基于 DMB 手机 EPG 系统的 XML 数据存储与传输技术研究*)

吴君林　许龙飞　赵聚雪　司徒浩臻
(暨南大学计算机科学系　广州 510632)

摘　要　数字多媒体广播电子节目指南(DMB EPG)是数字电视行业内一项重要的应用,也是用户通过手机获取节目信息的必不可少的工具。在 DMB EPG 系统中,减小文档大小、改善传输性能非常重要。本文探讨了 EPG 系统的 XML 数据存储技术和基于传输性能的 XML 数据编码问题。在分析研究 DMB EPG 协议原理的基础上,设计和实现了 DMB EPG 系统的 XML 数据存储和二元编码过程,并对三种编码方案的性能进行了分析比较。

关键词　数字多媒体广播,电子节目指南,纯 XML 数据库,二元 XML

Study of XML Data Storage and Transportation Technology Based on DMB EPG

WU Jun-Lin　XU Long-Fei　ZHAO Ju-Xue　SITU Hao-Zhen
(Department of Computer Science, Jinan University, Guangzhou 510632)

Abstract　Digital Multimedia Broadcasting Electronic Programme Guide (DMB EPG), is an important application of digital TV and an indispensably tools for users to know about programs by mobile telephone. In the DMB EPG system, it's very important to reduce the document size of transportation and improve transportation efficiency. In this paper, XML data store technology based on EPG system and XML data encoding problem based on transportation efficiency are discussed. On the basis of studying a series of protocols, principles of DMB EPG, the DMB EPG XML document storage scheme and binary encoding process are designed and implemented. Additionally, the efficiency of three encoding schemes is analyzed and compared.

Keywords　DMB, EPG, Native-XML database, Binary XML

1　引言

电子节目指南(Electronic Programme Guide),简称 EPG,是数字电视技术中一项极其重要的应用,是用户获取大量电视节目信息必不可少的工具,也是优良的广告载体。随着在 DAB 基础上发展起来的 DMB 技术标准已成为移动手机电视的主流标准。DMB 手机电视的电子节目指南(DMB EPG)成为数字电视的新兴领域和研究热点。由于 DMB EPG 系统采用了基于 XML 的数据交换技术,减少数据传输量改进传输性能就变得尤为重要。本文介绍了该系统前端机的 XML 数据存储技术以及基于传输性能的 XML 数据编码问题,并采用二元 XML 设计和实现了 DMB EPG 的编码器。

目前国际上研究 DMB 手机电视(EPG 是 DMB 手机电视系列产品之一)的主要有英国 Gcap 媒体集团、德国蓝宝、韩国三星,LG,英国 TTP、荷兰爱迪德等。国内的有北京悦龙数字广播、上海东方明珠、广东移动电视、香港商业电台等。DMB EPG 国内标准尚未出台,国内一些著名高校的专家对 EPG 系统的研究主要是基于 DVB 协议的数字电视 EPG,还较少看到基于 DMB(DAB)协议方面的 EPG 研究成果。

EPG 系统是一组系统,我们称之为前端系统和后端系统。前端也就是 EPG 数据的发送端,后端就是接收终端。因篇幅关系,本文研究仅涉及其前端系统的相关部分。

2　DMB EPG 系统的数据存储

图 1 是 DMB EPG 系统的前端机的 EPG 数据的转换图。

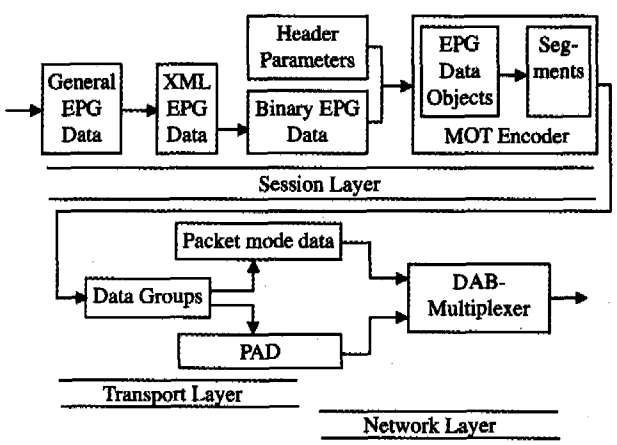

图 1　DMB EPG 数据流程图

如图 1 所示,EPG 数据的转换过程经历 3 层,分别是 Session Layer(会话层)、传输层(transport Layer)和网络层(network layer)。

以下仅介绍会话层的数据流程。会话层的输入是一般的无结构的 EPG 数据,我们将它统一转换为关系数据库数据,然后再转换为 XML 格式的 EPG 数据。这里的 XML 数据符合 EPG 的 Schema 模式文档定义的数据规范;XML 文档经过二进制编码转换成二进制的 EPG 数据;这些数据和一些 Header 参数经过 MOT 编码器被封装成 MOT 对象,MOT 对象的组成如图 2 所示。

其中的 Header core 和 header extension 由前面 header

吴君林　硕士研究生,主要研究方向为数据库系统与知识工程;许龙飞　教授,主要研究方向为数据库系统与知识工程等。

parameters 获得，body 由前面的 binary EPG data 得到。这些 MOT objects 被分成一系列的片段(Segment)，这些 Segments 就是会话层的输出，如图 3 所示。

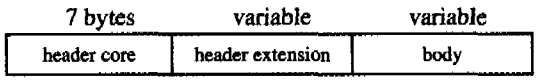

图 2　MOT 对象的结构

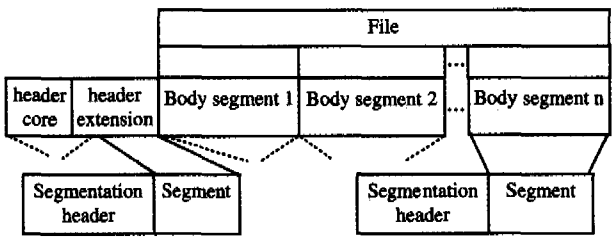

图 3　对象分块图

我们针对会话层的数据转换过程设计了一种数据处理模型，如图 4 所示。

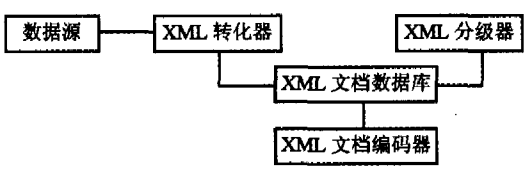

图 4　session Layer 数据处理模型

在这个模型中有 4 个子模块，分别是 XML 文档转换器、XML 分级器、XML 文档编码器以及位于这 3 个模块中心的 XML 文档数据库，我们称之为 EPG XML 数据库。DMB EPG 以 XML 文件作为分级编码的单位，所以 XML 数据库以文件作为 EPG 数据的存储单位。下面根据 EPG 数据的特点提出一种 EPG XML 数据库设计方案。

在这个模型中，数据源经过 XML 转换器转换为符合 Schema 模式的 XML 文档，这些文档存储在 XML 文档数据库资源库中。XML 文档分级器取出资源库中的 XML 文件，将其分割成 Basic Profile 和 Advance profile 2 个级别的文件，并将分级后的文件存储在 XML 资源库中。编码器取出分级后的 XML 文件将其编码成 Binary XML 文件，再将其存入 XML 资源库中。数据源在经过以上处理过程后，便可以将资源库中 Binary xml 文件上传到 MOT 编码器进行后续处理。

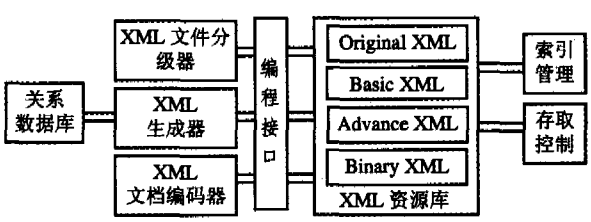

图 5　EPG XML 数据库模型

如图 5 所示，它是基于文本的纯 XML 数据库，以文档作为数据的存储单位，对 XML 数据库的读取查询更新操作都是以文本进和文本出的方式进行的。该数据库中的每个文档都对应一个逻辑模型，由于 EPG 数据是采用 Schema 模式定义的，XML 文档也必须符合 epgSchedule_11. xsd，epgSI_11. xsd.，epgDataTypes. xsd 三个模式文档，我们便将这三个模式文档作为 EPG XML 数据库文档的逻辑模型。下面介绍该模型的工作机制及每个模块的功能。

(1)XML 资源库　XML 资源库是为 XML 文件的存储所申请的空间。由于 XML 文件按照处理流程可以分为 4 类：Original XML，Basic XML，Advance XML，Binary XML，因此在 XML 资源库中开辟四个子库分别用来存储对应的 XML 文件。在图 5 中，Original XML 是用来存储从关系数据库中的结构化数据并经过 XML 生成器后转换成的 XML 文件；Basic XML 和 Advance XML 分别用来存储从 Original XML 文档经过 XML 文件分级器后生成的 Basic XML 文档和 Advance XML 文档；而 Binary XML 则用来存储从 Basic XML 和 Advance XML 中的文档经过 XML 文档编码器生成的二进制 XML 文件。

(2)编程接口　EPG XML 数据库根据需要提供了一系列供客户端调用的编程接口，包括将 Original XML 文件分级的接口，生成 XML 文件的接口，XML 文档编码的接口，还保留了一些暂未实现的接口，如 XML 文件的查询添加修改删除等接口。

(3)索引管理　在 XML 资源库中建立如下的索引方案，用户可根据这种索引来查询获取需要的文件。由于资源库以文件作为存储的基本单位，因此建立索引也是通过文件名来建立索引 。

(4)存储管理　EPG XML 数据库的存储管理是以 XML 文件为存储单位进行的。

3　XML 的传输性能问题

随着基于 XML 的数据交换技术的应用的深入，业界发现 XML 的一个越来越无法忍受的缺点，即：效能缓慢。该问题促使我们设法加速 XML 的流量，改善 XML 的传输性能，这就引出了 XML 的数据表示问题，也有人称之为 XML 数据的“肥胖”问题。

在改进 XML 传输性能的这个问题上，目前可用的方案有：ASCII 编码，压缩编码如 GZIP，为 XML 专门设计的编码以及二元 XML[5]等。

(1)ASCII 编码。为计算机提供了一种存储数据和与其他计算机及程序交换数据的方式。XML 文档最原始的编码方式便是采用 ASCII 的编码机制。

(2)压缩编码，如 GZIP。这种编码严格遵循 XML 文本特性，是基于文本的转换。这种类型的转换主要和文档大小有关。这种类型的任何算法都能方便地用于 XML 文档的文本表示。这类算法一般不会改善文档的处理速度，因为增加了压缩和解压缩的时间。

(3)专门设计的 XML 编码，如 XMill，XBIS 等。这类编码比一般的基于文本的转换更进一步。设计这种类型的方法是为了通过某种编码表示，以能够减少文档大小、降低处理开销或者二者兼具的方式利用 XML 的结构。从编码形式上重构原来的文档，仅仅损失 XML 认为无关的那些信息。由于损失了信息，原始文档和重构后的文档进行直接文本比较，可能不同，但是对于符合 XML 规则的任何应用程序而言它们是等价的。

(4)二元 XML。目前还没有一个通用的标准，一般根据应用程序使用的文档结构量身定做，发送方和接收方必须事先就具体的结构达成一致，并且实现适当的编码程序/解码程序，针对应用程序定制编码的多数方法都是基于交换文档的

W3C XML Schema 规范。模式中包含的类型和结构信息用于生成定制的编码程序/解码程序代码，可能包括能够代表文档数据内容并直接与编码程序/解码程序交互的对象[5]。它的优势在于采用二元格式后，传输代价和执行效能都有了显著提高。然而它的劣势也是明显的，目前只应用于特定行业、特定用途，存在应用范围小，不兼容、丧失互通性的问题[7]。美国的二元 XML 标准化工作组已申请国家专利研究该课题[8]。

二元 XML 有 WBXML，PDOM，采用 ASN.1 规则的二进制 XML 编码三类编码方式[9]。

4 采用二元 XML 为 DMB EPG 编码

由于 DMB EPG 的带宽有限，终端接收设备的内存容量以及处理器都是非常宝贵的资源，尽量压缩数据的传输量对于降低传输消耗，减小终端设备的负担是非常必要的，DMB EPG 采用 Binary XML 的编码方案[2]，由于 W3C 尚没有建立一套通用的标准，ETSI 建立一套广播系统(行业内)通用的二进制编码规范[1,2]。采用二元 XML 显著降低了数据的传输量，改善了传输性能。下面重点介绍 DMB EPG 的二进制编码协议以及我们针对该协议所设计的编码算法。

4.1 Binary XML 编码协议

4.1.1 编码模式

EPG 采用 tag-length-value 的模式进行编码，每个元素和属性都有一个其独有的标识 Tag，然后是标示该元素或属性含有数据的长度(以字节数为单位)Length 标识，最后是数据部分，如图 6 所示。

Tag	Length	Value	Tag	Length	

图 6 编码模式结构

这种编码结构可以使接收终端很方便地跳过那些它不需要的数据或者终端无法识别的数据。为了便于描述协议，可使用类似于 C 语言的伪代码描述，某些词语是缩写词，如 Uimsb 表示 Unsigned integer，Most significant bit first。生成最终二进制对象可以用图 7 描述。

Syntax	Size	Type
binary_object(){ top_level_elementan()		

图 7 二进制对象结构

Syntax	Size	Type
element(){		
element_tag	8 bits	uimsbf
element_length	8 bits	uimsbf
if (element_length==0xFE){		
extended_element_length	16 bits	uimsbf
}		
if (element_length==0xFF){		
Extended_element_length	24 bits	uimsbf
}		
for (i=0;i<element_length or		
extended_element_length;i++){	8 bits	uimsbf
element_data_byte		
}}		

图 8 元素编码结构图

每个二进制对象只含有一个顶级元素(top level element)。

4.1.2 元素编码结构

element_tag 这个字节用来标志元素类型，每个 element_tag 标示一个唯一类型的元素，它和 element 之间是 1-1 对应的关系，如果将来需要添加新的元素扩展，这里保留的未定义的 Tag 可供将来扩展之用。Element_length 这个字节用来标示这个元素包含的 data 的长度，注意它不包括本身的 tag 和 length 部分，但是包括子元素或属性的 tag 和 length 部分。如果 data 的长度大于 0xFD，小于等于 $2^{16}-1$，则用一个额外的 16 位的 extended_element_length 来标示数据长度，此时 element_length 等于 0xFE，当解码时读到 element_length 等于 0xFE，则自动根据下 2 个字节来确定 data 的长度。如果 data 的长度大于等于 2^{16} 小于 $2^{24}-1$，则用一个额外的 24 位的 extended_element_length 来标示数据长度，此时 element length 等于 0xFF，同理当解码时读到 element_length 等于 0xFF，则自动根据下 2 个字节来确定 data 的长度。Element_data_type：这段字节包含了该元素的属性，CDATA(字符串)以及子元素的编码。它们按照属性，子元素，CDATA 的顺序进行编码。

4.1.3 属性编码

Syntax	Size	Type
attribute(){		
attribute_tag	8 bits	uimsbf
attribute_length	8 bits	uimsbf
if (attribute_length==0xFE){		
extended_attribute_length	16 bits	uimsbf
}		
if (attribute_length==0xFF){		
extended_attribute_length	24 bits	uimsbf
}		
for (i=0;i<attribute_length or		
extended_attribute_length;i++){	8 bits	uimsbf
attribute_data_byte		
}}		

图 9 属性编码结构

如图 9 所示，属性编码结构类似于元素结点的编码，这里不再做详细解释。需要注意的是：如果一个属性存在默认的属性值，当该属性取默认值时，该属性不必编码，其所在的元素结点无须将该属性的编码包含进去，因为如果一个属性取默认值，当终端解码时找不到该属性的 tag 时就自动认为该属性取默认值，从而减少信息通信的代价。

4.1.4 CDATA 和 String 的编码

Syntax	Size	Type
cdata(){		
cdata_tag	8 bits	uimsbf
cdata_length	8 bits	uimsbf
if (cdata_length==0xFE){		
extended_cdata_length	16 bits	uimsbf
}		
if (cdata_length==0xFF){		
Extended_cdata_length	24 bits	uimsbf
}		
for (i=0;i<cdata_length or		
extended_element_length;i++){	8 bits	uimsbf
cdata_data_byte		
}}		

图 10 CDATA 和 String 的编码结构

所有的 CDATA 和 String 部分(除了属性值 String 外)的编码结构如图 10 所示,所有的 cdata_tag 都为 0x01,其他部分含义类似于上面的解释。值得提出的是,这里 CDATA 和 String 只适合对元素结点值,而不适合属性值,所有的 CDATA 和 String 将采用 UTF-8 编码(见 ISO/IEC 10646[4]),在 0xE000 和 0xF8FF 之间的用户自定义字符区的编码将被忽略,因为终端是无法对这些编码解码展示的。

4.1.5 枚举类型编码

由于枚举类型的属性值的数量有限,可以对每种属性采用一个字节的编码,终端解码器读取该字节时便自动将其影射为相应的枚举属性值。

4.1.6 其他公共数据类型编码

其他公共类型编码根据数据的特有的类型和结构设计编码,譬如属性 version 采用 16 位无符号整数等。

4.2 基于 ETSI 371 协议编码器的设计

基于 ETSI 371 协议编码算法的设计思路:

(1)首先根据协议,属性编码结构设计所有共用的属性类型编码为:

属性 tag	属性 length	属性值 data

(2)根据编码协议,枚举类型属性编码结构设计所有的枚举类型编码是:

属性 Tag	0x01(枚举类型 只需一个字节表示 data)	一个字节表示的 data

(3)根据协议,设计节点类型的编码结构 java 的 byte[] 数组对象 A,由 A. length 得到该节点的长度。

节点 tag	节点 data 长度	属性 1 编码	…	子节点 i 编码	…	CDATA (String)编码
			..		.	

(4)解析 XML 文件生成树,然后对 XML 树进行深度优先搜索的一次遍历。如图 11 所示,遍历的顺序依次是 1,2,5,3,6,7,10,11,8,4,9。

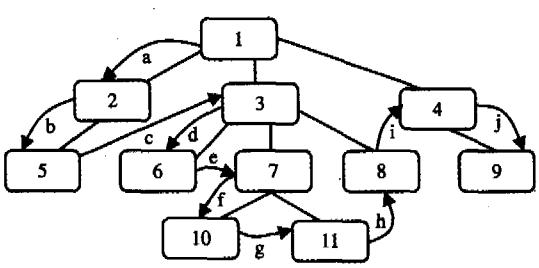

图 11 XML 树的遍历(深度优先搜索)

当访问节点时:

a)如果属性部分不为空,则对属性部分编码;

b)如果是非叶子节点,访问各个子节点的编码,并返回子节点的编码的 byte 数组;

c)如果存在节点值 CADA 或者 String,则对 String 部分编码;

d)连接以上编码 byte 数组得到该节点的 length;

e)连接该节点的 tag, length, data 部分,组成该节点的编码 byte 数组。

伪码表示为:

```
Public byte[] code (nodeType node)
{
  byte[] binaryData;//存储的编码后的 byte 数组
  byte [] temp=null;//连接各个编码部分的临时 byte 数组
  byte [] attribute;//存储所有属性的编码
  byte [] element;//存储子元素的编码
  byte [] cdata;//存储节点 cdata 部分编码
  byte tag=node. tag;//由协议定义
  byte [] length;//根据 data 字节的长度来确定使用几个字节的
                 length 来标示长度
  if  (node= =null)
      Return null;
  While (node has attributes which isn't coded){
   Attribute= codeAttribute(attribute(i));//对属性编码
   temp=ConcatBytes(temp, attribute); //返回连接 temp 和 attrib-
                                      ute 字节数组}
  While (node has more child Elements )
  {   element= code(ChildElement)
  //对子节点编码,同一类型的节点可共用一个编码接口 ,不同节点
  类型重载此接口。
      temp=concatBytes(temp, element);
  // 返回连接 temp 和 attribute 字节数组
  }
  if  (node has CDATA or String){
    cdata =Code(cdata or string);
    temp=concatBytes(temp, cdata);
   // 返回连接 temp 和 cdata 字节数组
   }
  if (temp= =null)
  //此元素没有属性也没有子元素和 CDATA 部分
  {Length=new byte [1];Length [0]=0x01;}
  else
   length=getLengthFromData (temp. length);
  //根据 data 的长度确定 length 的字节长度(1,3,4)并返回 Length
  部分的编码, temp. length 表示 temp 数组的长度.
  element=ConcatBytes(tag, length , temp);
  return element;
}
```

4.3 Binary XML 性能分析

在采用 Binary XML 为 EPG 编码后,我们从理论分析和实验上分别采用 ASCII 编码,GZIP 压缩编码,以及采用 Binary XML 编码,读一个 EPG XML 文件,写一个 EPG XML 文件所需要的时间,以及各种编码文档占用的存储空间,结果如表 1 所示。

表 1 3 种编码性能的比较表

性能指标 \ 编码方案	ASCII	GZIP	Binary Encoding
Read Time	较快	较慢	较快
Write Time	较快	较慢	较慢
Document Size	大	较小	较小

在表 1 中,先看读时间,由于读 GZIP 文本时首先要解压缩还原成 XML 文挡,因此相对 ASCII 编码读速度较慢,而在读 Binary XML 时,可以直接根据二进制编码结构读取,不需要还原成 XML 文挡,所以读速度较快。再看写时间,由于 GZIP 增加了压缩时间,Binary 编码增加了解析 XML 结构然后编码的时间,因此后两种写速度都较慢。最后看三种编码方案后文挡大小,由于 GZIP 的压缩,二元编码的重构,后两种方案在不同程度上都减小了文挡大小。由于终端更为关注文档的表示大小以及读取的速度,因此二元 XML 得到了青睐。

由于 XML 文档的传输性能主要取决于 XML 文档转换后的大小,经过一定数量 EPG XML 文档的实验后,就 XML 文档的几种编码后的大小进行比较,可见 GZIP 和二元编码在减小文档大小上效果突出,如图 12 所示。值得注意的是不同的文档由于内容不同,即使其 ASCII 编码大小相同,经过 GZIP 压缩或经过本文的 Binary Encoding 编码后其文档大小也会可能不同,所以针对某个具体的文档,二元编码不一定比 GZIP 压缩比高。

图 12 XML 的文档编码后的性能比较

总结 本文在分析研究了 DMB EPG 相关协议原理的基础上，设计实现了 DMB EPG 的数据存储方案，分析了 XML 传输性能问题，并采用二元 XML 设计，实现了 DMB EPG 的编码器，最后分析比较了三种编码的性能。

进一步的工作有以下几个方面值得研究，如数据经过编码后的后续处理过程，即文档封装成 MOT 对象、分段、打包和传输技术以及终端如何进一步改善解码技术等。

参 考 文 献

1 ETSI TS 102 818 (2005-01), XML Specification for DAB EPG. http://www.worlddab.org/irc.aspx? sub=10

2 ETSI TS 102 371 (2005-01). Transportation and Binary encoding for DAB EPG. http://www.worlddab.org/irc.aspx? sub=10

3 ETSI EN 301 234(1999-02). Multimedia Objects Transportation Protocol. http://webapp.etsi.org/action/OP/OP20060519/en_301234v020101o.pdf

4 ETSI TS 300 401(2001-05). Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. http://www.worlddab.org/irc.aspx? sub=10

5 万常选. XML 数据库技术. 清华大学出版社，2005

6 Sosnoski E. 改善 XML 的传输性能. http://www.ibm.com/developerworks/cn/xml/

7 Bayardo R J, Gruhl D, et al. An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing. http://www2004.org/proceedings/docs/1p345.pdf

8 XML Binary Characterization Working Group. http://www.w3.org/XML/Binary/

9 李莉明. 二进制 XML 浅析. 计算机与数字工程，2005，33(22)：1～5

(上接第 80 页)

3.3 XR 树＋XR-Stack 算法

在 Anc_Desc_B^+ 算法中，利用 B^+ 树索引能够有效地跳过后裔列表中所有不参加连接的结点，但是对于祖先列表中所有不参加的结点，并不能利用 B^+ 树索引来有效地跳过，这是因为：在祖先列表中无法利用 B^+ 树索引直接定位结点的第一个可能的祖先结点，需要多次定位才能跳到结点的第一个可能的祖先结点。为了解决这个问题，XR 树索引被提出[11]，基于 XR 树索引不仅能够有效地跳过后裔列表中所有不参加连接的结点，而且能够跳过祖先列表中所有不参加连接的结点。例如，图 3 中若要找到后裔结点 9 的父结点 8，在 Anc_Desc_B^+ 算法中不能够一次定位到结点 9 的父结点，需要多次在祖先列表中进行定位才能够找到结点 9 的父结点 8，而 XR 树＋XR-Stack 算法能够在祖先列表中一次定位到结点 9 的父结点 8。

3.4 XISS 系统

XISS 系统[11]的主要思想是：将一个正则路径表达式分解成子路径表达式，将子路径表达式产生的中间结果进行结构连接操作得到最终的查询结果，并且该系统支持文档更新操作。

XISS 系统主要由五个索引组成：元素索引、属性索引、结构索引、名称索引、值索引。这五个索引采用 B^+ 树作为索引结构。

XISS 系统将正则路径表达式分解成五个基本的子表达式：(1)单元素单属性子表达式：通过查找元素索引、属性索引得到查找结果。(2)一个元素和一个属性子表达式：EA-join。(3)具有两个元素的子表达式：EE-join。(4) Kleene cloure (＋，＊)：KC-join。(5)子表达式之间的联合操作。将分解的子表达式所得的中间结果进行连接得到最终的正则路径表达式的查询结果。

XISS 缺点：只能解决单路径查询，对于分支路径查询效率较低。

结束语 本文主要研究了 XML 查询技术中的两种主要方法，其中基于路径索引的 XML 查询方法只能够解决单路径查询，但是路径索引的创建不受 XML 文档结构的约束，即 XML 文档可以是树结构，也可以是图结构。基于编码方式下的 XML 索引查询方法能够有效地解决分支路径查询，但是这种方法都对相应的 XML 文档有要求，其所要查询的 XML 文档为树形结构。怎样很好地将上述两种 XML 查询方法中的优点结合起来，是将来研究的热点之一。

参 考 文 献

1 Goldman R, Widom J. DataGuide: enabling query formulation and optimization in semistructured databases. In: 23th International Conference on Very Large Data Bases, pages, Athens, Greece, 1997. 436～445

2 Milo T, Suciu D. Index structures for path expressions. In: 7th International Conference on Database Theory, Jerusalem, Israel, 1999. 277～255

3 Kaushik R, Shenoy P, Bohannon P, et al. Exploiting Local Similarity for Efficient Indexing of Paths in Graph Structured Data. In: 10th International Conference on Database Theory, San Jose, California, USA, 2002. 129～140

4 Chen Q, Lim A, Ong K W. D(k)-index: An adaptive structural summary for graph-structured data. In: Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data, San Diego, California, USA, 2003. 134～144

5 He Hao, Yang Jun. Multiresolution Indexing of XML for Frequent Queries. In: Proceeding of the 20th International Conference on Data Engineering. Boston, USA, 2004. 683～694

6 Chung C, Min J, Shim K. APEX: An adaptive path index for XML data. In: Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data, Madison, Wisconsin, 2002. 121～132

7 Cooper B F, Sample N, Franklin M J, et al. A Fast Index for Semistructured Data. In: Apers P M G, eds. Proceedings of the 27th VLDB International Conference on Very Large Database, Rome, Italy, SanFrancisco: MorganKaufmannPublishers, 2001. 341～350

8 Jiang Haifeng, Lu Hongjun, Wang Wei, et al. XR-Tree: Indexing XML Data for Efficient Stuctural Joins. In: Casati F, et al. eds. Proceedings of 19th IEEE ICDE International Conference on Data Engineering, Bangalore, India, 2003. 253～263

9 Chien S Y, Vagena Z, Zhang Donghui, et al. Efficient Structural Joins on Indexed XML Documents. In: Papadias D, et al. eds. Proceedings of the 28th VLDB International Conference on Very Large Database, Hong Kong, China, 2002. 263～274

10 Li Quanzhong, Moon B. Indexing and Querying XML Data for Regular Path Expressions. In: Proceedings of the 27th VLDB Conference, Roma, Italy, 2001. 361～370

11 Wirth N. Type Extensions. ACM Transactions on Programming Language and Systems, 1988, 10(2): 204～214

12 Dietz P F. Maintaining Order in a Linked List. In: Lewis H R ed. Prcoeedings of the 14th Annual ACM Symposium on Theory of Computing. San Francisco, California, USA, 1982. 122～127

13 Bruno N, Koudas N, Srivastava D. Holistic Twig Joins: Optimal XML Pattern Matching. In: Franklin M J, ed. Proceedings of the 21th ACM SIGMOD International Conference on Management of Data. Madison, Wisconsin, USA, 2003. 310～321