陆世潮于2004/3/5注：这是吴岑同学于2002/5/19写的存储管理模块的概要设计。后来OrientX系统又作了部分的改动；但为了忠于原作者，本说明书未作任何修改。

一．功能概述

（1）提供一个类标准数据库的基本操作集：对数据集的建立和删除，对数据集内 XML 文档的建立和删除；

（2）向上层模块提供一个物理页面为单位进行页面申请和读写操作的接口，管理系统物理存储空间的分配与回收；

（3）使上层尽量独立于本部分的具体实现方式，提供统一的接口；

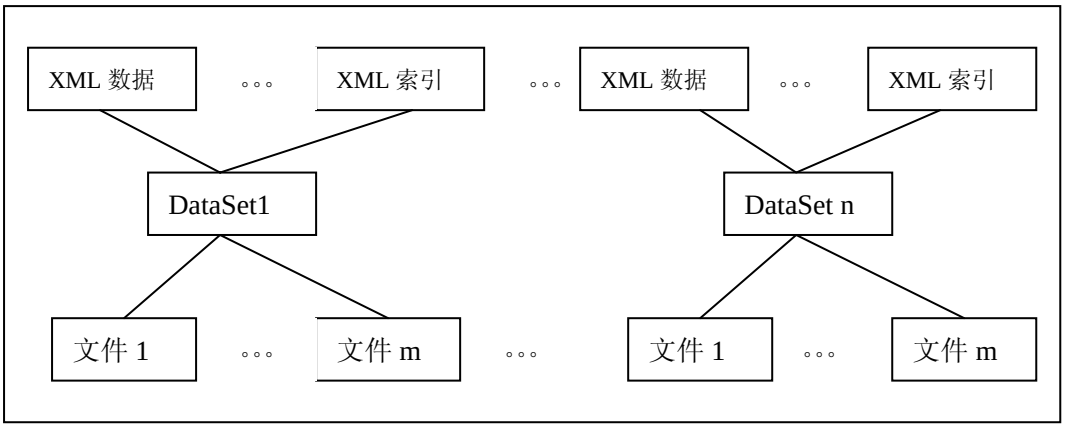
（4）为所有的涉及磁盘文件读写的部分提供独立于操作系统的接口，以便未来的扩充实现。

二．设计概述

1．基本概念

结构模型

整个数据库可由不同的 DataSet 构成，最多可有 65536 个 DataSet。在每一个 DataSet 中，可有多个 XML Doc（包括数据和索引）。不同的 DataSet 的 Schema 是不同的，但在同一个 DataSet 中，所有的 XML Doc 的 DTD 或 Schema 必须相同。在数据库中，实际物理存储的数据页是依靠 DataSet 的 SetId 和逻辑页号 lpno(int)这一序列来唯一标识。初步设定每一个逻辑页的容量为 4K，从而每个 DataSet 的最大容量为 16，384G，而整个数据库的最大容量为 1，048，576T。



2．类设计说明

（1）AccessManager：

作为上层（DataManager、IndexManager、DtdMataDataManager）访问的接口，以统一格式外包 StroageManager、BufferManager 的所有服务。

（2）StroageManager：

提供一个类标准数据库的基本操作集：对 DataSet 的建立和删除，对 DataSet 内 XML 文档的建立和删除；向上层模块提供一个物理页面为单位进行页面申请和读写操作的接口，管理系统物理存储空间的分配与回收。

①DataSetManager

管理各个 DataSet 的创建，打开和删除

②DocManager

用于管理各 DataSet 内 XML Doc 的创建，打开和删除

③FreePageManager

提供一个物理页面为单位进行页面申请和读写操作的接口，管理系统物理存储空间的分配与回收

④LongFileManager

实现对 XML 文档内过大的 Text 数据的读写和删除

（3）FileManager

为所有的涉及磁盘文件创建、打开、删除和读写的部分提供独立于操作系统的接口

3. 核心数据说明

（涉及整个数据库的，使用.nxdb 后缀；涉及一个 DataSet 控制的，使用.ctl 后缀；涉及具体文件存储的，使用.dat 后缀）

（1）DataSet 控制文件 DataSetManageFile.nxdb：

记录数据库中的 DataSet 数量以及所有 DataSet 的信息。

① 格式：

现有 DataSet 数量|DataSet 空闲标志|DataSet 1 名称|DataSet 2 名称|。。。|DataSet n 名称

② 说明：

现有 DataSet 库数量：2Byte

DataSet 空闲标志：全部 DataSet 的状况信息（8KB）。一个 bit 代表一个 DataSet。如果为 1，则表示 DataSet 存在，为 0，表示可创建该 SetId 的 DataSet。将某 DataSet 删除，则该位置的标记置 0。再建库时将使用最小的 SetId 来创建 DataSet。

DataSet 名称：每个 DataSet 名称的大小固定，为 NX_DATASET_NAME_SIZE（初定大小为 256Byte）。

（2）Doc 控制文件：

对应每个独立的 DataSet，有一个专属文件，用于存放本 DataSet 所属的 XML 文档集合的相应控制信息。命名规则：根据取得的 setId 号，做添加，比如 SetId=5，则该文件名为 Set00005.ctl。其内容如下：

① 格式：

现有文档数量|文档 1 名称|类型|首地址|文档 2 名称|类型|首地址|。。。|文档 n 名称|类型|首地址

② 说明：

现有文档数量：2Byte。所有的 Data 和 Index 文档总数

文档名称：文档大小固定，为 NX_DOC_NAME_SIZE（初定大小为 256Byte）

类型：NX_DATATYPE 或 NX_INDEXTYPE

首地址：文档的根节点地址（6Byte），组成：lpno（4Byte），RecordNo（2Byte）。根据这两个序号能够唯一标志出其在实际文件中存贮的位置。其中 RecordNo 可以为无用信息，具体须看实际的对物理页面的分配计划。

（3）页面存储管理文件：

对 DataSet 内数据存储空间的管理。命名规则：根据取得的 setId 号，做添加，比如 SetId=5，则该文件名为 FreePage00005.ctl。

① 格式：

现有文件数|页面空闲标志

② 说明：

现有文件数：FileNum（4Byte）。当为初始时，现有块数为 1；以后每增加一次就增 1。

页面空闲标志：2bit。00 表示空闲，01 表示内有数据但未满，11 表示已满。lpno 使用 int，所以共可以有 2 的 32 次方个逻辑页面。该文件初始大小为 NX_FREEPAGE_FILE_SIZE（初定为 32M/4k/4 =2048，这样大小一个 Block 就代表了一个文件的存储空间），以后每次都增加 NX_FREEPAGE_FILE_SIZE。

（4）数据文件：

所有的数据实际都存储于数据文件中。每个数据文件等长，初步定为 32M。根据取得的根据取得的 setId 号和 lpno 号，做添加，比如 SetId=5，而 lpno=0，则 FileNo= lpno/ NX_FREEPAGE_PER_BLOCK +1= 1；则该文件名为 Set00005Data000001.dat。

（5）Long 文件。

用来存储 XML 文档中过大的 Text 段数据。该文件与每一个 DataSet 一一对应（如 SetId=5，则该文件名为 LongDataFile00005）。文件结构如下：

总段数（4Byte）|Text1 长度(4Byte)| Text1 内容|。。。| Text n 长度(4Byte)| Text n 内容

4. 主要处理策略说明

（1）外存空间的分配、扩充和回收策略

整个外存空间由各个 DataSet 的外存空间共同构成。通过创建 DataSet 和往 DataSet 中添加新的 XML 文档的方法来实现外存空间的扩充。

①当需要新的外存空间时，则创建新的数据文件。首先在库控制文件中寻找头一个空闲的、可用来映射的逻辑页面段，然后创建相对应的文件。（假设在 SetId 为 5 的 DataSet 内，头一个空闲的完整逻辑页面段为[5*16*1024，6*16*1024]，即段号为 6，说明数据文件 Set00005Data000006 未创建，可用，则创建之，并把相应的逻辑段标为已用，并将之加入相应的数据库）。

②当在某一数据集上申请空间时（加入新的文档或是对已有的文档进行扩充），使用首次适配法。先顺序扫描页面空闲文件，找到第一个可用的页面（未满或是空闲）后，根据其页面号可找到对应的数据文件，读入其文件号可找到其对应的页面内分配情况。在文件内也采用首次适配法，找到满足条件的所需空间后放入。如果该页面因为此操作而没有空闲空间，则将该页面标记为满。如果该页面所剩的空间不足，则寻找下一个可用的页面（每一次写入的单个 Record 不跨页面）。如果最后没有找到可用空间，则添加新的数据文件到该库。

③外存空间通过删除数据集的方式进行回收，释放了的空间所对应的逻辑段都标记为空闲。如果对应的文件已空，可删除文件，修改相应的控制信息。也可删除某个库，则该库所属的文件全部被删除，以释放空间。

（2）数据文件打开策略

所有对文件的操作是通过文件 fd 进行的，在一次的操作中，可能要对同一数据文件多次打开，而每次打开都需要获取该文件记录在外存的信息，因此在打开文件表中记录 fd 与文件的对应就可减少真正打开的次数，从而提高系统效率。因此在打开文件前先检查打开文件表,如有对应 fd,直接返回，否则才真正打开文件。

（3）Long 文件

所有大于一个 Page 的 Text 段都存储在特定 DataSet 对应的 Long 文件中。在实际的 record

中，将会有相应的在 Long 文件中的 RecordId。

三．接口定义

1．DataSetManager

（1）int CreateDataSet(char* DataSetName);

创建名为 DataSetName 的 DataSet

（2）int DeleteDataSet(char* DataSetName);

删除名为 DataSetName 的 DataSet

（3）int OpenDataSet(char* DataSetName);

打开名为 DataSetName 的 DataSet，返回其 SetId

（4）int LookAllDataSet();

察看数据库中所有 DataSet 的信息（名称）

（5）int FindFreePos();

范围当前可用、空闲的最小 SetId（返回值+1）

（**说明：由于 SetId、DocId 都从 1 开始基数，所以 Pos+1=Id，下同，不再说明**）

（6）int UsePos(int Pos);

将为 Pos+1 的 SetId 标记为已用

（7）int FreePos(int Pos);

将为 Pos+1 的 SetId 标记为空闲

（8）int FindNextUsedPos();

查找从 FreePosPoint 开始的最小的可用的 SetId（返回值+1）

（9）int SetToBegin();

将 FreePosPoint 置为 0

2．DocManager

（1）int CreateDoc(int SetId,char* DocName,int Type);

在 Id 号为 SetId 的 DataSet 中创建 type 类型（XML 文档或索引）的 Doc，命名为 DocName

（2）int DeleteDoc(int SetId,char* DocName);

在 Id 号为 SetId 的 DataSet 中删除名为 DocName 的 XML Doc

（3）int OpenDoc(int SetId,char* DocName);

打开 Id 号为 SetId 的 DataSet 中名为 DocName 的 XML Doc，返回 DocId

（4）int WriteDocRoot(int SetId,int DocId,int lpno,int RecordNo);

在 Id 号为 SetId 的 DataSet 中名为 DocName 的 XML Doc 中写入其根节点的 lpno 和 RecordNo

（5）int ReadDocRoot(int SetId,int DocId,int& lpno,int& RecordNo);

读出 Id 号为 SetId 的 DataSet 中名为 DocName 的 XML Doc 根节点的 lpno 和 RecordNo

（6）int LookAllDoc(int SetId);

察看 Id 号为 SetId 的 DataSet 中所有的 Doc 信息（名称）

（7）int FindFreeDoc();

在当前 DataSet 中寻找空闲的最小的 DocId 号，返回

（8）int UseFreeDoc(int Pos);

在当前 DataSet 中标记为 Pos+1 的 DocId 被占用

（9）int MarkFreeDoc(int Pos);

在当前 DataSet 中标记为 Pos+1 的 DocId 空闲可用

（10）int OpenSetFile(int SetId);

打开 Id 号为 SetId 的 DataSet 的 Doc 控制文件

（11）int FindCurPosState(int Pos);

在当前 DataSet 中察看为 Pos+1 的 DocId 是否可用

3．FreePageManager

（1）int FindUsablePage(int SetId);

在 Id 号为 SetId 的 DataSet 中寻找可用页面

（2）int FindEmptyPage(int SetId);

在 Id 号为 SetId 的 DataSet 中寻找新的空页面

（3）int MarkPage(int SetId,int lpno,int Type);

在 Id 号为 SetId 的 DataSet 中标记 lpno 的页面为 type 所指的类型：可用、满、空

（4）int FindCurBlock(int type);

在当前的数据文件所代表的 lpno 范围内寻找 type 类型的页面，返回头一个符合条件的 lpno

（5）int OpenFreePageFile(int SetId);

打开 Id 号为 SetId 的 DataSet 的页面控制文件，将文件句柄放于私有成员变量 fp 中

4．LongFileManager

（1）int ReadLong(int SetId,int RecordId,int length,char* buffer);

读 Id 号为 SetId 的 DataSet 的 Long 数据文件的第 RecordId 号纪录，放入长度为 length，起始处为 buffer 的内存区中

（2）int WriteLong(int SetId,int &RecordId,int length,char* buffer);

在 Id 号为 SetId 的 DataSet 的 Long 数据文件中写入长度为 length，起始处为 buffer 的数据，返回的记录号放于 RecordId 中

（3）int DeleteLongRecord(int SetId,int RecordId);

在 Id 号为 SetId 的 DataSet 的 Long 数据文件中删除记录号为 RecordId 的 Long 数据

（4）int ReConstructLong(int SetId);

重整 Id 号为 SetId 的 DataSet 的 Long 数据文件

（5）int OpenLongFile(int SetId);

打开 Id 号为 SetId 的 DataSet 的 Long 数据文件，将文件句柄放于私有成员变量 fp 中

5．StroageManager

（1）int CreateDataSet(char* DataSetName);

创建名为 DataSetName 的 DataSet

（2）int DeleteDataSet(char* DataSetName);

删除名为 DataSetName 的 DataSet

（3）int OpenDataSet(char* DataSetName);

打开名为 DataSetName 的 DataSet，返回其 SetId

（4）int CreateDoc(char* DataSetName,char* DocName,int Type);

在名为 DataSetName 的 DataSet 中创建 type 类型（XML 文档或索引）的 Doc，命名为 DocName

（5）int DeleteDoc(char* DataSetName,char* DocName);

在名为 DataSetName 的 DataSet 中删除名为 DocName 的 XML Doc

（6）int OpenDoc(int SetId,char* DocName);

打开名为 DataSetName 的 DataSet 中名为 DocName 的 XML Doc，返回 DocId

（7）int WriteDocRoot(char* DataSetName,char* DocName,int lpno,int RecordNo);

在名为 DataSetName 的 DataSet 中名为 DocName 的 XML Doc 中写入其根节点的 lpno 和 RecordNo

（8） int ReadDocRoot(char* DataSetName,char* DocName,int& SetId,int& DocId,int& lpno,int& RecordNo);

读出名为 DataSetName 的 DataSet 中名为 DocName 的 XML Doc 根节点的 lpno 和 RecordNo

（9）int ReadLong(char* DataSetName,int RecordId,int length,char* buffer);

读名为 DataSetName 的 DataSet 的 Long 数据文件的第 RecordId 号纪录，放入长度为 length，起始处为 buffer 的内存区中

（10）int WriteLong(char* DataSetName,int RecordId,int length,char* buffer);

在名为 DataSetName 的 DataSet 的 Long 数据文件中写入长度为 length，起始处为 buffer 的数据，返回的记录号放于 RecordId 中

（11）int DeleteLongRecord(char* DataSetName,int RecordId);

在名为 DataSetName 的 DataSet 的 Long 数据文件中删除记录号为 RecordId 的 Long 数据

（12）int FindEmptyPage(int SetId);

在 Id 号为 SetId 的 DataSet 中寻找新的空页面

（13）int FindUsablePage(int SetId);

在 Id 号为 SetId 的 DataSet 中寻找可用页面

（14）int MarkFullPage(int SetId,int lpno);

在 Id 号为 SetId 的 DataSet 中标记 lpno 的页面为满

（15）int MarkEmptyPage(int SetId,int lpno);

在 Id 号为 SetId 的 DataSet 中标记 lpno 的页面为空闲

（16）int MarkUsablePage(int SetId,int lpno);

在 Id 号为 SetId 的 DataSet 中标记 lpno 的页面为可用

6．AccessManager

（1）StroageManager 的所有 public 函数的外包（1-16，见 5）。

（2）int ReadInData(int SetId,int lpno);

读特定 SetId 的 DataSet 数据空间内特定 lpno 的数据到内存控制数据区（在实现了 BufferManager 的情况下，则读入缓冲区），该数据区大小和页面大小相同。

（3）int PrepareNewPage(int SetId);

在特定 SetId 的 DataSet 数据空间内申请新的页面（因为页面为空，所以不需要读入）

（4）int ReadData (int offset,unsigned char* buffer,int length);

在控制数据区内 offset 的位置读出长度为 length 的数据，该数据将放于 buffer 所指的内存空间中。

（5）int WriteData(int offset,unsigned char* buffer,int length);

在控制数据区内 offset 的位置写入长度为 length 的数据，该数据将放于 buffer 所指的内存空间中。

（6）int ApplyChange();

将控制数据区内的数据写入到磁盘文件中。

（7）unsigned char* GetPagePointer();

得控制数据区的首地址指针。

7．FileManager

（1）int CreateFile(char *FileName);

创建文件名为 FileName 的可读写的二进制文件

（2）int DeleteFile(char *FileName);

删除文件名为 FileName 的二进制文件

（3）FILE* OpenFile(char *FileName);

打开文件名为 FileName 的二进制文件，进行读写

（4）int CloseFile(FILE* fp);

关闭文件

（5）int SeekFile(FILE* fp,long offset);

使文件句柄移到局文件起始处 offset 长度

（6）int GoToFileEnd(FILE* fp);

使文件句柄移到文件当前的末尾处

（7）long TellPos(FILE* fp);

读出文件句柄当前位置

（8）int ReadFile(FILE* fp,unsigned char *buffer,int size);

从文件句柄所指的当前处读出大小为 size 的数据，放于 buffer 所指的内存处

（9）int WriteFile(FILE* fp,unsigned char *buffer,int size);

在文件句柄所指的当前处写入大小为 size 的数据，这些数据的起始处为 buffer

（10）int CreateDataFile(int SetId,int FileNo);

创建 Id 号为 SetId 的 DataSet 的第 FileNo 号数据文件

（11）int MakeDataFileName(int SetId,int FileNo,char* FileName);

得 Id 号为 SetId 的 DataSet 的第 FileNo 号数据文件的实际磁盘文件名

（12）int MakeDocFileName(int SetId,char* DocFileName);

得 Id 号为 SetId 的 DataSet 的 Doc 控制文件名

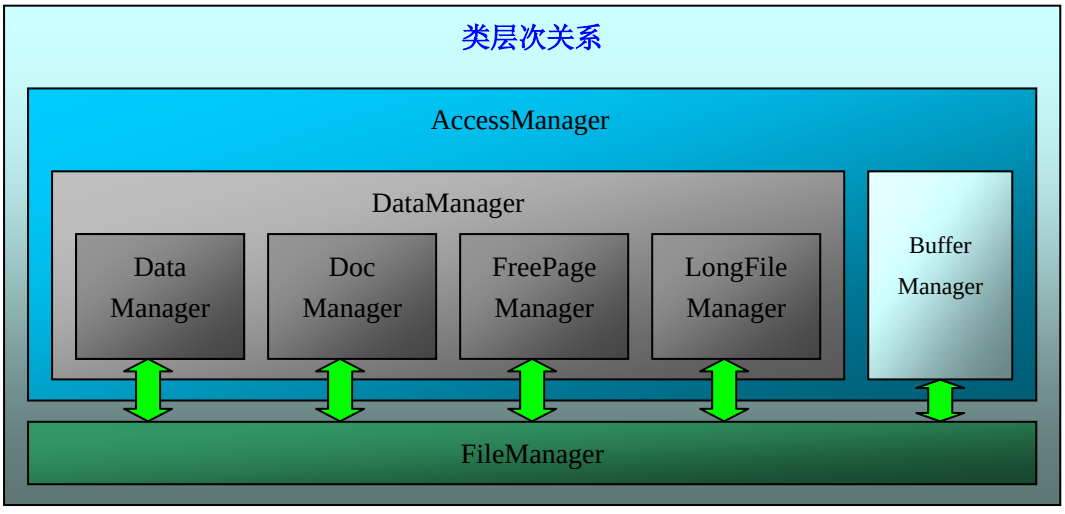
（13）int MakeLongFileName(int SetId,char* LongFileName);

得 Id 号为 SetId 的 DataSet 的 Long 数据文件名

（14）int MakeFreePageFileName(int SetId,char* FreePageFileName);

得 Id 号为 SetId 的 DataSet 的页面控制文件名

四．类关系说明



五．类函数概要设计

1．NX_DataSetManager

(1)

类名：NX_DataSetManager	成员函数名：NX_DataSetManager()	函数类型：建构函数
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：初始化，得到 DataSet 控制文件句柄、DataSet 数量、DataSet 状态信息		
上层函数：Null		
调用函数：		
函数调用格式：		
参数说明： Null		
返回值说明： Null		
全局变量： NX_FileManager FileManager;		
主要局部数据结构和变量：		
算法说明： 打开 DataSetManageFile.nxdb 控制文件，文件句柄放于 fp if(fp == NULL) 创建 DataSetManageFile.nxdb 控制文件 写入 DataSet 数量：0 写入控制信息段：全部 DataSet 的 SetId 为空闲 读出 DataSet 数量 读出控制信息段，放于私有成员变量数组 DataSetFree 中		

（2）

类名：NX_DataSetManager	成员函数名：~NX_DataSetManager()	函数类型：析构函数
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：写出 DataSet 数量、DataSet 状态信息，关闭文件		
上层函数：Null		
调用函数：		
函数调用格式：		
参数说明： Null		
返回值说明： Null		
全局变量：		

NX_FileManager FileManager;
主要局部数据结构和变量：
算法说明： 写入当前 DataSet 数量 写入控制信息段 关闭文件

（3）

类名：NX_DataSetManager	成员函数名：CreateDataSet ()	函数类型：Public
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：创建指定名称的 DataSet		
上层函数：Null		
调用函数： NX_DataSetManager::openDataSet(), NX_DataSetManager::FindFreePos(), NX_DataSetManager::UsePos()		
函数调用格式： int NX_DataSetManager:: CreateDataSet (char* DataSetName)		
参数说明： (in)char* DataSetName 欲创建的 DataSet 名称		
返回值说明： >=1 成功，SetId 号 <0 失败（ErrorCodes::SetFull – 数据集个数满，不能增加新的数据集）		
全局变量： NX_FileManager FileManager;		
主要局部数据结构和变量：		

算法说明：
if(SetId = OpenDataSet(DataSetName) > 0) //已创建
return SetId;
else
FindFreePos();//找到空闲的最小 SetId
在控制文件中相应位置写入 DataSet 的名称
UsePos();//标记这一 SetId 为已用
创建 Doc 控制文件，Doc 数量为 0
创建页面控制文件，所有页面为空		

（4）

类名：NX_DataSetManager	成员函数名：DeleteDataSet ()	函数类型：Public
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：删除指定名称的 DataSet		
上层函数：Null		
调用函数：		

NX_DataSetManager::openDataSet(), NX_DataSetManager:: FreePos(),
函数调用格式： int NX_DataSetManager:: DeleteDataSet (char* DataSetName)
参数说明： (in)char* DataSetName 欲删除的 DataSet 名称
返回值说明： ==0 删除成功 <0 失败（ErrorCodes::SetUnExist – 不存在此名称的 DataSet）
全局变量： Null
主要局部数据结构和变量：

算法说明：
if((SetId = OpenDataSet(DataSetName)) <= 0) //不存在此名称的 DataSet
出错返回
FreePos(SetId -1);// 标记该 SetId 为可用
DataSet 数量减一;

（5）

类名：NX_DataSetManager	成员函数名：OpenDataSet ()	函数类型：Public
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：打开指定名称的 DataSet，返回 SetId		
上层函数：Null		
调用函数： NX_DataSetManager:: SetToBegin(); NX_DataSetManager:: FindNextUsedPos()		
函数调用格式： int NX_DataSetManager:: OpenDataSet (char* DataSetName)		
参数说明： (in)char* DataSetName 欲打开的 DataSet 名称		
返回值说明： >0 该 DataSet 的 SetId， <0 失败（ErrorCodes::SetNotFind – 不存在此名称的 DataSet ErrorCodes::SetFileErr – DataSet 控制文件毁坏）		
全局变量： NX_FileManager FileManager		
主要局部数据结构和变量：		

算法说明：
SetToBegin();//FreePosPoint 置 0
For(i = 0 ; i < SetNum ; i ++)
if((Pos = FindNextUsedPos()) == -1) // DataSet 控制文件毁坏
return SetFileErr;		

SetId = Pos+1; 读出这一 DataSet 的名称，与 DataSetName 比较，相符则退出循环 如果没有找到，返回 SetNotFind 找到则返回 SetId

（6）

类名：NX_DataSetManager	成员函数名：FindNextUsedPos()	函数类型：Private
源文件名称：NxdbDataSetManager.cpp	头文件名称：NxdbSm.h	
函数功能：查找 DataSet 控制信息中与 DataSetFree 标记最近的 Pos，返回（SetId-1）		
上层函数：OpenDataSet()		
调用函数： Null		
函数调用格式： int NX_DataSetManager:: FindNextUsedPos ()		
参数说明：		
返回值说明： >=0 DataSet 控制信息的 Pos ==-1　没找到可用的 Pos		
全局变量： NX_FileManager FileManager		
主要局部数据结构和变量：		
算法说明： 保存当前的 Pos 记数 FreePosPoint 到 BeforePos do 如果当前的 Pos 为已用，退出循环 FreePosPointer++ While(FreePosPoint != BeforePos) 如果没有找到，返回-1 找到已用的 Pos，返回		

（7）SetToBegin()，FindFreePos()，UsePos()，FreePos()
使用字节移位操作，简单，不再赘述。

2．NX_DocManager

（1）

类名：NX_DocManager	成员函数名：NX_DocManager ()	函数类型：建构函数
源文件名称：NxdbDocManager.cpp	头文件名称：NxdbSm.h	
函数功能：文件句句柄、当前 SetID 号置空		
上层函数：NULL		
调用函数：Null		
函数调用格式：		
参数说明：		

返回值说明：
全局变量：
主要局部数据结构和变量：
算法说明： 文件句句柄 fp 为空 当前 SetID 号置为-1

（2）

类名：NX_DocManager	成员函数名：~NX_DocManager ()	函数类型：析构函数
源文件名称：NxdbDocManager.cpp	头文件名称：NxdbSm.h	
函数功能：把当前活动 DataSet 的控制信息回写到 Doc 控制文件		
上层函数：NULL		
调用函数：Null		
函数调用格式：		
参数说明：		
返回值说明：		
全局变量：NX_FileManager FileManager		
主要局部数据结构和变量：		
算法说明： if(当前 SetId 不为空) 回写当前 DataSet 中 Doc 数量 回写当前 DataSet 的 Doc 控制信息（空闲信息）		

（3）

类名：NX_DocManager	成员函数名：OpenSetFile()	函数类型：Private
源文件名称：NxdbDocManager.cpp	头文件名称：NxdbSm.h	
函数功能：根据需要打开的 SetId 号，读取相应 DataSet 的控制信息		
上层函数： NX_DocManager::CreateDoc(); NX_DocManager::DeleteDoc(); NX_DocManager::OpenDoc()		
调用函数：Null		
函数调用格式：NX_DocManager:: OpenSetFile(int SetId)		
参数说明： (in)int SetId　欲读取 Doc 控制信息的 DataSet 的 SetId		
返回值说明： ==0 成功读取控制信息 <0　失败（ErrorCodes:: DocCtlNotFind – 未找到控制文件）		
全局变量：NX_FileManager FileManager		
主要局部数据结构和变量：		

算法说明：
if(当前 SetId != 预读取的 SetId) //说明要重新读取信息
if(当前 SetId != -1) //说明已经打开了一个 DataSet 的控制文件，信息必须回写		

回写 Doc 数量 回写 Doc 控制信息 装配新的 DataSet 的 Doc 控制文件名 读取 Doc 数量 读取控制信息

（4）

类名：NX_DocManager	成员函数名：CreateDoc()	函数类型：Public
源文件名称：NxdbDocManager.cpp	头文件名称：NxdbSm.h	
函数功能：在制定 SetId 的 DataSet 中创建指定名称和类型的新文档		
上层函数：		
调用函数： NX_DocManager::OpenSetFile() NX_DocManager::OpenDoc(); NX_DocManager:: FindFreeDoc(); NX_DocManager:: UseFreeDoc();		
函数调用格式：NX_DocManager:: CreateDoc (int SetId,char* DocName,int Type)		
参数说明： (in)int SetId　欲创建 Doc 的 DataSet 的 SetId (in)char* DocName 欲创建的 Doc 名称 (in)int Type 欲创建的 Doc 类型（NX_XML_DATA_TYPE　XML 数据 NX_XML_INDEX_TYPE XML 索引）		
返回值说明： ==0 成功创建文档 <0　失败（ErrorCodes:: DocCtlNotFind – 未找到控制文件 ErrorCodes:: DocNumFull – Doc 数量已达最大，不能再添加）		
全局变量：NX_FileManager FileManager		
主要局部数据结构和变量：		

算法说明：
OpenSetFile(setid); //读取控制信息
if((DocId = OpenDoc(DocName)) > 0)//该文档存在，直接返回 DocId
return DocId;
FindFreeDoc();//找空闲的 DocId
在 Doc 控制文件中写入文件名
UseFreeDoc()；//标记该 DocId 已用
文件数量加一		

（5）

类名：NX_DocManager	成员函数名：OpenDoc()	函数类型：Public
源文件名称：NxdbDocManager.cpp	头文件名称：NxdbSm.h	
函数功能：找到指定名称的文档的 DocId。		
上层函数： NX_DocManager::CreateDoc() NX_DocManager::DeleteDoc()		

调用函数： NX_DocManager:: OpenSetFile( ) NX_DocManager: FindCurPosState();
函数调用格式：NX_DocManager:: OpenDoc(int SetId,char* DocName)
参数说明： (in)int SetId　欲创建 Doc 的 DataSet 的 SetId (in)char* DocName 欲打开的 Doc 名称
返回值说明： >=0 DocId 号 <0　失败（ErrorCodes:: DocCtlNotFind – 未找到控制文件 ErrorCodes:: OpenDocNotFind–未找到指定名的 Doc 信息）
全局变量：NX_FileManager FileManager
主要局部数据结构和变量：

算法说明：
OpenSetFile(setid); //读取控制信息
当前文档号置 0
while(当前文档号 < 文档总数)
DocId = FindCurPosState();找下一个使用的 DocId
读取名称，与参数 DocName 比较，相同则退出
找到，返回 DocId
未找到，返回 OpenDocNotFind

（6）DeleteDoc()，WriteDocRoot()，ReadDocRoot()比较简单
FindFreeDoc()，FindCurPosState()，MarkFreeDoc()，UseFreeDoc()字节操作，不再赘述

3．NX_FreePageManager
（1）建构函数、析构函数同 NX_DocManager()，不再赘述
（2）

类名：NX_FreePageManager	成员函数名：OpenFreePageFile()	函数类型：Private
源文件名称：NxdbFreePageManager.cpp	头文件名称：NxdbSm.h	
函数功能：根据需要打开的 SetId 号，读取相应 DataSet 的页面控制信息		
上层函数： NX_FreePageManager::FindUsablePage() NX_FreePageManager::FindEmptyPage() NX_FreePageManager::MarkPage()		
调用函数：Null		
函数调用格式：NX_DocManager:: OpenFreePageFile (int SetId)		
参数说明： (in)int SetId　欲读取页面控制信息的 DataSet 的 SetId		
返回值说明： ==0 成功读取控制信息 <0　失败（ErrorCodes:: FreeCtlNotfind– 未找到控制文件）		
全局变量：NX_FileManager FileManager		
主要局部数据结构和变量：		

算法说明：
if(当前 SetId != 预读取的 SetId) //说明要重新读取信息
if(当前 SetId != -1) //说明已经打开了一个 DataSet 的控制文件，信息必须回写
回写 FreePage 的 block 数量（一个 Block 代表一个数据文件）
装配新的 DataSet 的页面控制文件名
读取 FreePage 的 block 数量
读取头一个 Block(数据文件)的页面分配信息

（3）

类名：NX_FreePageManager	成员函数名：FindUsablePage();	函数类型：Public
源文件名称：NxdbFreePageManager.cpp	头文件名称：NxdbSm.h	
函数功能：找到最小的可用页面		
上层函数：		
调用函数： NX_FreePageManager::OpenFreePageFile(); NX_FreePageManager::FindCurBlock()		
函数调用格式：NX_DocManager:: OpenFreePageFile (int SetId)		
参数说明： (in)int SetId 欲读取页面控制信息的 DataSet 的 SetId		
返回值说明： >=0 成功，符合条件的页面 lp 号 <0 失败（ErrorCodes:: FreeCtlNotfind– 未找到控制文件）		
全局变量：NX_FileManager FileManager		
主要局部数据结构和变量：		

算法说明：
OpenFreePageFile();//打开页面控制文件，读取控制信息
如果读取出的 Block 数为 0，说明当前数据集没有数据，创建头一个数据文件
根据当前的 Block 数，遍历，找到最小的可用 lpno
如果所有 Block 都没有满足条件的页面
如果 Block 数达到了最大，返回，出错
否则创建新的数据文件，Block 数+1，在控制文件中增加该短的页面信息
返回该 Block 的头一个 lpno		

（4）FindEmptyPage()同(3)

（5）MarkPage()字节操作

4．NX_LongFileManager

5．NX_StroageManager

所有函数皆为 DataSetManager、DocManager、FreePageManager、LongFileManager 公有的外包

6．NX_AccessManager

所有函数皆为 StroageManager 和 BufferManager 的外包

7．NX_FileManager

只描述 OpenFile 和 CloseFile，其他的较简单。

（1）

类名：NX_FileManager	成员函数名：OpenFile();	函数类型：Public
源文件名称：NxdbFm.cpp	头文件名称：NxdbFm.h	
函数功能：打开指定名称的文件，获得文件句柄		
上层函数：		
调用函数：		
函数调用格式：FILE* NX_ FileManager:: OpenFile (char* FileName)		
参数说明： (in)char* FileName 欲打开的文件名		
返回值说明： >=0 成功，该文件句柄 NULL 失败		
全局变量：		
主要局部数据结构和变量：		
算法说明： 在文件名数组中遍历，如找到相同的文件名，说明已经打开，返回句柄 否则打开该文件 加文件名到 FileNameS 数组 加文件句柄到 FpS 数组 FileRefS 数组的对应最后元素置一 当前打开的文件数加一		

（2）

类名：NX_FileManager	成员函数名：CloseFile();	函数类型：Public
源文件名称：NxdbFm.cpp	头文件名称：NxdbFm.h	
函数功能：关闭文件		
上层函数：		
调用函数：		
函数调用格式：FILE* NX_ FileManager:: OpenFile (char* FileName)		
参数说明： (in)char* FileName 欲打开的文件名		
返回值说明： >=0 成功，该文件句柄 NULL 失败		
全局变量：		
主要局部数据结构和变量：		
算法说明： 在文件名数组中遍历，如找不到相同的文件名，出错，返回 否则 FileRefS 数组的对应元素减一 如果该元素变为 0，说再也没有应用使用该文件 关闭该文件		

三个数组该位置以后所有元素前移 当前打开的文件数减一