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 |
|  |

|  |
| --- |
| 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 |

日期和时间函数的处理

多查询api的处理

### 25.2.9. 多查询执行的C API处理

MySQL 5.1支持在单个查询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。

在默认情况下，mysql\_query()和mysql\_real\_query()仅返回第1个查询的状态，并能使用mysql\_more\_results()和mysql\_next\_result()对后续查询的状态进行处理。

/\* 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));

多语句功能可与mysql\_query()或mysql\_real\_query()一起使用。它不能与预处理语句接口一起使用。按照定义，预处理语句仅能与包含单个语句的字符串一起使用。

## 2.3MYSQL C API的调用关系分析

2.3.1api函数概览

|  |  |
| --- | --- |
| **函数** | **描述** |
| **mysql\_affected\_rows()** | 返回上次UPDATE、DELETE或INSERT查询更改／删除／插入的行数。 |
| **mysql\_autocommit()** | 切换 autocommit模式，ON/OFF |
| **mysql\_change\_user()** | 更改打开连接上的用户和数据库。 |
| **mysql\_charset\_name()** | 返回用于连接的默认字符集的名称。 |
| **mysql\_close()** | 关闭服务器连接。 |
| **mysql\_commit()** | 提交事务。 |
| **mysql\_connect()** | 连接到MySQL服务器。该函数已不再被重视，使用mysql\_real\_connect()取代。 |
| **mysql\_create\_db()** | 创建数据库。该函数已不再被重视，使用SQL语句CREATE DATABASE取而代之。 |
| **mysql\_data\_seek()** | 在查询结果集中查找属性行编号。 |
| **mysql\_debug()** | 用给定的字符串执行DBUG\_PUSH。 |
| **mysql\_drop\_db()** | 撤销数据库。该函数已不再被重视，使用SQL语句DROP DATABASE取而代之。 |
| **mysql\_dump\_debug\_info()** | 让服务器将调试信息写入日志。 |
| **mysql\_eof()** | 确定是否读取了结果集的最后一行。该函数已不再被重视，可以使用mysql\_errno()或mysql\_error()取而代之。 |
| **mysql\_errno()** | 返回上次调用的MySQL函数的错误编号。 |
| **mysql\_error()** | 返回上次调用的MySQL函数的错误消息。 |
| **mysql\_escape\_string()** | 为了用在SQL语句中，对特殊字符进行转义处理。 |
| **mysql\_fetch\_field()** | 返回下一个表字段的类型。 |
| **mysql\_fetch\_field\_direct()** | 给定字段编号，返回表字段的类型。 |
| **mysql\_fetch\_fields()** | 返回所有字段结构的数组。 |
| **mysql\_fetch\_lengths()** | 返回当前行中所有列的长度。 |
| **mysql\_fetch\_row()** | 从结果集中获取下一行 |
| **mysql\_field\_seek()** | 将列光标置于指定的列。 |
| **mysql\_field\_count()** | 返回上次执行语句的结果列的数目。 |
| **mysql\_field\_tell()** | 返回上次mysql\_fetch\_field()所使用字段光标的位置。 |
| **mysql\_free\_result()** | 释放结果集使用的内存。 |
| **mysql\_get\_client\_info()** | 以字符串形式返回客户端版本信息。 |
| **mysql\_get\_client\_version()** | 以整数形式返回客户端版本信息。 |
| **mysql\_get\_host\_info()** | 返回描述连接的字符串。 |
| **mysql\_get\_server\_version()** | 以整数形式返回服务器的版本号。 |
| **mysql\_get\_proto\_info()** | 返回连接所使用的协议版本。 |
| **mysql\_get\_server\_info()** | 返回服务器的版本号。 |
| **mysql\_info()** | 返回关于最近所执行查询的信息。 |
| **mysql\_init()** | 获取或初始化MYSQL结构。 |
| **mysql\_insert\_id()** | 返回上一个查询为AUTO\_INCREMENT列生成的ID。 |
| **mysql\_kill()** | 杀死给定的线程。 |
| **mysql\_library\_end()** | 最终确定MySQL C API库。 |
| **mysql\_library\_init()** | 初始化MySQL C API库。 |
| **mysql\_list\_dbs()** | 返回与简单正则表达式匹配的数据库名称。 |
| **mysql\_list\_fields()** | 返回与简单正则表达式匹配的字段名称。 |
| **mysql\_list\_processes()** | 返回当前服务器线程的列表。 |
| **mysql\_list\_tables()** | 返回与简单正则表达式匹配的表名。 |
| **mysql\_more\_results()** | 检查是否还存在其他结果。 |
| **mysql\_next\_result()** | 在多语句执行过程中返回/初始化下一个结果。 |
| **mysql\_num\_fields()** | 返回结果集中的列数。 |
| **mysql\_num\_rows()** | 返回结果集中的行数。 |
| **mysql\_options()** | 为mysql\_connect()设置连接选项。 |
| **mysql\_ping()** | 检查与服务器的连接是否工作，如有必要重新连接。 |
| **mysql\_query()** | 执行指定为“以Null终结的字符串”的SQL查询。 |
| **mysql\_real\_connect()** | 连接到MySQL服务器。 |
| **mysql\_real\_escape\_string()** | 考虑到连接的当前字符集，为了在SQL语句中使用，对字符串中的特殊字符进行转义处理。 |
| **mysql\_real\_query()** | 执行指定为计数字符串的SQL查询。 |
| **mysql\_refresh()** | 刷新或复位表和高速缓冲。 |
| **mysql\_reload()** | 通知服务器再次加载授权表。 |
| **mysql\_rollback()** | 回滚事务。 |
| **mysql\_row\_seek()** | 使用从mysql\_row\_tell()返回的值，查找结果集中的行偏移。 |
| **mysql\_row\_tell()** | 返回行光标位置。 |
| **mysql\_select\_db()** | 选择数据库。 |
| **mysql\_server\_end()** | 最终确定嵌入式服务器库。 |
| **mysql\_server\_init()** | 初始化嵌入式服务器库。 |
| **mysql\_set\_server\_option()** | 为连接设置选项（如多语句）。 |
| **mysql\_sqlstate()** | 返回关于上一个错误的SQLSTATE错误代码。 |
| **mysql\_shutdown()** | 关闭数据库服务器。 |
| **mysql\_stat()** | 以字符串形式返回服务器状态。 |
| **mysql\_store\_result()** | 检索完整的结果集至客户端。 |
| **mysql\_thread\_id()** | 返回当前线程ID。 |
| **mysql\_thread\_safe()** | 如果客户端已编译为线程安全的，返回1。 |
| **mysql\_use\_result()** | 初始化逐行的结果集检索。 |
| **mysql\_warning\_count()** | 返回上一个SQL语句的告警数。 |

总结: 5.1版本68个

5.6版本71个

|  |
| --- |
|  |
| 与MySQL交互时，应用程序应使用该一般性原则：  1.    通过调用mysql\_library\_init()，初始化MySQL库。库可以是mysqlclient C客户端库，或mysqld嵌入式服务器库，具体情况取决于应用程序是否与“-libmysqlclient”或“-libmysqld”标志链接。  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。（在下面的讨论中，详细介绍了使用它的方法）。  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。  调用mysql\_library\_init()和mysql\_library\_end()的目的在于，为MySQL库提供恰当的初始化和结束处理。对于与客户端库链接的应用程序，它们提供了改进的内存管理功能。如果不调用mysql\_library\_end()，内存块仍将保持分配状态（这不会增加应用程序使用的内存量，但某些内存泄漏检测器将抗议它）。对于与嵌入式服务器链接的应用程序，这些调用会启动并停止服务器。  mysql\_library\_init()和mysql\_library\_end()实际上是#define符号，这类符号使得它们等效于mysql\_server\_init()和mysql\_server\_end()，但其名称更清楚地指明，无论应用程序使用的是mysqlclient或mysqld库，启动或结束MySQL库时，应调用它们。对于早期的MySQL版本，可调用mysql\_server\_init()和mysql\_server\_end()取而代之。  如果愿意，可省略对mysql\_library\_init()的调用，这是因为，必要时，mysql\_init()会自动调用它。  要想连接到服务器，可调用mysql\_init()来初始化连接处理程序，然后用该处理程序（以及其他信息，如主机名、用户名和密码）调用mysql\_real\_connect()。建立连接后，在低于5.0.3版的API中，mysql\_real\_connect()会将再连接标志（MYSQL结构的一部分）设置为1，或在较新的版本中，将其设置为0。对于该标志，值“1”指明，如果因连接丢失而无法执行语句，放弃之前，会尝试再次连接到服务器。从MySQL 5.0.13开始，可以在mysql\_options()上使用MYSQL\_OPT\_RECONNECT选项，以控制再连接行为。完成连接后，调用mysql\_close()中止它。  当连接处于活动状态时，客户端或许会使用mysql\_query()或mysql\_real\_query()向服务器发出SQL查询。两者的差别在于，mysql\_query()预期的查询为指定的、由Null终结的字符串，而mysql\_real\_query()预期的是计数字符串。如果字符串包含二进制数据（其中可能包含Null字节），就必须使用mysql\_real\_query()。  对于每个非SELECT查询（例如INSERT、UPDATE、DELETE），通过调用mysql\_affected\_rows()，可发现有多少行已被改变（影响）。  对于SELECT查询，能够检索作为结果集的行。注意，某些语句因其返回行，类似与SELECT。包括SHOW、DESCRIBE和EXPLAIN。应按照对待SELECT语句的方式处理它们。  客户端处理结果集的方式有两种。一种方式是，通过调用mysql\_store\_result()，一次性地检索整个结果集。该函数能从服务器获得查询返回的所有行，并将它们保存在客户端。第二种方式是针对客户端的，通过调用mysql\_use\_result()，对“按行”结果集检索进行初始化处理。该函数能初始化检索结果，但不能从服务器获得任何实际行。  在这两种情况下，均能通过调用mysql\_fetch\_row()访问行。通过mysql\_store\_result()，mysql\_fetch\_row()能够访问以前从服务器获得的行。通过mysql\_use\_result()，mysql\_fetch\_row()能够实际地检索来自服务器的行。通过调用mysql\_fetch\_lengths()，能获得关于各行中数据大小的信息。  完成结果集操作后，请调用mysql\_free\_result()释放结果集使用的内存。  这两种检索机制是互补的。客户端程序应选择最能满足其要求的方法。实际上，客户端最常使用的是mysql\_store\_result()。  mysql\_store\_result()的1个优点在于，由于将行全部提取到了客户端上，你不仅能连续访问行，还能使用mysql\_data\_seek()或mysql\_row\_seek()在结果集中向前或向后移动，以更改结果集内当前行的位置。通过调用mysql\_num\_rows()，还能发现有多少行。另一方面，对于大的结果集，mysql\_store\_result()所需的内存可能会很大，你很可能遇到内存溢出状况。  mysql\_use\_result()的1个优点在于，客户端所需的用于结果集的内存较少，原因在于，一次它仅维护一行（由于分配开销较低，mysql\_use\_result()能更快）。它的缺点在于，你必须快速处理每一行以避免妨碍服务器，你不能随机访问结果集中的行（只能连续访问行），你不知道结果集中有多少行，直至全部检索了它们为止。不仅如此，即使在检索过程中你判定已找到所寻找的信息，也必须检索所有的行。  通过API，客户端能够恰当地对查询作出响应（仅在必要时检索行），而无需知道查询是否是SELECT查询。可以在每次mysql\_query()或mysql\_real\_query()后，通过调用mysql\_store\_result()完成该操作。如果结果集调用成功，查询为SELECT，而且能够读取行。如果结果集调用失败，可调用mysql\_field\_count()来判断结果是否的确是所预期的。如果mysql\_field\_count()返回0，查询不返回数据（表明它是INSERT、UPDATE、DELETE等），而且不返回行。如果mysql\_field\_count()是非0值，查询应返回行，但没有返回行。这表明查询是失败了的SELECT。关于如何实现该操作的示例，请参见关于mysql\_field\_count()的介绍。  无论是mysql\_store\_result()还是mysql\_use\_result()，均允许你获取关于构成结果集的字段的信息（字段数目，它们的名称和类型等）。通过重复调用mysql\_fetch\_field()，可以按顺序访问行内的字段信息，或者，通过调用mysql\_fetch\_field\_direct()，能够在行内按字段编号访问字段信息。通过调用mysql\_field\_seek()，可以改变当前字段的光标位置。对字段光标的设置将影响后续的mysql\_fetch\_field()调用。此外，你也能通过调用mysql\_fetch\_fields()，一次性地获得关于字段的所有信息。  为了检测和通报错误，MySQL提供了使用mysql\_errno()和mysql\_error()函数访问错误信息的机制。它们能返回关于最近调用的函数的错误代码或错误消息，最近调用的函数可能成功也可能失败，这样，你就能判断错误是在何时出现的，以及错误是什么。 |

### 25.2.13. 使用C API时的常见问题

[25.2.13.1. 为什么在mysql\_query()返回成功后，mysql\_store\_result()有时会返回NULL](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#null-mysql-store-result)

[25.2.13.2. What Results You Can Get from a Query](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#query-results)

[25.2.13.3. 如何获得上次插入行的唯一ID](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#getting-unique-id)

[25.2.13.4. 与C API有关的问题](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-linking-problems)

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)

#### 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()

mysql的有类型定义