第五届思特奇论坛

OTL技术应用

邓科峰
北京神州数码思特奇信息技术股份有限公司
2008年 04月 25日

神州数码 Digital China 思特奇 SI-TECH
www.si-tech.com.cn

目　　录

- OTL介绍
- OTL“ 流 ” 的概念
- OTL的主要类
- OTL的使用
- OTL的编译
- OTL参考资料

OTL介绍（1）

什么是OTL: OTL是 Oracle, Odbc and DB2-CLI Template Library的缩写，是一个操控关系数据库的C++模板库，它目前几乎支持所有的当前各种主流数据库，如下表所示：

数据库	访问接口	支持版本
Oracle	OCI	OCI7、OCI8、OCI8i、OCI9i、OCI10g
DB2	CLI	DB2 CLI
MS SQL Server、Sybase、Informix、MySQL、Interbase/Firebird、PostgreSQL、SQLite、SAP/DB、TimesTen、MS ACCESS	ODBC	ODBC2.5、ODBC3.x

备注：Oracle和DB2也可以由OTL间接使用ODBC的方式来进行操纵

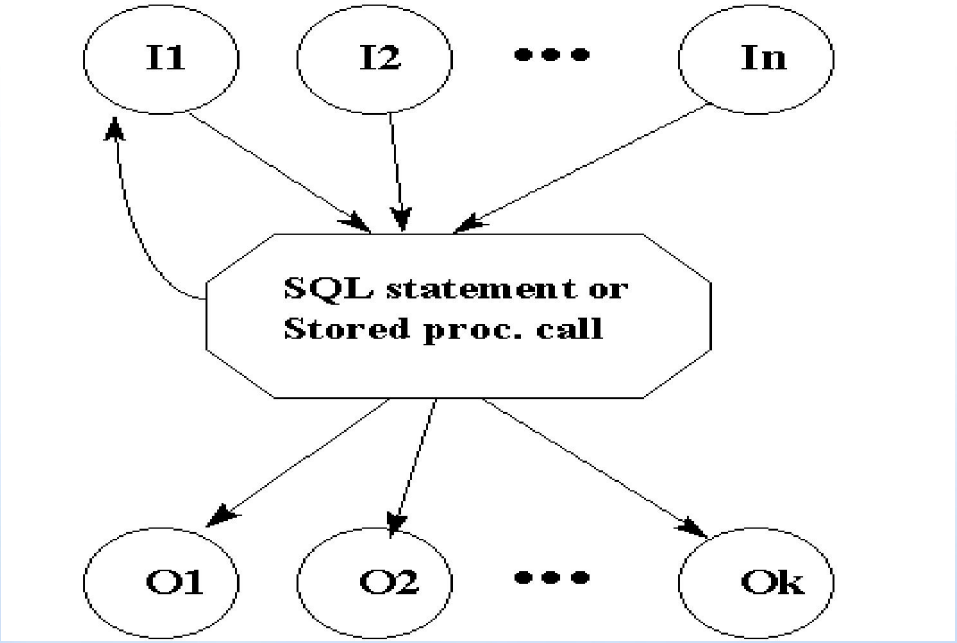
OTL介绍 (2)

OTL的特点：

优点	(1) 跨平台 (2) 运行效率高，与 C语言直接调用数据库 API相当 (3) 开发效率高，使用方便，繁在其内，简在其外，比 ADO.net使用起来更简单，更简洁 (4) 部署容易，不需要 ADO组件，不需要 .net framework 等
缺点	(1) 只能在 C++中使用

OTL“流”的概念(1)

任何 SQL 语句，PL/SQL 块的调用或者是存储过程调用都可以用“流”的 input/output [变量]来表现。



使用 SQL语句、PL/SQL 块 或者是存储过程调用，都可以看作是带输入输出流的黑盒。你可以不关心黑盒的内部工作（只需要依据黑盒的定义），需要关注的是黑盒的输入输出线。

OTL“流”的概念 (2)

Example 1. SELECT语句有用于WHERE子句的标量 input变量。 SELECT 语句同时定义了输出列．潜在的输出列是矢量参数，因为 SELECT语句可能返回多行。

Example 2. INSERT语句往表中写数据， 即它有输入参数 ,其输入参数是标量．UPDATE 语句亦是如此。

Example 3. DELETE 从表中删除行，如需要输入删除条件， DELETE语句需有输入参数，其输入参数是标量。

Example 4.存储过程可能含有 input和 (或)output参数。通常存储过程的参数是标量。有一个特例：存储过程返回一个游标（ cursor) （ORACL) 或者是结果集（MSSQL、 Sybase) ,此时 output参数是矢量。

Example 5.任意一个 PL/SQL块都可能有标量的输入或矢量的输出参数。

OTL“ 流 ” 的概念 (3)

OTL 流 ” 是缓存流

从概念上讲，OTL“ 流 ” 有两个独立的缓存：输入和输出。输入缓存由所有的输入变量共同组成。同样的，输出缓存由所有的输出变量共同组成。

OTL 流 ” 的操作

(1) .OTL“ 流 ” 的操作与 C++流一样 ,通过操作符 “ <<” 和 “ >>” 来进行操作，流的引用在操作符的左边。

s>>variable;

s<<variable;

两个箭头符号表示了数据流动方向：

>> - - 从流到数据容器 (变量)

<< - - 从数据容器 (变量)到流

(2) .OTL流需要用到OTL异常。即任何OTL流都潜在的会抛 otl_exception 类型的异常。为了截获异常并防止程序中断，请用相应 try & catch 块包装 OTL代码段。

OTL的主要类（1）

主要类包括：otl_stream,otl_connect,otl_exception

- otl_stream类

otl_stream类 是OTL“流”的概念的具体表现形式，任何通过输入/输出参数使用SQL语句、PL/SQL 块 或者是存储过程调用，在C++的编程中都能通过otl_stream类来实现。

其构造函数为:

(1)for Oracle 7/8/9/10:

```
otl_stream(const int arr_size, // 流的缓存大小
           const char* sqlstm, // SQL语句或PL/SQL块或存储过程
           otl_connect& db, // OTL数据库连接对象
           const char* ref_cur_placeholder=0, // 游标引用占位符名称
           const char* sqlstm_label=0 // SQL 语句标签);
```

OTL的主要类 (2)

(2)for ODBC/DB2-CLI:

```
otl_stream(const int arr_size,// 流的缓存大小
           const char* sqlstm,// SQL语句或 PL/SQL块或存储过程
           otl_connect& db,// OTL数据库连接对象
           const int implicit_select=otl_explicit_select ,
           const char* sqlstm_label=0//SQL 语句标签 );
```

otl_connect类

otl_connect类封装了一系列有关数据库连接的功能 :建立连接、断开连接、事务提交、事务回滚等等。换言之 ,otl_connect是在 C++编程中创建和使用数据库连接以及进行数据库事务管理的类，主要方法有：

(1)static int otl_initialize(const int threaded_mode=0);

该静态方法的主要功能是初始化OTL数据库环境，程序中第一次建立与数据库的连接之前，必须调用该方法一次，其后再建立与数据库的连接，就不需要调用该方法了。如果程序是在多线程环境下访问数据库 ,参数 threaded_mode需置为 1。另外在多线程环境下访问数据库，不要多个线程操作同一个 otl_connect对象，除非该 ot_connect对象有互斥锁机制。

OTL的主要类 (3)

(2) void rlogon(const char* connect_str,const int auto_commit=0);

该方法的主要功能是建立与数据库的连接。

参数 connect_str是数据库连接配置字符串 ,有两种表达形式

- OTL4.0/OCIx
  - "USER/PASSWORD" (本地数据库)
  - "USER/PASSWORD@TNS_ALIAS" (远程数据库)
- OTL4.0/ODBC和 OTL4.0/DB2_CLI
  - "USER/PASSWORD@DSN"
  - "DSN=value;UID=value;PWD=value"

参数 auto_commit设置数据库事务的提交模式 ,auto_commit设置为 1,表示数据库事务自动提交 ; auto_commit设置为 0,表示数据库事务非自动提交 ,auto_commit缺省为 0。

OTL的主要类 (4)

(3) void logoff(void);
该方法的主要功能是断开与数据库的连接。
(4) void commit(void);
该方法的主要功能是提交数据库事务。
(5) void rollback(void);
该方法的主要功能是回滚数据库事务。

otl_exception类

otl_exception类用于描述OTL操作数据时抛出的异常，有3个主要的成员变量：

(1)unsigned char msg[1000];
该成员变量用于保存存异常的具体错误信息。
(2)char stm_text[2048];
该成员变量用于保存导致发生异常错误的SQL语句。
(3)char var_info[256];
该成员变量用于保存导致发生异常错误的输入/输出变量。

OTL的使用（1）

OTL使用起来也很简单，使用不同的数据库连接，主要是根据需要在程序开始的宏定义来指定的。 OTL是首先根据这个宏定义来初始化数据库连接环境。 OTL中用来区分连接方式的宏定义主要有下面这些：

OTL_ORA7，OTL_ORA8，OTL_ODBC，OTL_DB2_CLI，OTL_ODBC_MYSQL...

不同的宏对应的数据库API，具体说明如下：

宏定义名	说明
OTL_ORA7	for OCI7
OTL_ORA8	for OCI8
OTL_ORA8I	for OCI8i
OTL_ORA9I	for OCI9i. All code that compiles and works under #define OTL_ORA7, OTL_ORA8, and OTL_ORA8I, should work when OTL_ORA9I is used
OTL_ORA10G	for OCI10g. All code that compiles and works under #define OTL_ORA7, OTL_ORA8, OTL_ORA8I, OTL_ORA9I, should work with OTL_ORA10G.
OTL_ORA10G_R2	for OCI10g, Release 2 (Oracle 10.2). All code that compiles and works under #define OTL_ORA7, OTL_ORA8, OTL_ORA8I, OTL_ORA9I, and OTL_ORA10G should work with OTL_ORA10G_R2

OTL的使用 (2)

宏定义名	说明
OTL_DB2_CLI	for DB2 Call Level Interface (CLI)
OTL_INFORMIX_CLI	for Informix Call Level Interface for Unix (when OTL_ODBC_UNIX is enabled).
OTL_IODBC_BSD	for ODBC on BSD Unix, when iODBC package is used
OTL_ODBC	for ODBC
OTL_ODBC_MYSQL	for MyODBC/MySQL. The difference between OTL_ODBC_MYSQL and OTL_ODBC is that transactional ODBC function calls are turned off for OTL_ODBC_MYSQL, since MySQL does not have transactions
OTL_ODBC_ POSTGRESQL	for the PostgreSQL ODBC driver 3.5 (and higher) that are connected to PostgerSQL 7.4 / 8.0 (and higher) servers.
OTL_ODBC_UNIX	for ODBC bridges in Unix
OTL_ODBC_zOS	for ODBC on IBM zOS.
OTL_ODBC_XTG_IBASE6	for Interbase 6.x via XTG Systems' ODBC driver. The reason for introducing this #define is that the ODBC driver is the only Open Source ODBC driver for Interbase. Other drivers, like Easysoft's ODBC for Interbase, are commercial products, and it beats the purpose of using Interbase, as an Open Source.database server.

OTL的使用（3）

SQL使用举例：

```
//“ OtlSqlExample.cpp”
#include <iostream>
using namespace std;
#include <stdio.h>

#define OTL_ORA9I // Compile OTL 4.0/OCI9i,
//#define OTL_UNICODE //Enable Unicode OTL for OCI9i
#include <otlv4.h> // include the OTL 4.0 header file
otl_connect db; // connect object

void insert();void insertConstant();void insertBatch();
void insertNoAutoCommit();
void select();
void update();void updateNoAutoCommit();void del();

int main()
{
   otl_connect::otl_initialize(); // initialize OCI enviroment
   try{
        db.rlogon("dbuser/dbpwd"); // connect to Oracle
```

OTL的使用 (4)

```
otl_cursor::direct_exec
(
 db,
 "drop table person_tab",
 otl_exception::disabled // disable OTL exceptions
 ); // drop table

otl_cursor::direct_exec
(
 db,
 "create table person_tab(age number, name varchar2(30))"
 );  // create table

 insert(); // insert one records into table
 insertConstant();//constand insert sql
 insertBatch(); // insert batch records into table
 insertNoAutoCommit();//insert no  auto commit;
 select(); // select records from table
 update(); // update records in  table
 updateNoAutoCommit(); // update no auto commit
 del(); // delete records from table
}
```

OTL的使用 (5)

```
    catch(otl_exception& p){ // intercept OTL exceptions
        cerr<<p.msg<<endl; // print out error message
        cerr<<p.stm_text<<endl; // print out SQL that caused the error
        cerr<<p.var_info<<endl; // print out the variable that caused
                                // the error
    }
 db.logoff(); // disconnect from Oracle
 return 0;
}
void insert()//插入单条数据数据
{ // create insert stream
  otl_stream o(1, // buffer size
           "insert into person_tab values(:v_age<int>,:v_name<char[31]>)",
              // INSERT statement
              db // connect object
              );
   o<<30;//assigning :v_age=30
   o<<" dengkf" ;//assigning :v_name=" dengkf"
   //char tmp[32];sprintf(tmp,"邓科峰");
   //o<<(unsigned char*)tmp;
   //INSERT automatically executes when all input variables are assigned
}
```

```
void insertConstant()//恒量 insert SQL语句
{ // create insert stream
  otl_stream o(1, // buffer size
            "insert into person_tab values(30,'dengkf')",
               // INSERT statement
               db // connect object
               );

   //INSERT automatically executes when all input variables are
   assigned
}
```

```
void insertBatch()//批量插入数据
{
  // create insert stream
  otl_stream o(10000, // buffer size
           "insert into person_tab
   values(:v_age<int>,:v_name<char[31]>)",
              db // connect object
              );
   char tmp[32];

   for(int i=1;i<=10000;i++){
     sprintf(tmp,"NAME%d",i);
     o<<i;
     o<<tmp;
   }
  //INSERT automatically executes when all input variables are
   assigned.
}
```

```
void insertNoAutoCommit()//插入数据 (事务手动提交 )
{
  // create insert stream
  otl_stream o(10001, // buffer size
           "insert into person_tab
   values(:v_age<int>,:v_name<char[31]>)",
              db // connect object
              );
   o.set_flush(false);//turning off the stream's autoflush flag
   o.set_commit(0);//turning off the stream's autocommit flag
   char tmp[32];

   for(int i=1;i<=10000;i++){
     sprintf(tmp,“ NAME%d” ,i);
     o<<i;
     o<<tmp;
   }
   o.flush();//flushing the stream's buffer
   db.commit();//committing the changes to the database
}
```

OTL的使用 (6)

```
void select()//检索数据
{
  // create select stream
  otl_stream i(50, // buffer size
               "select * from person_tab where name=:v_name<char[31]>",
               // SELECT statement
               db // connect object
               );
  i<<"dengkf"; // assigning :v_name = 8
  // SELECT automatically executes when all input variables are
  // assigned. First portion of output rows is fetched to the buffer

  int  r_age;
  char r_name[31];

  while(!i.eof()){ // while not end-of-data
    i>>r_age;
    i>>r_name;
    cout<<"age="<<r_age<<endl;
    cout<<"name="<<r_name<<endl;
  }
}
```

OTL的使用（7）

```
//修改数据 (事务自动提交 )
void update()
{
  // create update stream
  otl_stream s(1, // buffer size
               "update person_tab set age=:v_age<int> where
                name=:v_name<char[31]>",
               // UPDATE statement
               db // connect object
               );
  s<<31;//assgining :v_age =31
  s<<"dengkf"; //assigning :v_name = 8
  //UPDATE automatically executes when all input variables are assigned.
}
```

```
//修改数据(事务手动提交)
void updateNoAutoCommit()
{
  // create update stream
  otl_stream s(2, // buffer size
               "update person_tab set age=:v_age<int> where age<:v_age2<int>",//
     UPDATE statement
               db // connect object
               );
   s.set_flush(false);
   s.set_commit(0);

  s<<31;//assgining :v_age =31
  s<<2000; //assigning :v_age2 = 2000

  s.flush();
  db.commit();

}
```

OTL的使用（8）

```
//删除数据
void del()
{
  // create delete stream
  otl_stream l(1, // buffer size
               " delete from person_tab where name=:v_name<char[31]>",
               // DELETE statement
               db // connect object
               );
  l<<"dengkf"; //assigning :v_name = 8
  //DELETE automatically executes when all input variables are
   assigned.
}
```

OTL的使用 (9)

SQL使用举例 (常量 SQL使用):

常量 SQL就是不带任何绑定变量的 SQL, OTL通过一个静态方法来操作 :

```
long otl_cursor::direct_exec(otl_connect& db,//OTL数据库对象
                             const char* sqltm,//SQL语句
                             otl_exception_disable=0,//OTL异常忽略标志
                            );
```

返回值 :

- -1,如果 otl_exception_disable被设置成 1, 并且 OTL的底层 API发生错误
- >=0,SQL执行成功 ,返回实际处理成功的记录数 .

o Examples(Oracle)

```
otl_cursor::direct_exec
 (db, // connect object
  "create table person_tab(age number, name varchar2(30))"
  );  // create table

otl_cursor::direct_exec
 (db, // connect object
  "drop table persion_tab", // SQL statement
  otl_exception::disabled // disable OTL exceptions,
                          // in other words, ignore any
                          // database error
  ); // drop table
```

OTL的使用(10)

```
long rpc=otl_cursor::direct_exec
        (db,//connect object
        " delete from persion_tab" );
```

o Examples(ODBC,DB2-CLI)

```
otl_cursor::direct_exec
 (db, // connect object
  "create table person_tab(age numeric, name varchar(30))"
  );  // create table

otl_cursor::direct_exec
 (db, // connect object
  "drop table persion_tab", // SQL statement
  otl_exception::disabled // disable OTL exceptions,
                          // in other words, ignore any
                          //  database error
  ); // drop table

long rpc=otl_cursor::direct_exec
        (db,//connect object
        " delete from persion_tab" );
```

OTL的使用(11)

PL/SQL块使用举例：

```
//“ OtlPlsqlExample.cpp”
#include <iostream>
using namespace std;

#include <stdio.h>
#define OTL_ORA9I // Compile OTL 4.0/OCI9i
#include <otlv4.h> // include the OTL 4.0 header file

otl_connect db; // connect object

void plsql(void)
// invoking PL/SQL block
{
 otl_stream o(50, // buffer size
              "begin "
              " :A<int,inout> := :A+1; "
              " :B<char[31],out> := :C<char[31],in>; "
              "end;",
                 // PL/SQL block
              db // connect object
             );
```

OTL的使用(12)

```
o.set_commit(0); // set stream auto-commit off since
                 // the block does not have any transactions
                 // to commit
 o<<1<<"Test String1"; // assigning :A = 1, :C = "Test String1"
 o<<2<<"Test String2"; // assigning :A = 2, :C = "Test String2"
 o<<3<<"Test String3"; // assigning :A = 3, :C = "Test String3"

 o.flush(); // executing PL/SQL block 3 times

 int a;
 char b[32];

 while(!o.eof()){ // not end-of-data
  o>>a>>b;
  cout<<"A="<<a<<", B="<<b<<endl;
 }

}
```

OTL的使用(15)

```
int main()
{
 otl_connect::otl_initialize(); // initialize OCI environment
 try{

  db.rlogon(“ dbuser/dbpwd"); // connect to Oracle

  plsql();//invoking PL/SQL block

 }

 catch(otl_exception& p){ // intercept OTL exceptions
  cerr<<p.msg<<endl; // print out error message
  cerr<<p.stm_text<<endl; // print out SQL that caused the error
  cerr<<p.var_info<<endl; // print out the variable that caused the
                          // error
 }

 db.logoff(); // disconnect from Oracle

 return 0;
}
```

OTL的使用(16)

输出：

```
A=2, B=Test Stri ng1
A=3, B=Test String2
A=4, B=Test String3
```

常量 PL/SQL块的使用与常量 SQL的使用类似，在此不再赘述。

OTL的使用(17)

存储过程使用举例：

```
#include <iostream>
using namespace std;

#include <stdio.h>
#define OTL_ORA9I // Compile OTL 4.0/OCI9i
#include <otlv4.h> // include the OTL 4.0 header file

otl_connect db; // connect object

void stored_proc(void)
{
  otl_stream o(1, // buffer size should be equal to 1 in case of
                  // stored procedure call
               "begin my_proc("
               ":a,"
               ":b, "
               ":c "
               ");end;",
               // stored procedure call
               db // connect object
               );
```

OTL的使用(18)

```
    o.set_commit(0); // set stream auto-commit off since
                     // the stream does not generate transaction
    o<<1<<" Test String1" ;// assigning :a = 1, :c = "Test String1"

    o.flush();
    int a2;
    char b2[31];

    o>>a2>>b2;
    cout<<"A="<<a2<<endl;
    cout<<"B="<<b2<<endl;
}
```

OTL的使用(19)

```
int main()
{
  otl_connect::otl_initialize(); // initialize environment
  try{
    db.rlogon(“ dbuser/dbpwd");

    otl_cursor::direct_exec
    (
     db,
     "CREATE OR REPLACE PROCEDURE my_proc "
     " (A IN OUT NUMBER, "
     " B OUT VARCHAR2, "
     " C IN VARCHAR2) "
     "IS "
     "BEGIN "
     " A := A+1; "
     " B := C; "
     "END;"
     ); // 也可以直接用代码创建来测试用的过程
```

OTL的使用(20)

```
    stored_proc(); // invoking stored procedure
  }
  catch(otl_exception& p)
 {
    // intercept OTL exceptions
    cerr<<p.msg<<endl; // print out error message
    cerr<<p.stm_text<<endl;// print out SQL that caused the error
    cerr<<p.var_info<<endl;// print out the variable that caused th
                           // error
  }
  db.logoff(); // disconnect from the data source
  return 0;
}
```

常量存储过程的使用与常量 SQL的使用类似，在此不再赘述。

OTL的编译(1)

在编译OTL的程序时，需要使用到相应的数据库API，这就要程序在编译时联接lib库文件，不同的数据库对应的lib文件所在位置各不相同，下面是分别在windows与Unix下的数据库API所需要的头文件及lib文件所在的位置列表：

API	API header files for Windows	API libraries for Windows
OCI7	<ORACLE_HOME>\oci\include	<ORACLE_HOME>\oci\lib\<compiler_specific>\ociw32.lib
OCI8	<ORACLE_HOME>\oci\include	<ORACLE_HOME>\oci\lib\<compiler_specific>\oci.lib
OCI8i	<ORACLE_HOME>\oci\include	<ORACLE_HOME>\oci\lib\<compiler_specific>\oci.lib
OCI9i	<ORACLE_HOME>\oci\include	<ORACLE_HOME>\oci\lib\<compiler_specific>\oci.lib
OCI10g	<ORACLE_HOME>\oci\include	<ORACLE_HOME>\oci\lib\<compiler_specific>\oci.lib
ODBC	Normally, in one of the C++ compiler system directories, no need to include explicitly.	Normally, in one of the C++ compiler system directories: odbc32.lib
DB2 CLI	<DB2_HOME>\include	<DB2_HOME>\lib\db2api.lib <DB2_HOME>\lib\db2cli.lib

OTL的编译(2)

API	API header files for Unix	API libraries for Unix
OCI7	-I$(ORACLE_HOME)/rdbms/demo -I$(ORACLE_HOME)/rdbms/public	-L$(ORACLE_HOME)/lib/ -lclntsh
OCI8	-I$(ORACLE_HOME)/rdbms/demo -I$(ORACLE_HOME)/rdbms/public	-L$(ORACLE_HOME)/lib/ -lclntsh
OCI8i	-I$(ORACLE_HOME)/rdbms/demo -I$(ORACLE_HOME)/rdbms/public	-L$(ORACLE_HOME)/lib/ -lclntsh
OCI9i	-I$(ORACLE_HOME)/rdbms/demo -I$(ORACLE_HOME)/rdbms/public	-L$(ORACLE_HOME)/lib/ -lclntsh
OCI10g	-I$(ORACLE_HOME)/rdbms/demo -I$(ORACLE_HOME)/rdbms/public	-L$(ORACLE_HOME)/lib/ -lclntsh
ODBC	ODBC bridge specific	ODBC bridge specific
DB2 CLI	-I/<DB2_HOME>/sqllib/include	-L/<DB2_HOME>/sqllib/lib -ldb2

OTL参考资料

OTL最新版本为4.0,参见http://otl.sourceforge.net/，下载地址http://otl.sourceforge.net/otlv4_h.zip,目前提供有377个使用范例可参考，下载地址http://otl.sourceforge.net/otl4_examples.zip。

共同发展　共享成果　共同成功！

神州数码 Digital China 思特奇 SI-TECH
www.si-tech.com.cn