Module07-04
数据库开发：C++ OTL

数据库开发 – C++ OTL

- SQL 语句
- Oracle PL/SQL
- MySQL Procedure
- C++ OTL
- 数据库建模工具

- 关于 OTL
  - OTL： Oracle、 ODBC and DB2-CLI Template Library
  - 该类库目前支持几乎所有主流的关系数据库管理系统：
    - Oracle，通过 OCI 原生 API，版本自 Oracle7.3 ~ 11g
    - DB2，通过 DB2-CLI(Call Level Interface) 原生 API
    - MS SQL Server，通过 SNAC 原生 API
    - Informix，通过 Informix-CLI 原生 API
    - 通过 ODBC(Open DataBase Connectivity) 支持：Sybase、 MySQL、 PostgreSQL、 SQLLite、 Firebird 等等众多数据库
  - 该类库基于 C++ Template，整个类库放在一个单一的头文件中，其中由几个核心的类组成： otl_stream, otl_connect, otl_exception, otl_long_string 等

- OTL 类简介
  - 类 otl_stream：简单而言，该类的对象用于准备 sql 语句、 PL/SQL 语句块和相关参数，同时用于容纳各种操作的返回结果集
  - 类 otl_connect：代表与数据库服务器之间的会话，其功能：与数据库服务器建立连接、关闭连接、管理事务（ Transaction ）
  - 类 otl_exception：用于收集数据库服务器返回的出错信息

接下来将通过一个编程指南来展示 OTL 的使用。

OTL 编程指南 - 准备工作

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 准备工作

- 数据库安装
  - 关于 Oracle 安装，请参考：
    - 《 Practical - Oracle XE Installation.pdf 》
  - 关于 Oracle 和 MySQL 开发所涉及的共享库以及头文件设置，请参考：
    - 《 Practical - C++ Database Dev Environment.pdf 》

OTL 编程指南 - 连接数据库

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 连接数据库

- 连接 Oracle 数据库 - 示例

```
#include <iostream>
#define OTL_ORA10G_R2 // Compile OTL 4.0/OCI10gR2  #1
#include "otlv4.h" // include the OTL 4.0 header file

int main() {
    using namespace std;
    otl_connect db; // connect object    #2
    otl_connect::otl_initialize(); // initialize OCI environment
#3

    try {
        // 以下 用户名/口令 按实际情况而定
        db.rlogon("dbdev/q1w2e3"); // connect to Oracle  #4
        cout << "Connected to oracle xe 10g." << endl;
    } catch (otl_exception& p) { // intercept OTL exceptions
        cerr << p.msg << endl; // print out error message
    }
    db.logoff(); // disconnect from Oracle   #5
}
```

OTL 编程指南 - 连接数据库

- 连接 Oracle 数据库 - 代码解读
  - #1： 表示使用 OCI 10gR2， 该 #define 必须在 #include "otlv4.h" 之前。

    另外一种方式是不定义该宏，在 otlv4.h 内部打开该宏也可，但不建议这种做法
  - #2、 #3： 创建数据库连接对象、初始化之
  - #4： 发起到 Oracle 数据库服务器的连接
  - #5： 关闭与 Oracle 数据库服务器的连接
- 编译
  - `g++ -o otl_oracle_test otl_oracle_test.cpp -lclntsh -I$ORACLE_HOME/rdbms/public`

OTL 编程指南 - 连接数据库

- 连接 MySQL 数据库 - 示例

```
#include <iostream>
#define OTL_ODBC // Compile OTL 4.0/ODBC    #1
// The following #define is required with MyODBC 3.51.11+
#define OTL_ODBC_SELECT_STM_EXECUTE_BEFORE_DESCRIBE
#define OTL_ODBC_UNIX // uncomment this line if UnixODBC is used
#include "otlv4.h" // include the OTL 4.0 header file
int main() {
    using namespace std;
    otl_connect db; // connect object
    otl_connect::otl_initialize(); // initialize ODBC environment
    try {
        db.rlogon("UID=root;PWD=abcdef;DSN=attendance_sys"); // 或
        // db.rlogon("root/abcdef@attendance_sys");
        cout << "Connected to MySQL." << endl;
    } catch (otl_exception& p) { // intercept OTL exceptions
        cerr << p.msg << endl; // print out error message
    }
    db.logoff(); // disconnect from Oracle
}
```

OTL 编程指南 - 连接数据库

- 连接 MySQL 数据库 - 代码解读
  - #1 ： 表示使用 ODBC ， 该 #define 必须在 #include "otlv4.h" 之前。

    由于 OTL 目前只支持通过 ODBC 方式连接到 MySQL 数据库服务器，而不是象 Oracle 一样使用 OCI 原生 API
- 编译
  - `g++ -o otl_mysql_test otl_mysql_test.cpp -lmyodbc`

OTL 编程指南 - 执行 SQL 语句

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 执行 SQL 语句

- Static SQL 和 Dynamic SQL
  - OTL 的 otl_stream 支持 2 种类型的 SQL 语句：
    - Static SQL，一般的字符串方式，如：
      - `const char* sql = "select * from Employee";`
      - ```
        char buf[128];
        sprintf(buf,
            "select * from Employee where empl_id = '%s'",
            id.c_str());
        ```
    - Dynamic SQL，包含占位符的 SQL 语句，如：
      - ```
        const char* = "select * from Department "
                "where depart_id = :did<char(6)>";
        ```
      - 上面的 `:did<char(6)>` 为占位符，表示还需为该语句传入一个参数，来为这个占位符填值

OTL 编程指南 - 执行 SQL 语句

- Constant SQL 和 Dynamic SQL（续）
  - 2 种类型的 SQL 语句的差异：
    - Static SQL：
      - 简单明了，在字符串创建时即形成一条完整的 SQL 语句
      - 但很多情况下需进行字符串串接操作，如果使用 C-Style 字符串操作容易引起数组越界等问题
      - 对于日期型、字符串型的字段，必须被包含在 '' 之间
    - Dynamic SQL：
      - 没有字符串越界的问题，也不存在在日期、字符串字段的 ' ' 问题
      - 不过后续还需通过 otl_stream 的 << 操作符为占位符填值
      - 动态 SQL 一次构造，可以被多次执行

OTL 编程指南 - 执行 SQL 语句

- 按主键查询
  - 使用 Static SQL

```
struct Department {
    string id;
    string name;
    string location;
};

void findDepartmentC(otl_connect& db, const string& id,
        Department& d) {
    char sql[64] = "";
    sprintf(sql, "select * from Department "
        "where depart_id = '%s'", id.c_str());
    otl_stream stream(10, sql, db);

    if (!stream.eof()) {
        stream >> d.id >> d.name >> d.location;
    }
}
```

OTL 编程指南 - 执行 SQL 语句

- 按主键查询
  - 使用 Dynamic SQL

```
void findDepartmentD(otl_connect& db, const string& id,
        Department& d) {
    otl_stream stream(10,
            "select * from Department "
            "where depart_id = :did<char(6)>", db);
    stream << id; // 为占位符 :did<char(6)> 填值

    if (!stream.eof()) {
        stream >> d.id >> d.name >> d.location;
    }
}
```

OTL 编程指南 - 执行 SQL 语句

- 接收更多记录

```
void findDepartments(otl_connect& db, vector<Department>& ds) {
    otl_stream stream(50, "select * from Department", db);
    Department d;
    while (!stream.eof()) {  // 在一个循环中接收数据，遇到没有数据时停止
        stream >> d.id >> d.name >> d.location;
        ds.push_back(d);
    }
}
```

OTL 编程指南 - 执行 SQL 语句

- 使用 otl_stream 的几个注意点
  - 1， otl_stream(...) 的第一个参数： buffer size，该参数用于调控 sql 语句执行的性能，该参数表示字执行一次 sql 语句最多从数据库接收的记录数
  - 2，要让 otl_stream 对象能与 std::string 一起使用，必须在 #include "otlv4.h" 语句之前定义宏 OTL_STL

OTL 编程指南 - 执行 SQL 语句

- 插入数据

```
void addDepartments(otl_connect& db,
                    const vector<Department>& ds) {
    otl_stream stream(1, // 对于插入语句为1
            "insert into Department values "
            "(:did<char(6)>,:dname<char(32)>,:dloc<char(45)>)",
            db);
    stream << "ABC" << "ABC Department" << "Suzhou"; // 一次执行

    for (size_t i = 0; i < ds.size(); ++i) // 多次执行
        stream << ds[i].id << ds[i].name << ds[i].location;
}
```

delete 和 update 操作类似 insert

OTL 编程指南 - 执行 SQL 语句

- 执行 DDL 语句
  - 执行 DDL 语句，如 create table 、 drop table 或 truncate table 等操作，可以使用 otl_cursor::direct_exec() 静态函数，如：

```
otl_cursor::direct_exec(db,
        "truncate table Employee",
        otl_exception::disabled);
```

  - otl_cursor::direct_exec() 函数不仅可以执行 DDL 语句，也可以执行 DML 或 DCL 语句

OTL 编程指南 - 异常细节

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 异常细节

- 类 otl_exception
  - otl_exception 类有 4 个数据成员：
    - msg： 错误信息
    - stm_text： 引起错误的 sql 语句
    - sqlstate： SQLSTATE 信息
    - var_info： 引起错误的参数的信息

OTL 编程指南 - 异常细节

- 类 otl_exception 示例

```
int main() {
    otl_connect db; // connect object
    otl_connect::otl_initialize(); // initialize OCI environment

    try {
        // 以下 用户名/口令 按实际情况而定
        db.rlogon("dbdev/q1w2e3"); // connect to Oracle
        cout << "Connected to oracle xe 10g." << endl;
        addDepartments(db, dd);
    } 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.sqlstate << endl; // print out SQLSTATE message
        cerr << p.var_info << endl; // print out the variable
that caused the error
    }
    db.logoff(); // disconnect from Oracle
}
```

OTL 编程指南 - 调用存储过程

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 调用存储过程

- PL/SQL 方式调用存储过程

```
void callProc(otl_connect& db) {
    // Oracle PL/SQL语句块方式
    otl_stream stream(1,
                      "begin"
                      "  test_proc(:a1<char(6),in out>,"
                      "    :a2<char[16],in>,:a3<int,out>);"
                      "end;",
                      db);
    stream.set_commit(0);
    stream << "RND" << "Huang zhong";
    int res;
    stream >> res;
    cout << "test_proc result: " << res << endl;
}
```

OTL 编程指南 - 调用存储过程

- ODBC 方式调用存储过程

```
void callProc(otl_connect& db) {
    // ODBC方式
    otl_stream stream(1,
                      "{call test_proc(:a1<char[6],inout>,"
                      ":a2<char[16],in>,:a3<int,out>)}",
                      db);
    stream << "RND" << "Huang zhong";
    int res;
    stream >> res;
    cout << "test_proc result: " << res << endl;
}
```

注： OTL 针对 MySQL ODBC 方式调用含有 INOUT 或 OUT 的参选的存储过程有问题，在目前的 OTL 版本中尽量避免，调用带 INOUT 或 OUT 形式的参数的过程。

OTL 编程指南 - 管理事务

- OTL 编程指南：
  - 准备工作
  - 连接数据库
  - 执行 SQL 语句
  - 异常细节
  - 调用存储过程
  - 管理事务

OTL 编程指南 - 管理事务

- 处理事务的方式

```
void transactionTest(otl_connect& db) {
    try {
        db.auto_commit_off();
        // operation #1
        // ...
        // operation #2
        // ...

        // 正确执行到这里，提交事务
        db.commit();
    } catch (otl_exception& p) {
        cerr << p.msg << endl;
        // 发生异常，需回滚事务
        db.rollback();
    }
}
```

C++ OTL - 更多资源

- OTL 附带的 Examples 全面而详尽的展示了 OTL 的用法。
- 除 OTL 外，还有一些 C++ Database 编程类库，如：
  - SOCI - The C++ Database Access Library
  - MySQL++
  - DTL (Database Template Library)
  - …

  上述类库，总体而言，不如 OTL 支持的数据库范围广泛、使用方便。