(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 102456070 B
(45)授权公告日 2016.03.30

(21)申请号 201110247657.4
(22)申请日 2011.08.24
(30)优先权数据
238531/2010 2010.10.25 JP
(73)专利权人 株式会社东芝
地址 日本东京都
专利权人 东芝解决方案株式会社
(72)发明人 服部雅一
(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 陈萍
(51)Int.Cl.
G06F 17/30(2006.01)
(56)对比文件
US 2009/0132544 A1,2009.05.21,
CN 101160884 A,2008.04.09,
审查员 曹姝妹

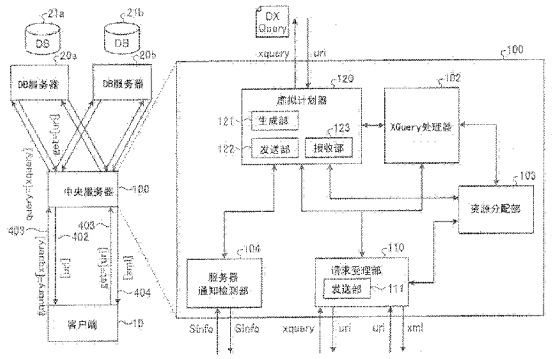
权利要求书2页 说明书17页 附图23页

(54)发明名称

检索装置和检索方法

(57)摘要

一种检索装置和检索方法。检索装置包括：第一受理部、第一生成部、第一发送部、第二发送部、第二受理部、接收部、执行部和第三发送部。第一受理部从客户端受理检索请求。第一生成部根据检索请求生成对服务器请求检索的分布式检索请求、以及对分布式检索请求的检索结果进行综合的综合请求。第一发送部向服务器发送分布式检索请求。第二发送部向客户端发送综合请求的执行结果的识别信息。第二受理部从客户端受理由识别信息所识别的执行结果的获取请求。接收部从服务器接收针对分布式检索请求的检索结果。执行部对接收到的检索结果执行综合请求。第三发送部向发出了获取请求的客户端发送综合请求的执行结果。



CN 102456070 B

1. 一种检索装置，由客户端和存储数据的多个服务器经由网络连接而成，其特征在于，具备：

第一受理部，从所述客户端受理所述数据的 XQuery 形式的检索请求；

第一生成部，根据所述第一受理部受理的检索请求，生成分别对多个所述服务器请求所述数据的检索的 XQuery 形式的分布式检索请求、以及对所述分布式检索请求的检索结果进行综合的 XQuery 形式的综合请求；

第一发送部，将所述分布式检索请求发送到多个所述服务器；

第二发送部，将所述综合请求的执行结果的识别信息发送到所述客户端；

第二受理部，从所述客户端受理由所述识别信息所识别的执行结果的获取请求；

接收部，从多个所述服务器接收针对所述分布式检索请求的 XML 形式的检索结果；

执行部，对分别从多个所述服务器接收到的所述检索结果执行所述综合请求；以及

第三发送部，对发送了所述获取请求的所述客户端发送所述综合请求的执行结果，

所述第一受理部受理的检索请求包含至少一个用于确定要检索的所述数据的函数，

所述第一生成部按照每个所述函数，生成包含所述函数和所述函数以下的路径表达式在内的所述分布式检索请求。

2. 如权利要求 1 所述的检索装置，其特征在于，

还具备分配部，确保用来保存所述综合请求的执行结果的区域并将所述区域的识别信息作为所述综合请求的执行结果的识别信息而分配给所述执行结果；

所述第二发送部将所分配的所述识别信息发送给所述客户端；

所述执行部将所述综合请求的执行结果保存到所述区域中；

所述第三发送部向发送了所述获取请求的所述客户端发送所述区域中保存的所述执行结果。

3. 如权利要求 1 所述的检索装置，其特征在于，还具备：

变换部，将所述分布式检索请求变换为满足所述服务器的检索能力的 XQuery 形式的检索请求；以及

第二生成部，根据变换后的检索请求的检索结果，生成用于验证变换后的检索请求的、XQuery 形式的检索请求即生成请求；

所述第一发送部将变换后的检索请求发送到所述服务器；

所述接收部从所述服务器接收针对变换后的检索请求的检索结果；

所述执行部还对所接收到的检索结果执行所述生成请求，并对所述生成请求的执行结果执行所述综合请求。

4. 如权利要求 3 所述的检索装置，其特征在于，

所述变换部通过所述分布式检索请求中包含的路径的省略、OR 条件的展开、以及所述数据中包含的元素名称的变换中的至少 1 个，将所述分布式检索请求变换为满足所述服务器的检索能力的检索请求。

5. 如权利要求 1 所述的检索装置，其特征在于，

其还具备检测部，用于检测与所述网络连接的所述服务器；

所述第一发送部将所述分布式检索请求发送到检测到的所述服务器。

6. 如权利要求 1 所述的检索装置，其特征在于，

所述函数是 doc() 函数。

7. 如权利要求 1 所述的检索装置，其特征在于，

所述第一生成部通过将所述第一受理部受理的检索请求中包含所述函数的部分置换为用于从所述执行结果中提取所述数据的 for 语句或 let 语句，来生成所述综合请求。

8. 一种检索方法，在由客户端和存储数据的多个服务器经由网络连接而成的检索装置中执行，其特征在于，

从所述客户端受理所述数据的 XQuery 形式的检索请求；

根据受理的所述检索请求，生成分别对多个所述服务器请求所述数据的检索的 XQuery 形式的分布式检索请求、以及对所述分布式检索请求的检索结果进行综合的 XQuery 形式的综合请求；

将所述分布式检索请求发送到多个所述服务器；

从多个所述服务器接收针对所述分布式检索请求的 XML 形式的检索结果；

对分别从多个所述服务器接收到的所述检索结果执行所述综合请求；

将所述综合请求的执行结果的识别信息发送到所述客户端；

从所述客户端受理由所述识别信息识别的执行结果的获取请求；

对发送了所述获取请求的所述客户端发送所述综合请求的执行结果，

受理的所述检索请求包含至少一个用于确定要检索的所述数据的函数，

按照每个所述函数，执行所述分布式检索请求的生成，所述分布式检索请求包含所述函数和所述函数以下的路径表达式。

检索装置和检索方法

[0001] 本申请享有于 2010 年 10 月 25 日在先提出的日本国专利申请号 2010-238531 的优先权，并将其全部内容都包含在本申请中。

技术领域

[0002] 本发明的实施方式涉及检索装置和检索方法。

背景技术

[0003] 现在已经开发了利用分布式系统实现 XQuery 处理的分布式 XQuery 处理技术。但是，分布式 XQuery 处理的尝试还只是刚刚开始，关于分布式 XQuery 处理的论文仅处于偶尔可见的程度。

[0004] 作为分布式 XQuery 处理之一，XRPC 是一种针对不同种类的分布式数据源的 XQuery 处理的语言扩展，作为 XQuery 的内部函数具备 RPC(Remote Procedure Call: 远程过程调用）功能，由此实现分布式 XQuery。

发明内容

[0005] 在分布式 XQuery 处理的代表性现有技术 XRPC 中存在着例如用户必须在 XQuery 内显式描述作为特殊语言扩展的 XPRC 这样问题。

[0006] 本实施方式的检索装置包括：第一受理部、第一生成部、第一发送部、第二发送部、第二受理部、接收部、执行部和第三发送部。第一受理部从客户端 (client) 接受检索请求。第一生成部根据所述第一受理部受理的检索请求，生成分别对多个所述服务器请求所述数据的检索的 XQuery 形式的分布式检索请求、以及对所述分布式检索请求的检索结果进行综合的 XQuery 形式的综合请求。第一发送部向服务器发送分布式检索请求。第二发送部将所述综合请求的执行结果的识别信息发送到所述客户端。第二受理部从所述客户端受理由所述识别信息所识别的执行结果的获取请求。接收部从多个所述服务器接收针对所述分布式检索请求的 XML 形式的检索结果。第三发送部对发送了所述获取请求的所述客户端发送所述综合请求的执行结果。所述第一受理部受理的检索请求包含至少一个用于确定要检索的所述数据的函数。所述第一生成部按照每个所述函数，生成包含所述函数和所述函数以下的路径表达式在内的所述分布式检索请求。

附图说明

[0007] 图 1 是表示包含使用 XRPC 的情况的检索装置的数据库系统的结构的一个实例的框图。

[0008] 图 2 是表示本实施方式的虚拟 XML 数据库系统的网络结构实例的框图。

[0009] 图 3 是表示 HTTP 上的通信步骤的图。

[0010] 图 4 是表示第一实施方式的中央服务器的结构实例的框图。

[0011] 图 5 是表示第一实施方式中的检索处理的整体流程的流程图。

[0012] 图 6 是用于说明对 GET 消息进行语法分析后对处理进行分配的请求处理的图。

[0013] 图 7 是表示由虚拟计划器 (planner) 执行的分布式 XQuery 处理的一个实例的流程图。

[0014] 图 8 是表示 DXQuery 生成处理的一个实例的流程图。

[0015] 图 9 是表示 GXQuery 生成处理的一个实例的流程图。

[0016] 图 10 是表示 XQuery 处理的一个实例的流程图。

[0017] 图 11 是表示获取处理的一个实例的流程图。

[0018] 图 12 是表示合并处理的一个实例的流程图。

[0019] 图 13 是按照时间序列展示出客户端和多个服务器的相互作用的状况的序列图。

[0020] 图 14 是表示从客户端输入的 XQuery 的一个实例的图。

[0021] 图 15 是表示图 2 的 DB 中存储的数据的一个实例的图。

[0022] 图 16 是表示根据图 14 的 XQuery 生成的 DXQuery 的一个实例的图。

[0023] 图 17 是表示作为图 16 的 DXQuery 的执行结果的结果 XML 的一个实例的图。

[0024] 图 18 是表示根据图 14 的 XQuery 生成的 GXQuery 的一个实例的图。

[0025] 图 19 是表示作为图 18 的 GXQuery 的执行结果的结果 XML 的一个实例的图。

[0026] 图 20 是表示从客户端输入的 XQuery 的另一个实例的图。

[0027] 图 21 是表示针对图 20 的 XQuery 生成的 DXQuery 的示意图。

[0028] 图 22 是表示 DXQuery1 的一个实例的图。

[0029] 图 23 是表示作为图 22 的 DXQuery1 的执行结果的结果 XML 的一个实例的图。

[0030] 图 24 是表示 DXQuery2 的一个实例的图。

[0031] 图 25 是表示作为图 24 的 DXQuery2 的执行结果的结果 XML 的一个实例的图。

[0032] 图 26 是表示根据图 20 的 XQuery 生成的 GXQuery 的一个实例的图。

[0033] 图 27 是表示作为图 26 的 GXQuery 的执行结果的结果 XML 的一个实例的图。

[0034] 图 28 是表示第二实施方式中的中央服务器的结构实例的框图。

[0035] 图 29 是表示由虚拟计划器和泛化处理器执行的分布式 XQuery 处理的一个实例的流程图。

[0036] 图 30 是表示泛化处理的一个实例的流程图。

[0037] 图 31 是表示分布式服务器定义的一个实例的图。

[0038] 图 32 是表示在输入了图 14 的 XQuery 和图 31 的分布式服务器定义时利用泛化处理所输出的 DXQuery’的一个实例的图。

[0039] 图 33 是表示在输入了图 14 的 XQuery 和图 31 的分布式服务器定义时利用泛化处理所输出的 VXQuery 的一个实例的图。

[0040] 图 34 是表示在第二实施方式中所处理的 XQuery、资源和 XML 的关系的一个实例的图。

[0041] 图 35 是表示第一或第二实施方式的检索装置的硬件结构的说明图。

具体实施方式

[0042] 下面参照附图详细说明本发明的检索装置的优选实施方式。下面以通过 XQuery 形式的检索请求对 XML(Extensible Markup Language: 可扩展标记语言）形式的数据进行

检索的系统为例进行说明，但可以应用本发明的系统并不限于此。

[0043] 在 XML 中，构成文档结构的各个部分 (parts) 被称为“元素 (Element)”。各元素使用标签 (Tag) 进行记述。具体而言，一个元素表现为由表示元素开始的标签（开始标签）和表示元素结束的标签（结束标签）这两个标签将文本数据包夹。另外，由开始标签和结束标签所包夹的文本数据就是由开始标签和结束标签表示的一个元素中所包含的文本元素（文本节点）。

[0044] XQuery 是用于查询 XML 数据库 (XML-DBMS) 的函数型语言，其特征是 FLWOR(for-let-where-order by-return) 语法。作为 RDB 中的查询语言的 SQL 是声明式语言，与此相对，XQuery 具有很多函数型语言的特征。下面从过程的角度出发来说明 XQuery 的语言规范。

[0045] for 语句的语法是“for 变量 in 表达式”。for 语句的语法具有将满足表达式的项目代入变量后执行循环的意思。let 语句的语法是“let 变量：＝表达式”。let 语句的语法具有将满足表达式的项目汇集起来作为序列代入变量的意思。序列指的是扁平的列表。where 语句对 for 语句中重复执行的循环进行限制。where 语句的语法是“where 表达式”。where 语句的语法具有仅针对满足表达式的项目执行循环、对于不满足表达式的项目则跳过循环的意思。return 语句对 XQuery 的处理结果进行格式化。return 语句的语法是“return 表达式”。在 return 语句的语法中可以记述包含变量在内的任意的 XML 数据。变量的语法是“$ 字符串”。除了在嵌套查询等中进行双重声明的情况以外，具有相同字符串的变量被看作是同一变量。作为用于指定 XML 数据的元素之间的层次条件的路径运算符，XQuery 中存在着以下运算符。

[0046] (1)“/”：表示元素之间是父子关系的运算符

[0047] (2)“//”：表示元素之间是祖孙关系的运算符

[0048] (3)“.”：任意元素

[0049] 如上所述，作为利用分布式系统来实现 XQuery 处理的分布式 XQuery 处理技术，已知有 XRPC。

[0050] 图 1 是表示包含使用 XRPC 的情况的检索装置（中央服务器 100’）的数据库系统的结构的一个实例的框图。如图 1 所示，数据库系统具有中央服务器 100’、客户端 10’、具备数据库 (DB)21 的 DB 服务器 20’经由网络 30 连接而成的结构。

[0051] 客户端 10’向中央服务器 100’请求以 XQuery 形式记述的查询 41。查询 41 的意思是：对于 ActorA 和 ActorB，调用 x.example.org 网站中存在的 XQuery43 的函数 filmsByActor。XQuery43 的意思是：从 XML 文件 42 提取具有与变量 $actor 匹配的 actorName 的 filmName。

[0052] 在受理了这种查询 41 时的处理概要如下所示。

[0053] (1) 对于“ActorA”，调用函数 filmsByActor。在 XML 文件 42 中存在 2 个 actorName 中包含“ActorA”的元素，因此返回这 2 个 filmName。

[0054] (2) 对于“ActorB”，调用函数 filmsByActor。在 XML 文件 42 中不存在 actorName 中包含“ActorB”的元素，因此返回空数据。

[0055] (3) 将所得到的 filmName 表示为查询 41 中记述的 XML 形式。结果 XML44 表示出此时所得到的 XML。结果 XML44 是将 films 元素附加在函数所返回的 filmName 元素的上下

而形成的。

[0056] 在 XRPC 中存在着如下所示的课题。

[0057] (1)XQuery 非透明：用户必须在 XQuery 内显式记述作为特殊的语言扩展的 XRPC。

[0058] (2) 同类综合 :DB 服务器 20’的查询处理能力必须支持 XQuery 和 XRPC。其结果是，针对不同种类的数据无法实现真正的虚拟化。

[0059] (3) 性能问题：一旦 XQuery 的 for 循环内存在 RPC 函数，RPC 消息 (SOAP 格式) 的发送接收次数就会增大。另外，RPC 函数返回单值，因此不适合双值的返回。另外，内部的 XQuery 被封装为 RPC 函数，因此，难以对嵌套的 XQuery 进行优化。

[0060] 因此，包含作为第一实施方式的检索装置的中央服务器 100 的虚拟 XML 数据库系统不使用 RPC 而实现分布式 XQuery 处理。

[0061] 图 2 是表示本实施方式的虚拟 XML 数据库系统的网络结构实例的框图。虚拟 XML 数据库系统由客户端 10、中央服务器 100、2 台 DB 服务器 20a、20b 经由网络 30 连接而成。

[0062] DB 服务器 20a、20b 分别具备用于存储例如 XML 形式数据的数据库 (DB)21a、21b。此外，DB 服务器 20a、20b 具备同样的功能，因此，在以下说明中有时候简称为 DB 服务器 20。

[0063] 客户端 10 向中央服务器 100 请求以 XQuery 形式记述的查询。网络 30 可以采用 LAN(Local Area Network: 局域网) 和 WAN(Wide Area Network: 广域网) 等任意的网络结构。

[0064] 网络 30 上存在多种多样的通信协议，以下以使用网际协议相互连接的计算机网络、即 IP 网络为例进行说明。使用 IP 以外的通信协议时也可以应用同样的手法。

[0065] 图 3 是表示 HTTP(Hypertext Transfer Protocol: 超文本传输协议) 上的通信步骤的图。图 3 表示出客户端 10 和中央服务器 100 之间的 HTTP 通信步骤，但是在中央服务器 100 和 DB 服务器 20 之间也按照同样的通信步骤进行通信。图 3 的通信步骤是由 REST(Representational State Transfer: 表述性状态转移) 扩展而成的。

[0066] 所谓的 REST 指的是用于分布式超媒体系统 (hypermedia system) 的软件架构 (architecture) 的类型，其特征为无状态 (stateless) 式客户端 / 服务器协议。HTTP 消息包含为理解该请求 (消息) 所必需的全部信息。因此，客户端 10 和中央服务器 100 的任意一方都不需要记住消息之间的会话状态。另外，定义了若干个高使用频度的方法。其中重要的方法是 get、post、put 和 delete。

[0067] 另外，在 REST 中使用了由 URI(Uniform Resource Identifier: 统一资源标识符) 所表达的唯一地址 (unique adress)，用于唯一地识别资源。URI 用作方法的参数。URI 是由 URL(Uniform Resource Locator: 统一资源定位符) 的概念扩展而得到的。URI 是按照固定的书写格式指示资源的标识符，在 1998 年规定为 RFC2396，并于 2005 年修订为 RFC3986。

[0068] 例如，“xxxx.ne.jp”上的资源“index.html”的 URI 表达为“http://www.xxxx.ne.jp/yyyy/public/index.html”。

[0069] 在本实施方式中，将 REST 扩展为包含方法“query”和方法“gquery”。如下面的 (1) 和 (2) 所示，这些方法是分别用于请求 XQuery 处理和分布式 XQuery 处理的执行的方法。

[0070] 下面展示本实施方式中使用的方法的一个实例。此外，在以下说明中，小写的“uri”和“uri_*(* 是任意的字符串)”表示其是按照 uri 的书写格式所表达的标识符。另

外，“资源 uri”或“资源 uri_*”表示由标识符“uri”或“uri_*”所识别的资源。

[0071] (1)query 方法

[0072] 指定 XQuery（图 3 的“XQuery”）（步骤 S101），取得保存有由指定的 XQuery 进行的 XQuery 处理的结果 XML 的资源 uri（步骤 S102）。

[0073] (2)gquery 方法

[0074] 指定 XQuery（图 3 的“XQuery”）（步骤 S101），取得保存有由指定的 XQuery 进行的分布式 XQuery 处理的结果 XML 的资源 uri（步骤 S102）。

[0075] (3)get 方法

[0076] 指定 uri（步骤 S103），获取所指定的 uri 的资源中所保存的结果 XML（步骤 S104）。

[0077] (4)put 方法

[0078] 指定 uri 和 XML（步骤 S105），将 XML 保存到所指定的 uri 的资源中，获取保存结果（图 3 的“status”）（步骤 S106）。

[0079] 图 4 是表示第一实施方式的中央服务器 100 的结构实例的框图。中央服务器 100 包括：请求受理部 110（第一受理部、第二受理部）、虚拟计划器 120、XQuery 处理器 102（执行部）、资源分配部 103（分配部）和服务器通知检测部 104（检测部）。

[0080] 请求受理部 110 受理来自客户端 10 或其他服务器（DB 服务器 20a、20b 等）的请求。例如，请求受理部 110 受理 query 或 gquery 等 XQuery 处理请求以及 get 或 put 等资源处理请求，调用必要的处理。请求受理部 110 具备发送部 111（第二发送部、第三发送部），用于发送对请求的响应。

[0081] 如图 4 的左侧所示，请求受理部 110 例如从客户端 10 受理数据的检索请求 401。发送部 111 将保存着结果 XML 的资源的 uri402 返回给发送了检索请求 401 的客户端 10。另外，请求受理部 110 从客户端 10 受理由 uri 指定的资源中所保存的结果 XML 的获取请求 403。发送部 111 向发送了获取请求 403 的客户端 10 返回结果 XML404。

[0082] 虚拟计划器 120 执行分布式 XQuery 处理中的计划。虚拟计划器 120 包括：生成部 121（第一生成部）、发送部 122（第一发送部）和接收部 123。

[0083] 当请求受理部 110 受理了请求执行分布式 XQuery 处理的分布式 gquery 时，生成部 121 根据所受理的检索请求（gquery）所指定的 XQuery 生成 DXQuery（分布式检索请求）和 GXQuery（综合请求），其中，DXQuery 向 DB 服务器 20 的集合请求数据的检索，GXQuery 对 DXQuery 的检索结果进行综合。

[0084] 发送部 122 将 DXQuery 的检索请求发送到 DB 服务器 20。接收部 123 从 DB 服务器 20 接收 DXQuery 的检索结果。

[0085] XQuery 处理器 102 执行 XQuery。

[0086] 资源分配部 103 对以 uri 为键值的资源进行管理。在 XQuery 中传递的是 XML 数据，但是为了对其传递进行控制，资源分配部 103 对各 XML 数据分配资源 uri。资源分配部 103 具备：确保用于保存资源的区域的功能；返回用于表达所确保的区域的 URI 的 uri 返回功能；将 XML 数据代入资源 uri 的功能；以及获取在 uri 所表示的区域中保存的 XML 数据的功能。

[0087] 服务器通知检测部 104 使用广播等在服务器之间进行服务器信息（SInfo）的传递，检测与网络 30 连接的服务器。虚拟计划器 120 向按照这种方式检测到的 DB 服务器 20

发送 DXQuery。此外，也可以不配备服务器通知检测部 104，而是采用对虚拟计划器 120 事先指定的 DB 服务器 20 进行访问的结构。

[0088] 接着，使用图 5 说明由按照这种方式构成的第一实施方式的中央服务器 100 进行的检索处理。图 5 是表示第一实施方式中的检索处理的整体流程的流程图。

[0089] 首先，通过中央服务器 100 的启动而生成主线程（步骤 S201）。以下的步骤 S202 到步骤 S208 表示在主线程内执行的各个处理。

[0090] 在主线程中生成 2 个服务器通知检测线程（服务器信息通知线程和服务器信息检测线程）和针对每个请求而生成的请求处理线程（步骤 S202、步骤 S204、步骤 S207）。

[0091] 在服务器信息通知线程中，服务器通知检测部 104 通过例如广播定期地向 DB 服务器 20 等与网络 30 连接的其他装置发送中央服务器 100 的服务器信息（步骤 S203）。该处理重复执行到服务器信息通知线程结束为止。

[0092] 在服务器信息检测线程中，服务器通知检测部 104 从 DB 服务器 20 等与网络 30 连接的其他装置接收服务器信息，更新现有的服务器信息（步骤 S205）。该处理重复执行到服务器信息检测线程结束为止。

[0093] 请求处理线程在套接字 (socket) 上的 accept 处理完成后生成。在 accept 处理中，受理等待连接的请求 (request) 并建立连接，创建新的套接字。

[0094] 请求受理部 110 判断是否已利用所创建的套接字受理了 XQuery 处理请求 (query、gquery) 或资源处理请求 (get、put) 等请求（步骤 S206）。在尚未受理的情况下（步骤 S206 :否），重复该处理，一直到受理为止。

[0095] 在已受理了请求的情况下（步骤 S206 :是），请求受理部 110 生成请求处理线程（步骤 S207）。在请求处理线程中执行请求处理，对所受理的请求进行处理（步骤 S208）。

[0096] 在请求处理中，对所接收到的 HTTP 消息、这里是 GET 消息进行语法分析。HTTP 消息指的是作为请求 (request) 从客户端 10 向中央服务器 100 发送、并且作为响应 (response) 从中央服务器 100 返回到客户端 10 的消息。

[0097] HTTP 消息的结构由多行“消息头”和“消息体”构成，分别以空行 (CR+LF) 隔开。在消息头中含有中央服务器 100 或客户端 10 需要处理的请求或响应的内容等。消息体中含有需要传送的数据本身。

[0098] GET 方法是 HTTP/0.9 中定义的唯一一个方法，在 HTTP 中使用得最频繁。遵循 HTTP/1.1 的服务器必须支持 GET 方法。

[0099] 图 6 是用于说明对 GET 消息进行语法分析后对处理进行分配的请求处理的图。

[0100] 请求受理部 110 从消息中提取方法，根据所取得的方法的种类（消息类型）进行条件分支。

[0101] (1)gquery 方法的情况下

[0102] 因为请求的是分布式 XQuery 处理，所以调用虚拟计划器 120。

[0103] (2)query 方法的情况下

[0104] 因为请求的是 XQuery 处理，所以利用 XQuery 处理器 102 执行 XQuery。

[0105] (3)post 方法的情况下

[0106] 利用资源分配部 103 创建资源，返回所创建的资源的 uri。

[0107] (4)get 方法的情况下

[0108] 获取由 uri 所指定的资源的数据。

[0109] (5)put 方法

[0110] 将数据代入由 uri 所指定的资源。

[0111] (6)merge 方法

[0112] 将数据合并后代入所指定的资源。

[0113] 图 7 是表示利用虚拟计划器 120 执行的分布式 XQuery 处理的一个实例的流程图。在分布式 XQuery 处理中，根据所输入的 XQuery 生成以下两种 XQuery。此外，针对每个 DB 服务器 20 生成多个 DXQuery。GXQuery 通常生成一个。

[0114] (1)DXQuery :是用于访问 DB 服务器 20 中存储的 XML 数据的 XQuery

[0115] (2)GXQuery :是用于对 DXQuery 所输出的 XML 数据进行综合的 XQuery

[0116] 首先，虚拟计划器 120 的生成部 121 对所输入的 XQuery 进行分析，由此执行生成多个 DXQuery 的 DXQuery 生成处理（步骤 S301）。DXQuery 生成处理将在后面详细叙述。

[0117] 虚拟计划器 120 从所生成的 DXQuery 之中选择一个（步骤 S302）。虚拟计划器 120 从多个 DB 服务器 20 之中选择一个（步骤 S303）。虚拟计划器 120 的发送部 122 向所选择的 DB 服务器 20 发送所选择的 DXQuery 的执行请求（步骤 S304）。接收到执行请求的 DB 服务器 20 将保存着 DXQuery 的执行结果（结果 XML）的资源的 URI(uri_d) 发送到中央服务器 100。由此，虚拟计划器 120 获取了保存着结果 XML 的资源的 URI、即 uri_d。

[0118] 接着，虚拟计划器 120 判断是否已经向所有 DB 服务器 20 发送了执行请求（步骤 S305）。在尚未全部发送的情况下（步骤 S305 :否），选择一个尚未处理的 DB 服务器 20，重复进行处理（步骤 S303）。

[0119] 在已经全部发送了的情况下（步骤 S305 :是），虚拟计划器 120 向自身服务器（中央服务器 100）请求用于对分别保存有从各 DB 服务器 20 得到的多个资源 uri_d 的数据进行合并的 merge 方法（步骤 S306）。中央服务器 100 的请求受理部 110 接收到请求了执行 merge 方法的消息后，首先向虚拟计划器 120 返回保存有合并的结果（结果 XML）的资源的 URI(uri_m)。由此，虚拟计划器 120 获取到保存有结果 XML 的资源的 URI、即 uri_m。

[0120] 接着，虚拟计划器 120 判断是否已处理了所有的 DXQuery（步骤 S307）。在尚未全部处理的情况下（步骤 S307 :否），选择一个尚未处理的 DXQuery，重复进行处理（步骤 S302）。

[0121] 在已经处理了全部 DXQuery 的情况下（步骤 S307 :是），虚拟计划器 120 的生成部 121 执行生成 GXQuery 的 GXQuery 生成处理（步骤 S308），GXQuer 输出满足所输入的 XQuery 的结果 XML。GXQuery 生成处理将在后面详细叙述。

[0122] 接着，虚拟计划器 120 将 GXQuery 的 doc() 函数改写为 uri_m（步骤 S309）。资源分配部 103 确保用于保存 GXQuery 的执行结果的资源 uri_g（步骤 S310）。虚拟计划器 120 将 uri_g 作为作为 GET 响应向客户端返回（步骤 S311）。

[0123] XQuery 处理器 102 执行 GXQuery（步骤 S312）。XQuery 处理器 102 将作为 GXQuery 的执行结果的结果 XML 代入资源 uri_g（步骤 S313）。

[0124] 如后文所述，客户端 10 向中央服务器 100 发送指定了 uri_g 的 get 方法。请求受理部 110 受理了该 get 方法的执行请求后，生成请求处理线程（图 5 的步骤 S207），用于等待针对资源 uri_g 的结果 XML 的代入。在步骤 S313 中，结果 XML 被代入资源 uri_g 后，在

该请求处理线程中向客户端 10 返回结果 XML 作为 GET 响应。

[0125] 接着详细说明步骤 S301 的 DXQuery 生成处理。图 8 是表示 DXQuery 生成处理的一个实例的流程图。在 DXQuery 生成处理中，与 XQueryX 相同，其中心是将 XQuery 看作树结构，从 doc() 函数开始执行节点遍历 (traverse) 处理。此外，所谓的 XQueryX 指的是能够将 XQuery 表达式描述为 XML 语法的规范。

[0126] 遍历的规则如下。

[0127] (1) 探索 doc() 函数以下的路径表达式。

[0128] (2) 探索相同 doc() 函数以下的路径表达式相互之间的比较表达式。

[0129] 返回值的规则如下。

[0130] (1) 将返回值变换为以下的由 <rec> 包围的形式的 XML（以下称为 REC 格式）后返回。

[0131] ＜ rec ＞＜ col ? ＞... ＜ /col ? ＞＜ /rec ＞

[0132] (2) ＜ col ? ＞用于分隔变量的值。

[0133] 首先，生成部 121 将所输入的 XQuery 进行标准化（步骤 S401）。在标准化过程中，生成部 121 执行将谓语分句展开为 FLWOR 句法结构的处理，并执行以 let 语句将 return 语句约束为由变量和标签组成的形式的处理等。

[0134] 接着，生成部 121 对所输入的 XQuery 中出现的 doc() 函数进行标记（检测）（步骤 S402）。此后则针对所出现的每个 doc() 函数重复执行从步骤 S403 到步骤 S415 的处理的循环。即，针对每个 doc() 函数生成 DXQuery。基本算法是从 doc() 函数开始边标记边求出经由路径或常数能够到达的范围。

[0135] 生成部 121 判断所标记的部分 (doc() 函数）是否包含在 FLWOR 语法的某个分句中，并对处理进行分配。即，生成部 121 判断所标记的部分是否包含在 for 语句中（步骤 S403）。在包含在 for 语句中的情况下（步骤 S403 :是），生成部 121 将由所标记的变量、常数、函数和路径构成的部分重新进行标记（步骤 S404）。即，生成部 121 以所标记的 doc() 函数为起点探索路径，对已到达的部分重新进行标记。生成部 121 将标记部分作为 DXQuery 的 for 语句输出（步骤 S405）。

[0136] 在不包含在 for 语句中的情况下（步骤 S403 :否），生成部 121 判断所标记的部分是否包含在 let 语句中（步骤 S406）。在包含在 let 语句中的情况下（步骤 S406 :是），生成部 121 将由所标记的变量、常数、函数和路径构成的部分重新进行标记（步骤 S407）。即，生成部 121 以所标记的 doc() 函数为起点探索路径，对已到达的部分重新进行标记。生成部 121 将标记部分作为 DXQuery 的 let 语句输出（步骤 S408）。

[0137] 在不包含在 let 语句中的情况下（步骤 S406 :否），生成部 121 判断所标记的部分是否包含在 order by 语句中（步骤 S409）。在包含在 order by 语句中的情况下（步骤 S409 :是），跳转到步骤 S415。

[0138] 在不包含在 order by 语句中的情况下（步骤 S409 :否），生成部 121 判断所标记的部分是否包含在 where 语句中（步骤 S410）。在包含在 where 语句中的情况下（步骤 S410 :是），生成部 121 将由所标记的变量、常数、函数和路径构成的部分重新进行标记（步骤 S411）。生成部 121 将标记部分作为 DXQuery 的 where 语句输出（步骤 S412）。

[0139] 在不包含在 where 语句中的情况下（步骤 S410 :否），生成部 121 判断所标记的部

分是否包含在 return 语句中（步骤 S413）。在包含在 return 语句中的情况下（步骤 S413：是），生成部 121 以“return<rec>{X} ＜/rec ＞”的形式输出 DXQuery 的 return 语句（步骤 S414）。在 X 的部分输出以下的 (1) 和 (2) 的形式的语句。

[0140] (1)：

[0141] ＜col0 ＞

[0142] {$ 变量 }

[0143] ＜/col0 ＞

[0144] (2)：

[0145] for$ 哑变量 in$ 变量

[0146] return

[0147] ＜col1 ＞

[0148] {$ 哑变量 }

[0149] ＜/col1 ＞

[0150] 在步骤 S413 中判断为不包含在 return 语句中的情况下（步骤 S413：否），跳转到步骤 S415。在步骤 S415 中，生成部 121 判断是否存在从标记部分能够到达的语句，即包含标记部分中所含的变量、常数、函数、路径等的其他语句（步骤 S415）。当存在时（步骤 S415：是），针对该语句重复执行步骤 S403 以后的处理。当不存在时（步骤 S415：否），跳转到步骤 S416。

[0151] 在步骤 S416 中，生成部 121 判断是否已处理了所输入的 XQuery 中出现的所有 doc() 函数（步骤 S416）。在尚未处理的情况下（步骤 S416：否），对接下来出现的 doc() 函数进行标记，重复进行处理（步骤 S402）。在已经处理的情况下（步骤 S416：是），结束 DXQuery 生成处理。

[0152] 接着详细说明步骤 S308 的 GXQuery 生成处理。图 9 是表示 GXQuery 生成处理的一个实例的流程图。在 GXQuery 生成处理中，使用由 DXQuery 生成处理标记的 XQuery 进行处理。在 GXQuery 生成处理中，主要针对所标记的部分，根据先前生成的 DXQuery 计算出用于提取相应数据的 for 语句或 let 语句，以该语句进行置换。

[0153] 首先，生成部 121 获取所输入的 XQuery 中被标记的部分（标记部分）（步骤 S501）。生成部 121 判断所标记的部分是否包含 for 语句（步骤 S502）。在包含 for 语句的情况下（步骤 S502：是），生成部 121 将所标记的部分置换为从 DXQuery 中提取相应数据的 for 语句并输出（步骤 S503）。生成部 121 输出例如如下 (1) 所示形式的 for 语句。

[0154] (1)：

[0155] for$ 哑变量 in

[0156] doc([uri])/root/rec

[0157] for$ 变量 in

[0158] $ 哑变量 /col/*

[0159] 在不包含 for 语句的情况下（步骤 S502：否），生成部 121 判断所标记的部分是否包含 let 语句（步骤 S504）。在包含 let 语句的情况下（步骤 S504：是），生成部 121 将所标记的部分置换为从 DXQuery 中提取相应数据的 let 语句并输出（步骤 S505）。生成部 121 输出例如如下 (2) 所示形式的 let 语句。

[0160] (2):

[0161] let$ 变量 : =

[0162] for$ 哑变量 1in

[0163] doc([uri])/root/rec

[0164] for$ 哑变量 2in

[0165] $ 哑变量 1/col/*

[0166] return$ 哑变量 2

[0167] 在不包含 let 语句的情况下 (步骤 S504 :否),生成部 121 判断所标记的部分是否包含 order by 语句 (步骤 S506)。在包含 order by 语句的情况下 (步骤 S506 :是),生成部 121 原样输出 order by 语句 (步骤 S507)。

[0168] 在不包含 order by 语句的情况下 (步骤 S506 :否),生成部 121 判断所标记的部分是否包含 where 语句 (步骤 S508)。在包含 where 语句的情况下 (步骤 S508 :是),生成部 121 将所标记的部分置换为从 DXQuery 中提取相应数据的 let 语句并输出 (步骤 S509)。生成部 121 输出例如如下 (3) 所示形式的 let 语句。

[0169] (3):

[0170] let$ 哑变量 : = in

[0171] $ 变量 /col/*

[0172] where 哑变量比较表达式

[0173] 在不包含 where 语句的情况下 (步骤 S508 :否),生成部 121 判断所标记的部分是否包含 return 语句 (步骤 S510)。在包含 return 语句的情况下 (步骤 S510 :是),生成部 121 原样输出 return 语句 (步骤 S511)。

[0174] 步骤 S503、步骤 S505、步骤 S507、步骤 S509 和步骤 S511 执行后,或者在步骤 S510 中判断为不包含 return 语句的情况下 (步骤 S510 :否),生成部 121 判断是否已经处理了所有的标记部分 (步骤 S512)。在尚未全部处理的情况下 (步骤 S512 :否),获取一个尚未处理的标记部分,重复进行处理 (步骤 S501)。在已经处理的情况下 (步骤 S512 :是),结束 GXQuery 生成处理。

[0175] 接着说明 DB 服务器 20 相应于虚拟计划器 120 的 DXQuery 的执行请求所执行的 XQuery 处理。此外,当从客户端 10 不是请求了分布式 XQuery 处理 (gquery)、而是请求了 XQuery 处理 (query) 的情况下,XQuery 处理器 102 所执行的 XQuery 处理也按照相同的步骤执行。图 10 是表示 XQuery 处理的一个实例的流程图。

[0176] DB 服务器 20 针对所请求的 DXQuery 确保资源 uri_d(步骤 S601)。DB 服务器 20 向中央服务器 100 返回 uri_d 作为 GET 响应 (步骤 S602)。DB 服务器 20 利用 XQuery 处理器执行 DXQuery(步骤 S603)。DB 服务器 20 将作为 DXQuery 的执行结果的结果 XML 代入资源 uri_d(步骤 S604)。在 XQuery 处理器 102 执行 XQuery 处理的情况下,生成用于等待针对资源 uri_d 的结果 XML 的代入的请求处理线程。

[0177] 接着说明利用 get 方法获取资源的数据的获取处理。图 11 是表示获取处理的一个实例的流程图。

[0178] 请求受理部 110 获取与 get 方法指定的针对资源 uri 的数据 (步骤 S701)。在无法获取的情况下,生成用于等待数据代入的请求处理线程,进入代入事件等待状态。一旦数

据被代入资源 uri，请求受理部 110 就将数据作为 GET 响应返回给 get 方法的请求方（步骤 S702）。

[0179] 接着说明利用 merge 方法合并资源的数据的合并处理。图 12 是表示合并处理的一个实例的流程图。

[0180] 请求受理部 110 确保用于保存合并处理的结果的资源 uri_m（步骤 S801）。请求受理部 110 将所确保的资源的 URI、即 uri_m 作为 GET 响应返回到请求方（步骤 S802）。请求受理部 110 对由 merge 方法请求了合并的资源集合中所包含的各资源执行合并处理（步骤 S803）。

[0181] 此外，图 12 的“U uri_d”表示被请求了合并的资源 uri_d 的集合。另外，所谓的合并指的是将各资源的数据串联地连接起来。

[0182] 请求受理部 110 将合并得到的 XML 数据、即结果 XML 代入 uri_m。此外，在等待其他线程等对 uri_m 进行结果 XML 的代入的情况下，生成用于等待结果 XML 的代入的请求处理线程。

[0183] 图 13 是按照时间序列展示出客户端 10 和多个服务器（中央服务器 100、DB 服务器 20a、DB 服务器 20b）的相互作用的状况的序列图。在图 13 中，假定时间进程为从左至右。

[0184] 首先，客户端 10 向中央服务器 100 以 gquery 方法请求分布式 XQuery 处理（步骤 S901）。

[0185] 中央服务器 100 向 DB 服务器 20a 以 query 方法请求 XQuery 处理（步骤 S902）。此时的 query 方法使用生成部 121 所生成的 DXQuery 作为参数。中央服务器 100 从 DB 服务器 20a 获取 uri_d1（步骤 S903），该 uri_d1 表示 query 方法的结果 XML 的资源的 uri。

[0186] 中央服务器 100 向 DB 服务器 20b 以 query 方法请求 XQuery 处理（步骤 S904）。此时的 query 方法使用生成部 121 所生成的 DXQuery 作为参数。中央服务器 100 从 DB 服务器 20b 获取 uri_d2（步骤 S905），该 uri_d2 表示 query 方法的结果 XML 的资源的 uri。

[0187] 中央服务器 100 以 merge 方法向自身服务器（中央服务器 100）请求将资源 uri_d1 和资源 uri_d2 合并（步骤 S906）。中央服务器 100 从自身服务器（中央服务器 100）获取表示 merge 方法的结果 XML 的资源的 URI 的 uri_m（步骤 S907）。

[0188] 中央服务器 100 以 get 方法向 DB 服务器 20a 请求 uri_d1 的结果 XML 的获取（步骤 S908）。中央服务器 100 以 get 方法向 DB 服务器 20b 请求 uri_d2 的结果 XML 的获取（步骤 S909）。

[0189] 中央服务器 100 将 uri_g 作为对 gquery 方法的 GET 响应返回到客户端 10（步骤 S910）。

[0190] 客户端 10 以 get 方法向中央服务器 100 请求 uri_g 的结果 XML 的获取（步骤 S911）。

[0191] 中央服务器 100 以 get 方法向自身服务器（中央服务器 100）请求针对在步骤 S907 中所获取的 uri_m 的结果 XML 的获取（步骤 S912）。

[0192] 另一方面，在生成了表示中央服务器 100 向 DB 服务器 20a 请求的 query 方法的执行结果的结果 XML 的情况下，DB 服务器 20a 将作为结果 XML 的 xml_d1 代入资源 uri_d1 中（步骤 S913）。

[0193] 同样地，在生成了表示中央服务器 100 向 DB 服务器 20b 请求的 query 方法的执行结果的结果 XML 的情况下，DB 服务器 20b 将作为结果 XML 的 xml_d2 代入资源 uri_d2 中（步骤 S914）。

[0194] 在生成了以 merge 方法向自身服务器（中央服务器 100）请求的 uri_d1 和 uri_d2 的合并结果、即结果 XML 的情况下，中央服务器 100 将作为结果 XML 的 xml_m 代入资源 uri_m（步骤 S915）。

[0195] 在生成了客户端 10 以 get 方法向中央服务器 100 请求的 uri_g 的结果 XML 的情况下，中央服务器 100 将结果 XML 代入 uri_g（步骤 S916）。

[0196] 此外，如图 13 所示，针对所请求的 query 方法，DB 服务器 20 在获得执行结果之前将用于通知保存执行结果的资源的响应返回（步骤 S903、步骤 S905 等）。即，针对 query 方法，DB 服务器 20 并不是在获得 XQuery 处理的执行结果之后返回用于通知执行结果的响应，而是必须具备在获得执行结果之前返回用于通知保存执行结果的资源的响应的功能。当 DB 服务器 20 不具备该功能的情况下，也可以采用在中央服务器 100 内构建能够代替该功能的功能的方式。

[0197] 在图 13 的右侧表示出在上述序列中处理的 XQuery、资源和 XML 的关系。即，在 DB 服务器 20a 和 20b 中分别执行 DXQuery，执行结果保存到资源 uri_d1 和资源 uri_d2 中。此外，嵌入了由资源 uri_d1 和资源 uri_d2 合并而成的资源 uri_m 的 GXQuery 的执行结果保存在资源 uri_g 中，该资源 uri_g 被发送到客户端 10。一旦获得了 GXQuery 的执行结果、即结果 XML，该结果 XML 就被发送到客户端 10。

[0198] 接着说明检索处理的具体实例。图 14 是表示从客户端输入的 XQuery 的一个实例的图。图 14 的 XQuery 表示“提取 column3 中包含‘神奈川’的所有 row”。

[0199] 图 15 是表示图 2 的 DB21a 中存储的数据的一个实例的图。如图 15 所示，DB21a 表示出存储着事务所名 (column1)、住所 (column2)、都道府县 (column3) 的事务所数据库的实例。在图 15 中示例出 4 个 row（事务所）的数据。

[0200] 图 16 是表示根据图 14 的 XQuery 生成的 DXQuery 的一个实例的图。基本结构与图 14 相同。图 16 的 DXQuery 表示“以 REC 格式提取 column3 中包含‘神奈川’的所有 row”。

[0201] 图 17 是表示图 16 的 DXQuery 的执行结果、即结果 XML 的一个实例的图。图 17 表示出利用 DXQuery 从图 15 所示的 DB21a 检索到的结果 XML(xml_d1) 的实例。

[0202] 图 18 是表示根据图 14 的 XQuery 生成的 GXQuery 的一个实例的图。图 18 的 GXQuery 由以下 2 个部分构成。

[0203] (1) 读入 DXQuery 的结果 XML 并提取各 rec 的部分

[0204] (2) 提取 rec 的 col ？ 的值的部分

[0205] 如图 16 和图 18 所示，GXQuery+DXQuery ＝所输入的 XQuery。

[0206] 这里进一步说明根据图 14 的 XQuery 生成图 16 的 DXQuery 和图 18 的 GXQuery 的处理。

[0207] 首先，从图 14 的 XQuery 中提取 doc() 函数“doc（“database.XML”）”（图 8 的步骤 S402）。

[0208] 针对 for 语句，从 doc() 函数直到“//row”，该分句的主体被标记，因此一直标记到“$x”。

[0209] 针对 where 语句，“$x”已经被标记了。“$x//column3”也被标记。“神奈川”是常数。因此，where 语句全部被标记。其结果，生成了如图 16 所示的一个 DXQuery。

[0210] 接着生成 GXQuery。在 DXQuery 中，for 语句和 where 语句被标记，因此，从 DXQuery 中提取相应数据的 for 语句就变成了如图 18 所示的样子。

[0211] 图 19 是表示图 18 的 GXQuery 的执行结果、即结果 XML 的一个实例的图。如图 19 所示，作为 GXQuery 执行结果的结果 XML 被保存在资源 xml_g 中。

[0212] 下面以比图 14 更复杂的 XQuery 为例进一步说明。图 20 是表示从客户端输入的 XQuery 的另一个实例的图。图 20 的 XQuery 表示“按照每个 row 的 column3 对 row 进行合计”。例如，针对图 15 这样的数据，其意味着“以都道府县为单位计算事务所的总数”。

[0213] 这里对几个内部函数加以说明。

[0214] (1)distinct-values :从所输入的序列中提取值不相同的元素序列。

[0215] (2)count :返回所输入的序列的元素数。

[0216] 图 21 是表示针对图 20 的 DXQuery 生成的示意图。

[0217] (1) 探索 doc() 函数以下的路径表达式。

[0218] (2) 探索相同 doc() 函数以下的路径表达式相互之间的比较表达式。

[0219] 按照这种遍历规则，就会生成 2 个 DXQuery(DXQuery1、DXQuery2)。

[0220] 图 22 是表示 DXQuery1 的一个实例的图。图 23 是表示图 22 的 DXQuery1 的执行结果、即结果 XML 的一个实例的图。图 24 是表示 DXQuery2 的一个实例的图。图 25 是表示图 24 的 DXQuery2 的执行结果、即结果 XML 的一个实例的图。图 26 是表示根据图 20 的 XQuery 生成的 GXQuery 的一个实例的图。

[0221] 如图 22、图 24 和图 26 所示，GXQuery+DXQuery(DXQuery1、DXQuery2) ＝所输入的 XQuery。图 27 是表示图 26 的 GXQuery 的执行结果、即结果 XML 的一个实例的图。如图 27 所示，“以都道府县为单位计算事务所的总数”的结果就作为结果 XML 被得到。

[0222] 按照这种方式，在第一实施方式的检索装置中，不使用 XRPC 就能够实现分布式 XQuery 处理。因此，可以得到下述效果。

[0223] (1)XQuery 透明 :用户不需要在 XQuery 内显式记述特殊的语言扩展。

[0224] (2) 异质综合 :在通常的网络和通信协议基础上，RDBMS、Web 服务等异质数据库也能够连接起来构成虚拟 XML 数据库。利用有限的定义，地图信息服务、天气信息服务等不支持 XQuery 的非 XML-DBMS 的 Web 服务也可以成为虚拟 XML 数据库的构成元素。

[0225] (3) 高速性 :即使 XQuery 中含有 for 循环，网络上也不会产生与该循环次数相当的消息流量。不会像 RPC 之类的函数执行那样串行化，而是能够由多个服务器并行处理。在通常的分布式数据库中进行结合运算时，作为使通信负载减少到最低程度的方法可以简单地使用半联接法 (semi-joins)。

[0226] 第二实施方式的检索装置对 DXQuery 进行变换（泛化），以满足 DB 服务器的检索能力（查询处理能力），再根据变换后的 DXQuery 的检索结果，生成变换前的 DXQuery 的检索结果。由此能够实现与 DB 服务器的查询处理能力相适应的高精度检索。

[0227] 图 28 是表示第二实施方式中的中央服务器 200 的结构实例的框图。中央服务器 200 包括 :请求受理部 110、虚拟计划器 220、XQuery 处理器 102、资源分配部 103、服务器通知检测部 104 和泛化处理器 230。

[0228] 在第二实施方式中增加了虚拟计划器 220 的功能和泛化处理器 230，这一点与第一实施方式不同。其他的结构和功能与第一实施方式的中央服务器 100 的框图、即图 4 相同，因此标注相同标号，这里省略其说明。

[0229] 虚拟计划器 220 向泛化处理器 230 请求所生成的 DXQuery 的泛化，并利用发送部 222 将泛化处理后的 DXQuery（以下称为 DXQuery'）发送到 DB 服务器 20，这一点与第一实施方式的虚拟计划器 120 不同。

[0230] 泛化处理器 230 包括变换部 231 和生成部 232（第二生成部）。变换部 231 将虚拟计划器 220 的生成部 121 所生成的 DXQuery 变换为匹配于 DB 服务器 20 的查询处理能力而泛化了的 DXQuery'。生成部 232 生成验证 DXQuery'的检索结果的 XQuery、即 VXQuery（生成请求）。VXQuery 是以 DXQuery' 的检索结果产生的、验证 DB 服务器 20 的查询处理能力的不足之处并对不足之处进行补充从而使所生成的检索结果与使用变换前的 DXQuery 的情况下相同的 XQuery。

[0231] 图 29 是表示利用虚拟计划器 220 和泛化处理器 230 执行的分布式 XQuery 处理的一个实例的流程图。在本实施方式的分布式 XQuery 处理中生成对将 DXQuery 泛化后的 DXQuery'、VXQuery 和 VXQuery 的结果进行综合的 GXQuery。

[0232] 在步骤 S1001 至步骤 S1003 中执行与第一实施方式的中央服务器 100 中的步骤 S301 至步骤 S303 相同的处理，因此省略其说明。

[0233] 在步骤 S1004 中执行根据 DXQuery 生成 DXQuery' 和 VXQuery 的泛化处理。在后文叙述泛化处理的详细情况。

[0234] 接着，虚拟计划器 220 的发送部 222 向所选择的 DB 服务器 20 发送泛化处理后的 DXQuery' 的执行请求（步骤 S1005）。接收到执行请求的 DB 服务器 20 将保存着 DXQuery' 的执行结果（结果 XML）的资源的 URI(uri_d) 发送到中央服务器 200。由此，虚拟计划器 220 获取了保存着结果 XML 的资源的 URI，即 uri_d。

[0235] 接着，虚拟计划器 220 利用所获取的 uri_d 改写 VXQuery 的 doc() 函数（步骤 S1006）。虚拟计划器 220 对自身服务器（中央服务器 200）指定 query 方法，请求执行 VXQuery（步骤 S1007），获取保存着 VXQuery 的结果 XML 的 uri_v。

[0236] 步骤 S1008 与图 7 的步骤 S305 相同，因此省略其说明。步骤 S1009 使用资源 uri_v 而不是资源 uri_d，这一点与图 7 的步骤 S306 不同。即，虚拟计划器 220 对自身服务器（中央服务器 200）请求 merge 方法（步骤 S1009），该 merge 方法用于合并分别保存在多个资源 uri_v 中的数据。

[0237] 在步骤 S1010 至步骤 S1016 中执行与第一实施方式的中央服务器 100 中的步骤 S307 至步骤 S313 相同的处理，因此省略其说明。

[0238] 接着说明步骤 S1004 的泛化处理。图 30 是表示泛化处理的一个实例的流程图。

[0239] 首先，泛化处理器 230 从各 DB 服务器 20 获取分布式服务器定义（步骤 S1101）。所谓的分布式服务器定义指的是表示 DB 服务器的查询处理能力的信息。图 31 是表示分布式服务器定义的一个实例的图。“SERVICE http://example.com/？ key ＝％ 1”表示能够处理赋予“key”的参数。“XQuery for$xin ～”表示赋予“key”的参数置换为“contains($x，“％ 1”)”。

[0240] 返回图 30，泛化处理器 230 将所选择的 DXQuery 和分布式服务器定义的 XQuery 模

式进行对照。在不吻合的情况下（步骤 S1103 :否），变换部 231 对 DXQuery 进行泛化（步骤 S1104），然后返回步骤 S1102 而重复处理。

[0241] 在 DXQuery 的泛化中，变换部 231 执行例如以下处理 :(1) 路径的省略 ;(2)OR 条件的展开 ;(3) 将标签名变换为“*”(XML 数据中所包含的元素名称的变换）等。

[0242] 在 DXQuery 和分布式服务器定义的 XQuery 模式吻合的情况下（步骤 S1103 :是），变换部 231 将泛化后 DXQuery（在未经泛化的情况下则是所选择的 DXQuery）作为 DXQuery’输出（步骤 S1105）。接着，生成部 232 根据原来的 DXQuery 生成 VXQuery（步骤 S1106），结束泛化处理。

[0243] 图 32 是表示在输入了图 14 的 XQuery 和图 31 的分布式服务器定义时利用泛化处理所输出的 DXQuery’的一个实例的图。与图 16 的 DXQuery 进行比较即可发现，图 32 的 DXQuery’省略了“row”这一标签名和“$x//column3”这一路径。这样，通过对 DXQuery 重复泛化操作而生成了与 DB 服务器 20 的查询处理能力相吻合的 DXQuery’。

[0244] 图 33 是表示在输入了图 14 的 XQuery 和图 31 的分布式服务器定义时经过泛化处理后所输出的 VXQuery 的一个实例的图。

[0245] 图 34 是表示在第二实施方式中所处理的 XQuery、资源和 XML 的关系的一个实例的图。在图 34 中示出的是仅针对发送给 DB 服务器 20b 的 DXQuery 执行了泛化处理的实例。在 DB 服务器 20a 和 20b 中分别执行 DXQuery 和 DXQuery’，执行结果保存到资源 uri_d1 和资源 uri_d2 中。对资源 uri_d2 执行 VXQuery，输出执行结果 uri_v2。嵌入了由资源 uri_d1 和资源 uri_v2 合并而成的资源 uri_m 的 GXQuery 的执行结果保存到资源 uri_g 中，该资源 uri_g 被发送到客户端 10。一旦获得了 GXQuery 的执行结果、即结果 XML，该结果 XML 就被发送到客户端 10。

[0246] 这里进一步说明根据图 14 的 XQuery 和图 31 的分布式服务器定义生成图 32 的 DXQuery’的过程。

[0247] 在图 31 的分布式服务器定义中记述着“该 DB 服务器在获取数据时只能使用关键字 (key) 进行检索 (contains)”这一与 DB 服务器 20 的查询处理能力相关的声明。

[0248] 泛化处理器 230 对图 14 的 XQuery 和图 31 的分布式服务器定义进行比较，列举出语法上的差异（图 30 的步骤 S1102）。泛化处理器 230 组合使用例如 Yacc&Lex 等语句分析和语法分析工具在存储器中展开 XQuery 的语法树，对 2 个存储器中的语法树进行比较。

[0249] 其结果是，泛化处理器 230 能够检测出“doc()//row”和“doc()/*”、以及“$x//column3”和“$x”的部分存在差异。这 2 个差异与 XQuery 的路径有关。因此，变换部 231 应用“路径的省略”这一泛化操作。泛化操作可以使用例如基于规则库系统技术来实现。即，利用由“if 存在路径上的差异 then 省略路径”这样的 if 语句和 then 语句这 2 部分所构成的规则来表达泛化操作，使用推理机重复应用这种规则集合，直到出现停止条件为止。在这种情况下的停止条件指的是不再有 DXQuery 与分布式服务器定义的差异。

[0250] 例如，就图 14 的 XQuery 而言，只要执行了 (1) 从“doc()//row”到“doc()/*”的路径省略、(2) 从“$x//column3”到“$x”的路径省略这 2 个泛化操作，就能够满足停止条件。

[0251] 其后，以“return ＜ rec ＞ {...} ＜ /rec ＞”的形式输出 return 语句，即可生成图 32 的 DXQuery。

[0252] 接着进一步说明根据图 14 的 XQuery 和图 31 的分布式服务器定义生成图 33 的 VXQuery 的过程。

[0253] 作为 VXQuery 生成对在上述泛化操作中被泛化的差异部分“doc()//row”和“doc()/*”以及“$x//column3”和“$x”进行检查的特殊的 XQuery。

[0254] 例如，生成部 232 将包含差异部分的语句“for$x in doc()//row”和“where contains($x//column3, “神奈川”)”嵌入到如下 (A) 所示的成为基础的 XQuery 中，由此生成 VXQuery。这时，将根据需要嵌入的差异部分内的变量改写为成为基础的 XQuery 内的变量。可以事先设定好作为基础的 XQuery。

[0255] (A):

[0256] For$_0 in doc([uri_d1])/rec

[0257] For$_1 in$_0/col0/*

[0258] return

[0259] ＜rec＞

[0260] {＜col0＞{$x}＜/col0＞}

[0261] ＜/rec＞

[0262] 嵌入后的 VXQuery 如下 (B) 所示。

[0263] (B):

[0264] For$_0 in doc([uri_d1])/rec

[0265] for$_1 in$_0/col0/*

[0266] for$x in$_1//row

[0267] where contains($x//column3,“神奈川”)

[0268] return

[0269] ＜rec＞

[0270] {＜col0＞{$x}＜/col0＞}

[0271] ＜/rec＞

[0272] 在上述 VXQuery 中，第三行和第四行被嵌入。另外，第三行的“$x”被置换为“$_1”。

[0273] 如上述所说明，根据第一和第二实施方式，无需使用 XRPC 就能够实现 XQuery 透明的分布式 XQuery 处理。

[0274] 接着，使用图 35 说明第一或第二实施方式的检索装置（中央服务器）的硬件结构。图 35 是表示第一或第二实施方式的检索装置的硬件结构的说明图。

[0275] 第一或第二实施方式的检索装置包括 :CPU(Central Processing Unit :中央处理器)51 等控制装置 ;ROM(Read Only Memory :只读存储器)52 或 RAM(Random Access Memory :随机存取存储器)53 等存储装置 ;连接到网络上进行通信的通信 I/F54 ;HDD(Hard Disk Drive :硬盘驱动器)、CD(Compact Disc) 驱动器装置等外部存储装置 ;显示器装置等显示装置 ;键盘或鼠标等输入装置 ;和连接各部分的总线 61 ;其表现为使用了通常的计算机的硬件结构。

[0276] 在第一或第二实施方式的检索装置中执行的检索程序被以可安装形式或可执行形式的文件记录到 CD-ROM(Compact Disk Read Only Memory)、软盘 (FD)、CD-R(Compact Disk Recordable)、DVD(Digital Versatile Disk) 等计算机可读的记录介质中，以计算机

程序产品的形式提供给用户。

[0277] 另外，也可以将在第一或第二实施方式的检索装置中执行的检索程序保存到与因特网等网络相连接的计算机上，通过经由网络下载提供给用户。另外，也可以将在第一或第二实施方式的检索装置中执行的检索程序经由因特网等网络提供或发布给用户。

[0278] 另外，也可以将第一或第二实施方式的检索程序预先嵌入到 ROM 等之中提供给用户。

[0279] 在第一或第二实施方式的检索装置中执行的检索程序具有模块结构，这些模块包含上述各部分（请求受理部、虚拟计划器、XQuery 处理器、资源分配部、服务器通知检测部）；作为实际的硬件，CPU51（处理器）从上述存储介质读取检索程序并执行，由此将上述各部分加载到主存储装置中，在主存储装置中生成上述各部分。

[0280] 以上说明了本发明的若干实施方式，但这些实施方式是作为实例而呈现的，其意图并不是为了限定发明的范围。这些新实施方式可以通过其他的各种各样的方式加以实施，在不脱离发明主旨的范围内能够作出各种各样的省略、替换、变更。这些实施方式或其变形既包含在发明的范围或主旨内，也包含在权力要求书中记载的发明及其均等的范围内。

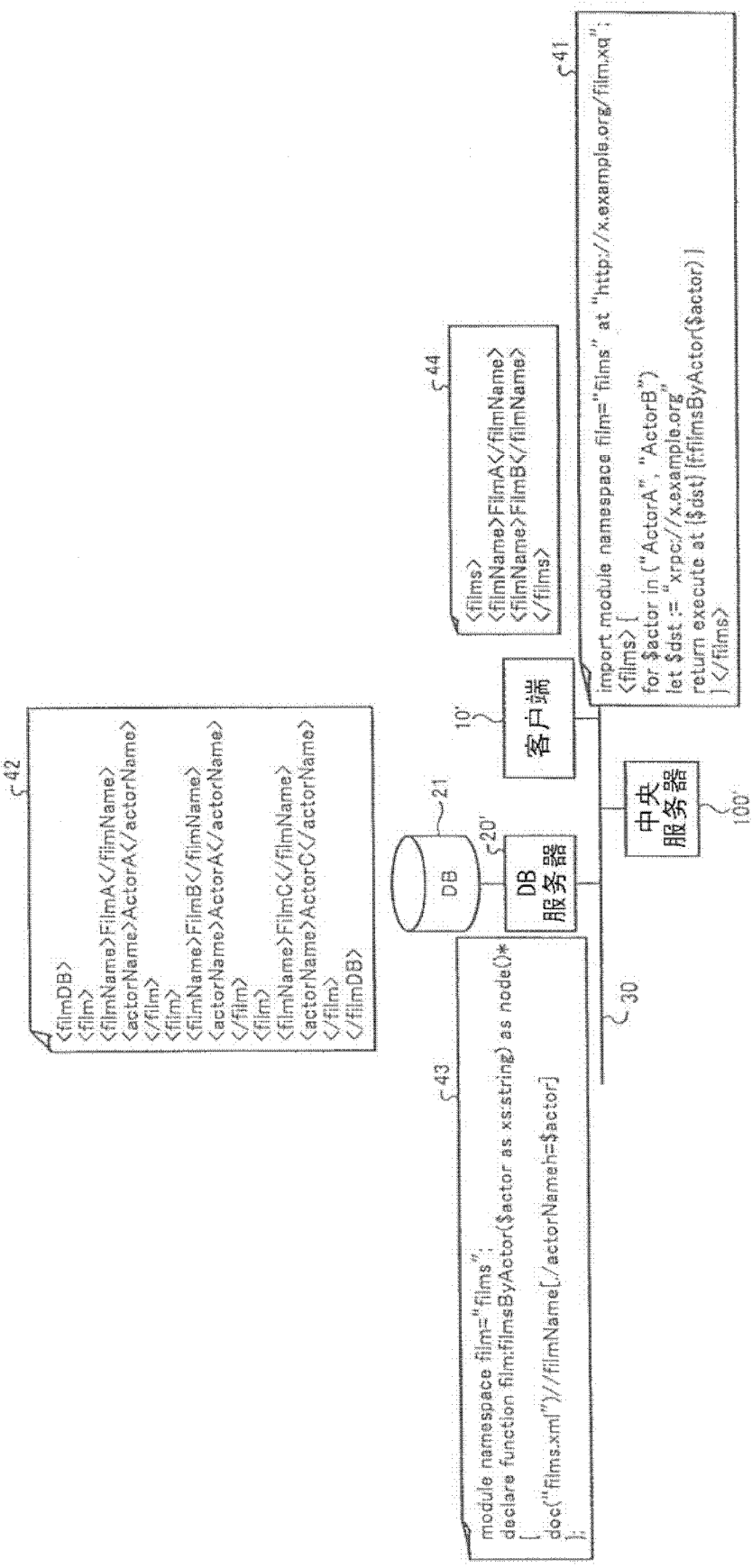

42
<filmDB>
<film>
<filmName>FilmA</filmName>
<actorName>ActorA</actorName>
</film>
<film>
<filmName>FilmB</filmName>
<actorName>ActorA</actorName>
</film>
<film>
<filmName>FilmC</filmName>
<actorName>ActorC</actorName>
</film>
</filmDB>
21
DB
20'
DB 服务器
10'
客户端
44
<films>
<filmName>FilmA</filmName>
<filmName>FilmB</filmName>
</films>
41
import module namespace films="films" at "http://x.example.org/film.xq";
<films>{
for $actor in ("ActorA", "ActorB")
let $dst := "xrpc://x.example.org"
return execute at {$dst} {f:filmsByActor($actor)}
}</films>
43
module namespace film="films";
declare function film:filmsByActor($actor as xs:string) as node()*
{
doc("films.xml")//filmName[./actorName=$actor]
};
30
中央服务器
100'

图 1

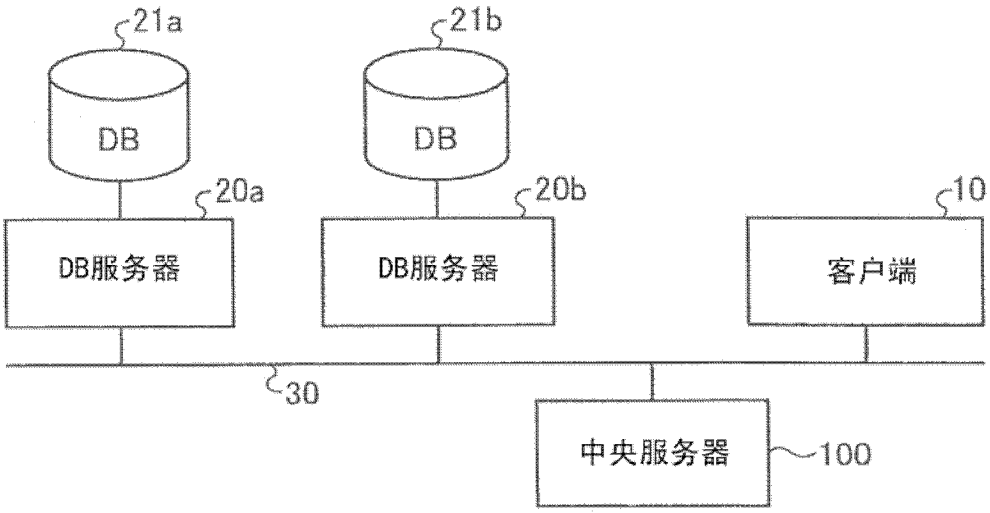

21a
21b
DB
DB
20a
20b
10
DB服务器
DB服务器
客户端
30
中央服务器
100

图 2

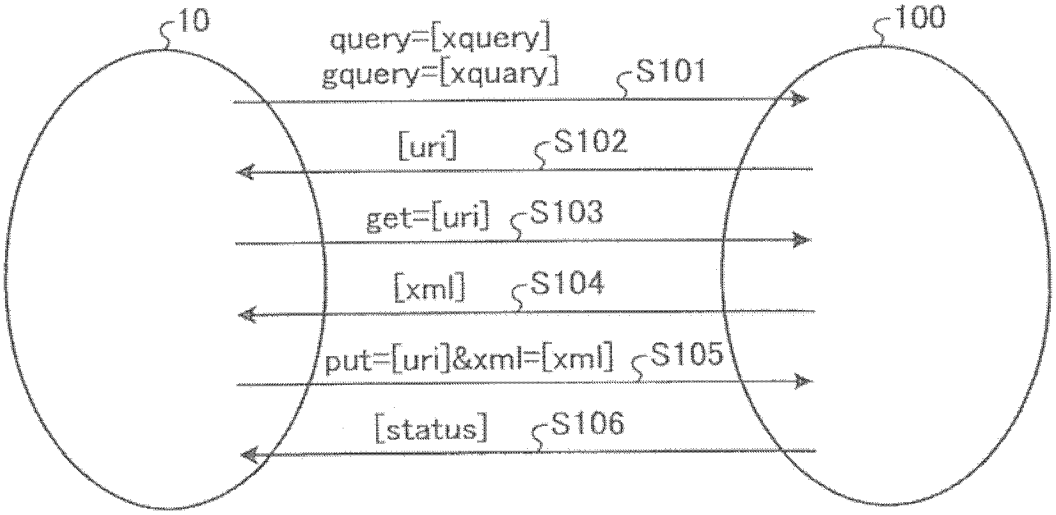

10
100
query=[xquery]
gquery=[xquary]
S101
[uri]
S102
get=[uri]
S103
[xml]
S104
put=[uri]&xml=[xml]
S105
[status]
S106

图 3

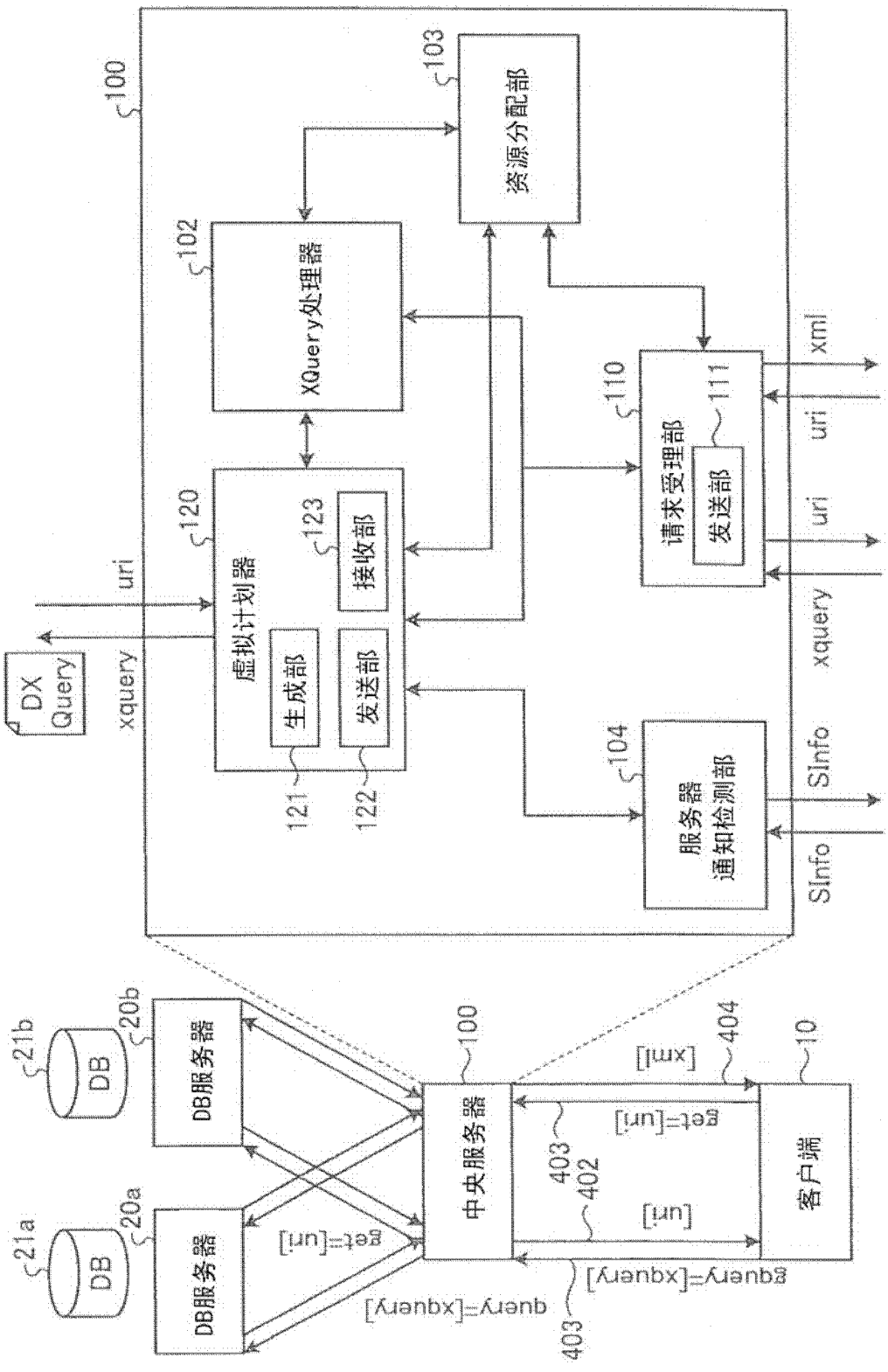

100
103
资源分配部
102
XQuery处理器
120
虚拟计划器
123
接收部
生成部
发送部
121
122
110
请求受理部
111
发送部
104
服务器
通知检测部
DX
Query
uri
xquery
xml
uri
uri
xquery
SInfo
SInfo
21b
DB
20b
DB服务器
21a
DB
20a
DB服务器
get=[uri]
query=[xquery]
100
中央服务器
403
402
403
get=[uri]
[uri]
query=[xquery]
404
[xml]
10
客户端

图 4

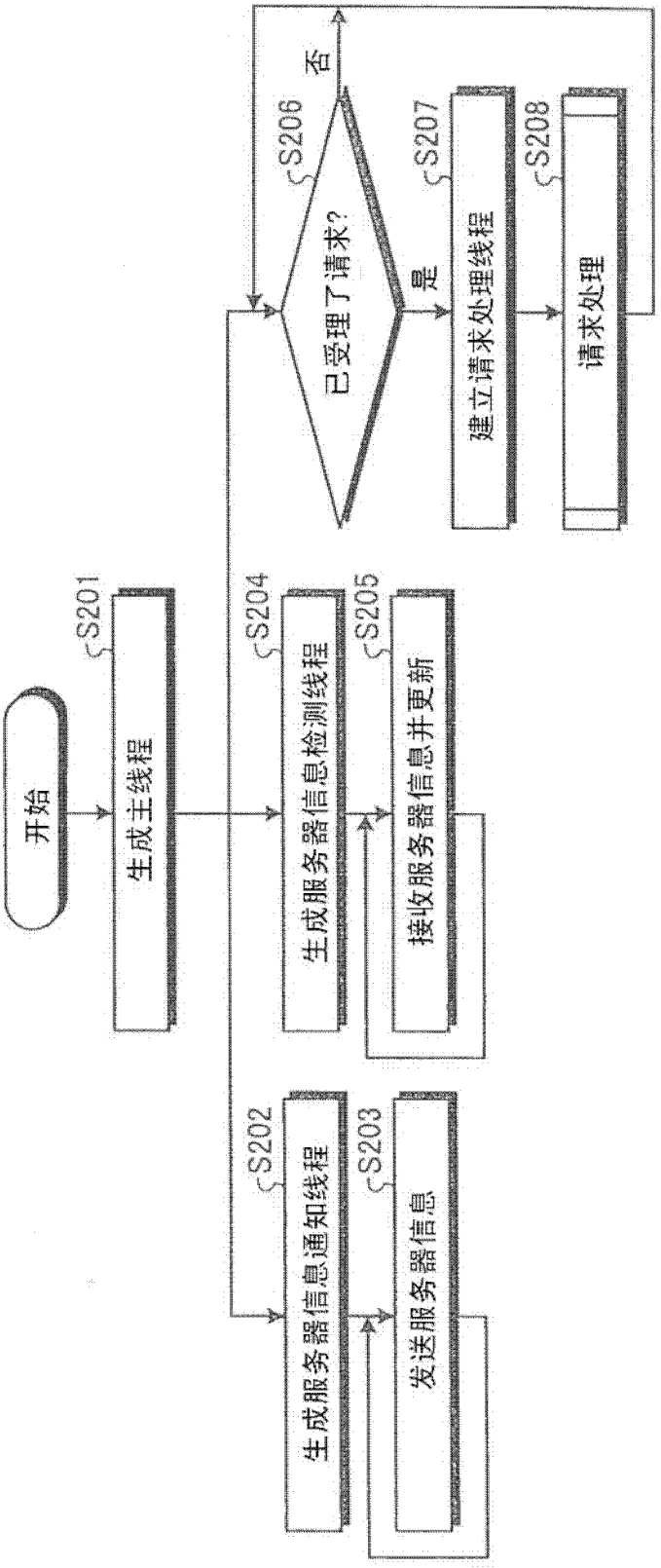

开始
S201
生成主线程
S202
生成服务器信息通知线程
S203
发送服务器信息
S204
生成服务器信息检测线程
S205
接收服务器信息并更新
S206
已受理了请求?
否
是
S207
建立请求处理线程
S208
请求处理

图 5

```
GET消息语法分析；
switch(消息类型){
case gquery:
 虚拟计划器执行；
 break;
case query:
 XQuery执行；
 break;
case post:
 资源生成；
 break;
case get:
 资源数据获取；
 break;
case put:
 将数据代入资源；
 break;
case merge:
 将数据合并代入资源；
 break;
default:
 break;
}
```

图 6

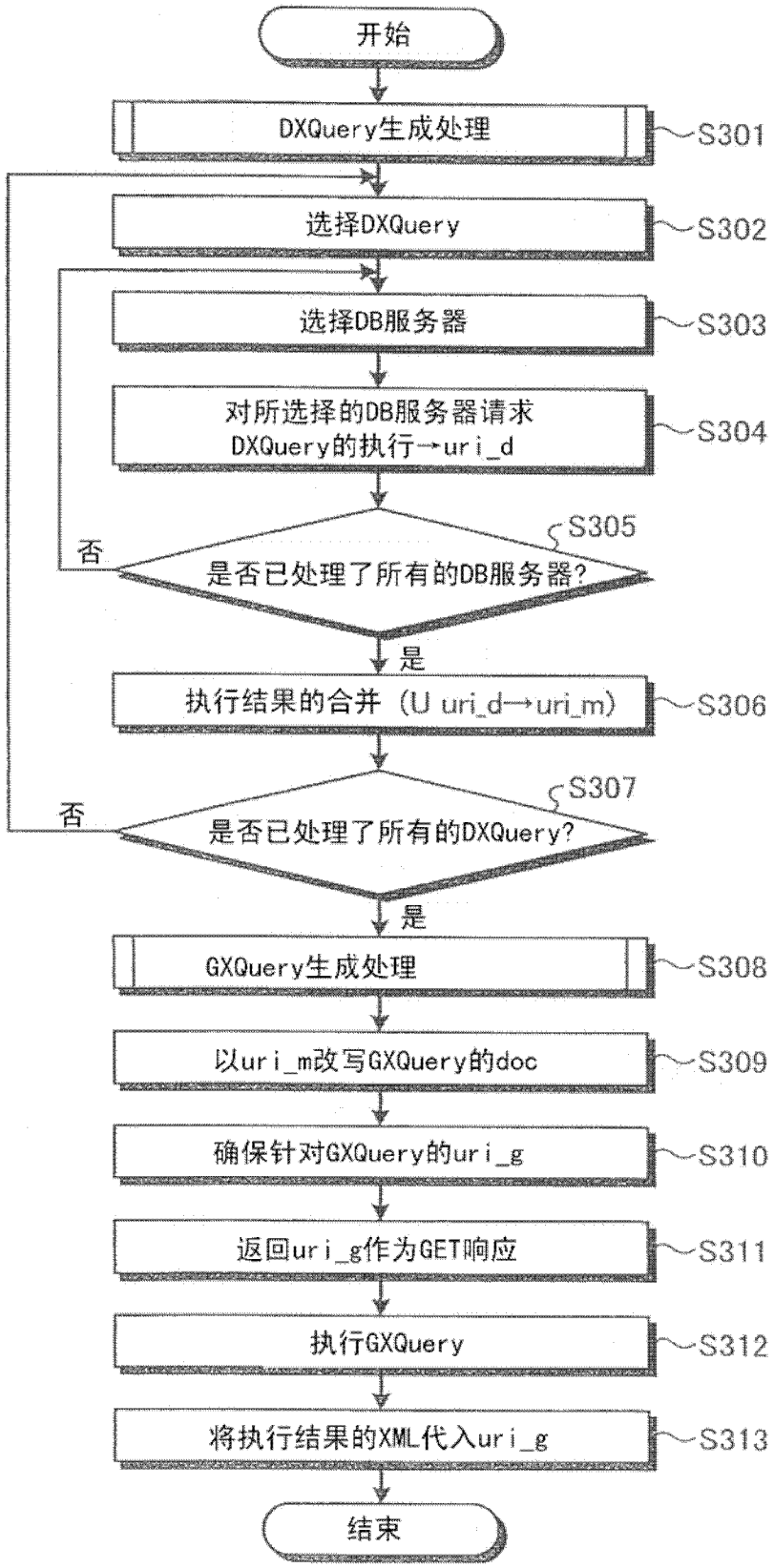

开始
DXQuery生成处理
S301
选择DXQuery
S302
选择DB服务器
S303
对所选择的DB服务器请求
DXQuery的执行→uri_d
S304
S305
否
是否已处理了所有的DB服务器?
是
执行结果的合并（U uri_d→uri_m）
S306
S307
否
是否已处理了所有的DXQuery?
是
GXQuery生成处理
S308
以uri_m改写GXQuery的doc
S309
确保针对GXQuery的uri_g
S310
返回uri_g作为GET响应
S311
执行GXQuery
S312
将执行结果的XML代入uri_g
S313
结束

图 7

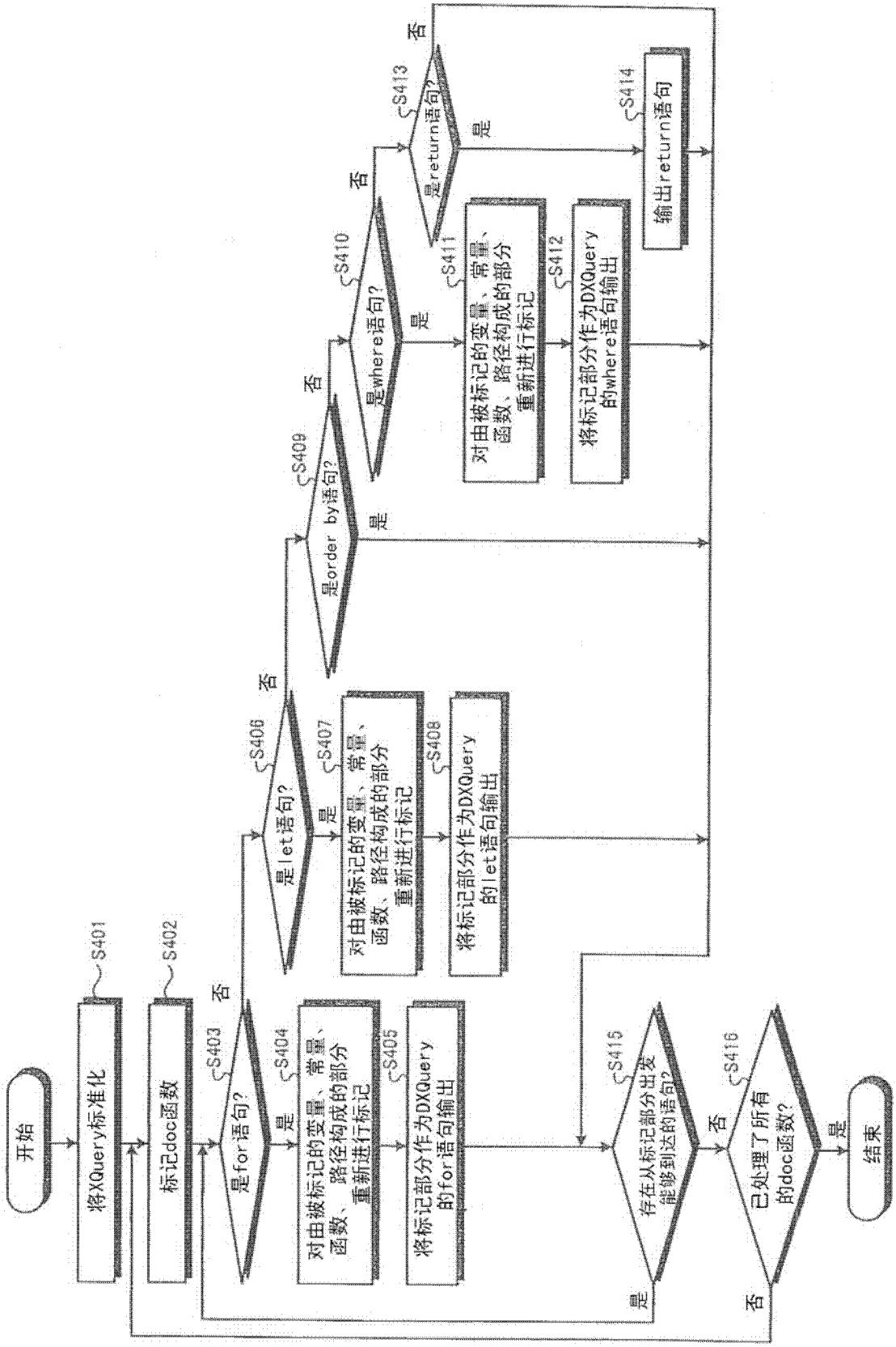

开始
S401
将XQuery标准化
S402
标记doc函数
S403
是for语句?
否
是
S404
对由被标记的变量、常量、函数、路径构成的部分重新进行标记
S405
将标记部分作为DXQuery的for语句输出
S406
是let语句?
否
是
S407
对由被标记的变量、常量、函数、路径构成的部分重新进行标记
S408
将标记部分作为DXQuery的let语句输出
S409
是order by语句?
否
是
S410
是where语句?
否
是
S411
对由被标记的变量、常量、函数、路径构成的部分重新进行标记
S412
将标记部分作为DXQuery的where语句输出
S413
是return语句?
否
是
S414
输出return语句
S415
存在从标记部分出发能够到达的语句?
否
是
S416
已处理了所有的doc函数?
否
是
结束

图 8

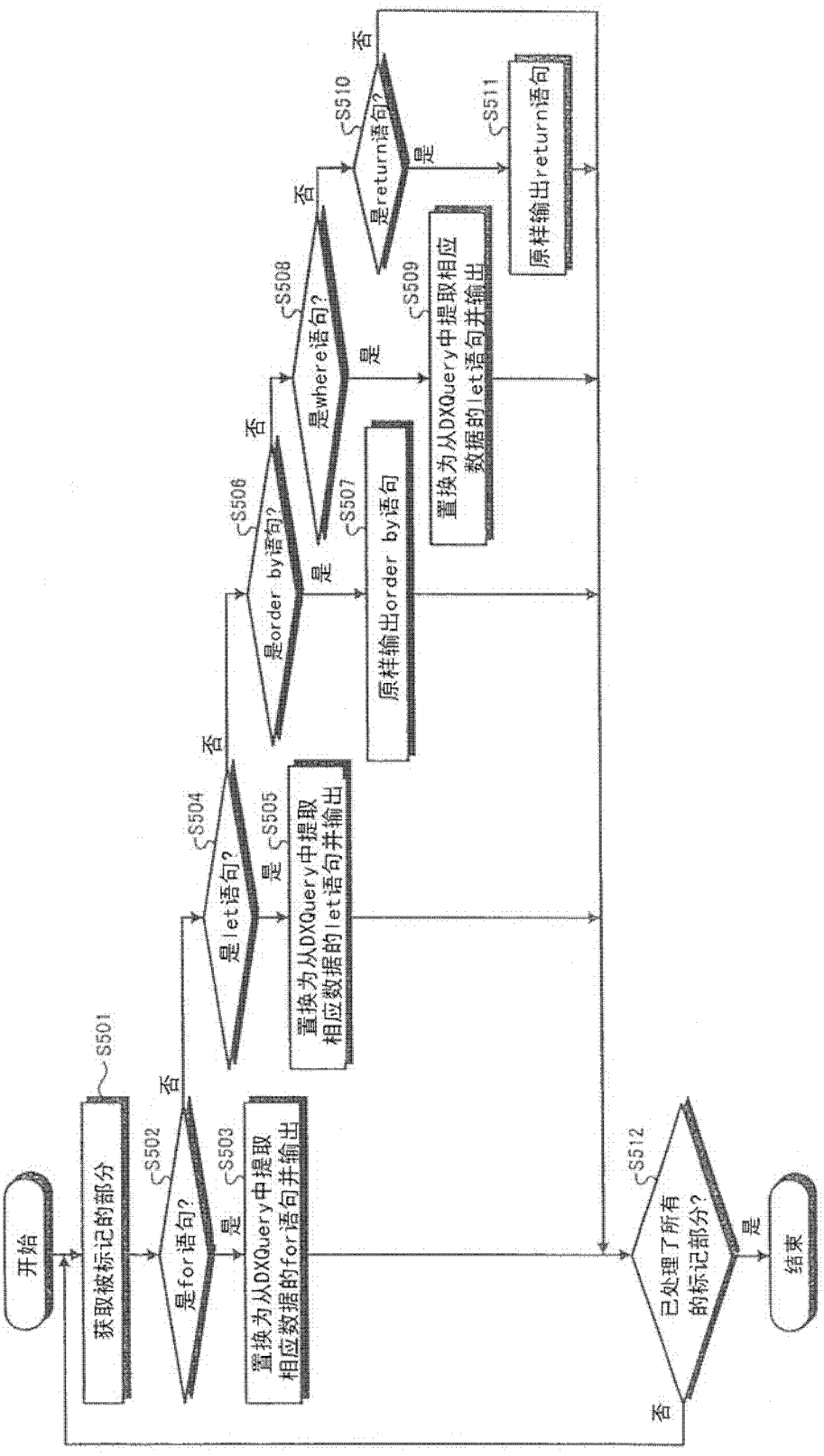

开始
S501
获取被标记的部分
S502
是for语句?
否
是
S503
置换为从DXQuery中提取相应数据的for语句并输出
S504
是let语句?
否
是
S505
置换为从DXQuery中提取相应数据的let语句并输出
S506
是order by语句?
否
是
S507
原样输出order by语句
S508
是where语句?
否
是
S509
置换为从DXQuery中提取相应数据的let语句并输出
S510
是return语句?
否
是
S511
原样输出return语句
S512
已处理了所有的标记部分?
否
是
结束

图 9

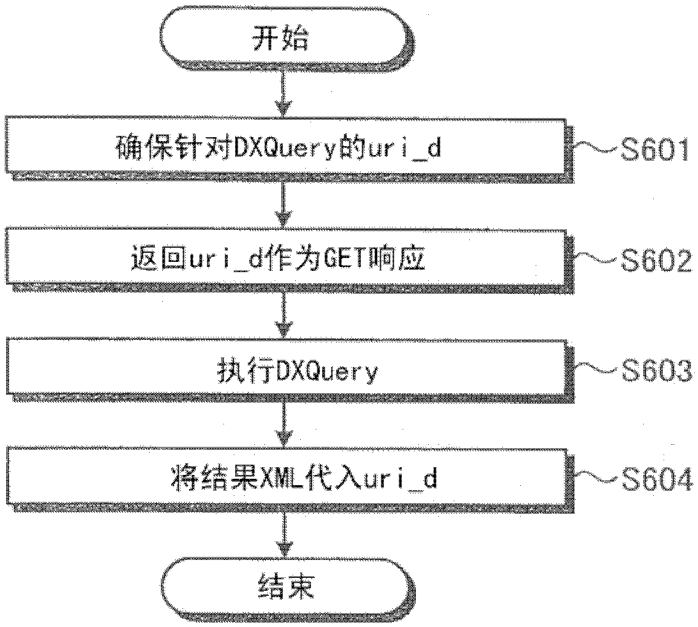

开始
确保针对DXQuery的uri_d
S601
返回uri_d作为GET响应
S602
执行DXQuery
S603
将结果XML代入uri_d
S604
结束

图 10

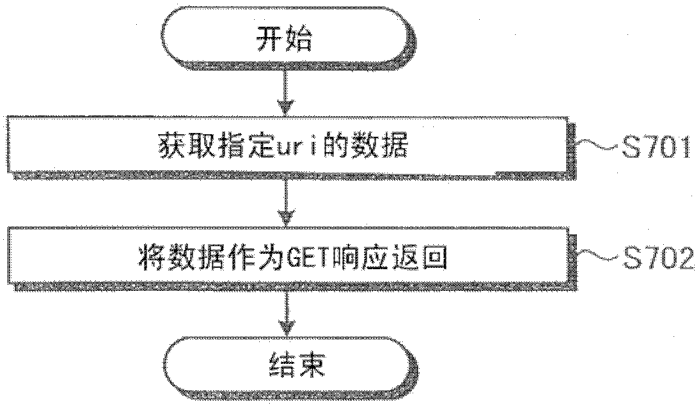

开始
获取指定uri的数据
S701
将数据作为GET响应返回
S702
结束

图 11

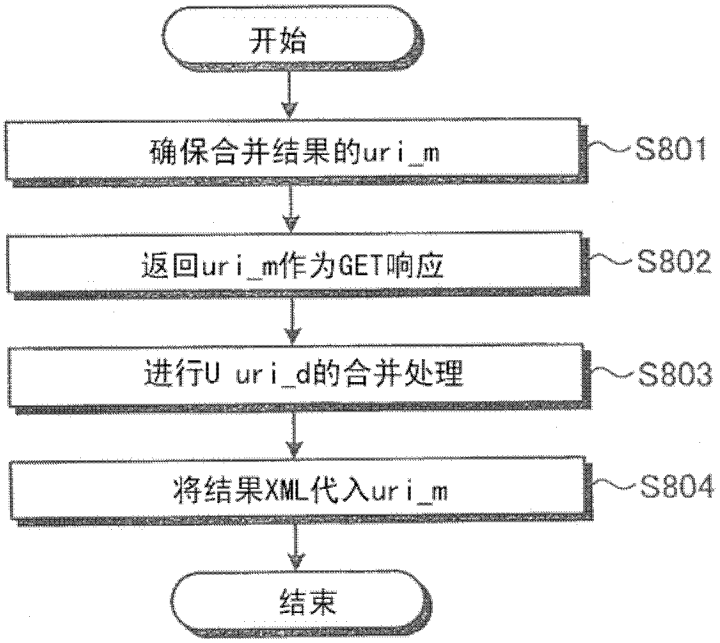

开始
确保合并结果的uri_m
S801
返回uri_m作为GET响应
S802
进行U uri_d的合并处理
S803
将结果XML代入uri_m
S804
结束

图 12

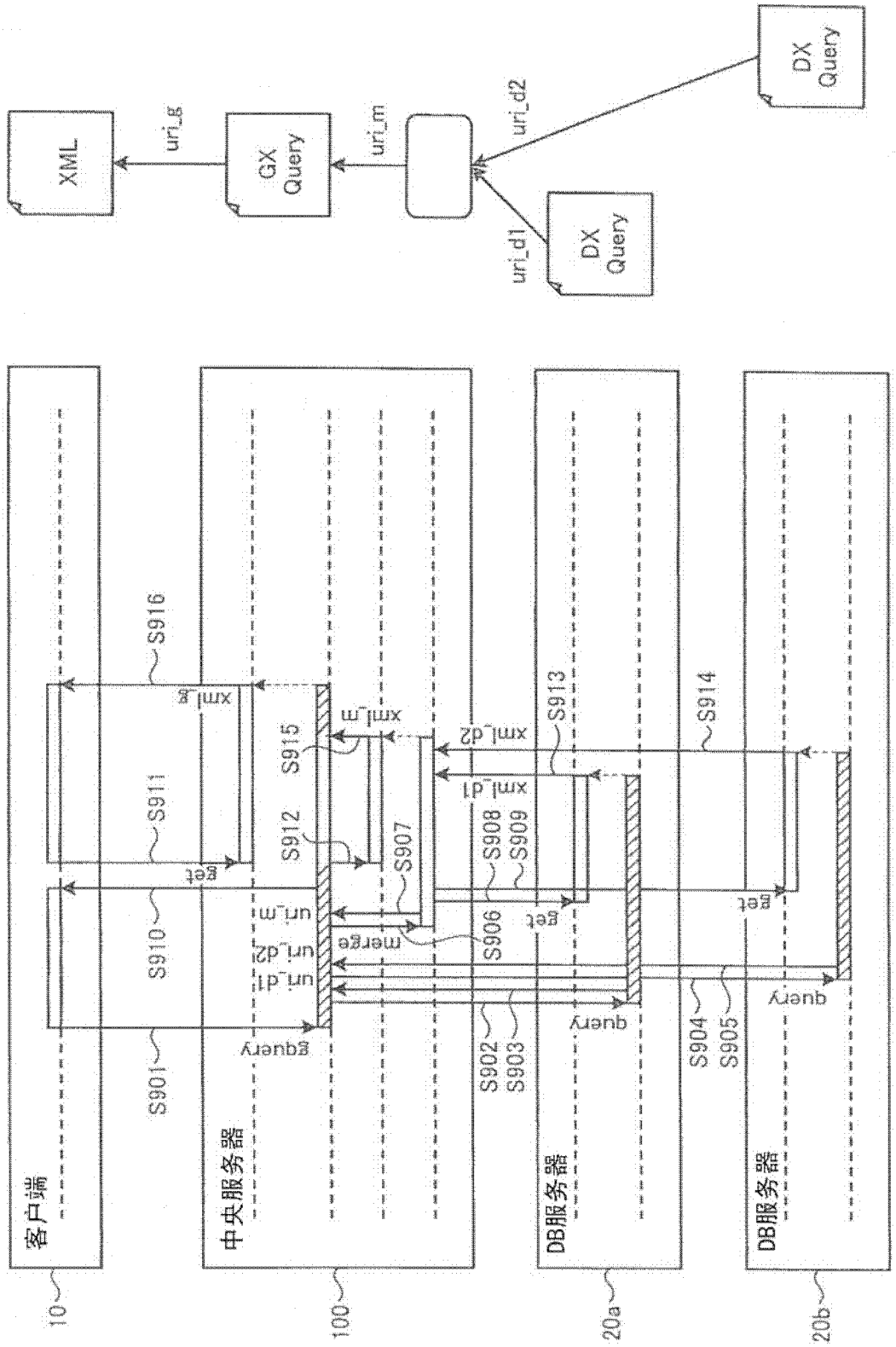

XML
uri_g
GX Query
uri_m
uri_d2
DX Query
uri_d1
DX Query
客户端
中央服务器
DB服务器
DB服务器
10
100
20a
20b
S901
S902
S903
S904
S905
S906
S907
S908
S909
S910
S911
S912
S913
S914
S915
S916
xml_g
xml_m
xml_d2
xml_d1
get
equery
merge
query

图 13

```
for $x in doc("database.xml")//row
where contains($x//column3, " 神奈川 ")
return $x
```

图 14

```
<doc>
 <row>
  <column1>XXX 机关部门 </column1>
  <column2>XXX 丁目 XXX 番地</column2>
  <column3> 神奈川县 </column3>
 </row>
 <row>
  <column1>XXX电气 </column1>
  <column2>XXX 丁目 XXX 番地 </column2>
  <column3> 神奈川县 </column3>
 </row>
 <row>
  <column1>XXX 汽车 </column1>
  <column2>XXX 丁目XXX番地 </column2>
  <column3> 神奈川县 </column3>
 </row>
 <row>
  <column1>XXX 百货店</column1>
  <column2>XXX 丁目XXX 番地</column2>
  <column3> 东京都</column3>
 </row>
 ....
</doc>
```

图 15

```
for $x in doc("database.xml")//row
where contains($x//column3, " 神奈川 ")
return
<rec>
{<col0>{$x}</col0>}
</rec>
```

图 16

xml_d1

```
<rec>
  <col0>
    <row>
      <column1>XXX 机关部门 </column1>
      <column2>XXX丁目 XXX 番地</column2>
      <column3> 神奈川县 </column3>
    </row>
  </col0>
</rec>
<rec>
  <col0>
    <row>
      <column1>XXX 电气</column1>
      <column2>XXX丁目 XXX 番地</column2>
      <column3> 神奈川县 </column3>
    </row>
  </col0>
</rec>
<rec>
  <col0>
    <row>
      <column1>XXX 汽车 </column1>
      <column2>XXX丁目 XXX 番地</column2>
      <column3> 神奈川县 </column3>
    </row>
  </col0>
</rec>
```

图 17

```
for $_0 in doc([uri_m])/root/rec
for $x in $_0/col0/*
return $x
```

图 18

xml_g

```
<row>
  <column1>XXX 机关部门 </column1>
  <column2>XXX 丁目 XXX 番地 </column2>
  <column3> 神奈川县 </column3>
</row>
<row>
  <column1>XXX 电气 </column1>
  <column2>XXX 丁目 XXX 番地 </column2>
  <column3> 神奈川县 </column3>
</row>
<row>
  <column1>XXX 汽车 </column1>
  <column2>XXX 丁目 XXX 番地 </column2>
  <column3> 神奈川县 </column3>
</row>
<row>
  <column1>YYY 事务所 </column1>
  <column2>YYY 丁目 YYY 番地 </column2>
  <column3> 神奈川县 </column3>
</row>
<row>
  <column1>YYY 银行 </column1>
  <column2>YYY 丁目 YYY 番地 </column2>
  <column3> 神奈川县 </column3>
</row>
```

图 19

```
let $d := doc("database.xml")//row/column3
hfor $x in distinct-values($d)
let $y :=
  for $z in doc("database.xml")//row
  where $z/column3 = $d
  return $z
let $w := count($y)
return
<shukei>
  <prefecture>{$x}</prefecture>
  <count>{$w}</count>
</shukei>
```

图 20

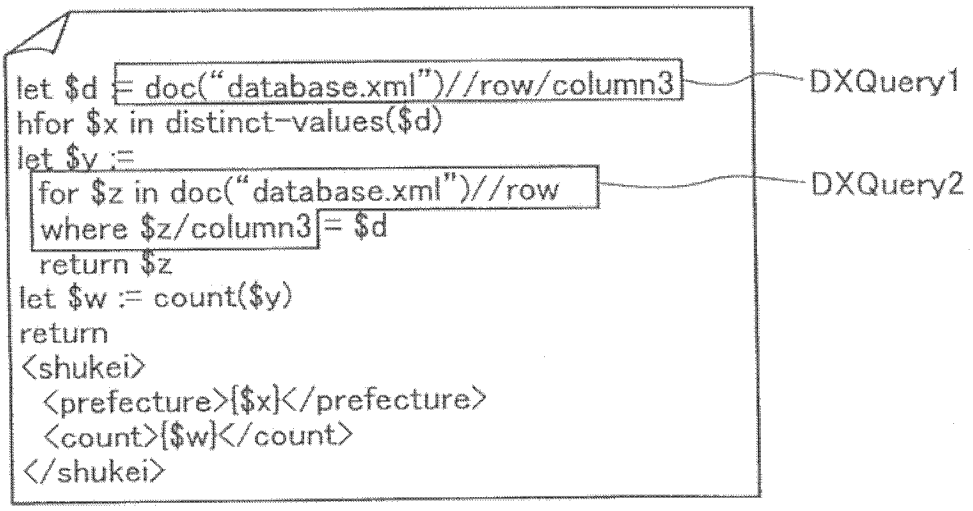


图 21

```
let $d := doc("database.xml")//row/column3
return
<rec>
{for $_0 in $d return <col0>{$_0}</col0>}
</rec>
```

图 22

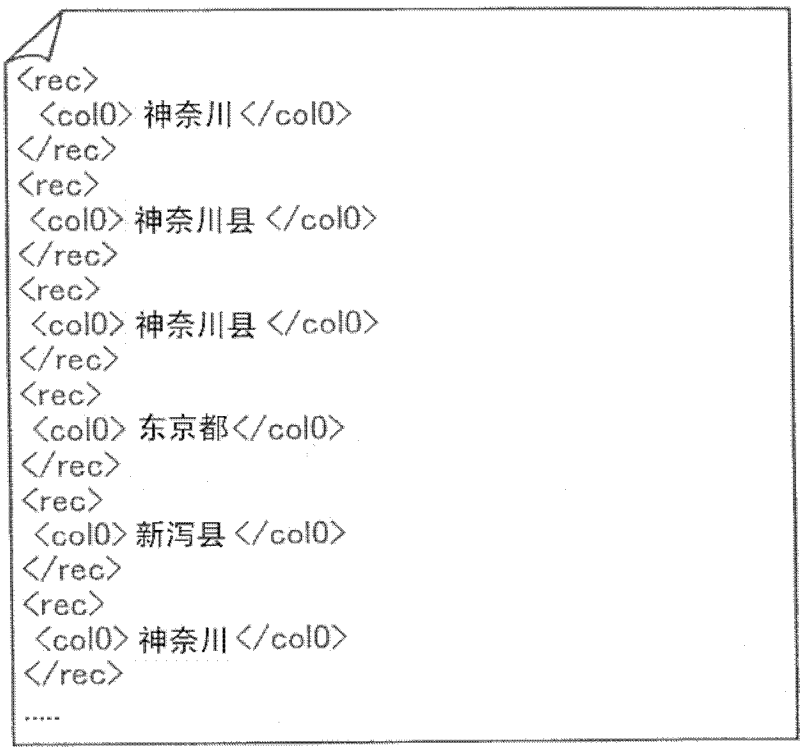


图 23

```
for $z in doc("database.xml")//row
let $_0 := $z/column3
return
<rec>
{<col0>{$z}</col0>}
{for $_1 in $_0 return <col1>{$_1}</col1>}
</rec>
```

图 24

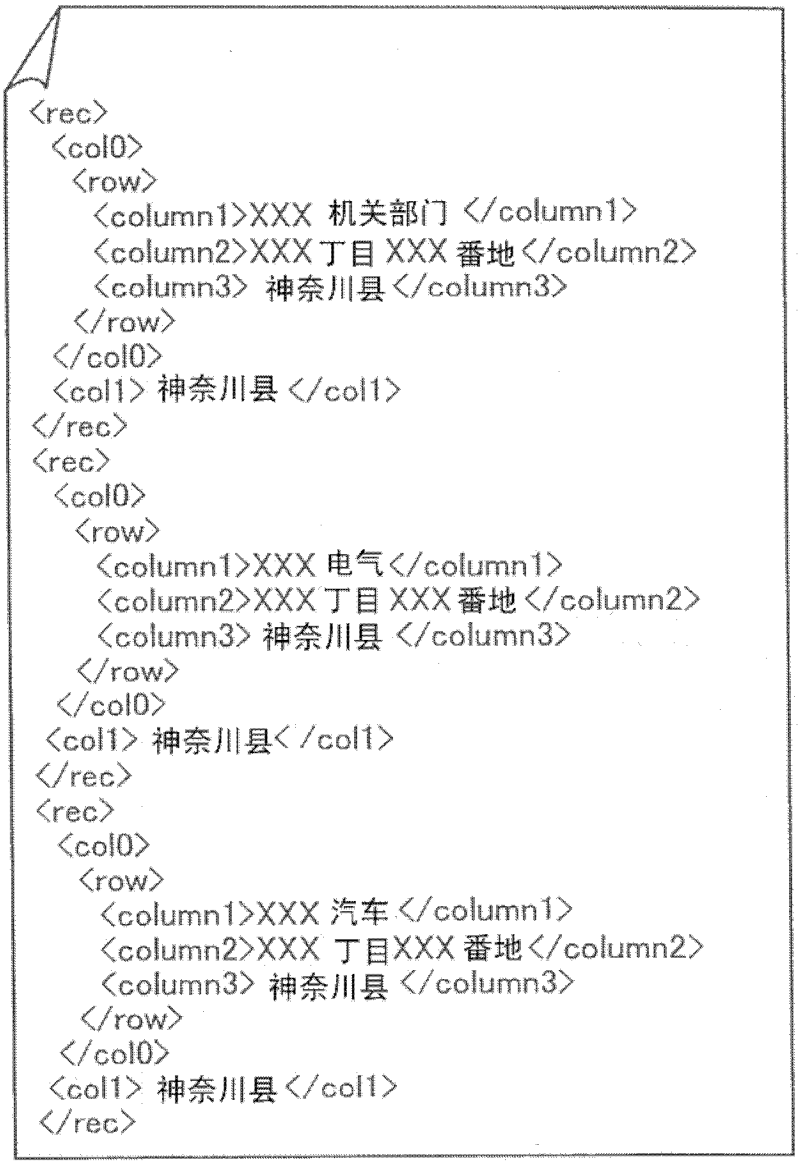

<rec>
<col0>
<row>
<column1>XXX 机关部门 </column1>
<column2>XXX 丁目 XXX 番地</column2>
<column3> 神奈川县</column3>
</row>
</col0>
<col1> 神奈川县 </col1>
</rec>
<rec>
<col0>
<row>
<column1>XXX 电气</column1>
<column2>XXX 丁目 XXX 番地 </column2>
<column3> 神奈川县 </column3>
</row>
</col0>
<col1> 神奈川县< /col1>
</rec>
<rec>
<col0>
<row>
<column1>XXX 汽车 </column1>
<column2>XXX 丁目XXX 番地</column2>
<column3> 神奈川县 </column3>
</row>
</col0>
<col1> 神奈川县</col1>
</rec>

图 25

```
let $d :=
  for $_0 in doc([uri_m1])/root/rec
  for $_1 in $_0/col0/*
  return $_1
for $x in distinct-values($d)
let $y :=
  for $_2 in doc([uri_m2])/root/rec
  for $z in $_2/col0/*
  let $_3 := $_2/col1/*
  where $_3 = $x
  return $z
let $w := count($y)
<shukei>
  <prefecture>{$x}</prefecture>
  <count>{$w}</count>
</shukei>
```

图 26

```
<shukei>
  <prefecture> 神奈川县 </prefecture>
  <count>942</count>
</shukei>
<shukei>
  <prefecture> 东京都 </prefecture>
  <count>2895</count>
</shukei>
<shukei>
  <prefecture> 新泻县 </prefecture>
  <count>127</count>
</shukei>
....
```

图 27

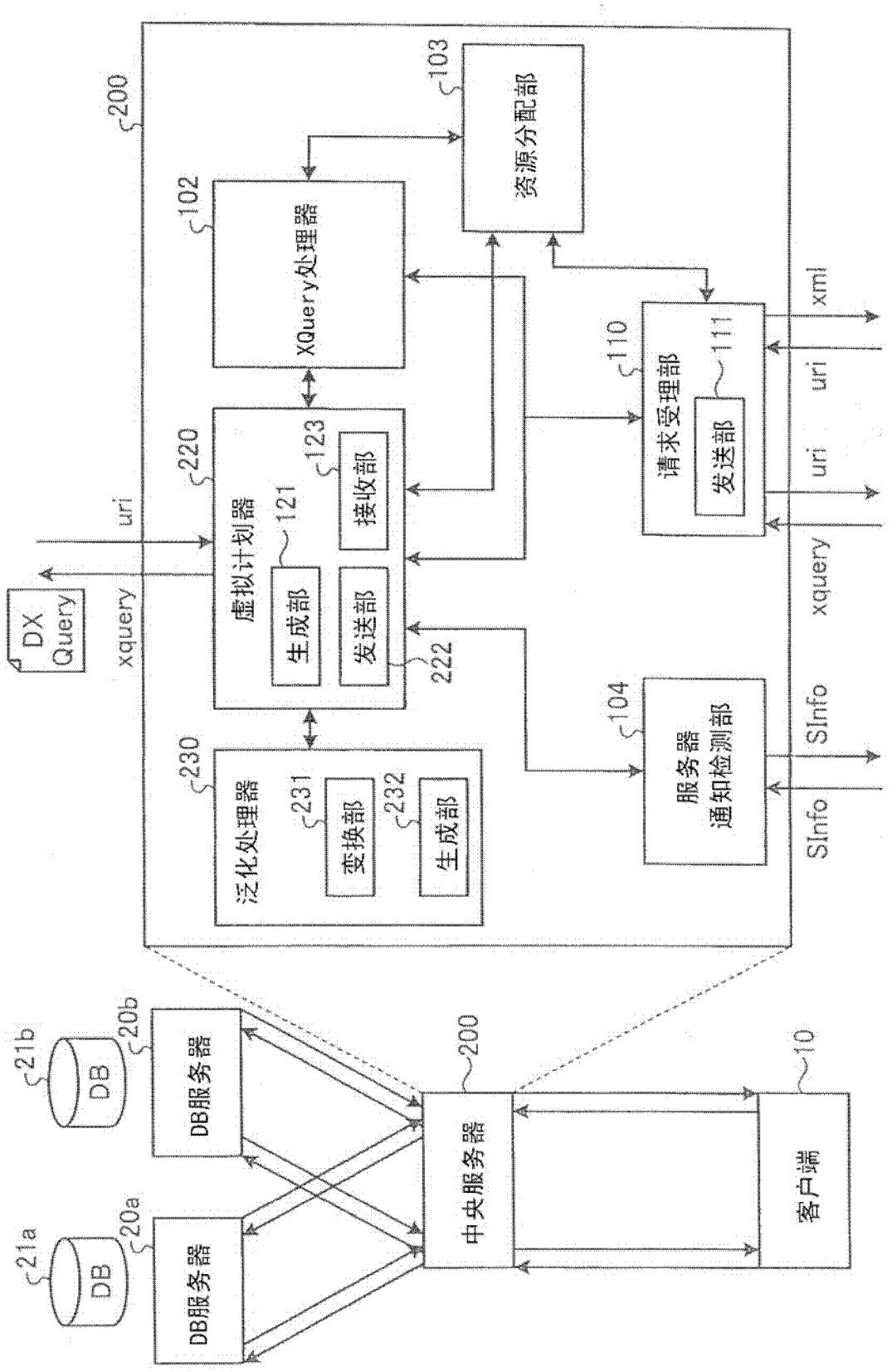

200
103
资源分配部
102
XQuery处理器
220
虚拟计划器
121
生成部
123
接收部
发送部
222
230
泛化处理器
231
变换部
232
生成部
110
请求受理部
111
发送部
104
服务器
通知检测部
uri
xquery
DX
Query
xml
uri
uri
xquery
SInfo
SInfo
21b
DB
20b
DB服务器
21a
DB
20a
DB服务器
200
中央服务器
10
客户端

图 28

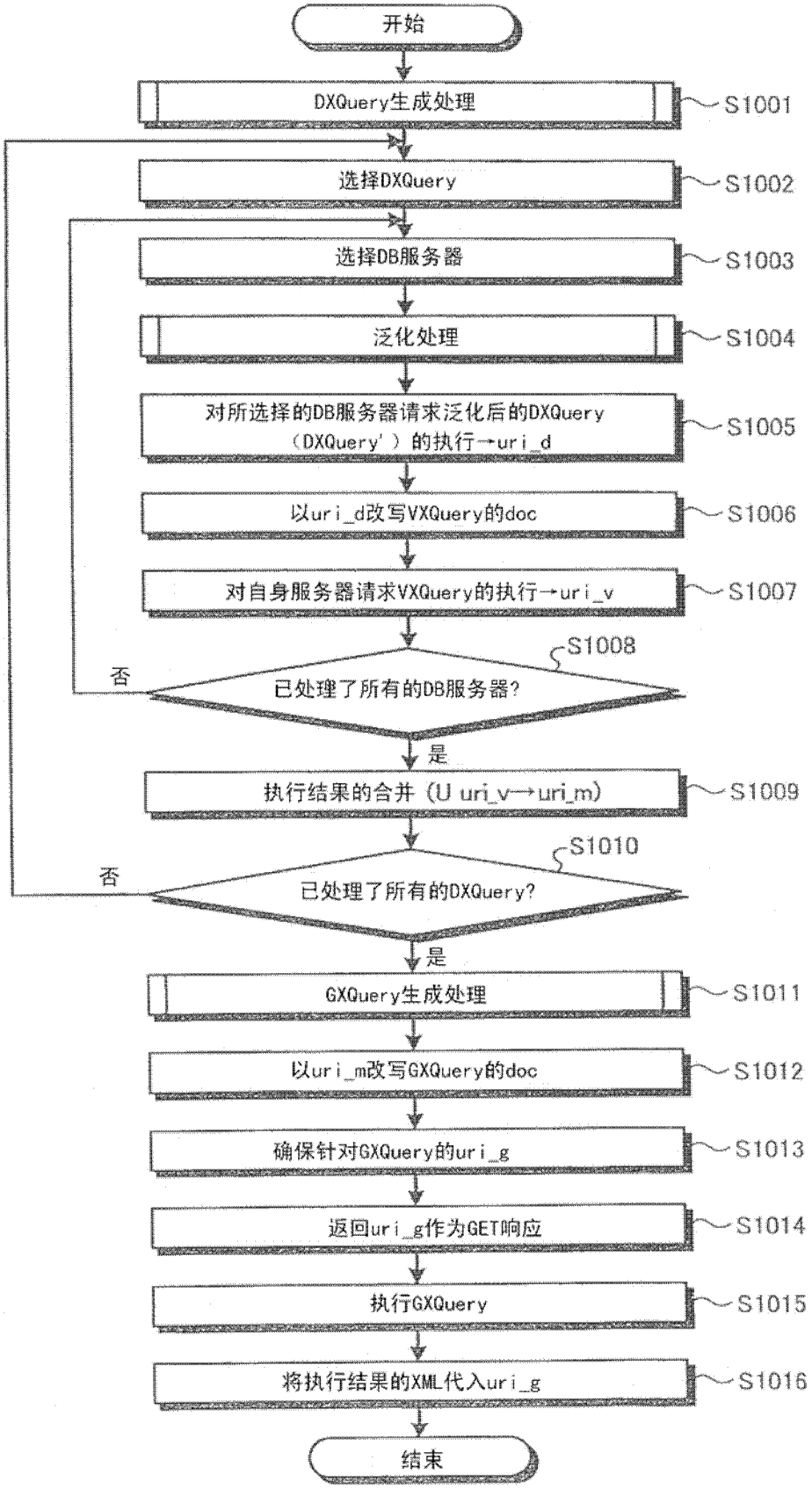

开始
DXQuery生成处理
S1001
选择DXQuery
S1002
选择DB服务器
S1003
泛化处理
S1004
对所选择的DB服务器请求泛化后的DXQuery（DXQuery'）的执行→uri_d
S1005
以uri_d改写VXQuery的doc
S1006
对自身服务器请求VXQuery的执行→uri_v
S1007
S1008
已处理了所有的DB服务器?
否
是
执行结果的合并（U uri_v→uri_m）
S1009
S1010
已处理了所有的DXQuery?
否
是
GXQuery生成处理
S1011
以uri_m改写GXQuery的doc
S1012
确保针对GXQuery的uri_g
S1013
返回uri_g作为GET响应
S1014
执行GXQuery
S1015
将执行结果的XML代入uri_g
S1016
结束

图 29

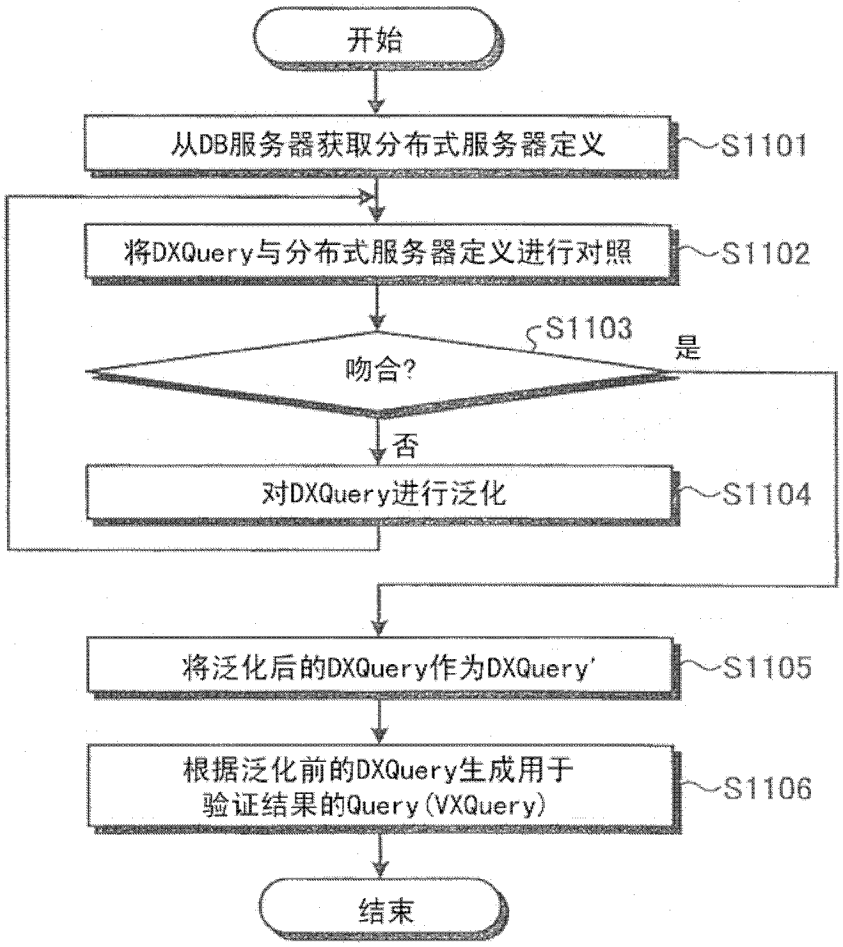


图 30

```
SERVICE http://example.com/?key=%1
XQuery for $x in doc("_")/* where contains($x, "%1") return $x
```

图 31

```
for $x in doc("database.xml")/*
where contains($x, "神奈川")
return
<rec>
{<col0>{$x}</col0>}
</rec>
```

图 32

```
for $_0 in doc([uri_d1])/rec
for $_1 in $_0/col0/*
for $x in $_1//row
where contains($x//column3, “神奈川”)
return
<rec>
{<col0>{$x}</col0>}
</rec>
```

图 33

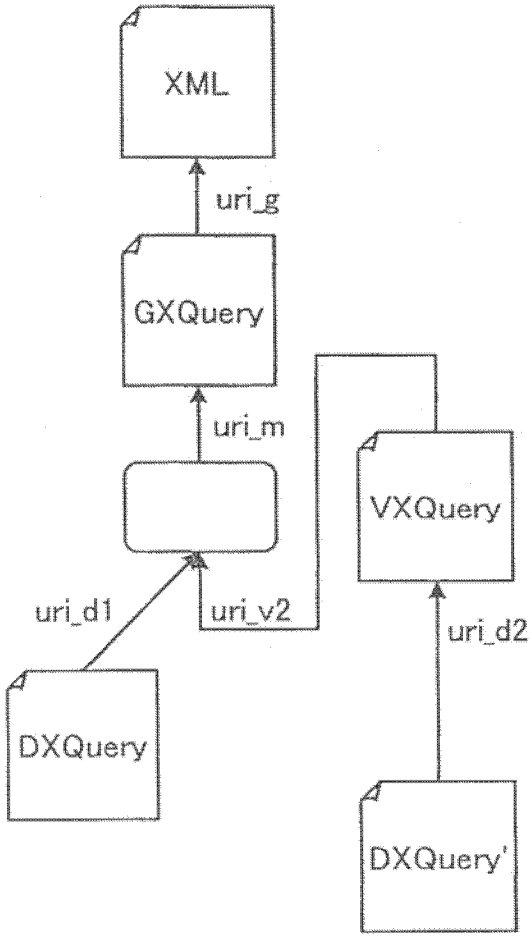


图 34

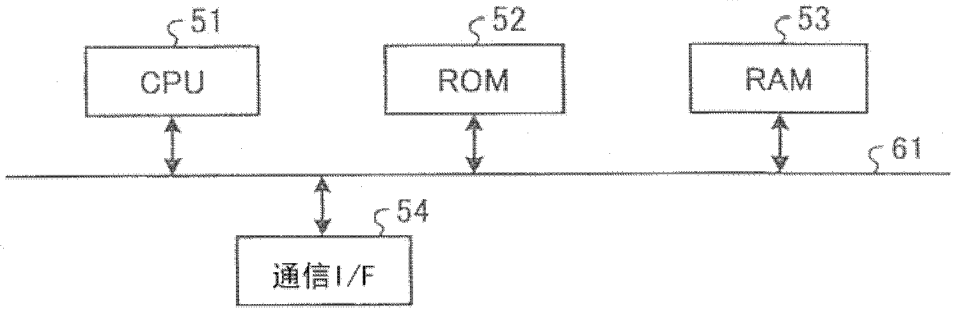


图 35