数据管理模块（Data Manager）概要设计说明书

负责人：罗道峰，安靖
编写人：安靖
系统版本号：OrientX Version 1.5
完成时间：2004/2/20
开发单位：中国人民大学 IDKE 实验室 XML 工作组

1．引言

编写本说明书是为了向用户介绍 OrientX 系统中，数据管理模块的设计思路及使用方法。本模块重点在数据在系统中的存储模型，数据模式的建立，数据的导入、导出和和如何向上层提供数据。

下面各章节的具体安排：第 2 节概述，介绍相关的背景知识，此模块的功能，以及此模块在整个系统中的地位和作用。第 3 节总体设计，介绍整个模块的处理思想，模块内部由哪些小模块构成，以及它们之间的关系。第 4 节接口设计为上层模块提供服务的接口，第 5 节数据结构设计，包括逻辑结构设计和物理结构设计两部分。第 6 节介绍出错处理。

2．概述

本模块介绍数据在 OrientX 系统中的管理，包括数据的存储方法，数据的导入导出，数据的存取等。向上层查询模块提供接口的是 DataManager 类。它通过调用 Schema 类实现模式文档的解析和数据集的建立，通过调用 ImportHandler 类实现 XML 格式的文档的导入，上层模块利用它的导航接口可以提取想要的数据对象。

3．总体设计

3.1XML 解析的相关知识

XML 文档有自己的格式，要经过相应的软件进行转换，转换成应用需要的文件格式。其中语法分析器和应用程序之间有两种接口：DOM 和 SAX

3.1.1 DOM 接口

DOM（Document Object Model）节点有 Document、Element、Comment、Type

等等节点类型，其中每一个 DOM 文档必须有一个 Document 节点，并且为节点树的根节点。它可以有子节点，或者叶子节点如 Text 节点、Comment 节点等。作为基于对象的接口，DOM 通过在内存中显示的构建对象树来与应用程序通信，对象树是 XML 文件中元素树的精确映射。

3.1.2 SAX 接口

SAX 的全称是 Simple APIs for XML，也即 XML 简单应用程序接口。SAX 提供了一种对 XML 文档进行顺序访问的模式。它的基本原理是由接口的使用者提供符合定义的处理器，XML 分析时遇到特定的事件，就去调用处理器中特定事件的处理函数。SAX 不在内存中显示的构建文档树，它使应用程序能用最有效率的方式存储数据。

在 OrientX 系统中这两个接口都用到了。在 parser schema 文件时用的是 DOM 接口，在 parser 普通的 XML 文档时用的是 SAX 接口。

3.2Schema Class

3.2.1 功能

根据用户指定的模式建立相应的数据集。在 OrientX 系统中，每个数据集有对应的模式，相同模式的 XML 文档存储在一个数据集中。也就是说本系统是按照数据集组织文档的。用户在创建数据集时需要指定相联系的模式。

3.2.2XML 模式的相关知识

XML 模式是指用来描述 XML 结构、约束等因素的语言，例如 XML Schema、XML DTD、XDR、SOX 等。目前有的比较多的是 XML DTD 和 XML Schema。

- XML DTD（Document Type Define）是目前使用最广泛的 XML 模式定义，但是，由于 XML DTD 并不能完全满足 XML 自动化处理得要求，例如不能很好实现应用程序不同模块间的相互协调，缺乏对文档结构、属性、数据类型等约束得足够描述等等，所以 W3C 于 2001 年 5 月正式推荐 XML Schema 为 XML 的标准模式。
- XML Schema 的格式与 XML DTD 的格式有着非常明显的区别，XML Schema 事实上也是 XML 的一种应用，也就是说 XML Schema 的格式与 XML 的格式是完全相同的，而作为 SGML DTD 的一个子集，XML DTD 具有着与 XML 格式完全不同的格式。
- OrientX 系统中，采用的是 XML Schema，并且利用 Apache 的 DOM 接口来解析用户指定的 Schema 文档。

3.2.3 实现

利用 Apache DOM 解析用户指定的 schema 文件建立数据集，并生成相应的

其他的数据结构。

- 每个 XML 文档都有个唯一的根节点，但是 schema 的语法中并没有明确规定各元素出现的顺序，也没有明确的指明哪个元素是对应 XML 文档的根元素，所以需要用户在参数中自己指定。前提是用户了解自己要用到的 schema。
- 模式中可能定义了两个没有祖先后代关系的元素间的引用关系，一般都是用属性来指定。通常是元素 A 有个 ID 类型的属性，根据该属性的值可以唯一确定元素 A，另一种元素 B 有个 IDREF 类型的属性，它的取值只能为空，或者为相联系的 ID 属性的某个值。这种情况类似于关系数据库中的外码完整性约束。在解析某个模式时，应该根据这些定义，来建立所有的引用对。
- 每个数据集都分配一个全系统唯一的 id，从 1 开始。
- 每个数据集建立的同时，会建立相应的 oid 集，oid 集的相关知识参考文章《数据存储模型设计》。
- 在解析新的模式文件的时候要生成相应的 DTDTREE，它是该模式文档在内存中的表示。DTDTREE 上的结点，称为 dtdnode，是模式中定义的元素。对每个 dtdnode 编码，这样能很快得比较两个 dtdnode 是否相同或者是否有祖先后代关系。

3.3ImportHandler Class

3.3.1 功能

实现从外部导入 xml 格式的数据到数据库中。根据指定的存储模式划分记录的粒度、选择记录的存储顺序、分配存储空间等。

3.3.2 系统中四种数据存储模型

在关系数据库中一个元组是一个记录，记录可以是顺序存放的，也可以是聚簇存放的。在 native xml 中，并不一定是一个元素对应一个记录，元素和记录之间的对应关系我们称之为记录的粒度划分。根据记录粒度的划分和记录的存储顺序的不同组合，OrientX 系统中有四种存储模型：DEB，CEB，DSB，CSE。
记录的粒度就是记录中包含几个结点。直观地，记录的粒度有三种：

1) 结点级　每一个结点是一个记录。
2) 子树级　一棵子树是一个记录。
3) 文档级　整个文档是一个记录。

记录的存储顺序，也就是记录在物理上的相邻关系。记录的存储顺序一般有以下几种：按深度优先顺序存储：按广度优先顺序存储：按同类记录聚簇存储。
把这两个因素综合考虑，就诞生了 7 种存储方法：

	结点级 (Element-based)	子树级 (Subtree-based)	文档级 (Document-based)
深度优先(Depth)	DEB	DSB	DB
广度优先(Broad)	BEB	BSB	
同类聚簇(Clustered)	CEB	CSB	

3.3.2.1DEB

DEB 存储方法是最简单的存储方法。每个元素结点是一个记录，记录按照深度优先的顺序存储。由于 SAX Parser 的后序处理特性，实际在处理中，记录是按照后序遍历的顺序存储的。这种方法很简单，只要随着 startElement()事件流生成相应的记录结点，随着 endElement()事件把记录结点写到磁盘就可以了。

3.3.2.2 CEB

CEB 存储方法与 DEB 存储方法类似，也是一个元素结点就是一个记录。不同的是，相同类型的元素结点，将会聚簇存储在一起，组织成一条该类型结点的实例链表。OrientStore 将记录该链表的链头，在查询中如果需要扫描某类型的结点，直接从链头开始扫描即可。

3.3.2.3 DSB

DSB 存储方法稍微复杂，它按照物理块大小划分子树作为记录，记录按照深度优先的顺序存储。同样地，由于 SAX Parser 的后序处理特性，实际上是按照后序遍历顺序存储的。

但是子树的大小不会总是那么理想。有的时候，需要若干棵子树加起来，才接近一个物理块的大小。为了处理这种情况，DSB 引入一种特殊的记录类型，叫做 Proxy Record。这个概念是 Natix 中提出来的。Proxy Record 不是由一棵子树组成的，而是由若干棵比较小的子树组成的，这几棵子树的大小之和，接近一个物理块的大小。让我们以下图为例，说明 OrientStore 是怎么划分 Record 子树和 ProxyRecord 子树的。为了简单起见，假设每个物理块能存放 4 个元素结点（实际是看这些结点打成字节流之后的长度，而不是简单的结点个数）。

3.3.2.4 CSB

CSB 存储方法的记录是一棵子树，而且同类型记录聚簇存储。所谓记录的类型，是指记录的根结点的类型。前面看到，DSB 划分子树的方法是按照物理块的大小，划分出来的记录类型是没有规律的。在 CSB 里面，由于要求同类型记录聚簇存储，系统需要某种划分子树的标准，使划分出来的子树只是某些特定类型。所以关键问题是选定某些特定类型的结点作为记录结点。既然是要选定某些特定类型的结点作为记录结点，OrientStore 需要预先知道都有哪些类型的结点。换句话说，用 CSB 方法存储的数据需要模式(DTD 或者 Schema)。模式是数据的定义，定义了数据的结构。就像关系数据的数据字典，定义了表的结构。

3.3.3 实现

利用 Apache Sax2 parser 将用户指定的文件转换成 OrientX 数据库支持的格式并存储在库中。

- 在导入文档时要根据它的模式指定要导入到的数据集，为其选择存储模式。
- 每个数据文件在 OrientX 中都被分配一个数据集内部唯一的 id，从 1 开始。

3.4DataManager Class

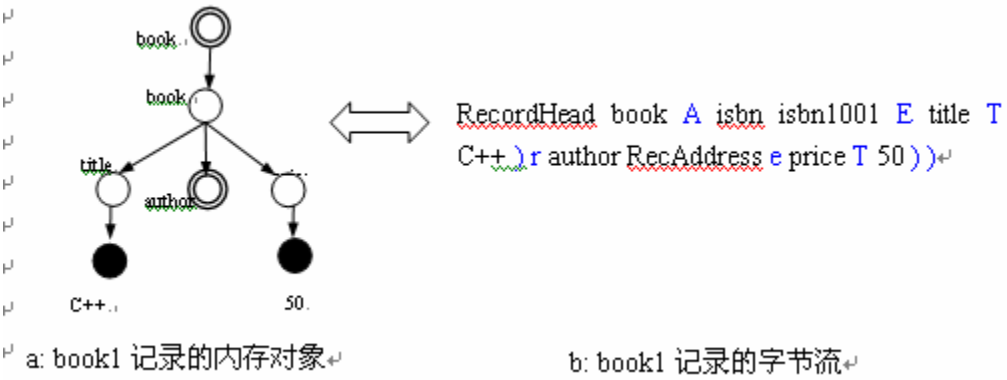
3.4.1 功能

向上层提供管理数据接口，包括：

1) 数据集的建立、删除
2) 数据文件的导入导出
3) 存取查询需要的数据

3.4.2 实现

- 建立数据集时，调用 Schema 类解析指定的模式，分配相应的存储空间，建立 DTDTree 等。
- 导入文档时，调用 ImportHandler 类根据指定的存储模式来划分记录的粒度和确定记录的存储顺序。
- 提供取记录结点、元素结点、属性结点的接口。可以在各种结点之间导航，可以取属性值、元素值和类型信息。NxdbDataManager 类提供的是读记录结点的接口，其它结点的导航和读取通过各自结点的类型来实现。Xml 文档数据在磁盘中是存储字节流的形式，但向上层（查询层）提供的是在 DOM 模型上进行导航的接口，因此存在字节流和内存数据对象之间的转换。
- 要把一个记录存储到磁盘中，需要把该记录的内存对象转化成字节流；要把一个记录从磁盘中读取出来，需要把它从字节流转换成内存对象。OrientStore 使用广义表的方法来快速地进行记录的内存对象和字节流的转换。
- 在 OrientStore 中，E 表示子元素结点，e 表示兄弟元素结点；T 表示子文本结点，t 表示兄弟文本结点；A 表示属性结点；R 表示子记录结点，r 表示兄弟记录结点；)表示结点的结束（属性结点没有结束标志）。以 book1 记录为例，其打成字节流后如图 5(b)所示。其中，分隔符以蓝色表示；RecordHead 表示记录的必要信息，包括记录的地址，父记录的地址，记录的(字节流)长度等；author 后面的 RecAddress 表示 author 记录的地址。



a: book1 记录的内存对象　　b: book1 记录的字节流

-

4. 接口设计

4.1 schema 的接口

见文件” \OrientX\Util\SchemaParser\SchemaParser.h”和对应的 cpp 文件定义实现的类 SchemaParser

DTDTree*　　Generator(char* schemaFile,char* rootName)

函数功能：根据指定的 schema 文档生成 dtdtree

参数说明：char* schemaFile　对应的 schema 文档的名字

　　char* rootName　　该模式中的根元素名字

返回值：如果成功则返回建立的 dtdtree 的指针，否则返回 NULL 指针

备注：这个接口是向 DataManager 提供的，由 DataManager 类的 CreateDataSet（）函数调用。

4.2 import handler 的接口

见文件” OrientX\DataManager\EBDataManager\EBImpHandler.h”和对应的 cpp 文件定义实现的类 EBImpHandler

见文件” OrientX\DataManager\LCRDataManager\LCRImpHandler.h”和对应的 cpp 文件定义实现的类 LCRImpHandler

见文件” OrientX\DataManager\LCRDataManager\CEBImpHandler.h”和对应的 cpp 文件定义实现的类 CEBImpHandler

见文件” OrientX\DataManager\SubTreeDataManager\SubTreeImpHandler.h”和对应的 cpp 文件定义实现的类 SubTreeImpHandler

上述四个类接口一样，只是实现的算法有所不同。

void startDocument();

函数功能：当语法分析器遇到文档开始时激发此事件，主要是为导入文档做一些准备工作，比如将统计值清零，生成虚节点等。

void endDocument();

函数功能：当语法分析器遇到文档结束时激发此事件，主要是保存一些文档信息，为文档中的元素编码，释放不用的页等。

```
void startElement(    const    XMLCh* const      uri,
                      const    XMLCh* const      localname,
                      const    XMLCh* const      qname,
                      const    Attributes&       attributes)
```

函数功能：当语法分析器遇到元素开始时激发此事件，主要是生成相应的新元素节点或记录节点如果有必要，生产新的属性节点，将新生成的节点连接到其父节点下。

```
void endElement( const XMLCh* const uri,
                 const XMLCh* const localname,
                 const XMLCh* const qname)
```

函数功能：当语法分析器遇到元素结束时激发此事件，主要是根据选择的存储方式，为要结束的元素或记录分配存储空间。

void characters(const XMLCh* const chars, const unsigned int length)
函数功能：当语法分析器遇到元素内容时激发此事件，主要是建立相应的textnode，并连接到其对应的元素节点下。

void fatalError(const ParserException& exception);
函数功能：当语法分析器遇到无法恢复的语法错误时激发此事件，抛出检测到的异常，停止parser文档。
备注：这个接口是向 DataManager 提供的，由 DataManager 类的 ImportDoc（）函数调用。

4.3 data manager 的接口

见文件” OrientX\DataManager\nxdbdatamanager.h”和相应的cpp文件定义实现的类NxdbDataManager
int CreateDataSet(char* dataSetName,char* dtdFileName,char* rootName)
函数功能：建立新的数据集，同时建立相应的dtdtree、blocktree，建立相应的oid集。
参数说明：char* dataSetName,指定要建立的数据集的名称
char* dtdFileName,该数据集对应的模式的名称
char* rootName 该模式对应的根元素的名称
返回值：成功返回0，否则返回-1

int DropDataSet(char* dataSetName);
函数功能：删除指定的数据集，同时删除相关的oid集，索引信息等。
参数说明：char* dataSetName指定要删除的数据集的名称
返回值：成功返回0，否则返回-1

int ImportDoc(char* dataSetName,char* docName ,int Type = NX_XML_DATA_TYPE,StorageMode storageMode = DEBMode);
函数功能：在指定的数据集中导入指定的文档，并按照指定的存储格式存储。
参数说明：char* dataSetName, 指定要导入到哪个数据集中。
char* docName ,指定要导入的文档，可以保护路径。
int Type = NX_XML_DATA_TYPE,要导入的事索引文件还是普通的数据文件。默认为数据文件。
StorageMode storageMode = DEBMode选择存储该文档的格式，默认为DEBMode格式。
返回值：成功返回0，否则返回-1

int ExportDoc(char* dataSetName,char* oldDocName,char* newDocName);
函数功能：将指定的数据集中的指定文件到处成XML的格式，并存储在指定位置。
参数说明：char* dataSetName,指定数据集的名字
char* oldDocName,指定要导出的文档名
char* newDocName指定导出的文档的新名字
返回值：成功返回0，否则返回-1

int DropDoc(char* dataSetName,char* docName);
函数功能：删除指定的文档，及相关的索引信息。
参数说明：char* dataSetName,要删除的文档属于的数据集的名字
char* docName要删除的文档的名字
返回值：成功返回0，否则返回-1

RecordNode* GetRecord(int setID,int docId ,int lpNo, short recNo);
函数功能：从指定的地址读取记录
参数说明：int setID,该记录所属的数据集的id
int docId 该记录所属的文档的id
int lpNo，该记录存储的logical page的id
short recNo该记录在page内部的编号
返回值：如果成功则返回新组成的记录的指针，否则返回NULL

void GetRecord(RecordNode* recNode);
函数功能：读取指定的记录，通常recNode是某个子记录在其父记录中的表示，只有oid信息，还没有将其表示的元素内容读进来。
参数说明：RecordNode* recNode要读取其元素内容的记录指针

ElementNode* GetRootElement(char* dataSetName,char* docName);
函数功能：读取指定文档的根元素
参数说明：char* dataSetName,指定的数据集的名称
char* docName指定的文档的名称
返回值：如果成功则返回根元素的指针，否则返回NULL

ElementNode* GetRootElement(int setID, int docID);
函数功能：读取指定文档的根元素，和上一个函数的功能相同。只是一个参数是名称，另一个函数的参数是id。
参数说明：int setID，指定的数据集的id
int docID指定的文档的id
返回值：如果成功则返回根元素的指针，否则返回NULL

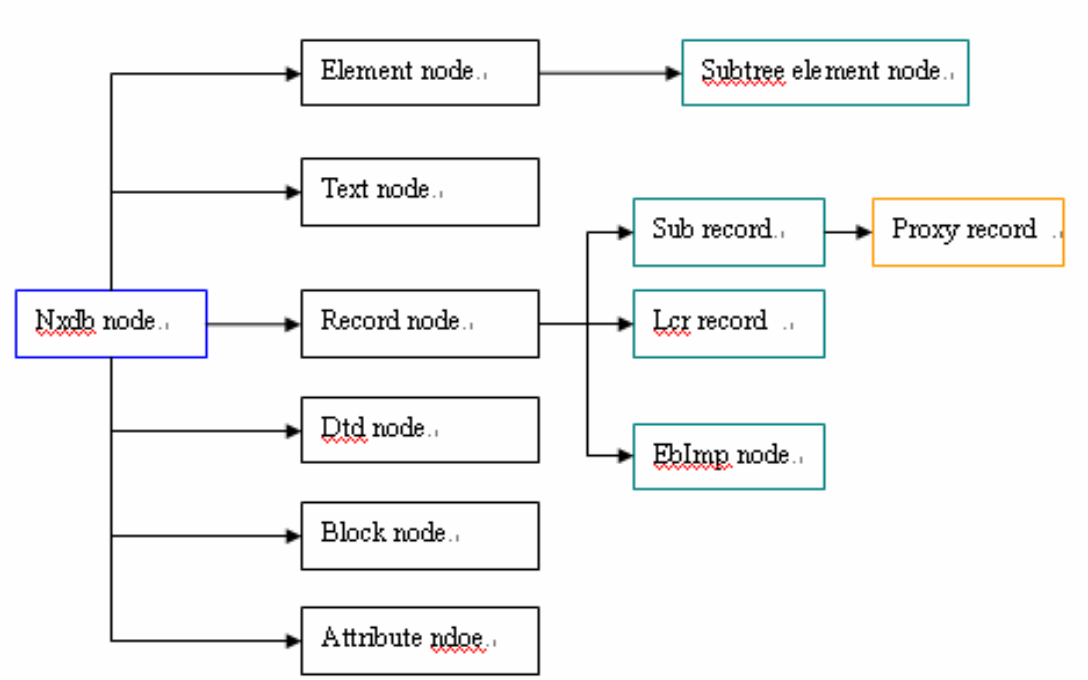
LCRRecNode * GetNextRecord(const LCRRecNode* curRecNode);
函数功能：读取指定的记录的下一个同类记录。
参数说明：const LCRRecNode* curRecNode作为参考的记录的指针
返回值：如果成功则返回下个同类记录的指针，否则返回NULL

LCRRecNode * GetPrevRecord(const LCRRecNode* curRecNode);
函数功能：读取指定的记录的前一个同类记录。
参数说明：const LCRRecNode* curRecNode作为参考的记录的指针
返回值：如果成功则返回下个同类记录的指针，否则返回NULL

5. 数据结构设计

5.1 各种结点类的继承关系

系统中的节点类型有：nxdbnode，elementnode，recordnode，attributenode，dtdnode，blocknode，textnode，subtree element node ，subtree record node，proxyrecnode，lcrrecnode 等。其中 nxdb node 是所有结点的基类，其他都是它的派生类。



5.2 各类结点的设计

名称	nxdbnode
主要功能	向所有的节点提供同一的基类
主要的数据成员	NxdbNode* parent; 指向父节点的指针 NxdbNode* firstChild; 指向第一个子节点的指针 NxdbNode* rightSibling; 指向左兄弟节点的指针 NxdbNode* leftSibling; 指向右兄弟节点的指针 unsigned int oid; 分配的oid unsigned int poid;其父节点的 oid

主要的函数成员	1 virtual short AppendChildAfter(NxdbNode* child, NxdbNode* insertPoint);等添加子节点的函数 2 bool IsDTDNode();等判断节点类型的函数 3 virtual NxdbNode* getRightSibling()等导航函数 4 virtual void Destroy();删除节点 5 virtual NxdbNode* CopyNode() const;等复制函数

名称	Element node
主要功能	对文档中元素的抽象
主要的数据成员	Attr* attr; 属性的指针，如果是多个属性，这是第一个属性的指针 RecAddress address;该元素的存储地址 DTDNodeCode EID; 该元素在模式中的编码
主要的函数成员	1 ElementNode* GetParent(short& flag);等导航函数 2 Attr* GetAttr(DTDNodeCode aid, short& flag); 等存取、更改属性或属性值的函数 3 void* GetTextValue(unsigned int& length);等存取、更改元素内容的函数 4 RegionCode* GetCode();等关于编码的函数

名称	Record node
主要功能	是物理存储的最小单位，element node 必然要属于某个 record node，才能分配存储空间。而 record node 和 element node 之间的对应关系取决于选择的文档的存储格式。如果一个 record node 内有多个 element node，则总有一个 element node 是其 root 成员。其它 element node 肯定都是该 root 的子元素。
主要的数据成员	RecAddress address;该记录的存储地址 DTDNodeCode EID;它的 root 元素在模式中的编码。 RegionCode* pCode 该记录的编码，是整个文档范围内的编码 ElementNode* root 它的 root 元素
主要的函数成员	1 RecordNode* GetLastRefChild(DTDNodeCode& eid);等导航函数 2 virtual unsigned int FormatToStream(char* tmpBuf);等在字节流和节点间的转换函数 3 virtual short storeAgain()因为更新的原因，需要将该 record node 重新存储

名称	Subtree record node
主要功能	继承自 record node，在按照 DSBMode 格式导入文档时生成
主要的数据成员	int endChildNo;它的 root 元素在其父元素中的序号
主要的函数成员	构造、析构函数

名称	Subtree element node
主要功能	继承自 element node，在按照 DSBMode 格式导入文档时生成
主要的数据成员	NxdbNode* curSubRecNode;最后一次被划分出去的 record node 的指针
主要的函数成员	unsigned int GetTotalLen();计算该节点的长度

名称	Proxy record node
主要功能	继承自 subtree record node，它和基类的不同在于，它可能有多个 element node 作为 root 节点，并且这些 element node 之间是兄弟关系
主要的数据成员	int startChildNo; 开始的元素在其父元素中的序号 EidList eidList;它的所有的 root 级的 element 的模式编码的链表
主要的函数成员	构造、析构函数

名称	Lcr record node
主要功能	继承自 Record Node，用于按照 CSB Mode 格式导入文档时
主要的数据成员	unsigned int nextRecOid;同类型的下一个节点的 oid unsigned int prevRecOid;同类型的上一个节点的 oid
主要的函数成员	1 short AppendRecChild(LCRRecNode* node);增加一个子记录 2 virtual short storeAgain();因为元素的更新，需要重新存储该记录，它和 Record Node 中的同名函数重新分配空间的策略不同

名称	Attribute node
主要功能	模式树上元素属性的抽象
主要的数据成员	DTDNodeCode AID;在模式树上的编码 char name[MAX_ELEMENT_NAME_LEN];对应的属性名 DataType attrType;属性值的类型 Attribute * next;属于同一个 dtdnode 的下一个属性 int valueIndexAddr;索引的控制页的地址
主要的函数成员	void setAttrType(char* str);设置属性值的类型

名称	Attr node
主要功能	文档中元素属性的抽象，它不同于 Attribute Node，因为它有具体的值，而不用存储名字
主要的数据成员	void* m_Value;属性值的指针 DataType m_Type;属性值的类型 unsigned short m_Len;属性值的长度 DTDNodeCode AID;属性在模式树上的编码 Attr* m_Next;同一个元素中的下一个属性的指针
主要的函数成员	1 void* GetValue();等取值、设定值的函数 2 unsigned int formatFromStream(char* tmpBuf,unsigned int& offset);将该属性节点转换成字节流

名称	Text node
主要功能	文档中元素内容的抽象
主要的数据成员	enum DataType m_DataType;该节点的值得类型 void* m_Value;节点的值得指针 unsigned int m_Length;节点值得长度
主要的函数成员	1 int setValue(const void* value,const enum DataType& DataType);等设定、取值的函数 2 unsigned int format2Stream(char* tmpBuf,unsigned int& offset);等用于节点和字节流之间的转换

名称	Dtd node

主要功能	模式树上节点的抽象
主要的数据成员	DTDNodeCode EID;模式节点的编码 char name[MAX_ELEMENT_NAME_LEN];节点的名称 DataType dataType;节点的类型 Attribute* attrs;节点属性的指针，如果有多个属性，这是第一个属性的指针，属性间用指针相连 CodeList m_AidList;引用该节点的其它节点的属性编码的列表 int valueIndexAddr;值索引的控制页的地址 IxRecordSet pathIndexAddr;路径索引的控制页的地址
主要的函数成员	1 short AppendAttr(Attribute *attr);增加属性节点 2 unsigned int FormatToStream(char* tmpBuf);转换成字节流的形式 3 DTDNode* GetChild(char* childName);等导航函数 4 bool IsBlock();等节点类型的判断函数 5 void SetCardinality(char cardinality);节点属性的设置函数

名称	Block node
主要功能	这种节点用于 CSB Mode 和 CEB Mode 中，它是特殊的模式节点，这种节点在某个文档中的所有实例元素聚集存放。
主要的数据成员	DTDNodeCode EID;模式上的编码 short bufNo;该模式节点占用的缓冲页的编号 unsigned int startOid;该节点对应的第一个实例的 oid unsigned int endOid; 该节点对应的最后一个实例的 oid LCRRecNode* lastNode;最后一个实例，插入新节点时用
主要的函数成员	1 BlockNode* GetChild(DTDNodeCode eid);等导航函数 2 short FormatToStream(char* tmpBuf,bool isAbbreviate);等用于节点和字节流之间的转换

名称	EbImp node
主要功能	继承自 Record Node，用于导入 DEB Mode 的文档，比父类多属性成员
主要的数据成员	Attr* attr;该元素对应的属性指针
主要的函数成员	1 short AppendAttr(Attr* a);增加属性 2 unsigned int FormatToStream(char* tmpBuf);转换成字节流

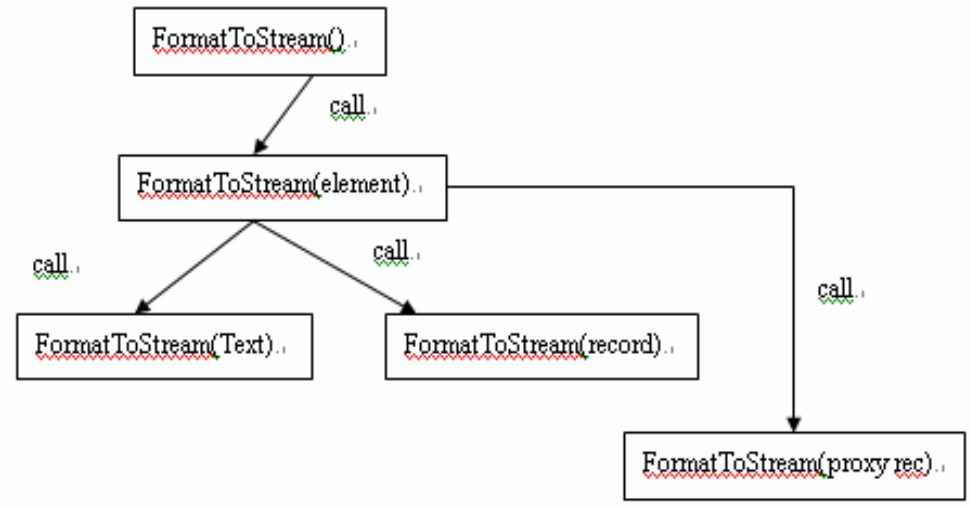
	3 void DeleteLocal();删除它的属性和元素内容

5.3 部分函数的算法

5.3.1 各种节点和字节流之间的转换

Record Node:: FormatToStream(char* tmpBuf)
算法：将该 record 节点转换成字节流并存储在制定的参数中，先存储该记录的头信息，包括记录的 root 元素的 oid，poid，记录得类型；然后存别转换并存储所有的字节点的信息，遇到该 record node 内不同类型的节点时，先存储一个表示节点类型的标志信息，然后分别调用下面相应的函数；最后存储该记录被转换成字节流后的长度。为了节省空间，element node 和 record node 的模式编码之存储 start 部分，而不同的模式编码的 start 部分是唯一的。在从字节流转换回来的时候根据这个标志信息，调用不同的转换函数。

FormatToStream(TextNode* node,char* tmpBuf,unsigned int& offset)
FormatToStream(ElementNode* node,char* tmpBuf,unsigned int& offset)
算法：依次存储该元素的 eid，存储它的属性信息，存储它的子节点
FormatToStream(RecordNode* node, char* tmpBuf, unsigned int& offset)
算法：这里存储的 record node 只是某个子记录的 oid 信息
FormatToStream(ProxyRecNode* node, char* tmpBuf, unsigned int& offset)
这些函数间的调用关系是：



Record Node:: FormatFromStream(char* tmpBuf,unsigned int length)
算法：将指定的字节流转换成节点，根据当前的指示符，分别调用下面的相关函

数。同一个 record node 里不同类型的节点在存储时其前都有一个字符指示它的类型。各函数间的调用关系类似于上图。

FormatFromStream(Attr* a, char* tmpBuf,unsigned int& offset)
FormatFromStream(TextNode* node, char* tmpBuf,unsigned int& offset)
FormatFromStream(ElementNode* node,char* tmpBuf,unsigned int& offset)
FormatFromStream(RecordNode* node,char* tmpBuf,unsigned int& offset)
FormatFromStream(ProxyRecNode* node,char* tmpBuf,unsigned int& offset)

5.3.2 record node 的重新存储

- **RecordNode::storeAgain() 算法**
  1) 计算该 record node 转换成字节流后的大小，以此值为参数分配内存空间，用来存储转换成的字节流，转换成字节流，
  2) 删除旧的记录，
  3) 重新存储新的字节流，如果原来的页里还有足够的空间，则存储在原来的页里，否则分配新的存储空间，添加到文档末尾，
  4) 对于上面提到的第二种情况还要修改 oid 集内的相应信息。
- **LCRRecordNode::storeAgain() 算法**
  ✧ 与上一个函数的不同主要体现在存储空间的分配上
  1) 计算转换成字节流后的大小，当然 lcr record 和 record node 的记录头的长度也不一样，转换成字节流，
  2) 删除旧的记录，
  3) 重新存储新的字节流，如果原来的页里还有足够的空间，则存储在原来的页里，否则在相应的 block node 中寻找空间，或者为此 block node 分配新的空间，
  4) 对于上面提到的第二种情况还要修改 oid 集内的相应信息。

6. 出错处理

主要是导入文档时的出错处理：

如果导入的文档不符合指定的 schema 格式，或者有无法恢复的语法错误，则报错，抛出异常，并停止导入文档的操作。

- 用到的类 ParserException 继承自 Apache sax2 的 SAXParseException
  功能：使用者可以在需要的地方用其来生成异常
- 用到的类：ParserError 继承自 Apache sax2 的 ErrorHandler
  用到的函数 ParserError:: fatalError(const SAXParseException& exception)
  如果遇到了语法问题，生成错误，并抛出，文法分析器会捕捉到抛出的任何异常。
- 用到的类各导入类
  用到的函数 fatalError(const ParserException& exception)
  使用者可以在需要的地方生成错误，并抛出，文法分析器会捕捉到抛出的任何异常。