SchemaManager 概要设计说明书

负责人：罗道锋，安靖
编写人：罗道锋
系统版本号：OrientX Version 1.5
完成时间：2004/02/20
开发单位：中国人民大学 IDKE 实验室 XML 工作组

1. 引言

本说明书详细说明了OrientX1.5中SchemaManager的设计和实现。SchemaManager主要是对Dadaset的Schema的管理。Schema记录了本数据集的文档的结构信息。

2. 概述

SchemaManager管理的对象是数据集的Schema。Schema类似与关系数据库中的数据字典。数据字典说明了数据库中的表结构、索引结构等信息。在OrientX中，一个数据集对应于一个Schema，每一个数据集由多个符合同样Shema定义的文档组成。Schema有两大作用：首先，Schema类似于DTD或者XMLSchema的作用，说明了文档的结构特征。其次，Schema还记录了关于数据集的索引、引用和被引用关系等。

SchemaManager在查询中的作用是：当一个查询提交时，首先用Schema检查该查询是否合法，比如路径是否存在，数据类型是否匹配等；如果合法，则根据Schema把查询中的TagName转换成结点名内部表示EID。

3. 总体设计

SchemaManager有五个数据结构类（Attribute, DTDNode、DTDTree，BlockNode，BlockTree）和一个管理类（MetaDataManager）。

每一个DTDTree上的结点都有一个唯一的ID。DTDNodeID使用(start, end)编码，Attribute结点的编码用<start, attrNo>，其中，start与它所属的DTDNode的start相同，attrNo从-1, -2，表示第几个attribute.

Attribute类和DTDNode类分别对应于DTD中的Attribute结点和DTD结点。DTDTree则代表由DTDNode和Attribute组成的整个Schema。

BlockNode和BlockTree是实现聚簇类存储方法(CSB, CEB)方法时需要的一个数据结构。聚簇类存储方法把DTDTree分解成若干语义块(Block)，根据语义块把文档分解成实例块(instance block)。符合某一个语义块的所有实例块聚簇存储。BlockNode表示一个语义块，记录了符合该语义块的所有实例块的信息，比如这些实例块占用的物理块数等。BlockTree则是BlockNode组成的树。每一个用CSB或者CEB存储的文档都有一个BlockTree。

MetaDataManager类即SchemaManager，由于历史原因，名字与SchemaManager不符。MetaDataManager负责DTDTree和BlockTree的保存（到磁盘）和（从磁盘）读取。

4. 接口设计

SchemaManager模块的接口由五个数据结构类（Attribute, DTDNode、DTDTree，BlockNode，BlockTree）和一个管理类（MetaDataManager）提供。数据结构类提供了访问改数据结构的结构，管理类提供了导入、导出DTDTree和BlockTree的结构。

5. 数据结构设计

5．1 逻辑结构设计

5．1．1 Attribute 类

Attribute类代表DTDTree上的属性结点。注意，它区别于Data Tree上的属性结点(Attr类)。Attr结点是Attribute结点的实例。Attribute类说明了属性的名字，ID，数据类型，是否有索引，是否必须等。它的主要数据成员和主要成员函数如下：

Attribute 类的数据成员	
数据成员定义	说明
DTDNodeCode AID;	属性的 ID
char name[MAX_ELEMENT_NAME_LEN];	属性名
DataType attrType;	属性的数据类型
char m_Required;	是否必须的。’Y’表示必须，’N’表示不是必须
Attribute * next;	下一个属性节点
int valueIndexAddr;	值索引的根物理块的 LpNo. 如果 valueIndexAddr==-1，表示没有值索引

Attribute 类的成员函数	
函数定义	功能说明
Attribute* CopyAttrbute();	拷贝以本 Attrributer 为链头的 Attribute 链
bool HasValueIndex();	检查本 ValueIndex 是否有值索引
void setAttrType(char* str);	设置数据类型

5．1．2 DTDNode 类

DTDNode类代表DTDTree上的DTD结点。ElementNode结点是DTDNode结点的实例。DTDNode类说明了属性的名字，ID，数据类型，是否有索引，是否有递归环等。它的主要数据成员和主要成员函数如下：

DTDNode 类的数据成员	
数据成员定义	说明
DTDNodeCode EID;	结点的 ID
char name[MAX_ELEMENT_NAME_LEN];	结点名
DataType attrType;	结点的数据类型
Attribute* attrs;	本 DTDNode 的属性列表
CodeList m_AidList;	引用这个结点的属性链表
int valueIndexAddr;	值索引的根物理块的 LpNo. 如果 valueIndexAddr==-1，表示没有值索引
IxRecordSet pathIndexAddr;	路径索引的地址。路径索引保存了该结点
char properties;	该 DTDNode 的结点特征
double updateWeight;	该 DTDNode 的实例结点的更新权重
与递归处理相关的数据成员	
char circleExitCount;	本结点是有多少个递归环的出口
char* circleExitTarget;	递归环的出口的通向的目标
char circleEntrCount;	本结点是有多少个递归环的入口
char* circleEntrSource;	递归环的入口的来源
与 Schema Parser 相关的数据成员	
enum ChildOrder childOrder;	孩子结点的顺序

DTDNode 类的成员函数	
导航式函数	
函数定义	说明
Attribute* GetAttribute(DTDNodeCode& AID);	得到指定 ID 的属性
Attribute* GetAttribute(char* attName);	得到指定属性名的属性
DTDNode* GetChild(DTDNodeCode& eid);	得到指定 ID 的孩子结点
DTDNode* GetChild(char* childName);	得到指定结点名的孩子结点
DTDNode* GetRightSibling(DTDNodeCode& eid);	得到第一个指定 ID 的右兄弟结点（不一定是直接右兄弟结点）
DTDNodeListItem* GetDescendant(const char* name);	得到指定名字的子孙结点的列表
DTDNode* GetDescendant(DTDNodeCode& nodeID);	得到按照前序遍历顺序第一个指定 ID 的子孙结点
DTDNode* GetDescendant(int index);	得到按照前序遍历顺序第 index 个子孙结点（如果 index=0，返回本结点）
DTDNode* GetAncestor(DTDNodeCode parentEID);	得到指定 ID 的父亲结点。由于有递归环的存在，一个结点可能有若干个父亲结点
DTDNode* GetAncestor(int index);	得到第 index 个祖先结点，如果 index=0，返回父亲结点；如果 index=1，返回父亲的父亲结点
获取和设置数据成员的函数	
char GetCardinality();	得到本结点的修饰符。修饰符可以是"+","*","?"等。
bool IsBlock();	本结点是否是语义块的根结点 (CEB 或者 CSB 存储方法要用到)
bool IsMixed();	本结点是否是混合结点（可以有 Text 子结点和 Element 子结点）
bool IsDescr();	本结点是否是描述性结点（文本段特别长的结点）
bool IsRepeatable();	本结点是否是可重复的结点（修饰符带）
bool IsTextChildAllowed();	本结点是否允许有 Text 子结点
bool HasValueIndex();	本结点是否有值索引
void SetName(char* nodeName);	设置本结点的结点名
void SetCardinality(char cardinality);	设置本结点的修饰符
void SetBlock(bool isBlock);	设置本结点是否语义块的根结点
void SetMixed(bool isMixed);	设置本结点是否是混合型的结点
void SetDescr(bool isDescr);	设置本结点是否是描述性结点
void SetDataType(char* str);	设置本结点的数据类型
DTDNodeCode getEid();	得到本结点的 ID
char* getTagName();	得到本结点的结点名
DataType getDataType();	得到本结点的数据类型

char* GetDataType();	得到本结点的数据类型
其它函数	
short AppendAttr(Attribute *attr);	插入一个属性结点（插在链尾）
DTDNode* CopyDTDNode();	拷贝以本结点为根的子树
double GetUpdateWeight();	得到本结点的更新权重
bool childHasExist(DTDNode* child);	子结点是否已经存在
void setChildCardi(char cardinality);	设置孩子结点的属性
与递归处理相关的函数	
bool IsInCircle();	本结点是否在递归环内
short GetCirCleExitCount();	本结点是有多少个递归环的出口
short GetCircleEntrCount();	本结点是有多少个递归环的入口
short GetCircleExitTarget(short index);	得到第 index 个出口目标
short GetCircleEntrSource(short index);	得到第 index 个入口目标
char* GetCircleExitTarget();	
char* GetCircleEntrSource();	
void SetInCircle(bool isInCircle);	
void SetCircleExitCount(short exitCount);	
void SetCircleEntrCount(short entrCount);	
short SetCircleExitTarget(short index,short target);	
short SetCircleEntrSource(short index,short target);	
void SetCircleExitTarget(char* exitTarget);	
void SetCircleEntrSource(char* EntrSource);	
与导入导出相关的函数	
unsigned int FormatToStream(char* tmpBuf);	把以本结点为根的 DTD 树打成字符流
unsigned int FormatFromStream(unsigned int Length,short& nCount)	从字符流组装成一棵以本结点为根的子树
unsigned int FormatToStream(Attribute* attr, char* tmpBuf, unsigned int& offset);	把一个属性结点打成字符流
unsigned int FormatToStream(DTDNode* root, char* tmpBuf,unsigned int& offset);	把一个 DTD 结点打成字符流
unsigned int FormatFromStream(Attribute* attr, char* tmpBuf, unsigned int& offset);	组装一个属性结点
unsigned int FormatFromStream(DTDNode* root, char* tmpBuf,unsigned int& offset);	组装一个 DTD 结点

5．1．3 DTDTree 类

DTDTree类是Schema Tree的抽象。它以一个DTDNode为根。DTDTree上记录了该Schema的实例文档数目、占用的物理块的数目等信息。同时，DTDTree还提供从TagName到EID和EID到TagName的双向映射。在导入导出文档和进行查询时，DTDTree作为数据集的Schema，将会被频繁使用。

每一个数据集又且只有一个DTDTreee。

DTDTree类的数据成员	
数据成员定义	说明
DTDNode* root;	DTDTree的根结点
char DTDName[MAX_DOC_NAME_LEN];	DTDTree的名字
int SetID;	DTDTree所在的数据集ID
int docCount;	本数据集包含的文档的数目。所有文档均符合本DTD定义
int totalPage;	本数据集占用的物理块数目
CharEncoding chEncoding;	本数据集文档使用的编码方式
char hasPathIndex;	是否有路径索引
char indexChanged;	索引（路径索引或者值索引）是否被更改过
建立TagName--EID映射关系所需的数据成员	
ListSet* hash[HASHSIZE];	Hash桶
short nodeCount;	本DTDTree的结点数目
bool hashIsNULL;	Hash桶是否为空
DTDNode* DTDNodeArray[MaxDTDNodeCount];	保存了本DTDTree所有DTDNode的指针。DTDNode的EID的start就是它在此数组中的下标

DTDTree类的成员函数	
函数定义	说明
DTDNodeListItem* GetDTDNodeList(char* eleName);	得到指定结点名的DTDNode链表
DTDNode* GetNode(DTDNodeCode eid);	得到指定ID的结点
DTDNode* getNode(unsigned short eidStart);	得到指定ID的start值的结点
int getEid(unsigned short eidStart,DTDNodeCode& eid);	得到指定start值的ID
Attribute* GetAttribute(DTDNodeCode& aid);	得到指定aid的属性结点
unsigned int FormatToStream(char* tmpBuf);	打成字符流
unsigned int FormatFromStream(char* tmpBuf,unsigned int Length);	从字符流组装DTDTree
unsigned int CountStreamLen();	计算打成字符流之后的长度
bool HasPathIndex();	是否有路径索引
bool IndexChanged();	索引（路径索引或者值索引）是否被更改过
void createRefAid(const AidIndexList list);	建立IDREF-REF映射关系
建立TagName--EID映射关系所需的成员函数	
void setupMap();	建立映射关系
void deleteMap();	删除映射关系
void SetupDTDNodeArray();	建立DTDNode数组

5．1．4 BlockNode 类

BlockNode类是Semantic Block的抽象。在聚簇存储方法中，需要把Schema划分成语义块，每个语义块的实例子树聚簇存储。BlockNode就是语义块的抽象，它记录了实例子树占用的物理块，第一个和最后一个物理块等。

BlockNode类的数据成员	
数据成员定义	说明
DTDNodeCode EID;	BlockNode的EID
short bufNo;	当前实例所在的物理块所在的缓冲区号
int tmpLpNo;	临时物理块号
int totalPages;	实例子树占用的所有物理块数
unsigned int startOid;	第一个实例子树的OID
unsigned int endOid;	最后一个实例子树的OID
int endLpNo;	实例子树占用的最后一个物理块号
bool firstNode;	是否是第一个结点
bool endChanged;	结点的endOID是否改变过
LCRRecNode* lastNode;	最后一个实例子树

BlockNode类的成员函数	
函数定义	说明
int GetStartOid();	得到开始OID
int GetEndOid();	得到结束OID
int GetTotalPages();	得到占用的物理块号
BlockNode* GetDescendant(DTDNodeCode eid);	得到指定ID的子孙Block结点
BlockNode* GetChild(DTDNodeCode eid);	得到指定ID的孩子结点
BlockNode* GetAncestor(DTDNodeCode eid);	得到指定ID的祖先结点
BlockNode* CopyBlockNode();	拷贝以本BlockNode为根的Block子树
short FormatToStream(char* tmpBuf,bool isAbbreviate);	以减缩或不减缩的方式打成字符流
short FormatFromStream(char* tmpBuf,bool isAbbreviate,short Length);	以减缩或不减缩的方式从字符流组装成BlockNode
short FormatToStreamNotAbbr(BlockNode* node,char* tmpBuf,short& offset);	以不缩减的方式打成字符流
short FormatToStreamAbbr(BlockNode* node,char* tmpBuf,short& offset);	以缩减的方式打成字符流
short FormatFromStreamNotAbbr(BlockNode* node,char* tmpBuf, short& offset);	以不缩减的方式从字符流组装成BlockNode
short FormatFromStreamAbbr(BlockNode* node,char* tmpBuf, short& offset);	以缩减的方式从字符流组装成BlockNode

5．1．5 BlockTree 类

BlockTree类代表语义块组成的树。它提供了查找BlockNode，从DTDTree生成BlockTree等方法。

每一个聚簇方法存储的数据集有一个总的BlockTree，该数据集中的每一个文档有一个BlockTree。我们把前者叫作Dataset BlockTree，后者叫做 Document BlockTree。

BlockTree类的数据成员	
数据成员定义	说明
BlockNode* root;	根结点
BlockNode* curBlockNode;	当前Block结点，在遍历树的处理中需要用到
int streamLen;	打成字符流之后的长度
int lastBlockLpNo;	实例子树最后使用的一个物理块号
short nodeCount;	BlockTree的结点数目
RecAddress address;	BlockTree打成字符流之后的存储地址

BlockTree类的成员函数	
函数定义	说明
void Generator(DTDTree *dtdTree, StorageMode mode);	从DTDTree生成BlockTree
BlockNode* GetBlockNode(DTDNodeCode eid);	得到指定id的BlockNode
BlockTree* CopyTreeStruc();	拷贝树结构
short FormatToStream(char* tmpBuf,bool isAbbreviate);	以减缩或不减缩的方式打成字符流
short FormatFromStream(char* tmpBuf,bool isAbbreviate,short length);	以减缩或不减缩的方式从字符流组装成BlockNode

5．1．6 MetaDataManager 类

MetaDataManager负责管理DTDTree和BlockTree，比如它们的导入导出等。该类的所有方法都是静态的。另外，系统还有一个DTDTreeCache，用来缓冲读进内存的DTDTree。

MetaDataManager类的成员函数	
函数定义	说明
static void FormatDTDPage(char* DataSetName);	格式化DTD页。DTDTree存在每个数据集的第0个物理块
static DTDTree* DTDLoader(char* DataSetName);	导入指定数据集的DTDTree
static DTDTree* DTDLoader(int setID);	导入指定数据集的DTDTree
static void DTDFlusher(DTDTree* dtdTree);	把DTDTree导出到磁盘
static DTDNode* GetDTDNode(int setID,DTDNodeCode& eid);	得到指定数据集的指定ID的DTD结点
static void FreeDTDTree(int setID);	释放DTDTree，但不刷出到磁盘
static void getEid(int setId,short eidStart,DTDNodeCode& eid);	得到指定数据集的指定start值的DTDNodeID
static BlockTree* DataSetBlockTreeLoader(char* DataSetName,StorageMode mode);	导入指定数据集的DataSetBlockTree
static BlockTree* DataSetBlockTreeLoader(int setId,StorageMode mode);	导入指定数据集的DataSetBlockTree
static void DataSetBlockTreeFlusher(BlockTree* blockTree,StorageMode mode);	导出指定数据集的DataSetBlockTree
static BlockTree* BlockTreeLoader(char* DataSetName,char* docName,StorageMode mode);	导入指定数据集的指定文档的DocumentBlockTree
static BlockTree* BlockTreeLoader(int setId,int docId,StorageMode mode);	导入指定数据集的指定文档的DocumentBlockTree
static void BlockTreeFlusher(BlockTree* blockTree,int& lastBlockLpNo);	导出指定数据集的指定文档的DocumentBlockTree
static int getSubEleEid(int setID, DTDNodeCode& srcEid, char* EleName, DTDNodeCode& destEid);	得到指定数据集的指定DTDNode下的指定TagName的儿子结点
static int getAttrAid(int setID,DTDNodeCode& srcEid, char* attrName, DTDNodeCode& destAid);	得到指定数据集的指定DTDNode下的指定AttrName的属性结点
static int getDescendantEleEid(int setID, DTDNodeCode& srcEid,char* EleName, vector<DTDNodeCode>& destEidList);	得到指定数据集的指定DTDNode下的指定TagName的子孙结点列表

5．2 物理结构设计

5.2.1 DTD中递归定义的结点的表示方法

DTDTree中常常有递归定义的结点，比如R(X), X(Y) Y(Z), Z(X). 我们把 (X, Y, Z) 叫做一个递归环，X，Y，Z都在环中。X是环的入口，入口的来源(EntrSource)是Z；Z是环的出口，出口指向的目标(ExitTarget)是X。

我们需要标示<出口，入口>对。对于每一个入口，我们用circleEntrSource来表示它对应的出口，表示对应的出口是前序遍历ircleEntrSource个子孙结点。比如，如果circleEntrSource＝3，表示它对应的出口是前序遍历第3个子孙结点；类似地，我们用circleExitTarget来表示每一个出口指向的目标（它对应的入口），表示它对应的入口是第circleExitTarget个祖先结点。比如，如果circleExitTarget＝3，表示该结点对应的入口是它的第三个祖先结点（从它自己算起）。

由于每一个DTDNode可能同时在一个以上的环内，而且可能是多个环的入口或出口，在实现中，我们用数组了表示<出口，入口>对。

5.2.2 DTDTree建立TagName到ID的映射关系的方法

DTDTree中经常需要用ID得到对应的TagName的名字，或者用TagName得到所有结点名为TagName的结点的ID。为了加速这种操作，我们建立了从TagName到ID的双向映射关系。

由于ID使用<start, end>编码，我们直接用一个数组：

DTDNode* DTDNodeArray[MaxDTDNodeCount];

来表示从ID到TagName的映射。ID中的start值对应于DTDNodeArray中的下标。

而从TagName到ID的映射则比较麻烦。我们使用Hash查找的方法，为TagName建立Hash桶。每一个Hash桶是一个ListSet，而每一个ListSet是DTDNodeListItem组成。

```
struct DTDNodeListItem
{
    DTDNode* node;
    DTDNodeListItem *next;
};

struct ListSet
{
    DTDNodeListItem* nodeList;
    ListSet* next;
};

ListSet* hash[HASHSIZE];
```

给定一个TagName，我们首先通过Hash函数计算它在哪个桶里，然后找到节点名是TagName的链表，返回。

6. 出错处理