OrientX 系统程序员开发手册

版本号：OrientX Version 2.5
完成时间：2005 年 4 月 30 日
开发单位：中国人民大学 IDKE 实验室 XML 工作组

1. 引言

1.1 编写目的

本说明书主要是介绍如何基于 OrientX 系统进行应用开发，包括简单介绍 OrientX 系统的主要功能、示例说明、各个模块的应用接口等。

1.2 系统概述

随着因特网应用的发展，XML 逐渐成为数据描述和数据交换的标准，大量的 XML 文档出现在网络中；有效地存储 XML 数据并提供高效的 XML 数据查询，成为当前急需解决的问题。

OrientX 系统正是在这样的应用背景下产生的，它以 Native 方式存储 XML 数据，并支持 XPath 和 XQuery 等 XML 查询以读取数据。所谓的 XML 的 Native 存储方式，就是存储时保留数据的树形模式；根据一个结点可以直接找到其孩子结点、左右兄弟结点或父亲节点等。以 Native 方式存取 XML 数据，就无需进行数据模式的转换，也不需要进行查询语言的转换。

OrientX 是一个 Native XML 数据库管理系统（Native XML DataBase Management System）。OrientX 是 **O**riginal **R**UC **I**DK**E** **Nat**ive **X**ML 的缩写，表示 OrientX 系统是中国人民大学 IDKE 实验室研发的 Native XML 数据管理系统。该系统的研发工作是从 2001 年至今，已经历时 3 年时间。

若想更详细地了解 OrientX 系统，请参照文档“OrientX 系统说明书”。

1.2.1 系统功能

OrientX 是 Native XML 数据库管理系统，主要功能如下的：

1. 数据库的建立和维护
2. 数据操纵,包括数据导入/导出，数据检索，数据更新等。
3. 数据库运行管理
4. 数据组织、存储和管理功能
5. 数据通信接口

1.2.2 系统特征

OrientX 系统作为一个数据库管理系统，其系统特征具有如下的

1. 它是基于模式（Schema based）的 Native XML 数据库管理系统。
2. 多样化的数据组织和存储方式
3. 数据存取路径
4. 数据模型遵循 XQuery 1.0 and XPath 2.0 Data Model 标准。
5. 数据检索支持 W3C 推荐的 XQuery1.0 标准

1.3 如何得到 OrientX 系统

OrientX 系统是中国人民大学 IDKE 实验室研发的成果。版权和所有权归中国人民大学 IDKE 实验室所有。但其内容可以免费提供予任意与商业利益无关的实验和研究活动。

如果用户需要 OrientX 系统相关的程序包、源代码、说明文档，请向我们索取：OrientX@ruc.edu.cn 或登录我们的主页：http://idke.ruc.edu.cn/OrientX

关于 OrientX 系统的功能演示，请登录我们的主页：http://idke.ruc.edu.cn/OrientX

1.4 OrientX Mailing List

如果您在使用 OrientX 系统过程中，遇到问题，请参照 OrientX 系统的相关文档；或者登录我们的主页：http://idke.ruc.edu.cn/OrientX 参考常见问题解答(Frequent Questions)；或者给我们发 email：OrientX@ruc.edu.cn ，我们将尽快给您回复解答。若您对我们的系统工作有什么意见，欢迎反馈：OrientX@ruc.edu.cn 。

2.系统结构

描述 OrientX 系统的总体架构的最好方法是，通过检查它是如何导入一个 XML 文档并存储到数据库中去；然后在其上进行一个 XML 查询，看看它是如何实现的，如何导出文档的实现过程。

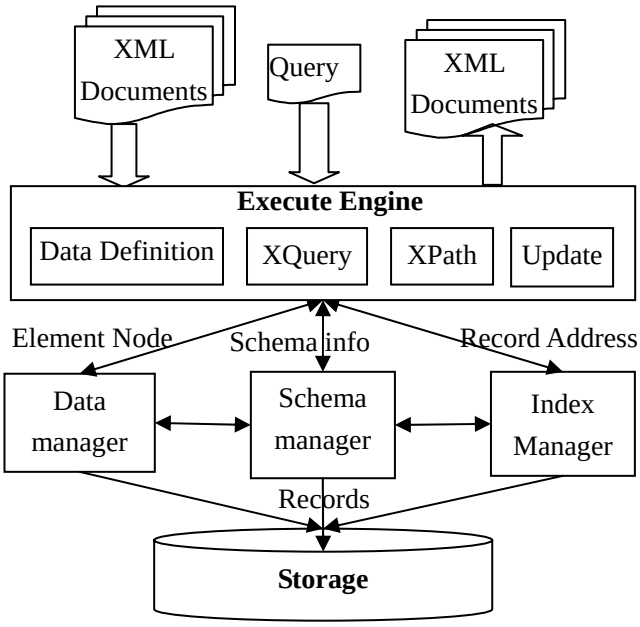

图 1 **系统框架图**

如图 1 所示，用户用 OrientX 系统管理 XML 数据，首先需要通过执行引擎（Execute Engine）模块建立一个数据库（前面已经说过，OrientX 系统中每一个数据库对应于具有相同的模式的数据集 dataset）。这就是数据定义（Data Definition），它确定了数据集内的所有文档的模式结构。导入文档时，执行引擎把文档传送到数据管理（Data Manager）模块；数据管理模块从逻辑上把 XML 文档划分成多个记录。然后传输到存储（Storage）模块，选择

适当的文件机构进行存储。

当需要对数据进行查询检索时，一个 XML 查询（XPath 或者 XQuery 查询）以文本的形式传送到查询执行引擎（XQuery 和 XPath 执行引擎的处理处理策略有很大不同）；在查询执行引擎中，XML 查询将被分析(parse)成一个查询执行计划，此过程中从模式管理模块（Schema manager）读取相关信息，判断该查询是否存在语义错误，例如目标文档或数据库是否存在，XPath 路径中的结点在对应的模式中是否存在等问题；如果存在这样的错误，则系统就报告错误，查询不再往下执行。查询执行引擎还可以对查询计划进行优化；如果存在合适的索引可以优化查询执行效率，查询执行引擎就可以通过索引管理模块（index manager）直接访问数据库，而不需要通过数据管理模块（Data Manager）导航式地访问数据库。

若想更详细地了解 OrientX 系统各个模块的设计，请参照文档“OrientX 系统说明书”以及各个模块的概要设计说明书。

3.运行环境

3.1 配置要求：

OrientX 系统运行的硬件平台要求：

- 操作系统：Windows 98/NT/2K/XP
- 编译器：Microsoft Visual C++ 6.0
- 最低的可用空间要求：10MB
- 最低的内存空间要求：64MB

3.2 目录设置(Dictionaries Settings)：

1. 以下的操作步骤假定您的工程所在的目录为 D:\OrientXAPI，如果您的工程文件在另外的位置，请对目录做相应调整。
2. 在 VC++中选择“Tools”菜单下“Options”菜单项，选择“Directories”面板，在“Show directories for”下拉框中选中“Include files”选项，添加一个目录 “D:\OrientXAPI\OrientX”。
3. 以上各步完成后，单击“OK”按钮，确认修改。

3.3 添加库文件(Add Lib Files to Project)：

1. 在 VC++中选择“Project”菜单下“Add to Project”，“Files”菜单项，弹出“Insert Files into Project”对话框，在“查找范围”下拉框中选择目录“D:\OrientXAPI”，在“文件类型”下拉框中选择“所有文件 (*.*)”，选中“OrientXd.lib”，单击“OK”按钮确认。

3.4 工程设置(Project Settings)：

OrientX 系统 2.0 版本只支持在 Debug 环境下进行开发，其工程设置如下：
1. 在 VC++中选择“Project”菜单下“Settings”菜单项，选择“Link”面板，在“Ignore libraries”文本框中添加一个库文件“libcd.lib”。

2. 在 VC++中选择“Project”菜单下“Settings”菜单项，选择“C/C++”面板，在“Code Generation”下拉框中选中“Code Generation”选项，在“Use run-time library”下拉框中选中“Debug Multithreaded”选项。
3. 以上各步完成后，单击“OK”按钮，确认修改。

4 使用示例

OrientX 2.0 版本中把系统所支持的主要功能封装到 CExecuteEngine 类中，向最终用户提供简单统一的接口，实现建库/删库，导入/导出/删除文档，查询，更新等功能。

4.1 系统初始化和结束

系统的初始化和系统结束时的资源释放，分别通过调用函数 CExecuteEngine::Initialize()和 CExecuteEngine::clear()实现。下面的程序演示了系统的主要功能；程序保存在文档“OrientX.cpp”中。

示例程序：

```
#include "include/share/ExecuteEngine.h"
int main()
{
    //to initialize the system.
    CExecuteEngine::Initialize();

    //create a database name "bib"
    CExecuteEngine::CreateDataSet("bib", "E:\\OrientX_2_0_API\\OrientX\\example\\bib.xsd");
    CExecuteEngine::PrintDataSetName();
    cout<<endl;

    //to import a document to database "bib"
    CExecuteEngine::ImportDoc("bib", " E:\\OrientX_2_0_API\\OrientX\\example\\bib.xml");

    //to export the document "bib.xml" in the database "bib"
    CExecuteEngine::ExportDoc("bib", "bib.xml", "output_bib.xml");
    CExecuteEngine::PrintDocName("bib");
    cout<<endl;

    //to commit an XQuery with the query kept in file "Xquery.txt"
    CExecuteEngine::ExecXQuery("E:\\OrientX_2_0_API\\OrientX\\example\\XQuery.txt",
"XQueryResult1.xml") ;

    //to commit an XQuery with the query in the parameter directly. note the 3rd parameter.
 CExecuteEngine::ExecXQuery("import default schema \"bib\"    document('bib@bib.xml')
 //book[price > 100]", "XQueryResult2.xml", false) ;
```

```
    //to commit an XPath    with the query in the parameter directly. note the 3rd parameter.
    CExecuteEngine::ExecXPath("document(\"bib@bib.xml\")//author", "XPathResult.xml",
false);

    //to commit an update in file.
    CExecuteEngine::ExecXQuery("E:\\OrientX_2_0_API\\OrientX\\example\\update.txt");
    CExecuteEngine::ExportDoc("bib","bib.xml","bib_update.xml");

    //to drop the document and dataset.
    CExecuteEngine::DropDoc("bib","bib.xml");
    CExecuteEngine::DropDataSet("bib");
    CExecuteEngine::clear();

    return 0;
}
```

通过 CExecuteEngine 只能提交任务，任务的运行结果（如果有的话）被输出到指定的文件中。但是无法直接访问到其内部的数据，例如用户通过 CExecuteEngine 提交了一个 XQuery，其结果输出到了指定的文件中，用户无法对查询的结果作进一步的开发！因此需要直接调用各个模块的接口。下面就是实现各个模块的功能示例程序。

4.2 查询执行（Query Process）

4.2.1XML 查询分析

下例示范如何声明 XPath 查询分析器，如何输入 XPath，并对其进行语法分析，得到相应的执行计划，然后再解析成 XPath 执行引擎所能够识别的内部结构。存放下列代码的文档名为“XPathParser.cpp”

```
#include "include/QueryParser/Xpath_parser.h"
int main()
{
    //初始化系统
    //… ….
    //XPath查询分析器
    Xpath_parser parser;
    //重置输入查询的文档路径和文档名
    parser.SetInputFile("c:\\input.txt");
    //重置输出分析过程的文档路径和文档名
    parser.SetOutputFile("c:\\DebugOutput.txt");
    //重置输出错误信息的文档路径和文档名
    parser.SetErrorFile("c:\\ErrorMessage.txt");
```

```
    //开始分析
    if( parser.Run() == 0)
        return -1;//分析失败：输入的查询有误

    //读取分析结果：用内部结构表示的XPath查询
    CxpXpathExpr* XPath=parser.GetXpath();

    //后续操作，例如传递给XPath查询执行引擎，释放系统资源等
    //... ...

}
```

4.2.2 XPath 查询执行

下面是 XPath 查询引擎的使用示例。说明如何对经过语法分析的 XPath 语句进行查询优化和执行。

```
#include "include/publicHeaders/stdafx.h"
#include "include/Datamanager/NxdbDataManager.h"
#include "include/queryprocess/QueryProcess.h"
#include "include/QueryParser/Xpath_parser.h"
#include "include/share/ExecuteEngine.h"

//引用外部变量
extern NxdbDataManager g_DataManager;
extern Xpath_parser g_XPathParser;
extern QueryProcess g_XPathExecute;

int main()
{
    //前续操作，初始化系统，建库导入文档，分析 XPath 查询等
    //… …

    //读取分析结果：用内部结构表示的 XPath 查询
    CxpXpathExpr* XPath= g_XPathParser.GetXpath();

    //假定输入的查询是一个完整的 XPath 查询，即以 document("XMLFile.xml")开始
    NxdbList result;
    g_XPathExecute.EvaluateXPath(NULL,XPath,result);

    //把结果集 result 打印到指定的文件 resultFile 上
    outputResult("C:\\OrientX\\test\\result.xml", result);

}
```

说明：在执行这段程序之前，需要将XPath语句写入分析器的输入文件（inputfile.txt）；执行的结果输出到文档“result.xml”，同时保存在变量result中，以待下一步处理。

4.2.3 XQuery 导航式处理

下面的程序演示了如何使用XQuery执行引擎进行数据检索，还可以在其执行结果上进行进一步的开发。

```
#include "include/share/ExecuteEngine.h"
extern CxqeXQueryExecute g_XQueryExecute;

int main()
{
	int retFlag = 0;
	CExecuteEngine::Initialize();

	char* inputfile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\XQuery\\query.txt";
	char* outputFile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\XQueryResult.txt";

	//to commit an XQuery and evaluate it.
	try{
	// ExecXQuery(const char* query, const char* outputFile = "resultFile.xml",
	//		bool inFile = true, bool inNavigation = true);
		retFlag = CExecuteEngine::ExecXQuery(inputfile, outputfile, true, true) ;
	}
	catch (NxdbException& e) {
		cout<<"Exception: "<< e.GetExceptionCode()<<" "<<e.GetExceptionMsg()<<endl;

	}

	CExecuteEngine::clear();
	return retFlag;
}
```

4.2.4 XQuery 代数式处理

下面的程序演示了如何使用XQuery执行引擎进行数据检索，还可以在其执行结果上进行进一步的开发。

```
#include "include/share/ExecuteEngine.h"
extern CxqeXQueryExecute g_XQueryExecute;

int main()
```

```
{
	int retFlag = 0;
	CExecuteEngine::Initialize();

	char* inputfile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\XQuery\\query.txt";
	char* outputFile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\XQueryResult.txt";

	//to commit an XQuery and evaluate it.
	try{
	// ExecXQuery(const char* query, const char* outputFile = "resultFile.xml",
	//		bool inFile = true, bool inNavigation = true);
		retFlag = CExecuteEngine::ExecXQuery(inputfile, outputfile, true, false) ;
	}
	catch (NxdbException& e) {
		cout<<"Exception: "<< e.GetExceptionCode()<<" "<<e.GetExceptionMsg()<<endl;

	}

	CExecuteEngine::clear();
	return retFlag;
}
```

4.3 创建/删除数据库，导入/导出文档

下面的示例程序介绍了如何建立、删除数据集，如何导入、导出和删除文件。存放下列代码的文档名为“datamanagementTest.cpp”

```
//要包含的头文件
#include "include/share/ExecuteEngine.h"
#include "include/datamanager/nxdbdatamanager.h"

//引用外部变量
extern	NxdbDataManager g_DataManager;

int main()
{
	//initialize the system

	//create a new dataSet
	if( g_DataManager.CreateDataSet("Xmark"," xmark.xsd") == -1)
		return -1;

	//import a new document
	if( g_DataManager.ImportDoc("Xmark"," xmark1.xml") == -1)
```

```
        return -1;

    //export the specified document with the xml format
    if( g_DataManager.ExportDoc("xmark","xmark1.xml"," output1.xml") == -1)
        return -1;

    //delete the specified document
    if( g_DataManager.DropDoc("xmark","xmark1.xml") == -1)
        return -1;

    //delete the dataset
    if( g_DataManager.DropDataSet("xmark") == -1)
        return -1;

    //clear the system and release resource
}
```

说明：

如果需要重新设定存放数据的目录，则在初始化系统之后，立刻调用函数SetDataPath(). 例如：设置Data目录为C:\OrientX\Data，则在函数g_SysManager.initiate()之后，调用函数g_DataManager.SetDataPath("c:\\orientx")。

4.4 创建/删除索引

下面的示例演示了如何建立路径索引，删除索引，最后关闭索引。此程序代码存为文件indextest.cpp。

```
#include "include/share/ExecuteEngine.h"
#include "include/accessmanager/nxdbsm.h"
#include "include/publicheaders/indexmanagerheads.h"

extern NX_AccessManager g_AccessManager;

int main()
{
    CExecuteEngine::Initialize();

    CExecuteEngine::CreateDataSet("xmark",
"E:\\project\\Orientx2.0Server\\OrientX_2_0\\data\\books.xsd");
    CExecuteEngine::PrintDataSetName();
    cout<<endl;

    CExecuteEngine::ImportDoc("xmark",
    "E:\\project\\Orientx2.0Server\\OrientX_2_0\\data\\books.xml");
```

```
//here suppose dataset "XMark" already exits in database.
int dataset = g_AccessManager.OpenDataSet("xmark");

NX_DocList doclist = g_AccessManager.GetDocList(dataset);

//define the pathindexmanager
NewPathIndex pimanager;
pimanager.Open();

for (int i=0; i< doclist.getLength(); i++){
	char* doc = doclist.getName(i);
	int docId = g_AccessManager.OpenDoc(dataset, doc);
	pimanager.CreatePathIndex (dataset, docId);
}


//search processing
PathIndexHandle handler;

//assume eID is a properly initiated DTDNodeCode
{
	char* doc = doclist.getName(0);
	int docId = g_AccessManager.OpenDoc(dataset, doc);
	pimanager.GetHandler (dataset, docId, eID, handler);
}

//Get All the record that has the same DTDNodeCode as eID
NxdbList list;
handler.GetAllRecords(list);

//delete path index
for (int i=0; i< doclist.getLength(); i++){
	char* doc = doclist.getName(i);
	int docId = g_AccessManager.OpenDoc(dataset, doc);
	pimanager.DropPathIndex (dataset, docId);
}

pimanager.Close()

CExecuteEngine::DropDataSet("xmark");
CExecuteEngine::clear();
```

```
    return 0;
}
```

4.5 更新

更新模块借用的是 XQuery 模块的查询引擎，可以说是 XQuery 上的一个开发实例，下面的示例程序演示了更新模块的使用，程序放在文件”XupdateTest.cpp”中，该程序和文件“XQueryTest.cpp”中的示例程序很类似。用户可以用更新语言实现自己的更新。所以第 5.6 节还将介绍从 XQuery 扩展的的更新语言。

文件 u8.up 的更新语句：

```
import default schema      "bib"
for $b in document("bib@bib.xml")/book[title = "TCP/IP Illustrated"]

insert    after    $b    element    values
   <book >
      { $b/author }
      <title>"Thinking in Java"</title>
   </book>
```

```
#include "INCLUDE/PublicHeaders/XQExeNavigation.h"
#include "include/share/ExecuteEngine.h"
extern CxqeXQueryExecute g_XQueryExecute;

int main()
{
    CExecuteEngine::Initialize();
    g_XQueryExecute.initialize();

    char* inputfile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\Update\\u8.cpp ";
    char* outputFile = "E:\\develop\\OrientX_2_0_API\\OrientX\\example\\XQueryResult.txt";

    //to commit an XQuery and evaluate it.
    if( g_XQueryExecute.run(inputfile,outputFile) == -1)
    {
        cout<<"failed."<<endl;
        return -1;
    }
    else
        cout<<"OK."<<endl;

    //...

    //to develop application based on the OrientX's XQuery engine.
```

```
    g_XQueryExecute.clear();
    CExecuteEngine::clear();
    return 0;
}
```

该更新语句的含义是：在文档 bib.xml 的某个 book 元素后插入新构造的元素，该 book 的 title 的值是"TCP/IP Illustrated"，新构造的元素的 title 是"Thinking in Java"，作者和前一个 book 的作者一样。

5.各模块的 API 接口

5.1 ExecuteEngine (uniform interface)

ExecuteEngine 是 OrientX 系统中集成的统一接口类。它将数据管理，模式管理，查询执行和用户交互等功能集成起来，极大的方便了用户应用的开发。

ExecuteEngine 中定义了一系列静态方法，分别支持以上的种种处理。注意，在这些处理执行的前，需要对调用 ExecuteEngine::Initialize()对系统的执行环境进行初始化；处理结束后，还需要调用 ExecuteEngine::Clear()做系统环境关闭的工作。下面是它的公共接口的简单描述。

另外，由于 ExecuteEngine 提供的接口实现了对系统底层的数据存取和数据管理的接口，所以这两个模块的细节这里不再介绍；查询处理模块增加了代数处理的内容，但是基本不涉及 OrientX 系统之上的开发，所以细节同样从略。

```
/**********************************************************************
** Description:
to initialize the OrientX system,such as build up the system dataset,allocate the buffer etc.
** Return Values:
      return 0 if successfully, -1 otherwise.
**********************************************************************/
static int Initialize();

/**********************************************************************
** Description:
      to the clear system environment and release the system resource
** Return Values:
      return 0 if successfully, -1 otherwise.
**********************************************************************/
static int clear();

/**********************************************************************
   ** Description:
       create a dataset with a schema,allocate space for the dataset,create dtdtree,block tree,
   ** Input: char* dataSetName,char* dtdFileName
```

```
            dataSetName---name of the dataset to be created
            dtdFileName---name of the schema file related to the dataset
                          each dataset has a unique schema file
  ** Return Value:
            0--success, -1--false
*****************************************************************/
static int    CreateDataSet(const char* dataSetName,const char* dtdFileName);

/*****************************************************************
 ** Description:
      drop the specified dataset,its oid dataset,and related index on the dataset,
 ** Input: char* dataSetName
          datasetName--name of set which to be drop
 ** Return Value:
            0--success, -1--false
*****************************************************************/
static int    DropDataSet(const char* dataSetName);

/*****************************************************************
** Description:
**          Just return the id of the dataset with the specified name.
** Input: char* DataSetName
**          DataSetName---The name of the data set.
** Output:
**          If success, return the id(beginning from 1)
**          else return <0 or some exception is thrown out.
*****************************************************************/
//static int OpenDataSet(const char* dataSetName);

/*****************************************************************
 ** Description:
    import a xml document into the specified dataset in the specified storage mode
 ** Input:
    datasetName--- name of the dataset that to be imported a document into
    docName--- name of the document which to be imported
                include the path,analyse the path to get the name of the document
                the document in the dataset has the same name with its source document
 ** Return Value:
            0--success, -1--false
*****************************************************************/
static int ImportDoc(const char* dataSetName, const char* docName);

/*****************************************************************
** Description: export an specified document in the xml format
```

```
** Input: char* dataSetName,char* oldDocName,char* newDocName
        datasetName---:specify the dataset which the document to be exported belongs to
        docName---: specify the document to be exported
        outputFile---: assign a new name to the export document
** Return Value:
        0--success,     -1--false
*****************************************************************/
static int ExportDoc(const char* dataSetName, const char* docName,const char* outputFile);

/*****************************************************************
** Description:
    drop a specified document
** Input: char* dataSetName,char* docName
        datasetName---specify the dataset which the document to be dropped belongs to
        docName---specify the document to be dropped
  ** Return Value:
        0--success, -1--false
*****************************************************************/
static int DropDoc(const char* dataSetName,const char* docName);


/*****************************************************************
** Description:
    to execute an XPath query by XPath processing module.
** Parameters:
    inFile -- pointing out the query is in file or string format.

    query -- if inFile is true, it is the file keeps the query expression.
          -- if inFile is false, it is the query expression
    outputFile -- the file to export the query result.
            -- as default,ouput to the file "resultFile.xml" in current directory .
** Return Values:
    return 0 if successful, return -1 otherwise.
*****************************************************************/
static int ExecXPath(const char* query, const char* outputFile = "resultFile.xml", bool inFile
= true);

/*****************************************************************
** Description:
    to execute an Xquery query by XQuery processing module.
** Parameters:
    inFile -- pointing out the query is in file or string format.
            -- as default,the query is input in file format.
    query -- the file name to keep the query express,if inFile is true.
```

```
                -- the query express if inFile is false.
        outputFile -- the file to export the query result.
                        -- as default,ouput to the file "resultFile.xml" in current directory .
        inNavigation -- to indicate to process the Query in navigation or in Algebra
    ** Return Values:
        return 0 if successful,return -1 otherwise.
    ************************************************************************/
    static int ExecXQuery(const char* query, const char* outputFile = "resultFile.xml", bool inFile = true, bool inNavigation = true);


    /************************************************************************
     ** Description:
          to get the names of all dataset
     ** Params:
          dataSetNameList -- to keep the data set names for the caller
    ************************************************************************/
    static void GetDataSetName(vector<char*>& dataSetNameList );

    /************************************************************************
     ** Description:
          to get all the document names in a specified data set.
     ** Params:
          dataSetName -- to specify the target data set
          docNameList -- to keep the document names for the caller
    ************************************************************************/
    static void GetDocName(const char* dataSetName, vector<char*>& docNameList);

    /************************************************************************
     ** Description:
        print all the dataset name.
    ************************************************************************/
    static void PrintDataSetName();

    /************************************************************************
     ** Description:
        to print all document name in the specified dataset.
     Params:
        dataSetName, the specified dataset
    ************************************************************************/
    static void PrintDocName(const char* dataSetName);
```

5.2 模式管理(Schema Manager)

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，结点的数据类型。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 有五个数据结构类（Attribute, DTDNode、DTDTree，BlockNode，BlockTree）和一个管理类（MetaDataManager）。

MetaDataManager 类负责 DTDTree 和 BlockTree 的保存（到磁盘）和（从磁盘）读取。

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。DTDTree 提供由 TagName 到 ID 的双向映射方法。

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。

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。

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。

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。

5.3.1MetaDataManager

1. 导入指定数据集的 DTDTree

static DTDTree* DTDLoader(char* DataSetName);

参数说明：

char* DataSetName：数据集的名称。

OrientX 系统支持有模式约束的 XML 文档，所以每个数据集都有一个模式文档与之相对应，而该模式文档在内存中的表现就是一个 DTDTree，故通过数据集的名称取得与之相关的 DTDTree。

返回值：成功返回模式树的指针，否则返回 NULL。

static DTDTree* DTDLoader();

参数说明：

int setID：OrientX 系统给数据集分配的唯一的 id。

返回值：成功返回模式树的指针，否则返回 NULL。

2. 导入指定数据集的 DataSetBlockTree

static BlockTree* DataSetBlockTreeLoader(char* DataSetName,StorageMode mode);

参数说明：

char* DataSetName：指定数据集的名称。

如果文档存储的模式是聚簇的，需要有个 BlockTree 用来记录聚簇信息。而每个数据集有个数据集相关的 BlockTree，每个文档有个文档相关的 BlockTree。本函数导入的是数据集相关的 BlockTree。

StorageMode mode：指明是 CEBMode 还是 CSBMode。

返回值：成功返回 BlockTree 的指针，否则返回 NULL。

3. 导入指定数据集的 DataSetBlockTree

static BlockTree* DataSetBlockTreeLoader(int setId,StorageMode mode);

参数说明：

int setId：要导入数据集的 id。

StorageMode mode：指明是 CEBMode 还是 CSBMode。

返回值：成功返回 BlockTree 的指针，否则返回 NULL。

5.3.2DTDNode

1. 得到指定 ID 的属性

Attribute* GetAttribute(DTDNodeCode& AID);

参数说明：

DTDNodeCode& AID：指定要读取的属性的编码。

返回值：如果成功返回读取的属性的指针，否则返回 NULL。

Attribute* GetAttribute(char* attName);

参数说明：

char* attName：指定要读取的属性的名称。

返回值：如果成功返回读取的属性的指针，否则返回 NULL。

2. 得到指定 ID 或名字的孩子结点

DTDNode* GetChild(DTDNodeCode& eid);

参数说明：

DTDNodeCode& eid：指定子结点的编码。

返回值：如果成功返回读取的 DTDNode 的指针，否则返回 NULL。

DTDNode* GetChild(char* childName);

参数说明：

char* childName：指定子结点的名称。

返回值：如果成功返回读取的 DTDNode 的指针，否则返回 NULL。

3. 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）

DTDNode* GetRightSibling(DTDNodeCode& eid);

参数说明：

DTDNodeCode& eid：指定右兄弟结点的编码。

返回值：如果成功返回读取的 DTDNode 的指针，否则返回 NULL。

4. 得到指定名字的子孙结点的列表

DTDNodeListItem* GetDescendant(const char* name);

参数说明：

const char* name：得到指定名称的所有后代子孙结点的列表。

返回值：如果成功返回读取的 DTDNode 的指针列表，否则返回 NULL。

5. 得到指定编码的后代结点

DTDNode* GetDescendant(DTDNodeCode& nodeID);

参数说明：

DTDNodeCode& nodeID：指定结点的编码

返回值：如果成功返回读取的 DTDNode 的指针列表，否则返回 NULL。

6. 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* GetDescendant(int index);

参数说明：

int index：指定子孙结点的序号。

返回值：如果成功返回读取的 DTDNode 的指针列表，否则返回 NULL。

7. 得到指定 ID 的祖先结点。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(DTDNodeCode parentEID);

参数说明：

DTDNodeCode parentEID：指定的父结点的 eid。

返回值：如果成功返回读取的 DTDNode 的指针列表，否则返回 NULL。

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

DTDNode* GetAncestor(int index);

参数说明：

int index：如果 index=0，返回父亲结点；如果 index=1，返回父亲的父亲结点

返回值：如果成功返回读取的 DTDNode 的指针列表，否则返回 NULL。

5.3 LogicalElement 说明

LogicalElement 类封装了 OrientX 中底层不同的数据存储方法（基于结点|基于子树），使得上层应用（查询执行、索引以及其他的用户开发应用）可以通过统一的接口对数据进行访问。实际的应用中，LogicalElement 总是绑定到 XML document 中的一个 Element 结点上，它的功能类似于 DOM 中的结点，但是包括了更丰富的内容。下面对 LogicalElement 提供的公共接口进行说明。

1. 返回 element 的 EID（element 的类型在 schema tree 上的标识）

DTDNodeCode GetEid() const;

2. 获得 element 的标签名（tag name ）

char* GetTagName() const;

3. 唯一标识相关接口。

OrientX 中，我们为数据集（DataSet）的每个 XML element 赋予一个唯一的 OID，结合数据集的 ID（setID），可以在系统中唯一的定位一个 element。

void SetSetID(int setID);
void SetOid(unsigned int oid);
int GetSetID()const;
unsigned int GetOid()const;

4. 数据导航相关接口，导航到的 element 绑定到引用参数上。成功，返回 0；否则返回-1。

int GetParent(LogicalElement& node) const;
int GetFirstChild(LogicalElement& node) const;
int GetRightSibling(LogicalElement& node) const;
//导航寻找 eid 为指定 eid 的孩子 element
int GetFirstChild(const DTDNodeCode& eid,LogicalElement& node) const;
//导航寻找 eid 为指定 eid 的右兄弟 element
int GetRightSibling(const DTDNodeCode& eid,LogicalElement& node) const;
//导航寻找 eid 为指定 eid 的所有孩子结点并返回
int GetChildNodes(const DTDNodeCode& eid,NxdbList& eleList) const;
//导航寻找 eid 为指定 eid 的所有后代结点并返回
int GetDescendantNodes(const DTDNodeCode& eid,NxdbList& eleList) const;

5. 获得 element 的 text node

int GetTextNodeNum();
//获得第 index 个 text node，如果失败，返回-1，成功返回 0。（index 的基数是 1）
int GetTextValue(int index, CConstValue& value) const;

6. element 属性的相关函数

//获得属性的个数
int GetAttrNum()const;
//获得第 index 个属性值，在引用参数中返回，成功 0，失败，-1。（index 的基数是 1）
int GetAttrValue(int index, CConstValue& value)const;
//获得 aid 为指定 aid 的属性值，成功返回 0，否则返回-1。
int GetAttrValue(const DTDNodeCode& aid,CConstValue& value)const;

7. 祖先后代关系判断

//父子关系
bool IsParentOf(const LogicalElement& ele)const;
//祖先后代关系
bool IsAncestorOf(const LogicalElement& ele)const;

8. **通过比较 setID 和 OID 判断两个 LogicalElement 是否绑定到同一个 element。**
   bool operator==(const LogicalElement& ele);

9. **调试接口，打印以 element 为根的子树中的所有结点**
   virtual void PrintSubTree(unsigned int offset = 0, FILE* fp = NULL)const;

10. **释放对于 element 的绑定**
    void FreeElement();

5.4 索引管理(Index Manager)

5.4.1 功能介绍

OrientX 系统支持路径索引和值索引两种索引类型。用户可以为数据集的 DTD 中的某个 Element 节点或者 Attribute 节点建立单独的值索引；也可以在一个数据集的某个文档上，或是对某一个特定 Element 节点建立路径索引。

值得注意的是，路径索引中的索引项是按照节点的编码排序的，这样是出于能够方便的执行 Structural Join 操作。

5.4.2 路径索引接口

1. 建立路径索引，提供两种参数形式，一种以文档号作为参数，另一种以文档的节点编号作为参数。成功返回 0，否则错误。
   int CreatePathIndex(int datasetID, int docID);
   int CreatePathIndex(int datasetID, int docID, DTDNodeCode &eID);
2. 删除路径索引，提供两种参数形式，一种以文档号作为参数，另一种以文档的节点编号作为参数。成功返回0，否则错误。
   int DropPathIndex(int datasetID, int docID);
   int DropPathIndex(int datasetID, int docID, DTDNodeCode &eID);
3. 打开路径索引模块，初始化控制信息，在使用索引模块之前，需要调用此方法。
   void Open()
4. 关闭路径索引，将控制信息写回，在使用完索引模块后，应该调用此方法
   void Close()
5. 获得某个路径索引项（一个节点列表），返回eID对应的节点列表。
   int GetHandler (int datasetID,
   int docID, DTDNodeCode &eID, PathIndexHandler &handler);

5.4.3 值索引接口

1. 建立值索引，提供两种参数形式，一种以文档号作为参数，另一种以文档的节点编号作为参数。成功返回 0，否则错误。
   int CreateValueIndex(int datasetID, int docID);
   int CreateValueIndex(int datasetID, int docID, DTDNodeCode &eID);
2. 删除值索引，提供两种参数形式，一种以文档号作为参数，另一种以文档的节点编号作为参数。成功返回0，否则错误。
   int DropValueIndex(int datasetID, int docID);
   int DropValueIndex(int datasetID, int docID， DTDNodeCode &eID);
3. 打开值索引模块，初始化控制信息，在使用索引模块之前，需要调用此方法。
   void Open()
4. 关闭值索引，将控制信息写回，在使用完索引模块后，应该调用此方法
   void Close()
5. 获得某个值索引项（一个节点列表），返回满足op操作的eID对应的节点列表。
   int GetHandler（int datasetID, int docID, DTDNodeCode &eID,
   OperatorType op, CConstVaslue &value,
   ValueNodeHandler &handler);

异常处理

OrientX 系统中的基本异常类为 NxdbException，其中定义了系统中的各种 ErrorCode，它包含一系列的继承类。NxdbException 的类声明如下：

```
class NxdbException: public exception{
public:
    virtual const char* GetExceptionMsg() const {return m_Message;};
    virtual const ErrorCodes GetExceptionCode() const {return m_ExceptionCode;};

    virtual void Print()const
    {
        cout<<"Error"<<m_ExceptionCode;
        if(m_Message[0] == '\0')
            cout<<endl;
        else
            cout<<m_Message<<endl;
    }
protected:
    char m_Message[MSG_LEN];
    ErrorCodes m_ExceptionCode;
};
```

现在所有的 ErrorCodes 定义如下：

```
enum ErrorCodes {

        //General Errors
        GEN_UNKNOWN_ERR=-1000,
        GEN_PARAMETER_ERR,
        GEN_ERASE_NULL_VALUE_ERR,
        GEN_NO_ENOUGH_MEMORY,

        //Buffer Errors
        BUF_FILE_NOT_FOUND_ERR,
        BUF_PAGE_NOT_FOUND_ERR,
        BUF_NO_FREE_BUFFER_ERR,
        BUF_READ_OVERFLOW_ERR,
        BUF_WRITE_OVERFLOW_ERR,
        BUF_READ_ONLY_ERR,
        BUF_BUF_OUT_OF_RANGE_ERR,
        BUF_REPEATEDLY_FREE_ERR,
        BUF_CAN_NOT_FLUSH_ERR,
        BUF_PHYSICAL_PAGE_NOT_EXIST_ERR,
        BUF_PARAMETER_ERR,

        //Storage Manager
        STO_DATASET_FULL_ERR,
        STO_DOC_NUM_FULL_ERR,
        STO_SET_DATAFILE_FULL_ERR,
        STO_DATASET_NOT_FOUND_ERR,
        STO_DOC_NOT_FOUND_ERR,
        STO_DATASET_HAS_EXIST_ERR,
        STO_DOCNAME_HAS_EXIST_ERR,
        STO_SETID_OVERFLOW_ERR,
        STO_LPNO_OVERFLOW_ERR,
        STO_DEL_FILE_FAILED_ERR,
        STO_FILE_PATH_FAILED_ERR,
        STO_CREATE_CTL_DIR_FAILED_ERR,

        //FileManager part
        FILE_DELETE_FILE_FAILED_ERR,
        FILE_CREATE_FILE_FAILED_ERR,
        FILE_FILE_NOT_FOUND_ERR,

        //record cache
        RECCACHE_PARA_ERR,

        //MetaDataManageMent
```

META_APPEND_NULL_ATTR_ERR,
META_APPEND_HOMONYMY_ATTR_ERR,
META_FORMAT_DTDPAGE_FAILED_ERR,
META_LOAD_DTDTREE_FAILED_ERR,
META_FLUSH_DTD_FAILED_ERR,
META_FLUSH_NULL_DTDTREE_ERR,
META_DTDTREE_NOT_IN_CACH_ERR,
META_DTDTREE_CACH_OVERFLOW_ERR,
META_LOAD_DATASET_BLOCKTREE_FAILED_ERR,
META_FLUSH_DATASET_BLOCKTREE_FAILED_ERR,
META_LOAD_DOC_BLOCKTREE_FAILED_ERR,
META_FLUSH_DOC_BLOCKTREE_FAILED_ERR,
META_BLOCK_TREE_CACH_OVERFLOW_ERR,
META_DTDTREE_NULL_ERR,
META_DTDNODE_NULL_ERR,
META_CIRCLE_INDEX_OUTOF_RANGE,
META_PARAMETER_ERR,
META_DTDNODE_VALIDATE_ERR,

//datamanager
DATA_GETRECORD_ERR,
DATA_NEW_OPER_ERR,
DATA_DELETE_OPER_ERR,
DATA_RECORD_ADDRESS_ERR,
DATA_RECORD_CACHE_ERR,
DATA_RECORD_RESTORE_ERR,
DATA_FORMAT_EMPTY_PAGE_ERR,
DATA_RECORD_ADD_ERR,
DATA_RECORD_DELETE_ERR,
DATA_RECORD_READ_ERR,
DATA_RECORD_MODIFY_ERR,
DATA_FIND_IDLE_RECORD_ERR,
DATA_RECONSTRUCT_PAGE_ERR,
DATA_MODIFY_RECORD_OID_ERR,
DATA_PRESTORE_OVERFLOW_ERR,
DATA_ELEMENT_GET_ERR,
DATA_ELEMENT_DELETE_ERR,
DATA_ELEMENT_RELATION_ERR,
DATA_ELEMENT_ATTRIBUTE_ERR,
DATA_APPEND_NULL_ATTR_ERR,
DATA_PARAMETER_ERR,
DATA_ELEMENT_VALIDATE_ERR,//validate

```
//schema parser
DATA_SCHEMA_GENERATE_ERR,

//update
UPDATE_OPTYPE_ERR,

//XQueryExecute Algebra Processing
XQA_VARNAME_OVERFLOW_ERR,
XQA_VARNAME_NOT_DEFINED_ERR,
XQA_CONTEXT_NULL_ERR,
XQA_UNSOPPORTED_FUNCTION_ERR,
XQA_DATASET_NOT_SPECIFIED_ERR,
XQA_INVALID_FORLETCLAUSE_ERR,
XQA_INVALID_CONTEXT_NODE_ERR,
XQA_PARAMETER_ERR,
XQA_QName_INCONSISTENT_ERR,

//Utility:RecordNodeCache
RecNodeCacheOverFlow,
RecNodeNotFound,

//ElementNode
DataInconsistency,

//IndexManager
PathIndexNotExist,
INDEX_OPEN_ERROR,
INDEX_CLOSE_ERROR,
INDEX_INIT_ERROR,
INDEX_CREATE_ERROR,
INDEX_DROP_ERROR,
INDEX_GENERAL_ERROR,

//CodeTransformer
CharacterCodeNotSupport,

//usermanager
UserHasExist,
UserNotExist,

//roleManager
RoleHasExist,
RoleNotExist,
UserRoleHasExist,
```

```
//DataCode
//RegionCode
RegionUpdaterNoEnoughSpace,
RegionEncoderNoEnoughSpace,

//XQueryProcess
NoDataSource,
DataSetNotExist,
StepJoinErr,
EleNameNotexist,

//XPath-parser

//for XQuery parser
NotDeclaredNodeType,
xqpSyntaxError,
NotPrintValueType,
UnknownCharacter,

//for transforming to XPath processor
NULLValueOfPointerParam,
NotSupportedXpathCase,
NotSupportedDataType,
ConflictDataType,
NULLValue,
NULLTerminalValue,
MorePredicates,
FailedToSetPredicates,

//Xquery Execute
XqueryRuntimeError,
XqueryPlanParameterError,
XQueryExecuteParameterError,
XQueryExecuteDataError,
XQueryExecuteRuntimeError,
//treefrog
TreeFrogJump,

//LogicalElement
LOGICAL_ELEMENT_GETELEMENTNODE_ERR,
LOGICAL_ELEMENT_GETEID_ERR,
LOGICAL_ELEMENT_GETTAGNAME_ERR,
LOGICAL_ELEMENT_GETTEXTNODENUM_ERR,
```

```
    LOGICAL_ELEMENT_GETTEXTVALUE_ERR,
    LOGICAL_ELEMENT_GETATTRNUM_ERR,
    LOGICAL_ELEMENT_DEFAULT_ERR

};
```

6. 其他

关于 OrientX 系统的文件列表，类列表，类继承关系，类成员和文件索引，请参照 OrientX 系统的联机帮助文档。

如果您在使用 OrientX 系统过程中，发现系统漏洞，欢迎反馈：OrientX@ruc.edu.cn 我们将尽快给您回复解答。

谢谢您使用我们的 OrientX 系统!