[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 17/30

[12] 发明专利申请公开说明书

[21] 申请号 200410071413.5

[43] 公开日 2005 年 4 月 27 日

[11] 公开号 CN 1609856A

[22] 申请日 2004.6.23
[21] 申请号 200410071413.5
[30] 优先权
[32] 2003.6.23 [33] US [31] 10/601,444
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 M·L·布鲁恩答基
C·A·苏弗尔

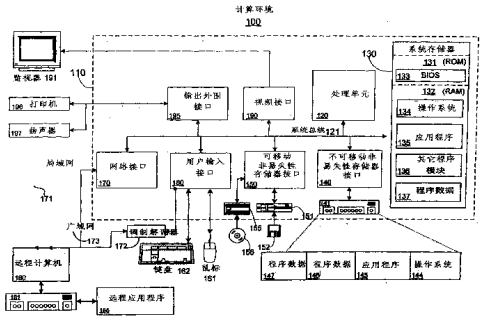
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 李家麟

权利要求书 3 页 说明书 41 页 附图 4 页

[54] 发明名称 查询中间语言的方法和系统

[57] 摘要

一种计算机系统和方法，其生成了跨越关系和非相关数据源的一个或多个 XML 语言询问的语义表示。语义中间语言表示明确地描述了一种或多种 XML 语言询问的含义。语义中间语言可以是具有节点的图形结构，所述节点描述了初始查询的操作。语义图形中赋值给节点的运算符允许初始 XML 查询的明确定义。语义中间语言可以用来遍及单个或多个数据源执行 XML 查询。一种方法，包括：接收至少一个询问，为接收到的询问内的每个操作定义至少一个节点对象，利用运算符来变换每个节点对象，以及从运算符生成语义表示。



ISSN1008-4274

1. 一种跨越关系和非相关数据源的一种或多种XML语言询问的语义表示方法，包括：

接收至少一个询问；

为至少一个接收到的询问中的每个操作定义至少一个节点对象；

利用运算符来变换至少一个节点对象中的每一个；以及

从运算符生成语义表示；

其中所述语义表示显式地描述一个或多个XML语言询问的含义。

2. 如权利要求1所述的方法，其中：所述语义表示是为目标查询引擎进行解译和执行而形成的中间语言表示。

3. 如权利要求2所述的方法，其中：所述非相关数据源包括文本文档、电子表格和非关系数据库中的一个或多个。

4. 如权利要求1所述的方法，其中所述生成步骤还包括：将接收到的询问的高级操作分解成显式部分。

5. 如权利要求4所述的方法，其中：所述显式部分是跨越多种XML语言公用的。

6. 如权利要求1所述的方法，其中所述运算符包括以下运算符中的一个或多个：特定运算符、数据源、文字、布尔运算符、序列运算符、算术运算符、串运算符、值比较运算符、节点比较运算符、元组空间、函数定义与调用、XML导航、XML组成、XML属性存取器、类型运算符、特定语言的运算符以及数据操作运算符。

7. 如权利要求1所述的方法，进一步包括：形成至少一个节点对象的图形数据结构。

8. 如权利要求1所述的方法，其中：所述至少一个接收到的询问包括XML查询语言和XML视图定义语言中的一种或多种。

9. 如权利要求1所述的方法，其中：所述至少一个接收到的询问包括XPath、XSLT、XQuery、DML、OPath和注释模式询问中的一个或多个。

10. 如权利要求1所述的方法，其中：所述语义语言表示允许XML遍及相关数据的XML视图来进行查询。

11. 一种用于表示跨越多个数据源的XML查询和XML视图中一个或多个的

语义解释器，包括：

输入端，用于接收形成询问的XML查询和XML视图中的一个或多个；

图形结构发生器，用于为询问内的每个操作定义节点对象；

翻译器，用于为每个节点对象赋值运算符，其中所述运算符将询问的操作分解成显式部分；和

输出端，用于提供显式部分以作为用于表示XML查询和XML视图中一个或多个的中间语言表示。

12. 如权利要求11所述的语义解释器，其中：所述多个数据源包括关系和非相关数据源。

13. 如权利要求12所述的语义解释器，其中：所述非相关数据源包括文本文档、电子表格和非关系数据库中的一个或多个。

14. 如权利要求11所述的语义解释器，其中所述运算符包括以下运算符中的一个或多个：特定运算符、数据源、文字、布尔运算符、序列运算符、算术运算符、串运算符、值比较运算符、节点比较运算符、元组空间、函数定义与调用、XML导航、XML组成、XML属性存取器、类型运算符、特定语言的运算符以及数据操作运算符。

15. 如权利要求11所述的语义解释器，其中：所述显式部分是跨越多种XML语言公用的。

16. 如权利要求11所述的语义解释器，其中：所述中间语言表示是为目标查询引擎进行解译和执行而形成的。

17. 一种计算机可读介质，其具有用于执行接收到的询问的中间语言表示方法的计算机可执行指令，包括：

接收形成接收到的询问的XML查询和XML视图中的一个或多个；

为接收到的询问内的每个操作定义节点对象；

利用运算符来变换每个节点，其中该运算符将接收到的询问的操作分解成显式部分；以及

生成对应于形成中间语言表示的显式部分的指令，以供遍及关系数据源和非相关数据源中的一个或多个进行后续查询。

18. 如权利要求17所述的计算机可读介质，其中所述运算符包括以下运算符中的一个或多个：特定运算符、数据源、文字、布尔运算符、序列运算符、算术运算符、串运算符、值比较运算符、节点比较运算符、元组空间、函数定义与调用、

XML导航、XML组成、XML属性存取器、类型运算符、特定语言的运算符以及数据操作运算符。

19. 如权利要求17所述的计算机可读介质，其中：所述显式部分是跨越多种XML语言公用的。

20. 如权利要求17所述的计算机可读介质，其中：所述接收到的询问包括XML查询语言和XML视图定义语言中的一种或多种。

21. 一种用于生成询问的语义表示的计算机系统，包括：

处理器，用于执行计算机指令，和

至少一个模块，包括：

输入函数，用于接收形成询问的XML查询和XML视图中的一个或多个；

图形结构发生器用于为询问内的每个操作定义节点对象；

翻译器函数，用于为每个节点对象赋值运算符，其中所述运算符将询问的操作分解成显式部分；和

输出端，用于提供显式部分以作为用于表示XML查询和XML视图的含义的中间语言表示；

其中所述至少一个模块包括一个或多个软件模块和一个或多个硬件模块中的一个或多个。

22. 如权利要求21所述的计算机系统，其中所述运算符包括以下运算符中的一个或多个：特定运算符、数据源、文字、布尔运算符、序列运算符、算术运算符、串运算符、值比较运算符、节点比较运算符、元组空间、函数定义与调用、XML导航、XML组成、XML属性存取器、类型运算符、特定语言的运算符以及数据操作运算符。

23. 如权利要求21所述的计算机系统，其中：所述显式部分是跨越多种XML语言公用的。

查询中间语言的方法和系统

发明领域

本发明总体上涉及遍及数据源进行查询的软件的领域，更具体而言，涉及利用中间语言来遍及一个或多个XML或虚拟XML数据源进行查询的软件的领域。

发明背景

可扩展标记语言（XML）是签署文件格式化标准的万维网协会（W3C），所述文件格式化标准提供了具有人类可读标记的用于标记数据的遗传语法。XML不具有固定的标记集和元素集，因此只要这类标记符合XML标准就允许用户定义它们。可以将数据作为由文本标记围绕的文本串存储在XML文档中。W3C已经以一种称为XML信息集（XML Infoset）的技术规范编纂了XML的抽象数据模型。目前的Infoset根据包含属性的节点描述了XML文档的逻辑结构。尽管XML可能易于以定义明确的格式来描述文档内容，但是仍存在不易描述的其它数据源，这是因为它们的结构与标准文本文档的结构不一致或者是由于其它的非XML可兼容特性的原因。这类数据源的一个例子可能是电子表格或关系数据库。

虚拟XML是这样一种概念，即：其跨越多种不同形式的数据访问程序设计模型而建立一致性，并允许用户按照他们考虑数据的方式而不是实际存储格式来利用他们的所述数据进行工作。遍及虚拟XML数据进行查询的概念涉及：把数据当成仿佛它就是XML一样来对待，而无需实际上曾经把它转换成XML。这一概念中的一个优点就是：将XML编码的开销保持到最少。如果虚拟XML情形有这样的优点，即它能够使用询问语言来遍及非XML数据源进行查询，就仿佛该数据源是XML一样，那么是人们所希望的。同样，在实际数据与虚拟XML表示之间的映射具有高保真度也是人们所希望的。

实现虚拟XML存在许多的内在挑战。一个问题就是效率。人们仅仅能够利用虚拟XML接口来陈列数据源，比如像XMLReader™，然后利用现有的XML查询实施方案（比如XPathNavigator™）经该接口进行查询。然而，所有的工作

都发生在XML查询引擎中，而不是由数据源本身执行的。

举例来说，考虑下列经嵌入在SQL服务器中的结构化查询语言（SQL）来进行的虚拟XML查询：

sql:database ("Northwind")/Customer[@ ID='ALKFI']/Order

在这次查询中，sql:database("Northwind")把SQL服务器的Northwind数据库陈列为虚拟XML，然后XPath/Customer[@ ID='ALKFI']/Order选择来自于消费者元素之一的所有指令元素（Order elements）。实施方案可以类似下列这样进行尝试：

SQLServerMapping map = new SQLServerMapping("Northwind");

SQDServerXmlReader Data = new SQLServerXmlReader(map);

XPathNavigator nav = new XpathNavigator(data,"/Customer[@ID='ALKFI']/Order;

在这个方法中存在至少两个缺陷。第一，全部映射都是通过XmlReader来执行的，即便只有其一部分是用于查询的。第二个缺陷就是：SQL服务器能够比XpathNavigator可以的更加有效地通过ID来选择消费者。注意：上面的例子已经XPathNavigator执行所有的工作。对于这一挑战的最佳解决方案可能就是：尽可能把更多的查询下载转移到数据源（这里，是SQL服务器数据库）中。然而，这可能会包含显著的查询分析及重写。

实现虚拟XML过程中的另一个问题就是：XML数据模型未必总是能与下层数据模型及其类型系统很好地保持谐调。一个人能够将所有类型的下层数据源映射到XML类型中，但是这一过程失掉了保真度而且也很低效。此外，一个系统中的类型可能不会明显等同于另一个系统的类型。例如，以XML表示二进制数据（比如图像）需要代价很高地将其转换成XML字符集（例如，base64-编码）。

早先试图遍及虚拟XMI进行查询，通过构建两种不同的数据结构来解决上述问题；所述两种数据结构一个用于查询一个用于映射，然后，一前一后地遍历它们以便直接遍及原始数据源产生高效查询，而无需曾经使虚拟XML视图（view）具体化。尽管这个方法最初运行良好，但是由于查询和映像语言复杂度增加而使得开发变得极其困难。查询或映射中的概念往往未直接转换成目标数据模型，并且利用复杂视图构成的复杂查询需要许多语义分析及重写。

因此，存在统一表示的需要，以经XML和非XML数据源为XML查询和视图实现虚拟XML。人们期望这样一种方法和系统，其通过将复杂映射变成构成查询的较少复杂问题并经较少复杂映射执行所述问题来实现查询。本发明针对上述需要和缺陷，并且利用这里所表述的附加优点来解决它们。

发明概述

本发明涉及一种被称为查询中间语言（QIL）的XML中间语言定义，，它表示XML语言查询或视图的含义。QIL是对XML中查询或视图的含义的抽象，并且可以用来执行查询或者可以用作到语言翻译器或编译器的输入，以便执行目标语言中的查询。QIL可以用来遍及相关和非相关数据源进行查询。

根据本发明的一些方面，QIL是通过接收XML语言询问、为XML语言询问内的每个操作定义至少一个节点对象、利用特定QIL运算符来变换每个定义好的节点对象、然后从节点图形中的运算符生成初始询问的查询中间语言表示来构建的。

通常，将QIL语言用作为计算机系统的的一部分，所述计算机系统包括：用于XML语言的前端编译器，在其中生成XML查询和视图；QIL发生器和后端引擎，所述后端引擎可以使用XML查询或视图的QIL语义表示来遍及一个或多个数据源进行搜索。所查询的数据源可以是相关数据或非相关数据。

前端XML语言的例子是：XPath、XSLT、XQuery、DML、OPath和注释概要。后端引擎类型的例子可以是：用于查询语言转换器的QIL，在该查询语言转换器中查询语言可以是任何大量的数据搜索语言，仅举几个例子来讲，比如是SQL、MSIL和XQuery。

示例性的QIL定义包括：一组基于XMLInfoset的明确定义的运算符，所述XMLInfoset可以用作为提供明确的且显式的指令的节点定义运算符，以用来表示XML查询和视图表达式的含义。举例来说，期望的运算符包括：元组、文字、布尔运算符、序列运算符、算术运算符、串运算符、值比较运算符、节点比较运算符、函数定义与调用、XML导航、XML组成、XML属性存取器、类型运算符、特定语言的运算符、数据操作、特定运算符和数据源运算符。

通过参照附图阅读下列说明性实施例的详述，本发明的附加特征和优点将变得明显。

附图简要说明

当结合附图阅读前述概述及下列优选实施例的详细说明时，将会更好地理解本发明。为了举例说明本发明，在附图中示出了本发明的示例性组成结构；然而，本发明不限于所公开的特定方法和手段。在附图中：

图1是示出示例性计算环境的框图，在该计算环境中可以实现本发明的多个方面；

图2是描绘具体化本发明示例性方面的示例性功能结构的框图；

图3是描绘依照本发明实施例的中间语言生成的一些示例性元素的框图；和

图4是描绘依照本发明实施例的中间语言生成的示例性过程的流程图。

说明性实施例详述

概观

本发明的XML中间语言显式地表示查询的含义或语义。将XML中间语言称为查询中间语言（QIL）。

XML中间语言QIL提供了（1）XML查询和XML视图的统一表示，由此大大地简化了查询/视图组成问题，并且（2）把所有的视图都当作“虚拟XML”对待，大大地简化了系统接口。不是具有一个API接口为每种可能的语言和数据模型服务，而是所有API都能够共享一个公用数据模型；XML中间语言QIL的运算符。

QIL也针对解决众所周知的编译程序问题。通常，就语言来讲，编译器需要在N个后端目标机上实现M种前端语言。如果实现每个成对组合，那么就需要M次N编译器实施方案来覆盖所需的组合。然而，如果通过引入公用的中间表示来使上述两者脱开关系，那么编译器的复杂度缩减为仅仅M加N。

查询中间语言提供了XML查询/视图语言与目标数据模型（.NET中的真实XML数据，在SQL数据或文件系统上的虚拟XML视图）之间的抽象，来适应这种编译器复杂度缩减。

查询中间语言可能比任何XML询问语言本身更加有表达力的。这允许新类型的分析和最佳化，如果原始语言仅仅用于查询和视图的话，则所述新类型的分析和最佳化是不可能的。

示例性的计算装置

图1及以下论述是用来提供对适合的计算环境的简要概述，在该计算环境中可以实现本发明。然而，应该理解的是，如上所述，打算结合本发明来使用各

类手提式、便携式及其它计算装置和计算对象。因此，尽管在下面描述了通用计算机，但是这只是一个例子，可以利用其它计算装置来实现本发明，比如具有网络/总线互用性和交互作用的客户端。因此，本发明可以在网络化寄宿服务的环境中加以实现，在所述服务中，牵涉微乎其微的或极少的客户端资源，例如，网络环境，在该网络环境中客户端装置仅仅作为对网络/总线接口进行服务，比如像置于设备中的对象，或者其它计算装置和对象也可以。实质上，可以存储数据的任何地方或者可以从中检索数据的任何地方，都是根据本发明的操作而期望的或适合的环境。

虽然是不需要的，但是本发明可以通过操作系统来实现以供为装置或对象服务的开发人员使用，和/或包括在根据本发明操作的应用软件之内。可以在普通的计算机可执行指令上下文中描述软件，比如由一个或多个计算机执行的程序模块，比如客户端工作站、服务器或者其它装置。一般来讲，程序模块包括执行特定任务或实现特定抽象数据类型的例行程序、程序、对象、组件、数据结构等等。典型地，可以将程序模块的功能组合起来或者根据各种实施例中的需要将其发布。此外，本技术领域的技术人员将会认识到：本发明可以利用其它的计算机配置来实施。可能适用于同本发明一起使用的其它众所周知的计算系统、环境和/或结构包括但不限于：个人电脑（PC）、自动出纳机、服务器计算机、手提式的或膝上型的装置、多处理器系统、基于微处理器的系统、可编程用户电子设备、网络PC、设备、照明器、环境控制元件、小型计算机、大型计算机等等。本发明还可以在分布计算环境中实施，在所述分布计算环境中，任务是由经通信网络/总线或其它数据传输介质链接的远程处理装置执行的。在分布计算环境中，程序模块可以位于包含记忆体存储器装置的本地计算机存储介质和远程计算机存储介质两者当中，并且客户端节点又可以充当服务器节点。

因此，图1举例说明了适合的计算系统环境100的例子，在该计算系统环境中可以实现本发明，尽管正如上面阐明的那样，所述计算系统环境100仅仅是适合的计算环境的一个例子，并不意在暗示对本发明使用或功能的范围的任何限制。也不应该将该计算环境100解释成具有与示例性操作环境100中举例说明的任意一个元件或任意元件组合有关的任意相关性或需求。

参照图1，用于实现本发明的示例性系统包括以计算机系统110形式出现的通用计算装置。计算机系统110的组件可以包括但不限于：处理单元120、系统

存储器130以及将包含系统存储器的各种系统组件耦合到处理单元120的系统总线121。系统总线121可以是任何几种类型的总线结构，包括使用各种总线体系结构中任意一种的存储器总线或存储器控制器、外围总线和局部总线。作为举例而非限制，这类体系结构包括：工业标准结构（ISA）总线、微通道结构（MCA）总线、增强型ISA（EISA）总线、视频电子标准协会（VESA）局部总线、以及外设部件互连（PCI）总线（亦称附加板总线）。

计算机可读介质可以是计算机系统110能访问的任何可用介质，并且既包括易失性介质又包括非易失性介质，还包括可拆卸的和不可拆卸的介质。作为举例而非限制来讲，计算机可读介质可以包括计算机存储器介质和通信介质。计算机存储器介质包括以用于信息存储的任何方法或技术来实现的易失性和非易失性、可拆卸的和不可拆卸的介质，所述信息比如像计算机可读指令、数据结构、程序模块或其它数据。计算机存储器介质包括但不限于：随机存取存储器（RAM）、只读存储器（ROM）、电可擦可编程只读存储器（EEPROM）、闪速存储器或其它存储技术、只读光盘（CDROM）、可重写光盘（CDRW）、数字通用盘（DVD）或其他光盘存储器、磁盒、磁带、磁盘存储器或其他磁存储器装置、或可用于存储所需信息和可以由计算机系统110访问的任何其他介质。通信介质典型地包含计算机可读指令、数据结构、程序模块或已调制数据信号（例如载波）中的其他数据、或其他传送机构并且包括任何信息传递介质。术语"已调制数据信号"指具有一个或多个其特征集或在信号中以编码信息方式变化的信号。作为举例但不限定，通信介质包括有线介质例如有线网络或直接有线连接和无线介质例如声、RF、红外及其他无线介质。任何上述的组合也包括在计算机可读介质范围内。

系统存储器130包括计算机存储介质，其为易失性和/或非易性失存储器形式，例如只读存储器（ROM）131和随机存取存储器（RAM）132。包含例如在启动期间，有助于在计算机系统110内部元件之间传递信息的基本例程基本输入／输出系统133（BIOS）典型地保存在ROM 131中。RAM 132典型地包含可由处理单元120立即访问和/或立刻操作的数据和/或程序模块。举例来说，而不限定，图1示出了操作系统134、应用程序135、其它程序模块136和程序数据137。

该计算机系统110也可以包括其他可拆卸/不可拆卸、易失/非易失计算机存储介质。仅仅举例来说，图1示出了读写到不可拆卸、非易失性磁介质的硬盘驱动

器141、读写到可拆卸、非易失性磁盘152的磁盘驱动器151和读写到可拆卸、非易失性光盘156的光盘驱动器155，例如CD ROM、CDRW、DVD或其他光介质。可用于该示范性操作环境的其他可拆卸/不可拆卸、易失/非易失计算机存储介质包括，但不局限于， 磁带盒、闪存卡、数字通用盘、数字视频磁带、固态RAM、固态ROM等等。该硬盘驱动器141典型地经由不可拆卸存储器接口例如接口140连接到系统总线121，并且磁盘驱动器151和光盘驱动器155典型地经由可拆卸存储器接口例如接口150连接到系统总线121。

上述所描述的和在图 1 中示出的驱动器和他们相联系的计算机存储介质为计算机系统110提供计算机可读指令、数据结构、程序模块、及其他数据的存储。在图1中，例如，将硬盘驱动器141举例说明成存储操作系统144、应用程序145、其它程序模块146和程序数据147。注意：这些组件要么与操作系统134、应用程序135、其它程序模块136和程序数据137相同，要么与它们不同。在这里，给操作系统144、应用程序145、其它程序模块146和程序数据147指定了不同的数字标记，以便至少说明它们是不同的副本。用户可以通过诸如键盘162和指点器161之类的输入装置将命令和信息输入到计算机系统110中，所述指点器通常称为鼠标器、轨迹球或触摸板。其它的输入装置（未示出）可能包括：麦克风、操纵杆、游戏板、圆盘式卫星电视天线、扫描仪等等。这些及其它输入装置往往通过耦合于系统总线121的用户输入接口160连接于处理单元120，但是可以通过其它接口和总线结构来连接，比如并行端口、游戏端口或通用串行总线（USB）。监控器191或其它类型的显示装置也通过诸如视频接口190之类的接口连接于系统总线121，所述视频接口可能又与视频存储器（未示出）相通信。除了监控器191之外，计算机系统还可以包括其它的外围输出装置，比如像扬声器197和打印机196，可以通过输出外围接口195加以连接。

计算机系统110可以使用到一个或多个远程计算机（比如，远程计算机180）的逻辑连接来在网络或分布式环境中工作。远程计算机180可以是：个人电脑、服务器、路由器、网络化PC、对等装置或其它公用网络节点，并且典型地包括上面针对计算机系统110所描述的许多或所有元件，不过在图1中仅仅已经举例说明了记忆体存储器装置181。图1中描绘的逻辑连接包括：局域网（LAN）171和广域网（WAN）173，但是还可以包括其它的网络/总线。在家庭、办公室、企业用计算机网络、内联网和因特网中这类网络环境是很常见的。

当用在LAN网络环境中时，计算机系统110通过网络接口或适配器170连接于LAN 171。当用在WAN网络环境中时，计算机系统110典型地包括：调制解调器172或用于通过诸如因特网之类的WAN 173建立通信的其它装置。调制解调器172可以是内部的或外部的，它可以通过用户输入接口160或其它适当的机构连接于系统总线121。在网络环境中，针对计算机系统110所描述的程序模块或其部分可以存储在远程记忆体存储装置中。作为举例而非限制，图1把远程应用程序185举例说明成驻留在存储装置181上。将会认识到的是，所示出的网络连接是示例性的，也可以使用用于在计算机之间建立通信链路的其它装置。

根据个人计算与因特网的交汇，各种分布式计算框架已经开发出来并正在继续进行开发。为像个人用户和商业用户这样的用户提供了一种用于应用和计算装置可无缝互操作且启用Web的接口，使得面向 Web浏览器或网络的计算活动增加。

例如，微软公司的.NET平台包括：服务器、构件块服务，比如基于Web的数据储存软件和可下载装置软件。尽管在这里结合驻留在计算装置上的软件描述了示例性的实施例，但是本发明的一个或多个部分同样也可以通过操作系统、应用编程接口（API）或任何协处理器之间的“中间商”对象、显示装置和请求对象来实现，以便可以通过在所有.NET语言和服务中得到支持或者经由所有.NET的语言和服务被访问，来执行根据本发明的操作，也可以在其它的分布式计算框架中执行本发明。

示例性的实施例

图2描绘了使本发明具体化的系统体系结构的框图。该体系结构描绘了用于为基于多个数据源的有效XML查询进行接收、编译、解释、正规化和存取数据所需的基本块。可以实现与XML或XML相关标准一致的多个前端语言编译器210。这些前端编译器210产生XML查询、XML视图、或其它有关的XML语言询问，以便产生编译后的XML代码215。

中间语言发生器220产生所输入的编译后的XML代码的中间语言表示225。体系结构200中的功能230可以对中间语言表示进行正规化，以便跨越多个数据源提供实用性（utility）。正规化器230还可以用来优化中间语言以便产生高效的中间语言表示235。应当注意的是：对于图2的体系结构200而言，并非绝对需要中间语言正规化器。

可以有多个后端查询引擎240来支持数据源260的多样性，所述数据源可以经由中间代码表示加以访问。可以对每个后端引擎进行构建，以便它有效地处理与每个支持数据源内的数据模型相一致的数据源。后端引擎240生成对应于中间语言中的XML表示的目标数据源查询。后端查询引擎240将适当的机器代码245传递给查询执行引擎250，所述查询执行引擎能够执行指定的查询。查询执行引擎250通常访问（步骤265）一个或多个数据源260，以便该引擎可以输出（步骤255）查询结果。

可以在一个或多个不同类型的不同数据源上使用一个或多个初始的查询，来产生期望的查询结果。

图3是依照本发明的示例性的中间语言发生器（例如，图2中的元件）的功能框图。图3描绘了可能存在馈送中间语言（IL）发生器220的多个（达到M个）XML语言编译器源210a-d。这些XML语言编译器总体上涉及：扫描、解析、语义分析和转换成某种类型的XML语言。这种语言的典型例子就是：XPath、XML Stylesheet语言（XSL）、XSLT、XML查询语言（XQuery）以及微软公司查询语言（比如像XQuery DML™、OPath™和WPath™）的W3C标准。还可以存在XML视图编译器，并且它包括XQuery™的W3C XML标准视图以及微软公司注释模式，比如SQLXML™。本发明的中间语言表示技术也可兼容于用于查询和视图的其它XML语言。一般来讲，所有XML可兼容的查询和视图都可以称作为XML语言询问。

图3中描绘的中间语言发生器220分成了两个主要部分：一个是操作标识符310，而另一个是节点运算符处理器320。它们共同描绘了XML中间语言产生器220的语义解释器功能。操作标识符310执行标识来自于编译器210a-d的初始XML查询或视图内的操作的任务。将标识后的操作映射到由操作标识符310构建的图形结构中的节点上。节点运算符处理器320把XML IL特定的操作者映射到已构建的图形节点中，以便结束查询的图形表示。然后，节点运算符处理器320将运算符解码成初始查询或视图的XML中间语言表示。将这一输出传递到一个或多个后端引擎240a-d上。

可以将达到N个之多的后端引擎240a-d添加到体系结构中，以便利用中间语言表示来跨越多个数据源实施查询。查询语言和视图语言与多个目标模型之间的中间语言抽象，允许在关系（例如，SQL）数据和非关系（例如，平面文件

）数据上使用真实的XML数据（比如来自于微软公司.NETTM）和虚拟的XML数据。

图4表示本发明实施例的流程图，其表示示例性的中间语言产生器的语义解释器功能。在步骤410，由中间语言发生器来接收XML查询或视图。在步骤420，将接收到的查询传递（步骤415）给标识函数，所述标识函数首先标识并生成对应于接收到的XML查询或视图当中的每个XML操作的图形节点对象。在步骤430，一旦标识了节点对象，就将它们传递给变换函数（translation function）。翻译函数将已标识的对象分配到中间语言运算符中。典型地，每个节点都有一个中间语言运算符。现在，在步骤440，将变换后的节点对象（现在是中间语言运算符）传递（步骤435）于中间语言形成引擎。这将中间语言运算符映射到显式的XML部分中，所述XML部分形成了构建后的中间语言225。继而，例如，如图2中编号230所示，可以将构建后的中间语言输出到中间语言正规化器中，或者例如，如图3中编号240a-d所示，可以将其输出到一个或多个后端引擎中。

查询中间语言属性和性能

XML中间语言是XML查询或视图的表示。照此，可以将查询中间语言（QIL）术语化，这是因为它是XML查询的含义的显式表示。可以把QIL看作跨越XML查询和视图语言编译器的公用语义表示。QIL类似于普通的抽象语法树（AST），不过区别之处在于：QIL不具有query的语法而是具有查询的语义或含义。另一个区别就是：QIL是图形结构而不是像AST这样的树型结构。

QIL在各种不同的目标数据源（比如像，关系和非相关数据）上启用了多个不同的XML查询语言和视图定义语言（例如，诸如像XPath、XSLT、XQuery、DML和注释模式）的抽象。照此，QIL启用了公用组成来支持所有兼容的XML语言。每个操作都既是显式的又是明确的，这造成QIL冗长，不过将参与生成QIL的前端编译器从使用QIL的后端引擎上断开了联系。

查询中间语言还提供了一种将任意数据源作为虚拟XML建模的方法。可以直接通过XML数据模型数据源来陈列数据源，或者QIL用户可以首先分析查询所生成的中间语言并确定怎样最好地发布该查询。例如，一些查询结果可以表明：可以容易地把一些运算符牵引到适合于那个数据源的查询引擎中，并由此以逻辑方式来划分查询。QIL允许将查询任务牵引到能最有效执行该查询任务的引擎中。

查询中间语言利用表示初始查询的图形结构的节点来使用其自己的运算符。这些运算符允许新种类的查询优化、查询重写以及早前不能用相应的数据源实现的其它分析。大部分的QIL运算符跨越多种语言都是公用的，并且其中一些是专门对应特殊XML查询语言的。尽管每种XML语言都具有其自己隐含的任务和专门的情况，但是查询中间语言对这些含义进行显式且明确地编码。

查询中间语言是节点对象的图形。这种中间语言包含最多一个有根图形，该有根图形是主要的查询，还包含零个或多个定义XML中间语言函数的附加图形。存在许多不同的方式来表示存储器中的图形；使用节点对象的网络就是一个实施例。QIL使用了跨越许多不同节点类型共享的类，而不是为每个节点类型定义一个类。

即使初始查询语言不是强类型的，QIL也是强类型的。QIL可以支持遍及XML数据的XML查询、遍及相关数据的XML视图的XML查询、遍及相关数据的XSLT以及遍及非相关数据的XML查询，所述非相关数据比如像文本文件、电子表格和非关系数据库。QIL也可以用来跨越多个数据源执行不同种类的、已发布的查询，并且还可以将XML和虚拟XML数据源融合起来。

QIL表示查询的含义，且因而是一种逻辑查询设计而非物理查询设计。QIL存在多种执行策略。可以直接地以左侧深度优先的方式、在全部QIL图形上反复执行语言。或者，可以将查询中间语言变换成另一种语言来执行，比如像SQL。另一个可选方案就是：忽略QIL来作出分析，提取由一个引擎执行的某些部分，根据需要修改图形，并且通过另一个现有引擎来保持执行的平衡性。

QIL是强类型的。QIL具有详细的类型系统，并且这个类型系统反映在如下表当中，所述表指的是示例性的QIL运算符，其中一些QIL运算符在这里作了描述。除非另作说明，否则该说明表示了表达式的固定类型，而不表示它要评估的值的自动类型。

除了像DocumentOrder和DML这样的少数几个已选定的“运算符”之外，每个QilNode表达式都是一些基数的序列。除非另有陈述，否则这个基数总是以下基数的其中之一：None（未知）、Zero（空）、One（单元素集合）、More（2+）或这些的组合。

所述序列由一些项组成，由于允许不同种类的序列，因此这些项要么可以是原子值或节点要么两者都是。进一步把原子值分为一组类型，比如像Int32、

String和DateTime。对于大部分，QIL原子类型是XQuery原子类型的子集，并且反映了实际的实施方案选择。当表达式其不是原子的（即，节点或选定的运算符类型，像DML），那么就具有原子类型None。

示例性的QIL运算符

优选地，QIL的每个运算符具有唯一相应的NodeType和表示它的类。实质上，运算符存在于QIL图形结构的节点中。因此，分配给节点的运算符具有节点类型，并且该运算符变为节点类型的函数。每个类都有许多节点类型，不过每个节点类型仅仅有一个类。期望每个运算符都具有单一的、显式的、明确的含义。总体上讲，这种含义与其操作数的类型或使用它的环境无关。根据这里所述的QIL类型系统，每个运算符都是强类型的。

这里所提供的是对应每个运算符节点类型的一些示例性属性的表。除非另有说明，否则没有操作数可以是null。QIL运算符可以分类成下列示例性的组：特殊算符（错误、无操作、未知）、数据源、文字（标量常数）、布尔运算符、序列算符、算术运算符、串算符、值比较算符、节点比较算符、元组空间（包括排序）、函数定义与调用、XML导航、XML组成、XML属性存取器、类型运算符、用于XPath、XSLT、XQuery和数据操作语言（DML）的特定语言运算符。

元组空间算符

QIL中的示例性算符是元组（tuple）。可以频繁地在QIL中使用元组算符，并且可以以嵌套方式来使用它。该算符是由元组节点类型（类QIL元组）来表示的，并且期望该算符总是具有刚好三个子节点：一列构建元组空间的迭代器，过滤元组空间的where子句，以及产生元组空间结果的returen子句。

显现嵌套元组的一个方式是：假想一张行列表，在该表中，每个表元（cell）本身都可以包含表。一对六边骰子就是一个例子。正如在下列表1中局部描绘的那样，存在36种可能的组合。

表1：2个六边骰子的元组空间

die1	die2	结果
1	1	（1，1）
1	2	（1，2）
1	3	（1，3）
1	4	（1，4）
1	5	（1，5）
1	6	（1，6）
2	1	（2，1）
2	2	（2，2）
…	…	…

这个元组空间具有三列。开头两个分别对应于第一个和第二个骰子。第三列用来保存结果；在这种情况下，所述结果是一对值。这个元组空间具有36行，它们对应于通过翻滚两个骰子而产生的不同组合的数目。以下是通过将集合（1,2,3,4,5,6）与其自身结合而产生的元组空间，例如，XQuery：

for $die1 in (1, 2, 3, 4, 5, 6),

$die2 in (1, 2, 3, 4, 5, 6)

return ($die1, $die2)

这个XQuery计算两个集合的叉积，并为此叉积中的每个成员返回结果序列；即一对die值。该结果是列表的有序表：

((1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1,6), (2, 1), (2, 2) ....)

然而，XQuery将一列列表处理为所有它们的级联，所以这些实际上仅仅是平伸列表：

(1, 1, 1,2, 1,3, 1,4, 1, 5, 1,6,2, 1,2,2 ....)

注意：该查询还可以写成：

for $die1 in (1, 2, 3, 4, 5, 6)

for $die2 in (1, 2, 3, 4, 5, 6)

let $result: = ($die1, $die2)

return $result

在这种形式下，每个XQuery变量都刚好对应于元组空间中的一列。这同样

也是QIL的概念。QIL元组节点或算符中的每个迭代器都对应于元组空间中的一列。returen子句影响上面作为变量赋值而使用LET写入的附加列。

执行元组节点的结果是结果的有序表，并且当该结果本身是一列表时，将它们像以前那样级联在一起（平伸）。像XQuery一样，QIL不区分列表和列表的列表。

where子句提供了通过消除元组空间中的一些行来过滤该元组空间的时机。期望该where子句不会对元组空间有其它的影响。在所有上面的例子当中，where子句由单个真实节点组成，这意味着接收每一行（不会发生过滤）。然而，所述结果可能限于共计7次的六个die翻滚，如下：

```
for $die1 in ( 1, 2, 3, 4, 5, 6)
for $die2 in ( 1, 2, 3, 4, 5, 6)
let $result: = ( $die1, $die2 )
where $die1 + $die2 = 7
return $result
```

这创建了过滤后的表2中所描绘的元组空间。

表2：共计7次的骰子翻滚的元组空间

die1	die2	结果
1	6	（1，6）
2	5	（2，5）
3	4	（3，4）
4	3	（4，3）
5	2	（5，2）
6	1	（6，1）

这个XQuery例子可以利用下列查询中间语言表示来代表：

注意：这只不过是QIL数据结构的一种XML表示。

```
<Tuple>
<! - - First the iterator list - - >
<For id = "$a">
<List>
<LiteralInteger>1</LiteralInteger>
```

```
<LiteralInteger>2</LiteralInteger>
<LiteralInteger>3</Literalinteger>
<LiteralInteger>4</LiteralInteger>
<LiteralInteger>5</LiteralInteger>
<LiteralInteger>6</LiteralInteger>
</List>
</For>
<For id = "$b">
<List>
<LiteralInteger>1</LiteralInteger>
<LiteralInteger>2</LiteralInteger>
<LiteralInteger>3</LiteralInteger>
<LiteralInteger>4</LiteralInteger>
<LiteralInteger>5</LiteralInteger>
<LiteralInteger>6</LiteralInteger>
</List>
</For>
<Let id = "$c">
<List>
<For id = "$a"/>
<For id = "$b"/>
</List>
</Let>
<! - Then the where clause - - >
<Eq>
<Plus>
<For id = "$a"/>
<For id = '$b"/>
</Plus>
<LiteralInteger>7</LiteralInteger>
```

</Eq>

<! - Then the return clause - - >

<Let id = "Sc"/>

</Tuple>

将迭代器节点（为Let）引入迭代器列表中，然后稍后在查询中再次参考该迭代器节点。为了象征性地参考它们，可以给它们指定唯一的id属性，然后稍后按这些名称来参考它们。注意：这些id串实际上在图形中是不存在的，而且也与初始查询无关；只有当将图形打印成XML时才生成它们。优选地，这些参考是指向初始节点实例的存储器指针。

因此，这个QIL图形包含二十二个唯一的节点以及以粗体突出显示的五种参考（两个在$c的定义中，两个在元组的where子句中，而一个在元组的return子句中）。

作为另一个例子，考虑该XQuery：

for $i in document ('foo.xml')/x/y

return <z/>

其相当于XQuery

for $a in document ('foo.xml')

return

for $b in $i/x

return

for $c in $j/x

return <z/>

并且两者均可以表示为如下QIL表达式：

<Tuple>

<For id = "$a">

<DataSource>

<String>foo.xml</String>

< DataSource >

</For>

<True/><!--no where clause -- >

<Tuple>

<For id = "$b"/>
</Content>
<For id = "$b"/>
</Content>
</For>
</Eq>
<NameOf>
<For id = "$b"/>
</ NameOf >
<QName local - name = "x"/>
</Eq>
<Tuple>
<For id = "$c">
<Content>
<For id = "$b">
</Content>
</For>
<Eq>
<NameOf>
<For id = "$c"/>
</ NameOf >
<QName local - name + "y"/>
</Eq>
<ElementCtor>
<QName local - name = "z"/>
<List / >
</ElementCtor>
</Tuple>
</Tuple>

如果原始文档foo.xml由下列XML组成：

<x id = "x1" >
<y id ="y1"/>
<y id = "y2" />
</x>
<x id = "x2">
<y id = "y3"/>
<y id = "y4"/>
<y id = "y5"/>
</x>

那么该QIL和XQuery例子就生成表3中所描绘的嵌套元组空间。

表3：简易Xquery的元组空间

<table>
<tr><th>$a</th><th>结果</th></tr>
<tr><td>文档节点</td><td>
<table>
<tr><th>$b</th><th>结果</th></tr>
<tr><td>x1</td><td>
<table>
<tr><th>$c</th><th>结果</th></tr>
<tr><td>y1</td><td><z/></td></tr>
<tr><td>y2</td><td><z/></td></tr>
</table>
</td></tr>
<tr><td>x2</td><td>
<table>
<tr><th>$c</th><th>结果</th></tr>
<tr><td>y3</td><td><z/></td></tr>
<tr><td>y4</td><td><z/></td></tr>
<tr><td>y5</td><td><z/></td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

因为示例性的数据模型不包括列表的列表，所以包含具有N列的表的列可以用那些N列来替换；依照要求对行进行扩展和拷贝。因此，这个元组空间通过一系列无嵌套操作而等同于表4的“平伸”元组空间。

表4：简易Xquery的平伸元组空间

$a	$b	$c	结果
文档节点	x1	y1	<z/>
文档节点	x1	y2	<z/>
文档节点	x2	y3	<z/>
文档节点	x2	y4	<z/>
文档节点	x2	y5	<z/>

以期望的文档次序来在x/y上迭代，执行QIL会产生五种<z/>元素。

相对于表5，在下面描述了一组分组为元组空间运算符的运算符。元组引入变量约束（赋值或迭代），利用where子句来对其进行过滤，并返回结果表达式。当返回的结果本身是元组时，那么就进行叉积或加入结果。

存在两种类型的迭代器节点；FOR和LET，LET仅仅是变量赋值，并且具有基数1。FOR在序列上迭代，依次约束于每个成员。因此，FOR的基数为其约束的长度。元组空间的基数最多是其迭代器的基数，所述迭代器记录了其returen子句的迭代次数。对于元组空间中的每一项，评估where子句，如果为真则评估并返回returen子句。元组的结果是所有这些返回值的序列级联（可能是空序列）。当引入到元组中时，每个迭代器都处于所有的where子句和return字句的范围内。LET迭代器保持其自变量的节点特性。除了这种复杂行为之外，QIL是明晰参考的（意味着LET可以用其约束来替换，反之亦然）。

可以将POSITIONOF运算符应用于迭代器。它确定其约束内迭代器的当前位置。这个位置是基于1的索引。应用于LET迭代器的POSITIONOF总是1。应用于FOR迭代器的POSITIONOF在1与其长度之间。

表5：元组空间运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
For	QilIterator	迭代	一	-
Let	QilIterator	赋值	约束的类型	约束的类型
Tupe	QilTuple	元组空间	任意	-
PositionOf	QilUnary	迭代的位置	一	Int32

文字运算符

在下面的表6中描述了一组分组为文字运算符的运算符。文字表示常数。单个类，QILLiteral，表示所有的文字类型（除布尔型之外，该布尔型是利用QILNode本身来表示的，而Name是利用QILName来表示的）。文字值是作为对象来存储的。

表6：文字运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
True	QilNode	布尔常数（真）	一	布尔型
False	QilNode	布尔常数（假）	一	布尔型
LiteralString	QilLiteral	串常数	一	串型
LiteralInt32	QilLiteral	位整数常数	一	Int32
LiteralInt64	QilLiteral		一	Int64

表6：文字运算符（续表）

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
LiteralDouble	QilLiteral	Double constant	一	双整型
LiteralFloat	QilLiteral	Float constant	一	浮点型
LiteralDecimal	QilLiteral	Decimal constant	一	十进制型
LiteralQName	QilName	Qualified name literal	一	QName
LiteralDuration	QilLiteral	Duration constant	一	时间间隔
LiteralYearMonThDuration	QilLiteral	YearMonth Duration constant	一	年月时间间隔
LiteralDayTimeDuration	QilLiteral	dayTime Duration constant	一	日期时间时间间隔
LiteralDateTime	QilLiteral	DateTime constant	一	日期时间
LiteralDate	QilLiteral	Date constant	一	日期型
LiteralTime	QilLiteral	Time constant	一	时间型
LiteralOther	QilLiteral	Other-typed constant	一	其他

布尔运算符

在下面的表7中描述了一组分组为布尔运算符的运算符。期望QIL支持常规的双值逻辑运算符，AND、OR和NOT，以及CONDITIONAL（隐式）。优选地，通用的和存在的量化是通过IsEmpty和Tuple的组合来实现的。

QIL支持合取范式、析取范式和一类转换（switch）的便利运算符。对于AND和OR，这两种操作数都是理想的布尔型单元素。NOT运算符的操作数是理想的布尔型单元素集合。

对于运算符CONDITIONAL，期望条件表达式（QILTernary节点的左侧属性）是布尔型单元素集合，而真/假分支（分别是中心和右侧属性）可以为任何类型。else分支不是任意的。然而，任何人通常都可以使用空表来达到“dummy（假的）”分支的效果。整个表达式的类型是真/假分支类型的联合。

为了将操作数强制为布尔型，依据应用的需要，人们可以使用运算符，比如XQUERYEBV或CONVERT。

表7：布尔运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
And	QilBinary	布尔“与”	一	布尔型
Or	QilBinary	布尔“或”	一	布尔型
Not	QilUnary	布尔“非”	一	布尔型
Conditional	QilTernary	If/then/else	中心和右侧类型的联合	

序列运算符

在下面的表8中描述了一组分组为序列运算符的运算符。优选地，每个QIL表达式都是一些基数的有序序列。为了与序列一同工作，XQuery定义了许多运算符和集合器（aggregator）。 LIST运算符对其成员执行序列级联。空表有大部分的类型都设置为None；否则，列表的类型是所有其成员类型的联合。LIST未按照特性或重复值来删除重复节点。为了从序列中删除重复的节点，人们可以显式地应用有差别的运算符（DISTINCT、DISTINCTBYVALUE）中的一个，否则人们可以使用SBT运算符。

与XQuery TO运算符相似，RANGE在左侧约束和右侧约束之间构建整数的连续序列。如果右侧约束小于左侧约束的话，那么序列递减。如果它们相等，

那么序列具有单成员（那个值）。

NTIMES是二进制算符，它的左侧操作数可以是任何类型，而期望右侧操作数是非负整数单元素集合。它产生包含左侧操作数N次的序列，其中N是右侧操作数的值。注意：它不拷贝原始值。(NTIMES left right)相当于表达式 (Tuple (For (Range 1 Right)) (True) (Left)).

DISTINCT从列表中删除重复节点（按照特性）。删除节点的次序最好是定义好的实施方案。

将DISTINCTBY的值作为其在序列上应用于FOR迭代器的第一自变量，继而该值处于第二自变量的范围内以供参考。第二自变量规定了如何计算这样的值，通过该值将会删除重复的节点。

UNION、INTERSECTION和DIFFERENCE运算符利用节点特性来执行相应的集合运算符，从而删除重复的节点。它们的操作数可以是节点序列。集合的差是不对称的；对于两个序列A和B的对称差而言，人们可以获得它们成对差的UNION；(Union(Difference A B))(Difference B A)，也可以等效的，利用它们的INTERSECTION获得它们UNION的DIFFERENCE；(Difference(Union A B))(Intersection A B)。

用数字表示的集合函数（AVERAGE、SUM、MINIMUM、MAXIMUM）都具有这样的操作数，期望这些操作数是数值的序列。它们将该序列中的所有成员都提升为公用类型，然后计算适当的函数。例如，(Average (Decimal 1.0) (Int32 3))是(Decimal 2.0). 应用于空序列的所有这四个运算符将导致空序列。

对于这些运算符中的大多数来讲，整个表达式的类型要取决于其自变量。例如，具有属性序列的元素序列的联合作为其NodeKind两类的联合。所有的组序列运算符都接收任何基数的操作数。

表8：序列运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Length	QilUnary	序列长度	G	Int32
List	QilList	序列级联	-	-
Range	QilBinary	整数序列构建	-	Int32

表8：序列运算符（续表）

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Ntimes	QilBinary	序列构建	-	-
Distinct	QilUnary	重复删除（按节点特性）	任意	-
DistinctByValue	QilBinary	重复删除（按数值）	任意	-
Intersection	QilBinary	集合插入（按节点特性）	任意	无
Difference	QilBinary	集合区分（按节点特性）	任意	无
Union	QilBinary	集合联合（按节点特性）	任意	无
Average	QilUnary	算术平均值	-	-
Sum	QilUnary	算术和	-	-
Min	QilUnary	算术最小值	-	-
Max	QilUnary	算术最大值	-	-

算术运算符

在下面的表9中描述了一组分组为算术运算符的运算符。算术运算符具有这样的特性，即依其操作数的类型而变化。所有算术运算符都要求它们的操作数具有相同的（数值）类型和基数1，然后在该类型中执行运算。除法是这些规则的一个例外。DIVIDE总是将整数操作数提升为双整型。INTEGERDIVIDE执行整数除法，因此其操作数是整数。

算术例外（比如，溢出和回流）都是定义好的实施方案。对于浮点类型而言，NaN是无信号的，并且允许无穷大和负零。

不为非数值类型或非单元素集合定义算术运算符。为了在串上进行运算，人们可以使用串运算符（比如像STRCONCAT）或者把它们转换成数值类型。

为了在数列上执行运算，人们可以使用TUPLE来贯穿序列进行迭代，并且执行成员法则操作，否则考虑使用一些序列集合器（比如SUM）。

表9：算术运算符。

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Nagate	QilUnary	算术求反	—	-
Plus	QilBinary	加法	—	-
Minus	QilBinary	减法	—	-
Multiply	QilBinary	乘法	—	-
Divide	QilBinary	除法	—	-
Modulo	QilBinary	求模	—	-
IntegerDivide	QilBinary	整数除法	—	-

串运算符

在下面的表10中描述了一组分组为串运算符的运算符。串运算符提供了公用的串功能，比如级联和子串匹配。

STRLENGTH计算串的长度。期望其自变量是单元素集合串。STRCONCAT是像LIST一样的变量-自变量运算符。它将其自变量级联到单个串中。每个自变量都可以是一列串或空表。它将空表处理成空串。STRCONCAT接收任意的串属性、分隔符，这对于空串而言是缺省的。它在每一对串成员之间插入分隔符值。注意：尽管STRCONCAT（"","）产生了分隔符，但STRCONCAT（（），（））仍旧还是空串。

STRSUBSTRING节点操作符计算串的子串。第一自变量必须是单元素集合串，而期望第二和第三自变量是单元素集合整数。它获得0或第二自变量（称为i0）的最大值，和0或第三自变量的最大值（称为i1），并且计算由i0和i0+i1之间的字符组成的子串。

STRCONTAINS把两个单元素集合串作为自变量，并且如果第一个方案包含第二自变量的话则就返回真；否则返回假。每个串都包含空串。STRENDSWITH和STRBEGINSW1TH都具有相同的功能，不过强制匹配以便分别出现在串的末端或起始处。

表10：串运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
StrLength	QilUnary	串长度	一	Int32
StrConcat	QilStrConcat	串级联	一	串
StrSubstring	QilTernary	子串	一	串
StrContains	QilBinary	串保存	一	布尔
StrBeginsWith	QilBinary	串起始于子串	一	布尔
StrEndsWith	QilBinary	串结束于子串	一	布尔

值比较运算符

在下面的表11中描述了一组分组为值比较运算符的运算符。QIL为比较值定义了几种运算符。这些相当于同名的XQuery（EQ、NE、LT、LE、GT和GE）的值比较运算符。在下文描述节点比较运算符。

期望每个操作数都是单元素集合，与期望这些操作数都具有相同的类型。根据需要，人们可以使用TREAT、CONVERT或其它类型的运算符来提升或转换操作数。

字符串比较通常将考虑到对照（collations）。能够以许多不同方式（存在的、通用的、最重要的、成员法则等等）来比较序列。

表11：值比较运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Ne	QilBinary	不相等	一	布尔
Eq	QilBinary	相等	一	串
Gt	QilBinary	大于	一	布尔
Lt	QilBinary	小于	一	布尔
Ge	QilBinary	大于等于	一	布尔
Le	QilBinary	小于等于	一	布尔

节点比较运算符

在下面的表12中描述了一组分组为节点比较运算符的运算符。QIL按特性或文档次序为比较节点定义了几种运算符。这些相当于XQuery的节点比较运算符（IS、ISNOT、BEFORE（<<），和AFIER（>>））。

如果按照文档次序、左侧操作数出现在右侧操作数的前面（后面），则BEFORE（AFTER）就返回真。对于来自于不同文档的节点（或根本不是文档中的节点），定义次序，并且该次序在执行查询过程内是稳定的，或者也可以使用定义好的实施方案。

期望每个操作数都是单元素集合节点。如果两个操作数是相同节点（按照特性），则节点类型IS就返回真；否则为假。ISNOT返回相反的结果。从BEFORE/AFTER的定义来看，其遵循这样的规律，即：如果且只有当BEFORE和AFTER两者都为其返回假时，则两个节点都具有相同的特性。因此，(Is Not $x $y)等于(Or (Before $x $y) (After $x $y)). 可以以许多不同的方式来比较序列，比如存在的、通用的、最重要的、成员法则等等。

表12：节点比较运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Is	QilBinary	相同特性	一	布尔
IsNot	QilBinary	不同特性	一	串
Before	QilBinary	出现在前面	一	布尔
After	QilBinary	出现在后面	一	布尔

排序与对照运算符

在下面的表13中描述了一组分组为排序与对照运算符的运算符。QIL定义了单个SORT节点或运算符，它们获得两个自变量。第一自变量在将要排序的集合（collection）上引入了FOR迭代器。这个迭代器处于第二自变量的范围内，该第二自变量表示排序键（sort key）。

每个排序键都是ASCENDING或者是DESCENDING。像DOCUMENTORDER一样键表达式可以是常量或更复杂的表达式。

DOCORDERDISTINCT是按照文档次序来对序列进行排序并且按照特性来删除重复节点的便利运算符。(DocOrderDistinct $x)等于(Distinct (Sort X (Ascending DocOrder)))。它的操作数必须是可能为空的节点序列。

表13：排序与对照运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Sort	QilBinary	排序	-	-
Ascending	QilNode	升序排序键	无	无
Descending	QilNode	降序排序键	无	无
DocumentOrder	QilNode	表示文档的排序键	无	无
DocOrderDistinct	QilUnary	按文档排序并且删除重复文档	-	无

函数定义与调用运算符

在下面的表14中描述了一组分组为函数定义与调用运算符的运算符。QIL既能够表示函数定义也能够表示函数调用。调用出现了几种类型；利用QIL来定义本身的函数调用，内嵌于运行时函数的调用，以及任意方法的早先约束和最近约束的调用（例如，XSLT扩展对象）。

自变量最好是定位的。如果人们需要通过QIL将换名调用语义进行管道传输，那么人们可能会需要将所有名称映射到位置（正如XSLT编译器做的那样），或者需要构建这样一种结构，即利用值来给名称配对（比如，对于过程所存储的SQLXML，ms:variables()可能会处理得好），并且在QIL中对名称匹配算法进行编码。

调用利用一系列LET迭代器来传递自变量。在FUNCTION定义中，这些迭代器约束于充当符号参考的空行（null），在运行时从外部提供所述符号参考的值。

表14：函数定义与调用运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Function	QilFunction	函数定义	-	-
Invoke	QilInvoke	调用QIL函数	-	-
Invoke Early Bound	QilDelegate	利用 MethodInfo（早期约束的）来调用CLR方法	-	-
Invoke Late Bound	QilDelegate	利用名字/答名（后期约束的）来调用CLR方法	-	-
Invoke Builtin	QILInvoke Builtin	调用内嵌函数	-	-

XML导航运算符

在下面的表15中描述了一组分组为XML导航运算符的运算符。在QIL中，在句法上保持了路径风格（Path-style）的导航；它拆分为元组空间和导航运算符。

一般而言，QIL中的导航是利用CONTENT节点操作符来进行的。期望这个运算符选择节点内的所有内容（例如，所有属性、域名空间、文本、处理指令、注释、以及元素内部的元素节点）。接着，可以（例如，利用TUPLE与ISTYPE）将内容序列过滤，以便有选择地仅仅保持某些节点种类。

为了模式识别中的便利性，提供了ATTRIBUTE运算符；它按名称来选择匹配0或1属性节点的属性节点（右侧操作数）。为图形中的公用方向提供了其它导航节点；这些节点大都取自于XPath而且具有相同的含义。一个节点一次应用每个运算符；以便施加节点的整个序列，人们可以使用TUPLE在节点序列上进行迭代。一个例外是DEREF，它的自变量必须是单个ID值。它计算id()查表的结果。

表15：XML导航运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Content	QilUnary	选择节点内容	任意	无
Attribute	QilBinary	按名称选择属性	零或一	无
Parent	QilUnary	parent::node()	零或一	无
Self	QilUnary	self::node()	一	无
Root	QilUnary	选择根	零或一	无
XmlContext	QilNode	current()	-	-
Descendant OrSelf	QilUnary	descendant::node()	任意	无
Deref	QilUnary	Id()	零或一	无

XML组成运算符

在下面的表16中描述了一组分组为排序和核对运算符的运算符。QIL定义运算符来构建每个XML节点类型。这些构建器全部都遵循普通的XML数据模型约束。例如，没有节点能够包含文档节点，并且合并连续文本节点。

优选的是，每个已构建的节点获得一个唯一的特性。就像为XQuery定义特性那样，为QIL定义这个特性：跨越像LET这样的操作保持它，并且应当由运算符来考虑侧面影响，特别是DML。

每个已构建的节点获得相对于每隔一个节点的次序。即使它应用到文档以外的节点上以及应用于来自于不同文档的节点上，也被称为为“文档次序”。完全像为XQuery定义词序那样，为QIL定义此次序。

ELEMENTCTOR运算符

ELEMENTCTOR是利用类QilBinary来实现的。左侧操作数是名称，右侧操作数是内容。如果内容中的节点有文档节点的话，那么可能会是错误。该内容中的所有节点都被隐式地拷贝。

可以不把ATTRIBUTE和NAMESPACE节点构建在元素中的其它非属性/域名空间内容后面。这种情况下的行为是定义好的实施方案。如果元素名称处于域名空间中且该域名空间不在范围内，那么一个实施方案就可以发出人工合成的域名空间声明和/或前缀。对于承兑（honor）名称的前缀值而言，是不需要实施方案的。它仅仅是系列化提示。

ATTRIBUTECTOR运算符

ATTRIBUTECTOR是利用QilBinary类来实现的。左侧操作数是名称，右侧操作数是内容。可以利用诸如XQUERYATTRIBUTECONTENT之类的运算符来将右侧操作数强制为期望的表达式。如果属性名称处于域名空间中且域名空间不在范围内，那么一个实施方案可以发出人工合成的域名空间声明和/或前缀。对于承兑名称的前缀值而言，不需要实施方案；它仅仅是系列化提示。为了构建类型化属性，人们可以利用XML模式类型信息来构建它，或者可以应用诸如XQUERYVALIDATE之类的类型运算符。

TEXTCTOR和CDATACTOR运算符

TEXTCTOR是利用QilUnary类来实现的。子操作数是文本内容，并且必须是串类型的单元素集合。CDATACTOR是表明任意系列化提示的TEXTCTOR（即，使用CDATA部分）。实施方案可以忽略这个提示。

COMMENTCTOR运算符

COMMENTCTOR是利用QilUnary类来实现的。CLASS操作数是文本内容，并且必须是串类型的单元素集合。

PICTOR运算符

PICTOR是利用QilBinary类来实现的。左侧操作数是目标NCNAME，并且可以为空。否则，它必须是QName类型的单元素集合。右侧操作数是保持的内容，并且必须是串类型的单元素集合。

NAMESPACEDECL运算符

NamespaceDecl是利用QilBinary类实现的。它构建了域名空间声明。左侧操作数是前缀，并且期望它是QName类型的单元素集合。右侧操作数是域名空间，并且期望它是串类型的单元素集合。如果左侧操作数是空串的话，那么NAMESPACECTOR相当于缺省的域名空间声明。

如果出现矛盾的域名空间声明（例如，在元素的内容中对于同一个前缀有两个NAMESPACECTORS，或者与已经发出的加前缀的名称相冲突的NAMESPACECTOR），那么该行为就是定义好的实施方案。

RTFCTOR运算符

RTFCTOR是利用QilUnary类来实现的，并且期望它的孩子都是节点序列或空序列。它构建结果—树—碎片（来自于XSLT）。事实上，RTFCTOR是到导航的一个阻碍，它阻止了组合的发生。在许多方面，比起节点，它工作起来更像是一个原子值。

表16：XML组成运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
ElementCtor	QilBinary	构建元素	一	无
AttributeCtor	QilBinary	构建属性	一	无
CommentCtor	QilUnary	构建注释	一	无
PICtor	QilBinary	构建处理指令	一	无
TextCtor	QilUnary	构建文本节点	一	无
CdataCtor	QilUnary	构建文本节点	一	无
DocumentCtor	QilUnary	构建文档节点	一	无
NamespaceDecl	QilBinary	构建域名空间声明	一	无
RtfCtor	QilUnary	构建结果树碎片	一	无

节点属性运算符

在下面的表17中描述了一组分组为节点属性运算符的运算符。QIL支持用于访问某些节点属性（比如像，名称）的运算符。

NAMEOF运算符计算节点的QName。如果该节点没有名称，则它就返回空序列。LOCALNAMEOF、NAMESPACEOF和PREFIXOF运算符计算节点名称的对应部分。如果所述节点没有名称或遗漏了这些部分的一些，那么它们就返回空串。所有这些运算符都需要它们的操作数是单元素集合节点。

表17：节点属性运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
NameOf	QilUnary	计算节点的Qname	零或一	QName
LocalNameOf	QilUnary	计算节点的局部名称	一	串
NamespaceOf	QilUnary	计算节点的域名空间	一	串
PrefixOf	QilUnary	计算节点的前缀	一	串

拷贝与投影运算符

在下面的表18中描述了一组分组为拷贝与投影运算符的运算符。QIL支持几类拷贝与投影运算符。拷贝是期望其对原子值没有影响的操作。对节点的影响实质上产生了初始节点的准确副本，不过具有新的节点特性。投影“保持”了节点特性，但是有选择地删节节点来保持其仅仅一些内容。

DEEPCOPY以节点为首构建整个子图的拷贝。除了元素和文档节点之外，对于所有的节点种类而言，SHALLOWCOPY与DEEPCOPY是相同的，对于文档而言，它拷贝文档节点但是不拷贝任何其内容，对于元素而言，它拷贝元素节点（即，它的名称）但是不拷贝任何其内容（即，无属性）。

PROJECTINCLUDE和PROJECTEXCLUDE两者都是二进制算符。左侧操作数必须是遍及一序列节点的FOR迭代器。右侧操作数必须是可能参考那个迭代器的布尔型单元素集合。这些运算符都在由左侧操作数描述的树形结构（forest）上进行递归迭代、并且分别包含或排除所有节点以及所有它们的祖先来执行投影，其中对于所述祖先将右侧操作数评估成真。

表18：拷贝与投影运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
DeepCopy	QilUnary	深度拷贝	-	无
ShallowCopy	QilUnary	Dhallow拷贝	-	无
ProjectInclude	QilBinary	包含性投影	任意	无
ProjectExclude	QilBinary	排他性投影	任意	无

类型运算符

在下面的表19中描述了一组分组为类型运算符的运算符。QIL定义了许多用于动态地与QIL类型系统一起工作的运算符。

表19：类型运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Atomize	QilUnary	执行原子化	零或一	-
TreatAs	QilUnary Type	当作类型	-	-
CanTreatAs	QilUnary Type	是否当作成功	一	布尔型
Convert	QilUnary Type	转换成类型	-	-
IsConvertible QilUnaryType	QilUnary Type	是否转换成功	一	布尔型
IsType	QilUnary Type	类型是否匹配	一	布尔型
IsEmpty	QilUnary	如果自变量为空（基数为0）则为真	一	布尔型

ATOMIZE运算符

ATOMIZE通过执行XQuery标准中所描述的XQuery原子化规则，来有效地检索XML节点的类型化值。具体地说，ATOMIZE获得任何基数的序列，并且

返回由原始序列成员的原子值组成的新序列，对于原子值来说，该原始序列是无操作；对于节点来说，该原始序列就像是到类型化值存取器的调用。例如，(XQueryAtomize (List (1 "b" 3))导致列表无变化，而(XQueryAtomize(List $x $y))，其中$x是具有类型化值（1,2）的节点而$y是具有类型化值（"3",4）的节点，将会导致组合的序列：（1,2,"3",4）。

TREATAS运算符

因为QIL是强类型且是静态类型的，所以可能会出现这样的情况，即：比起表达式具有静态类型来，期望的是更强类型或更弱类型。例如，可能会想要双整型，其中表达式是整数。对于这种情况（例如，上抛或下落）来讲，可以使用TREATAS来把表达式强制成期望的类型或基数。

TREATAS获得一自变量和一类型。它试图把自变量当作为具有那一类型的自变量。动态影响是：无动作（即，该值保持无变化），否则错误。静态影响是改变表达式的静态类型。不同于Convert或XQueryValidate，跨过类型分级结构进行移动并且不执行解析是不可能的。例如，试图把串当作为整数或者把十进制数当作双整型数都是错误的。

使用目标类型的所有属性。当目标类型的属性等同于变元类型的属性时，那么什么都不做。当属性不同时，则有以下影响：如果自变量基数与目标基数相一致的话，那么无变化地返回值。否则，可能会出现错误。表20：描述基数兼容性。自变量的（准确的、动态的）基数在左侧，而（静态的）目标基数在顶部。“OK”意味着TREAT-AS成功，而“err”意味着它失败。

表20：基数兼容性

序列长度	目标基数						
	零	一	更多	零或一	非一	一或更多	任意
0	ok	err	err	ok	ok	err	ok
1	err	ok	err	ok	err	ok	ok
2+	err	err	ok	err	ok	ok	ok

如果自变量条目种类与目标条目种类相一致的话，那么就无变化地返回该值。否则，发射错误。表21描述了项类型兼容性：

表21：项类型兼容性

自变量种类	目标种类				
	无	AV	节点	两者都是	Dml
无	ok	ok	ok	ok	err
AV	err	ok	err	ok	err
节点	err	err	ok	ok	err
两者都是	err	err	err	ok	err
Dml	err	err	err	err	ok

如果自变量原子类型与目标原子类型相一致的话，那么无变化地返回该值。否则，可能会出现错误。这种兼容性是由子类型可替代性规则来确定的，在该子类型可替代性规则中，当且仅当来自另一节点的一个继承节点受到限制而不得到排列/联合，两种类型才是可兼容的。如果目标类型不是严谨的，那么人们可以查看XML类型信息来确定自变量类型是否是静态类型的子类型。否则，这可以根据QIL原子类型来确定。如果具有目标节点类型的自变量节点类型位法则ORed为零，那么可能会出现错误。否则，节点类型是可兼容的。

最后，如果目标类型不是严谨的，那么不检查严谨性。如果目标类型是严谨的而自变量类型不是严谨的，那么该自变量应当动态地具有严谨的目标类型，否则可能会出现错误。

CANTREATAS运算符

如果TREATAS成功的话，则CANTREATAS就返回真，否则返回假（如果TREATAS将发生错误）。这个运算符主要是为具有CONVERT/ISCONVERTIBLE的对称性而存在的。

CONVERT运算符

CONVERT从一种类型的值到另一种类型的值运用数据类型转换。例如，(Convert (String "1") Iht32)得到整数1。CONVERT不同于TREATAS，它只能投递成所述值已经具有或者将要具有的类型。

所运用的专门的类型转换将匹配XQuery所使用的那些。

ISCONVERTIBLE运算符

如果CONVERT将成功的话，则ISCONVERTIBLE就返回真，否则返回假（如果CONVERT将错误）。一般来讲，它与执行CONVERT一样费用浩大，不

过虑及到条件转移而不是带有错误的终止计算。存在这个运算符主要是为了支持XQuery CASTABLE运算符。

ISTYPE运算符

如果一个值具有已指明的类型，则ISTYPE就返回真。它对于实现类型转换和等效来讲是有用的。对于QilType的每一个方面，都存在一个总是匹配每个值的通配符值（例如，QilCardinality.Any、QilAtomicType.Any等等）。所以，举例来说，为了测试目的，表达式是任意类型的元素，那么人们可以使用QilType，该QilType的XmlNodeKind是QilXmlNodeKind。元素和所有其它的值都是通配符。为了测试出表达式是否具有指定的基数，人们可以使用QilType，该QilType的基数是那个值，并且该类型的所有其它方面都是通配符。

ISEMPTY运算符

与ISTYPE相似（使用QilCardinality.Zero），如果序列为空的话，则这个运算符就返回真，否则返回假。

XQuery运算符

在下面的表22中描述了一组分组为XQuery运算符的运算符。某些语言具有附加运算符，这些附加运算符不能利用其它的QIL运算符来表示，否则它们太过常见以致于它们应当有它们自己的节点类型。

表22：XQuery运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
XqueryEbv	QilUnary	有效的布尔值	一	布尔型
XqueryElement Content	QilUnary	参见下面	任意	-
XqueryAttribute Contert	QilUnary	参见下面	一	串
XQueryValidate	QilUnaryType	参见下面	-	-

XQUERYEBV运算符

这个一元运算符具有一个值序列，并根据XQuery技术规范中所描述的规则将其转换成布尔值。

具体地说，如果操作数是空序列、布尔值假、空串、零（任何数值类型）或NaN（双整型或浮点型）的话，则它就返回假。否则，它就返回真。

XQUERYELEMENTCONTENT运算符

XQUERYELEMENTCONTENT把一个序列作为它的自变量，并返回节点的序列。保持原始序列中的每个节点；将相邻原子值的每个序列转换成包含所述原子值的文本节点，所述原子值与独立的空间级联在一起。

XQUERYATTRIBLrrECONTENT运算符

XQUERYELEMENTCONTENT把一个序列作为它的自变量，并返回一个串值。原始序列中的每个节点都用它的串值来替换；每个原子值也都被转换成串。然后，原子值的结果序列将它们自己与独立的空间级联在一起。

XQUERYVALIDATE运算符

XQUERYVALIDATE 运算符验证节点的序列，这要么会造成合法性错误，要么会产生类型注释后的节点的新序列。它与XQuery VALIDATE运算符具有相同的含义。

XPath运算符

在下面的表X23中描述了一组分组为XPath运算符的运算符。XPath具有表示附加导航坐标（axe）和一些类型转换所需的少量运算符。这些当中的一些可以利用现有的QIL运算符来表示，但是十分的常见，从而应当有它们自己的表示。

XPATHNODEVALUE 、 XPATHNUMBERTOSTRING 和XPATHSTRINGTONUMBER实现XPath 类型转换规则。这三个运算符都是一元运算符。对于XPATHNODEVALUE这个操作数必须是单元素集合节点。操作数 XPATHNUMBERTOSTRING 必须是单元素集合数值类型。对于XPATHSTRINGTOMBER这个操作数必须是单元素集合串。

其它的运算符执行导航。全部都是一元运算符，这些一元运算符的操作数必须是单元素集合节点，并且通过该导航操作，它们产生了期望的节点序列。

表23：XPath运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
XPathNodeValue	QilUnary	根据Xpathe 1.0规则将节点转换成串	一	串
XpathNumber ToString		根据Xpathe 1.0规则将数字转换成串	一	串
XpathStringTo Number	QilUnary	根据Xpathe 1.0规则将串转换成数字	一	Double
XpathAncester	QilUnary	ancestor::node()	任意	无
XpathAncestor OrSelf	QilUnary	ancestor-or-self::node()	任意	无
XpathDescend Ant	QilUnary	descendant::node()	任意	无
XpathFollowing	QilUnary	following::node()	任意	无
XpathFollowing Sibling	QilUnary	following-sibling::node()	任意	无
XpathNamespace	QilUnary	namespsce::node()	任意	无
XpathPreceding	QilUnary	preceding::node()	任意	无
XpathPreceding Sibling	QilUnary	preceding-sibling::node()	任意	无

DML运算符

在下面的表24中描述了一组分组为DML运算符的运算符。QIL能够表示数据变化的操作，比如像插入、删除和更新。

XQuery DML语法改变Xquery语法的条件产生式，以便允许DML语句作为每一个分支。对于条件的QIL表示来说，这要求无变化；编译器将把DML表达式配置成条件节点的真/假分支。XQuery DML语法也改变FLWR表达式，来允许DML语句出现在普通的return子句的位置上。FLWR表达式的QIL表示（元组）是TUPLE节点的返回成员。XQuery DML语法添加了四个新的运算符：INSERT、UPDATE、DELETE和REPLACE。列在表24上的QIL运算符表示这些与Xquery

DML技术规范所定义的语义具有相同语义的运算符。

因为DML操作没有返回值，所以它们没有相应的QILDataType。值QilItemKind.Dml表示DML表达式类型，以便于在编译器和最佳化器中检查的静态类型。为了测试QIL表达式是否可以执行DML，人们需要查看最上节点的MemberKind。

表24：DML运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
DmlInsertAfter	QilBinary	参见Xquery DML技术规范	无	无
DmlInsertBefor	QilBinary	参见Xquery DML技术规范	无	无
DmlInsertInto	QilBinary	参见Xquery DML技术规范	无	无
DmlInsertIntoFirst	QilBinary	参见Xquery DML技术规范	无	无
DmlInsertIntolast	QilBinary	参见Xquery DML技术规范	无	无
DmlDelete	QilUnary	参见Xquery DML技术规范	无	无
DmlUpdate	QilTernary	参见Xquery DML技术规范	无	无
DmlReplace	QilBinary	参见Xquery DML技术规范	无	无

特定运算符

在下面的表25中描述了一组分组为特殊算符的运算符。存在NOP节点类型主要是为了支持构建QIL立即响应或支持便于QIL的自顶向下组成。举例来说，开发人员可以将NOP节点置于图形中，并且稍后随着组成发展来安排其内容。注意：仍然必须适当地设置节点类型。

存在UNKNOWN节点类型主要是为了支持在编译时期间局部构建QIL，并且允许应用为它们自己的目的而“再使用”QIL。基本上，UNKNOWN是不透

明的节点类型，它没有定义好的语义，并且完全可以利用任何类来表示它。

ERROR节点类型具有两个自变量。第一个必须是串序列，并且向用户描述消息。第二个必须是布尔型单元素集合，并且当它为真时，表明错误是致命的（re：终止执行）。当其为假时，除消息以外执行可以继续。

在操作数的类型或基数方面没有限制。

表25：特定运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
Unknown	-	n/a	-	-
Nop	QILUnary	无操作（委托给其孩子节点）	-	-
Error	QILBinary	动态错误或消息	无	无

外部数据源运算符

在下面的表26中描述了一个被称为外部数据源的运算符。外部数据的主源是DATASOURCE节点类型，它是利用QILDATASOURCE类来实现的。这个运算符具有串标记值，该串标记值描述了数据源的种类（例如，像域名空间）以及任意的User Data对象，该User Data对象不是由QIL来解释的，但是这个运算符可以具有一些后端处理器的含义。

数据源可以是到目录的路径名或标题，或者可以是通用的资源定位地址。对于这个数据源类型来说，User Data对象是赋值给一个串的QIL节点表达式。串值标识了传递给函数的位置，以供加载XIVIL文档（例如，完全类似于对document()函数的调用）。XML文档不必是实际的XML；所述文档可以是XML的抽象，比如一种类型的虚拟XML。

表26：外部数据源运算符

QIL节点运算符	类	语义	QIL类型	
			基数	原子类型
DataSource	QILDataSource	任意的（可能是虚拟XML文档或碎片）	任意	无

结论

如上所述，尽管已经结合各种计算装置和网络体系结构描述了本发明的示

例性实施例，但是可以将下位概念应用于任何期望在其中实现XML的中间语言表示的计算装置或系统。因此，本发明的方法和系统可以应用于各种应用程序和装置。尽管在这里选择了代表各种不同选择特征的示例性程序设计语言、名称和例子，但是这些语言、名称和例子都不意在限制本发明。本领域的普通技术人员将会认识到：存在提供能实现由本发明所实现的相同、类似或等效的系统和方法的目标代码的许多种方式。

这里所描述的各种技术都可以结合硬件或软件来实现，或者在适当的情况下，结合上述两者的组合来实现。因此，本发明的方法和设备或者其某些方面或部分都可以采取嵌入在有形介质中的程序代码（即，指令）的形式，所述有形介质比如是软磁盘、CD-ROM、硬盘或任何其它的机器可读存储介质，其中当程序代码被载入并由机器（比如，计算机）执行时，所述机器就变为了用于实施本发明的设备。在程序代码执行在可编程计算机上的情况下，计算装置通常将包括：处理器、由处理器可读的存储介质（包括，易失性和非易失性存储器和/或存储元件）、至少一个输入装置以及至少一个输出装置。可以应用本发明的信号处理服务的一个或多个程序，例如，通过使用数据处理API等等，最好是用高级程序或面向对象编程语言来实现，以便与计算机进行通信。然而，如有需要，可以用汇编或机器语言来实现所述程序。在任何情况下，所述语言都可以是编译语言或解释语言，并且同硬件实施方案相结合。

本发明的方法和设备还可以经由以程序代码形式嵌入的通信来实现，在一些传输介质上发送所述程序代码，比如在电线或电缆上、通过光纤，或通过任何其它形式进行发送，其中当接收程序代码并将其载入并且由机器执行时，该机器比如是EPROM、门阵列、可编程逻辑器件（PLD）、客户端计算机、录像机等等、或具有如上面示例性实施例所描述的信号处理能力的接收机，所述机器就变为了用于实施本发明的设备。当在通用处理器上实现时，所述程序代码与处理器结合，从而提供操作以调用本发明函数的唯一设备。另外，结合本发明所使用的任何存储技术常常都是硬件和软件的组合。

尽管已经结合各个图的优选实施例描述了本发明，但是需要理解，在不脱离本发明的情况下，也可以使用或修改其它类似的实施例，并且可以对用于执行本发明的相同功能的所述实施例进行添加。此外，应当强调的是，特别是随着无线联网装置的数量继续增加，可以考虑各种不同的计算机平台，包括手提

式装置操作系统及其它应用程序特定的操作系统。因此，本发明不应该限于任何单个实施例，而是应该在依照所附权利要求的广度和范围上考虑。

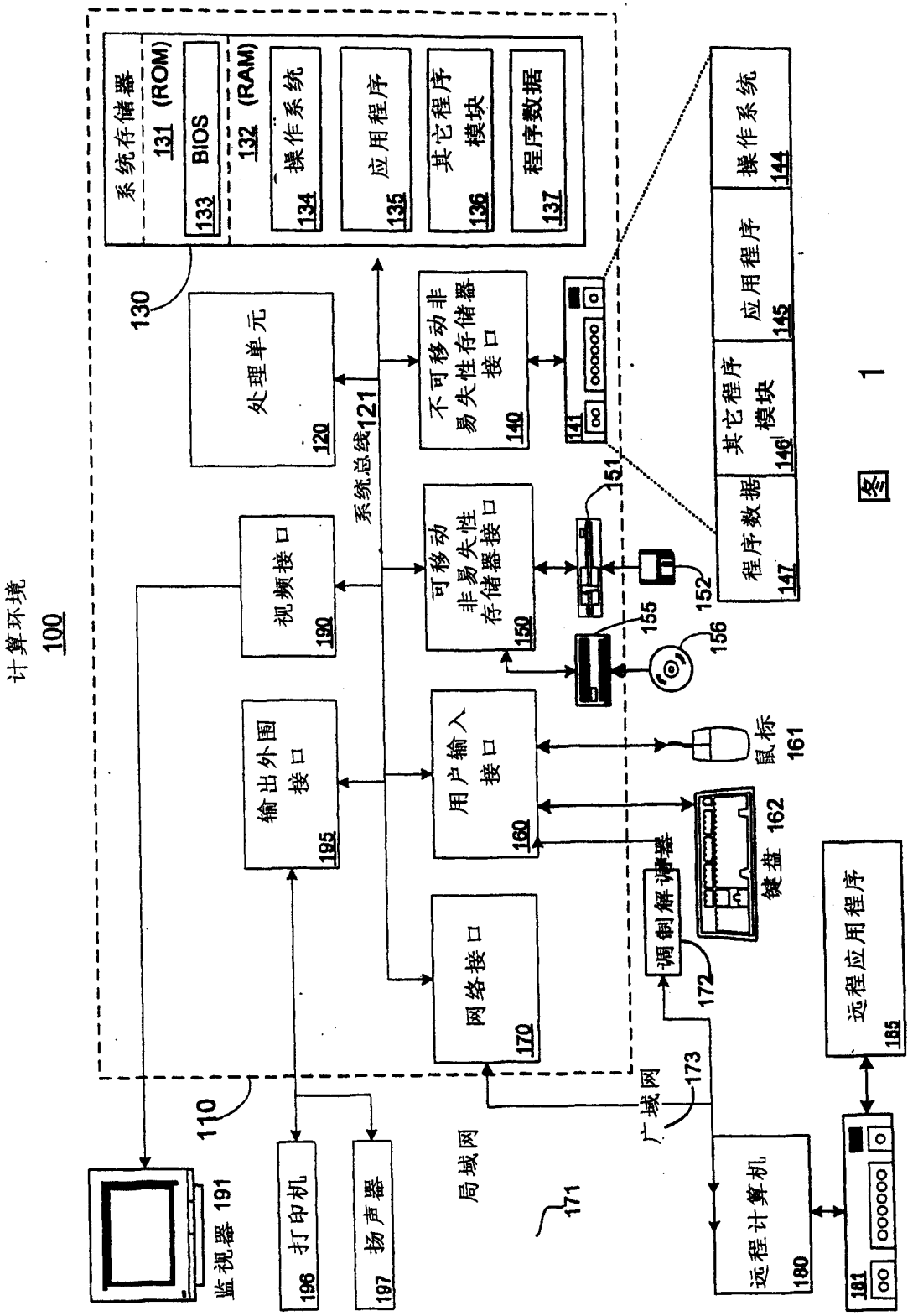

计算环境
100
系统存储器
131 (ROM)
133 BIOS
132 (RAM)
操作系统
134
应用程序
135
其它程序模块
136
程序数据
137
130
处理单元
120
系统总线121
不可移动非易失性存储器接口
140
141
151
操作系统
144
应用程序
145
其它程序模块
146
程序数据
147
视频接口
190
可移动非易失性存储器接口
150
152
155
156
输出外围接口
195
用户输入接口
160
鼠标
161
键盘
162
网络接口
170
调制解调器
172
远程应用程序
185
110
监视器 191
打印机
196
扬声器
197
局域网
171
广域网
173
远程计算机
180
181

图 1

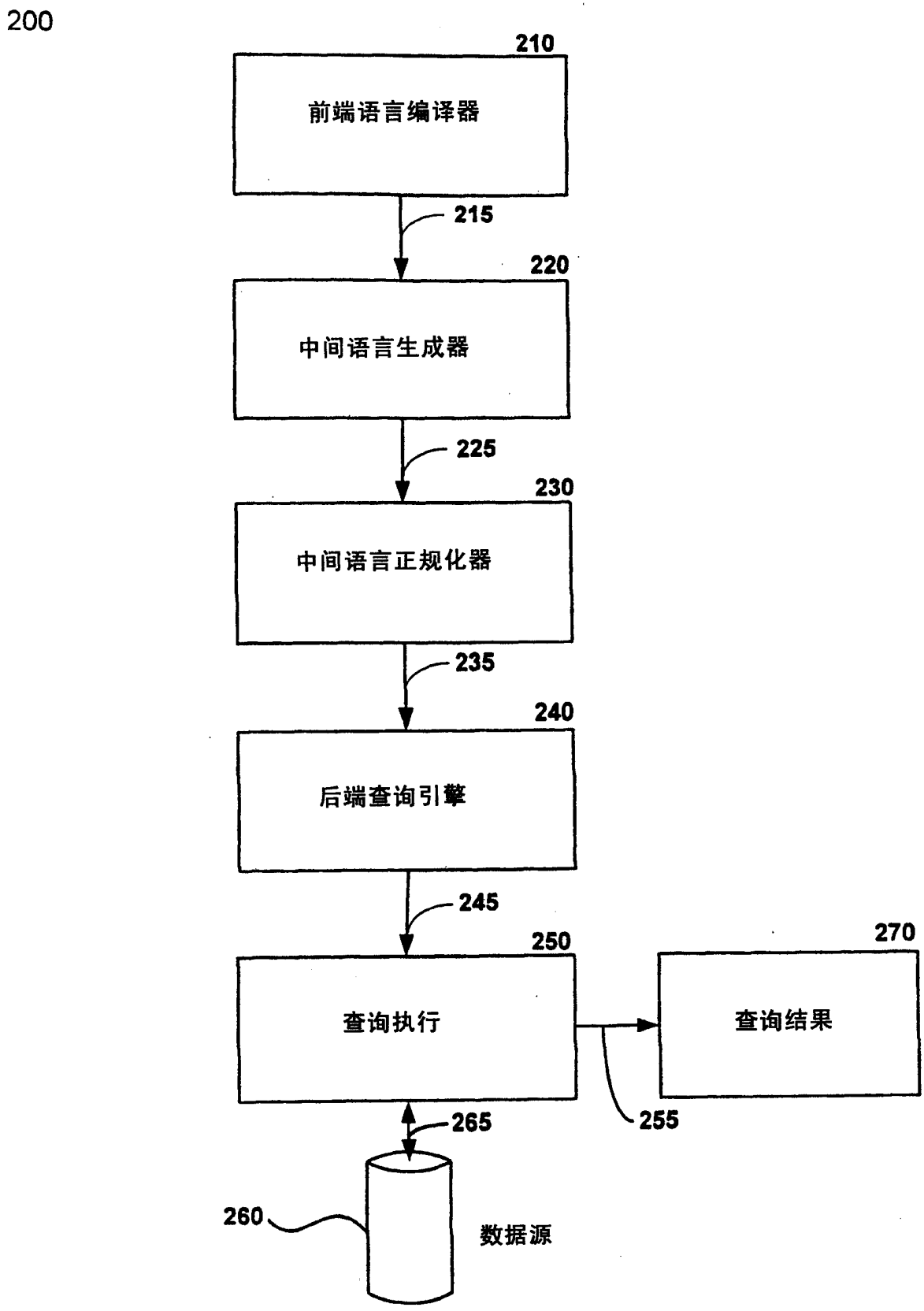

200
210
前端语言编译器
215
220
中间语言生成器
225
230
中间语言正规化器
235
240
后端查询引擎
245
250
查询执行
270
查询结果
255
265
260
数据源

图 2

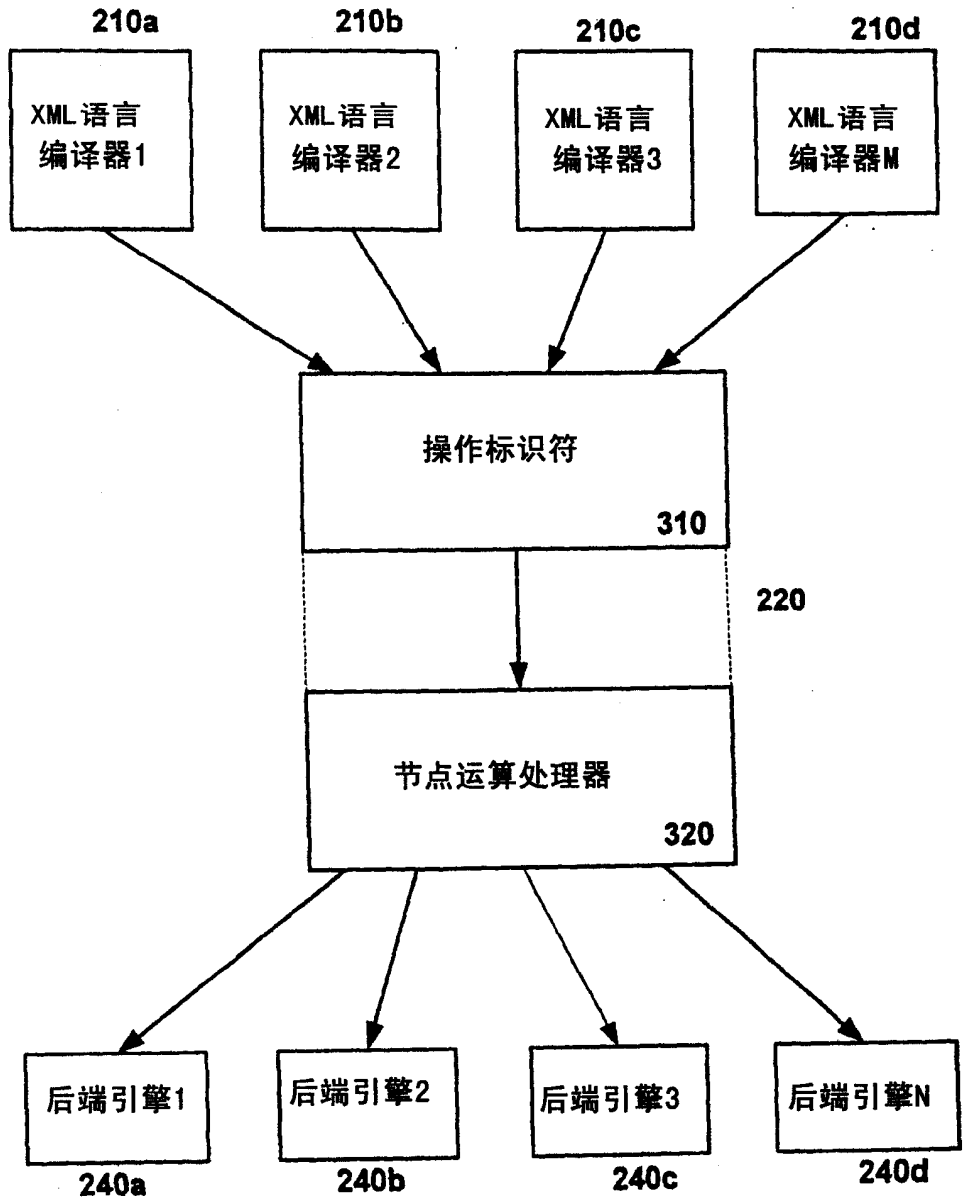

210a
210b
210c
210d
XML语言编译器1
XML语言编译器2
XML语言编译器3
XML语言编译器M
操作标识符
310
220
节点运算处理器
320
后端引擎1
后端引擎2
后端引擎3
后端引擎N
240a
240b
240c
240d

图 3

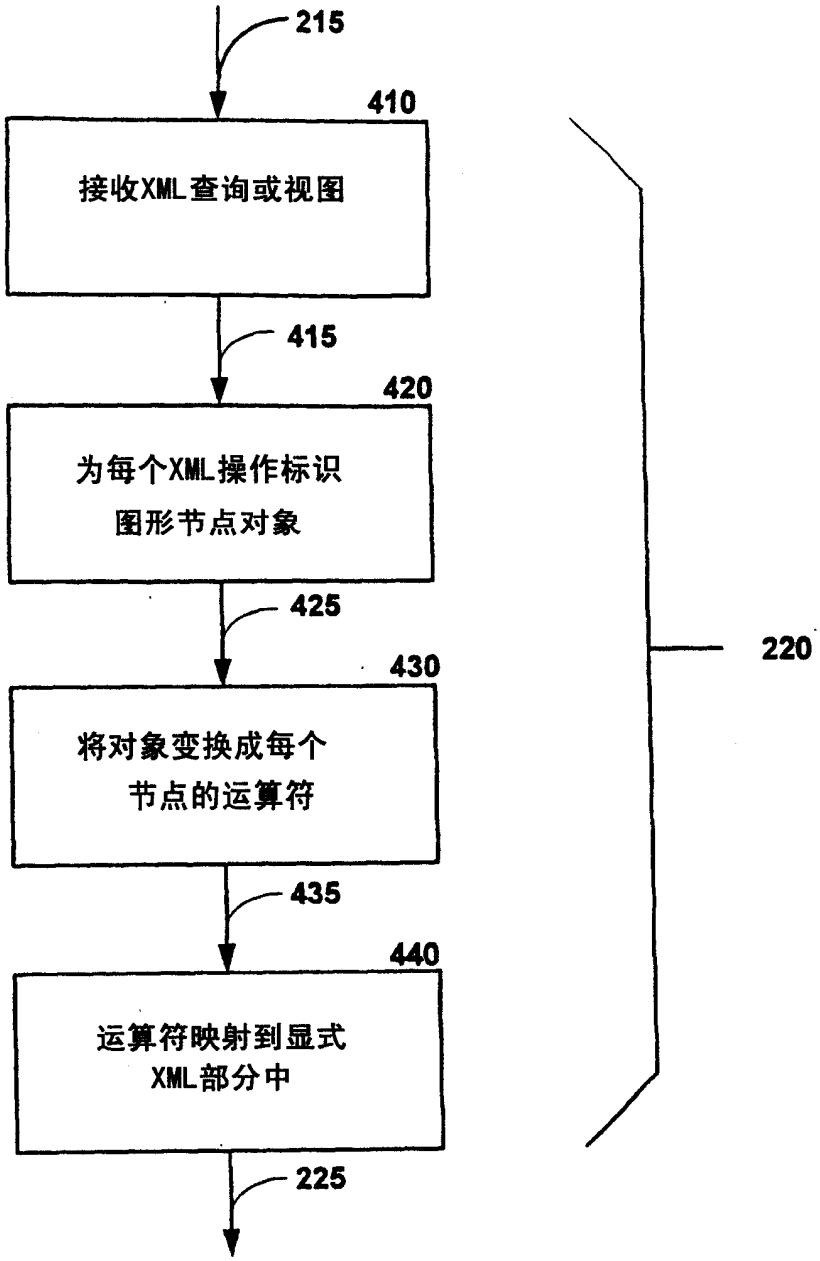

215
410
接收XML查询或视图
415
420
为每个XML操作标识
图形节点对象
425
430
将对象变换成每个
节点的运算符
435
440
运算符映射到显式
XML部分中
225
220

图 4