

## XML 数据流上基于关键字的多查询处理

周军锋<sup>1, 2</sup> 孟小峰<sup>1</sup> 张新<sup>1</sup> 黄静<sup>1</sup>

<sup>1</sup>(中国人民大学信息学院, 北京 100872)

<sup>2</sup>(燕山大学计算机系, 秦皇岛 066004)

(zhoujf@ysu.edu.cn)

**摘要** 本文试图将基于 XML 文档的关键字查询技术引入数据流环境中,在同时处理大量基于关键字的查询的基础上为用户返回有意义的片段.提出了一种基于有向无环图的索引来高效组织大量基于关键字的查询,用以降低查询匹配的代价;针对数据流的特点,提出了一种基于栈的临时结果缓存方法,用于过滤大量查询无关的数据节点;通过实验从不同角度对本文提出的算法的各项性能指标进行了实验验证.

**关键词** XML 数据流, 关键字查询, 最小相关连通子树

中图法分类号 TP391

## Keyword Based Multiple Query Processing over XML Streams

Zhou Junfeng<sup>1,2</sup>, Meng Xiaofeng<sup>1</sup>, Zhang Xin<sup>1</sup> and Huang Jing<sup>1</sup>

<sup>1</sup>(School of Information, Renmin University of China, Beijing 100872, China)

<sup>2</sup>(College of Computer Science, Yanshan University, Qin Huangdao, 066004)

**Abstract** In this paper, we aim at providing users with friendly interface over XML streams and useful data fragments by introducing keyword search related technology into XML streams. We propose a new index which is based on directed acyclic graph to organize the large number of keywords and queries to reduce the cost of query processing, then a stack based method is proposed to cache temporal results and filter out large number of useless data elements without redundant processing cost. The experimental results on various datasets indicate that the method proposed in this paper performs significantly better than the existing ones.

**Keywords** XML Streams, Keyword Search, SLCA

### 1 引言

随着 XML 应用的不断扩展, XML 已经成为互联网上数据表示和交换事实上的标准,许多结构化或半结构化的数据都以 XML 格式表示和传输.与此同时,出现了大量涉及 XML 的数据流应用,如股票交易信息、实时新闻的订阅和发布、监控关键设备的运行以及监控环境的变化情况等.为了给用户提供更大的灵活性,可以使用基于关键字的查询来方便用户的使用,系统根据用户查询所包含的关键字信息,将满足条件的数据片段返回给用户.与传统信息检索不考虑结构信息的结果相比,这里返回的结果更有实际意义.然而,实际的 XML 数据流系统

中总的查询数量十分庞大,必须同时检测所有查询的匹配情况.为此需要解决两个主要问题:(1)如何构建高效索引,(2)如何进行高效查询处理.

本文的贡献可表述如下:(1)提出了一种新的用于对大量基于关键字的查询进行索引的方法.(2)提出了一种基于栈的多查询结果缓存策略.(3)根据不同评价指标从不同角度通过丰富的实验结果对本文提出的方法进行了实验验证.

### 2 数据模型及相关工作

处理 XML 数据流的一种常用程序接口是 SAX.在本文中, XML 数据流被表示成一系列 SAX 事件,主要事件有: BEGIN(tag), END(tag)和 TEXT(),其中

收稿日期:

基金项目: \*本文得到国家自然科学基金(60573091),北京市自然科学基金(4073035),教育部新世纪优秀人才支持计划的资助.

tag 是指处理过程中遇到的节点名称. BEGIN(tag)表示遇到一个 tag 的开始标志, END(tag)表示遇到一个 tag 的结束标志, TEXT()表示遇到节点的文本.

文献[1,2]主要解决如何快速找到 SLCA 节点集合.文献[3~6]主要讨论在 XML 数据流上进行有效的XPath 查询,文献[7]主要关注在XML数据流上进行 XQuery 查询.对于系统中注册的大量基于关键字的查询,文献[8,9]提出了两种不同的按照关键字对所有相关的查询建立索引的方法.文献[10]强调了在 XML 数据流上做单个关键字查询的问题.

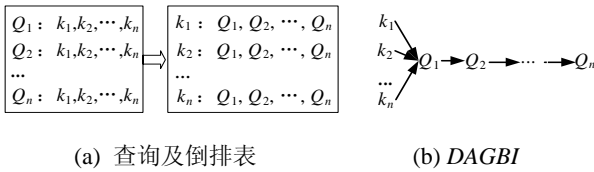


图1 极端情况示例

### 3 DAGBI:基于有向无环图的关键字索引

#### 3.1 问题的提出

观察图 1(a)不难发现,  $Q_1$  的关键字是  $Q_2$  关键字集合的子集.假设每个查询  $Q$  都是关键字的集合,则有如下结论成立:

**定理 1:**假设  $Q_i \subseteq Q_j$ .对于文档  $T$ ,若  $Q_i$  没有匹配的结果,则  $Q_j$  一定没有匹配的结果.

**性质 1:查询匹配的有序性.** 若  $Q_i \subseteq Q_j$ ,则当遇到某个关键字  $k \in Q_i$  需要处理时,若  $Q_i$  没有匹配成功,则无需处理  $Q_j$ .

根据性质 1,我们将所有关键字及对应的查询用有向无环图的方式来组织, **DAGBI(Directed Acyclic Graph Based Index)**,如图 1(b)所示

**例 1:**如图 1(a)所示,当遇到关键字  $k_1$  时,已有的方法需要判断  $n$  个查询是否匹配.如果将这些关键字和查询以图 1(b)的方式组织,则只须检测  $Q_1$ .可见,基于 DAGBI 处理关键字查询的匹配检测问题时可以有效避免很多不必要的冗余操作.

#### 3.2 DAGBI 的构造

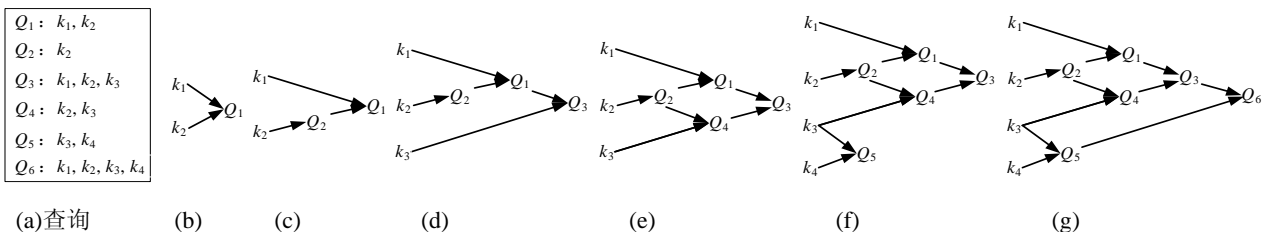


图2 查询及相应 DAGBI 的构造过程

DAGBI 索引的构造过程就是将查询节点及关键字节点不断插入有向图的过程.为了减少插入时的冗余操作,如果  $Q_1 = Q_2$  且  $Q_1$  已经在图中,则限定  $Q_2$  必须插入  $Q_1$  之前.

算法1. *InsertQueryToGraph* ( Query &  $Q$ , Graph &  $G$  )

```

for each keyword  $k \in Q$ 
  if not isIn( $k$ ,  $G$ ) then /*如果G中不含k*/
    Add  $k$  to  $G$ ;
    Add  $Q$  to  $k.outNodeList$ ;
     $Q.InDegree++$ ; /*Q的入度加1*/
  else
     $bInclude = FALSE$ ;
    for each  $q \in k.outNodeList$  /*处理k所指的每个查询*/
      if  $q \notin QInList$  then
        if  $q \subseteq Q$  then
          AddToTail( $InQList$ ,  $q$ );
           $bInclude = TRUE$ ;
        else if  $Q \subseteq q$  then
          AddToTail( $QInList$ ,  $q$ );
           $bInclude = TRUE$ ;
      if  $bInclude = FALSE$  then
        /*和k关联的所有查询都与Q没有包含关系*/
        Add  $k$  to  $G$ ;
        Add  $Q$  to  $k.outNodeList$ ;
         $Q.InDegree++$ ; /*Q的入度加1*/
    if not isEmpty( $InQList$ ) then
      MinimizeInQList (  $InQList$ ,  $QInList$ );
    if not isEmpty( $QInList$ ) then
      InsertQBefore( $Q$ ,  $QInList$ );
    return;
  InsertQAfter( $Q$ ,  $InQList$ );

```

算法 1 的第一部分for循环做两件事,对于不在图  $G$  中的关键字  $k$ ,或者  $k$  已在  $G$  中但与  $k$  关联的所有查询都与  $Q$  没有包含关系时,将  $k$  直接插入  $G$  并让其指向  $Q$ .对于和  $Q$  有包含关系的查询  $q$ ,若  $q \subseteq Q$ ,则将  $q$  暂时保存在  $Q$  包含的查询列表  $InQList$  中;反之若  $Q \subseteq q$ ,则将  $q$  暂时保存在包含  $Q$  的查询列表  $QInList$  中.算法 1 的第二部分if语句主要用于将  $Q$  插入图中,为此,首先要最小化  $InQList$ ,即求在  $Q$  的插入位置之前与之关联的查询节点,对应于图 2(d),为了插入  $Q_3$ ,首先得到的  $InQList=\{Q_1, Q_2\}$ ,最小化之后  $InQList=\{Q_1\}$ ,然后将  $Q_3$  插入  $Q_1$  之后.对于图 2(e)需要插入  $Q_4$  情况,由于  $InQList= \{Q_2\}$ ,  $QInList=\{Q_3\}$ ,则直接将  $Q_4$  插入  $Q_2$  和  $Q_3$  之间.如果图中所有查询都和当前被插入查询没有包含关系,则在第一部分的for循环语句块中对这些关键字进行处理即可.算法 1

中MinimizeInQList函数的作用是 minimized InQList,即找到最靠近Q且能被Q包含的查询集合,同时补充QInList,即找到在Q之后最近的包含Q的查询集合.

### 3.3 DAGBI 的更新

由于 DAGBI 用于对多个查询进行索引,其更新涉及到三种情况:

- (1) 新用户提交的查询需要插入索引中.
- (2) 已有的查询已经到期需要从图中删除.
- (3) 用户修改已提交的查询.

由于这些操作思路简单,在此不再详述.

## 4 基于 DAGBI 的查询处理算法

### 4.1 主要思想及数据结构

BEGIN 事件发生时,将节点推入栈中; TEXT 事件统计当前节点的文本中包含的所有关键字及关联的所有查询; END 事件中判断当前节点包含的关键字对应的查询中哪些已经匹配成功,对于匹配成功的查询,依次向后判断与其关联的查询是否匹配成功,对于匹配成功的查询输出相关的数据片断.

本文提出的查询处理算法中关键字用哈希表

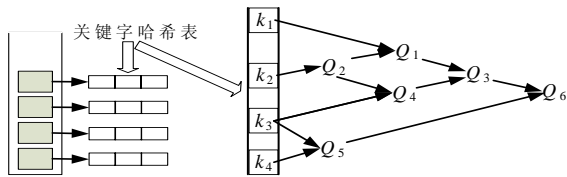


图 3 算法所用数据结构

来组织,如图 3 所示,对碰到的 XML 文档中的每个数据节点,将其入栈,栈元素中含有在其 END 事件前碰到的关键字列表,也用哈希表方式组织,每个哈希表元素对应一个子树,其中子树没有在图中显示.

### 4.2 算法描述

如算法 2 所示,在 XML 数据流上处理查询的代码主要由 3 部分组成:

- (1) BEGIN 事件.将新来的节点入栈保存.
- (2) TEXT 事件.判断文本节点中是否包含图中关键字,将每个在图中出现的关键字插入栈顶元素的哈希表中,同时统计包含该关键字的所有查询.
- (3) END 事件.将栈顶元素出栈,对于每个关联的查询 q,判断其是否匹配,若匹配,则调用 ProcessRelatedQuery 函数递归的判断与 q 直接和间接关联的所有后续查询是否匹配;最后将当前出栈元素 e 关联的所有关键字对应的子树以及剩余查询信息插入当前栈顶元素 E 对应的 E.KeywordList 和 E.RelatedQueryList 中.

例 3:对图 4 所示的文档和图 3 的查询,其状态

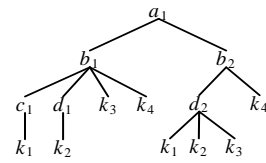


图 4 XML 文档示例

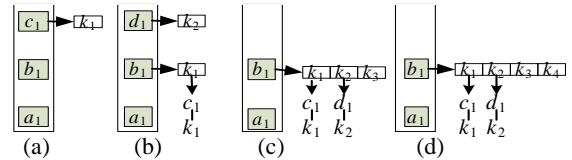


图 5 XML 文档示例

算法 2: /\*基于 DAGBI 索引的查询处理算法\*/

```

BEGIN (tag) /*遇到某个节点的BEGIN事件*/
    e.tag=tag;
    push(S, e); /*将e入栈*/
TEXT()
    e = top(S); /*处理当前栈顶元素的文本*/
    for each k ∈ text /*对当前文本中遇到的每个关键字*/
        if k ∈ e.KeywordList then
            for each q ∈ q.outNodeList
                if q ∈ e.RelatedQueryList then
                    Insert q into e.RelatedQueryList;
                    /*统计和e的文本内容关联的查询*/
            Insert (k, k) into e.KeywordList;
            /*将k对应的子树和k关联起来*/
END (tag)
    e = pop(S); /*栈顶元素出栈*/
    for each q ∈ e.RelatedQueryList; /*判断q是否满足*/
        if e is in q.uselessElementList then continue;
        for each k ∈ q
            if k ∈ e.KeywordList then q.InDegree --;
        if q.InDegree = 0 then
            OutputSRCT(q);
            /*输出和q的关键字匹配的最小相关连通子树*/
            for each stack element e ∈ S
                RemoveFrom(e.RelatedQueryList, q);
                Add e into q.uselessElementList;
            ProcessRelatedQuery(q);
            /*递归处理所有和q关联的后续查询*/
    E = top(S); /*将出栈节点信息传递到栈顶元素*/
    for each q ∈ e.RelatedQueryList
        if q ∈ E.RelatedQueryList then
            Insert q into E.RelatedQueryList;
    for each k ∈ e.KeywordList then
        t = ConstructTree(e, e.KeywordList[k]);
        Insert (k, t) into E.KeywordList;
    
```

变化如图 5 所示.当处理  $c_1$  时,  $a_1$  和  $b_1$  已经入栈保存, 首先找关键字  $k_1$  对应的查询, 即  $c_1.RelatedQueryList = \{Q_1\}$ ,  $c_1.KeywordList = \{k_1\}$ , 如图 5(a) 所示.当  $c_1$  出栈时, 将与  $k_1$  关联的子树插入栈顶元素  $b_1$  中, 同时将  $d_1$  节点入栈, 如图 5(b) 所示.这时,  $d_1.RelatedQueryList = \{Q_2\}$ ,  $d_1.KeywordList = \{k_2\}$ . 当遇到  $d_1$  对应的 END 事件时, 判断  $Q_2$  是否已经匹配, 由于已经匹配, 因此输出  $Q_2$  的一个解  $d_1/k_2$ . 同时, 调用函数 ProcessRelatedQuery( $Q_2$ ) 处理和  $Q_2$  关联的所

有后续查询 $\{Q_1, Q_4\}$ ,将二者的入度均减 1, $Q_1$ 和 $Q_4$ 在入度减 1 之后仍不为 0,因此无需继续处理,最后将 $d_1$ 的关键字对应的子树插入 $b_1$ 的关键字哈希表中并记录  $Q_2$  对应的无需处理的节点  $Q_2.uselessElementList=\{b_1, a_1\}$ .碰到 $k_3$ 时的栈状态如图 5(c)所示,这时 $b_1.RelatedQueryList=\{Q_1, Q_4, Q_5\}$ , $b_1.KeywordList=\{k_1, k_2, k_3\}$ .遇到 $k_4$ 时的栈状态如图 5(d)所示,这时 $b_1.RelatedQueryList=\{Q_1, Q_4, Q_5\}$ , $b_1.KeywordList=\{k_1, k_2, k_3, k_4\}$ .碰到 $b_1$ 的END事件时,需要判断 $Q_1, Q_4, Q_5$ 的查询匹配情况,在三者都匹配成功并输出满足条件的解之后,处理后续的查询 $Q_3$ 和 $Q_6$ .由于处理思路类似,此处不再详述.

## 5 实验

### 5.1 实验环境

本文的实验都是在处理器为 Pentium 4 2.8G,内存 512M,操作系统为 Windows XP 的机器上进行的.我们在 VC++6.0 环境下实现了基于 DAGBI 的查询算法以及文献[8]提出的查询处理算法.

### 5.2 实验数据集

本文所用数据集一部分由XML文档生成工具 xmlgen 自动生成,符合特定DTD.另一部份来自 <http://www.cs.washington.edu/research/xmldatasets/>,该网站提供各种特征的测试数据集.数据集按照大小分为两类:小文档(30~100k),大文档(500~2000k).

### 5.3 实验方案及评价标准

为了客观评价本文提出的基于 DAGBI 的查询处理方法,我们首先从这些数据集中抽取关键字,并使用随机算法从这些关键字中生成查询,每个查询的关键字个数也由随机算法指定,大小从 1 至 5 不等,查询个数从 10,000 到 100,000 不等.

**定义 1:查询的可满足性.**对于给定的查询  $Q$  和文档  $D$ ,若查询匹配的结果不为空,则说  $Q$  相对于文档  $D$  来说是可满足的.

我们用以下几个指标进行实验比较和分析:(1)关键字对应的查询个数.该指标可以反映使用 DAGBI 索引后,每个关键字关联的查询数目.(2)查询可满足性的检测时间.该指标用以和文献[8]提出的方法进行比较,如果一个文档中包含某个查询中的所有关键字,则该文档就是满足查询的文档.(3)查询求解的时间.由于已有文献中没有和本文解决问题相同的工作,该指标用以单纯的展示本文提出的算法的运行效果.(4)内存消耗量.该指标用以展示本文提出的算法在不同规模的文档上对大量查

询进行处理时需要的内存用量.

### 5.4 实验结果及分析

如图 6 所示,DAGBI 代表本文提出的索引,Old

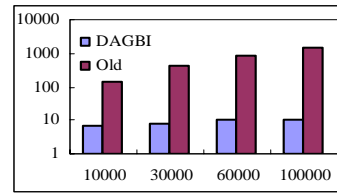


图 6 索引中关键字对应的查询数量对比

代表文献[8]提出的方法,横坐标表示查询的数量,可以看出,使用本文提出的索引方法,每个关键字直接对应的平均查询个数大大减少.原因在于文献[8]中采用的是倒排表机制,因此,随着查询数量的增加,每个关键字对应的查询数量增长非常快.

图 7 展示了本文提出的算法和文献[8]提出的算法在小文档上做可满足性检测的运行时间,横坐标表示文档的大小,单位是 k,图中的四个系列表示四种不同规模的查询数量.可以看出,本文提出的方法所用时间明显少于文献[8]提出的方法.原因在于:本文的方法在查询没有匹配成功的情况下,只需要检测少量查询即可.图 8 展示了本文提出的算法和文献[8]提出的算法在大文档上做可满足性检测的运行时间.

图 9 展示了本文提出的基于 DAGBI 的查询处

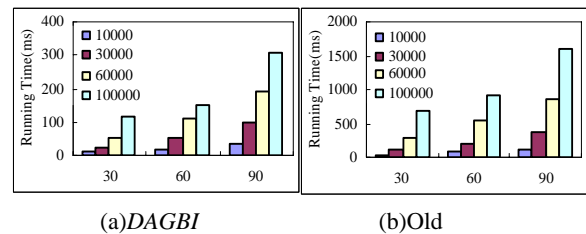


图 7 新旧两种方法使用不同数量的查询在小文档上的实验结果比较

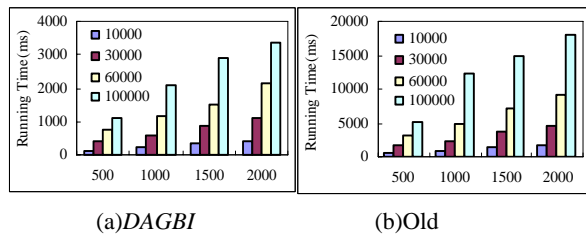
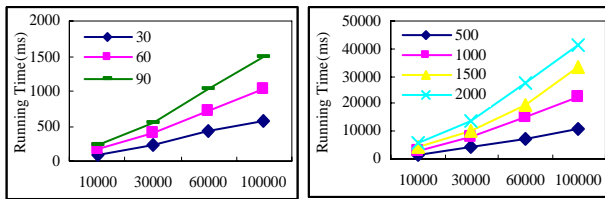


图 8 新旧两种方法使用不同数量的查询在大文档上的实验结果比较

理方法在进行查询处理时的实验结果,其中横坐标表示查询的数量,图中系列表示文档大小,这里有 7 种文档,分别是 30k、60k、90k、500k、1000k、1500k 和 2000k.可以看出,与只检测可满足性相比,返回给用户最小相关连通子树需要花费更多的时间.



(a)小文档

(b)大文档

图9 基于DAGBI的查询处理算法对不同规模的文档和查询进行处理的实验结果

图10展示了本文提出的基于DAGBI的查询算法在查询处理时的内存占用情况,其中横坐标表示文档大小,单位是k,图中的四个系列表示查询的规模,可以看出,不同数量的查询在相同文档上所用内存的量相差不多,而相同数量的查询在不同文档上对应的内存使用情况基本呈平缓线性增长的状态。这是因为数据只需缓存一份,然而,当查询数量保持不变而文档大小发生变化的时候,内存用量的变化比较明显,这是因为需要缓存的数据量也随之增大。

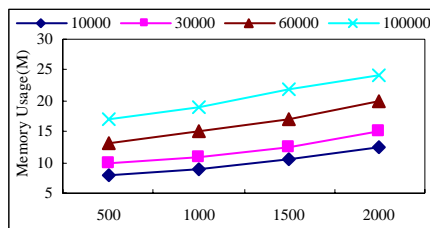


图10 不同规模的查询在不同文档上处理查询时的内存使用情况

## 6 结论

如何在XML数据流环境下为用户提供友好查询接口并在返回尽量少且不丢失语义信息的结果的同时处理大量查询的匹配检测是一个亟待解决的问题。本文在深入分析已有关键字查询技术的基础上,提出了一种对大量基于关键字的查询进行索引的方法,用以快速匹配查询;针对数据流的特点,提出了一种基于栈的临时结果缓存方法,用于中间结果的压缩存储和一次性处理大量查询的匹配检测问题;最后通过实验从不同角度对本文提出的算法的各项性能指标进行了实验验证。实验结果显示,由于使用DAGBI对系统中的大量查询进行索引,系统的整体处理性能得到了明显提升,查询的规模越大,系统性能的提升越明显。

已有工作中有在XML文档上处理关键字查询的方法,其中重要的一个问题就是对结果划分等级(Rank),这在本文中并未考虑,以后的工作中我们会

针对这一问题结合数据流的特点进行深入研究。

## 参考文献

- [1] Y Xu, Y Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In: ACM SIGMOD Conference. Maryland: ACM Press, 2005. 537-538
- [2] Y Li, C Yu, H V Jagadish. Schema-Free XQuery. In: *Proceedings of VLDB*. Toronto: Morgan Kaufmann, 2004.72-83
- [3] M. Altinel, M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information, In *Proceedings of VLDB2000*, pages 53-64. 2000.
- [4] I. Avila-Campillo, A. Gupta etc. XMLTK: An XML toolkit for scalable XML stream processing. In *Proceedings of PLAN-X*, 2002.
- [5] D. Olteanu, T. Kiesling etc. An evaluation of regular path expressions with qualifiers against XML streams, In *Proceedings of ICDE2003*, pages 702-704, 2003.
- [6] C Y Chan, P Felber, M N Garofalakis, R Rastogi. Efficient filtering of XML documents with XPath expressions. In: *Proceedings of ICDE*. California: IEEE Computer Press, 2002. 235-244
- [7] V. Josifovski, M. Fontoura and A. Barta. Querying XML streams. *The VLDB Journal*, 14(1): 2005: 197-210
- [8] F. Fabret, H.A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems, In *Proceedings of SIGMOD2001*.
- [9] New Era of Networks Inc.  
<http://www.neonsoft.com/products/NEONet.html>.
- [10] W. Xiaofeng, Z. Xin, X. Min, M. Xiaofeng, Z. Junfeng. Keyword Search on XML Streams. *Journal of Computer Research and Development*. 2006 43(Suppl.): 484~489

**周军锋**, 男, 1977年生, 博士研究生, 研究方向: xml数据库。

**孟小峰**, 男, 1964年生, 教授(博导), 研究领域: Web数据管理, XML数据库, 移动数据管理。曾先后在香港中文大学、香港城市大学、新加坡国立大学访问研究。主持或参加过十多项国家科技攻关项目、国家自然科学基金以及国家863项目。

**张新**, 男, 1983年生, 硕士研究生, 研究领域: xml数据库。

**黄静**, 女, 1984年生, 硕士研究生, 研究方向: xml数据库。

联系人： 周军锋  
手机： 13611246044  
实验室电话： 010-62519440  
电子邮件： [zhoujf@ysu.edu.cn](mailto:zhoujf@ysu.edu.cn)  
通讯地址： 中国人民大学信息学院理工楼配楼（原信息楼）103B  
邮编： 100872