MySQL\_C\_API编程实践-专题讲座

written by 王保明

# 1MySQL C API helloWorld

## 1.1有关mysqlclient库

|  |
| --- |
| C APIs包含在*mysqlclient*库文件当中，与MySQL的源代码一块发行，用于连接到数据库和执行数据库查询 |

## 1.2helloworld应用程序编写

|  |
| --- |
| int main()  {  int ret = NULL;    MYSQL mysql;  MYSQL \*connect;  //MYSQL\_RES \*res;  //MYSQL\_ROW row;  //char \*query;  //int t, r;    /\*  if (mysql\_library\_init(0, NULL, NULL)) {  fprintf(stderr, "could not initialize MySQL library\n");  exit(1);  }  \*/    mysql\_init(&mysql);    /\*  MYSQL \*mysql\_real\_connect(MYSQL \*mysql, const char \*host, const char \*user,  const char \*passwd, const char \*db, unsigned int port, const char \*unix\_socket,  unsigned long client\_flag)  \*/  /\*  unsigned int mysql\_errno(MYSQL \*mysql);  const char \*mysql\_error(MYSQL \*mysql);  \*/    connect = mysql\_real\_connect(&mysql, "localhost", "root", "123456", "mydb2", 0, NULL, 0 );  if (connect == NULL)  {  ret = mysql\_errno(connect);  printf("func mysql\_real\_connect() err\n");  return ret;  }  else  {  printf(" ok......\n");  }    printf("connect:%d &mysql:%d \n",connect, &mysql );    mysql\_close(connect);    //mysql\_library\_end();  } |

## 1.3helloworld应用程序调试

|  |
| --- |
| **MySQL开发环境熟悉**   * mysq的开发头文件目录为 /usr/include/mysql。 * mysq的开发 动态库 * gcc -o dm01\_hello dm01\_hello.c -I/usr/include/mysql -L/usr/lib64/mysql –lmysqlclient * 典型错误1   /\*  /usr/lib64/mysql/libmysqlclient.a(dso\_dlfcn.o): In function `dlfcn\_globallookup':  dso\_dlfcn.c:(.text+0x31): undefined reference to `dlopen'  dso\_dlfcn.c:(.text+0x44): undefined reference to `dlsym'  dso\_dlfcn.c:(.text+0x4f): undefined reference to `dlclose'  //libmysqlclient.a dlfcn\_globallookup undefined reference to `dlopen'  \*/   * 典型错误2   /\*  dm01\_hello.c:49: 警告：初始化将指针赋给整数，未作类型转换  /usr/lib64/mysql/libmysqlclient.a(net\_serv.cc.o):(.data.DW.ref.\_\_gxx\_personality\_v0[DW.ref.\_\_gxx\_personality\_v0]+0x0): undefined reference to `\_\_gxx\_personality\_v0'  /usr/lib64/mysql/libmysqlclient.a(password.c.o): In function `scramble\_323':  /pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: undefined reference to `floor'  /pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: undefined reference to `floor'  /pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: undefined reference to `floor'  /pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: undefined reference to `floor'  /pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: undefined reference to `floor'  /usr/lib64/mysql/libmysqlclient.a(password.c.o):/pb2/build/sb\_0-12734909-1406113305.48/rpm/BUILD/mysqlcom-pro-5.6.20/mysqlcom-pro-5.6.20/sql/password.c:184: more undefined references to `floor' follow  /usr/lib64/mysql/libmysqlclient.a(my\_getsystime.c.o): In function `my\_getsystime':  \*/   * 典型错误3   dm01\_hello.c:70: 警告：初始化将指针赋给整数，未作类型转换  //导致没有链接 libstdc++.so  /usr/lib64/mysql/libmysqlclient.a(net\_serv.cc.o):(.data.DW.ref.\_\_gxx\_personality\_v0[DW.ref.\_\_gxx\_personality\_v0]+0x0): undefined reference to `\_\_gxx\_personality\_v0' |
| **MySQL开发环境GCC写法**   * gcc -o dm01\_hello dm01\_hello.c -I/usr/include/mysql -L/usr/lib64/mysql -lmysqlclient -ldl -lpthread -lm -lrt -lstdc++ |
| **MySQL开发环境makefile写法**  .PHONY:clean all  CC=gcc  CFLAGS=-Wall -g  LFLAGS=-L/usr/lib64/mysql -lmysqlclient -ldl -lpthread -lm -lrt -lstdc++  BIN=dm01\_hello dm02\_query  all:$(BIN)  %.o:%.c  $(CC) $(CFLAGS) -c $< -o $@    dm01\_hello:dm01\_hello.o  $(CC) $(CFLAGS) $^ $(LFLAGS) -o $@      dm02\_query:dm02\_query.o  $(CC) $(CFLAGS) $^ $(LFLAGS) -o $@  clean:  rm -f \*.o $(BIN) |

# 2MySQL C API CRUD

## 2.1MYSQL C API的常用函数

**数据类型**

|  |
| --- |
| * MYSQL   + 该结构代表1个数据库连接的句柄。几乎所有的MYSQL函数均使用它。 * MYSQL\_RES   + 该结构代表返回行的查询结果 * MYSQL\_ROW   + 这是1行数据的”类型安全”表示。 * MYSQL\_FIELD   + 该结构包含关于字段的信息，如字段名、类型和大小等 * MYSQL\_FIELD\_OFFSET   + 这时MYSQL字段列表偏移量的”类型安全”表示 * my\_ulonglong   + 用于行数以及mysql\_affected\_rows()、mysql\_num\_rows()和mysql\_insert\_id()的类型 |

**编程步骤**

|  |
| --- |
| * 1 通过调用mysql\_library\_init(),初始化MYSQL库 * 2 通过调用mysql\_init()初始化连接处理程序，并通过调用mysql\_real\_connect()连接到服务器 * 3 发出SQL语句并处理其结果 * 4 通过调用mysql\_close()，关闭与MYSQL服务器的连接 * 5 通过调用mysql\_library\_end()，结束MYSQL库的使用 |

**数据库连接**

|  |
| --- |
| * 初始化一个连接句柄结构   #include <mysql.h>  MYSQL \*mysql\_init(MYSQL \*);   * 和数据库建立物理连接   MYSQL \*mysql\_real\_connect(MYSQL \*connection,  const char \*server\_host,const char \*sql\_user\_name,  const char \*sql\_password,const char \*db\_name,  unsigned int port\_number,const char  \*unix\_socket\_name,unsigned int flags); |
| **连接参数** |
| Int mysql\_options(MYSQL \*connection,enum  option\_to\_set,const char \*argument) |
| 2009-2-20 16-24-03 |
|  |

**错误处理**

|  |
| --- |
| * unsigned int mysql\_errno(MYSQL \*connection);   对于由mysql指定的连接，该函数返回最近调用的API函数的错误代码，该函数调用可能成功也可能失败。”0”返回值表示未出现错误   * char \*mysql\_error(MYSQL \*connection)   如果想获得错误的文本信息，可以调用该函数 |

**执行SQL语句**

|  |
| --- |
| * int mysql\_query(MYSQL \*mysql,const char \*query) * 功能描述：   + 执行由”Null终结的字符串”查询指向的SQL查询。正常情况下，字符串必须包含1条SQL语句，而且不应为语句添加终结分号或”\g”   + 不能用于包含二进制数据的查询，应使用mysql\_real\_query来完成 |
| **对于操纵语句** |
| * My\_ulonglong mysql\_affected\_rows(MYSQL \*mysql)   返回上次操纵语句所涉及到记录的行数   * 对于UPDATE、DELETE或INSERT语句，可在mysql\_query()后立刻调用。对于SELECT语句，该函数的工作方式与mysql\_num\_rows()类似 |
| **对于查询语言** |
| * MYSQL\_RES \*mysql\_store\_result(MYSQL \*mysql) * 功能说明：   + 对于成功检索了数据的每个查询，必须调用mysql\_store\_result()或mysql\_use\_result()   + 该函数将查询的全部结果读取到客户端，分配1个MYSQL\_RES结构，并将结果置于该结构中   + 可以通过mysql\_num\_rows来找出结果集中的行数 |

**事务处理**

|  |
| --- |
| * my\_bool mysql\_autocommit(MYSQL \*mysql,my\_bool mode)   + 函数说明:如果模式为1,启动autocommit模式；如果模式为0，禁止autocommit模式 * my\_bool mysql\_commit(MYSQL \*mysql)   + 提交当前事务 * my\_bool mysql\_rollback(MYSQL \*mysql)   + 回滚当前事务 |

**处理结果集**

|  |
| --- |
| * MYSQL\_ROW mysql\_fetch\_row(MYSQL\_RES \*result) * 功能描述   + 检索结果集的下一行   + 行内值的数目由mysql\_num\_fields(result)给出   + 可以调用mysql\_fetch\_lengths()来获得行中字段值的长度 |
| * void mysql\_data\_seek(MYSQL\_RES \*result,my\_ulonglong offset) * 功能描述：   + 在查询结果集中寻找任意行。偏移值为行号。   + 该函数要求结果集结构包含查询的所有结果 * MYSQL\_ROW\_OFFSET mysql\_row\_tell(MYSQL\_RES \*result)   + 返回光标的当前位置 * MYSQL\_ROW\_OFFSET mysql\_row\_seek(MYSQL\_RES \*result,MYSQL\_ROW\_OFFSET offset)   + 将行光标置于查询结果集中的任意行 |
| **每次接收一行数据** |
| * MYSQL\_RES \*mysql\_use\_result(MYSQL \*mysql) * 功能说明：   + 不像mysql\_store\_result()那样把结果集实际读取到客户端。它必须通过对mysql\_fetch\_row()的调用，对每一行分别进行检索。   + Mysql\_store\_result()相比，速度更快而且使用的内存更少 |

**多查询执行的C API处理**

|  |
| --- |
| * 支持在单个字符串中指定的多语句的执行。要想与给定的连接一起使用该功能，打开连接时，必须将标志参数中的CLIENT\_MULTI\_STATEMENTS选项指定给mysql\_real\_connect()。也可以通过调用mysql\_set\_server\_option(MYSQL\_OPTION\_MULTI\_STATEMENTS\_ON)，为已有的连接设置它 |
| 2009-2-20 17-37-34 |
|  |

**详细api列表**

|  |
| --- |
| MYSQL \*mysql\_init(MYSQL \*mysql)   * 分配或初始化与mysql\_real\_connect()相适应的MYSQL对象。 * 如果mysql是NULL指针，该函数将分配、初始化、并返回新对象。否则，将初始化对象，并返回对象的地址。 * 如果mysql\_init()分配了新的对象，当调用mysql\_close()来关闭连接时。将释放该对象。 |
| void mysql\_close(MYSQL \*mysql)   * 关闭前面打开的连接。 * 如果句柄是由mysql\_init()或mysql\_connect()自动分配的，mysql\_close()还将解除分配由mysql指向的连接句柄。 |
| mysql\_real\_connect函数   * MYSQL \*mysql\_real\_connect(MYSQL \*mysql, const char \*host, const char \*user, const char \*passwd, const char \*db, unsigned int port, const char \*unix\_socket, unsigned long client\_flag) * 函数功能:尝试与运行在主机上的MySQL数据库引擎建立连接 * connect = mysql\_real\_connect(&mysql, "localhost", "root", "123456", "mydb2", 0, NULL, 0 ); |

查询api

mysql\_query()函数和mysql\_real\_query()函数

|  |
| --- |
| * int mysql\_query(MYSQL \*mysql, const char \*query) |
| * int mysql\_real\_query(MYSQL \*mysql, const char \*query, unsigned long length) * 对于包含二进制数据的查询，必须使用mysql\_real\_query()而不是mysql\_query()， * mysql\_real\_query()比mysql\_query()快，这是因为它不会在查询字符串上调用strlen()。 |
|  |

获取结果集api

mysql\_store\_result()函数和mysql\_use\_result

|  |
| --- |
| * MYSQL\_RES \*mysql\_store\_result(MYSQL \*mysql) * mysql\_store\_result()将查询的全部结果读取到客户端，分配1个MYSQL\_RES结构，并将结果置于该结构中； |
| * 可调用mysql\_num\_rows()来找出结果集中的行数。可以调用mysql\_fetch\_row()来获取结果集中的行，或调用mysql\_row\_seek()和mysql\_row\_tell()来获取或设置结果集中的当前行位置。 |
| * 一旦完成了对结果集的操作，必须调用mysql\_free\_result()。 |
|  |
| MYSQL\_RES \*mysql\_use\_result(MYSQL \*mysql) **每次接收一行数据** |
| * 功能说明： * 不像mysql\_store\_result()那样把结果集实际读取到客户端。它必须通过对mysql\_fetch\_row()的调用，对每一行分别进行检索。 * Mysql\_store\_result()相比，速度更快而且使用的内存更少 |

## 2.2调用思路分析

1 简介

访问MySQL服务器，这需要使用mysqlclient库，MySQL的大多数客户端API（除了

Java和.NET）都是通过这个库来和MySQL服务器通讯的，而这个库正是用C编写的。

2代码编写和编译

3连接MySQL服务器

初始化一个MYSQL结构，该结构在几乎所有的MySQL C API函数（除了预处理语句相关的函数）中都会用到。MYSQL \*mysql\_init(MYSQL \*mysql)

连接MySQL服务器使用MYSQL \*mysql\_real\_connect(MYSQL \*mysql, const char \*host, const char \*user, const char \*passwd,const char \*db, unsigned int port, const char \*unix\_socket, unsigned long client\_flag);

也可另一种方式来进行连接。先用MYSQL\_READ\_DEFAULT\_FILE作为选项名来调用mysql\_options(), 再调用mysql\_real\_connect()来连接服务器;

mysql\_options(\*mysql, MYSQL\_READ\_DEFAULT\_FILE, "my.cnf")

mysql\_real\_connect(\*mysql, NULL, NULL, NULL, NULL, 0, NULL, 0)

说明：my.cnf文件中记录了连接MySQL服务器所需的各项参数（地址，端口，用户名，密

码，数据库，字符集，Unix Socket等）。这样可以灵活的修改连接参数而不必重新编译程序。

要关闭连接，则调用mysql\_close()

void mysql\_close(MYSQL \*mysql)

4执行查询

1）调用mysql\_query()来执行SQL语句，如果语句中包含二进制数据，则需要调用

mysql\_real\_query()

int mysql\_query(MYSQL \*mysql, const char \*stmt\_str)

int mysql\_real\_query(MYSQL \*mysql, const char \*stmt\_str, unsigned long length)

2）若执行的是UPDATE, DELETE或INSERT语句，则可通过mysql\_affected\_rows()获知

受影响的记录数。my\_ulonglong mysql\_affected\_rows(MYSQL \*mysql) 。

还可以通过mysql\_insert\_id()来获取由最近的UPDATE或INSERT语句生成的自增值。

my\_ulonglong mysql\_insert\_id(MYSQL \*mysql)

3）若执行的是SELECT语句，则有两种方式来获取结果集。

3-1）一种方式是通过mysql\_store\_result()将整个结果集全部取回来。

MYSQL\_RES \*mysql\_store\_result(MYSQL \*mysql)

3-2）另一种方式则是调用mysql\_use\_result()初始化获取操作，但暂时不取回任何记录。MYSQL\_RES \*mysql\_use\_result(MYSQL \*mysql)；

3-3两种方法均通过mysql\_fetch\_row()来访问每一条记录。

MYSQL\_ROW mysql\_fetch\_row(MYSQL\_RES \*result)

注意：若先前调用的是mysql\_store\_result(), 则直接在本地访问记录；若先前调用的是mysql\_use\_result(), 则此时才到服务器上去获取记录。

4）当处理完结果集后，调用mysql\_free\_result()来释放它所占的内存。void mysql\_free\_result(MYSQL\_RES \*result)。

5）可调用mysql\_errno()和mysql\_error()来获知最近执行的API函数的错误代码和错误信息。

unsigned int mysql\_errno(MYSQL \*mysql)

const char \*mysql\_error(MYSQL \*mysql)

5. 预处理语句(Prepared Statements)

MySQL C API还提供了另一种方式来执行SQL语句，即先预处理(prepare)再执行

(execute). 对于多次执行的SQL语句，该方式可以提高其执行效率。具体步骤如下：

1. 调用mysql\_stmt\_init()创建语句句柄，该句柄在随后的函数调用中都要用到。

MYSQL\_STMT \*mysql\_stmt\_init(MYSQL \*mysql)

2. 调用mysql\_stmt\_prepare()对SQL语句进行预处理

int mysql\_stmt\_prepare(MYSQL\_STMT \*stmt, const char \*stmt\_str, unsigned long length)

3. 如果SQL语句中有参数，则需要调用mysql\_stmt\_bind\_param()进行参数绑定。

my\_bool mysql\_stmt\_bind\_param(MYSQL\_STMT \*stmt, MYSQL\_BIND \*bind)

如果参数的类型为TEXT或BLOB, 并且数据量很大，可以调用

mysql\_stmt\_send\_long\_data()来向服务器发送数据。

my\_bool mysql\_stmt\_send\_long\_data(MYSQL\_STMT \*stmt, unsigned int parameter\_number, const char \*data, unsigned long length)

4. 调用mysql\_stmt\_execute()来执行查询。

int mysql\_stmt\_execute(MYSQL\_STMT \*stmt)

5. 若查询不产生结果集，可以调用

mysql\_stmt\_affected\_rows()和

my\_ulonglong mysql\_stmt\_insert\_id(MYSQL\_STMT \*stmt)来获得被改变的记录数和生成的自增值。

my\_ulonglong mysql\_stmt\_affected\_rows(MYSQL\_STMT \*stmt)

my\_ulonglong mysql\_stmt\_insert\_id(MYSQL\_STMT \*stmt)

否则，执行mysql\_stmt\_bind\_result()对结果集中的字段进行绑定。

my\_bool mysql\_stmt\_bind\_result(MYSQL\_STMT \*stmt, MYSQL\_BIND \*bind)

6. 调用mysql\_stmt\_fetch()来逐行获取结果集中的记录。

int mysql\_stmt\_fetch(MYSQL\_STMT \*stmt)

在调用mysql\_stmt\_fetch()之前，还可以执行mysql\_stmt\_store\_result()将结果

集预先缓存到本地。

int mysql\_stmt\_store\_result(MYSQL\_STMT \*stmt)

7. 重复步骤3-6, 每次使用不同的实参来执行查询。

8. 调用mysql\_stmt\_close()关闭句柄，释放资源

my\_bool mysql\_stmt\_close(MYSQL\_STMT \*)

此外，可以调用mysql\_stmt\_errno()和mysql\_stmt\_error()来获知最近执行的预处

理语句API函数的错误代码和错误信息。

unsigned int mysql\_stmt\_errno(MYSQL\_STMT \*stmt)

const char \*mysql\_stmt\_error(MYSQL\_STMT \*stmt)

9.其他说明：

mysql\_stmt\_execute()中有调用案例

<http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-multiple-queries>

25.2.7.11. mysql\_stmt\_fetch()

## 2.3 MYSQL C API预处理语句

|  |
| --- |
| 参考 2.2节，调用思路分析：5预处理语句(Prepared Statements) |
| MySQL客户端／服务器协议提供了预处理语句。该功能采用了由mysql\_stmt\_init()初始化函数返回的MYSQL\_STMT语句处理程序数据结构。对于多次执行的语句，预处理执行是一种有效的方式。首先对语句进行解析，为执行作好准备。接下来，在以后使用初始化函数返回的语句句柄执行一次或多次。  对于多次执行的语句，预处理执行比直接执行快，主要原因在于，仅对查询执行一次解析操作。在直接执行的情况下，每次执行语句时，均将进行查询。此外，由于每次执行预处理语句时仅需发送参数的数据，从而减少了网络通信量。  预处理语句的另一个优点是，它采用了二进制协议，从而使得客户端和服务器之间的数据传输更有效率。 |
| 编程步骤   * 1. 用mysql\_stmt\_init()创建预处理语句句柄。要想在服务器上准备预处理语句，可调用mysql\_stmt\_prepare()，并为其传递包含SQL语句的字符串 * 2. 如果语句生成了结果集，调用mysql\_stmt\_result\_metadata()以获得结果集元数据。虽然与包含查询返回列的结果集不同，该元数据本身也采用了结果集的形式。元数据结果集指明了结果中包含多少列，并包含每一列的信息。 * 3.  使用mysql\_stmt\_bind\_param()设置任何参数的值。必须设置所有参数。否则，语句执行将返回错误，或生成无法预料的结果。 * 4.  调用mysql\_stmt\_execute()执行语句。 * 5.  如果语句生成了结果集，捆绑数据缓冲，通过调用mysql\_stmt\_bind\_result()，检索行值。 * 6.  通过重复调用mysql\_stmt\_fetch()，按行将数据提取到缓冲区，直至未发现更多行为止。 * 7.  通过更改参数值并再次执行语句，重复步骤3到步骤6。 |

## 2.4多查询执行的C API处理

|  |
| --- |
| * 支持在单个字符串中指定的多语句的执行。要想与给定的连接一起使用该功能，打开连接时，必须将标志参数中的CLIENT\_MULTI\_STATEMENTS选项指定给mysql\_real\_connect()。也可以通过调用mysql\_set\_server\_option(MYSQL\_OPTION\_MULTI\_STATEMENTS\_ON)，为已有的连接设置它 |
| /\* Connect to server with option CLIENT\_MULTI\_STATEMENTS \*/  mysql\_real\_connect(..., CLIENT\_MULTI\_STATEMENTS);    /\* Now execute multiple queries \*/  mysql\_query(mysql,"DROP TABLE IF EXISTS test\_table;\  CREATE TABLE test\_table(id INT);\  INSERT INTO test\_table VALUES(10);\  UPDATE test\_table SET id=20 WHERE id=10;\  SELECT \* FROM test\_table;\  DROP TABLE test\_table");  do  {  /\* Process all results \*/  ...  printf("total affected rows: %lld", mysql\_affected\_rows(mysql));  ...  if (!(result= mysql\_store\_result(mysql)))  {  printf(stderr, "Got fatal error processing query\n");  exit(1);  }  process\_result\_set(result); /\* client function \*/  mysql\_free\_result(result);  } while (!mysql\_next\_result(mysql)); |

2.5

|  |
| --- |
| 二进制协议允许你使用MYSQL\_TIME结构发送和接受日期和时间值（DATE、TIME、DATETIME和TIMESTAMP）。在25.2.5节，“C API预处理语句的数据类型”中，介绍了该结构的成员。 |
| int main()  {  int ret = 0, status = 0;  MYSQL \*mysql;  MYSQL\_RES \*result;    mysql =mysql\_init(NULL);  mysql =mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb2", 0, NULL, CLIENT\_MULTI\_STATEMENTS );  if (mysql == NULL)  {  ret = mysql\_errno(mysql);  printf("%s", mysql\_error(mysql));  printf("func mysql\_real\_connect() err :%d\n", ret);  return ret;  }  else  {  printf(" ok......\n");  }    MYSQL\_TIME ts;  MYSQL\_BIND bind[3];  MYSQL\_STMT \*stmt;    //注意：  // 创建的表语句  // create table test\_table (date\_field date, time\_field time, timestamp\_field timestamp );  char query[1024] = "INSERT INTO test\_table(date\_field, time\_field, timestamp\_field) VALUES(?,?,?)";    stmt = mysql\_stmt\_init(mysql);  if (!stmt)  {  fprintf(stderr, " mysql\_stmt\_init(), out of memory\n");  exit(0);  }  if (mysql\_stmt\_prepare(stmt, query, strlen(query)))  {  fprintf(stderr, "\n mysql\_stmt\_prepare(), INSERT failed");  fprintf(stderr, "\n %s", mysql\_stmt\_error(stmt));  exit(0);  }    /\* set up input buffers for all 3 parameters \*/  bind[0].buffer\_type= MYSQL\_TYPE\_DATE;  bind[0].buffer= (char \*)&ts;  bind[0].is\_null= 0;  bind[0].length= 0;  //  bind[1]= bind[2]= bind[0];  //...    mysql\_stmt\_bind\_param(stmt, bind);    /\* supply the data to be sent in the ts structure \*/  ts.year= 2002;  ts.month= 02;  ts.day= 03;    ts.hour= 10;  ts.minute= 45;  ts.second= 20;    mysql\_stmt\_execute(stmt);    // Close the statement //  if (mysql\_stmt\_close(stmt))  {  fprintf(stderr, " failed while closing the statement\n");  fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));  exit(0);  }    mysql\_close(mysql);  } |

## 2.5事务概念

|  |
| --- |
| /\*  MYSQL默认是自动提交的，也就是你提交一个QUERY，它就直接执行！我们可以通过  set autocommit=0 禁止自动提交  set autocommit=1 开启自动提交  mysql中INNODB引擎才支持事务处理，默认是自动提交的；  另外一种常用的MYISAM引擎是不支持事务的，本身就没有事务的概念  \*/  #define BEGIN\_TRAN "START TRANSACTION"  #define SET\_TRAN "SET AUTOCOMMIT=0"  #define UNSET\_TRAN "SET AUTOCOMMIT=1"  #define COMMIT\_TRAN "COMMIT"  #define ROLLBACK\_TRAN "ROLLBACK" |
| int mysql\_BeginTran(MYSQL \*mysql)  {  int ret = 0;  //--执行事务开始SQL  ret = mysql\_query(mysql, BEGIN\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }  //--设置事务手动提交  ret = mysql\_query(mysql, SET\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }  return ret;  }      int mysql\_Rollback(MYSQL \*mysql)  {  int ret = 0;    //--事务回滚操作  ret = mysql\_query(mysql, ROLLBACK\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }    //--恢复事务自动提交标志  ret = mysql\_query(mysql, UNSET\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }    return ret;  }  int mysql\_Commit(MYSQL \*mysql)  {  int ret = 0;    //--执行事务提交SQL  ret = mysql\_query(mysql, COMMIT\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }    //--恢复自动提交设置  ret = mysql\_query(mysql, UNSET\_TRAN);  if (ret != 0)  {  printf("func mysql\_query() err: %d\n", ret);  return ret;  }    return ret;  }  #define sql01 "INSERT INTO test\_table(col1,col2,col3) VALUES(10, '10', '1')"  #define sql02 "INSERT INTO test\_table(col1,col2,col3) VALUES(20, '20', '2')"  #define sql03 "INSERT INTO test\_table(col1,col2,col3) VALUES(30, '30', '3')"  #define sql04 "INSERT INTO test\_table(col1,col2,col3) VALUES(40, '40', '4')"  int main()  {  int ret = NULL;    MYSQL \*mysql;    MYSQL\_RES \*res;  MYSQL\_ROW row;  char \*query;    mysql = mysql\_init(NULL);    mysql =mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb2", 0, NULL, 0 );  if (mysql == NULL)  {  ret = mysql\_errno(mysql);  printf("func mysql\_real\_connect() err\n");  return ret;  }  else  {  printf(" ok......\n");  }    ret = mysql\_BeginTran(mysql);  if (ret != 0)  {  printf("mysql\_BeginTran() err:%d\n", ret);  return ret;  }  ret = mysql\_query(mysql, sql01);  if (ret != 0)  {  printf("mysql\_query() err:%d\n", ret);  return ret;  }  ret = mysql\_query(mysql, sql02);  if (ret != 0)  {  printf("mysql\_query() err:%d\n", ret);  return ret;  }  ret = mysql\_Commit(mysql);  if (ret != 0)  {  printf("mysql\_Commit() err:%d\n", ret);  return ret;  }      ret = mysql\_BeginTran(mysql);  if (ret != 0)  {  printf("mysql\_BeginTran() err:%d\n", ret);  return ret;  }  ret = mysql\_query(mysql, sql03);  if (ret != 0)  {  printf("mysql\_query() err:%d\n", ret);  return ret;  }  ret = mysql\_query(mysql, sql04);  if (ret != 0)  {  printf("mysql\_query() err:%d\n", ret);  return ret;  }  ret = mysql\_Rollback(mysql);  if (ret != 0)  {  printf("mysql\_Rollback() err:%d\n", ret);  return ret;  }    mysql\_close(mysql);    } |