(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 101686146 B
(45)授权公告日 2013.01.30

(21)申请号 200810166965.2
(22)申请日 2008.09.28
(73)专利权人 华为技术有限公司
地址 518129 广东省深圳市龙岗区坂田华为总部办公楼
(72)发明人 张彬 金丰鑫 李国辉
(74)专利代理机构 北京挺立专利事务所(普通合伙) 11265
代理人 叶树明
(51)Int.Cl.
H04L 12/24(2006.01)
G06F 17/30(2006.01)

(56)对比文件
US 2005/0050066 A1,2005.03.03, 全文.
CN 101222468 A,2008.07.16, 全文.
US 2004/0153469 A1,2004.08.05, 说明书第3-4页,权利要求47-48.

审查员 闫洪波

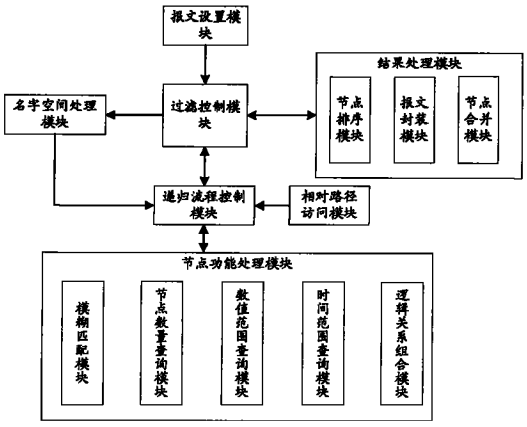
权利要求书 3 页 说明书 46 页 附图 8 页

(54)发明名称

模糊查询、查询结果处理和过滤条件处理的方法及设备

(57)摘要

本发明实施例公开了一种模糊查询、查询结果处理和过滤条件处理的方法及设备,其中基于子树过滤的模糊查询方法包括:接收待过滤数据流;通过不完全匹配的方式过滤所述数据流,处理用户没有给出完全信息的字符串形式的过滤条件。本发明实施例解决NETCONF协议子树过滤机制仅局限于精确匹配和绝对路径,数据查询中要求给出较多已知信息带来的不便。



CN 101686146 B

1. 一种基于子树过滤的模糊查询方法，其特征在于，包括：

接收待过滤数据流；

通过不完全匹配的方式过滤所述数据流，处理用户没有给出完全信息的字符串形式的过滤条件；

其中，当出现节点的命名空间不匹配时，继续处理所述节点的子节点，保留命名空间不匹配的元素作为其后代节点的路径；

所述通过不完全匹配的方式过滤所述数据流为：

通过绝对路径过滤所述数据流；或通过相对路径过滤所述数据流；

所述通过绝对路径过滤所述数据流具体为：用户给出的过滤器包含被选择节点的从根节点开始的所有祖先，过滤结果中只包含在过滤器中的祖先和在数据模型中的祖先完全一致的节点及节点后代；

所述通过相对路径过滤所述数据流具体为：

在过滤器中满足父子关系的节点在过滤结果中至少满足祖先后代关系；在过滤器中为根节点的节点在过滤结果中至少包括非根节点。

2. 如权利要求 1 所述的方法，其特征在于，所述字符串包括：节点名字、节点内容、属性名字、属性值或命名空间。

3. 如权利要求 1 所述的方法，其特征在于，所述不完全匹配的方式包括：路径不完整和 / 或元素名不完整和 / 或元素值不完整和 / 或属性名不完整和 / 或属性值不完整和 / 或命名空间不完整。

4. 一种子树过滤条件的逻辑组合的扩展方法，其特征在于，包括：

接收待过滤数据流；

采用多个属性或多个内容匹配节点对所述数据流中的元素类型节点进行过滤；

所述多个内容匹配节点对所述数据流中的元素类型节点进行过滤，具体为：

使用内容匹配节点作为从数据模型中选择特定的父节点的过滤条件；

通过在所述内容匹配节点中添加第一属性表示所述节点的匹配程度，匹配程度包括必须匹配、可选匹配和不匹配；

根据所述内容匹配节点和所述匹配程度对所述数据流中的节点进行过滤；

其中，根据所述内容匹配节点和所述匹配程度对所述数据流中的节点进行过滤，具体为：

过滤器中所有匹配程度是必须匹配的内容匹配节点都与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

过滤器中可选匹配程度是可选匹配的内容匹配节点中至少有一个节点与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

过滤器中匹配程度不匹配的内容匹配节点与数据模型中相应的节点完全匹配时，其父节点不被添加到过滤结果中。

5. 如权利要求 4 所述的方法，其特征在于，还包括：

向节点中添加第二属性，以实现所述节点的多个属性间的逻辑关系。

6. 如权利要求 4 所述的方法，其特征在于，还包括：进行命名空间查询：

获取数据模型中命名空间的定义情况，并给出命名空间的列表，其中，命名空间查询的

关键字用一个元素表示。

7. 如权利要求 4 所述的方法，其特征在于，过滤条件中还包括范围条件：

所述范围条件用于选择具有某些特定数值或者日期范围的节点，在表示数量或时间的选择节点中添加表示属性范围或时间范围的属性，将该选择节点看作是表示时间或数量范围的内容匹配节点，在过滤处理时作为过滤条件。

8. 一种过滤器，其特征在于，包括：

接收模块，用于接收待过滤数据流；

过滤模块，用于通过不完全匹配的方式过滤所述数据流，处理用户没有给出完全信息的字符串形式的过滤条件，其中，当出现节点的命名空间不匹配时，继续处理所述节点的子节点，保留命名空间不匹配的元素作为其后代节点的路径；

所述通过不完全匹配的方式过滤所述数据流为：

通过绝对路径过滤所述数据流；或通过相对路径过滤所述数据流；

所述通过绝对路径过滤所述数据流具体为：用户给出的过滤器包含被选择节点的从根节点开始的所有祖先，过滤结果中只包含在过滤器中的祖先和在数据模型中的祖先完全一致的节点及节点后代；

所述通过相对路径过滤所述数据流具体为：

在过滤器中满足父子关系的节点在过滤结果中至少满足祖先后代关系；在过滤器中为根节点的节点在过滤结果中至少包括非根节点。

9. 如权利要求 8 所述过滤器，其特征在于，所述过滤模块包括：

第一过滤子模块，用于通过绝对路径过滤所述数据流；或者，

第二过滤子模块，用于通过相对路径过滤所述数据流。

10. 一种逻辑过滤装置，其特征在于，包括：

接收模块，用于接收待过滤数据流；

逻辑过滤模块，用于采用多个属性或多个内容匹配节点对所述数据流中的元素类型节点进行过滤；

所述逻辑过滤模块包括：

设置子模块，用于设置内容匹配节点作为从数据模型中选择特定的父节点的过滤条件；

添加子模块，用于通过在所述内容匹配节点中添加第一属性表示所述节点的匹配程度，并实现所述节点过滤条件之间的逻辑关系组合；

处理子模块，用于根据所述过滤条件之间的逻辑关系组合对所述数据流中的元素类型节点进行过滤；

其中，根据所述内容匹配节点和所述匹配程度对所述数据流中的节点进行过滤，具体为：

过滤器中所有匹配程度是必须匹配的内容匹配节点都与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

过滤器中可选匹配程度是可选匹配的内容匹配节点中至少有一个节点与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

过滤器中匹配程度不匹配的内容匹配节点与数据模型中相应的节点完全匹配时，其父

节点不被添加到过滤结果中。

模糊查询、查询结果处理和过滤条件处理的方法及设备

技术领域

[0001] 本发明涉及通信技术领域，尤其涉及一种模糊查询、查询结果处理和过滤条件处理的方法及设备。

背景技术

[0002] NETCONF（网络配置协议）是一种网管协议，使用XML(ExtensibleMarkup Language，可扩展标记语言）描述操作请求和网管数据。XML是一套语义标记的规则，将文档分成许多部件并用标记对这些部件加以标识。这些标记通常采用自然语言，因此具有很高的可读性。标记看起来如下：

[0003] <age>10</age>

[0004] <name language =“English”>

[0005] <first-name>George</first-name>

[0006] <last-name>Bush</last-name>

[0007] </name>

[0008] <hobby></hobby>

[0009] <hobby/>

[0010] 每一对标记由起始标记（如<age>）和结束标记（如</age>）组成，每一对标记及其内容称为一个元素（element），在以上例子中就有<age>、<name>元素。元素可以拥有值，如<age>的值是10；也可以拥有子元素，如<name>拥有子元素<first-name>和<last-name>；也可以没有值，如<hobby>，这种情况称为空元素，可缩写记为<hobby/>；元素还可以拥有属性（attribute），属性是一个“名字-值”对，如<name>拥有属性language，属性值是English。

[0011] 元素间的层次嵌套关系经常表达为一个树型数据结构，如图1所示，因此元素也被称作节点。在一对嵌套关系中，外层节点被称作内层节点的父节点；内层节点被称作外层节点的子节点。如图1中，<a>是<b1>、<b2>、<b3>的父节点，<b1>、<b2>、<b3>是<a>的子节点。拥有同一父节点的几个节点相互称为兄弟节点，如<b1>、<b2>、<b3>彼此是兄弟节点。

[0012] 在网络管理中，需要监视网络设备的配置和状态，为达到此目的，网管设备发出查询报文，描述所需要查询的数据，被管设备收到查询报文后，依照查询报文中的条件执行查询，并向网管设备应答查询结果。

[0013] NETCONF协议拥有两种查询操作：<get>和<get-config>。<get>操作用于查询当前正在运行的配置和状态数据；<get-config>用于查询不同配置数据存储中的配置数据。尽管两者的查询对象有所区别，但拥有相同的查询条件表达方式，即子树过滤和XPATH(XML路径语言）方式。

[0014] 现有技术一中，子树过滤使用XML方式描述过滤条件，通过一组简单的匹配规则，对符合条件的数据元素进行选取。例1是一个子树查询的例子，<rpc>元素表明这是一个网

管设备发出的请求，<get>元素表明网管设备请求执行查询操作，<filter>元素指明了查询方式和查询条件，其属性 type =“subtree”指出该查询采用子树过滤方式，<filter>包含的子节点表明：该查询希望查询命名空间" http://example.com/schema/1.2/stats" 内的<top>元素中的<interfaces>元素中的某个<interface>元素，并且该<interface>元素必须具有 ifName 属性，ifName 的属性值是" eth0" 。

[0015] 例 1：一个使用子树查询的<get>请求：

[0016] <rpc message-id =" 101"

[0017] xmlns =" um:ietf:params:xml:ns:netconf:base:1.0" >

[0018] <get>

[0019] <filter type =“subtree”>

[0020] <t:top xmlns:t =" http://example.com/schema/1.2/stats" >

[0021] <t:interfaces>

[0022] <t:interface t:ifName =" eth0" />

[0023] </t:interfaces>

[0024] </t:top>

[0025] </filter>

[0026] </get>

[0027] </rpc>

[0028] 其中，<filter>元素被称为过滤器，在子树过滤中，过滤器由零到多个代表选择标准的元素子树构成。在子树的各层，被管设备对每个兄弟节点集合进行逻辑判断，以决定各兄弟节点所牵头的子树是否包含在过滤器的输出结果中。所有在过滤器子树中出现的元素，必须与服务器概念数据模型中的对应节点相匹配。

[0029] 过滤器可分为 5 种组件，每种组件执行不同的过滤规则，包括：命名空间选择、属性匹配表达式、包含节点、选择节点、内容匹配节点。

[0030] 命名空间选择：如果使用命名空间，则过滤器的输出只包含指定的命名空间中的元素。即：过滤器中”xmlns”属性的内容应与被管设备数据模型中”xmlns”属性的内容相同。在例 1 中 <t:top

[0031] xmlns:t =" http://example.com/schema/1.2/stats" >指出了选择的命名空间是 http://example.com/schema/1.2/stats，只有该命名空间中的<top>中的子元素被输出。

[0032] 属性匹配表达式：出现在过滤器中的属性是属性匹配表达式的一部分。被选择的数据除了要与过滤器中的节点匹配，还必须与过滤器的属性相匹配。如果一个元素不包含指定的属性，则该元素将不被选择。例 1 中，ifName =" eth0" 就是属性匹配表达式。过滤结果将输出拥有 ifName 属性，并且 ifName 的属性值是" eth0" 的<interface>元素，如例 2 所示。例 2 中，<rpc-reply>元素表明这是一个应答，<data>元素表明应答的数据，应答数据的命名空间、各级子元素和属性都和过滤器中的对应部分相匹配。

[0033] 例 2 属性匹配的查询结果：

[0034] <rpc-reply message-id =" 101"

[0035] xmlns =" um:ietf:params:xml:ns:netconf:base:1.0" >

[0036] <data>

[0037] <t:top xmlns:t =" http://example.com/schema/1.2/stats" >

[0038] <t:interfaces>

[0039] <t:interface t:ifName =" eth0" >

[0040] <t:ifInOctets>45621</t:ifInOctets>

[0041] <t:ifOutOctets>774344</t:ifOutOctets>

[0042] </t:interface>

[0043] </t:interfaces>

[0044] </t:top>

[0045] </data>

[0046] </rpc-reply>

[0047] 包含节点:过滤器中包含子元素的节点被称为包含节点。包含节点的子元素也可以是包含节点,或其他类型的节点。例1中,<top>是一个包含节点,其子元素<interfaces>也是一个包含节点。

[0048] 选择节点:过滤器中的空节点被称为选择节点,表示对数据模型的一个显式选择。如果在数据模型的兄弟节点中出现了选择节点,则只有该选择节点下的子树被选择,该选择节点的兄弟节点不被选择。如例3中,<name>元素是个选择节点。在被管设备的数据模型中,如果<user>除了<name>还有其他子节点,那么只有<name>才被选择,<name>的兄弟节点将不被选择。

[0049] 例3过滤器中的选择节点:

[0050] <filter type =“subtree”>

[0051] <top xmlns =" http://example.com/schema/1.2/config" >

[0052] <users>

[0053] <user>

[0054] <name/>

[0055] </user>

[0056] </users>

[0057] </top>

[0058] </filter>

[0059] 内容匹配节点:包含文本值的叶子节点被称为内容匹配节点,表示对父节点的选择条件。例4中,<name>元素是内容匹配节点,表示所有<name>为”fred”的<user>元素(及其子元素)将被选择输出。该过滤器的查询结果如例5所示。

[0060] 多个兄弟内容匹配节点间执行“与”的逻辑运算;

[0061] 例4过滤器中的内容匹配节点:

[0062] <filter type =“subtree”>

[0063] <top xmlns =" http://example.com/schema/1.2/config" >

[0064] <users>

[0065] <user>

[0066] <name>fred</name>

[0067] </user>

[0068] </users>

[0069] </top>

[0070] </filter>

[0071] 例 5 针对内容匹配节点的应答输出：

[0072] <rpc-reply message-id ＝″ 101″

[0073] xmlns ＝″ um:ietf:params:xml:ns:netconf:base:1.0″ >

[0074] <data>

[0075] <top xmlns ＝″ http://example.com/schema/1.2/config″ >

[0076] <users>

[0077] <user>

[0078] <name>fred</name>

[0079] <type>admin</type>

[0080] <full-name>Fred Flintstone</full-name>

[0081] <company-info>

[0082] <dept>2</dept>

[0083] <id>2</id>

[0084] </company-info>

[0085] </user>

[0086] </users>

[0087] </top>

[0088] </data>

[0089] </rpc-reply>

[0090] 综上所述，子树过滤的流程可概括如下：

[0091] 过滤器的输出开始是空的。被管设备将它所支持的数据模型与过滤器中的每个数据分枝进行比较，如果一个节点所代表的子树的所有分枝与被管设备的数据模型的对应部分实现了完全匹配，则该节点及其所有子节点包含在结果数据中。

[0092] 被管设备将具有相同父节点的节点（即兄弟节点集合）集中在一起进行处理，从根节点到叶子节点。对于每个兄弟节点集合，被管设备首先确定该集合中出现的节点类型，根据不同类型的选择规则进行处理。如果集合中有某些节点被选择，则该处理过程被叠代应用到每个被选中节点的下层兄弟集合之中。算法持续到所有过滤器子树中的兄弟节点集合被处理完毕。

[0093] 子树过滤中过滤条件要求较为严格，进行查询时需要较多的已知条件。协议中对子树过滤机制的描述，认为子树过滤基于简单的精确匹配，仅包含较少类型的过滤器，以对 XML 型子树实现用途广泛但局限较大的选择。这种选择不对数据模型做任何预处理，也不使用任何针对数据模型的特殊规范。子树过滤机制完全采用 XML 语言定义，并没有附加任何针对子树过滤本身的规范，故而查询和过滤功能相对较弱，需要较多的已知条件。过滤条件必须是以绝对路径表达的树形结构。因而如果对查询实体或过滤条件的组成实体在数据模型中的位置不完全清楚，则无法满足任何需求。只能舍弃过滤器，返回整个数据模型。

[0094] 子树过滤只能过滤与过滤请求完全吻合的数据。所有过滤中节点名字、节点值、属性名字、属性值、命名空间等字符只能精确匹配，即过滤器中的字符串必须和数据模型中的字符串完全相同，否则会认为在数据模型中不存在符合过滤条件的节点。包括数字、日期等也必须精确匹配，不能实现范围过滤。

[0095] 比如要查询一个属性为name、值为Ethernet2的<interface>元素，必须完整给出元素的路径为 /netconf/routing/interface，netconf 和 routing 节点的命名空间为 um:ietf:params:xml:ns:netconf:base:1.0，interface的命名空间为urn:bupt:pris:priser:agent:module:interface:1.0，属性名为 name，属性值为 Ethernet2，如例 6 所示，很显然这个查询需要太多的已知条件：

[0096] 例 6 查询属性 name 值为 Ethernet2 的 interface 元素信息：

[0097] <filter type =“subtree”>

[0098] <netconf xmlns =" urn:ietf:params:xml:ns:netconf:base:1.0" >

[0099] <routing>

[0100] <interface xmlns =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0101] name =" Ethernet2" />

[0102] </routing>

[0103] </netconf>

[0104] </filter>

[0105] 子树过滤倾向于向用户返回所有的可能有需要的管理信息，容易造成冗余和混淆。例如当用户试图查询某个节点本身的信息时，子树过滤会将其所有后代一并输出，因为无法从过滤器定义中判断出用户需要的是该节点本身，还是其属性抑或是其某一个子孙。这样可能会造成在网络中传输大量不必要的加密信息以及用户人工查找信息的困难。

[0106] 比如用户只想获得<interface>元素的属性信息，使用了例 7 所示的查询请求，该请求的结果如例 8 所示：

[0107] 例 7 查询 interface 元素的属性：

[0108] <filter type =“subtree”>

[0109] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0" >

[0110] <routing>

[0111] <interface xmlns =" urn:bupt:pris:priser:agent:module:interface:1.0" />

[0112] </routing>

[0113] </netconf>

[0114] </filter>

[0115] 例 8 例 7 的查询结果：

[0116] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0117] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0118] xmlns:mt =" um:bupt:pris:priser:agent:module:monitor:1.0" >

[0119] <routing>

[0120] <if:interface name =" Ethernet1" ip-address =”59.64.139.65”>

[0121] <.../>

[0122] </if:interface>

[0123] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0124] <.../>

[0125] </if:interface>

[0126] </routing>

[0127] </netconf>

[0128] 从过滤结果中可以看出,过滤结果不但返回了<interface>节点的全部属性信息,而且还返回了如<interface>的后代节点等信息(用<.../>表示),对于这个用户的需求来说,这些信息是冗余的。

[0129] 因此,现有技术一中的子树过滤缺少对过滤结果的统计和处理机制。除了返回xml语言规定的实体外,对这些实体的统计信息用户无法获得,同时也无法控制这些实体的返回方式,用户将无法控制返回数据量的大小。

[0130] 现有技术二中,XPath 即 XML 路径语言,是 W3C 关于查询 XML 文档的通用语言标准。从外观上看,XPath 是用文件路径的形式表示 XML 文档查询条件的方法。例如,例 4 中的子树过滤用 XPath 表达就如例 9 所示。例中 type = " xpath " 表明使用 XPath 查询方法,select 属性的值为一个 XPath 表达式,指出了查询的内容和条件。

[0131] 例 9 :XPath 过滤表达式 :

[0132] <filter type =" xpath" select =" top/users/user[name =“fred”]" />

[0133] XPath 本质上是与具有层次结构的 XML 数据模型相匹配的查询语言,可以通过任何方向浏览树来选择节点,并根据节点的值和位置应用谓词。XPath 遵循文档对象模型(Document Object Model,DOM)的路径格式。由于每个XML文档都可以看成是一棵拥有许多节点的树,节点的类型包括 :根节点 (root)、元素节点 (element)、属性节点 (attribute)、文本节点 (text)、命名空间节点 (namespace)、处理指令节点 (processinginstruction) 和注释 (comment)。

[0134] XPath 的目标是定义一种定位 XML 文档各个部分的语言,其功能是在数据存储区中查询某一个节点或节点集,为了实现这个目标,XPath 规范定义了两个主要部分 :一部分是表达式语法,另一部分是一组名为 XPath 核心库的基本函数。

[0135] XPath 的基本语法由表达式构成,在计算表达式的值之后产生一个对象,这种对象有以下四种基本类型 :节点集、布尔型、数字型和字符串型。XPath 基本上和电脑文件系统中的寻找类似,如果路径是以“/“开头的,就表明该路径表示的是一个绝对路径 ;以“//”开头则表示在文档中的任意位置查找。通过 XPath 路径表达式,可以在 XML 文档中轻松地定位数据、确定节点。

[0136] 现有技术二中的 XPath 机制较为复杂,难于掌握。XPath 使用路径表达式来选取 XML 文档中的节点或者节点集,且含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值等。包括各种轴的定义和使用,同时又不具有与 XML 文档相关的直观结构,掌握起来较为困难。

[0137] XPath 需求以非 XML 结构表述,需提供额外机制进行解析和处理。在子树过滤中过

滤器本身也是XML结构，使用起来相对直观，处理起来与NETCONF协议中的管理数据模块方法也基本类似。但是XPath是通过路径表达式来选取XML文档中的节点或者节点集，这样对其解析还需要另外的解析和处理方法。

[0138] XPath是通过路径表达式来选取XML文档中的节点或者节点集，每次只能获取一条查询记录，这样对于用户想一次实现多个结果的查询就无法实现，查询效率比较低。

[0139] 在实现本发明的过程中，发明人发现现有技术至少存在以下缺点：

[0140] 现有技术一中的子树过滤基于精确匹配和绝对路径，需要用户提供较多已知条件，缺少对过滤结果的统计和处理机制；而现有技术二中XPath机制较为复杂，难于掌握。

发明内容

[0141] 本发明实施例提供了一种模糊查询、查询结果处理和过滤条件处理的方法及设备，以解决NETCONF协议子树过滤机制仅局限于精确匹配和绝对路径，数据查询中要求给出较多已知信息带来的不便；解决NETCONF协议数据模型查询中，对命名空间查询、过滤、处理手段匮乏的问题；丰富子树过滤机制的查询和过滤手段，提高子树过滤的灵活性和功能性；处理子树过滤得到的XML过滤结果文档，减少冗余信息的处理和传输；修正NETCONF协议中子树过滤部分不合理的过滤报文处理方法。

[0142] 本发明实施例提供了一种基于子树过滤的模糊查询方法，包括：

[0143] 接收待过滤数据流；

[0144] 通过不完全匹配的方式过滤所述数据流，处理用户没有给出完全信息的字符串形式的过滤条件

[0145] 本发明实施例还提供了一种子树过滤条件的逻辑组合的扩展方法，包括：

[0146] 接收待过滤数据流；

[0147] 采用多个属性或多个内容匹配节点对所述数据流中的元素类型节点进行过滤。

[0148] 本发明实施例还提供了一种子树过滤查询结果的处理方法，包括：

[0149] 在过滤器中相应节点处设置第三属性的值，表明所述节点处是否显示节点后代；

[0150] 所述第三属性的取值为：第一值或者第二值，当所述第三属性默认为显示所有在同一命名空间的后代，则第三属性＝第二值；当设置第三属性＝第一值，则过滤结果只输出该节点的信息，不输出所述在同一命名空间的后代节点。

[0151] 本发明实施例还提供了一种过滤器，包括：

[0152] 接收模块，用于接收待过滤数据流；

[0153] 过滤模块，用于通过不完全匹配的方式过滤所述数据流，处理用户没有给出完全信息的字符串形式的过滤条件。

[0154] 本发明实施例还提供了一种逻辑过滤装置，包括：

[0155] 接收模块，用于接收待过滤数据流；

[0156] 逻辑过滤模块，用于采用多个属性或多个内容匹配节点对所述数据流中的元素类型节点进行过滤。

[0157] 本发明实施例还提供了一种子树过滤查询结果处理装置，包括：

[0158] 设置模块，用于在过滤器中相应节点处设置第三属性的值；

[0159] 确定模块，用于根据所述设置模块设置所述第三属性的值，表明所述节点处是否

显示节点后代。

[0160] 本发明的实施例中，提供一种基于NETCONF子树过滤机制，以XML形式表达，对XML网络管理数据进行查询和过滤，使对NETCONF协议中管理信息的获取更为方便，也避免了为数据查询和过滤重新定义一套规范带来的诸多问题。

[0161] 本发明实施例提供了一种对XML数据过滤条件的细化和组合机制。XML文档中的节点有时包含了具体的数据信息，类似于关系数据库中一个字段；有时只是封装数据信息的手段，类似于关系数据库中的一条记录。

[0162] 本发明实施例中细化了节点的匹配程度，用户可以根据节点在数据存储中的具体作用选择相应的查询方法。另外，属性、命名空间方式表达的过滤条件可以以逻辑运算方式组合到一起，使查询更为灵活。

[0163] 本发明实施例提供一种在过滤条件不完整的情况下，比如不能完整拼写出数据存储的节点或属性名字，或者不能完全给出数据在XML文档中定义的位置，仍可以对XML数据进行子树过滤操作，并得到用户想要的结果。

[0164] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解XML文档中数据分布的结构；弥补了NETCONF协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[0165] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[0166] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

附图说明

[0167] 为了更清楚地说明本发明实施例或现有技术中的技术方案，下面将对实施例或现有技术描述中所需要使用的附图作一简单介绍，下面描述中的附图仅仅是本发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

[0168] 图1是现有技术中一种树型数据结构示意图；

[0169] 图2是本发明实施例中NETCONF协议中子树过滤系统结构图；

[0170] 图3是本发明实施例中总体流程图；

[0171] 图4是本发明实施例中过滤控制模块工作流程图；

[0172] 图5是本发明实施例中过滤控制调用名字空间处理组件流程图；

[0173] 图6是本发明实施例中递归流程控制流程图；

[0174] 图7是本发明实施例中结果处理实现流程图；

[0175] 图8是本发明实施例二的基于子树过滤的模糊查询方法流程图；

[0176] 图9是本发明实施例三的子树过滤条件的逻辑组合的扩展方法流程图；

[0177] 图10是本发明实施例四的子树过滤查询结果的处理方法实现流程图；

[0178] 图11是本发明实施例五的过滤器结构示意图；

[0179] 图12是本发明实施例六的逻辑过滤装置结构示意图；

[0180] 图13是本发明实施例七的子树过滤查询结果处理装置结构示意图。

具体实施方式

[0181] 为了使本发明的目的、技术方案及优点更加清楚明白，以下结合附图及实施方式，对本发明进行进一步详细说明。应当理解，此处所描述的具体实施方式仅仅用以解释本发明，并不用于限定本发明。

[0182] 本发明实施例提供一套基于 NETCONF 协议并对 NETCONF 协议中子树过滤机制加以扩展的，实现对 XML 语言定义的网络管理信息模型的数据查询和数据过滤功能的定义方法。定义方法包括对于子树过滤功能的详细说明和与功能定义相关的关键字定义。

[0183] 为与NETCONF协议中子树过滤机制相区别，本发明实施例为<filter>节点的type属性新增取值advancedSubtree，表示其中可能含有进行子树过滤机制扩展功能操作的请求。

[0184] 子树过滤扩展机制主要包括以下功能：

[0185] 1. 模糊匹配

[0186] 模糊匹配是指在过滤流程中，处理节点名字、节点内容、属性名字、属性值、命名空间等字符串匹配中，以不完全匹配的方式处理用户没有给出完全查询信息的字符串。

[0187] 由于 XML 文档标签对于字符串有着较为严格的限制，所以节点名、属性名等标签内容与节点内容、属性值、命名空间等文本内容的模糊匹配手段是不一样的。对于标签内容来说，仅可使用字符“_”表示匹配任何字符。

[0188] 对于节点内容、属性值、命名空间等带有双引号的字符串，则进行字母匹配或者利用”*”匹配所有字符。

[0189] 例 10，例 11：过滤 <routing> 节点下具有某属性值为 Ethernet1 的 <interface> 节点。过滤中找到属性值 Ethernet1 对应的属性和属性所在节点，将所在节点、属性名字和属性值均输出，则过滤结果如例 11。

[0190] 例 10 属性名字的模糊匹配：

[0191] <filter type =“advancedSubtree”>

[0192] <netconfxmlns =“*”>

[0193] <routing>

[0194] <interface_ =″ Ethernet1″ />

[0195] </routing>

[0196] </netconf>

[0197] </filter>

[0198] 例 11 属性名字模糊查询的返回结果：

[0199] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0200] xmlns:if =″ urn:bupt:pris:priser:agent:module:interface:1.0″ >

[0201] <routing>

[0202] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0203] <../>

[0204] </if:interface>

[0205] </routing>

[0206] </netconf>

[0207] 上述实例中，过滤器中的过滤器<filter>节点指定了希望选择的数据模型中的<interface>节点的绝对路径，并通过在过滤器节点<interface>中设置属性＝"Ethernet1"实现了根据值查询属性的模糊查询功能。另外也可以根据属性名字查询值，只要将该<interface>中的属性改为name＝“*”(例如例12)，过滤中找到属性name的值和所在节点，将所在节点、属性名字和属性值均输出，则过滤结果与例11一致。

[0208] 例12属性值的模糊匹配：

[0209] <filter type＝" advancedSubtree”>

[0210] <netconf xmlns＝“*”>

[0211] <routing>

[0212] <interfacename＝“*”/>

[0213] <routing/>

[0214] <netconf/>

[0215] </filter>

[0216] 节点名字与值的模糊查询与之类似。如模糊了<name>的值和<name>自身的查询可如下所示：

[0217] <interface><name>*</name></interface>

[0218] 或者：

[0219] <interface><_>Ethernet1</_></interface>

[0220] 此外，还可以实现对节点值、属性值、命名空间内容进行不完全内容匹配的模糊过滤。如例13，例14。

[0221] 例13不完全内容匹配：

[0222] <filter type＝" advancedSubtree”>

[0223] <netconf xmlns＝“*”>

[0224] <routing>

[0225] <interface>

[0226] <mac-address>0006G*</mac-address>

[0227] </interface>

[0228] </routing>

[0229] </netconf>

[0230] </filter>

[0231] 例14不完全内容匹配的过滤结果：

[0232] <netconf xmlns＝" urn:ietf:params:xml:ns:netconf:base:1.0"

[0233] xmlns:if＝" um:bupt:pris:priser:agent:module:interface:1.0" >

[0234] <routing>

[0235] <if:interface name＝" Ethernet1" ip-address＝”59.64.139.65”>

[0236] <if:mac-address>0006GG93KAG8</if:mac-address>

[0237] <.../>

[0238] </if:interface>

[0239] </routing>

[0240] </netconf>

[0241] 上述实例 13、14 中，希望从数据模型中选择与过滤器指定路径相匹配并且其子节点 <mac-address> 的文本值与 0006G 部分匹配的 <interface> 节点。在过滤器中，<mac-address>0006G*</mac-address> 表示对内容进行非完全匹配。从过滤结果中看到，数据模型中存在节点 <interface>，该节点的子节点 <mac-address> 的值可以与过滤器中相应节点的值非完全匹配。

[0242] “*”代表一个非空白字符串，可以出现在节点值、属性值、命名空间字符串等以“”标记的字符串的开头、末尾或其他地方。

[0243] 由于 XML 文档规定，属性名字或者值不能单独出现，例如 xml 规定节点 <interface name> 为非法节点（此处 interface 为节点名字，name 为该节点的属性名字，属性值未知）。因此如果只知道属性名字或属性值，则在过滤器的节点中应该给出利用特殊字符匹配属性名字或属性值的形式如 attrName =“attrValue”的完整形式，如 <interfacename =“*”/> 或者 <interface_ =″ Ethernet1″ />，前者属性值未知，利用”*”来代替；后者属性名字未知，利用”_”来匹配。

[0244] 除此之外，标签内的特殊字符是不被允许的，只能使用“;&lt”之类的转义字符来替换，对于用户来说非常不友好。因此只允许”_”字符匹配所有可能。

[0245] 凡本文中提到的内置关键字及其可能的取值，不能够使用模糊匹配。

[0246] 2. 相对路径与跨层次访问

[0247] 协议中子树过滤采用绝对路径匹配：即过滤器、数据模型、过滤结果必须有相同的根节点，其他所有元素必须给出从根节点开始的所有祖先，过滤结果中只包含在过滤器中的祖先和在数据模型中的祖先完全一致的节点。

[0248] 为了提高过滤的灵活性，特增加相对路径匹配机制，即过滤器中的子节点对应的过滤结果节点未必是过滤器中父节点对应的过滤结果节点的子节点，也可能是其更深层次的后代；过滤器中的根节点未必是相应过滤结果中的根节点。

[0249] 相对路径过滤在 <filter> 节点定义属性相关关键字 _nodepath，且该属性 _nodepath 只能存在于 <filter> 节点之中。该属性的取值只有两个，一个是 absolute，表示过滤器中所有节点采用绝对路径过滤，即过滤结果中各个节点相互之间的关系必须与过滤器中各节点之间的关系严格一致，是该属性的默认选项（此时可以不使用 relative 属性）；另一个是 relative，表示采用相对路径过滤，即在过滤器中满足父子关系的节点可以在过滤结果中只满足祖先后代关系。

[0250] 实际处理中当数据模型节点满足不了绝对路径关系时，即在数据模型对应位置上的兄弟节点中找不到与过滤器节点匹配的节点时，再在兄弟节点的后代中寻找可以匹配的节点。如果仍然找不到，那么认为该过滤器节点不存在匹配节点。

[0251] 例如例 15，例 16 希望查询数据模型中的 <interface> 节点，但不知其在数据模型中的具体位置，所以无法在过滤器中给出该节点的绝对路径：

[0252] 例 15 相对路径过滤：

[0253] <filter type =“advancedSubtree”_nodepath =″ relative″ >

[0254] <interface xmlns =”*”/>

[0255] </filter>

[0256] 例 16 相对路径过滤的过滤结果：

[0257] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0258] xmlns:if =″ um:bupt:pris:priser:agent:module:interface:1.0″ >

[0259] <routing>

[0260] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0261] <.../>

[0262] </if:interface>

[0263] <if:interface name =″ Ethernet2″ ip-address =”59.64.139.69”>

[0264] <.../>

[0265] </if:interface>

[0266] </routing>

[0267] </netconf>

[0268] 上述实例 15、16 中，过滤器构造了一个过滤器 <filter>，该过滤器希望过滤出数据模型中所有的 <interface> 节点。由于过滤器未给定节点 <interface> 在数据模型中的路径，所以采用相对路径过滤，需要在过滤器 <filter> 中添加属性 _nodepath =″ relative″ 。

[0269] 在相对路径过滤中，如果有过滤器节点与数据模型节点不匹配，则并不直接退出对该节点的进一步处理，而是继续寻找数据模型中该节点后代是否有匹配于过滤器节点的节点。所以上述实例的过滤结果报文将输出数据模型中所有 <interface> 节点的所有父辈节点直至 <netconf> 根节点，并输出 <interface> 节点的后代节点。

[0270] 如果已知 interface 节点所在数据模型的根节点是 netconf，也可以这样改写例 15 为：

[0271] <filter type =“advancedSubtree”_nodepath =″ relative″ >

[0272] <netconf>

[0273] <interface/>

[0274] </netconf>

[0275] </filter>

[0276] 过滤结果与例 16 完全相同。

[0277] 相对路径过滤也可以结合模糊匹配，从而能够对属性名字或属性值的直接查询。如例 17 和例 18 查询具有 name 属性的节点和 name 属性可能的值：

[0278] 例 17 查询属性 name 的所有取值：

[0279] <filter type =“advancedSubtree”_nodepath =″ relative″ >

[0280] <_name =“*”/>

[0281] </filter>

[0282] 例 18 属性 name 的查询结果：

[0283] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0284] xmlns:if =″ urn:bupt:pris:priser:agent:module:interface:1.0″ >

[0285] <routing>

[0286] <iffinterface name =" Ethernet1" ip-address =”59.64.139.65”>

[0287] ……

[0288] </if:interface>

[0289] <iffinterface name =" Ethernet2" ip-address =”59.64.139.69”>

[0290] ……

[0291] </if:interface>

[0292] </routing>

[0293] </netconf>

[0294] 这个实例(例17、18)的过滤器构造了一个过滤器<filter>,希望从数据模型中过滤出所有拥有属性名为name、属性值为任意值的节点,另外由于需要过滤出所有符合该条件的节点,所以过滤器无法给出各个节点的绝对路径,则需要采用相对路径过滤,在过滤器<filter>中添加属性 _nodepath =" relative" 。

[0295] 从过滤结果可以看出,数据模型中name属性的值可能为Ethernet1或Ethernet2,而且在数据模型中具有name属性的节点只有<interface>节点,由于过滤要求只返回节点本身的信息,所以不会返回<interface>节点的后代节点的信息。

[0296] 3. 查询条件细化与组合

[0297] 在子树过滤中,可以采用多个属性或多个内容匹配节点对元素类型节点进行过滤。过滤条件只能完全符合或者相反,过滤条件之间默认为是“与”且只能是“与”的关系。而对于命名空间,逻辑关系“与”是不存在的。

[0298] 为了提高过滤条件设置的灵活性,特增加其他两种逻辑关系:或、非,并细分以元素节点方式定义的过滤条件的匹配程度。

[0299] 3.1 元素节点过滤条件的逻辑关系

[0300] 由于xml语言本身具有树型层次结构,可以较清晰的反应数据模型的树状层次结构。在不打乱该结构的情况下,通过在每个内容匹配节点中添加属性 _matchType来表示该节点的匹配程度,并在一定程度上间接地实现节点过滤条件之间的逻辑关系组合。

[0301] 注意,由于内容匹配节点的作用主要是作为从数据模型中选择特定的父节点的过滤条件,所以属性 _matchType只有出现在内容匹配节点才有意义,如果过滤器中该属性出现在选择节点或者容器节点时,则该属性并无多少意义。属性 _matchType的取值有三个:must(属性 _matchType的默认值),may和not。其中,must表示过滤器中内容匹配节点与数据模型中相应的节点必须完全匹配(即节点路径、节点名字、命名空间、属性、节点的文本内容等必须完全匹配)时,才将其父节点添加到过滤结果中,为默认项;may表示如果过滤器中的某个内容匹配节点与数据模型中相应的节点完全匹配(即节点路径、节点名字、属性,命名空间、节点文本内容等完全匹配)时,则将该内容匹配节点的父节点添加到过滤结果中;若不匹配,则在过滤该内容匹配节点的父节点(假设为节点<a>)时忽略该内容匹配节点,查看过滤器中是否还有其他的内容匹配节点与节点<a>的子节点相匹配;not表示如果过滤器中内容匹配节点与数据模型中相应的节点完全匹配(即节点路径、节点名字、命名空间、属性、节点的文本内容等必须完全匹配)时,则不添加该内容匹配节点的父节点到过滤结果中。

[0302] 例 19 内容匹配节点逻辑关系组合的过滤器：

[0303] 过滤包含节点 <name>Ethernet1</name> 并且包含节点 <received>49</received> 的节点 <interface>,则过滤器如下所示：

[0304] <filter type =“advancedSubtree”>

[0305] <netconf xmlns =" *" >

[0306] <routing>

[0307] <interface>

[0308] <name_matchType =“must”>Ethernet1</name>

[0309] <!--<name>Ethernet1</name>-->

[0310] <received_matchType =“must”>49</received>

[0311] <!--<received>49</received>-->

[0312] </interface>

[0313] </routing>

[0314] </netconf>

[0315] </filter>

[0316] 在上述实例 19 中,过滤器 <filter> 指定了节点 <interface> 在数据模型中的路径,并且要求只有同时满足拥有子节点 <name> 和 <received> 并且这两个子节点与过滤器中相应子节点的文本内容完全匹配的 <interface> 节点才会被添加到过滤结果中。由于数据模型中不存在与过滤器中指定的过滤条件相匹配的 <interface> 节点,所以过滤结果返回的报文是空。

[0317] 而过滤包含 <name>Ethernet1</name> 或者 <received>49</received> 的节点 <interface>,则过滤器和过滤结果分别如下例 20 所示：

[0318] 例 20 内容匹配节点之间“或”逻辑关系的实现：

[0319] <filter type =“advancedSubtree”>

[0320] <netconf xmlns =" *" >

[0321] <routing>

[0322] <interface>

[0323] <name_matchType =“may”>Ethernet1</name>

[0324] <received_matchType =“may”>49</received>

[0325] </interface>

[0326] </routing>

[0327] </netconf>

[0328] </filter>

[0329] 例 21 :例 20 的过滤结果：

[0330] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0331] xmlns:if =" um:bupt:pris:priser:agent:module:interface:1.0" >

[0332] <routing>

[0333] <if:interface name =" Ethernet1" ip-address =”59.64.139.65”>

[0334] <if:name>Ethernet1</if:name>

[0335] <.../>

[0336] </if:interface>

[0337] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0338] <if:received>49</if:received>

[0339] <.../>

[0340] </if:interface>

[0341] </routing>

[0342] </netconf>

[0343] 在上述实例 20 中,过滤报文构造了一个过滤器 <filter>,首先从数据模型中过滤出过滤器指定路径的 <interface> 节点,然后在这些 <interface> 节点中,那些拥有子节点 <name> 或者 <received> 或者两者都有,并且满足子节点 <name> 或者 <received> 的文本内容与过滤器中相应的节点完全匹配的 <interface> 节点才会被添加到过滤结果中。

[0344] 在本实例 21 中,过滤结果返回两个 <interface> 节点,第一个 <interface> 节点由于其子节点 <name> 与过滤器中指定的内容匹配节点 <name> 相匹配而被选中 ;第二个 <interface> 节点由于其子节点 <received> 与过滤器中指定的内容匹配节点 <received> 相匹配而被选中。

[0345] 例 22,例 23 演示了 _matchType =“not”的例子,查询希望输出 <name> 节点的值不是 Ethernet1 的 <interface> 节点,因此只有 <name> 值是 Ethernet2 的 <interface> 节点被输出。过滤器和过滤结果分别如下所示 :

[0346] 例 22 内容匹配节点的“非”过滤条件 :

[0347] <filter type =“advancedSubtree”>

[0348] <netconf xmlns =" *" >

[0349] <routing>

[0350] <interface>

[0351] <name_matchType =“not”>Ethernet1</name>

[0352] </interface>

[0353] </routing>

[0354] </netconf>

[0355] </filter>

[0356] 例 23 例 22 的过滤结果 :

[0357] <netconfximlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0358] xmlns:if =" um:bupt:pris:priser:agent:module:interface:1.0" >

[0359] <routing>

[0360] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0361] <if:name>Ethernet2</if:name>

[0362] <.../>

[0363] </if:interface>

[0364] </routing>

[0365] </netconf>

[0366] 例 24,例 25 显示了不在同一个层次的节点的逻辑关系过滤的过滤器和过滤结果：

[0367] 例 24 不同层次内容匹配节点逻辑关系组合：

[0368] <filter type =“advancedSubtree”>

[0369] <netconf xmlns =″ *″ >

[0370] <routing>

[0371] <interface>

[0372] <name_matchType =“may”>Ethernet1</name>

[0373] <ipv4>

[0374] <address-v4

[0375] _matchType =“may”>59.64.139.69</iaddress-v4>

[0376] </ipv4>

[0377] </interface>

[0378] </routing>

[0379] </netconf>

[0380] </filter>

[0381] 例 24 的结果：

[0382] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0383] xmlns:if =″ um:bupt:pris:priser:agent:module:interface:1.0″ >

[0384] <routing>

[0385] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0386] <if:name>Ethernet1</if:name>

[0387] <.../>

[0388] </if:interface>

[0389] <if:interface name =″ Ethernet2″ ip-address =”59.64.139.69”>

[0390] <if:ipv4>

[0391] <if:address-v4>59.64.139.69</if:address-v4>

[0392] <if:flags up =″ false″ />

[0393] </if:ipv4>

[0394] </if:interface>

[0395] </routing>

[0396] 上述实例 24 过滤出了数据模型中包含 <if:name>Ethernet1</if:name> 子节点的 <interface> 节点以及包含 <if:address-v4>59.64.139.69</if:address-v4> 子节点的 <ipv4> 节点。

[0397] 如果将 <address-v4> 的 _matchType 属性值设置为”must”,如下面的例 25、例 26,则生成的过滤器和过滤结果如下：

[0398] 例 25 内容匹配节点逻辑关系组合的过滤器：

[0399] <filter type =“advancedSubtree”>

[0400] <netconf xmlns =″ *″ >

[0401] <routing>

[0402] <interface>

[0403] <name_matchType =“may”>Ethernet1</name>

[0404] <ipv4>

[0405] <address-v4_matchType =“must”>59.64.139.69</iaddress-v4>

[0406] </ipv4>

[0407] </interface>

[0408] </routing>

[0409] </netconf>

[0410] </filter>

[0411] 例 26 例 25 的过滤结果：

[0412] <netconf xmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0413] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0" >

[0414] <routing>

[0415] <if:interface name =" Ethernet1" ip-address =”59.64.139.65”>

[0416] <if:name>Ethernet1</if:name>

[0417] <if:mac-address>0006GG93KAG8</if:mac-address>

[0418] <if:received>19</if:received>

[0419] </if:interface>

[0420] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0421] <if:ipv4>

[0422] <if:address-v4>59.64.139.69</if:address-v4>

[0423] <if:flags up =" false" />

[0424] </if:ipv4>

[0425] </if:interface>

[0426] </routing>

[0427] 上述实例 25 中，从数据模型中过滤出两个 <interface> 节点，第一个 <interface> 节点包含子节点 <name> 并且该子节点与过滤器 <filter> 中的内容匹配节点 <name> 完全匹配，但是由于它的子节点 <ipv4> 的子节点 <address-v4> 与过滤器中响应的内容匹配节点 <address-v4> 不匹配而且过滤器中内容匹配节点 <address-v4> 的 _matchType 属性值为 must，所以输出的 <interface> 后代节点中没有 <ipv4> 子节点；另一个 <interface> 节点，它的子节点 <ipv4> 的子节点 <address-v4> 与过滤器中相应的内容匹配节点完全匹配，所以输出该 <interface> 节点的子节点 <ipv4> 的全部信息。

[0428] 另外，允许用户对容器节点和选择节点应用属性 _matchType，但是在这两种类型的节点中该属性的值只能是”must”。如果用户在容器节点或选择节点中对 _matchType 属性赋予了其他值（如 not/may），则过滤机制将对在这两种类型节点中出现的 _matchType 属性忽略其值，因为其并无意义可用。

[0429] 3.2 节点属性和命名空间的逻辑关系过滤

[0430] 节点的多个属性间的逻辑关系可以通过向节点中添加属性 _attrLogic 来标示。

命名空间逻辑关系条件添加 _nsLogic 来标示。节点的普通属性（不包括命名空间）之间可以有逻辑与“ ^ ”、逻辑或“ ‖ ”（这两者优先级相等），以及逻辑非“ ! ”（优先级高于“ ‖ ”和“ ^ ”）的关系，并可以使用（ ）改变其优先级；而节点的命名空间之间只有逻辑或“ ‖ ”的关系，表示可以同时从多个命名空间中查找满足其过滤条件的元素。命名空间逻辑关系与默认命名空间一样，有继承关系。即除非子节点定义了 _nsLogic 属性新的值比如取 _nsLogic =“”表示逻辑关系无效，否则默认其与父节点有相同的命名空间逻辑组合过滤条件。

[0431] 在 _attrLogic 引出的逻辑表达式中，节点普通属性使用逻辑表达式所在节点的其他属性的名字标记，例如 name =”Ethernet1”，可以在同一节点定义属性 _attrLogic =”! name”，属性值包含“name”表示 name 属性；如果节点未定义属性 _attrLogic，那么按照 NETCONF 协议规定，各个属性之间是“与”的关系。即：

[0432] <interface name =″ Ethernet1″ ip-address =”59.64.139.69”/>

[0433] 与下述定义

[0434] <interface name =″ Ethernet1″ ip-address =” 59.64.139.69”

[0435] _attrLogic =“name ^ ip-address”/>

[0436] 等同。

[0437] 在 _nsLogic 引出的逻辑表达式中，命名空间使用逻辑表达式所在文档的命名空间前缀定义标记，例如文档中某处层定义 xmlns:if =”urn:bupt:pris:priser:agent:module:interface:1.0”，可以在需要多重命名空间过滤的节点处定义属性 _nsLogic =”if ‖ mt”，属性值包含“if”表示命名空间 urn:bupt:pris:priser:agent:module:interface:1.0；mt 代表了另一个命名空间；如果节点未曾定义属性 _nsLogic，那么按照 NETCONF 协议规定，过滤时将数据模型节点的命名空间与相应过滤器节点本身的命名空间匹配。

[0438] 如例 27，例 28 所示，过滤数据模型中属性 name 的值为 Ethernet1 或者 ip-address 值为 59.64.139.69 的 <interface> 节点。

[0439] 例 27 属性或逻辑关系组合查询：

[0440] <filter type =“advancedSubtree”>

[0441] <netconf xmlns =″ *″ >

[0442] <routing>

[0443] <interface name =″ Ethernet1″ ip-address =” 59.64.139.69”

[0444] _attrLogic =”name ‖ ip-address”/>

[0445] </routing>

[0446] </netconf>

[0447] </filter>

[0448] 例 28 例 27 的过滤结果：

[0449] <netconf xmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0450] xmlns:if =″ um:bupt:pris:priser:agent:module:interface:1.0″ >

[0451] <routing>

[0452] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0453] <... />

[0454] </if:interface>

[0455] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0456] <.../>

[0457] </if:interface>

[0458] </routing>

[0459] </netconf>

[0460] 从实例 27 中可以看到,由于使用了两个属性表达式,并用 _attrLogic 属性指出了两个属性表达式之间是“或”关系,因此,两个 <interface> 节点分别只满足了两个属性表达式中的一个,这两个 <interface> 节点也都被输出。

[0461] 如果修改过滤条件,查询 name 属性值不是”Ethernet1”并且 ip-address 属性值是”59.64.139.69”的 <interface> 节点,这个查询的表达方式如下 :

[0462] <interface name =" Ethernet1" ip-address =”59.64.139.69”_attrLogic =“!name ^ ip-address”/>

[0463] 还可使用括号表达属性过滤条件的逻辑关系,如选取属性 name 和 ip-address 全部满足或者全部不满足给定值的节点 :

[0464] <interface name =" Ethernet1" ip-address =”59.64.139.69”

[0465] _attrLogic =“(name ^ ip-address)‖(! name ^ ! ip-address)”/>

[0466] 例 29、例 30 是一个节点的多重命名空间逻辑或关系查询的过滤器和过滤结果 :

[0467] 例 29 节点多重命名空间查询 :

[0468] <filter type =“advancedSubtree”>

[0469] <netconf xmlns =“*”>

[0470] <routing>

[0471] <interface

[0472] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0473] xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0"

[0474] _nsLogic =“if‖mt”/>

[0475] <routing/>

[0476] <netconf/>

[0477] </filter>

[0478] 例 30 例 29 的过滤结果 :

[0479] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0480] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0481] xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0" >

[0482] <routing>

[0483] <if:interface name =" Ethernet1" ip-address =”59.64.139.65”>

[0484] <.../>

[0485] </if:interface>

[0486] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0487] <.../>

[0488] </if:interface>

[0489] </routing>

[0490] </netconf>

[0491] 上述实例 29 中,命令报文构造了一个过滤器 <filter>,希望从数据模型中过滤出与过滤器指定的节点路径相匹配,属于命名空间 xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0" 或者命名空间 xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0" 的 <interface> 节点。

[0492] 从实例 30 的返回的过滤结果可知,在数据模型中存在两个 <interface> 节点,这两个节点都属于命名空间 xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0" ,满足过滤器指定的节点路径,不存在属于命名空间 xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0" 的 <interface> 节点。

[0493] 4. 命名空间查询

[0494] 命名空间查询可以获取数据模型中命名空间的定义情况,并给出命名空间的列表。命名空间查询的关键字为标签元素 <_xmlns>,当过滤器 <filter> 的子节点为 <_xmlns>,则表示对该数据模型进行命名空间相关查询 ;同时定义了属性相关关键字 prf,该属性位于过滤返回结果的元素 <_xmlns> 中,表示该命名空间在此文档中对应的前缀,如果该命名空间是默认命名空间 (即无前缀),则不输出 prf 属性。

[0495] 例如例 31,例 32 分别显示了查询数据模型中所有命名空间的过滤器定义和过滤结果。

[0496] 例 31 查询所有命名空间 :

[0497] <filter type =“advancedSubtree”>

[0498] <_xmlns/>

[0499] </filter>

[0500] 例 32 例 31 的过滤结果 :

[0501] <data>

[0502] <_xmlns>um:ietf:params:xml:ns:netconf:base:1.0</_xmlns>

[0503] <_xmlns prf =”if”>

[0504] urn:bupt:pris:priser:agent:module:interface:1.0</_xmlns>

[0505] <_xmlns prf =”mt”>

[0506] um:bupt:pris:priser:agent:module:module:1.0</_xmlns>

[0507] </data>

[0508] 上述实例 31 通过关键字元素 <_xmlns> 告知过滤器,查询数据模型中存在的所有命名空间。过滤结果则以 <_xmlns>um:ietf:params:xml:ns:netconf:base:1.0</_xmlns> 的形式将数据模型中所有的命名空间返回。

[0509] 另外,还可以查询部分数据模型的命名空间,如例 33 所示 :

[0510] 例 33 查询部分元素的命名空间 :

[0511] <filter type =“advancedSubtree”>

[0512] <_xmlns>

[0513] <netconf>

[0514] <routing/>

[0515] </netconf>

[0516] </_xmlns>

[0517] </filter>

[0518] 过滤结果如下：

[0519] <_xmlns prf =”if”>urn:bupt:pris:priser:agent:module:interface:1.0</_xmlns>

[0520] 上述实例中，过滤器 <filter> 通过设置子节点 <_xmlns> 告知过滤器查询数据模型的命名空间，同时通过设置节点 <netconf><routing/></netconf> 告知过滤器具体查询的是子树 /netconf/routing 的命名空间，所以过滤结果中只返回了数据模型中 <netconf> 和 <routing> 节点的命名空间“urn :bupt :pris :priser :agent :module :interface:1.0”。

[0521] 5. 后代输出控制

[0522] 当过滤器给出一个元素类型节点，过滤结果将包括该节点的所有匹配后代；某些时候，这将返回大量无用的数据。

[0523] 为了提高过滤的效率及方便用户获取所需信息，可以在过滤器中相应节点处设置属性 _show 表明是否显示节点后代。该属性的取值为 self 或 descendant。该属性默认为显示所有在同一命名空间的后代，即 _show =“descendant”；当设置 _show =“self”，则过滤结果只输出该节点的信息，不输出其后代节点。属性 _show 只能位于包含节点中。

[0524] 对于内容匹配节点一类的叶子节点来说，输出其本身或输出后代并无区别。

[0525] 例如例 34、例 35、例 36、例 37 分别显示了如何过滤出指定节点的完整信息和部分信息。

[0526] 例 34 输出节点完整信息的过滤器：

[0527] <filter type =“advancedSubtree”_nodepath =″ relative″ >

[0528] <interface_show = descendant/>

[0529] </filter>

[0530] 例 35 例 34 的过滤结果：

[0531] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0532] xmlns:iff =″ um:bupt:pris:priser:agent:module:interface:1.0″ >

[0533] <routing>

[0534] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0535] <.../>

[0536] </if:interface>

[0537] <if:interface name =″ Ethernet2″ ip-address =”59.64.139.69”>

[0538] <.../>

[0539] </if:interface>

[0540] </routing>

[0541] </netconf>

[0542] 例 36 只输出节点本身信息：

[0543] <filter type =“advancedSubtree”_nodepath =″ relative″ >

[0544] <interface_show =”self”/>

[0545] </filter>

[0546] 例 37 例 36 的过滤结果：

[0547] <netconf xmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0548] xmlns:if =″ urn:bupt:pris:priser:agent:module:interface :1.0″ >

[0549] <routing>

[0550] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”/>

[0551] <if:interface name =″ Ethernet2″ ip-address =”59.64.139.69”/>

[0552] </routing>

[0553] </netconf>

[0554] 对比上述实例 37 的过滤结果，可以明显看到当在过滤器 <filter> 中的节点 <interface> 中设置属性 _show =”self”时，则只会输出节点本身的信息，可以避免返回过多的冗余信息，提高过滤效率。

[0555] 6. 个数查询

[0556] 个数查询功能可以统计数据模型中某一类元素的个数，即同一过滤器节点对应的元素个数。通过在过滤器中某元素节点中使用 _count 属性实现，该属性的取值有两个：true 或 false。当设置 _count 属性值为”true”时，则过滤结果输出查询实体的个数；当设置 _count 属性为”false”时，则过滤结果不输出查询实体的个数，”false”是属性 _count 的默认值。如果过滤结果返回查询实体的个数，则在过滤结果中相应实体元素节点下面添加标签元素 <_countNum>，该元素的文本值标识查询实体的个数。

[0557] 例如例 38、例 39 显示了如何查询 <monitor> 节点的子节点 <configuration> 的个数，过滤器和过滤结果如下：

[0558] 例 38 查询指定节点的个数：

[0559] <filter type =“advancedSubtree”>

[0560] <netconf xmlns =”*”>

[0561] <monitor xmlns =”um:bupt:pris:priser:agent:module:monitor :1.0”>

[0562] <configuration_count =“true”/>

[0563] </monitor>

[0564] </netconf>

[0565] </filter>

[0566] 例 39 例 38 的过滤结果：

[0567] <netconf>

[0568] <monitor xmlns =”um:bupt:pris:priser:agent:module:monitor:1.0”>

[0569] <configuration>

[0570] <_countNum>3</_conutNum>

[0571] </configuration>

[0572] </monitor>

[0573] </netconf>

[0574] 上述实例 38 的过滤器 <filter> 设置为在数据模型中查询出指定路径的节点 <configuration> 的个数。实例 39 的过滤结果显示，数据模型中的 <monitor> 节点的子节点 <configuration> 的个数是 3。

[0575] 个数查询也可以满足诸如查询属性 name 值为 Ethernet1 或者节点值为 running-state 的节点个数这样带有限制条件的节点个数统计：

[0576] <interface name =”Ethernet1”_count =“true”>

[0577] <configuration_count =“true”>running-state</running>

[0578] 7. 范围条件

[0579] 用于选择具有某些特定数值或者日期范围的节点。由于标签内不能使用“<”或“>”之类的符号，如果在文本内容中添加“<”或“>”，将会和标签的开始符和结束符混淆，形成非法的 xml 文档。所以，在表示数量或时间的选择节点中添加表示属性范围或时间范围的属性，以此将该选择节点看作是表示时间或数量范围的内容匹配节点，从而在过滤处理时作为过滤条件。

[0580] 其中表示数量范围的属性关联关键字有：_morethan，表示该属性所在的选择节点的值大于该属性的值；_lessthan，表示该属性所在的选择节点的值小于该属性的值；_notmorethan，表示该属性所在的选择节点的值不大于该属性的值；_notlessthan，表示该属性所在的选择节点的值大于该属性的值；注意使用属性的过滤器节点所对应的数据模型节点必须可转化为数值类型，比如“14”、“23.6”等，否则会认为是不匹配的节点。

[0581] 同时，定义了范围属性条件但没有文本内容的过滤器节点同样是内容匹配节点。

[0582] 表示时间范围的属性关联关键字有：_timebefore：表示该属性所在的选择节点的时间值超前于该属性的值；_timeafter：表示该属性所在的选择节点的时间值滞后于该属性的值；_timenotbefore：表示该属性所在的选择节点的时间值等于或滞后于该属性的值；_timenotafter：表示该属性所在的选择节点的时间值等于或超前于该属性的值。注意使用属性的过滤器节点所对应的数据模型节点必须可转化为时间类型，即规范满足 2007-07-0105:24:06 的格式，以满足网络管理中对时间的存储读取需求。

[0583] 例如欲查询 <received> 的值大于 12 且小于 25 的 <interface> 节点，过滤器表达如例 40 所示，其中 <received_morethan =”12”_lessthan =”25”/> 表达了这一范围条件。例 41 显示该查询的结果。

[0584] 例 40 数值范围过滤的过滤器：

[0585] <filter type =“advancedSubtree”>

[0586] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0587] xmlns:if =″ urn:bupt:pris:priser:agent:module:interface:1.0″ >

[0588] <routing>

[0589] <if:interface>

[0590] <received_morethan =”12”_lessthan =”25”/>>

[0591] </if:interface>

[0592] </routing>

[0593] </netconf>

[0594] </filter>

[0595] 例 41 例 40 的过滤结果：

[0596] <netconfxmlns ＝″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0597] xmlns:if ＝″ um:bupt:pris:priser:agent:module:interface:1.0″ >

[0598] <routing>

[0599] <if:interface name ＝″ Ethernet1″ ip-address ＝″ 59.64.139.65″ >

[0600] <if:received>19</if:received>

[0601] <.../>

[0602] </if:interface>

[0603] </routing>

[0604] </netconf>

[0605] 下面是根据时间范围条件查询的例子：

[0606] <startTime_timenotafter ＝”2007-07-0105:24:06”_timeafter ＝”2007-06-3005:24:06”/>

[0607] 表示查询表示时间晚于但不包括 2007 年 06 月 30 日 05:24:06，并早于且包括 2007 年 07 月 01 日 05:24:06 的 <startTime> 节点。

[0608] 或者添加字符串匹配功能：

[0609] <startTime_timeafter ＝”2005-06-3005:24:06”>*-07-*</startTime>

[0610] 符合 -07- 格式的只有月份的表达，上述定义可以查询 2005 年 06 月 30 日 05:24:06 以后每年 7 月的相关数据。

[0611] 8. 子树排序合并

[0612] 对同一个过滤器节点得到的过滤结果按照需要进行排序，如根据某个节点内容进行升序或者降序排列，合并查询结果中内容完全相同的节点。

[0613] 排序通过在要排序子树根节点处设定属性 _ascOrder 和 _descOrder 实现。属性 _ascOrder 表示根据某节点或属性做升序排列，属性 _descOrder 表示根据某节点或属性做降序排列。这两个属性的取值为排序依据的节点名或属性名。排序所依据的节点应当有值，并且是该属性所在节点的子节点。以属性进行排序时需要在属性名前加“@”，标识这是一个属性名。节点值为排序依据的索引，默认按文本方式排序；如果按数值大小排序，在节点或属性名后加标识“(n)”，比如 _ascOrder ＝“recived(n)”；如果该节点或属性的值为非数值类型，则向用户返回错误信息。

[0614] 当需要以多个节点、属性进行排序时，多个节点名、属性名间以“ ^ ”连接，排在前面的节点、属性名字优先级高于后面的节点、属性名字。

[0615] 例如，希望按 <name> 节点的值以降序的顺序输出 <interface> 节点，具有相同 <name> 值的 <interface> 节点再按 ip 属性排序。这样的查询入例 42 所示、其中 _descOrder ＝”name ^ @ ip”表达了这个排序要求。例 43 显示了排序的结果。

[0616] 例 42 按指定节点值降序输出元素：

[0617] <filter type ＝“advancedSubtree”>

[0618] <netconfxmlns ＝ ″ urn:ietf:params:xml:ns:netconf:bas

e:1.0″ >

[0619] <routing>

[0620] <interface

[0621] xmlns =″ urn:bupt:pris:priser:agent:module:interface:1.0″

[0622] _descOrder =”name ^ @ ip”/>

[0623] </routing>

[0624] </netconf>

[0625] </filter>

[0626] 例 43 :例 42 的过滤结果如下所示 :

[0627] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0628] xmlns:if =″ um:bupt:pris:priser:agent:module:interface:1.0″

[0629] xmlns:mt =″ urn:bupt:pris:priser:agent:module:monitor:1.0″ >

[0630] <routing>

[0631] <if:interface name =″ Ethernet2″ ip-address =”59.64.139.69”>

[0632] <if:name>Ethernet2</if:name>

[0633] ……

[0634] </if:interface>

[0635] <if:interface name =″ Ethernet1″ ip-address =”59.64.139.65”>

[0636] <if:name>Ethernet1</if:name>

[0637] ……

[0638] </if:interface>

[0639] </routing>

[0640] </netconf>

[0641] 由于对同一个过滤器节点可能得到许多相同结构、相同内容的子树,所以可以在排序子树根节点处设定属性 _merge 来实现相同子树的合并。该属性的取值为 :true 或 false。当 _merge =“true”,则表示对过滤结果进行相同子树合并后再返回 ;当 _merge =“false”,表示对过滤结果不进行子树合并,属性 _merge 的默认值是”false”。

[0642] 假设数据模型中 <interface> 节点不具有任何属性,那么将对于下述报文 :例 44 未使用合并的过滤器 :

[0643] <filter type =“advancedSubtree”>

[0644] <netconf xmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″ >

[0645] <routing>

[0646] <interface

[0647] xmlns = ″ urn:bupt:pris:priser:agent:module:interface:1.0 ″ _show =”self”/>

[0648] </routing>

[0649] </netconf>

[0650] </filter>

[0651] 例 45 未使用合并的过滤结果 :

[0652] <netconfxmlns ＝″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0653] xmlns:if ＝″ urn:bupt:pris:priser:agent:module:interface:1.0″

[0654] xmlns:mt ＝″ urn:bupt:pris:priser:agent:module:monitor:1.0″ >

[0655] <routing>

[0656] <if:interface/>

[0657] <if:interface/>

[0658] </routing>

[0659] </netconf>

[0660] 此时用户实际上只想知道数据模型中是否 interface 节点的存在而不关心其他。如果使用 _merge 属性，<interface> 节点修改为 <interface xmlns ＝″ urn:bupt :pris : priser :agent :module :interface :1.0″ _show ＝“self”_merge ＝”true”/>，过滤结果变为：

[0661] 例 46 合并后的过滤结果：

[0662] <netconfxmlns ＝″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0663] xmlns:if ＝″ urn:bupt:pris:priser:agent:module:interface:1.0″

[0664] xmlns:mt ＝″ um:bupt:pris:priser:agent:module:monitor:1.0″ >

[0665] <routing>

[0666] <iff :interface/>

[0667] </routing>

[0668] </netconf>

[0669] 多个相同的 <interface> 节点（或者说子树）中压缩只输出一个，这在数据模型中具有大量同名节点的情况下，可能会避免大量冗余信息在管理网络中传输的弊病。

[0670] 综上，本发明扩展功能的定义规范总结如表 1 所示：

[0671] 表 1 子树过滤功能定义

[0672]

实现功能	实现方法	标识符	取值	用法/含义
子树过滤扩展	为<filter>节点的type属性扩展新取值	type	advancedSubtree	表明采用本发明增强的子树过滤模式
模糊匹配	定义通配符，通配符可匹配元素名、元素值、属性名、属性值、名字空间中的任意文字	_		匹配任意的节点和属性名
		*		匹配名字空间、节点值和属性值中的任意文字
路径匹配	为<filter>节点扩展新属性	_nodepath	absolute	对于节点的路径严格按绝对路径匹配。属性默认值
			relative	对于节点的路径不按绝对路径匹配

[0673]

内容匹配节点的逻辑关系	为内容匹配节点扩展新属性	_matchType	must	多个内容匹配节点间是“与”的关系。属性默认值
			may	多个内容匹配节点间是“或”的关系
			not	对属性所在的内容匹配节点执行“非”运算
属性间逻辑关系	为执行属性过滤的节点扩展新属性	_attrLogic	该节点的属性的逻辑表达式，逻辑运算符包括: ^、||、!、(、)	被过滤出的节点的属性应满足该逻辑表达式
命名空间逻辑关系	为元素节点扩展新属性	_nsLogic	名字空间前缀的逻辑表达式，逻辑运算符包括: ||	被过滤出的节点应属于逻辑表达式所指示的名字空间
命名空间查询	扩展新元素	<_xmlns>	无子节点或其子节点为一子树	查询子树归属的命名空间
后代输出控制	为选择节点和包含节点扩展新属性	_show	self	仅仅输出该属性所在节点自身
			descendant	输出该属性所在节点的自身及后代。属性默认值
个数查询	为元素节点扩展新属性	_count	true	查询满足过滤条件的某节点的个数
			false	查询满足过滤条件的某节点的内容。属性默认值
	为查询应答结果扩展新元素	<_countNum>	节点的个数，为正整数	为<data>元素的子元素，携带匹配节点的个数
数值范围过滤	为数值数据类型的元素扩展新属性	_morethan	数值	数值节点值大于某数值
		_lessthan	数值	数值节点值小于某数值
		_notmorethan	数值	数值节点值小于等于某数值
		_notlessthan	数值	数值节点值大于等

[0674]

				于某数值
时间范围过滤	为时间数据类型的元素扩展新属性	_timebefore	符合时间规范表达的字符串	超前于某设定时间
		_timeafter	符合时间规范表达的字符串	滞后于某设定时间
		_timenotbefore	符合时间规范表达的字符串	等于或滞后于某设定时间
		_timenotafter	符合时间规范表达的字符串	等于或超前于某设定时间
子树排序	为节点扩展新属性	_ascOrder	所在节点的某些后代节点的名字，以与运算符号"^"连接	按属性所在节点升序排列
		_descOrder	所在节点的某些后代节点的名字，以与运算符号"^"连接	按属性所在节点降序排列
子树合并	为节点扩展新属性	_merge	true	合并完全相同的节点
			false	不合并完全相同的节点

[0675] 带有"<>"表示该关键字以元素的形式存在；不带有"<>"而只带有"_"表示该关键字以属性的形式存在；两者皆不带有表示该关键字以属性值或节点值形式存在。

[0676] 除此之外，本发明对 NETCONF 协议子树过滤机制中的以下处理情形进行了修正：

[0677] 1. 舍弃不匹配节点的祖先

[0678] 如果有下例过滤请求：

[0679] 例 47 不匹配节点的查询：

[0680] <filter type =" subtree" >

[0681] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0" >

[0682] <routing>

[0683] <interface name =" Ethernet"

[0684] xmlns =" urn:bupt:pris:priser:agent:module:interface:1.0" />

[0685] <routing/>

[0686] <netconf/>

[0687] </filter>

[0688] 而数据模型中没有一个属性 name 值为 Ethernet 的 interface 节点。那么在 NETCONF 协议子树过滤机制中，NETCONF 协议中得到的过滤结果如下：

[0689] 例 48NETCONF 协议中对不匹配节点的过滤结果：

[0690] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0691] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0692] xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0" >

[0693] <routing/>

[0694] </netconf>

[0695] 由于数据模型中,不存在匹配的 interface 节点,故而不输出 interface 节点,但返回其祖先 routing 和 netconf.。对于用户来说,过滤结果含义表达并不明确,也带来了无用的冗余信息。

[0696] 本发明中,当所有需要匹配的节点没有匹配数据模型节点,因而全部被舍弃时,这些节点的父节点也不返回。因而例 47 的过滤会返回空的过滤结果 :<data></data>

[0697] 例 49 扩展功能中不匹配节点的过滤结果

[0698] 2. 命名空间过滤

[0699] 由于 NETCONF 协议中的子树过滤机制将具有相同父节点的节点(即兄弟节点集合)集中在一起进行处理,从根节点到叶子节点。因此一旦匹配失败,会将不匹配节点和其所有子节点全部删除。例如在 NETCONF 协议中下述过滤请求 :

[0700] 例 50 :NETCONF 协议中命名空间过滤 :

[0701] <filter type =" subtree" >

[0702] <netconf xmlns = " urn:bupt:pris:priser:agent:module:interface:1.0" />

[0703] </filter>

[0704] 处理根节点 netconf 时,节点的名字匹配,但命名空间不匹配,于是丢弃 netconf 节点,从而得到空过滤结果 :

[0705] 例 51 :NETCONF 协议中命名空间过滤结果 :

[0706] <data></data>

[0707] 而本发明实施例中,当出现节点的命名空间不匹配时,仍然继续处理其子节点,因为可能存在其后代与给定的命名空间匹配。同时由于要保证过滤结果的层次关系仍然有效,故也可能保留命名空间不匹配的元素作为其他节点的路径。用户可以从过滤结果中很清楚的分辨出这些带有其他命名空间的元素。

[0708] 例 50 在子树过滤功能扩展中得到的过滤结果如下 :

[0709] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0710] xmlns:if =" urn:bupt:pris:priser:agent:module:interface:1.0"

[0711] xmlns:mt =" urn:bupt:pris:priser:agent:module:monitor:1.0" >

[0712] <routing>

[0713] <if:interface name =" Ethernet1" ip-address =”59.64.139.65”>

[0714] ……

[0715] </if:interface>

[0716] <if:interface name =" Ethernet2" ip-address =”59.64.139.69”>

[0717] ……

[0718] </if:interface>

[0719] </routing>

[0720] </netconf>

[0721] 例 52 扩展功能中命名空间过滤结果

[0722] 实现这种机制后，可以很容易的查找文档内或某节点下所有属于某命名空间的元素。例如查询命名空间 urn:bupt:pris:priser:agent:module:interface:1.0 中的元素，直接将该命名空间作为过滤器中根节点的命名空间，虽然在数据模型中该命名空间与根节点的命名空间不同。

[0723] 本发明实施例提供一套基于并对 NETCONF 协议中子树过滤机制加以扩展的，实现对 XML 语言定义的网络管理信息模型的数据查询和数据过滤功能的处理系统。处理系统详细说明了系统实施的总体结构和结构中每个模块的具体功能。

[0724] 本发明扩展后的 NETCONF 子树过滤系统主要体系架构如图 2 所示。该系统包括过滤控制模块、报文设置模块、名字空间处理模块、相对路径访问模块、递归流程控制模块、节点功能处理模块和结果处理模块。

[0725] 各模块的主要功能如下：

[0726] 1. 过滤控制模块

[0727] 过滤控制模块是整个子树过滤部分的核心，为整个子树过滤过程提供控制机制，同时可以根据报文中的参数执行协议子树过滤和 XPath 过滤操作。该模块解析 rpc 报文中过滤相关部分，并且根据报文的具体内容按顺序调用其他模块完成相应功能，相应模块功能处理完毕后需向其汇报处理结果并由过滤控制模块决定如何进行下一步的操作。在获取初始过滤结果后，如果报文中有对过滤结果进行处理的要求，此模块即调用结果处理模块对初始过滤结果进行处理，获得最终过滤结果并返回。

[0728] 2. 报文设置模块

[0729] 即 rpc 报文，rpc 报文中包含过滤控制模块所需的过滤信息，过滤控制模块首先判断 rpc 报文的正确性，如果正确则根据报文的信息设置具体的参数，包括过滤类型（子树过滤、扩展子树过滤、Xpath）和在扩展子树过滤中是否需要相对路径匹配。并将结果返回给过滤控制模块。

[0730] 3. 名字空间处理模块

[0731] 名字空间处理模块用于处理与名字空间查询相关的操作请求，例如查询配置文档中所有节点的名字空间、查询某个子树中所有节点的名字空间等，在实现中需要递归流程模块和结果处理模块的支持。

[0732] 4. 相对路径访问模块

[0733] 相对路径访问模块用于当 rpc 消息中给出的是节点的相对路径时，对配置文档进行过滤。因此避免了在协议中的条件下必须给出节点完整路径的限制，可以在不给出节点完整路径的情况下进行过滤。主要功能是为递归流程模块找到合适的配置文档节点作为参数。

[0734] 5. 递归流程控制模块

[0735] 将配置文档由顶层到底层划分，一次处理过滤器中一群兄弟节点，将其分为容器节点、选择节点、内容匹配节点进行相关处理；每当添加节点到过滤结果时，对其进行节点功能处理。

[0736] 6. 节点功能处理模块

[0737] 节点功能处理模块包括：模糊匹配模块、节点数量查询模块、数值范围查询模块、时间范围查询模块和逻辑关系组合模块，用于在进行过滤时对过滤器文档中的节点和数据模型中的节点的判断比较，并给出判断结果。

[0738] 7. 结果处理模块

[0739] 结果处理模块是对过滤结果进行操作，具体包括报文封装模块、节点排序模块和节点合并模块。当过滤器为名字空间过滤时，结果处理模块需要对过滤结果进行封装，将所有的名字空间封装成 xml 的格式返回。当过滤器报文中要求对过滤结果进行排序时，结果处理部分根据排序的关键字对初始结果进行处理，获得排序后的结果并提交给过滤控制模块。节点合并用于对初始过滤结果中相同子树的合并。

[0740] 本发明实施例提供一套基于并对 NETCONF 协议中子树过滤机制加以扩展的，实现对 XML 语言定义的网络管理信息模型的数据查询和数据过滤功能的处理方法。该处理方法包括扩展后的 NETCONF 子树过滤实施的总体流程和具体模块的详细流程图。

[0741] 图 3 为本发明实施例的总体流程图，主要包括报文检错、根据关键字设置运行参数、之后将给出的过滤器对 Agent 本地保留的数据模型进行过滤，遇到节点调用节点处理模块，待所有节点处理完毕后，对过滤结果做进一步处理，具体包括以下步骤：

[0742] 步骤 3010，开始；

[0743] 步骤 3020，解析 rpc 报文；

[0744] 步骤 3030，判断 rpc 报文设置是否正确，如果正确则转步骤 3050，如果不正确则转步骤 3040；

[0745] 步骤 3040，报错；

[0746] 步骤 3050，判断关键字，以确定是命名空间过滤，还是跨层次访问；如果是命名空间过滤，则转步骤 3060，如果是跨层次访问，则转步骤 3070；

[0747] 步骤 3060，调用命名空间过滤，转步骤 3080；

[0748] 步骤 3070，调用跨层次过滤，转步骤 3080；

[0749] 步骤 3080，调用递归流程控制；

[0750] 步骤 3090，调用节点功能处理；

[0751] 步骤 3100，根据关键字，对过滤结果进行处理；

[0752] 步骤 3110，结束。

[0753] 1. 过滤控制

[0754] 过滤控制模块是整个子树过滤流程的管理者，它调用其他组件完成工作，并接收其他组件工作的结果来决定下一步的工作内容。其主要流程为如图 4 所示，包括以下步骤：

[0755] 步骤 4010，调用图 2 中的报文设置模块；

[0756] 步骤 4020，判断是否采用扩展子树过滤，如果是则转步骤 4030，如果否，则判断查询对象类型是否为不同配置数据存储中的配置数据，如果是则采用 Xpath 过滤，如果否，则判断查询对象类型是否为当前正在运行的配置和状态数据，如果是，则采用协议子树过滤；

[0757] 步骤 4030，根据 rpc 报文判断报文的正确性，如果正确则转步骤 4060，否则转步骤 4100；

[0758] 步骤 4060，进行名字空间查询，如果查到，则转步骤 4070，否则，转步骤 4072 ；

[0759] 步骤 4070，调用名字空间处理模块 ；

[0760] 步骤 4072，根据关键字判断是否采用跨层次过滤，如果是，则转步骤 4075，否则转步骤 4080 ；

[0761] 步骤 4075，调用跨层次过滤模块，转步骤 4080 ；

[0762] 步骤 4080，调用递归流程控制模块，开始进行子树过滤，转步骤 4090 ；

[0763] 步骤 4090，对过滤结果进行相关处理，转步骤 4100 ；

[0764] 步骤 4100，将操作结果封装报文返回给相关操作。

[0765] 2. 名字空间处理

[0766] 当 filter 的根节点为 <_xmlns> 时，过滤控制调用名字空间处理组件，声明其报文仅是名字空间的过滤条件。返回结果只包含名字空间的信息。实现方法如图 5 所示，包括以下步骤 ：

[0767] 步骤 5001，获取报文定义中 <_xmins> 元素的子节点 ；

[0768] 步骤 5010，判断是否 <_xmins> 无子节点，如果是，则转步骤 5020，否则转步骤 5030 ；

[0769] 步骤 5020，将配置文档整体作为过滤初始结果，转步骤 5040 ；

[0770] 步骤 5030，执行子树过滤递归 ；

[0771] 步骤 5040，获得节点的命名空间插入集合 ；

[0772] 步骤 5050，判断是否所有节点获取完毕，如果是，则转步骤 5060，否则转步骤 5030。

[0773] 步骤 5060，封装获取内容返回给过滤控制模块。

[0774] 3. 递归流程控制

[0775] 递归流程控制与协议中实现方法基本类似，流程图如图 6 所示，包括以下步骤 ：

[0776] 步骤 6001，解析报文设置该节点的操作参数 ；

[0777] 步骤 6010，节点判断本身信息是否匹配，如果匹配，则转步骤 6020，如果不匹配，则转步骤 6015 ；其中，节点本身的信息包括节点类型、节点名字、节点属性、节点名字空间等，节点属性和节点名字空间可能有多种逻辑关系 ；

[0778] 步骤 6015，该节点判断是否有相对路径，如果有则转步骤 6017 ；

[0779] 步骤 6017，寻找相对路径上的节点，转步骤 6090 ；

[0780] 步骤 6020，该节点判断与内容匹配节点是否匹配，如果是，则转步骤 6030，否则，转步骤 6100 ；其中，内容匹配节点包括具有文本值的节点和具有数据范围等其他条件的节点，同样可能会存在多种逻辑关系，并可能存在扩展查询子句 ；

[0781] 步骤 6030，创建符合要求的当前节点副本，进入步骤 6040 ；

[0782] 步骤 6040，在过滤结果中添加内容匹配节点，进入步骤 6050 ；

[0783] 步骤 6050，处理选择节点，选择节点包括过滤器中不含子节点的节点及子节点为内置关键字如 <_count> 等。如果选择节点设定为不显示其子孙，则只复制本身 ；如果设定为不显示祖先，则将选择节点复制到结果根节点下 ；进入步骤 6060

[0784] 步骤 6060，获取容器节点列表 ；

[0785] 步骤 6070，判断容器节点和选择节点是否都为空，如果是，则转步骤 6080，否则，

转步骤 6090 ；

[0786] 步骤 6080，将其它节点复制到当前节点副本下，转步骤 6100 ；

[0787] 步骤 6090，递归调用处理容器节点，转步骤 6100 ；

[0788] 步骤 6100，退出当前流程，返回上一级递归；返回结果包括符合过滤条件的当前节点及其子孙，同时也包括由于保留其他节点而必须保留的部分不符合过滤条件的节点，不包括对用户来说没有意义的节点。

[0789] 4. 节点功能处理

[0790] 节点功能处理模块接受递归流程控制的调用，实现针对单个节点的处理功能。该节点功能处理模块从过滤器定义中某一个节点出发，保留配置文档中符合该节点代表的过滤条件的节点，并根据其表述的要求对保留的配置文档节点做一些处理。

[0791] 该节点功能处理模块的功能包括：节点的匹配、选择文本内容为特定数值的节点、选择文本内容为特定时间的节点、子节点个数查询、属性及命名空间或与非逻辑的运算，查找与当前节点相似但是路径不完全的配置文档节点。

[0792] 该节点功能处理模块接受递归控制部分传递的参数，进行相应的处理，并且将处理结果返回给递归控制模块。

[0793] 主要实现方法如下：

[0794] (1) 模糊匹配：

[0795] 判断配置文档和过滤器文档中两个节点的属性是否模糊匹配。首先将文本节点的内容进行适当预处理，然后根据正则表达式进行判断是否匹配。

[0796] (2) 选择文本内容为特定数值范围的节点：

[0797] 首先判断文本内容是否可以转换成数值类型，若是则继续判断数值是否在指定的数值范围内，数值范围的判断包括 4 个部分：大于、小于、不大于和不小于。

[0798] (3) 选择文本内容为特定时间范围的节点：

[0799] 首先判断文本内容是否可以转换成表示时间的数据，若是则继续判断数据是否在指定的时间范围内，时间范围的判断包括 4 个部分：早于、晚于、不早于和不晚于指定的时间。

[0800] (4) 子节点个数查询：

[0801] 设置节点计数器，当寻找到匹配节点时，计数器加 1；将节点划归到过滤结果文档时只复制节点本身及其个数。

[0802] (5) 查找与当前节点相似但是路径不完全的配置文档节点：

[0803] 根据节点部分特性在当前节点后代中查找节点，并与当前节点进行匹配，供相对路径功能模块使用。

[0804] (6) 属性、命名空间逻辑关系：

[0805] 首先确定过滤器中给出的各命名空间、属性，配置文档节点是否符合。以 true 或 false 表示。然后计算布尔类型表达式的值。

[0806] 5. 结果处理模块实现以下功能：

[0807] 报文封装：报文封装主要是将过滤结果根据过滤控制中设置的参数封装为适当的报文格式发回给过滤控制；

[0808] 节点合并：合并具有相同结构的过滤结果；

[0809] 节点排序：对过滤结果根据给定节点的值进行排序，并将排序后的结果返回。结果处理的基本流程如图 7 所示，包括以下步骤：

[0810] 步骤 7010，判断过滤器是否为空，如果是，则转步骤 7020；

[0811] 步骤 7020，拷贝整个配置文档；

[0812] 步骤 7030，判断是否进行命名空间查询，如果是，则转步骤 7040；

[0813] 步骤 7040，检索命名空间和前缀；

[0814] 步骤 7050，进行节点排序或进行节点合并；

[0815] 步骤 7060，封装报文返回。

[0816] 本发明实施例一，

[0817] 假设 Agent 端管理的数据模型如下例 53 数据模型所示：

[0818] <netconfxmlns ＝" urn:ietf:params:xml:ns:netconf:base:1.0"

[0819] xmlns:mt ＝" urn:bupt:pris:priser:agent:module:monitor:1.0"

[0820] xmlns:if ＝ " urn:bupt:pris:priser:agent:module:interface:1.0 " >

[0821] <routing>

[0822] <if:interface name ＝" Ethternet1" ip ＝" 59.64.139.65" >

[0823] <if:name>Ethernet1</if:name>

[0824] <if:mac-address>0006GG93KAG8</if:mac-address>

[0825] <if:ipv4>

[0826] <if:address-v4>59.64.139.65</if:address-v4>

[0827] <if:flags up ＝" true" />

[0828] <if:received>19</if:received>

[0829] </if:ipv4>

[0830] </if:interface>

[0831] <if:interface name ＝" Ethternet2" ip ＝" 59.64.139.69" >

[0832] <if:name>Ethernet2</if:name>

[0833] <if:mac-address>000E35A83K4K</if:mac-address>

[0834] <if:ipv4>

[0835] <if:address-v4>59.64.139.69</if:address-v4>

[0836] <if:flags up ＝ " false " />

[0837] <if:received>49</if:received>

[0838] </if:ipv4>

[0839] </if:interface>

[0840] </routing>

[0841] <mt:monitor>

[0842] <mt:configuration>running-state</mt:configuration>

[0843] <mt:configuration>startup-config</mt:configuration>

[0844] <mt:configuration>running-config</mt:configuration>

[0845] <if:configuration>http://pris.bupt.cn/example/</if:configuration>

[0846] </mt:monitor>

[0847] </netconf>

[0848] 现对其进行查询,过滤器定义如下例 54 实施例过滤器定义 :

[0849] <filter type =”advancedSubtree”_nodepath =”relative”>

[0850] <if:interface name = " Ethternet1 " ip = " 59.64.139.65 " _attrLogic = " (name ^ ip)

[0851] ‖(! name ^ ! ip) "

[0852] xmlns:if = " *interface* " />

[0853] <_ _count = true/>

[0854] </if:interface>

[0855] <mt:monitor xmlns:mt = " *monitor:1.0 " >

[0856] <configuration_nsLogic = " if‖mt " _order = " asc " />

[0857] </mt:monitor>

[0858] </filter>

[0859] 当get/get-config操作发现操作报文中含有filter节点,将之交给图2中过滤控制模块,该模块调用图2中参数设置功能模块,由filter的属性type =”advancedSubtree”得知需要进行子树过滤扩展功能操作,并解析到需要相对路径匹配,将参数 nodepath 送给流程控制开始扩展子树过滤操作。如图 4 中步骤 4020 所示。

[0860] 过滤器的定义主要流程如下 :

[0861] 操作开始前进行报文检查,没有发现语法错误。

[0862] 没有找到<xmlns>元素,认为非命名空间查询后。

[0863] 将 filter 第一个子节点 interface 与数据模型根节点 netconf 匹配。由于名字不同,匹配失败。

[0864] 由于是非绝对路径,因此在 netconf 节点的后代中寻找到与 interface 匹配的节点 :<if:interface name = ″ Ethternet1 ″ ip = ″ 59.64.139.65 ″ \>与<iffinterfacename =″ Ethternet2″ ip =″ 59.64.139.69″ \>。

[0865] 查找匹配节点时,涉及到属性的逻辑关系组合过滤条件。name = ″ Ethternet1 ″ ip = ″ 59.64.139.65″ _attrLogic = ″ (name ^ ip)‖(! name ^ !ip)″ 表示,过滤结果中 interface 节点要么 name 和 ip 均满足给定条件,要么均不满足。匹配中,对于数据模型节点<if:interface name = ″ Ethternet1 ″ ip =″ 59.64.139.65″ >,name 属性条件满足,ip 属性条件也满足。则 _attrLogic 引出的表达式的匹配结果为 :(true ^ true)‖(false ^ false) = true,即该数据模型节点满足属性的逻辑关系过滤条件。类似的,routing 下另一个 interface 节点同样满足。

[0866] 查找匹配节点时,涉及到命名空间的模糊匹配。urn:bupt:pris:priser:agent:module:interface:1.0 含有 interface 字符串,能够匹配。

[0867] 由于<if:interface name =″ Ethternet″ ip =″ 59.64.139.65″ >节点和过滤器中的 interface 节点匹配,则处理该节点的子节点,这些子节点是兄弟节点,同时进行处理。

[0868] 过滤器 interface 节点下只有<_>节点,且没有属性限制条件,匹配任何命名空间

包含 interface 字符串的元素节点。故而 name、mac-address、ipv4 节点均符合过滤条件。

[0869] <_> 节点带有 _count 属性值为 true，统计与 <_> 节点匹配的数据模型节点个数为 3。从而实现统计 interface 子节点个数的功能。

[0870] 则过滤器子树：

[0871] <if:interface name =" Ethternet1" ip =" 59.64.139.65" _attrLogic =" (name ^ ip)

[0872] ‖ (!name ^ !ip)" xmlns:if = " *interface*" />

[0873] <_count = true/>

[0874] </if:interface>

[0875] 得到的过滤结果为：

[0876] <if:interface name =" Ethternet1" ip =" 59.64.139.65" >

[0877] <_countNum>3<_countNum>

[0878] </if:interface>

[0879] <if:interface name =" Ethternet2" ip =" 59.64.139.69" >

[0880] <_countNum>3<_countNum>

[0881] </if:interface>

[0882] 将匹配节点和其祖先复制到过滤结果中。

[0883] 处理过滤器中 interface 的兄弟节点 monitor。

[0884] 数据模型中 monitor 的命名空间 urn:bupt:pris:priser:agent:module:interface:1.0 包含 monitor :1.0 字符串在末尾。无其他匹配条件，mt:monitor 节点匹配。

[0885] 处理过滤器 monitor 节点的子节点 configuration。数据模型中 configuration 元素的命名空间有两种 :urn:bupt:pris:priser:agent:module:monitor:1.0、um:bupt:pris:priser:agent:module:interface:1.0，要么匹配 mt 代表的 *monitor:1.0，要么符合 if 代表的 *interface*，数据模型中四个同父 configuration 节点均匹配。

[0886] 则过滤器子树：

[0887] <mt:monitor xmlns:mt =" *monitor:1.0" >

[0888] <configuration_nsLogic =" if‖mt" _order =" asc" />

[0889] </mt:monitor>

[0890] 得到的过滤结果为：

[0891] <mt:monitor>

[0892] <mt:configuration>running-state</mt:configuration>

[0893] <mt:configuration>startup-config</mt:configuration>

[0894] <mt:configuration>running-config</mt:configuration>

[0895] <if:configuration>http://pris.bupt.cn/example/</if:configuration>

[0896] </mt:monitor>

[0897] 将匹配节点和其祖先复制到过滤结果中。

[0898] 过滤器中节点匹配完毕，得到初始过滤结果：

[0899] <netconfxmlns =" urn:ietf:params:xml:ns:netconf:base:1.0"

[0900] xmlns:mt ＝" um:bupt:pris:priser:agent:module:monitor:1.0"

[0901] xmlns:if ＝" um:bupt:pris:priser:agent:module:interface:1.0" >

[0902] <routing>

[0903] <if:interface name ＝" Ethternet1" ip ＝" 59.64.139.65" >

[0904] <_countNum>3<_countNum>

[0905] </if:interface>

[0906] <if:interface name ＝" Ethternet2" ip ＝" 59.64.139.69" >

[0907] <_countNum>3<_countNum>

[0908] </if:interface>

[0909] </routing>

[0910] <mt:monitor>

[0911] <mt:configuration>running-state</mt:configuration>

[0912] <mt:configuration>startup-config</mt:configuration>

[0913] <mt:configuration>running-config</mt:configuration>

[0914] <if:configuration>http://pris.bupt.cn/example/</if:configuration>

[0915] </mt:monitor>

[0916] </netconf>

[0917] 对过滤器中 configuration 节点得到的过滤结果进行排序。文本内容不为空,按升序排列。则 monitor 子树变为 :

[0918] <mt:monitor>

[0919] <if:configuration>http://pris.bupt.cn/example/</if:configuration>

[0920] <mt:configuration>running-config</mt:configuration>

[0921] <mt:configuration>running-state</mt:configuration>

[0922] <mt:configuration>startup-config</mt:configuration>

[0923] </mt:monitor>

[0924] 将排序后的过滤结果交给控制模块,返回给上级操作。

[0925] 最终得到的过滤结果为例 55 :例 54 的过滤结果 :

[0926] <netconfxmlns ＝" urn:ietf:params:xml:ns:netconf:base:1.0"

[0927] xmlns:mt ＝" urn:bupt:pris:priser:agent:module:monitor:1.0"

[0928] xmlns:if ＝" um:bupt:pris:priser:agent:module:interface:1.0" >

[0929] <routing>

[0930] <if:interface name ＝" Ethternet1" ip ＝" 59.64.139.65" >

[0931] <_countNum>3<_countNum>

[0932] </if:interface>

[0933] <if:interface name ＝" Ethternet2" ip ＝" 59.64.139.69" >

[0934] <_countNum>3<_countNum>

[0935] </if:interface>

[0936] </routing>

[0937] <mt:monitor>

[0938] <if:configuration>http://pris.bupt.cn/example/</if:configuration>

[0939] <mt:configuration>running-config</mt:configuration>

[0940] <mt:configuration>running-state</mt:configuration>

[0941] <mt:configuration>startup-config</mt:configuration>

[0942] </mt:monitor>

[0943] </netconf>

[0944] 现对如例 53 中定义的数据模型进行查询,过滤器定义如下例 56 实施例过滤器定义 :

[0945] <filter type =”advancedSubtree”_nodepath =”relative”>

[0946] <_xmlns>

[0947] <interface>

[0948] <received_morethan =”12”_lessthan =”25”_matchType =”may”/>

[0949] <received_morethan =”42”_lessthan =”55”_matchType =”may”/>

[0950] </interface>

[0951] </_xmlns>

[0952] </filter>

[0953] 过滤器定义的主要方法如下 :

[0954] 过滤控制模块解析设置过滤类型”advancedSubtree”和相关参数 _nodepath =”relative”。

[0955] 经过报文语法检查后,进行扩展子树过滤操作。

[0956] 检查到 filter 有唯一根节点 _xmlns,下面进行命名空间查询,如图 5 所示。

[0957] 得到命名空间查询所用子树

[0958] <interface>

[0959] <received_morethan =”12”_lessthan =”25”_matchType =”may”/>

[0960] <received_morethan =”42”_lessthan =”55”_matchType =”may”/>

[0961] </interface>

[0962] 以此进行如图 6 中的正常扩展子树过滤,但是取消命名空间匹配机制。将 filter 第一个子节点 interface 与数据模型根节点 netconf 匹配。由于名字不同,匹配失败。

[0963] 由于是非绝对路径,因此在 netconf 节点的后代中寻找到与 interface 匹配的节点 :<if:interface name = ″ Ethternet1 ″ ip = ″ 59.64.139.65 ″ /> 与 <if:interfacename =″ Ethternet2″ ip =″ 59.64.139.69″ />。

[0964] 对于 <if:interface name =″ Ethternet1″ ip =″ 59.64.139.65″ /> 继续匹配下一层兄弟节点 <received_morethan =”12”_lessthan =”25”/> 和 <if:name/>、<if:mac-address/>、<if:ipv4>。均不匹配

[0965] 由于是非绝对路径,因此在 interface 节点的后代中寻找到与 received 匹配的节点 :<if:received>19</if:received>。

[0966] 匹配涉及到数量范围条件和节点逻辑关系,对于数据模型节点

<if:received>19</if:received>和过滤器内容匹配节点<received_morethan =”12”_lessthan =”25”_matchType =”may”/>,_morethan、_lessthan给出的范围条件均符合。由于节点匹配方式是may,则即使其祖先节点<if:interface name =″ Ethternet1″ ip =″ 59.64.139.65″ />的后代received节点的值仅满足过滤器中两个received节点条件之一,但该interface节点仍然是匹配的。

[0967] 数据模型节点<if:interface name =″ Ethternet2 ″ ip =″ 59.64.139.69″ />具有后代节点received,其值大于42、小于55,和过滤器节点interface也是匹配的。

[0968] 将匹配节点及其祖先复制到初始过滤结果中：

[0969] <netconfxmlns =″ urn:ietf:params:xml:ns:netconf:base:1.0″

[0970] xmlns:mt =″ urn:bupt:pris:priser:agent:module:monitor:1.0″

[0971] xmlns:if =″ urn:bupt:pris:priser:agent:module:interface:1.0″ >

[0972] <routing>

[0973] <if:interface name =″ Ethternet1″ ip =″ 59.64.139.65″ >

[0974] ……

[0975] <if:ipv4>

[0976] ……

[0977] <if:received>19</if:received>

[0978] </if:ipv4>

[0979] </if:interface>

[0980] <if:interfacename = " Ethternet2 " ip = " 59.64.139.69 " >

[0981] ……

[0982] <if:ipv4>

[0983] ……

[0984] <if:received>49</if:received>

[0985] </if:ipv4>

[0986] </if:interface>

[0987] </routing>

[0988] </netconf>

[0989] 在结果处理模块中检索初始过滤结果各元素的命名空间,得到隐式命名空间um:ietf:params:xml:ns:netconf:base:1.0及带有前缀为if的命名空间um:bupt:pris:priser:agent:module:interface:1.0,封装为最后的过滤结果为例57:例56的过滤结果：

[0990] <data>

[0991] <_xmlns>urn:ietf:params:xml:ns:netconf:base:1.0</_xmlns>

[0992] <_xmlns prf =”if”>um:bupt:pris:priser:agent:module:interface:1.0</_xmlns>

[0993] </data>

[0994] 本发明实施例二为一种基于子树过滤的模糊查询方法,如图8所示,包括：

[0995] 步骤s801,接收待过滤数据流；

[0996] 步骤 s802，通过不完全匹配的方式过滤数据流，处理用户没有给出完全信息的字符串形式的过滤条件。

[0997] 所述字符串包括：节点名字、节点内容、属性名字、属性值或命名空间；所述不完全匹配的方式包括：路径不完整；元素名不完整、元素值不完整、属性名不完整、属性值不完整、命名空间不完整。

[0998] 上述通过不完全匹配的方式过滤数据流，具体为：

[0999] 通过绝对路径过滤数据流；或通过相对路径过滤数据流。

[1000] 通过绝对路径过滤数据流具体为：用户给出的过滤器包含被选择节点的必须给出从根节点开始的所有祖先，过滤结果中只包含在过滤器中的祖先和在数据模型中的祖先完全一致的节点及节点后代；

[1001] 通过相对路径过滤的数据流具体为：

[1002] 在过滤器中满足父子关系的节点在过滤结果中至少满足祖先后代关系；在过滤器中为根节点的节点在过滤结果中至少包括非根节点。

[1003] 当出现节点的命名空间不匹配时，继续处理前述节点的子节点，保留命名空间不匹配的元素作为其后代节点的路径。

[1004] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[1005] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1006] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

[1007] 本发明实施例三为一种子树过滤条件的逻辑组合的扩展方法，如图 9 所示，包括：

[1008] 步骤 s901，接收待过滤数据流；

[1009] 步骤 s902，采用多个属性或多个内容匹配节点对该数据流中的元素类型节点进行过滤。

[1010] 步骤 s902 中多个内容匹配节点对数据流中的元素类型节点进行过滤，具体为：

[1011] 使用内容匹配节点作为从数据模型中选择特定的父节点的过滤条件；

[1012] 通过在所述内容匹配节点中添加第一属性表示所述节点的匹配程度，匹配程度包括必须匹配、可选匹配和不匹配；

[1013] 根据所述内容匹配节点和所述匹配程度对所述数据流中的节点进行过滤。

[1014] 根据所述内容匹配节点和所述匹配程度对所述数据流中的节点进行过滤，具体为：

[1015] 过滤器中所有匹配程度是必须匹配的内容匹配节点都与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

[1016] 过滤器中可选匹配程度是可选匹配的内容匹配节点中至少有一个节点与数据模型中相应的节点完全匹配时，其父节点才可以被添加到过滤结果中；

[1017] 过滤器中匹配程度不匹配的内容匹配节点与数据模型中相应的节点完全匹配时，其父节点不被添加到过滤结果中。

[1018] 通过向节点中添加第二属性，实现该节点的多个属性间的逻辑关系，该第二属性为 _attrLogic 属性。

[1019] 上述多个内容匹配节点对数据流中的元素类型节点进行过滤的过程中还包括：进行命名空间查询，进行命名空间查询具体为：

[1020] 获取数据模型中命名空间的定义情况，并给出命名空间的列表，其中，命名空间查询的关键字用一个元素表示。

[1021] 上述多个内容匹配节点对数据流中的元素类型节点进行过滤的步骤中，过滤条件中还包括范围条件，该范围条件用于选择具有某些特定数值或者日期范围的节点，在表示数量或时间的选择节点中添加表示属性范围或时间范围的属性，将该选择节点看作是表示时间或数量范围的内容匹配节点，在过滤处理时作为过滤条件。

[1022] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[1023] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1024] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

[1025] 本发明实施例四为一种子树过滤查询结果的处理方法，如图 10 所示，包括：

[1026] 步骤 s1001，在过滤器中相应节点处设置第三属性的值，表明该节点处是否显示节点后代，该第三属性为 _show 属性；

[1027] 步骤 s1002，该第三属性的取值为：第一值或者第二值，第一值为 self 值，第二值为 descendant 值，当该第三属性默认为显示所有在同一命名空间的后代，则第三属性＝第二值；当设置第三属性＝第一值，则过滤结果只输出该节点的信息，不输出在同一命名空间的后代节点。

[1028] 该处理方法还包括：

[1029] 通过在过滤器中某元素节点中使用第四属性实现个数查询，统计数据模型中某一类元素的个数，通过在过滤器中某元素节点中使用第四属性实现个数查询，统计同一过滤器节点对应的元素个数。该第四属性为 _count 属性。

[1030] 该第四属性的取值有两个：第一值或第二值，该第一值为 true，第二值为 false，当设置第四属性值为第一值时，则过滤结果输出查询实体的个数；当设置第四属性为第二值时，则过滤结果不输出查询实体的个数。

[1031] 该处理方法还包括：

[1032] 对同一个过滤器节点得到的过滤结果按照需要进行排序，合并查询结果中内容完全相同的节点。

[1033] 该处理方法还包括：

[1034] 当所有需要匹配的节点没有匹配数据模型节点，全部被舍弃时，不返回上述节点的父节点。

[1035] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提

供完整手段实现命名空间查询的缺陷。

[1036] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1037] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

[1038] 本发明实施例五为一种过滤器，如图 11 所示，包括：

[1039] 接收模块 1110，用于接收待过滤数据流；

[1040] 过滤模块 1120，用于通过不完全匹配的方式过滤该数据流，处理用户没有给出完全信息的字符串形式的过滤条件。

[1041] 过滤模块 1120 包括：

[1042] 第一过滤子模块 1121，用于通过绝对路径过滤该数据流；和 / 或，

[1043] 第二过滤子模块 1122，用于通过相对路径过滤所述数据流。

[1044] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[1045] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1046] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

[1047] 本发明实施例六为一种逻辑过滤装置，如图 12 所示，包括：

[1048] 接收模块 1210，用于接收待过滤数据流；

[1049] 逻辑过滤模块 1220，用于采用多个属性或多个内容匹配节点对该数据流中的元素类型节点进行过滤。

[1050] 逻辑过滤模块 1220 包括：

[1051] 设置子模块 1221，用于设置内容匹配节点作为从数据模型中选择特定的父节点的过滤条件；

[1052] 添加子模块 1222，用于通过在该内容匹配节点中添加第一属性表示该节点的匹配程度，并实现该节点过滤条件之间的逻辑关系组合；

[1053] 处理子模块 1223，用于根据该过滤条件之间的逻辑关系组合对该数据流中的元素类型节点进行过滤。

[1054] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[1055] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1056] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要时，丢弃无用的后代节点，大大减少了子树过滤结果可能存在的大量冗余信息。

[1057] 本发明实施例七为一种子树过滤查询结果处理装置，如图 13 所示，包括：

[1058] 设置模块 1310，用于在过滤器中相应节点处设置第三属性的值，该第三属性为 _

show 属性；

[1059] 确定模块 1320，用于根据设置模块 1310 设置第三属性的值，表明所述节点处是否显示节点后代。

[1060] 该处理装置还包括：

[1061] 排序模块 1330，用于对同一个过滤器节点得到的过滤结果按照需要进行排序，

[1062] 节点合并模块 1340，用于合并查询结果中内容完全相同的节点。

[1063] 综上，可以看到本发明可以实现以下 NETCONF 协议中子树过滤机制无法实现或实现效果不好或处理方法有缺陷的功能，如表 2 所示：

[1064] 表 2 扩展子树过滤机制实现附加功能

[1065]

功能类别	扩展功能实现	备注
模糊查询	节点名字、节点值、属性名字、属性值、命名空间的模糊匹配	

[1066]

过滤	属性名字与值的互相查询	通过匹配所有非空字符实现
	节点名字与值的互相查询	通过匹配所有非空字符实现
	相对路径或未知路径的查询	可以实现在整个文档中查询节点或属性的功能
过滤条件逻辑关系	命名空间过滤的或、非逻辑关系	
	属性过滤条件的或、与、非逻辑关系	
	内容匹配节点的或、与、非逻辑关系	
命名空间	查询文档中所有的命名空间	
	查询子树中所有的命名空间	
	查询某命名空间中的所有元素	命名空间匹配的处理与原协议有所不同
	查询前缀对应的命名空间	通过模糊过滤实现
	过滤时命名空间匹配无效	
去除冗余信息	选择节点只输出本身	
	当查询目标不匹配时，也舍弃其祖先	
	相同子树合并	
查询子句扩展	实体个数查询	
	数量、时间范围条件	
	同结构子树排序	

[1067] 由表中可以看到：

[1068] 本发明实施例提供一种基于NETCONF子树过滤机制，以XML形式表达的，对XML网络管理数据进行查询和过滤的方法，使对NETCONF协议中管理信息的获取更为方便，也避免了为数据查询和过滤重新定义一套规范带来的诸多问题。

[1069] 本发明实施例提供了一种对XML数据过滤条件的细化和组合机制。XML文档中的节点有时包含了具体的数据信息，类似于关系数据库中一个字段；有时候只是封装数据信

息的手段，类似于关系数据库中的一条记录。本发明中细化了节点的匹配程度，用户可以根据节点在数据存储中的具体作用选择相应的查询方法。另外，属性、命名空间方式表达的过滤条件可以以逻辑运算方式组合到一起，使查询更为灵活。

[1070] 本发明实施例提供一种在过滤条件不完整的情况下，比如不能完整拼写出数据存储的节点或属性名字，或者不能完全给出数据在 XML 文档中定义的位置，仍可以对 XML 数据进行子树过滤操作，并得到用户想要的结果。

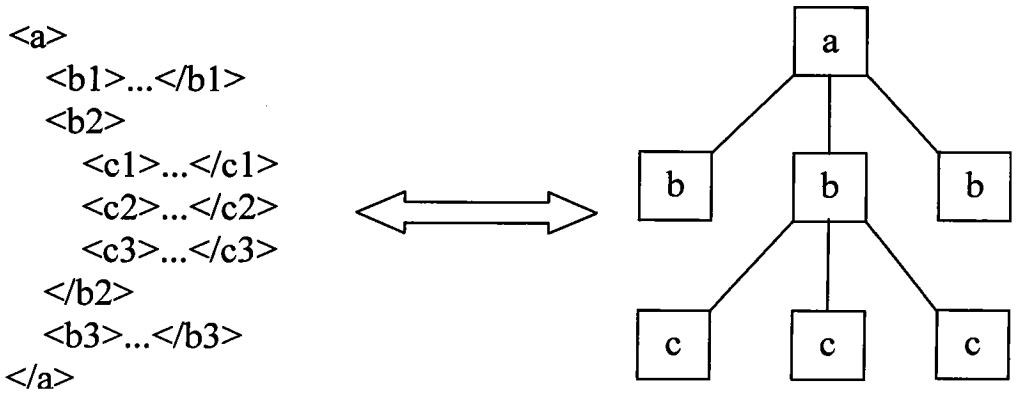
[1071] 本发明实施例增加子树过滤中命名空间的查询功能，帮助用户快速的了解 XML 文档中数据分布的结构；弥补了 NETCONF 协议子树过滤中必须给出命名空间条件，但又不提供完整手段实现命名空间查询的缺陷。

[1072] 本发明实施例处理对某些数据的特殊查询需求，比如时间、日期、数量等的范围查询、数据的数量查询等等。使子树过滤更能满足实际网络管理或其他实际应用中的需求。

[1073] 本发明实施例实现对过滤结果的优化，实现对相同或类似数据的合并或排序；在需要的时候，丢弃无用的后代节点。大大减少了子树过滤结果可能存在的大量冗余信息。

[1074] 通过以上的实施方式的描述，本领域的技术人员可以清楚地了解到本发明可以通过硬件实现，也可以借助软件加必要的通用硬件平台的方式来实现基于这样的理解，本发明的技术方案可以以软件产品的形式体现出来，该软件产品可以存储在一个非易失性存储介质（可以是 CD-ROM，U 盘，移动硬盘等）中，包括若干指令用以使得一台计算机设备（可以是个人计算机，服务器，或者网络设备等）执行本发明各个实施例所述的方法。

[1075] 以上所述仅是本发明的优选实施方式，应当指出，对于本技术领域的普通技术人员来说，在不脱离本发明原理的前提下，还可以做出若干改进和润饰，这些改进和润饰也应视为本发明的保护范围。


<a>
<b1>...</b1>
<b2>
<c1>...</c1>
<c2>...</c2>
<c3>...</c3>
</b2>
<b3>...</b3>

a
b
b
b
c
c
c

图 1

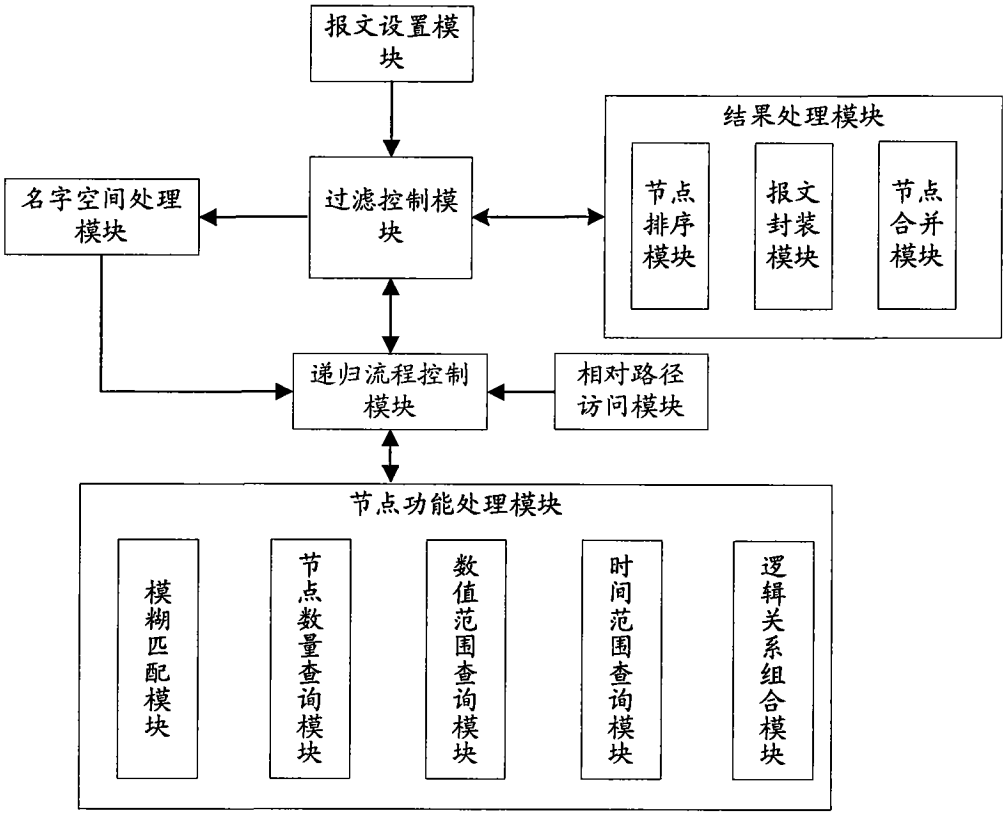

报文设置模块
过滤控制模块
名字空间处理模块
结果处理模块
节点排序模块
报文封装模块
节点合并模块
递归流程控制模块
相对路径访问模块
节点功能处理模块
模糊匹配模块
节点数量查询模块
数值范围查询模块
时间范围查询模块
逻辑关系组合模块

图 2

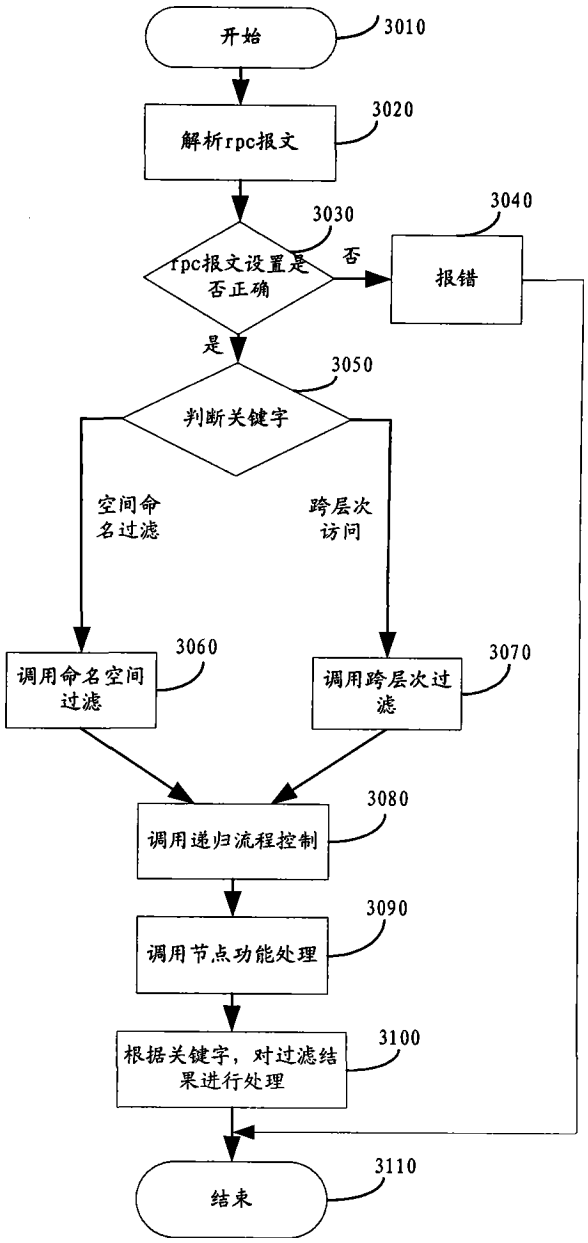

开始
3010
解析rpc报文
3020
rpc报文设置是否正确
3030
否
报错
3040
是
判断关键字
3050
空间命名过滤
跨层次访问
调用命名空间过滤
3060
调用跨层次过滤
3070
调用递归流程控制
3080
调用节点功能处理
3090
根据关键字，对过滤结果进行处理
3100
结束
3110

图 3

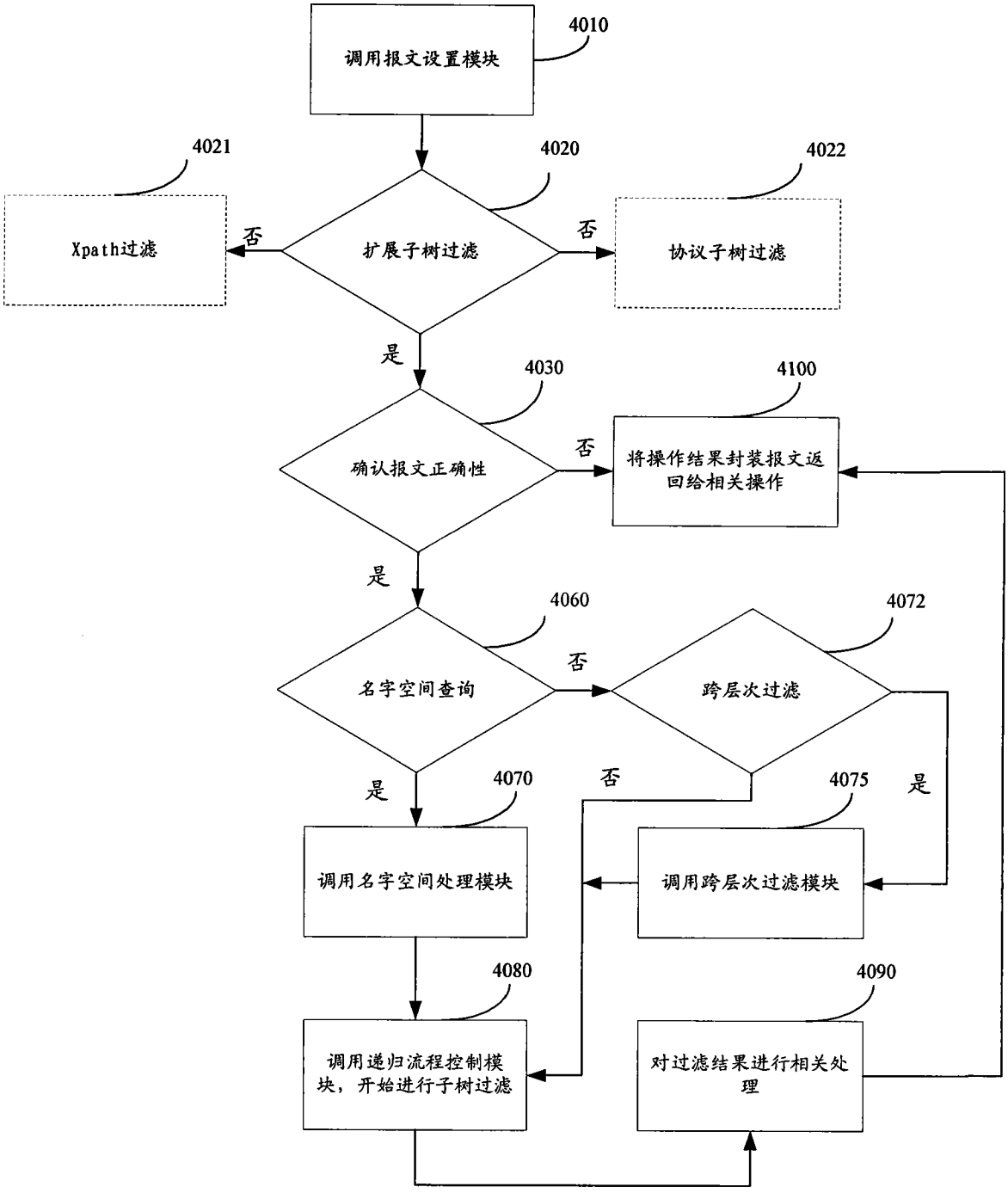

调用报文设置模块
4010
4021
4020
4022
Xpath过滤
否
扩展子树过滤
否
协议子树过滤
是
4030
4100
确认报文正确性
否
将操作结果封装报文返回给相关操作
是
4060
4072
名字空间查询
否
跨层次过滤
是
4070
否
4075
是
调用名字空间处理模块
调用跨层次过滤模块
4080
4090
调用递归流程控制模块，开始进行子树过滤
对过滤结果进行相关处理

图 4

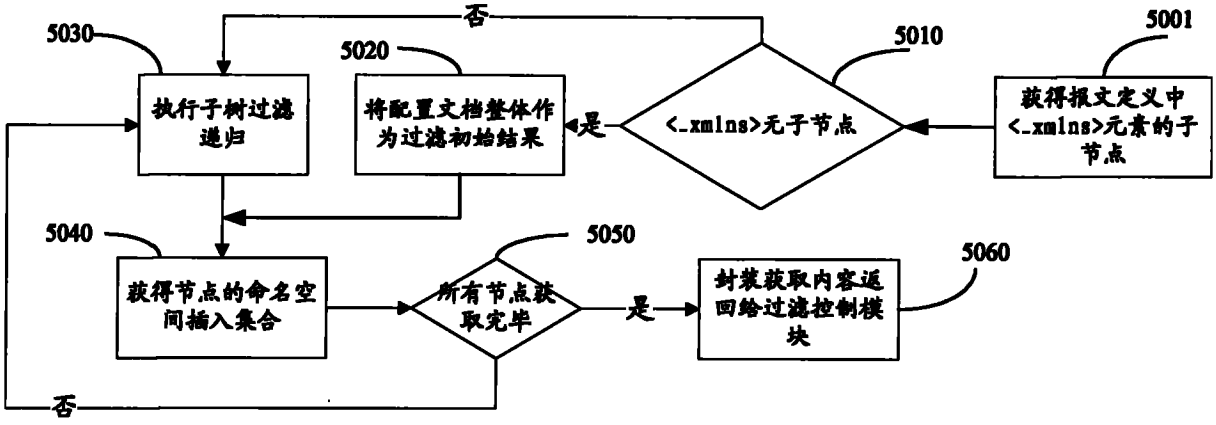

5001
获得报文定义中<_xmlns>元素的子节点
5010
<_xmlns>无子节点
是
否
5020
将配置文档整体作为过滤初始结果
5030
执行子树过滤递归
5040
获得节点的命名空间插入集合
5050
所有节点获取完毕
是
否
5060
封装获取内容返回给过滤控制模块

图 5

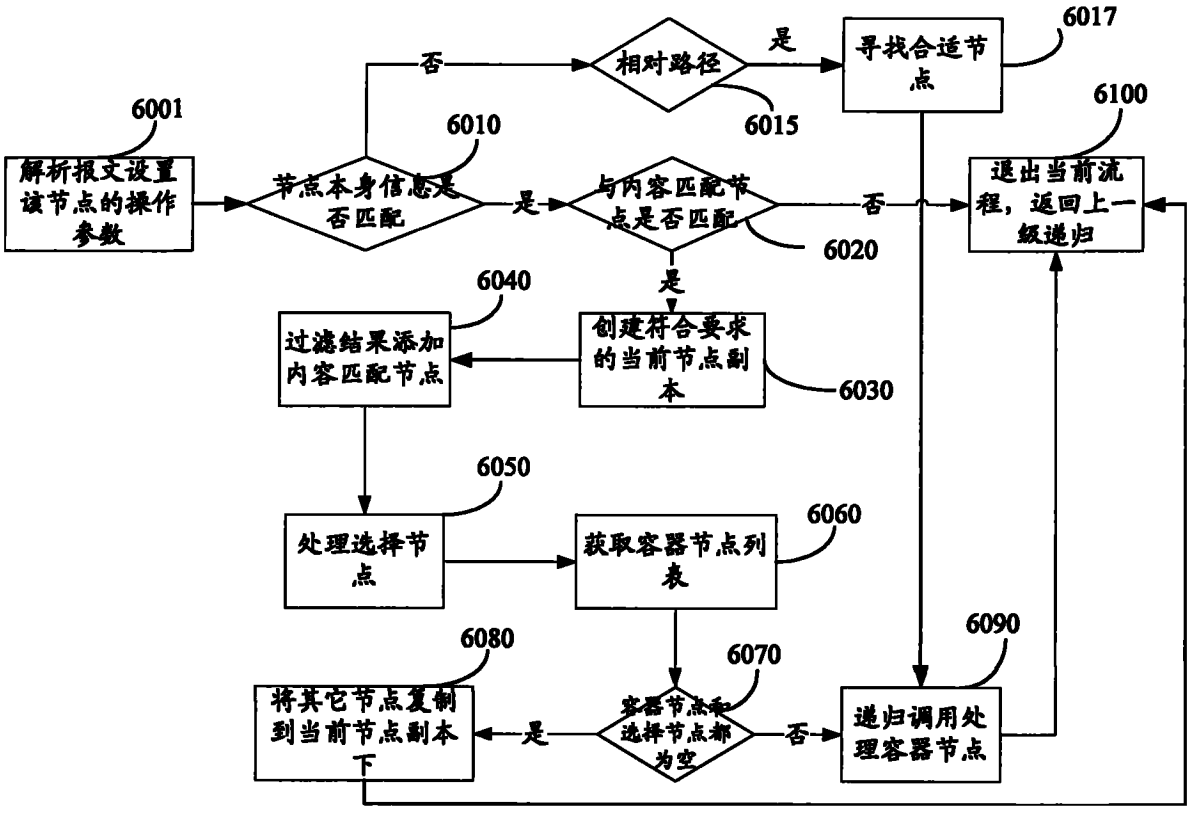

6001
解析报文设置该节点的操作参数
6010
节点本身信息是否匹配
是
否
相对路径
是
否
6015
寻找合适节点
6017
与内容匹配节点是否匹配
否
6020
是
退出当前流程，返回上一级递归
6100
6030
创建符合要求的当前节点副本
6040
过滤结果添加内容匹配节点
6050
处理选择节点
6060
获取容器节点列表
6070
容器节点和选择节点都为空
是
否
6080
将其它节点复制到当前节点副本下
6090
递归调用处理容器节点

图 6

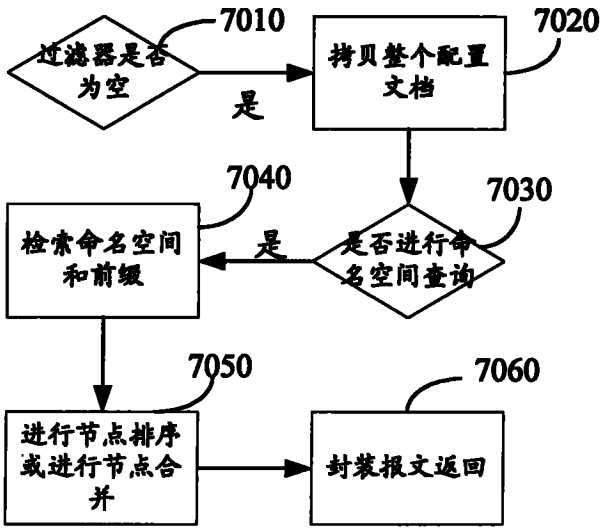

过滤器是否为空
7010
是
拷贝整个配置文档
7020
是否进行命名空间查询
7030
是
检索命名空间和前缀
7040
进行节点排序或进行节点合并
7050
封装报文返回
7060

图 7

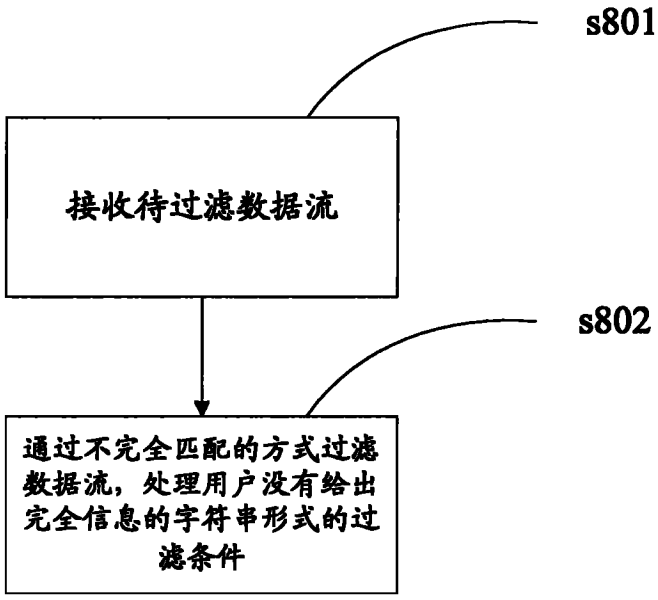

s801
接收待过滤数据流
s802
通过不完全匹配的方式过滤数据流，处理用户没有给出完全信息的字符串形式的过滤条件

图 8

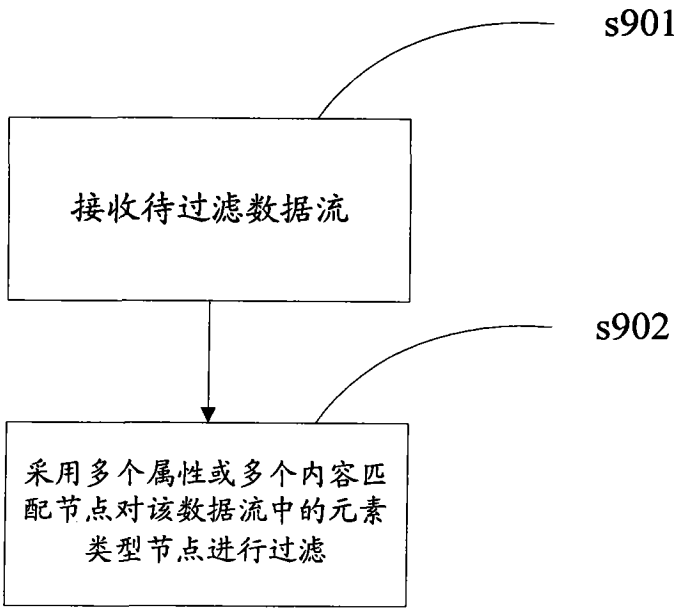

s901
接收待过滤数据流
s902
采用多个属性或多个内容匹配节点对该数据流中的元素类型节点进行过滤

图 9

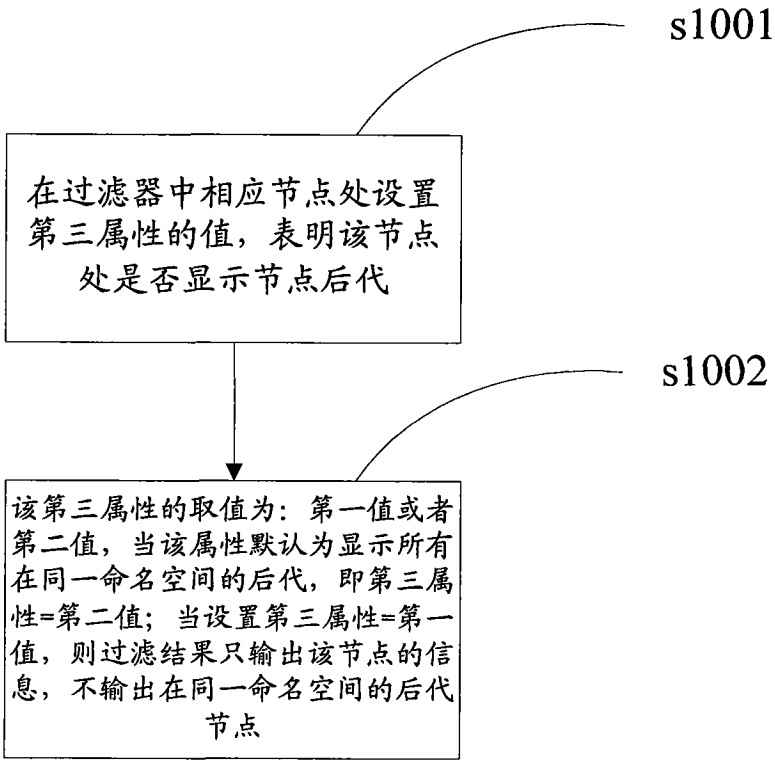

s1001
在过滤器中相应节点处设置第三属性的值，表明该节点处是否显示节点后代
s1002
该第三属性的取值为：第一值或者第二值，当该属性默认为显示所有在同一命名空间的后代，即第三属性=第二值；当设置第三属性=第一值，则过滤结果只输出该节点的信息，不输出在同一命名空间的后代节点

图 10

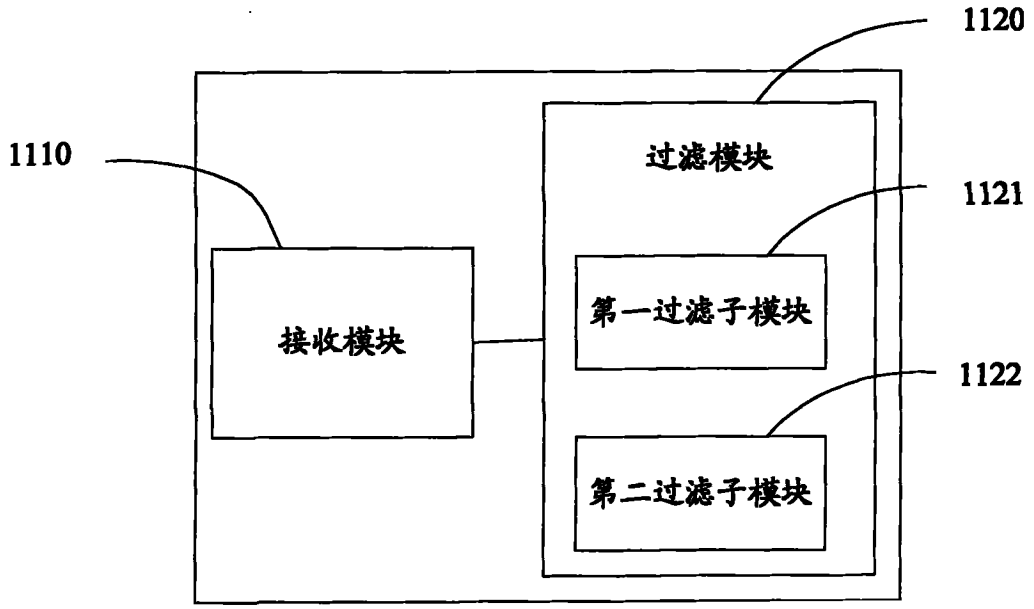

1120
过滤模块
1110
1121
接收模块
第一过滤子模块
1122
第二过滤子模块

图 11

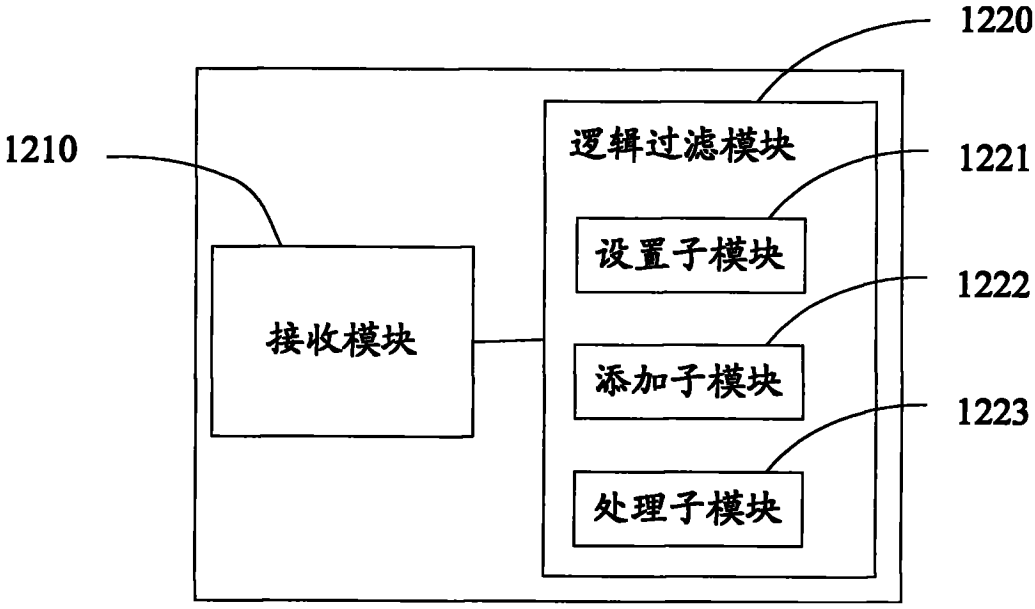

1220
逻辑过滤模块
1210
1221
设置子模块
1222
接收模块
添加子模块
1223
处理子模块

图 12

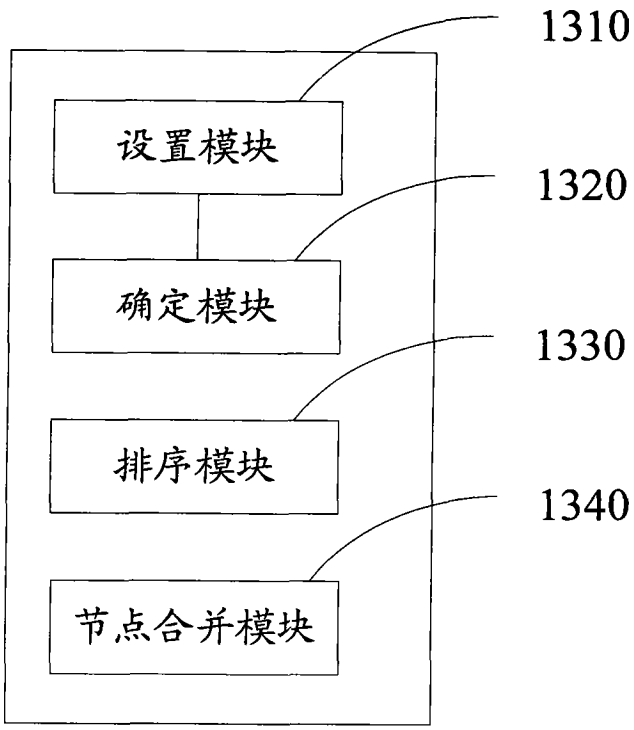


图 13