

目录

| | |
|------------------------------|----|
| XML 指南 - 版权..... | 3 |
| 前言..... | 6 |
| Informatica 资源..... | 7 |
| XML 概念..... | 8 |
| XML 概念概览..... | 8 |
| XML 文件..... | 8 |
| DTD 文件..... | 11 |
| XML 架构文件..... | 13 |
| XML 元数据的类型..... | 14 |
| 基数..... | 16 |
| 简单和复杂 XML 类型..... | 18 |
| 任何类型的元素和属性..... | 22 |
| 组件组..... | 24 |
| XML 路径..... | 26 |
| 代码页..... | 26 |
| 将 XML 用于 PowerCenter..... | 27 |
| 使用 XML 与 PowerCenter 概览..... | 27 |
| 导入 XML 元数据..... | 28 |
| 了解 XML 视图..... | 33 |
| 了解层次结构关系..... | 34 |
| 了解实体关系..... | 38 |
| 使用循环引用..... | 43 |
| 了解视图行..... | 44 |
| 透视列..... | 46 |
| 使用 XML 源..... | 48 |
| 使用 XML 源概览..... | 48 |
| 导入 XML 源定义..... | 49 |
| 使用 XML 视图..... | 50 |
| 生成实体关系..... | 52 |
| 生成层次结构关系..... | 52 |

| | |
|-----------------------------|-----|
| 创建自定义 XML 视图..... | 52 |
| 同步 XML 定义..... | 54 |
| 编辑 XML 源定义属性..... | 54 |
| 从存储库定义中创建 XML 定义..... | 56 |
| XML 源故障排除..... | 56 |
| 使用 XML 编辑器..... | 58 |
| 使用 XML 编辑器概览..... | 58 |
| 创建和编辑视图..... | 60 |
| 创建 XPath 查询谓词..... | 65 |
| 维护视图关系..... | 68 |
| 查看架构组件..... | 69 |
| 设置 XML 视图选项..... | 72 |
| 对使用 XML 编辑器进行故障排除..... | 80 |
| 使用 XML 目标..... | 80 |
| 使用 XML 目标概览..... | 80 |
| 通过 XML 文件导入一个 XML 目标定义..... | 80 |
| 通过 XML 源定义创建目标..... | 81 |
| 编辑 XML 目标定义属性..... | 82 |
| 验证 XML 目标..... | 83 |
| 在映射中使用的 XML 目标..... | 84 |
| XML 对象故障排除..... | 87 |
| XML 源限定符转换..... | 88 |
| XML 源限定符转换概览..... | 88 |
| 向映射添加一个 XML 源限定符..... | 89 |
| 编辑 XML 源限定符转换..... | 89 |
| 在映射中使用 XML 源限定符..... | 91 |
| XML 源限定符转换故障排除..... | 94 |
| 中游 XML 转换..... | 94 |
| 中游 XML 转换概览..... | 94 |
| XML 解析器转换..... | 95 |
| XML 生成器转换..... | 98 |
| 创建中游 XML 转换..... | 98 |
| 同步中游 XML 定义..... | 99 |
| 编辑中游 XML 转换属性..... | 99 |
| 生成传递端口..... | 102 |
| 中游 XML 转换故障排除..... | 103 |

| | |
|-----------------------|-----|
| XML 数据类型引用..... | 103 |
| XML 和转换数据类型..... | 103 |
| XPath 查询函数参考..... | 106 |
| XPath 查询函数概览..... | 106 |
| 函数快速参考..... | 106 |
| 布尔型..... | 108 |
| ceiling..... | 109 |
| concat..... | 109 |
| contains..... | 110 |
| false..... | 111 |
| floor..... | 112 |
| lang..... | 112 |
| normalize-space..... | 113 |
| not..... | 114 |
| number..... | 114 |
| round..... | 115 |
| starts-with..... | 116 |
| 字符串..... | 116 |
| string-length..... | 117 |
| substring..... | 118 |
| substring-after..... | 119 |
| substring-before..... | 120 |
| translate..... | 121 |
| true..... | 122 |

XML 指南 - 版权

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

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

本软件和/或文档的某些部分受第三方版权制约，包括但不限于：版权所有 DataDirect Technologies。保留所有权利。版权所有 (C) Sun Microsystems。保留所有权利。版权所有 (C) RSA Security Inc. 保留所有权利。版权所有 (C) Ordinal Technology Corp. 保留所有权利。版权所有 (C) Aandacht c.v. 保留所有权利。版权所有 Genivia, Inc. 保留所有权利。版权所有 Isomorphic Software。保留所有权利。版权所有 (C) Meta Integration Technology, Inc. 保留所有权利。版权所有 (C) Intalio。保留所有权利。版权所有 (C) Oracle。保留所有权利。版权所有 (C) Adobe Systems Incorporated。保留所有权利。版权所有 (C) DataArt, Inc. 保留所有权利。版权所有 (C) ComponentSource。保留所有权利。版权所有 (C) Microsoft

Corporation。保留所有权利。版权所有 (C) Rogue Wave Software, Inc. 保留所有权利。版权所有 (C) Teradata Corporation。保留所有权利。版权所有 (C) Yahoo! Inc. 保留所有权利。版权所有 (C) Glyph & Cog, LLC。保留所有权利。版权所有 (C) Thinkmap, Inc. 保留所有权利。版权所有 (C) Clearpace Software Limited。保留所有权利。版权所有 (C) Information Builders, Inc. 保留所有权利。版权所有 (C) OSS Nokalva, Inc. 保留所有权利。版权所有 Edifecs, Inc. 保留所有权利。版权所有 Cleo Communications, Inc. 保留所有权利。版权所有 (C) International Organization for Standardization 1986。保留所有权利。版权所有 (C) ej-technologies GmbH。保留所有权利。版权所有 (C) Jaspersoft Corporation。保留所有权利。版权所有 (C) International Business Machines Corporation。保留所有权利。版权所有 (C) yWorks GmbH。保留所有权利。版权所有 (C) Lucent Technologies。保留所有权利。版权所有 (C) University of Toronto。保留所有权利。版权所有 (C) Daniel Veillard。保留所有权利。版权所有 (C) Unicode, Inc. 版权所有 IBM Corp. 保留所有权利。版权所有 (C) MicroQuill Software Publishing, Inc. 保留所有权利。版权所有 (C) PassMark Software Pty Ltd. 保留所有权利。版权所有 (C) LogiXML, Inc. 保留所有权利。版权所有 (C) 2003-2010 Lorenzi Davide, 保留所有权利。版权所有 (C) Red Hat, Inc. 保留所有权利。版权所有 (C) The Board of Trustees of the Leland Stanford Junior University。保留所有权利。版权所有 (C) EMC Corporation。保留所有权利。版权所有 (C) Flexera Software。保留所有权利。版权所有 (C) Jinfonet Software。保留所有权利。版权所有 (C) Apple Inc. 保留所有权利。版权所有 (C) Telerik Inc. 保留所有权利。版权所有 (C) BEA Systems。保留所有权利。版权所有 (C) PDFlib GmbH。保留所有权利。版权所有 (C) Orientation in Objects GmbH。保留所有权利。版权所有 (C) Tanuki Software, Ltd. 保留所有权利。版权所有 (C) Ricebridge。保留所有权利。版权所有 (C) Sencha, Inc. 保留所有权利。版权所有 (C) Scalable Systems, Inc. 保留所有权利。版权所有 (C) jqWidgets。保留所有权利。版权所有 (C) Tableau Software, Inc. 保留所有权利。版权所有 (C) MaxMind, Inc. 保留所有权利。版权所有 (C) TMate Software s.r.o. 保留所有权利。版权所有 (C) MapR Technologies Inc. 保留所有权利。版权所有 (C) Amazon Corporate LLC。保留所有权利。版权所有 (C) Highsoft。保留所有权利。版权所有 (C) Python Software Foundation。保留所有权利。版权所有 (C) BeOpen.com。保留所有权利。版权所有 (C) CNRI。保留所有权利。

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

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

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

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

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

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

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

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

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

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

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

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

concurrent/intro.html、<https://aws.amazon.com/asl/>、<https://github.com/twbs/bootstrap/blob/master/LICENSE> 和 <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt> 下许可的软件。

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

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

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

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

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

声明

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

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

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

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

前言

*XML 指南*面向负责在数据仓库环境中使用 XML 的开发人员和软件工程师。在使用 *XML 指南*之前，请确保您充分理解 XML 概念、您的环境中的操作系统、平面文件或大型机系统。此外，请确保您了解支持的应用程序的接口要求。

Informatica 资源

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

Informatica Network

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

作为 Informatica Network 成员，您可以选择以下服务：

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

Informatica 知识库

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

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

Informatica 文档

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

如果您对产品信息有任何疑问、意见或建议，请与 Informatica 文档团队联系，电子邮件地址为 infa_documentation@informatica.com。

Informatica 产品可用性矩阵

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

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

Informatica Velocity

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

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

Informatica Marketplace

Informatica Marketplace 是一个论坛，该论坛中提供的解决方案可扩展和增强您的 Informatica 实施。利用 Informatica 开发人员和合作伙伴在 Marketplace 中提供的数以百计的解决方案，可提高您的工作效率并

加快项目实施时间。您可以在以下网址中找到 Informatica Marketplace：
<https://marketplace.informatica.com>。

Informatica 全球客户支持部门

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

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

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

XML 概念

XML 概念概览

可扩展标记语言 (XML) 是一种灵活的方式，可用于创建通用信息格式以及在应用程序之间和 Internet 上共享格式和数据。

可以从以下文件类型将 XML 定义导入到 PowerCenter®：

- **XML 文件。**XML 文件包含数据和元数据。XML 文件可以引用文档类型定义 (DTD) 文件或 XML 架构定义 (XSD) 进行验证。
- **DTD 文件。**DTD 文件定义 XML 文件中的元素类型、属性和实体。DTD 文件对 XML 文件结构带来了某些约束，但 DTD 文件不包含任何数据。
- **XML 架构。**XML 架构定义元素、属性和类型定义。架构包含简单类型和复杂类型。简单类型是包含文本的 XML 元素或属性。复杂类型是包含其他元素和属性的 XML 元素。

架构支持元素、属性和置换组，您可以在整个架构中进行引用。使用置换组可将 XML 实例文档中的一个元素置换为另一个元素。架构还支持继承元素、复杂类型以及元素和属性组。

XML 文件

XML 文件包含的标记可标识 XML 文件中的数据，但不能标识数据格式。XML 文件的基本组件是元素。XML 元素包括元素开始标记、元素内容和元素结束标记。所有 XML 文件必须包含由文件顶部和底部的单个标记定义的根元素。根元素包括文件中的所有其他元素。

XML 文件可对层次结构数据库建模。元素在 XML 层次结构中的位置表示其与其他元素之间的关系。元素可以包含子元素，并且可以从其他元素继承特性。

例如，以下 XML 文件描述了一本书：

```
<book>
  <title>Fun with XML</title>
  <chapter>
    <heading>Understanding XML</heading>
    <heading>Using XML</heading>
  </chapter>
  <chapter>
    <heading>Using DTD Files</heading>
    <heading>Fun with Schemas</heading>
  </chapter>
</book>
```



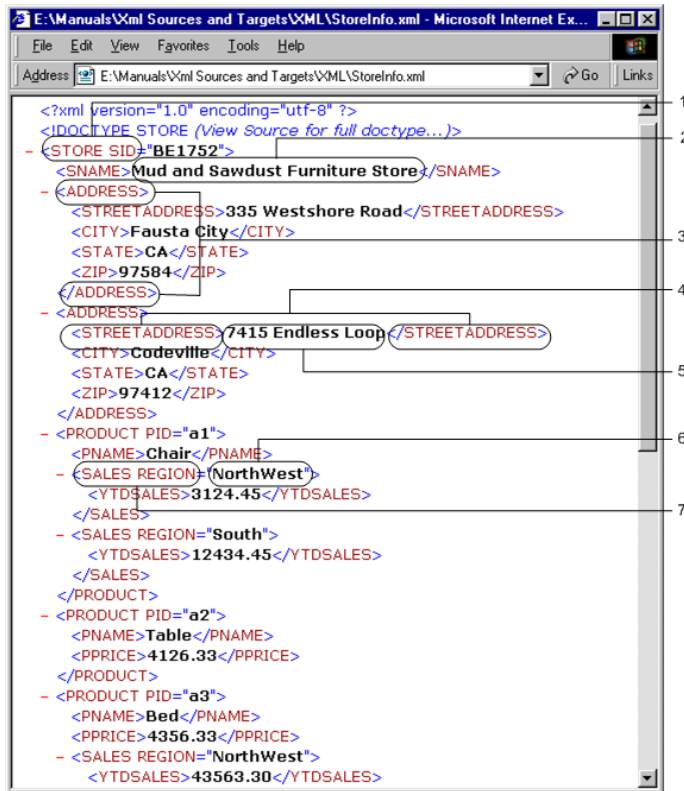
```
</chapter>
</book>
```

Book 是根元素，它包含 title 和 chapter 元素。Book 是 title 和 chapter 的父元素，chapter 是 heading 的父元素。Title 和 chapter 是同级元素，因为它们具有相同的父元素。

元素可具有提供有关元素的额外信息的属性。在以下示例中，属性 graphic_type 描述了图片的内容：

```
<picture graphic_type="gif">computer.gif</picture>
```

下图显示了 XML 文件中的结构、元素和属性：



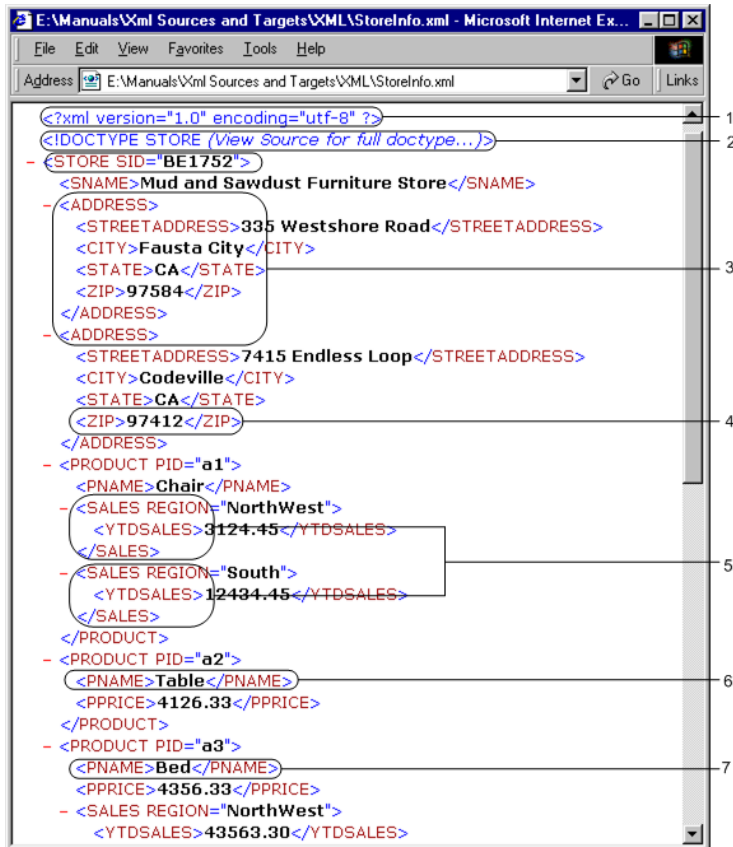
1. 根元素。
2. 元素数据。
3. Enclosure 元素。
4. 元素标记。
5. 元素数据。
6. 属性值。
7. 属性标记。

XML 文件具有结构层次。XML 层次结构包含以下元素：

- **子元素。** 包含在另一个元素内的元素。
- **Enclosure 元素。** 包含其他元素但不包含数据的元素。可以包含其他 enclosure 元素的 enclosure 元素。
- **全局元素。** 根元素的直接子元素。您可以在整个 XML 架构中引用全局元素。
- **叶元素。** 不包含其他元素的元素。叶元素是 XML 层次结构中最低级别的元素。

- **局部元素。** 嵌套在另一个元素中的元素。您可以仅在父元素的上下文内引用局部元素。
- **多次出现的元素。** 在其父元素内出现多次的元素。Enclosure 元素可以是多次出现的元素。
- **父链。** 跟踪从元素到根的路径的子-父元素的继承。
- **父元素。** 包含其他元素的元素。
- **单次出现的元素。** 在其父元素内出现一次的元素。

下图显示了 XML 层次结构中的某些元素：



1. 编码属性标识代码页。
2. DOCTYPE 标识关联的 DTD 文件。
3. Enclosure 元素：元素 Address 包括元素 StreetAddress、City、State 和 Zip。元素 Address 也是一个父元素。
4. 叶元素：元素 Zip 及其所有同级元素是元素 Address 内最低级别的元素。
5. 多次出现的元素：元素 Sales Region 在元素 Product 内出现多次。
6. 单次出现的元素：元素 PName 在元素 Product 内出现一次。
7. 子元素：元素 PName 是 Product 的子元素，Product 是 Store 的子元素。

使用 DTD 或架构验证 XML 文件

有效的 XML 文件符合关联 DTD 或架构文件的结构。

要引用 DTD 文件的位置和名称，请使用 XML 文件中的 DOCTYPE 声明。此外，DOCTYPE 声明还命名 XML 文件的根元素。

例如，以下 XML 文件引用了 note.dtd 文件的位置：

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">
<note>
  <body>XML Data</body>
</note>
```

要引用架构，请使用 schemaLocation 声明。schemaLocation 包含架构的位置和名称。

以下 XML 文件引用了外部位置中的 note.xsd 架构：

```
<?xml version="1.0"?>
<note xsi:SchemaLocation="http://www.w3schools.com note.xsd">
  <body>XML Data</body>
</note>
```

Unicode 编码

XML 文件包含指示文件中的代码页的编码属性。最常见的编码是 UTF-8 和 UTF-16。UTF-8 表示包含 1 到 4 个字节的字符，具体取决于 Unicode 符号。UTF-16 将字符表示为 16 位字。

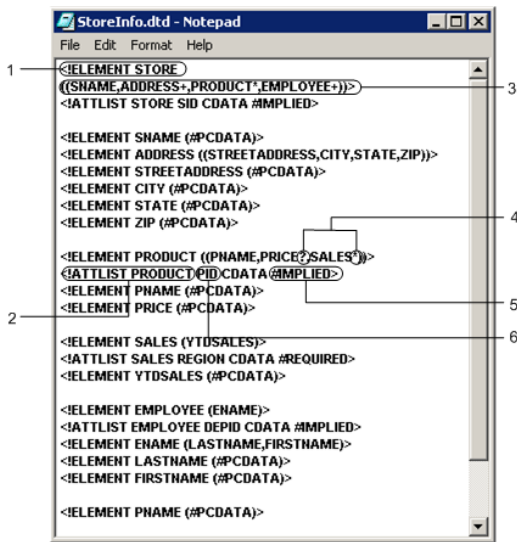
以下示例显示了 XML 文件中的 UTF-8 属性：

```
<?xml version="1.0" encoding="UTF-16LE"?>
<note xsi:SchemaLocation="http://www.w3schools.com note.xsd">
  <body>XML Data</body>
</note>
```

DTD 文件

文档类型定义 (DTD) 文件定义 XML 文件中的元素类型和属性。此外，DTD 文件还对 XML 文件结构带来一些约束。DTD 文件不包含任何数据或元素数据类型。

下图显示了在 DTD 文件中的元素和属性：



1. 元素
2. 属性
3. 元素列表
4. 元素出现次数
5. 属性值选项
6. 属性名称

DTD 元素

在 DTD 文件中，元素声明定义了 XML 元素。元素声明具有以下语法：

```
<!ELEMENT product (#PCDATA)>
```

DTD 说明定义 XML 标记 <product>。说明 (#PCDATA) 指定已解析字符数据。已解析数据是 XML 元素的开始标记和结束标记之间的文本。已解析字符数据是不含子元素的文本。

以下示例显示了包含两个子元素的元素的 DTD 说明：

```
<!ELEMENT boat (brand, type) >
<!ELEMENT brand (#PCDATA) >
<!ELEMENT type (#PCDATA) >
```

Brand 和 type 是 boat 的子元素。每个子元素都可以包含字符。在本示例中，brand 和 type 可以在元素 boat 内部出现一次。以下 DTD 说明指定 brand 必须针对 boat 出现一次或多次：

```
<!ELEMENT boat (brand+) >
```

DTD 属性

属性提供有关元素的其他信息。在 DTD 文件中，属性出现在元素的开始标记内。

以下语法描述了 DTD 文件中的属性：

```
<!ATTLIST element_name attribute_name attribute_type "default_value" >
```

以下参数标识了 DTD 文件中的属性：

- **Element_name**。具有该属性的元素的名称。
- **Attribute_name**。属性的名称。
- **Attribute_type**。属性的类型。最常见的属性类型是 CDATA。CDATA 属性是字符数据。
- **Default_value**。XML 文件中没有出现属性值时的属性值。

使用具有默认值的以下选项：

- **#REQUIRED**。XML 文件必须包含属性值。
- **#IMPLIED**。属性值是可选值。
- **#FIXED**。XML 文件必须包含 DTD 文件中的默认值。有效的 XML 文件可以包含与 DTD 相同的属性值，或 XML 文件可以不含任何属性值。必须使用此选项指定默认值。

以下示例显示了具有固定值的属性：

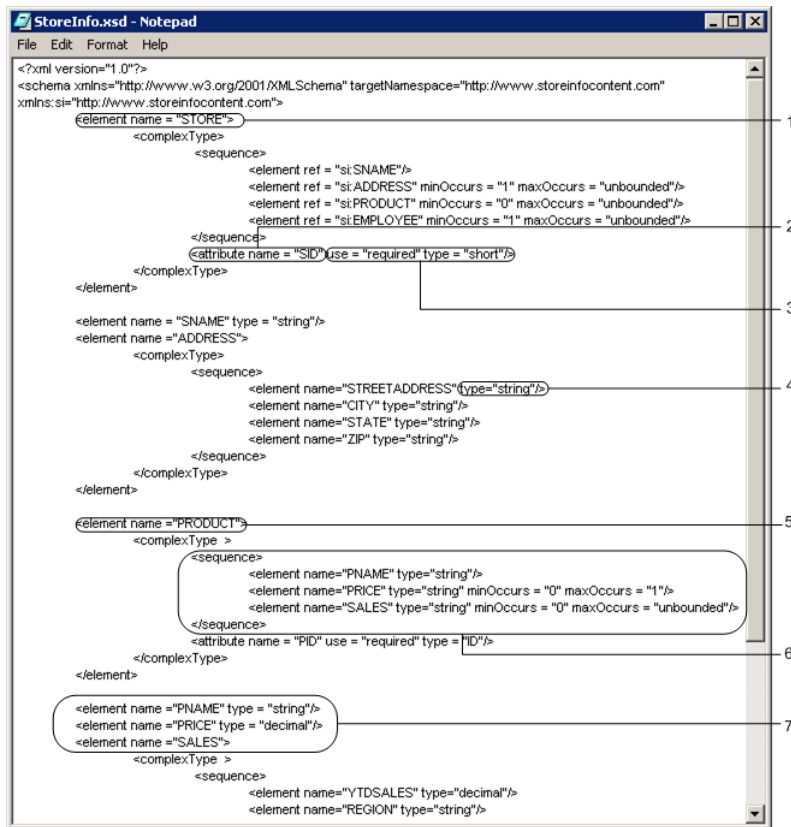
```
<!ATTLIST product product_name CDATA #FIXED "vacuum" >
```

元素名称为 product。属性为 product_name。属性的默认值为 vacuum。

XML 架构文件

XML 架构是定义 XML 文件的有效内容的文档。XML 架构文件与 DTD 文件类似，仅包含元数据。XML 架构定义关联 XML 文件的元素和属性的结构和类型。使用架构定义 XML 文件时，可以限制数据、定义数据格式以及在各种数据类型之间转换数据。XML 架构支持复杂类型和类型之间的继承。架构提供了一种指定元素和属性组、ANY 内容和循环引用的方式。

下图显示了 XML 架构组件：



1. 元素名称。
2. 属性
3. 属性类型和空构造
4. 元素数据类型
5. 元素数据
6. 元素列表和出现次数
7. 元素列表和数据类型

相关主题：

- [“简单和复杂 XML 类型” 页面上 18](#)
- [“组件组” 页面上 24](#)

XML 元数据的类型

您可以从 XML、DTD 或 XML 架构文件创建 PowerCenter XML 定义。XML 文件提供数据和元数据。DTD 文件和 XML 架构文件提供元数据。

PowerCenter 从 XML、DTD 和 XML 架构文件提取以下类型的元数据：

- **命名空间。**元素和属性名称的集合，由 XML 文件中的统一资源标识符 (URI) 引用标识。命名空间可区分来自不同源的元素。

- **名称。**包含元素或属性名称的标记。
- **层次结构。**元素在与 XML 文件中其他元素的关系中所处的位置。
- **基数。**元素在 XML 文件中出现的次数。
- **Datatype.**数据元素的分类，如数值、字符串、布尔或时间。XML 支持自定义数据类型和继承。

命名空间

命名空间包含用于标识架构位置的 URI。URI 是标识 Internet 资源的字符串。URI 是 URL 的抽象。URL 用于定位资源，但 URI 用于标识资源。DTD 或架构文件并不一定存在于 URI 位置。

XML 命名空间标识元素组。命名空间可以标识来自不同 XML 文件的元素和属性或区分不同元素的含义。例如，可以通过声明不同的命名空间（如 *math:table* 和 *furniture:table*）区分元素“table”的含义。XML 区分大小写。命名空间 *Math:table* 与命名空间 *math:table* 不同。

您可以在 XML 文件的根级别声明命名空间，或者可以在 XML 结构中的任何元素内部声明命名空间。在同一 XML 文件中声明多个命名空间时，需要使用命名空间前缀将元素与命名空间相关联。命名空间声明在 XML 文件中显示为以 xmlns 开头的属性。声明具有 xmlns 属性的命名空间前缀。您可以创建任意长度的前缀名称。

以下示例显示了 XML 实例文档中的两个命名空间：

```
<example>
  xmlns:math = "http://www.mathtables.com"
  xmlns:furniture = "http://www.home.com" >
  <math:table>4X6</math:table>
  <furniture:table>Brueners </furniture:table>
</example>
```

一个命名空间具有 math 元素，另一个命名空间具有 furniture 元素。每个命名空间都具有名为“table”的元素，但是这些元素包含的数据类型不同。命名空间前缀可区分 math 表和 furniture 表。

以下文本显示了常见的架构声明：

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3XML.com"
  xmlns="http://www.w3XML.com"
  elementFormDefault="qualified">...
...</xs:schema>
```

下表介绍了命名空间声明的每个部分：

| 架构声明 | 说明 |
|---|--|
| xmlns:xs="http://www.w3.org/2001/XMLSchema" | 包含本地 XML 架构和数据类型的命名空间。在本示例中，每个架构组件都具有前缀“xs”。 |
| targetNamespace="http://www.w3XML.com" | 包含架构的命名空间。 |
| xmlns="http://www.w3XML.com" | 默认命名空间声明。架构中没有属于默认命名空间的前缀的所有元素。通过使用没有前缀的 xmlns 属性声明默认命名空间。 |
| elementFormDefault="qualified" | 指定架构中的任何元素都必须在 XML 文件中具有命名空间。 |

名称

在 XML 文件中，每个标记都是元素或属性的名称。在 DTD 文件中，标记 <!ELEMENT> 指定元素的名称，标记 <!ATTLIST> 指示元素的属性集。在架构文件中，<元素名称> 指定元素的名称，<属性名称> 指定属性的名称。

默认情况下，导入 XML 定义时，元素标记将成为 PowerCenter 定义中的列名称。

层次结构

XML 文件可对层次结构数据库建模。元素在 XML 层次结构中的位置表示其与其他元素之间的关系。例如，元素可以包含子元素，并且可以继承其他元素的特性。

基数

DTD 或架构文件中的元素基数是元素在 XML 文件中出现的次数。元素基数影响您在 XML 定义中构造组的方式。元素的绝对基数和相对基数影响 XML 定义的结构。

绝对基数

元素的绝对基数是元素在 XML 层次结构中其父元素内出现的次数。DTD 和 XML 架构文件描述了元素在层次结构中的绝对基数。DTD 文件使用符号，XML 架构文件则使用 <minOccurs>和 <maxOccurs> 属性来描述元素的绝对基数。

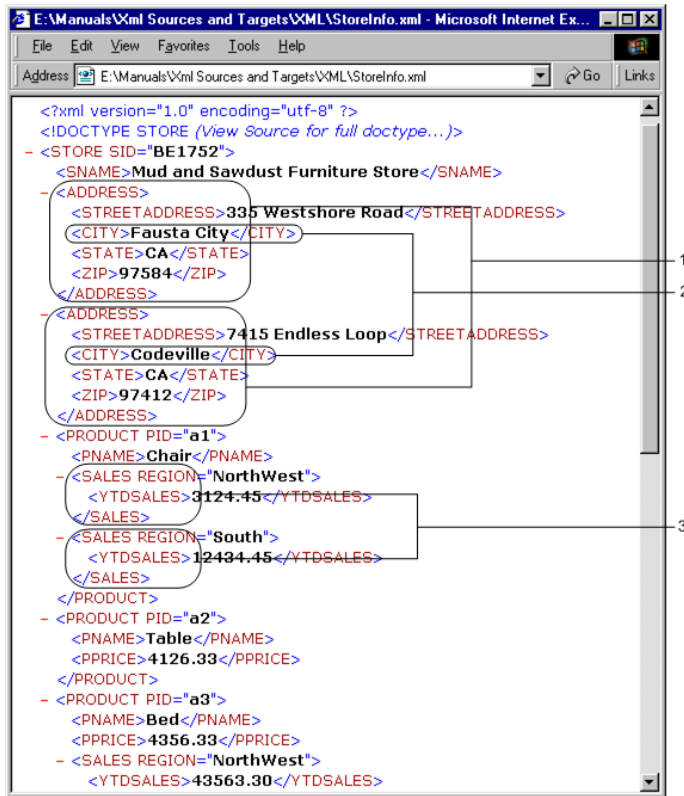
例如，如果某个元素在其父元素中出现一次，则该元素的绝对基数为一次 (1)。但是，如果父元素的基数为一次或多次 (+)，则该元素可能会在 XML 层次结构中出现多次。

元素的绝对基数用于确定其空约束。绝对基数为一次或多次 (+) 的元素不能具有空值，但绝对基数为零次或多次 (*) 的元素可以具有空值。在 XML 架构或 DTD 文件中标记为固定或必需的属性不能具有空值，但隐式属性可以具有空值。

下表介绍了 DTD 和 XML 架构文件如何表示基数：

| 绝对基数 | DTD | 架构 |
|-------------------------------|-----|--|
| 零或一次 | ? | minOccurs=0 maxOccurs=1 |
| 零次、一次或多次 | * | minOccurs=0 maxOccurs=unbounded minOccurs=0 maxOccurs=n |
| 一次 | - | minOccurs=1 maxOccurs=1 |
| 一次或多次 | + | minOccurs=1 maxOccurs=unbounded minOccurs=1 maxOccurs=n |
| 注意: 您可以声明在架构中出现的最大次数或无限的出现次数。 | | |

下图显示了示例 XML 文件中元素的绝对基数：



1. 元素 `Address` 在 `Store` 内出现多次。它的绝对基数是一次或多次(+).
2. 元素 `City` 在其父元素 `Address` 内出现一次。它的绝对基数是一次(1).
3. 元素 `Sales` 在其父元素 `Product` 内出现零次或多次。它的绝对基数是零次或多次(*)。

相对基数

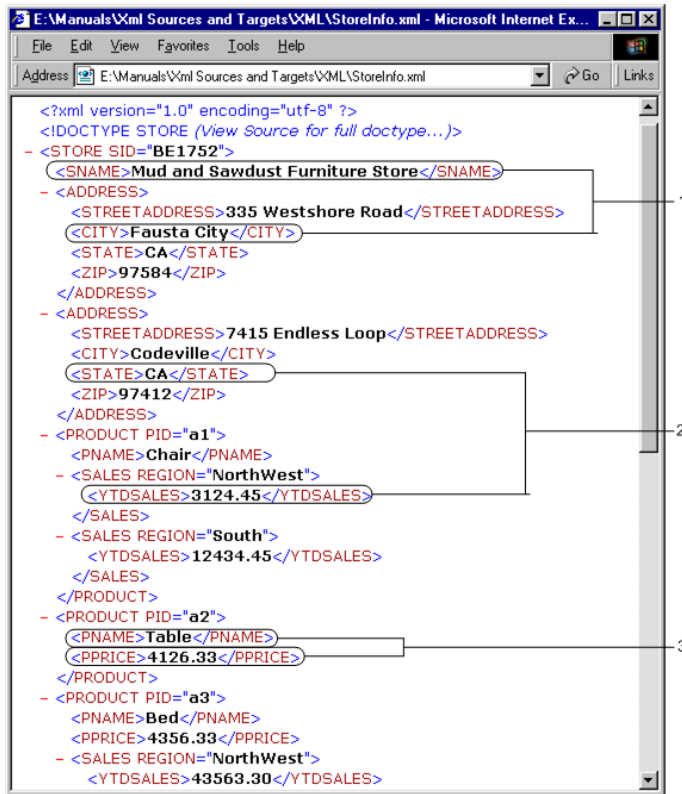
相对基数是元素与 XML 层次结构中另一个元素的关系。元素与层次结构中的另一个元素之间可以是一对一关系、一对多关系或多对多关系。

如果每次出现一个元素时，另一个元素都会出现一次，则该元素与另一个元素具有一对一关系。例如，员工元素可以有一个社会保障号元素。员工和社会保障号具有一对一关系。

如果每次出现一个元素时，另一个元素都会出现多次，则该元素与另一个元素具有一对多关系。例如，员工元素可以有多个电子邮件地址。员工和电子邮件地址具有一对多关系。

如果 XML 文件的两个元素都可以多次出现，则一个元素与另一个元素具有多对多关系。例如，员工可能有多个电子邮件地址和多个街道地址。电子邮件地址和街道地址具有多对多关系。

下图显示了示例 XML 文件中元素之间的相对基数：



1. 一对多关系。每次出现 SNAME 时，ADDRESS 都会出现多次，因此，CITY 也会出现多次。
2. 多对多关系。每次出现 STATE 时，YTDSALES 都会出现多次。每次出现 YTDSALES 时，每个出现的过程中，STATE 都会出现多次。
3. 一对一关系。每次出现 PNAME 时，PPRICE 都会出现一次。

简单和复杂 XML 类型

XML 架构语言拥有超过 40 种内建数据类型，包括数值、字符串、时间、XML 和二进制。这些数据类型称为简单类型。它们包含文本，但不包含其他元素和属性。您可以从基本的 XML 简单类型派生新的简单类型。

您可以创建复杂 XML 数据类型。复杂数据类型是包含多种简单类型的数据类型。复杂数据类型还可以包含其他复杂类型和属性。

有关 XML 数据类型的详细信息，请参阅 XML 数据类型的 W3C 规范，网址为：

<http://www.w3.org/TR/xmlschema-2>。

简单类型

简单数据类型是包含文本的 XML 元素或属性。简单类型是不可分割的。简单类型不能具有属性，但属性是简单类型。

PowerCenter 支持以下简单类型：

- **原子类型。**基本数据类型，例如布尔、字符串或整数。
- **列表。**原子类型的数组集合。

- **联合**。映射到 XML 文件中的简单类型的一个或多个原子或列表类型的组合。

原子类型

原子数据类型是一个基本数据类型，例如布尔、字符串、整数、十进制或日期。要定义自定义原子数据类型，请对原子数据类型添加限制，以限制内容。使用方面可以定义要限制或允许哪些值。

方面是定义最小值或最大值、特定值或有效值的数据模式的表达式。例如，pattern 方面可将元素限制为数值的表达式。enumeration 方面可列出元素的合法值。

以下示例包含的 pattern 方面将元素限制为 a 到 z 之间的小写字母：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType></xs:element>
```

以下示例包含的 enumeration 方面将字符串限制为 a、b 或 c：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="a"/>
      <xs:enumeration value=" b" />
      <xs:enumeration value=" c" />
    </xs:restriction>
  </xs:simpleType></xs:element>
```

列表

列表是原子类型的数组集合，例如，表示名称的字符串列表。列表 itemType 定义了列表组件的数据类型。

以下示例显示了名为 names 的列表：

```
<xs:simpleType name="names">
  <xs:list itemType="xs:string" />
</xs:simpleType>
```

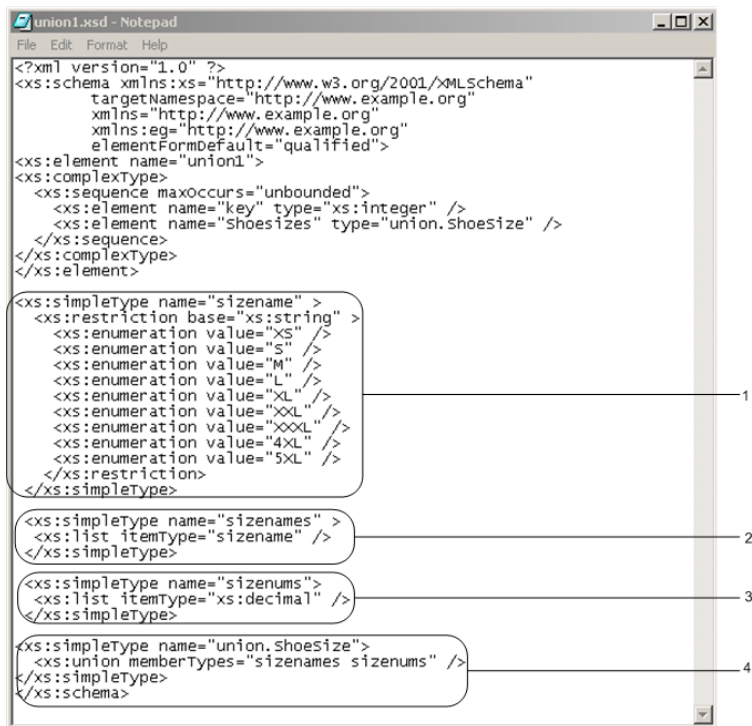
XML 文件可能包含 names 列表中的以下数据：

```
<names>Joe Bob Harry Atlee Will</names>
```

联合

联合是映射到 XML 文件中的简单类型的一个或多个原子或列表类型的组合。定义联合类型时，可以指定要组合的类型。例如，可以创建名为 size 的类型。Size 可以包含字符串数据，如 S、M 和 L，或者 size 可能包含十进制大小，如 30、32 和 34。如果定义联合类型的元素，XML 文件可以包含字符串大小的 sizeName 类型和数值大小的 sizeNum 类型。

下图显示了包含 shoesize 联合的架构文件，该联合包含 sizenames 和 sizenums 列表：



1. SizeName 是受限制字符串类型。
2. Sizenames 类型接受字符串列表。
3. Sizenums 类型接受十进制列表。
4. Shoesize 联合接受十进制和字符串列表。

联合将 sizenames 和 sizenums 定义为联合成员类型。Sizenames 定义字符串值列表。Sizenums 定义十进制值列表。

复杂类型

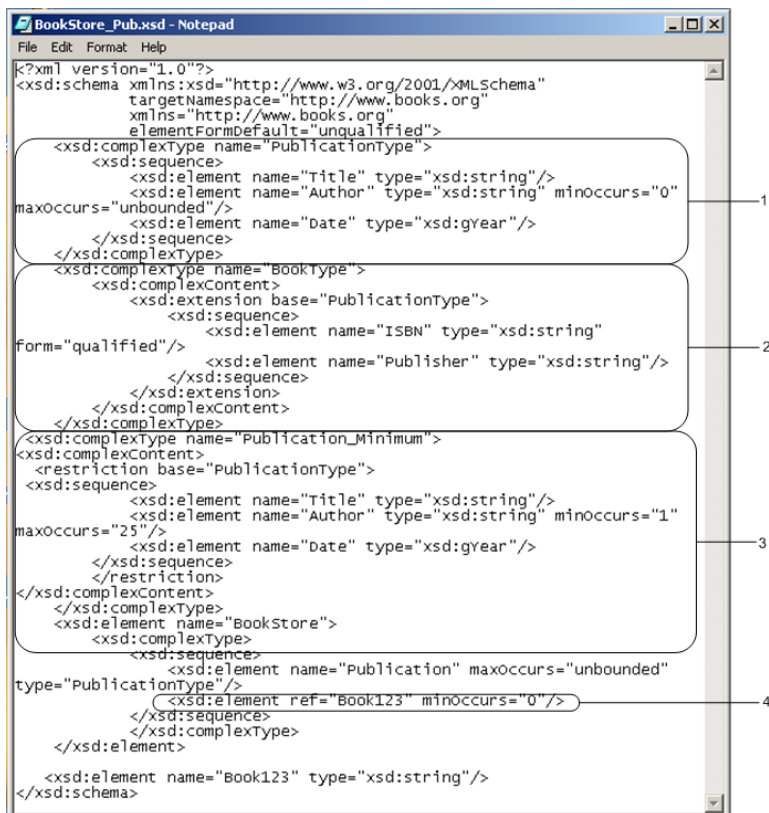
复杂类型可将简单类型的集合汇总到逻辑单元。例如，客户类型可能包括客户编号、名称、街道地址、乡镇、城市和邮政编码。复杂类型还可以引用其他复杂类型或元素和属性组。

XML 支持复杂类型继承。定义复杂类型时，可以创建集成基本类型组件的其他复杂类型。在类型关系中，基本类型是指派生了其他类型的复杂类型。派生的复杂类型继承基本类型中的元素。

扩展复杂类型是指继承了基本类型中的元素并包含其他元素的派生类型。例如，customer_purchases 类型可能会从客户复杂类型继承其定义，但 customer_purchases 类型添加了项目、成本和 date_sold 元素。

受限复杂类型是指限制基本类型中的部分元素的派生类型。例如，mail_list 可能会继承客户的元素，但会通过将 minOccurs 和 maxOccurs 边界设置为零限制 phone_number 元素。

下图显示了限制和扩展基本复杂类型的派生复杂类型：



1. 基本复杂类型
2. 扩展复杂类型
3. 受限制复杂类型
4. 元素引用

在上图中，基本类型是 PublicationType。BookType 扩展了 PublicationType，并包括 ISBN 和出版商元素。Publication_Minimum 限制了 PublicationType。Publication_Minimum 需要 1 到 25 个作者，并将日期限制到年。

抽象元素

有时，架构包含定义复杂元素的基本结构，但不包含所有组件。派生的复杂类型使用更多组件扩展基本类型。由于基本类型不是一个完整的定义，您可能不希望在 XML 文件中使用基本类型。您可以声明基本类型为抽象元素。抽象元素在 XML 文件中无效。只有派生元素是有效的。

要定义抽象元素，请添加值为“true”的抽象属性。默认值为 false。

例如，PublicationType 是一个抽象元素。BookType 继承了 PublicationType 中的元素，但还包括 ISBN 和出版商元素。由于 PublicationType 是抽象元素，因此 PublicationType 元素在 XML 文件中无效。XML 文件可以包含派生类型 BookType。

以下架构包含 PublicationType 和 BookType：

```
<xsd:complexType name="PublicationType" abstract="true">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
```

```

        <xsd:element name="Author" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookType">
    <xsd:complexContent>
        <xsd:extension base="PublicationType" >
            <xsd:sequence>
                <xsd:element name="ISBN" type="xsd:string"/>
                <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

任何类型的元素和属性

部分架构元素和属性允许在 XML 文件中使用任何类型的数据。当您需验证具有未标识的元素和属性类型的 XML 文件时，请使用这些元素和属性。

使用以下元素和属性，它们允许使用任意类型的数据：

- **anyType 元素**。允许关联 XML 文件中的元素属于任意数据类型。
- **anySimpleType 元素**。允许关联 XML 文件中的元素属于任意 simpleType。
- **ANY 内容元素**。允许元素属于已在架构中定义的任何元素。
- **anyAttribute 属性**。允许元素属于已在架构中定义的任何属性。

anyType 元素

anyType 元素可以是 XML 实例文档中的任何数据类型。如果元素包含不同类型的数据，则可以声明该元素属于 anyType。

以下架构描述了人员及名字、姓氏和属于 anyType 的 age 元素：

```

<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
            <<xs:element name="age" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

以下 XML 实例文档包括日期类型和 age 元素中的数字：

```

<person>
    <firstname>Danny</firstname>
    <lastname>Russell</lastname>
    <age>1959-03-03</age>
</person>
<person>
    <firstname>Carla</firstname>
    <lastname>Havers</lastname>
    <age>46</age>
</person>

```

这两种类型对架构均有效。如果不声明架构中元素的数据类型，则在将架构导入 Designer 时，该元素默认为 anyType。

anySimpleType 元素

anySimpleType 元素可以包含任何原子类型。原子类型是一个基本数据类型，例如布尔、字符串、整数、十进制或日期。

以下架构描述了人员及名字、姓氏和属于 anySimpleType 的其他元素：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="other" type="xs:anySimpleType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

以下 XML 实例文档将 anySimpleType 元素置换为字符串数据类型：

```
<person>
  <firstname>Kathy</firstname>
  <lastname>Russell</lastname>
  <other>Cissy</other>
</person>
```

以下 XML 实例文档将 anySimpleType 元素置换为数值数据类型：

```
<person>
  <firstname>Kathy</firstname>
  <lastname>Russell</lastname>
  <other>34</other>
</person>
```

ANY 内容元素

ANY 内容元素接受 XML 文件中的任何内容。声明架构中的 ANY 内容元素时，可以将其置换为 XML 实例文档中任意名称和类型的元素。架构中必须存在置换元素。

指定 ANY 内容时，可以使用关键字 ANY，而非元素名称和元素类型。

以下架构描述了人员及名字、姓氏和属于 ANY 内容的元素：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="son" type="xs:string"/>
<xs:element name="daughter" type="xs:string"/>
```

该架构包括 son 元素和 daughter 元素。您可以置换 XML 实例文档中 son 或 daughter 元素的 ANY 元素：

```
<person>
  <firstname>Danny</firstname>
  <lastname>Russell</lastname>
```

```

    <son>Atlee</son>
  </person>
</person>
<person>
  <firstname>Christine</firstname>
  <lastname>Slade</lastname>
  <daughter>Susie</daughter>
</person>

```

AnyAttribute 属性

anyAttribute 属性接受 XML 文件中的任何属性。将属性声明为 anyAttribute 后，可以置换架构中任何属性的 anyAttribute 元素。

以下架构描述了人员及名字、姓氏和属于 anyAttribute 的属性：

```

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>

```

以下架构包括性别属性：

```

<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

以下 XML 实例文档将 anyAttribute 置换为性别属性：

```

<person gender="female">
  <firstname>Anita</firstname>
  <lastname>Ficks</lastname>
</person>
<person gender="male">
  <firstname>Jim</firstname>
  <lastname>Geimer</lastname>
</person>

```

组件组

可以在 XML 架构中创建以下组件组：

- **元素和属性组。**可以在整个架构中引用的元素或属性组。
- **置换组。**可以使用同一组中的其他元素置换的元素组。

元素和属性组

您可以将元素和属性置于可在架构中引用的组中。在引用组之前，必须声明元素或属性组。

以下示例显示了元素组的架构语法：

```
<xs:group name="Songs">
  <xs:element name="songTitle" type="xs:string" />
  <xs:element name="artist" type="xs:string" />
  <xs:element name="publisher" type="xs:string" />
</xs:group>
```

以下示例显示了属性组的架构语法：

```
<xs:attributeGroup name="Songs">
  <xs:attribute name="songTitle" type="xs:string" />
  <xs:attribute name="artist" type="xs:string" />
  <xs:attribute name="publisher" type="xs:string" />
</xs:attributeGroup>
```

以下元素组对 XML 数据带来了约束：

- **序列组。**XML 文件中的所有元素必须按照架构列出它们的顺序出现。例如，OrderHeader 首先需要 customerName，然后依次是 orderNumber 和 orderDate：

```
<xs:group name="OrderHeader">
  <xs:sequence>
    <xs:element name="customerName" type="xs:string" />
    <xs:element name="orderNumber" type="xs:number" />
    <xs:element name="orderDate" type="xs:date" />
  </xs:sequence>
</xs:group>
```

- **选项组。**组中的一个元素可以出现在 XML 文件中。例如，CustomerInfo 组列出了 XML 文件的元素选项：

```
<xs:group name="CustomerInfo">
  <xs:choice>
    <xs:element name="customerName" type="xs:string" />
    <xs:element name="customerID" type="xs:number" />
    <xs:element name="customerNumber" type="xs:integer" />
  </xs:choice>
</xs:group>
```

- **全部组。**所有元素必须全部或一个都不出现在 XML 文件中。元素可以按任意顺序出现。例如，CustomerInfo 需要全部三个元素或一个也不需要：

```
<xs:group name="CustomerInfo">
  <xs:all>
    <xs:element name="customerName" type="xs:string" />
    <xs:element name="customerAddress" type="xs:string" />
    <xs:element name="customerPhone" type="xs:string" />
  </xs:all>
</xs:group>
```

置换组

使用置换组可将 XML 文件中中的一个元素替换为另一个元素。例如，如果您的地址来自加拿大和美国，则可以以为加拿大创建一种地址类型，为美国创建另一种类型。您可以创建接受两种地址类型的置换组。

以下架构片段显示了 Address 基本类型以及派生类型 CAN_Address 和 USA_Address：

```
<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Street" type="xs:string"
      minOccurs="1" maxOccurs="3" />
  </xs:sequence>
</xs:complexType>
```

```

        <xs:element name="City" type="xs:string" />
    </xs:sequence>
</xs:complexType>
<xs:element name="MailAddress" type="Address" />
<xs:complexType name="CAN_Address">
    <xs:complexContent>
        <xs:extension base="Address">
            <xs:sequence>
                <xs:element name="Province" type="xs:string" />
                <xs:element name="PostalCode" type="CAN_PostalCode"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="USA_Address">
    <xs:complexContent>
        <xs:extension base="Address">
            <xs:sequence>
                <xs:element name="State" type="USPS_StateCode" />
                <xs:element name="ZIP" type="USPS_ZIP"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:element name="AddrCAN" type="CAN_Address"
substitutionGroup="MailAddress"/>
<xs:element name="AddrUSA" type="USA_Address"
substitutionGroup="MailAddress"/>

```

CAN_Address 包括 Province 和 PostalCode，USA_Address 包括 State 和 Zip。MailAddress 置换组包括两个地址类型。

相关主题：

- [“在 XML 定义中使用置换组” 页面上 42](#)

XML 路径

XMLPath (XPath) 是一种语言，描述了一种在 XML 文件中查找项目的方法。XPath 使用基于路由（通过从根到元素或属性的层次结构）的寻址语法。XML 路径可以包含长架构组件名称。

XPath 使用斜杠 (/) 来区分层次结构中的元素。在 XPath 中，XML 属性前面有 “@”。

可以针对元素或属性 XPath 创建查询，以筛选 XML 数据。

相关主题：

- [“使用 XPath 查询谓词” 页面上 45](#)

代码页

XML 文件包含指示文件中使用的代码页的编码声明。XML 中最常见的代码页是 UTF-8 和 UTF-16。所有 XML 解析器都支持这些代码页。有关 XML 字符编码规范的信息，请参见 W3C 网站，网址为：

<http://www.w3c.org>。

PowerCenter 支持的 XML 文件的代码页集与它所支持的关系数据库和其他平面文件的代码页集相同。PowerCenter 不支持用户定义的代码页。

当您创建 XML 源或目标定义时，Designer 会为定义分配 PowerCenter 客户端代码页。如果您导入包含代码页分配的 XML 架构，XML 向导将显示架构的代码页。但是，XML 向导不会将该代码页应用于在存储库中创建的 XML 定义。

您无法配置 XML 源定义的代码页。集成服务会在解析 XML 源文件时将其转换为 Unicode。

您可以在 Designer 中配置目标 XML 定义的代码页。还可以在会话属性中更改 XML 目标实例的代码页。

将 XML 用于 PowerCenter

使用 XML 与 PowerCenter 概览

您可以在 PowerCenter 中从 XML 文件、DTD 文件、XML 架构、平面文件定义或关系表定义创建 XML 定义。创建 XML 定义时，Designer 将提取 XML 元数据并在存储库中创建架构。通过该架构提供的结构，您可以编辑和验证 XML 定义。

XML 定义可以包含多个组。在 XML 定义中，组称为视图。XML 层次结构中元素之间的关系定义了视图之间的关系。

创建 XML 定义时，默认情况下 Designer 将为架构中多次出现的元素和复杂类型创建视图。XML 层次结构中元素的相对基数会影响 PowerCenter 在 XML 定义中创建视图的方式。相对基数确定元素是否可以属于同一视图。

Designer 根据键定义 XML 定义中视图之间的关系。源定义不需要键，但目标视图必须具有键。每个视图都有一个作为 XML 元素或生成的键的主键。

创建 XML 定义时，可以创建 XML 数据的层次结构模型或实体关系模型。创建层次结构模型时，可以创建规范化或非规范化层次结构。规范化层次结构包含多次出现的元素的单独视图。非规范化层次结构具有一个视图，其中包含多次出现的元素的重复数据。

如果创建实体模型，Designer 将为复杂类型和多次出现的元素创建视图。Designer 创建的 XML 定义可对架构提供的继承和循环关系进行建模。

PowerCenter 可以使用少于 400 个元素的 XML 架构。PowerCenter 配置文件最多可以包含三个层次结构级别，并且可以包含以下复杂类型的元素：

- 序列
- 任意
- 选项

PowerCenter XML 导入向导最多可以创建 400 个视图。

限制

以下限制适用于在 PowerCenter 中处理 XML：

- XML 架构必须小于 400 个元素。
- XML 架构文件必须小于 100 KB。

- XML 文件大小必须是 10 MB 或更小。
- 复杂性配置文件仅限于三个层次结构级别。
- XML 导入向导最多可以创建 400 个视图。

PowerCenter 不支持以下功能：

- **连接列。**列不能由两个元素连接而成。例如，不能创建引用 FIRSTNAME 和 LASTNAME 两个元素的连接元素的列 FULLNAME。
- **复合键。**键不能由两个元素连接而成。例如，不能创建引用 LASTNAME 和 PHONENUMBER 两个元素的连接元素的键 CUSTOMERID。
- **已解析列表。**PowerCenter 将列表类型存储为一个包含所有数组元素的字符串。PowerCenter 不从字符串解析各个简单类型。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

导入 XML 元数据

导入 XML 定义时，Designer 将在存储库中为该定义创建架构。通过该存储库架构提供的结构，您可以编辑和验证 XML 定义。

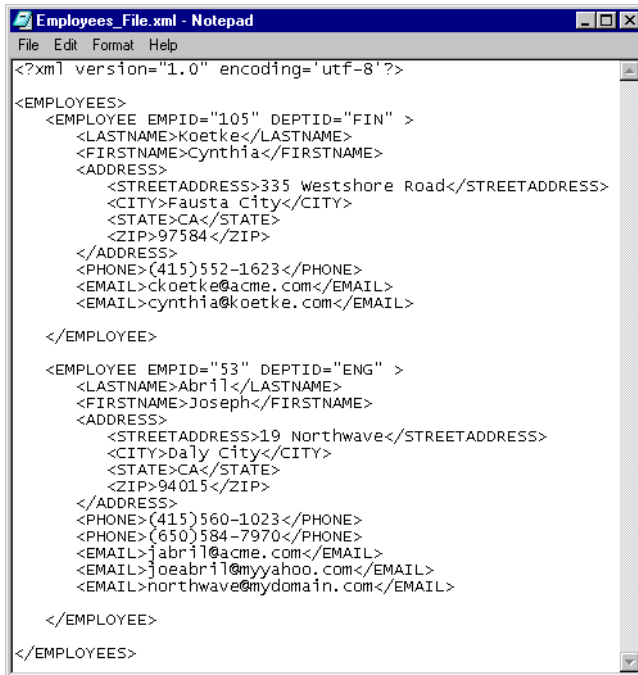
您可以从以下文件类型创建元数据：

- XML 文件
- DTD 文件
- XML 架构文件
- 关系表
- 平面文件

从 XML 文件导入元数据

在 XML 文件中，标记对用于标记每个数据元素的开头和结尾。这些标记是 PowerCenter 从 XML 文件中提取的元数据的基础。如果导入的 XML 文件没有关联的 DTD 或 XML 架构，Designer 将读取 XML 标记来确定元素、元素可能出现的次数及其在层次结构中的位置。Designer 会检查元素标记中的数据，并根据数据表示形式分配数据类型。您可以更改 XML 定义中这些元素的数据类型。

下图显示了一个示例 XML 文件：



根元素是 Employees。Employee 是一个多次出现的元素。Employee 元素包含 LastName、FirstName 和 Address。此外，Employee 元素还包含多次出现的元素：Phone 和 Email。

Designer 从 XML 数据确定架构结构。

下图显示了默认 XML 源定义以及根元素和多次出现的元素的单独视图：

| Name | Datatype |
|-------------------------|-------------|
| EMPLOYEES (X_EMPLOYEES) | |
| XPk_EMPLOYEES | xsd:integer |
| EMPLOYEE (X_EMPLOYEE) | |
| XPk_EMPLOYEE | xsd:integer |
| FK_EMPLOYEES | xsd:integer |
| DEPTID | xsd:string |
| EMPID | xsd:integer |
| LASTNAME | xsd:string |
| FIRSTNAME | xsd:string |
| STREETADDRESS | xsd:string |
| CITY | xsd:string |
| STATE | xsd:string |
| ZIP | xsd:integer |
| EMAIL (X_EMAIL) | |
| XPk_EMAIL | xsd:integer |
| FK_EMPLOYEE0 | xsd:integer |
| EMAIL | xsd:string |
| PHONE (X_PHONE) | |
| XPk_PHONE | xsd:integer |
| FK_EMPLOYEE | xsd:integer |
| PHONE | xsd:string |

导入 XML 文件时，不需要使用所有 XML 数据来创建 XML 定义。您需要足够的数据来准确显示 XML 文件的层次结构。

Designer 可以从引用 DTD 文件或 XML 架构的 XML 文件创建 XML 定义。如果 XML 文件引用了另一个节点上的 DTD 或 XML 架构，则承载 PowerCenter 客户端的节点必须对架构所在的节点拥有访问权限，以便 Designer 能够读取该架构。XML 文件包含通用资源标识符 (URI)，这是 DTD 或 XML 架构的地址。

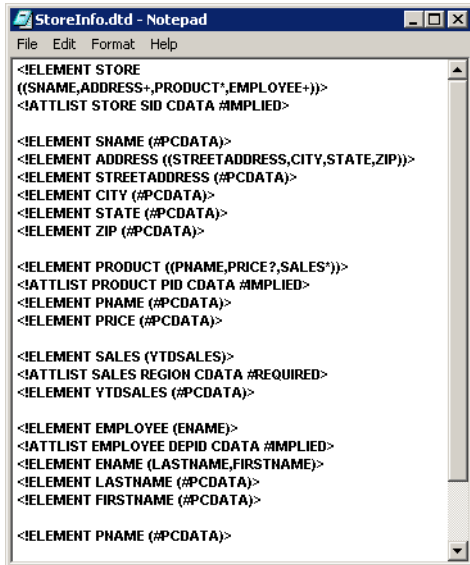
从 DTD 文件导入元数据

DTD 文件用于对 XML 文档结构提供约束。DTD 文件列出了 XML 文档的元素、属性、实体和符号。DTD 文件指定组件之间的关系。DTD 指定基数和空约束。但是，DTD 文件不包含任何数据或数据类型。

导入 DTD 文件时，可以更改 XML 定义中元素的数据类型。您可以更改空约束，但不能更改元素基数。

如果导入的 XML 文件包含关联的 DTD，Designer 将基于 DTD 结构创建定义。

下图显示了 XML 文件的一个示例，在该示例中，StoreInfo.dtd 包含 Store 元素，Product 是 Store 的子元素之一：



```
<!ELEMENT STORE
((SNAME,ADDRESS+,PRODUCT*,EMPLOYEE+))>
<!ATTLIST STORE SID CDATA #IMPLIED>

<!ELEMENT SNAME (#PCDATA)>
<!ELEMENT ADDRESS ((STREETADDRESS,CITY,STATE,ZIP))>
<!ELEMENT STREETADDRESS (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT ZIP (#PCDATA)>

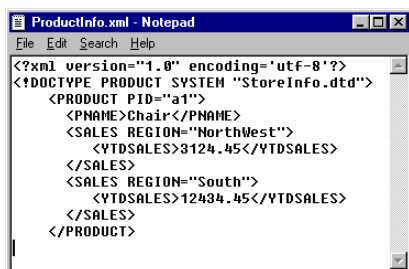
<!ELEMENT PRODUCT ((PNAME,PRICE?,SALES*))>
<!ATTLIST PRODUCT PID CDATA #IMPLIED>
<!ELEMENT PNAME (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>

<!ELEMENT SALES (YTD SALES)>
<!ATTLIST SALES REGION CDATA #REQUIRED>
<!ELEMENT YTD SALES (#PCDATA)>

<!ELEMENT EMPLOYEE (ENAME)>
<!ATTLIST EMPLOYEE DEPID CDATA #IMPLIED>
<!ELEMENT ENAME (LASTNAME,FIRSTNAME)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>

<!ELEMENT PNAME (#PCDATA)>
```

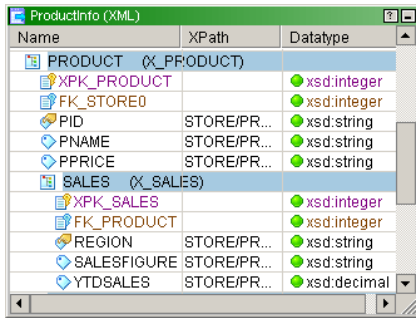
下图显示了关联的 DTD：



```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE PRODUCT SYSTEM "StoreInfo.dtd">
<PRODUCT PID="a1">
  <PNAME>Chair</PNAME>
  <SALES REGION="NorthWest">
    <YTD SALES>3124.45</YTD SALES>
  </SALES>
  <SALES REGION="South">
    <YTD SALES>12434.45</YTD SALES>
  </SALES>
</PRODUCT>
```

在关联的 DTD 中，ProductInfo.xml 使用了 StoreInfo.dtd 的 Product 元素。Product 包括多次出现的 Sales 元素。

下图显示了 Designer 创建的源定义：



| Name | XPath | Datatype |
|---------------------|-------------|-------------|
| PRODUCT (X_PRODUCT) | | |
| XPK_PRODUCT | | xsd:integer |
| FK_STORE0 | | xsd:integer |
| PID | STORE/PR... | xsd:string |
| PNAME | STORE/PR... | xsd:string |
| PPRICE | STORE/PR... | xsd:string |
| SALES (X_SALES) | | |
| XPK_SALES | | xsd:integer |
| FK_PRODUCT | | xsd:integer |
| REGION | STORE/PR... | xsd:string |
| SALESFIGURE | STORE/PR... | xsd:string |
| YTDSALES | STORE/PR... | xsd:decimal |

ProductInfo 定义包含 Product 和 Sales 组。XML 文件确定要在定义中包括的元素。DTD 文件确定 XML 定义的结构。

从 XML 架构导入元数据

架构文件定义 XML 文件中元素和属性的结构。架构文件包含有关文件中元素和属性类型的说明。当您导入 XML 架构时，Designer 将确定元素的数据类型、精度和基数。如果元素定义来自架构，则无法在 PowerCenter 中更改元素定义。

从 XML 架构导入元数据时，.xsd 文件可能会包含引用其他 .xsd 文件的 import 或 include 语句。导入包含其他架构的架构时，其他架构不得引用相同的命名空间。

示例：

```
<IMPORT
  schemaLocation="../../../administration/process/bo/LocationTextB0.xsd"
  namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/process/bo">
<IMPORT
  schemaLocation="../../../administration/process/bo/LineOfBusinessB0.xsd"
  namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/process/bo">
<IMPORT
  schemaLocation="../../../administration/process/bo/ClaimExceptionB0.xsd"
  namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/process/bo">
```

您可以将多条“import schemalocation”语句替换为一条语句：

```
<xsd:import schemalocation="imported.xsd" namespace="http://EnterpriseLibrary/com/acs/enterprise/
common/program/administration/process/bo"/>
```

imported.xsd 文件使用以下语法包含其他 XSD 文件：

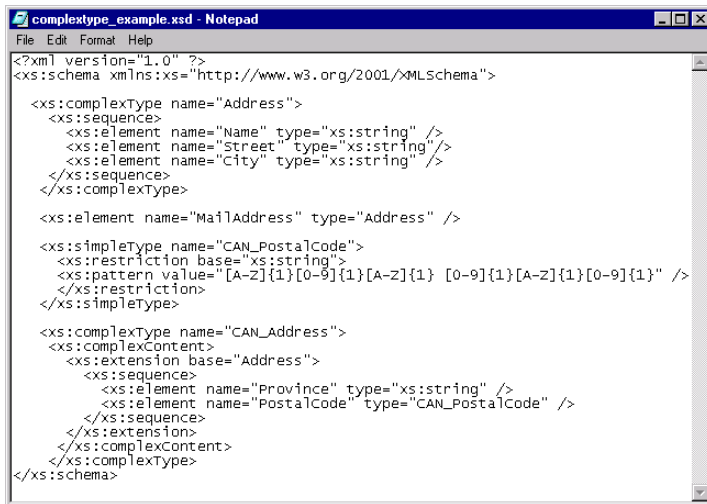
```
<xsd:schema targetNamespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/
administration/process/bo" elementFormDefault="qualified" >
  <xsd:include schemalocation=" LocationTextB0.xsd" />
  <xsd:include schemalocation=" LineOfBusinessB0.xsd" />
  <xsd:include schemalocation=" ClaimExceptionB0.xsd" />
</xsd:schema>
```

有关详细信息，请参阅知识库文章 158334。

XML 架构中的每个简单类型定义都是对架构原子数据类型（如布尔、字符串或整数）的另一个简单类型定义的限制，限制 anySimpleType 数据类型。当您在 XML 架构中定义简单数据类型时，请从现有数据类型派生新的数据类型。例如，您可以派生仅包含介于 1 到 20 之间的数字的受限制整数类型。基本类型是整数。

从其他数据类型派生复杂数据类型时，将创建一个包含基本类型元素的新数据类型。您可以将新元素添加到派生类型或对继承元素创建限制。Designer 将为派生类型创建视图，而无需复制表示继承组件的列。这样可以减少元数据，并减少存储库中 XML 定义的大小。

下图显示了一个具有简单和复杂派生类型的架构：



MailAddress 元素是属于复杂类型的 Address 类型。派生类型 CAN_Address 从 Address 类型继承了 Name、City 和 Street，并通过添加 Province 和 PostalCode 扩展了 Address。PostalCode 是一个称为 CAN_PostalCode 的简单类型。

导入 XML 架构时，复杂类型中的每个简单类型或属性都可以成为 XML 定义中的列。复杂类型将成为视图。

如果使用默认选项导入架构，下图将显示架构中的 XML 定义：

| Name | Datatype |
|-----------------------------|----------------|
| MailAddress (X_MailAddress) | |
| XPK_MailAddress | xsd:integer |
| MailAddress | Address |
| Address (X_Address) | |
| XPK_Address | xsd:integer |
| Name | xsd:string |
| Street | xsd:string |
| City | xsd:string |
| CAN_Address (X_CAN_Address) | |
| XPK_CAN_Address | xsd:integer |
| FK_Address | xsd:integer |
| Province | xsd:string |
| PostalCode | CAN_PostalCode |

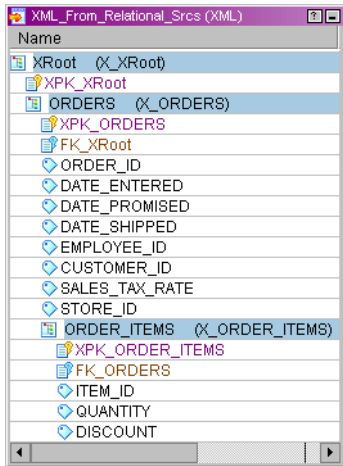
CAN_Address 视图包含其类型独有的元素。根元素是 MailAddress。Address 类型包含 Name、Street 和 City。CAN_Address 包含指向 Address 的外键。CAN_Address 包括 Province 和 PostalCode。

该视图不包含它从 MailAddress 继承的 Name、Street 和 City。

从关系定义创建元数据

您可以通过选择多个关系定义并在各个定义之间创建关系来创建 XML 定义。Designer 可为导入的每个关系定义创建 XML 视图。Designer 将转换关系定义中的每个列，并生成主键-外键关系。您可以选择创建根视图。

下图显示了关系定义（Orders 和 Order_Items）中的示例 XML 目标定义：

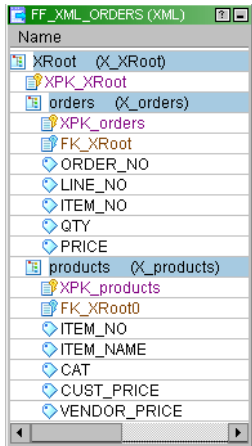


根是 XRoot。XRoot 包含 Orders 和 Order Items。Order_Items 包含指向 Orders 的外键。

从平面文件创建元数据

您可以通过从存储库导入平面文件定义来创建 XML 定义。如果导入多个平面文件定义，Designer 将创建具有每个平面文件的视图的 XML 定义。在 XML 定义中，这些视图彼此没有任何关系。如果您选择创建根视图，Designer 将创建具有指向根的外键的视图。

下图显示了平面文件 orders 和 products 中的示例 XML 源定义：



Products 和 Orders 具有指向根视图的外键，并已突出显示。

了解 XML 视图

XML 层次结构中元素之间的关系定义了 PowerCenter 定义中 XML 视图之间的关系。在源定义中，一个视图不必与任何其他视图相关。因此，源定义中的视图不需要主键或外键。非规范化视图可以独立于任何其他视图。但是，当视图与其他视图相关时，如果不指定键列，Designer 将生成键。

目标定义中的每个视图都必须至少与一个其他组相关。因此，每个视图都需要至少一个键来建立与另一个视图的关系。如果未指定键，Designer 将在目标视图中生成主键和外键。如果您在 XML 编辑器中创建视图和关系，而不是允许 Designer 为您创建，则可以定义视图的主键和外键。

当 Designer 创建主键或外键列时，它将分配具有前缀的列名称。在 XML 定义中，生成的主键列的前缀为 XPK_，生成的外键列的前缀为 XFK_。Designer 对指向主键的外键使用前缀 FK_。

例如，当 Designer 为 Sales 组创建主键列时，Designer 会将该列命名为 XPK_Sales。当 Designer 创建将 sales 组连接到另一个组的外键列时，它会将该列命名为 XFK_Sales。您可以重命名 Designer 创建的任何列名称。

如果映射中包含 XML 源，则当您运行会话时，集成服务将为源定义中生成的主键列创建值。您可以配置生成的键的起始值。

创建自定义 XML 视图

自定义视图是使用 XML 向导或 XML 编辑器创建的组。如果使用 XML 向导创建自定义视图，该向导将创建包含架构中所有组件的视图。如果使用 XML 编辑器，则可以定义每个视图并选择组件。

视图中的元素和视图之间的关系与 Designer 在您导入定义时在存储库中创建的架构有关。XML 编辑器使用有效视图的规则验证 XML 定义。

XML 视图的规则和准则

在使用视图键和关系时，请考虑以下规则和准则：

- PowerCenter XML 定义最多可以包含 400 个视图。
- 一个视图只能有一个主键。
- 一个视图可以与多个其他视图相关，并且可以有多个外键。
- 一个列不能既是主键，又是外键。
- 源定义中的视图不需要键。
- 目标定义中的视图需要至少一个键。
 - 目标根视图需要主键，但不需要外键。
 - 目标叶视图需要外键，但不需要主键。
- Enclosure 元素不能是键。
- 外键始终引用其他组中的主键。不能使用自引用键。
- 生成的外键列始终引用生成的主键列。
- XML 层次结构中元素的相对基数会影响 PowerCenter 在 XML 定义中创建视图的方式。以下规则可确定元素何时可以属于同一视图：
 - 具有一对一关系的元素可以属于同一视图。
 - 具有一对多关系的元素可以属于同一规范化或非规范化视图。
 - 具有多对多关系的元素不能属于同一视图。

了解层次结构关系

对于具有层次结构视图关系的 XML 定义，层次结构中的每个元素都显示在视图中其父元素下。多次出现的元素可以成为视图。复杂类型不会成为视图，派生复杂类型所特有的元素不会出现在任何视图中。

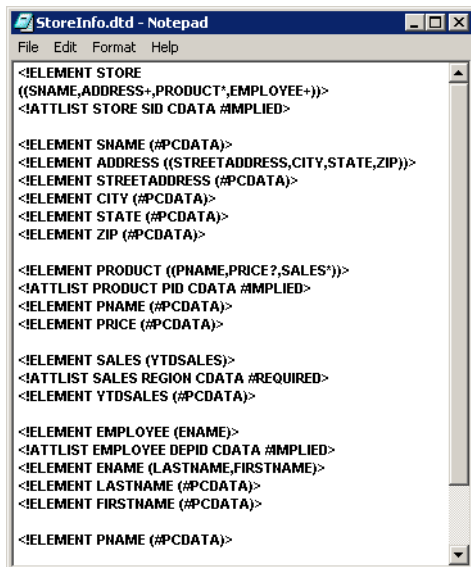
您可以生成以下类型的层次结构视图：

- **规范化视图。**具有规范化视图的 XML 定义通过将多次出现的数据划分为单独的视图来减少冗余。这些视图通过主键和外键关联。
- **非规范化视图。**具有非规范化视图的 XML 定义包含层次结构中非视图中的派生复杂类型所独有的所有元素。源或目标定义可以包含一个非规范化视图。

规范化视图

当 Designer 生成规范化视图时，它会建立根元素和成为 XML 定义中的视图的多次出现的元素。

下图显示了 DTD 文件和成为规范化 XML 定义中的视图的元素：



Store 是根元素。Address、product、employee 和 sales 是多次出现的元素。

下图显示了基于上图中 DTD 文件的源定义：

| Name | XPath | Datatype |
|-----------------------|-------------|-------------|
| STORE (X_STORE) | | |
| XP_K_STORE | | xsd:integer |
| SID | STORE/@SID | xsd:string |
| SNAME | STORE/SN... | xsd:string |
| ADDRESS (X_ADDRESS) | | |
| XP_K_AD... | | xsd:integer |
| FK_STO... | | xsd:integer |
| STREET... | STORE/AD... | xsd:string |
| CITY | STORE/AD... | xsd:string |
| STATE | STORE/AD... | xsd:string |
| ZIP | STORE/AD... | xsd:string |
| PRODUCT (X_PRODUCT) | | |
| XP_K_PR... | | xsd:integer |
| FK_STO... | | xsd:integer |
| PID | STORE/PR... | xsd:string |
| PNAME | STORE/PR... | xsd:string |
| PPRICE | STORE/PR... | xsd:string |
| SALES (X_SALES) | | |
| XP_K_S... | | xsd:integer |
| FK_P... | | xsd:integer |
| REGION | STORE/PR... | xsd:string |
| SALES... | STORE/PR... | xsd:string |
| EMPLOYEE (X_EMPLOYEE) | | |
| XP_K_EM... | | xsd:integer |
| FK_STO... | | xsd:integer |
| DEPID | STORE/EM... | xsd:string |
| LASTNA... | STORE/EM... | xsd:string |

定义包含规范化视图。根视图是 Store。Address、Product 和 Sales 视图具有指向 Store 的外键。Sales 视图具有指向 Product 视图的外键。

下表显示了 Store 视图的数据预览中的行：

| XPK_si_STORE | si_SID | si_NAME |
|--------------|--------|---------------------------------|
| 1 | BE1752 | Mud and Sawdust Furniture Store |

下表显示了 Address 视图的数据预览中的行：

| XPK_si_ADDRESS | FK_si_ADDRESS | si_STREETADDRESS | si_CITY | si_STATE | si_ZIP |
|----------------|---------------|--------------------|-------------|----------|--------|
| 1 | 1 | 335 Westshore Road | Fausta City | CA | 95784 |
| 2 | 1 | 7415 Endless Loop | Codeville | CA | 97412 |

下表显示了 Product 视图的数据预览中的行：

| XPK_si_PRODUCT | FK_si_PRODUCT | si_PNAME | si_PRICE | si_PID |
|----------------|---------------|----------|----------|--------|
| 1 | 1 | Chair | 5690.00 | a1 |
| 1 | 1 | Table | 1240.00 | a2 |
| 1 | 1 | Bed | 1364.99 | a3 |

下表显示了 Sales 视图的数据预览中的行：

| XPK_si_SALES | FK_si_SALES | si_REGION | si_YTDSALES |
|--------------|-------------|-----------|-------------|
| 1 | a1 | Northwest | 4565.44 |
| 2 | a2 | South | 8793.99 |
| 3 | a3 | East | 23110.00 |
| 4 | a4 | South | 5500.00 |
| 5 | a5 | Northwest | 10095.34 |
| 6 | a6 | East | 200.00 |

下表显示了 Employee 视图的数据预览中的行：

| XPK_si_EMPLOYEE | FK_si_EMPLOYEE | si_FIRSTNAME | si_LASTNAME |
|-----------------|----------------|--------------|-------------|
| 1 | 1 | James | Bond |
| 2 | 1 | Austin | Powers |

| XPk_si_EMPLOYEE | FK_si_EMPLOYEE | si_FIRSTNAME | si_LASTNAME |
|-----------------|----------------|--------------|-------------|
| 3 | 1 | Indiana | Jones |
| 4 | 1 | Foxie | Brown |
| 5 | 1 | Bonnie | Bell |
| 6 | 1 | Laura | Croft |

非规范化视图

当 Designer 生成非规范化视图时，它将创建一个视图并将层次结构中的所有元素置于该视图。非规范化视图中的所有元素都属于同一个父链。非规范化视图与非规范化表类似，可生成重复数据。

如果多次出现的元素具有一对多关系，并且全部属于相同父链，则 Designer 可以为包含多个多次出现的元素的 XML 定义生成非规范化视图。

下图显示了包含多次出现的元素的 DTD 文件，在本例中，这些元素为 Product 和 Sales：

```

<!ELEMENT STORE (SNAME, PRODUCT*)>
<?ATTLIST STORE SID CDATA #REQUIRED>
<?ELEMENT SNAME (#PCDATA)>

<?ELEMENT PRODUCT (PNAME, PPRICE?, SALES*)>
<?ATTLIST PRODUCT PID ID #REQUIRED>
<?ELEMENT PNAME (#PCDATA)>
<?ELEMENT PPRICE (#PCDATA)>

<?ELEMENT SALES (YTDSALES)>
<?ATTLIST SALES REGION CDATA #REQUIRED>
<?ELEMENT YTDSALES (#PCDATA)>

```

Product 和 Sales 是多次出现的元素。由于多次出现的元素具有一对多关系，Designer 可以创建包含所有元素的单个非规范化视图。

下图显示了源定义中 ProdAndSales.dtd 的非规范化视图：

| Name | XPath | Datatype |
|-----------------|------------|------------|
| SALES (X_STORE) | | |
| YTDSALES | ./YTDSALES | xsd:string |
| REGION | ./@REGION | xsd:string |
| PRICE | ./PRICE | xsd:string |
| PNAME | ./PNAME | xsd:string |
| PID | ./@PID | xsd:string |
| SNAME | ./SNAME | xsd:string |
| SID | ./@SID | xsd:string |

Designer 为 ProdAndSales 层次结构中的所有元素创建单一视图。由于 DTD 文件不会定义数据类型，因此 Designer 会将字符串的数据类型分配给所有列。非规范化视图不需要主键或外键。

下图显示了非规范化视图的数据预览：

| SID | SNAME | PID | PNAME | PPRICE | REGION | YTDSALES |
|--------|---------------------------------|-----|---------|---------|-----------|----------|
| BE1752 | Mud and Sawdust Furniture Store | a1 | Chair | NULL | NorthWest | 3124.45 |
| BE1752 | Mud and Sawdust Furniture Store | a1 | Chair | NULL | South | 12434.45 |
| BE1752 | Mud and Sawdust Furniture Store | a2 | Table | 4126.33 | South | 8252.66 |
| BE1752 | Mud and Sawdust Furniture Store | a3 | Bed | 4356.33 | NorthWest | 43563.30 |
| BE1752 | Mud and Sawdust Furniture Store | a3 | Bed | 4356.33 | South | 21781.65 |
| BE1752 | Mud and Sawdust Furniture Store | a3 | Bed | 4356.33 | East | 26137.98 |
| BE1752 | Mud and Sawdust Furniture Store | a4 | Etagere | 5000.00 | South | 20000.00 |
| BE1752 | Mud and Sawdust Furniture Store | a4 | Etagere | 5000.00 | East | 5000.00 |
| BE1752 | Mud and Sawdust Furniture Store | a4 | Etagere | 5000.00 | NorthWest | 15000.00 |

了解实体关系

您可以从 XML 架构创建实体关系。创建包含实体关系的 XML 定义时，Designer 会为多次出现的元素、元素组和复杂类型生成单独的视图。Designer 包括所有派生复杂类型的视图。Designer 会基于类型和层次结构关系在视图之间创建链接和键。

使用 XML 架构时，可以引用架构的一部分，而不是重复架构组件中的相同信息。一个组件可以继承另一个组件的元素和属性，并限制或扩展该组件的元素。例如，您可能会使用复杂类型作为创建新的复杂类型的基础。您可以将更多元素添加到新类型，以创建扩展复杂类型。或者，您可以创建受限制复杂类型，它是另一个复杂类型的子集。

如果手动创建视图或在 XML 编辑器中重新创建实体关系，则可以选择构造元数据的方式。如果基于使用继承的 XML 架构创建 XML 定义，则可以为基本类型和派生类型生成单独的视图。如果计划将 XML 数据映射到规范化关系表，则可以创建继承关系。

XML 类型 I 继承关系是两个视图之间的关系。每个视图根都是全局复杂类型。一个视图派生自另一个视图。

可以在列和视图之间创建继承关系。这是 XML 类型 II 继承关系。

Designer 可为置换组生成单独的视图。

实体关系的规则和准则

Designer 可基于以下准则生成实体：

- 实体表示 XML、DTD 或 XML 架构层次结构的一部分。这种层次结构不需要从 XML 文件的根开始。
- Designer 使用 DTD 文件中定义的实体来创建实体关系。
- Designer 使用 XML 架构中定义的类型结构来生成实体关系。
- 当遇到父元素下的多次出现的元素时，Designer 将创建新的实体。
- Designer 为置换组的每个成员生成单独的视图。
- Designer 生成主键和外键来关联单独的实体。

类型 1 实体关系示例

XML 类型 1 实体关系是两个视图之间的关系。每个视图必须以全局复杂类型为根。一个视图必须派生自另一个视图。

以下架构包含 PublicationType、BookType 和 MagazineType。PublicationType 是基本类型。出版物包括 Title、Author 和 Date。BookType 和 MagazineType 是派生类型，它们扩展了 PublicationType。Book 包括 ISBN 和 Publisher，Magazine 包括 Volume 和 Edition。

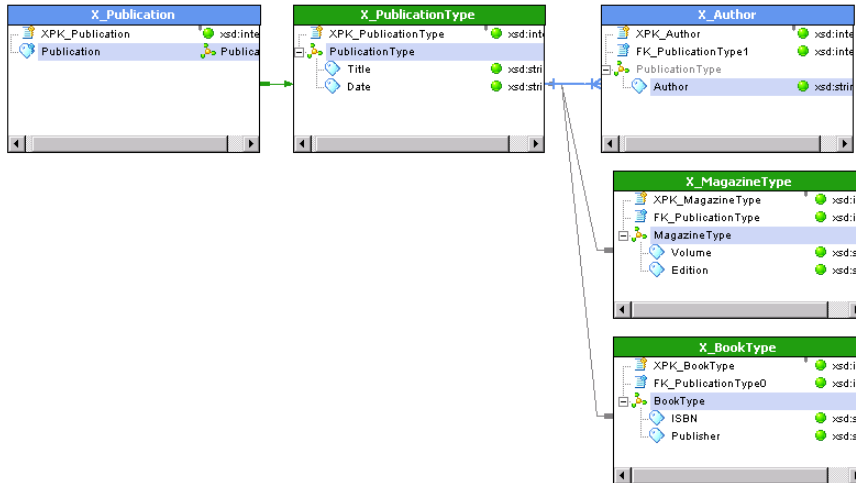
```
<xsd:complexType name="PublicationType">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Publication" type="PublicationType"/>
<xsd:complexType name="BookType">
  <xsd:complexContent>
    <xsd:extension base="PublicationType">
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MagazineType">
  <xsd:complexContent>
    <xsd:extension base="PublicationType">
      <xsd:sequence>
        <xsd:element name="Volume" type="xsd:string"/>
        <xsd:element name="Edition" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

作为 XML 定义中的实体创建 XML 视图时，PublicationType 中的 Title 和 Date 元数据不会在 BookType 或 MagazineType 中重复。相反，这些视图包含的元数据可将其与 PublicationType 区分开：针对 BookType 的 ISBN 和 Publisher 以及针对 MagazineType 的 Volume 和 Edition。它们具有可将其链接到 PublicationType 的外键。

本示例使用的元数据分解减少，因为基本类型中的元素均不会在派生类型中重复。

Author 是 Publication 中的多次出现的元素。Author 成为了 XML 视图。

下图显示了 Designer 从架构中生成的默认视图：



下图显示了包含出版物、杂志和书籍的 XML 文件：

```

Bookstore_TYPES.xml - Notepad
File Edit Format Help
<?xml version="1.0"?>
<bk:BookStore xmlns:bk="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.books.org
    BookStore.xsd">
  <Publication>
    <Title>Staying Young Forever</Title>
    <Author>Karin Granstrom Jordan, M.D.</Author>
    <Date>1999</Date>
  </Publication>
  <Publication xsi:type="bk:BookType">
    <Title>Illusions The Adventures of a Reluctant Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </Publication>
  <Publication xsi:type="bk:MagazineType">
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <Volume>IV</Volume>
    <Edition>Spring</Edition>
  </Publication>
  <Publication xsi:type="bk:BookType">
    <Title>Working with XML</Title>
    <Author>Dell Skidmore</Author>
    <Date>1999</Date>
    <ISBN>0-444-84429-4</ISBN>
    <Publisher>Purdum Publishing Co.</Publisher>
  </Publication>
</bk:BookStore>

```

如果使用上图中的 XML 定义处理示例 XML 文件，则可以在以下视图中创建数据：

- **PublicationType** 视图。包含每个出版物的标题和日期。

下图显示了 PublicationType 视图：

| XPK_boo_PublicationType | Title | Date | |
|-------------------------|-----------------|------|--|
| 1 | Staying Yo... | 1999 | |
| 2 | Illusions Th... | 1977 | |
| 3 | The First a... | 1954 | |
| 4 | Working wi... | 1999 | |

- **BookType** 视图。包含 ISBN 和出版商。BookType 包含指向 PublicationType 的外键。

下图显示了 BookType 视图：

| XPK_boo_BookType | FK_boo_PublicationType1 | ISBN | Publisher |
|------------------|-------------------------|---------------|-----------------------|
| 1 | 2 | 0-440-34319-4 | Dell Publishing Co. |
| 2 | 4 | 0-444-84429-4 | Purdam Publishing Co. |

- **MagazineType 视图。**包含卷和版。MagazineType 也包含指向 PublicationType 的外键。

下图显示了 MagazineType 视图：

| XPK_boo_MagazineType | FK_boo_PublicationType | Volume | Edition |
|----------------------|------------------------|--------|---------|
| 1 | 3 | IV | Spring |

- **Author 视图。**包含所有出版物的作者。Designer 会为 Author 生成单独的视图，因为 Author 是多次出现的元素。每个出版物都可以包含多个作者。

下图显示了 Author 视图：

| XPK_Author | FK_boo_PublicationType0 | Author |
|------------|-------------------------|------------------------------|
| 1 | 1 | Karin Granstrom Jordan, M.D. |
| 2 | 2 | Richard Bach |
| 3 | 3 | J. Krishnamurti |
| 4 | 4 | Dell Skidmore |

类型 II 实体关系示例

可以在列和复杂类型视图之间创建继承关系。该列必须是本地复杂类型的元素。该视图必须以全局复杂类型为根。本地复杂类型必须派生自全局复杂类型。

例如，以下架构定义了称为 EmployeeType 的复杂类型。EmployeeType 包含 EmployeeNumber 和 EmployeeName 元素。

EmployeeStatusType 包括称为 Employee 的元素，该元素扩展了 EmployeeType。Employee 包括 EmployeeStatus 元素。

```
<xs:element name="Employee_Payroll">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EmployeeStatus" type="EmpStatusType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="EmpStatusType">
  <xs:sequence>
    <xs:element name="Employee" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="EmployeeType">
            <xs:sequence>
              <xs:element name="EmployeeStatus" type="xs:string">
            </xs:element>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EmployeeType">
  <xs:sequence>
    <xs:element name="EmployeeName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

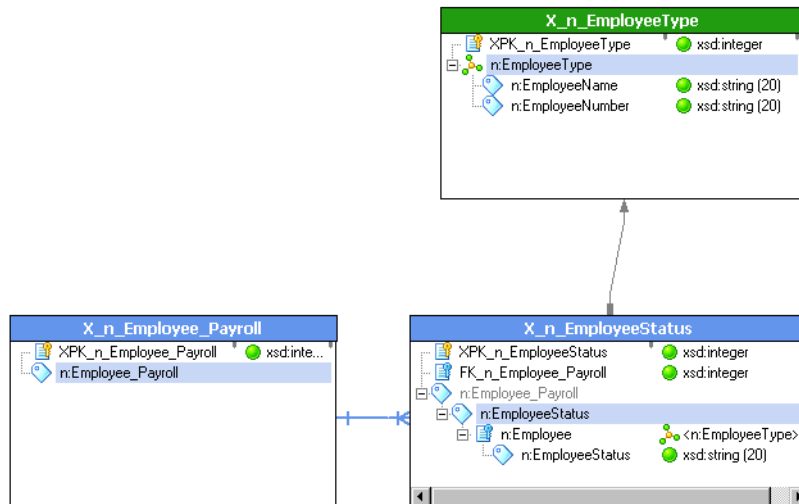
```

        <xs:element name="EmployeeNumber" type="xs:string"></xs:element>
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

导入架构时，Designer 将为 Employee_Payroll、EmployeeType 和 EmployeeStatus 创建视图。EmployeeStatus 视图包含称为 Employee 的列。Employee 派生自 EmployeeType。

下图显示了 Employee_Payroll 视图、EmployeeType 视图和 EmployeeStatus XML 视图：



Employee_Payroll 视图包含 Employee_Payroll 元素和主键 PK_Employee_Payroll。Employee_Payroll 视图通过蓝色横线连接到 EmployeeStatus 视图，该横线指示这两个视图之间存在一对多关系。Employee_Payroll 包含多次出现的 EmployeeStatus。

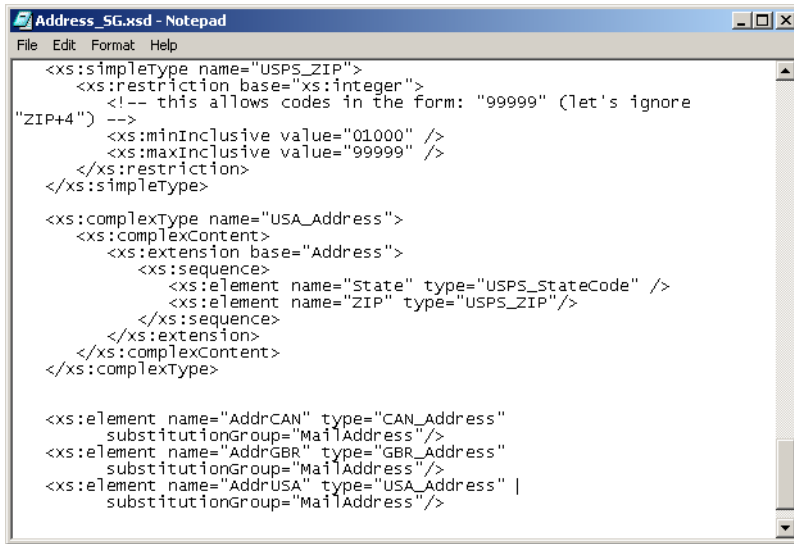
EmployeeStatus 视图包含 EmployeeType 类型的 Employee 元素。Employee 元素通过包括 EmployeeStatus 元素扩展了 EmployeeType。EmployeeStatus 视图还包含指向 Employee_Payroll 的外键。EmployeeStatus 视图通过灰色箭头连接到 EmployeeType 视图。该箭头指示这两个视图之间存在类型关系。

EmployeeType 视图包含由 EmployeeName 和 EmployeeNumber 组成的 EmployeeType。

在 XML 定义中使用置换组

创建包含实体关系的 XML 定义时，Designer 会为元素组和复杂类型生成单独的视图。导入使用置换组的 XML 架构时，Designer 会将置换组的每个成员作为单独的实体导入。Designer 为每个组生成单独的视图。

下图显示了 XML 架构的示例部分，其中包含置换组 MailAddress：



下图显示了具有置换组每个成员的视图的 XML 定义，包括 AddrCAN、AddrGBR、AddrUSA、ShortAddress 和 Street：

| Address_Subgrp (XML) | | |
|-------------------------------|----------------|---------------|
| Name | XPath | Datatype |
| AddrCAN (X_AddrCAN) | | |
| XPk_Addr... | | xsd:integer |
| AddrCAN | AddrCAN | xsd:integer |
| Province | AddrCAN/Pr... | xsd:string |
| PostalCode | AddrCAN/P... | CAN_Postal... |
| AddrGBR (X_AddrGBR) | | |
| XPk_Addr... | | xsd:integer |
| AddrGBR | AddrGBR | xsd:integer |
| County | AddrGBR/C... | xsd:string |
| Postcode | AddrGBR/P... | GBR_Postc... |
| AddrUSA (X_AddrUSA) | | |
| XPk_Addr... | | xsd:integer |
| AddrUSA | AddrUSA | xsd:integer |
| State | AddrUSA/St... | USPS_State... |
| ZIP | AddrUSA/ZIP | USPS_ZIP |
| ShortAddress (X_ShortAddress) | | |
| XPk_Short... | | xsd:integer |
| Name | type(ShortA... | xsd:string |
| City | type(ShortA... | xsd:string |
| Street (X_Street2) | | |
| XPk_Str... | | xsd:integer |
| FK_Shor... | | xsd:integer |
| Street | type(ShortA... | xsd:string |

使用循环引用

循环关系是 XML 定义中两个视图之间或 XML 定义的单个视图中的循环层次结构关系。例如，称为 Part 的复杂元素可能包含 ID、部分名称和对其他部分的引用。

以下示例显示了 Part 元素组件：

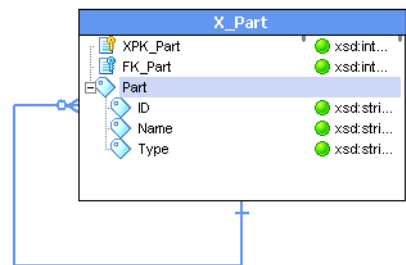
```
<xs:element name="Part">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Type" type="xs:string"/>
      <xs:element ref="Part" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    /xs:sequence>
  </xs:complexType>
</xs:element>

```

下图显示了 XML 编辑器工作区中的循环引用，其中包含称为 Part 的复杂元素：



您可以使用 Part XML 定义读取会话中的以下 XML 文件：

```

<Part>
  <ID>1</ID>
  <Name>Big Part</Name>
  <Type>L</Type>
  <Part>
    <ID>1.A</ID>
    <Name>Middle Part</Name>
    <Type>M</Type>
    <Part>
      <ID>1.A.B</ID>
      <Name>Small Part</Name>
      <Type>S</Type>
    </Part>
  </Part>
</Part>

```

在 XML 文件中，Part 1 包含 Part 1.A，Part 1.A 包含 Part 1.A.B。

下图显示了会话可能会从 XML 源生成的数据和键：

| XPK_Part | FK_Part | ID | Name | Type |
|----------|---------|-------|-------------|------|
| 1 | NULL | 1 | Big Part | L |
| 2 | 1 | 1.A | Middle Part | M |
| 3 | 2 | 1.A.B | Small Part | S |

注意: 如果为基于约束的加载启用了某个会话，且会话中包含循环 XML 引用，则不能运行该会话。该会话会拒绝所有行。

了解视图行

要从 XML 文档中提取数据，请指定要生成的行、要包括的数据列以及生成行的时间。当您在 XML 编辑器中定义视图时，可以创建视图行、元素或集成服务生成数据行所需的全局复杂类型。

集成服务使用视图行来确定何时读取和写入 XML 视图的数据。您可以在任意单次或多次出现的元素中设置视图行。设置视图行后，向视图中添加的每个元素与视图行一一对应。

例如，Employees 视图包含元素 Employee、Name、Firstname 和 Lastname。将视图行设置为 Employee 时，集成服务将使用以下算法提取数据：

```
For every (Employees/Employee)
extract ./Name/Firstname/Lastname
```

Employees XML 架构可能包含以下元素：

```
EMPLOYEES
  EMPLOYEE+
    ADDRESS+
    NAME
    FIRSTNAME
    LASTNAME
    EMAIL+
```

Employee、Address 和 Email 是多次出现的元素。可以创建包含以下元素的视图：

```
EMPLOYEE
  ADDRESS
  NAME
```

如果将视图行设置为 Address，集成服务将提取 XML 数据中每个 Employee/Address 的 Name。不能向此视图添加 Email，因为您将在 Address 和 Email 之间创建多对多关系。

可以向视图添加多次出现的转换列。视图行可以包含转换列。

例如，可以将 Email 的一个实例作为转换列添加到 Employee 视图。该视图将包含以下元素：

```
EMPLOYEE
  ADDRESS
  NAME
  EMAIL[1]
```

该视图可能包含视图行 EMPLOYEE/ADDRESS/EMAIL[1]。集成服务将提取 Employee/Address/Email 的第一个实例的数据。

使用 XPath 查询谓词

使用 XML 视图中的查询筛选 XML 源数据。集成服务基于该查询从源 XML 文件中提取数据。如果查询为 true，集成服务将提取视图的数据。

要在 XML 视图中创建查询，请在 XML 编辑器中创建 XPath 查询谓词。XPath 是一种描述如何定位 XML 文档中的项的语言。XPath 使用基于从根组件通过整个 XML 层次结构的路径的定位语法。可以针对视图行中的元素或包含的 XPath 中具有视图行的元素和属性创建 XPath 查询谓词。

XPath 查询谓词包括要提取的元素或属性以及确定条件的查询谓词。您可以验证元素或属性的值，或者验证源 XML 数据中是否存在某个元素或属性。

使用视图行的规则和准则

使用以下规则和准则在 XML 定义中使用视图行：

- 视图行必须是类型或元素。视图行不能是属性。
- 每个视图都必须具有视图行，该视图行必须是元素或复杂类型。
- 视图根是视图中的顶级元素。视图根是视图中所有其他元素的父元素。
- 视图行可以与视图根相同，除非该视图为非规范化视图。
- 两个视图可以包含 XML 源或 XML 解析器转换中的同一视图行。

- 视图行元素必须是视图中最低级别的多次出现的元素。视图不能包含多对多关系。
- 如果将多次出现的元素添加到没有其他多次出现的任何元素的视图中，则默认情况下会将视图行更改为新元素。如果视图中已有多次出现的元素，则无法添加其他多次出现的元素。
- 创建空视图时，无需指定视图行。但是，一旦向该视图添加列，Designer 就会创建视图行。即使仅添加主键也是如此。
- 您可以稍后更改视图行，但不能更改视图根，除非视图中没有架构组件。
- 您可以指定由转换元素组成的视图行，例如：
`Product/Order[2]/Customer`
- 视图的有效视图行是从层次结构关系顶部到视图中的视图行的视图行路径。一个视图可以有多个有效视图行，因为视图在 XML 定义中可以有多层次结构关系。

您可以在 XML 编辑器中指定影响视图行和有效视图行影响数据输出的方式的选项。

透视列

有时，出现多次的元素是一组包含不同的值的相同元素。例如，称为 Sales 的元素出现了 12 次，它可能包含一年中每个月的销售数字。或者，称为 Address 的元素出现了两次，它可能是家庭住址和办公室地址。

如果 XML 源中存在此类型的元素，则请使用转换将元素的出现次数视为组中单独的列。要转换 XML 视图中元素的出现次数，请在定义中为要表示的每次出现创建列。在月度销售示例中，如果要将 12 次出现表示为列，请在视图中创建 12 个销售列。如果要表示一个季度的销售情况，请创建三个列。运行会话时，集成服务将忽略未包括在定义中的出现的任何 XML 数据。

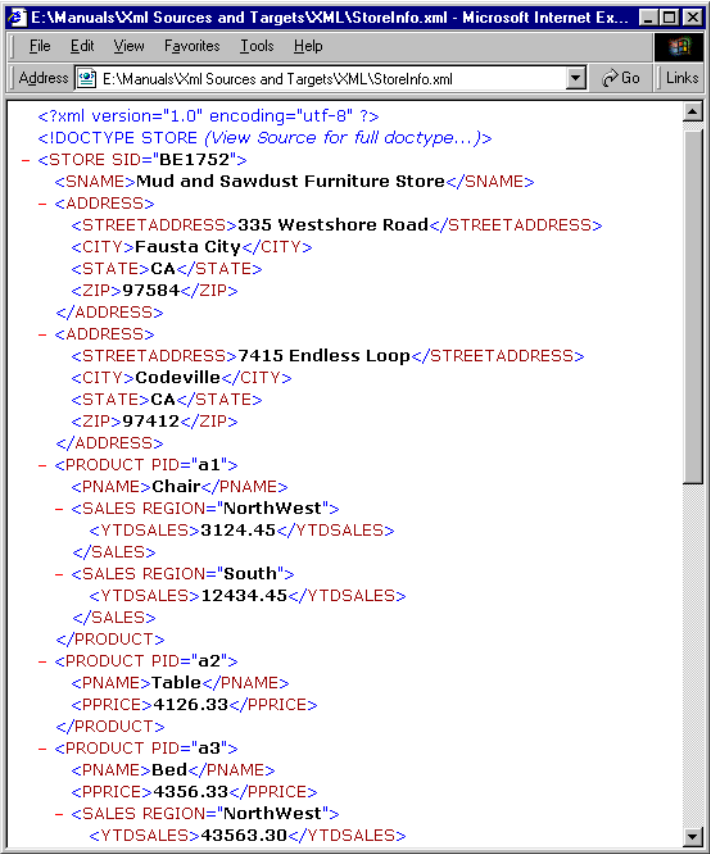
在 XML 源定义中添加或编辑视图时，可以转换列。

可以转换简单类型和复杂类型。不能转换主键列。转换视图中的列后，由此产生的组结构必须遵循有效的规范化或非规范化视图的规则。如果转换列使视图失效，Designer 将显示警告和错误。

转换会影响转换元素的视图中的元素。转换视图中的元素时，不会更改另一个视图中的相同元素。

注意：不能转换 XML 目标中的列。

下图显示了 StoreInfo XML 文件中 Address 元素的两次出现：



首次出现的 Address 转换至前缀为 HOM_ 的家庭住址列。第二次出现的 Address 转换至前缀为 OFC_ 的办公室地址列。XPath 显示来自相同元素的两个列集。

下图显示了 StoreInfo XML 文件的转换为两组地址列的 ADDRESS 元素：

| PowerCenter Column ... | Type | Not ... | XPath |
|------------------------|-----------------|--------------------------|---|
| SID | xsd:string (30) | <input type="checkbox"/> | ./@SID |
| st_SNAME | xsd:string (30) | <input type="checkbox"/> | ./st:SNAME |
| HOM_STREETADDRESS | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS1/STREETADDRESS |
| HOM_CITY | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS1/CITY |
| HOM_STATE | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS1/STATE |
| HOM_ZIP | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS1/ZIP |
| OFC_STREETADDRESS | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS2/STREETADDRESS |
| OFC_CITY | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS2/CITY |
| OFC_STATE | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS2/STATE |
| OFC_ZIP | xsd:string (30) | <input type="checkbox"/> | ./st:ADDRESS2/ZIP |

在下图中，第一次和第二次出现的地址（前缀为 HOM_ 和 OFC_）显示为组中的列：

| GPX_ADDRESS | FK_SID | HOM_STREETADDRESS | HOM_CITY | HOM_STATE | HOM_ZIP | OFC_STREETADDRESS | OFC_CITY | OFC_STATE | OFC_ZIP |
|-------------|--------|------------------------|-------------|-----------|---------|-------------------|-----------|-----------|---------|
| 1 | 1 | 335 West Bayshore Road | Fausta city | CA | 97584 | 7415 Endless Loop | Codeville | CA | 97412 |

使用多级转换

通过为列的 XPath 中多次出现的元素指定固定偏移量，可以转换视图中多个级别的元素。例如，视图中可能有以下元素：

```
STORE
  PRODUCT+
    PNAME
    ORDER+
      ORDERNAME
      CUSTOMER+
        CUSTNAME
```

XPath `STORE/PRODUCT[2]/ORDER[1]/ORDERNAME` 是指商店中第二个产品的第一个订单的订单名称。XPath `STORE/PRODUCT[2]/ORDER/CUSTOMER[1]` 是指第二个产品的所有订单的第一个客户。

如果转换某个视图行，则 XML 视图中出现在该视图行下方的任何列都必须具有与该视图行的 XPath 匹配的 XPath。

例如，视图可能包含以下视图行：

```
Transaction/Trade[1]
```

以下各列的 Trade 在 XPath 中的出现次数相同：

```
Transaction/Trade[1]/Date
Transaction/Trade[1]/Price
Transaction/Trade[1]/Person[1]/Firstname
```

不能在视图中创建具有以下 XPath 的列：

```
Transaction/Trade[2]/Date
```

使用 XML 源

使用 XML 源概览

Designer 提供了 XML 向导，可供您在存储库中创建 XML 定义。可以从 URL 或本地节点导入文件，创建 XML 定义；也可以从 PowerCenter 存储库导入关系或平面文件定义。可以从下列文件类型创建 XML 定义：

- XML 文件
- XML 架构文件
- DTD 文件
- 关系定义
- 平面文件定义

创建 XML 定义时，使用 XML 向导导入文件并将元数据组织到 XML 视图中。XML 视图是 XML 文件中包含元素和属性在内的列组。该向导可以为您生成视图，您也可以创建自定义视图。

您可以在 XML 向导中创建视图之间的关系：既可以创建层次结构关系，也可以创建实体关系。

如果更改了架构的结构，可以针对 XML 架构文件同步 XML 定义。

导入 XML 源定义

当从 XML 架构或 DTD 文件导入源定义时，Designer 可以基于 DTD 或 XML 架构文件中提供的说明提供数据的准确定义。当基于无关联 DTD 或 XML 架构的 XML 文件导入源定义时，XML 向导基于在 XML 文件中表示的数据决定数据的类型和出现。创建 XML 定义时，可能会得到意想不到的结果。例如，Designer 为字符串列定义的小数位数属性可能不准确。如果您导出 XML 源定义并导入小数位数属性不准确的定义，则会发生错误。

创建 XML 源定义后，不能将源定义更改为任何其他源类型。反过来说，您也不能将其他类型的源定义更改为 XML 定义。

XML 向导使用键关联 XML 视图和重建 XML 层次结构。可以选择生成视图和主键，也可以创建视图，并指定键。创建自定义视图时，可以选择根并选择如何处理元数据扩展。

XML 向导将 XML 层次结构和视图信息另存为存储库中的 XML 架构。导入 XML 定义时，能否改变层次结构中元素的基数和数据类型取决于要导入的文件类型。例如，DTD 和 XML 文件不存储数据类型信息。导入这些文件以创建 XML 定义时，可以在 Designer 中配置数据类型、精度和小数位数。如果导入 XML 架构，则可以更改精度和小数位数。

无法从所导出存储库对象的 XML 文件创建 XML 源定义。导入源定义时，Designer 将对存储库中的 XML 定义应用默认代码页。代码页基于 PowerCenter 客户端代码页。不能更改 XML 源定义的代码页，但在创建后，可以更改 XML 目标定义的代码页。

使用 XML 向导导入 XML 源定义。

要导入 XML 文件，请执行以下操作：

1. 单击“源” > “导入 XML 定义”。
2. 单击“高级选项”。

将出现“更改 XML 视图创建和命名选项”对话框。选择指定 Designer 如何创建和命名 XML 视图的选项。

下表描述了 XML 视图选项：

| 选项 | 描述 |
|-------------------------|---|
| 替代所有无限长度 | 对于字符串等未定义长度的组件，可以指定组件的默认长度。如果不设置默认长度，这些组件的精度会设置为无限。运行包含大文件的会话时，无限精度可能导致 DTM 缓冲区大小错误。 |
| 作为全局声明分析独立 XML 中的元素/属性 | 选择此选项可创建独立的 XML 元素或属性的全局声明。可以通过在架构其他部分中加以引用而重用全局元素。清除此选项时，独立 XML 为本地声明。 |
| 为 enclosure 元素创建 XML 视图 | 如果 enclosure 元素及其子元素出现一次以上，则可以为 enclosure 元素创建单独的视图。enclosure 元素是一种不含文本内容或属性但有子元素的元素。 |
| 透视列中的元素 | 如果叶元素有匹配项限制，则可以对它们进行透视。只可以透视源定义中的元素。 |

| 选项 | 描述 |
|-------------|---|
| 忽略固定元素和属性值 | 可以忽略某个架构中的固定值，并允许数据中的其他元素值。 |
| 忽略禁止的属性 | 可以将某个属性声明为 XML 架构中的禁止属性。禁止的属性会限制复杂类型。导入架构或文件时，可以选择忽略这些禁止的属性。 |
| 为 XML 列生成名称 | <p>您可以选择使用数字序列或架构中的元素或属性名称命名 XML 列。如果使用名称，请从以下选项中进行选择：</p> <ul style="list-style-type: none"> - 当 XML 列引用某个属性时，将元素名称作为其前缀。对于 XML 列的名称，PowerCenter 使用以下格式： NameOfElement_NameOfAttribute - 为每个 XML 列添加 XML 视图名称作为其前缀。对于 XML 列的名称，PowerCenter 使用以下格式：NameOfView_NameOfElement - 为每个外键列添加 XML 视图名称作为其前缀。对于所生成外键列的名称，PowerCenter 使用以下格式： FK_NameOfView_NameOfParentView_NameOfPKColumn <p>列名称的最大长度为 80 个字符。PowerCenter 将截断长度超过 80 个字符的列名称。如果列名称不唯一，PowerCenter 将添加数字后缀以使名称保持唯一。</p> |

- 单击“确定”以应用更改。
- 选择要导入文件的类型。可以选择以下选项：
 - **从本地 XML 文件或 URL 导入定义。**从 XML、DTD 或 XML 架构文件创建源定义。如果导入带关联 DTD 或架构的 XML 文件，XML 向导会使用 DTD 或架构生成 XML 文档。
 - **从非 XML 源或目标导入定义。**使用此选项可以从平面文件或关系定义创建源定义。除根元素组外，新的源定义还包含每个输入定义的一个组。
- 单击“下一步”完成 XML 向导。

多行属性值

XML 向导不允许属性值包含换行符且跨多个行。当从包含带换行符的属性的 XML 文件导入源或目标定义时，XML 向导将显示错误且不导入该文件。

使用 XML 视图

Designer 将视图显示为源定义中的组。

XML 向导提供用于在定义中创建视图的选项，您也可以在 XML 编辑器中手动创建视图。

可以从下列选项中选择以创建 XML 视图：

- **生成实体关系。**如果创建实体关系，XML 向导将为多次出现或被引用的元素和复杂类型生成视图。
- **生成层次结构关系。**创建层次结构关系时，每个对组件的引用在其父元素下展开。可以在层次结构关系中生成规范化或非规范化 XML 视图。
 - **规范化 XML 视图。**生成规范化 XML 视图时，元素和属性出现一次。多次出现的元素或一对多关系中的元素出现在由键关联的不同视图中。
 - **非规范化的 XML 视图。**生成非规范化的 XML 视图时，所有的元素和属性都显示在一个视图中。在 XML 定义中，Designer 不对元素和属性之间的多对多关系进行建模。

- **创建自定义的 XML 视图。**创建自定义 XML 视图时，可以将任何全局元素指定为根。可以针对元素、复杂类型和继承的复杂类型选择减少元数据分解。
- **同步 XML 定义。**可以更新基本架构被更改的一个或多个 XML 定义。
- **跳过创建 XML 视图。**如果选择跳过创建 XML 视图，可以稍后在 XML 编辑器中定义它们。在 XML 编辑器中定义视图时，可以定义要匹配目标和简化映射的视图。
- **在 XML 文件中创建元素和属性的 XML 视图。**导入包含关联架构的 XML 文件时，可以只为 XML 文件中的元素和属性（而不是架构中的所有组件）创建 XML 视图。

如果选择生成实体或层次结构关系，Designer 会选择默认根，并创建 XML 视图。如果 XML 定义需要的视图超过 400 个，将出现一条消息，指示定义过大。可以在 XML 编辑器中手动创建视图。导入 XML 定义并选择创建自定义视图或跳过生成 XML 视图。

如果从没有全局元素的 XML 架构导入定义，Designer 无法在 XML 定义中创建根视图。Designer 会显示一条消息称没有全局元素。

创建 XML 视图后，则无法更改为视图设置的配置选项。例如，如果创建规范化的 XML 视图，则不能将视图更改为非规范化。必须导入新的 XML 源定义，并选择非规范化选项。

有关 PowerCenter 中的 XML 规模估算的信息，请参阅 [“使用 XML 与 PowerCenter 概览” 页面上 27](#)。有关 PowerCenter 中 XML 处理所适用的局限性的详细信息，请参阅 [“限制” 页面上 27](#)。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

导入部分 XML 架构

导入 XML 架构时，Designer 将创建表示整个架构的 XML 定义。导入包含关联架构的 XML 文件时，可以只为 XML 文件中的元素和属性（而不是架构中的所有组件）创建 XML 视图。Designer 创建仅限于 XML 文件中组件的定义。

要导入部分架构，请导入引用该架构的 XML 文件。选择只为 XML 文件中的元素和属性创建 XML 视图的选项。

导入部分 XML 架构的规则和指南

要导入部分架构，需考虑以下规则和准则：

- Designer 将元数据限制为 XML 文件中的元数据。如果更改 XML 文件并再次导入 XML 架构，则 XML 定义会更改。
- 如果元素或属性在 XML 文件层次结构中出现一次，但在架构层次结构的多个部分出现，Designer 会在 XML 定义中包含元素或属性的所有出现次数。
例如，架构可能有两个地址元素：一个“店面/地址”，一个“员工/地址”。如果导入带有该架构的 XML 文件，而 XML 文件只有“店面/地址”，则 Designer 将创建包括“店面/地址”和“员工/地址”元素的所有“地址”元素。
- 如果 XML 文件包含派生的复杂类型，Designer 会将所有基本类型作为单独的视图纳入 XML 定义。Designer 不会展开派生的类型，以便将基本类型纳入同一视图。
- 如果 XML 文件有多个级别的循环引用，Designer 不在 XML 定义中展开循环引用。

生成实体关系

可以作为实体关系模型生成 XML 层次结构。生成 XML 视图实体关系时，Designer 将完成以下操作：

- 为多次出现的被引用的元素和复杂类型生成视图。
- 创建视图之间的关系。

当 Designer 生成实体关系时，Designer 可根据架构中的关系，为复杂类型、全局元素以及多次出现的元素生成不同的实体。

如果想要创建默认组以外的组，或者如果想要合并来自不同复杂类型的元素，则可以创建自定义的 XML 视图。

在 XML 编辑器中查看 XML 源定义时，可以看到 XML 层次结构中每个元素之间的关系。针对视图之间的每个关系，XML 编辑器将根据视图之间的关系类型生成链接。

要生成实体关系，请执行以下操作：

1. 在 Source Analyzer 中，单击“源”>“导入 XML 定义”。
将打开 XML 向导。
2. 导航到要导入的源，并单击“打开”。
3. 输入文件的名称并单击“下一步”。
4. 选择“实体关系”，然后单击“完成”。
XML 向导将生成使用实体关系的 XML 定义。

生成层次结构关系

创建层次结构关系时，会展开其父元素下每个组件的引用。XML 向导选择默认根，并使用默认设置来创建 XML 组。

要生成层次结构关系，请执行以下操作：

1. 在 Source Analyzer 中，单击“源”>“导入 XML 定义”。
将打开 XML 向导。
2. 导航到要导入的源，并单击“打开”。
3. 输入文件的名称并单击“下一步”。
4. 选择“层次结构关系”。
5. 选择“规范化 XML 视图”或“非规范化 XML 视图”，然后单击“完成”。
XML 向导基于层次结构关系生成 XML 视图。

创建自定义 XML 视图

可以使用 XML 向导创建自定义 XML 视图。创建自定义视图时，可以选择根，并指定如何生成元数据。基于根信息是否适用于要处理的数据，可以选择包括或排除全局元素。例如，如果架构包含有关店面和客户的信息，则可能需要创建只处理客户的 XML 定义。

可以指定要如何生成与视图关联的元数据。可以通过生成实体关系，减少元素、复杂类型和继承的复杂类型的元数据分解。如果不能减少元数据引用，Designer 会生成层次结构关系，并展开父元素下的所有子元素。

要使用 XML 向导创建自定义的视图，请执行以下操作：

1. 在 Source Analyzer 中，单击“源” > “导入 XML 定义”。

将打开 XML 向导。

2. 导航到要导入的源，并单击“打开”。
3. 输入文件的名称并单击“下一步”。
4. 选择“创建自定义 XML 视图”，然后单击“下一步”。

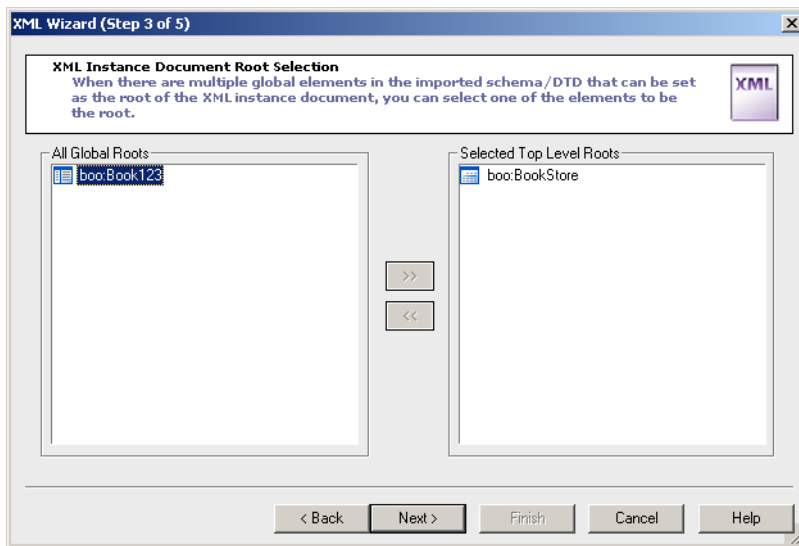
注意：要在 XML 编辑器中手动创建所有 XML 视图，请选择“跳过创建 XML 视图”。XML 向导在存储库中创建架构，但不创建 XML 视图。

5. 从全局根元素的列表中选择根元素并单击“下一步”。
6. 选择减少元素、复杂类型或继承的复杂类型的元数据，并单击“完成”。

选择根元素

创建自定义视图时，可以选择导入的架构中的全局元素，以设置 XML 实例文档的根。全局元素是 XML 架构层次结构中顶部根元素正下方的元素。

下图显示了“根选择”页：



在此示例中，Bookstore 元素作为根被选中，Book123 元素作为根元素被清除。

减少元数据分解

当 Designer 基于使用继承的 XML 架构创建 XML 定义时，Designer 可以在引用元数据的视图内为每个引用的元素或组展开元数据。或者，Designer 可以为引用的对象创建单独的视图，并创建对象与其他视图之间的关系。

如果在 XML 架构内使用引用，则可能希望减少 Designer 包含与引用关联的元数据的次数。XML 向导将提供以下选项以减少元数据引用：

- **减少元素分解。** Designer 为多次出现的任何元素或由一个以上其他元素引用的任何元素创建视图。每个视图可以与定义中的其他视图有多个层次结构关系。

- **减少复杂类型分解。** Designer 为每个引用的复杂类型或多次出现的元素创建 XML 视图。XML 视图可以与其他视图有多个类型关系。如果架构使用继承的复杂类型，也可以减少继承的复杂类型的分解。
- **减少复杂类型继承分解。** 对于任何继承的类型，XML 向导创建类型关系。

减少元数据分解时，Designer 将在生成的 XML 视图之间创建实体关系。

同步 XML 定义

使用 XML 定义时，可能会更改用于创建 XML 定义的文件或源。例如，可能会将新元素或复杂类型添加到 XSD 文件。可以将 XML 定义与任何下列用于创建 XML 定义的存储库定义或文件同步：

- 关系源定义
- 关系目标定义
- 平面文件
- URL
- XML 文件
- DTD
- 架构文件

同步 XML 定义时，Designer 会在架构导航器中更新 XML 架构；但它不更新 XML 定义中的视图。同步 XML 定义后，可以在 XML 编辑器中手动更新视图列。

提示：使用架构文件同步 XML 定义。

要同步 XML 源定义，请执行以下操作：

1. 在 Source Analyzer 中，单击“源”>“导入 XML 定义”。
将打开 XML 向导。
2. 导航到用于创建 XML 定义的存储库定义或文件，并单击“打开”。
3. 在向导的第 1 步中，单击“下一步”。向导将忽略对名称所做的任何更改。
4. 在 XML 向导的第 2 步中，选择同步 XML 定义，并单击“下一步”。
XML 向导跳转至第 5 步。
5. 在 XML 向导的第 5 步，选择想要同步的 XML 定义。
XML 向导将源与所选定义同步。

使用此方法来同步 XML 目标定义。如果修改 XML 源定义，可能还需要同步目标定义。

注意：请验证 XML 定义是否与用于创建定义的源同步。如果要将 XML 定义与未用于创建定义的源同步，Designer 将无法同步定义且会丢失元数据。单击“编辑”>“恢复至已保存的版本”恢复 XML 定义。

编辑 XML 源定义属性

导入 XML 源定义之后，可以编辑定义名称等源定义属性。如果将会话配置为读取文件列表，则可以配置映射将源文件名称写入每一目标行。还可以添加元数据扩展。

要编辑 XML 源定义属性，请执行以下操作：

1. 右键单击 Source Analyzer 工作区中定义的顶部。选择“编辑”。

2. 在“表”选项卡上编辑以下设置：

| 表设置 | 说明 |
|-------|--|
| 选择表 | 显示要编辑的源定义。 |
| 业务名称 | 源定义的描述性名称。可以通过单击“重命名”按钮来编辑“业务名称”。 |
| 所有者名称 | 不适用于 XML 文件。 |
| 说明 | 源的说明。字符限制是 2000 字节/K，其中 K 是存储库代码页中每个字符的最大字节数。输入到业务文档的链接。 |
| 数据库类型 | 源或数据库类型。 |
| 代码页 | 只读。不适用于 XML 源文件。集成服务将所有 XML 源文件转换为 Unicode。 |

3. 单击“列”选项卡。

在“列”选项卡上，可以在定义中查看有关列的信息。要更改列的名称或值，请使用 XML 编辑器。可以查看以下信息：

| 列设置 | 说明 |
|-------|--|
| 选择表 | 要编辑的源定义。 |
| 列名称 | 列的名称。 |
| 数据类型 | PowerCenter 数据类型。 |
| 精度 | 列的长度。 |
| 小数位数 | 数值数据中的小数位数。 |
| 非空 | 指示列是否可以接受空值。 |
| 键类型 | 主键、外键或不是键。 |
| XPath | XML 层次结构中的当前列引用的元素的路径。XPath 对生成的主键或外键不作显示。 |
| 业务名称 | 针对列的用户定义描述性名称。如果“业务名称”在窗口中不可见，请滚动到右侧以查看或修改列。 |

4. 如果将会话配置为读取文件列表，且希望将源文件名称写入每一目标行，请单击“属性”选项卡并选择“添加当前处理的平面文件名端口”。

Designer 将 CurrentlyProcessedFileName 端口添加到“列”选项卡。它是第一组的最后一列。集成服务使用此端口返回源文件的名称。CurrentlyProcessedFileName 端口是默认精度为 256 个字符的字符串端口。

要删除 CurrentlyProcessedFileName 端口，请单击“属性”选项卡并清除“添加当前处理的平面文件名端口”。

5. 单击“元数据扩展”选项卡以创建、编辑和删除用户定义的元数据扩展。

6. 单击“确定”。
7. 单击“存储库” > “保存”，将更改保存到存储库中。

从存储库定义中创建 XML 定义

可以从存储库中的关系或平面文件定义导入 XML 源或目标定义。当从存储库定义中导入 XML 定义时，XML 向导根据所选对象之间的关系创建 XML 层次结构。该 XML 向导创建层次结构的根元素。可以从所创建的组选择根；也可以创建单独的根，将组同其关联。

创建 XML 目标定义时，XML 向导生成将每个组关联到根的键。

要从存储库中的源或目标创建 XML 定义，请执行以下操作：

1. 在 Source Analyzer 中，单击“源” > “导入 XML 定义”。
-或-
在 Warehouse Designer 中，单击“目标” > “导入 XML 定义”。
2. 在“XML 导入”对话框中，单击“非 XML 源”或“非 XML 目标”。
3. 从源或目标列表中选择定义。单击“箭头”按钮以将定义添加到选定的源列表。
可以选择一个以上的输入和输入类型。如果定义是通过主键和外键关联的，则 XML 向导在生成层次结构时会使用键来关联组。
4. 要为根元素创建单独的组，请输入可选的 XML Rootname。
根名称默认为 XRoot。根组包含所有其他组。如果想要将其他组之一用作根，请删除根名称。XML 向导为所选的每个输入源或目标定义创建组，并在每个组中生成主键。XML 向导在每个组中创建一个外键。外键指向根组链接键。
5. 单击“打开”。
将显示 XML 向导。
6. 使用 XML 向导来生成源或目标组。

XML 源故障排除

如何将具有相同父元素的两个多次出现的元素置入一个视图中？例如，我需要把 EMPLOYEE 的所有元素都放在一个视图中：

```
<!ELEMENT EMPLOYEE (EID, EMAIL+, PHONE+)>
```

EMAIL 和 PHONE 属于相同的父元素，但它们不属于同一个父链。不能将它们放在同一非规范化视图中。要将 Employee 的所有元素放在一个视图中，可以转换其中一个多次出现的元素。

请按照这些步骤将多次出现的两个元素添加到同一视图：

1. 创建 EMPLOYEE 视图。
2. 将 EID 和 EMAIL 元素添加到 EMPLOYEE 视图。
3. 对于想要包括在视图中的 EMAIL 的出现次数进行转换。EMAIL 的每次出现都成为视图中出现一次的元素。
4. 添加 PHONE 元素。

我在 DTD 中定义了下列元素：

```
<!ELEMENT EMPLOYEE (EMPNO, SALARY)+>
```

如何在同一视图中实现 EMPNO 和 SALARY 的匹配？

DTD 示例含糊不清。此定义等效于以下内容：

```
<!ELEMENT EMPLOYEE (EMPNO+, SALARY+)>
```

在 DTD 示例中，EMPLOYEE 有多次出现的元素 EMPNO 和 SALARY。同一视图中不能有多次出现的两个元素。

请使用以下解决方法之一：

- **重写元素定义，以使定义明确。**

可以按以下方式定义 EMPLOYEE 元素：

```
<!ELEMENT EMPLOYEES (EMPLOYEE+)>
<!ELEMENT EMPLOYEE (EMPNO, SALARY)>
```

使用此语法时，是为每个 EMPLOYEE 定义一个 EMPNO 和一个 SALARY。EMPLOYEE 视图包含两个元素。将 EMPLOYEE 加入为 EMPLOYEES 中多次出现的元素。

- **将这些元素置于单独的视图中，且两次在映射中使用源定义。**

当 EMPNO 和 SALARY 位于不同的视图中时，仍可以将数据合并到一个映射中。使用同一源定义的两个实例并使用联接器转换。

我导入的 XML 文件具有以下结构：

```
<Bookstore>
    <Book>Book Name</Book>
    <Book>Book Name</Book>
    <ISBN>051022906630</ISBN>
</Bookstore>
```

当导入该 XML 文件时，Designer 丢弃了 ISBN 元素。为什么会这样？怎样才能使 Designer 包括 ISBN 元素？

- **使用架构导入 XML 定义。**使用 XML 文件导入 XML 定义时，因为第一个元素没有子元素，Designer 将第一个元素读取为简单内容。Designer 从第二个 Book 实例中丢弃 ISBN 子元素。如果使用架构导入定义，Designer 将使用架构定义来确定如何读取 XML 数据。
- **验证 XML 文件准确地表示了关联的架构。**如果使用 XML 文件导入源定义，请验证 XML 文件准确地表示了相应 XML 架构的结构。

有关 PowerCenter 中的 XML 规模估算的信息，请参阅 [“使用 XML 与 PowerCenter 概览” 页面上 27](#)。有关适用于 PowerCenter 中 XML 处理的限制的信息，请参阅 [“限制” 页面上 27](#)。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

使用 XML 编辑器

使用 XML 编辑器概览

当您在 Designer 中导入 XML 定义时，使用默认视图、自定义视图或不使用视图创建一个 XML 定义。创建一个 XML 定义后，可以使用 XML 编辑器对定义进行更改。

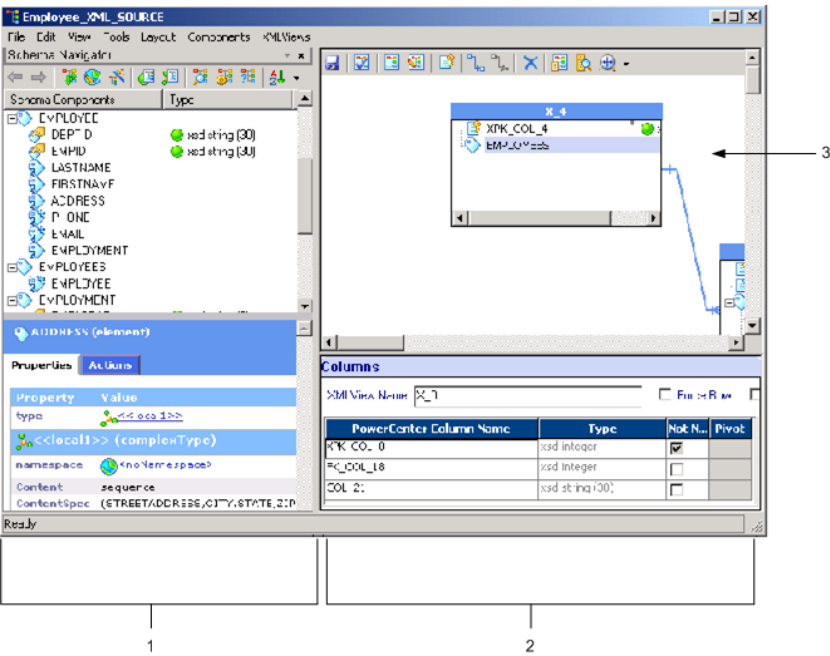
使用 XML 编辑器创建视图、修改组件、添加列以及维护工作区中的视图关系。当您更新 XML 定义时，Designer 将更改传播到包括该源的任何映射。对 XML 定义的某些更改可能会使映射失效。

注意: 如果对用来创建 XML 定义的源进行了大量更改，您可以将定义同步到新源，而不是手动编辑定义。

XML 编辑器包含以下窗口：

- 导航器
- XML 工作区
- 列窗口

下图显示了 XML 编辑器：



1. 导航器
2. 列窗口
3. XML 工作区

XML 编辑器使用图标来表示 XML 组件类型。要查看描述 XML 编辑器中的图标的图例，请单击“视图”>“图例”。

XML 导航器

导航器以层次结构形式显示架构，并提供有关所选组件的信息。您可以在导航器中按类型、层次结构或命名空间对组件进行排序。您也可以展开组件以在层次结构中查看组件下的组件。

导航器工具栏提供了大多数导航器功能的快捷方式。工具栏还提供了导航箭头，单击箭头可以在层次结构中快速查找以前显示的组件。

导航器具有以下选项卡：

- **“属性”选项卡。**显示有关所选组件的信息，如组件类型、长度和出现。
- **“操作”选项卡。**提供选项列表，以查看有关所选组件的详细信息。

“属性”选项卡

“属性”选项卡显示有关您在导航器中选择的组件的信息。如果该组件是一个复杂元素，您可以在架构中查看元素属性，如命名空间、类型和内容。当您查看一个简单的元素或属性时，“属性”选项卡显示元素的类型和长度。“属性”选项卡还显示批注。

如果您从 XML 文件导入定义，则可以通过“属性”选项卡编辑数据类型和基数。如果您通过 DTD 文件创建定义，您可以编辑组件类型。

如果架构使用命名空间，您可以更改命名空间、前缀以及命名空间的架构位置。前缀标识属于命名空间的元素或属性。默认命名空间中的元素和属性没有前缀。您可以为 XML 目标选择一个命名空间作为默认命名空间。

操作选项卡

“操作”选项卡列出了您用来查看有关所选组件的详细信息的选项。您也可以从“操作”选项卡恢复对组件所做的更改。

根据您选择的组件的属性，以下选项显示在“操作”选项卡上：

- **ComplexType 引用。**显示属于这种类型的架构组件。
- **ComplexType 层次结构。**显示从选定组件派生的复杂类型。
- **SimpleType 引用。**显示此类型的所有组件。
- **传播 SimpleType 值。**将长度和小数位数的值传播到该 SimpleType 的所有组件。
- **元素引用。**显示引用所选元素的组件。
- **子组件。**显示所选组件使用的全局架构组件。
- **还原 simpleType。**如果您更改了类型、长度和精度值，将它们更改回原始值。
- **XML 视图引用。**显示引用所选组件的所有 XML 视图和列。

XML 工作区

XML 工作区显示 XML 视图和视图之间的关系。您可以在工作区中创建 XML 视图，并定义视图之间的关系。

XML 工作区工具栏提供了您可以在工作区中执行的大部分功能的快捷方式。

您可以通过以下方式修改 XML 工作区的大小：

- **隐藏“列”窗口。**单击“视图”>“列属性”。
- **隐藏导航器。**单击“视图”>“导航器”。
- **缩小工作区。**单击“工作区”工具栏上的“缩放”按钮。

列窗口

“列”窗口显示工作区中视图的列。使用“列”窗口为添加的列命名。如果您使用透视列，使用“列”窗口选择和重命名多次出现的元素的出现。您还可以指定选项，如非空、强制行、层次结构或类型关系行以及非递归行。这些选项影响集成服务将数据写入 XML 目标的方式。

创建和编辑视图

使用 XML 编辑器来创建自定义的 XML 视图或编辑使用 XML 向导创建的 XML 视图。要创建视图，定义视图并指定视图中的列。如果架构具有多次出现的元素，则可以指定要包含在视图中的元素出现。您还可以为 XML 目标文件名创建特殊端口，为 XML 解析器和 XML 生成器转换创建传递端口。

请完成以下任务来创建和编辑 XML 视图：

- **创建 XML 视图。**将视图添加到工作区。
- **将列添加到视图。**在视图中创建新列。
- **从视图中删除列。**从视图中删除列。
- **展开一个复杂类型。**选择要添加到视图的派生复杂类型。
- **导入 anyType 元素。**导入 anyType 元素。
- **将内容应用到 anyAttribute 元素。**定义为 anyAttribute 元素的内容。
- **使用 anySimpleType 元素。**在 XML 定义中使用 anySimpleType 元素。
- **添加传递端口。**添加端口以通过 XML 转换传递非 XML 数据。
- **添加 FileName 列。**添加列来为每个目标 XML 文件生成一个新文件名。

创建 XML 视图

您可以在 XML 工作区中创建视图。当创建一个不包含视图的 XML 定义时，XML 编辑器将显示一个空白工作区。您可以创建一个视图并向视图添加列和视图行。

在工作区中创建新的 XML 视图：

1. 在 XML 编辑器中打开 XML 定义。
2. 单击“XML 视图” > “创建 XML 视图”。
3. 在“列”窗口中输入视图的名称。
该名称显示在工作区中的 XML 视图上。
4. 在要从中创建视图的架构导航器中突出显示该节点。
5. 单击“组件” > “XPath 导航器”。
6. 将 XPath 导航器中的“列模式”设置为“视图行模式”，以添加视图行。
7. 在导航器中选择该元素并将该元素拖动到工作区的视图中。

XML 编辑器以蓝色突出显示视图行。

第一次将一列添加到视图中时，Designer 将验证该列是否可以作为视图行。即使您没有指定要添加视图行，也可能发生这一情况。

8. 要将视图行更改为另一列，在视图中右键单击相应的行，选择“设置为视图行”。

将列添加到视图

可以将列添加到 XML 工作区中的 XML 视图。要添加一列，请从 XPath 导航器中选择该列。

当以下条件为真时，可以将一列添加到 XML 视图：

- 组件路径从作为该视图的视图根的架构中的元素开始。
- 该组件不是 enclosure 元素。
- 该组件并不违反规范化或透视规则。例如，您不能向一个视图添加多个多次出现元素。
- 您可以作为简单或复杂类型添加混合内容元素。
- 两个视图都可以共享同一列，并且一个视图可能包含相同的多个列。

要将列添加到视图中，请按以下步骤操作：

1. 在工作区中选择 XML 视图。
2. 在导航器中突出显示包含要添加的组件的父元素。
3. 单击“组件” > “XPath 导航器”。

XPath 导航器将显示在“导航器”窗口中。

4. 单击“模式”按钮，选择添加列或视图行。
5. 选择“高级”以将透视的列添加到视图。

透视的列是多次出现元素的一个出现。您可以在“高级”模式下添加单次出现的列。

注意：无法在 XML 目标定义中创建透视的列。

6. 将列从 XPath 导航器拖动到 XML 工作区中适当的视图中。您可以一次选择多列。

XML 编辑器将验证您添加的列。如果列对于该视图无效，当您拖动该列时消息将显示在状态栏中。当您新列添加到视图将显示它们。

添加一个透视的列

XML 定义中的透视的列是一个多次出现元素，它构成视图中元素出现的分隔列。您可以通过 XML 定义中任何多次出现元素创建透视的列。您还可以透视图行中的元素。

如果您将透视的列添加到视图，“列”窗口中会显示一个默认出现编号。这个编号表明在列中使用元素的哪次出现。您可以更改出现编号，或作为新列添加该元素的更多出现。如果您不重命名这些列，XML 编辑器会将一个序列号添加到每个透视的列名。

注意：如果透视值是视图行的一部分，则不能更改透视值。

要将透视的列添加到视图，请按以下步骤操作：

1. 在工作区中选择 XML 视图。
2. 在导航器中突出显示要透视的元素。
或者在要透视的元素的父链中突出显示任何元素。
3. 单击“组件” > “XPath 导航器”。
4. 选择“高级”模式。
5. 将通过 XPath 导航器透视的列拖动到 XML 工作区中的视图中。

Designer 在“列”窗口中添加默认出现编号。这个编号表明在列中使用元素的哪次出现。
在下图中，针对 Product 的第三次出现，视图包含 Sales 的两次出现：

| Columns | | | | |
|-------------------------|-----------------|--------------------------|---------------------------------|--|
| XML view Name | | Product_Sales_By_Store | | |
| View Row XPath | | STORE/SNAME | | |
| PowerCenter Column Name | Type | Not Null | XPath | |
| FIRST_HALF_SALES | xsd:string (30) | <input type="checkbox"/> | .../PRODUCT[3]/SALES[1]/@REGION | |
| SECOND_HALF_SALES | xsd:string (30) | <input type="checkbox"/> | .../PRODUCT[3]/SALES[2]/@REGION | |

第一次 Sales 出现是在名为 First_Half_Sales 的列中。第二次 Sales 出现是 Second_Half_Sales。区域是一个属性。

- 6. 单击 XPath 链接可更改元素的出现编号。
将显示“为 XPath 指定查询谓词”窗口。
- 7. 选择要编辑的多次出现元素。
- 8. 在“编辑透视”框中更改出现编号，单击“确定”。

从视图中删除列

您可以从视图中删除列。

要从视图中删除某一列，请按以下步骤操作：

- 1. 右键单击工作区中的视图中的列。
- 2. 选择“删除此列”。
XML 编辑器会提示您确认要删除该列。
- 3. 单击是进行确认。
XML 编辑器从视图中删除该列。然而，该列仍然在 XPath 导航器层次结构中。

删除透视的列

要删除透视的列的一次出现，请从“列”窗口中选择并删除该列。

要删除透视的列，请按以下步骤操作：

- 1. 在“列”窗口中右键单击您要删除的列。
- 2. 单击“删除”>“透视”。
- 3. 单击“是”以确认删除。

扩展复杂类型

架构可以定义一个复杂类型，以作为多个类型的基本类型。例如，出版物可以是一本杂志或报纸。当创建 XML 视图时，您可以选择使用出版物作为杂志或报纸类型。

当您在 XPath 导航器中查看一个复杂类型时，您可以查看派生类型。

要展开一个复杂类型，请按以下步骤操作：

- 1. 在 XPath 导航器中突出显示该复杂类型。
“扩展复杂类型”列表显示派生类型。

2. 选择要使用的类型。

如果将组件添加到 XML 定义，定义包含您选择的类型。

导入 anyType 元素

您可以导入包含 anyType 元素的 XML 架构。类型 anyType 的元素可以包含显示在 XML 文档中的任何数据类型。例如，XML 架构的以下部分包含属于 anyType 的元素 Document：

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Publication" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="Document" type="xsd:anyType"/>
</xsd:schema>
```

导入带有 anyType 元素的架构时，anyType 元素作为 anyType 显示在架构导航器中。Designer 不会为类型 anyType 的元素创建一个端口。

您必须在 Designer 中将 anyType 元素更改为全局复杂类型，以在 PowerCenter 中使用 anyType 元素。

要将 anyType 元素更改为另一个全局复杂类型，请按以下步骤操作：

1. 在架构导航器中突出显示 anyType 元素。
anyType 属性将显示在架构导航器的“属性”选项卡中。
2. 单击 anyType 属性。
如果该架构不包含全局复杂类型，Designer 会显示一个错误，指明没有全局复杂类型可供选择。
3. 选择一个复杂类型并单击“确定”。

将内容应用到 anyAttribute 或者 ANY 元素

您可以导入包含 anyAttribute 或 ANY 内容元素的 XML 架构。要在视图中使用该元素，必须在 XML 编辑器中定义该元素的内容。当您导入包含 anyAttribute 或 ANY 内容元素的架构时，该元素显示在架构导航器中，其中没有属性。元素并不显示在视图中。

例如，以下架构元素包含 ANY 内容：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

要将内容应用于 anyAttribute 或 ANY 内容，请按以下步骤操作：

1. 在架构导航器中单击元素内容链接。
将显示“编辑 Any 或 anyAttribute 类型内容”对话框。
2. 单击“添加类型”。
将显示新行。
3. 从 XML 定义的架构的有效元素列表选择一个元素。

4. 选择一个基数，并单击“确定”。

替换类型显示在架构导航器中。

在 XML 编辑器中使用 anySimpleType

anySimpleType 元素可以包含任何原子数据类型，如字符串、整数、小数或日期。当您不知道 XML 实例文档中的元素的数据类型时，在架构中使用 anySimpleType。

如果将元素定义为 anySimpleType，当您导入架构时，Designer 将为元素创建 anySimpleType 列。当您在映射中使用列时，XML 源限定符将这一类型映射到一个字符串。

添加一个传递端口

您可以将传递端口添加到 XML 解析器或 XML 生成器转换来通过转换传递非 XML 数据。当您在转换中定义端口时，您将端口添加到 XML 解析器转换中的 DataInput 组或 XML 生成器转换中的 DataOutput 组。

一旦您生成传递端口，您将添加另一个端口通过转换传递数据。此端口是引用端口。在 XML 解析器转换中，传递端口将数据传入转换，而引用端口将数据传出转换。在 XML 生成器中，传递端口将数据传出转换，而引用端口将数据传入转换。

如果您在 XML 定义中具有传递端口，可以确定相应的引用端口。

要确定传递端口的引用端口，请按以下步骤操作：

1. 右键单击传递端口。
2. 选择“导航到”>“引用列”。

XML 编辑器在工作区中突出显示引用列。

添加 FileName 列

当您运行会话时，每次数据中出现一个新根值时，集成服务输出一个新的目标 XML 文件。您可以将 FileName 列添加到 XML 视图，从而为每个 XML 文件生成唯一的文件名。该文件名将替代会话属性中的默认输出文件名。

当您使用 FileName 列时，在映射中设置表达式转换或其他转换，以生成传递到 FileName 列的唯一文件名。

要在 XML 视图中创建 FileName 列，请按以下步骤操作：

1. 在 XML 编辑器中右键单击该视图。
2. 选择“创建 FileName 列”。

XML 编辑器在视图中创建一个新字符串列。

3. 在“列”窗口中更改列名称。
4. 退出 XML 编辑器。

该列显示在 XML 定义中。XPath 是 \$Filename。

注意：当使用 XML FileName 端口时，您需要为该文件指定目录名称。

创建 XPath 查询谓词

通过在 XML 层次结构中视图行下方的视图行或列上创建 XPath 查询谓词，您可以在会话中筛选 XML 数据。当您创建 XPath 查询谓词时，XML 编辑器将 XPath 查询谓词添加到视图行 XPath。

当您在 XML 编辑器中创建 XPath 查询谓词时，XML 编辑器提供元素、属性、运算符和函数以生成查询。您可以选择组件，输入组件，或将组件复制到查询中。XML 编辑器将验证您创建的每个查询。

您可以查询元素或属性的值，或者可以验证是否存在某一元素或属性。

查询属性元素的值

您可以创建 XPath 查询谓词，以在视图中筛选元素或属性值。例如，要提取部门 100 的员工，请创建以下 XPath 查询谓词：

```
EMPLOYEE [./DEPT = '100']
```

查询表达式在括号中。部门 XPath 缩写为 “./” 以表示该路径跟随 “Employee”。

如果姓是 Smith，以下 XPath 查询谓词提取员工：

```
EMPLOYEE[./NAME/LASTNAME='SMITH']
```

姓名是 Employee 的一个子元素，并且是 Lastname 的父元素。

在 XPath 查询谓词中使用布尔或数值运算符。您还可以在查询中使用字符串、数值和布尔函数。

查询混合内容

如果 XML 元素包含一个值且包含子元素，XML 元素包含混合内容。您可以创建 XPath 查询谓词，以筛选由子元素分开的元素值。然而，集成服务不计算混合内容中第一个子元素后出现的谓词。

例如，XML 文件可能包含具有混合内容的 NAME 元素：

```
<NAME>
  Kathy
  <MIDDLE> Mary </MIDDLE>
  Russell
</NAME>
```

元素 NAME 具有值 “Kathy”、子元素 “MIDDLE” 和第二个值 “Russell”。NAME 列的值是 “KathyRussell”。然而，集成服务评估 NAME “Kathy”。

以下查询为假：

```
EMPLOYEE [./NAME = 'KathyRussell']
```

以下查询为真：

```
EMPLOYEE [./NAME = 'Kathy']
```

布尔运算符

在 XPath 查询谓词中使用以下布尔运算符：

```
and or < <= > >= = !=
```

使用以下 XPath 查询谓词提取部门 100 中姓氏为 Jones 的员工：

```
EMPLOYEE [./DEPT = '100' and ./ENAME/LASTNAME = 'JONES']
```

数值运算符

在 XPath 查询谓词中使用以下数值运算符：

+ - * div mod

使用以下 XPath 查询谓词提取价格大于成本加税的产品：

```
PRODUCT[./PRICE > ./COST + ./TAX]
```

函数

可在 XPath 查询谓词中使用以下类型的函数：

- **字符串**。使用字符串函数测试子字符串值，连接字符串，或者将字符串转换为其他字符串。以下 XPath 查询谓词确定员工的全名是否等于姓氏和名字相连：

```
EMPLOYEE[./FULLNAME=concat(./ENAME/LASTNAME,./ENAME/FIRSTNAME)]
```

- **数值**。将数值函数与元素和属性值一起使用。数值函数对数值进行操作并返回整数。例如，以下 XPath 查询谓词舍入折扣并测试结果是否大于 15：

```
ORDER_ITEMS[round(./DISCOUNT > 15)]
```

- **布尔型**。布尔函数返回真或假。使用它们来测试元素，检查语言属性，或者强制返回 True 或 False 结果。例如，如果字符串值是大于零，则字符串为 True：

```
boolean(string)
```

测试元素或属性

您可以确定元素或属性是否出现在 XML 数据中。以下 XPath 查询谓词确定部门是否有部门名称属性：

```
COMPANY/DEPT[@DEPTNAME]/EMPLOYEE
```

Deptname 是一个属性。在 XML 查询谓词表达式中，属性前面加 “@”。

当运行会话时，如果员工的部门有部门名称，集成服务通过 XML 源提取员工数据。否则，集成服务不会提取该员工数据。

XPath 查询谓词规则和准则

创建 XPath 查询谓词时，可以使用以下规则和准则：

- 您为视图行中的任何元素配置 XPath 查询谓词。

例如，如果视图行是 Company/Dept，您可以创建以下 XPath 查询谓词：

```
COMPANY[./DEPT=100]
```

- 您可以与内容相匹配。
- 如果列出现在视图 XML 层次结构中的视图行下面，并且列 XPath 中包含该视图行，您可以向列中添加 XPath 查询谓词。

例如，如果视图行是 Product/Toys[1]，您可以创建以下 XPath 查询谓词：

```
Product/Toys[1][./Sales > 100]
```

以下示例为 Product/Toys[1] 视图行显示无效的 XPath 查询谓词：

```
Product/Toys[2][./Sales > 100]
```

Product/Toys[1] 是视图行。您不能使用 Product/Toys[2]。

- 使用单次出现的元素或属性。您不能在多次出现的元素上创建 XPath 查询谓词。

- 因为 enclosure 元素不包含任何值，无法在 enclosure 元素上创建 XPath 谓词查询。

创建 XPath 查询谓词的步骤

您可以在 XML 源定义中为视图创建 XPath 查询谓词。

要创建 XPath 查询谓词，请按以下步骤操作：

1. 在 XML 编辑器中打开 XML 源定义。
 2. 在 XML 编辑器工作区中选择一个视图。
在“列”窗口中显示视图列。
 3. 单击“视图行 XPath”。
- 将显示“为 XPath 指定查询谓词”窗口。您可以在 XPath 谓词窗口中输入一个 XPath 查询谓词，或从对话框的选项卡中选择元素、运算符和函数。
4. 单击“子元素和属性”选项卡显示可以添加到一个 XPath 查询谓词的元素和属性。
 5. 双击元素或属性以将其添加到 XPath 查询谓词。
该组件将出现在面板中。
 6. 单击“运算符”选项卡。
- 使用运算符创建表达式。您可以将元素与值或其他元素进行比较，或者您可以创建数学表达式。
- 下表描述了您可以添加到 XPath 查询谓词的运算符：

| 运算符 | 说明 |
|-----|-------|
| + | 加 |
| - | 减 |
| * | 乘 |
| div | 除 |
| mod | 模数 |
| and | 布尔与 |
| or | 布尔或 |
| < | 小于 |
| <= | 小于或等于 |
| > | 大于 |
| >= | 大于或等于 |

| 运算符 | 说明 |
|-----|-----|
| = | 等于 |
| != | 不等于 |

7. 双击运算符，将该运算符添加到 XPath 查询谓词。
8. 要添加 PowerCenter XPath 查询谓词函数，请单击“函数”选项卡。函数接受参数并返回值。
9. 通过“子元素和属性”选项卡选择另一个元素或属性，或在“XPath 谓词”窗口输入一个值来完成表达式。
10. 单击“验证”来验证 XPath 查询谓词。

维护视图关系

您可以完成以下任务来在 XML 编辑器中维护视图关系：

- **创建视图之间的关系。**在工作区中定义视图之间的关系。
- **创建类型关系。**定义视图中的列和工作区中的类型视图之间的类型关系。
- **重新创建实体关系。**使用与 XML 向导中相同的选项生成视图和关系。

创建视图之间的关系

使用 XML 编辑器来创建层次结构或视图之间的继承关系。您不能创建 XML 源和非 XML 源之间的关系。

要创建 XML 视图之间的关系，请按以下步骤操作：

1. 在工作区中右键单击子 XML 视图的顶部。
2. 选择“创建关系”。
3. 将指针移动到父视图，以建立一种关系。
当您移动指针时，在视图之间会出现一个链接，而 XML 编辑器会验证该关系是否有效。如果关系无效，在状态栏中会显示一条错误消息。
4. 如果未显示任何错误消息，请单击父视图以建立关系。
XML 编辑器创建关系，并添加相应的外键。
5. 要查看关系的详细信息，请将光标放在视图之间的链接上。
编辑器将显示关系类型以及主键和外键。

创建类型关系

您可以创建视图中的列和类型视图之间的类型关系。如果该列已经透视，您可以选择包含在关系中的出现。

要创建类型关系，请按以下步骤操作：

1. 在视图中右键单击您想要使用的列。
2. 选择“创建关系”。
3. 如果该列已经透视，选择要使用的出现。

4. 将指针移动到类型视图，以建立一种关系。

当您移动指针时，在视图之间会出现一个链接，而 XML 编辑器会验证该关系是否有效。如果关系无效，在状态栏中会显示一条错误消息。

5. 如果未显示任何错误消息，请单击父视图以建立关系。

XML 编辑器创建该关系。

重新创建实体关系

您可以为 XML 定义重新创建实体关系。使用“重新创建实体关系”对话框来生成新的 XML 视图，所使用的选项与 XML 向导中的选项相同。当您重新生成视图时，可以选择保持现有视图。XML 定义包含所有视图。

要重新创建一个 XML 定义的实体关系：

1. 在 XML 编辑器中打开 XML 定义。

2. 在导航器中突出显示 XML 根。

如果突出显示另一个组件，XML 编辑器会将该组件用作根。

3. 单击“XML 视图” > “创建实体关系”。

4. 从以下选项中选择以减少元数据分解：

- **减少元素分解。**对于任何多次出现或引用元素，XML 向导创建一个具有多个层次结构关系的 XML 视图。
- **减少复杂类型分解。**对于任何多次出现或引用的复杂类型，XML 向导使用多个类型关系创建一个 XML 视图。如果架构使用继承的复杂类型，也可以减少继承的复杂类型的分解。
- **减少复杂类型继承分解。**对于任何继承的类型，XML 向导使用多个类型关系创建一个 XML 视图。
- **分享现有的 XML 视图。**请不要删除现有的 XML 视图。
- **刷新共享的 XML 视图。**保存现有视图，但对其进行更新。

5. 单击“下一步”。

显示“重新创建实体关系”对话框。

6. 要显示子组件，请选择一个共享元素或复杂类型并单击名称。

7. 要排除子组件，请在“排除子组件”窗格中清除该元素。

要生成一个新视图，请选择元素或复杂类型。当创建新实体关系时，以该元素作为视图根生成视图。

查看架构组件

完成以下任务可以在导航器和工作区查看组件：

- **更新命名空间。**更改 XML 目标中架构的位置或默认的命名空间。
- **导航至组件。**通过从 XML 编辑器窗口中的一个组件导航到另一个组件或区域来查找组件。
- **排列工作区中的视图。**按层次结构排列工作区中的视图。您可以将视图安排到工作区中的层次结构排列。要排列工作区中的视图，单击“布局” > “排列”，或者右键单击工作区并选择“排列”。
- **搜索组件。**在导航器或工作区中查找组件。
- **显示简单或复杂类型的层次结构。**查看 XML 架构中的简单或复杂类型层次的层次结构。
- **查看 XML 元数据。**查看 XML 编辑器通过 XML 定义创建的 XML 文件、架构或 DTD。

- **预览 XML 数据。**使用来自外部 XML 文件的示例数据显示 XML 视图。
- **验证 XML 定义。**验证 XML 定义和视图错误。

更新命名空间

当您创建 XML 定义时，您可以在 XML 编辑器中更改命名空间前缀和架构位置。您还可以向命名空间添加架构。

如果创建包含一个或多个命名空间的目标 XML 定义，可以选择一个默认命名空间。当运行一个会话时，集成服务通过没有命名空间前缀的默认命名空间写入元素和属性。

请勿使用“xml”或“xmlns”作为命名空间前缀。默认情况下，“xml”前缀指向 <http://www.w3.org/XML> 命名空间。“xmlns”将元素链接到 XML 架构中的命名空间。

注意：您不能使用 XML 编辑器添加一个命名空间。

要更新命名空间，请按以下步骤操作：

1. 在导航器中选择一个元素。
2. 单击“属性”选项卡。
3. 单击“命名空间”链接。
将显示“编辑命名空间前缀和架构位置”对话框。
4. 要更改前缀或架构位置，请选择要更改的文本并输入新值。
5. 要将多个架构添加到命名空间中，选择一个架构位置并单击“添加”。
空白架构显示在命名空间“架构位置”中。
6. 要删除一个架构，请突出显示该架构位置并单击“删除”。
7. 要在 XML 目标中创建一个默认命名空间，请选择一个命名空间。

您选择的命名空间中的所有组件都属于 XML 目标中的默认命名空间。当您运行会话时，集成服务不在 XML 目标文件中写入默认的命名空间前缀。

导航至组件

要快速查找组件，在大型 XML 定义中选择一个从中导航的工作区组件并选择导航选项。例如，如果您在视图中单击外键，可以导航到相关主键或导航到“列”窗口中的列。您可以在工作区、“列”窗口和导航器中的组件之间导航。

要导航至组件，请按以下步骤操作：

1. 在工作区中或“列”窗口中右键单击组件。
2. 选择“导航到”。
3. 选择可用的选项。

您可以选择以下选项，具体取决于您选择的要从中导航的组件：

- **架构组件。**在导航器中突出显示该组件。
- **PowerCenter 列。**在“列”窗口中突出显示列。
- **主键。**突出显示与选定外键相关联的主键。
- **引用列。**突出显示与 XML 解析器或生成器转换中的传递端口相关联的引用列。

- **XPath 导航器。**显示到所选组件的路径。
- **XML 视图。**在工作区中突出显示包含来自“列”窗口的所选列的视图。

搜索组件

您可以在 XML 定义中搜索组件。XML 编辑器显示组件的所有出现。您可以单击搜索结果，XML 编辑器将在架构或视图中突出显示该组件。您可以在工作区中搜索 XML 架构或 XML 视图。

搜索 XML 架构

您可以按名称、类型和命名空间搜索组件。您可以指定要搜索的属性，如批注、固定或默认值或长度。您可以使用枚举属性搜索合法值。

要搜索架构中的组件，请按以下步骤操作：

1. 单击“编辑”>“在架构中搜索”。
- 显示“搜索组件”对话框。
2. 要按组件属性搜索，请单击“高级选项”链接，以查看可以搜索的属性。
3. 请输入要搜索的名称、类型或属性。
4. 如果您要在特定命名空间中搜索，单击“全部”，并从列表选择一个命名空间。
5. 单击“搜索”。
- 搜索结果显示在对话框中。
6. 单击搜索结果，可在“属性”窗口中查看组件。

在 XML 视图中搜索

您可以在 XML 视图中搜索组件和列。如果您按组件名称搜索，您可以找到定义中的所有相关联的列。例如，如果您搜索组件“Number”，结果包含包括组件“Number”的视图和列。

要使用部分键进行搜索，请输入列或组件名称的前几个字符。

要搜索 XML 视图中的组件，请按以下步骤操作：

1. 单击“编辑”>“搜索 XML 视图”。
- 将显示“搜索 XML 视图和列”对话框。
2. 输入搜索条件。
- 您可以搜索所有列类型、常规列类型、生成的键或其他类型。其他类型包括 FileName 列、引用端口和传递字段。
3. 单击“搜索”。
- 搜索结果显示在对话框中。
4. 要清除搜索字段，请单击“新建搜索”。

查看一个简单或复杂类型层次结构

您可以查看架构定义中 XML 简单类型的层次结构。要查看简单类型的层次结构，请单击“视图”>“SimpleType 层次结构”。窗口显示简单类型的层次结构。

可以显示架构定义中复杂类型的层次结构。要查看复杂类型的层次结构，请单击“视图”>“ComplexType 层次结构”。窗口显示架构中复杂类型的层次结构。从“ComplexType 层次结构”窗口中选择一个组件，以导航至架构中的组件。

查看 XML 元数据

您可以作为 XML、DTD 或 XML 架构文件查看 XML 定义。

要查看 XML 元数据，请按以下步骤操作：

1. 要作为示例 XML 文档查看元数据，请在导航器中选择全局组件。
2. 单击“视图”>“XML 元数据”。
- 将显示“查看 XML 元数据”对话框。
3. 选择作为 XML 文件、DTD 文件或 XML 架构显示 XML 定义。
- 如果您使用多个命名空间，请选择要使用的命名空间。
- 默认应用程序或文本编辑器显示该元数据。
4. 要保存 XML、DTD 或 XML 架构文件的副本，请单击“另存为”。
5. 输入新文件名。
- 如果默认显示应用程序是一个文本编辑器，需要包括带有该文件名的相应文件后缀。后缀是 .xml、.dtd 或 .xsd，这取决于您正在使用的文件类型。

验证 XML 定义

您可以在 XML 编辑器中验证 XML 定义。

要验证 XML 定义，请按以下步骤操作：

1. 在 XML 编辑器中打开该定义。
2. 在工作区中单击。
3. 单击“XML 视图”>“验证 XML 定义”。
- “验证”窗口显示结果。

设置 XML 视图选项

默认情况下，集成服务为在视图行中包含数据的每个视图生成行。要更改集成服务为部分 XML 源定义视图生成行的方式，请在“XML 列窗口”中选择“XML 视图选项”。

使用以下方法来确定集成服务通过 XML 源生成行的时间：

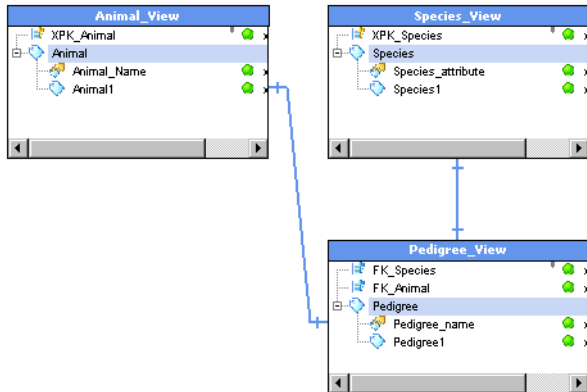
- **在视图中生成所有外键。**默认情况下，集成服务为视图中的一个外键生成值。如果一个视图有多个外键，其他外键包含空值。您可以为所有外键生成值。
- **停止循环关系中的递归读取。**默认情况下，集成服务为循环关系中数据的所有出现生成行。您可以为递归数据的首次出现生成一行。
- **如果存在父视图，为子视图生成行。**默认情况下，集成服务为视图行中包含数据的所有视图创建行。仅当父视图包含数据时，可以为子视图生成一行。
- **为没有视图行数据的视图生成行。**默认情况下，当视图行包含数据时，集成服务为该视图生成数据。您可以为视图行中没有数据的视图生成行。

- **为类型关系中的特定复杂类型生成行。**默认情况下，集成服务为 XML 定义中的所有视图生成行。您可以为类型关系中的特定视图生成行。

生成所有层次结构外键

默认情况下，集成服务为视图中的一个外键生成值。如果一个视图有多个外键，其他外键包含空值。您可以选择为视图中的所有外键生成键值。选择“所有层次结构外键”选项。

下图显示了具有两个外键 FK_Species 和 FK_Animal 的 Pedigree_View：



如果您为 Pedigree_View 选择“所有层次结构外键”选项，集成服务为 FK_Species 和 FK_Animal 生成键值。

下图显示了使用“所有层次结构外键”选项的 Pedigree_View 的示例数据：

| Preview data with file MULTIPLE_FK_EXAMPLE_DATA.xml | | | | |
|---|------------|-----------|-----------|--|
| Pedigree_... | FK_Species | FK_Animal | Pedigree1 | |
| Plains Cheeta | 1 | 1 | 100 | |
| African Lion | 2 | 2 | 200 | |

Number of rows: 2

如果清除“所有层次结构外键”选项，集成服务为一个外键列生成键值。在此示例中，集成服务为 FK_Species 生成值，因为在 XML 层次结构中 Species_View 是 Pedigree_View 的最近的父视图。FK_Animal 外键具有空值。

下图显示了如果清除“所有层次结构外键”选项，Pedigree_View 的示例数据：

| Preview data with file MULTIPLE_FK_EXAMPLE_DATA.xml | | | | |
|---|------------|-----------|-----------|--|
| Pedigree_... | FK_Species | FK_Animal | Pedigree1 | |
| Plains Cheeta | 1 | NULL | 100 | |
| African Lion | 2 | NULL | 200 | |

Number of rows: 2

集成服务生成到最近的父级的外键。

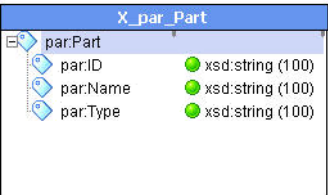
在循环关系中生成行

默认情况下，集成服务为循环关系中数据的所有出现生成行。您可以选择仅为第一次出现创建一行。选择非递归行选项。

以下 XML 文件包含具有循环引用的 Part 元素：

```
<?xml version="1.0" encoding="utf-8"?>
<Vehicle xmlns="http://www.PartInvoice.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.PartInvoice.org part.xsd">
  <VehInfo>
    <make>Honda</make>
    <model>Civic</model>
    <type>Compact</type>
  </VehInfo>
  <Part>
    <ID>123</ID>
    <Name>Body</Name>
    <Type>Exterior</Type>
    <Part>
      <ID>111</ID>
      <Name>DoorFL</Name>
      <Type>Exterior</Type>
      <Part>
        <ID>1112</ID>
        <Name>KnobL</Name>
        <Type>Exterior</Type>
      </Part>
    </Part>
    <Part>
      <ID>1113</ID>
      <Name>Window</Name>
      <Type>Exterior</Type>
    </Part>
  </Part>
</Vehicle>
```

下图显示了 Part 元素是 XML 定义中的 X_par_Part 视图的视图行：



下图显示了当您运行一个会话时，集成服务为 Part 123 和所有组件生成行：

☐ NonRecursive Row

| par_ID | par_Name | par_Type |
|--------|----------|----------|
| 123 | Body | Exterior |
| 111 | DoorFL | Exterior |
| 112 | DoorFR | Exterior |
| 113 | DoorRL | Exterior |

下图显示了当您选择“NonRecursive 行”时，集成服务读取 Part 元素的第一次出现，并为 Part 123 生成一行数据：

☒ NonRecursive Row

| par_ID | par_Name | par_Type |
|--------|----------|----------|
| 123 | Body | Exterior |

生成层次结构关系行

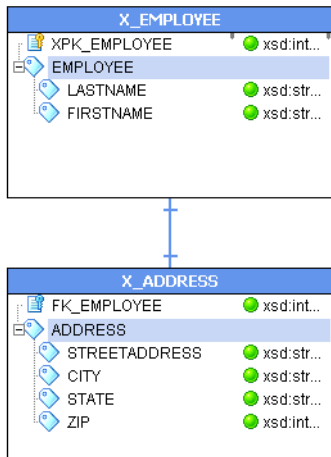
默认情况下，集成服务为视图行中包含数据的所有视图创建行。如果父视图在层次结构关系中有相应的数据，选择“层次结构关系行”来为子视图生成行。父视图必须包含数据来为子视图生成行。

例如，XML 定义可以包含 Employee 视图和 Address 视图组成的层次结构。员工是父视图。地址数据可以包含 Employee\Addresses 或 Store\Addresses。您可以选择输出 Employee\Address。

以下 XML 文件在 Store 元素中包含一个 Address，在 Employee 元素中包含一个 Address：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE STORE >
<STORE SID=" BE1752" >
  <SNAME>Mud and Sawdust Furniture Store</SNAME>
  <ADDRESS>
    <STREETADDRESS>335 Westshore Road</STREETADDRESS>
    <CITY>Fausta City</CITY>
    <STATE>CA</STATE>
    <ZIP>97584</ZIP>
  </ADDRESS>
  <EMPLOYEE DEPID=" 34" >
    <ENAME>
      <LASTNAME>Bacon</LASTNAME>
      <FIRSTNAME>Allyn</FIRSTNAME>
    </ENAME>
    <ADDRESS>
      <STREETADDRESS>1000 Seaport Blvd</STREETADDRESS>
      <CITY>Redwood City</CITY>
      <STATE>CA</STATE>
      <ZIP>94063</ZIP>
    </ADDRESS>
    <EPHONE>(408)226-7415</EPHONE>
    <EPHONE>(650)687-6831</EPHONE>
  </EMPLOYEE>
</STORE>
```

下图显示 Employee 视图和 Address 视图之间的层次关系：



Employee 视图通过一条蓝线连接到 Address 视图，此蓝线定义父视图和子视图之间的一对一关系。该 Employee 视图具有一个主键，XPK_Employee 和 Employee 元素由 LastName 和 FirstName 组成。Address 视图有一个外键 FK_Employee 以及由 StreetAddress、City、State 以及 Zip 组成的 Address 元素。

默认情况下，集成服务为 Address 元素的每次出现生成一行。集成服务为 Store\Address 生成一行，为 Employee\Address 生成另一行。

如果您清除“层次结构关系行”选项，下图显示地址 XML 数据：

☐ Hierarchy Relationship Row

| STREETADDRESS | FK_EMPL... | CITY | STATE | ZIP |
|--------------------|------------|--------------|-------|-------|
| 335 Westshore Road | NULL | Fausta City | CA | 97584 |
| 1000 Seaport Blvd | 1 | Redwood City | CA | 94063 |

当您选择“层次结构关系行”选项时，集成服务在如下所示的会话中生成行：

- 如果 Employee 视图包含会话中的相应数据，集成服务为 Address 视图生成行。
- 集成服务生成表示 Employee\Address 层次结构关系的行。
- 集成服务不会为 Store\Address 生成行。

如果您选择“层次结构关系行”选项，下图显示地址数据：

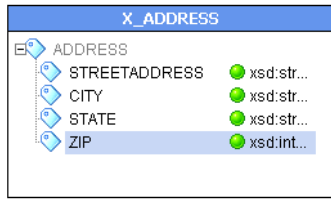
☒ Hierarchy Relationship Row

| STREETADDRESS | FK_EMPL... | CITY | STATE | ZIP |
|-------------------|------------|--------------|-------|-------|
| 1000 Seaport Blvd | 1 | Redwood City | CA | 94063 |

设置强制行选项

默认情况下，当视图行包含数据时，集成服务为该视图生成数据。无论视图行元素是否包含数据，都选择“强制行”来为 XML 视图生成行。例如，如果视图行是 address\zip，即使视图行没有任何 zip 数据，您也可以选择输出地址。

下图作为视图行显示了 zip 元素：



zip 元素在列表底部突出显示，上面列出 street、city 和 state 元素行。

默认情况下，集成服务为 address 元素内 zip 元素的每次出现生成行。

例如，您可能会在一个会话中处理以下 XML 文件：

```
<?xml version="1.0" ?>
<company xmlns="http://www.example.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org forcerow.xsd" >
  <name>company1</name>
  <address>
    <street>stree1</street>
    <city>city1</city>
    <zip>1001</zip>
  </address>
  <employee>
    <name>emp1</name>
    <address>
      <street>emp1_street</street>
      <city>empl_city</city>
    </address>
  </employee>
  <employee>
    <name>emp2</name>
    <address>
      <street>emp2_street</street>
      <city>emp2_city</city>
      <zip>2001</zip>
    </address>
  </employee>
</company>
```

默认情况下，集成服务为 Address 视图生成 stree1 和 emp2_street 行。

下图显示 Address 视图的 stree1 和 emp2_street 行：

☐ Force Row

| STREET | CITY | ZIP |
|-------------|-----------|------|
| stree1 | city1 | 1001 |
| emp2_street | emp2_city | 2001 |

集成服务不会为 emp1 生成行，因为视图行是 zip，而 emp1 没有 zip 元素的数据。

如果您启用“强制行”，您可以输出 street 和 city 元素，可以带有或不带有 zip。会话为 emp1 生成行，即使 emp1 没有 zip 元素的数据。

下图显示了启用“强制行”时集成服务生成的行：

☒ Force Row

| STREET | CITY | ZIP |
|-------------|-----------|------|
| street1 | city1 | 1001 |
| emp1_street | emp1_city | NULL |
| emp2_street | emp2_city | 2001 |

为类型关系中的视图生成行

默认情况下，集成服务为类型关系中的所有视图生成行。选择“类型关系行”选项可以为类型关系中的特定复杂类型生成行。为要输出的每个视图设置“类型关系行”。

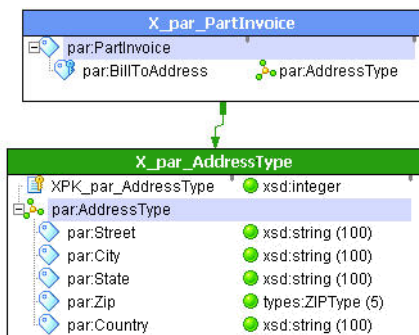
例如，一个定义可能包含一个层次结构，其中包括 BillToAddress 和 ShipToAddress。如果您要为 BillToAddress 生成行，请使用“类型关系行”选项。

定义类型关系

以下架构定义了 BillToAddress 和 ShipToAddress。BillToAddress 是 AddressType，而 ShipToAddress 是 USAddressType。USAddressType 扩展 AddressType。

```
<xsd:sequence>
  <xsd:element name="Street" type="xsd:string" />
  <xsd:element name="City" type="xsd:string" />
  <xsd:element name="State" type="xsd:string" />
  <xsd:element name="Zip" type="types:ZIPTType" />
  <xsd:element name="Country" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="USAddressType">
  <xsd:complexContent>
    <xsd:extension base="AddressType">
      <xsd:sequence>
        <xsd:element name="PostalCode" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="BillToAddress" type="AddressType" />
<xsd:element name="ShipToAddress" type="USAddressType" />
```

下图显示了 XML 定义中的类型关系，其中 PartInvoice 视图在 X_par_PartInvoice XML 视图中显示 BillToAddress 元素，在下面显示关联的 X-par_AddressType XML 视图：



PartInvoice 视图包含发票数据。该视图包括 BillToAddress。XML 定义中的类型关系是 BillToAddress 和 AddressType 之间的关系。

要将 AddressType 数据限制为 BillToAddress，请在 XML 编辑器工作区中选择 X_par_PartInvoice 视图。选择“类型关系行”选项。当您运行会话时，集成服务为 BillToAddress 但不为 ShipToAddress 生成地址行。ShipToAddress 不在类型关系中。

示例

以下示例演示如何将数据限制为类型关系中的特定类型。该示例使用 PartInvoice 视图和 AddressType 视图。

以下 XML 文件包含包括 BillToAddress 和 ShipToAddress 的发票数据：

```
<xsd:complexType name="AddressType">
<?xml version="1.0" encoding="utf-8"?>
<Invoices xmlns="http://www.PartInvoice.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.PartInvoice.org Part.xsd">
  <PartInvoice InvoiceNum="185" DateShipped="2005-01-01">
    <PartOrder>
      <PartID>HLG100</PartID>
      <PartName>Halogen Bulb</PartName>
      <Quantity>2</Quantity>
      <UnitPrice>35</UnitPrice>
    </PartOrder>
    <BillToAddress>
      <Street>2100 Seaport Blvd</Street>
      <City>Redwood City</City>
      <State>CA</State>
      <Zip>94063</Zip>
      <Country>USA</Country>
    </BillToAddress>
    <ShipToAddress xsi:type="USAddressType">
      <Street>3350 W Bayshore Rd</Street>
      <City>Palo Alto</City>
      <State>CA</State>
      <Zip>97890</Zip>
      <Country>USA</Country>
      <PostalCode>97890</PostalCode>
    </ShipToAddress>
  </PartInvoice>
</Invoices>
```

当您运行会话时，集成服务为每个 AddressType 生成一个 X_par_AddressType 行。

下图显示了集成服务在默认情况下生成的 BillToAddress 和 ShipToAddress 行：

☐ Type Relationship Row

| XPK_par_... | par_Street | par_City | par_State | par_Zip | par_Country |
|-------------|--------------------|-------------|-----------|---------|-------------|
| 1 | 2100 Seaport Blvd | Redwood ... | CA | 94063 | USA |
| 2 | 3350 W Bayshore Rd | Palo Alto | CA | 97890 | USA |

要生成与 PartInvoice 视图相关的 AddressType 行，为 PartInvoice 视图设置“类型关系行”选项。

下图显示了“类型关系行”选项生成的 BillToAddress 行：

| XPK_par_... | par_Street | par_City | par_State | par_Zip | par_Country |
|-------------|--------------|-------------|-----------|---------|-------------|
| 1 | 2100 Seap... | Redwood ... | CA | 94063 | USA |

因为 ShipToAddress 不在类型关系中，它不会为 ShipToAddress 生成行。

对使用 XML 编辑器进行故障排除

当验证 XML 定义时，遇到错误，显示 XML 定义太大。原因是什么？

如果导入一个 XML 文件，其中包含使用无限长度定义的组件，您可以很容易超过 500 MB 的总列长度限制。您可以在 XML 编辑器中更改列长度，或者可以设置选项以覆盖所有无限长度，并重新导入该文件。

我找不到查看 XML 元数据时创建的 DTD 或 XML 架构文件。

您可以查看的 DTD 或 XML 架构文件是 Designer 创建的用于查看的临时文件。如果您要为其他目的使用该文件，查看时请使用另一名称和目录保存该文件。

当将列添加到 XML 源视图时，源 XML 文件中的层次结构保持不变。

将列添加到 XML 源视图时，您不能向基本层次结构添加元素。无论您如何创建视图或将视图中的列映射到层次结构中的元素，导入的 XML 层次结构保持不变。您可以修改数据类型和元素的基数，但不能修改层次结构。

有关 PowerCenter 中的 XML 规模估算的信息，请参阅[“使用 XML 与 PowerCenter 概览” 页面上 27](#)。有关 PowerCenter 中 XML 处理所适用的局限性的详细信息，请参阅[“限制” 页面上 27](#)。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

使用 XML 目标

使用 XML 目标概览

您可以通过以下方式创建 XML 目标定义：

- **从 XML 架构或文件中导入定义。**您可以通过 XML、DTD 或 XML 架构文件创建目标定义。您可以从 URL 或本地节点导入 XML 文件定义。如果您使用关联的 DTD 导入 XML 文件，XML 向导将使用 DTD 来生成该 XML 文件。
- **根据 XML 源定义创建一个 XML 目标定义。**您可以将现有 XML 源定义拖动到 Target Designer 中。如果您创建一个 XML 目标定义，Designer 根据 XML 定义的层次结构创建目标定义。
- **根据关系文件定义创建 XML 目标。**您可以通过关系或平面文件存储库定义导入 XML 目标定义。

在创建 XML 目标定义后，您可以在 Target Designer 中使用 XML 目标完成以下任务：

- **编辑目标属性。**编辑 XML 目标定义，将注释记录更改添加到目标 XML、DTD 或 XML 架构文件。
- **同步目标定义。**如果您需要进行更改，可以将目标 XML 定义与架构同步。当您同步定义时，如果架构发生更改，请更新 XML 定义而不是导入架构。

通过 XML 文件导入一个 XML 目标定义

您可以通过 XML 架构、DTD 文件或 XML 文件导入 XML 定义。您可以导入本地文件或通过 URL 引用的文件。为确保 Designer 可以提供数据的准确定义，请从 XML 架构导入目标定义。

可以从下列选项中选择以创建 XML 视图：

- **创建实体关系。**使用此选项为多次出现或引用的元素和复杂类型创建视图。您可以在视图之间创建关系，而不是创建一个大型层次结构。
- **创建层次结构关系。**使用此选项可以创建根，并在根下扩展 XML 组件。您可以选择创建规范化或非规范化视图。如果选择规范化，每个元素或属性出现一次。一对多关系成为单独的 XML 视图，其中包含用于关联视图的键。如果创建非规范化 XML 视图，所有的元素和属性显示在一个层次结构组中。
- **创建自定义 XML 视图。**使用此选项可以为 XML 视图选择多个全局元素作为根，然后选择选项来减少元数据分解。
- **跳过创建 XML 视图。**使用此选项可以在不创建任何视图的情况下导入定义。如果您选择此选项，请使用 XML 编辑器之后在工作区中创建 XML 视图。
- **同步 XML 定义。**使用此选项可以在 XML 定义基本架构更改时更新一个或多个 XML 定义。

提示：导入 DTD 或 XML 架构文件而不是 XML 文件。如果您使用关联的 DTD 导入 XML 文件，XML 向导将使用 DTD。

要导入 XML 目标定义，请按以下步骤操作：

1. 在 Target Designer 中，单击“目标”>“导入 XML 定义”。
将打开“导入 XML 定义”窗口。默认情况下显示本地文件夹架构文件。
2. 单击“本地文件”或 URL 可浏览 XML 文件。
3. 要浏览 DTD 或 XML 文件，请从“文件类型”列表中选择相应的文件扩展名。

通过 XML 源定义创建目标

当您想要创建类似于现有源定义的目标定义时，请使用源定义或源定义的快捷方式创建目标定义。将 XML 源定义拖动到 Target Designer 中来创建 XML 目标定义或关系目标定义。

使用以下准则来创建 XML 目标定义：

- 当您通过 XML 源定义创建 XML 目标定义时，您创建了 XML 源定义的副本。
- 有效的 XML 源定义不一定创建有效的 XML 目标定义。要确保创建有效的目标定义，请验证目标定义。

注意：XML 目标定义不能包含透视列。

使用以下准则来创建关系目标定义：

- 如果创建关系目标定义，Designer 根据 XML 源定义中的组创建关系目标定义。XML 源定义中的每个组将成为目标定义。
- Designer 创建的目标定义之间的关系与源定义的组之间的关系相同。

要通过 XML 源定义创建目标定义，请按以下步骤操作：

1. 将 XML 源定义从导航器拖到 Target Designer 工作区。
将显示“XML 导出”对话框。
2. 选择创建关系或 XML 目标。单击“确定”。

目标定义将显示在 Target Designer 工作区中。如果选择关系目标，根据源，多个目标定义可能显示在工作区中。

编辑 XML 目标定义属性

创建 XML 目标定义后，您可以编辑属性以反映目标数据的更改、添加业务名称和注释或更改代码页。

要编辑 XML 目标定义属性，请按以下步骤操作：

1. 在 Target Designer 中打开 XML 目标。
2. 右键单击并选择“编辑”。
3. 在“表”选项卡上编辑设置。

下表描述了“表”设置：

| 表设置 | 说明 |
|-------|---|
| 选择表 | 目标定义的名称。要更改该名称，单击“重命名”按钮。 |
| 业务名称 | 目标表的描述性名称。使用“重命名”按钮编辑业务名称。 |
| 约束 | 不适用于 XML 目标。任何条目将被忽略。 |
| 创建选项 | 不适用于 XML 目标。任何条目将被忽略。 |
| 说明 | 目标表描述。字符限制是 2000 字节/K，其中 K 是存储库代码页中每个字符的最大字节数。输入到业务文档的链接。 |
| 代码页 | 选择要在目标定义中使用的代码页。 |
| 数据库类型 | 指示目标定义是 XML 目标。 |
| 关键字 | 使用关键字来组织和标识目标。关键字可以包括开发人员姓名、映射或 XML 架构名称。 关键字用于在 Repository Manager 中执行搜索。 |

4. 在“列”选项卡上，您可以查看 XML 列定义。

下表描述了“列”设置：

| 列设置 | 说明 |
|------|---|
| 选择表 | 显示您正在编辑的目标定义。要选择其他定义进行编辑，在工作区中从您的定义列表中选择一个定义。 |
| 列名称 | 列的名称。 |
| 数据类型 | 列的 PowerCenter 数据类型。 |
| 精度 | 列的大小。您可以更改字符串等部分数据类型的精度。 |
| 小数位数 | 数值的小数点后的最大位数。 |
| 非空 | 指示列是否可以包含空值。 |

| 列设置 | 说明 |
|-------|-----------------------|
| 键类型 | XML 向导生成的用于链接视图的键的类型。 |
| XPath | 通过 XML 文件层次结构定位项目的路径。 |

5. 在“属性”选项卡上，您可以修改目标定义的转换属性。

如果您使用基于源的提交会话或带有 XML 目标的事务控制转换，可以定义如何对目标刷新数据。

下表说明了您可以编辑的属性：

| 列设置 | 说明 |
|--------|---|
| 选择表 | 显示要编辑的源定义。要选择不同源定义进行编辑，请从列表中选择源定义。 |
| 重复组行处理 | 选择这些选项之一来处理目标中的重复行： <ul style="list-style-type: none"> - 第一行。集成服务将第一个重复行传递给目标。跟随相同主键的行被拒绝。 - 最后一行。集成服务将最后一个重复行传递到目标。 - 出错。集成服务将第一行传递到目标。具有重复主键的行增加错误计数。如果错误计数达到错误阈值，会话失败。 |
| DTD 引用 | 目标 XML 文件的 DTD 或 XML 架构文件名称。当您创建 XML 文件时，集成服务向 XML 文件添加文档类型声明。 |
| 提交后 | 集成服务可以生成多个 XML 文件，或在提交后附加到一个 XML 文件。使用以下选项之一： <ul style="list-style-type: none"> - 忽略提交。集成服务将创建一个 XML 文件，并在文件末尾写入 XML 文件。 - 创建新文档。每次提交时创建一个新的 XML 文件。 - 附件到文档。每次提交后，写入同一 XML 文件。 |
| 缓存目录 | XML 目标缓存文件的目录。默认为 \$PMCacheDir 服务进程变量。 |
| 缓存大小 | XML 目标缓存的总大小（以字节为单位）。默认为“自动”。 |

6. 在“元数据扩展”选项卡上，可以创建、修改、删除和升级非可重用元数据扩展，并更新它们的值。您还可以更新可重用元数据扩展的值。

7. 单击“确定”。

8. 单击“存储库”>“保存”。

验证 XML 目标

您可以创建说明如何将数据提取到 XML 文件的自定义 XML 视图。然而，并不是所有视图之间的结构或关系在 XML 定义中都是有效的。一些视图结构可能对于 XML 源是有效的，对于 XML 目标无效。Designer 可以防止您创建不明确的定义。

当您执行以下任务时，PowerCenter 验证目标 XML 视图：

- 当您从存储库保存或提取 XML 目标时，Designer 会进行有限验证。
- 当您在 XML 工作区中编辑 XML 时，XML 编辑器验证每一步。
- 您可以选择验证正在 XML 编辑器中编辑的目标定义。

- 当 Designer 验证映射时，Designer 将验证 XML 目标连接。

Designer 使用规则来验证层次结构关系、类型关系和继承关系。

注意：集成服务不会针对会话中的架构验证目标 XML 实例。您可以设置“验证目标”会话属性来验证数据中的简单类型。

层次结构关系验证

Designer 使用以下规则来验证层次结构关系：

- 在一种类型有一个根的视图不能作为独立视图。视图必须是继承关系中的一个子视图，或必须具有与另一视图的类型关系。如果 XML 目标不包含以一个元素为根的视图，XML 目标无效。
- 您必须将带有多次出现视图行的视图连接到另一个视图。
- 两个视图不能具有相同的有效视图行。
- 如果 XML 目标在某一元素没有视图根，XML 目标无效。
- 您可以通过其他元素分隔父视图和子视图，但如果对于一个视图，您可以在两个父视图选择，则必须使用最接近的一个。通过有效视图行的路径来确定最接近的父视图。一个父视图早于路径中的其他视图。选择路径中出现的第二个视图。
- 您必须使用同一层次结构中的同一视图根连接所有视图。对于同一视图根，该定义不能包含多个树。
- 如果对于该视图，视图行和视图根是相同的，XML 视图可以包含到其自身的层次结构关系。

类型关系验证

类型关系是列和视图之间的关系。类型关系不是两个视图之间的关系。以下规则适用于类型关系：

- 如果视图根是相同的类型，或 V2 视图根类型是从 V1 视图根派生的，则视图 V1 中的列可以包含到视图 V2 的类型关系。这两个视图的根必须是全局复杂类型。
- 如果一个视图中的一列具有到另一视图的类型关系，您不能展开该列。

继承验证

您可以使用 XML 视图创建两种类型的继承关系：

- **视图-视图继承。**视图是另一视图的派生类型。这两种视图必须是全局复杂视图根。
如果某个视图的视图根是从另一视图的视图根类型派生的复杂类型，该视图可以包含到另一视图的继承关系。
视图可以是多重继承关系中的父视图，但视图可以只是一个继承关系的子视图。
- **列-视图继承。**该列是本地复杂类型 Type1 的一个元素，而视图以全局复杂类型 Type2 为根。Type1 是从 Type2 派生的。
如果列是本地复杂类型，该类型是从另一个视图的视图根类型派生的，那么视图中的列可以包含到另一视图的继承关系。
如果视图 V1 中的一列具有到视图 V2 的继承关系，您不能将视图 V2 的内容放入视图 V1。

在映射中使用的 XML 目标

当您将 XML 目标添加到映射时，您需要遵守多组转换映射准则。

以下组件会影响您在映射中进行 XML 目标映射的方式：

- 活动源
- 根元素
- 目标端口连接
- 抽象元素
- 事务控制点
- FileName 列

活动源

活动源是可以为每个输入行返回不同数量的行的转换。集成服务可以从不同活动源将数据加载到 XML 目标。然而，XML 目标的单个组中的所有端口必须从相同的活动源接收数据。

以下转换是活动源：

- 汇总器
- 应用程序源限定符
- 自定义，配置为一个活动转换
- Java，配置为一个活动转换
- 联接器
- MQ 源限定符
- Normalizer (VSAM 或管道)
- 等级
- 排序器
- 源限定符
- SQL
- XML 源限定符
- Mapplet，如果 Mapplet 包含以上转换之一

选择根元素

如果 XML 定义具有多个可能的根，您可以为一个目标实例指定一个根元素。

要指定根元素，请按以下步骤操作：

1. 右键单击 Mapping Designer 中的目标定义，选择“编辑”。
2. 单击“属性”选项卡。
3. 单击“根元素”值列中的箭头。
将显示“选择根”对话框。
4. 从列表中选择一个元素。

连接目标端口

您需要正确连接映射中的 XML 目标端口，所以集成服务可以在会话期间创建有效的 XML 文件。当您通过 XML 目标保存或验证映射时，Designer 将验证目标端口的连接。

当您连接映射的端口时，请使用以下准则：

- 如果连接组中的一个端口，您必须同时连接该组的外键和主键端口。
- 如果您连接在组中的外键端口，则必须连接另一组中的关联主键端口。如果不连接根组的主键端口，您不需要连接其他组中关联外键端口。
- 如果您使用带有默认属性值的 XML 架构，则必须连接属性端口，以便在目标中创建默认属性。如果您通过连接端口传递一个空值，集成服务将默认值写入目标。

连接抽象元素

抽象元素不能直接在 XML 实例文件中出现。而是必须使用一个从抽象元素派生的元素。默认情况下，Designer 为任何抽象复杂元素创建一个视图。要减少元数据，来自抽象类型的元素不在任何派生类型中重复。当您数据映射到抽象类型时，需要也将数据映射到至少一个派生类型。

会话期间，如果集成服务将数据加载到一个抽象类型，那么集成服务也应该使非抽象派生类型的数据与抽象类型相关联。如果派生类型没有任何数据，那么集成服务不将该抽象元素写入目标 XML 文件。

对目标刷新 XML 数据

您可以在一个会话中的每个提交点将数据刷新到 XML 目标。然而，每个输入组必须通过映射中的同一个事务控制点接收数据。当您根据此映射创建一个会话时，可以在每次提交时将数据附加到 XML 文件目标，或每次提交时创建一个新文件。您可以使用“提交后”目标属性指定任一选项。

当您 XML 目标输入组连接到多个事务控制点时，集成服务在处理所有源行之后将数据写入 XML 文件目标。

动态命名 XML 文件

您可以将 FileName 列添加到 XML 目标定义，以动态创建 XML 文件的文件名。当集成服务将数据传递到 FileName 列时，集成服务将覆盖目标属性中的输出文件名。例如，如果您将字符串“Harry”传递到 FileName 列，集成服务将 XML 文件命名为 Harry。

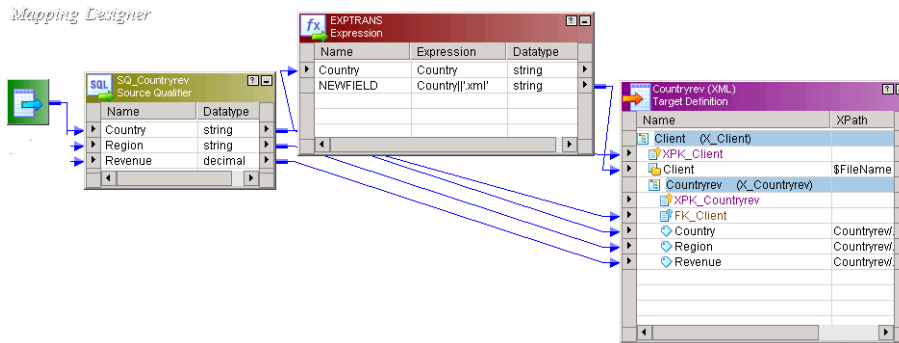
注意：如果您在每个提交创建一个新的 XML 文件，需要动态命名创建的每个 XML 文件。如果您不动态命名每个 XML 文件，集成服务将覆盖之前提交产生的 XML 文件。

集成服务为目标的根组中每个不同的主键值生成新的 XML 文件。添加 FileName 列，为每个文件设置不同名称。每个名称在会话属性中重写输出文件名。

示例

表达式转换通过 Country XML 元素生成一个文件名，并将值传递给 FileName 列。映射将国家/地区传递到目标根（称为 Client）。每当客户端的值更改时，集成服务创建新的 XML 文件。集成服务创建包含每个 XML 目标文件名的列表文件。集成服务列出列表中每个文件的绝对路径。

下图显示了包含带有 FileName 列的 XML 目标的映射：



集成服务将以下行传递到目标：

```
Country,Region,Revenue
USA,region1,1000
France,region1,10
Canada,region1,100
USA,region2,200
USA,region3,300
USA,region4,400
France,region2,20
France,region3,30
France,region4,40
```

此会话根据国家/地区名称生成以下文件：

```
Canada.xml
France.xml
USA.xml
```

列表文件名是来自会话属性的输出文件名：

```
revenue_file.xml.lst
```

XML 对象故障排除

我从 XML 文件导入源定义。然后我从相同的 XML 文件导入目标定义。源和目标的定义的默认组不相同。

如果您在导入目标时更改部分选项，XML 向导有时为源定义和目标定义创建不同的组结构。

例如，以下 DTD 中的 ContactInfo 元素是 enclosure 元素。enclosure 元素没有文本内容，但包含 maxOccurs > 1。子元素也包含 maxOccurs > 1。

```
<!ELEMENT HR (EMPLOYEE+)>
<!ELEMENT EMPLOYEE (LASTNAME,FIRSTNAME,ADDRESS+,CONTACTINFO+)>
<!--ATTLIST EMPLOYEE EMPID CDATA #REQUIRED-->
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT ADDRESS (STREETADDRESS,CITY,STATE,ZIP)>
<!ELEMENT STREETADDRESS (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT ZIP (#PCDATA)>
<!ELEMENT CONTACTINFO (PHONE+,EMERGCONTACT+)>
```



```

<!ELEMENT PHONE (#PCDATA)>
<!ELEMENT EMERGCONTACT (#PCDATA)>

```

如果您不在源定义中为 enclosure 元素创建 XML 视图，则不在源中创建 ContactInfo 元素。

下图显示了 XML 向导创建的源和目标定义：

| Name | Datatype |
|-------------------------------|-------------|
| EMPLOYEE (X_EMPLOYEE) | |
| XPK_EMPLOYEE | xsd:integer |
| EMPID | xsd:string |
| LASTNAME | xsd:string |
| FIRSTNAME | xsd:string |
| ADDRESS (X_ADDRESS) | |
| XPK_ADDRESS | xsd:integer |
| FK_ADDRESS | xsd:integer |
| STREETADDRESS | xsd:string |
| CITY | xsd:string |
| STATE | xsd:string |
| ZIP | xsd:string |
| PHONE (X_PHONE) | |
| XPK_PHONE | xsd:integer |
| FK_PHONE | xsd:integer |
| PHONE | xsd:string |
| EMERGCONTACT (X_EMERGCONTACT) | |
| XPK_EMERGCONTACT | xsd:integer |
| FK_EMERGCONTACT | xsd:integer |
| EMERGCONTACT | xsd:string |

源定义并不包括 ContactInfo 元素。目标定义包括 ContactInfo 元素。该向导在源定义中不包括 ContactInfo 元素，因为当您创建源时选择了不为 enclosure 元素创建视图。然而，向导在目标定义中包括 ContactInfo 元素。

我通过关系数据源创建的 XML 目标定义包含所有元素，但没有属性。如何修改目标层次结构才能将某些数据标记为属性？

您不能修改向导通过关系表创建的组件类型。但是，您可以查看目标 XML 层次结构的 DTD 或 XML 架构文件。使用新文件名保存 DTD 或 XML 架构文件。打开此新文件并修改层次结构，设置属性和元素。然后，使用该文件导入带有新层次结构的目标定义。

XML 源限定符转换

XML 源限定符转换概览

将 XML 源定义添加到映射时，您需要将源定义连接到 XML 源限定符转换。XML 源限定符转换定义集成服务在会话过程中读取的数据元素。源限定符确定 PowerCenter 如何读取源数据。XML 源限定符转换是一种活动转换。

您可以手动添加源限定符转换，还可以在将源定义添加到映射时默认创建源限定符转换。

您可以编辑部分属性并将元数据扩展添加到 XML 源限定符转换。

在映射中连接 XML 源限定符转换时，您必须遵循创建有效映射的规则。

向映射添加一个 XML 源限定符

对于 XML 源中的每一列，XML 源限定符转换有一个与之对应的输入/输出端口。创建源定义的 XML 源限定符转换时，Designer 将 XML 源定义中的每个端口链接到 XML 源限定符转换中的端口。您不能删除或编辑这些链接。如果从映射中删除 XML 源定义，Designer 也会删除对应的 XML 源限定符转换。您可以将一个 XML 源定义链接到一个 XML 源限定符转换。

您可以将一个 XML 源限定符组的端口链接其他转换的端口以形成独立的数据流。但是，您不能将一个 XML 源限定符转换中的多个组的端口链接到同一目标转换中的端口。

如果将多个组中的列拖动到某个转换，Designer 将所有组的列复制到该转换。但是，Designer 仅将第一组的端口链接到转换中新键列的对应端口。

通过将 XML 源定义拖动到 Mapping Designer 工作区或通过手动创建源限定符，您可以将 XML 源限定符转换添加到映射。

默认创建 XML 源限定符转换

将 XML 源定义拖动到 Mapping Designer 工作区时，Designer 默认创建 XML 源限定符转换。

要默认创建 XML 源限定符转换，请按以下步骤操作：

1. 在 Mapping Designer 中，创建新映射或打开现有映射。
2. 将 XML 源定义拖动到映射。

Designer 创建 XML 源限定符转换，并且将 XML 源定义中的各个端口链接到 XML 源限定符转换中的端口。

手动创建 XML 源限定符转换

如果映射包含无源限定符的 XML 源定义，或者从映射中删除了 XML 源限定符转换，您可以在映射中创建 XML 源限定符转换。

要手动创建 XML 源限定符转换，请按以下步骤操作：

1. 在 Mapping Designer 中，创建新映射或打开现有映射。

注意：映射中至少有一个无源限定符的 XML 源定义。

2. 单击“转换”>“创建”。

此时将显示“创建转换”对话框。

3. 选择 XML 源限定符转换，然后键入转换的名称。

XML 源限定符转换的命名约定为 `XSQ_TransformationName`。

4. 单击“创建”。

Designer 列出映射中无对应 XML 源限定符转换的所有 XML 源定义。

5. 选择源定义，然后单击“确定”。

Designer 在映射中创建 XML 源限定符转换，并且将 XML 源定义中的各个端口链接到 XML 源限定符转换中的端口。

编辑 XML 源限定符转换

您可以编辑 XML 源限定符转换属性，如转换名称和说明。

要编辑 XML 源限定符转换，请按以下步骤操作：

1. 在 Mapping Designer 中，打开 XML 源限定符转换。
2. 在“转换”选项卡上编辑属性。

下表介绍了转换属性：

| 转换设置 | 说明 |
|------|-------------------------------|
| 选择转换 | 显示正在编辑的转换。要编辑其他转换，请从列表中选择该转换。 |
| 重命名 | 编辑转换的名称。 |
| 说明 | 描述转换。 |

3. 单击“端口”选项卡查看 XML 源限定符转换端口。

使用序列列设置 XML 组中生成的键的起始值。您可以为生成的每个键输入不同的值。序列键的数据类型为 bigint。更改这些值时，序列号在下次运行会话时重新开始。

4. 单击“属性”选项卡配置在会话过程中影响集成服务运行映射的方式的属性。

下表介绍了 XML 源限定符属性：

| 属性设置 | 说明 |
|------|---|
| 选择转换 | 显示正在编辑的转换。要编辑其他转换，请从列表中选择该转换。 |
| 跟踪级别 | 确定集成服务在运行工作流时写入会话日志的有关此转换的信息量。您可以在配置会话时替换此跟踪级别。 |
| 重置 | 会话结束时，集成服务将起始值重置为当前会话的起始值。 |
| 重新开始 | 会话开始时，集成服务从 1 开始对所有组的生成的键序列计数。 |

5. 单击“元数据扩展”选项卡以创建、编辑和删除用户定义的元数据扩展。

您可以创建、修改、删除和升级非可重用元数据扩展，并更新它们的值。您还可以更新可重用元数据扩展的值。

6. 单击“确定”。

设置生成的键的序列号

XML 源限定符定义中的每个视图具有主键和该键的序列值。在会话过程中，集成服务通过序列值生成键并增加序列值。

在会话结束时，集成服务将存储库中的各个序列值更新为当前值加 1。这些值将成为集成服务下次处理序列生成器转换时的起始值。

存储库维护以下序列值：

- **默认值。**首次创建源限定符时，显示在 XML 源限定符中的键的序列值。各个键的默认值为 1。
- **起始值。**会话开始时键的序列号值。运行工作流之前，您可以在 XML 源限定符转换中查看起始值。
- **当前值。**会话过程中的键的序列值。

生成的键的起始值显示在 XML 源限定符中的序列列。

注意: 如果在“端口”选项卡上编辑序列起始值，您必须在运行工作流之前保存更改并退出 Designer。

更改序列起始值

在会话开始之前或结束之后，您可以通过使用 XML 源限定符转换“属性”选项卡上的以下选项来更改序列起始值：

- **重置。**会话结束时，集成服务将起始值重置回当前会话的起始值。例如，会话开始时，键的起始值为 2000。会话结束时，当前值为 2500。会话完成时，存储库中的起始值仍然是 2000。如果您正在进行测试，要在下次运行会话时生成相同键编号，则可以使用此选项。
- **重新开始。**会话开始时，集成服务使用默认值重新开始起始值。例如，如果键的起始值为 1005，选择“重新开始”后，集成服务将起始值更改为 1。当键不断增大时，您可以使用此选项，重新编号后不会产生重复键冲突。

在映射中使用 XML 源限定符

XML 源定义中的各个组类似于关系表，Designer 将 XML 源限定符转换中的各个组视为独立的数据源。

连接映射中的对象时，Designer 强制执行串联规则。因此，您需要组织 XML 源定义中的组，以便各个组包含单个管道分支中所需的所有信息。

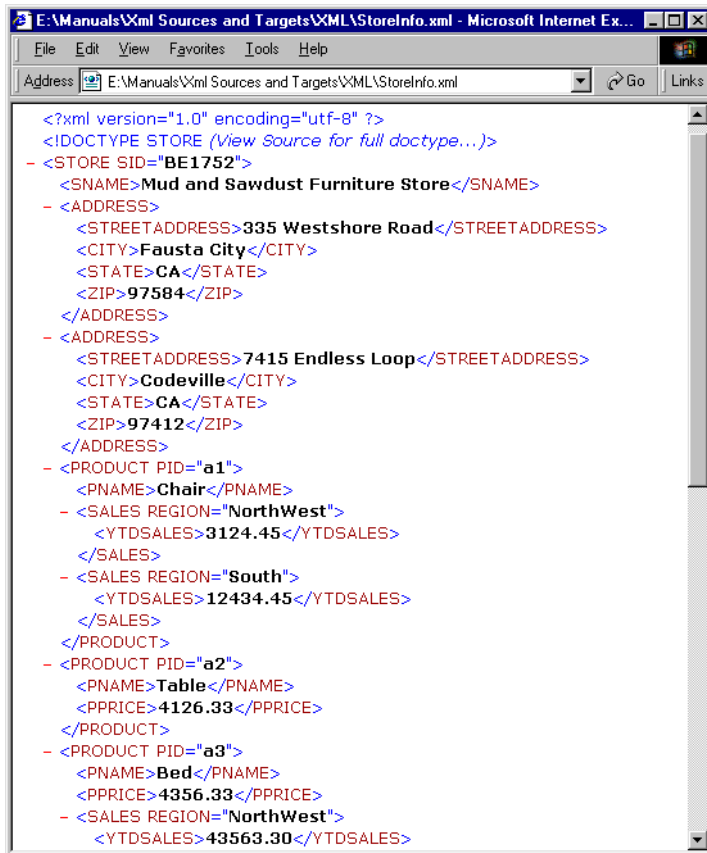
连接映射中的 XML 源限定符转换时，请考虑以下规则：

- 您可以将 XML 源限定符转换中的一个组的端口链接到另一转换的一个输入组中的端口。您可以将多个组的列复制到一个转换，但仅能将一个组的端口链接到转换中的对应端口。
- 您可以将 XML 源限定符转换中的一个组的端口链接到多个转换中的端口。XML 源限定符转换中的各个组可以是多个管道分支的数据源。可将数据从一个组传递到多个不同转换。
- 您可以将一个 XML 源限定符转换中的多个组链接到某个转换中的其他输入组。您可以将 XML 源限定符转换中的多个组链接到大多数多输入组转换（如联接器转换或自定义转换）中的不同输入组。但是，如果联接器有排序的输入，您可以将一个 XML 源限定符转换中的多个组链接到一个联接器转换。要将两个 XML 源限定符转换组链接到带有无排序输入的联接器转换，您必须创建同一 XML 源的两个实例。

XML 源限定符转换示例

本节显示了映射中的 XML 源限定符转换的示例。该示例使用包含商店、产品和销售额信息的 XML 文件。

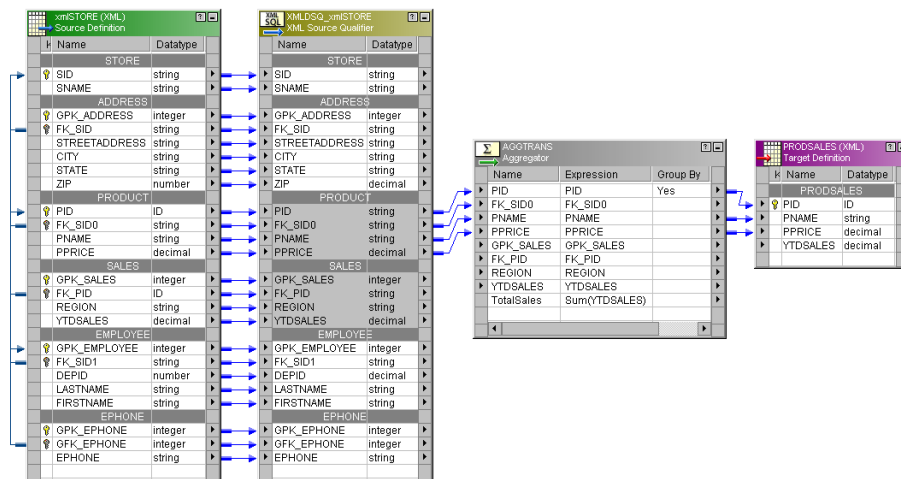
下图显示了 StoreInfo.xml 文件：



假设要计算 XML 文件中各种产品年初至今在所有区域的总销售额。除销售额外，还要获取各种产品的名称和价格。

要实现此目的，在同一转换中需要产品和销售额信息。然而，在默认情况下，当导入 StoreInfo.xml 文件时，Designer 为产品和销售额创建单独的组。

下图显示了 StoreInfo 文件的默认组（产品和销售额信息位于单独的组）：



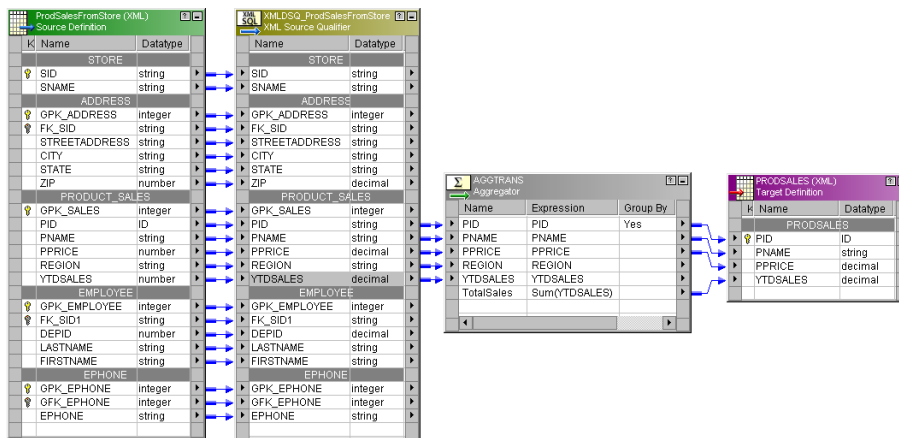
由于无法将 Product 和 Sales 组都链接到同一单输入组转换，您可以使用以下方法之一创建映射：

- 使用包含所有所需信息的非规范化组。
- 使用联接器转换联接两个组的数据。

使用单个非规范化组

您可以在源定义中组织组，以使所有信息来自相同的组。例如，您可以将 Product 和 Sales 组组合成源定义中的一个非规范化组。您可以通过一个数据流处理来自非规范化组的所有销售额汇总信息。

下图显示了包含 Product 和 Sales 组中的列的组合的非规范化组 Product_Sales：



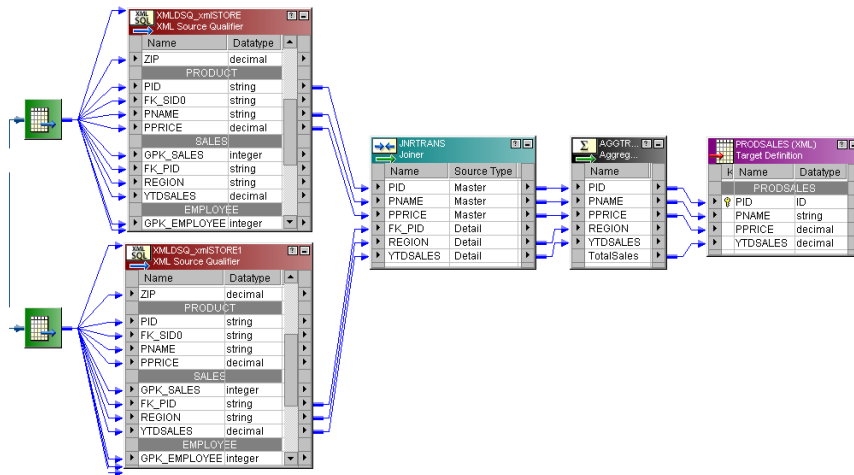
要创建非规范化组，请在 Source Analyzer 中编辑源定义。您可以创建新组或修改现有的组。将产品和销售额列添加到该组以在汇总器转换中执行销售额计算。使用 XML 编辑器创建组并验证该组。

联接两个 XML 源限定符转换组

您可以将两个源组中的数据联接成一个数据流。可使用联接器转换联接组中的数据。为排序输入配置联接器转换时，您可以将一个 XML 源限定符转换实例中的两个组链接到联接器转换。使用为非排序输入配置的联接器转换时，您必须使用同一 XML 源的两个实例，并且将各个 XML 源限定符转换实例的组链接到联接器转换。

然后，您可以将联接器转换中的数据发送到汇总器转换以计算各种产品的年初至今的销售额。

下图显示了如何创建同一 XML 源的两个实例及联接两个 XML 源限定符转换的数据：



XML 源限定符转换故障排除

将 XML 源限定符转换中的两个组拖动到某个转换时，Designer 复制列而不链接所有端口。

您可以将 XML 源限定符转换中的一个组链接到一个转换。将多个组拖动到某个转换时，Designer 将所有列名称复制到该转换。但是，Designer 仅链接第一个组的列。

不能断开 XML 源定义与其源限定符之间的链接。

XML 源限定符转换列与对应的 XML 源定义列相匹配。您不能删除或修改 XML 源定义与其 XML 源限定符转换之间的链接。删除 XML 源定义时，Designer 将删除其 XML 源限定符转换。

中游 XML 转换

中游 XML 转换概览

XML 定义读取或创建 XML 数据。但是，有时需要在管道内提取或生成 XML。例如，假设要将消息作为数据字段发送到包含 XML 文档的 TIBCO 目标。在此情况下，将消息发送到 TIBCO 之前，您需要生成 XML 文档。可以使用 XML 转换生成 XML。

您可以创建以下类型的中游 XML 转换：

- **XML 解析器转换。**XML 解析器转换从一个输入端口读取 XML 并将数据输出到一个或多个组。
- **XML 生成器转换。**XML 生成器转换从一个或多个源读取数据并生成 XML。XML 生成器转换有一个输出端口。

可使用中游 XML 转换从消息系统（如 TIBCO 或 WebSphere MQ）或其他源（如文件或数据库）提取 XML 数据。XML 转换的功能类似于 XML 源和目标功能，但中游 XML 转换解析 XML 或在管道中生成文档。

中游 XML 转换支持的 XML 架构组件与 XML 向导和 XML 编辑器支持的相同。此外，XML 转换支持以下功能：

- **传递端口。**使用传递端口通过中游转换传递非 XML 数据。这些字段不属于 XML 架构定义，但可用于生成非规范化 XML 组。这些字段的使用方式与顶级 XML 元素的使用方式相同。您可以使用传递字段作为 XML 定义中顶级组的主键。
- **实时处理。**使用中游 XML 转换将数据作为来自消息系统的 BLOB 来进行处理。
- **支持多分区。**您可以为每个分区生成不同的 XML 文档。

XML 解析器转换

集成服务处理 XML 解析器转换时，它会读取一行 XML 数据，解析 XML，然后通过输出组返回数据。XML 解析器转换在传递端口返回非 XML 数据。您可以解析来自源（如 JMS 或 IBM WebSphere MQ）的 XML 消息。XML 解析器转换是一种活动转换。

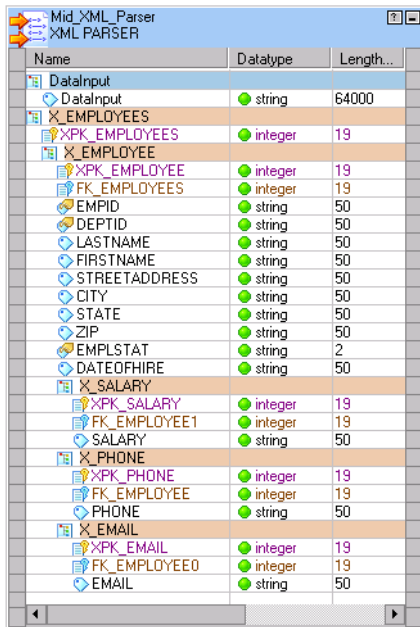
XML 解析器转换有一个输入组和一个或多个输出组。输入组有一个输入端口 DataInput，该端口接受字符串中的 XML 文档。

创建 XML 解析器转换时，可使用 XML 向导导入 XML、DTD 或 XML 架构文件。例如，您可以导入以下员工 DTD 文件：

```
<!ELEMENT EMPLOYEES (EMPLOYEE+)>
<!ELEMENT EMPLOYEE (LASTNAME, FIRSTNAME, ADDRESS, PHONE+, EMAIL*, EMPLOYMENT)>
  <!ATTLIST EMPLOYEE EMPID CDATA #REQUIRED
    DEPTID CDATA #REQUIRED>
  <!ELEMENT LASTNAME (#PCDATA)>
  <!ELEMENT FIRSTNAME (#PCDATA)>
  <!ELEMENT ADDRESS (STREETADDRESS, CITY, STATE, ZIP)>
  <!ELEMENT STREETADDRESS (#PCDATA)>
  <!ELEMENT CITY (#PCDATA)>
  <!ELEMENT STATE (#PCDATA)>
  <!ELEMENT ZIP (#PCDATA)>
  <!ELEMENT PHONE (#PCDATA)>
  <!ELEMENT EMAIL (#PCDATA)>
  <!ELEMENT EMPLOYMENT (DATEOFHIRE, SALARY+)>
  <!ATTLIST EMPLOYMENT EMPLSTAT (PF|PP|TF|TP|O) "PF">
  <!ELEMENT DATEOFHIRE (#PCDATA)>
  <!ELEMENT SALARY (#PCDATA)>
```

XML 解析器转换显示了根视图 X_Employees，X_Employees 显示为 X_Employee 的父视图。X_Employee 是 X_Salary、X_Phone 和 X_Email 的父视图。

下图显示了选择创建实体关系时 Designer 创建的 XML 解析器转换：



| Name | Datatype | Length... |
|---------------|----------|-----------|
| DataInput | | |
| DataInput | string | 64000 |
| X_EMPLOYEES | | |
| XPK_EMPLOYEES | integer | 19 |
| X_EMPLOYEE | | |
| XPK_EMPLOYEE | integer | 19 |
| FK_EMPLOYEES | integer | 19 |
| EMPID | string | 50 |
| DEPTID | string | 50 |
| LASTNAME | string | 50 |
| FIRSTNAME | string | 50 |
| STREETADDRESS | string | 50 |
| CITY | string | 50 |
| STATE | string | 50 |
| ZIP | string | 50 |
| EMPLSTAT | string | 2 |
| DATEOFHIRE | string | 50 |
| X_SALARY | | |
| XPK_SALARY | integer | 19 |
| FK_EMPLOYEE1 | integer | 19 |
| SALARY | string | 50 |
| X_PHONE | | |
| XPK_PHONE | integer | 19 |
| FK_EMPLOYEE | integer | 19 |
| PHONE | string | 50 |
| X_EMAIL | | |
| XPK_EMAIL | integer | 19 |
| FK_EMPLOYEE0 | integer | 19 |
| EMAIL | string | 50 |

Designer 创建根视图 X_Employees。X_Employees 是 X_Employee 的父视图。X_Employee 是 X_Salary、X_Phone 和 X_Email 的父视图。

XML 解析器转换中的每个视图至少有一个键，用于建立与其他视图的关系。如果不在 XML 编辑器中指定这些键，Designer 则创建各个视图的主键和外键。这些键的数据类型为 bigint。由于集成服务在每次从 XML 解析器转换返回行时创建键值，因此这些键称为生成的键。

当 Designer 创建主键或外键列时，它将分配具有前缀的列名称。在 XML 定义中，生成的主键列的前缀为 XPK_，生成的外键列的前缀为 XFK_。外键始终引用其他组中的主键。生成的外键列始终引用生成的主键列。

例如，X_Employee 组具有 XPK_Employee 主键。Designer 创建将 X_Phone、X_Email 和 X_Salary 连接到 X_Employee 组的外键列。各个组具有外键列 XFK_Employee。

存储库存储键值。您不能更改存储库中的值，但是可以选择在会话结束后重置或重新开始序列号。

XML 解析器输入验证

您可以将 XML 解析器转换配置为在解析 XML 之前验证该 XML。XML 解析器转换对照架构验证 XML。如果 XML 对架构无效，则出现行错误。XML 解析器转换将 XML 和关联错误消息返回到独立的输出组。您可以将无效 XML 和错误消息传递到目标。

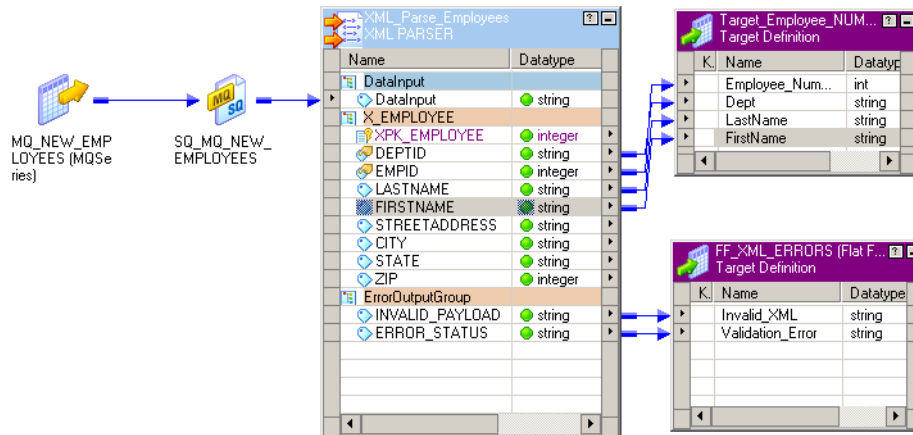
例如，实时 PowerCenter 会话从 WebSphere MQSeries 源读取 XML 消息。使用基于源的提交运行该会话。提交事务中的消息具有无效的 XML 负载。为防止提交失败，您可以配置 XML 解析器转换以将无效 XML 返回到有效数据之外的独立输出组。XML 解析器转换处理有效 XML 消息并完成事务。

会话日志包含指示何时启用“通过数据流路由无效负载”的消息。将会话跟踪级别设置为“普通”时，集成服务将指示验证是否成功的消息写入会话日志。该日志消息包含 XML 解析器为验证 XML 而访问的架构的位置。当启用 XML 流且 XML 无效时，集成服务截断 XML 并将其传递到 Invalid_Payload 端口。集成服务在会话日志中记录该无效 XML。

要配置 XML 解析器转换以验证 XML，请在“中游 XML 解析器”选项卡上启用“通过数据流路由无效负载”选项。Designer 将以下端口添加到 XML 解析器转换：

- **Invalid_Payload**。将无效 XML 消息返回到管道。如果 XML 负载有效，Invalid_Payload 端口包含空值。此端口的精度与 DataInput 端口的精度相同。
- **Error_Status**。包含从 XML 验证返回的错误字符串或状态。如果 XML 对当前行有效，Error_Status 包含空值。此端口的精度与 DataInput 端口的精度相同。

下图显示了将无效 XML 消息路由到“错误”目标表的 XML 解析器转换：



该映射包含以下对象：

- **MQSeries 源定义**。包含消息数据字段中的员工 XML 数据。
- **源限定符转换**。从 WebSphere MQ 读取数据。包含一组表示消息表头字段和消息数据字段的端口。
- **XML 解析器转换**。在 DataInput 端口接收 XML 消息数据。XML 有效时，XML 解析器转换返回员工数据并将其传递到目标。XML 无效时，XML 解析器转换在 Invalid_Payload 端口返回 XML。它还在 Error_Status 端口中返回错误消息。
- **员工目标定义**。接收有效员工数据行。
- **XML_Errors 目标定义**。接收无效 XML 和错误消息。

在转换的会话属性中配置“XML 架构位置”属性。输入验证 XML 所依据的架构的名称和位置。您可以配置 XML 架构定义的工作流、会话或映射变量和参数。如果使用分号将多个架构隔开，可以配置多个用于验证的架构。

如果将 DTD 包含在输入 XML 负载中，则可以将其用于验证。您不能在“XML 架构位置”属性中配置 DTD，也不能将其用于将无效 XML 数据路由到无效负载端口。

如果启用 XML 流，请验证 Invalid_Payload 端口的精度是否与最大消息大小匹配。如果端口精度小于消息大小，XML 解析器转换在 Invalid_Payload 端口返回截断的 XML，并且将错误写入会话日志。

流 XML 到 XML 解析器转换

您可以配置会话，以使 XML 从非结构化数据转换、JMS 源或 WebSphere MQ 源传输到 XML 解析器转换。当 PowerCenter 集成服务传输 XML 数据时，它会将 XML 数据拆分为多个段。

您可以在 XML 解析器转换中配置较小的输入端口，并减少 XML 解析器转换处理大型 XML 文件所需的内存量。您可以解析大于 100 MB 的 XML 文件。

启用 XML 流时，XML 解析器转换接收小于或等于端口大小的分段数据。XML 文件大于端口大小时，PowerCenter 集成服务将多个行传递到 XML 解析器转换。每个 XML 行包含一个流行类型。最后一行包含插入行类型。

输入端口精度必须等于或大于将 XML 传递到 XML 解析器转换的对象的输出端口精度。如果大多数 XML 文档较小，但部分消息较大，可将 XML 解析器转换端口大小设置为较小消息的大小以获得最佳性能。

如果启用 XML 流，还必须对将 XML 数据传递到 XML 解析器转换的源或转换启用 XML 流。如果不启用流，XML 解析器逐行接收 XML，这可能会降低性能。

要在 XML 解析器转换中启用 XML 流，请选择 XML 解析器转换会话属性中的“启用 XML 输入流”。如果在源或转换中启用 XML 流，但未对 XML 解析器转换启用该功能，XML 解析器转换无法处理该 XML 文件。

启用 XML 流且 XML 文档中出现错误时，PowerCenter 集成服务默认将 XML 文档写入会话日志。您可以配置会话以在出现错误时将 XML 文档写入错误日志文件。

启用会话属性中的“记录源行数据”。启用日志记录且 XML 文档出现错误时，PowerCenter 集成服务生成行错误。PowerCenter 集成服务将 XML 文档写入错误日志文件并增加错误计数。

有关 PowerCenter 中的 XML 规模估算的信息，请参阅 [“使用 XML 与 PowerCenter 概览” 页面上 27](#)。有关 PowerCenter 中 XML 处理所适用的局限性的详细信息，请参阅 [“限制” 页面上 27](#)。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

XML 十进制数据类型

将 XML 十进制元素的精度定义为大于 34 位数字时，集成服务调用外部函数将 XML 十进制数据类型转换为 XML 解析器转换中的双精度。该函数返回双精度，其精度长度取决于运行集成服务的节点。在所有平台上，保证精度为舍入数字之前的 17 位，但在某些平台上精度会更高。

例如，在 32 位的 Windows 上，集成服务舍入第 17 位后的数字。

1234.567890123456789 is converted to 1234.567890123460800.

在 32 位的 HP-UX 上，集成服务舍入第 34 位后的数字。

XML 生成器转换

使用 XML 生成器转换合并来自多个源的输入以创建 XML 文档。例如，使用该转换将来自两个 TIBCO 源的 XML 数据合并为一个 TIBCO 目标。其中一个源可包含员工和薪资信息，另一个可包含员工的电话和电子邮件信息。XML 生成器转换是一种活动的已连接转换。

XML 生成器转换类似于 XML 目标定义。集成服务在处理 XML 生成器转换时写入 XML 数据行。集成服务还可以在转换中处理包含非 XML 数据的传递字段。

XML 生成器转换有一个或多个输入组及一个输出组。输出组有一个生成字符串数据 BLOB XML 文档的端口“DataOutput”。创建传递字段时，输出组包含传递端口。

创建中游 XML 转换

创建中游 XML 转换时，请使用 XML 向导和 XML 编辑器定义 XML 组。您可以在 Transformation Developer 和 Mapping Designer 中创建转换。

要创建中游 XML 转换，请执行以下操作：

1. 打开 Transformation Developer 或 Mapping Designer。
2. 单击“转换” > “创建”。
- 此时将显示“创建转换”对话框。
3. 选择 XML 解析器或 XML 生成器转换类型。
4. 输入转换名称，然后单击“创建”。
- 出现“导入 XML 定义”对话框。
5. 选择要导入的文件，然后单击“打开”。
- 将显示 XML 向导。
6. 使用 XML 向导创建 XML 定义。
7. 在 XML 向导中单击“完成”。
- 此时，中游 XML 转换显示在工作区中。
8. 要编辑中游 XML 转换属性，请在工作区中双击转换。

同步中游 XML 定义

您可以使用导入以创建转换的架构或源文件的其他版本来同步中游 XML 转换。例如，要将新元素添加到导入以创建 XML 解析器转换的架构文件。您可以使用新架构更新 XML 解析器转换，而不是删除转换后重新创建该转换。

要同步中游 XML 转换，请使用 Transformation Developer 或 Mapping Designer。

要同步中游 XML 转换，请执行以下操作：

1. 打开 Transformation Developer 或 Mapping Developer。
2. 将要更新的中游 XML 转换或映射拖动到工作区。
3. 右键单击中游 XML 转换的顶部。
4. 选择“同步 XML”转换。
- 出现“导入 XML 定义”对话框。
5. 导航到用于创建 XML 解析器或 XML 生成器转换的存储库定义或文件。
6. 单击“打开”更新转换。

Designer 无法使用不是用于创建转换的文件同步转换。

您可以使用 Source Analyzer 和 Target Designer 同步源和目标 XML 定义。

编辑中游 XML 转换属性

您可以编辑部分中游 XML 转换属性。但是，由于使用 XML 向导和 XML 编辑器定义转换，您必须使用这些工具更改 XML 定义。

在 Mapping Designer 中创建中游 XML 转换时，应遵循以下规则：

- 如果使转换可重用，可以从 Mapping Designer 更改部分转换属性。您不能添加传递端口或元数据扩展。
- 如果创建非可重用转换，您可以通过 Mapping Designer 编辑该转换。

配置中游 XML 转换时，您可以在以下选项卡上配置组件：

- **“转换”选项卡。**在“转换”选项卡上重命名转换和添加说明。
- **“端口”选项卡。**显示在“XML 解析器”或“XML 生成器”选项卡上创建的转换端口和属性。
- **“属性”选项卡。**更新跟踪级别。
- **“初始化属性”选项卡。**创建在初始化过程中外部过程使用的运行时属性。
- **“元数据扩展”选项卡。**通过将信息与存储库对象（如 XML 转换）关联来扩展存储在存储库中的元数据。
- **“端口属性定义”选项卡。**定义应用到转换中的所有端口的端口属性。
- **中游 XML 解析器或 XML 生成器选项卡。**使用此选项卡创建传递端口。传递端口允许您通过转换传递非 XML 数据。对于 XML 解析器转换，如果使用序列编号生成 XML 列名称，您可以选择重置序列号。对于 XML 生成器转换，您可以选择在提交时创建新 XML 文档。

“属性”选项卡

在“属性”选项卡上配置中游 XML 转换属性。

下表介绍了可在“属性”选项卡上更改的选项：

| 转换 | 说明 |
|---------|--|
| 运行时位置 | 包含 DLL 或共享库的位置。默认为 \$PMExtProcDir。输入相对于运行 XML 会话的集成服务节点的路径。 如果此属性为空，集成服务使用在集成服务节点上定义的环境变量查找 DLL 或共享库。 您必须将所有 DLL 或共享库复制到运行时位置或在集成服务节点上定义的环境变量。如果不能找到 DLL、共享库或引用文件，集成服务无法加载过程。 |
| 跟踪级别 | 此转换的会话日志中显示的详细信息量。默认值为“普通”。 |
| 转换范围 | 指示集成服务如何将转换逻辑应用至传入的数据。您可以为 XML 解析器转换选择以下转换范围值之一： <ul style="list-style-type: none">- 行。每次将转换逻辑应用至一行数据。处理下一行之前刷新为所有输出组生成的行。- 事务。将转换逻辑应用至事务中的所有行。当输出块填满且位于文件末尾时，刷新在事务边界生成的行。- 全部输入。将转换逻辑应用至所有传入数据。当输出块填满且位于文件末尾时，仅刷新生成的行。 对于 XML 生成器转换，如果将“提交后”设置设置为“忽略提交”，Designer 将转换范围设置为全部输入。如果将“提交后”设置为“创建新文档”，Designer 将转换范围设置为事务级别。 |
| 输出是可重复的 | 指示多次会话运行的输出数据顺序是否一致。 <ul style="list-style-type: none">- 从不。会话运行之间的输出数据顺序不一致。- 基于输入顺序。当会话运行之间的输入数据顺序一致时，会话运行之间的输出顺序一致。- 始终。即使会话运行之间的输入数据顺序不一致，会话运行之间的输出数据顺序也一致。 对于 XML 解析器转换，默认为“基于输入顺序”。对于 XML 生成器转换，默认为“始终”。 |

| 转换 | 说明 |
|------------|--|
| 要求每个分区一个线程 | 指明集成服务是否必须使用一个线程处理一个分区。 |
| 输出具有确定性 | 指明转换在会话运行之间是否生成相同的输出数据。必须启用此属性以对使用此转换的会话执行恢复。默认为“已启用”。 |

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

中游 XML 解析器选项卡

使用“中游 XML 解析器”选项卡修改 DataInput 端口的大小。还可以在此选项卡上添加传递端口。

您可以通过“中游 XML 解析器”选项卡访问 XML 编辑器。单击“XML 编辑器”按钮。

注意: 访问 XML 编辑器时，只能在退出 XML 编辑器后更新“编辑转换”。

下表介绍了可在“中游 XML 解析器”选项卡上更改的选项：

| 转换 | 说明 |
|-------------|---|
| 精度 | 列的长度。默认 DataInput 端口精度为 64K。传递端口的默认精度为 20。您可以提高该精度。 |
| 重新开始 | 生成的键序列始终从 1 开始。每次运行会话时，XML 定义的所有组中的键序列值重新从 1 开始。 |
| 重置 | 会话结束时，重置所有组中的所有生成的键的值序列。将序列号重置为会话开始前的值。 |
| 通过数据流路由无效负载 | 对照架构验证 XML。如果 XML 对架构无效，则出现行错误。XML 解析器转换将 XML 和关联错误消息返回到独立的输出组。 |
| 说明 | 描述转换。 |

注意: 如果不选择“重置”或“重新开始”，则生成的键中的序列号逐个会话增加。如果选择“重新开始”或“重置”选项，则更新在“初始化属性”选项卡上显示的“重新开始”或“重置”属性。但是，您不能从“初始化属性”选项卡上更改这些属性。

“中游 XML 生成器”选项卡

使用“XML 生成器”选项卡修改 DataOutput 端口的大小。还可以在此选项卡上添加传递端口。

您可以从“中游 XML 生成器”选项卡访问 XML 编辑器。单击“XML 编辑器”按钮。访问 XML 编辑器时，只能在退出 XML 编辑器后编辑转换属性。

下表介绍了可在“XML 生成器转换”选项卡上更改的选项：

| 转换设置 | 说明 |
|------|--|
| 精度 | 列的长度。默认 DataOutput 端口精度为 64K。传递端口的默认精度为 20。您可以提高该精度。 |
| 提交后 | 提交后，集成服务可以生成多个 XML 文档。使用以下选项之一： <ul style="list-style-type: none">- 忽略提交。集成服务创建 XML 文档，并在文件结尾将数据写入文档。如果将两个不同源连接到 XML 生成器转换，请使用此选项。- 创建新文档。每次提交时创建新 XML 文档。如果运行实时会话，请使用此选项。 如果会话使用多个分区，集成服务忽略“提交后”设置，为每个分区生成独立的 XML 文档。如果选择“创建新文档”，集成服务为每个分区创建新文档。 |
| 说明 | 描述转换。 |

注意：将“提交后”设置设置为“忽略提交”时，Designer 将转换范围设置为“全部输入”。如果将“提交后”设置为“创建新文档”，Designer 将转换范围设置为事务级别。

生成传递端口

传递端口是通过中游 XML 转换传递非 XML 数据的列。例如，可以使用 XML 为 MQSeries 源和目标传递消息 ID。使用消息 ID 将请求和回复的输入和输出消息关联。

在中游转换中定义传递端口时，您需要将传递端口添加到 XML 解析器转换中的 DataInput 组或 XML 生成器转换中的 DataOutput 组。

生成端口后，使用 XML 编辑器将对应的引用端口添加到 XML 定义中的其他视图。在 XML 解析器转换中，传递端口为输入端口，对应的引用端口为输出端口。在 XML 生成器转换中，传递端口为输出端口，关联的引用端口为输入端口。

要在中游 XML 转换中创建传递端口，请执行以下操作：

1. 在 Transformation Developer 或 Mapping Designer 中打开转换。
2. 双击转换打开“编辑转换”。
3. 单击“中游 XML 生成器”或“中游 XML 解析器”选项卡。
根据转换类型显示 DataInput 或 DataOutput 端口。
4. 单击“添加”按钮为传递添加输出端口。
此时默认字段显示在“字段名称”列中。
5. 修改字段名称。您还可以根据用于创建定义的文件修改类型、精度和小数位数。
6. 单击“XML 编辑器”打开转换的 XML 定义。
此时定义中的 XML 视图显示在工作区中。
7. 右键单击视图顶部添加引用端口。
8. 选择“添加引用端口”。
此时将打开“引用端口”对话框。
该对话框列出在转换中添加的传递端口。
9. 在视图中选择将与新引用端口对应的传递端口，然后单击“确定”。

此时，对应的输出引用端口显示在视图中。您可以在“列”窗口中将端口重命名为有意义的名称。

10. 单击“应用更改”并退出 XML 编辑器。

11. 在转换中单击“确定”。

非 XML 数据从称为 Pass_thru_field 的输入端口进入并通过对应的 COL_0 引用输出端口传出。

中游 XML 转换故障排除

我需要从包含 XML CLOB 的数据库表提取 XML 文件。每个 XML 文件可能达到 2 GB。创建 XML 解析器转换时，我需要为 CLOB 列定义固定最大长度。但是，CLOB 数据类型的最大长度为 104 MB。

XML 数据过大而无法将其从表直接传递到 XML 解析器转换。您需要将 CLOB 表数据暂存到平面文件，然后从文件创建 XML 源定义。

有关 PowerCenter 中的 XML 规模估算的信息，请参阅[“使用 XML 与 PowerCenter 概览” 页面上 27](#)。有关 PowerCenter 中 XML 处理所适用的局限性的详细信息，请参阅[“限制” 页面上 27](#)。

要创建具有其他元素类型的转换，并转换较大的 XML 输入文件，请使用数据处理器转换。有关如何创建数据处理器转换的详细信息，请参阅《*Informatica Data Transformation 用户指南*》和《*Informatica Data Transformation 入门指南*》。

XML 数据类型引用

XML 和转换数据类型

PowerCenter 支持于 2001 年 5 月 2 日发布的 W3C 推荐中指定的所有 XML 数据类型。下表列出了 XML 数据类型并将它们与 XML 源限定符转换中的转换数据类型进行了对比。有关 XML 数据类型的详细信息，请参阅 XML 数据类型的 W3C 规范，网址为：<http://www.w3.org/TR/xmlschema-2>。

如果导入 XML 文件以创建定义，您可以更改 XML 定义和中游 XML 转换中的数据类型。从 XML 架构导入 XML 数据类型时，不能进行更改。您不能更改映射中 XML 源的转换数据类型。

下表介绍了 XML 和对应的转换数据类型：

| 数据类型 | 转换 | 范围 |
|---------------|-------|--|
| anySimpleType | 字符串 | 1 至 104,857,600 个字符 |
| anyURI | 字符串 | 1 至 104,857,600 个字符 |
| base64Binary | 二进制 | 1 到 104,857,600 个字节 |
| 布尔型 | 小整数 | 精度 5；小数位数 0 |
| 字节 | 小整数 | 精度 5；小数位数 0 |
| 日期 | 日期/时间 | 公元 0001 年 1 月 1 日到公元 9999 年 12 月 31 日（精确到纳秒） |

| 数据类型 | 转换 | 范围 |
|------------|-------|--|
| 日期时间 | 日期/时间 | 公元 0001 年 1 月 1 日到公元 9999 年 12 月 31 日（精确到纳秒） |
| 小数 | 小数 | 精度为 1 到 28；小数位数为 0 到 28 |
| 双精度 | 双精度 | 精度 15，小数位数 0 |
| 持续时间 | 字符串 | 1 至 104,857,600 个字符 |
| ENTITIES | 字符串 | 1 至 104,857,600 个字符 |
| ENTITY | 字符串 | 1 至 104,857,600 个字符 |
| 浮点型 | 双精度 | 精度 15，小数位数 0 |
| gDay | 整型 | -2,147,483,648 到 2,147,483,647 精度 10；小数位数 0 |
| gMonth | 整型 | -2,147,483,648 到 2,147,483,647 精度 10；小数位数 0 |
| gMonthDay | 日期/时间 | 公元 0001 年 1 月 1 日到公元 9999 年 12 月 31 日（精确到纳秒） |
| gYear | 整型 | 精度 10；小数位数 0 |
| gYearMonth | 日期/时间 | 公元 0001 年 1 月 1 日到公元 9999 年 12 月 31 日（精确到纳秒） |
| hexBinary | 二进制 | 1 到 104,857,600 个字节 |
| ID | 字符串 | 1 至 104,857,600 个字符 |
| IDREF | 字符串 | 1 至 104,857,600 个字符 |
| IDREFS | 字符串 | 1 至 104,857,600 个字符 |
| int | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| 整型 | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| 语言 | 字符串 | 1 至 104,857,600 个字符 |
| 长整型 | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| 名称 | 字符串 | 1 至 104,857,600 个字符 |
| Ncname | 字符串 | 1 至 104,857,600 个字符 |

| 数据类型 | 转换 | 范围 |
|--------------------|-------|--|
| negativeInteger | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| NMTOKEN | 字符串 | 1 至 104,857,600 个字符 |
| NMTOKENS | 字符串 | 1 至 104,857,600 个字符 |
| nonNegativeInteger | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| nonPositiveInteger | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| normalizedString | 字符串 | 1 至 104,857,600 个字符 |
| NOTATION | 字符串 | 1 至 104,857,600 个字符 |
| positiveInteger | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| QName | 字符串 | 1 至 104,857,600 个字符 |
| 短整数 | 小整数 | 精度 5；小数位数 0 |
| 字符串 | 字符串 | 1 至 104,857,600 个字符 |
| 时间 | 日期/时间 | 公元 0001 年 1 月 1 日到公元 9999 年 12 月 31 日（精确到纳秒） |
| 标志 | 字符串 | 1 至 104,857,600 个字符 |
| unsignedByte | 小整数 | 精度 5；小数位数 0 |
| unsignedInt | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| unsignedLong | 长整型 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 精度 19；小数位数 0 |
| unsignedShort | 整型 | -2,147,483,648 到 2,147,483,647 精度 10；小数位数 0 |

XML 日期格式

对于日期、时间和日期时间数据类型，PowerCenter 支持以下格式：

CCYY-MM-DDThh:mm:ss:sss

请在 XML 文件中使用此格式或此格式的任何部分。PowerCenter 不支持负日期的日期时间格式。

请在会话过程中使用以下格式的 date、time 或 datetime 元素。

CCYY-MM

-或-

CCYY-MM-DD/Thh

XML 文件中的第一个 datetime 元素的格式确定该元素后面所有值的格式。如果集成服务读取具有不同格式的同一天、time 或 datetime 元素的值时，集成服务将拒绝该行。

例如，如果 datetime 元素的第一个值的格式如下：

CCYY-MM-DDThh:mm:ss

集成服务拒绝包含以下格式的元素行：

CCYY-MM-DD

XML 解析器将输入 XML 中的日期时间值转换为托管集成服务的计算机的本地时区。如果在 Windows 上启用调整时钟以适应夏令时变更的选项，XML 解析器将日期时间值增加一个小时。为了获得一致的日期时间值转换，请勿在 Windows 上启用调整时钟以适应夏令时变更的选项。

XPath 查询函数参考

XPath 查询函数概览

XPath 是一种描述如何定位 XML 文档中的项的语言。XPath 使用基于从根组件通过整个 XML 层次结构的路径的定位语法。您可以为视图行或具有包含视图行的 XPath 的列中的元素创建 XPath 查询谓词。

可以使用 XML 视图中的 XPath 查询谓词筛选 XML 源数据。在会话中，集成服务根据查询从源 XML 文件提取数据。如果所有查询都返回 TRUE，集成服务提取该视图的数据。

XPath 查询谓词包含要提取的元素或属性及确定条件的查询。您可以验证元素或属性的值，或者验证源 XML 数据中是否存在某个元素或属性。

本附录介绍了用于 XPath 查询谓词的各个函数。函数接受参数并返回值。创建函数时，您可以包含 XML 视图中的元素和属性的组件，还可以添加文字值。指定文字时，必须使用单引号或双引号将文字括起来。

函数快速参考

可在 XPath 查询谓词中使用以下类型的函数：

- **字符串。**使用字符串函数测试子字符串值，连接字符串，或者将字符串转换为其他字符串。例如，以下 XPath 查询谓词确定员工的全名是否等于姓氏和名字相连：

```
EMPLOYEE[./FULLNAME=concat(./ENAME/LASTNAME,./ENAME/FIRSTNAME)]
```

- **数值。**将数值函数与元素和属性值一起使用。数值函数对数值进行操作并返回整数。例如，以下 XPath 查询谓词舍入折扣并测试结果是否大于 15：

```
ORDER_ITEMS[round(./DISCOUNT) > 15]
```

- **布尔型。**使用布尔函数测试元素，检查语言属性，或者强制返回 True 或 False 结果。例如，如果值大于零，以下 XPath 查询谓词返回 True：

```
boolean(string)
```

下表介绍了 XPath 查询谓词字符串函数：

| 函数 | 语法 | 说明 |
|------------------|--|----------------------------|
| concat | concat (string1, string2) | 连接两个字符串。 |
| contains | contains (string, substring) | 确定字符串是否包含另一个字符串。 |
| normalize-space | normalize-space (string) | 删除字符串的前导空格和尾随空格。 |
| starts-with | starts-with (string, substring) | 确定 string1 是否以 string2 开头。 |
| 字符串 | string (value) | 将数字或布尔值转换为字符串。 |
| string-length | string-length (string) | 返回字符串中的字符数，其中包括尾随空白。 |
| substring | substring (string, start [,length]) | 返回字符串中从指定位置开始的部分。 |
| substring-after | substring-after (string, substring) | 返回字符串中从指定位置开始的部分。 |
| substring-before | substring-before (string, substring) | 返回字符串中的子字符串之前的字符。 |
| translate | translate (string1, string2, string3) | 将字符串中的字符转换为其他字符。 |

下表介绍了 XPath 查询谓词数字函数：

| 函数 | 语法 | 说明 |
|---------|--------------------|------------------------|
| ceiling | ceiling (number) | 将数字舍入为大于或等于传递的数字的最小整数。 |
| floor | floor (number) | 将数字舍入为小于或等于传递的数字的最大整数。 |
| number | number (value) | 将字符串或布尔值转换为数字。 |
| round | round (number) | 将数字舍入为最接近的整数。 |

下表介绍了 XPath 查询谓词布尔函数：

| 函数 | 语法 | 说明 |
|-------|--------------------|----------------------------------|
| 布尔型 | boolean (object) | 将对象转换为布尔值。 |
| false | false () | 始终返回 FALSE。 |
| lang | lang (code) | 确定元素是否有与 code 参数匹配的 xml:lang 属性。 |

| 函数 | 语法 | 说明 |
|------|-------------------|--|
| not | not (condition) | 如果布尔值 condition 为 FALSE，则返回 True；如果布尔值 condition 为 TRUE，则返回 FALSE。 |
| true | true () | 始终返回 TRUE。 |

布尔型

将值转换为布尔值。

语法

```
boolean ( object )
```

下表介绍了布尔型参数：

| 参数 | 说明 |
|---------------|--------------------------|
| <i>object</i> | 数值或字符串数据类型。传递要测试的数字或字符串。 |

返回值

布尔型。

该函数返回布尔值，如下所示：

- 字符串长度不为零时返回 TRUE；否则，返回 FALSE。
- 数字为零或为非数字 (NaN) 时返回 FALSE；否则，返回 TRUE。

示例

以下示例验证名称是否包含字符：

```
boolean ( NAME )
```

下表包含参数和返回值的示例：

| NAME | RETURN VALUE |
|-------|--------------|
| Lilah | TRUE |
| - | FALSE |

以下示例验证邮政编码是否为数值：

```
boolean ( ZIP_CODE )
```

下表包含参数和返回值的示例：

| ZIP_CODE | RETURN VALUE |
|----------|--------------|
| 94061 | TRUE |
| 94005 | TRUE |

| ZIP_CODE | RETURN VALUE |
|----------|--------------|
| 9400g | FALSE |

ceiling

将数字舍入为大于或等于传递的数字的最小整数。

语法

`ceiling (number)`

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|------------|
| <i>number</i> | 数值。要舍入的数字。 |

返回值

整数。

示例

以下表达式返回舍入为最小整数的价格：

`ceiling (PRICE)`

下表包含参数和返回值的示例：

| PRICE | RETURN VALUE |
|---------|--------------|
| 39.79 | 40 |
| 125.12 | 126 |
| 74.24 | 75 |
| NULL | NULL |
| -100.99 | -100 |
| 100 | 100 |

concat

连接两个字符串。

语法

`concat (string1, string2)`

下表介绍了此函数的参数：

| 参数 | 说明 |
|----------------|-----------------------|
| <i>string1</i> | 字符串数据类型。传递要连接的第一个字符串。 |
| <i>string2</i> | 字符串数据类型。传递要连接的第二个字符串。 |

返回值

字符串。

如果其中一个字符串为空，则 concat 忽略它并返回另一个字符串。

示例

以下表达式连接 FIRSTNAME 和 LASTNAME：

```
concat( FIRSTNAME, LASTNAME )
```

下表包含参数和返回值的示例：

| FIRSTNAME | LASTNAME | RETURN VALUE |
|-----------|----------|--------------|
| John | Baer | JohnBaer |
| NULL | Campbell | Campbell |
| Greg | NULL | Greg |
| NULL | NULL | NULL |

提示

concat 函数不会将空格添加到字符串。要在两个字符串之间添加空格，您可以编写包含嵌套的 concat 函数的表达式。例如，以下表达式在名字的结尾添加一个空格，然后连接名字和姓氏：

```
concat ( concat ( FIRST_NAME, " " ), LAST_NAME )
```

下表显示参数和返回值的示例：

| FIRST_NAME | LAST_NAME | RETURN VALUE |
|------------|-----------|--|
| John | Baer | John Baer |
| NULL | Campbell | Campbell (<i>includes leading space</i>) |
| Greg | NULL | Greg |

contains

确定字符串是否包含另一个字符串。

语法

```
contains( string, substring )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|------------------|----------------------------------|
| <i>string</i> | 字符串数据类型。传递要检查的字符串。该参数区分大小写。 |
| <i>substring</i> | 字符串数据类型。传递要在字符串中搜索的字符串。该参数区分大小写。 |

返回值

布尔型。

示例

如果 NAME 包含 SHORTNAME，以下表达式返回 TRUE：

```
contains( NAME, SHORTNAME )
```

下表包含参数和返回值的示例：

| NAME | SHORTNAME | RETURN VALUE |
|----------------|-----------|--------------|
| John | Baer | FALSE |
| SuzyQ | Suzy | TRUE |
| WorldPeace | World | TRUE |
| CASE_SENSITIVE | case | FALSE |

false

始终返回 FALSE。使用此函数将布尔值设置为 FALSE。

语法

```
false ()
```

false 函数不接受参数。

返回值

FALSE.

示例

将 false 函数与其他函数结合以强制返回 FALSE 结果。

下表包含返回 FALSE 的表达式：

| EXPRESSION | RETURN VALUE |
|-------------------------------------|--------------|
| (salary) = false() | FALSE |
| A/B = false () | FALSE |
| starts-with (name, 'T') = false() | FALSE |

floor

将数字舍入为小于或等于传递的数字的最大整数。

语法

```
floor( number )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|-------------|
| <i>number</i> | 数值。使用数值表达式。 |

返回值

整数。

传递至函数的值为空时返回空值。

示例

以下表达式返回小于或等于 BANK_BALANCE 中的值的最大整数：

```
floor( BANK_BALANCE )
```

下表包含参数和返回值的示例：

| BANK_BALANCE | RETURN VALUE |
|--------------|--------------|
| 39.79 | 39 |
| NULL | NULL |
| -100.99 | -101 |
| 5 | 5 |

lang

元素所包含的 xml:lang 属性与 code 参数指定的语言相同时返回 TRUE。使用 lang 函数以按语言选择 XML。xml:lang 属性是标识元素内容语言的代码。元素可包含多种语言的文本。

语法

```
lang ( code )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|-------------|---------------------|
| <i>code</i> | 字符串数据类型。传递元素内容语言代码。 |

返回值

布尔型。

示例

以下表达式检查元素内容语言代码：

```
lang ( 'en' )
```

下表包含参数和返回值的示例：

| XML | RETURN VALUE |
|--|--------------|
| <Phrase xml:lang=" es" > El perro esta en la casa. </Phrase> | FALSE |
| <Phrase xml:lang=" en" > The dog is in the house. </Phrase> | TRUE |

normalize-space

删除字符串中的前导空格和尾随空格。空格包含不显示的字符，如空格字符和制表符。此函数将空格序列替换为单个空格。

语法

```
normalize-space ( string )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|---------------------|
| <i>string</i> | 字符串数据类型。传递包含空格的字符串。 |

返回值

字符串。

字符串为空时返回空值。

示例

以下表达式删除名称中的多余空格：

```
normalize-space ( NAME )
```

下表包含参数和返回值的示例：

| NAME | RETURN VALUE |
|-----------|--------------|
| Jack Dog | Jack Dog |
| Harry Cat | Harry Cat |

not

返回布尔值 *condition* 的相反值。如果 *condition* 为 False，该函数返回 TRUE；如果 *condition* 为 True，则返回 FALSE。

语法

```
not ( condition )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|------------------|-----------|
| <i>condition</i> | 布尔型表达式或值。 |

返回值

布尔型。

condition 为空时返回空值。

示例

以下表达式返回布尔型 *condition* 的相反值。

```
not ( EMPLOYEE = concat ( FIRSTNAME, LASTNAME ) )
```

下表包含参数和返回值的示例：

| EMPLOYEE | FIRSTNAME | LASTNAME | RETURN |
|----------|-----------|----------|--------|
| Fullname | Full | Name | FALSE |
| Lastname | Lastname | First | TRUE |

number

将字符串或布尔值转换为数字。

语法

```
number ( value )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|--------------|-------------|
| <i>value</i> | 使用布尔值或字符串值。 |

返回值

该函数为以下数据返回数字：

- 如果字符串包含数值字符，则该字符串转换为数字。该字符串可以包含空格且在数字之前可包含减号。在字符串中，数字之后可存在空格。其他任何字符串则为非数字 (NaN)。

- 布尔值 TRUE 将转换为 1。布尔值 FALSE 将转换为 0。

如果作为参数传递到该函数的值不是数字，则函数返回“非数字 (NaN)”。

示例

以下表达式将付款转换为数字：

```
number ( PAYMENT )
```

下表包含参数和返回值的示例：

| PAYMENT | RETURN VALUE |
|----------|--------------|
| '850.00' | 850.00 |
| TRUE | 1 |
| FALSE | 0 |
| AB | NaN |

round

将数字舍入为最接近的整数。如果数字介于两个整数之间，round 返回较大的数字。

语法

```
round ( number )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|-----------------------|
| <i>number</i> | 数值。传递数值数据类型或产生数字的表达式。 |

返回值

整数。

示例

以下表达式舍入 BANK_BALANCE：

```
round( BANK_BALANCE )
```

下表包含参数和返回值的示例：

| BANK_BALANCE | RETURN VALUE |
|--------------|--------------|
| 12.34 | 12 |
| 12.50 | 13 |
| -18.99 | -19 |
| NULL | NULL |

starts-with

如果第一个字符串以第二个字符串开头，则返回 TRUE。否则，返回 FALSE。

语法

```
starts-with ( string, substring )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|------------------|------------------------------------|
| <i>string</i> | 字符串数据类型。传递要搜索的字符串。该字符串区分大小写。 |
| <i>substring</i> | 字符串数据类型。传递要在字符串中搜索的子字符串。子字符串区分大小写。 |

返回值

布尔型。

示例

以下表达式确定 NAME 是否以 FIRSTNAME 开头：

```
starts-with ( NAME, FIRSTNAME )
```

下表包含参数和返回值的示例：

| NAME | FIRSTNAME | RETURN VALUE |
|---------------|-----------|--------------|
| Kathy Russell | Kathy | TRUE |
| Joe Abril | Mark | FALSE |

字符串

将数字或布尔值转换为字符串。

语法

```
string ( value )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|----------|------------------|
| <i>值</i> | 数值或布尔值。传递数值或布尔值。 |

返回值

字符串。

未传递值时返回空字符串。传递空值时返回空值。

string 函数按以下方法将数字转换为字符串：

- 如果数字为整数，该函数返回以十进制表示的无小数点和前导零的字符串。
- 如果数字不是整数，该函数返回包含小数点的字符串，并且小数点之前和之后都至少有一位数。
- 如果数字为负数，该函数返回包含减号 (-) 的字符串。

string 函数将布尔值转换为字符串，如下所示：

- 如果布尔值为 FALSE，该函数返回字符串 “false”。
- 如果布尔值为 TRUE，该函数返回字符串 “true”。

示例

以下表达式从数值参数 speed 返回字符串：

```
string( SPEED )
```

下表包含参数和返回值的示例：

| SPEED | RETURN VALUE |
|----------|--------------|
| 10.99 | '10.99' |
| 15.62567 | '15.62567' |
| 0 | '0' |
| 10 | '10' |
| 50 | '50' |
| 1.3 | '1.3' |

以下表达式由布尔型参数 STATUS 返回字符串：

```
string( STATUS )
```

下表显示参数和返回值的示例：

| STATUS | RETURN VALUE |
|--------|--------------|
| TRUE | 'true' |
| FALSE | 'false' |
| NULL | NULL |

string-length

返回字符串中的字符数，其中包括尾随空白。

语法

```
string-length ( string )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|------------------|
| <i>string</i> | 字符串数据类型。要计算的字符串。 |

返回值

整数。

传递至函数的值为空时返回空值。

示例

以下表达式返回客户名称的长度：

```
string-length ( CUSTOMER_NAME )
```

下表包含参数和返回值的示例：

| CUSTOMER_NAME | RETURN VALUE |
|---------------|--------------|
| Bernice Davis | 13 |
| NULL | NULL |
| John Baer | 9 |

substring

返回字符串中从指定位置开始的部分。子字符串包含作为字符串中的字符的空白。

语法

```
substring ( string, start [ , length ] )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|---------------|---|
| <i>string</i> | 字符串数据类型。要搜索的字符串。 |
| <i>start</i> | 整数数据类型。传递字符串中开始计数的位置。如果起始位置为正数，则 substring 从字符串的开头计数以查找起始位置。第一个字符为一。如果起始位置为负数，substring 从字符串的末尾计数以查找起始位置。 |
| <i>length</i> | 整数数据类型。必须大于零。传递字符串中要返回的字符数。如果省略 length 参数，substring 返回从起始位置至字符串末尾的所有字符。 |

返回值

字符串。

如果字符串包含数值，该函数返回字符串。

如果传递负整数或零，则函数返回空字符串。

传递至函数的值为空时返回空值。

示例

以下表达式返回 PHONE 中的区号：

```
substring( PHONE, 1, 3 )
```

| PHONE | RETURN VALUE |
|--------------|--------------|
| 809-555-3915 | 809 |
| NULL | NULL |

以下表达式返回不带区号的 PHONE：

```
substring ( phone, 5, 8 )
```

下表包含参数和不带区号的返回值的示例：

| PHONE | RETURN VALUE |
|--------------|--------------|
| 808-555-0269 | 555-0269 |
| NULL | NULL |

可以传递负起始值以从字符串的右边开始计数。以下表达式针对 *length* 参数从左到右读取源字符串：

```
substring ( PHONE, -8, 3 )
```

下表包含参数和表达式从左到右读取源字符串时的返回值的示例。

| PHONE | RETURN VALUE |
|--------------|--------------|
| 808-555-0269 | 555 |
| 809-555-3915 | 555 |
| 357-687-6708 | 687 |
| NULL | NULL |

length 参数大于字符串长度时，substring 返回从起始位置至字符串末尾的所有字符。例如：

```
substring ( 'abcd', 2, 8 )
```

返回 “bcd” 。

```
substring ( 'abcd', -2, 8 )  
returns 'cd.'
```

substring-after

返回字符串中子字符串之后的字符。

语法

```
substring-after ( string, substring )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|------------------|--------------------------|
| <i>string</i> | 字符串数据类型。传递要搜索的字符串。 |
| <i>substring</i> | 字符串数据类型。传递要在字符串中搜索的子字符串。 |

返回值

字符串。

未找到子字符串时返回空字符串。

传递至函数的值为空时返回空值。

示例

以下表达式返回 PHONE 中的区号 (415) 之后的字符串：

```
substring-after ( PHONE, (415) )
```

下表包含参数和返回值的示例：

| PHONE | RETURN VALUE |
|---------------|--------------|
| (415)555-1212 | 555-1212 |
| (408)368-4017 | - |
| NULL | NULL |
| (415)366-7621 | 366-7621 |

substring-before

返回子字符串之前的字符串部分。

语法

```
substring-before ( string, substring )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|------------------|--------------------------|
| <i>string</i> | 字符串数据类型。传递要搜索的字符串。 |
| <i>substring</i> | 字符串数据类型。传递要在字符串中搜索的子字符串。 |

返回值

字符串。

未找到子字符串时返回空字符串。

传递至函数的值为空时返回空值。

示例

以下表达式返回地址 “Third Street” 中存在的编号：

```
substring-before ( ADDRESS, Third Street )
```

下表包含属性和返回值的示例：

| ADDRESS | RETURN VALUE |
|------------------|--------------|
| 100 Third Street | 100 |
| 250 Third Street | 250 |
| 600 Third Street | 600 |
| NULL | NULL |

translate

将字符串中的字符转换为其他字符。该函数使用另外两个字符串作为转换对。

语法

```
translate ( string1, string2, string3 )
```

下表介绍了此函数的参数：

| 参数 | 说明 |
|----------------|--|
| <i>字符串 1</i> | 字符串数据类型。传递要转换的字符串。 |
| <i>字符串 2</i> | 字符串数据类型。传递定义要转换的字符的字符串。Translate 将 <i>string1</i> 中的每个字符替换为表示该字符在 <i>string2</i> 中的位置的数字。 |
| <i>string3</i> | 字符串数据类型。传递定义将加密的 <i>string1</i> 中的字符转换为哪些内容的字符串。Translate 将加密的 <i>string1</i> 中的每个字符替换为位于 <i>string2</i> 的位置编号的 <i>string3</i> 中的字符。 |

返回值

字符串。

示例

以下表达式使用另外两个字符串转换字符串：

```
translate ( EXPRESSION, STRING2, STRING3 )
```

下表包含参数和返回值的示例：

| EXPRESSION | STRING2 | STRING3 | RETURN VALUE |
|-----------------|---------|---------|-----------------|
| A Space Odissei | i | y | A Space Odyssey |
| rats | tras | TCAS | CATS |

| EXPRESSION | STRING2 | STRING3 | RETURN VALUE |
|------------|---------|---------|--------------|
| bar | abc | ABC | BAr |

Translate 不更改 EXPRESSION 中的不出现在 string2 中的字符。如果某个字符出现在 EXPRESSION 和 string2 中，但不出现在 string3 中，则返回的字符串不包含该字符。

true

始终返回 TRUE。使用此函数将布尔值设置为 TRUE。

语法

```
true ()
```

true 函数不接受参数。

返回值

布尔值 TRUE。

示例

下表包含返回 TRUE 的表达式示例：

| EXPRESSION | RETURN VALUE |
|------------------------------------|--------------|
| (decision) = true () | TRUE |
| A/B = true () | TRUE |
| (starts-with (name, 'T'))= true | TRUE |