

# Informatica PowerCenter 权威指南

杜绍森 著



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

# Informatica PowerCenter 权威指南

在大数据时代，掌控数据首先需要掌握数据的处理能力。俗话说：“工欲善其事，必先利其器。” Informatica PowerCenter作为业界广泛使用的数据处理工具之一，被全球多数大型机构、组织认可并采用。

本书全面地介绍了Informatica PowerCenter的主要功能及高级特性。

本书分为3个部分：第一部分为基础篇，包括第1~4章，系统介绍了PowerCenter的基础组件和常用功能，并在其中穿插了大量实践案例；第二部分为高级篇，包括第5~8章，系统介绍了PowerCenter并行、集群、性能调优和字符集管理等高级内容；第三部分为扩展篇，包括第9章，简要介绍了CDC的相关知识，PowerCenter与SAP、MPP、Hadoop集成，以及非结构化和半结构化数据处理能力。

本书适合PowerCenter的入门者及有一定PowerCenter使用经验的用户参考，也可作为各数据仓库、大数据专业培训机构的培训教材。

---

不同于普通的产品手册，本书作者以其十几年的理论研究和教育培训，以及主导或参与诸多中外企业“数据集成项目”实施的经验，将PowerCenter的很多功能细节描述得淋漓尽致。本书对于有意进行ETL教学的教育培训机构，不失为一本经典的教材；而对于有意培养自己成为ETL应用高手的IT人士，则是一本有益而又有趣的读物。

Informatica大中国区总经理 王晨杰



《Informatica PowerCenter权威指南》

官方QQ群：212017946



博文视点Broadview



@博文视点Broadview

上架建议：大数据、数据处理

ISBN 978-7-121-27045-1



9 787121 270451 >



责任编辑：徐津平  
封面设计：侯士卿

定价：69.00元

# Informatica PowerCenter 权威指南

杜绍森 著 /

电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING

## 内 容 简 介

在大数据时代，掌控数据首先需要掌握数据的处理能力。俗话说：“工欲善其事，必先利其器。”**Informatica PowerCenter** 作为业界广泛使用的数据处理工具之一，被全球多数大型机构、组织认可并采用。

本书全面地介绍了 **Informatica PowerCenter** 的主要功能及高级特性。

本书分为 3 个部分：第一部分为基础篇，包括第 1~4 章，系统介绍了 PowerCenter 的基础组件和常用功能，并在其中穿插了大量实践案例；第二部分为高级篇，包括第 5~8 章，系统介绍了 PowerCenter 并行、集群、性能调优和字符集管理等高级内容；第三部分为扩展篇，包括第 9 章，简要介绍了 CDC 的相关知识，PowerCenter 与 SAP、MPP、Hadoop 集成，以及非结构化和半结构化数据处理能力。

本书适合 PowerCenter 的入门者及有一定 PowerCenter 使用经验的用户参考，也可作为各数据仓库、大数据专业培训机构的培训教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

### 图书在版编目（CIP）数据

**Informatica PowerCenter** 权威指南 / 杜绍森著. —北京：电子工业出版社，2015.9

ISBN 978-7-121-27045-1

I. ①I… II. ①杜… III. ①企业管理—数据管理 IV. ①F270.7

中国版本图书馆 CIP 数据核字（2015）第 203208 号

责任编辑：徐津平

特约编辑：赵树刚

印 刷：北京京师印务有限公司

装 订：北京京师印务有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：22.75 字数：475 千字

版 次：2015 年 9 月第 1 版

印 次：2015 年 9 月第 1 次印刷

印 数：3000 册 定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：(010) 88258888。

# 推 荐 序

犹豫了很久，以我现今的职位给作者写序，是否有些自吹自擂？但读完书稿，我决定了：一本好书，介绍一个好产品，既然与我受用，何不推荐给更多的人呢？

“IT”是信息（Information）和技术（Technology）的缩写，它的发展不过三十多年的时间。在 IT 发展前期的大部分时间里，其所有进步大部分体现在“T”上，例如计算性能、存储容量、网络拓展及打印效果等。直到近些年“大数据时代”的出现，人们才开始了对于信息数据，也就是“I”的关注。我相信，这个变化是 IT 发展的必然，是一个破茧成蝶的过程，并且这个关注也一定会延续很多年。

同样，正是因为“大数据”日益深入人心，企业的 IT 规划和发展越来越与“大数据”相关联，PowerCenter 才得以从一个 IT 人员得心应手的工具，蜕变为大数据应用的一个重要环节。记得在 2014 年国务院工业和信息化部颁布的大数据白皮书中，就明确地将“数据准备”定义为大数据发展的第一个环节。由此，作为在数据集成领域里历年排名第一的 PowerCenter，也就承担起了“帮助企业实现大数据应用的第一步”的重要使命。

本书前 6 章中规中矩，如同一本深入浅出的教科书，将具备一些基本 IT 知识的人士引进数据迁移的奇妙世界，加上作者风趣的调侃，学来丝毫不觉得枯燥单调。第 7 章开始是实战描述，实际上是一系列的应用经验分享，这些宝贵的经验之谈，可以让初学者在未来的实践中少走弯路，还可以将本书作为可以随时受教的参考书。更值得一提的是，不同于普通的产品手册，本书作者以其十几年的理论研究和教育培训，以及主导或参与诸多中外企业“数据集成项目”实施的经验，将 PowerCenter 的很多功能细节描述得淋漓尽致。本书对于有意进行 ETL 教学的教育培训机构，不失为一本经典的教材；而对于有意培养自己成为 ETL 应用高手的 IT 人士，则是一本有益而又有趣的读物。

曾经有不少朋友问我：当成了 ETL 的行家里手以后，下一个职业目标会有哪些发展方

向？所以，我想借此序的一角，分享一些我的认知，供大家参考。

**第一，云数据集成和管理。**根据 IDC 的预测，2017 年全球 SaaS 和云软件模式将占软件开支的 1/6。越来越多的云应用系统承诺并交付更简单、更快捷和更智能的业务营运方法，所以，掌握云数据集成，会让你在不可阻挡的云服务趋势下游刃有余。

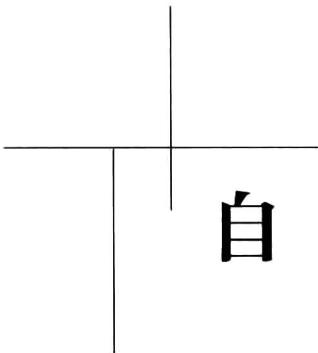
**第二，下一代数据洞察。**不同于第一代商业智能（BI）对展示形态和分析过去的重视，大数据时代的数据洞察，更加关注数据的质量而不是数据的展现形式，更重视预测未来的行为模式而非过去的行为分析。所以，要想成为大数据分析专家，你必须懂得数据质量管理和前瞻性的分析。当然，保障分析结果正确的前提是确保数据的统一性、完整性，并找到数据的关联性。

**第三，数据治理。**大数据时代，越来越多的企业将数据纳入其固定资产；在金融和医疗行业，数据相关的合规性成为政府监管的重要指标；为了应对客户需求和市场业务模式的变化，许多企业开始考虑应用整合和迁移……这些巨大的变化，不断催生出数据治理的高手，他们必须在行业规范、企业应用系统、数据的关联性和安全性方面具备独特的技能。因此，了解行业特性、行业应用，使之与数据集成相结合，便成为你进行数据治理的更高境界。

近年来，关于大数据的定义一直在调整，而大数据应用的目标却始终没变，那就是：发现数据价值，帮助企业降低成本并实现业务创新。在过去短短的两三年里，中国作为自然的大数据国家，已经在大数据的理论研究和实际应用方面取得了巨大的进展。大数据的应用会推动各行各业诞生越来越多的数据科学家，那是行业知识和数据治理兼备的卓越人才。IT 的发展已经实现了由“计算机科学”向“数据科学”的转换，近年来，“数据科学”又开始向行业应用进行大规模迁移。所以，数据科学家既是数据价值的挖掘者，更是行业产品和流程的创新者，他们的价值不是向企业的高管提供分析报表，他们本身就是企业的高管，他们在用数据作为依据，实现企业面向客户、市场、产品和流程方面的创新。

千里之行，始于足下。与各位读者共勉。

Informatica 大中国区总经理 王晨杰



# 自序

初识 Informatica，大概是在十四五年前的一个偶然的机会，公司接到一个叫作决策支持系统（DSS）的项目。尽管当时作为工程师和客户一起整理了项目的需求，完成了需求的确认和签字，但我现在几乎无法记起任何关于需求的内容，对项目实施过程的某些环节却仍然记忆犹新。项目开始时，公司安排了两位工程师参与 ETL（Extract Transformation Load），一位是我，另一位与我现在仍是同事，我们当时使用了一个叫作 PowerMart 的工具，版本是 5.1。这就是我和 Informatica 的第一次亲密接触。当时 Informatica 的总代理也是曾经大名鼎鼎的 Sybase，据说我们的这个项目是 Informatica 进入中国后的第二个项目。

从此，我开始了自己漫长 Informatica 之路。当时我所就职的公司敏锐地察觉到数据仓库/商业智能是未来的趋势之一，开始着手准备发展数据仓库方面的业务。当年有个著名的第三方调研机构，叫 IDC。我所就职的公司通过查阅 IDC 报告，发现 Informatica 是当时 ETL 市场份额最高的公司，于是果断决定采用这个工具作为自己的数据仓库的开发平台。当年公司的果断、决心，让我至今想起，仍非常钦佩。在 IT 人才严重短缺的年代，虽然年纪很轻、经验不足，但我还是作为经营分析项目的项目经理、技术经理等开始了自己的数据管理生涯。

此后一段时间，我在不经意间进入悲催的计费岁月，加班、加班、通宵、通宵……每个项目都以几年来计算，历经两个完整的移动计费项目，在此期间认识了很多好朋友和师长。这是我与 Informatica 断绝联系的几年，也是在技术方面拓展能力的几年。

我与 Informatica 的缘分并没有结束。有一天，原来的同事告诉我 Informatica 在招售前工程师，我就毅然决定去应聘，满足自己转向咨询领域的一点梦想，后来发现售前和咨询还是有所区别的，这是后话。因此，8 年前我加入了 Informatica 中国，开始成为一名专职的售前工程师。当年的 Informatica 只有这一个产品，人不是很多。我仍清晰地记着当时的

版本为 PowerCenter 8.1.1。现在的 Informatica 已经与早期差别非常大了，但是很多人还是习惯把 Informatica 的数据集成产品 PowerCenter 叫作 Informatica。

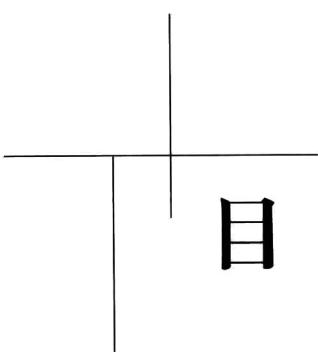
为什么要写一本关于 PowerCenter 的书呢？其实我内心里一直有这样的冲动。PowerCenter 是一个非常好的产品，在国内也有近千家用户，有大量的开发者和管理者，随着大数据的推广，还有大量的后继者会陆续开始 PowerCenter 之旅。一本中文材料会帮助所有的用户更加快速、更加全面地了解 PowerCenter，充分利用自己在 PowerCenter 方面的投资。这个冲动持续了很久，包括期间陆续说服数位同事参与，但是大家都有繁忙的本职工作，一次一次被耽搁。直到 2015 年春节前，内心的冲动促使我开始动笔了。前两天坐在我对面的北区销售总监说，我写这本书像写回忆录，想想确实有道理，我的自序也是回忆录的样子，希望大家能够谅解。

如何写好这本书？这也是我非常纠结的一个问题。什么样的深度？适合什么样的人群？是否有读者愿意来读？如何帮助读者了解 PowerCenter？和同事们讨论过很多次，似乎还是没有下定决心。当我下笔的时候，尤其是写到 50 多页的时候，我觉得我已经坚定了这本书的方向：不求全面，但求让读者快速地掌握 PowerCenter；不求精深，但求将最常用的功能展现给读者；不求华丽的词藻，但求读者能读懂。

PowerCenter 是什么？它是 ETL 工具。什么是 ETL？大数据及数据仓库 70% 左右的工作都在做 ETL，在 Gartner 报告中它被划为 Data Integration 产品。Informatica 也曾定义自己是一家 Data Integration 公司。我是这样解释 ETL 工具的：它是 Data Integration 产品在数据仓库、大数据项目中的一个应用场景，它同时还有其他的应用场景，比如数据交换、数据安全，这些也是 PowerCenter 在后期的扩展。

希望本书能够成为分享我这些年掌握的 PowerCenter 相关知识的一个载体，成为初学者的入门教材，成为有经验者的开发人员的一本参考书。

杜绍森  
2015 年 8 月



# 目 录

---

## 第 1 章 PowerCenter Hello World 世界

1.1	Informatica Hello World.....	1
1.2	PowerCenter 架构和客户端简介 .....	3
1.2.1	PowerCenter 架构 .....	3
1.2.2	PowerCenter 客户端 .....	5
1.3	PowerCenter Hello World .....	7

---

## 第 2 章 PowerCenter 基础组件

2.1	Source.....	27
2.1.1	数据库源 .....	28
2.1.2	文本文件源 .....	30
2.2	Target.....	33
2.2.1	数据库目标 .....	33
2.2.2	文本文件目标 .....	34
2.3	Expression 表达式 .....	35
	Expression 中的变量端口 ( Variable Port ) .....	40

2.4 Filter .....	41
2.5 Source Qualifier .....	43
2.5.1 Source Qualifier 的作用 .....	43
2.5.2 数据库数据源的 Source Qualifier .....	44
2.5.3 Source Qualifier 自定义 SQL .....	47
2.5.4 Source Qualifier 复杂关联 .....	48
2.6 Sorter .....	49
2.7 Joiner .....	51
2.7.1 关联类型 .....	52
2.7.2 Sorted Joiner .....	54
2.7.3 Joiner 的独特作用 .....	55
2.7.4 自关联 (Self-Join) .....	56
2.8 Lookup .....	57
2.8.1 Lookup Caching enabled .....	59
2.8.2 非连接的 Lookup .....	61
2.8.3 Lookup SQL Override .....	63
2.8.4 共享 Lookup Cache .....	65
2.8.5 Dynamic Lookup .....	65
2.8.6 Lookup、Source Qualifier 和 Joiner 的对比 .....	69
2.9 Stored Procedure .....	70
2.9.1 Connected Stored Procedure .....	70
2.9.2 Unconnected Stored Procedure .....	72
2.9.3 Pre- or Post-Session Stored Procedure .....	74
2.10 Union .....	76
2.11 Transaction Control .....	78
2.11.1 Transaction Control 有效性问题 .....	79
2.11.2 Transaction Control 组件 .....	80

2.12 Sequence.....	80
2.12.1 Sequence 的常规用法.....	80
2.12.2 共享 Sequence.....	82
2.12.3 可重用的 Sequence.....	83
2.13 Aggregator.....	84
2.13.1 条件聚合 .....	85
2.13.2 使用 Aggregator 进行行列转换 .....	86
2.14 Rank .....	88
2.15 Update strategy.....	90
2.15.1 Treat source rows as 属性的使用 .....	91
2.15.2 Update strategy 使用 .....	93
2.15.3 如何实现 Update else Insert .....	94
2.15.4 Update Stagety 案例：缓慢变化维 .....	98
2.16 SQL Transformation.....	104
2.16.1 Script Mode.....	104
2.16.2 Static Query Mode.....	106
2.16.3 Dynamic Query Mode .....	108
2.17 Java Transformation .....	109
2.17.1 Java Transformation 简介 .....	109
2.17.2 Passive Java Transformation.....	114
2.17.3 Active Java Transformation.....	121
2.17.4 常见错误说明 .....	123
2.18 Normalizer.....	124
2.19 Router.....	126
2.20 Custom Transformation.....	128
2.21 HTTP Transformation .....	129
2.22 XML 组件组 .....	132

2.23 Transformation 中的一些概念 .....	135
2.23.1 Connect 与 Unconnect.....	135
2.23.2 Active 与 Passive .....	136

## 第 3 章 Workflow 执行、监控

138

3.1 Session.....	139
3.1.1 Reusable Session .....	139
3.1.2 非 Reusable Session .....	141
3.2 最简单、最常用的 Workflow.....	143
3.2.1 并行执行.....	143
3.2.2 串行执行.....	144
3.2.3 调度.....	146
3.3 Worklet .....	147
3.4 Command .....	148
3.5 Control.....	150
3.6 发送 E-mail.....	151
3.6.1 配置发送 E-mail .....	151
3.6.2 在 Workflow 中使用 E-mail .....	151
3.7 Event Tasks.....	155
3.7.1 用户自定义事件使用 .....	156
3.7.2 预定义事件使用 .....	158
3.8 Timer .....	159
3.9 Decision.....	159
3.10 Assignment.....	160

**第 4 章 常用功能汇集**

163

4.1	Debugger .....	163
4.2	Mapplet/Reusable Transformation .....	165
4.2.1	Reusable Transformation.....	165
4.2.2	Mapplet.....	167
4.3	使用 Shortcut .....	169
4.3.1	Local Shortcut .....	170
4.3.2	Global Shortcut.....	171
4.4	Session 相关属性.....	173
4.4.1	Properties Tab 相关属性 .....	173
4.4.2	Config Object Tab 相关属性.....	174
4.5	参数和变量.....	176
4.5.1	Mapping 参数.....	176
4.5.2	Mapping 变量.....	180
4.5.3	系统/Session 参数与变量 .....	184
4.5.4	Workflow/Worklet 变量 .....	189
4.5.5	Local 变量 ( Local Variables ) .....	191

**第 5 章 PowerCenter 高级应用**

193

5.1	任务分区 (Partition) .....	193
5.1.1	Database Partitioning.....	196
5.1.2	Hash Partitioning .....	201
5.1.3	Key Range Partitioning .....	204
5.1.4	Pass Through Partitioning.....	205
5.1.5	Round-Robin Partitioning.....	211
5.2	内存管理 .....	214

5.2.1 DTM 内存 .....	215
5.2.2 Transformation Cache .....	216
5.3 网格计算 .....	219
5.3.1 Grid 架构 .....	219
5.3.2 Grid 负载均衡 .....	221
5.3.3 Grid 与任务分区（Partition） .....	224
5.4 高可用性（HA） .....	227
5.4.1 PowerCenter 自带的 HA 方案 .....	228
5.4.2 依托第三方厂商的 HA 方案 .....	229
5.4.3 两种 HA 方案对比 .....	230
5.5 Web Service 应用 .....	230
5.5.1 Web Service Hub .....	231
5.5.2 Web Service 调度/监控接口 .....	232
5.5.3 Web Service Provider .....	234
5.5.4 Web Service Consumer .....	246
5.6 Pushdown Optimization .....	251
5.6.1 Pushdown 优化是什么 .....	252
5.6.2 Pushdown 优化类型 .....	252
5.7 版本控制及部署 .....	256
5.7.1 Check In/Check Out .....	256
5.7.2 Team-Based 开发的一些有用功能 .....	258
5.7.3 Label 与 Deployment Group .....	260
5.7.4 复制对象从开发 Repository 到生产 Repository .....	264

## 第 6 章 PowerCenter 实战汇总

266

6.1 PowerCenter 字符集 .....	266
6.1.1 Oracle 数据库 .....	267

6.1.2 DB2 字符集 .....	268
6.1.3 AS/400 字符集 .....	268
6.1.4 ODBC 字符集 .....	269
6.1.5 文本文件字符集 .....	270
6.1.6 Repository Service 字符集 .....	271
6.1.7 Integration Service 字符集 .....	272
6.1.8 Data Movement Mode .....	273
6.2 UNIX ODBC 配置 .....	274
6.2.1 ODBC 常规配置 .....	274
6.2.2 MySQL 社区版 ODBC 配置 .....	276
6.3 使用 Mapping 动态分发文件 .....	277
6.4 超越 EDW，商品自动价格跟踪 .....	279
6.5 pmcmd 命令 .....	283
6.6 pmrep 命令 .....	284
6.7 infasetup 命令 .....	284
6.8 Mapping Architect for Visio .....	286
6.9 MX View 语句 .....	293
6.10 PowerCenter 与其他工具集成 .....	294

## 第 7 章 性能调优

297

7.1 性能调优过程 .....	298
7.2 发现瓶颈 .....	299
7.2.1 定位目标写瓶颈及调优 .....	301
7.2.2 定位源读瓶颈及调优 .....	302
7.2.3 定位 Mapping/Session 瓶颈 .....	303
7.2.4 定位系统瓶颈 .....	305

7.3 Mapping 调优 .....	305
7.3.1 Transformation 优化 .....	305
7.3.2 列级别的优化 .....	310
7.3.3 其他方面的优化 .....	312
7.4 Session 调优.....	313
7.4.1 内存调优.....	313
7.4.2 PowerCenter 高级特性支持高性能.....	313
7.4.3 其他手段.....	314
7.5 SQL Override 调优 .....	316

## 第 8 章 PowerCenter Troubleshooting

317

8.1 安装、启动过程的错误 .....	317
8.2 开发过程的错误 .....	319
8.3 Session 运行错误.....	320
8.4 源读或者目标写的错误 .....	321

## 第 9 章 PowerCenter 扩展能力

322

9.1 PowerExchange CDC（变化数据捕捉） .....	322
9.1.1 PowerExchange CDC 的 3 种模式 .....	323
9.1.2 开放数据库 CDC 基本原理 .....	325
9.1.3 CDC 常见的一些讨论 .....	326
9.1.4 CDC Real-Time for Oracle 安装配置（实例） .....	327
9.1.5 CDC 定义注册组和添加捕获注册（实例续） .....	331
9.1.6 CDC Mapping 开发及运行（实例） .....	334

9.2 PowerCenter 与 SAP .....	336
9.2.1 R/3、mySAP、ECC .....	337
9.2.2 PowerCenter 与 BW .....	338
9.3 PowerCenter 与 MPP 数据库 .....	339
9.4 PowerCenter 与 Hadoop .....	340
9.4.1 接口能力 .....	341
9.4.2 PowerCenter on Hadoop .....	344
9.5 元数据管理与业务术语管理 .....	345
9.5.1 元数据的血缘分析 .....	346
9.5.2 元数据影响分析 .....	346
9.5.3 业务数据管理 .....	347
9.6 B2B Data Transformation .....	347

# 第1章

# PowerCenter Hello World 世界

---

Hello World 程序是学习 IT 的入门术语，是初步了解一种技术的有效途径，因此本书也从 Hello World 开始。除了 PowerCenter Hello World，本书还提供了 Informatica Hello World 的个人版。

## 1.1 Informatica Hello World

---

Informatica 曾经是一个 ETL 工具的供应商，也是最好的 ETL 产品的供应商。在开始使用 ETL 工具，并加入到 Informatica 之后的很长一段时间内，我都认为 Informatica 是一家这样的公司。那时的 Informatica 只有一个产品，叫作 PowerCenter。那时，它的 Logo 下有一句话 “The Data Integration Company”。

此后，“幸福的日子”结束了，Informatica 陆续有了很多新产品，包括除 PowerCenter 外的 Data Quality、Data Archive、Master Data Management、Data Masking、Ultra Message、Data Integration Hub 等，这时，其 Logo 下的 “The Data Integration Company” 显著的 IT 蓝依然没有变化。这时的我也在思考什么是 Data Integration，并有了自己对 Data Integration 的诠释。我将 Infomratica 所谓的 Data Integration 诠释为三类数据集成，即下游集成、中游集成和上游集成。

下游集成指的是数据仓库，这是典型的数据集成项目，它有一个显著特点：从数据流

的角度看，数据仓库的主体功能是所有应用系统的下游，所有的数据都会流向数据仓库。当然有人会提出，数据仓库也会作为某些应用的数据提供者，但这不影响下游集成的主要特点，即数据是在数据流的下游进行集成的。

中游集成指数据交换平台，如 Data Integration Hub 等，它的特点是：任何平台和它的关系都是对等的，它是这个数据 Hub 的中心点，用来支持所有系统之间数据层的数据交换，用于解决数据集成毛团的问题。

上游集成指的是主数据平台，而且是交易型主数据平台。它用于管理企业核心数据的黄金记录，作为企业核心记录的黄金数据的标准平台。从数据流的角度看，它成了企业的新中心，我把主数据管理定义为上游集成。但我也注意到很多主数据的成功案例，似乎交易型不一定占据绝对主流，大量的主数据案例都是分析型的。当然，这也是后话，本书毕竟不是讨论主数据，对于选择交易型主数据还是分析型主数据，适合自己的才是最好的。

在这个时期，Informatica 有频繁的收购活动，所以才有了这么多的产品，也实现了高速增长，我也在仔细观察这家公司的发展。这时候我观察到公司的一个脉络，尤其是在接触 DAMA 中国之后。我发现 Informatica 那个时期的收购似乎是围绕着数据治理的 10 个功能要素在运行，似乎有从产品角度填满数据治理所需基础软件的想法，它的产品家族开始包括数据仓库 ETL、数据质量、元数据管理、主数据管理、数据隐私保护、部分半结构化/非结构化解决方案等。Informatica 在这个期间实现了连续 32 个季度近 20% 的增长。

不久之前，Informatica 的 Logo 从传统的 IT 蓝变成了彩色的 Informatica，原来 Logo 下的“*The Data Integration Company*”也变成了“*Put Potential to Work*”，演绎着从传统的 IT 公司向时尚公司的转变。不过一家公司骨子里的东西要想转变需要很长的时间。Informatica 属于 IT 公司里的技术型公司，从我的老板们大家就可以看出来。我的第一任老板是个美国人，当年是位 Main Frame 高手，现在是云部门的技术 VP；第二任老板，在 2008 年 5 月 12 日剧烈地震那天还在长安街的某栋楼的顶层培训教室给我们培训数据质量管理，此后他也曾是亚太区的 VP。

此时此刻，我个人认为 Informatica 的定位是数据管理领域的专家，它向客户提供三类核心的解决方案，支撑数据管理的发展。这三类方案包括：发掘数据价值、保障数据安全、数据云。

这些只是我个人的感受，并不代表公司的战略与方向。既然是一本书，总要有些与众不同，对 Informatica 公司的理解完全是个人的感觉，也可能会随着时间的改变而继续改变。

## 1.2 PowerCenter 架构和客户端简介

在很久之前，PowerCenter 叫作 PowerMart。它的功能很简单，只是简单的 E (Extract) -T (Transformation) -L (Load)，即数据抽取、转换和加载。其安装、使用过程没有任何服务器的感觉，用起来也极其简单。不过当年很多时候 Informatica 会强调 PowerCenter 是元数据驱动的架构。这包括了两层含义：一是运行一个 ETL 过程不需要编译；二是 Mapping、Session 等对象完全存放在知识库中，作为元数据管理，满足行业的元数据标准。

### 1.2.1 PowerCenter 架构

经过二十几年的发展，PowerCenter 的架构也在不断地改变，甚至可以说还在改变或者演进过程中。我试图从自己学习的时代开始介绍 PowerCenter 架构的演进过程，以帮助读者更清晰地了解每个 PowerCenter 架构组件的基本功能。

PowerMart 的架构如图 1-1 所示。

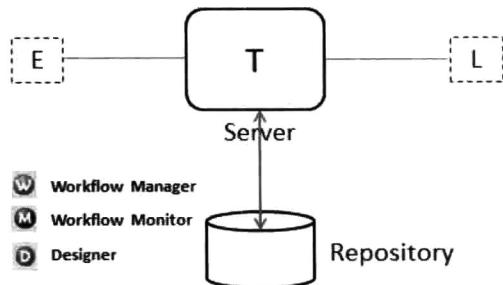


图 1-1

在 PowerMart 的架构中，有一个 Server (即 T)，有一个 Repository，还有 3 个主要客户端。Repository 存放在数据库中，有时会被认为是一个数据库的用户，Server 和所有的客户端直接访问 Repository，数据库里存放了元数据，或者叫作程序。3 个客户端分别叫作 Designer，用于开发程序；Workflow Manager，用于开发 Session；Workflow Monitor，用于监控。当年没有 Workflow 和调度的概念。

后来，PowerCenter 世界发生了重要的变化，增加了一个 Server，叫作 Repository Server。

这个服务被增加在 Server 和 Repository 之间。Repository Server 是一个进程，Repository 是数据库中的一些表及表中管理的数据，如 1-2 所示。

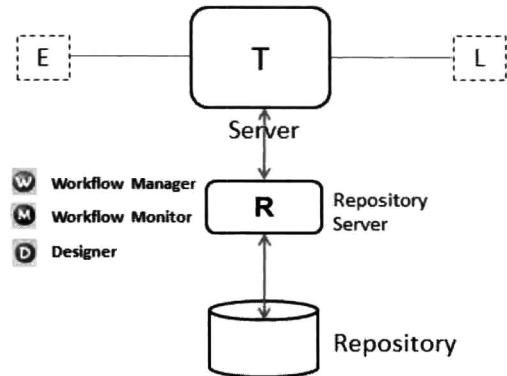


图 1-2

这时的 Server, 包括 Session 和所有的客户端都要通过 Repository Server 访问 Repository 数据库。我的理解是 Repository Server 的功能就是管理, 尤其是提升管理能力, 当 Server 和所有客户端直接访问 Repository 的时候, 各个 Session 和客户端之间有可能争用, 也有可能死锁。类似人多了之后需要有一个 Manager 来管理, 客户端多了也需要有一个 Server 来管理, 这个 Repository Server 为 PowerCenter 后来的扩展打下了坚实的基础。

数据管理时代很快就到了, 数据管理已经形成了一个小圈子, 大家在讨论什么是数据、什么是信息、什么是知识, ETL 工具的作用也越来越大。这时候面临的第一个问题就是: 一台服务器无法满足现在数据处理的需要, 需要扩展为多台服务器。第二个问题是: Repository Server 既然是一个进程, 就拥有程序的全部特性, 它必然运行在某台机器上, 机器就有可能宕机。既然所有的服务器端和客户端都需要通过它来访问 Repository, 它最好有 HA(高可用性)的机制, 保证其不会出现单点故障。其实 Server 也有类似的需求, PowerCenter Server 也是支持 HA 的。这时候它的架构就演进到了类似图 1-3 所示的状态。

图 1-3 中各项的含义如下。

- Domain: 是 PowerCenter 中所有服务器对象及服务对象的集合, 最常见的对象包括节点、Integration Service、Repository Service、Web Service Hub、Grid 等。它可以包括一个或者多个如上的对象。

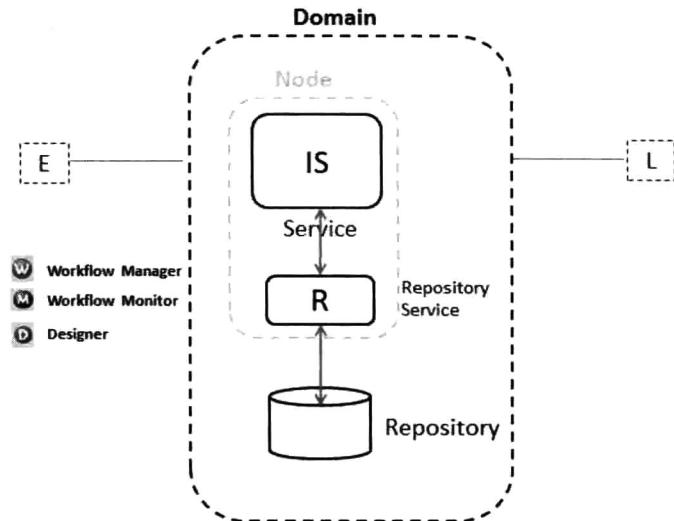


图 1-3

- ◎ **Node:** 它是现实中物理服务器在 PowerCenter 架构中的逻辑名，事实上它是一系列的服务，用于表示物理的服务器。
- ◎ **IS:** 即 Integration Service，所有的 ETL 工作都由其来指挥。当一个 Session 启动时，会启动一个或者多个 pmdtm 进程，这些进程或者其中的线程都是由 Integration Service 负责指挥的。
- ◎ **R:** 即 Repository Service。它是 Repository 的管理人；当需要访问 Repository 时，需要通过 Repository Service 进行。
- ◎ **Repository:** 是数据库某个用户下一系列的表及其中存放的数据，是 PowerCenter 在配置过程中通过 Create Repository Content 创建的。这些表是 PowerCenter 自动创建且由 PowerCenter 来管理的，与数据源或者目标表无关。在 PowerCenter 的联机文档中，可以通过 Repository Guide 中的 Using Metadata Exchange (MX) Views 了解部分表的定义及其存放的内容。

## 1.2.2 PowerCenter 客户端

PowerCenter 由 4 个最常用的客户端组成，分别是 PowerCenter Designer、PowerCenter Workflow Manager、Powercenter Workflow Monitor 和 PowerCenter Repository Manager。各客

户端的作用如下。

(1) **PowerCenter Designer:** 用于导入 ETL 元数据，开发 ETL 程序。在 PowerCenter 中 ETL 程序被叫作 Mapping，即源到目标的映射。为了更方便地实现这些 Mapping，PowerCenter 提供了几十个在 ETL 过程中常用的组件 (Transformation)，并提供了数百个常用的函数，以帮助开发人员迅速地开发出自己需要的 ETL 程序。

(2) **PowerCenter Workflow Manager:** 在 PowerCenter 中，开发的 Mapping 是无法直接运行的。在真正开始运行前，需要给 Mapping 配置一系列的参数，如具体的数据源、数据目标、使用的字符集及调优等，这些工作都是在 Workflow Manager 中完成的。此外，既然叫作 Workflow (工作流)，它还提供了基本的调度和排程的能力，如定时调度、持续运行等，也包括并行执行、串行执行及更多的高级功能。

(3) **PowerCenter Workflow Monitor:** 用于监控运行时的 Workflow 和 Session，通过 Workflow Monitor 可以监控到 ETL 运行是否正常、执行效率及异常时的错误信息。

(4) **PowerCenter Repository Manager:** 顾名思义，这个客户端用于管理 Repository 本身，如创建文件夹，导入/导出 Mapping、Workflow，版本管理、部署，Repository 的清除等。

还有几个客户端工具，在特定的情况下也会使用，分别是 Data Transformation Studio、PowerCenter Mapping Architecture for Visio 和 Infomratica Developer。各客户端工具的作用如下。

(1) **Data Transformation Studio:** 主要用于开发、解析非结构化、半结构化数据的客户端工具，例如，解析、抽取 Excel 表格、PDF 文档或者某些行业规范数据，如 Jason、HL7、FIX、XBRL 等。在这里开发完成的程序，在发布之后，可以使用 PowerCenter UDO (Unstructure Data Option) Transformation 直接调用，与其他客户端实现无缝集成。

(2) **PowerCenter Mapping Architecture for Visio:** 在 ETL 开发过程中，有时需要开发大量的相似 Mapping，这个客户端工具主要用于支持 PowerCenter 批量开发，在后续的章节中将有详细的介绍。

(3) **Informatica Developer:** 这是一个新的、基于 Eclipse 的客户端，未来可能成为主流的开发工具，同样可以完成开发 Mapping 所需的所有功能。但此时此刻，该工具还有很多地方尚待完善。

## 1.3 PowerCenter Hello World

学习程序开发的人很多时候都是从一个 Hello World 程序开发开始学习的，它是一个简单入门、但很重要的工具和学习方法。学习 PowerCenter 同样从一个简单的 Hello World 开始，在上面的章节中基本了解了 PowerCenter 的架构和几个常用的客户端，通过 Hello World 开发过程会进一步加深读者对与开发相关的客户端的理解。

在大学阶段，多数人都学习过 C/C++程序的编写，本节我们用与 C/C++程序类比的方法介绍 PowerCenter Hello World，并展示在各个客户端中具体的开发过程，帮助初学者迅速完成一个 PowerCenter 全流程的开发，具体如表 1-1 所示。

表 1-1

步 骤	类 C/C++语言	PowerCenter 开发过程
1	<pre>#ifndef _A_H #define _A_H typedef struct EMP {     int ID;     char NAME; }EMP; #endif</pre>	<p>在客户端 PowerCenter Designer 中导入源表和目标表的结构定义 注：在 PowerCenter Designer 中导入的仅仅是表结构，与执行过程的表名无强相关</p>
2.	<pre>#include &lt;stdio.h&gt; #include "a.h" int main() {     一： 定义变量     struct emp emp_source;     struct emp emp_target;     二： 对变量进行赋值     emp_target.ID=emp_source.ID     emp_target.Name=emp_source.NAME     return 0; }</pre>	<p>在 PowerCenter Designer 执行的是：</p> <ol style="list-style-type: none"> <li>(1) 创建 Mapping (C 主程序)。</li> <li>(2) 拖动源和目标进入 Mapping (类似定义变量)。</li> <li>(3) 建立源和目标映射 (对变量进行赋值)</li> </ol>
3.	编译 cc -c one.c	PowerCenter 是一个非编译的环境，运行之前不需要进行编译
4.	运行 .\\a.out config_file	运行 Workflow，并提供相应的配置信息及参数。在 PowerCenter 中的载体是 Workflow 和 Session
5	监控	通过 Workflow Monitor 客户端进行监控

假如现在是一个全新的环境，在开始开发 PowerCenter 程序之前，还需要完成两个步骤，这两个步骤是一次性的或者叫公用的操作，即这两个步骤只需要做一次，后面的开发中不需要重复进行。这两个步骤是：

- ◎ 使用 PowerCenter 客户端连接域（Domain）和 Repository Service。
- ◎ 建立一个文件夹（Folder），用于开发或者学习。

### 1. 通过 PowerCenter 客户端连接域和 Repository Service

通过选择 Windows “开始” → “所有程序” → Informatica 9.6.1→Client→PowerCenter Client 命令找到 PowerCenter Repository Manager，即打开 Repository Manager 客户端，如图 1-4 所示。

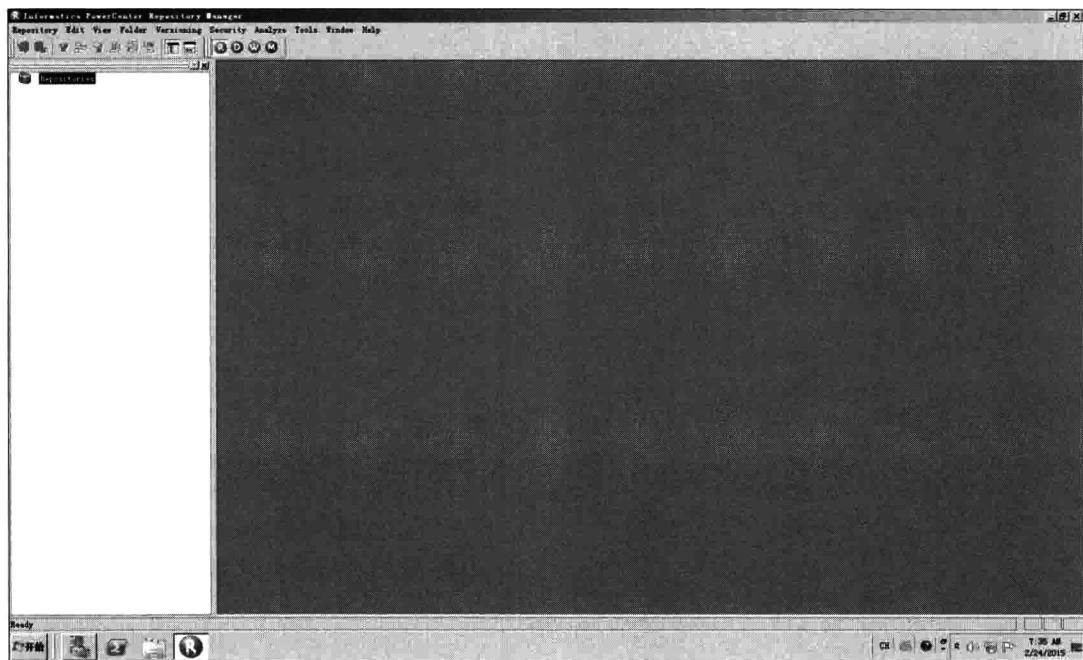


图 1-4

这时可选择菜单 Repository→Configure Domains 命令，弹出 Configure Domains 对话框，如图 1-5 所示。

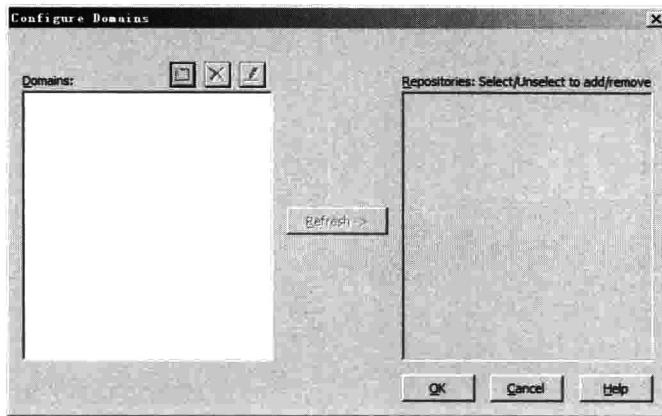


图 1-5

单击 Add a new domain 图标，在弹出的对话框中填写必要的信息，如图 1-6 所示。

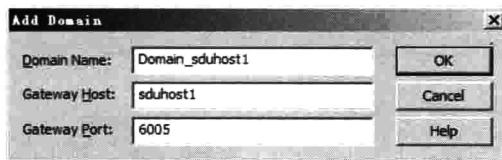


图 1-6

#### → 注释

- ◎ Domain Name：打开 Admin Console。Admin Console 是一个基于 Web 的管理控制台，一般是通过类似 URL “<http://sduhost1:6005>”进行访问的。sduhost1 是安装 PowerCenter 服务器的机器名，6005 是默认的端口号。在 Domain Navigator 树的最顶层就是 Domain Name。
- ◎ Gateway Host：这里要填写服务器的主机名。在 Windows 客户端，即安装 PowerCenter 客户端的 Windows 机器上还需要配置 hosts 文件，hosts 文件位于目录 C:\Windows\System32\drivers\etc 下，在 hosts 文件中增加类似条目：192.168.75.129 sduhost1。
- ◎ Gateway Port：默认端口为 6005，这是在安装过程中确定的。如果在安装过程中进行了修改，则需要使用修改后的端口号。

填写完必要的信息后，单击 OK 按钮，即可看到如图 1-7 所示的对话框。

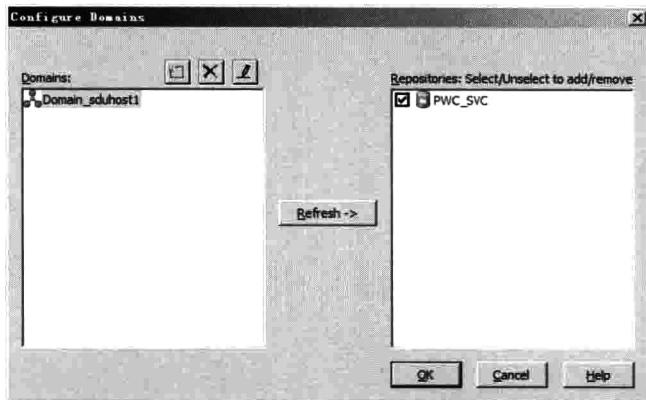


图 1-7

勾选 PWC\_SVC, 这里的 PWC\_SVC 即 Repository Service 的名字。同样也可以在 Admin Console 中找到相应的信息, 如图 1-8 所示。然后单击 OK 按钮。



图 1-8

双击 PWC\_SVC, 输入用户名和密码, 如图 1-9 所示。

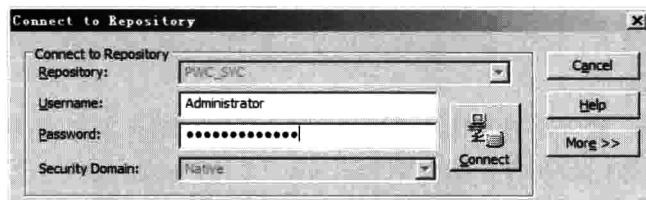


图 1-9

PowerCenter 安装之后默认的用户名是 Administrator, 密码是在安装时确定的。

→ **注释**

PowerCenter 的用户名、密码区分大小写。

单击 Connect 按钮, 进入如图 1-10 所示的界面, 显示客户端连接服务器成功。

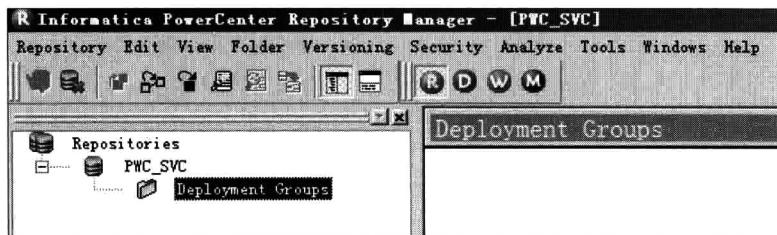


图 1-10

## 2. 创建一个文件夹 (Folder), 用于开发或者学习

在 Repository Manager 中, 选择菜单 Folder→Create 命令, 弹出 Create Folder 对话框, 如图 1-11 所示。

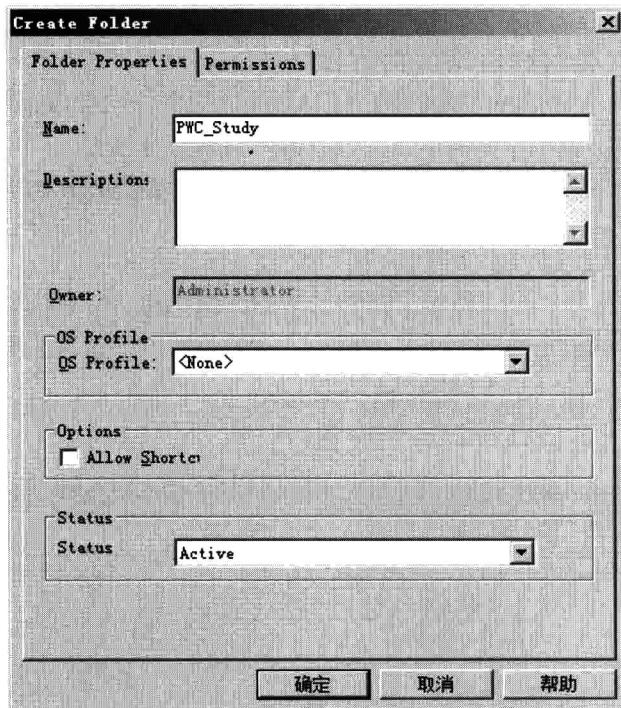


图 1-11

输入 Folder Name, 如 PWC\_Study, 单击“确定”按钮即可。结果如图 1-12 所示。

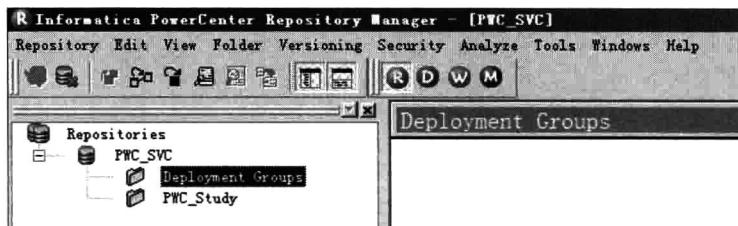


图 1-12

至此，已经完全准备好了开发一个 PowerCenter Hello World 的工作环境。

进行一个完整的 PowerCenter hello World 程序的开发，需要使用 PowerCenter Designer、PowerCenter Workflow Manager 和 PowerCenter Workflow Monitor 3 个客户端，它们的作用分别如下。

- ① PowerCenter Designer：定义源和目标的结构；开发一个 ETL 程序（Mapping）。
- ② PowerCenter Workflow Manager：将 Mapping 实例化，给 Mapping 运行提供必要的信息，如指定源和目标数据库，或者源、目标文件路径。
- ③ PowerCenter Workflow Monitor：对 Workflow 的运行过程进行监控。

仍然以 C/C++进行类比，具体操作步骤如下。

**步骤一：**定义头文件 (.h)，即定义源和目标的数据结构，在 ETL 过程中就是导入源和目标的表结构（注：这里需要的仅仅是表结构，与表存放的数据库无关）。

(1) 打开客户端 PowerCenter Designer，输入用户名、密码，并双击 Folder PWC\_Study，如图 1-13 所示。

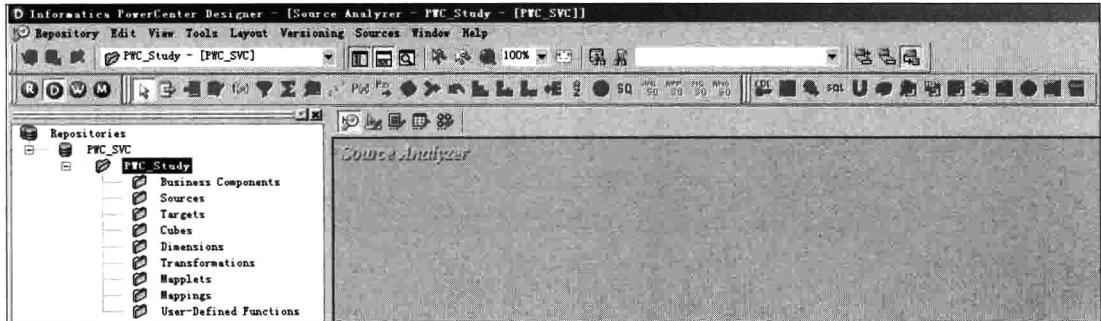


图 1-13

这时,注意这几个图标,分别表示 Source Analyzer、Target Designer、Transformation Developer、Mapplet Designer、Mapping Designer,选择不同的图标,菜单也会根据选择的功能不同动态地改变。初次参与开发的人员有时候找不到菜单往往是这个原因。

(2) 单击 Source Analyzer 图标, 定义源表的结构。

定义源表的结构有多种方式,在这里既然是 Hello World 程序,选择最常用、最简单的方式进行。选择菜单 Sources→Import from database 命令,弹出 Import Tables 对话框,如图 1-14 所示。

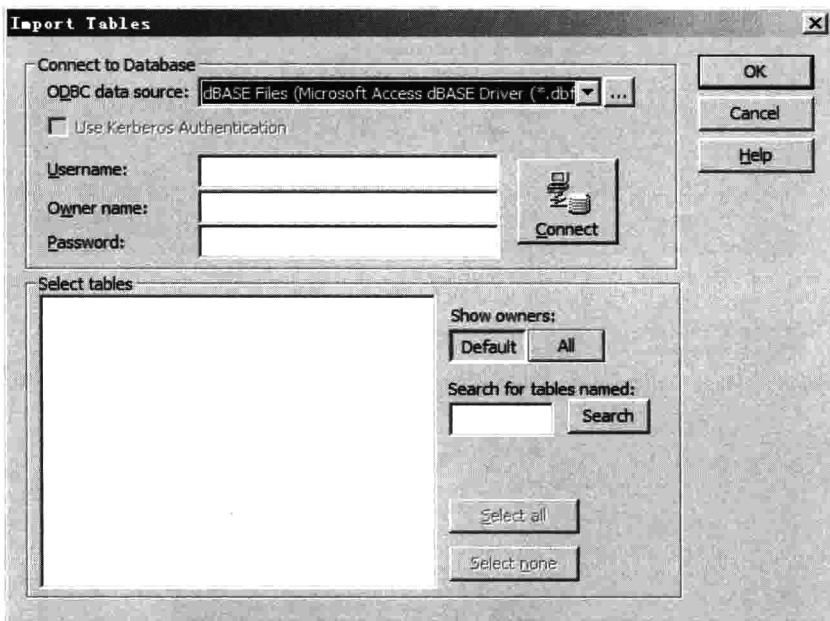


图 1-14

(3) PowerCenter 是通过 ODBC 导入源和目标表结构的。如果是第一次使用,则需要创建一个 ODBC DSN。步骤是:单击图标→选择“系统 DSN”选项卡→单击“添加”按钮,如图 1-15 所示。

(4) 选择驱动程序“DataDirect 7.1 Oracle Wire Protocol”,以 Oracle 数据库为例,如图 1-16 所示。

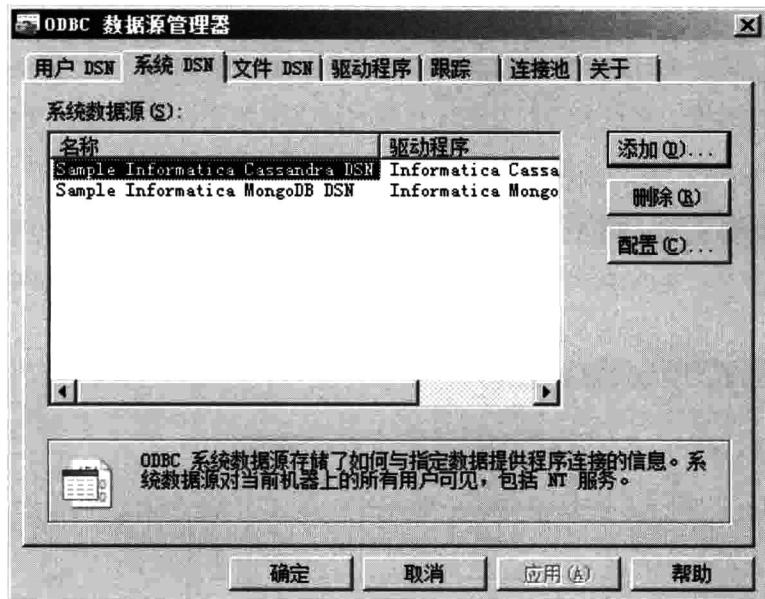


图 1-15

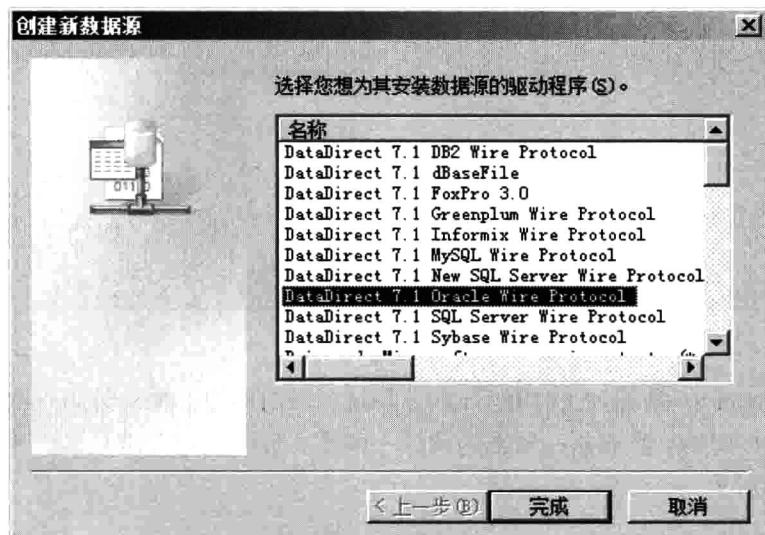


图 1-16

(5) 填写必要的数据库信息，如图 1-17 所示。

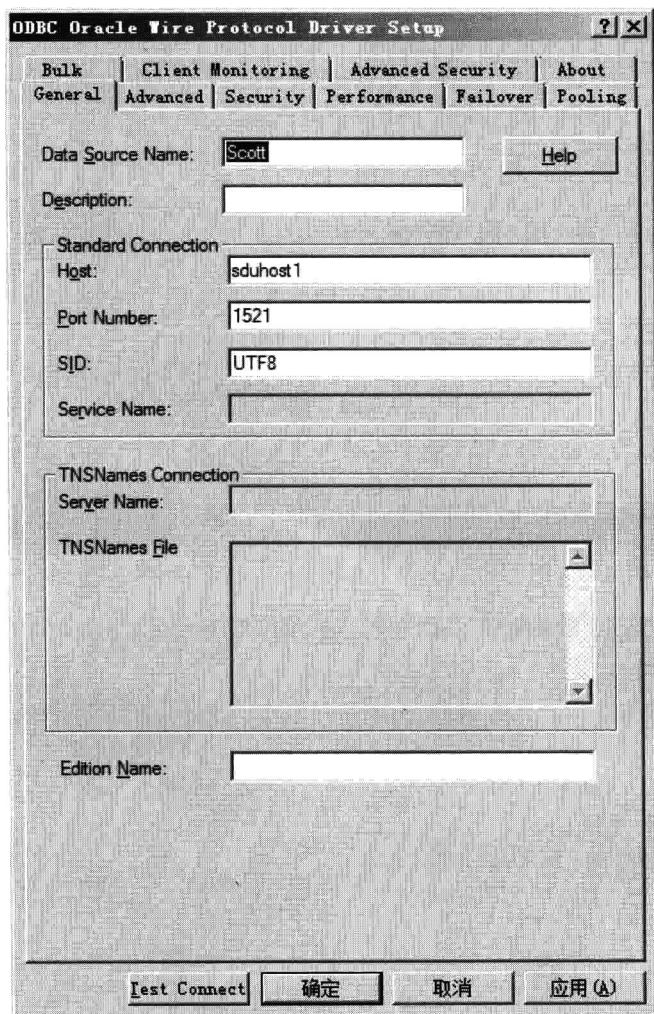


图 1-17

(6) 返回到如图 1-18 所示的界面，选择刚刚建立的 ODBC data source: Scott，并输入相应的用户名、密码。

(7) 单击 Connect 按钮，可看到如图 1-19 所示的对话框，默认显示的是 Scott 用户下的列表。

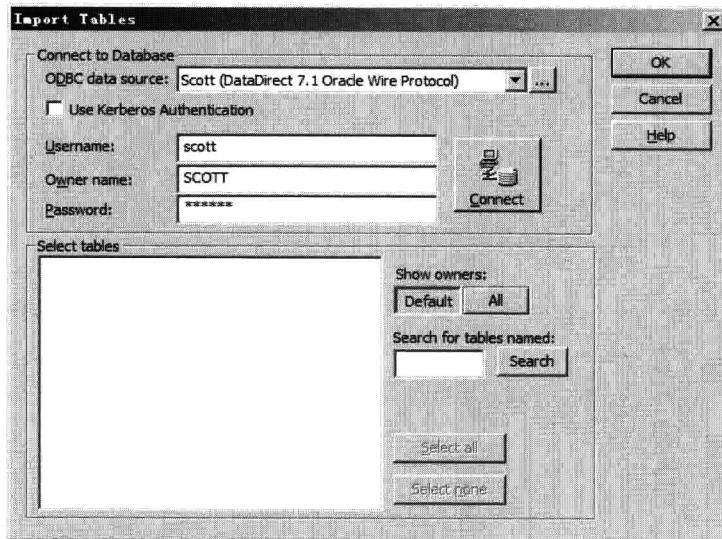


图 1-18

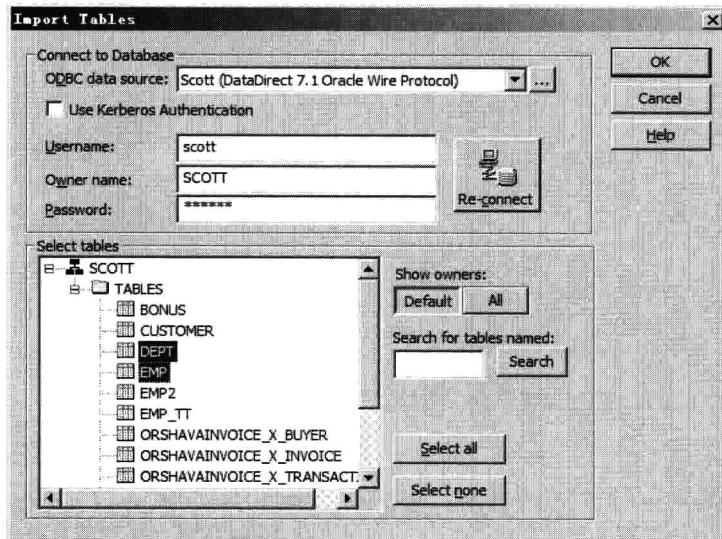


图 1-19

(8) 选择需要导入的表，如图 1-19 中的 DEPT 和 EMP，单击 OK 按钮，这样就定义了源表的结构，如图 1-20 所示。

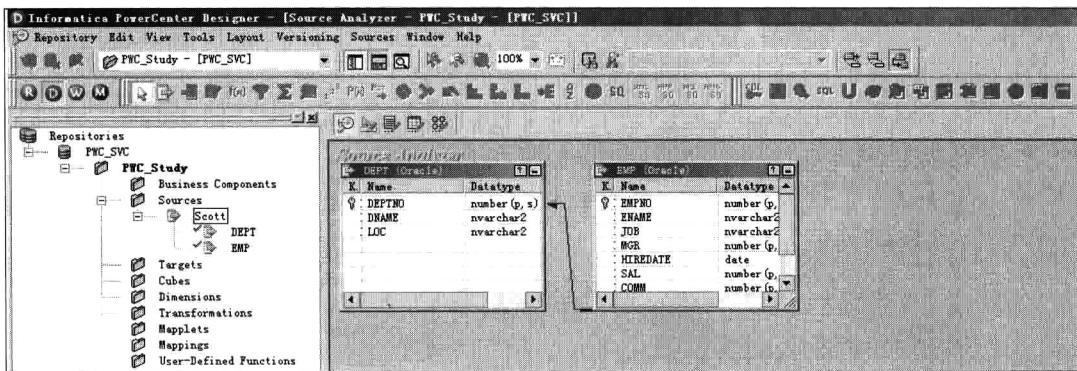


图 1-20

(9) 可以用同样的方法定义目标表的结构。单击 Target Designer 图标，其他步骤与定义源的结构相似。选择菜单 Targets→Import from Database 命令，弹出如图 1-21 所示的对话框。

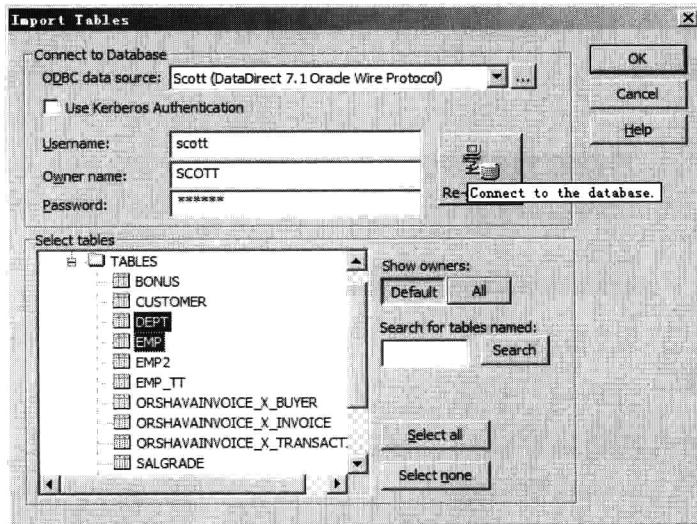


图 1-21

(10) 选择 ODBC data source；输入用户名、密码；单击 Connect 按钮，选择需要导入的表结构。

如果看见图 1-22，则表明你已经成功定义了完整的源和目标的结构，完成了头文件(.h)的创建。

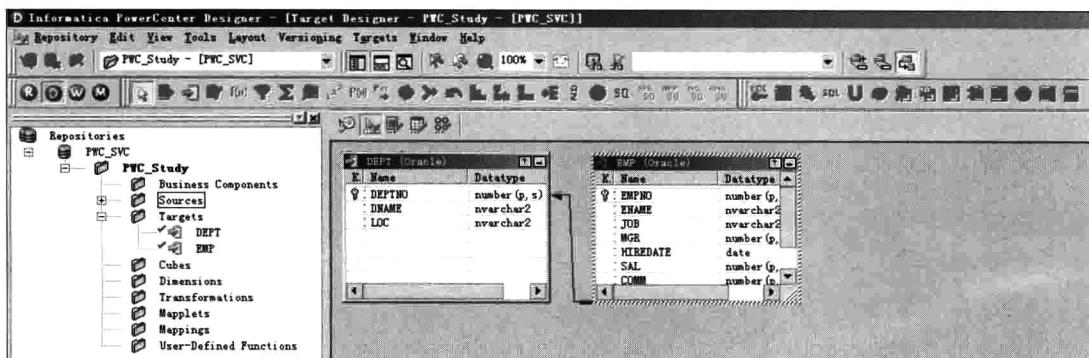


图 1-22

## 步骤二：开发一个简单的 ETL 过程（一个 C Main 程序）。

(1) 定义一个主程序名称，即在 PowerCenter 中定义一个 Mapping 名。

单击 Mapping Designer 图标 ，选择菜单 **Mappings→Create** 命令，弹出如图 1-23 所示的对话框，输入新的 Mapping name，这里输入的名字为 m\_emp。Informatica Velocity Methodology 提供了 PowerCenter 内部建议的命名规范，如果你对这个命名规范感兴趣，请参照此文档。

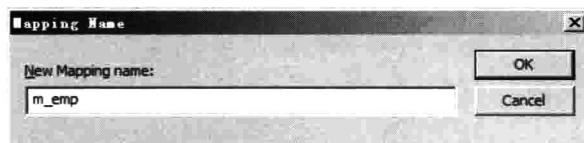


图 1-23

(2) 定义变量。

在 PowerCenter 中定义一个源和目标的变量非常简单，不用写代码，只需要拖动即可完成。

在 PWC\_Study 文件夹下选择 **Sources→Scott→EMP 表**，按住鼠标左键，将 EMP 表拖入 Mapping Designer 的工作区。

同样定义目标表的变量，在 PWC\_Study 文件夹下选择 **Targets→EMP 表**，按住鼠标左键，将 EMP 表拖入 Mapping Designer 的工作区。这时两个变量定义结束，结果如图 1-24 所示。

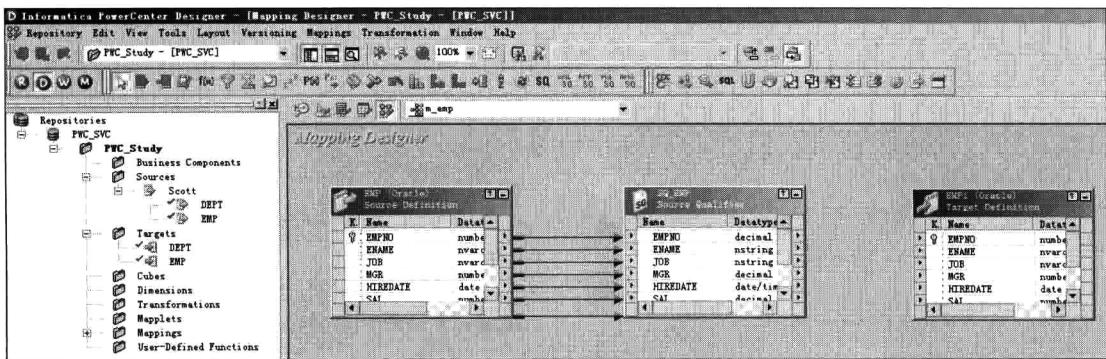


图 1-24

### (3) 为变量赋值。

在 PowerCenter 中为变量赋值不用写  $A := B$  这样的表达式，只需选中  $SQ\_EMP$  中所有的列，按住鼠标左键，将其向目标表拖动，并在目标表上释放鼠标左键。最后的结果如图 1-25 所示。

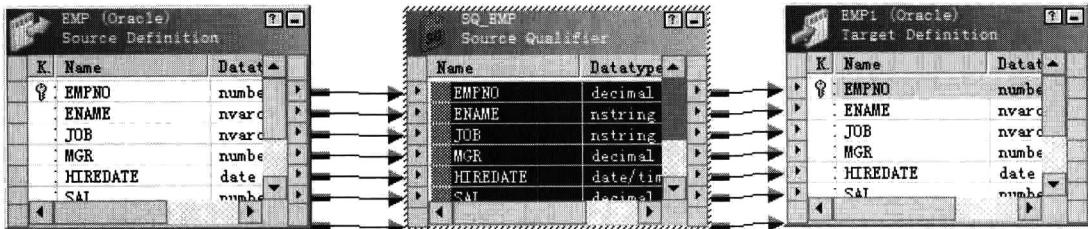


图 1-25

现在一个简单的 Mapping 开发已经完成，还有一个重要的步骤千万不能忘记，那就是保存。可以选择菜单 Repository→Save 命令进行保存，更常用的是使用  $Ctrl+S$  组合键方式进行保存。

假如看到图 1-26，在  $m\_emp$  前有个红色的感叹号，这时候的 Mapping 是无效的，需要修正后才能进行下一步的工作。

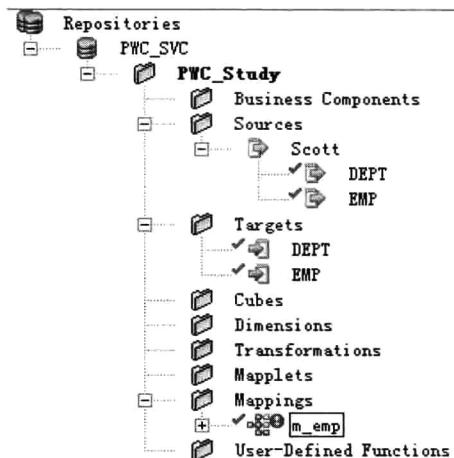


图 1-26

正确的保存结果如图 1-27 所示。

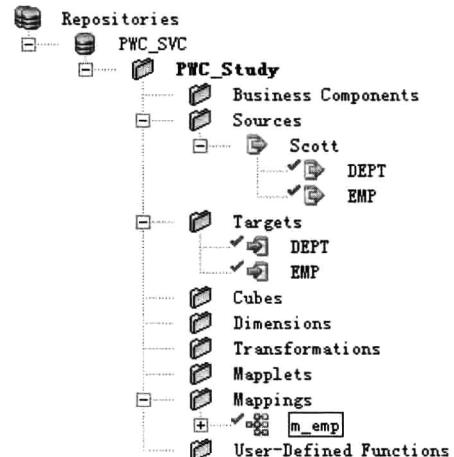


图 1-27

**步骤三：**编译。PowerCenter 是元数据驱动的架构，不需要进行编译，因此这个步骤可以忽略。

**步骤四：**提供相应的配置文件，并执行。这个步骤主要在 PowerCenter Workflow Manager 中进行。

(1) 打开 Workflow Manager，并选中 Workflow Designer，如图 1-28 所示。这里也是所谓的动态菜单，选择不同的功能区，菜单会动态地发生变化。



图 1-28

(2) 选择菜单 Workflows→Wizard 命令，在弹出的对话框中输入 Workflow 的名字，如 wf\_emp (命名规范请参考 Velocity Methodology); 选择 PowerCenter Integration Service，如图 1-29 所示的 PWC\_IS。

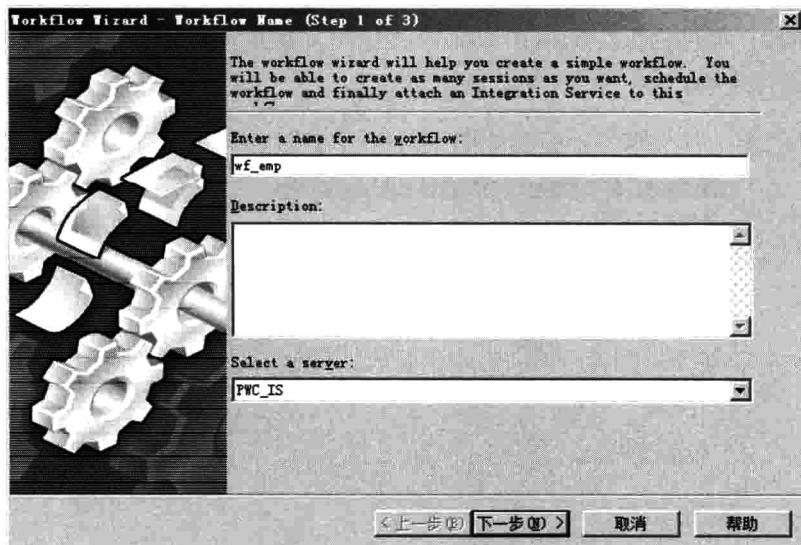


图 1-29

(3) 单击“下一步”按钮，可见如图 1-30 所示的对话框。

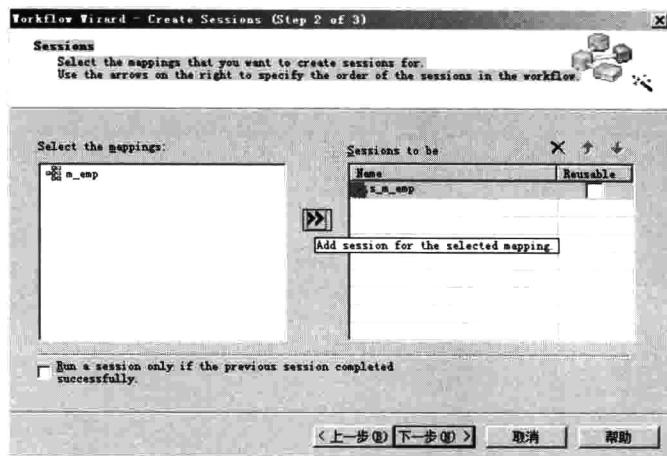


图 1-30

这时，在图 1-30 的左边可以看到刚刚创建的 Mapping: m\_emp，然后单击 Add Session for the selected Mapping 按钮 ，这时，在图 1-30 右边出现了 s\_m\_emp，s\_m\_emp 是一个 Session。现在 PowerCenter 中最重要的 3 个概念（Mapping、Session、Workflow）全部出现了。Mapping 是一个程序，但它不直接可以执行；Session 是一个 Mapping 的实例，指定相关的配置信息后，可以执行；Workflow 可以执行一个或者多个 Session，对 Session 或者其他 Task 组件进行排程。

(4) 单击“下一步”按钮，弹出如图 1-31 所示的对话框。

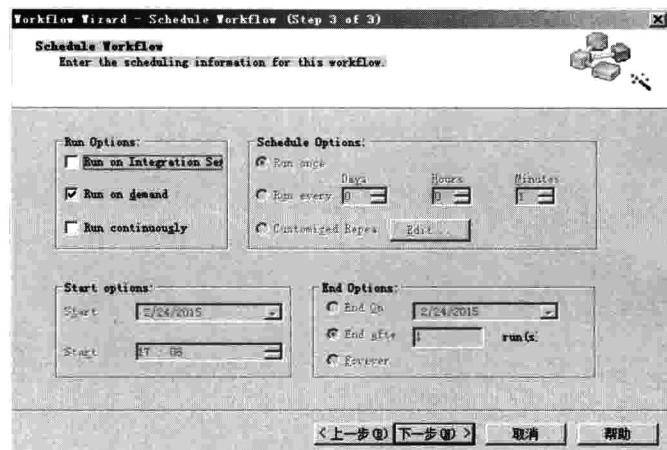


图 1-31

(5) 这里主要是 PowerCenter 的调度过程，不影响 Hello World 过程，直接单击“下一步”按钮，弹出如图 1-32 所示的对话框。

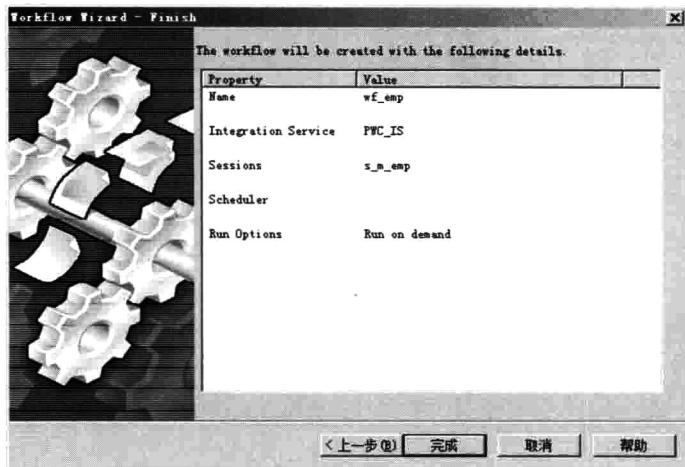


图 1-32

(6) 单击“完成”按钮，完成配置文件设置，结果如图 1-33 所示。



图 1-33

既然是一个 ETL 过程，重要的是指定数据源和目标，即为 Mapping 执行提供相关的参数，这个参数并非针对 Mapping，而是指定在 Mapping 的实例 Session 上。为什么不是在 Mapping 上，而是在 Session 上？这方面的好处留给大家来思考。

这时双击 s\_m\_emp，选择 Mapping Tab；选择 Sources→SQ\_EMP。为源表指定实际的数据库，这里选择了 scott，如图 1-34 所示。scott 连接是从哪里来的，又是如何配置的呢？

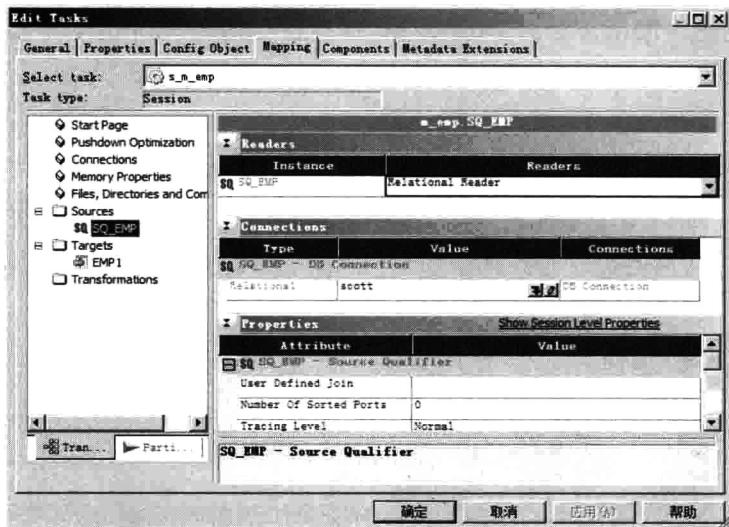


图 1-34

如何在 Workflow Manager 中建立一个数据库连接？

这个配置也是一次性的，不需要重复进行，建立的连接是对所有的 Session 和 Workflow 公用的，不需要对每个 Session 单独建立数据库连接。建立过程如下所述。

首先，选择菜单 Connections→Relational 命令，进入如图 1-35 所示的界面。



图 1-35

其次，单击 New 按钮，输入连接的名字如 scott；以及数据库的用户名、密码和连接串，如图 1-36 所示。以 Oracle 为例，这个连接串是 Oracle tnsnames.ora 文件中设置的连接串，tnsnames.ora 文件指的是在 PowerCenter 服务器上的文件，并非 Oracle 服务器端的文件。

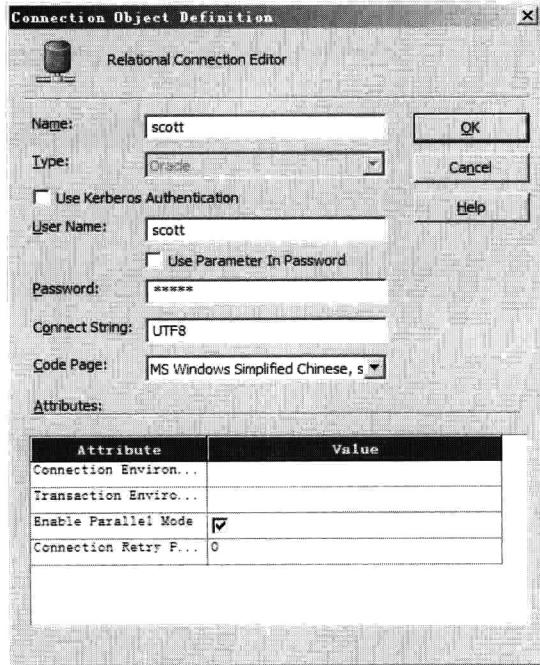


图 1-36

在这个例子中，我们希望把数据库的数据导出成为一个由“|”分隔的、UTF-8 格式的文件。

(7) 这时在图 1-34 中选择 Targets→EMP1，设置 Writers 为 File Writer；Output Type 为 File；Output file directory 为 \$PMTargetFileDir\；Output filename 为 emp.out。单击“确定”按钮，如图 1-37 所示。

还有最重要的一步是保存，按 Ctrl+S 组合键即可。

现在一个完整的 ETL 过程已经开发完成，可以执行了。

启动 Workflow 有很多种方法，在 Hello World 程序中，可以选择菜单 Workflows→Start Workflow 命令。

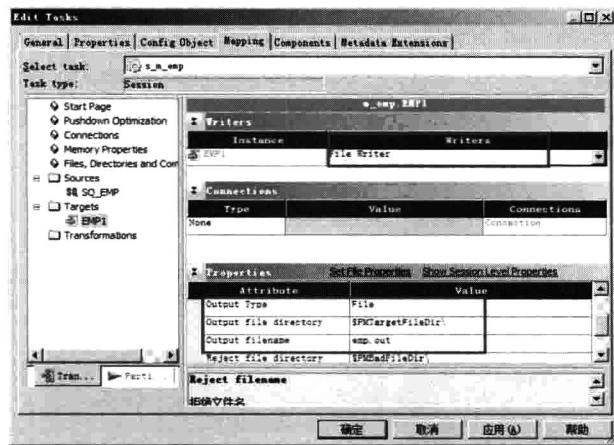


图 1-37

### 步骤五：监控运行。

启动 Workflow 之后，打开 Workflow Monitor 客户端，即可看见如图 1-38 所示的视图。

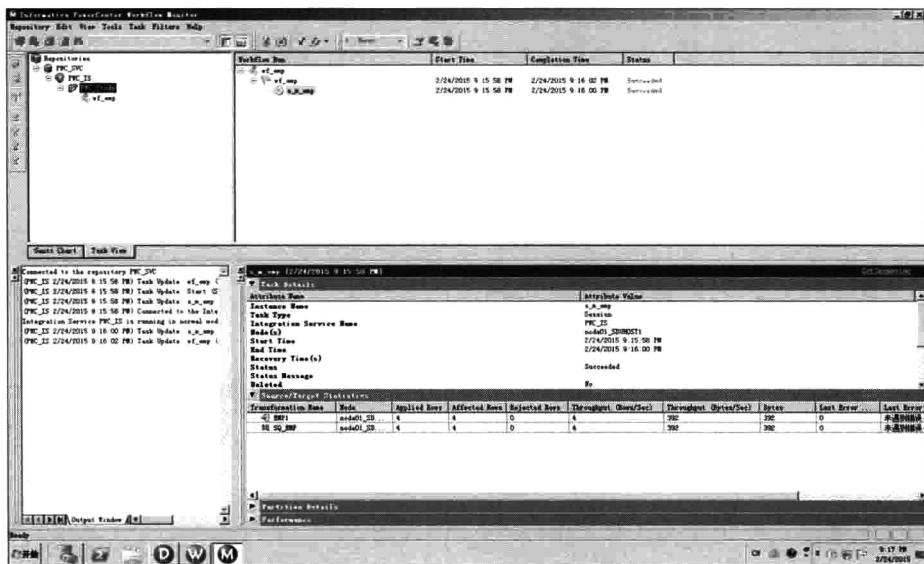


图 1-38

双击 Session，如 s\_m\_emp，即可查看该 Session 运行的详细信息，包括运行状态，开始、结束时间，读取、写入行数，吞吐量及运行时的错误等。

# 第 2 章

# PowerCenter 基础组件

---

PowerCenter 最经常被使用的场景是作为数据仓库或者大数据项目的 ETL 工具，其中 T 的主要工作是由其包含的各种各样的组件组合实现的，这些组件被叫作 Transformation。本章将详细介绍每个组件的功能及基本用法。

## 2.1 Source

---

PowerCenter 是一个 ETL 工具，E（Extract）即为对数据源的抽取。因此数据源是 PowerCenter 中一个最重要的概念。数据源可以是数据库表、文本文件、XML 文件、SAP 等应用系统、Hadoop、MQ 等，这也是 PowerCenter 强大功能的一部分。

本书无法将所有的数据源做一一介绍，只能重点介绍一下两个最普通但最重要的数据源：文本文件源和数据库源。

数据源定义的创建有两种方式：

- 手工创建。
- 通过数据库、样例文件导入。

手工创建时，在 PowerCenter Designer 中选择菜单 Sources→Create 命令，即可看到如图 2-1 所示的对话框，选择 Database type，输入源名字即可。麻烦的事情在后面，还需要输入每一个字段名字、数据类型、主外键。在一个项目中少则几十张、多则上百张表的情

况下，这种创建方法几乎没有人使用，多数情况下是通过数据库或者文本文件导入。

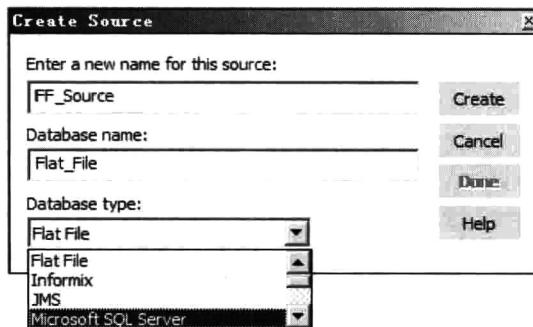


图 2-1

通过数据库、样例文本文件导入是最常用、最简捷的方法，也符合数据仓库项目的开发过程。首先定义模型，然后才进行 ETL 开发。执行的步骤是：首先建立对应数据库的 ODBC 连接（这个连接仅仅是为了导入数据库表结构，与执行过程无关），然后通过菜单 Sources→Import from Databases 或者 Import from File 命令导入源的结构。

下面是对文本文件源和数据库源的相关特性的介绍，以帮助大家理解、配置这两种典型的数据源。

### 2.1.1 数据库源

上面提到过，为了方便起见，一般数据库源定义都是通过 ODBC 导入的。默认导入的源表结构继承了数据库中表结构的定义。

#### 1. 在 PowerCenter Designer 中

第一个关注点：源定义是按照 Owner Name 在左边的树结构中进行分组的。Owner 名字继承了 ODBC 的名字，这个 Owner 名字在导入后也可以进行修改，如图 2-2 所示。

#### → 注释

为了预防源的重复定义，建议项目组在客户端使用统一的 ODBC 命名规范。

第二个关注点：理论上，源表结构定义继承了数据库中的表定义，但是实践中有可能导入后数据类型发生了变化，如表中为 varchar2，而导入后变为 nvarchar2，从而引起 Session

执行异常。另外 PowerCenter 的主外键是逻辑定义主外键，可以与数据库不同，如图 2-3 所示。这两点在实践的项目中经常用到，因此特此说明。

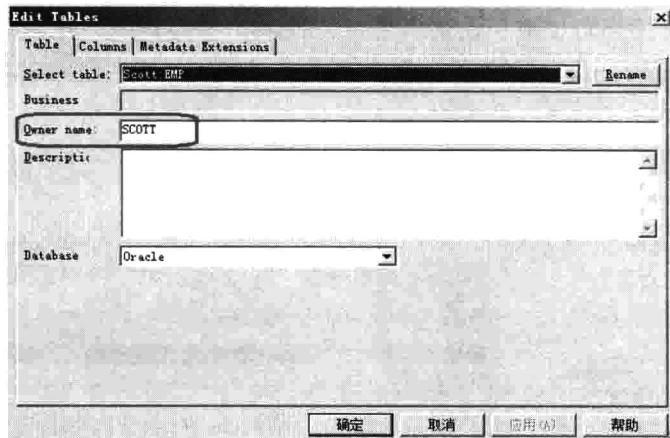


图 2-2

	Column Name	Datatype	Prec	Scale	Not Null	Key type
1	EMPNO	number(5,0)	4	0	<input checked="" type="checkbox"/>	PRIMARY KEY
2	ENAME	varchar2	10	0	<input type="checkbox"/>	NOT A KEY
3	JOB	varchar2	9	0	<input type="checkbox"/>	NOT A KEY
4	MGR	number(5,0)	4	0	<input type="checkbox"/>	NOT A KEY
5	HIREDATE	date	10	0	<input type="checkbox"/>	NOT A KEY
6	SAL	number(5,2)	7	2	<input type="checkbox"/>	NOT A KEY
7	COMM	number(5,2)	7	2	<input type="checkbox"/>	NOT A KEY
8	DEPTNO	number(5,0)	2	0	<input type="checkbox"/>	NOT A KEY

图 2-3

第三个关注点：建立导入数据源的 ODBC 使用哪个驱动程序，是 Data Direct 驱动还是数据库自带的驱动。使用哪个驱动没有一定之规，PowerCenter 不同的版本需要用不同的驱动，建议大家根据自己的 PowerCenter 版本进行尝试，如果发现导入的数据类型不一致，那么选择更换其他的驱动。

## 2. 在 Workflow Manager 中

除了大家已经非常熟悉的指定数据库连接，还有一个经常使用的功能，就是在 Workflow Manager 中还可以重新指定表名和 Schema（有时也叫用户名）。如图 2-4 方框标识部分，设置 Owner Name（用户名或者 Schema 名）为 Scott2，设置 Source Table Name 为 EMPX。

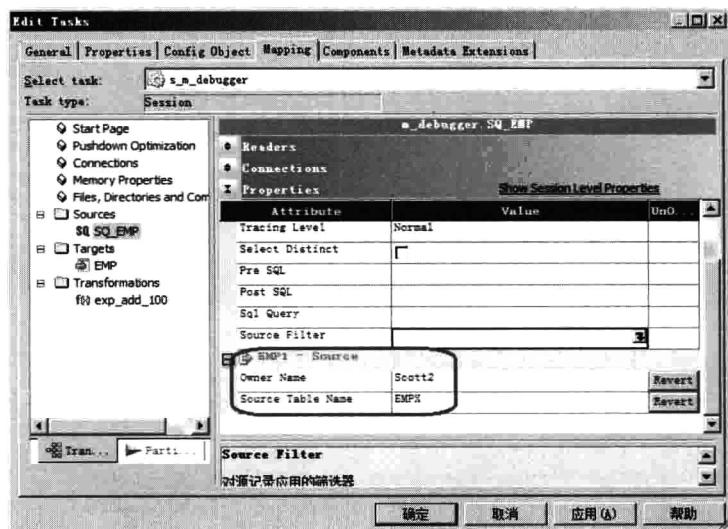


图 2-4

### 2.1.2 文本文件源

相对于数据库源，文本源更复杂一些，有更多的属性需要关注。

#### 1. PowerCenter Designer 和 Workflow Manager 的公用属性

有些属性在 PowerCenter Designer 和 Workflow Manager 中都可设置、变更，但 Workflow Manager 中的优先级更高。笔者建议在项目中最好有个规范文档，说明这些属性应该设置的位置及修改顺序建议。

在 PowerCenter Designer 中，在 Source Analyzer 中选择一个文本文件源，双击打开其 Table Tab，单击 Advanced 按钮，或者在 PowerCenter Workflow Manager 中选择包含文本文件源的 Session→选择 Mapping Tab→选择 Sources，找到关注的文本文件数据源，有一个非

常不起眼的 Set File Properties 选项，选择该选项，就可以看到如图 2-5 所示的文本文件源定义窗口。

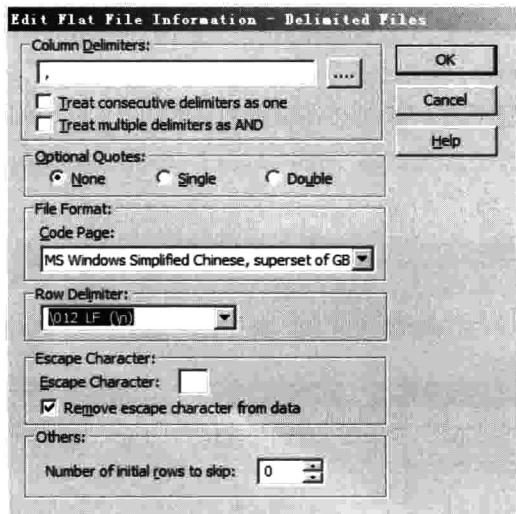


图 2-5

在图 2-5 中以下信息都非常关键，而且常常被用到。

- ① **Column Delimiters**（分隔符）：指定字段间分隔符，也可以支持多字符的分隔符，如“|||”。
- ② **Optional Quotes**（引号）：有的文件中对某些字符值使用引号，如“新闻。报道”，以保证类型的正确性。PowerCenter 可以支持字符类型数据使用单引号或者双引号。
- ③ **Code Page**（字符集）：文本文件与数据库一样是有自己的字符集的，需要选择正确的字符集。如希望文本的编码是 UTF-8，还是 GB 2312 等。
- ④ **Row Delimiter**（行分隔符）：可以指定不同的行分隔符。假如在 UNIX/Linux 上需要指定多分隔符，还可以在表的最后一列后面附加一个字段，指定为常量，即分隔符。
- ⑤ **Escape Character**：可以删除所有的 Escape Character。
- ⑥ **Number of initial rows to skip**（忽略的行数）：有时要读取的文本文件有表头，就需要标明从第几行开始读取真正的数据。忽略一行，则为从文件的第二行开始读取数据。

## 2. 在 PowerCenter Designer 中

在 Flat File 源的 Properties Tab 中，有一个常用属性 Add Currently Processed Flat File Name Port，选中这个属性，PowerCenter 将在 Column 中增加一个列，名为 CurrentlyProcessedFileName，如图 2-6 所示。这个列包含的是源文件的“文件名”，并可以在后续的 ETL 过程中进行处理。例如，在电信行业的项目中，接口文件的命名经常会保存一个时间戳，在 ETL 处理中还会对这个时间戳进行处理，比如将此时间戳作为目标表的一个字段，这时就可以使用这个选项。

Column Name	Datatype	Prec	Scale	Not ...	Format	Key Type
Input_SQL	string	500	0	<input type="checkbox"/>		NOT A KEY
CurrentlyProcessedFileName	string	256	0	<input type="checkbox"/>		NOT A KEY

图 2-6

## 3. 在 Workflow Manager 中

在 Workflow Manager 中，对于 Flat File 有两个关键属性需要说明。

- ◎ **Input Type:** Input Type 有两个选项。
  - ❖ **File:** 默认值，即文本文件输入。
  - ❖ **Command:** 当选择 Command 时，PowerCenter 将把命令的输出作为输入。如 Command:ls -al|tail 10，PowerCenter 将读取 ls -al 命令输出的后 10 行数据。在新的版本中，还可以用 Command 生成一个文件列表。如输入文件的文件名不是固定名字，而是动态的，这时候使用 Command 输入就是一个比较好的选择，当然还有其他的方法。
- ◎ **Source File Type:** Source File Type 可以是 Direct 或者 Indirect。当选择 Direct 时，需要给 PowerCenter 指定一个包含实际数据的文件名；而当使用 Indirect 时，为 PowerCenter 指定的是一个文件列表。如 All\_customer\_list.txt，它的内容是 Beijing\_customer.dat 和 Shanghai\_customer.dat，这两个\*.dat 文件才是真正包含数据的文件。

## 2.2 Target

在 PowerCenter 中创建 Target 的过程与创建源的过程相同，可以手工创建，也可以通过 ODBC 导入。如希望了解详细的目标创建过程，请参考上一节 Source 的定义。

除了手工创建和使用 ODBC 导入，还可以在 PowerCenter Designer 中将 Source 直接拖入 Target Designer 的工作区而生成目标。但 PowerCenter 不支持将 Target 拖动到 Source Analyzer 工作区生成新的 Source 定义。

### 2.2.1 数据库目标

数据库目标表的定义与在 PowerCenter Designer 中数据库源表的定义过程基本相同。

在 Workflow Manager 中，与 Source 的定义也比较相似，可以是默认的表名（Mapping 中使用的 Target 表名）和 Schema 名字（Schema 与默认的数据库连接用户名一致）。如果不使用 Table Name Prefix 和 Target Table Name，PowerCenter 将默认使用 PowerCenter Designer 中使用的目标表名。修改 Table Name Prefix，可以修改默认的 Schema 名字；修改 Target Table Name，可以修改默认的目标表名，如图 2-7 所示。

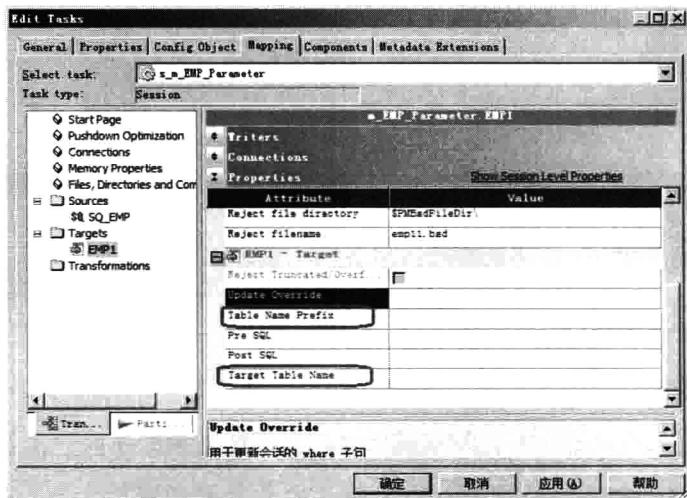


图 2-7

在 Workflow Manager 中还有 3 个经常使用的选项。

- ◎ Truncate target table option: 在将数据导入目标表之前将目标表清空。
- ◎ Target Load Type: 有两个选项 Normal 和 Bulk, 一般情况下使用 Normal。在 Bulk 可用的情况下, 可以提高一定的写性能, 比较适合 SQL Server、Sybase ASE 和 Oracle 无索引、无主键的情况。当 Oracle 表有索引或者主键时, 将引起数据库出错, 造成 Session 终止运行。
- ◎ Reject File: Reject File 是非常有效的异常数据的控制机制。当发生主键冲突时, 数据无法插入数据库。在这种情况下, PowerCenter 不会将数据直接丢弃, 而是将无法写入目标表的数据写入 Reject File 中。开发或者管理人员可以在问题得到解决后, 从该文件中进行数据恢复。

## 2.2.2 文本文件目标

文本文件作为目标与文本文件作为源, 基本属性是相似的。

### 1. 在 PowerCenter Designer 中

可以为文本文件目标定义文件名端口, 这样就可以将不同的数据写入到不同的文件中。如果需要将不同的记录写入不同的文件, 则选择图 2-8 中标注的部分 Add Filename Column to this table, PowerCenter 会自动增加一个新列 Filename。

	Column Name	Datatype	Prec	Scale	No.	Format	Key Type
1	SQL_Statement	string	500	0	<input type="checkbox"/>		NOT A KEY
2	Output_Data	string	500	0	<input type="checkbox"/>		NOT A KEY
3	Output_Error	string	500	0	<input type="checkbox"/>		NOT A KEY
4	Filename	string	255	0	<input checked="" type="checkbox"/>		NOT A KEY

图 2-8

### 2. 在 Workflow Manager 中

当文本文件作为目标时, 可以直接将结果写入本地的文本文件, 还可以将此文本文件写入远程的 FTP、Queue, 或者使用数据库 Loader 工具将文本文件快速加载到数据库, 如 Oracle SQL Loader、Sybase IQ 或者 Teradata Fastload 等, 如图 2-9 所示。

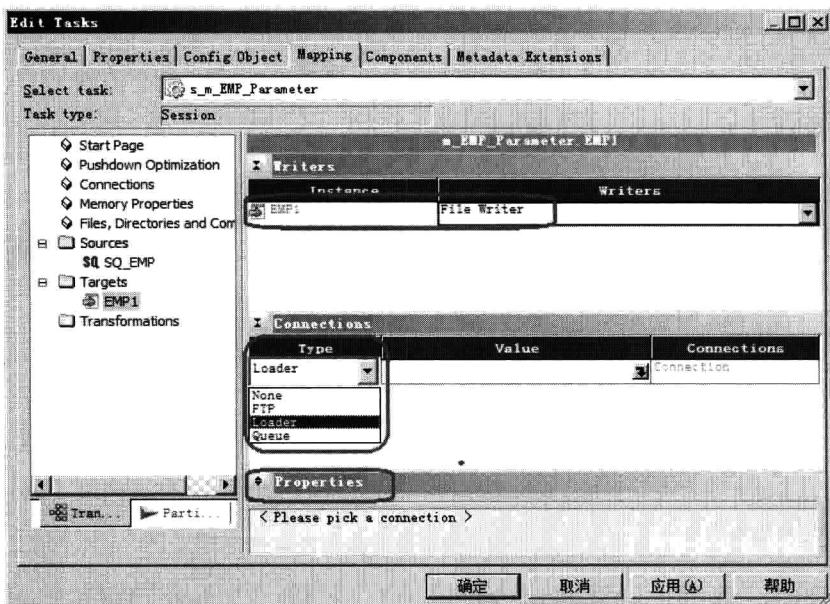


图 2-9

在图 2-9 中，选择打开 Properties 部分，会发现有更多的选项，如下所述。

- Append if Exists：如果文件存在，那么可以将新数据追加到原有的文件中。
- Create Target Directory：如果目录不存在，则自动创建目录。
- Header Options：是否在输出文件中添加文件头，可以直接将列名创建为文件头，也可以使用 Head Command 创建文件头。
- Head Command：创建文件头的命令。
- Footer Command：创建文件尾的命令。
- Merge Type：该选项经常与 Partition 配合使用，请参照 Partition 部分的介绍。

## 2.3 Expression 表达式

Expression 是最常用的 Transformation 之一。在 PowerCenter 中它使用的图标是

Expression 主要用于行级表达式的计算，如对某个字段进行四则运算、对某些字符串进行计算等。

**案例：**虚拟一个简单的场景来介绍 Expression 组件的一些基本属性和功能。现在是年底，公司要将所有人的工资调涨 10%，将调涨后的结果输出为文本文件给财务部审核。

最终完成设计的 Mapping 结果如图 2-10 所示。

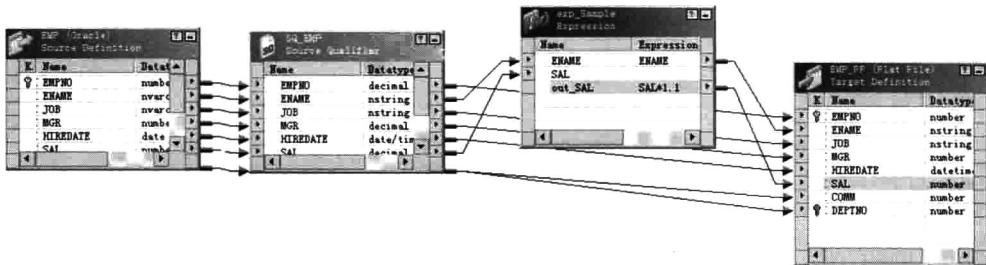


图 2-10

具体操作步骤如下。

(1) 以 Hello World 的 Mapping 为起点进行。复制一个 Hello World 的 Mapping，与 Microsoft Word 中的复制操作一样，按 Ctrl+C 组合键复制，然后按 Ctrl+V 组合键粘贴即可，重命名复制后的 Mapping 为 m\_emp\_exp。

(2) 在工具栏中单击 Expression 图标 ，如图 2-11 所示。

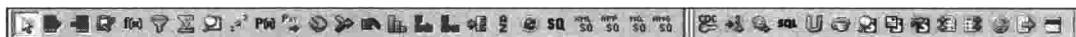


图 2-11

(3) 单击 Mapping Designer 的工作区，即可以在 m\_emp\_exp 中增加一个空白的 Expression Transformation，如图 2-12 所示。

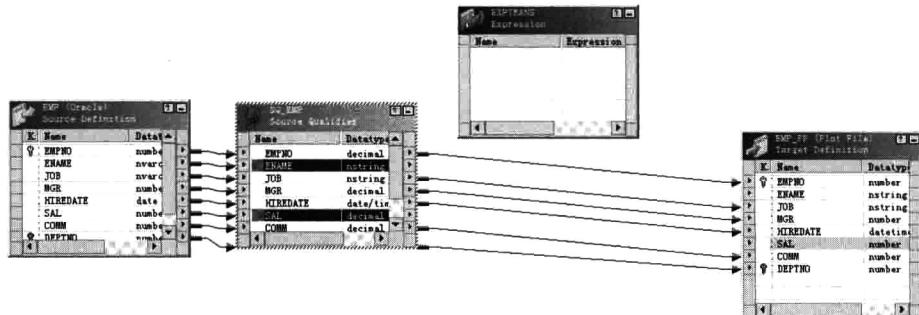


图 2-12

(4) 在 SQ\_EMP 中选中 ENAME 和 SAL 字段，将这两个字段拖入 Expression 组件中，如图 2-13 所示。

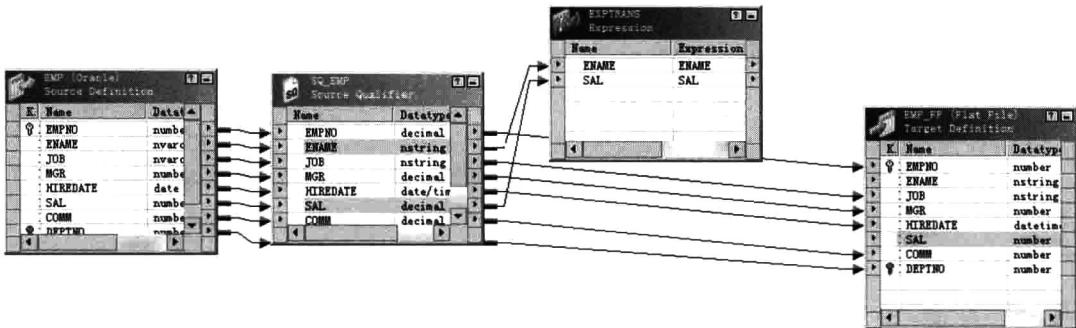


图 2-13

(5) 双击 Expression 组件，该组件即进入编辑状态，如图 2-14 所示。

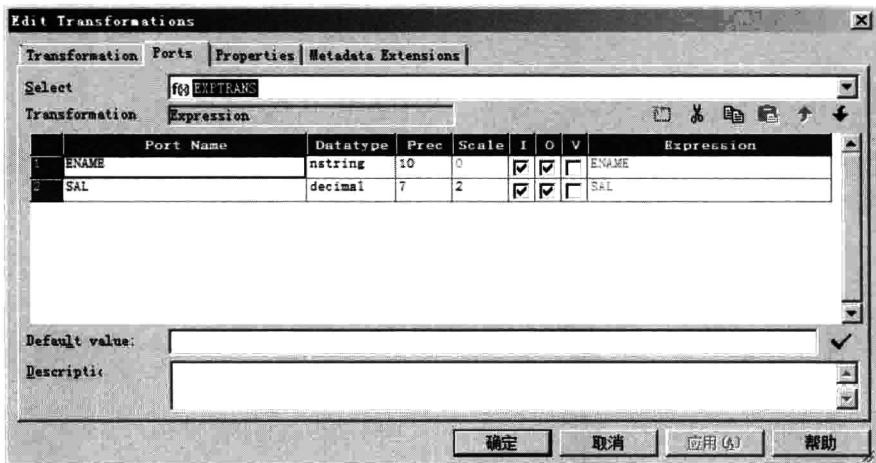


图 2-14

可以注意到，目前有两个端口，均是刚刚拖入的字段，分别是 ENAME 和 SAL，它们有各自的数据类型、精度和长度。

特别的是，在图 2-14 中有几个复选框分别标注了 I、O、V，其中 I 代表 Input，O 代表 Output，V 代表 Variable。当一个端口的 I 和 O 分别被选中时，说明这个字段同时作为输入和输出。

(6) 根据约定的计算逻辑，需要对工资调增 10%，即需要完成一个  $SAL * 1.1$  的运算。这时，需要将 SAL 的 O 部分取消勾选，并增加一个新字段 OUT\_SAL，将 OUT\_SAL 赋值为  $SAL * 1.1$ ，如图 2-15 所示。

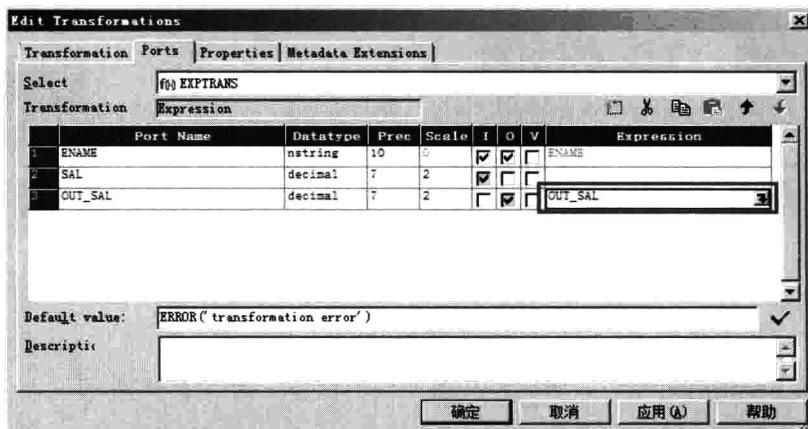


图 2-15

注意图 2-15 中 SAL 目前仅为 Input，OUT\_SAL 仅为 Output。

(7) 如何将 OUT\_SAL 赋值为  $SAL * 1.1$  呢？单击图 2-15 方框部分，进入表达式的编辑界面，如图 2-16 所示，输入表达式“ $SAL * 1.1$ ”，单击 OK 按钮。

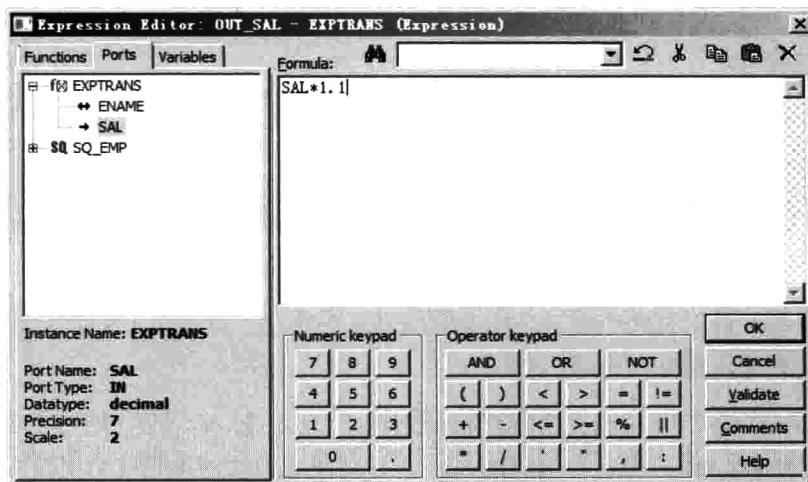


图 2-16

### → 注释

表达式可以是简单的  $SAL*1.1$ ，也可以是略为复杂的表达式。下面列举两个在 PowerCenter 经常会用到的表达式。

(1)  $IIF(DEPTNO=10 \text{ OR } DEPTNO=20, SAL*1.1, SAL*1.2)$ : 该表达式的含义是，部门 10 和 20 的工资增加 10%，其他部门的工资增加 20%。

(2)  $DECODE(DEPTNO, 10, SAL*1.1, 20, SAL*1.2, SAL*1.3)$ : 该表达式的含义是，部门 10 的员工，工资增长 10%；部门 20 的员工，工资增长 20%；其他部门员工的工资增长 30%。

PowerCenter 中包含了大量的函数， $IIF$  和  $DECODE$  是其中两个常用的函数，此外还包括大量的字符串函数、日期函数、数值函数、加密函数等，以满足日常 ETL 过程中格式转换的需求。

(8) 再次单击“确定”按钮，进入如图 2-17 所示的界面。分别选中 Expression 组件中的 ENAME 和 OUT\_SAL，将它们与目标表中的 ENAME 和 SAL 连接。方法是：选中 ENAME，按住鼠标左键将其拖动到希望关联的目标表字段上即可。

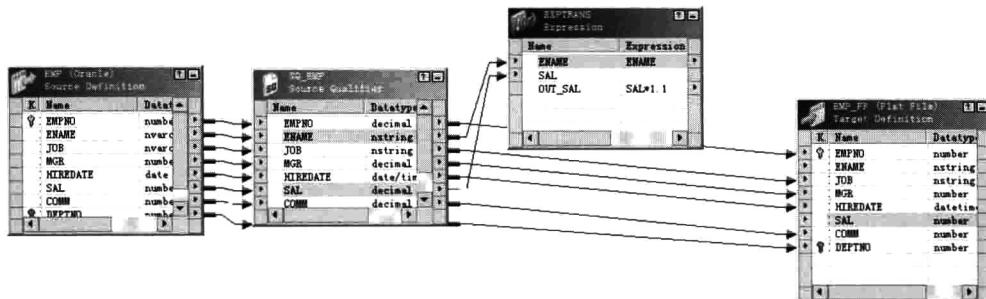


图 2-17

最终完成的 Mapping 如图 2-18 所示。

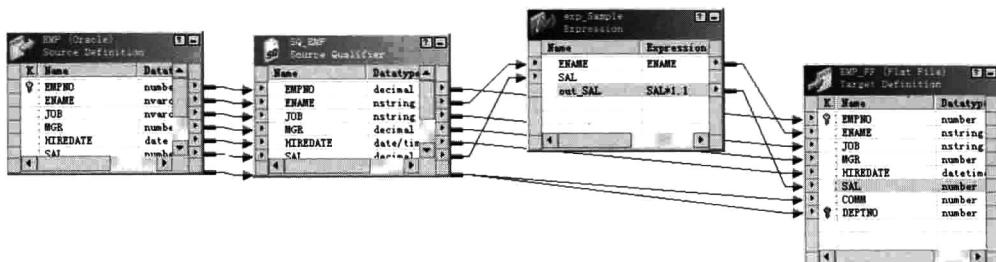


图 2-18

## Expression 中的变量端口 (Variable Port)

在上面的例子中着重介绍的端口是 I 和 O 类型的情况，这里再花一点时间介绍变量端口 (Variable Port)。变量端口主要用于存储在 Transformation 中多次使用的复杂计算的中间结果，使用 Variable Port 有利于提高程序的可读性，提升 Transformation 的性能。

例如，在一个 ETL 过程中，需要计算同一数据集合满足条件的员工的工资增长和年度奖金发放，它们的计算公式分别如下（见图 2-19）。

**工资增长：**  $IIF(((JOB\_STATUS='Full-time') \text{ AND } (OFFICE\_ID=1000)), SAL * 1.1, SAL * 1.05)$ 。

**年度奖金：**  $IIF(((JOB\_STATUS='Full-time') \text{ AND } (OFFICE\_ID=1000)), 10000, 5000)$ 。

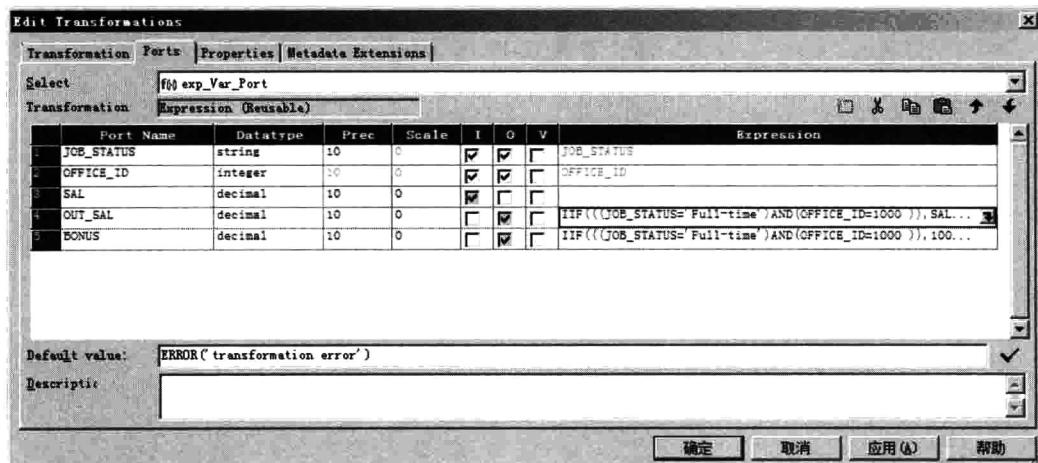


图 2-19

仔细观察如上两个表达式，可以注意到 $(JOB\_STATUS='Full-time') \text{ AND } (OFFICE\_ID=1000)$ 被计算了两次。

这时可以使用变量端口进行优化。可以定义变量 v\_Flag，它的计算公式是 $(JOB\_STATUS='Full-time') \text{ AND } (OFFICE\_ID=1000)$ ，优化后的结果如下。

**V\_Flag:**  $IIF((JOB\_STATUS='Full-time') \text{ AND } (OFFICE\_ID=1000), 1, 0)$ 。

**工资增长:**  $IIF(V\_Flag=1, SAL * 1.1, SAL * 1.05)$ 。

**年度奖金:**  $IIF(v\_Flag=1, 10000, 5000)$ 。

在 Expression 组件中，具体的展示如图 2-20 所示。

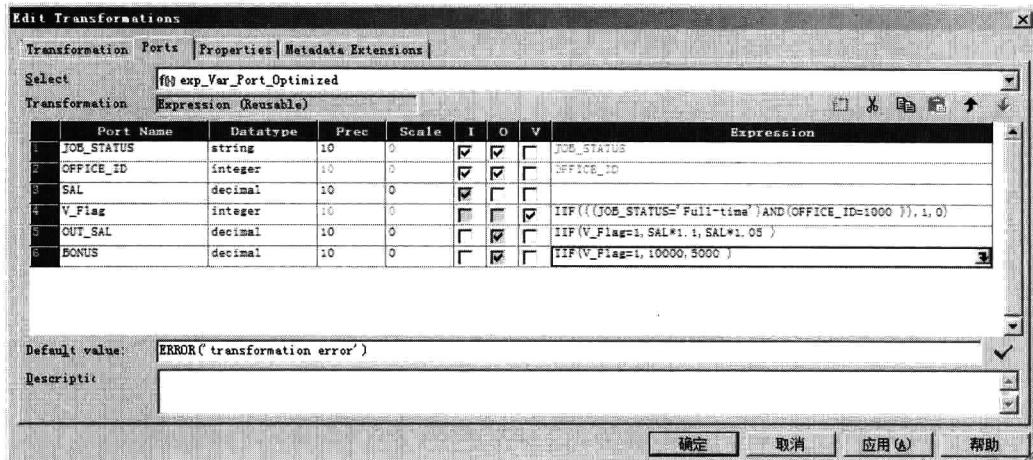


图 2-20

## 2.4 Filter

Filter 组件，顾名思义为过滤组件。在 Mapping 中增加一个 Filter 组件，满足条件的数据将通过这个组件，不满足条件的数据会被该组件过滤掉。在 PowerCenter 中它的图标是一个漏斗 。

**案例：**2014 年年底会只对工资小于 5000 元的员工涨工资，需要用 PowerCenter 获取这些员工的基本信息。它的数据源是包含全体员工的员工信息表。

通过 Hello World 和 Expression 的学习，我们相信现在实现这样的 Mapping 非常简单，只需要关注 Filter 组件如何设计即可。在 Mapping 的工作区完成一个如图 2-21 所示的 Mapping。双击 Filter 组件，进入 Filter 组件的编辑模式，然后选择 Properties Tab，如图 2-22 所示。



图 2-21

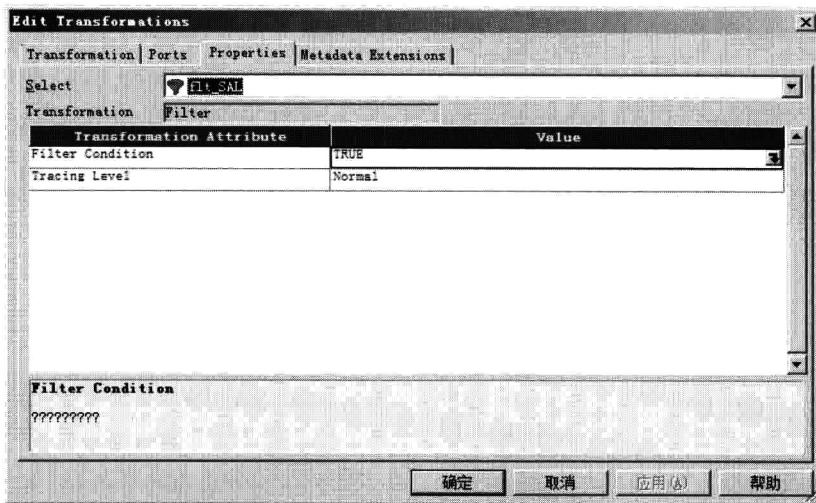


图 2-22

这时,需要做的是将 Filter Condition 的值 TRUE 修改为需要的过滤条件,如 SAL<5000。当 Filter Condition 为 TRUE 时,意味着所有的数据都将通过 Fliter 组件;当条件为 SAL<5000 时,意味着仅仅工资小于 5000 元的员工信息会通过这一组件。Filter 组件的最终实现结果如图 2-23 所示。

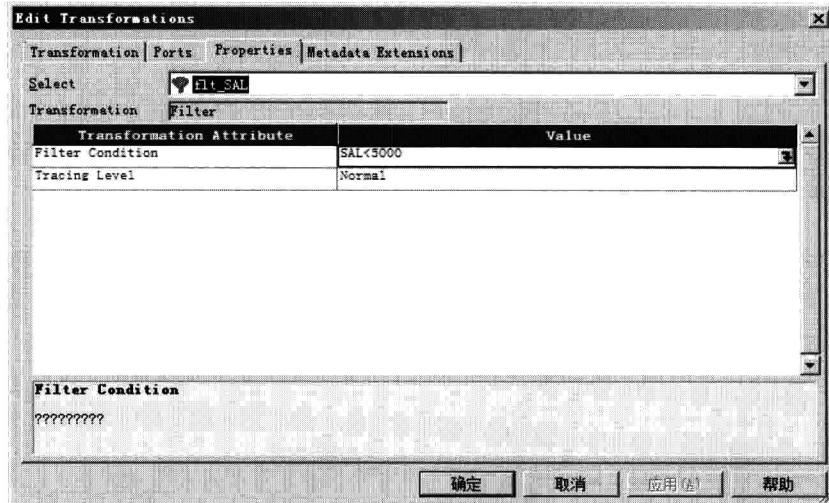


图 2-23

## 2.5 Source Qualifier

Source Qualifier 组件在所有的场景中会被用到，它主要用于连接数据源，将数据源的数据类型转换为 PowerCenter 的数据类型。Source Qualifier 组件针对不同的类型源系统会有很大的不同，如数据库的 Source Qualifier 和 PowerExchange CDC 的 Source Qualifier 组件差别很大。本节重点介绍两种最常用的情况，即当数据源分别是数据库和文本文件时，Source Qualifier 的作用和用法。

### 2.5.1 Source Qualifier 的作用

笔者曾经被很多人问过：当把数据源拖入 Mapping 之后，为什么会出现两个组件，而不是仅仅一个数据源？为什么要莫名增加一个组件？这个莫名增加的组件就叫作 Source Qualifier，如图 2-24 所示。

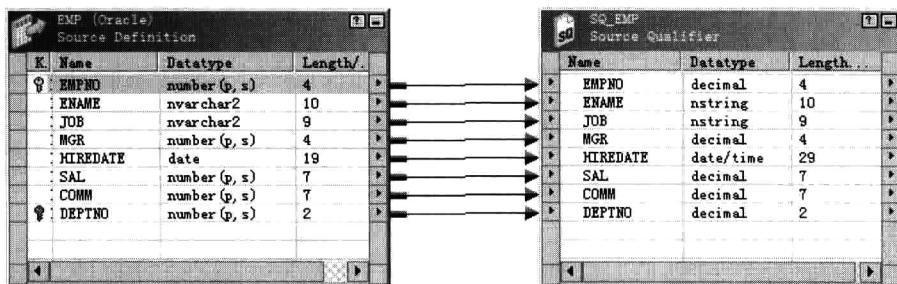


图 2-24

仔细观察这个新增的组件 (Source Qualifier)，会发现它与数据源的字段名称一模一样，但数据类型与源不同，同时又非常相似。为什么会是这样的？设想两个常见的 ETL 场景，就会理解为什么需要这样一个组件。

(1) 一个场景是，有两张表分别来自 Oracle 和 DB2 数据库，这两张表要进行关联 (Join) 计算，来自 Oracle 的关联字段的类型为 VARCHAR2，来自 DB2 的关联字段的类型为 VARCHAR。

为了实现这种关联，需要保证关联字段的数据类型是一致的。可能采取的方法包括：

将 Oracle 的数据类型转换为 DB2 的数据类型；或者将 DB2 的数据类型转换为 Oracle 的数据类型。但这两种方法都不是最佳方案，假如还有一个 SQL Server 怎么办？最佳的方案是有一种中间类型，不管 PowerCenter 读取哪种数据库，都自动进行转换以方便后续的互操作。因此 PowerCenter 提供了自己的数据类型，并通过 Source Qualifier 自动实现数据类型映射。

(2) 另一个场景是，假如数据源和数据目标是不同的数据库，当将源数据插入目标数据库时，也需要一个中间的媒介类型作支持。

从以上分析中可以注意到，Source Qualifier 和数据源有很大的关系，PowerCenter 支持各种各样的数据源，包括数据库、应用、文件、消息中间件等，也支持批量、实时的数据源，无法在这里一一列举，而是把它们放到专题中进行介绍。这里重点介绍一下数据库作为数据源时 Source Qualifier 的使用。

## 2.5.2 数据库数据源的 Source Qualifier

介绍数据库数据源的 Source Qualifier，首先从一个简单的 Mapping 开始，参考图 2-25。

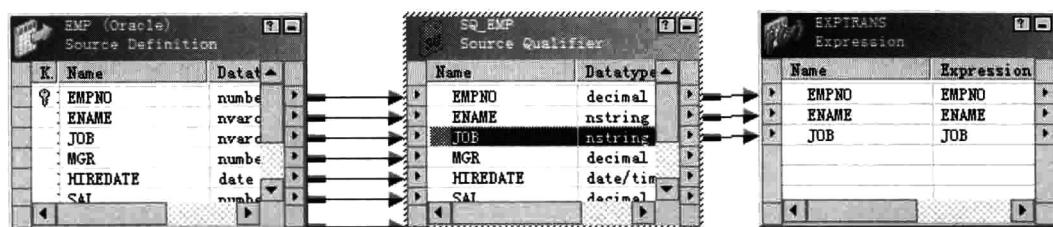


图 2-25

在图 2-25 中，已经将 3 个列 EMPNO、ENAME 和 JOB 从 Source Qualifier 连接到了 Expression 中。这时双击 Source Qualifier 组件，选择 Properties Tab，如图 2-26 所示。

关注 SQL Query 属性，单击 Value 处的箭头，即进入如图 2-27 所示的 SQL 编辑界面，当单击 Generate SQL 按钮后，PowerCenter 将自动生成如下 SQL 语句：

```
SELECT EMP.EMPNO, EMP.ENAME, EMP.JOB
FROM
EMP
```

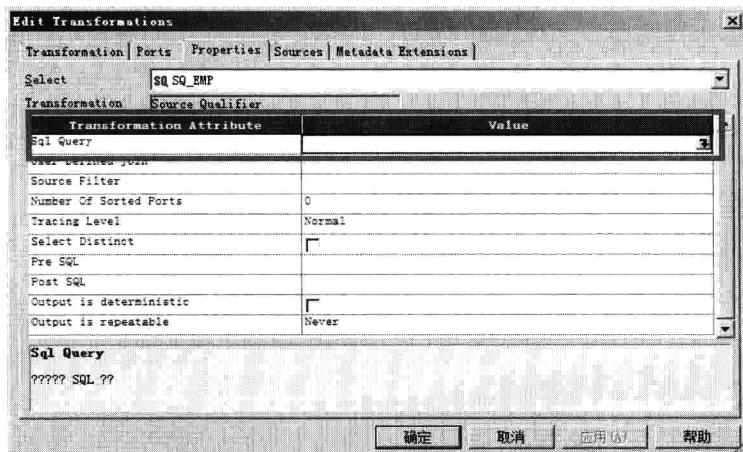


图 2-26

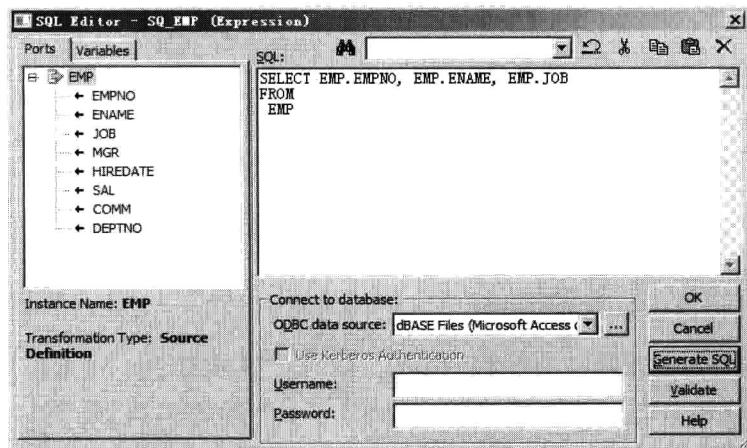


图 2-27

Select 后面的 3 个字段正是拖入到 Expression 中的 3 个字段，并且顺序是一样的。对初学者而言，在使用自定义 SQL 时这里经常出问题。主要问题就是没有将 Select 后面的字段与拖入后续组件的字段一一对应。关注一下 Source Qualifier 中的其他几个属性。

- ◎ Source Filter：如果在 Source Filter 中输入 EMP.SAL>5000，PowerCenter 将从数据库中读取工资大于 5000 元的员工的信息。

假如这时使用 SQL Query 中的 Generate SQL，则会生成如下 SQL 语句（这就是说在

Source Filter 中输入过滤条件与在 SQL Query 中修改 SQL 语句的效果是一样的):

```
SELECT EMP.EMPNO,EMP.ENAME,EMP.JOB  
FROM  
EMP  
WHERE  
EMP.SAL>5000
```

- ◎ Number of Sorted Port: 如果在 Number of Sorted Port 中输入 2, 会有什么作用呢?

同样用 SQL Query→Generate SQL, 生成的 SQL 语句如下:

```
SELECT EMP.EMPNO,EMP.ENAME,EMP.JOB  
FROM  
EMP  
WHERE EMP.SAL>5000  
ORDER BY EMP.EMPNO,EMP.ENAME
```

即按照 Select 中的前两个字段进行排序。

- ◎ Select Distinct: 同样, 启动这个选项后, 生成的 SQL 语句如下:

```
SELECT DISTINCT EMP.EMPNO,EMP.ENAME,EMP.JOB  
FROM  
EMP  
WHERE EMP.SAL>5000  
ORDER BY EMP.EMPNO,EMP.ENAME
```

现在大家应该已经理解这 3 个选项的功能了。它们其实是 SQL 图形化、选项化实现的一种形式。在数据库中执行的 SQL 本质上就是上面看到的 SQL 语句。

到这里, 相信很多人心里会有一个疑问: 在 Source Qualifier 中可以实现选择工资大于 5000 元的员工记录, 使用 Filter 组件也可以实现, 这两种方法有什么不同?

这个问题涉及 PowerCenter 的执行原理: PowerCenter 作为 ETL 工具, 其中包含的多数组件 (Transformation) 的功能绝大多数是在 PowerCenter 服务器上执行的。以 Filter 组件中 SAL>5000 为例, PowerCenter 会读取全部 EMP 表中的数据, Filter 组件会一条一条地检查这些数据, 只有 SAL>5000 的数据才会放行, 流入更下游的组件。假如 EMP 表有 100 万行数据, 其中工资大于 5000 元的有 10 万行, PowerCenter 会把数据库中的 100 万条全部读入 PowerCenter, 然后通过 Filter 组件进行过滤。

而在 Source Qualifier 中的 Source Filter 定义 SAL>5000，PowerCenter 向数据库提交的 SQL 为 select ... from EMP where SAL>5000，这种情况下仅仅 90 万条数据被读入 PowerCenter 中。

简单的解释就是 Filter 组件是由 PowerCenter 服务器执行的，而 Source Qualifier 中定义的 Source Filter 实际是转换为 SQL 语句由数据库服务器负责执行。虽然这两种方式可以实现相同的功能，但是其各有优缺点。如果希望对数据性能影响最小，最佳的解决方案是尽量使用 Filter，但是这时数据库会有更大的 I/O。而如果使用 Source Qualifier 中的 Source Filter，PowerCenter 的压力会比较小。同时，如果数据库在 SAL 上，即过滤条件上有索引，可以达到最理想的效果。但是如果在 SAL 上没有索引，这种情况可能引起数据库的全表扫描，效果则更差，会极大地影响数据库的性能。因此，采用哪种方式，需要斟酌具体的场景及需求。

### 2.5.3 Source Qualifier 自定义 SQL

自定义 SQL 是很多工程师喜欢的方式，前面可以看到在 SQL Query 中生成了一些预定的 SQL，如：

```
SELECT DISTINCT EMPNO,ENAME,JOB  
FROM  
EMP  
WHERE EMP.SAL>5000
```

这个 SQL 是可以修改的，可以在 WHERE 的后面增加 EMP.SAL>5000 AND EMP.SAL<10000；也可以将 WHERE 后的语句修改成一个子查询。只需要满足以下条件，PowerCenter 都是可以执行的。

- 这个修改后的 SQL 语句满足源数据库的语法要求。如果源数据库是 Oracle 数据库，则需要使用 Oracle 的语法；如果源数据库是 DB2 数据库，则需满足 DB2 的语法。
- SELECT 后的字段必须与流入下游组件的字段名字和顺序一致。名字和顺序保持一致也可以保证程序的可读性。如果对 SELECT 后的字段使用函数，建议使用字段别名，如 SELECT DISTINCT EMPNO,ENAME,substr(JOB,1,1) AS JOB...。假如使用这个 SQL，则只能将 3 个字段与下游的 Transformation 相连。

还有一个来自开发人员的常见问题：是否可以在 SQL Query 中使用关联，如 SELECT

EMPNO,ENAME,substr(JOB,1,1) AS JOB FROM EMP a, DEPT b where a.DEPTNO= b.DEPTNO?

答案是：可以。再追问一下：如果使用前面的 SQL 语句，是否需要将 EMP 和 DEPT 表都拖入 Mapping 中呢？答案是：不需要。但是建议尽量将这两张表都加入到 Mapping 中，这样可以提高程序的可读性。这就是后面要介绍的复杂关联。

#### 2.5.4 Source Qualifier 复杂关联

既然可以在 Source Qualifier.SQL Query 中使用自定义 SQL，很容易理解其实在 SQL Query 中也可以完成任意的数据库表关联，甚至是任意的 SQL 语句。

例如，人力资源部需要获取部门名称包含 Accounting 且工资大于 5000 元的员工的信息。通过将业务需求转换为技术语言，用一个 SQL 语句表示如下：

```
SELECT EMP.EMPNO,EMP.ENAME,EMP.JOB  
FROM  
EMP,DEPT  
WHERE  
DEPT.DEPTNO=EMP.DEPTNO AND  
EMP.SAL >5000 AND  
DNAME LIKE '%Accounting%'
```

上面 SQL 语句描述的逻辑通过 PowerCenter 来实现，实现的 Mapping 如图 2-28 所示。

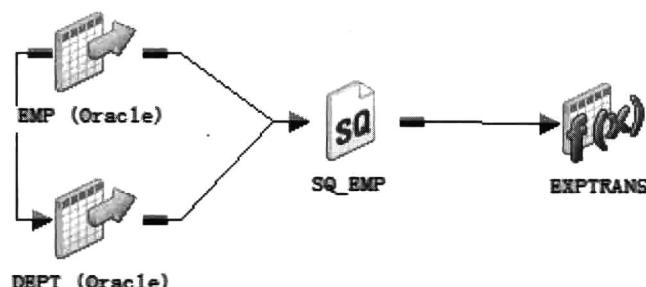


图 2-28

这时候需要做的仅仅是在 Source Qualifier 的 SQL Query 中粘贴如上的 SQL 语句，如图 2-29 所示。

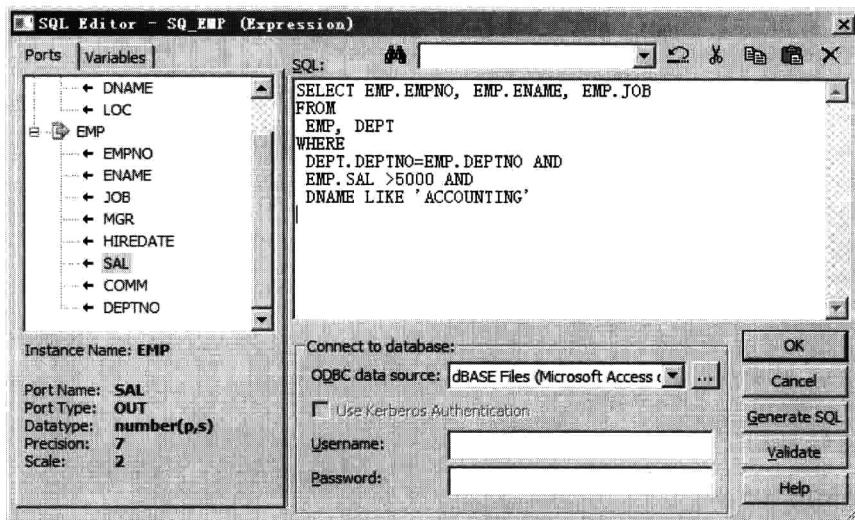


图 2-29

### → 注释

在图 2-29 中，在 Source Qualifier 后增加了一个 Expression Transformation，这是为什么？这个 Expression Transformation 的作用是模拟任意一个 Source Qualifier 下游的组件。开发人员需要做的是将 EMPNO、ENAME、JOB 拖入下游组件，否则，如果单击 Validate 按钮，将提示一个错误信息，在运行时也将报错。

## 2.6 Sorter

Sorter 在 PowerCenter 中使用  图标。

Sorter 组件并不复杂，本来是计划放到后面来写，不过这个组件经常会与其他组件搭配使用，因此将其提前进行介绍。Sorter 顾名思义就是排序。

既然是排序，就需要选择按照哪个特性（字段）进行排序，是升序还是降序，如图 2-30 所示，这是 Sorter 组件最常用的两个选项。

另外，在 Properties Tab 中还有几个功能点需要关注，如图 2-31 所示。

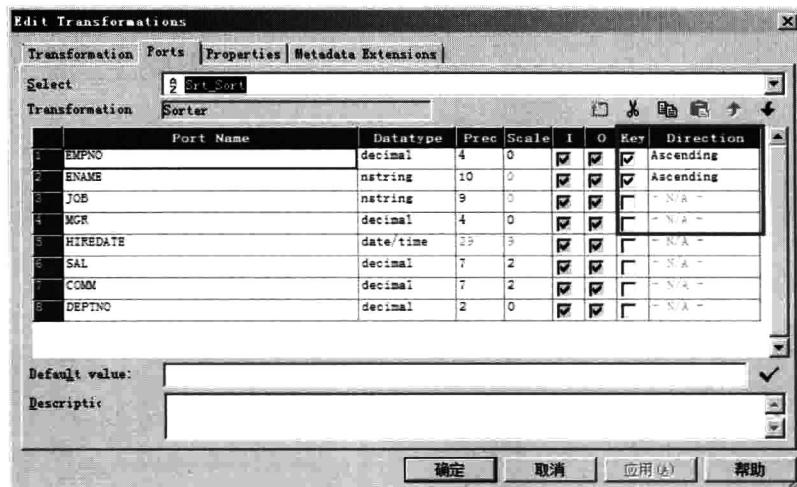


图 2-30

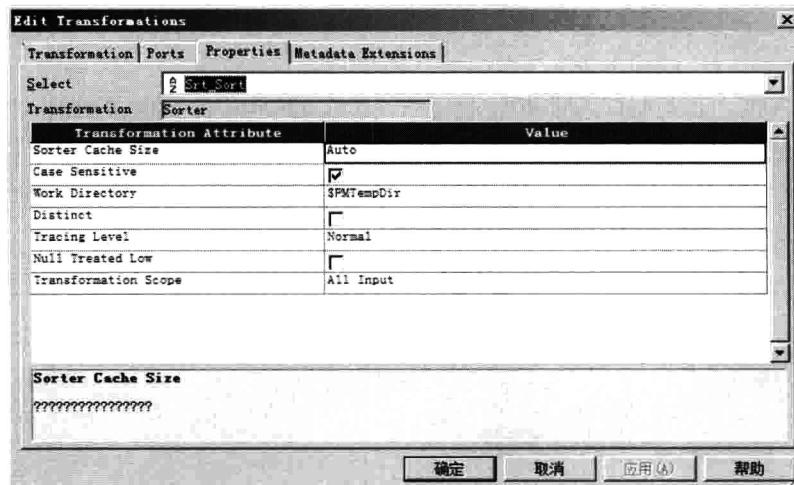


图 2-31

- ① Case Sensitive: 排序时大小写是否敏感。
- ② Distinct: 获取唯一值, 这就是说, 可以通过 Sorter 组件实现类似 SQL 中 Distinct 的功能。
- ③ Null Treated Low: 是否将 NULL 值认为是小数。

## 2.7 Joiner

Joiner 在 PowerCenter 中使用  图标。

Joiner 组件是另一个非常常用的组件，主要是实现表之间的关联。在 Source Qualifier 组件中已经展示过一种关联的实现方式，同样的需求通过 Joiner 组件也可以实现。

实现后的 Mapping 如图 2-32 所示。Expression 组件仅仅是一个 Dummy 组件，可以不考虑。

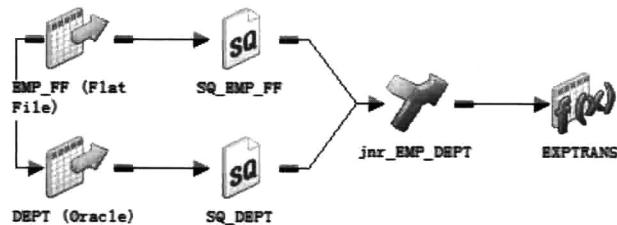


图 2-32

观察如上 Mapping 中的 Joiner 组件是如何定义的，它可以实现哪些功能？作为关联，首先要定义关联条件，双击 Joiner 组件，选择 Condition Tab，按照人力资源部的需求（参考 SQ 组件），定义关联条件 Master.DEPTNO=Detail.DEPTNO1，如图 2-33 所示。



图 2-33

这样就实现了一个基本关联。在这个对话框中，除了 DEPTNO 字段，还出现了两个新的关键字：Master 和 Detail。哪张表应该是 Master，哪张表应该是 Detail？返回 Ports Tab 中，可以看到 Master 和 Detail 的定义，如图 2-34 所示。

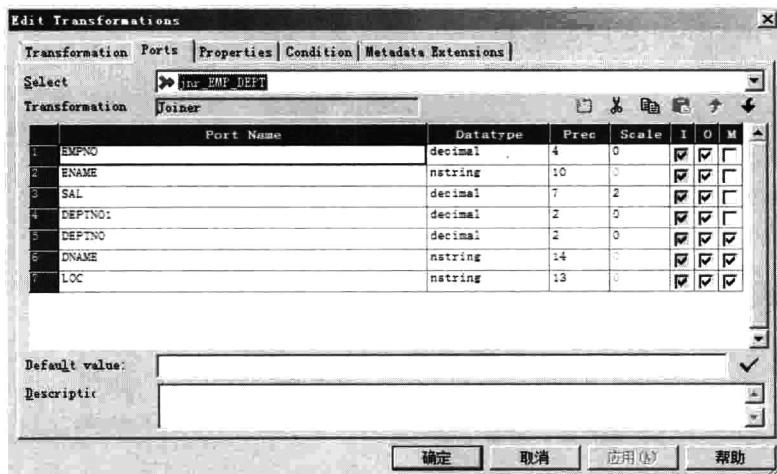


图 2-34

端口类型在 I、O 之外出现了一个 M，这个 M 即 Master 的意思，选中 M 的表即为 Master 表，否则是 Detail 表。那么哪张表应该作为 Master 表呢？理解这个问题，还需要简单介绍一下 Joiner 组件的执行过程。

Joiner 组件的执行过程是这样的：它会首先将 Master 表中用到的所有数据读入 PowerCenter 服务器上，并将这些数据（主要是关联字段和向下游继续传递的字段）创建为一棵索引树；然后才开始读 Detail 表的数据，将读到的 Detail 的数据与 Master 的索引树关联，从而实现 Joiner 的功能。这时相信你已经可以理解，从功能上看，谁作为 Master 是无所谓的，但是从性能的角度来看，最好是小表作为 Master 表。

### 2.7.1 关联类型

既然是关联，不能回避的话题就是关联类型，SQL 的世界存在 4 种典型的关联方法，它们是 Inner Join、Left Join、Right Join 和 Full Join。不用担心，在 PowerCenter 中这些功能都能非常方便地实现。打开 Joiner 组件的 Properties Tab，如图 2-35 所示。

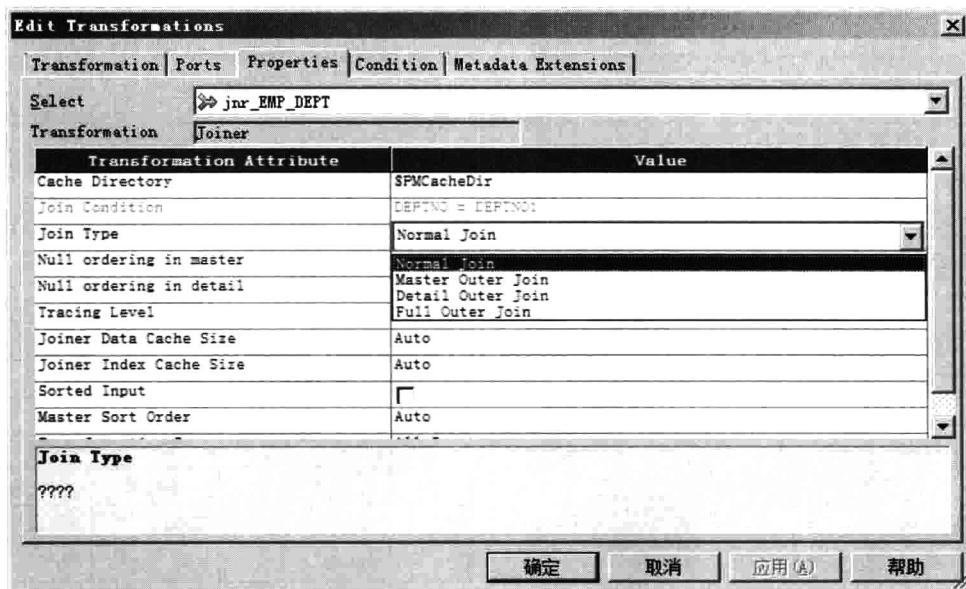


图 2-35

Join Type 在 PowerCenter 中实现这些功能,有 4 个选项: Normal Join、Master Outer Join、Detail Outer Join、Full Outer Join, 分别对应 Inner Join、Left Join、Right Join、Full Join。PowerCenter 用两个圆圈简单地解释了这几种 Join 类型。假设有两个数据集合,如表 2-1 (Master 表) 和表 2-2 (Detail 表) 所示,其关联结果如表 2-3 所示。

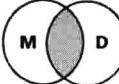
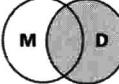
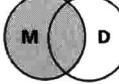
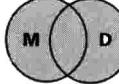
表 2-1

ID	公司名	总收入 (亿)
1	IBM	1000
2	Oracle	800
3	Informatica	30

表 2-2

ID	公司名	员工数 (万)
2	Oracle	20
3	Informatica	3
4	Microsoft	10

表 2-3

关联类型	图示结果	结果数据																				
Inner Join 结果集：中间交叉部分		<table border="1"> <thead> <tr> <th>ID</th><th>公司</th><th>总收入</th><th>员工数</th></tr> </thead> <tbody> <tr> <td>2</td><td>Oracle</td><td>800</td><td>20</td></tr> <tr> <td>3</td><td>Informatica</td><td>30</td><td>3</td></tr> </tbody> </table>	ID	公司	总收入	员工数	2	Oracle	800	20	3	Informatica	30	3								
ID	公司	总收入	员工数																			
2	Oracle	800	20																			
3	Informatica	30	3																			
Master Outer Join 结果集：Detail 部分		<table border="1"> <thead> <tr> <th>ID</th><th>公司</th><th>总收入</th><th>员工数</th></tr> </thead> <tbody> <tr> <td>2</td><td>Oracle</td><td>800</td><td>20</td></tr> <tr> <td>3</td><td>Informatica</td><td>30</td><td>3</td></tr> <tr> <td>4</td><td>Microsoft</td><td>NULL</td><td>10</td></tr> </tbody> </table>	ID	公司	总收入	员工数	2	Oracle	800	20	3	Informatica	30	3	4	Microsoft	NULL	10				
ID	公司	总收入	员工数																			
2	Oracle	800	20																			
3	Informatica	30	3																			
4	Microsoft	NULL	10																			
Detail Outer Join 结果集：Master 部分		<table border="1"> <thead> <tr> <th>ID</th><th>公司</th><th>总收入</th><th>员工数</th></tr> </thead> <tbody> <tr> <td>1</td><td>IBM</td><td>1000</td><td>NULL</td></tr> <tr> <td>2</td><td>Oracle</td><td>800</td><td>20</td></tr> <tr> <td>3</td><td>Informatica</td><td>30</td><td>3</td></tr> </tbody> </table>	ID	公司	总收入	员工数	1	IBM	1000	NULL	2	Oracle	800	20	3	Informatica	30	3				
ID	公司	总收入	员工数																			
1	IBM	1000	NULL																			
2	Oracle	800	20																			
3	Informatica	30	3																			
Full Outer Join 结果集：全部		<table border="1"> <thead> <tr> <th>ID</th><th>公司</th><th>总收入</th><th>员工数</th></tr> </thead> <tbody> <tr> <td>1</td><td>IBM</td><td>1000</td><td>NULL</td></tr> <tr> <td>2</td><td>Oracle</td><td>800</td><td>20</td></tr> <tr> <td>3</td><td>Informatica</td><td>30</td><td>3</td></tr> <tr> <td>4</td><td>Microsoft</td><td>NULL</td><td>10</td></tr> </tbody> </table>	ID	公司	总收入	员工数	1	IBM	1000	NULL	2	Oracle	800	20	3	Informatica	30	3	4	Microsoft	NULL	10
ID	公司	总收入	员工数																			
1	IBM	1000	NULL																			
2	Oracle	800	20																			
3	Informatica	30	3																			
4	Microsoft	NULL	10																			

从表 2-3 中可以清楚地了解在 PowerCenter 中 4 种典型的关联方式的含义及其计算结果，也可以了解 PowerCenter 完全支持所有的常见关联方式。

## 2.7.2 Sorted Joiner

Sorted Joiner 是一种提升 PowerCenter 关联性能的方法，它的要求是流入到 Joiner 组件中的数据是有序的，这样可以在多数情况下提升 Joiner 性能。

在 PowerCenter 中，Sorted Joiner 和非 Sorted Joiner 使用不同的算法。Sorted Joiner 使用 Sorted-Merge 算法。如果有机会运行一个包含 Sorted Joiner 的 Mapping，在 Session Log 中就会找到如下日志：Message:The Joiner transformation [JNRTRANS] is configured with the join algorithm:Sorted-Merge。

而非 Sorted Joiner 使用 Nested-Loop 算法，在 Session Log 中的信息类似“INFO: (3540 | MAPPING) : (IS | PWC\_IS) : node01\_SDHOST1 : TT\_11098 : The Joiner transformation [JNRTRANS] is configured with the join algorithm: Nested-Loop”。

常见的文章中都曾提到 Sorted Joiner 与非 Sorted Joiner 相比性能是有所提升的，笔者在实践中发现大多数情况下确实如此。但是，Sorter 也会消耗一定的资源。例如在使用 Partition 的情况下，有时候也会造成性能下降。因此，在实践中需要通过测试来确定使用 Sorted Joiner 对性能的影响。

一个 Sorted Joiner Mapping 的样例实现如图 2-36 所示。

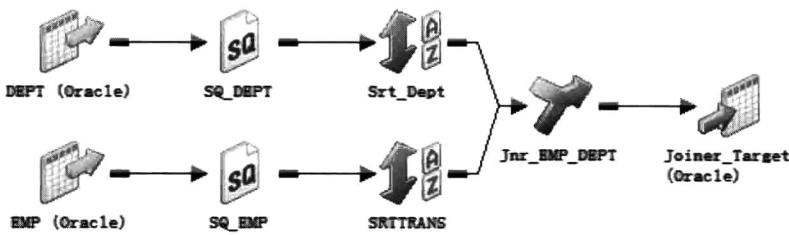


图 2-36

- ◎ Sorter 组件对 Joiner 组件中关联的字段排序。假如关联字段是 DEPTNO，那么就需要对 DEPTNO 进行排序。
- ◎ 要求 Master 表和 Detail 表输入到 Sorted Joiner 组件中的数据都是有序的。

另外一种获得排序数据的方法是，通过在 Source Qualifier 中使用自定义 SQL 语句 Order By 将需要的列进行排序。

### 2.7.3 Joiner 的独特作用

在谈 Joiner 的过程中经常会被问到的问题是：为什么要有个单独的 Joiner 组件，在 Source Qualifier 中实现关联不就可以了？什么情况下使用 Joiner 比较合适？

Joiner 适用的常见场景如下。

- ◎ 异构数据库表的关联，比如 Oracle 表和 DB2 表进行关联。
- ◎ 数据库表与文件的关联，比如 Oracle 表与文本文件进行关联。
- ◎ 文件与文件的关联。

- ◎ 同构表关联，但是两张表不在同一个数据库中，比如两张表都是 Oracle 表，但是分属不同的 Oracle 数据库。
- ◎ 同一个数据库中的表，但是在表上没有合适的索引或者源数据库 DBA 不允许在源库中进行 ETL 计算。

Joiner 也有一个限制，即一个 Joiner 只能同时对两张表进行关联。如果需要对 3 张表进行关联的话，则需要两个 Joiner 组件。以此类推，如果需要关联的表很多，如  $n$  张表，则需要  $n-1$  个 Joiner。

#### 2.7.4 自关联（Self-Join）

自关联指的是同一张表在计算过程中，需要进行自我关联才能获得期望结果的计算过程。即将同一张表分别作为 Joiner 的 Master 和 Detail。为了说明这个问题及其解决方案，还是以一个现实中的场景为例来帮助理解。

**案例：**以员工信息（EMP 表）为例，财务部要求数据仓库项目组提供一组关于员工信息和员工工资的数据，要求提供的数据中包括员工 ID、员工姓名、所属部门、工资信息及工资占本部门总工资的百分比。

这个 Mapping 有个非常简单的实现方法，如图 2-37 所示。

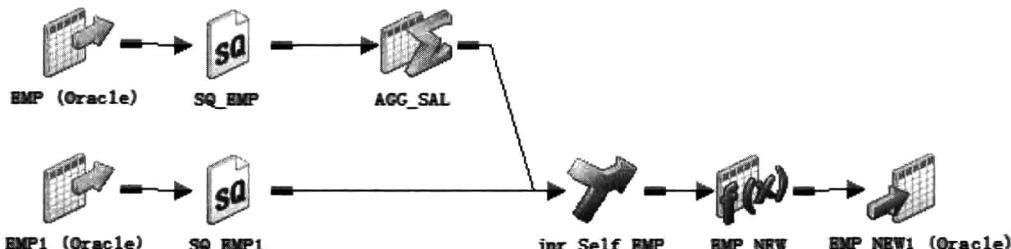


图 2-37

以上的 Mapping 中，上面的 Pipeline 通过 Agg\_SAL 组件进行部门工资求和计算，然后通过 Joiner 组件将上下两个 Pipeline 的信息组合到一起，最后通过一个 Expression 组件进行结果计算。仔细观察这个 Mapping，就会发现其有一个比较大的问题，那就是数据源 EMP 表被读了两次。当 EMP 表数据比较多时，会造成较大的性能困扰。消除两次读表的问题的解决方案就是通过“自关联（Self-Join）”。

最终使用 Self-Join 实现的 Mapping 如图 2-38 所示。

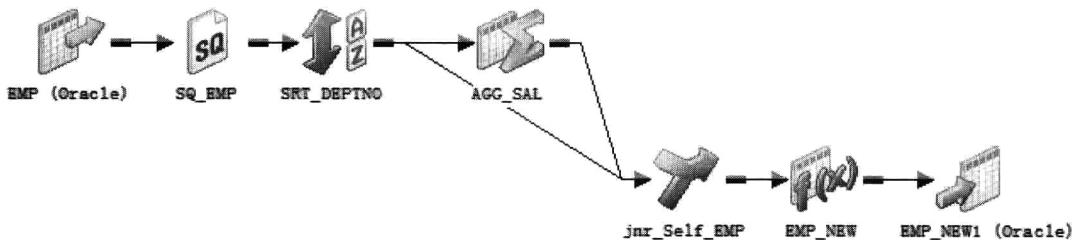


图 2-38

这个 Mapping 与第一次完成的 Mapping 相比，最大的变化是数据源 EMP 表只读了一次，这个改变对 Mapping 的执行效率将有比较大的提升。我们知道 Aggregation 是 Active 组件，一般无法将其上游组件（Sorter）与下游组件（Joiner）直接相连。现在的这种实现是因为在 Joiner 中使用了 Sorted Input 选项，因此与前面 Mapping 的不同之处就是在 Source Qualifier 后增加了一个 Sorter 组件。这个 Mapping 的逻辑是：首先，通过 AGG\_SAL 对部门总收入进行求和；其次，通过 Joiner 组件，以 DEPTNO 为关联条件，将个人信息与部门工资信息关联到一起；最后，通过 Expression 组件计算“SAL 除以 SUM\_SAL”获得个人工资占部门总工资的百分比。这样就完成了一个 Self-Join 的 Mapping，提升了 Mapping 的执行效率。

上面以一个案例的实现介绍了 Self-Join，如果用比较形式化的语言介绍如何实现 Self-Join，那么一个 Self-Join Mapping 必须满足以下条件。

- ◎ 在从 Source Qualifier 到 Joiner 的两个分支上，至少一个分支必须在 Source Qualifier 和 Joiner 之间增加一个其他组件。
- ◎ Joiner Key 必须是经过排序的。
- ◎ Joiner 需要使用 Sorted Input。

## 2.8 Lookup

Lookup 在 PowerCenter 中使用图标。

Lookup 组件在 PowerCenter 的组件中属于选项最多、功能最繁杂的一个。用一个例子

说明它的功能：在员工的工资表（EMP）中只有工资信息，没有部门信息，如果财务部需要了解每个部门的平均工资，就需要使用工资表中的部门号从部门表（DEPT）中查询（Lookup）员工所属的部门名，然后汇总、平均获得期望的结果。这就是 Lookup 组件的最基本功能。

现在已经有两张表 EMP 和 DEPT，需要建立一个 Mapping 完成财务部所需的功能。

(1) 在工具栏中单击  图标，将其放入 Mapping Designer 的工作区的空白处，PowerCenter 会弹出如图 2-39 所示的对话框。

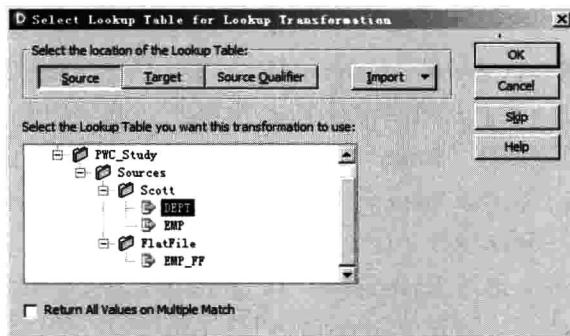


图 2-39

#### → 注释

这时可以从 Source、Target、Source Qualifier 中选择要查询的表。为什么有这么多选项呢？在 Hello World 部分曾经提到过，在 Mapping 中使用的对象和物理的数据库没关系，在 Mapping 中只关心表的结构，因此不管这个结构是来自 Source、Target 或者其他地方都没关系，在这里仅仅关心表的结构。

“Return All Values on Multiple Match”，这是一个在 PowerCenter 9 中才有的新功能，它可以支持 Lookup 返回多个值，如在 EMP 表中员工张三的 DEPTNO 是 10，而在被查询的表 DEPT 中 10 分别对应 Accounting 和 R&D 部门，Lookup 将返回两条数据。

(2) 在本例中，单击 Source 按钮，并在 Source 中选择 DEPT 表。也可以从 Target 中选择，或者如果 DEPT 尚未导入，也可以单击 Import 按钮导入一个新的 DEPT 表的结构。选择完后，单击 OK 按钮。将 EMP Source Qualifier 中的 DEPTNO 列拖入 Lookup 组件，并拖入一个 Aggregator 组件（Aggregator 组件后续有详细介绍，这里不再详述），如图 2-40 所示。

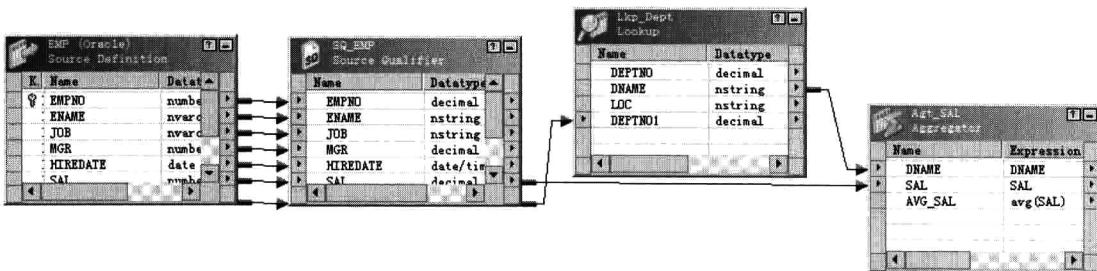


图 2-40

(3) 双击 Lookup 组件，进入 Condition Tab，定义 Lookup 条件 DEPTNO=DEPTNO1，其中 DEPTNO 来自被查询的表 DEPT，DEPTNO1 来自 EMP 表，如图 2-41 所示。

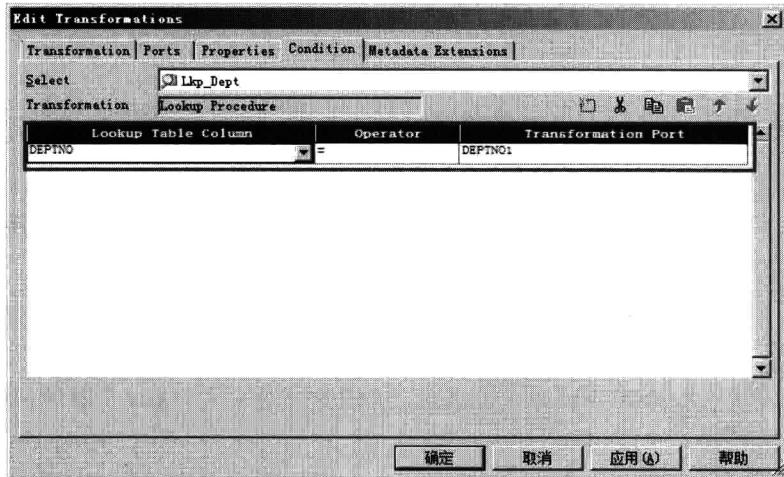


图 2-41

至此，可以说读者已经基本会用 Lookup 组件了。但要真正理解 Lookup，还有很多知识要学习。Lookup 的 Operator 可以使用=、>、<等，但在 Joiner 组件中只能使用=。

### 2.8.1 Lookup Caching enabled

Lookup Caching enabled 是一种提升 Lookup 组件性能的选项，也是 Lookup 组件的默认选项，如图 2-42 所示。



图 2-42

当选择 Lookup caching enabled 之后，Lookup 的执行过程如下：

- ◎ 首先被查询的表的全部数据（如果有 SQL Override，则是 SQL Override 返回的结果集）被读入 PowerCenter，PowerCenter 将 Lookup Condition 条件涉及的 key 和使用的其他字段建成一个索引。
- ◎ 然后 Lookup 表会一条一条地读取数据，这些数据会与上一步建立的索引进行匹配。
- ◎ 如果发现了要查找的 key(关联条件，如 DEPTNO)，则与 key 关联的值(如 DNAME) 向下游传递。
- ◎ 如果没有找到要查找的 key，则与 key 关联的值会以 NULL 值（如 DNAME 会被赋予 NULL）向下游传递。

#### → 注释

当没有选择选项 Return All Values on Multiple Match 时，向下游组件传递的数据量永远与 Lookup 表一致。以前面的样例为例，向 Lookup 下游组件传递的行数等于 EMP 表的行数。

当被 Lookup 的数据集为文本文件时，Lookup caching enabled 强制使用，不可以取消。

当被 Lookup 的数据集为数据库时，Lookup caching enabled 可以选择使用或者不使用。当选择使用时，会先建索引再查询，这样做的好处是性能得到了很大的提升；坏处是如果 Lookup 过程中被 Lookup 的数据库的数据发生了变化，这个变化不会体现在 Lookup 结果中。

当选择不使用 Cache 时，每一条从 EMP 来的数据都会向数据库提交一条查询语句，这样做的好处是数据库的变化会及时地体现到查询结果中；缺点是性能非常差，根据表的大小，性能可能仅是使用 Lookup caching enabled 的 1/10 甚至 1/100。

## 2.8.2 非连接的 Lookup

Unconnected Lookup 也是一种 Lookup 方式。Unconnected Lookup 的特点是，Lookup 组件不会与任何一个源、目标或者其他组件直接相连。

**案例：**财务部有个新需求，财务人员需要计算每个部门工资大于 5000 元的人员的平均工资，如果工资小于 5000 元，这些人将会被归到一个新的虚拟部门 Low Cost。已知工资大于 5000 元的员工只占公司总员工数的 20%。

Lookup 是比较消耗 ETL 服务器资源的组件之一，因此很多时候开发人员都会试图优化这个组件的性能，如减少 Lookup 的次数。基于这个考虑，PowerCenter 提供了 Unconnected Lookup，确保只有必需的数据才执行 Lookup 操作。

下面用 Unconnected Lookup 来实现财务部的新需求，并确保性能最好。最终的实现结果如图 2-43 所示。

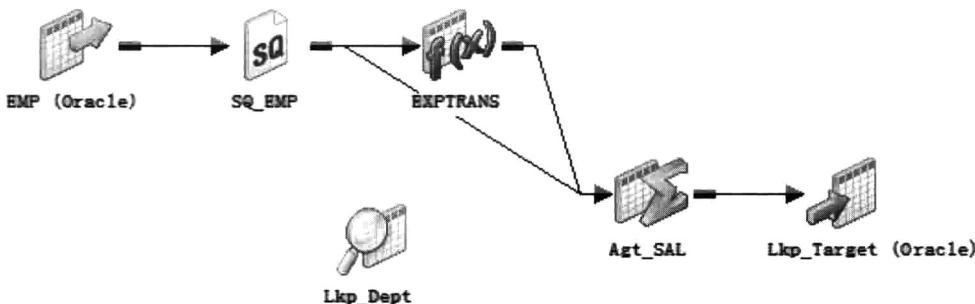


图 2-43

在图 2-43 中，Lookup 组件游离在 Mapping 的其他组件之外，这就是所谓的 Unconnected Lookup。那么上面的 Mapping 是怎么实现的？在前面的组件介绍中已经完成了大量的 Mapping 练习，后面会省略掉与 Unconnected Lookup 无关部分的操作步骤。

(1) 从工具栏中拖入一个 Lookup 组件，将其重命名为 Lkp\_Dept。这里的名字很重要，后面会继续使用这个名字。

(2)在 Lkp\_Dept 中做如下设置:字段 DEPTNO、DNAME 作为输出,添加字段 DEPTNO1 作为输入,并确保关联字段的数据类型是一致的、兼容的,如图 2-44 和图 2-45 所示。

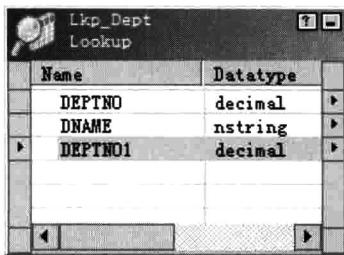


图 2-44

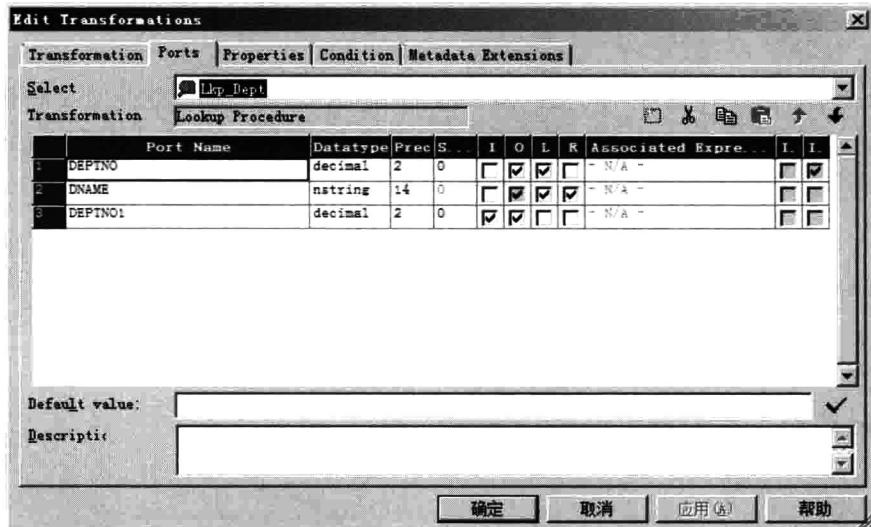


图 2-45

设置 DNAME 为返回值,即在图 2-45 中勾选 R 复选框。在本例中,如果将 Unconnected Lookup 看作一个函数的话, R 类型的字段就是这个函数的返回值。

(3) 在 Lkp\_Dept Condition Tab 中添加关联条件 DEPTNO=DEPTNO1。

(4)添加一个 Expression 组件,增加 SAL 和 DEPTNO 作为输入/输出,增加 OUT\_DNAME 作为输出,如图 2-46 所示。

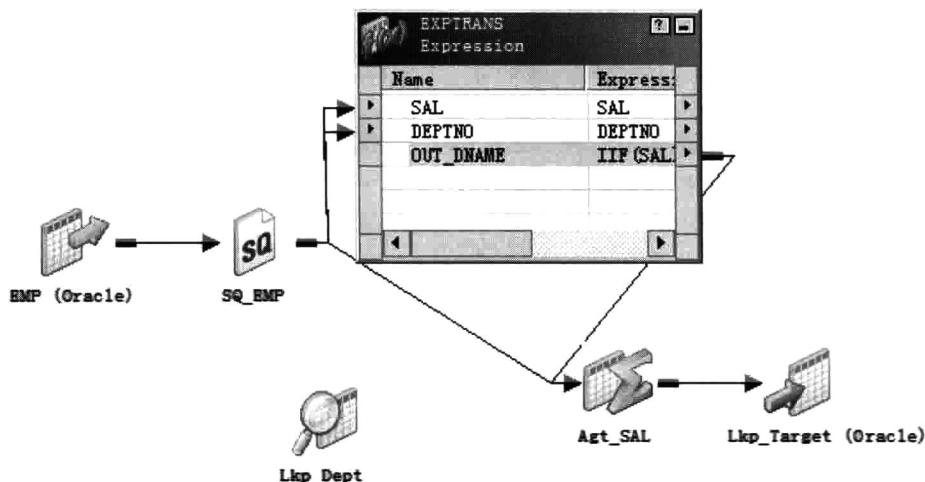


图 2-46

(5) 编辑 OUT\_DNAME 的表达式为 “IIF(SAL>5000,;LKP.lkp\_dept(DEPTNO), 'Low Cost')”。这个表达式的含义是：如果 SAL 大于 5000，则通过 LKP.lkp\_dept 查询其所在的部门；否则设为 Low Cost。

这样就通过 Unconnected Lookup 组件用最少的资源消耗实现了财务部的需求。同时也相信大家已经基本了解了 Unconnected Lookup 的基本用法。

### 2.8.3 Lookup SQL Override

以这一节开始的 Mapping 为例，双击 Lookup 组件，选择 Properties Tab，如图 2-47 所示。

选择 Lookup Sql Override 的 Generate SQL 后，观察到的 SQL 语句如下：

```
SELECT DNAME, LOC, DEPTNO FROM DEPT
```

但是提交到数据库中的 SQL 却增加了 Order by DEPTNO，如下：

```
SELECT DNAME, LOC, DEPTNO FROM DEPT Order by DEPTNO,DNAME
```

实际在数据库中执行的 SQL 语句可在 Session Log 中看到。

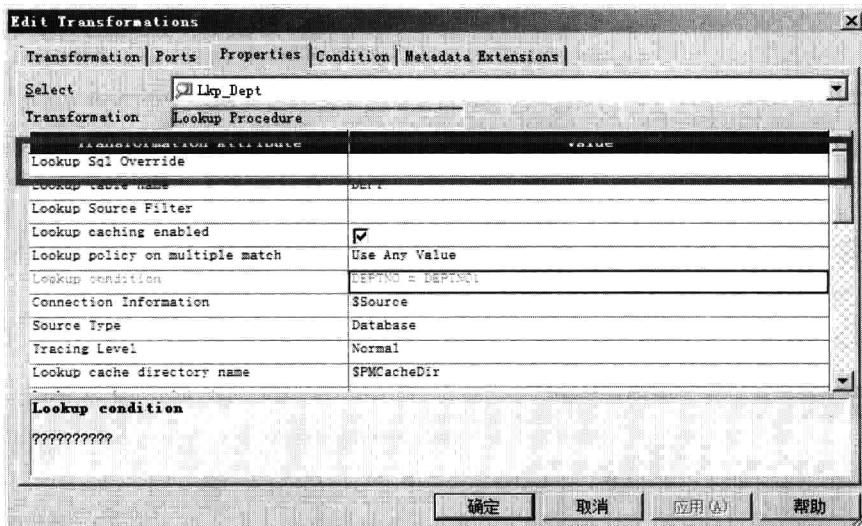


图 2-47

增加 Order by 子句的目的是保证 PowerCenter 从数据库获得的数据是有序的，加速创建 Lookup 索引的过程。但有时候 DBA 会担心消耗源数据库的资源，希望去掉 Order by 语句。这就是一种特殊的 Lookup SQL Override。如何去掉默认的 Order by 呢？做法很简单：在 Lookup Sql Override 中，选择 Generate SQL，并在默认生成的 SQL 后增加 “--”。修改后的 SQL 如下：

```
SELECT DNAME, LOC, DEPTNO FROM DEPT --
```

这样提交到数据库的 SQL 语句就变成了“SELECT DNAME,LOC,DEPTNO FROM DEPT-- **Order by DEPTNO,DNAME**”。其实 Order by 子句并不是被删除了，而是使用了 SQL 语法中的注释方法，使 Order by 没有生效。

还有一种情况经常被用到：开发人员希望修改默认的 SQL 语句。与 Source Qualifier 一样，PowerCenter 支持这样的做法，但需要满足以下几个条件：

- 修改后的语句满足源数据库的语法要求。
- 修改后 SELECT 后面的字段与对应的 Lookup 字段一致，如果对 SELECT 字段使用了函数，需要使用字段别名，如：“SELECT DNAME, substr(LOC,1,1) AS LOC, DEPTNO FROM DEPT”。

### 2.8.4 共享 Lookup Cache

在某些情况下，一张表的 Lookup 会多次出现在同一个 Mapping 中，或者同一张表的 Lookup 会出现在多个 Mapping 中。这种情况下，开发、管理人员希望这个 Lookup Cache 不要重复创建多次，一次创建的 Cache 可以被后续的 Lookup 所共享，从而提升整体的性能。

因此在 PowerCenter 中提供了多种不同的 Lookup Cache 方便 Cache 的共享，包括 Named Lookup Cache 和 Persistent Lookup Cache。

Lookup Cache 涉及的范围很广，这里先不详述，详细内容请参考性能调优部分。如需获得更详细的信息，还可以参考 PowerCenter 手册。

### 2.8.5 Dynamic Lookup

**案例：**ABC 公司有一个在线客户注册系统，每天有大量的客户通过这个在线注册系统进行注册，数据仓库项目组也需要每天将这些客户信息同步到数据仓库中。在同步数据的过程中，数据仓库组发现在线系统发送给数据仓库的数据除了新客户注册，还有大量的客户变更信息，甚至有当天注册当天变更的情况，也有一天内多次变更的情形。为了方便理解，项目组提供了一组样例数据。

历史数据（见表 2-4）：

表 2-4

ID	Name	Telephone	Zip	Address
1	Du	123	110000	
2	Wang	123	110000	
3	Li	123	110000	

日增量数据（见表 2-5）：

表 2-5

ID	Name	Telephone	Zip	Address
4	Du	123	110000	
3	Wang	123	110003	
5	Li	123	110000	
4	Yang	123	110004	

如果按照前面几个小节中的 Lookup 逻辑，需要从目标表查询新的数据是否存在于目标表中，如果不存在，则执行插入操作，否则执行 Update。但是对 ID=4 的记录存在一个问题：Lookup Cache 是 Session 运行时一次性创建的，其中没有对 ID=4 的记录，因此新增的两条数据都会执行插入操作，这样的结果会是错误的。解决这个问题有多种方法，如下所述。

- ◎ 不使用 Lookup caching enabled，这个方法效率很低。
- ◎ 使用 Session 属性中的 Treat Source rows as = Update 与目标属性 Update Else Insert 组合。这是个不错的方法，在 Update Strategy 中会介绍类似的功能。
- ◎ 使用 Dynamic Lookup Cache。

前面的两种方法将在其他章节进行介绍，本节的内容主要是帮助大家理解 Dynamic Lookup 的作用及使用方法。Dynamic Lookup 的主要作用是解决当 Lookup 表为目标表时，如何同步 Lookup Cache 和目标表的问题。

使用上面的案例创建一个 Mapping，这个 Mapping 如图 2-48 所示。

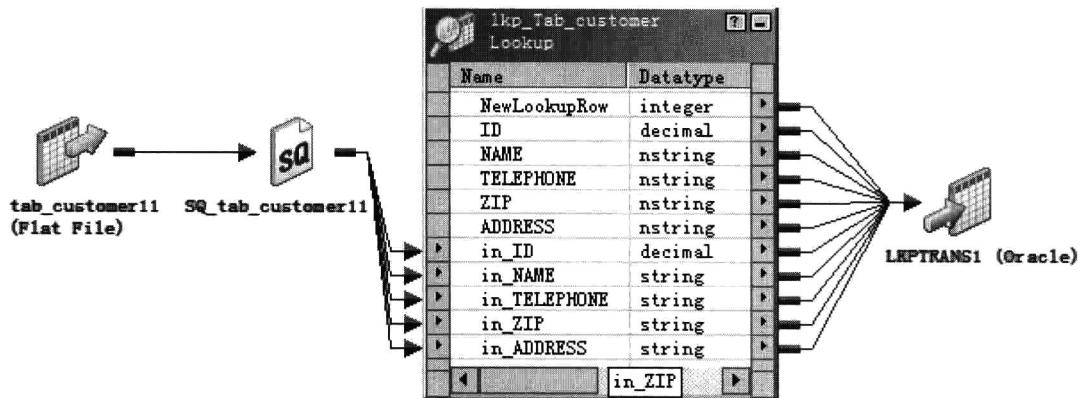


图 2-48

在这个 Mapping 中，Lookup 表和目标表是同一张表，将源的所有字段拖入了 Lookup 中，并重命名为 in\_开头的字段。这时关注一下 Lookup 的设置。在 Lookup 的 Properties Tab 中使用了两个新的属性：Dynamic Lookup Cache 和 Insert Else Update，如图 2-49 所示。

Insert Else Update 是最经常使用的选项，因为 PowerCenter 默认的数据全部为 Insert，这是 Session 的 Treat Source as Rows 的一个值，所以这里的 Insert Else Update 与 Treat Source as Rows=Insert 配合使用，而 Update Else Insert 与 Treat Source as Rows=Update 配合使用。

如果需要同时选择 Update Else Insert 和 Insert Else Update，那就需要在 Dynamic Lookup 前增加一个 Update Strategy 组件。

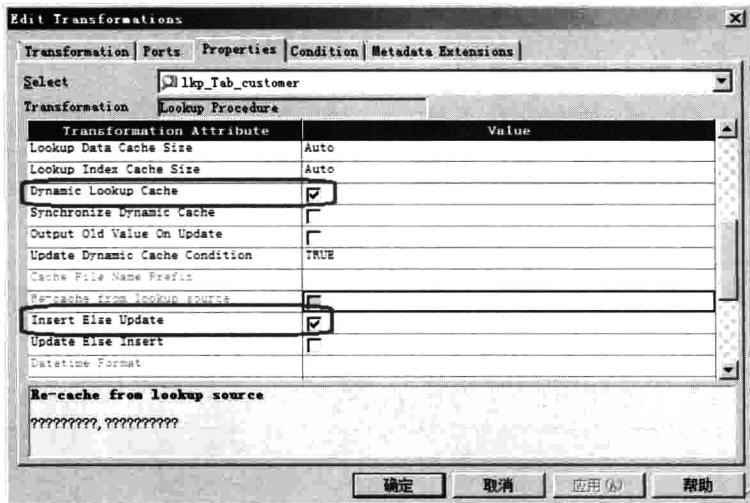


图 2-49

选择 Dynamic Lookup Cache 会导致 Ports Tab 与常见的 Mapping 不同，在这个 Mapping 中，Mapping Tab 如图 2-50 所示。

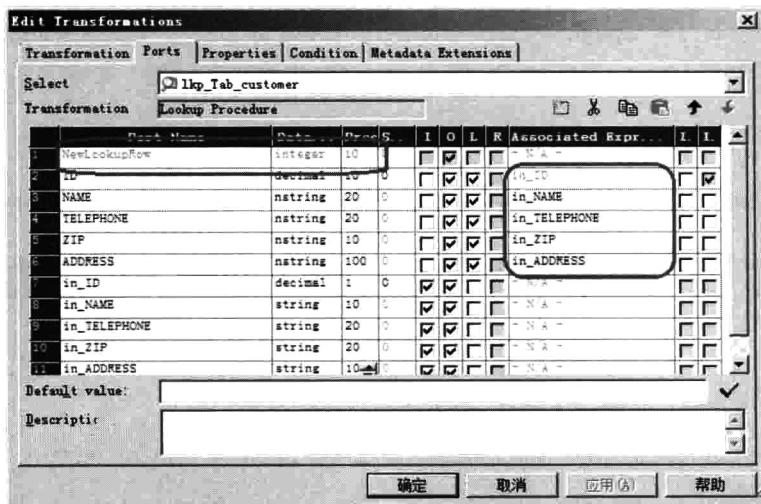


图 2-50

在 Associated Expression 中将 Lookup 字段与输入字段一一关联。当 Lookup 字段为整型 Integer、Bigint、Smallint 时，在选择项中还会增加一个 Sequence\_ID 选项，Integratoin Service 将为选择 Sequence\_ID 的字段生成一个序列号。

NewLookupRow 的含义如表 2-6 所示。

表 2-6

NewLookupRow	含    义
0	Integration Service 不对 Cache 执行 Insert 或者 Update
1	Integration Service 对 Cache 执行 Insert
2	Integration Service 对 Cache 执行 Update

按照如上的设置，运行该 Mapping 的话，得到的结果如表 2-7 所示。

表 2-7

NewLookupRow	ID	Name	Tele	Zip	ID	Name	Tele	Zip
1	4	Du	123	110000	4	Du	123	110000
2	3	Wang	123	110003	3	Wang	123	110003
2	4	Li	123	110000	4	Li	123	110000
1	5	Yang	123	110004	5	Yang	123	110004

关注 ID=4 的情况：对第一条 ID=4，NewLookupRow=1，即 Insert；而第二条 NewLookupRow=2，即 Update。后续可以使用 Router/Filter 与 Update Stratety 完善这个 Mapping，也可以在 Update Stratety 中使用 decode() 函数解决问题，实现的最终 Mapping 如图 2-51 所示。



图 2-51

在 Update Stratety 中使用了如下表达式：

```
decode (NewLookupRow, 1, DD_INSERT, 2, DD_UPDATE, DD_REJECT)
```

此表达式根据不同的 NewLookupRow 返回值，决定对数据库的操作是插入、更新还是抛弃，并与更新 Dynamic Lookup Cache 方法保持一致。

在 Dynamic Lookup 的 Ports Tab 中还有两个属性也经常被用到，如图 2-52 标记的部分。

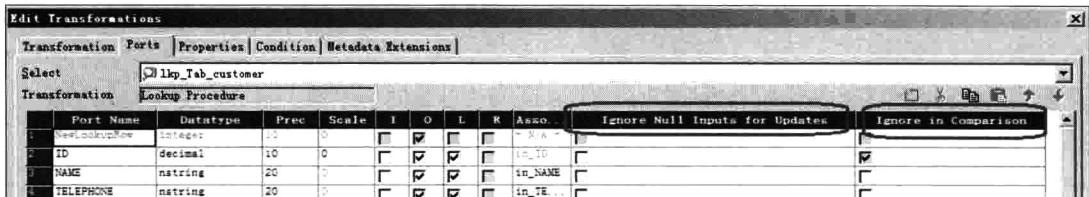


图 2-52

- ◎ Ignore Null Inputs for Updates：如果新输入的客户 Name 为 NULL，但在目标表中 Name 为非 NULL，这时业务部门不希望数据被更新为 NULL，则可以选择此选项。
- ◎ Ignore in Comparison：当客户不希望跟踪某些字段的变化时，如数据仓库项目组不想跟踪 Comments 字段的变化，则可以选择此选项。

## 2.8.6 Lookup、Source Qualifier 和 Joiner 的对比

Lookup、Joiner 与 Source Qualifier 看起来非常相似，但又有一些差异，因此有必要对它们的功能做一个详尽的比较。

### 1. 功能

Source Qualifier 因为可以执行任意的 SQL 语句，因此功能最为强大，依赖开发人员的 SQL 能力。Joiner 可以实现 4 种关联类型：Inner Join、Left Join、Right Join 和 Full Join，而 Lookup 不能。正因为 Lookup 相对简单，因此性能可能更好。

Joiner 中仅能使用等于 (=) 比较方式，而在 Lookup 中可以支持更多的比较方式，如>、<等。

Lookup Cache 有可能实现共享，而 Joiner 不行。

### 2. Active/Passive

Source Qualifier 和 Joiner 是 Active 组件，而 Lookup 可以是 Active (Multi-Output) 或者 Passive 组件。

### 3. 执行引擎

Lookup 和 Joiner 的执行引擎是 PowerCenter，而 Source Qualifier 的执行引擎是数据库。

## 2.9 Stored Procedure

Stored Procedure 在 PowerCenter 中使用的图标是 。

Stored Procedure 组件不是指可以在 PowerCenter 中写存储过程，而是指 PowerCenter 可以调用数据库存储过程。这在日常的开发过程中非常常见，开发人员由于某些原因已经开发了大量的 Stored Procedure，或者某些场景在 PowerCenter 中实现更加复杂而需要用存储过程来实现，这时就需要将 PowerCenter 与存储过程进行集成，Stored Procedure 组件就是一个不错的选择。

**案例：**人力资源部需要给员工涨工资，IT 部开发人员已经创建了一个存储过程，这个存储过程的名字是 SP\_ADD\_SALARY。存储过程如下：

```
create or replace procedure SP_ADD_SALARY(x in number, y out number) as
begin
    y:=x+100;
end SP_ADD_SALARY;
```

其中 x 是输入参数，y 是输出参数。

按照项目的要求，需要使用 PowerCenter 调用此存储过程实现工资的调整。

### 2.9.1 Connected Stored Procedure

对如何创建 Mapping 不再进行详细叙述，直接进入 Stored Procedure 部分，假设 Mapping 已经存在。实现财务部要求的设计过程如下。

(1) 在 Designer 中选择 Stored Procedure，拖入 Mapping Designer 工作区，可看到如图 2-53 所示的对话框，选择恰当的 ODBC，并输入数据库用户名/密码，单击“连接”按钮，可以看到数据库中已经创建的 Stored Procedure。

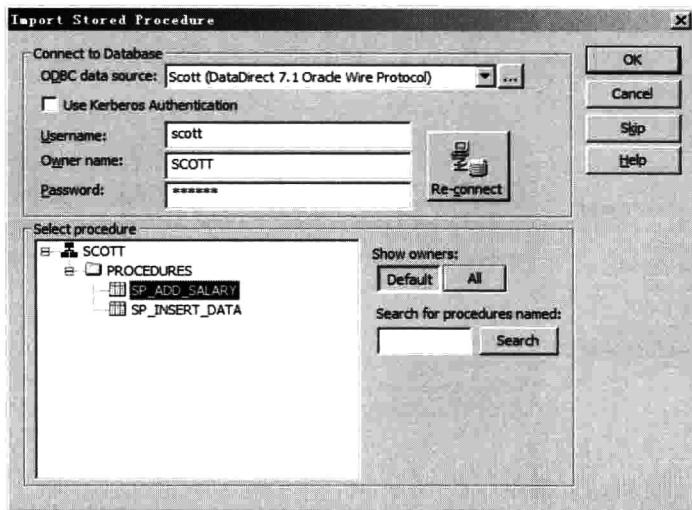


图 2-53

(2) 选择确认。将 SQ\_EMP 中的 SAL 与 X 连接，将存储过程的 Y 与目标表 EMP1 的 SAL 相连，Mapping 即完成，如图 2-54 所示。

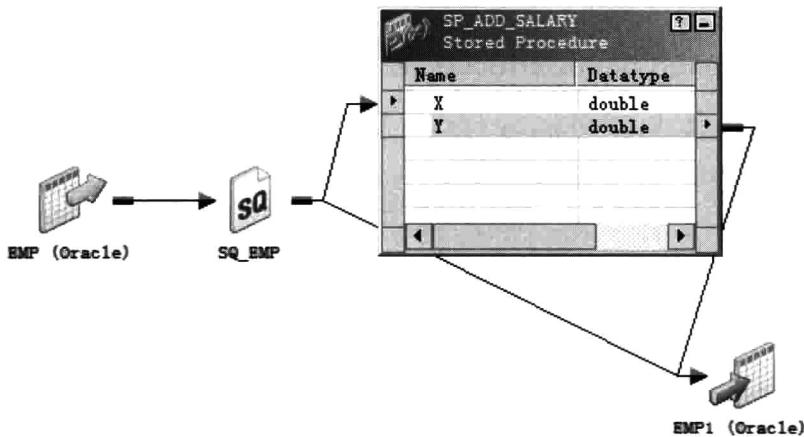


图 2-54

补充一下，使用存储过程时，在 Session 配置中除了给数据源和数据目标提供连接，还需要给 Stored Procedure 提供数据库连接，这个数据库连接指向的是存放执行存储过程的数据库，如图 2-55 所示。

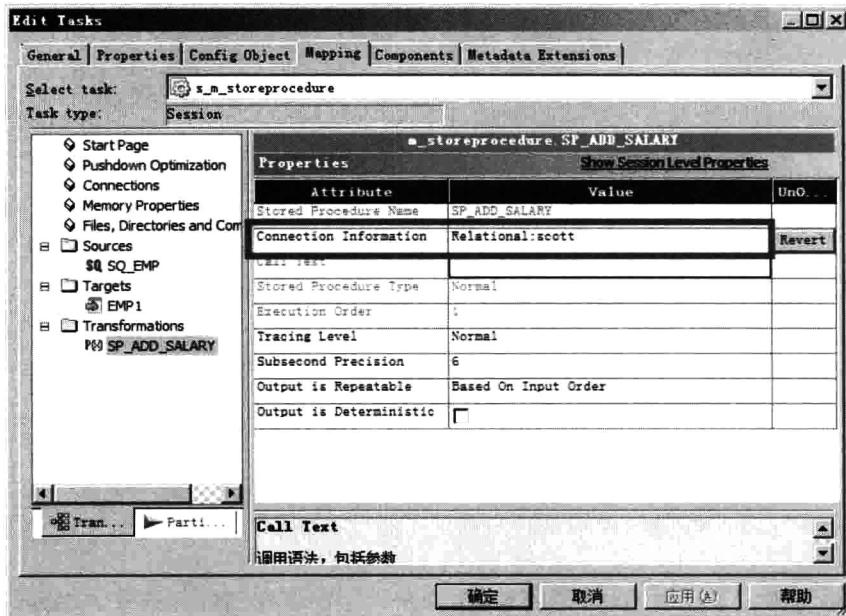


图 2-55

## 2.9.2 Unconnected Stored Procedure

Unconnected Stored Procedure 与 Unconnected Lookup 相似，调用存储过程是非常耗费资源的过程，尤其是每条数据都要调用存储过程的话。这种情况下 Unconnected Stored Procedure 就显得非常有必要了，它能满足随需调用的需求。同样以一个开发人员已经开发好的存储过程为例。

与上一个存储过程不同，这个存储过程为一个 Oracle Function，如下：

```
create or replace function SP_ADD_SALARY(x in number)
return number
as
y number(7,2);
begin
y:=x+100;
return (y);
end SP_ADD_SALARY;
```

(1) 在 Mapping Designer 中选择存储过程图标，将其加入到 Mapping Designer 中，输入用户名/密码，单击“连接”按钮，如图 2-56 所示。

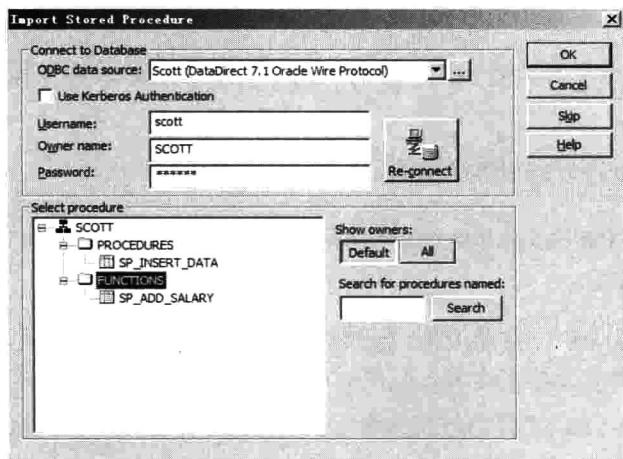


图 2-56

(2) 选择需要的函数，如 SP\_ADD\_SALARY，并单击 OK 按钮，就已经创建好的 Mapping 中增加了这个函数，如图 2-57 所示。

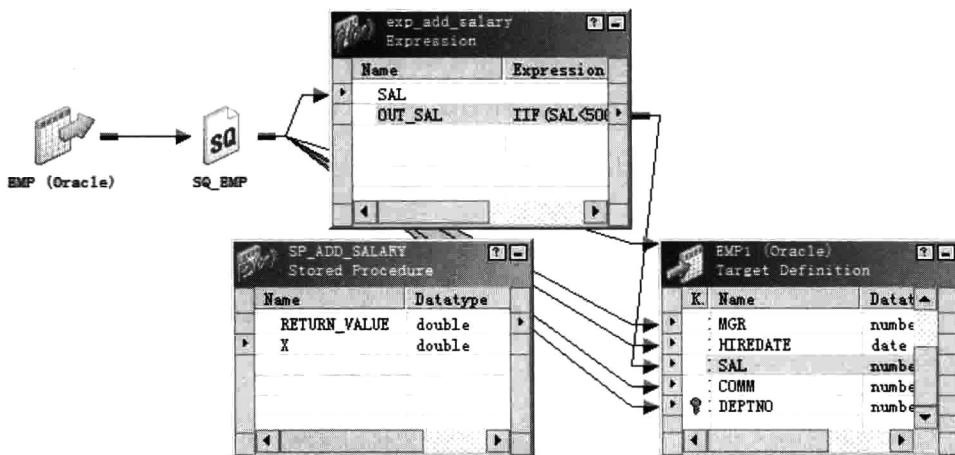


图 2-57

观察如上的 Stored Procedure 组件，X 是其输入端口，RETURN\_VALUE 是其输出端口。在 Oracle Function 中 X 是其输入，而 RETURN\_VALUE 是该函数的返回值。

现在的重点是如何在 Expression 中调用存储过程。选择 OUT\_SAL，编辑其表达式“ IIF(SAL<5000,;SP.SP\_ADD\_SALARY(SAL),SAL) ”，其中 SP 是一个关键字，SP\_ADD\_SALARY 是数据库中的存储过程名。

与 Connected Stored Procedure 一样，在 Session 中也需要给函数提供数据库连接。

### 2.9.3 Pre- or Post-Session Stored Procedure

Pre- or Post-Session Stored Procedure 也是一种特殊的 Unconnected Stored Procedure，其主要用在如下几个方面。

- **Source Pre-Load:** 存储过程或者 SQL 语句运行在 Session 从源读数据之前。主要用于校验表是否存在或者在临时表中执行一些操作，如关联等。
- **Source Post-Load:** 存储过程或者 SQL 语句运行在 Session 从源读取数据之后。主要用于清除临时表的数据。
- **Target Pre-Load:** 存储过程或者 SQL 语句运行在 Session 向目标表写数据之前。主要用于校验目标表或者确定目标系统的空间等。
- **Target Post-Load:** 存储过程或者 SQL 语句运行在 Session 向目标表完成写之后。主要用于重建数据库索引等。

以两个虚拟的无意义的存储过程为例，演示一下这样的 Mapping 的开发过程。在数据库中创建两个存储过程，如 SP\_ADD\_NEWROW 和 SP\_ADD\_NEWROW2。

创建一个 Mapping 如图 2-58 所示，这个 Mapping 已经导入了这两个存储过程。

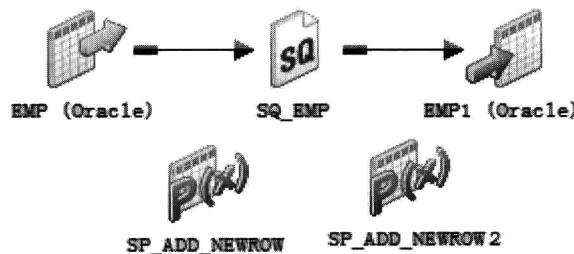


图 2-58

双击存储过程，打开存储过程的 Properties Tab，Call Text 分别为 SP\_ADD\_NEWROW() 和 SP\_ADD\_NEWROW2()。设置 Stored Procedure Type 为 Source Pre Load，如图 2-59 所示。

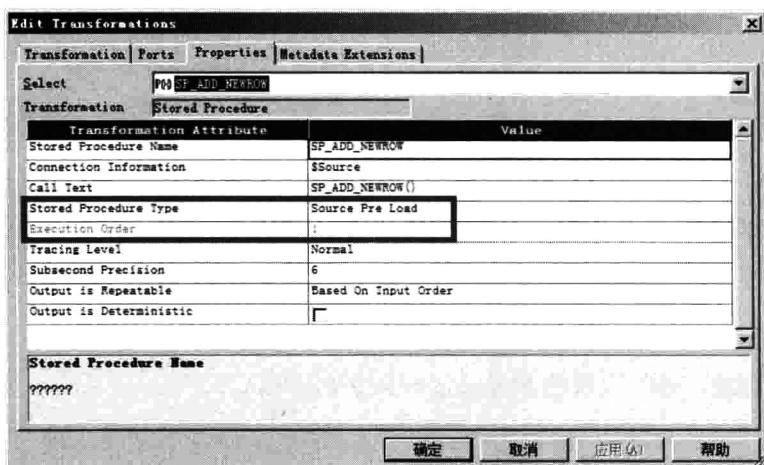


图 2-59

但是 Execution Order 是不可编辑的，这时两个存储过程的值均为 1。如果这样的话，保存后的 Mapping 将是一个无效的 Mapping，因为 Mapping 不知道应该先执行哪个存储过程。这时，需要一个附加步骤。

选择菜单 Mapping→Stored Procedure Plan 命令，这时可以看到如图 2-60 所示的对话框，在 Source Pre Read 下可以看到 Mapping 中已经存在的存储过程，使用上、下箭头调整它们的顺序，就会修改 Execution Order 的值，也就是指定了存储过程的执行顺序。

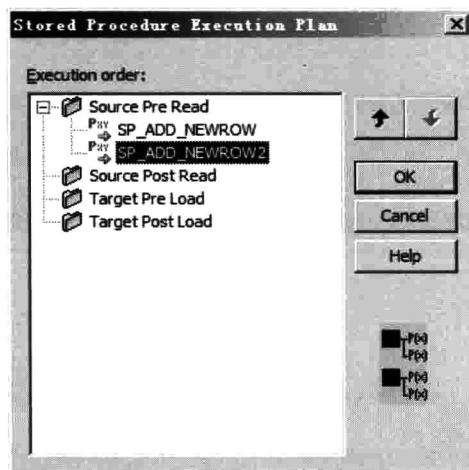


图 2-60

## 2.10 Union

Union 组件与 SQL 语句的 Union 功能非常相似，它用于将两张或者多张表联合到一起。例如，一个企业的员工信息一部分存放在数据库的 EMP 表中，一部分存放在文本文件中，数据仓库项目组希望获取完整的员工信息记录，这时候就会用到 Union 组件。

Union 组件在 PowerCenter 中的图标为 。

完成这个功能，最终实现的 Mapping 如图 2-61 所示。

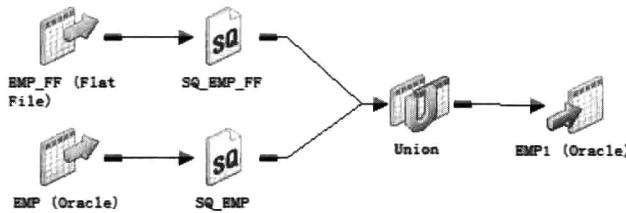


图 2-61

下面详细介绍如上 Mapping 的实现步骤。

首先向 Mapping 拖入 Union 组件，并在 Union 组件中建立两个组 EMP\_Table 和 EMP\_FlatFile，每个组各代表一种数据来源，步骤是：双击 Union 组件，选择 Groups Tab，如图 2-62 所示。



图 2-62

假如这时看到 Union 组件如图 2-63 所示，EMP\_Table 组在上面，就从 SQ\_EMP 中将所有字段拖入 Union 组件的空白处，然后将 SQ\_EMP\_FF 拖入 EMP\_FlatFile 对应组中（注：这个顺序是非必需的，仅仅是为了减少工作量）。



图 2-63

最后完成的 Union 组件如图 2-64 所示，为了达到清晰的目的，仅仅使用了 3 个字段。

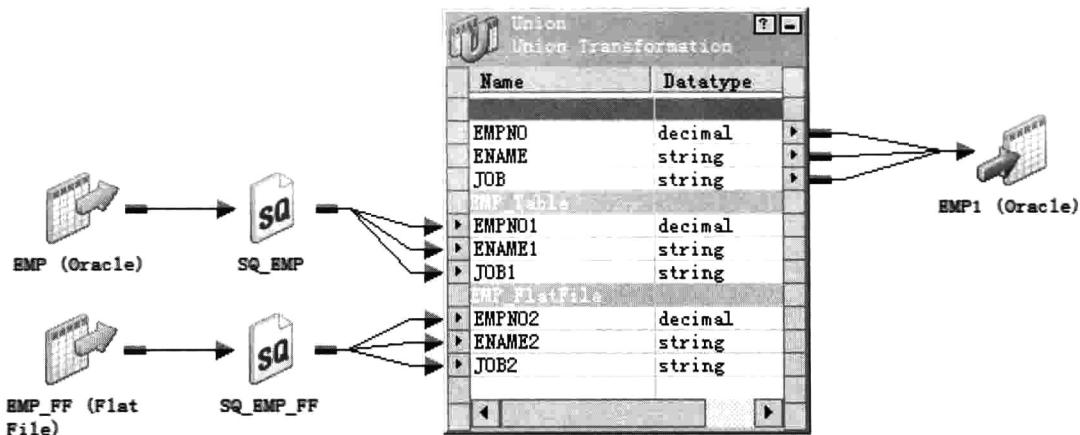


图 2-64

PowerCenter Union 组件实现的 Union 功能是 Union ALL，即如果在多个数据源中存在相同的数据，那么在 Union 的输出中这些重复数据还是存在的。那么如何消除这些重复数据呢？不知道读者是否还记得前面曾经提到过的 Sorter 组件，其中就提供了 Distinct 的功能。

## 2.11 Transaction Control

PowerCenter 使用 Transaction Control 组件控制事务的提交(Commit)或回滚(Rollback)。 Transaction Control 组件是一个事务产生器, 它在 Mapping 中定义或者重新定义事务的边界。 Transaction Control 同时清除了任何来自上游的事务。

Transaction Control 在 PowerCenter 中使用  图标。

使用 Transaction Control 控制事务的操作步骤是: 双击 Transaction Control 组件, 选择 Properties Tab。 Transaction Control 组件主要是通过设置 Transaction Control Condition 的值, 类似 IIF(Condition,Value1,Value2)来控制事务是否提交或回滚, 参数 Value1 和 Value2 表示提交、回滚或者继续事务等, 如图 2-65 所示。

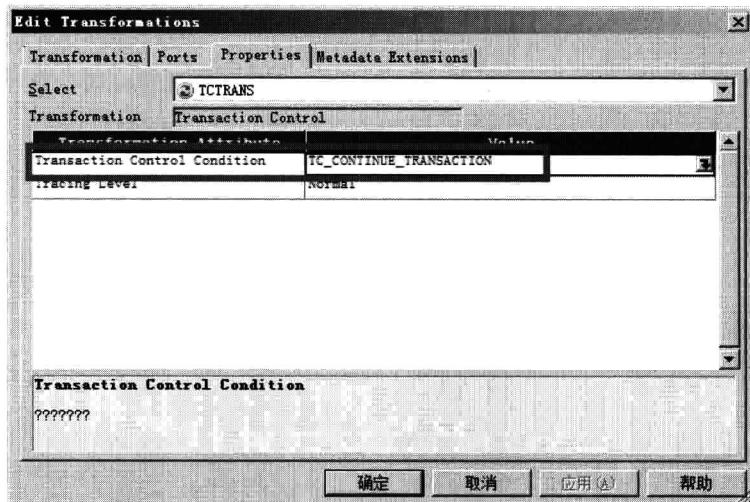


图 2-65

PowerCenter 提供了 5 个支持事务控制的常量。

- **TC\_CONTINUE\_TRANSACTION:** 对于该行数据, Integration Service 不进行任何事务变更。 TC\_CONTINUE\_TRANSACTION 也是 Transaction Control 的默认值。
- **TC\_COMMIT\_BEFORE:** Integration Service 提交当前事务, 开始一个新事务, 将当前事务的数据写入目标表。当前行进入新事务中。

- ◎ TC\_COMMIT\_AFTER: Integration Service 提交当前事务，开始一个新事务，将当前事务的数据写入目标表。当前行在本事务中。
- ◎ TC\_ROLLBACK\_BEFORE: Integration Service 回滚当前事务、开始新事务。当前行归入新事务中。
- ◎ TC\_ROLLBACK\_AFTER: Integration Service 回滚当前事务、开始新事务。当前行归入回滚事务中。

以一个简单的、极端的、没有实际意义的 Mapping 为例，说明 BEFORE 和 AFTER 的区别（但对理解非常有用），如图 2-66 所示。

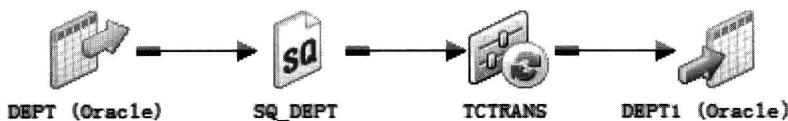


图 2-66

如果在 Transaction Control Condition 中使用 TC\_ROLLBACK\_BEFORE，只有源的最后一行被写入数据库；而如果使用 TC\_ROLLBACK\_AFTER，所有的数据将会被回滚，而不向数据库提交任何数据。

### 2.11.1 Transaction Control 有效性问题

从前面关于 Transaction Control 的介绍中已经知道，Transaction Control 的作用是结束旧事务、开始新事务。同时，Transaction Control transformations 对下游的组件或者目标而言，也可能是有效的，也可能是无效的，这就是事务的范围。当遇到以下的 Active 组件时，Transaction Control 将失效，并开始新的事务。

- ◎ Aggregator: 当 Transaction Scope 使用 All Input 时。
- ◎ Joiner: 当 Transaction Scope 使用 All Input 时。
- ◎ Rank: 当 Transaction Scope 使用 All Input 时。
- ◎ Sorter: 当 Transaction Scope 使用 All Input 时。
- ◎ Custom transformation: 用于产生新事务或者当 Transaction Scope 使用 All Input 时。

## 2.11.2 Transaction Control 组件

以一个 PowerCenter Manual 中自带的 Mapping 为例来说明，如图 2-67 所示。

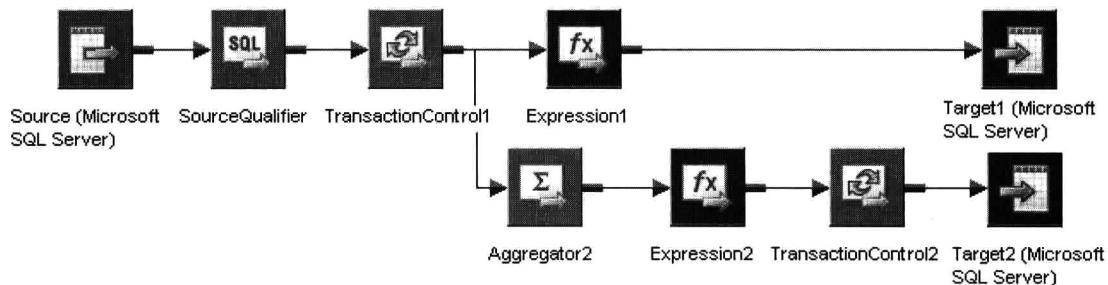


图 2-67

TransactionControl1 对 Target1 是有效的，因为在这两个组件之间没有任何使 Transaction 失效的组件。但 TransactionControl1 对 Target2 是无效的，因为在两个组件之间 Aggregator2 和 TransactionControl2 均可能使 TransactionControl1 失效，对 Target2 有效的 Transaction Control 组件是 TransactionControl2。

## 2.12 Sequence

Sequence 组件类似数据库的 Sequence，每次取值时它会自动增加。它是由 PowerCenter 提供的自增长的序列，通常用于需要生成数据记录的自增长键或序号的地方。

Sequence 在 PowerCenter 中使用的图标是

### 2.12.1 Sequence 的常规用法

**案例：**在数据仓库项目中，需要对所有的员工事实表生成一个代理主键（代理主键是数据仓库模型设计的一个概念，可以简单地理解为是一个表的无意义主键），生成主键的列为 EMPNO。开发人员开发了一个 Mapping，结果如图 2-68 所示，其中使用了 Sequence 组件。

在图 2-68 所示的 Mapping 中有几点需要特别关注：使用 NEXTVAL 连接代理主键 EMPNO，而不是使用 CURRVAL：使用 NEXTVAL 产生的 EMPNO 是一个自增量，使用 CURRVAL 产生的数据将是一个常量，在这里使用 CURRVAL 是一个错误设计。

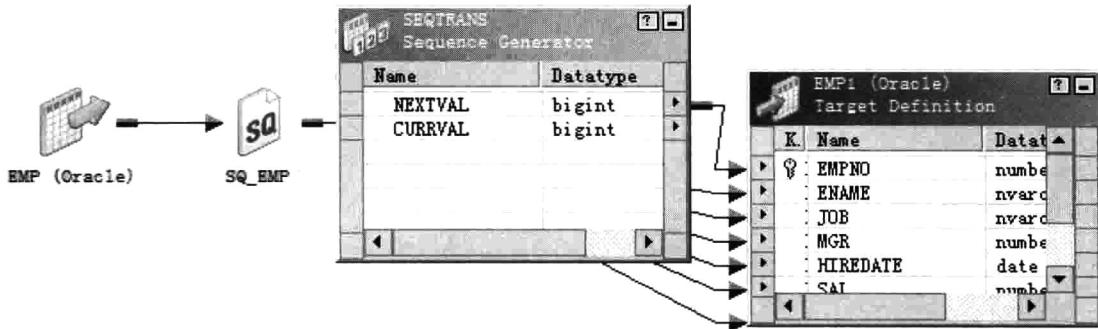


图 2-68

Sequence 组件有若干属性，双击 Sequence 组件，选择 Properties Tab，如图 2-69 所示。

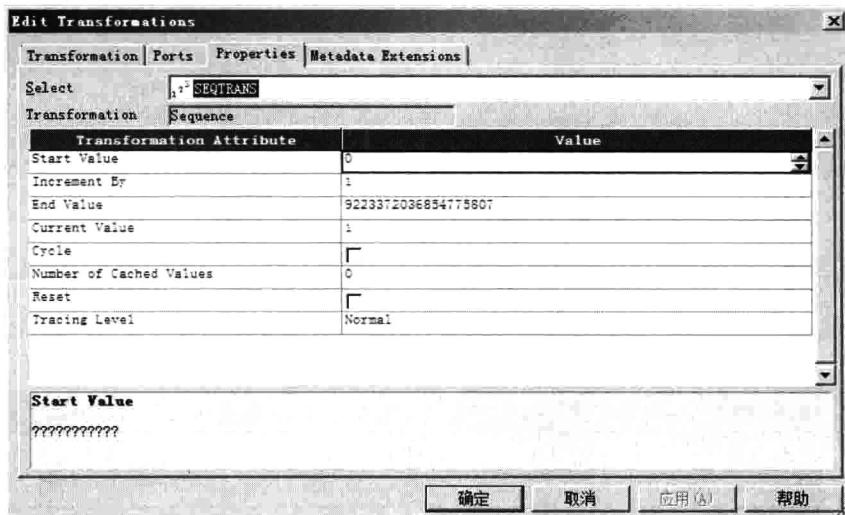


图 2-69

- ② Start Value：使用 Sequence 的开始值。如果使用了 Cycle 选项，并且当前值达到 End Value 时，会从开始值重新开始计数。
- ③ Increment By：每次运行 Sequence 的增长量。

- ◎ End Value: Sequence 的最大值, Sequence 支持的最大值是 9 223 372 036 854 775 807。
- ◎ Current Value: 每次运行后, Current 值都会发生变化, 它是下次运行时的起始值。
- ◎ Cycle: 如果启用, 当 Sequence 达到最大值时, 将从 Start Value 重新产生。

## 2.12.2 共享 Sequence

在一个 Mapping 中, 有时候希望两个事实表生成相同的代理键或者序号。程序员小 A 按照要求开发了一个 Mapping, 如图 2-70 所示。

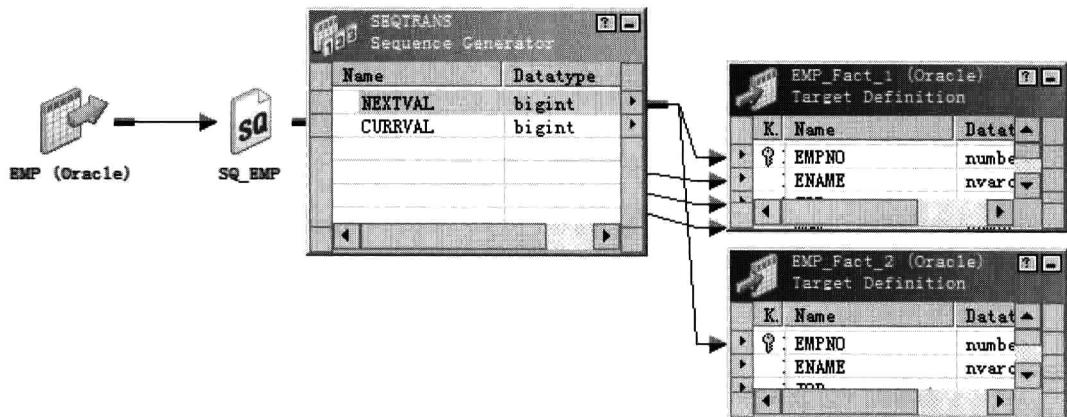


图 2-70

运行完 Session 后, 当程序员小 A 在查看结果时, 看到了一个奇怪的结果, 如表 2-8 所示。

表 2-8

EMP_FACT_1.EMPNO		EMP_FACT_1.EMPNO
1		5
2		6
3		7
4		8

程序员小 A 向一位资深工程师请教后, 修改了 Mapping, 如图 2-71 所示。

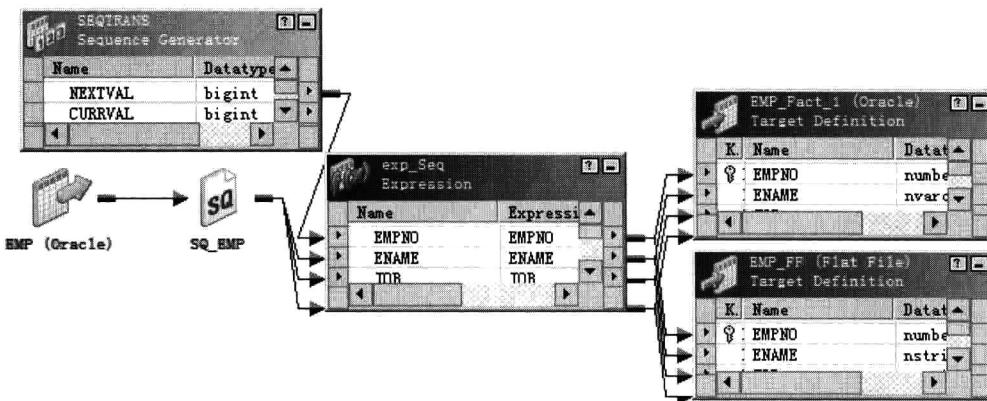


图 2-71

当小 A 再次查看结果时，得到了期望的结果，如图 2-9 所示。

表 2-9

EMP_FACT_1.EMPNO	EMP_FACT_1.EMPNO
1	1
2	2
3	3
4	4

### 2.12.3 可重用的 Sequence

当多个 Session 操作同一数据集时，例如，有 3 个来源不同、格式不同的文件都需要转换后写入同一张目标表时，需要一个 Sequence 为该表生成主键。为了保证主键不会冲突，就需要利用可重用的 Sequence。

这种情况下会用到 Sequence 的一个属性 Cached Values，其默认值是 1000。假如对此可重用的 Sequence 做了如下设置：Cached Values = 100，Current Value = 1，Increment By = 1。PowerCenter 的运行过程是这样的：当第一个 Session 启动时，PowerCenter 自动 Cache 100 个值，并将此 Sequence 在 Repository 中的 Current Value 更新为 100；当第二个 Session 启动时，Sequence 会从 101 开始使用，并同时将 Sequence Current Value 更新为 200；当 Session 使用完自己 Cache 的值之后，PowerCenter 会继续为其 Cache 100 个新值，从而保证多个 Session 之间生成的 Sequence 不会产生冲突。

## 2.13 Aggregator

Aggregator 组件是用于支持数据聚合的一个组件，完成如 SUM、AVG、MAX 等聚合运算。Aggregator 在 PowerCenter 中使用的图标是 。

Aggregator 组件在日常业务中经常会被用到。例如，需要根据部门计算各个部门的总工资。实现这样的一个典型 Mapping，如图 2-72 所示。



图 2-72

熟悉 SQL 语句的开发人员会想到这个需求同样可以用如下的 SQL 语句实现：

```
SELECT DEPTNO, SUM(SAL) FROM EMP GROUP BY DEPTNO
```

使用 PowerCenter，则是使用 Aggregator 组件来实现的，如图 2-73 所示。

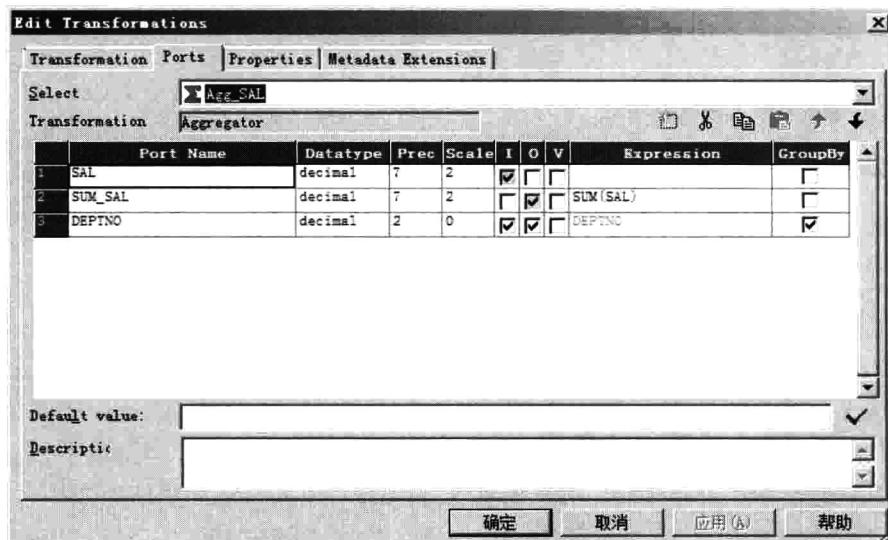


图 2-73

- 选择 GroupBy 字段：DEPTNO。
- 建立一个数据端口 SUM\_SAL，设置其表达式为 SUM(SAL)。

在 Aggregator Properties Tab 中没有特别多的属性需要关注，只有一个 Sorted Input。当使用 Sorted Input 属性时，实现逻辑与 Joiner 和 Lookup 方式非常相似，就是在 Aggregator 前面添加一个 Sorter 组件，保证到达 Aggregator 中的数据是有序的。实现的 Mapping 如图 2-74 所示。



图 2-74

在 Sorter 组件中，选择的排序列必须与 Aggregator 组件中的 GroupBy 列一致或者包含，并在 Aggregator 组件的 Properties Tab 中选中 Sorted Input。在 Aggregator 中使用 Sorted Input 会提升 Aggregator 的性能，但 Sorter 也会消耗一定的资源。

### 2.13.1 条件聚合

Aggregator 与 SQL 语句 sum... Group by 不同的是，它可以实现根据指定的条件进行汇总，如使用如下表达式：

`SUM(SAL, SAL>900)`

这个表达式表示仅仅对工资大于 900 元的员工工资进行汇总。

如目前数据库中的数据如表 2-10 所示。

表 2-10

DEPTNO	SAL
20	1100
30	950
20	3000
10	10

使用条件聚合，计算得到的结果如表 2-11 所示。

表 2-11

DEPTNO	SUM(SAL,SAL>900)
10	*
20	4100.00
30	950.00

### 2.13.2 使用 Aggregator 进行行列转换

行列转换是数据仓库项目经常用到的一种 ETL 方式，以至于出现在很多公司的面试题中。使用 Aggregator 的条件聚合可以实现从行到列的转换。假如一家企业的员工工资包括基本工资、绩效工资、津贴、岗位工资 4 种类型，目前应用系统 DBA 设计的工资管理系统表结构如下：

```
CREATE TABLE TAB_SAL
(
    EMPNO                NUMBER(4),
    INCOME_TYPE          NUMBER(4),
    INCOME_AMOUNT        NUMBER(7,2)
);
```

而数据仓库 DBA 设计的表结构如下：

```
CREATE TABLE TAB_SAL_DW
(
    EMPNO                NUMBER(4),
    BASE_SAL              NUMBER(7,2),
    PERFORMANCE_SAL       NUMBER(7,2),
    LEVEL_SAL             NUMBER(7,2),
    OTHER_SAL             NUMBER(7,2)
);
```

同时，工资管理系统 DBA 为数据仓库开发团队提供了样例数据，并说明了编码含义，如表 2-12 和表 2-13 所示。

表 2-12

EMPNO	INCOME_TYPE	INCOME_AMOUNT
1	11	1000
1	22	100
1	33	10
1	44	1
2	11	10000
2	22	1000
2	33	100
2	44	10

表 2-13

INCOME_TYPE	编码含义
11	基本工资
22	绩效工资
33	岗位工资
44	津贴

这时候摆在数据仓库开发人员面前的问题是如何使用 PowerCenter 实现将这些数据从工资管理系统结构转向数据仓库系统结构。这是一个非常高效的开发团队，他们的实现如图 2-75 所示。

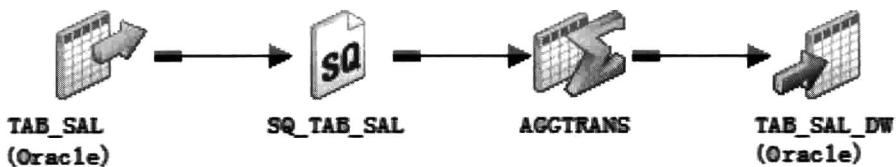


图 2-75

上面的 Mapping 看起来和普通的数据聚合 Mapping 是一样的，不同之处在于 Aggregator 中的实现。双击打开 Aggregator 的 Ports Tab，可以看到在 Aggregator 中的配置如图 2-76 所示。

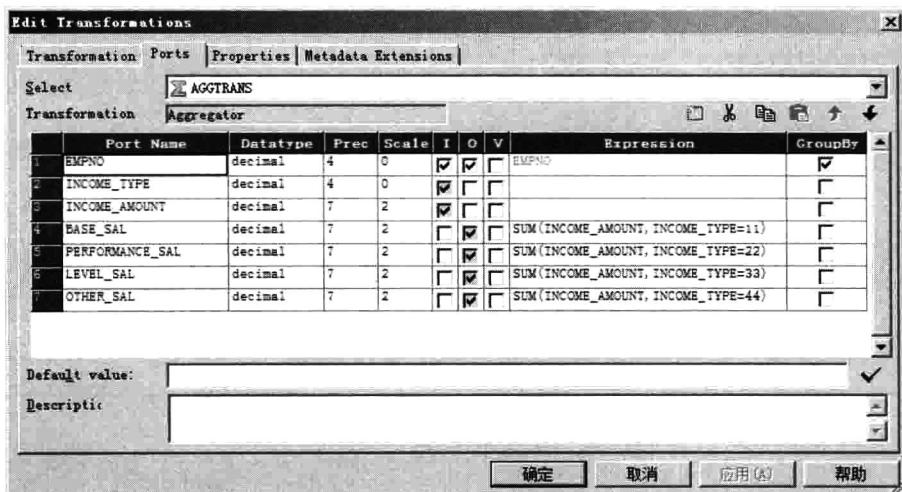


图 2-76

其中 GroupBy 字段选择了 EMPNO，工资的 4 个字段分别使用了如下表达式：

BASE_SAL	SUM(INCOME_AMOUNT, INCOME_TYPE=11)
PERFORMANCE_SAL	SUM(INCOME_AMOUNT, INCOME_TYPE=22)
LEVEL_SAL	SUM(INCOME_AMOUNT, INCOME_TYPE=33)
OTHER_SAL	SUM(INCOME_AMOUNT, INCOME_TYPE=44)

运行对应的 Workflow，通过 SQL 语句查看结果如图 2-77 所示，完全满足数据库 DBA 的要求。这是一个利用 PowerCenter Aggregator 实现行列转换的简单案例。

SQL> select * from tab_sal_dv;				
EMPNO	BASE_SAL	PERFORMANCE_SAL	LEVEL_SAL	OTHER_SAL
2	10000		1000	100
1	1000		100	10

图 2-77

## 2.14 Rank

Rank 组件用于实现获取 Top 或者 Bottom 的数据。例如，选择全公司或者各个部门工资最高的前 10 名员工，这时就需要考虑使用 Rank 组件。Rank 组件在 PowerCenter 中使用的图标是

同样以一个实际的场景为例建立一个 Mapping，这个场景就是获取全公司工资最高的两位员工的工资及员工信息，按照这个需求建立的 Mapping 如图 2-78 所示。

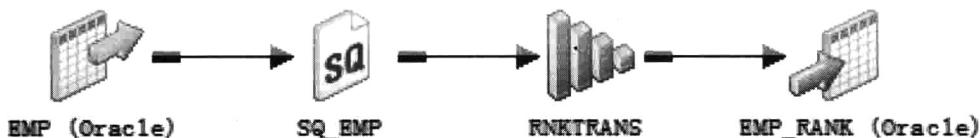


图 2-78

这个 Mapping 的核心是如何设置 Rank 组件。双击 Rank 组件，进入 Ports Tab，如图 2-79 所示。

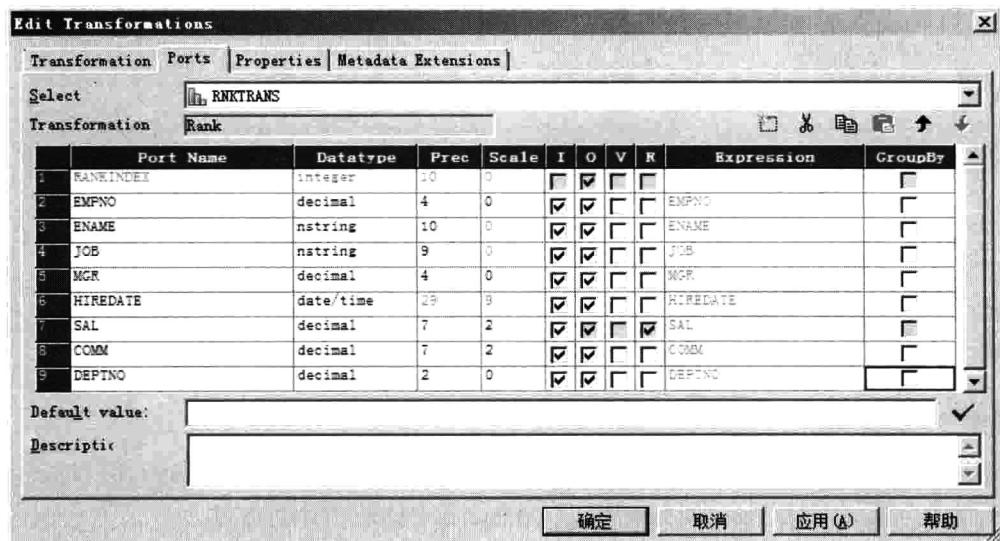


图 2-79

- ◎ **RANKINDEX:** 即最后结果排名的序列号，如排名 1、2、3、4……。
- ◎ **R:** 进行排序的列，图 2-79 中选择 SAL，即按照工资进行排序。
- ◎ **GroupBy:** 分组列。以上面的场景为例，GroupBy 没有选任何字段，即以全量数据（全公司）为基础获取 Top N 或者 Bottom N 的数据。如果 GroupBy 字段选择 DEPTNO，这样选择的含义则变成了获取各个部门工资 Top N 或者 Bottom N 的员工信息。

设置取 Top 或 Bottom，则是在 Properties Tab 中，先选择是选取最高还是最低（Top/Bottom），再选择是选取前几名还是后几名（Number of Ranks），如图 2-80 所示。

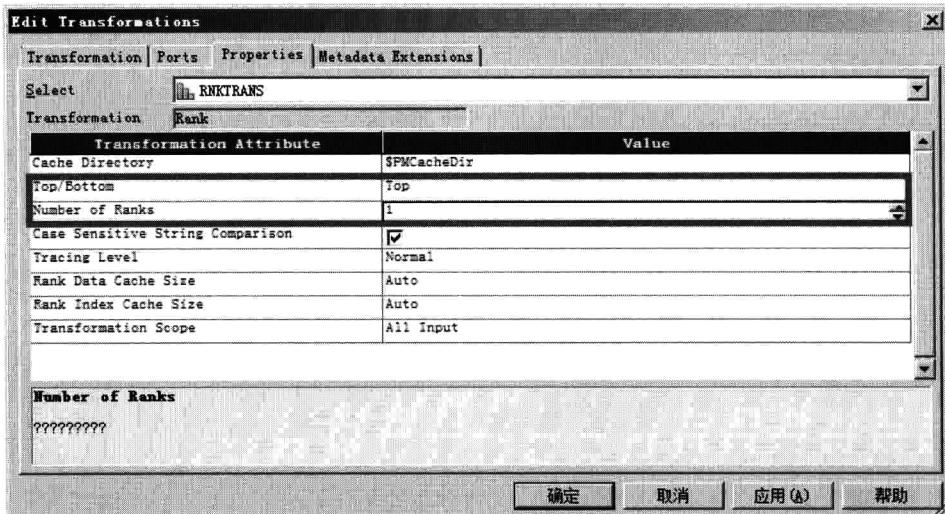


图 2-80

## 2.15 Update strategy

Update Strategy 在 PowerCenter 中使用的图标是

很多人会想到，PowerCenter 除进行 Insert 操作，是否还能进行 Update 和 Delete 等操作？答案是可以。使用 PowerCenter 进行 Update 和 Delete 有两种方法。

**方法一：** 使用 Session 属性 Treat source rows as，如图 2-81 所示。

Treat source rows as 的默认值是 Insert，所以所有读到的数据均被插入到目标表中；当选择 Update 时，所有的数据均会被更新到目标表中；当选择 Delete 时，所有传送目标的数据将试图从目标表中删除对应的数据。

**Data Driven:** 开发人员可以指定对每条数据的操作方式是插入、删除或者更新，需要在 Mapping 的 Update Strategy 组件中进行指定。

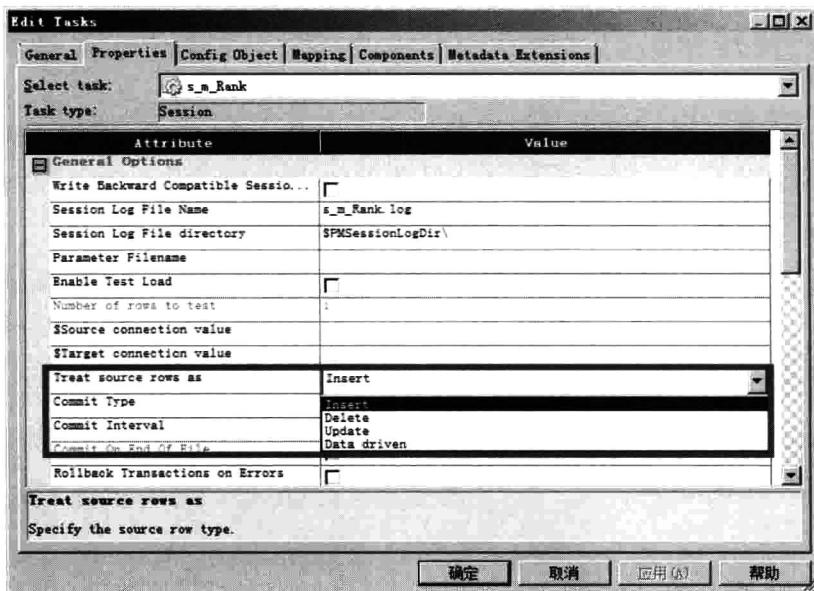


图 2-81

### → 注释

Update 和 Delete 操作都是有条件的, PowerCenter 生成的 Update 和 Delete 语句的条件依赖于 PowerCenter Designer 中逻辑主键的定义。

方法二: 使用 Update Strategy 组件。

#### 2.15.1 Treat source rows as 属性的使用

理解 Session 属性 Treat source rows as, 从下面的一个简单 Mapping 开始。

建立一个普通的 Mapping, 源表和目标表分别为 DEPT\_S 和 DEPT\_T, 这两张表的字段完全相同, 如图 2-82 所示。



图 2-82

为这个 Mapping 创建一个 Session，修改 Treat source rows as 属性为 Update，并指定对应的数据源和数据目标连接。

运行 Workflow，不出意外的话，虽然 Session 运行成功，但是会发现目标表没有被更新。同时在 Session Log 中会看到这样一条信息“Message: No column marked as primary key for table [DEPT\_T]. UPDATES Not Supported.”。

问题来了：如何创建主键？是在数据库中创建主键吗？答案：不是。正确的做法是在 PowerCenter Designer 的 Target Designer 中指定一个逻辑主键，如图 2-83 所示。

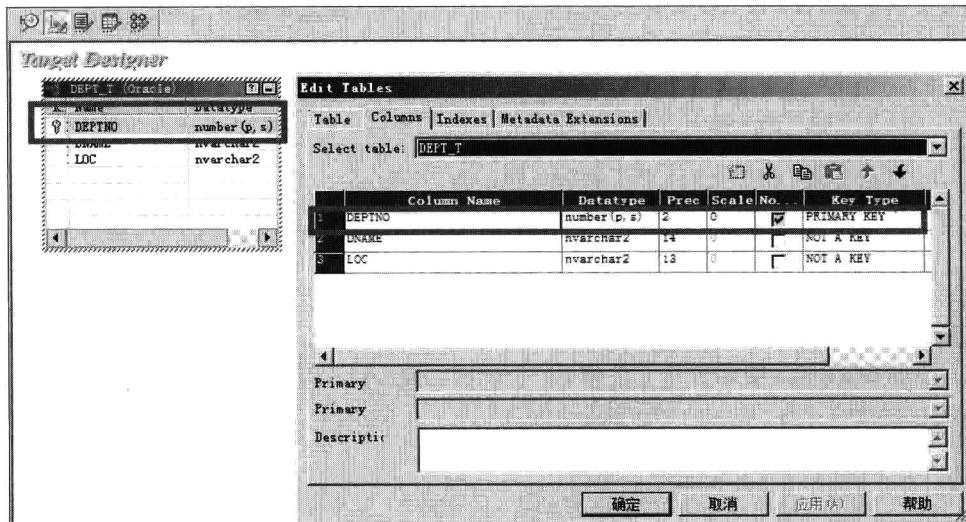


图 2-83

要注意如下事项。

(1) 这个主键是逻辑主键，是在 PowerCenter Designer 中指定的，与数据库是否有主键没有直接关系。

(2) 当指定逻辑主键后，在 Session Log 中看到的 PowerCenter 执行的 SQL 语句如下：

```
Message: Target Table DEPT_T: SQL UPDATE statement: UPDATE DEPT_T SET DNAME
= ?, LOC = ? WHERE DEPTNO = ?
```

(3) 使用这种方法后，源表中的所有数据均会执行 Update 操作，这意味着如果源表中有 100 万条数据，则将执行 100 万条 Update 操作。

(4) Update 相对 Insert 来说，执行效率要低，尤其是在目标表没有索引的情况下。因此在执行 Update 时，需要检查目标表逻辑主键对应的字段是否有索引。如果目标表很大，并没有索引，一般来讲性能都无法接受。

同样的 Mapping，如果将 Treat source rows as 设置为 Delete 的话，同样需要为 Mapping 的目标表指定逻辑主键。并建议在目标数据库中，逻辑主键上需要有恰当的索引。

刚刚提到，使用设置 Treat source rows as 的方法，所有数据都会执行统一的操作 Insert/Delete/Update，那有没有按需 Insert/Update/Delete 的方法呢？解决方法就是使用 Update Strategy 组件。

### 2.15.2 Update strategy 使用

**案例：**由于源表中的部门信息发生了更新，需要把这些信息同步到目标表中。更新的逻辑是，对部门号为 10 的部门信息执行更新；当部门编号不等于 10 时，删除相关的信息。

同样使用上一步创建的 Mapping，在上面的 Mapping 中插入一个 Update strategy 组件，如图 2-84 所示。

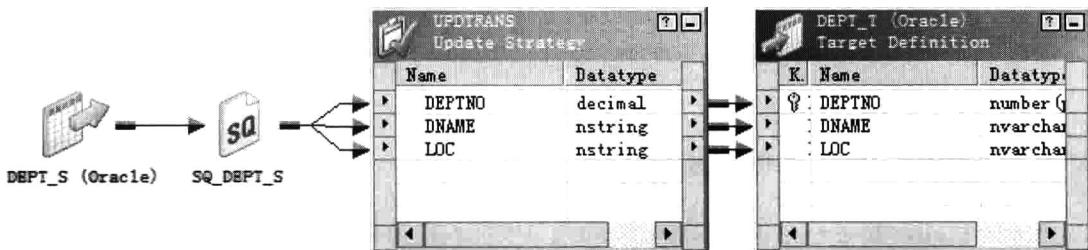


图 2-84

双击 Update Strategy 组件，选择 Properties Tab，修改 Update Strategy Expression 为表达式 “IIF(DEPTNO=10, DD\_UPDATE, DD\_DELETE）”，如图 2-85 所示。

这个表达式的含义是：当部门号为 10 时，执行 Update 操作，否则执行 Delete 操作。

把这个 Mapping 实例化为 Session 后，Session 的默认属性也有一个重要的变化。Treat source rows as 的值不再是 Insert，而是 Data Driven。它的含义是每一条数据按照 Update Strategy 组件给每一条数据赋予的属性执行操作。

另外，当使用 Update Strategy 组件时，也需要对所有的目标表指定逻辑主键，否则，PowerCenter 无法生成正确的对目标表执行某一操作的 SQL 语句。

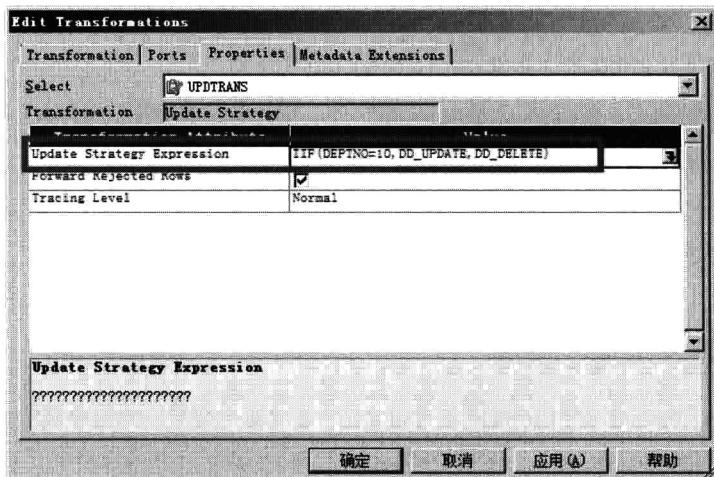


图 2-85

#### → 注释

DD\_INSERT、DD\_UPDATE、DD\_DELETE 和 DD\_REJECT 是 PowerCenter 提供的 4 个常量，在 Update Strategy 组件中使用，用于指定每一行数据的执行方式。

### 2.15.3 如何实现 Update else Insert

Update else Insert 是在数据仓库项目中进行数据同步时经常遇到的场景，在 PowerCenter 中如何实现 Update else Insert 也是一个经典的话题。在 PowerCenter 中实现该功能有两种方法。以下仍然以 DEPT\_S 表和 DEPT\_T 表为例：DEPT\_S 作为源表，DEPT\_T 作为目标表，表结构一样。DEPTNO 为逻辑主键。这两种方法分别是：

- ◎ 通过 Session 属性组合实现。
- ◎ 通过 Update Strategy 组件实现。

#### 1. 通过 Session 属性组合实现

通过 Session 属性组合实现，不需要设计复杂的 Mapping，只需要设计一个 Pass-Through Mapping 即可。

下面的例子重点是介绍设置的要点，不会再进行详细的介绍。

第一步：创建一个 Pass-Through Mapping，如图 2-86 所示。



图 2-86

第二步：重点是设置此 Mapping 实例化后的 Session，需要修改的第一个属性是 Session 的 Treat source rows as 为 Update，如图 2-87 所示。

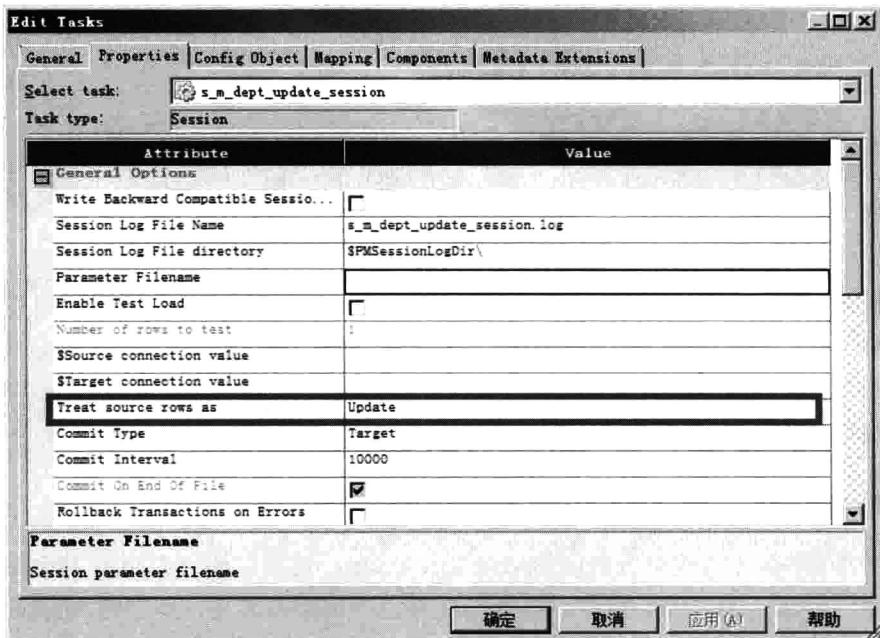


图 2-87

第三步：进入 Session 的 Mapping tab，选择 Update else Insert 和 Insert，Uncheck 包括 Delete、Update as Delete、Update as Insert 等，如图 2-88 所示。

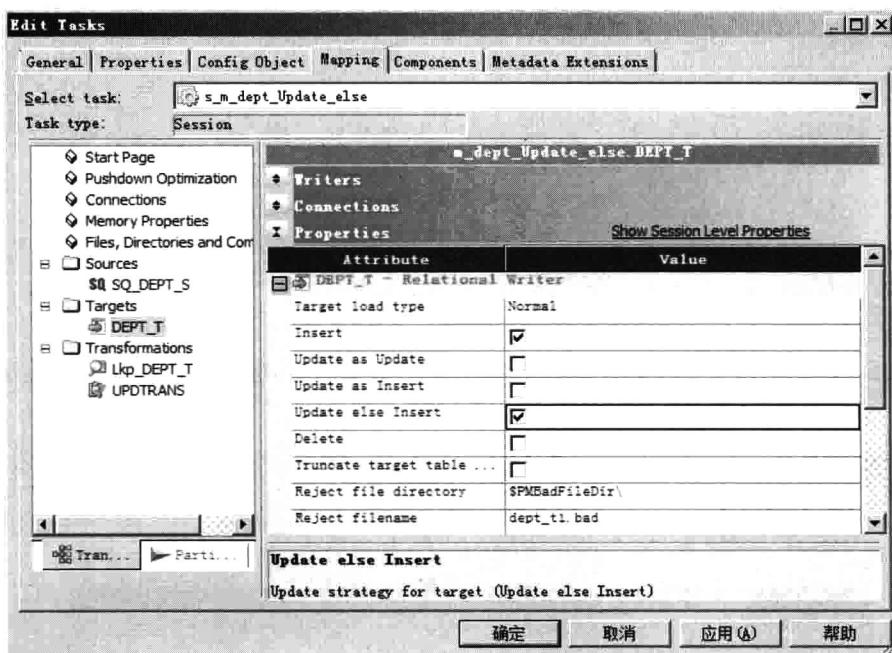


图 2-88

### → 注释

使用 Session 属性组合实现的 Update else Insert 的执行逻辑是：PowerCenter 将每条数据都使用 Update 向目标数据库写数据。当执行 Update 操作时，如果发现需要更新的对应数据，则执行完成；如果没有发现需要更新的数据，则将该条数据插入数据库。

## 2. 通过 Update Strategy 和 Lookup 组件组合实现

除了使用 Session 属性组合实现 Update else Insert，还可以通过 Update Strategy 组件更灵活地实现 Update else Insert 操作。这部分还是以介绍要点为主，不会对 Mapping 与此无关的细节进行介绍。

第一步：首先以 DEPT\_S 为源表，DEPT\_T 为目标表，查询表为目标表 DEPT\_T。完成后的 Mapping 如图 2-89 所示。

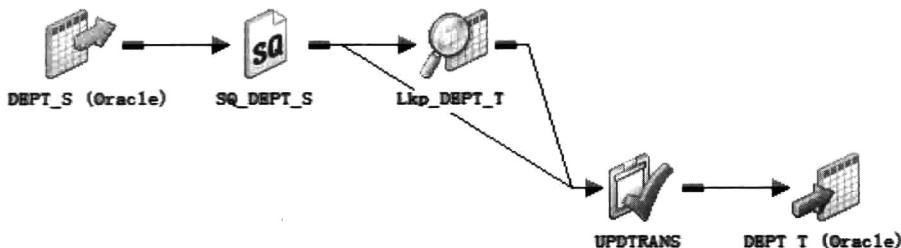


图 2-89

第二步：字段的关联如图 2-90 所示，通过 DEPTNO 进行 Lookup，将 DNAME 从源表连入 Update Strategy。

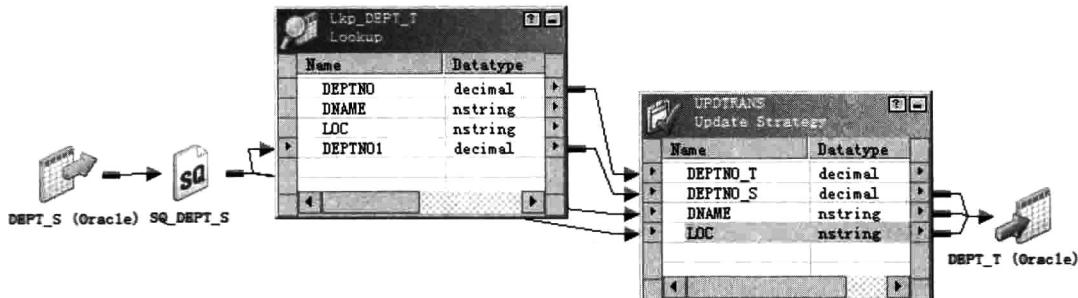


图 2-90

第三步：这里是重点，关注 Update Strategy 是如何配置的。双击 Update Strategy 组件，打开 Properties，编辑 Update Strategy Expression 为“`IIF(ISNULL(DEPTNO_T), DD_INSERT, DD_UPDATE)`”，如图 2-91 所示。这个表达式的含义是：当源的 DEPTNO 在目标表中无法发现一样的 DEPTNO 时 (`DEPTNO_T=NULL`)，执行插入操作；如果找到对应的 DEPTNO，则执行更新操作。

这时，当把此 Mapping 实例化为 Session 后，将此 Session 的 Properties Tab 中的 Treat source rows as 的默认值变为 Data Driven。

#### → 注释

细心的读者可能会注意到这是个偷懒的方法，当执行 Update 操作时，并没有考虑除主键列外的其他列是否发生变化，全部进行了 Update。如果需要做得更加精细，可以参考 Dynamic Lookup 部分。

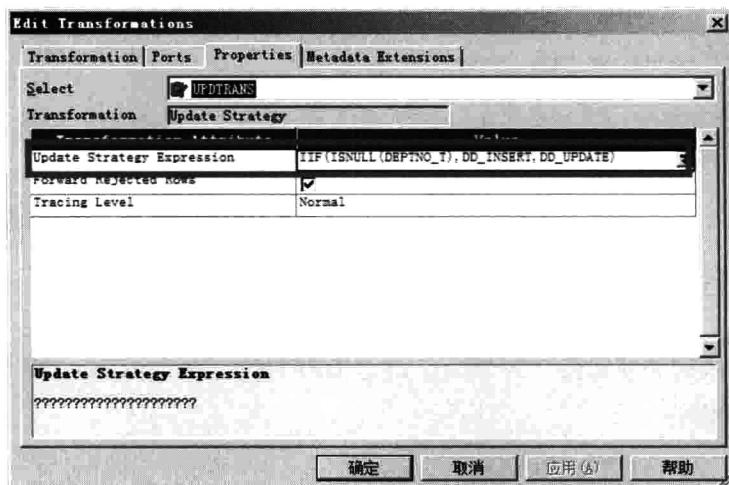


图 2-91

#### 2.15.4 Update Stagety 案例：缓慢变化维

在维度建模的数据仓库中，有一个著名概念叫 Slowly Changing Dimensions，中文一般翻译成“缓慢变化维”，经常被简写为 SCD。缓慢变化维的提出是因为在现实世界中，维度的属性并不是静态的，它会随着时间的流逝发生缓慢的变化。这种随时间发生变化的维度一般称为缓慢变化维，并且把处理维度表的历史变化跟踪的问题称为处理缓慢变化维的问题，有时也简称为处理 SCD 的问题。

处理缓慢变化维的方法通常分为 3 种。

第一种方式是直接覆盖原值。这种处理最容易实现，但是没有保留历史数据，无法分析历史变化信息。第一种方式通常简称为 TYPE 1。

第二种方式是添加维度行。这种处理需要代理键的支持。实现方式是当有维度属性发生变化时，生成一条新的维度记录，主键是新分配的代理键，通过自然键可以和原维度记录保持关联。第二种方式通常简称为 TYPE 2。

TYPE 2 在 PowerCenter 中有 3 种实现方式。

- Version Data: 在维度表中增加一个新的 Version Number 字段，当数据发生更新时，通过更新这个字段跟踪历史版本的变化。

- ◎ Flag CURRENT：在维度表中增加一个新的字段 CURRENT，即当前标志，当数据发生变化时，更新 CURRENT 字段来跟踪当前的有效值和历史信息。
- ◎ Effective Date Range：通过在表中增加两个日期字段“生效日期”和“失效日期”来跟踪历史数据的变化。

第三种方式是添加属性列。这种处理的实现方式是对于需要分析历史信息的属性添加一列，来记录该属性变化前的值，而本属性字段使用 TYPE 1 来直接覆盖。这种方式的优点是可以同时分析当前及前一次变化的属性值，缺点是只保留了最近一次变化信息。第三种方式通常简称为 TYPE 3。

在上面的 3 种类型、5 种实现方式中，其中 TYPE 2 的 Effective Date Range 方式是比较复杂的一种。下面以这种类型为例演示如何在 PowerCenter 中实现缓慢变化维的经典模式。

### 1. SCD 样本数据

创建一个原始表 SCD\_ITEM 作为数据源，并假设其有两条原始数据如下：

```
CREATE TABLE SCD_ITEM
(
    ITEM      VARCHAR2 (10),
    STYLES    NUMBER (10)
);
```

ITEM	STYLES
Sock	13
Boot	20

### 2. SCD PowerCenter 实现过程

PowerCenter 已经内置了缓慢变化维的实现向导。在 PowerCenter 中针对表 SCD\_ITEM 实现 TYPE 2 的 Effective Date Range 缓慢变化维的过程如下。

第一步：首先导入 SCD\_ITEM 表作为数据源。

第二步：选择 Mapping 菜单 Wizards→Slowly Changing Demision 命令，输入 Mapping Name，如 m\_SCD\_ITEM\_TYPE2，并选择 Type 2 Dimension-keep a full history of changes in the 单选按钮，如图 2-92 所示。

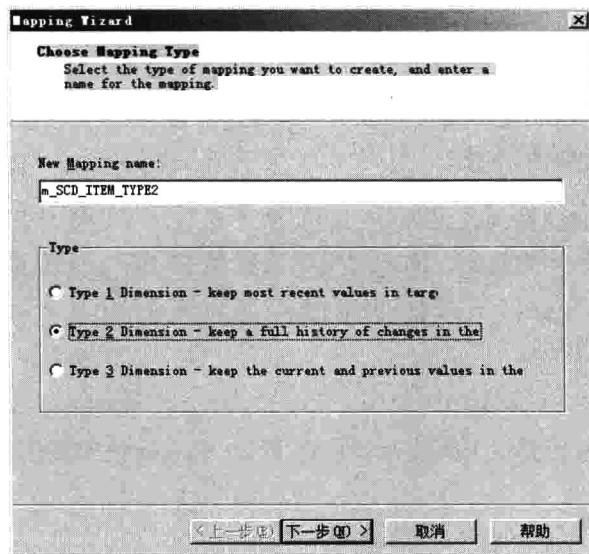


图 2-92

第三步：单击“下一步”按钮选择数据源 SCD\_ITEM，并为目标表输入一个表名（注：目标表是 PowerCenter 自动生成的），如图 2-93 所示。

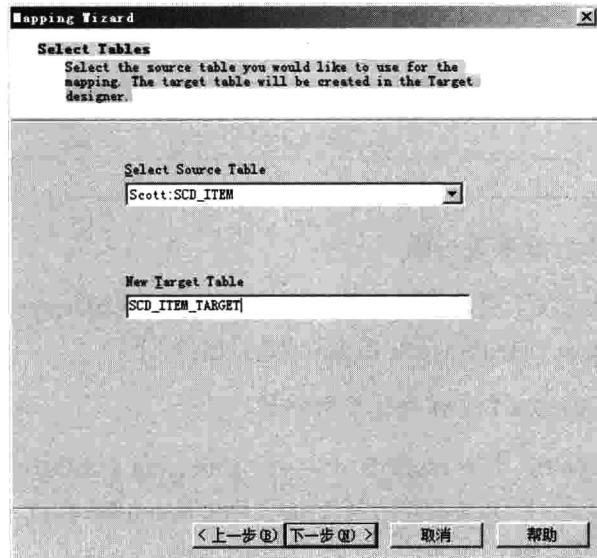


图 2-93

第四步：继续单击“下一步”按钮，选择表的 Logic Key 和要跟踪变化的列，这里选择 ITEM 列作为 Logic Key，STYLES 列作为跟踪变化的列，如图 2-94 所示。

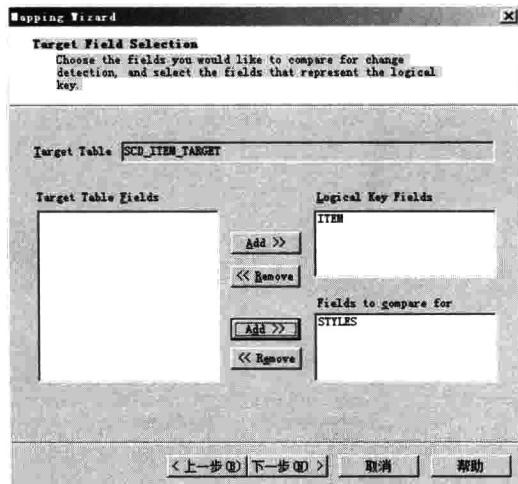


图 2-94

第五步：继续单击“下一步”按钮，这里需要选择使用哪种跟踪历史的方法，如前所述，选择使用“生效日期”和“失效日期”的方法作为样例。其他的两种方法分别是：使用版本号和使用当前标志，如图 2-95 所示。

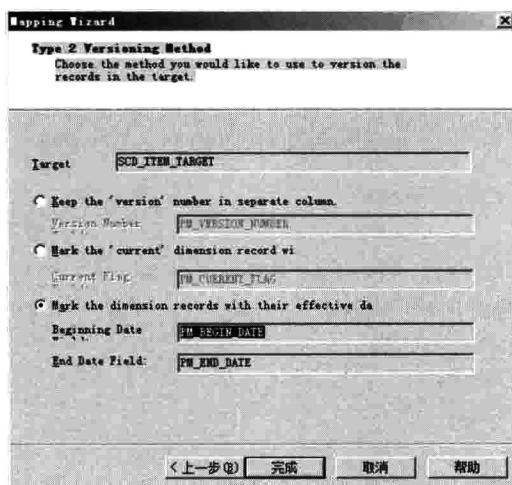


图 2-95

第六步：单击“完成”按钮，PowerCenter 自动生成一个经典的可以支持缓慢变化维的 Mapping，如图 2-96 所示。

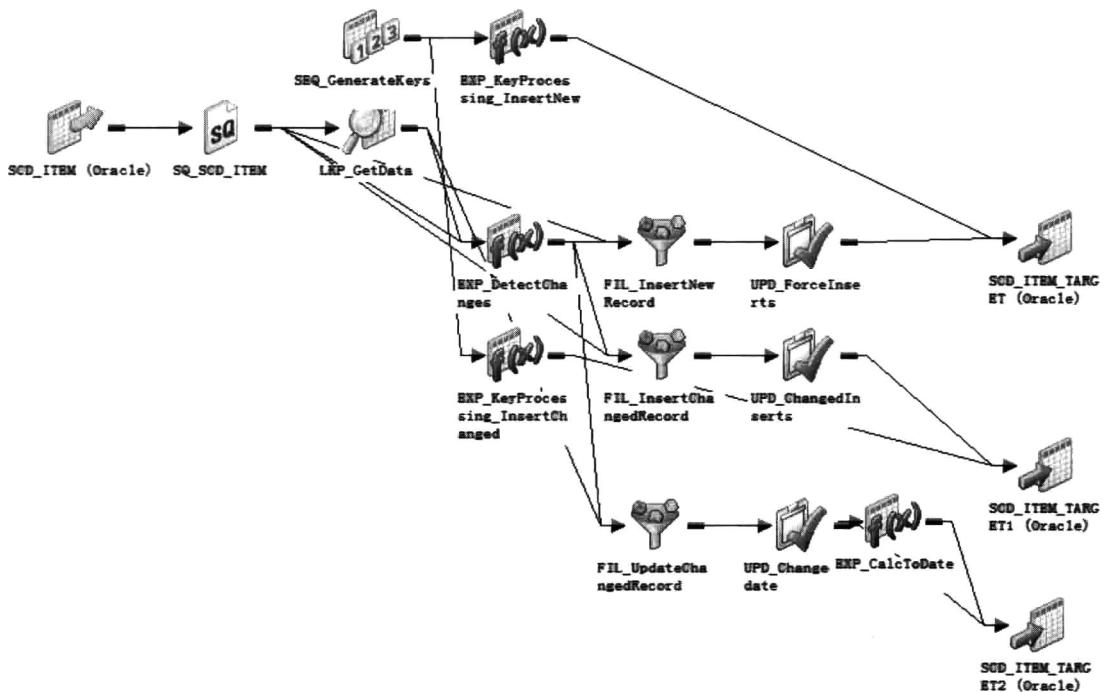


图 2-96

这个 Mapping 与此前见到的 Mapping 相比有点复杂，首先关注一下自动生成的目标表的结构。

```

CREATE TABLE SCD_ITEM_TARGET
(
    PM_PRIMARYKEY number(10) NOT NULL,
    ITEM          nvarchar2(10),
    STYLES        number(10),
    PM_BEGIN_DATE date,
    PM_END_DATE   date
);
ALTER TABLE SCD_ITEM_TARGET ADD PRIMARY KEY (PM_PRIMARYKEY);

```

目标表与原始表相比增加了 3 个字段。

- ◎ PM\_PRIMARYKEY：代理主键。
- ◎ PM\_BEGIN\_DATE：维度值的生效日期。
- ◎ PM\_END\_DATE：维度值的失效日期。

下面以一组数据实例说明一下这个 Mapping 的运行过程和执行逻辑。

首先，已知 SCD\_ITEM 源中有两行数据，第一次运行，在目标表中的结果如表 2-14 所示。

表 2-14

PM_PRIMARYKEY	ITEM	STYLES	PM_BEGIN_DATE	PM_END_DATE
1	Sock	13	2015-03-03	
2	Boot	20	2015-03-03	

假如第二天，SCD\_ITEM 数据发生了一些变化。如 Sock 的 Styles 由 13 变成了 23，同时新增了一条数据 Shoe，STYLES 为 33，如表 2-15 所示。

表 2-15

ITEM	STYLES
Sock	23（修改）
Boot	20
Shoe	33（新增）

这时运行对应的 Session，目标表的数据如表 2-16 所示。

表 2-16

PM_PRIMARYKEY	ITEM	STYLES	PM_BEGIN_DATE	PM_END_DATE
1	Sock	13	2015-03-03	2015-03-04
2	Boot	20	2015-03-03	
3	Sock	23	2015-03-04	
4	Shoe	33	2015-03-04	

上面的数据说明：Sock、13 的生效时间范围为 2015-03-03~2015-03-04，PM\_END\_DATE 刚刚被更新；同时 Sock、23 是一条新插入的数据，它的生效日期是 2015-03-04。而 Shoe、33 是一条全新的数据，它的生效日期是 2015-03-04。

上面的演示过程是一个样例，是作为一个基础的模板供开发人员修改的。在现实的业务中，一般都需要根据实际的业务和数据对 Mapping 做些修改。

## 2.16 SQL Transformation

SQL Transformation 在 PowerCenter 中使用的图标是 。

SQL Transformation 为 PowerCenter 在 Mapping 中执行 SQL 语句提供了一种能力。例如，一个虚拟的需求是这样的：业务部门并不希望直接使用报表，而是希望进行自助分析。这时，他们的需求是：我们需要数据，而不是报表。具体为：他们希望输入一个表名，开发人员用 PowerCenter 开发一个 Mapping 返回全表的数据。这时可以使用 SQL Transformation。

SQL Transformation 提供了两种模式，以最大化 SQL Transformation 的灵活性。这两种模式包括 Script Mode 和 Query Mode，Query Mode 又分为 Static Query Mode 和 Dynamic Query Mode。

下面将对这几种模式一一介绍。

### 2.16.1 Script Mode

脚本模式是 SQL Transformation 的一个简单模式，开发人员可以向 SQL Transformation 直接发送一个 SQL 语句，PowerCenter 则执行这个语句，并返回这个语句执行成功与否。以一个样例开始，每天启动 ETL 过程时，在数据库中用一个时间戳标识当天的启动时间，因此希望 PowerCenter 执行如下 SQL：Insert into TAB\_Control Values(sysdate)。

实现如上的 Mapping，需要完成如下步骤。

第一步：创建一个 Mapping，将其命名为 m\_SQL\_Script，并在此 Mapping 中添加一个 SQL Transformation。这时弹出的第一个对话框如图 2-97 所示。

选择脚本模式，选择数据库类型，并决定使用 Static Connection 还是 Dynamic Connection。Dynamic Connection 的含义是有可能这个 SQL 根据需要会在不同的数据库中执行，Mapping 会传递连接信息给 SQL Transformation；Static Connection 的含义是数据库是固定的。

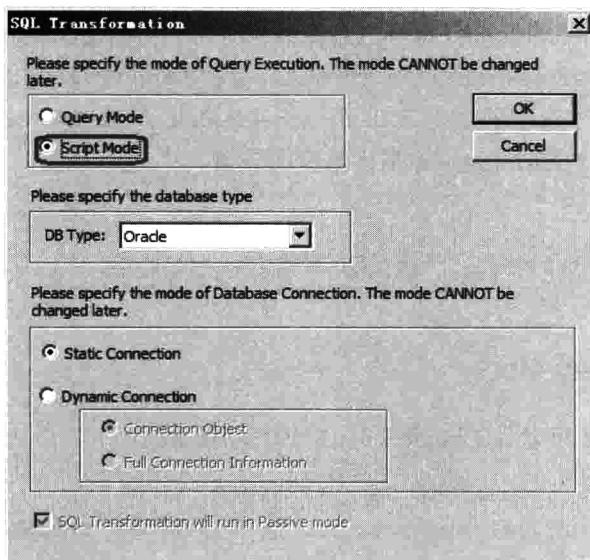


图 2-97

第二步：当 SQL Transformation 被加入到 Mapping 中后，我们会留意到它有 3 个端口：一个输入端口和两个输出端口，如表 2-17 所示。

表 2-17

类 型	端 口	描 述
Input	ScriptName	输入包含 Script 的文件名（注：不是脚本本身）
Output	ScriptResult	Scirpt 执行成功返回 Passed，失败返回 Failed
Output	ScriptError	当脚本执行错误时，返回错误的具体信息

第三步：为 SQL Transformation 指定数据源和目标，并创建 Mapping，如图 2-98 所示。

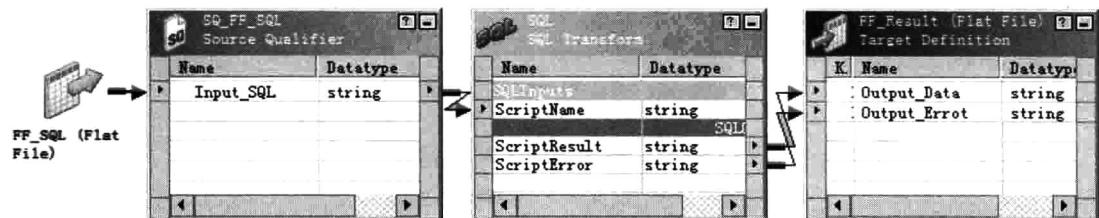


图 2-98

第四步：创建 Workflow wf\_SQL\_Script。在 Workflow 中有两个需要重点留意的点：

- ◎ 假如 Session 中的 Input File 是 FF\_SQL.txt，在 FF\_SQL.txt 中存放的是 Script 的文件名，如 C:\Informatica\9.6.1\server\infa\_shared\SrcFiles\script.txt，而不是脚本本身。在 Script.txt 中存放的才是 SQL 语句 Insert into tab\_control values(sysdate)。
- ◎ 需要为 SQL Transformation 指定数据库连接。

→ **注释**

在 Script.txt 中可以同时指定多条 SQL 语句。

## 2.16.2 Static Query Mode

Static Query Mode 可以在 PowerCenter 中执行以下语句：

```
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
SELECT Name, Address FROM Employees
WHERE Employee_Num =?Employee_ID? and Dept = ?Dept?
```

两个问号（??）之间即为从上游 Transformation 传入 SQL 组件的参数，如上面的**?Employee\_ID?**。

同样，以一个具体的案例说明 Static Query Mode。业务部门希望通过输入 Department 编号（DEPTNO）返回表 DEPT 中的部门名称（DNAME）和部门所在城市（LOC）。要实现的 SQL 语句如“select DNAME, LOC from DEPT where DEPTNO=?DEPTNO?”。这个 Mapping 的实现过程如下。

第一步：创建一个 Mapping m\_SQL\_Static\_Query，并添加一个 Query Mode 的 SQL Transformation，这时的 SQL Transformation 如图 2-99 所示，还需为其添加输入/输出端口。

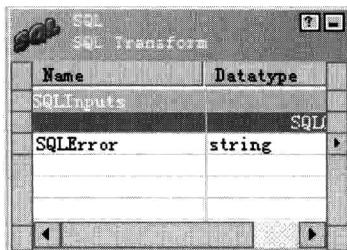


图 2-99

第二步：双击 SQL Transformation，选择 SQL Ports Tab，为其添加 SQLInputs 端口 INPUT\_Parameter 作为输入参数，添加 SQLOutputs 端口 DNAME 和 LOC，如图 2-100 所示。

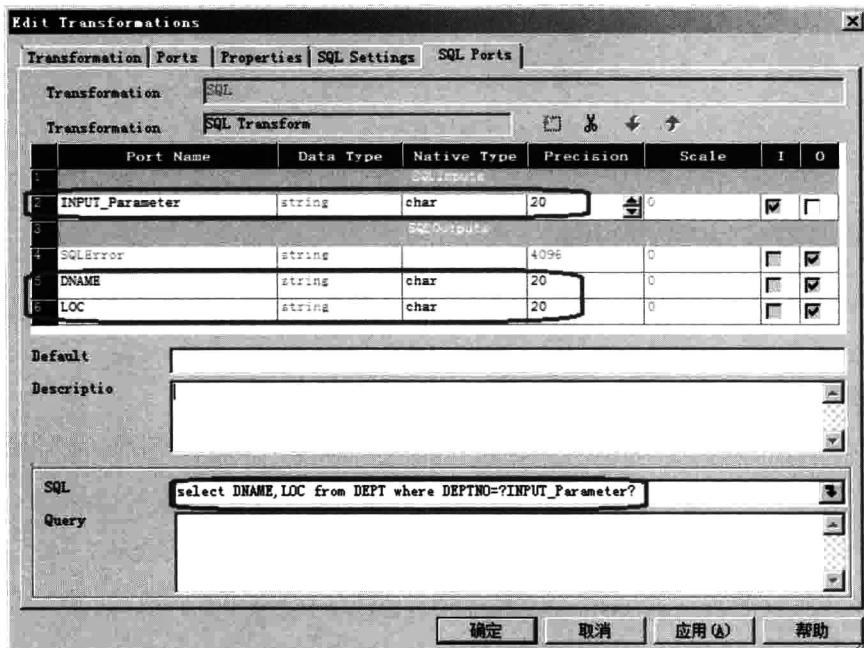


图 2-100

还需要在 SQL Editor 中添加如下 SQL 语句：

```
SELECT DNAME, LOC from DEPT where DEPTNO=?INPUT_Parameter?
```

#### → 注释

在添加 SQLOutputs 端口时，添加了端口 DNAME 和 LOC。这两个字段最好与 SQL Editor 中的 Select 后返回的字段名相同。但这不是强制要求。事实上只要 SQL 语句返回的字段的数量与添加端口的数量和类型保持一致即可。但为了程序的可读性，最好是保持二者完全一致。

第三步：添加源和目标，创建 Mapping，如图 2-101 所示。

这时，在输入文件中输入的是部门编码（DEPTNO），在目标文件中返回的就是部门名称和部门所在城市。

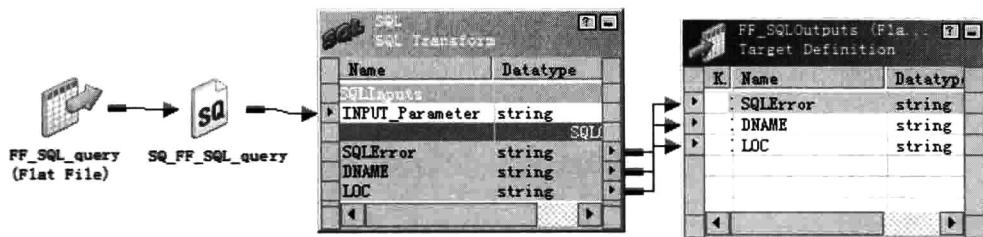


图 2-101

### 2.16.3 Dynamic Query Mode

Dynamic Query 模式更加灵活，可以执行如下 SQL 语句：

```
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'  
~Query_Port~
```

如黑体字体所描述的，Dynamic Query 可以在 Query 中动态地使用表名或者完全动态地生成整个 SQL 语句。它的开发过程与 Static Query 模式非常相似。

第一步：创建 Mapping，添加 SQL Transformation，选择 Query Mode。

第二步：为 SQL Transformation 添加端口，如图 2-102 所示。

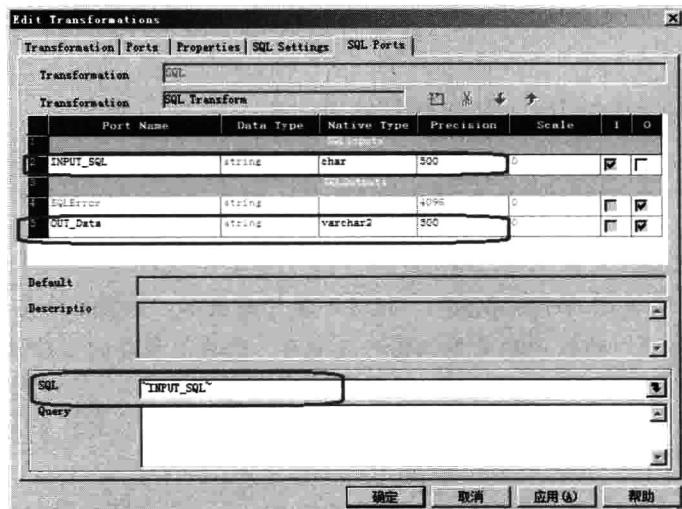


图 2-102

与 Static Query 模式不同点在于，这里使用的是双波浪号 (~)，而非双问号 (??)。

第三步：添加数据源和目标，完成 Mapping 设计。

假如完全按照如上的设计开发完成。这时在输入 SQL 中使用如下 SQL 语句：

```
Select * from emp
```

这条语句的特别之处是直接使用 “\*”，而不是字段名。这时返回的值为表的第一个字段名。

假如使用如下 SQL 语句：

```
Select ENAME from EMP
```

这时返回的数据为所有的员工姓名。

思考：如像这节开始提到的需求，输入一个表名，返回这张表的所有值到一个文本文件。该如何实现呢？

提示：表的字段数量可能是不确定的。难点 1：如何动态地拼接 SQL？难点 2：如果希望每张表对应不同的输出文件，如何实现？

## 2.17 Java Transformation

Java Transformation 在 PowerCenter 中使用的图标是 。

它是 Informatica PowerCenter 中的一个特殊转换组件，可以使用 Java 代码处理一些特殊的业务场景，比如判断字段是否含有中文、汉字转拼音、加密/解密、行列转换等 PowerCenter 内置组件不能完成，或者使用其他组件完成比较复杂的场景。

如需使用 Java 组件，开发人员需具备 Java 语言基础。

### 2.17.1 Java Transformation 简介

#### 1. Passive 与 Active 的区别

一个 Passive 的 Java 组件，每个输入行进行处理后，都只生成一个输出行。

一个 Active 的 Java 组件，每个输入行进行处理后，可能会生成一个或多个输出行。

## 2. Java Code 标签介绍

Java Code 标签如图 2-103 所示。

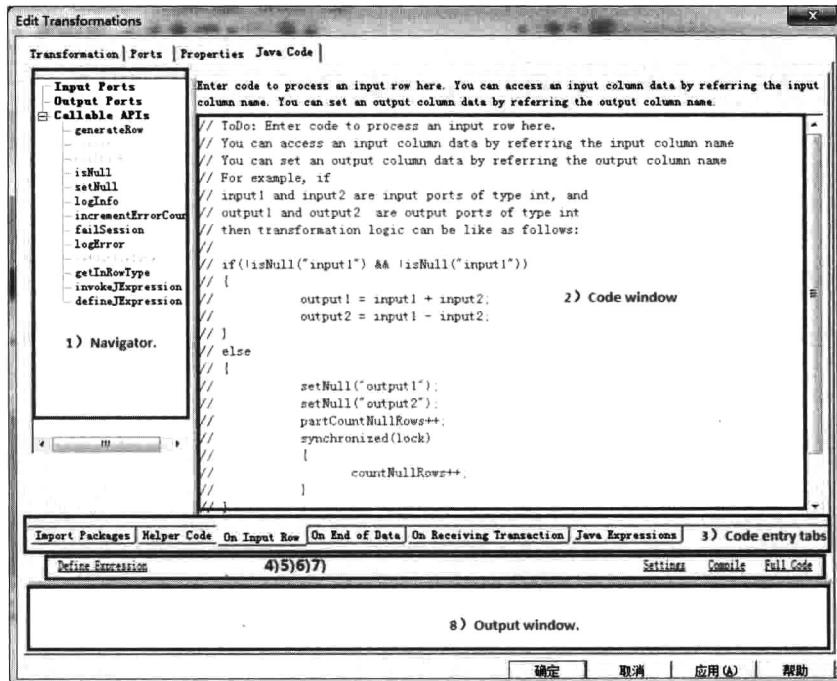


图 2-103

(1) Navigator: 左侧导航栏，主要包括以下 3 部分。

- ① Import Ports: 该区域显示 Java 组件的所有输入项。
- ② Output Ports: 该区域显示 Java 组件的所有输出项。
- ③ Callable APIs: Java 组件中可以调用的 API 方法，多用于 Active 类组件。

(2) Code Window: Java 代码编辑窗口，在此窗口进行 Java 代码编写。

(3) Code Entry Tabs: 定义组件的不同部分，每个标签与 Code Window 关联。

- ④ Import Packages: 编写 Java 代码中的 Import 部分，导入第三方或者用户自定义的依赖包，如 import java.io.\*;。

- ◎ Helper Code: 定义 Java 变量或者 Java 方法中用到的变量, 如 private boolean generateRow;。
- ◎ On Input Row: 针对 Input Port 进行的业务处理。
- ◎ On End of Data: 针对 On Input Row 中处理过的 Input Port 进行处理。
- ◎ On Receiving Transaction: 用于 Active Java 组件, 处理接收到 Transaction Notification 后的业务逻辑。
- ◎ Java Expressions: 编写 Java 类的 method, 可以在 Helper Code、On Input Row、On End of Data 和 On Transaction Code 中调用。

(4) Define Expression Link: Java 方法的定义, 如图 2-104 所示。

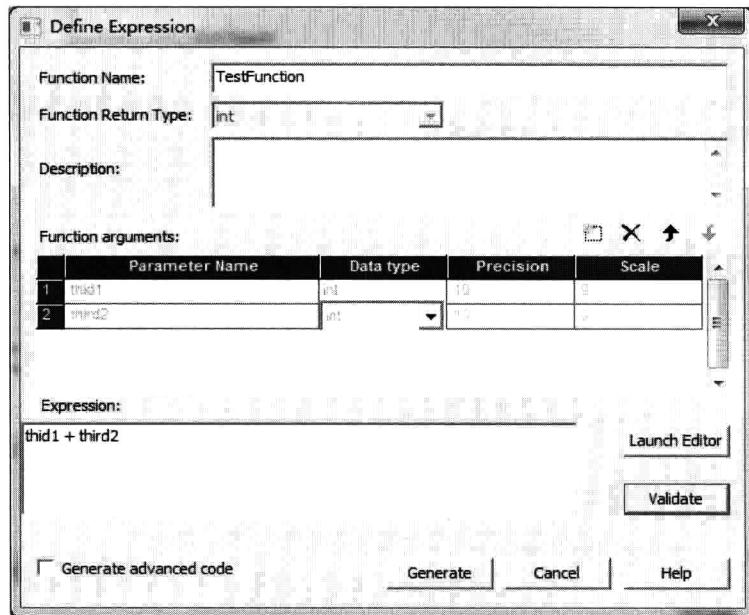


图 2-104

编辑后, 会在 Java Expressions 中生成相应的代码, 如图 2-105 所示。

(5) Settings Link: 添加第三方的.jar 包, 用户 PowerCenter Client 编译 Java 类, 如图 2-106 所示。

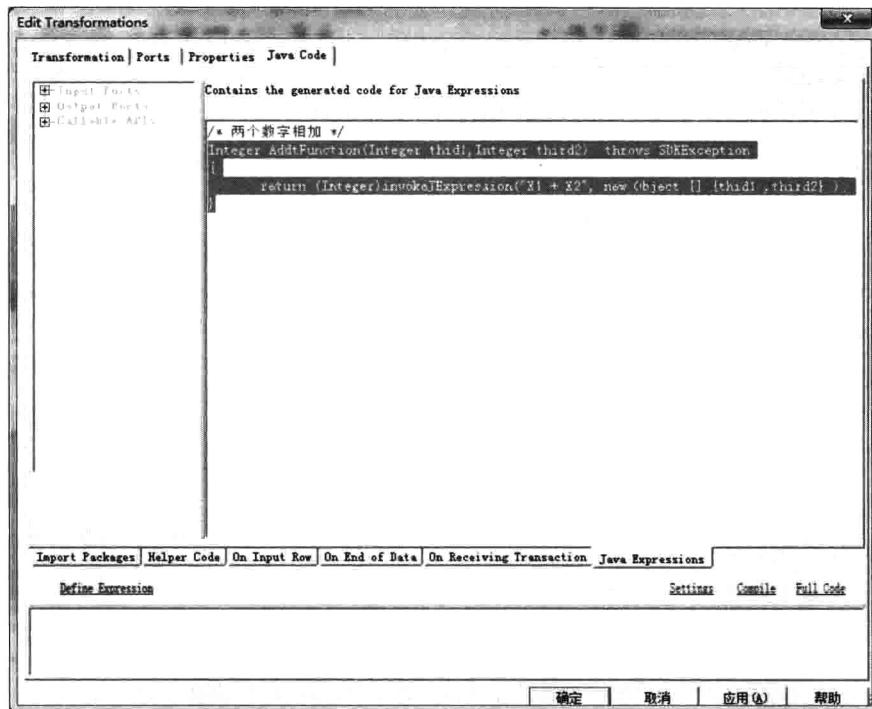


图 2-105

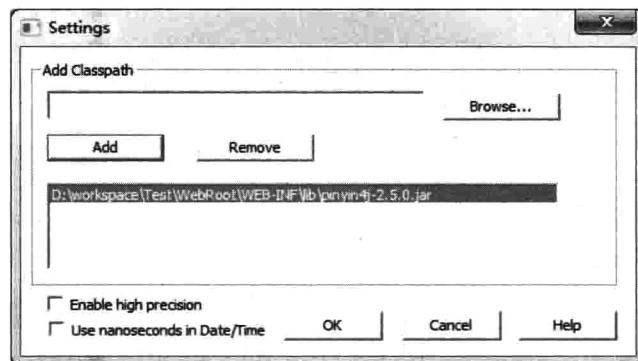


图 2-106

(6) Compile Link: 编译 Java 代码, 编译结果显示在 Output Window 中, 如图 2-107 所示。

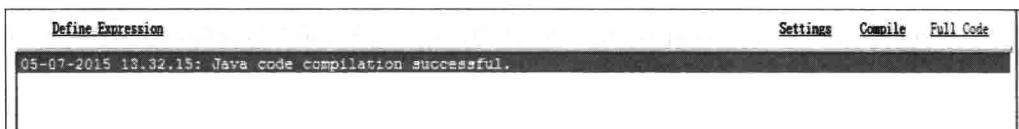


图 2-107

(7) Full Code Link: 查看 Java 组件生成的完整的 Java 代码。

(8) Output Window: Java 编译结果输出窗口。

### 3. Java 与 PowerCenter 类型转换

Java Transformation 会自动进行 PowerCenter DataType 与 Java DataType 之间的转换，转换关系如表 2-18 所示。

表 2-18

PowerCenter Datatype	Java Datatype
Char	String
Binary	byte[]
Long (INT32)	int
Double	double
Decimal	double BigDecimal
BIGINT	long
Date/Time	BigDecimal long (number of milliseconds since January 1, 1970 00:00:00.000 GMT)

### 4. Java Transformation API 简介

- Commit: 生成提交事务。
- failSession: 异常处理，使用 failSession 可以停止一个 Session。
- generateRow: 在 Active 类的 Java 组件中，生成输出行。
- getInRowType: 返回当前行的输入类型，返回值有 Insert、Update、Delete 和 Reject。
- getMetadata: 获取 Java Transformation 的元数据信息。
- incrementErrorCount: Session 错误统计。
- isNull: 判断输入行的某一列是否为空值。
- logError: 记录 Java Transformation 的错误日志到 Session Log 中。

- ◎ logInfo：记录 Java Transformation 的普通信息到 Session Log 中。
- ◎ rollBack：生成一个回滚事物。
- ◎ setNull：将输出行的某一列设置为空值。
- ◎ setOutRowType：设置输出行的操作策略，可以为 Insert、Update、Delete。
- ◎ storeMetadata：存储 Java Transformation 的元数据信息。

## 2.17.2 Passive Java Transformation

**场景：**在数据加载的过程中，增加一列中文名字的拼音列，如图 2-108 所示。

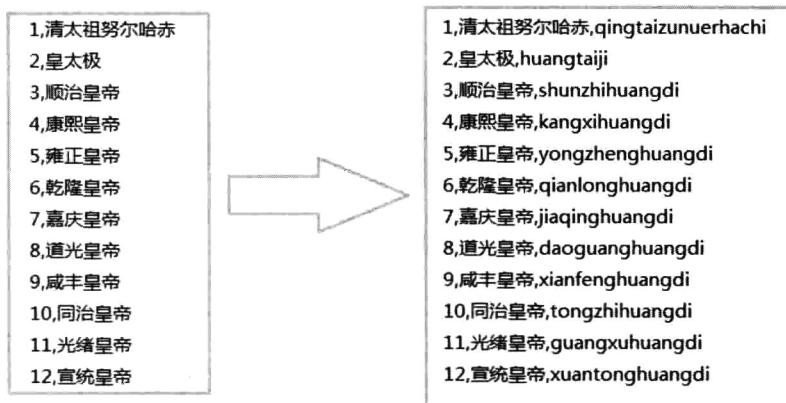


图 2-108

**解决方案：**利用 PowerCenter 中的 Java Transformation 组件及 Java 代码中的 pinyin4j-2.5.0.jar 来实现。

Java 实现方式如图 2-109 所示。

与 PowerCenter 的 Java Transformation 实现相比较，需要注意 Import 部分、属性部分、方法部分、实现部分。

PowerCenter+Java Transformation 实现方法如下：

- (1) 导入数据源、目标，创建 Mapping。
- (2) 在 Mapping 中添加 Java Transformation 组件，单击 Desginer 中的 按钮，即可添加，出现 Passive 和 Active 选择页面，如图 2-110 所示。

```
//Import部分
import net.sourceforge.pinyin4j.PinyinHelper;
import net.sourceforge.pinyin4j.format.HanyuPinyinCaseType;
import net.sourceforge.pinyin4j.format.HanyuPinyinOutputFormat;
import net.sourceforge.pinyin4j.format.HanyuPinyinToneType;
import net.sourceforge.pinyin4j.format.exception.BadHanyuPinyinOutputFormatCombination;

public class Test {

    private String pinyinName;//属性部分

    /**汉字转拼音的方法*/
    private String HanyuToPinyin(String name){//方法部分
        pinyinName="";
        char[] nameChar = name.toCharArray();
        HanyuPinyinOutputFormat defaultFormat =
            new HanyuPinyinOutputFormat();
        defaultFormat.setCaseType(HanyuPinyinCaseType.LOWERCASE);
        defaultFormat.setToneType(HanyuPinyinToneType.WITHOUT_TONE);
        for (int i = 0; i < nameChar.length; i++) {
            if (nameChar[i] > 128) {
                try {
                    pinyinName += PinyinHelper.toHanyuPinyinStringArray(nameChar[i], defaultFormat)[0];
                } catch (BadHanyuPinyinOutputFormatCombination e) {
                    e.printStackTrace();
                }
            }
        }
        return pinyinName;
    }

    public static void main(String[] args) {//实现部分
        System.out.println(new Test().HanyuToPinyin("测试中文转拼音"));
    }
}
```

图 2-109

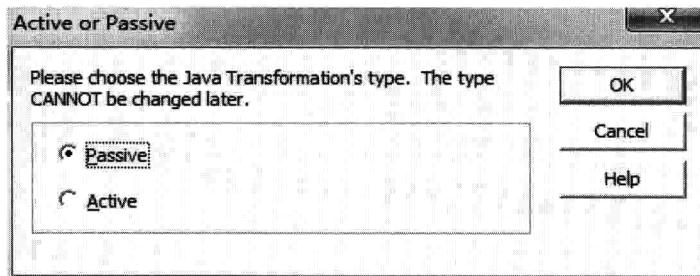


图 2-110

此处选择 Passive 组件，单击 OK 按钮（组件类型选定后，不可修改）。

(3) Java Transformation 组件中的输入/输出项添加。

双击 Java 组件，选择 Port 项，即可新建输入/输出项，如图 2-111 所示。

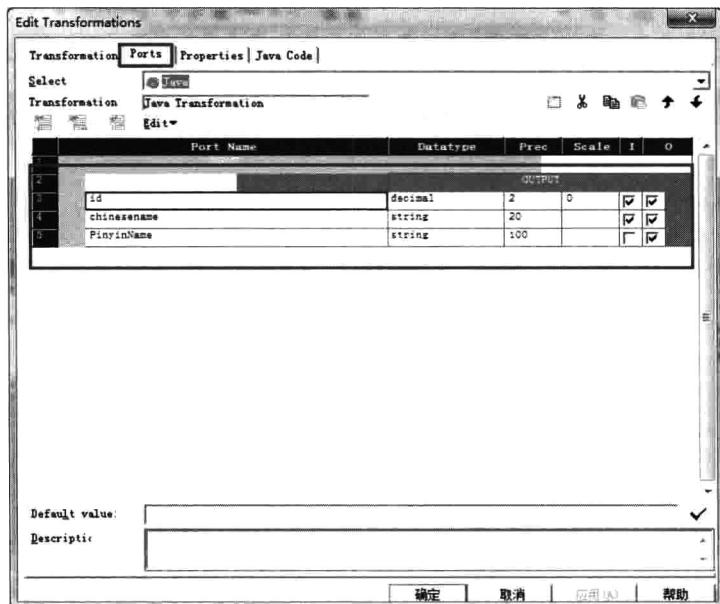


图 2-111

(4) 选择 Java Code 进入 Java 代码编辑页面, 如图 2-112 所示。

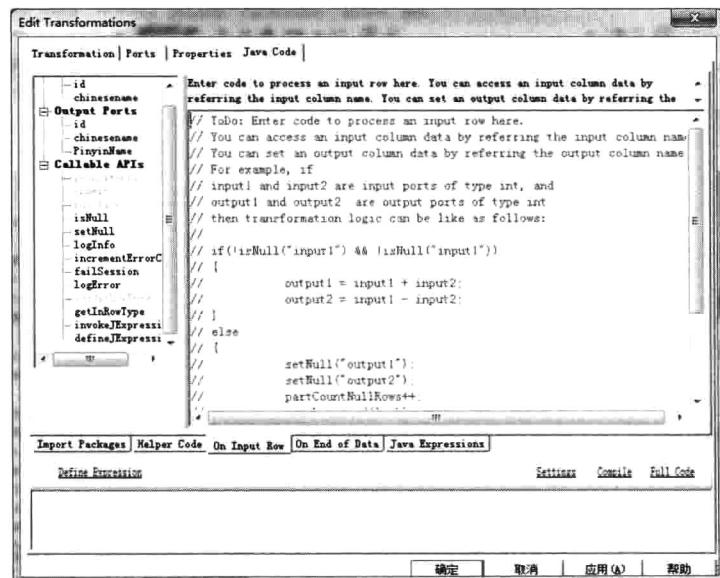


图 2-112

- Import Packages: 在此标签页中, 可以将 Java 类中的 Import 部分写入, 如图 2-113 所示。

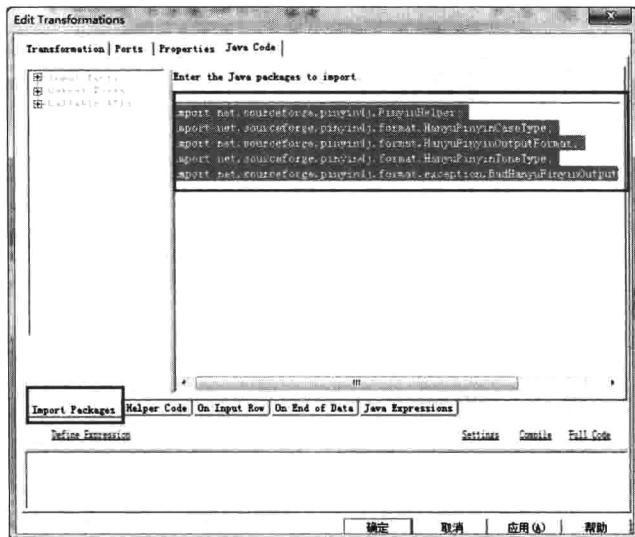


图 2-113

- Helper Code: 在此标签页中, 可以定义 Java 类型的变量, 如图 2-114 所示。

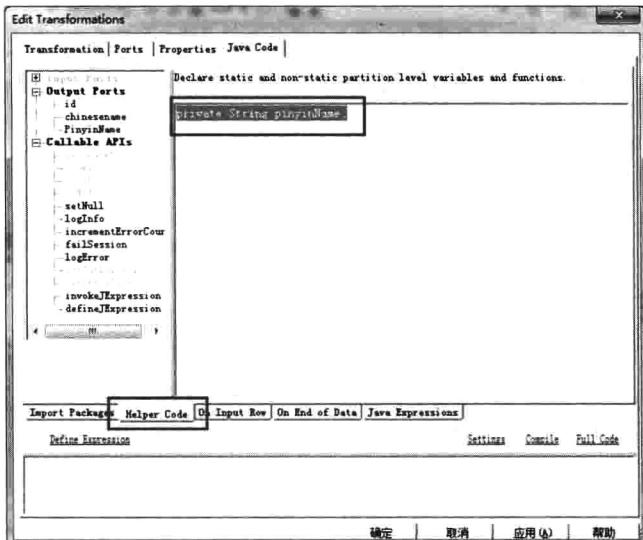


图 2-114

- Java Expressions: 在此标签页中，可以编写 Java 代码中的一些 Method，即方法，如图 2-115 所示。

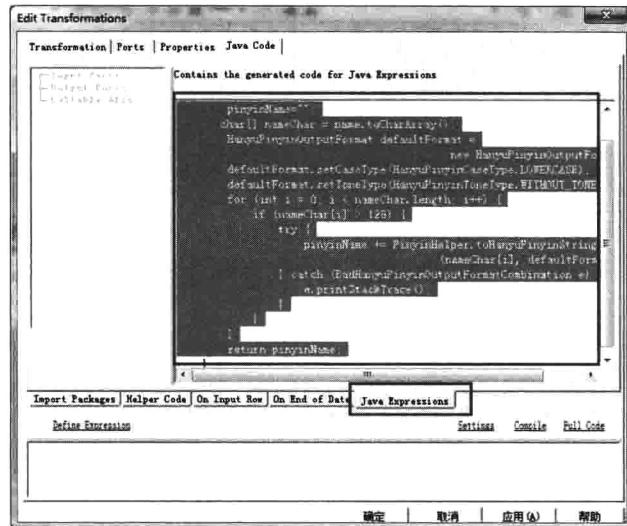


图 2-115

- On Input Row: 在该标签页中，可以编写本次业务转换的实现，如图 2-116 所示。

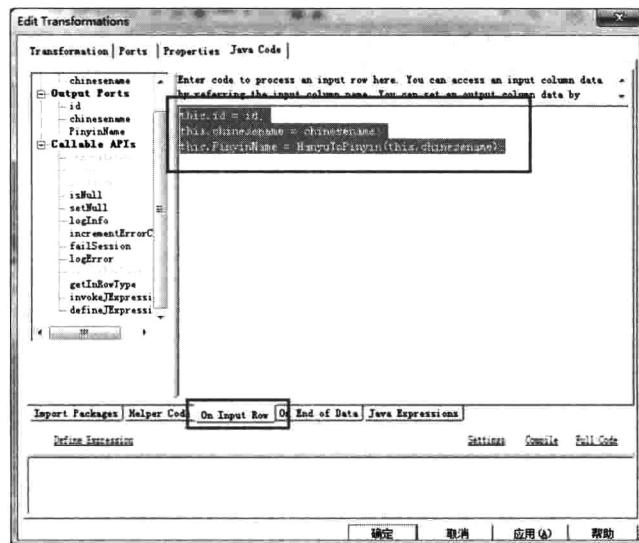


图 2-116

**注意：**编写输入项时，需要添加 this.outputPort。

- ◎ **Settings：**单击 Settings 链接后，在弹出的对话框中单击 Browse 按钮，选择客户端本地的.jar 文件，然后单击 Add 按钮，即可添加 PowerCenter 客户端，用于 Java 代码编译，如图 2-117 所示。

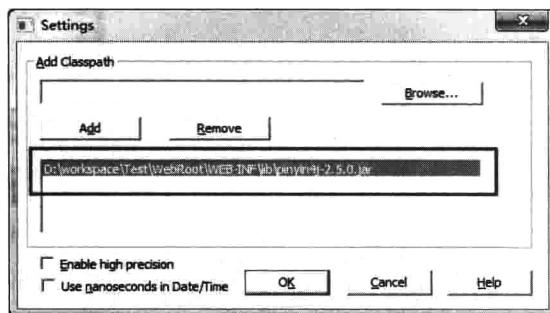


图 2-117

- ◎ **Compile：**Java 编译。Java 代码编写与.jar 文件导入完成后，单击 Compile 链接测试 Java 代码的编译正确性。如果 Java 编译无误，窗口下方显示“Java code compilation successful.”，如图 2-118 所示。

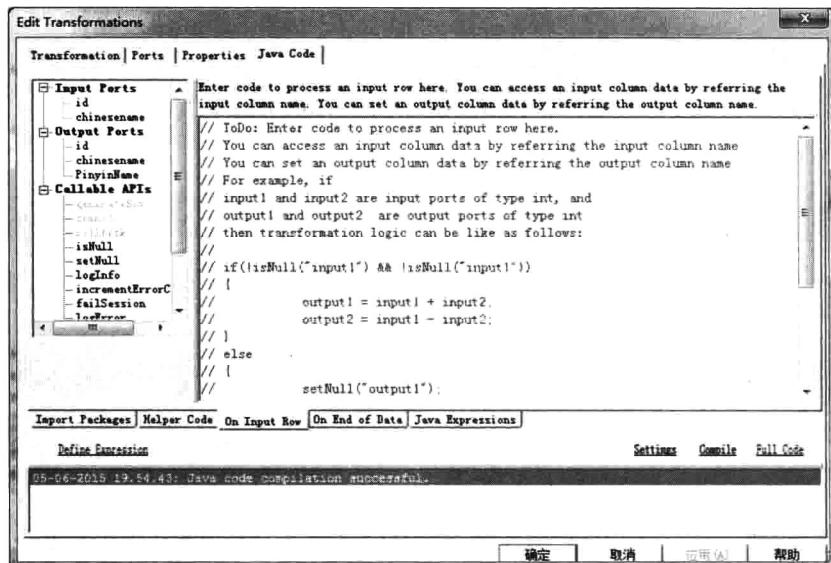
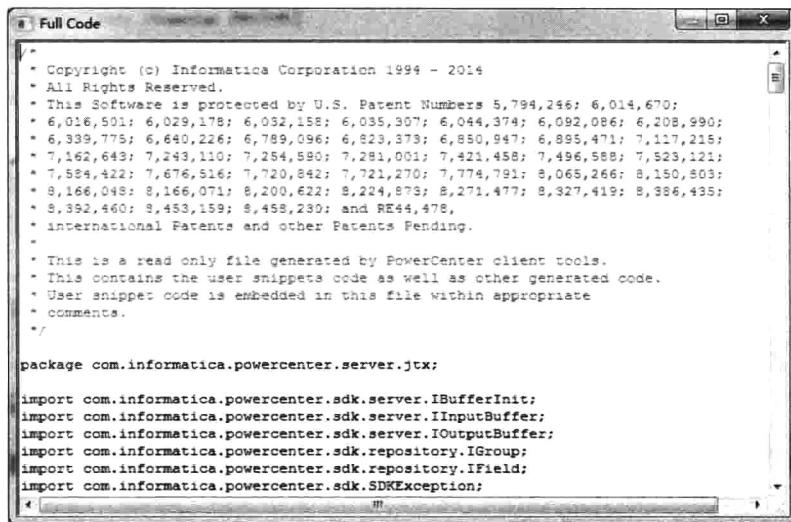


图 2-118

如果 Java 编译有误，则会在窗口的下方显示编译错误信息。

- Full Code：完整代码查看。单击 Full Code 链接后，可以查看 Java Transformation 生成的完整代码，如图 2-119 所示。



```

/*
 * Copyright (c) Informatica Corporation 1994 - 2014
 * All Rights Reserved.
 * This Software is protected by U.S. Patent Numbers 5,794,246; 6,014,670;
 * 6,016,501; 6,029,178; 6,032,158; 6,035,307; 6,044,374; 6,092,086; 6,208,990;
 * 6,339,775; 6,640,226; 6,789,056; 6,823,373; 6,850,947; 6,895,471; 7,117,215;
 * 7,162,643; 7,243,110; 7,254,580; 7,281,001; 7,421,458; 7,496,588; 7,523,121;
 * 7,584,422; 7,676,516; 7,720,842; 7,721,270; 7,774,791; 8,065,266; 8,150,803;
 * 8,166,049; 8,166,071; 8,200,622; 8,224,873; 8,271,477; 8,327,419; 8,386,435;
 * 8,392,460; 8,453,159; 8,458,230; and RE44,478;
 * international Patents and other Patents Pending.
 *
 * This is a read only file generated by PowerCenter client tools.
 * This contains the user snippets code as well as other generated code.
 * User snippet code is embedded in this file within appropriate
 * comments.
 */

package com.informatica.powercenter.server.jtx;

import com.informatica.powercenter.sdk.server.IBufferInit;
import com.informatica.powercenter.sdk.server.IInputBuffer;
import com.informatica.powercenter.sdk.server.IOutputBuffer;
import com.informatica.powercenter.sdk.repository.IGroup;
import com.informatica.powercenter.sdk.repository.IField;
import com.informatica.powercenter.sdk.SDKException;

```

图 2-119

此时，Java Transformation 编辑完成。

- (5) 继续完成 Mapping 设计，创建 Session、Workflow，配置数据源、目标等，如图 2-120 所示。

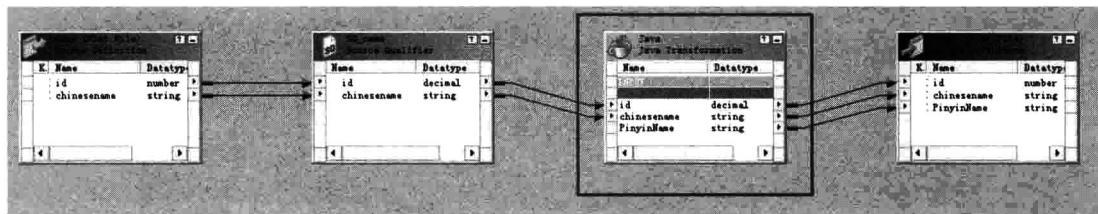


图 2-120

- (6) 将 Java 代码中依赖的.jar 包上传到 PowerCenter 服务器，默认目录为\$INFA\_HOME/server/bin/javalib。同时，也可以在 Session→Properties→Java Classpath 中配置.jar 文件的目录，如图 2-121 所示。

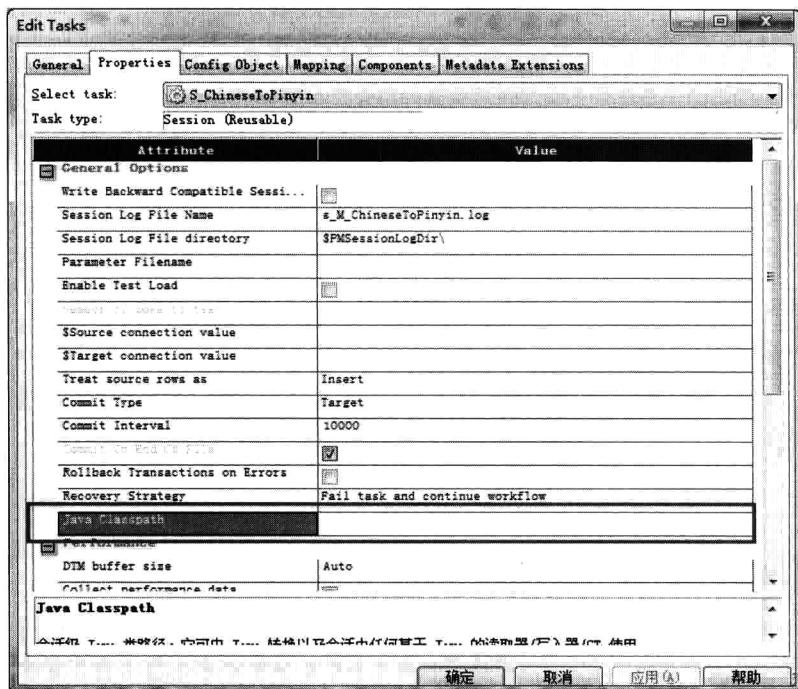


图 2-121

(7) 运行测试，查看执行结果。

### 2.17.3 Active Java Transformation

场景：行列转换，如图 2-122 所示。

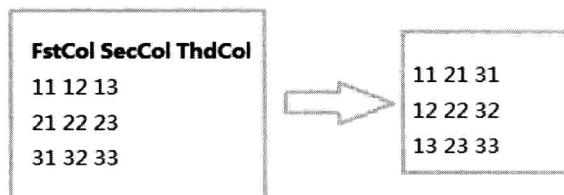


图 2-122

利用 Java Transformation 实现该功能，与 2.17.2 节的开发步骤基本一致，差异的部分如下。

(1) 创建 Java Transformation 时要选择 Active 组件，如图 2-123 所示。

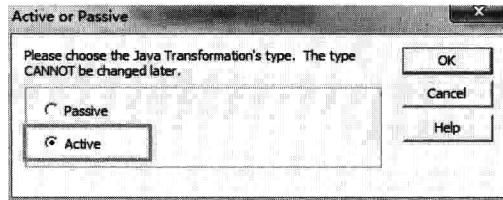


图 2-123

(2) 输入/输出项不可相同，如图 2-124 所示。

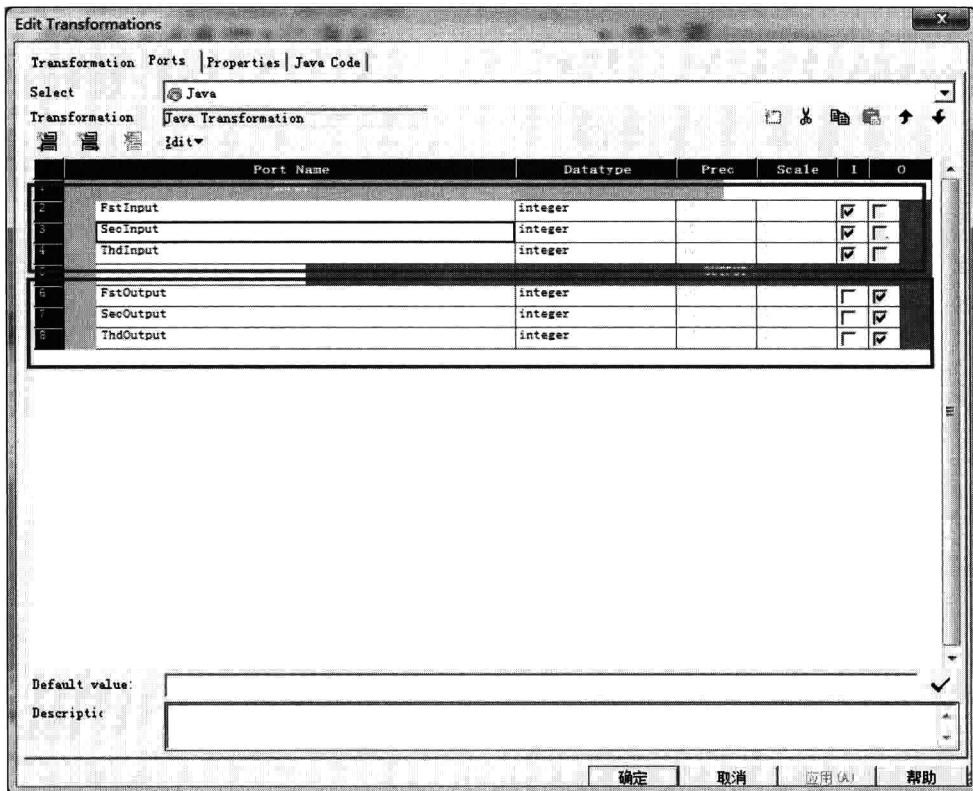


图 2-124

(3) 编写代码部分与上一部分相同，差异部分如图 2-125 方框部分所示。

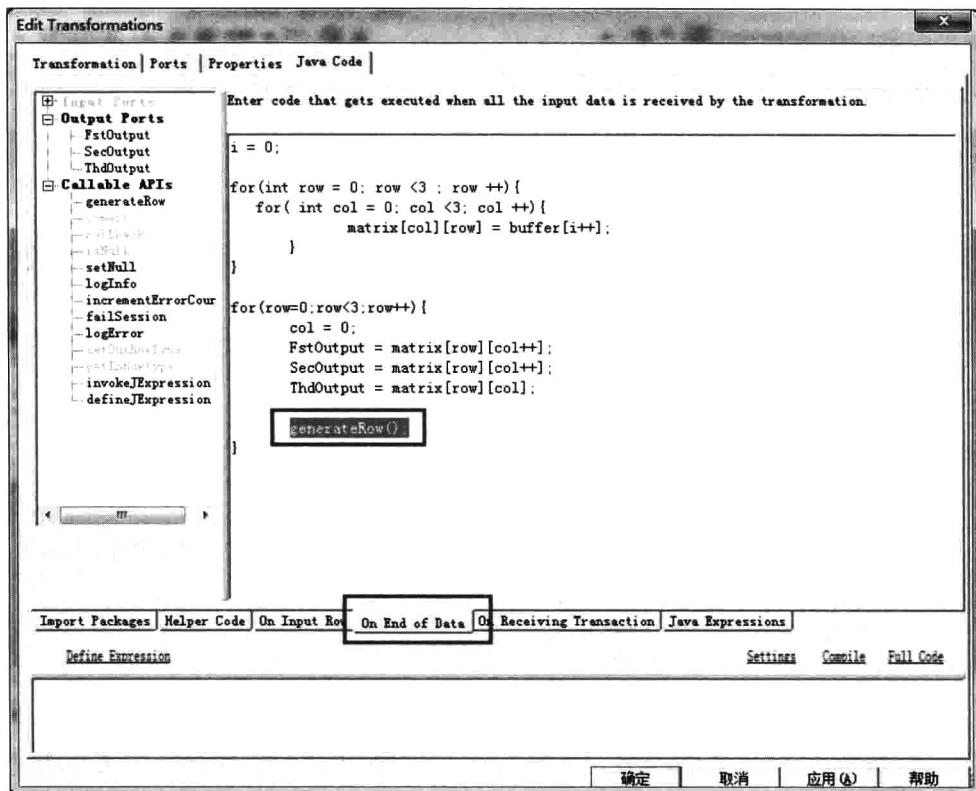


图 2-125

在 On End of Data 部分，编写输出项的生成。

利用 generateRow()方法生成输出项的一行。

#### 2.17.4 常见错误说明

(1) 编译 Java 时，出现如图 2-126 所示的错误。

```

JTXPartitionDriverImplGen.java:36: 软件包 net.sourceforge.pinyin4j 不存在
import net.sourceforge.pinyin4j.PinyinHelper;
^
JTXPartitionDriverImplGen.java:37: 软件包 net.sourceforge.pinyin4j.format 不存在

```

图 2-126

**解决方案：**单击 Settings 链接，将 Java 依赖包导入 PowerCenter 客户端。

(2) Workflow 执行时，出现“无法加载类”错误，如图 2-127 所示。

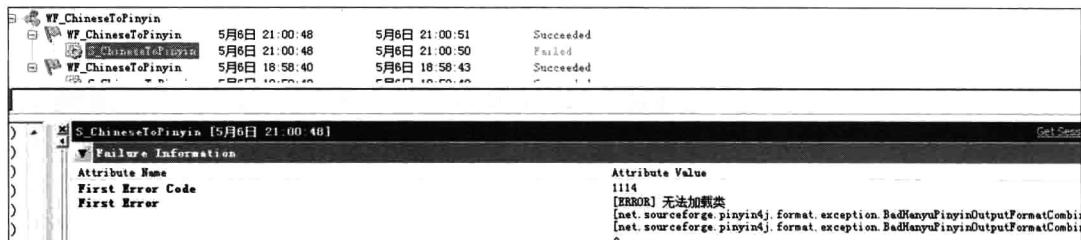


图 2-127

**解决方案：**将依赖包上传到\$INFA\_HOME/server/bin/javalib 目录下。

## 2.18 Normalizer

Normalizer 在 PowerCenter 中使用的图标是

Normalizer 在实际中最常用的就是将列表转换为行表。因此，仍然以 Aggregator 中使用的两张表 TAB\_SAL 和 TAB\_SAL\_DW 做一个反向处理的样例，回顾一下这两张表中的数据。

源表：TAB\_SAL\_DW，如图 2-128 所示。

SQL> select * from tab_sal_dw;					
EMPNO	BASE_SAL	PERFORMANCE_SAL	LEVEL_SAL	OTHER_SAL	
2	10000	1000	100	10	
1	1000	100	10	1	

图 2-128

目标表：TAB\_SAL，期望的目标表中的数据如表 2-19 所示。

表 2-19

EMPNO	INCOME_TYPE	INCOME_AMOUNT
1	11	1000
1	22	100
1	33	10

续表

EMPNO	INCOME_TYPE	INCOME_AMOUNT
1	44	1
2	11	10000
2	22	1000
2	33	100
2	44	10

实现这个需求，完成 Mapping 的步骤大致如下。

第一步：建立一个 Mapping 如图 2-129 所示，在 Mapping 中使用了 Normalizer 和 Expression 组件。Normalizer 组件用于实现从列表到行表的转换，Expression 组件用于支持收入类型的编码的转换。



图 2-129

第二步：重点关注 Normalizer 组件。在 Normalizer 组件的 Normalizer Tab 中增加两列 EMPNO 和 SAL\_AMOUNT，并设置 Occurs 分别为 0 和 4，如图 2-130 所示。

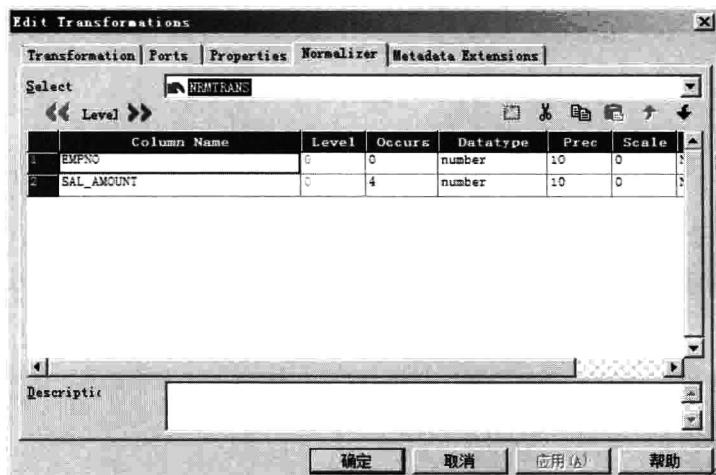


图 2-130

第三步：单击“确定”按钮后，在 Normalizer 中产生了 EMPNO\_in 和 4 个 AMOUNT 字段作为输入，将 TAB\_SAL\_DW 的 Source Qualifier 中的字段分别对应连接，如图 2-131 所示。

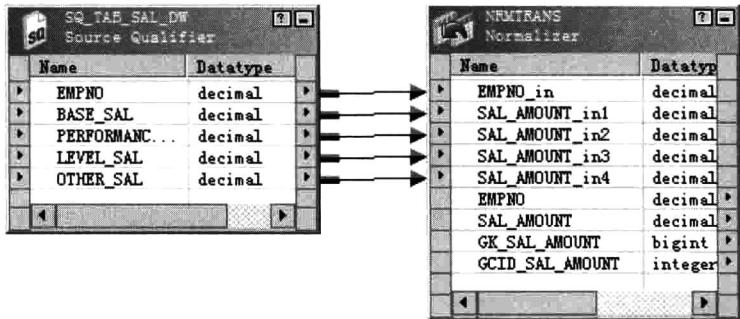


图 2-131

分别介绍一下输出字段：在输出中 EMPNO 就是 EMPNO\_in；SAL\_AMOUNT 是源表展开后的字段；GK\_SAL\_AOMOUNT 是一个序列号，从 1,2,3,4,5,6... 一直增长；GCID\_SAL\_AMOUNT 为 1、2、3、4 重复进行。

如何将 GCID\_SAL\_AMOUNT 的 1、2、3、4 转换为 INCOME\_TYPE 的 11、22、33、44 呢？这个要求是通过在 Normalizer 后面的 Expression 组件实现的。在 Expression 组件中增加如下表达式：

```
DECODE(GCID_SAL_AMOUNT,
1,11,
2,22,
3,33,
4,44)
```

在上面的例子中介绍 Normalizer 组件仅仅介绍了 Occures，并没有介绍 Level。Level 在数据源为 VSAM\COBOL 时会用到，现在使用的场景并不多，因此不再用更多的篇幅进行介绍。

## 2.19 Router

A 公司是一家区域分布的全国性的公司，公司内部已经进行了核心交易系统的集中。

基于各地业务的巨大差异性，分析型系统仍然由各家分公司独立建设、运行，因此需要将总部的信息分发到各地的接口机和数据库上。A公司的IT开发人员在PowerCenter中找到一个数据分发组件，这个组件就是Router。

Router组件在PowerCenter中使用的图标是。

按照公司的业务要求，开发人员开发了如图2-132所示的Mapping。

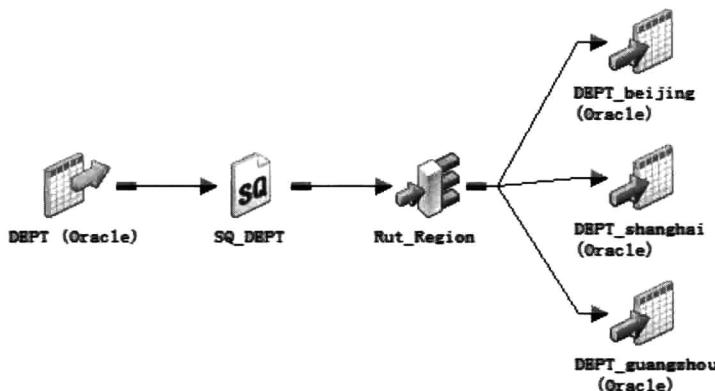


图 2-132

在这个Mapping中重点的组件就是Router组件。双击Router组件，选择Groups Tab增加分组，例如这里增加3个组——Beijing、Shanghai、Guangzhou，并分别设置Group Filter Condition，如图2-133所示。



图 2-133

Beijing 的 Group Filter Condition 是 LOC='Beijing'; Shanghai 的是 LOC='Shanghai'; Guangzhou 的是 LOC='Guangzhou'。

→ **注释**

当每一条数据到达 Router 组件后，这条数据将与所有的 Group Filter Condition 匹配，匹配成功后，数据将向该组的下游传递。这就意味着根据设置的 Group Filter Condition 的不同，有可能一条数据会同时传递给多个组的下游。

## 2.20 Custom Transformation

Custom Transformation 在 PowerCenter 中使用的图标是 。

Custom Transformation 为 PowerCenter 提供了一种扩展能力。例如，一个项目需要的加密方法在 PowerCenter 中没有提供；正在使用 PowerCenter 的集成商提供了一个新的分词算法，对新闻稿件的分词特别有效，等等。这些都可以作为 PowerCenter 的 Custom Transformation，将这些功能与 PowerCenter 的其他能力进行整合。这样整合可以充分利用 PowerCenter 的接口能力，可以利用 PowerCenter 的上下游数据处理能力，还可以利用 PowerCenter 的集群、分区能力等。

在现有的 PowerCenter 提供的 Transformation 中，部分是 Native Transformation，如 Aggregator、Expression、External Procedure、Filter、Joiner、Lookup、Normalizer、Rank、Router、Sequence Generator、Sorter、Source Qualifier、Stored Procedure、Transaction Control、Update Strategy；其他 Transformation 也是通过 Custom Transformation 进行开发的。

Custom Transformation 可以使用 C/C++、Java 等语言进行开发，它可以是 Active 或者 Passive Transformation，这是在创建 Custom Transformation 时选择确定的，如图 2-134 所示。

笔者并没有进行过 Custom Transformation 的开发，可能无法对这部分进行更详细的描述。如果确有需求进行 Custom Transformation 的开发，请参考 Custom Transformation 开发手册，里面提供了对 Custom Transformation API 接口的详尽描述。

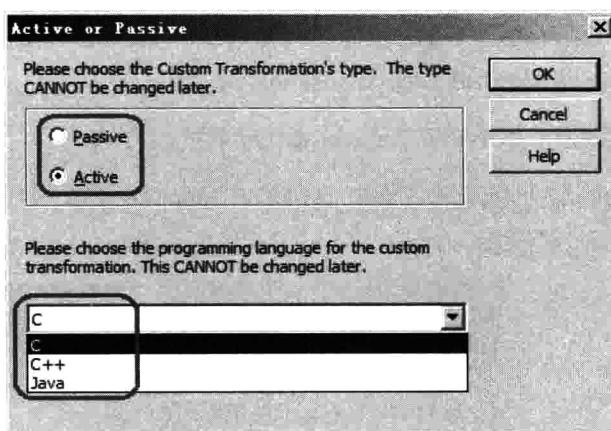


图 2-134

## 2.21 HTTP Transformation

HTTP Transformation 在 PowerCenter 中使用的图标是 。

HTTP Transformation 为 PowerCenter 开发人员提供了可以通过 HTTP 协议连接到 HTTP Server 使用其服务和应用的能力。它提供的能力或支持的功能包括：

- ① 从 HTTP Server 获取数据；
- ② 更新 HTTP Server 上的数据。

提供的授权方式包括：

- ① Basic——基于非加密的用户名和密码进行授权；
- ② Digest——基于加密的用户名和密码进行授权；
- ③ NTLM——基于加密的用户名、密码和 Domain 进行授权。

作为大数据时代的数据从业人员，大家偶尔也会投入一两万元炒股，因此希望能够获取一些基础数据进行比大智慧更有针对性的分析。同时，大家也会买一些货币基金，并积极关注其每天收益。因此每天都要搜索一下该基金的收益，希望能获得最新的行情数据。

下面就以货币基金为例，开发一个通过 HTTP Transformation 获得行情数据的小程序。

如货币基金 003003，可以使用 URL 可以访问其相关数据。这个 URL 就是 <http://www.howbuy.com/fund/003003/index.htm?source=aladdin>。

通过下面的开发过程演示 HTTP Transformation 的使用方法。

第一步：创建 Mapping 并将其命名为 m\_HTTP，在 Mapping 中添加 HTTP Transformation，完成的 Mapping 如图 2-135 所示。

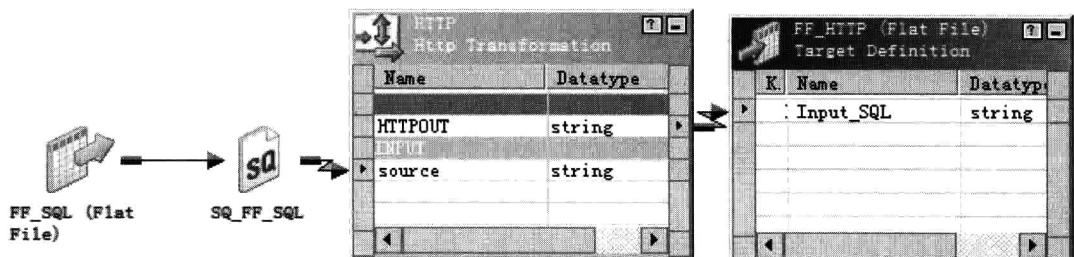


图 2-135

第二步：关键点是编辑 HTTP Transformation。双击 HTTP Transformation，选择 HTTP Tab，如图 2-136 所示。

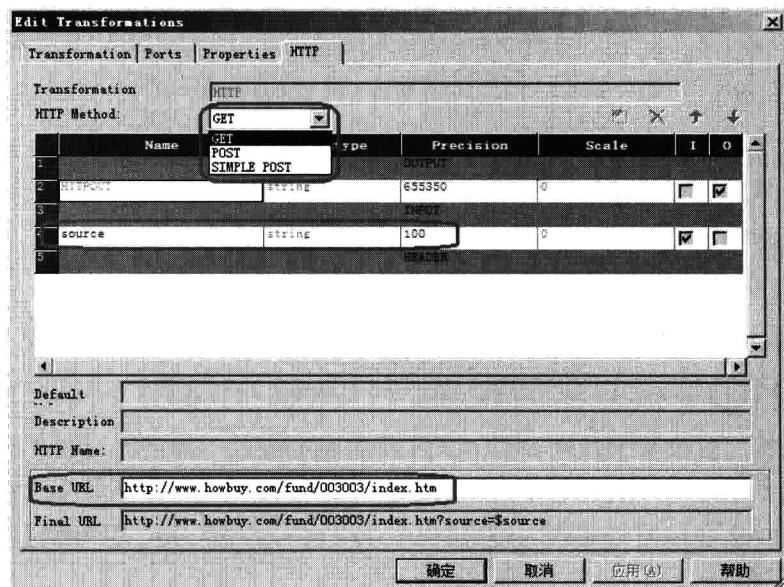


图 2-136

首先选择 HTTP Method。HTTP Transformation 提供了 3 种 HTTP Method, 分别是 GET、POST 和 SIMPLE POST。

这个 Mapping 的目标是获取行情数据, 显然需要选择 GET 方法。回顾一下前面提到的 URL:

```
http://www.howbuy.com/fund/003003/index.htm?source=aladdin
```

斜体部分设置为 URL BASE。为 HTTP Transformation 添加一个端口, 命名为 Source。这时在 Final URL 部分显示的信息如下:

```
http://www.howbuy.com/fund/003003/index.htm?source=$source
```

\$source 为上游传递的参数, 在本例中, \$source 来自上游的文件。

第三步: 为 Mapping m\_HTTP 创建 Workflow。在 Session 中, 可以为其指定 Connection 参数, 也可以保持默认值, 这时将使用在 HTTP Transformation 中指定的 URL。

当访问 HTTP Server 需要认证信息时, 则需要为 Session 指定相关的连接信息。创建一个 HTTP Connection, 在 Workflow Manager 中选择菜单 Connection→Application→New→HTTP Transformation 命令, 出现如图 2-137 所示的对话框, 在这里可以指定访问 HTTP Server 认证所需的信息。

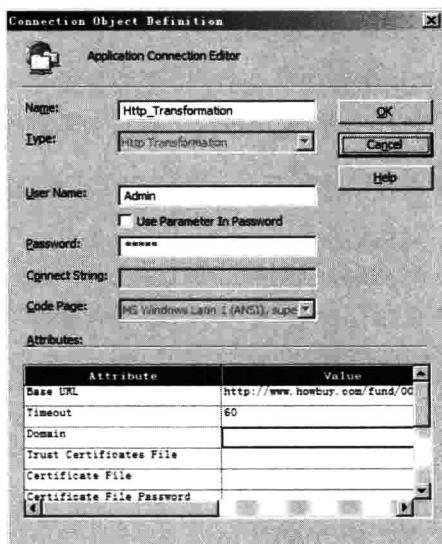


图 2-137

## 2.22 XML 组件组

之所以称为 XML 组件组, 是因为在 PowerCenter 中有 3 个常用的 XML Transformation, 如下。

- ◎ **XML Source Qualifier:** 当数据源是 XML 文本文件时, PowerCener 需要读取其数据进行相关的处理。
- ◎ **XML Parser:** 当 PowerCenter 想通过某些服务, 如 MQ、Web Service 和 UDO 组件获取的数据是 XML 时, 需要通过 XML Parser 组件进行处理。
- ◎ **XML Generator:** 当需要生成 XML 文件作为 Target 时, 要使用 XML Generator。

在处理 XML 数据时, 需要首先了解 XML 文件的结构。因此最重要的一步是获取 XML Schema 文件, 这类文件一般是一个扩展名为.xsd 的文件。假如无法获取 XML Schema 文件, 也可以使用一个扩展名为.xml 的数据文件样例。但要求该 XML 包含足够的数据元素可以描述 XML Schema。使用 XML 数据文件样例有一定的风险, 主要是因为它不一定包含 XML 所有元素的数据, 但可以作为一种变通的方法。

XML 处理涉及的范围很广泛, 本节很难充分描述这些组件的全部功能。下面仅仅以一个实例为样例, 帮助用户初步掌握 XML 的开发。

**需求:** ABC 公司经常收到供应商的一些数据, 这些数据是一些 XML 格式的文件, ABC 公司的开发人员希望将这些 XML 文件进行解析, 并写入数据库中。

同时, ABC 公司的开发人员已经从开发商处获取了一个 XML Schema 文件, 如下:

```
<?xml version="1.0" encoding="windows-1252"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault=
"unqualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Invoice">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Period_Ending" type="xs:string" />
                <xs:element name="Current_Total" type="xs:float" />
                <xs:element name="Balance_Due" type="xs:float" />
                <xs:element minOccurs="0" maxOccurs="unbounded" name="Buyer">
```

```

<xs:complexType>
  <xss:sequence>
    <xss:element minOccurs="0" maxOccurs="unbounded" name="Transaction">
      <xss:complexType>
        <xss:sequence>
          <xss:element name="Product" type="xs:string" />
          <xss:element name="Total" type="xs:float" />
        </xss:sequence>
        <xss:attribute name="date" type="xs:string" use="required" />
        <xss:attribute name="ref" type="xs:string" use="optional" />
      </xss:complexType>
    </xss:element>
  </xss:sequence>
  <xss:attribute name="name" type="xs:string" use="required" />
  <xss:attribute name="total" type="xs:float" use="optional" />
</xss:complexType>
</xss:element>
</xss:sequence>
<xss:attribute name="account" type="xs:string" use="required" />
</xss:complexType>
</xss:element>
</xss:schema>

```

具体的开发过程如下：

在 PowerCenter Designer 中选择菜单 Sources→Import XML Definition 命令，可以看到如图 2-138 所示的界面，重点关注一下 Hierarchy Relationships 的 Normalized XML Views 和 Denormalized XML Views 选项。

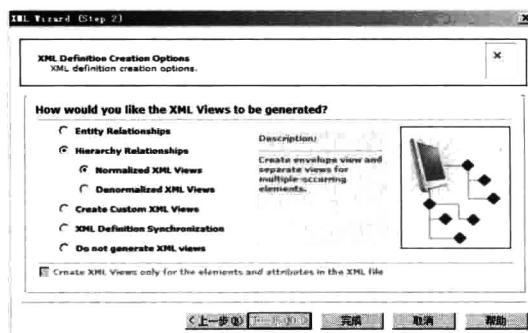


图 2-138

通过这两种方式导入如上的 XML Schema 后的两种视图如图 2-139 所示。

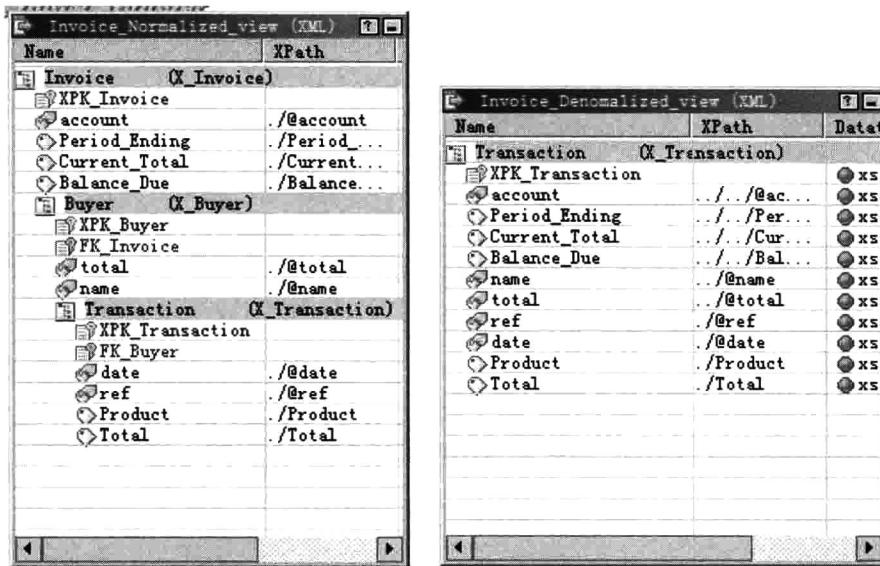


图 2-139

在左边的视图中，还保持了原有的 xsd 层次结构。可以非常清晰地看到这个账单 (Invoice) 结构包括若干的客户 (Buyer)，每个客户做了若干笔交易 (Transaction)。这是一个典型的采购单，分别描述了两个客户的采购情况，结构比较清晰。

如果开发人员希望把这些数据写入数据库，并已经按照三范式设计了数据库的模型 (表)，例如 TAB\_invoice、TAB\_Buyer 和 Tab\_Transaction，可以直接将数据写入这 3 张表。这 3 张表的主外键关系可以直接通过 XML 中的 XPK\_Invoice 和 PK\_Invoice 等实现；也可以通过将这些字段二次转换后形成新的键值，并保证其关联关系。

如果开发人员希望把这些数据进行转换后，生成新的 XML 格式的文件继续与其他系统进行交换，还可以组合使用 PowerCenter 的其他转换组件和 XML Generator。

在右边的视图中，已经将 xsd 进行了反规范化。如果说左边的结果符合 IT 人员的使用习惯，右边的结果更适合业务人员使用数据的习惯。虽然它的输出中包含了一些重复数据，但是对使用诸如 Microsoft Excel 分析工具的人来说非常方便。与左边的视图一样，PowerCenter 也支持将这些数据继续进行加工，并写入数据库或者期望的文本文件。

不管是 XML Source Qualifier，还是 XML Parser，都可以支持相同的功能。XML Source Qualifier 和 XML Parser 的最大不同在于其数据来源不同，XML Source Qualifier 主要用于直接读取 XML 文件，而 XML Parser 可以读取其他组件的输出，例如 MQ、Web Service、交易中间件或者一个 CLOB 字段等。

## 2.23 Transformation 中的一些概念

PowerCenter Designer 中的主要组件（Transformation）在前面的小节都进行了比较详细的描述，也曾经提到过一些关于 Transformation 的概念，比如 Connect 和 Unconnect、Active 和 Passive 等。为了帮助大家理解，本节针对这几个概念在一个集中的位置做一些更详尽的描述。

### 2.23.1 Connect 与 Unconnect

Connect 与 Unconnect 是关于 Transformation 的一个特性，它们是一组相对的概念。二者之间的不同在于，在 Mapping 中 Connect Transformation 与其他组件直接通过连线进行连接；Unconnect Transformation 在 Mapping 中不与其他组件连接，而是在 Expression 表达式中通过 LKP.xxx 或者 SP.xxx 进行调用。典型的 Unconnect Transformation 如 Unconnect Lookup 和 Unconnect Stored Procedure。在图 2-140 中，Expression 组件是 Connect Transformation，而 Lkp\_Dept 是一个 Unconnect Transformation 组件。

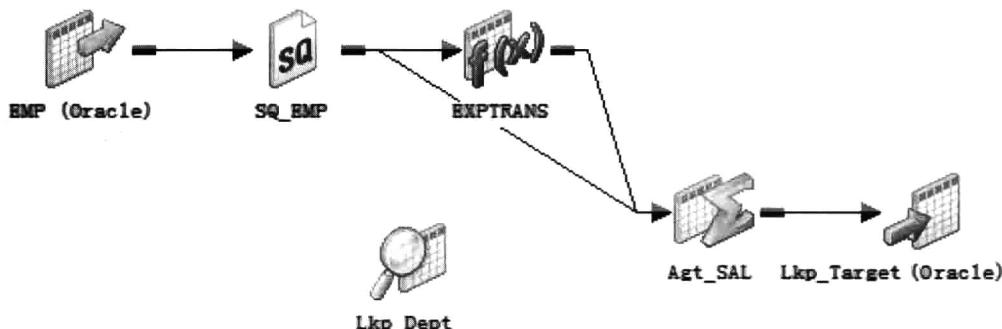


图 2-140

### 2.23.2 Active 与 Passive

在 Mapping 中的每一个 Transformation 都可以看成一个水渠的截面，当数据流过 Transformation 后会发生变化，比如颜色会加深、水量会变少等。在 PowerCenter 中这些变化有个关键性指标，就是水量，对应到数据就是数据量的变化，如图 2-141 所示。



图 2-141

有  $n$  条数据进入 Transformation，有  $m$  条数据从 Transformation 中输出。根据  $n$  和  $m$  的大小关系不同，PowerCenter 中提供了两个关于 Transformation 的基本概念：Active Transformation 和 Passive Transformation。在 Mapping 拖线的过程中，这是最困扰开发者的一个地方。

(1) Passive：如果把  $n$  当作流入的数据条数， $m$  作为流出的数据条数，当  $m$  一定等于  $n$  时 ( $m=n$ )，这个 Transformation 就被叫作 Passive Transformation。典型的 Passive Transformation 包括 Expression、Data Masking 等。

(2) Active：如果把  $n$  当作流入的数据条数， $m$  作为流出的数据条数，当  $m$  可能不等于  $n$  时，注意只要有可能  $m \neq n$ ，这个 Transformation 就被叫作 Active Transformation。典型的 Active Transformation 包括 Joiner、Aggregator 等。早期的 Lookup 是 Passive 组件，后来由于功能的扩展，Lookup 也可以实现类似 Joiner 的功能，因此 Lookup 可以说是 Active 或者 Passive Transformation。

相信很多开发人员对 Active 和 Passive 的第一印象是在完成类似图 2-142 所示的 Mapping 时，试图将 Source Qualifier 中的 EMPNO 与 Target 中的 EMPNO 连接时，系统自动提示“Concatenation disallowed on transformation EMP1. Active transformations (agg\_SUM\_SAL,SQ\_EMP)”，无法完成这个连接操作。

这是因为 Aggregator Transformation 是一个 Active Transformation。在这种情况下，如果确实需要将 Source Qualifier 中的 EMPNO 与 Target 中的 EMPNO 相连，正确的做法是将 EMPNO 先拖入 Aggregator，然后将 Aggregator 中的 EMPNO 与 Target 中的 EMPNO 相连。这时需要开发人员确认逻辑的正确性。

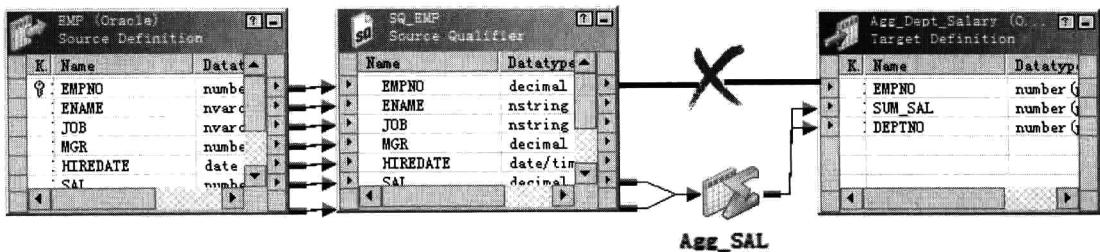


图 2-142

此外，Active Transformation 还是事务的边界，也就是说，事务的有效性以 Active 组件为界。

# 第3章

# Workflow 执行、监控

---

Workflow 是 PowerCenter 的最重要执行单元，一个 Workflow 包括一个或者多个 Session 或者 Task。在 Workflow 中可以定义 Session/Task 执行的前后继顺序，定义 Session/Task 并行或者串行执行，还可以对 Workflow 进行调度，如设定某天、某个时刻执行或者按照某个间隔执行。为了提高 Workflow 管理、调度的灵活性，PowerCenter 提供了大量辅助的 Task，帮助开发人员和用户快速设计需要的 Workflow。常用的 Task 包括如下类型。

- Assignment：为 Workflow 变量进行赋值。
- Command：在 Workflow 中执行一个 Shell 或者命令。
- Control：在 Workflow 中控制 Workflow 状态，如 Stop/Abort 当前的 Workflow。
- Decision：组合条件评价。
- Email：在 Workflow 中发送邮件。
- Event-Raise：通知 Event-Wait 任务某个事件已经发生，触发 Event-Wait 任务。
- Event-Wait：与 Event-Raise 任务搭配使用，等待 Event-Raise 触发的事件。
- Session：Mapping 的实例化，在 Session 中可以为 Mapping 指定相关参数，将其丰富为一个可执行的单元，如指定数据源和目标、指定 Schema（用户）和表名等。Session 提供了大量的参数，以控制、优化其执行，在后续的章节会有详细的介绍。
- Timer：等待一个时间事件触发。

后续的章节将会一一介绍这些 Task。在此之前，首先介绍 Workflow 中最简单、最常用的部分。

## 3.1 Session

Session 是 Workflow 最基础的组成单元，在介绍 Workflow 之前，需要对 Session 有一个清晰的理解。在 Hello World 程序中已经有一部分简单的介绍。我们认为 Session 是 Mapping 的实例化，这个解释使用了纯粹的计算机语言。通俗一点说，Mapping 是一个程序，仅仅描述了业务逻辑，但是对执行来说，它有很多具体的参数我们尚不完全清楚，例如实际的源数据库、目标数据库、字符集等，这些需要在 Session 中进行配置。用这个解释理解 Session 的话，Session 就是给 Mapping 提供配置文件的场所或者机制。有了这些配置信息，Mapping 就可以实际执行了，Session 是 Mapping 的一个可执行实例。因此，为 Mapping 提供不同的参数形成的 Session 也就是 Mapping 的不同实例。

Session 有两种不同的类型：Reusable Session 和非 Reusable Session。创建 Session 有三种不同的方法。在前面的例子中，一直在使用的是 Wizard 方法，这种方法同时创建了 Workflow 和 Session。Wizard 方法适合学习或者交流，但在实际的项目中使用这种方法不一定适合。下面通过介绍 Session 的两种类型，同时帮助读者掌握创建 Session 的另外两种方法。

### 3.1.1 Reusable Session

Reusable Session 顾名思义就是可以被重用的 Session，即创建一次，可以被多次重用。那么如何创建一个 Reusable Session 呢？

如图 3-1 所示，在 Workflow Manager 中选中 Task Developer，选择菜单 Tasks→Create 命令，这时会弹出一个对话框，在这里输入要创建的 Session 的名字即可，如图 3-2 所示。

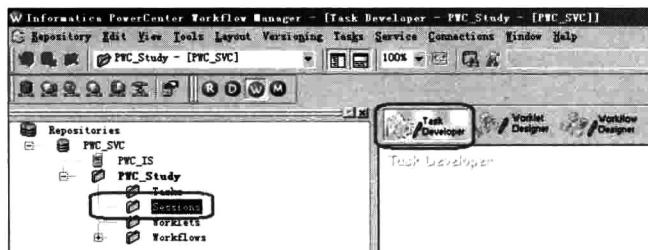


图 3-1

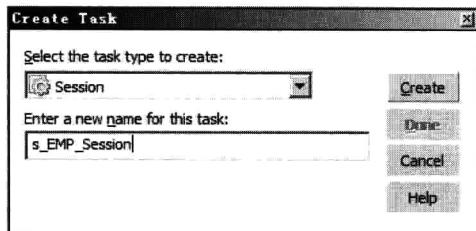


图 3-2

单击 Create 按钮，进入如图 3-3 所示的对话框。

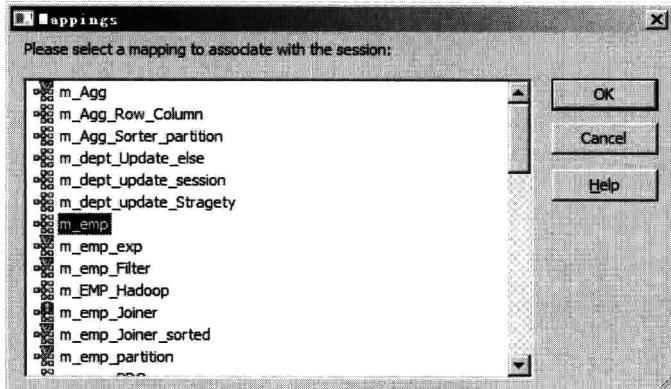


图 3-3

选中要实例化的 Mapping，这些 Mapping 是在 PowerCenter Designer 中开发并成功保存的对象。单击 OK 按钮即可。

#### → 注释

在图 3-3 中看到的是在当前 Folder 中已经创建的 Mapping。我们也会注意到有的 Mapping 头上有黄色的问号或者红色的感叹号，这些 Mapping 都是有警告的或者无效的 Mapping，将无效或者有告警的 Mapping 实例化是没有意义的。这种情况下需要返回到 PowerCenter Designer 中修正存在的问题，然后再进行实例化。

当 Mapping 被创建后，在 Workflow Manage 左侧的树形目录中就可以看到这个 Session，这样创建的 Session 就是 Reusable Session。Reusable Session 可以用于多个 Workflow 或者 Worklet，如图 3-4 所示。

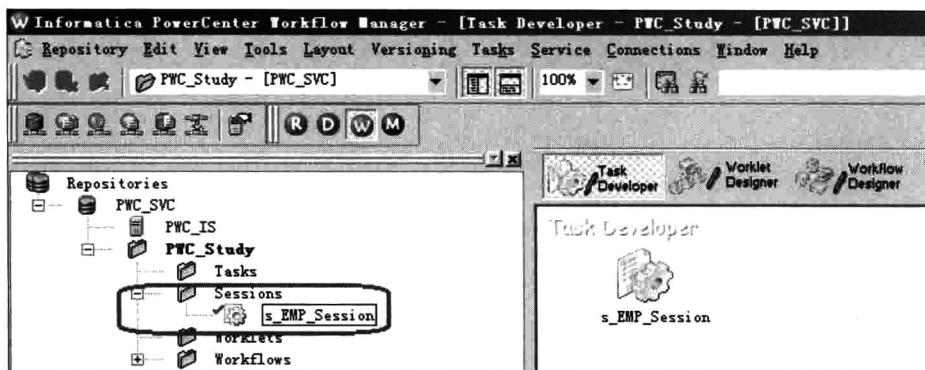


图 3-4

使用 Wizard 创建 Workflow 时也可以创建 Reusable Session, 只是一个创建过程的选项, 不再进行详述。

### 3.1.2 非 Reusable Session

创建一个非 Reusable Session, 需要首先有一个 Workflow。这里也演示一下不通过 Wizard 创建 Workflow 的方式。在 Workflow Manager 客户端中选择 Workflow Designer, 如图 3-5 所示, 然后选择菜单 Workflows→Create 命令, 出现一个创建 Workflow 对话框, 输入要创建的 Workflow 名字, 如 wf\_emp\_demo\_workflow, 单击 OK 按钮即可。

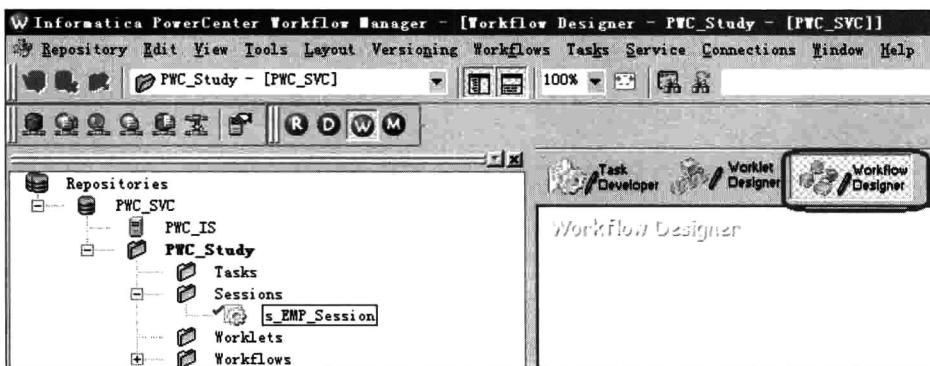


图 3-5

这时创建的 Workflow 如图 3-6 所示。

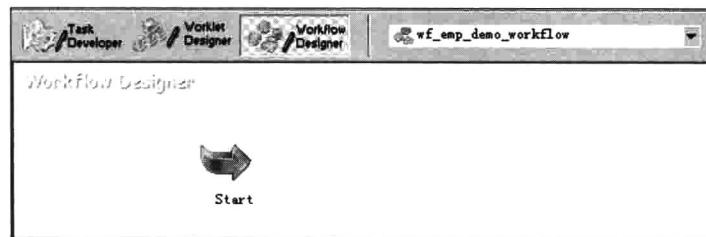


图 3-6

在这个新创建的 Workflow 中仅仅有一个 Start 图标, 它是 Workflow 的一个虚拟起始点, 所有的 Workflow 启动都是从这样一个虚拟起始点开始的。这时就可以开始创建非 Reusable Session 了。创建过程与 Reusable Session 完全一样, 不同的是这个操作是在 Workflow Designer 的 Workbench 中完成的。

新增了非 Reusable Session 的 Workflow 如图 3-7 所示, 这时需要用一个组件将 Start 和新建的 Session 关联起来。

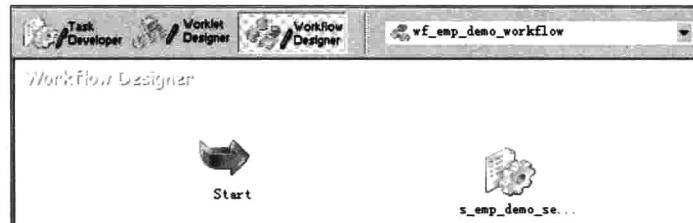


图 3-7

在工具栏中单击 Link Tasks 按钮, 将 Start 与新建的 Session 关联起来即可, 如图 3-8 所示。



图 3-8

完成后的 Workflow 如图 3-9 所示。



图 3-9

在 Workflow 中 Reusable Session 和非 Reusable Session 使用的图标有所差异，如图 3-10 所示。

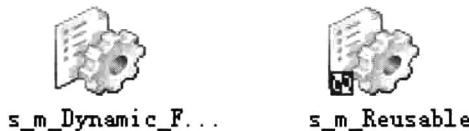


图 3-10

左侧是一个非 Reusable Session，而右侧是一个 Reusable Session。

占用 Session 的篇幅介绍了一下 Workflow 的创建过程，希望读者能在这个小节获得更多的信息量。

非 Reusable Session 可以转换为 Reusable Session，反向则不行。原因也比较简单，Reusable Session 可能会被其他的 Workflow 引用，如果被引用的 Reusable Session 变成非 Reusable Session，那些引用了这个 Session 的 Workflow 将变得非常混乱，因此这个操作是不被支持的。那么如何将一个非 Reusable Session 变成 Reusable Session 呢？双击 Session，在 Session 的 General Tab 中有个选项 Make Reusable，选中即可。

## 3.2 最简单、最常用的 Workflow

在 Workflow 管理中最简单、最常用的就是并行执行、串行执行及调度。这是最简单，也是最实际的。为了帮助初学者快速掌握这些内容，本节将对其进行简单的介绍。

### 3.2.1 并行执行

例如，开发人员已经开发了 3 个 Mapping 并将它们实例化为了 3 个 Session，并且这 3 个 Session 没有依赖关系，为了达到最高的性能，希望这 3 个 Session 能够并行执行。

设计这样的 Workflow 非常简单，相信多数读者看一下图 3-11 就足够了。

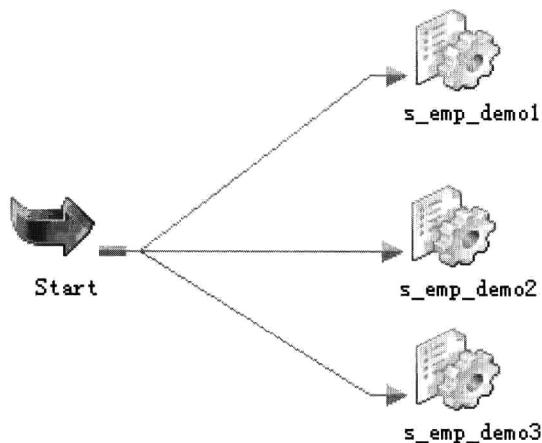


图 3-11

这个 Workflow 的含义是：当启动 Workflow 后，Workflow 从虚拟起始点开始运行，此后将并行启动 3 个 Session。

### 3.2.2 串行执行

现在的项目需求是这样的：开发人员已经开发了 3 个 Mapping 并将它们实例化为了 3 个 Session。这 3 个 Session 存在依赖关系，Session3 只能等 Session2 执行完成后才能执行；Session2 只能等 Session1 执行完成后才能执行。在这种情况下，设计完成后的 Session 应该如图 3-12 所示。

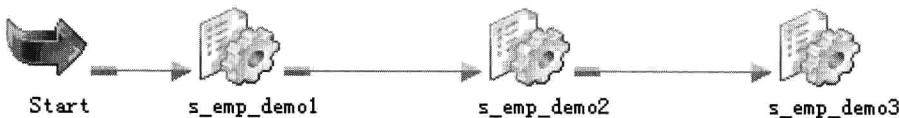


图 3-12

非常简单，首先是虚拟 Start 启动，然后顺序运行 Session1、Session2、Session3。

不过项目组设计完成后，发现一个新问题：在 s\_emp\_demo2 运行之前，s\_emp\_demo1 运行状态必须是成功的，如果在失败的情况下，应该运行一个例外处理 Session。PowerCenter 也可以支持这样的情况，并且操作非常简单。

双击 s\_demo1 和 s\_demo2 之间的连线，这时弹出一个 Expression Editor 对话框，在对话框中添加条件 \$s\_demo1.Status=succeeded，如图 3-13 所示。这个条件的含义是：当 s\_demo1 执行成功后，s\_demo2 才可以执行。

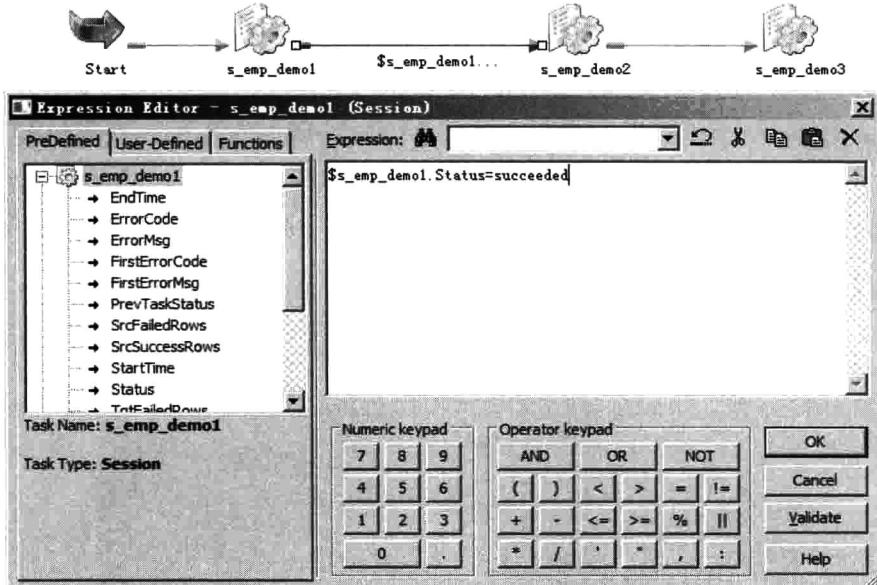


图 3-13

从图 3-13 中可以注意到，在 Link Tasks 中还可以捕捉到的变量很多，包括EndTime、ErrorCode、ErrorMsg、SrcSuccessRows 等，这些都可以当作判断条件使用。

继续对这个 Workflow 进行完善，增加例外控制，完成后的 Workflow 如图 3-14 所示。

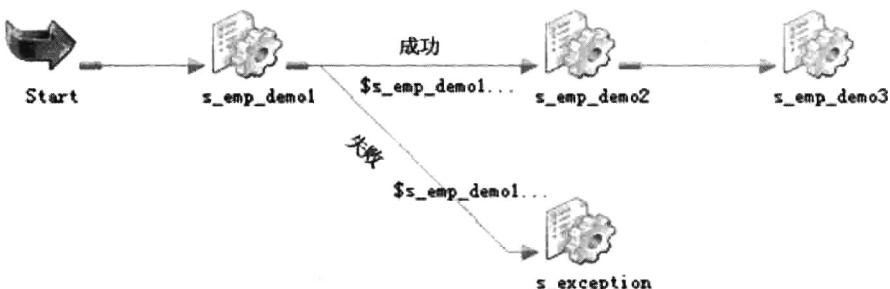


图 3-14

图 3-14 中从 s\_emp\_demo1 到 s\_exception 的连线上设置的条件是 \$s\_emp\_demo1. Status=Failed。它的含义是：当 s\_emp\_demo1 执行成功后，执行 s\_emp\_demo2；当 s\_emp\_demo1 执行失败后，执行 Session s\_exception。

### 3.2.3 调度

PowerCenter 的调度是以 Workflow 为单位的，所以调度的相关属性也是配置在 Workflow 上的。PowerCenter 的调度是藏得最深的一个功能，在 Workflow Designer 中打开一个 Workflow，选择菜单 Workflows→Edit→Scheduler Tab 命令，打开如图 3-15 所示的对话框，选择标注的部分。

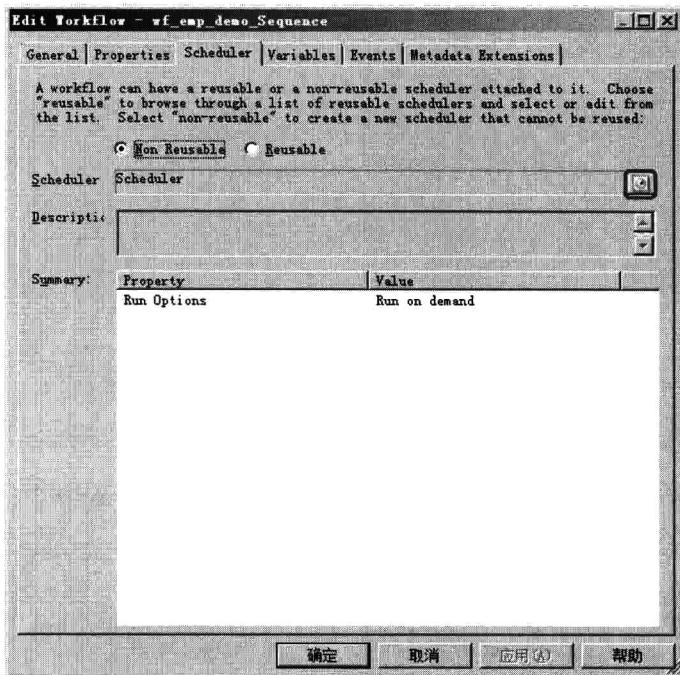


图 3-15

可以注意到，Scheduler 也是可以重用的，这里不再赘述。这时进入一个新的对话框，在 Run Options 中有 3 个选项，如图 3-16 所示。

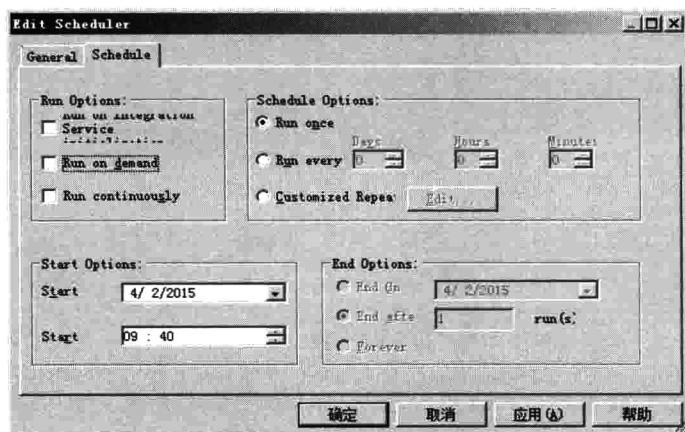


图 3-16

- ① Run On Integration Service Initialization: 在 PowerCenter Integration Service 启动时, Workflow 将启动。有的开发人员发现, 一旦 PowerCenter 重启, 所有的 Workflow 都会运行, 就是无意中使用了这个功能。
- ② Run on demand: Workflow 在请求启动时才启动。如在 Workflow Manager 的图形界面上单击 Start Workflow 或者通过 pmcmd 进行启动。
- ③ Run continuously: Workflow 将不停地运行。当上一个批次运行完后, 重新启动继续运行, 循环不止。

注意, 当设置了启动时间并保存后, PowerCenter Workflow 的设置并没有完全结束。需要通过图形界面执行 Schedule Workflow 或者通过命令执行 Schedule Workflow, 一个 Workflow 才真正被 Schedule 了。

### 3.3 Worklet

Worklet 与 Workflow 一样, 是 Tasks 的集合。不过 Worklet 不能作为一个独立运行的单元, 它只能放到 Workflow 中作为 Workflow 的子集运行, 同时 Worklet 也是一个可被多个 Workflow 重用的对象。设计 Worklet 时, 打开 Workflow Manager, 选择 Worklet Designer, 这时选择菜单 Worklets→Create 命令, 其余的创建过程和 Workflow 完全相同。同样 Worklet 可以包括多个 Session, 也可以有其他的 Tasks, 如图 3-17 所示。



图 3-17

不同的是，Worklet 虽然也有虚拟的起始 Task，但是它不是一个可以独立执行的对象。只有将 Worklet 加入到一个 Workflow 中，它才可以被执行。

已经被创建的 Worklet 可以被集成到 Workflow 中，图 3-18 所示就是一个包含 Worklet 的 Workflow。



图 3-18

### 3.4 Command

Command 在 PowerCenter 中使用的图标是

Command 是 Workflow 中可以使用的一个 Task。开发人员可以使用 Command 调用外部的命令或者 Shell。创建一个包含 Command 的 Workflow 如图 3-19 所示。

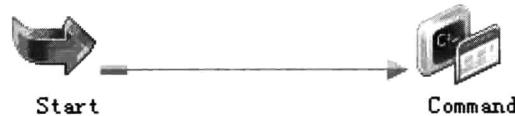


图 3-19

双击 Command，选择 Commands Tab，单击 Add a New Command 按钮可以增加多个

Command。如图 3-20 所示，在这个 Command Task 中增加了两个 Command，分别调用了一个 Shell Script 和 ls 命令。

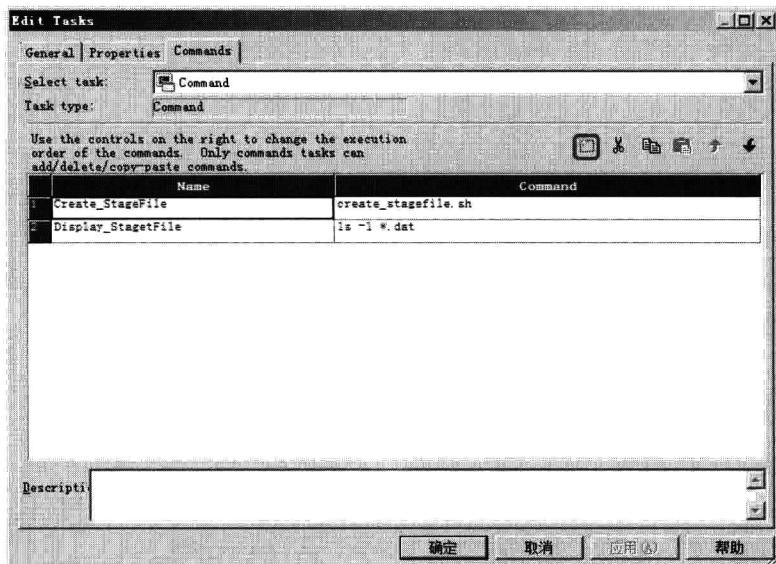


图 3-20

#### → 注释

一个经常被问到的关于 Command Task 的问题是：不管 Command Task 调用的脚本是否成功，Command Task 的状态永远是成功的，这是为什么？产生这个问题的主要原因是很多 Shell Hello World 教材中的样例脚本都没有写 Exit，导致很多人使用的 Shell 只写了功能，并没有用 Exit 语句表示这个 Shell 是否运行并成功退出。

如下是一个完整的 Hello World Shell，在这个 Shell 中，显式地调用了 Exit；否则 PowerCenter 就认为只要成功调用了这个 Shell Script，它的状态就是成功的。

```
#!/bin/bash
echo "Hello World !"
exit 1
```

在 Properties Tab 中，还有两个属性也经常被用到。

- Fail task if any command failed：如果 Command Task 调用多个 Shell，任何一个 Shell 失败，PowerCenter 都会将这个 Command Task 标记为 Failed。

- ⑤ Recovery Strategy: 它有两个选项，即 Fail Task and continue workflow 和 Restart Task。

## 3.5 Control

Control Task 在 PowerCenter 中使用的图标是 。

Control Task 提供了通过 Workflow/Worklet 内的任务，控制 Workflow/Worklet 状态的能力。例如，在 Workflow 中一个关键任务失败，需要停止整个 Workflow，并将 Workflow 标记为失败。如图 3-21 所示的 Workflow。

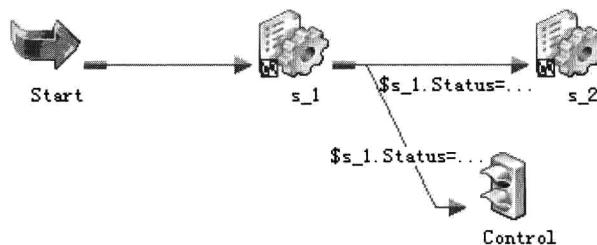


图 3-21

在这个 Workflow 中使用了 Session.status。当 s\_1 执行成功时，将继续执行 s\_2；当 s\_1 执行失败后，执行 Control Task。停止、Failed 还是 Abort 整个 Workflow，依赖于在 Control 组件中的设置。Control Task 的 Control Option（在 Control Task 的 Properties Tab 中）包括如表 3-1 所示的选项。

表 3-1

Control Option	描述
Fail Me	将 Control Task 本身标记为 Failed
Fail Parent	将包含 Control Task 的 Worklet/Workflow 标记为 Failed
Stop Parent	停止包含 Control Task 的 Worklet 或者 Workflow
Abort Parent	Abort 包含 Control Task 的 Worklet 或者 Workflow
Fail Top-Level Workflow	将包含 Control Task 的 Workflow 标记为 Failed
Stop Top-Level Workflow	停止包含 Control Task 的 Workflow
Abort Top-Level Workflow	Abort 包含 Control Task 的 Workflow

## 3.6 发送 E-mail

E-mail Task 在 PowerCenter 中使用的图标是 。

确保 PowerCenter 可以发送 E-mail，首先需要做些配置，对 Windows、UNIX 或者 Linux 操作系统都有所不同，本节介绍为了能够发送 E-mail 需要提前进行的配置。

### 3.6.1 配置发送 E-mail

支持 E-mail 发送，需要至少完成以下两步设置：

- 首先，需要配置 Integration Service 确保其能够发送 E-mail。
- 其次，在 PowerCenter 服务器端安装 PowerCenter 支持的发送 E-mail 的客户端。

(1) UNIX/Linux：在多数 UNIX/Linux 平台上，PowerCenter 使用 Rmail 发送邮件；但在 SUSE 11 64bits 平台上，PowerCenter 使用 Sendmail 发送邮件。

(2) Microsoft Windows：PowerCenter 可以支持 SMTP 或者 MAPI 协议发送 E-mail，默  
认状态下 PowerCenter 使用 Microsoft Outlook 通过 MAPI 接口发送 E-mail。

#### → 注释

有时候笔者会被问到：PowerCenter 是否支持通过 Lotus Notes（Domino）发送邮件？  
答案是肯定的。但本质问题是，PowerCenter 是否能发送邮件事实上和邮件服务器无关，  
重要的是邮件客户端是否支持相关的协议，并且该客户端可以支持对应的邮件服务器。例  
如，在 Microsoft Windows 平台上安装 Outlook，并使用 Outlook 与 Lotus Notes 服务器相连，  
这也是一种变通的方式。

### 3.6.2 在 Workflow 中使用 E-mail

在 Workflow 中发送 E-mail，有 3 个地方可以配置 E-mail。

#### 1. 在 Workflow/Worklet 中发送 E-mail

如图 3-22 所示是一种典型的在 Workflow/Worklet 中发送 E-mail 的方式，在 Workflow

中添加一个 E-mail Task，并在 E-mail Task 中填写收件人、主题和邮件正文即可。



图 3-22

双击 E-mail，选择 E-mail Task 的 Properties Tab，如图 3-23 所示。

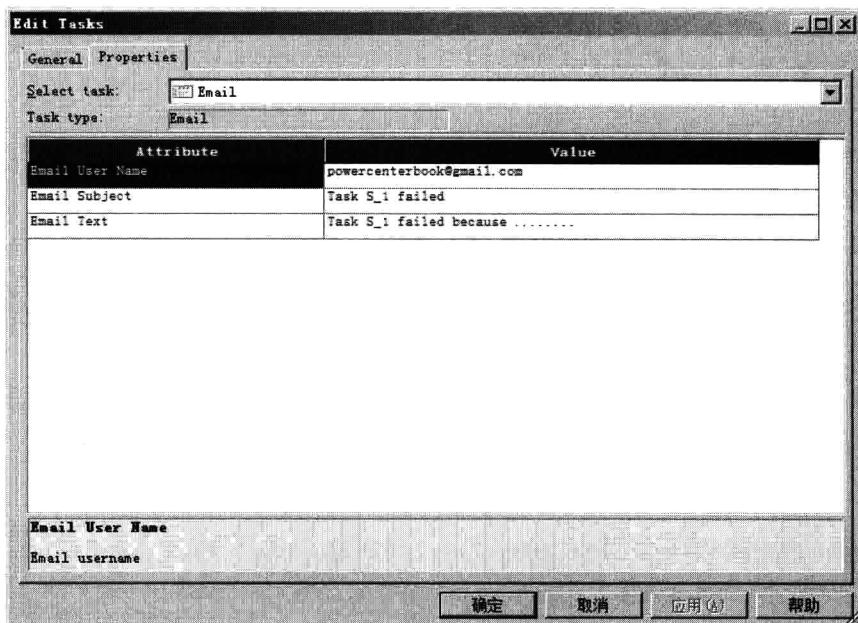


图 3-23

有的情况下希望在 Email Task 中发送的不仅仅是一个描述信息，还需要包括当前 Session 的上下文信息。使用现在的方法就无法实现了。如果想在 E-mail 中发送 Session 相关的上下文信息，需要使用在 Session 的 Components Tab 中发送 E-mail 的功能。

## 2. 在 Session 的 Components Tab 中发送 E-mail

在任意的 Session 中选择 Components Tab，有两个关于 Email 的选项：On Success E-mail 和 On Failure E-mail，分别表示当 Session 运行成功时发送 E-mail 和当 Session 运行失败后发送 E-mail，如图 3-24 所示。

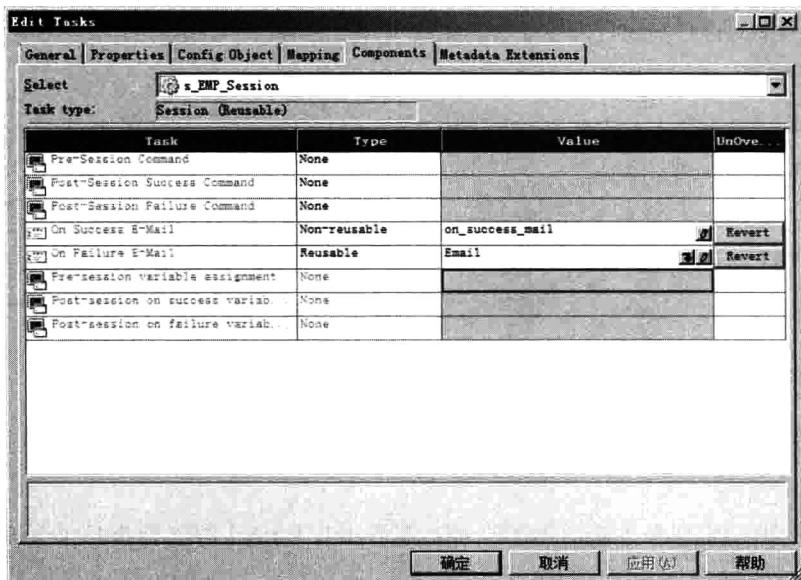


图 3-24

E-mail Type 也有两个选项：Reusable 和 Non-Reusable。Reusable E-mail 如何创建将在下面的小节中有所介绍，这里不再赘述。

与 Session 相关的 E-mail 重点是 Non-Reusable，单击后面的铅笔图标，可以编辑 E-mail 的内容，如图 3-25 所示。

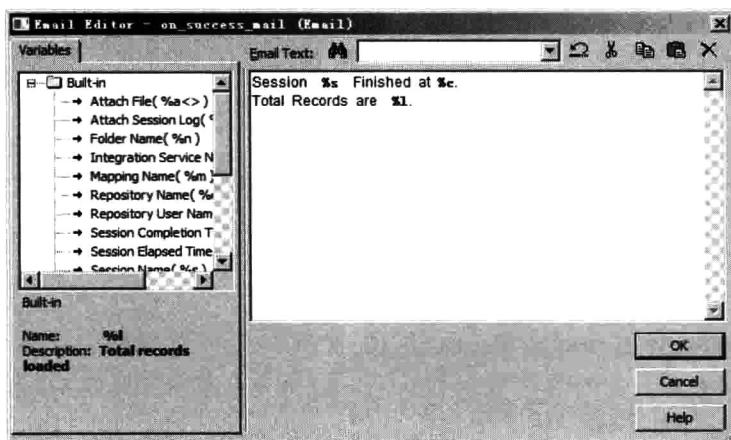


图 3-25

使用如下格式的 E-mail，内容更加丰富，更有利于收件人理解正在运行的 Session 及存在的问题。下面是一个简单的带有变量的 E-mail 的样例。

```
Session %s Finished at %c.  
Total Records are %l.
```

其中%*s*、%*c*、%*l* 均为 PowerCenter 内置的变量，分别代表 Session Name、完成时间和加载到目标的记录数。PowerCenter 还提供了更多的其他变量，详细内容请参考联机帮助。

### 3. 在 Workflow 的 General Tab 中发送 E-mail

PowerCenter 发送的 E-mail 有两种类型，一种是 Reusable E-mail，另一种是 Non-Reusable E-mail。

首先介绍如何创建一个 Reusable E-mail。Reusable E-mail 是在 Task Developer 中创建的，创建 Reusable E-mail 与创建 Reusable Session 的方法是一样的。在这里创建的 Reusable E-mail 同样只能填写固定的、可重用的信息。当创建的 Reusable E-mail 在 Session 的 Components Tab 中使用时，还可以添加更多的信息，包括使用 PowerCenter 提供的相关变量。因此这里 Reusable E-mail 一般用来指定收件人、统一的 E-mail 主题和 E-mail 内容的通用内容，在具体使用时再增加变量相关的内容，如图 3-26 所示。



图 3-26

当创建了 Reusable E-mail 之后，就可以配置 Workflow 发送 E-mail 了。选择要配置发送 E-mail 的 Workflow，选择菜单 Workflows→Edit 命令，弹出如图 3-27 所示的对话框。

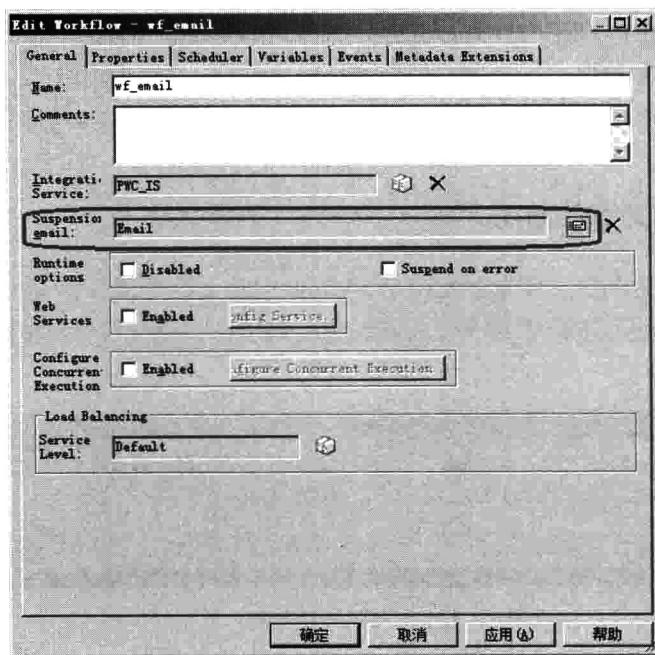


图 3-27

在 Suspension email 中选择刚刚创建的 Reusable E-mail 即可。

## 3.7 Event Tasks

Event Tasks 包括两个任务组件：Event Wait 和 Event Raise，它们在 PowerCenter 中使用的图标分别是  和 。一般情况下这两个 Task 在 Workflow 中相互配合使用。例如，通过 Event Raise 触发事件，Event Wait 等待 Event Raise 触发的事件，一旦它获得了正在等待的事件，就会触发其后续的 Task。

它们组合使用一般解决的是在比较复杂的 Workflow 中定义两个组件的前后继关系。Event 是 Workflow 级别的一个对象，因此它无法跨 Workflow 使用。同理，Event 是在 Workflow 中定义的。

在 PowerCenter 中有两类事件。

- 用户自定义的事件：开发人员在 Workflow 中定义的一个 Event 对象。

- ◎ 预定义事件：这是一种 File-Watch 事件，即 Event Wait 等待一个触发文件。这在项目中经常使用，一个项目的两个团队约定了一个文件接口。为了通知对方接口文件已经生成，经常会使用.trigger 文件告知对方接口文件已经生成。

### 3.7.1 用户自定义事件使用

使用用户自定义事件，首先需要在 Workflow 中定义这些事件。选择菜单 Workflows→Edit 命令，在弹出的对话框中选择 Events Tab，增加一个新事件，如 EventX，如图 3-28 所示。



图 3-28

这时，在 Workflow 中才可以使用 Event Raise 和 Event Wait。如图 3-29 所示的 Workflow。

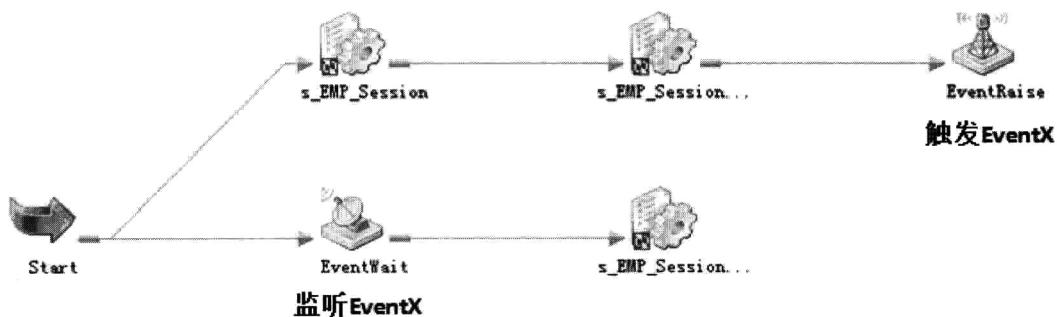


图 3-29

在 Event Raise 中的配置如图 3-30 所示。

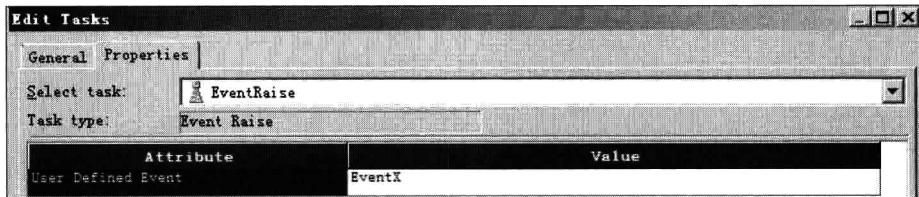


图 3-30

在 Event Wait 中的配置如图 3-31 所示。

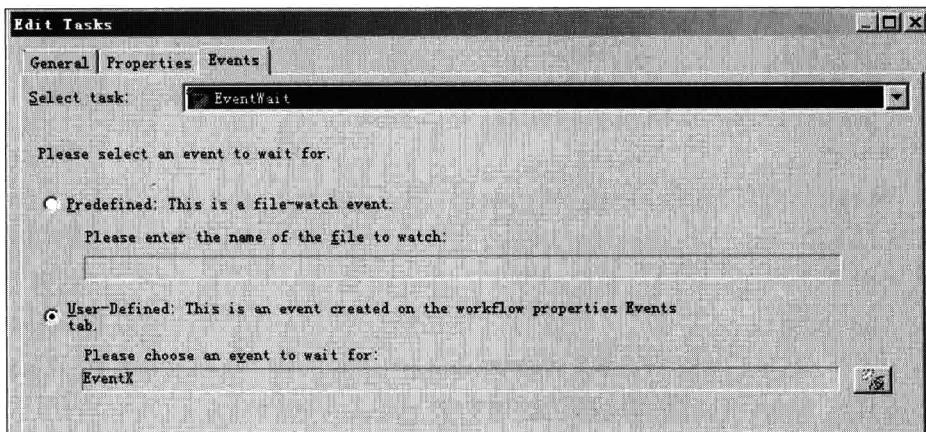


图 3-31

这样就形成了一个 Event Raise 和 Event Wait 的组合对。当 EventX 被 Event Raise 触发后，Event Wait 将收到相关的事件，从而触发后续的流程。

### 3.7.2 预定义事件使用

前面已经提到，预定义事件事实上是一个 File Watch 事件，仅使用 Event Wait 任务即可。即可以配置 Event Wait 任务让其等待一个事件 (.trigger 文件)，当其发现该文件后，后续任务就会被触发。假如，有一个 Workflow 如图 3-32 所示。



图 3-32

在 Event Wait 任务中，进行如图 3-33 所示的配置。

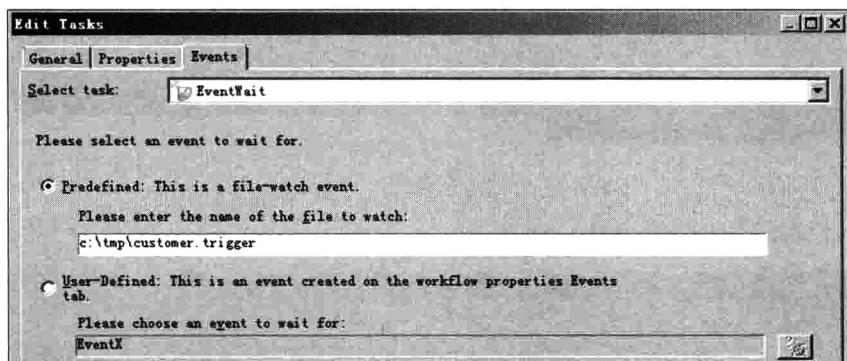


图 3-33

这个 Workflow 的功能是，当该 Workflow 启动后，Event Wait 就在等待一个.trigger 文件，即在文件夹 C:\tmp 下的 customer.trigger 文件。一旦发现这个文件，后续的任务将被触发。默认情况下，即使后续的任务被触发后，.trigger 文件也不会被删除，仍然留在其目录里。因此 PowerCenter 还提供了一个选项，让开发人员决定当任务被触发后，如何处理这个.trigger 文件。这个选项位于 Event Wait 的 Properties Tab 中，叫作 Delete Filewatch File。这个功能支持当 Event Wait 发现文件后触发后续任务，并删除.trigger 文件。

在实际的项目中要求会更严格，比如，需要保存被触发的历史。因此，并非仅仅删除就足够了。理想的方法是不删除.trigger 文件，而是当任务完成后，使用一个统一的 Shell 对.trigger 文件进行归档管理。

## 3.8 Timer

Timer Task 在 PowerCenter 中使用的图标是。

Timer 是一个定时任务，它可以指定其后续任务的启动时间。一个典型的包含 Timer 的 Workflow 如图 3-34 所示。

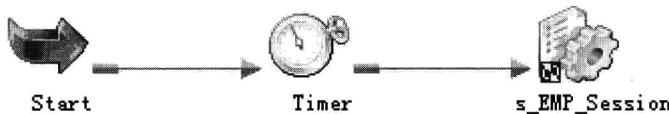


图 3-34

Timer 中的主要属性都在 Timer 组件的 Timer Tab 中，开发人员可以在 Timer Tab 中指定后续任务执行的时间，包括绝对时间和相对时间。

- **绝对时间：**指定后续任务运行的绝对时间。指定运行的绝对时间有两种方法：一是直接使用物理的时间，如 2015/3/15 10:10；二是使用时间变量，通过时间变量指定后续任务启动时间。时间变量的值可以在参数文件中进行指定，也可以通过 Assignment 组件进行动态计算。如果变量值为空，会导致 Timer 失败。
- **相对时间：**指定后续任务在此任务启动或者父任务启动或者上一级任务启动后，后续任务相隔多长时间启动。例如，可以在 Timer 启动 10 分钟后启动后续任务；或者当 Workflow 启动 10 分钟后启动 Timer 后续任务等。

## 3.9 Decision

Decision Task 在 PowerCenter 中使用的图标是。

**案例：**假如有 4 个 Session，分别是 s\_1、s\_2、s\_3 和 s\_4，要求在 s\_1、s\_2、s\_3 任何一个成功执行的情况下都可以执行 s\_4。如何设计这样一个 Workflow 呢？

设计的 Workflow 解决方案如图 3-35 所示。

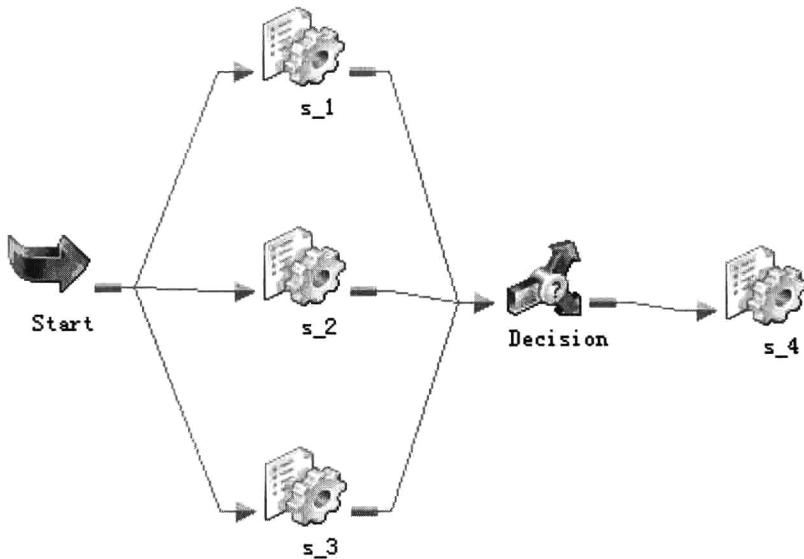


图 3-35

在 Decision 组件中指定的表达式是 “`$$s_1.Status=succeeded OR $$s_2.Status=succeeded OR $$s_3.Status=succeeded`”。这个表达式的含义是：`s_4` 的任意一个前序任务执行成功都可以执行 `s_4`。

当一个任务的前序包括两个以上的任务时，使用 Decision 是一种简单直观的进行条件判断的方法。笔者建议在这种情况下尽量采用 Decision Task，而不是使用 Link Condition 进行判断，虽然某种情况下也可能用多个 Link Condition 来实现。使用多个 Link Condition 进行判断是一种比较麻烦的方法，而且不直观，建议尽量不要使用。

## 3.10 Assignment

Assignment Task 在 PowerCenter 中使用的图标是

Assignment 的作用就是给用户定义的变量赋值，从而可以使 Workflow 的运行更加灵活。开发人员可以给用户定义的变量赋予一个常量或者一个经过表达式计算的值。

例如，一个 Workflow 要执行按日（凌晨 1 点启动）的 ETL，同时还要进行周汇总。当

`s_daily` 成功运行 7 次之后，`s_weekly` 将被触发。设计完成的 Workflow 如图 3-36 所示。



图 3-36

首先，设置这个 Workflow 每天凌晨 1 点启动。

其次，定义一个 Workflow Variable `$$var_int`，并定义它的默认值为 0，指定其为 Persistent 变量。Persistent 变量的含义是：是否保存此 Workflow 上次运行时该变量的值。

定义 Workflow 变量，选择菜单 Workflows→Edit 命令，在弹出的对话框中选择 Variables Tab。具体的定义如图 3-37 所示。



图 3-37

再次，在 Assignment 任务中定义 `$$var_int := $$var_int + 1`，具体如图 3-38 所示。

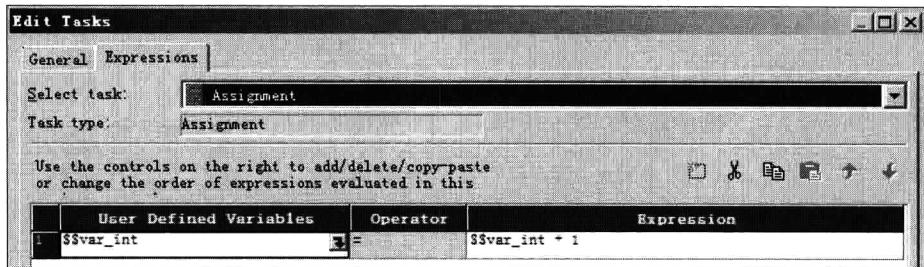


图 3-38

最后，定义从 Assignment 任务到 s\_weekly Session 的 Link Condition 为 $\$\$\text{var\_int} = 7$ 。这样就完成了这个样例的 Workflow。虽然这个样例不一定适合实际的业务场景，但通过此样例，希望读者能够简明地理解 Workflow Variable、Assignment Task 和使用 Workflow Variable 作为条件的一个 Workflow 如何开发。

# 第 4 章

## 常用功能汇集

---

### 4.1 Debugger

任何一个程序员都知道开发过程中出现错误是正常的事情，跟踪错误产生的原因是非常困难的事情，不出现错误则是万万不可能的事情。作为一个老程序员，对任何没有 Debug 能力的开发 IDE 都会冒一身冷汗，因此特意将 PowerCenter 的 Debugger 拿出来介绍一下，希望对所有的开发者能有所帮助。

开始使用 Debugger 之前，需要确保已经开发了一个可以运行的 Mapping，即创建一个 Mapping，并为其创建 Session 和配置相应的参数；运行 Workflow，确保其是可以运行的。这时，就可以开始 Debug 这个 Mapping 了。

(1) 在 PowerCenter Designer 客户端中，选择菜单 Mapping→Debugger→Start Debugger 命令（快捷键 F9）。

(2) 在此后的对话框中依次选择 Next→Using Existing Session→Next→Next 即可。

#### ➔ 注释

看一下 PowerCenter 菜单提供的功能：F10（Next Instance），可以继续运行；F5（Continue），运行到下一个断点，如果没有断点，则运行结束；Alt+F9（Edit Breakpoints），可以编辑断点；Evaluate Expression 还可以对表达式进行计算，如图 4-1 所示。

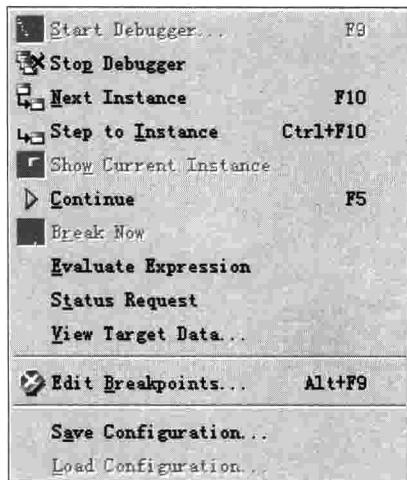


图 4-1

当进入 Debug 模式时，会在窗口的最下方出现如图 4-2 所示的窗口，这时可以看到任意组件正在流入的数据和计算的结果。

图 4-2

### → 注释

- ◎ 在图 4-2 中，对浮点数使用科学计数法进行显示，这并不影响实际计算的结果。
- ◎ 在 Debug 过程中，数据量不宜过多。数据量过多，PowerCenter Debug 有可能有死机的感觉。

对 Debug 的细节不再进行详细介绍，本节的目的在于帮助开发人员方便地找到 Debug 功能，开始 Debug。对于如何配置断点等细节操作，可以参考 PowerCenter 联机手册。其实即使不使用联机手册，多数程序员也能非常快速地掌握这些功能。

## 4.2 Maplet/Reusable Transformation

Maplet/Reusable Transformation 与数据库中的存储过程和函数比较相似，它们都是一个某种功能的集合，是为了提供公共功能的可重用性。假如，很多程序都会使用“ $x+100$ ”这个功能，因此创建了一个函数叫 Add100。在 PowerCenter 中类似的功能是通过 Maplet 和 Reusable Transformation 来实现的。

Maplet 与 Reusable Transformation 的区别在于，Reusable Transformation 仅仅能使用一个 Transformation；而 Maplet 可以使用多个 Transformation 的组合，从而实现更复杂的、可重用的功能。

### 4.2.1 Reusable Transformation

前面已经提到过，Reusable Transformation 是通过一个 Transformation 实现可重用功能的。还是以“ $x+100$ ”这个需求来展示如何创建一个 Reusable Transformation。

(1) 在 PowerCenter 中，单击 Transformation Developer 图标按钮，如图 4-3 所示。

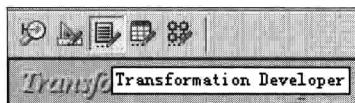


图 4-3

(2) 将一个 Expression Transformation 拖入 Transformation Developer Workspace，并将其重命名为 exp\_add\_100。名称可以是任意的，这里为其指定一个固定的名字是为了后续描述更加容易。

为 exp\_add\_100 增加两个端口，分别为 Var\_in 和 Var\_out，数据类型为 Integer，并将 Var\_out 赋值为“Var\_in+100”，如图 4-4 所示。

这样一个简单的 Reusable Transformation 就创建完毕了。这时在 PowerCenter Designer Navigator 的 Transformation 下面增加了一个名为 exp\_add\_100 的 Reusable Transformation，如图 4-5 所示。

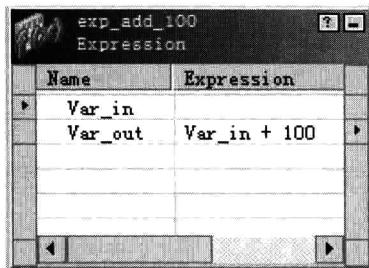


图 4-4

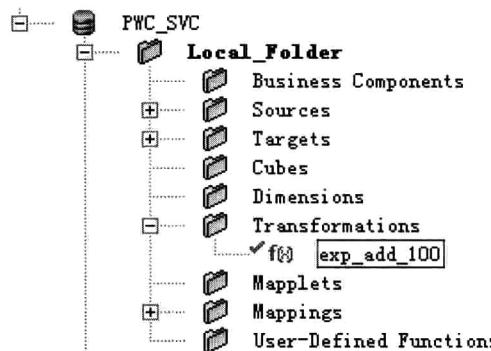


图 4-5

(3) 这时可以将 exp\_add\_100 用于需要的 Mapping，如图 4-6 所示。

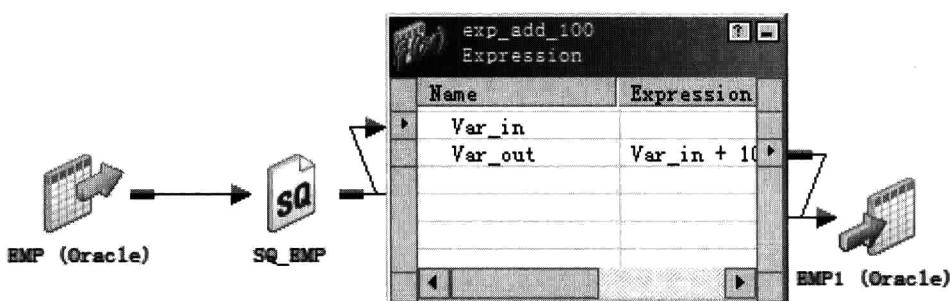


图 4-6

Reusable Transformation 在 Mapping 中是不可以被编辑的。当 exp\_add\_100 这个 Reusable Transformation 在 Transformation Developer 中被修改时，所有引用它的 Mapping 也将自动被修改。

### 4.2.2 Mapplet

与 Reusable Transformation一样，Mapplet也是一个类似数据库函数和存储过程的可重用对象。与Reusable Transformation不同，Mapplet可以实现更加复杂的功能，因为它可以是多个Transformation的组合。

还是以“ $x+100$ ”这个需求来展示如何创建一个Mapplet。

(1) 在PowerCenter Designer中，单击Mapplet Designer图标按钮，如图4-7所示。

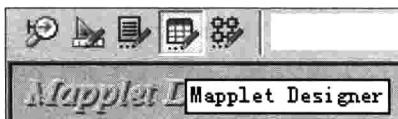


图 4-7

(2) 通过菜单 Mapplet→Create 创建一个 Mapplet，并为其命名，如 mplt\_add\_100。

(3) 为这个新创建的 Mapplet 添加一个 Expression Transformation，并为其增加两个端口，分别为 Var\_in 和 Var\_out，数据类型为 Integer，并将 Var\_out 赋值为“Var\_in+100”。这些并不够，还需要给其增加两个额外的 Transformation：Mapplet Input 和 Mapplet Output，如图4-8所示。



图 4-8

这时，创建的 Mapplet 如图 4-9 所示，看起来和创建一个 Mapping 非常相似，只是这里创建的 Mapplet 的源和目标分别是 Mapplet Input 和 Mapplet Output。如果计算逻辑不像“ $x+100$ ”这么简单，还可以在 Mapplet Input 和 Mapplet Output 之间增加更多的 Transformation。

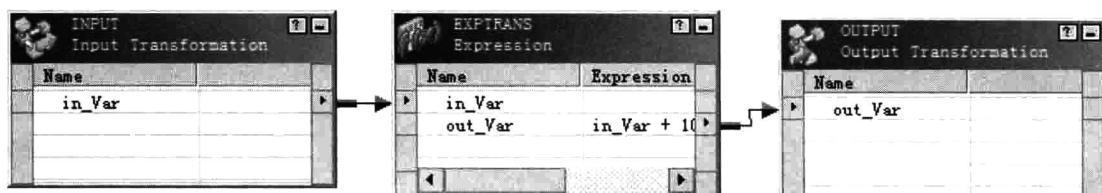


图 4-9

(4) Mapplet 创建完成后，在 PowerCenter Designer Navigator 的 Mapplets 下面增加了一个名为 mplt\_add\_100 的 Mapplet，如图 4-10 所示。

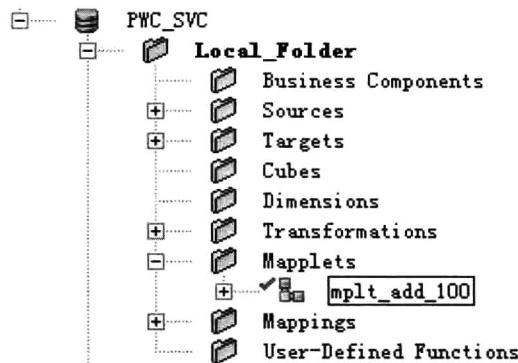


图 4-10

(5) 在 Mapping 中使用 Mapplet 非常简单，只要将 Mapplet 拖入需要使用它的 Mapping，为其指定需要的输入和输出即可。这时的 Mapplet 就是一个完成特定功能的 Transformation 的集合。一个包括 Mapplet 的样例 Mapping 如图 4-11 所示。

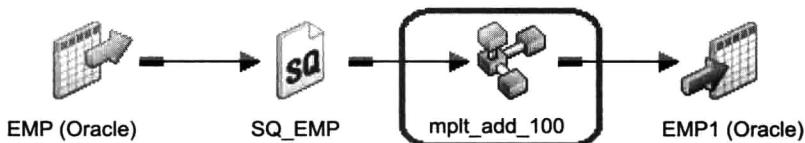


图 4-11

#### → 注释

Mapplet 在 Mapping 中是不可以直接修改的。同时，在 Mapplet Developer 中对 Mapplet 的所有修改都会反映到使用该 Mapplet 的 Mapping 中。

#### 一种特殊的 Mapplet

Mapplet 可以包含数据源，但是不能包含目标。

如图 4-12 所示，这是一个包含数据源的 Mapplet。它包括一个数据源、一个 Mapplet Output Transformation 和若干个 Transformation。

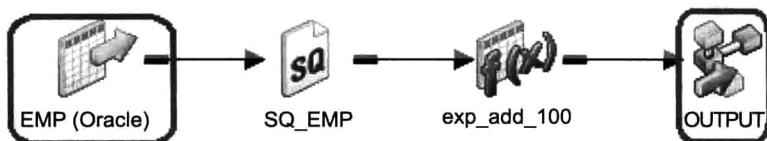


图 4-12

## 4.3 使用 Shortcut

Shortcut 俗称快捷方式，在 PowerCenter 中 Shortcut 主要是为了解决数据源、数据目标或其他对象在跨 Folder 和跨 Repository 方面的一致性及可重用能力。

**案例:** A 公司的大数据平台在 PowerCenter 中创建了 10 个文件夹，分别负责保存物料、生产、物流等模块的 ETL 工作。这么划分的最初目标是为了保持业务线的清晰和独立性。但在开发过程中，项目团队发现在这些业务线中有大量的公用对象，如供应商、物料等。开发人员不得不在所有的文件夹中创建这些数据对象，这是第一个问题。第二个问题很快出现了，公司海外业务扩展迅速，欧洲及南亚公司很快实现盈利，它们也需要自己的数据仓库平台，这些分公司和总部也有大量的公共对象。如果使用这些重复对象的每个项目组，每个业务线都保存一个副本在自己的 Repository 或者 Folder 内，一旦基础的对象发生变更，项目组就需要修改数个、甚至十几个对象。项目经理考虑到，如在项目初期，由于模型可能频繁进行微调，这将使得项目组疲惫不堪；如果项目上线后，大量公共对象重复存储，一旦修改可能会造成部分任务失败。他们希望能找到合适的方案解决这些共享对象如何管理的问题。在讨论和研读文档过程中，他们发现了 Shortcut，这正是他们解决问题的手段。

在这个项目中他们创建了一个 Global Repository 作为公司级的共享对象管理、存放位置，并且在每个 Repository 中创建了一个共享文件夹作为分公司、部门级别共享对象统一管理、存放位置。这两种管理方法在 PowerCenter 中有专有的名字，分别是 Global Shortcut 和 Local Shortcut。

PowerCenter 中可以创建 Shortcut 的对象包括：

- 源定义；
- 可重用的 Transformation；
- Mapplet；

- ④ Mapping;
- ⑤ 目标定义;
- ⑥ 业务组件。

下面详细介绍 Shortcut 的两种类型。

#### 4.3.1 Local Shortcut

创建 Local Shortcut 是在同一个 Repository 中从共享文件夹向其他文件夹创建共享对象的过程。共享对象，即对象 Shortcut 将继承被共享对象的所有属性。这样可以保证当共享文件夹中的对象被修改后，基础对象和它的 Shortcut 的一致性。

(1) 创建 Local Shortcut 首先需要创建一个共享文件夹。打开 Repository Manager，选择菜单 Folder→Create 命令，在弹出的对话框中填写要创建的文件夹名，并勾选 Allow Shortcut 复选框，如图 4-13 所示。

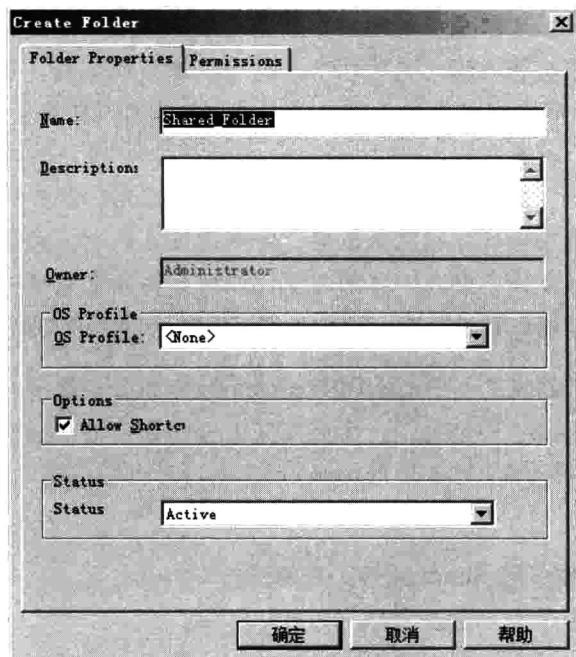


图 4-13

新创建的共享文件夹与默认创建的文件夹不同，在共享文件夹下有一个手形的图标  
 Shared\_Folder。

- (2) 在文件夹中创建一个对象，如一张源表、目标表或者其他对象。
- (3) 同时打开共享文件夹和其他非共享文件夹，将共享文件夹中的一个对象拖入非共享文件夹。在释放鼠标之前，在PowerCenter的提示栏有一条信息“Create shortcut to this object in the selected folder”，释放鼠标，一个Shortcut即创建完成。
- (4) 在工作区中双击Shortcut，即可查看Shortcut的详细信息，包括Shortcut命名、它引用的来源等，如图4-14所示。

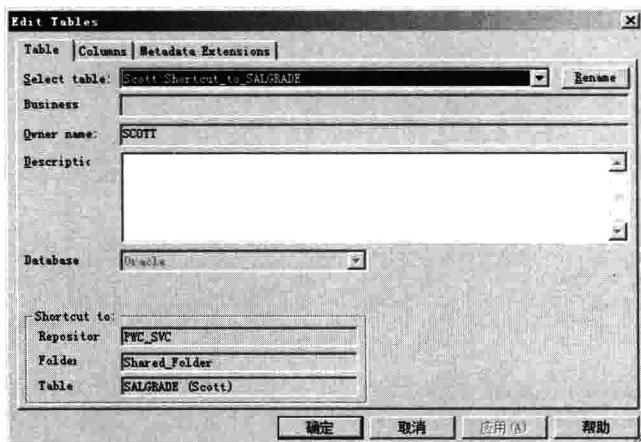


图 4-14

### 4.3.2 Global Shortcut

Global Shortcut用于管理Repository之间的共享对象。创建Global Shortcut，首先必须有一个Global Repository和Local Repository。PowerCenter默认创建的Repository都是Local Repository。创建Global Shortcut，首先从创建Global Repository开始。

- (1) 升级一个Local Repository为Global Repository。在Admin Console中，选择一个要升级的Local Repository，在Properties Tab中单击编辑的笔形图标，将Global Repository的值由false修改为true即可，如图4-15所示。提示：在此过程中该Repository Mode必须为Exclusive。



图 4-15

修改后，重启 Repository 即可将 Local Repository 升级为 Global Repository。提示：这个升级过程是不可逆的。

(2) 注册 Local Repository 是必须完成的第二步，否则无法创建 Global Shortcut。在 Admin Console 中，选择一个 Local Repository，然后选择右侧的菜单 Action→Repository Domain→Register Local Repository 命令，如图 4-16 所示，会弹出如图 4-17 所示的对话框。

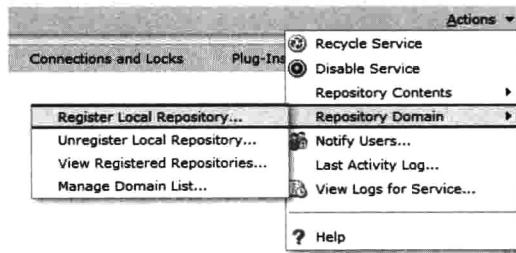


图 4-16

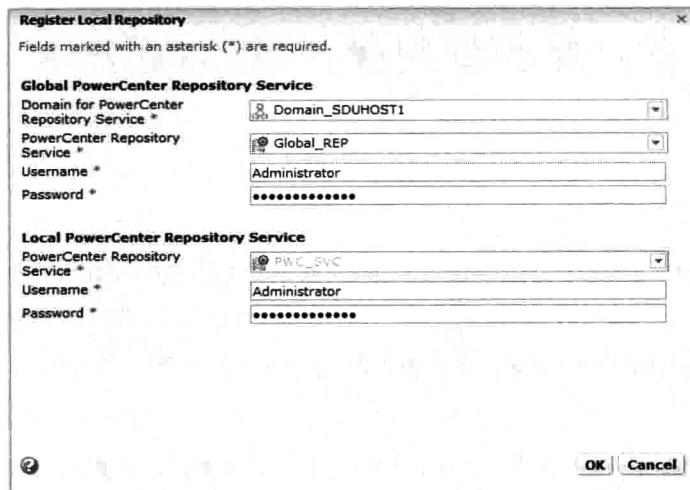


图 4-17

选择 Global Repository 和 Local Repository，并输入密码，单击 OK 按钮，完成注册。

(3) 打开 PowerCenter Designer，在 Navigator 中会看到 Global Repository 被加入到了 Local Repository 中，即可以在 Local Repository 中看到 Global Repository 的对象，如图 4-18 所示。

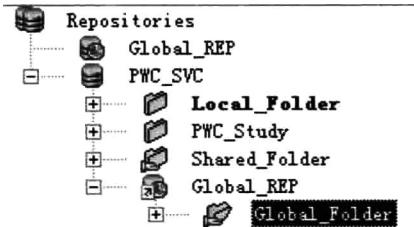


图 4-18

(4) 此外，还需要在 Global Repository 中创建共享文件夹和要共享的对象，并将共享对象拖入非共享文件夹，这些过程与 Local Shortcut 完全相同。

#### → 注释

非常遗憾的是，Global Shortcut 和 Local Shortcut 在显示方面没有明显的区分，一般的建议是可以采用增加注释或者不同的命名方式加以区分。

## 4.4 Session 相关属性

Session 是 Mapping 的可运行的实例，一个 Mapping 可以有多个 Session。Session 提供了大量的属性和配置选项以控制、优化 Session 的运行。有些属性随着某个主题的特点已经进行了详细的介绍，但是还有些属性并没有机会进行解释，但是这些内容在项目中同样重要。下面对这些属性、配置做一个简单的介绍。

### 4.4.1 Properties Tab 相关属性

重点介绍一下 Properties Tab 中的以下选项。

(1) Write Backward Compatible Session Log File：很早之前，PowerCenter 的日志文件是文本文件，用任何文本编辑器都可以阅读。后来日志文件被写成了一个\*.bin 文件，只能

通过 PowerCenter 的日志浏览器阅读。为了提供与前期版本的兼容性，PowerCenter 提供了这个选项。与这个选项相关的还有两个属性。

- Session Log File Name: 可以指定日志文件的文件名。
- Session Log File directory: 可以修改存放日志的默认目录，将日志存放在指定的目录下。

(2) Enable Test Load & Number of Rows to Test: 使用这两个参数可以指定以较少的数据对 Session 进行测试。如果使用 Enable Test Load, Integration Service 不会将数据写入目标，但会执行 Session 的全部其他逻辑。

(3) Treat source rows as: 有 4 个值可以选择，即 Insert、Delete、Update 和 Data Driven。默认值是 Insert。选择这些值类似给每行数据增加了一个操作的标签。数据在进入目标时，PowerCenter 将按照标签指定的操作执行。如果标签为 Insert, PowerCenter 将执行插入；如果标签是 Delete 或者 Update, PowerCenter 将删除或者更新对应的行。为了能够生成完整的 SQL 语句，Target 的目标必须指定逻辑主键，否则将无法正确执行。Data Driven 需要与 Update Strategy 配合使用，由 Update Strategy 指定要执行的动作，实现数据驱动的操作。

(4) Rollback Transactions on Errors: Integration Service 遇到非致命错误时，也会回滚当前事务。如果在不选择该选项的情况下，向表内插入 10 条数据，其中一条由于主键冲突被 Reject，另外的 9 条会成功插入。如果选择了此选项，且这 10 条数据在同一个事务中，这 10 条数据将全部被回滚。

(5) Java Classpath: 如果在 PowerCenter 中使用了第三方的 Java 类包或者自定义的类，需要在此指定相关的 Java 类包的路径，尤其是使用 Java Transformation 时。

(6) Session retry on deadlock: 在使用 Normal 写的情况下，发生目标写死锁，PowerCenter 将尝试再次向目标写入。尝试的次数取决于参数 Number of Deadlock Retries，但这个参数是 Integration Service 的一个属性。

#### 4.4.2 Config Object Tab 相关属性

重点介绍一下 Config Object Tab 中的以下选项。

- (1) Constraint Based Load Ordering: PowerCenter 基于主外键的约束向目标加载数据。
- (2) Custom Properties: PowerCenter 在遇到问题时，有时需要使用一些内部参数辅助

执行或者解决遇到的问题。一般情况下这些属性为非公开属性，需要 Informatica 售后指导。例如，OptimizeODBCWrite 和 OptimizeODBCRead 是 PowerCenter 使用 ODBC 读/写时经常用到的两个参数，用于避免读/写异常。

(3) Log Options: 为了跟踪 Session 的历史运行情况，包括解决问题、性能调优、满足合规要求等，PowerCenter 提供了一系列参数。

- Save Session log by: 按次或者按照时间戳保存日志。如果使用按次运行，需要提供保存日志的个数，使用参数 Save Session log for these runs。
- Session Log File Max Size: 决定 Session Log 文件的最大值。默认值为 0，表示无限制。当 Session Log 特别大或者运行 Real-Time Session 时，最好设置 Session Log 文件的最大限制。单位为 MB。
- Session Log File Max Time Period: 决定一个 Session Log 文件存放日志的最大周期，如 10 小时，同样是为了避免 Session Log 过大影响使用、管理。适合 Real-Time Session 的场景。单位为小时。
- Maximum Partial Session Log Files: 保存的最大 Log 数量，如果超过这个数量，旧的日志将被重写。该参数需要与 Session Log File Max Size 和 Session Log File Max Time Period 配合使用。
- Writer Commit Statistics Log Frequency: 默认值为 1，每个 Commit 都会写入日志中。
- Writer Commit Statistics Log Interval: 写 Commit 统计的时间间隔，单位是分钟。

(4) Stop on errors: 默认情况下，在 Session 运行时即使有些数据被 Reject 了，Session 也将继续运行，且 Session 的最终运行状态是成功。假如这个 Session 特别关键，即使有一行数据错误，Session 也需要停止运行，并标记为失败，这时候就需要设置此选项。

(5) Override tracing: 主要是为了当 Session 错误发生时跟踪 Session 运行错误而启动不同的日志级别。同时日志级别过高也会影响 Session 的性能，因此在完成调试后，需要将此值恢复到默认值。

(6) On Stored Procedure ERROR: 默认值为 Stop。当 Stored Procedure 发生错误时，停止整个 Session。也可以根据情况选择“Continue” Session 运行。

(7) On Pre-Session command task ERROR: 默认值为 Stop。当 Pre-Session Command 发生错误时，停止整个 Session。也可以根据情况选择“Continue” Session 运行。

(8) On Pre-Post SQL ERROR: 默认值为 Stop。当 Pre-Post SQL 发生错误时，停止整个 Session。也可以根据情况选择“Continue” Session 运行。

(9) Error Log Type: 有 3 个选项，即 None、Database、Flat File，默认值为 None。选择其他选项，可以将错误信息，包括异常数据存放在单独的日志。

## 4.5 参数和变量

“不能把程序写死”，这是刚刚学习写应用程序时，老师傅就经常提及的注意事项。那如何才能不把程序写死呢？参数和变量是有力的武器，在程序中有可能变化的地方都应该尽量使用参数和变量。下面共享几个典型的参数和变量的应用场景，帮助读者了解参数和变量的使用意义。

**场景 1：**在一个保险公司的项目中，项目组希望每天早上获取前一天新的保单记录。这些记录存放在表 TAB\_INSURANCE 中，在这张表中有一个字段 LAST\_UPDATE，数据类型为 DATE，数据库为 Oracle。如何实现上面的功能呢？答案是使用 PowerCenter 参数。

**场景 2：**同样是这个保险公司的项目，在这个项目中有 3 个环境，分别是开发、测试和生产，它们使用的数据库分别是 ABC\_DEV、ABC\_QA 和 ABC\_PROD。项目组希望产品部署过程实现自动化。但是他们发现整个项目有上万个 Mapping 和 Session，手工修改数据库连接几乎不可能。怎么办呢？解决方案是使用 PowerCenter 参数。

**场景 3：**批量修改问题。项目组得到一个通知，由于公司重新规划 IT 基础设施，要对所有的目录命名规范进行重新定义。数万个 Session、数万个目录都需要进行修改，悲剧将要发生。幸好他们发现，项目组在项目设计时已经考虑到了这点，所有的目录都使用了参数，他们只需要修改参数文件中的几个参数即可。

参数和变量还有更多的应用场景，在此无法一一叙述，笔者尽力将更多的场景嵌入到后面的叙述中，帮助大家掌握参数和变量的基本用法。

### 4.5.1 Mapping 参数

以场景 1 为例，介绍一下 Mapping 参数。在使用 Mapping 参数之前首先需要定义 Mapping 参数。为了方便起见，仍然使用 EMP 表，这是一张非常伟大的表，它拥有几乎所有常见的数据类型，这次使用的字段是 HIREDATE，它是 DATE 类型。需要获取的数据为前一天入职的员工信息。

## 1. 定义参数

Mapping 参数是在 Mapping 层面的参数，也就是说这个参数是不能跨 Mapping 使用的，它只能在 Mapping 层面上进行定义。新建一个 Mapping，选择菜单 Mappings→Parameters and Variables 命令，为这个 Mapping 定义一个参数，如参数名为 \$\$VARDATE，参数类型为 String，如图 4-19 所示。

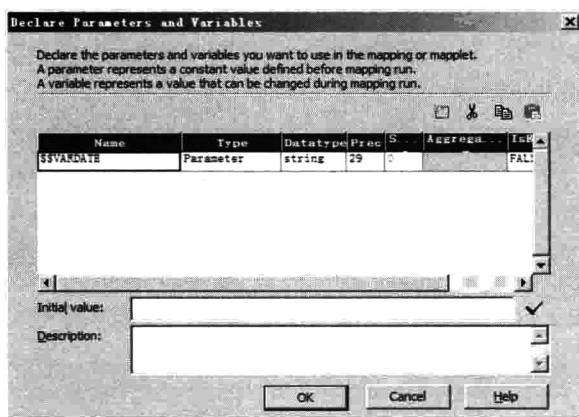


图 4-19

### → 注释

这里不知道有没有人会有疑问，为什么不设置为 DateTime 类型？这个问题稍后再讨论。

还可以为参数设置初始值 Intial Value。对日期格式而言，初始值可以是 MM/DD/RR、MM/DD/RR HH24:MI:SS、MM/DD/YYYY、MM/DD/YYYY HH24:MI:SS.US 格式。

## 2. 使用参数

在 Mapping 中使用参数，选择此 Mapping 的 Source Qualifier，打开 Properties Tab，选择 SQL Query。在 SQL Query 中自定义 SQL 如下：

```
SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO
FROM
EMP
WHERE HIREDATE BETWEEN TO_DATE($$VARDATE, 'MM/DD/YYYY') AND TO_DATE
($$VARDATE, 'MM/DD/YYYY')+1
```

上述代码的重点在于 WHERE 语句后如何使用变量 \$\$VARDATE。

后续 Mapping 定义与日常的设计没有差异，只需要确保这个新设计的 Mapping 是有效的即可。

### 3. 定义、使用参数文件

创建一个可以运行的 Workflow，编辑 Workflow（选择菜单 Workflows→Edit 命令）。在 Workflow 的 Properties Tab 中，在 Parameter Filename 后添加参数文件的所在目录，如 c:\para\para.txt，如图 4-20 所示。

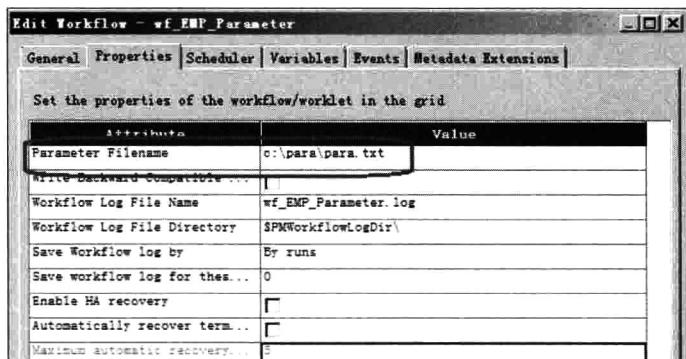


图 4-20

这是一个最简单的参数文件，它的格式如图 4-21 所示。

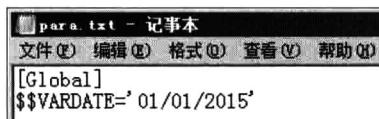


图 4-21

#### → 注释

当参数定义为 String 类型时，在参数文件中定义参数值，必须使用单引号。如果不在此处使用单引号，请在 SQL Query 中使用单引号。

既然 HIREDATE 是日期类型，为什么 \$\$VARDATE 不定义为日期类型呢？这是由于 PowerCenter 对日期类型的解释问题。如果已经将 \$\$VARDATE 定义为日期类型，并在参数文件中为其赋值，如 \$\$VARDATE=01/01/2015，当 Session 运行时，它自动生成的 SQL 如下：

```
Select .... From EMP where HIREDATE BETWEEN 01/01/2015 00: 00: 00 AND....
```

这样也并不能满足日期类型直接使用的问题，还是需要再次使用 `to_date()` 等类似的函数。因此，如果是上面案例涉及的问题，一般情况下会使用 `String` 类型，而不是 `DATE` 类型。

参数文件都可以在哪里指定？答案是：可以在 Workflow 的属性或者在 Session 的属性中指定。但实践中一般都会指定在 Workflow 的属性中。这样可以减少参数文件设置的数量，减少未来的维护工作量。

#### 4. 参数文件格式

前面已经看到了一个简单的参数文件，也初步了解到 PowerCenter 参数是通过文件进行传递的。那么一个完整的参数文件应该是什么样子的？先看一个具体的例子，这个例子引用自 PowerCenter 手册。

---

```
File created by RSmith 11/12/2005
-----
[Global]
$$VARDATE=01/01/2015
[Service:IntSvs_01]
$PMSuccessEmailUser=pcadmin@mail.com
$PMFailureEmailUser=pcadmin@mail.com
[HET_TGTS.WF:wf_TCOMMIT_INST_ALIAS]
$$platform=unix
[HET_TGTS.WF:wf_TGTS_ASC_ORDR.ST:s_TGTS_ASC_ORDR]
$$platform=unix
$DBConnection_ora=Ora2
[ORDERS.WF:wf_PARAM_FILE.WT:WL_PARAM_Lvl_1]
$$DT_WL_lvl_1=02/01/2005 01:05:11
$$Double_WL_lvl_1=2.2
[ORDERS.WF:wf_PARAM_FILE.WT:WL_PARAM_Lvl_1.WT:NWL_PARAM_Lvl_2]
$$DT_WL_lvl_2=03/01/2005 01:01:01
$$Int_WL_lvl_2=3
$$String_WL_lvl_2=cccccc
```

这是一个比较完整的参数文件，包括多种常用的 Parameter 设置方式，包括带有 \$\$ 或者仅一个 \$ 的参数。Parameter 的使用范围也有多种，它们用中括号 [] 进行标注。

当变量是在 Workflow 上指定时，在 Workflow 启动后，可以在 Workflow Log 中看到相关的信息：Message: Parameter file [c:\para\para.txt] is opened for [workflow [wf\_EMP\_Parameter]]。

#### 4.5.2 Mapping 变量

与参数不同，变量的值是可以在 PowerCenter 中重置，也就是在 Mapping 中这些变量的值可以通过 PowerCenter 提供的函数进行修改。PowerCenter 提供了几个变量修改的函数，开发人员可以在 Mapping 中动态地修改变量，为其进行赋值。定义变量和定义参数步骤一样，但是变量有更多的选项需要关注。先通过一个具体的场景了解 Mapping 变量，后续再对其特性进行详细描述。

**案例 1：**变量有很多用法，稍微修改一下前面提到的一个案例，希望通过这个案例帮助大家初步了解 Mapping 变量。对 ABC 保险公司而言，他们现在希望 Session/Mapping 按照 Continous 方式运行，不断地将新加入的保单导入数据仓库系统。为了统一起见，不再创建新的保单表，而是用 EMP 表模拟保单表，用 EMP.Hiredate 模拟保单的录入日期。

按照这个需求设计完成的 Mapping 如图 4-22 所示。

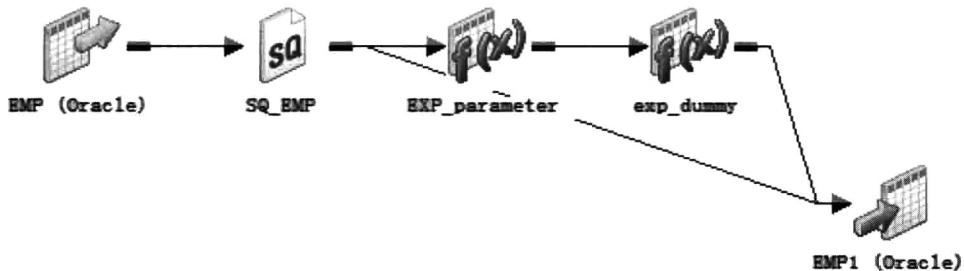


图 4-22

在这个 Mapping 中包括很多组件，每个组件有不同的作用，后续会一一进行说明。

第一步：创建一个 Mapping，并为 Mapping 创建一个 Max 类型的变量。

选择菜单 Mappings→Parameters and Variables 命令，定义一个 Mapping 变量 \$\$Hiredate，

数据类型为 String, Aggregation 类型为 Max, 为其定义 Initial Value ='19001010101010', 如图 4-23 所示。

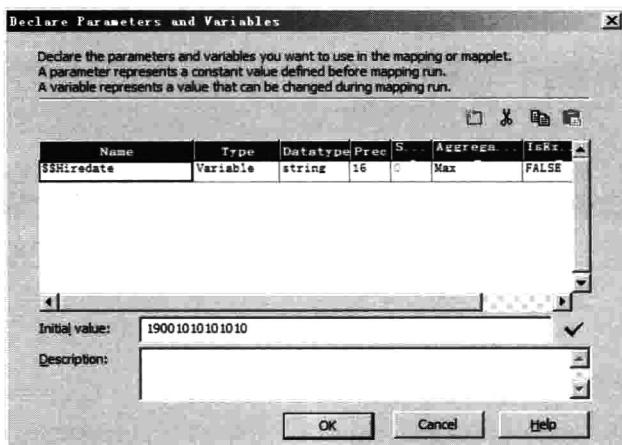


图 4-23

使用正确的 Aggregation 类型非常有必要, 这里定义为 Max 的目的是为了获取上次运行的 Hiredate 的最大值。只有设置为 Max, 当后续使用函数 SetMaxVariable 时才能正确地执行。错误的 Aggregation 类型将在运行时引发类似如下的错误:

```
Transformation: EXPTRANS Field: HIREDATE_OUT
<<PM Parse Error>> [SETMAXVARIABLE]: function cannot be used for min
aggregate type Mapping variable.
... SetMaxVariable(>>>>$$MAXHIREDATE<<<<, HIREDATE)
...there are parsing errors.
```

第二步: 编辑 Source Qualifier, 定义属于自己的 SQL Query。在 SQL Query 中使用刚刚定义的变量。

```
SELECT EMP.EMPNO, EMP.ENAME, EMP.JOB, EMP.MGR, EMP.HIREDATE, EMP.SAL,
EMP.COMM, EMP.DEPTNO
FROM
EMP
where EMP.HIREDATE >to_date($$Hiredate,'yyyy-mm-dd hh24:mi:ss')
```

与前面介绍 Parameter 部分类似的使用方式。但为了支持 Continous 运行, 这里使用了一个开放的模式, 直接使用大于号 (>) 获取从上次运行结束到当前最新的全部数据。

第三步：使用函数 SetMaxVariable 获取变量 \$\$Hiredate 的最大值，并将其存入 Repository 中。在 Expression 组件 EXP\_parameter 中使用函数 SetMaxVariable 获取 Hiredate 的最大值，如图 4-24 所示。

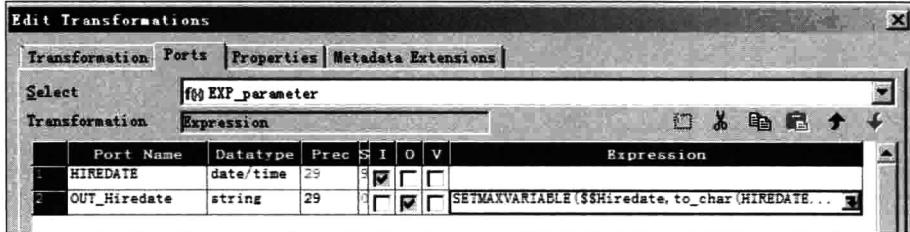


图 4-24

完整的表达式如下：

```
SETMAXVARIABLE ($$Hiredate, to_char(HIREDATE, 'yyyyymmddhh24miss')).
```

第四步：将刚刚创建的 Mapping 实例化为 Session，并运行 Session(第一次运行 Session)。这时在 Session Log 中有如下几条重要的信息。

```
Message: Use default value [190010101010] for mapping variable:  
[$$Hiredate]
```

因为没有使用参数文件为变量赋值，因此这个 Session 使用了在变量定义时的初始化值。因此该 Session 向数据库提交的 SQL 语句如下：

```
Message: SQ Instance [SQ_EMP] User specified SQL Query [SELECT EMP.EMPNO,  
EMP.ENAME, EMP.JOB, EMP.MGR, EMP.HIREDATE, EMP.SAL, EMP.COMM, EMP.DEPTNO  
FROM  
EMP  
where EMP.HIREDATE >to_date(190010101010,'yyyyymmddhh24:mi:ss')]
```

第五步：重复运行 Session，继续查看 Session Log，以更加充分地了解 PowerCenter 变量及变量函数的执行原理和操作方法。

在再次运行 Session 之前，在 Workflow Manager 中，用鼠标右键单击该 Session，在弹出的快捷菜单中选择 View Persistent Values 命令，就可以看到 Repository 中存放的上次 Session 运行后的变量值，如图 4-25 和图 4-26 所示。

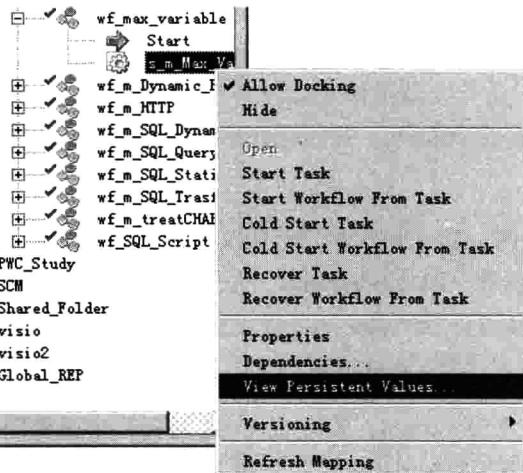


图 4-25

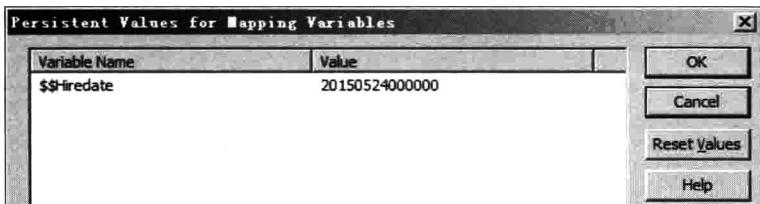


图 4-26

如图 4-26 所示，PowerCenter 通过函数 SetMaxVariable 将 Hiredate 的最大值放进了 Repository 中。这时再次启动 Session，并查看 Session Log，在 Session Log 的首行有如下信息：Message: Use persisted repository value [20150524000000] for mapping variable:[\$\$Hiredate]。这条信息显示 PowerCenter 在该次运行中使用了 Persistent Value。这说明再次运行时，PowerCenter 变量 \$\$Hiredate 使用的是上次存放在 Repository 中的变量值。

#### → 注释

细心的读者一定注意到在设计完成的 Mapping 中有一个名为 exp\_dummy 的 Expression 组件。为什么要使用这样一个组件呢？这是因为在 exp\_parameter 中使用了函数 SetMaxVariable，但是 exp\_parameter 的下游组件是 Target，在 Target 中并不会使用 Out\_Hiredate 端口。PowerCenter 为了执行优化的目的，这种没有被下游使用的端口将不被计算。这种情况下 Persistent Values 将不被更新，一直是 Initial value。

## 变量操作函数

PowerCenter 提供了几个变量操作函数，分别是 SetMaxVariable (\$\$Variable, value)、SetMinVariable (\$\$Variable,value)、SetVariable (\$\$Variable,value) 和 SetCountVariable (\$\$Variable)。

SetMaxVariable、SetMinVariable 和 SetVariable 比较相似，都是通过 Value 值更新 \$\$Variable。不同的是函数 SetMaxVariable 从 Value 和 \$\$Variable 中选择最大值更新 \$\$Variable；函数 SetMinValue 从 Value 和 \$\$Variable 中选择最小值更新 \$\$Variable；而函数 SetVariable 直接使用 Value 为 \$\$Variable 赋值。

SetCountVariable 更新 \$\$Variable 的方式与上面的 3 个函数有很大的不同，它是根据 Session 运行时的行标签更新 \$\$Variable 的，如表 4-1 所示。

表 4-1

(row marked for...)	DIST_ID	DISTRIBUTOR	\$\$Variable
(update)	000015	MSD Inc.	23
(insert)	000024	Darkroom Co.	24
(insert)	000025	Howard's Supply	25
(update)	000003	JNR Ltd.	25
(delete)	000024	Darkroom Co.	24
(insert)	000026	Supply.com	25

此表格来自 PowerCenter 手册。当行标签为 Insert 时，变量加 1；当行标签为 Delete 时，变量减 1；当行标签为 Update 时，变量值不变。

### 4.5.3 系统/Session 参数与变量

在学习 PowerCenter 的过程中，最先遇到的 PowerCenter 的参数和变量都是在配置 Session 时，常常看到的如图 4-27 所示的\$PMTargetFileDir、\$PMBadFileDir 等。笔者也经常被问到这些变量有哪些、在哪里定义的、是否可以修改等。

这个问题的答案在 Admin Console 中。打开 Admin Console→Integration Service→Processes Tabs，这些参数的定义尽收眼底。在这里修改对应的值，将会影响使用这个 Integration Service 的所有 Session，如图 4-28 所示。

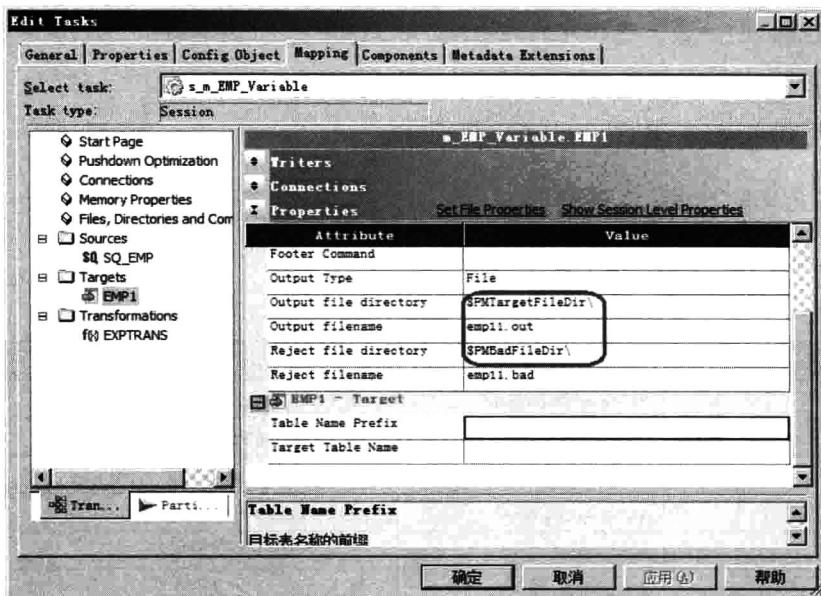


图 4-27

PWC_IS		Properties	Associated R																														
Process Configuration	Process State	Node	Node Status																														
<input checked="" type="checkbox"/> Enabled	<input checked="" type="radio"/> Running	node01_SDHOST1	<input checked="" type="checkbox"/> Available																														
<b>Service Process Properties</b>																																	
<b>General Properties</b> <table border="0"> <tr> <td>Codepage</td> <td>MS Windows Latin 1 (ANSI), superset of Latin1</td> </tr> <tr> <td>\$PMRootDir</td> <td>C:\Informatica\9.6.1\server\infa_shared</td> </tr> <tr> <td>\$PMSessionLogDir</td> <td>\$PMRootDir\SessLogs</td> </tr> <tr> <td>\$PMBadFileDir</td> <td>\$PMRootDir\BadFiles</td> </tr> <tr> <td>\$PMCacheDir</td> <td>\$PMRootDir\Cache</td> </tr> <tr> <td>\$PMTargetFileDir</td> <td>\$PMRootDir/TgtFiles</td> </tr> <tr> <td>\$PMSourceFileDir</td> <td>\$PMRootDir/SrcFiles</td> </tr> <tr> <td>\$PMExtProcDir</td> <td>/ExtProc</td> </tr> <tr> <td>\$PMTempDir</td> <td>\$PMRootDir\Temp</td> </tr> <tr> <td>\$PMWorkflowLogDir</td> <td>\$PMRootDir\WorkflowLogs</td> </tr> <tr> <td>\$PMLookupFileDir</td> <td>\$PMRootDir\LkpFiles</td> </tr> <tr> <td>\$PMStorageDir</td> <td>\$PMRootDir\Storage</td> </tr> <tr> <td>Java SDK ClassPath:</td> <td></td> </tr> <tr> <td>Java SDK Minimum Memory</td> <td></td> </tr> <tr> <td>Java SDK Maximum Memory:</td> <td></td> </tr> </table>				Codepage	MS Windows Latin 1 (ANSI), superset of Latin1	\$PMRootDir	C:\Informatica\9.6.1\server\infa_shared	\$PMSessionLogDir	\$PMRootDir\SessLogs	\$PMBadFileDir	\$PMRootDir\BadFiles	\$PMCacheDir	\$PMRootDir\Cache	\$PMTargetFileDir	\$PMRootDir/TgtFiles	\$PMSourceFileDir	\$PMRootDir/SrcFiles	\$PMExtProcDir	/ExtProc	\$PMTempDir	\$PMRootDir\Temp	\$PMWorkflowLogDir	\$PMRootDir\WorkflowLogs	\$PMLookupFileDir	\$PMRootDir\LkpFiles	\$PMStorageDir	\$PMRootDir\Storage	Java SDK ClassPath:		Java SDK Minimum Memory		Java SDK Maximum Memory:	
Codepage	MS Windows Latin 1 (ANSI), superset of Latin1																																
\$PMRootDir	C:\Informatica\9.6.1\server\infa_shared																																
\$PMSessionLogDir	\$PMRootDir\SessLogs																																
\$PMBadFileDir	\$PMRootDir\BadFiles																																
\$PMCacheDir	\$PMRootDir\Cache																																
\$PMTargetFileDir	\$PMRootDir/TgtFiles																																
\$PMSourceFileDir	\$PMRootDir/SrcFiles																																
\$PMExtProcDir	/ExtProc																																
\$PMTempDir	\$PMRootDir\Temp																																
\$PMWorkflowLogDir	\$PMRootDir\WorkflowLogs																																
\$PMLookupFileDir	\$PMRootDir\LkpFiles																																
\$PMStorageDir	\$PMRootDir\Storage																																
Java SDK ClassPath:																																	
Java SDK Minimum Memory																																	
Java SDK Maximum Memory:																																	

图 4-28

Admin Console 一般是系统管理员的领地。作为程序开发人员，通过 Admin Console 修改这些值的机会比较少，尤其是在分工比较细的组织机构中。因此 PowerCenter 还提供了

大量 Session 级别的参数，根据它们的特性，这些参数被分为两类：

- ① 用户定义的 Session 参数（User-Defined Parameter）。
- ② 内置的 Session 参数（Build-in Parameter）。

### 1. 用户定义的 Session 参数

用户定义的 Session 参数如表 4-2 所示。

表 4-2

参数类型	命名规范	描述
Session Log File	\$PMSSessionLogFile	定义 Session Log 文件名
Number of Partitions	\$DynamicPartitionCount	定义 Session Partition 的数量
Source File	\$InputFileName	定义 Source File 文件名
Lookup File	\$LookupFileName	定义 Lookup File 文件名
Target File	\$OutputFileName	定义 Target File 文件名
Reject File	\$BadFileName	定义 Reject File 文件名
Database Connection	\$DBConnectionName	定义关系型数据库连接的连接名，包括源、目标、Lookup 和存储过程等
External Loader Connection	\$LoaderConnectionName	定义 Loader 连接的名字
FTP Connection	\$FTPConnectionName	定义 FTP 连接名
Queue Connection	\$QueueConnectionName	定义 Queue 连接名
Application Connection \$AppConnectionName	\$AppConnectionName	定义 Application 连接名
General Session Parameter	\$ParamName	通用 Session 参数定义，可以通过此参数定义如 table owner name、table name prefix、FTP file or directory name、lookup cache file name prefix、email address 等

在表 4-2 中，可变更部分以斜体表示，如 *Name*。以 \$DBConnectionName 为例，当定义数据库连接参数时，正体部分不能修改，斜体部分是用户的自定义部分。定义数据库连接有两种方法：一种是直接指定数据库连接；另一种是为其指定一个参数，如图 4-29 所示的 \$DBConnectionScott。这个参数不需要预定义，只要直接使用即可。

#### → 注释

参数不能在 PowerCenter 中预定义，因为在某些情况下有可能因错误书写而造成运行错误。

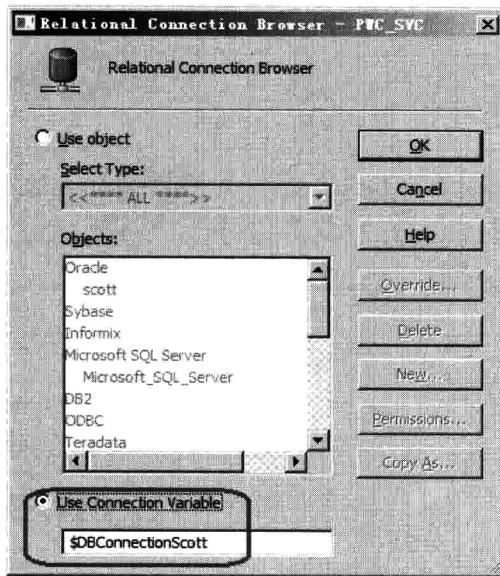


图 4-29

当将数据库连接定义为参数后，需要在参数文件中为此参数赋值，如图 4-30 所示。



图 4-30

使用 Session 用户自定义参数，就可以解决在本节开头提到的场景 2 和场景 3 的问题。当迁移或者批量修改时，仅需修改参数文件即可。

## 2. 内置的 Session 参数

除了自定义参数，PowerCenter 还定义了大量 Session 内置的参数，帮助开发人员在 Post-Session shell commands、SQL commands 和 E-mail 中获取 Session 运行时状态或者信息。这些参数如表 4-3 所示。

表 4-3

参数类型	命名规范
Folder name	\$PMFolderName
Integration Service name	\$PMIntegrationServiceName
Repository Service name	\$PMRepositoryServiceName
Repository user name	\$PMRepositoryUserName
Session name	\$PMSessionName
Session run mode	\$PMSessionRunMode
Source number of affected rows	\$PMSourceQualifierName@numAffectedRows
Source number of applied rows	\$PMSourceQualifierName@numAppliedRows
Source number of rejected rows	\$PMSourceQualifierName@numRejectedRows
Source table name	\$PMSourceName@TableName
Target number of affected rows	\$PMTargetName@numAffectedRows
Target number of applied rows	\$PMTargetName@numAppliedRows
Target number of rejected rows	\$PMTargetName@numRejectedRows
Target table name	\$PMTargetName@TableName
Workflow name	\$PMWorkflowName
Workflow run ID	\$PMWorkflowRunId
Workflow run instance name	\$PMWorkflowRunInstanceName

(1) 以一个 Post Session Success Command 为例，双击 Session→Component Tab，选择 Post-Session Success Command，使用 Type=Non-reusable，可以输入如下命令：

```
echo $PMMappingName >c:\a.txt
```

这样当 Session 执行成功后，会将 Mapping 的名字写入到 C 盘的 a.txt 文件中。

(2) 在 Post SQL 中也可以使用上面的多数变量。双击 Session→Targets，可以在 Post SQL 中使用 SQL，如 insert into Tab\_Session\_log values('\$PMMappingName')，如图 4-31 所示。表 Tab\_Session\_log 是自己创建的一张表，位于 Target 连接串所在的数据库中。在这个 SQL 中可以使用变量 \$PMMappingName。由于表中的字段定义为 Varchar2，因此需要在 \$PMMappingName 前后加单引号。

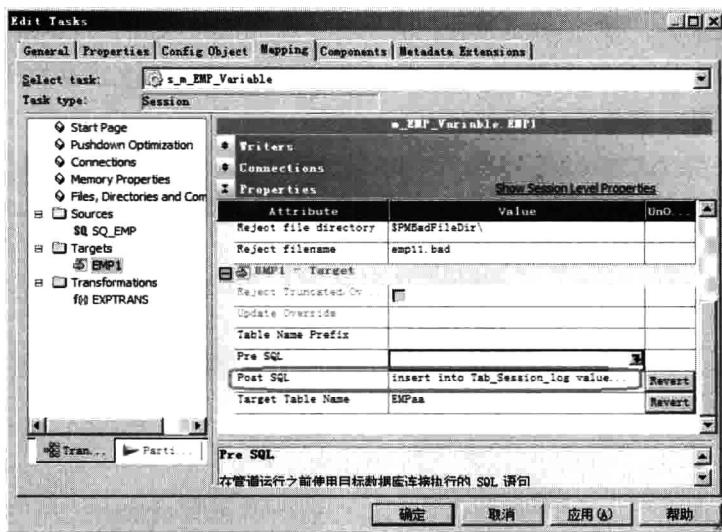


图 4-31

#### 4.5.4 Workflow/Worklet 变量

在介绍 Workflow 的相关 Task 时，曾经介绍过 Workflow 变量。在 Worklet 中同样支持变量。本节使用一个特殊但非常常用的例子来介绍 Workflow 变量的使用，那就是生成带时间戳的目标文件或者文件夹。

**案例：**项目组要求将 Session X 的输出文件写入对应日期的文件夹，如今天是 2015 年 4 月 18 日，输出文件夹为 2015-04-18。如果文件夹不存在，则自动创建文件夹。

(1) 为 Workflow 创建变量 \$\$FOLDER\_NAME。在 Workflow Manager 中，选择菜单 Workflows→Edit→Variables Tab 命令，增加一个变量 \$\$FOLDER\_NAME，类型为 nstring，如图 4-32 所示。

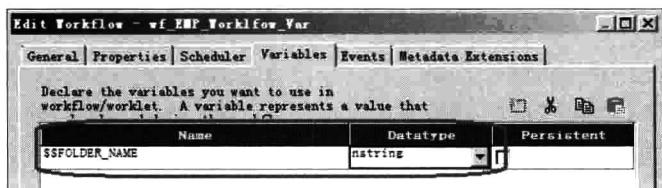


图 4-32

(2) 双击 Session，在 Session Target 的 Properties 中选中 Create Target Directory，它的含义是“如果目录不存在则创建目录”，并将 Output file directory 设置为\$PMTargetFileDir\ \$\$FOLDER\_NAME，如图 4-33 所示。

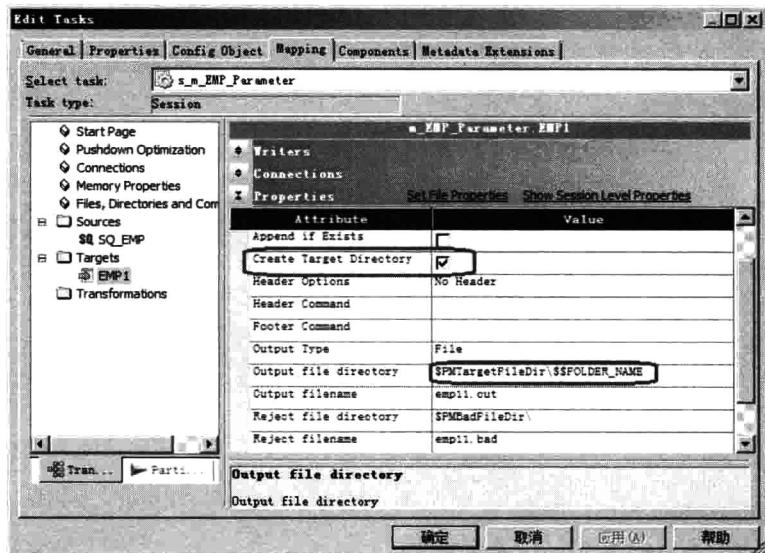


图 4-33

(3) 创建 Workflow 如图 4-34 所示，并使用 Assignment 为变量 \$\$FOLDER\_NAME 赋值。



图 4-34

Assignment 使用如下代码为 \$\$FOLDER\_NAME 赋值：

```

SUBSTR(TO_CHAR(SYSDATE), 1, 2) || '-' || SUBSTR(TO_CHAR(SYSDATE), 4, 2) || '-'
|| SUBSTR(TO_CHAR(SYSDATE), 7, 4)
  
```

(4) 运行 Workflow 查看目标文件夹，对应的文件夹已经创建，相应的文件也被写入到文件夹中，如图 4-35 所示。

名称	修改日期 -	类型	大小
04-19-2015	4/19/2015 2:46 PM	文件夹	

图 4-35

#### 4.5.5 Local 变量 (Local Variables)

Local 变量是 PowerCenter 中的一种特殊用法，用于比较相邻的两条记录的某一个字段的值。如计算相邻记录的某个字段的差，或者比较相邻记录的某个字段的值是否相等。

**案例：**ABC 超市正在进行数据仓库开发，客户服务部门希望了解客户到超市采购的时间间隔，即本次采购距上次采购的间隔天数。假设采购表有 3 个关键字段，分别是 ACCOUNT\_NO NUMBER(4)、ORDER\_AMOUNT NUMBER(10,2)、ORDER\_DATE DATE。

(1) 创建一个 Mapping，如图 4-36 所示。

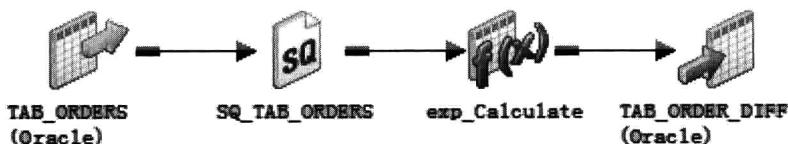


图 4-36

关键点在两个地方：一是 Source Qualifier 中的配置；二是 Expression 中的配置。

(2) Source Qualifier 配置。在 Source Qualifier 中使用 SQL 语句“SELECT ACCOUNT\_NO, ORDER\_DATE FROM TAB\_ORDERS ORDER by ACCOUNT\_NO, ORDER\_DATE”，使用 Order by 语句确保数据是按照 ACCOUNT\_NO 和 ORDER\_DATE 排序的。

(3) 重点是编辑 Expression Transformation。Expression 的设置如图 4-37 所示，v\_LastValue 和 V\_Last\_Order\_date 分别存放了上一条记录的值，而 V\_Current\_value 和 V\_Current\_Order\_date 分别存放了当前记录的值。

在 Day\_DIFF 中使用表达式 “IIF(V\_CurrentValue=v\_LastValue, Date\_DIFF (V\_Current\_Order\_date, V\_Last\_Order\_date,'DD'),0)” 为 Day\_DIFF 赋值。它的含义是：如果 ACCOUNT\_NO 与上一条记录相同，则使用 Date\_DIFF() 函数计算差异天数；如果 ACCOUNT\_NO 与上一条记录不同，即为两个不同客户，则将 Day\_DIFF 赋值为 0。这样就巧妙地利用 Local

Variable 计算出相邻两次购买的间隔天数。



图 4-37

### → 注释

Local 变量使用有两个关键技巧。

- ◎ 假设数据是有序的，可以使用 SQL 语句的 Order By 子句进行排序，也可以通过 Sorter 进行排序。
- ◎ 使用 Local Variables 存放上一条记录的值。基本原理如表 4-4 所示。

表 4-4

Name	Type	Expression
v_LastValue	Variable	V_CurrentValue
V_CurrentValue	Variable	Group_ID
Group_ID	Input/Output	
OUT_Values	Output	IIF(v_LastValue= V_CurrentValue,1,0)

# 第 5 章

# PowerCenter 高级应用

---

PowerCenter 除了在前面提到的 Mapping 设计、Workflow 设计等基本功能，为了提升引擎的执行效率、充分利用系统资源、实现更高的扩展性、提升资源的可用性等，还提供了很多高级特性帮助用户实现最大价值。

## 5.1 任务分区 ( Partition )

任务分区是 PowerCenter 最常用的高级特性之一，Partition 通过并行提高 PowerCenter 的执行效率。PowerCenter 提供的分区类型包括 Database Partitioning、Hash Auto-keys、Hash User-keys、Key Range、Pass Through 和 Round Robin。

开始介绍 PowerCenter 分区之前，先来介绍一下 PowerCenter 的执行原理。默认情况下，一个 Session 在运行时，在服务器上呈现的是一个进程，进程名字为 pmdtm。现实中，有人担心一个进程无法充分利用系统的资源。其实提问者不必有这方面的担心，现在的进程已经不是 20 年前的概念，一个进程同样可以使用一台 64 颗 CPU、256GB 的服务器的所有资源。

这里的原理介绍主要是为了支持后续的 Partition 部分的介绍，首先看一下无 Partition 状态下的一个 Mapping，它的执行过程如图 5-1 所示。

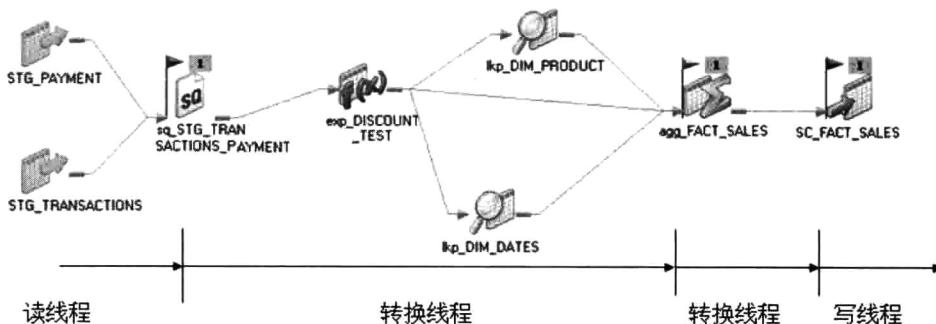


图 5-1

一个 Mapping 就像一个处理数据的流水线(进程)，这条流水线上有若干个工人(线程)在工作。一个最简单的 Pass Through Mapping 由 3 个线程组成，分别是读线程、转换线程和写线程。图 5-1 所示的 Mapping 由 4 个线程组成，包括 1 个读线程、2 个转换线程和 1 个写线程。它的执行过程和流水线一样：

- (1) 读线程读取一组数据，并将这组数据发给转换线程 1；
- (2) 转换线程开始处理第一组数据，同时读线程开始读第二组数据；
- (3) 以此类推，每个线程都在不停歇地工作，并将完成的工作传给下一个线程。

简单地说，就是每个线程完成自己的工作后，就向下传递；传递完成后，不会等待后续线程处理结束，而是继续处理下一组数据。更简单一句话：各个线程间是异步的。

用一个时序图表格解释，可能更容易理解，如表 5-1 所示。

表 5-1

读线程	转换线程 1	转换线程 2	写线程
Row Set 1			
Row Set 2	Row Set 1		
Row Set 3	Row Set 2	Row Set 1	
Row Set 4	Row Set 3	Row Set 2	Row Set 1
.....	.....	.....	.....
Row Set n	Row Set n-1	Row Set n-2	Row Set n-3

如果 Mapping 是一个生产流水线，为了提高性能，有两种方法：

- (1) 在流水线上增加工人;
- (2) 增加相同的流水线。

对 Mapping 而言,

- (1) 流水线增加工人就是增加线程的数量, 即将 Mapping 进行纵向分割。

如图 5-2 所示, 在 Expression 上插上一个小旗子(在 PowerCenter 中叫作增加 Partition Point), 原来的流水线就由 4 个工人变成了 5 个工人。理论上讲, 工作效率应该有一定的提高。

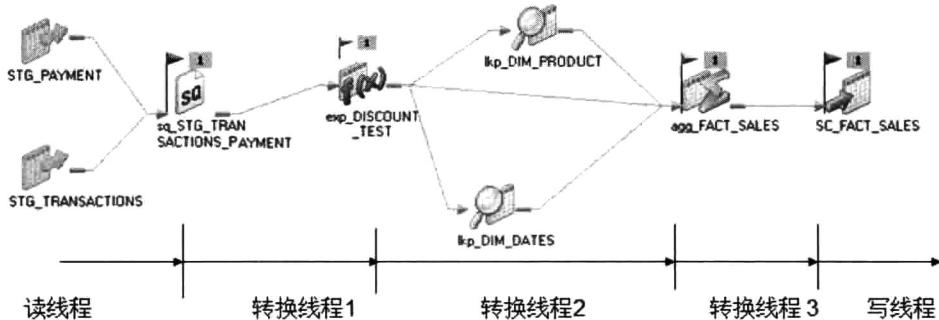


图 5-2

- (2) 增加流水线, 即将 Mapping 进行横向分割, 形成多条流水线。

如图 5-3 所示, 将小旗子顶着的 1 变为 3 (在 PowerCenter 中叫作增加 Partition), 线程的数量将瞬间翻番, 读线程由 1 变成了 3, 转化线程由 2 变成了 6, 写线程由 1 变成了 3。

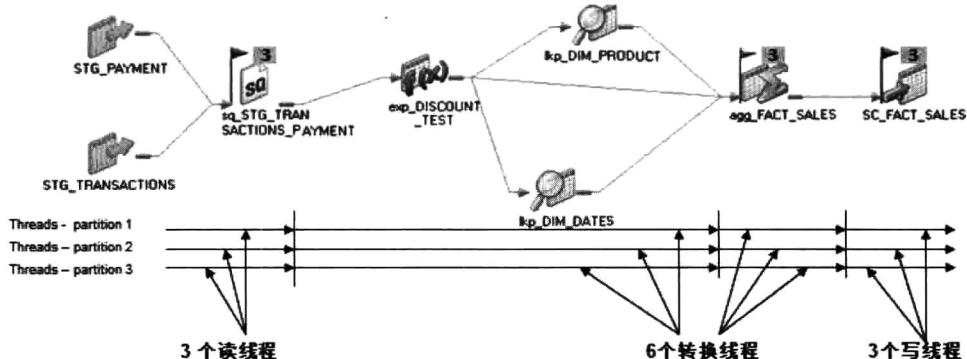


图 5-3

到目前为止，相信大家已经基本了解了 Partition 的功能。在 PowerCenter 中可以认为 Partition 是并行的代名词，不过这个并行既包括横向并行，也包括纵向并行。

### 5.1.1 Database Partitioning

Database Partitioning 是 PowerCenter Partition 和 Database Partition 配合使用的一种利用并行提升性能的方法，目前主要支持的数据库是 Oracle 和 DB2。当数据库的表是 Partition 表时，Integration Service 会读取数据库的 Partition 信息，按照设置的 PowerCenter Partition 数量并行地读取和写数据。也就是说，当数据库表是 Partition 表时，PowerCenter 可以根据数据库的 Partition 数量实现自动并行。

遵循前面的书写方式，仍然以一个实际的例子来演示这个功能。假设数据库为一个有 10 个 Partition 的表，每个分区有 10 万行数据，共有 100 万行数据，如下：

```
Create table empX
(
    EMPNO           NUMBER(10),
    ENAME          VARCHAR2(10),
    DEPTNO         NUMBER(2)
)
PARTITION BY LIST (DEPTNO)
(
    PARTITION PROB_1 VALUES (1),
    PARTITION PROB_2 VALUES (2),
    PARTITION PROB_3 VALUES (3),
    PARTITION PROB_4 VALUES (4),
    PARTITION PROB_5 VALUES (5),
    PARTITION PROB_6 VALUES (6),
    PARTITION PROB_7 VALUES (7),
    PARTITION PROB_8 VALUES (8),
    PARTITION PROB_9 VALUES (9),
    PARTITION PROB_0 VALUES (0)
);
```

现在需要将这张表中的数据以最快的速度导出为一个文本文件。

第一步：首先设计一个最简单的 Mapping，如图 5-4 所示。



图 5-4

实现 Database Partitioning 有两个典型的方法，分别是静态分区和动态分区。下面对这两种分区进行详细介绍。

### 1. 使用 PowerCenter 静态 Partition 实现并行

实例化如上的 Mapping，双击 Session，选择 Mapping Tab，并选择 Partition，如图 5-5 所示。

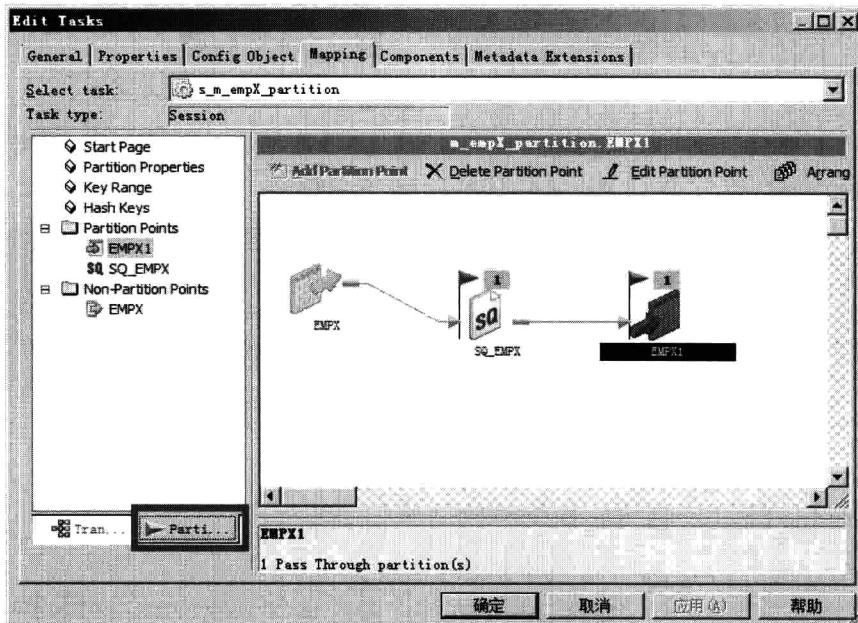


图 5-5

这时候双击 Source Qualifier，添加 Partition，并选择 Partition 类型为 Database Partitioning，如图 5-6 所示。

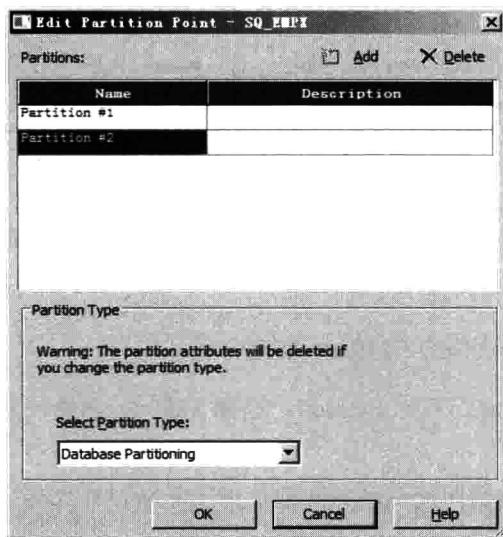


图 5-6

保存上面的配置，运行此 Session，在 Workflow Monitor 中会看到如图 5-7 所示的结果，两个 Partition 同时运行，每个 Partition 读取了 50 万条数据。

Session: emp1_partition [3/5/2015 6:39:07 PM]								
Task Details								
Source/Target Statistics								
Transformation Name	Node	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	
EMPX1	Partition #1	node01_SDUMOST1	500000	500000	0	500000	19000000	19000000
	Partition #2	node01_SDUMOST1	500000	500000	0	500000	19000000	19000000
SQ_EMPX	SQ Partition #1	node01_SDUMOST1	500000	500000	0	500000	19000000	19000000
	SQ Partition #2	node01_SDUMOST1	500000	500000	0	500000	19000000	19000000

图 5-7

打开 Session Log，会看到更详细的日志信息，如下：

```
Message: SQ instance [SQ_EMPX] SQL Query [
SELECT EMPX.EMPNO,EMPX.ENAME,EMPX.DEPTNO FROM EMPX PARTITION (PROB_1) UNION ALL
SELECT EMPX.EMPNO,EMPX.ENAME,EMPX.DEPTNO FROM EMPX PARTITION (PROB_3) UNION ALL
SELECT EMPX.EMPNO,EMPX.ENAME,EMPX.DEPTNO FROM EMPX PARTITION (PROB_5) UNION ALL
SELECT EMPX.EMPNO,EMPX.ENAME,EMPX.DEPTNO FROM EMPX PARTITION (PROB_7) UNION ALL
SELECT EMPX.EMPNO,EMPX.ENAME,EMPX.DEPTNO FROM EMPX PARTITION (PROB_9) ]
```

上述日志信息显示 PowerCenter 一个并行读取了数据库 5 个 Partition 的数据，另外的并行读取了数据库另外 5 个 Partition 的数据。

假如将 PowerCenter Partition 数量设为 3，数据库的 10 个 Partition 也会在这 3 个分区中尽量平均分配，最可能的是 3、3、4 的组合。

当 PowerCenter Session 被分区后，并且目标是文本文件时，默认状态下会生成多个文本文件，因此 PowerCenter 提供了 Merge 选项，可以将生成的多个文件自动合并为一个文件，如图 5-8 所示。

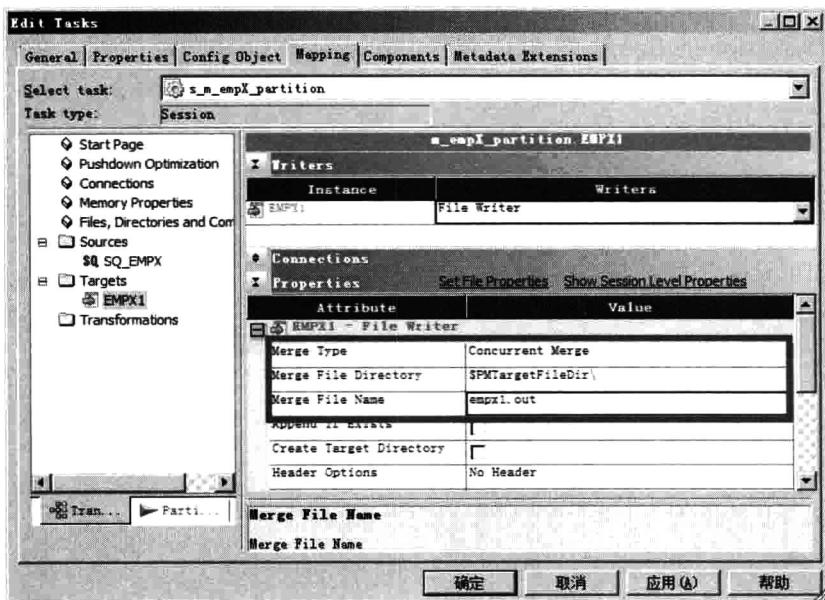


图 5-8

图 5-8 中 Merge Type 共提供了 4 个选项。

- **No Merge:** 不进行合并，这是默认选项。当并行数量是 2 时，生成两个文件。
- **Sequential Merge:** 顺序合并。第一步先生成两个文件，再将两个文件合并成一个文件。性能较差。
- **Concurrent Merge:** 并行合并。在多个文件输出过程中，同时进行合并。性能比较好。
- **File List:** 生成一个文件列表。生成两个文件后，再生成第三个文件，第三个文件的内容是前两个文件的文件名。

这就是所谓的静态 Partition，需要手工增加 Partition 的数量。

## 2. 使用 PowerCenter 动态 Partition 实现并行

当数据库表是 Partition 表时，还有一种方法实现 PowerCenter 的并行，即使用 PowerCenter Dynamic Partition。实现 Dynamic Partition 要完成两步操作。

(1) 双击 Session，选择 Mapping Tab，并选择 Partition Tab，将 Source Qualifier 的分区属性设置为 Database Partitioning。

(2) 选择 Session 的 Config Object，选择 Dynamic Partitioning 的类型，在图 5-9 中选择 Based on number of partitions；Number of Partitions 设置为 4。这时 PowerCenter 将以 4 个并行度运行。

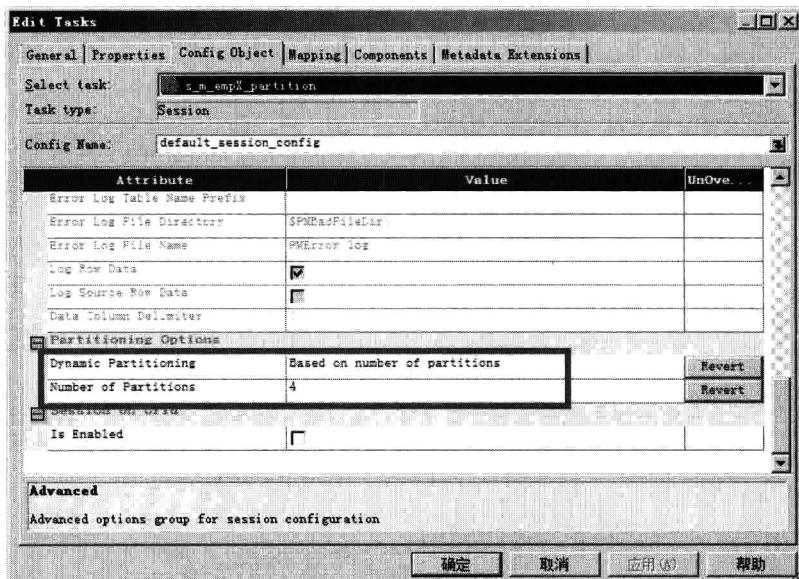


图 5-9

Dynamic Partitioning 还有更多的选项，分别介绍如下。

- ◎ Disabled: 不启用 Dynamic Partition。
- ◎ Based on number of partitions: 启用 Dynamic Partition，并行度由 Number of Partitions 中设置的数值决定。
- ◎ Based on number of nodes in Grid: 启动 Dynamic Partition，这里涉及另外的概念，PowerCenter 本身支持集群，在 PowerCenter 中集群被称为 Grid，集群中的服务器

被称为 Node。当选择这个选项时，Session 的并行度等于节点的数量。

- ◎ Based on Source partitioning：启动 Dynamic Partition，Partition 的数量等于源表的 Partition 的数量。
- ◎ Based on number of CPUs：启动 Dynamic Partition，Partition 的数量等于 ETL 服务器 CPU 的数量。

### 5.1.2 Hash Partitioning

Hash 一般译为“散列”，也有直接音译为“哈希”的，就是把任意长度的输入（又叫作预映射，Pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。Hash Partitioning 就是使用 Hash 函数的输出（散列值）作为分区的依据。Auto-key 和 User-key 就是以散列函数的输入为依据进行分类，如果 Key 是由 PowerCenter 自动选择的就叫作 Auto-key，如果 Key 是由开发人员指定的就叫作 User-key。简单来说，Hash Partitioning 就是使用 Hash 函数，根据输入的 Key 计算其散列值，并据此进行分区。

同样以一个实际计算的样例开始。假设有如表 5-2 所示的一组输入数据，数据量为 1 亿条，要求计算每个部门的工资总额。

表 5-2

EMPNO	ENAME	DEPTNO	SAL
1	Simon	01	100
2	Scott	02	101
3	Andy	01	102
4	Tim	02	103
...	...	...	...

这个 Mapping 的核心组件是 Aggregator，为了提高性能，建议使用 Partition 计算。但是不管是用 Pass Through Partition、Round-Robin Partition 还是 Database Partition，都有可能将同一部门的数据分给不同的分区。而 Aggregator 是按照分区进行计算的，这样同一部门的数据可能在不同的分区中，那么 Aggregator 将永远无法得到正确的结果。这时候需要解决的是如何将同一部门的数据分到同一分区中，解决方法就是 Hash Partitioning。

#### 1. Hash Auto-keys

Hash Auto-keys 是一个非常有用的分区方式，经常会用在 Rank、Sorter、Joiner 和 Unsorted

Aggregator 需要进行分区的情况下。对于上面的例子，使用如图 5-10 所示的分区方式（Hash Auto-keys）将会得到正确的结果。



图 5-10

Hash Auto-keys 类似将数据进行分桶，相同的输入 Key 会产生相同的 Hash 值。对 Aggregator 组件而言，Hash Auto-keys 分区的 Key 是 PowerCenter 自动选择的，分区 Key 就是 Group 组件的 Group by 列。这样就保证了相同 Group 的数据将被分到同一分区，在同一分区进行计算，从而保证了结果的正确性。

在前面及后面的演示中，有一个现象是：在同一个 Pipe Line 中，一个组件增加分区数量，这个 Pipe Line 中所有组件的分区数量都会相应增加。

后面用含 Joiner 组件的一个 Mapping 演示一个 Hash Auto-keys 分区的使用，如图 5-11 所示。

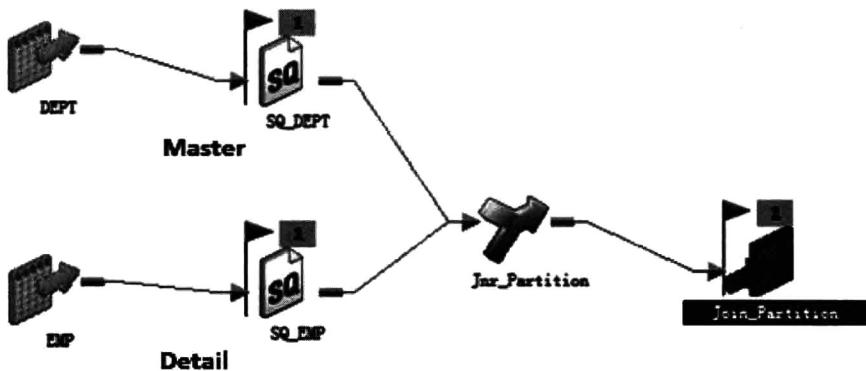


图 5-11

这是一个普通的 Join 运算，在这个 Mapping 中 DEPT 作为 Master，EMP 作为 Detail。

为了提升性能，给上面的 Mapping 添加分区。如果选择 Master 的 Source Qualifier 增加分区数量，PowerCenter 会提示一个错误信息：“The stage connects to the master side of the

joiner transformation which does not allow multiple partition without repartition point”。

### 1) 1 : N Partition

如果选择 Detail 的 Source Qualifier 增加分区，会看到如图 5-12 所示的情况，这就是所谓的 1 : N Partition (Master 部分的 Pipeline 并行度是 1，而 Detail 部分的 Pipeline 并行度是 2)。这时候 Master 会作为一个数据流流入 Joiner 组件。Detail 部分会有 N 个数据流与 Master 的数据创建的索引进行 Join 运算。

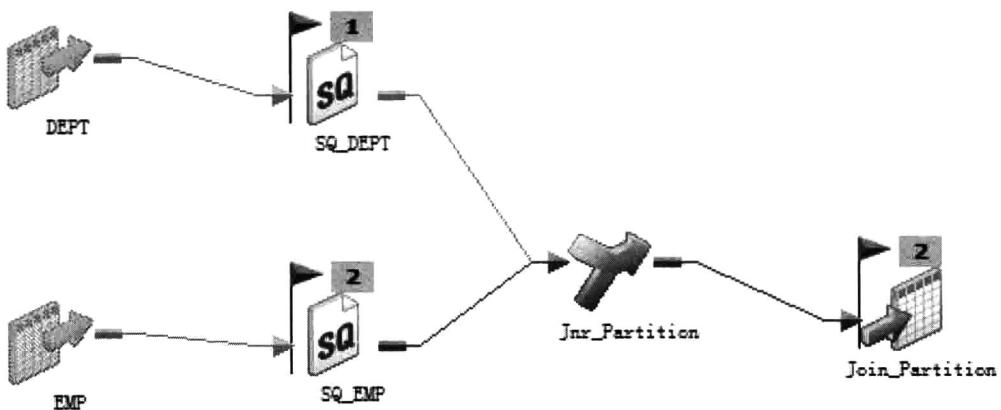


图 5-12

这种情况下，Master 创建索引的效率与未分区前是相同的，但是 Detail 与 Master 关联的效率最高可提升近两倍。

### 2) N : N Partition

如果选择 Joiner 组件增加分区，分区状况将变成如图 5-13 所示的形式 (Master 的 Pipeline 和 Detail 的 Pipeline 各有两个分区)。这时 Master 的分区数也会跟随 Joiner 的分区数变化，所有组件的分区数量相同。

思考一下：Joiner 的分区类型应该是什么？这种情况下 Joiner 的分区类型只有一个选择：Hash Auto-keys。Auto-key 的 Key 就是自动选择 Joiner 组件的关联字段。与前面的 Aggregator 组件一样，这样做的目的是保证可关联字段在数据流动过程中一定能够相遇。

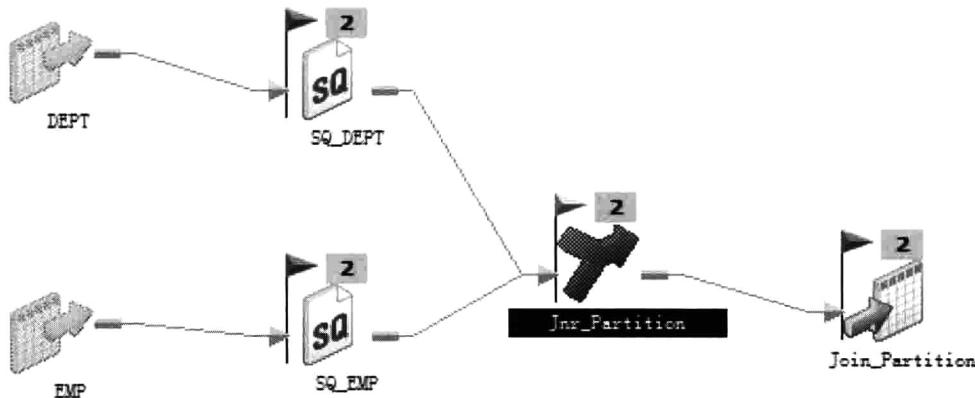


图 5-13

## 2. Hash User-keys

理解了 Hash Auto-keys，再讨论 Hash User-keys 就会变得很容易。最后的目标都是生成 Hash 值，只有一点不同，Auto-keys 是由 PowerCenter 决定使用哪个 Key 作为 Hash 计算的输入，而 User-keys 需要开发人员手工指定 Key。

这种场景用得并不是很多，这里不详细讲述。

### 5.1.3 Key Range Partitioning

Key Range Partitioning 是 PowerCenter 根据指定的某个字段的值的范围进行分区。例如，对员工信息而言，以员工号为 Key，希望将数据进行如下分布。

- Partition #1: 1~2999，包括 2999。
- Partition #2: 3000~5999，包括 3000 和 5999。
- Partition #3: 6000~9999，包括 6000 和 9999。

那么在 Session 中，配置如下，选择的 Key 是员工号（EMPNO），范围如下。

- Partition #1: 最小值~3000。
- Partition #2: 3000~6000。
- Partition #3: 6000~9999。

这里能够看出，可以用空表示下限或者上限。用数学的方式解释这个范围，则右侧是开区间，左侧是闭区间，如图 5-14 所示。

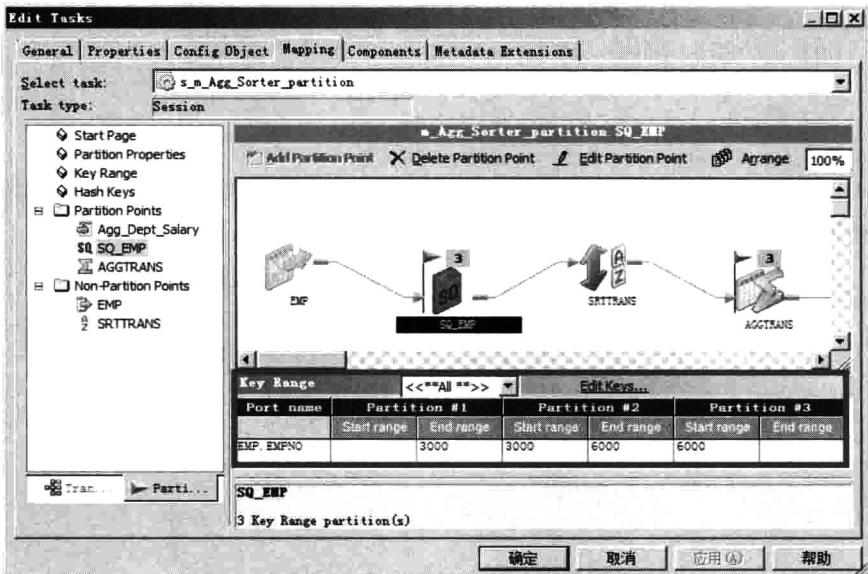


图 5-14

**Key Range Partitioning** 其实也是一个小众方法，用的场景并不多。笔者个人认为主要原因是很多场景无法控制数据的分布情况，或者说每个分区的数据的数量是难以控制的。并且后续组件往往多数不能支持 Key Range 的分区方法，需要对数据进行再分配。

#### 5.1.4 Pass Through Partitioning

通过前面的小节学习，相信大家已经了解了什么是分区及分区的特性。**Pass Through Partition** 是 PowerCenter 中一种最懒惰的分区方式。如果一个组件设置为 Pass Through 分区方式，组件中的 Partition 不会对数据进行任何操作，这个分区接收到什么数据，它会原样向下传递。

以一个极端的例子来说明 Pass-through 分区。同样是读取数据库中的一张表的数据，数据量为 100 万条，并将数据输出成文件。与 Database Partitioning 中的例子不同，这张表可以是分区表，也可以是不分区的表。这个例子的设计过程如下。

第一步：创建的 Mapping 仍然如图 5-15 所示。



图 5-15

第二步：在 Session 的 Mapping Tab 中对 Session 进行分区，例如，设置为两个分区，并且分区类型为 Pass Through，如图 5-16 所示。

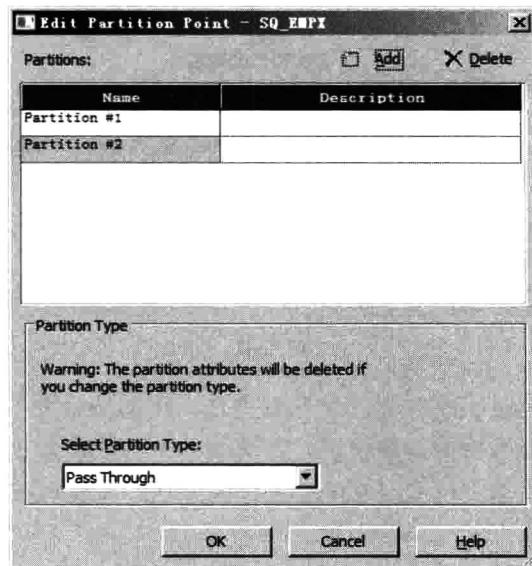


图 5-16

第三步：运行 Session，奇怪的结果出现了。观察 Workflow Monitor 中的统计信息，发现每个分区都读取了 100 万条数据，显然数据被重复读取了，如图 5-17 所示。

Source/Target Statistics								
Transformation Name	Mode	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	
EMPX1	Partition #1	node01_SDUM...	1000000	1000000	0	500000	19000000	38000000
	Partition #2	node01_SDUM...	1000000	1000000	0	333334	12666692	38000000
SQ_EMPX	SQ Partition #1	node01_SDUM...	1000000	1000000	0	500000	19000000	38000000
	SQ Partition #2	node01_SDUM...	1000000	1000000	0	333334	12666692	38000000

图 5-17

解决这个问题有两种方法，这里先介绍其中的一种，另外一种方法由于要与 Round Robin Partitioning 配合，因此放到下一节中进行介绍。

先看第一种方法。

打开 Session 的 Mapping Tab，大家会注意到如果设置了两个 Partition，这时候会有两个线程（Partition #1 和 Partition #2）同时读同一张表的全部数据，这就造成了如上的数据重复读取的问题。这种方法要做的就是为 Partition #1 和 Partition #2 分别限定读取数据的条件。如图 5-18 所示，Partition #1 读取 DEPTNO in (0,1,2,3,4)的数据，而 Partition #2 读取 DEPTNO in (5,6,7,8,9)的数据。

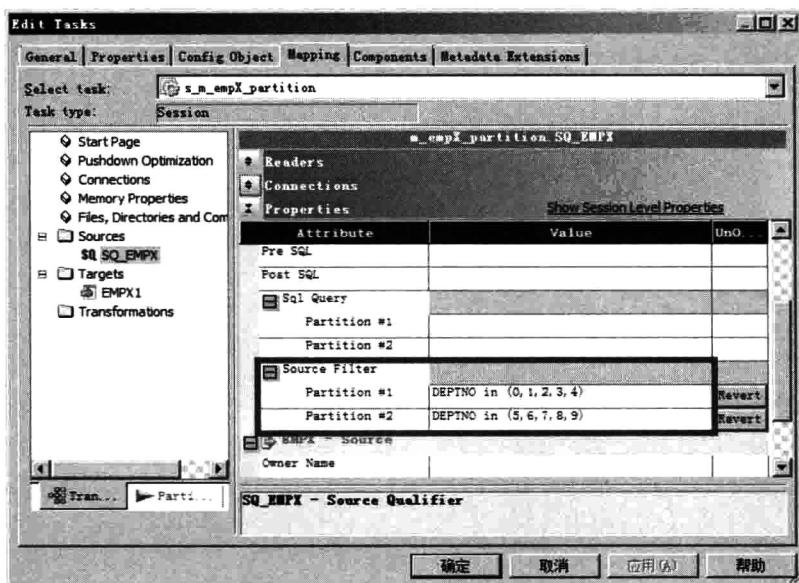


图 5-18

这时出现一个新的问题：如果部门不断增加，未来 DEPTNO 将出现 11、12、甚至更多，这个方法还需要进行手工干预，否则运行结果将会出现问题。另外一个问题：各个 Partition 处理的数据量不均衡，有的处理得多，有的处理得少。但并不是说 Pass Through 没有适合的场景，一个非常好的使用 Pass Through 的例子将在 5.1.5 节介绍。

### Flat File 数据源与 Pass Through Partition

当数据源是文本文件时，使用 Pass Through Partition 是个不错的选择。

**案例：**数据仓库项目组需要将一个包含 100 万条数据的文本文件读取并加载到数据库中。要求使用 PowerCenter，并保证性能最优。由于前面的章节未提及如何创建文本数据源，所以这里先介绍如何创建一个文本数据源。

第一步：创建一个文本数据源。

在 PowerCenter Designer 中选择菜单 Sources→Import from File 命令找到文件所在的目录，选择相应的文件打开，会看到图 5-19。从图中可以看到，提供的模板文件是以“|”分隔的文件包括文件头的文本文件，因此，选择 Delimited（分隔符分隔的文件），而不是 Fixed Width（固定宽度的文件），并勾选 Import field names from first 复选框（这个选项是指通过文件的第一行获取列名）。

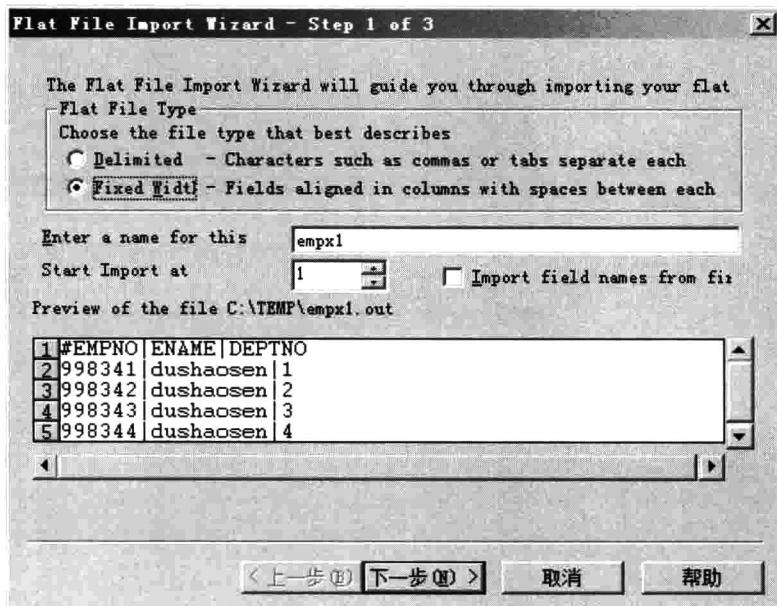


图 5-19

单击“下一步”按钮，输入分隔符“|”，如图 5-20 所示。

继续单击“下一步”按钮，可以看到如图 5-21 所示的界面，如果需要的话，修改一下数据类型，即可完成一个文本数据源的格式导入。

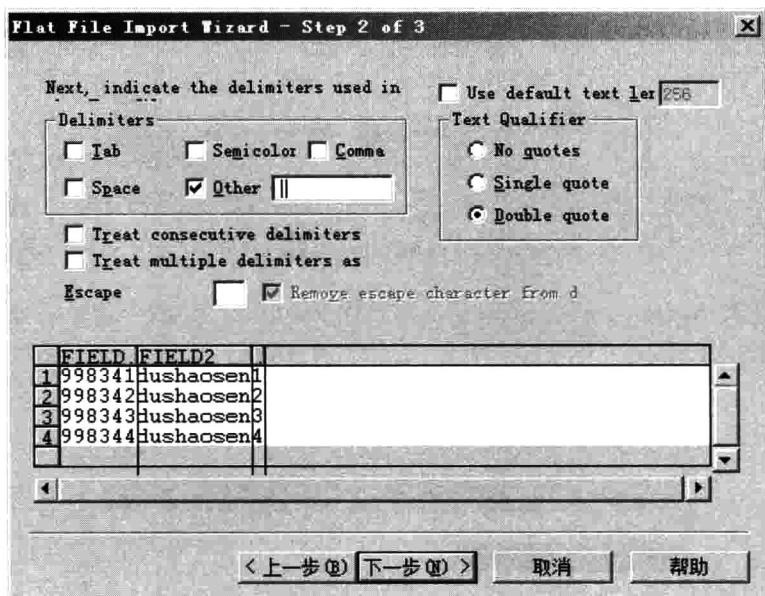


图 5-20

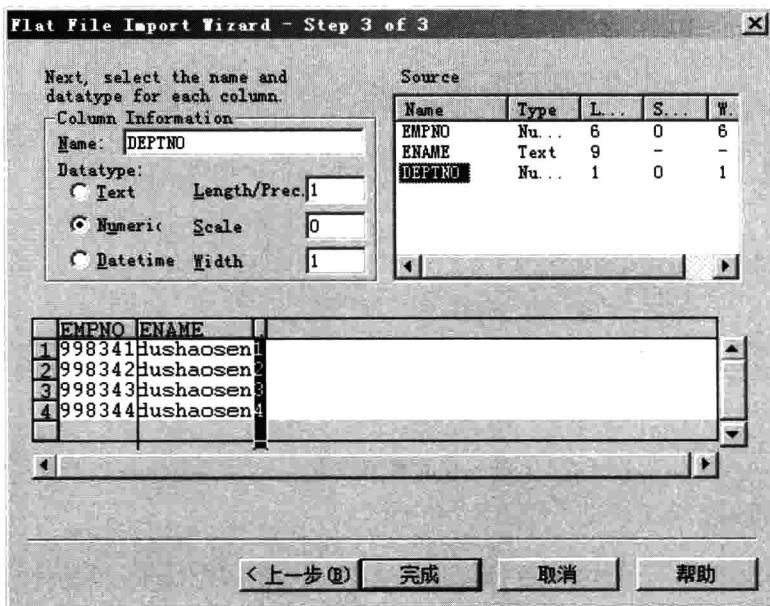


图 5-21

第二步：创建如图 5-22 所示的 Mapping。注意：源是 Flat File，目标是数据库。



图 5-22

第三步：为此 Mapping 创建 Session，并设置 Partition，如两个分区，分区类型为 Pass Through，如图 5-23 所示。

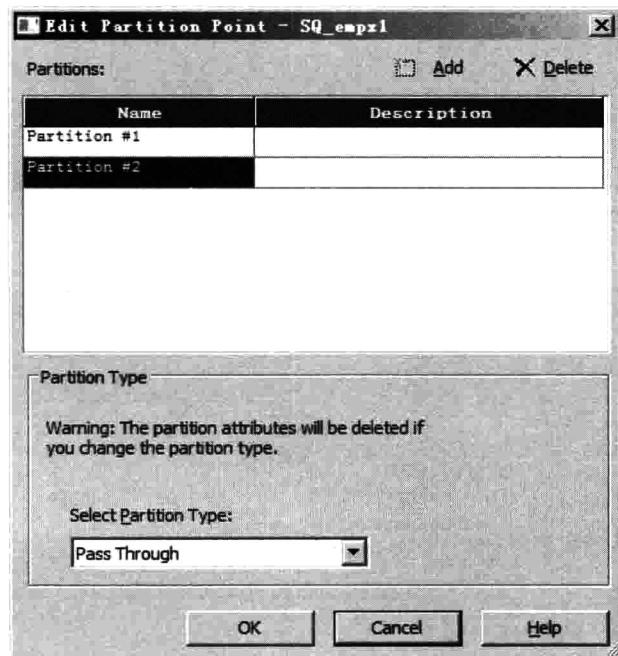


图 5-23

第四步：回到 Session→Mapping Tab 的默认界面，仅在 Partition #1 中指定文件路径和文件名，Partition #2 等均留空，如图 5-24 所示。

第五步：运行对应的 Workflow，观察如图 5-25 所示的运行结果，可以看到每个 Partition 处理的数量约为 50 万行，但并不是精确的 50 万行。

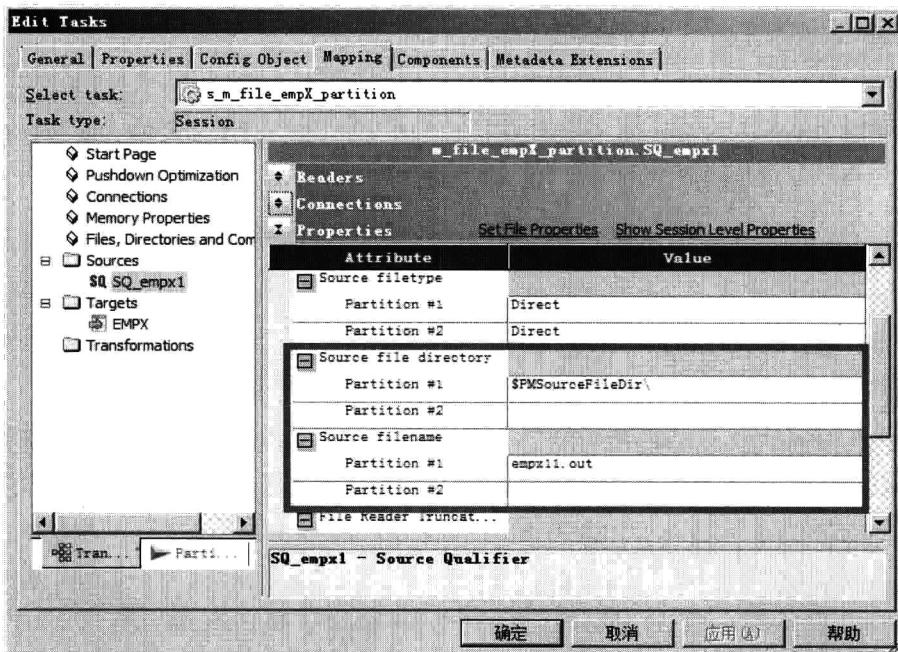


图 5-24

Source/Target Statistics								
Transformation Name	Node	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	
EMPX								
Partition #1	node01_SDUM...	512759	512759	0	170920	6494960	19484842	
Partition #2	node01_SDUM...	487241	487241	0	162414	6171732	18515158	
SQ.empx1								
SQ Partition #1	node01_SDUM...	512759	512759	1	170920	6153120	18459324	
SQ Partition #2	node01_SDUM...	487241	487241	0	162414	5846904	17540676	

图 5-25

### → 注释

按照上面的设计思路和方法，当数据源是文本时，PowerCenter 会自动计算文本文件的大小，然后按照 Partition 的数量将文件进行均分，每个 Partition 读取一份数据。例如包括 100 万行数据的文本文件，分区数量为 2，每个分区处理的数据数量约为 50 万行。

## 5.1.5 Round-Robin Partitioning

使用 Round-Robin Partition 时，Round-Robin 会将其收到的数据块向各个分区分发，当数据量很大时，理论上各个分区得到的数据行数基本相同。因此有时候也简单地解释为

Round-Robin 会向各个分区平均分发数据。

在介绍 Database Partitioning 时，遗留的问题是如果数据库表并没有使用数据库分区特性，如何使用 Database Partition 简化配置、提高性能？这个例子需要将 Database Partition 和 Round-Robin 配合使用。

### 数据库 Partition 的一个特殊、有用的调优方法

对多数的 ETL 操作，一般性能瓶颈出现的顺序首先是 L，然后是 T，最后才是 E。基于这个特性，设计这个样例的 Mapping，它的逻辑是将一个包含 100 万行数据的未分区表的数据导出为一个文本文件。设计这个 Mapping 的大致步骤如下。

第一步：完成的 Mapping 如图 5-26 所示，这个 Mapping 比常规的 Mapping 多了一个 Expression 组件，这个 Expression 组件是一个 Dummy Transformation，没有任何业务的逻辑。



图 5-26

第二步：重点在于 Session 属性中配置 Partition 部分，默认情况下 Partition 如图 5-27 所示。

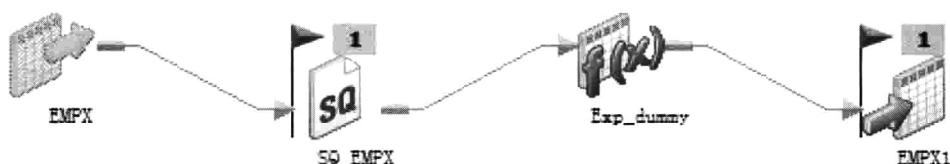


图 5-27

第三步：在这里学习一个新的功能“Add Partition Point”，这个操作就是在 Mapping 的流水线上增加新工人，即增加一个新线程。选中 *Exp\_dummy* 组件，单击 Add Partition Point 按钮，这时可以看到 *Exp\_dummy* 组件上多了一个新的小旗子，如图 5-28 所示。

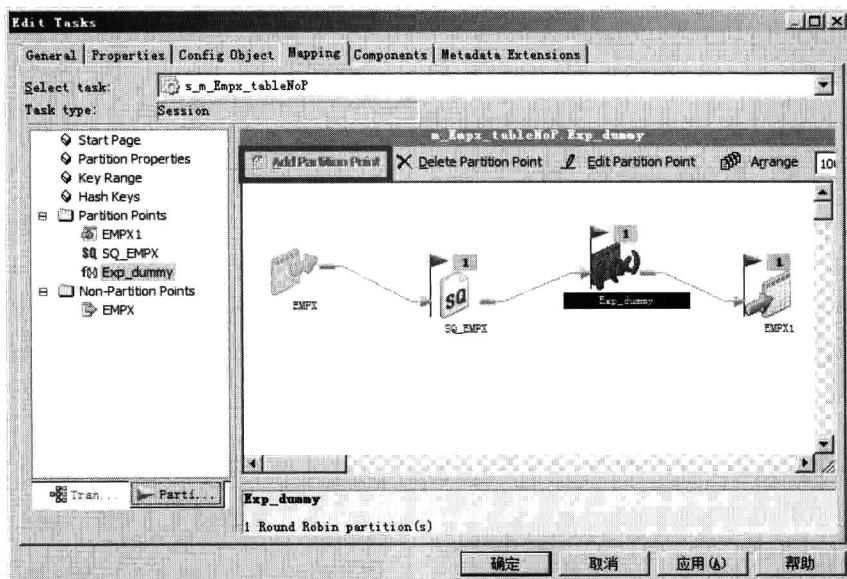


图 5-28

第四步：设置 Partition 类型是样例 Mapping 的关键。将 Source Qualifier 组件设置为 Database Partition，Exp\_dummy 组件设置为 Round-Robin Partition 并保存，如图 5-29 所示。

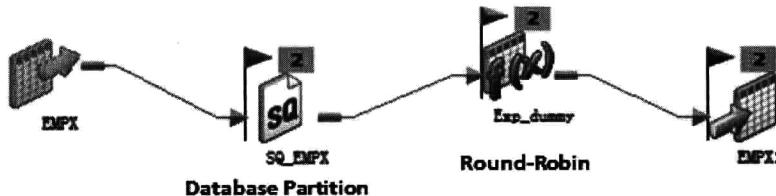


图 5-29

运行其对应的 Session，结果如图 5-30 所示。

Source/Target Statistics								
Transformation Name	Node	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	
EMPX1	Partition #1 node01_SDUMH...	502348	502348	0	502348	19089224	19089224	
	Partition #2 node01_SDUMH...	497652	497652	0	497652	18910776	18910776	
SQ_EMPIX	SQ Partition #1 node01_SDUMH...	1000000	1000000	0	1000000	38000000	38000000	
	SQ Partition #2 node01_SDUMH...	0	0	0	0	0	0	

图 5-30

Source Qualifier 的 Partition #1 读取了全部 100 万行数据, Partition #2 没有读任何数据。而 EMPX 写的 Partition #1 和 Partition #2 均收到了约 50 万行数据。

→ **注释**

这个 Session 的执行过程是这样的: Source Qualifier 使用 Database Partition, 但是因为这张表并没有分区, PowerCenter 就认为表只有一个分区, 因此在 Monitor 中看到的是只有一个分区在读数据, 但它读取了全部数据。当数据传送到 Exp\_dummy 组件时, 数据按照 Round-Robin 进行了平均分配, 因此每个后续组件的分区都处理了约 1/2 的数据量。在真实的 Mapping 中, 不一定需要这个 Dummy Transformation, 可以选择 Source Qualifier 后任意支持 Round-Robin 的 Transformation 进行设置。

这样做有以下两个好处。

- (1) 提升了 T (Transformation) 和 L (Load) 的并行度与总体性能。
- (2) 不用在 Session 的 Source Filter 中增加独立的 Filter 语句, 例如 DEPTNO in (0,1,2,3,4) 和 DEPTNO in (5,6,7,8,9)。这样做的优势是显而易见的, 即使 DEPTNO 增加或者减少都不用修改 Filter, 同时也不用考虑 DEPTNO in (0,1,2,3,4) 和 DEPTNO in (5,6,7,8,9) 包含的数据量是否接近。

## 5.2 内存管理

PowerCenter 的内存管理并不复杂, 但由于 ETL 是一个高 I/O 的工作, 这种情况下了解 PowerCenter 内存使用的机制就变得非常重要, 通过合理地使用内存可以降低 I/O, 并提升 PowerCenter 的总体性能。

在 PowerCenter 中有以下两个与内存管理有关的地方。

- (1) DTM 内存。
- (2) Transformation Cache。

一个 Session 进程使用的内存总量=DTM + Transformation Cache。

### 5.2.1 DTM 内存

DTM Buffer 用于支持 Session 的 reader、writer 线程将数据从源传送到目标的过程（在有些文档中也提到，DTM Buffer 还支持 Session 中 Log 生成需要的内存）。DTM Buffer 是由大量的 DTM Block 组成的。

再强调一下 DTM Buffer 是在 Session 级别来管理的，因此它的增加、降低都是在 Session 的属性里，任何值的修改仅仅影响当前 Session。如图 5-31 所示，双击任意 Session→Mapping→Memory Properties。

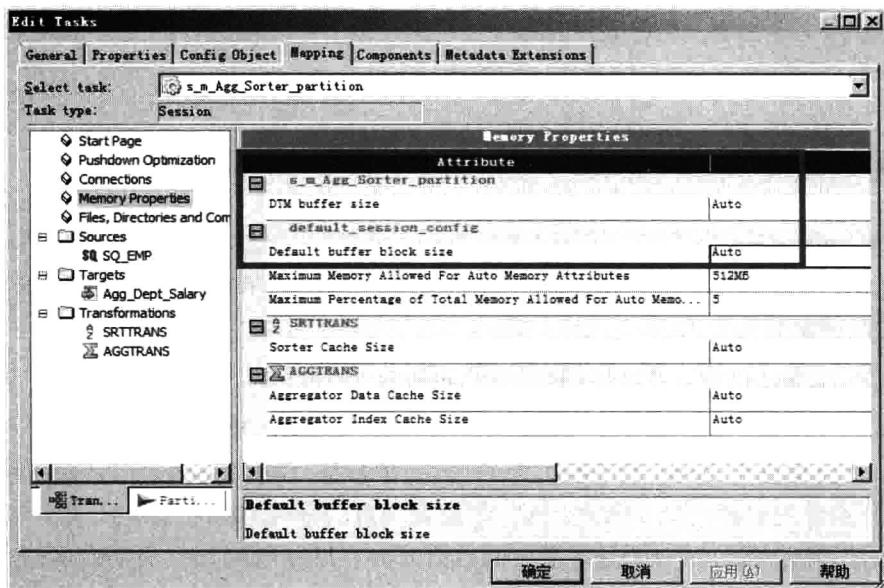


图 5-31

(1) DTM buffer size：它的默认值是 Auto，这时候将由 Integration Service 决定它的大小，默认值是 12MB。也可以根据需要修改这个值为 128MB 或者其他值。出现下面的几种情况时，就需要考虑调整 DTM Buffer 的大小了。

- ◎ 当 Session 使用 Partition 时。
- ◎ 源和目标包括大量的比较长的字符串类型时。
- ◎ 源或者目标包括 Clob 或者 Blob 时。
- ◎ 存在 XML 源或者目标时。

但 DTM Buffer 调优确实没有一个很好的方法，现实中笔者一般是尝试修改为 128MB、64MB 或者 32MB，如果在某个范围内发现性能有明显变化，则在这个范围内进行微调，否则继续尝试其他值。同时从 DTM Buffer 的用途也可以看出，它的值不可能是一个非常大的值。

(2) DTM buffer block size: Block Size 就是组成 DTM Buffer 每个内存块的大小，每个读/写线程至少需要两个数据块。PowerCenter 有一种性能瓶颈，是由于将一条数据分割放在两个 DTM Block 中，也就是在这种情况下，才需要修改 Block Size 的大小。这种情况发生时，优化也比较简单，只需计算所有源、目标和 Transformation 的单行数据的长度，将 Block Size 设置为更大的值。一般来讲，如果一个 Block 容纳的数据条数小于 10 条时，就会出现明显的性能瓶颈。相对优化的方式是设置一个 Block 能够容纳大于 100 条的数据甚至更多。

## 5.2.2 Transformation Cache

PowerCenter 的很多组件都有自己独立的 Cache 设置，包括 Joiner、Lookup、Rank、Sorter 和 Aggregator 等，它们是整个 Session 使用内存的一部分。Transformation Cache 的调整对 Session 的性能影响比较大。

PowerCenter 的排序、创建索引都与 Cache 相关。以一个 Joiner 组件为例，当 Master 表的数据被读入 PowerCenter 之后，Joiner 组件会利用 Cache 建立索引。当 Cache 用完时，PowerCenter 会将数据 Dump 到磁盘上（对有的操作系统，比如 Windows，笔者注意到 Dump 过程对操作系统的资源消耗非常大。这一现象表现为，PowerCenter 会将此前在内存中的数据完全 dump 到磁盘后，才开始处理后续的数据）。并且只有当 Master 表的索引建立结束后，才开始读取 Detail 表的数据，并将 Detail 表的数据与 Master 表的 Cache 关联。

同样，如何设置 Transformation Cache？如图 5-32 所示。

以 Joiner 为例，它的 Cache 包括两个值：一个是 Data Cache Size，另一个是 Index Cache Size，它们的大小是怎么计算得来的？请参考下面的公式。

### Data Cache Size:

$$(Number\ of\ master\ rows) \times (\text{row}\ size + 8)$$

其中，row size 是指包括 Join 条件字段与其他向下游传递的字段的长度和。

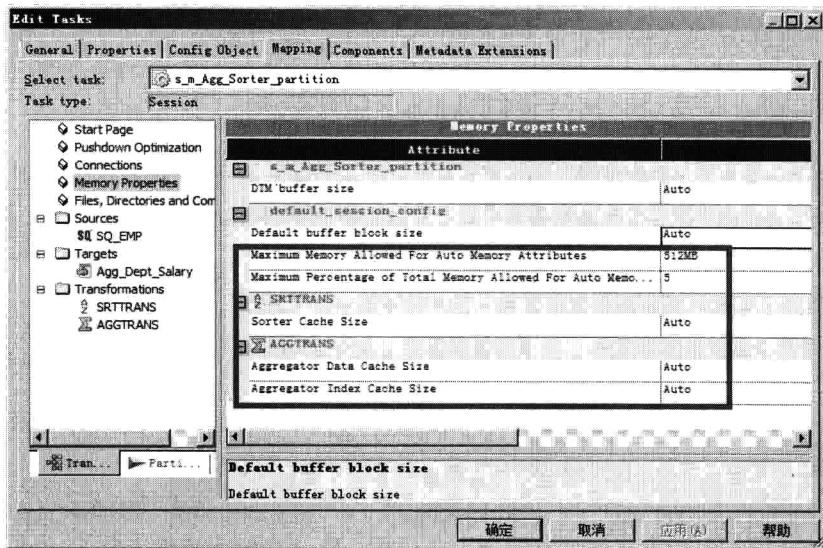


图 5-32

### Index Cache Size:

$$(Number \text{ of } master \text{ rows}) \times (\text{row size} + 16)$$

其中，row size 是指 Join 条件字段的长度。

在 PowerCenter 旧版本的介绍中，所有包含 Cache 的组件都有类似的计算公式。新的版本将这部分内容做了很大的简化，如果开发人员希望手工设置 Cache 大小，只需要单击 Auto 后的小箭头，就会看到如图 5-33 所示的对话框。

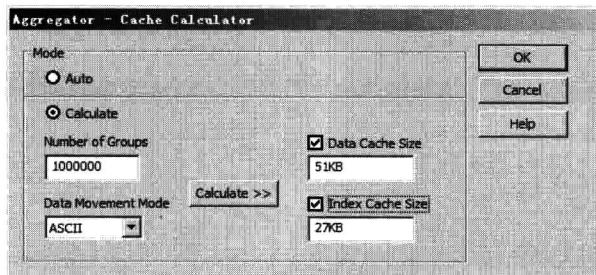


图 5-33

执行以下几步，可以迅速获得比较可靠的 Data Cache Size 和 Index Cache Size：选择 Calculate 单选按钮 → 输入 Number of Groups（一般是行数）→ 选择 Data Movement Mode

(ASCII or Unicode)，这个选择要与 Integration Service 上的配置一致→单击计算按钮即可算出 Data Cache Size 和 Index Cache Size。这个功能使得 Cache 计算变得非常简单，但是这个值不一定是精确的，有时会偏小一点点，造成性能没有得到实质提升。

如果你发现虽然计算并设置了 Cache 值，但这个组件的性能没有得到实质性提升，这时候请打开 Session Log。这种问题发生时，一般情况下会在 Session Log 中有一条提示信息，对 Cache 大小提出建议。

在一个 Session 中，使用了多个有 Cache 的组件，这种情况下并不需要对所有组件的 Cache 进行一一调整。PowerCenter 提供了两个参数来管理使用 Auto 时的默认值。

- Maximum Memory Allowed for Auto Memory Attributes：当 Transformation 的 Cache 设置为 Auto 时，其可以使用的最大内存大小。
- Maximum Percentage of Total Memory Allowed for Auto Memory Attributes：当 Transformation 的 Cache 设置为 Auto 时，其可以使用的最大物理内存的百分比。

最终设置为 Auto 的 Transformation Cache 的大小会选择这两个参数的最小值。

在一个项目中，使用 Joiner、Lookup、Sorter 等组件的 Mapping 非常多，对每一个组件进行手工调整更不现实，尤其是很多组件需要的 Cache 并不多，没有必要进行一一计算。这时就需要一个统一的配置影响所有 Session 的默认 Auto 设置。PowerCenter 提供了这样的能力，步骤是：打开 Workflow Manager→Tasks→Session Configuration，编辑 default\_session\_config，可以看到如图 5-34 所示的对话框。

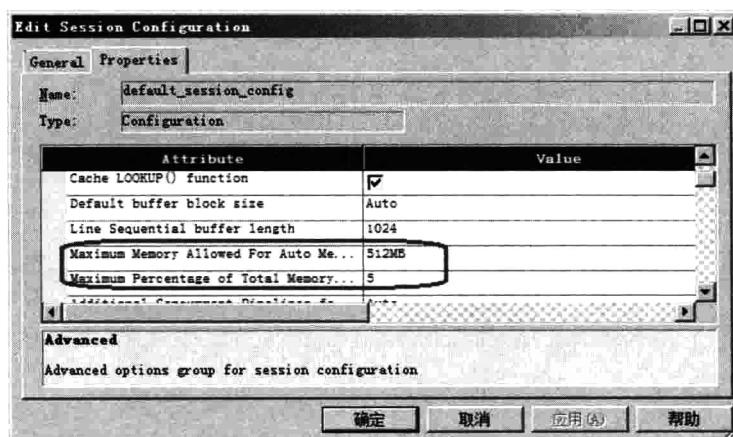


图 5-34

修改这些参数可影响所有 Session 的默认配置。这个方法可以做到一招制敌，一次性修改所有 Session 的默认设置。

#### ➔ 注释

在项目中，最佳实践是：(1) 通过参数 Maximum Memory Allowed for Auto Memory Attributes、Maximum Percentage of Total Memory Allowed for Auto Memory Attributes 对付绝大多数的 Cache；(2) 对需要超大 Cache 的组件进行手工调整，以实现满足性能要求的同时工作量不太大的目的。

## 5.3 网格计算

PowerCenter 的网格计算（Grid）即 PowerCenter 的集群功能。集群功能最大的作用在于提升了 PowerCenter 的扩展能力，使 ETL 的开发人员开发的程序可以在不需要修改的情况下利用其扩展能力，提升处理能力。这种能力在大数据时代尤为重要。在大数据时代，各个组织都在推进业务数字化，提升组织洞察力，同时带来的是数据呈几何倍数的增长。同时，在大数据时代，人们利用数据的意愿在增强，包括使用内、外部数据的意愿，使用更多的历史数据的意愿等，这就是笔者经常向客户推荐 PowerCenter 的集群能力的原因。

同时，PowerCenter 除了在数据仓库作为 ETL，还经常被用作企业数据交换平台的核心组件，这样的应用场景对 PowerCenter 提出了更高的要求，比如：

- 减少非正常宕机时间。
- 减少由于系统维护产生的系统停机。
- 提升扩展能力。
- 提高服务器处理能力。

这些正是 PowerCenter Grid 所能提供的。接下来我们将对 PowerCenter 的 Grid 能力做一个简要的介绍。

### 5.3.1 Grid 架构

谈到 Grid 首先要讨论 Grid 架构，这部分可以认为是第 1 章的延续，是对 PowerCenter

架构的进一步阐述。一图胜千言，首先奉献一幅 PowerCenter Grid 架构图，在此架构图的基础上介绍 Grid 的基础架构，如图 5-35 所示。

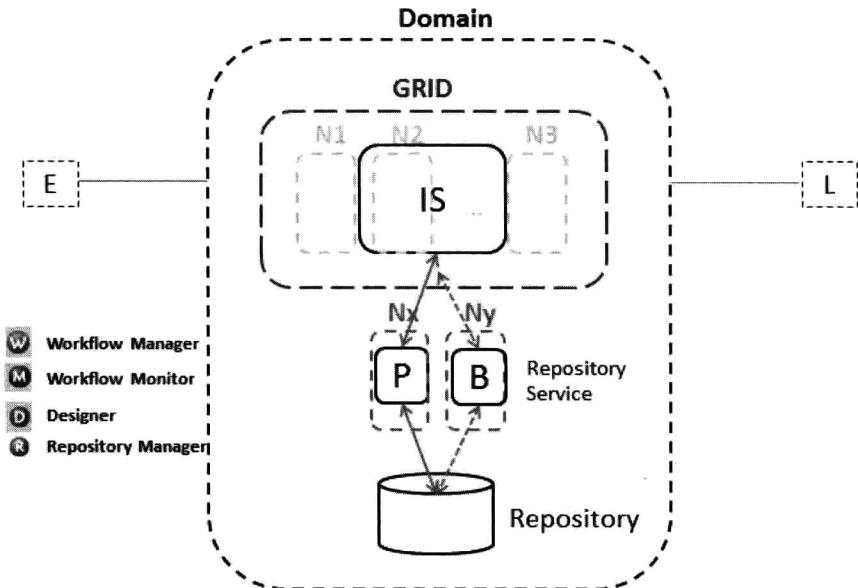


图 5-35

看到这张图相信已经有读者感到有些凌乱了，下面将此图展开来作一些详细的介绍。

(1) **Domain**: 一组管理进程或者线程，用于管理和协调 Domain 中的所有服务。它是在安装 N1 (第一个节点) 的过程中创建的，即第一次安装过程中选择 “Create Domain”。

(2) **Grid (网格)**: 由若干个节点 (N1、N2、N3，但不限于 3 个) 组成。映射到安装配置过程，分为两个步骤：① 创建 Domain 时添加节点，即安装 N1 时选择“Create Domain”，安装 N2、N3 时需要选择“Add into Existing Domain”，这时 Grid 尚未被创建，还需要执行第二步，即创建 Grid；② 在 Admin Console 上创建 Grid，并且把 N1、N2、N3 作为它的成员。一个 Domain 可以包含多个 Grid。

(3) **IS (Integration Service)**: Integration Service 可以创建在 Grid 或者 Node 上，只有创建在 Grid 上的 Integration Service 才支持集群，这是在创建 Integration Service 时选择的。创建在 Grid 上的 Integration Service 逻辑上是一个名字，但是这个 Integration Service 会在集群内的所有节点上各运行一个进程。同时，一个 Grid 上可以创建多个 Integration Service。

但是实际使用中这种情形并不多，只有特殊的需求才会这么做，比如特殊字符集或者 Integration Service 需要不同的环境变量时。

(4) Repository Service：主要是负责与 Repository 交互的协调工作，所以一般情况下压力都很小，因此没有 Grid 方式。但是当 Integration Service 采用 Grid 时，会建议 Repository Service 采用 HA（高可用性）方式。这样可以保证当 Repository Service 的一个节点失效时，另一个节点能及时地接管此前的工作。P 指 Primary，即主节点。B 指 Backup，即备份节点。Nx 和 Ny 可以是 N1、N2、N3 中的一个节点，也可以是其不相关的其他节点。

#### → 注释

(1) Gateway 设置几个合适？一般来讲最好是所有的节点都设置为 Gateway，这样就能保证，假如域中有  $n$  个节点，当  $n-1$  个节点失效时，还能够访问。

(2) 共享存储问题。Grid 需要共享存储支持，比如 SAN、NAS、NFS 等。这时一般是性能和价格的平衡。曾经我们认为 NFS 的性能比较差，但在网络状况极佳的情况下性能也非常好。但是使用 NFS 还是要考虑是否有单点失效的问题。

(3) 哪些目录需要放到共享存储上？最简单的办法是把./server/infa\_shared/下的所有目录都放到共享存储上。

(4) 最少需要将哪些目录放到共享存储上？包括\$PMStorageDir、\$PMLookupFileDir、\$PMSourceFileDir、\$PMTargetFileDir 和\$PMCacheFileDir。

(5) N1、N2、N3 机器上的用户名必须相同，如果是 UNIX/Linux 操作系统，用户 ID 和组 ID 也必须相同。ID 指的是使用 UNIX/Linux 命令 id 显示的用户编号和组编号，如 501 等。

### 5.3.2 Grid 负载均衡

PowerCenter 负载均衡包括两种模式：Workflow on Grid 和 Session on Grid。

Workflow on Grid 是将 Grid 中的所有节点当作资源池，以 Tasks 为单位进行任务分发，确保充分利用 Grid 的资源。这种模式是默认方式。

Session on Grid 是将 Grid 中的所有节点当作资源池，以 Session 的 Partition 为单位进行任务分发。这部分将在 5.3.3 节进行详细阐述。

首先了解一下 Grid 支持的任务分发模式及其相关的概念。Grid 提供了 3 种基本的任务

分发方式，分别是 Round-Robin、Metric-Based 和 Adaptive。

任务分发模式是 Domain 的属性，而不是 Grid 的属性，这一点需要特别留意。尤其是在 Domain 中存在多个 Grid 的情况下，一旦设置了 Domain 的任务分发模式，这个 Domain 中的所有 Integration Service 均将采用这一设置。

### 1. Round-Robin 模式

在这种模式下，Load Balance 分发器以 Round-Robin 模式进行任务分发。Load Balance 管理器检查 Maximum Processes（Maximum Process 是 Node 的属性，在 Admin Console 中进行管理）阈值设置，如果增加当前任务不会超过它的阈值设置，这个任务将被分配给这个节点执行；如果增加此任务会导致超过某个阈值，Load Balance 管理器将继续寻找可用的服务器，直到找到为止。

在 Round-Robin 模式下，Load Balance 管理器不会 Bypass 任何任务。如果一个资源需求密集的任务被提交，而且所有任务优先级均相同的情况下，有可能出现所有的任务都需要等待这个资源密集任务被分配的情况（其实这种情况几乎不可能发生，因为在这种模式下，它申请的资源仅仅是进程数一个值，这样的需求很容易满足）。

这种模式一般用在节点资源比较平均的情况下。如果节点配置差别较大，就有可能将资源需求密集的任务分配给配置较差的服务器。

### 2. Metric-Based 模式

在这种模式下，Load Balance 分发器还是以 Round-Robin 模式进行任务分发。这时，Load Balance 管理器会检查所有的资源阈值设置，同时检查 Swap 空间。如果要分发任务的资源需求超过评估节点的剩余资源，任务将不会被分配。Load Balance 管理器会检查其他节点，直到发现有足够资源的节点，然后将此任务分配给此节点。

在这种模式下，PowerCenter 会自动统计 Task 最近的 3 次运行所需的资源，从而决定该任务需要的资源。如果是首次运行，PowerCenter 会使用默认值 40MB memory 和 15% CPU。

在 Metric-Based 模式下，Load Balance 管理器同样不会 Bypass 任何任务，如果一个资源密集的任务被提交，而且所有任务优先级均相同的情况下，有可能出现所有的任务都需要等待这个资源密集任务被分配的情况（这种情况的确会发生）。

### 3. Adaptive 模式

在这种模式下，PowerCenter 会评估所有 Node 的资源可用性。它会使用 CPU 空闲最多的 Node，同时评估所有的阈值和 Swap 空间。如果分发该任务不会超过阈值设置，任务将被分发。

在这种模式下，PowerCenter 会使用 CPU Profile 和任务运行的最近 3 次资源的统计值。如果 Repository 中尚无资源需求统计值，同样，它会使用默认资源需求 40MB memory 和 15% CPU。

在 Adaptive 模式下，Load Balance 管理器根据任务资源需求和任务优先级决定任务的分配。例如，大量有相同优先级的任务在分发队列中等待，并且无节点能满足一个资源需求密集型任务，这时，Load Balance 管理器为资源需求密集型任务保留一个节点，而继续对队列中的其他任务进行分发，这样就可以避免其他任务等待资源需求密集任务的情况。

一个节点的资源设置包括如下项目，它们被设置在节点的属性中：

- Maximum Processes。
- Maximum CPU run Queue Length。
- Maximum Memory %。

打开 Admin Console，在 Node→Properties Tab 中可以看到如图 5-36 所示的属性。

<b>Resource Provision Threshold</b>	
Maximum Processes	10
Maximum CPU run Queue Length	10
Maximum Memory %	150

图 5-36

这里还需要了解两个概念：Service Level 和 CPU Profile。

(1) Service Level：在 Domain 中进行定义，在 Workflow 中使用。这就意味着，一个 Workflow 中的所有 Task 均使用相同的 Service Level。在分发队列中，Load Balance 管理器将优先分发高优先级的任务。假如，在 Domain 中预定义了两种 Service Level，如图 5-37 所示。

Service Level Management		
Service Level Name	Dispatch Priority	Maximum Dispatch Wait Time(sec)
Default	5	1800
Low	6	3600

图 5-37

Default 作为高优先级, Dispatch Priority 为 5, Maximum Dispatch Wait Time 为 1800 秒; Low 作为低优先级, Dispatch Priority 为 6, Maximum Dispatch Wait Time 为 3600 秒。这里需要解释两个问题: 第一, 数值越小优先级越高; 第二, Maximum Dispatch Wait Time 的含义是什么? 因为在分发队列中, 高优先级任务首先被分发, 可能会出现低优先级任务等待的情况, 因此当等待时间超过 Maximum Dispatch Wait Time 时, 这个任务将被设为最高优先级, 这就是 Maximum Dispatch Wait Time 的作用。

(2) CPU Profile: 它是根据 PowerCenter 提供的基线来评估 CPU 能力的一个值。Baseline 系统使用 Pentium 2.4 GHz CPU 运行 Windows 2000 的环境进行评估。例如, 一台 SPARC 480 MHz 计算机, 它是基线环境性能的 0.28 倍, 那么它的 CPU Profile 就是 0.28。

这个值是 PowerCenter 进行计算的, 具体计算方法是: 选择 Node, 选择 Actions → Recalculate CPU Profile Benchmark。

CPU Profile 仅用于 Adaptive 模式下, 对其他的模式无效。

最后用一个表格来对比这 3 种分发模式, 如表 5-3 所示。

表 5-3

Dispatch Mode	Checks resource provision thresholds?	Uses task statistics?	Uses CPU profile?	Allows bypass in dispatch queue?
Round-Robin	Checks maximum processes	No	No	No
Metric-Based	Checks all thresholds	Yes	No	No
Adaptive	Checks all thresholds	Yes	Yes	Yes

### 5.3.3 Grid 与任务分区 (Partition)

5.3.2 节提到一个概念, 叫 Session on Grid, 它的作用是将一个 Session 分发到多个节点

上执行，从而提高 Session 的执行效率。前面的章节曾经提到，Session 在运行时一般表现为服务器上一个名为 pmdtm 的进程，那么如何将一个进程分发到多台服务器上呢？在 PowerCenter 中配置 Session on Grid 有两种不同的组合，分别是：

- Session on Grid + Partition。
- Session on Grid + 资源选项。

(1) Session on Grid：启动 Session on Grid 选项非常简单，它是 Session 的一个属性。双击需要启动 Session on Grid 的 Session，在 Config Object 的最下方选择 Enable Session on Grid 即可，如图 5-38 所示。

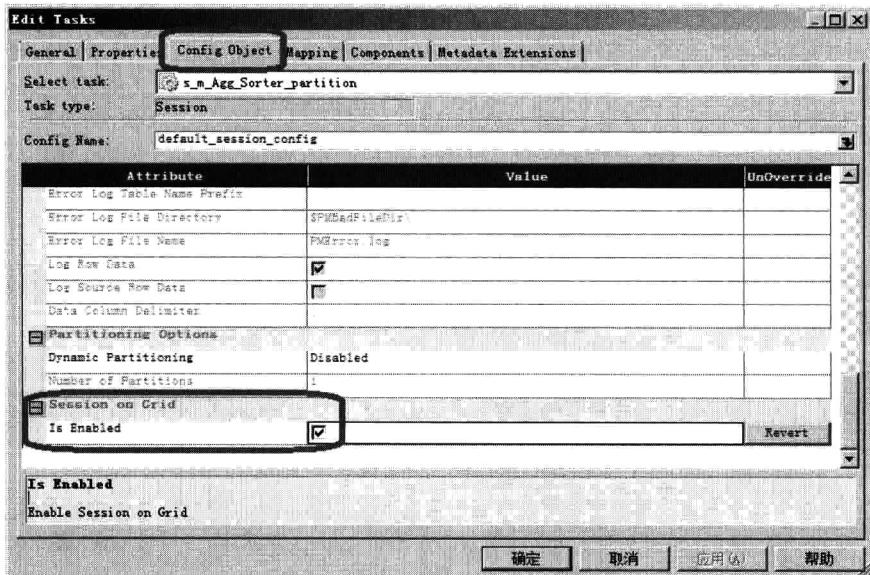


图 5-38

(2) 资源选项：资源选项也是 Session 的属性，在 Session 的 General Tab 中。资源可以是操作系统，也可以是一个节点。资源选项可以指定到某个 Session 或者 Mapping 的组件（Transformation）上，例如，可以指定某个 Session A 或者 Session A 的 Source Qualifier 组件仅仅可以运行在 Node 1 上，如图 5-39 所示。

理解这部分内容，重要的是掌握 Partition Group 的概念。当将 Session 配置为 Session on Grid 后，DTM（Data Transformation Manager）进程将 Session 中的线程进行分组，每个分组将运行在一个节点上。每一个分组就是一个 Partition Group。

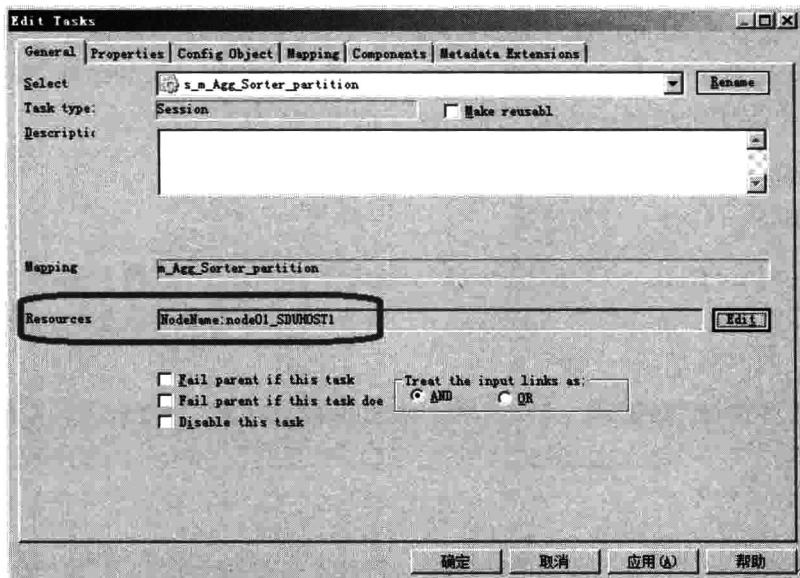


图 5-39

(3) 无资源设置的 Partition Group: 例如, 一个 Mapping 被分为两个 Partition。Partition Group 的情况如图 5-40 所示 (此图来自 PowerCenter 手册)。

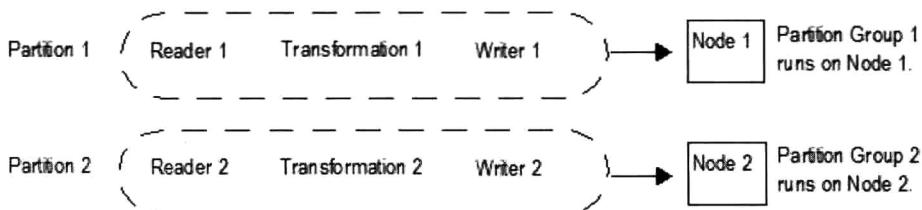


图 5-40

从图 5-40 中可以看到, Partition 1 属于 Partition Group1, 并运行在 Node 1 上。Partition 1 在 Node 1 上作为一个独立的 DTM 进程。这就是说 Session 运行时可能是一个进程, 也可能是多个进程。一个特别的, 但很常见的场景: 如果 Transformation 是 Joiner 组件并使用 n:n 的 Partition。这时候数据将在这两个节点上进行交换, 引起大量的 TCP/IP 流量。这句话的意思是说, 在 Session on Grid 的情况下, 网络性能是影响性能的另一个关键因素。

(4) 有资源设置的 Partition Group: 当为 Mapping 指定资源需求时, DTM 进程根据资源需求划分 Partition Group。例如, 如果源文件只能在某个节点上读取, 这时候就需要在

Source Qualifier 上指定资源需求为该节点。在这种情况下, Partition Group 划分情况如图 5-41 所示。

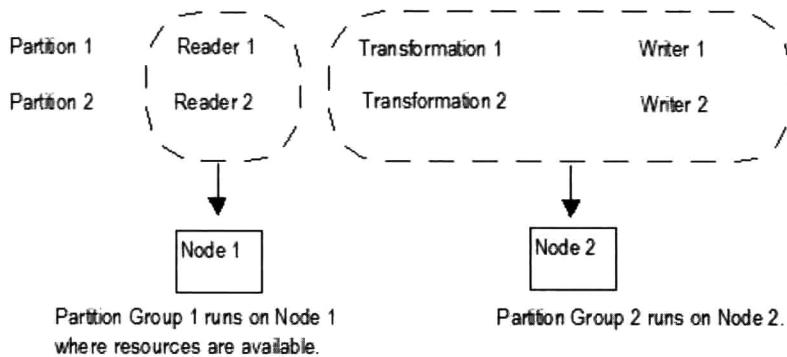


图 5-41

图 5-41 中有两个 Partition Group, 两个 Reader 将运行在 Node 1 上, 这样就保证 Reader 不会在多个节点上浮动, 从而影响源文件的读取。Transformation 和 Writer 则属于 Partition Group 2, 运行在 Node 2 上。

从上面的解释可以观察到, 在 PowerCenter 中, 提供并行的方式有两种:

- ◎ 横向分割, 增加数据加工的流水线, 提升 Session 性能。
- ◎ 纵向分割, 提升性能, 保证资源的可用性。

## 5.4 高可用性 ( HA )

HA (High Available) 即高可用性群集, 是保证业务连续性的有效解决方案。一般有两个或两个以上的节点, 分为活动节点及备用节点。和 Grid 不同, 高可用性方案是主、备的模式, 假如有两个节点, 一个节点处于活动状态, 另一个节点处于非活动状态; 如果有多个节点, 也仅支持一个节点处于活动状态, 其他所有的节点都处于备份状态。PowerCenter 的高可用性有两种实现方式:

- ◎ PowerCenter 自带的 HA 方案。
- ◎ 依托如 HP Service Guard 和 IBM HACMP 等提供的第三方 HA 方案。

### 5.4.1 PowerCenter 自带的 HA 方案

如图 5-42 所示是 PowerCenter HA 的典型架构。

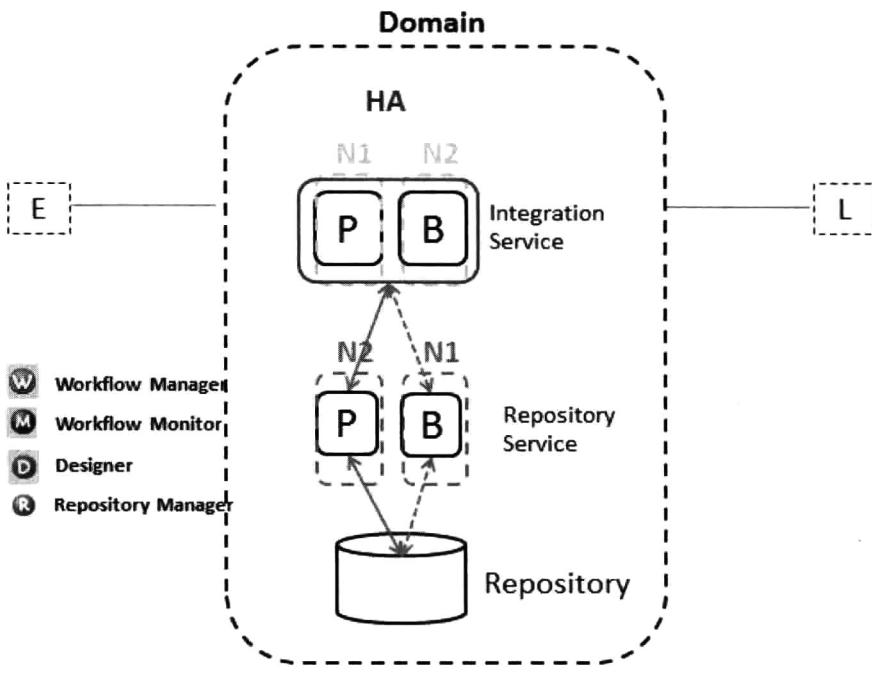


图 5-42

在图 5-42 中有两个典型的 PowerCenter 服务，分别是 Integration Service 和 Repository Service，它们均被配置为 HA 模式。

HA 的安装过程和 Grid 一样，第一个节点安装时选择“Create Domain”，第二个节点安装时选择“Add into existing Domain”，区别在 Admin Console 中的配置过程。在安装结束后，在 Admin Console 中创建一个 Repository Service。创建 Repository 时，需要选择 Primary 进程运行的节点，并选择 Backup 进程运行的节点。主节点只能选择一个，但备份节点可以选择多个，这样就出现一个 Primary 节点、多个 Backup 节点的情形。同样，在 Admin Console 中创建 Integration Service，创建过程和 Repository Service 的创建过程非常相似。

安装完成后，Repository Service 有唯一一个逻辑名称，但是分别在 Primary Node 和 Backup Node 上运行了一个进程。Integration Service 也是如此，虽然在主、备节点上都有运

行的进程，但它们共享唯一的一个逻辑名称。只有 Primary 节点进程对外部的请求进行响应，当 Primary 节点由于某些原因失效后，Backup 将接管原来的 Primary 节点承担的工作，成为新的 Primary 节点。

HA 管理不仅仅是系统的问题，还需要程序的支持。笔者曾经有很长时间的程序员工作经历，那个时间经常会被问道：你写的程序是否能支持 HA 环境，即程序异常退出是否支持从断点进行恢复？这样的程序可以运行在 HA 环境下，也可以运行在非 HA 环境下。PowerCenter 的 Session 具有这样的能力。在 Session 的 Properties Tab 中有一项设置叫作 Recovery Strategy，它有 3 个选项，分别是 Fail task and Continue workflow、Resume from last checkpoint 和 Restart task。这 3 个选项不是仅在 HA 环境中才能使用，它们的含义分别如下。

- ◎ Fail task and Continue workflow：将当前任务标记为失败，但继续运行后续任务。
- ◎ Resume from last checkpoint：从断点进行恢复。这个选项经常被用在 HA 环境中。
- ◎ Restart task：重新启动该任务。对有些重新运行不影响结果的任务，也可在 HA 场景下使用。

#### → 注释

##### PowerCener HA 的应用场景探讨

这里使用探讨，而不是建议，是因为 HA 适用哪种场景本身不是纯粹的技术问题，而是业务需求与技术支持能力的一种平衡。假如把 PowerCenter 支持的任务分为两类。

- ◎ 批量任务：作为批处理任务，一般的特点是数据量大、运行时间长。假如启用 Resume from last checkpoint 选项，PowerCenter 在运行前期会将要写入目标的数据写到本地磁盘上，以保存断点，满足任务失败再次启动的需要。这样的操作势必会损失一定的性能。对批量任务是否采用 HA 模式及 Resume from last checkpoint 选项，需要开发管理人员一起讨论决定。
- ◎ 实时任务：实时任务多数以小批量、多批次、需要事务支持等特点出现。这种场景使用 HA 往往收益比较高、代价比较小。

#### 5.4.2 依托第三方厂商的 HA 方案

依托第三方厂商的 HA 方案一般指将 PowerCenter 与 Service Guard、HACMP 等技术结合，依托这些软件提供的 HA 能力实现 PowerCenter HA。

依托第三方厂商的 HA 安装、配置时与 PowerCenter 自带的 HA 安装有很大的不同。

在实施依托第三方厂商提供的 HA 方案时，只需安装一次 PowerCenter 即可，与单机安装相同。

在安装配置过程中，有两个重点：

- ◎ PowerCenter 应安装在共享的存储上，即随着操作系统 HA 切换自动切换的存储上。
- ◎ 将 PowerCenter 相关的服务绑定在浮动 IP 或者浮动 IP 对应的主机名上，并将 PowerCenter 服务整合到第三方 HA 软件的 Package 中，保证当主机节点发生切换时，PowerCenter 可以在另外的节点上自动启动。

依托第三方 HA 方案实现的 PowerCenter HA 同样需要 Session 级别的 Recovery Strategy 的支持，需要与 Recovery Strategy 配合使用。

### 5.4.3 两种 HA 方案对比

表格是进行事物比较的最佳方式之一，这里使用表格形式对两种 HA 实现方式做一个比较，如表 5-4 所示。

表 5-4

HA 方案 比较项目	PowerCenter 自带的 HA	依托第三方的 HA
切换时间	秒级别。由于 Backup 服务已经启动，仅仅是处于 Backup 状态，所以切换时间非常快	分钟级别。需要 unmount 文件系统，重新 mount 到备份系统，并启动 PowerCenter 服务。这样就需要数分钟，甚至 10 分钟以上
安装配置	分别在主节点、备份节点上安装 PowerCenter，共需要安装两次	仅需在浮动 IP 上安装一次
是否需要共享存储	需要。并且要求两个节点同时可读/写	需要。在某一时刻，某一节点可读/写
是否需要 Recovery Strategy 支持	需要	需要

## 5.5 Web Service 应用

在 PowerCenter 中，曾经遇到不同的客户、开发人员从不同的角度关心 PowerCenter 对

Web Service 支持的问题。虽然这几个方面不一定紧密相关（但也是相互联系的），为了使读者在想到 Web Service 时，能在本书中找到一个集中的地方，因此将它们放到了同一个章节中。

(1) Web Service Hub: PowerCenter 提供的 Web Service 的集中访问点。在 Web Service hub 上，可以看到 PowerCenter 内置的 Web Service 和通过 PowerCenter 开发的新的 Web Service（详细内容请参见下面的小节）。

(2) Web Service 调度/监控接口: PowerCenter 内置的 Web Service 服务，通过这些服务可以查看 PowerCenter 元数据、对 Workflow/Session 等进行调度、监控及与第三方平台进行集成。

(3) Web Service Consumer: 在 PowerCenter 中，Web Service Consumer 有个商业名字叫作 PowerExchange for Webservice。PowerExchange 是 PowerCenter 所有对外接口的名字，比如 PowerExchange for Oracle，即 PowerCenter 访问 Oracle 的接口。所以 PowerExchange for Web Service 就是 PowerCenter 访问 Web Service 的接口的名字。

(4) Web Service Provider: Provide 的功能是将 PowerCenter 中的一个逻辑，如 Transformation、Maplet、Mapping 等发布成 Web Service。即用 PowerCenter 对外提供 Web Service 服务。

### 5.5.1 Web Service Hub

Web Service Hub 是 Domain 管理的一个应用服务，作为外部客户端访问 Web Service 的网关。Web Service Hub 处理来自客户端访问 PowerCenter 功能的 SOAP 请求，如外部客户端访问 Integration Service 和 Repository Service。

当 Web Service Hub 创建后，可以通过 Admin Console 进行访问，如图 5-43 所示在 Admin Console 中创建了一个名为 WSH\_SVC 的服务。

在图 5-43 中可以看到，可以通过 `http://<hostname>:7333/wsh` 访问 Web Services Hub。打开如上 URL，可以看到在 Web Service 中管理了两类 Web Services，如图 5-44 所示。

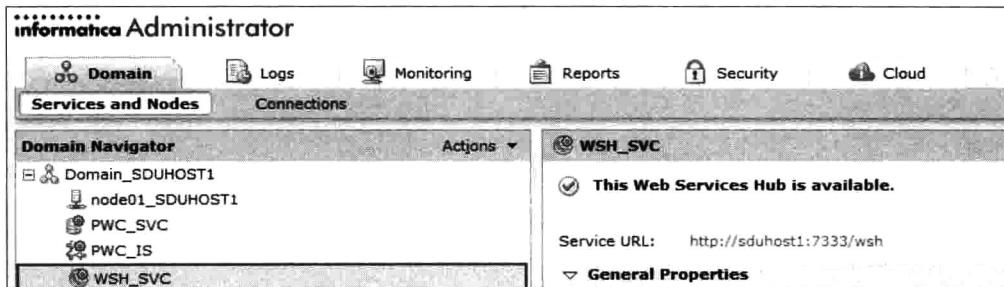


图 5-43

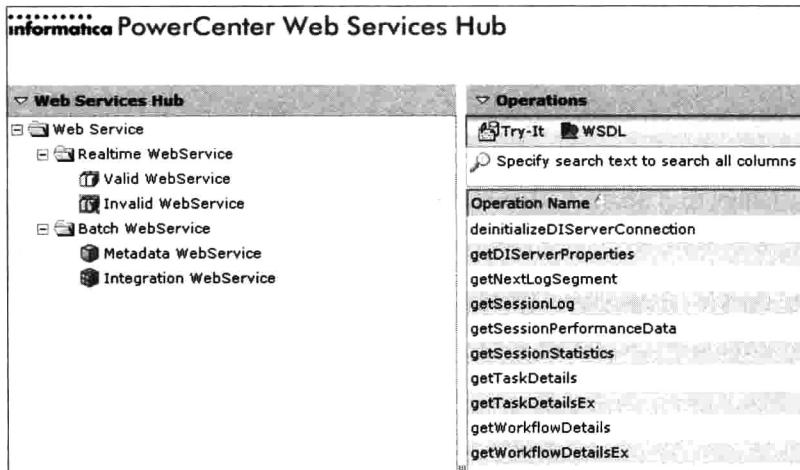


图 5-44

(1) Realtime WebService: 将 PowerCenter 中的 Workflow 发布为 Web Service 服务。响应外部的 SOAP 请求，达到数据输入、输出和提供特定计算逻辑的要求（与 6.5.4 节对应）。

(2) Batch WebService: 主要用于运行、监控 Session 和 Workflow，还包括一些访问 Repository Metadata 的 Web Service API（与 6.5.2 节对应）。

## 5.5.2 Web Service 调度/监控接口

批量 Web Service 提供了大量的调度、监控 PowerCenter 的 API，可以用于第三方工具的二次开发，也可以用于项目中调度、管理的需要。例如，已经有第三方厂商提供的调度工具就是基于这些接口的，最有名的是 BMC 公司的调度工具 Control-M；也有的项目中将

这些接口封装为给第三方接口调用的自助服务。面向 Data Integration 的 Web Service Operation 包括如下几类。

◎ 连接 Repository Service 接口:

```
Login  
Logout
```

◎ 连接 Integration Service 并获得其相关信息接口:

```
pingDIServer  
getDIServerProperties  
initializeDIServerConnection (deprecated)  
deinitializeDIServerConnection (deprecated)
```

◎ 调度、启动 Workflows 接口:

```
startWorkflow  
startWorkflowEx  
stopWorkflow  
scheduleWorkflow  
startWorkflowFromTask  
unscheduleWorkflow  
waitTillWorkflowComplete
```

◎ 启动、停止 Tasks 接口:

```
recoverWorkflow  
resumeWorkflow (deprecated)  
startTask  
stopTask  
waitTillTaskComplete
```

◎ 获得 Session 的监控、统计接口:

```
getNextLogSegment (deprecated)  
getSessionLog  
getSessionPerformanceData  
getSessionStatistics  
getTaskDetails  
getTaskDetailsEx
```

```
getWorkflowDetails  
getWorkflowDetailsEx  
getWorkflowLog  
monitorDI Server  
startSessionLogFetch (deprecated)  
startWorkflowLogFetch (deprecated)
```

关于 Data Integration web services operation 的定义在 di.wsdl 中，它位于安装目录的..\\services\\WebServiceHub\\wsh 目录下。

同时在 Web Service Hub 中提供了一个基于 Web 的 Web Service 测试界面。PowerCenter 用户可以通过这个界面测试 PowerCenter 提供的 Batch Web Service，同时也可以测试开发人员自己发布的 Web Service。为了节省篇幅，不再提供截图，在 Web Service Hub 中寻找 Try-It 按钮，输入相关的参数就可以测试关心的 Web Service。

### 5.5.3 Web Service Provider

Web Service Provider 是将 PowerCenter 的相关对象发布为 Web Service 的过程。这个功能方便了 EDW 项目开发人员分享数据的过程，可以通过 PowerCenter 非常简单地分享 EDW 产生的数据标准或者分析的有效结果。PowerCenter Web Service Provider 支持高效的开发过程，满足业界标准的开发方法，是释放 EDW 价值的有效途径。

还是通过一个例子解释如何通过 PowerCenter 发布一个 Web Service。假如，EDW 项目组发布了一个地址清洗的功能和一个算数加法服务。这两个算法自身复杂性差别很大，但将其发布为一个 Web Service 服务的过程在 PowerCenter 中是非常相似和简单的。因此这个例子选择一个简单算法“ $x+100$ ”，演示通过 PowerCenter 发布 Web Service 服务的过程。

在 PowerCenter 中可以将 Transformation、Maplet 或者 Mapping 发布为 Web Service。这个例子演示如何将一个 Reusable Transformation 发布为 Web Service。

第一步：创建一个 Reusable Transformation 完成“ $x+100$ ”的逻辑。

在 PowerCenter Designer 中，选择 Transformation Developer，将一个 Expression Transformation 拖入 Transformation Developer 开发空间。

创建变量 IN\_VAR 为输入，OUT\_VAR 为输出，数据类型均为 Integer，并将 OUT\_VAR 的表达式设为“IN\_VAR+100”，将 Expression Transformation 命名为 exp\_WS\_Add100，如

图 5-45 所示。这个计算逻辑比较简单，仅仅几个简单步骤，已经完成了“ $x+100$ ”的业务逻辑。



图 5-45

### 第二步：创建一个支持 Web Service 的 Mapping。

仍然在 PowerCenter Designer 中选择菜单 Mapping→Create Web Service Mapping 命令，下面有 3 个选项：Import from WSDL、Use Source/Target Definitions 和 Use Transformation/Mapplet Definitions。从这些选项可以看到，PowerCenter 可以将如上的 3 种对象发布成 Web Service。

在这个例子中要使用第一步创建的 Reusable Transformation 来创建 Web Service，因此选择 Use Transformation/Mapplet Definitions 命令，这时弹出如图 5-46 所示的对话框。

选择刚刚创建的 Reusable Transformation exp\_WS\_Add100，这时在对话框的下半部分会看到刚刚创建的两个变量 IN\_VAR 和 OUT\_VAR，单击 OK 按钮即可。这时 PowerCenter 自动创建了一个名为 m\_exp\_WS\_Add100 的 Mapping，如图 5-47 所示。



图 5-46

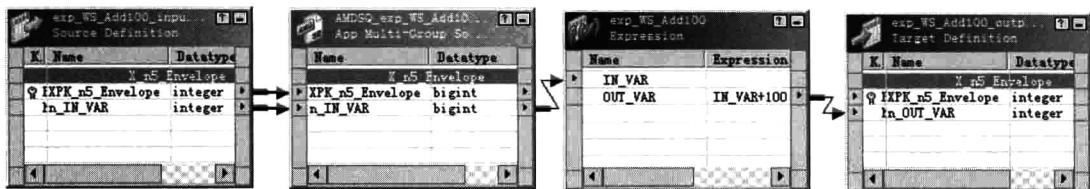


图 5-47

在这个 Mapping 中，自动引用了刚刚创建的 Expression Transformation，并产生了一个奇怪的源和目标。这个源和目标是对 Web Service 的一个封装，在这里完全可以不必关注，全部交给 PowerCenter 来处理。

**第三步：创建 Workflow，并将其发布为一个 Web Service。**

在 Workflow Manager 中，为刚刚 PowerCenter 自动创建的 Mapping m\_exp\_WS\_Add100 创建一个 Workflow，如图 5-48 所示。依次选择 Workflows→Wizard→Next→Next 即可完成。



图 5-48

这个 Workflow 和其他的 Workflow 到目前为止没有任何区别。假如 Mapping 中使用了数据库对象，需要为其指定数据库连接；如果有文本，也需要为其指定目录和文件名。

这时的重点是选择菜单 Workflows→Edit 命令，进入 Workflow 的编辑界面，在 Web Services 选项后，勾选 Enabled 复选框，如图 5-49 所示。

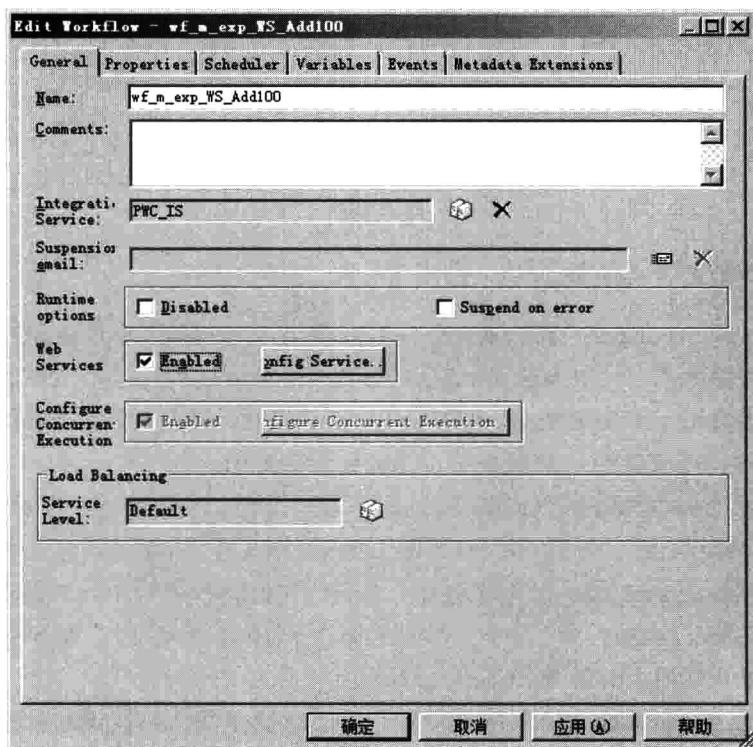


图 5-49

单击 Config Service 按钮，进入如图 5-50 所示的编辑界面。设置 Web Service Name，它的名字可以是任意希望的字符串。其他的选项设置如图 5-50 所示。

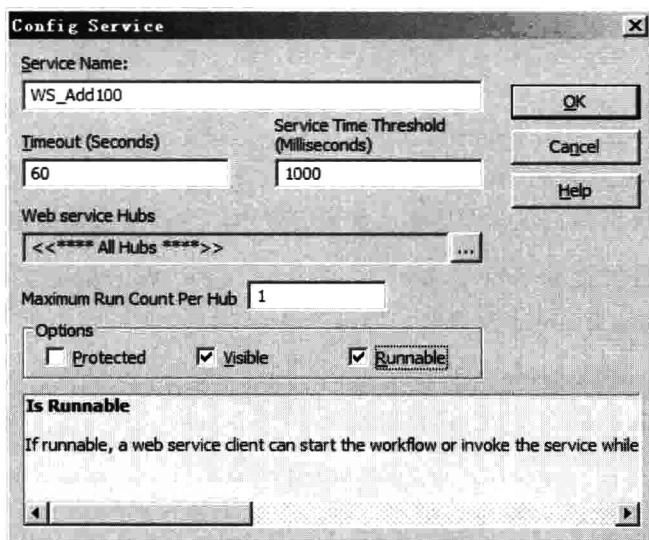


图 5-50

几个关键参数的含义如下。

- **Timeout (Seconds):** Web Service Hub 获得 Web Service Request 和产生响应时间的最大忍耐时间，如果 Web Service Hub 在 Timeout 之前无法产生相应信息，它将向客户端发送一个失败信息。当 Web Service 逻辑比较复杂、响应时间比较长，或者当 Hub 比较繁忙时，需要谨慎考虑如何设置此值。
- **Service Time Threshold (Milliseconds):** 以一个例子来说明这个参数。假如 Services Time Threshold 设为 1000，当 Web Service Hub 在 1000 毫秒内无法响应处理请求时，Web Service Hub 就会启动另一个实例来处理后续的 SOAP 请求。
- **Web service Hubs:** 可以指定一个运行此 Workflow 的特定的 Web Service Hub，也可以默认使用所有的 Web Service Hub。
- **Maximum Run Count Per Hub:** Web Service Hub 可以启动的最大的 Web Services 实例数。
- **Protected:** 启动或者运行 Web Service 要求提供授权信息，即输入用户名、密码。只有拥有运行 Workflow 权限的用户才可以请求 Web Service。

- ◎ **Visible:** 决定 Web Service 在 Web Service Hub 上是否可见。当选择可见后，Web Service 就会被发布到 Web Service Hub 上，开发人员可以在 Web Service Hub 上查看、测试 Web Service。
- ◎ **Runnable:** 允许 Web Service 请求启动 Web Service Workflow。如果 Runnable 是 Enabled，Web Service 可以启动 Web Service Workflow，也可以在 Workflow 处于运行状态时发送 Web Service 请求。当 Runnable 是 Disable 时，Web Service 客户端将无法启动 Web Service Workflow，仅当 Workflow 处于运行状态时，才可以响应外部的 Web Service 请求，这时只能通过 UI 或者 infacmd 命令启动 Workflow。

保存如上的配置后，如果登录 Web Service Hub（默认是 <http://<hostname>:7333/wsh/>），就可以看到刚刚发布的 Web Service，并可以对其进行测试、调用。

选择 Realtime WebService→Valid WebService，一般情况下，就可到已经发布的 Web Service 及其相关联的 Workflow，如图 5-51 所示。但有些情况下 Web Service 可能是无效的，这时就需要选择 Invalid WebService 查看无效的 Web Service。

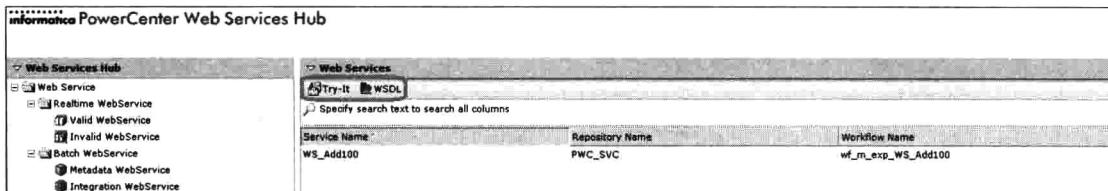


图 5-51

图 5-51 标记了两个常用按钮，一个是 Try-It，另一个是 WSDL。Try-It 是 PowerCenter 提供的一个 Web Service 测试工具，通过这个工具可以帮助开发人员测试 Web Service 逻辑。WSDL 提供了调用 Web Service 的 WSDL 定义和调用此 Web Service 的 URL。

如果在 Workflow 配置中勾选了 Runnable 复选框，当单击 Try-It 按钮后，输入 IN\_VAR 参数，单击 Send 按钮，就会看到 Workflow 运行的结果，如图 5-52 所示。

这时，在 PowerCenter Workflow Monitor 中会看到一个被启动的 Workflow 处于 Running 状态，正是这个 Workflow 在响应 Try-It 工具发出的处理请求。

在 Workflow 定义中，还有几个参数需要关注，它们位于 Session 属性中。双击 Session→选择 Mapping Tab→选择 Sources 下的对象，可以看到图 5-53。

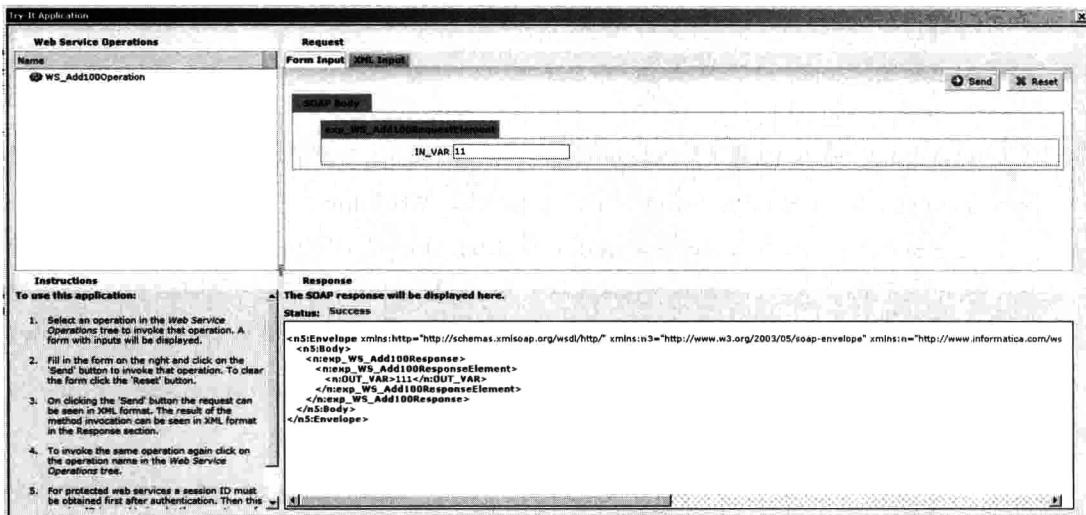


图 5-52

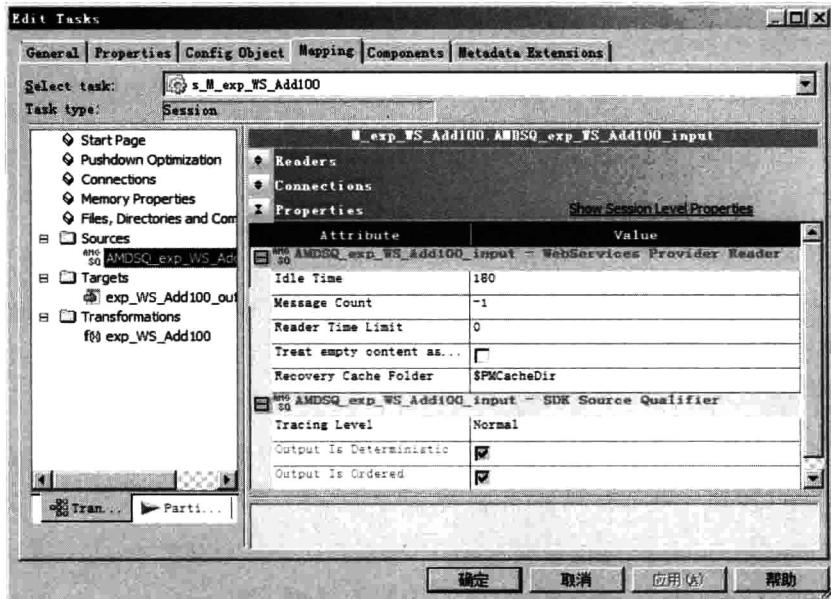


图 5-53

Web Service Workflow 运行实例是否停止，由下面的 3 个参数及系统资源可用性所决定。

- ① **Idle Time:** 当一个运行的 Web Service Workflow 实例在持续的一段时间 (Idle Time 指定) 内没有获得客户端请求时, 这个 Web Service Workflow 实例将停止。例如, 指定 Idle Time 为 180, 默认单位为秒, 如果此实例在 180 秒内没有获得新的客户端请求, 此实例将自动停止。如果 Idle Time 设置为 -1, 此 Workflow 将一直处于 Running 状态。
- ② **Message Count:** Web Service Workflow 实例获得的 Message 数量达到 Message Count 的设置时, 此 Workflow 实例将停止。默认值为 -1, 含义是无限制。
- ③ **Reader Time Limit:** Web Service Workflow 实例启动后的最大时间, 如果达到最大时间后, Web Service Workflow 实例将停止。默认值为 0, 含义是无限制。

### 1. 将一张表发布成 Web Service

将一张数据库表发布成 Web Service, 提供一个针对数据库表的查询接口, 是一种比较常见的业务需求。例如, 对于部门表 DEPT, 希望完成一个输入 DEPTNO, 返回部门相关信息的 Web Service 服务。实现这个需求, 请参照如下的演示过程。

第一步: 创建支持 Web Service 的 Mapping 模板。

选择菜单 Mappings→Create Web Service Mapping→Use Source Target Definitions 命令, 弹出如图 5-54 所示的对话框。

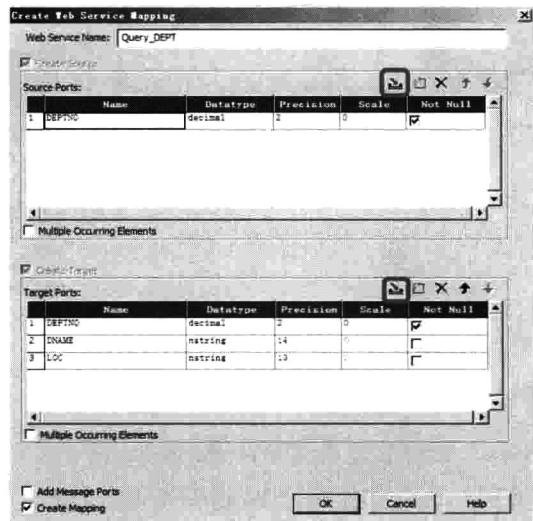


图 5-54

分别使用方框标记的按钮“Import from Source/Target”，定义输入/输出。如图 5-54 所示，Source Ports 为 DEPTNO，Target Ports 为 DEPTNO、DNAME 和 LOC。单击 OK 按钮后，生成一个 Mapping 模板，如图 5-55 所示。

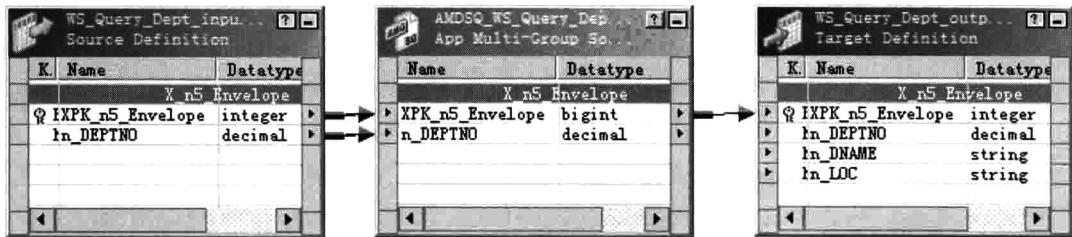


图 5-55

第二步：增加 Lookup 组件，保存 Mapping。

在 Mapping 模板中增加一个 Lookup 组件，将 n\_DEPTNO 拖入 Lookup 组件，并创建 Condition "n\_DEPTNO=DEPTNO"，将 Lookup 组件的输出端口连入 Target，如图 5-56 所示。

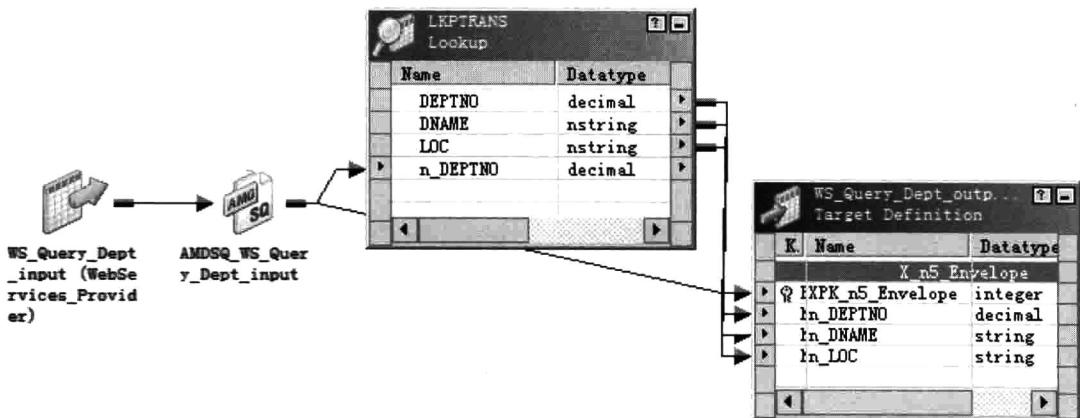


图 5-56

现在已经完成了整个 Mapping 的设计。

第三步：创建 Workflow，并将其发布为 Web Service。这里需要关注的一点就是需要为 Lookup 组件指定数据库连接，这样才能保证 Workflow 是有效的，并可以正常运行。

如果这个查询还包括复杂的计算，如对输入的校验、对输出结果的重构等，就需要将

这个逻辑包装成 Mapplet，再将 Mapplet 发布成 Web Service。将 Mapplet 发布为 Web Service 的过程和将 Reusable Transformation 发布成 Web Service 的过程一样，因此不再进行详细叙述。

## 2. 已有 WSDL，自定义一个 Web Service

在设计 Web Service 时，最常见的事情是：已经与其他小组或者项目完成了需求调研，约定了输入和输出。这就意味着双方已经有了 WSDL（Web Service Definition Language）的约定，这时候需要完成一个 Web Service 的设计。在 PowerCenter 中的设计过程如下。

首先查看已经约定好的 WSDL，如下：

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='phonebook'
  targetNamespace='http://www.mysoapservice.cn/'
  xmlns: tns='http://www.mysoapservice.cn/'
  xmlns: soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns: xsd='http://www.w3.org/2001/XMLSchema'
  xmlns: soapenc='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns: wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
<types>
  <xsd: schema xmlns: xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.mysoapservice.cn/">
    </xsd: schema>
  </types>
<message name='GetPhoneBookRequest'>
  <part name="name" type="xsd: string"/>
</message>
<message name='GetPhoneBookResponse'>
  <part name="phonebookInfo" type="xsd: string"/>
</message>
<portType name='PhoneBookToEveryoneProt'>
  <operation name='GetPhoneBook'>
    <input message='tns: GetPhoneBookRequest' />
    <output message='tns: GetPhoneBookResponse' />
  </operation>
```

```
</portType>
<binding name='PhoneBookSOAP' type='tns: PhoneBookToEveryOneProt'>
    <soap: binding style='rpc'
        transport='http: //schemas.xmlsoap.org/soap/http' />
    <operation name='GetPhoneBook'>
        <soap: operation soapAction='http: //www.cwtservice.cn/newOperation/' />
        <input>
            <soap: body use='encoded' namespace='urn: xmethos-delayed-quotes'
                encodingStyle='http: //schemas.xmlsoap.org/soap/encoding/' />
        </input>
        <output>
            <soap: body use='encoded' namespace='urn: xmethos-delayed-quotes'
                encodingStyle='http: //schemas.xmlsoap.org/soap/encoding/' />
        </output>
    </operation>
</binding>
<service name='PhoneBookService'>
    <port name='PhoneBookSOAP' binding='tns: PhoneBookSOAP'>
        <soap: address location='http: //www.mysoapservice.cn/service.php' />
    </port>
</service>
</definitions>
```

这个 WSDL 描述的是输入 name、返回 phonebookInfo 的一个 Web Service。完成这样的 Mapping 与完成 Hello World Mapping 非常相似。

第一步：导入源（输入）和目标（输出）。

导入源定义：选择菜单 Sources→Web Service Provider→Import from WSDL 命令，弹出如图 5-57 所示的对话框。

这时有两个选项：URL 和 Local File。因为已经准备好了本地 WSDL 文件，因此选择 Local File，找到以.WSDL 为扩展名的文件，选择打开，就可以看到完整的 WSDL 的定义，如图 5-58 所示。



图 5-57

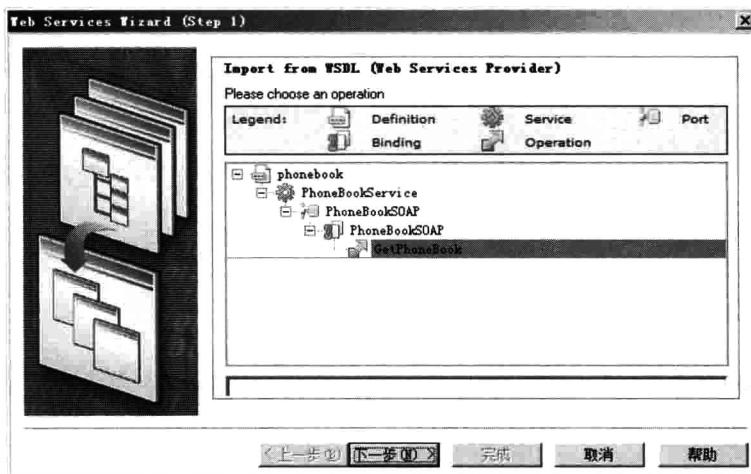


图 5-58

这时，可以看到 Service 名字是 PhoneBookService，Operation 是 GetPhoneBook，只要继续单击“下一步”按钮就可以完成 Web Service Source 的定义。

导入目标的过程和导入源的过程基本一致，不再进行重复叙述。

#### → 注释

因为 WSDL 是 Web Service 输入、输出和 Operation 的说明文件，因此 PowerCenter 支持在导入源的同时导入目标定义。

第二步：完成源和目标的映射，如图 5-59 所示。

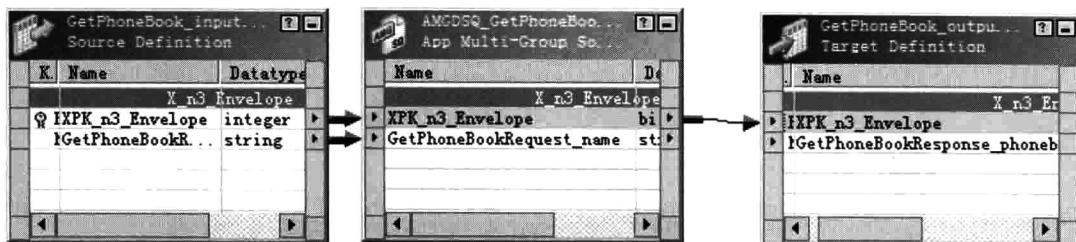


图 5-59

完成源和目标的映射主要是完成业务逻辑的定义，即在输入 Name 和输出 PhoneBookInfo 之间建立真实的业务逻辑。这个过程和其他 Mapping 非常相似，可以很简单，也可能非常复杂。

在此有一个特别关注点，就是需要将 Source 的 Envelope 和 Target 的 Envelope 关联到一起。

通过上面的演示，相信读者已经可以建立属于自己的简单 Web Service，为其他用户提供数据服务了。

#### 5.5.4 Web Service Consumer

PowerCenter 提供了一个 Web Service Consumer Transformation 组件，在 PowerCenter 中使用的图标是 。

在前面提到过，Web Service Consumer 的商品名叫作 PowerExchange for Web Service，它的功能就是访问外部提供的 Web Service。例如，气象局提供了一个基于 SOAP 的查询天气的 Web Service 接口，而公司的数据仓库项目需要天气信息作为数据分析的一个纬度，这时就可以利用 Web Service Consumer 或者称为 PowerExchange for Web Service 的功能。

这个功能同样通过一个简单的例子进行演示。这个例子是通过一个 Mapping 调用在 5.5.3 节已经创建的 Web Service——WS\_Add100。仍然采用员工涨工资的案例，数据源是员工表 EMP，目标表是和员工表一样的一张表。

第一步：开发 Mapping，调用 Web Service——WS\_Add100。

打开 Web Service Hub，通过 `http://<hostname>:7333/wsh/` 获取 WSDL 调用的 URL，例如 `http://<hostname>:7333/wsh/services/RealTime/WS_Add100?WSDL`。如果是在实际的项目中，就需要向 Web Service 服务提供方索要相关的信息。

此处建立 Mapping 的过程和一般的 Mapping 是一样的，将数据源和目标表拖入到 Mapping 开发区。核心点是从图标栏选取 Web Service Consumer Transformation 加入到 Mapping 中，如图 5-60 所示，在 Address 栏输入 WSDL 的 URL。

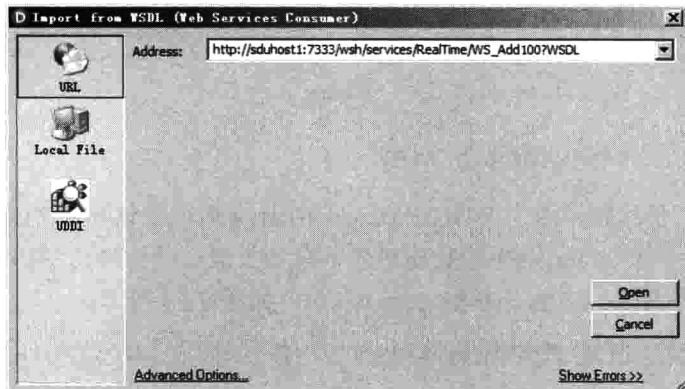


图 5-60

单击 Open 按钮，这时就可以看到在该 Web Service 中已经定义的 Operations，如图 5-61 所示。

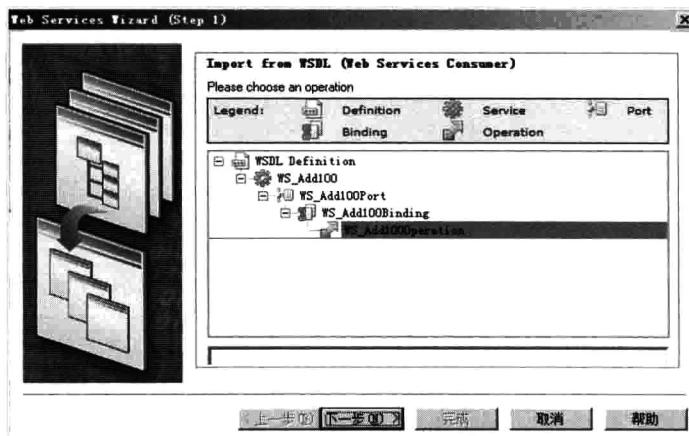


图 5-61

单击“下一步”按钮直到完成。

这时将 Source Qualifier 的 SAL 与 Web Service Consumer 的输入端口 n\_IN\_VAR 连接，将 Web Service Consumer 的输出端口 n\_OUT\_VAR 与 Target 的 SAL 连接，如图 5-62 所示。

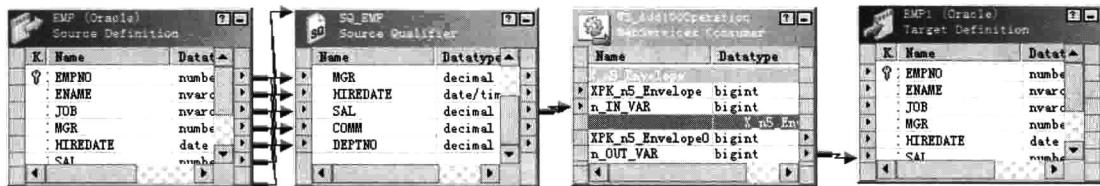


图 5-62

### 第二步：如何应对 Pass Through 字段？

这时会遇到一个问题，因为 Web Service Consumer 组件是 Active 组件，无法将 Source Qualifier 中的其他字段与 Target 中的其他对应字段相关联。这时，双击 Web Service Consumer 组件，选择 Web Services Consumer Properties Tab，如图 5-63 所示。

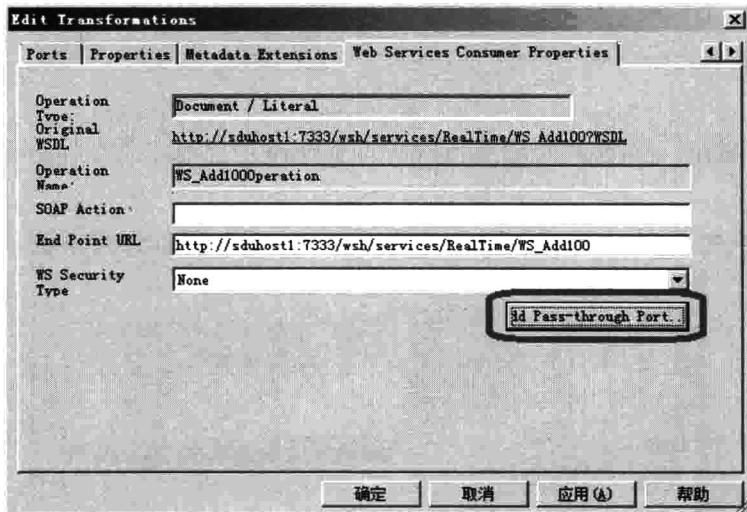


图 5-63

在 Properties Tab 中可以看到非常多的 Web Service 属性。为了解决 Active 组件引起的无法关联字段的问题，单击 Add Pass-through Port 按钮，将目标表的其他字段添加进来。为了达到演示的目的，仅增加两个字段 EMPNO 和 ENAME，如图 5-64 所示。

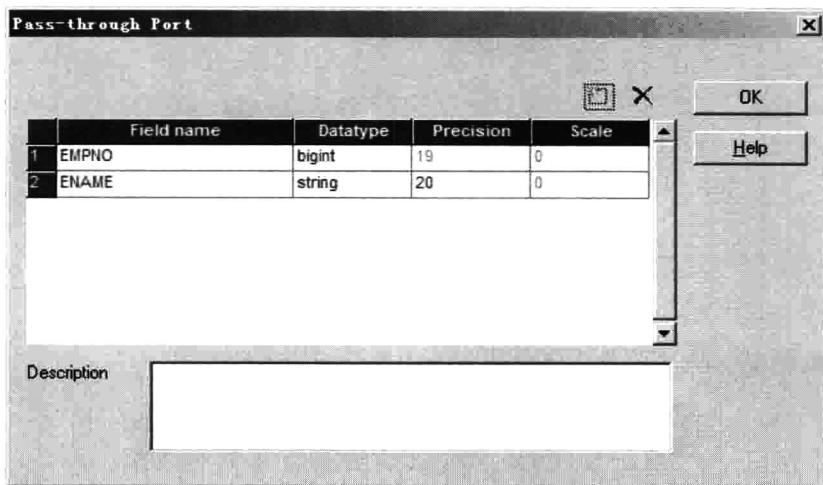


图 5-64

这时的 Web Service Consumer Transformation 如图 5-65 所示。

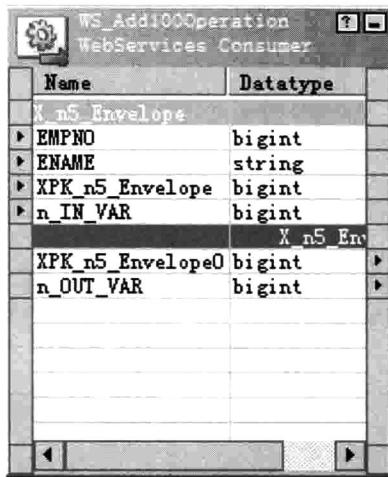


图 5-65

仍然没有看到新增的输出字段，还需要额外的几步操作。选中 Web Service Consumer 组件的标题部分，一定要是标题部分，单击鼠标右键，在弹出的快捷菜单中选择 WSDL WorkSpace→Output Mode 命令，进入 XML Editor。这时同样选择标题栏 X\_n5\_Envelope0，单击鼠标右键，在弹出的快捷菜单中选择 Add a Reference Port 命令，如图 5-66 所示。

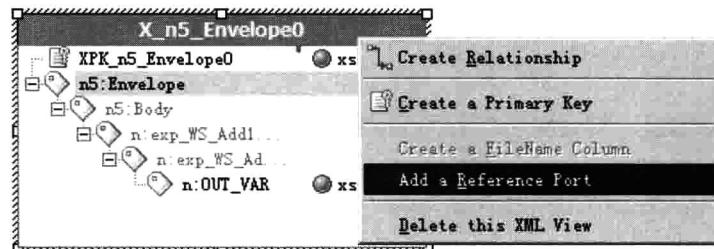


图 5-66

在弹出的如图 5-67 所示的对话框中，选中两个新增加的端口。

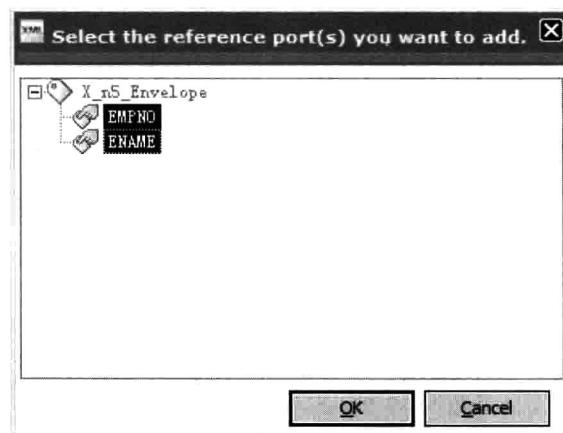


图 5-67

保存、退出。进入 Mapping 编辑界面，按照表 5-5 所示关联相关字段。

表 5-5

Source Qualifier	Web Service Consumer	Target
EMPNO	EMPNO	
ENAME	ENAME	
EMPNO	XPK_n5_Envelope	
	REF_EMPNO	EMPNO
	REF_ENAME	ENAME

完成的 Mapping 如图 5-68 所示。

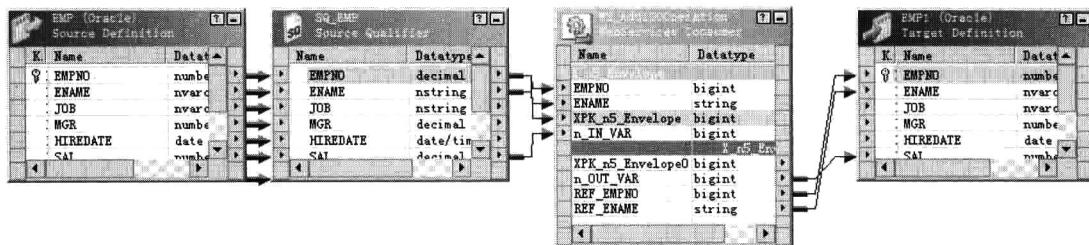


图 5-68

这样就完成了一个完整的调用 Web Service 的过程，并且演示了如何在 Web Service Consumer 中增加 Pass-through 端口。

## 5.6 Pushdown Optimization

**案例：**ABC 公司使用 ETL 工具已经有数年的时间，近期随着公司业务快速发展，数据仓库数据量急剧增加，已经很难及时地为每天早上 8 点的管理层会议提供必要的支撑数据。项目组在讨论系统扩容的问题，系统管理员对 EDW 系统的现状进行了分析，总结如下。

**硬件环境：**硬件 ETL 服务器 1 台、数据库服务器 8 台、报表服务器 1 台。

白天 ETL 服务器空闲，数据库服务器和报表服务器非常繁忙；晚上 11 点到第二天早晨 6 点 ETL 服务器非常繁忙，数据库服务器利用率低于 40%，报表服务器几乎全部空闲。

总的来看，系统服务器的资源利用率并不高，这种情况下申请增加新的服务器很难被批准。总体上看 ETL 工作占系统资源和时间的 70%，如何充分利用现有的硬件资源、加快数据处理的效率是解决问题的关键。因此项目组找来了 Informatica 工程师讨论 ETL 的问题，通过几天的讨论，Informatica 工程师给出了两个解决方案。

- ◎ 考虑 ETL 服务器和报表服务器繁忙时间完全不重叠，使用 Informatica Grid 功能，将两台服务器构建为一个集群。
- ◎ 引入 PowerCenter Pushdown 能力，将部分 ETL 负载转移到数据仓库服务器上。

ABC 公司的开发人员对 Informatica 集群已经有所了解，希望看一下 Pushdown 的详细介绍。

### 5.6.1 Pushdown 优化是什么

了解什么是 Pushdown，还是要介绍 PowerCenter 的执行原理。日常使用的 PowerCenter 的执行过程是：第一步，PowerCenter 从 Repository 中读取元数据；第二步，PowerCenter 的 DTM 引擎解析读取的元数据，DTM 引擎在安装 PowerCenter 的服务器上执行。Pushdown 与传统的 PowerCenter 执行过程不同的是在第二步，在这一步，PowerCenter 试图将 Mapping 转换为一组 SQL 语句，并将生成的 SQL 语句推送到数据库服务器上执行。

理论上，所有的 Mapping 均可以完整地转换为 SQL 语句，但是实际上 PowerCenter 并非能把所有的 Mapping 或者 Mapping 中所有的组件都转换为 SQL 语句，因此就衍生出多种 Pushdown 的类型，以最大化系统的性能，充分利用所有的硬件资源。

### 5.6.2 Pushdown 优化类型

如图 5-69 所示是 ABC 公司正在使用的一个 Mapping，ABC 公司的开发人员希望不修改 Mapping 就能支持前面提到的 Pushdown 功能。这个 Mapping 功能非常简单，从 EMP 表读取数据，在 Expression 中对工资加 100 (SAL+100)，并将数据写入到 EMPX，并且表 EMP 和 EMPX 在同一个数据库中。



图 5-69

已经为该 Mapping 创建了对应的 Session，双击已经创建的 Session，选择 Mapping Tab，在对话框的左上方有一个 Pushdown Optimization 选项，如图 5-70 所示。

单击此选项，PowerCenter 会自动打开 Pushdown Optimization Viewer 窗口，如图 5-71 所示。

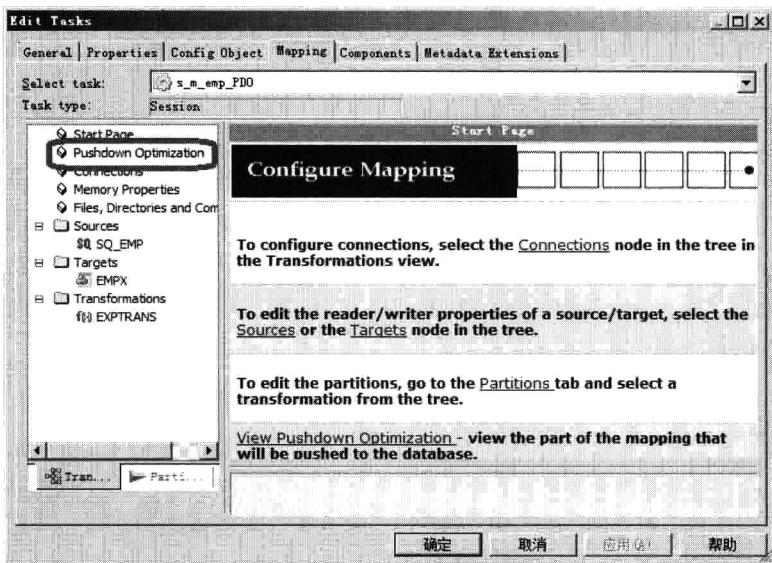


图 5-70

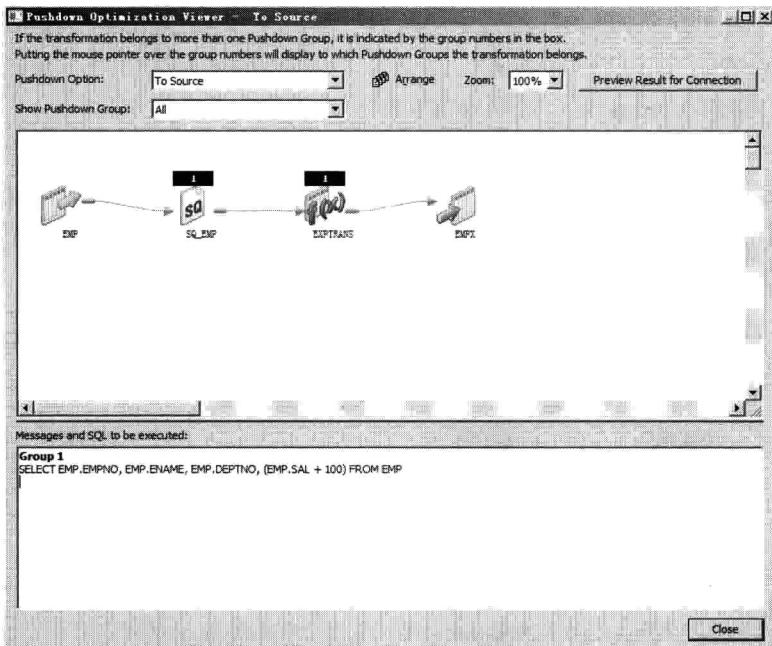


图 5-71

这个窗口是 Pushdown 预览窗口。Pushdown Option 有 4 个选项：To Source、To Target、Full 和 \$\$PushdownConfig。

- ◎ To Source：当选择 To Source 时，对如上的 Mapping，PowerCenter 生成的 SQL 语句为：SELECT EMP.EMPNO,EMP.ENAME,EMP.DEPTNO,(EMP.SAL + 100) FROM EMP。当选择 Source-Side Pushdown 优化时，Integration Service 从源向目标分析 Mapping，直到遇到一个下游的组件无法下推到源时，它会将前面的所有组件生成的 SQL 语句下推到源系统。对照前面的 SQL 和 Mapping，相信大家很容易理解 To Source 的含义。
- ◎ To Target：当选择 To Target 时，PowerCenter 生成的 SQL 语句为：INSERT INTO EMPX (EMPNO,ENAME,DEPTNO,SAL) VALUES (?,?,?,(? + 100))。当选择 Target-Side 的 Pushdown 优化时，Integration Service 从目标向源分析 Mapping，直到遇到一个上游的组件无法下推到目标时，它会将生成的 SQL 语句下推到目标系统。根据转换逻辑的不同，生成的 SQL 语句可能是 Insert、Delete 或者 Update。
- ◎ Full：当选择 Full 时，PowerCenter 生成的 SQL 语句为：INSERT INTO EMPX (EMPNO,ENAME,DEPTNO,SAL) SELECT EMP.EMPNO,EMP.ENAME,EMP.DEPTNO,(EMP.SAL + 100) FROM EMP。当选择 Full Pushdown 优化时，Intergration Service 会尽力将 Mapping 完全下推到数据库中。
- ◎ \$\$PushdownConfig：这是 PowerCenter 提供的一个 Pushdown 参数。有时 PowerCenter 管理员希望 Pushdown 可以根据数据库的负载动态地设置 Source-Side、Target-Side 或者 Full Pushdown Optimization，从而最优化 ETL 的性能。这时就可以使用这个参数根据不同的条件启动不同的 Pushdown 类型。

为了说明 Full Pushdown 功能，使用一个稍复杂的 Mapping，当选择 Full Pushdown 时，在 Pushdown Optimization Viewer 中看到的视图如图 5-72 所示。



图 5-72

从图 5-72 中可以看到，这个 Mapping 中一部分组件头顶 1 的图标，一部分组件头顶 2 的图标，这是 PowerCenter 在使用 Pushdown 功能时进行管理的一个概念，叫作 Pushdown Group。

Group 1 的 SQL 语句如下：

```
SELECT SUM(EMP.SAL),EMP.DEPTNO FROM EMP GROUP BY EMP.DEPTNO
```

Group 2 的 SQL 语句如下：

```
INSERT INTO EMPX(EMPNO,DEPTNO,SAL) VALUES (CAST(?) AS NUMBER (10,0)),?,(?) + 10)
```

从上面的例子可以看出，当选择 Full Pushdown 时，Integration Service 尽力从源和目标两侧分析 Mapping，尽力将 Mapping 逻辑下推到数据库中。但在图 5-72 中的 Rank 组件仍然运行在 PowerCenter 服务器上。

“Pushdown Optimization Viewer”是 Pushdown 的预览窗口。如果要配置、允许 Mapping 进行 Pushdown 的话，双击 Session，选择 Properties Tab，在 Pushdown Optimization 中选择需要 Pushdown 的模式，只有这里的选择才是在执行中生效的，如图 5-73 所示。

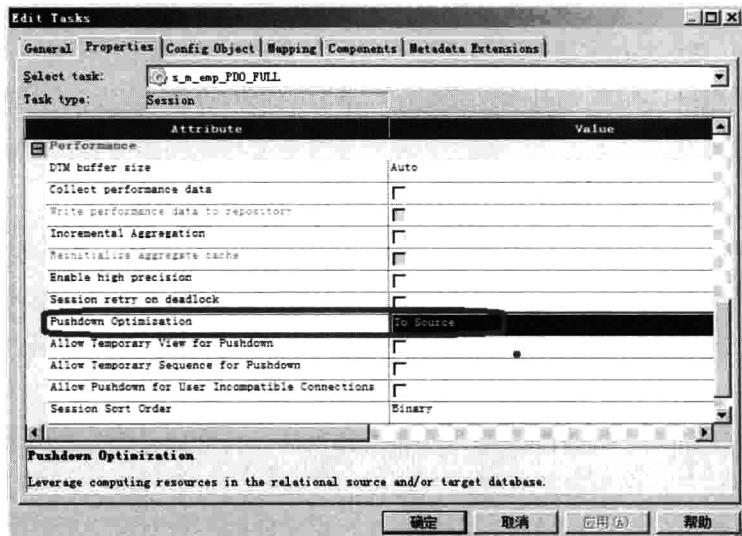


图 5-73

在 Pushdown 设置时，还有几个选项需要关注。

- Allow Temporary View for Pushdown：允许 PowerCenter 在 Pushdown 过程中在数据库中创建临时的视图。在 PowerCenter 中，如果在 Source Qualifier 或者 Lookup 中使用了 SQL Override、Filtered Lookup 或者 Unconnected Lookup，都需要启动这个选项。

- ◎ Allow Temporary Sequence for Pushdown: 允许 PowerCenter 在数据库中创建临时的 Sequence 对象。如果在 Mapping 中包含 Sequence Generator，则必须启动这个选项。

至此，已经介绍了 Pushdown 最核心的部分，在 Pushdown 优化中还有一些更加复杂的概念及针对数据库的、函数的兼容性等问题，在 PowerCenter 联机帮助中均有详细的解释。

## 5.7 版本控制及部署

PowerCenter 提供了 Team Based Development 功能。这个功能集包括了版本管理、团队开发及部署相关的支持，这是项目管理不可或缺的部分。假如把项目按照大小做个简单分类，可以分为小项目和大中型项目两类。

- ◎ 小项目：1~2 人完成 ETL 工作。
- ◎ 大中型项目：ETL 开发人员较多，需要有一定的协作。

不同规模的项目在版本管理、团队开发及部署方面有不同的要求。理解这部分功能还是从案例开始。

**案例：**一个企业的大数据项目，数据开发、管理团队 10 人以上，按照开发人员的配比，ETL 工作至少有 5 人以上参与。这种项目的开发、测试、部署和上线过程如下。

**需求：**项目中至少有两套环境，如开发环境和生产环境，有的情况下还会有测试环境。为了便于描述，以两套环境为例进行介绍。开发人员在开发环境中导入数据源和目标结构，开发 Mapping、Session、Workflow 等，在完成测试后，将这些元数据部署到生产环境。在开发过程中希望锁定正在编辑的对象，以防被团队其他成员意外修改。为了保证历史版本的可用性，最好有版本的管理功能。还需要将开发完成的部分对象进行标记，支持后续自动化部署等。

### 5.7.1 Check In/Check Out

第一步：开发创建初始版本。

使用 PowerCenter Team Base Development，首先需要在 Admin Console 的 Repository

Service→Properties Tab 中打开 Version Control 选项，只有打开这个选项后才可以进行版本控制的相关操作，如图 5-74 所示。

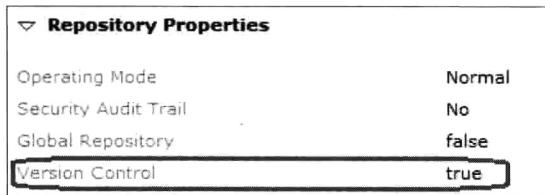


图 5-74

将 Version Control 设置为 true，即启动了版本控制。当一个 Repository 启动版本控制后，将不能取消这个功能。这时就可以进行有版本控制的对象的开发，如开发一个简单的 Mapping，包含一个 Expression Transformation，如图 5-75 所示。

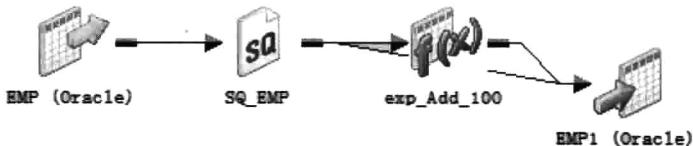


图 5-75

表达式的逻辑是将  $\text{SAL} := \text{SAL} + 100$ 。

启动版本控制后的 Repository，完成 Mapping 的开发之后，需要有一个额外的步骤，就是将开发、测试完成的对象 Check In 到 Repository 中。选择菜单 Versioning→Check In 命令，如图 5-76 所示。

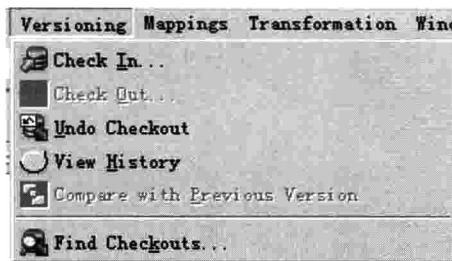


图 5-76

被 Check In 的对象是不可以被修改的，如果需要修改，则需要首先将对象 Check Out。

被用户 A Check Out 的对象，用户 B 将不能够对其进行修改，这样就可以防止两个用户同时对同一对象进行修改而造成冲突。

### 5.7.2 Team-Based 开发的一些有用功能

第二步：创建后续版本。

在这一步将展示 Team-Based Development 一些有用的功能。

- 查看历史（View History）。
- 对象比较（Compare Object）。
- 查看对象依赖（View Dependencies）。

当创建第一个版本后，后续可能由于各种原因，原来创建的 Mapping 有可能被修改，这时该对象就产生了多个版本。继续以上面的 Mapping 为例，后续将 SAL 修改为 SAL:=SAL-100。

在如图 5-76 所示的截图中，菜单中除了 Check In、Check Out 还有一个功能：View History。选择 View History 命令将看到如图 5-77 所示的窗口。

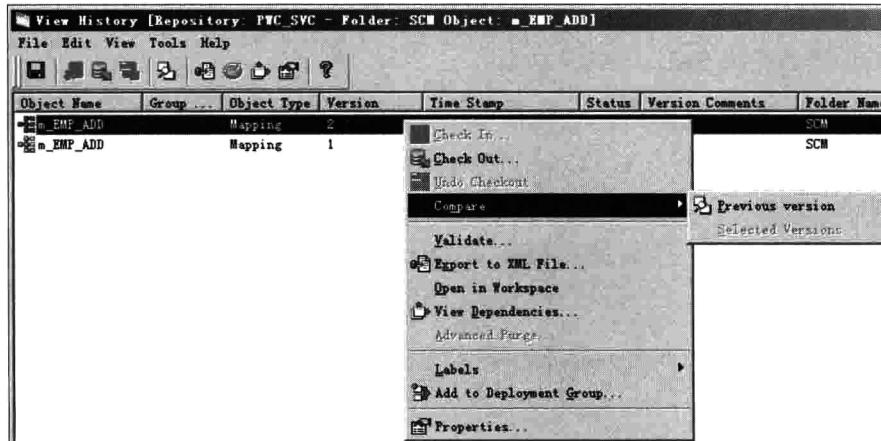


图 5-77

在图 5-77 中选择 Version 2，单击鼠标右键，在弹出的快捷菜单中选择 Compare→Previous version 命令，弹出如图 5-78 所示的对话框。这个功能就是进行对象比较，对象比较功能在很多操作界面中都可以通过菜单使用。

方框标记的部分为有差异的对象。这时，双击该对象，可以看到该对象更细节的差异。如图 5-79 方框标记部分，表达式由 SAL+100 修改为后续版本的 SAL-100。这也是 PowerCenter 元数据驱动架构的一个独特优势，可以跟踪到一个表达式，甚至是一个字段的类型、字段的长度，如字段的长度由 10 变为 20 等。

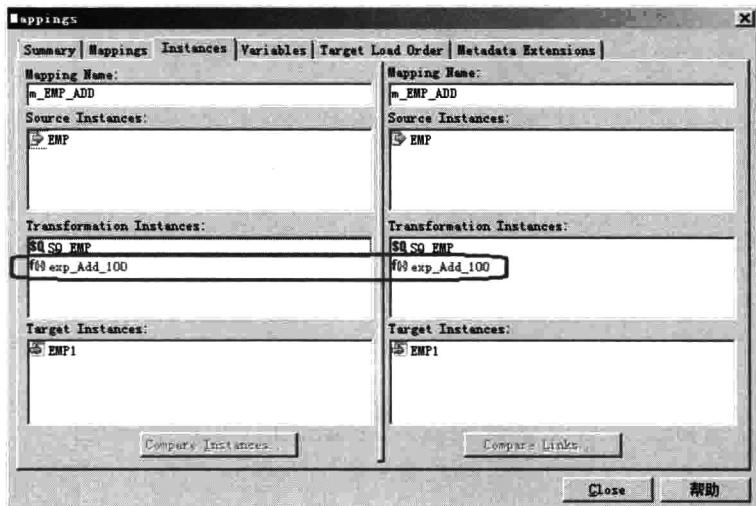


图 5-78

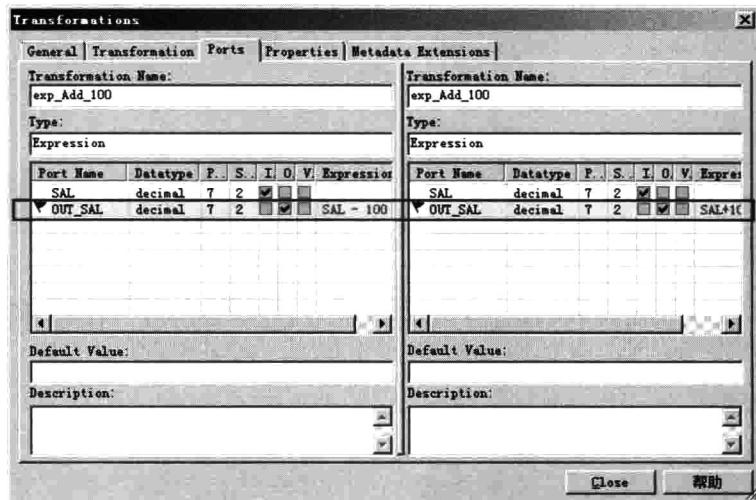


图 5-79

还有一个功能叫作 View Dependency（查看依赖关系）。例如，需要对数据源 EMP 进行修改，希望查看所有受此修改影响的 Mapping 及其他对象，这时候就需要用到“查看依赖关系”这个功能。可以在 Repository Manager 中用鼠标右键单击任意对象，在弹出的快捷菜单中选择 View Dependency 命令，可以看到如图 5-80 所示的对话框。

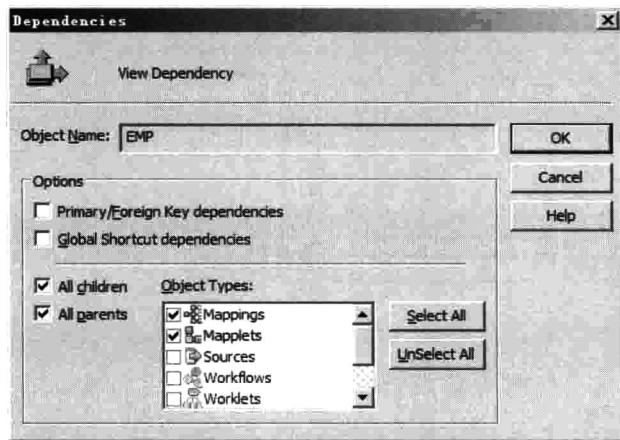


图 5-80

图 5-80 所示对话框的含义是对对象 EMP 而言，可以查看所有与其相关的 Mapping、Mapplet、Source 等。选择关注的对象类型，然后单击 OK 按钮，就可以查看所有与 EMP 相关的对象。

### 5.7.3 Label 与 Deployment Group

Label 和 Deployment Group 是 PowerCenter 为版本控制及部署提供的两种支持能力，使用它们可以支持更加快捷、灵活的部署。

Informatica 为了使项目顺利执行，并符合项目管理的规范，提供了多种部署方案，包括：

- 使用 Deployment Group 支持部署。
- 使用 pmrep 命令进行部署。
- 使用图形化拖曳的方式进行部署。
- 与第三方软件配置管理、部署工具进行集成。

这几种方式并非完全孤立，可以根据项目的需要相互配合，支持项目的配置管理工作。

第三步：使用标签，并进行部署。

当一个 Mapping/Session 经过充分测试后，将按照项目的时间要求进入准备部署状态。这时就需要使用 PowerCenter Label 功能。Label 的概念和管理方法与软件配置管理中的概念相同，用于支持项目的 SCM（Software Configuration Management）。

在使用 Label 之前，必须预定义一些 Label。在 Repository Manager 中，选择菜单 Versioning→Labels 命令来创建 Label。Label 在没有使用之前有一些预定义的名字，当为其赋予一个对象后才有含义。

如图 5-81 所示，已经创建了两个为部署服务的 Label，分别是 Deploy 1 和 Deploy 2。

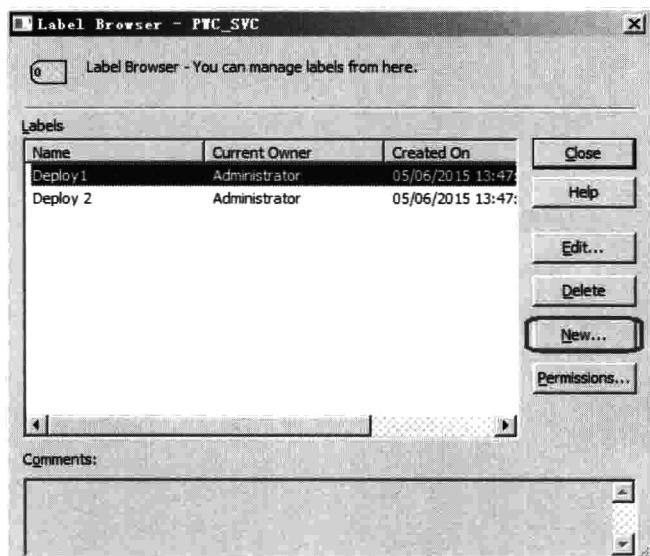


图 5-81

定义好 Label 之后，就可以对已经测试完成的 Mapping/Session 或者其他对象使用 Apply Label。在 Repository Manager 中选择菜单 Versioning→ Apply Label 命令，选择需要 Label 的对象，并使用 Label。如图 5-82 和图 5-83 所示是对 Mapping m\_EMP\_ADD 使用 Label Deploy 1 的过程。图 5-82 为选择要 Label 的对象。图 5-83 为是否要对相关的对象使用 Label。

在 Label 完成之后，就可以以 Label 为过滤条件使用 Deployment Group 进行部署。当

然进行部署并非一定要有 Label，不过在大中型项目中一般是需要的。可以根据具体的项目需求综合考虑配置管理、部署的方式。

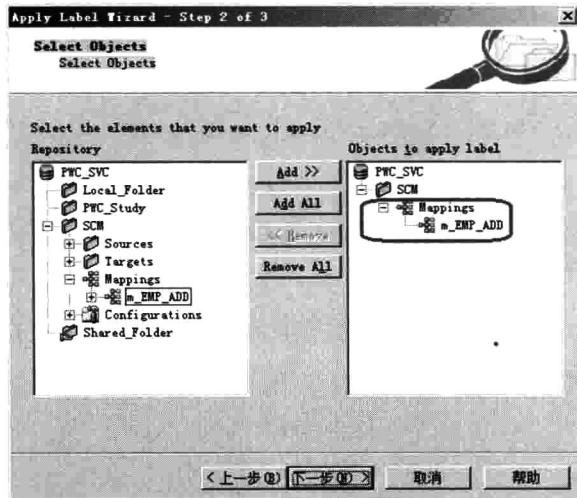


图 5-82

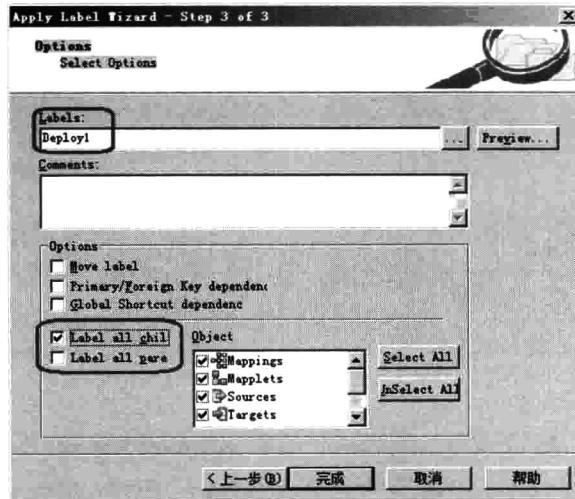


图 5-83

Deployment Group 是将要进行部署的对象整合到一个管理对象中的一种机制。PowerCenter 中提供了两种类型的 Deployment Group。

- ◎ 静态的 Deployment Group。静态的 Deployment Group 中物理地包含具体的 PowerCenter 对象。
- ◎ 动态的 Deployment Group。动态的 Deployment Group 包含的是一个查询对应的对象集合，只要满足这些查询条件的对象都属于这个 Deployment Group。例如，Label 为 Deploy 1 的所有对象。

创建 Deployment Group，打开 Repository Manager，选择菜单 Tools→Deployment→Group 命令，在弹出的如图 5-84 所示的对话框中就可以创建新的 Deployment Group。



图 5-84

如果创建的是 Static Deployment Group，只需要为其提供一个名字即可。如果创建的是 Dynamic Deployment Group，还需要为其指定一组查询条件，如图 5-84 所示。假如，在这里创建两个 Deployment Group，分别是 Dynamic\_deploy\_Group 和 Static\_deploy\_Group，那么在 Repository Manager 的 Navigator 中会看到如图 5-85 所示的视图。

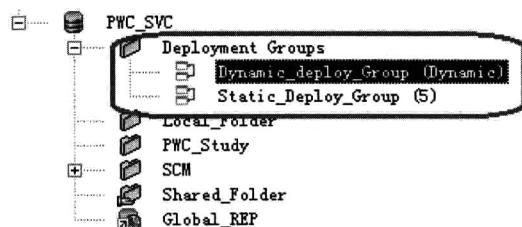


图 5-85

Dynamic\_deploy\_Group (Dynamic) 括号中的 Dynamic 表明其是一个动态的部署组，而 Static\_Deploy\_Group (5) 中的数字 5 表明在这个静态部署组中已经包含了 5 个待部署的对象。由于动态部署组的内容是由一个查询动态产生的，若想查看其保存的对象，需用鼠标右键单击 Deployment Groups，在弹出的快捷菜单中选择 View contents 命令，如图 5-86 所示。

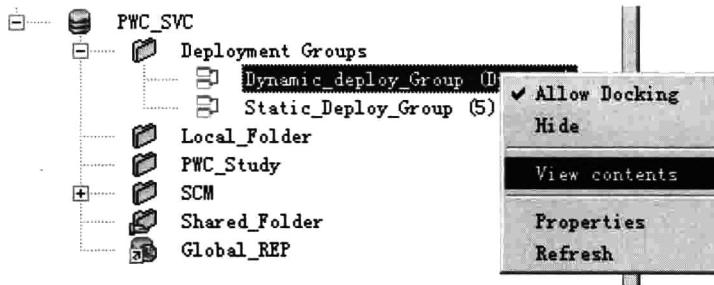


图 5-86

将对象加入静态部署组，只需要将 Folder 中的对象拖入静态部署组即可。

#### 5.7.4 复制对象从开发 Repository 到生产 Repository

PowerCenter 支持多种方法将对象从开发 Repository 复制到生产 Repository。在 PowerCenter 的各个客户端中都有部分的导入、导出功能，可直接在 Repository 之间复制不同对象。如果是进行生产的部署，笔者建议使用 Repository Manager。从开发 Repository 向生产 Repository 部署一般采用两种方式：

- ◎ 使用 pmrep 命令进行部署。
- ◎ 使用图形化拖曳的方式进行部署。

可以拖曳或者部署的对象包括：

- ◎ 开发对象，如 Source、Target、Mapping、Session、Workflow 等。
- ◎ Folder。
- ◎ Deployment Group。

使用命令行的方式进行部署，请参考后续章节关于 pmrep 命令的介绍。如果使用拖曳方式进行部署，有以下两点需要注意：

(1) 拖曳一个开发对象到生产 Repository 时，在放下对象时，应该将对象放入对应的文件夹。

(2) 如果拖曳 Folder 或者 Deployment Group，在放下对象时，应该将对象放到 Repository 上，而不是具体的 Folder 或者 Deployment Group 上。

# 第 6 章

# PowerCenter 实战汇总

---

这一章叫作实战汇总，主体并非介绍具体的 Mapping 怎么开发，关于 Mapping 的开发技巧及方法已经以案例的形式嵌入到组件的介绍中。因此在这一章主要关注的并非 Mapping 开发，而是在 PowerCenter 实战中非常有用、常用的一些功能，希望对读者实际的使用有些助益。

## 6.1 PowerCenter 字符集

---

字符集问题是一个极其复杂的问题，笔者这么多年的从业经验也没有真正完全理解和解决所有的字符集问题。原因是世界上的语言极其复杂、多样，一种语言又可以通过多种字符集实现，这些字符集之间虽然多数相同，但也有细微的差别，经常出现个别生僻字的乱码。本节争取为大家提供一些设置字符集的最佳实践，帮助用户解决在 PowerCenter 的安装、配置、使用过程中遇到的多数字符集问题。

### 最佳实践

- ◎ 了解源或者目标数据库的字符集，并在 PowerCenter 服务器上设置相关的环境变量或者完成相关的设置，不同的数据库有不同的设置方法。
- ◎ 多数字符集的问题，需要源读和目标写分开来考虑，而不是混为一谈。

下面针对每种不同的类型分开来讨论。

### 6.1.1 Oracle 数据库

#### 1. 获取 Oracle 数据库字符集的方法

以 DBA 授权的用户登录，例如：

```
conn system/<password> as sysdba  
SELECT value$ FROM PROPS$ WHERE NAME='NLS_CHARACTERSET'
```

这条 SQL 语句可以返回数据库本身的字符集，如 AL32UTF8。

#### 2. 设置 Oracle 环境变量

解决 Oracle 字符集，首选的方法是使用 Oracle 提供的 NLS\_LANG 环境变量。例如，  
NLS\_LANG= AMERICAN\_AMERICA.AL32UTF8。其中 AL32UTF8 为上面的 SQL 语句的  
返回值。

Microsoft Windows 平台需要在注册表或者环境变量中进行设置；UNIX/Linux 平台需要  
在 Profile 文件中进行设置。

#### 3. 特殊情况一：数据库字符集和 NLS\_LANG 不一致

例如，Oracle 数据库字符集是 ASCII 码，但是其写入数据时使用的 NLS\_LANG 为  
ZHS16GBK，这种情况如何处理？

解决方法：以写入字符集为准，即设置 PowerCenter 服务器环境变量为 NLS\_LANG=  
AMERICAN\_AMERICA.ZHS16GBK。

这种情况也会出现在 Java 程序写数据库的情况，很多程序员开发的 Java 程序并没有使  
用 NLS\_LANG 变量，而是在 Java 代码中进行了指定。这时候要参照 Java 代码使用的字符  
集的情况。

#### 4. 特殊情况二：数据源有两个 Oracle 数据库，但字符集不一样

数据源有两个 Oracle 数据库，但字符集不一样，如数据库 A 的字符集是 AL32UTF8，  
另外一个数据库 B 的字符集是 ZHS16GBK。

这时可以创建两个 Integration Service，分别服务于这两个字符集。这时 NLS\_LANG 的  
环境变量可以设置在 Integration Service 上。

打开 Admin Console，选择 Integration Service，在 Processes Tab 中设置 Environment Variables，如图 6-1 所示。



图 6-1

也可以是一个数据库访问使用 Oracle Native 驱动，Native 驱动使用 NLS\_LANG 环境变量；另外一个数据库访问使用 ODBC 驱动，ODBC 驱动使用参数 IANAAppCodePage。

### 5. 特殊情况三：数据源和目标都是 Oracle，但是它们的字符集不一样

在这种情况下，笔者一般建议采用两种接口的方式，例如，源使用 Native 驱动，使用 NLS\_LANG，目标写采用 ODBC，在 odbc.ini 文件中设置对应的字符集信息，使用参数 IANAAppCodePage，113 是 GBK。

#### 6.1.2 DB2 字符集

##### 1. 获取 DB2 字符集的方法

执行 db2set -all 命令可以查看服务器的字符集。

##### 2. 设置 DB2 客户端字符集

使用如下命令：

```
db2set DB2CODEPAGE=819
```

DB2CODEPAGE 有几个常用的值，如 1386(中文)、1208(UTF8)、819 对应 ISO8859-1、1200 对应 16 位 Unicode。

假如有多个 DB2 数据库，且字符集不一样；假如同时将 DB2 作为源和目标，但是字符集不一样；假如……和 Oracle 数据库遇到的问题是一样的，因此处理的方法也类似。

#### 6.1.3 AS/400 字符集

AS/400 是一个古老的系统，或者叫 Mid-Range 系统，它的字符集非常复杂，其复杂性

表现在以下方面：

- 可以使用自定义字符集。
- 每个字段都可以有独立的字符集。
- 有的用户在使用 65535 这个保留字存放中文。

AS/400 上的数据库叫作 DB2/400，因此在简单的情况下，你可以认为它就是一个 DB2 数据库，可以使用 DB2（UDB）客户端访问存放在里面的数据。

假如出现不同字段使用不同字符集的情况或者其他复杂状况，一般建议使用 PowerExchange for AS/400，在 PowerExchange for AS/400 中可以针对字段定义对应的字符集，也可以处理在 65535 中存放中文的情况。如果是不同字段使用了不同的字符集，最佳的方案是使用 PowerExchange Data Map。

当使用 ODBC 或者 UDB 客户端访问 AS/400 时，可以使用 CAST 函数控制 AS/400 字符集。例如，CAST(CAST(CUSNM AS CHAR(45) CCSID 65535) AS CHAR(45) CCSID 1388)，其中 CUSNM 是一个字段名，CCSID 可以简单地认为是 DB2/400 字符集。这个例子展示的就是在保留字 65535 中存放了 1388 字符集的情况。

在通过 PowerExchange 实时访问 AS/400 的情况下，经常会在 dbmover.cfg 文件中使用的参数包括：

```
DB2_BIN_AS_CHAR=Y
DB2_BIN_CODEPAGE=(037,935)
CODEPAGE=(,ibm-1388_P103-2001,ibm-1388_P103-2001)
```

但这不是全部的参数，还有一些内部的参数可供使用。因此在访问 AS/400 的过程中遇到字符集的问题，求助的路径有两条：一是尝试 PowerExchange；二是请求 Informatica 支持。

#### 6.1.4 ODBC 字符集

在 Oracle 字符集的部分，曾提到过 ODBC 字符集的设置。对 UNIX/Linux 操作系统而言，字符集是通过参数 IANAAppCodePage 来控制的，这个参数设置在 odbc.ini 文件中。

IANAAppCodePage 比较常用的编码包括：113（GBK）、2026（Big5）、4（ISO 8859-1）。

→ **注释**

假如 Oracle 字符集是 ZHS16GBK，而 ODBC 中使用 IANAAppCodePage=113，对某些生僻字就有可能产生乱码，因为 ZHS16GBK 不完全与 113 相等。笔者没有仔细考证过这两个字符集，但是理论上，这两个字符集在某些特例（如生僻字）中可能会不一致，从而造成显示乱码。也就是说此中文不一定是彼中文，大多数编码可能相同，但是也有微小的差异。

### 6.1.5 文本文件字符集

有的项目中，因为接口非常多，常常会对文件的编码方式有明确的约定，例如，尽管都是中文的数据，但是接口文件可能约定为必须使用 GBK 或者 UFT-8 的编码。这时候需要在 PowerCenter 的数据源或者目标中特别指定文本文件的编码方式。

在 PowerCenter 中对文本文件源和目标定义字符集有两个地方。

- 在 PowerCenter Designer 中的源/目标定义阶段。双击任意一个文件源或者目标，选择 Advanced，就会看见如图 6-2 所示的对话框，可以设置文本文件的相关属性，其中一个重要属性就是字符集设置。

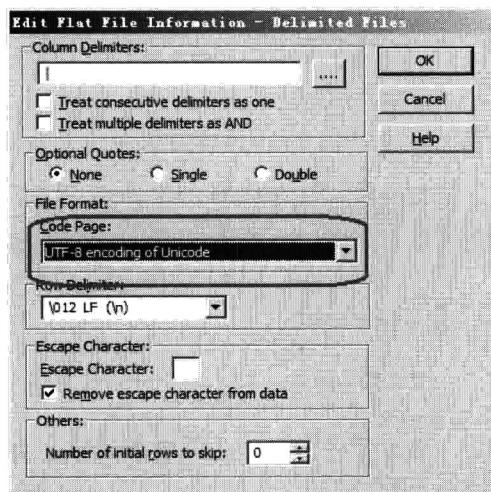


图 6-2

- 在 Workflow Manager 中 Session 的源和目标属性中。对一个包含文本文件源或者

目标的 Session，双击 Session，选择 Mapping Tab，单击 Set File Properties 链接，就可以看见和在 PowerCenter Designer 中看到的类似的对话框，可以在其中设置文本文件的字符集。Set File Properties 不是很容易看到的，请参考图 6-3。

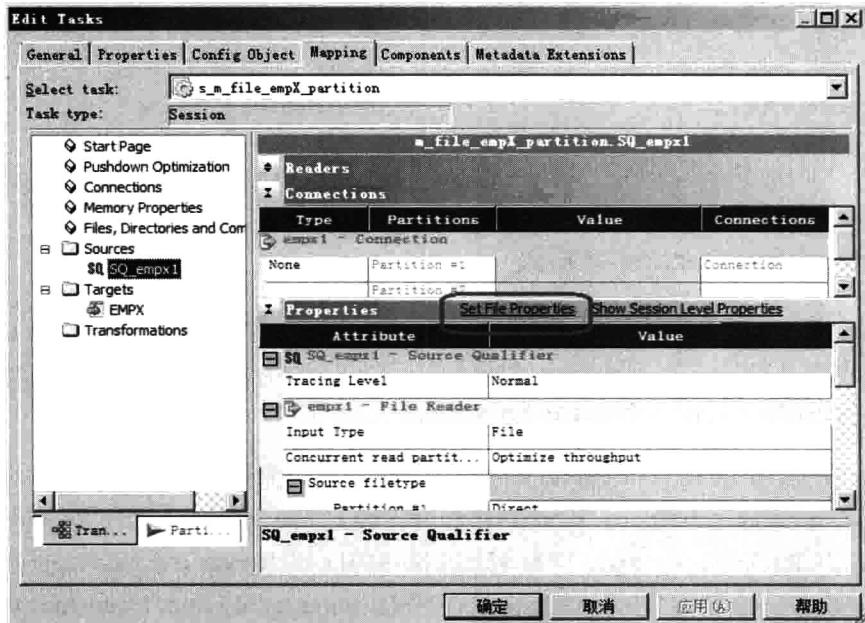


图 6-3

假如在 Designer 和 Workflow Manager 中设置的字符集不一致，以 Session 中的设置更为优先。在项目中，建议做统一的规范。

### 6.1.6 Repository Service 字符集

PowerCenter 要求 Repository Service 字符集必须是 Integration Service 字符集的超集，或者说它应该是任意数据字符集的超集。因此在中文环境的实践中，Repository Service 经常设置的字符集只有两种：“UTF-8 encoding of Unicode”或者“Microsoft Windows Simplified Chinese, superset of GB 2312-80, EUC encoding”。

Repository Service 的字符集一旦创建将不能修改。并且如果备份/恢复 Repository 内容时，要求导入的备份 Repository 必须是新 Repository 字符集的子集，或者二者完全相同。

查看 Repository Service 字符集，需要打开 Admin Console，选择要查看的 Repository Service，再选择其 Overview Tab 就可以进行查看，但不能修改。如图 6-4 所示是一个设置为 UTF-8 Encoding of Unicode 的 Repository Service。

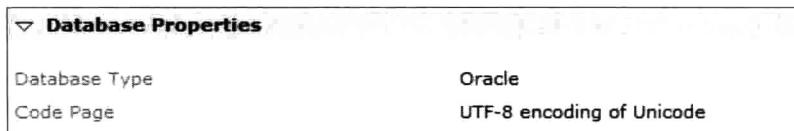


图 6-4

### 6.1.7 Integration Service 字符集

Integration Service 字符集设置是比较关键的，一般建议使用“UTF-8 encoding of Unicode”或者“Microsoft Windows Simplified Chinese, superset of GB 2312-80, EUC encoding”。这两种字符集是在中文环境中最常用。

但是，在启动 Integration Service 时，有时会遇到类似的错误提示：“serviceType serviceName severity timestamp threadName messageCode message IS ISwinutF8 0\$:\_ \$FATAL 09/26/2012 22: 55: 51.237 PM 1520 LM\_36011 Code page mismatch. Service process is running in code page [MS Windows Latin 1 (ANSI), superset of Latin1] whereas the service is configured in the Admin Console to run in code page [UTF-8 encoding of Unicode]。”这是由于在默认情况下，PowerCenter 会使用操作系统的默认设置。对 Microsoft Windows 操作系统，PowerCenter 会自动识别操作系统字符集设置。对 UNIX/Linux 操作系统，PowerCenter 会默认使用环境变量 LANG。那么如何设置需要的字符集呢？PowerCenter 提供了一个环境变量：INFA\_CODEPAGENAME。

在中文环境下，常用的 INFA\_CODEPAGENAME 值如表 6-1 所示。

表 6-1

字符集	INFA_CODEPAGENAME 值
HKSCS	Hong Kong Supplementary Character Set
MS936	Microsoft Windows Simplified Chinese, superset of GB 2312-80, EUC encoding
MS950	Microsoft Windows Traditional Chinese, superset of Big 5
UTF-8	Unicode Transformation Format, multibyte

- ② 在 UNIX/Linux 操作系统上设置环境变量使用：`export INFA_CODEPAGENAME = UTF-8。`
- ③ 在 Windows 操作系统上设置：选择高级系统设置→环境变量。

Integration Service 的字符集设置是在 Integration Service 创建的时候，也可以在创建后进行修改。在 Admin Console 中选择 Integration Service，选择 Processes Tab，如图 6-5 所示，在这里可以查看和修改 Integration Service 的字符集。

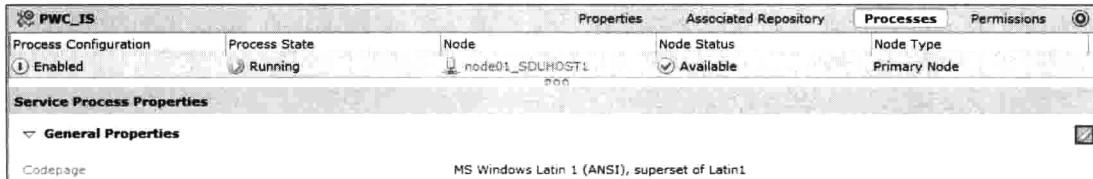


图 6-5

### 6.1.8 Data Movement Mode

PowerCenter Integration Service 除了 Codepage 属性，还有一个关于字符集的属性，叫作 Data Movement Mode。Data Movement Mode 有两个选项：ASCII 和 Unicode。

- ④ ASCII Data Movement Mode：当源或者目标都是 7bit ASCII 或者 EBCDIC 时使用 ASCII Data Movement Mode。在这种模式下，PowerCenter 以单字节的方式存储、传输数据。在传输过程中不做任何的字符集转换。

但在多字节环境下偶尔也会使用 ASCII 模式，主要是利用其不进行字符集转换的特点实现所谓的字符集 Garbage in, Garbage Out。当源和目标是同构数据库时，这种模式对特别复杂的字符集情况有时候非常有效。

- ⑤ Unicode Data Movement Mode：当源和目标是多字节数据时，要使用 Unicode Data Movement Mode。当使用 Unicode 时，PowerCenter 会执行字符集转换。当从数据源读到数据后，PowerCenter 先将其转换为 UCS-2 再进行处理；当向目标写数据时，PowerCenter 将数据由 UCS-2 转换为目标字符集。

哪种 Data Movement Mode 性能更快呢？这个问题比较简单，肯定是不做字符集转换的更快，那答案就是 ASCII 模式更快。

## 6.2 UNIX ODBC 配置

根据 PowerCenter 的不同版本, PowerCenter 支持通用的 ODBC Type II 或者 ODBC Type III。同时 PowerCenter 安装包中集成了 DataDirect ODBC 驱动, DataDirect ODBC 包含了 ODBC 驱动管理器和 ODBC 驱动程序。PowerCenter 默认安装的 ODBC 驱动如下。

- DataDirect 7.1 DB2 Wire Protocol。
- DataDirect 7.1 Greenplum Wire Protocol。
- DataDirect 7.1 MySQL Wire Protocol。
- .....
- Informatica Cassandra ODBC Driver。
- Informatica Data Services ODBC Driver 9.6.1。
- Informatica MongoDB ODBC Driver。

### 6.2.1 ODBC 常规配置

配置 PowerCenter ODBC 驱动支持, 有几个例行步骤。

- 安装 ODBC 驱动。如果需要访问的数据库, PowerCenter 安装过程中已经提供了对应的驱动程序, 可以不再重新安装。对 PowerCenter 没有提供的 ODBC 驱动, 则需要单独安装, 如 Sybase IQ 或者 MySQL 社区版等。
- 编辑/创建 odbc.ini 文件, 设置 ODBCINI 环境变量。

PowerCenter 安装过程中提供了一个 odbc.ini 样例文件, 可以将此文件作为模板进行修改, 或者创建一个全新的 odbc.ini 文件。如下是一个 SQL Server 的 odbc.ini 文件的样例。

```
[MysqlServer]
Driver=/opt/informatica/datadirect/lib/XOmsss22.so
Description=DataDirect 5.2 SQL Server Wire Protocol
Address=<SQLServer_host, SQLServer_server_port>
AlternateServers=
AnsiNPW=Yes
ConnectionRetryCount=0
ConnectionRetryDelay=3
```

```

Database=<database_name>
LoadBalancing=0
LogonID=
Password=
QuotedId=YES
SnapshotSerializable=0

```

当 Session 运行时，PowerCenter 如何找到这个文件呢？PowerCenter 搜索 odbc.ini 有个优先顺序，不过记住准确的顺序作用并不高。一般情况下只要坚持两个原则，就不需要记住所谓搜索 odbc.ini 文件的优先顺序问题：一是不要编辑多个 odbc.ini 文件；二是正确设置如下两个环境变量。

```

export ODBCHOME=<Informatica server home>/ODBC7.1
export ODBCINI=$ODBCHOME/odbc.ini

```

- 设置 Library Path 环境变量。设置环境变量非常重要，不同的环境变量设置会影响 ODBC 驱动的调用，一定要保证将新安装的 ODBC 驱动或者 PowerCenter 自带的驱动程序路径设置到 Library Path 中。不同操作系统的 Library Path 的指定方法略有不同，参考表 6-2。

表 6-2

操作系统	环境变量
Solaris	LD_LIBRARY_PATH
Linux	LD_LIBRARY_PATH
AIX	LIBPATH
HP-UX	SHLIB_PATH
Windows	PATH

- 测试相关配置是否有效。测试相关的环境变量是否有效有很多不同的方法，最常用的工具有以下两个。
  - ◆ ldd 命令。使用 ldd 命令可以查看 UNIX/Linux Library 的调用顺序；检查 Library Path 的设置或者变更的影响。使用方法如：

```
ldd /opt/informatica/datadirect/lib/XOmsss22.so
```

如果发现其中某些 Library 无法找到或者与期望不一致，就需要修改 Library Path 环境变量。

这个命令在 ODBC 设置时非常重要，因为目前提供 ODBC 驱动的厂家很多，并且不同的厂家可能在驱动中使用了相同的文件名，但实现并不完全一致，如文件 libodbcinst.so。这是造成 ODBC 使用时报错的一个重要原因。因此，在配置或者变更 Library Path 后最好用 ldd 命令做一次检查，查看这些配置和变更的影响。

- ◆ 使用 ODBC 测试工具进行测试。一般数据库本身提供了 ODBC 客户端工具，如 Syabase IQ 的 dbisql 等。同时 PowerCenter 也提供了一个通用测试工具 ssgodbc，可以通过 mysupport.informatica.com 网站下载其可执行文件和使用说明。

## 6.2.2 MySQL 社区版 ODBC 配置

现在大多数的数据库都提供 ODBC 驱动，但不一定是 DataDirect 支持的 ODBC 或者不是包括在 Informatica 与 DataDirect 协议中的 ODBC。这时候就会经常遇到 ODBC 报错的情况，如使用 MySQL 社区版的用户经常遇到 PowerCenter 提示不支持此版本。

解决方法也很简单，配置 PowerCenter 支持 MySQL 社区版，需要完成以下 3 个步骤。

(1) 到 Oracle 官网下载相应版本的 MySQL 的 ODBC 驱动 RPM 包并在 PowerCenter 服务器上安装。

(2) 将安装后的 MySQL Library 路径添加到 LD\_LIBRARY\_PATH (假设是 RHEL) 中。这里有个建议，尽量将新安装的 MySQL 驱动放到 LD\_LIBRARY\_PATH 的最前面，避免继续使用 DataDirect 驱动管理器造成的报错。

(3) 修改 odbc.ini 文件，添加 MySQL ODBC 驱动信息，参考如下样例：

```
[MySql2]
Description=MySQL Driver
Driver=/home/infa/Informatica/9.1.0/ODBC6.1/mysql-odbc/lib/libmyodbc5w.so
SERVER=<IP Address>
PORT=3306
USER=
Password=
Database=<Database>
OPTION=3
SOCKET=
```

```

prefetch=500000
DriverUnicodeType=1

```

确保这 3 个步骤配置的正确性，就可以安然无忧地使用 PowerCenter 访问 MySQL 社区版了。从原理上来讲，这样配置可以成功的原因是：PowerCenter 声明的是支持 ODBC Type II 或者 Type III，同时在安装过程中集成了 DataDirect ODBC 驱动。那么，对 MySQL 社区版来说，它是支持 ODBC Type II 或者 Type III 的，但是 DataDirect 所带的 ODBC 驱动仅仅支持 MySQL 商业版。因此当配置 MySQL ODBC 驱动时，看到 PowerCenter 已经自带了一个 MySQL 驱动，那么当然就要使用它了，这样就会遇到 PowerCenter 提示不支持社区版的问题。

因此，解决方案就是下载、安装 MySQL 驱动程序，设置 Library Path 确保新安装的驱动程序被使用，利用 PowerCenter 支持 ODBC Type II 或者 Type III 来支持 MySQL 社区版。这个方法对解决通过 ODBC 访问其他的数据库也是一样的。

## 6.3 使用 Mapping 动态分发文件

**案例：**ABC 公司是一个地区分布的组织，在一个项目中他们希望将总部的数据分发到各个区域公司去。由于网络速度的限制，无法将数据直接写入区域公司的数据库，他们希望为不同的区域公司生成不同的文件，以方便各个子公司分别获取各自的数据。

以下的演示以部门（DEPT）表为例，LOC 列为区域标志。这张表中共有 5 条数据，期望将属于不同区域的数据放入以区域名称命名的文件中。表中的数据如下：

DEPT	NO	DNAME	LOC
50	R		DALLAS
10		ACCOUNTING	NEW YORK
20		RESEARCH	DALLAS
30		SALES	CHICAGO
40		OPERATIONS	BOSTON

期望的输出是 4 个文件，分别为 DALLAS.dat、NEW YORK.dat、CHICAGO.dat 和 BOSTON.dat，它们分别包含各自区域的数据。

在这个 Mapping 中，综合使用多个 Transformation，包括 Sorter、Expression、Transaction

Control 组件，完成后的 Mapping 如图 6-6 所示。

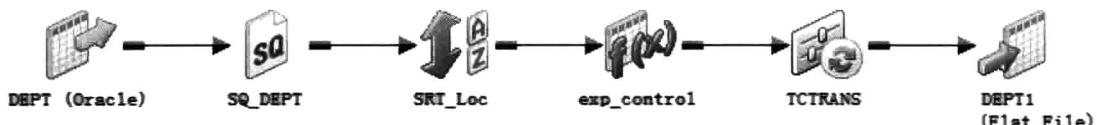


图 6-6

在这个实战案例中，还会使用 Target 的文件名列；使用 Expression 的 Local Variable；使用 Sorter 进行排序；使用 Transaction Control 控制文件写。

首先看目标文件设计，通过使用“Add Filename column to this table”增加了 FileName 列，如图 6-7 所示。

Edit Tables							
		Table		Columns		Properties	
		Select table:		DEPT			
	Column Name	Datatype	Prec	Scale	No.	Format	Key Type
1	DEPTNO	number	2	0	<input checked="" type="checkbox"/>		PRIMARY KEY
2	DNAM	nstring	14	0	<input type="checkbox"/>		NOT A KEY
3	LOC	nstring	13	0	<input type="checkbox"/>		NOT A KEY
4	FileName	string	255	1	<input checked="" type="checkbox"/>		NOT A KEY

图 6-7

其次看 Sorter 组件，对 LOC，即区域字段进行排序，如图 6-8 所示。

Edit Transformations							
		Transformation		Ports		Properties	
		Select		SRT_Loc			
		Transformation		Sorter			
	Port Name	Datatype	Prec	Scale	I	O	Key
1	DEPTNO	decimal	2	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	DNAM	nstring	14	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	LOC	nstring	13	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

图 6-8

再次看 Expression Transformation，使用 Local variable 判断分组是否结束，并生成动态的文件名，如图 6-9 所示。

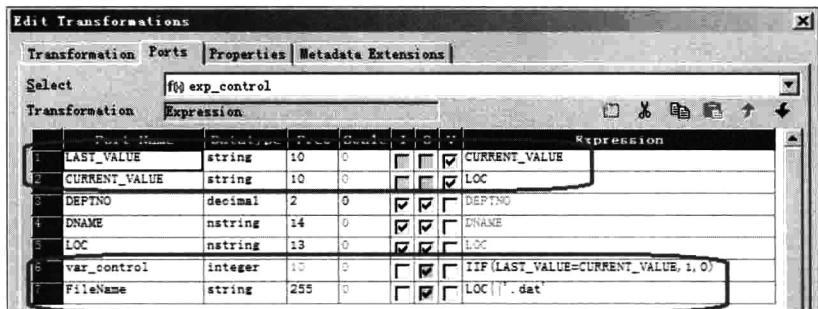


图 6-9

最后在 Transaction Control 中使用如下的事务控制表达式：

```
IIF(var_control=1, TC_CONTINUE_TRANSACTION, TC_COMMIT_BEFORE)
```

这时，一个完整的 Mapping 已经设计完毕。Workflow 设计与其他的 Workflow 并没有任何不同，尤其是目标文件名的位置也不需要进行任何修改。运行这个 Workflow，会看到一个 Size 为 0 的文件，它是 PowerCenter 默认生成的文件；其他的 4 个文件为期望的文件，分别包含了各自区域的数据。

## 6.4 超越 EDW，商品自动价格跟踪

**案例：**任何一家商品生产、销售型企业都需要根据市场的价格、产品的销量确定自己的生产规划、市场投放计划及价格体系。以一家电话机生产、销售企业为例，这是一个竞争非常激烈的行业，因此公司市场部需要非常紧密地跟踪市场变化，了解竞争对手商品的销售价格和销量。这家企业已经建立了内部的 BI 系统，但是他们更想了解的是市场上同类商品的销售情况。以往他们是通过市场调研公司获取这部分数据的。通过多年的观察，这家企业也注意到，市场调研公司获取的数据不及时，甚至有时也不一定准确，无法提供个性化的信息。例如，这家电话机公司曾想获取按照颜色或者特定功能分类的电话机的销售情况，这时市场调研公司是无法完成的。因此市场部总监想到在互联网非常发达的今天，是否有可能通过互联网平台获取这部分数据，如淘宝网、京东商城的数据。

根据市场部的需求，Informatica 工程师帮用户整理了项目的逻辑架构，并与用户对项目的实现细节、难点进行了沟通。系统架构如图 6-10 所示。

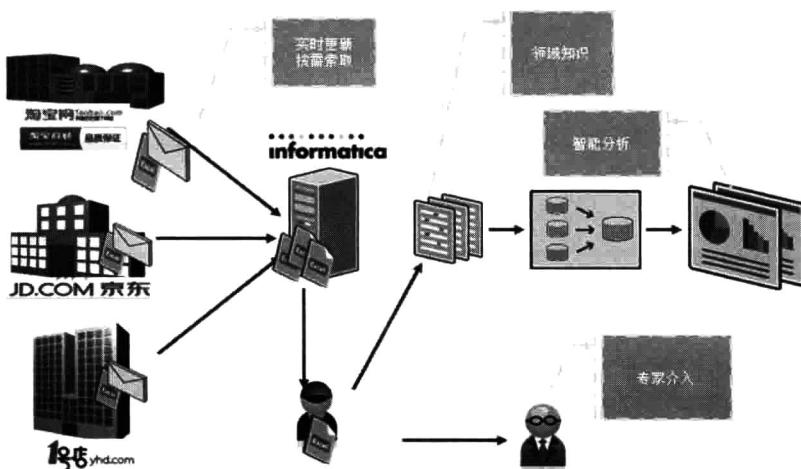


图 6-10

这个系统包括以下几个重要的组成部分。

(1) 数据获取：按条件获取淘宝网、京东商城的数据。淘宝网、京东商城的平台上成千上万的商品信息，对一家企业而言，关心的仅仅是某一类商品，如电话机。京东商城、淘宝网是一个 Web 平台，因此这个阶段的输出是 HTML 文件，如图 6-11 所示。



图 6-11

解决这个问题使用 Informatica Data Transformation 是个不错的选择。当然可以实现这部分功能的产品很多，开发人员可以根据自己的需要灵活选择。

(2) 获取企业关心的信息：从上一步获取的 HTML 文件中获取企业关心的数据，这部分是关键步骤之一，如图 6-12 所示。



图 6-12

解决这个问题，非 Informatica Data Transformation 莫属。Data Transformation 可以帮助开发人员以图形化的方式对非结构化数据进行解析，如解析 HTML、Word、Excel、PDF 等，将各种有价值的数据从非结构化数据中分离出来。在本书中无法对 Data Transformation 的使用进行详细的介绍，对这方面开发感兴趣的读者可以参考 Data Transformation 提供的 Tutorials。Tutorials 的详细信息在 PowerCenter Client 安装目录~\clients\DT\tutorials\Exercises 下，里面有详细的案例需求及开发过程文档。

Data Transformation 的输出如下所述：

- ◎ 它的直接数据输出是 XML 文件，这是因为非结构化数据包含的信息不一定能用一个简单的二维结构来描述，使用 XML 描述则具有更强的包容性。
- ◎ 其输出还可以是一个部署的服务，可以通过其他中间件进行调用，如 PowerCenter 中的 UDO（Unstructured Data Option）。

(3) 解析 DT XML 输出为二维结构，这个工作在 PowerCenter 中完成。使用 UDO 和 XML Parser 两个 Transformation 组合。这个 Mapping 大致如图 6-13 所示。

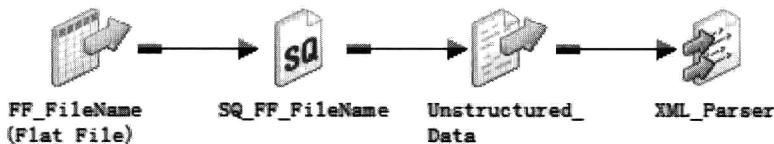


图 6-13

这个 Mapping 并不复杂, FF\_FileName 用于获取 HTML 文件路径; Unstructured Data 用于调用 Data Transformation 生成的服务, 将 HTML 解析为 XML; XML\_Parser 用于将 XML 转换为多张二维数据表。所有的操作都是在内存中完成的, 并不产生任何的落地文件。

到目前为止, 数据处理工作并没有结束。假如仔细观察淘宝网关于产品的描述, 会注意到这是一个很长的字符串, 里面包含了大量用于吸引客户、增加被搜索命中信息, 如“新款 摩托罗拉 数字无绳电话机 办公子母机 家用无线座机 一拖一 中文菜单 免提通话 免费对讲 分机扩展 套餐优惠”。

#### (4) 分词、语义识别与专家知识集成。

对于本案例中关于产品的描述:“新款 摩托罗拉 数字无绳电话机 办公子母机 家用无线座机 一拖一 中文菜单 免提通话 免费对讲 分机扩展 套餐优惠”, 所有的词已经用空格进行了分隔, 分词已经比较简单, 但是还有很多产品的描述还未使用分隔符进行分隔。这时比较有效的方法是使用 Data Quality 中的 Label/Parser Transformation 进行分词或识别。

语义识别同样需要 Data Quality 中的 Label Transformation, 将这些关键词进行分类管理, 用于后续的分析和使用。

不管是在分词还是语义识别过程中, 并非仅仅使用技术手段就可以完成的, 还需要集成领域专家知识, 在 PowerCenter 中被称为字典。字典需要领域专家不断地进行维护和增强。

不考虑分词过程, 最终的输出如图 6-14 所示。

产 品	价 格	促 销 价	月 销 量	累 计 评 价
新款 摩托罗拉 数字无绳电话机 办公子母机 家用无线座机 一拖一 中文菜单 免提通话 免费对讲 分机扩展 套餐优惠	588 元	219 元	1995 部	873 条

图 6-14

以上的数据只是一个样例输出，实际可获得的信息会更多。汇总如上的信息，可以帮助该电话公司以天为单位获取最新的电话机销售数据，跟踪最新的市场趋势，加强公司对市场的理解。

## 6.5 pmcmd 命令

pmcmd 命令是 PowerCenter 提供的一个与 Integration Service 交互的命令接口。如果你想找到一个命令行接口启动、停止、监控 PowerCenter Workflow、Session 等，那非 pmcmd 莫属。

pmcmd 命令提供了大量的子命令，包括：aborttask、abortworkflow、connect、disconnect、exit、getrunningSessionsdetails、getservicedetails、getserviceproperties、getSessionstatistics、gettakdetails、getworkflowdetails、help、pingservice、recoverworkflow、scheduleworkflow、setfolder、setnowait、setwait、showsettings、starttask、startworkflow、stoptask、stopworkflow、unscheduleworkflow、unsetfolder、version、waittask、waitworkflow。

如果你想获得 Informatica PowerCenter Server 的安装版本，可以执行如下命令：

```
[infa@duhost1 9.5.1]$ export INFA_HOME=/home/infa/Informatica/9.5.1
[infa@duhost1 9.5.1]$ pmcmd version
Informatica(r) PMCMD, version [9.5.1 HotFix3], build [261.0906], LINUX 64-bit
Copyright (c) Informatica Corporation 1994 - 2013
All Rights Reserved.

Invoked at Sat Apr 11 10:58:34 2015
```

pmcmd 提供了两种模式：交互式模式和脚本模式。

如果你希望通过 pmcmd 调度 PowerCenter 的相关任务，请参考 PowerCenter 手册。

一个通过 pmcmd 命令启动 Workflow 的样例如下：

```
pmcmd startworkflow -sv MyIntService -d MyDomain -u seller3 -p jackson -f
SalesEast wf_SalesAvg
```

## 6.6 pmrep 命令

pmrep 是 PowerCenter 提供的一个与 Repository 交互的命令行接口。通过这个命令可以导入、导出 PowerCenter 的相关对象；执行版本控制相关功能；注册新的 plugin、创建或者更新数据库连接等。

pmrep 提供的子命令包括：addtodeploymentgroup、applylabel、assignpermission、backup、changeowner、checkin、cleanup、cleardeploymentgroup、connect、create、createconnection、createdeploymentgroup、createfolder、createlabel、delete、deleteconnection、deletedeploymentgroup、deletefolder、deletelabel、deleteobject、deploydeploymentgroup、deployfolder、executequery、exit、findcheckout、getconnectiondetails、generateabaprogramtofile、help、installabaprogram、killuserconnection、listconnections、listobjectdependencies、listobjects、listtablesbysess、listuserconnections、massupdate、modifyfolder、notify、objectexport、objectimport、purgeversion、register、registerplugin、restore、rollbackdeployment、run、showconnectioninfo、switchconnection、truncatelog、undochekout、unregister、unregisterplugin、updateconnection、updateemailaddr、updateseqgenvals、updatesrcprefix、updatestatistics、updatetargprefix、upgrade、uninstallabaprogram、validate、version。

与 pmcmd 一样，pmrep 也提供了两种操作模式：交互式模式和脚本模式。

一个通过 pmrep 命令备份 Repository 的样例如下：

```
pmrep connect -r Production -n Administrator -x Adminpwd -d MyDomain -h  
Machine -o 8080 - 连接 Repository  
pmrep backup -o c:\backup\Production_backup.rep -- 备份 Repository
```

如果你希望了解 pmrep 子命令的相关细节，请参考 PowerCenter 手册。

## 6.7 infasetup 命令

infasetup 命令是一个用于管理 PowerCenter Domain 和节点的命令行接口，当节点 IP 地址变更、保存 Domain 数据的数据库变更时，这个命令都可以发挥作用。

在每次安装 PowerCenter 后，在 INFA\_HOME 目录下都会有一个类似 `Informatica_9.5.1_Services_HotFix3.log` 命名的文件，里面包括不少关于 infasetup 命令使用的好样例。例如，如何定义一个新的 Domain，命令如下：

```
infasetup.sh defineDomain -du domain_user -dp ***** -dt oracle -dn Domain_duhost1 -cs jdbc: informatica: oracle://duhost1: 1521;ServiceName=orcl; MaxPooledStatements=20; CatalogOptions=0; BatchPerformanceWorkaround=true -nn node01_duhost1 -na duhost1: 6005 -ad Administrator -pd ***** -rf /home/infa/ Informatica/9.5.1/isp/bin/nodeoptions.xml -sp 6006 -ap 6008 -mi 6013 -ma 6113 -sv 6007 -asp 6009 -f -ld /home/infa/Informatica/9.5.1/isp/logs -bd /home/infa/ Informatica/9.5.1/server/infa_shared/Backup -tls false
```

infasetup 使用的次数并不是很多，一般的开发人员对这个命令往往会觉得生疏。这里用一个实际场景帮助大家了解 infasetup 的使用。假如 A 公司很早就在使用 PowerCenter，随着 PowerCenter 使用的场景逐渐增加，对 PowerCenter 可靠性的要求也在逐渐提高，尤其是保护 Repository 的可靠性。因此，他们决定将 Repository 使用的数据库更新为 Oracle RAC。这时除了更新 Repository 信息（这部分可以通过 Admin Console 更新），还要更新 Domain 使用的数据库（使用 infasetup 才能更新）。这时 A 公司的 PowerCenter 开发管理人员在讨论是否需要重新安装 PowerCenter，他们想到了 infasetup 命令。下面就是他们完成这个实际需求采取的步骤。

首先使用 `infaservice.sh shutdown` 命令关闭整个 Domain。

其次在所有的 gateway 节点运行下面的命令：

```
infasetup.sh updateGatewayNode -cs jdbc: informatica: oracle: //<dbhost1>; <dbport1>; ServiceName=<service1>; AlternateServers=(<dbhost2>; <dbport2>; <dbhost3>; <dbport3>; <dbhost4>; <dbport4>); ConnectionRetry=10;ConnectionDelay=10" -du <dbuser> -dp <dbpasswd> -dt ORACLE -dn <domain name> -nn <nodename> -na <node hostname>; <nodehost port>
```

最后使用 `infaservice.sh startup` 命令重新启动 Domain。

## 6.8 Mapping Architect for Visio

PowerCenter 开发的优势之一就是不需要懂开发语言，可以图形化拖曳实现快速开发。但若有大量的表要实现类似的 ETL 逻辑（如表对表的无转换抽取逻辑），是件非常痛苦的事情，需要成千上万次的鼠标单击，拖曳完成这些 Mapping。多次单击往往也会产生错误。

为了解决批量开发的问题，Informatica 提供了免费的 PowerCenter Mapping Architect for Visio 工具，利用 Visio 和 PowerCenter Designer 的接口可实现快速批量开发，大大减少了重复开发，也可以避免逐个开发产生的单击错误。开发过程非常简单，分为以下几步：

- ◎ 创建 Mapping 模板。
- ◎ 将模板导入 Mapping Architect for Visio，并参数化。
- ◎ 准备源和目标。
- ◎ 批量生成 Mapping/Session/Workflow。

在这个例子中，使用一对一的简单 Mapping 进行演示。

第一步：创建一个 Mapping，并导出为 XML 文件，如图 6-15 所示。



图 6-15

创建的 Mapping 名为 m\_EMP，并将其导出为 m\_EMP.xml 文件。

第二步：将模板导入 Mapping Architect for Visio，并参数化。

打开客户端 PowerCenter Mapping Architect for Visio，这个客户端和其他的 PowerCenter 客户端在同一个目录下。打开后，如图 6-16 所示，标记的部分是后续主要使用的功能按钮。

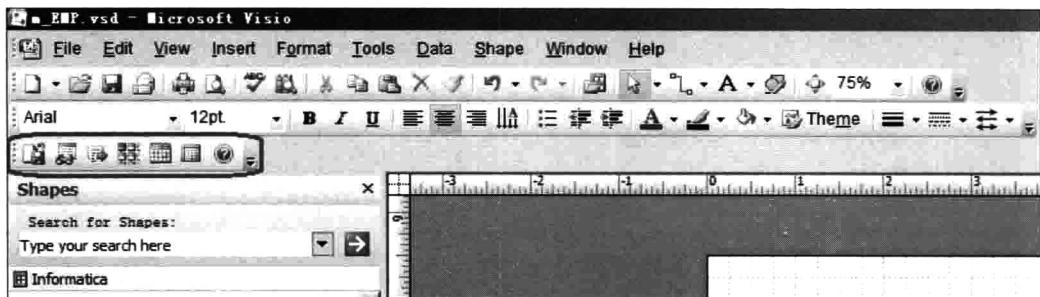


图 6-16

首先单击 Create Template from Mapping XML 按钮，选择上一步导出的 m\_EMP.xml 文件作为模板。导入后的 Mapping 模板如图 6-17 所示。

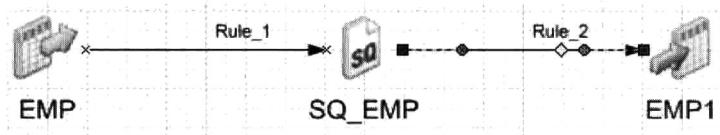


图 6-17

这时，需要对其中所有的对象进行参数化。双击 Source EMP，将 Transformation Name 和 Source Table 修改为\$SRC\$，如图 6-18 所示。

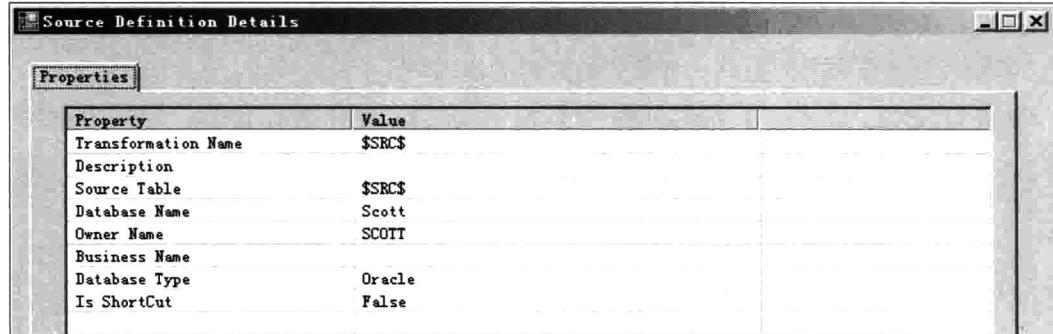


图 6-18

双击 Source Qualifier，将 Transformation Name 修改为 SQ\_\$SRC\$，如图 6-19 所示。

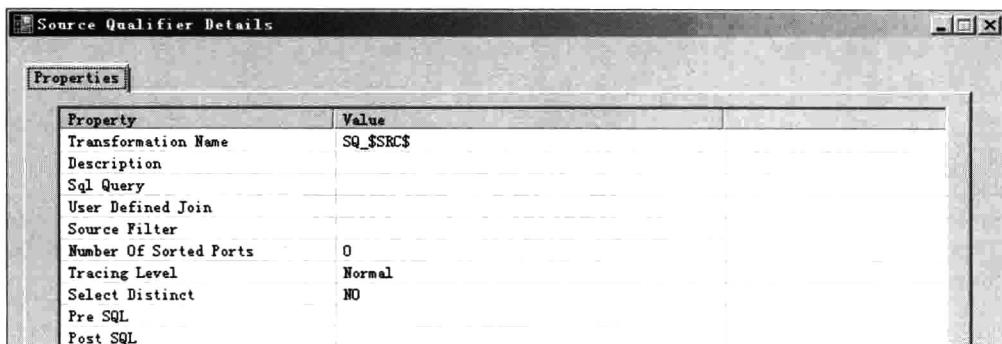


图 6-19

双击 Target，将 Transformation Name 和 Target 修改为\$SRC\$，如图 6-20 所示。

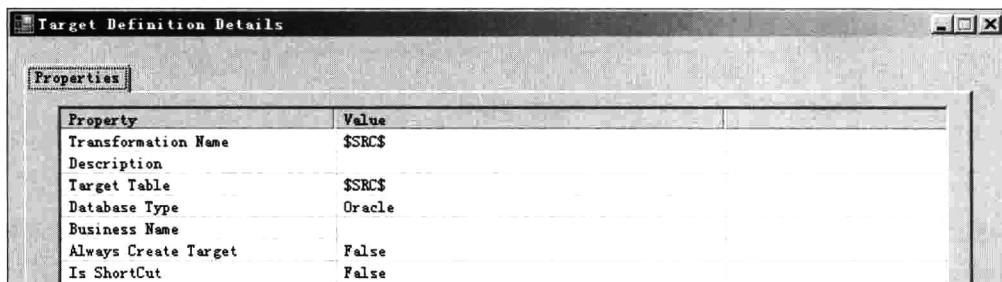


图 6-20

如果其中含有更多的组件，都需要做类似的修改。

双击 Rule\_1，删除默认的所有端口，使用 All Ports，如图 6-21 所示。

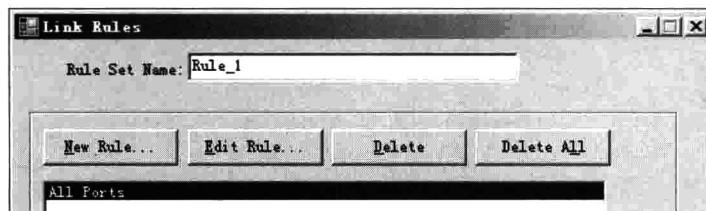


图 6-21

双击 Rule\_2，删除默认的所有端口，使用 All Ports，如图 6-22 所示。

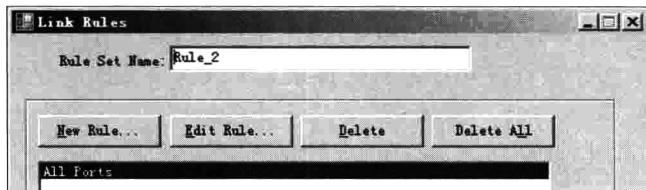


图 6-22

如图 6-23 所示, 用鼠标右键单击 Visio 空白处, 在弹出的快捷菜单中选择 Properties 命令, 弹出如图 6-24 所示的对话框, 将 Mapping 名参数化。

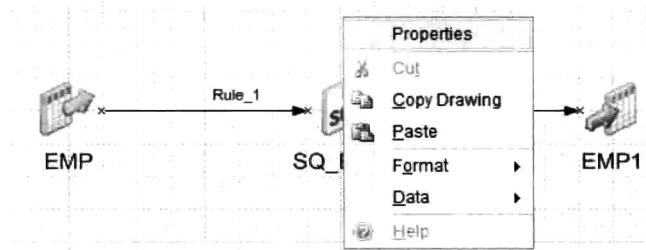


图 6-23

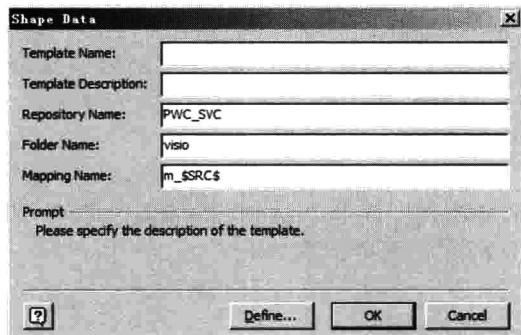


图 6-24

参数化完毕后的 Mapping Template 如图 6-25 所示。

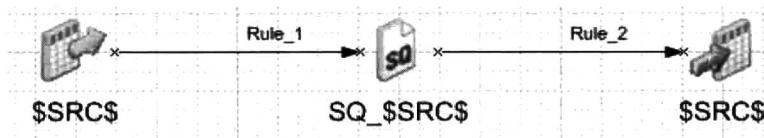


图 6-25

这时，单击 Validate Mapping Template 按钮检查修改是否有效。如果修改有效，即可单击 Publish Template 按钮进行发布。这时在目录下生成两个文件，分别是 m\_EMP.xml 和 m\_EMP\_param.xml。

m\_EMP\_param.xml 的默认信息如下：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE PARAMETERS SYSTEM "parameters.dtd">
<PARAMETERS REPOSITORY_NAME="PWC_SVC" REPOSITORY_VERSION="184" REPOSITORY_CODEPAGE="UTF-8" REPOSITORY_DATABASETYPE="Oracle">
  <MAPPING NAME="m_$SRC$" FOLDER_NAME="visio" DESCRIPTION="m_$SRC$">
    <PARAM NAME="$SRC$" VALUE="" />
  </MAPPING>
</PARAMETERS>
```

每一组<Mapping> 和</Mapping>为一个 Mapping。如需生成两个 Mapping，需要复制一个类似的<Mapping> 和</Mapping>对，并修改相关的值如下：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE PARAMETERS SYSTEM "parameters.dtd">
<PARAMETERS          REPOSITORY_NAME="PWC_SVC"          REPOSITORY_VERSION="184"
REPOSITORY_CODEPAGE="UTF-8" REPOSITORY_DATABASETYPE="Oracle">
  <MAPPING NAME="m_DEPT" FOLDER_NAME="visio" DESCRIPTION="m_DEPT">
    <PARAM NAME="$SRC$" VALUE="" />
  </MAPPING>
  <MAPPING NAME="m_BONUS" FOLDER_NAME="visio" DESCRIPTION="m_BONUS">
    <PARAM NAME="$SRC$" VALUE="" />
  </MAPPING>
</PARAMETERS>
```

第三步：准备源和目标。

准备源和目标比较简单，在 Designer 或者 Repository Manager 中选中要导出的对象，单击鼠标右键，在弹出的快捷菜单中选择 Export Objects 命令，如图 6-26 所示。源和目标都需要相同的操作，可以将导出文件命名为 Source.xml、Target.xml，并将其存放在目录..\\MappingTemplate\\tabledefs 下。



图 6-26

第四步：批量生成 Mapping/Session/Workflow。

这时，所有的原料准备完毕，可以批量生成 Mapping/Session/Workflow 了。在 PowerCenter Designer 中选择菜单 Mapping→Import Mapping Template 命令，如图 6-27 所示。

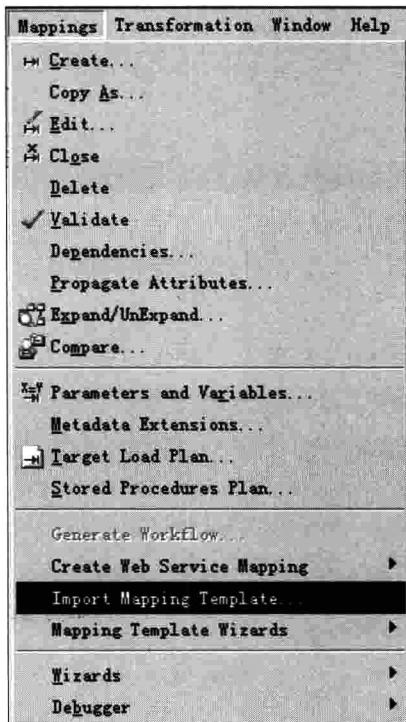


图 6-27

选择 Mapping 模板文件，如 m\_EMP.xml，如图 6-28 所示。

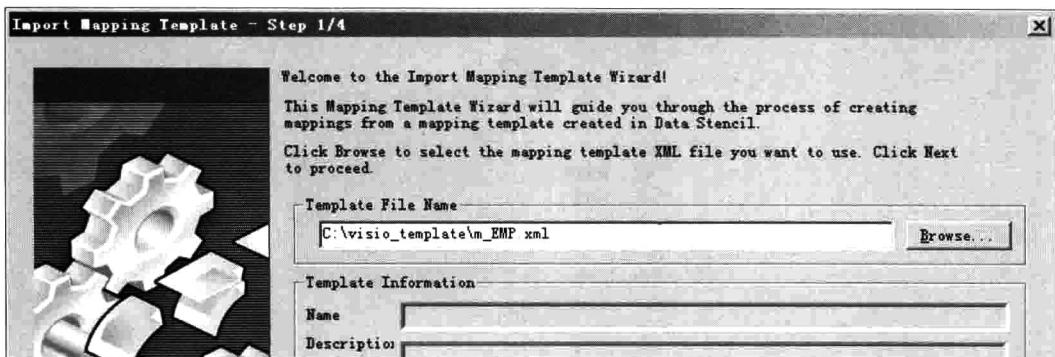


图 6-28

使用刚刚修改完成的 m\_EMP\_para.xml，并对参数进行赋值，如图 6-29 中标记的部分。

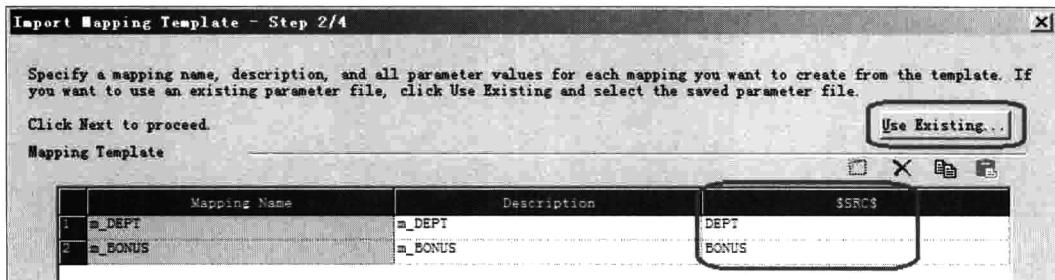


图 6-29

单击 Next 按钮，可以看到如图 6-30 所示的对话框，这里的选项一般采用默认设置。

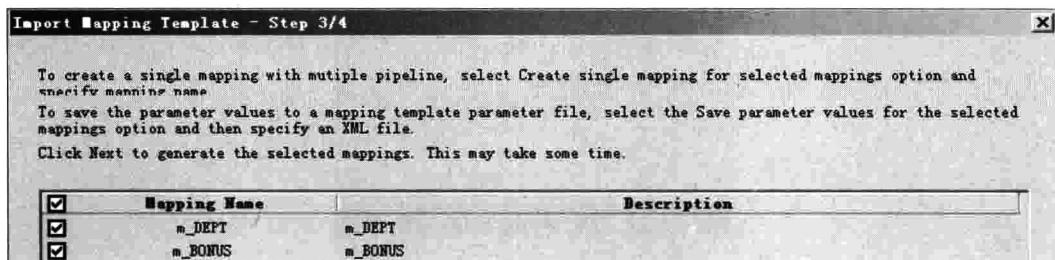


图 6-30

继续单击 Next 按钮，会看到 Mapping 创建成功的对话框，如图 6-31 所示。

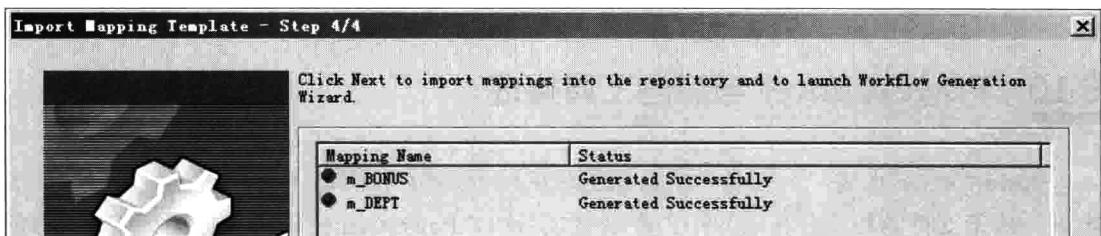


图 6-31

这时，可以选择是否继续创建 Session/Workflow。后续步骤比较简单，不再详述。

希望大家通过如上的介绍，可以掌握基本的 PowerCenter Architect for Visio。

## 6.9 MX View 语句

大家已经知道 PowerCenter 的 Repository 存放在数据库的一个 Schema (User) 下，它由一组表及表中的数据共同组成。笔者在过去的工作中，经常被问到是否有对 Repository 中表的说明文档，而且越详细越好。提出这些问题的开发人员有的是希望开发自己的监控报表，有的是为了能批量更新 Mapping 或者 Session 的某些属性。那么从哪里才能了解这些表的定义呢？

第一种方法：PowerCenter 的联机手册 *Repository Guide* 中有一个章节 *Using Metadata Exchange (MX) Views*，专门提供 PowerCenter 开放视图的解释。

第二种方法：网上有些专家已经将自己的经验汇总成了文档。但本书的篇幅有限，如果大家想获取的话，请发送邮件到 [PowerCenterbook@gmail.com](mailto:PowerCenterbook@gmail.com)。

第三种方法：以 Oracle 为例，查询 Oracle 系统表 v\$sql。这张表提供了实时的在 Oracle 中执行的 SQL 语句。例如，想知道存放数据源定义的 Repository 表有哪些，只要在 PowerCenter Designer 中增加一个新表定义，然后查询 v\$sql 表，就有机会从中筛选出所需的 SQL 语句及表名。提醒一下，v\$sql 是 Oracle 的系统表，需要以 DBA 用户登录才可以访问。

## 6.10 PowerCenter 与其他工具集成

PowerCenter 作为一个工具，最常见的应用场景是作为 ETL 工具应用在数据仓库、大数据、决策支持等数据分析项目中。在这些项目中，PowerCenter 作为数据搬运工，70%的工作量都是由它来执行的。大量的 ETL 都需要排程、调度、监控，因此经常需要与其他系统进行集成。PowerCenter 为与外部系统集成提供了 3 种方式。

- ◎ 命令行接口：pmcmd 命令及其子命令。
- ◎ Web Service 接口。
- ◎ API 集成。

(1) 命令行接口：即 pmcmd 命令及其子命令，它们提供了 PowerCenter 调度监控的基本功能。外部调度程序可以通过这些子命令启动 Workflow、中断 Workflow、获得运行日志等。笔者曾经有使用 pmcmd 与 Control-M 集成的经历，也曾经使用 pmcmd 命令用 Shell 包装的方式完成调度与监控。

(2) Web Service 接口：在前面的小节中曾经提到过 Web Service Hub。PowerCenter 的 Metadata Web Service 接口提供了与 pmcmd 几乎功能相同的接口，可以帮助客户快速构建自己的调度程序，支持自定义的监控等。著名的调度工具 Control-M 内置了调度 PowerCenter 的接口，它使用的也是 Web Service 接口。有的客户，如海尔，也通过 Web Service 接口构建了一个非常完备的调度监控平台。将调度与业务对象整合起来，将监控与业务对象整合起来。

PowerCenter 除了能够完成数据分析领域的 ETL 任务，还经常被用作 Data Hub。一些在 Data Hub 实施中比较有前瞻性的客户已经在提供 Data Provider 和 Data Consumer 的自助服务，他们也经常通过二次包装 Web Service 接口为 Data Consumer 提供自助服务的能力。

(3) Informatica Development Platform (IDP)：Informatica Development Platform 是 Informatica 为 PowerCenter 提供的二次开发平台，它提供了大量的 API 帮助用户对 PowerCenter 进行二次开发。与前两种接口相比，IDP 具有更大的灵活性，同时难度更高。IDP 提供的接口包括 Java 接口和 C 接口，以适应不同背景的客户的要求。

IDP 提供的接口包括如下几种。

- ◎ **PowerExchange API:** Java/C++ 接口。通过这些 API 可以为 PowerCenter 增加新的数据接口支持。如非常著名的 Greenplum，当年就是通过这个 API 实现了 PowerExchange for Greenplum，并自行维护，后来才将这个接口提供给 Informatica。目前微软、QlikView 分别为自己的一体机和报表工具提供了接口。
- ◎ **Transformation API:** Java/C 接口。用户可以通过 Transformation API 为 PowerCenter 增加新的 Transformation 或者增加新的功能。例如，一个客户的项目中需要一个独有的数学算法，这并非是 PowerCenter 内置的功能，那么它就可以通过 Transformation API 进行扩展。

**案例：**笔者曾经遇到一个项目，一家银行需要设计一个外包项目管理的流程。在这个流程中，他们将项目组分成两个团队；一个是银行内部开发人员；另一个是外包团队。这两个团队承担相似但是有区别的工作。在这个项目中，这两个团队使用两份相似但不同的数据副本。内部团队使用的全部是明文数据，外包团队使用的是关键信息加密的数据。银行研发中心对 PowerCenter 强大的分发和预处理能力非常感兴趣，虽然 PowerCenter 内置了加密算法，但是出于银行安全规则的要求，他们希望使用自己的加密／解密算法，并与密钥分发系统直接接口。这时候他们就考虑了 Transformation API。

- ◎ **Design API:** 接口 Java。假如你不喜欢 PowerCenter 提供的客户端工具，如 Designer、Workflow Manager，希望构建属于自己的新的开发界面，同时希望使用 PowerCenter 强大的引擎与接口能力；PowerCenter 提供了这方面的开放性，那就是 Design API，只要学习、掌握这些 API 你就可以做到。

很多年前，笔者曾看到国内的一家公司通过 Design API 设计出一款基于 Web 的 PowerCenter 设计界面。

- ◎ **Operation API:** Java/C 接口。Operation API 提供了外部程序与 Integration Service 交互的能力，尤其适合 PowerCenter 与第三方调度与监控的集成。Operation API 提供的主要功能如下。
  - ◆ 连接、访问 Integration Service。
  - ◆ 运行、管理 Workflow 和 Session 等。
  - ◆ 监控或者报告运行状况。
  - ◆ 查看日志。
  - ◆ 错误处理等。

举个例子，假如希望将 PowerCenter 与 HP OpenView 或者 IBM Tivoli 集成，Operation API 就是一个不错的选择。

Operation API 和 pmcmd、Web Service 接口在功能上有所重合，功能更加强大、灵活。

# 第7章

## 性能调优

---

性能调优是一个复杂的过程，需要全面的、综合的知识。读者需要了解的不仅仅是 PowerCenter 本身，很多时候会涉及源、目标数据库，并且还与操作系统、存储和网络相关。PowerCenter 的调优和其应用程序调优过程非常相似，是一个迭代的过程，无法一蹴而就。

调优还是个细致活，细节决定成败也是调优过程的真理。随着一个一个瓶颈的解决，性能问题才能被解决。用北京的道路拥堵问题解释性能问题，现象也很相似。

第一个现象是：一个四车道的快速路在某点突然变成三车道，这个位置往往在高峰期发生大范围拥堵。车速不是降为原来的  $3/4$ ，而可能仅仅是原来的  $1/5$ ，甚至  $1/10$ 。在计算机世界中也经常发生这种现象，当资源充足时，假如性能是 100，而一旦超出最高可用资源，发生资源争用，性能可能急剧下降，而非线性下降。

第二个现象是：一个很宽的路上有个新手，车速很低，后续的车辆速度也会随之下降，随着拥堵增加，道路资源被耗尽。这时后续车流的速度将远远低于那个新手的车速。

这两个现象说明调优中的一个魔鬼定律：为什么有的时候当一个很小的瓶颈被解决后，整体性能会有很大的提升？有时可能是个非常小的改动，如一个小的字符转换、删除某个字段等。

## 7.1 性能调优过程

前面已经提到过，性能调优是个迭代过程，包括建立性能基线、发现瓶颈、解决瓶颈、运行 Session 并与基线比较，如此迭代不止，直到满足性能需求，如图 7-1 所示。

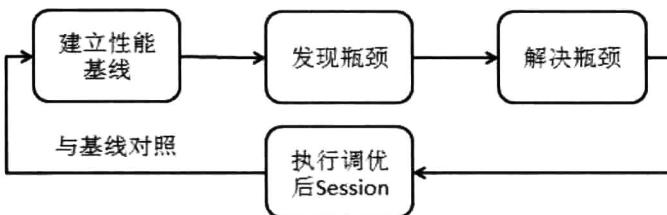


图 7-1

### 1. 建立基线

建立基线包括记录性能基线和保留基线 Mapping/Session。建立性能基线非常重要，这是对现有 Mapping/Session 的基础了解，也是在此基础上建立性能目标的依据。保留基线 Mapping 和 Session 也很重要，这是因为尽管在记录性能基线时记录了大量的基础数据，但是很难保证记录的全面性。尤其是在调优过程中，一个新的想法需要查看新的数据，这时有可能需要重复运行旧的 Mapping/Session。例如，前面曾经提到，可以将 Mapping 看成一条流水线，Transformation 可以看成流水线上的工人。尽管生产一批商品的时间相同，但随着工作流程的调整，各个工人的工作量却发生了变化。在记录基线时，基本不可能记录每个 Transformation 的具体数据，从而当发生这些变化时，无法了解调优的效果，这时也可能需要运行旧的 Mapping/Session 进行比较。

### 2. 发现瓶颈

性能调优中发现瓶颈就如同医生看病，有通用的过程，同时也依赖经验的积累。有经验的医生的优势就是依据历史经验可以快速地发现病因，并对症下药；普通的医生还是需要一步一步地望闻问切而发现病因所在。PowerCenter 的望闻问切过程按照这样的常规顺序进行分析：目标→源→Mapping→Session/Workflow→系统。

发现瓶颈的过程是聚焦关注点的过程，将当前要解决的问题聚焦，从而快速解决问题。在 PowerCenter 调优过程中，常见的瓶颈一般出现顺序是目标、源、Mapping，这是官方的

说法。笔者的经验是，瓶颈顺序一般是目标、Mapping 和源。

后续还会对如何发现瓶颈做更详细的介绍。

### 3. 解决瓶颈

解决瓶颈需要根据发现的问题对症下药。如果是目标写的问题，就需要对目标进行调优；如果是源读的问题，则需要对源进行调整。解决瓶颈问题有3个难点：

- ◎ 症状不等于原因。
- ◎ 对 PowerCenter 特性、原理的熟悉及灵活运用程度。
- ◎ 对外围系统的了解和熟悉程度，如对数据库、操作系统的熟悉程度。

比如目标数据库写慢，可能的原因可能有几种：数据库自身比较慢；数据写入过多；提交间隔太小；没有使用正确的写入方法；甚至是网络问题等。这时候可能需要一一尝试、逐个隔离，直到发现真实的原因并解决这个问题。

## 7.2 发现瓶颈

PowerCenter 发现瓶颈有个非常有效的工具（信息），这些信息藏在 Session Log 中，它有个专有的名字叫作 Thread Statistics。打开 Session Log，一直向下翻页，在接近 Session 的结尾处有一串关键的、关于性能的信息，如下：

```
Severity: INFO
Timestamp: 4/28/2015 8:07:01 AM
Node: node01_SDHOST1
Thread: MANAGER
Process ID: 4780
Message Code: PETL_24031
Message:
***** RUN INFO FOR TGT LOAD ORDER GROUP [1], CONCURRENT SET [1] *****
Thread [READER_1_1_1] created for [the read stage] of partition point
[SQ_EMPX] has completed.
Total Run Time = [2.903320] secs
Total Idle Time = [0.320313] secs
Busy Percentage = [88.967373]
```

```
Thread [TRANSF_1_1_1] created for [the transformation stage] of partition
point [SQ_EMPX] has completed.

    Total Run Time = [2.897461] secs
    Total Idle Time = [1.603516] secs
Busy Percentage = [44.657904]

Transformation-specific statistics for this thread were not accurate
enough to report.

Thread [WRITER_1_*_1] created for [the write stage] of partition point [EMPX1]
has completed.

    Total Run Time = [2.926758] secs
    Total Idle Time = [0.220703] secs
Busy Percentage = [92.459126]
```

这是一个非常简单的 Mapping 的运行统计结果，这个 Mapping 包括一个数据源、一个 Expression、一个 Filter 和一个数据目标。如前面介绍的 PowerCenter 的执行原理，一个简单的 Mapping/Session 执行时由 3 个线程执行，这 3 个线程就像这条数据处理流水线上的 3 个工人，它们分别是 Thread [READER\_1\_1\_1]、Thread [TRANSF\_1\_1\_1]、Thread [WRITER\_1\_\*\_1]，下面则是对这 3 个工人工作过程的统计。

- ◎ Total Run Time = [xxx] secs: 工作时间。
- ◎ Total Idle Time = [xxx] secs: 休息时间，这个值没有计入被操作系统 Blocked 的时间。
- ◎ Busy Percentage = [xxx] secs: 繁忙程度，它的计算公式是 $(\text{run time} - \text{idle time}) / \text{run time} \times 100$ 。

将这 3 个线程假设为流水线上的 3 个工人，瓶颈会在什么地方呢？结论就是：最忙的工人就是瓶颈所在。在上面的例子中，Busy Percentage = [92.459126]是最繁忙的线程，也是整个 Mapping/Session 的瓶颈。这就是说对这个样例 Mapping/Session 而言，性能瓶颈在于数据库写。

同理可以类推，如果 Thread [TRANSF\_1\_1\_1]的 Busy Percentage 最高，则转换过程是整个 Mapping 的瓶颈；如果 Thread [READER\_1\_1\_1]的 Busy Percentage 最高，则读是整个 Mapping 的瓶颈。

### 7.2.1 定位目标写瓶颈及调优

#### 1. 关系型数据库目标写

关系型数据库目标写是最常见的瓶颈。隔离目标写瓶颈有以下几个简单方法。

- ◎ 复制一个完全相同的 Session。
- ◎ 将数据库写修改为文件写，最好是分隔符分隔的文件，因为分隔符分隔的文件占用空间小，并将文件写在本地磁盘或者高速存储上。
- ◎ 在 Target 前增加一个 Filter Transformation，将 Filter 条件设为 False。

修改之后，如果性能有明显提升，那么就可以基本定位性能瓶颈在目标写。不过也有特例，当数据库是 MPP、Hadoop 或者其他可并行加载的数据库时，瓶颈也有可能与网络的性能有关。

如果定位为数据库写瓶颈后，可能的优化方法包括如下几种。

- ◎ 使用 Bulk Load：有些数据库支持 Bulk Load 比较好，比如 SQL Server 或者 Sybase ASE；有些数据库一般，比如 Oracle，有索引、有约束的情况下甚至不能使用；有的很差，比如 DB2，很可能会死锁，引起更多的麻烦。
- ◎ 使用外部 Loader：有些数据库提供了 Loader 接口，如 Oracle、Sybase IQ、Greenplum、Teradata 等。一般 MPP 数据库都会提供 Loader 接口，并且在加载数据时成为推荐的方式。
- ◎ 删除索引或者 Key 约束，加载结束后重建索引或者约束：速度会有比较明显的提升，但是很多时候大家还是不习惯采用。这种情况可以使用 Session 的 Pre-SQL 删 除或者 Disable 索引、约束；使用 Post-SQL 重建或者 Rebuild 索引、约束。
- ◎ 提升 Checkpoint、Commit 间隔：这也是比较常见且有效的方法，不过对数据库提出了一些新的要求。比如 Oracle，提升这些间隔有可能要求更多的回滚区或者 Redo 空间。
- ◎ 使用 PowerCenter Partition：PowerCenter Partition 提供了并行写能力，可以迅速地提高写性能，尤其对 Oracle、DB2 分区表性能提升更加明显。
- ◎ 优化目标数据库：这就要考验数据库功力了。不同的数据库有不同的要求，需要仔细研读相关数据库的文档。

## 2. 文本文件目标写

假如写文件时发现写目标的瓶颈问题，这时候问题就比较难处理了。提升文件写性能的手段并不多，常见的包括如下几种。

- ◎ 尽量减少写的数据量。
- ◎ 使用 PowerCenter Partition 的并行写能力：如果不选择 Merge 选项，性能提升还是比较明显的。但是如果选择 Sequential Merge，有时候性能还会下降；如果选择 Concurrent Merge，性能还是不错的；如果选择 File List，那性能一定是最优的。

### 7.2.2 定位源读瓶颈及调优

定位源瓶颈与定位目标瓶颈非常相似，需要将原有的 Mapping/Session 复制一份，尽量不要破坏原有的设计。在复制的 Mapping 中，在每一个 Source Qualifier 后增加一个 Filter Transformation，并将过滤条件设置为 false，这样就保证了后续的转换和写都不会发生，只是进行读。在这种情况下，如果读性能没有明显的加快，就说明数据存在读瓶颈。

#### 1. 关系型数据源读优化方式

- ◎ 优化查询语句，比如 Oracle，可以尽量优化 SQL 语句的查询计划；或者使用 hint，加快索引读或者全表读。
- ◎ 使用 PowerCenter 分区。如果数据库是 Oracle 或者 DB2，并且数据库是分区的，可以利用 PowerCenter 分区特性进行并行读数据。
- ◎ 使用 IPC 协议。如果是 Oracle 数据库，并且 PowerCenter 和数据库服务器安装在同一台机器上，可以尝试使用 IPC 协议进行读取。
- ◎ 使用专门的读接口，如 Teradata。当使用 fastexport 时，性能明显优于 ODBC。

#### 2. 文本数据源优化方式

文本文件读的优化方式并不多，比较常用的有下面几种。

- ◎ 使用 PowerCenter 分区并行读。当使用 PowerCenter 分区读文件时，PowerCenter 会自动启动多个线程尽量均衡地读取相同数据量的数据，从而加快性能（具体的配置请参考介绍分区的章节）。
- ◎ 调整 Line Sequential Buffer。尽量保证 Line Sequential Buffer 能够容纳 4~8 行的数据。当文件记录比较长时，需要关注并调整这个参数。

### 7.2.3 定位 Mapping/Session 瓶颈

如果已经消除了源读和目标写的瓶颈，这时需要定位 Mapping/Session 瓶颈。当消除了源读和目标写瓶颈后，在每个目标写前加一个 Filter Transformation。这时运行对应的 Session，如果性能无明显的变化，则主要的瓶颈就在于 Mapping/Session。

Mapping 的调优类似程序调优，本身非常复杂。这里简单地介绍 PowerCenter 提供的一个定位 Mapping 瓶颈的方法。在 Session 的 Properties Tab 中有一个 Collect performance data 选项，如图 7-2 所示。

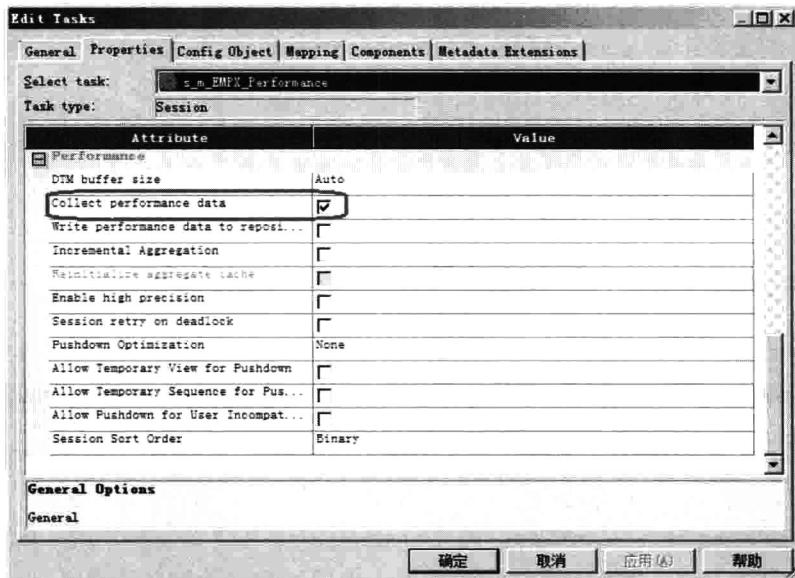


图 7-2

当打开这个选项后，PowerCenter 会自动收集更详细的性能数据，这些数据叫作 Performance Counter。有两个地方可以查看 Performance Counter。

(1) 在 Workflow Monitor 中，当打开 Collect performance data 选项后，在 Workflow Monitor 中可以看到更多的性能信息，这些信息出现在 Workflow Monitor 的最下端，如图 7-3 所示。

Performance Counter	Counter Value
EXPTRANS	node01_SDUMOST1 1000019
Expression_inputrows	1000019
Expression_outputrows	1000019
FILTRANS	node01_SDUMOST1 1000019
Filter_inputrows	1000019
Filter_outputrows	1000019

图 7-3

(2) 当打开这个开关后，在 Session Log 的目录下会生成一个与 Session Log 同名，但是扩展名不同的文件，如<s\_m\_EMPX\_Performance>.perf。名字为\*.perf。文件的内容如下：

Monitor Counter Information At Tue Apr 28 14:38:31 2015

Transformation Name	Counter Name	Counter Value
EXPTRANS	Expression_inputrows	1000019
	Expression_outputrows	1000019
FILTRANS	Filter_inputrows	1000019
	Filter_outputrows	1000019

这两个方法的不同之处在于，在 Workflow Monitor 中可以实时地看到每个 Counter 的变化；而在\*.perf 文件中，则可以方便地查询。下面介绍几个常用的 Performance Counter。

- ◎ **Rowsinlookupcache**: 当这个值比较高时，可能有 Lookup Transformation 的性能问题。
- ◎ <Transformation>\_errorrows: 当 Transformation 计算发生错误时，PowerCenter 性能会急剧下降，因此一旦这个值非零，需要尽快消除这些错误。
- ◎ **BufferInput\_efficiency** 和 **BufferOutput\_efficiency**: 当这两个值比 Sources 或者 Targets 值高时，提高 DTM buffer pool size 有可能一定程度地提高 PowerCenter 性能。
- ◎ <Transformation>\_readfromdisk 和 <Transformation>\_writetodisk: 这两个 Counter 对 Aggregator、Rank 和 Joiner 3 个 Transformation 有效。当这些值非零时，提升 Data Cache 和 Index Cache 的大小有利于迅速提升此 Mapping 的性能。有个例外的场景是 Incremental Aggregation，它使用文件系统保存历史数据，所以即使 Aggregator\_readtodisk 和 Aggregator\_writetodisk 为非零值，也不一定是性能问题。

### 7.2.4 定位系统瓶颈

系统瓶颈也是经常发生的，因为 PowerCenter 开发、管理人员一般都不是系统管理员，这里仅仅给出一些提示。本小节介绍一下 PowerCenter 在调优过程中经常遇到的系统问题。

- ◎ I/O 瓶颈：ETL 是 I/O 密集的操作，因此经常出现的问题是 I/O 瓶颈。
- ◎ 内存瓶颈：内存存在某些程度上与 I/O 的操作行为相似，可以认为是 I/O 的 Cache，尤其是在 Joiner、Lookup、Aggregation 等需要 Cache 的场景中，大的内存可以适当地缓解 I/O 压力。因此，在最近的项目中，一般建议的硬件配置为 CPU Cores : Memory 为 1:4 或者 1:6。
- ◎ 网络瓶颈：这是在 MPP Database 或者 Hadoop 环境中的新烦恼，这些场景普遍是分布式存储、分布式计算，I/O 瓶颈不再是主要问题，这时候主要的瓶颈可能是网络问题。

## 7.3 Mapping 调优

Mapping 调优是 PowerCenter 调优最复杂、最细节的部分，既包括整体 Mapping 的设计，也包括具体 Transformation 的调整。整体 Mapping 的设计本质上是一个开发人员编写程序的能力，这里很难做全面的描述，仅仅在需要时做些阐述，因此这节重点关注在 Mapping 中更细节的调优能力，会细化到 Transformation 的级别，甚至 Transformation 属性的级别。

### 7.3.1 Transformation 优化

#### 1. 优化 Lookup Transformation

第一招：Cache Enable。Lookup Cache Enable 性能优于 Cache Disable。Cache Enable 的执行过程是将被 Lookup 表的所有需要的列都装载到 PowerCenter 中，构建索引，然后再进行 Lookup 计算。Cache Disable 需要一条一条地将数据提交到数据库中进行 SQL 查询，因此 Cache Enable 的性能远远高于 Cache Disable。当被 Lookup 的对象是 Flat File 时，Cache Enable 强制选择。

第二招：Cache 设置要足够大。如果设置的 Cache 足够大到可以容纳所有 Data 和

PowerCenter 自建的 Index，所有的 Lookup 将在内存中计算，性能也会有质的提升。如果 Cache 比较小，将产生 I/O 交换，相比全内存的 Lookup，性能有巨大差别。可以设置 Cache 的组件都可以考虑这个建议，如 Joiner、Aggregator、Rank 等。

第三招：尽可能少使用被 Lookup 表的列。这是因为如果使用的列多，必然需要更多的内存。例如，Lookup 表和被 Lookup 表都包括 Dept\_Name 列，并且 Dept\_Name 列将在后续的 Transformation 中被使用，这时应该尽量使用 Lookup 表的 Dept\_Name 列，而非被 Lookup 表的 Dept\_Name 列。

第四招：使用 Unconnect Lookup 可以提升性能。如果不是所有行都需要 Lookup，尽量使用 Unconnect Lookup，因为与 Connect Lookup 相比，Unconnect Lookup 的查询次数比较少。

第五招：如果有多个 Lookup 条件，将等值比较排在前面会提升性能。

第六招：如果 Lookup 可以替代 Joiner，尽量使用 Lookup。一般情况下 Lookup 可以比 Joiner 提升 10%~20% 的性能。

第七招：共享 Lookup Cache。

- ◎ 同一个 Mapping 中多个相同的 Lookup：PowerCenter 会自动地共享它们的 Lookup Cache。
- ◎ 使用同一个 Mapping 的多个 Session：使用 Persistent Cache 将帮助它们共享 Cache。Persistent Cache 是 Lookup Transformation 的 Properties Tab 的一个选项。
- ◎ 不同的 Mapping 中使用相同的 Lookup：这时候需要使用 Named Persistent Cache。图 7-4 可以帮助大家理解如何配置。

勾选 Lookup cache persistent 复选框，并且为 Cache File Name Prefix 添加一个名字前缀。

第八招：定制 SQL Override 减少被读入 Lookup 中的数据。定制 SQL Override 可使 Lookup 中的数据尽量少，有利于提升 Lookup 的性能。

## 2. 优化 Filter 和 Router Transformation

第一招：尽量将 Filter 前置。在 Mapping 中尽量将 Filter 放到 Mapping 中靠前的位置，这样可以减少后续组件处理的数据量，从而提升整个 Mapping 的性能。

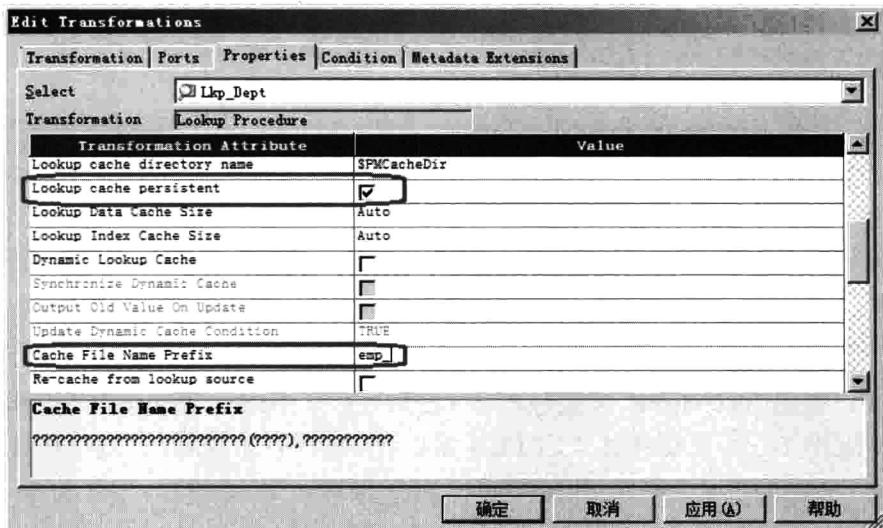


图 7-4

第二招：在 Filter 条件中，尽量避免使用复杂表达式，Integer 和 True/False 效果比较好。

第三招：如果数据确实没有必要传递到目标，使用 Filter 或 Router 将其过滤掉。

第四招：使用 Router 替换多个 Filter 也可以提升性能。如果在 Mapping 中使用多个 Filter 对数据进行分流，替换成 Router 效果可能更好。还记得生成缓慢变化维的模板吗？这个就可以替换成一个 Router。

### 3. 优化 Aggregator Transformation

第一招：使用简单列作为 Group By 列。例如，使用 Integer 类型的列进行 Group By 比使用 String、Date 类型的列进行 Group By 性能更好；使用长度为 5 的 String 类型列比长度为 10 的性能更好。

第二招：使用 Sorted Input 选项可以提升性能。在 Aggregator Properties Tab 中勾选 Sorted Input 复选框后，Aggregator 假设所有的数据是排序的，当看到不同的 Group By 值时，它就可以将上一组 Group By 计算所得的结果数据向下游传递。而未使用 Sorted Input 时，Aggregator 必须等待所有的数据到达后才有可能完成计算。使用 Sorted Input 需要在 Aggregator 前增加 Sorter Transformation，并使用相同的 Group By 列，或者在 Source Qualifier

中对数据进行排序。不过 Sorter 组件也会消耗一定的时间，也曾经遇到过在 Partition 的情况下 Sorted Input Aggregation 性能更差。

**第三招：使用 Expression 和 Update Strategy 替代 Aggregator。**使用这个方法的 Mapping 如 Source→Sorter→Expression→Update Strategy→Target。首先使用 Sorter 对数据进行排序，然后在 Expression 中使用 Local Variable 对同组数据进行累加，最后使用 Update Strategy 将每组第一条数据执行插入操作，其他执行 Update 操作。这样就模拟了一个 Sorted Input 的 Aggregator。这个方法用得并不多，主要问题是未来可能会看不懂这个 Mapping 的功能，这样的 Mapping 不直观。

**第四招：使用 Incremental Aggregation（增量聚合）。**例如，数据仓库每天都可以获得前一天的增量数据，并需要将前一天的数据累加到前面所有的数据上，这种场景比较适合 Incremental Aggregation。启动 Incremental Aggregation 比较简单，当 Mapping 包含 Aggregation 组件时，只需要在 Session 的 Properties Tab 中勾选 Incremental Aggregation 复选框即可，如图 7-5 所示。

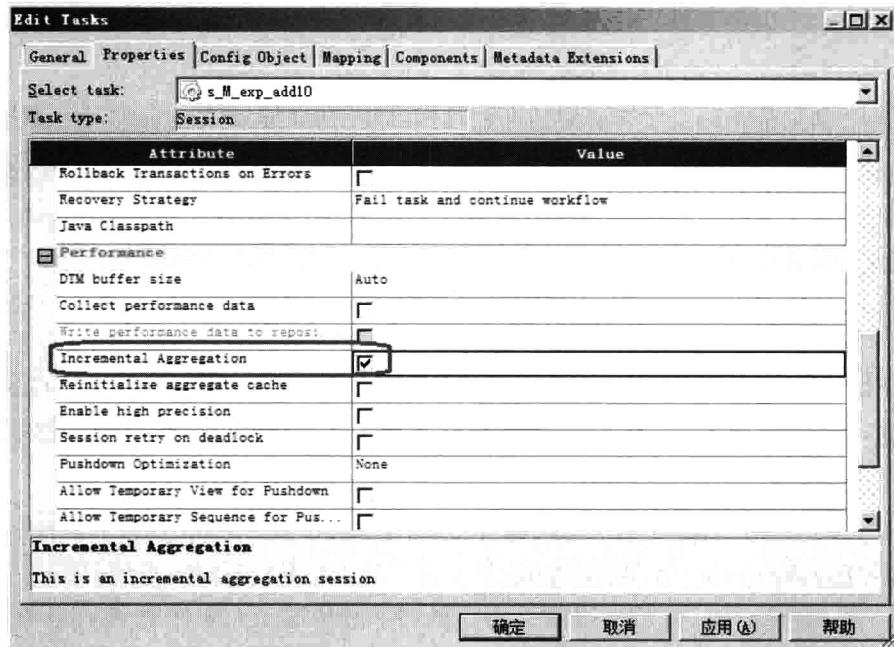


图 7-5

#### 4. 优化 Joiner Transformation

第一招：使用数据库的 Join 语句完成所需要的计算。

第二招：Master 选择尽量小的表。这种方法在介绍 Joiner 时曾经提到过。

第三招：使用 Sorted Input。和 Aggregator 一样，Joiner Transformation 也有 Sorted Input 选项，这时候需要确保 Master 和 Detail 的关联条件都是有序的，可以在 Joiner 前面增加 Sorter Transformation 或者在 Source Qualifier 中用 SQL 语句对数据进行排序。Joiner 在使用 Sorted Join 与 Unsorted Join 时使用了不同的 Join 算法。

第四招：尽量使用 Normal Join。

#### 5. 优化 Sequence Generator

每行数据到达时，Sequence Generator 需要决定 Next Value 的值。默认情况下 Sequence Generator 需要查询 Repository 以获得需要的值，这种情况必然降低 PowerCenter 的执行效率。PowerCenter 提供了一个选项来缓解这个问题，这个选项是 Number of Cached Values，默认值是 0，建议值为 1000 或更大，如图 7-6 所示。

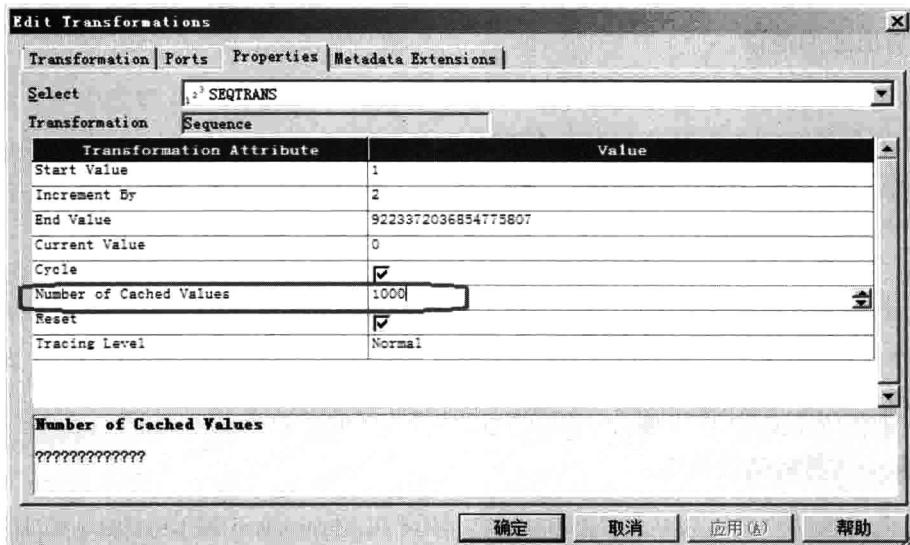


图 7-6

## 6. 避免调用 External Procedure Transformation

在调用 External Procedure 时一般都会大幅度地降低 PowerCenter 性能，因此从性能的角度考虑，建议尽量避免调用外部过程。这些外部过程包括 Stored Procedures、External Procedures 和 Advanced External Procedures。

但是，有的情况下 PowerCenter 实现比较复杂，则需要调用外部过程以简化开发过程，提高可读性。以 Stored Procedure 为例，需要尽量避免的是每行数据都去调用一次 Stored Procedure，这样会极大地降低性能。建议的方法是，尽量将其作为一个批处理一次性执行，或者尽量减少调用的次数。

### 7.3.2 列级别的优化

列级别的调优是性能调优的终极手段，复杂的表达式可能会影响性能的 10%~20%，也就是说复杂表达式的调优空间为 10%~20%。同时也需要考虑道路拥堵逻辑（四车道变三车道，在拥堵的情况下，通行量不是 75%），量变同样会引起质变。因此，不要放弃任何的调优手段，尤其是列级别的优化。列级别调优仍然有很多可调整部分。

#### 1. 析出通用逻辑

析出通用逻辑指的是找出 Mapping 中的通用逻辑，尽量减少其计算的次数。例如，计算银行卡的卡号，由于发卡行不同，有若干的逻辑计算，包括 IIF(Substr(Credit\_card,0,4),xxxxxxxx)、IIF(Substr(Credit\_card,0,4),yyyyyyy)、IIF(Substr(Credit\_card,0,4),yyyyyyy)。这时对性能最有利的方法是，定义一个变量 Var=Substr(Credit\_card,0,4)，在其他的表达式中直接引用这个变量。假如有 10 亿条数据，这样做就可以减少 10 亿次的重复运算。再举一个非表达式的例子：假如某个 Mapping 中有若干个业务逻辑都需要通过身份证号码查询个人信息，这时，对性能比较有利的方法是将这个 Lookup 尽量提前到 Mapping 的上游，这样可以减少重复创建 Lookup Cache 的次数，从而提升整体的性能。

#### 2. 最小次数的函数调用

函数调用也是一个比较耗费资源的计算，因此在 Mapping 中减少函数调用的次数也是一种重要的调优手段。

例如， $\text{SUM}(\text{Column A}) + \text{SUM}(\text{Column B})$  可以优化为  $\text{SUM}(\text{Column A} + \text{Column B})$ 。

又如，`CONCAT(CONCAT(FIRST_NAME,),LAST_NAME)`可以优化为 `FIRST_NAME || LAST_NAME`。

使用操作符比使用函数的性能更好。

### 3. 数值类型与字符串

在某些情况下，有些字段可以是字符类型，也可以是数值类型，那么数值类型的计算性能会优于字符串，尤其是 Joiner 中的关联字段或者 Lookup 中的 Lookup 条件。例如，`EMPLOYEE_ID` 为字符型，但是实际上存放的全部为数值型数据，在 Lookup 或者 Join 时，额外做一次类型转换的代价要小于 Lookup 或者 Join 关联性能提升带来的收益。如果 Mapping 有性能调优的需求，这也是可以考虑的一种手段。

### 4. CHAR 和 VARCHAR 类型的数据比较

相信大家都知道 CHAR 类型的数据是保留末尾空格的，而 VARCHAR 类型的数据是不保留末尾空格的。例如，同样在 CHAR(10) 和 VARCHAR(10) 字段中插入“DSS”，在 CHAR(10) 字段中，它的数据长度为 10；在 VARCHAR(10) 字段中，它的数据长度为 3。

在 PowerCenter 的 Integration Service 中有一个选项叫作 TreatCHARAsCHAROnRead (Treat CHAR as CHAR on Read)，当设置 TreatCHARAsCHAROnRead 为 no 时，PowerCenter 将 CHAR 字段后的空格自动删除。如果需要将 CHAR 字段自动删除末尾空格，可以选择使用这个选项。这时进行 CHAR 和 VARCHAR 的比较，性能会有所提升。

#### → 注释

笔者在使用 PowerCenter 9.6.1 及 Oracle 数据库进行测试时，这个选项似乎不起作用。

### 5. 减少 Mapping 中的 Transformation

每个 Transformation 都会消耗一定的资源，在 Mapping 中尽量减少或者合并不必要的 Transformation，这样可以在一定程度上提升性能。但这不是绝对的，在有的场景下，开发人员也可以通过增加 Transformation 提高性能，尤其是在增加新的 Partition Point 的情况下。大家谨慎使用这个建议。

举一个比较形象的例子说明这个问题：假如你是老板，增加一个 Transformation 就像雇用一个新工人，代价就是付出的工资。同时这个工人努力劳动为老板赚钱，如果这个工人为老板赚的钱大于老板付出的工资，性能就得以提升，否则则反之。所以加不加工人，应该是你说了算。

### 7.3.3 其他方面的优化

#### 1. 单源读 Mapping/Session

假如在一个项目中多个 Mapping/Session 都使用同一个数据源进行计算，一个简单的调优方法是尽量将这几个 Mapping 整合为同一个 Mapping，这样对该数据源只需读一次就可以支持后续的计算过程，从而提高整体的处理性能。

同样，假如一个 Mapping 中的两个 Pipeline 都需要对某字符串进行相同的 substr()运算，一个可以提升性能的方法是尽量将 substr()运算提前到 Pipeline 分拆之前，这样做是为了避免重复计算，从而提升 Mapping 的整体性能。

#### 2. SQL Override 优化

在 Source Qualifier、Lookup 和 Target 中都可以使用 SQL Override，在这里有必要对 SQL Override 中使用的 SQL 进行优化。这部分内容比较复杂，涉及 SQL 语句调优问题，后面用单独的一节进行描述，因此不在这里进行详述。

#### 3. 避免不必要的数据类型转换

PowerCenter 自动对兼容的数据类型进行转换，这个转换过程是比较耗费资源的。例如，一个 Integer 类型的数据被转换为 Decimal，最后又转换为 Integer。

但是存在某些例外，例如，一个本来是 Decimal 类型的列存放的数据事实上全部为 Integer，并且需要对这列进行 Lookup，这时将 Decimal 转换为 Integer 将极大地提升性能。

#### 4. 消除 Transformation Error

前面曾经提到，如果存在 Transformation Error，对性能的影响将非常大。一定要消除所有的 Transformation Error。

什么是 Transformation Error 呢？例如在某个 Expression 组件中使用表达式 to\_date(mydate, 'YYYYMMDD')，假如 mydate 列的很多值并不能转换为日期，如 mydate='This is String'，PowerCenter 将报错，并将错误信息写入 Session Log，这就是所谓的 Transformation Error。

## 7.4 Session 调优

首先谈一下调优顺序问题。正确的、理想的调优顺序是，在进行 Session 调优之前，首先应该完成目标写、源读和 Mapping 的调优。否则，有可能给小脚穿上大鞋，单个 Session 的性能可能得到了提升，但是整个系统的性能可能会有所下降，资源被极大地浪费。

### 7.4.1 内存调优

PowerCenter 的运行是以 Session 为单位的，Session 调优的一个重点是进行内存调优。在 Session 中有两组可以调整的内存值，分别是：

- ◎ 设置 DTM Buffer 相关值。
- ◎ 设置 Transformation Cache。

这两部分内容已经在 PowerCenter 内存管理部分做过详细的介绍，因此不在这里继续叙述。

但大家需要记住的是，内存调整是性能效果明显的设置，进行调优时一定不能忽略内存调优。

### 7.4.2 PowerCenter 高级特性支持高性能

在性能提升方面，PowerCenter 还提供了一些高级特性，帮助用户合理地利用系统资源，从而提升系统的整体性能。这些高级特性包括：

- ◎ PowerCenter Partition。
- ◎ Pushdown 优化。
- ◎ 使用 PowerCenter Grid。
- ◎ 并行执行 Workflow 和 Session。

#### 1. PowerCenter Partition（并行）

PowerCenter Partition 是 PowerCenter 提供的一种 Session 级别的并行处理能力。Partition 提供了两种能力：一种方法是增加流水线，可以简单地从一条流水线扩展为两条、三条（多

个并行)；另一种方法是在流水线上增加工人(增加 Partition Point)。

此外 Partition 还提供了不同的 Partition Type，以适应不同的计算逻辑(详细内容请参考 Partition 所在章节)。

## 2. Pushdown 优化

Pushdown 优化是 PowerCenter 提供的一种充分利用数据仓库、大数据生态资源的能力。PowerCenter 可以将其图形化 Mapping 部分或全部转换为数据仓库或者源系统可以运行的 SQL 语句，从而利用数据库资源，释放 ETL 服务器负担，减少数据流转，提高数据处理的整体性能。

Pushdown 有 3 种基本类型：Source Side Pushdown、Target Side Pushdown 和 Full Pushdown(详细内容请参考 Pushdown 所在章节)。

## 3. 使用 PowerCenter Grid

PowerCenter Grid 提供了 ETL 服务器的平行扩展能力，可以在基本不修改 Mapping/Session 的设计情况下，利用多个 ETL 服务器资源以达到最佳的处理能力。PowerCenter 提供的 Grid 包括两种类型：Workflow on Grid 和 Session on Grid。

- ◎ Workflow on Grid：可以将一个 Workflow 中的 Session 根据集群中各个 ETL 服务器的繁忙程度，动态地分配其执行的任务，从而达到最优的处理能力。
- ◎ Session on Grid：Session on Grid 需要与 Partition 配合使用，当一个 Session 是 Partition 的情况下，可以将不同的 Partition 的任务分配到不同的 ETL 服务器上，从而提供 PowerCenter 处理单个超大任务的能力，并提升整体性能(详细内容请参考 Grid 所在章节)。

### 7.4.3 其他手段

除了上面提到的性能调优方法，在 Session 调优方面还有以下几个关注点，它们对 Session 的性能也有一定的影响，在某些情况下影响还比较大。这些手段包括：提升 Target Commit Interval、Disable 高精度、降低 Session 的错误跟踪( Error Tracking )。

- ◎ 提升 Target Commit Interval：提升 PowerCenter Commit Interval 可以减小向数据库提交的次数，从而提高数据库写入的性能。Commit Interval 的缩小必然导致单个

事务数据量的增加，这对回滚段的大小提出了更高的要求。PowerCenter Commit Interval 的默认值是 1000，这是为了与默认回滚段的大小相匹配。一般生产数据库对回滚段的大小都有所调整，因此可适当提高默认值的大小。

- ◎ Disable 高精度：Decimal 类型最大的长度是 28。当 Enable 高精度后，Decimal 将使用最大 28 的长度。但这在一定程度上会影响 PowerCenter 的执行效率。
- ◎ 降低 Session 的错误跟踪（Error Tracking）：当 Session 错误发生时，有时为了跟踪错误的目的，需要在 Session 的 Config Object Tab 中将 Override Tracing 设置为 verbose initialization 或者 verbose data。这种情况下 PowerCenter 的执行效率会受到一定的影响。因此，一般的建议是，当问题解决后应将 Override Tracing 重置为默认值。在 Session 的 Config Object Tab 中的修改将影响所有 Transformation 的 Tracing Level。另外一个建议是，跟踪错误尽量修改 Transformation 上的 Tracing Level，而不是整个 Session 的 Tracing Level。

#### → 注释

Commit Interval 和 Disable 高精度都是 Session Level 的属性，可以在任意 Session 的 Properties Tab 中进行设置，如图 7-7 所示。

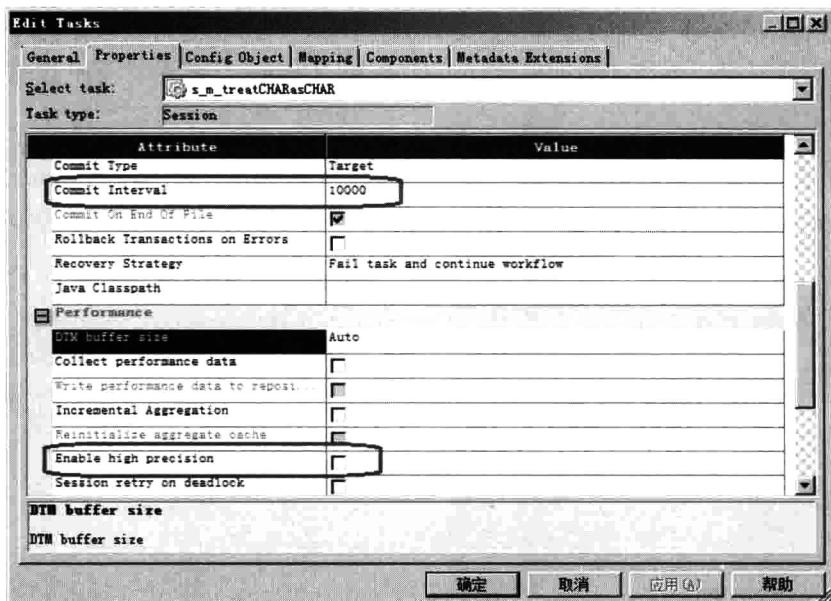


图 7-7

## 7.5 SQL Override 调优

在 PowerCenter 的 SQL Qualifier、Lookup、Target 和 SQL Transformation 中都会直接使用数据库的 SQL 语句，这些语句将被直接提交给数据库执行。这些语句是否优化对 PowerCenter 的执行效率有非常大的影响，因此在 PowerCenter 调优过程中需要对这些语句进行调优。本节并非介绍 SQL 语句怎么调优，而是要强调几个关注点：

- ◎ SQL Override 或者 Source Qualifier 中 SQL 性能对 PowerCenter 性能影响很大。
- ◎ 这些 SQL 语句是在对应数据库连接串的数据库中执行的。
- ◎ SQL 的调优需要对对应的数据库比较熟悉。

因为涉及的数据库非常多，在这里无法一一介绍所有数据库 SQL 语句的调优方法。任意一个数据库的 SQL 调优都可以单独成为一本书。

笔者曾经被问到：PowerCenter 的 SQL 是否支持 Oracle Hint?答案是肯定的。“如果对应连接串指向的是 Oracle 数据库，Hint 是完全没有问题的。”

此外，PowerCenter 运行的宿主是操作系统，PowerCenter 运行的数据源和目标一般都是数据库，因此对操作系统和数据库调优也是 PowerCenter 调优的重要组成部分。希望大家同样理解这些调优的重要性，不要因为笔者用的篇幅少就忽略这部分内容。

延伸一句：假如正在使用的是 PowerCenter on Hadoop，这时 PowerCenter 运行的引擎就是 Hadoop。因此在使用 PowerCenter on Hadoop 时，Hadoop 调优也会成为一个课题。

# 第8章

# PowerCenter Troubleshooting

---

Troubleshooting 是一个简单同时又是最复杂的工作。把这部分内容当成单独的一章来写似乎有点小题大做，但是，笔者认为这部分工作非常重要，它直接影响开发人员的开发效率，影响遇到问题的解决速度。提升 Troubleshooting 能力有 4 件事情要做：

- ◎ 掌握 PowerCenter 原理。
- ◎ 知道日志在哪儿，并查找日志。
- ◎ 查询 Information Knowledge Base。
- ◎ 联系 Informatica GCS (Global Customer Service)。

## 8.1 安装、启动过程的错误

当运行 `install.sh` 安装 PowerCenter 或者运行 `Infaservice.sh start` 启动 PowerCenter 服务器的时候，经常会遇到各种各样的错误。这两个命令最重要的作用一个是创建 PowerCenter Domain，一个是启动 PowerCenter Domain。这两个命令都是 Java Based 的程序。需要关注的日志包括如下几个。

- ◎ 安装过程：执行 `install.sh` 时，如果安装过程中失败，有两个日志一定要看：`Informatica_9.6.1_HotFix_1_Services_InstallLog.log` 和 `Informatica_9.6.1_Services_HotFix1.log`。`9.6.1` 和 `HotFix1` 是版本信息。从中找到相关的错误提示非常重要，

典型的问题包括：端口被占用、数据库无法连接（注：Domian 通过 JDBC 连接数据库）、错误的 Java 环境变量或者 JVM 版本。

- ◎ PowerCenter 启动过程：当修改了环境变量等某些配置时，PowerCenter 服务器需要重新启动，需要运行 Infaservice.sh startup 命令。假如这时发生错误，PowerCenter 服务器没有启动成功，首先要看的是目录\$INFA\_HOME/tomcat/logs 下的日志。这是因为 PowerCenter 的 Admin Console 是基于 Tomcat 的一个应用，所以这时要查看的日志是在 Tomcat 的目录下。
- ◎ Repository Service 或者 Integration Service 启动过程：假如已经可以成功地登录 Admin Console，但是发现 Repository Service 或者 Integration Service 启动失败，这时需要在 Admin Console 中选择 Logs Tab，查看相关的日志，如图 8-1 所示。

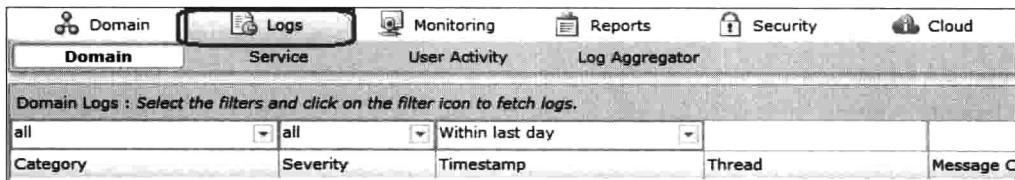


图 8-1

请记住 Repository Service 或者 Integration Service 是基于 C 语言的应用，它们的错误可能和环境变量相关，可能和安装的数据库的版本与 PowerCenter 服务器不兼容相关，等等，如经常出现的错误 pmora\*.dll 或者 libpmora\*.so 无法调用成功。这些错误的主要原因是无法找到这个文件。为什么无法找到呢？究其原因如下。

- (1) PowerCenter 安装不成功，所以文件不全。
- (2) 数据库客户端安装可能不成功或者没安装。
- (3) 环境变量 PATH (Windows 平台) 或者 LD\_LIBRARY\_PATH/LIBPATH (UNIX 平台) 设置不正确。
- (4) PowerCenter 是 64 位的，而安装的数据库客户端是 32 位的等版本位数兼容问题。
- (5) 还要记住一个命令叫 ldd，如果不知道怎么用这个命令，请查询百度帮助一下。

如果在 Repository Service 启动过程中遇到 ORA-XXXXXX 的错误提示，则请查看 Oracle 帮助文档，或者查询百度。如果是 DB2 的错误信息，则请查看 DB2 帮助文档。90%的情况下，这是最快的解决方法。

如果在启动 Integration Service 过程中遇到类似“Code Page Mismatch. Service Process is running in code page [UTF - 8 encoding of Unicode] whereas the service is configured in Admin Console to run the code page [ISO 8859-1 Western European]" when starting Integration Service (322874)”的错误信息，设置环境变量 INFA\_CODEPAGENAME 一定可以帮你解决问题。

请记住这 3 种日志的存放位置，这非常重要：安装目录下的日志；tomcat/logs 目录下的日志；Admin Console 中的日志。没有日志，基本无法定位、解决问题。

## 8.2 开发过程的错误

在 Mapping 或者 Workflow 开发过程中，也常常会遇见各种各样的错误。这些错误情况更加复杂，与更多的细节问题相关，需要大家对 PowerCenter 非常熟悉。

(1) 在 Designer 中：主要查看的是 Output 窗口的日志。如果提示对象是有效的，那就没有问题，继续后面的工作。如果提示某个对象是 INVALID，向上翻页，找到详细的错误提示，通过提示定位问题，如图 8-2 所示。

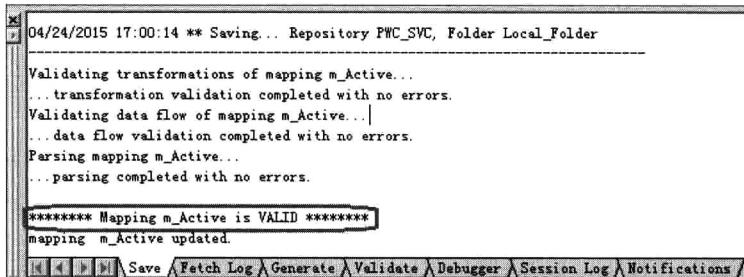


图 8-2

假如 Output 窗口已经被清空，没有历史的提示信息。如果是 Mapping 有问题，需要选择菜单 **Mappings→Validate** 命令，相关对象将被重新校验一次，并将校验日志显示在 Output 窗口中。如果是 Mapplet 无效，就需要选择菜单 **Mapplets→Validate** 命令。

(2) 在 Workflow Manager 中：假如在 Output 窗口看到错误提示或者 Workflow 等对象是无效的，需要查看的也是 Output 窗口中的信息，如图 8-3 所示。

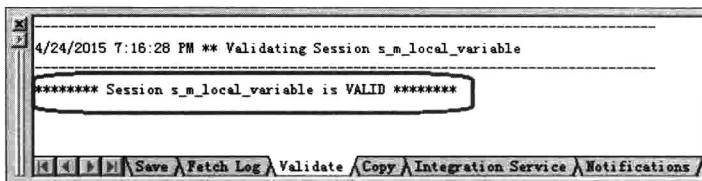


图 8-3

处理 Workflow 中的相关错误有两个常规步骤：① 用鼠标右键单击 Session，在弹出的快捷菜单中选择 Refresh Mapping 命令；② 用鼠标右键单击 Session，在弹出快捷菜单中选择 Validate 命令。有时候 Session 的错误与 Mapping 相关，因此查看对应 Mapping 是否有效也是一个比较有用的方法。

### 8.3 Session 运行错误

如果 Session 运行时发生错误，查找相关的 Log 有 3 个位置：Session Log、Workflow Log 和 Integration Service Log。

- **Session Log：**第一步需要查看的是 Session Log，用鼠标右键单击运行错误的 Session，在弹出的快捷菜单中选择 Get Session Log 命令，请一定仔细阅读 Session Log。
- **Workflow Log：**如果在 Session Log 中无法找到相关有效信息，还需要查看 Workflow Log。用鼠标右键单击包含该 Session 的 Workflow，在弹出的快捷菜单中选择 Get Workflow Log 命令。
- **Integration Service Log：**如果在 Session Log 和 Workflow Log 中都没找到相关信息，打开 Admin Console，选择菜单 Logs，查看 Integration Service Log 的相关信息，有时也非常有帮助。

Session 运行的错误信息处理起来有时候会比较复杂，如果无法很快解决，请登录 [mysupport.informatica.com](http://mysupport.informatica.com) 网站，查询 Informatica Knowledge Base，这个知识库可以帮你解决 80% 以上的常见问题。

若问题仍然无法解决，及时联系 Informatica GCS（Global Customer Service）。

另外，Session 运行时通过一个 pmdtm 进程，这是一个 C 语言程序。

## 8.4 源读或者目标写的错误

源读和目标写错误可能和 PowerCenter 相关，也有可能和数据库相关。一般情况下与 PowerCenter 相关的问题比较容易处理，错误信息一般会出现在 Session Log 中。比较难处理的错误是与源读和目标写相关的错误，这些错误一般和数据库相关。

前面已经提到，如果在 Session Log 中看到诸如 ORA-XXX 的错误，需要查找 Oracle 相关的文档；如果是 DB2 的相关错误信息，请查看 DB2 的相关文档。

如果使用的是 Loader 接口，这时在 Session Log 中的信息往往是不充分的，需要查看 Loader 自己的日志文件，如 sqldlr 的日志文件、IQ 的 loader 日志文件等。

如果是数据库锁引起的错误，日志里往往没有详细的信息。这时，需要请 DBA 帮忙查看数据库是否有相关的锁或者其他有帮助的信息。如果正在使用的是 DB2 或者 Teradata 数据库，尤其需要关注锁的情况。学会如何查看锁和解锁的技巧对 Troubleshooting 非常有帮助。

如果使用 ODBC 作为数据接口，有时候 Session Log 中的提示信息确实不足以支持解决遇到的问题，这时需要使用 ODBC trace 文件。以 DataDirect ODBC 为例，设置 ODBC Trace 如下（在 odbc.ini 文件中设置）：

```
Trace=[0 | 1]
TraceFile=trace_filename
TraceDll=ODBCHOME/lib/xxtrcyy.zz
ODBCTraceMaxFileSize=file_size
ODBCTraceMaxNumFiles=file_number
TraceOptions=0
```

解释一下其中的两个参数。

- ◎ Trace=[0 | 1]: 启动 Trace，设置为 1，否则设置为 0。默认值为 0。
- ◎ TraceOptions=[0 | 1 | 2 | 3]: 设置为 0，使用标准的 ODBC Trace，0 为默认值；设置为 1，在 Log 文件的每个 ENTRY 和 EXIT 中增加时间戳；设置为 2，在 Log 文件中打印文件头；设置为 3，相当于同时设置了 1 和 2。

# 第9章

# PowerCenter 扩展能力

---

这一章包括了若干非常大的主题，可以说这些主题都可以独立成章，甚至独立成书，很难用这么小的篇幅把这些主题都描述得非常清楚。在这里主要关注它们的主要功能、主要特点和一些常见问题，不求能够详细地描述，也更谈不上能使读者看完之后可以独立使用这些功能，但希望能帮助读者基本了解这个功能，以及它的基本特点和使用场景。

## 9.1 PowerExchange CDC（变化数据捕捉）

---

CDC 的全名是 Change Data Capture，即变化数据捕获。CDC 的执行原理是通过读取数据库日志从而获得数据库表的增量数据。Informatica 是最早提供 CDC 功能的厂家之一。早在 2008 年，IBM 尚未收购 Data Mirror、Oracle 尚未收购 Golden Gate 之前，Informatica 就开始在中国推广 CDC 功能。

PowerExchange CDC 与 Data Mirror、Golden Gate 虽然都使用数据库日志的读取技术，但是定位有所不同。在 Informatica 中还有另外的一个产品是与 Data Mirror 和 Golden Gate 相对应的，这个产品叫 Informatica Data Replication，这是一款数据复制产品。既然叫作复制，那就是说源和目标的数据是相同的，或者近乎相同。从反向理解，就是说这种产品的数据转换（计算）能力比较差，比较适合的场景是进行复制。PowerExchange CDC 是与 PowerCenter 紧密集成的一个产品，它充分利用了 PowerCenter 强大的 Transformation 能力，与数据复制产品相比，在实时计算方面更有优势。

PowerExchange CDC 支持从各种数据库和操作系统获取数据增量，主要包括 Oracle、Microsoft SQL Server、IBM UDB（DB2）、AS/400 和大机等。

### 9.1.1 PowerExchange CDC 的 3 种模式

PowerExchange 利用 CDC 技术，在不断的发展变化中，衍生出了 3 个基本的功能：CDC Real-Time、CDC Batch 和 CDC Continous。

#### 1. CDC Real-Time

PowerExchange CDC 在支持数据库的 CDC 技术发展过程中，一般最先开发的是 CDC Real-Time，它通过 CDC 技术读取数据库日志，从而获得数据库的增量数据，并对这些数据进行实时的计算或者直接将这些数据写入到目标数据库中。这个过程中展现了它的两个基本特点：一是能够获取数据库增量数据；二是非常及时、甚至是准实时的数据同步，延时是秒级别的。以 Oracle 数据库为例，它的架构如图 9-1 所示。

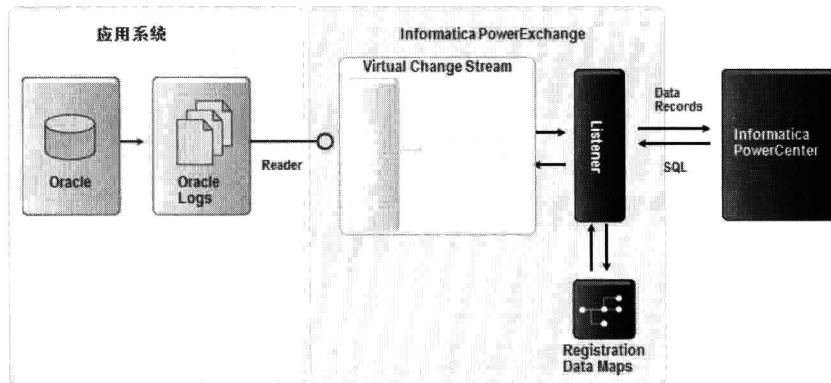


图 9-1

这种模式有非常多的应用场景，如风险实时监控、数据实时分析，还有的企业用它来复制一个与生产相近的数据库，从而释放生产环境的随机查询压力。这些场景都有很多非常成功的案例。

但是，这种方式也有弱点，也需要更多功能的补充，以满足多样化的业务需求。正因为实时，在某些情况下，占用源数据库的资源会比较多，比如 3% 或者更多，不同数据库因为实现不同，差别也比较大。但是很多企业要求的数据及时性不一定都是秒级，有可能分

钟级就可以满足要求。还有一个问题，PowerExchange CDC 为了满足实时计算，它从开始进行设计时就不是从全库或者整个 Schema 开始考虑的。因此一个实时 Session 就会产生一个读取日志的进程。这样，如果需要同时读多张表的数据时，有可能会出现多个进程对日志读的抢夺。考虑一下同时向一个磁盘复制两个大文件的情形，与这种情况非常相似，这样并不能提高效率，反而降低了性能。因此，Informatica 开发了一种新技术——CDC Batch。

## 2. CDC Batch

不管哪种 CDC 方式，PowerCenter 在执行 CDC 之前，有一个过程叫作注册。注册的对象是表，它告诉 PowerExchange CDC 它希望从日志中获取哪些表的增量数据。由于很多客户对获得增量数据的时效性要求不高，只要分钟级别的获取增量数据即可。因此，Informatica 在 CDC Real-Time 之后，增加了一种新的模式——CDC Batch。还是以 Oracle 数据库为例，它的架构如图 9-2 所示。

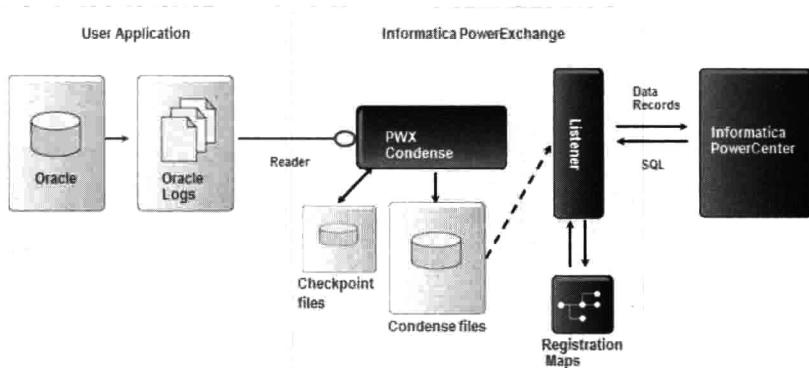


图 9-2

在图 9-2 所示的架构中，PowerExchange Reader 根据注册的表不断地读取 Oracle Logs 中的增量数据，但是它并没有实时地将数据向下游，即 PowerCenter 传送，而是放在了一些临时文件中，这里叫作 Condense File。当下游系统需要时，PowerCenter 会按照定时或者按需索取的方式，从 Condense File 中查看是否有新的增量数据产生。

这样既可以满足对增量数据时效性要求不高的用户的需求，同时也能限制读取生产数据库日志的进程的数量，最大限度地降低对源系统资源的消耗。但是，它也有弱点，就是时效性太差。因此 Informatica 在 CDC Batch 的基础上增加了更新的模式，就是 CDC Continous。

### 3. CDC Continous

前面提到 CDC Batch 虽然保证了只有一个进程对数据库日志的访问，降低了对数据库资源的使用，但是获取增量数据的时效性比较差。天才的开发人员们进行了新一轮改进。他们发现 CDC Batch 时效性差的主要原因是，只有当 Condense 文件关闭时，下游的 PowerCenter 才可以获取该 Condense 文件的数据，因为文件关闭需要满足时间、文件大小的限制，所以造成了时效性差。

能否在 Condense 文件不关闭时，下游的 PowerCenter 也可以读取 Condense 文件的数据？开发人员成功了，这就诞生了一种新模式——CDC Continous。

#### 9.1.2 开放数据库 CDC 基本原理

因为不同数据库的日志管理方式不同，因此 PowerExchange CDC 的实现方法和实现原理也不尽相同。

##### 1. Oracle

最初针对 Oracle CDC 的实现采用的是 Logminer API，后来增加了 Direct Read(Express) 的方式。后者与前者相比大幅度地提升了性能，降低了对 Oracle 资源的消耗。目前 PowerExchange CDC for Oracle 仍然同时保留这两种方式。同时 Logminer API 方式也有部署灵活性的优势。

Logminer API 是 Oracle 提供的日志管理接口，在 Oracle 9i 中曾经有不少的问题，但后来不断修改，还是有了不少的改善。Logminer 模式目前仍然比较慢，但是可以满足绝大多数客户数据量的需要。并且这种模式有个特点，即它不需要在源数据库上安装任何东西。

Direct Read (Express) 模式采用的是直接读取 Oracle 日志的方式，这种方式加快了日志读取和解析的效率，但要求 PowerExchange CDC 必须安装在源系统上。

##### 2. UDB

PowerExchange CDC for UDB 采用的是 db2readlog API（详细内容可以查看 DB2 API 的说明文档）。

### 9.1.3 CDC 常见的一些讨论

#### 1. 占用资源问题

PowerExchange CDC 占用多少源系统资源？3%、5%还是 10%？这是经常被讨论的问题。其实这个问题非常难回答。有几个原因：不同数据库占用的资源是不同的；不同系统配置占用的资源也不同；瓶颈不同占用资源的表现也不相同，等等。

当然资源是很多用户考虑的最重要因素，系统安全需要保证，有的行业还有合规性的要求。其实笔者有时候认为 3% 和 5% 差别不大，尤其是当整个系统资源总量比较充足的时候。但是当系统资源非常有限的情况下，确实应该考虑。同时，如果再增加 2% 的资源可以解决一个业务问题，这似乎也不应该成为障碍。

当然，在实际的使用中，大家都有个目标，即尽力保证所有系统处于最优的状态，这些讨论也是可以理解的。对 PowerExchange CDC，笔者曾经见过仅占用 1%，甚至更少的情况，也曾见过占用 10% 以上的情形。如果非常关注系统资源占用，测试可能是最好的选择。

#### 2. 断点问题

PowerExchange CDC 在 PowerCenter 端存放了 Token 文件，可以作为数据的断点。还有一个方法，在每条数据抽取过程中，观察 CDC Source，会发现有些额外的字段，那里也存放了断点信息，即每一条数据跟随一个断点信息。

#### 3. 字符集问题

CDC 字符集问题更加复杂，对开放数据库，建议首先应该考虑数据库自身的字符集的设置，如，Oracle 的 NLS\_LANG、DB2 的 db2set。但是，当遇到 AS/400 和大机时，字符集会变得非常复杂，典型的包括：不同的字段有不同的字符集；在 65536 中存放了中文字符集；用户使用了企业自定义字符集。但可以肯定的是 PowerExchange 都可以处理，不过要采用不同的方法，如 Datamap、dbmover.cfg 中强制字符集转换等。在开放数据库中有时也会出现混杂字符集的状况，如 Oracle 数据库本身字符集是 ASCII-7，但是用户在里面存放了中文 GB2312，同时有的程序员还用 Java 程序写入了 GB10086 字符的中文。

上面主要介绍了 PowerExchange CDC 的原理，下面讲解 CDC 安装、配置的过程，帮忙读者快速理解和学习。

### 9.1.4 CDC Real-Time for Oracle 安装配置（实例）

考虑是否应该增加这部分内容，是个非常纠结的过程，因为完整地介绍 CDC 需要非常大的篇幅。而 CDC 的 Mapping/Session 开发并不难，主要的难点在安装配置过程。但是，笔者最终还是决定在这部分增加一个 CDC 引子，以最常用的数据库 Oracle 为例，演示 Oracle CDC Real-Time 的部分，以帮助读者了解一个最基本的 CDC 安装、配置过程。主要演示分为以下几个部分。

第一步：CDC 部署架构及安装。

第二步：CDC 服务器及客户端配置。

第三步：PowerCenter 开发及运行、测试。

这部分内容不求精确、全面，目标仅仅是帮助读者能够自己使用 CDC。了解 CDC 最好是从 CDC 架构开始。

以一个典型的 CDC 部署场景为例，在这个环境中由 3 台机器，分别是数据库服务器 A、ETL 服务器 B 和 ETL 客户端工具 C。CDC 需要在这些机器上进行安装。在数据库服务器 A 上安装的是 CDC Server，在 ETL 服务器 B 上安装的是 CDC Client，在 ETL 客户端上安装的也是 CDC Client，如图 9-3 所示。

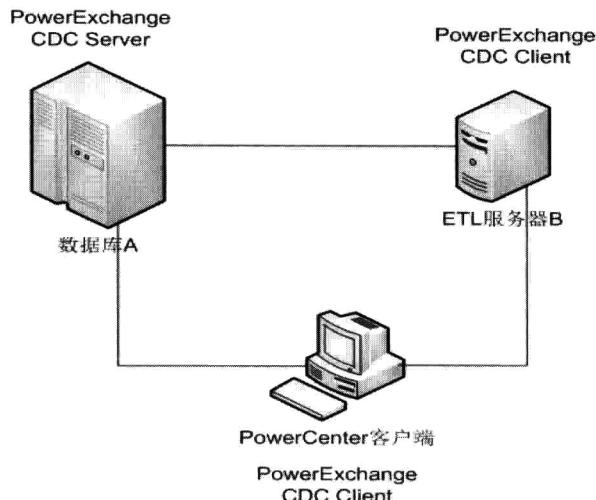


图 9-3

→ **注释**

PowerExchange 的安装包没有区分客户端与服务器，安装过程只要找到对应的操作系统的安装包即可。但 ETL 客户端只能安装 PowerExchange CDC 32 位 Windows 版，因为 PowerCenter 客户端只有 32 位 Windows 版的。

安装过程也比较简单，只要拿到 PowerExchange CDC Key，一路单击 Next 按钮即可。

服务器端的配置过程分为如下两部分。

- ◎ Oracle 数据库配置。
- ◎ PowerExchange CDC 配置。

在后续的配置中，以 PowerExchange CDC for Oracle 的 Express 模式为例。

## 1. Oracle 数据库配置

在 Express 模式下，所有关于数据库的配置脚本都存放在\$PWX\_HOME 目录下的 ora\_orad.sql 文件中。脚本主要包括如下几部分。

- ◎ 启动归档模式。
- ◎ 创建 Oracle 用户并为用户授权。
- ◎ 添加 Supplemental Log。

所有的操作都需要 DBA 用户执行，详细内容请参考脚本 ora\_orad.sql。

## 2. PowerExchange CDC 配置

PowerExchange CDC 的配置主要是 PowerExchange 在服务器端的配置，客户端的配置比较简单。

- ◎ 在服务器端首先要配置的是 dbmover.cfg 文件。

```
NODE=(local,TCPIP,127.0.0.1,2480)
NODE=(node1,TCPIP,127.0.0.1,2480,262144,262144,262144,262144)
NODE=(default,TCPIP,x,2480)
APPBUFSIZE=256000
COLON=:
CONSOLE_TRACE=N
DECPOINT=.
```

```

DEFAULTCHAR=*
DEFAULTDATE=19800101
MAXTASKS=30
MSGPREFIX=PWX
NEGSIGN=-
PIPE=|
POLLTIME=1000
PRE861_COMPAT=N
EXT_CP_SUPPT=Y
ORACLE_CAPTURE_TYPE=D
CAPT_PATH=/home/infa/Informatica/PowerExchange9.6.1/capture/
CAPT_XTRA=/home/infa/Informatica/PowerExchange9.6.1/capture/xtramaps
LOGPATH=/home/infa/Informatica/PowerExchange9.6.1/logs/

ORACLEID=(COLLNAM1,orcl)
CAPI_CONNECTION=(NAME=CAPIORAD,TYPE=(ORAD,ORACOLL=COLLNAM1,PARMFILE=/ho
me/infa/Informatica/PowerExchange9.6.1/pwxorad.cfg))

```

这个 dbmover.cfg 的例子仅仅考虑了 CDC Real-Time 的情况，里面有几个关键的参数需要注意。

- ◆ 参数 ORACLE\_CAPTURE\_TYPE：设置选择 Logminer 模式还是 Express 模式。值 L 为 Logminer 模式，并且需要在 CAPI\_CONNECTION 中使用 ORCL；值 D 为 Express 模式，并且需要在 CAPI\_CONNECTION 使用 ORAD。
- ◆ 参数 LOGPATH：指定 PowerExchange 的 Log 文件 Detail.log 的存放目录。
- ◆ 参数 ORACLEID：指定 Oracle 数据库的连接串。
- ◆ 参数 CAPI\_CONNECTION，必须配置，参考如上样例。

完成 dbmover.cfg 配置后，还需要配置 Express 模式的配置文件 pwxorad.cfg。

◎ 参数文件 pwxorad.cfg 的内容如下：

```

DATABASE
CONNECT_STRING=orcl
EPWD=1C9FA8F7AC7CA213
PASSWORD=oracapt1
ORACLEID=orcl
USERID=oracapt1

```

```
;  
DICTIONARY  
MODE=STATIC  
SOURCE=ONLINE  
EXCEPTIONS=FAIL  
;  
READER MODE=ACTIVE  
;
```

在 pwxorad.cfg 文件中需要注意：① 分号一定要有；② EPWD 是加密密码，它是通过命令 dtlcrypx 产生的；③ 当 READER MODE 的值为 ACTIVE 时可以读取 Redo Log 的内容，当值为 ARCHIVEONLY 时仅仅读取 Archive Log 中的内容。

#### ◎ 环境变量配置。

安装完 PowerExchange 后，至少还需要配置 3 个环境变量，分别是：PWX\_HOME、PATH 和 LD\_LIBRARY\_PATH。

另外，Oracle 的相关环境变量也要添加到 Profile 中，以保证安装 PowerExchange 的用户可以使用 Oracle 的 Library 和可执行文件，并能读取 Oracle 中的数据。

#### ◎ 启动 PowerExchange 和验证是否启动成功。

PowerExchange CDC 使用命令 dtlslst node1 进行启动。如果希望启动到后台，则需要配合使用 nohup & 命令。

检查是否启动成功，使用命令 dtlrexec prog=ping loc=node1。这个命令还可以用于检查客户端与服务器的连通性。

#### ◎ 配置 PowerExchange 客户端。

PowerExchange 客户端的配置非常简单，同样是配置 dbmover.cfg 文件。

只需修改一个参数：NODE=(node1,TCPIP,192.168.75.141,2480)。其中 192.168.75.141 是服务器 IP 地址；2480 是服务器端口号；node1 是服务器在本地的别名。配置完成后，还需检查是否能与服务器连通，使用命令 dtlrexec prog=ping loc=node1 即可。

至此，已经完成了 PowerExchange CDC for Oracle 的 Express 模式的基本安装、配置。

### 9.1.5 CDC 定义注册组和添加捕获注册（实例续）

在使用 PowerCenter 开发 Mapping 之前，还有一个过程叫作注册“注册组”。安装 PowerExchange 之后，在 Windows“开始”菜单中会找到一个名为 PowerExchange Navigator 的客户端。

首先打开 PowerExchange Navigator 客户端，用鼠标右键单击“注册组”，在弹出的快捷菜单中选择“添加注册组”命令，如图 9-4 所示。



图 9-4

之后，会看到如图 9-5 所示的对话框，在该对话框中输入一个名字（最好是数据库名），选择节点、数据库类型，输入用户名/密码，输入集合标识符。集合标识符是 PowerExchange 服务器端的 dbmover.cfg 文件中配置的 ORACLEID 的一个属性。

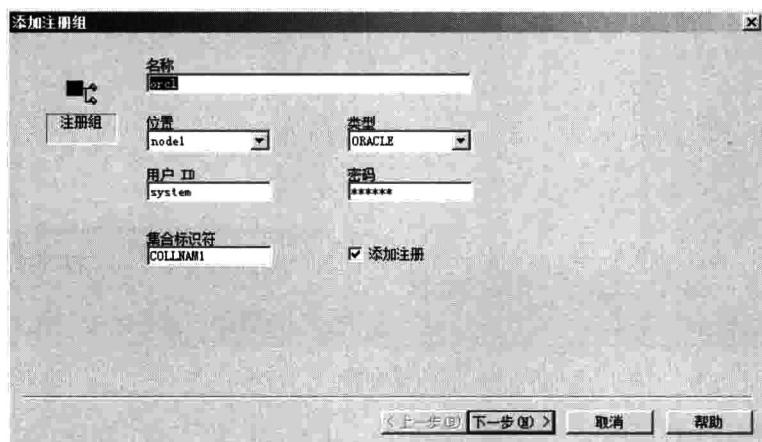


图 9-5

单击“下一步”按钮，进入对象发现、过滤窗口，如图 9-6 所示。这里的名称可以认为是表名。“表筛选”是过滤选项，目标是为了快速获取要注册的表。“架构”是数据库的用户名或者 Schema 名。

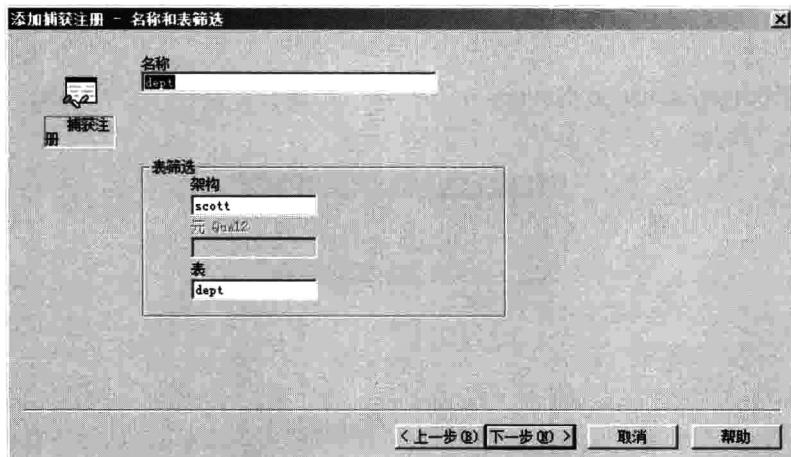


图 9-6

单击“下一步”按钮，在图 9-7 中选择要跟踪变化的列，继续单击“下一步”按钮。

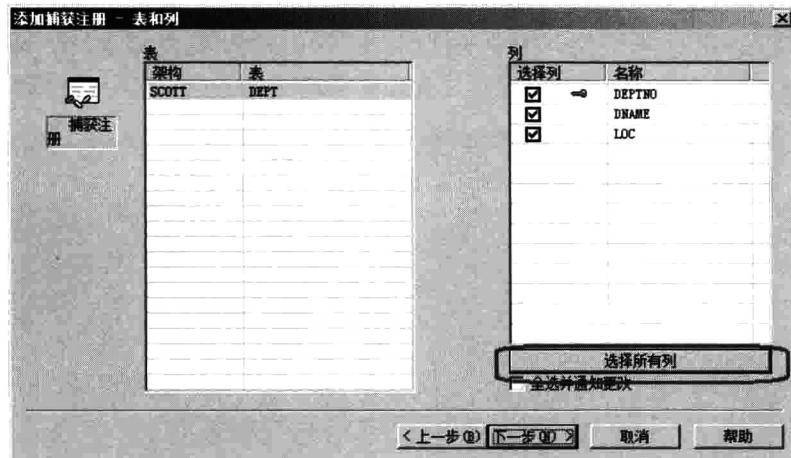


图 9-7

在图 9-8 所示的对话框中，选择类型=同步、状态=活动、压缩=部分，勾选“立即执行”复选框。PowerExchange 将在正在注册的表上增加 SUPPLEMENTAL Log，会自动执行类似

语句“ALTER TABLE "SCOTT"."DEPT" ADD SUPPLEMENTAL LOG GROUP SCOTT\_DEPT ("DEPTNO", "DNAME", "LOC") ALWAYS;”。

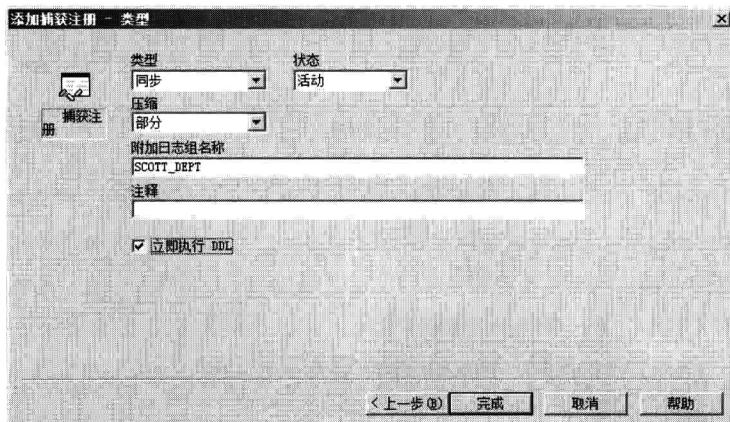


图 9-8

这时，就在 PowerExchange 中注册了一个注册对象 (Dept)，增加了一个注册组 (Orcl)。在 PowerExchange Navigator 的“资源浏览器”中会看到一个新的“注册组”、“提取组”和“应用组”，都叫作 Orcl。

双击“提取组”的 Orcl，可以进行“提取组”内对象的测试，即 CDC 测试。如对 Dept 表进行测试，可以看到如图 9-9 所示的对话框，按照图中的信息填写即可，单击“执行”按钮可以检查前面的安装、配置是否正确。

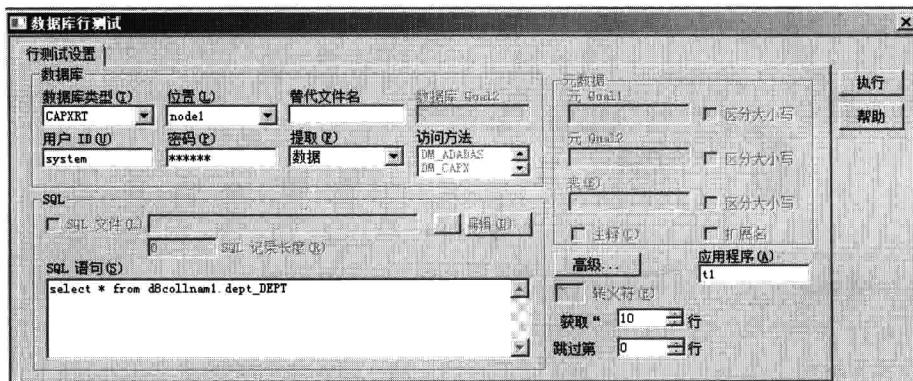


图 9-9

选择执行后，如果可以在图 9-9 的输出窗口中看到输出数据，则证明 PowerExchange 注册成功完成。

对象注册之后，就可以进入 Mapping 的开发。

### 9.1.6 CDC Mapping 开发及运行（实例）

与正常的 Mapping 开发一样，首先需要导入数据源，但与导入关系型数据源有所不同。具体步骤是在 PowerCenter Designer 中选择菜单 Sources→Import from PowerExchange 命令。这时看到的是如图 9-10 所示的对话框，填写用户名/密码，勾选 CDC Datamaps 复选框，选择 Source Type 为 ORACLE，单击 Connect 按钮后就可以看到已经注册的 Datamaps。

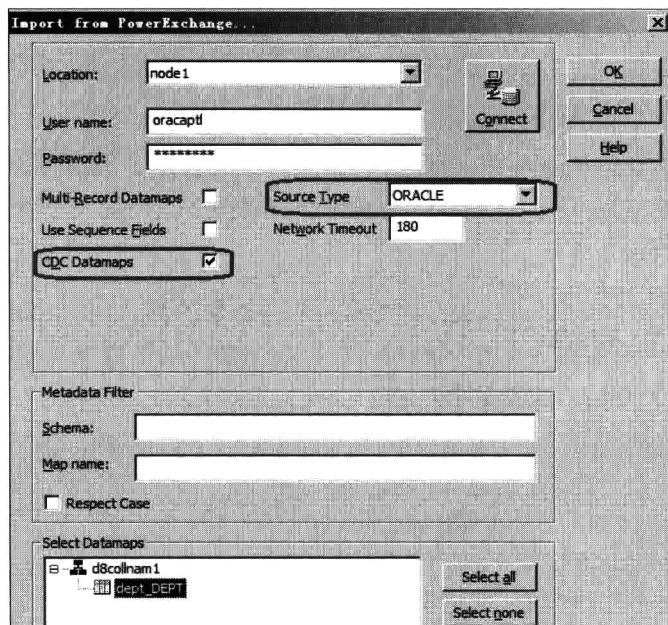
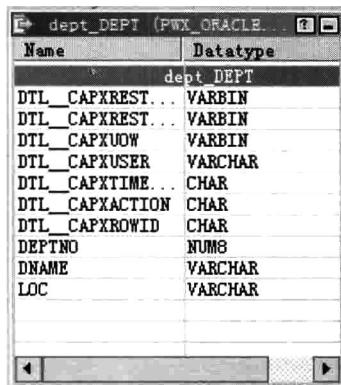


图 9-10

导入后的数据源与关系型数据源也不尽相同，在这张源表中增加了一些 DTL 开头的列，如图 9-11 所示。这些列有很多的用途，有助于开发更灵活的 Mapping。但在当前的例子中，不会演示这些列的更复杂功能。



This screenshot shows the Oracle database schema for the DEPT table. The table has 14 columns:

Name	Datatype
dept_DEPT	
DTL_CAPXREST...	VARBIN
DTL_CAPXREST...	VARBIN
DTL_CAPXUOW	VARBIN
DTL_CAPXUSER	VARCHAR
DTL_CAPXTIME...	CHAR
DTL_CAPXACTION	CHAR
DTL_CAPXROWID	CHAR
DEPTNO	NUM8
DNAME	VARCHAR
LOC	VARCHAR

图 9-11

为其创建一个简单的 Mapping，如图 9-12 所示。

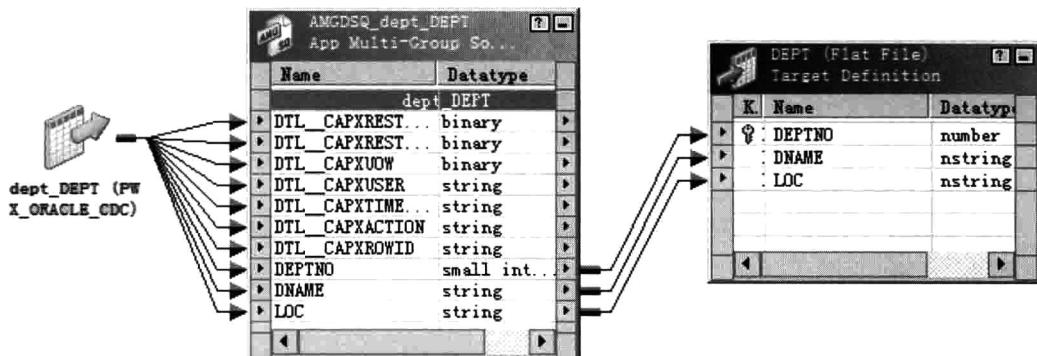


图 9-12

在开发 Workflow 之前，需要为 CDC 定义一个数据源。CDC 需要专门的数据源定义，它的定义过程是：在 Workflow Manager 中选择菜单 Connection→Application→New→PWX\_Oracle\_CDC\_Real\_Time 命令。参考图 9-13。

输入 Location，即在 PowerCenter 服务器端的 dbmover.cfg 文件中配置的节点名。还有两个地方需要注意：一是字符集需要选择 UTF-8 encoding of Unicode；二是 Image Type 一般选择 AI。Image Type 有两个选项：AI（After Image）和 BA（Before and After Image），它指的是 DML 语句的操作结果。选择 AI，PowerExchange 会返回 Insert、Delete 和 Update 执行结果的那条数据；选择 BA，Insert 和 Delete 与 AI 的返回值一样，但如果是 Update，会返回两条数据，分别是 Update 前后的数据。

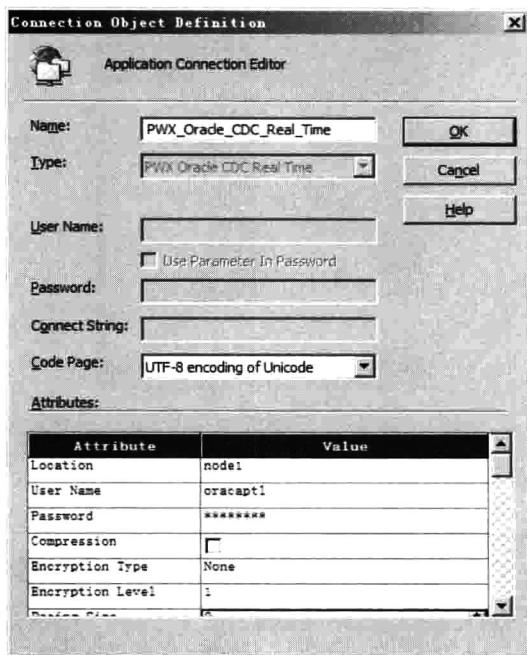


图 9-13

为这个 Session 指定源和目标之后，即可以执行。与传统的 Session 不同，这个 Session 如果不手工停止，将一直处于 Running 状态，实时监控 Oracle 服务器端数据的变化，如图 9-14 所示。

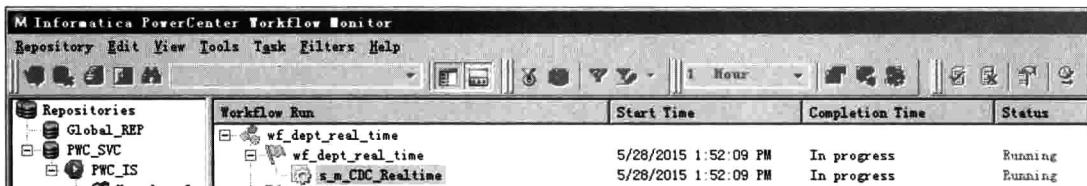


图 9-14

## 9.2 PowerCenter 与 SAP

不得不承认 SAP 是迄今为止最成功的企业软件，大量的制造、零售、集团型客户都在使用 SAP。SAP 系统保存的数据是这些企业正待挖掘的财富，但 SAP 对很多用户而言是神

秘并且复杂的。客户无法或者很难获取 SAP 内部保存的数据，或者在获取 SAP 数据时效率很低。Informatica 发现了这个问题，及时开发了针对 SAP 的接口，包括 R/3、BW 的接口，帮助客户简单、高效地获取这些保存在 SAP 中非常有价值的数据。

### 9.2.1 R/3、mySAP、ECC

借用一个 Informatica 的标准图，说明一下 PowerCenter 可以支持通过哪些协议获取 SAP 的数据，如图 9-15 所示。

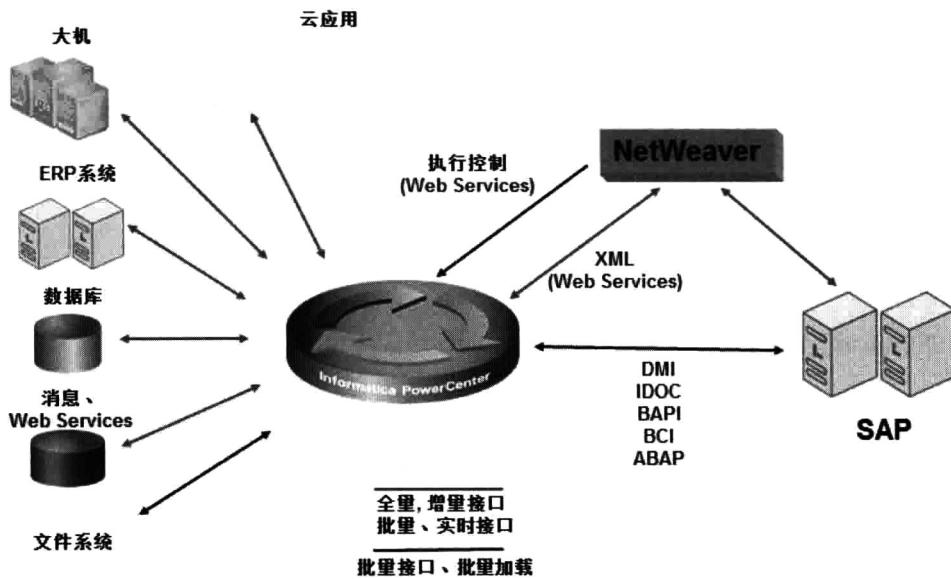


图 9-15

对 R/3 而言，PowerCenter 可以支持 SAP 的接口包括 ABAP、BCI、RFC/BAPI、IDOC 和 Web Service，并且可以和 SAP 双向交换数据。开发过程是全图形化开发，如果有机会尝试使用 PowerCenter 抽取 SAP 数据，相信你一定会惊讶访问 SAP 数据变得如此简单。

另外需要说明的是，在 PowerCenter 访问 SAP 时，并非直接访问存放数据的数据库，而是访问 SAP 的应用层，这使得数据更有意义，更加容易理解。一图胜千言，当使用 PowerCenter 通过 ABAP 访问 SAP 透明表、层次和 IDOC 时，只需输入用户、Client#等，就可以看到 SAP 中实际存放的数据结构，如图 9-16 所示。

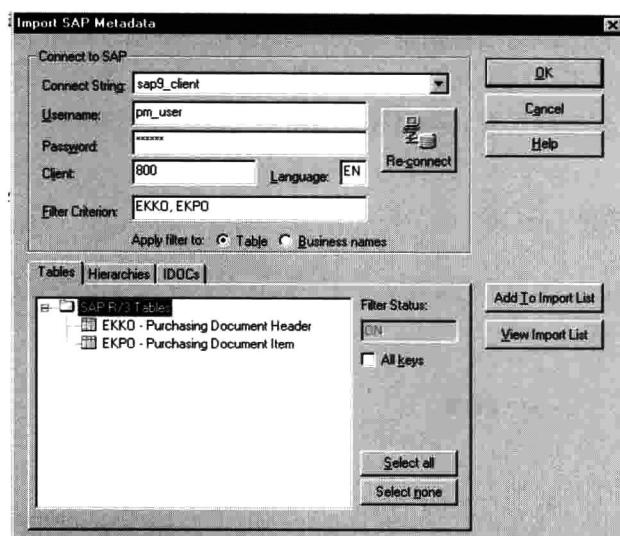


图 9-16

当遇到的问题是如何访问 SAP BW 数据时，同样可以考虑 PowerCenter。

### 9.2.2 PowerCenter 与 BW

PowerCenter 可以使用 Open Hub Destination 或者 Open Hub Service 抽取 SAP BW 的数据。可以抽取的对象包括 InfoCubes、DataStore Objects、InfoObjects 和 InfoSets。图 9-17 展示了 PowerCenter 抽取 SAP BW 的过程。

图 9-17 的上半部分是 SAP BW 的相关设计组件，包括 Open Hub Service、各种数据源及 Process Chain。在图 9-17 中，可以认为 Open Hub Service 作为一个中间媒介从各种 BW 数据对象中获取数据，并将数据放入一张 SAP 透明表中，PowerCenter 再从透明表中抽取数据。

因为 SAP BW 是一个 ROLAP 系统，这意味着 SAP BW 本身不是多维存储的，因此笔者也曾经遇到过用户不使用上面的架构，而是使用 ABAP/RFC 从透明表中直接获取数据。

#### → 注释

ROLAP 是联机分析处理的一种，它对存储在关系数据库而非多维数据库中的数据进行动态多维分析。

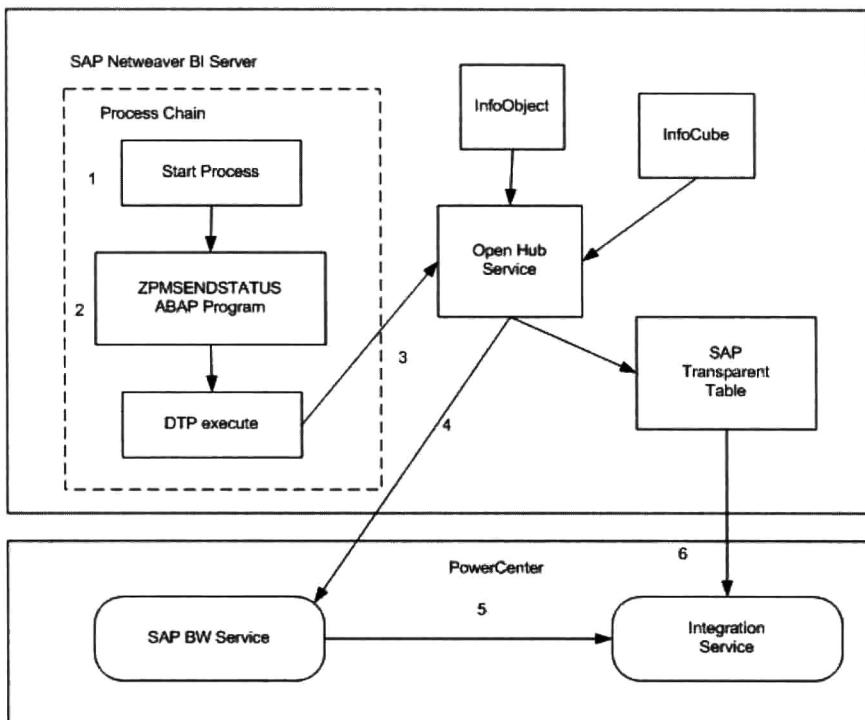


图 9-17

在 SAP BW 抽取架构中，有一个 SAP BW Service，它是创建在 PowerCenter 的 Admin Console 上的一个服务，这个服务通过 SAP External System 实现与 PowerCenter 进行通信，可以认为它是站在 SAP 门口与 PowerCenter 沟通的媒介。当 PowerCenter 抽取 BW 数据时，源头是从 SAP 的 Process Chain 触发的，当所有要抽取的数据进入透明表后，SAP 会告诉 SAP BW Service（这是 PowerCenter 上的一个服务），它已经完成了 SAP 内部的处理任务，PowerCenter 可以从透明表抽取数据了。这就是图 9-17 中 1、2、3、4、5、6 的含义。PowerCenter 也提供了一个外部命令触发 Process Chain。

### 9.3 PowerCenter 与 MPP 数据库

MPP（Massively Parallel Processing）数据库主要用于数据仓库的世界，它的大致定义是：数据分布存储在各个节点上，运算时将任务并行的分散到多个服务器和节点上，在每

个节点上计算完成后，将各自部分的结果汇总在一起得到最终的结果。MPP 数据库的典型代表包括 Greenplum、Teradata、Netezza 等。

当 PowerCenter 与这类 MPP 数据库集成时，最典型的特点是读取或者加载的速度非常快，例如，PowerCenter 可以利用 Greenplum 的分布式数据加载，实现并行向 MPP 数据库的各个节点并行加载数据。其次是功能全，例如对 Teradata，PowerCenter 可以支持几乎所有 Teradata 提供的接口能力，包括如下几种。

- ◎ FastLoad：快速向 Teradata 表加载数据，但要求表必须是空表。
- ◎ FastExport：快速从 Teradata 数据库中读取数据的批量接口。
- ◎ TPump：支持流式数据处理，但是性能略慢。
- ◎ MultiLoad：支持对 Teradata 数据进行 Insert、Update、Delete 和 Upsert 等批量操作，当这个接口与 PowerCenter 结合时，可以实现不使用 Teradata 临时表而方便地构建拉链表，帮助用户节省昂贵的 Teradata 节点投资。
- ◎ Teradata Parallel Transporter：是 Teradata 为了简化 Load 接口提供的一种新的接口方式。

**Pushdown for MPP 功能：**PowerCenter 还陆续提供了一些针对 MPP 数据库的 Pushdown 功能，它可以将 PowerCenter 的 Mapping 自动转换为 MPP 数据库支持的 SQL 语句，充分利用 MPP 数据库分布式计算的能力，加快数据加工的过程。同时 Pushdown 功能还能减少数据的移动，从而节省数据加工的时间。

## 9.4 PowerCenter 与 Hadoop

Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构。

用户可以在不了解分布式底层细节的情况下，开发分布式程序，充分利用集群的威力进行高速运算和存储。

Hadoop 实现了一个分布式文件系统（Hadoop Distributed File System，HDFS）。HDFS 有高容错性的特点，并且支持部署在低廉的（Low-cost）硬件上；而且它提供高吞吐量（High Throughput）来访问应用程序的数据，适合那些有着超大数据集（Large Data Set）的应用程序。

Hadoop 框架最核心的设计就是 HDFS 和 MapReduce。HDFS 为海量的数据提供了存储能力，MapReduce 为海量的数据提供了计算能力。

Hadoop 由于其扩展能力及起源于开源的特点，几乎成为大数据技术的代名词。Informatica 也不会放弃这样的机会，借 Hadoop 的东风扩展其市场占有率，因此在业界首先提供了对 Hadoop 的支持。

PowerCenter 在 Hadoop 支持方面，主要提供了如下两个能力。

- ◎ 接口能力，即访问 Hadoop 及其生态的能力。
- ◎ PowerCenter on Hadoop，即将 Hadoop 当作 ETL 引擎的能力。

#### 9.4.1 接口能力

截至笔者书写这个章节时，PowerCenter 对 Hadoop 接口主要支持 HDFS、HIVE、HBase 和 Pivotal HD HAWQ。

HDFS、HIVE、HBase 均同时支持读和写的能力。因为 HAWQ 继承了 Greenplum 数据库的一些特性，因此 PowerCenter 提供了非常强大的并行写 HAWQ 的能力。依笔者的经验，如果不存在网络和源系统的瓶颈，写 HAWQ 的速度非常惊人。另外，使用 Cloudera 的用户会有关于 PowerCenter 是否支持 Impala 的疑问，尤其是写 Impala。因为 Impala 和 Hive 共享元数据，因此可以通过写 Hive 实现写 Impala。

提到 PowerCenter 与 Hadoop 的关系，不得不提一下 PowerCenter 客户端。目前 PowerCenter 主要在用的客户端包括 Designer、Workflow Manager 和 Workflow Monitor 的客户端，被称为 PowerCenter Classic Client。如果你使用的是 PowerCenter 比较新的版本，安装完成后，还会有一个客户端叫作 PowerCenter Developer，这个客户端是在 Eclipse 的基础上定制的。

对 PowerCenter Classic Client，目前支持访问 Hadoop 的能力主要如下。

- ◎ HDFS。
- ◎ HIVE。
- ◎ Pivotal HD HAWQ。

而 PowerCenter Developer 客户端支持访问 Hadoop 的能力主要如下。

- ① HDFS。
- ② HIVE。
- ③ HBase。

相信随着时间的推移，这些能力会不断得到增强。

下面以 PowerCenter Client Classic 为例，演示 Hadoop PowerCenter 如何访问 Hadoop。以写 Hadoop 为例，有如下几个关键步骤。

第一步：创建以文本为目标的 Mapping。

源可以是任意的数据库或者应用，但目标需要是文本文件。

第二步：在 Workflow Manager 中创建 Hadoop 连接。

在 Workflow Manager 中创建针对 Hadoop 的连接，选择菜单 Connections→Application 命令。

新建一个 Hadoop HDFS Connection，并完成如图 9-18 的信息。

- ① HDFS Connection URI：一般为 `hdfs://Namenode:Port`。
- ② Hive URL：`jdbc:hive2://Namenode:Port/<Database Name>`。
- ③ Hadoop Distribution：选择不同的 Hadoop 发行版本。

第三步：设置 Session 访问 Hadoop。

双击 Session，打开 Mapping Tab，如图 9-19 所示。Writers 选择 HDFS Flat File Writer，Connections 选择在第二步创建的 Hadoop HDFS Connection。

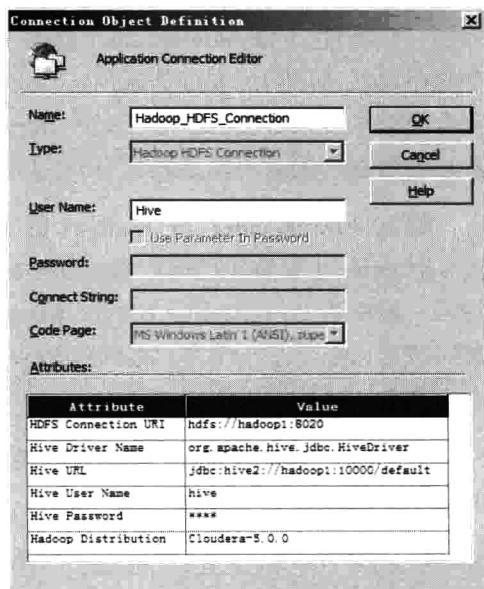


图 9-18

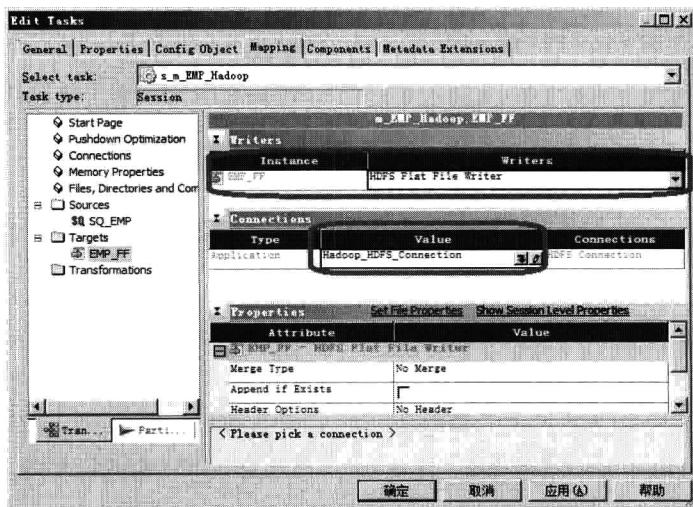


图 9-19

同样是在 Session 的 Mapping Tab 中, 将 Writers 和 Connections 收回, 可以看到如图 9-20 所示的更详尽的信息。

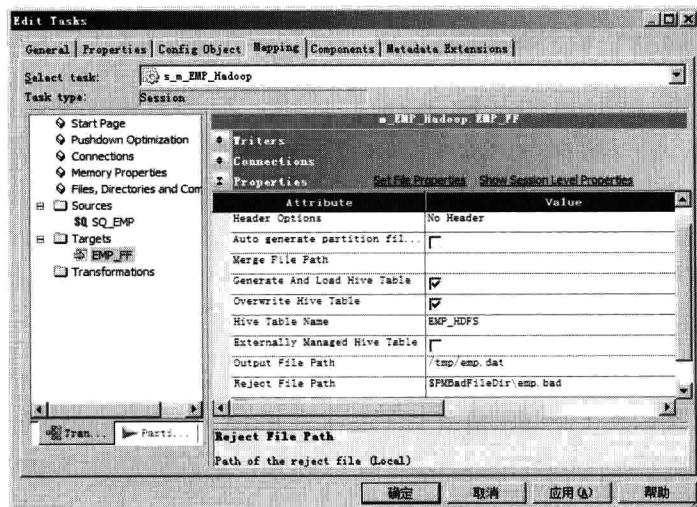


图 9-20

如果仅仅写文件到 HDFS, 只需要关注如下两项。

- ◎ Output File Path: HDFS 文件路径, 而不是 PowerCenter 服务器文件路径。

- ◎ Reject File Path: PowerCenter 服务器文件路径, 而不是 HDFS 文件路径。

如果需要写 Hive, 仅仅关注上面的两项就不够了, 至少需要关注如下三项。

- ◎ Generate And Load Hive Table: 这是生成 Hive table 的必选项。
- ◎ Hive Table Name: 表名, 这个选项一定要有。
- ◎ Overwrite Hive Table: 这是一个可选项, 从名字可以看出, 重点是是否已经存在这张表。

大家一定知道, Hive 是 HDFS 的一种 Shell, 即为访问 HDFS 提供了一种类似 SQL 的 Shell 脚本。PowerCenter 生成 Hive Table 的过程是: 首先将文件传送到 HDFS 上, 即将数据写入指定的 HDFS 目录中; 然后再将上一步目录中的数据转移到 `hive/warehouse/...` 目录下; 最后为此文件建立 Hive Catalog。这时, 一个 Hive Table 就创建完成了。

在这个过程中, 经常发生错误是由于权限的问题。因为在 Hadoop 安装过程中不同的发行版本都创建了很多用户, 如 Hive、HDFS、Hadoop 等。在这个过程中相互之间的授权都有细微的差别, 因此经常遇见的错误是权限问题。

#### 9.4.2 PowerCenter on Hadoop

PowerCenter on Hadoop 的功能类似前面提到的 Pushdown 的功能。Pushdown 是将 PowerCenter 的 Mapping 转换为 SQL 语句, 并将这些语句运行在数据库服务器上, 充分利用数据库的资源。PowerCenter on Hadoop 是将通过 PowerCenter Developer 开发的 Mapping 转换为 MapReduce, 然后通过 Hadoop 引擎执行 ETL 过程。这样可以充分利用 Hadoop 集群的能力, 同时可以简化 MapReduce 的开发。换句话说, 就是为 Hadoop 平台的 ETL 程序开发提供了一个图形化的界面。

实际上, 当 PowerCenter 在 Hadoop 执行时, 并不是直接将 Mapping 转换为了 MapReduce。它采用的是一个间接的方法, PowerCenter Mapping 在转换过程中使用 HQL (Hive SQL) 作为媒介, 首先将 Mapping 转换为 HQL, 然后将 HQL 转换为 MapReduce。

谈到这里, 笔者经常会被问到: PowerCenter 为什么不直接将 Mapping 转换为 MapReduce? 笔者的理解是这样的: 首先, PowerCenter 转换为 HQL 比较简单, Informtica 开发团队更熟悉 HQL; 其次, PowerCenter 直接转换为 MapReduce 会面临大量的优化问题, 如果采用间接的方法, PowerCenter 则可以间接利用 HQL 进化的成果。最近 Cloudera 宣布会

继续推广 Hive on Spark，是为了考虑包括前期大量 Hive 用户的开发成果。同理，随着 Hive on Spark 的完善，PowerCenter 也会更容易在 Spark 上运行，从而充分利用 Spark 的优势。

PowerCenter on Hadoop 还包括大量的内容，如架构、执行原理等。Hadoop 的世界是一个快速变化的世界，这一章的内容也会具有一定的时效性。随着 Hadoop 自身技术的发展，以及 PowerCenter 自身技术的改进，很多内容可能很快就会发生变化。因此，这部分内容仅供参考。

## 9.5 元数据管理与业务术语管理

什么是元数据？元数据为数据提供了一个参考框架。Forrester Research 将元数据定义为“用于描述数据、内容、业务流程、服务、业务规则及组织信息系统的支持政策或为其提供上下文的信息”。通俗地讲，元数据是关于数据的数据。例如，苹果公司旗下的 App Store 在网上销售软件应用程序，在此情况下数据是应用程序。元数据则是关于这些应用程序的信息，包括应用程序描述、价格、用户评级、评论和开发公司等。在数据管理环境下，存在着多种相关类型的元数据。

- ◎ 技术元数据：提供有关数据的技术信息，如数据源表名称、列名及数据类型（如字符串、整数等）。
- ◎ 业务元数据：提供数据的业务背景，如业务术语名称、定义、责任人或管理员及相关的参考数据。
- ◎ 操作元数据：提供关于数据使用方面的信息，如最近更新的数据、访问次数或最后访问的数据的时间、方式等。

元数据的内涵是比较明确的，但是元数据的外延是无限扩展的。只要是关于数据的数据都是元数据。Informatica 产品起源于数据仓库项目，元数据管理在数据仓库元数据管理方面有比较大的优势。

使用 Informatica 元数据管理，首先需要在 Admin Console 中创建一个 Metadata Manager Service。步骤是：登录 <http://<hostname>:6005>，选择菜单 Action→New→Metadata Manager Service 命令。当 Metadata Manager Service 创建成功后，即可以执行元数据管理的相关功能。

Informatica 元数据管理主要提供了以下能力。

### 9.5.1 元数据的血缘分析

Metadata Manager 界面提供了元数据血缘分析功能，提供数据流过数据集成环境时的直观视图。Metadata Manager 从企业应用程序、数据库、平面文件、PowerCenter、数据建模工具及商业智能工具中采集元数据，并将这些元数据关联起来，以直观的方式进行展现，如图 9-21 所示。

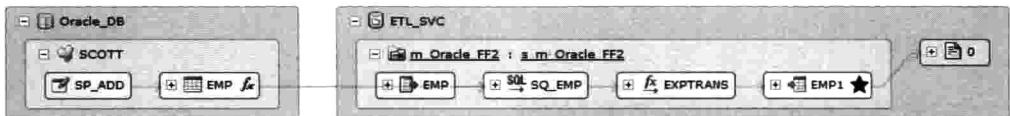


图 9-21

在数据集成环境中，随着规模的不断扩大，元数据管理的作用越来越明显。很难想象在一个企业级的数据集成环境中没有元数据管理，一个细微的变更或者审计需求带来的复杂性。下面列举几个血缘分析的常用功能。

- 支持全局数据视图：使用元数据管理可以清晰地了解企业全局的数据视图。
- 解决数据质量问题：如“这个报告中的这个值似乎不对，能否快速地找到它的数据源和责任人呢？”
- 审计与合规性：允许审计人员对数据从源到目标进行追踪，确保获得正确的数据源，使用正确的计算方式。

### 9.5.2 元数据影响分析

这是一项重要功能。下面通过两个案例了解元数据影响分析的重要性。

**案例 1：**在一家企业中，随着业务的发展，需要对一个描述客户的关键字段的长度进行变更。在不考虑其他交易型系统的情况下，仅仅以数据仓库项目为例，如何找到修改这个关键字段影响的程序，都是一个巨型任务。因为这个字段可能被数以百计的程序所引用，同时又产生了数个衍生字段，这些衍生字段继续在整个数据仓库环境中进行扩散。试想一下，如果没有元数据管理，如何实现这个变更？

**案例 2：**如果业务用户要求 IT 部门对某一关键绩效指标（KPI）的计算方法进行变更，将会造成什么影响？改变计算方法相对来说比较容易。分析的难点是了解在一个可能相当庞大的数据集成环境中，有哪些数据对象依赖于该指标。查找并处理此类相互依赖性可能花费很长时间。

而元数据影响分析正是解决这方面问题的利器，在企业中部署了全面的元数据管理后，这些问题将不再复杂。

### 9.5.3 业务数据管理

Business Glossary 为定义和管理业务术语、沟通参考数据、链接、临时记录及备注的生命周期提供了集中管理点。业务词汇表对于团队之间和跨业务部门的有效交流意义重大。它同时还为 IT 部门就各种项目进行交流合作提供了重要的上下文环境。Informatica 最新的业务术语管理还提供了一个桌面的查询工具，客户可以方便地通过此桌面查询工具快速地查找业务术语的定义，如图 9-22 所示。

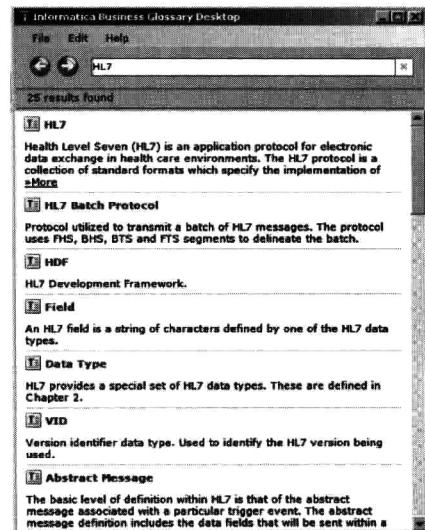


图 9-22

## 9.6 B2B Data Transformation

Informatica B2B Data Transformation 支持结构化、非结构化与半结构化数据类型的转换，同时还支持多个行业的数据标准。先讨论一下概念。

- ◎ 非结构化。
  - ◆ 构成 80% 的企业数据。
  - ◆ 无合适的元数据对其进行全面描述。
  - ◆ 传统的自定义编码转换。
  - ◆ 示例：PDF 文件、电子数据表、Word 文档、遗留系统格式、打印流。
- ◎ 半结构化。
  - ◆ 可利用部分元数据对其进行描述。
  - ◆ 示例：EDI（通常为客户特别要求）、遗留 COBOL 衍生的数据。
- ◎ 结构化。
  - ◆ 可利用元数据明确定义。
  - ◆ 深层次和递归结构。
  - ◆ 示例：XML 标准（如 ISO 20022、HL7 v3、ACORD、RosettaNet、XBRL）。

笔者参与过的一些 B2B Data Transformation 的应用案例主要是通过 B2B Data Transformation 支持包括企业总部与分支机构、上下游企业之间通过 Excel 表格交换数据。一个典型的案例是：某制药企业在国内有超过 800 家的分销机构，包括医院、药店等。这些医院、药店每天都要通过 Excel 表格将销售数据发送给这家制药企业。为了更加迅速地掌握企业的销售情况，这家制药企业需要迅速、准确地将这些数据整合起来。由于表格的数量非常巨大，纯手工的整理已经不可能实现。B2B Data Transformation 是一个比较理想的解决方案，它通过一个图形化的界面进行开发，可以自动读取大量 Excel 表格的数据，并将其解析为 XML 格式的数据，随后这些被转换为 XML 的数据被转发给 PowerCenter，PowerCenter 对其进行清洗、标准化后，使这些数据变成可分析、可统计的有用资源。

除了对如上的非结构化或者半结构化文档进行转换、整理，B2B Data Transformation 内置支持大量的行业标准，以方便用户使用。这些标准包括如下几类。

- ◎ 跨行业。
  - ◆ UN/EDIFACT。
  - ◆ XBRL。
  - ◆ RosettaNet。
  - ◆ COBOL。
- ◎ 医疗保健。
  - ◆ HL7。
  - ◆ HIPAA。
  - ◆ NCPDP。
- ◎ 金融服务。
  - ◆ BAI。
  - ◆ SWIFT (MT and MX)。
  - ◆ UNIFI (ISO 20022)。
  - ◆ NACHA。
  - ◆ SEPA。
  - ◆ FIX, FIXML。
  - ◆ FpML。
  - ◆ Telekurs VDF。
  - ◆ TWIST。
  - ◆ ACORD (AL3 和 XML)。
  - ◆ DTCC。
  - ◆ OAGIS XML。

封面

书名

版权

前言

目录

## 第1章 PowerCenter Hello World世界

  1.1 Informatica Hello World

  1.2 PowerCenter架构和客户端简介

    1.2.1 PowerCenter架构

    1.2.2 PowerCenter客户端

  1.3 PowerCenter Hello World

## 第2章 PowerCenter基础组件

  2.1 Source

    2.1.1 数据库源

    2.1.2 文本文件源

  2.2 Target

    2.2.1 数据库目标

    2.2.2 文本文件目标

  2.3 Expression表达式

    Expression中的变量端口 (Variable Port)

  2.4 Filter

  2.5 Source Qualifier

    2.5.1 Source Qualifier的作用

    2.5.2 数据库数据源的Source Qualifier

    2.5.3 Source Qualifier自定义SQL

    2.5.4 Source Qualifier复杂关联

  2.6 Sorter

  2.7 Joiner

    2.7.1 关联类型

    2.7.2 Sorted Joiner

    2.7.3 Joiner的独特作用

    2.7.4 自关联 (Self-Join)

  2.8 Lookup

    2.8.1 Lookup Caching enabled

    2.8.2 非连接的Lookup

    2.8.3 Lookup SQL Override