定的位置创建 COM\_VB\_BankSoft.DLL。还会在本地注册表中注册类 COM\_VB\_BankSoft.BSoftFin。

注册组件后，该主机上运行的集成服务便可访问该组件。

# 开发 Informatica 外部过程

可以创建在 32 位或 64 位集成服务计算机上运行的外部过程。完成以下创建 Informatica 样式外部过程的步骤：

1. 在 Transformation Developer 中创建外部过程转换。  
外部过程转换可定义过程签名。端口名称、数据类型和端口类型（输入或输出）必须与外部过程的签名匹配。
2. 生成外部过程的模板代码。  
执行此命令时，Designer 将使用外部过程转换中的信息创建多个 C++ 源代码文件和生成文件。其中一个源代码文件包含的“存根”可用于签名在转换中定义的函数。
3. 修改代码，以添加过程逻辑。使用实施填充存根，并使用 C++ 编译器编译源代码文件，并将其链接至动态链接库或共享库。  
当集成服务发现与 Informatica 过程绑定的外部过程转换时，它将加载 DLL 或共享库，并调用定义的外部过程。
4. 构建库，并将其复制到集成服务计算机。
5. 使用外部过程转换创建映射。
6. 在工作流中运行会话。

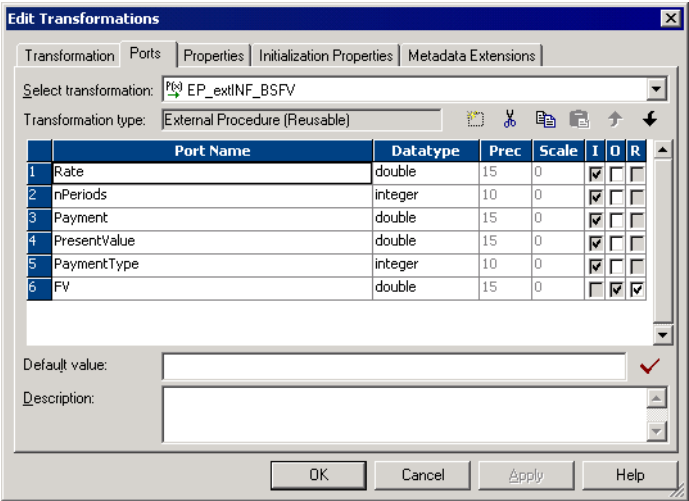
BankSoft 示例阐述了如何实施此功能。

# 步骤 1。创建外部过程转换

- 1. 打开 Transformation Developer 并创建外部过程转换。
- 2. 打开转换并输入该转换的名称。  
在 BankSoft 示例中，输入 EP\_extINF\_BSFV。
- 3. 在“端口”选项卡上，为传递到您计划定义的过程的每个参数定义端口。  
请确保使用正确的数据类型。  
下表介绍了这些端口：

端口名称	数据类型	精度	小数位数	输入/输出	可重用
比率	双精度	15	0	输入	否
nPeriods	整型	10	0	输入	否
付款	双精度	15	0	输入	否
PresentValue	双精度	15	0	输入	否
PaymentType	整型	10	0	输入	否
FV	双精度	15	0	输出	是

以下 BankSoft 示例显示了外部过程转换的示例：



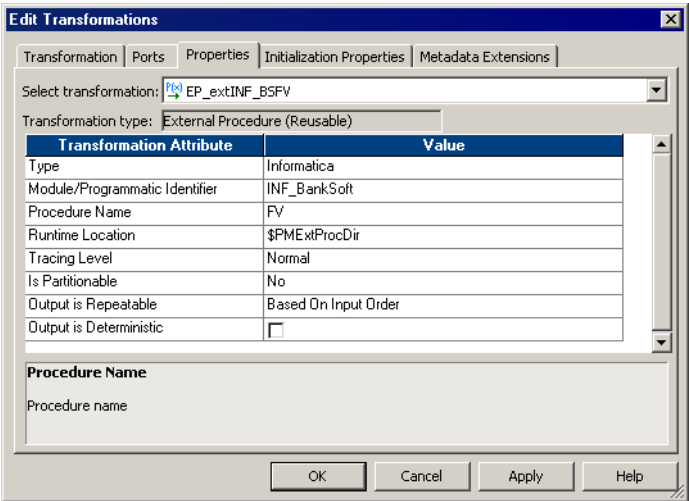
最后一个端口 FV 可从过程中捕获返回值。

- 4. 选择“属性”选项卡并将该过程配置为一个 Informatica 过程。

在 BankSoft 示例中，配置以下属性：

转换属性	值
类型	Informatica
模块/编程标识符	INF_BankSoft
过程名称	FV
运行时位置	\$PMEExtProcDir
跟踪级别	普通
是否可分区	否
输出是可重复的	基于输入顺序
输出具有确定性	否

以下 BankSoft 示例显示了 Informatica 过程的示例：



**注意:** 模块/编程标识符：

下表介绍模块名称如何确定各个平台上的 DLL 名称或共享对象名称：

操作系统	模块标识符	库文件名
Windows	INF_BankSoft	INF_BankSoft.DLL
AIX	INF_BankSoft	libINF_BankSoftshr.a
HPUX	INF_BankSoft	libINF_BankSoft.sl
Linux	INF_BankSoft	libINF_BankSoft.so
Solaris	INF_BankSoft	libINF_BankSoft.so.1

5. 单击“确定”。

创建调用过程的外部过程转换后，下一步要生成 C++ 文件。

## 步骤 2. 生成 C++ 文件

创建外部过程转换后，生成该代码。由于在 UNIX 映射的驱动器上创建的文件始终以小写字母表示，因此 Designer 会生成以小写字母表示的文件名。以下规则适用于生成的文件：

- **文件名。**前缀“tx”用于 TX 模块文件。
- **模块类名称。**生成的代码具有针对包含 TX 过程的模块的类声明。前缀 *Tx* 用于 TX 模块类。例如，如果外部过程转换具有模块名称 Mymod，则类名称为 TxMymod。

要为外部过程生成代码，请执行以下操作：

1. 选择转换并单击“转换”>“生成代码”。
2. 选中您刚创建的过程名称旁边的复选框。  
在 BankSoft 示例中，选择 *INF\_BankSoft.FV*。
3. 指定要生成文件所在的目录，然后单击“生成”。

Designer 会在您指定的目录中创建子目录 INF\_BankSoft。

在 Designer 中创建的每个外部过程转换都必须指定模块和过程名称。Designer 会为共享通用模块名称的所有转换在单一目录中生成代码。在一个目录中构建代码会创建单一共享库。

Designer 将生成以下文件：

- **tx<moduleName>.h.**定义外部过程模块类。该类派生自基类 TINFExternalModule60。在生成的代码中未为该类定义任何数据成员。但是，可以在此添加新数据成员和方法。
- **tx<moduleName>.cpp.**实现外部过程模块类。可以扩展 InitDerived() 方法以包含您添加的任何新数据成员的初始化。集成服务仅在成功完成基类 Init() 方法后调用衍生类 InitDerived() 方法。

该文件定义模块中所有外部过程转换的签名。这些签名的任何更改都会导致与 Designer 中定义的外部过程转换不一致。因此，不得更改这些签名。

该文件还包含 C 函数 CreateExternalModuleObject，该函数使用在该文件中定义的构造函数创建外部过程模块类的对象。集成服务会调用 CreateExternalModuleObject 而非直接调用该构造函数。

- **<procedureName>.cpp.**Designer 会在该模块中为每个外部过程生成上述文件之一。该文件包含实现过程逻辑的代码，例如数据清理和筛选。对于数据清理，创建代码以从输入端口读入值并为输出端口生成值。对于筛选，创建代码以通过返回 INF\_NO\_OUTPUT\_ROW 抑制输出行的生成。
- **stdafx.h.**用于在 UNIX 系统上进行构建的存根文件。各种 \*.cpp 文件都包含该文件。在 Windows 系统上，Visual Studio 会生成 stdafx.h 文件，应使用该文件而非 Designer 生成的文件。
- **version.cpp.**这是一个小文件，其中承载该实现的版本号。在早期版本中，以不同的方式处理外部过程实现。该文件允许集成服务确定外部过程模块的版本。
- **makefile.aix、makefile.aix64、makefile.hp、makefile.hp64、makefile.hpparisc64、makefile.linux、makefile.sol.**让文件可用于 UNIX 平台。将 makefile.aix、makefile.hp、makefile.linux 和 makefile.sol 用于 32 位平台。将 makefile.aix64 用于 64 位 AIX 平台，将 makefile.hp64 用于 64 位 HP-UX (Itanium) 平台。

### 示例 1

在 BankSoft 示例中，Designer 将生成以下文件：

- **txinf\_banksoft.h.**包含模块类 TxINF\_BankSoft 和外部过程 FV 的声明。
- **txinf\_banksoft.cpp.**包含模块类 TxINF\_BankSoft 的代码。
- **fv.cpp.**包含过程 FV 的代码。
- **version.cpp.**返回 TX 版本。

- **stdafx.h.**在 UNIX 中编译的必需项。在 Windows 中，stdafx.h 由 Visual Studio 生成。
- **readme.txt.**包含常规帮助信息。

## 示例 2

如果使用名为“Myproc1”和“Myproc2”的过程创建两个均包含 Mymod 模块的外部过程转换，Designer 将生成以下文件：

- **txmymod.h.** 包含模块类 TxMymod 及外部过程 Myproc1 和 Myproc2 的声明。
- **txmymod.cpp.** 包含模块类 TxMymod 的代码。
- **myproc1.cpp.** 包含过程 Myproc1 的代码。
- **myproc2.cpp.** 包含过程 Myproc2 的代码。
- **version.cpp.**
- **stdafx.h.**
- **readme.txt.**

## 步骤 3。使用实施填充方法存根

最后一步是对过程编码。

1. 打开为过程生成的 *<Procedure\_Name>.cpp* 存根文件。

在 BankSoft 示例中，打开 fv.cpp 以对 TxINF\_BankSoft::FV 过程编码。

2. 为过程输入 C++ 代码。

以下代码实施 FV 过程：

```
INF_RESULT TxINF_BankSoft::FV()
{
    // Input port values are mapped to the m_pInParamVector array in
    // the InitParams method. Use m_pInParamVector[i].IsValid() to check
    // if they are valid. Use m_pInParamVector[i].GetLong or GetDouble,
    // etc. to get their value. Generate output data into m_pOutParamVector.
    // TODO: Fill in implementation of the FV method here.
    ostringstream ss;
    char* s;
    INF_BOOLEAN bVal;
    double v;
    TINFPParam* Rate = &m_pInParamVector[0];
    TINFPParam* nPeriods = &m_pInParamVector[1];
    TINFPParam* Payment = &m_pInParamVector[2];
    TINFPParam* PresentValue = &m_pInParamVector[3];
    TINFPParam* PaymentType = &m_pInParamVector[4];
    TINFPParam* FV = &m_pOutParamVector[0];
    bVal =
        INF_BOOLEAN(
            Rate->IsValid() &&
            nPeriods->IsValid() &&
            Payment->IsValid() &&
            PresentValue->IsValid() &&
            PaymentType->IsValid()
        );
    if (bVal == INF_FALSE)
    {
        FV->SetIndicator(INF_SQL_DATA_NULL);
        return INF_SUCCESS;
    }
}
```

```

v = pow((1 + Rate->GetDouble()), (double)nPeriods->GetLong());
FV->SetDouble(
    -(
        (PresentValue->GetDouble() * v) +
        (Payment->GetDouble() *
            (1 + (Rate->GetDouble() * PaymentType->GetLong())) *
            ((v - 1) / Rate->GetDouble())
        )
    );
ss << "The calculated future value is: " << FV->GetDouble() <<ends;
s = ss.str();
(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);
(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);
delete [] s;
return INF_SUCCESS;
}

```

Designer 会生成函数配置文件，包括参数和返回值。需要在函数内输入实际代码，如注释中所示。由于您已引用 POW 函数并定义了 ostream 变量，因此还必须包含以下预处理器语句：

在 Windows 中：

```

#include <math.h>
#include <sstream> using namespace std;

```

在 UNIX 中，包含语句如下：

```

#include <math.h>
#include <sstream.h>

```

3. 保存修改的文件。

## 步骤 4。构建模块

在 Windows 上，使用 Visual C++ 编译 DLL。

### 在 Windows 上构建模块

要在 Windows 上构建 DLL，请执行以下操作：

1. 启动 Visual C++。
2. 单击“文件” > “新建”。
3. 在“新建”对话框中，单击“项目”选项卡，然后选择“MFC AppWizard (DLL)”选项。
4. 输入其位置。

在 BankSoft 示例中，输入 *c:\pmclient\tx\INF\_BankSoft*，假定在 *c:\pmclient\tx* 中生成文件。

5. 输入项目的名称。

其必须与为外部过程转换输入的模块名称相同。在 BankSoft 示例中，其为 *INF\_BankSoft*。

6. 单击“确定”。

Visual C++ 现在通过向导指导您逐步进行操作，定义项目的所有组件。

7. 在该向导中，单击“MFC Extension DLL”（使用共享的 MFC DLL）。

8. 单击“完成”。

该向导会生成多个文件。

9. 单击“项目” > “添加至项目” > “文件”。

10. 向上导航一个目录级别。该目录包含您创建的外部过程文件。选择所有 .cpp 文件。

在 BankSoft 示例中，添加以下文件：

- fv.cpp

- txinf\_banksoft.cpp
  - version.cpp
11. 单击“项目” > “设置”。
  12. 单击 C/C++ 选项卡，从“类别”字段中选择“预处理器”。
  13. 在“附加包括目录”字段中，输入 `..; <pmserver install dir>\extproc\include`。
  14. 单击“链接”选项卡，然后从“类别”字段中选择“常规”。
  15. 在“对象/库模块”字段中，输入 `<pmserver install dir>\bin\pmtx.lib`。
  16. 单击“确定”。
  17. 单击“构建” > “构建 INF\_BankSoft.dll”，或按 F7 构建项目。
- 编译器现在会创建 DLL，并将其放在项目目录下的调试或发行目录中。

### 在 UNIX 上构建模块

要在 UNIX 中构建共享的库，请执行以下操作：

1. 如果无法直接访问 PowerCenter 客户端工具，请将共享库所需的所有文件复制到计划执行构建的 UNIX 计算机。  
例如，在 BankSoft 过程中，使用 FTP 或另一种机制将 INF\_BankSoft 目录中的所有内容复制到 UNIX 计算机。
2. 将环境变量 INFA\_HOME 设置为 PowerCenter 安装目录。  
**警告：** 如果您为 INFA\_HOME 环境变量指定了不正确的目录路径，则集成服务无法启动。
3. 输入命令以创建项目。  
该命令取决于 UNIX 的版本，汇总如下：

UNIX 版本	命令
AIX (32 位)	<code>make -f makefile.aix</code>
AIX (64 位)	<code>make -f makefile.aix64</code>
Linux	<code>make -f makefile.linux</code>
Solaris	<code>make -f makefile.sol</code>

### 步骤 5. 创建映射

在 Mapping Designer 中，创建使用该外部过程转换的映射。

### 步骤 6. 运行会话

运行会话时，集成服务会查找指定为运行时位置的目录，以查找您在步骤 4 中构建的库 (DLL)。会话属性中“运行时位置”属性的默认值为 `$PMExtProcDir`。

要运行会话，请执行以下操作：

1. 在 Workflow Manager 中，创建一个工作流。
  2. 在工作流中为此映射创建一个会话。
- 提示：** 或者，可以在任务开发程序中创建可重用会话并在工作流中使用。



3. 将库 (DLL) 复制到运行时位置目录。
4. 运行包含会话的工作流。

## 在 Windows 上使用模块的调试版本运行会话

在 Windows 上, Informatica PowerCenter 附带有外部过程转换库的发行内部版本 (pmtx.dll) 和调试内部版本 (pmtxdbg.dll)。这些库安装在服务器 bin 目录中。

如果在步骤 4 中构建模块的发行版本, 请在工作流中运行该会话以使用外部过程转换库的发行内部版本 (pmtx.dll)。您不需要完成以下任务。

如果在步骤 4 中构建模块的调试版本, 请按照下面的过程使用外部过程转换库的调试内部版本 (pmtxdbg.dll)。

要使用模块的调试版本运行会话, 请执行以下操作:

1. 在 Workflow Manager 中, 创建一个工作流。
2. 在工作流中为此映射创建一个会话。  
还可以在任务开发程序中创建可重用会话并在工作流中使用。
3. 将库 (DLL) 复制到运行时位置目录。
4. 要使用外部过程转换库的调试内部版本, 请执行以下操作:
  - 通过重命名 pmtx.dll 或从服务器 bin 目录中移动该文件来保留该文件。
  - 将 pmtxdbg.dll 重命名为 pmtx.dll。
5. 运行包含会话的工作流。
6. 要将外部过程转换库的发行内部版本还原回默认库, 请执行以下操作:
  - 将 pmtx.dll 重命名回 pmtxdbg.dll。
  - 将原始 pmtx.dll 文件返回/重命名到服务器 bin 目录。

**注意:** 如果在 Windows 上通过模块的调试版本运行包含该会话的工作流, 则必须将原始 pmtx.dll 文件返回为其原始名称和位置, 然后才能运行非调试会话。

## 分发外部过程

假定您开发了一系列外部过程, 并要将这些过程分发到多个运行集成服务的服务器上。具体分发方法需要根据外部过程的类型和构建这些过程使用的操作系统来判断。

您也可以使用这些步骤将外部过程分发给外部客户。

## 分发 COM 过程

构建项目时, Visual Basic 和 Visual C++ 会将 COM 类注册到本地注册表中。注册之后, 在编译 DLL 的计算机上运行的集成服务可访问这些类。例如, 如果您在 HOST1 上构建项目, 该项目中的所有类均将注册到 HOST1 注册表中, 并可供 HOST1 上运行的集成服务访问。假设, 您希望在 HOST2 上运行的集成服务也可以访问这些类。为此, 您必须在 HOST2 注册表中注册这些类。

Visual Basic 具有一种用于创建设置程序的实用程序, 这类设置程序可在 Windows 计算机上安装 COM 类, 并将这些类注册到该计算机的注册表中。Visual C++ 中没有此类实用程序, 不过, 您可以非常轻松自主地注册类。

## 分发 COM Visual Basic 过程

分发 COM Visual Basic 过程：

1. 构建 DLL 之后，退出 Visual Basic 并启动 Visual Basic Application Setup 向导。
2. 跳过向导的第一个面板。
3. 在第二个面板上，指定项目的位置，然后选择 *创建设置程序* 选项。
4. 在第三个面板上，选择计划使用的分发方法。
5. 在接下来的面板中，指定写入设置文件的目标目录。  
对于简单的 ActiveX 组件，可以继续浏览到向导的最后一个面板。否则，根据文件的类型和分发方法，您可能需要添加更多信息。
6. 单击最后一个面板上的“完成”。

Visual Basic 此时已为 DLL 创建设置程序。在运行集成服务的任意 Windows 计算机上运行此设置程序。

## 手动分发 COM Visual Basic 过程

手动分发 COM Visual C++/Visual Basic 过程：

1. 将 DLL 复制到新计算机上的任意保存目录。
2. 登录此 Windows 计算机并打开 DOS 提示符。
3. 导航到包含 DLL 的目录，并执行以下命令：

```
REGSVR32 project_name.DLL
```

*project\_name* 是所创建 DLL 的名称。在 BankSoft 示例中，项目名称为 *COM\_VC\_BankSoft.DLL*。或 *COM\_VB\_BankSoft.DLL*。

此命令行程序随之会注册 DLL 及其包含的任何 COM 类。

## 分发 Informatica 模块

可以在多个存储库中分发外部过程。

在多个存储库中分发外部过程：

1. 将包含外部过程的 DLL 或共享对象移动至计算机上集成服务可以访问的目录。
2. 使用 Designer 客户端工具将外部过程转换从原存储库复制到目标存储库。  
您也可以将外部过程转换导出到 XML 文件，并将其导入目标存储库。

# 开发说明

本部分介绍有关开发 COM 和 Informatica 外部过程的一些其他准则和信息。

## COM 数据类型

使用 Visual C++ 或 Visual Basic 开发 COM 过程时，需要使用与集成服务读取和转换数据时使用的内部数据类型对应的 COM 数据类型。当集成服务尝试在外部过程转换的端口之间和该转换所调用过程中的参数（或返回值）之间映射数据类型时，这些数据类型匹配非常重要。

下表对比了 Visual C++ 和转换数据类型：

Visual C++ COM 数据类型	转换数据类型
VT_I4	整型
VT_UI4	整型
VT_R8	双精度
VT_BSTR	字符串
VT_DECIMAL	小数
VT_DATE	日期/时间

下表对比了 Visual Basic 和转换数据类型：

Visual Basic COM 数据类型	转换数据类型
长整型	整型
双精度	双精度
字符串	字符串
小数	小数
日期	日期/时间

如果数据类型未能正确匹配，集成服务可能会尝试进行转换。例如，如果您将整数数据类型分配给端口，而对应参数的数据类型为 BSTR，则集成服务将尝试将整数值转换为 BSTR。

## 行级过程

所有外部过程转换都将使用来自传递转换的某一行的值来调用过程。不能在一个过程调用中使用来自多行的值。例如，不能将汇总函数 SUM 或 AVG 的等价函数编码到过程调用中。从这个意义上讲，所有外部过程都必须无状态。

## 过程返回值

调用过程时，集成服务除捕获您输入到过程的所有返回值之外，还捕获一个附加返回值。该附加值指示集成服务是否已成功调用过程。

对于 COM 过程，该返回值使用类型 HRESULT。

Informatica 过程使用类型 INF\_RESULT。如果返回的值为 S\_OK/INF\_SUCCESS，则说明集成服务成功调用过程。必须返回正确的值来指示外部过程成功与否。Informatica 过程返回四个值：

- **INF\_SUCCESS.**外部过程已成功处理该行。集成服务会将该行传递到映射中的下一转换。
- **INF\_NO\_OUTPUT\_ROW.**由于外部过程逻辑，集成服务不会写入当前行。这不是错误。使用 INF\_NO\_OUTPUT\_ROW 筛选行时，外部过程转换的行为类似于筛选器转换。

**注意：**在外部过程中使用 INF\_NO\_OUTPUT\_ROW 时，请确保已将外部过程转换连接到只从外部过程转换接收行的其他转换。

- **INF\_ROW\_ERROR.**等同于转换错误。集成服务会丢弃当前行，但可以处理下一行，除非将会话配置为在  $n$  个错误后停止。
- **INF\_FATAL\_ERROR.**等同于 `ABORT()` 函数调用。集成服务会中止会话且不再处理任何行。

## 过程调用中的异常

通过外部过程转换调用 COM 或 Informatica 过程时，集成服务可捕获大部分异常。例如，如果过程调用产生除数为零错误，集成服务会捕获这种异常。

有时候，集成服务无法捕获过程调用生成的错误。由于集成服务仅支持进程内 COM 服务器，并且由于所有 Informatica 过程均存储在共享库和 DLL 中，运行外部过程的代码在内存中与集成服务同处于一个地址空间。因此，可能会出现外部过程代码覆盖集成服务内存的情况，导致集成服务停止。如果 COM 或 Informatica 过程导致此类停止，请检查源代码是否存在内存访问问题。

## 过程内存管理

Informatica 过程所用的任何数据类型均为固定长度，因此，Informatica 外部过程不存在内存管理问题。对于 COM 过程，只有当过程中的 [out] 参数使用 BSTR 数据类型时才需要分配内存。在这种情况下，您需要为每一个过程调用分配内存。在会话过程中，集成服务会在调用函数后释放内存。

## 已有 C/C++ 库或 VB 函数的包装类

假设 BankSoft 包含由 C 或 C++ 函数组成的库，并且想将这些函数插入到集成服务中。尤其是，该库还包含 BankSoft 自己实施的 FV 函数，名为 PreExistingFV。完成此任务的常规方法与用于 COM 和 Informatica 外部过程的方法相同。Visual Basic 中提供了相似的解决方案。只需调用已有的 Visual Basic 函数即可，或者针对 Visual Basic 可访问的对象调用相应方法。

## 生成错误和跟踪消息

[“步骤 4. 构建模块” 页面上 159](#) 中的 Informatica 外部过程 TxINF\_BankSoft::FV 的实施包含以下代码行：

```
ostream ss;

char* s;

...

ss << "The calculated future value is: " << FV->GetDouble() << ends;

s = ss.str();

(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);

(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);

delete [] s;
```

当集成服务创建 Tx<MODNAME> 类型的对象时，它会将回调函数指针传递给构造函数，而利用该回调函数可将错误或调试消息写入会话日志。（Tx<MODNAME> 构造函数的代码位于 Tx<MODNAME>.cpp 文件中。）此指针存储在 Tx<MODNAME> 成员变量 m\_pfnMessageCallback 中。指针的类型在文件 \$PMExtProcDir/include/infemsg.h: 的 typedef 中定义：

```
typedef void (*PFN_MESSAGE_CALLBACK)(

    enum E_MSG_TYPE eMsgType,

    unsigned long Code,

    char* Message
```

```
);
```

该文件同时还定义了 E\_MSG\_TYPE:

```
enum E_MSG_TYPE {  
  
    E_MSG_TYPE_LOG = 0,  
  
    E_MSG_TYPE_WARNING,  
  
    E_MSG_TYPE_ERR  
  
};
```

如果您将回调函数的 eMsgType 指定为 E\_MSG\_TYPE\_LOG, 回调函数会在会话日志中写入 *日志* 消息。如果指定 E\_MSG\_TYPE\_ERR, 回调函数会在会话日志中写入 *错误* 消息。如果指定 E\_MSG\_TYPE\_WARNING, 回调函数会在会话日志中写入 *警告* 消息。使用这些消息可在 Informatica 外部过程中提供简单的调试功能。

要调试 COM 外部过程, 您可以使用 Visual Basic 或 C++ 类内部可用的输出功能。例如, 在 Visual Basic 中使用 MsgBox 打印出每行的计算结果。当然, 调试过程中只能在少量示例数据上执行此操作, 并确保在投产前移除 MsgBox。

**注意:** 尝试从 Visual Basic 或 C++ 类内部使用任何输出功能之前, 必须在注册表中添加以下值:

1. 在 Windows 注册表中添加以下条目:

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo  
\RunInDebugMode=Yes
```

此选项将以常规应用程序 (非服务) 形式启动集成服务。在集成服务运行过程中, 可以在不更改集成服务的调试特权的情况下进行调试。

2. 使用命令 PMSERVER.EXE 从命令行启动集成服务。

集成服务当前在调试模式下运行。

调试完成后, 确保从注册表中删除此条目或将 “RunInDebugMode” 设为 “否”。否则, 尝试以服务形式启动 PowerCenter 时将无法启动。

1. 停止集成服务, 并将之前添加的注册表条目改为以下设置:

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=No
```

2. 以 Windows 服务形式重新启动集成服务。

## TINFPParam 类和指示器

<PROCNAME> 方法使用两个参数数组访问输入和输出参, 而每个数组元素的数据类型均为 TINFPParam。TINFPParam 数据类型是一种 C++ 类, 用作 “变量” 数据结构, 可以容纳任何 Informatica 内部数据类型。可以通过表单 Get<Type> 和 Set<Type> (其中 <Type> 是一种 Informatica 内部数据类型) 的成员函数访问 TINFPParam\* 类型参数中的实际数据。TINFPParam 还包含用于为每个参数获取和设置指示器的方法。

您负责在进入外部过程时检查这些指示器, 以及在退出时设置这些指示器。在进入时, 所有输出参数的指示器将被显式设置为 INF\_SQL\_DATA\_NULL, 因此, 如果您在从外部过程返回之前没有重置这些指示器, 则对于所有输出参数, 只能获得空值。TINFPParam 类还支持用于为特定参数获取元数据的函数。有关 TINFPParam 类的所有成员函数的完整说明, 请参阅 tx/include 目录中的 infemdef.h include 文件。

Informatica 外部过程相对于 COM 外部过程的主要优势之一是 Informatica 外部过程直接支持指示器操作。也就是说, 可以检查输入参数, 以查看其是否为空, 还可以将输出参数设置为空。COM 未提供指示器支持。因此, 如果进入 COM 样式外部过程的行包含任何空值, 将无法处理该行。可以使用 Designer 中的默认值设备来克服这个缺点。不过, 无法从 COM 函数向外传递空值。

## 未连接的外部过程转换

将外部过程转换的实例添加到映射时，可以选择将其作为管道的一部分进行连接，也可以将其保留为未连接。某行每次传递转换时，已连接的外部过程转换都会调用 COM 或 Informatica 过程。

要从未连接的外部过程转换获取返回值，请使用以下语法在表达式中调用它：

```
:EXT.transformation_name(arguments)
```

如果某行传递包含表达式的转换，则集成服务会调用与外部过程转换关联的过程。表达式通过外部过程转换返回端口（应选中“结果 (R)”选项）捕获过程的返回值。

## 初始化 COM 和 Informatica 模块

一些外部过程必须在初始化时配置。根据外部过程的类型，此初始化可能会表现出两种形式：

- **Informatica 样式外部过程初始化。** 包含外部过程的 Tx<MODNAME> 类还包含初始化函数 Tx<MODNAME>::InitDerived。集成服务非常熟悉该初始化函数的签名，它包含三个参数：

- **nInitProps.** 通过此参数，初始化函数可得知即将传入的初始化属性的数量。
- **属性。** 此参数是表示初始化属性名称的 nInitProp 字符串阵列。
- **Values.** 此参数是表示初始化属性的值的 nInitProp 字符串阵列。

集成服务首先调用基类中的 Init() 函数。成功完成 Init() 函数后，基类将调用 Tx<MODNAME>::InitDerived() 函数。

集成服务将创建 Tx<MODNAME> 对象，然后调用初始化函数。外部过程开发人员必须提供 Tx<MODNAME>::InitDerived() 函数中解释初始化属性，并用来对外部过程初始化的部分。对象创建和初始化完成后，集成服务可对每一行中的对象调用外部过程。

- **COM 样式外部过程初始化。** 包含外部过程（或 EP 对象）的对象不包含初始化函数。相反，其他对象（CF 对象）将充当 EP 对象的类工厂。CF 对象具有可创建 EP 对象的方法。

CF 对象方法的签名从类型库确定。集成服务创建 CF 对象，然后在该对象上调用来创建 EP 对象，从而将此方法传递至所需的任何参数。这就要求方法的签名中包含类型可从类型库确定的输入参数组，以及一个 IUnknown\*\* 或 IDispatch\*\* 或指向 IUnknown\* 或 IDispatch\* 的 VARIANT\* 的输出参数。

输入参数具有初始化 EP 对象所需的值，输出参数则接收经过初始化的对象。输出参数的属性可以是 [out]，也可以是 [out, retval]。也即，经过初始化的对象可作为输出参数返回，也可以作为方法的返回值返回。支持的输入参数数据类型包括：

- COM VC 类型
- VT\_UI1
- VT\_BOOL
- VT\_I2
- VT\_UI2
- VT\_I4
- VT\_UI4
- VT\_R4
- VT\_R8
- VT\_BSTR
- VT\_CY
- VT\_DATE

## 在 Designer 中设置初始化属性

在“编辑转换”对话框的“初始化属性”选项卡上输入外部过程初始化属性。该选项卡会显示不同的字段，具体取决于外部过程是 COM 样式还是 Informatica 样式。

COM 样式的外部过程转换在“初始化属性”选项卡上包含以下字段：

- **Class Factory 的编程标识符。** 输入 Class Factory 的编程标识符。
- **构造函数。** 指定创建 EP 对象的 Class Factory 方法。

可以输入任意数量的初始化属性，以传递到 COM 样式和 Informatica 样式外部过程转换的构造函数方法。

要添加新初始化属性，请单击“添加”按钮。在“属性”列中输入参数名称，在“值”列中输入参数值。例如，您可以输入以下参数：

参数	值
Param1	abc
Param2	100
Param3	3.17

**注意：**必须在您于 Designer 中定义的初始化属性与 Class Factory 构造函数方法的输入参数之间创建一对一关系。例如，如果构造函数具有  $n$  个参数且最后一个参数作为接收初始化对象的输出参数，则必须在 Designer 中定义  $n - 1$  个初始化属性，构造函数方法中的每个输入参数对应一个。

还可以在初始化属性中使用进程变量。

## TX 中分发和使用的其他文件

下文是路径 `$PMExtProcDir/include` 下包含的表头文件，编译外部过程需要使用这些文件：

- `infconfig.h`
- `infem60.h`
- `infemdef.h`
- `infemmsg.h`
- `infparam.h`
- `infsigtr.h`

下文是路径 `<PMInstallDir>` 下包含的库文件，链接外部过程和运行会话时需要使用这些文件：

- `libpmtx.a` (AIX)
- `libpmtx.so` (Linux)
- `libpmtx.so` (Solaris)
- `pmtx.dll` 和 `pmtx.lib` (Windows)

# 初始化属性中的服务进程变量

PowerCenter 支持外部过程转换初始化属性列表中的内置进程变量。如果属性值包含内置流程变量，集成服务将展开它们，然后再将它们传递到外部过程库中。这对写入可移植性外部过程转换可能非常有用。

下表介绍了外部过程转换在转换属性的“初始化属性”选项卡上的初始化属性和值。

表 2. 外部过程初始化属性

属性	值	传递到外部过程库中的展开值
mytempdir	\$PMTempDir	/tmp
memorysize	5000000	5000000
input_file	\$PMSourceFileDir/file.in	/data/input/file.in
output_file	\$PMTargetFileDir/file.out	/data/output/file.out
extra_var	\$some_other_variable	\$some_other_variable

在运行工作流时，集成服务会展开属性列表，并将其传递到外部过程初始化函数中。假设内置进程变量 \$PMTempDir 为 `/tmp`，\$PMSourceFileDir 为 `/data/input`，\$PMTargetFileDir 为 `/data/output`，[“初始化属性中的服务进程变量” 页面上 168](#) 中的最后一列将包含属性和展开值信息。集成服务不会展开最后一项属性“\$some\_other\_variable”，因为它不是内置进程变量。

## 外部过程接口

集成服务将以下主要函数与外部过程结合使用：

- 分派
- 外部过程
- 属性访问
- 参数访问
- 代码页访问
- 转换名称访问
- 过程访问
- 分区相关
- 跟踪级别

### 分派函数

集成服务调用分派函数，以将每个输入行传递至外部过程模块。分派函数反过来会调用指定的外部过程函数。

外部过程会使用针对输入端口的成员变量 `m_pInParamVector` 和针对输出端口的变量 `m_pOutParamVector` 直接访问转换中的端口。



## 签名

分派函数具有包含一个索引参数的固定签名。

```
virtual INF_RESULT Dispatch(unsigned long ProcedureIndex) = 0
```

## 外部过程函数

外部过程函数是外部过程模块的主入口点，也是外部过程转换的一个属性。分派函数会对每一个输入行调用外部过程函数。对于外部过程转换，可对外部过程模块的输入和输出使用外部过程函数。此函数可访问每一个输入行的 IN 和 IN-OUT 端口值，并可设置 IN 和 IN-OUT 端口值。外部过程函数包含所有输入和输出处理逻辑。

## 签名

外部过程函数没有参数。输入参数数组已传递 InitParams() 方法并存储在成员变量 m\_pInParamVector 中。数组中的每个条目按相同的顺序与外部过程转换的对应 IN 和 IN-OUT 端口匹配。集成服务会先填充该向量，然后再调用分派函数。

先使用成员变量 m\_pOutParamVector 传递输出行，然后再返回 Dispatch() 函数。

对于 MyExternal 过程转换，外部过程函数如下，其中输入参数位于成员变量 m\_pInParamVector 中，输出值位于成员变量 m\_pOutParamVector 中：

```
INF_RESULT Tx<ModuleName>::MyFunc()
```

## 属性访问函数

属性访问函数可提供与外部过程转换关联的初始化属性的信息。编辑外部过程转换时，初始化属性名称和值将显示在“初始化属性”选项卡上。

Informatica 可在基类和 TINFConfigEntriesList 类中同时提供访问函数。可分别使用 TINFConfigEntriesList 类中的 GetConfigEntryName() 和 GetConfigEntryValue() 函数访问初始化属性名称和值。

## 签名

Informatica 在基类中提供了以下函数：

```
TINFConfigEntriesList* TINFBaseExternalModule60::accessConfigEntriesList();  
const char* GetConfigEntry(const char* LHS);
```

Informatica 在 TINFConfigEntriesList 类中提供了以下函数：

```
const char* TINFConfigEntriesList::GetConfigEntryValue(const char* LHS);  
const char* TINFConfigEntriesList::GetConfigEntryValue(int i);  
const char* TINFConfigEntriesList::GetConfigEntryName(int i);  
const char* TINFConfigEntriesList::GetConfigEntry(const char* LHS)
```

**注意：**在 TINFConfigEntriesList 类中，使用 GetConfigEntryName() 和 GetConfigEntryValue() 属性访问函数可以访问初始化属性名称和值。

可以通过 TX 程序调用这些函数。TX 程序随后会转换此字符串值，例如使用 atoi 或 sscanf 转换为数字。在以下示例中，“addFactor”是一项初始化属性。accessConfigEntriesList() 是 TX 基类的成员变量，因此无需定义。

```
const char* addFactorStr = accessConfigEntriesList()-> GetConfigEntryValue("addFactor");
```

## 参数访问函数

参数访问函数与数据类型关联。使用参数访问函数 GetDataType 可返回参数的数据类型。然后，使用与此数据类型对应的参数访问函数可返回参数的相关信息。

传递至外部过程的参数的数据类型为 TINFParam\*。表头文件 infparam.h 定义了相关访问函数。Designer 可生成包含指示参数数据类型的注释的存根代码。您还可以在 Designer 的对应外部过程转换中确定参数的数据类型。

签名

传递到外部过程的参数是一个指针，指向一个 TINFParam 类的对象。此固定签名函数是该类的一种方法，它将返回参数数据类型作为枚举值。

有效数据类型包括：

- INF\_DATATYPE\_LONG
- INF\_DATATYPE\_STRING
- INF\_DATATYPE\_DOUBLE
- INF\_DATATYPE\_RAW
- INF\_DATATYPE\_TIME

下表介绍了一些参数访问函数：

参数访问函数	说明
INF_DATATYPE GetDataType(void);	获取参数的数据类型。可以使用参数数据类型来确定在访问参数值时，使用哪中数据类型特定的函数。
INF_BOOLEAN IsValid(void);	验证输入数据是否有效。如果参数包含被截断的字符串或原始数据类型的数据，则返回 FALSE。当输入数据超出精度，或者输入数据为空值时，也会返回 FALSE。
INF_BOOLEAN IsNULL(void);	验证输入数据是否为空。
INF_BOOLEAN IsInputMapped (void);	验证向此参数传递数据的输入端口是否已连接到某一转换。
INF_BOOLEAN IsOutput Mapped (void);	验证从此参数接收数据的输出端口是否已连接到某一转换。
INF_BOOLEAN IsInput(void);	验证参数是否与某一输入端口对应。
INF_BOOLEAN IsOutput(void);	验证参数是否与某一输出端口对应。
INF_BOOLEAN GetName(void);	获取参数的名称。
SQLIndicator GetIndicator(void);	获取参数指示器的值。IsValid 和 IsNULL 函数是此函数的特殊情况。此函数还有可能返回 INF_SQL_DATA_TRUNCATED。
void SetIndicator(SQLIndicator Indicator);	设置输出参数指示器，如无效或已截断。
long GetLong(void);	获取具有“长整型”或“整型”数据类型的参数的值。仅当已知参数数据类型为“整型”或“长整型”时，才能调用此函数。此函数无法将数据从其他数据类型转换为“长整型”。
double GetDouble(void);	获取具有“浮点型”或“双精度型”数据类型的参数的值。仅当已知参数数据类型为“浮点型”或“双精度型”时，才能调用此函数。此函数无法将数据从其他数据类型转换为“双精度型”。

参数访问函数	说明
char* GetString(void);	获取参数的值，作为以空值终止的字符串。仅当已知参数数据类型为“字符串型”时，才能调用此函数。此函数无法将数据从其他数据类型转换为“字符串型”。 指针中的值将在读取下一行数据时更改。如果想要存储来自某一行的值供以后使用，请将此字符串显式复制到自己分配的缓冲区中。
char* GetRaw(void);	获取参数的值，作为以非空值终止的字节数组。仅当已知参数数据类型为“原始”时，才能调用此函数。此函数无法将数据从其他数据类型转换为“原始”。
unsigned long GetActualDataLen(void);	获取 GetRaw 返回的数组的当前长度。
TINFTime GetTime(void);	获取具有“日期/时间”数据类型的参数的值。仅当已知参数数据类型为“日期/时间”时，才能调用此函数。此函数无法将数据从其他数据类型转换为“日期/时间”。
void SetLong(long lVal);	设置具有“长整型”数据类型的输出参数的值。
void SetDouble(double dblVal);	设置具有“双精度型”数据类型的输出参数的值。
void SetString(char* sVal);	设置具有“字符串型”数据类型的输出参数的值。
void SetRaw(char* rVal, size_t ActualDataLen);	设置以非空值终止的字节数组。
void SetTime(TINFTime timeVal);	设置具有“日期/时间”数据类型的输出参数的值。

仅当对 64 位集成服务运行外部过程时，才能使用 SetInt32 或 GetInt32 函数。不要使用以下任何函数：

- GetLong
- SetLong
- GetpLong
- GetpDouble
- GetpTime

使用两个参数列表传递参数。

下表列出了外部过程基类的成员变量：

变量	说明
m_nInParamCount	输入参数的数量。
m_pInParamVector	实际输入参数数组。
m_nOutParamCount	输出参数的数量。
m_pOutParamVector	实际输出参数数组。
<b>注意：</b> 定义为输入/输出的端口将在这两个参数列表中显示。	

## 代码页访问函数

Informatica 提供两个可返回集成服务代码页的代码页访问函数，以及两个可返回外部过程所处理数据的代码页的代码页访问函数。当集成服务在 Unicode 模式下运行时，传递至外部过程程序的字符串数据可以包含多字节字符。代码页决定外部过程以何种方式解释多字节字符串。当集成服务在 Unicode 模式下运行时，外部过程程序处理的数据必须与集成服务代码页双向兼容。

### 签名

使用以下函数通过外部过程程序获得集成服务代码页。两个函数会返回相同的信息。

```
int GetServerCodePageID() const;
const char* GetServerCodePageName() const;
```

使用以下函数获得外部过程通过外部过程程序处理的数据的代码页。两个函数会返回相同的信息。

```
int GetDataCodePageID(); // returns 0 in case of error
const char* GetDataCodePageName() const; // returns NULL in case of error
```

## 转换名访问函数

Informatica 提供两个转换名称访问函数，可返回外部过程转换的名称。GetWidgetName() 函数可返回转换的名称，GetWidgetInstanceName() 函数可返回 Maplet 或映射中转换实例的名称。

### 签名

转换名称访问函数返回的 char\* 是集成服务代码页中的 MBCS 字符串。它不在数据代码页中。

```
const char* GetWidgetInstanceName() const;
const char* GetWidgetName() const;
```

## 过程访问函数

Informatica 提供两个可提供与外部过程转换关联的外部过程相关信息的过程访问函数。GetProcedureName() 函数可返回外部过程转换的“过程名称”字段中所指定外部过程的名称。GetProcedureIndex() 函数可返回外部过程的索引。

### 签名

使用以下函数获取与外部过程转换关联的外部过程的名称：

```
const char* GetProcedureName() const;
```

使用以下函数获取与外部过程转换关联的外部过程的索引：

```
inline unsigned long GetProcedureIndex() const;
```

## 分区相关函数

可在具有多个分区的会话中使用针对外部过程的分区相关函数。对包含外部过程转换的会话进行分区时，集成服务可为每个分区创建这些转换的实例。例如，如果您为会话定义了五个分区，集成服务会在会话运行时为每个外部过程创建五个实例。

### 签名

使用以下函数获得会话中的分区数：

```
unsigned long GetNumberOfPartitions();
```

使用以下函数获得调用该外部过程的分区索引：

```
unsigned long GetPartitionIndex();
```

## 跟踪级别函数

跟踪级别函数将返回会话跟踪级别，例如：

```
typedef enum
{
    TRACE_UNSET = 0,
    TRACE_TERSE = 1,
    TRACE_NORMAL = 2,
    TRACE_VERBOSE_INIT = 3,
    TRACE_VERBOSE_DATA = 4
} TracingLevelType;
```

## 签名

可以使用以下函数返回会话跟踪级别：

```
TracingLevelType GetSessionTraceLevel();
```

## 第 9 章

# 筛选器转换

本章包括以下主题：

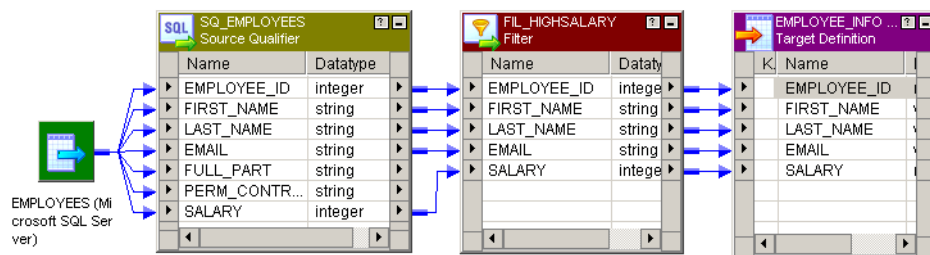
- [筛选器转换概览, 174](#)
- [筛选转换组件, 175](#)
- [筛选条件, 175](#)
- [创建筛选器转换的步骤, 176](#)
- [筛选器转换提示, 176](#)

## 筛选器转换概览

使用筛选器转换筛选出映射中的行。作为主动转换，筛选器转换可更改所传递的行数。筛选器转换允许传递满足指定筛选条件的行。该转换将删除不满足条件的行。可以根据一个或多个条件筛选数据。筛选器转换是主动转换。

对于集成服务计算的每一行，根据该行是否满足指定条件，筛选条件将返回 TRUE 或 FALSE。对于返回 TRUE 的每一行，集成服务将经过转换。对于返回 FALSE 的每一行，集成服务将删除该行并在会话日志中写入一条消息。

以下映射会通过筛选器转换传递包含员工数据的人力资源表中的行。该筛选器可查找行中薪资不低于 \$30,000 美元员工。



无法将来自多个转换的端口连接到筛选器转换中。筛选器的输入端口必须来自单个转换。

**提示:** 将筛选器转换放置在尽可能接近映射中的源的位置，以最大限度提高会话性能。您可以在从源到目标的数据流中尽早筛选出不需要的数据，而不是将打算放弃的行传递到映射中。

# 筛选转换组件

可以在 Transformation Developer 或 Mapping Designer 中创建筛选器转换。筛选器转换包含以下选项卡：

- **转换。**输入转换的名称和说明。筛选器转换的命名约定为 `FIL_TransformationName`。也可以使转换可重用。
- **端口。**创建和配置端口。
- **属性。**配置筛选条件以筛选行。使用表达式编辑器输入筛选条件。还可以配置跟踪级别以确定会话日志文件中报告的事务详情量。
- **元数据扩展。**创建不可重用的元数据扩展以扩展转换元数据。配置扩展名称、数据类型、精度和值。如果希望将元数据扩展提供给所有转换，还可以将其提升为可重用。

## 配置筛选器转换端口

您可以在“端口”选项卡上创建和修改端口。

可以在“端口”选项卡上配置以下属性：

- **端口名称。**端口的名称。
- **数据类型、精度和小数位数。**配置每个端口的数据类型并设置精度和小数位数。
- **Port type.**所有端口都是输入/输出端口。输入端口接收数据，输出端口传递数据。
- **默认值和说明。**设置端口的默认值并添加说明。

## 筛选条件

筛选条件是一个表达式，该表达式可返回 TRUE 或 FALSE。使用“属性”选项卡上的表达式编辑器输入条件。

返回单个值的任何表达式都可用作筛选器。例如，如果要筛选出薪资低于 \$30,000 的员工所在的行，请输入以下条件：

```
SALARY > 30000
```

可以使用 AND 和 OR 逻辑运算符指定条件的多个组成部分。如果要筛选出薪资低于 \$30,000 和高于 \$100,000 的员工，请输入以下条件：

```
SALARY > 30000 AND SALARY < 100000
```

您也可以为筛选条件输入一个常量。与 FALSE 等效的数值为零 (0)。任何非零值均等效于 TRUE。例如，转换包含一个名为 NUMBER\_OF\_UNITS 的端口，该端口的数据类型为数值。可以配置筛选条件，使其在 NUMBER\_OF\_UNITS 的值等于零时返回 FALSE。否则，该条件将返回 TRUE。

无需将 TRUE 或 FALSE 指定为表达式中的值。TRUE 和 FALSE 是您设置的任何条件的隐式返回值。如果筛选条件的计算结果为空，则会将该行视为 FALSE。

**注意：**筛选条件区分大小写。

## 筛选具有空值的行

要筛选包含空值或空格的行，请使用 ISNULL 和 IS\_SPACES 函数测试端口的值。例如，如果要在 FIRST\_NAME 端口中筛选出包含空值的行，请使用以下条件：

```
IIF(ISNULL(FIRST_NAME),FALSE,TRUE)
```

此条件指出，如果 FIRST\_NAME 端口为空，则返回值为 FALSE，并且应放弃该行。否则，该行将传递到下一个转换。

## 创建筛选器转换的步骤

按照以下步骤创建筛选器转换：

1. 在 Mapping Designer 中，打开一个映射。
2. 单击“转换” > “创建”。选择筛选器转换。
3. 输入转换的名称。单击“创建”，然后单击“完成”。
4. 从源限定符或其他转换中选择并拖动所有端口，将其添加到筛选器转换中。
5. 双击标题栏，然后单击“端口”选项卡。还可以在转换内手动创建端口。
6. 单击“属性”选项卡以配置筛选条件和跟踪级别。
7. 在筛选条件的“值”部分，打开表达式编辑器。
8. 输入要应用的筛选条件。默认条件将返回 TRUE。

使用来自转换中其中一个输入端口的值作为该条件的一部分。但是，也可以使用来自其他转换中的输出端口的值。

9. 输入表达式。单击“验证”，验证您输入的条件语法。
10. 选择跟踪级别。
11. 在“元数据扩展”选项卡上添加元数据扩展。

## 筛选器转换提示

在映射中尽早使用筛选器转换。

要最大限度提高会话性能，请使筛选器转换尽可能接近映射中的源。可以在从源到目标的数据流中尽早筛选出不需要的数据，而不是将打算放弃的行传递到映射中。

可以使用源限定符转换进行筛选。

源限定符转换为筛选行提供了一种备选方法。源限定符转换将在从源读取行时筛选行，而不是在映射中筛选行。主要差异在于源限定符限制从源中提取的行集，而筛选器转换则限制发送到目标中的行集。由于源限定符将减少整个映射中使用的行数，因此它能提供更出色的性能。

不过，源限定符转换仅允许筛选关系源中的行，而筛选器转换则可筛选任何类型的源中的行。此外，还请注意，由于源限定符转换在数据库中运行，因此必须确保源限定符转换中的筛选条件仅使用标准 SQL。筛选器转换可以定义使用任何语句或转换函数的条件，可以返回 TRUE 或 FALSE 值。



## 第 10 章

# HTTP 转换

本章包括以下主题：

- [HTTP 转换概览, 177](#)
- [创建 HTTP 转换, 178](#)
- [配置“属性”选项卡, 178](#)
- [配置“HTTP”选项卡, 179](#)
- [示例, 185](#)

## HTTP 转换概览

HTTP 转换让您能够连接至 HTTP 服务器以使用其服务和应用程序。HTTP 转换是一种被动转换。当运行具有 HTTP 转换的会话时，集成服务会连接至 HTTP 服务器，并根据您对转换的配置发出在 HTTP 服务器上检索数据或更新数据的请求：

- **从 HTTP 服务器读取数据。**当集成服务从 HTTP 服务器读取数据时，它会从 HTTP 服务器检索数据并将数据传递到目标或映射中的下游转换。例如，您可以连接至 HTTP 服务器读取当前的库存数据、在 PowerCenter 会话期间对数据执行计算，并将数据传递到目标。
- **更新 HTTP 服务器上的数据。**当集成服务写入 HTTP 服务器时，它会将数据发布至 HTTP 服务器并将 HTTP 服务器响应传递到目标或映射中的下游转换。例如，您可以在会话过程中将提供计划信息的数据从上游转换发布到 HTTP 服务器。

集成服务将数据从上游转换或源传递到 HTTP 转换、读取在 HTTP 转换或应用程序连接中配置的 URL，并将读取或更新数据的 HTTP 请求发送到 HTTP 服务器。

请求包含表头信息并可能包含主体信息。表头包含身份验证参数、用于激活 HTTP 服务器上驻留的程序或 Web 服务的命令，以及应用于整个 HTTP 请求的其他信息。主体包含集成服务发送到 HTTP 服务器的数据。

当集成服务发送读取数据的请求时，HTTP 服务器会发回含有所请求数据的 HTTP 响应。集成服务将请求的数据发送到下游转换或目标。

当集成服务发送更新数据的请求时，HTTP 服务器会写入其收到的数据并发回更新成功的 HTTP 响应。HTTP 转换将响应代码 200、201 和 202 视为成功。HTTP 转换将其他所有响应代码视为失败。当 HTTP 服务器将失败的响应代码发送至 HTTP 转换时，会话日志会显示错误。然后，集成服务会将该 HTTP 响应发送至下游转换或目标。

您可以配置 HTTP 转换以设置 HTTP 响应的表头。HTTP 响应主体数据通过 HTTPOUT 输出端口传递。

## 身份验证

HTTP 转换使用以下形式的身份验证：

- **基本**。基于未加密的用户名和密码。
- **摘要**。基于加密的用户名和密码。
- **NTLM**。基于加密的用户名、密码和域。

## 连接至 HTTP 服务器

配置 HTTP 转换时，您可以配置连接的 URL。还可以在 Workflow Manager 中创建 HTTP 连接对象。在以下情况下配置 HTTP 应用程序连接：

- HTTP 服务器需要身份验证。
- 您要配置连接超时。
- 您要替代 HTTP 转换中的基础 URL。

## 创建 HTTP 转换

您可以在 Transformation Developer 或 Mapping Designer 中创建 HTTP 转换。HTTP 转换具有以下选项卡：

- **转换**。配置转换的名称和说明。
- **端口**。查看转换的输入和输出端口。您无法在“端口”选项卡上添加或编辑端口。当您在“HTTP”选项卡上添加端口至表头组时，Designer 会在“端口”选项卡上创建端口。
- **属性**。“属性”选项卡用于配置 HTTP 转换的属性。
- **HTTP**。“HTTP”选项卡用于配置方法、端口和 URL。

要创建 HTTP 转换，请执行以下操作：

1. 在 Transformation Developer 或 Mapping Designer 中，单击“转换”>“创建”。
2. 选择 HTTP 转换。
3. 输入转换的名称。
4. 单击“创建”。

HTTP 转换此时将显示在工作区中。

5. 单击“完成”。
6. 配置转换中的选项卡。

## 配置“属性”选项卡

HTTP 转换是使用自定义转换构建的。一些自定义转换属性不适用于 HTTP 转换或不可配置。

下表说明了可配置的 HTTP 转换属性：

选项	说明
运行时位置	包含 DLL 或共享库的位置。默认为 \$PMExtProcDir。输入相对于运行 HTTP 转换会话的集成服务节点的路径。 如果此属性为空，集成服务将使用在该集成服务上定义的环境变量来查找 DLL 或共享库。 您必须将所有 DLL 或共享库复制到运行时位置或在集成服务节点上定义的环境变量。如果不能找到 DLL、共享库或引用文件，集成服务无法加载过程。
跟踪级别	此转换的会话日志中显示的详细信息量。默认值为“普通”。
是否可分区	指示是否可以在使用此转换的管道中创建多个分区： <ul style="list-style-type: none"><li>- 否。无法对转换进行分区。转换和同一渠道中的其他转换只有一个分区。</li><li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。当自定义转换的不同分区必须共享内存中的对象时，请选择“本地”。</li><li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li></ul> 默认值为“否”。
输出是可重复的	指示多次会话运行的输出数据顺序是否一致。 <ul style="list-style-type: none"><li>- 从不。会话运行之间的输出数据顺序不一致。</li><li>- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。</li><li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li></ul> 默认值为“基于输入顺序”。
要求每个分区一个线程	指示集成服务在处理分区时是否为过程的每个分区使用一个线程。
输出具有确定性	指示转换在多次会话运行时是否生成一致的输出数据。启用此属性可对使用此转换的会话执行恢复操作。默认为“已启用”。

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

## 配置“HTTP”选项卡

在“HTTP”选项卡上，您可以配置转换以从 HTTP 服务器读取数据或写入数据到 HTTP 服务器。在“HTTP”选项卡上配置以下信息：

- **选择方法。** 根据要读取 HTTP 服务器中的数据还是将数据写入到 HTTP 服务器、执行部分更新还是替换整块数据，或者删除 HTTP 服务器中的数据来选择 GET、POST、SIMPLE POST、SIMPLE PATCH、SIMPLE PUT 或 SIMPLE DELETE 方法。
- **配置组和端口。** 通过配置输入和输出端口管理 HTTP 请求/响应主体和表头详细信息。还可以配置含特殊字符的端口名称。
- **配置基础 URL。** 配置要连接到的 HTTP 服务器的基础 URL。

# 选择方法

在转换中定义的组和端口取决于您选择的方法。要从 HTTP 服务器读取数据，请选择 GET 方法。要将数据写入到 HTTP 服务器，请选择 POST 或 SIMPLE POST 方法。

下表解释了不同的方法：

方法	说明
GET	从 HTTP 服务器读取数据。
POST	将数据从多个输入端口写入到 HTTP 服务器。
SIMPLE POST	将数据作为单块数据从一个输入端口写入到 HTTP 服务器。
SIMPLE PATCH	将部分数据作为补丁从一个输入端口更新到资源。
SIMPLE PUT	替换或写入资源。
SIMPLE DELETE	删除 HTTP 服务器中的资源。

# 配置组和端口

您添加至 HTTP 转换的端口取决于您选择的方法和组。HTTP 转换使用以下组：

- **输出。**包含 HTTP 响应的主体数据。将响应从 HTTP 服务器传递到下游转换或目标。默认情况下，包含一个输出端口，HTTPOUT。输出组中无法添加端口。您可以修改 HTTPOUT 输出端口的精度。
- **输入。**包含 HTTP 请求的主体数据。还包含 Designer 用于构建连接到 HTTP 服务器的最终 URL 的元数据。为了将数据写入 HTTP 服务器，输入组会将主体信息传递到 HTTP 服务器。默认情况下，包含一个输入端口。
- **表头。**包含请求和响应的表头数据。在集成服务发送 HTTP 请求时将表头信息传递到 HTTP 服务器。您添加至表头组的端口会传递 HTTP 表头的信息。当您添加端口至表头组时，Designer 会将端口添加至“端口”选项卡上的输入和输出组。默认情况下，不包含端口。对于所有方法，您可以为 HTTP 请求表头信息使用表头组。

**注意：**通过 HTTP 转换传递的数据必须是字符串数据类型。字符串数据包括 HTTP 通信中常用的所有标记语言，如 HTML 和 XML。

## GET 方法

从 HTTP 服务器读取数据。要为 HTTP 请求定义元数据，请使用输入组添加 Designer 用于为 HTTP 服务器构建最终 URL 的输入端口。

下表介绍了 GET 方法的组和端口：

请求/响应	组	说明
REQUEST	输入	Designer 使用输入端口的名称和值来构建最终 URL。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"><li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li><li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li></ul>

请求/响应	组	说明
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## POST 方法

将数据从多个输入端口写入到 HTTP 服务器。要为 HTTP 请求定义元数据，请对定义 HTTP 请求主体的数据使用输入组。

下表介绍了 POST 方法的端口：

请求/响应	组	说明
REQUEST	输入	您可以将多个端口添加至输入组。根据您添加至表头组的端口，HTTP 请求的主体数据可通过一个或多个输入端口传递。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li> </ul>
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## SIMPLE POST 方法

POST 方法的简化版本。将数据作为单块数据从一个输入端口写入到 HTTP 服务器。要为 HTTP 请求定义元数据，请对定义 HTTP 请求主体的数据使用输入组。

下表介绍了 SIMPLE POST 方法的端口：

请求/响应	组	说明
REQUEST	输入	可以添加一个输入端口。HTTP 请求的主体数据可通过一个输入端口传递。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li> </ul>

请求/响应	组	说明
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## SIMPLE PATCH 方法

将部分数据作为补丁从一个输入端口更新到资源。将数据作为完整或部分数据块从一个输入端口写入到 HTTP 服务器。要为 HTTP 请求定义元数据，请对定义 HTTP 请求主体的数据使用输入组。

下表介绍了 SIMPLE PATCH 方法的端口：

请求/响应	组	说明
REQUEST	输入	可以添加一个输入端口。HTTP 请求的主体数据可通过一个输入端口传递。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li> </ul>
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## SIMPLE PUT 方法

替换或写入资源。将数据作为单块数据从一个输入端口写入到 HTTP 服务器。

如果数据不存在，SIMPLE PUT 方法将发布数据。如果数据存在，SIMPLE PUT 方法会将数据作为单块数据从一个输入端口更新到 HTTP 服务器。

要为 HTTP 请求定义元数据，请对定义 HTTP 请求主体的数据使用输入组。

下表介绍了 SIMPLE PUT 方法的端口：

请求/响应	组	说明
REQUEST	输入	可以添加一个输入端口。HTTP 请求的主体数据可通过一个输入端口传递。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li> </ul>

请求/响应	组	说明
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## SIMPLE DELETE 方法

删除 HTTP 服务器中的资源。如果需要请求主体，SIMPLE DELETE 会作为单块数据从一个输入端口中删除 HTTP 服务器的数据。要为 HTTP 请求定义元数据，请使用输入组添加 Designer 用于为 HTTP 服务器构建最终 URL 的输入端口。

下表介绍了 SIMPLE DELETE 方法的端口：

请求/响应	组	说明
REQUEST	输入	Designer 使用输入端口的名称和值来构建最终 URL。
REQUEST	表头	您可以配置 HTTP 请求的输入和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入和输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输入/输出端口创建输出端口。</li> </ul>
RESPONSE	表头	您可以配置 HTTP 响应的输出和输入/输出端口。Designer 会根据您添加至表头组的端口将端口添加至输入和输出组： <ul style="list-style-type: none"> <li>- 输入组。根据表头组中的输入/输出端口创建输入端口。</li> <li>- 输出组。根据表头组中的输出和输入/输出端口创建输出端口。</li> </ul>
RESPONSE	输出	HTTP 响应的所有主体数据均通过 HTTPOUT 输出端口传递。

## 添加 HTTP 名称

Designer 不允许在端口名称中使用特殊字符，如短划线 (-)。如果需要在端口名称中使用特殊字符，可配置 HTTP 名称以替代端口名称。例如，如果需要名为 Content-type 的输入端口，可将端口命名为 ContentType 并输入 Content-Type 作为 HTTP 名称。

## 配置 URL

选择方法并配置输入和输出端口后，必须配置 URL。输入基础 URL，Designer 将构建最终 URL。如果选择 GET 或 SIMPLE DELETE 方法，最终 URL 会包含基础 URL 和基于输入组中端口名称的参数。如果选择 SIMPLE PATCH、SIMPLE PUT、POST 或 SIMPLE POST 方法，最终 URL 将与基础 URL 相同。

您可以使用映射参数或变量来配置基础 URL。例如，声明映射参数 \$\$ParamBaseURL、在基础 URL 字段中输入映射参数 \$\$ParamBaseURL，然后在参数文件中定义 \$\$ParamBaseURL。

还可以在配置 HTTP 应用程序连接时指定 URL。在 HTTP 应用程序连接中指定的基础 URL 将替代在 HTTP 转换中指定的基础 URL。

**注意:** HTTP 服务器可将 HTTP 请求重定向至另一个 HTTP 服务器。当发生这种情况时，HTTP 服务器会将一个 URL 发回集成服务，然后集成服务再建立到其他 HTTP 服务器的连接。集成服务最多可再建立五个连接。

### GET 方法的最终 URL 构建

Designer 根据基础 URL 和输入组中的端口名称构建 GET 方法的最终 URL。它将 HTTP 参数以 HTTP 查询字符串的形式附加到基础 URL 来构建最终 URL。查询字符串包含一个问号(?), 后跟名称/值对。Designer 会附加与您添加至输入组的输入端口的名称和值相对应的问号和名称/值对。

当您选择 GET 方法并将输入端口添加至输入组时, Designer 会将以下组和端口信息附加到基础 URL 来构建最终 URL:

`?<input group input port 1 name> = $<input group input port 1 value>`

对于第一个输入组输入端口后面的每个输入端口, Designer 附加以下组和端口信息:

`& <input group input port n name> = $<input group input port n value>`

其中 *n* 表示输入端口。

例如, 如果您输入基础 URL `www.company.com` 并将输入端口 ID、EmpName 和 Department 添加至输入组, Designer 会构建以下最终 URL:

`www.company.com?ID=$ID&EmpName=$EmpName&Department=$Department`

您可以编辑最终 URL 以修改或添加运算符、变量或其他参数。有关 HTTP 请求和查询字符串的更多信息, 请参见 <http://www.w3c.org>。

### 参数化基础 URL

您可以参数化 GET 和 SIMPLE DELETE 方法的基础 URL。

例如, 您具有以下基础 URL:

`www.informatica.com`

Designer 使用基础 URL 中的信息并可能构建最终 URL, 如下所示:

`www.informatica.com?firstname=1&lastname=2`

选中此复选框选项以在 Designer 中为 GET 或 SIMPLE DELETE 方法参数化基础 URL 时, 可以编辑基础 URL 以使用映射参数或变量, 如下所示:

`www.informatica.com/$firstname$lastname`

集成服务将源文件值发送至 HTTP 转换的名字和姓氏输入端口, 并将 HTTP 请求发送至最终 URL 中指定的 HTTP 服务器。

然后最终 URL 可能会如下所示:

`www.informatica.com/12`

### URL 中的特殊字符

配置 URL 时, 最终 URL 可能包含以 UTF-8 字符集编码的特殊字符。

下表介绍了如何在 UTF-8 环境中配置 URL 中使用的一些特殊字符:

特殊字符	已编码 (UTF-8)
'	%27
(	%28
)	%29



特殊字符	已编码 (UTF-8)
*	2A
空格	%20

如果在集成服务级别或在会话级别使用 `UseURLAsEnteredinCURLEncoding` 自定义标志，则您传递的 URL 会与基础 URL 中输入的 URL（CURL 编码格式）完全一样。

## 示例

本节包含每种方法类型的示例：

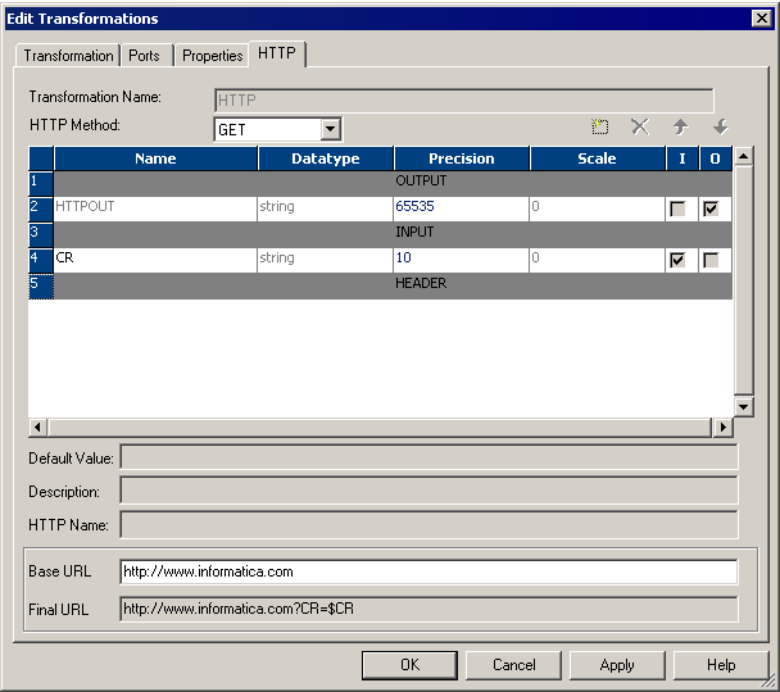
- GET
- POST
- SIMPLE POST
- SIMPLE PATCH
- SIMPLE PUT
- SIMPLE DELETE

### GET 示例

此示例使用的源文件包含以下数据：

```
78576
78577
78578
```

下图显示了 GET 示例中 HTTP 转换的“HTTP”选项卡：



Designer 将问号 (?)、输入组输入端口名称、美元符号 (\$)，以及又一个输入组输入端口名称附加到基础 URL 来构建最终 URL：

`http://www.informatica.com?CR=$CR`

集成服务将源文件值发送至 HTTP 转换的 CR 输入端口，并将以下 HTTP 请求发送至 HTTP 服务器：

`http://www.informatica.com?CR=78576`  
`http://www.informatica.com?CR=78577`  
`http://www.informatica.com?CR=78578`

HTTP 服务器将 HTTP 响应发回集成服务，然后集成服务通过 HTTP 转换的输出端口将数据发送至目标。

## POST 示例

此示例使用的源文件包含以下数据：

`33,44,1`  
`44,55,2`  
`100,66,0`

下图显示了具有对应输入端口的源文件中的每个字段：

Transformation Name: HTTP

HTTP Method: POST

	Name	Datatype	Precision	Scale	I	O
1	OUTPUT					
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	INPUT					
4	num1	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	num2	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	num3	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	HEADER					
8	hdr2	string	10	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL:

Final URL:

OK Cancel Apply Help

集成服务通过 HTTP 转换的输入端口发送每行中三个字段的值，并将 HTTP 请求发送至最终 URL 中指定的 HTTP 服务器。

## SIMPLE POST 示例

以下文本显示该示例中使用的 XML 文件：

```
<?xml version="1.0" encoding="UTF-16LE"?>
<n4:Envelope xmlns:cli="http://localhost:8080/axis/Clienttest1.jws" xmlns:n4="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <n4:Header>
  </n4:Header>
  <n4:Body n4:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><cli:smp1source>
    <Metadatainfo xsi:type="xsd:string">smp1sourceRequest.Metadatainfo106</Metadatainfo></cli:smp1source>
  </n4:Body>
</n4:Envelope>,capeconnect:Clienttest1services:Clienttest1#smp1source
```

下图显示 SIMPLE POST 示例中 HTTP 转换的 HTTP 选项卡：

**Edit Transformations**

Transformation Name:

HTTP Method:

	Name	Datatype	Precision	Scale	I	O
1	OUTPUT					
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	INPUT					
4	SOAPRequest	string	65535	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	HEADER					
6	soapaction	string	100	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	host	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
8	Date	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
9	server	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL:

Final URL:

OK Cancel Apply Help

集成服务通过输入端口发送源文件的正文，并将 HTTP 请求发送到在最终 URL 中指定的 HTTP 服务器。

## SIMPLE PATCH 示例

此示例使用的源文件包含以下数据：

```
{"firstname": "Raj"}
```

下图显示了具有对应输入端口的源文件中的每个字段：

Edit Transformations

Transformation

Ports

Properties

HTTP

Static

Transformation Name: HTTP3

HTTP Method: SIMPLE PATCH

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	update	string	1000	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	ContentType	string	1000	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

<

>

Default Value: 'application/json'

Description:

HTTP Name: Content-Type

Base URL: \$\$BASEURL1

Final URL: \$\$BASEURL1

☐ Parameterize Base URL

OK

Cancel

Apply

Help

在“基础 URL”和“最终 URL”中加入 \$\$BASEURL1 映射变量。

\$\$BASEURL1 映射变量指向以下链接：

http://avengers5:8181/emp\_all/1/

SIMPLE PATCH 支持将部分数据从一个输入端口更新至 HTTP 服务器。集成服务通过 HTTP 转换的输入端口发送受影响行中的值，并将 HTTP 请求发送至最终 URL 中指定的 HTTP 服务器。

## SIMPLE PUT 示例

此示例使用的源文件包含以下数据：

```
{"emp_id": 10,"firstname": "Raj","lastname": "Kumar","designation": "QA","department": "RND","age": 24}
```

下图显示了具有对应输入端口的源文件中的每个字段：

Edit Transformations

Transformation

Ports

Properties

HTTP

Static

Transformation Name: HTTP

HTTP Method: SIMPLE PUT

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	update	string	1000	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	contentType	string	100	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URLhttp://avengers5:8181/emp\_all/10/

Final URLhttp://avengers5:8181/emp\_all/10/

☐ Parameterize Base URL

OK

Cancel

Apply

Help

SIMPLE PUT 将数据作为单块数据从一个输入端口写入到 HTTP 服务器。集成服务通过 HTTP 转换的输入端口发送源文件主体，并将 HTTP 请求发送至最终 URL 中指定的 HTTP 服务器。

## SIMPLE DELETE 示例

此示例使用的源文件包含以下数据：

2  
1

下图显示了已选中“参数化基础 URL”选项的 SIMPLE DELETE 示例的 HTTP 转换的 HTTP 选项卡：

Static

Transformation Name: HTTP

HTTP Method: SIMPLE DELETE

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HITPOUT	string	65535	0		<input checked="" type="checkbox"/>
3			INPUT			
4	id	string	10	0	<input checked="" type="checkbox"/>	
5			HEADER			
6	ContentType	string	10	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL: http://avengers5:8181/emp\_all/\$id/

Final URL: Use Base URL for Parameterization

☒ Parameterize Base URL

OK Cancel Apply Help

选中“参数化基础 URL”复选框时，Designer 会将输入组输入端口名称附加到基础 URL 来构建最终 URL：

http://www.avengers5:8181/emp\_all/\$id/

集成服务将源文件值发送至 HTTP 转换的 id 输入端口，并将以下 HTTP 请求发送至 HTTP 服务器：

http://www.avengers5:8181/emp\_all/2  
http://www.avengers5:8181/emp\_all/1

SIMPLE DELETE 方法会从 HTTP 服务器中删除所需数据。HTTP 服务器将 HTTP 响应发回集成服务，然后集成服务通过 HTTP 转换的输出端口将更新后的数据发送至目标。

## 第 11 章

# Identity Resolution 转换

本章包括以下主题：

- [Identity Resolution 转换概览, 192](#)
- [创建和配置转换, 192](#)
- [Identity Resolution 转换选项卡, 194](#)
- [组和端口, 195](#)

## Identity Resolution 转换概览

Identity Resolution 转换是一种主动转换，可用于搜索和匹配 Informatica Identity Resolution (IIR) 中的数据。PowerCenter 集成服务使用您在 Identity Resolution 转换中指定的搜索定义来搜索和匹配 IIR 表中的数据。系统中的输入和输出视图确定转换的输入和输出端口。

在 Identity Resolution 转换中配置匹配偏差和搜索宽度参数可设置匹配方案和搜索级别。运行会话时，IIR 会将候选项记录返回 Identity Resolution 转换。

IIR 提供对存储在 Oracle、DB2/UDB 和 Microsoft SQL Server 表中的所有类型标识数据的联机 and 批量搜索、匹配、筛选、链接及重复数据发现。IIR 使用 IIR 搜索服务器执行搜索和匹配操作。

## 创建和配置转换

在 Transformation Developer 中创建 Identity Resolution 转换。创建 Identity Resolution 转换时，首先连接至 Informatica Identity Resolution 搜索服务器。该搜索服务器提供对 IIR 表的访问。

连接后，您可配置系统及搜索和匹配参数。在 **配置 IR 转换** 窗口中配置以下信息：

- IR 搜索服务器连接
- 系统和搜索配置
- 输入和输出视图

## 搜索服务器连接

要连接到搜索服务器，您需要提供主机名和端口号。然后，配置规则库连接字符串。规则库包含有关 IIR 用于搜索和匹配数据的系统的信息。Identity Resolution 转换中的规则库连接指定包含规则库的数据库。



**注意:** 创建转换后便无法更改主机名或规则库连接。

**IR 主机名**

运行 Informatica Identity Resolution 服务器的计算机的主机名。

**端口**

搜索服务器的端口号。

**规则库连接字符串**

到搜索服务器的连接字符串。可以将规则库连接指定为 ODBC 或 SSA。在 IIR 主机中配置 ODBC 和 SSA。

**ODBC**

下表介绍了 ODBC 连接信息：

连接属性	说明
规则库编号	为 IR 规则库指定的编号。
用户名	数据库用户名。
密码	数据库用户名的密码。
服务名称	在 odbc.ini 文件中定义的服务名称。

**SSA**

要通过 SSA 进行连接，您需要提供别名。别名会对应用程序隐藏连接字符串。IIR 使用规则库、数据库和源名称的别名。

**系统和搜索配置**

连接到搜索服务器后，选择系统以及搜索和匹配参数。

创建转换后，可以在 **IR 设置**选项卡中更改这些参数。

**系统**

在规则库中选择系统。系统是 IIR 规则库中最高级别的逻辑定义。

**搜索定义**

选择包含系统规则的搜索定义。搜索定义驻留在系统中，包含记录的搜索、匹配和显示规则。

**搜索宽度**

选择可确定您要执行之搜索的范围的搜索宽度。

下表描述了可以选择的搜索宽度：

搜索宽度	说明
精优化	提供可快速响应的精优化搜索。在以下情况下使用该选项：与缺少匹配项关联的风险较低时、不需要高匹配准确性时或数据量很大且响应时间很关键时。
典型	在质量和响应时间之间做出平衡。对在线或批处理事务搜索使用该选项。默认为典型。
全面	提供延长响应时间的详细搜索。在以下情况下使用该选项：与缺少匹配项关联的风险较高时、数据质量很重要时或数据量很小且响应时间不重要时。

### 匹配偏差

选择匹配偏差，确定匹配架构要选择候选项记录的积极程度。

下表描述可以选择的匹配偏差选项：

匹配偏差	说明
极限	提供与一组待处理的可能方法匹配的候选项。处理的响应时间更多。当您需查找一个匹配项（即使包含错误和变体）时使用该选项。
保守	提供并非过量或极限的匹配项。对在线或批处理事务搜索使用该选项。
典型	提供近似匹配项。当需要准确匹配项时在批处理系统中使用该选项。
宽松	提供比典型级别具有更高变体程度的匹配项。在缺少匹配项的风险很高的系统中以及您可以查看结果的情况下使用该选项。

## 视图选择

在选定系统和搜索定义后，可以选择输入视图和输出视图。

### 输入视图

输入视图中的字段可以确定转换输入端口。输入视图可以包含用于搜索和匹配操作的字段。如果没有选择输入视图，则转换会将 IIR 标识表 (IDT) 布局用于输入视图。

**注意:** 如果系统在视图定义中包含 XFORM 子句，将多个字段连接成为一个字段，将无法在 Identity Resolution 转换中使用视图。但可以在 Identity Resolution 转换之外连接多个字段，然后将已连接的字符串传递到 Identity Resolution 转换的输入端口。

### 输出视图

输出视图中的字段可以确定转换输出端口。输出视图将对搜索服务器返回的搜索结果进行格式化。如果没有选择输出视图，则转换会将 IIR 标识表 (IDT) 布局用于输出视图。

# Identity Resolution 转换选项卡

Identity Resolution 转换具有以下选项卡：

### 转换

配置转换说明。

### 端口

查看代表输入和输出视图的组和端口。您无法编辑组和端口。

### 属性

下表介绍了可在**属性**选项卡上配置的属性：

转换属性	说明
运行时位置	包含 DLL 或共享库的位置。默认为 \$PMExtProcDir。
跟踪级别	此转换的会话日志中显示的详细信息量。默认值为“普通”。
是否可分区	指示是否可以在使用此转换的管道中创建多个分区： <ul style="list-style-type: none"><li>- 否。无法对转换进行分区。该转换和相同管道中的其他转换限制为一个分区。</li><li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。当自定义转换的不同分区必须共享内存中的对象时，选择“本地”。</li><li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li></ul> 默认选项是“跨整个网格”。

### IR 设置

查看创建转换时配置的设置。您可以更改系统和搜索信息。

## 组和端口

Identity Resolution 转换包含一个输入组和一个输出组。输入组具有代表搜索定义输入视图中字段的端口。输出组具有代表搜索定义输出视图中字段的端口以及说明搜索结果的端口。

您可以在**端口**选项卡上查看组和端口。您还可以在 **IR 设置**选项卡上查看端口。

### 输入组和端口

Identity Resolution 转换包含一个含输入端口的输入组。输入组中的端口数与所选输入视图中的字段数相匹配。将搜索所需的所有输入端口链接到 Identity Resolution 转换。

### 输出组和端口

输出组包含输出端口和一个输入/输出端口。输出组包含输出视图中的端口以及每个 Identity Resolution 转换的默认端口。

基于输出视图的输出端口包含来自搜索服务器的候选项记录。

下表介绍了默认输出端口：

默认输出端口	说明
IR_Search_Link	输入/输出搜索链接端口。使用此传递端口可在输入端上源数据和结果输出候选项之间传递链接。
IR_Score_Out	包含来自搜索服务器所执行搜索操作的得分的输出端口。值为从 0 到 100 的正数。 例如，如果搜索输入为“Rob”的名字，结果可能包含得分为 100 的名字为“Rob”的候选项记录。结果还可能包含得分为 90 的名字为“Bob”的候选项记录。Informatica Identity Resolution 会计算与结果关联的得分。
IR_Result_Count	包含记录数的输出端口。

## 第 12 章

# Java 转换

本章包括以下主题：

- [Java 转换概览, 197](#)
- [使用“Java 代码”选项卡, 200](#)
- [配置端口, 200](#)
- [配置 Java 转换属性, 201](#)
- [开发 Java 代码, 204](#)
- [配置 Java 转换设置, 208](#)
- [编译 Java 转换, 210](#)
- [修复编译错误, 210](#)

## Java 转换概览

通过 Java 转换扩展 PowerCenter 功能。Java 转换提供了一个简单的本地编程接口，用于使用 Java 编程语言定义转换功能。使用 Java 转换，无需高深的 Java 编程语言知识，也无需具备外部 Java 开发环境，即可快速定义简单或相对复杂的转换功能。Java 转换可以是被动或主动转换。

PowerCenter 客户端可使用 Java 开发工具包 (JDK) 编译 Java 代码，并为转换生成字节代码。PowerCenter 客户端会将字节代码存储在 PowerCenter 存储库中。

集成服务使用 Java 运行时环境 (JRE) 在运行时执行生成的字节代码。当集成服务使用 Java 转换运行会话时，该集成服务会使用 JRE 执行字节代码、处理输入行并生成输出行。

编写 Java 代码段来定义转换逻辑，以创建 Java 转换。请根据以下事件定义 Java 转换的转换行为：

- 转换接收输入行。
- 转换已处理所有输入行。
- 转换会收到事务通知，例如提交或回滚。

相关主题：

- [“Java 转换 API 引用” 页面上 212](#)
- [“Java 表达式” 页面上 224](#)

# 定义 Java 转换的步骤

完成以下步骤以写入和编译 Java 代码并修复 Java 转换中的编译错误：

- 1. 在 Transformation Developer 或 Mapping Designer 中创建转换。
- 2. 配置转换的输入和输出端口和组。在 Java 代码段中使用端口名称作为变量。
- 3. 配置转换属性。
- 4. 使用转换中的代码输入选项卡写入并编译转换的 Java 代码。
- 5. 查找并修复转换的 Java 代码中的编译错误。

# 主动和被动 Java 转换

创建 Java 转换后，可将其类型定义为主动或被动。

设置转换类型后，将无法对其进行更改。

Java 转换会对每个输入数据行运行一次在**输入输入行**选项卡上定义的 Java 代码。

Java 转换会按如下所示根据转换类型处理输出行：

- 处理完每个输入行后，被动 Java 转换会为转换中的每个输入行生成一个输出行。
- 主动 Java 转换会为转换中的每个输入行生成多个输出行。

使用 generateRow 方法可生成每个输出行。例如，如果转换包含两个输入端口，这两个端口分别代表开始日期和结束日期，则可以使用 generateRow 方法为开始日期和结束日期之间的每个日期生成一个输出行。

# 数据类型转换

Java 转换会根据 Java 转换端口类型将 PowerCenterDeveloper tool 数据类型转换为 Java 数据类型。

当 Java 转换读取输入行时，它会将输入端口数据类型转换为 Java 数据类型。

当 Java 转换写入输出行时，它会将 Java 数据类型转换为输出端口数据类型。

例如，在 Java 转换中，会对 Integer 数据类型的输入端口进行以下处理：

- 1. Java 转换会将输入端口的 Integer 数据类型转换为 Java 基元 Int 数据类型。
- 2. 在转换过程中，转换会将输入端口的值视为 Java 基元 Int 数据类型。
- 3. 当转换生成输出行时，它会将 Java 基元 Int 数据类型转换为 Integer 数据类型。

下表显示了 Java 转换是如何将 PowerCenterDeveloper tool 数据类型映射到 Java 基元和复杂数据类型的：

PowerCenter 数据类型	Java 数据类型
字符	字符串
Binary	byte[]
Long (INT32)	int
Double	double
Decimal	double BigDecimal

PowerCenter 数据类型	Java 数据类型
BIGINT	long
Date/Time	BigDecimal long（自格林尼治标准时间 1970 年 1 月 1 日 00:00:00.000 起的毫秒数）

Developer Tool 数据类型	Java 数据类型
array*	java.util.List
bigint	long
binary	byte[]
date/time	如果启用了纳秒处理，则为具有纳秒精度的 BigDecimal 如果禁用了纳秒处理，则为具有毫秒精度的 long（自格林尼治标准时间 1970 年 1 月 1 日 00:00:00.000 起的毫秒数）
decimal	如果禁用了高精度处理，则为精度为 15 的 double 如果启用了高精度处理，则为 BigDecimal
double	double
integer	int
map*	java.util.Map
string	String
struct*	自定义的 JavaBean 类，具有适用于 struct 字段元素的 getter 和 setter
text	String
* 仅在 Spark 引擎上受支持。	

在 Java 中，java.util.List、java.util.Map、String、byte[] 和 BigDecimal 数据类型为复杂数据类型。Double、int 和 long 数据类型为基元数据类型。

在 Developer tool 中，array、struct 和 map 数据类型为复杂数据类型。

Java 转换会将基元数据类型中的空值设置为零。您可以使用在输入行输入选项卡上的 isNull 和 setNull API 方法将输入端口中的空值设置为输出端口中的空值。有关示例，请参阅[“setNull”页面上 221](#)。

**注意:** 启用高精度时，decimal 数据类型将映射到 BigDecimal。BigDecimal 不能与某些运算符结合使用，例如 + 运算符。如果 Java 代码包含的表达式使用 decimal 端口，并且该端口与这些运算符之一结合使用，Java 代码将无法编译。

**注意:** 启用高精度时，decimal 数据类型将映射到 BigDecimal。BigDecimal 不能与某些运算符结合使用，例如 + 运算符。如果 Java 代码包含的表达式使用 decimal 端口或具有 decimal 数据类型元素的复杂端口，并且该端口与这些运算符之一结合使用，Java 代码将无法编译。

# 使用“Java 代码”选项卡

使用“Java 代码”选项卡定义、编译和修复 Java 代码中的编译错误。在代码输入选项卡中创建代码段。

定义 java 代码时，可以执行以下任务：

- 定义静态代码或静态块、实例变量和用户定义的方法。
- 定义 Java 表达式，并定义转换逻辑。
- 使用 Java 转换 API 方法和标准 Java 语言构造。
- 导入第三方 Java API、内建 Java 包或自定义 Java 包。可以使用 Java 包中的 Java 代码段。

开发代码段后，可以编译 Java 代码并在“输出”窗口中查看编译结果或查看完整的 Java 代码。

“Java 代码”选项卡包含以下组件：

- **导航器。**将输入或输出端口或 API 添加到代码段。导航器列出了转换的输入和输出端口、可用的 Java 转换 API 以及端口或 API 函数的说明。对于输入和输出端口，说明包含端口名称、类型、数据类型、精度和小数位数。对于 API 函数，说明包含 API 函数的语法和用法。  
导航器禁用了不可用于代码输入选项卡的任何端口或 API 函数。例如，无法从“导入包”代码输入选项卡添加端口或调用 API 函数。
- **代码窗口。**为转换开发 Java 代码。代码窗口使用基本的 Java 语法突出显示方法。
- **代码输入选项卡。**定义转换行为。每个代码输入选项卡都具有关联的代码窗口。要为代码输入选项卡输入 Java 代码，请单击该选项卡并在代码窗口中写入 Java 代码。
- **“定义表达式”链接。**打开用于创建 Java 表达式的“定义表达式”对话框。
- **“设置”链接。**打开“设置”对话框。“设置”对话框用于为第三方和自定义 Java 包设置类路径，以实现小数数据类型的高精度以及处理子秒数据。PowerCenter 客户端在编译 Java 代码时会包含此类路径中的文件。
- **“编译”链接。**编译转换的 Java 代码。Java 编译器中的输出（包含错误和参考消息）显示在“输出”窗口中。
- **“完整代码”链接。**打开“完整代码”窗口以显示 Java 转换的完整类代码。转换的完整代码包含已添加到 Java 转换类模板的代码输入选项卡中的 Java 代码。
- **输出窗口。**显示 Java 转换类的编译结果。可以在“输出”窗口中右键单击错误消息，查找“完整代码”窗口中 Java 转换类的代码段代码或完整代码中的错误。还可以在“输出”窗口中双击错误，以查找错误源。

## 配置端口

Java 转换可以具有输入端口、输出端口和输入/输出端口。在“端口”选项卡上可以创建和编辑组和端口。可为端口指定默认值。在向转换添加端口后，在 Java 代码段中使用端口名称作为变量。

## 创建组和端口

创建 Java 转换时，该转换包含一个输入组和一个输出组。

一个 Java 转换始终只有一个输入组和一个输出组。如果某个转换具有多个输入组或输出组，则该转换无效。您可以通过在组表头中键入来更改现有组名称。如果删除了一个组，您可以通过单击“创建输入组”或“创建输出组”图标来添加一个新的分组。

创建端口时，DesignerDeveloper 工具会将该端口添加到当前所选行或组的下方。显示在输入组下方的输入/输出端口也属于输出组。显示在输出组下方的输入/输出端口也属于输入组。



## 设置默认端口值

可以为 Java 转换中的端口定义默认值。

Java 转换会根据端口的数据类型将端口变量初始化为默认端口值。

### 输入和输出端口

Java 转换将初始化未连接的输入端口或输出端口的值，这些端口在 Java 代码段中未分配值。

Java 转换根据以下 Java 数据类型初始化端口：

**基元数据类型。**

如果为不为空的端口定义默认值，转换会将端口变量的值初始化为默认值。否则，会将端口变量的值初始化为 0。

**复杂数据类型**

如果为端口定义默认值，转换则会创建新的对象，并将该对象初始化为默认值。否则，转换会将端口变量初始化为空值。例如，如果为字符串端口定义默认值，转换则会创建新的 String 对象，并将该 String 对象初始化为默认值。

**注意:** 如果在 Java 代码中访问具有空值的输入端口变量，将出现 NullPointerException。

您可以启用输入端口作为分区键和排序键，您可以指定排序方向。数据集成服务对数据进行分区，并按排序键和排序方向在每个分区中排序数据。当转换范围设置为“所有输入”时，分区键和排序键有效。

使用下列属性对数据进行分区和排序：

**分区键**

确定要分组到同一分区中的数据行的输入端口。

**排序键**

确定每个分区内的排序条件的输入端口。

**方向**

升序或降序。默认设置是升序。

### 输入/输出端口

Java 转换将输入/输出端口视为传递端口。如果未在转换的 Java 代码中为端口设置值，输出值将与输入值相同。Java 转换初始化输入/输出端口值的方式与其初始化输入端口的方式相同。

如果在 Java 代码中为输入/输出端口设置了端口变量值，Java 转换在生成输出行时会使用该值。如果未设置输入/输出端口的值，则 Java 转换在生成输出行时，对于简单数据类型会将端口变量的值设置为 0，对于复杂数据类型会将端口变量的值设置为空。

## 配置 Java 转换属性

Java 转换包含适用于转换代码和转换的属性。如果您在 Transformation Developer 中创建 Java 转换，则当在映射中使用该转换时，可以替代转换属性。

下表介绍了 Java 转换属性：

属性	说明
语言	用于转换代码的语言。您无法更改此值。
类名称	用于转换的 Java 类的名称。您无法更改此值。
跟踪级别	<p>此转换的会话日志中显示的详细信息量。使用以下跟踪级别：</p> <ul style="list-style-type: none"> <li>- 简洁</li> <li>- 普通</li> <li>- 详细初始化</li> <li>- 详细数据</li> </ul> <p>默认值为“普通”。</p>
是否可分区	<p>管道中的多个分区可以使用此转换。请使用以下选项：</p> <ul style="list-style-type: none"> <li>- 否。无法对转换进行分区。转换和同一渠道中的其他转换只有一个分区。如果转换一起处理所有输入数据（例如数据清理），可选择“否”。</li> <li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。如果转换的不同分区必须共享内存中的对象，可选择“本地”。</li> <li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li> </ul> <p>默认值为“否”。</p>
必须阻止输入	与转换关联的过程必须能够阻止传入数据。默认为“已启用”。
是否活动	<p>转换可为每个输入行生成多个输出行。</p> <p>不能在创建 Java 转换后更改此属性。如果需要更改此属性，请创建一个新的 Java 转换。</p>
更新策略转换	<p>转换可定义输出行的更新策略。您可以为主动 Java 转换启用此属性。</p> <p>默认为“已禁用”。</p>
转换范围	<p>集成服务将转换逻辑应用于传入数据的方法。请使用以下选项：</p> <ul style="list-style-type: none"> <li>- 行</li> <li>- 事务</li> <li>- 全部输入</li> </ul> <p>对于被动转换，此属性始终为“行”。主动转换的默认设置为“所有输入”。</p>
生成事务	<p>转换将生成事务行。您可以为主动 Java 转换启用此属性。</p> <p>默认为“已禁用”。</p>
输出是可重复的	<p>会话运行间输出数据的顺序一致。</p> <ul style="list-style-type: none"> <li>- 从不。会话运行之间的输出数据顺序不一致。</li> <li>- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。</li> <li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li> </ul> <p>主动转换的默认设置为“从不”。对于被动转换，默认设置为“基于输入顺序”。</p>

属性	说明
要求每个分区一个线程	单个线程处理每个分区的数据。 您无法更改此值。
输出具有确定性	转换将在会话运行之间生成一致的输出数据。启用此属性可对使用此转换的会话执行恢复操作。 默认为“已启用”。

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

## 使用事务控制

可以使用以下属性为 Java 转换定义事务控制：

- **转换范围。** 确定集成服务如何将转换逻辑应用到传入数据。
- **生成事务。** 指示用于转换的 Java 代码生成事务行，然后将它们传递到输出组中。

### 转换范围

可以配置集成服务如何将转换逻辑应用到传入数据。可以选择以下值之一：

- **行。** 一次将转换逻辑应用至一行数据。当转换的结果取决于某一行数据时，选择“行”。对于被动转换，必须选择“行”。
- **事务。** 将转换逻辑应用至事务中的所有行。当转换的结果取决于同一事务中的所有行（但不取决于其他事务中的行）时，选择“事务”。例如，当 Java 代码对某一事物中的数据执行汇总计算时，可以选择“事务”。
- **全部输入。** 将转换逻辑应用至所有传入数据。选择“全部输入”时，集成服务将删除事务边界。当转换的结果取决于源中的所有数据行时，选择“全部输入”。例如，当转换的 Java 代码对所有传入数据排序时，可以选择“全部输入”。

### 生成事务

您可以在主动 Java 转换中定义 Java 代码以生成事务行，如提交和回滚行。要生成事务行，请启用该转换。

当您配置转换以生成事务行时，集成服务会像对待事务控制转换一样对待 Java 转换。映射中适用于事务控制转换的多数规则也同样适用于 Java 转换。例如，当您配置 Java 转换以生成事务行时，无法串联包含转换的管道或管道分支。

当您使用配置为生成事务行的 Java 转换编辑或创建会话时，请为其配置用户定义的提交。

### 相关主题：

- [“提交”页面上 213](#)
- [“rollBack”页面上 220](#)

## 设置更新策略

使用活动 Java 转换为映射设置更新策略。可以在以下级别设置更新策略：

- **在 Java 代码内。** 可以写入 Java 代码以为输出行设置更新策略。Java 代码可以标记要插入、更新、删除或拒绝的行。有关更新策略的详细信息，请参阅 [“setOutRowType”页面上 221](#)。

- **在映射内。**在映射中使用 Java 转换以标记要插入、更新、删除或拒绝的行。为 Java 转换选择“更新策略转换”属性。
- **在会话内。**配置会话以将源行视为数据驱动。

如果未配置 Java 转换来定义更新策略，或未将会话配置为数据驱动，则集成服务不会使用 Java 代码来标记输出行，而是会将输出行标记为插入。

# 开发 Java 代码

使用代码输入选项卡输入定义 Java 转换功能的 Java 代码段。可以使用代码输入选项卡写入 Java 代码，以便导入 Java 包、写入帮助程序代码、定义 Java 表达式及写入定义特定转换事件转换行为的 Java 代码。您可以按任何顺序在代码输入选项卡上开发代码段。

在以下代码输入选项卡中输入 Java 代码：

- **导入包。**导入第三方 Java 包、内置 Java 包或自定义 Java 包。
- **帮助程序代码。**定义可用于“导入包”之外所有选项卡的变量和方法。
- **输入行时。**定义其在接收到输入行时的转换行为。
- **数据结尾时。**定义其处理完所有输入数据时的转换行为。
- **在接收事务中。**定义其在接收到事务通知时的转换行为。用于活动的 Java 转换。
- **Java 表达式。**定义调用 PowerCenter 表达式的 Java 表达式。在“帮助程序代码”、“输入行时”、“数据结尾时”和“在接收事务中”代码输入选项卡中可以使用 Java 表达式。

在“输入行时”选项卡上访问输入数据和设置输出数据。对于主动转换，还可以在“数据结尾时”和“在接收事务中”选项卡上设置输出数据。

## 创建 Java 代码段

要创建用于定义转换行为的 Java 代码段，请使用 **Java 代码** 代码输入选项卡上的 **代码 Java 代码** 窗口。

1. 单击相应的代码输入选项卡。

下表介绍了可在 **Java** 视图的代码输入选项卡上完成的任务：

选项卡	说明
导入	为主动或被动 Java 转换导入第三方、内置和自定义 Java 包。导入包后，可在其他代码输入选项卡上使用这些包。
帮助程序	在主动或被动 Java 转换中为 Java 转换类声明用户定义的变量和方法。声明变量和方法后，可在除导入选项卡之外的任何其他代码输入选项卡中使用这些变量和方法。
输入	定义主动或被动 Java 转换在收到输入行时的行为。在此选项卡上定义的 Java 代码会对每个输入行运行一次。 在此选项卡上，您还可以访问并使用输入和输出端口数据、变量和 Java 转换 API 方法。

选项卡	说明
结尾	定义主动或被动 Java 转换在处理完所有输入数据后的行为。 在此选项卡上，您还可以为主动转换设置输出数据，并调用 Java 转换 API 方法。
函数	使用 Java 编程语言定义在 Java 转换中调用表达式的函数。例如，您可以定义一个函数，该函数可调用一个表达式来查找输入或输出端口的值，或查找 Java 转换变量的值。 在 <b>函数</b> 选项卡上，您可以手动定义函数，也可以单击 <b>新建函数</b> 调用 <b>定义函数</b> 对话框，此对话框可用于轻松定义函数。
优化器接口	定义早期选择或推入筛选器优化。在导航器中选择优化方法。更新代码段以启用优化。定义输入端口以及关联的输出端口来推送筛选器逻辑。
完整代码	只读。在此选项卡上，可以查看并编译 Java 转换的完整类代码。

- 要访问代码段中的输入或输出列变量，请双击导航器中的端口名称请展开导航器中的**输入**或**输出**列表，然后双击端口名称。
- 要调用代码段中的 Java 转换 API，请在导航器中双击 API 的名称。如有需要，请配置导航器中相应的 API 输入值请展开导航器中的**可调用 API** 列表，然后双击方法的名称。如有需要，请为方法配置相应的输入值。
- 根据代码段代码输入选项卡类型编写相应的 Java 代码。  
在**完整代码**选项卡的**完整代码 Java 代码**窗口中查看 Java 转换的完整类代码。

## 导入 Java 包

在**导入包导入**选项卡上，可以为主动或被动 Java 转换导入 Java 包。

可以导入第三方、内置或自定义 Java 包。导入 Java 包后，可在其他代码输入选项卡上使用这些导入的包。

**注意:** 在**导入包导入**选项卡上，不能声明或使用静态变量、实例变量或用户方法。

在 PowerCenter 客户端 Developer 工具中，当导出或导入包含 Java 转换的元数据时，不会导出或导入包含此 Java 转换所需的第三方或自定义包的 jar 文件或类文件。

如果要导入包含 Java 转换的元数据，则必须将包含所需的第三方或自定义包的 jar 文件或类文件复制到 PowerCenter 客户端和集成服务节点 Developer 工具客户端和数据集成服务节点。

例如，要导入 Java I/O 包，请在**导入包导入**选项卡上输入以下代码：

```
import java.io.*;
```

如果要导入非标准 Java 包，请将包或类添加到 Java 转换设置的类路径中。

### 相关主题：

- [“配置 Java 转换设置” 页面上 208](#)

## 定义帮助程序代码

在**帮助程序代码帮助程序**选项卡上，可以为主动或被动 Java 转换中的 Java 转换类声明用户定义的变量和方法。

在**帮助程序代码帮助程序**选项卡上声明变量和方法后，可在除**导入包导入**选项卡之外的任何其他代码输入选项卡中使用这些变量和方法。

在**帮助程序代码帮助程序**选项卡上，可以声明以下类型的代码、变量和方法：

- 静态代码和静态变量。

在静态块中，可以声明静态变量和静态代码。某个可重用 Java 转换在映射中的所有实例以及某个会话中的所有分区会共享静态代码和变量。在 Java 转换中，静态代码会在任何其他代码之前运行。

例如，以下代码用于声明一个静态变量来存储某个 Java 转换在映射中的所有实例的错误阈值：

```
static int errorThreshold;
```

使用此变量可存储该转换的错误阈值，并从该 Java 转换在映射中的所有实例以及某个会话中的任何分区访问该阈值。

**注意：**必须在一个多分区会话或一个可重用 Java 转换中同步静态变量。

- **实例变量。**

可以声明分区级实例变量。某个可重用 Java 转换在映射中的多个实例或某个会话中的多个分区不会共享实例变量。请使用前缀来声明实例变量以防止发生冲突，并初始化非原始实例变量。

例如，以下代码使用一个布尔变量来确定是否生成输出行：

```
// boolean to decide whether to generate an output row
// based on validity of input
private boolean generateRow;
```

- **用户定义的静态方法或实例方法。**

扩展 Java 转换的功能。在**帮助程序代码帮助程序**选项卡上声明的 Java 方法可使用或修改输出变量或本地声明的实例变量。不能在**帮助程序代码帮助程序**选项卡上从 Java 方法中访问输入变量。

例如，在**帮助程序代码帮助程序**选项卡上使用以下代码可声明一个函数以添加两个整数：

```
private int myTXAdd (int num1,int num2)
{
    return num1+num2;
}
```

## “在输入行” 选项卡

使用“在输入行”选项卡可定义 Java 转换在收到输入行时的行为。此选项卡中的 Java 代码为每个输入行执行一次。只能在“在输入行”选项卡中访问输入行数据。

从“在输入行”选项卡访问和使用以下输入和输出端口数据、变量以及方法：

- **输入端口和输出端口变量。** 使用端口名称作为变量名称，以变量的形式访问输入和输出端口数据。例如，如果“in\_int”是一个整数输入端口，则可以使用 Java 原始数据类型 Int 以变量的形式引用“in\_int”来访问该端口的数据。无需将输入和输出端口声明为变量。

请不要为输入端口变量赋值。如果在“在输入行”选项卡中为输入变量赋值，则无法在当前行上获取相应端口的输入数据。

- **实例变量和用户定义的方法。** 使用在“帮助程序代码”选项卡中声明的任何实例、静态变量或用户定义的方法。

例如，一个主动 Java 转换具有两个 Integer 数据类型的输入端口：BASE\_SALARY 和 BONUSES，以及一个 Integer 数据类型的输出端口：TOTAL\_COMP。您在“帮助程序代码”选项卡中创建了一个用户定义的方法 myTXAdd，以将两个整数相加并返回结果。在“在输入行”选项卡中使用以下 Java 代码可将输入端口的总值分配给输出端口，并生成输出行：

```
TOTAL_COMP = myTXAdd (BASE_SALARY,BONUSES);
generateRow();
```

当 Java 转换收到输入行时，它会添加 BASE\_SALARY 和 BONUSES 输入端口的值，将该值分配给 TOTAL\_COMP 输出端口，并生成输出行。

- **Java 转换 API 方法。** 可以调用 Java 转换提供的 API 方法。

## “在数据末尾” 选项卡

在主动或被动 Java 转换中使用“在数据末尾”选项卡来定义 Java 转换处理了所有输入数据后的行为。要在“在数据末尾”选项卡中生成输出行，请将转换范围设置为“事务”或“全部输入”。不能在此选项卡中访问或设置输入端口变量的值。

从“在数据末尾”选项卡访问和使用以下变量和方法：

- **输出端口变量。**使用输出端口的名称作为变量来访问或设置主动 Java 转换的输出数据。
- **实例变量和用户定义的方法。**使用在“帮助程序代码”选项卡中声明的任何实例变量或用户定义的方法。
- **Java 转换 API 方法。**调用 Java 转换提供的 API 方法。使用提交和回滚 API 方法生成事务。

例如，使用以下 Java 代码在达到数据末尾时向会话日志写入信息：

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

## “在接收事务中” 选项卡

使用主动 Java 转换中的“在接收事务中”选项卡可以定义主动 Java 转换收到事务通知时的行为。“在接收事务中”选项卡的代码段仅在转换的事务范围设置为“事务”时执行。不能在此选项卡中访问或设置输入端口变量的值。

从“在接收事务中”选项卡访问和使用以下输出数据、变量和方法：

- **输出端口变量。**使用输出端口的名称作为变量来访问或设置输出数据。
- **实例变量和用户定义的方法。**使用在“帮助程序代码”选项卡中声明的任何实例变量或用户定义的方法。
- **Java 转换 API 方法。**调用 Java 转换提供的 API 方法。使用提交和回滚 API 方法生成事务。例如，在转换收到事务后使用以下 Java 代码生成事务：

```
commit();
```

## 使用 Java 代码解析平面文件

可以开发 Java 代码来解析平面文件。使用 Java 代码从不同架构或 JMS 消息的平面文件中提取特定列的数据。

例如，您想要从带分隔符的平面文件中读取前两列数据。创建一个映射，从带分隔符的平面文件中读取数据并将数据传递到一个或多个输出端口。

该映射包含以下组件：

- **源定义。**源是带分隔符的平面文件。配置源以将平面文件的行以字符串的形式传递到 Java 转换。源文件具有以下数据：

```
1a,2a,3a,4a,5a,6a,7a,8a,9a,10a
1b,2b,3b,4b,5b,6b,7b,8b,9b
1c,2c,3c,4c,5c,6c,7c
1d,2d,3d,4d,5d,6d,7d,8d,9d,10d
```

- **Java 转换。**在“Java 代码”选项卡上定义 Java 转换功能。

使用“Java 代码”的“导入包”选项卡导入 Java 包。在“导入包”选项卡上输入以下代码：

```
import java.util.StringTokenizer;
```

使用“Java 代码”选项卡的“在输入行”选项卡从字符串中读取每行数据，并将数据传递到输出端口。在“在输入行”选项卡上输入以下代码，以检索前两列数据：

```
StringTokenizer st1 = new StringTokenizer(row, ",");
f1 = st1.nextToken();
f11= st1.nextToken();
```



- **目标定义。**将目标配置为从 Java 转换接收数据行。运行包含映射的工作流后，目标会包含两列数据。目标文件具有以下数据：

```
1a,2a  
1b,2b  
1c,2c  
1d,2d
```

## 配置 Java 转换设置

可以配置 Java 转换设置，从而为第三方和自定义 Java 包设置类路径，并为 Decimal 数据类型启用高精度。还可以配置转换以处理子秒数据。

### 配置类路径

在“导入包”选项卡中导入非标准 Java 包时，请将 PowerCenter 客户端和集成服务的类路径设置为与该 Java 包关联的每个 JAR 文件或类文件。在 UNIX 上，使用冒号来分隔类路径条目。在 Windows 上，使用分号来分隔类路径条目。在 PowerCenter 客户端和集成服务节点上必须可以访问这些 JAR 或类文件。

例如，您在“导入包”选项卡中导入了 Java 包 converter，并在 converter.jar 中对该包进行了定义。您必须将 converter.jar 添加到类路径中，才能为 Java 转换编译 Java 代码。

不需要为内置 Java 包设置类路径。例如，java.io 为内置 Java 包。如果导入了 java.io，则无需为 java.io 设置类路径。

### 为集成服务配置类路径

可以将 JAR 文件或类文件目录添加到集成服务类路径中。要在集成服务节点上设置类路径，完成以下任务之一：

- **配置“Java 类路径”会话属性。**使用“Java 类路径”会话属性设置类路径。此类路径将应用到会话。
- **配置 Java SDK 类路径。**在 Informatica Administrator 中“集成服务”属性的“进程”选项卡上配置 Java SDK 类路径。此设置将应用到集成服务上运行的所有会话。
- **配置 CLASSPATH 环境变量。**在集成服务节点上设置 CLASSPATH 环境变量 设置环境变量后重新启动集成服务。此设置将应用到集成服务上运行的所有会话。

### 在 UNIX 上配置集成服务的类路径

要在 UNIX 上配置 CLASSPATH 环境变量：

- ▶ 在 UNIX C shell 环境中，键入：

```
setenv CLASSPATH <classpath>
```

在 UNIX Bourne shell 环境中，键入：

```
CLASSPATH = <classpath>  
export CLASSPATH
```

### 在 Windows 上配置集成服务的类路径

要在 Windows 上配置 CLASSPATH 环境变量，请执行以下操作：

- ▶ 输入环境变量 CLASSPATH，并将值设置为默认类路径。



## 为 PowerCenterDeveloper 工具客户端配置类路径

可以将 jar 文件或类文件目录添加到 PowerCenter 客户端 Developer 工具客户端的类路径中。

要为运行 PowerCenter 客户端 Developer 工具客户端的计算机设置类路径，请完成以下任务之一：

- 配置 CLASSPATH 环境变量。在 PowerCenter 客户端 Developer 工具客户端客户端计算机上设置 CLASSPATH 环境变量。该变量适用于在该计算机上运行的所有 Java 进程。
- 对于不可重用 Java 转换，在 Java 转换的设置高级属性中配置配置类路径。此类路径适用于包含此 Java 转换的会话映射。PowerCenter 客户端 Developer 工具客户端客户端在编译 Java 代码时会包含此类路径中的文件。

要向 Java 转换中的类路径添加 jar 或类文件目录，请完成以下步骤：

1. 在 **Java 代码高级**选项卡上，单击**设置**链接单击**类路径**旁边**值**列中的向下箭头图标。  
此时将显示**设置编辑类路径**对话框。
2. 要添加类路径，请完成以下步骤：
  - a. 单击**添加**。  
此时将显示**另存为**窗口。
  - b. 在**另存为**窗口中，导航到 jar 文件所在的目录。
  - c. 单击**确定**。  
此时将在**编辑类路径**对话框中显示类路径。
3. 单击**添加类路径**下方的“浏览”，选择已导入包的 jar 文件或类文件目录。单击**确定**。
4. 单击**添加**。  
jar 文件或类文件目录将显示在该转换的 jar 文件和类文件目录列表中。
5. 要删除 jar 文件或类文件目录，请选择 jar 文件或类文件目录，然后单击**删除**。  
该目录将不再显示在目录列表中。

## 启用高精度

默认情况下，Java 转换将类型为 Decimal 的端口转换为双精度数据类型，精度为 15。如果要处理精度超过 15 的小数数据类型，请启用高精度以通过 Java 类 BigDecimal 处理小数端口。

启用高精度时，可以通过 BigDecimal 处理精度小于 28 的小数端口。Java 转换会将精度大于 28 的小数数据转换为双精度数据类型。Java 转换表达式会处理二进制、整数、双精度和字符串数据。Java 转换表达式无法处理 Bigint 数据。

例如，Java 转换具有类型为小数的输入端口，其收到值 40012030304957666903。如果启用高精度，则该端口的值会按照显示内容进行处理。如果不启用高精度，则该端口的值为  $4.00120303049577 \times 10^{19}$ 。

**注意：**如果为 Java 转换启用高精度，还会为包括 Java 转换的会话启用高精度。如果在 Java 转换中启用了高精度并且其未在会话中启用，则会话可能会失败并显示以下错误：

```
[ERROR]Failed to bind column with index <index number> to datatype <datatype name>.
```

## 处理子秒

在 Java 代码中对亚秒数据的处理可达到纳秒级精度。配置设置以便在日期时间值中使用纳秒时，生成的 Java 代码会将转换日期/时间数据类型转换为具有纳秒精度的 Java BigDecimal 数据类型。

默认情况下，生成的 Java 代码会将转换日期/时间数据类型转换为具有毫秒精度的 Java Long 数据类型。

# 编译 Java 转换

PowerCenter 客户端 Developer 工具可使用 Java 编译器编译 Java 代码，并为转换生成字节代码。

Java 编译器将编译 Java 代码，并在代码输入选项卡上**输出窗口**编译属性的**结果**窗口中显示编译结果。Java 编译器会与 PowerCenter 客户端 Developer 工具一起安装在 java/bin 目录中。

要编译 Java 转换的完整代码，请在 **Java 代码完整代码** 选项卡的**编译**属性中单击**编译**。

创建 Java 转换时，此转换将包含一个 Java 类，用于定义 Java 转换的基本功能。该 Java 类的完整代码包含此转换的模板类代码以及在代码输入选项卡上定义的 Java 代码。

编译 Java 转换时，PowerCenter 客户端 Developer 工具会将代码输入选项卡中的代码添加至此转换的模板类，以为此转换生成完整类代码。然后，PowerCenter 客户端 Developer 工具会调用 Java 编译器来编译完整类代码。Java 编译器将编译此转换，并为此转换生成字节代码。

编译结果将显示在**输出结果**窗口中。使用编译结果可识别并查找 Java 代码错误。

**注意：**在 Java 转换中单击**确定**后，也会编译此转换。

## 修复编译错误

可以在“输出”窗口中确定 Java 代码错误并找到 Java 转换的 Java 代码错误源。Java 转换错误可能是由于代码输入选项卡的代码错误所致，或者可能由于 Java 转换类的完整代码中存在错误所致。

要为 Java 转换排除故障，请执行以下操作：

- **找到错误源。**可以在 Java 代码段代码中或在转换的完整类代码中找到错误源。
- **识别错误类型。**使用输出窗口中的编译结果以及错误的位置来识别错误类型。

确定错误的源和类型后，在代码输入选项卡中修复 Java 代码并再次编译转换。

## 定位编译错误来源

编译 Java 转换时，“输出”窗口将显示编译的结果。使用编译的结果可标识编译错误。使用“输出”窗口定位错误来源时，PowerCenter 客户端将在代码输入选项卡或“完整代码”窗口中突出显示错误来源。

您可以在“完整代码”窗口中定位错误，但不能在“完整代码”窗口中编辑 Java 代码。要修复在“完整代码”窗口中定位的错误，请在相应的代码输入选项卡中修改代码。您可能需要使用“完整代码”窗口查看由于向转换的完整类代码添加用户代码而导致的错误。

使用“输出”窗口中的编译结果标识以下位置中的错误：

- 代码输入选项卡
- “完整代码”窗口

### 在代码输入选项卡中找到错误

要在代码输入选项卡中找到错误的源，请右键单击“输出”窗口中的错误，然后选择查看代码段中的错误，或双击“输出”窗口中的错误。PowerCenter 客户端会在适当的代码输入选项卡中突出显示错误的源。

### 在“完整代码”窗口中定位错误

要在“完整代码”窗口中定位错误源，请右键单击“输出”窗口中的错误并在“完整代码”中选择“查看错误”，或者在“输出”窗口中双击该错误。PowerCenter 客户端将在“完整代码”窗口中突出显示该错误源。

## 标识编译错误来源

编译错误可能会在用户代码中显示为错误的结果。

用户代码中的错误还可能会在该类的非用户代码中生成错误。Java 转换的用户代码和非用户代码中均会产生编译错误。

### 用户代码错误

代码输入选项卡上的用户代码可能发生错误。用户代码错误包含标准 Java 语法错误以及语言错误。

当 PowerCenter 客户端 Developer 工具将用户代码从代码输入选项卡添加至完整类代码时，也可能发生用户代码错误。

例如，Java 转换有一个输入端口，名称为 int1，数据类型为 integer。该类的完整代码使用以下代码声明输入端口变量：

```
int int1;
```

但是，如果在**输入行输入**选项卡上使用了相同的变量名称，Java 编译器将发出一个错误，要求对变量进行重新声明。要修复此错误，请在**输入行输入**选项卡上重命名该变量。

### 非用户代码错误

代码输入选项卡上的用户代码可在非用户代码中导致错误。

例如，某个 Java 转换具有输入端口 int1 和输出端口 out1，数据类型为 Integer。在**输入行输入**代码输入选项卡上输入以下代码以计算输入端口 int1 的目标并将其指定给输出端口 out1：

```
int interest;  
interest = CallInterest(int1); // calculate interest  
out1 = int1 + interest;  
}
```

编译转换时，PowerCenter 客户端 Developer 工具会将**输入行输入**代码输入选项卡上的代码添加到此转换的完整类代码中。Java 编译器编译 Java 代码时，不匹配的大括号会导致完整类代码中的某个方法过早结束，从而导致 Java 编译器发出错误。

## 第 13 章

# Java 转换 API 引用

本章包括以下主题：

- [Java 转换 API 方法概览, 212](#)
- [提交, 213](#)
- [defineJExpression, 214](#)
- [failSession, 214](#)
- [generateRow, 215](#)
- [getInRowType, 215](#)
- [getMetadata, 216](#)
- [incrementErrorCount, 217](#)
- [invokeJExpression, 217](#)
- [isNull, 218](#)
- [logError, 219](#)
- [logInfo, 219](#)
- [resetNotification, 220](#)
- [rollBack, 220](#)
- [setNull, 221](#)
- [setOutRowType, 221](#)
- [storeMetadata, 222](#)

## Java 转换 API 方法概览

在 Java 转换的 **Java 代码**选项卡上在编辑器的 **Java** 视图的代码输入选项卡上，可以将 API 方法添加到 Java 代码中以定义转换行为。

要将 API 方法添加到代码中，请在代码输入选项卡上的导航器中展开**可调用 API** 列表，然后双击要添加到代码中的方法的名称。

此外，也可以将方法从导航器拖动到 Java 代码段中或在 Java 代码段中手动输入 API 方法。

可以将以下 API 方法添加至 Java 转换中的 Java 代码：

提交

生成事务。

defineJExpression

定义 Java 表达式。

failSession

引发异常并显示错误消息，会话映射失败。

generateRow

为主动 Java 转换生成输出行。

getInRowType

返回转换中当前行的输入类型。

incrementErrorCount

递增会话映射的错误计数。

invokeJExpression

调用使用 defineJExpression 方法定义的 Java 表达式。

isNull

检查输入列中是否存在空值。

logError

将错误消息写入会话日志中。

logInfo

将信息性消息写入会话日志中。

resetNotification

如果数据集成服务计算机在重启模式下运行，则在映射运行后将重置 Java 代码中使用的变量。

回滚

生成回滚事务。

setNull

将主动或被动 Java 转换中输出列的值设置为空值。

setOutRowType

设置输出行的更新策略。可以将行标记为插入、更新或删除。

## 提交

生成事务。

可在除“导入包”或“Java 表达式”代码输入选项卡之外的任意选项卡中使用 commit。只能在配置为生成事务的主动转换中使用 commit。如果在未配置为生成事务的主动转换中使用 commit，则集成服务将引发错误且会话将失败。

请使用以下语法：

```
commit();
```

使用以下 Java 代码可实现在 Java 转换每处理 100 行后生成事务，然后将 rowsProcessed 计数器设置为 0：

```
if (rowsProcessed==100) {
```

```

        commit();

        rowsProcessed=0;
    }

```

## defineJExpression

定义表达式，包括表达式字符串和输入参数。defineJExpression 方法的参数包括：一个 JExprParamMetadata 对象数组（包含输入参数）以及一个用于定义表达式语法的字符串值。

请使用以下语法：

```

defineJExpression(
    String expression,
    Object[] paramMetadataArray
);

```

下表介绍了参数：

参数	类型	数据类型	说明
表达式	输入	String	表示表达式的字符串。
paramMetadataArray	输入	Object[]	JExprParamMetadata 对象数组，其中包含表达式的输入参数。

可以将 defineJExpression 方法添加到除**导入**和**函数**选项卡之外的任何代码输入选项卡上的 Java 代码中。

要使用 defineJExpression 方法，必须实例化 JExprParamMetadata 对象数组，该数组代表表达式的输入参数。可以为这些参数设置元数据值，然后将该数组作为参数传递给 defineJExpression 方法。

例如，以下 Java 代码可创建一个表达式，用于查找两个字符串的值：

```

JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
defineJExpression(":lkp.mylookup(x1,x2)",params);

```

**注意：**必须对连续传递给表达式的参数进行编号，这些参数必须以字母 x 开头。例如，要将三个参数传递给表达式，请将这些参数分别命名为 x1、x2 和 x3。

## failSession

引发异常并显示错误消息，会话映射失败。

请使用以下语法：

```

failSession(String errorMessage);

```

下表介绍了参数：

参数	参数类型	数据类型	说明
errorMessage	输入	String	错误消息字符串。

可使用 failSession 方法结束会话映射。不要在代码输入选项卡上的 try/catch 块中使用 failSession 方法。

可以将 failSession 方法添加到除**导入包**和**函数 Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

以下 Java 代码显示了如何测试 input1 输入端口是否为空值，如果是空值，则会话映射将失败：

```
if(isNull("input1")) {
    failSession( "Cannot process a null value for port input1." );
}
```

# generateRow

为主动 Java 转换生成输出行。

请使用以下语法：

```
generateRow();
```

调用 generateRow 方法时，Java 转换将使用输出端口变量的当前值生成输出行。如果要生成与某输入行对应的多个行，可以为每个输入行多次调用 generateRow 方法。如果未在主动 Java 转换中使用 generateRow 方法，则该转换不会生成输出行。

可以将 generateRow 方法添加到除**导入包**和**函数 Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

只能在主动转换中调用 generateRow 方法。如果在被动转换中调用 generateRow 方法，则会话数据集成服务将生成错误。

使用以下 Java 代码可实现生成一个输出行、修改输出端口的值和生成其他输出行：

```
// Generate multiple rows.
if(!isNull("input1") && !isNull("input2"))
{
    output1 = input1 + input2;
    output2 = input1 - input2;
}
generateRow();
// Generate another row with modified values.
output1 = output1 * 2;
output2 = output2 * 2;
generateRow();
```

# getInRowType

返回转换中当前行的输入类型。该方法返回要插入、更新、删除或拒绝的值。

请使用以下语法：

```
rowType getInRowType();
```

下表介绍了参数：

参数	参数类型	数据类型	说明
rowType	输出	String	返回更新策略类型，该类型是以下值之一： - DELETE - INSERT - REJECT - UPDATE

可以将 `getInRowType` 方法添加到**输入输入行**代码输入选项卡上的 Java 代码中。

可以在配置为设置更新策略的主动转换中使用 `getInRowType` 方法。如果在未配置为设置更新策略的主动转换中调用此方法，则会话数据集成服务将生成错误。

使用以下 Java 代码可完成以下操作：

- 将当前输入行类型传播到输出行。
- 如果 `input1` 输入端口的值大于 100，则将输出行类型设置为 `DELETE`。

```
// Set the value of the output port.  
output1 = input1;  
// Get and set the row type.  
String rowType = getInRowType();  
setOutRowType(rowType);  
// Set row type to DELETE if the output port value is > 100.  
if(input1 > 100  
    setOutRowType(DELETE);
```

# getMetadata

在运行时检索 Java 转换元数据。 `getMetadata` 方法检索使用 `storeMetadata` 方法保存的元数据，例如从优化器传递至 `pushFilter` 函数中 Java 转换的筛选条件。

请使用以下语法：

```
getMetadata (String key);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
键	输入	String	确定元数据。 <code>getMetadata</code> 方法使用键来确定要检索的元数据。

可以将 `getMetadata` 方法添加到以下代码输入选项卡上的 Java 代码中：

- 帮助程序
- 输入
- 结尾
- 优化器接口
- 函数



可以配置 `getMetadata` 方法检索用于推入优化的筛选条件。 `getMetadata` 方法可以从优化器中检索存储的每个筛选条件。

```
// Retrieve a filter condition
String mydata = getMetadata ("FilterKey");
```

# incrementErrorCount

递增会话的错误计数。 如果错误计数达到会话的错误阈值，则会话映射将失败。

请使用以下语法：

```
incrementErrorCount(int nErrors);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
nErrors	输入	Integer	用于递增会话错误计数的数字。

可以将 `incrementErrorCount` 方法添加到除**导入包**和**函数 Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

以下 Java 代码显示了转换的输入端口存在空值时如何递增错误计数：

```
// Check if input employee id and name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);
    // if input employee id and/or name is null, don't generate a output row for this input row
    generateRow = false;
}
```

# invokeJExpression

调用表达式并返回该表达式的值。

请使用以下语法：

```
(datatype)invokeJExpression(
    String expression,
    Object[] paramMetadataArray);
```

`invokeJExpression` 方法的输入参数是字符串值，该值表示表达式和包含表达式输入参数的对象数组。

下表介绍了参数：

参数	参数类型	数据类型	说明
表达式	输入	String	表示表达式的字符串。
paramMetadataArray	输入	Object[]	包含表达式的输入参数的对象数组。

可以将 `invokeJExpression` 方法添加到除**导入**和**函数导入包**和 **Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

使用 `invokeJExpression` 方法时，请遵循以下规则和准则：

- 返回数据类型。 `invokeJExpression` 方法的返回数据类型是一个对象。 您必须使用相应的数据类型转换函数的返回值。  
可以返回数据类型为 `Integer`、`Double`、`String` 和 `byte[]` 的值。
- 行类型。 `invokeJExpression` 方法的返回值的行类型为 `INSERT`。  
要为该返回值使用其他行类型，请使用高级接口。
- 空值。 如果将空值作为参数传递或者 `invokeJExpression` 方法的返回值为空，则将该值视为空指示符。  
例如，如果表达式的返回值为空且返回数据类型为 `String`，则返回一个具有空值的字符串。
- `Date` 数据类型。 必须将数据类型为 `Date` 的输入参数转换为 `String` 数据类型。  
要将表达式中的字符串用作 `Date` 数据类型，请使用 `to_date()` 函数将字符串转换为 `Date` 数据类型。  
或者，您必须将返回 `Date` 数据类型的所有表达式的返回类型转换为 `String` 数据类型。

以下示例将字符串“John”与“Smith”连接，并返回字符串“John Smith”：

```
(String)invokeJExpression("concat(x1,x2)", new Object [] { "John ", "Smith" });
```

**注意：**必须对连续传递给表达式的参数进行编号，这些参数必须以字母 `x` 开头。例如，要将三个参数传递给表达式，请将参数分别命名为 `x1`、`x2` 和 `x3`。

# isNull

检查输入列的值是否为空值。

请使用以下语法：

```
Boolean isNull(String strColName);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
<code>strColName</code>	输入	<code>String</code>	输入列的名称。

可以将 `isNull` 方法添加到**输入输入行**代码输入选项卡上的 Java 代码中。

以下 Java 代码显示了在将 `SALARY` 输入列添加至 `totalSalaries` 实例变量之前如何检查其值是否为空值：

```
// if value of SALARY is not null
if (!isNull("SALARY")) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

或者，也可以使用以下 Java 代码获得相同结果：

```
// if value of SALARY is not null
String strColName = "SALARY";
if (!isNull(strColName)) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

# logError

将错误消息写入会话日志中。

请使用以下语法：

```
logError(String msg);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
msg	输入	String	错误消息字符串。

可以将 logError 方法添加到除**导入包**和**函数 Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

以下 Java 代码显示了输入端口为空值时如何记录错误：

```
// check BASE_SALARY
if (isNull("BASE_SALARY")) {
    logError("Cannot process a null salary field.");
}
```

当代码运行时，会话日志中将显示以下消息：

```
[JTX_1013] [ERROR] Cannot process a null salary field.
```

# logInfo

将信息性消息写入会话日志中。

请使用以下语法：

```
logInfo(String msg);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
msg	输入	String	信息消息字符串。

可以将 logInfo 方法添加到除**导入包**和**函数 Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

以下 Java 代码显示了在 Java 转换处理达到消息阈值（1000 行）后如何将消息写入会话日志：

```
if (numRowsProcessed == messageThreshold) {
    logInfo("Processed " + messageThreshold + " rows.");
}
```

# resetNotification

如果数据集成服务计算机在重启模式下运行，则在映射运行后将重置 Java 代码中使用的变量。

在重新启动模式中，数据集成服务无法取消初始化，但可以在请求后对其进行重置以便数据集成服务可以处理下一个请求。

对于 Java 转换，可以在运行映射之后使用 resetNotification 方法来重置 Java 代码中的变量。

请使用以下语法：

```
public int resetNotification(IGroup group) {
    return EStatus.value;
}
```

下表介绍了参数：

参数	参数类型	数据类型	说明
int	输出	EStatus.value	返回值，其中 value 是以下值之一： - SUCCESS. 成功。 - FAILURE. 失败。 - NOIMPL. 未实现。
组	输入	IGroup	输入组。

在**帮助程序**选项卡的代码输入选项卡上，可以将 resetNotification 方法添加到 Java 代码。

resetNotification 方法不会显示在“可调用 API”列表中。

例如，假设 Java 代码声明名为 out5\_static 的静态变量并将其初始化为 1，则下次运行映射后，以下 Java 代码会将 out5\_static 变量重置为 1。

```
public int resetNotification(IGroup group) {
    out5_static=1;
    return EStatus.SUCCESS;
}
```

该方法不是必需的。但是，如果数据集成服务在重新启动模式中运行，并且映射包含未实现 resetNotification 方法的 Java 转换，则日志中将显示 JSDK\_42075 警告消息。

# rollBack

生成回滚事务。

可以在除“导入包”或“Java 表达式”代码输入选项卡以外的任意选项卡中使用回滚。您只能在配置为生成事务的主动转换中使用回滚。如果在未配置为生成事务的主动转换中使用回滚，集成服务将生成错误并导致会话失败。

请使用以下语法：

```
rollBack();
```

使用以下代码生成回滚事务，如果输入行存在非法条件，则会导致会话失败，或者如果处理的行数为 100，则生成事务：

```
// If row is not legal, rollback and fail session.
if (!isRowLegal()) {
```

```

        rollback();
        failSession( "Cannot process illegal row." );
    } else if (rowsProcessed==100) {

        commit();

        rowsProcessed=0;
    }

```

## setNull

将主动或被动 Java 转换中的输出列值设置为空值。

请使用以下语法：

```
setNull(String strColName);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
strColName	输入	String	输出列名称。

setNull 方法可以将主动或被动 Java 转换中的输出列值设置为空值。如果将输出列值设置为空值，则生成输出行后才能修改该值。

可以将 setNull 方法添加到任意代码输入选项卡（除**导入导入包和函数 Java 表达式**选项卡之外）上的 Java 代码。

以下 Java 代码显示了如何检查输入列值并将相应的输出列值设置为空值：

```

// check value of Q3RESULTS input column
if(isNull("Q3RESULTS")) {
    // set the value of output column to null
    setNull("RESULTS");
}

```

或者，可以使用以下 Java 代码获得相同效果：

```

// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}

```

## setOutRowType

设置输出行的更新策略。setOutRowType 方法可以标记要插入、更新或删除的行。

只能在“输入行”代码输入选项卡中使用 setOutRowType。只能在配置为设置更新策略的主动转换中使用 setOutRowType。如果在未配置为设置更新策略的主动转换中使用 setOutRowType，则会话将生成错误并失败。

请使用以下语法：

```
setOutRowType(String rowType);
```

下表介绍了此方法的参数：

参数	数据类型	输入/输出	说明
rowType	字符串	输入	更新策略类型。值可以是 INSERT、UPDATE 或 DELETE。

如果当前行的输入类型是 UPDATE 或 INSERT，并且输入端口 input1 的值小于 100，则使用以下 Java 代码来传播行类型（如果 input1 的值大于 100，则将输出类型设置为 DELETE）：

```
// Set the value of the output port.  
output1 = input1;  
  
// Get and set the row type.  
String rowType = getInRowType();  
setOutRowType(rowType);  
  
// Set row type to DELETE if the output port value is > 100.  
if(input1 > 100)  
    setOutRowType(DELETE);
```

# storeMetadata

存储可以在运行时使用 getMetadata 方法检索的 Java 转换元数据。

请使用以下语法：

```
storeMetadata (String key String data);
```

下表介绍了参数：

参数	参数类型	数据类型	说明
键	输入	String	确定元数据。storeMetadata 方法需要一个用于标识元数据的键。将该键定义为任何字符串。
数据	输入	String	要存储为 Java 转换元数据的数据。

可以将 storeMetadata 方法添加到以下代码输入选项卡上的 Java 代码：

- 帮助程序
- 输入
- 结尾
- 优化器接口
- 函数

通过在主动转换中配置 `storeMetadata` 方法可以接受用于推入优化的筛选条件。`storeMetadata` 方法会存储优化器从映射推送到 Java 转换的筛选条件。

```
// Store a filter condition  
storeMetadata ("FilterKey", condition);
```

## 第 14 章

# Java 表达式

本章包括以下主题：

- [Java 表达式概览, 224](#)
- [使用定义表达式定义函数对话框来定义表达式, 225](#)
- [使用简单接口, 227](#)
- [使用高级接口, 228](#)
- [JExpression 类 API 引用, 232](#)

## Java 表达式概览

可以在 Java 转换中使用 Java 编程语言调用 PowerCenter 表达式。

使用表达式可扩展 Java 转换的功能。例如，可以通过在 Java 转换中调用表达式来查找输入或输出端口的值，或者查找 Java 转换变量的值。

要在 Java 转换中调用表达式，请生成 Java 代码或使用 Java 转换 API 方法来调用表达式。调用表达式后，可在相应的代码输入选项卡上使用该表达式的结果。可以生成调用表达式的 Java 代码，或使用 API 方法编写调用表达式的 Java 代码。

下表介绍了可用于在 Java 转换中创建和调用表达式的方法：

方法	说明
定义表达式定义函数对话框	可用于创建函数以调用表达式，并为表达式生成代码。
简单接口	可用于调用一个 API 方法以调用表达式并获取该表达式的结果。
高级接口	可用于定义表达式、调用表达式和使用表达式的结果。 如果您熟悉面向对象的编程，并且希望对表达式调用操作进行更多控制，请使用高级接口。

## 表达式函数类型

通过使用**表达式编辑器**、在**定义表达式**对话框中编写表达式、使用**定义函数**对话框或使用简单或高级接口，可以为 Java 转换创建表达式。

可以在输入的表达式中使用输入或输出端口变量或 Java 代码中的变量作为输入参数。

如果使用**定义表达式**对话框，则在 Java 转换中使用表达式之前可以使用**表达式编辑器**来验证该表达式。



如果使用**定义函数**对话框，则在 Java 转换中使用表达式之前可以验证该表达式。

可以在 Java 转换中调用以下类型的表达式函数：

表达式函数类型	说明
转换语言函数	类似于 SQL 的函数，用于处理通用表达式。
用户定义的函数	基于转换语言函数在 PowerCenter Developer 工具中创建的函数。
自定义函数	使用自定义函数 API 创建的函数。

还可以在表达式中使用未连接的转换和、内置变量、用户定义的映射和工作流变量以及预定义的工作流变量。例如，可以在表达式中使用未连接的查找转换。

# 使用定义表达式定义函数对话框来定义表达式

定义 Java 表达式时，请配置函数、创建表达式并生成调用该表达式的代码。

可在**定义表达式定义函数**对话框中定义函数和创建表达式。

要创建一个表达式函数并在 Java 转换中使用该表达式，请完成以下高级别任务：

1. 配置调用表达式的函数，包括函数名称、说明和参数。创建表达式时，请使用函数参数。
2. 创建表达式语法并验证该表达式。
3. 生成调用表达式的 Java 代码。

Designer 将该代码放置在 Transformation Developer 的 **Java 表达式**代码输入选项卡上。

Developer 将该代码放置在**函数**代码输入选项卡上。

生成 Java 代码后，基于您使用的是简单接口还是高级接口，在相应的代码输入选项卡上调用生成的函数，以便调用一个表达式或获得一个 JExpression 对象。

**注意:** 要在创建表达式时验证该表达式，必须使用**定义表达式定义函数**对话框。

## 步骤 1。配置函数

为调用表达式的 Java 函数配置函数名称、说明和输入参数。

配置函数时，请使用以下规则和准则：

- 使用不与转换或预留的 Java 关键字中现有 Java 函数冲突的唯一函数名称。
- 必须配置参数名称、Java 数据类型、精度和小数位数。输入参数是调用转换的 Java 代码中的函数时传递的值。
- 要将 Date 数据类型传递给表达式，请对输入参数使用 String 数据类型。

如果表达式返回 Date 数据类型，可以将返回值用作简单接口中的 String 数据类型以及高级接口中的 String 或 long 数据类型。

## 步骤 2。创建并验证表达式

创建表达式时，请使用为函数配置的参数。

您还可以在表达式中使用转换语言函数、自定义函数或其他用户定义函数。您可以在**定义函数**对话框**定义表达式**对话框或**表达式编辑器**对话框中创建并验证表达式。

## 步骤 3。生成表达式的 Java 代码

配置函数和函数参数并定义和验证表达式后，可以生成调用表达式的 Java 代码。

DesignerDeveloper 将生成的 Java 代码置于 **Java 表达式函数** 代码输入选项卡上。使用生成的 Java 代码调用函数，该函数调用 Transformation Developer 的代码输入选项卡中的表达式。您可以生成简单或高级 Java 代码。

生成调用表达式的 Java 代码后，无法编辑表达式或重新对其进行验证。要在代码生成后修改表达式，必须重新创建表达式。

## 使用定义表达式定义函数对话框创建表达式并生成 Java 代码

使用**定义表达式定义函数**对话框可以创建调用表达式的函数。

要创建调用表达式的函数，请完成以下步骤：

1. 在 Transformation Developer 中，打开 Java 转换或创建新的 Java 转换。
2. 在 **Java 代码** 选项卡中，单击**新建函数定义表达式**链接。  
此时将显示**定义表达式定义函数**对话框。
3. 输入函数名称。
4. 或者，输入表达式的说明。  
最多可输入 2,000 个字符。
5. 为函数创建参数参数。  
创建参数参数时，请配置参数参数名称、数据类型、精度和小数位数。
6. 单击**启动编辑器**使用创建的参数创建表达式。在**表达式**选项卡上，使用创建的参数创建表达式。
7. 要验证表达式，请单击**验证**。
8. 或者，在**表达式**框中输入表达式。然后单击**验证**来验证表达式。
9. 要使用高级接口生成 Java 代码，请选择**生成高级代码**选项。然后单击**生成**。

DesignerDeveloper 将在 **Java 表达式函数** 代码输入选项卡中生成调用表达式的函数。

## Java 表达式模板

可以使用表达式的简单或高级 Java 代码为表达式生成 Java 代码。

表达式的 Java 代码是基于表达式的模板生成的。

以下示例显示了为简单 Java 代码生成的 Java 表达式的模板：

```
Object function_name (Java datatype x1[,  
                        Java datatype x2 ...] )  
    throws SDK Exception  
{  
    return (Object)invokeJExpression( String expression,  
                                     new Object [] { x1[, x2, ... ]} );  
}
```

以下示例显示了使用高级接口生成的 Java 表达式的模板：

```
JExpression function_name () throws SDKException
{
    JExprParamMetadata params[] = new JExprParamMetadata[number of parameters];
    params[0] = new JExprParamMetadata (
        EDataType.STRING, // data type
        20, // precision
        0 // scale
    );

    ...
    params[number of parameters - 1] = new JExprParamMetadata (
        EDataType.STRING, // data type
        20, // precision
        0 // scale
    );

    ...
    return defineJExpression(String expression,params);
}
```

# 使用简单接口

使用 invokeJExpression Java API 方法调用简单接口中的表达式。

## invokeJExpression

调用表达式并返回该表达式的值。

请使用以下语法：

```
(datatype)invokeJExpression(
    String expression,
    Object[] paramMetadataArray);
```

invokeJExpression 方法的输入参数是字符串值，该值表示表达式和包含表达式输入参数的对象数组。

下表介绍了参数：

参数	参数类型	数据类型	说明
表达式	输入	String	表示表达式的字符串。
paramMetadataArray	输入	Object[]	包含表达式的输入参数的对象数组。

可以将 invokeJExpression 方法添加到除**导入**和**函数导入包**和 **Java 表达式**选项卡之外的任意代码输入选项卡上的 Java 代码中。

使用 invokeJExpression 方法时，请遵循以下规则和准则：

- 返回数据类型。 invokeJExpression 方法的返回数据类型是一个对象。 您必须使用相应的数据类型转换函数的返回值。  
可以返回数据类型为 Integer、Double、String 和 byte[] 的值。
- 行类型。 invokeJExpression 方法的返回值的行类型为 INSERT。  
要为该返回值使用其他行类型，请使用高级接口。
- 空值。 如果将空值作为参数传递或者 invokeJExpression 方法的返回值为空，则将该值视为空指示符。  
例如，如果表达式的返回值为空且返回数据类型为 String，则返回一个具有空值的字符串。

- Date 数据类型。必须将数据类型为 Date 的输入参数转换为 String 数据类型。  
要将表达式中的字符串用作 Date 数据类型，请使用 to\_date() 函数将字符串转换为 Date 数据类型。  
或者，您必须将返回 Date 数据类型的所有表达式的返回类型转换为 String 数据类型。

**注意：**必须对连续传递给表达式的参数进行编号，这些参数必须以字母 x 开头。例如，要将三个参数传递给表达式，请将参数分别命名为 x1、x2 和 x3。

## 简单接口示例

您可以定义并调用表达式，这些表达式使用[帮助程序代码帮助程序](#)和[输入行输入代码](#)输入选项卡上的 invokeJExpression API 方法。

以下示例显示如何在 Java 转换的 NAME 和 ADDRESS 输入端口完成查找，并将返回值分配给 COMPANY\_NAME 输出端口。

在[输入行输入代码](#)输入选项卡上输入以下代码：

```
COMPANY_NAME = (String)invokeJExpression(":lkp.my_lookup(X1,X2)", new Object [] {str1 ,str2} );
generateRow();
```

## 使用高级接口

在高级接口中，可以使用面向对象的 API 方法来定义、调用并获得表达式的结果。

下表介绍了高级接口中可用的类和 API 方法：

类或 API 方法	说明
EDatatype 类	枚举表达式的数据类型。
JExprParamMetadata 类	包含一个表达式中每个参数的元数据。 参数元数据包含数据类型、精度和小数位。
defineJExpression API 方法	定义表达式。 包含 PowerCenter 表达式字符串和参数。
invokeJExpression API 方法	调用表达式。
JExpression 类	包含用于创建、调用、获得元数据、获得表达式结果以及检查返回数据类型的方法。

## 通过高级接口调用表达式

可以使用高级接口来定义和调用表达式以及获取表达式的结果。

1. 在[帮助程序代码](#)或[输入行帮助程序](#)或[输入代码](#)输入选项卡上，为表达式的每个参数参数创建一个 JExprParamMetadata 类实例并设置元数据的值。 或者，可以在 defineJExpression 方法中实例化 JExprParamMetadata 对象。
2. 使用 defineJExpression 方法获取表达式的 JExpression 对象。
3. 在相应的代码输入选项卡上，使用 invokeJExpression 方法调用表达式。
4. 使用 isResultNull 方法检查返回值的结果。

5. 可以使用 `getResultDataType` 和 `getResultMetadata` 方法获取返回值的数据类型或返回值的元数据。
6. 使用相应的 API 方法获取表达式的结果。可以使用 `getInt`、`getDouble`、`getStringBuffer` 和 `getBytes` 方法。

## 使用高级接口的规则和准则

使用高级接口时，您必须了解规则和准则。

使用以下规则和准则：

- 如果您将空值作为参数传递或表达式结果为空值，该值将被视为空指示器。例如，如果表达式的结果为空值且返回的数据类型为 `String`，则返回的字符串为空值。您可以使用 `isResultNull` 方法检查表达式的结果。
- 您必须先将 `Date` 数据类型的输入参数转换为 `String`，才可以将其用于表达式。要将表达式中的字符串用作 `Date` 数据类型，请使用 `to_date()` 函数将字符串转换为 `Date` 数据类型。

您可以获取表达式的结果，该表达式返回的 `Date` 数据类型为 `String` 或 `long` 数据类型。

使用 `getStringBuffer` 方法可以获取表达式的结果，该表达式返回的 `Date` 数据类型为 `String` 数据类型。使用 `getLong` 方法可以获取表达式的结果，该表达式返回的 `Date` 数据类型为 `long` 数据类型。

## EDatatype 类

枚举表达式中所使用的 Java 数据类型。获取表达式的返回数据类型，或分配 `JExprParamMetadata` 对象中参数的数据类型。无需实例化 `EDatatype` 类。

下表列出了表达式中 Java 数据类型的枚举值：

数据类型	枚举值
INT	1
DOUBLE	2
STRING	3
BYTE_ARRAY	4
DATE_AS_LONG	5

以下 Java 代码示例显示如何使用 `EDatatype` 类向 `JExprParamMetadata` 对象分配 `String` 数据类型：

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata (
    EDatatype.STRING, // data type
    20, // precision
    0 // scale
);
...
```

## JExprParamMetadata 类

实例化代表表达式参数的对象，并为这些参数设置元数据。

可以使用 `JExprParamMetadata` 对象数组作为 `defineJExpression` 方法的输入来为输入参数设置元数据。可以在 **Java 表达式函数** 代码输入选项卡上或在 `defineJExpression` 中创建 `JExprParamMetadata` 对象的实例。

请使用以下语法：

```
JExprParamMetadata paramMetadataArray[] = new JExprParamMetadata[numberOfParameters];
paramMetadataArray[0] = new JExprParamMetadata(datatype, precision, scale);
...
paramMetadataArray[numberOfParameters - 1] = new JExprParamMetadata(datatype, precision, scale);;
```

下表介绍了以下参数：

参数	参数类型	参数数据类型	说明
数据类型	输入	EDatatype	参数的数据类型。
精度	输入	整数	参数的精度。
小数位数	输入	整数	参数的小数位数。

例如，通过以下 Java 代码可使用数据类型 String、精度 20、小数位数 0 实例化由两个 JExprParamMetadata 对象组成的数组：

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
return defineJExpression("LKP.LKP_addresslookup(X1,X2)",params);
```

defineJExpression

定义表达式，包括表达式字符串和输入参数。defineJExpression 方法的参数包括：一个 JExprParamMetadata 对象数组（包含输入参数）以及一个用于定义表达式语法的字符串值。

请使用以下语法：

```
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

下表介绍了参数：

参数	类型	数据类型	说明
表达式	输入	String	表示表达式的字符串。
paramMetadataArray	输入	Object[]	JExprParamMetadata 对象数组，其中包含表达式的输入参数。

可以将 defineJExpression 方法添加到除导入和函数选项卡之外的任何代码输入选项卡上的 Java 代码中。

要使用 defineJExpression 方法，必须实例化 JExprParamMetadata 对象数组，该数组代表表达式的输入参数。可以为这些参数设置元数据值，然后将该数组作为参数传递给 defineJExpression 方法。

例如，以下 Java 代码可创建一个表达式，用于查找两个字符串的值：

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
defineJExpression("lkp.mylookup(x1,x2)",params);
```

**注意：**必须对连续传递给表达式的参数进行编号，这些参数必须以字母 x 开头。例如，要将三个参数传递给表达式，请将这些参数分别命名为 x1、x2 和 x3。

# JExpression 类

包含多个方法，可用于创建和调用表达式、返回表达式的值以及检查返回的数据类型。

下表列出了 JExpression 类包含的方法：

方法名称	说明
invoke	调用表达式。
getResultDataType	返回表达式结果的数据类型。
getResultMetadata	返回表达式结果的元数据。
isResultNull	检查表达式结果的结果值。
getInt	以 Integer 数据类型返回表达式结果的值。
getDouble	以 Double 数据类型返回表达式结果的值。
getStringBuffer	以 String 数据类型返回表达式结果的值。
getBytes	以 byte[] 数据类型返回表达式结果的值。

## 相关主题：

- “JExpression 类 API 引用” 页面上 232

# 高级接口示例

可以使用高级接口在 Java 转换中创建并调用查找表达式。

以下 Java 代码示例说明了如何创建一个函数来调用表达式以及如何调用该表达式以获取返回值。本示例将具有 String 数据类型的两个输入端口（NAME 和 COMPANY）的值传递到函数 myLookup。myLookup 函数使用查找表达式来查找 ADDRESS 输出端口的值。

**注意：**本示例假定在名为 LKP\_addresslookup 的映射中有一个未连接的查找转换。

在 Transformation Developer 的**帮助程序代码帮助程序**选项卡上使用以下 Java 代码：

```
JExpression addressLookup() throws SDKException
{
    JExprParamMetadata params[] = new JExprParamMetadata[2];
    params[0] = new JExprParamMetadata (
        EDataType.STRING,    // data type
        50,                  // precision
        0,                   // scale
    );
    params[1] = new JExprParamMetadata (
        EDataType.STRING,    // data type
        50,                  // precision
        0,                   // scale
    );
    return defineJExpression(":LKP.LKP_addresslookup(X1,X2)",params);
}
JExpression lookup = null;
boolean isJExprObjCreated = false;
```

在**输入行输入**选项卡上使用以下 Java 代码调用表达式并返回 ADDRESS 端口的值：

```
if(!isJExprObjCreated)
```

```

{
    lookup = addressLookup();
    isJExprObjCreated = true;
}
lookup = addressLookup();
lookup.invoke(new Object [] {NAME,COMPANY}, ERowType.INSERT);
EDatatype addressDataType = lookup.getResultDataType();
if(addressDataType == EDatatype.STRING)
{
    ADDRESS = (lookup.getStringBuffer()).toString();
} else {
    logError("Expression result datatype is incorrect.");
}
...

```

## JExpression 类 API 引用

JExpression 类包含多个 API 方法，可用于创建和调用表达式、返回表达式的值以及检查返回值的数据类型。

JExpression 类包含以下 API 方法：

- getBytes
- getDouble
- getInt
- getLong
- getResultDataType
- getResultMetadata
- getStringBuffer
- invoke
- isResultNull

### getBytes

以 byte[] 数据类型返回表达式结果的值。获取一个表达式的结果，该表达式可使用 AES\_ENCRYPT 函数对数据进行加密。

请使用以下语法：

```
objectName.getBytes();
```

使用下面的示例 Java 代码可获取以下表达式的结果：该表达式使用 AES\_ENCRYPT 函数对二进制数据进行加密（其中，JExprEncryptData 是一个 JExpression 对象）：

```
byte[] newBytes = JExprEncryptData.getBytes();
```

### getDouble

以 Double 数据类型返回表达式结果的值。

请使用以下语法：

```
objectName.getDouble();
```



使用下面的示例 Java 代码可获取以下表达式的结果：该表达式以 Double 数据类型返回薪资值（其中，JExprSalary 是一个 JExpression 对象）：

```
double salary = JExprSalary.getDouble();
```

## getInt

以 Integer 数据类型返回表达式结果的值。

请使用以下语法：

```
objectName.getInt();
```

例如，使用下面的 Java 代码可获取以下表达式的结果：该表达式以 Integer 数据类型返回员工 ID 编号（其中，findEmpID 是一个 JExpression 对象）：

```
int empID = findEmpID.getInt();
```

## getLong

以 Long 数据类型返回表达式结果的值。获取使用 Date 数据类型的表达式的结果。

请使用以下语法：

```
objectName.getLong();
```

使用下面的示例 Java 代码可获取以下表达式的结果：该表达式以 Long 数据类型返回日期值（其中，JExprCurrentDate 是一个 JExpression 对象）：

```
long currDate = JExprCurrentDate.getLong();
```

## getResultDataType

返回表达式结果的数据类型。返回 EDataType 的值。

请使用以下语法：

```
objectName.getResultDataType();
```

使用下面的示例 Java 代码可调用表达式并将结果的数据类型赋值给变量 dataType：

```
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);  
EDataType dataType = myObject.getResultDataType();
```

## getResultMetadata

返回表达式结果的元数据。可以使用 getResultMetadata 获取表达式结果的精度、小数位数和数据类型。可以将表达式返回值的元数据赋值给 JExprParamMetadata 对象。使用 getScale、getPrecision 和 getDataType 对象方法可检索结果元数据。

请使用以下语法：

```
objectName.getResultMetadata();
```

使用下面的示例 Java 代码可将 myObject 返回值的小数位数、精度和数据类型赋值给变量：

```
JExprParamMetadata myMetadata = myObject.getResultMetadata();  
int scale = myMetadata.getScale();  
int prec = myMetadata.getPrecision();  
int datatype = myMetadata.getDataType();
```

**注意：**getDataType 对象方法将按 EDataType 中的枚举内容返回数据类型的整数值。

# getStringBuffer

以 String 数据类型返回表达式结果的值。

请使用以下语法：

```
objectName.getStringBuffer();
```

使用下面的示例 Java 代码可获取以下表达式的结果：该表达式将返回两个连接的字符串（其中，JExprConcat 是一个 JExpression 对象）：

```
String result = JExprConcat.getStringBuffer();
```

# invoke

调用表达式。invoke 的参数中包括用于定义输入参数和行类型的对象。必须先实例化 JExpression 对象，才可以使用 invoke 方法。对于行类型，请使用 ERowType.INSERT、ERowType.DELETE 和 ERowType.UPDATE。

请使用以下语法：

```
objectName.invoke(  
    new Object[] { param1[, ... paramN ]},  
    rowType  
);
```

下表介绍了以下参数：

参数	数据类型	输入/ 输出	说明
objectName	JExpression	输入	JExpression 对象名称。
参数	-	输入	包含表达式输入值的对象数组。

例如，在 **Java 表达式函数** 代码输入选项卡上创建了名为 address\_lookup() 的函数，该函数返回表示该表达式的 JExpression 对象。使用下面的代码可调用使用输入端口 NAME 和 COMPANY 的表达式：

```
JExpression myObject = address_lookup();  
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);
```

# isResultNull

检查表达式结果值。

请使用以下语法：

```
objectName.isResultNull();
```

使用下面的示例 Java 代码可调用一个表达式，并且在表达式的返回值不为空的情况下将返回值赋值给变量 address：

```
JExpression myObject = address_lookup();  
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);  
if(!myObject.isResultNull()) {  
    String address = myObject.getStringBuffer();  
}
```

## 第 15 章

# Java 转换示例

本章包括以下主题：

- [Java 转换示例概览, 235](#)
- [步骤 1。导入映射, 236](#)
- [步骤 2。创建转换和配置端口, 236](#)
- [步骤 3。输入 Java 代码, 237](#)
- [“导入包”选项卡, 237](#)
- [“帮助程序代码”选项卡, 237](#)
- [“在输入行”选项卡, 238](#)
- [步骤 4。编译 Java 代码, 239](#)
- [步骤 5。创建会话和工作流, 240](#)

## Java 转换示例概览

您可以在此示例中使用 Java 代码来创建和编译主动 Java 转换。导入示例映射并创建和编译 Java 转换。然后，可以创建和运行包含映射的会话和工作流。

Java 转换会处理某个虚构公司的员工数据。它会从平面文件源读取输入行，并将输出行写入平面文件目标。源文件包含员工数据，包括员工身份证号、姓名、职位和经理身份证号。

该转换会根据经理身份验证号查找指定员工的经理姓名，并生成包含员工数据的输出行。输出数据包括员工身份证号、姓名、位置和员工经理的姓名。如果员工的源数据中没有经理，则转换假定员工处于公司组织结构图层次结构的顶端。

**注意：**转换逻辑假定员工职位在源文件中按照降序排列。

完成以下步骤以导入示例映射，创建和编译 Java 转换，以及创建包含该映射的会话和工作流：

1. 导入示例映射。
2. 创建 Java 转换并配置 Java 转换端口。
3. 在适当的代码输入选项卡中为转换输入 Java 代码。
4. 编译 Java 代码。
5. 创建并运行会话和工作流。

PowerCenter 客户端安装包可在此示例中使用的映射 m\_jtx\_hier\_useCase.xml 和平面文件源 hier\_data。

相关主题：

- “Java 转换” 页面上 197

步骤 1。导入映射

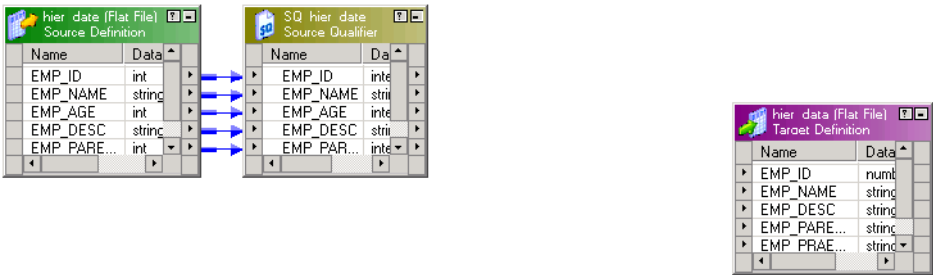
在 Designer 中为示例映射导入元数据。该示例映射包含以下组件：

- 源定义和源限定符转换。**平面文件源定义 hier\_input，它定义了转换的源数据。
- 目标定义。**平面文件目标定义 hier\_data，它接收来自转换的输出数据。

可以从以下位置为该映射导入源数据：

```
<PowerCenter Client installation directory>\client\bin\m_jtx_hier_useCase.xml
```

下图显示了该示例映射：



步骤 2。创建转换和配置端口

在 Mapping Designer 中创建 Java 转换并配置端口。可以使用输入和输出端口名称作为 Java 代码中的变量。在 Java 转换中，在输入或输出组中创建输入和输出端口。Java 转换只能包含一个输入组和一个输出组。

在 Mapping Designer 中，创建活动 Java 转换并配置端口。在本示例中，转换命名为 jtx\_hier\_useCase。

**注意：**要在本示例中使用 Java 代码，必须使用输入和输出端口的确切名称。

下表显示了转换的输入和输出端口：

端口名称	端口类型	数据类型	精度	小数位数
EMP_ID_INP	输入	整型	10	0
EMP_NAME_INP	输入	字符串	100	0
EMP_AGE	输入	整型	10	0
EMP_DESC_INP	输入	字符串	100	0
EMP_PARENT_EMPID	输入	整型	10	0
EMP_ID_OUT	输出	整型	10	0

端口名称	端口类型	数据类型	精度	小数位数
EMP_NAME_OUT	输出	字符串	100	0
EMP_DESC_OUT	输出	字符串	100	0
EMP_PARENT_EMPNAME	输出	字符串	100	0

## 步骤 3。输入 Java 代码

在以下代码输入选项卡中输入转换的 Java 代码：

- **导入包。**导入 java.util.Map 和 java.util.HashMap 包。
- **帮助程序代码。**包含用于在 Java 转换中跟踪数据状态的映射对象、锁定对象和布尔变量。
- **输入行时。**定义 Java 转换在接收输入行时的行为。

### “导入包”选项卡

在“导入包”选项卡中导入第三方 Java 包、内置 Java 包或自定义 Java 包。该示例转换使用 Map 和 HashMap 包。

在“导入包”选项卡上输入以下代码：

```
import java.util.Map;
import java.util.HashMap;
```

Designer 会将导入语句添加到转换的 Java 代码。

**相关主题：**

- [“配置 Java 转换设置”页面上 208](#)

### “帮助程序代码”选项卡

在“帮助程序代码”选项卡上为 Java 转换声明用户定义的变量和方法。“帮助程序代码”选项卡会定义以下变量供“在输入行”选项卡中的 Java 代码使用：

- **empMap**.源中存储身份证号和员工姓名的 Map 对象。
- **lock**.Lock 对象用于在各分区中同步对 empMap 的访问。
- **generateRow**.布尔型变量用于确定是否应为当前输入行生成输出行。
- **isRoot**.布尔型变量用于确定员工是否位于公司组织结构的顶端（根部）。

在“帮助程序代码”选项卡中输入以下代码：

```
// Static Map object to store the ID and name relationship of an
```

```

// employee. If a session uses multiple partitions, empMap is shared
// across all partitions.
private static Map <Integer, String> empMap = new HashMap <Integer, String> ();
// Static lock object to synchronize the access to empMap across
// partitions.
private static Object lock = new Object();
// Boolean to track whether to generate an output row based on validity
// of the input data.
private boolean generateRow;
// Boolean to track whether the employee is root.
private boolean isRoot;

```

## “在输入行” 选项卡

转换收到输入行时，Java 转换会执行“在输入行”选项卡上的 Java 代码。在此示例中，转换可能会也可能不会生成输出行，具体取决于输入行的值。

在“在输入行”选项卡中输入以下代码：

```

// Initially set generateRow to true for each input row.
generateRow = true;

// Initially set isRoot to false for each input row.
isRoot = false;

// Check if input employee id and name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);
    // If input employee id and/or name is null, don't generate a output
    // row for this input row.
    generateRow = false;
} else {
    // Set the output port values.
    EMP_ID_OUT = EMP_ID_INP;
    EMP_NAME_OUT = EMP_NAME_INP;
}

if (isNull ("EMP_DESC_INP"))
{

```

```

        setNull("EMP_DESC_OUT");
    } else {
        EMP_DESC_OUT = EMP_DESC_INP;
    }
    boolean isParentEmpIdNull = isNull("EMP_PARENT_EMPID");

    if(isParentEmpIdNull)
    {
        // This employee is the root for the hierarchy.
        isRoot = true;
        logInfo("This is the root for this hierarchy.");
        setNull("EMP_PARENT_EMPNAME");
    }

    synchronized(lock)
    {
        // If the employee is not the root for this hierarchy, get the
        // corresponding parent id.
        if(!isParentEmpIdNull)
            EMP_PARENT_EMPNAME = (String) (empMap.get(new Integer (EMP_PARENT_EMPID)));
        // Add employee to the map for future reference.
        empMap.put (new Integer(EMP_ID_INP), EMP_NAME_INP);
    }

    // Generate row if generateRow is true.
    if(generateRow)
        generateRow();

```

## 步骤 4。编译 Java 代码

在 Transformation Developer 中单击“编译”，为转换编译 Java 代码。“输出”窗口会显示编译的状态。如果 Java 代码编译失败，请在代码输入选项卡中更正这些错误并重新编译 Java 代码。成功编译转换后，将转换保存到存储库。

相关主题：

- [“编译 Java 转换” 页面上 210](#)
- [“修复编译错误” 页面上 210](#)

## 步骤 5。创建会话和工作流

使用 m\_jtx\_hier\_useCase 映射为 Workflow Manager 中的映射创建会话和工作流。

配置会话时，可以使用以下位置中的示例源文件：

```
<PowerCenter Client installation directory>\client\bin\hier_data
```

### 示例数据

以下数据摘自示例源文件：

```
1,James Davis,50,CEO,
4,Elaine Masters,40,Vice President - Sales,1
5,Naresh Thiagarajan,40,Vice President - HR,1
6,Jeanne Williams,40,Vice President - Software,1
9,Geetha Manjunath,34,Senior HR Manager,5
10,Dan Thomas,32,Senior Software Manager,6
14,Shankar Rahul,34,Senior Software Manager,6
20,Juan Cardenas,32,Technical Lead,10
21,Pramodh Rahman,36,Lead Engineer,14
22,Sandra Patterson,24,Software Engineer,10
23,Tom Kelly,32,Lead Engineer,10
35,Betty Johnson,27,Lead Engineer,14
50,Dave Chu,26,Software Engineer,23
70,Srihari Giran,23,Software Engineer,35
71,Frank Smalls,24,Software Engineer,35
```

以下数据摘自示例目标文件：

```
1,James Davis,CEO,
4,Elaine Masters,Vice President - Sales,James Davis
5,Naresh Thiagarajan,Vice President - HR,James Davis
6,Jeanne Williams,Vice President - Software,James Davis
9,Geetha Manjunath,Senior HR Manager,Naresh Thiagarajan
10,Dan Thomas,Senior Software Manager,Jeanne Williams
14,Shankar Rahul,Senior Software Manager,Jeanne Williams
20,Juan Cardenas,Technical Lead,Dan Thomas
21,Pramodh Rahman,Lead Engineer,Shankar Rahul
22,Sandra Patterson,Software Engineer,Dan Thomas
23,Tom Kelly,Lead Engineer,Dan Thomas
35,Betty Johnson,Lead Engineer,Shankar Rahul
50,Dave Chu,Software Engineer,Tom Kelly
```



70,Srihari Giran,Software Engineer,Betty Johnson

71,Frank Smalls,Software Engineer,Betty Johnson

## 第 16 章

# 连接器转换

本章包括以下主题：

- [连接器转换概览, 242](#)
- [连接器转换属性, 243](#)
- [定义联接条件, 244](#)
- [定义联接类型, 244](#)
- [使用已排序输入, 247](#)
- [联接单个源中的数据, 249](#)
- [阻止源管道, 250](#)
- [使用事务, 251](#)
- [创建连接器转换, 252](#)
- [连接器转换提示, 253](#)

## 连接器转换概览

使用连接器转换联接驻留在不同位置或文件系统中的两个相关异构源中的源数据。还可以从同一源联接数据。联接器转换会将源与至少一个匹配列联接。联接器转换将使用与两个源之间的一个或多个列对相匹配的条件。联接器转换是主动转换。

两个输入管道中包括一个主管道和一个详细管道或一个主管道和一个详细分支。主管道结束于联接器转换，但详细管道将继续连接到目标。

要联接映射中两个以上的源，请将联接器转换中的输出与其他源管道联接。将联接器转换添加到映射中，直到您已联接所有源管道。

联接器转换接受来自大多数转换的输入。但是，请考虑您连接到联接器转换的管道上存在的以下限制：

- 如果任一输入管道包含更新策略转换，则不能使用联接器转换。
- 如果在联接器转换之前直接连接序列生成器转换，则不能使用联接器转换。

## 使用联接器转换

当使用联接器转换时，必须配置转换属性、联接类型和联接条件。可以对已排序输入配置联接器转换，以提高集成服务的性能。还可以配置转换范围来控制集成服务如何应用转换逻辑。要使用联接器转换，请完成以下任务：

- **配置联接器转换属性。**联接器转换的属性可确定缓存目录的位置、集成服务处理转换的方式以及集成服务处理缓存的方式。

- **配置联接条件。**联接条件包含两个输入源中的端口，集成服务匹配这两个输入源来联接两个行。根据选定联接的类型，集成服务将行添加到结果集中，或者放弃该行。
- **配置联接类型。**联接是一个关系运算符，用于将不同数据库或平面文件内多个表中的数据合并为单一结果集。可以将联接器转换配置为使用“普通”、“主外联接”、“详细外联接”或“完全外联接”联接类型。
- **配置已排序或未排序输入的会话。**通过将联接器转换配置为使用已排序输入，可以提高会话性能。要将映射配置为使用已排序数据，需要在映射中建立并维护排序顺序，以便集成服务在处理联接器转换时可以使用已排序数据。
- **配置事务范围。**当集成服务处理联接器转换时，可以将转换逻辑一次性应用到事务中的所有数据、所有传入数据或一行数据。

如果在 PowerCenter 中具有分区选项，则可以增加管道中的分区数以提高会话性能。

# 联接器转换属性

联接器转换的属性可确定缓存目录的位置、集成服务处理转换的方式以及集成服务处理缓存的方式。这些属性还确定集成服务如何联接表和文件。

在创建映射时，为每个联接器转换指定属性。在创建会话时，可以替代一些属性，例如每个转换的索引和数据缓存大小。

下表介绍了联接器转换属性：

选项	说明
缓存目录	指定用于缓存主行或详细信息行以及这些行的索引的目录。默认情况下，将在进程变量 \$PMCacheDir 指定的目录中创建缓存文件。如果您改写该目录，请确保目录存在并包含足够的磁盘空间用于存储缓存文件。目录可以是映射的驱动器，也可以是装载的驱动器。
联接类型	指定联接的类型：“普通联接”、“主外联接”、“详细外联接”或“完全外联接”。
主空值排序	不适用于此转换类型。
空值排序详情	不适用于此转换类型。
跟踪级别	此转换的会话日志中显示的详细信息量。选项包括“简洁”、“普通”、“详细数据”和“详细初始化”。
联接器数据缓存大小	转换的数据缓存大小。默认缓存大小为 2,000,000 字节。如果配置的总缓存大小为 2 GB 或更多，则必须在 64 位的集成服务中运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。
联接器索引缓存大小	转换的索引缓存大小。默认缓存大小为 1,000,000 字节。如果配置的总缓存大小为 2 GB 或更多，则必须在 64 位的集成服务中运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。
已排序输入	指定已排序数据。选择“已排序输入”联接已排序数据。使用已排序输入可以提高性能。

选项	说明
主排序顺序	指定主源数据的排序顺序。如果主源数据按升序排列，则选择“升序”。如果选择“升序”，也会启用已排序输入。默认为“自动”。
转换范围	指定集成服务如何将转换逻辑应用至传入的数据。可以选择“事务”、“所有输入”或“行”。

# 定义联接条件

联接条件包含两个输入源中的端口，集成服务匹配这两个输入源来联接两个行。根据选定联接的类型，集成服务将行添加到结果集中，或者放弃该行。联接器转换将生成基于联接类型、条件和输入数据源的结果集。

定义联接条件之前，请验证是否已配置主源和详细源以实现最佳性能。会话期间，集成服务会将主源的每个行与详细信息源进行比较。要提高未排序联接器转换的性能，请将行数较少的源作为主源。要提高已排序联接器转换的性能，请将重复键值数较少的源作为主源。

默认情况下，向联接器转换添加端口时，第一个源管道中的端口会显示为详细信息源。从第二个源管道添加的端口会将它们设置为主源。要更改这些设置，请针对要设置为主源的端口单击“端口”选项卡上的 M 列。这会将此源中的端口设置为主端口，并将其他源中的端口设置为详细信息源。

根据指定的主源与详细信息源之间的等式定义一个或多个条件。例如，如果带有名为 EMPLOYEE\_AGE 和 EMPLOYEE\_POSITION 的表的两个源均包含员工身份证号码，则以下条件将匹配员工在两个源中均列出的行：

EMP\_ID1 = EMP\_ID2

在联接条件中使用来自联接器转换的输入源的一个或多个端口。使用其他端口将增加联接两个源所需的时间。条件中端口的顺序会影响联接器转换的性能。如果在联接条件中使用多个端口，则集成服务将按照您指定的顺序来比较端口。

Designer 会验证条件中的数据类型。条件中的两个端口必须具有相同的数据类型。如果需要在条件中使用数据类型不匹配的两个端口，请转换相应数据类型，以使数据类型相匹配。

如果联接 Char 和 Varchar 数据类型，则集成服务会将填充 Char 值的所有空格计入字符串：

Char(40) = "abcd"  
Varchar(40) = "abcd"

Char 值是使用 36 个空格填充的“abcd”，集成服务不会联接这两个字段，因为 Char 字段的结尾包含空格。

**注意：**联接器转换不会匹配空值。例如，如果 EMP\_ID1 和 EMP\_ID2 都包含具有空值的行，则集成服务不会将其视作匹配项，并且不会联接这两个行。要联接具有空值的行，请将空输入替换为默认值，然后联接默认值。

# 定义联接类型

在 SQL 中，联接是将多个源表中的数据合并为一个结果集的关系运算符。联接器转换类似于 SQL 联接，不同之处在于数据可以来自不同类型的源。

在“属性”选项卡上定义转换中的联接类型。联接器转换支持以下类型的联接：

- 普通联接
- 主外联接

- 详细外联接
- 完全外联接

**注意:** 普通或主外部联接的执行速度比完整外部联接或详细外部联接的执行速度快。

如果结果集中包括不包含任一源中的数据的数据的字段，则联接器转换将使用空值填充空字段。如果知道有字段返回空值但不希望在目标中插入空值，可以在“端口”选项卡上为相应端口设置默认值。

## 普通联接

通过普通联接，集成服务将根据条件丢弃主源和详细源中所有不匹配的数据行。

例如，您可能具有两个用于自动部分的数据源，称为 PARTS\_SIZE 和 PARTS\_COLOR，它们具有以下数据：

**PARTS\_SIZE (master source)**

PART_ID1	DESCRIPTION	SIZE
1	Seat Cover	Large
2	Ash Tray	Small
3	Floor Mat	Medium

**PARTS\_COLOR (detail source)**

PART_ID2	DESCRIPTION	COLOR
1	Seat Cover	Blue
3	Floor Mat	Black
4	Fuzzy Dice	Yellow

要通过匹配两个源中的 PART\_ID 联接两个表，请按如下所示设置条件：

PART\_ID1 = PART\_ID2

通过普通联接来联接这些表时，结果集中将包括以下数据：

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black

以下示例显示了等价 SQL 语句：

```
SELECT * FROM PARTS_SIZE, PARTS_COLOR WHERE PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2
```

## 主外部联接

主外部联接将保留详细源的所有数据行和主源的匹配行。它将丢弃主源中不匹配的行。

通过主外部联接和相同条件来联接示例表时，结果集将包括以下数据：

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

因为没有为 Fuzzy Dice 指定大小，所以集成服务将使用空值填充该字段。

以下示例显示了等价 SQL 语句：

```
SELECT * FROM PARTS_SIZE RIGHT OUTER JOIN PARTS_COLOR ON (PARTS_COLOR.PART_ID2 = PARTS_SIZE.PART_ID1)
```

## 详细外部联接

详细外部联接将保留主源的所有数据行和详细源的匹配行。它将丢弃详细源中不匹配的行。

通过详细外部联接和相同条件来联接示例表时，结果集将包括以下数据：

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

因为没有为 Ash Tray 指定颜色，所以集成服务将使用空值填充该字段。

以下示例显示了等价 SQL 语句：

```
SELECT * FROM PARTS_SIZE LEFT OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

## 完整外部联接

完整外部联接将保留主源和详细源中的所有数据行。

通过完整外部联接和相同条件来联接示例表时，结果集将包括：

PART_ID	DESCRIPTION	SIZE	Color
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

因为没有为 Ash Tray 指定颜色并且没有为 Fuzzy Dice 指定大小，所以集成服务将使用空值填充这些字段。

以下示例显示了等价 SQL 语句：

```
SELECT * FROM PARTS_SIZE FULL OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

# 使用已排序输入

通过将连接器转换配置为使用已排序输入，可以提高会话性能。将连接器转换配置为使用已排序数据时，集成服务会通过将磁盘输入和输出降至最低来提高性能。处理大型数据集时，将会获得最显著的性能提高。

要配置映射以使用已排序数据，需要在映射中建立并维护排序顺序，以便集成服务在处理连接器转换时可以使用已排序数据。完成以下任务以配置映射：

- **配置排序顺序。**配置要联接的数据的排序顺序。可以联接已排序平面文件，也可以使用源限定符转换对关系数据排序。还可以使用排序器转换。
- **添加转换。**使用维护已排序数据顺序的转换。
- **配置连接器转换。**配置连接器转换以使用已排序数据，并配置联接条件以使用排序来源端口。排序来源表示已排序数据的源。

在配置会话中的排序顺序时，可以选择与集成服务代码页相关联的排序顺序。在 Unicode 模式下运行集成服务时，它将使用选定的会话排序顺序对字符数据排序。在 ASCII 模式下运行集成服务时，它将使用二进制排序顺序对所有字符数据排序。要确保按照集成服务的要求对数据排序，数据库排序顺序必须与用户定义的会话排序顺序相同。

在联接来自已分区管道的已排序数据时，必须配置分区，以维护已排序数据的顺序。

## 配置排序顺序

您必须配置排序顺序以确保集成服务将已排序数据传递给连接器转换。

使用以下方法之一配置排序顺序：

- **使用排序的平面文件。**平面文件包含排序数据时，请验证排序列的顺序在每个源文件中是否匹配。
- **使用排序的关系数据。**在源限定符转换中使用排序端口可对源数据库中的列进行排序。将已排序端口的顺序配置为在每个源限定符转换中都相同。
- **使用排序器转换。**使用排序器转换对关系数据或平面文件数据进行排序。将排序器转换置于主管道和详细信息管道中。将每个排序器转换配置为使用相同顺序的排序键端口和排序顺序方向。

如果向配置为使用已排序数据的连接器转换传递未排序或排序不正确的数据，则会话将失败并且集成服务将在会话日志文件中记录错误。

## 将转换添加到映射

在排序源与连接器转换之间添加转换时，请遵循以下准则维护已排序数据：

- 请勿在排序源与连接器转换之间放置以下任何转换：
  - 自定义
  - 未排序汇总器
  - 规范器
  - 等级
  - 联合转换
  - XML 解析器转换
  - XML 生成器转换
  - Maplet（如果包含上述转换之一）
- 如果遵循以下准则，则可以在排序源与连接器转换之间放置已排序汇总器转换：
  - 为已排序输入配置汇总器转换。

- 在汇总器转换中使用的分组依据列的端口与排序源上的端口相同。
- 分组依据端口的顺序必须与排序源上的端口顺序相同。
- 将联接器转换的结果集与其他管道联接在一起时，请验证第一个联接器转换中的数据输出是否已排序。  
**提示:** 可以在排序源后直接放置联接器转换以维护已排序数据。

## 配置联接器转换

要配置联接器转换，请完成以下任务：

- 在“属性”选项卡上启用“已排序输入”。
- 定义联接条件以接收与排序来源顺序相同的排序数据。

## 定义联接条件

配置联接条件以保持与排序来源处建立的排序顺序：排序的平面文件、源限定符转换或排序器转换。如果在排序来源和联接器转换之间使用了已排序的汇总器转换，请在定义联接条件时将已排序汇总器转换视作排序来源。请在定义联接条件时遵循以下准则：

- 在联接条件中使用的端口必须与排序来源中的端口匹配。
- 配置多个联接条件时，第一个联接条件中的端口必须与排序来源中的前几个端口匹配。
- 配置多个条件时，条件的顺序必须与排序来源中的端口顺序匹配，并且不得跳过任何端口。
- 排序来源中已排序端口的编号可以大于或等于联接条件中端口的编号。

## 联接条件示例

例如，可以使用以下已排序端口，在主管道和详细信息管道中配置排序器转换：

1. ITEM\_NO
2. ITEM\_NAME
3. PRICE

配置联接条件时，请遵循以下准则维护排序顺序：

- 必须在第一个联接条件中使用 ITEM\_NO。
- 如果要添加第二个联接条件，则必须使用 ITEM\_NAME。
- 如果要在联接条件中使用 PRICE，则必须同时在第二个联接条件中使用 ITEM\_NAME。

如果跳过 ITEM\_NAME 并联接 ITEM\_NO 和 PRICE，则会失去排序顺序，并且集成服务会话将失败。

使用联接器转换联接主管道和详细信息管道时，可以配置以下任一联接条件：

```
ITEM_NO = ITEM_NO
```

或

```
ITEM_NO = ITEM_NO1
ITEM_NAME = ITEM_NAME1
```

或

```
ITEM_NO = ITEM_NO1
ITEM_NAME = ITEM_NAME1
PRICE = PRICE1
```



# 联接单个源中的数据

如果希望对部分数据执行计算并联接已转换数据与原始数据，则可从同一源联接数据。使用此方法联接数据时，可以维护原始数据并在某个映射内转换部分数据。可以通过以下方式从同一源联接数据：

- 联接同一管道的两个分支。
- 联接同一源的两个实例。

## 联接同一管道的两个分支

从同一源联接数据时，可以创建该管道的两个分支。使管道产生分支时，必须至少将源限定符和联接器转换之间的转换添加到管道的一个分支中。必须联接已排序数据并为已排序输入配置联接器转换。

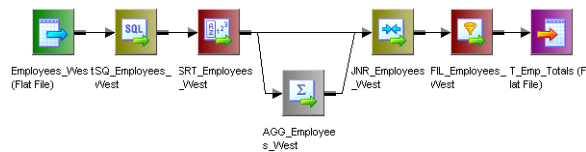
例如，具有使用以下端口的源：

- 员工
- 部门
- 销售总额

在目标中，希望查看生成的销售额高于部门平均销售额的员工。要执行该操作，请使用以下转换创建映射：

- **排序器转换**。对数据进行排序。
- **已排序汇总器转换**。计算销售数据的平均值并按部门进行分组。执行此汇总时，将丢失单个员工的数据。要维护员工数据，必须将管道的分支传递给汇总器转换，并将具有相同数据的分支传递给联接器转换以维护原始数据。联接管道的两个分支时，将联接汇总数据与原始数据。
- **已排序联接器转换**。使用已排序联接器转换将已排序的汇总数据与原始数据联接起来。
- **筛选器转换**。比较平均销售数据和每个员工的销售数据，并筛选销售额低于上述平均值的员工。

下图显示了将同一管道的两个分支联接起来的映射：



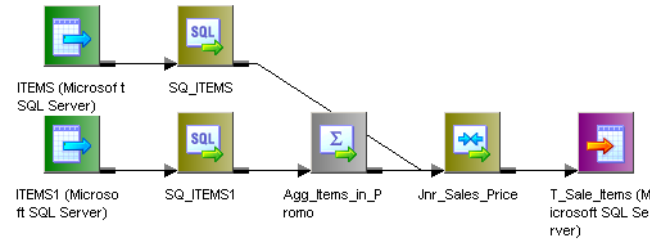
**注意：**还可以从同一转换的输出组中联接数据，例如自定义转换或 XML 源限定符转换。在每个输出组和联接器转换之间放置排序器转换，并将联接器转换配置为接收已排序的输入。

如果联接器转换从一个分支接收到数据的时间远远晚于从另一个分支接收到数据的时间，则联接这两个分支可能会影响性能。联接器转换将缓存第一个分支中的所有数据，并在缓存已满时将其写入磁盘。然后，联接器转换接收第二个分支中的数据时，必须从磁盘中读取数据。这会减慢处理速度。

## 联接同一源的两个实例

还可以通过创建源的第二个实例来联接同一源数据。创建其他源实例后，可以联接两个源实例中的管道。如果希望联接未排序数据，则必须创建同一源的两个实例并联接管道。

下图显示了同一源中通过联接器转换联接的两个实例：



**注意：**使用此方法联接数据时，集成服务会读取每个源实例的源数据，因此其性能低于联接管道两个分支时的性能。

## 有关联接单个源中的数据的准则

决定是联接管道的分支还是联接源的两个实例时，请遵循以下准则：

- 源较大或者如果只能读取源数据一次时，请联接管道的两个分支。例如，您可以仅从消息队列中读取一次源数据。
- 使用已排序数据时，请联接管道的两个分支。如果源数据未排序，而您使用排序器转换对该数据进行排序，请在数据排序完成后对管道进行分支。
- 需要将编块转换添加到源和联接器转换之间的管道中时，请联接源的两个实例。
- 如果某个管道的处理速度慢于其他管道，请联接源的两个实例。
- 如果需要联接未排序数据，请联接源的两个实例。

## 阻止源管道

使用联接器转换运行会话时，根据映射配置以及您是否为已排序输入配置联接器转换，集成服务将阻止和解锁源数据。

### 未排序联接器转换

当集成服务处理未排序的联接器转换时，它将先读取所有主行，之后再读取详细信息行。要确保其在读取详细信息行之前先读取所有主行，集成服务会在缓存来自主源的行时阻止详细信息源。集成服务读取并缓存所有主行后，将取消阻止详细信息源并读取详细信息行。某些包含未排序联接器转换的映射将违反数据流验证。

### 已排序联接器转换

集成服务处理已排序的联接器转换时，它将基于映射配置阻止数据。如果联接器转换的主输入和详细信息输入源自不同的源，阻止逻辑是有可能的。

如果可以在不同时阻止目标加载顺序组中所有源的情况下执行操作，集成服务将使用阻止逻辑来处理联接器转换。否则，它将不使用阻止逻辑。相反，它将在缓存中存储更多行。

集成服务可以使用阻止逻辑来处理联接器转换时，它将在缓存中存储较少行，以提高性能。

## 缓存主行

集成服务处理连接器转换时，会同时从两个源读取行，并基于主行建立索引和数据缓存。然后集成服务会基于详细源数据和缓存数据执行联接。集成服务在缓存中存储的行数取决于分区类型、源数据以及是否为已排序输入配置连接器转换。要提高未排序连接器转换的性能，请将行数较少的源作为主源。要提高已排序连接器转换的性能，请将重复键值数较少的源作为主源。

# 使用事务

当集成服务处理连接器转换时，可以将转换逻辑一次性应用到事务中的所有数据、所有传入数据或一行数据。集成服务可以删除也可以保留事务边界，具体取决于映射配置和转换范围。可以使用转换范围属性来配置集成服务如何应用转换逻辑以及处理事务边界。

可以根据映射配置以及您想保留还是想删除事务边界，来配置转换范围值。

当联接以下源时，可以保留事务边界：

- **联接同一源管道的两个分支。**可以使用事务转换范围来保留事务边界。
- **联接两个源，并想为详细信息源保留事务边界。**可以使用行转换范围在详细信息管道中保留事务边界。

当联接以下源时，可以删除事务边界：

- **联接两个源或两个分支，并想删除事务边界。**可以使用“全部输入”转换范围，将转换逻辑应用于所有传入数据，并为两个管道删除事务边界。

下表总结了如何使用包含连接器转换的转换范围来保留事务边界：

转换范围	输入类型	集成服务行为
行	未排序	在详细信息管道中保留事务边界。
行	已排序	会话失败。
事务	已排序	当主管道和详细信息管道源自同一事务生成器时，将保留事务边界。当主管道和详细信息管道并非源自同一事务生成器时，会话将失败
事务	未排序	会话失败。
全部输入	已排序、未排序	删除事务边界。

**注意:** 如果使用包含“所有输入”或“事务”转换范围的实时数据，则会话将会失败。

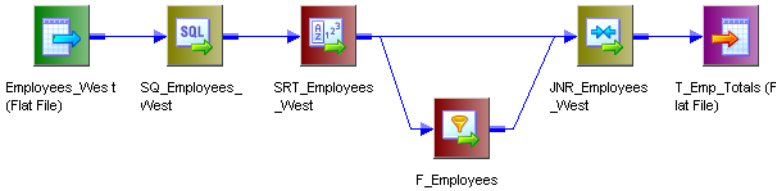
## 保留单个管道的事务边界

联接同一个源中的数据时，使用“事务”转换范围可保留单个管道的传入事务边界。当连接器转换联接同一个源中的数据时（同一管道的两个分支或一个事务生成器的两个输出组），请使用“事务”转换范围。此转换范围可用于已排序的数据及任何联接类型。

使用“事务”转换范围时，请确认主管道和详细管道是否源自同一事务控制点以及您使用的是否为已排序输入。例如，在[“保留单个管道的事务边界”页面上 251](#)中，排序器转换即事务控制点。在排序器转换和连接器转换之间不能放置其他事务控制点。在映射中，主管道和详细管道分支源自同一个事务控制点，并且集成服务使用连接器转换联接管道分支，保留事务边界。

下图显示了一个联接管道两个分支且保留事务边界的映射：

图 1. 联接两个管道分支时保留事务边界



## 在详细管道中保留事务边界

要在详细管道中保留事务边界，选择“行”转换范围。“行”转换范围允许集成服务一次处理一行数据。集成服务会缓存主数据并将详细数据与缓存的主数据相匹配。

如果源数据来源于实时源（如 IBM MQ Series），集成服务会在从详细源读取每条消息时，将缓存的主数据与该消息相匹配。

请使用普通或主外部联接类型（使用未排序数据）的“行”转换范围。

## 删除两个管道的事务边界

要联接两个源或两个分支的数据且无需保留事务边界时，可使用“所有输入”转换范围。使用“所有输入”时，集成服务将删除两个管道的传入事务边界，并作为打开事务输出转换中的所有行。在联接器转换中，可以根据您配置排序顺序的方式并发缓存或联接主管道中的数据。此转换范围可用于排序和未排序的数据及任何联接类型。

# 创建联接器转换

要使用联接器转换，请向映射添加联接器转换，设置输入源，并为转换配置条件、联接类型和排序类型。

要创建联接器转换，请执行以下操作：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择联接器转换。输入名称，然后单击“确定”。联接器转换的命名约定为 `JNR_TransformationName`。输入转换的说明。Designer 将创建联接器转换。
2. 将所有输入/输出端口从第一个源拖动到联接器转换中。默认情况下，Designer 会在联接器转换中为源字段创建输入/输出端口作为详细信息字段。您可以稍后编辑该属性。
3. 从第二个源中选择所有输入/输出端口并将它们拖动到联接器转换中。默认情况下，Designer 会配置第二个源字段集和主字段。
4. 双击联接器转换的标题栏可打开转换。
5. 单击“端口”选项卡。
6. 单击 M 列的任何框可切换源的主端口/详细端口关系。

**提示：**要提高未排序联接器转换的性能，请将行数较少的源作为主源。要提高已排序联接器转换的性能，请将重复键值数较少的源作为主源。

7. 为特定端口添加默认值。  
一些端口很可能包含空值，因为其中一个源中的字段可能为空。如果目标数据库不能处理空值，可以指定默认值。
8. 单击“条件”选项卡并设置联接条件。
9. 单击“添加”按钮可添加条件。可以添加多个条件。  
主端口和详细信息端口必须具有匹配的数据类型。联接器转换仅支持对等 (=) 联接。
10. 单击“属性”选项卡并配置转换的属性。  
**注意:** 您可以从“条件”选项卡中编辑联接条件。关键字 AND 可分隔多个条件。
11. 单击“确定”。
12. 单击“元数据扩展”选项卡可配置元数据扩展。

## 联接器转换提示

### 尽可能在数据库中执行联接。

与在会话中执行联接相比，在数据库中执行联接的速度更快。在某些情况下，不会发生这种现象，如联接来自两个不同数据库或平面文件系统的表。如果要在数据库中执行联接，请使用以下选项：

- 创建会话前存储过程以联接数据库中的表。
- 使用源限定符转换执行联接。

### 尽可能联接已排序数据。

通过将联接器转换配置为使用已排序输入，可以提高会话性能。将联接器转换配置为使用已排序数据时，集成服务会通过将磁盘输入和输出降至最低来提高性能。处理大型数据集时，将会获得最显著的性能提高。

### 对于未排序联接器转换，请将具有较少行的源指定为主源。

为了实现最佳性能和磁盘存储，请将具有较少行的源指定为主源。在会话期间，联接器转换会将主源的每一行与详细信息源进行比较。主源中的唯一行数越少，联接比较发生的迭代数越少，从而加快联接进程的速度。

### 对于已排序联接器转换，请将具有较少重复键值的源指定为主源。

为了实现最佳性能和磁盘存储，请将具有较少重复键值的源指定为主源。在处理已排序联接器转换时，集成服务一次缓存一百个键对应的行。如果主源中包含许多具有相同键值的行，集成服务必须缓存更多行，并且性能会降低。

## 第 17 章

# 查找转换

本章包括以下主题：

- [查找转换概览, 254](#)
- [查找源类型, 255](#)
- [已连接和未连接的查找, 257](#)
- [查找组件, 259](#)
- [查找属性, 261](#)
- [查找查询, 265](#)
- [查找条件, 269](#)
- [查找缓存, 270](#)
- [返回多个行, 271](#)
- [配置未连接的查找转换, 271](#)
- [数据库死锁弹性, 273](#)
- [创建查找转换, 274](#)
- [查找转换提示, 275](#)

## 查找转换概览

在映射中使用查找转换来查找平面文件、关系表、视图或同义词中的数据。可以从 PowerCenter 客户端和集成服务都可连接到的任何平面文件或关系数据库中导入查找定义。还可以从源限定符创建查找定义。您可以在一个映射中使用多个查找转换。查找转换可以是主动转换或被动转换。可以配置已连接或未连接的查找转换。

集成服务根据转换和查找条件中的查找端口来查询查找源。查找转换会将查找的结果返回到目标或其他转换中。您可以将查找转换配置为返回单个行或多个行。

使用查找转换执行以下任务：

- **获取相关值。**根据源中的值从查找表检索值。例如，源具有员工 ID。从查找表检索员工姓名。
- **获取多个值。**从查找表检索多个行。例如，返回部门中的所有员工。
- **执行计算。**从查找表检索值，并在计算中使用该值。例如，检索销售税百分比、计算税款并将税款返回到目标。
- **更新缓慢变化的维度表。**确定行是否存在于目标中。

配置查找转换以执行以下类型的查找：

- **关系或平面文件查找。**对平面文件或关系表执行查找。使用关系表作为查找源创建查找转换时，可以使用 ODBC 连接到查找源，并导入表定义作为查找转换的结构。使用平面文件作为查找源创建查找转换时，Designer 会调用平面文件向导。
- **管道查找。**在应用程序源（如 JMS 或 MSMQ）上执行查找。将源拖动到映射中，并将查找转换与源限定符相关联。集成服务为查找缓存检索源数据时，配置分区以提高性能。
- **已连接或未连接查找。**已连接查找转换会收到源数据，执行查找，并将数据返回到管道。未连接的查找转换未连接到源或目标。管道中的转换会通过 :LKP 表达式调用查找转换。未连接的查找转换会向调用转换返回一个列。
- **已缓存或未缓存的查找。**缓存查找源可提高性能。如果缓存查找源，可以使用动态缓存或静态缓存。默认情况下，查找缓存会保持静态，在会话期间不会更改。通过动态缓存，集成服务会在缓存中插入或更新行。缓存目标表作为查找源时，可以在缓存中查找值以确定目标中是否存在值。查找转换标记要插入的行或更新目标。

## 查找源类型

创建查找转换时，可以选择关系表、平面文件或源限定符作为查找源。

### 关系查找

使用关系表作为查找源创建查找转换时，可以使用 ODBC 连接到查找源，并导入表定义作为查找转换的结构。

对关系查找使用以下选项：

- 替代默认 SQL 语句以添加 WHERE 子句或查询多个表。
- 根据数据库支持，按高值或低值对空数据进行排序。
- 根据数据库支持，执行区分大小写的比较。

### 平面文件查找

使用平面文件作为查找源创建查找转换时，在存储库中选择平面文件定义或在创建转换时导入源。导入平面文件查找源时，Designer 会调用平面文件向导。

对平面文件查找使用以下选项：

- 通过配置一个文件列表作为查找文件名，使用间接文件作为查找源。
- 将排序的输入用于查找。
- 按高值或低值对空数据进行排序。
- 对平面文件查找使用区分大小写的字符串比较。

### 使用已排序输入

为已排序输入配置平面文件查找转换时，必须对条件列进行分组。如果条件列未分组，查找转换会返回错误的结果。要获得最佳缓存性能，请对条件列排序。

例如，某个查找转换具有以下条件：

```
OrderID = OrderID1
CustID = CustID1
```

在以下平面文件查找源中，键已分组，但未排序。集成服务可以缓存该数据，但性能可能不是最佳的。

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA502	F895S	Flashlight	Key data is grouped, but not sorted. CustID is out of order within OrderID.
1001	CA501	C530S	Compass	
1001	CA501	T552T	Tent	
1005	OK503	S104E	Safety Knife	Key data is grouped, but not sorted. OrderID is out of order.
1003	CA500	F304T	First Aid Kit	
1003	TN601	R938M	Regulator System	

在以下平面文件查找源中，键未分组。查找转换会返回错误的结果。

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA501	T552T	Tent	-
1001	CA501	C530S	Compass	-
1005	OK503	S104E	Safety Knife	-
1003	TN601	R938M	Regulator System	-
1003	CA500	F304T	First Aid Kit	-
1001	CA502	F895S	Flashlight	Key data for CustID is not grouped.

如果选择间接文件的已排序输入，则数据范围不能在文件中重叠。

## 管道查找

创建管道查找转换以对并非关系表或平面文件的应用程序源执行查找。管道查找转换具有源限定符作为查找源。您可以对除应用程序多组源限定符转换之外的所有数据源执行管道查找。

配置管道查找转换时，查找源和源限定符位于查找转换的不同管道中。源和源限定符位于不包含目标的部分管道中。集成服务读取此管道中的源数据，并将数据传递到查找转换以创建缓存。您可以在部分管道中创建多个分区以提高性能。

要在处理关系或平面文件查找源时提高性能，请创建管道查找转换而不是关系或平面文件查找转换。您可以创建分区以处理查找源并将其传递到查找转换。

创建连接或未连接的管道查找转换。

**注意：** 不要为具有实时管道查找源的会话启用 HA 发现。可能会产生意外结果。

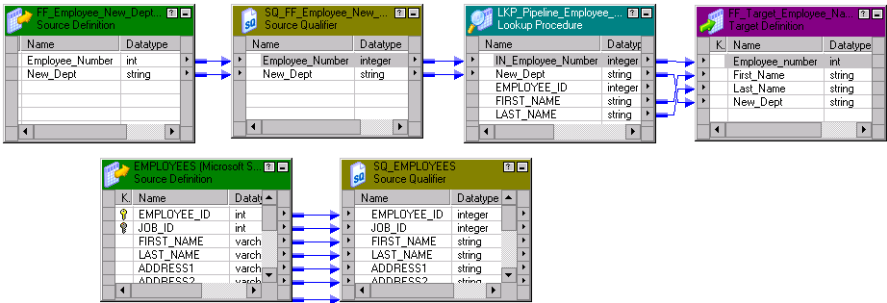


## 在映射中配置管道查找转换

包含管道查找转换的映射包括具有查找源和源限定符的部分管道。部分管道不包括目标。集成服务会检索此管道中的查找源数据，并将数据传递到查找缓存。

部分管道在会话属性中位于单独的目标加载顺序组中。可以在管道中创建多个分区以提高性能。不能通过部分管道配置目标加载顺序。

以下映射显示了一个映射，其包含管道查找转换以及处理查找源的部分管道。



该映射包含以下对象：

- 查找源定义和源限定符位于单独的管道中。集成服务在处理管道中的查找源数据后，会创建查找缓存。
- 平面文件源包含按员工编号排列的新部门名称。
- 管道查找转换会从源文件接收 Employee\_Number 和 New\_Dept。管道查找会对查找缓存中的 Employee\_ID 执行查找。它会从查找缓存中检索员工名字和姓氏。
- 平面文件目标会从查找转换中收到 Employee\_ID、First\_Name、Last\_Name 和 New\_Dept。

## 已连接和未连接的查找

可以配置已连接的查找转换或未连接的查找转换。已连接的查找转换是具有连接到映射中其他转换的输入端口和输出端口的转换。未连接的查找转换将显示在映射中，但未连接到其他转换。

未连接的查找转换将从表达式转换或汇总器转换等转换中的 :LKP 表达式的结果中接收输入。:LKP 表达式会将参数传递至查找转换，并从查找转换中接收结果。:LKP 表达式可以将查找结果传递至表达式转换或汇总器转换中的其他表达式以筛选结果。

下表列出了已连接的查找与未连接的查找之间的差异：

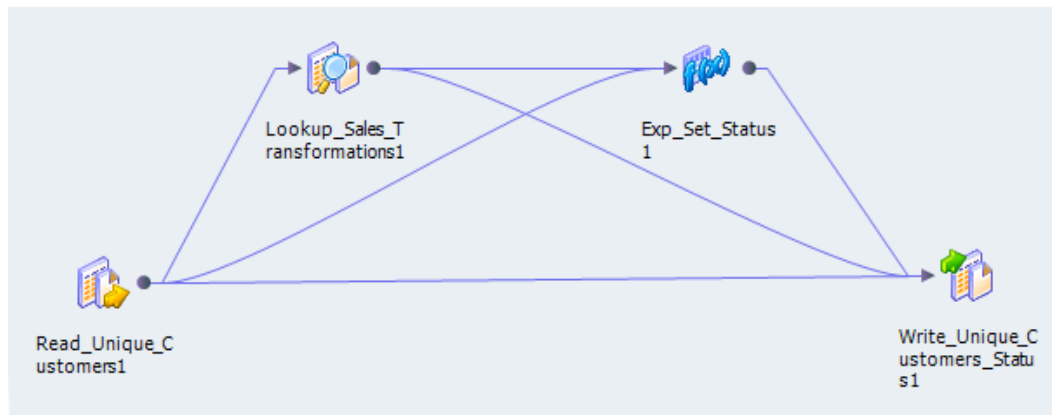
已连接的查找	未连接的查找
直接从管道中接收输入值。	从其他转换的 :LKP 表达式的结果中接收输入值。
使用动态或静态缓存。	使用静态缓存。
缓存中包括查找条件中的查找源列和作为输出端口的查找源列。	缓存中包括查找条件中的所有查找端口和输出端口以及查找/返回端口。
从同一行返回多列或者插入到动态查找缓存中。	从每一行返回一列至返回端口。

已连接的查找	未连接的查找
如果不存在与查找条件匹配的项，则集成服务将为所有输出端口返回默认值。如果配置动态缓存，集成服务会将行插入到缓存中或者保持行不变。	如果不存在与查找条件匹配的项，则集成服务将返回 NULL。
如果存在与查找条件匹配的项，则集成服务将为所有查找/输出端口返回查找条件的结果。如果配置动态缓存，集成服务会更新缓存中的行或者保留行不变。	如果存在与查找条件匹配的项，则集成服务会将查找条件的结果返回至返回端口。
将多个输出值传递到其他转换。将查找/输出端口链接至其他转换。	将一个输出值返回至其他转换。查找转换返回端口会将值传递到其他转换中包含 :LKP 表达式的端口。
支持用户定义的默认值。	不支持用户定义的默认值。

## 已连接的查找

已连接的查找转换是连接到映射中的源或目标的查找转换。

下图显示了具有已连接的查找转换的映射：



当运行包含已连接查找转换的映射时，集成服务将执行以下步骤：

1. 集成服务会将值从其他转换传递至查找转换的输入端口。
2. 对于每个输入行，集成服务将根据转换中的查找端口和查找条件查询查找源或缓存。
3. 如果该转换未缓存，或者该转换使用静态缓存，则集成服务将返回查找查询中的值。  
如果转换使用动态缓存，则集成服务在缓存中找不到行时将行插入缓存。集成服务在缓存中找到行时，会更新缓存中的行或者保留不更改。它将行标记为插入、更新或未更改。
4. 集成服务将返回查询中的数据，并将该数据传递至映射中的下一转换。  
如果转换使用动态缓存，可以将行传递给筛选器或路由器转换以将新行筛选到目标中。

**注意：**除非另行指定，否则本章将讨论已连接的查找转换。

## 未连接的查找

未连接的查找转换是指未连接到映射中的源或目标的查找转换。在支持表达式的转换中使用 :LKP 表达式调用查找。

更新缓慢变化的维度表是未连接的查找转换的常见用法。有关缓慢变化的维度表的详细信息，请访问 Informatica 知识库，网址为 <http://mysupport.informatica.com>。

查找表达式的语法为 :LKP lookup\_transformation\_name(argument, argument, ...)

列出每个参数的顺序必须与查找转换中查找条件的顺序一致。查找转换通过查找转换返回端口返回查询的结果。调用此查找的转换会在包含 :LKP 表达式的端口中收到查找结果值。如果此查找查询无法返回值，则端口服务将收到空值。

执行未连接的查找时，可在同一映射中多次执行相同的查找。可在另一表达式中测试查找结果，并基于这些结果来筛选行。

如果运行的映射包含未连接的查找转换，则集成服务将执行以下步骤：

1. 未连接的查找转换将从汇总器转换、表达式转换或更新策略转换等其他转换中的 :LKP 表达式的结果中接收输入值。
2. 集成服务根据查找转换中的查找端口和条件来查询查找源或缓存。
3. 集成服务通过查找转换的返回端口返回一个值。
4. 集成服务将返回值传递到包含 :LKP 表达式的端口。

## 查找组件

在映射中配置查找转换时，定义以下组件：

- 查找源
- 端口
- 属性
- 条件

## 查找源

对查找源使用平面文件、关系表或源限定符。创建查找转换时，可以从以下位置创建查找源：

- 存储库中的关系源或目标定义
- 存储库中的平面文件源或目标定义
- 集成服务和 PowerCenter 客户端计算机可以连接到的表或文件
- 映射中的源限定符定义

查找表可以是单个表，或者可以使用查找 SQL 替代在同一数据库中联接多个表。集成服务会查询查找表或表的内存中缓存，以获取进入查找转换的所有传入行。

集成服务可以使用 ODBC 或本机驱动程序连接到查找表。配置本机驱动程序以实现最佳性能。

## 索引和查找表

如果您有特权修改包含查找表的数据库，可以通过向查找表添加索引来缩短查找初始化时间。可以为大量查找表提高性能。由于集成服务会查询、排序和比较查找列中的值，因此索引需要在查找条件中包括每个列。

您可以通过对以下类型的查找编制索引来提高性能：

- **已缓存查找。**您可以通过对查找 ORDER BY 中的列编制索引来提高性能。会话日志包含 ORDER BY 子句。
- **未缓存查找。**由于集成服务为传递到查找转换中的每个行发出 SELECT 语句，因此可以通过为查找条件中的列编制索引来提高性能。

## 查找端口

“端口”选项卡包含输入和输出端口。“端口”选项卡还包括查找端口，其表示要从查找源中返回的数据列。未连接的查找转换会将一个数据列返回到此端口中的调用转换。未连接的查找转换具有一个返回端口。

下表介绍了查找转换中的端口类型：

端口	查找的类型	说明
I	已连接 未连接	输入端口。为您要在查找条件中使用的每个查找端口创建输入端口。您必须在每个查找转换中具有至少一个输入或输入/输出端口。
O	已连接 未连接	输出端口。为您要链接到另一个转换的每个查找端口创建输出端口。可以指定输入端口和查找端口作为输出端口。对于连接的查找，必须至少具有一个输出端口。对于未连接的查找，选择查找端口作为返回端口 (R) 以传递返回值。
L	已连接 未连接	查找端口。Designer 会指定查找源中的各个列作为查找 (L) 和输出端口 (O)。
R	未连接	返回端口。仅在未连接的查找转换中使用。指定要根据查找条件返回的数据列。您可以指定一个查找端口作为返回端口。

查找转换还会启用您使用动态缓存时配置的关联表达式属性。关联的表达式属性包含用于更新查找缓存的数据。它可以包含表达式来更新动态缓存或者可以包含输入端口名称。

使用以下准则来配置查找端口：

- 如果从平面文件查找中删除查找端口，会话会失败。
- 如果映射不使用查找端口，则可以从关系查找中删除查找端口。这会减少集成服务运行会话所需的内存量。

## 查找属性

在“属性”选项卡上，配置诸如关系查找的 SQL 替代等属性以及查找源名称缓存属性。

**相关主题：**

- [“查找属性”页面上 261](#)

## 查找条件

在“条件”选项卡上，输入希望集成服务用于在查找源中查找数据的一个或多个条件。

相关主题：

- “查找条件” 页面上 269

# 查找属性

在“查找属性”选项卡上，配置诸如缓存和多个匹配项等查找属性。配置查找条件或 SQL 语句来查询查找表。还可以更改查找表名称。

在创建映射时，为每个查找转换配置属性。在创建会话时，可以为每个转换替代诸如索引和数据缓存大小等属性。

下表介绍了查找转换属性：

选项	查找类型	说明
查找 SQL 替代	关系	替代默认 SQL 语句以查询查找表。 指定您希望集成服务用于查询查找值的 SQL 语句。在启用了查找缓存的情况下使用。
查找表名称	管道 关系	转换从中查找和缓存值的表或源限定符的名称。在创建查找转换时，选择源、目标或源限定符作为查找源。在创建查找转换时，还可以从另一个数据库导入表、视图或同义词。 如果输入查找 SQL 替代，则无需输入查找表名称。
查找源筛选器	关系	限制集成服务根据查找转换中任意端口的数据值所执行的查找。在启用了查找缓存的情况下使用。
启用查找缓存	平面文件 管道 关系	指示集成服务在会话期间是否缓存查找值。 启用查找缓存时，集成服务会在会话期间查询一次查找源，对值进行缓存，并查找缓存中的值。缓存查找值可以提高会话性能。 如果禁用缓存，则每次将行传入转换时，集成服务都将对查找源发出一条用于查找值的 select 语句。 <b>注意：</b> 集成服务始终缓存平面文件查找和管道查找。
多项匹配时的查找策略	平面文件 管道 关系	在查找转换发现匹配查找条件的多个行时，确定要返回的行。选择以下值之一： <ul style="list-style-type: none"><li>- 使用第一个值。返回与查找条件匹配的第一行。</li><li>- 使用最后一个值。返回与查找条件匹配的最后一个行。</li><li>- 使用所有值。返回所有匹配的行。</li><li>- 使用任何值。集成服务返回与查找条件匹配的第一个值。它会根据键端口而不是所有查找转换端口创建索引。</li><li>- 报表错误。集成服务会报告错误，且不返回行。如果不启用“更新时输出旧值”选项，则对于动态查找，“多项匹配时的查找策略”选项会设置为“报表错误”。</li></ul>
查找条件	平面文件 管道 关系	显示在“条件”选项卡中设置的查找条件。

选项	查找类型	说明
连接信息	关系	<p>指定包含查找表的数据库。可以在映射、会话或参数文件中定义数据库：</p> <ul style="list-style-type: none"> <li>- 映射。选择连接对象。还可以指定数据库连接类型。如果连接为关系连接，则在连接名称之前键入 <b>Relational:</b>。如果连接为应用程序连接，则在连接名称之前键入 <b>Application:</b>。</li> <li>- 会话。使用 \$Source 或 \$Target 连接变量。如果使用其中一个变量，则查找表必须驻留在源或目标数据库中。在每个变量的会话属性中指定数据库连接。</li> <li>- 参数文件。使用会话参数 \$DBConnectionName 或 \$AppConnectionName，并在参数文件中对其进行定义。</li> </ul> <p>默认情况下，在创建查找转换时如果选择源表，则 Designer 会指定 \$Source，如果选择目标表，则会指定 \$Target。可以在会话属性中替代这些值。</p> <p>如果无法确定数据库连接的类型，则集成服务处理会话失败。</p>
源类型	平面文件 管道 关系	指示查找转换从关系表、平面文件或源限定符中读取值。
跟踪级别	平面文件 管道 关系	设置包括在会话日志中的详细信息量。
查找缓存目录名称	平面文件 管道 关系	<p>指定在配置查找转换时用于构建查找缓存文件以缓存查找源的目录。此外，在选择“查找持久性”选项时，还会保存持久性查找缓存文件。</p> <p>默认情况下，集成服务使用为集成服务配置的 \$PMCacheDir 目录。</p>
查找缓存持久性	平面文件 管道 关系	指示集成服务是否使用持久性查找缓存（至少由两个缓存文件组成）。如果查找转换是为持久性查找缓存配置的，但不存在持久性查找缓存文件，则集成服务将在会话期间创建这些文件。在启用了查找缓存的情况下使用。
查找数据缓存大小查找索引缓存大小	平面文件 管道 关系	<p>默认为“自动”。指示集成服务分配给数据缓存和内存中索引的大小上限。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。</p> <p>如果初始化会话期间集成服务无法分配配置的内存量，则处理会话失败。集成服务无法在内存中存储所有数据缓存数据时，会对磁盘进行分页。</p> <p>在启用了查找缓存的情况下使用。</p>
动态查找缓存	平面文件 管道 关系	<p>指示使用动态查找缓存。查找缓存将行传递到目标表时，插入或更新查找缓存中的行。</p> <p>在启用了查找缓存的情况下使用。</p>

选项	查找类型	说明
更新时输出旧值	平面文件 管道 关系	<p>在启用了动态缓存的情况下使用。如果启用此属性，集成服务通过查找/输出端口输出旧值。集成服务更新缓存中的行时，其在基于输入数据更新行之前会输出存在于查找缓存中的值。当集成服务在缓存中插入行时，将输出空值。</p> <p>如果禁用此属性，集成服务通过查找/输出和输入/输出端口输出相同的值。</p> <p>默认情况下该属性处于启用状态。</p>
更新动态缓存条件	平面文件 管道 关系	<p>指示是否更新动态缓存的表达式。使用查找端口或输入端口创建表达式。表达式可以包含输入值或查找缓存中的值。如果条件为 true 并且数据存在于缓存中，集成服务将更新缓存。在启用了动态缓存的情况下使用。默认值为 true。</p>
缓存文件名前缀	平面文件 管道 关系	<p>与持久性查找缓存配合使用。指定要用于持久性查找缓存文件的文件名前缀。集成服务会使用该文件名前缀作为其保存到磁盘的持久性缓存文件的文件名。输入前缀。不要输入 .idx 或 .dat。</p> <p>您可以为文件名前缀输入参数或变量。可使用您在参数文件中可定义的任何参数或变量类型。</p> <p>如果指定的持久性缓存文件已存在，则集成服务会通过这些文件构建内存缓存。如果指定的持久性缓存文件不存在，则集成服务将重建持久性缓存文件。</p>
从查找源重新缓存	平面文件 管道 关系	<p>在启用了查找缓存的情况下使用。如果选择该项，集成服务会在首次调用查找转换实例时通过查找源重新构建查找缓存。</p> <p>如果使用持久性查找缓存，其会先重新构建持久性缓存文件，然后再使用该缓存。如果不使用持久性查找缓存，其会先在内存中重新构建查找缓存，然后再使用该缓存。</p>
插入 Else 更新	平面文件 管道 关系	<p>在启用了动态缓存的情况下使用。适用于进入查找转换且行类型为“插入”的行。如果启用该项，集成服务会在缓存中插入行并更新现有行。如果禁用该项，集成服务不会更新现有行。</p>
更新 Else 插入	平面文件 管道 关系	<p>在启用了动态缓存的情况下使用。适用于进入查找转换且行类型为“更新”的行。</p> <p>如果启用该项，集成服务将更新现有行，如果是新行则插入行。如果禁用该项，集成服务不插入新行。</p>
日期时间格式	平面文件	<p>单击“打开”按钮可选择日期时间格式。定义格式和字段宽度。毫秒、微秒或纳秒格式的字段宽度为 29。</p> <p>如果没有为端口选择日期时间格式，则可以输入任意日期时间格式。默认值为 MM/DD/YYYY HH24:MI:SS。日期时间格式不会更改端口的大小。</p>
千位分隔符	平面文件	<p>如果不为端口定义千位分隔符，则集成服务会使用此处定义的属性。</p> <p>可以选择无分隔符、逗号或句点。默认设置为无分隔符。</p>
小数分隔符	平面文件	<p>如果不在查找定义或“端口”选项卡上为特定字段定义小数分隔符，集成服务会使用此处定义的属性。</p> <p>可以选择逗号或句点作为小数分隔符。默认设置为句点。</p>
区分大小写的字符串比较	平面文件 管道	<p>对字符串列执行查找时，集成服务将使用区分大小写的字符串比较。</p> <p>对于关系查找，区分大小写的比较取决于数据库支持。</p>

选项	查找类型	说明
空值排序	平面文件 管道	确定集成服务对空值进行排序的方式。您可以选择按高值或低值对空值进行排序。默认情况下，集成服务将按高值对空值进行排序。该操作将覆盖集成服务配置，以便将比较运算符中的空值视为高值、低值或空值。 对于关系查找，空值排序取决于数据库默认值。
已排序输入	平面文件 管道	指示查找文件数据是否采用已排序的顺序。这会提高文件查找的查找性能。如果启用已排序输入，则条件列不会进行分组，集成服务处理会话将会失败。如果条件列已分组但未排序，集成服务会像未配置已排序输入一样处理查找。
查找源为静态	平面文件 管道 关系	查找源不会在会话中更改。
预构建查找缓存	平面文件 管道 关系	允许集成服务在查找转换接收数据之前构建查找缓存。集成服务可以同时构建多个查找缓存文件来提高性能。 您可以在映射或会话中配置此选项。如果将查找转换选项配置为“自动”，集成服务会使用会话级别的设置。 配置以下选项之一： <ul style="list-style-type: none"> <li>- 自动。集成服务使用在会话中配置的值。</li> <li>- 始终允许。集成服务可在查找转换接收第一个源行之前构建查找缓存。集成服务创建其他管道来构建缓存。</li> <li>- “始终不允许”。集成服务无法在查找转换接收第一行之前构建查找缓存。</li> </ul> 您必须配置集成服务可以并发构建的管道数。配置“用于查找缓存创建的其他并发管道”会话属性。如果该属性的值大于零，则集成服务可以预构建查找缓存。
子秒精度	关系	指定日期时间端口的子秒精度。 对于关系查找，可以更改日期时间数据具有可编辑小数位数的数据库的精度。可以为 Oracle 时间戳、Informix 日期时间和 Teradata 时间戳数据类型更改子秒精度。 输入在从 0 到 9 范围内的正整数值。默认值为 6 微秒。如果启用下推优化，则数据库会返回完整的日期时间值，不管子秒精度设置如何。

## 在会话中配置查找属性

配置会话时，可以配置对于会话唯一的查找属性：

- **平面文件查找。**配置查找位置信息，例如源文件目录、文件名和文件类型。
- **关系查找。**可以在会话属性中定义 `$Source` 和 `$Target` 变量。还可以替代连接信息，以使用 `$DBConnectionName` 或 `$AppConnectionName` 会话参数。
- **管道查找。**配置查找源文件属性，例如源文件目录、文件名和文件类型。如果源为关系表或应用程序源，则配置连接信息。

## 在会话中配置平面文件查找

在会话中配置平面文件查找时，在“映射”选项卡的“转换”视图上配置查找源文件属性。选择查找转换，然后在转换的会话属性中配置平面文件属性。



下表介绍了为平面文件查找配置的会话属性：

属性	说明
查找源文件目录	输入目录名称。默认情况下，集成服务会在进程变量目录 \$PMLookupFileDir 中寻找查找文件。 可以输入完整路径和文件名。如果在“查找源文件名”字段中指定目录和文件名，则清除该字段。集成服务在运行会话时，会将该字段与“查找源文件名”字段连接在一起。 还可以输入 \$InputFileName 会话参数来配置文件名。
查找源文件名	查找文件的名称。如果使用间接文件，则输入希望集成服务读取的间接文件的名称。 还可以输入查找文件参数 \$LookupFileName，为会话更改查找文件的名称。 如果在“源文件目录”字段中配置目录和文件名，则清除“查找源文件名”。集成服务会将查找源文件名与会话的“查找源文件目录”字段连接在一起。例如，“查找源文件目录”字段包含“C:\lookup_data\”，而“查找源文件名”字段包含“filename.txt”。当集成服务开始会话时，它会查找“C:\lookup_data\filename.txt”。
查找源文件类型	指示查找源文件包含的是源数据还是具有相同文件属性的文件的列表。如果查找源文件包含源数据，则选择“直接”。如果查找源文件包含文件列表，则选择“间接”。 选择“间接”时，集成服务会为所有文件创建一个缓存。如果将已排序输入与间接文件一起使用，则确保文件中数据的范围不重叠。如果数据范围重叠，则集成服务会将查找作为未排序的查找源进行处理。

### 在会话中配置关系查找

在会话中配置关系查找时，在“映射”选项卡的“转换”视图上为查找数据库配置连接。选择查找转换，然后在转换的会话属性中配置连接。

从以下选项中进行选择，从而为关系查找转换配置连接：

- 选择关系或应用程序连接。
- 使用 \$Source 或 \$Target 连接变量配置数据库连接。
- 配置会话参数 \$DBConnectionName 或 \$AppConnectionName，然后在参数文件中定义会话参数。

### 在会话中配置管道查找

在会话中配置管道查找时，在“映射”选项卡的“源”节点上配置查找源文件的位置或查找表的连接。选择表示查找源的源限定符。

## 查找查询

集成服务根据在查找转换中配置的端口和属性来查询查找。当第一行进入查找转换时，集成服务将运行默认查找查询。

如果对关系表使用关系查找或管道查找，可以替代查找查询。如果对关系表使用查找，可以替代查找查询。可以使用替代来更改 ORDER BY 子句以及添加 WHERE 子句或在缓存查找数据之前转换该数据。可以使用替代来添加 WHERE 子句或在缓存查找数据之前转换该数据。

如果在查找查询上配置 SQL 替代和筛选器，集成服务将忽略筛选器。

## 默认查找查询

默认查找查询包含以下语句：

SELECT

SELECT 语句包括映射中的所有查找端口。要查看查找查询的 SELECT 语句，请选择查找 SQL 替代属性。

SELECT

SELECT 语句包括映射中的所有查找端口。要查看查找查询的 SELECT 语句，请选择使用自定义查询属性。

ORDER BY

ORDER BY 子句按照列在查找转换中出现的顺序对列进行排序。集成服务生成 ORDER BY 子句。生成默认 SQL 时不能对此进行查看。

## 替代查找查询

查找 SQL 替代与在源限定符转换中输入自定义查询类似。您可以替代关系查找的查找查询。可以输入整个替代，也可以生成和编辑默认的 SQL 语句。当 Designer 针对查找 SQL 替代生成默认 SQL 语句时，将包括查找条件中的查找/输出端口和查找/返回端口。

可以输入不同的查找查询或编辑现有查找查询，其中包括查找端口、输出端口和返回端口。

### 替代 ORDER BY 子句

默认情况下，集成服务生成已缓存查找的 ORDER BY 子句。ORDER BY 子句包含所有查找条件端口。要提高性能，可禁止默认 ORDER BY 子句，并输入包含较少列的替代 ORDER BY。

**注意：**替代 SQL 必须返回按查找键排序的数据。当转换在查找中检索所有行时，集成服务将按已排序顺序使用键构建数据缓存。如果未对行排序，集成服务将无法从缓存中检索所有行。如果数据未基于键排序，可能会获得异常结果。

如果使用下推优化，则无法替代 ORDER BY 子句或禁止生成的带有注释符号的 ORDER BY 子句。

集成服务始终生成 ORDER BY 子句，即使您在替代中输入了一个子句也是如此。将两个短划线 “--” 置于 ORDER BY 替代之后可禁止生成的 ORDER BY 子句。例如，某个查找转换使用以下查找条件：

```
ITEM_ID = IN_ITEM_ID  
PRICE <= IN_PRICE
```

该查找转换中包含映射中使用的两个查找条件端口，即 ITEM\_ID 和 PRICE。输入包含一个或更多列的 ORDER BY 子句时，请按照与查找条件中的端口相同的顺序输入列。还必须用引号引起所有数据库保留字。在查找 SQL 替代中输入以下查找查询：

```
SELECT ITEMS_DIM.ITEM_NAME AS ITEM_NAME, ITEMS_DIM.PRICE AS PRICE, ITEMS_DIM.ITEM_ID AS ITEM_ID FROM  
ITEMS_DIM ORDER BY ITEMS_DIM.ITEM_ID --
```

要替代关系查找的默认 ORDER BY 子句，请完成以下步骤：

1. 在该查找转换中生成查找查询。
2. 输入包含与查找条件中所示相同顺序条件端口的 ORDER BY 子句。
3. 将两个短划线 “--” 作为注释符号置于 ORDER BY 子句之后可禁止集成服务生成的 ORDER BY 子句。

如果使用 ORDER BY 子句替代该查找查询而不添加注释符号，查找将失败。

**注意：**Sybase 存在 16 列 ORDER BY 的限制。如果该查找转换包含 16 个以上查找/输出端口（包括查找条件中的端口），请替代 ORDER BY 子句或使用多个查找转换来查询查找表。

## 保留字

如果任何查找名称或列名称包含数据库预留字（例如 MONTH 或 YEAR），则当集成服务对数据库执行 SQL 时会话失败，并出现数据库错误。您可以在集成服务安装目录中创建并维护一个预留字文件 reswords.txt。

您可以在集成服务安装目录中创建并维护一个预留字文件 reswords.txt。当集成服务初始化会话时，它会搜索 reswords.txt 文件并在预留字两侧加上引号，然后对源、目标和查找数据库执行 SQL。

您可以在集成服务安装目录中创建并维护一个预留字文件 reswords.txt。当集成服务初始化映射时，它会搜索 reswords.txt 文件并在预留字两侧加上引号，然后对源、目标和查找数据库执行 SQL。

您可能需要启用某些数据库（如 Microsoft SQL Server 和 Sybase）才能使用关于加引号的标识符的 SQL-92 标准。使用连接环境 SQL 发出命令。例如，对于 Microsoft SQL Server，使用以下命令：

您可能需要启用某些数据库（如 Microsoft SQL Server 和 Sybase）才能使用关于加引号的标识符的 SQL-92 标准。使用环境 SQL 发出命令。例如，对于 Microsoft SQL Server，使用以下命令：

```
SET QUOTED_IDENTIFIER ON
```

## 替代查找查询的准则

替代查找查询时，应使用特定规则和准则。

替代查找 SQL 查询时，请考虑以下准则：

- 可以替代关系查找的查找 SQL 查询。
- 生成默认查询，然后配置替代。这样可以确保所有查找/输出端口均包含在查询中。如果在 SELECT 语句中增加或减少端口，会话将失败。
- 添加源查找筛选器以筛选添加到查找缓存中的行。这样可确保集成服务在与 WHERE 子句匹配的动态缓存和目标表中插入行。
- 如果多个查找转换共享一个查找缓存，请对每个查找转换使用相同的查找 SQL 替代。
- 在配置返回所有行的查找转换时，集成服务将使用已排序键构建查找缓存。当转换在查找中检索所有行时，集成服务将按已排序顺序使用键构建数据缓存。如果未对行排序，集成服务将无法从缓存中检索所有行。如果数据未基于键排序，可能会获得异常结果。
- ORDER BY 子句必须按照列在查找条件中出现的顺序包含条件端口。
- 如果要替代 ORDER BY 子句，请使用注释符号禁止查找转换生成的 ORDER BY 子句。
- 如果使用下推优化，则无法替代 ORDER BY 子句或禁止生成的带有注释符号的 ORDER BY 子句。
- 如果查找查询中的表名或列名包含保留字，应为保留字加上引号。
- 要替代未缓存查找的查找查询，应选择集成服务找到多个匹配项时返回任何值。
- 不能在默认 SQL 语句中添加或删除任何列。
- Developer tool 不会验证 SQL 查询的语法。如果未连接的查找查询中的 SQL 替代无效，映射将失败。

## 替代查找查询的步骤

按照以下步骤替代默认的查找 SQL 查询：

1. 在“属性”选项卡上，在“查找 SQL 替代”字段中打开 SQL 编辑器。
2. 单击“生成 SQL”生成默认 SELECT 语句。输入查找 SQL 替代。
3. 连接到数据库，然后单击“验证”以测试查找 SQL 替代。
4. 单击“确定”以返回“属性”选项卡。

## 未缓存查找的 SQL 替代

可以为未缓存查找定义 SQL 替代。集成服务不会从未缓存查找的替代语句中构建缓存。可以在替代 SELECT 语句中使用 SQL 函数。可以替代所有 SQL 查询，包括 WHERE 和 ORDER BY 子句。

当生成默认 SELECT 语句时，Designer 会根据查找条件生成 SELECT 语句，其中包含查找和输出端口以及 WHERE 子句。如果查找转换是未连接查找，则 SELECT 语句包含查找端口和返回端口。集成服务不会从您在查找转换的“条件”选项卡中配置的条件生成 WHERE 子句。

SELECT 查询中的每列使用别名定义输出列。请不要在 SQL 语句中更改此语法，否则查询会失败。要引用 WHERE 子句中的输入端口，请配置参数绑定。以下示例包括引用名称端口的 WHERE 子句：

```
SELECT EMPLOYEE.NAME as NAME, max(EMPLOYEE.ID) as ID from EMPLOYEE WHERE EMPLOYEE.NAME=?NAME1?
```

未缓存查找的 SQL 编辑器在“端口”选项卡中显示输入端口和查找端口。

如果向 SQL 语句中添加函数，返回数据类型必须与 ALIAS 列的数据类型匹配。例如，ID 的数据类型与 MAX 函数的返回类型匹配：

```
SELECT EMPLOYEE.NAME as NAME, MAX(EMPLOYEE.ID) as ID FROM EMPLOYEE
```

**注意：**不能将 SQL 替代中的子查询用于未缓存的查找。

## 查找源筛选器

您可以为已启用缓存的关系查找转换配置查找源筛选器。添加查找源筛选器以限制集成服务在查找源表中执行的查找数量。

配置查找源筛选器时，集成服务基于筛选器语句的结果执行查找。例如，您可能需要检索 ID 大于 510 的每位员工的姓氏。

在 EmployeeID 列上配置以下查找源筛选器：

```
EmployeeID >= 510
```

EmployeeID 是查找转换中的输入端口。当集成服务读取源行时，如果 EmployeeID 值大于 510 它将在缓存上执行查找。当 EmployeeID 小于或等于 510 时，查找转换不会检索姓氏。

当为针对下推优化配置的映射向查找查询添加查找源筛选器时，集成服务会创建一个视图来表示 SQL 替代。集成服务针对此视图运行 SQL 查询以将转换逻辑推送到数据库中。

当为针对下推优化配置的会话向查找查询添加查找源筛选器时，集成服务会创建一个视图来表示 SQL 替代。集成服务针对此视图运行 SQL 查询以将转换逻辑推送到数据库中。

## 筛选查找源行

添加查找源筛选器可限制集成服务对关系查找源执行的查找数目。

1. 在 Mapping Designer 或 Transformation Developer 中，打开查找转换。
2. 选择**属性**选项卡。
3. 验证缓存是否已启用。
4. 在**查找源筛选器**字段中单击**打开**按钮。
5. 在 SQL 编辑器中，选择输入端口或输入您想要筛选的任何查找转换端口。
6. 输入筛选条件。  
不在筛选条件中包括关键字 WHERE。将字符串映射参数和变量放在字符串标识符内。
7. 要验证条件，选择具有包含在查询中的源的 ODBC 数据源。
8. 输入用于连接到此数据库的用户名和密码。

## 9. 单击验证。

此时，Designer 将运行查询并报告其语法是否正确。

# 查找条件

集成服务使用查找条件在查找源中查找数据。查找条件类似于 SQL 查询中的 WHERE 子句。配置查找转换中的查找条件时，将源数据中一列或多列的值与查找源或缓存中的值进行比较。

例如，源数据中包含 employee\_number。查找源表中包含 employee\_ID、first\_name 和 last\_name。配置以下查找条件：

```
employee_ID = employee_number
```

对于每个 employee\_number，集成服务将从查找源返回 employee\_ID、last\_name 和 first\_name 列。

集成服务可以从查找源返回多行。配置以下查找条件：

```
employee_ID > employee_number
```

集成服务将返回所有大于源员工编号的 employee\_ID 编号的行。

输入查找转换的条件时，请使用以下准则：

- 查找条件中列的数据类型必须匹配。
- 必须在所有查找转换中输入查找条件。
- 将一个输入端口用于查找条件中的每个查找端口。在转换的多个条件中使用同一输入端口。
- 输入多个条件时，集成服务会将每个条件按照 AND（而不是 OR）进行计算。集成服务会返回与配置的所有条件匹配的行。
- 如果包含多个条件，则按照以下顺序输入条件以提高查找性能：
  - 等于 (=)
  - 小于 (<)、大于 (>)、小于或等于 (<=)、大于或等于 (>=)
  - 不等于 (!=)
- 集成服务会匹配空值。例如，如果某个输入查找条件列为空值，则集成服务会将该空值视为查找中的空值。
- 如果为已排序输入配置平面文件查找，则条件列未分组时，集成服务处理会话将会失败。如果列已分组但未排序，集成服务会像未配置已排序输入一样处理查找。

集成服务将根据是将转换配置为动态缓存、未缓存还是静态缓存，对查找匹配项进行不同的处理。

## 未缓存或静态缓存

当配置具有静态查找缓存或非缓存查找源的查找转换时，使用以下准则：

- 创建查找条件时，请使用以下运算符：

```
=, >, <, >=, <=, !=
```

如果您将多个查找条件包括在内，请按以下顺序放置条件以优化查找性能：

- 等于 (=)
- 小于 (<)、大于 (>)、小于或等于 (<=)、大于或等于 (>=)
- 不等于 (!=)

例如，创建以下查找条件：

```
ITEM_ID = IN_ITEM_ID
```

```
PRICE <= IN_PRICE
```

- 输入值必须满足所有查找条件，才能返回值。

该条件可以匹配等价值或提供阈值条件。例如，可以查找不居住在加利福尼亚州的客户或工资高于 30,000 美元的员工。根据源和条件的特性，查找可能会返回多个值。

## 动态缓存

如果将查找转换配置为使用动态缓存，可以在查找条件中仅使用等式运算符 (=)。

## 处理多项匹配

查找转换会根据您在转换中配置的条件查找值。如果查找条件不基于唯一的键，或者查找源非规范化，则集成服务可能在查找源或查找缓存中找到多个匹配项。

您可以将查找转换配置为通过以下方式处理多个匹配项：

- **使用第一个匹配值，或使用最后一个匹配值。**您可以将转换配置为返回第一个匹配值或最后一个匹配值。第一个值和最后一个值是在查找缓存中找到的与查找条件匹配的的第一个值和最后一个值。在缓存查找源时，集成服务会为查找缓存中的每个列生成 ORDER BY 子句，以确定缓存中第一个行和最后一个行。然后，集成服务会按升序排序每个查找源列。

集成服务按数字升序（例如 0 到 10）排序数字列。它会按从一月到十二月以及从当月第一天到最后一天的顺序排序日期/时间列。集成服务会根据为会话配置的排序顺序对字符串列排序。

- **使用任何匹配值。**您可以将该查找转换配置为返回匹配查找条件的任意值。将查找转换配置为返回任意匹配值时，该转换将返回匹配查找条件的第一个值。该转换会根据键端口而不是所有查找转换端口创建索引。使用任何匹配值时，会提高性能，因为对索引行的处理更简单。
- **使用所有值。**查找转换会返回所有匹配的行。要使用此选项，必须在创建查找转换时将其配置为返回所有匹配项。该转换会成为主动转换。创建转换后，不能在被动与主动模式之间进行更改。
- **返回错误。**查找转换使用静态缓存或没有缓存时，集成服务会将行标记为错误。默认情况下，查找转换将该行写入会话日志并将错误计数加一。如果查找转换具有动态缓存，集成服务会在遇到多个匹配项时处理会话失败。集成服务缓存查找表或查找重复的键值时，会话会失败。此外，如果将查找转换配置为在更新时输出旧值，查找转换会在遇到多个匹配项时返回错误。该转换会根据键端口而不是所有查找转换端口创建索引。

**相关主题：**

- [“返回多个行”页面上 271](#)

## 查找缓存

可以将查找转换配置为缓存查找文件或表。集成服务处理已缓存的查找转换中的第一行数据时，将在内存中构建缓存。该服务根据您在转换或会话属性中配置的内存量为缓存分配内存。集成服务将条件值存储在索引缓存中，将输出值存储在数据缓存中。集成服务在缓存中查询输入到转换中的每一行。

默认情况下，集成服务还会在 \$PMCacheDir 中创建缓存文件。如果内存缓存装不下数据，集成服务会将溢出值存储在缓存文件中。除非将查找转换配置为使用持久性缓存，否则在会话完成后，集成服务会释放缓存内存，并删除缓存文件。

在配置查找缓存时，可以配置以下选项：

- 持久性缓存
- 从查找源重新缓存
- 静态缓存
- 动态缓存
- 共享缓存
- 预构建查找缓存

**注意：**您可以为关系或平面文件查找使用动态缓存。

## 返回多个行

如果将查找转换配置为返回所有匹配行，则查找转换将返回与查找条件相匹配的所有行。在创建转换时，必须将该转换配置为返回所有匹配行。查找转换会成为主动转换。“多项匹配时的查找策略”属性为“使用所有值”。该属性将成为只读属性。在创建转换后，将无法更改该属性。

在一个关系表中可以有多个客户订单数据。在改表中，每个客户都可以有多份订单。可以将查找转换配置为通过查找为返回某一客户的所有订单。可以缓存查找表，以提高性能。

如果配置查找转换进行缓存，集成服务将缓存从查找源中读取的所有行。集成服务将按照键索引缓存查找键的所有行。

## 返回多个行的规则和准则

将查找转换配置为返回多行时，使用以下规则和准则：

- 集成服务可为缓存的查找缓存查找源的所有行。
- 可以为可返回多行的缓存或非缓存查找配置 SQL 替代。替代 SQL 必须返回按查找键排序的数据。如果数据未基于键排序，可能会获得异常结果。
- 无法为可返回多行的查找转换启用动态缓存。
- 无法从未连接的查找转换返回多行。
- 如果多个查找转换在多个匹配策略上具有匹配缓存查找，则可以将这些查找转换配置为共享指定的缓存。
- 可返回多行的查找转换无法与为每个输入行返回一个匹配行的查找转换共享缓存。

**相关主题：**

- [“创建查找转换”页面上 274](#)

## 配置未连接的查找转换

未连接的查找转换是指未连接到源或目标的查找转换。使用 :LKP 表达式从另一个转换调用查找。

您可以在通过表达式调用查找时执行以下任务：

- 在表达式中测试查找的结果。



- 根据查找结果筛选行。
- 根据查找结果标记要更新的行，并更新缓慢更改的维度表。
- 在一个映射中多次调用相同的查找。

要配置未连接的查找转换，请添加输入端口，配置查找条件，指定返回值，并在不同的转换中配置查找表达式。

## 步骤 1. 添加输入端口

在查找转换中，为 :LKP 表达式中的每个参数创建一个输入端口。对于您计划创建的每项查找条件，需要将输入端口添加到查找转换中。可以为每项条件创建一个不同的端口，或者在多项条件中使用同一输入端口。

例如，一家零售店在上个月提高了所有部门的价格。会计部门只想将提价商品的行加载到目标中。要实现这一目的，请完成以下任务：

- 创建一项查找条件，对源中的 ITEM\_ID 与目标中的 ITEM\_ID 进行比较。
- 对源中每件商品的价格与目标表中的价格进行比较。
  - 如果该商品存在于目标表中，并且源中的商品价格低于或等于目标表中的价格，您会想要删除该行。
  - 如果源中的价格高于目标表中的商品价格，您会想要更新该行。
- 创建一个数据类型为 Decimal (37,0) 的输入端口 (IN\_ITEM\_ID)，以匹配 ITEM\_ID；以及一个数据类型为 Decimal (10,2) 的 IN\_PRICE 输入端口，以匹配 PRICE 查找端口。

## 步骤 2. 添加查找条件

在配置端口之后，请定义查找条件，以便将转换输入值与查找源或缓存中的值进行比较。要提高性能，首先请使用等号添加条件。

在这种情况下，添加以下查找条件：

```
ITEM_ID = IN_ITEM_ID
PRICE <= IN_PRICE
```

如果项存在于映射源和查找源中，并且映射源价格低于或等于查找价格，则条件为 true，查找将返回由返回端口指定的值。如果查找条件为 false，查找将返回空。在编写更新策略表达式时，使用潜逃在 IIF 函数中的 ISNULL 来测试空值。

## 步骤 3. 指定返回值

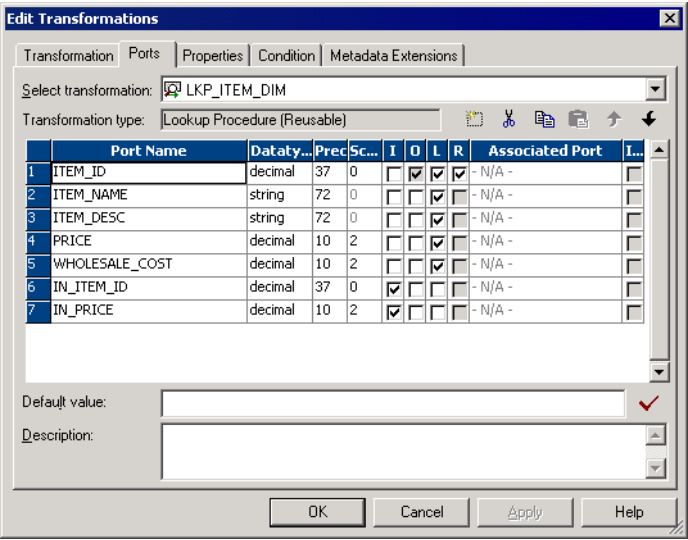
可以将多个输入值传递到一个查找转换并返回一系列数据。请勿将查找策略配置为对未连接的查找转换使用多个匹配项的所有值。指定一个查找/输出端口作为返回端口。可以选择使用与查找条件匹配的最后一个值、第一个值或任意值。

如果从更新策略或筛选器表达式调用未连接的查找，通常要检查是否有空值。在这种情况下，返回端口可以是任意端口。如果从执行计算的表达式调用查找，则返回值需要是您要在计算中包括的值。

要继续更新策略示例，可以将 ITEM\_ID 端口定义为返回端口。更新策略表达式会检查是否有空值返回。如果查找条件为 True，则集成服务会返回 ITEM\_ID。如果条件为 False，则集成服务将返回空。



下图显示了查找转换中的返回端口：



## 步骤 4。通过表达式调用查找

通过其他转换中的 :LKP 表达式为未连接的查找转换提供输入值。参数是与查找条件中使用的查找转换输入端口相匹配的本地数据端口。可将以下语法用于 :LKP 表达式：

`:LKP.lookup_transformation_name(argument, argument, ...)`

要继续进行关于零售店的示例，在编写更新策略表达式时，该表达式中的端口顺序必须与查找条件中的顺序相匹配。在这种情况下，ITEM\_ID 条件是第一项查找条件，因此，它是更新策略表达式中的第一个参数。

`IIF(ISNULL(:LKP.lkpITEMS_DIM(ITEM_ID, PRICE)), DD_UPDATE, DD_REJECT)`

可以使用以下准则编写用于调用未连接查找转换的表达式：

- 列出每个参数的顺序必须与查找转换中查找条件的顺序一致。
- 该表达式中端口的数据类型必须与查找转换中输入端口的数据类型相匹配。如果数据类型不匹配，则 Designer 不会验证该表达式。
- 如果查找条件中的一个端口不是查找/输出端口，则 Designer 不会验证该表达式。
- 该表达式中的参数端口的顺序必须与查找条件中输入端口的顺序相同。
- 如果使用不正确的 :LKP 语法，则 Designer 会将映射标记为无效。
- 如果在 :LKP 表达式中调用已连接的查找转换，则 Designer 会将映射标记为无效。

**提示：**在使用点击式方法选择函数和端口来输入表达式时，应该避免语法错误。

## 数据库死锁弹性

查找转换可恢复未缓存查找的数据库死锁。出现数据库死锁错误时，会话不会失败。集成服务会在指定的重试期限内尝试重新执行最后一个语句。

您可以为集成服务配置死锁重试数和死锁休眠间隔。这些值还影响关系写入器的数据库死锁。您可以在会话级别将这些值替代为自定义属性。

配置以下集成服务属性：

- **NumOfDeadlockRetries**.PowerCenter 集成服务对数据库死锁重试目标写入的次数。最小值为 0。默认值为 10。如果希望会话在死锁时失败，请将 NumOfDeadlockRetries 设置为零。
- **DeadlockSleep**。遇到数据库死锁时 PowerCenter 集成服务在重试目标写入前等待的秒数。

如果出现死锁，集成服务会尝试运行该语句。集成服务会在每次重试尝试之间等待延迟时限。如果所有尝试因死锁失败，则会话会失败。集成服务重试某条语句时，都会在会话日志中记录一条消息。

## 创建查找转换

在 Transformation Developer 中创建可重用的查找转换。在 Mapping Designer 中创建不可重用的查找转换。

要创建查找转换，请执行以下操作：

1. 要创建可重用的查找转换，请打开 Transformation Developer。  
要创建不可重用的查找转换，请在 Mapping Designer 中打开映射。如果要创建管道查找转换，请拖动一个源定义中以用作查找源。
2. 单击“转换”>“创建”。选择查找转换。
3. 输入转换的名称。单击“创建”。  
查找转换的命名约定为 LKP\_ *TransformationName*。
4. 选择转换是主动还是被动。单击“确定”。不可以更改此选项。
5. 在“选择查找表”对话框中，选择以下选项之一来导入查找定义：
  - 存储库中的源定义。
  - 存储库中的目标定义。
  - 映射中的源限定符。
  - 从存储库中导入关系表或文件。

**注意：**可以手动添加查找端口而不是导入定义。可以选择哪些查找端口也是输出端口。

选择查找源时，Designer 会在转换中根据所选对象中的端口创建端口。Designer 会将各个端口配置为查找端口和输出端口。查找端口表示查找源中的列。查找转换会在各个查找端口中收到查找源中的数据，并将数据传递到目标。

6. 如果希望查找转换返回所有匹配的行，请启用“多项匹配时返回所有行”。创建转换后不能更改此选项。查找转换会成为主动转换。
7. 单击“确定”，或者如果要手动添加查找端口而不是导入定义，请单击“跳过”。可以选择哪些查找端口也是输出端口。
8. 对于连接的查找转换，添加输入和输出端口。  
可以通过转换传递数据，并将查找表中的数据返回到目标。
9. 对于未连接的查找转换，创建返回端口以获得要从查找中返回的值。  
可以将一个列返回到称为查找的转换。
10. 单击“属性”选项卡以配置查找转换属性。配置查找缓存。  
默认情况下，为管道和平面文件查找转换启用了查找缓存。
11. 对于具有动态查找缓存的查找转换，将输入端口、输出端口或序列 ID 与各个查找端口相关联。

集成服务会通过各个关联表达式中的数据在查找缓存中插入或更新行。如果您关联序列 ID，集成服务会为查找缓存中插入的行生成一个主键。

12. 在“条件”选项卡上添加查找条件。

查找条件会将源列值与查找源中的值进行比较。“条件”选项卡上的转换端口表示源列值。查找表表示查找源。

#### 相关主题：

- [“创建可重用管道查找转换” 页面上 275](#)
- [“创建不可重用管道查找转换” 页面上 275](#)

## 创建可重用管道查找转换

在 Transformation Developer 中创建可重用的管道查找转换。创建转换时，为查找表位置选择源。Transformation Developer 会显示存储库中源定义的列表。

选择表示关系表或平面文件源定义的源限定符，Designer 会创建关系或平面文件查找转换。源限定符表示应用程序源时，Designer 会创建管道查找转换。要为关系或平面文件查找源创建管道查找转换，请在创建转换后将源类型更改为源限定符。在查找表属性中输入源定义的名称。

将可重用的管道查找转换拖动到映射中时，Mapping Designer 会将查找表属性中的源定义添加到映射。Designer 会添加源限定符转换。

要在映射中将查找表名称更改为另一个源限定符，请在“查找表名称”属性中单击“打开”按钮。从列表中选择源限定符。

## 创建不可重用管道查找转换

在 Mapping Designer 中创建不可重用的管道查找转换。将源定义拖动到映射中。在 Mapping Designer 中创建转换时，选择源限定符作为查找表位置。Mapping Designer 会在映射中显示源限定符的列表。选择源限定符时，Mapping Designer 会使用所选源限定符中的端口名称和属性填充查找转换。

## 查找转换提示

将索引添加到查找条件中使用的列。

如果您有修改包含查找表的数据库的特权，则可以提高缓存和非缓存查找的性能。这对非常大的查找表很重要。由于集成服务需要查询、排序并比较这些列中的值，因此索引需要包括查找条件中使用的所有列。

请先放置使用等式运算符 (=) 的条件。

如果您将多个查找条件包括在内，请按以下顺序放置条件以优化查找性能：

- 等于 (=)
- 小于 (<)、大于 (>)、小于或等于 (<=)、大于或等于 (>=)
- 不等于 (!=)

缓存小型查找表。

通过缓存小型查找表来提高会话性能。无论是否缓存查找表，查找查询和处理的结果都相同。

联接数据库中的表。

如果查找表与映射中的源表位于相同的数据库中且缓存不可行，请联接源数据库中的表，而不是使用查找转换。

将持久性查找缓存用于静态查找。

如果查找源在会话之间未更改，则可以将查找转换配置为使用持久性查找缓存。然后，集成服务将在逐个会话中保存并重用缓存文件，以消除读取查找源所需的时间。

使用 :LKP 引用限定符调用未连接的查找转换。

当使用 :LKP 引用限定符调写入表达式时，只调用未连接的查找转换。如果尝试调用已连接的查找转换，Designer 会显示错误并将映射标记为无效。

在处理关系或平面文件查找源时，配置管道查找转换以提高性能。

在定义查找源作为源限定符时，可以创建分区来处理关系或平面文件查找源。在处理查找源的部分管道中配置不可重用管道查找转换并创建分区。

## 第 18 章

# 查找缓存

本章包括以下主题：

- [查找缓存概览, 277](#)
- [构建连接的查找缓存, 279](#)
- [使用持久性查找缓存, 280](#)
- [使用未缓存的查找或静态缓存, 281](#)
- [共享查找缓存, 281](#)
- [查找缓存的提示, 286](#)

## 查找缓存概览

可以将查找转换配置为缓存查找源，以提高查找性能。查找表或文件较大时可启用查找缓存。

集成服务处理已缓存的查找转换中的第一行数据时，将在内存中构建缓存。集成服务在源行输入到查找转换时创建缓存。该服务根据您在转换或会话属性中配置的内存量为缓存分配内存。集成服务将条件值存储在索引缓存中，将输出值存储在数据缓存中。集成服务在缓存中查询输入到转换中的每一行。

如果内存缓存装不下数据，集成服务会将溢出值存储在缓存文件中。集成服务还会在指定缓存目录中创建缓存文件。除非将查找转换配置为使用持久性缓存，否则在会话映射完成后，集成服务会释放缓存内存，并删除缓存文件。

如果使用平面文件查找或管道查找，集成服务始终会对查找源进行缓存。如果为已排序输入配置平面文件查找，则当条件列未分组时，集成服务将无法缓存该查找。如果列已分组但未排序，集成服务会像未配置已排序输入一样处理查找。

配置查找缓存时，可以配置以下缓存设置：

### 连续和并发缓存

可以将会话配置为连续或并发构建缓存。构建连续缓存时，集成服务将在源行输入到查找转换时创建缓存。将会话配置为构建并发缓存时，集成服务在等待第一行输入到查找转换前不会创建缓存。而是并发构建多个缓存。

### 持久性缓存

可以保存查找缓存文件，并在集成服务下次处理配置为使用缓存的查找转换时重新使用这些文件。

### 从源重新缓存

如果持久性缓存没有与查找源同步，可以将查找转换配置为重建查找缓存。

可以将查找转换配置为重建持久性查找缓存。

静态缓存

可以为任何查找源配置静态缓存。默认情况下，集成服务将创建静态缓存。它会缓存查找文件或表，并在缓存中查找进入转换的每一行的值。如果查找条件为 true，集成服务将从查找缓存返回一个值。集成服务在处理查找转换时不会更新缓存。

动态缓存

要缓存查找源和更新缓存，请使用动态缓存配置查找转换。集成服务将动态插入或更新查找缓存中的数据，并将数据传递到目标。动态缓存与目标同步。

共享缓存

可以在多个转换之间共享查找缓存。可以在相同映射中的转换之间共享未命名缓存。可以在相同或不同映射中的转换之间共享命名缓存。如果缓存共享规则匹配，查找转换可在同一目标加载顺序组内共享未命名静态缓存。查找转换不能在同一目标加载顺序组内共享动态缓存。

可以在相同映射中的多个转换之间共享查找缓存。

如果不对查找转换进行配置以实现缓存，集成服务将在查找源中查询每个输入行。无论是否缓存查找源，查找查询的结果和处理方式都相同。但如果启用查找缓存，可以提高大查找源的查找性能。

缓存比较

集成服务根据所配置的查找缓存的类型执行不同的操作。

下表将查找转换与未缓存查找、静态缓存和动态缓存进行了比较：

未缓存	静态缓存	动态缓存
集成服务不在缓存中插入或更新缓存。	集成服务不在缓存中插入或更新缓存。	集成服务可以在向目标传递行时在缓存中插入或更新行。
可以使用关系查找。	可以使用关系查找、平面文件查找或管道查找。 可以使用关系查找或平面文件查找。	可以使用关系查找、平面文件查找或源限定符查找。 可以使用关系查找或平面文件查找。
如果条件为 true，集成服务将从查找表或缓存返回一个值。 如果条件不为 true，集成服务将为连接的转换返回默认值，为未连接的转换返回空值。	如果条件为 true，集成服务将从查找表或缓存返回一个值。 如果条件不为 true，集成服务将为连接的转换返回默认值，为未连接的转换返回空值。	如果条件为 true，集成服务将更新缓存中的行，或者保持缓存不变，具体视行类型而定。这表明行在缓存和目标表中。可以将更新的行传递到目标。 如果条件不为 true，集成服务会在缓存中插入行，或者保持缓存不变，具体视行类型而定。这表明行不在缓存或目标中。可以将插入的行传递到目标表。

相关主题：

- “使用未缓存的查找或静态缓存” 页面上 281
- “动态查找缓存更新” 页面上 294

# 构建连接的查找缓存

集成服务可以通过以下方式为连接的查找转换构建查找缓存：

- **连续缓存。**集成服务连续构建查找缓存。集成服务处理已缓存的查找转换中的第一行数据时，将在内存中构建缓存。
- **并发缓存。**集成服务并发构建查找缓存。它不需等待数据到达查找转换。

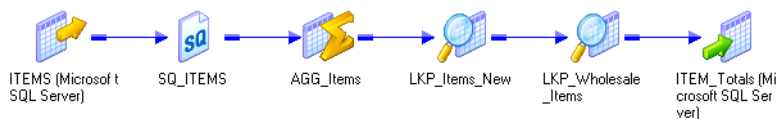
**注意：**集成服务为未连接的查找转换顺序构建缓存，不管配置缓存构建的方式如何都是如此。如果配置会话为未连接的查找转换构建并发缓存，则集成服务会忽略该设置并顺序构建未连接的查找转换缓存。

## 顺序缓存

默认情况下，当集成服务处理缓存的查找转换中的第一行数据时，它将在内存中构建缓存。集成服务将按顺序在管道中创建每个查找缓存。集成服务将等待任何上游活动转换完成处理，然后再开始处理查找转换中的行。集成服务不会为下游查找转换构建缓存，直到上游查找转换完成构建缓存为止。

例如，以下映射包含一个未排序汇总器转换，后跟两个查找转换：

图 2. 按顺序构建查找缓存



集成服务将处理未排序汇总器转换的所有行，在未排序汇总器转换完成后，再开始处理第一个查找转换。在集成服务处理第一个输入行时，集成服务将开始构建第一个查找缓存。在集成服务完成第一个查找缓存的构建之后，即可开始处理查找数据。当第一行数据到达查找转换时，集成服务将开始构建下一个查找缓存。

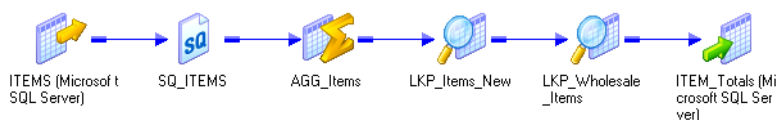
如果查找转换可能无法处理行数据，您可能想要按顺序处理查找缓存。如果将转换逻辑配置为根据条件将数据路由到不同的管道，则查找转换可能无法处理行数据。通过配置顺序缓存，可以避免构建不必要的查找缓存。例如，如果某一条件的解析结果为 True，则路由器转换可将数据路由到一个管道；如果该条件的解析结果为 False，则它会将数据路由到其他管道。在这种情况下，查找转换可能根本无法接收数据。

## 并发缓存

您可以配置集成服务以并发创建查找缓存。您可使用并发缓存提高会话性能。管道包含查找转换的活动转换上游时，性能尤其会提升。如果您确定需要为会话中的每个查找转换构建缓存，则可能要配置会话以创建并发缓存。

在配置查找转换以创建并发缓存时，它不会等待上游转换完成才创建查找缓存，而且不需要完成构建查找缓存的过程即可开始构建其他查找缓存。

下图显示了并发构建的查找转换缓存：



在运行会话时，集成服务会并发构建查找缓存。它不会等待上游转换完成，而且不会等待其他查找转换完成缓存构建。

**注意：**您不能为未连接的查找转换并发处理缓存。

要配置会话以创建并发缓存，请为会话配置属性“用于查找缓存创建的其他并发管道”配置一个值。

# 使用持久性查找缓存

可以将查找转换配置为使用非持久性或持续性缓存。集成服务会根据“查找缓存持久性”属性在会话成功后保存或删除查找缓存文件。

如果查找表在会话之间未更改，则可以将查找转换配置为使用持久性查找缓存。集成服务将在逐个会话中保存并重用缓存文件，以消除读取查找表所需的时间。

## 使用非持久性缓存

默认情况下，在查找转换中启用缓存时，集成服务将使用非持久性缓存。集成服务在会话结束时删除缓存文件。下次运行会话时，集成服务将从数据库构建内存缓存。

## 使用持久性缓存

如果要保存并重用缓存文件，可以将转换配置为使用持久性缓存。如果您知道查找表在会话运行之间保持不变，请使用持久性缓存。

集成服务第一次使用持久性查找缓存运行会话时，会将缓存文件保存到磁盘中而不是删除这些文件。下次运行会话时，集成服务将通过这些缓存文件构建内存缓存。如果查找表偶尔更改，可以替代会话属性以便重新从数据库缓存查找。

使用持久性查找缓存时，可以指定缓存文件的名称。指定命名缓存时，可以在会话之间共享查找缓存。

## 重建查找缓存

如果您认为查找源自上次集成服务构建持久性缓存后发生变化，可以指示集成服务重建查找缓存。

重建缓存时，集成服务将创建新的缓存文件，覆盖现有的持久性缓存文件。重建缓存后，集成服务会将消息写入到会话日志中。

如果映射包含一个查找转换或映射包含共享缓存的多个目标加载顺序组中的查找转换，可以重建缓存。如果是同一个映射中的动态查找与静态查找共享缓存，则无需重建缓存。

如果集成服务无法重用缓存，则它将重新缓存数据库中的查找或者会话失败，具体因映射和会话属性而异。

下表总结了集成服务处理命名和未命名缓存的持久性缓存的方式：

会话之间的映射或会话更改	命名缓存	未命名缓存
集成服务无法找到缓存文件。例如，该文件已不存在或集成服务在网格中运行且缓存文件不能用于所有节点。	重建缓存。	重建缓存。
启用或禁用会话属性中的“启用高精度”选项。	会话失败。	重建缓存。



会话之间的映射或会话更改	命名缓存	未命名缓存
在 Mapping Designer、Mapplet Designer 或可重用 Transformation Developer 中编辑转换。 <sup>1</sup>	会话失败。	重建缓存。
编辑映射（不含查找转换）。	重用缓存。	重建缓存。
更改包含查找转换的管道中的分区数。	会话失败。	重建缓存。
更改用于访问查找表的数据库连接或文件位置。	会话失败。	重建缓存。
更改集成服务数据移动模式。	会话失败。	重建缓存。
在 Unicode 模式下更改排序顺序。	会话失败。	重建缓存。
将集成服务代码页更改为兼容的代码页。	重用缓存。	重用缓存。
将集成服务代码页更改为不兼容的代码页。	会话失败。	重建缓存。

<sup>1</sup> 编辑转换说明或端口说明等属性不会影响持久性缓存处理。

# 使用未缓存的查找或静态缓存

默认情况下，集成服务将在您为实现缓存而配置查找转换时创建静态查找缓存。集成服务在处理第一个查找请求时构建缓存。该服务根据传递到转换中的每一行的查找条件查询缓存。集成服务不会在处理转换时更新缓存。集成服务处理未缓存查找的方式与处理缓存查找的方式相同，但不同之处在于其查询查找源，而不是构建和查询缓存。

如果查找条件为 true，集成服务将从查找源或缓存返回值。对于已连接的查找转换，集成服务返回由查找/输出端口表示的值。对于未连接的查找转换，集成服务将返回由返回端口表示的值。

如果条件不为 true，集成服务将返回空值或默认值。对于已连接的查找转换，如果不满足条件，集成服务将返回输出端口的默认值。对于未连接的查找转换，如果不满足条件，集成服务将返回空。

在使用静态缓存的管道中创建多个分区时，集成服务将为每个分区创建一块内存缓存，为每个转换创建一块磁盘缓存。

# 共享查找缓存

可以将映射中的多个查找缓存配置为共享一个查找缓存。集成服务在处理第一个查找转换时构建缓存。它将使用同一个缓存为共享该缓存的后续查找转换执行查找。

可以共享未命名和已命名的缓存：

- **未命名缓存。**当映射中的查找转换具有兼容的缓存结构时，默认情况下，集成服务将共享缓存。只能共享静态未命名缓存。
- **命名缓存。**如果想要在多个映射之间共享一个缓存文件，或者共享动态和静态缓存，则可使用持久性命名缓存。缓存结构必须匹配，或与命名缓存兼容。可以共享静态和动态命名缓存。

当集成服务共享查找缓存时，它会在会话日志中写入一条消息。

# 共享未命名查找缓存

默认情况下，对于映射中缓存结构兼容的查找转换，集成服务会共享缓存。例如，如果在一个映射中有相同的可重用查找转换的两个实例，并且对这两个实例使用相同的输出端口，则默认情况下查找转换将共享查找缓存。

当两个查找转换共享未指定的缓存时，集成服务会为查找转换保存缓存并将其用于具有相同查找缓存结构的后续查找转换。

如果转换属性或缓存结构不允许共享，集成服务将创建新的缓存。

## 共享未命名查找缓存的准则

将查找转换配置为共享未命名缓存时，请遵循以下准则：

- 可以共享静态未命名缓存。
- 共享转换在查找条件中必须使用相同的端口。这些条件可以使用不同的运算符，但端口必须相同。
- 您必须配置一些转换属性，才能启用未命名缓存共享。
- 共享转换的缓存结构必须兼容。
  - 如果您使用哈希自动键分区，则每个转换的查找/输出端口必须匹配。
  - 如果不使用哈希自动键分区，则第一个共享转换的查找/输出端口必须匹配，或者是后续转换的查找/输出端口的子集。
- 如果使用哈希自动键分区的查找转换位于不同的目标加载顺序组中，则必须为每个组配置相同数量的分区。如果不使用哈希自动键分区，可以为每个目标加载顺序组配置不同数量的分区。

下表显示了可共享未命名静态和动态缓存的情况：

共享缓存	转换的位置
静态与静态	映射中的任意位置。
动态与动态	无法共享。
动态与静态	无法共享。

下表介绍了将查找转换配置为共享未命名缓存时应遵循的准则：

属性	未命名共享缓存的配置
查找 SQL 替代	如果使用“查找 SQL 替代”属性，则必须在所有共享转换中使用相同的替代。
查找表名称	必须匹配。
启用查找缓存	必须启用。
多项匹配时的查找策略	-
查找条件	共享转换在查找条件中必须使用相同的端口。这些条件可以使用不同的运算符，但端口必须相同。
连接信息	连接必须相同。在配置会话时，数据库连接必须匹配。
源类型	必须匹配。

属性	未命名共享缓存的配置
跟踪级别	-
查找缓存目录名称	无需匹配。
查找缓存持久性	可选。可以共享持久性和非持久性。
查找数据缓存大小	集成服务为每个管道阶段的第一个共享转换分配内存。其不为同一管道阶段的后续共享转换分配额外内存。
查找索引缓存大小	集成服务为每个管道阶段的第一个共享转换分配内存。其不为同一管道阶段的后续共享转换分配额外内存。
动态查找缓存	无法共享未命名动态缓存。
更新时输出旧值	无需匹配。
缓存文件名前缀	请勿使用。无法共享命名缓存与未命名缓存。
从查找源重新缓存	<p>如果将查找转换配置为从源重新缓存，则目标加载顺序组中的后续查找转换可共享现有缓存，无论您是否将它们配置为从源重新缓存都是如此。如果将后续查找转换配置为从源重新缓存，则当集成服务处理后续查找转换时，将共享缓存而不会重建缓存。</p> <p>如果没有将目标加载顺序组中的第一个查找转换配置为从源重新缓存，但确实将后续查找转换配置为从源重新缓存，则这些转换无法共享缓存。集成服务在处理每个查找转换时构建缓存。</p>
查找/输出端口	第二个查找转换的查找/输出端口必须匹配，或者是集成服务用来构建缓存的转换中的端口的子集。端口的顺序无需匹配。
插入 Else 更新	-
更新 Else 插入	-
日期时间格式	-
千位分隔符	-
小数分隔符	-
区分大小写的字符串比较	必须匹配。
空值排序	必须匹配。
已排序输入	-

## 共享命名的查找缓存

还可以通过使用持久性查找缓存并命名缓存文件在多个查找转换之间共享缓存。可以在同一映射中或跨映射在查找转换之间共享一个缓存。

集成服务使用以下流程共享指定查找缓存：

1. 当集成服务处理第一个查找转换时，会搜索具有相同文件名前缀的缓存文件的缓存目录。
2. 如果集成服务查找这些缓存文件并且您未指定从源重新缓存，则集成服务会使用保存的缓存文件。

3. 如果集成服务不查找这些缓存文件或者您指定从源重新缓存，则集成服务会使用数据库表构建查找缓存。
4. 集成服务在处理每个目标加载顺序组后，会将缓存文件保存到磁盘。
5. 集成服务使用以下规则处理第二个具有相同缓存文件名前缀的查找转换：
  - 如果转换位于同一目标加载顺序组中，则集成服务会使用内存缓存。
  - 如果转换位于不同的目标加载顺序组中，则集成服务会通过持久性文件重新构建内存缓存。
  - 如果将转换配置为从源重新缓存并且第一个转换位于不同的目标加载顺序组中，则集成服务会从数据库重新构建缓存。
  - 如果没有将目标加载顺序组中的第一个查找转换配置为从源重新缓存，但确实将后续查找转换配置为从源重新缓存，则集成服务不会重建缓存。
  - 如果缓存结构不匹配，则集成服务处理该会话会失败。

如果同时运行共享查找缓存的两个会话，则集成服务会使用以下规则来共享缓存文件：

- 当查找转换只需要读取缓存文件时，集成服务会同时处理多个会话。
- 如果一个会话更新某个缓存文件，而另一个会话尝试读取或更新该缓存文件，则集成服务处理该会话会失败。例如，查找转换在配置为使用动态缓存或从源重新缓存时更新缓存文件。

## 共享命名查找缓存的准则

将查找转换配置为共享命名缓存时，请遵循以下准则：

- 如果缓存为动态查找缓存或转换配置为从源重新缓存，请勿在会话之间共享查找缓存。
- 可以在动态与静态查找转换之间共享缓存，但必须遵循该缓存位置的准则。
- 您必须配置一些转换属性，才能启用命名缓存共享。
- 如果命名缓存包含重复行，则动态查找不能共享缓存。
- 使用任何查找策略的静态或动态查找转换可以共享查找策略为多项匹配时错误的动态查找转换创建的命名缓存。
- 查找策略相同的查找转换可以共享查找策略为使用第一个值、使用最后一个值或使用所有值的动态查找转换创建的命名缓存。
- 共享转换在映射中必须使用相同的端口。缓存的标准和结果列必须与缓存文件匹配。
- 一个动态查找转换不能与同一目标加载顺序组内的另一个查找转换共享缓存。在目标加载顺序组中，将并行处理缓存并在目标加载后完成。PowerCenter 集成服务不允许共享缓存，因为后续查找转换无法使用完整的缓存。

集成服务可能使用内存缓存，也可能从文件构建内存缓存，具体因查找转换的类型和位置而异。

下表显示了可以在静态查找转换和动态查找转换之间共享命名缓存的情况：

表 3. 共享命名缓存的位置

共享缓存	转换的位置	共享的缓存
静态与静态	<ul style="list-style-type: none"><li>- 同一个目标加载顺序组。</li><li>- 独立的目标加载顺序组。</li><li>- 独立映射。</li></ul>	<ul style="list-style-type: none"><li>- 集成服务使用内存缓存。</li><li>- 集成服务使用内存缓存。</li><li>- 集成服务从文件构建内存缓存。</li></ul>
动态与动态	<ul style="list-style-type: none"><li>- 独立的目标加载顺序组。</li><li>- 独立映射。</li></ul>	<ul style="list-style-type: none"><li>- 集成服务使用内存缓存。</li><li>- 集成服务从文件构建内存缓存。</li></ul>
动态与静态	<ul style="list-style-type: none"><li>- 独立的目标加载顺序组。</li><li>- 独立映射。</li></ul>	<ul style="list-style-type: none"><li>- 集成服务从文件构建内存缓存。</li><li>- 集成服务从文件构建内存缓存。</li></ul>

下表介绍了将查找转换配置为共享命名缓存时应遵循的准则：

表 4. 共享命名缓存的属性

属性	命名共享缓存的配置
查找 SQL 替代	如果使用“查找 SQL 替代”属性，则必须在所有共享转换中使用相同的替代。
查找表名称	必须匹配。
启用查找缓存	必须启用。
多项匹配时的查找策略	<ul style="list-style-type: none"><li>- 使用任何查找策略的静态或动态查找转换可以共享查找策略为多项匹配时错误的动态查找转换创建的命名缓存。</li><li>- 查找策略相同的查找转换可以共享查找策略为使用第一个值或使用最后一个值的动态查找转换创建的命名缓存。</li><li>- 如果一个查找策略为使用所有值的动态查找转换要与查找策略相同的另一个活动查找转换共享缓存，则该动态查找转换可以共享命名缓存。</li></ul>
查找条件	共享转换在查找条件中必须使用相同的端口。这些条件可以使用不同的运算符，但端口必须相同。
连接信息	连接必须相同。在配置会话时，数据库连接必须匹配。
源类型	必须匹配。
跟踪级别	-
查找缓存目录名称	必须匹配。
查找缓存持久性	必须启用。
查找数据缓存大小	同一个映射内的转换共享缓存时，集成服务将为每个管道阶段的第一个共享转换分配内存。其不为同一管道阶段的后续共享转换分配额外内存。
查找索引缓存大小	同一个映射内的转换共享缓存时，集成服务将为每个管道阶段的第一个共享转换分配内存。其不为同一管道阶段的后续共享转换分配额外内存。

属性	命名共享缓存的配置
动态查找缓存	有关共享静态和动态缓存的详细信息，请参阅 <a href="#">“共享命名查找缓存的准则” 页面上 284。</a>
更新时输出旧值	无需匹配。
更新动态缓存	无需匹配。
缓存文件名前缀	必须匹配。仅输入前缀。不要输入 .idx 或 .dat。无法共享命名缓存与未命名缓存。
从源重新缓存	<p>如果将查找转换配置为从源重新缓存，则目标加载顺序组中的后续查找转换可共享现有缓存，无论您是否将它们配置为从源重新缓存都是如此。如果将后续查找转换配置为从源重新缓存，则当集成服务处理后续查找转换时，将共享缓存而不会重建缓存。</p> <p>如果没有将目标加载顺序组中的第一个查找转换配置为从源重新缓存，但确实将后续查找转换配置为从源重新缓存，则集成服务不会重建缓存。</p>
查找/输出端口	查找/输出端口必须相同，但它们的顺序无需相同。
插入 Else 更新	-
更新 Else 插入	-
千位分隔符	-
小数分隔符	-
区分大小写的字符串比较	-
空值排序	-
已排序输入	必须匹配。

**注意:** 不能共享在不同操作系统中创建的查找缓存。例如，只有 UNIX 上的集成服务才能读取集成服务在 UNIX 上创建的查找缓存，并且只有 Windows 上的集成服务才能读取集成服务在 Windows 上创建的查找缓存。

# 查找缓存的提示

缓存小型查找表。

通过缓存小型查找表提高性能。

将持久性查找缓存用于静态查找表。

如果查找表在会话之间未更改，则可以将查找转换配置为使用持久性查找缓存。集成服务将保存并重用缓存文件，以消除读取查找表所需的时间。

如果查找表在映射运行之间未更改，则可以将查找转换配置为使用持久性查找缓存。集成服务将保存并重用缓存文件，以消除读取查找表所需的时间。

## 第 19 章

# 动态查找缓存

本章包括以下主题：

- [动态查找缓存概览, 287](#)
- [动态查找属性, 288](#)
- [查找转换值, 291](#)
- [动态查找缓存更新, 294](#)
- [具有动态查找的映射, 295](#)
- [条件动态缓存更新, 297](#)
- [表达式结果的动态缓存更新, 298](#)
- [同步缓存与查找源, 299](#)
- [动态查找缓存示例, 300](#)
- [动态查找缓存的规则和准则, 300](#)

## 动态查找缓存概览

利用动态查找缓存可以使缓存与目标保持同步。可以将动态缓存与关系查找、平面文件查找或管道查找共用。可以将动态缓存与关系查找或平面文件查找共用。

集成服务在处理第一个查找请求时构建动态查找缓存。该服务根据传递到转换中的每一行的查找条件查询缓存。集成服务在处理每一行时更新查找缓存。

集成服务根据查找查询的结果、行类型以及查找转换属性，在从源读取行时对动态查找缓存执行以下操作之一：  
在缓存中插入行

如果行不在缓存中，并且已将查找转换配置为在缓存中插入行，则集成服务会插入行。可以将转换配置为根据输入端口或生成的序列 ID 在缓存中插入行。集成服务将行标记为插入。

更新缓存中的行

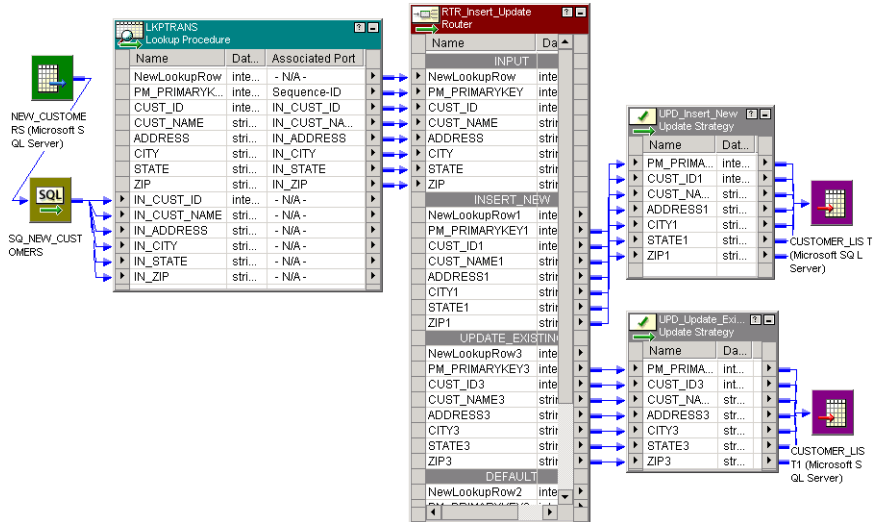
如果行存在于缓存中，并且已将查找转换配置为更新缓存中的行，则集成服务会更新行。集成服务根据输入端口更新缓存中的行。集成服务将行标记为更新行。

不对缓存进行更改

在以下情况下，集成服务不会对缓存进行更改：如果行存在于缓存中，并且已将查找转换配置为仅插入新行。或者，行不存在于缓存中并且指定了仅更新现有行。或者，行存在于缓存中，但根据查找条件，未进行任何更改。集成服务将行标记为未更改。

根据 NewLookupRow 的值，还可以配置路由器或筛选器转换的动态查找转换，以将插入行或更新行路由到目标表。可以将未更改的行路由到其他目标表或平面文件，也可以删除这些行。

下图显示的映射中包含一个使用动态查找缓存的查找转换：



# 动态查找属性

使用动态缓存的查找转换具有以下属性：

- **NewLookupRow.Designer** 会将此端口添加至配置为使用动态缓存的查找转换。通过一个数值指示集成服务是在缓存中插入或更新行，还是不对缓存做任何更改。为使查找缓存与目标表保持同步，可在 NewLookupRow 值等于 1 或 2 时将行传递到目标。
- **关联表达式**。将查找端口或关联端口与表达式、输入/输出端口或序列 ID 关联。集成服务使用关联表达式中的数据在查找缓存中插入或更新行。如果您关联序列 ID，集成服务会为查找缓存中插入的行生成一个主键。
- **忽略更新的空值输入**。将查找转换配置为使用动态缓存时，Designer 会为查找/输出端口激活此端口属性。如果不希望集成服务在列中的数据包含空值时更新缓存中的列，则选择该属性。
- **在比较中忽略**。将查找转换配置为使用动态缓存时，Designer 会为查找条件中未使用的查找/输出端口激活此端口属性。默认情况下，集成服务将所有查找端口中的值与其关联输入端口中的值相比较。如果希望集成服务在更新行之前比较值时忽略端口，则选择该属性。
- **更新动态缓存条件**。允许集成服务有条件地更新动态缓存。您可以创建布尔表达式，用于确定是否更新输入行的缓存。或者，您可以让集成服务使用输入行的表达式结果更新缓存。该表达式可包含来自输入行或查找缓存的值。

## NewLookupRows

将查找转换配置为使用动态缓存时，Designer 会将 NewLookupRow 端口添加至该转换。集成服务根据其对该查找缓存执行的操作向该端口分配一个值。



下表列出了 NewLookupRow 的可能值：

NewLookupRow 值	说明
0	集成服务不在缓存中更新或插入行。
1	集成服务在缓存中插入行。
2	集成服务更新缓存中的行。

当集成服务读取行时，其会根据查找查询的结果和您定义的查找转换属性更改查找缓存。它将为 NewLookupRow 端口分配值 0、1 或 2 来指示是在缓存中插入或更新行，还是不做更改。

该 NewLookupRow 值指示集成服务如何更改查找缓存。它不会更改行类型。因此，使用筛选器或路由器转换和更新策略转换可使目标表与查找缓存保持同步。

配置筛选器转换以将新的和更新的行先传递到更新策略转换，然后再传递到缓存目标。使用更新策略转换可根据 NewLookupRow 值将每一行的行类型更改为插入或更新。

您可以删除不会更改缓存的行，也可将它们传递到另一个目标。

根据 NewLookupRow 的值定义筛选器转换中的筛选器条件。例如，使用以下条件可同时将插入和更新的行传递到缓存目标：

NewLookupRow != 0

### 相关主题：

- [“配置上游更新策略转换” 页面上 295](#)

## 关联表达式

配置动态查找缓存时，您必须将每个查找端口与输入端口、输入/输出端口、序列 ID 或表达式相关联。选择输入/输出端口时，Designer 会将该输入/输出端口与查找条件中使用的查找/输出端口相关联。配置表达式时，集成服务会使用关联表达式的结果更新缓存。表达式可包含输入值或来自查找缓存的值。

要将查找端口与输入/输出端口相关联，单击查找端口的“关联表达式”列。从列表中选择一个端口。

要创建表达式，单击查找端口的“关联”列。从列表中选择“关联表达式”。此时将显示表达式编辑器。

要为目标表中的列创建生成键，在“关联表达式”列中选择“序列 ID”。您可以将生成键而不是查找端口的输入端口与 Bigint、Integer 或 Small Integer 数据类型相关联。对于 Bigint 查找端口，生成键最大值为 9,223,372,036,854,775,807。对于 Integer 或 Small Integer 查找端口，生成键最大值为 2,147,483,647。

如果在“关联表达式”列中选择了“序列 ID”，则集成服务会在将行插入查找缓存时生成一个键。

集成服务使用以下过程生成序列 ID：

1. 当集成服务创建动态查找缓存时，它会在动态查找缓存中跟踪具有序列 ID 的每个端口的值范围。
2. 当集成服务在缓存中插入一行数据时，它会通过将最大序列 ID 值增加 1 来为端口生成一个键。
3. 当集成服务达到最大的生成序列 ID 数目时，它将从 1 重新开始。集成服务将每个序列 ID 增加 1，直到其达到最小现有值减 1。如果集成服务用完唯一的序列 ID 号，会话将失败。

集成服务会为其插入缓存中的每一行生成一个序列 ID。

相关主题：

- “表达式结果的动态缓存更新” 页面上 298

空值

更新动态查找缓存和目标表时，源数据可能包含一些空值。集成服务可通过以下方式处理这些空值：

- 插入空值。**集成服务使用来自源的空值，并且使用来自源的所有值更新查找缓存和目标表。
- 忽略空值。**集成服务忽略源中的空值，并且仅使用来自源的非空值更新查找缓存和目标表。

如果您知道源数据包含空值，并且不希望集成服务使用空值更新查找缓存或目标，请选择相应查找/输出端口的“忽略空值”属性。

例如，如果您要更新主客户表。源包含新客户和姓氏已更改的当前客户。源包含姓名已更改客户的客户 ID 和姓名，但其地址列包含空值。您希望在主客户表中插入新客户并更新当前客户的姓名，同时保留当前的地址信息。

例如，主客户表包含以下数据：

Primary Key	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion James	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Jones	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210

源包含以下数据：

CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
80001	Marion Atkins	NULL	NULL	NULL	NULL
80002	Laura Gomez	NULL	NULL	NULL	NULL
99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

在映射的查找转换中选择“插入否则更新”。为查找转换中的所有查找/输出端口选择“忽略空值”选项。当运行会话时，集成服务将忽略源数据中的空值，并使用非空值更新查找缓存和目标表：

PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion Atkins	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Gomez	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210
100004	99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

**注意：**当选择忽略空值时，您必须确认将集成服务写入查找缓存的相同值输出到目标。当选择忽略空值时，如果将空输入值传递给目标，则查找缓存和目标表可能不同步。根据您希望集成服务在更换缓存中的行时从查找/输出端口输出的值配置映射：

- 新值。**仅将查找/输出端口从查找转换连接至目标。
- 旧值。**在查找转换后，筛选器或路由器转换前添加一个表达式转换。请在表达式转换中为目标表中的每个端口添加输出端口并创建表达式，以确保不会将空输入值输出到目标中。

## 在比较中忽略端口

当运行使用动态查找缓存的会话时，默认情况下，集成服务会将所有查找端口中的值与其关联输入端口中的值相比较。比较这些值是为了确定是否更新查找缓存中的行。当输入端口中的值与查找端口中的值不同时，集成服务将更新缓存中的行。

如果不希望比较所有端口，您可以选择希望集成服务在比较端口时忽略的端口。Designer 仅会为查找条件中未使用的查找/输出端口启用此属性。您可以通过在比较中忽略一些端口来提高性能。

当源数据中包括指示行是否包含需更新数据的列时，您可能要执行此操作。除了指示是否要在缓存和目标表中更新新的端口，为其他所有查找端口选择“在比较中忽略”属性。

您必须将查找转换配置为比较至少一个端口。如果忽略所有端口，集成服务将无法处理会话。

要清除“在比较中忽略”属性，请将一个端口与查找端口相关联。当清除了“在比较中忽略”属性时，PowerCenter 集成服务会更新缓存中的行。

## SQL 替代

使用动态缓存在查找转换中添加 WHERE 子句时，在查找转换之前连接筛选器转换以筛选您不想插入到缓存或目标表中的行。如果您不包括筛选器转换，可能会在缓存和目标表之间获得不一致的结果。

例如，配置查找转换以按 EMP\_ID 在员工表、EMP、匹配行上执行动态查找。定义以下查找 SQL 替代：

```
SELECT EMP_ID, EMP_STATUS FROM EMP ORDER BY EMP_ID, EMP_STATUS WHERE EMP_STATUS = 4
```

首次运行会话映射时，集成服务基于查找 SQL 替代从目标表构建查找缓存。缓存中的所有行均匹配 WHERE 子句中的条件，EMP\_STATUS = 4。

例如，集成服务读取符合您指定的查找条件的源行，但是 EMP\_STATUS 的值为 2。虽然目标可能包含 EMP\_STATUS 为 2 的行，但是集成服务因 SQL 替代仍未在缓存中找到该行。集成服务将行插入缓存并将行传递到目标表。当集成服务将此行插入目标表时，如果该行已存在则可能会获得不一致的结果。此外，并不是缓存中的所有行都匹配 SQL 替代中 WHERE 子句中的条件。

要验证您是否仅将行插入匹配 WHERE 子句的缓存，请在查找转换之前添加筛选器转换并将筛选器条件定义为查找 SQL 替代中 WHERE 子句中的条件。

对于上述示例，在筛选器转换中输入以下筛选器条件，在 SQL 替代中输入 WHERE 子句。

```
EMP_STATUS = 4
```

## 查找转换值

查找转换包含输入端口、查找和输出端口的值。如果启用动态查找缓存，输出端口值将因动态查找缓存的配置方式而异。

查找转换包含以下类型的值：

输入值

集成服务传递到查找转换的值。

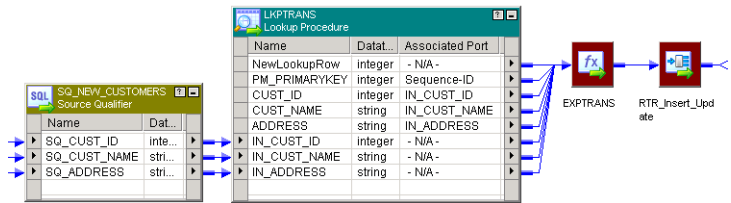
查找值

集成服务插入到缓存中的值。

输出值

集成服务从查找转换的输出端口传递的值。查找/输出端口的输出值取决于在集成服务更新行时选择输出旧值还是输出新值。

例如，以下查找转换使用动态查找缓存：



例如，创建一个含有以下对象的映射：

Source > Lookup transformation > Expression transformation> Router transformation > Target

为查找转换启用动态查找缓存。

您定义了以下查找条件：

IN\_CUST\_ID = CUST\_ID

默认情况下，所有输入到查找转换中的行的类型均为“插入”。要在缓冲和目标表中同时执行插入和更新，可以在查找转换中选择**插入 Else 更新**属性。

以下部分介绍运行会话时缓存中的行、输入行、查找行以及输出行的值。

以下部分介绍运行映射时缓存中的行、输入行、查找行以及输出行的值。

## 初始缓存值

运行会话映射时，集成服务会通过具有以下数据的目标表构建查找缓存：

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion James	100 Main St.
100002	80002	Laura Jones	510 Broadway Ave.
100003	80003	Shelley Lau	220 Burnside Ave.

## 输入值

源包含目标表中存在的行和不存在的行。以下行将从源限定符转换传递到查找转换：

SQ_CUST_ID	SQ_CUST_NAME	SQ_ADDRESS
80001	Marion Atkins	100 Main St.
80002	Laura Gomez	510 Broadway Ave.
99001	Jon Freeman	555 6th Ave.

## 查找值

集成服务根据查找条件在缓存中查找值。该服务更新缓存中 ID 为 80001 和 80002 的现有客户的行。对于客户 ID 99001，该服务在缓存中插入一个行。集成服务为新生成新键 (PK\_PRIMARYKEY)。

下表显示了从查找返回的行和值：

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion Atkins	100 Main St.
100002	80002	Laura Gomez	510 Broadway Ave.
100004	99001	Jon Freeman	555 6th Ave.

## 输出值

集成服务根据其动态缓存执行的插入和更新来标记查找转换中的行。这些行通过表达式转换传递到路由器转换，然后路由器转换进行筛选并将插入和更新的行传递给更新策略转换。更新策略转换根据 NewLookupRow 端口的值标记这些行。

查找/输出端口和输入/输出端口的输出值取决于在集成服务更新行时选择输出旧值还是输出新值。但是，新行和已更新行的 NewLookupRow 端口及使用序列 ID 的任何查找/输出端口的输出值相同。

如果选择输出新值，查找/输出端口将输出以下值：

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion Atkins	100 Main St.
2	100002	80002	Laura Gomez	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

如果选择输出旧值，查找/输出端口将输出以下值：

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion James	100 Main St.
2	100002	80002	Laura Jones	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

集成服务更新查找缓存中的行时，对缓存和目标表中的行使用主键 (PK\_PRIMARYKEY) 值。

对于未在缓存中找到的客户，集成服务使用序列 ID 生成主键。集成服务将主键值插入到查找缓存中，并将该值返回到查找/输出端口。

集成服务从输入/输出端口输出与输入值匹配的值。

**注意：**如果输入值为空，并对关联的输入端口选择“忽略空值”属性，则输入值不等于查找值或来自输入/输出端口的值。选择“忽略空值”属性时，如果将空值传递给目标，则查找缓存和目标表可能不同步。必须确认未将空值传递给目标。

# 动态查找缓存更新

集成服务根据行类型、查询结果和查找转换属性来更新动态缓存。

当您使用动态查找缓存时，将输入查找转换的行的行类型定义为插入或更新。您可以将一些行定义为插入，一些行定义为更新，也可将所有行定义为插入，或所有行定义为更新。默认情况下，所有输入查找转换的行的行类型均为插入。您可以在查找转换前添加一个更新策略转换以将行类型定义为更新。

集成服务会在缓存中插入或更新行，或者不对缓存进行更改。默认情况下，当行类型是插入时查找转换在缓存中插入行，当行类型是更新时查找转换在缓存中更新行。

但是，您可以配置以下查找属性来更改集成服务处理缓存中插入和更新的方式：

- **插入 Else 更新**。适用于输入查找转换的行类型为插入的行。使用此选项可在数据已存在于缓存中的情况下更新缓存中的行。如果不启用此选项，则不管行是否存在于缓存中，集成服务都会将它们插入缓存中。
- **更新 Else 插入**。适用于输入查找转换的行类型为更新的行。使用此选项可在数据不存在于缓存中的情况下插入更新的行。如果不启用此选项，集成服务会忽略缓存中不存在的行。

**注意：**您可以选择“插入 Else 更新”或“更新 Else 插入”属性，也可以同时选择或不选两个属性。

您可以配置查找转换以同步查找源和动态缓存。当您同步查找源和缓存时，查找转换会直接在查找源中插入行。您不会将插入行传递到目标。使用此配置可使用将行插入相同目标的查找转换运行多个会话。

## 插入 Else 更新

使用“插入 Else 更新”属性以在插入行类型时更新动态查找缓存中的现有行。

该属性仅适用于输入查找转换且行类型为“插入”的行。当任何其他类型的行（如“更新”）输入查找转换时，**插入 Else 更新**属性对集成服务处理行的方式没有影响。

如果选择**插入 Else 更新**并且输入查找转换的行类型为插入，则集成服务会在行为新行时将其插入到缓存中。如果行存在于索引缓存中但数据缓存不同于当前行，则集成服务会更新数据缓存中的行。

如果不选择**插入 Else 更新**并且输入查找转换的行类型为插入，则集成服务会在行为新行时将其插入到缓存中，并且如果该行已存在则不对缓存进行更改。

下表介绍了当输入到查找转换的行的类型为插入时，集成服务如何更改查找缓存：

插入 Else 更新选项	在缓存中发现行	数据缓存不同	查找缓存结果	NewLookupRow 值
已清除 - 仅插入	是	-	无更改	0
已清除 - 仅插入	否	-	插入	1
已选定	是	是	更新	2 <sup>1</sup>
已选定	是	否	无更改	0
已选定	否	-	插入	1

<sup>1</sup> 如果对查找条件中不存在的所有查找端口选择“忽略空值”，并且如果所有这些端口中都包含空值，集成服务将不更改缓存，并且 NewLookupRow 值等于 0。

## 更新 Else 插入

更新行类型时，使用“更新否则插入”属性可在动态查找缓存中插入新行。

可以在查找转换中选择**更新 Else 插入**属性。此属性仅适用于进入要更新行类型的查找转换的行。当任何其他行类型的行（例如插入）进入查找转换时，此属性对集成服务处理该行的方式不产生任何影响。

选择此属性并更新进入查找转换的行类型时，如果索引缓存中存在该行，但缓存数据与现有行不同，集成服务将更新缓存中的相应行。如果行是新行，集成服务会将其插入缓存中。

如果在未选择此属性时更新进入查找转换的行类型，则只要该行存在，集成服务就会更新缓存中的相应行，如果该行是新行，则不对缓存进行更改。

如果对查找条件中不存在的所有查找端口选择**忽略空值**，并且如果所有这些端口中都包含空值，集成服务将不更改缓存，并且 NewLookupRow 值等于 0。

下表介绍了更新进入查找转换的行的行类型时，集成服务如何更改查找缓存：

更新 Else 插入选项	在缓存中发现行	数据缓存不同	查找缓存结果	NewLookupRow 值
已清除（仅限更新）	是	是	更新	2
已清除（仅限更新）	是	否	无更改	0
已清除（仅限更新）	否	-	无更改	0
已选定	是	是	更新	2
已选定	是	否	无更改	0
已选定	否	-	插入	1

## 具有动态查找的映射

当配置包含动态查找转换的映射时，配置查找转换的上游转换可影响查找转换更新动态缓存的方式。配置查找转换的下游转换可确保集成服务在目标中更新标记为更新的行并插入标记为插入的行。

### 配置上游更新策略转换

您可以配置上游更新策略转换以更改查找转换接收的行的行类型。

使用动态查找缓存时，使用更新策略转换可为以下行定义行类型：

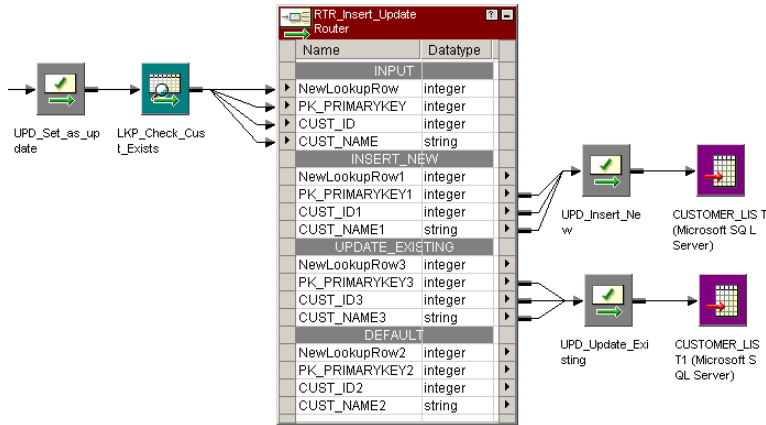
- **进入查找转换的行。**默认情况下，进入查找转换的所有行的行类型为插入。但是，在查找转换之前使用更新策略转换可将行定义为更新，或将一些行定义为更新并将一些行定义为插入。
- **离开查找转换的行。**NewLookupRow 值指示集成服务如何更改查找缓存，但是其不更改行类型。在查找转换后使用筛选器或路由器转换以根据 NewLookupRow 值指导离开查找转换的行。在筛选器或路由器转换之后，使用更新策略转换来标记要在映射中的目标定义之前插入或更新的行。

**注意：**如果要删除未更改的行，请勿将筛选器或路由器转换中的行连接到 NewLookupRow 等于 0 的目标。

对于进入查找转换的行，将行类型定义为插入时，在查找转换中使用“插入 Else 更新”属性。对于进入查找转换的行，将行类型定义为更新时，在查找转换中使用“更新 Else 插入”属性。如果将进入查找转换的一些行定义为更新并将一些行定义为插入，则使用“更新 Else 插入”或“插入 Else 更新”属性，或者同时使用两个属性。



下图显示了具有多个更新策略转换以及一个使用动态缓存的查找转换的映射：



在此示例中，查找转换之前的更新策略转换将所有行标记为更新。在查找转换中选择“更新 Else 插入”属性。路由器转换会将插入的行发送到“Insert\_New 更新策略”转换，并将更新的行发送到“Update\_Existing 更新策略”转换。未连接到目标的输出行将被删除。查找转换右侧的两个更新策略转换针对目标将行标记为插入或更新。

## 配置下游转换

配置下游转换可确保动态查找缓存和目标保持同步。

使用动态查找缓存时，集成服务在写入目标表之前先写入查找缓存。如果集成服务不将数据写入目标，查找缓存和目标表可能会不同步。例如，目标数据库可能会拒绝数据。

请考虑以下指导原则以保持查找缓存与查找表同步：

- NewLookupRow 值等于一或二时，请对缓存的目标使用路由器转换。
- 如果 NewLookupRow 值等于零，请使用路由器转换以删除行。或者，将行输出至其他目标。
- 在查找转换之后使用更新策略转换以标记要插入或更新到目标的行。
- 运行会话时将错误阈值设置为 1。将错误阈值设置为 1 时，会话将在遇到第一个错误时失败。集成服务不会将新缓存文件写入磁盘。相反，它会还原原始缓存文件（如果存在）。您还必须将前期会话目标表还原至目标数据库。
- 验证查找转换输出到目标的值与集成服务写入查找缓存的值是否相同。选择在更新上输出新值时，仅将查找/输出端口连接至目标表（而非输入/输出输出端口）。如果选择在更新时输出旧值，请在查找转换之后路由器转换之前添加一个表达式转换。请在表达式转换中为目标表中的每个端口添加输出端口并创建表达式，以确保不会将空输入值输出到目标中。
- 在会话属性中将“将源行视为”属性设置为“数据驱动”。
- 定义更新策略目标表选项时选择“插入”和“更新”。这将确保集成服务更新标记为更新的行，插入标记为插入的行。在会话属性的“映射”选项卡上的“转换”视图选择这些选项。

## 查找条件列中的空值

集成服务处理查找条件列中含空值的方式不同，具体取决于该行是否存在于缓存中。

如果该行不存在于查找缓存中，集成服务会将该行插入到缓存中并将其传递到目标表。如果该行存在于查找缓存中，集成服务则不在缓存或目标表中更新该行。

**注意：**如果源数据的查找条件列中包含空值，请将错误阈值设置为 1。这样可确保集成服务在缓存中插入行时查找缓存和表保持同步，但数据库会由于非空约束拒绝该行。



## 使用动态查找缓存配置会话

当配置具有更新策略转换和动态查找缓存的会话时，您必须配置以下更新策略表选项之一：

- 选择插入
- 选择更新时更新
- 不要选择删除

更新策略目标表选项可确保集成服务更新标记为更新的行，插入标记为插入的行。

在会话属性中“属性”选项卡的“常规选项”设置上，将“将源行视为”选项定义为“数据驱动”。如果不选择“数据驱动”，集成服务会标记您在“将源行视为”选项中所指定行类型的所有行，而不会在映射中使用更新策略转换来标记行。集成服务不会插入和更新正确的行。如果不选择“更新时更新”，集成服务不会正确更新目标表中标记为要更新的行。因此，查找缓存和目标表可能会不同步。

## 条件动态缓存更新

可以根据布尔表达式的结果更新动态查找缓存。该表达式为 true 时，集成服务更新缓存。

例如，可能在目标表中有产品编号、现存数量以及时间戳列。需要使用最新源值更新现存数量。当源数据的时间戳大于动态缓存中的时间戳时，可以更新现存数量。在查找转换中创建一个类似以下表达式的表达式：

```
lookup_timestamp < input_timestamp
```

该表达式可以包含查找和输入端口。您可以访问内置、映射和参数变量。您可以包含用户定义的函数并引用未连接的转换。

表达式返回 true、false 或 NULL。如果表达式的结果是 NULL，则表达式为 false。集成服务不会更新缓存。如果需要将表达式结果更改为 true，则可以为表达式中的 NULL 值添加复选框。默认表达式值为 true。

使用 Transformation Developer 创建表达式。不能覆盖会话级别的**动态缓存更新条件**。

## 会话处理

配置条件动态缓存更新时，如果数据不存在，集成服务不会考虑条件表达式。启用了“插入否则更新”时，如果数据不存在，集成服务会在缓存中插入一行。如果数据存在，它会在条件表达式为 true 时更新缓存。启用“更新否则插入”时，如果缓存中存在数据且条件表达式为 true，则集成服务会更新缓存。当缓存中不存在数据时，集成服务会在缓存中插入行。

当表达式为 true 时，NewLookupRow 值为 1，集成服务会更新该行。当表达式为 false 或空值时，NewLookupRow 值为 0。集成服务不会更新该行。

如果将查找转换配置为同步动态缓存，则集成服务在向缓存插入新行时在查找源中插入一个行。

## 配置条件动态缓存查找

必须先启用查找转换以执行条件动态缓存查找，然后才能创建条件表达式。

使用以下步骤来配置条件动态缓存查找：

1. 在 Transformation Developer 中创建一个动态查找转换。
2. 单击“属性”选项卡。
3. 启用“查找缓存”和“动态查找缓存”属性。
4. 要访问表达式编辑器，单击“更新动态缓存条件”属性的“执行”按钮。

5. 定义表达式条件。可以为表达式选择输入端口、查找端口和函数。
6. 单击“验证”以验证表达式是否有效。

如果您不验证表达式，Designer 会在您关闭表达式编辑器的时候对其进行验证。如果表达式无效，Designer 会显示警告。

## 表达式结果的动态缓存更新

您可以使用表达式的结果更新动态查找缓存。

例如，产品表目标具有一个包含顺序计数的数值列。每次查找转换收到产品的顺序，都会通过以下表达式的结果更新动态缓存 `order_count`：

```
order_count = order_count + 1
```

查找转换返回 `order_count`。

可以配置集成服务如何处理表达式计算结果为 NULL 的情况。

### 空表达式值

如果表达式中的一个值为空，则表达式将返回空值。但可以将表达式配置为返回非空值。

如果表达式引用查找端口，但源数据为新数据，则查找端口将包含一个默认值。默认值可能为 NULL。可以配置 `IsNull` 表达式以检查空值。

例如，以下表达式将检查 `lookup_column` 是否为 NULL：

```
iif (isnull(lookup_column), input_port, user_expression)
```

如果列为空，则返回 `input_port` 值。否则返回表达式的值。

### 会话处理

集成服务可以根据表达式结果在动态查找缓存中插入和更新行。表达式结果可能会因查找端口值是否为空以及是否包含在表达式中而有所不同。

启用“插入 Else 更新”后，如果数据不在缓存中，集成服务会插入含有表达式结果的行。如果数据不存在于缓存中，查找端口值为 NULL。如果表达式引用查找端口值，集成服务将置换表达式中的默认端口值。如果启用“插入 Else 更新”，并且数据存在于缓存中，则集成服务会用表达式结果更新缓存。

启用“更新 Else 插入”后，如果数据存在于缓存中，集成服务会用表达式结果更新缓存。如果数据不存在于缓存中，集成服务会插入一个包含表达式结果的行。如果表达式引用查找端口值，集成服务将置换表达式中的默认端口值。

如果将查找转换配置为同步动态缓存，集成服务会在查找缓存中插入一个含有表达式结果的行。其在查找源中插入一个含有表达式结果的行。

### 配置动态缓存更新的表达式

必须先启用查找转换以执行动态查找，然后才能创建条件表达式。

使用以下步骤来配置动态缓存查找的表达式：

1. 在 Transformation Developer 中创建一个动态查找转换。
2. 打开“属性”选项卡。

3. 启用“查找缓存”和“动态查找缓存”属性。
4. 要创建表达式，单击要更新的查找端口的“关联表达式”列表。
5. 选择“关联表达式”。  
此时将显示表达式编辑器。
6. 通过选择输入端口、查找端口和函数来定义表达式。表达式返回值必须与查找端口类型相匹配。
7. 单击“验证”以验证表达式是否有效。  
如果您不验证表达式，Designer 会在您关闭表达式编辑器的时候对其进行验证。如果表达式无效，Designer 会显示警告。

## 同步缓存与查找源

查找转换将保留动态查找缓存，以跟踪其传递到目标的行。当多个会话更新同一目标时，可以配置每个会话中的查找转换，以便将动态查找缓存同步到同一查找源（而不是目标）。

在配置查找转换以使缓存与查找源同步时，查找转换将对查找源执行查找。如果查找源中不存在数据，则查找转换会在更新动态查找缓存之前，将行插入到查找源中。

如果其他会话已经插入了该行，则查找源中可能会存在数据。要将查找缓存同步到查找源，集成服务会从查找源中检索最新的值。查找转换会将值从查找源插入到动态查找缓存中。查找源必须是关系表。

例如，您有多个会话同时运行。每个会话都会为新产品名称生成产品编号。当某一会话生成了产品编号后，其他会话必须使用同一产品编号来标识该产品。产品编号将生成一次，然后插入到查找源中。如果其他会话要处理包含该产品的行，则该会话必须使用位于该查找源中的产品编号。每个会话都会对该查找源执行查找，以确定已经生成了哪些产品编号。

集成服务将为插入行执行以下任务：

- 集成服务将对动态查找缓存执行查找。如果动态查找缓存中不存在数据，则集成服务将对查找源执行查找。
- 如果查找源中存在数据，则集成服务将检索查找源中的数据。它将在动态查找缓存中插入一行，其中包含来自查找源中的列。它不会使用源行更新缓存。
- 如果查找源中没有数据，则集成服务会将数据插入到查找源中，然后将行插入到缓存中。

查找源包含与查找缓存相同的列。集成服务不会将列插入到查找缓存中，除非该列是从查找转换突出的，或者该列是查找条件的组成部分。

## NewLookupRow

将动态缓存与查找源同步时，查找转换针对更新行的行为不会更改。当集成服务接收到更新行时，它将更新动态查找缓存并将 NewLookupRow 的值设置为 2。它不会更新查找源。您可以通过更新策略转换路由更新行，并将这些行传递到目标。

当集成服务将这些行插入到查找源时，会将 NewLookupRow 设置为 1。集成服务更新动态查找缓存。将转换配置为同步动态缓存且 NewLookupRow 为 1 时，无需将插入行传递到目标。

在查找转换中配置了“插入 Else 更新”属性且源行为插入行时，集成服务将在缓存和查找源中插入该行或更新缓存。

在查找转换中配置了“更新 Else 插入”属性且源行为更新行时，集成服务将更新缓存或在缓存和查找源中插入该行。

**注意：**不同的会话处理同一个表的更新行时，处理顺序不尽相同。可能会产生意外结果。

# 配置动态缓存同步

创建查找转换时，您可以将其配置为同步动态缓存与插入行的查找源。

要配置动态缓存同步：

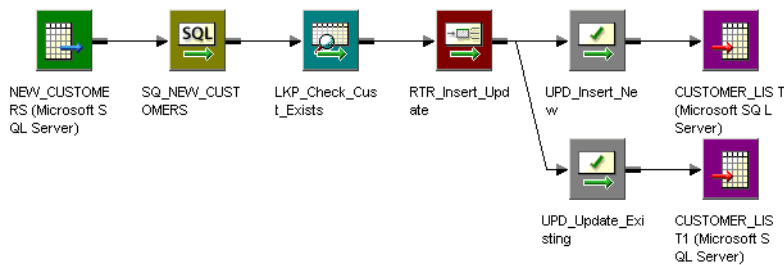
- 1. 打开“查找转换属性”选项卡。
- 2. 启用“动态查找缓存”。
- 3. 启用“同步动态缓存”。

# 动态查找缓存示例

当需要在目标中插入和更新行时，使用动态查找缓存。使用动态查找缓存时，您可以在缓存中插入和更新传递到要插入和更新的目标的相同数据。

例如，需要更新包含客户数据的表。源数据包含要在目标中插入或更新的客户数据行。创建代表目标的动态缓存。配置查找转换以查找缓存中的客户。

下图显示了具有动态缓存的映射：



查找转换使用动态查找缓存。会话开始时，集成服务从 Customer\_List 表构建查找缓存。Customer\_List 表也是映射中的目标。集成服务读取不在查找缓存中的行时，会在缓存中插入该行。

查找转换将该行返回到路由器转换。路由器转换将该行定向至 UPD\_Insert\_New 或 UPD\_Update\_Existing 转换。更新策略转换将该行标记为插入或更新，然后再将其传递到目标。

Customer\_List 表在会话运行时发生更改。集成服务在查找缓存中插入新行并更新现有行。集成服务使查找缓存和 Customer\_List 表保持同步。

要生成目标的键，可在关联端口中使用序列 ID。序列 ID 为集成服务插入到目标表中的新行生成主键。

由于从数据库构架一次缓存，因此使用动态查找缓存可以提高会话性能。即使目标表中的数据发生更改，您也可以继续使用查找缓存。

# 动态查找缓存的规则和准则

使用动态查找缓存时应用的某些规则和准则。

使用动态查找缓存时，请考虑以下准则：

- 使用动态查找缓存时，必须设置**多项匹配时**属性以报告错误。要重置该属性，请将动态查找更改为静态查找、更改该属性，然后将静态查找更改为动态查找。

- 无法在同一目标加载顺序组中的动态查找转换与静态查找转换之间共享缓存。
- 可以为关系、平面文件或源限定符转换查找启用动态查找缓存。
- 可以为关系或平面文件查找启用动态查找缓存。
- 查找转换必须是已连接的转换。
- 可以使用持久或非持久缓存。
- 如果动态缓存不持久，则即使不启用**从查找源重新缓存**，集成服务也始终从数据库重建缓存。
- 如果动态缓存不持久，则即使不启用**从查找源重新缓存**，集成服务也始终从数据库重建缓存。
- 将动态缓存文件与查找源表同步时，查找转换将在查找源表和动态查找缓存中插入行。如果源行为更新行，查找转换将仅更新动态查找缓存。
- 可以仅创建一个等式查找条件。不能在动态缓存中查找数据的范围。
- 必须将不在查找条件中的每个查找端口与输入端口、序列 ID 或关联的表达式相关联。
- NewLookupRow 值等于一或二时，请对缓存的目标使用路由器转换。
- 如果 NewLookupRow 值等于零，请使用路由器转换以删除行。或者，可以将行输出到不同的目标。
- 验证集成服务是否将相同的值输出到写入查找缓存的目标。如果选择在更新时输出新值，则仅将查找/输出端口（而不是输入/输出端口）连接到目标表。如果选择在更新时输出旧值，请在查找转换之后路由器转换之前添加一个表达式转换。请在表达式转换中为目标表中的每个端口添加输出端口并创建表达式，以确保不会将空输入值输出到目标中。
- 使用查找 SQL 替代时，请将正确的列映射到恰当的查找目标。
- 将 WHERE 子句添加到查找 SQL 替代时，请先应用筛选器转换，然后再应用查找转换。这样可确保集成服务在与 WHERE 子句匹配的动态缓存和目标表中插入行。
- 配置可重用查找转换以使用动态缓存时，无法在映射中编辑条件或禁用**动态查找缓存**属性。
- 请先应用查找转换，然后再应用更新策略转换，以便为目标标记要插入或更新的行。
- 如果要在查找转换中使用“更新 Else 插入”属性，请先应用更新策略转换，然后再应用查找转换，以将部分或全部行定义为更新。
- 在会话属性中将行类型设置为“数据驱动”。
- 在会话属性中选择“插入并更新”作为目标表选项的更新。

## 第 20 章

# 规范器转换

本章包括以下主题：

- [规范器转换概览, 302](#)
- [规范器转换要素, 303](#)
- [规范器转换生成的键, 305](#)
- [VSAM 规范器转换, 306](#)
- [管道规范器转换, 310](#)
- [在映射中使用规范器转换, 313](#)
- [规范器转换故障排除, 316](#)

## 规范器转换概览

规范器转换接收到包含多次出现的列的行，并为多次出现的数据的每个实例返回一行。转换将处理每个源行中多次出现的列或多次出现的列组。规范器转换是主动转换。

规范器转换解析 COBOL 源、关系表或其他源中多次出现的列。它可以处理包含 REDEFINES 子句的 COBOL 源中的多种记录类型。

例如，您可能有一个关系表按商店存储四个季度的销售额。现在您需要为每个出现的销售额返回一个行。可以将规范器转换配置为针对每个季度返回单独的一行。

以下源行包含各个商店四个季度的销售数据：

```
Store1 100 300 500 700
Store2 250 450 650 850
```

规范器将针对每个商店和销售数据的组合返回一行。另外还会返回标识季度数的索引：

```
Store1 100 1
Store1 300 2
Store1 500 3
Store1 700 4
Store2 250 1
Store2 450 2
Store2 650 3
Store2 850 4
```

规范器转换将为每个源行生成一个键。集成服务每次处理源行时都会增大生成的键序列号。如果源行包含多次出现的列或多次出现的列组，则规范器转换将针对每次出现返回一行。每一行包含相同的生成键值。

当规范器返回源行中的多个行时，它将针对单次出现的源列返回重复数据。例如，对于销售数据的每个实例，Store1 和 Store2 的返回数据重复。

可以创建 VSAM 规范器转换或管道规范器转换：

- **VSAM 规范器转换。**不可重用转换，是 COBOL 源的源限定符转换。Mapping Designer 从映射中的 COBOL 源创建 VSAM 规范器列。列属性为只读。VSAM 规范器通过一个输入端口接收多次出现的源列。
- **管道规范器转换。**处理关系表或平面文件中多次出现的数据的转换。在 Transformation Developer 或 Mapping Designer 中可手动创建列并对其进行编辑。每次源列出现时，管道规范器转换将通过一个输入端口表示多次出现的列。

## 规范器转换要素

规范器转换包含以下选项卡：

- **转换。**输入转换的名称和说明。规范器转换的命名约定是 `NRM_TransformationName`。也可以使管道规范器转换可重用。
- **端口。**查看转换端口和属性。
- **属性。**配置跟踪级别，以确定会话日志文件中报告的事务详细信息量。选择该项将在下个会话中重置或重新启动生成的键序列值。
- **规范器。**定义源数据的结构。“规范器”选项卡将源数据定义为列和列组。
- **元数据扩展。**配置扩展名称、数据类型、精度和值。也可以创建可重用元数据扩展。

### “端口”选项卡

定义规范器转换时，可在“规范器”选项卡中配置列。Designer 将创建端口。您可以在“端口”选项卡上查看规范器端口和属性。

管道和 VSAM 规范器转换表示的多次出现的源列不同。VSAM 规范器转换对于多次出现的列有一个输入端口。管道规范器转换对于多次出现的列有多个输入端口。

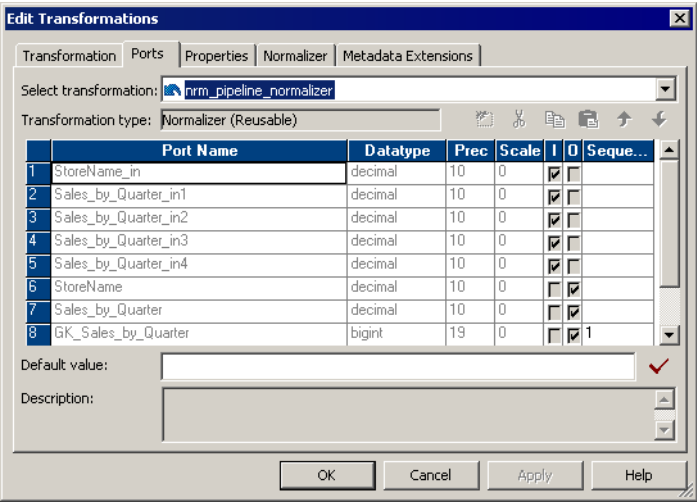
规范器转换对于每个单次出现的输入端口有一个输出端口。当源列多次出现时，管道和 VSAM 规范器转换针对该列都有一个输出端口。每次源列出现，转换都会返回一行。

对于每个多次出现的列，规范器转换都有一个生成列 ID (GCID) 端口。生成的列 ID 是多次出现数据的实例的索引。例如，如果某个列在源记录中出现四次，规范器会基于多次出现数据出现在行中的具体实例，在生成的列 ID 中返回值 1、2、3 或 4。

规范器生成列 ID 的命名约定是 `GCID_<occurring_field_name>`。

规范器转换至少有一个生成键端口。集成服务每次处理源行时都会增大生成的键序列号。

下图显示了规范器转换的“端口”选项卡：



在本例中，Sales\_By\_Quarter 在源中多次出现。该规范器转换对于 Sales\_By\_Quarter 有一个输出口。它针对每个源行返回四行。生成的键起始值为 1。

通过编辑“规范器”选项卡上的列可以更改管道规范器转换中的端口。要更改 VSAM 规范器转换，需要更改 COBOL 源并重新创建转换。

## “属性”选项卡

在“属性”选项卡上可配置规范器转换的常规属性。

可以配置的规范器转换属性如下：

属性	必需/ 可选	说明
重置	必需	在会话结束时，将每个生成键值的值序列重置为其在会话前的值。
重新开始	必需	从 1 开始生成的键序列。每次运行会话时，键序列值从 1 开始并替代“端口”选项卡上的序列值。
跟踪级别	必需	设置运行包含此转换的会话时会话日志中包含的详细信息量。

## “规范器”选项卡

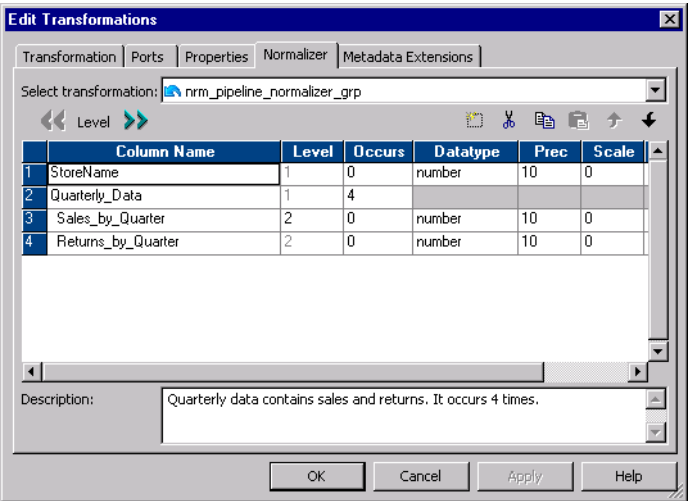
“规范器”选项卡定义源数据的结构。“规范器”选项卡将源数据定义为列和列组。列组可能定义 COBOL 源中的记录，也可能定义源中多次出现的字段组。

列级别数标识数据中的列组。级别数定义数据层次结构。一个组中列的级别数相同，并按顺序显示在组级列下方。组级列的级别数较低，且其中不含数据。



下图显示了管道规范器转换的“规范器”选项卡：

图 3. “规范器”选项卡



Quarterly\_Data 是一个组级列。它是 1 级别。Quarterly\_Data 组在每行中出现四次。Sales\_by\_Quarter 和 Returns\_by\_Quarter 是 2 级列，归属于该组。

每个列都有“出现次数”属性。“出现次数”属性标识源行中多次出现的列或列组。

创建管道规范器转换时，可以编辑这些列。创建 VSAM 规范器转换时，“规范器”选项卡为只读状态。

下表介绍了 VSAM 和管道规范器转换通用的“规范器”选项卡属性：

属性	说明
列名称	源列的名称。
级别	对列进行分组。同一组中的列显示在具有较低级别编号的列下方。当每个列都为同一级别时，转换中不包含列组。
出现次数	一个列或一组列在源行中的实例的数量。
数据类型	转换列数据类型可以是 String、Nstring 或 Number。
Prec	精度。列的长度。
小数位数	数字列的小数位数。

VSAM 规范器转换的“规范器”选项卡包含的属性与管道规范器转换相同，但前者包含特定于 COBOL 源定义的属性。

## 规范器转换生成的键

规范器转换在输出行中至少返回一个生成的键列。集成服务每次处理源行时都会增大生成的键序列号。集成服务根据规范器转换“端口”选项卡中生成的键值确定初始键值。创建规范器转换时，默认情况下生成的键值为 1。规范器生成键的命名约定是 GK\_<redefined\_field\_name>。

相关主题：

- [“生成键值” 页面上 315](#)

## 存储生成的键值

可以在规范器转换的“端口”选项卡中查看当前生成的键值。在每个会话结束时，集成服务会将规范器转换中生成的键值更新成为该会话生成的最后一个值加一。生成的最大键值为 9,223,372,036,854,775,807。如果存储库中有规范器转换的多个实例，则集成服务会在运行会话时更新在所有版本中生成的键值。

**注意：**无法在规范器转换的“端口”选项卡中更改当前生成的键值。

## 更改生成的键值

可通过以下方式更改生成的键值：

- **重置生成的键序列。**在规范器转换的“属性”选项卡上重置生成的键序列。重置生成的键序列时，集成服务会将生成的键起始值重置回其在会话前的值。如果希望在每次运行会话时创建相同的生成键值，请重置生成的键序列。
- **重新启动生成的键序列。**在规范器转换的“属性”选项卡上重新启动生成的键序列。重新启动生成的键序列后，集成服务将在下次运行会话时从 1 开始生成的键序列。重新启动生成的键序列后，规范器转换中的生成键起始值在运行会话前不会更改。运行会话时，集成服务将替代“端口”选项卡上的序列号值。

重置或重新启动生成的键序列时，重置或重新启动会影响下次运行会话时的生成键序列值。不要更改规范器转换中的当前生成键序列值。重置或重新启动生成的键序列后，系统会为每个会话都启用该选项，直到您将该选项禁用。

# VSAM 规范器转换

VSAM 规范器转换是 COBOL 源定义的源限定符。COBOL 源是一个平面文件，可以包含多次出现的数据和同一文件中多种类型的记录。

VSAM（虚拟存储访问方法）是一种适用于 IBM 大型机操作系统的文件访问方法。VSAM 文件在索引文件或连续平面文件中整理记录。不过，可将 VSAM 规范器转换用于使用 COBOL 源定义定义的任何平面文件源。

COBOL 源定义可以包含 OCCURS 语句，用于定义多次出现的列。COBOL 源定义还可包含 REDEFINES 语句，用于在文件中定义多种类型的记录。

以下 COBOL 复写簿定义了销售记录：

```
01 SALES_RECORD.  
  03 HDR_DATA.  
    05 HDR_REC_TYPE      PIC X.  
    05 HDR_STORE         PIC X(02).  
  03 STORE_DATA.  
    05 STORE_NAME        PIC X(30).  
    05 STORE_ADDR1       PIC X(30).  
    05 STORE_CITY        PIC X(30).  
  03 DETAIL_DATA REDEFINES STORE_DATA.  
    05 DETAIL_ITEM       PIC 9(9).  
    05 DETAIL_DESC       PIC X(30).  
    05 DETAIL_PRICE      PIC 9(4)V99.  
    05 DETAIL_QTY        PIC 9(5).  
    05 SUPPLIER_INFO OCCURS 4 TIMES.  
      10 SUPPLIER_CODE   PIC XX.  
      10 SUPPLIER_NAME   PIC X(8).
```

销售文件可以包含两种类型的销售记录。Store\_Data 定义了一家商店，而 Detail\_Data 则定义了该商店中出售的商品。REDEFINES 子句指示 Detail\_Data 字段可以出现在记录中，代替 Store\_Data 字段。

每个销售记录的前三个字符是表头。表头包括记录类型和商店 ID。Hdr\_Rec\_Type 的值定义记录的其余部分是包含商店信息，还是包含商品信息。例如，如果 Hdr\_Rec\_Type 为“S”，则记录包含商店数据。如果 Hdr\_Rec\_Type 为“D”，则记录包含详细数据。

如果记录包含详细数据，则将包括 Supplier\_Info 字段。OCCURS 子句在每个 Detail\_Data 记录中定义了四家供应商。

下图显示了可以通过 COBOL 复写簿创建的 Sales\_File COBOL 源定义：

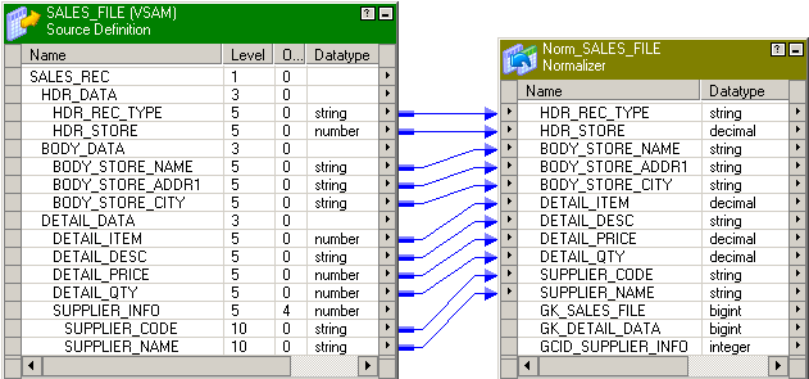
K	Name	Level	Occurs	Datatype	Length
	SALES_REC	1	0		0
	HDR_DATA	3	0		0
	HDR_REC_TYPE	5	0	string	1
	HDR_STORE	5	0	number	2
	STORE_DATA	3	0		0
	STORE_NAME	5	0	string	30
	STORE_ADDR1	5	0	string	30
	STORE_CITY	5	0	string	30
	DETAIL_DATA	3	0		0
	DETAIL_ITEM	5	0	number	9
	DETAIL_DESC	5	0	string	30
	DETAIL_PRICE	5	0	number	6
	DETAIL_QTY	5	0	number	5
	SUPPLIER_INFO	5	4	number	0
	SUPPLIER_CODE	10	0	string	2
	SUPPLIER_NAME	10	0	string	8

Sales\_Rec、Hdr\_Data、Store\_Data、Detail\_Data 和 Supplier\_Info 列是组级别的列，用于标识更低级别数据的组。组级别的列长度为 0，因为它们不包含数据。这些列都不是源定义中的输出端口。

Supplier\_Info 组包含 Supplier\_Code 和 Supplier\_Name 列。Supplier\_Info 组在每个 Detail\_Data 记录中出现了四次。

在通过 COBOL 源定义创建 VSAM 规范器转换时，Mapping Designer 会根据 COBOL 源定义在规范器转换中创建输入/输出端口。规范器转换包含至少一个生成键输出端口。如果 COBOL 源包含多次出现的列，则规范器转换将包含一个生成列 ID 输出类型。

下图显示了 Mapping Designer 通过源定义创建的规范器转换端口：



多个列的 Supplier\_Info 组在每个 COBOL 源行中出现了四次。

COBOL 源行可以包含以下数据：

Item1 ItemDesc 100 25 A Supplier1 B Supplier2 C Supplier3 D Supplier4

Supplier\_Code 和 Supplier\_Name 列每出现一次，规范器转换都将返回一行。每个输出行都包含相同的项、说明、价格和数量值。

规范器将通过 COBOL 源行返回以下详细数据行：

```
Item1 ItemDesc 100 25 A Supplier1 1 1
Item1 ItemDesc 100 25 B Supplier2 1 2
Item1 ItemDesc 100 25 C Supplier3 1 3
Item1 ItemDesc 100 25 D Supplier4 1 4
```

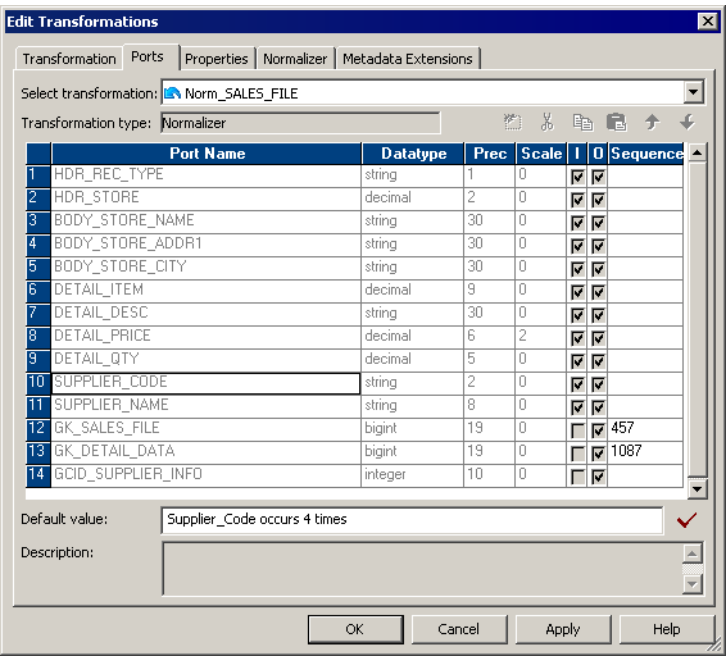
每个输出行都包含一个生成键和一个列 ID。集成服务将在处理新源行时更新生成键的值。在详细数据行中，生成键的值为 1。

列 ID 定义 Supplier\_Info 列的出现次数。Supplier\_Info 每出现一次，集成服务都会更新列 ID。在详细数据行中，列 ID 值为 1、2、3、4。

## “VSAM 规范器端口” 选项卡

“VSAM 规范器端口” 选项卡显示转换输入和输出端口。每个 COBOL 源列具有一个输入/输出端口。每个多次出现的列具有一个输入/输出端口。转换对于组级列没有输入或输出端口。

下图显示了 “VSAM 规范器端口” 选项卡：



在本示例中，Supplier\_Code 和 Supplier\_Name 在 COBOL 源中出现了四次。“端口” 选项卡显示了一个 Supplier\_Code 端口和一个 Supplier\_Name 端口。生成键起始值为 457 和 1087。

## “VSAM 规范器” 选项卡

在创建 VSAM 规范器转换时，Mapping Designer 将通过 COBOL 源创建列。“规范器” 选项卡将显示与 COBOL 源定义相同的信息。无法编辑 “VSAM 规范器” 选项卡上的列。

下表介绍了“VSAM 规范器”选项卡上的属性：

属性	说明
POffs	物理偏移量。文件中字段的位置。文件中的第一个字节为 0。
Plen	物理长度。字段中的字节数。
列名称	源字段的名称。
级别	提供列组层次结构。级别编号越高，则数据在层次结构中的位置越低。同一组中的列显示在具有较低级别编号的列下方。当每个列都为同一级别时，转换中不包含列组。
出现次数	一个列或一组列在源行中的实例的数量。
数据类型	转换的数据类型可能是 String、Nstring 或 Number。
Prec	精度。列的长度。
小数位数	数字列的小数位数。
图片	数据在源中存储或显示的方式。图片 99V99 定义一个数字字段，包含两位隐含小数。图片 X(10) 表示 10 个字符。
用法	COBOL 数据存储格式，如 COMP、BINARY 和 COMP-3。如果“用法”为 DISPLAY，则图片子句将定义在查看源数据时如何对其进行格式化。
键类型	应用于 VSAM 文件字段的键约束的类型。选择以下键类型之一： <ul style="list-style-type: none"> <li>- 不是键。该字段不是 VSAM 文件中的索引。</li> <li>- 主键。该字段是 VSAM 文件中的主索引。该字段包含唯一值。</li> <li>- 替代键。该字段是 VSAM 文件中的二级索引。该字段包含唯一值。</li> <li>- 主重复键。该字段是 VSAM 文件中的主索引。该字段可以包含重复值。</li> <li>- 替代重复键。该字段是 VSAM 文件中的二级索引。该字段可以包含重复值。</li> </ul>
带符号 (S)	指示数字值是否带符号。
尾随符号 (T)	指示该字段的最后一位存在的符号 (+ 或 -)。如果未启用，则符号将显示为该字段的第一个字符。
已包含符号 (I)	指示该字段内显示的任何值中是否包括符号。
真实小数点 (R)	指示在数字字段中，小数点是使用圆点 (.)，还是使用 V 字符表示。
重新定义	指示该列重新定义其他列。
业务名称	提供给某一列的描述性名称。

## 创建 VSAM 规范器转换的步骤

在创建 VSAM 规范器转换时，可将 COBOL 源拖动到映射中，而 Mapping Designer 将从该源创建转换列。规范器转换是适用于映射中 COBOL 源的源限定符。

在将 COBOL 源添加到映射中后，Mapping Designer 将创建并配置规范器转换。Mapping Designer 将标识 COBOL 源中的嵌套记录和多次出现的字段。它将通过源列在规范器转换中创建列和端口。

要创建 VSAM 规范器转换：

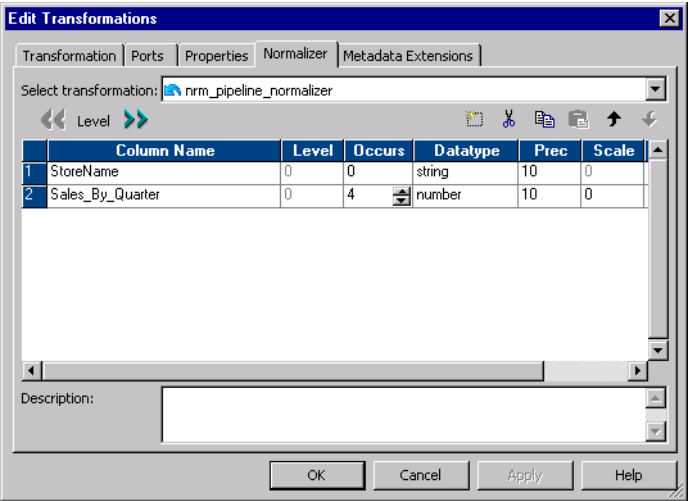
1. 在 Mapping Designer 中，创建一个新映射，或者打开一个现有映射。

2. 将 COBOL 源定义拖动到该映射中。
- Designer 将添加规范器转换，并将其连接到 COBOL 源定义。默认情况下，如果尚未启用创建源限定符的选项，将显示“创建规范器转换”对话框。
3. 如果显示了“创建规范器转换”对话框，可以从以下选项中进行选择：
- **VSAM 源。**在映射中通过 COBOL 源定义创建转换。
  - **管道。**创建转换，但不通过 COBOL 源定义列。可以在“规范器”选项卡上手动定义列。如果想要处理映射内其他转换中多次出现的数据，可以选择此选项。
- 要创建 VSAM 规范器转换，请选择“VSAM 规范器转换”选项。该对话框将显示映射中 COBOL 源定义的名称。选择 COBOL 源定义，然后单击“确定”。
4. 打开规范器转换。
5. 选择“端口”选项卡，以查看规范器转换中的端口。
- 默认情况下，Designer 将通过 COBOL 源定义创建端口。
6. 单击“规范器”选项卡，以查看源列的组织。
- “规范器”选项卡包含的信息与 COBOL 源的“列”选项卡相同。不过，您无法修改规范器转换中的列属性。要更改列属性，请更改 COBOL 复写簿，导入 COBOL 源，然后重新创建规范器转换。
7. 选择“属性”选项卡，以设置跟踪级别。
- 还可以将转换配置为在下一个会话开始时重置生成键序列号。

# 管道规范器转换

在 Transformation Developer 中创建规范器转换时，默认情况下会创建管道规范器转换。创建管道规范器转换时，可基于该转换从源限定符转换等其他类型的转换接收到的数据定义列。Designer 将基于您定义的列创建输入和输出规范器转换端口。

下图显示了某个转换的规范器转换列，该转换接收每个关系源行中的四个销售数据列：



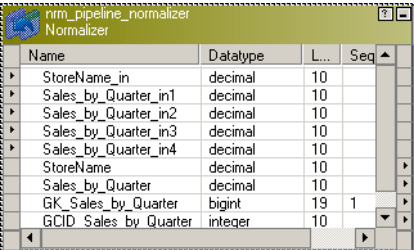
每个源行有一个 StoreName 列和四个 Sales\_By\_Quarter 实例。

源行可能包含以下数据：

```
Dellmark 100 450 650 780
Tonys    666 333 444 555
```

管道规范器转换对于多次出现的列的每个实例有一个输入端口。

下图显示了 Designer 基于规范器转换中的列创建的端口：



Name	Datatype	Length	Sequence
StoreName_in	decimal	10	
Sales_by_Quarter_in1	decimal	10	
Sales_by_Quarter_in2	decimal	10	
Sales_by_Quarter_in3	decimal	10	
Sales_by_Quarter_in4	decimal	10	
StoreName	decimal	10	
Sales_by_Quarter	decimal	10	
GK_Sales_by_Quarter	bigint	19	1
GCID_Sales_by_Quarter	integer	10	

该规范器转换针对多次出现的列的每个实例返回一行。

```
Dellmark 100 1 1
Dellmark 450 1 2
Dellmark 650 1 3
Dellmark 780 1 4
Tonys    666 2 1
Tonys    333 2 2
Tonys    444 2 3
Tonys    555 2 4
```

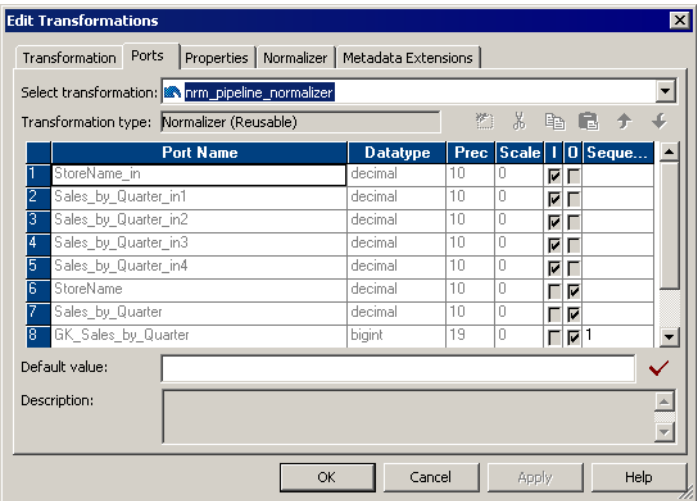
集成服务每次处理源行时都会增大生成的键序列号。生成的键将每季度销售数据与同一个商店链接起来。在本例中，Dellmark 行的生成键为 1。Tonys 商店的生成键为 2。

对于多次出现的字段的每个实例，该转换都会返回一个生成列 ID (GCID)。在本例中，GCID\_Sales\_by\_Quarter 值始终为 1、2、3 或 4。

## 管道规范器“端口”选项卡

管道规范器的“端口”选项卡显示转换的输入和输出端口。对于您在转换中定义的每个单次出现的列，它都有一个输入/输出端口。对于多次出现的列的每次出现，都有一个端口。转换对于组级列没有输入或输出端口。

下图显示了管道规范器转换的“端口”选项卡：



Designer 将针对多次出现的列的每次出现创建一个输入端口。

要更改管道规范器转换中的端口，请修改“规范器”选项卡中的列。增加列出现次数时，Designer 将增加一个输入端口。Designer 将为最低级别创建端口，而不会为组级列创建端口。

## 管道“规范器”选项卡

创建管道规范器转换时，可在“规范器”选项卡中定义列。Designer 将基于您在“规范器”选项卡上输入的列创建输入和输出端口。

下表介绍了管道“规范器”选项卡属性：

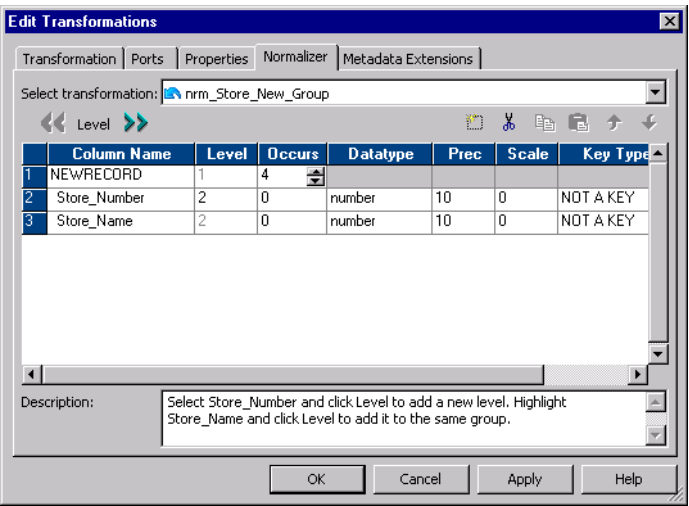
属性	说明
列名称	列的名称。
级别	标识列组。同组中的列具有相同的级别数。默认值为零。当每个列都为同一级别时，转换中不包含列组。
出现次数	一个列或一组列在源行中的实例的数量。
数据类型	列数据类型可以是 String、Nstring 或 Number。
Prec	精度。列的长度。
小数位数	数值中的小数位数。

### “规范器”选项卡列组

当源行包含重复列的组时，可以在“规范器”选项卡上定义列组。规范器转换针对每次列组出现（而不是每次列出现）返回一行。

“规范器”选项卡上的级别数标识列的层次结构。组级列标识列组。组级列比组中列的级别数低。同组中列的级别数相同，并且按顺序显示在“规范器”选项卡的组级列下方。

下图显示了“规范器”选项卡中一组多次出现的列：



在本例中，NEWRECORD 列中没有数据。它是 1 级组列。该组在每个源行中出现四次。Store\_Number 和 Store\_Name 是 2 级列。它们属于 NEWRECORD 组。



# 创建管道规范器转换的步骤

- 创建管道规范器转换时，可在“规范器”选项卡中定义列。
- 可以在 Transformation Developer 或 Mapping Designer 中创建规范器转换。
- 要创建规范器转换，请执行以下操作：
- 在 Transformation Developer 或 Mapping Designer 中，单击“转换”>“创建”。选择规范器转换。输入规范器转换的名称。  
规范器转换的命名约定为 *NRM\_TransformationName*。
  - 单击“创建”和“完成”。
  - 打开规范器转换并单击“规范器”选项卡。
  - 单击“添加”以添加新列。  
Designer 会使用默认属性创建新列。可以更改名称、数据类型、精度和小数位数。
  - 要创建多次出现的列，在“出现次数”列中输入出现次数。
  - 要创建一组多发生列，请在“规范器”选项卡上至少输入一列。选择列。单击“级别”。  
Designer 会在选定的列之上添加一个 NEWRECORD 组级别列。NEWRECORD 将成为级别 1。选定的列将成为级别 2。您可以重命名 NEWRECORD 列。  
默认情况下，所有列都处于同一级别。级别定义组合在一起的列。
  - 您可以更改其他列的级别，以将它们添加到同一组。选择一列并单击“级别”，将其更改为与其上面的列相同的级别。  
同一组中的列必须按顺序显示在“规范器”选项卡中。
  - 在组级别更改出现次数，以使列组多次出现。
  - 单击“应用”保存列并创建输入和输出端口。  
Designer 会创建规范器转换输入和输出端口。此外，Designer 会为每个多发生列或列组创建生成的键列和列 ID。
  - 选择“属性”选项卡，在下一会话后更改跟踪级别或重置生成的键序列号。

# 在映射中使用规范器转换

当规范器转换收到来自 COBOL 源的多种类型的数据时，需要根据每行中数据的类型将规范器输出端口连接到不同的目标。以下示例介绍了如何通过规范器转换将 Sales\_File COBOL 源定义映射到多个目标。

Sales\_File 源记录包含商店信息或有关商店销售物品的信息。销售文件包含两种类型的记录。

以下示例包含两项销售文件记录：

Record Type	Data
Store Record	H01Software Suppliers Incorporated 1111 Battery Street San Francisco
Item Record	D01123456789USB Line - 10 Feet 001495000020 01Supp1 02Supp2 03Supp3 04Supp4

COBOL 源定义和规范器转换具有表示两种类型记录中的字段的列。您需要从物品行中筛选出商店行，并将其传递到其他目标。

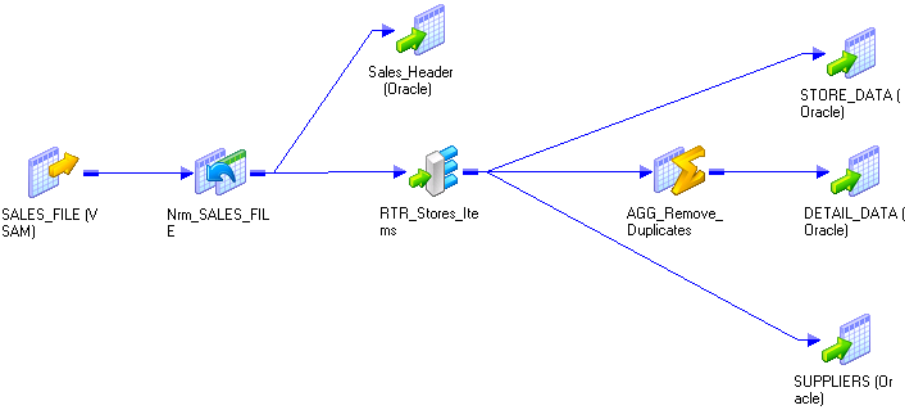
下图显示了 Sales\_File COBOL 源及其对应的 Store\_Data（具有值 “S” ）和 Detail\_Data（具有值 “D” ）：

K	Name	Level	Occurs	Datatype	Length
	SALES_REC	1	0		0
	HDR_DATA	3	0		0
	HDR_REC_TYPE	5	0	string	1
	HDR_STORE	5	0	number	2
	STORE_DATA	3	0		0
	STORE_NAME	5	0	string	30
	STORE_ADDR1	5	0	string	30
	STORE_CITY	5	0	string	30
	DETAIL_DATA	3	0		0
	DETAIL_ITEM	5	0	number	9
	DETAIL_DESC	5	0	string	30
	DETAIL_PRICE	5	0	number	6
	DETAIL_QTY	5	0	number	5
	SUPPLIER_INFO	5	4	number	0
	SUPPLIER_CODE	10	0	string	2
	SUPPLIER_NAME	10	0	string	8

Hdr\_Rec\_Type 定义记录是包含商店数据还是商品数据。如果 Hdr\_Rec\_Type 值为 “S”，则记录包含 Store\_Data。如果 Hdr\_Rec\_Type 为 “D”，则记录包含 Detail\_Data。Detail\_Data 始终包含 Supplier\_Info 字段的四次出现。

要筛选数据，请将规范器输出行连接到路由器转换，以将商店、物品和供应商数据路由到不同目标。可以根据 Hdr\_Rec\_Type 的值筛选路由器转换中的行。

下图显示了将 Sales\_File 记录路由到不同目标的映射：



该映射将多个记录类型从 COBOL 源筛选到关系目标。多发生源列会映射到单独的关系表中。通过源行中的出现为每行编制索引。

该映射包含以下转换：

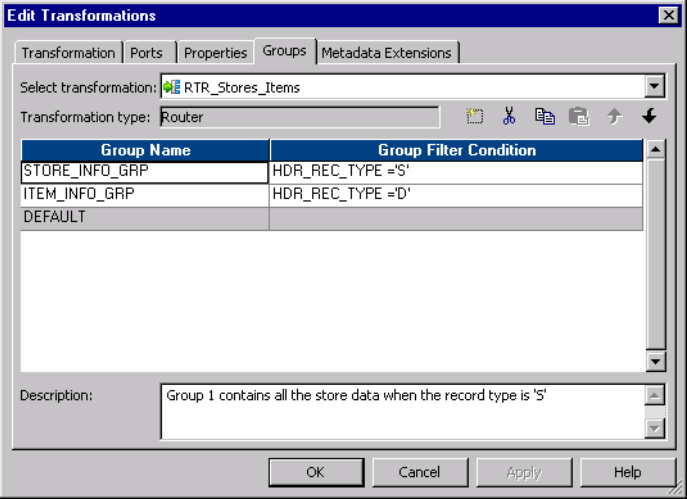
- **规范器转换。**当源包含多发生 Detail\_Data 时，规范器转换会返回多行。它还处理来自同一源的不同记录类型。
- **路由器转换。**路由器转换根据 Hdr\_Rec\_Type 的值将数据路由到目标。
- **汇总器转换。**汇总器转换会删除随 Supplier\_Info 的每次出现而出现的重复 Detail\_Data 行。

该映射包含以下功能：

1. 规范器转换将表头记录类型和表头商店编号列传递到 Sales\_Header 目标。每条 Sales\_Header 记录都具有一个生成的键，该键将 Sales\_Header 行链接到 Store\_Data 或 Detail\_Data 目标行。规范器在每行返回 Hdr\_Data 和 Store\_Data 一次。
2. 规范器转换将所有列传递到路由器转换。它会每行传递 Detail\_Data 数据四次，Supplier\_Info 列的每次出现一次。Detail\_Data 列包含重复数据，但 Supplier\_Info 列除外。

3. Hdr\_Rec\_Type 为 “S” 时，路由器转换会将商店名称、地址、城市和生成的键传递到 Store\_Data。生成的键会将 Store\_Data 行链接到 Sales\_Header 行。  
路由器转换包含一个用户定义的商店数据组和一个用户定义的商品项组。
4. 当 Hdr\_Rec\_Type 为 “D” 时，路由器转换会将物品、物品说明、价格、数量和 Detail\_Data 生成的键传递到汇总器转换。
5. 当 Hdr\_Rec\_Type 为 “D” 时，路由器转换会将供应商代码、名称和列 ID 传递到 “供应商” 目标。会将链接 “供应商” 行的生成的键传递到 Detail\_Data 行。
6. 汇总器转换会删除重复的 Detail\_Data 列。汇总器会将物品、说明、价格、数量和生成的键的一个实例传递到 Detail\_Data。Detail\_Data 生成的键会将 Detail\_Data 行链接到 “供应商” 行。Detail\_Data 还具有将 Detail\_Data 行链接到 Sales\_Header 行的键。

下图显示路由器转换中用户定义的组和筛选条件：



路由器转换根据记录类型传递商店数据或物品数据。

## 生成键值

规范器转换会在 COBOL 源包含一组多次出现的列时创建生成键。您可以将一组多次出现的列传递到与行中其他列不同的目标。您可以在具有生成键的目标之间创建主键-外键关系。

下图显示了包含一组多次出现的列的 COBOL 源定义：

Detail_Sales (VSAM)				
Name	Level	Occurs	Datatype	Length
DETAIL_RECORD	1	0		0
DETAIL_ITEM	3	0	number	9
DETAIL_DESC	3	0	string	30
DETAIL_PRICE	3	0	number	6
DETAIL_QTY	3	0	number	5
DETAIL_SUPPLIERS	3	4	number	0
SUPPLIER_CODE	10	0	string	2
SUPPLIER_NAME	10	0	string	8

在此示例中，Detail\_Suppliers 列组在 Detail\_Record 中出现四次。

规范器转换为每个源行生成一个 GK\_Detail\_Sales 键。该 GK\_Detail\_Sales 键代表一个 Detail\_Record 源行。

下图显示了目标之间的主外键关系：

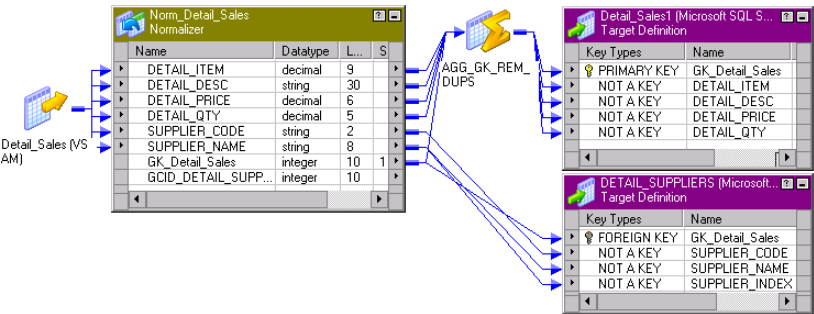
Key Types	Name	Datatype	Length...
FOREIGN KEY	GK_Detail_Sales	int	10
NOT A KEY	SUPPLIER_CODE	varchar	2
NOT A KEY	SUPPLIER_NAME	varchar	8

Key Types	Name	Datatype	Length...
PRIMARY KEY	GK_Detail_Sales	int	10
NOT A KEY	DETAIL_ITEM	numeric	9
NOT A KEY	DETAIL_DESC	varchar	30
NOT A KEY	DETAIL_PRICE	numeric	6
NOT A KEY	DETAIL_QTY	numeric	5

多次出现的 Detail\_Supplier 行具有将它们链接到相同 Detail\_Sales 行的外键。Detail\_Sales 目标与 Detail\_Suppliers 目标间存在一对多关系。

下图显示了连接到目标中主键和外键的 GK\_Detail\_Sales 生成键：



将 GK\_Detail\_Sales 传递到 Detail\_Sales 的主键和 Detail\_Suppliers 的外键。

将规范器输出列链接到以下对象：

- **Detail\_Sales\_Target.**将 Detail\_Item、Detail\_Desc、Detail\_Price 和 Detail\_Qty 列传递到 Detail\_Sales 目标。将 GK\_Detail\_Sales 键传递到 Detail\_Sales 主键。
- **汇总器转换。**通过汇总器转换传递每个 Detail\_Sales 行以删除重复的行。规范器针对 Detail\_Suppliers 的每次出现返回重复的 Detail\_Sales 列。
- **Detail\_Suppliers.**将 Detail\_Suppliers 列的每个实例传递到 Detail\_Suppliers 目标。将 GK\_Detail\_Sales 键传递到 Detail\_Suppliers 外键。Detail\_Suppliers 列的每个实例具有一个将 Detail\_Suppliers 行关联到 Detail\_Sales 行的外键。

# 规范器转换故障排除

使用关系源时，我无法编辑规范器转换中的端口。

当您手动创建端口时，将其添加到转换中的“规范器”选项卡而非“端口”选项卡。

导入 COBOL 文件时因发生数字错误而失败。我应该怎么处理？

验证 COBOL 程序是否遵循 COBOL 标准，包括空格、选项卡和行尾字符。COBOL 文件标题应类似于以下文本：

```
identification division.  
    program-id. mead.  
environment division.  
    select file-one assign to "fname".  
data division.  
file section.  
fd FILE-ONE.
```

Designer 不读取 COBOL 程序中的隐藏字符。使用纯文本编辑器更改 COBOL 文件。请不要使用 Word 或 Wordpad。删除多余的空格。

读取二进制数据已完成但目标表中的信息不正确的会话。

在 Workflow Manager 中编辑该会话并验证源文件格式是否设置正确。文件格式可以是 EBCDIC 或 ASCII。记录之间要跳过的字节数必须设置为 0。

我有使用非 IBM COMP 类型的 COBOL 字段说明。我该如何导入源？

在源定义中，清除 IBM COMP 选项。

在我的映射中，我使用一个表达式转换和一个查找转换修改规范器转换中的两个输出端口。映射会将这两个转换连接到一个转换。所有端口都位于同一级别下。当我检查在目标中加载的数据时，发现数据不正确。这是什么原因？

您只能连接级别 1 中的端口。删除连接。

## 第 21 章

# 等级转换

本章包括以下主题：

- [等级转换概览, 318](#)
- [等级转换中的端口, 319](#)
- [定义组, 320](#)
- [创建等级转换, 321](#)

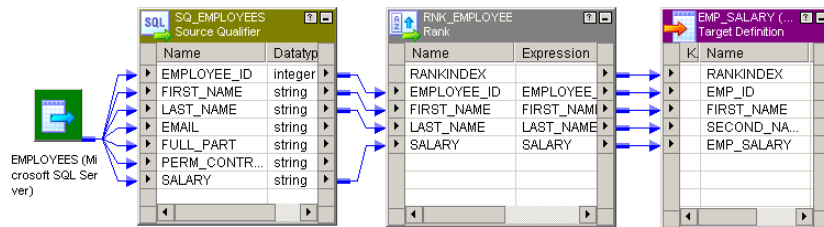
## 等级转换概览

可以通过等级转换仅选择等级最高或最低的数据。等级转换是主动转换。使用等级转换可返回端口或组中的最大或最小数值。还可以使用等级转换返回位于会话排序顺序顶部或底部的字符串。在会话期间，集成服务会缓存输入数据，直到执行等级计算。

等级转换与转换函数 MAX 和 MIN 不同，它允许您选择一组顶部或底部值，而不只是一个值。例如，使用等级可以选择给定区域中的前 10 名销售员。或者，等级转换也可用于生成财务报表，可以使用该转换找出薪资和管理费用最低的三个部门。尽管 SQL 语言提供了许多用于处理成组数据的函数，但是无法使用标准 SQL 函数在一组行中确定最高或最低等级。

可以将表示同一行集的所有端口连接到该转换。只有落在该等级（基于配置转换时设置的某个度量）内的行将传递给等级转换。还可以写入表达式以转换数据或执行计算。

下图显示了通过等级转换从人力资源表传递员工数据的映射。等级转换仅将薪资最高的前 10 位员工对应的行传递到下一个转换。



作为一种主动转换，等级转换可能会更改传递它的行的数量。您可能将 100 个行传递给等级转换，但只选择排名前 10 位的行，将它们从等级转换传递到另一个转换。

可以仅将某一转换的端口连接到等级转换。还可以创建本地变量和编写非汇总表达式。

## 为字符串值评级

集成服务在 ASCII 数据移动模式下运行时，会使用二进制排序顺序对会话数据进行排序。

集成服务在 Unicode 数据移动模式下运行时，会使用为会话配置的排序顺序。在会话属性中选择会话排序顺序。会话属性会根据集成服务使用的代码页列出所有可用的排序顺序。

例如，您将等级转换配置为返回字符串端口的前三个值。配置工作流时，选择要运行工作流的集成服务。会话属性会显示与所选集成服务的代码页关联的所有排序顺序，例如法语、德语和二进制。如果将会话配置为使用二进制排序顺序，则集成服务会计算每个字符串的二进制值，并返回具有字符串最高二进制值的三个行。

## 等级缓存

在会话期间，集成服务会将输入行与数据缓存中的行进行比较。如果输入行的等级高于缓存行，则集成服务会将缓存行替换为输入行。如果将等级转换配置为跨多个组排名，则集成服务会为找到的每个组递增排名。

集成服务将组信息存储在索引缓存中，将行数据存储在数据缓存中。如果在管道中创建多个分区，则集成服务会为每个分区创建单独的缓存。

## 等级转换属性

创建等级转换时，可以配置以下属性：

- 输入缓存目录。
- 选择最高或最低等级。
- 选择用于确定等级的值所在的输入/输出端口。只能选择一个端口来定义等级。
- 选择属于等级内的行数。
- 定义等级组，如每个制造商的价格最低的 10 种产品。

## 等级转换中的端口

等级转换包括连接到映射中的另一转换的输入端口或输入/输出端口。它还包括变量端口和等级端口。使用等级端口可指定要确定等级的列。

下表介绍了等级转换中的端口。

端口	所需数量	说明
I	最少为一	输入端口。创建输入端口以从另一转换接收数据。
O	最少为一	输出端口。为您要链接到另一个转换的每个端口创建输出端口。可以将输入端口指定为输出端口。
V	不需要	变量端口。可以用于存储要在表达式中使用的值或计算。变量端口不能是输入端口或输出端口。变量端口仅在该转换内传递数据。
R	仅一个	等级端口。用于指定要为值进行等级划分的列。可在等级转换中仅指定一个等级端口。等级端口是输入/输出端口。必须将等级端口链接到另一个转换。

# 等级索引

Designer 会为每个等级转换创建 RANKINDEX 端口。集成服务将使用该等级索引端口来存储组中每一行的等级位置。

例如，如果要创建一个等级转换，按各个季度为前五名销售人员划分等级，则等级索引将对销售人员进行编号，编号依次为 1 至 5：

RANKINDEX	SALES_PERSON	SALES
1	Sam	10,000
2	Mary	9,000
3	Alice	8,000
4	Ron	7,000
5	Alex	6,000

RANKINDEX 仅用作输出端口。可以将等级索引传递给映射中的其他转换，或者直接传递给目标。

# 定义组

与汇总器转换一样，等级转换使您可以对信息进行分组。例如，如果要按制造商选择最昂贵的 10 个产品，则可能首先要为每个制造商定义一个组。配置等级转换时，可以将它的一个输入/输出端口设置为分组依据端口。对于该组端口中的每个唯一值，该转换将创建一组符合等级定义的行（每个等级的最高点或最低点以及某一特定点）。

因此，等级转换将以两种不同的方式更改行数。通过筛选除最高等级或最低等级的行之外的所有行，可以减少传递给转换的行数。通过定义组，可为每个组创建一组分级的行。

例如，您可能会创建一个等级转换来标识公司中薪资最高的前 50 位员工。在此情况下，可以将 SALARY 列标识为用于度量等级的输入/输出端口，并配置该转换，使其筛选掉除前 50 位员工之外的所有行。

在等级转换标识了属于最高或最低等级的所有行之后，便可分配等级索引值。对于薪资最高的前 50 位员工，薪资最高的员工的等级索引为 1。薪资第二位的员工的等级索引为 2，以此类推。在度量最低等级（例如，清单中价格最低的 10 个产品）时，等级转换会按从低到高的顺序分配等级索引。因此，价格最低的产品的等级索引为 1。

如果两个等级值匹配，则它们将获得相同的等级索引值，因此，转换将跳过下一个值。例如，如果要查看某个国家/地区排名前五位的零售商店，并且其中两家商店的销售额相同，则返回数据可能类似于以下形式：

RANKINDEX	SALES	STORE
1	10000	Orange
1	10000	Brea
3	90000	Los Angeles
4	80000	Ventura



# 创建等级转换

在源限定符之后，可以在映射中的任意位置添加等级转换。

要创建等级转换，请执行以下操作：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择等级转换。输入等级名称。等级转换的命名约定为 *RNK\_TransformationName*。  
输入转换的说明。该说明显示在 Repository Manager 中。
2. 单击“创建”，然后单击“完成”。  
Designer 将创建等级转换。
3. 将输入转换中的列链接到等级转换。
4. 单击“端口”选项卡并为等级端口选择“等级 (R)”选项。  
如果要为划分等级的行创建组，请为定义组的端口选择“分组依据”。
5. 单击“属性”选项卡并选择要进行顶部还是底部等级划分。
6. 对于“等级数”选项，输入要为等级选择的行数。
7. 如有必要，更改其他等级转换属性。

下表介绍了等级转换属性：

设置	说明
缓存目录	集成服务创建索引和数据缓存文件的本地目录。默认情况下，集成服务使用 Workflow Manager 中输入的用于进程变量 \$PMCacheDir 的目录。如果输入新的目录，请确保目录存在并且包含缓存文件所需的足够磁盘空间。
顶部/底部	指定是否要获取列的最高或最低等级。
等级数	要进行等级划分的行数。
区分大小写的字符串比较	在 Unicode 模式下运行时，集成服务会根据为会话选择的排序顺序对字符串进行等级划分。如果会话排序顺序区分大小写，请选择此选项来启用区分大小写的字符串比较，清除此选项可使集成服务忽略字符串的大小写。如果排序顺序不区分大小写，集成服务会忽略此设置。默认情况下，此选项被选中。
跟踪级别	确定集成服务向会话日志写入的有关在会话中通过此转换所传递的数据的信息量。
等级数据缓存大小	转换的数据缓存大小。默认值为 2,000,000 个字节。如果配置的总会话缓存大小为 2 GB（2,147,483,648 字节）或更多，则必须在 64 位集成服务中运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。

设置	说明
等级索引缓存大小	转换的索引缓存大小。默认值为 1,000,000 个字节。如果配置的总会话缓存大小为 2 GB (2,147,483,648 字节) 或更多，则必须在 64 位集成服务中运行会话。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。
转换范围	指定集成服务如何将转换逻辑应用至传入数据： <ul style="list-style-type: none"><li>- 事务。将转换逻辑应用至事务中的所有行。当一行数据取决于同一事务中的所有行，但不取决于其他事务中的行时，请选择“事务”。</li><li>- 全部输入。将转换逻辑应用至所有传入数据。选择“全部输入”时，PowerCenter 将删除传入的事务边界。当一行数据取决于源中的所有行时，请选择“全部输入”。</li></ul>

8. 单击“确定”。

## 第 22 章

# 路由器转换

本章包括以下主题：

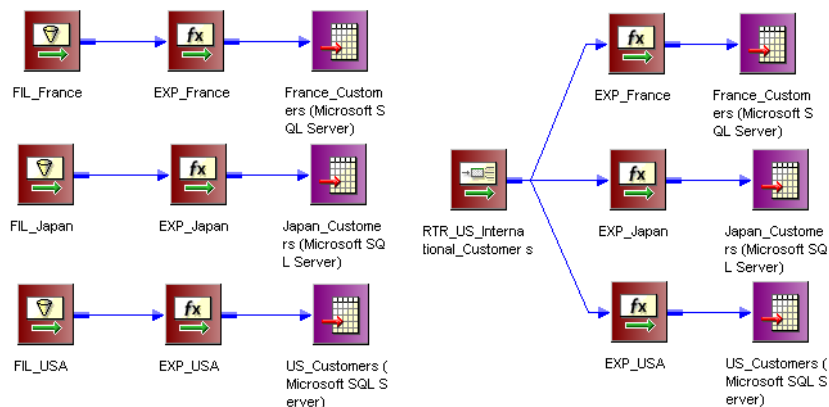
- [路由器转换概览, 323](#)
- [使用组, 324](#)
- [使用端口, 326](#)
- [在映射中连接路由器转换, 327](#)
- [创建路由器转换, 327](#)

## 路由器转换概览

路由器转换与筛选器转换类似，因为这两个转换都允许使用条件来测试数据。筛选器转换会根据一个条件来测试数据，并删除不满足该条件的数据行。但是，路由器转换会根据一个或多个条件来测试数据，并可让您选择将不满足其中任意条件的数据行路由到默认的输出组。路由器转换是主动转换。

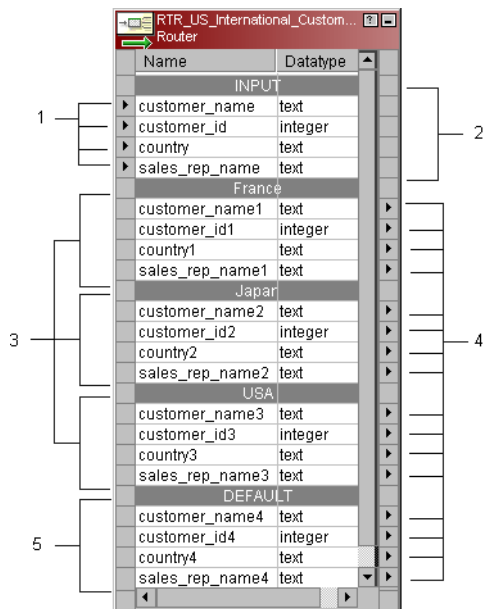
如果需要根据多个条件测试相同的输入数据，请在映射中使用一个路由器转换（而不是创建多个筛选器转换来执行同一任务）。路由器转换效率更高。例如，要根据三个条件测试数据，只需要一个路由器转换，而不是三个筛选器转换来执行该任务。同样，在映射中使用路由器转换时，集成服务将一次性处理所有传入数据。在映射中使用多个筛选器转换时，集成服务将处理每个转换的传入数据。

下图显示了执行同一任务的两个映射。第一个映射使用三个筛选器转换，而第二个映射使用一个路由器转换，二者产生的结果相同：



路由器转换由输入和输出组、输入和输出端口、组筛选条件以及属性（在 Designer 中配置）组成。

下图显示了示例路由器转换：



1. 输入端口。
2. 输入组。
3. 用户定义的输出组。
4. 输出端口
5. 默认输出组

# 使用组

路由器转换具有以下类型的组：

- 输入
- 输出

## 输入组

Designer 将复制输入组的输入端口中的属性信息，以便为每个输出组创建一组输出端口。

## 输出组

有两种类型的输出组：

- 用户定义的组
- 默认组

您无法修改或删除输出端口或其属性。

## 用户定义的组

可创建用户定义的组以便基于传入数据测试条件。用户定义的组由输出端口和组筛选条件组成。可以在“组”选项卡上使用 Developer 创建和编辑用户定义的组。为要指定的每个条件创建一个用户定义的组。

集成服务将使用该条件来计算传入数据的每一行。它将在处理默认组之前测试每个用户定义组的条件。集成服务将根据所连接的输出组的顺序确定每个条件的计算顺序。集成服务将处理连接到转换或映射中的目标的用户定义组。如果将默认组连接到转换或目标，则集成服务只会处理映射或目标中未连接的用户定义组。

如果某一行满足多个组筛选条件，则集成服务将多次传递此行。

## 默认组

在创建一个新的用户定义的组后，Designer 将创建默认组。Designer 不允许编辑或删除默认组。该组没有与其相关联的组筛选条件。如果所有条件的计算结果均为 FALSE，则集成服务会将相应行传递到默认组。如果希望集成服务丢弃默认组中的所有行，请不要将其连接到转换或映射中的目标。

在从列表中删除最后一个用户定义的组后，Designer 将删除默认组。

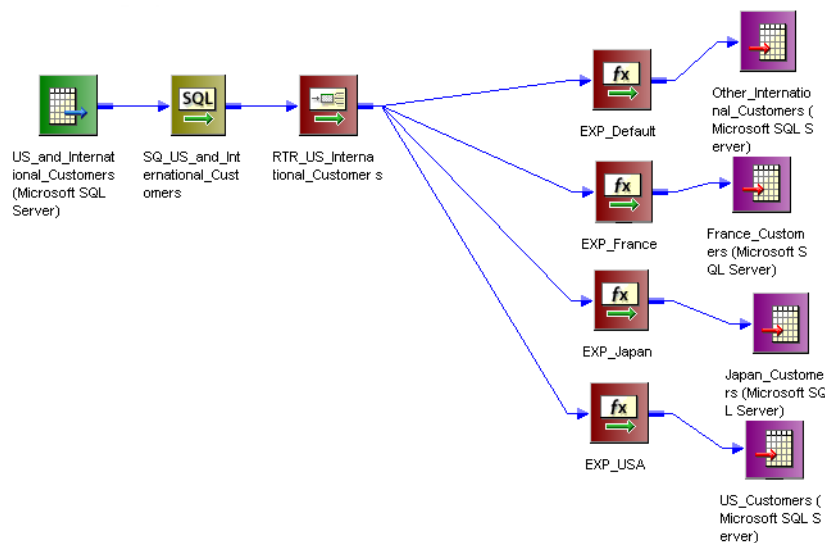
## 使用组筛选条件

可以根据一个或多个组筛选条件测试数据。可以使用表达式编辑器在组选项卡上创建组筛选条件。可以输入任意表达式以返回单一值。还可以为条件指定一个常量。对于传递转换的每一行，组筛选条件都会根据该行是否满足指定条件来返回 TRUE 或 FALSE。零 (0) 等价于 FALSE，任何非零值等价于 TRUE。集成服务会将计算结果为 TRUE 的数据行传递给与用户定义的每个组相关联的每个转换或目标。

例如，您的客户分布在九个国家/地区，您希望只对其中三个国家/地区的数据执行不同的计算。您可能想要在映射中使用一个路由器转换将此数据筛选到三个不同的表达式转换中。

没有与默认组相关联的组筛选条件。但是，您可以创建一个表达式转换以根据另外六个国家/地区的数据来执行计算。

下图显示了一个使用路由器转换的映射，该转换将根据多个条件来筛选数据：



由于您要根据三个不同国家/地区的数据执行多个计算，因此在“组”选项卡中创建三个用户定义的组，并指定三个组筛选条件。

下表显示了用于筛选客户数据的组筛选条件：

组名称	组筛选条件
法国	customer_name= 'France'
Japsn	customer_name= 'Japan'
USA	customer_name= 'USA'

在该会话中，集成服务会将计算结果为 TRUE 的数据行传递给与用户定义的每个组（Japan、France 和 USA）相关联的每个转换或目标。如果 *所有* 条件的计算结果均为 FALSE，则集成服务会将该行传递给默认组。如果发生这种情况，集成服务会将其他六个国家/地区的数据传递给与默认组相关联的转换或目标。如果希望集成服务丢弃默认组中的所有行，请不要将其连接到转换或映射中的目标。

路由器转换会将数据传递给满足条件的每个组。因此，如果数据满足三个输出组的条件，则路由器转换会通过三个输出组传递此数据。

例如，在路由器转换中配置了以下组条件：

组名称	组筛选条件
输出组 1	employee_salary > 1000
输出组 2	employee_salary > 2000

如果路由器转换处理的输入行数据的 employee\_salary=3000，则会将此数据路由到输出组 1 和 2。

## 添加组

添加组类似于在其他转换中添加端口。Designer 会将属性信息从输入端口复制到输出端口。

要将组添加到路由器转换，请执行以下操作：

1. 单击“组”选项卡。
2. 单击“添加”按钮。
3. 在“组名称”部分中输入新组的名称。
4. 单击“组筛选条件”字段并打开表达式编辑器。
5. 输入组筛选条件。
6. 单击验证检查条件的语法。
7. 单击“确定”。

## 使用端口

路由器转换具有输入端口和输出端口。输入端口位于输入组中，输出端口位于输出组中。您可以通过以下方式创建输入端口：从其他转换复制输入端口或者在端口选项卡中手动创建输入端口。

Designer 会通过从输入端口复制以下属性来创建输出端口：

- 端口名称

- 数据类型
- 精度
- 小数位数
- 默认值

对输入端口进行更改后，Designer 会更新输出口以反映这些更改。您不能编辑或删除输出口。输出口显示在路由器转换的普通视图中。

Designer 会根据输入端口名称创建输出口名称。对于每个输入端口，Designer 会在每个输出组中创建相应的输出口。

## 在映射中连接路由器转换

将转换连接到映射中的路由器转换时，请考虑以下规则：

- 可以将一个组连接到一个转换或目标。
- 可以将组中的一个输出口连接到多个转换或目标。
- 可以将一个组中的多个输出口连接到多个转换或目标。
- 无法将多个组连接到一个目标或单个输入组转换。
- 将每个输出组与不同输入组连接时，可以将多个组连接到多个输入组转换，但联接器转换除外。

## 创建路由器转换

要向映射添加路由器转换，请完成以下步骤：

1. 在 Mapping Designer 中，打开一个映射。
2. 单击“转换” > “创建”。  
选择路由器转换，并输入新转换的名称。路由器转换的命名约定为 `RTR_TransformationName`。单击“创建”，然后单击“完成”。
3. 选择并拖动转换中的所有端口，将它们添加到该路由器转换，也可以在“端口”选项卡上手动创建输入端口。
4. 双击该路由器转换的标题栏以编辑转换属性。
5. 单击“转换”选项卡并配置转换属性。
6. 单击“属性”选项卡并配置跟踪级别。
7. 单击“组”选项卡，然后单击“添加”按钮以创建用户定义的组。  
创建第一个用户定义的组后，Designer 将创建默认组。
8. 单击“组筛选条件”字段打开表达式编辑器。
9. 输入组筛选条件。
10. 单击“验证”检查所输入条件的语法。
11. 单击“确定”。
12. 将组输出口连接到转换或目标。

## 第 23 章

# 序列生成器转换

本章包括以下主题：

- [序列生成器转换概览, 328](#)
- [序列生成器端口, 329](#)
- [序列生成器转换属性, 334](#)
- [序列数据对象, 338](#)
- [创建序列生成器转换, 340](#)
- [创建序列生成器转换, 341](#)
- [常见问题, 342](#)
- [非本地环境下的序列生成器转换, 342](#)

## 序列生成器转换概览

序列生成器转换是一种可生成数值的被动转换。使用序列生成器转换可创建唯一主键值、替换缺少的主键或循环生成一系列有序数字。

序列生成器转换是一种连接转换。它包含两个输出端口，可将这两个端口连接到一个或多个转换。每当有行块进入一个连接转换时，集成服务就会生成一个序列号块。如果连接 CURRVAL，则集成服务会处理每个块中的一行。如果将 NEXTVAL 连接到另一转换的输入端口，则集成服务会生成一系列数字。如果将 CURRVAL 连接到另一转换的输入端口，则集成服务生成的 NEXTVAL 值会加上增量值。

序列生成器包含传递端口和输出端口。将 NEXTVAL 端口连接到其他转换的输入端口。映射运行时集成服务将增加序列。

可以创建序列生成器转换用于单个映射，也可以创建可重用序列生成器转换以用于多个映射。可重用序列生成器转换将保持使用序列生成器转换实例的每个映射中序列的完整性。

可以基于新序列或序列数据对象创建序列生成器转换。序列数据对象指创建和维护值序列的对象。

可以使序列生成器转换可重用，从而在多个映射中使用它。对一个目标执行多个加载时，可以重用序列生成器转换。

例如，如果存在一个大型输入文件并将其分成了三个并行运行的会话，则可以使用序列生成器转换生成主键值。如果使用不同的序列生成器转换，则集成服务可能会生成重复的键值。相反，对所有这三个会话使用可重用序列生成器转换可为每个目标行提供一个唯一值。



# 序列生成器端口

序列生成器转换具有以下两个输出端口：NEXTVAL 和 CURRVAL。您不能编辑或删除这些端口。同样，也不能向此转换添加端口。

序列生成器转换具有传递端口和一个输出端口 NEXTVAL。不能编辑或删除输出端口。

## 传递端口

可以向序列生成器转换中添加一个端口作为传递端口。传递端口是接收输入数据并将相同数据返回映射而不进行任何更改的输入和输出端口。

您必须至少将一个输入端口添加到转换中，并在将 NEXTVAL 和 CURRVAL 输出端口链接到目标之前将该端口与上游源或转换连接。要将传递端口添加至转换，请从映射中的上游源或转换中将某个端口拖动到序列生成器转换中。

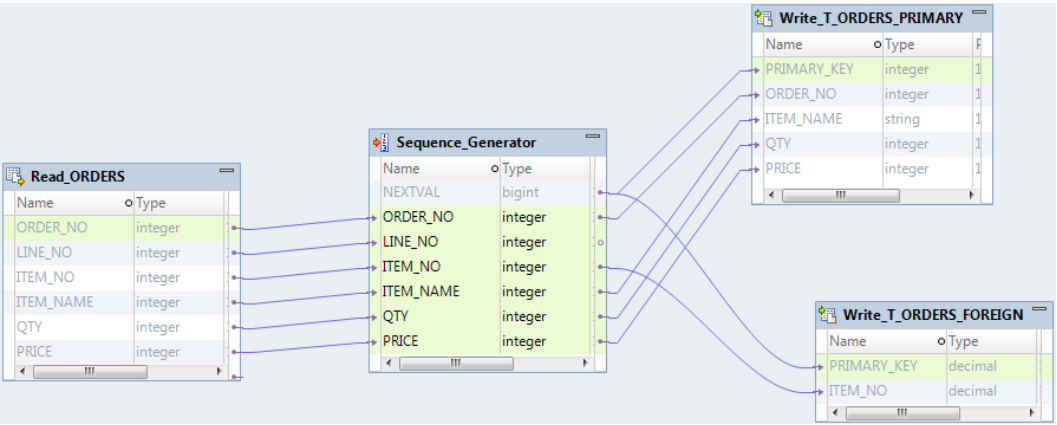
您必须至少将一个输入端口添加到转换中，并在将 NEXTVAL 输出端口链接到目标之前将该端口与上游源或转换连接。要将传递端口添加至转换，请从映射中的上游源或转换中将某个端口拖动到序列生成器转换中。

## NEXTVAL 端口

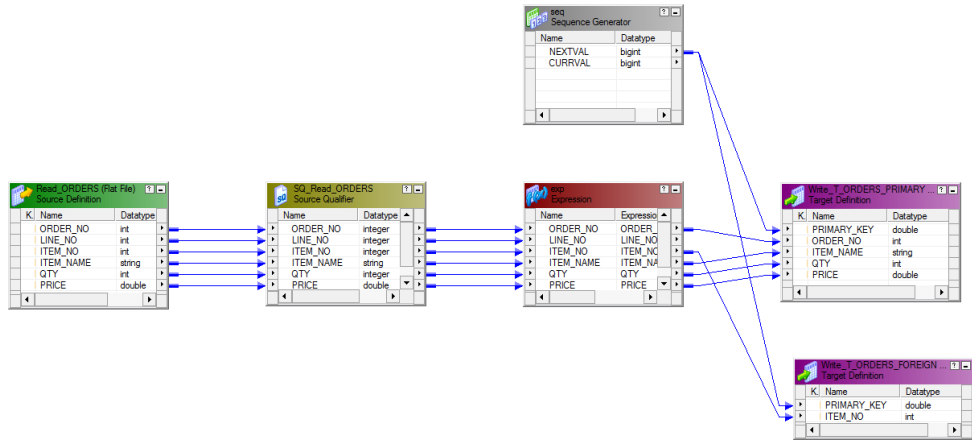
您可以将 NEXTVAL 连接到一个转换以针对该转换中的每行生成唯一值。将 NEXTVAL 端口连接到下游转换或目标以生成数字序列。如果将 NEXTVAL 连接到多个转换，集成服务将为每个转换都生成相同的数字序列。

连接 NEXTVAL 端口以基于“起始值”和“增量值”属性生成序列。如果未将序列生成器配置为在序列范围内循环，则 NEXTVAL 端口将一直生成序列号，直到配置的“结束值”。

下图显示了连接到一个源和两个目标的序列生成器转换 NEXTVAL 端口的映射：



下图显示了一个映射，该映射所包含的序列生成器转换 NEXTVAL 端口连接到两个目标以生成主键值和外键值：



对序列生成器转换进行如下配置时，集成服务将为 T\_ORDERS\_PRIMARY 和 T\_ORDERS\_FOREIGN 目标表生成相同主键值：起始值 = 1，增量值 = 1。

将 NEXTVAL 连接到多个转换以为每个转换中的每行生成唯一值。通过将 NEXTVAL 端口连接到下游转换或目标来生成序列号。连接 NEXTVAL 端口以基于“当前值”和“增量”属性生成序列。如果未将序列生成器配置为在序列范围内循环，则 NEXTVAL 端口将一直生成序列号，直到配置的“结束值”。

例如，您可以在一个映射中将 NEXTVAL 连接到两个目标以生成唯一主键值。集成服务会为每个目标表创建一个唯一主键值列。唯一主键值列将作为一个序列号块发送到一个目标表。第一个目标接收序列号块后，其他目标会从序列生成器转换中接收该序列号块。

例如，对序列生成器转换进行如下配置：当前值 = 1，增量 = 1。集成服务将为 T\_ORDERS\_PRIMARY 和 T\_ORDERS\_FOREIGN 目标表生成以下主键值：

**T\_ORDERS\_PRIMARY TABLE:  
PRIMARY KEY**

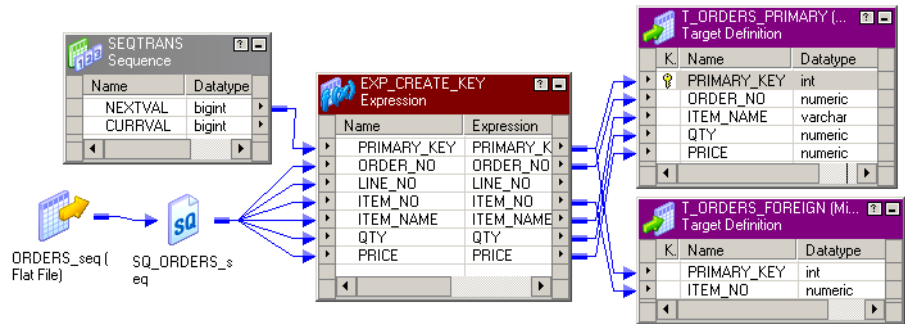
- 1
- 2
- 3
- 4
- 5

**T\_ORDERS\_FOREIGN TABLE:  
PRIMARY KEY**

- 6
- 7
- 8
- 9
- 10

如果希望多个从单个转换接收数据的目标具有相同值，可将序列生成器转换连接到之前的转换。集成服务将这些值处理成一个序列号块。从而允许集成服务将唯一值传递到转换，然后将行从转换中路由到目标。

下图显示的映射中含有一个将唯一值传递到表达式转换的序列生成器：



表达式转换使用相同的主键值填充这两个目标。

例如，对序列生成器转换进行如下配置：当前值 = 1，增量 = 1。集成服务将为 T\_ORDERS\_PRIMARY 和 T\_ORDERS\_FOREIGN 目标表生成以下主键值：

**T\_ORDERS\_PRIMARY TABLE:**  
**PRIMARY KEY**

1  
2  
3  
4  
5

**T\_ORDERS\_FOREIGN TABLE:**  
**PRIMARY KEY**

1  
2  
3  
4  
5

**注意：**在网络上运行经过分区的会话时，序列生成器转换将根据每个分区的行数跳过某些值。

## 创建键

可以通过将 NEXTVAL 端口连接到目标或下游转换来使用序列生成器转换创建主键值或外键值。可以使用的值必须在 1 到 9,223,372,036,854,775,807 范围之内，最小间隔为 1。

可以通过将 NEXTVAL 端口连接到目标或下游转换来使用序列生成器转换创建主键值或外键值。可以使用的值必须在 1 到 9,223,372,036,854,775,807 范围之内，最小间隔为 1。此外，还可以创建复合键来标识表中各行。

创建主键或外键时，可以使用“循环”选项防止集成服务创建重复的主键。为此，可以选择会话属性中的“截断目标表”选项或创建复合键。

要创建复合键，可以配置集成服务，使其在一组更少的值之间进行循环。例如，假定有三个商店生成订单号，则可以配置序列生成器转换，使其在值 1 到 3 之间循环，增量为 1。将 ORDER\_NO 端口连接到序列生成器转换时，生成的值将创建唯一的复合键。

以下示例显示了复合键和订单号：

**COMPOSITE\_KEY**

1  
2  
3

**ORDER\_NO**

12345  
12345  
12345

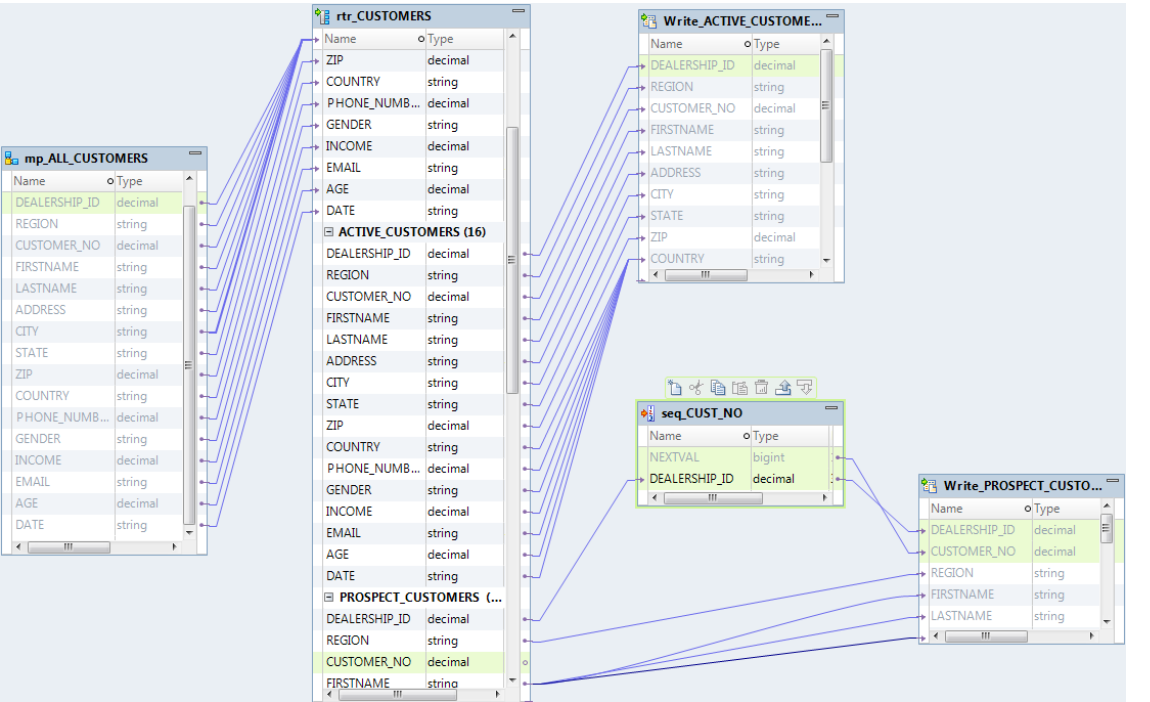
COMPOSITE_KEY	ORDER_NO
1	12346
2	12346
3	12346

### 替换缺少的值

在使用序列生成器转换替换缺少的键时，可以同时使用路由器转换从已分配值的列中筛选出空值。可以将路由器转换连接到序列生成器转换，并使用 NEXTVAL 生成一个数值序列来填充空值。

例如，要替换 CUSTOMER\_NO 列中的空值，可使用包含客户数据的源创建一个映射。可以添加一个路由器转换，以筛选出从具有空值的列中分配了客户编号的客户。可以添加一个序列生成器转换，以生成唯一的 CUSTOMER\_NO 值。可以添加要写入数据的客户目标。

下图显示了一个映射，该映射将替换 CUSTOMER\_NO 列中的空值：



使用序列生成器转换，通过在 IIF 和 ISNULL 函数中使用 NEXTVAL 来替换缺少的键。

例如，要替换 ORDER\_NO 列中的空值，可创建一个具有这些属性的序列生成器转换，并将 NEXTVAL 端口拖动至表达式转换。在表达式转换中，将 ORDER\_NO 端口与任何其他所需端口一起拖动到转换中。然后创建一个输出端口 ALL\_ORDERS。

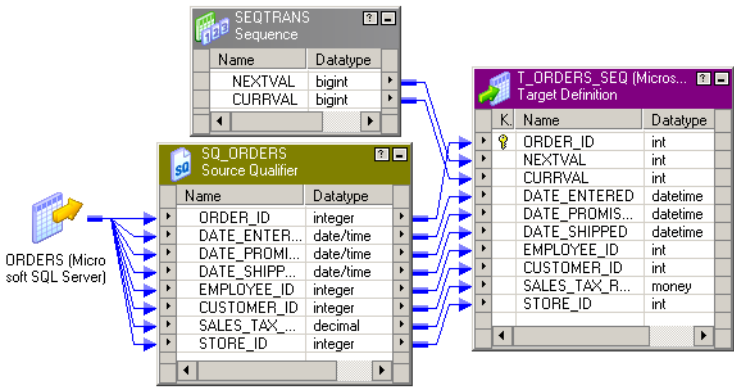
在 ALL\_ORDERS 中，可输入以下表达式来替换空值订单：

```
IIF( ISNULL( ORDER_NO ), NEXTVAL, ORDER_NO )
```

## CURRVAL

CURRVAL 等于 NEXTVAL 加上“增量值”。通常，如果 NEXTVAL 端口已连接到下游转换，则只需连接 CURRVAL 端口。如果某行输入了连接到 CURRVAL 端口的转换，则集成服务传递的值为上次创建的 NEXTVAL 值加一。

下图显示了如何将 CURRVAL 和 NEXTVAL 端口连接到目标：



例如，对序列生成器转换进行如下配置：当前值 = 1，增量 = 1。集成服务将为 NEXTVAL 和 CURRVAL 生成以下值：

**NEXTVAL**

1  
2  
3  
4  
5

**CURRVAL**

2  
3  
4  
5  
6

如果连接 CURRVAL 端口，而不连接 NEXTVAL 端口，则集成服务将为每一行传递一个常量值。如果连接序列生成器转换中的 CURRVAL 端口，则集成服务将在每个块中处理一行。可以仅连接映射中的 NEXTVAL 端口以优化性能。

**注意：**在网络上运行经过分区的会话时，序列生成器转换可能会根据每个分区的行数跳过某些值。

# 序列生成器转换属性

配置集成服务用来生成连续值的转换属性。

下表列出了可为序列数据对象和新序列配置的属性：

属性	说明
起始值	在使用“循环”选项时希望集成服务使用的生成序列的起始值。如果选择了“循环”，则集成服务在达到结束值时会回到此值。 默认值为 0。 最大值为 9,223,372,036,854,775,806。
结束值	集成服务所生成的最大值。如果集成服务在会话期间达到此值且未将序列配置为循环，则会话将失败。 最大值为 9,223,372,036,854,775,807。
增量值	NEXTVAL 端口中两个相邻值之间的差值。 默认值为 1。 必须为正整数。 最大值为 2,147,483,647。
循环	如果启用，则集成服务会从起始值重新开始序列范围中进行循环。 如果禁用，则集成服务会在配置的结束值处停止序列。如果集成服务在达到结束值后仍有要处理的行，则会话会失败并出现溢出错误。

以下列表介绍了可配置的序列生成器转换属性：

## 起始值

在使用“循环”选项时希望集成服务使用的生成序列的起始值。如果选择了“循环”，则集成服务在达到结束值时会回到此值。

默认值为 0。

最大值为 9,223,372,036,854,775,806。

## 增量

NEXTVAL 端口中两个相邻值之间的差值。

默认值为 1。

必须为正整数。

最大值为 2,147,483,647。

## 结束值

集成服务所生成的最大值。如果集成服务在会话期间达到此值且未将序列配置为循环，则会话将失败。

最大值为 9,223,372,036,854,775,807。

如果将 NEXTVAL 端口连接到下游整数端口，应将“结束值”设置为一个不大于最大整数值的值。如果 NEXTVAL 超过下游端口的数据类型最大值，会话将失败。

## 当前值

序列的当前值。输入希望集成服务用作序列中第一个值的值。要在一系列值间循环，则该值必须大于或等于起始值且小于结束值。

如果将“缓存值数”设置为 0，则集成服务会更新当前值以反映上次为会话生成的值加 1 后的结果，然后使用更新后的当前值作为下次运行此会话的基础。但是，如果使用了“重置”选项，则集成服务会在每个会话结束后将此值重置为其原始值。

**注意:** 如果编辑了此设置，即是将序列重置为新设置。如果将“当前值”重置为 10，且增量为 1，则下次使用此会话时，集成服务生成的第一个值将为 10。

最大值为 9,223,372,036,854,775,806。如果当前值超出了最大值，则集成服务会将此值设置为空值。

## 循环

如果启用，则集成服务会从起始值重新开始序列范围中进行循环。

如果禁用，则集成服务会在配置的结束值处停止序列。如果集成服务在达到结束值后仍有要处理的行，则会话会失败并出现溢出错误。

## 缓存值数

集成服务一次缓存的序列值数。当多个会话同时使用同一可重用序列生成器时，请使用此选项以确保每个会话收到唯一值。集成服务在缓存每个值时会更新存储库。如果设置为 0，则集成服务不会缓存值。

默认值为 0。

可重用序列生成器的默认值为 1,000。

最大值为 9,223,372,036,854,775,807。

此属性仅适用于可重用序列生成器转换。

## 重置

如果启用，则集成服务将基于每个会话的原始当前值生成值。如果禁用，集成服务会更新当前值以反映上次为会话生成的值加 1 后的结果，然后使用更新后的当前值作为下次会话运行的基础。

此属性仅适用于不可重用序列生成器转换。

## 跟踪级别

集成服务写入会话日志中的转换的相关详细信息的级别。

# 起始值

使用“循环”可生成重复的序列，例如对应于一年中月份的数字 1-12。

1. 输入序列中希望集成服务用于“起始值”的最低值。
2. 输入要用作“结束值”的最高值。
3. 选择“循环”。

循环时，如果集成服务达到为序列配置的结束值，其将返回并再次启动循环（从配置的起始值开始）。

# 增量

集成服务根据序列生成器转换中的“当前值”属性和“增量”属性，在 NEXTVAL 端口中生成一个序列。

“当前值”属性是集成服务开始为每个会话创建序列的值。“增量”属性是集成服务添加到现有值以创建序列中的新值的整数。默认情况下，“当前值”设置为 1，“增量”设置为 1。

例如，可以创建当前值为 1,000、增量为 10 的序列生成器转换。如果要通过映射来传递三行，则集成服务将生成以下值集：

```
1000
1010
1020
```

## 结束值

“结束值”是您希望集成服务生成的最大值。如果集成服务达到结束值，且序列生成器未配置为在序列中循环，则会话将失败并显示一个溢出错误。

结束值是您希望集成服务生成的最大值。如果集成服务达到结束值，且序列生成器未配置为在序列中循环，则映射运行将失败并显示一个溢出错误。

可以将结束值设置为 1 到 9,233,372,036,854,775,807 之间的任何整数。如果将 NEXTVAL 端口连接到下游整数端口，应将“结束值”设置为一个不大于最大整数值。例如，如果将 NEXTVAL 端口连接到小整数端口，设置结束值时最大值为 32,767。如果 NEXTVAL 超过下游端口的数据类型最大值，会话将失败。

可以将结束值设置为 1 到 9,233,372,036,854,775,807 之间的任何整数。如果将 NEXTVAL 端口连接到下游整数端口，应将“结束值”设置为一个不大于最大整数值。例如，如果将 NEXTVAL 端口连接到小整数端口，设置结束值时最大值为 32,767。如果 NEXTVAL 超过下游端口的数据类型最大值，映射将失败。

## 增量值

集成服务根据序列生成器转换中的“当前值”属性和“增量”属性，在 NEXTVAL 端口中生成一个序列。

“当前值”属性是集成服务开始为每个会话创建序列的值。“增量”属性是集成服务添加到现有值以创建序列中的新值的整数。默认情况下，“当前值”设置为 1，“增量”设置为 1。

例如，可以创建当前值为 1,000、增量为 10 的序列生成器转换。如果要通过映射来传递三行，则集成服务将生成以下值集：

```
1000
1010
1020
```

## 在值范围内循环

可以为序列生成器转换建立一个值范围。如果使用循环选项，序列生成器转换会在达到结束值时重复该范围。

例如，如果您设置一个从 10 开始到 50 结束的序列范围，并设置增量值 10，则序列生成器转换会创建值 10、20、30、40、50。该序列将从 10 重新开始。

## 当前值

集成服务会基于当前值为每个会话生成值。要指示集成服务在首次使用序列生成器转换时应使用的值，必须将该值作为当前值输入。如果要通过序列生成器转换来循环使用一系列值，则当前值必须大于等于起始值并小于结束值。

集成服务会基于当前值为每个映射运行生成值。集成服务始终会在序列生成器转换中使用起始值作为当前值。

在每个会话结束时，如果序列生成器的“缓存值数”为 0，则集成服务会将当前值更新成为该会话生成的最后一个值加一。例如，如果集成服务在结束会话时为该会话生成的值为 101，则该服务会在存储库中将序列生成器的当前值更新为 102。下次使用序列生成器时，集成服务将使用 102 作为下一个生成值的基础。如果序列生成器的增量为 1，则在集成服务使用该序列生成器开始另一个会话时，第一个生成值为 102。

在每个映射运行结束时，如果序列生成器的“缓存值数”为 0，则集成服务会将当前值更新成为该会话生成的最后一个值加一。例如，如果集成服务在结束会话时为该会话生成的值为 101，则该服务会在存储库中将序列生成器的当前值更新为 102。下次使用序列生成器时，集成服务将使用 102 作为下一个生成值的基础。如果序列生成器的增量为 1，则在集成服务使用该序列生成器开始另一个会话时，第一个生成值为 102。

如果序列生成器转换的版本不止一个，则集成服务会在运行会话时更新所有版本的当前值。无论您签出的是序列生成器转换还是父映射，集成服务都会更新各个版本的当前值。如果为序列生成器转换更新的当前值与已编辑的当前值不同，则更新的当前值将替代已编辑的当前值。



例如，用户 1 创建了一个序列生成器转换并签入该转换，然后为序列生成器版本 1 保存当前值 10。然后用户 1 签出该序列生成器转换，并为序列生成器版本 2 输入新的当前值 100。用户 1 保持序列生成器转换处于签出状态。与此同时，用户 2 运行一个会话，该会话使用序列生成器转换版本 1。当用户 2 运行该会话时，集成服务将使用签入值 10 作为当前值。该会话完成后，当前值为 150。尽管用户 1 已签出序列生成器转换，集成服务也会将序列生成器转换版本 1 和版本 2 的当前值更新为 150。

如果在运行会话后打开映射，则当前值将显示为该会话生成的最后一个值加一。由于集成服务使用当前值来确定每个会话的第一个值，因此，只有在希望重置序列时，才应编辑当前值。

如果序列生成器转换的版本不止一个，则要重置序列，必须在修改当前值后签入该映射或可重用序列生成器转换。

**注意:** 如果将序列生成器配置为“重置”，则集成服务将基于当前值为每个会话生成第一个值。

## 缓存值数

“缓存值数”可确定集成服务一次可缓存的值数量。如果“缓存值数”大于零，则集成服务会缓存所配置数量的值，并在每次缓存值时更新当前值。

如果多个会话同时使用同一个可重用序列生成器转换，则该序列生成器转换可能会有多个实例。要避免为每个会话生成相同的值，可通过配置“缓存值数”为每个会话保留一个序列值范围。

**提示:** 要提高在网格上运行会话时的性能，可增加序列生成器转换的缓存值数。这样可减少主 DTM 进程和执行工作的 DTM 进程与存储库之间所需的通信量。

## 不可重用的序列生成器

默认情况下，对于不可重用的序列生成器转换，“缓存值数”将设置为零，即集成服务不会在会话期间缓存值。如果集成服务未缓存值，则它会在会话开始时访问存储库以获取当前值。然后，集成服务将为序列生成值。会话结束时，集成服务会更新存储库中的当前值。

如果将“缓存值数”设置为大于零，则集成服务将在会话期间缓存值。会话开始时，集成服务将访问存储库以获取当前值、缓存所配置数量的值并相应地更新当前值。如果集成服务使用缓存中的所有值，则它将访问存储库以获取下一组值并更新当前值。会话结束时，集成服务会放弃缓存中所有剩余的值。

对于不可重用的序列生成器转换，将“缓存值数”设置为大于零可以增加集成服务在会话期间访问存储库的次数。此设置还会显示已跳过值部分，因为在每个会话结束时将放弃未使用的缓存值。

例如，对序列生成器转换进行如下配置：缓存值数 = 50，当前值 = 1，增量 = 1。集成服务启动会话时，它将为该会话缓存 50 个值，并在存储库中将当前值更新为 50。集成服务将对会话使用值 1 至 39，并放弃未使用的值 40 至 49。集成服务再次运行会话时，它将检查存储库以获取当前值 (50)。然后，集成服务将缓存接下来的 50 个值并将当前值更新为 100。在会话期间，它将使用值 50 至 98。为两个会话生成的值分别为 1 至 39 和 50 至 98。

## 可重用序列生成器

如果某个可重用序列生成器转换存在于多个会话中，且这些会话同时运行，请使用“缓存值数”，以确保每个会话在同一序列中接收的值是唯一的。默认情况下，可重用序列生成器的“缓存值数”设置为 1000。

如果多个会话同时使用同一序列生成器转换，则会存在为每个会话生成相同值的风险。为避免此风险，请通过配置“缓存值数”使集成服务为每个会话缓存一组值数。

例如，请按如下方式配置可重用序列生成器转换：缓存值数 = 50，当前值 = 1，增量 = 1。两个会话使用该序列生成器，且计划使这两个会话在几乎相同的时间运行。集成服务启动第一个会话时，将为此会话缓存 50 个值，并在存储库中将当前值更新为 50。集成服务开始在会话中使用 1 到 50 之间的值。当集成服务启动第二个会话时，将检查存储库来获得当前值，该值为 50。然后，集成服务将缓存接下来的 50 个值并将当前值更新为 100。随后，集成服务开始在第二个会话中使用 51 到 100 之间的值。当两个会话均已使用了所有缓存值，集成服务将缓存一组新值，并更新当前值，以保证这些值对于序列生成器仍是唯一的。

对于可重用序列生成器转换，可以通过减少“缓存值数”来将已丢弃值降到最低，但“缓存值数”必须大于 1。减少“缓存值数”时，可能要增加集成服务在会话期间访问存储库以缓存值的次数。

## 重置

如果为不可重用的序列生成器转换选择了“重置”，则集成服务会基于其每次启动会话的原始起始值生成值。否则，集成服务会更新当前值以反映上次生成的值加增量值后的结果，然后在下次使用该序列生成器转换时使用更新后的值。

如果将不可重用序列生成器转换配置为使用重置属性，则集成服务会将原始起始值用于每次映射运行。否则，集成服务将对当前值递增并在下次映射运行时使用该值。

例如，配置一个序列生成器转换以创建从 1 到 1,000 的值，其增量为 1。选择“重置”以将起始值重置为 1。在第一个会话运行期间，集成服务将生成 1 到 234 的数字。以后每次运行映射时，集成服务都会从初始值 1 开始重新生成数字。

例如，配置一个序列生成器转换以创建从 1 到 1,000 的值，其增量为 1，起始值为 1，并选择“重置”。在第一个映射运行期间，集成服务将生成 1 到 234 的数字。以后每次运行映射时，集成服务都会从起始值 1 开始重新生成数字。

如果没有重置，则集成服务会在第一次运行结束时将当前值更新为 235。下次使用该序列生成器转换时，生成的第一个值为 235。

**注意：**对于可重用的序列生成器转换，将禁用“重置”。

## 保持行顺序

保持转换输入数据的行顺序。如果集成服务不应执行任何可能改变行顺序的优化，请选择此选项。

集成服务执行优化时，可能会失去之前在映射中建立的顺序。您可以在具有已排序的平面文件源、已排序的关系源或排序器转换的映射中设置顺序。当您转换配置为保持行顺序时，集成服务将在执行映射优化时考虑此配置。如果集成服务可以保持顺序，将为转换执行优化。如果优化会改变行顺序，集成服务将不会为转换执行优化。

# 序列数据对象

序列数据对象创建和维护数字值序列。序列生成器转换使用序列数据对象为转换生成值。

可以在多个序列生成器转换中使用可重用序列数据对象。如果使用同一个序列数据对象的所有序列生成器转换在相同集成服务中运行，则它们将使用相同的值序列。也可以在不可重用序列生成器转换中使用可重用序列数据对象。可以在不可重用序列生成器转换中使用不可重用序列数据对象。

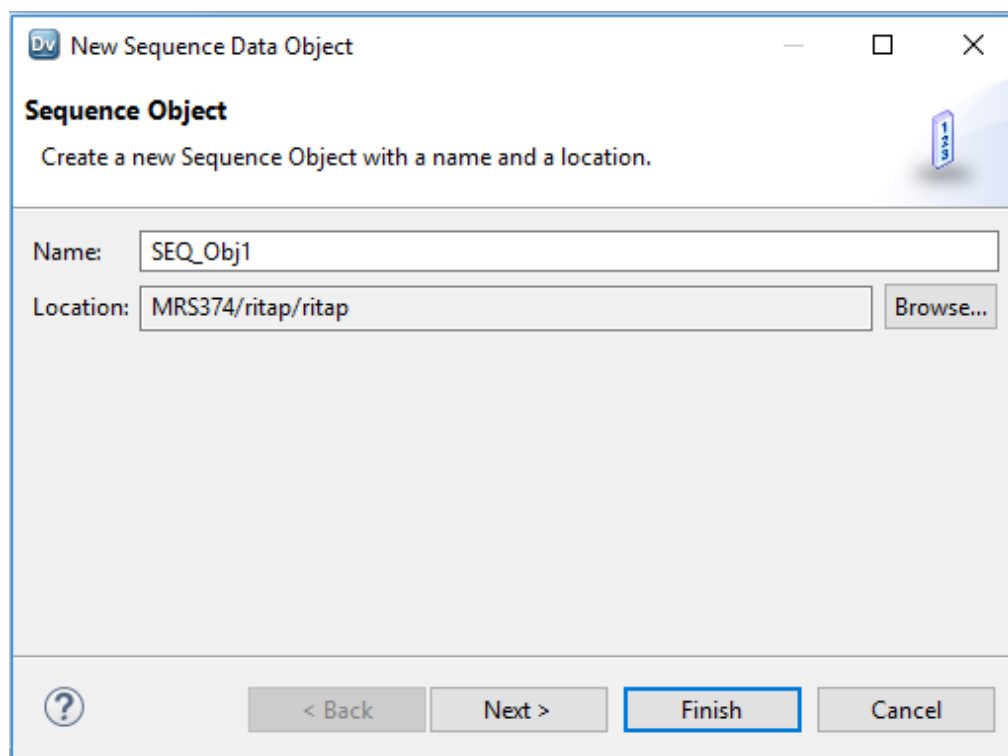
例如，创建多个在关系表中写入相同主键字段的映射。每个映射使用相同的可重用序列生成器转换，而可重用序列生成器转换使用同一个可重用序列数据对象并在相同的集成服务中运行。每个映射将向主键字段写入唯一值。

## 创建序列数据对象

要使用序列数据对象创建序列生成器转换，请创建序列数据对象，配置对象属性，然后在“序列生成器转换”对话框中选择对象。

1. 在映射编辑器中，在映射选项板中向下滚动以找到该序列生成器转换，然后将其拖动到映射中。  
此时将打开**新建转换向导**。

- 单击**新建序列数据对象**。  
此时将打开**新建数据对象向导**。



**New Sequence Data Object**

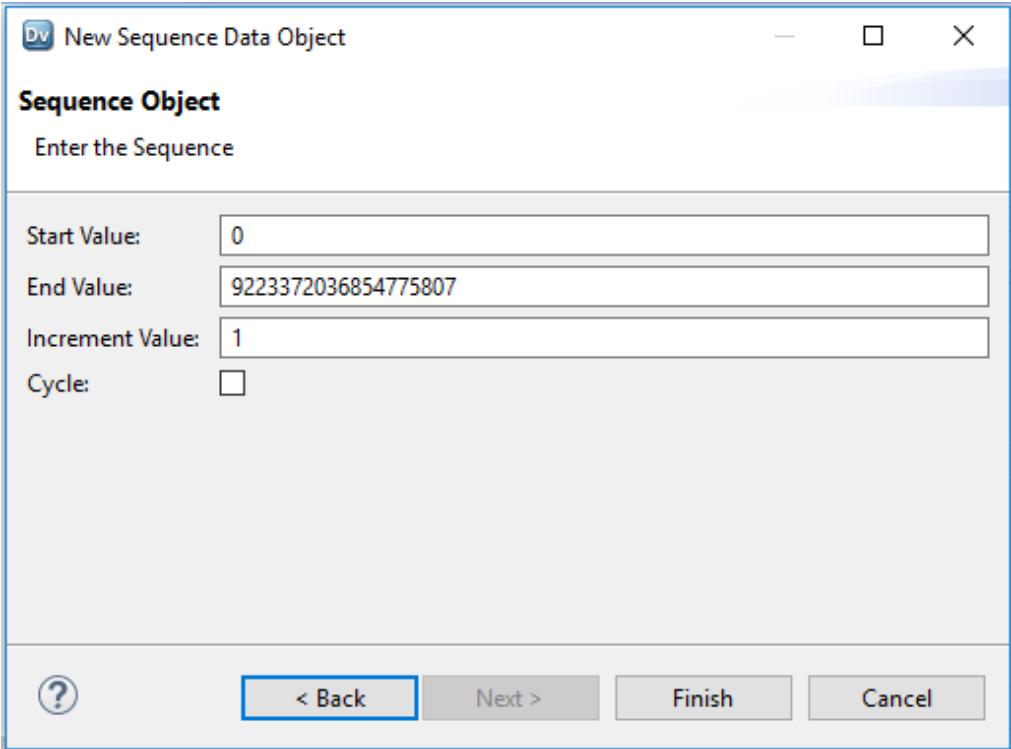
**Sequence Object**  
Create a new Sequence Object with a name and a location.

Name: SEQ\_Obj1

Location: MRS374/ritap/ritap

- 输入序列数据对象的名称。  
序列数据对象的命名约定是 SEQ\_<数据对象名称>。
- 单击**下一步**以配置序列数据对象属性。

如果从对象创建序列生成器转换，转换将使用您为该数据对象输入的属性。下图显示了可以配置的属性：



5. 配置数据对象属性后，可以使用该序列数据对象创建序列生成器转换。创建转换时，请为序列生成器转换命名，然后单击**选择现有序列对象**。导航到该数据对象，然后单击**确定**。

该序列生成器转换将显示在具有 NEXTVAL 仅输出端口的映射编辑器中。可以将 NEXTVAL 端口连接到下游转换或目标来生成数字序列。

## 创建序列生成器转换

要在映射中使用序列生成器转换，请将其添加到映射中并配置转换属性，然后将 NEXTVAL 或 CURRVAL 连接到一个或多个转换。

要创建序列生成器转换，请执行以下操作：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择“序列生成器转换”。

序列生成器转换的命名规则为 `SEQ_TransformationName`。

2. 输入“序列生成器”的名称，然后单击“创建”。单击“完成”。

Designer 将创建序列生成器转换。

3. 双击转换的标题栏。

4. 输入转换的说明。

5. 选择“属性”选项卡。输入设置。

**注意：**不能在会话级替代序列生成器转换属性。这样做可以保护所生成序列值的完整性。

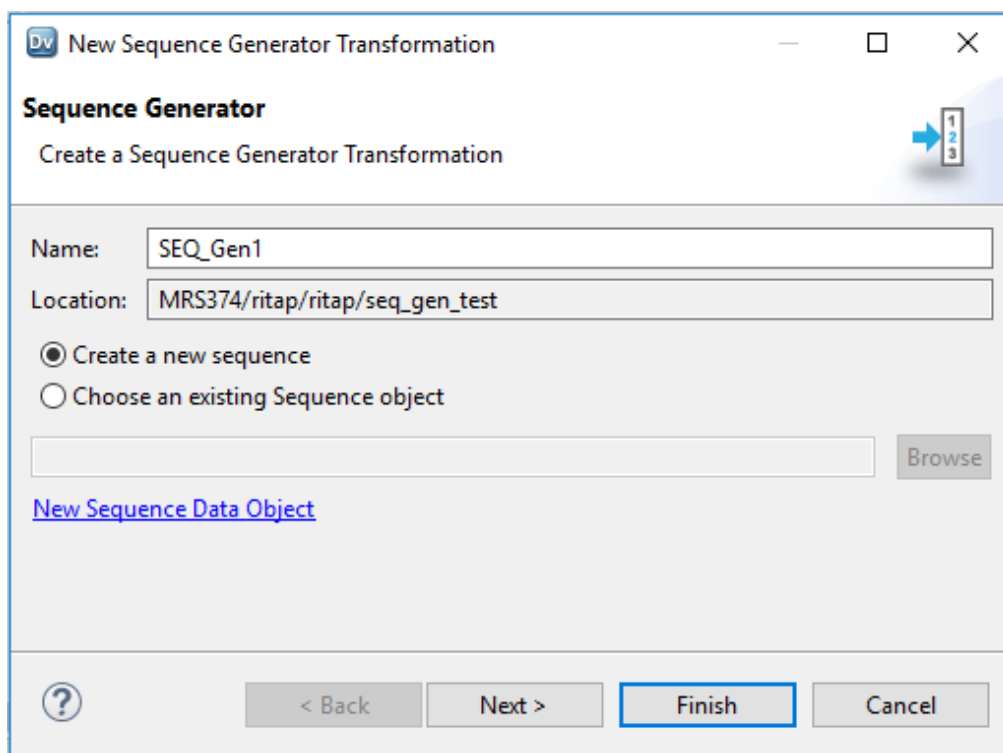
6. 单击“确定”。

7. 要在会话过程中生成新的序列，请将 NEXTVAL 端口至少连接到映射中的一个转换。  
在其他转换中，使用表达式中的 NEXTVAL 或 CURRVAL 端口。

## 创建序列生成器转换

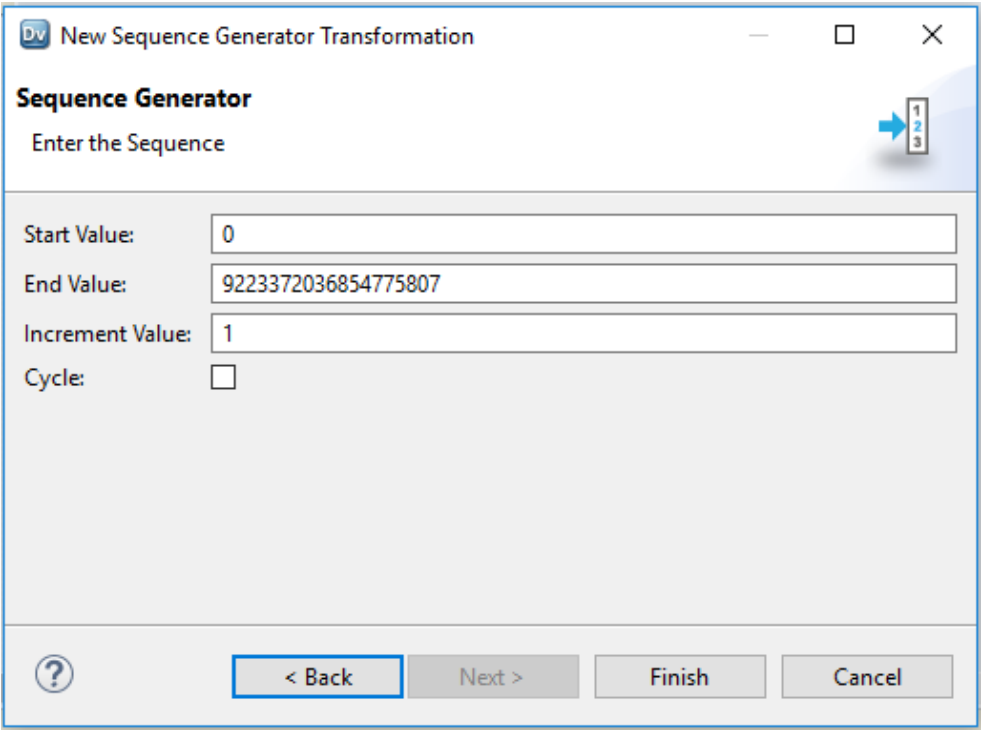
要在映射中使用序列生成器转换，请将其添加到映射中并配置转换属性，然后将 NEXTVAL 连接到一个或多个转换。

1. 在映射编辑器中，在映射选项板中向下滚动以找到该序列生成器转换，然后将其拖动到映射中。  
此时将打开**新建转换**向导。



2. 输入序列生成器转换的名称。  
序列生成器转换的命名约定是 SEQ\_<转换名称>。
3. 选择要创建新序列还是使用现有序列对象。

- 要创建新序列，请选择**创建新序列**。单击**下一步**以配置序列属性。下图显示了可以配置的属性：



- 要使用现有序列对象，请单击**选择现有序列对象**。导航到该序列对象，然后单击**确定**。

该序列生成器转换将显示在具有 NEXTVAL 仅输出端口的映射编辑器中。可以将 NEXTVAL 端口连接到下游转换或目标来生成数字序列。

# 常见问题

我是否可以更改不可重用序列生成器转换以将其设置为可重用？

您无法使该转换重新可用，但可以将该转换改为使用序列数据对象。无论有多少转换在使用序列数据对象，该对象都可以保持序列的完整性。

我是否可以在 Mapplet 中放入一个不可重用序列生成器转换？

不，您不能。Mapplet 是可重用对象，因此其中的所有对象也必须是可重用对象。可改用可重用序列生成器转换。

# 非本地环境下的序列生成器转换

非本地环境下的序列生成器转换处理取决于运行此转换的引擎。

请注意以下非本地运行时引擎的支持情况：

- Blaze 引擎。有限支持。
- Spark 引擎。在批处理映射中有限支持。在流映射中不受支持。

- Databricks Spark 引擎。不受支持。

## Blaze 引擎上的序列生成器转换

当以下条件成立时，包含序列生成器转换的映射会消耗大量资源：

- 在转换中将**维持行顺序**属性设置为 *true*。
- 映射在单个分区上运行。

## Spark 引擎上的序列生成器转换

序列生成器转换不会在输出数据中维持行顺序。如果在转换中启用了**维持行顺序**属性，数据集成服务将会忽略此属性。

## 第 24 章

# 排序器转换

本章包括以下主题：

- [排序器转换概览, 344](#)
- [数据排序, 344](#)
- [排序器转换属性, 345](#)
- [创建排序器转换, 347](#)

## 排序器转换概览

可以使用排序器转换对数据排序。您可以根据指定排序键按升序或降序对数据进行排序。还可以配置排序器转换，以便进行区分大小写的排序，以及指定输出行是否应该相异。排序器转换是一种主动转换。必须将其连接到数据流。

可以对关系源或平面文件源中的数据排序。还可以使用排序器转换，对通过汇总器转换（已配置为使用排序输入）进行的数据传递进行排序。

在映射中创建排序器转换时，可以指定一个或多个端口作为排序键，并配置每个排序键端口，以便按升序或降序排序。还可以配置集成服务应用于所有排序键端口及其分配的系统资源的排序条件，以执行排序操作。

## 数据排序

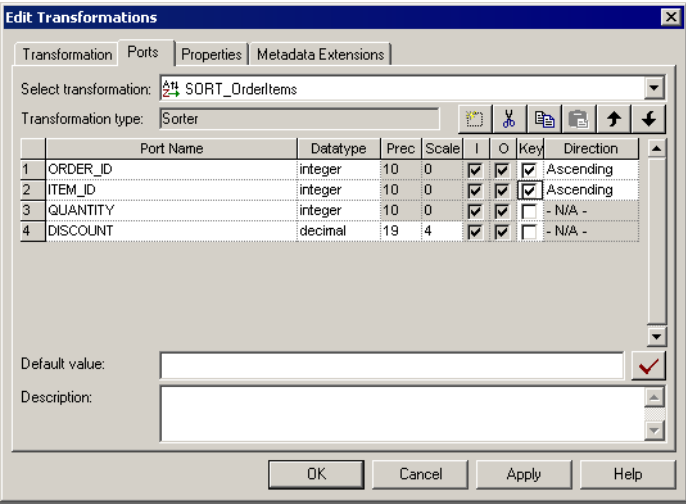
排序器转换仅包含输入/输出端口。通过排序器转换进行的所有数据传递，都将根据排序键排序。排序键是指希望用作排序条件的一个或多个端口。

可以指定一个以上的端口作为排序键的组成部分。在为排序键指定多个端口后，集成服务将按顺序对每个端口排序。这些端口在“端口”选项卡中的显示顺序，将确定排序操作的顺序。排序器转换会将通过各个顺序排序键端口进行的数据传递视为上一个端口的第二次排序。

在会话运行时，集成服务将按照在会话属性中指定的排序顺序，对数据排序。排序顺序将确定特殊字符和符号的排序条件。



下图显示了“端口”选项卡配置，该配置适用于排序器转换按订单 ID 和项 ID 对数据进行升序排序：



在会话运行时，集成服务会将以下行传递到排序器转换中：

ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
45	123456	3	3.04
45	456789	2	12.02
43	000246	6	34.55
41	000468	5	.56

在对数据排序之后，集成服务会将以下行传递到排序器转换以外：

ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
41	000468	5	.56
43	000246	6	34.55
45	123456	3	3.04
45	456789	2	12.02

## 排序器转换属性

排序器转换包含多项属性，用于指定其他排序条件。集成服务会将这些条件应用于所有排序键端口。排序器转换属性还可确定集成服务在对数据排序时分配的系统资源。

### 排序器缓存大小

集成服务将使用“排序器缓存大小”属性来确定可为执行排序操作分配的最大内存量。集成服务会在执行排序操作之前，将所有传入数据传递到排序器转换中。可以对缓存使用数值，可以使用参数文件中的缓存值，也可以将

集成服务配置为使用“自动”设置来设置缓存大小。如果配置集成服务来确定缓存大小，也可以为集成服务配置可分配给缓存的最大内存量。如果配置的会话缓存总大小为 2 GB (2,147,483,648 字节) 或更大，则必须在 64 位集成服务上运行会话。

在开始执行排序操作之前，集成服务会分配为排序器缓存大小配置的内存量。如果集成服务运行分区会话，它将为每个分区分配指定的排序器缓存内存量。

如果集成服务无法分配足够的内存，则其会话也将失败。为了实现最佳性能，请将排序器缓存大小的值配置为小于或等于集成服务计算机上的可用物理 RAM 量。要使用排序器转换对数据排序，至少需要分配 16 MB (16,777,216 字节) 的物理内存。默认情况下，排序器缓存大小设置为 16,777,216 字节。

如果传入数据量大于排序器缓存大小量，集成服务会将数据暂时存储在排序器转换工作目录中。对工作目录中的数据进行排序时，集成服务所需的磁盘空间至少为传入数据量的两倍。如果传入数据量远大于排序器缓存大小，集成服务所需的磁盘空间可能不止是工作目录可用的磁盘空间量的两倍。

在将排序器转换跟踪级别配置为“普通”后，集成服务还会写入排序器转换用于会话日志的内存量。

## 区分大小写

“区分大小写”属性可确定集成服务在对数据排序时是否考虑大小写。如果启用了“区分大小写”属性，则集成服务会将大写字符排在小写字符之前。

## 工作目录

必须指定集成服务用于在对数据排序时创建临时文件的工作目录。在集成服务对数据排序之后，它会删除临时文件。可以指定集成服务计算机上的任意目录作为工作目录。默认情况下，集成服务将使用为 \$PMTempDir 进程变量指定的值。

对包含排序器转换的会话进行分区时，可以为管道中的每个分区指定不同的工作目录。要提高会话性能，请在集成服务系统中物理独立的磁盘上指定工作目录。

## 相异输出

可以配置排序器转换，以将输出视为相异。如果将排序器转换配置为生成相异的输出，则 Mapping Designer 会将所有端口配置为排序键的一部分。集成服务会丢弃在排序操作期间比较出的重复行。

## 跟踪级别

配置排序器转换跟踪级别，以控制集成服务写入到会话日志中的排序器错误和状态消息的数量及类型。在“普通”跟踪级别，集成服务会写入传递到排序器转换的行的数量，以及排序器转换为排序操作分配的内存量。集成服务还会写入将第一个和最后一个输入行传递到排序器转换的时间和日期。

如果将排序器转换跟踪级别配置为“详细数据”，则集成服务将写入排序器转换完成将所有数据传递到管道中下一个转换的时间。此外，集成服务还会向会话日志中写入排序器转换释放内存资源以及从工作目录中删除临时文件的时间。

## 空值视为低值

您可以配置排序器转换对待空值的方式。如果希望集成服务在执行排序操作时将空值视为比其他任何值都低，请启用此属性。如果希望集成服务将空值视为比其他任何值都高，请禁用此属性。

## 转换范围

转换范围将指定集成服务如何将转换逻辑应用于传入数据：

- **事务**。将转换逻辑应用至事务中的所有行。当一行数据取决于同一事务中的所有行，但不取决于其他事务中的行时，请选择“事务”。
- **全部输入**。将转换逻辑应用于所有传入数据。选择“全部输入”时，PowerCenter 将删除传入的事务边界。当一行数据取决于源中的所有行时，请选择“全部输入”。

## 创建排序器转换

要向映射添加排序器转换，请完成以下步骤：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择排序器转换。  
排序器转换的命名规则为 `SRT_TransformationName`。输入转换的说明。此说明显示在 Repository Manager 中，帮助理解转换所执行的操作。
2. 输入排序器的名称，然后单击“创建”。  
此时 Designer 将创建排序器转换。
3. 单击“完成”。
4. 将要排序的端口拖到排序器转换中。  
Designer 将为您添加的每个端口创建输入/输出端口。
5. 双击转换的标题栏，可打开“编辑转换”对话框。
6. 选择“端口”选项卡。
7. 选择要用作排序键的端口。
8. 对于作为排序键一部分选择的每个端口，指定希望集成服务按升序还是降序对数据进行排序。
9. 选择“属性”选项卡。修改排序器转换属性。
10. 选择“元数据扩展”选项卡。创建或编辑排序器转换的元数据扩展。
11. 单击“确定”。

## 第 25 章

# 源限定符转换

本章包括以下主题：

- [源限定符转换概览, 348](#)
- [源限定符转换属性, 350](#)
- [默认查询, 351](#)
- [联接源数据, 352](#)
- [添加 SQL 查询, 354](#)
- [输入用户定义的联接, 355](#)
- [外部联接支持, 355](#)
- [输入源筛选器, 361](#)
- [使用排序端口, 362](#)
- [选择相异, 363](#)
- [添加会话前和会话后 SQL 命令, 363](#)
- [创建源限定符转换, 364](#)
- [源限定符转换故障排除, 365](#)

## 源限定符转换概览

当您向映射中添加关系源定义或平面文件源定义时，需要将其连接到源限定符转换。源限定符转换表示集成服务在运行会话时读取的行。源限定符转换是一种活动转换。

使用源限定符转换完成以下任务：

- **联接源自同一源数据库的数据。** 可以通过将源链接到一个源限定符转换来联接具有主键-外键关系的两个或更多表。
- **集成服务读取源数据时将筛选行。** 如果包含筛选条件，则集成服务会向默认查询中添加 WHERE 子句。
- **指定外部联接替代默认内部联接。** 如果包含用户定义的联接，则集成服务会替换 SQL 查询中的元数据所指定的联接信息。
- **指定已排序端口数。** 如果指定已排序端口数，则集成服务会在默认 SQL 查询中添加 ORDER BY 子句。
- **仅从源中选择相异值。** 如果选择“选择相异”，则集成服务会向默认 SQL 查询中添加 SELECT DISTINCT 语句。
- **为集成服务创建发出特殊 SELECT 语句的自定义查询以读取源数据。** 例如，可以使用自定义查询来执行汇总计算。

## 转换数据类型

源限定符转换会显示转换数据类型。转换数据类型确定集成服务读取数据时源数据库如何绑定该数据。请不要更改源限定符转换中的数据类型。如果源定义与源限定符转换中的数据类型不匹配，则 Designer 会在您保存映射时将其标记为无效。

## 目标加载顺序

根据映射中的源限定符转换指定目标加载顺序。如果有连接到多个目标的多个源限定符转换，则可以指定集成服务将数据加载到目标的顺序。

如果一个源限定符转换为多个目标提供数据，则可以在会话中启用基于约束的加载，以让集成服务根据目标表主键和外键的关系加载数据。

例如，映射包含三个不同的管道。每个管道都包含一个单独的源、源限定符、转换以及在一个映射中链接在一起的平面文件目标。您可以为映射配置目标加载顺序，以便集成服务按顺序处理每个目标加载顺序组。

## 日期时间值

当您在 SQL 查询中使用日期时间值或日期时间参数或变量时，请将日期格式更改为源中使用的格式。在 SQL 查询中，集成服务将以字符串形式将日期时间值传递到源系统。集成服务会根据源数据库将日期时间值转换为字符串。

下表介绍了每种数据库类型的日期时间格式：

源	日期格式
DB2	YYYY-MM-DD-HH24:MI:SS
Informix	YYYY-MM-DD HH24:MI:SS
Microsoft SQL Server	MM/DD/YYYY HH24:MI:SS
ODBC	YYYY-MM-DD HH24:MI:SS
Oracle	MM/DD/YYYY HH24:MI:SS
Sybase	MM/DD/YYYY HH24:MI:SS
Teradata	YYYY-MM-DD HH24:MI:SS

某些数据库要求您使用其他标点符号（如单引号）或数据库特定函数标识日期时间值。例如，要转换 Oracle 源中的 \$\$\$SessStartTime 值，在 SQL 替代中使用以下 Oracle 函数：

```
to_date ( '$$$SessStartTime' , 'mm/dd/yyyy hh24:mi:ss' )
```

对于 Informix，在 SQL 替代中使用以下 Informix 函数来转换 \$\$\$SessStartTime 值：

```
DATETIME ( $$$SessStartTime ) YEAR TO SECOND
```

关于数据库特定函数的信息，请参见数据库文档。

## 参数和变量

在 SQL 查询、用户定义的联接、源筛选器及源限定符转换的会话前和会话后 SQL 命令中，可以使用参数和变量。可使用您在参数文件中可定义的任何参数或变量类型。可以在 SQL 语句内输入参数或变量，也可以使用参数

或变量作为 SQL 查询。例如，您可以使用会话参数 \$ParamMyQuery 作为 SQL 查询，并在参数文件中将 \$ParamMyQuery 设置为 SQL 语句。

集成服务首先会生成一个 SQL 查询，并展开每个参数或变量。它将使用其起始值替代每个映射参数、映射变量和工作流变量。然后针对源数据库运行该查询。

使用源限定符转换中的字符串映射参数或变量时，请使用适用于源系统的字符串标识符。大多数数据库都使用单引号作为字符串标识符。例如，要将源筛选器中的字符串参数 \$\$IPAddress 用于 Microsoft SQL Server 数据库表，请为该参数加上单引号，如下所示： '\$\$IPAddress' 。

使用日期时间映射参数或变量或使用内置变量 \$\$SessStartTime 时，请将日期格式更改为源中使用的格式。在 SQL 查询中，集成服务将以字符串形式将日期时间值传递到源系统。

**提示:** 要确保日期时间参数或变量的格式与源中使用的格式匹配，请验证 SQL 查询。

## 源限定符转换属性

在“属性”选项卡上配置以下源限定符转换属性：

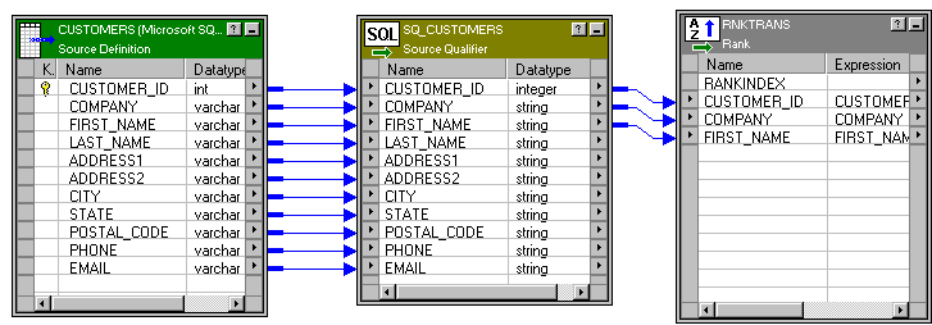
选项	说明
SQL 查询	定义自定义查询，替换集成服务用于从该源限定符转换中表示的源中读取数据的默认查询。自定义查询可替代自定义联接或源筛选器的项目。
用户定义的联接	指定用于从同一源限定符转换中表示的多个源中联接数据的条件。
源筛选器	指定查询行时集成服务应用的筛选条件。
已排序端口数	指示在对从关系源中查询的行排序时使用的列数。如果选择该选项，集成服务会在读取源行时将 ORDER BY 加入到默认查询。ORDER BY 包含指定的端口数（从转换顶部开始）。如果已选定该选项，则数据库排序顺序必须与会话排序顺序匹配。
跟踪级别	设置运行包含此转换的会话时会话日志中包含的详细信息量。
选择相异	指定是否要只选择唯一的行。如果选择该选项，则集成服务会包含 SELECT DISTINCT 语句。
Pre-SQL	要在集成服务读取源之前对照源数据库运行的会话前 SQL 命令。
Post-SQL	要在集成服务写入到目标之后对照源数据库运行的会话后 SQL 命令。
输出具有确定性	关系源或转换输出，如果输入数据在两次会话运行之间保持一致则不会更改。当配置该属性时，如果管道中的转换始终生成可重复数据，则集成服务不会暂存源数据进行恢复。
输出是可重复的	关系源或转换输出，当输入数据的顺序一致时，其顺序会在两次会话运行间保持一致。如果输出具有确定性且可重复，则集成服务不会为恢复暂存源数据。

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

# 默认查询

对于关系源，集成服务将在运行会话时针对每个源限定符转换生成查询。默认查询为映射中使用的每个源列的 SELECT 语句。换句话说，集成服务只读取已连接到其他转换的列。

下图显示了一个连接到源限定符转换的源定义，其中包含许多源定义列，但只有三个连接到了最终转换：



虽然该源定义中有许多列，但只有三个列连接到了其他转换。这种情况下，集成服务生成的默认查询将仅选择这三个列：

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME
FROM CUSTOMERS
```

如果任何表名称或列名称包含数据库预留字，则可以创建并维护一个包含预留字的文件 reswords.txt。在集成服务初始化会话时，它将在集成服务安装目录中搜索 reswords.txt。如果该文件存在，集成服务将在对数据库执行 SQL 时将匹配的预留字加上引号。如果要替代 SQL，则必须将所有预留字都加上引号。

生成默认查询时，Designer 将使用双引号分隔包含以下字符的表和字段名称：

```
/ + - = ~ ` ! % ^ & * ( ) [ ] { } ' ; ? , < > \ | <space>
```

## 查看默认查询

可以在源限定符转换中查看默认查询。

要查看默认查询，请执行以下操作：

- 1. 从“属性”选项卡中，选择“SQL 查询”。
- SQL 编辑器会显示集成服务用于选择源数据的默认查询。
- 2. 单击“生成 SQL”。
- 3. 单击“取消”退出。

如果不取消 SQL 查询，集成服务会使用自定义 SQL 查询替代默认查询。

请勿连接到源数据库。在输入替代默认查询的 SQL 查询时，只能连接到源数据库。

必须先将源限定符转换中的列连接到其他转换或目标，然后才能生成默认查询。

## 替代默认查询

通过更改转换属性的默认设置可以更改或替代源限定符转换中的默认查询。请勿更改选定端口的列表或它们在查询中的顺序。此列表必须与已连接的转换输出端口匹配。

编辑转换属性时，源限定符转换将在默认查询中包含这些设置。但如果输入 SQL 查询，集成服务将仅使用定义的 SQL 语句。SQL 查询将替代源限定符转换中的“用户定义的联接”、“源筛选器”、“已排序端口数”和“选择相异”设置。

**注意:** 替代默认 SQL 查询时, 必须将所有数据库预留字都加上引号。

# 联接源数据

使用一个源限定符转换可联接多个关系表中的数据。这些表必须可从同一个实例或数据库服务器访问。

在映射使用相关的关系源时, 可以在一个源限定符转换中联接两个源。在会话期间, 源数据库将在传递数据到集成服务前执行联接。为源表建立索引后, 这样可提高性能。

**提示:** 对于异构源及要联接平面文件, 请使用联接器转换。

## 默认联接

在一个源限定符转换中联接相关表时, 集成服务将基于每个表中的相关键联接表。

此默认联接是内部等值联接, 在 WHERE 子句中使用以下语法:

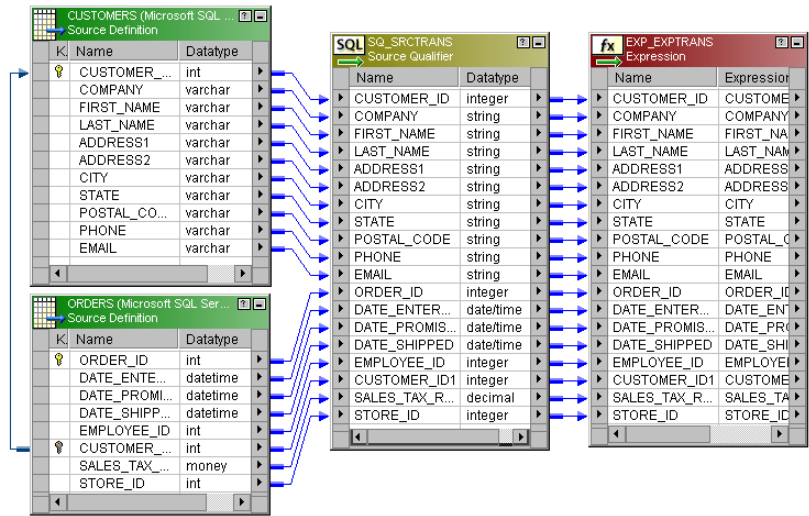
Source1.column\_name = Source2.column\_name

默认联接中的列必须具有:

- 主键-外键关系
- 匹配的数据类型

例如, 您可以查看月份的所有订单, 包括订单号、订购量和客户名称。ORDERS 表包括每个订单的订单号和订购量, 但没有客户名称。要包含客户名称, 需要联接 ORDERS 和 CUSTOMERS 表。两个表都包括客户 ID, 所以可以在一个源限定符转换中联接两个表。

下图显示了使用一个源限定符转换联接两个表的方式:



包括多个表时, 集成服务将为映射中使用的所有列生成一个 SELECT 语句。这种情况下, SELECT 语句看起来与以下语句类似:

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME, CUSTOMERS.LAST_NAME,
CUSTOMERS.ADDRESS1, CUSTOMERS.ADDRESS2, CUSTOMERS.CITY, CUSTOMERS.STATE, CUSTOMERS.POSTAL_CODE,
CUSTOMERS.PHONE, CUSTOMERS.EMAIL, ORDERS.ORDER_ID, ORDERS.DATE_ENTERED, ORDERS.DATE_PROMISED,
ORDERS.DATE_SHIPPED, ORDERS.EMPLOYEE_ID, ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE, ORDERS.STORE_ID
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```



WHERE 子句是等值联接，其中包括 ORDERS 和 CUSTOMER 表中的 CUSTOMER\_ID。

## 自定义联接

如果需要替代默认联接，您可以输入在自定义查询中指定联接的 WHERE 子句的内容。如果查询执行外部联接，数据集成服务会根据数据库语法在 WHERE 子句或 FROM 子句中插入联接语法。

在以下情况下，可能需要替代默认联接：

- 列没有主键-外键关系。
- 用于联接的列的数据类型不匹配。
- 要指定不同类型的联接（如外部联接）。

## 异构联接

要执行异构联接，请使用联接器转换。需要联接以下类型的源时，请使用联接器转换：

- 联接不同源数据库中的数据
- 联接不同平面文件系统中的数据
- 联接关系源和平面文件

## 创建键关系

如果表具有主键-外键关系，则可以在源限定符转换中联接表。不过，可在 Source Analyzer 中通过链接不同表中的匹配列创建主键-外键关系。这些列不必一定是键，但应包含在每个表的索引中。

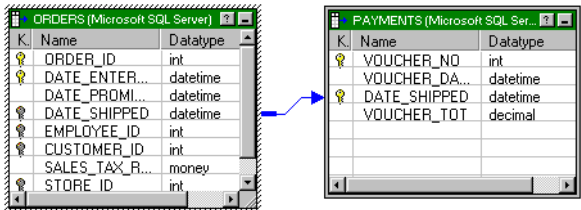
**提示：**如果源表包含超过 1,000 行，可以通过检索主键-外键提高性能。如果源表包含不到 1,000 行，检索主键-外键可能会降低性能。

例如，一家零售连锁店的公司办公室希望基于订单提取收到的付款。ORDERS 和 PAYMENTS 表没有共同的主键和外键。不过，两个表都包含 DATE\_SHIPPED 列。可以在 Source Analyzer 的元数据中创建主键-外键关系。

请注意，这两个表没有链接在一起。因此，Designer 不会识别 DATE\_SHIPPED 列中的关系。

通过链接 DATE\_SHIPPED 列在 ORDERS 和 PAYMENTS 表之间创建关系。Designer 将向 ORDERS 和 PAYMENTS 表定义的 DATE\_SHIPPED 列中添加主键和外键。

下图显示了两个表之间的 DATE\_SHIPPED 关系：



如果没有连接这些列，Designer 则无法识别这些关系。

主键-外键关系仅存在于元数据中。无需生成 SQL 或更改源表。

建立键关系后，使用源限定符转换联接两个表。默认基于 DATE\_SHIPPED 创建联接。

## 添加 SQL 查询

源限定符转换提供可替代默认查询的 SQL 查询选项。可以输入源数据库支持的 SQL 语句。输入查询之前，请连接要在映射中使用的所有输入和输出口。

编辑 SQL 查询时，可以生成和编辑默认查询。生成默认查询时，Designer 可以合并所有其他配置的选项，例如，筛选器或已排序端口数。生成的查询将替代后期您可能在转换中配置的所有其他选项。

可以将参数或变量用作 SQL 查询，也可以在查询中包含参数和变量。添加字符串映射参数或变量时，请使用对源系统适用的字符串标识符。对于大部分数据库，您需要将字符串参数或变量的名称放在单引号中。

在 SQL 查询中添加日期时间值或日期时间映射参数或变量时，请更改日期格式，以与源系统使用的格式匹配。集成服务将根据源系统将日期时间值转换为字符串。

输入自定义 SQL 查询时，请遵循以下规则和准则：

- SELECT 语句必须按照端口名称在转换中显示的顺序列出这些端口名称。
- 如果源系统为 Microsoft SQL Server，查询中 SELECT 语句的列数必须与源限定符转换中的端口数匹配。否则，会话可能会失败，并出现以下错误：SQL 错误 [FnName:提取优化 -- [Informatica][ODBC SQL Server 连线协议驱动程序] 绑定列的数量超出结果列的数量。]。

替代配置用于下推优化的会话默认 SQL 查询时，集成服务将创建视图来表示 SQL 替代。然后，集成服务将对此视图运行 SQL 查询，以将转换逻辑推送至数据库。

如果编辑 SQL 查询，必须将所有数据库预留字放在引号中。

1. 打开源限定符转换，单击“属性”选项卡。

2. 单击“SQL 查询”字段中的“打开”按钮。

此时将显示“SQL 编辑器”对话框。

3. 单击“生成 SQL”。

查询来自源限定符转换所包含任何源中的行时，Designer 将显示生成的默认查询。

4. 在显示默认查询的区域输入查询。

任何列名称均必须由显示该列名称的表、视图或同义词的名称限定。例如，如果要包含 ORDERS 表中的 ORDER\_ID 列，可输入 ORDERS.ORDER\_ID。可以双击“端口”窗口中显示的列名称，不必输入每个列的名称。

可以将参数或变量用作查询，也可以在查询中包含参数和变量。

将字符串映射参数和变量放在字符串标识符内。必要时，可更改日期时间映射参数的日期格式和变量。

5. 选择包含查询中所含源的 ODBC 数据源。

6. 输入用于连接此数据库的用户名和密码。

“使用 Kerberos 身份验证”选项表示连接中的数据库在使用 Kerberos 身份验证的网络上运行。如果选择此选项，则无法输入用户名和密码。连接将使用登录运行 Designer 的计算机的用户帐户的凭据。

7. 单击“验证”。

Designer 将运行查询，并报告查询语法是否正确。

8. 单击“确定”可返回“编辑转换”对话框。再次单击“确定”可返回 Designer。

**Tip:** 可以调整表达式编辑器的大小。通过从边界拖动可以展开对话框。Designer 会将对话框的新大小保存为一个客户端设置。

# 输入用户定义的联接

输入用户定义的联接与输入自定义 SQL 查询类似。不过，您只能输入 WHERE 子句而不是整个查询的内容。执行外部联接时，集成服务可能插入查询 WHERE 子句或 FROM 子句中的联接语法，具体视数据库语法而异。

添加用户定义的联接时，源限定符转换将包括默认 SQL 查询中的设置。不过，如果在添加用户定义的联接后修改默认查询，集成服务将仅使用源限定符转换“SQL 查询”属性中定义的查询。

您可以使用参数或变量作为用户定义的联接，或者在联接内加入参数或变量。添加字符串映射参数或变量时，请使用对源系统适用的字符串标识符。对于大部分数据库，您需要将字符串参数或变量的名称放在单引号中。

加入日期时间参数或变量时，您可能需要更改日期格式以与源中使用的格式匹配。集成服务基于源系统将日期时间参数和变量转换为字符串。

要创建用户定义的联接，请执行以下操作：

1. 创建一个包含多个源或关联源中数据的源限定符转换。
2. 打开源限定符转换，单击“属性”选项卡。
3. 单击“用户定义的联接”字段中的“打开”按钮。

此时将显示“SQL 编辑器”对话框。

4. 输入联接的语法。

不要在联接的开头输入关键字 WHERE。查询行时，集成服务将添加此关键字。

将字符串映射参数和变量放在字符串标识符内。必要时，可更改日期时间映射参数的日期格式和变量。

5. 单击“确定”返回到“编辑转换”对话框，然后单击“确定”返回到 Designer。

## 外部联接支持

使用源限定符转换和应用程序源限定符转换可对同一个数据库中的两个源执行外部联接。当集成服务执行外部联接时，将返回来自一个源表的所有行以及第二个源表中符合联接条件的行。

如果要联接两个表并返回其中一个表的所有行，请使用外部联接。例如，如果要将注册客户表与月度采购表联接以确定注册客户的活动，可执行外部联接。使用外部联接，可以将注册客户表与月度采购表联接，并返回注册客户表中的所有行（包括上月未进行采购的客户）。如果执行普通联接，集成服务将只返回在当月进行了采购的注册客户，以及仅由注册客户进行的采购。

使用外部联接时，可以生成与联接器转换中的主外部联接或详细外部联接相同的结果。但使用外部联接可减少数据流中的行数。这样可以提高性能。

集成服务支持两种外部联接：

- **左联接。**集成服务在联接语法左侧返回表的所有行以及两个表中符合联接条件的行。
- **右联接。**集成服务在联接语法右侧返回表的所有行以及两个表中符合联接条件的行

**注意：**替代默认查询时，在嵌套查询语句中使用外部联接。

## Informatica 联接语法

输入联接语法时，使用 Informatica 或数据库特定的联接语法。使用 Informatica 联接语法时，集成服务将在会话期间对语法进行转换，并将其传递给源数据库。

**注意：**对联接条件始终使用数据库特定的语法。

使用 Informatica 联接语法时，将整个联接语句括在括号内 ({Informatica syntax})。使用数据库语法时，输入源数据库支持的语法（不带括号）。

使用 Informatica 联接语法时，将表名称用作列名称的前缀。例如，如果 REG\_CUSTOMER 表中有一个列的名称为 FIRST\_NAME，则在联接语法中输入“REG\_CUSTOMER.FIRST\_NAME”。此外，为表名称使用别名时，请在 Informatica 联接语法内使用别名，以确保集成服务能够识别该别名。

下表列出了创建外部联接时，您可在不同源限定符转换的不同位置中输入的联接语法：

转换	转换设置	说明
源限定符转换	用户定义的联接	创建联接替代。集成服务将该联接替代附加到默认查询的 WHERE 或 FROM 子句。
源限定符转换	SQL 查询	在默认查询中紧靠 WHERE 之后输入联接语法。
应用程序源限定符转换	联接替代	创建联接替代。集成服务将该联接替代附加到默认查询的 WHERE 子句。
应用程序源限定符转换	提取替代	在默认查询中紧靠 WHERE 之后输入联接语法。

可以在单个源限定符内结合使用左外部联接和右外部联接与正常联接。使用多个正常联接和多个左外部联接。

结合联接时，按照以下顺序输入它们：

1. 普通联接
2. 左外部联接
3. 右外部联接

**注意：**一些数据库将限制为仅使用一个右外部联接。

### 普通联接语法

可以使用源限定符中的联接条件创建普通联接。但是，如果要创建外部联接，则需要替代默认联接以执行外部联接。因此，需要在联接替代中包含普通联接。在联接替代中合并普通联接时，将普通联接列在外部联接的前面。可以在联接替代中输入多个普通联接。

要创建正常联接，请使用以下语法：

```
{ source1 INNER JOIN source2 on join_condition }
```

下表显示了联接替代中普通联接的语法：

语法	说明
<i>source1</i>	源表名称。集成服务将返回此表中与联接条件匹配的行。
<i>source2</i>	源表名称。集成服务将返回此表中与联接条件匹配的行。
<i>join_condition</i>	联接的条件。使用源数据库支持的语法。可以用 AND 运算符将多个联接条件组合。

例如，您有一个包含注册客户数据的 REG\_CUSTOMER 表：

CUST_ID	FIRST_NAME	LAST_NAME
00001	Marvin	Chi
00002	Dinah	Jones

CUST_ID	FIRST_NAME	LAST_NAME
00003	John	Bowden
00004	J.	Marks

PURCHASES 表每月进行刷新，其包含以下数据：

TRANSACTION_NO	CUST_ID	DATE	AMOUNT
06-2000-0001	00002	6/3/2000	55.79
06-2000-0002	00002	6/10/2000	104.45
06-2000-0003	00001	6/10/2000	255.56
06-2000-0004	00004	6/15/2000	534.95
06-2000-0005	00002	6/21/2000	98.65
06-2000-0006	NULL	6/23/2000	155.65
06-2000-0007	NULL	6/24/2000	325.45

要返回显示 6 月份各项交易的客户名的行，请使用以下语法：

```
{ REG_CUSTOMER INNER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

集成服务将返回以下数据：

CUST_ID	DATE	AMOUNT	FIRST_NAME	LAST_NAME
00002	6/3/2000	55.79	Dinah	Jones
00002	6/10/2000	104.45	Dinah	Jones
00001	6/10/2000	255.56	Marvin	Chi
00004	6/15/2000	534.95	J.	Marks
00002	6/21/2000	98.65	Dinah	Jones

集成服务将返回具有匹配客户 ID 的行。不包含未在 6 月份进行采购的客户。也不包含非注册用户进行的采购。

## 左外部联接语法

可以使用联接替代创建左外部联接。可以在一个联接替代中输入多个左外部联接。将左外部联接与其他联接结合使用时，请将所有左外部联接一起列在语句中所有正常联接的后面。

要创建左外部联接，请使用以下语法：

```
{ source1 LEFT OUTER JOIN source2 on join_condition }
```

下表显示了联接替代中左外部联接的语法：

语法	说明
<i>source1</i>	源表名称。使用左外部联接，数据集成服务将返回此表中的所有行。
<i>source2</i>	源表名称。集成服务将返回此表中与联接条件匹配的行。
<i>join_condition</i>	联接的条件。使用源数据库支持的语法。可以用 AND 运算符将多个联接条件组合。

例如，利用“[普通联接语法](#)”[页面上 356](#) 所述的相同的 REG\_CUSTOMER 和 PURCHASES 表，您可以通过以下联接替代确定在 6 月份有多少客户进行了采购：

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

集成服务将返回以下数据：

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
00002	Dinah	Jones	6/3/2000	55.79
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95
00002	Dinah	Jones	6/10/2000	104.45
00002	Dinah	Jones	6/21/2000	98.65

集成服务将返回 REG\_CUSTOMERS 表中的所有已注册客户（对在 6 月份未进行采购的客户使用空值）。不包含非注册用户进行的采购。

使用多个联接条件确定在 6 月份有多少注册客户单笔采购额在 \$100.00 以上：

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on (REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID AND PURCHASES.AMOUNT > 100.00) }
```

集成服务将返回以下数据：

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
00002	Dinah	Jones	6/10/2000	104.45
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95

如果要合并同一时段的利润的相关信息，可以使用多个左外部联接。例如，RETURNS 表包含以下数据：

CUST_ID	CUST_ID	RETURN
00002	6/10/2000	55.79
00002	6/21/2000	104.45

要确定进行采购的客户数以及 6 月份的利润，请使用两个左外部联接：

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID LEFT OUTER JOIN
  RETURNS on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

集成服务将返回以下数据：

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT	RET_DATE	RETURN
00001	Marvin	Chi	6/10/2000	255.56	NULL	NULL
00002	Dinah	Jones	6/3/2000	55.79	NULL	NULL
00003	John	Bowden	NULL	NULL	NULL	NULL
00004	J.	Marks	6/15/2000	534.95	NULL	NULL
00002	Dinah	Jones	6/10/2000	104.45	NULL	NULL
00002	Dinah	Jones	6/21/2000	98.65	NULL	NULL
00002	Dinah	Jones	NULL	NULL	6/10/2000	55.79
00002	Dinah	Jones	NULL	NULL	6/21/2000	104.45

集成服务对缺少的值使用空值。

### 右外部联接语法

可以使用联接替代创建右侧外部联接。如果在联接语法中反转表的顺序，则右侧外部联接返回的结果与左侧外部联接相同。在一个联接替代中仅使用一个右侧外部联接。如果要创建多个右侧外部联接，请尝试反转源表的顺序，并将联接类型更改为左侧外部联接。

将右外部联接与其他联接结合使用时，请将右外部联接输入到联接替代的末尾。

要创建右外部联接，请使用以下语法：

```
{ source1 RIGHT OUTER JOIN source2 on join_condition }
```

下表显示了联接替代中右外部联接的语法：

语法	说明
<i>source1</i>	源表名称。集成服务将返回此表中与联接条件匹配的行。
<i>source2</i>	源表名称。使用右侧外部联接时，集成服务将返回该表中的所有行。
<i>join_condition</i>	联接的条件。使用源数据库支持的语法。可以用 AND 运算符将多个联接条件组合。

可以与左侧外部联接配合使用右侧外部联接，以从两个表中联接并返回所有数据，模拟完整外部联接。例如，可以使用以下联接替代提取六月份的所有已注册客户和所有购买：

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID RIGHT OUTER JOIN
  PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

集成服务将返回以下数据：

CUST_ID	FIRST_NAME	LAST_NAME	TRANSACTION_NO	DATE	AMOUNT
00001	Marvin	Chi	06-2000-0003	6/10/2000	255.56

CUST_ID	FIRST_NAME	LAST_NAME	TRANSACTION_NO	DATE	AMOUNT
00002	Dinah	Jones	06-2000-0001	6/3/2000	55.79
00003	John	Bowden	NULL	NULL	NULL
00004	J.	Marks	06-2000-0004	6/15/2000	534.95
00002	Dinah	Jones	06-2000-0002	6/10/2000	104.45
00002	Dinah	Jones	06-2000-0005	6/21/2000	98.65
NULL	NULL	NULL	06-2000-0006	6/23/2000	155.65
NULL	NULL	NULL	06-2000-0007	6/24/2000	325.45

## 创建外部联接

您可以输入外部联接作为联接替代或作为默认查询替代的一部分。

创建联接替代时，Designer 会将联接替代附加到默认查询的 WHERE 子句。在会话过程中，集成服务会转换 Informatica 联接语法，并将其包含在用于提取源数据的默认查询中。在可能的情况下，输入联接替代而不是替代默认查询。

替代默认查询时，在默认查询的 WHERE 子句中输入联接语法。在会话期间，集成服务会转换 Informatica 联接语法，然后使用查询提取源数据。如果在创建替代之后更改转换，集成服务会忽略更改。因此，在可能的情况下，输入外部联接语法作为联接替代。

### 创建外部联接作为联接替代

要创建外部联接作为联接替代，请执行以下操作：

1. 打开源限定符转换，单击“属性”选项卡。
2. 在源限定符转换中，单击“用户定义的联接”字段中的按钮。  
在应用程序源限定符转换中，单击“联接替代”字段中的按钮。
3. 输入联接的语法。  
不要在联接的开头输入 WHERE。集成服务会在查询行时添加此内容。  
将 Informatica 联接语法括在大括号 ({} ) 内。  
在表和 Informatica 联接语法中使用别名时，请使用 Informatica 联接语法内部的别名。  
使用表名称作为列名称的前缀，例如“table.column”。  
使用源数据库支持的联接条件。  
输入多个联接时，请按类型对联接分组，然后按以下顺序列出联接：普通联接、左外部联接、右外部联接。  
每个嵌套查询中仅包含一个右外部联接。  
从“端口”选项卡中选择端口名称，以保证准确性。
4. 单击“确定”。

### 创建外部联接作为提取替代

创建外部联接作为提取替代：

1. 连接应用程序源限定符转换的输入和输出端口之后，双击转换标题栏，并选择“属性”选项卡。
2. 在应用程序源限定符转换中，单击“提取替代”字段中的按钮。



3. 单击“生成 SQL”。
4. 输入 WHERE 子句之后立即在 WHERE 子句中输入联接语法。  
将 Informatica 联接语法括在大括号 ({} ) 内。  
在表和 Informatica 联接语法中使用别名时，请使用 Informatica 联接语法内部的别名。  
使用表名称作为列名称的前缀，例如“table.column”。  
使用源数据库支持的联接条件。  
输入多个联接时，请按类型对联接分组，然后按以下顺序列出联接：普通联接、左外部联接、右外部联接。  
每个嵌套查询中仅包含一个右外部联接。  
从“端口”选项卡中选择端口名称，以保证准确性。
5. 单击“确定”。

## 常见数据库语法限制

不同数据库的外部联接语法有不同的限制。创建外部联接时，请遵循以下限制：

- 不要在外部联接语法的 ON 子句中使用 OR 运算符组合联接条件。
- 不要在外部联接语法的 ON 子句中使用 IN 运算符比较列。
- 不要在外部联接语法的 ON 子句中比较列与子查询。
- 组合两个或多个外部联接时，不要使用同一个表作为多个外部联接的内部表。例如，不要使用以下任一外部联接：

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA TABLE3 LEFT OUTER JOIN TABLE2 ON  
TABLE3.COLUMNB = TABLE2.COLUMNB }  
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA TABLE2 RIGHT OUTER JOIN TABLE3 ON  
TABLE2.COLUMNB = TABLE3.COLUMNB }
```

- 不要在普通联接条件中使用外部联接的两个表。例如，不要使用以下联接条件：

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB =  
TABLE2.COLUMNC }
```

不过，可在筛选条件中使用两个表，例如以下筛选条件：

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB = 32 AND  
TABLE2.COLUMNC > 0 }
```

**注意：**在 ON 子句中输入一个条件返回的结果可能与在 WHERE 子句中输入同一条件返回的结果不同。

- 对于表使用别名时，会将别名用作表中列的前缀。例如，如果调用 REG\_CUSTOMER 表 C，则在引用列 FIRST\_NAME 时，使用“C.FIRST\_NAME”。

## 输入源筛选器

可以输入源筛选器以减少集成服务查询的行数。如果在源筛选器中包括字符串“WHERE”或大型对象，则集成服务处理会话会失败。

如果在映射中将源筛选器添加到源限定符转换，默认 SQL 查询会包含筛选条件。但是，如果在添加源筛选器后修改默认查询，则集成服务仅使用在源限定符转换的 SQL 查询部分中定义的查询。

可以使用参数或变量作为源筛选器或在源筛选器中包括参数和变量。添加字符串映射参数或变量时，请使用对源系统适用的字符串标识符。对于大部分数据库，您需要将字符串参数或变量的名称放在单引号中。

加入日期时间参数或变量时，您可能需要更改日期格式以与源中使用的格式匹配。集成服务基于源系统将日期时间参数和变量转换为字符串。

**注意:** 在会话属性中输入 SQL 查询时, 可在映射级别替代筛选条件和 SQL 查询。

要输入源筛选器, 请执行以下操作:

1. 在 Mapping Designer 中, 打开源限定符转换。  
此时将显示“编辑转换”对话框。
2. 选择“属性”选项卡。
3. 在“源筛选器”字段中单击“打开”按钮。
4. 在“SQL 编辑器”对话框中, 输入筛选器。  
包括表名称和端口名称。不要在筛选器中包括关键字 WHERE。  
将字符串映射参数和变量放在字符串标识符内。必要时, 可更改日期时间映射参数的日期格式和变量。
5. 单击“确定”。

## 使用排序端口

使用已排序端口时, 集成服务会将这些端口添加到默认查询的 ORDER BY 子句中。集成服务会添加配置的端口数 (从源限定符转换顶部开始)。如果端口的子集是已连接的下游, 则默认查询仅包含端口的子集。已排序端口会在已连接端口而非从源限定符转换顶部开始的端口上应用。

当在映射中包含任意以下转换时, 可以使用已排序端口提高性能:

- **汇总器。** 为已排序输入配置汇总器转换时, 可以通过使用已排序端口发送已排序数据。汇总器转换中的分组依据端口必须与源限定符转换中的已排序端口的顺序相匹配。
- **联接器。** 为已排序输入配置联接器转换时, 可以通过使用已排序端口发送已排序数据。将已排序端口的顺序配置为在每个源限定符转换中都相同。

**注意:** 还可以使用排序器转换, 在汇总器转换和联接器转换之前将关系和平面文件数据排序。

仅为关系源使用已排序端口。使用已排序端口时, 源数据库的排序顺序必须与为会话配置的排序顺序匹配。集成服务会创建用于提取源数据的 SQL 查询, 包括用于已排序端口的 ORDER BY 子句。数据库服务器会执行查询并将生成的数据传递给数据集成服务。为确保数据按照集成服务的要求排序, 数据库排序顺序必须与用户定义的会话排序顺序一样。

当在 Unicode 数据移动模式中为数据代码页验证配置集成服务并运行工作流时, 集成服务会使用选定的排序顺序对字符数据进行排序。

当为宽松数据代码页验证配置集成服务时, 集成服务会使用选定的排序顺序对选定排序顺序的语言范围内的所有字符数据进行排序。集成服务会根据标准 Unicode 排序顺序对选定排序顺序的语言范围外的所有字符数据进行排序。

当集成服务在 ASCII 模式中运行时, 会忽略该设置并使用二进制排序顺序对所有字符数据进行排序。默认排序顺序取决于集成服务的代码页。

源限定符转换包含默认 SQL 查询中的已排序端口数。但是, 如果在选择“已排序端口数”后修改默认查询, 则集成服务仅使用在“SQL 查询”属性中定义的查询。

要使用已排序端口, 请执行以下操作:

1. 在 Mapping Designer 中, 打开源限定符转换, 然后单击“属性”选项卡。
2. 单击“已排序端口数”, 然后输入您想要排序的端口数。  
集成服务会将配置的列数添加到 ORDER BY 子句 (从源限定符转换顶部开始)。

源数据库排序顺序必须对应于会话排序顺序。

**提示:** Sybase 支持在 ORDER BY 子句中最多使用 16 列。如果源为 Sybase, 请不要排序超过 16 列。

3. 单击“确定”。

## 选择相异

如果想要集成服务从源中选择唯一值, 请使用“选择相异”选项。可以使用该功能从列出总销售额的表中提取唯一客户 ID。使用“选择相异”筛选出数据流中不必要的早期数据, 这可能会提高性能。

默认情况下, Designer 会生成 SELECT 语句。如果选择“选择相异”, 则源限定符转换会包含默认 SQL 查询中的设置。

例如, 在[“联接源数据”页面上 352](#)中的源限定符转换中, 启用“选择相异”选项。Designer 会将 SELECT DISTINCT 添加到默认查询中, 如下所示:

```
SELECT DISTINCT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME, CUSTOMERS.LAST_NAME,
CUSTOMERS.ADDRESS1, CUSTOMERS.ADDRESS2, CUSTOMERS.CITY, CUSTOMERS.STATE, CUSTOMERS.POSTAL_CODE,
CUSTOMERS.EMAIL, ORDERS.ORDER_ID, ORDERS.DATE_ENTERED, ORDERS.DATE_PROMISED, ORDERS.DATE_SHIPPED,
ORDERS.EMPLOYEE_ID, ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE, ORDERS.STORE_ID
FROM
CUSTOMERS, ORDERS
WHERE
CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```

但是, 如果在选择“选择相异”后修改默认查询, 则集成服务仅使用在“SQL 查询”属性中定义的查询。换句话说, SQL 查询会替代“选择相异”设置。

要使用“选择相异”, 请执行以下操作:

1. 在映射中打开源限定符转换, 然后单击“属性”选项卡。
2. 选中“选择相异”, 然后单击“确定”。

## 在会话中替代为选择相异

在 Workflow Manager 中配置会话时, 可将转换级别选项替代为“选择相异”。

要替代为“选择相异”选项:

1. 在 Workflow Manager 中, 打开会话任务, 然后单击“映射”选项卡。
2. 单击“转换”视图, 然后在“源”节点下单击“源限定符转换”。
3. 在“属性”设置中, 启用“选择相异”, 然后单击“确定”。

## 添加会话前和会话后 SQL 命令

可以在源限定符转换的“属性”选项卡上添加会话前和会话后 SQL 命令。在会话开始时, 您可能希望使用会话前 SQL 将时间戳行写入源表。

集成服务将在读取源之前对源数据库运行会话前 SQL 命令。在写入目标后, 对源数据库运行会话后 SQL 命令。

您可以在会话属性“映射”选项卡的“转换”视图中替代 SQL 命令。另外, 也可以将集成服务配置为在运行会话前或会话后 SQL 命令出错时停止或继续。

在源限定符转换中输入会话前和会话后 SQL 命令时，请遵循以下准则：

- 使用任何对数据库类型有效的命令。尽管数据库可能允许嵌套注释，但集成服务不允许。
- 可以在源会话前和会话后 SQL 命令中使用参数和变量，也可以使用参数或变量作为命令。可使用您在参数文件中可定义的任何参数或变量类型。
- 使用分号 (;) 分隔多个语句。集成服务在每个语句后发出一个提交命令。
- 集成服务会忽略 /\*...\*/ 内的分号。
- 如果需要在注释外部使用分号，可以用反斜线 (\) 将其转义。将分号转义时，集成服务会忽略反斜线，并且不将分号用作语句分隔符。
- Designer 并不验证该 SQL。

**注意：**另外，您也可以在映射中目标实例的“属性”选项卡上输入会话前和会话后 SQL 命令。

## 创建源限定符转换

您可以将 Designer 配置为在将源拖动到映射中时默认创建源限定符转换，也可以手动创建源限定符转换。

### 手动创建源限定符转换

您可以在 Mapping Designer 中手动创建源限定符转换。

要手动创建源限定符转换，请执行以下操作：

1. 在 Mapping Designer 中，单击“转换”>“创建”。
2. 输入转换的名称，然后单击“创建”。
3. 选择源，并单击“确定”。
4. 单击“完成”。

### 配置源限定符转换选项

创建源限定符转换后，可以配置多个选项。

要配置源限定符转换，请执行以下操作：

1. 在 Designer 中打开一个映射。
2. 双击该源限定符转换的标题栏。
3. 在“编辑转换”对话框中，单击“重命名”，输入该转换的描述性名称，然后单击“确定”。  
源限定符转换的命名约定是 *SQ\_TransformationName*，例如 *SQ\_AllSources*。
4. 单击“属性”选项卡。
5. 输入该源限定符转换的属性。
6. 单击“源”选项卡，指示您要为此转换定义的任何关联源定义。  
仅在需要联接多个数据库或平面文件系统中的数据时标识关联的源。
7. 单击“确定”。

# 源限定符转换故障排除

我无法执行拖放操作，例如连接端口。

有关详细信息，请查看状态栏上的错误消息。

我无法将源定义连接到目标定义。

无法直接将源连接到目标。而是需要通过关系和平面文件源的源限定符转换进行连接，或者通过 COBOL 源的规范器转换进行连接。

我无法将多个源连接到一个目标。

Designer 不允许您将多个源限定符转换连接到一个目标。有两种解决方法：

- **重用目标。** 由于目标定义可重用，因此可以将同一目标多次添加到映射。然后将每个源限定符转换连接到每个目标。
- **联接源限定符转换中的源。** 然后从 SQL 查询中删除 WHERE 子句。

源在某些列中具有 QNAN（非数字）值，但目标显示 1。#QNAN.

操作系统具有不同的 NaN 字符串表示形式。集成服务将 QNAN 值转换为 1。Win64EMT 平台上的 #QNAN。

1.#QNAN 是有效的 QNAN 表示形式。

我输入了一个自定义查询，但在运行包含会话的工作流时不起作用。

请确保先为源限定符转换测试该设置，然后再运行该工作流。返回到源限定符转换，重新打开您要其中输入自定义查询的对话框。可以连接到数据库并单击“验证”按钮来测试 SQL。Designer 会显示任何错误。如果需要更多信息，请查看会话日志文件。

会话失败的最常见原因是会话和源限定符转换中的数据库登录者不是表所有者。您需要在会话中以及在源限定符转换中生成 SQL 查询时指定表所有者。

可以剪切 SQL 查询并粘贴到数据库客户端工具（如 Oracle Net）来测试该 SQL 查询，看是否会返回错误。

我在源筛选器中使用了映射变量，但现在会话失败。

通过在源限定符转换中生成并验证 SQL，尝试测试查询。如果变量或参数为字符串，则可能需要使用单引号将其引起。如果为日期时间变量或参数，则可能需要为源系统更改其格式。

## 第 26 章

# SQL 转换

本章包括以下主题：

- [SQL 转换概览, 366](#)
- [脚本模式, 367](#)
- [查询模式, 368](#)
- [连接到数据库, 372](#)
- [会话处理, 375](#)
- [输入行至输出行基数, 378](#)
- [SQL 转换属性, 381](#)
- [SQL 语句, 384](#)
- [创建 SQL 转换, 385](#)

## SQL 转换概览

SQL 转换将在管道中游处理 SQL 查询。SQL 转换可以是主动转换，也可以是被动转换。可以插入、删除、更新和检索数据库中的行。可以在运行时将数据库连接信息传递到 SQL 转换中，作为输入数据。该转换可以处理在 SQL 编辑器中创建的外部 SQL 脚本或 SQL 查询。SQL 转换可以处理查询、返回行和数据库错误。

例如，在添加新事务之前，您可能需要创建数据库表。您可以创建一个 SQL 转换，以便在工作流中创建这些表。该 SQL 转换将在输出端口中返回数据库错误。如果该 SQL 转换没有返回任何错误，您可以配置其他工作流运行。

在创建 SQL 转换时，需要配置以下选项：

- **模式。**SQL 转换可在以下某一模式下运行：
  - **脚本模式。**SQL 转换将运行位于外部的 ANSI SQL 脚本。可以通过每个输入行将脚本名称传递到该转换。SQL 转换将为每个输入行输出一行。
  - **查询模式。**SQL 转换将执行在查询编辑器中定义的查询。可将字符串或参数传递到查询，以定义动态查询或更改选择参数。当查询包含 SELECT 语句时，可以输出多行。
- **被动转换或主动转换。**默认情况下，SQL 转换是主动转换。在创建转换时，可以将其配置为被动转换。
- **数据库类型。**SQL 转换连接到的数据库的类型。
- **连接类型。**将数据库连接信息传递到 SQL 转换，或者使用连接对象。

# 脚本模式

在脚本模式下运行的 SQL 转换将通过文本文件运行 SQL 脚本。可将每个脚本文件名从源传递到 SQL 转换 ScriptName 端口。脚本文件名包含指向脚本文件的完整路径。

在配置转换以在脚本模式下运行时，可以创建被动转换。转换将为每个输入行返回一行。输出行包含查询结果和任何数据库错误。

当 SQL 转换在脚本模式下运行时，查询语句和查询数据不会更改。当需要在脚本模式下运行不同查询时，可在源数据中传递脚本。可以使用脚本模式运行数据定义查询，如创建或删除表。

在将 SQL 转换配置为在脚本模式下运行时，Designer 将向该转换添加 ScriptName 输入端口。在创建映射时，可将 ScriptName 端口连接到包含要为每行执行的脚本名称的端口。可为每个输入行执行一种不同的 SQL 脚本。Designer 将创建返回关于查询结果的信息的默认端口。

为脚本模式配置的 SQL 转换包含以下默认端口：

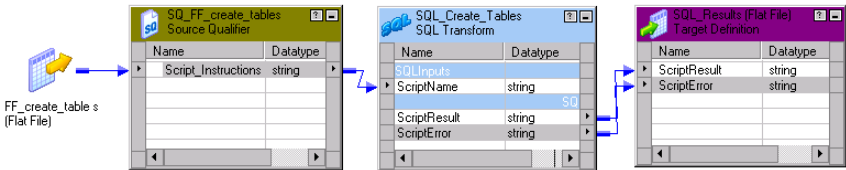
端口	类型	说明
ScriptName	输入	接收要为当前行执行的脚本的名称。
ScriptResult	输出	如果为相应行成功执行了脚本，则返回 PASSED。否则包含 FAILED。
ScriptError	输出	返回当脚本在某行失败时发生的错误。

## 示例

在将新数据添加至表前，您需要创建订单和库存表。您可以创建 SQL 脚本用来创建这些表格，并配置 SQL 转换以运行该脚本。

您创建了一个名为 create\_order\_inventory.txt 的文件，其中包含用于创建表的 SQL 语句。

以下映射显示了如何将脚本名称传递给 SQL 转换：



集成服务从源读取行。该源行包含 SQL 脚本文件名和路径：

```
C:\81\server\shared\SrcFiles\create_order_inventory.txt
```

转换在 ScriptName 端口中接收文件名。集成服务找到脚本文件并解析脚本。它创建一个 SQL 过程并将其发送到数据库进行处理。数据路验证 SQL 并执行查询。

SQL 转换返回 ScriptResults 和 ScriptError。如果脚本执行成功，ScriptResult 输出端口返回“通过”。否则，ScriptResult 端口返回“失败”。当 ScriptResult 失败时，SQL 转换会在 ScriptError 端口中返回错误消息。SQL 转换针对其收到的每个输入行返回一行。

## 脚本模式的规则和准则

- 对于在脚本模式下运行的 SQL 转换，请使用以下规则和准则：
- 可以将静态或动态数据库连接与脚本模式一起使用。



- 要在脚本中包括多个查询语句，可以使用分号分隔它们。
- 可以在脚本文件名中使用映射变量或参数。
- 脚本代码页默认为操作系统的区域设置。可以更改脚本的区域设置。
- 脚本文件必须可供集成服务访问。对于包含脚本的目录，集成服务必须拥有读取权限。如果集成服务使用操作系统配置文件，则对于包含脚本的目录，操作系统配置文件的操作系统用户必须拥有读取权限。
- 集成服务将忽略包括在 SQL 脚本中的任何 SELECT 语句的输出。脚本模式下的 SQL 转换不会为每个输入行输出一行以上的数据。
- 不能在脚本中使用脚本编写语言，如 Oracle PL/SQL 或 Microsoft/Sybase T-SQL。
- 在 SQL 脚本调用其他 SQL 脚本的情况下，不能使用嵌套脚本。
- 脚本不能接受运行时参数。

## 查询模式

SQL 转换在查询模式运行时，会执行在转换中定义的 SQL 查询。从转换输入端口将字符串或参数传递到查询以更改查询语句或查询数据。

在配置 SQL 转换以在查询模式下运行时，创建主动转换。该转换可以为每个输入行返回多个行。

在 SQL 转换 SQL 编辑器中创建查询。要创建查询，请在 SQL 编辑器主窗口中键入查询语句。SQL 编辑器提供了可在查询中引用的转换端口列表。您可以右键单击某个端口名称以将其添加为查询参数。

创建查询时，SQL 编辑器会验证查询中的端口名称。它还会验证用于字符串替换的端口是否为字符串数据类型。SQL 编辑器不会验证 SQL 查询的语法。

可以在 SQL 转换中创建以下类型的 SQL 查询：

- **静态 SQL 查询。**查询语句不会更改，但可以使用查询参数更改数据。集成服务会准备查询一次，然后为所有输入行运行查询。
- **动态 SQL 查询。**可以更改查询语句和数据。集成服务为每个输入行准备查询。

创建静态查询时，集成服务准备 SQL 过程一次，然后为每个行执行查询。创建动态查询时，集成服务为每个输入行准备 SQL。可以通过创建静态查询来优化性能。

## 使用静态 SQL 查询

如果需要为每个输入行运行相同的查询语句，但又想要为每个输入行更改查询中的数据，则请创建静态 SQL 查询。在创建静态 SQL 查询时，可以使用 SQL 编辑器中的参数绑定来为查询数据定义参数。

要更改查询中的数据，请配置查询参数，并将其绑定到转换中的输入端口。将参数绑定到输入端口后，可按查询中的名称标识端口。SQL 编辑器会用问号 (?) 将名称括起来。查询数据将根据输入端口中数据的值进行更改。

SQL 转换输入端口将为查询中的数据值或查询的 WHERE 子句中的值接收数据。

以下静态查询使用了参数绑定：

```
DELETE FROM Employee WHERE Dept = ?Dept?
INSERT INTO Employee(Employee_ID, Dept) VALUES (?Employee_ID?, ?Dept?)
UPDATE Employee SET Dept = ?Dept? WHERE Employee_ID > 100
```

以下静态 SQL 查询包含绑定到 SQL 转换的 Employee\_ID 和 Dept 输入端口的查询参数：

```
SELECT Name, Address FROM Employees WHERE Employee_Num =?Employee_ID? and Dept = ?Dept?
```



源可能包含以下行：

Employee_ID	Dept
100	Products
123	HR
130	Accounting

集成服务将从这些行中生成以下查询语句：

```
SELECT Name, Address FROM Employees WHERE Employee_ID = '100' and DEPT = 'Products'
SELECT Name, Address FROM Employees WHERE Employee_ID = '123' and DEPT = 'HR'
SELECT Name, Address FROM Employees WHERE Employee_ID = '130' and DEPT = 'Accounting'
```

## 选择多个数据库行

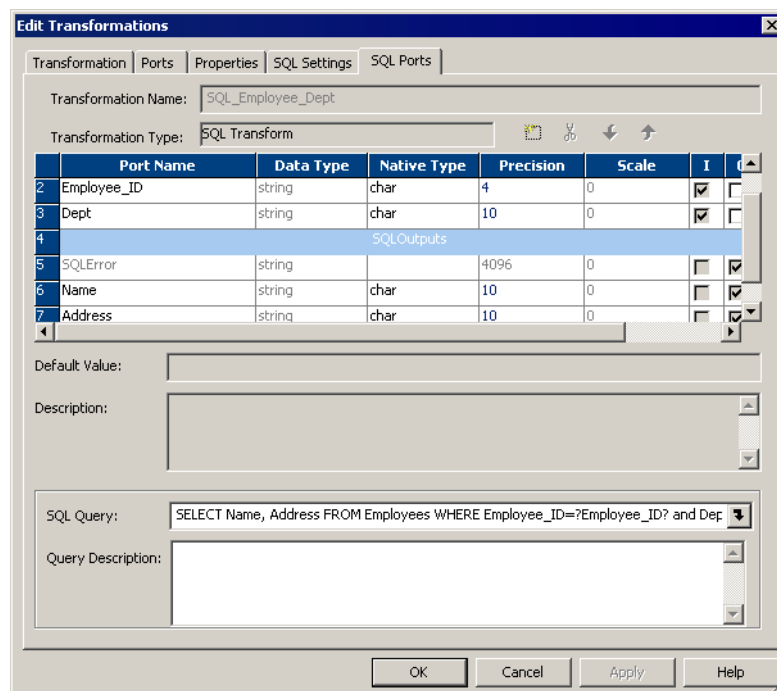
当 SQL 查询包含 SELECT 语句时，转换会为检索的每个数据库行返回一行。必须为 SELECT 语句中的每个列配置一个输出端口。输出端口的顺序必须与 SELECT 语句中的列顺序相同。

为数据库列配置输出端口时，需要配置您选择的每个数据库列的数据类型。从列表选择一个本地数据类型。当您选择本地数据类型时，Designer 会为您配置转换数据类型。

转换中的本地数据类型必须与数据库列数据类型匹配。运行时，集成服务会将数据库中的列数据类型与转换中的本地数据库类型匹配。如果数据类型不匹配，则集成服务会生成行错误。

**注意：**虽然 Teradata 数据库允许长整型列，但 Transformation Developer 不会将长整型数据类型包含为可以在 SQL 转换中使用的一种本地数据类型。

下图显示了配置为在查询模式中运行的转换中的端口：



输入端口接收 WHERE 子句中的数据。输出端口返回 SELECT 语句中的列。SQL 查询选择员工表中的姓名和地址。SQL 转换将行写入到其检索的每个数据库行的目标。

# 使用动态 SQL 查询

动态 SQL 查询可以为每个输入行执行不同的查询语句。在创建动态 SQL 查询时，使用字符串替代定义查询中的字符串参数并将其链接到转换中的输入端口。

要更改查询语句，请为您要更改的查询部分配置查询中的字符串变量。要配置字符串变量，请按查询中的名称标识输入端口，并使用波浪符 (~) 将该名称括起来。查询将基于端口中数据的值进行更改。包含查询参数的转换输入端口必须是字符串数据类型。可以使用字符串替代更改查询语句和查询数据。

在创建动态 SQL 查询时，集成服务会为每个输入行准备查询。可以在输入端口中传递完整查询或部分查询：

- **完整查询。**可以使用来自源数据的查询语句替代整个 SQL 查询。
- **部分查询。**可以替代部分查询语句，例如表名。

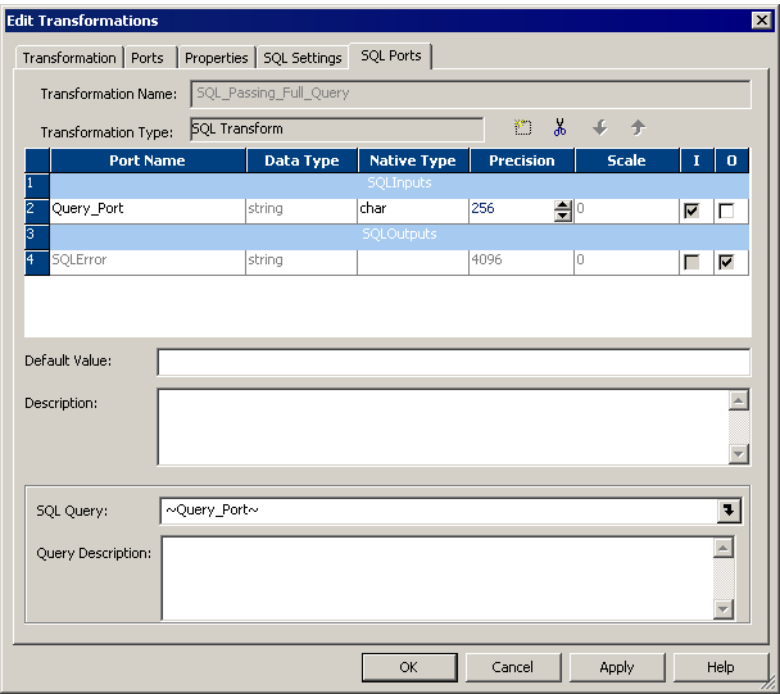
## 传递完整查询

您可以通过转换中的输入端口传递完整 SQL 查询。要传递完整查询，在 SQL 编辑器中创建包含一个字符串变量的查询以代表完整查询：

~Query\_Port~

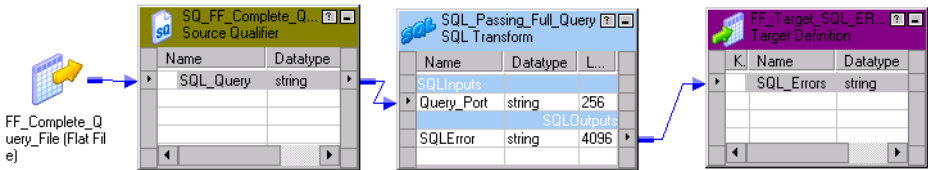
转换在 Query\_Port 输入端口中接收该查询。

下图显示了 SQL 转换中的端口：



集成服务会在动态查询中将 ~Query\_Port~ 变量替换为来自源的 SQL 语句。它会准备查询并将其发送至数据库以进行处理。数据库将执行查询。SQL 转换将数据库错误返回 SQLError 端口。

以下映射显示了如何将查询传递到 SQL 转换：



当传递完整查询时，您可以为每个输入行传递多个查询语句。例如，源可能包含以下行：

```
DELETE FROM Person WHERE LastName = 'Jones' ; INSERT INTO Person (LastName, Address) VALUES ('Smith',  
'38 Summit Drive')  
DELETE FROM Person WHERE LastName = 'Jones' ; INSERT INTO Person (LastName, Address) VALUES ('Smith',  
'38 Summit Drive')  
DELETE FROM Person WHERE LastName = 'Russell' ;
```

您可以传递源数据中任何类型的查询。当您在查询中配置 SELECT 语句时，必须为从数据库检索的数据库列配置输出端口。当您混用 SELECT 语句和其他类型的查询时，代表数据库列的输出端口在未检索到数据库列时会包含空值。

### 置换字符串中的表名称

可以置换查询中的表名称。要置换表名称，请配置输入端口以接收每个输入行的表名称。按查询中的名称标识输入端口，并使用预化符 (~) 将该名称括起来。

以下动态查询包含字符串变量 ~Table\_Port~：

```
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'
```

源可能将以下值传递到 Table\_Port 列：

#### Table\_Port

Employees\_USA

Employees\_England

Employees\_Australia

集成服务将 ~Table\_Port~ 变量替换为输入端口中的表名称：

```
SELECT Emp_ID, Address from Employees_USA where Dept = 'HR'  
SELECT Emp_ID, Address from Employees_England where Dept = 'HR'  
SELECT Emp_ID, Address from Employees_Australia where Dept = 'HR'
```

### 相关主题：

- [“动态更新示例” 页面上 386](#)

## 传递端口配置

可以将传递端口添加到 SQL 转换。传递端口是输入/输出端口，通过转换传递数据。无论 SQL 查询是否返回行，SQL 转换都会返回传递端口中的数据。

源行包含 SELECT 查询语句时，SQL 转换会为数据库中返回的每个行返回传递端口中的数据。如果查询结果包含多行，SQL 转换会在每行中重复传递数据。

如果查询未返回行，SQL 转换将在输出列中返回带有空值的传递列数据。例如，包含 INSERT、UPDATE 和 DELETE 语句的查询不返回任何行。如果查询出现错误，SQL 转换将在输出端口中返回传递列数据、SQLError 消息和空值。

要创建传递端口，请执行以下操作：

- 创建输入端口并启用它用于输出。Designer 会创建输出端口，并向端口名添加 “\_output” 后缀。
- 将端口从源限定符转换拖动到 SQL 转换。Designer 会创建传递端口。

## 被动模式配置

创建 SQL 转换时，可以将 SQL 转换配置为在被动模式而不是主动模式下运行。被动转换不会更改传递它的行数。它将维护事务边界及行类型。

如果将该转换配置为在查询模式下运行，可以在创建该转换时配置被动模式。创建该转换后无法更改模式。

### 被动模式的规则和准则

配置要在被动模式中运行的 SQL 转换时，请使用以下规则和准则：

- 如果 SELECT 查询返回多行，则集成服务会将第一行和错误返回到 SQL\_Error 端口。错误表明 SQL 转换生成了多行。
- 如果 SQL 查询具有多个 SQL 语句，则集成服务会执行所有语句。集成服务只会为第一个 SQL 语句返回数据。SQL 转换返回一行。SQL\_Error 端口包含来自所有 SQL 语句的错误。如果发生多个错误，会在 SQL\_Error 端口中通过分号将这些错误隔开。
- 如果 SQL 查询具有多个 SQL 语句并且已启用统计信息端口，则集成服务会返回第一个 SQL 语句的数据和统计信息。SQL\_Error 端口包含所有 SQL 语句的错误。

## 查询模式的规则和准则

配置要在查询模式下运行的 SQL 转换时，请使用以下规则和准则：

- 输出端口的数量和顺序必须与查询 SELECT 子句中字段的数量和顺序匹配。
- 转换中输出端口的本地数据类型必须与数据库中相应列的数据类型相匹配。如果数据类型不匹配，则集成服务会生成行错误。
- 如果 SQL 查询包含 INSERT、UPDATE 或 DELETE 子句，则转换会将数据返回到 SQL\_Error 端口、传递端口和 NumRowsAffected 端口（如果已启用）。如果添加输出端口，则这些端口会收到空数据值。
- 如果 SQL 查询包含 SELECT 语句并且转换具有传递端口，则转换会将数据返回到传递端口，而不管查询是否会返回数据库数据。SQL 转换会在输出端口中返回具有空数据的行。
- 无法将 “\_output” 后缀添加到您创建的输出端口名称。
- 无法使用传递端口从 SELECT 查询中返回数据。
- 如果输出端口数大于 SELECT 子句中的列数，则额外的端口将接收空值。
- 如果输出端口数小于 SELECT 子句中的列数，则集成服务将生成行错误。
- 可以在查询中使用字符串替代而非参数绑定。但是，输入端口必须为字符串数据类型。

## 连接到数据库

可以使用静态数据库连接，也可以在运行时将数据库连接信息传递到 SQL 转换。

对于 SQL 转换，可以使用连接环境 SQL 语句或事务 SQL 语句。在关系连接对象中配置 SQL 语句。集成服务在连接到数据库时运行连接环境 SQL，在发起每个事务前运行事务 SQL 语句。

使用以下连接类型之一将 SQL 转换连接到数据库：

- **静态连接。**配置会话中的连接对象。您必须先在 Workflow Manager 中创建连接对象。
- **逻辑连接。**在运行时将连接名称作为输入数据传递到 SQL 转换。您必须先在 Workflow Manager 中创建连接对象。
- **完全数据库连接。**在运行时将连接字符串、用户名、密码及其他连接信息传递到 SQL 转换输入端口。

**注意：**如果一个会话有多个分区，SQL 转换将为每个分区创建单独的数据库连接。

## 使用静态数据库连接

可以将 SQL 转换配置为使用静态连接连接到数据库。静态数据库连接是在 Workflow Manager 中定义的数据库连接。

要使用静态连接，请在配置会话时选择关系连接对象。要避免数据类型转换错误，请针对转换中配置的相同数据库类型使用关系连接。

## 传递逻辑数据库连接

可以将 SQL 转换配置为使用逻辑数据库连接连接到数据库。逻辑数据库连接是指您在运行时传递到转换的连接对象名称。定义 Workflow Manager 中的关系连接对象。将转换配置为使用逻辑数据库连接时，Designer 将创建 LogicalConnectionObject 输入端口。

可以针对每个输入行传递逻辑连接。配置映射以将连接对象名称传递到 LogicalConnectionObject 端口。要避免数据类型转换错误，请针对转换中配置的相同数据库类型使用关系连接。

## 传递完整连接信息

您可以将所有数据库连接信息作为输入端口数据传递到 SQL 转换。将 SQL 转换配置为通过完整连接连接到数据库时，Designer 将针对连接组件创建输入端口。数据库类型默认为您为该转换配置的数据库类型。

下表介绍了将 SQL 转换配置为通过完整连接连接到数据库时 Designer 创建的端口：

端口	必需/ 可选	说明
ConnectionString	必需	包含数据库名称和数据库服务器名称。
DBUser	必需	具有数据库读写权限的用户名。
DBPasswd	必需	DBUser 密码。
代码页	可选	集成服务用来对数据库读写的代码页。使用 ISO 代码页名称，如 ISO-8859-6。代码页名称不区分大小写。
AdvancedOptions	可选	连接属性。以名称-值对形式传递属性。使用分号分隔各个属性。属性名称不区分大小写。

## 传递连接字符串

本地连接字符串包含数据库名称和数据库服务器名称。连接字符串允许 PowerCenter 和数据库客户端直接调用正确的数据库。

下表列出了每个数据库的本地连接字符串语法：

数据库	连接字符串语法	示例
IBM DB2	<i>dbname</i>	mydatabase
Microsoft SQL Server	<i>servername@dbname</i>	sqlserver@mydatabase
Oracle	<i>dbname.world</i> （与 TNSNAMES 条目相同）	oracle.world
Sybase ASE <sup>1</sup>	<i>servername@dbname</i>	sambrown@mydatabase
Teradata <sup>2</sup>	<i>ODBC_data_source_name</i> or <i>ODBC_data_source_name@db_name</i> or <i>ODBC_data_source_name@db_user_name</i>	TeradataODBC TeradataODBC@mydatabase TeradataODBC@sambrown
Vertica	<i>ODBC_data_source_name</i>	VerticaDSN

<sup>1</sup>. Sybase ASE *servername* 为接口文件中的 Adaptive Server 的名称。

<sup>2</sup>. 使用 Teradata ODBC 驱动程序可与源数据库和目标数据库连接。

## 传递高级选项

可以配置可选连接属性。要配置属性，请以名称值对的形式传递属性。属性之间用分号分隔。属性名称不区分大小写。

例如，您可以传递以下字符串，以配置连接属性：

Use Trusted Connection = 1; Connection Retry Period = 5

下表介绍了可配置的高级选项：

属性	数据库类型	说明
连接重试时限	全部	整数。 集成服务在连接失败时尝试重新连接到数据库的秒数。
数据源名称	Teradata	字符串。 Teradata ODBC 数据源的名称。
数据库名称	Sybase ASE Microsoft SQL Server Teradata	字符串。 替代 ODBC 中的默认数据库名。如果不输入数据库名，有关连接的消息将不显示数据库名。
域名	Microsoft SQL Server	字符串。 运行 Microsoft SQL Server 的域的名称。
启用并行模式	Oracle	整数。 在批量模式下加载数据时启用并行处理。 0 表示未启用。 1 表示已启用。 默认为“已启用”。

属性	数据库类型	说明
所有者名称	全部	字符串。 表所有者名称。
数据包大小	Sybase ASE Microsoft SQL Server	整数。 优化到 Sybase ASE 和 Microsoft SQL Server 的 ODBC 连接。
服务器名称	Sybase ASE Microsoft SQL Server	字符串。 数据库服务器的名称。
使用受信任连接	Microsoft SQL Server	整数。 启用后，集成服务将使用 Windows 身份验证来访问 Microsoft SQL Server 数据库。启动集成服务的用户名必须是有效的 Windows 用户并拥有对 Microsoft SQL Server 数据库的访问权限。 0 表示未启用。 1 表示已启用。

## 数据库连接的规则和准则

为 SQL 转换配置数据库连接时使用以下规则和准则：

- 需要 PowerCenter 许可证密钥才能连接到不同的数据库类型。如果 PowerCenter 未获许可连接到数据库，则会话会失败。
- 要提高性能，请使用静态数据库连接。配置动态连接时，集成服务会为每个输入行建立一个新连接。
- 如果要在会话中使用的连接数有限，则可以配置多个 SQL 转换。将每个 SQL 转换配置为使用不同的静态连接。使用路由器转换根据行中的连接信息将行路由到 SQL 转换。
- 将 SQL 转换配置为使用完整连接数据时，数据库密码是纯文本。如果需要在会话中使用的连接数有限，则可以传递逻辑连接。逻辑连接提供与完整连接相同的功能，并且数据库密码是安全的。
- 将逻辑数据库连接传递到 SQL 转换时，集成服务会访问存储库以检索每个输入行的连接信息。如果有多行要处理，则传递逻辑数据库连接可能会影响性能。
- 默认情况下，SQL 转换使用本地数据库类型。如果要使用 ODBC 运行会话，则必须使用 ODBC 数据库类型配置转换。如果转换包含 datetime 或 datetime2 端口，则将相应的本地类型设置为时间戳。

## 会话处理

当集成服务处理 SQL 转换时，它将在管道中的中游运行 SQL 查询。当 SELECT 查询检索数据库行时，SQL 转换将在输出端口中返回数据库列。对于其他类型的查询，SQL 转换将在输出端口中返回查询结果、传递数据或数据库错误。

配置为在脚本模式下运行的 SQL 转换始终会为每个输入行返回一行。在查询模式下运行的 SQL 转换可为每个输入行返回不同数量的行。SQL 转换返回的行数取决于它所运行的查询的类型，以及查询成功与否。

可以查看集成服务传递到数据库以供处理的 SQL 查询的日志。在将日志记录设置为详细后，集成服务会将每次 SQL 查询写入到会话日志中。当需要调试包含 SQL 转换的会话时，应将日志记录设置为详细。

在将 SQL 转换配置为使用静态数据库连接时，可以使用包含该转换的事务控制。还可以在查询中执行提交和回滚语句。

SQL 转换提供了一定的数据库连接弹性。它为数据库死锁提供了弹性。

### 相关主题：

- [“输入行至输出基数” 页面上 378](#)
- [“高可用性” 页面上 376](#)

## 事务控制

在脚本模式下运行的 SQL 转换将删除任何来自上游源或事务生成器的传入事务边界。集成服务将在为 SQL 转换中的每个输入行执行脚本后进行提交。事务包含受脚本影响的行的集合。

在查询模式下运行的 SQL 转换将根据数据库连接类型在不同的点提交事务：

- **动态数据库连接。**集成服务将在为每个输入行执行 SQL 之后进行提交。事务是受脚本影响的行的集合。不能在查询模式下使用采用动态连接的事务控制转换。
- **静态连接。**集成服务将在处理所有输入行后进行提交。事务包括要更新的所有数据库行。可以通过使用事务控制转换以控制事务，或者通过在 SQL 查询中使用提交和回滚语句，来覆盖默认行为。

在配置 SQL 语句以提交或回滚行时，请配置 SQL 转换，以生成包含生成事务转换属性的事务。针对用户定义的提交配置会话。

以下事务控制 SQL 语句对于 SQL 转换无效：

- **SAVEPOINT。**标识事务中的回滚点。
- **SET TRANSACTION。**更改事务选项。

### 相关主题：

- [“动态连接示例” 页面上 391](#)

## 自动提交

可以将 SQL 转换数据库连接配置为在自动提交模式下操作。配置自动提交模式后，完成 SQL 语句时将会出现提交。

要启用自动提交，请在 SQL 转换的“SQL 设置”选项卡上选择“AutoCommit”。

如果对自动提交启用了 HA 恢复，当查询包含 insert、update 或 delete 语句时，会话日志将包含无法保证数据完整性的警告消息。

## 高可用性

具有高可用性时，SQL 转换会为静态连接和动态连接提供数据库连接弹性。集成服务无法连接到数据库时，会重试进行连接。可以为连接配置连接重试时限。集成服务无法在配置的时限内连接到数据库时，对于动态连接会生成一行错误，或对于静态连接会使会话失败。

**注意：**SQL 转换数据库连接对于 Informix 或 ODBC 连接没有弹性。

## 实时会话的仅一次处理

集成服务通过 SQL 转换提供实时源消息的仅一次递送。实时消息必须递送到 SQL 转换一次。如果处理发生中断，集成服务在恢复时不会要求再次发送该消息。如果再次发送该消息，集成服务不会运行消息中的 DML 语句两次。集成服务可能再次运行其他 SQL 语句，如 SELECT 或 SET。



为了执行仅一次处理，集成服务会在 PM\_REC\_STATE 表中存储检查点的操作状态。每个 SQL 转换均具有单独的操作状态。每个 SQL 状态维持一致的状态，且不会分享连接。您必须具有仅一次处理的高可用性。

使用以下规则和准则恢复具有 SQL 转换的实时会话：

- 必须将转换范围设置为“事务”才能为包含 SQL 转换的会话启用恢复功能。
- 不可在 SQL 查询中包含自动提交、提交语句或 DDL 语句。
- 如果 SQL 转换是被动转换，或者如果对其进行了动态连接或脚本模式配置，则无法为该转换启用 HA 恢复。
- 当您为 SQL 转换启用恢复，且工作流已启用并发执行或会话在多个分区中运行时，会话可能会失败。PM\_REC\_STATE 表上可能会发生数据库死锁。

## 查询模式弹性

如果在集成服务执行输入行的 SQL 查询时发生数据库连接错误，集成服务会尝试重新连接数据库。

对于查询模式中的弹性，使用以下规则和准则：

- 当集成服务执行列表中的第一个查询时，如果发生数据库连接故障，集成服务会为行重新执行列表中的第一个查询。它将为该行执行剩余的查询。
- 如果第一个查询语句是 SELECT 语句且通信故障发生在一些行已刷新到管道中的下游之后，会话将失败。
- 如果连接故障发生时集成服务正在执行的查询不是列表中的第一个查询，集成服务会重新连接到数据库并跳过当前行的剩余查询。先前行的查询结果可能会丢失。
- 如果集成服务无法重新连接到数据库，若 SQL 转换使用的是静态连接，则会话将失败。若 SQL 转换使用的是动态连接，则会话将继续。
- 如果列表中的 SQL 查询具有显式提交或回滚语句，则会话继续时，在提交点之后创建的所有查询结果都将丢失。

## 脚本模式弹性

如果在集成服务为输入行执行脚本时发生通信故障，则集成服务会尝试重新连接到数据库。

在脚本模式下使用以下弹性规则和准则：

- 如果集成服务无法连接到数据库且连接为静态，则会话会失败。若 SQL 转换使用的是动态连接，则会话将继续。
- 如果集成服务重新连接到数据库，则会跳过处理当前行并继续处理下一行。

## 数据库死锁复原

启用“死锁时重试会话”属性后，SQL 转换将复原数据库死锁错误。SQL 转换在查询模式下复原数据库死锁错误，但在脚本模式下不会复原死锁错误。如果在查询模式下出现死锁，集成服务将尝试重新连接数据库，次数为您配置的死锁重试次数。配置集成服务可设置死锁重试次数和死锁睡眠时间段。

出现死锁时，集成服务将在当前行（如果当前行不含 DML 语句）重试 SQL 语句。如果该行包含 INSERT、UPDATE 或 DELETE 等 DML 语句，集成服务则不会再次处理当前行。

对于动态连接，如果重试尝试失败，集成服务将在 SQLError 端口中返回错误。集成服务将基于“行内出现 SQL 错误时继续”属性处理下一个语句。如果该属性被禁用，集成服务将跳过当前行。如果当前行包含 INSERT、UPDATE 或 DELETE 等 DML 语句，集成服务将会递增错误计数。

对于静态连接，如果重试尝试失败，集成服务将在 SQLError 端口中返回错误。如果当前行包含 DML 语句，集成服务的会话将失败。集成服务将基于“行内出现 SQL 错误时继续”属性处理下一个语句。如果该属性被禁用，集成服务将跳过当前行。

# SQL 查询日志

可以查看集成服务传递到数据库以供处理的 SQL 查询的日志。在将日志记录设置为详细后，集成服务会将每次 SQL 查询写入到会话日志中。当需要调试包含 SQL 转换的会话时，应将日志记录设置为详细。

如果查询包含 CLOB 或 BLOB 数据，则会话日志不包含查询。会话日志包含一条描述数据（如 CLOB 数据）的消息。

# 输入行至输出行基数

集成服务运行 SELECT 查询时，SQL 转换将为检索到的每一行返回一行。如果查询未检索到数据，则 SQL 转换将为每一输入行返回零行或一行。

SQL 转换返回的输出行数取决于以下因素：

- **查询语句处理。**当查询包含 SELECT 语句时，集成服务可检索到多个输出行。SELECT 查询成功时，SQL 转换可能会检索到多行。如果查询包含其他语句，则集成服务可能会生成包含 SQL 错误或受影响行数的行。
- **端口配置。**NumRowsAffected 输出端口包含更新、插入或删除一个输入行影响到的总行数。如果 SQL 转换包含传递端口，则该转换将至少针对每个源行返回一次列数据。
- **最大行计数配置。**“最大输出行计数”限制了 SQL 转换为 SELECT 查询返回的行数。
- **错误行。**集成服务会在遇到连接错误或语法错误时返回行错误。当 SQL 转换在查询模式下运行时，错误将返回到 SQLException 端口。当 SQL 转换在脚本模式下运行时，错误将返回到 ScriptError 端口。
- **出现 SQL 错误时继续。**可以配置 SQL 转换以使其在 SQL 语句中存在错误时继续处理。此时 SQL 转换不会生成行错误。

# 查询语句处理

查询类型确定 SQL 转换返回的行数。在查询模式下运行的 SQL 转换可返回零行、一行或多行。查询包含 SELECT 语句时，SQL 转换会将数据库中的每一列返回到输出端口。此转换将返回所有满足条件的行。

下表列出了在查询模式下未发生错误时 SQL 转换针对不同类型的查询语句所生成的输出行：

查询语句	输出行
仅 UPDATE、INSERT、DELETE	零行。
一个或多个 SELECT 语句	检索到的数据库行总数。
DDL 查询，例如 CREATE、DROP、TRUNCATE	零行。

# 受影响的行数

您可以启用 NumRowsAffected 输出端口以返回受每个输入行中的 INSERT、UPDATE 或 DELETE 查询语句影响的行数。集成服务将针对查询中的每个语句返回 NumRowsAffected。默认情况下，NumRowsAffected 处于禁用状态。

在查询模式中启用 NumRowsAffected 但 SQL 查询不包含 INSERT、UPDATE 或 DELETE 语句时，每个输出行中的 NumRowsAffected 均为零。

**注意：**当启用了 NumRowsAffected 且转换配置为在脚本模式下运行时，NumRowsAffected 始终为空。

下表列出了在查询模式中启用了 NumRowsAffected 时 SQL 语句将生成的输出行：

查询语句	输出行
仅 UPDATE、INSERT、DELETE	为语句中每个带有 NumRowsAffected 的语句生成一行
一个或多个 SELECT 语句	检索到的数据库行总数。 在每行中，NumRowsAffected 为零。
DDL 查询，例如 CREATE、DROP、TRUNCATE	在 NumRowsAffected 为零时生成一行。

当 SQL 转换在查询模式下运行且查询包含多个语句时，集成服务将针对每个语句返回 NumRowsAffected。NumRowsAffected 包含受输入行中的 INSERT、UPDATE 和 DELETE 语句影响的行数总和。

例如，查询可包含以下语句：

```
DELETE from Employees WHERE Employee_ID = '101' ;
SELECT Employee_ID, LastName from Employees WHERE Employee_ID = '103' ;
INSERT into Employees (Employee_ID, LastName, Address)VALUES ( '102' , 'Gein', '38 Beach Rd')
```

DELETE 语句影响一行。SELECT 语句不影响任何行。INSERT 语句影响一行。

集成服务将从 DELETE 语句返回一行。NumRowsAffected 等于一。它将从 SELECT 语句返回一行，NumRowsAffected 为零。它将从 INSERT 语句返回一行，NumRowsAffected 等于一。

当以下所有条件均为 true 时，NumRowsAffected 端口返回零。

- 数据库为 Informix。
- 转换在查询模式下运行。
- 查询不包含参数。

## 最大输出行计数

您可以限制 SQL 转换为 SELECT 查询返回的行数。可以配置最大输出行计数属性来限制行数。当查询中包含多个 SELECT 语句时，SQL 转换将限制所有 SELECT 语句的总行数。

例如，将最大输出行计数设置为 100。查询中包含两个 SELECT 语句：

```
SELECT * FROM table1; SELECT * FROM table2;
```

如果第一个 SELECT 语句返回 200 行，第二个 SELECT 语句返回 50 行，则 SQL 转换将返回第一个 SELECT 语句中的 100 行。它将不会返回第二个语句中的任何行。

要将输出行数配置为不受限制，请将最大输出行计数设置为零。

## 了解错误行

当集成服务遇到连接错误或语法错误时，会返回行错误。SQL 转换包含以下用于输出错误文本的默认端口：

- **SQLException**.返回 SQL 转换在查询模式下运行时的数据库错误。
- **ScriptError**.返回 SQL 转换在脚本模式下运行时的数据库错误。

当 SQL 查询包含语法错误时，错误端口会包含数据库中的错误文本。例如，以下 SQL 查询会从 Oracle 数据库生成一行错误：

```
SELECT Product_ID FROM Employees
```

Employees 表不包含 Product\_ID。集成服务将生成一行。SQLException 端口在一行中包含以下错误文本：

```
ORA-0094: "Product_ID" : invalid identifier Database driver error... Function Name: Execute SQL Stmt:
SELECT Product_ID from Employees Oracle Fatal Error
```

如果某一查询包含多个语句，并将 SQL 转换配置为在出现 SQL 错误时继续，则 SQL 转换可能会为某一查询语句返回数据库中的行，而为另一个查询语句返回数据库错误。SQL 转换会在单独的一行中返回任何数据库错误。

配置传递端口或 NumRowsAffected 端口后，SQL 转换将针对每个源行至少返回一行。查询未返回数据时，SQL 转换将返回传递数据和 NumRowsAffected 值，但在输出端口中将返回空值。您可以利用筛选器转换传递输出行以删除带有空值的行。

下表介绍了 SQL 转换根据查询语句的类型返回的输出行。

下表介绍了 SQL 转换针对 UPDATE、INSERT 或 DELETE 查询语句生成的行：

已配置 NumRowsAffected 端口或传递端口	SQLException	行输出
两个端口均未配置。	否	零行。
两个端口均未配置。	是	在 SQLException 端口中生成一个包含错误的行。
配置了其中一个端口。	否	针对每条查询语句均生成一个含有 NumRowsAffected 列数据或传递列数据的行。
配置了其中一个端口。	是	在 SQLException 端口中生成一个包含错误且带有 NumRowsAffected 端口数据或传递端口数据的行。

下表介绍了 SQL 转换针对 SELECT 语句生成的输出行数：

已配置 NumRowsAffected 端口或传递端口	SQLException	行输出
两个端口均未配置。	否	基于从每个 SELECT 语句返回的行生成零行或多行。
两个端口均未配置。	是	生成一行（大于成功语句的总输出行数）。在 SQLException 端口中，最后一行包含错误。
配置了其中一个端口。	否	基于针对每个 SELECT 语句返回的行生成一行或多行： <ul style="list-style-type: none"><li>- 如果启用 NumRowsAffected，每行中将包含一个值为零的 NumRowsAffected 列。</li><li>- 如果配置了传递端口，每行中将包含传递列数据。当查询返回多行时，每行中的传递列数据都是重复的。</li></ul>
配置了其中一个端口。	是	基于针对每个 SELECT 语句返回的行生成一行或多行。在 SQLException 端口中，最后一行包含错误： <ul style="list-style-type: none"><li>- 如果启用 NumRowsAffected，每行中将包含一个值为零的 NumRowsAffected 列。</li><li>- 如果配置了传递端口，每行中将包含传递列数据。当查询返回多行时，每行中的传递列数据都是重复的。</li></ul>

下表介绍了 SQL 转换针对 DDL 查询（如 CREATE、DROP 或 TRUNCATE）生成的输出行数：

已配置 NumRowsAffected 端口或传递端口	SQLError	行输出
两个端口均未配置。	否	- 零行。
两个端口均未配置。	是	- 在 SQLError 端口中生成一个包含错误的行。
配置了其中一个端口。	否	- 生成一个包含值为零的 NumRowsAffected 列数据和传递列数据的行。
配置了其中一个端口。	是	- 在 SQLError 端口中生成一个包含错误且带有值为零的 NumRowsAffected 列或传递列数据的行。

## 出现 SQL 错误时继续

可以通过在“行”选项中启用“出现 SQL 错误时继续”以选择忽略语句中的 SQL 错误。集成服务将继续对该行运行剩余的 SQL 语句。集成服务不会生成行错误。但是，SQLError 端口会包含失败的 SQL 语句和错误消息。行错误计数超过会话错误阈值时，会话会失败。

例如，查询可能包含以下语句：

```
DELETE FROM Persons WHERE FirstName = 'Ed' ;  
  
INSERT INTO Persons (LastName, Address)VALUES ('Gein', '38 Beach Rd')
```

如果 DELETE 语句失败，SQL 转换将从数据库返回错误消息。集成服务将继续处理 INSERT 语句。

**提示：**禁用“出现 SQL 错误时继续”选项可调试数据库错误。否则，您可能无法将错误与导致错误的查询语句相关联。

# SQL 转换属性

在创建 SQL 转换后，可以定义端口，并在以下转换选项卡中设置属性：

- **端口。**显示您在“SQL 端口”选项卡上创建的转换端口和属性。
- **属性。**SQL 转换常规属性。
- **SQL 设置。**SQL 转换的独有属性。
- **SQL 端口。**SQL 转换端口和属性。

**注意：**不能更新“端口”选项卡上的列。在“SQL 端口”选项卡上定义端口后，这些端口将显示在“端口”选项卡上。

## “属性”选项卡

在“属性”选项卡上配置 SQL 转换的常规属性。有些转换属性不适用于 SQL 转换或不可配置。

下表介绍了 SQL 转换属性：

属性	说明
运行时位置	包含 DLL 或共享库的位置。 输入一个相对于运行 SQL 转换会话的集成服务节点的路径。 如果此属性为空，集成服务使用在集成服务节点上定义的环境变量查找 DLL 或共享库。 您必须将所有 DLL 或共享库复制到运行时位置或集成服务节点上定义的环境变量。如果不能找到 DLL、共享库或引用文件，集成服务无法加载过程。
跟踪级别	设置运行包含此转换的会话时会话日志中包含的详细信息量。如果将 SQL 转换跟踪级别配置为“详细数据”，集成服务会将准备的每条 SQL 查询写入会话日志。
IsPartitionable	管道中的多个分区可以使用此转换。请使用以下选项： <ul style="list-style-type: none"> <li>- 否。无法对转换进行分区。转换和同一渠道中的其他转换只有一个分区。如果转换一起处理所有输入数据（例如数据清理），可选择“否”。</li> <li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。如果转换的不同分区必须共享内存中的对象，可选择“本地”。</li> <li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li> </ul> 默认值为“否”。
更新策略转换	转换可定义输出行的更新策略。对于查询模式 SQL 转换可以启用此属性。 默认为“已禁用”。
转换范围	集成服务将转换逻辑应用于传入数据的方法。请使用以下选项： <ul style="list-style-type: none"> <li>- 行</li> <li>- 事务</li> <li>- 全部输入</li> </ul> 在静态查询模式下使用事务控制时，请将事务范围设置为“事务”。 对于脚本模式转换，默认值为“行”。 对于查询模式转换，默认值为“所有输入”。
输出是可重复的	指示多次会话运行的输出数据顺序是否一致。 <ul style="list-style-type: none"> <li>- 从不。会话运行之间的输出数据顺序不一致。</li> <li>- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。</li> <li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li> </ul> 默认值为“从不”。
生成事务	转换将生成事务行。对于提交 SQL 查询中数据的查询模式 SQL 转换，请启用此属性。 默认为“已禁用”。
要求每个分区一个线程	指示集成服务是否使用一个线程处理过程的每个分区。
输出具有确定性	转换将在会话运行之间生成一致的输出数据。启用此属性可对使用此转换的会话执行恢复操作。默认为“已启用”。

**警告：**如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

# “SQL 设置” 选项卡

可以在“SQL 设置”选项卡上配置 SQL 转换属性。SQL 属性是 SQL 转换独有的。

下表列出了可以在“SQL 设置”选项卡上配置的属性：

选项	说明
行内出现 SQL 错误时继续	出现 SQL 错误后，继续处理查询中的剩余 SQL 语句。
添加静态输出端口	添加 NumRowsAffected 输出端口。端口为输入行返回受 INSERT、DELETE 和 UPDATE 查询语句影响的数据库行总数。
AutoCommit	为每个数据库连接启用自动提交。查询中的每个 SQL 语句均可定义一项事务。将在 SQL 语句完成或执行下一个语句（以其中较早发生者为准）时进行提交
最大输出行计数	定义 SQL 转换可以从 SELECT 查询输出的最大行数。要配置无限多行，请将“最大输出行计数”设置为零。
脚本区域设置	标识 SQL 脚本的代码页。可以从列表中选择代码页。默认值为操作系统区域设置。
使用连接池	为数据库连接维护连接池。仅可以为动态连接启用连接池。
池中连接的最大数量	连接池中可用活动连接的最大数量。最少为 1 个。最多为 20 个。默认值为 10。

# “SQL 端口” 选项卡

在创建 SQL 转换时，Designer 将根据转换的配置方式添加默认端口。在创建转换之后，可以在“SQL 端口”选项卡上为转换添加端口。

下表列出了 SQL 转换端口：

端口	类型	模式	说明
ScriptName	输入	脚本	要为每行执行的脚本的名称。
ScriptError	输出	脚本	当脚本失败时，数据库向转换传回的 SQL 错误。
ScriptResult	输出	脚本	查询执行的结果。当查询执行成功时，将包含 PASSED。当查询失败时，将包含 FAILED。
SQLError	输出	查询	数据库向转换传回的 SQL 错误。
LogicalConnectionObject	输入	查询	动态连接。Workflow Manager 连接中定义的连接的名称。
连接字符串	输入	查询脚本	仅连接对象。数据库名称和数据库服务器名称。
DBUser	输入	查询脚本	仅完整连接。具有数据库读取和写入权限的用户的名称。

端口	类型	模式	说明
DBPasswd	输入	查询脚本	仅完整连接。数据库密码。
代码页	输入	查询脚本	仅完整连接。集成服务用于从数据库读取和向数据库写入的代码页。
高级选项	输入	查询脚本	仅完整连接。可选连接属性，如数据包大小、连接重试时限和启用并行模式。
NumRowsAffected	输出	查询脚本	输入行受 INSERT、DELETE 和 UPDATE 查询语句影响的数据库总行数。

# SQL 语句

下表列出了可以用于 SQL 转换的语句：

语句类型	语句	说明
数据定义	ALTER	修改数据库的结构。
数据定义	COMMENT	向数据字典中添加注释。
数据定义	CREATE	创建数据库、表或索引。
数据定义	DROP	删除索引、表或数据库。
数据定义	RENAME	重命名数据库对象。
数据定义	TRUNCATE	删除表中的所有行。
数据操作	CALL	调用 PL/SQL 或 Java 子程序。
数据操作	DELETE	从表中删除行。
数据操作	EXPLAIN PLAN	将语句的访问计划写入到数据库解释表中。
数据操作	INSERT	将行插入到表中。
数据操作	LOCK TABLE	阻止并发应用程序进程使用或更改表。
数据操作	MERGE	使用源数据更新表。
数据操作	SELECT	检索数据库中的数据。
数据操作	UPDATE	更新表中各行的值。
数据控制语言	GRANT	向数据库用户授予权限。



语句类型	语句	说明
数据控制语言	REVOKE	删除数据库用户的访问特权。
事务控制	COMMIT	保存工作单元，并为该工作单元执行数据库更改。
事务控制	ROLLBACK	对自上次 COMMIT 以来的数据库进行逆向更改。

# 创建 SQL 转换

可以在 Transformation Developer 或 Mapping Designer 中创建 SQL 转换。

要创建 SQL 转换，请执行以下操作：

- 单击“转换”>“创建”。
- 选择 SQL 转换。
- 输入转换的名称。  
SQL 转换的命名约定是 SQL\_TransformationName。
- 输入转换的说明，然后单击“创建”。
- 配置 SQL 转换模式：
  - 查询模式。**配置执行动态 SQL 查询的主动转换。
  - 脚本模式。**配置执行外部 SQL 脚本的被动转换。
- 配置 SQL 转换连接到的数据库类型。从列表中选择该数据库类型。
- 配置 SQL 转换连接选项：

选项	说明
静态连接	使用您为会话配置的连接对象。在会话过程中，SQL 转换连接到数据库一次。
动态连接	根据您在映射中传递到转换的连接信息连接到数据库。配置动态连接时，选择该转换是需要连接对象名称还是需要所有连接信息。默认值为连接对象。
连接对象	仅限动态连接。使用 LogicalConnectionObject 端口接收连接对象名称。定义 Workflow Manager 连接中的连接对象。
完整连接信息	仅限动态连接。使用输入端口接收所有连接组件。

- 要将 SQL 转换配置为在被动模式下运行，请选择“SQL 转换将在被动模式下运行”。
- 单击“确定”配置该转换。  
Designer 将基于您选择的选项创建转换中的默认端口。无法更改除数据库类型之外的配置。
- 单击“端口”选项卡，将端口添加到转换中。将传递端口添加到数据库端口之后。

## 第 27 章

# 在映射中使用 SQL 转换

本章包括以下主题：

- [SQL 转换示例概览, 386](#)
- [动态更新示例, 386](#)
- [动态连接示例, 391](#)

## SQL 转换示例概览

SQL 转换将在管道中游处理 SQL 查询。该转换可以处理在 SQL 编辑器中创建的外部 SQL 脚本或 SQL 查询。可以在运行时将数据库连接信息传递到 SQL 转换中，作为输入数据。

本章提供了两个示例，用于说明 SQL 转换的功能。可以使用本章中的示例创建和执行动态 SQL 查询，以及动态连接到数据库。本章提供了映射中可能包括的转换的示例数据和说明。

本章提供了以下示例：

- **创建动态 SQL 查询以更新数据库。**动态查询更新示例显示了如何根据从源文件收到的价格代码更新表中的产品价格。
- **配置动态数据库连接。**动态连接示例显示了如何根据源行中客户位置的值连接到不同数据库。

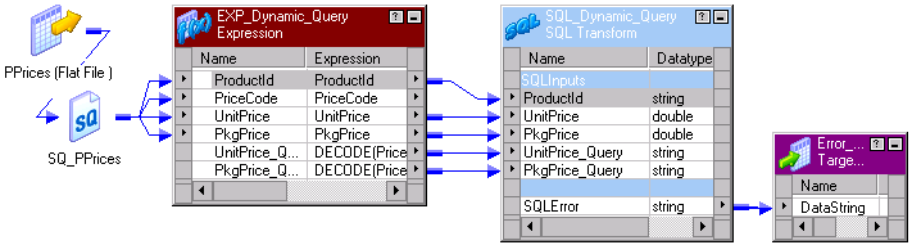
## 动态更新示例

此示例展示了如何配置表达式转换和 SQL 转换以基于源文件中列的值生成 SQL 查询。

在此示例中，您有一个包含产品价格的数据库表。您需要从交易文件更新价格。每个交易行根据价格代码列更新数据库中的批发、零售或制造价格。

源文件是一个平面文件。您可以配置表达式转换以根据每个源行中价格代码列的值返回要更新的列名称。该表达式转换将这些列名称传递到 SQL 转换。SQL 转换运行动态 SQL 查询以根据其收到的列名称更新 Prod\_Cost 表中的列。SQL 转换将数据库错误返回 Error\_File 目标。

下图显示了表达式转换如何将列名称传递到 SQL 转换：



该映射包含以下组件：

- **PPrices 源定义。** PPrices 平面文件包含产品 ID、组合价格、单价和价格代码。价格代码定义组合价格和单价是批发价、零售价，还是制造价格。
- **Error\_File 平面文件目标定义。** 目标包含“Datastring”字段，其接收来自 SQL 转换的数据库错误。
- **Exp\_Dynamic\_Expression 转换。** 该表达式转换根据 PriceCode 列的值定义要更新的 Prod\_Cost 列名称。它在 UnitPrice\_Query and PkgPrice\_Query 端口中返回这些列名称。
- **SQL\_Dynamic\_Query 转换。** 此 SQL 转换具有动态 SQL 查询，用于更新 Prod\_Cost 表中的 UnitPrice 列和 PkgPrice 列。它会更新 UnitPrice\_Query 和 PkgPrice\_Query 列中指定的列。

**注意：**映射不包含 Prod\_Cost 表的关系表定义。SQL 转换具有与包含 Prod\_Cost 表的数据库的静态连接。该转换生成 SQL 语句以更新表中的单价和组合价格。

## 定义源文件

事务文件是包含要在数据库中更新的价格的平面文件源。每个行都有一个代码，用于定义价格是批发价格、零售价格还是制造价格。源文件名为 PPrices.dat。

可以在 Srcfiles 中创建包含以下行的 PPrices.dat file 文件：

```
100,M,100,110
100,W,120,200
100,R,130,300
200,M,210,400
200,W,220,500
200,R,230,600
300,M,310,666
300,W,320,680
300,R,330,700
```

可以导入 PPrices.dat 文件以在存储库中创建 PPrices 源定义。

PPrices 文件包含以下列：

列	数据类型	精度	说明
ProductID	字符串	10	确定要更新的产品的唯一编号。
PriceCode	字符串	2	M、W 或 R。定义价格是制造价格、批发价格还是零售价格。
UnitPrice	数字	10	每单位产品的价格。
PkgPrice	数字	10	包装产品的价格。

## 创建目标定义

Error\_File 是接收来自 SQL 转换的数据库错误消息的平面文件目标定义。

Error\_File 定义包含以下输入端口：

端口名称	数据类型	精度	小数位数
DataRow	字符串	4000	0

在 Target Designer 中创建目标定义。

## 创建数据库表

SQL 转换会将产品价格写入 Prod\_Cost 关系表。Prod\_Cost 表包含每个产品的单位价格和包装价格。它会按单位和包装存储批发价格、零售价格和制造价格。

不用在存储库中为 Prod\_Cost 表创建关系目标定义。SQL 转换会生成 SQL 语句以更新数据库中的表。

Prod\_Cost 表包含以下列：

成本类型	数据类型	说明
ProductID	变长字符型	确定要更新的产品的唯一编号。
WUnitPrice	数字	批发单位价格。
WPkgPrice	数字	批发包装价格。
RUnitPrice	数字	零售单位价格。
RPkgPrice	数字	零售包装价格。
MUnitPrice	数字	制造单位价格。
MPkgPrice	数字	制造包装价格。

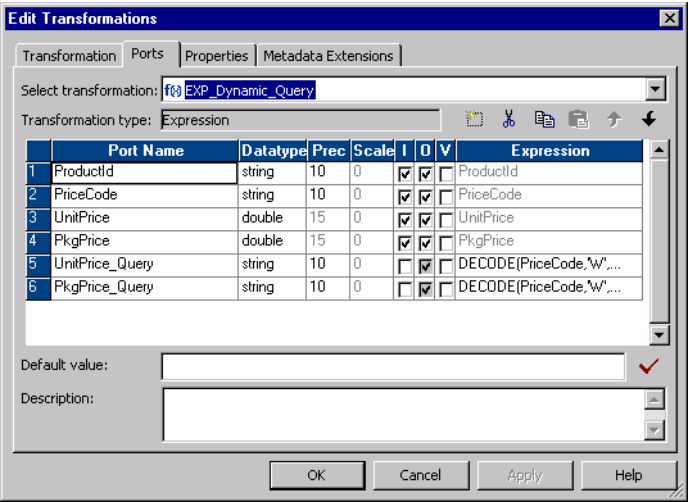
以下 SQL 语句在 Oracle 数据库中创建 Prod\_Cost 表和三个产品行：

```
Create table Prod_Cost (ProductID varchar (10), WUnitPrice number, WPkgPrice number, RUnitPrice number,
RPkgPrice number,MUnitPrice number, MPkgPrice number );
insert into Prod_Cost values('100',0,0,0,0,0,0);
insert into Prod_Cost values('200',0,0,0,0,0,0);
insert into Prod_Cost values('300',0,0,0,0,0,0);
commit;
```

## 配置表达式转换

表达式转换针对每个源列都有一个输入/输出端口。它根据 PriceCode 列的值将列名称传递到 SQL 转换。

下图显示了表达式转换中返回列名称的端口：



SQL 转换包含以下列，其中包含表达式的结果：

- **UnitPrice\_Query.**根据价格代码为“M”、“R”还是“W”，返回列名称“MUnitprice”、“RUnitPrice”或“WUnitPrice”。  
`DECODE(PriceCode,'M','MUnitPrice','R','RUnitPrice','W','WUnitPrice')`
- **PkgPrice\_Query.**根据价格代码为“M”、“R”还是“W”，返回列名称“MPkgPrice”、“RPkgPrice”或“WPkgPrice”。  
`DECODE(PriceCode,'M','MPkgPrice','R','RPkgPrice','W','WPkgPrice')`

## 定义 SQL 转换

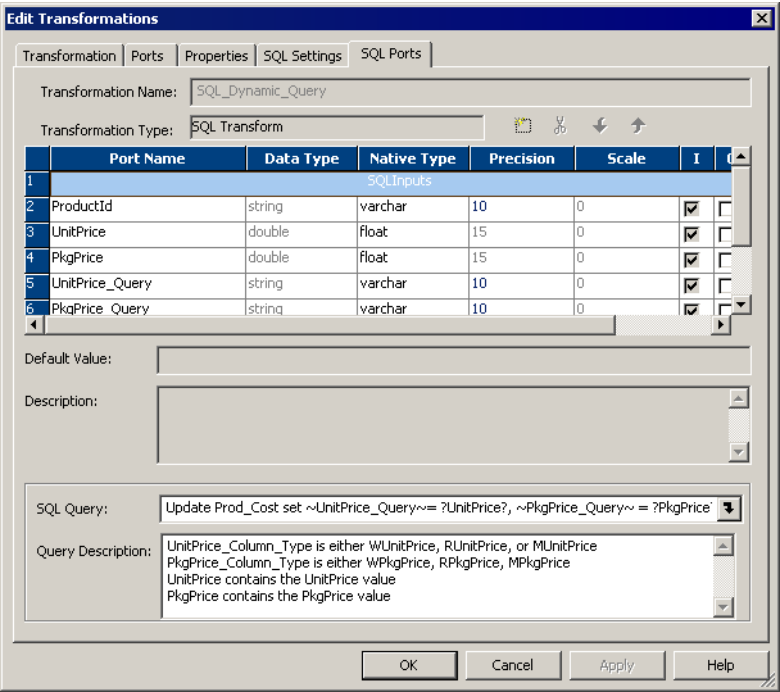
SQL 转换执行动态 SQL 查询，从而将单价和包价格数据插入 Prod\_Cost 表。SQL 转换在 UnitPrice\_Query 和 PkgPrice\_Query 端口中接收要更新的列名称。

创建 SQL 转换时，可定义转换模式、数据库类型和连接类型。创建该转换后无法更改模式或连接类型。

使用以下属性创建 SQL 转换：

- **查询模式。**SQL 转换执行动态 SQL 查询。
- **静态连接。**SQL 转换使用您在 Workflow Manager 中定义的连接对象连接到数据库一次。

下图显示了 SQL 转换的“端口”选项卡及其“SQL 查询”和“查询说明”：



SQL 转换包含一个动态 SQL 查询，可基于其在 UnitPrice\_Query 和 PkgPrice\_Query 端口中收到的列名称更新 UnitPrice 列之一和 PkgPrice 列之一。

SQL 转换具有以下查询：

```
Update Prod_Cost set ~UnitPrice_Query~ = ?UnitPrice?, ~PkgPrice_Query~ = ?PkgPrice? where ProductId = ? ProductId?;
```

SQL 转换可使用要更新的列名称替代 UnitPrice\_Query 和 PkgPrice\_Query 字符串变量。

SQL 转换可将查询中的 ProductId、UnitPrice 和 PkgPrice 参数与其在相应端口中收到的数据绑定在一起。

例如，以下源行包含产品 100 的单价和包价格：

```
100,M,100,110
```

当 PriceCode 为“M”时，这些价格为制造价格。表达式转换将 MUnitprice 和 MPkgPrice 列名称传递到 SQL 转换进行更新。

SQL 转换将执行以下查询：

```
Update Prod_Cost set MUnitprice = 100, MPkgPrice = 110 where ProductId = '100' ;
```

以下源行包含产品 100 的批发价格：

```
100,W,120,200
```

表达式转换将 WUnitprice 和 WPkgPrice 列名称传递到 SQL 转换。SQL 转换将执行以下查询：

```
Update Prod_Cost set WUnitprice = 120, WPkgPrice = 200 where ProductId = '100' ;
```

# 配置会话属性

配置会话时，需配置源文件、目标文件和数据库连接：

属性	值
源文件名	PPrices.dat
输出文件名	ErrorFile.dat
SQL_Dynamic_Query 关系连接	到包含 Prod_Cost 表的测试数据库的连接。

# 目标数据结果

使用示例数据运行会话时，集成服务会使用以下数据填充 Prod\_Cost 目标表：

ProductID	WUnitPrice	WPkgPrice	RUnitPrice	RPkgPrice	MUnitPrice	MPkgPrice
100	120	200	130	300	100	110
200	220	500	230	600	210	400
300	320	680	330	700	310	666

如果数据库向 SQL 转换返回了任何错误，Error\_File 目标会包含错误文本。

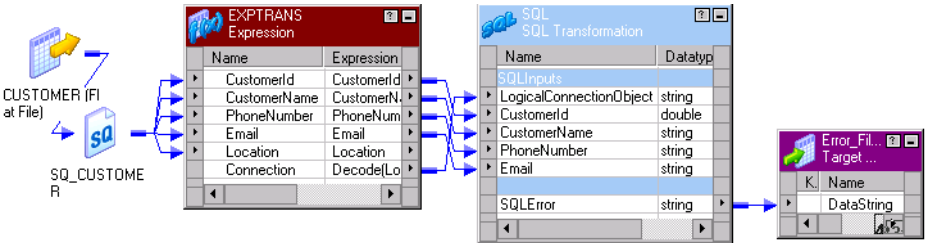
# 动态连接示例

动态连接 SQL 转换示例显示如何基于源文件数据动态连接到数据库。

在本例中，您有一个针对美国、英国和加拿大的客户数据库。您需要基于客户所在的位置将事务文件中的客户数据插入到数据库中。

表达式转换基于位置列的值返回一个数据库连接对象名称。表达式转换将连接对象名称传递到 SQL 转换 LogicalConnectionObject 端口。SQL 转换基于 LogicalConnectionObject 列的值连接到数据库。

下图显示了映射中的表达式转换和 SQL 转换：



该映射包含以下组件：

- **客户源定义。**包括客户信息的平面文件源定义。客户位置决定插入客户数据时 SQL 转换将连接到的数据库。
- **Error\_File 目标定义。**该目标包含一个 Datastring 字段，用于接收 SQL 转换中的数据库错误。

- **Exp\_Dynamic\_Connection 转换。**该表达式转换基于位置列的值定义要连接到的数据库。该表达式转换在“连接”端口中返回连接对象名称。连接对象是指在 Workflow Manager 中定义的数据库连接。
- **SQL\_Dynamic\_Connection 转换。**该 SQL 转换在 LogicalConnectionPort 中接收连接对象名称。它将连接到数据库并在该数据库中插入客户数据。

# 定义源文件

交易文件是一个包含要添加到数据库中的客户的平面文件源。每行具有一个定义客户所居住国家/地区的位置。源文件名称为 Customer.dat。

一条客户文件记录包含以下字段：

端口名称	数据类型	精度	小数位数	说明
CustomerID	数字	10	0	唯一标识客户的客户编号。
CustomerName	字符串	25	0	客户的姓名。
PhoneNumber	字符串	15	0	电话号码包括区号和七位数字，无空格。
电子邮件	字符串	25	0	客户电子邮件地址。
位置	字符串	10	0	客户位置。值为 US、UK、CAN。

您可以在包含以下行的 Srcfiles 中创建 Customer.dat 文件：

```
1,John Smith,6502345677,jsmith@catgary.com,US
2,Nigel Whitman,5123456754,nwhitman@nwhitman.com,UK
3,Girish Goyal,5674325321,ggoyal@netcompany.com,CAN
4,Robert Gregson,5423123453,rgregson@networkmail.com,US
```

您可以导入此 Customer.dat 文件以在存储库中创建客户源定义。

# 创建目标定义

Error\_File 是接收来自 SQL 转换的数据库错误消息的平面文件目标定义。

Error\_File 定义包含以下输入端口：

端口名称	数据类型	精度	小数位数
DataString	字符串	4000	0

在 Target Designer 中创建目标定义。

# 创建数据库表

在本示例中，您需要创建三个 Oracle 数据库，其中包括一个美国数据库、一个英国数据库以及一个加拿大数据库。每个数据库包含一个该地区客户数据的 Cust 表。

以下 SQL 语句可在 Oracle 数据库上创建 Cust 表：

```
Create table Cust (CustomerId number, CustomerName varchar2(25), PhoneNumber varchar2(15), Email
varchar2(25));
```



**注意:** 本示例包含三个数据库，以阐述动态数据库连接。如果表格位于同一个数据库中，使用静态数据库连接可改善性能。

## 创建数据库连接

使用 Workflow Manager 可创建指向 US、UK 和 CAN 数据库的数据库连接。

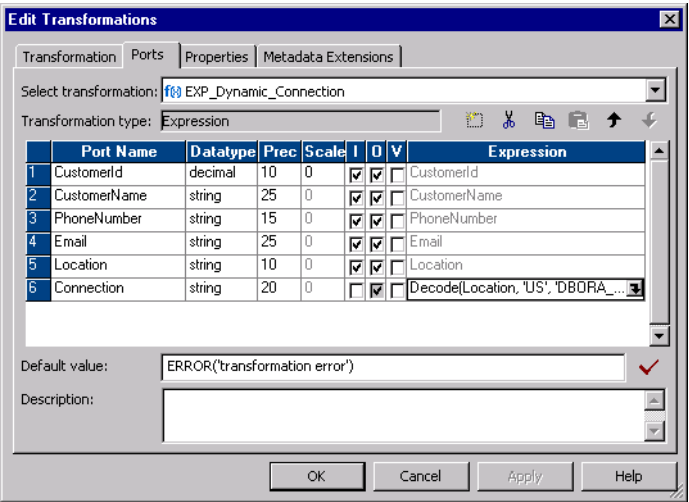
在本例中，SQL 转换将使用以下连接名称连接到数据库：

数据库	连接对象名称
CN	DBORA_US
UK	DBORA_UK
加拿大	DBORA_CAN

## 配置表达式转换

表达式转换针对每个源列都有一个输入/输出端口。此转换会根据表达式的结果在连接端口中返回一个连接对象名称。

下图显示了表达式转换中的“端口”选项卡，以及每个源列的输入/输出端口列：



连接端口具有以下表达式：

`Decode(Location, 'US', 'DBORA_US', 'UK', 'DBORA_UK', 'CAN', 'DBORA_CAN', 'DBORA_US')`

该表达式根据位置是 US、UK 还是 CAN 返回一个连接对象名称。当位置与表达式中的位置不匹配时，连接对象名称默认为“DBORA\_US”。

例如，一名美国客户的以下源行客户信息：

`1, John Smith, 6502345677, jsmith@catgary.com, US`

当客户位置是“US”时，表达式转换会返回“DBORA\_US”连接对象名称。表达式转换会将“DBORA\_US”传递到 SQL 转换 LogicalConnectionObject 端口。

# 定义 SQL 转换

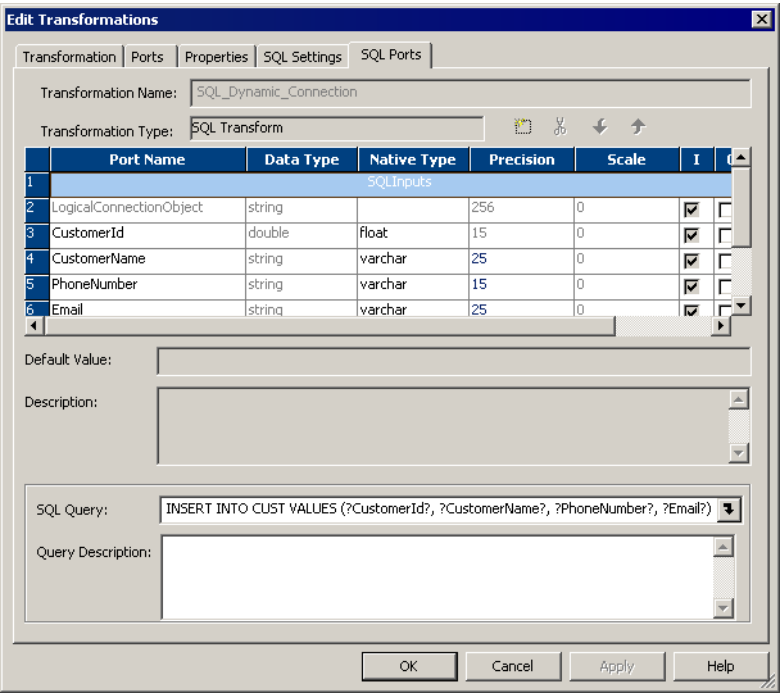
SQL 转换可连接到数据库和运行 SQL 查询，以便将客户数据插入 CUST 表。

创建 SQL 转换时，可定义转换模式、数据库类型和连接类型。创建该转换后无法更改模式或连接类型。

使用以下属性创建 SQL 转换：

- **查询模式。**SQL 转换执行动态 SQL 查询。
- **动态连接。**SQL 转换根据您在映射中传递到转换的连接信息连接到数据库。
- **连接对象。**SQL 转换包含一个 LogicalConnectionObject 端口，用于接收连接对象名称。连接对象必须在 Workflow Manager 连接中进行定义。

下图显示了 SQL 转换中的端口：



SQL 转换在 LogicalConnectionObject 端口中接收连接对象名称。它在每次处理行时使用连接对象名称连接到数据库。

该转换通过以下动态 SQL 查询将客户数据插入 CUST 表：

```
INSERT INTO CUST VALUES (?CustomerId?,?CustomerName?,?PhoneNumber?,?Email?);
```

SQL 转换将使用该转换输入端口中的客户数据置换查询中的参数。例如，以下源行包含客户编号 1 的客户信息：

```
1, John Smith,6502345677,jsmith@catgary.com,US
```

SQL 转换将使用 DBORA\_US 连接对象连接到数据库。它将执行以下 SQL 查询：

```
INSERT INTO CUST VALUES (1,' John Smith' , ' 6502345677' , ' jsmith@catgary.com' );
```

## 配置会话属性

配置会话以访问下列源和目标：

属性	值
源文件名	Customer.dat
输出文件名	Error_File.dat

**注意：**请勿为会话配置数据库连接。SQL 转换会在其每次处理输入行时连接至不同的数据库。

## 目标数据结果

使用示例数据运行会话时，集成服务会填充美国、英国或加拿大数据库中的自定义表。

美国数据库包含以下行：

CustomerID	CustomerName	PhoneNumber	电子邮件
1	John Smith	6502345677	jsmith@catagary.com
4	Robert Gregson	5423123453	rgregson@networkmail.com

英国数据库包含以下行：

CustomerID	CustomerName	PhoneNumber	电子邮件
2	Nigel Whitman	5123456754	nwhitman@nwhitman.com

加拿大数据库包含以下行：

CustomerID	CustomerName	PhoneNumber	电子邮件
3	Girish Goyal	5423123453	ggoyal@netcompany.com

如果数据库向 SQL 转换返回了任何错误，Error\_File 目标会包含错误文本。

## 第 28 章

# 存储过程转换

本章包括以下主题：

- [存储过程转换概览, 396](#)
- [在映射中使用存储过程, 399](#)
- [编写存储过程, 399](#)
- [创建存储过程转换, 401](#)
- [配置连接转换, 405](#)
- [配置未连接转换, 405](#)
- [错误处理, 408](#)
- [受支持的数据库, 409](#)
- [表达式规则, 410](#)
- [有关存储过程转换的提示, 410](#)
- [排除存储过程转换的故障, 411](#)

## 存储过程转换概览

存储过程转换是一个重要工具，用于填充和维护数据库。数据库管理员可以创建存储过程，以自动执行对于标准 SQL 语句而言过于复杂的任务。存储过程转换是一种被动转换。可以配置已连接或未连接的存储过程转换。

存储过程是预编译的 Transact-SQL、PL-SQL 或其他数据库过程语句以及可选的流控制语句的集合，与可执行脚本相似。存储过程存储在数据库中，并在数据库中运行。可以使用数据库客户端工具中的 EXECUTE SQL 语句运行存储过程，就像您能运行 SQL 语句一样。但与标准 SQL 不同，存储过程允许使用用户定义的变量、条件语句以及其他功能强大的编程功能。

并非所有数据库都支持存储过程，并且存储过程语法因数据库而异。可以使用存储过程完成以下任务：

- 向目标数据库加载数据前检查该数据库的状态。
- 确定数据库空间是否充足。
- 执行专门的计算。
- 删除然后再创建索引。

数据库开发人员和程序员可将存储过程用于数据库中的各种任务，因为存储过程可以实现比 SQL 语句更大的灵活性。存储过程还可以提供关键任务所必需的错误处理和日志记录。开发人员可以在数据库中使用数据库随附的客户端工具创建存储过程。

数据库中必须存在存储过程，然后才能创建存储过程转换；存储过程可以存在于与集成服务具有有效连接的源、目标或任何数据库中。

可以使用存储过程执行要在映射中包含的查询或计算。例如，如果已有一个用于计算销售税的经过测试的存储过程，则可通过该存储过程执行此计算，而不必在表达式转换中重新创建同样的计算。

## 输入和输出数据

存储过程最有用的功能之一就是能够向存储过程发送数据以及接收存储过程中的数据。在集成服务和存储过程之间传递的数据类型有三种：

- 输入/输出参数
- 返回值
- 状态代码

传递数据存在某些限制，视乎数据库实施而异，本章将对此进行讨论。另外，并不是所有存储过程都能发送和接收数据。例如，如果在会话末尾写入存储过程来重建数据库索引，则无法接收到数据，因为该会话已经完成。

### 输入/输出参数

对于许多存储过程，您提供一个值，并会收到一个返回值。这些值称为输入和输出参数。例如，销售税计算存储过程可使用一个输入参数，如项目价格。进行计算后，存储过程将返回两个输出参数，即税额和项目总成本（含税）。

存储过程转换使用端口、变量或通过在表达式中输入 10 或 SALES 等值发送和接收输入和输出参数。

### 返回值

大多数数据库都在运行存储过程后提供返回值。根据数据库实施情况，此值可以是用户可定义的（这意味着其行为可与单个输出参数相似），也可以仅返回整数值。

存储过程转换将根据捕获输入/输出参数的方法，按与输入/输出参数相似的方式返回值。在某些情况下，只能捕获参数或返回值。

如果存储过程返回了结果集而不是单个返回值，则存储过程转换只会获取该过程返回的第一个值。

**注意：**Oracle 存储函数与 Oracle 存储过程相似，不同之处在于存储函数支持输出参数或返回值。在本章中，任何与存储过程相关的语句也可应用于存储函数，除非另行说明。

### 状态代码

状态代码在工作流过程中为集成服务提供错误处理。存储过程发出状态代码，通知是否已成功完成存储过程。您看不到该值。集成服务使用该值确定是继续运行会话还是停止会话。如果发生存储过程错误，可以配置 Workflow Manager 中的选项以继续运行会话或停止会话。

## 已连接和未连接

存储过程在已连接或未连接模式下运行。使用的模式取决于存储过程的目的以及计划如何在会话中使用它。您可以在映射中配置已连接和未连接的存储过程转换。

- **已连接。**在已连接模式下通过映射传递的数据流还会经过存储过程转换。通过输入端口进入转换的所有数据都会影响存储过程。当您需要以输入参数形式将数据从输入端口发送到存储过程或者将存储过程的结果以输出参数形式发送到另一转换时，应使用已连接的存储过程转换。
- **未连接。**未连接的存储过程转换不会直接连接到映射流。它会在会话之前或之后运行，或者由映射中另一转换中的表达式调用。

下表将已连接转换与未连接转换进行了比较：

如果需要	请使用此模式
在会话之前或之后运行存储过程。	未连接
在映射期间运行一次存储过程，在会话之前或之后运行。	未连接
每次行经过存储过程转换时运行存储过程。	连接或未连接
根据经过映射的数据运行存储过程，例如在特定端口不含空值时。	未连接
将参数传递到存储过程并接收单个输出参数。	连接或未连接
将参数传递到存储过程并接收多个输出参数。 <b>注意:</b> 要从未连接存储过程转换中获得多个输出参数，必须为每个输出参数创建变量。	连接或未连接
运行嵌套存储过程。	未连接
在映射中调用多次。	未连接

**相关主题：**

- [“配置未连接转换” 页面上 405](#)
- [“配置连接转换” 页面上 405](#)

## 指定何时运行存储过程

除了指定存储过程转换的模式以外，还可指定何时运行存储过程。如果发生上述未连接存储过程的情况，表达式转换将引用存储过程，这意味着每次行通过表达式转换进行传递时都会运行存储过程。不过，如果没有任何转换引用存储过程转换，则您可以选择在会话之前或之后运行一次存储过程。

以下列表介绍了运行存储过程转换的选项：

- **普通视图。** 存储过程将在映射中存在转换的情况下以逐行的方式运行。这对于为通过映射传递的每行数据调用存储过程非常有用，如针对输入端口运行计算。连接存储过程仅以普通模式运行。
- **预加载源。** 在会话从源检索数据之前，存储过程会运行。这对于验证表的存在性或在临时表中执行数据联接非常有用。
- **加载后源。** 在会话从源检索数据之后，存储过程会运行。这对于删除临时表非常有用。
- **预加载目标。** 在会话将数据发送到目标之前，存储过程会运行。这对于验证目标系统上的目标表或磁盘空间非常有用。
- **加载后目标。** 会话将数据发送到目标之后，存储过程会运行。这对于在数据库中重新创建索引非常有用。

可以在相同映射中以不同模式运行多个存储过程转换。例如，预加载源存储过程可以检查表的完整性，普通存储过程可以填充表，加载后存储过程可在数据库中重建索引。但您不能在映射中同时以连接和未连接模式运行存储过程转换的同一实例。您必须创建转换的不同实例。

如果映射调用某一映射中的多个源或目标预加载或加载后存储过程，则集成服务将按您在该映射中指定的执行顺序执行存储过程。

集成服务将使用您在转换属性中指定的数据库连接执行每个存储过程。集成服务将在遇到第一个存储过程时打开数据库连接。数据库连接将保持打开，直到集成服务处理完该连接的所有存储过程为止。集成服务将在遇到使用不同数据库连接的存储过程时关闭数据库连接，然后打开一个新的数据库连接。

要运行多个使用同一数据库连接的存储过程，请将这些存储过程设置为连续运行。如果不将它们设置为连续运行，可能会在目标中获得意外结果。例如，您有两个存储过程：存储过程 A 和存储过程 B。存储过程 A 开始事务处理，而存储过程 B 则提交该事务。如果您在存储过程 B 之前运行存储过程 C，请使用其他数据库连接，存储过程 B 无法提交该事务，因为集成服务将在运行存储过程 C 时关闭该数据库连接。

请使用以下准则，以在一个数据库连接内运行多个存储过程：

- 存储过程将使用在存储过程属性中定义的同一数据库连接字符串。
- 将存储过程设置为按照连续顺序运行。
- 存储过程具有相同的存储过程类型：
  - 源预加载
  - 源加载后
  - 目标预加载
  - 目标加载后

## 在映射中使用存储过程

必须执行几个步骤才能在映射中使用存储过程转换。由于存储过程存在于数据库中，因此不仅必须配置映射和会话，还必须在数据库中配置存储过程。

要使用存储过程转换，请完成以下步骤：

1. 在数据库中创建存储过程。

在使用 Designer 创建转换之前，必须在数据库中创建存储过程。还应通过提供的数据库客户端工具测试存储过程。
2. 导入或创建存储过程转换。

使用 Designer 导入或创建存储过程转换，为任何必要的输入/输出和返回值提供端口。
3. 确定是以连接方式还是以未连接方式使用转换。

在配置转换之前，必须确定存储过程如何与映射相关。
4. 如果已连接，请映射适当的输入和输出端口。

使用连接存储过程转换的方式与大多数其他转换的方式相同。将适当的输入流端口拖动到该转换，然后创建从输出端口到其他转换的映射。
5. 如果未连接，则将存储过程配置为运行前期会话或后期会话，或者将其配置为从其他转换中的表达式运行。

由于存储过程可在会话之前或之后运行，因此可能需要指定何时应该运行未连接的转换。另一方面，如果是从其他转换调用存储过程，则请在调用该存储过程的其他转换中编写表达式。该表达式可以包含变量，并且可以包括（也可以不包括）返回值。
6. 配置会话。

Workflow Manager 中的会话属性包括用于在运行存储过程时处理错误的选项，以及一些 SQL 覆盖选项。

## 编写存储过程

编写 SQL 语句，以便在数据库中创建存储过程，还可以添加其他 Transact-SQL 语句和特定于数据库的函数。它们可以包括用户定义的数据类型和执行命令语句。

# 存储过程示例

在以下示例中，元数据库包含一个存储过程，该存储过程获取员工 ID 编号的输入参数，然后返回员工姓名的输出参数。此外，还会返回一个等于 0 的返回值，作为存储过程成功完成的通知。

包含员工 ID 和姓名的数据库表将按以下方式显示：

员工 ID	员工姓名
101	Bill Takash
102	Louis Li
103	Sarah Ferguson

存储过程接收员工 ID 101 作为输入参数，然后返回姓名 Bill Takash。根据映射调用此存储过程的方式，可将任何或所有 ID 传递到该存储过程。

由于语法因数据库而异，因此用于创建此存储过程的 SQL 语句也会有所不同。用于将 SQL 语句传递到数据库的客户端工具也会有所不同。大多数数据库都提供了一组客户端工具，包括标准 SQL 编辑器。有些数据库（如 Microsoft SQL Server）提供了多种能够创建某些初始 SQL 语句的工具。

**注意：**集成服务将使包含存储过程参数（具有大型对象）的会话失败。

## Informix

在 Informix 中，声明输出参数的语法与其他数据库不同。对于大多数数据库，使用 IN 或 OUT 声明变量以指定是否将该变量用作输入或输出参数。Informix 使用关键字 RETURNING，使得输入/输出参数和返回值难以区分。例如，使用 RETURN 命令返回一个或多个输出参数：

```
CREATE PROCEDURE GET_NAME_USING_ID (nID integer)
RETURNING varchar(20);
define nID integer;
define outVAR as varchar(20);
SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID
return outVAR;
END PROCEDURE;
```

请注意，这种情况下，RETURN 语句传递 outVAR 的值。但与其他数据库不同的是，outVAR 不是返回值，而是输出参数。将按以下方式返回多个输出参数：

```
return outVAR1, outVAR2, outVAR3
```

Informix 确实会传递返回值。返回值不是用户定义的，而是生成错误检查值。在转换中，必须检查 R 值。

## Oracle

在 Oracle 中，返回值的任何存储过程都称为存储函数。根据示例，可以使用 CREATE FUNCTION 语句创建新的存储过程，而不是使用 CREATE PROCEDURE 语句。在此示例中，变量声明为 IN 和 OUT，但是 Oracle 还支持 INOUT 参数类型，这使您可以传入参数、对其进行修改以及返回修改的值：

```
CREATE OR REPLACE FUNCTION GET_NAME_USING_ID (
nID IN NUMBER,
outVAR OUT VARCHAR2)
RETURN VARCHAR2 IS
RETURN_VAR varchar2(100);
BEGIN
SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID;
RETURN_VAR := 'Success';
RETURN (RETURN_VAR);
END;
/
```



请注意，返回值是字符串值 (Success)，数据类型为 VARCHAR2。Oracle 是唯一允许具有字符串数据类型返回值的数据库。

## Sybase ASE 和 Microsoft SQL Server

Sybase 和 Microsoft 实施存储过程的方式相同，如以下语法所示：

```
CREATE PROCEDURE GET_NAME_USING_ID @nID int = 1, @outVar varchar(20) OUTPUT
AS
SELECT @outVar = FIRST_NAME FROM CONTACT WHERE ID = @nID
return 0
```

请注意，返回值不需要是变量。在这种情况下，如果 SELECT 语句成功，则返回 0 作为返回值。

## IBM DB2

以下文本是 IBM DB2 上的 SQL 存储过程示例：

```
CREATE PROCEDURE get_name_using_id ( IN id_in int,
                                     OUT emp_out char(18),
                                     OUT sqlcode_out int)
LANGUAGE SQL
P1: BEGIN
  -- Declare variables
  DECLARE SQLCODE INT DEFAULT 0;
  DECLARE emp_TMP char(18) DEFAULT ' ';
  -- Declare handler
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SET SQLCODE_OUT = SQLCODE;
  select employee into emp_TMP
  from doc_employee
  where id = id_in;
  SET emp_out = EMP_TMP;
  SET sqlcode_out = SQLCODE;
END P1
```

## Teradata

以下文本是 Teradata 上的 SQL 存储过程的示例。该存储过程获取员工 ID 编号作为输入参数，然后返回员工姓名作为输出参数：

```
CREATE PROCEDURE GET_NAME_USING_ID (IN nID integer, OUT outVAR varchar(40))
BEGIN
  SELECT FIRST_NAME INTO :outVAR FROM CONTACT where ID = :nID;
END;
```

# 创建存储过程转换

在数据库中配置和测试存储过程后，必须在 Mapping Designer 中创建存储过程转换。配置存储过程转换的方式有两种：

- 使用“导入存储过程”对话框配置存储过程使用的端口。
- 手动配置转换，为任何输入或输出参数创建适当的端口。

默认情况下，存储过程转换以“普通”类型创建，表示它们将在映射期间运行，而不是会话前后。

新存储过程转换不会创建为可重用转换。要创建可重用转换，请在创建转换后单击“转换”属性中的“设为可重用”。

**注意：**在 Transformation Developer（而不是 Mapping Designer）中配置该可重用转换的属性，使更改全局应用于转换。

## 导入存储过程

导入存储的过程时，Designer 会根据存储过程输入和输出参数创建端口。您应尽可能导入存储过程。

有三种方式在 Mapping Designer 中导入存储过程：

- 选择存储过程图标并添加存储过程转换。
- 单击“转换”>“导入存储过程”。
- 单击“转换”>“创建”，然后选择“存储过程”。

导入名称中包含句点 (.) 的存储过程时，Designer 会在存储过程转换名称中将句点替换为下划线 (\_)。

要导入存储过程：

1. 在 Mapping Designer 中，单击“转换”>“导入存储过程”。
2. 从 ODBC 源列表中选择包含存储过程的数据库。输入用于连接至数据库的用户名和密码并单击“连接”。

对话框中的文件夹此时显示 FUNCTIONS。此文件夹中的存储过程包含输入参数、输出参数或返回值。如果存储过程存在于不包含参数或返回值的数据库中，它们将显示在名为 PROCEDURES 的文件夹中。这种情况主要适用于 Oracle 存储过程。对于函数列表中显示的普通已连接存储过程，需要至少一个输入端口和一个输出端口。

**提示：**您可以选择“跳过”来添加存储过程转换而不导入存储过程。在此情况下，您需要在转换内手动添加端口和连接信息。

3. 可选择使用搜索字段限制显示的过程数目。
4. 选择要导入的过程并单击“确定”。

存储过程转换将显示在映射中。该存储过程转换的名称与您选择的存储过程相同。如果存储过程包含输入参数、输出参数或返回值，则存储过程转换中会显示匹配每个参数或返回值的相应端口。

在此存储过程转换中，您可以看到存储过程包含以下值和参数：

- 一个具有输出端口的整数返回值，RETURN\_VALUE。
- 一个具有输入端口的字符串输入参数，名为 nNAME。
- 一个具有输入和输出端口的整数输出参数，名为 outVar。

**注意：**如果更改转换名称，则需要在转换属性中配置存储过程的名称。如果在一个映射中具有同一存储过程的多个实例，也必须配置存储过程的名称。

5. 打开转换，并单击“属性”选项卡。

从“连接信息”行中选择存储过程所在的数据库。如果已将存储过程转换的名称更改为存储过程名称之外的其他名称，输入“存储过程名称”。

6. 单击“确定”。

## 手动创建存储过程转换

要手动创建存储过程转换，您需要知道存储过程的输入参数、输出参数和返回值（如果有）。另外，还必须知道这些参数的数据类型及存储过程的名称。所有上述项目均可通过导入存储过程进行配置。

要创建存储过程转换，请执行以下操作：

1. 在 Mapping Designer 中，单击“转换”>“创建”，然后选择“存储过程”。

存储过程转换的命名约定是存储过程的名称，该名称将自动生成。如果要更改转换名称，需要在“转换”属性中配置存储过程的名称。如果映射中存在同一存储过程的多个实例，则必须执行此步骤。

2. 单击“跳过”。

存储过程转换将显示在 Mapping Designer 中。

3. 打开该转换，并单击“端口”选项卡。
- 必须基于存储过程中的输入参数、输出参数和返回值创建端口。在存储过程转换中，为以下每个存储过程参数创建一个端口：
- 整数型输入参数
  - 字符串型输出参数
  - 返回值
- 对于整数型输入参数，可创建一个整数型输入端口。参数和端口的数据类型和精度必须相同。对于输出参数和返回值重复此步骤。
- 对于返回值应选择 R 列和输出端口。对于包含多个参数的存储过程，必须按存储过程中显示的相同顺序列出端口。
4. 单击“属性”选项卡。
- 在“存储过程名称”行中输入该存储过程的名称，然后从“连接信息”行中选择该存储过程所在的数据库。
5. 单击“确定”。
- 虽然存储库会验证和保存映射，但 Designer 不会验证手动输入的存储过程转换。尚未完成检查来确认存储过程中存在的参数或返回值是否正确。如果存储过程转换的配置不正确，会话将失败。

## 设置存储过程的选项

下表介绍了存储过程转换的属性：

设置	说明
存储过程名称	数据库中的存储过程的名称。如果转换的名称不同于数据库中的实际存储过程名称，则集成服务会使用该文本调用存储过程。如果转换名称与存储过程名称匹配，则将该字段留空。使用“导入存储的过程”功能时，该名称会与存储过程匹配。
连接信息	指定包含存储过程的数据库。可以在映射、会话或参数文件中定义数据库： <ul style="list-style-type: none"><li>- 映射。选择关系连接对象。</li><li>- 会话。使用 \$Source 或 \$Target 连接变量。如果使用这些变量中的一个，则存储过程必须驻留在您运行会话时指定的源或目标数据库中。为会话属性中的每个变量指定数据库连接。</li><li>- 参数文件。使用会话参数 \$DBConnectionName，并在参数文件中进行定义。</li></ul> 默认情况下，Designer 为普通存储过程类型指定 \$Target。对于加载前和加载后源，Designer 指定 \$Source。对于加载前和加载后目标，Designer 指定 \$Target。可以在会话属性中替代这些值。
调用文本	用于调用存储过程的文本。仅在“存储过程类型”并非为“普通”时使用。必须在调用文本内包含传递到存储过程的所有输入参数。 还可以在调用文本中使用 PowerCenter 参数或变量。可使用您在参数文件中可定义的任何参数或变量类型。
存储过程类型	确定集成服务何时调用存储过程。这些选项在源或目标数据库上包含“普通”（映射期间）或者加载前或加载后。默认值为“普通”。

设置	说明
跟踪级别	会话日志文件中报告的事务详细信息量。使用以下跟踪级别： <ul style="list-style-type: none"> <li>- 简洁</li> <li>- 普通</li> <li>- 详细初始化</li> <li>- 详细数据</li> </ul> 默认值为“普通”。
执行顺序	集成服务调用转换中使用的存储过程的顺序，相对于同一映射中的任何其他存储过程。仅当“存储过程类型”设置为除“普通”以外的任何其他类型并且存在多个存储过程时使用该选项。
子秒精度	指定日期时间端口的子秒精度。 可以更改具有可编辑日期时间数据小数位数的数据库的精度。可以为 Oracle 时间戳、Informix 日期时间和 Teradata 时间戳数据类型更改子秒精度。 输入在从 0 到 9 范围内的正整数值。默认值为 6（微秒）。
输出是可重复的	指示转换是否在会话运行之间以相同的顺序生成行。当输出是可重复的确定性输出时，集成服务可以从上次检查点恢复会话。使用以下值： <ul style="list-style-type: none"> <li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li> <li>- 基于输入顺序。当所有输入组的输入数据顺序在会话运行之间一致时，转换可在会话运行之间生成可重复数据。如果所有输入组中的输入数据均未排序，输出数据也不会排序。</li> <li>- 从不。会话运行之间的输出数据顺序不一致。如果转换不生成可重复数据，则无法将恢复配置为从上次检查点继续。</li> </ul> 默认值为“基于输入顺序”。
输出具有确定性	指示转换在多次会话运行时是否生成一致的输出数据。必须启用此属性以对使用此转换的会话执行恢复。 默认为“已禁用”。

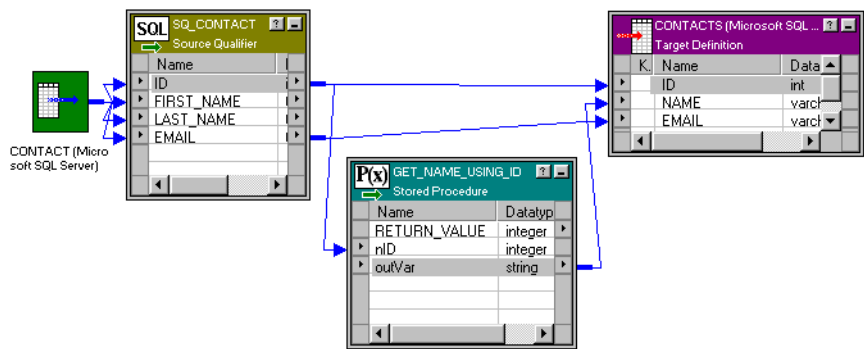
**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

## 更改存储过程

如果存储过程中的参数或返回值数更改，您可以手动重新导入它们或编辑存储过程转换。Designer 不会在您每次打开映射时都验证存储过程转换。导入或创建转换后，Designer 不会验证存储过程。如果存储过程与转换不匹配，会话将失败。

# 配置连接转换

下图显示了将 ID 从源限定符发送至存储过程转换中输入参数的映射：



存储过程转换将输出参数传递到目标。源限定符转换中的每行数据通过存储过程转换来传递。尽管不是必需的，但几乎所有连接的存储过程转换都包含输入和输出参数。必需的输入参数作为存储过程转换的输入端口来指定。输出参数显示为转换中的输出端口 返回值也是输出端口，其具有在转换端口配置中选择的“R”值。对于函数列表中显示的普通已连接存储过程，需要至少一个输入端口和一个输出端口。

来自存储过程的输出参数和返回值用作转换中的任意输出端口。您可以将这些端口链接到另一个转换或目标。

要配置连接的存储过程转换：

1. 在映射中创建存储过程转换。
2. 将存储过程的输出端口拖动到其他转换或目标。
3. 打开存储过程转换，并选择“属性”选项卡。
4. 如果在创建转换时未进行选择，请在“连接信息”中选择相应的数据库。
5. 选择转换的跟踪级别。  
如果要测试映射，选择“详细初始化”选项以在转换失败时提供多数信息。
6. 单击“确定”。

# 配置未连接转换

未连接的存储过程转换不会通过映射直接连接到数据流。而是，存储过程通过以下方式运行：

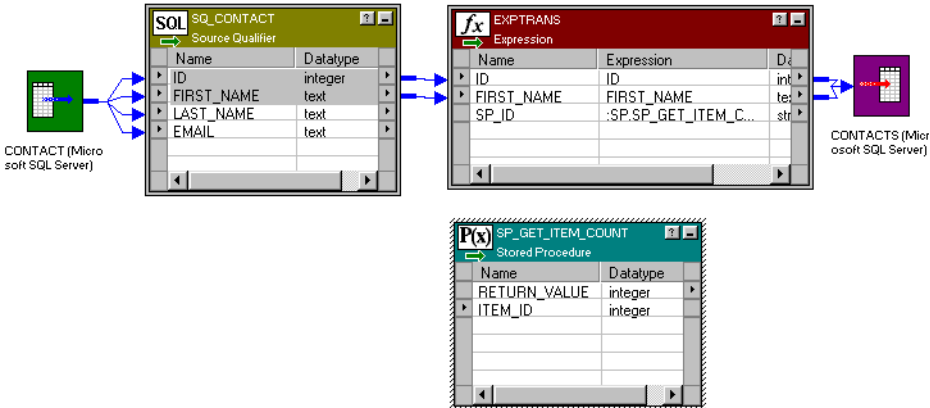
- **从表达式。** 从在映射中另一个转换内的表达式编辑器中写入的表达式进行调用。
- **会话前或会话后。** 在会话前或后运行。

下面的部分说明了可如何运行未连接的存储过程转换。

## 从表达式调用存储过程

在未连接映射中，存储过程转换不会连接到管道。

下图显示了一个表达式转换中引用存储过程转换的映射：



不过，同已连接映射一样，可以通过该映射将存储过程应用于数据流。实际上，您具有更大的灵活性，因为您使用表达式调用存储过程，这意味着您可以选择作为输入参数传递到存储过程的数据。

在表达式中使用未连接存储过程转换时，您需要一种将输出参数的值返回到端口的方法。请使用以下方法之一捕获输出值：

- 将输出值分配到本地变量。
- 将输出值分配到系统变量 PROC\_RESULT。

通过使用 PROC\_RESULT，将返回参数的值直接分配到输出端口，进而可直接应用于目标。也可以通过将一个输出参数分配为 PROC\_RESULT，将另一个参数分配为变量，合并两个选项。

PROC\_RESULT 仅在表达式内使用。如果不使用 PROC\_RESULT 或变量，包含表达式的端口将捕获到空值。在已连接查找转换或存储过程转换的调用文本内不能使用 PROC\_RESULT。

如果需要嵌套存储过程（一个存储过程的输出参数由此传递到另一个存储过程），请使用 PROC\_RESULT 传递值。

集成服务将调用表达式转换中的未连接存储过程转换。请注意，存储过程转换有两个输入端口和一个输出端口。所有三个端口都是字符串数据类型。

要从表达式内调用存储过程，请执行以下操作：

1. 在映射中创建存储过程转换。
2. 在支持输出和变量端口的任何转换中，在调用该存储过程的转换中创建一个新的输出端口。对该输出端口命名。  
该调用存储过程的输出端口必须支持表达式。视乎表达式的配置方式，该输出端口包含输出参数的值或返回值。
3. 针对该端口打开表达式编辑器。  
在表达式编辑器中，将该新端口的值设置为使用转换语言中的 .SP 关键字调用存储过程。设置此项最简便的方法是在表达式编辑器中选择存储过程节点，单击所列出的存储过程转换的名称。对于函数列表中显示的普通已连接存储过程，需要至少一个输入端口和一个输出端口。  
此时该存储过程将显示在表达式编辑器中，带有一对空括号。必要的输入和/或输出参数显示在表达式编辑器的左下角。
4. 将该表达式配置为发送输入参数和捕获输出参数或返回值。  
您必须知道表达式编辑器中显示的参数是输入参数还是输出参数。按变量或端口在存储过程中显示的顺序在括号之间插入变量或端口名称。这些端口或变量的数据类型必须与传递到存储过程的参数的数据类型匹配。

例如，单击该存储过程时，将显示类似于以下内容的信息：

```
:SP.GET_NAME_FROM_ID()
```

此特定存储过程需要整数值作为输入参数，并会返回字符串值作为输出参数。如何捕获输出参数或返回值因输出参数的数量和是否需要捕获返回值而异。

如果该存储过程返回一个输出参数或返回值（但并非两者），应使用预留变量 PROC\_RESULT 作为输出变量。在上个示例中，表达式显示如下：

```
:SP.GET_NAME_FROM_ID(inID, PROC_RESULT)
```

inID 可以是转换的输入端口或转换中的变量。将 PROC\_RESULT 的值应用到表达式的输出端口。

如果该存储过程返回多个输出参数，必须为每个输出参数创建变量。例如，如果为该存储过程表达式创建了一个名为 varOUTPUT2 的端口和一个名为 varOUTPUT1 的变量，则该表达式将显示如下：

```
:SP.GET_NAME_FROM_ID(inID, varOUTPUT1, PROC_RESULT)
```

将第二个输出端口的值应用到表达式的输出端口，并将第一个输出端口的值应用到 varOUTPUT1。输出参数将按在该存储过程中声明的顺序返回。

对于所有这些表达式，端口和变量的数据类型必须与输入/输出变量和返回值的数据类型匹配。

5. 单击“验证”验证该表达式，然后单击“确定”关闭表达式编辑器。

验证表达式可确保存储过程中参数的数据类型与在表达式中输入的数据类型匹配。

6. 单击“确定”。

保存映射时，Designer 不会验证存储过程表达式。如果存储过程表达式的配置不正确，会话将失败。使用存储过程测试映射时，请将“替代跟踪”会话设置为详细模式，并配置“在存储过程中”会话选项以便在存储过程失败时停止运行。请在会话属性“配置对象”选项卡的“错误处理”设置中配置这些会话选项。

在表达式中为端口输入的存储过程不一定会影响通过该端口传递的所有值。例如，使用 IIF 语句仅可将特定值（例如以 5 开头的 ID 编号）传递到存储过程，并将跳过所有其他值。另外，也可以设置嵌套存储过程，由此一个存储过程的返回值将变成另一个存储过程的输入参数。

## 调用会话前或会话后存储的过程

您可能希望为每个会话运行一次存储过程。例如，如果需要在运行映射之前验证表是否存在于目标数据库中，加载前目标存储过程会检查表，然后继续运行工作流或将其停止。可以对源、目标或任何其他连接的数据库运行存储过程。

要创建加载前或加载后存储过程，请执行以下操作：

1. 在映射中创建存储过程转换。
2. 双击存储过程转换，然后选择“属性”选项卡。
3. 输入存储过程的名称。  
如果导入了存储过程，默认情况下会显示存储过程名称。如果手动设置存储过程，请输入存储过程的名称。
4. 在连接信息中选择包含存储过程的数据库。
5. 输入存储过程的调用文本。

调用文本是存储过程的名称，后面是用括号括起来的所有适用的输入参数。如果没有输入参数，则必须包括一对空括号，否则对存储过程的调用会失败。无需包括 SQL 语句 EXEC，也无需使用 :SP 关键字。例如，要调用称为 check\_disk\_space 的存储过程，请输入以下文本：

```
check_disk_space()
```

要传递字符串输入参数，请输入它，且不包括引号。如果字符串中包含空格，则用双引号将参数括起来。例如，如果存储过程 check\_disk\_space 需要计算机名称作为输入参数，请输入以下文本：

```
check_disk_space(oracle_db)
```



通过会话前或会话后存储过程传递日期时间值时，该值必须采用 Informatica 默认日期格式并且用双引号括起来，如下所示：

SP( "12/31/2000 11:45:59" )

可以在调用文本中使用 PowerCenter 参数和变量。可使用您在参数文件中可定义的任何参数或变量类型。可以在调用文本中输入参数或变量，或者可以使用参数或变量作为调用文本。例如，可以使用会话参数 \$ParamMyCallText 作为调用文本，并在参数文件中将 \$ParamMyCallText 设置为调用文本。

**注意：**必须为会话前和会话后过程的输入参数输入值而不是过程参数。

#### 6. 选择存储过程类型。

存储过程类型的选项包括：

- **源加载前。**在会话从源检索数据之前，存储过程会运行。这对于验证表的存在性或在临时表中执行数据联接非常有用。
- **源加载后。**在会话从源检索数据之后，存储过程会运行。这对于删除临时表非常有用。
- **目标加载前。**在会话将数据发送到目标之前，存储过程会运行。这对于验证目标系统上的目标表或磁盘空间非常有用。
- **目标加载后。**会话将数据发送到目标之后，存储过程会运行。这对于在数据库中重新创建索引非常有用。

#### 7. 选择“执行顺序”，然后根据需要单击向上或向下箭头来更改顺序。

如果在会话中添加了同时执行的多个存储过程（例如在源加载后同时运行的两个过程），则可以设置存储过程执行计划以确定集成服务调用这些存储过程的顺序。需要为要更改的每个存储过程重复执行该步骤。

#### 8. 单击“确定”。

尽管存储库会验证并保存映射，但 Designer 不会验证存储过程表达式的运行过程中是否存在错误。如果存储过程表达式的配置不正确，会话将失败。使用存储过程测试映射时，请将“替代跟踪”会话设置为详细模式，并配置“在存储过程中”会话选项以便在存储过程失败时停止运行。在会话属性中“配置对象”选项卡的“错误处理”设置上配置这些会话选项。

将丢失输出参数或返回在会话前或会话后存储过程中调用的值，因为没有地方捕捉值。如果需要捕捉值，可能要配置存储过程以在数据库的表中保存值。

## 错误处理

有时候，存储过程会返回数据库错误，例如，“除数为零”或“没有更多的行”。存储过程中数据库错误的最终结果取决于存储过程发生的时间和会话的配置方式。

可以将会话配置为遇到会话前或会话后存储过程错误时停止运行或继续运行。默认情况下，集成服务会在出现会话前或会话后存储过程数据库错误时停止会话。

## 会话前错误

读取前和加载前过程被视为会话前存储过程。两者都在集成服务开始读取源数据之前运行。如果在会话前存储过程中出现数据库错误，则集成服务会根据会话配置执行不同的操作。

- 如果将会话配置为在出现存储过程错误时停止，则集成服务处理会话会失败。
- 如果将会话配置为在出现存储过程错误时继续，则集成服务会继续处理会话。



## 会话后错误

读取后和加载后存储过程被视为会话后存储过程。两者都在集成服务将所有数据提交到数据库之后运行。如果在会话后存储过程中出现数据库错误，集成服务会根据会话配置执行其他操作。

- 如果将会话配置为在出现存储过程错误时停止，则集成服务处理会话会失败。  
但是，集成服务已将所有数据提交到会话目标。
- 如果将会话配置为在出现存储过程错误时继续，则集成服务会继续处理会话。

## 会话错误

在会话期间发生的已连接或未连接存储过程错误不会受会话错误处理选项的影响。如果数据库为某一特定行返回了错误，则集成服务将跳过该行，并继续处理下一行。与其他行转换错误一样，被跳过的行将显示在会话日志中。

## 受支持的数据库

适用于 Oracle 以及其他数据库（如 Informix、Microsoft SQL Server 和 Sybase）的受支持选项将在下面介绍。

相关主题：

- [“编写存储过程” 页面上 399](#)

## SQL 声明

在数据库中，创建存储过程的语句看起来与以下 Oracle 存储过程相似：

```
create or replace procedure sp_combine_str
(str1_inout IN OUT varchar2,
str2_inout IN OUT varchar2,
str_out OUT varchar2)
is
begin
str1_inout := UPPER(str1_inout);
str2_inout := upper(str2_inout);
str_out := str1_inout || ' ' || str2_inout;
end;
```

在这种情况下，Oracle 语句将以 CREATE OR REPLACE PROCEDURE 开头。由于 Oracle 既支持存储过程又支持存储函数，因此只有 Oracle 使用可选的 CREATE FUNCTION 语句。

## 参数类型

存储过程中有三种可能的参数类型：

- **IN.**将参数定义为必须传递到存储过程的信息。
- **OUT.**将参数定义为存储过程返回的值。
- **INOUT.**将参数定义为输入和输出。仅 Oracle 支持此参数类型。

## 映射中的输入/输出端口

因为 Oracle 支持 INOUT 参数类型，所以可将存储过程转换中的某个端口用作同一个存储过程参数的输入和输出端口。其他数据库对于端口则不应同时选中输入和输出复选框。

## 受支持的返回值的类型

不同数据库支持不同类型的返回值数据类型，只有 Informix 不支持用户定义的返回值。

## 表达式规则

可从另一个转换中的表达式中调用未连接存储过程转换。配置表达式时使用以下规则和准则：

- 使用变量 PROC\_RESULT 返回单个输出参数。
- 在表达式中使用存储过程时，使用 :SP 引用限定符。为了避免键入错误，在表达式编辑器中选择“存储过程”节点，然后双击存储过程的名称。
- 但是，存储过程转换的同一实例不可在映射中同时以已连接和未连接模式运行。您必须创建转换的不同实例。
- 表达式中的输入/输出参数必须匹配存储过程转换中的输入/输出端口。如果存储过程具有一个输入参数，存储过程转换中也必须有一个输入端口。
- 写入包括存储过程的表达式时，以参数在存储过程和存储过程转换中的显示顺序列出参数。
- 表达式中的参数必须包括存储过程转换中的所有参数。您不能遗漏输入参数。如有必要，传递一个伪变量到存储过程。
- 表达式中的参数必须与存储过程转换中的参数具有相同的数据类型和精度。
- 使用 PROC\_RESULT 将存储过程表达式的输出参数直接应用到目标。您不能使用输出参数的变量将结果直接传递到目标。使用局部变量将结果传递到相同转换内的输出端口。
- 嵌套的存储过程允许将一个存储过程的返回值作为另一个存储过程的输入参数进行传递。例如，如果具有以下两个存储过程：
  - get\_employee\_id (employee\_name)
  - get\_employee\_salary (employee\_id)并且 get\_employee\_id 的返回值是员工 ID 号，则嵌套的存储过程的语法为：

```
:sp.get_employee_salary (:sp.get_employee_id (employee_name))
```

您可以具有多级嵌套的存储过程。
- 不要在字符串参数两边使用单引号。如果输入参数不包含空格，请勿使用任何引号。如果输入参数包含空格，使用双引号。

## 有关存储过程转换的提示

请勿运行不必要的存储过程实例。

在映射期间每次存储过程运行时，会话都必须等待存储过程在数据库中完成。可以使用两个选项避免这种情况：

- **减少行计数。**在存储过程转换之前使用一个活动转换，减少必须传递到存储过程的行数。或者，在将值传递到存储过程之前，创建一个表达式，对值进行测试，以确保值并非确实需要传递。
- **创建表达式。**存储过程中使用的大多数逻辑均可使用 Designer 中的表达式轻松复制。

# 排除存储过程转换的故障

我在会话日志文件中收到了错误“未找到存储过程”。

请确保存储过程正在正确的数据库中运行。默认情况下，存储过程转换使用目标数据库来运行存储过程。双击映射中的转换，选择“属性”选项卡，然后检查在“连接信息”中选择了哪个数据库。

我的输出参数不是使用 Microsoft SQL Server 存储过程返回。

检查是否在存储过程中将保留返回值的参数声明为 OUTPUT。对于 Microsoft SQL Server，OUTPUT 表示输入/输出。在映射中，您可能同时为端口选中了“I”和“O”框。清除输入端口。

会话以前并未发生过错误，但现在对于存储过程失败。

对于涉及存储过程转换的问题，最常见的原因在于对数据库中的存储过程进行了更改。如果存储过程中的输入/输出参数或返回值发生更改，则存储过程转换将变得无效。必须再次导入存储过程；或者手动配置存储过程，以添加、删除或修改适当的端口。

会话自我上次编辑映射后便已失效。为什么？

您对存储过程转换所做的任何更改都可能使会话失效。最常见的原因是您更改了存储过程的类型，如类型从“普通”更改为“加载后源”。

## 第 29 章

# 事务控制转换

本章包括以下主题：

- [事务控制转换概览, 412](#)
- [事务控制转换属性, 412](#)
- [在映射中使用事务控制转换, 414](#)
- [映射准则和验证, 416](#)
- [创建事务控制转换, 416](#)

## 事务控制转换概览

PowerCenter 允许您基于一组传递事务控制转换的行来控制提交和回滚事务。事务控制转换是一种活动转换。事务是被提交或回滚行绑定的行的集合。可以根据不同数量的输入行来定义事务。您可能想要根据一组按某一公共键（如员工 ID 或订单输入日期）排序的行来定义事务。

在 PowerCenter 中，可在以下级别定义事务控制：

- **在映射中。**在映射中，使用事务控制转换来定义事务。可在事务控制转换中使用表达式来定义事务。根据该表达式的返回值，可以选择提交、回滚或继续而不进行任何事务更改。
- **在会话中。**在配置会话时，可以针对用户定义的提交配置会话。如果集成服务未能转换目标或向目标写入任何行，则可选择提交或回滚事务。

在运行会话时，集成服务会针对进入转换的每一行计算表达式。当集成服务计算某一提交行时，它会将事务中的所有行提交到一个或多个目标。当集成服务计算某一回滚行时，它会从一个或多个目标回滚事务中的所有行。

如果映射包含平面文件目标，则集成服务每次开始新事务时，均可生成一个输出文件。可以动态命名每个目标平面文件。

**注意：**还可使用其他转换属性中的转换范围来定义事务。

## 事务控制转换属性

可以使用事务控制转换来定义提交事务以及从事务型目标回滚事务的条件。事务型目标包括关系、XML 和动态 MQSeries 目标。可在“属性”选项卡上定义事务控制表达式中的这些参数。事务是被提交或回滚行绑定的行或行的集合。每个事务的行数可能有所不同。

在配置事务控制转换时，可以定义以下组件：

- “**转换**”选项卡。可以在“转换”选项卡上重命名转换，并添加描述。
- “**端口**”选项卡。可向事务控制转换添加输入/输出端口。
- “**属性**”选项卡。可以定义事务控制表达式，该表达式将事务标记为提交、回滚，或不标记为任何操作。
- “**元数据扩展**”选项卡。可以通过将信息与事务控制转换关联起来，对存储库中存储的元数据进行扩展。

## “属性”选项卡

在“属性”选项卡上可以配置以下属性：

- 事务控制表达式
- 跟踪级别

在“事务控制条件”字段中输入事务控制表达式。事务控制表达式使用 IIF 函数根据条件测试每行。为该表达式使用以下语法：

IIF (condition, value1, value2)

该表达式包含表示集成服务基于该条件的返回值所执行操作的值。集成服务按行计算该条件。返回值决定集成服务是提交、回滚该行的事务还是不对其做出更改。在集成服务基于表达式的返回值发出提交或回滚后，它将开始新的事务。创建事务控制表达式时，在表达式编辑器中使用以下内置变量：

- **TC\_CONTINUE\_TRANSACTION**。集成服务不对此行执行任何事务更改。这是该表达式的默认值。
- **TC\_COMMIT\_BEFORE**。集成服务提交事务、开始新事务并将当前行写入目标。当前行在新事务中。
- **TC\_COMMIT\_AFTER**。集成服务将当前行写入目标、提交事务并开始新事务。当前行在提交的事务中。
- **TC\_ROLLBACK\_BEFORE**。集成服务回滚当前事务、开始新事务并将当前行写入目标。当前行在新事务中。
- **TC\_ROLLBACK\_AFTER**。集成服务将当前行写入目标、回滚事务并开始新事务。当前行在回滚的事务中。

当事务控制表达式的计算结果为除提交、回滚或继续之外的值时，集成服务会话将失败。

## 示例

您希望使用事务控制以基于订单输入日期写入订单信息。您希望确保在任何指定日期输入的所有订单均已提交到同一事务中的目标。要完成此操作，可使用以下转换创建映射：

- **排序器转换**。按订单输入日期对源数据排序。
- **表达式转换**。使用本地变量确定输入的日期是否为新日期。

下表介绍了表达式转换中的端口：

端口名称	表达式	说明
DATE_ENTERED	DATE_ENTERED	输入/输出端口。 接收和传递输入的日期。
NEW_DATE	IIF(DATE_ENTERED=PREVDATE, 0,1)	变量端口。 针对变量端口 PREV_DATE 中 DATE_ENTERED 的存储值测试 DATE_ENTERED 的当前值。

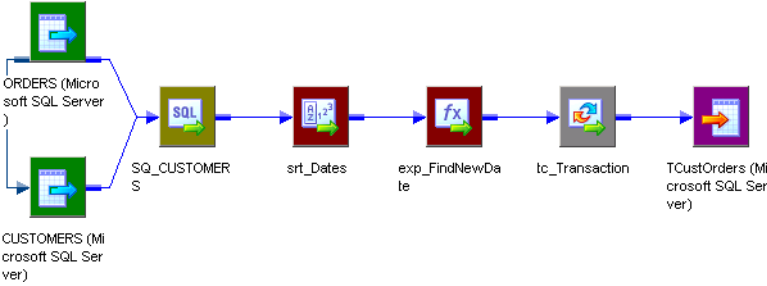
端口名称	表达式	说明
PREV_DATE	DATE_ENTERED	变量端口。 在集成服务计算 NEW_DATE 端口后接收 DATE_ENTERED 的值。
DATE_OUT	NEW_DATE	输出端口。 将 NEW_DATE 中的标记传递到事务控制转换。

**注意:** 集成服务将根据相关性计算端口。转换中端口的显示顺序必须与计算顺序匹配: input ports, variable ports, output ports.

- **事务控制转换。** 创建以下事务控制表达式以在集成服务遇到新订单输入日期时提交数据:

IIF(NEW\_DATE = 1, TC\_COMMIT\_BEFORE, TC\_CONTINUE\_TRANSACTION)

下图显示了一个使用事务控制转换的映射示例:



## 在映射中使用事务控制转换

事务控制转换是事务生成器。它们可以定义和重新定义映射中的事务边界。它们将删除任何来自上游活动源或事务生成器的传入事务边界，然后在下游生成新的事务边界。

还可以使用已配置的自定义转换来生成事务，以定义事务边界。

事务控制转换对映射中的下游转换和目标可能有效，也可能无效。如果放置一个转换，而该转换会删除事务控制转换之后的传入事务边界，则该事务控制转换对下游转换或目标将无效。这包括任何以下活动源或转换:

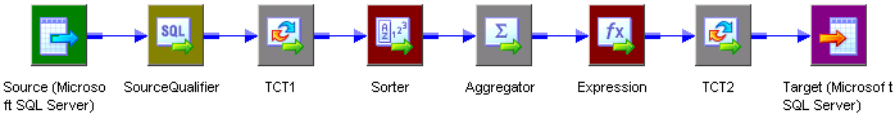
- 具有“全部输入”级别转换范围的汇总器转换
- 具有“全部输入”级别转换范围的联接器转换
- 具有“全部输入”级别转换范围的等级转换
- 具有“全部输入”级别转换范围的排序器转换
- 具有“全部输入”级别转换范围的自定义转换
- 已配置为生成事务的自定义转换
- 事务控制转换
- 已连接到多个上游事务控制点的多个输入组转换，如自定义转换

如果映射包含对目标无效的事务控制转换，则该映射可能有效，也可能无效。在保存或验证映射时，Designer 将显示一条消息，指示哪些事务控制转换对目标无效。

虽然事务控制转换可能对目标无效，但可能对下游转换有效。具有“事务”级别转换范围的下游转换，可以使用由上游事务控制转换定义的事务边界。

下图显示了包含事务控制转换的有效映射，其所包含的事务控制转换对排序器转换有效，但对目标无效。

在本示例中，TCT1 转换对目标无效，但对排序器转换有效。排序器转换的“转换范围”属性为“事务”。它使用由 TCT1 定义的事务边界。汇总器的“转换范围”属性为“全部输入”。它会删除由 TCT1 定义的事务边界。TCT2 转换是对目标有效的事务控制转换。

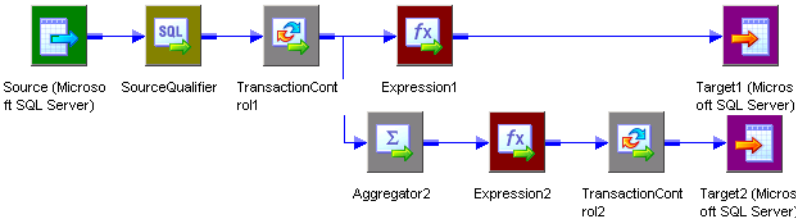


## 包含多个目标的事务控制映射示例

事务控制转换可能对目标有效，而对另一个目标无效。

如果每个目标都连接到了有效的事务控制转换，则映射有效。如果映射中的一个目标未连接到有效的事务控制转换，则该映射无效。

下图显示了一个有效映射，其中同时包含一个无效和一个有效的事务控制转换：

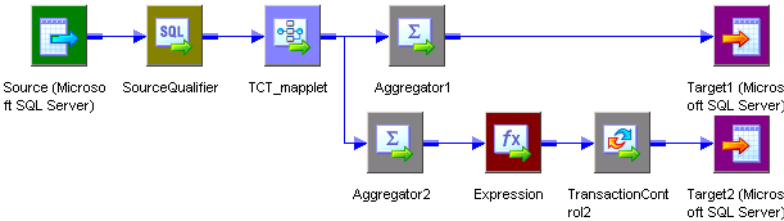


集成服务将处理 TransactionControl1、计算事务控制表达式，并创建事务边界。该映射不包括任何会删除 TransactionControl1 与 Target1 之间的事务边界的转换，因而 TransactionControl1 对 Target1 有效。集成服务将使用由 TransactionControl1 为 Target1 定义的事务边界。

不过，该映射包括一个转换，该转换将删除 TransactionControl1 与 Target2 之间的事务边界，因而 TransactionControl1 对 Target2 无效。当集成服务在“转换范围”设置为“全部输入”的情况下处理 Aggregator2 时，它将删除由 TransactionControl1 定义的事务边界，并输出打开事务中的所有行。随后，集成服务将计算 TransactionControl2、创建事务边界，然后将它们用于 Target2。

如果 TransactionControl1 中发生回滚，则集成服务将仅回滚 Target1 中的行。它不会回滚 Target2 中的任何行。

下图显示了一个无效映射，其中同时包含一个无效和一个有效的事务控制转换：



Mapplet TCT\_mapplet 包含一个事务控制转换。它对 Target1 和 Target2 无效。Aggregator1 转换的“转换范围”属性为“全部输入”。它是 Target1 的一个活动源。Aggregator2 转换的“转换范围”属性为“全部输入”。它是 Target2 的一个活动源。TransactionControl2 转换对 Target2 有效。

该映射无效，因为 Target1 未连接到有效的事务控制转换。

## 映射准则和验证

创建事务控制转换的映射时，请遵循以下规则和准则：

- 如果该映射包含 XML 目标且您选择在提交时附加或创建新文档，则输入组必须接收同一个事务控制点的数据。
- 对于这些目标，连接到关系、XML 或动态 MQSeries 目标之外任何目标的事务控制转换均无效。
- 您必须将每个目标实例都连接到事务控制转换。
- 可以将多个目标连接到一个事务控制转换。
- 只能将一个有效事务控制转换连接到目标。
- 不能在以序列生成器转换开头的管道分支中放置事务控制转换。
- 如果在同一个映射中使用动态查找转换和事务控制转换，回滚事务可能会导致目标数据不同步。
- 事务控制转换可能对一个目标有效，而对另一个目标无效。如果每个目标都连接到了有效的事务控制转换，则映射有效。
- 应将映射中的所有目标都连接到有效的事务控制转换，或映射中的任何目标都不连接到有效的事务控制转换。

## 创建事务控制转换

按照以下步骤向映射添加事务控制转换：

1. 在 Mapping Designer 中，单击“转换”>“创建”。选择事务控制转换。
2. 输入转换的名称。  
事务控制转换的命名约定是 `TC_TransformationName`。
3. 输入转换的说明。  
在 Repository Manager 中查看转换详细信息时将显示此说明，使得更便于了解转换的用途。
4. 单击“创建”。  
Designer 将创建事务控制转换。
5. 单击“完成”。
6. 将端口拖动到该转换中。  
Designer 将为您添加的每个端口创建输入/输出端口。
7. 打开“编辑转换”对话框，选择“端口”选项卡。  
可以添加端口、编辑端口名称、添加端口说明和输入默认值。
8. 选择“属性”选项卡。输入定义提交和回滚行为的事务控制表达式。
9. 选择“元数据扩展”选项卡。创建或编辑该事务控制转换的元数据扩展。
10. 单击“确定”。



## 第 30 章

# 联合转换

本章包括以下主题：

- [联合转换概览, 417](#)
- [使用组和端口, 418](#)
- [创建联合转换, 418](#)
- [在映射中使用联合转换, 418](#)

## 联合转换概览

联合转换是一个多输入组转换，可用于将来自多个管道或管道分支的数据合并到一个管道分支。它合并多个源的数据，类似于 UNION ALL SQL 语句合并两个或更多 SQL 语句的结果。联合转换类似于 UNION ALL 语句，不会删除重复的行。联合转换是活动转换。

数据集成服务将并行处理所有输入组。同时，它还将读取连接到联合转换的源，并将数据块推送给联合转换的输入组。联合转换会基于从集成服务接收数据块的顺序处理这些数据块。

可以将异构源连接到联合转换。该转换会将源与匹配端口合并，输出来自一个与输入组具有相同端口的输出组的数据。

使用自定义转换开发联合转换。

## 联合转换的规则和准则

在使用联合转换时，请遵循以下规则和准则：

- 可以创建多个输入组，但只能创建一个输出组。
- 所有输入组和输出组都必须具有匹配的端口。所有组的精度、数据类型和小数位数都必须相同。
- 联合转换不会删除重复的行。要删除重复的行，必须添加其他转换，例如路由器或筛选器转换。
- 联合转换不会生成事务。

## 联合转换组件

当配置联合转换时，请定义以下组件：

- **“转换”选项卡。**可以重命名转换并添加说明。
- **“属性”选项卡。**可以指定跟踪级别。
- **“组”选项卡。**可以创建和删除输入组。Designer 会显示在“端口”选项卡上创建的组。

- “**组端口**”选项卡。可以创建和删除输入组的端口。Designer 会显示在“端口”选项卡上创建的端口。无法修改联合转换中的“端口”选项卡。

## 使用组和端口

一个联合转换具有多个输入组和一个输出组。在“组”选项卡上创建输入组，在“组端口”选项卡上创建端口。

可以在“组”选项卡上创建一个或多个输入组。默认情况下，Designer 会创建一个输出组。不能编辑或删除输出组。

可以通过从转换复制端口来创建端口，也可以手动创建端口。在“组端口”选项卡上创建端口时，Designer 会在每个输入组中创建输入端口，在输出组中创建输出端口。Designer 会使用在“组端口”选项卡上为每个输入和输出端口指定的端口名称，并附加一个数字以使转换中的每个端口名称独一无二。还为每个端口使用相同的元数据（例如数据类型、精度和小数位数）。

“端口”选项卡显示您创建的组和端口。无法“端口”选项卡上编辑组和端口信息。使用“组”和“端口组”选项卡编辑组和端口。

## 创建联合转换

按照以下步骤创建联合转换：

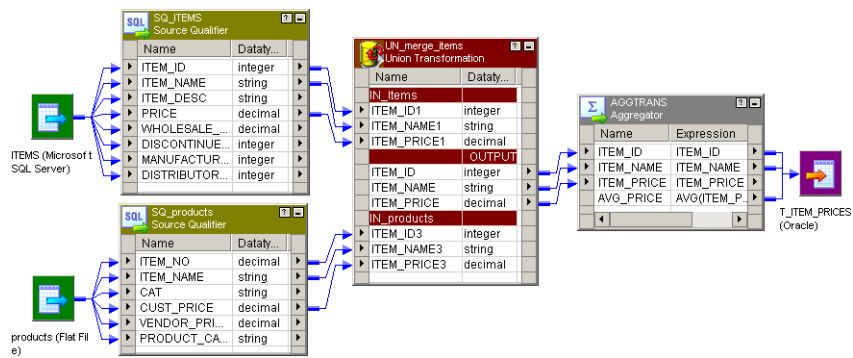
1. 在 Mapping Designer 中，单击“转换”>“创建”。
2. 选择“联合转换”，然后输入转换的名称。  
联合转换的命名约定为 `UN_TransformationName`。
3. 输入转换的说明。单击“创建”，然后单击“完成”。
4. 单击“组”选项卡。
5. 为要合并的每个管道或管道分支添加输入组。  
Designer 会为每个组分配默认名称，但是可以对其重命名。
6. 单击“组端口”选项卡。
7. 为您要合并的每个数据行添加新端口。
8. 输入端口属性，例如名称和数据类型。
9. 单击“属性”选项卡以配置跟踪级别。
10. 单击“确定”。

## 在映射中使用联合转换

联合转换是一个非阻塞式多输入组转换。可以将输入组连接到一个管道中的不同分支或不同源管道。

向某个映射添加联合转换时，必须确认在所有输入组中连接了相同的端口。如果在一个输入组中连接了所有端口，但在另一个输入组中未连接某个端口，则集成服务会向未连接的端口传递空值。

下图显示了包含联合转换的映射：



当映射中的联合转换接收来自单一事务生成器的数据时，集成服务会传播事务边界。当转换接收来自多个事务生成器的数据时，集成服务会丢弃开放式事务中所有的传入事务边界和输出行。

## 第 31 章

# 非结构化数据转换

本章包括以下主题：

- [非结构化数据转换概览, 420](#)
- [配置非结构化数据选件, 421](#)
- [Data Transformation 服务类型, 422](#)
- [非结构化数据转换组件, 422](#)
- [非结构化数据转换端口, 425](#)
- [非结构化数据转换服务名称, 427](#)
- [关系层次结构, 427](#)
- [映射, 428](#)
- [创建非结构化数据转换, 430](#)

## 非结构化数据转换概览

非结构化数据转换是一种可处理非结构化和半结构化文件格式（如消息传递格式、HTML 页面和 PDF 文档）的转换。它还可转换结构化格式，如 ACORD、HIPAA、HL7、EDI-X12、EDIFACT 和 SWIFT。

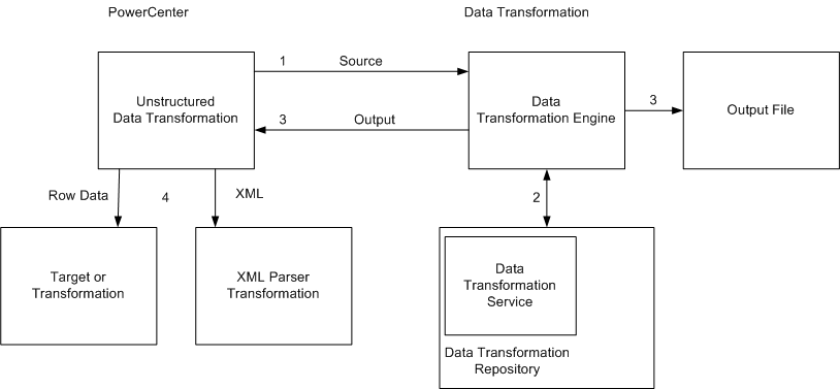
非结构化数据转换从 PowerCenter 会话调用 Data Transformation 服务。Data Transformation 是可转换非结构化和半结构化文件格式的应用程序。您可以将数据从非结构化数据转换传递到 Data Transformation 服务，转换数据，然后将转换后的数据返回到管道。非结构化数据转换可以是主动转换，也可以是被动转换。

Data Transformation 具有以下组件：

- Informatica Developer。提供用于设计和配置转换项目的功能的开发工具。
- Data Transformation 服务。该 Data Transformation 项目已部署到 Data Transformation 存储库并已准备好运行。
- Data Transformation 存储库。用于存储您在 Informatica Developer 中创建的可执行服务的目录。可以将项目部署到其他存储库，如用于测试和生产服务的存储库。
- Data Transformation 引擎。该处理器运行您部署到存储库的服务。

当 Data Transformation 引擎运行服务时，会写入输出数据，或将输出数据返回到集成服务。当 Data Transformation 引擎将输出返回到集成服务时，会返回 XML 数据。可以配置非结构化数据转换以在输出端口中返回 XML，也可以配置输出组以返回行数据。

下图显示 PowerCenter 非结构化数据转换与从 PowerCenter 会话转换数据的 Data Transformation 服务之间的接口。



当调用来自 PowerCenter 会话的 Data Transformation 服务时，会发生以下事件：

1. 非结构化数据转换会将源数据传递到 Data Transformation 引擎。
2. Data Transformation 引擎运行 Data Transformation 服务来转换数据。Data Transformation 服务位于 Data Transformation 存储库文件夹中。
3. Data Transformation 引擎将转换后的数据直接写入到输出文件中，或将转换后的数据返回到非结构化数据转换。
4. 非结构化数据转换会返回 XML 数据或数据行。如果非结构化数据转换返回 XML，请将非结构化数据转换连接到映射中的 XML 解析器转换。

# 配置非结构化数据选件

非结构化数据转换随 PowerCenter 一起安装。Data Transformation 具有单独的安装程序。安装 PowerCenter 后安装 Data Transformation 服务器和客户端组件。

要安装 Unstructured Data 选件，请完成以下步骤：

1. 安装 PowerCenter。
2. 安装 Data Transformation。  
安装 Data Transformation 时，您会收到选择 Java 运行时环境的提示。在提示符下，浏览到 PowerCenter 使用的 JRE。  
默认情况下，安装程序会将 Data Transformation 引擎配置为在进程内运行。要将 Data Transformation 用于非结构化数据转换，请勿将 Data Transformation 配置为在进程外运行。
3. 配置 Data Transformation 存储库文件夹。

## 配置 Data Transformation 存储库目录

Data Transformation 存储库包含可执行的 Data Transformation 服务。在安装 Data Transformation 时，安装程序会在以下文件夹中创建存储库：

<Data\_Transformation\_install\_dir>\DataTransformation\ServiceDB

要创建不同的存储库文件夹位置，请从“开始”菜单打开 Data Transformation 配置。存储库位置在 Data Transformation 配置的以下路径中：

## CM 配置 > CM 存储库 > 文件系统 > 基本路径

如果 Informatica Developer 可以访问远程文件系统，可以将 Data Transformation 存储库更改为远程位置并将服务从 Informatica Developer 直接部署到运行集成服务的系统。

将自定义文件从 <Data\_Transformation\_install\_dir>\DataTransformation\autoInclude\user 目录和 <Data\_Transformation\_install\_dir>\DataTransformation\externLibs\user 目录复制到运行集成服务的节点上的相同目录。

# Data Transformation 服务类型

在 Informatica Developer 中创建数据处理器转换时，请选择一个对象或组件来定义转换服务类型。Data Transformation 具有以下转换数据的服务类型：

- 解析器。将源文档转换为 XML 或 JSON。解析器的输出始终为 XML。输入可为任意格式，如文本、HTML、Word、PDF 或 HL7。
- 序列化程序。将 XML 或 JSON 文件转换为任意格式的输出文档。序列化程序的输出可为任意格式，如文本文档、HTML 文档或 PDF。
- 映射程序和 XMap。将 XML 或 JSON 源文档转换为其他 XML 或 JSON 结构。映射程序处理输入的方式与序列化程序类似，生成输出的方式则与解析器类似。输入和输出是完全结构化的 XML 或 JSON。
- 转换器。修改任意格式的数据。添加、删除、转换或更改文本。可将转换器与解析器、映射程序或序列化程序结合使用。还可以将转换器作为独立的组件运行。
- 流转化器。将较大的输入文档（如几千兆字节的数据流）拆分为多个段。流转化器处理包含多个消息或记录的文档，如 HIPAA 或 EDI 文件。

# 非结构化数据转换组件

非结构化数据转换包含以下选项卡：

- 转换。输入转换的名称和说明。非结构化数据转换的命名约定为 UD\_TransformationName。还可将非结构化数据转换设置为可重用。
- 属性。配置非结构化数据转换常规属性（如 IsPartitionable）且“输出”为“可重复”。
- UDT 设置。修改非结构化数据转换设置，例如输入类型、输出类型和服务名称。
- UDT 端口。配置非结构化数据转换端口和属性。
- 输入层次结构。定义输入组和端口的层次结构，以让非结构化数据转换读取数据行。
- 输出层次结构。定义输出组和端口的层次结构，以让非结构化数据转换将行写入到关系目标。

## “属性”选项卡

在**属性**选项卡上可配置非结构化数据转换的常规属性。

下表介绍了可在**属性**选项卡上配置的属性：

属性	说明
跟踪级别	运行包含此转换的会话时会话日志中包含的详细信息量。默认值为“普通”。
IsPartitionable	转换可在超过一个分区中运行。选择以下选项之一： <ul style="list-style-type: none"><li>- 否。无法对转换进行分区。</li><li>- 本地。转换可以分区，但集成服务必须在同一个节点上运行管道中的所有分区。</li><li>- 在整个网格范围内。转换可以分区，且集成服务可以将每个分区分发到不同节点。</li></ul> 默认选项是“跨整个网格”。
输出是可重复的	会话运行间输出数据的顺序一致。 <ul style="list-style-type: none"><li>- 从不。会话运行之间的输出数据顺序不一致。</li><li>- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。</li><li>- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。</li></ul> 主动转换的默认设置为“从不”。对于被动转换运行，默认设置为“基于输入顺序”。
输出具有确定性	指示转换在多次会话运行时是否生成一致的输出数据。启用此属性可对使用此转换的会话执行恢复操作。

**警告:** 如果将转换配置为可重复的和确定性的，必须确保数据为可重复的和确定性的。如果尝试使用不在会话或恢复之间产生相同数据的转换来恢复会话，恢复过程可能产生受损数据。

## “UDT 设置” 选项卡

在 **UDT 设置**选项卡中配置非结构化数据转换属性。

下表介绍了 **UDT 设置**选项卡上的属性：

属性	说明
InputType	非结构化数据转换传递到 Data Transformation 引擎的输入数据的类型。请选择以下输入类型之一： <ul style="list-style-type: none"><li>- 缓冲区。非结构化数据转换会在 InputBuffer 端口中接收源数据，然后将数据从端口传递到 Data Transformation 引擎。</li><li>- 文件。非结构化数据转换会在 InputBuffer 端口中接收源文件路径，然后将源文件路径传递到 Data Transformation 引擎。Data Transformation 引擎将打开源文件。</li></ul>
OutputType	非结构化数据转换或 Data Transformation 引擎返回的输出数据的类型。请选择以下输出类型之一： <ul style="list-style-type: none"><li>- 缓冲区。除非您配置输出端口的关系层次结构，否则非结构化数据转换会通过 OutputBuffer 端口返回 XML 数据。如果您配置端口的关系层次结构，则非结构化数据转换不会写入到 OutputBuffer 端口。</li><li>- 文件。Data Transformation 引擎会将输出写入到文件。除非您在非结构化数据转换中配置端口的关系层次结构，否则不会将数据返回到非结构化数据转换。</li><li>- 分割。非结构化数据转换会将大型 XML 输出文件分割为适合 OutputBuffer 端口的更小的文件。必须将分割 XML 文件传递到 XML 解析器转换。</li></ul>
ServiceName	要运行的 Data Transformation 服务的名称。该服务必须位于本地 Data Transformation 存储库中。

属性	说明
流转化器块大小	Data Transformation 服务运行流转化器时，非结构化数据转换传递到 Data Transformation 引擎的数据的缓冲区大小。有效值为 1-1,000,000 KB。默认值为 256 KB。
动态服务名称	为每个输入行运行不同的 Data Transformation 服务。启用“动态服务名称”后，非结构化数据转换会在“服务名称”输入端口接收服务名称。 禁用“动态服务名称”后，非结构化数据转换会为每个输入行运行相同的服务。“UDT 设置”中的“服务名称”属性必须包含服务名称。默认为“已禁用”。
状态跟踪级别	设置来自 Data Transformation 服务的状态消息的级别。 - 仅限说明。返回状态代码和简短说明，以指示 Data Transformation 服务成功与否。 - 全部状态。以 XML 的形式从 Data Transformation 服务中返回状态代码和状态消息。 - 无。请不要从 Data Transformation 服务返回状态。默认值为“无”。
允许输入刷新	配置端口的输入组时启用“允许输入刷新”。 当非结构化数据转换具有某个组的所有数据时会创建 XML。配置映射以便转换接收按组排序的数据。 启用输入刷新后，非结构化数据转换会于内存中存储数据并在收到所有组的数据后创建 XML。

## 查看状态跟踪消息

可以查看来自 Data Transformation 服务的状态消息。将状态跟踪级别设置为“仅限说明”或“全部状态”。Designer 会在非结构化数据转换中创建 UDT\_Status\_Code 端口和 UDT\_Status\_Message 输出端口。

当选择“仅限说明”时，Data Transformation 引擎会返回一个状态代码和以下状态消息之一：

状态代码	状态消息
1	成功
2	警告
3	失败
4	错误
5	致命错误

当选择“全部状态”时，Data Transformation 引擎会从 Data Transformation 返回状态代码和错误消息。消息为 XML 格式。



# 非结构化数据转换端口

当创建非结构化数据转换时，Designer 会创建默认端口。它会根据您配置转换的方式创建其他端口。非结构化数据转换输入和输出类型确定非结构化数据转换如何将数据传递到 Data Transformation 引擎以及如何从中接收数据。

下表介绍非结构化数据转换的默认端口：

端口	输入/输出	说明
InputBuffer	输入	输入类型为缓冲区时接收源数据。 输入类型为文件时接收源文件名和路径。
OutputBuffer	输出	输出类型为缓冲区时返回 XML 数据。 输出类型为文件时返回输出文件名。 在配置端口的层次结构输出组时不返回任何数据。

下表介绍在您配置转换时 Designer 创建的其他非结构化数据转换端口：

端口	输入/输出	说明
OutputFileName	输入	当输出类型为文件时接收输出文件的名称。
ServiceName	输入	启用“动态服务名称”时接收 Data Transformation 服务的名称。
UDT_Status_Code	输出	当状态跟踪级别为“仅限说明”或“全部状态”时，从 Data Transformation 引擎返回状态代码。
UDT_Status_Message	输出	当状态跟踪级别为“仅限说明”或“全部状态”时，从 Data Transformation 引擎返回状态消息。

**注意：**可以在输入或输出层次结构选项卡上为关系目标添加输出端口组。配置端口组时，会在 **UDT 端口** 选项卡上显示一条消息，说明层次结构组和端口在另一选项卡上定义。

## 输入和输出类型

输入类型确定集成服务传递到 Data Transformation 引擎的数据类型。输入类型确定输入是数据还是源文件路径。

请配置以下输入类型之一：

- 缓冲区。非结构化数据转换会在 InputBuffer 端口中接收源数据。集成服务将来自 InputBuffer 端口的源行传递到 Data Transformation 引擎。
- 文件。非结构化数据转换会在 InputBuffer 端口中接收源文件路径。集成服务将源文件路径传递到 Data Transformation 引擎。Data Transformation 引擎将打开源文件。请使用文件输入类型解析二进制文件，如 Microsoft Excel 或 Microsoft Word 文件。

如果您未定义输出组和端口，非结构化数据转换将基于输出类型返回数据。

请配置以下输出类型之一：

- 缓冲区。非结构化数据转换将通过 Outputbuffer 端口返回 XML。您必须将 XML 解析器转换连接到 Outputbuffer 端口。

- 文件。Data Transformation 引擎将写入输出文件，而不是将数据传递到集成服务。Data Transformation 引擎将基于 OutputFilename 端口中的文件名对输出文件命名。要将 XML 转换为二进制数据（如 PDF 文件或 Microsoft Excel 文件），请选择“文件”输出类型。  
对于每个源行，集成服务将在 OutputBuffer 端口中返回输出文件名。如果输出文件名为空，则集成服务将返回行错误。出现错误时，集成服务会向 OutputBuffer 端口写入一个空值并返回行错误。
- 分割。非结构化数据转换将 Data Transformation 引擎中的 XML 数据分割为多个段。当非结构化数据转换返回的 XML 文件对于 OutputBuffer 端口而言过大时，请选择“分割”输出类型。配置“分割”输出时，请将 XML 数据传递到 XML 解析器转换。将 XML 解析器转换配置为作为一个 XML 文件处理多个 XML 行。

## 附加非结构化数据转换端口

Data Transformation 服务可能需要多个输入文件、文件名和参数。它可以返回多个输出文件。在创建非结构化数据转换时，Designer 会创建一个 InputBuffer 端口和一个 OutputBuffer 端口。如果需要在非结构化数据转换和 Data Transformation 引擎之间传递附加文件或文件名，请添加输入或输出端口。可以手动添加端口，或通过 Data Transformation 服务添加端口。

下表介绍了可以在 **UDT 端口** 选项卡上创建的端口：

端口类型	输入/输出	说明
其他输入（缓冲区）	输入	接收要传递到 Data Transformation 引擎的输入数据。
其他输入（文件）	输入	接收要通过 Data Transformation 引擎打开的文件名和路径。
服务参数	输入	接收 Data Transformation 服务的输入参数。
其他输出（缓冲区）	输出	从 Data Transformation 引擎接收 XML 数据。
其他输出（文件）	输出	从 Data Transformation 引擎接收输出文件名。
传递	输入/输出	通过非结构化数据转换在不更改数据的情况下传递数据。

## 从 Data Transformation 服务中创建端口

Data Transformation 服务可以请求输入参数、其他输入文件或用户定义的变量。该服务可能会向非结构化数据转换返回多个输出文件。您可以添加端口来传递参数、其他输入文件和其他输出文件。Designer 将在 Data Transformation 服务中创建与这些端口对应的端口。

**注意：**您必须配置一个服务名称，以便从服务中填充端口。

1. 单击非结构化数据转换的**端口**选项卡。
2. 单击**从服务填充**。  
Designer 将显示 Data Transformation 服务中的服务参数、其他输入端口和其他输出端口要求。服务参数包括 Data Transformation 系统变量和用户定义的变量。
3. 选择这些端口以创建每个端口并将其配置为缓冲区端口或文件端口。
4. 单击**填充**以创建您选择的端口。可以选择出现的所有端口。

## 非结构化数据转换服务名称

在创建非结构化数据转换时，Designer 会显示位于 Data Transformation 存储库中的 Data Transformation 服务的列表。选择您要从非结构化数据转换中调用的 Data Transformation 服务的名称。创建转换后可以更改服务名称。服务名称显示在 **UDT 设置** 选项卡中。

要为每个源行运行其他 Data Transformation 服务，请启用“动态服务名称”属性。传递与每个源行相关的服务名称。当启用动态服务名称时，Designer 会创建 ServiceName 输入端口。

启用动态服务名称后，无法从 Data Transformation 服务创建端口。

定义输入端口的关系结构后，无法启用动态服务名称。每个 Data Transformation 服务可能需要不同的输入数据。

## 关系层次结构

您可以定义端口组并为这些组定义关系结构。要构建输入端口的层次结构，在**输入层次结构**选项卡上配置端口。要将行数据传递到关系表或其他目标，在**输出层次结构**选项卡上配置输出端口。

定义输入端口的关系结构时，非结构化数据转换会生成要传递到 Data Transformation 服务的 XML。要提高性能，让 PowerCenter 集成服务将 XML 刷新到非结构化数据转换。启用了输入刷新时，PowerCenter 集成服务会在收到根值的所有数据后刷新来自每个组的 XML。例如，您有一个员工组和一个员工地址组。PowerCenter 集成服务可在数据中每次出现不同员工时同时将两个组的数据刷新到非结构化数据转换。您必须按根组的主键对每个组中的数据进行排序。如果组没有相同的键，您可以将管道中的组相结合。

配置输出组时，输出组代表您要将数据传递到的关系表或目标。Data Transformation 引擎会将行返回到组端口，而不是将 XML 文件写入 OutputBuffer 端口。转换会根据输出类型写入行。

在**输出层次结构**选项卡的左侧窗格中创建组的层次结构。所有组均在根组之下。您无法删除根。每个组可包含端口和其他组。组结构表示目标表之间的关系。在组内定义组时，是在组之间定义一种父子关系。Designer 会在具有生成键的组之间定义主键-外键关系。

选择组可显示该组的端口。您可以在组中添加或删除端口。添加端口时，Designer 会创建默认的端口配置。更改端口名称、数据类型和精度。如果端口必须包含数据，选择“非空”。否则，输出数据是可选的。

在工作区中查看非结构化数据转换时，转换组中的每个端口均具有包含组名称的前缀。

删除组时，会删除组中的端口以及子组。

## 导出层次结构架构

在非结构化数据转换中定义层次结构输出组时，必须在您创建的用于转换数据的 Data Transformation 服务中定义相同的结构。从非结构化数据转换中将层次结构导出为 XML 架构文件。将该架构导入您的数据处理器转换。然后，您可以在此数据处理器转换中将源文档的内容映射到 XML 元素和属性。

要从**关系层次结构**选项卡导出组层次结构，单击**导出到 XML 架构**。选择 XSD 文件的名称和位置。在通过 Information Developer 导入架构时选择您可以访问的位置。

Designer 将使用以下命名空间创建架构：

```
"www.informatica.com/UDT/XSD/<mappingName_<Transformation_Name>>"
```

该架构包括以下注释：

```
<!-- ===== AUTO-GENERATED FILE - DO NOT EDIT ===== -->
<!-- ===== This file has been generated by Informatica PowerCenter ===== -->
```

如果您修改该架构，Data Transformation 引擎所返回数据的格式可能与非结构化数据转换中的输出端口不同。架构中的 XML 元素代表层次结构中的输出端口。可包含空值的列具有 minOccurs=0 和 maxOccurs=1 XML 属性。

## 映射

创建映射时，根据要运行的 Data Transformation 服务的类型对其进行设计。例如，Data Transformation 解析器和映射程序会生成 XML 数据。您可以将非结构化数据转换配置为从 XML 数据中返回行，或将其配置为返回 XML 文件。

Data Transformation 序列化程序组件可从 XML 生成任何输出。它可以生成 HTML 或二进制文件，例如 Microsoft Word 或 Microsoft Excel。输出为二进制数据时，Data Transformation 引擎会将输出写入文件而不是将其传递回非结构化转换。

以下示例显示了如何通过非结构化数据转换配置映射。

### 解析关系表的 Word 文档

可以提取 Microsoft Word 文档中的订单信息，并将订单信息写入到订单表头表和订单详细信息表。配置非结构化数据转换，以调用 Data Transformation 解析器服务和传递要解析的每个 Word 文档的名称。Data Transformation 引擎将打开该 Word 文档，对其进行解析并将行返回到非结构化数据转换。非结构化数据转换将订单表头和订单详细信息传递到关系目标。

映射包含以下对象：

- 源限定符转换。将每个 Microsoft Word 文件名传递到非结构化数据转换。源文件名包含指向包含订单信息的文件的完整路径。
- 非结构化数据转换。输入类型为文件。输出类型为缓冲区。该转换包含订单表头输出组和订单详细信息输出组。这些组包含主键-外键关系。

非结构化数据转换会在 InputBuffer 端口中收到源文件名。它会将名称传递到 Data Transformation 引擎。Data Transformation 引擎将运行解析器服务，以提取 Word 文档中的订单表头和订单详细信息行。Data Transformation 引擎将数据返回到非结构化数据转换。非结构化数据转换将订单表头组和订单详细信息组中的数据传递到关系目标。

- 关系目标。接收来自非结构化数据转换的行。

### 通过 XML 创建 Excel 工作表

可以通过 XML 文件提取员工姓名和地址，并通过姓名列表创建 Microsoft Excel 工作表。

映射具有以下组件：

- XML 源文件。包含员工姓名和地址。
- 源限定符转换。将 XML 数据和输出文件名传递到非结构化数据转换。XML 文件包含员工姓名。
- 非结构化数据转换。输入类型为缓冲区，而输出类型为文件。非结构化数据转换在 InputBuffer 端口中接收 XML 数据，并在 OutputFileName 端口中接收文件名。它会将 XML 数据和文件名传递到 Data Transformation 引擎。

Data Transformation 引擎会运行序列化服务以将 XML 数据转换为 Microsoft Excel 文件。它会根据 OutputFilename 的值，写入具有该文件名的 Excel 文件。

非结构化数据转换仅从 Data Transformation 引擎接收输出文件名。非结构化数据转换 OutputBuffer 端口会返回 OutputFilename 的值。

- 平面文件目标。接收输出文件名。

## 分割 XML 文件输出

Data Transformation 解析器和映射程序组件可以从任何格式转换数据并生成 XML 数据。当 XML 数据很大时，可以将 XML 分割为多个段并将这些段传递到 XML 解析器转换。XML 解析器转换会接收这些段并将 XML 数据作为一个文档进行处理。

当配置非结构化数据转换以分割 XML 输出时，非结构化数据转换会根据 OutputBuffer 端口大小返回 XML。如果 XML 文件大小大于输出端口精度，集成服务会将 XML 分为多个文件，每个文件的大小等于或小于端口大小。XML 解析器转换会解析 XML 并将行传递到关系表或其他目标。

例如，可以从具有 Data Transformation 解析器服务的 Microsoft Word 文档中提取订单表头和详细信息。

映射具有以下组件：

- 源限定符转换。将 Word 文档文件名传递到非结构化数据转换。源文件名包含指向包含订单信息的文件的完整路径。
- 非结构化数据转换。输入类型为文件。输出类型为分割。非结构化数据转换会在 InputBuffer 端口中收到源文件名。它会将文件名传递到 Data Transformation 引擎。Data Transformation 引擎会打开源文件，解析该文件，然后将 XML 数据返回到非结构化数据转换。

非结构化数据转换会接收 XML 数据，将 XML 文件分割为更小的文件，然后将这些段传递到 XML 解析器转换。非结构化数据转换会在大小小于 OutputBuffer 端口大小的段中返回数据。当转换在多个段中返回 XML 数据时，会为每行生成相同的传递数据。无论行成功与否，非结构化数据转换都会在传递端口中返回数据。

- XML 解析器转换。“启用输入流”会话属性已启用。XML 解析器转换会在 DataInput 端口中接收 XML 数据。输入数据已分为多个段。XML 解析器转换会将 XML 数据解析到订单表头和详细信息行。会将订单表头和详细信息行传递到关系目标。会将传递数据返回到筛选器转换。
- 筛选器转换。先删除重复的传递数据，然后再将其传递到关系目标。
- 关系目标。从 XML 解析器转换和筛选器转换中的每个组接收数据。

## 非结构化数据映射的规则和准则

创建非结构化数据映射时，请使用以下规则和准则：

- 配置输出端口的层次结构组时，集成服务会写入到端口组而非写入到 OutputBuffer 端口。不管为转换定义的输出类型为何，集成服务都会写入到端口组。
- 如果非结构化数据转换具有“文件”输出类型，并且尚未定义组输出端口，则必须将 OutputBuffer 端口链接到下游转换。否则，映射无效。当 Data Transformation 服务写入输出文件时，OutputBuffer 端口包含输出文件名。
- 启用“动态服务名称”以将服务名称传递到在“服务名称”输入端口中的非结构化数据转换。启用“动态服务名称”时，Designer 会创建“服务名称”输入端口。
- 必须通过非结构化数据转换配置服务名称，或启用“动态服务名称”选项。否则，映射无效。
- 将 XML 输出从非结构化数据转换链接到 XML 解析器转换。

# 创建非结构化数据转换

使用 PowerCenter Transformation Developer 或 the Mapping Designer 创建非结构化数据转换。

- 1. 在 Mapping Designer 或 Transformation Developer 中，单击**转换 > 创建**。
- 2. 选择**非结构化数据转换**作为转换类型。
- 3. 输入转换的名称。
- 4. 单击**创建**。  
此时将显示**非结构化数据转换**对话框。
- 5. 配置以下属性：

属性	说明
服务名称	您希望使用的 Data Transformation 服务的名称。Designer 将显示 Data Transformation 存储库文件夹中的 Data Transformation 服务。如果计划启用动态服务名称，请勿选择名称。创建转换后，可以在“UDT 设置”选项卡上添加服务名称。
输入类型	说明 Data Transformation 引擎如何接收输入数据。默认为缓冲区。
输出类型	说明 Data Transformation 引擎如何返回输出数据。默认为缓冲区。

- 6. 单击**确定**。
- 7. 可以在 **UDT 设置**选项卡上更改服务名称、输入和输出类型。
- 8. 在**属性**选项卡上配置非结构化数据转换的属性。
- 9. 如果 Data Transformation 服务包含多个输入或输出文件或其请求输入参数，可以在 **UDT 端口**选项卡上添加端口。也可以在**端口**选项卡上添加传递端口。
- 10. 如果希望返回非结构化数据转换中的行数据而不是 XML 数据，请在**关系层次结构**选项卡上创建输出端口组。
- 11. 如果创建了端口组，从**关系层次结构**选项卡导出说明它们的架构。
- 12. 将该架构导入到 Data Transformation 项目以定义项目输出。

## 第 32 章

# 更新策略转换

本章包括以下主题：

- [更新策略转换概览, 431](#)
- [在映射中标记行, 432](#)
- [为会话设置更新策略, 433](#)
- [更新策略清单, 435](#)

## 更新策略转换概览

更新策略转换是一种活动转换。在设计数据仓库时，需要决定将什么类型的信息存储在目标中。作为目标表设计的一部分，需要确定是保留所有历史数据，还是仅保留最新更改。

例如，您可能拥有目标表 T\_CUSTOMERS，其中包含客户数据。当客户地址更改时，您可能想将原始地址保存在表中，而不是更新客户行的这一部分。在这种情况下，您需要新建一行，其中包含更新后的地址，同时保留包含旧客户地址的原始行。此示例显示了如何将历史信息存储在目标表中。不过，如果您想将 T\_CUSTOMERS 表作为当前客户数据的快照，则需要更新现有客户行，并丢弃原始地址。

您选择的模型将决定您如何处理对现有行的更改。在 PowerCenter 中，可在两个不同级别设置更新策略：

- **在会话中。**在配置会话时，可以指示集成服务以相同方式处理所有行（例如，将所有行视为插入），或者使用编码到会话映射中的指令将行标记为各种不同的数据库操作。
- **在映射中。**在映射中，使用更新策略转换来将行标记为插入、删除、更新或拒绝。

**注意：**还可以使用自定义转换来将行标记为插入、删除、更新或拒绝。

## 设置更新策略

要定义更新策略，请完成以下步骤：

1. 要控制映射中的行标记为插入、更新、删除或拒绝的方式，请向该映射添加更新策略转换。如果想要为针对同一目标的行标记不同的数据库操作，或者如果想要拒绝行，那么更新策略转换至关重要。
2. 定义在配置会话时如何标记行。可以将所有行标记为插入、删除或更新；也可以选择数据驱动选项，在这种情况下，集成服务将遵循编码到会话映射内更新策略转换中的指令。
3. 在配置会话时，为每个目标定义插入、更新和删除选项。可按逐个目标的方式允许或禁止插入和删除，并且可以选择三种不同方式来处理更新。



相关主题：

- “为会话设置更新策略” 页面上 433

# 在映射中标记行

要最大程度地控制更新策略，可向映射中添加更新策略转换。此转换最重要的功能是其更新策略表达式，可用来标记要插入、删除、更新或拒绝的各个行。

下表列出了用于每个数据库操作的常量及其等效数值：

操作	常量	数值
插入	DD_INSERT	0
更新	DD_UPDATE	1
删除	DD_DELETE	2
拒绝	DD_REJECT	3

集成服务会将任何其他值视为插入。

## 转发拒绝的行

可以将更新策略转换配置为将拒绝的行传递至下一个转换或删除这些行。默认情况下，集成服务会将拒绝的行转发到下一个转换。集成服务会标记要拒绝的行，并将其写入会话拒绝文件。如果未选择“转发拒绝的行”，集成服务将删除拒绝的行，并将其写入会话日志文件。

如果启用行错误处理，集成服务会将拒绝的行和删除的行写入行错误日志。集成服务不会生成拒绝文件。如果除行错误日志之外，还要将删除的行写入会话日志，可以启用详细数据跟踪。

## 更新策略表达式

更新策略表达式通常使用转换语言的 IIF 或 DECODE 函数来测试每行，查看其是否满足特定条件。如果满足，则为每行分配一个数字代码，将每行标记为用于特定的数据库操作。例如，如果条目日期在应用日期之后，以下 IIF 语句将设置一行的拒绝标志。否则，其将设置该行的更新标志：

```
IIF( ( ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE )
```

要创建更新策略转换，请执行以下操作：

- 在 Mapping Designer 中，将更新策略转换添加到映射。
- 单击“布局”>“链接列”。
- 从表示您想要传递更新策略转换的数据的其他转换拖动所有端口。

在更新策略转换中，Designer 会创建您拖动的每个端口的副本。Designer 还会将新端口连接到原始端口。更新策略转换中的每个端口都是输入/输出端口的组合。

通常，您会选择针对特定目标的所有列。传递更新策略转换后，会将该信息标记为待更新、插入、删除或拒绝。

- 打开更新策略转换并对其重命名。



更新策略转换的命名约定为 `UPD_TransformationName`。

5. 单击“属性”选项卡。
6. 单击“更新策略表达式”字段中的按钮。  
此时将显示表达式编辑器。
7. 输入更新策略表达式以将行标记为插入行、删除行、更新行或拒绝行。
8. 验证该表达式并单击“确定”。
9. 单击“确定”。
10. 将更新策略转换中的端口连接到其他转换或目标实例。

## 汇总器和更新策略转换

将汇总器和更新策略转换连接起来作为同一管道的一部分时，请将汇总器置于更新策略转换之前。集成服务将按照此顺序执行汇总计算，然后将包含此计算结果的行标记为插入、更新、删除或拒绝。

如果将更新策略置于汇总器转换之前，则必须考虑汇总器转换如何处理标记为不同操作的行。集成服务将按照此顺序将行标记为插入、更新、删除或拒绝，然后再执行汇总计算。行的标记方式将决定汇总器转换如何处理该行中用于计算的值。例如，如果将某行标记为删除，然后使用该行计算总和，则集成服务将减去此行中的值。如果将某行标记为拒绝，然后使用该行计算总和，则集成服务不会包括此行中的值。如果将某行标记为插入或更新，然后使用该行计算总和，则集成服务会将此行中的值添加到总和中。

## 查找和更新策略转换

当创建的映射包含使用动态查找缓存的查找转换时，必须使用更新策略转换标记用于目标表的行。使用更新策略转换和动态查找缓存配置会话时，必须定义特定会话属性。

必须将“将源行视为”选项定义为“数据驱动”。在会话属性的“属性”选项卡上指定此选项。

还必须定义以下更新策略目标表选项：

- 选择插入
- 选择更新时更新
- 不要选择删除

这些更新策略目标表选项可确保集成服务更新标记要更新的行，插入标记要插入的行。

如果不选择“数据驱动”，集成服务会为您在“将源行视为”选项中指定的数据库操作标记所有行，而且不会在映射中使用更新策略转换来标记行。集成服务不会插入和更新正确的行。如果不选择“更新时更新”，集成服务不会正确更新目标表中标记为要更新的行。因此，查找缓存和目标表可能会不同步。

# 为会话设置更新策略

在配置会话时，可以选择多个选项用于处理数据库操作，包括更新。

## 为所有行指定一项操作

在配置会话时，可以使用“将源行视为”设置为所有行选择一项数据库操作。

下表显示了“将源行视为”设置的选项：

设置	说明
插入	将所有行视为插入。如果插入行违反数据库中的主键或外键约束，则集成服务将拒绝该行。
删除	将所有行视为删除。对于每一行，如果集成服务在目标表中找到了相对应的行，则集成服务将删除该行。请注意，存储库内的目标定义中必须存在主键约束。
更新	将所有行视为更新。对于每一行，集成服务将在目标表中寻找匹配的主键值。如果存在匹配的主键值，则集成服务将更新该行。目标定义中必须存在主键约束。
数据驱动	集成服务将遵循编码到会话映射内更新策略转换和自定义转换中的指令，来确定如何将行标记为插入、删除、更新或拒绝。 如果会话的映射包含更新策略转换，则默认情况下，此字段将被标记为“数据驱动”。 如果在映射包含更新策略转换或自定义转换时没有选择“数据驱动”，则 Workflow Manager 将显示一条警告。在运行会话时，集成服务不会遵循映射内更新策略转换或自定义转换中的指令来确定如何标记行。

下表介绍了各项设置的更新策略：

设置	用于
插入	首次填充目标表，或者保留历史记录数据仓库。在后一种情况中，必须为整个数据仓库设置此策略，而不仅是一组选定的目标表。
删除	清除目标表。
更新	更新目标表。无论数据仓库包含历史数据还是快照，均可选择此项设置。随后，在配置如何更新各个目标表时，可以确定是将更新后的行作为新行插入，还是使用更新后的信息来修改目标中的现有行。
数据驱动	可对如何将行标记为插入、删除、更新或拒绝进行更精确的控制。如果偶尔需要为针对同一个表的行标记一项操作（例如，更新），或者标记其他操作（例如，拒绝），则请选择此设置。此外，此设置还提供了可以将行标记为拒绝的唯一方式。

## 为各个目标表指定操作

在确定了如何处理会话中的所有行后，还需要为各个目标设置更新策略选项。可在会话属性“映射”选项卡上的“转换”视图中定义更新策略选项。

可以设置以下更新策略选项：

- **插入。**选择此选项可向目标表中插入一行。
- **删除。**选择此选项可从表中删除一行。
- **更新。**在这种情况下，可以选择以下选项：
  - **更新为更新。**更新已标记为更新的每一行（如果该行存在于目标表中）。
  - **更新为插入。**插入已标记为更新的每一行。
  - **更新否则插入。**如果存在相应的行，则更新该行。否则插入该行。
- **截断表。**选择此选项可在加载数据之前截断目标表。

# 更新策略清单

选择更新策略需要在会话中设置适当的选项，并有可能需要向映射中添加更新策略转换。本节总结了实施不同版本的更新策略需要哪些条件。

- 仅向目标表中执行插入。

在配置会话时，为“将源行视为”会话属性选择“插入”。另外，还要确保为会话中的所有目标实例选择“插入”选项。

- 删除目标表中的所有行。

在配置会话时，为“将源行视为”会话属性选择“删除”。另外，还要确保为会话中的所有目标实例选择“删除”选项。

- 仅对目标表中的内容执行更新。

在配置会话时，为“将源行视为”会话属性选择“更新”。在为每个目标表实例配置更新选项时，确保为每个目标实例选择“更新”选项。

- 通过针对同一目标表的不同行执行不同的数据库操作。

向映射中添加更新策略转换。在编写转换更新策略表达式时，使用 DECODE 或 IIF 函数将行标记为不同的操作（插入、删除、更新或拒绝）。在配置使用此映射的会话时，为“将源行视为”会话属性选择“数据驱动”。确保为每个目标表实例选择“插入”、“删除”或其中一个“更新”选项。

- 拒绝数据。

向映射中添加更新策略转换。在编写转换更新策略表达式时，使用 DECODE 或 IIF 指定拒绝行的条件。在配置使用此映射的会话时，为“将源行视为”会话属性选择“数据驱动”。

## 第 33 章

# XML 转换

本章包括以下主题：

- [XML 源限定符转换, 436](#)
- [XML 解析器转换, 436](#)
- [XML 生成器转换, 437](#)

## XML 源限定符转换

可以将 XML 源限定符转换添加到映射中，方法是将 XML 源定义拖动到 Mapping Designer 工作区中，或者手动创建一个 XML 源定义。在将 XML 源定义添加到映射中时，需要将其连接到 XML 源限定符转换。XML 源限定符转换将定义数据元素，集成服务将在执行会话时读取这些数据元素。它可确定 PowerCenter 如何读取源数据。XML 源限定符转换是一种活动转换。

XML 源限定符转换始终拥有一个输入或输出端口，用于 XML 源中的每一列。创建源定义的 XML 源限定符转换时，Designer 将 XML 源定义中的每个端口链接到 XML 源限定符转换中的端口。您不能删除或编辑这些链接。如果从映射中删除 XML 源定义，Designer 也会删除对应的 XML 源限定符转换。可以将某一 XML 源定义连接到某一 XML 源限定符转换

您可以将一个 XML 源限定符组的端口链接其他转换的端口以形成独立的数据流。但是，您不能将一个 XML 源限定符转换中的多个组的端口链接到同一目标转换中的端口。

您可以编辑部分属性并将元数据扩展添加到 XML 源限定符转换。

## XML 解析器转换

可以使用 XML 解析器转换在管道内部提取 XML。借助 XML 解析器转换，可以从消息传送系统（如 TIBCO 或 MQ Series）以及其他源（如文件或数据库）中提取 XML 数据。XML 解析器转换功能与 XML 源功能相似，唯一不同之处在于前者在管道中解析 XML。例如，您可能想要从 TIBCO 源中提取 XML 数据，然后将这些数据传递到关系目标中。XML 解析器转换是一种活动转换。

XML 解析器转换可从一个输入端口读取 XML 数据，然后将数据写入到一个或多个输出端口。

# XML 生成器转换

可以使用 XML 生成器转换在管道内部创建 XML。借助 XML 生成器转换，可以从消息传送系统（如 TIBCO 或 MQ Series）或其他源（如文件或数据库）中读取数据。XML 生成器转换功能与 XML 目标功能相似，唯一不同之处在于前者在管道中生成 XML。例如，您可能想要从关系源中提取数据，然后将 XML 数据传递到目标中。XML 生成器转换是一种活动转换。

XML 生成器转换从多个端口接受数据，然后通过一个输出端口写入 XML。

# 索引

## A

- ABORT 函数
  - 使用 [38](#)
- address.dic
  - 屏蔽数据 [135](#)
- API 方法
  - Java 转换 [212](#)
- API 函数
  - 自定义转换 [84](#)
- ASCII
  - 外部过程转换 [144](#)
  - 自定义转换 [55](#)
- ASCII 模式
  - 为联接器转换配置排序顺序 [247](#)
- AutoCommit
  - “SQL 设置”选项卡 [383](#)

## B

- “帮助程序代码”选项卡
  - Java 转换 [205](#)
  - 示例 [237](#)
- “帮助程序”选项卡
  - Java 转换 [205](#)
- BankSoft 示例
  - 概览 [145](#)
  - Informatica 外部过程 [154](#)
- 保持行顺序
  - 序列生成器转换 [338](#)
- 保留字
  - 查找查询 [267](#)
  - resword.txt [351](#)
  - 生成 SQL [351](#)
- 包装器类
  - 用于已有的库或函数 [164](#)
- 被动模式
  - SQL 转换 [372](#)
- 被动转换
  - 表达式 [141](#)
  - 存储过程 [396](#)
  - 概览 [25](#)
  - Java [197](#), [198](#)
  - 配置 SQL 转换 [372](#)
  - 序列生成器 [328](#)
  - 查找 [254](#)
- 本地数据类型
  - SQL 转换 [369](#)
- 编程标识符 (属性)
  - 外部过程转换 [145](#)
- 编块转换
  - 说明 [29](#)
- 变量
  - 初始化 [36](#)
  - 存储过程结果, 捕获 [35](#)

- 变量 (续)
  - 端口计算顺序 [35](#)
  - 概览 [34](#)
  - Java 转换 [205-207](#)
- 变量端口
  - 概览 [34](#)
- 编译
  - Java 转换 [210](#)
  - Windows 系统上的 DLL [159](#)
  - 自定义转换过程 [71](#)
- 编译错误
  - 标识 Java 转换中的来源 [211](#)
- 表
  - 创建键关系 [353](#)
- 表达式
  - 存储过程转换规则 [410](#)
  - 调用查找 [273](#)
  - 调用存储过程 [405](#)
  - 返回值 [30](#)
  - 非汇总 [50](#)
  - 更新策略 [432](#)
  - 汇总器转换 [48](#)
  - Java 转换 [224](#)
  - 简化 [34](#)
  - 筛选条件 [175](#)
  - 输入 [30](#), [31](#)
  - 验证 [32](#)
- 表达式编辑器
  - 概览 [31](#)
  - 使用 Java 表达式 [226](#)
  - 使用表达式编辑器验证表达式 [32](#)
  - 语法颜色 [32](#)
- 表达式屏蔽
  - 规则和准则 [128](#)
  - 可重复屏蔽 [127](#)
  - 可重复屏蔽示例 [127](#)
  - 说明 [126](#)
- 表达式转换
  - 创建 [142](#)
  - 概览 [141](#)
  - 路由数据 [142](#)
  - 使用变量 [34](#)
- BigDecimal 数据类型
  - Java 转换 [209](#)
- 并发缓存
  - 请参阅缓存 [279](#)
- 必须阻止输入 (属性)
  - Java 转换 [201](#)
- 部分查询
  - 置换 [371](#)
- 本地转换
  - 概览 [25](#)

## C

C/C++

链接到集成服务 [164](#)

参数绑定

SQL 转换查询 [368](#)

参数访问函数

64 位 [170](#)

说明 [169](#)

拆分 XML

非结构化数据转换 [429](#)

常量

替换空值 [38](#)

插入 Else 更新 (属性)

说明 [294](#)

查询

查找转换 [265](#)

替代查找 [266](#)

源限定符转换 [351](#), [354](#)

查询模式

SQL 转换 [368](#)

规则和准则 [372](#)

查找 SQL 替代

动态缓存 [291](#)

查找表

索引 [259](#), [275](#)

名称 [261](#)

查找查询

保留字 [267](#)

默认查询 [266](#)

ORDER BY [266](#)

说明 [265](#)

Sybase ORDER BY 限制 [266](#)

替代 [266](#)

查找端口

说明 [260](#)

查找属性

在会话中配置 [264](#)

查找条件

定义 [260](#)

概览 [269](#)

配置 [269](#)

数据屏蔽转换 [119](#)

持久性查找缓存

从数据库重新缓存 [279](#)

概览 [280](#)

共享 [281](#)

命名和未命名 [280](#)

命名文件 [281](#)

池中连接的最大数量 (属性)

SQL 转换 [383](#)

重新绑定数据类型函数

自定义转换函数 [94](#)

重新初始化查找缓存

请参阅从数据库重新缓存 [279](#)

重置

序列生成器转换 [338](#)

传递端口

默认值 [36](#)

添加到 SQL 转换 [371](#)

创建

表达式转换 [142](#)

COM 外部过程 [147](#)

存储过程转换 [401](#)

Custom transformation: 自定义转换 [63](#)

等级转换 [321](#)

端口 [29](#)

更新策略转换 [432](#)

创建 (续)

汇总器转换 [52](#)

Informatica 外部过程 [154](#)

可重用转换 [43](#)

可重用转换的不可重用实例 [44](#)

联合转换 [418](#)

联接器转换 [252](#)

路由器转换 [327](#)

筛选器转换 [176](#)

序列生成器转换 [340](#), [341](#)

转换 [28](#)

自定义转换 [55](#)

序列数据对象 [338](#)

初始化

变量 [36](#)

集成服务变量支持 [168](#)

外部过程 [166](#)

自定义转换过程 [63](#)

初始化函数

自定义转换 [79](#)

初始化属性中的

进程变量 [168](#)

出现次数属性

规范器转换 [304](#)

CLASSPATH

Java 转换, 配置 [208](#)

COBOL

VSAM 规范器转换 [306](#)

COBOL 源定义

创建规范器转换 [309](#)

OCCURS 语句 [306](#)

COM 服务器

COM 外部过程的类型 [148](#)

COM 外部过程

创建 [147](#)

创建目标 [151](#)

创建源 [151](#)

初始化 [166](#)

返回值 [163](#)

分发 [161](#)

服务器类型 [148](#)

概览 [147](#)

开发说明 [162](#)

内存管理 [164](#)

使用存储库注册 [151](#)

数据类型 [162](#)

添加到存储库 [151](#)

调试 [164](#)

未连接 [166](#)

行级过程 [163](#)

异常处理 [164](#)

与 Informatica 外部过程比较 [145](#)

在 Visual Basic 中开发 [153](#)

在 Visual C++ 中开发 [148](#), [151](#)

company\_names.dic

屏蔽数据 [135](#)

从数据库重新缓存

概览 [279](#)

命名缓存 [280](#)

未命名缓存 [280](#)

存储表

表达式屏蔽 [127](#)

置换数据屏蔽 [118](#)

存储过程

处理顺序, 指定 [398](#)

错误处理 [408](#)

导入 [402](#)

定义 [396](#)

## 存储过程 (续)

- 更改参数 [404](#)
- 会话错误 [409](#)
- 会话后错误 [409](#)
- 会话前错误 [408](#)
- IBM DB2 示例 [401](#)
- Informix 示例 [400](#)
- 加载类型 [407](#)
- Microsoft 示例 [401](#)
- Oracle 示例 [400](#)
- 受支持的数据库 [409](#)
- Sybase 示例 [401](#)
- 特定于数据库的语法注意事项 [400](#)
- Teradata 示例 [401](#)
- 为会话前或会话后运行创建会话 [407](#)

- 写入 [399](#)

- 写入变量 [35](#)

- 设置类型 [403](#)

## 存储过程转换

- 表达式规则 [410](#)

- 导入存储过程 [402](#)

- 返回值 [397](#)

- 概览 [396](#)

- 故障排除 [411](#)

- 会话前和会话后 [407](#)

- 会话前或会话后运行 [407](#)

- 会话运行时, 指定 [398](#)

- 配置 [399](#)

- 配置未连接存储过程 [405](#)

- 配置已连接存储过程 [405](#)

- 手动创建 [402](#), [403](#)

- 输出数据 [397](#)

- 输入/输出参数 [397](#)

- 输入数据 [397](#)

- 通过导入创建 [401](#)

- 未连接 [397](#), [405](#)

- 性能提示 [410](#)

- 修改 [404](#)

- 已连接 [397](#)

- 指定何时运行 [398](#)

- 状态代码 [397](#)

- 存储过程类型 (属性) [403](#)

- 调用文本 (属性) [403](#)

- 跟踪级别 (属性) [403](#)

- 连接信息 (属性) [403](#)

- 日期时间端口的子秒精度 [403](#)

- 设置选项 [403](#)

- 输出具有确定性 (属性) [403](#)

- 输出是可重复的 (属性) [403](#)

- 属性 [403](#)

- 执行顺序 (属性) [403](#)

## 存储加密键

- 数据屏蔽转换 [133](#)

## 存储库

- COM 外部过程 [151](#)

- 注册 COM 过程 [151](#)

## 存储提交时间间隔

- 数据屏蔽转换 [133](#)

## 错误

- 处理 [40](#)

- 有动态查找缓存 [296](#)

- 在表达式编辑器中验证 [32](#)

- 增加 Java 转换中的阈值 [217](#)

## 错误处理

- 对于存储过程 [408](#)

- 有动态查找缓存 [296](#)

## 错误计数

- 为 Java 转换递增 [217](#)

## 错误消息

- 对于外部过程 [164](#)

- 跟踪外部过程 [164](#)

## 错误行

- SQL 转换 [379](#)

## CURRRVAL 端口

- 序列生成器转换 [332](#)

## Custom transformation: 自定义转换

- 创建 [63](#)

- 创建端口 [56](#)

- 创建组 [56](#)

- 初始化属性 [63](#)

- 端口属性 [58](#)

- “更新策略”属性 [60](#)

- 规则和准则 [56](#)

- 过程属性 [63](#)

- 将行传递到过程 [78](#)

- 生成代码文件 [65](#)

- 设置更新策略 [60](#)

- 事务边界 [61](#)

- 事务控制 [60](#)

- 线程 [60](#)

- 元数据扩展 [63](#)

- “转换范围”属性 [60](#)

## 查找 SQL 替代选项

- 减少缓存大小 [266](#)

- 映射参数和变量 [266](#)

- 说明 [261](#)

## 查找缓存

- 持久性 [280](#)

- 处理第一个和最后一个值 [270](#)

- 从数据库重新缓存 [279](#)

- 定义 [277](#)

- 动态 [287](#)

- 动态, 错误阈值 [296](#)

- 动态, 与目标同步 [296](#)

- 动态, 与查找源同步 [299](#)

- 概览 [277](#)

- 共享 [281](#), [282](#)

- 共享未命名查找 [282](#)

- 静态 [281](#)

- 拒绝加载 [296](#)

- 命名持久性缓存 [281](#)

- 替代 ORDER BY [266](#)

- 未命名缓存的分区准则 [282](#)

- 查找缓存持久性属性 [261](#)

- 查找数据缓存大小属性 [261](#)

- 启用缓存 [261](#)

- 预构建缓存 [261](#)

## 查找缓存持久性 (属性)

- 说明 [261](#)

## 查找缓存目录名称 (属性)

- 说明 [261](#)

## 查找源筛选器

- 限制查找 [268](#)

- 说明 [261](#)

## 查找转换

- 表达式 [273](#)

- 不可重用管道 [275](#)

- 查找端口 [260](#)

- 持久性缓存 [280](#)

- 创建已连接查找 [274](#)

- condition [272](#)

- 从数据库重新缓存 [279](#)

- 错误阈值 [296](#)

- 动态缓存 [287](#)

- 端口 [260](#)

- 多个匹配项 [270](#)



## 查找转换 (续)

- 返回多个行 [271](#)
- 更新策略组合 [433](#)
- 管道查找示例 [256](#), [257](#)
- 关联的输入端口 [289](#)
- 缓存 [277](#)
- 缓存共享 [281](#), [282](#)
- 忽略空值 [290](#)
- 拒绝加载 [296](#)
- 可重用管道 [275](#)
- 连接 [257](#), [258](#)
- 命名持久性缓存 [281](#)
- 默认查询 [265](#)
- 配置管道查找会话属性 [265](#)
- 平面文件查找 [254](#), [255](#)
- 使动态缓存与查找源同步 [299](#)
- 使动态缓存与目标同步 [296](#)
- 输出端口 [260](#)
- 输入端口 [260](#)
- 输入自定义查询 [267](#)
- 条件 [269](#)
- 替代默认查询 [266](#)
- 未连接 [257](#), [259](#), [271](#)
- 性能提示 [275](#)
- 序列 ID [289](#)
- 映射参数和变量 [266](#)
- 在比较中忽略 [291](#)
- 在映射中使用管道 [257](#)
- 组件 [259](#)
- 查找源 [254](#)
- 返回值 [272](#)
- 概览 [254](#)
- 管道查找说明 [254](#)
- 日期时间端口的子秒精度 [261](#)
- 属性 [261](#)
- 存储过程类型 (属性)
  - 存储过程转换 [403](#)

## D

- 代码段
  - 为 Java 转换创建 [204](#)
- 代码页
  - 访问函数 [172](#)
  - 外部过程转换 [144](#)
  - 自定义转换 [55](#)
- 代码页 ID
  - 自定义转换, 更改 [103](#)
- 带引号的标识符
  - 保留字 [351](#)
- 带有 TX 前缀的文件
  - 外部过程 [157](#)
- 当前值
  - 序列生成器转换 [336](#)
- “导入包”选项卡
  - Java 转换 [205](#)
  - 示例 [237](#)
- “导入”选项卡
  - Java 转换 [205](#)
- Data Transformation
  - 存储库位置 [421](#)
  - 概览 [420](#)
- DB2
  - 请参阅 IBM DB2 [373](#)
- defineJExpression
  - Java 表达式 API 方法 [230](#)

## defineJExpression 方法

- Java 转换 [214](#)
- 等级划分
  - 数据组 [320](#)
  - 字符串值 [319](#)
- 等级转换
  - 创建 [321](#)
  - 定义组 [320](#)
  - 端口 [319](#)
  - 概览 [318](#)
  - RANKINDEX 端口 [320](#)
  - 使用变量 [34](#)
  - 选项 [319](#)
- 调用文本
  - 存储过程, 进入 [407](#)
- 定义
  - 自定义转换中的端口相关性 [57](#)
- DLL (动态链接库)
  - 编译外部过程 [159](#)
- 动态 SQL 查询
  - SQL 转换 [370](#)
  - SQL 转换示例 [386](#)
- 动态查找缓存
  - 查找 SQL 替代 [291](#)
  - 错误阈值 [296](#)
  - 拒绝加载 [296](#)
  - 使用平面文件源 [287](#)
  - 说明 [287](#)
  - 与目标同步 [296](#)
- 动态服务名称
  - 非结构化数据转换 [423](#)
- 动态连接
  - SQL 转换 [373](#)
  - SQL 转换示例 [391](#)
  - 性能注意事项 [375](#)
- 端口
  - 变量端口 [34](#)
  - 查找转换 [260](#)
  - 创建 [28](#), [29](#)
  - Custom transformation: 自定义转换 [56](#)
  - 等级转换 [319](#)
  - 非结构化数据转换 [426](#)
  - 分组依据 [49](#)
  - HTTP 转换 [180](#)
  - 汇总器转换 [47](#)
  - Java 转换 [200](#)
  - 计算顺序 [35](#)
  - 联合转换 [418](#)
  - 路由器转换 [326](#)
  - 默认值概览 [36](#)
  - 配置 [28](#), [29](#)
  - 序列生成器转换 [329](#)
  - 已排序 [51](#), [362](#)
  - 已排序端口选项 [362](#)
  - 源限定符 [362](#)
- 端口属性
  - 编辑 [58](#)
  - 概览 [58](#)
- 端口相关性
  - 自定义转换 [57](#)
- 端口值
  - Java 转换 [201](#)
- 对存储进行加密
  - 数据屏蔽转换 [133](#)
- 对输出排序 (属性)
  - Java 转换 [201](#)
- 多组
  - 转换 [29](#)

- 当前值（属性）
  - 序列生成器转换 [334](#)
- 导出到 XML 架构
  - 非结构化数据转换 [427](#)
- 调用文本（属性）
  - 存储过程转换 [403](#)
- 动态查找缓存（属性）
  - 说明 [261](#)
- 多个匹配项
  - 查找转换 [270](#)
  - 查找策略属性 [261](#)
- 多项匹配时的查找策略（属性）
  - 说明 [261](#)

## E

- EDatatype 类
  - Java 表达式 [229](#)
- ERROR 函数
  - 使用 [38](#)

## F

- failSession 方法
  - Java 转换 [214](#)
- 方法
  - Java 转换 [206](#), [207](#)
  - Java 转换 API [212](#)
- 返回端口
  - 查找转换 [260](#), [272](#)
- 返回多个行
  - 查找转换 [271](#)
- 返回值
  - 从外部过程 [163](#)
  - 存储过程转换 [397](#)
  - 查找转换 [272](#)
- 范围
  - 屏蔽数值 [125](#)
- 非汇总表达式
  - 概览 [50](#)
- 非汇总函数
  - 示例 [49](#)
- 非用户代码错误
  - 在 Java 转换中 [211](#)
- 分发
  - 外部过程 [161](#)
  - 自定义转换过程 [55](#)
- 分派函数
  - 说明 [168](#)
- 分区相关函数
  - 说明 [172](#)
- 分组依据端口
  - 非汇总表达式 [50](#)
  - 汇总器转换 [49](#)
  - 使用默认值 [50](#)
- firstnames.dic
  - 屏蔽数据 [135](#)
- 复合键
  - 使用序列生成器转换创建 [331](#)
- 非本地转换
  - 概览 [25](#)
- 非结构化数据转换
  - 拆分 XML 输出 [429](#)
  - 从 Data Transformation 填充端口 [426](#)
  - 端口 [426](#)
  - InputBuffer 端口 [425](#)

- 非结构化数据转换 (续)
  - OutputBuffer 端口 [425](#)
  - OutputFileName 端口 [425](#)
  - 配置 [430](#)
  - 设置状态跟踪级别 [424](#)
  - 刷新输入数据 [427](#)
  - “UDT 设置”选项卡 [423](#)
  - 文件输出类型示例 [428](#)
  - 写入到 XML 文件中 [429](#)
  - 写入关系目标 [427](#), [428](#)
  - 组件 [422](#)
  - 导出到 XML 架构 [427](#)
  - 概览 [420](#)

## G

- 高级接口
  - 调用 Java 表达式 [228](#)
  - EDatatype 类 [229](#)
  - Java 表达式 [228](#)
  - JExpression 类 [231](#), [232](#)
  - JExprParaMetadata 类 [229](#)
  - 示例 [231](#)
- 高精度
  - 为 Java 转换启用 [209](#)
- 高级选项
  - SQL 转换 [374](#)
- generateRow 方法
  - Java 转换 [215](#)
- 更新策略
  - 使用 Java 转换进行设置 [203](#)
  - 使用自定义转换进行设置 [60](#)
  - 行策略函数（基于行） [104](#)
- 更新策略转换
  - 查找组合 [433](#)
  - 创建 [432](#)
  - 概览 [431](#)
  - 汇总器组合 [433](#)
  - 配置步骤 [431](#)
  - 清单 [435](#)
  - 输入表达式 [432](#)
  - 为会话设置选项 [433](#), [434](#)
  - 转发拒绝的行 [432](#)
- 更新策略转换（属性）
  - Java 转换 [201](#)
- 更新否则插入（属性）
  - 说明 [295](#), [434](#)
- 更新为插入（属性）
  - 说明 [434](#)
- 更新为更新（属性）
  - 说明 [434](#)
- 跟踪级别
  - 概览 [42](#)
  - 会话属性 [47](#)
  - Java 转换属性 [201](#)
  - 简洁 [42](#)
  - 联接器转换属性 [243](#)
  - 排序器转换属性 [346](#)
  - 普通 [42](#)
  - 替代 [42](#)
  - 外部过程转换属性 [145](#)
  - 详细初始化 [42](#)
  - 详细数据 [42](#)
  - 存储过程转换属性 [403](#)
- 跟踪级别函数
  - 说明 [173](#)

- 跟踪消息
  - 对于外部过程 [164](#)
- getBytes 方法
  - Java 转换 [232](#)
- getDouble 方法
  - Java 转换 [232](#)
- getInRowType 方法
  - Java 转换 [215](#)
- getInt 方法
  - Java 转换 [233](#)
- getLong 方法
  - Java 转换 [233](#)
- getMetada 方法
  - Java 转换 [216](#)
- getResultDataType 方法
  - Java 转换 [233](#)
- getResultMetadata 方法
  - Java 转换 [233](#)
- getStringBuffer 方法
  - Java 转换 [234](#)
- 共享
  - 命名查找缓存 [283](#)
  - 未命名查找缓存 [282](#)
- 共享存储表
  - 数据屏蔽转换 [133](#)
- 工作目录
  - 排序器转换, 指定 [346](#)
- 管道
  - 与联合转换合并 [417](#)
- 管道分区
  - HTTP 转换 [178](#)
  - 自定义转换 [58](#)
- 管道规范器转换
  - 创建 [313](#)
  - “端口”选项卡 [311](#)
  - “规范器”选项卡 [312](#)
  - 说明 [310](#)
- 关联的端口
  - 查找转换 [289](#)
  - 序列 ID [289](#)
- 关系层次结构
  - 非结构化数据转换 [427](#)
- 关系数据库
  - 联接 [242](#)
- 规范器转换
  - 创建 VSAM 规范器转换 [309](#)
  - 创建管道规范器转换 [313](#)
  - 出现次数属性 [304](#)
  - “端口”选项卡 [303](#)
  - 概览 [302](#)
  - 管道规范器 [310](#)
  - 管道规范器 “端口”选项卡 [311](#)
  - “规范器”选项卡 [304](#)
- 故障排除 [316](#)
  - “键类型”属性 [308](#)
- 级别属性 [308, 312](#)
- 生成的键 [305](#)
- 生成的列 ID [303](#)
- 示例 [313](#)
  - “属性”选项卡 [304](#)
- VSAM 规范器 [306](#)
- 映射示例 [313](#)

- 规则
- 默认值 [41](#)
- 故障排除
- 存储过程转换 [411](#)
- 规范器转换 [316](#)
- 汇总器转换 [53](#)

- 故障排除 (续)
  - Java 转换 [210](#)
  - 源限定符转换 [365](#)
- 更新时输出旧值 (属性)
  - 说明 [261](#)
- 管道查找
  - 不可重用转换 [275](#)
  - 查找转换示例 [256](#)
  - 查找转换属性 [256](#)
  - 可重用转换 [275](#)
  - 配置会话属性 [265](#)
  - 映射 [257](#)
  - 映射示例 [257](#)
  - 说明 [254](#)

## H

- “行”转换范围
  - 联接器转换中的行为 [251](#)
- 函数
  - 非汇总 [49](#)
  - 汇总 [48](#)
  - 自定义转换 API [84](#)
- HTTP 转换
  - 创建 [178](#)
  - 身份验证 [178](#)
  - 是否可分区 (属性) [178](#)
  - 特定于线程的代码 [178](#)
  - 响应代码 [177](#)
  - 要求每个分区一个线程属性 [178](#)
  - 基础 URL [183](#)
  - 配置 HTTP 选项卡 [179](#)
  - 配置属性 [179](#)
  - 配置组和端口 [180](#)
  - 示例 [185](#)
  - 组 [180](#)
- 缓存
  - 并发 [279](#)
  - 查找转换 [277](#)
  - 动态查找缓存 [287](#)
  - 共享查找 [281, 282](#)
  - 静态查找缓存 [281](#)
  - 联接器转换 [251](#)
  - 联接器转换中的主行 [251](#)
  - 命名持久性查找 [281](#)
  - 顺序 [279](#)
- 缓存大小
  - 数据屏蔽转换 [133](#)
- 缓存目录
  - 数据屏蔽转换 [133](#)
  - 配置查找缓存目录名称 [261](#)
- 缓存目录 (属性)
  - 联接器转换 [243](#)
- 环境变量
  - Java 包的设置 [208](#)
- 会话
  - \$\$\$SessStartTime [349](#)
  - 存储过程前后, 运行 [407](#)
  - 存储过程转换 [397](#)
  - 配置以处理存储过程错误 [408](#)
  - 设置更新策略 [433](#)
  - 替代选择相异 [363](#)
  - 外部过程转换 [152](#)
  - 增量汇总 [46](#)
- 会话后
  - 存储过程 [407](#)
  - 错误 [409](#)

- 会话前
  - 存储过程 [407](#)
  - 错误 [408](#)
- 会话前和会话后 SQL
  - 源限定符转换 [363](#)
- 会话日志
  - Java 转换 [219](#)
- 会话失败
  - Java 转换 [214](#)
- 汇总函数
  - 概览 [48](#)
  - 空值 [49](#)
  - 列表 [48](#)
  - 在表达式中使用 [48](#)
- 汇总器转换
  - 创建 [52](#)
  - 端口 [47](#)
  - 非汇总函数示例 [49](#)
  - 分组依据端口 [49](#)
  - 概览 [46](#)
  - 更新策略组合 [433](#)
  - 跟踪级别 [47](#)
  - 故障排除 [53](#)
  - 函数列表 [48](#)
  - 空值 [49](#)
  - 嵌套汇总 [49](#)
  - 使用变量 [34](#)
  - STDDEV (标准偏差) 函数 [48](#)
  - VARIANCE 函数 [48](#)
  - 已排序端口 [51](#)
  - 优化性能 [53](#)
  - 与表达式转换比较 [46](#)
  - 与连接器转换一起使用 [248](#)
  - 组件 [46](#)
  - 条件子句示例 [49](#)
- 忽略空值 (属性)
  - 选择 [290](#)
- 缓存文件名
  - 为持久性查找缓存指定 [261](#)
- 缓存文件名前缀
  - 概览 [281](#)
  - 说明 [261](#)
- 缓存值数
  - 序列生成器转换属性 [337](#)
  - 序列生成器属性值 [334](#)

## I

- IBM DB2
  - 连接字符串语法 [373](#)
- IDispatch 接口
  - 定义类 [147](#)
- IDM\_Dictionary
  - 数据屏蔽连接 [120](#)
- IDM\_Storage
  - 数据屏蔽连接 [120](#)
- IIF 函数
  - 使用序列生成器转换替换缺少的键 [332](#)
- incrementErrorCount 方法
  - Java 转换 [217](#)
- Informatica 外部过程
  - 初始化 [166](#)
  - 返回值 [163](#)
  - 分发 [162](#)
  - 开发 [154](#)
  - 开发说明 [162](#)
  - 内存管理 [164](#)

- Informatica 外部过程 (续)
  - 生成 C++ 代码 [157](#)
  - 调试 [164](#)
  - 未连接 [166](#)
  - 行级过程 [163](#)
  - 异常处理 [164](#)
  - 与 COM 比较 [145](#)
- Informix
  - 存储过程注意事项 [400](#)
- InputBuffer 端口
  - 非结构化数据转换 [425](#)
- InputType
  - 非结构化数据转换 [423](#)
- Integration Service: 集成服务
  - 事务边界 [61](#)
- invoke
  - Java 表达式 API 方法 [234](#)
- invokeJExpression
  - API 方法 [227](#)
- invokeJExpression 方法
  - Java 转换 [217](#)
- isNull 方法
  - Java 转换 [218](#)
- isResultNull 方法
  - Java 转换 [234](#)

## J

- Java 包
  - 导入 [205](#)
- Java 表达式
  - 表达式函数类型 [224](#)
  - 创建 [226](#)
  - 调用 [217](#)
  - 调用的规则和准则 [217](#)
  - EDatatype 类 [229](#)
  - 高级接口 [228](#)
  - 高级接口示例 [231](#)
  - 规则和准则 [227, 229](#)
  - invokeJExpression API 方法 [227](#)
  - Java 转换 [224](#)
  - JExpression 类 [231, 232](#)
  - JExprParaMetadata 类 [229](#)
  - 简单接口 [227](#)
  - 简单接口示例 [228](#)
  - 配置 [225](#)
  - 配置函数 [225](#)
  - 生成 [225](#)
  - 生成 Java 代码 [226](#)
  - 使用高级接口调用 [228](#)
  - 使用简单接口调用 [227](#)
  - 使用用户定义的函数 [224](#)
  - 使用转换语言函数 [224](#)
  - 使用自定义函数 [224](#)
  - 在“定义表达式”对话框中创建 [226](#)
  - 在“定义函数”对话框中创建 [226](#)
- Java 表达式 API 方法
  - defineJExpression [230](#)
  - getBytes [232](#)
  - getDouble [232](#)
  - getInt [233](#)
  - getLong [233](#)
  - getResultDataType [233](#)
  - getResultMetadata [233](#)
  - getStringBuffer [234](#)
  - invoke [234](#)
  - isResultNull [234](#)

- Java 代码段
  - 示例 [237](#)
  - 为 Java 转换创建 [204](#)
  - “Java 代码”选项卡
    - 使用 [200](#)
- Java 类路径
  - 会话属性 [208](#)
- Java 转换
  - API 方法 [212](#)
    - “帮助程序代码”选项卡 [205](#)
    - “帮助程序”选项卡 [205](#)
  - 被动 [198](#)
  - 编译 [210](#)
  - 编译错误 [210](#)
  - 标识编译错误来源 [211](#)
  - 必须阻止输入（属性） [201](#)
  - 查找错误 [210](#)
  - 重置变量 [220](#)
  - 创建 Java 代码 [204](#)
  - 创建 Java 代码段 [204](#)
  - 创建端口 [200](#)
  - 创建组 [200](#)
  - 处理子秒 [209](#)
  - 存储元数据 [222](#)
    - “导入包”选项卡 [205](#)
    - “导入”选项卡 [205](#)
  - defineJExpression 方法 [214](#)
  - 调试 [210](#)
  - 对输出排序属性 [201](#)
  - failSession 方法 [214](#)
  - 非用户代码错误 [211](#)
  - 概览 [197](#)
  - generateRow 方法 [215](#)
  - 更新策略（属性） [203](#)
  - 跟踪级别（属性） [201](#)
  - getInRowType 方法 [215](#)
  - getMetadata 方法 [216](#)
  - 会话级别类路径 [208](#)
  - 会话日志 [219](#)
  - 会话失败 [214](#)
  - 获取输入行类型 [215](#)
  - incrementErrorCount 方法 [217](#)
  - invokeJExpression 方法 [217](#)
  - isNull 方法 [218](#)
    - “Java 代码”选项卡 [200](#)
  - Java 基元数据类型 [198](#)
  - 检查空值 [218](#)
  - 检索元数据 [216](#)
  - 解析平面文件 [207](#)
  - 类名称（属性） [201](#)
  - logError 方法 [219](#)
  - logInfo 方法 [219](#)
  - 默认端口值 [201](#)
  - resetNotification 方法 [220](#)
  - 日志 [219](#)
  - setNull 方法 [221](#)
  - 生成事务（属性） [203](#)
  - 生成事务属性 [201](#)
  - 设置 CLASSPATH [208](#)
  - 设置更新策略 [203](#)
  - 设置空值 [221](#)
  - 设置输出行类型 [221](#)
  - 是否可分区（属性） [201](#)
  - 示例 [235](#)
  - 事务控制 [203](#)
  - 输出具有确定性属性 [201](#)
  - 属性 [201](#)
  - storeMetadata 方法 [222](#)
- Java 转换 (续)
  - 要求每个分区一个线程属性 [201](#)
  - 映射失败 [214](#)
  - 用户代码错误 [211](#)
  - 语言（属性） [201](#)
    - “在接收事务中”选项卡 [207](#)
    - “在数据末尾”选项卡 [207](#)
    - “在输入行”选项卡 [206](#)
  - 转换范围（属性） [201, 203](#)
  - 转换级别类路径 [208](#)
  - 主动 [198](#)
  - 输出端口 [201](#)
  - 输入端口 [201](#)
  - 数据类型转换 [198](#)
- Java 转换 API 方法
  - 回滚 [220](#)
  - setOutRowType [221](#)
  - 提交 [213](#)
- Java 基元数据类型
  - Java 转换 [198](#)
- JDK
  - Java 转换 [197](#)
- JExpression 类
  - Java 表达式 [231, 232](#)
- JExprParaMetadata 类
  - Java 表达式 [229](#)
- 键
  - 创建用于联接 [353](#)
  - 使用序列生成器转换创建 [331](#)
  - 源定义 [353](#)
- 简单接口
  - Java 表达式 [227](#)
  - Java 转换 API 方法 [227](#)
  - 示例 [228](#)
- 将源行视为
  - 更新策略 [433](#)
- 简洁跟踪级别
  - 已定义 [42](#)
  - “键类型”属性
  - 规范器转换 [308](#)
- 键屏蔽
  - 屏蔽日期时间值 [117](#)
  - 屏蔽数值 [117](#)
  - 屏蔽字符串值 [116](#)
  - 说明 [116](#)
  - 数值 [116](#)
- 脚本模式
  - 规则和准则 [367](#)
  - SQL 转换 [367](#)
- 脚本区域设置选项
  - SQL 转换 [383](#)
- 加载类型
  - 存储过程 [407](#)
- 加载顺序
  - 源限定符 [349](#)
- 级别属性
  - 管道规范器转换 [312](#)
  - VSAM 规范器转换 [308](#)
- 集成服务
  - 变量支持 [168](#)
  - 处理存储过程出错 [408](#)
  - 汇总数据 [49](#)
  - 数据类型 [162](#)
  - 在调试模式下运行 [164](#)
- 结果字符串替换字符
  - 数据屏蔽转换 [125](#)
- 静态 SQL 查询
  - 配置端口 [369](#)

静态 SQL 查询 (续)

SQL 转换 [368](#)

静态变量

Java 转换 [205](#)

静态查找缓存

概览 [281](#)

静态代码

Java 转换 [205](#)

静态数据库连接

说明 [373](#)

性能注意事项 [375](#)

计算

汇总 [46](#)

使用变量 [35](#)

使用表达式转换 [141](#)

基于数组的函数

概览 [106](#)

设置输入错误行 [111](#)

数据处理 [108](#)

行策略 [110](#)

行是否有效 [108](#)

行数 [107](#)

最大行数 [106](#)

基于行的函数

数据处理 [96](#)

行策略 [104](#)

JRE

Java 转换 [197](#)

句柄

Custom transformation: 自定义转换 [74](#)

局部变量

概览 [34](#)

拒绝文件

更新策略 [432](#)

基础 URL

配置 [183](#)

结束值 (属性)

序列生成器转换 [334](#)

解析器

Data Transformation [422](#)

数据转换输出 [429](#)

## K

开发

COM 外部过程 [147](#)

Informatica 外部过程 [154](#)

可重复的输出

数据屏蔽转换 [115](#)

可重复的相关屏蔽

说明 [121](#)

可重用

序列生成器转换 [338](#)

可重用转换

创建 [43](#)

创建不可重用实例 [44](#)

概览 [43](#)

更改 [44](#)

添加到映射 [44](#)

映射变量 [43](#)

空值

汇总函数 [49](#)

Java 转换设置 [221](#)

排序器转换 [346](#)

筛选 [175](#)

使用常量替换 [38](#)

使用汇总函数替换 [50](#)

空值 (续)

跳过 [39](#)

在 Java 转换中检查 [218](#)

空值排序详情 (属性)

联接器转换 [243](#)

库

用于 C++ 外部过程 [150](#)

用于 Informatica 外部过程 [159](#)

用于 VB 外部过程 [154](#)

## L

类名称 (属性)

Java 转换 [201](#)

联接

创建键关系用于 [353](#)

Informatica 语法 [355](#)

用户定义 [355](#)

源限定符默认 [352](#)

自定义 [353](#)

连接

SQL 转换 [372](#)

连接字符串示例 [373](#)

连接到数据库

SQL 转换 [372](#)

联接类型

联接器属性 [244](#)

普通联接 [245](#)

完整外部联接 [246](#)

详细外部联接 [246](#)

右外部联接 [355](#)

源限定符转换 [355](#)

主外部联接 [245](#)

左外部联接 [355](#)

联接类型 (属性)

联接器转换 [243](#)

联接器缓存

联接器转换 [251](#)

联接器数据缓存大小

联接器转换属性 [243](#)

联接器索引缓存大小

联接器转换属性 [243](#)

联接器转换

包含“全部输入”转换范围的行为 [251](#)

包含“事务”转换范围的行为 [251](#)

包含“行”转换范围的行为 [251](#)

保留事务边界 [251](#)

创建 [252](#)

处理实时数据 [252](#)

从同一源联接数据 [249](#)

概览 [242](#)

“行”转换范围 [251](#)

缓存 [251](#)

联接多个数据库 [242](#)

联接类型 [244](#)

配置联接条件以使用排序源端口 [247](#)

配置排序顺序 [247](#)

“全部输入”转换范围 [251](#)

删除事务边界 [252](#)

实时数据 [251](#)

事务 [251](#)

“事务”转换范围 [251](#)

输入规则 [242](#)

属性 [243](#)

条件 [244](#)

性能提示 [253](#)

与排序器转换一起使用 [247](#)

- 联接器转换 (续)
  - 在 ASCII 模式下配置排序顺序 [247](#)
  - 在 Unicode 模式下配置排序顺序 [247](#)
  - 转换范围 [251](#)
  - 阻止源数据 [250](#)
- 连接设置
  - SQL 转换 [373](#)
- 联接条件
  - 定义 [248](#)
  - 概览 [244](#)
  - 使用排序源端口 [247](#)
- 联接替代
  - 普通联接语法 [356](#)
  - 右外部联接语法 [359](#)
  - 左外部联接语法 [357](#)
- 联接已排序数据
  - 配置以优化联接性能 [247](#)
  - 使用排序器转换 [247](#)
  - 使用已排序关系数据 [247](#)
  - 使用已排序平面文件 [247](#)
- 联接语法
  - 普通联接 [356](#)
  - 右外部联接 [359](#)
  - 左外部联接 [357](#)
- 流转化器块大小
  - 非结构化数据转换 [423](#)
- logError 方法
  - Java 转换 [219](#)
- LogicalConnectionObject
  - 说明 [373](#)
  - SQL 转换示例 [394](#)
- logInfo 方法
  - Java 转换 [219](#)
- 逻辑数据库连接
  - 传递到 SQL 转换 [373](#)
  - 性能注意事项 [375](#)
- 路由器转换
  - 创建 [327](#)
  - 端口 [326](#)
  - 概览 [323](#)
  - 筛选规范器数据 [313](#)
  - 示例 [325](#)
  - 在映射中连接 [327](#)
  - 组 [324](#)
  - 组筛选条件 [325](#)
- 路由行
  - 转换 [323](#)
- 连接变量
  - 使用查找转换 [261](#)
  - 在存储过程转换中使用 [403](#)
- 连接对象
  - 在查找转换中配置 [261](#)
  - 在存储过程转换中配置 [403](#)
- 连接信息 (属性)
  - 查找转换 [261](#)
  - 存储过程转换 [403](#)
- 连接字符串
  - 语法 [373](#)
- 联合转换
  - 创建 [418](#)
  - 端口 [418](#)
  - 组 [418](#)
  - 组件 [417](#)
  - 概览 [417](#)
  - 指导原则 [417](#)
- 流转化器组件
  - Data Transformation [422](#)

## M

- Mapping Designer
  - 添加可重用转换 [44](#)
- MFC AppWizard
  - 概览 [159](#)
- Microsoft SQL Server
  - 存储过程注意事项 [401](#)
  - 连接字符串语法 [373](#)
- 命名持久性查找缓存
  - 概览 [281](#)
  - 共享 [283](#)
- 命名缓存
  - 持久性 [280](#)
  - 从数据库重新缓存 [280](#)
  - 共享 [283](#)
- 模糊
  - 日期值 [126](#)
  - 数值 [126](#)
- 模块 (属性)
  - 外部过程转换 [145](#)
- 默认查询
  - 查看 [351](#)
  - 概览 [351](#)
  - 使用源限定符替代 [354](#)
  - 替代方法 [351](#)
- 默认联接
  - 源限定符 [352](#)
- 默认值
  - 传递端口 [36](#), [37](#)
  - 概览 [36](#)
  - 规则 [41](#)
  - 汇总器按端口分组 [50](#)
  - 输出端口 [36](#), [37](#)
  - 数据屏蔽 [132](#)
  - 输入 [42](#)
  - 输入端口 [36](#), [37](#)
  - 验证 [42](#)
  - 用户定义 [37](#)
- 默认组
  - 路由器转换 [324](#)
- 目标
  - 更新 [431](#)
- 目标表
  - 插入 [435](#)
  - 删除行 [435](#)
  - 设置更新策略 [434](#)
- 目标加载顺序
  - 源限定符 [349](#)
- \$Target
  - 查找转换 [261](#)
  - 存储过程转换 [403](#)

## N

- NaN
  - 转换为 1。#QNAN [365](#)
- 内存管理
  - 对于外部过程 [164](#)
- NEXTVAL 端口
  - 序列生成器 [329](#)
- NumRowsAffected 端口
  - SQL 转换 [378](#)



## O

OCCURS 语句  
  COBOL 示例 [306](#)

Oracle  
  存储过程注意事项 [400](#)  
  连接字符串语法 [373](#)

ORDER BY  
  查找查询 [266](#)  
  替代 [266](#)

OutputBuffer 端口  
  非结构化数据转换 [425](#)

OutputFileName 端口  
  非结构化数据转换 [425](#)

OutputType  
  非结构化数据转换 [423](#)

## P

排序器转换  
  创建 [347](#)  
  概览 [344](#)  
  工作目录 [346](#)  
  配置 [345](#)  
  配置排序器缓存大小 [345](#)  
  属性 [345](#)  
  与联接器转换一起使用 [247](#)

排序顺序  
  汇总器转换 [51](#)  
  为联接器转换配置 [247](#)  
  源限定符转换 [362](#)

排序源  
  定义 [247](#)  
  配置要使用的联接条件 [247](#)

配置  
  端口 [28](#), [29](#)

屏蔽参数  
  数据屏蔽 [115](#)

屏蔽格式  
  屏蔽字符串值 [123](#)

屏蔽规则  
  范围 [125](#)  
  结果字符串替换字符 [125](#)  
  模糊 [125](#)  
  屏蔽格式 [123](#)  
  应用 [123](#)  
  源字符串字符 [124](#)

屏蔽数据  
  查找数据示例 [135](#)  
  地址示例 [135](#)  
  公司名称示例 [135](#)  
  名字示例 [135](#)  
  姓氏示例 [135](#)  
  “屏蔽属性”选项卡  
  数据屏蔽转换 [114](#)

平面文件  
  查找 [255](#)  
  联接数据 [242](#)

平面文件查找  
  说明 [255](#)  
  已排序输入 [255](#)

普通跟踪级别  
  概览 [42](#)

普通联接  
  保留事务边界 [251](#)  
  创建 [356](#)  
  定义 [245](#)

普通联接 (续)  
  语法 [356](#)

## Q

恰好一次传送  
  SQL 转换 [376](#)

起始数字  
  社会保险号 [132](#)

起始值  
  序列生成器转换 [335](#)

启用 XML 输入流  
  XML 解析器会话属性 [429](#)  
  “全部输入”转换范围  
  联接器转换中的行为 [251](#)

缺少的值  
  使用序列生成器进行替换 [332](#)

区分大小写 (属性)  
  排序器转换 [346](#)

取消初始化函数  
  自定义转换 [82](#)

启用查找缓存 (属性)  
  说明 [261](#)

起始值 (属性)  
  序列生成器转换 [334](#)

## R

resetNotification 方法  
  Java 转换 [220](#)

resilience: 弹性  
  SQL 转换数据库 [377](#)

日期时间值  
  数据屏蔽 [117](#)  
  源限定符转换 [349](#)

日期值  
  随机数据屏蔽 [122](#)

日志  
  Java 转换 [219](#)

rollBack  
  Java 转换 API 方法 [220](#)

## S

ScriptError 端口  
  说明 [367](#)

ScriptName 端口  
  SQL 转换 [367](#)

ScriptResults 端口  
  SQL 转换 [367](#)

setNull 方法  
  Java 转换 [221](#)

setOutRowType  
  Java 转换 API 方法 [221](#)

筛选器转换  
  创建 [176](#)  
  概览 [174](#)  
  开发提示 [176](#)  
  示例 [174](#)  
  条件 [175](#)  
  性能提示 [176](#)

筛选行  
  源限定符作为筛选器 [176](#)  
  转换 [174](#), [344](#)



- 筛选源行
  - 查找转换 [268](#)
- 社会保障号
  - 可重复的数据屏蔽 [129](#)
  - 区号屏蔽 [129](#)
- 生成 Java 代码
  - Java 表达式 [226](#)
- 生成的函数
  - 自定义转换 [79](#)
- 生成的键
  - 规范器转换 [305](#)
- 生成的列 ID
  - 规范器转换 [303](#)
- 生成回滚行
  - Java 转换 [220](#)
- 生成事务
  - Java 转换 [203](#), [213](#)
  - 自定义转换 [61](#)
- 生成事务（属性）
  - Java 转换 [201](#)
- 生成输出行
  - Java 转换 [215](#)
- 升级
  - 不可重用转换 [44](#)
- 设置数据代码页函数
  - 自定义转换函数 [103](#)
- 是否活动（属性）
  - Java 转换 [201](#)
- 是否可分区（属性）
  - HTTP 转换 [178](#)
  - Java 转换 [201](#)
  - 外部过程转换 [145](#)
  - 自定义转换 [58](#)
- 实例
  - 创建可重用转换 [43](#)
- 实例变量
  - Java 转换 [205-207](#)
- 实时数据
  - 包含连接器转换 [251](#)
  - 通过连接器转换处理 [252](#)
- 事务
  - 定义 [412](#)
  - 生成 [61](#), [203](#), [213](#)
  - 在连接器转换中使用 [251](#)
- 事务边界
  - Custom transformation：自定义转换 [61](#)
  - 在连接器转换中删除 [252](#)
  - 在联接转换中保留 [251](#)
- 事务控制
  - 表达式 [413](#)
  - Custom transformation：自定义转换 [60](#)
  - 概览 [412](#)
  - Java 转换 [203](#)
  - 示例 [413](#)
  - SQL 转换 [376](#)
  - transformation：转换 [412](#)
- 事务控制转换
  - 创建 [416](#)
  - 概览 [412](#)
  - 属性 [412](#)
  - 无效 [414](#)
  - 映射验证 [416](#)
  - 有效 [414](#)
  - 在映射中 [414](#)
  - “事务”转换范围
    - 连接器转换中的行为 [251](#)
- 使用连接池（属性）
  - SQL 转换 [383](#)
- 双精度数据类型
  - Java 转换 [209](#)
- 输出参数
  - 存储过程 [397](#)
  - 未连接存储过程转换 [405](#)
  - “输出层次结构”选项卡
  - 非结构化数据转换 [427](#)
- 输出端口
  - 表达式转换必需的 [142](#)
  - 错误处理 [36](#)
  - 概览 [29](#)
  - Java 转换 [201](#)
  - 默认值 [36](#)
  - 用作变量 [206](#)
- 输出具有确定性（属性）
  - Java 转换 [201](#)
  - 外部过程转换 [145](#)
  - 存储过程转换 [403](#)
- 输出是可重复的（属性）
  - 外部过程转换 [145](#)
  - 存储过程转换 [403](#)
- 输出行
  - 设置 Java 转换中的行类型 [221](#)
- 数据
  - 存储临时 [34](#)
  - 联接 [242](#)
  - 通过更新策略转换拒绝 [435](#)
  - 选择相异 [363](#)
  - 预先排序 [51](#)
- 数据处理函数
  - 基于数组 [108](#)
  - 基于行 [96](#)
- 数据访问模式函数
  - 自定义转换函数 [85](#)
- 数据集成服务)
  - 重新启动模式 [220](#)
- 数据库
  - 联接数据自不同的 [242](#)
  - 受支持的选项 [409](#)
- 数据库弹性
  - SQL 转换 [376](#)
- 数据库连接
  - SQL 转换 [372](#)
- 数据类型
  - COM [162](#)
  - Java 转换 [198](#)
  - 源限定符 [349](#)
  - 转换 [162](#)
- 数据屏蔽
  - 包含查找转换的映射 [135](#)
  - 通过表达式转换映射 [138](#)
- 数据屏蔽转换
  - 表达式屏蔽 [126](#)
  - 表达式屏蔽准则 [128](#)
  - 存储表 [118](#), [127](#)
  - 存储提交时间间隔 [133](#)
  - 范围 [125](#)
  - 共享存储表 [133](#)
  - 规则和准则 [134](#)
  - 缓存大小 [133](#)
  - 缓存目录 [133](#)
  - 会话属性 [133](#)
  - 可重复 SSN [129](#)
  - 可重复表达式屏蔽 [127](#)
  - 可重复的 SIN 编号 [132](#)
  - 可重复的相关屏蔽 [121](#)
  - 连接要求 [120](#)
  - 模糊 [125](#)

- 数据屏蔽转换 (续)
  - 默认值文件 [132](#)
  - 配置关系字典 [119](#)
  - 屏蔽 IP 地址 [132](#)
  - 屏蔽 URL [132](#)
  - 屏蔽电话号码 [130](#)
  - 屏蔽电子邮件地址 [130](#)
  - 屏蔽格式 [123](#)
  - 屏蔽日期值 [125](#)
  - 屏蔽社会保险号 [131](#)
  - 屏蔽社会保障号 [128](#)
  - 屏蔽属性 [114](#)
  - 使用映射参数 [115](#)
  - 数据屏蔽转换 [133](#)
  - 随机屏蔽 [122](#)
  - 唯一输出 [133](#)
  - 相关数据屏蔽 [120](#)
  - 用于置换屏蔽的字典 [118](#)
  - 源字符串字符 [124](#)
  - 置换屏蔽 [117](#)
  - 置换屏蔽属性 [118](#), [119](#)
  - 字典名称表达式屏蔽 [127](#)
- 数据驱动
  - 概览 [433](#)
- 顺序缓存
  - 查看缓存 [279](#)
  - 请参阅缓存 [279](#)
- 输入
  - 表达式 [30](#), [31](#)
  - SQL 查询替代 [354](#)
  - 用户定义的联接 [355](#)
  - 源筛选器 [361](#)
- 输入/输出端口
  - 概览 [29](#)
- 输入参数
  - 存储过程 [397](#)
  - “输入层次结构”选项卡
  - 非结构化数据转换 [427](#)
- 输入端口
  - 概览 [29](#)
  - Java 转换 [201](#)
  - 默认值 [36](#)
  - 用作变量 [206](#)
- 输入行
  - 获取行类型 [215](#)
- 属性 ID
  - 自定义转换 [88](#)
- 属性访问函数
  - 说明 [169](#)
- 属性函数
  - 自定义转换 [87](#)
- 数值
  - 键屏蔽 [117](#)
  - 随机屏蔽 [122](#)
- SIN 号
  - 可重复的数据屏蔽 [132](#)
  - 屏蔽社会保险号 [131](#)
- Source Analyzer
  - 创建键关系 [353](#)
- SQL
  - 查看默认查询 [351](#)
  - 添加自定义查询 [354](#)
  - 替代默认查询 [351](#), [354](#)
- SQL 查询
  - 查看默认查询 [351](#)
  - 动态更新示例 [386](#)
  - 动态连接示例 [391](#)
  - 添加自定义查询 [354](#)

- SQL 查询 (续)
  - 替代默认查询 [351](#), [354](#)
  - “SQL 端口”选项卡
  - SQL 转换 [383](#)
  - “SQL 设置”选项卡
  - SQL 转换 [383](#)
- SQL 替代
  - 默认查询 [351](#)
- SQL 语句
  - 受 SQL 转换的支持 [384](#)
- SQL 转换
  - 被动模式 [372](#)
  - 本地数据类型列 [369](#)
  - 查询模式 [368](#)
  - 传递端口 [371](#)
  - 传递完整连接信息 [373](#)
  - 动态 SQL 查询 [370](#)
  - 动态查询示例 [386](#)
  - 动态连接示例 [391](#)
  - 高级选项 [374](#)
  - HA 恢复准则 [376](#)
  - 脚本模式 [367](#)
  - 静态 SQL 查询 [368](#)
  - 静态查询端口 [369](#)
  - NumRowsAffected [378](#)
  - 配置连接 [373](#)
    - PM\_REC\_STATE 表
    - SQL 转换恢复 [376](#)
  - 恰好一次消息传送 [376](#)
  - ScriptError 端口 [367](#)
  - ScriptName 端口 [367](#)
  - ScriptResults 端口 [367](#)
  - SELECT 查询 [369](#)
  - 设置 SQL 属性 [383](#)
  - 设置详细数据 [381](#)
  - 事务控制 [376](#)
  - 使用字符串置换 [370](#)
  - 受支持的 SQL 语句 [384](#)
  - 数据库弹性 [376](#)
  - 数据库复原 [377](#)
  - 说明 [366](#)
  - 属性 [381](#)
  - SQL 端口说明 [383](#)
- storeMetadata 方法
  - Java 转换 [222](#)
- 随机屏蔽
  - 屏蔽日期值 [122](#)
  - 屏蔽字符串值 [122](#)
  - 数值 [122](#)
- 索引
  - 查找表 [259](#), [275](#)
  - 查找条件 [275](#)
- surnames.dic
  - 屏蔽数据 [135](#)
- Sybase ASE
  - 存储过程注意事项 [401](#)
  - ORDER BY 限制 [266](#)
  - 连接字符串语法 [373](#)

## T

- TC\_COMMIT\_AFTER 常量
  - 说明 [413](#)
- TC\_COMMIT\_BEFORE 常量
  - 说明 [413](#)
- TC\_CONTINUE\_TRANSACTION 常量
  - 说明 [413](#)

- TC\_ROLLBACK\_AFTER 常量
  - 说明 [413](#)
- TC\_ROLLBACK\_BEFORE 常量
  - 说明 [413](#)
- 特定于线程的操作
  - HTTP 转换 [178](#)
  - 写入 [60](#)
  - 自定义转换 [58](#)
- Teradata
  - 连接字符串语法 [373](#)
- 特殊格式屏蔽
  - 电话号码 [130](#)
  - 电子邮件地址 [130](#)
  - IP 地址 [132](#)
  - 可重复的 SIN 编号 [132](#)
  - 社会保险号 [131](#)
  - 社会保障号 [128](#)
  - URL [132](#)
- 填充端口
  - 非结构化数据转换 [426](#)
- 添加
  - 提交到表达式 [31](#)
  - 组 [326](#)
- 添加静态输出端口（属性）
  - SQL 转换 [383](#)
- 条件
  - 查找转换 [269, 272](#)
  - 连接器转换 [244](#)
  - 路由器转换 [325](#)
  - 筛选器转换 [175](#)
- 调试
  - Java 转换 [210](#)
  - 外部过程 [164](#)
- 替代
  - 默认源限定符 SQL 查询 [354](#)
- 提交
  - Java 转换 API 方法 [213](#)
- TINFParm 参数类型
  - 定义 [165](#)
- 通知函数
  - 自定义转换 [81](#)
- Transformation Developer
  - 可重用转换 [43](#)
- Transformation Exchange (TX)
  - 定义 [144](#)

## U

- UDT 设置
  - 非结构化数据转换 [423](#)
- Unicode 模式
  - 外部过程转换 [144](#)
  - 为连接器转换配置排序顺序 [247](#)
  - 自定义转换 [55](#)
- URL
  - 通过业务文档链接添加 [31](#)

## V

- Vertica
  - 连接字符串语法 [373](#)
- Visual Basic
  - 包装器类 [164](#)
  - COM 数据类型 [162](#)
  - 将函数添加到集成服务中 [164](#)
  - 开发 COM 外部过程 [153](#)

- Visual Basic (续)
  - 手动分发过程 [162](#)
  - 外部过程的代码 [145](#)
  - 应用程序设置向导 [162](#)
- Visual C++
  - 包装器类 [164](#)
  - COM 数据类型 [162](#)
  - 将库添加到集成服务中 [164](#)
  - 开发 COM 外部过程 [148](#)
  - 手动分发过程 [162](#)
- VSAM 规范器转换
  - 创建 [309](#)
  - “端口”选项卡 [308](#)
  - “规范器”选项卡 [308](#)
  - 说明 [306](#)

## W

- 外部过程
  - 分发 [161](#)
  - 分发 Informatica 外部过程 [162](#)
  - 接口函数 [168](#)
  - 开发说明 [162](#)
  - 调试 [164](#)
- 外部过程转换
  - 64 位 [外部过程转换 64] [170](#)
  - ATL 对象 [148](#)
  - BankSoft 示例 [145](#)
  - 包装器类 [164](#)
  - 编程标识符（属性） [145](#)
  - 参数访问函数 [169](#)
  - 成员变量 [170](#)
  - 初始化 [166](#)
  - COM 数据类型 [162](#)
  - COM 外部过程 [147](#)
  - COM 与 Informatica 类型 [145](#)
  - 代码页访问函数 [172](#)
  - 多线程代码 [144](#)
  - 返回值 [163](#)
  - 分派函数 [168](#)
  - 分区相关函数 [172](#)
  - 概览 [144](#)
  - 跟踪级别函数 [173](#)
  - 跟踪级别（属性） [145](#)
  - IDispatch 接口 [147](#)
  - Informatica 外部过程 [154](#)
  - 接口函数 [168](#)
  - 进程变量支持 [168](#)
  - 开发说明 [162](#)
  - MFC AppWizard [159](#)
  - 模块（属性） [145](#)
  - 内存管理 [164](#)
  - session：会话 [152](#)
  - 是否可分区（属性） [145](#)
  - 使用 BankSoft 示例的 Informatica 外部过程 [154](#)
  - 输出具有确定性（属性） [145](#)
  - 输出是可重复的（属性） [145](#)
  - 说明 [145](#)
  - 属性 [145](#)
  - 属性访问函数 [169](#)
  - 调试 [164](#)
- Visual Basic [153](#)
- Visual C++ [148](#)
- 外部过程函数 [169](#)
- 为 C++ 外部过程构建库 [150](#)
- 为 Informatica 外部过程构建库 [159](#)

- 外部过程转换 (续)
  - 为 Visual Basic 外部过程构建库 [154](#)
  - 未连接 [166](#)
  - 行级过程 [163](#)
  - 需要的文件 [167](#)
  - 异常处理 [164](#)
  - 运行时位置 (属性) [145](#)
  - 在 Designer 中创建 [155](#)
  - 在映射中使用 [152](#)
- 外部联接
  - 另请参阅联接类型[outer join aab] [360](#)
  - 创建 [360](#)
  - 创建为联接替代 [360](#)
  - 创建为提取替代 [360](#)
  - 集成服务支持的类型 [355](#)
- 外键
  - 使用序列生成器转换创建 [331](#)
- “完整代码”窗口
  - Java 编译错误 [210](#)
- 完整数据库连接
  - 传递到 SQL 转换 [373](#)
- 完整外部联接
  - 定义 [246](#)
- Web 链接
  - 添加至表达式 [31](#)
- 未连接的查找转换
  - 输入端口 [272](#)
  - 返回端口 [272](#)
- 未连接的转换
  - 查找转换 [259](#), [271](#)
  - 存储过程转换 [396](#)
  - 外部过程转换 [166](#)
  - 查找 [254](#)
- 未命名缓存
  - 持久性 [280](#)
  - 从数据库重新缓存 [280](#)
  - 共享 [282](#)
- 未排序联接器转换
  - 处理详细信息行 [250](#)
  - 处理主行 [250](#)
- 唯一输出
  - 数据屏蔽转换 [119](#)
- 文件输出类型
  - 非结构化数据转换 [428](#)
- 文件输入类型
  - 非结构化数据转换 [425](#)
- Windows 系统
  - 编译 DLL [159](#)
- 无效的事务控制转换
  - 定义 [414](#)
- 未连接的查找
  - 概览 [259](#), [271](#)
  - 添加查找条件 [272](#)
  - 通过表达式调用 [273](#)
  - 说明 [257](#)
  - 指定返回值 [272](#)

## X

- 线程
  - Custom transformation: 自定义转换 [60](#)
- 向导
  - ATL COM AppWizard [148](#)
  - MFC AppWizard [159](#)
  - Visual Basic 应用程序设置向导 [162](#)

- 相关列
  - 数据屏蔽 [120](#)
- 相关屏蔽
  - 可重复屏蔽 [121](#)
  - 说明 [120](#)
- 相关性
  - 自定义转换中的端口 [57](#)
- 详细初始化跟踪级别
  - 概览 [42](#)
- 详细数据跟踪级别
  - 概览 [42](#)
  - SQL 转换 [381](#)
- 详细外部联接
  - 说明 [246](#)
- 详细信息行
  - 联接器转换中的阻塞 [250](#)
  - 在排序的联接器转换中处理 [251](#)
  - 在未排序联接器转换中处理 [250](#)
- 相异输出行
  - 排序器转换 [346](#)
- 行
  - 标记为更新 [432](#)
  - 删除 [435](#)
- 行策略函数
  - 基于数组 [110](#)
  - 基于行 [104](#)
- 行内出现 SQL 错误时继续 (属性)
  - SQL 转换 [383](#)
- 性能
  - 查找转换 [275](#)
  - 改进筛选器 [176](#)
  - 汇总器转换 [51](#)
  - 静态数据库连接 [375](#)
  - 联接器转换 [253](#)
  - 逻辑数据库连接 [375](#)
  - 使用变量提高 [34](#)
- XML
  - 拆分大型输出 [429](#)
- XML 解析器转换
  - 概览 [436](#)
  - 启用拆分输入 [429](#)
- XML 生成器转换
  - 概览 [437](#)
- XML 转换
  - XML 解析器 [436](#)
  - XML 生成器 [437](#)
  - 源限定符 [436](#)
- 选择相异
  - “源限定符”选项 [363](#)
  - 在会话中替代 [363](#)
- 序列 ID
  - 查找转换 [289](#)
- 序列生成器转换
  - 保持行顺序 [338](#)
  - Blaze 引擎 [343](#)
  - 不可重用 [337](#)
  - 重置 [338](#)
  - 创建 [340](#), [341](#)
  - 创建复合键 [331](#)
  - CURRVAL 端口 [332](#)
  - 当前值 [336](#)
  - 端口 [329](#)
  - 概览 [328](#)
  - 缓存值数 [337](#)
  - 可重用 [337](#)
  - NEXTVAL 端口 [329](#)
  - 起始值 [335](#)
  - 使用 IIF 函数替换缺少的键 [332](#)

序列生成器转换 (续)

- 属性 [334](#), [338](#)
- Spark 引擎 [343](#)
- 循环 [335](#), [336](#)
  - “增量”属性 [335](#), [336](#)
- 值范围 [336](#)
- 非本地环境 [342](#)

序列数据对象

- 属性 [338](#)
- 创建 [338](#)

循环

- 序列生成器转换属性 [335](#)

序列化程序

- Data Transformation [422](#)

循环 (属性)

- 序列生成器转换属性 [334](#)

## Y

验证

- 表达式 [32](#)
- 默认值 [42](#)
- “验证”按钮
- 转换 [42](#)
- 要求单个线程 (属性)
  - 自定义转换 [58](#)
- 要求每个分区一个线程 (属性)
  - HTTP 转换 [178](#)
  - Java 转换 [201](#)

异常

- 从外部过程 [164](#)

已分区管道

- 联接已排序数据 [247](#)

已连接转换

- 表达式 [141](#)
- 存储过程 [396](#)
- 等级 [318](#)
- 更新策略 [431](#)
- 规范器 [302](#)
- 汇总器 [46](#)
- Java [197](#)
- 联合转换 [417](#)
- 联接器 [242](#)
- 路由器 [323](#)
- 筛选器 [174](#)
- SQL [366](#)
- XML 解析器 [436](#)
- XML 生成器 [437](#)
- XML 源限定符 [436](#)
- 序列生成器 [328](#)
- 源限定符 [348](#)
- 自定义 [54](#)
- 查找 [254](#)

映射

- 查找组件 [259](#)
  - 将行标记为更新 [432](#)
  - 配置未连接存储过程转换 [405](#)
  - 配置已连接存储过程转换 [405](#)
  - 使用路由器转换 [327](#)
  - 使用外部过程转换 [152](#)
  - 受存储过程影响 [397](#)
  - 添加 COBOL 源 [306](#)
  - 添加可重用转换 [44](#)
  - 添加转换 [28](#)
  - 修改可重用转换 [44](#)
- 映射变量
- 可重用转换 [43](#)

映射变量 (续)

- 在查找 SQL 替代中 [266](#)
- 在源限定符转换中 [349](#)

映射参数

- 在查找 SQL 替代中 [266](#)
- 在源限定符转换中 [349](#)

映射失败

- Java 转换 [214](#)

已排序端口

- 不使用的原由 [51](#)
- 缓存要求 [48](#)
- 汇总器转换 [51](#)
- 排序顺序 [362](#)
- 源限定符 [362](#)
- 预先排序数据 [51](#)

已排序关系数据

- 在联接器转换中使用 [247](#)

已排序平面文件

- 在联接器转换中使用 [247](#)

已排序数据

- 从已分区管道联接 [247](#)
- 在联接器转换中使用 [247](#)

已排序输入

- 平面文件查找 [255](#)

已排序输入 (属性)

- 汇总器转换 [51](#)
- 联接器转换 [243](#)

用户代码错误

- Java 转换 [211](#)

用户定义的方法

- Java 转换 [205-207](#)

用户定义的函数

- 使用 Java 表达式 [224](#)

用户定义的联接

- 输入 [355](#)

用户定义的组

- 路由器转换 [324](#)

右外部联接

- 创建 [359](#)
- 语法 [359](#)

有效的事务控制转换

- 定义 [414](#)

源

- 从同一源联接数据 [249](#)
- 合并 [417](#)
- 联接 [242](#)
- 联接多个 [242](#)

源筛选器

- 添加到源限定符 [361](#)

元数据扩展

- 自定义转换中的 [63](#)

源限定符转换

- \$\$\$SessStartTime [349](#)
- 查看默认查询 [351](#)
- 创建键关系 [353](#)
- 概览 [348](#)
- 故障排除 [365](#)
- 会话前和会话后 SQL [363](#)
- 进入用户定义的联接 [355](#)
- 进入源筛选器 [361](#)
- 联接 [353](#)
- 联接源数据 [352](#)
- 默认查询 [351](#)
- 默认联接 [352](#)
- 目标加载顺序 [349](#)
- 配置 [364](#)
- 使用汇总器排列顺序 [51](#)
- 数据类型 [349](#)

- 源限定符转换 (续)
  - 属性 [364](#)
  - SQL 替代 [354](#)
  - 替代默认查询 [351](#), [354](#)
  - 外部联接支持 [355](#)
  - XML 源限定符 [436](#)
  - “选择相异”选项 [363](#)
  - 映射参数和变量 [349](#)
  - “已排序端口数”选项 [362](#)
  - 自定义联接 [353](#)
  - 作为查找源 [256](#)
- 源字符串符
  - 数据屏蔽转换 [124](#)
- 语法
  - 常见数据库限制 [361](#)
  - 创建普通联接 [356](#)
  - 创建右外部联接 [359](#)
  - 创建左外部联接 [357](#)
- 运算符
  - 查找条件 [269](#)
- 运行时位置 (属性)
  - 外部过程转换 [145](#)
- 语言 (属性)
  - Java 转换 [201](#)
- 已连接的查找
  - 创建 [274](#)
  - 概览 [258](#)
  - 说明 [257](#)
- 映射程序
  - Data Transformation [422](#)
- 预构建查找缓存
  - 查找属性 [261](#)
- \$Source
  - 查找转换 [261](#)
  - 存储过程转换 [403](#)

## Z

- 在比较中忽略 (属性)
  - 说明 [291](#)
  - “在接收事务中”选项卡
    - Java 转换 [207](#)
  - “在数据末尾”选项卡
    - Java 转换 [207](#)
  - “在输入行”选项卡
    - Java 转换 [206](#)
  - 示例 [238](#)
- 增量
  - 设置序列间隔 [335](#), [336](#)
- 值
  - 通过表达式转换计算 [142](#)
- 置换屏蔽
  - 配置关系字典 [119](#)
  - 屏蔽属性 [118](#)
  - 数据屏蔽规则和准则 [120](#)
  - 说明 [117](#)
- 种子值
  - 数据屏蔽转换 [115](#)
- 转发拒绝的行
  - 配置 [432](#)
  - 选项 [432](#)
- 状态代码
  - 存储过程转换 [397](#)
- 状态跟踪级别
  - 非结构化数据转换 [424](#)
- 转换范围
  - 包含联接器转换的“行”转换范围 [251](#)

- 转换范围 (续)
  - 包含联接器转换的“全部输入”转换范围 [251](#)
  - 包含联接器转换的“事务”转换范围 [251](#)
  - Custom transformation: 自定义转换 [60](#)
  - Java 转换 [203](#)
  - 联接器转换属性 [243](#)
  - 排序器转换 [347](#)
  - 为联接器转换定义 [251](#)
- 转换范围 (属性)
  - Java 转换 [201](#)
- 转换语言
  - 汇总函数 [48](#)
  - 使用 Java 表达式 [224](#)
- 注册
  - 包含存储库的 COM 过程 [151](#)
- 主动转换
  - 等级 [318](#)
  - 概览 [24](#)
  - 更新策略 [431](#)
  - 规范器 [302](#)
  - 汇总器 [46](#)
  - Java [197](#), [198](#)
  - 联接器 [242](#)
  - 路由器 [323](#)
  - 排序器 [344](#)
  - 筛选器 [174](#)
  - 事务控制 [412](#)
  - XML 解析器 [436](#)
  - XML 生成器 [437](#)
  - XML 源限定符 [436](#)
  - 源限定符 [348](#), [366](#)
  - 自定义 [54](#)
  - 联合 [417](#)
- 主键
  - 使用序列生成器转换创建 [331](#)
- 主空值排序 (属性)
  - 联接器转换 [243](#)
- 主排序顺序 (属性)
  - 联接器转换 [243](#)
- 注释
  - 添加至表达式 [31](#)
- 主外部联接
  - 保留事务边界 [251](#)
  - 说明 [245](#)
- 主行
  - 缓存 [251](#)
  - 在排序的联接器转换中处理 [251](#)
  - 在未排序联接器转换中处理 [250](#)
- 字典
  - 可重复表达式屏蔽 [127](#)
  - 置换数据屏蔽 [118](#)
- 字典信息
  - 数据屏蔽转换 [119](#)
- 自定义函数
  - 使用 Java 表达式 [224](#)
- 自定义转换
  - 编译过程 [71](#)
  - 创建 [55](#)
  - 创建过程 [63](#)
  - 代码页 [55](#)
  - 定义端口关系 [57](#)
  - 分发 [55](#)
  - 概览 [54](#)
  - 构建模块 [71](#)
  - 函数 [74](#)
  - 生成代码文件 [55](#)
  - 生成事务属性 [61](#)
  - 是否可分区 (属性) [58](#)

- 自定义转换 (续)
  - 输入可能编块属性 [61](#)
  - 属性 [58](#)
  - 属性 ID [88](#)
  - 特定于线程的代码 [58](#)
  - 要求单个线程属性 [58](#)
  - 组件 [56](#)
  - 阻止数据 [61](#)
- 自定义转换过程
  - 创建 [63](#)
  - 生成代码文件 [65](#)
  - 使用行 [78](#)
  - 特定于线程 [58](#)
  - 示例 [66](#)
- 自定义转换函数
  - API [84](#)
  - 重新绑定数据类型 [94](#)
  - 初始化 [79](#)
  - 错误 [99](#)
  - 导航 [85](#)
  - 递增错误计数 [100](#)
  - 更改默认行策略 [105](#)
  - 更改字符串模式 [103](#)
  - 会话日志 [99](#)
  - 基于数组 [106](#)
  - 取消初始化 [82](#)
  - 生成 [79](#)
  - 设置传递端口 [98](#)
  - 设置数据代码页 [103](#)
  - 设置数据访问模式 [85](#)
  - 设置输入错误行 [111](#)
  - 使用句柄 [74, 85](#)
  - 输出通知 [98](#)
  - 数据边界输出 [99](#)
  - 数据处理 (基于数组) [108](#)
  - 数据处理 (基于行) [96](#)
  - 属性 [87](#)
  - 通知 [81](#)
  - 行策略 (基于数组) [110](#)
  - 行策略 (基于行) [104](#)
  - 行是否有效 [108](#)
  - 行数 [107](#)
  - 已终止 [101](#)
  - 指针 [102](#)
  - 最大行数 [106](#)
  - 阻止逻辑 [101](#)
- 自动提交
  - 配置 SQL 转换 [376](#)
  - 说明 [383](#)
- 字符串
  - 等级划分 [319](#)
- 字符串值
  - 键数据屏蔽 [116](#)
  - 自定义数据屏蔽 [122](#)
- 字符串置换
  - SQL 转换查询 [370](#)
- 子秒
  - 在 Java 转换中处理 [209](#)
- 组
  - Custom transformation: 自定义转换 [56](#)
  - HTTP 转换 [180](#)
  - Java 转换 [200](#)
  - 联合转换 [418](#)
  - 路由器转换 [324](#)
- 组 (续)
  - 添加 [326](#)
  - 用户定义 [324](#)
  - 自定义转换规则 [57](#)
  - 最大输出行计数 (属性)
    - SQL 转换 [383](#)
  - 左外部联接
    - 创建 [357](#)
    - 语法 [357](#)
  - 组筛选条件
    - 路由器转换 [325](#)
  - 阻止
    - 联接器转换中的详细信息行 [250](#)
  - 阻止数据
    - 联接器转换 [250](#)
    - 自定义转换 [61](#)
    - 自定义转换函数 [101](#)
  - 增量 (属性)
    - 序列生成器转换属性 [334](#)
  - 执行顺序 (属性)
    - 存储过程转换 [403](#)
  - 转换
    - 表达式 [141](#)
    - 创建 [28](#)
    - 处理错误 [40](#)
    - 存储过程 [396](#)
    - 等级 [318](#)
    - 多组 [29](#)
    - 更新策略 [431](#)
    - 跟踪级别 [42](#)
    - 规范器 [302](#)
    - 汇总器 [46](#)
    - Java [197](#)
    - 可重用转换 [43](#)
    - 联合 [417](#)
    - 连接 [25](#)
    - 联接器 [242](#)
    - 路由器 [323](#)
    - 筛选器 [174](#)
    - 设为可重用 [44](#)
    - SQL [366](#)
    - 添加到映射 [28](#)
    - 提升为可重用 [44](#)
    - 未连接 [25](#)
    - XML 解析器 [436](#)
    - XML 生成器 [437](#)
    - XML 源限定符 [436](#)
    - 序列生成器 [328](#)
    - 源限定符 [348](#)
    - 允许使用表达式的类型 [30](#)
    - 主动和被动 [24](#)
    - 自定义 [54](#)
    - 本地和非本地 [25](#)
    - 查找 [254](#)
    - 定义 [24](#)
    - 概览 [24](#)
    - 类型 [24](#)
    - 说明 [25](#)
  - 转换器组件
    - Data Transformation [422](#)
  - 子秒精度
    - 查找转换 [261](#)
    - 存储过程转换 [403](#)