SQL Server XML

基础指南

SQL Server XML 基础指南

自 SQL Server 2000 发布以来，处理 XML 数据就成为了数据库管理员们经常讨论的一个话题。而且随着 Web 应用的进一步加深，XML 在传统数据库管理系统中的应用也越来越广泛。在本次技术手册中，我们将对 SQL Server 中的 XML 应用进行一定的介绍，读者可以同之前的《DB2 XML 文档拆分指南》进行一下对比，相信将有助您对 XML 技术的进一步理解。

SQL Server XML 基础知识

XML 相比 HTML 给予了 Web 开发人员更大的编程灵活性。这种技术驱动开发机构尝试把 XML 同自己的产品集成起来。微软就是采取如此举措的先驱者。微软的几乎所有产品中都能看到 XML 的身影。

- SQL Server 2005 中 XML 数据建模简介
- SQL Server 和 XML 的集成
- SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一
- SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之二
- 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（一）
- 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（二）
- 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（三）

SQL Server XML 文档处理

自 SQL Server 2000 发布以来，处理 XML 数据就成为了数据库管理员们经常讨论的一个话题。常见的用法是前端应用程序以入参方式传给存储过程 XML 文档。你可以在几种技术中做选择，但这些技术大部分都十分复杂。

- 用 SQL Server 提供的函数处理 XML 文档
- SQL Server XML：释放的利与弊
- SQL Server XML 拆分选项对比
- SQL Server XML 拆分示例

SQL Server XML 进阶技巧

在同 SQL Server XML 打交道时，DBA 需要记住一些比较高级的技巧，才能到达事半功倍的效果，在本部分里，我们就介绍几个较为深入的 XML 技巧。

- 对比 XML AUTO 与 T-SQL 命令（上）
- 对比 XML AUTO 与 T-SQL 命令（下）
- 使用 XML 在 SQL Server 上创建计算列（上）
- 使用 XML 在 SQL Server 上创建计算列（下）
- 在 SSAS 中使用 XMLA 命令进行跟踪管理（上）
- 在 SSAS 中使用 XMLA 命令进行跟踪管理（下）

SQL Server 2005 中 XML 数据建模简介

如果您的数据是高度结构化的，具有已知的架构，则关系模型可能对于数据存储最为有效。

Microsoft SQL Server 提供了您可能需要的必要功能和工具。另一方面，如果结构是灵活的（半结构化和非结构化）或未知的，则必须适当地考虑如何对此类数据进行建模。

如果您需要独立于平台的模型，以便确保使用结构化和语义标记的数据的可移植性，则 XML 是一种不错的选择。而且，如果满足下列某些属性，则它还是一种适当的选择：

- 您的数据比较稀疏，或者您不了解数据的结构，或者数据的结构将来可能发生重大更改。
- 您的数据表示容器层次结构（与实体中的引用相对），并且可能是递归的。
- 您的数据具有内在的顺序。
- 您希望对数据进行查询，或者基于其结构更新部分数据。

如果上述任一条件都不满足，则您应该使用关系数据模型。例如，如果您的数据是 XML 格式，但您的应用程序很少使用数据库来存储和检索数据，则[n]varchar(max)列就能满足您的全部需要。在 XML 列中存储数据可以带来其他好处 - 引擎将检查数据格式规范或者有效，并且支持对 XML 数据进行细粒度的查询和更新。

在 SQL Server 2005 中存储 XML 数据的理由

以下为一些使用 SQL Server 2005 中的原生 XML 功能而不是在文件系统中管理 XML 数据的理由：

- 您希望使用数据库服务器的管理功能来管理 XML 数据（例如，备份、恢复和复制）。
- 您希望以高效的方式和事务处理方式来共享、查询和修改 XML 数据。细粒度的数据访问对于您的应用程序而言很重要。例如，您可能需要提取 XML 文档内部的某些节，或者您可能需要插入一个新节而不是替换整个文档。

- 您具有关系数据和 SQL 应用程序，您希望在应用程序内部的关系数据和 XML 数据之间进行互操作。对于跨域应用程序，您需要有关查询和数据修改的语言支持。
- 您希望服务器能够保证数据格式规范，并能够视情况根据 XML 架构来验证数据。
- 您需要将 XML 数据编入索引以便实现高效的查询处理和良好的可伸缩性，并且使用一流的查询优化器。
- 您希望对 XML 数据进行 SOAP、ADO.NET 和 OLE DB 访问。

如果不满足上述任一条件，您最好将数据存储为非 XML 的大型数据类型，如 [n]varchar(max) 或 varbinary(max)。

XML 存储选项

SQL Server 2005 中的 XML 的存储选项如下所示：

- 本机存储采用 XML 数据类型：

用能够保留数据的 XML 内容（如容器层次结构、文档顺序、元素和属性值等等）的内部表示形式存储数据。具体说来，就是保留 XML 数据的信息集内容（有关信息集的详细信息，请参阅 http://www.w3.org/TR/xml-infoset）。它可能不是文本 XML 的精确副本，因为未保留以下信息：无关紧要的空格、属性顺序、命名空间前缀和 XML 声明。

对于类型化的 XML 数据类型（即绑定到 XML 架构的 XML 数据类型）而言，负责向信息集添加类型信息的后架构验证信息集 (Post Schema Validation Infoset, PSVI) 以内部表示形式编码。这会显著提高分析速度。

- XML 和关系存储之间的映射：

使用带有批注的架构 (AXSD)，XML 将被分解到一个或多个表中的列，并且在关系级别保留数据的保真度 - 保留层次结构，但忽略元素顺序。架构不能是递归的。

- 大型对象存储（[n]varchar(max)和 varbinary(max)）：

存储了数据的精确副本。这对于特殊用途的应用（如法律文档）很有用。大多数应用不要求精确副本，XML 内容（信息集保真度）即可满足需要。

通常情况下，可能需要组合使用这些方法。例如，您可能需要用 XML 数据类型列存储 XML 数据，并将其中的属性提升到关系列中。相反，您可能希望使用映射技术，将非递归部分存储到非 XML 列中，而仅将递归部分存储到 XML 数据类型列中。

XML 技术的选择

XML 技术（原生 XML 与 XML 视图）的选择通常取决于下列因素：

存储选项：

您的 XML 数据可能更适合于大型对象存储（例如，产品手册），或者更适合于存储在关系列中（例如，转换到 XML 的行项目）。每个存储选项都在不同程度上保留了文档保真度。

- 查询功能：

基于查询的性质以及对 XML 数据进行查询的程度，您可能发现一个存储选项比其他存储选项更为适合。细粒度的 XML 数据查询（例如，XML 节点上的谓词计算）在这两个存储选项中受到不同程度的支持。

- 将 XML 数据编入索引：

您可能希望将 XML 数据编入索引，以便提高 XML 查询性能。索引选项随存储选项的不同而不同；您需要进行适当的选择以优化工作量。

- 数据修改功能：

某些工作量涉及到对 XML 数据进行细粒度的修改（例如，在文档内添加新节），而其他工作量则不涉及（例如，Web 内容）。对于您的应用程序而言，数据修改语言支持可能很重要。

- 架构支持：

您的 XML 数据可能通过架构进行描述，这可能是也可能不是 XML 架构文档。对架构绑定 XML 的支持取决于 XML 技术。

不用说，不同的选择具有不同的性能特性。

原生 XML 存储

可以将您的 XML 数据存储在服务器的 XML 数据类型列中。在下列情况下，这将是一个适当的选择：

- 您需要一种在服务器上存储 XML 数据的简单方法，同时需要保留文档顺序和文档结构。
- 您的 XML 数据可能有也可能没有架构。
- 您需要查询和修改您的 XML 数据。
- 您需要将 XML 数据编入索引以便实现更为快速的查询处理。
- 您的应用程序需要系统目录视图以管理您的 XML 数据和 XML 架构。

当您的 XML 文档具有多种结构时，或者当您的 XML 文档符合不同的或复杂的架构，而这些架构很难映射到关系结构时，原生 XML 存储将很有用。

示例：使用 XML 数据类型对 XML 数据进行建模

考虑一个 XML 格式的产品手册，其中每个主题对应单独的一章，而每章内又有多节。一节可以包含多个子节，因此 是一个递归元素。产品手册包含大量混合内容、图表和技术资料；数据是半结构化的。用户可能希望对感兴趣的主题执行与上下文有关的搜索（例如，在有关"索引"的章内部搜索有关"聚集索引"的节），并且查询技术数量。

XML 文档的合适存储模型是 XML 数据类型列。这可以保留 XML 数据的信息集内容。将 XML 列编入索引可以提高查询性能。

示例：保留 XML 数据的精确副本

假设政府法令要求您保留 XML 文档（例如，已签署的文档、法律文档或股票交易订单）的精确文本副本。您可能需要将您的文档存储在[n]varchar(max) 列中。

对于查询，可在运行时将数据转换为 XML 数据类型，然后对其执行 Xquery。运行时转换可能代价高昂，尤其是在文档很大时。如果您经常进行查询，可以采用冗余方式将文档

存储在 XML 数据类型列中并将其编入索引，同时从 [n]varchar(max) 列返回精确的文档副本。

XML 列可能是基于 [n]varchar(max) 列的计算列。您不能在 XML 计算列上创建 XML 索引，也不能在[n]varchar(max)或 varbinary(max) 列上生成 XML 索引。

XML 视图技术

通过在 XML 架构和数据库的表之间定义映射，可以创建持久性数据的"XML 视图"。可以使用 XML 批量负载来填充使用 XML 视图的基础表。您可以查询使用 XPath 1.0 的 XML 视图；该查询将被转换为针对表的 SQL 查询。与此类似，更新也会被传递到这些表。

在以下情况下，此技术很有用：

- 您希望拥有以 XML 为中心的编程模型，该模型使用现有关系数据上的 XML 视图。
- 您的 XML 数据具有架构 (XSD, XDR)，它可能由外部合作伙伴提供。
- 数据的顺序不重要，或者您的可查询数据不是递归的，或者预先已经知道最大递归深度。
- 您希望通过使用 XPath 1.0 的 XML 视图来查询和修改数据。
- 您希望批量加载 XML 数据，并将其分解到使用 XML 视图的基础表中。

这方面的例子包括以 XML 形式公开以便用于数据交换和 Web 服务的关系数据，以及具有固定架构的 XML 数据。有关详细信息，请参阅 SQLXML 开发人员中心。

示例：使用带有批注的 XML 架构 (AXSD) 对数据进行建模

假设您现有一些希望以 XML 形式进行操作的关系数据（例如，客户、订单和行项目）。请使用 AXSD 在关系数据上定义 XML 视图。通过 XML 视图，可以将 XML 数据批量加载到表中，以及使用 XML 视图查询和更新关系数据。如果您需要在自己的 SQL 应用程序持续工作时与其他应用程序中的 XML 标记交换数据，则该模式很有用。

混合模型

很多时候，适合将关系数据和 XML 数据类型列结合起来进行数据建模。可以将 XML 数据中的某些值存储在关系列中，而将其余或全部 XML 值存储在 XML 列中。这可能会产生更

好的性能（例如，可以完全控制在关系列上创建的索引）和锁定特性。然而，这需要您承担更多的责任来管理数据存储。

要存储在关系列中的值取决于您的工作负荷。例如，如果您基于路径表达式 /Customer/@CustId 检索全部 XML 值，则通过将 CustId 属性的值提升到关系列中以及将其编入索引，可能产生更高的查询性能。另一方面，如果您的 XML 数据被广泛且非冗余地分解到关系列中，则重新组合的成本可能很大。

对于高度结构化的 XML 数据（例如，表的内容已经转换到 XML），可以将所有值映射到关系列（可能使用 XML 视图技术）。

（作者：Shankar Pal 来源：微软）

原文标题：SQL Server 2005 中 XML 数据建模简介

链接：http://www.searchdatabase.com.cn/showcontent_8759.htm

SQL Server 和 XML 的集成

XML 相比 HTML 给予了 Web 开发人员更大的编程灵活性。这种技术驱动开发机构尝试把 XML 同自己的产品集成起来。微软就是采取如此举措的先驱者。微软的几乎所有产品中都能看到 XML 的身影。举个例子：微软是如何在其 SQL Server 产品线中集成 XML 的呢？下面咱们就来看看关键的 FOR XML 子句。

以 XML 的名义获取信息

SQL Server 和 XML 之间的集成首要一点就是根据 SQL 数据创建 XML 文件。XML 文件的构造并不复杂，用简单的脚本和 ADO 记录集就可以轻松产生。这个任务虽然不算麻烦，但开发人员却需要针对他们从服务器获取的结果集合产生不同的脚本，或者编写更为复杂的通用脚本。select 语句则由此而配备了新的 FOR XML 子句。

该子句的语法如下所示：

```
[ FOR { XML { RAW | AUTO | EXPLICIT }
[ , XMLDATA ]
[ , ELEMENTS ]
[ , BINARY BASE64 ] } ]
```

FOR XML 子句的 XML 模式由三种参数值表示：RAW、AUTO 或者 EXPLICIT。模式决定了结果 XML 的形式和组成。下面我们就更深入些地通过以下示例了解以上各个 XML 选项。

RAW 示例

我们执行以下的 SQL 语句：

```
SET ROWcount 3
```

```
select Orders.OrderID, Orders.OrderDate, ProductID
FROM Orders, [Order Details]
where Orders.OrderID = [Order Details].OrderID
ORDER BY Orders.OrderID
FOR XML RAW
```

执行后产生的结果如下：

```
<row OrderID="10248" OrderDate="1996-07-04T00:00:00" ProductID="11"/>
<row OrderID="10248" OrderDate="1996-07-04T00:00:00" ProductID="42"/>
<row OrderID="10248" OrderDate="1996-07-04T00:00:00" ProductID="72"/>
```

我们执行以下的 SQL 语句：

```
‘结果限制为 3 条记录。
SET ROWcount 3
select Orders.OrderID, Orders.OrderDate, ProductID
FROM Orders, [Order Details]
where Orders.OrderID = [Order Details].OrderID
ORDER BY Orders.OrderID
FOR XML AUTO
```

产生的结果如下所示：

```
<Orders OrderID="10248" OrderDate="1996-07-04T00:00:00">
<Order_x0020_Details ProductID="11"/>
```

```
<Order_x0020_Details ProductID="42"/>
<Order_x0020_Details ProductID="72"/>
</Orders>
```

EXPLICIT 示例

Explicit 模式给予查询编程人员对产生 XML 的完全控制能力。然而这种控制力度却要价不菲：你得编写每一查询以便 SQL 语句能包含 XML 信息。

有关的语法很复杂，而且超出了本文的讨论范围。[, XMLDATA] [, ELEMENTS] [, BINARY BASE64] 是相应的可选参数。

可选元素

示例可以让我们对各种设置的内部工作机理有更多的了解，下面我们就进一步研究下 FOR XML 语句的可选元素 XMLDATA。

如果你设置该选项，那么 XML-Data schema 就会包含在结果集合里。以下是 SQL 语句：

```
SET ROWcount 3
select Orders.OrderID, Orders.OrderDate, ProductID
FROM Orders, [Order Details]
where Orders.OrderID = [Order Details].OrderID
ORDER BY Orders.OrderID
FOR XML AUTO, XMLDATA
```

以上的 SQL 语句产生以下结果：

```
<Schema name="Schema2" xmlns=
```

```
"urn:schemas-microsoft-com:xml-data" xmlns:dt=

"urn:schemas-microsoft-com:datatypes">

<ElementType name="Orders" content="eltOnly" model="closed"

order="many"><element type="Order_x0020_Details" maxOccurs="*"/>…
```

ELEMENTS

ELEMENTS 选项指示各数据列作为子元素而非属性返回。假如你采用 AUTO 模式就可以只采用该选项。

BINARY BASE64

使用该选项表示你希望采用 base64 编码格式表示二进制数据。

注：本文只是对 FOR XML 子句的简单说明，这里要提醒你的是这一部分不过是 XML 同 SQL Server 集成需要注意的一点，其他方面的问题还包括 IIS 的 OPENXML 函数和模版文件等。

（作者：佚名 来源：网络）

原文标题：SQL Server 和 XML 的集成

链接：http://www.searchdatabase.com.cn/showcontent_8735.htm

SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一

作为一个数据库管理员，我试着仔细分析性能问题以及如何保证使用 XML 时不要影响 SQL Server 性能。在本文中，我将向你介绍一个例子，它使用了两个表，一个用于插入和查询 XML 数据，而另一个使用 VARCHAR(MAX)数据类型。然后我们观察存储、CPU 和 I/O 的测试结果，从而为你的 SQL Server 环境作出最佳选择。

注意：这里使用的测试仅仅针对基本的表，不使用索引。

如果你对于在 SQL Server 使用 T-SQL 命令和 XML AUTO 的应用性能比较有兴趣，请阅读我的上一篇技巧文章。

XML 数据类型

XML 数据类型会由 SQL Server 在进行检查时与 VARCHAR(MAX)比较，以保证它的内容是合法的 XML。

测试环境描述

对于我的测试，我将使用从一个*.rdl 文件（Reporting Services Report）拷贝的 XML，大小为 265KB。我将创建两个有相同结构的表，只是其中的 MyXML 域，一个表使用 XML 数据类型，而另一个表使用 VARCHAR(MAX)数据类型。

```
<create table TryVACRCHARDatatype(
    id int identity not null,
     MyXML VARCHAR(MAX) null
              ) Go
  create table TryXMLDatatype (
    id int identity not null,
         MyXML XML null
              )Go
```

我将用相同的方法向每一个表插入 XML 数据：

```
set statistics io on
Go
declare @XML XML
--
My big XML (for space reasons I am not including all of it here) SET @XML = '
<? xml version="1.0" encoding="utf-8"? >
.....
'

insert into TryXMLDatatype (MyXML) values (@XML)
go
declare @Varch VARCHAR(MAX)
-- My big XML (for space reasons I am not including all of it here)
SET @Varch = '<? xml version="1.0" encoding="utf-8"? >
.....
'

insert into TryXMLDatatype (MyXML) values (@Varch)
Go
```

插入 XML 数据到表中

我在 SQL Profiler 中监控上面的插入命令，并得到 I/O 统计，运行两次。

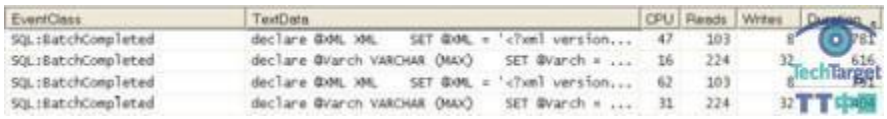
I/O 统计结果：

TryXMLDatatype 表。扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：18，慢速物理读取次数：0，慢速预读次数：0。

TryVACRCHARDatatype 表。扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：90，慢速物理读取次数：0，慢速预读次数：0。

执行计划显示了两个命令的精确计划。

Profiler 结果（每一个插入的两次执行）：



EventClass	TextData	CPU	Reads	Writes	Duration
SQL:BatchCompleted	declare @XML XML SET @XML = '<?xml version...	47	103	8	781
SQL:BatchCompleted	declare @Varch VARCHAR (MAX) SET @Varch = ...	16	224	32	616
SQL:BatchCompleted	declare @XML XML SET @XML = '<?xml version...	62	103	8	851
SQL:BatchCompleted	declare @Varch VARCHAR (MAX) SET @Varch = ...	31	224	32	

注意：XML 插入占用更多的 CPU 而读/写则比较少。同时插入 XML 数据需要时间更长。我将在下一部分分析这个行为。

查询表

I ran SELECT * FROM in each table and monitored with Profiler and Statistics I /O:

```
SELECT * FROM TryXMLDatatype
Go
SELECT * FROM TryVACRCHARDatatype
Go
```

结果：

注意：域中的值是不同的！

I/O 统计结果：

TryXMLDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：56，慢速物理读取次数：0，慢速预读次数：0。

TryVACRCHARDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：200，慢速物理读取次数：0，慢速预读次数：0。

XML 数据类型占用更少的 I/O。、

同样，执行计划成本显示了两个命令的完全相同的计划。

EventClass	TextData	CPU	Reads	Writes	Duration
SQL:BatchCompleted	SET STATISTICS XML ON	0	0	0	1
SQL:BatchCompleted	SELECT * FROM TryXMLDatatype	0	59	0	308
SQL:BatchCompleted	SELECT * FROM TryVACRCHARDatatype	0	203	0	102
SQL:BatchCompleted	SET STATISTICS XML OFF	0	0	0	1
SQL:BatchCompleted	SET STATISTICS XML ON	0	0	0	0
SQL:BatchCompleted	SELECT * FROM TryXMLDatatype	0	59	0	200
SQL:BatchCompleted	SELECT * FROM TryVACRCHARDatatype	0	203	0	51
SQL:BatchCompleted	SET STATISTICS XML OFF	0	0	0	

为什么两个表的读取操作差别这么大呢？

注意：查询 XML 的时间更长。时间延长的很多不同的因素是基于机器的活动。我将忽略这些不同点，主要是因为没有 CPU 显示。如果涉及到 CPU 活动，它可能解释了时间延长的不同点但是这并不是重点。

让我查询一下每一个表的字段长度：

```
SELECT datalength(MyXML) FROM TryXMLDatatype
Go
SELECT datalength(MyXML) FROM TryVACRCHARDatatype
Go
```

很意外，XML 域的字符比另一个更少。原因如下：

XML--less I/O:

当插入 XML 数据类型时，“附加”数据，如引号和制表符会从域中删除。结果是更加节约了存储空间。

我尝试从 XML 字段拷贝值到 VARCHAR(MAX)字段，如下：

```
truncate table TryVARCHARDatatype
Go
insert into TryVARCHARDatatype (MyXML)
select convert(varchar(max), MyXML) from TryXMLDatatype
Go
```

(作者：Michelle Gutzait 译者：曾少宁 来源：TT 中国)

原文标题：SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一

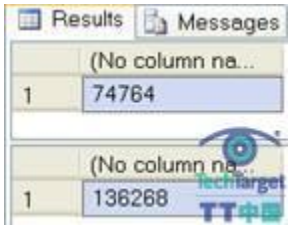
链接：http://www.searchdatabase.com.cn/showcontent_17306.htm

SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之二

结果：

看起来是一样的，但是……

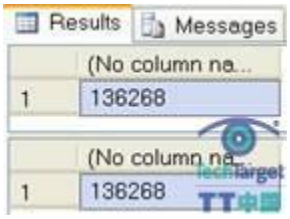
```
SELECT datalength(MyXML) FROM TryXMLDatatype
Go
SELECT datalength(MyXML) FROM TryVACRCHARDatatype
Go
```



这比前面好一些（现在是 136，268，而不是 271，210），但情况仍然相同。如果我将 XML 转化到 VARCHAR(MAX)然后计算长度：

```
SELECT
datalength(convert(varchar(max), MyXML)) FROM TryXMLDatatype
go
SELECT datalength(MyXML)FROM TryVARCHARDatatype
```

```
Go
```



长度相同。

这意味着 XML 数据类型的存储方式更加高效。现在两个表包括了相同的数据，让我们再看看 Profiler 的跟踪命令 select * from both tables (three executions)的结果：

TextData	CPU	Reads	Writes	Duration
SELECT * FROM TryXMLDatatype	16	83	0	273
SELECT * FROM TryVARCHARDatatype	15	103	0	182
SELECT * FROM TryXMLDatatype	16	83	0	95
SELECT * FROM TryVARCHARDatatype	0	103	0	100
SELECT * FROM TryXMLDatatype	0	83	0	77
SELECT * FROM TryVARCHARDatatype	0	103	0	196

XML 数据类型的读取数少了 20%。

I/O 统计：

TryXMLDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：56，慢速物理读取次数：0，慢速预读次数：0。

TryVARCHARDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：100，慢速物理读取次数：0，慢速预读次数：0。

同样，这里 XML 数据类型的读操作也效率更高。

查询到 XML 转换为 VARCHAR（MAX）是怎样的呢？

```
SELECT convert(varchar(max), MyXML) FROM TryXMLDatatype
Go
```

```
SELECT MyXML FROM TryVARCHARDatatype
Go
```

TextData	CPU	Reads	Writes	Duration
SELECT convert(varchar(max),MyXML) ...	0	23	0	293
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	141
SELECT convert(varchar(max),MyXML) ...	15	23	0	199
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	[illegible]
SELECT convert(varchar(max),MyXML) ...	16	23	0	105
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	264

TryXMLDatatype 表：扫描次数：1，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：20，慢速物理读取次数：0，慢速预读次数：9。

TryVARCHARDatatype 表：扫描次数：1，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：100，慢速物理读取次数：0，慢速预读次数：0。

对于不变行为的唯一度量标准是 I/O 读取。而且，XML 转换查询还会使用 CPU 资源来实现转换——这很正常。

结论

XML 数据类型表示存储纯 XML 数据，它不包括不必要的头尾字符。这带来的结果是通过 I/O 方法实现更划算的存储方式，然而这里仍然有一些 CPU 资源用于验证 XML 是有否效。

（作者：Michelle Gutzait 译者：曾少宁 来源：TT 中国）

原文标题：SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之二

链接：http://www.searchdatabase.com.cn/showcontent_17307.htm

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（一）

当 Microsoft 发布 SQL Server 2005 时，它引进了一个新的数据类型：XML。与其它的 SQL Server 数据类型一样，你可以使用 XML 数据类型来定义字段、存储过程和用户自定义方法的变量和参数，并且你可以整体存取 XML 数据——即作为一个值——就像一个 XML 文档一样。然而，与 XML 文档一样，可能有时候当你想存取的只是 XML 数据中的指定值时，你就需要使用到 XQuery 了。

XQuery 是一个功能强大的、专门为存取 XML 数据而设计的脚本语言。SQL Server2005 支持一个允许在 XML 列、变量或参数中存取值的 XQuery 语言子集。通过调用 XML 数据类型支持的方法，就可以在 Transact-SQL 语句中使用 XQuery。其中两种方法分别是 value()和 query()——这对于在 XML 数据中检索指定的元素是特别有用的。

在本文中，我将阐述这两种方法并分别举例说明。注意，本文假定你已经熟悉 T-SQL 和 SQL Server 的 XML。

XML value()方法

当在 Transact-SQL 语句中调用 XML 方法时，需要指定一个 XML 字段、变量或参数名称，并且在其后面括号中包含一个周期、方法名称以及一个 XQuery 表达式。比如，在一个命名为 XmlInfo 的 XML 字段上调用一个 value()方法，你可以使用下面的语法：

```
XmlInfo.value(<XQuery expression>)
```

当你调用方法 value()时，该 XML 方法返回一个指定类型的标量（单一的）值。这个 value()方法有两个参数。第一个是检索的元素和值，而第二个是返回值的数据类型。让我们举一个例子来说明它是如何工作的。下面的语句在 HumanResources 表中的 JobCandidateID 和 Resume 字段中检索数据。JobCandidate 表格（AdventureWorks 示例数据库的一部分）：

```
SELECT JobCandidateID, Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
(/ns:Resume/ns:Name/ns:Name.Last)[1]',
'nvarchar(30)') AS LastName
FROM HumanResources.JobCandidate
```

SELECT 列表中的第二字段（Resume）是以 XML 数据类型定义的。因此，你可以调用任何 XML 方法。在这里，我调用了 value()方法并且传输了两个参数。每个参数都包含在单引号中并且由一个逗号分隔。现在让我们仔细地看一下第一个参数。

第一个参数被分成两个部分并由分号分隔。第一部分表示的是命名空间并赋予“ns”别名。你必须任意类型化的 XML 字段指定命名空间。每一个类型化的字段都关联到一个指定的模式。在 SQL Server Management Studio 中，你可以通过检索 XML 列中整体值，然后点击返回值来确定与 XML 文档相关的模式。XML 文档打开的是一个单独的窗口，如图 1 所示。注意，该模式被定义为根元素的属性。

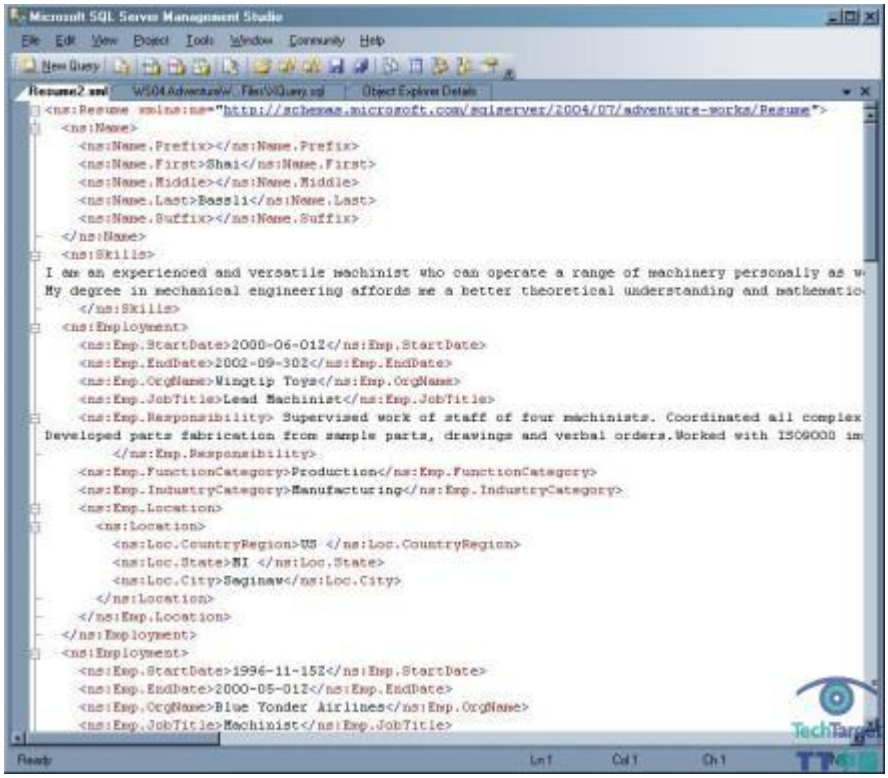


图 1：确定与 XML 文档关联的模式

当调用 value()方法时，在 XML 文档中列出的模式就是作为命名空间的模式。如果 XML 字段是非类型化的（未关联到一个模式），那么就不需要声明命名空间，并且可以省略参数的第一部分。

参数的第二部分定义包含检索的值的 XML 元素。

元素基本上是一个由正斜线分隔的路径名称。当处理类型化的 XML 字段时，你必须先在路径名中的每个节点前加上命令空间别名，然后是冒号。在这里，我在每个节点前加上“ns:”。再次参照图 1，你可以在文档的上方附近看到这个元素。

注意，路径名被包含在括号中并且其后面是[1]。因为 value()方法只能返回多个数量值，因此你必须在括号的后面指定[1]以确保只有一个元素实例返回，即使在 XML 中只有一个实例。在你指定了路径之后，你必须指定数据类型。SQL 语句现在就检索每个职位候选人的姓氏并将它作为 NVARCHAR 数据类型返回。

与其它的语言一样，XQuery 支持广泛的各种不同方法。比如，下面的语句使用的是一个 concat 方法（在第一个参数的第二个部分）来连接名和姓：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName
FROM HumanResources.JobCandidate
```

concat 方法的多个参数是由逗号分隔的。每一个参数都必须串连。注意，第一个和第三个参数使用的是上例中一样的路径架构。更多关于 concat 方法和所有 XQuery 方法的信息，可以阅读 Microsoft SQL Server 2005 Books Online。

（作者：Robert Sheldon 译者：曾少宁 来源：TT 中国）

原文标题：在 SQL Server 2005 中使用 XQuery 检索 XML 数据（一）

链接：http://www.searchdatabase.com.cn/showcontent_17716.htm

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（二）

XML 的 query()方法

虽然 value()方法可以很便捷地在 XML 字段中检索一个值。然而，当经常检索多个值时，你必须使用 XML 的 query()方法。query()方法只需要一个参数并返回一个指定的 XML 元素。比如，在下面的 SQL 语句中返回一个 Education 元素以及每个职位候选人的的子节点。

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
/ns:Resume/ns:Education') AS Education
FROM HumanResources.JobCandidate
```

query()方法的参数是包含在单一的引号中并分成两个部分。同样，当在一个类型化的字段中检索数据时，你必须指定一个命名空间。命名空间的声明方式与 value()方法中的一样。在声明了命名空间之后，便可以指定检索的元素。在这种情况下，如图 2 所示，将返回 Education 元素以及所有的子节点。

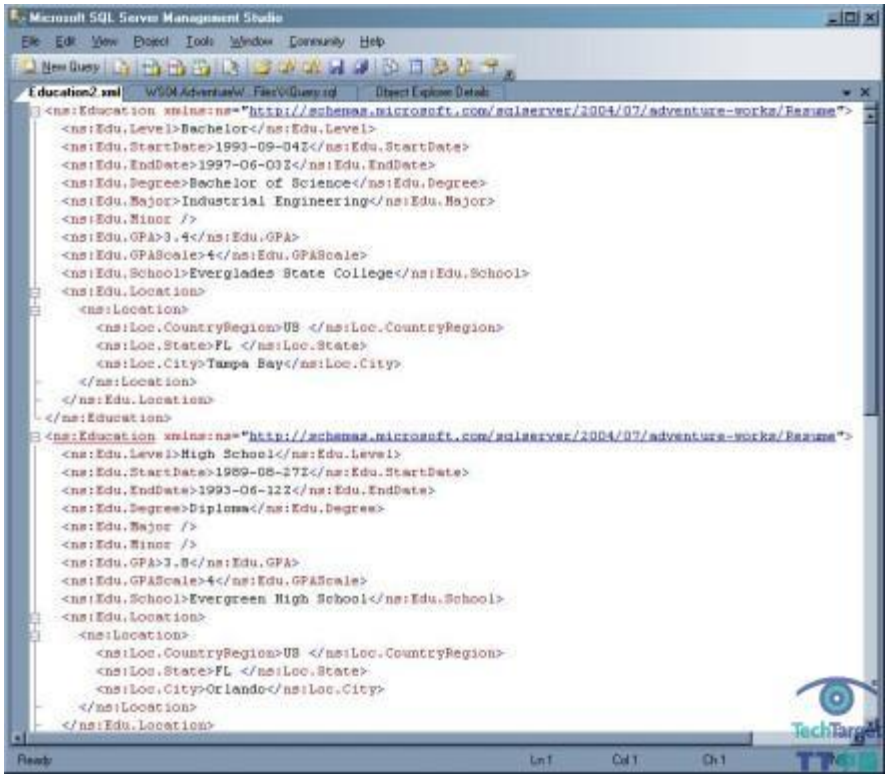


图 2：一旦声明了命名空间，便可以指定检索的 XML 元素。

通过将第二部分的参数转换为 FLWOR（发音为 flower）表达式，我们也可以获得同样的结果。按照定义，该表达式是由“FOR、LET、WHERE、ORDER BY 和 RETURN”子句组成的。但是，注意，目前 SQL Server 并不支持 LET 子句。

下面的 XML query()方法使用的是“FOR”和“RETURN”子句来检索 Education 元素以及子节点：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
```

```
                return $ed')AS Education
        FROM HumanResources.JobCandidate
```

正如你所看到的，FOR 子句包含$ed 变量。你并不需要显式地声明这个变量。在 FOR 子句中使用就可以了。这个变量是用于遍历 Education 元素的。（只要遵循 SQL Server 的命名规范，变量可以使用任何名称。）而 RETURN 子句引用$ed 变量。因此，它返回每一个 Education 元素和子节点。

当然，一般不会只是使用一个 FLWER 表达式来检索一个参数和子节点。但是，你可以在 FOR 子句（在括号中）的路径名称上包含一个表达式来限制返回的结果。比如，下面的语句限制了结果中 Edu.Level 值为“Bachelor”的 Education 元素：

```
      SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
          concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
                (/ns:Resume/ns:Name/ns:Name.Last)[1])',
                     'nvarchar(60)') AS FullName,
                  Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
      for $ed in /ns:Resume/ns:Education[ns:Edu.Level="Bachelor"]
                        return $ed') AS Education
                  FROM HumanResources.JobCandidate
```

注意，我使用的是相等的比较操作符（=）来比较 Edu.Level 元素值和字符串值。这样就构成一个 Boolean 表达式，当结果中包含该元素时为 true。XQuery 支持各种创建 Boolean 表达式的操作符号。你可以在 Microsoft SQL Server 2005 Books online 上阅读支持的操作符。

（作者：Robert Sheldon 译者：曾少宁 来源：TT 中国）

原文标题：在 SQL Server 2005 中使用 XQuery 检索 XML 数据（二）

链接：http://www.searchdatabase.com.cn/showcontent_17760.htm

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（三）

除了在 FOR 子句中使用 Boolean 表达式，还可以在 WHERE 子句中定义相同的逻辑，如下例所示：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
where $ed/ns:Edu.Level="Bachelor"
return $ed') AS Education
FROM HumanResources.JobCandidate
```

在这种情况下，FOR 子句中仅仅包含一个简单的路径，而 WHERE 子句中包含一个 Boolean 逻辑。注意，路径名称使用了$ed 变量来指出了 FOR 子句循环访问的正确的 Education 元素。

在 WHERE 子句中使用这个逻辑可以更易于阅读和编写代码，特别是在使用复杂的 Boolean 表达式的时候。比如，下面的语句使用了 WHERE 子句中的“AND”逻辑操作符号，结果仅限于学士学位中主修学科为商务的记录。

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query(('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
where $ed/ns:Edu.Level="Bachelor" and $ed/ns:Edu.Major="Business"
```

```
                return $ed') AS Education
            FROM HumanResources.JobCandidate
```

两个路径结果由“AND”操作符号连接。因此，只有当这两个条件都必须为真时才返回 Education 元素。

让我们来看一个包含 ORDER BY 子句的例子。下面的语句检索的是与技术和产品相关的就业信息：

```
      SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
          concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
                (/ns:Resume/ns:Name/ns:Name.Last)[1])',
                    'nvarchar(60)') AS FullName,
                  Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
                  for $emp in /ns:Resume/ns:Employment
          where $emp/ns:Emp.FunctionCategory = "Production" or
              $emp/ns:Emp.FunctionCategory = "Technology"
                order by $emp/ns:Emp.EndDate descending
                      return $emp') AS Employment
                  FROM HumanResources.JobCandidate
```

在上面的例子中，语句使用 FOR 子句来确定所需的元素（如，Employment）和 WHERE 子句来限制结果。但是，注意，语句也包含了 ORDER BY 子句。你可以使用这个子句来基于指定节点的排序结果。因此，我基于 Emp.EndDate 元素以降序的形式排序结果。这样，最近的就业信息将出现在列表的最前面。

正如你所看到的，当检索 XML 数据时，value()和 query()方法是非常有用的。然而，我这里所涉及到的都只是些皮毛的知识。XQuery 是一个强大的语言，它可以用来编写复杂查询并明确地按照你需要的方式来返回 XML 数据。由于 XQuery 支持大量各式各样的方法，操作符号和表达式，因此，你可以检索你的 XML 字段、变量和参数中存储的任意元素和属性——以任意需要的格式。同样的，你可以参考 SQL Server 2005 Books Online 上其它关于 XQuery 的信息以及更多的示例。

（作者：Robert Sheldon 译者：曾少宁 来源：TT 中国）

原文标题：在 SQL Server 2005 中使用 XQuery 检索 XML 数据（三）

链接：http://www.searchdatabase.com.cn/showcontent_17790.htm

用 SQL Server 提供的函数处理 XML 文档

自 SQL Server 2000 发布以来，处理 XML 数据就成为了数据库管理员们经常讨论的一个话题。常见的用法是前端应用程序以入参方式传给存储过程 XML 文档。然而，有时你会遇到要处理一个文件夹中一组 XML 文件，将它们加载到数据库中并随后处理到 SQL Server 数据表的情况。这一点更困难，而且介绍如何来做的文档资料也很少。

你可以在几种技术中做选择，但这些技术大部分都十分复杂。我曾用过 SQL Server 集成服务(SSIS)，数据转换服务(DTS)和自己编写的 Windows 应用程序来读取和加载文件。然而，我最喜欢的技术是采用 OPENROWSET 函数来实现它。该函数提供了极大的灵活性，因为你可以在 T-SQL 存储过程中从各个角度控制整个过程。

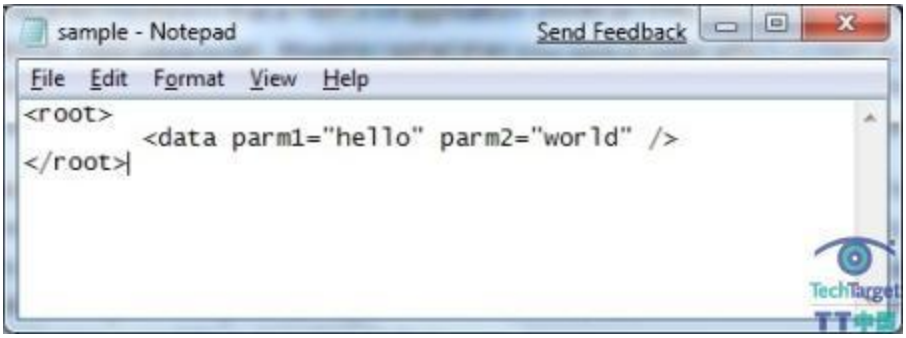
采用这种方法的第一步是创建一个只有一列的表，并把该列定义为 XML 数据类型。临时表也可以：

```
CREATE TABLE #WorkingTable

(Data XML)
```

然后，你可以使用 OPENROWSET 函数，把数据加载到这个表中的一行：

```
INSERT INTO #WorkingTable

SELECT * FROM OPENROWSET (BULK 'D:\Temp\Sample.xml', SINGLE_BLOB) AS data
```

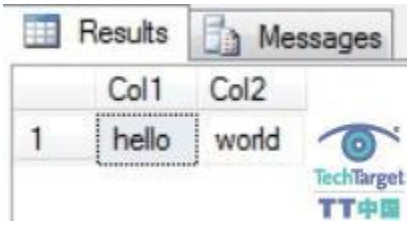
现在，你可以把该表#WorkingTable 数据列中的值放到一个变量里，然后调用 OPENXML 函数来解析 XML 文档，并根据需要使用这些 XML 数据。在本例中，XML 文档内容非常简单，如下图：



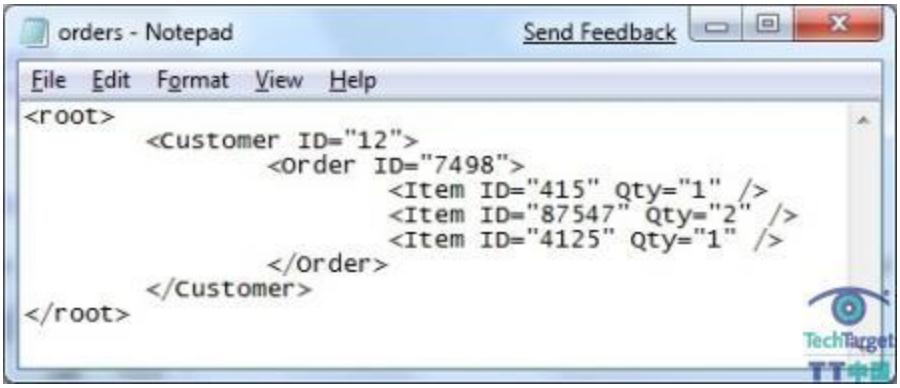
OPENXML 函数需要从临时表中读取这些数据，示例代码如下：

```
DECLARE @XML AS XML, @hDoc AS INT
SELECT @XML = Data FROM #WorkingTable
EXEC sp_xml_preparedocument @hDoc OUTPUT, @XML
SELECT *
FROM OPENXML(@hDoc, '/root/data', 1)
WITH (Col1 VARCHAR(5) '@parm1',
Col2 VARCHAR(5) '@parm2')
EXEC sp_xml_removedocument @hDoc
```

这段查询的输出结果如下图所示：



这是所有 OPENXML 语句的基本语法。在这里，你可以给“root/data”段，“@parm1””和其他 XPath 表达式添加更复杂的 XML 内容。这样你可以处理更复杂的 XML 文档内容中的数据。下面就是一个 XML 内容更复杂一些的例子，其中列出了一些费用清单数据：



正如你所看到的，该 XML 文档有三级内容，需要的数据在“Customer ID”，“Order ID”和“Item ID”三个节点中。在示例代码中，你可以看到我们从“Item ID”节点开始往回解析，从而获取到“Customer ID”和“Order ID”的值：

```
DECLARE @XML AS XML, @hDoc AS INT
SELECT @XML = Data FROM #WorkingTable
EXEC sp_xml_preparedocument @hDoc OUTPUT, @XML
SELECT *
FROM OPENXML(@hDoc, '/root/Customer/Order/Item')
WITH (CustomerId INT '../../@ID',
      OrderId INT '../@ID',
      ItemId INT '@ID',
      Qty INT '@Qty')
EXEC sp_xml_removedocument @hDoc
```

执行后，会产生如下图所示的结果集：

	CustomerId	OrderId	ItemId	Qty
1	12	7498	415	1
2	12	7498	87547	2
3	12	7498	4125	1

如果我们使用同样的该 XML 文档，给第二个客户添加一个订单，我们就会看到如下图的输出内容：

	CustomerId	OrderId	ItemId	Qty
1	12	7498	415	1
2	12	7498	87547	2
3	12	7498	4125	1
4	84	8390	12	1

正如我前面说的，我们从 XML 文档内容的“Item”节点开始处理。如果我们从文档的“Customer”节点开始，并向下解析该文档，我们得到的输出就是不正确的。那样的话，我们就只能得到文档中每一个“Customer ID”的第一个“item”。

OPENXML 函数会得出不正确的结果集，如下图：

	CustomerId	OrderId	ItemId	Qty
1	12	7498	415	1
2	84	8390	12	1

```
SELECT *
FROM OPENXML(@hDoc, '/root/Customer')
WITH (CustomerId INT '@ID',
  OrderId INT 'Order/@ID',
ItemId INT 'Order/Item/@ID',
  Qty INT 'Order/Item/@Qty')
```

在本文中，我们讨论了使用“XPath”最简单的例子。要了解更复杂的实例，你可以参考微软官方的“XPath 语法”选项，来提高使用 T-SQL 语句 XML 引擎的能力。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：用 SQL Server 提供的函数处理 XML 文档

链接：http://www.searchdatabase.com.cn/showcontent_30582.htm

SQL Server XML：释放的利与弊

虽然 SQL Server 支持 XML 但并不意味着在任何情况下 XML 都是最合适的选择。这让我想起了那些相信他们能够在多个 SQL Server 实例之间利用 XML 进行数据传递的架构师们。他们把一组相互关联的关系数据，采用 XML 进行分列存储，利用 c#服务程序来选择数据，并将数据传递给准备接收的 SQL Server 实例。在接收端再对其进行分解，然后采用跟原系统同样的格式进行存储。

采用这种方式，主要的问题是在解析 XML 的时候会增加原系统服务器的 CPU/RAM 资源开销;还会因为 XML 的文本特性而增加网络带宽的占用;而且由于在接收服务器端分解 XML 也一样会给接收服务器带来附加的 CPU/RAM 开销。

我们需要紧记使用 XML 的目的：采用同一种数据格式在不同类型的系统之间传递数据。不可避免的，XML 与生俱来的就具有一些缺点，例如：附加的标记名称会使数据量变大，而且还会因为对 XML 的切割和分析而增大对 CPU/RAM 的资源占用;由于 XML 是基于文档格式的，也就意味着对于那些小数据类型的数据，比如整型，在作为 XML 进行数据传送的时候，会被转化成文本类型再加上它们的标记名称，需要占用的空间从而立刻变大。

过去，已经被使用的 XML 的优点之一，集成数据块(例如，一连串的行数据)，分解数据，然后处理数据将其存储在相关格式当中。在 SQL Server 2008 之前，没有使用 XML，那么就要多次连续地调用相关存储过程来反复执行。从 SQL Server 2008 开始，SQL Server 支持 table-valued 类型的参数传递。table-valued 参数没有与 XML 相关的开销，很大程度的减小了 CPU/RAM 的资源和网络带宽的占用。

由于 CPU/RAM 的负载瓶颈，在剖析和分解 XML 然后添加到 SQL Server 时，也同样关系到可测量性的问题。如果 SQL Server 正处于高负荷运行(或者是准备处于高负荷运行)，那么有一件事情必须要引起注意，就是预期分解的容积不要超过 CPU/RAM 所能负担的能力。当 SQL Server 处于高负荷运行而且 XML 分解还需要占用大量的负载，有一个更好的方法是将处理分解的操作分发到多个服务器上去运行各自运行自定义的写服务，将会在 SQL Server 上分解 XML 和调用存储过程来处理关系数据结果。系统在超负荷运行状态下，通常测量多个服务器的服务会比单个 SQL Server 更容易一些。

（作者：Matthew Schroeder 译者：高奎 来源：TT 中国）

原文标题：SQL Server XML：释放的利与弊

链接：http://www.searchdatabase.com.cn/showcontent_32398.htm

SQL Server XML 拆分选项对比

XML 拆分选项概述

1、用其他语言在一个特定服务器上执行拆分操作

优势：

- 对较大的 XML 文件，拆分和剖析操作具有很快速的可测量性，通过 T-SQL 存储过程在相关的格式下将测量结果传入 SQL Server。
- 在 SQL Server 上对 CPU/RAM 资源占用，显得更为友好。
- 利用 C#，VB.NET 等编程语言可以很容易的对 XML 进行维护，尤其是在分解复杂文件时的速度更快。

缺点：

- 在另外的应用程序层去开发和管理也就意味着会增加开发的工作量和时间。

2、利用 T-SQL/ XQuery /Xpath 在 SQL Server 的内部执行拆分

优点：

- T-SQL 源于 SQL Server，因此它的执行速度照比执行 CLR 代码要快。对比每台独立的服务器，具体的执行效率依赖于 XML 文档，处理方式，网络带宽和等等其它因素的不同而不同。
- 可以很容易的进行灾难恢复。

缺点：

- 对于所有的编程语言来说，目前 XML 功能的通用性还是很有限。
- CPU/RAM 资源占用较大。
- 当 SQL Server 在超负荷工作的时候可测量性将会成为一个问题。

3、利用 CLR 存储过程在 SQL Server 本身执行分解

优点：

- 对于任何一种 CLR 编译语言如 C# ， VB.NET 等都可以提供完整的功能。
- 利用读取器可以更有效的提高处理复杂 XML 文件的能力。

缺点：

- 由于需要在服务器上部署和维护程序集，使得可测量性和灾难恢复等问题变得复杂。
- SQL Server 需要利用 CLR 引擎来执行解释性代码。这种方式在 IL(内部代码)向机器代码转换的过程中会产生一个轻微的延迟，但是这也依赖于 JIT(运行时编译执行技术)编译器的缓存能力所产生的影响。
- 当 SQL Server 在超负荷工作的时候可测量性将会成为一个问题。

(作者：Matthew Schroeder 译者：高奎 来源：TT 中国)

原文标题：SQL Server XML 拆分选项对比

链接：http://www.searchdatabase.com.cn/showcontent_32401.htm

SQL Server XML 拆分示例

利用 Xquery 剖析元素来完成 XML 的分解。记住，Xpath 是 Xquery 的一个子集。Xquery 利用 Xpath 来描述一个 XML 的文档部分。利用“For, Let, Where, Order By, Return”等表达式进行补充说明，或者用 FLWOR(上面五个单词的缩写)进行化简。Xquery 不支持更新操作和全文搜索功能，而这两种功能可能会在以后推出但是目前 Xquery 还只是一种查询语言。

XML 示例

```
<sample1>
<company>TheRealLife</company>
<company>TheRealLife2</company>
<city>Miami</city>
<year-founded>1998</ year-founded>
<industry>software</industry>
</sample1>
```

调用存储过程示例

```
declare @m as xml;
set @m = ' <sample1>
<company>TheRealLife</company>
<company>TheRealLife2</company>
<city>Miami</city>
<year-founded>1998</year-founded>
<industry>software</industry>
</sample1>'
exec dbo.XMLShred @d = @m;
Result set1:
TheRealLife TheRealLife2
Result set2:
<company>TheRealLife</company>
<company>TheRealLife2</company>
```

分解存储过程示例

```
create proc dbo.XMLShred (@d as XML)
AS
select @d.query('
for $step in /sample1/company
return string($step)
')
select @d.query ('/sample1/company')
return 0;
```

第一条 select 语句是一个典型的 FLWOR，将返回一个单个节点的内容，如果 XML 变量中包含多个节点则返回第一个节点内容。结果见(result set 1)。第二条 select 语句则返回所有匹配的节点。在这个示例当中包含多个“company”节点，来举例说明不同之处。如果想要选择不同的行节点可以用下面的语句：select @d.query ('/sample1/company[2]');用这样的方式可以返回指定位置的节点。

考虑到 T-SQL 中处理复杂 XML 的操作功能比较有限，可以选择采用 CLR 存储过程来实现。这种方式主要是基于用 Microsoft .NET Framework CLR 来实现的存储过程和那些支持 CLR 的语言编写的存储过程，例如 C# 或者是 VB.NET。记住，在这种结构下必须要能够将程序集部署到物理的 SQL Server 上面。这是否会成为一个问题，完全依赖于生产服务器是否被锁定。而且在做系统还原的时候，程序集也必须要包含在还原的范围当中，所以一定要仔细的作好计划，以正确的处理其在灾难恢复，可测量性和其它方面的影响。

当你要决定如何分解 XML 文件的时候，一定要注意这些潜在的影响。现在我们已经对每个 XML 分解选项的优点，缺点和注意事项都做了基本的说明，可以针对每个给定的体系结构选择最为合适的处理方式。

(作者：Matthew Schroeder 译者：高奎 来源：TT 中国)

原文标题：SQL Server XML 拆分示例

链接：http://www.searchdatabase.com.cn/showcontent_32402.htm

对比 XML AUTO 与 T-SQL 命令（上）

XML AUTO 函数与 T-SQL 命令对比

作为一个 DBA，我倾向于关注性能方面的问题，因此我要确定使用 XML（扩展标记语言）不会对性能产生影响。在本文中，我会通过比较 XML AUTO 功能和标准的 T-SQL 命令来显示性能上的差异。在测试过程中，我仅会涉及到 XML AUTO 功能的一个比较小的基础部分，同时我还建议大家在自己的环境中测试将要使用的 XML 功能。

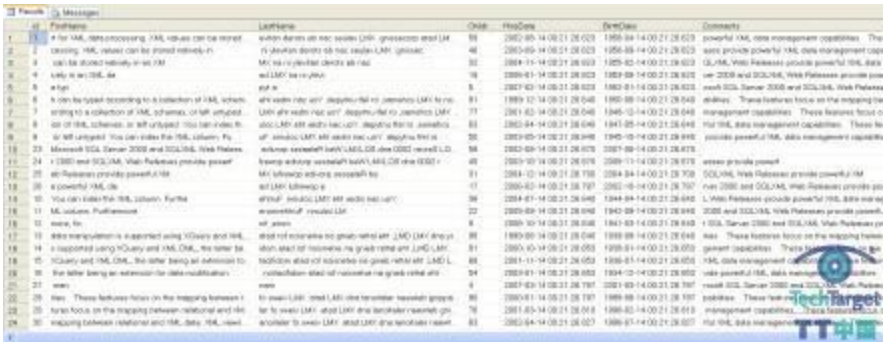
测试环境描述

为了测试,我已经创建了一张表,往里插入了 10000 条记录,描述如下:

```
create table Employes (
      id                int identity      not null,
      FirstName         varchar(100)      not null,
      LastName          varchar(100)      not null,
      Children          smallint          null,
      HireDate          datetime          not null,
      BirthDate         datetime          null,
      Comments          varchar(1000)     null
)
go
declare     @i          int,
            @Name       varchar(1000),
            @rand1      smallint,
            @rand2      tinyint

set @Name = 'Microsoft SQL Server 2000 and SQLXML Web Releases provide
powerful XML data management capabilities.
These features focus on the mapping between relational and XML data.
XML views of relational data can be defined
using annotated XSD (AXSD) to provide an XML-centric approach that
supports bulk load of XML data,
and query and update capabilities on XML data. Transact-SQL extensions
provide SQL-centric approach
for mapping relational query results to XML (using FOR XML) and
generating relational views from XML (using OpenXML).
Microsoft SQL Server 2005 provides extensive support for XML data
processing. XML values can be stored natively in an XML data type
column, which can be typed according to a collection of XML schemas, or
left untyped. You can index the XML column. Furthermore, fine-grained
data manipulation is supported using XQuery and XML DML, the latter
being an extension for data modification.
In addition, the SQLXML, FOR XML and OpenXML features have been
extended in SQL Server 2005. Together
'     -- Total actual length of string is 1000

set @i = 1

-- Create 10,000 rows in the table with random values:
while @i <= 100000
begin
      -- Two random numbers will be used to create the random data:
      SELECT    @rand1 =
                convert(smallint,substring(convert(varchar(20),
                convert(decimal(20,9),RAND( @i))),6,3))
      SELECT    @rand2 =
                convert(tinyint,substring(convert(varchar(20),
                convert(decimal(20,9),RAND( @i*10))),7,2))

      insert into Employes
      (FirstName, LastName, Children, HireDate, BirthDate, Comments)
      select substring (@Name,@rand1,@rand2),
             reverse(substring (@Name,@rand1,@rand2)), @rand2,
             dateadd (mm,@rand2*(-1),getdate()),
             dateadd (mm,@rand1*(-1),getdate()),
             substring (@Name,@rand2,@rand1)

      set @i = @i + 1
end
go
```


此表包含以下记录:

```
select * from Employes
```



XML Auto 处理大量数据

我在刚才那张表中使用如下查询语句,以一个标准的查询开始,并且之后每次都向命令上加上更多的 XML 结构:

```
SELECT * FROM Employes
Go
SELECT * FROM Employes FOR XML AUTO
go
SELECT * FROM Employes FOR XML AUTO, TYPE
go
SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS
go
SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS,
ROOT
go
```

我使用 SQL Profiler 来监控这些命令。我发现 Duration 列在 SQL Profiler 中是不准确的。当 Query Analyzer 仍在处理结果时，执行过了的 XML 命令已经在 Profiler 中显示了。因此，我将会忽视此列，并查找其它的资源:CPU 和 I/O。下面是对五个命令执行三次的监控结果：

Event...	TextData	CPU	Reads	Writes	Duration
SQL...	SELECT * FROM Employes	313	8253	0	45804
SQL...	SELECT * FROM Employes FOR XML AUTO	3422	8253	0	53765
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE	2796	677187	4798	28159
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS	2859	684832	250	23571
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS, ROOT	2579	678068	6950	9901
SQL...	SELECT * FROM Employes	468	8253	0	11207
SQL...	SELECT * FROM Employes FOR XML AUTO	3891	8253	0	11501
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE	2453	673746	8234	12680
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS	3172	683532	8183	5124
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS, ROOT	2984	683792	1262	6319
SQL...	SELECT * FROM Employes	485	8253	0	[illegible]
SQL...	SELECT * FROM Employes FOR XML AUTO	3390	8253	0	[illegible]042
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE	2703	673503	8465	[illegible]
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS	3047	684902	2884	4713
SQL...	SELECT * FROM Employes FOR XML AUTO, TYPE, ELEMENTS, ROOT	2812	684005	1002	[illegible]

这看起来似乎 T-SQL 查询比其它方式执行的效果要好一些。XML AUTO 的 XML 查询产生同样的 I/O，却多使用了八倍的 CPU 资源。复杂的 XML 命令比 T-SQL 命令消耗了超过 80 倍的读取操作，同时还有许多写操作和上面的六倍的 CPU。

使用以下命令来分析 I/O 统计：

```
Table 'Employes'. Scan count 1, logical reads 8247, physical
  reads 0, read-ahead reads 7520, lob logical reads 0, lob
          physical reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 8247, physical
  reads 0, read-ahead reads 4221, lob logical reads 0, lob
          physical reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 8247, physical
 reads 500, read-ahead reads 2586, lob logical reads 0, lob
          physical reads 0, lob read-ahead reads 0.
 Table 'Worktable'. Scan count 0, logical reads 7, physical
 reads 1, read-ahead reads 0, lob logical reads 229128, lob
       physical reads 285, lob read-ahead reads 43082.
Table 'Employes'. Scan count 1, logical reads 8247, physical
  reads 0, read-ahead reads 1394, lob logical reads 0, lob
          physical reads 0, lob read-ahead reads 0.
 Table 'Worktable'. Scan count 0, logical reads 7, physical
 reads 0, read-ahead reads 0, lob logical reads 230170, lob
        physical reads 0, lob read-ahead reads 43115.
Table 'Employes'. Scan count 1, logical reads 8247, physical
 reads 125, read-ahead reads 3835, lob logical reads 0, lob
          physical reads 0, lob read-ahead reads 0.
 Table 'Worktable'. Scan count 0, logical reads 7, physical
 reads 0, read-ahead reads 0, lob logical reads 230170, lob
       physical reads 140, lob read-ahead reads 43115.
```

很明显，更为复杂的 XML 命令正创建一张工作表。 也显示了每一个命令在处理比之前更多的工作。

(作者：Michelle Gutzait 译者：张峰 来源：TT 中国)

原文标题：对比 XML AUTO 与 T-SQL 命令（上）

链接：http://www.searchdatabase.com.cn/showcontent_27868.htm

对比 XML AUTO 与 T-SQL 命令（下）

XML Auto 处理比较少的数据

我将使用同一个表做同样的测试，只是读取更少的数据。

首先，我将在 ID 列上创建一个索引：

Ø create unique clustered index UQ_Employes on Employes （id）

然后，我用 WHERE 子句查询此表，先查询 100 条记录，然后查询 1000 条记录：

100 条记录如下：

```
SELECT * FROM Employes where id between 5000 and
                        5100
                         go
SELECT * FROM Employes where id between 5000 and
                        5100
                    FOR XML AUTO
                         go
SELECT * FROM Employes where id between 5000 and
                        5100
                FOR XML AUTO, TYPE
                         go
SELECT * FROM Employes where id between 5000 and
                        5100
           FOR XML AUTO, TYPE, ELEMENTS
                         go
SELECT * FROM Employes where id between 5000 and
                        5100
       FOR XML AUTO, TYPE, ELEMENTS, ROOT
                         go
```

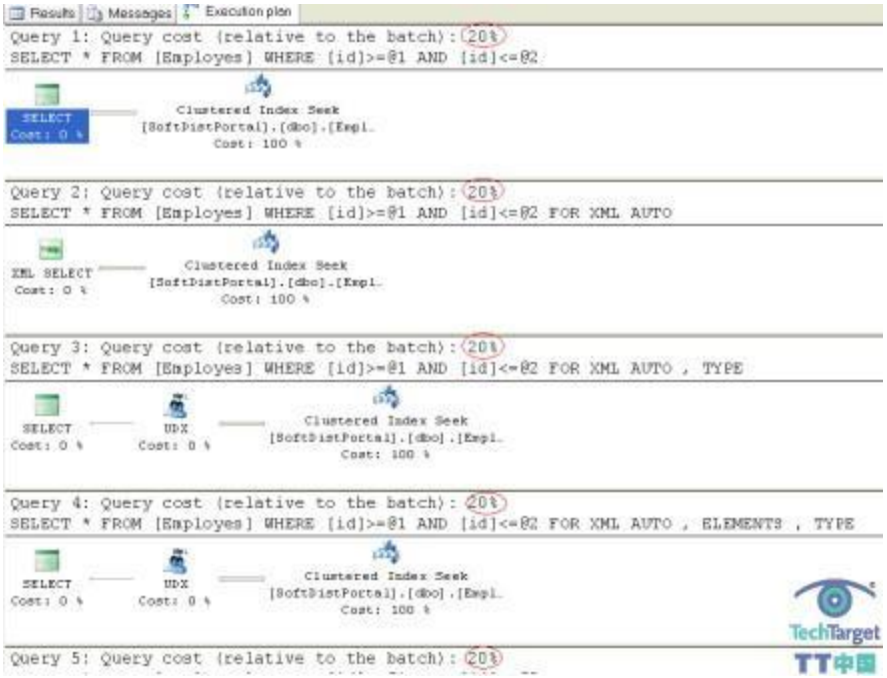
SQL Profiler 显示这些命令的使用没有实质差别：

TextData	CPU	Reads	Writes	Duration
SELECT * FROM Employes where id between 5000 and 5100	0	12	0	276
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO	0	12	0	359
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE	0	12	0	100
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS	0	12	0	86
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS, ROOT	0	12	0	85
SELECT * FROM Employes where id between 5000 and 5100	0	12	0	172
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO	0	12	0	312
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE	16	12	0	91
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS	0	12	0	94
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS, ROOT	0	12	0	95
SELECT * FROM Employes where id between 5000 and 5100	0	12	0	[illegible]
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO	0	12	0	332
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE	0	12	[illegible]	[illegible]
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS	0	12	0	81
SELECT * FROM Employes where id between 5000 and 5100 FOR XML AUTO, TYPE, ELEMENTS, ROOT	0	12	[illegible]	[illegible]

同样，没有创建工作表，且所有的命令有着相同数量的工作：

```
Table 'Employes'. Scan count 1, logical reads 12, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 12, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 12, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 12, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
reads 0, lob read-ahead reads 0.
Table 'Employes'. Scan count 1, logical reads 12, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
reads 0, lob read-ahead reads 0.
```

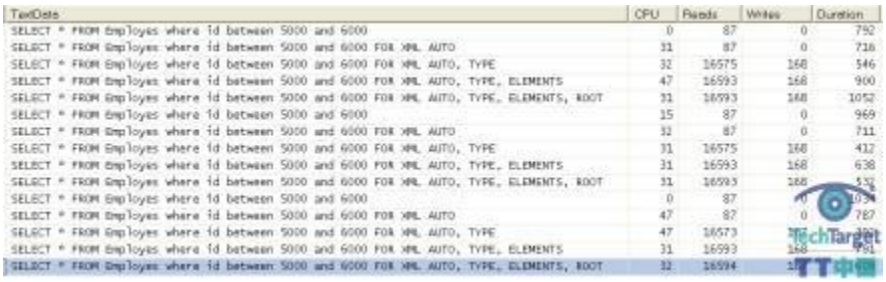
执行计划同样也显示了相同的工作量：



1000 条记录：

```
SELECT * FROM Employes where id between 5000 and
                        6000
                         go
SELECT * FROM Employes where id between 5000 and
                        6000
                    FOR XML AUTO
                         go
SELECT * FROM Employes where id between 5000 and
                        6000
                 FOR XML AUTO, TYPE
                         go
SELECT * FROM Employes where id between 5000 and
                        6000
            FOR XML AUTO, TYPE, ELEMENTS
                         go
SELECT * FROM Employes where id between 5000 and
                        6000
        FOR XML AUTO, TYPE, ELEMENTS, ROOT
                         go
```

SQL Profiler:

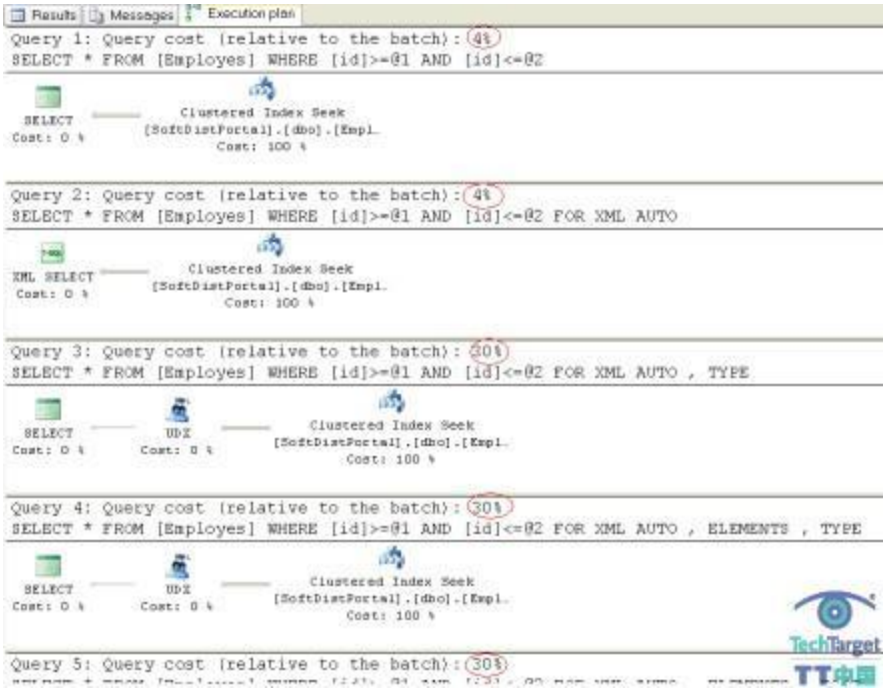


TextData	CPU	Reads	Writes	Duration
SELECT * FROM Employes where id between 5000 and 6000	0	87	0	792
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO	31	87	0	716
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE	32	16575	168	546
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS	47	16593	168	900
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS, ROOT	31	16593	168	1052
SELECT * FROM Employes where id between 5000 and 6000	15	87	0	969
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO	32	87	0	711
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE	31	16575	168	412
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS	31	16593	168	638
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS, ROOT	31	16593	168	532
SELECT * FROM Employes where id between 5000 and 6000	0	87	0	1024
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO	47	87	0	787
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE	47	16573	168	791
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS	31	16593	168	791
SELECT * FROM Employes where id between 5000 and 6000 FOR XML AUTO, TYPE, ELEMENTS, ROOT	32	16594	168	791

I/O 统计：

```
  Table 'Employes'. Scan count 1, logical reads 87, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
              reads 0, lob read-ahead reads 0.
  Table 'Employes'. Scan count 1, logical reads 87, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
              reads 0, lob read-ahead reads 0.
  Table 'Employes'. Scan count 1, logical reads 87, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
              reads 0, lob read-ahead reads 0.
  Table 'Worktable'. Scan count 0, logical reads 7, physical
   reads 0, read-ahead reads 0, lob logical reads 5222, lob
         physical reads 0, lob read-ahead reads 242.
  Table 'Employes'. Scan count 1, logical reads 87, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
              reads 0, lob read-ahead reads 0.
  Table 'Worktable'. Scan count 0, logical reads 7, physical
   reads 0, read-ahead reads 0, lob logical reads 5229, lob
         physical reads 0, lob read-ahead reads 245.
  Table 'Employes'. Scan count 1, logical reads 87, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical
              reads 0, lob read-ahead reads 0.
  Table 'Worktable'. Scan count 0, logical reads 7, physical
   reads 0, read-ahead reads 0, lob logical reads 5229, lob
         physical reads 0, lob read-ahead reads 245.
```

执行计划：



何时使用工作表?

在优化器中使用工作表有何限制呢?这取决于优化器处理内存的工作量及 XML 分析器中的数据量。我的查询返回的 XML 被存储在内存中的一个大的 XML 变量中。以上限制不是一个明确的数据。SQL Server Developer Center 的技术人员表示：“XML 类型的变量和参数可达 2GB。数据量小的时候，它们使用主存，但是，大数据量时被存储在 tempdb 中”。

结论：

XML 数据和函数比标准的 T-SQL 要使用更多的资源。因此，如果查询处理大量数据时或 XML 函数更加复杂，就考虑在数据库级别上使用标准的 T-SQL。通常建议大家测试 XML 性能，来确保在数据库中使用 XML 不会降低应用的性能。

(作者：Michelle Gutzait 译者：张峰 来源：TT 中国)

原文标题：对比 XML AUTO 与 T-SQL 命令（下）

链接：http://www.searchdatabase.com.cn/showcontent_27871.htm

使用 XML 在 SQL Server 上创建计算列（上）

在 SQL Server 数据库中，当你想使用一个数据，而这个数据不保存在表中，计算列很有用。例如，你有一张表，它包括列 dollar amounts, wholesale prices 和 retail prices。你肯定不想在每次查询表时来计算那两列之间的差值，你希望将其值保存在第三列中，让其自动计算前两列之间的差值。而此列就是计算列。

在 SQL Server 中使用 XML 数据来创建计算列，你的列定义必须包含必要的用来检测向列中插入的是什么数据的表达式。例如，在上面的例子中，你的表达式应该从 retail 列中的值减去 wholesale 列中的值。当你添加或更新表中的数据行时，差值将自动插入至计算列中。

你可以很容易地在两个或更多的包含字符串或数字类型值的列的基础上创建计算列。(更多关于如何创建此类型的计算列的详细信息，请参考 Microsoft SQL Server Books Online)。然而，如果你想要基于指定的 XML 列中元素值创建一个计算列，该过程相对更加复杂一些。因为你必须使用 Xquery 表达式来从 XML 列中获取指定元素数据,且 SQL Server 不支持在计算列的定义中使用 Xquery 表达式。

要解决此问题，可以创建一个函数来接收你想包含在计算列中的 XML 数据，并在计算列定义中调用此函数。更好的示范这是如何工作的，我们在这给出一个例子。我在 SQL Server 2005 的示例数据库 AdventureWorks 中创建以下的架构和表:

```
USE AdventureWorks;
GO
CREATE SCHEMA hr
GO
SELECT TOP 10 JobCandidateID AS CandidateID,
[Resume] AS JobResume
INTO hr.CandidateNames
FROM HumanResources.JobCandidate
GO
```

正如名称所示，HumanResources.JobCandidate 表中的 Resume 列是一个 XML 列，它包含侯选人的履历信息。我从这张表中提取数据来创建 hr 架构中的 CandidateNames 表。(我创建了一个单独的表，因为我希望可以修改表定义，从而可以增加计算列)

在建立好测试环境后，你可以创建函数。函数应该包括在从指定的 XML 列中获取数据时所需的 XQuery 表达式。例如，以下函数接收工作候选人的姓名，并保存在 JobResume 列中：

```
CREATE FUNCTION hr.FullName (@name XML)
RETURNS NVARCHAR(60) AS
BEGIN
RETURN @name.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1], " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])','nvarchar(60)')
END
```

正如你所看到的，函数 FullName 带一个输入参数，该参数被定义成 XML 类型。这个做法是当调用此函数时，可以把包含所需提取的数据的 XML 列名称作为输入值来使用。

(作者：Robert Sheldon 译者：张峰 来源：TT 中国)

原文标题：使用 XML 在 SQL Server 上创建计算列（上）

链接：http://www.searchdatabase.com.cn/showcontent_27730.htm

使用 XML 在 SQL Server 上创建计算列（下）

Value()方法带两个参数。第一个参数定义了目标 XML 列使用的名称空间，第二个参数包含接收实际数据的 Xquery 表达式。在这个例子中，表达式使用 concat()方法来连接姓与名，就像它们在 XML 文件中。要想了解更多的关于如何使用 value()方法，以及如何创建 Xquery 表达式，请查看我的文章《Retrieve XML data values with XQuery》。

一旦创建了函数，你可以通过从 hr.CandidateNames 表的 JobResume 列中接收数据测试:

```
SELECT CandidateID, hr.FullName(JobResume) AS FullName

FROM hr.CandidateNames
```

正如你所看到的，我已经传入了 XML 列名称，将其做为函数 FullName 的一个参数。SELECT 语句应该返回以下结果：

CandidateID	FullName
1	Shai Bassli
2	Max Benson
3	Krishna Sunkammurali
4	Stephen Jiang
5	Thierry D'Hers
6	Christian Kleinerman
7	Lionel Penuchot
8	Peng Wu
9	Shengda Yang

10	Tai Yee

(影响 10 行)

注意，以上结果包含姓与名，正如它们在 XML 列中显示的一样。如果回到函数定义，可发现在 value()方法中使用的 Xquery 表达式指定了此表达式返回值为 NVARCHAR(60)类型，以适应 Unicode 字符，如查询结果集的最后三行中的那些字符。

一旦函数经过测试，你就可以开始创建计算列：以下 ALTER TABLE 语句添加了 FullName 列到 CandidateNames 表中来:

```
ALTER TABLE hr.CandidateNames

ADD FullName AS hr.FullName(JobResume)
```

我已经在计算列表达式中使用 FullName 函数，并将列 JobResume 作为参数传入函数。在运行 ALTER TABLE 语句后，你可以用以下 SELECT 语句测试数据是否已经被插入到计算列中：

```
SELECT CandidateID, FullName FROM hr.CandidateNames
```

运行以上语句后，应该返回与上文中相同的结果集。

这就是在 SQL Server 中基于 XML data 数据创建的一个计算列。关键是创建一个函数来运行 Xquery 表达式，且稍后在计算列中使用此函数定义。要了解更多关于计算列、XML 列、Xquery 表达式的详细信息，请查看 Microsoft SQL Server 在线书籍。

(作者：Robert Sheldon 译者：张峰 来源：TT 中国)

原文标题：使用 XML 在 SQL Server 上创建计算列（下）

链接：http://www.searchdatabase.com.cn/showcontent_27731.htm

在 SSAS 中使用 XMLA 命令进行跟踪管理（上）

你可以使用 SQL Server Profiler 来监控，解决和调试 Microsoft Analysis Services（MSAS） 2005 故障。如果你仅是偶尔用一下此工具，那么用 SQL Profiler 的图形界面来启动、改变、停止跟踪就足够了。

如果你正管理着多个 MSAS 的实例，有一个更好的选择是使用 XMLA 命令来使跟踪自动处理。你可以通过 SQL Server Management Studio 或 ASCMD.exe 工具包向 Analysis Services 提交 XMLA 命令。在这篇文章里，我将描述如何在 SQL Server 2005 Analysis Services 中使用 XMLA 来管理跟踪。

创建，修改，删除跟踪

你可以使用 CREATE TRACE 命令来启动一个新的跟踪。此命令允许指定跟踪的标识，名称，日志文件（扩展名为.TRC）保存的位置，此命令还定义了你希望在跟踪里监控的事件和列。跟踪识别标识必须是唯一的,或者是 Analysis Services 的当前实例。你可以直接使用 SQL Profiler 中的基本语法。所有的语句大致如下：

```
<Batch
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <Create
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
    <ObjectDefinition>
      <Trace>
        <ID>My sample trace</ID>
        <Name>My sample trace</Name>
        <LogFileName>C:\sample_trace.trc</LogFileName>
        <LogFileAppend>1</LogFileAppend>
        <AutoRestart>1</AutoRestart>
        <LogFileSize>5</LogFileSize>
        <LogFileRollover>1</LogFileRollover>
        <Events>
          <Event>
            <EventID>5</EventID>
            <Columns>
              <ColumnID>7</ColumnID>
              <ColumnID>15</ColumnID>
              <ColumnID>39</ColumnID>
              <ColumnID>8</ColumnID>
              <ColumnID>12</ColumnID>
              <ColumnID>28</ColumnID>
              <ColumnID>32</ColumnID>
              <ColumnID>40</ColumnID>
              <ColumnID>1</ColumnID>
              <ColumnID>13</ColumnID>
              <ColumnID>25</ColumnID>
              <ColumnID>33</ColumnID>
              <ColumnID>41</ColumnID>
              <ColumnID>2</ColumnID>
              <ColumnID>14</ColumnID>
              <ColumnID>42</ColumnID>
              <ColumnID>3</ColumnID>
              <ColumnID>11</ColumnID>
              <ColumnID>43</ColumnID>
            </Columns>
    </Event>
          <Event>
            <EventID>6</EventID>
            <Columns>
              <ColumnID>7</ColumnID>
              <ColumnID>15</ColumnID>
              <ColumnID>23</ColumnID>
              <ColumnID>39</ColumnID>
              <ColumnID>8</ColumnID>
              <ColumnID>24</ColumnID>
              <ColumnID>32</ColumnID>
              <ColumnID>40</ColumnID>
              <ColumnID>1</ColumnID>
              <ColumnID>9</ColumnID>
              <ColumnID>25</ColumnID>
              <ColumnID>33</ColumnID>
              <ColumnID>41</ColumnID>
              <ColumnID>2</ColumnID>
              <ColumnID>6</ColumnID>
              <ColumnID>10</ColumnID>
              <ColumnID>14</ColumnID>
              <ColumnID>22</ColumnID>
              <ColumnID>42</ColumnID>
              <ColumnID>3</ColumnID>
              <ColumnID>11</ColumnID>
              <ColumnID>43</ColumnID>
              <ColumnID>4</ColumnID>
              <ColumnID>12</ColumnID>
              <ColumnID>28</ColumnID>
              <ColumnID>5</ColumnID>
              <ColumnID>13</ColumnID>
            </Columns>
```

注意一下 LogFileSize 和 LogFileRollover 标签。前者指定了每个.trc 文件的最大尺寸;后者告诉 MSAS 当现有文件达到最大尺寸后是否要新建一个文件。如果 LogFileRollover 被设置为 1，则一旦达到了最大日志文件尺寸，MSAS 将会创建一个新的文件，并在文件名后追加一个序列数值。否则，只要日志文件达到其最大尺寸，跟踪将会悄悄地停止。AutoRestart 标签提示 MSAS 是否在服务启动时是否启动跟踪。尽管这个选项很有用，但是如果不小心，很容易引起问题。

在一个负载比较大的服务器上，Analysis Services 跟踪增长速度很快，典型地是，如果你正监控着大量的详细的事件,例如：“Query Subcube Verbose”事件。如果你允许此种跟踪在每次 MSAS 服务重启时重启，你可能会很快用完这个存储跟踪文件的磁盘的空间。如果你告诉 MSAS 不要重启跟踪服务，跟踪要么通过 DELETE 命令显性地停止，要么通过停止服务隐性地停止。Filter 标签允许你为跟踪中的包含或排除的事件指定标准。例如，以上跟踪示例仅包含那些持续时间在 100 毫秒或以上的事件。它也排除了任何联合某个 SQL Server Profiler 跟踪的事件。

一旦向 Analysis Services 实例提交了 CREATE 命令，跟踪将会启动记录事件 - 不像 Profiler 的图形界面,XMLA 是不需要“start trace”命令的）。

如果你在创建跟踪之后改变了主意，想修改跟踪定义， 你可以使用 ALTER 命令。例如，我们可以使用以下命令将自动启动属性设置为 false。记住，尽管你可能希望仅修改跟踪的一个属性，但你必须还要把剩下的标签用 ALTER 命令包含进来:

```
<Batch
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <Alter
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
    <Object>
      <TraceID>my sample trace</TraceID>
    </Object>
    <ObjectDefinition>
      <Trace>
        <ID>My sample trace</ID>
        <Name>My sample trace</Name>
        <LogFileName>C:\sample_trace.trc</LogFileName>
        <LogFileAppend>1</LogFileAppend>
        <AutoRestart>0</AutoRestart>
        <LogFileSize>1</LogFileSize>
        <LogFileRollover>1</LogFileRollover>
        <Events>
          <Event>
            <EventID>5</EventID>
            <Columns>
              <ColumnID>7</ColumnID>
              <ColumnID>15</ColumnID>
              <ColumnID>39</ColumnID>
              <ColumnID>8</ColumnID>
              <ColumnID>12</ColumnID>
              <ColumnID>28</ColumnID>
              <ColumnID>32</ColumnID>
              <ColumnID>40</ColumnID>
              <ColumnID>1</ColumnID>
              <ColumnID>13</ColumnID>
              <ColumnID>25</ColumnID>
              <ColumnID>33</ColumnID>
              <ColumnID>41</ColumnID>
              <ColumnID>2</ColumnID>
              <ColumnID>14</ColumnID>
              <ColumnID>42</ColumnID>
              <ColumnID>3</ColumnID>
              <ColumnID>11</ColumnID>
              <ColumnID>43</ColumnID>
            </Columns>
          </Event>
          <Event>
            <EventID>6</EventID>
            <Columns>
              <ColumnID>7</ColumnID>
              <ColumnID>15</ColumnID>
              <ColumnID>23</ColumnID>
              <ColumnID>39</ColumnID>
              <ColumnID>8</ColumnID>
              <ColumnID>24</ColumnID>
              <ColumnID>32</ColumnID>
              <ColumnID>40</ColumnID>
              <ColumnID>1</ColumnID>
              <ColumnID>9</ColumnID>
              <ColumnID>25</ColumnID>
              <ColumnID>33</ColumnID>
              <ColumnID>41</ColumnID>
              <ColumnID>2</ColumnID>
              <ColumnID>6</ColumnID>
              <ColumnID>10</ColumnID>
              <ColumnID>14</ColumnID>
              <ColumnID>22</ColumnID>
              <ColumnID>42</ColumnID>
              <ColumnID>3</ColumnID>
              <ColumnID>11</ColumnID>
              <ColumnID>43</ColumnID>
              <ColumnID>4</ColumnID>
              <ColumnID>12</ColumnID>
              <ColumnID>28</ColumnID>
              <ColumnID>5</ColumnID>
```

如果你想停用跟踪,就使用 DELETE 命令，它有一个相当简单易懂的语法 - 你只需指定先前创建的跟踪的标识，如下所示：

```
<Delete
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <Object>
    <TraceID>my sample trace</TraceID>
  </Object>
</Delete>
```

此命令停止跟踪，但是不会删除.trc 文件。你看过这些文件文件后，就可以手动删除它们。

（作者：Baya Pavliashvili 译者：张峰 来源：TT 中国）

原文标题：在 SSAS 中使用 XMLA 命令进行跟踪管理（上）

链接：http://www.searchdatabase.com.cn/showcontent_28452.htm

在 SSAS 中使用 XMLA 命令进行跟踪管理（下）

回顾当前跟踪

如果你想通过自动启动选项删除几周前创建的跟踪,但你又忘了此跟踪的标识，这该如何处理呢?别担心,你可以运行 DISCOVER_TRACES 命令,就像下边的命令一样,来重新找回正运行在你的分析服务器上的跟踪:

```
<Discover xmlns="urn:schemas-microsoft-com:xml-analysis">
      <RequestType>DISCOVER_TRACES</RequestType>
    <Restrictions>
            <RestrictionList>
            </RestrictionList>
      </Restrictions>
    <Properties>
            <PropertyList>
            </PropertyList>
      </Properties>
</Discover>
```


除了 XSD 架构（与此篇文章无关），以上命令返回以下输出：

```
<return xmlns="urn:schemas-microsoft-com:xml-analysis">
  <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <row>
      <TraceID>FlightRecorder</TraceID>
      <TraceName>FlightRecorder</TraceName>
      <LogFileName>\\?\C:\Program Files\Microsoft SQL
Server\MSSQL.3\OLAP\Log\FlightRecorderCurrent.trc</LogFileName>
      <LogFileSize>10485760</LogFileSize>
      <LogFileRollover>true</LogFileRollover>
      <AutoRestart>false</AutoRestart>
      <CreationTime>2008-05-03T21:04:20</CreationTime>
    </row>
    <row>
      <TraceID>MicrosoftProfilerTrace1210007958</TraceID>
      <TraceName>MicrosoftProfilerTrace1210007958</TraceName>
      <AutoRestart>false</AutoRestart>
      <CreationTime>2008-05-05T17:19:18</CreationTime>
    </row>
    <row>
      <TraceID>My sample trace</TraceID>
      <TraceName>My sample trace</TraceName>
      <LogFileName>C:\sample_trace.trc</LogFileName>
      <LogFileSize>1048576</LogFileSize>
      <LogFileRollover>true</LogFileRollover>
      <AutoRestart>true</AutoRestart>
      <CreationTime>2008-05-05T17:29:38</CreationTime>
    </row>
  </root>
</return>
```

此输出显示了当前正在在给出的 Analysis Services 实例上执行的三个跟踪 - Flight Recorder 跟踪，我刚创建的示例跟踪，以及用 SQL Profiler 启动的一个跟踪。Flight Recorder 是一个默认的当 MSAS 启动时自动启动的跟踪。使用 out-of-the-box 配制，此跟踪收集最少的问题处理信息;如果你需要此跟踪收集更多的信息，编辑 Analysis Services 安装目录下“bin”子目录下的 flightrecordertracedef.xml 文件。如果你想通过配制 Analysis Services 将 MDX 查询日志写到 SQL Server 数据库,你可以注意另一个专为查询日志创建的内部 MSAS 跟踪。假设我像之前一样重新运行同一个 DISCOVER_TRACES 命令,我将会查询到查询日志跟踪的以下行:

```
<row>
    <TraceID>DBQueryLog</TraceID>
    <TraceName>DBQueryLog</TraceName>
    <AutoRestart>false</AutoRestart>
    <CreationTime>2008-05-05T17:37:09</CreationTime>
  </row>
```

不管此跟踪是如何创建的,你可以通过 SQL Profiler 重新查看其输出。一旦你打开了跟踪文件，你可以将其内容保存到一个 SQL Server 表中以获更多细节的分析信息。

选择适当的事件和列来跟踪

如何着手在 CREATE 或 ALTER 跟踪语句里指定合适的事件和列标识呢?不幸的是,列和事件标识没有备有证明文件。然而，如同往常一样，我们可以找我们的朋友 SQL Profiler 来寻求帮助。在 SQL Profiler 中启动两个跟踪，一个是用默认的事件和列来启动，另一个使用你自己想通过 XMLA 创建的跟踪来监控的事件和列。不要忘记根据自己的喜好设置任何所需的过滤器和秩序输出列。然而当你创建第二个跟踪时检查向第一个 SQL Profiler 窗口发送的 CREATE 语句。确实，努力记住列和事件标识是不必须的;你可以简单地从 SQL Profiler 将事件和列标识进行复制与粘贴，然后调整为你在 XMLA 中所见。

(作者：Baya Pavliashvili 译者：张峰 来源：TT 中国)

原文标题：在 SSAS 中使用 XMLA 命令进行跟踪管理（下）

链接：http://www.searchdatabase.com.cn/showcontent_28454.htm