



XML 服务

**Adaptive Server® Enterprise**

15.5

文档 ID: DC20145-01-1550-01

最后修订日期: 2009 年 10 月

版权所有 © 2010 Sybase, Inc. 保留所有权利。

除非在新版本或技术声明中另有说明, 否则本出版物适用于 Sybase 软件 and 任何后续版本。本文档中的信息如有更改, 恕不另行通知。此处说明的软件按许可协议提供, 其使用和复制必须符合该协议的条款。

若要订购附加文档, 美国和加拿大的客户请拨打客户服务部门电话 (800) 685-8225 或发传真至 (617) 229-9845。

持有美国许可协议的其它国家 / 地区的客户可通过上述传真号码与客户服务部门联系。所有其它国际客户请与 Sybase 子公司或当地分销商联系。仅在定期安排的软件发布日期提供升级。未经 Sybase, Inc. 的事先书面许可, 本书的任何部分不得以任何形式、任何手段 (电子的、机械的、手动、光学的或其它手段) 进行复制、传播或翻译。

可在位于 <http://www.sybase.com/detail?id=1011207> 的 “Sybase 商标页” (Sybase trademarks page) 处查看 Sybase 商标。Sybase 和列出的标记均是 Sybase, Inc. 的商标。® 表示已在美国注册。

Java 和所有基于 Java 的标记都是 Sun Microsystems, Inc. 在美国和其它国家 / 地区的商标或注册商标。

Unicode 和 Unicode 徽标是 Unicode, Inc. 的注册商标。

IBM 和 Tivoli 是 International Business Machines Corporation 在美国和 / 或其它国家 / 地区的注册商标。

提到的所有其它公司和产品名均可能是与之相关的相应公司的商标。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# 目录

关于本手册 .....	ix
第 1 章	XML 服务简介 ..... 1
	XML capabilities ..... 1
	概述 ..... 2
	数据库中的 XML ..... 2
	XML 示例文档 ..... 3
	XML 文档类型 ..... 7
第 2 章	XML 查询函数 ..... 9
	XML 查询函数 ..... 9
	示例部分 ..... 10
	xmlextract ..... 10
	xmltest ..... 17
	xmlparse ..... 22
	xmlrepresentation ..... 25
	xmlvalidate ..... 27
	option_strings: 一般格式 ..... 36
	查询函数的选项值 ..... 37
第 3 章	XML 语言和 XML 查询语言 ..... 39
	字符集支持 ..... 39
	URI 支持 ..... 39
	命名空间支持 ..... 40
	XML 模式支持 ..... 40
	XML 文档中的预定义实体 ..... 40
	XPath 查询中的预定义实体 ..... 42
	空白 ..... 42
	空元素 ..... 43
	XML 查询语言 ..... 43
	支持的 XPath 语法和标识 ..... 43
	XPath 运算符 ..... 45
	XPath 函数 ..... 47

	带括号的表达式 .....	52
	括号和下标 .....	52
	括号和联合 .....	53
<b>第 4 章</b>	<b>XML 映射函数 .....</b>	<b>55</b>
	for xml 子句 .....	55
	for xml 子查询 .....	59
	for xml schema 和 for xml all .....	61
	forxmlj、forxmldtdj、forxmlschemaj、forxmlallj .....	65
	forsqlcreatej、forsqlinsertj、forsqlscriptj .....	70
	使用 Java 函数映射分层 XML 文档和 SQL 数据 .....	74
	示例数据及其树形结构的 XML 表示形式 .....	75
	使用 ForXmlTree 将 SQL 数据映射到分层 XML .....	76
	使用 OpenXml 将分层 XML 映射到 SQL .....	77
	用于多个结果集查询的 Java SQLX 映射 .....	79
	forxmlmultiplej .....	79
<b>第 5 章</b>	<b>XML 映射 .....</b>	<b>81</b>
	SQLX 选项 .....	82
	SQLX 选项定义 .....	83
	SQLX 数据映射 .....	92
	映射重复列名和未命名列 .....	92
	将 SQL 名称映射到 XML 名称 .....	95
	将 SQL 值映射到 XML 值 .....	97
	SQLX 模式映射 .....	99
	概述 .....	99
	选项: columnstyle=element .....	101
	选项: columnstyle=attribute .....	102
	选项: nullstyle=omit .....	103
	选项: nullstyle=attribute .....	104
<b>第 6 章</b>	<b>对 I18N 提供的 XML 支持 .....</b>	<b>105</b>
	概述 .....	105
	Unicode 数据类型 .....	105
	代理对 .....	106
	数值字符表示 .....	106
	客户端 / 服务器转换 .....	106
	字符集和 XML 数据 .....	107
	for xml 中的 I18N .....	107
	选项字符串 .....	108
	for xml 中的数值字符表示 .....	108
	header 选项 .....	109

	异常 .....	109
	示例 .....	109
	xmlparse 中的 l18N .....	113
	选项 .....	113
	xmlextract 中的 l18N .....	114
	NCR 选项 .....	114
	xmlextract 中的排序顺序 .....	114
	XML 服务中的排序顺序 .....	115
	xmlvalidate 中的 l18n .....	116
	NCR 选项 .....	116
<b>第 7 章</b>	<b>xmltable() .....</b>	<b>117</b>
	简介 .....	117
	xmltable 及派生表的语法 .....	117
	xmltable .....	118
<b>附录 A</b>	<b>sample_docs 示例表 .....</b>	<b>135</b>
	sample_docs 表的列和行 .....	135
	Sample_docs 表的列 .....	135
	sample_docs 表的行 .....	136
	sample_docs 表 .....	138
	表脚本（publishers 表） .....	138
	Publishers 表的表示形式 .....	139
	Titles 表的表示形式 .....	140
<b>附录 B</b>	<b>XML 服务和外部文件系统访问 .....</b>	<b>145</b>
	入门 .....	145
	启用 XML 服务和外部文件系统访问 .....	145
	外部文件系统的字符集转换 .....	146
	示例 .....	146
	设置 XML 文档并创建代理表 .....	146
	示例：从 XML 文档中提取书名 .....	148
	示例：将 XML 文档或 XML 查询结果 导入 Adaptive Server 表 .....	148
	示例：将经过分析的 XML 文档存储在文件系统中 .....	149
	示例：针对外部文件访问的“xmlerror”选项功能 .....	150
	示例：在 xmlextract 中指定“xmlerror=message”选项 .....	151
	示例：用“xmlerror=message”选项分析 XML 和 非 XML 文档 .....	151
	示例：对非 XML 文档使用“xmlerror=null”选项 .....	153

<b>附录 C</b>	<b>设置 XML 服务 .....</b>	<b>155</b>
	安装基于 Java 的 SQLX 映射函数 .....	155
	基于 Java 的 XML 函数 .....	155
	映射函数安装 .....	156
 <b>附录 D</b>	 <b>基于 Java 的 XQL 处理器 .....</b>	 <b>159</b>
	设置基于 Java 的 XQL 处理器 .....	159
	设置 CLASSPATH 环境变量 .....	159
	在 Adaptive Server 中安装基于 Java 的 XQL 处理器 .....	160
	在 Adaptive Server 内运行基于 Java 的 XQL 处理器时的 内存要求 .....	161
	使用基于 Java 的 XQL 处理器 .....	161
	将原始 XML 文档转换为已分析的版本 .....	161
	插入 XML 文档 .....	162
	更新 XML 文档 .....	162
	删除 XML 文档 .....	162
	使用 XQL .....	162
	影响性能的查询结构 .....	165
	示例 .....	165
	基于 Java 的 XQL 处理器的其它用法 .....	166
	com.sybase.xml.xql.XqlDriver 语法 .....	166
	验证文档 .....	168
	将基于 Java 的 XQL 处理器用于独立应用程序 .....	169
	com.sybase.xml.xql.Xql 中的方法 .....	171
	parse(String xmlDoc) .....	171
	parse(InputStream xml_document, boolean validate) .....	171
	query(String query, String xmlDoc) .....	172
	query(String query, InputStream xmlDoc) .....	173
	query(String query, SybXmlStream xmlDoc) .....	173
	sybase.aseutils.SybXmlStream .....	173
	com.sybase.xml.xql.store.SybMemXmlStream .....	174
	com.sybase.xml.xql.store.SybFileXmlStream .....	174
	setParser(String parserName) .....	174
	resetParser .....	175

---

<b>附录 E</b>	<b>在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移 .....</b>	<b>177</b>
	简介 .....	177
	迁移文档和查询 .....	178
	在基于 Java 的 XQL 处理器和	
	本机 XML 处理器之间迁移文档 .....	178
	在基于 Java 的 XQL 处理器和	
	本机 XML 处理器之间迁移文本文档 .....	178
	从重新生成的副本迁移文档 .....	178
	从基于 Java 的 XQL 处理器重新生成文本文档 .....	179
	从本机 XML 处理器重新生成文本文档 .....	180
	在本机 XML 处理器和	
	基于 Java 的 XQL 处理器之间迁移查询 .....	180
<b>附录 F</b>	<b>关于 xmldata() 的示例应用程序 .....</b>	<b>181</b>
	示例表 .....	181
	使用 depts 文档 .....	185
	depts 文档的结构 .....	185
	用 depts 文档创建 SQL 表 .....	186
<b>索引</b>	<b>.....</b>	<b>189</b>





# 关于本手册

## 读者

需要执行以下操作的客户：

- 将完整的 XML 文档存储在 SQL 数据库中
- 对 SQL 数据库中的 XML 文档进行测试和数据抽取
- 存储从 XML 文档抽取的数据
- 用 SQL 数据生成 XML 文档
- 将 SQL 数据作为 XML 处理

## 如何使用本手册

本手册中的信息是按以下方式组织的：

- 第 1 章“XML 服务简介”介绍数据库中的 XML 以及 Sybase XML 服务的新 XML 功能。
- 第 2 章“XML 查询函数”探讨在 SQL 语句内处理和查询 XML 文档。您既可以将这些查询函数应用于存储的 XML 文档（即常规的用户文档），也可以应用于通过 `for xml` 子句或 `forxmlj` 函数（或通过提供 SQL 数据 XML 视图的任何类似工具）生成的 SQLX-XML 文档。有关这些函数的详细信息，请参见第 4 章“XML 映射函数”。
- 第 3 章“XML 语言和 XML 查询语言”介绍 XML 查询函数所支持的 XML 文档和查询语言功能，包括所支持的 XPath 语言子集的规范。
- 第 4 章“XML 映射函数”讨论在 SQL 数据和 SQLX-XML 格式的 XML 文档之间进行映射的函数。
- 第 5 章“XML 映射”介绍 XML 映射函数支持的 XML 文档的 SQLX-XML 格式。
- 第 6 章“对 I18N 提供的支持”介绍为支持非 ASCII 数据而对 XML 服务进行的扩展。
- 附录 A “设置 XML 服务”包括关于安装本机 C++ 处理器和 Adaptive Server 版本 12.5 及更高版本附带的 Java 处理器的指南。
- 附录 B “*sample\_docs* 示例表”介绍函数示例中使用的 *sample\_docs* 表。

- 
- 附录 C “XML 服务和外部文件系统访问” 包含如何通过 XFS 使用 XML 功能的示例。
  - 附录 D “基于 Java 的 XQL 处理器” 介绍如何使用 XQL 从 Adaptive Server 中选择原始数据，使用 XQL 以及以 XML 文档的形式显示结果。
  - 附录 E “在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移” 介绍用于实现查询语言和以经过分析的格式返回文档的不同函数和方法，并讨论了如何从一种函数或方法切换到另一种函数或方法。
  - 附录 F 是一个关于 `xmltable()` 的示例应用程序。

## 相关文档

Adaptive Server<sup>®</sup> Enterprise 文档集由以下文档组成：

- 针对所用平台的发行公告 — 包含未能及时写入手册的最新信息。  
最新版本的发行公告可以从万维网上获得。若要了解本产品 CD 发行之后增加的重要产品或文档信息，请使用 Sybase Technical Library。
- 针对所用平台的《安装指南》— 介绍所有 Adaptive Server 产品及相关的 Sybase 产品的安装、升级和某些配置过程。
- 《Adaptive Server Enterprise 中的新增功能》— 介绍 Adaptive Server 15.0 版中的新功能，为支持这些功能所做的系统更改，以及可能会影响现有应用程序的更改。
- 《ASE Replicator 用户指南》— 介绍如何使用 Adaptive Server 的 Adaptive Server Replicator 功能，实现从主服务器到一个或多个远程 Adaptive Server 的基本复制。
- 《组件集成服务用户指南》— 介绍如何使用 Adaptive Server 组件集成服务功能连接远程 Sybase 数据库和非 Sybase 数据库。
- 针对所用平台的《配置指南》— 提供对 Adaptive Server 执行特定配置任务的说明。
- 《Enhanced Full-Text Search Specialty Data Store 用户指南》— 介绍如何借助 Verity 的全文本搜索功能对 Adaptive Server Enterprise 数据进行搜索。
- 《词汇表》— 定义 Adaptive Server 文档中使用的技术术语。
- 《Historical Server 用户指南》— 介绍如何使用 Historical Server 获取 SQL Server<sup>®</sup> 和 Adaptive Server 的性能信息。
- 《Adaptive Server Enterprise 中的 Java》— 介绍在 Adaptive Server 数据库中如何安装 Java 类，如何将它们用作数据类型、函数及存储过程。

- 《Job Scheduler 用户指南》— 提供有关如何使用命令行或图形用户界面 (GUI) 在本地或远程 Adaptive Server 上进行安装和配置以及创建和调度作业的操作说明。
- Messaging Service User's Guide (《消息传送服务用户指南》)— 介绍如何使用实时消息传送服务, 将 TIBCO Java Message Service 和 IBM WebSphere MQ 消息传送服务与所有 Adaptive Server 数据库应用程序集成在一起。
- 《Monitor Client Library 程序员指南》— 介绍如何编写访问 Adaptive Server 性能数据的 Monitor Client Library 应用程序。
- 《Monitor Server 用户指南》— 介绍如何使用 Monitor Server 从 SQL Server 和 Adaptive Server 获取性能统计信息。
- Performance and Tuning Series (《性能和调优系列》)— 是一系列丛书, 介绍如何调节 Adaptive Server 以获得最佳性能:
  - Basics (《基础知识》)— 包含通晓和研究 Adaptive Server 中的性能问题需具备的基础知识。
  - Locking and Concurrency Control (《锁定和并发控制》)— 介绍了如何在 Adaptive Server 中使用各种锁定方案改进性能, 以及如何选择索引以最大限度地减少并发。
  - Query Processing and Abstract Plans (《查询处理和抽象计划》)— 介绍优化程序如何处理查询以及如何使用抽象计划更改某些优化程序计划。
  - Physical Database Tuning (《物理数据库调优》)— 介绍如何管理物理数据放置、为数据分配的空间及临时数据库。
  - Monitoring Adaptive Server with sp\_sysmon (《使用 sp\_sysmon 监控 Adaptive Server》)— 讨论如何使用 sp\_sysmon 监控 Adaptive Server 的性能。
  - Improving Performance with Statistical Analysis (《利用统计分析改进性能》)— 介绍 Adaptive Server 如何存储和显示统计信息, 及如何使用 set statistics 命令分析服务器统计信息。
  - Using the Monitoring Tables (《使用监控表》)— 介绍如何从 Adaptive Server 的监控表中查询统计信息和诊断信息。
- 《快速参考指南》— 这是一本袖珍手册, 完整地列出了各种命令、函数、系统过程、扩展系统过程、数据类型和实用程序的名称和语法 (该手册在以 PDF 格式阅读时采用正常大小)。

- 
- 《参考手册》— 是一套共计 4 本的系列丛书，详细介绍了下列 Transact-SQL® 信息：
    - 《构件块》— Transact-SQL 数据类型、函数、全局变量、表达式、标识符、通配符和保留字。
    - 《命令》— 各种 Transact-SQL 命令。
    - 《过程》— Transact-SQL 系统过程、目录存储过程、系统扩展存储过程和 dbcc 存储过程。
    - 《表》— Transact-SQL 系统表和 dbcc 表。
  - 《系统管理指南》—
    - 《卷 1》— 介绍系统管理的基础知识，包括对配置参数、资源问题、字符集、排序顺序以及系统问题诊断的说明。该手册的第二部分深入说明了安全性管理。
    - 《卷 2》— 包括管理物理资源、镜像设备、配置内存和数据高速缓存、管理多处理器服务器和用户数据库、装入和卸下数据库、创建和使用段、使用 **reorg** 命令及检查数据库一致性的操作说明和指导。本手册的后半部分介绍如何备份和恢复系统数据库和用户数据库。
  - System Tables Diagram（《系统表框图》）— 以张贴画的形式阐明系统表及其实体关系。大图只提供印刷品；采用 PDF 格式时提供缩略图。
  - 《Transact-SQL 用户指南》— 提供有关 Transact-SQL 这一 Sybase 关系数据库语言增强版的文档资料。本手册可用作数据库管理系统初级用户的教科书。本手册还包括对 pubs2 和 pubs3 示例数据库的说明。
  - Troubleshooting Series（《故障排除系列》）（适用于 15.0 版）—
    - Troubleshooting: Error Messages Advanced Resolutions（《故障排除：错误消息高级解析》）— 包含针对在使用 Sybase® Adaptive Server® Enterprise 时可能遇到的问题的故障排除过程。此处讨论的问题是 Sybase 技术支持部门的员工最常听到的问题。
    - Troubleshooting and Error Messages Guide（《故障排除和错误消息指南》）— 包含有关如何解除最常出现的 Adaptive Server 错误消息的详细说明。其中提供的大部分消息都包含错误号（来自 master..sysmessages 表），但有些错误消息没有错误号，并且仅出现在 Adaptive Server 的错误日志中。

- 《加密列用户指南》— 介绍如何利用 Adaptive Server 配置和使用加密列。
- Using Adaptive Server Distributed Transaction Management Features (《使用 Adaptive Server 分布式事务管理功能》)— 介绍如何在分布式事务处理环境中配置、使用 Adaptive Server DTM 功能以及如何排除其中的故障。
- 《在高可用性系统中使用 Sybase 故障切换》— 提供有关使用 Sybase 的故障切换功能将 Adaptive Server 配置为高可用性系统中的协同服务器的操作说明。
- Unified Agent and Agent Management Console (《Unified Agent 和 Agent Management Console》)— 介绍用于提供管理、监控和控制分布式 Sybase 资源的运行期服务的 Unified Agent。
- 《实用程序指南》— 提供有关在操作系统级别执行的 Adaptive Server 实用程序 (如 isql 和 bcp) 的文档资料。
- 《Web 服务用户指南》— 介绍如何配置、使用 Adaptive Server Web 服务以及如何排除其中的故障。
- 《适用于 CICS、Encina 和 TUXEDO 的 XA 接口集成指南》— 提供有关将 Sybase DTM XA 接口与 X/Open XA 事务管理器配合使用的说明。
- 《Adaptive Server Enterprise 中的 XML 服务》— 介绍 Sybase 本机 XML 处理器和 Sybase 基于 Java 的 XML 支持以及数据库中的 XML, 并提供有关构成 XML 服务的查询和映射函数的文档资料。

## 其它信息来源

使用 Sybase Getting Started CD、SyBooks™ CD 及 Sybase® Product Manuals 网站了解有关您的产品的详细信息:

- Getting Started CD 包含 PDF 格式的发行公告和安装指南, 还可能包含 SyBooks CD 中未收纳的其它文档或更新信息。它随软件一起提供。若要阅读或打印 Getting Started CD 上的文档, 需要使用 Adobe Acrobat Reader, 该软件可以使用 CD 上提供的链接从 Adobe 网站免费下载。
- SyBooks CD 含有产品手册, 它随软件一起提供。基于 Eclipse 的 SyBooks 浏览器使您能够以易于使用的、基于 HTML 的格式阅读手册。

有些文档可能是以 PDF 格式提供的, 您可以通过 SyBooks CD 上的 PDF 目录访问这些文档。若要阅读或打印 PDF 文件, 您需要使用 Adobe Acrobat Reader。

---

有关安装和启动 SyBooks 的说明，请参见 Getting Started CD 上的《SyBooks 安装指南》或 SyBooks CD 上的 *README.txt* 文件。

- Sybase Product Manuals 网站是 SyBooks CD 的联机版本，您可以使用一种标准 Web 浏览器来访问它。除了产品手册之外，还可以找到有关 EBFs/Maintenance（EBF/维护）、Technical Documents（技术文档）、Case Management（案例管理）、Solved Cases（解决的案例）、Newsgroups（新闻组）和 Sybase Developer Network（Sybase 开发员网络）的链接。

若要访问 Sybase Product Manuals 网站，请转到位于 <http://www.sybase.com/support/manuals/> 的“产品手册” (Product Manuals)。

**Web 上的 Sybase 认证**      Sybase 网站上的技术文档不断在更新。

❖ **查找有关产品认证的最新信息**

- 1 将 Web 浏览器定位到位于 <http://www.sybase.com/support/techdocs/> 的“技术文档” (Technical Documents)。
- 2 单击“认证报告” (Certification Report)。
- 3 在“认证报告” (Certification Report) 过滤器中选择相应的产品、平台和时间范围，然后单击“查找” (Go)。
- 4 单击“认证报告” (Certification Report) 标题显示此报告。

❖ **查找有关组件认证的最新信息**

- 1 将 Web 浏览器定位到位于 <http://certification.sybase.com/> 的“可用性和认证报告” (Availability and Certification)。
- 2 在“按基本产品搜索” (Search by Base Product) 下选择产品系列和产品，或在“按平台搜索” (Search by Platform) 下选择平台和产品。
- 3 选择“搜索” (Search) 以显示所选项目的可用性和认证报告。

❖ **创建个性化的 Sybase 网站（包括支持页）视图**

建立 MySybase 配置文件。MySybase 是一项免费服务，用于创建个性化的 Sybase 网页视图。

- 1 将 Web 浏览器定位到位于 <http://www.sybase.com/support/techdocs/> 的“技术文档” (Technical Documents)。
- 2 单击“我的 Sybase” (MySybase) 并创建 MySybase 配置文件。

Sybase EBF 和软件维护

❖ 查找有关 EBF 和软件维护的最新信息

- 1 将 Web 浏览器定位到位于 <http://www.sybase.com/support> 的“Sybase 支持页” (Sybase Support Page)。
- 2 选择 “EBF/ 维护” (EBFs/Maintenance)。如果出现提示信息，请输入您的 MySybase 用户名和口令。
- 3 选择一个产品。
- 4 指定时间范围并单击 “查找” (Go)。即会显示 EBF/ 维护版本的列表。

锁形图标表示因为您没有注册为 “技术支持联系人” (Technical Support Contact)，因此您没有某些 EBF/ 维护版本的下载授权。如果您尚未注册，但拥有 Sybase 代表提供的或通过支持合同获得的有效信息，请单击 “编辑角色” (Edit Roles) 将 “技术支持联系人” (Technical Support Contact) 角色添加到 MySybase 配置文件中。

- 5 单击信息图标可显示 EBF/ 维护报告，单击产品说明可下载软件。

约定

以下各部分将说明在本手册中使用的约定。

SQL 是一种形式自由的语言。没有规定每一行中的单词数量或者必须折行的地方。然而，为便于阅读，本手册中所有示例和大多数语法语句都经过了格式设置，以便语句的每个子句都在一个新行上开始。有多个成分的子句会扩展到其它行，这些行会有缩进。复杂命令使用已修改的 Backus Naur Form (BNF) 表示法进行了格式处理。

表 1 说明本手册中出现的语法语句的约定：

表 1：本手册的字体和语法定义

元素	示例
命令名、过程名、实用程序名和其它关键字用 sans serif 字体显示。	<code>select</code> <code>sp_configure</code>
数据库名和数据库类型用 sans serif 字体显示。	<code>master</code> 数据库
书名采用正常字体并加书名号；文件名、变量和路径名用斜体显示。	《系统管理指南》 <code>sql.ini</code> 文件 <code>column_name</code> <code>\$SYBASE/ASE</code> 目录
变量（即代表您要填充的值的词语）作为查询或语句的一部分出现时用斜体的 Courier 字体显示。	<code>select column_name</code> <code>from table_name</code> <code>where search_conditions</code>
将小括号作为命令的一部分键入。	<code>compute row_aggregate (column_name)</code>

元素	示例
双冒号加等号表示语法是用 BNF 表示法编写的。请勿输入此符号。表示“被定义为”。	::=
大括号表示必须至少选择括号中的一个选项。不要输入大括号。	{cash, check, credit}
中括号表示可以选择括号中的一个或多个选项，也可不选。不要输入中括号。	[cash   check   credit]
逗号表示可以选择任意多个显示的选项。可用逗号作为命令的一部分来分隔选项。	cash, check, credit
竖线 ( ) 表示只可选择所显示的选项中的一个。	cash   check   credit
省略号 (...) 表示可以将最后一个单元重复任意次。	buy thing = price [cash   check   credit] [, thing = price [cash   check   credit]]... 您必须至少购买一件产品，并给出其价格。可以选择一种付款方式：中括号中的选项之一。还可选择购买其它物品：可根据需要购买任意数量的物品。对于要买的每种产品，给出其名称、价格和付款方式（可选）。

- 语法语句（显示命令的语法和所有选项）显示如下：

`sp_dropdevice [device_name]`

对于具有多个选项的命令：

`select column_name  
from table_name  
where search_conditions`

在语法语句中，关键字（命令）采用常规字体，而标识符为小写。斜体表示用户提供的内容。

- 说明 Transact-SQL 命令用法的显示如下：

`select * from publishers`

- 计算机输出的示例显示如下：

pub_id	pub_name	city	state
0736	New Age Books	Boston	MA
0877	Binnet & Hardley	Washington	DC
1389	Algodata Infosystems	Berkeley	CA

(3 rows affected)

本手册中的大多数示例都用小写显示。不过，输入 Transact-SQL 关键字时可以忽略大小写。例如，SELECT、Select 和 select 之间没有区别。



Adaptive Server 是否区分数据库对象（如表名）的大小写，取决于安装在 Adaptive Server 上的排序顺序。通过重新配置 Adaptive Server 的排序顺序，可改变单字节字符集的区别大小写设置。有关详细信息，请参见《系统管理指南》。

## 辅助功能特性

本文档具有为提供辅助功能而进行了专门设计的 HTML 版本。可以利用适应性技术（如屏幕阅读器）浏览 HTML 文档，也可以用屏幕放大器进行查看。

Adaptive Server HTML 文档已经过测试，符合美国政府“第 508 节辅助功能”的要求。符合“第 508 节”的文档一般也符合美国之外的辅助功能原则，如 World Wide Web 协会 (W3C) 针对网站的原则。

---

**注释** 可能需要配置辅助功能工具，以便获得最佳使用效果。某些屏幕阅读器按照大小写来辨别文本，例如将“ALL UPPERCASE TEXT”看作首字母缩写，而将“MixedCase Text”看作单词。对工具进行配置，规定语法约定，您可能会感觉更方便。有关工具的信息，请查阅文档。

---

有关 Sybase 如何支持辅助功能的信息，请参见位于 <http://www.sybase.com/accessibility> 的“Sybase 辅助功能” (Sybase Accessibility)。“Sybase 辅助功能” (Sybase Accessibility) 站点包括指向“第 508 节”和 W3C 标准相关信息的链接。

## 如果需要帮助

对于购买了支持合同的客户安装的每一个 Sybase 产品，都会有一位或多位指定人员获得与 Sybase 技术支持部门联系的授权。如果使用手册或联机帮助不能解决问题，可让指定人员与 Sybase 技术支持部门联系或与所在区域的 Sybase 子公司联系。

---

# XML 服务简介

本章介绍 Adaptive Server Enterprise 的 XML 服务功能。

主题	页码
<a href="#">XML capabilities</a>	<a href="#">1</a>
<a href="#">概述</a>	<a href="#">2</a>

## XML capabilities

XML 服务提供以下功能：

- 生成 XML：select 命令中的 for xml 子句，以标准 SQLX 格式将结果集作为 XML 文档返回。
- 存储 XML：
  - 支持将 XML 文档作为字符数据存储在 char、varchar、text、unichar、univarchar 或 unitext 列中，或存储为经过分析的 XML。
  - xmlparse，分析和索引 XML 文档并生成已分析的和已为其创建索引的存储表示形式。
  - xmlvalidate，根据 DTD 或 XML 模式定义验证 XML 文档。
- 查询和提取 XML 数据：xmltest 和 xmlextract，用于查询和提取 XML 文档中的数据。
- I18N 支持：支持 XML 文档中的 Unicode 和非 ASCII 服务器字符集，包括支持生成、存储、查询和提取包含非 ASCII 数据的 XML 文档。

## 概述

与 HTML（超文本标记语言）相似，XML 也是一种标记语言，是 SGML（标准化通用标记语言）的一个子集。但 XML 更为完整和严谨，它允许您自行定义面向应用的标记。这些属性使 XML 尤其适用于数据交换。

您可以从存储在 Adaptive Server 中的数据生成 XML 格式的文档，也可以将 XML 文档以及从其中抽取的数据存储在 Adaptive Server 中。还可以使用 Adaptive Server 来搜索存储在 Web 上的 XML 文档。

XML 是一种标记语言，是 SGML 的子集，它为 Web 出版和分布式文档处理提供了比 HTML 更为强大的功能。

## 数据库中的 XML

- XML 文档具有严格的短语结构，这使数据查找和访问变得非常容易。例如，所有元素必须既有开始标记又有相应的结束标记：  
`<p>` 一个段落。 `</p>`。
- XML 使您可以开发和使用标记来区分不同类型的数据，如客户编号或项目编号。
- 使用 XML 可以创建特定于应用程序的文档类型，这样便可以将一种文档类型和另一种文档类型区分开来。
- XML 文档允许 XML 数据以不同形式显示。与 HTML 文档一样，XML 文档只包含标记和内容；它们不包含格式指令。格式指令通常在客户端提供。

XML 比 SGML 简单，但比 HTML 复杂和灵活。尽管通常可以使用相同的浏览器和处理器来读取 XML 和 HTML，但是 XML 的某些特性使其在文档共享方面比 HTML 更有效。

在 Adaptive Server 中可将 XML 文档存储为：

- 数据类型为 `char`、`varchar`、`unichar`、`univarchar`、`text`、`unitext`、`java.lang.String` 或 `image` 的列中的字符数据。
- `image` 列中的已分析的 XML

## XML 示例文档

此 Order 文档示例是为采购订单应用程序而设计的。客户提交由日期和客户 ID 来标识的订单。每个订单项都包括项目 ID、项目名称、数量和单位名称。

订单项可能在屏幕上显示如下：

ORDER

Date: July 4, 2003

Customer ID: 123

Customer Name: Acme Alpha

Items:

Item ID	Item Name	Quantity
987	Coupler	5
654	Connector	3 dozen
579	Clasp	1

下面是该数据的一种 XML 表示形式：

```
<?xml version="1.0"?>
  <Order>
    <Date>2003/07/04</Date>
    <CustomerId>123</CustomerId>
    <CustomerName>Acme Alpha</CustomerName>
    <Item>
      <ItemId> 987</ItemId>
      <ItemName>Coupler</ItemName>
      <Quantity>5</Quantity>
    </Item>
    <Item>
      <ItemId>654</ItemId>
      <ItemName>Connector</ItemName>
      <Quantity unit="12">3</Quantity>
    </Item>
    <Item>
      <ItemId>579</ItemId>
      <ItemName>Clasp</ItemName>
      <Quantity>1</Quantity>
    </Item>
  </Order>
```

XML 文档有两个独特的特性：

- XML 文档不会为指定项显示而指示类型、样式或颜色。
- 标记是严格嵌套的。每个开始标记 (`<tag>`) 都有一个相应的结束标记 (`</tag>`)。

订单数据的 XML 文档包含四个主要元素：

- XML 声明 `<?xml version="1.0"?>`，它将 “Order” 标识为一个 XML 文档。

每个文档的 XML 声明都显式或隐式地指定字符编码（字符集）。XML 将文档表示为字符数据。要显式地指定字符集，应将其包括在 XML 声明中。例如：

```
<?xml version="1.0" encoding="ISO-8859-1">
```

如果未在 XML 声明中包括字符集，Adaptive Server 中的 XML 将使用缺省字符集 UTF8。

---

**注释** 如果客户端和服务器的缺省字符集不同，Adaptive Server 将绕过常规字符集转换。声明的字符集仍与实际的字符集匹配。请参见第 105 页的“[对 I18N 提供的 XML 支持](#)”。

---

- 用户创建的元素标记，例如 `<Order>...</Order>`、`<CustomerId>...</CustomerId>`、`<Item>...</Item>`。
- 文本数据，例如 “Acme Alpha”、“Coupler” 和 “579”。
- 嵌入到元素标记中的属性，如 `<Quantity unit = “12”>`。这种嵌入允许您自定义元素。

如果文档中包含这些组成部分，而且元素标记严格嵌套，则该文档就被称为**组织良好的 XML 文档**。在以上示例中，元素标记描述了所含的数据，并且文档中不含有格式指令。

下面是另一个 XML 文档示例：

```
<?xml version="1.0"?>
<Info>
  <OneTag>1999/07/04</OneTag>
  <AnotherTag>123</AnotherTag>
  <LastTag>Acme Alpha</LastTag>
</Info>
<Thing>
  <ThingId> 987</ThingId>
  <ThingName>Coupler</ThingName>
  <Amount>5</Amount>
</Thing/>
```

```

<Thing>
  <ThingId>654</ThingId>
  <ThingName>Connector</ThingName>
</Thing>
  <Thing>
    <ThingId>579</ThingId>
    <ThingName>Clasp</ThingName>
    <Amount>1</Amount>
  </Thing>
</Info>

```

此示例名为 “Info”，它也是一个组织良好的 XML 文档，并且与 XML 文档 Order 具有相同的结构和数据。但是，由于 Info 所使用的文档类型定义 (DTD) 与 Order 文档的不同，因此为 Order 文档设计的处理器将无法识别 Info 文档。有关 DTD 的详细信息，请参见第 7 页的“XML 文档类型”。

## Order 数据的 HTML 显示

假设有一个采购订单应用程序。客户提交订单，订单由 Date 和 CustomerID 来标识，并列出一个或多个订单项，每个订单项都有 ItemID、ItemName、Quantity 和 units。

Order 的数据可能在屏幕上显示为如下内容：

ORDER

Date: July 4, 1999

Customer ID: 123

Customer Name: Acme Alpha

Items:

Item ID	Item Name	Quantity
987	Coupler	5
654	Connector	3 dozen
579	Clasp	1

以上数据表示，一个名叫 Acme Alpha、客户 ID 为 123 的客户在 1999 年 7 月 4 日提交了一个购买 coupler、connector 和 clasp 的订单。

以上显示的 Order 数据的 HTML 文本如下所示：

```

<html>
<body>
<p>ORDER

```

[illegible]

这种 HTML 文本有一定的限制:

- 它同时包含数据和格式说明。
  - 数据是客户 ID 和各个客户名称、项目名称和数量。
  - 格式说明表示类型样式 (`<b>....</b>`)、颜色 (`bcolor=white`) 和布局 (`<table>....</table>`)，以及补充字段名如 *Customer Name* 等。
- HTML 文档的结构不是很适合于抽取数据。

某些元素（如表）要求打开标记和关闭标记严格对称，而其它元素（如段落标记“<p>”）则有可选的关闭标记。

某些元素（如段落标记“<p>”）用于多种数据；因此，如果不根据周围的字段名推断上下文，就很难将作为客户 ID 的 123 和作为项目 ID 的 123 区分开。

这种数据和格式的合并，以及缺少严格的短语结构，使 HTML 文档很难适应于不同的表示风格，而且也很难使用 HTML 文档进行数据交换和存储。XML 与 HTML 类似，但是它包括解决这些缺点的限定和扩展。



## XML 文档类型

**文档类型定义 (DTD)** 定义了 XML 文档的类结构，从而可以区分不同的类。一个 DTD 就是某个类特有的元素及属性定义的列表。一旦已经设置了一个 DTD，就可以在另一个文档中引用该 DTD，或者把它嵌入到当前的 XML 文档中。

第 3 页 “XML 示例文档” 中所述的用于 XML 文档 Order 的 DTD 如下所示：

```
<!ELEMENT Order (Date, CustomerId, CustomerName, Item+)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT CustomerId (#PCDATA)>
<!ELEMENT CustomerName (#PCDATA)>
<!ELEMENT Item (ItemId, ItemName, Quantity)>
<!ELEMENT ItemId (#PCDATA)>
<!ELEMENT ItemName (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ATTLIST Quantity units CDATA #IMPLIED>
```

此 DTD 一行一行地指定了以下信息：

- 一个订单必须包含日期、客户 ID、客户名称以及一个或多个订单项。加号 “+” 表示一个或多个订单项。用加号标记的项是必需的。在同一位置处的问号表示一个可选项。元素中的星号表示元素可以出现多次，也可以不出现。（例如，如果上面第一行中的词 “Item\*” 已经加上了星号，则在订单中可以没有此项，也可以有任何数量的此项。）
- 由 “(#PCDATA)” 定义的元素是字符文本。
- 最后一行中的 “<ATTLIST...>” 定义指定，quantity 元素具有 “units” 属性；最后一行末尾的 “#IMPLIED” 表示 “units” 属性是可选的。

XML 文档的字符文本不受约束。例如，由于无法指定 quantity 元素的文本应该为数值类型，因此下面显示的数据是有效的：

```
<Quantity unit="Baker's dozen">three</Quantity>
<Quantity unit="six packs">plenty</Quantity>
```

对元素文本的限制必须通过处理 XML 数据的应用程序来处理。

XML 的 DTD 必须遵循 <?xml version="1.0"?> 指令。您可以将 DTD 包括在 XML 文档中，也可以引用外部 DTD。

- 若要从外部引用 DTD，请使用类似如下代码的代码：

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "Order.dtd">
<Order>
...
</Order>
```

- 嵌入的 DTD 可能如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE Order [
<ELEMENT Order (Date, CustomerId, CustomerName,
Item+)>
<ELEMENT Date (#PCDATA)
<ELEMENT CustomerId (#PCDATA)>
<ELEMENT CustomerName (#PCDATA)>
<ELEMENT Item (ItemId, ItemName, Quantity)>
<ELEMENT ItemId (#PCDATA)>
<ELEMENT ItemName (#PCDATA)>
<ELEMENT Quantity (#PCDATA)>
<!ATTLIST Quantity units CDATA #IMPLIED>
]>
<Order>
  <Date>1999/07/04</Date>
  <CustomerId>123</CustomerId>
  <CustomerName>Acme Alpha</CustomerName>
  <Item>
    ...
  </Item>
</Order>
```

DTD 并不是 XML 文档所必需的。但是，**有效的 XML 文档**应该有一个 DTD 并且符合该 DTD。

本章详细介绍 XML 查询函数，并描述 *option\_string* 参数的一般格式。

主题	页码
<a href="#">XML 查询函数</a>	9
<a href="#">xmlextract</a>	10
<a href="#">xmltest</a>	17
<a href="#">xmlparse</a>	22
<a href="#">xmlrepresentation</a>	25
<a href="#">xmlvalidate</a>	27
<a href="#">option_strings: 一般格式</a>	36

XML 查询函数

本节介绍用于在 SQL 语句中访问和处理 XML 文档的 SQL 扩展。  
xmlextract、xmlparse 和 xmltest 是 XML 服务引入的新保留字。

这些函数是：

表 2-1: XML 查询函数

函数	说明
xmlextract	内置函数，它将 XML 查询表达式应用于 XML 文档并返回选择的结果。
xmltest	SQL 谓词，它将 XML 查询表达式应用于 XML 文档并返回布尔结果。
xmlparse	内置函数，它对 XML 文档进行分析和索引，以提高处理效率。
xmlrepresentation	内置函数，它确定给定的图像列是否包含经过分析的 XML 文档。
xmlvalidate	内置函数，它对 XML 文档进行 DTD 或 XML 模式验证。

## 示例部分

对这些函数的说明包括引用[附录 A “sample\\_docs 示例表”](#) 的示例，此附录包含了用于创建和填充该表的脚本。

## xmlextract

内置函数，它将 *XML\_query\_expression* 应用于 *xml\_data\_expression* 并返回结果。此函数类似于 SQL substring 操作。

### 语法

```
xmlextract_expression ::=  
    xmlextract (xml_query_expression,xml_data_expression  
        [optional_parameters])  
xml_query_expression ::=basic_string_expression  
xml_data_expression ::= general_string_expression  
optional_parameters ::=  
    options_parameter  
    |returns_type  
    | options_parameter returns_type  
options_parameter ::= [,] option option_string  
returns_type ::= [,] returns datatype  
datatype ::= {string_type | computational_type | date_time_type }  
string_type ::= char (integer) | varchar (integer)  
    | unichar (integer) | univarchar (integer)  
    | text | unitext | image  
computational_type ::= integer_type | decimal_type | real_type  
    | date_time_type  
integer_type ::= [ unsigned ] {integer | int | tinyint | smallint | bigint}  
decimal_type ::= {decimal | dec | numeric } [ (integer [, integer ] ) ]  
real_type ::= real | float | double precision  
date_time_type ::= date | time | datetime  
option_string ::= [,] basic_string_expression
```

说明

**注释** 有关 I18N 数据的信息，请参见第 105 页的“对 I18N 提供的 XML 支持”。

- *basic\_string\_expression* 是数据类型为 `character`、`varchar`、`unichar`、`univarchar` 或 `java.lang.String` 的 *sql\_query\_expression*。
- *general\_string\_expression* 是数据类型为 `text`、`image`、`character`、`varchar`、`unitext`、`unichar`、`univarchar` 或 `java.lang.String` 的 *sql\_query\_expression*。
- *xmlextract* 表达式可以在 SQL 语言中任何允许字符表达式的地方使用。
- *options\_parameter* 的缺省值是一个空字符串。空值选项参数将作为空字符串处理。
- 如果 *xml\_query\_expression* 或 *xmlextract()* 的文档参数的值为空值，则 *xmlextract()* 的结果为空值。
- *xml\_data\_expression* 参数的值是用于执行 XML 查询表达式的运行期环境。
- *xmlextract()* 的数据类型由 *returns\_type* 指定。
- *returns\_type* 的缺省值为 `text`。
- 如果 *returns\_type* 指定无整数的 `varchar`，则缺省值为 255。
- 如果 *returns\_type* 指定无精度（第一个整数）的 `numeric` 或 `decimal`，则缺省值为 18。如果指定无标度（第二个整数）的 `numeric` 或 `decimal`，则缺省值为 0。
- 如果查询或文档参数为空，则 *xmlextract* 返回空值。
- 如果 XPath 查询无效，则 *xmlextract* 将引发一个异常。
- *xmlextract* 的初始结果是将 *xml\_query\_expression* 应用于 *xml\_data\_expression* 的结果。该结果按照 XPath 标准指定。
- 如果 *returns\_type* 指定 *string\_type*，则初始结果值返回为该数据类型的字符串文档。

- 如果 *returns\_type* 指定 *computational\_type* 或 *datetime\_type* 数据类型，则初始值将转换为该数据类型并被返回。此转换遵循为 *convert* 内置函数指定的规则。

---

**注释** 初始值必须是适合 *convert* 内置函数的值。这要求使用 XML 查询表达式中的 *text()* 引用。请参见下列示例。

---



---

**注释** 有关下列主题的信息，请参见第 3 章 “XML 语言和 XML 查询语言”：

- 对外部 URI 引用、XML 命名空间和 XML 模式的限制。
  - 预定义的实体及其对应字符的处理：& ( & )、&lt; ( < )、&gt; ( > )、&quot; ( “ ” ) 和 &apos; ( ‘ ’ )。请注意，应将分号作为实体的一部分包含在内。
  - 空白的处理。
  - 空元素的处理。
- 

## option\_string

*option\_string* 的一般格式在第 36 页的 “[option\\_strings: 一般格式](#)” 中说明。

xmlextract 函数所支持的选项为：

```
xmlerror = {exception | null | message}
ncr = {no | non_ascii | non_server}
```

有关 ncr 选项的说明，包括其缺省值，请参见第 105 页的 “[对 I18N 提供的 XML 支持](#)”。

## 异常

如果 *xml\_data\_expression* 的值不是有效的 XML，或者是全空白或空的字符串：

- 如果显式或缺省选项指定 *xmlerror=exception*，则将引发异常。
- 如果显式或缺省选项指定 *xmlerror=null*，则将返回空值。
- 如果显式或缺省选项指定 *xmlerror=message*，则将返回一个字符串，其中包含一个内含异常消息的 XML 元素。该值是有效的 XML。
- 全局变量 @@error 返回最近一次错误的错误号，而不考虑 *xmlerror* 的值是 *exception*、*null*，还是 *message*。

如果 *xmlextract\_expression* 的 *returns\_type* 为 *string\_type* 并且 *xml\_query\_expression* 参数的运行期求值结果大于此类型的最大长度，则将引发异常。

## 示例

下列示例使用附录 A “sample\_docs 示例表” 中介绍的 *sample\_docs* 表。

此示例选择满足以下条件的文档的标题：文档的 `bookstore/book/price` 为 55 或文档的 `bookstore/book/author/degree` 的 `from` 属性为 “Harvard”。

```
select xmlextract('/bookstore/book[price=55
  | author/degree/@from="Harvard" ]/title'
  text_doc )
from sample_docs
-----
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>

NULL

NULL
```

下例选择符合以下条件的文档的 `row/pub_id` 元素：文档的行元素包含小于 10 的 `price` 元素或等于 “Boston” 的 `city` 元素。

此查询返回三行：

- 来自 `bookstore` 行的空值
- 来自 `publishers` 行的单个 “<row>...</row>” 元素
- 来自 `titles` 行的 4 个 “<row>...</row>” 元素

```
select xmlextract('//row[price<10 | city="Boston" ]/pub_id',
  text_doc) from sample_docs2>
-----
NULL
XML 服务 <pub_id>0736</pub_id>

<pub_id>0736</pub_id>
<pub_id>0877</pub_id>
<pub_id>0736</pub_id>
<pub_id>0736</pub_id>

(3 rows affected)
```

下例以整数形式选择 “Seven Years in Trenton” 价格。此查询包含多个步骤。

- 1 以 XML 元素的形式选择 “Seven Years in Trenton” 的价格：

```
select xmlextract
  ('/bookstore/book[title="Seven Years in Trenton"]/price',text_doc)
from sample_docs
where name_doc='bookstore'
-----
<price>12</price>
```

- 2 以下语句尝试通过添加 `returns integer` 子句以 `integer` 形式选择全价：

```
select xmlextract
  ('/bookstore/book[title="Seven Years in Trenton"]/price',
   text_doc returns integer)
from sample_docs
where name_doc='bookstore'
Msg 249, Level 16, State 1:
Line 1:
Syntax error during explicit conversion of VARCHAR value
'<price>12</price>' to an INT field.
```

- 3 为了指定含 `numeric`、`money` 或 `date-time` 数据类型的 `returns` 子句，XML 查询必须返回适合转换为指定数据类型的值。因此，该查询必须使用 `text()` 引用来删除 XML 标记：

```
select xmlextract
  ('/bookstore/book[title="Seven Years in Trenton"]/price/text()',
   text_doc returns integer)
from sample_docs
where name_doc='bookstore'
-----
12
```

- 4 为了指定含 `numeric`、`money` 或 `date-time` 数据类型的 `returns` 子句，XML 查询还必须返回单个值，而不是列表。例如，下面的查询返回的是价格列表：

```
select xmlextract
  ('/bookstore/book/price',
   text_doc)
from sample_docs
where name_doc='bookstore'
-----
<price>12</price>
<price>55</price>
<price intl="canada" exchange="0.7">6.50</price>
```



5 添加 `text()` 引用会产生如下结果:

```
select xmlextract
  ('/bookstore/book/price/text()',
   text_doc)
from sample_docs
where name_doc='bookstore'
-----
12556.50
```

6 指定 `returns integer` 子句将产生异常, 指示组合值不适合转换为整数:

```
select xmlextract
  ('/bookstore/book/price/text()',
   text_doc returns integer)
from sample_docs
where name_doc='bookstore'
Msg 249, Level 16, State 1:
Line 1:
Syntax error during explicit conversion of VARCHAR
value '12556.50' to an INT field.
```

为了说明 `xmlerror` 选项, 下面的命令将一个无效的文档插入到 `sample_docs` 表中:

```
insert into sample_docs (name_doc, text_doc)
values ('invalid doc', '<a>unclosed element<a>')

(1 row affected)
```

在下例中, `xmlerror` 选项决定 `xmlextract` 函数如何处理无效的 XML 文档:

- 如果 `xmlerror=exception` (这是缺省值), 则将引发异常:

```
select xmlextract('/row', text_doc
  option 'xmlerror=exception')
from sample_docs

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
  <<The input ended before all started tags
were ended.Last tag started was 'a'>>
  at line 1, offset 23.
```

- 如果 `xmlerror=null`，则返回一个空值：

```
select xmlextract('//row', test_doc
  option 'xmlerror=null')
from sample_docs

(0 rows affected)
```

- 如果 `xmlerror=message`，则将返回一个含错误消息并经过分析的 XML 文档：

```
select xmlextract('//row', test_doc
  option 'xmlerror=message')
from sample_docs
-----

<xml_parse_error>The input ended before all
startedtags were ended.Last tag started was
'a'</xml_parse_error>
```

`xmlerror` 选项不适用于经过分析的 XML 文档，也不适用于 `xmlparse` 执行的显式嵌套调用所返回的文档。

例如，在以下 `xmlextract` 调用中，`xml_data_expression` 是未分析的字符串文档，因此 `xmlerror` 选项对它适用。该文档是无效的 XML，因此将引发异常，`xmlerror` 选项指示异常消息应该以含以下异常消息的 XML 文档形式返回：

```
select xmlextract('/', '<a>A<a>' option'xmlerror=message')
-----
<xml_parse_error>The input ended before all started tags were ended.
  Last tag started was 'a'</xml_parse_error>
```

在下面的 `xmlextract` 调用中，`xml_data_expression` 由 `xmlparse` 函数（请参见第 22 页的“[xmlparse](#)”一节）执行的显式调用返回。因此，显式 `xmlparse` 调用的缺省 `xmlerror` 选项适用，而外部 `xmlextract` 调用的 `xmlerror` 选项则不适用。该缺省 `xmlerror` 选项是 `exception`，因此显式 `xmlparse` 调用会引发异常：

```
select xmlextract('/', xmlparse('<a>A<a>')
  option 'xmlerror=message'))
-----
Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
  <<The input ended before all started tags were ended.
  Last tag started was 'a'>> at line 1, offset 8.
```

若要将 `xmlerror=message` 选项应用于 `xmlparse` 的显式嵌套调用，请指定它作为该调用中的一个选项：

```
select xmlextract('/',
    xmlparse('<a>A<a>' option 'xmlerror=message'))
-----
<xml_parse_error>The input ended before all started
tags were ended.Last tag started was
'a'</xml_parse_error>
```

未分析 XML 文档和 `xmlparse` 嵌套调用的 `xmlerror` 选项处理总结：

- 只有当文档操作数是未分析的文档时，`xmlextract` 才使用 `xmlerror` 选项。
- 当文档操作数是显式 `xmlparse` 调用时，该调用的隐式或显式 `xmlerror` 选项将覆盖 `xmlextract` 的隐式或显式 `xmlerror` 选项。

下面的命令将 `sample_docs` 表恢复到其初始状态：

```
delete from sample_docs
where na_doc='invalid doc'
```

## xmltest

对 XML 查询表达式求值的谓词，它可以引用 XML 文档参数，并将返回布尔结果。类似于 SQL like 谓词。

语法

```
xmltest_predicate ::=
    xml_query_expression [not] xmltest xml_data
    [option option_string]
xml_data ::=
    xml_data_expression | (xml_data_expression)
xml_query_expression ::= basic_string_expression
xml_data_expression ::= general_string_expression
option_string ::= basic_string_expression
```

## 说明

**注释** 有关处理 I18N 数据的信息，请参见第 6 章 “对 I18N 提供的 XML 支持”。

- *basic\_string\_expression* 是数据类型为 character、varchar、unichar、univarchar 或 java.lang.String 的 *sql\_query\_expression*。
- *general\_string\_expression* 是数据类型为 character、varchar、unichar、univarchar、text、unitext 或 java.lang.String 的 *sql\_query\_expression*。
- 在 SQL 语言中，凡是允许使用 SQL 谓词的位置，都可以使用 xmltest 谓词。
- 指定以下内容的 xmltest 调用：  

```
X not xmltest Y options Z
```

  
等效于：  

```
not X xmltest Y options Z
```
- 如果 xmltest() 的 *xml\_query\_expression* 或 *xml\_data\_expression* 为空值，xmltest() 的结果就是未知的。
- 如果 *xml\_query\_expression* 或 xmlextract() 的文档参数的值为空值，则 xmlextract() 的结果为空值。
- *xml\_data\_expression* 参数的值是用于执行 XPath 表达式的运行期环境。
- xmltest() 的求值结果为布尔值 true 或 false，具体如下：
  - 如果 xmltest() 的 *xml\_query\_expression* 是结果为 empty (not empty) 的 XPath 表达式，则 xmltest() 将返回 false (true)。
  - 如果 xmltest() 的 *xml\_query\_expression* 是结果为布尔值 false (true) 的 XPath 表达式，则 xmltest() 将返回 false (true)。
  - 如果 XPath 表达式无效，则 xmltest 将引发一个异常。

**注释** 有关下列主题的信息，请参见第 3 章 “XML 语言和 XML 查询语言”：

- 对外部 URI 引用、XML 命名空间和 XML 模式的限制。
- 预定义的实体及其对应字符的处理：&amp; (&)、&lt; (<)、&gt; (>)、&quot; (“) 和 &apos; (')。请注意，应将分号作为实体的一部分包含在内。
- 空白的处理。
- 空元素的处理。

**选项** option\_string 的一般格式在第 36 页的 “option\_strings: 一般格式” 中说明。

xmltest 谓词所支持的选项为 xmlerror = {exception | null}。

xmlextract 和 xmlparse 所支持的 message 替代选项对于 xmltest 无效。请参见 “异常” 一节。

**异常** 如果 xml\_data\_expression 的值不是有效的 XML，或者是全空白或空的字符串：

- 如果显式或缺省选项指定 xmlerror=exception，则将引发异常。
- 如果显式或缺省选项指定 xmlerror=null，则将返回空值。
- 如果您指定 xmlerror=message，则返回空值。

**示例** 以下示例使用附录 A “sample\_docs 示例表” 中介绍的 sample\_docs 表。

下面的示例选择满足以下条件的每个行的 name\_doc: 行的 text\_doc 包含等于 “Boston” 的 row/city。

```
select name_doc from sample_docs
where '//'row[city="Boston"]' xmltest text_doc
      name_doc
-----
publishers

(1 row affected)
```

在下例中，对于布尔型的 *false/true* 结果以及 *empty/not-empty* 结果，*xmltest* 谓词返回 *false/true*。

```
-- 布尔值 true 为 “true”:
select case when '/a="A"' xmltest '<a>A</a>'
           then 'true' else 'false' end2>
-----
true

-- 布尔值 false 为 “false”
select case when '/a="B"' xmltest '<a>A</a>'
           then 'true' else 'false' end
-----
false

-- 非空结果为 “true”
select case when '/a' xmltest '<a>A</a>'
           then 'true' else 'false' end

----- true
-- 空结果为 “false”
select case when '/b' xmltest '<a>A</a>'
           then 'true' else 'false' end
-----
false

-- 空结果为 “false”（第二个示例）
select case when '/b="A"' xmltest '<a>A</a>'
           then 'true' else 'false' end
-----
false
```

为了说明 *xmlerror* 选项，下面的命令将一个无效的文档插入到 *sample\_docs* 表中：

```
insert into sample_docs (name_doc, text_doc)
values ('invalid doc', '<a>unclosed element<a>')

(1 row affected)
```

在以下示例中，`xmlerror` 选项决定 `xmltest` 谓词如何处理无效的 XML 文档。

- 如果 `xmlerror=exception`（缺省结果），则引发一个异常，且全局变量 `@@error` 包含错误消息 14702。

```
select name_doc from sample_docs
where '//@price<10/*' xmltest text_doc
option 'xmlerror=exception'
```

```
Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE():XML parser fatal error
    <<The input ended before all started tags were
ended.Last tag started was 'a'>> at line 1,
offset 23.
```

若要显示 `@@error` 的内容，请输入：

```
select @@error
-----
14702
(1 row affected)
```

- 如果 `xmlerror=null` 或 `xmlerror=message`，则返回一个空值（未知值），且全局变量 `@@error` 包含错误消息 14701。

```
select name_doc from sample_docs
where '//@price<10/*' xmltest text_doc
option 'xmlerror=null'
```

```
(0 rows affected)
```

若要显示 `@@error` 的内容，请输入：

```
select @@error
-----
14701
(1 row affected)
```

下面的命令将 `sample_docs` 表恢复到其初始状态：

```
delete from sample_docs
where name_doc='invalid doc'
```

## xmlparse

内置函数，它分析作为参数传递的 XML 文档并返回包含经过分析的文档的 `image` 值。

### 语法

```
xmlparse_call ::=  
  xmlparse(general_string_expression  
           [options_parameter][returns_type])  
options_parameter ::= [,] option option_string  
option_string ::= basic_string_expression  
returns_type ::= [,] returns {image | binary | varbinary [(integer )]}
```

### 说明

---

**注释** 有关处理 I18N 数据的信息，请参见第 6 章 “对 I18N 提供的 XML 支持”。

---

- 如果省略 `returns` 子句，则缺省为 `returns image`。
- *basic\_string\_expression* 是数据类型为 `character`、`varchar`、`unichar`、`univarchar` 或 `java.lang.String` 的 *sql\_query\_expression*。
- *general\_string\_expression* 是数据类型为 `character`、`varchar`、`unichar`、`univarchar`、`text`、`unitext`、`image` 或 `java.lang.String` 的 *sql\_query\_expression*。
- 如果 `xmlparse()` 的任何参数为空，则该调用的结果为空。
- 如果 *general\_string\_expression* 是全空白字符串，则 `xmlparse` 的结果是空的 XML 文档。
- `xmlparse()` 将 *general\_string\_expression* 作为 XML 文档进行分析，并返回包含已分析文档的 `image` 值。
- 如果 *general\_string\_expression* 为 `image` 表达式，则认为它包含服务器字符集中的字符。

---

**注释** 有关下列主题的信息，请参见第 3 章 “XML 语言和 XML 查询语言”：

- 对外部 URI 引用、XML 命名空间和 XML 模式的限制。
  - 预定义的实体及其对应字符的处理：`&amp;` (`&`)、`&lt;` (`<`)、`&gt;` (`>`)、`&quot;` (`"`) 和 `&apos;` (`'`)。请注意，应将分号作为实体的一部分包含在内。
  - 空白的处理。
  - 空元素的处理。
-



## 选项

- option\_string 的一般格式在第 36 页的 “option\_strings: 一般格式” 中说明。xmlparse 函数所支持的选项为：

`dtdvalidate = {yes | no}`

`xmlerror = {exception | null | message}`

如果指定 `dtdvalidate=yes`，则将对照其嵌入的 DTD（如果有）验证 XML 文档。此选项用于与 Adaptive Server Enterprise 12.5 的基于 Java 的 XQL 处理器兼容。

如果指定 `dtdvalidate=no`，则不执行任何 DTD 验证。这是缺省值。

`xmlerror = {exception | null | message}`

有关 `xmlerror` 选项，请参见下文中的 “异常”。

## 异常

如果 `xml_data_expression` 的值不是有效的 XML：

- 如果显式或缺省选项指定 `xmlerror=exception`，则将引发异常。
- 如果显式或缺省选项指定 `xmlerror=null`，则将返回空值。
- 如果显式或缺省选项指定 `xmlerror=message`，则将返回一个字符串，该字符串包含一个内含异常消息的 XML 元素。该值是有效且经过分析的 XML。
- 全局变量 `@@error` 返回最近一次错误的错误号，而不考虑 `xmlerror` 的值是 `exception`、`null`，还是 `message`。

如果 `xml_data_expression` 的值不是有效的 XML：

- 如果显式或缺省选项指定 `xmlerror=exception`，则将引发异常。
- 如果显式或缺省选项指定 `xmlerror=null`，则将返回空值。
- 如果显式或缺省选项指定 `xmlerror=message`，则将返回一个字符串，它包含一个内含异常消息的 XML 元素。该值是有效且经过分析的 XML。

## 示例

以下示例使用附录 A “sample\_docs 示例表” 中介绍的 `sample_docs` 表

在创建和初始化后，`sample_docs` 表的 `text_doc` 列包含文档，而 `image_doc` 列为空。您可以更新 `image_doc` 列以包含 `text_doc` 列经过分析的 XML 版本：

```
update sample_docs
set image_doc = xmlparse(text_doc)

(3 rows affected)
```

然后可以将 `xmlextract` 函数应用于 `image` 列中经过分析的 XML 文档，具体方式与将其应用于 `text` 列中的未分析 XML 文档相同。对经过分析的 XML 文档的操作通常比对未分析 XML 文档的操作执行得快。

```
select name_doc,
       xmlextract('/bookstore/book[title="History of Trenton"]/price', text_doc)
       as extract_from_text_doc,
       xmlextract('/bookstore/book[title="History of Trenton"]/price', image_doc)
       as extract_from_image_doc
from sample_docs
```

name_doc	extract_from_text_doc	extract_from_image_doc
bookstore	<price>55</price>	<price>55</price>
publishers	NULL	NULL
titles	NULL	NULL

(3 rows affected)

为了说明 `xmlerror` 选项，下面的命令将一个无效的文档插入到 `sample_docs` 表中

```
insert into sample_docs (name_doc, text_doc) ,
values ('invalid doc', '<a>unclosed element<a>')
```

(1 row affected)

在下面的示例中，`xmlerror` 选项决定 `xmlparse` 函数如何处理无效的 XML 文档：

- 如果 `xmlerror=exception`（这是缺省值），则将引发异常：

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=exception')
```

Msg 14702, Level 16, State 0:

Line 2:

XMLPARSE():XML parser fatal error

<<The input ended before all started tags were ended.Last tag started was 'a'>> at line 1, offset 23.

- 如果 `xmlerror=null`，则返回一个空值：

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=null')
```

```
select image_doc from sample_docs
where name_doc='invalid doc'
```

-----  
NULL

- 如果 `xmlerror=message`，则将返回一个内含错误消息的经过分析的 XML 文档：

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=message')

select xmlextract('/', image_doc)
from sample_docs
where name_doc = 'invalid doc'
-----
<xml_parse_error>The input ended before all started tags were ended.Last
tag started was 'a'</xml_parse_error>
```

下面的命令将 `sample_docs` 表恢复到其初始状态：

```
delete from sample_docs
where name_doc='invalid doc'
```

## xmlrepresentation

检查 `image` 参数并返回一个整数，此整数指示该参数是否包含经过分析的 XML 数据或其它类型的 `image` 数据。

语法	<code>xmlrepresentation_call::=</code> <code>xmlrepresentation(<i>parsed_xml_expression</i>)</code>
说明	<ul style="list-style-type: none"> <li>• <i>parsed_xml_expression</i> 是数据类型为 <code>image</code>、<code>binary</code> 或 <code>varbinary</code> 的 <i>sql_query_expression</i>。</li> <li>• 如果 <code>xmlrepresentation()</code> 的参数为空，则该调用的结果为空。</li> <li>• 如果操作数是经过分析的 XML 数据，<code>xmlrepresentation</code> 将返回整数 0；如果操作数不是经过分析的 XML 数据或者是全空白或空的字符串，则将返回正整数。</li> </ul>
示例	以下示例使用附录 A “ <code>sample_docs</code> 示例表”中介绍的 <code>sample_docs</code> 表。

**示例 1** 此示例说明基本的 `xmlrepresentation` 函数。

```
-- 对于不是经过分析的
-- XML 的文档，返回非零值
select xmlrepresentation(
      xmlextract('/', '<a>A</a>' returns image)
-----
```

```
1

-- 对于是经过分析的 XML 的文档，返回零
select xmlrepresentation(
      xmlparse(
        xmlextract('/', '<a>A</a>' returns image))
-----
0
```

**示例 2** 数据类型为 `image` 的列既可以包含经过分析的 XML 文档（由 `xmlparse` 函数生成），也可以包含未分析的 XML 文档。此示例中的 `update` 命令执行后，`sample_docs` 表的 `image_doc` 列为 `titles` 文档包含一个经过分析的 XML 文档，为 `bookstore` 文档包含一个未分析的（字符串）XML 文档，为 `publishers` 文档包含一个空值（初始值）。

```
update sample_docs
set image_doc = xmlextract('/', text_doc returns image)
where name_doc = 'bookstore'
```

```
update sample_docs
set image_doc = xmlparse(text_doc)
where name_doc = 'titles'
```

**示例 3** 可以使用 `xmlrepresentation` 函数确定 `image` 列的值是否是经过分析的 XML 文档：

```
select name_doc, xmlrepresentation(image_doc) from
sample_docs
```

```
name_doc
-----
bookstore      1
publishers     NULL
titles         0
```

(3 rows affected)

**示例 4** 您可以更新 `image` 列并将其所有值设置为经过分析的 XML 文档。如果 `image` 列既包含经过分析的 XML 文档，又包含未分析的 XML 文档，则简单更新将引发异常。

```
update sample_docs set image_doc = xmlparse(image_doc)
Msg 14904, Level 16, State 0:
Line 1:
XMLPARSE: Attempt to parse an already parsed XML
document.
```

**示例 5** 可以使用 `xmlrepresentation` 函数避免引发这种异常：

```
update sample_docs
set image_doc = xmlparse(image_doc)
where xmlrepresentation(image_doc) != 0

(1 row affected)
```

**示例 6** 下面的命令将 `sample_docs` 表恢复至其初始状态。

```
update sample_docs
set image_doc = null
```

## xmlvalidate

验证 XML 文档。

语法

```
xmlvalidate_call ::=
  xmlvalidate ( general_string_expression, [optional_parameters])
optional_parameters ::= options_parameter
  | returns_type
  | options_parameter returns_type
options_parameter ::= [,] option_option_string
options_string ::= basic_string_expression
returns_type ::= [,] returns_string_type
string_type ::= char (integer) | varchar (integer)
  | unichar (integer) | univarchar (integer)
  | text | unitext | image | java.lang.String
```

说明

**注释** 有关验证 Unicode 的信息，请参见第 6 章“对 I18N 提供的 XML 支持”。

- *basic\_string\_expression* 是数据类型为 `character`、`varchar`、`unichar`、`univarchar` 或 `java.lang.String` 的 *sql\_query\_expression*。
- *general\_string\_expression* 是数据类型为 `character`、`varchar`、`unichar`、`univarchar`、`text`、`unitext` 或 `java.lang.String` 的 *sql\_query\_expression*。
- 如果 `xmlvalidate()` 的任何参数为空，则该调用的结果为空。
- `xmlvalidate_call` 的结果数据类型是由 *returns\_type* 指定的数据类型。

## 选项

*option\_string* 的一般格式在[第 36 页](#)的“*option\_strings*: 一般格式”中说明。*xmlvalidate* 所支持的选项为:

```
validation_options ::=
    [dtdvalidate = {no | yes | strict}]
    [schemavalidate = {no | yes}]
    [nonamespaceschemalocation = 'schema_uri_list']
    [schemalocation = 'namespace_schema_uri_list']
    [xmlerror = {exception | null | message}]
    [xmlvalid = {document | message}]
    schema_uri_list ::=
        schema_uri [schema_uri]...
    namespace_schema_uri_list ::=
        namespace_name schema_uri
        [ namespace_name schema_uri]...
    schema_uri ::= character_string
    namespace_name ::= character_string
```

## 选项说明

- *validation\_options* 的缺省值如下:
  - dtdvalidate = 请参见下面的说明
  - schemavalidate = *no*
  - schemalocation = ""
  - nonamespaceschemalocation = ""
  - xmlerror = *exception*
  - xmlvalid = *document*
- *validation\_option* 中的关键字不区分大小写, 但 *schema\_uri\_list* 和 *namespace\_schema\_uri\_list* 区分大小写。
- 请参考您将其作为主题 XML 文档进行分析或存储的文档。
- *dtdvalidate* 的缺省值取决于 *schemavalidate* 选项的隐式值或显式值。如果 *schemavalidate* 选项值为 *no*, 则 *dtdvalidate* 的缺省值为 *no*。如果 *schemavalidate* 选项值为 *yes*, 则缺省的 *dtdvalidate* 选项值为 *strict*。
- 如果指定 *schemavalidate* = *yes*, 则必须指定 *dtdvalidate* = *strict* 或忽略 *dtdvalidate*。
- 如果指定 *dtdvalidate* = *no*, 并指定 *schemavalidate* = *no*, 则只检查文档是否组织良好。
- 如果指定 *schemavalidate* = *no*, 则子句 *nonamespaceschemalocation* 和 *schemalocation* 将被忽略。

- 子句 `nonamespaceschemalocation` 和 `schemalocation` 中指定的值为字符文字。如果 Transact-SQL `quoted_identifier` 选项为 *off*，则可以选择撇号 (') 或引号 (") 括住 *option\_string*，并使用其它符号括住由 `nonamespaceschemalocation` 和 `schemalocation` 指定的值。如果 Transact-SQL `quoted_identifier` 选项为 *on*，则必须用撇号 (') 括住 *option\_string*，而且必须用引号 (") 括住由 `nonamespaceschemalocation` 和 `schemalocation` 指定的值。
- `nonamespaceschemalocation` 指定了模式 URI 列表，该列表将替换主题 XML 文档的 `xsi:noNameSpaceschemalocation` 子句中指定的模式 URI 列表。
- `schemalocation` 指定一个由 `namespace` 名称和模式 URI 构成的对的列表。
  - `namespace` 名称是 `xmlns` 属性为 `namespace` 指定的名称。  
*http://acme.com/schemas.contract* 在下面的示例中被声明为缺省的 `namespace`：

```
<contract xmlns="http://acme.com/schemas.contract">
```

不过，在下面的示例中，它则被声明为前缀 “co” 的 `namespace`：

```
<co:contract xmlns:co="http://acme.com/schemas.contract">
```

`namespace` 名称是 `namespace` 声明本身中指定的 URI，而不是前缀。

- `schema_uri` 是包含模式 URI 的字符串文字。*URI\_string* 的最大长度为 1927 个字符，并且它必须指定 *http*。`schema_uri` 引用的模式必须被编码为 UTF8 或 UTF16。
- `dtdvalidate` 选项值为：
  - `dtdvalidate=no`：不执行任何 DTD 或模式验证；检查文档以确保其组织良好。
  - `dtdvalidate=yes`：根据文档指定的任何 DTD 对文档进行验证。
  - `dtdvalidate=strict`：此选项取决于 `schemavalidate` 选项。
    - `schemavalidate=no`：  
您必须指定主题 XML 文档中的 DTD；验证该文档需要根据此 DTD 进行。
    - `schemavalidate=yes`：  
您必须声明 DTD 或模式的主题 XML 文档中的每个元素；验证各个元素需要根据这些声明进行。

- **schemavalidate** 选项值如下：

如果指定 **schemavalidate=no**，则对主题 XML 文档不执行任何模式验证。

如果指定 **schemavalidate=yes**，将执行模式验证。

- 当 *general\_string\_expression*（例如 *XC*）是传递 *option\_string* 子句中指定的验证选项的 XML 文档时，会出现以下结果：

如果 **xmlvalid** 指定了 **doc**，则 **xmlvalidate** 的结果为：

```
convert(text, XC)
```

如果 **xmlvalid** 指定了消息，则 **xmlvalidate** 的结果为此 XML 文档：

```
<xmlvalid/>
```

- 当 *general\_string\_expression* 不是传递 *option\_string* 子句中指定的验证选项的 XML 文档时，会出现以下结果：

如果 *option\_string* 指定了 **xmlerror=exception**，则将引发带有 *exception* 消息的异常。

如果 *option\_string* 指定了 **xmlerror=message**，则会返回具有以下形式的 XML 文档。E1、E2 等都是描述验证错误的消息。

```
<xml_validation_errors>
  <xml_validation_error>E1</xml_validation_error>
  <xml_validation_error>E2</xml_validation_error>
  ...
  <xml_validation_warning>W1</xml_validation_warning>
  <xml_validation_fatal_error>E3</xml_validation_fatal_error>
</xml_validation_errors>
```

如果 *option\_string* 指定 **xmlerror=null**，则将返回空值。

## 异常

如果 *xml\_data\_expression* 的值不是有效的 XML：

- 如果显式或缺省选项指定 **xmlerror=exception**，则将引发异常。
- 如果显式或缺省选项指定 **xmlerror=null**，则将返回空值。
- 如果显式或缺省选项指定 **xmlerror=message**，则将返回一个字符串，该字符串包含一个内含所有异常消息的 XML 元素。该值是有效且经过分析的 XML。
- 全局变量 **@@error** 返回最近一次错误的错误号，而不考虑 **xmlerror** 的值是 **exception**、**null**，还是 **message**。
- 如果验证所需的 Web 资源不可用，则会发生异常。



- 如果源 XML 文档无效或未良好组织，则会发生异常。其消息会说明验证失败。

示例

表 2-2 中显示的 XML DTD 和模式说明了验证子句。

- *dtd\_emp* 和 *schema\_emp* 定义单个文本元素 “<emp\_name>”
- *dtd\_cust* 和 *schema\_cust* 定义单个文本元素 “<cust\_name>”
- *ns\_schema\_emp* 和 *ns\_schema\_cust* 是指定目标 namespace 的变体。

表 2-2: DTD 和模式的示例以及它们的 URI

URI	文档
http://test/dtd_emp.dtd	<!ELEMENT emp_name (#PCDATA) >
http://test/dtd_cust.dtd	<!ELEMENT cust_name (#PCDATA) >
http://test/schema_emp.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_emp"> <xsd:element name="emp_name" type="xsd:string"/>  </xsd:schema>
http://test/ns_schema_emp.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_emp"> <xsd:element name="emp_name" type="xsd:string"/>  </xsd:schema>
http://test/schema_cust.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema"> <xsd:element name="cust_name" type="xsd:string"/>  </xsd:schema>
http://test/ns_schema_cust.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_cust"> <xsd:element name="cust_name" type="xsd:string"/>  </xsd:schema>

**示例 1** 此示例创建了一个表，将在该表的 `text` 列中存储 XML 文档。使用此表可说明 `xmlvalidate` 的示例调用。换句话说，`xmlvalidate` 将显式验证存储于 `text` 列中的文档。

```
create table text_docs(xml_doc text null)
```

**示例 2** 此示例显示 `xmlvalidate` 指定了一个无 DTD 声明、使用了验证选项 `dtdvalidate=yes` 的文档。由于插入的文档组织良好，并且 `dtdvalidate` 没有被指定为 `strict`，因此该命令执行成功。

```
insert into text_docs
values (xmlvalidate(
  '<employee_name>John Doe</employee_name>',
  option 'dtdvalidate=yes'))
-----
(1 row inserted)
```

**示例 3** 此示例显示 `xmlvalidate` 指定了无 DTD 声明、使用验证选项 `dtdvalidate=strict` 的文档。因为严格的 DTD 验证要求文档中的每个元素都要经 DTD 指定，所以 `xmlvalidate` 将引发异常。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>',
  option 'dtdvalidate=strict'))
-----
EXCEPTION
```

**示例 4** 上一个示例在验证失败后引发了异常。不过，您可以使用选项 `xmlerror` 指定 `xmlvalidate` 在验证失败时应返回空值。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>'
  option 'dtdvalidate=strict xmlerror=null'))
-----
null
```

**示例 5** 您也可以使用 `xmlerror` 指定 `xmlvalidate` 在验证失败时以 XML 文档形式返回 XML 错误消息：

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>'
  option 'dtdvalidate=strict xmlerror=message'))
-----
<xml_validation_errors>
<xml_validation_error>(1:15)Document is invalid:
  no grammar found.</xml_validation_error>
<xml_validation_error>(1:15)Document root element
```

```

        "employee name", must match DOCTYPE root
        "null."</xml_validation_error>
    </xml_validation_errors>

```

**示例 6** 此示例显示 `xmlvalidate` 指定了一个同时引用 DTD 和验证选项 `dtdvalidate=yes` 的文档。此命令可成功执行。

```

insert into text_docs
values(xmlvalidate(
    '<DOCTYPE emp_name PUBLIC "http://test/dtd_emp.dtd">
      <emp_name>John Doe</emp_name>',
    option 'dtdvalidate=yes'))
-----
(1 row inserted)

```

**示例 7** 此示例显示 `xmlvalidate` 指定了一个引用 DTD 和验证选项 `dtdvalidate=yes` 的文档。由于插入的文档与文档中引用的 DTD 不匹配，因此 `xmlvalidate` 引发了异常。

```

insert into text_docs
values(xmlvalidate(
    '<DOCTYPE emp_name PUBLIC "http://test/dtd_cust.dtd">
      <emp_name>John Doe</emp_name>',
    option 'dtdvalidate=yes'))
-----
EXCEPTION

```

**示例 8** 此示例显示 `xmlvalidate` 指定了一个无模式声明、使用了验证选项 `schemavalidate=yes` 的文档。由于 “<emp\_name>” 元素没有声明，所以此命令会失败。

```

insert into text_docs
values(xmlvalidate('<emp_name>John Doe</emp_name>',
    option 'schemavalidate=yes'))
-----
EXCEPTION

```

**示例 9** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 的文档。此文档不使用命名空间。因为此文档与文档中引用的模式相匹配，所以此命令可成功执行。

```

insert into text_docs
values(xmlvalidate(
    '<emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="http://test/schema_emp.xsd">
      John Doe</emp_name>'
    option 'schemavalidate=yes'))
-----
(1 row inserted)

```

**示例 10** 此示例显示 `xmlvalidate` 指定了一个文档，此文档指定了一个命名空间和验证选项 `schemavalidate=yes`。因为此文档与文档中引用的模式相匹配，所以此命令可成功执行。

```
insert into text_docs
values(xmlvalidate(
  '<emp:emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:emp="http://test/ns_schema_emp"
    xsi:SchemaLocation="http://test/ns_schema_emp
      http://test/ns_schema_emp.xsd">
    John Doe</emp:emp_name>'
  option 'schemavalidate=yes'))
-----
(1 row inserted)
```

**示例 11** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 的文档。因为此文档与文档中引用的模式不匹配，所以此命令失败。

```
insert into text_docs
values (xmlvalidate(
  '<emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://test/schema_cust.xsd">
    John Doe</emp_name>'
  option 'schemavalidate=yes'))
-----
EXCEPTION
```

**示例 12** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 的文档。此文档指定了一个命名空间。因为此文档与文档中引用的模式不匹配，所以此命令失败。

```
insert into text_docs
values(xmlvalidate(
  '<emp:emp_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:emp="http://test/ns_schema_cust"
    xsi:schemaLocation=
      "http://test/ns_schema_cust http://test/ns_schema_cust.xsd">
    John Doe</emp:emp_name>',
  option 'schemavalidate=yes'))
-----
EXCEPTION
```

`xmlvalidate` 的验证选项指定的 `nonamespaceschemalocation` 为 `http://test/ns_schema_emp.xsd`。

**示例 13** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 以及子句 `schemalocation` 和 `nonamespaceschemalocation` 的文档。

该文档将 `schemaLocation` 指定为 `http://test/schema_cust.xsd`，而 `xmlvalidate` 中的验证选项将 `schemalocation` 指定为 `http://test/ns_schema_emp.xsd`。

因为此文档与 `xmlvalidate` 中引用的模式（该模式将覆盖文档中引用的模式）相匹配，所以此命令可成功执行。

```
insert into text_docs
values (xmlvalidate(
  '<emp:emp_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:emp="http://test/schema_emp"
    xsi:schemaLocation="HTTP://test/ns_schema_emp
      http://test/schema_cust.xsd">
    John Doe</emp:emp_name>',
  option 'schemavalidate=yes,
    schemalocation= "http://test/ns_schema_emp
      http://test/ns_schema_emp.xsd"
    nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-----
(1 row inserted)
```

**示例 14** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 以及子句 `schemalocation` 和 `nonamespaceschemalocation` 的文档。

该文档指定的 `noNamespaceSchemaLocation` 为 `http://test/schema_cust.xsd`，`xmlvalidate` 中的验证选项指定的 `nonamespaceschemalocation` 为 `http://test/ns_schema_emp.xsd`。

因为此文档与 `xmlvalidate` 中引用的模式不匹配，所以此命令失败。不过此文档确实与文档中引用的模式相匹配。

```
insert into text_docs
values (xmlvalidate(
  '<customer_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://test/schema_cust.xsd">
    John Doe</customer_name>'
  option 'schemavalidate=yes,
    schemalocation="http://test/ns_schema_emp http://test/ns_schema_emp.xsd"
    nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-----
EXCEPTION
```

**示例 15** 此示例显示 `xmlvalidate` 指定了一个使用模式声明和验证选项 `schemavalidate=yes` 以及子句 `schemalocation` 和 `nonamespaceschemalocation` 的文档。

该文档指定的 `schemaLocation` 为 `http://test/schema_cust.xsd`，`xmlvalidate` 的验证选项指定的 `schemalocation` 为 `http://test/ns_schema_emp.xsd`。

因为此文档与 `xmlvalidate` 中引用的模式不匹配，所以此命令失败。不过此文档确实与文档中引用的模式相匹配。

```
insert into text_docs
values(xmlvalidate(
  '<cust:cust_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cust="http://test/schema_cust"
    xsi:schemaLocation="http://test/schema_cust
      http://test/schema_cust.xsd">
    John Doe</cust:cust_name>',
  option 'schemavalidate=yes,
    schemalocation="http://test/ns_schema_emp
http://test/ns_schema_emp.xsd"
    nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-----
(1 row inserted)
```

## option\_strings: 一般格式

本节说明 XML 服务中选项字符串参数的一般格式、语法和处理。各个选项的操作在引用它们的函数中介绍。

任何具有 `option_string` 参数的函数接受所有选项的联合，而忽略任何不适用于该特定函数的选项。

例如，`forxmlj` 没有 `XML document` 参数，但它仍接受包含 `xmlerror` 选项（该选项指定针对无效 XML 操作数的操作）的 `option_string`。

通过此“联合选项”方式，您可以将单个 `option_string` 变量用于所有 XML 服务函数。

- 语法
- option\_string ::= *basic\_string\_expression*
- 说明
- option\_string* 参数的运行期值的完整语法是：

*option\_string\_value* ::= *option* [,] *option* ] ...  
*option* ::= *name* = *value*  
*name* ::= 下面列出的选项名  
*value* ::= *simple\_identifier* | *quoted\_string*
  - 如果 *option\_string* 参数为空值，则空字符串全部为空白。
  - 在第一个 *option* 之前，最后一个 *option* 之后，*option* 之间以及在等号的两侧，可以使用任意数量的空格。
  - 可以使用逗号或空格分隔 *option*。
  - option\_value* 可以是简单标识符（以字母开头，后面是字母、数字和下划线）或加引号的字符串。加引号的字符串按照常规的嵌入引号 SQL 约定形成。
  - [表 2-3](#) 显示了 *options* 的集合以及它们适用的函数。有关选项的说明，请参见特定函数的说明。

查询函数的选项值

**注释** 下划线表示 *options* 的缺省值，这些值指定了此表中的关键字。小括号显示用于指定 SQL 名称的 *options* 缺省值。空字符串或单空格字符指定那些用于指定字符串值的 *options* 缺省值。

表 2-3：选项字符串值

选项名	选项值	函数
<i>binary</i>	hex   base64	forxmlj 和 for xml 子句
<i>columnstyle</i>	element   attribute	forxmlj 和 for xml 子句
<i>dtdvalidate</i>	yes   no	xmlvalidate
<i>entitize</i>	yes   no   conditional	forxmlj 和 for xml 子句
<i>format</i>	yes   no	forxmlj 和 for xml 子句
<i>header</i>	yes   no   encoding	forxmlj 和 for xml 子句
<i>incremental</i>	yes   no	for xmlj 子句
<i>nonnamespaceschemalocation</i>	请参见 xmlvalidate	xmlvalidate
<i>ncr</i>	non_ascii   non_server   no 有关缺省值，请参见函数说明。	forxmlj、for xml 子句、xmlextract
<i>nullstyle</i>	attribute   omit	forxmlj 和 for xml 子句

选项名	选项值	函数
<i>nullclause</i>	null   empty	forsqlcreatej forsqlscriptj
<i>prefix</i>	SQL 名称 (C) 缺省值为 C。	forxmlj 和 for xml 子句
<i>root</i>	yes   no	forxmlj 和 for xml 子句
<i>rowname</i>	SQL 名称 (row)	forxmlj 和 for xml 子句
<i>schemalocation</i>	请参见 xmlvalidate	forxmlj 和 for xml 子句
<i>schemavalidate</i>	yes   no	xmlvalidate
<i>statement</i>	yes   no	for xmlj 和 forxml 子句
<i>tablename</i>	SQL 名称 (resultset)	forxmlj 和 for xml 子句
<i>targetns</i>	含 URI 的加引号字符串	forxmlj 和 for xml 子句
<i>xmlerror</i>	exception   null   message	含 XML 操作数的所有函数
<i>xmlvalid</i>	document   message	xmlvalidate
<i>xsidecl</i>	yes   no	forxmlj 和 for xml 子句



# XML 语言和 XML 查询语言

对于 XML 文档，XML 查询函数支持 XML 1.0 标准；对于 XML 查询，XML 查询函数支持 XPath 1.0 标准。本章介绍了 XML 服务所支持的标准的一部分。

主题	页码
字符集支持	39
URI 支持	39
命名空间支持	40
XML 模式支持	40
XML 文档中的预定义实体	40
XPath 查询中的预定义实体	42
空白	42
空元素	43
XML 查询语言	43
带括号的表达式	52

## 字符集支持

XML 服务支持 SQL Server 支持的字符集。有关 I18N 的详细信息，请参见第 6 章“对 I18N 提供的 XML 支持”。

## URI 支持

XML 文档在两种环境中指定 URI（通用资源标识符），即作为 href 属性或文档文本，以及作为 DTD、实体定义、XML 模式和命名空间声明的外部引用。

将 URI 用作 href 属性或文档文本时，不存在任何限制，XML 服务会解析指定 http URI 的外部引用 URI。

不支持指定 file、ftp 或 relative URI 的外部引用 URI。

## 命名空间支持

您可以没有限制地使用命名空间声明和引用分析和存储 XML 文档。

但是，在 `xmlextract` 和 `xmltest` 中的 XML 表达式中引用带命名空间前缀的 XML 元素和属性名称时，将把命名空间前缀和冒号当作元素或属性名称的一部分。它们不作为命名空间引用进行处理。

## XML 模式支持

有关 `xmlvalidate` 的信息，请参见第 7 章 “`xmltable()`”。

## XML 文档中的预定义实体

表示引号 (")、撇号 (')、小于号 (<)、大于号 (>) 和 “与” 符号 (&) 的特殊字符在 XML 中用作标点符号，通过预定义的实体来表示：&quot;、&apos;、&lt;、&gt; 和 &amp;。请注意，分号是实体的一部分。

不能像在下列示例中所示的那样，在属性或元素中使用 “<” 或 “&”。

```
select xmlparse("<a atr='<'/>")
```

```
Msg 14702, Level 16, State 0:
```

```
Line 1:
```

```
XMLPARSE(): XML parser fatal error <<A '<' character  
cannot be used in attribute 'atr', except through <&gt;>  
at line 1, offset 14.
```

```
select xmlparse("<a atr1='&'>")
```

```
Msg 14702, Level 16, State 0:
```

```

Line 1:
XMLPARSE(): XML parser fatal error
<<Expected entity name for reference>>
at line 1, offset 11

select xmlparse("<a> < /a>")

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
<<Expected an element name>>
at line 1, offset 6.

select xmlparse(" & ")
Msg 14702, Level 16, State 0:
Line 1:
XMLPARSE(): XML parser fatal error
<<Expected entity name for reference>>
at line 1, offset 6.

```

应改用预定义实体 `&lt;` 和 `&amp;`，如下所示：

```

select xmlextract("/",
    "<a atr='&lt; &amp;'> &lt; &amp; </a>" )
-----
<a atr="&lt; &amp;"> &lt; &amp; </a>

```

在用撇号分隔的属性内可以使用引号，反之亦可。这些符号被预定义实体 `&quot;` 或 `&apos;` 替换。在下列示例中，请注意“yes”一词两边的引号或撇号被加倍，以符合 SQL 字符文字约定：

```

select xmlextract("/", "<a atr='""yes""' /> " )
-----
<a atr=" "yes" "></a>

select xmlextract('/', '<a atr=" "yes" "/> ' )
-----
<a atr=" 'yes' "></a>

```

在元素内可以使用引号和撇号。预定义实体 `&quot;` 和 `&apos;` 会替换它们，如下例所示：

```

select xmlextract("/", " ""yes"" and 'no' " )
-----
&quot;yes&quot; and 'no'

```

## XPath 查询中的预定义实体

当利用包含 XML 特殊字符的字符文字指定 XML 查询时，可以将其写为纯字符或预定义实体。下例显示两点：

- XML 文档包含一个 “<a>” 元素，它的值为 XML 特殊字符 &lt;>”，这些字符由其预定义实体 `&amp;&lt;&gt;&quot;` 来表示
- 下面的 XML 查询指定包含相同 XML 特殊字符的字符文字，这些字符也由其预定义实体来表示。

```
select xmlextract('/a="&amp;&lt;&gt;&quot;"',
                  "<a>&amp;&lt;&gt;&quot;</a>")
-----
<a>&amp;&lt;&gt;&quot;</a>
```

下面的示例与此相同，只不过 XML 查询使用纯 XML 特殊字符指定字符文字。在对查询求值之前，这些 XML 特殊字符将被预定义实体替换。

```
select xmlextract("/a='&lt;&gt;' " ,
                  "<a>&amp;&lt;&gt;&quot;</a>")
-----
<a>&amp;&lt;&gt;&quot;</a>
```

## 空白

将保留所有空白，它们在查询中是有意义的。

```
select xmlextract("/a[@atr=' this or that ']",
                  "<a atr=' this or that '><b> which or what  
</b></a>")
-----
<a atr=" this or that ">
<b> which or what </b></a>

select xmlextract("/a[b=' which or what ']",
                  "<a atr=' this or that '><b> which or what  
</b></a>")
-----
<a atr=' this or that '>
<b> which or what </b></a>
```

## 空元素

以 “<a/>” 样式输入的空元素将以 “<a></a>” 样式存储并返回：

```
select xmlextract("/",
  "<doc><a/> <b></b></doc>")
-----
      <doc>
      <a></a>
      <b></b></doc>
```

## XML 查询语言

XML 服务支持标准 XPath 语言的一个子集。该子集由下节中的语法和标识来定义。

### 支持的 XPath 语法和标识

XML 服务支持以下 XPath 语法：

```
xpath::= or_expr
or_expr::= and_expr | and_expr TOKEN_OR or_expr
and_expr::= union_expr | union_expr TOKEN_AND and_expr
union_expr::= intersect_expr
            | intersect_expr TOKEN_UNION union_expr
intersect_expr::= comparison_expr
            | comparison_expr TOKEN_INTERSECT intersect_expr
comparison_expr::= range_expr
            | range_expr general_comp comparisonRightHandSide
general_comp::= TOKEN_EQUAL | TOKEN_NOTEQUAL
            | TOKEN_LESSTHAN | TOKEN_LESSTHANEQUAL
            | TOKEN_GREATERTHAN | TOKEN_GREATERTHANEQUAL
range_expr::= unary_expr | unary_expr TOKEN_TO unary_expr
unary_expr::= TOKEN_MINUS path_expr
            | TOKEN_PLUS path_expr
            | path_expr
comparisonRightHandSide::= literal
path_expr::= relativepath_expr | TOKEN_SLASH
            | TOKEN_SLASH relativepath_expr
            | TOKEN_DOUBLESASH relativepath_expr
relativepath_expr::= step_expr
            | step_expr TOKEN_SLASH relativepath_expr
            | step_expr TOKEN_DOUBLESASH relativepath_expr
step_expr::= forward_step predicates
```

```

    | primary_expr predicates
    | predicates
primary_expr ::= literal | function_call | (xpath)
function_call ::=
    tolower([xpath])
    | toupper([xpath])
    | normalize-space([xpath])
    | concat([xpath [,xpath]...])
forward_step ::= abbreviated_forward_step
abbreviated_forward_step ::= name_test
    | TOKEN_ATEXPR name_test
    | TOKEN_PERIOD
name_test ::= q_name | wild_card | text test
text_test ::= TOKEN_TEXT TOKEN_LPAREN TOKEN_RPAREN
literal ::= numeric_literal | string_literal
wild_card ::= TOKEN_ASTERISK
q_name ::= TOKEN_ID
string_literal ::= TOKEN_STRING
numeric_literal ::= TOKEN_INT | TOKEN_FLOATVAL
    | TOKEN_MINUS TOKEN_INT
    | TOKEN_MINUS TOKEN_FLOATVAL
predicates ::=
    | TOKEN_LBRACKET expr TOKEN_RBRACKET predicates
    | TOKEN_LBRACKET expr TOKEN_RBRACKET

```

XPath 的 XML 服务子集支持以下标识:

```

APOS ::= "'"
DIGITS ::= [0-9]+
NONAPOS ::= "^"
NONQUOTE ::= "\""
NONSTART ::= LETTER | DIGIT | ':' | '-' | '_' | '.'
QUOTE ::= ""
START ::= LETTER | '_'
TOKEN_AND ::= 'and'
TOKEN_ASTERISK ::= '*'
TOKEN_ATEXPR ::= '@'
TOKEN_COMMA ::= ','
TOKEN_DOUBLESLEASH ::= '//'
TOKEN_EQUAL ::= '='
TOKEN_GREATERTHAN ::= '>'
TOKEN_GREATERTHANEQUAL ::= '>='
TOKEN_INTERSECT ::= 'intersect'
TOKEN_LESSTHAN ::= '<'
TOKEN_LESSTHANEQUAL ::= '<='
TOKEN_LPAREN ::= '('
TOKEN_LBRACKET ::= '['
TOKEN_MINUS ::= '-'
TOKEN_NOT ::= 'not'
TOKEN_NOTEQUAL ::= '!='
TOKEN_OR ::= 'or'
TOKEN_PERIOD ::= '.'

```

```
TOKEN_PLUS ::= '+'
TOKEN_RPAREN ::= ')'
TOKEN_RSQUARE ::= ']'
TOKEN_SLASH ::= '/'
TOKEN_TO ::= 'to'
TOKEN_UNION ::= '|' | 'union'
TOKEN_ID ::= START [NONSTART...]
TOKEN_FLOATVAL ::= DIGITS | '.'DIGITS | DIGITS'.'DIGITS
TOKEN_INT ::= DIGITS
TOKEN_STRING ::=
    QUOTE NONQUOTE...QUOTE
    | APOS NONAPOS...APOS
TOKEN_TEXT ::= 'text'
```

XPath 运算符

本节说明 XML 处理器所支持 XPath 子集。

XPath 基本运算符

表 3-1 显示所支持的基本 XPath 运算符。

表 3-1: XPath 基本运算符

运算符	说明
/	路径（子项）：子项运算符（/）从左侧集合的直接子项选择。
//	子代：子代运算符（//）从左侧集合的任意子代选择。
*	收集元素子项：可以通过替代 “*” 集合，在不使用其名称的情况下引用元素。
@	属性：属性名称前加有 “@” 符号。
[]	过滤器：可以通过将过滤器子句 “[ ]” 添加到集合来向任何集合应用约束和分支。过滤器类似于含 any 语义的 SQL where 子句。过滤器内含称作子查询的查询。如果将一个集合放在过滤器内，则当该集合包含任何成员时，将生成布尔值 “true”，当该集合为空时，则生成 “false”。
[n]	索引：索引主要用于在一组节点内查找特定节点。将索引包含在方括号内。第一个节点是索引 1。
text()	选择当前环境节点的文本节点。

XPath 集合运算符

第 46 页的表 3-2 显示所支持的 XPath 集合运算符。

表 3-2: XPath 集合运算符

运算符	说明
union	并集: union 运算符 (快捷方式为 “ ”) 返回左侧查询和右侧查询组合值的集合。重复项将被滤出, 并且结果列表会按文档顺序排序。
intersect	交集: intersect 运算符返回两个集合之间相同元素的集合。
()	分组: 可以使用括号将集合运算符分组。
. (句点)	句点: 句点项按照搜索环境求值。该项求值为仅包含此搜索环境的引用节点的集合。
布尔运算符 (and 和 or)	布尔表达式可以在子查询中使用。
and	布尔型 “and”。
or	布尔型 “or”。

XPath 比较运算符

表 3-3 显示了支持的 XPath 比较运算符。

表 3-3: XPath 比较运算符

运算符	说明
=	等于
!=	不等于
<	小于
>	大于
>=	小于等于
<=	大于等于



## XPath 函数

Adaptive Server 支持以下 XPath 字符串函数：

- toupper
- tolower
- normalize-space
- concat

### 一般准则和示例

本节介绍关于在 XPath 表达式中使用函数的一般准则。这些准则适用于列出的所有函数。下面的所有示例都使用了 `tolower` 函数，该函数将以小写形式返回单个参数。

可在使用梯级表达式的任意位置使用函数调用。

**示例 1**                      用作 XPath 查询顶级的函数称为顶级函数调用。下面的查询显示了作为顶级函数调用的 `tolower`：

```
select xmlextract
('tolower(//book[title="Seven Years in Trenton"]//first-name)', text_doc)
from sample_docs where name_doc='bookstore'
-----
joe
```

顶级函数调用的参数必须是绝对路径表达式；即参数必须以斜杠 (/) 或双斜杠 (//) 开头。

**示例 2**                      函数调用的参数可以是包含谓词的复杂 XPath 表达式，也可以是嵌套的函数调用：

```
select xmlextract
('//book[normalize-space(tolower(title))="seven years in trenton"]/author',
text_doc)
from sample_docs where name_doc='bookstore'
-----
<author>
  <first-name>Joe</first-name>
  <last-name>Bob</last-name>
  <award>Trenton Literary Review
  Honorable Mention</award>
</author>
```

**示例 3** 可以将函数用作相对梯级，也称为相对函数调用。下面的查询显示了作为相对函数调用的 **tolower**：

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

-----

joe
```

此示例说明相对函数的参数必须是相对路径表达式；即参数不能以斜杠 (/) 或双斜杠 (//) 开头。

**示例 4** 顶级函数和相对函数都可以使用文字作为参数。例如：

```
select xmlextract( 'tolower("aBcD")' ,text_doc),
       xmlextract( '/bookstore/book/tolower("aBcD")', text_doc)
from sample_docs where name_doc='bookstore'

-----

abcd      abcd
```

**示例 5** 字符串函数对其参数的文本进行运算。这是 **text()** 的隐式应用。例如，下面的查询以 XML 片段的形式返回 **first-name** 元素：

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//firstname', text_doc)
from sample_docs where name_doc='bookstore'

-----

<first-name>Joe</first-name>
```

下面的查询返回该 **first-name** XML 片段的文本：

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//first-name/text()', text_doc)
from sample_docs where name_doc='bookstore'

-----

Joe
```

下一个查询将对该 **first-name** 元素应用 **tolower**。此函数对该元素的文本进行隐式运算：

```
select xmlextract
( '//book[title="Seven Years in Trenton"] //tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

-----

joe
```

这与下一个示例将该 XML 元素的文本作为参数显式传递具有相同的效果：

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//tolower(first-name/text()),
text_doc)
from sample_docs where name_doc='bookstore'

-----

joe
```

**示例 6** 您在路径中将相对函数调用作为一个梯级加以应用。对该路径求值将生成一系列 XML 节点，然后对每个节点执行相对函数调用。其结果是一系列函数调用的结果。例如，下面的查询将生成一系列 `first_name` 节点：

```
select xmlextract( '/bookstore/book/author/first-name', text_doc)
from sample_docs where name_doc='bookstore'

-----
<first-name>Joe</first-name><first-name>Mary</first-name>
<first-name>Toni</first-name>
```

下面的查询用对 `toupper` 的调用替换上一查询的最后一个梯级，从而产生这两个函数调用的一系列结果。

```
select xmlextract('/bookstore/book/author/toupper(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

-----
JOEMARYTONI
```

现在，您可以使用 `concat` 为这一系列函数结果加标点。请参见第 51 页的“`concat`”中的示例。

**示例 7** `tolower`、`toupper` 和 `normalize-space` 都只有一个参数。如果在相对函数调用中指定这些函数时省略了参数，则当前节点成为隐含参数。例如，下面的示例显示了显式指定参数的 `tolower` 相对函数调用：

```
select xmlextract
('//book[title="Seven Years in Trenton"]//tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

-----

joe
```

同样是这个查询，下面的示例则隐式指定参数：

```
select xmlextract
('//book[title="Seven Years in Trenton"]//first-name/tolower()', text_doc)
from sample_docs where name_doc='bookstore'

-----

joe
```

对多个节点应用相对函数调用时，也可以在调用中隐式指定参数。例如：

```
select xmlextract('/book//first-name/tolower()', text_doc)
from sample_docs where name_doc='bookstore'
-----
joemarymarytoni
```

## 函数

本节介绍可增强 XML 服务的各个函数。

### tolower 和 toupper

**说明** tolower 和 toupper 分别以小写和大写形式返回其参数值。

**语法**                                `tolower(string-parameter)`  
                                      `toupper(string-parameter)`

**示例**                                下面的示例使用 `toupper` 以大写形式返回参数值。

```
select xmlextract
('//book[title="Seven Years in Trenton"]//toupper(first-name)', text_doc)
from sample_docs where name_doc='bookstore'
-----
JOE
```

### normalize-space

**说明**                                此函数返回其参数值时进行两处更改：

- 删除参数值前后的空白字符。
- 使用一个空白字符替换所有的两个或多个空白字符（非前导字符）子串。

**语法**                                `normalize-space(string-parameter)`

**示例**                                下面的示例将 `normalize-space` 应用于一个前后都有空格并且嵌有 `newline` 和 `tab` 字符的参数：

```
select xmlextract
('normalize-space(" Normalize space example.")', text_doc)
from sample_docs where name_doc='bookstore'
-----
Normalize space example.
```

当进行测试的值中的空格和大小写使用情况未知时，可以在 XPath 谓词中使用 `normalize-space` 和 `tolower` 或 `toupper`。以下谓词不受 `title` 元素中的大小写和空格使用情况的影响：

```
select xmlextract
  ('//magazine[normalize-space(tolower(title))='tracking trenton']//price',
   text_doc)
from sample_docs where name_doc='bookstore'
-----
<price>55</price>
```

## concat

**说明** `concat` 返回多个参数值的字符串并置结果。该函数可以有零个或多个参数。

**语法** `concat(string-parameter [,string-parameter]...)`

**示例** 在一个 `xmlextract` 调用中，`concat` 可以返回多个元素。例如，下面的查询将返回 `first-name` 和 `last-name` 两个元素：

```
select xmlextract ('//author/concat(first-name, last-name)', text_doc)
from sample_dcs where name_doc='bookstore'
-----
JoeBobMaryBobToniBob
```

也可以使用 `concat` 设置结果的格式并为结果加标点。例如：

```
select xmlextract
  ('//author/concat(",first(",first-name, ")-last(",last-name, " )")',
   text_doc)
from sample_docs where name_doc='bookstore'
-----
first (Joe) -last (Bob) first (Mary) -last (Bob) first (Toni) -last (Bob)
```

## 带括号的表达式

Adaptive Server 支持带括号的表达式。本节介绍 XPath 中带括号的表达式的常规语法。以下各节将介绍如何将括号与下标和联合一起使用。

### 括号和下标

下标应用于其前面紧邻的表达式。使用括号可对路径中的表达式分组。本节中的示例说明了如何将括号与下标一起使用。

下面是一个不使用下标的一般示例，它返回 **book** 元素中的所有书名。

```
select xmlextract('/bookstore/book/title', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Treanton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

若要只列出第一个书名，可以使用 “[1]” 下标，并输入下面的查询：

```
select xmlextract
('/bookstore/book/title[1]', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Treanton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

但是，上面的查询不会返回书店中的第一个书名，而是返回每本书中的第一个书名。同样，下面的查询（使用下标 “[2]”）返回每本书中的第二个标题，而不是书店中的第二个书名。因为每本书只有一个书名，所以结果是空的。

```
select xmlextract
('/bookstore/book/title[2]', text_doc)
from sample_docs where name_doc='bookstore'
-----
NULL
```

因为下标运算（谓词通常也是）应用于其前面紧邻的项，所以这些查询返回书中的（而不是书店的）第  $i$  个书名。若要返回整个书店中的第二个书名，而不是返回书中的第二个书名，请使用括号将应用下标的元素括起来。例如：

```
select smlextract
  ('(//bookstore/booktitle)[2]', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>History of Trenton</title>
```

可以用括号将任意路径分组。例如：

```
select xmlextract('(//title)[2]', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>History of Trenton</title>
```

## 括号和联合

还可以使用括号对同一梯级中的操作进行分组。例如，下面的查询返回书店中的所有书名。

```
select xmlextract('/bookstore/book/title', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Who's Who in Trenton</title>
```

上面的查询只返回书名。若要返回杂志名，请将此查询更改为：

```
select xmlextract('/bookstore/magazine/title', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Tracking Trenton</title>
```

要返回书店中所有项的名称，可以按照如下所示更改查询：

```
select xmlextract('/bookstore/*/title', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

如果书店包含除书籍和杂志以外的元素（例如，日历和报纸），则可以通过使用 **union**（竖线）运算符，并在查询路径中用括号将其括起来，来只查询书籍和杂志的名称。例如：

```
select xmlextract('/bookstore/(book|magazine)/title', text_doc)
from sample_docs where name_doc='bookstore'
-----
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```



# XML 映射函数

本章详细介绍 XML 映射函数并提供它们的示例。

主题	页码
<a href="#">for xml 子句</a>	<a href="#">55</a>
<a href="#">forxmlj、forxmldtdj、forxmlschemaj、forxmlallj</a>	<a href="#">65</a>
<a href="#">forsqlcreatej、forsqlinsertj、forsqlscriptj</a>	<a href="#">70</a>
<a href="#">使用 Java 函数映射分层 XML 文档和 SQL 数据</a>	<a href="#">74</a>
<a href="#">用于多个结果集查询的 Java SQLX 映射</a>	<a href="#">79</a>

## for xml 子句

指定一个 SQL select 语句，它返回结果集的 XML 表示形式。

语法

```
select ::=
  select [ all | distinct ] select_list
  [into_clause ]
  [where_clause ]
  [group_by_clause ]
  [having_clause ]
  [order_by_clause ]
  [compute_clause ]
  [read_only_clause ]
  [isolation_clause ]
  [browse_clause ]
  [plan_clause]
  [for_xml_clause]
for_xml_clause ::=
  for xml [schema | all] [option option_string] [returns_clause]
option_string ::= basic_string_expression
returns_clause ::=
  returns { char [(integer)] | varchar [(integer)] }
```

|unichar [(integer)] | univarchar [(integer)]  
|text | unitext | java.lang.String}

**注释** 有关选项字符串的详细信息，请参见第 36 页的“option\_strings: 一般格式”。

**注释** 有关使用具有 I18N 数据的 for xml 的详细信息，请参见第 6 章，第 105 页的“对 I18N 提供的 XML 支持”。

说明

- for xml 子句是 SQL select 语句中的一个新子句。以上所示的 select 语法包含所有子句，其中包括 for xml 子句。
  - 有关其它 select 语句子句的语法和说明，请参见 《Sybase Adaptive Server 参考手册》第 2 卷：《命令》。
  - for xml 子句支持 java.lang.string 数据类型，该数据类型表示为 string。任何其它 Java 类型都表示为 objectID。
- 
- **注释** 有关 for xml schema 和 for xml all 的说明，请参见第 61 页的“for xml schema 和 for xml all”。

for xml 子句的几种变体如下所示：

- a 如果 select 语句指定 for xml 子句，请将 select 语句本身作为基本 select 引用，将带有 for xml select 的 select 语句作为 for xml select 引用。例如，在以下语句中

```
select 1, 2 for xml
```

基本 select 是 select 1, 2， for xml select 是 select 1, 2 for xml。
- b for xml schema select 命令或子查询包含用于指定 schema 的 *for\_xml\_clause*。
- c for xml all select 命令或子查询包含用于指定 all 的 *for\_xml\_clause*。
- for xml select 语句不能包括 into\_clause、compute\_clause、read\_only\_clause、isolation\_clause、browse\_clause 或 plan\_clause。
- 不能在命令 create view、declare cursor、subquery 或 execute command 中指定 for xml select。

- `for xml select` 不能在联合中进行连接，但可以包含联合。例如，允许使用以下语句：

```
select * from T
union
select * from U
for xml
```

但不允许使用以下语句：

```
select * from T for xml
union
select * from U
```

- `for xml select` 的值是基本 `select` 语句结果的 XML 表示形式。该 XML 文档的格式为 SQLX 格式，有关此格式的说明，请参见第5章“XML 映射”。
- `returns` 子句指定由 `for xml` 查询或子查询生成的 XML 文档的数据类型。如果 `returns` 子句没有指定任何数据类型，则缺省值为 `text`。
- `for xml select` 语句返回的结果集取决于 *incremental* 选项：
  - *incremental = no* 返回含单行和单列的结果集。该列的值为基本 `select` 语句结果的 SQLX-XML 表示形式。这是缺省选项。
  - *incremental = yes* 返回的结果集对于基本 `select` 语句的每一行均包含一行。如果 `root` 选项指定 *yes*（缺省选项），则最初的行指定开始的 XML 根元素，而最后的行指定结束的 XML 根元素。

例如，以下 `select` 语句分别返回两行、一行、两行和四行：

```
select 11, 12 union select 21, 22
select 11, 12 union select 21, 22 for xml
select 11, 12 union select 21, 22
      for xml option "incremental=yes root=no"
select 11, 12 union select 21, 22
      for xml option "incremental=yes root=yes"
```

- `for xml` 查询结果中 `datetime` 值中的 `date` 和 `time` 字段由分隔符“T”（字母 T）加以分隔，这是 ANSI SQL-XML 标准的现行规定。如不采用此格式，将无法通过标准 XML 分析程序的验证。

例如，如果在 Adaptive Server 12.5.2 中执行此查询，结果将为：

```
select getdate() for xml

<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
< row >
< C1 > 2008-05-30 11:42:19 < /C1 >
```

```
< /row >
< /resultset >
```

但在 Adaptive Server 15.0.2 中执行同一查询时，结果将为：

```
select getdate() for xml

<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
< row >
< C1 > 2008-05-30T11:41:42 < /C1 >
< /row >
< /resultset >
```

选项 *option\_string* 的一般格式在[第 36 页](#)的“[option\\_strings: 一般格式](#)”中说明。for xml 子句的选项在[第 82 页](#)的“[SQLX 选项](#)”中说明。

异常 基本 select 语句执行过程中引发的任何 SQL 异常均由 for xml | select 引发。例如，下面两个语句都将引发除零异常：

```
select 1/0
select 1/0 for xml
```

示例 for xml 子句：

```
select pub_id, pub_name
from pubs2.dbo.publishers
for xml
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
  <pub_id>0736</pub_id>
  <pub_name>NewAgeBooks</pub_name>
</row>

<row>
  <pub_id>0877</pub_id>
  <pub_name>Binnet & Hardley</pub_name>
</row>

<row>
  <pub_id>1389</pub_id>
  <pub_name>Algodata Infosystems</pub_name>
</row>

</resultset>
```

## for xml 子查询

在 Transact-SQL 中，表达式子查询是带括号的子查询。它只包含一个列，该列的值是表达式子查询结果，而且必须返回一个行。您几乎可以在任何能够使用表达式的地方使用表达式子查询。有关子查询的详细信息，请参见《Transact-SQL® 用户指南》。

使用 for xml 子查询功能，您可以将任何包含 for xml 子句的子查询用作表达式子查询。

### 语法

```
subquery ::= select [all | distinct ] select_list
            (select select_list
             [from table_reference [, table_reference]... ]
             [where search_conditions]
             [group by aggregate_free_expression [aggregate_free_expression]...]
             [having search_conditions]
             [for_xml_clause])
for_xml_clause ::= 请参见第 64 页上的 “for xml schema 和 for xml all”
table_reference ::= table_view_name | ANSI_join | derived_table
table_view_name ::= 请参见《参考手册》第 2 卷《命令》中的 “SELECT”
ANSI_join ::= 请参见《参考手册》第 2 卷《命令》中的 “SELECT”
derived_table ::= (subquery) as table_name
```

### 说明

- 包含 for xml 子句的 select 命令生成一个表示 select 语句结果的 XML 文档，并将该 XML 文档返回为一个包含单行单列的结果集。您可以使用用于处理结果集的常规技术访问该结果集。
- 有关 for xml 子句及其 option\_string 的一般说明，请参见第 55 页的“for xml 子句”。有关支持 SCHEMA 关键字和 return 子句的 for xml 子句的扩展的说明，请参见第 61 页的“for xml schema 和 for xml all”。
- for xml 子查询是包含 for xml 子句的子查询。
- 您可以将 for xml 子查询用作表达式子查询，尽管它们之间存在一些差异；例如，普通表达式子查询具有以下限制，而 for xml 子查询则没有：
  - select 列表中不能包含多个项目
  - select 列表中不能有 text 和 image 列
  - 不能使用 group by 或 having 子句
- 不能在 for xml select 中或在其它 for xml 子查询中指定 for xml 子查询。
- 不能在以下命令中使用 for xml 子查询：
  - for xml select

- create view
- declare cursor
- select into
- 亦不能将其用作定量判定子查询，如 any/all、in/not in、exists/not exists
- for xml 子查询不能是相关子查询。有关相关子查询的详细信息，请参见《Transact-SQL 用户指南》。
- 不能在嵌套的标量子查询中使用返回 text 或 unitext 数据类型的 for xml 子句。
- for xml 子查询的数据类型由 *for\_xml\_clause* 的 returns 子句指定。如果 returns 子句没有指定任何数据类型，则缺省数据类型为 text。
- 异常与为 *for\_xml\_clause* 说明的异常相同。
- 如果 returns 子句指定了您不能转换子查询结果的数据类型，则会引发异常：结果不能转换成指定的数据类型。

异常

示例

**示例 1**

for xml 子查询将 XML 文档作为字符串值返回，您可以将这种值分配给字符串列或变量，也可以将其作为参数传递给存储过程或内置函数。  
例如：

```
declare @doc varchar(16384)
set @doc = (select * from systypes for xml returns varchar(16384))
select @doc
-----
```

**示例 2** 若要传递 for xml 子查询的结果作为字符串参数，请输入：

```
select xmlextract('//row[usertype = 18]',
                (select * from systypes for xml))
-----
```

**示例 3** 若要指定 for xml 子查询作为 insert 或 update 中的值，请输入：

```
create table docs_xml(id integer, doc_xml text)
insert into docs_xml
    select(1, (select * from systypes for xml))
-----

update docs_xml
set doc_xml = (select * from sysobjects for xml)
where id = 1
-----
```

## for xml schema 和 for xml all

本节说明 for xml 子句的其它形式。您可以生成 XML 模式、XML 模式和 XML DTD 或 XML 数据文档。

### 说明

- 带有 for xml schema 子句的 select 语句或子查询生成一个 XML 文档，该文档可用于描述 select 语句在包含 for xml 子句但不包含 schema 谓词时生成的同一 SQLX XML 结果集。

- 下面的 for xml 子查询的结果是 xml 值：

```
(subquery for xml schema option option_string)
```

此 xml 值与下面的查询的 java.lang.String 值结果相同：

```
forxmlj('subquery', option_string)
```

- 带有 for xml all 子句的 select 语句或子查询生成一个 XML 文档，该文档包含 SQLX 结果集、XML 模式以及用于描述该结果集的 XML DTD。这些内容包含在由以下元素构成的单个 XML 文档中：

- `<multiple-results>` — 根元素
- `<multiple-results-item type="result-set">` — 包含以下内容的一个元素：
  - `<multiple-results-item-dtd>` — 结果集的 DTD
  - `<multiple-result-item-schema>` — 结果集的 XML 模式
  - `<multiple-result-item-data>` — 结果集

### 选项

option\_string 的一般格式在[第 36 页](#)的“[option\\_strings: 一般格式](#)”中说明。for xml 子句的选项在[第 82 页](#)的“[SQLX 选项](#)”中说明。

### 异常

扩展异常与在[第 82 页](#)的“[SQLX 选项](#)”中说明的异常相同。

### 示例 — 用法

以下示例说明 for xml schema 和 for xml all 的用法。

**示例 1** 在此示例中，for xml all 子查询返回

- XML 模式
- XML 模式和 XML DTD
- XML 文档形式的结果集

这些都以字符串值形式返回，您可以将其分配给 string 列或变量，也可以将其作为字符串参数传递给存储过程或函数。

```
declare @doc varchar(16384)
set @doc = (select * from systypes for xml all returns varchar(16384))
select @doc
```

-----

**示例 2** 下面的示例传递 for xml schema 子查询的结果作为字符串参数:

```
select xmlextract('/row[usertype=18]'
  (select * from systypes for xml all))
-----
```

**示例 3** 下面的示例指定 for xml all 子查询作为 insert 或 update 命令中的值:

```
create table docs_xml(id integer, doc_xml xml)
insert into docs_xml
  values(1,(select * from sysobjects for xml all))
where id=1
```

示例 — 结果

下面这组示例显示了由上述示例中的命令生成的结果。

**示例 1** 下面的示例显示了基本 select for xml 语句的结果。

```
select "a", 1 for xml
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>a</C1>
    <C2>1</C2>
  </row>
</resultset>

(1 row affected)
```

**示例 2** 下面的示例显示的是 for xml schema, 该语句返回用于描述示例 1 中的结果集的 XML 模式。

```
select "a", 1 for xml schema
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
    schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd"/>

  <xsd:complexType name="RowType.resultset"
    <xsd:sequence>
      <xsd:element name="C1" type="VARCHAR_1"/>
      <xsd:element name="C2" type="INTEGER"/>
    </xsd:sequence>
  </xsd:complexType>
```



```

    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TableType.resultset"
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="VARCHAR_1">
    <xsd:restriction base="xsd:string".
      <xsd:length value="1"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:integer">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name+"resultset" type="TableType.resultset"/>
</xsd:schema>

(1 row affected)

```

**示例 3** 下面的示例使用 `for xml all` 返回结果集的模式、DTD 和数据。

```

select 'a', 1 for xml all
-----
<multiple results>

  <multiple-results-item type="result-set">
    <multiple-results-item-dtd>

      <!DOCTYPE resultset [
        <!ELEMENT resultset (row*)>
        <!ELEMENT row (C1,C2)>
        <!ELEMENT C1 (#PCDATA)>
        <!ELEMENT C2 (#PCDATA)>
      ]>

    </multiple-results-item-dtd>

```

```
</multiple-results-item-schema>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
    schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd"/>

  <xsd:complexType name="RowType.resultset">
    <xsd:sequence>
      <xsd:element name="C1" type="VARCHAR_1" />
      <xsd:element name="C2" type="INTEGER" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TableType.resultset">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="VARCHAR_1">
    <xsd:restriction base="xsd:string">
      <xsd:length value="1"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>

</multiple-results-item-data>

<multiple-results-item-data>

  <resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <row>
      <C1>a</C1>
      <C2>1</C2>
    </row>

  </resultset>

</multiple-results-item-data>
```

```
</multiple-results-item>

</multiple-results>

(1 row affected)
```

## forxmlj、forxmldtdj、forxmlschemaj、forxmlallj

---

**注释** 本节中的函数是基于 Java 的，必须先服务器上安装后才能使用。有关说明，请参见附录 D “基于 Java 的 XQL 处理器”。

---

基于 Java 的 forxml 函数将 SQL 查询的结果集映射到 SQLX-XML 模式和 / 或结果集文档中。SQL 查询指定为包含任意 SQL 查询表达式的字符串。

forxmlj 是由 select 语句的 for xml 子句提供的映射的函数形式。不同之处在于：

- 在某些环境（如函数参数、update 语句 set 子句和 insert 语句值列表）中，可以将 forxmlj 函数（但不能是 select 语句）与 for xml 一起使用。
- 具有 for xml 子句的 select 语句以 *returns\_clause* 中指定的数据类型返回结果。forxmlj 函数将结果返回为 java.lang.String。
- 带有 for xml 子句的 select 语句可以返回一行，也可以返回多行，具体取决于 *incremental* 选项。forxmlj 函数返回单个结果。

## 语法

```
forxmljfunction ::=  
    forxmlj(sql_query_expression, option_string)  
    | forxmldtdj(sql_query_expression, option_string)  
    | forxmlschemaj(sql_query_expression, option_string)  
forxmlallj_procedure ::=  
    execute forxmlallj  
        sql_query_expression, option_string  
        rs_target_out, schema_target_out, dtd_target_out  
sql_query_expression ::= basic_string_expression  
option_string ::= basic_string_expression
```

## 说明

- *basic\_string\_expression* 是一个 *sql\_query\_expression*，它的数据类型是 char、varchar、unichar、univarchar 或 java.lang.String。
- 如果 forxmlj 的任何参数为空，则该调用的结果为空。
- 如果 *sql\_query\_expression* 是全空白或空的字符串，则该调用的结果为空字符串。
- *sql\_query\_expression* 必须包含有效的 SQL select 语句，该语句可以包含 from 子句、where 子句、group by 子句、having 子句和 order by 子句。它不能包含 into 子句、compute 子句、read\_only 子句、isolation 子句、browse 子句或 plan 子句。
- forxmlj 对 *sql\_query\_expression* 求值，并返回包含 SQLX 格式结果集的 SQLX-XML 文档。
- forxmldtdj 对 *sql\_query\_expression* 求值，并返回一个 XML DTD，它描述该查询的 SQLX-XML 结果集。
- forxmlschemaj 对 *sql\_query\_expression* 求值，并返回一个 SQLX-XML 模式，它描述该查询的 SQL-XML 结果集。
- forxmlallj 过程对 *sql\_query\_expression* 求值，并返回该查询的 SQLX-XML 结果集、模式和 DTD。

---

**注释** 有关 SQL 结果集的 SQLX-XML 表示形式的说明，请参见第 5 章“XML 映射”。

---

## 选项

option\_string 的一般格式在第 36 页的“option\_strings: 一般格式”中说明。有关 for xml 子句的选项的说明，请参见第 5 章“XML 映射”。

## 异常

*sql\_query\_expression* 执行过程中引发的任何 SQL 异常均由 forxmlj 函数引发。

示例

forxmlj 函数:

```
set stringsize 16384
select forxmlj
    ("select pub_id, pub_name
     from pubs2.dbo.publishers", "")
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
    <pub_id>0736</pub_id>
    <pub_name>New AgeBooks</pub_name>
</row>

<row>
    <pub_id>0877</pub_id>
    <pub_name>Binnet & Hardley</pub_name>
</row>

<row>
    <pub_id>1389</pub_id>
    <pub_name>Algodata Infosystems</pub_name>
</row>

</resultset>
```

forxmldtdj 函数:

```
set stringsize 16384
select forxmldtdj
    ("select pub_id, pub_name
     from pubs2.dbo.publishers",
     "tablename=extract nullstyle=omit")
-----
<!ELEMENT extract (row*)>
<!ELEMENT row (pub_id, pub_name?)>
<!ELEMENT pub_id (#PCDATA)>
<!ELEMENT pub_name (#PCDATA)>
```

forxmlschemaj 函数:

```
set stringsize 16384
select forxmlschemaj
    ("select pub_id, pub_name
     from pubs2.dbo.publishers",
     "tablename=extract nullstyle=omit")

<xsd:schema
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sqlxml=
"http://www.iso-standards.org/mra/9075/sqlx">

<xsd:simpleType name="CHAR_4">
  <xsd:restriction base="xsd:string">
    <xsd:length value="4"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="VARCHAR_40">
  <xsd:restriction base="xsd:string">
    <xsd:length value="40"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="RowType.extract">
  <xsd:sequence>
    <xsd:element name="pub_id" type="CHAR_4"
      minOccurs="0" MaxOccurs="1"/>
    <xsd:element name="pub_name" type="VARCHAR_40"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.extract">
  <xsd:sequence>
    <xsd:element name="row" type="RowType.extract"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="extract" type="TableType.extract"/>
</xsd:schema>
```

forxmlallj 过程:

```
set stringsize 16384
declare @rs varchar(16384)
declare @schema varchar(16384)
declare @dtd varchar(16384)
execute forxmlallj
  "select pub_id, pub_name from pubs2.dbo.publishers",
  "name=extract null=attribute",
  @rs out, @schema out, @dtd out
-----
<extract
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
  <pub_id>0736</pub_id>
  <pub_name>New Age Books</pub_name>
</row>

<row>
  <pub_id>0877</pub_id>
  <pub_name>Binnet & Hardley</pub_name>
</row>

<row>
  <pub_id>1389</pub_id>
  <pub_name>Algodata Infosystems</pub_name>
</row>
</extract>

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml=
    "http://www.iso-standards.org/mra/9075/sqlx">
<xsd:simpleType name="CHAR_4">
  <xsd:restriction base="xsd:string">
    <xsd:length value="4"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="VARCHAR_40">
  <xsd:restriction base="xsd:string">
    <xsd:length value="40"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="RowType.extract">
  <xsd:sequence>
    <xsd:element name="pub_id" type="CHAR_4"
      nullable="true" />
    <xsd:element name="pub_name" type="VARCHAR_40"
      nullable="true" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.extract">
  <xsd:sequence>
    <xsd:element name="row" type="RowType.extract"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

```
</xsd:sequence>
</xsd:complexType>
<xsd:elementname="extract" type="TableType.extract">
</xsd:schema>
```

```
<!ELEMENT extract (row*)>
<!ELEMENT row (pub_id, pub_name)>
<!ELEMENT pub_id (#PCDATA)>
<!ELEMENT pub_name (#PCDATA)>
```

## forsqlcreatej、forsqlinsertj、forsqlscriptj

---

**注释** 本节中的函数是基于 Java 的，必须先服务器上安装后才能使用。

---

基于 Java 的 forsql 函数将 SQLX-XML 模式和 SQLX-XML 结果集文档映射到 SQL 脚本。

- SQLX-XML 模式和结果集文档具有 forxmlj 函数所生成的格式。
- forsqlschemaj 函数将 SQLX-XML 模式映射到一个 SQL create 命令，并创建一个对 SQLX-XML 模式所描述的数据适合的表。
- forxmlinsertj 函数将 SQLX-XML 结果集映射到一系列 SQL insert 命令，并重新创建 SQLX-XML 结果集所描述的数据。
- forxmlscriptj 函数将 SQLX-XML 模式和 SQLX-XML 结果集都映射到 SQL create 命令，并创建一个适合于 SQLX-XML 模式所描述的数据的表以及一系列 SQL insert 命令，这些命令重新创建 SQLX-XML 结果集所描述的数据。

语法

```
sqlx_to_sql_script_function ::=
    forsqlcreatej(sqlx_schema, option_string)
    | forsqlinsertj(sqlx_resultset, option_string)
    | forsqlscriptj(sqlx_schema, sqlx_resultset, option_string)
sqlx_schema ::= basic_string_expression
sql_resultset ::= basic_string_expression
option_string ::= basic_string_expression
```



## 说明

- *basic\_string\_expression* 是一个 *sql\_query\_expression*，它的数据类型是 `char`、`varchar`、`unichar`、`univarchar` 或 `java.lang.String`。
- 如果 `forsqlcreatej`、`forsqlschemaj` 或 `forsqlscriptj` 的任何参数为空，则该调用的结果为空。
- 如果 *sqlx\_schema* 或 *sqlx\_resultset* 为全空白或空的字符串，则该调用的结果为空字符串。
- *sqlx\_schema* 必须包含有效的 XML 文档，该文档包含 SQLX-XML 模式。
- *sqlx\_resultset* 必须包含有效的 XML 文档，该文档包含 SQLX-XML 结果集。
- `forsqlcreatej` 生成 SQL `create` 命令以创建适合于 *sqlx\_schema* 所描述的数据的 SQL 表。
- `forsqlinsertj` 生成一系列 SQL `insert` 命令以使用 *sqlx\_resultset* 的数据填充 SQL 表。

由于此函数在没有对应模式的 SQLX-XML 结果集上操作，因此所生成的 `insert` 命令假定所有数据均为 `varchar`。

- `forsqlscriptj` 生成 SQL `create` 命令和一系列 SQL `insert` 命令以使用 *sqlx\_resultset* 的数据填充 SQL 表。

由于此函数既在 SQLX-XML 模式上操作又在结果集上操作，`create` 指定 *sqlx\_schema* 的列数据类型，而 `insert` 命令则采用这些数据类型。

- 所生成的脚本会将带引号的标识符用于所有标识符。这不会影响随后对任何常规标识符的引用。

## 选项

*option\_string* 的一般格式在 [第 36 页](#) 的“[option\\_strings: 一般格式](#)”中说明。

`forsqlcreatej`、`forsqlinsertj` 和 `forsqlscriptj` 函数支持下面的选项，此选项在下文中“异常”一节中说明。

```
xmlerror={exception | null | message}
```

## 异常

如果 *sqlx\_schema* 或 *sqlx\_resultset* 的值不是有效的 XML:

- 如果显式或缺省选项指定:

```
xmlerror=exception
```

则引发如下异常:

```
invalid XML data
```

- 如果显式或缺省选项指定:

```
xmlerror=null
```

则返回空值。

- 如果显式或缺省选项指定:

```
xmlerror=message
```

则返回一个字符串, 它包含内含异常消息的 XML 元素。该值采用 SQL 注释的形式, 因此返回值是有效的 SQL。

## 示例

**forsqlcreatej 函数:**

```
set stringsize 16384
declare @schema varchar(16384)
select @schema = forxmlschema(
    "select pub_id, pub_name from pubs2.dbo.publishers",
    "tablename=extract null=attribute")
select forsqlcreatej(@schema, "")
-----
CREATE TABLE "extract"(
    "pub_id" CHAR(4) null,
    "pub_name" VARCHAR(40) null )
```

**forsqlinsertj 函数:**

```
set stringsize 16384
declare @rs varchar(16384)
select @rs = forxmlj(
    "select pub_id, pub_name from pubs2.dbo.publishers")
select forsqlinsertj(@rs, "")
-----
-- 开始处理表 "resultset"
insert into "resultset"
    ("pub_id", "pub_name")
    values ( '0736', 'New Age Books')
insert into "resultset"
    ("pub_id", "pub_name")
    values ( '0877', 'Binnet & Hardley')
insert into "resultset"
```

```

        ("pub_id", "pub_name")
        values ( '1389', 'Algodata Infosystems')
-- 表 “resultset” 处理结束

```

forsqlscriptj 函数:

```

set stringsize 16384
declare @rs varchar(16384)
declare @schema varchar(16384)
declare @dtd varchar(16384)
execute forxmlallj
    "select pub_id, pub_name from pubs2.dbo.publishers",
    "tablename=extract null=attribute",
    @rs out, @schema out, @dtd out
declare @script varchar(16384)

select @script = forsqlscriptj(@schema, @rs, "")
select @script
execute ("set quoted_identifier on " + @script )
execute ("select pub_id, pub_name from extract")
execute ("drop table extract")
-----
(return status = 0)

Return parameters:

*****Values of @rs, @schema, and @dtd omitted*****
(1 row affected)
(1 row affected)

CREATE TABLE "extract"(
    "pub_id" CHAR(4) null,
    "pub_name" VARCHAR(40) null)

-- 开始处理表 “extract”
insert into "extract"
    ("pub_id", "pub_name")
    values ( '0736', 'New Age Books')
insert into "extract"
    ("pub_id", "pub_name")
    values ( '0877', 'Binnet & Hardley')
insert into "extract"
    ("pub_id", "pub_name")
    values ( '1389', 'Algodata Infosystems')
-- 表 “extract” 处理结束

(1 row affected)

```

```
(1 row affected)
(1 row affected)
(1 row affected)

pub_id pub_name
-----
1) New Age Books
2) Binnet & Hardley
3) Algodata Infosystems

(3 rows affected)
```

## 使用 Java 函数映射分层 XML 文档和 SQL 数据

Adaptive Server 支持两种面向客户端的基于 Java 的 XML 函数，以便在 SQL 表或结果集和分层 XML 文档之间映射数据。这两个参数为：

- **ForXmlTree** — 将一组 SQL 表或结果集映射到树形结构的 XML 文档。
- **OpenXml** — 将树形结构的 XML 文档中的重复数据提取到 SQL 表中。

以下几节提供了示例数据以及有关如何使用 **ForXmlTree** 和 **OpenXml** 的概述和示例。有关详细说明，请参见 *\$SYBASE/\$SYBASE\_ASE/sample/XML/xml-util.{doc,pdf}*。

## 示例数据及其树形结构的 XML 表示形式

SQL 数据存储表中，使用外键列和主键列提供各表之间的树形结构关系。在 XML 中描述这样的数据时，树形结构关系通常用嵌套的元素表示。

例如，假定有带有表 4-1 中显示的数据的表。

**表 4-1: 示例表**

**表数据**

```
depts(dept_id, dept_name)
emps(emp_id, emp_name, dept_id)
emp_phones(emp_id, phone_no)
projects(project_id, dept_id)
```

表 4-1 中数据的树形结构 XML 表示形式为：

```
<sample xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<depts>
<dept>
  <dept_id>D123</dept_id>
  <dept_name>Main</dept_name>
  <emps>
    <emp>
      <emp_id>E123</emp_id>
      <emp_name>Alex Allen</emp_name>
      <salary>912.34</salary>
      <phones>
        <phone><phone_no>510.555.1987</phone_no></phone>
        <phone><phone_no>510.555.1876</phone_no></phone>
        <!-- 此 emp 的其它 phone 元素 -->
      </phones>
    <!-- 此 dept 的其它 emp 元素 -->
  </emps>

  <projects>
    <project>
      <project_id>PABC</project_id>
      <budget>598.76</budget>
    </project>
    <!-- 此 dept 的其它 project 元素 -->
  </projects>
</dept>
<!-- 此 depts 集的其它 dept 元素。 -->
</depts>
</sample>
```

## 使用 ForXmlTree 将 SQL 数据映射到分层 XML

基于 Java 的函数 ForXmlTree 将一组 SQL 表或结果集映射到一个树型结构的 XML 文档。这基于 SQL select 命令的 for xml 子句，该子句是在 Adaptive Server 12.5.1 中引入的。

select...for xml 可执行以下任务：

- 将单个 SQL 结果集映射到单个 XML 文档。
- 生成 SQL 结果集到 XML 的直接映射。例如，如果 select 返回一个共有 1000 行、每行各有 20 列的结果集，则 for xml 返回的 XML 文档有 1000 个行元素，而且每个行元素有 20 个列元素。

基于 Java 的函数 ForXmlTree：

- 可以在 SQL 服务器、客户端命令行或者客户端或服务器 Java 应用程序中调用。
- 将结果集集合映射到单个树形结构的 XML 文档。
- 需要 <forxmltree> 规范参数，用以描述输出树和树的每个节点要包含的 SQL 数据。
- 生成输出树形结构的 XML 文档的每个节点上的 XML 数据的 for xml 类型的映射。

因此，可以将 ForXmlTree 功能视为二维 for xml 映射。例如，为 ForXmlTree 提供以下 <forxmltree> 输入时将生成第 75 页的“示例数据及其树形结构的 XML 表示形式”中显示的 XML 文档。

```

1) <!-- 用于 depts-emps-phones-projects 的 forxmltree 规范，包含集合 -->
2) <forxmltree treename="sample">
3) <node> <!-- depts 的节点元素 -->
4)   <query> select * from depts order by dept_id </query>
5)   <options> tablename=depts rowname=dept </options>
6)   <link variablename="@dept_id" columnname="dept_id" type="char(11)" />
7)   <node> <!-- depts 下的 emps 的节点元素 -->
8)     <query>
9)       select emp_id, emp_name, salary from emps e
10)      where e.dept_id = @dept_id order by emp_id
11)    </query>
12)    <options> tablename=emps rowname=emp </options>
13)    <link variablename="@emp_id" columnname="emp_id" type="char(6)"/>
14)    <node> <!-- emps 下的 phones 的节点元素 -->
15)      <query>
16)        select phone_no from emp_phones ep where ep.emp_id = @emp_id
17)      </query>
18)      <options> tablename=phones rowname=phone </options>

```

```

19)         </node> <!-- phones 的节点结束 -->
20)     </node> <!-- emps 的节点结束 --
21) <node> <!-- dept 下的 projects 的节点元素 -->
22)     <query>
23)         select project_id, budget from projects p
24)         where p.dept_id = @dept_id order by project_id
25)     </query>
26)     <options> tablename=projects rowname=project </options>
27) </node> <!-- projects 的节点结束 -->
28) </node> <!-- depts 的节点结束 -->
29) </forxmldata>

```

## 使用 OpenXml 将分层 XML 映射到 SQL

第 76 页的“使用 ForXmlTree 将 SQL 数据映射到分层 XML”中所述的 ForXmlTree 函数可将一组 SQL 表或结果集映射到分层 XML 文档。

OpenXml 函数执行相反的过程：从输入 XML 文档中为 SQL 表提取数据。

OpenXml 类似于 Adaptive Server 12.5.1 中引入的 `xmlextract` 函数，可从给定的 XML 文档中提取指定的数据值。`xmlextract` 可指定 XML 文档和单个 XPath 查询表达式，并返回向 XML 文档应用 XPath 查询所得的结果。

基于 Java 的 OpenXml 函数：

- 可以从客户端命令行或客户端 Java 应用程序中调用，不适用于在 SQL 服务器中使用。
- 需要在参数中包括指定的 XML 文档和一组选项，用于指定提取所需的输出行的 XPath 查询以及在每个输出行中提取所需列的 Xpath 查询。

因此，可以将 OpenXml 当作二维的 `xmlextract`。

OpenXml 将执行以下的一个或两个操作：

- 生成 SQL 脚本，用提取的数据创建和填充 SQL 表。
- 执行 SQL 脚本，用提取的数据创建 SQL 表。

以下示例假设第 75 页的“示例数据及其树形结构的 XML 表示形式”中的 XML 文档存储在 `example-document.xml` 中。

**示例 1**                    下面的示例显示了四个客户端命令行调用，它们分别从该 XML 文档中提取 **depts**、**emps**、**emp\_phones** 和 **projects** 表。

```
java jcs.xmlutil.OpenXml -i "file:example-document.xml" \  
    -r "file:depts.opt" -o "depts.sql"  
  
java jcs.xmlutil.OpenXml -i "file:example-document.xml" \  
    -r "file:emps.opt" -o "emps.sql"  
  
java jcs.xmlutil.OpenXml -i "file:example-document.xml" \  
    -r "file:emp_phones.opt" -o "emp_phones.sql"  
  
java jcs.xmlutil.OpenXml -i "file:example-document.xml" \  
    -r "file:projects.opt" -o "projects.sql"
```

**示例 2**                    下面的示例显示了命令行在**示例 1** 引用中调用的选项的内容。这些选项用于指定 **OpenXml** 调用应该提取的数据，以及应将这些数据存储到的 SQL 表。

```
-- 输入文件 “depts.opt” 的内容  
tablename='depts_ext'  
rowpattern='//dept'  
columns=  
    ' dept_id char( 4 ) "/"@dept_id"  
      dept_name varchar(50) "/"@dept_name" '  
  
-- 输入选项文件 “emps.opt” 的内容  
tablename='emps_ext'  
rowpattern='//dept/emps/emp'  
columns=  
    ' emp_id char( 4 ) "/"emp_id/text()"   
      emp_name varchar(50) "/"emp_name/text()"   
      dept_id char(4) "/"../..@dept_id"   
      salary dec(7,2) "/"salary/text()"   
  
'-- 输入选项文件 “emp_phones.opt” 的内容  
tablename='emp_phones_ext'  
rowpattern='/sample/dept/emps/emp/phone'  
columns= ' emp_id char( 4 ) "/"../emp_id/text()"   
          phone_no varchar(20) "/"@phone_no" '  
  
-- 输入选项文件 “projects.opt” 的内容  
tablename='projects_ext'  
rowpattern='//dept/projects/project'  
columns=  
    ' project_id char( 4 ) "/"project_id/text()"   
      dept_id char(4) "/"../..@dept_id"   
      budget dec(7,2) "/"budget/text()" '
```



## 示例 3

下面的示例显示了由第一个 OpenXml 调用生成的 SQL 脚本。此脚本使用提取的 `depts` 表中的数据创建和填充表。示例 1 中显示的后续 OpenXml 调用分别为 `emps`、`emp_phones` 和 `projects` 数据生成类似的脚本。

```
-- 输出文件 depts.sql

create table depts_ext
    (dept_id char( 4 ) null, dept_name varchar(50) null
    )

insert into depts_ext values('D123', 'Main')

insert into depts_ext values('D234', 'Auxiliary')

insert into depts_ext values('D345', 'Repair')
```

## 用于多个结果集查询的 Java SQLX 映射

`select ... for xml` 语句和基于 Java 的 SQLX 映射函数可将单个 SQL 结果集映射到 SQLX 格式的 XML 文档。Adaptive Server 提供了基于 Java 的 SQLX 映射函数 `forxmlmultiplej`，这个函数可以将 SQL 查询的多个结果集映射到一个 XML 文档。

### forxmlmultiplej

## 说明

将可以包含多个结果集的 SQL 查询结果集映射到一个 XML 文档。

## 语法

```
forxmlmultiplej_function ::=
    forxmlmultiplej(sql_query_expression, option_string)
```

## 选项

有关 `sql_query_expression` 和 `option_string` 的介绍，请参见《XML 服务》第 4 章“XML 映射函数”中的“`forxmlj`、`forxmldtdg`、`forxmlschemaj`、`forxmlallj`”。

## 用法

- `sql_query_expression` 可以返回多个结果集，并可以包含 SQL `print` 命令。
- 有关 `forxmlmultiplej` 的示例和完整介绍，请参见 `$$SYBASE/$SYBASE_ASE/sample/XML/Using-SQLX-mappings.htm` 中的“多个结果集”。



使用 ANSI SQLX 标准所定义的 SQLX-XML 格式，select 语句中的 for xml 子句以及 forxmlj 函数可将 SQL 结果集映射到 SQLX-XML 文档。本章介绍 SQLX-XML 格式以及 for xml 子句和 forxmlj 函数均支持的选项。

**注释** 通过将 isql 与 for xml 子句搭配使用来生成 XML 文档时，由于 isql 会添加一段前导空白作为列分隔符，因此所生成的文档可能无效。

主题	页码
<a href="#">SQLX 选项</a>	82
<a href="#">SQLX 数据映射</a>	92
<a href="#">SQLX 模式映射</a>	99

# SQLX 选项

**注释** 表 5-1 中，带下划线的词指定的是缺省值。

**表 5-1: SQLX 映射的选项**

选项名	选项值	目的
<i>binary</i>	<u>hex</u>   base64	二进制的表示形式。应用于 forxmlj。
<i>columnstyle</i>	<u>element</u>   attribute	SQL 列的表示形式
<i>entitize</i>	yes   no   <u>cond</u>	forxmlj 和 for xml 子句
<i>format</i>	<u>yes</u>   no	包含格式设置
<i>header</i>	yes   no   encoding  缺省值取决于返回类型。请参见第 6 章“对 I18N 提供的 XML 支持”	包含 XML 声明
<i>incremental</i>	yes   <u>no</u>	从指定 for xml 的 select 语句中返回单行或多行
<i>multipleentitize</i>	yes   <u>no</u>	forxmlemultiplej 和 for xml 子句
<i>nullstyle</i>	attribute   <u>omit</u>	<i>columnstyle=element</i> 时空值的表示形式
<i>ncr</i>	<u>non_ascii</u>   non_server   no	forxmlj 和 for xml 子句
<i>prefix</i>	SQL 名称	所生成名称的基名。缺省值为 C。
<i>root</i>	<u>yes</u>   no	包含表名的根元素
<i>rowname</i>	SQL 名称	行元素的名称。 缺省值为 row。
<i>schemaloc</i>	含 URI 的加引号字符串	<i>schemalocation</i> 值
<i>statement</i>	yes   <u>no</u>	包含 SQL 查询
<i>tablename</i>	SQL 名称	根元素的名称。缺省值为 resultset。
<i>targetns</i>	含 URI 的加引号字符串	<i>targetnamespace</i> 值（如果有）
<i>xsidecl</i>	<u>yes</u>   no	forxmlj 和 for xml 子句

## SQLX 选项定义

本节定义表 5-1 中显示的 SQLX 选项。

**binary={hex | base64}**

此选项指示是用 hex 编码还是用 base64 编码来表示其数据类型为 **binary**、**varbinary** 或 **image** 的列。这一选择将取决于用来处理所生成文档的应用程序。Base64 编码比 hex 编码更紧凑。

下面的示例显示了缺省选项 *binary=hex*。

```
select forxmlj("select 0x012131415161718191a1b1c1d1e1f1",
"binary=hex")
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 服务 ce">
  <row>
    <C1>012131415161718191A1B1C1D1E1F1</C1>
  </row>
</resultset>
```

下面的示例显示了 *binary=base64*：

```
select forxmlj("select 0x012131415161718191a1b1c1d1e1f1",
"binary=base64")
-----
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
  <C1>ASEXQVFhcYGRobHB0eHx</C1>
</row>
</resultset>
```

**columnstyle=**  
**{element | attribute}**

此选项指示是将 SQL 列表示为元素还是表示为 XML “行” 元素的属性。

下面的示例显示了缺省选项 *columnstyle=element*：

```
select pub_id, pub_name from pubs2..publishers
for xml option "columnstyle=element"
-----
<resultset xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">

  <row>
    <pub_id>0736</pub_id>
    <pub_name>New Age Books</pub_name>
  </row>

  <row>
    <pub_id>0877</pub_id>
```

```

        <pub_name>Binnet & Hardley</pub_name>
    </row>

    <row>
        <pub_id>1389</pub_id>
        <pub_name>Algodata Infosystems</pub_name>
    </row>

</resultset>

```

下面的示例显示了 *columnstyle=attribute*:

```

select pub_id, pub_name from pubs2..publishers
for xml option "columnstyle=attribute"
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <row
        pub_id="0736"
        pub_name="New Age Books"
    />
    <row
        pub_id="0877"
        pub_name="Binnet & Hardley"
    />
    <row
        pub_id="1389"
        pub_name="Algodata Infosystems"
    />
</resultset>

```

entitize =  
{yes | no | cond}

此选项指定了是否将字符串列中的 XML 保留字符（“<”、“&”、“'”、“”）转换为 XML 实体（&lt; &apos; &gt; &amp; &quote;）。使用 **yes** 或 **no** 可指示是否要将保留字符实体化。**cond** 只有在列中的第一个非空字符不是 “<” 时才会将保留字符实体化。**for xml** 将第一个字符为 “<” 的字符串列视作 XML 文档，因而不将它们实体化。

例如，下面的示例实体化了所有的字符串列：

```

select 'a<b' for xml option 'entitize=yes'
-----
<resultset>
    <row>
        <C1><a&lt;b</C1>
    </row>
</resultset>

```

但是，下面的示例没有实体化任何字符串列：

```
select '<ab>' for xml option 'entitize=no'
-----
<resultset>
  <row>
    <C1><ab></C1>
  </row>
</resultset>
```

下面的示例实体化了开头不是 “<” 的字符串列：

```
select '<ab>', 'a<b' for xml option 'entitize=cond'
-----
<resultset>
  <row>
    <C1><ab></C1>
    <C2>a< b</C2>
  </row>
</resultset>
```

**format={yes | no}**

此选项指定是否包含换行符和制表符的格式设置。

例如：

```
select 11, 12 union select 21, 22
for xml option "format=no"
-----
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row><C1>11</C1><C2>12</C2></row>
<row><C1>21</C1><C2>22</C2></row>
</resultset>
```

**header=**  
**{yes | no | encoding}**

此选项指定是否将 XML 标头行包含在所生成的 SQLX-XML 文档中。  
XML 标头行如下所示：

```
<?xml version="1.0"?>
```

如果将所生成的 SQLX-XML 文档用作独立的 XML 文档，请包含这样的标头行。如果将所生成的文档与其它 XML 组合，则忽略标头行。

有关编码选项的说明，请参见第 105 页的“[对 I18N 提供的 XML 支持](#)”。

例如：

```
select 1,2 for xml option "header=yes"
-----
<?xml version="1.0" ?>
<resultset xmlns:xsi="http://www.w3.org/2001
```

```

        /XMLSchema-instance">
    <row>
        <C1>1</C1>
        <C2>2</C2>
    </row>
</resultset>

```

**incremental={yes | no}** 此选项仅适用于 `for xml` 子句，而不适用于 `forxml` 函数。它指定含 `for xml` 子句的 `select` 语句返回下面的哪一项：

- *incremental=no* — 返回单个行，此行只有数据类型为 **text** 的单个列，其中包含用于存放 `select` 语句结果的完整 SQLX-XML 文档。  
*incremental=no* 是缺省选项。
- *incremental=yes* — 为 `select` 语句结果的每个行分别返回一个行，此行只有数据类型为 **text** 的单个列，列中包含用于该结果行的 XML 元素。
  - 如果 *root* 选项为 *yes*（缺省值），则 *incremental=yes* 选项还将返回两个额外的行，其中包含 *tablename* 的打开和关闭元素。
  - 如果 *root* 选项为 *no*，则将忽略 *tablename* 选项（显式或缺省）。没有两个额外的行。

例如，以下三个 `select` 语句将分别返回一行、两行和四行。

```

select 11, 12 union select 21, 22
for xml option "incremental=no"

select 11, 12 union select 21, 22
for xml option "incremental=no root=no"

select 11, 12 union select 21, 22
for xml option "incremental=no root=yes"

```

**multipleentitize={yes | no}** 此选项用于 `for xml all`。有关实体化的说明，请参见选项 “Entitize = yes | no”。

**ncr={no | non\_ascii | non\_server}** 请参见第 106 页的 “数值字符表示”。

**nullstyle={attribute | omit}** 此选项指定当 *columnstyle* 已指定或缺省为 *columnstyle=element* 时，将使用空值的哪种可选 SQLX 表示形式。当 *columnstyle=attribute* 已指定时，*nullstyle* 选项无关。*nullstyle=omit* 选项（缺省选项）指定应该从包含空列的行中省略空列。*nullstyle=attribute* 选项指示应该包含空列作为具有 *xsi:nil=true* 属性的空元素。



下面的示例显示了 *nullstyle=omit* 选项，它也是缺省选项：

```
select 11, null union select null, 22
for xml option "nullstyle=omit"
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>11</C1>
  </row>
  <row>
    <C2>22</C2>
  </row>
</resultset>
```

下面的示例显示了 *nullstyle=attribute*：

```
select 11, null union select null, 22
for xml option "nullstyle=attribute"
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>11</C1>
    <C2 xsi:nil="true"/>
  </row>
  <row>
    <C1 xsi:nil="true"/>
    <C2>22</C2>
  </row>
</resultset>
```

root= {yes | no}

此选项指定 SQLX-XML 结果集是否应包含 *tablename* 的 *root* 元素。缺省值为 *root=yes*。如果 *root=no*，则将忽略 *tablename* 选项。

```
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <row>
    <C1>11</C1>
    <C2>12</C2>
  </row>

  <row>
    <C1>21</C1>
    <C2>22</C2>
  </row>
```

```
</resultset>
```

```
select 11, 12 union select 21, 22
for xml option "root=no"
```

```
-----
```

```
<row>
  <C1>11</C1>
  <C2>12</C2>
</row>
```

```
<row>
  <C1>21</C1>
  <C2>22</C2>
</row>
```

```
select forxmlj("select 11, 12 union select 21, 22","root=no")
```

**rowname=sql\_name**

此选项指定“行”元素的名称。缺省 *rowname* 为“row”。*rowname* 选项是 SQL 名称，它可以是常规的标识符或分隔的标识符。分隔的标识符将映射到第 95 页的“将 SQL 名称映射到 XML 名称”中所述的 XML 名称。

下面的示例显示了 *rowname=RowElement*:

```
select 11, 12 union select 21, 22
forxml option "rowname=RowElement"
-----
<resultset xmlns:xsi="http://www.w3.org/2001
  /XMLSchema-instance">

  <RowElement>
    <C1>11</C1>
    <C2>12</C2>
  </RowElement>

  <RowElement>
    <C1>21</C1>
    <C2>22</C2>
  </RowElement>

</resultset>
```

`schemaloc=uri`

此选项指定要作为 *xsi:SchemaLocation* 或 *xsi:noNamespaceSchemaLocation* 属性包含在所生成的 SQLX-XML 文档中的 URI。此选项缺省为空字符串，它指示应该省略模式位置属性。

模式位置属性用作对支持模式的 XML 分析程序的提示。如果您知道将存储对应 SQLX-XML 模式的 URI，请为 SQLX-XML 结果集指定此选项。

如果指定 *schemaloc* 选项时未指定 *targetns* 选项，则 *schemaloc* 将放入 *xsi:noNamespaceSchemaLocation* 属性中，如下例所示：

```
select 1,2
for xml option "schemaloc='http://thiscompany.com/schemalib' "
-----
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://thiscompany.com/schemalib">
  <row>
    <C1>1</C1>
    <C2>2</C2>
  </row>
</resultset>
```

如果指定 *schemaloc* 选项时指定了 *targetns* 选项，则 *schemaloc* 将放入 *xsi:schemaLocation* 属性中，如下例所示：

```
select 1,2
for xml option "schemaloc='http://thiscompany.com/schemalib'
  targetns='http://thiscompany.com/samples'"
-----
<resultset xmlns:xsi="http://www.w3.org/2001
  /XMLSchema-instance"
  xsi:schemaLocation="http://thiscompany.com/schemalib"
  xmlns="http://thiscompany.com/samples">

  <row>
    <C1>1</C1>
    <C2>2</C2>
  </row>

</resultset>
```

**statement={yes | no}** 此选项指定是否将 **statement** 属性包含在 **root** 元素中。如果指定 **root=no**，则将忽略 **statement** 选项。

```
select name_doc from sample_doc
where name_doc like "book%"
for xml option "statement=yes"
-----
<resultset statement="select name_doc
  from sample_docs where name_doc like &quot;book&quot;;"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <name_doc>bookstore</name_doc>
  </row>
</resultset>
```

**tablename=sql\_name** 此选项指定结果集的名称。缺省的 **tablename** 为 “*resultset*”。**tablename** 选项是 SQL 名称，它可以是常规的标识符或分隔的标识符。分隔的标识符将映射到第 95 页的“将 SQL 名称映射到 XML 名称”中所述的 XML 名称。

下面的示例显示了 **tablename=SampleTable**。

```
select 11, 12 union select 21, 22
for xml option "tablename=SampleTable"
-----
<SampleTable xmlns:xsi="http://www.w3.org/2001
  /XMLSchema-instance">

  <row>
    <C1>11</C1>
    <C2>12</C2>
  </row>

  <row>
    <C1>21</C1>
    <C2>22</C2>
  </row>

</SampleTable>
```

**targetns=uri** 此选项指定要作为 **xmlns** 属性包含在所生成的 SQLX-XML 文档中的 URI。此选项缺省为空字符串，它指示应该省略 **xmlns** 属性。有关 **schemaloc** 和 **targetns** 属性之间的交互的说明，请参见 **schemaloc** 属性。

```
select 1,2
for xml
option "targetns='http://thiscompany.com/samples'"
-----
```

```

<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://thiscompany.com/samples">
  <row>
    <C1>1</C1>
    <C2>2</C2>
  </row>
</resultset>

```

`xsidecl={yes | no}`

使用此选项，可以指定是否要声明 XML `xsi` 属性。

例如：

```

select 1 for xml option 'xsidecl=yes'
-----
<resultset
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>1</C1>
  </row>
</resultset>

```

```

select 1 for xml option 'xsidecl=no'
-----
<resultset>
  <row>
    <C1>1</C1>
  </row>

```

对于 `nullstyle=attribute` 中的空值，使用 `xsi` 属性：

```

select null for xml
  option 'nullstyle=attribute xmldecl=yes'

If you specify xsidecl=no or <resultset
  xmlns:xsi="http://www.w3.org/2001
  /XMLSchema-instance">
  <row>
    <C1 xsi:nil="true"/>
  </row>
</resultset>

```

如果指定 `nullstyle=element` 或者 `nullstyle=attribute`，同时您计划在一个较大的、已经包含 `xsi` 属性声明的 XML 文档中嵌入生成的 XML 文档，则可以指定 `xsidecl=no`。

# SQLX 数据映射

本节介绍由 select 语句中的 for xml 子句以及由 forxmlj 函数生成的文档所使用的 SQLX-XML 格式。SQLX-XML 格式按照 ANSI SQLX 标准指定。

## 映射重复列名和未命名列

下面的查询返回两个同名列和三个无名列：

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
title_id title_id
-----
BU2075    MC3021    4,875.00    55,978.78    66,515.54
MC2222    BU1032    5,000.00    40,619.68    81,859.05
MC2222    BU7832    5,000.00    40,619.68    81,859.05
```

当这些数据映射到 XML 时，这些列将成为元素或属性（取决于 *columnstyle* 选项），此类元素和属性必须具有唯一的名称。因此，所生成的 XML 将给重复列名添加整数后缀，并为未命名的列生成唯一的带后缀的名称。例如（使用以上查询）：

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml
-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">

  <row
    <title_id1>BU2075</title_id1>
    <title_id2>MC3021</title_id2>
    <C1>4875.00</C1>
    <C2>55978.78</C2>
    <C3>66515.54</C3>
  </row>

  <row>
    <title_id1>MC2222</title_id1>
    <title_id2>BU1032</title_id2>
```

```

        <C1>5000.00</C1>
        <C2>40619.68</C2>
        <C3>81859.05</C3>
    </row>

    <row>
        <title_id1>MC2222</title_id1>
        <title_id2>BU7832</title_id2>
        <C1>5000.00</C1>
        <C2>40619.68</C2>
        <C3>81859.05</C3>
    </row>

</resultset>

```

如果 XML 为未命名列生成的名称对应于现有列名，则将跳过生成的该名称。在下例中，最后一个未命名列具有显式列名 “C1”，因此不会将 “C1” 用作生成的列名：

```

select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales,t2.price*t2.total_sales as C1
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml

-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">

    <row>
        <title_id1>BU2075</title_id1>
        <title_id2>MC3021</title_id2>
        <C2>4875.00</C2>
        <C3>55978.78</C3>
        <C1>66515.54</C1>
    </row>

    <row>
        <title_id1>MC2222</title_id1>
        <title_id2>BU1032</title_id2>
        <C2>5000.00</C2>
        <C3>40619.68</C3>
        <C1>81859.05</C1>
    </row>

    <row>
        <title_id1>MC2222</title_id1>
        <title_id2>BU7832</title_id2>

```

```

        <C2>5000.00</C2>
        <C3>40619.68</C3>
        <C1>81859.05</C1>
    </row>

</resultset>

```

在以上示例中，为未名列生成的名称采用“C1”、“C2”等诸如此类的形式。这些名称由基名“C”和一个整数后缀组成。可以用 *prefix* 选项指定其它基名。

下面的示例显示了 *prefix='column\_'*：

```

select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml option "prefix=column_"
-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">
    <row>
        <title_id1>BU2075</title_id1>
        <title_id2>MC3021</title_id2>
        <column_1>4875.00</column_1>
        <column_2>55978.78</column_2>
        <column_3>66515.54</column_3>
    </row>

    <row>
        <title_id1>MC2222</title_id1>
        <title_id2>BU1032</title_id2>
        <column_1>5000.00</column_1>
        <column_2>40619.68</column_2>
        <column_3>81859.05</column_3>
    </row>

    <row>
        <title_id1>MC2222</title_id1>
        <title_id2>BU7832</title_id2>
        <column_1>5000.00</column_1>
        <column_2>40619.68</column_2>
        <column_3>81859.05</column_3>
    </row>

</resultset>

```



## 将 SQL 名称映射到 XML 名称

SQL 表和结果集的 SQLX 表示形式将 SQL 名称用作 XML 元素和属性的名称。但是，SQL 名称可以包含在 XML 名称中无效的各种字符。特别是，SQL 名称包含“分隔的”标识符，即加引号的名称。分隔的标识符可以包含任意字符，如空格和标点符号。例如：

```
"salary + bonus: "
```

是有效的 SQL 分隔标识符。因此，SQLX 标准指定了此类字符到有效 XML 名称字符的映射。

SQLX 名称映射的目标是：

- 处理所有可能的 SQL 标识符
- 确保存在可重新生成初始标识符的逆向映射

SQLX 名称映射以字符的 Unicode 表示形式为基础。SQLX 名称映射的基本约定是具有以下 Unicode 表示形式的无效字符：

```
U+nnnn
```

替换为以下形式的字符串：

```
_xnnnn_
```

无效名称字符的 SQLX 映射给由 4 个十六进制位组成的 Unicode 表示形式添加以下前缀：

```
_x
```

并给它们添加下划线后缀。

例如，假设有以下 SQL 结果集：

```
set quoted_identifier on
select 1 as "a + b < c & d", 2 as "<a xsi:nil="true">"
-----

a + b < c & d <a xsi:nil="true">
-----
1                                2
```

此示例中的选择列表指定常量值（1 和 2），并使用 **as** 子句指定这些值的列名。这些列名是分隔的标识符，它们包含在 XML 名称中无效的字符。

该结果集的 SQLX 映射如下所示：

```
set quoted_identifier on
select 1 as "a + b < c & d", 2 as "<a xsi:nil="true">"
for xml
-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">

<row>
<a_x0020__x002B__x0020_b_x0020__x003C__x0020_c_x0020__x0026__x0020_d_x0020_>
1
</a_x0020__x002B__x0020_b_x0020__x003C__x0020_c_x0020__x0026__x0020_d_x0020_>
<_x003C_a_x0020_xsi_x003A_nil_x003D__x0022_true_x0022__x003E_>
2
</_x003C_a_x0020_xsi_x003A_nil_x003D__x0022_true_x0022__x003E_></row>

</resultset>
```

所得的 SQLX 结果集并不便于阅读，但 SQLX 映射主要供应用程序使用。`_xnnnn_` 约定能解决大多数 SQLX 名称映射问题。

但是，另外一项要求是 XML 名称不能以字母“XML”（包括其任何大小写字母组合形式）开头。因此，SQLX 名称映射指定此类名称中的前导“x”或“X”被替换为值 `_xnnnn_`。由于替代词首的“X”即足以掩饰短语“XML”，因此不更改“M”和“L”（无论大小写）。

例如：

```
select 1 as x, 2 as X, 3 as X99, 4 as xML, 5 as xMLdoc
forxml
-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">

<row>
  <x>1</x>
  <X>2</X>
  <X99>3</X99>
  <_x0078_ML>4</_x0078_ML>
  <_x0058_mLdoc>5</_x0058_mLdoc>
</row>

</resultset>
```

在 SQL 名称映射到 XML 名称方面的要求也适用于在 *tablename*、*rowname* 和 *prefix* 选项中指定的 SQL 名称。例如：

```
select 11, 12 union select 21, 22
for xml option "tablename='table @ start' rowname=' row & columns '
              prefix='C '"
-----
<table_x0020__x0040__x0020_start xmlns:xsi="http://www.w3.org/2001
    /XMLSchema-instance">

<_x0020_row_x0020__x0026__x0020_columns_x0020_>
  <C_x0020_1>11</C_x0020_1>
  <C_x0020_2>12</C_x0020_2>
</_x0020_row_x0020__x0026__x0020_columns_x0020_>

<_x0020_row_x0020__x0026__x0020_columns_x0020_>
  <C_x0020_1>21</C_x0020_1>
  <C_x0020_2>22</C_x0020_2>
</_x0020_row_x0020__x0026__x0020_columns_x0020_>

</table_x0020__x0040__x0020_start>
```

## 将 SQL 值映射到 XML 值

SQL 结果集的 SQLX 表示形式将列的值映射到表示这些列的 XML 属性或元素的值。

### 数值

数值数据类型在 SQLX 映射中表示为字符串文字。例如：

```
select 1, 2.345, 67e8 for xml
-----
<resultset xmlns:xsi="http://www.w3.org/2001
    /XMLSchema-instance">

  <row>
    <C1>1</C1>
    <C2>2.345</C2>
    <C3>6.7E9</C3>
  </row>

</resultset>
```

## 字符值

char、varchar 或 text 列中包含的字符值需要进行附加的处理。SQL 数据中的字符值可以包含那些在 XML 中具有特殊意义的字符：引号 (")、撇号 (')、小于号 (<)、大于号 (>) 和 “与” 符号 (&) 字符。当 SQL 字符值表示为 XML 属性或元素值时，必须替换为表示它们的 XML 实体：@quot;、&apos;、&lt;、&gt; 和 &amp;。

下例显示包含 XML 标记字符的 SQL 字符值。采用关于内嵌引号和撇号的 SQL 约定，SQL select 命令中的字符文字会将撇号加倍。

```
select '<name>"Baker'"s"</name>'
-----
<name>"Baker'"s"</name>
```

下例显示该字符值的 SQLX 映射，其中 XML 标记字符替换为它们的 XML 实体表示形式。forxmlj 函数中的字符文字参数将内嵌的引号加倍。

```
select '<name>"Baker'"s"</name>' for xml
-----
<resultset xmlns:xsi="http://www.w3.org/2001
    /XMLSchema-instance">

    <row>
    <C1>&lt;name&gt;&quot;Baker&apos;s&quot;&lt;/name&gt;&lt;
        /C1>
    </row>

</resultset>
```

## 二进制值

binary、varbinary 或 image 列中包含的二进制值以 hex 或 base64 编码来表示，这取决于选项 *binary={hex|base64}*。base64 编码更为紧凑。这两种表示形式之间的选择取决于处理 XML 数据的应用程序。

请参见第 82 页的“SQLX 选项”中的示例。

## SQLX 模式映射

`forxmlschemaj` 函数和 `forxmlalj` 函数会生成一个 XML 模式，该模式描述了指定结果集的 SQLX-XML 文档。本节概述这种生成的 XML 模式。这些 XML 模式通常只由 XML 工具使用，因此您不需要详细了解每一行的含义。

### 概述

以下 SQL 结果集包含 5 列，其数据类型分别为 `varchar(3)`、`numeric(3,1)`、`varbinary(2)`、`numeric(3,1)` 和 `numeric(3,2)`。

```
select 'abc', 12.3, 0x00, 45.6, 7.89
-----
abc    12.3 0x00  45.6  7.89
```

上述数据的 SQLX-XML 结果集为：

```
select forxmlj("select 'abc', 12.3, 0x00, 45.6, 7.89", "")
-----
<resultset xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">
  <row>
    <C1>abc</C1>
    <C2>12.3</C2>
    <C3>00</C3>
    <C4>45.6</C4>
    <C5>7.89</C5>
  </row>
</resultset>
```

描述此文档的 SQLX-XML 模式为：

```
select forxmlschemaj("select 'abc', 12.3, 0x00, 45.6, 7.89", "")
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
    schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

  <xsd:complexType name="RowType.resultset">
    <xsd:sequence>
```

```
<xsd:element name="C1" type="VARCHAR_3" />
<xsd:element name="C2" type="NUMERIC_3_1" />
<xsd:element name="C3" type="VARBINARY_2" />
<xsd:element name="C4" type="NUMERIC_3_1" />
<xsd:element name="C5" type="NUMERIC_3_2" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.resultset">
  <xsd:sequence>
    <xsd:element name="row" type="RowType.resultset"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="VARCHAR_3">
  <xsd:restriction base="xsd:string">
    <xsd:length value="3"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="NUMERIC_3_1">
  <xsd:restriction base="xsd:decimal">
    <xsd:totalDigits value="3"/>
    <xsd:fractionDigits value="1"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="VARBINARY_2">
  <xsd:restriction base="xsd:hexBinary">
    <xsd:length value="2"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="NUMERIC_3_2">
  <xsd:restriction base="xsd:decimal">
    <xsd:totalDigits value="3"/>
    <xsd:fractionDigits value="2"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>
```

此 XML 模式有五个组成部分：

- 此示例 XML 模式的最后一个部分是三个 *xsd:simpleType* 元素，它们为 XML 文档中的四个不同数据类型声明简单的 XML 类型。这些 *simpleType* 声明指定每种类型的 XML 基类型，并指定定义 SQL 数据长度特性的 *xsd:restriction* 元素。每个 *simpleType* 声明都有一个 XML 名称：VARCHAR\_3、NUMERIC\_3\_1、VARBINARY\_2 和 NUMERIC\_3\_2。
- 对于 SQL 数据类型、长度和精度的每种不同属性组合，该 XML 模式都包含一个单独的 *xsd:simpleType*。例如，对于 NUMERIC\_3\_1 和 NUMERIC\_3\_2，存在不同的类型。但是对于 NUMERIC\_3\_1，即使有两个该类型的列，仍只有一个 *xsd:simpleType* 声明。这些列的元素声明均引用相同的简单类型名 NUMERIC\_3\_1。
- 示例 XML 模式的第一部分是 row 类型的 *xsd:complexType*，它为每个列定义一个元素。其中的每个元素声明指定具有上述简单类型名的元素的数据类型。
- 示例 XML 模式的中间部分是结果集的 *xsd:complexType*，此命令将其声明为一系列其类型为先前定义的行类型的 row 元素。
- 最后，示例 XML 模式的最后一行声明结果集文档的根元素。

## 选项：columnstyle=element

采用 *columnstyle=element* 时生成的 XML 模式的格式将列指定为 rowtype 声明的 XML 元素。例如：

```
select forxmlschema j("select 1,2", "columnstyle=element")
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">
<xsd:import namespace="http://www.w3.org/2001/XMLSchema"
  schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

  <xsd:complexType name="RowType.resultset">
    <xsd:sequence>
      <xsd:element name="C1" type="INTEGER" />
      <xsd:element name="C2" type="INTEGER" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TableType.resultset">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
```

```
        minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:integer">
        <xsd:maxInclusive value="2147483647"/>
        <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>
```

## 选项: *columnstyle=attribute*

采用 *columnstyle=attribute* 时生成的 XML 模式的格式类似于采用 *columnstyle=element* 时的 XML 模式。唯一的不同在于列被指定为 *rowtype* 声明的 XML 属性。例如:

```
select forxmlschema('select 1,2', "columnstyle=attribute")
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

<xsd:import namespace="http://www.w3.org/2001/XMLSchema"
  schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

  <xsd:complexType name="RowType.resultset">

    <xsd:attribute name="C1" type="INTEGER" use="required"/>
    <xsd:attribute name="C2" type="INTEGER" use="required"/>

  </xsd:complexType>

  <xsd:complexType name="TableType.resultset">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="INTEGER">
```



```

    <xsd:restriction base="xsd:integer">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>

```

## 选项：nullstyle=omit

采用 *nullstyle=omit* 时生成的 XML 模式的格式在每个可为空值的列声明中指定 *minOccurs="0"* 和 *maxOccurs="1"* 属性。例如：

```

select forxmlschema j("select 1,null", "nullstyle=omit")
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
    schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

  <xsd:complexType name="RowType.resultset">
    <xsd:sequence>
      <xsd:element name="C1" type="INTEGER" />
      <xsd:element name="C2" type="INTEGER"
        minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TableType.resultset">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:integer">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>

```

```
<xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>
```

## 选项: *nullstyle=attribute*

采用 *nullstyle=attribute* 时生成的 XML 模式的格式在每个可为空值的列声明中指定 *nullable="true"* 属性。例如:

```
select forxmlschema j("select 1,null", "nullstyle=attribute"
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
    schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

  <xsd:complexType name="RowType.resultset">
    <xsd:sequence>
      <xsd:element name="C1" type="INTEGER" />
      <xsd:element name="C2" type="INTEGER" nullable="true"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TableType.resultset">
    <xsd:sequence>
      <xsd:element name="row" type="RowType.resultset"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:integer"
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>
```

# 对 I18N 提供的 XML 支持

本章介绍用于支持非 ASCII 数据的 XML 服务扩展。支持用于指定 Unicode 库的 XML 标准以及跨多种语言支持基于 XML 的应用程序都需要该扩展。

在此文档中，术语“**I18N**”代表国际化，该术语以“**I**”开头，后跟 18 个字符，最后以“**n**”结束。此术语表示对 ASCII 字符集以外的 Unicode 字符和其它字符的支持。

主题	页码
<a href="#">概述</a>	<a href="#">105</a>
<a href="#">for xml 中的 I18N</a>	<a href="#">107</a>
<a href="#">xmlparse 中的 I18N</a>	<a href="#">113</a>
<a href="#">xmlextract 中的 I18N</a>	<a href="#">114</a>
<a href="#">xmlvalidate 中的 I18n</a>	<a href="#">116</a>

## 概述

I18N 扩展分为三类：

- `for xml` 子句中的 I18N 支持，用于生成包含非 ASCII 数据的文档。
- `xmlparse` 中的 I18N，用于存储包含非 ASCII 数据的文档。
- `xmlextract` 和 `xmltest` 中的 I18N，用于处理包含非 ASCII 数据的 XML 文档和查询。

## Unicode 数据类型

以下术语表示用于 Unicode 的数据类型的类别：

- “字符串数据类型”是指 `char`、`varchar`、`text` 和 `java.lang.String`。
- “Unicode 数据类型”是指 `unichar`、`univarchar`、`unitext` 和 `java.lang.String`。

- “字符串 /Unicode 列”是指数据类型为“字符串数据类型”或“Unicode 数据类型”的列。

## 代理对

“代理对”是指一个由 16 位值组成的对，由 Unicode 用来表示需要的位数可能多于 16 的任意字符。

大多数字符都可在 [0x20, 0xFFFF] 范围内表示，而且可用单个 16 位值来表示。代理对是一个 16 位值对，用来表示 [0x010000..0x10FFFF] 范围中的字符。有关详细信息，请参见第 112 页的“示例 7”。

## 数值字符表示

数值字符表示 (NCR) 是一种用于表示 XML 文档中以 ASCII 十六进制表示法表示的任意字符的方法。例如，欧元符号“€”的 NCR 表示形式为“&#x20AC;”。这种表示法类似于 SQL 十六进制字符表示法 `u&'\20ac'`。

## 客户端 / 服务器转换

服务器中的 Unicode 数据可以是：

- UTF-16 数据，存储在 `unichar`、`univarchar`、`unitext` 和 `java.lang.String` 中。
- UTF-8 数据，在服务器字符集为 UTF-8 时，存储在 `char`、`varchar` 和 `text` 中。

**在客户端和服务器之间传送数据** 以下三种方法中的任何一种都可以在客户端和服务器之间传送 `univarchar` 或 `unitext` 数据：

- 使用 CTLIB 或 BCP。将数据作为位字符串传送。客户端数据为 UTF-16；会针对客户端 / 服务器差异调整字节顺序。
- 使用 ISQL 或 BCP。指定“-J UTF-8”。在客户端 UTF-8 和服务器 UTF-16 之间转换数据。

- 使用 **Java**。指定数据传送中的客户端字符集（源或目标）。您可以指定 UTF-8、UTF-16BE、UTF-16L、UTF-16LE、UTF-16（带 BOM）、US-ASCII 或其它客户端字符集。

---

**注释** 如果要通过 JDBC 存储 Unicode XML 文档，则必须纳入连接属性 “DISABLE\_UNICODE\_SENDING”，此属性是一个 “false” 属性，用于将 Unicode 数据从 JDBC 连接发送到 Adaptive Server。

---

指定客户端 **Java** 应用程序中客户端文件（不论是输入还是输出）字符集的方法显示在以下示例目录中的 **Java** 应用程序中。

`$SYBASE/$SYBASE_ASE/sample/JavaXml/JavaXml.zip`

此目录还提供了文档 Using-SQLX-mappings 中的 “Unicode 和 SQLX 结果集文档” 一节。

## 字符集和 XML 数据

如果您将 XML 文档存储在字符串列或变量中，XML 服务则假定该文档位于服务器字符集中。如果您将 XML 文档存储在 Unicode 列或变量中，XML 服务则假定其为 UTF-16。XML 文档中的任何编码子句都将被忽略。

## for xml 中的 I18N

本节讨论如何扩展 **for xml** 子句以处理非 ASCII 数据。

您可以在 **for xml** 子句的 *select\_list* 中指定包含非 ASCII 字符的 Unicode 列和字符串列。

**returns** 子句中的缺省数据类型为 **text**。

结果 XML 文档将以 Unicode 字符串的形式在内部生成，如有必要，结果 XML 文档还将被转换为 **returns** 子句的数据类型。

有关此子句的详细文档，请参见第 55 页的 “for xml 子句”。

## 选项字符串

for xml 子句的选项字符串可以指定 **u&** 格式的文字，并包含字符的 SQL 符号。这样您就可以为 **rowname**、**tablename** 和 **prefix** 选项指定 Unicode 字符。例如，可输入：

```
select * from T
for xml
options u'tablename = \0415\0416 rowname =
        \+01d6d prefix = \0622'
```

如果指定的 **tablename**、**rowname** 或 **prefix** 选项包含了在简单标识符中无效的字符，您必须将该选项指定为带引号的标识符。例如，可输入：

```
select * from T
for xml
options u'tablename = "chars\0415 and \0416"
        rowname = "\+01d6d1 & \+01d160"
        prefix = "\0622-"
```

## for xml 中的数值字符表示

select for xml 语句的 *option\_string* 包括一个 **ncr** 选项，此选项指定字符串列和 Unicode 列的表示形式：

```
ncr = {no | non_ascii | non_server}
```

- **ncr = no** 指定用明文值表示字符串列和 Unicode 列。这些明文值是否实体化，取决于 **entitize** 选项。
- **ncr = non\_ascii** 和 **ncr = non\_server** 指定用 NCR 表示非 ASCII 的字符串列和非缺省服务器字符集的 Unicode 列。任何未转换为 NCR 的字符是否实体化，取决于 **entitize** 选项。

for xml 子句中的缺省 NCR 选项为 **ncr = non\_ascii**。

**ncr** 选项仅应用于列值，而不应用于列名或在 **tablename**、**rowname** 或 **prefix** 选项中指定的名称。XML 不允许出现以元素名或属性名表示的 NCR。

## header 选项

for xml 子句的 header 选项是用一个新编码值扩展的：

```
header = {yes | no| encoding}
```

当 header=encoding 时，标头为：

```
<?xml version = "1/0" encoding = "UTF-16?">
```

使用 encoding 值表示应包含 XML 标头，且 XML 标头应包含 XML 编码声明。

如果满足下列情况，缺省的 header 选项为 no：

- returns 数据类型为 Unicode 数据类型
- ncr 选项为 *non-ascii*
- 服务器字符集为 ISO1、ISO8859\_15、ascii\_7 或 UTF-8。

如果不是这三种情况，缺省的 header 选项为 encoding。

## 异常

无。

## 示例

以下所有示例均使用由下面的命令生成的示例表。

```
create table example_I18N_table (name varchar(10) null,
    uvcol univarchar(10) null)
-----
insert into example_I18N_table values('Arabic',
    u&'\622\623\624\625\626')

insert into example_I18N_table values('Hebrew',
    u&'\5d2\5d3\5d4\5d5\5d6')

insert into example_I18N_table values('Russian',
    u&'\410\411\412\413\414')
```

图 6-1 中的示例表包含两列：

- 表示一种语言的 **varchar** 列。
- 包含此种语言的示例字符的 **univarchar** 列。示例字符由字符串组成，字符串由连续的字母构成。

```
select * from example_I18N_table
name          uvcol
-----
Arabic        0x06220623062406250626
Hebrew        0x05d205d305d405d505d6
Russian       0x04100411041204130414

(3 rows affected)
```

## 示例 1

未指定任何变量时，**select** 命令显示下表：

```
select * from example_I18N_table
name          uvcol
-----
Arabic        0x06220623062406250626
Hebrew        0x05d205d305d405d505d6
Russian       0x04100411041204130414
3 rows affected)
```

## 示例 2

要用 **for xml** 子句生成 SQL XML 文档，请输入：

```
select * from example_I18N_table for xml
-----
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <name>Arabic</name>
    <uvcol>&#x622;&#x623;&#x624;&#x625;&#x626;</uvcol>
  </row>
  <row>
    <name>Hebrew</name>
    <uvcol>&#x5d2;&#x5d3;&#x5d4;&#x5d5;&#x5d6;</uvcol>
  </row>
  <row>
    <name>Russian</name>
    <uvcol>&#x410;&#x411;&#x412;&#x413;&#x414;</uvcol>
  </row>
</resultset>
```



### 示例 3

缺省情况下，生成的 SQLX XML 文档使用 NCR 表示非 ASCII 字符。如果将浏览器的字符集属性设为 Unicode，该文档将显示实际的非 ASCII 字符（分别为 Arabic、Hebrew 或 Russian）或您选择的任何非 ASCII 字符。

如果未将浏览器的字符集属性设为 Unicode，Arabic、Hebrew 和 Russian 字符将显示为问号。

### 示例 4

如果想要使 SQLX XML 文档包含明文字符形式的非 ASCII，请在 ncr 选项中指定 *no*。

```
select * from example_I18N_table for xml
      option 'ncr=no' returns unitext
-----
0x000a003c0072006500730075006c007400730065007400200078006d...etc
```

### 示例 5

如果将示例 3 中生成的 Unicode 文档检索到客户端文件中，同时指定 UTF-16 或 UTF-8 作为目标字符集，您便可以在浏览器中显示此文档。此时它将显示您选择的实际的非 ASCII 字符。

### 示例 6

只有当字符不为 ASCII 或不在缺省服务器字符集中时，ncr 中的选项 *ncr=non\_ascii* 和 *ncr=non\_server* 才将字符转换为 NCR。在此示例中，表达式将并置带有 ASCII name 列和 Unicode uvcol 列的 ASCII 字符串值。此表达式的结果为包含 ASCII 和非 ASCII 字符的字符串。生成的 SQLX XML 文档中仅将非 ASCII 字符转换为 NCR：

```
select name + '(' + uvcol + ')' from example_I18N_table2>
      for xml option 'ncr=non_ascii'
-----
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>Arabic(&#x622;&#x623;&#x624;&#x625;&#x626;)</C1>
  </row>
  <row>
    <C1>Hebrew(&#x5d2;&#x5d3;&#x5d4;&#x5d5;&#x5d6;)</C1>
  </row>
  <
row>
```

```
<C1>Russian(&#x410;&#x411;&#x412;&#x413;&#x414;)</C1>
</row>
</resultset>
```

浏览器将显示带有实际的非 ASCII 字符（分别为 Arabic、Hebrew 和 Russian）的文档。

## 示例 7

大多数字符都由 [0x20, 0xFFFF] 范围中的代码点表示，并可用单个 16 位值表示。代理对是一个 16 位值对，用来表示 [0x010000..0x10FFFF] 范围中的字符。该代理对的前半部分位于 [0xD800..0xDBFF] 范围中，后半部分位于 [0xDC00..0xDFFF] 范围中。这样的值对 (H, L) 表示按以下方式计算的字符（十六进制算术运算）：

$$(H - 0xD800) * 400 + (L - 0xDC00)$$

例如，字符 “&#x01D6D1” 是小写粗体数学符号，由代理对 D835, DED1 表示：

```
select convert(unitext, u&'\+1d6d1')
-----
0xd835ded1
```

当您指定 *ncr=non\_ascii* 或 *ncr=non\_server* 以生成包含非 ASCII 数据（带有代理对字符）的 SQLX XML 文档时，代理对显示为单个 NCR 字符，而不是对：

```
select convert(unitext, u&'\+1d6d1')
for xml option 'ncr=non_ascii'
-----
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>&#x1d6d1;</C1>
  </row>
</resultset>
```

## ***xmlparse* 中的 I18N**

*xmlparse* 支持将 Unicode 数据类型（`unichar`、`univarchar`、`unitext` 和 `java.lang.String`）用于输入 XML 文档。

### **选项**

*xmlparse* 分析 XML 文档并以包含分析的文档及其内部索引的 `image` 值的形式来返回其表示形式。此表示形式称为 Unicode 分析的图像 XML。*Unicode 分析的图像 XML* 存储在 `image` 列中。

*xmlparse* 可将字符串数据类型转换为 Unicode。由于字符串数据类型位于服务器字符集（服务器字符集始终为 Unicode 的子集）中，因此转换是一种永远不会引发转换异常的数据类型更改。

### ***xmlparse* 中的排序顺序**

有关 XML 排序顺序的详细信息，请参见第 115 页的“XML 服务中的排序顺序”。

*xmlparse* 使用 `sp_configure` 选项 `default xml sort order` 指定的排序顺序和 XML 索引的顺序。XML 将排序顺序名称存储在由 *xmlparse* 生成的 `image` 中，此名称称为文档的经过分析的 XML 排序顺序。

当已分析的 XML 排序顺序不同于当前缺省的 XML 排序顺序时，所有引用已分析的 XML 文档的函数都将引发异常。

## xmlextract 中的 I18N

xmlextract 将 XML 查询表达式应用于 XML 文档，并返回您选择的结果。输入文档可以是包含字符数据或已分析的 XML 的 `string` 数据类型、Unicode 数据类型或 `image` 数据类型。

`returns` 子句可以指定 Unicode 数据类型作为提取的值的的数据类型。

## NCR 选项

xmlextract 支持下面的 `ncr` 选项：

```
ncr = {non_ascii|non_server|no}
```

运行期间，如果符合下列情况，则应用 `ncr` 选项：

- 结果数据类型是字符串或 Unicode 数据类型，而不是其它数据类型（例如，`numeric`、`datetime` 或 `money`）。
- XPath 查询没有指定 `text()`。

缺省的 `ncr` 选项是：

- 如果 `returns` 数据类型是 Unicode 数据类型，则缺省值为 `ncr=no`。
- 如果 `returns` 数据类型是字符串数据类型，则缺省值为 `ncr=non_server`。

## xmlextract 中的排序顺序

[第 115 页的“XML 服务中的排序顺序”](#)中讨论了 xmlextract 中的排序顺序。

xmlextract 使用存储在输入 XML 文档中的经过分析的 XML 排序顺序，而不是服务器中当前的缺省排序顺序。

## XML 服务中的排序顺序

**sp\_configure 选项** XML 服务定义了 `sp_configure` 选项 `default xml sort order`，此选项具有三个显著特性：

- 此选项是静态的；您必须重新启动 Adaptive Server 以执行此配置。
- 此选项值是 Unicode 排序顺序的名称。有关详细信息，请参见《系统管理指南，卷 1》中的“缺省的 Unicode 排序顺序”表。
- 缺省选项值为 *binary*。

**xmlparse** `xmlparse` 返回经过分析的参数文档表示形式，其中包括文档的元素和属性的索引以及它们的值。当分析文档时，经过分析的表达式指定实际存在的缺省 *xml* 排序顺序。

**xmlextract** `xmlextract` 对用于比较条件（如“`//book[author='John Doe']`”）的 XPath 查询求值。`xmlextract` 将当前的缺省 *xml* 排序顺序与文档的经过分析的 *xml* 排序顺序进行比较。如果二者不同，`xmlextract` 将引发异常。

`xmlextract` 使用存储在输入 XML 文档中的 XML 排序顺序，而不是当前服务器中的缺省排序顺序。

---

**注释** XML 服务使用单个缺省顺序 `default xml sort order`。它不会同时使用 `default Unicode xml sort order` 和 `default xml sort order`。

---

**修改缺省的 xml 排序顺序** 您可以用 `sp_configure` 修改缺省的 *xml* 排序顺序。

修改了缺省的 *xml* 排序顺序之后，您可以使用 Adaptive Server 的 `update` 命令重新分析以前分析过的 XML 文档。有关 `update` 的信息，请参见《参考手册》第二卷《命令》。

```
update xmldocs
set doc = xmlparse(xmlextract('/', doc))
```

## xmlvalidate 中的 l18n

xmlvalidate() 支持 Unicode 数据类型 unichar、univarchar、unitext 和 java.lang.String，以及 string 和 image 数据类型。xmlvalidate 的 returns 子句可以指定 Unicode 数据类型作为提取的值的的数据类型。

## NCR 选项

xmlvalidate() 支持下面的 ncr 选项：

```
ncr={non_ascii | non_server | no}
```

在运行期间，仅当 result 子句的数据类型为 string 或 Unicode 数据类型时，ncr 选项才会应用。例如，此选项不适用于 numeric、datetime 和 money 数据类型。

### 缺省 NCR 选项

- 如果 returns 数据类型为以下某一 Unicode 数据类型，则缺省的 NCR 选项值为 ncr=no：unichar、univarchar、unitext 或 java.lang.String。
- 如果 returns 数据类型是以下某一 string 数据类型，则缺省的 NCR 选项值为 ncr=non\_server：char、varchar 或 text。

本章详细介绍 xmltable() 函数。

主题	页码
简介	117
xmltable 及派生表 的语法	117

简介

xmltable() 用于从 XML 文档中提取一系列多值元素，并将这些元素组合成一个 SQL 表。对 xmltable() 的单个调用可取代在每次迭代时对 xmlextract 进行多次调用的 T-SQL 循环。此函数作为派生表（在另一个 SQL 查询的 from 子句中指定的带括号的子查询）进行调用。调用 xmltable() 相当于对由 xmltable() 生成的表的每行执行单个 xmlextract 表达式。

xmltable() 是 xmlextract 的一般形式。这两个函数都返回从作为该函数中参数的 XML 文档中提取的数据。不同之处在于：

- xmlextract 返回由单个 XPath 查询识别到的数据。
- xmltable() 提取由 XPath 查询识别到的数据的序列或行模式，并从该序列的每个元素中提取由一系列其它 XPath 查询识别到的数据（列模式）。它以一个 SQL 表的形式返回所有数据。

xmltable 及派生表的语法

下面的语法部分说明了 xmltable() 的基本语法，以及在何处使用和如何使用 xmltable()。

## xmltable

说明

从 XML 文档中提取数据，并以 SQL 表的形式返回该数据。

语法

```
xmltable_expression ::= xmltable
    ( row_pattern passing xml_argument
      columns column_definitions
      options_parameter )
row_pattern ::= character_string_literal
xml_argument ::= xml_expression
column_definitions ::=
    column_definition [ { , column_definition } ]
column_definition ::=
    ordinality_column | regular_column
ordinality_column ::= column_name datatype for ordinality
regular_column ::=
    column_name datatype [ default_literal ] [ null | not null ]
[ path column_pattern ]
column_pattern ::= character_string_literal
options_parameter ::= [, ] option option_string
options_string ::= basic_string_expression
```

**派生表语法** 从 SQL from 子句返回一个 SQL 表。

```
from_clause ::= from table_reference [ , table_reference ] ...
table_reference ::= table_view_name | ANSI_join | derived_table
table_view_name ::= 请参见 《参考手册》第二卷 《命令》中的
                    select 命令。
ANSI_join ::= 请参见 《参考手册》第二卷 《命令》中的
              select 命令。
derived_table ::=
    (subquery) as table_name [ (column_name [ , column_name ] ...)
    xmltable_expression as table_name
```

示例

**作为派生表的 xmltable** 下面的示例显示一个简单的 xmltable() 调用，此调用返回一个派生表。

```
select * from xmltable('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
           + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name') as items_table
id          name
-----
1           Box
2           Jar

(2 rows affected)
```



**示例 1** 派生表的语法要求您指定表名 (items\_table)，即使您不引用该表也不例外。例如，下面的示例是不正确的。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
           + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name')
-----
Msg 102 Level 15, State 1:
Incorrect syntax near ')'
```

**简单的文档引用示例** 在文档引用中，紧跟 passing 的参数是输入 XML 文档。在下面的示例中，以字符串文字的形式指定文档。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
           + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name') as items_table

id          name
-----
1           Box
2           Jar

(2 rows affected)
```

**示例 2** 下面的示例显示将文档存储在一个 T-SQL 变量中，并在 xmltable() 调用中引用该变量。

```
declare @doc varchar(16384)
set @doc='<doc><item><id>1</id><name>Box</name></item>'
        + '<item><id>2</id><name>Jar</name></item></doc>'

select * from xmltable('/doc/item' passing @doc
    columns id int path 'id', name varchar(20) path 'name') as items_table

id          name
-----
1           Box
2           Jar

(2 rows affected)
```

**示例 3** 将文档存储在一个表中，并使用一个标量子查询引用它：

```
select 100 as doc_id,
'<doc><item><id>1</id><name>Box</name></item><item><id>2</id>
<name>Jar</name></item></doc>' as doc
into #sample_docs
```

```
select* from xmltable('/doc/item'
    passing(select doc from #sample_docs where doc_id=100)
    columns id int path 'id',name varchar(20) path 'name') as items_table
```

id	name
1	Box
2	Jar

(2 rows affected)

**行模式** `xmltable` 调用中的第一个参数 *row-pattern* ('/doc/item') 是一个 XPath 查询表达式，其结果是来自指定文档的一个元素序列。`xmltable` 调用返回一个表，该序列中的每个元素在该表中都有对应的一行。

**示例 4** 如果行模式返回一个空序列，则结果是一个空表：

```
select * from xmltable ('//item_entry'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id',
    name varchar(20) path 'name') as items_table
```

id	name
----	------

(0 rows affected)

**示例 5** 行模式表达式不能为 XPath 函数：

```
select * from xmltable ('/doc/item/tolower()'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id',
    name varchar(20) path 'name') as items_table
```

id	name
----	------

Msg 14825, Level 16, State 0:

Line1:

XPath function call must be at leaf level.

**列模式** 紧跟 `columns` 关键字的参数是一个列定义列表。每个列定义都指定一个列名称和一个数据类型（如同在 `create table` 中一样），以及一个称作列模式的路径。*column-pattern* 是一个 XPath 查询表达式，它适用于由 *row-pattern* 返回的序列的元素，用于提取结果表中某列的数据。

**示例 6** 如果某列的数据包含在 XML 属性中，请通过使用 “@” 引用属性来指定相应的列模式。例如：

```
select * from xmltable ('/doc/item'
    passing '<doc><item id="1"><name>Box</name></item>'
           + '<item id="2">/id><name><Jar</name></item></doc>'
    columns id int path '@id', name varchar(20)) as items_table
```

id	name
1	Box
2	Jar

(2 rows affected)

**缺省列模式** *column-pattern* 通常与指定的 *column\_name* 相同，例如 *name* 就是这样。在这种情况下，省略 *column-pattern* 会导致缺省为 *column\_name*：

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
           + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int, name varchar(20)) as items_table
```

id	name
1	Box
2	Jar

(2 rows affected)

**示例 7** 如果希望列模式缺省为列名称，请在其值位于 XML 属性中的列中使用带引号的标识符。之后，在结果中引用此类标识符时，必须为它们加上引号：

```
set quoted_identifier on
select "@id", name from xmltable ('/doc/item'
    passing '<doc><item id="1"><name>Box</name></item>'
           + '<item id="2"><name>Jar</name></item></doc>'
    columns "@id" int, name varchar(20)) as items_table
```

@id	name
1	Box
2	Jar

(2 rows affected)

**示例 8** 您还可以使用带引号的标识符指定列名称作为缺省列模式，所采用的列名称可以为较复杂的 XPath 表达式。例如：

```
set quoted_identifier on
select "@id", "name/short", "name/full" from xmltable ('/doc/item'
  passing '<doc><item id="1"><name><short>Box</short>
        <full>Box, packing, moisture resistant, plain</full>
        </name></item>'
  + '<item id="2"><name><short>Jar</short>
        <full>Jar, lidded, heavy duty</full>
        </name></item></doc>'
  columns "@id" int, "name/short" varchar(20), "name/full" varchar(50))
as items_table
```

@id	name/short	name/full
1	Box	Box, packing, moisture resistant, plain
2	Jar	Jar, lidded, heavy duty

(2 rows affected)

**隐含的 text()** 下面的示例说明了通常隐含在列模式中的 text() 函数。text() 用于删除 XML 元素标记。例如，下面的 XPath 查询返回的所选元素带有 XML 标记：

```
1> declare @doc varchar(16384)
2> set @doc= '<doc><item><id>1</id></name>Box</name></item>'
        + '<item><id>2</id><name>Jar</name></item></doc>'
3> select xmlextract('/doc/item[2]/name', @doc)
-----
<name>Jar</name>
```

### 示例 9

将 text() 添加到 XPath 查询中可删除 XML 标记：

```
1> declare @doc varchar(16384)
2> set @doc= '<doc><item><id>1</id></name>Box</name></item>'
        + '<item><id>2</id><name>Jar</name></item></doc>'

3> select xmlextract('/doc/item[2]/name/text()', @doc)
-----
Jar
```

**示例 10** 大多数列模式中都隐含了 text()。下面的示例在 id 和 name 列的列模式中均未指定 text():

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
           + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name') as items_table

id          name
---          -
1           Box
2           Jar

(2 rows affected)

)
```

**数据类型转换** 可以通过对从列模式中提取的数据应用隐式 SQL convert 语句，在数据类型转换过程中派生列值。例如：

```
select * from xmltable ('/emps/emp'
    passing '<emps>
<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp>
+<emp><id>2</id><salary>234.56</salary><hired>2/3/2004</hired></emp>'
           + '</emps>'
    columns id int path 'id', salary dec(5,2), hired date)
as items_table

id          salary      hired
-----
1           123.45      Jan 2, 2003
2           234.56      Feb 3, 2004

(2 rows affected)
```

**示例 11** 为列提取的 XML 数据必须能够转换为列数据类型，否则会引起异常：

```
select * from xmltable ('/emps/emp'
    passing '<emps>
+<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp>
+<emp><id>2</id><salary>234.56 C$</salary><hired>2/3/2004</hired></emp>'
           + '</emps>'
    columns id int path 'id', salary dec(5,2), hired date)
as items_table
-----
Msg 14841, Level 16, State 3:
Line 1:
XMLTABLE:Failed to convert column pattern result to DECML for column 1.
```

**示例 12** 若要处理其格式不适合于 SQL `convert` 函数的 XML 数据，请将该数据提取到字符串列（`varchar`、`text`、`image`、`java.lang.String`）中。

```
select * from xmltable ('/emps/emp'
    passing '<emps>
+<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp>
+<emp><id>2</id><salary>234.56 </salary><hired>2/3/2004</hired></emp>
+</emps>'
    columns id int, salary varchar(20), hired date)
as items_table
```

id	salary	hired
1	123.45	Jan 2, 2003
2	234.56	Feb 3, 2004

(2 rows affected)

**Ordinality 列** XML 文档中各元素的顺序可能会很重要。

元素有时会按其包含的元素的值排序。在下面的示例中，`<item>` 元素按其包含的 `<id>` 元素的值排序。

```
<doc>
  <item><id>1<name>Box</name></item>
  <item><id>2<name>Jar</name></item>
</doc>
```

也可以用一种任意但有意义的方式对元素进行排序。在下面的示例中，`<item>` 元素的顺序不基于任何值，但可能反映一种优先级顺序：先进先出。这种顺序对数据的应用而言可能很重要。

```
<doc>
  <item><id>25<name>Box</name></item>
  <item><id>15<name>Jar</name></item>
</doc>
```

**示例 13** 可以在 `xmltable` 中使用 `ordinality_column`，以记录输入 XML 文档中各元素的顺序：

```
declare @doc varchar(16384)
set @doc = '<doc><item><id>25<name>Box</name></item>'
  + '<item><id>15</id><name>Jar</name></item></doc>'
select * from xmltable('/doc/item' passing @doc
    columns item_order int for ordinality,
           id int path 'id',
           name varchar(20) path 'name') as items_table
order by item_order
```

item_order	id	name
------------	----	------

```

-----
1          25          Box
2          15          Jar
(2 rows affected)
-----

```

如果不使用 `for ordinality` 子句和 `item_order` 列，则 `id` 和 `name` 列中就不会有任何内容指示 `id` 为 25 的行位于 `id` 为 15 的行之前。通过 `for ordinality` 子句可确保输出 SQL 行的排序方式与这些元素在输入 XML 文档中的排序方式相同。

`ordinality` 列的数据类型可以是任何固定的数值数据类型：`int`、`tinyint`、`bigint`、`numeric` 或 `decimal`。`numeric` 和 `decimal` 的标度必须为 0。`ordinality` 列的数据类型不能为 `real`，也不能为 `float`。

**空值** 如果列模式返回空结果，则采取的操作取决于 `default` 和 `{null | not null}` 子句。

**示例 14** 下面的示例省略了第二个 `<item>` 中的 `<name>` 元素。缺省情况下，`name` 列允许名称为空值。

```

select * from xmltable ('//item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id></item></doc>'
    columns id int path 'id', name varchar(20), path 'name')
as items_table
-----
id          name
-----
1          Box
2          NULL
(2 rows affected)

```

**示例 15** 下面的示例省略了第二个 `<item>` 中的 `<name>` 元素，并为 `name` 列指定了 `not null`：

```

select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id></item></doc>'
    columns id int path 'id', name varchar(20) not null path 'name')
as items_table
-----
Msg 14847, Level 16, State 1:
Line 1:
XMLTABLE column 0, does not allow null values.

```

**示例 16** 下面的示例将 `default` 子句添加到 `name` 列中，并省略了第二个 `<item>` 中的 `<name>` 元素。

```
select * from xmldata ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id></item></doc>'
    columns id int path 'id' name varchar(20) default '***' path 'name')
as items_table
```

id	name
1	Box
2	***

(2 rows affected)

**xmldata 调用环境** 以下示例显示了一些 SQL 命令，您可以在这些命令中的派生表表达式中使用 `xmldata` 调用。

**示例 17** `select` — 可以在简单的 `select` 语句中使用 `xmldata()`:

```
select * from xmldata ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    + '<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id'
    name varchar(20) path 'name') as items_table
```

id	name
1	Box
2	Jar

(2 rows affected)

**示例 18** 视图定义 — 在视图定义中指定使用 `xmldata` 的 `select`。下面的示例将一个文档存储在一个表中，并在 `create view` 语句中引用所存储的该文档，同时使用 `xmldata` 从该表中提取数据：

```
select 100 as doc_id,
'<doc><item><id>1</id><name>Box</name></item>'
+ '<item><id>2</id><name>Jar</name></item></doc>' as doc
into sample_docs
create view items_table as
select * from xmldata('/doc/item'
    passing (select doc from sample_docs where doc_id=100)
    columns id int path 'id'
    name varchar(20) path 'name') as xml_extract
```

id	name
----	------



```

1          Box
2          Jar
(2 rows affected)

```

**示例 19** 游标声明 — 指定使用 xmltable 声明游标的 select:

```

declare cursor C for
select * from xmltable ('/doc/item'
    passing (select doc from sample_docs where id=100)
    columns id int path 'id'
    name varchar(20) path 'name') as items_table
declare @idvar int
declare @namevar varchar(20)
open C
while @@sqlstatus=0
begin
    fetch C into @idvar, @namevar
    print 'ID "%1!" NAME "%2!"', @idvar, @namevar
end
-----
ID "1" NAME "Box"
ID "2" NAME "Jar"

(2 rows affected)

```

在需要对生成的每行执行多项操作（例如，从其它表执行 `update`、`insert` 或 `delete`）的应用场合中，可以根据所生成的每行中的数据，使用游标循环处理 `xmltable` 结果。另外，也可以将 `xmltable` 结果存储在一个临时表中，然后使用游标循环处理该表。

**示例 20** `select into` — 在 `select into` 中指定使用 `xmltable` 的 `select`:

```

select * into #extracted_table
from xmltable('/doc/item'
    passing (select doc from sample_docs where doc_id=100)
    columns id int path 'id'
    name varchar(20) path 'name') as items_table

select * from #extracted_table

id          name
----          -
1          Box
2          Jar

```

**示例 21 insert** — 在 insert 命令中指定使用 xmltable 的 select:

```
create table #extracted_data (idcol int, namecol varchar(20))
insert into #extracted_data
select * into #extracted_table from xmltable('/doc/item'
      passing (select doc from sample_docs where doc_id=100
      columns id int path 'id', name varchar(20) path 'name') as items_table
select * from extracted_data
```

```
id          name
-----
1           Box
2           Jar
(2 rows affected)
```

**示例 22 标量子查询** — 在标量子查询中指定使用 xmltable 的 select。  
xmltable 返回一个 SQL 表，因此，标量子查询必须执行集合或选择操作，才能为标量子查询结果返回单个行和列。

```
declare @idvar int
set @idvar = 2
select @idvar,
(select name from xmltable ('/doc/item'
      passing(select doc from sample_docs where doc_id=100
      columns id int path 'id',name varchar(20) path 'name') as item_table
where items_table.id=@idvar)
-----
2           Jar
(1 rows affected)
```

**示例 23 连接** — 使用逗号列表连接或外部连接将 xmltable 结果与其它表连接:

```
create table prices (id int, price decimal (5,2))
insert into prices values(1,123.45)
insert into prices values (2,234.56)
select prices.id,extracted_table.name, prices.price
from prices,(select * from xmltable('/doc/item'
      passing (select doc from sample_docs where doc_id=100
      columns id int path 'id', name varchar(20) path 'name') as a) as
      extracted_table
where prices.id=extracted_table.id

id          name          price
-----
1           Box          123.45
2           Jar          234.56
(2 rows affected)
```

**处理文档表** 可以对 XML 文档表的每行中的 XML 文档应用 `xmltable()`。例如，下一示例创建一个包含以下两列的表：

- `pubs2_publishers` 表中的三个出版商之一的 `pub_id`。
- 一个 XML 文档，其中包含了由该出版商出版的每个文档的书目和价格。为减小示例表的大小，仅包含价格高于 \$15.00 的书目：

```
create table high_priced_titles
(pub_id char(4), titles varchar (1000))
insert into high_priced_titles
select p.pub_id,
      (select title_id, price from pubs2..titles t, pubs2..publishers p
       where price > 15 and t.pub_id = p.pub_id
       for xml
       option 'tablename=expensive_titles, rowname=title')
      returns varchar(1000)) as titles
from pubs2..publishers p
select * from high_priced_titles
-----
pub_id    titles
-----
0736      <expensive_titles>
          <title> <title_id>PS3333</title_id> <price>19.99</price></title>
          </expensive_titles>

0877      <expensive_titles>
          <title> <title_id>MC2222</title_id> <price>19.99</price></title>
          <title> <title_id>PS1372</title_id> <price>21.59</price></title>
          <title> <title_id>TC3218</title_id> <price>20.95</price></title>
          </expensive_titles>

01389     <expensive_titles>
          <title> <title_id>BU1032</title_id> <price>19.99</price></title>
          <title> <title_id>BU7832</title_id> <price>19.99</price></title>
          <title> <title_id>PC1035</title_id> <price>22.95</price></title>
          <title> <title_id>PC8888</title_id> <price>20.00</price></title>
          </expensive_titles>
(3 rows affected)
```

**示例 24** 在标量子查询中使用 `xmltable` 可将每一行中的 XML 文档作为 SQL 表进行处理。例如，下面列出了每个出版商的最高书目价格：

```
select pub_id
(select max(price)
 from xmltable('/title' passing hpt.titles
               columns title_id char(4), price money)
       as extracted_titles, high_priced_titles hpt) as max_price
from high_priced_titles hpt
-----
pubid          max_price
-----
0736           19.99
0877           21.59
1389           22.95
```

此 `high_priced_titles` 表实质上是分层的：每一行都是一个中间节点，并在其 `title` 列中包含与 XML 文档中的每个 `title` 元素对应的一个叶节点。  
`high_priced_titles` 包含三行。

您可以展平该层次，从而产生一个由每个 `title` 元素各占一行的表。若要展平 `titles` 列中的数据并生成一个包含八行（每个 `titles/title` 元素各占一行）的表 `high_priced_titles_flattened`，可采用以下解决方案之一。

**解决方案 1** 可以通过使用处理 `high_priced_titles` 并将 `xmltable` 应用于每行中的书目文档的一个循环，来生成 `high_priced_titles_flattened`。在下面的示例中，应注意 `from` 子句：

```
from(select @pub_id_var)as ppp,
     xmltable('/title' passing @titles_var
               columns title_id char(6),price money)as ttt
```

变量 `@pub_id_var` 和 `@titles_var` 分别是 `high_priced_titles` 当前行中的 `pub_id` 和 `titles` 列。`from` 子句连接以下两个派生表：

- `(select @pub_id_var) as ppp`

这是一个包含一行、一列的表，它包含 `pub_id`。

- `xmltable(...)as ttt`

它生成一个表，`high_priced_titles` 当前行的 `titles` 文档中的每个 `title` 元素在该表中各占一行。

为展平该层次，需将这两个派生表连接起来，这会将 `pub_id` 列追加到用 `titles` 列生成的每一行：

```
create table high_priced_titles_flattened_1
(pub_id char(4), title_id(char(6), price money)
```

```

declare C cursor for select * from high_priced_titles
declare @pub_id_var char(4)
declare @titles_var char(1000)
open C

while @@sqlstatus =0
begin
fetch C into @pub_id_var, @titles_var

insert into high_priced_titles_flattened_1
select *
from (select @pub_id_var) as ppp,(col1),
     xmltable('///title' passing @titles_var
              columns title_id char (6), price money) as ttt
end
select * from high_priced_titles_flattened_1

```

pub_id	title_id	price
0736	PS3333	19.99
0877	MC2222	19.99
0877	PS1372	21.59
0877	TC3218	20.95
1389	BU1032	20.95
1389	BU7832	19.99
1389	PC1035	19.99
1389	PC8888	20.00

**解决方案 2** 还可以使用一种特殊连接生成 high\_priced\_titles 表。

下面的示例连接两个表：high\_priced\_titles as hpt 以及由 xmltable 生成的表。xmltable 的传递参数引用前面的 hpt 表。通常，在 from 子句中的派生表表达式中引用同一个 from 子句中的表是非法的。不过，允许 xmltable 引用同一个 from 子句中的其它表，只要这些表在同一 from 子句中位于 xmltable 调用之前即可。

```

select hpt.pub_id, extracted_titles.*
into high_priced_titles_flattened_3
from high_priced_titles as hpt,
     xmltable('///title'
              passing htp.titles,
              columns
                  title_id char(6)
                  price money)as extracted_titles

```

pub_id	title_id	price
-----	-----	-----

0736	PS3333	19.99
0877	MC2222	19.99
0877	PS1372	21.59
0877	TC3218	20.95
1389	BU1032	20.95
1389	BU7832	19.99
1389	PC1035	19.99
1389	PC8888	20.00

## 用法

- `xmltable` 是一个内置的表值函数。
- `xmltable` 表达式的结果类型是 SQL 表，其列名称和数据类型由 `column_definitions` 指定。
- 以下关键字与 `xmltable` 关联：
  - 保留关键字：for、option
  - 非保留关键字：columns、ordinality、passing、path、xmltable
- `xmltable` 调用的参数中的表达式可以引用包含此 `xmltable` 调用的 `from` 子句中前面表的列名。只能引用 `xmltable` 调用之前的表。对同一 `from` 子句中前面表的列的引用称为 *侧面引用*。例如：

```
select * from T1, xmltable(...passing T1.C1...)
as XT2, xmltable(...passing XT2.C2...)as XT3
```

对第一个 `xmltable` 调用中 `T1.C1` 的引用就是对表 `T1` 的 `C1` 列的侧面引用。对第二个 `xmltable` 调用中 `XT2.C2` 的引用就是对由第一个 `xmltable` 调用生成的表的 `C2` 列的侧面引用。

- 不能在 `update` 或 `delete` 语句的 `from` 子句中使用 `xmltable`。例如，以下语句将失败：

```
update T set T.C=...
from T,xmltable(...)
where...
```

- 不能更新由 `xmltable` 表达式返回的 SQL 表。
- `regular_columns` 中的数据类型可以是任何 SQL 数据类型。
- `regular_column` 中 `default` 后紧跟的 `literal` 必须能够赋给相应列的数据类型。
- 最多只能有一个 `ordinality_column`；为此变量指定的数据类型必须为 `integer`、`smallint`、`tiny int`、`decimal` 或 `numeric`。`decimal` 和 `numeric` 的标度必须为零。

- 如果有 *ordinality\_column*，则它不能为空值。其它列的可为空值的属性由 {null | not null} 子句指定。缺省值为 null。

---

**注释** 此缺省值与 create table 的缺省值不同。

---

- set quoted\_identifier 的当前设置适用于 xmltable 表达式的子句。例如，
  - 如果 set quoted\_identifier 是 on，则列名可以为带引号的标识符，并且必须用单引号引起 row\_pattern 和 column\_pattern 中的字符串文字以及缺省文字。
  - 如果 set quoted\_identifier 是 off，则列名不能为带引号的标识符，并且可以用单引号或双引号引起 row\_pattern 和 column\_pattern 中的字符串文字以及缺省文字。
- option\_string 的一般格式在 “option\_strings: 一般格式” 一节中进行了介绍。

**xmltable 行模式和列模式** xmltable 的行模式和列模式只能是简单路径。XPath 中的简单路径只包含使用 “/” 和元素 / 属性名称的前向遍历。

- 如果 row\_pattern 没有从由 xml\_argument 指定的文档的根级别开始，则会引发异常。*行模式必须从 XML 文档的根级别开始。*
- 如果 row\_pattern 指定了 XML 函数，则会引发异常。*行模式不能指定 XML 函数。*
- 如果 column\_definition 并未指定路径，则缺省 column\_pattern 为该列定义的 column\_name。此缺省值受服务器是否区分大小写的影响。例如，假设有下面的语句：

```
select * from xmltable(...columns name
varchar(30),...)
```

如果服务器不区分大小写，此语句等效于下面的语句：

```
select * from xmltable(...columns name varchar(30)
path 'name',...)
```

如果服务器区分大小写，则第一个语句等效于下面的语句：

```
select * from xmltable
(...columns name varchar(30)path 'NAME',...)
```

**生成结果表的行**

xmltable 表达式的结果值是 T-SQL 表 RT，其定义如下：

- 对 xml\_argument 应用 row\_pattern 后产生的 XML 序列中的每个元素在 RT 中各占一行。

- RT 行为每个 *column\_definition* 提供一列，并在 *column\_definition* 中指定 *column\_name* 和数据类型。
- 如果 *column\_definition* 是 *ordinality\_column*，则其 Nth 行的值为整数 N。
- 如果 *column\_definition* 是 *regular\_column*，则其 Nth 行的值与下列内容相对应：
  - 让 XVAL 作为将此 XPath 表达式应用于 *xml\_argument* 的结果：  
`(row_pattern[N])/column_pattern/text()`
  - 如果 XVAL 为空，并且 *column\_definition* 包含缺省子句，则此列的值为缺省值。  
如果 XVAL 为空，并且 *column\_definition* 指定非空值，则会发生异常。  
否则，该列的值为空值。
  - 如果 XVAL 不为空，并且该列的数据类型为 `char`、`varchar`、`text`、`unitext`、`unichar`、`univarchar` 或 `java.lang.String`，则取消对 XVAL 的实体化。
  - 列的值由以下语句产生：  
`convert(datatype,XVAL)`

另请参见

有关使用 `xmldata` 的示例应用程序，请参见《XML 服务》中的[附录 F “关于 `xmldata\(\)` 的示例应用程序”](#)。



## *sample\_docs* 示例表

XML 查询函数的说明引用一个名为 *sample\_docs* 的示例表。本章介绍如何创建和填充该表。

*sample\_docs* 表包含三个列和三个行。

### *sample\_docs* 表的列和行

本节显示 *sample\_docs* 表的结构。

#### Sample\_docs 表的列

*sample\_docs* 表包含以下三列：

- name\_doc
- text\_doc
- image\_doc

在指定的示例文档中，name\_doc 指定标识名称，text\_doc 指定 text 表示形式的文档，image\_doc 指定存储在 image 列中、经过分析的 XML 表示形式的文档。下面的脚本将创建该表：

```
create table sample_docs
(name_doc varchar(100),
text_doc text null,
image_doc image null)
```

## sample\_docs 表的行

sample\_docs 表包含以下三行：

- 示例文档 “bookstore.xml”。
- pubs2 数据库中 publishers 表的 XML 表示形式。
- pubs2 数据库中 titles 表 （的选定列）的 XML 表示形式。

下面的脚本将示例 “bookstore.xml” 文档插入 sample\_docs 表的一行中：

```
insert into sample_docs
  (name_doc, text_doc)
  values ( "bookstore",

  "<?xml version='1.0' standalone = 'no'?>
<?PI_example Process Instruction ?>
<!-- 示例注释 -->
<bookstore specialty='novel'>
<book style='autobiography'>
  <title>Seven Years in Trenton</title>
  <author>
    <first-name>Joe</first-name>
    <last-name>Bob</last-name>
    <award>Trenton Literary Review
      Honorable Mention</award>
  </author>
  <price>12</price>
</book>
<book style='textbook'>
  <title>History of Trenton</title>
  <author>
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    <publication>Selected Short Stories of
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    </publication>
  </author>
  <price>55</price>
</book>
<?PI_sample Process Instruction ?>
<!-- 示例注释 -->
<magazine style='glossy' frequency='monthly'>
  <title>Tracking Trenton</title>
  <price>2.50</price>
  <subscription price='24' per='year'/>
  </magazine>
</bookstore>
"
```

```

</magazine>
<book style='novel' id='myfave'>
  <title>Trenton Today, Trenton Tomorrow</title>
  <author>
    <first-name>Toni</first-name>
    <last-name>Bob</last-name>
    <degree from='Trenton U'>B.A.</degree>
    <degree from='Harvard'>Ph.D.</degree>
    <award>Pulizer</award>
    <publication>Still in Trenton</publication>
    <publication>Trenton Forever</publication>
  </author>
  <price intl='canada' exchange='0.7'>6.50</price>
  <excerpt>
    <p>It was a dark and stormy night.</p>
    <p>But then all nights in Trenton seem dark and
      stormy to someone who has gone through what
      <emph>I</emph> have.</p>
    <definition-list>
      <term>Trenton</term>
      <definition>misery</definition>
    </definition-list>
  </excerpt>
</book>

<book style='leather' price='29.50'
xmlns:my='http://www.placeholdernamehere.com/schema/'>
  <title>Who's Who in Trenton</title>
  <author>Robert Bob</author>
</book>

</bookstore>")

```

## sample\_docs 表

sample\_docs 表的另外两行是 pubs2 数据库中 publishers 表和 titles 表的 XML 表示形式。pubs2 数据库是《Transact-SQL 用户指南》中所述的示例表数据库。

publishers 表和 titles 表是该示例数据库中的两个表。为了简化示例，titles 表的 XML 表示形式仅包含选定列。

下面的脚本用 forxmlj 函数（在第 65 页的“forxmlj、forxmltdj、forxmlschemaj、forxmlallj”中介绍）生成 publishers 和 titles 表的 XML 表示形式。

### 表脚本（publishers 表）

以下两个 insert 语句将 publishers 表的一行和 authors 表的一行添加到 sample\_docs 表中。每行都包含一个标识该行的列（“publishers”、“authors”），还包含一个 text\_doc 列，

该列提供对应的 pubs2 表的 XML 表示形式。您可以通过调用 Java forxmlj 函数来生成 XML 文档。

```
insert into sample_docs (name_doc, text_doc)
  values ('publishers',
    forxmlj('select * from pubs2..publishers',
      'tablename=publishers'))

insert into sample_docs (name_doc, text_doc)
  values ('authors',
    forxmlj('select title_id, title
      type, pub_id, price,
      advance, total_sales
      from pubs2..authors',
      'tablename=authors'))
```

---

**注释** 此脚本使用 forxmlj 函数，这一基于 Java 的函数必须先安装，然后才能使用。有关安装此函数的说明，请参见附录 C “设置 XML 服务”。

---

## Publishers 表的表示形式

下面的代码示例显示 Pubs 2 数据库中 *publishers* 表的 XML 表示形式，此表示形式由第 138 页的“sample\_docs 表”中的脚本生成。

```
set stringsize 16384
select text_doc from sample_docs
where name_doc='publishers'

text_doc
-----
<publishers
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  instance">

  <row>
    <pub_id>0736</pub_id>
    <pub_name>New Age Books</pub_name>
    <city>Boston</city>
    <state>MA</state>
  </row>

  <row>
    <pub_id>0877</pub_id>
    <pub_name>Binnet & Hardley</pub_name>
    <city>Washington</city>
    <state>DC</state>
  </row>

  <row>
    <pub_id>1389</pub_id>
    <pub_name>Algodata Infosystems</pub_name>
    <city>Berkeley</city>
    <state>CA</state>
  </row>

</publishers>
(1 row affected)
```

## Titles 表的表示形式

本节显示 titles 表的选定列的 XML 表示形式。

```
set stringsize 16384
select text_doc from sample_docs
where name_doc='titles'

text_doc
-----
<titles
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <row>
    <title_id>BU1032</title_id>
    <title>The Busy Executive's Data Base
      Guide</title>
    <type>business</type>
    <pub_id>1389</pub_id>
    <price>19.99</price>
    <advance>5000.00</advance>
    <total_sales>4095</total_sales>
  </row>

  <row>
    <title_id>BU1111</title_id>
    <title>Cooking with Computers:
      Surreptitious Balance Sheets</title>
    <type>business </type>
    <pub_id>1389</pib_id>
    <price>11.95</price>
    <advance>5000.00</advance>
    <total_sales>3876</total_sales>
  </row>

  <row>
    <title_id>BU2075</title_id>
    <title>You Can Combat Computer Stress!</title>
    <type>business </type>
    <pub_id>0736</pub_id>
    <price>2.99</price>
    <advance>10125.00</advance>
    <total_sales>18722</total_sales>
  </row>

  <row>
    <title_id>BU7832</title_id>
```

```
<title>Straight Talk About Computers</title>
<type>business </type>
<pub_id>1389</pub_id>
<price>19.99</price>
<advance>5000.00</advance>
<total_sales>4095</total_sales>
</row>

<row>
  <title_id>MC2222</title_id>
  <title>Silicon Valley Gastronomic Treats</title>
  <type>mod_cook</type>
  <pub_id>0877</pub_id>
  <price>19.99</price>
  <advance>0</advance>
  <total_sales>2032</total_sales>
</row>

<row>
  <title_id>MC3021</title_id>
  <title>The Gourmet Microwave</title>
  <type>mod_cook</type>
  <pub_id>0877</pub_id>
  <price>2.99</price>
  <advance>15000.00</advance>
  <total_sales>22246</total_sales>
</row>

<row>
  <title_id>MC3026</title_id>
  <title>The Psychology of Computer Cooking</title>
  <type>UNDECIDED</type>
  <pub_id>0877</pub_id>
</row>

<row>
  <title_id>PC1035</title_id>
  <title>But Is IT User Friendly?</title>
  <type>popular_comp</type>
  <pub_id>1389</pub_id>
  <price>22.99</price>
  <advance>7000.00</advance>
  <total_sales>8780</total_sales>
</row>

<row>
```

```
<title_id>PC8888</title_id>
<title>Secrets of Silicon Valley</title>
<type>popular_comp</type>
<pub_id>1389</pub_id>
<price>20.00</price>
<advance>8000.00</advance>
<total_sales>4095</total_sales>
</row>

<row>
  <title_id>PC9999</title_id>
  <title>Net Etiquette</title>
  <type>popular_comp</type>
  <pub_id>1389</pub_id>
</row>

<row>
  <title_id>PS1372</title_id>
  <title>Computer Phobic and Non-Phobic
    Individuals: Behavior Variations</title>
  <type>psychology </type>
  <pub_id>0877</pub_id>
  <price>21.59</price>
  <advance>7000.00</advance>
  <total_sales>375</total_sales>
</row>

<row>
  <title_id>PS2091</title_id>
  <title>Is Anger the Enemy?</title>
  <type>psychology </type>
  <pub_id>0736</pub_id>
  <price>10.95</price>
  <advance>2275.00</advance>
  <total_sales>2045</total_sales>
</row>

<row>
  <title_id>PS2106</title_id>
  <title>Life Without Fear</title>
  <type>psychology </type>
  <pub_id>0736</pub_id>
  <price>7.99</price>
  <advance>6000.00</advance>
  <total_sales>111</total_sales>
</row>
```



```
<row>
  <title_id>PS3333</title_id>
  <title>Prolonged Data Deprivation:
    Four Case Studies</title>
  <type>psychology</type>
  <pub_id>0736</pub_id>
  <price>19.99</price>
  <advance>2000.00</advance>
  <total_sales>4072</total_sales>
</row>

<row>
  <title_id>PS7777</title_id>
  <title>Emotional Security:
    A New Algorithm</title>
  <type>psychology </type>
  <pub_id>0736</pub_id>
  <price>7.99</price>
  <advance>4000.00</advance>
  <total_sales>3336</total_sales>
</row>

<row>
  <title_id>TC3218</title_id>
  <title>Onions, Leeks, and Garlic:
    Cooking Secrets of the Mediterranean</title>
  <type>trad_cook </type>
  <pub_id>0877</pub_id>
  <price>20.95</price>
  <advance>7000.00</advance>
  <total_sales>375</total_sales>
</row>

<row>
  <title_id>TC4203</title_id>
  <title>Fifty Years in Buckingham
    Palace Kitchens</title>
  <type>trad_cook </type>
  <pub_id>0877</pub_id>
  <price>11.95</price>
  <advance>4000.00</advance>
  <total_sales>15096</total_sales>
</row>

<row>
```

```
<title_id>TC7777</title_id>
<title>Sushi, Anyone?</title>
<type>trad_cook </type>
<pub_id>0877</pub_id>
<price>14.99</price>
<advance>8000.00</advance>
<total_sales>4095</total_sales>
</row>

</titles>

(1 row affected)
```

# XML 服务和外部文件系统访问

Adaptive Server 的外部文件系统访问功能以 SQL 表的形式提供对操作系统文件的访问。本附录介绍如何将本机 XML 处理器用于文件系统访问功能。有关详细信息，请参见《Adaptive Server 组件集成服务用户指南》。

当您使用文件系统访问功能时，就会创建一个代理表，此代理表使用 Adaptive Server 的组件集成服务 (CIS) 功能映射外部文件系统中的整个目录树。然后您就可以对此代理表中的数据使用本机 XML 处理器的内置函数，以查询外部文件系统中存储的 XML 文档。

利用外部目录递归访问，可以将代理表映射到父目录，并映射到它的所有下级文件和子目录。

## 入门

本节介绍如何通过外部文件系统访问功能设置 XML 服务。

### 启用 XML 服务和外部文件系统访问

- 使用 `sp_configure` 启用 XML 服务、CIS 和文件访问：

```
sp_configure "enable xml", 1
```

- 检验是否将配置参数 `enable cis` 设置为 1：

```
sp_configure "enable cis", 1
```

- 使用 `sp_configure` 启用文件访问：

```
sp_configure "enable file access", 1
```

## 外部文件系统的字符集转换

通常，外部文件系统表的内容列被视为 `image`。但是，当把内容列分配给一个 `Unicode` 列（即，数据类型为 `unichar`、`univarchar`、`unitext` 或 `java.lang.String` 的列）时，将会执行特殊转换。出现以下情况时，内容列将被分配给 `Unicode` 列：

- `insert` 命令用于插入引用了内容列的子查询中的 `Unicode` 列。
- `update` 命令用引用了内容列的新值来更新 `Unicode` 列。
- `convert` 函数调用指定目标 `Unicode` 数据类型和作为一个内容列的源值。

在将内容列分配给 `Unicode` 时，请遵循以下规则：

- 如果源文档有 `BOM`（字节顺序标记），要转换源文档，`BOM` 必须指示 `UTF-8` 或 `UTF-16`。如果 `BOM` 指示 `UCS-4`，将出现错误。不支持 `UCS-4`。
- 如果源文档有一个包括编码子句的 `XML` 标头，但没有 `BOM`，则编码子句必须指定服务器字符集或 `UTF-8` 以转换源数据。当编码子句指定了非服务器字符集或 `UTF-8` 的字符集时，将引发错误。
- 如果源文档没有 `XML` 标头，而只有一个没有编码子句的标头，也没有 `BOM`，则处理器将字符集视为 `UTF-8`，并对源数据进行转换。
- 如果在转换过程中发生了故障，将会显示错误，但会继续执行语句。

## 示例

以下示例显示如何使用各种 `XML` 内置函数查询外部文件系统上的 `XML` 文档。

## 设置 XML 文档并创建代理表

这些示例使用两个 `XML` 文档，它们存储在您创建的名为 `bookstore.1.xml` 和 `bookstore.2.xml` 的文件中：

```
cat bookstore.1.xml
```

```
<?xml version='1.0' standalone = 'no'?>
<!-- bookstore.1.xml 示例文档 --!>
<bookstore specialty='novel'>
<book style='autobiography'>
```

```

<title>Seven Years in Trenton</title>
<author>
  <first-name>Joe</first-name>
  <last-name>Bob</last-name>
  <award>Trenton Literary Review Honorable Mention</award>
</author>
<price>12</price>
</book>
</bookstore>

```

```
cat bookstore.2.xml
```

```

<?xml version='1.0' standalone = 'no'?>
<!-- bookstore.2.xml 示例文档 -->
<bookstore specialty='novel'>
  <book style='compbook'>
    <title>Modern Database Management</title>
    <author>
      <first-name>Jeffrey</first-name>
      <last-name>Hoffer</last-name>
    </author>
    <price>112.00</price>
  </book>
</bookstore>

```

您可以使用 **create proxy table** 通过文件系统访问功能来引用这些 XML 文档。

下面的代码示例说明了 **create proxy table** 的用法。**at** 子句中的目录路径名必须引用 **Adaptive Server** 既能看到又能搜索到的文件系统目录。如果在路径名的末尾添加 “;R”（表示“递归”）扩展名，**CIS** 将从该路径名的每个下级目录中提取文件信息。

```

create proxy_table xmlxfsTab external directory
at "/remote/nets3/bharat/xmldocs;R"
select filename from xmlxfsTab f

filename
-----
bookstore.1.xml
bookstore.2.xml

(2 rows affected)

```

重要的列为 **filename** 和 **content**。其它列包含关于访问权限等的信息。**filename** 列包含文件名（此例中为 XML 文档文件名），而 **content** 列则包含该文件的实际数据。**content** 列的数据类型为 **image**。

## 示例：从 XML 文档中提取书名

```
select filename, xmlextract("//book/title" , content)
from xmlxfsTab

filename
-----
bookstore.1.xml
<title>Seven Years in Trenton</title>
bookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

## 示例：将 XML 文档或 XML 查询结果导入 Adaptive Server 表

您可以在一个文件访问目录结构和一个数据库表或另一个文件访问目录结构之间传送完整的 XML 文档或 XML 查询结果。若要引用完整的 XML 文档，请使用带根路径 XPath 运算符（“/”）的 `xmlextract` 函数。

```
insert into xmldoctab select filename,xmlextract("/",content) into
from xmlxfsTab
-----
(2 rows affected)
```

在此示例中，`xmlxfsTab.content` 列的数据类型为 `image`，`xmlextract` 内置函数所返回的缺省数据类型为 `text`。因此，请在 `xmlextract` 调用中指定 `returns image` 子句，以便将结果作为 `image` 值返回。

若要保留标头，请使用 `xmlvalidate()` 来代替 `xmlextract()`：

```
insert into xmldoctab select filename,xmlvalidate(content)
from xmlxfsTab
-----
(2 rows affected)
```

下面的语句将创建一个新的子目录 *XmlDir*：

```
insert into xmlxfsTab(filename,content)
select filename = 'XmlDir/'+filename,
       xmlextract("/",xmlcol returns image) from xmldoctab
-----
(2 rows affected)
```

下面的代码示例从新的 *XmlDir* 子目录中查询这些 XML 文档：

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where filename like '%XmlDir%' and filetype = 'REG'

filename
-----
XmlDir/bookstore.1.xml
<title>Seven Years in Trenton</title>
XmlDir/bookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

## 示例：将经过分析的 XML 文档存储在文件系统中

您可以分析存储在外部文件系统中的 XML 文档，然后将分析的结果存储在 Adaptive Server 表或文件访问系统中。

```
insert xmlxfsTab(filename, content)
select 'parsed'+t.filename,xmldata(t.content) from xmlxfsTab
-----
(2 rows affected)
```

下面的代码示例查询存储在 XFS 文件系统中的经过分析的文档。

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where filename like 'parsed%' and filetype = 'REG'
filename
-----
parsedbookstore.1.xml
<title>Seven Years in Trenton</title>
parsedbookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

下面的代码示例使用 `xmlrepresentation` 内置函数仅查询作为经过分析的 XML（而不是其它类型的外部文件）的文件访问文档：

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where xmlrepresentation(content) = 0
filename
-----
parsedbookstore.1.xml
<title>Seven Years in Trenton</title>
parsedbookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

## 示例：针对外部文件访问的“xmlerror”选项功能

外部 (O/S) 文件系统可能包含多种数据格式，并且可能同时包含有效和无效的 XML 文档。您可以使用 `xmlextract` 和 `xmltest` 函数的 `xmlerror` 选项来为不是有效 XML 的文档指定错误操作。

例如，文件访问目录结构可能包含 *picture.jpg* 和 *nonxmldoc.txt* 文件，以及 *bookstore1.xml* 和 *bookstore.2.xml* 文件：

```
select filename from xmlxfsTab
filename
-----
picture.jpg
bookstore.1.xml
bookstore.2.xml
nonxmldoc.txt

(4 rows affected)
```

下面的代码示例显示对 XML 数据和非 XML 数据的 XML 查询：

```
select filename, xmlextract("//book/title",content)
from xmlxfsTab
-----
Msg 14702, Level 16, State 0:
Line 1:
XMLEXTRACT():XML parser fatal error <<An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered>> at line 1, offset 1.
```



## 示例：在 xmlextract 中指定 “xmlerror=message” 选项

此示例在 xmlextract 调用中指定 'xmlerror=message' 选项。对于是有效 XML 的 XML 文档，这将返回 XML 查询结果；对于不是有效 XML 的文档，这将返回 XML 错误消息元素。

```
select filename, xmlextract("//book/title",content
    option 'xmlerror = message') from xmlxfsTab
filename
-----
picture.jpg
<xml_parse_error>An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered</xml_parse_error>

bookstore.1.xml
<title>Seven Years in Trenton</title>

bookstore.2.xml
<title>Modern Database Management</title>
nonxmldoc.txt
<xml_parse_error>Invalid document structure</xml_parse_error>

(4 rows affected)
```

## 示例：用 “xmlerror=message” 选项分析 XML 和非 XML 文档

此代码示例在 xmlparse 调用中指定 'xmlerror= message' 选项。对于是有效 XML 的 XML 文档，这将存储经过分析的 XML；对于不是有效 XML 的文档，这将存储经过分析的 XML 错误消息元素。

```
insert xmlxfsTab(filename, content)
select 'ParsedDir/'+filename, xmlparse(content option
    'xmlerror = message')
from xmlxfsTab
-----

(4 rows affected)
```

下面的代码示例将 `xmlextract` 内置函数应用于经过分析的数据，并获取非 XML 数据的列表以及异常消息信息。

```
select filename, xmlextract('/xml_parse_error', content)
from xmlxfsTab
where '/xml_parse_error' xmltest content and filename like 'ParsedDir%'
-----
Or with xmlrepresentation builtin
select filename, xmlextract('/xml_parse_error', content)
from xmlxfsTab
where xmlrepresentation(content) = 0
and '/xml_parse_error' xmltest content
filename
-----
ParsedDir/picture.jpg
<xml_parse_error>An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered</xml_parse_error>

ParsedDir/nonxmldoc.txt

<xml_parse_error>Invalid document structure
</xml_parse_error>

(2 rows affected)
```

## 示例：对非 XML 文档使用 “xmlerror=null” 选项

下面的代码示例针对一个文件访问表指定 'xmlerror = null' 选项：

```
select filename, xmlextract("//book/title", content
    option 'xmlerror = null')
from xmlxfstb
filename
-----
picture.jpg
NULL
bookstore.1.xml
<title>Seven Years in Trenton</title>

bookstore.2.xml
<title>Modern Database Management</title>
nonxmldoc.txt
NULL

(4 rows affected)
```

下面的代码示例用 'xmlerror = null' 选项选择非 XML 文档名的列表。

```
select filename from xmlxfstb
where '/' not xmltest content
    option 'xmlerror = null'
filename
-----
picture.jpg
nonxmldoc.txt

(2 rows affected)
```

如果用户要保留标头，则应使用 xmlvalidate()，而非 xmlextract()。

```
insert into xmldoctab select filename ,xmlvalidate(content)
from xmlxfstb
```



# 设置 XML 服务

本附录提供关于设置集成 XML 处理器和基于 Java 的处理器说明。

## 安装基于 Java 的 SQLX 映射函数

由于第 4 章“XML 映射函数”中的函数基于 Java，因此必须先将它们安装在服务器上，然后才能使用。本节提供关于安装基于 Java 的函数的说明。

### 基于 Java 的 XML 函数

必须将这些函数安装在服务器上，然后才能使用：

- forxmlj
- forxmldtdj
- forxmlschemaj
- forxmlallj
- forsqlcreatej
- forsqlinsertj
- forsqlscriptj

在以下目录中可以找到关于安装这些工具的指南和设置脚本以及它们的源代码和 JavaDoc：

`$SYBASE/$SYBASE_ASE/sample`

## 映射函数安装

若要安装基于 Java 的 SQLX 映射函数，请按照本节所述的过程操作。

## 环境变量

表 C-1 中的环境变量已存在于服务器工具中。

**表 C-1：环境变量**

变量	值
<code>\$ISERVER</code>	isql 和 installjava 实用程序的 “-S” 参数
<code>\$INTERFACES</code>	isql 和 installjava 实用程序的 “-I” 参数
<code>\$DB</code>	isql 和 installjava 实用程序的 “-D” 参数

## 安装分析程序

使用目录 `$SYBASE/$SYBASE_ASE/sample` 中引用的 `setup` 目录中的 `make install-xerces` 命令或者使用诸如以下命令的客户端实用程序命令，安装基于 Java 的 XML 分析程序：

```
installjava -f $SYBASE/$SYBASE_ASE/lib/xerces.jar\  
-j "xerces_jar"\  
-D $DB -S $ISERVER -I $INTERFACES\  
-update -Usa -P"
```

**注释** forsqlcreatej、forsqlinsertj 和 forsqlscriptj 需要基于 Java 的 XML 分析程序；forxmlj、forxmldtdj、forxmlschemaj 或 forxmlallj 则不需要。

## 安装映射函数

使用 `$SYBASE/$SYBASE_ASE/sample` 中引用的 `setup` 目录中的 `make install-sqlx` 命令或者使用诸如以下命令的客户端实用程序命令，安装基于 Java 的 SQLX 映射类。

```
installjava  
-f ../SQLX-examples/sqlx.jar -j "sqlx_jar"\  
-D $DB -S $ISERVER -I $INTERFACES  
-update -Usa -P"
```

## 创建别名名称

使用 `$SYBASE/$SYBASE_ASE/sample` 中引用的 `setup` 目录中的 `make sqlx-aliases` 命令或者使用诸如以下命令的服务器 SQL 命令，可以为 SQLX 映射类的 Java 方法创建 SQL 别名名称：

```
create procedure forxmlallj
    (queryparmparm java.lang.String, optionparm
      java.lang.String,
      out rsout java.lang.String,
      out schemaout java.lang.String,
      out dtdout java.lang.String )
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXmlAll"

create function forxmlj
    (queryparm java.lang.String, optionparm
      java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXml"

create function forxmlschemaj
    (queryparm java.lang.String,optionparm
      java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXmlSchema"

create function forxmldtdj
    (queryparm java.lang.String, optionparm
      java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXmlDTD"

create function forsqlcreatej
    (schemax java.lang.String, optionparm
      java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.SqlxCommand.forSqlCreate"

create function forsqlinsertj
    (inDoc java.lang.String, optionparm java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.SqlxCommand.forSqlInsert"
```

```
create function forsqlscriptj
(schemax java.lang.String, inDoc java.lang.String,
optionparm java.lang.String)
returns java.lang.String
language java parameter style java
external name "jcs.sqlx.SqlxCommand.forSqlScript"
```



## 基于 Java 的 XQL 处理器

本章介绍如何使用 XQL 从 Adaptive Server 选择原始数据（使用 XQL 语言）以及将结果显示为 XML 文档。

XML 服务提供基于 Java 的 XQL 处理器。基于 Java 的 XQL 处理器实现 XQL 语言，该语言是 XPath 的扩展。

基于 Java 的 XQL 处理器是基于 XPath 的 XML 查询工具的初步实现。其功能由本机 XML 处理器的功能所取代。

您可以将基于 Java 的 XQL 处理器安装在服务器中，也可以在服务器外运行它。在服务器外运行它与在命令行运行任何 Java 程序一样。

本附录首先介绍如何将基于 Java 的 XQL 处理器当作独立程序在 Adaptive Server 外运行，然后介绍如何在 Adaptive Server 中运行它。

### 设置基于 Java 的 XQL 处理器

在使用基于 Java 的 XQL 处理器之前，您必须设置 classpath 变量、安装该处理器并设置内存要求。

#### 设置 CLASSPATH 环境变量

若要在 Adaptive Server 外创建一个独立程序，必须设置 CLASSPATH 环境变量，使之包括含有 *xerces.jar* 和 *xml.zip* 的目录。对于 UNIX，请输入：

```
setenv CLASSPATH $SYBASE/$SYBASE_ASE/lib/xerces.jar
$SYBASE/_SYBASE_ASE/lib/xml.zip
```

对于 Windows，请输入：

```
set CLASSPATH =
D:\%SYBASE%\_SYBASE_ASE\lib\xerces.jar
D:\%SYBASE%\_SYBASE_ASE\lib\xml.zip
```

## 在 Adaptive Server 中安装基于 Java 的 XQL 处理器

本节假定已经在 Adaptive Server 中启用了 Java。有关启用 Java 的信息，请参见《Adaptive Server® Enterprise 中的 Java》。

`installjava` 将 JAR 文件复制到 Adaptive Server 中，并使该 JAR 文件中的 Java 类可在当前数据库中使用。语法为：

```
installjava
-f file_name
[-new | -update ]
...
```

其中：

- *file\_name* 是要在服务器中安装的 JAR 文件的名称。
- `new` 通知服务器这是一个新文件。
- `update` 通知服务器将更新一个现有 JAR 文件。

有关 `installjava` 的详细信息，请参见《实用程序指南》。

若要在 Adaptive Server 中添加对 XML 的支持，则必须安装 `xml.zip` 和 `xerces.jar` 文件。这些文件位于目录 `$SYBASE/_SYBASE_ASE/lib/xml.zip` 和 `$SYBASE/_SYBASE_ASE/lib/xerces.jar` 中

例如，若要安装 `xml.zip`，请输入：

```
installjava -Usa -P -Sserver_name -f $SYBASE/_SYBASE_ASE/lib/xml.zip
```

若要安装 `xerces.jar`，请输入：

```
installjava -Usa -P -Sserver_name -f $SYBASE/_SYBASE_ASE/lib/xerces.jar
```

---

**注释** 若要在数据库中安装 `xerces.jar`，必须将 `tempdb` 的大小增加 10 MB。

---

## 在 Adaptive Server 内运行基于 Java 的 XQL 处理器时的内存要求

根据要通过基于 Java 的 XQL 处理器引用的 XML 数据的大小，可能需要增加内存。对于大小为 2 K 的典型 XML 文档，Sybase 建议将 Java 服务中的配置参数设置为表 D-1 中所示的值。有关配置参数的详细信息，请参见《Sybase Adaptive Server 系统管理指南》。

**表 D-1: Java 服务内存参数**

节	重新设置值
enable java	1
size of process object heap	5000
size of shared class heap	5000
size of global fixed heap	5000

## 使用基于 Java 的 XQL 处理器

### 将原始 XML 文档转换为已分析的版本

使用 `parse()` 方法转换和分析原始的 `text` 或 `image` XML 文档并存储结果。使用 `alter table` 命令转换原始 XML 文档。例如：

```
alter table XMLTEXT add xmldoc IMAGE null
update XMLTEXT
set xmldoc = com.sybase.xml.xql.Xql.parse(xmlcol)
```

本示例将 XMLTEXT 表的 `xmlcol` 列转换为经过分析的数据，并将其存储在 `xmldoc` 列中。

## 插入 XML 文档

使用 `parse()` 方法插入 XML 文档，该方法将 XML 文档作为参数并返回 `sybase.aseutils.SybXmlStream`。

Adaptive Server 在 `image` 或 `text` 数据和 `InputStream` 之间有隐式映射。无需进行任何转换，即可将 `image` 或 `text` 列传递到 `parse()`。`parse()` UDF 分析文档并返回 `sybase.ase.SybXmlStream`，Adaptive Server 使用它来将数据写入 `image` 列。Adaptive Server 只将此数据写入 `image` 列，而不写入 `text` 列。下面是一个 `insert` 语句，其中的 `XMLDAT` 是一个含 `image` 列 `xmldoc` 的表：

```
insert XMLDAT
values (... ,
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>"),
...)
```

## 更新 XML 文档

若要更新文档，请删除原始数据，然后插入新数据。对文档或文档一部分更新的次数要少于读取的次数。更新类似于：

```
update XMLDAT
set xmldoc =
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>")
```

## 删除 XML 文档

删除 XML 文档与删除任何文本列类似。例如，若要删除名为 `XMLDAT` 的表，请输入：

```
delete XMLDAT
```

## 使用 XQL

XML 查询语言 (XQL) 被设计成 XML 通用查询语言。XQL 是基于路径的查询语言，它用于寻址和过滤 XML 文档的元素和文本，是 `SPath` 的自然扩展。XQL 提供简洁、易懂的符号，用于指向特定的元素及搜索具有特定特性的节点。XQL 导航在 XML 树中遍历元素。

最常见的 XQL 运算符包括：

- 子运算符 / — 表示层次。下面的示例返回 `<book>` 元素，它们是 `xmlimage` 表 `xmlcol` 列中 `<bookstore>` 元素的子项：

```
select
  com.sybase.xml.xql.Xql.query("/bookstore/book",
    xmlcol)
from xmlimage
```

- 子代运算符 // — 表示查询将在任意数量的介入级进行搜索。也就是说，使用子代运算符的搜索将在 XML 结构的任何级查找元素的位置。下面的查询将查找在 `<excerpt>` 元素中出现的所有 `<emph>` 元素实例：

```
select com.sybase.xml.xql.Xql.query
  ("/bookstore/book/excerpt//emph",xmlcol)
from xmlimage
```

```
<xql_result>
  <emph>I</emph>
</xql_result>
```

- 等号运算符 = — 指定元素的内容或属性的值。下面的查询查找“last-name = Bob”的所有示例：

```
select com.sybase.xml.xql.Xql.query
  ("/bookstore/book/author[last-name='Bob']", xmlcol)
from xmlimage
```

```
<xql_result>
  <author>
    <first-name>Joe</first-name>
    <last-name>Bob</last-name>
    <award>Trenton Literary Review Honorable Mention</award>
  </author> <author>
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    <publication>Selected Short Stories of
    <first-name>Mary</first-name>
    <last-name>Bob</last-name></publication></author>
  <author>
    <first-name>Toni</first-name>
    <last-name>Bob</last-name>
    <degree from=Trenton U>B.A.</degree>
    <degree from=Harvard>Ph.D.</degree>
    <award>Pulizer</award>
    <publication>Still in Trenton</publication>
```

```

    <publication>Trenton Forever</publication></author>
</xql_result>

```

- 过滤运算符 `[]` — 根据括号内的条件过滤其左边的一组节点。下例查找在 `book` 元素中列出的所有名为 `Mary` 的作者：

```

select com.sybase.xml.xql.Xql.query
  ("/bookstore/book[author/first-name = 'Mary']", xmlcol)
from xmlimage
<xql_result>
  <book style=textbook>
    <title>History of Trenton</title>
    <author>
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    <publication>Selected Short Stories of
      <first-name>Mary</first-name>
      <last-name>Bob</last-name></publication></author>
    <price>55</price></book>

```

- 下标运算符 `[index_ordinal]` — 查找元素的特定实例。下例查找在 XML 文档中列出的第二本书。要记住，XQL 是以零为基的，因此从 0 开始编号：

```

select com.sybase.xml.xql.Xql.query("/bookstore/book[1]", xmlcol)
from xmlimage
Query returned true and the result is
<xql_result>
  <book style=textbook>
    <title>History of Trenton</title>
    <author>
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    <publication>Selected Short Stories of
      <first-name>Mary</first-name>
      <last-name>Bob</last-name></publication></author>
    <price>55</price></book>
</xql_result>

```

- 布尔表达式 — 可以在过滤运算符中使用布尔表达式。例如，下面的查询返回至少包含一个 `<degree>` 和一个 `<award>` 的所有 `<authors>` 元素。

```

select com.sybase.xml.xql.Xql.query
  ("/bookstore/book/author[degree and award]", xmlcol)
from xmlimage
<xql_result>

```

```

    <author>
    <first-name>Toni</first-name>
    <last-name>Bob</last-name>
    <degree from=Trenton U>B.A.</degree>
    <degree from=Harvard>Ph.D.</degree>
    <award>Pulizer</award>
    <publication>Still in Trenton</publication>
    <publication>Trenton Forever</publication></author>
</xql_result>

```

## 影响性能的查询结构

本节描述以不同方式使用基于Java的XQL处理器的示例。

### 示例

**where** 子句在查询中的位置会对处理产生影响。例如，下面的查询选择所有作者名为 Mary 的书：

```

select com.sybase.xml.xql.Xql.query
    ("/bookstore/book[author/first-name = 'Mary']", xmlcol)
from XMLDAT
where
    com.sybase.xml.xql.Xql.query
    ("/bookstore/book
    [author/first-name= 'Mary']", xmlcol) !=
    convert(com.sybase.xml.xql.Xql, null)>>EmptyResult
-----
<xql_result ><book style="textbook">
    <title>History of Trenton</title>
    <author>
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    <publication>
    Selected Short Stories of
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    </publication>
    </author>
    <price>55</price>
</book></xql_result>

```

## 基于 Java 的 XQL 处理器的其它用法

**注释** Sybase 不支持 XQL 软件包的这些用法。这些用法要求 JDK 1.2 或更高版本。

---

可以使用独立的应用程序 `com.sybase.xml.xql.XqlDriver` 从命令行查询 XML 文档。

可以使用 `com.sybase.xml.xql.Xql` 中提供的 Java 软件包方法来查询 Java 应用程序中的 XML 文档。利用 Java VM 功能，也可使用这些 Java 软件包方法查询 Adaptive Server 中的 XML 文档。

`com.sybase.xml.xql.XqlDriver` 只能分析和查询在本地系统中保存为文件的 XML 文档。不能使用 `com.sybase.xml.xql.XqlDriver` 分析或查询存储在数据库中或网络上的 XML 文档。

`com.sybase.xml.xql.XqlDriver` 对于开发 XQL 脚本和学习 XQL 非常有用。但是，Sybase 建议只将 `com.sybase.xml.xql.XqlDriver` 作为独立程序使用，而不要将其作为其它 Java 应用程序的一部分，因为 `com.sybase.xml.xql.XqlDriver` 包含 `main()` 方法。一个 Java 程序只能包含一个 `main()` 方法，如果将 `com.sybase.xml.xql.XqlDriver` 包括在其它包含 `main()` 的 Java 程序中，应用程序将试图执行这两个 `main()` 方法，从而导致 Java 中出现错误。

Sybase 建议应用程序使用 `com.sybase.xml.xql.Xql` 类与 XML 查询引擎对接。第 171 页的“[com.sybase.xml.xql.Xql 中的方法](#)”一节中说明了此类的方法。

## com.sybase.xml.xql.XqlDriver 语法

`com.sybase.xml.xql.XqlDriver` 的语法为：

```
java com.sybase.xml.xql.XqlDriver
-qstring XQL_query
-validate true | false
-infile string
-outfile string
-help
-saxparser string
```



其中：

- `qstring` 指定要运行的 XQL 查询。
- `validate` 检查 XML 文档的有效性。
- `infile` 是要查询的 XML 文档。
- `outfile` 是操作系统文件，将在其中存储已分析的 XML 文档。
- `help` 显示 `com.sybase.xml.xql.XqlDriver` 语法。
- `saxparser` 指定符合 SAX 2.0 要求的 CLASSPATH 分析程序的名称。

## 示例查询

下面的查询从 *bookstore.xml* 中选择所有书名：

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore/book/title"
    -infile bookstore.xml
```

Query returned true and the result is

```
<xql_result>
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
</xql_result>
```

下例列出 *bookstore.xml* 中所有作者的名字。XQL 使用以零为基的编号系统；也就是说，“0”指定某一元素在文件中的第一次出现。

```
java com.sybase.xml.xql.XqlDriver
    -qstring "/bookstore/book/author/first-name[0]"
    -infile bookstore.xml
```

Query returned true and the result is

```
<xql_result>
    <first-name>Joe</first-name>
    <first-name>Mary</first-name>
    <first-name>Toni</first-name>
</xql_result>
```

下例列出 *bookstore.xml* 中姓氏为 “Bob” 的所有作者：

```
java com.sybase.xml.xql.XqlDriver
-qstring "/bookstore/book/author[last-name='Bob'] "
-infile bookstore.xml
```

Query returned true and the result is

```
<xql_result>
  <author>
    <first-name>Joe</first-name>
    <last-name>Bob</last-name>
    <award>Trenton Literary Review Honorable Mention</award></author>
  <author>
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    <publication>Selected Short Stories of
    <first-name>Mary</first-name>
    <last-name>Bob</last-name></publication></author>
  <author>
    <first-name>Toni</first-name>
    <last-name>Bob</last-name>
    <degree from=Trenton U>B.A.</degree>
    <degree from=Harvard>Ph.D.</degree>
    <award>Pulizer</award>
    <publication>Still in Trenton</publication>
    <publication>Trenton Forever</publication></author>
</xql_result>
```

## 验证文档

**valid** 选项将调用分析程序，以确保所查询的 XML 文档符合其 DTD。在运行 **validate** 选项之前，独立的 XML 文档必须具有一个有效的 DTD。

例如，下面的命令将确保 *bookstore.xml* 文档符合其 DTD：

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore" -validate
-infile bookstore.xml
```

## 将基于Java的XQL处理器用于独立应用程序

可以使用XQL开发独立的应用程序、JDBC客户端、JavaBean以及EJB来处理XML数据。`com.sybase.xml.xql.Xql`中的`query()`和`parse()`方法可用于查询和分析XML文档。因为可以编写独立的应用程序，所以不必依赖Adaptive Server来提供结果集。相反，您可以查询存储为操作系统文件或存储在Web上的XML文档。

### 独立应用程序示例

下例使用`FileInputStream()`查询读取`bookstore.xml`，使用`URI()`方法读取包含书店中所有书籍信息的Web页`bookstore.xml`。

```
String result;
FileInputStream XmlFile = new FileInputStream("bookstore.xml");
if ((result =
    Xql.query("/bookstore/book/author/first-name", XmlFile))
    != Xql.EmptyResult )
{
    System.out.println(result);
}else{
    System.out.println("Query returned false\n");
}
URI _uri = new URI("http://mybookstore/bookstore.xml");
if ((result =
    Xql.query("/bookstore/book/author/first-name", uri.openStream()))
    != Xql.EmptyResult )
{
    System.out.println(result);
}else{
    System.out.println("Query returned false\n");}
```

### EJB 示例

可以编写EJB代码段，以将其用作EJB服务器上的查询引擎。

下列代码段包含一个称为`XmlBean`的EJB。`XmlBean`包含`query()`方法，它允许您查询Web上的任何XML文档。在此组件中，`query()`首先创建一个`XmlDoc`对象，然后查询文档。

远程接口如下所示:

```
public interface XmlBean extends javax.ejb.EJBObject
{
    /**
     * XQL 方法*/
    public String XQL(String query, URI location)
        throws java.rmi.RemoteException;}
}
```

Bean 实现如下所示:

```
public class XmlBean extends java.lang.Object implements
javax.ejb.SessionBean
{
    ....
    /**
     * XQL 方法
     */
    public String XQL(String query, java.net.URI location) throws
        java.rmi.RemoteException
    {
        try {
            String result;
            if((result =
                Xql.query(query, location.openStream())) !=
                Xql.EmptyResult)
            {
                return (result);
            }else{
                return (null);
            }
        }catch(Exception e){
            throw new java.rmi.RemoteException(e.getMessage());
        }
    }
    ....
}
```

客户端代码如下所示:

```
....Context ctx = getInitialContext();
// 在 Jaguar 中创建该类的实例
XmlBeanHome beanHome =
    (XmlBeanHome)ctx.lookup("XmlBean");
XmlBean _xmlBean = (XmlBean)beanHome.create();
URI u = new URI("http://mywebsite/bookstore.xml");
String res= xmlBean.XQL("/bookstore/book/author/first-name",u);
```

## com.sybase.xml.xql.Xql 中的方法

下列方法是 com.sybase.xml.xql.Xql 特有的。

### parse(String xmlDoc)

**说明** 将 Java 字符串作为参数并返回 *SybXmlStream*。可用它来查询使用 XQL 的文档。

**语法** `parse(String xml_document)`

其中：

- `String` 是 Java 字符串。
- `xml_document` 是包含该字符串的 XML 文档。

**示例** 下面的示例返回 *SybXmlStream*：

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>")
```

**用法** 分析程序不会：

- 验证文档（如果提供了 DTD）。
- 分析任何外部 DTD
- 执行任何外部链接（例如 XLink）
- 在 IDREF 中导航

### parse(InputStream xml\_document, boolean validate)

**说明** 以 *InputStream* 和布尔标志作为参数。该标志表示分析程序应按照指定的 DTD 验证文档。返回 *SybXmlStream*。可用它来查询使用 XQL 的文档。

**语法** `parse(InputStream xml_document, boolean validate)`

其中：

- *InputStream* 是输入流。
- `xml_document` 是输入流源自的 XML 文档。

- 示例                      下面的示例返回 *SybXmlStream*
- ```
SybXmlStream is = Xql.parse(new  
FileInputStream("file.xml"), true);
```
- 用法
- 标志值为 **true** 表示分析程序应按照指定的 DTD 验证文档。
  - 标志值为 **false** 表示分析程序不按照指定的 DTD 验证文档。
  - 分析程序不会：
    - 分析任何外部 DTD
    - 执行任何外部链接（例如 XLink）
    - 在 IDREF 中导航

## query(String query, String xmlDoc)

说明                      查询 XML 文档。使用 XML 文档作为输入参数。

语法                      **query(String query,String xmlDoc)**

其中：

- *String query* 是要搜索的字符串。
- *String xmldoc* 是要查询的 XML 文档。

示例                      下面的语句以 Java 字符串的形式返回结果：

```
String result= Xql.query("/bookstore/book/author",  
"<xml>...</xml>");
```

用法                      返回一个 Java 字符串。

## query(String query, InputStream xmlDoc)

**说明** 使用输入流作为第二参数来查询 XML 文档。

**语法** `query(String query,InputStream xmlDoc)`

其中：

- *String query* 是要搜索的字符串。
- *Input Stream xmlDoc* 是要查询的 XML 文档。

**示例** 下例将在书店中查询 *bookstore.Xql* 中列出的作者。

```
FileInputStream xmlStream = new FileInputStream("doc.xml");
String result = Xql.query("/bookstore/book/author", xmlStream);
```

下例使用 URI 作为搜索参数来查询 Web 上的 XML 文档：

```
URI xmlURI = new URI("http://mywebsite/doc.xml");
String result = Xql.query("/bookstore/book/author", xmlURI.openStream());
```

**用法** 返回一个 Java 字符串。

## query(String query, SybXmlStream xmlDoc)

**说明** 使用已分析的 XML 文档作为第二参数来查询 XML 文档。

**语法** `query(String query, SybXmlStream )`

其中：

- *String query* 是要搜索的字符串。
- *xmlDoc* 是要查询的已分析 XML 文档。

**示例** 下例将在书店中查询 *bookstore.Xml* 中列出的作者。

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>");
String result = Xql.query("/bookstore/book/author",xmlStream);
```

## sybase.aseutils.SybXmlStream

**说明** 定义查询时 `InputStream` 需要用来访问已分析 XML 数据的接口。

**语法** `sybase.aseutils.SybXmlStream` 接口

## com.sybase.xml.xql.store.SybMemXmlStream

|    |                                                    |
|----|----------------------------------------------------|
| 说明 | 将已分析的 XML 文档保留在主内存中；它是 Sybase 提供的 SybXMLStream 实现。 |
| 语法 | com.sybase.xml.xql.store.SybMemXmlStream           |
| 用法 | 分析完 XML 文档后，parse() 方法将返回一个 SybMemXmlStream 实例。    |

## com.sybase.xml.xql.store.SybFileXmlStream

|    |                                                                                                    |
|----|----------------------------------------------------------------------------------------------------|
| 说明 | 用于查询存储着已分析 XML 文档的文件。                                                                              |
| 语法 | com.sybase.xml.xql.store.SybFileXmlStream {file_name}<br>其中 <i>file_name</i> 是存储着已分析 XML 文档的文件的名称。 |
| 示例 | 在下例中，RandomAccessFile 的一个成员将读取文件并定位数据流：                                                            |

```
SybXmlStream xis = Xql.parse("<xml>..</xml>");  
FileOutputStream ofs = new FileOutputStream("xml.data");  
( (SybMemXmlStream) xis ).writeToFile (ofs);  
  
SybXmlStream is = new SybFileXmlStream("xml.data");  
String result = Xql.query("/bookstore/book/author", is);
```

## setParser(String parserName)

|    |                                                                 |
|----|-----------------------------------------------------------------|
| 说明 | 此静态方法指定 parse 方法应使用的分析程序。应确保分析程序类能通过 CLASSPATH 访问并与 SAX 2.0 兼容。 |
| 语法 | setParser (String parserName)<br>其中 <i>string</i> 是分析程序类的名称。    |
| 示例 |                                                                 |

```
Xql.setParser ("com.yourcompany.parser")
```



## resetParser

**说明** 此静态方法将分析程序重置为 Sybase 提供的缺省分析程序 (*xerces.jar*)。

**语法** `resetParser`

**示例** 下例将分析程序重置为 Sybase 的缺省分析程序。

```
xql.resetParser()
```



# 在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移

## 简介

基于 Java 的 XQL 处理器和本机 XML 处理器都实现查询语言且都返回经过分析的文档，但所使用的函数和方法有所不同。

- 本机 XML 处理器实现 XML 查询语言。它提供内置函数 `xmlparse`，该函数返回经过分析的文档，这种文档适合用 `xmlextract` 和 `xmltext` 内置函数进行有效处理。
- 基于 Java 的 XQL 处理器是一种实现 XQL 查询语言的早期工具。它提供 Java 方法 `com.sybase.xml.xql.Xql.parse`，该方法返回已分析的文档，这种文档是 `sybase.aseutils.SybXmlStream` 对象，适合用 `com.sybase.xml.xql.Xql.query` 方法进行处理。

如果要在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移文档，应注意以下可能性和限制：

- 文本格式的文档可由基于 Java 的 XQL 处理器或由本机 XML 处理器直接处理。
- `com.sybase.xml.xql.Xql.parse` 所生成的 `sybase.aseutils.SybXmlStream` 文档只能由基于 Java 的 XQL 处理器来处理。它们不能用内置函数 `xmlextract` 或 `xmltest` 处理。
- `xmlparse` 内置函数所生成的已分析文档只能由 `xmlextract` 和 `xmltest` 内置函数来处理。它们不能用基于 Java 的 XQL 处理器处理。

## 迁移文档和查询

以下各节介绍用于在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移文档和查询的技术。

### 在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移文档

可以使用两种方法在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移文档：

- 可以使用文本格式的文档（如果可行）。
- 可以从经过分析的文档生成文本格式的文档。

### 在基于 Java 的 XQL 处理器和本机 XML 处理器之间迁移文本文档

假设您有如下表，并在其 `xmlsource` 列中存储了文本格式的文档：

```
create table xmltab (xmlsource text, xmlindexed image)
```

如果要使用 `xmlextract` 和 `xmltest` 内置函数来通过本机 XML 处理器处理这些文档，则可以按照如下方法更新该表：

```
update xmltab  
set xmlindexed = xmlparse(xmlsource)
```

如果要使用 `com.sybase.xml.xql.Xql.query` 方法来通过基于 Java 的 XQL 处理器处理这些文档，则可以按照如下方法更新该表：

```
update xmltab  
set xmlindexed  
= com.sybase.xml.xql.Xql.parse(xmlsource)
```

### 从重新生成的副本迁移文档

假设您使用本机 XML 处理器的 `xmlparse` 内置函数或使用基于 Java 的 XQL 处理器的 `com.sybase.xml.xql.Xql.parse` 方法仅存储了一些已分析的文档。例如，在诸如下表的表中可能会有此类文档：

```
create table xmltab (xmlindexed image)
```

如果要为此类文档重新生成文本，可以改变此表，增加一个文本列：

```
alter table xmltab add xmlsource text null
```

## 从基于 Java 的 XQL 处理器重新生成文本文档

本节演示从为基于 Java 的 XQL 处理器生成的格式重新生成文本格式的文档。

如果 `xmlindexed` 列包含 `com.sybase.xmlxql.Xql.parse` 所生成的 `sybase.aseutils.SybXmlStream` 数据，则可以用以下 SQL 语句在新的 `xmlsource` 列中重新生成文本格式的文档：

```
update xmltab
set xmlsource
    = xmlextract("/xql_result/*",
        com.sybase.xml.xql.Xql.query("/",xmlindexed) )
```

下面的语句分两个步骤生成文本格式的文档：

- 1 用 “/” 查询进行的 `com.sybase.xml.xql.Xql.query` 调用生成文本格式的文档，它包含在 XML 标记 `<xql_result>...</xql_result>` 中。
- 2 用 “/xql\_result/\*” 查询进行的 `xmlextract` 调用删除 `<xql_result>...</xql_result>` 标记，并返回文本格式的初始文档。

然后可以使用 `xmlextract` 和 `xmltest` 内置函数来直接通过本机 XML 处理器处理 `xmlsource` 列，也可以更新本机 XML 处理器的 `xmlindexed` 列，如下所示：

```
update xmltab
set xmlindexed = xmlparse(xmlsource)
```

如果不想添加 `xmlsource` 列，则可以合并这些步骤，如以下 SQL 语句所示：

```
update xmltab
set xmlindexed
    = xmlparse(xmlextract("/xql_result/*",
        com.sybase.xml.xql.Xql.query("/",xmlindexed) ) )
```

在执行此 `update` 语句之前，`xmlindexed` 列包含 `sybase.aseutils.SybXmlStream` 格式的文档，它们由 `com.sybase.xml.xql.Xql.parse` 方法生成。在执行 `update` 语句之后，该列包含已分析的文档，这些文档适合用 `xmlextract` 和 `xmlparse` 方法来处理。

## 从本机 XML 处理器重新生成文本文档

本节演示从为本机 XML 处理器生成的格式重新生成文本格式的文档。

如果 `xmlindexed` 列包含 `xmlparse` 函数生成的数据，则可以用以下 SQL 语句在新的 `xmlsource` 列中重新生成文本格式的文档：

```
update xmltab
set xmlsource = xmlextract("/", xmlindexed)
```

然后可以

- 使用 `com.sybase.xml.xql.Xql.query` 来直接通过基于 Java 的 XQL 处理器处理 `xmlsource` 列，或者
- 使用以下 `update` 语句以适合用基于 Java 的 XQL 处理器进行处理的已分析格式来更新 `xmlindexed` 列：

```
update xmltab
set xmlindexed
= com.sybase.xml.xql.Xql.parse(xmlsource)
```

如果不想添加 `xmlsource` 列，则可以合并这些步骤，如以下 SQL 语句所示：

```
update xmltab
set xmlindexed
= com.sybase.xml.xql.Xql.parse
(xmlextract("/", xmlindexed))
```

在执行此 `update` 语句之前，`xmlindexed` 列包含已分析的文档，这些文档由 `xmlparse` 内置函数生成。在执行 `update` 语句之后，该列包含由 `com.sybase.xml.xql.Xql.parse` 生成的已分析文档，它们适合用 `com.sybase.xml.xql.Xql.query` 来处理。

## 在本机 XML 处理器和基于 Java 的 XQL 处理器之间迁移查询

基于 Java 的 XQL 处理器实现的 XQL 语言和本机 XML 处理器实现的 XML 查询语言均基于 XPath 语言。这两者之间有两个主要差异：

- 在 XML 查询语言中下标从 “1” 开始，而在 XQL 语言中下标从 “0” 开始。
- 基于 Java 的 XQL 处理器将结果包含在 “`<xql_result>...</xql_result>`” 标记中返回，而本机 XML 处理器则非如此。

## 示例表

本节显示了一个 XML 示例文档 `depts.xml`，该示例文档说明了使用 `xmltable()` 实现的一个应用程序。

```
<sample>
<depts>
  <dept>
    <dept_id>D123</dept_id>
    <dept_name>Main</dept_name>
  <emps>
    <emp>
      <emp_id>E123</emp_id>
      <emp_name>Alex Allen</emp_name>
      <salary>912.34</salary>
      <phones>
        <phone><phone_no>510.555.1987</phone_no></phone>
        <phone><phone_no>510.555.1867</phone_no></phone>
      </phones>
    </emp>
    <emp>
      <emp_id>E234</emp_id>
      <emp_name>Bruce Baker</emp_name>
      <salary>923.45</salary>
      <phones>
        <phone><phone_no>230.555.2333</phone_no></phone>
      </phones>
    </emp>
    <emp>
      <emp_id>E345</emp_id>
      <emp_name>Carl Curtis</emp_name>
      <salary>934.56</salary>
      <phones>
        <phone><phone_no>408.555.3123</phone_no></phone>
        <phone><phone_no>415.555.3987</phone_no></phone>
        <phone><phone_no>650.555.3777</phone_no></phone>
      </phones>
    </emp>
  </emps>
</depts>
</sample>
```

```

<emps_summary>
<salary_summary>
  <max_salary>934.56</max_salary>
  <total_salary>2770.35</total_salary>
</salary_summary>
</emps_summary>
<projects>
<project>
  <project_id>PABC</project_id>
  <budget>598.65</budget>
</project>
<project>
  <project_id>PBDC</project_id>
  <budget>587.65</budget>
</project>
<project>
  <project_id>PCDE</project_id>
  <budget>576.54</budget>
</project>

</projects>
<projects_summary>
<budget_summary>
  <max_budget>598.76</max_budget>
  <total_budget>1762.95</total_budget>
</budget_summary>
</projects_summary>
</dept>
<dept>
  <dept_id>D234</dept_id>
  <dept_name>Auxiliary</dept_name>
<emps>
  <emp>
    <emp_id>E345</emp_id>
    <emp_name>Don Davis</emp_name>
    <salary>945.67</salary>
    <phones>
<phone><phone_no>650.555.5001</phone_no></phone>
    </phones>
  <emp>
    <emp_id>E345</emp_id>
    <emp_name>Earl Evans</emp_name>
    <phones>
<phone><phone_no>650.555.5001</phone_no></phone>
    </phones>
  
```



```

    </emp>
  </emps>
<emps_summary>
<salary_summary>
  <max_salary>945.67</max_salary>
  <total_salary>945.67</total_salary>
</salary_summary>

</emps_summary>
<projects>
  <project>

    <project>
      <project_id>PDEF</project_id>
    </project>
    <project>
      <project_id>PEFG</project_id>
      <budget>554.32</budget>
    </project>
  </project>
  </projects>
  <projects_summary>
  <budget_summary>
    <max_budget>554.32</max_budget>
    <total_budget>554.32</total_budget>
  </budget_summary>
  </projects_summary>
</dept>
</dept>
<dept>
  <dept_id>D345</dept_id>
  <dept_name>Repair</dept_name>
  <emps>
    <emp>
      <emp_id>E678</emp_id>
      <emp_name>Fred Frank</emp_name>
      <salary>967.89</salary>
      <phones>
        <phone><phone_no>408.555.6111</phone_no></phone>
      </phones>
    </emp>
    <emp>>
      <emp_id>E789</emp_id>
      <emp_name>George Gordon</emp_name>

      <salary>978.90</salary>

```

```

    <phones>
    <phone><phone_no>510.555.7654</phone_no></phone>
    </phones>
</emp>
<emp>
    <emp_id>E901</emp_id>
    <emp_name>Hank Hartley</emp_name>
    <salary>990.12</salary>
    <phones\>
</emp>
<emp>
    <emp_id>E678</emp_id>
    <emp_name>Isaak Idle</emp_name>
    <salary>990.12</salary>
    <phones>
    <phone><phone_no>925.555.9991</phone_no></phone>
    <phone><phone_no>650.555.9992</phone_no></phone>
    <phone><phone_no>415.555.9993</phone_no></phone>
    </phones>
</emp>
    <emps>
    <emps_summary>
<salary_summary>
    <max_salary>990.12</max_salary>
    <total_salary>2936.91</total_salary>
</salary_summary>
</emps_summary>
<projects>
    <project>
    <project_id>PFGH</project_id>
    <budget>543.21</budget>
    </project>
    <project>
    <project_id>PGHI</project_id>
    </project>
    <project>
    <project_id>PHIJ</project_id>
    <budget>521.09</budget>
    </project>
    </project>
</projects>
    <projects_summary>
<budget_summary>
    <max_budget>543.21</max_budget>
    <total_budget>1064.30</total_budget>

```

```

</budget_summary>
</projects_summary>
</dept>
</depts>
</sample>

```

## 使用 *depts* 文档

*depts* 文档存储在 `sample_docs` 表的一个新行中，该表位于[附录 A](#) “[sample\\_docs 示例表](#)”要在示例中引用此文档，请使用以下语句：

```

declare @dept_doc xml
select @dept_doc from sample_docs where name_doc='depts'

```

## *depts* 文档的结构

*depts* 文档的结构如下：

```

<depts>
  <dept>
    <emps> - repeats under <depts>
    <emp> - one for each <dept>
      <emp_id> - one for each <emp>
      <emp_name> - one for each <emp>
      <phones> - one for each <emp>
        <phone> - repeats under <phones>
          <phone_no> - one for each <phone>
    <projects> - one for each <dept>
    <project> - repeats for each under <projects>
      <project_id> - one for each <project>
      <dept_id> - one for each <project>

```

## 用 depts 文档创建 SQL 表

### 对来自 depts 文档的数据进行规范化

可以通过规范化将这些数据组合成 SQL 表。例如：

- **depts** — 每个 `<dept>` 元素在该表中都有对应的一行，该元素则包含 `<dept_id>` 和 `<dept_name>`。
- **emps** — 每个 `<emp>` 元素在该表中都有对应的一行，该元素则包含 `<emp_id>` 和 `<emp_name>`，以及其所隶属的 `<dept_id>` 元素。
- **emp\_phones** — 每个 `<phone>` 元素在该表中都有对应的一行，该元素则包含 `<phone_no>` 和所有这些元素所隶属的 `<dept_id>` 元素。
- **projects** — 每个 `<project>` 元素在该表中都有对应的一行，该元素则包含 `<project_id>` 和 `<budget>` 元素，以及其所隶属的 `<dept_id>` 元素。

### 使用 select 生成表

本节中生成的所有表，除 **depts** 以外，都有使用 XPath 祖先表示法引用以下元素的列模式：

- 叶元素，如 `<projects>` 下的 `<project>`，或 `<emp>` 下的 `<salary>`。
- 包含定义表的元素的元素，例如包含 `<emp_id>` 的 `<emp>`。

这种表示法会“展平”嵌入的数据。有关展平 XML 数据的详细信息，请参见第 7 章“[xmltable\(\)](#)”。

#### emps 表

在下面的 `select` 语句中，`dept_id` 列的列模式引用包含当前 `<emp>` 的 `<dept>` 中的 `<dept_id>` 元素。

```
declare @ dept_doc xml
select @dept_doc = doc from sample_docs where name_doc = 'depts'
select * into emps from xmltable('//emp' passing @dept_doc
    columns emp_id char(4),
            emp_name varchar(50),
            salary money,
            dept_id char(4) pattern '../../dept_id') as dept_extract
select * from emps
-----
emp_id      emp_name      salary      dept_id
-----
```

E123	Alex Allen	912.34	D123
E234	Bruce Baker	923.45	D123
E345	Carl Curtis	934.56	D123
E456	Don Davis	945.67	D234
E567	Earl Evans	956.78	D234
E678	Fred Frank	967.89	D345
E789	George Gordon	978.90	D345
E890	Hank Hartley	NULL	D345
E901	Isaak Idle	990.12	D345

**phones 表**

在 `phones` 表中，`emp_id` 列的列模式引用包含当前元素 `<phone>` 的 `<emp>` 中的 `<emp_id>` 元素。

```
declare @ dept_doc xml
select @ dept_doc
      = doc from sample_docs where name_doc='depts'
select * into phones
      from xmltable('//phone' passing @ dept_doc
                    columns emp_id char(4), '../.. /emp_id'
                          phone_no varchar(50)) as dept_extract
select * from phones
```

emp_id	phone_no
E123	510.555.1987
E123	510.555.1876
E234	203.555.2333
E345	408.555.3123
E345	415.555.3987
E345	650.555.3777
E567	650.555.5001
E678	408.555.6111
E678	408.555.6222
E789	510.555.7654
E901	925.555.9991
E901	650.555.9992
E901	415.555.9993

projects 表

在 projects 表中，dept\_id 列的列模式引用包含当前 <project> 的 <dept> 中的 <dept\_id> 元素。

```
declare @ dept_doc xml
select @ dept_doc
      = doc from sample_docs where name_doc='depts'
select * into projects
      from xmltable('//project' passing @ dept_doc
                    columns project_id char(4),
                           budget money,
                           dept_id char(4)pattern '../..dept_id')
      as dept_extract
select * from projects
-----
project_id      budget      dept_id
-----
PABC           598.76      D123
PBCD           587.65      D123
PCDE           576.54      D123
PDEF           565.43      D234
PEFG           554.32      D234
PFGH           543.21      D345
PGHI           NULL        D345
PHIJ           521.09      D345
```

depts 表

```
declare @ dept_doc xml
select @ dept_doc
      = doc from sample_docs where name_doc='depts'
select * into depts
      from xmltable('//dept' passing @ dept_doc
                    columns dept_id char(4),
                           dept_name char(4)) as dept_extract
select * from depts
-----
dept_id      dept_name
-----
D123         Main
D234         Auxiliary
D345         Repair
```

# 索引

## 符号

- ::= (BNF 表示法)
  - SQL 语句中的 xvi
- { } (大括号)
  - SQL 语句中的 xvi
- , (逗号)
  - SQL 语句中的 xvi
- () (小括号)
  - SQL 语句中的 xv
- [ ] (中括号)
  - SQL 语句中的 xvi
- , 安装
  - XML 处理器 155
- @@error
  - 全局变量 12
  - 最近一次错误的错误号 12
- \any/all 定量判定子查询, 不能将 for xml 子查询用作 60

## 数字

- 16 位值, 代理对 106

## 英文

- alter table
  - 命令 161
- ample\_docs 表
  - titles 表, XML 表示形式 140
- aseutils 方法, com.sybase.xml.xql.Xql
  - 方法, 特有的 173
- at 子句命令 147
- Backus Naur Form (BNF) 表示法 xv, xvi
- base64, SQLX 选项 83
- BCP, 用于传送数据 106

- binary
  - SQLX 选项 83
  - 数据类型 83
  - 选项 83
- binary 选项 37
- bookstore, 示例文档 136
- bookstore.xml
  - DTD 符合 168
  - validate 命令 168
  - XML 示例 167
  - 网页 169
  - 文件名 167
  - 作者示例 168
- char 数据类型 2, 98
- CIS (组件集成服务) 145
- CLASSPATH
  - UNIX 和 NT 的环境变量 159
  - xerces.jar, xml.zip, runtime.zip 159
  - 独立程序 159
  - 环境变量 159
- classpath 环境变量, 设置 159
- columnstyle 选项 37
- columnstyle, SQLX 选项 83, 92
- com 174
- com.sybase.xml.xql.store 方法 174
- com.sybase.xml.xql.store.SybMemXmlStream, XQL 接口 174
- com.sybase.xml.xql.Xql
  - 方法, 特有的 171, 172, 174, 175
- com.sybase.xml.xql.XqlDriver
  - 本地文件 166
  - 查询 XML 文档 166
  - 独立程序 166
  - 使用 166
  - 语法 166
- com.sybase.xml.xql.XqlDriver, 示例 167
- concat 字符串函数 47, 51
- concat, XPath 字符串函数 47

- concat, 函数 51
- create proxy table 命令 147
- create proxy table, 命令 147
- create view 命令, 不能使用 for xml 子查询 60
- CTLIB, 用于传送数据 106
- DB 环境变量 156
- debug 命令 167
- declare cursor 命令, 不能使用 for xml 子查询 60
- default xml sort order, sp\_configure 中 115
- default xml sort order, 修改 115
- disable\_unichar\_sending 属性 107
- DTD 7
  - #IMPLIED 7
  - #PCDATA DTD 元素 7
  - ATTLIST 7
  - ELEMENT 7
  - 不是在所有文档中都要求 8
  - 对于有效的 XML 文档 8
  - 加号 (+) 7
  - 嵌入的 8
  - 问号 (?) 7
  - 星号 (\*) 7
- DTD 和模式, xmlvalidate 31
- dtdvalidate 选项 37
- dtdvalidate, 验证选项 28
- EFS
  - 示例 146
- EFS 访问 145
- EJB 示例 169
- entitize
  - SQLX 选项 84
  - 选项 37, 82
- exists/not exists 定量判定子查询, 不能将 for xml 子查询用作 60
- exists/not exists, 定量判定子查询 60
- FileInputStream(), 命令 169
- for xml
  - select 语句的扩展 1
  - SQLX-XML 格式 92
  - 说明 56
  - 在数据映射中 92
  - 子句 81, 86
  - 子句, 语法和示例 55
- for xml all 61
- for xml all 选项 86
- for xml schema 61
- for xml select 命令, 不能使用 for xml 子查询 60
- for xml 的 header 选项 109
- for xml 子查询 59
  - 示例 60
  - 说明 59
  - 异常 60
  - 语法 59
- for xml 子句
  - 不能是相关子查询 60
  - 不能用作定量判定子查询 60
  - 不能在命令中使用 60
  - 不能在嵌套的标量子查询中 60
  - 扩展 61
  - 扩展示例 61
  - 扩展说明 61
  - 生成非 ASCII 数据 105
  - 示例 58, 109
  - 选项 58
  - 异常 58
  - 与 isql 搭配使用 81
  - 语法 55
- for xml 子句扩展异常 61
- for xml, Unicode 107
- format {yes | no}
  - SQLX 选项 85
- format 选项 37
- forsqlcreate 安装 155
- forsqlcreatej
  - 函数说明 70
- forsqlinsertj 70
  - 说明, 语法, 示例 70
- forsqlscript 安装 155
- forsqlscriptj
  - 函数 70
  - 说明, 语法, 示例 70
- forxml 函数 86
- forxmlallj
  - 函数 65
  - 说明, 语法, 示例 65
- forxmlallj 安装 155
- forxmlallj 函数 99
- forxmldtd 安装 155



- forxmldtdj
  - 函数 65
  - 说明, 语法, 示例 65
- forxmlinsertj
  - 示例 72
  - 说明 71
  - 选项 71
  - 异常 72
  - 语法 70
- forxmlj 155
  - SQLX-XML 格式 92
  - 函数 81
  - 基于 Java 的函数 65
- forxmlmultiplej
  - 说明 79
  - 选项 79
  - 用法 79
  - 语法 79
- forxmlschemaj
  - 基于 Java 的函数 65
  - 说明, 语法, 示例 65
- forxmlschemaj 安装 155
- forxmlschemaj 函数 99
- forxmlscriptj
  - 示例 72
  - 说明 71
  - 选项 71
  - 异常 72
  - 语法 70
- ForXmlTree XML 函数 74
- fprsqlinsert 安装 155
- fprsqlschemaj
  - 示例 72
  - 说明 71
  - 选项 71
  - 异常 72
  - 语法 70
- header
  - SQLX 选项 82, 85
- header 选项 37, 109
- help 命令 167
- hex, SQLX 选项 83
- HTML
  - DTD 元素 7
  - Order 代码示例 5
  - Order 数据的显示 5
  - 不一致的元素标记 6
  - 括号不一致的元素 6
  - 限制 6
- I18N
  - 非 ASCII 数据 105
  - 示例 109
- image 列 2
- image 数据类型 83, 98, 161
- image, 数据类型 116, 162
- image\_doc, sample\_docs 表的列 135
- in/not in 定量判定子查询, 不能将 for xml 子查询用作 60
- incremental 选项 37
- infile 命令 167
- info, XML 代码示例 4
- InputStream xmlj\_document
  - 示例 172
  - 用法 172
- installjava 实用程序 160
- installjava, 实用程序 160
- INTERFACES 环境变量 156
- 环境变量 \SISERVER 156
- ISQL, 用于传送数据 106
- isql, 与 for xml 子句搭配使用 81
- Java
  - 客户端字符集的示例目录 107
- Java 处理器 1
- Java 方法, com.sybase.xml.xql.Xql 特有的 171
- Java 服务
  - 表, 内存参数 161
  - 内存要求表 161
- Java 命令。请参见命令
- Java 映射函数, 安装 156
- java.lang.String 数据类型 2
- java.lang.String, 数据类型 116
- JDBC 代码
  - 使用属性 disable\_unichar\_sending< 发送 Unicode 数据 107
- make install-sqlx 命令 156
- make install-xerces 命令 156
- make sqlx-aliases 命令 157

- multiplentities 选项 86
- NCR 选项
  - xmlvalidate 支持的 116
  - 缺省数据类型 116
- ncr 选项 37, 82, 86
- ncr 选项, 缺省 108
- ncr, 支持的选项字符串 12
- NCR (数值字符表示), Unicode 106
- nonamespaceschemalocation 选项 37
- normalize-space 字符串函数 47, 50
- normalize-space, XPath 字符串函数 47
- normaliz-space, 函数 50
- nullclause 选项 38
- nullstyle
  - SQLX 选项 82, 86
- nullstyle 选项 37, 86
- nullstyle=attribute, 选项, 示例 104
- nullstyle=omit, 选项, 示例 103
- OpenXML 74
- OpenXML, XML 函数 77
- option xsidecl={yes | no} 91
- option\_string 值, 表 37
- option\_strings
  - 参数 9
  - 说明 37
  - 说明和示例 36
  - 讨论 36
  - 一般格式 36
  - 语法 37
- Order DTD, 示例代码 7
- Order 示例
  - HTML 5
  - XML 代码 3
- outfile 命令 167
- parse()
  - Java 方法, 命令 161
  - 返回 sybase.aseutils.SybXmlStream 162
  - 命令 162
- parse(), Java 方法, 命令 161
- parse(InputStream xmlll\_document), XML 方法 171
- parse(String xmllDoc), XQL 方法 171
- prefix 选项 38
- publishers, pubs2 数据库表 136
- qstring 命令 167
- root 选项 38
- rowname 选项 38
- rowname, SQLX 选项 82, 88, 97
- sample\_doc 表
  - 行 136
- sample\_docs 表
  - publishers 表, XML 表示形式 139
  - publishers 和 titles 表 138
  - 表脚本 (publishers) 138
  - 表脚本, Java 138
  - 结构 135
  - 列 135
- sample\_docs 表的列
  - image\_doc 135
  - name\_doc sample\_docs 表的列 135
  - text\_doc 135
- saxparser, com.sybase.xml.xql.XqlDriver 选项 167
- schemalocation 选项 38
- schemavalidate 选项 38
- select into 命令, 不能使用 for xml 子查询 60
- select 命令 86
- select 语句
  - ncr 选项 108
- setParser, XQL 方法 174
- SGML, 标准化通用标记语言 2
- sp\_configure
  - 启用 XML 服务和外部文件系统访问 146
- sp\_configure 选项 default xml sort order 115
- sp\_configure, 启用 XML 服务和外部文件系统访问 145
- SQL 扩展, 查询函数 9
- SQL 名称, 示例 97
- SQL 语句中的 BNF 表示法 xv, xvi
- SQL 语句中的大括号 ({} ) xvi
- sql\_name, SQLX 选项 88
- SQLX
  - 模式映射 99
  - 模式映射示例 99
  - 数据 92
  - 数据映射 92
  - 选项, 表 82
  - 选项, 定义 83
  - 映射函数, 安装过程 155

- SQLX 选项 82
  - base64 83
  - binary 82, 83
  - columnstyle 82, 83, 92
  - columnstyle=attribute, 示例 102
  - columnstyle=element, 示例 101
  - entitize 84
  - format 82
  - format={yes | no} 85
  - header 82, 85
  - hex 83
  - incremental 82, 86
  - nullstyle 82, 86
  - nullstyle=omit 103
  - prefix 82, 97
  - root 82, 87
  - rowname 82, 88, 97
  - schemaloc 82, 89
  - sql\_name 88
  - statement 82, 90
  - tablename 82, 90, 97
  - tablename=sqlname 90
  - targetns 82, 90
  - targetns=url 90
- SQLX 选项 binary 82
- SQLX 选项 columnstyle 82
- SQLX 选项 format 82
- SQLX 选项 incremental 82, 86
- SQLX 选项 prefix 82, 97
- SQLX 选项 root 82, 87
- SQLX 选项 schemaloc 82, 89
- SQLX 选项 statement 82, 90
- SQLX 选项 targetns 82, 90
- SQLX-XML
  - format 81
  - 模式, columnstyle=attribute, 示例 102
  - 模式, columnstyle=element 选项 101
  - 模式, nullstyle=attribute, 示例 104
  - 模式, nullstyle=omit, 示例 103
  - 模式, 示例 99
- SQLX-XML 格式 92
- statement 选项 38
- 参数
  - infile 166
  - outfile 166
- string, 数据类型 116
- sybase.asciutils 173
- sybase.aseutils.SybXmlStream, 由 parse() 命令返回 162
- SybFileXmlStream, XQL 方法 174
- SybXmlStream 171
  - 变量 171
- tablename 选项 38
- tablename, SQLX 选项 82, 90, 97
- tablename=sqlname, SQLX 选项 90
- targetns 选项 38
- targetns=url, SQLX 选项 90
- tempdb, 增加以安装 xerces.jar 160
- text doc sample\_docs 表的列 135
- text 数据类型 2, 98, 161
- text() XPath 基本运算符 45
- text, 数据类型 162
- titles, pubs2 数据库表 136
- tolower 字符串函数 47, 50
- tolower, toupper, 函数 50
- tolower, XPath 字符串函数 47
- toupper 字符串函数 47
- toupper** 字符串函数 50
- toupper, XPath 字符串函数 47
- unichar 数据类型 2
- unichar, 数据类型 116
- Unicode 114
  - for xml 107
  - select 语句中的 ncr 选项 108
  - xmlextract 114
  - xmlparse 113
  - xmlvalidate 116
  - 代理对 106
  - 列 106
  - 数据类型 106
  - 数据类型的类别 105
  - 数值字符表示 (NCR) 106
  - 选项字符串 108
- Unicode 列
  - java.lang.String 146
  - unichar 146
  - unitext 146
  - univarchar 146
- Unicode 示例, 示例表 109
- Unicode 数据类型 unichar, univarchar, unitext, java.lang.String 116
- Unicode, XML 形式 105

- Unicode, 非 ASCII 数据 105
- Unicode, 非 ASCII 数据 I18N 扩展 105
- univarchar 数据类型 2
- univarchar, 数据类型 116
- univarchar 数据类型 2
- univarchar, 数据类型 116
- URI, 不支持的 39
- URI (通用资源标识符) 39
- URI (通用资源标识符), 支持的 39
- URL, 命令 169
- UTF-16 (Unicode 转换格式, 两个字节) 数据类型 106
- UTF8, 缺省字符集 4
- UTF-8 (Unicode 转换格式, 最多四个字节) 数据类型 106
- validate 命令 167
- validate, 命令 168
- varbinary 数据类型 98
- varchar univarchar 列 110
- varchar 数据类型 2, 98
- Web, XML 文档的存储 169
- where 子句
  - 命令 165
  - 影响处理 165
- where 子句, 示例 165
- where 子句示例 165
- xerces.jar
  - 目录 160
- xerces.jar, 增加大小以便安装 160
- XFS
  - 指定 xmlerror=message 151
- XML
  - DTD 示例代码, 从外部引用 7
  - DTD 示例代码, 嵌入 7
  - DTD 元素, 限制 7
  - DTD, 指令 7
  - publishers 表的表示形式 139
  - SGML 的子集 2
  - titles 表的表示形式 140
  - 不是在所有文档中都要求 DTD 8
  - 查询函数 9
  - 查询语言 39
  - 名称, 映射到 95
  - 模式, 示例 101
  - 模式声明 40
  - 声明, 用于指定字符集 4
  - 示例, bookstore.xml 167
  - 示例文档 3
  - 适用于数据交换 2
  - 特殊应用的文档类型 2
  - 严格的短语结构 2
  - 已分析的 2
  - 映射 81
  - 映射函数 55
  - 由 HTML 浏览器和处理器来读取 2
  - 与 HTML 比较 2
  - 与 SGML 和 HTML 比较 2
  - 值, 映射到 97
  - 自定义标记 2
- xml
  - 示例文档 136
- xml all 选项 86
- XML EFS 访问 145
- XML 查询语言 (XQL) 162
- XML 查询语言, XPath 的子集 43
- XML 方法
  - parse(InputStream xml\_document) 171
- XML 方法, 查询
  - (String query, String xmlDoc) 172
  - (String query, String xmlDoc), XML 方法 172
- XML 服务
  - XPath 字符串函数 47
  - 带括号的表达式 52
  - 使用 sp\_configure 145
  - 通过外部文件系统访问 145
  - 通过外部文件系统访问, 使用 sp\_configure 146
- XML 服务中的排序顺序 115
- XML 函数 50, 74
- XML 示例模式 101
- XML 示例模式的组成部分 101
- XML 文档
  - DTD 示例代码 7
  - 标记 2
  - 部分 4
  - 查询 166
  - 创建代理表, 示例代码 146

- 从 Adaptive Server 中生成 2
- 从子目录中查询, 示例代码 149
- 存储为 OS 文件 169
- 存储在 Adaptive Server 中 2
- 存储在 Web 上 169
- 导入 ASE 表, 示例代码 148
- 对 DTD 有效的 8
- 没有格式指令 4
- 命名空间支持, XML 文档 40
- 嵌套标记 4
- 删除, 使用 XQL 处理器 162
- 使用 XQL 处理器更新 162
- 使用基于 Java 的 XQL 处理器插入 162
- 示例, info 4
- 示例代码, Order 3
- 提取书名, 示例代码 148
- 严格的短语结构 2
- 在 Web 上搜索 2
- 组织良好 4
- 作为字符数据 4
- XML 文档, 示例 13
- XML 文档类型定义中的加号 (+) 7
- XML 文档示例 13
- xml.zip, 目录 160
- XMLBean 169
- xmlcol, 数据库对象 161, 163
- xmlerror 选项 38
- xmlerror, 支持的选项字符串 12
- xmlparse
  - 选项 23
- xmllexrepresentation
  - 说明 25
- xmltest
  - 说明 19
  - 选项 19
- xmlextract 105, 114, 115
  - ncr 选项 114
  - 对 XQuery 语言子集的支持 40
  - 函数说明 10
  - 命名空间前缀 40
  - 内置查询函数 9
  - 排序顺序 114
  - 示例 13
  - 说明 11
  - 提取 XML 1
  - 异常 12
  - 语法 10
- xmlextract 命令 1
- xmlextract, 查询函数 9
- xmlimage, 数据库对象 163
- xmlparse 113, 115
  - Unicode 113
  - 存储非 ASCII 数据 105
  - 函数说明 22
  - 内置函数 9
  - 排序顺序 113
  - 示例 23
  - 说明 22
  - 异常 23
  - 用户对选项的修改 115
  - 语法 22
- xmlparse 命令 1
- xmlparse 中的非 ASCII 数据 113
- xmlparse, 编码选项 113
- xmlrepresentation
  - 函数说明 25
  - 示例 25
  - 语法 25
- xmlrepresentation, 查询函数, 确定经过分析的图像列 9
- xmltable
  - ordinality 列示例 124
  - 关于处理文档表的示例 129
  - 函数 117, 118
  - 另请参见 134
  - 示例 118
  - 说明, 语法 118
  - 用法 132
- xmltable, 派生表语法 117
- xmltest 105
  - 查询 XML 1
  - 函数 17
  - 函数谓词 17
  - 示例 19
  - 说明 18
  - 语法和说明 17

- xmltest 命令 1
  - xmltest, 查询函数 9
  - xmltext
    - SQL 谓词, 返回布尔结果 9
    - 命名空间前缀 40
  - XMLTEXT, 数据库对象 161
  - xmlvalid 选项 38
  - xmlvalidate
    - 查询函数 9
    - 命令 27
    - 示例 31
    - 说明 27
    - 选项 28
    - 选项说明 28
    - 异常 30
    - 语法 27
  - xmlvalidate 命令 1
  - xmlvalidate, 命令 27
  - xmlvalidate, 支持 NCR 选项 116
  - xmlvalidate, 支持 Unicode 116
  - XPath
    - 比较运算符 46
    - 比较运算符, 表 46
    - 带括号的表达式 52
    - 基本运算符, 表 45
    - 集合运算符, 表 46
    - 示例 47
    - 一般准则 47
    - 语法和标识 43
    - 语言 1
    - 语言子集 43
    - 运算符和函数 45
    - 支持的标识 44
  - XPath 1.0 39
  - XPath 中的比较运算符 46
  - XPath 字符串函数 47
    - 示例 47
  - XQL
    - EJB 169
    - JavaBean 169
    - JDBC 客户端 169
    - 编号系统 167
    - 查询结构 165
    - 导航 162
    - 基于路径的查询语言 162
    - 接口,
      - com.sybase.xml.xql.store.SybMemXmlStream 174
    - 开发独立应用程序 169
    - 使用 162
    - 显示为 XML 159
    - 以零为基 167
    - 运算符 162
    - 在 Adaptive Server 中安装 160
  - XQL 处理器
    - 删除 XML 文档 162
  - XQL 处理器, 基于 Java, 插入 162
  - XQL 处理器, 迁移文档 178
  - XQL 方法
    - parse(String xmlDoc) 171
    - query(String query, InputStream xmlDoc) 173
    - query(String query, SybXmlStream xmlDoc) 173
    - setParser 174
    - SybFileXmlStream 174
    - SybXmlStream 173
  - XQuery 语言, 由 xmlextract 和 xmltest 支持 40
  - xscdecl 选项 38
  - xsidecl 选项 82
  - xsidecl={yes|no} 选项 91
  - XSL, 可扩展样式语言 2
  - €, 欧元 106
- ## A
- 安装
    - Adaptive Server 中的 XQL 160
    - SQLX 映射函数, 基于 Java 155
    - SQLX 映射类 156
    - 分析程序 156
    - 基于 Java 的 XQL 处理器 160
    - 安装基于 Java 的分析程序 160

**B**

- 保留空白 42
- 本地文件, com.sybase.xml.xql.XqlDriver 166
- 本机 XML 处理器 1
  - 从基于 Java 的处理器迁移 177, 181
  - 针对基于 Java 的处理器进行迁移 178
  - 重新生成文本文档 180
- 编码选项, xmlparse 113
- 变量, HTML
  - CustomerID 5
  - ItemID 5
  - ItemName 5
  - order 5
  - Quantity 5
  - units 5
- 变量, SybXmlStream 171
- 变量, XML 标记 4
- 标记
  - HTML, 段落 6
  - HTML, 括号不一致的 6
  - XML 中的自定义 2
  - 严格嵌套的 XML 4
  - 用户创建的 4
- 标识
  - XPath, 支持的 44
- 标准化通用标记语言 (SGML) 2
- 表
  - option\_string 值 37
  - publishers, XML 表示形式 139
  - SQLX 选项 82
  - titles, XML 表示形式 140
  - XPath 比较运算符 46
  - XPath 基本运算符 45
  - XPath 集合运算符 46
  - 环境变量 156
- 表达式, XPath 中带括号的 52
- 别名名称, 创建 157
- 不一致的元素标记, HTML 6
- 布尔表达式, 在过滤运算符中 164

**C**

- 参数
  - 166
  - help 166
  - qstring 166
  - validate 166
- 查询
  - (String query, InputStream xmlDoc), XQL 方法 173
  - (String query, SybXmlStream xmlDoc), XQL 方法 173
  - 迁移 178
  - 在基于 Java 的处理器和本机处理器之间迁移 180
- 查询 XML
  - 使用 xmltest, 和 xmlextract 1
- 查询方法, com.sybase.xml.xql.Xql 172
  - 方法, 特有的 173
- 查询函数
  - XML 9
  - xmlextract 9
  - xmlrepresentation 9
  - xmltest 9
  - xmlvalidate 9
  - 表 9
  - 语法和示例 10
- 查询函数, SQL 扩展 9
- 查询函数, 选项值 37, 42
- 查询结构 165
- 查询结构, XQL 中 165
- 查询命令 169
- 查询引擎
  - 内存要求 161
- 重复列名, 映射 92
- 重新生成
  - 文本文档, 从本机 XML 处理器 180
  - 文本文档, 从基于 Java 的处理器 179
- 处理
  - where 子句的影响 165

- 处理非 ASCII 数据 114
- 处理器
  - XML, 本机 1
  - XML, 基于 Java 1
- 传送客户端 - 服务器数据 106
- 传送客户端 - 服务器数据, 使用 CTLIB、ISQL、BCP 106
- 创建别名名称 157
- 纯字符或预定义实体 42
- 存储 XML
  - 在 image 列中分析 2
  - 作为数据类型 2

## D

- 带括号的表达式
  - 和联合 53
  - 和下标 52
- 代理表, 由文件系统访问功能创建 145
- 代理对, 16 位值 106
- 代理对, Unicode 106
- 代码示例
  - HTML, Order 示例 5
  - XML, Info 示例 4
  - XML, Item 示例 5
- 导航 XQL 162
- 等号, 运算符 163
- 逗号 (, )
  - SQL 语句中的 xvi
- 独立程序, com.sybase.xml.xql.XqlDriver 166
- 独立应用程序
  - 使用 XQL 169
  - 示例 169
- 多种语言, 基于 XML 的应用程序 105

## E

- 二进制
  - 数据类型 98
  - 值 98

## F

- 方法, com.sybase.xml.xql.XQL 中 171
- 方法。请参见 Java 方法
- 非 ASCII 数据
  - XML 文档和查询中的处理 105
  - 处理 114
  - 存储在 xmlparse 中 105
  - 在 for xml 子句中生成 105
- 非 ASCII 数据, Unicode, I18N 105
- 非 ASCII 数据, 支持 105
- 分层 XML, 映射到 76, 77
- 分析
  - 方法 171
  - 命令 169
- 符号
  - SQL 语句中的 xv
- 服务器字符集, 不同于客户端 107

## G

- 格式设置
  - SQLX-XML 81
  - 指令, 在 XSL 中提供 2
- 更新, 命令 160
- 过滤运算符 164
- 过滤运算符, 使用布尔表达式 164

## H

- 函数
  - concat 47, 51
  - forsqlcreatej, 说明, 语法, 示例 70
  - forsqlinsertj 70
  - forsqlinsertj, 基于 Java 的映射函数 70
  - forsqlscriptj 70
  - forxml 86
  - forxmlallj 65, 99
  - forxmldtdj 65
  - forxmlj 81
  - forxmlj, 说明、语法和示例 65
  - forxmlschemaj 65, 99
  - ForXmlTree 74



- normalize-space 50
- OpenXML 74, 77
- tolower, toupper 50
- xmlextract 10
- xmlparse 22
- xmlrepresentation 25
- xmltable 117, 118
- XPath 45
- 基于 Java 65
- 基于 Java 的 SQLX 映射 155
- 映射 55
- 函数 concat, XPath 字符串函数 47
- 函数 normalize-space, XPath 字符串函数 47
- 函数 tolower, XPath 字符串函数 47
- 函数 toupper, XPath 字符串函数 47
- 函数, XML, 说明 50
- 函数谓词 xmltest 17
- 环境变量
  - \\$DB 156
  - DB 156
  - \\$INTERFACES 156
  - 表 156
- 环境变量 \$ISERVER 156

## J

- 基本运算符, XPath, 支持 45
- 基于 Java
  - SQLX 映射函数, 安装过程 156
  - SQLX 映射函数, 列表 155
  - 分析程序, 安装 156
  - 映射函数 65, 70
- 基于 Java 的 SQLX 映射部分, 安装 155
- 基于 Java 的 XQL 处理器
  - 插入 XML 文档 162
  - 将 XML 转换为已分析的 161
- 基于 Java 的 XQL 处理器, 安装 160
- 基于 Java 的 XQL 处理器, 内存要求 161
- 基于 Java 的 XQL 处理器, 设置 159

- 基于 Java 的处理器
  - 安装 160
  - 重新生成文本文档 179
  - 迁移到本机 XML 处理器 177, 181
  - 针对本机 XML 处理器进行迁移 178
- 基于 Java 的函数
  - 示例 67
  - 说明 66
  - 选项 66
  - 异常 66
  - 语法 66
- 基于 Java 的映射函数 65, 70
- 结构, 查询 165

## K

- 可扩展样式语言。请参见 XSL
- 客户端, 字符集 107
- 客户端 - 服务器数据
  - 传送 106
  - 使用 Java 106
- 客户端和服务器的字符集, 差异 107
- 空, 白, 保留 42
- 空白, 保留 42
- 空元素 43
- 括号。请参见中括号 []
- 扩展
  - for xml 子句 61
  - for xml 子句, 示例 61
  - for xml 子句, 说明 61
  - for xml 子句, 异常 61

## L

- 联合和括号 53
- 列
  - image, 已分析的 XML 2
- 列字符串 /unicode 106

## M

## 命令

- alter table 161
  - at 子句 147
  - create proxy table 147
  - debug 167
  - FileInputStream() 169
  - help 167
  - infile 167
  - make install-sqlx 156
  - make install-xerces 156
  - make sqlx-aliases 157
  - new 160
  - outfile 167
  - parse() 161, 162
  - qstring 167
  - select 86
  - update 160
  - URL 169
  - validate 167, 168
  - where 子句 165
  - xmlparse xmltest 1
  - xmltest 1
  - xmlextract 1
  - xmlvalidate 1, 27
  - 查询 169
  - 分析 169
  - 有效的 168
- 命名空间
- 声明和引用 40
  - 支持, XML 文档 40
- 模式支持 40

## N

- 内存参数, Java 服务, 表 161
- 内存要求
  - Java 服务 161
  - 对于查询引擎 161
  - 配置 161
- 内存要求, 基于 Java 的 XQL 处理器 161
- 内容列, Unicode, 分配规则 146

## O

- 欧元符号, € 106

## P

## 排序顺序

- xmlextract 114
- xmlparse 113
- 派生表语法 118
- 派生表语法, xmltable 中 117
- 配置内存要求 161

## Q

## 迁移

- 查询 180
- 从重新生成的副本迁移文档 178
- 文档和查询 178
- 限制 177
- 在 XQL 处理器和本机 XML 处理器之间迁移文档 178
- 在基于 Java 的处理器和本机 XML 处理器之间迁移文本文档 178
- 在基于 Java 的处理器和本机 XML 处理器之间迁移文档 178
- 在基于 Java 的处理器与本机处理器之间 177, 181
- 嵌入的 DTD 8
- 嵌套的标量子查询, 不能使用 for xml 子句 60
- 区分大小写
  - 在 SQL 中 xvi
- 全局变量
  - @@error 12
- 缺省 ncr 选项 108

## S

## 删除

- XML 文档, 使用 XQL 处理器 162
- 设置 classpath 环境变量 159
- 设置 XQL 处理器 159

## 实体

预定义, XML 查询语言中 42

预定义, XML 语言中 40

## 使用

com.sybase.xml.xql.XqlDriver 166

使用 for xml 生成 XML 1

使用 ForXmlTree 映射 SQL 数据 76

使用 OpenXml 映射 SQL 数据 77

使用 XQL 处理器更新 XML 文档 162

使用 XQL (XML 查询语言) 162

使用数据类型存储 XML, xmlparse, 和 xmlvalidate  
1

## 示例

for xml 子句 58, 109

xmlerror 选项, 使用 XFS 150

分析 XML 文档, 将 xmlerror=message 与 XFS 一起使用 151

将经过分析的 XML 文档存储在外部文件系统中  
149

使用 xmlerror=null 153

指定 xmlerror=message 选项, XFS 151

子查询, for xml 60

## 示例表

sample\_docs 135

Unicode 示例 109

示例表, sample\_docs 135

## 示例代码

DTD, Order 示例 7

HTML, Order 示例 5

XML, Info 示例 4

XML, Order 示例 DTD 7

示例数据 树形结构 74

示例数据中的树形结构 74

属性, 嵌入在元素标记中 4

数据, 用 XQL 选择 159

## 数据库对象

xmlcol 161, 163

xmlimage 163

XMLTEXT 161

XQL, XML 通用查询语言 162

## 数据类型

binary 83

char 2, 98

image 2, 83, 98, 161, 162

text 2, 98, 161, 162

Unicode 中的类别 105

varbinary 83, 98

varchar 98

二进制 98

数值 97

数据类型 Unicode, Unicode

数据类型 105

数据类型, Unicode

char 105

text 105

unitext 105

univarchar 105

varchar 105

## 数值

数据类型 97

值 97

数值字符表示 (NCR), Unicode 106

## 说明

子查询, for xml 59

搜索, 存储在 Web 上的 XML 文档 2

## T

提取 XML 数据, 使用 xmltest 和 xmlextract 1

通用资源标识符 (URI) 39

## W

外部目录递归访问, 映射代理表 145

外部文件系统, Unicode 列 146

外部文件系统, 字符集转换 146

外部文件系统访问, 在 XML 中 145

未命名列, 映射 92

文本数据, XML 4

## 文档

查询, 迁移 178

验证 168

在 Java 和本机处理器之间迁移 178

文档类型定义。请参见 DTD

文件名, bookstore.xml 167

文件系统访问, 功能 145

## X

- 下标,带括号的表达式中 52
- 下标运算符 164
- 限制,在迁移文档中 177
- 相对函数调用 48
- 相关子查询,不能是 for xml 子句 60
- 小括号 ()
  - SQL 语句中的 xv
- 新的,命令 160
- 选项
  - binary 37
  - columnstyle 37
  - dtdvalidate 37
  - entitize 37, 82
  - for xml 子句 58
  - format 37
  - header 37
  - incremental 37
  - multipleentitize 82
  - ncr 37, 82
  - nonamespaceschemalocation 37
  - nullclause 38
  - nullstyle 37
  - prefix 38
  - root 38
  - rowname 38
  - saxparser 167
  - schemalocation 38
  - schemavalidate 38
  - SQLX 82
  - SQLX, 定义 83
  - statement 38
  - tablename 38
  - targetns 38
  - xmlerror 38
  - xmlvalid 38
  - xscdecl 38
  - xsidecl 82
- 选项 multipleentitize 82
- 选项,验证 28
- 选项语法,xmlvalidate 28
- 选项值,查询函数 37, 42
- 选项字符串
  - for xml 108
  - ncr 中的规范 108

- Unicode 108
- 带引号的标识符 108
- 简单标识符 108
- 支持的 12
- 选项字符串值 37

## Y

- 验证文档 168
- 验证选项 28
  - dtdvalidate 28
- 已分析的 XML 2
- 异常
  - for xml 子句 58
  - 引发原因 12
  - 子查询, for xml 60
- 引用,XML DTD 内部 7
- 映射
  - SQL 名称,目标 95
  - SQL 值 97
  - SQL 值,示例 97
  - SQLX 模式,概述 99
  - SQLX 模式,示例 99
  - 到 XML 名称 95
  - 到 XML 值 97
  - 未命名列 92
  - 重复列名 92
  - 重复列名,示例 92
- 映射到分层 XML 76, 77
- 映射多个结果集 79
- 映射分层 XML 和 SQL 数据 74
- 映射函数
  - 基于 Java 65, 70
- 用 com.sybase.xml.xql.XqlDriver 查询 XML 166
- 用 for xml 子句处理非 ASCII 数据 107
- 用 XQL 选择数据 159
- 用户创建的元素标记 4
- 用户修改,xmlparse 中的选项 default xml sort order 115
- 用于处理包含非 ASCII 数据的 XML 文档 105
- 用于存储 XML 的数据类型 1
- 用于映射的 ForXmlTree 76
- 用于映射的 OpenXml 77

- 有关 79
- 有效的 XML 文档 8
- 有效的, 命令 168
- 语法
  - for xml 子句 55
  - for xml 子句, 说明 55
  - for sqlcreatej 70
  - for sqlinsertj 70
  - for sqlscriptj 70
  - for xmlallj 65
  - for xmltdtj 65
  - for xmlj 65
  - for xmlschemaj 65
  - option\_strings 36
  - xmlparse 22
  - xmltest 17
  - XPath 标识 43
  - 示例, xmlextract 10
  - 子查询, for xml 59
- 语法, xmlrepresentation 25
- 语法约定, Transact-SQL xv
- 语言
  - XML 和 XML 查询 39
  - XPath 1
  - XQL 1
- 预定义实体 40, 42
- 元素, 空 43
- 元素标记
  - HTML 6
  - 嵌入属性 4
  - 严格嵌套 4
  - 用户创建的 4
  - 自定义 4
- 约定
  - Transact-SQL 语法 xv
  - 另请参见 语法
  - 在参考手册中使用 xv
- 运算符
  - XPath 中的比较 46
  - 等号 163
  - 过滤 164
  - 下标 164
  - 子 163
  - 子代 163

## Z

- 在 XML 中嵌入 DTD 7
- 针对 Unicode 内容列的规则 146
- 值
  - 二进制 98
  - 数值 97
  - 字符 98
- 指定字符集 4
- 中括号 []
  - SQL 语句中的 xvi
- 中删除语言
  - 安装 XQL 160
- 准则, XPath 函数 47
- 子查询, for xml 59
- 子代运算符 163
- 子运算符 163
- 自定义元素 4
- 字符编码。请参见 字符集
- 字符串 /unicode 列 106
- 字符集
  - XML 4
  - 和 XML 数据 107
  - 客户端与服务器的差异 107
  - 缺省 UTF8 4
  - 声明的匹配实际的 4
  - 指定 4
  - 转换, 绕过 4
- 字符集传送, Java 107
- 字符集传送的示例目录, Java 107
- 字符集支持 39
- 字符集转换 146
- 字符文字 42
- 字符值, 示例 98
- 组成部分, XML 示例模式中的 101
- 组件集成服务 (CIS) 145
- 组织良好的 XML 文档 4

