附录F C API 参 考

本附录介绍 MySQL 客户机库的C语言应用程序编程接口。API 由一组函数以及一组用于函数的数据类型组成，这些函数与 MySQL 服务器进行通信并访问数据库。

本附录是一个参考，因此它只包括简要的代码段来说明客户机库的使用。有关完整的客户机程序和编写说明，请参阅第 6章。

F.1 编译和连接

在源程序级，客户机库定义在 mysql.h 头文件中，在源文件中可包括：

```
#include <mysql.h>
```

为了指示编译器在哪里查找该文件，需要指定 -I/path/to/include/dir 选项。例如，如果您的 MySQL 头文件被安装在 /usr/include/mysql 或 /usr/local/mysql/include 中，可以通过使用以下的命令编译源文件：

```
% gcc -I/usr/include/mysql -c myprog.c
% gcc -I/usr/local/mysql/include -c myprog.c
```

如果需要使用其他的 MySQL 头文件，它们应放在与 mysql.h 相同的目录中。例如，mysql_com.h 包含解释查询结果元数据的常量和宏指令（ mysql_com.h 通过 mysql.h 来包含，因此，尽管可能会直接查看该文件中的内容，但通过包括 mysql.h 您可以简单地访问它）。头文件 errmsg.h 和 mysqld_error.h 中包含错误代码的常量。

在目标级，客户机库作为mysqlclient 库被提供。为了将该库连接到程序中，需要在连接命令中指定 -lmysqlclient。您或许还需要用 -L/path/to/lib/die 选项指示连接器在哪里查找该库。例如：

```
% gcc -o myprog myprog.o -L/usr/lib/mysql -lmysqlclient
% gcc -o myprog myprog.o -L/usr/local/mysql/lib -lmysqlclient
```

F.2 C API 数据类型

MySQL 客户机库的数据类型被设计来表示在与服务器的会话期间您所处理的实体。有针对连接本身的、查询结果的、结果中的一行的以及元数据（描述有关构成列结果的信息）的类型。

请注意，在以下讨论中，术语“ 列 ”和“ 字段 ”是同义词。

F.2.1 标量数据类型

MySQL 的标量数据类型表示像超大整数、逻辑值和字段偏移量这样的值。

- my_ulonglong 长整型，用于函数的返回值，如 mysql_affected_rows()、mysql_num_rows() 和 mysql_insert_id()，这些函数都返回行计数或其他可能的大数值。

  为了打印 my_ulonglong 值，可以定义它为 unsigned long（无符号长整型）并使用‘ %lu ’的格式。如果不这样做，在某些系统中该值将不能正确地打印。
- my_bool 逻辑型，用于 mysql_change_user() 和 mysql_eof() 的返回值。

- MYSQL_FIELD_OFFSET 该数据类型用于 mysql_field_tell() 和 mysql_field_seek() 函数，表示当前结果集中 MYSQL_FIELD 结构集合中的偏移量。

F.2.2 非标量数据类型

MySQL 的非标量类型表示结构或数组。所有的 MYSQL 和 MYSQL_RES 结构都被认为是一个“黑匣子”。换言之，您只能引用结构本身，而不能引用结构中的元素。 MYSQL_ROW 和 MYSQL_FIELD类型则没有这样的限制，您可以直接访问它们来获得作为查询结果返回的数据和元数据。

- MYSQL 主要的客户机库类型是 MYSQL 结构的，它用于连接处理程序。处理程序中包含有关服务器的连接状态的信息。初始化一个 MYSQL 结构，然后经过该结构到一个连接，来打开服务器的会话。在建立该连接后，可以使用该处理程序发布查询、生成结果集、获得错误信息，等等。
- MYSQL_RES 像 SELECT 或 SHOW 这样的查询借助于一个结果集将数据返回给客户机，这个结果集是用 MYSQL_RES 结构表示的。该结构中包含由查询返回的行的信息。

在得到结果集后，可以调用 API 函数得到结果集数据（集合的每行中的数据值）或元数据（关于该结果的信息，如由多少列、类型和长度等等）。

- MYSQL_ROW MYSQL_ROW 类型包含一个数据行的值，表示为计数字节串（counted byte）的数组。所有值（甚至是数字）都以串的形式返回，除了当行中的值为 NULL时它通过 CNULL 指针以 MYSQL_ROW 结构表示外。

行中值的数目由 mysql_num_fields() 指定。行中 i-th 列值由 row[i] 指定，这里 i 的范围从 0 到 mysql_num_fields(res_set)-1(res_set 是指向 MYSQL_RES 结果集的一个指针)。请注意，MYSQL_ROW 类型已经是一个指针，因此可以这样声明一个行变量：

```
MYSQL_ROW row;      /*正确*/
```

但不能这样声明：

```
MYSQL_ROW *row;    /*不正确*/
```

MYSQL_ROW 类型的值有终结的空值（terminating null），因此，非二进制值可以认为是空终结串。但是，包含二进制数据的数据值可以在内部包含空字节并且被看作是计数串。为了得到行中值的长度，这样调用 mysql_fetch_lengths()：

```
unsigned long *length;
length = mysql_fetch_lengths (res_set);
```

对于 NULL 值来说，mysql_fetch_lengths() 返回的长度为零。

- MYSQL_FIELD 客户机库使用 MYSQL_FIELD 结构表示关于该结果集中列的元数据——每列一个结构。MYSQL_FIELD 结构的数目由 mysql_num_fields() 指定。您可以通过 mysql_ num_field_() 访问连续的字段结构，或用 mysql_field_tell() 和 mysql_field_seek() 在结构之间来回移动。

MYSQL_FIELD 结构对于显示或解释数据行的内容很有用。它类似于：

```
typedef struct st_mysql_field {
    char *name;
    char *table;
    char *def;
```

```
    enum enum_field_types type;
    unsigned int length;
    unsigned int max_length;
    unsigned int flags;
    unsigned int decimals;
} MYSQL_FIELD;
```

MYSQL_FIELD 结构的成员由如下含义：

- name 列名，为空终结串。如果该列值作为一个表达式的结果被计算，则 name 是以串形式表示的该表达式。如果给列或表达式一个列的别名，则 name 是该别名。例如，下列的查询产生“my_col”、”4*(my_col+1)”和“mc”的name 值：

```
SELECT my_col, 4*(my_col+1), my_col AS mc ...
```

- table 列所在的表的名字，为空终结串。如果列值作为表达式的值被计算，则 table 是空串。例如，如果发布类似下面的查询，第一列 (my_col) 的表名是 my_tbl，而第二列 (my_col+0) 的表名为空串：

```
SELECT my_col, my_col+0 FROM my_tbl ...
```

- def 列的缺省值，为空终结串。MYSQL_FIELD 结构的该成员仅在调用 mysql_list_fields() 时设置，其他情况为 NULL（列缺省值可以通过发布 DESCRIBE tbl_name 或 SHOW FIELDS FROM tbl_name 查询和检查该结果集来获得）。
- type 列的类型。作为表达式结果被计算的列的类型从该表达式元素的类型中确定。例如，如果 my_col 是 VARCHAR(20) 列，则 type 是 FIELD_TYPE_VAR_STRING，而 LENGTH(my_col) 的 type 为 FIELD_TYPE_LONGLONG。

可能的 type 值列在 mysql_com.h 中并在表F-1 中示出。

表F-1 MYSQL_FIELD 的 type 成员值

type 值	列 类 型
FIELD_TYPE_BLOB	BLOB 或 TEXT
FIELD_TYPE_DATE	DATE
FIELD_TYPE_DATETIME	DATETIME
FIELD_TYPE_DECIMAL	DECIMAL 或 NUMERIC
FIELD_TYPE_DOUBLE	DOUBLE 或 REAL
FIELD_TYPE_ENUM	ENUM
FIELD_TYPE_FLOAT	FOLAT
FIELD_TYPE_INT24	MEDIUMINT
FIELD_TYPE_LONG	INT
FIELD_TYPE_LONGLONG	BIGINT
FIELD_TYPE_NULL	NULL
FIELD_TYPE_SET	SET
FIELD_TYPE_SHORT	SMALLINT
FIELD_TYPE_STRING	CHAR
FIELD_TYPE_TIME	TIME
FIELD_TYPE_TIMESTAMP	TIMESTAMP
FIELD_TYPE_TINY	TINYINT
FIELD_TYPE_VAR_STRING	VARCHAR
FIELD_TYPE_YEAR	YEAR

您可能在旧的源文件中看到对 FIELD_TYPE_CHAR 的引用，它是一字节的类型，现在称 FIELD_TYPE_TINY。

- length 列的长度，该长度在创建该表的 CREATE TABLE 语句中指定。作为表达式结果被计算的列的长度从该表达式的元素中确定。
- max_length 实际显示在结果集中的最长的列值长度。例如，如果结果集中的列包含值“ Bill ”、“ Jack ”和“ Belviders ”，则该列的 max_length 值为 9。
  由于 max_length 值只在所有行已经知晓后才能确定，因此它只对用 mysql_store_result() 创建的结果集有意义。对于 用 mysql_use_result() 所创建的结果集，max_length 为 0。
- flags flags 成员指定列的属性。属性由 flags 值中的单个的比特位表示。这些比特位可以通过表F-2 中的比特掩码常量进行测试。例如，为了确定列值是否为 UNSIGNED，应测试flags 的值：

```
if (field -> flags & UNSIGNED_FLAG)
   printf ( “ %s values are UNSIGNED \n ” , field -> name);
```

表F-2 MYSQL_FIELD 的 flags 成员值

flag 值	含 义
AUTO_INCREMENT_FLAG	列具有 AUTO_INCREMENT 属性
BINARY_FLAG	列具有 BINARY 属性
MULTIPLE_KEY_FLAG	列是NON-UNIQUE 索引的一部分
NOT_NULL_FLAG	列不能包含 NULL 值
PRI_KEY_FLAG	列是 PRIMARY KEY的一部分
UNIQUE_KEY_FLAG	列是 UNIQUE 索引的一部分
UNSIGNED_FLAG	列具有 UNSIGNED 属性
ZEROFILL_FLAG	列具有 ZEROFILL 属性

有些 flags 常量指定列的类型而非列属性，但现在它们不受欢迎，因为可使用 field->type 确定列的类型。表F-3列出这些不受赞成的常量。

表F-3 不赞成的 MYSQL_FIELD 的 flags 成员值

flags 值	含 义
BLOB_FLAG	列包含 BLOB 值
ENUM_FLAG	列包含 ENUM 值
SET_FLAG	列包含 SET 值
TIMESTAMP_FLAG	列包含 TIMESTAMP 值

- decimals 数值列的十进制小数位，非数值列的 decimals 为零。例如，DECIMAL（8,3）列的 decimals 值为 3，而 BLOB 列为 0 值。

F.2.3 存取器宏指令

mysql.h 包含一些宏指令，它们允许您更方便地测试 MYSQL_FIELD 成员。IS_NUM() 测试 type 成员，其他的测试 flags 成员。

如果列值有一个数值类型，则 IS_NUM() 为真（非零）：

```
if (IS_NUM (field->type))
    printf ("Field %s is numeric\n", field->name);
```

如果列是主键的一部分，则 IS_PRI_KEY() 为真：

```
if (IS_PRI_KEY (field->flags))
    printf ("Field %s is part of primary key\n", field->name);
```

如果列不包含 NULL 值，则 IS_NOT_NULL() 为真：

```
if (IS_NOT_NULL (field->flags))
    printf ("Field %s values cannot be NULL\n", field->name);
```

如果列是 BLOB 或 TEXT，则 IS_BLOB() 为真。但是，该宏指令测试了不受赞成的 flags 成员的 BLOB_FLAG 比特位，因此 IS_BLOB() 也不受欢迎。

F.3 C API 函数

客户机库函数可以分为以下几类：

- 连接管理例程，用于建立和终止对服务器的连接。
- 状态和错误报告例程，用于获得错误代码和消息。
- 查询构造和运行例程，用于构造查询并发送它们到服务器中。
- 结果设置处理例程，用于处理来自返回数据的查询的结果。
- 信息例程，用于提供有关客户机、服务器、协议版本和当前连接的信息。
- 管理例程，用于控制服务器的操作。
- 调试例程，用于产生调试信息。
- 考虑废除的不赞成的例程。

每个类别中的客户机库函数将在以下小节中详细介绍，并按字母顺序列出。某些参数重复出现在一些函数的描述中并具有标准的含义：

- conn 是一个指向服务器连接的 MYSQL 连接处理程序的指针。
- res_set 是一个指向 MYSQL_RES 结果集结构的指针。
- field 是一个指向 MYSQL_FIELD 列的信息结构的指针。
- row 是结构集的一个 MYSQL_ROW 数据行。

简单地说，这些参数将不在函数的描述中再讨论，您可以认为它们的含义是刚才指定的。

除非另有说明，否则可以假定任何给出的函数早在 MySQL 3.21.10 时就出现在客户机库中。

F.3.1 连接管理例程

这些函数允许建立和终止对服务器的连接，设置影响连接建立的选项，以及重新建立超时的连接。

- my_bool

```
mysql_change_user (MYSQL *conn, char *user_name, char *password, char
*db_name);
```

修改由 conn 所指定的连接中的用户和缺省的数据库。对于不包括数据库说明符的表的引

用来说，数据库是缺省的。如果 db_name 为 NULL，则不选择任何缺省数据库。

如果允许用户对该服务器的连接，则 mysql_change_user() 返回真值，并且，如果指定数据库，则用户具有访问该数据库的许可权。否则该函数失败且当前的用户和数据库仍然保持不变。

mysql_change_user() 是在 MySQL 3 .23.3 中引入的。

■ void

```
mysql_close (MYSQL *conn);
```

关闭由 conn 指定的连接。当完成一个服务器会话时调用该例程。如果该连接处理程序由 mysql_init() 自动分配，则 mysql_close() 解除该分配。

如果打开一个连接的企图失败，则不要调用 mysql_close()。

■ MYSQL *

```
mysql_init (MYSQL *conn);
```

初始化一个连接处理程序并返回指向它的指针。如果 conn 指向一个已有的 MYSQL 结构，则 mysql_init() 初始化该处理程序并返回其地址：

```
MYSQL conn_struct, *conn;
conn = mysql_init (&conn_struct);
```

如果 conn 为 NULL，则 mysql_init() 分配一个新的处理程序，对它进行初始化，并返回其地址：

```
MYSQL *conn;
conn = mysql_init (NULL);
```

如果 mysql_init() 失败，它会返回 NULL。如果 mysql_init() 不能分配一个新的处理程序，这种情况可能会发生。

如果 mysql_init() 分配该处理程序，则 mysql_close() 在您关闭连接时自动解除它。

mysql_init() 是在 MySQL 3.22.1 中引入的。

■ int

```
mysql_options (MYSQL *conn, enum mysql_option option, char *arg);
```

该函数允许您制作比单独使用 mysql_real_connect() 时可能的更精确的连接性能。应在 mysql_init() 后以及 mysql_real_connect() 前调用它。如果想要设置几个选项，可以多次调用 mysql_options。

option 参数指定您要设置哪个连接选项。设置该选项的附加功能（如果有的话）由 arg 参数指定。（请注意 arg 是一个指针）如果不需要任何附加信息，则 arg 为 NULL。

以下选项是可用的：

■ MYSQL_INIT_COMMAND 指定在连接到服务器后要执行的查询。 arg 是包含该查询的空终结串。该查询也在重新连接后运行（例如，如果调用 mysql_ping()）。由该查询返回的任何结果集都被丢弃。

■ MYSQL_OPT_COMPRESS 指定连接应该使用的压缩客户机 /服务器协议（如果服务器支持该协议）。arg 为 NULL。

它还可能在调用 mysql_real_connect() 时指定压缩。

■ MYSQL_OPT_CONNECT_TIMEOUT 指定连接超时（以秒为单位）。arg 是指向一个

包含超时值的 unsigned int 的指针。

- MYSQL_OPT_NAMED_PIPE 说明对服务器的连接应该使用指定的管道。arg 为 NULL。该选项只针对 Windows 95/98/NT 的客户机并且只针对Windows NT 服务器的连接。
- MYSQL_READ_DEFAULT_FILE 指定一个选项文件以读取连接参数。选项从文件的 [client] 组中读取。
- MYSQL_READ_DEFAULT_GROUP 指定要从选项文件中读取的组，该文件用 MYSQL_READ_DEFAULT_FILE 命名。读取除 [client] 组外的指定组。如果没有指定任何选项文件，则客户机库将查找标准选项文件并读它们。

如果多次调用 mysql_options() 来设置一个指定的选项，则 mysql_real_connect() 使用为该选项最近设置的指定值。

mysql_options() 运行成功时返回零，如果 option 值未知则返回非零值。

下列选项可用在选项文件中（在 [client] 组或在用 MYSQL_READ_DEFAULT_GROUP 选项指定的组中）：

```
compress
database=db_name
debug
host=host_name
init-command=query
password=your_password
pipe
port=port_num
return-found-rows
socket=socket_name
timeout=seconds
user=user_name
```

下例中的mysql_options() 调用使用压缩协议和 10 秒钟的超时设置连接选项，以便使 mysql_real_connect() 与指定的管道连接。它读取 c:\my.cnf.special 中的 [client] 和 [mygroup] 组的信息。在该连接建立时，执行 SET SQL_BIG_TABLES 语句。

```
MYSQL *conn;
unsigned int timeout;

if ((conn = mysql_init (NULL)) == NULL)
    ... deal with error ...
timeout = 10;
mysql_options (conn, MYSQL_OPT_CONNECT_TIMEOUT, (char *) &timeout);
mysql_options (conn, MYSQL_OPT_COMPRESS, NULL);
mysql_options (conn, MYSQL_OPT_NAMED_PIPE, NULL);
mysql_options (conn, MYSQL_READ_DEFAULT_FILE, "C:\my.cnf.special");
mysql_options (conn, MYSQL_READ_DEFAULT_GROUP, "mygroup");
mysql_options (conn, MYSQL_INIT_COMMAND, "SET SQL_BIG_TABLES=1");
if (mysql_real_connect (conn, ...) == NULL)
    ... deal with error ...
```

mysql_options() 是在 MySQL 3.22.1 中引入的。

MYSQL_INIT_COMMAND、MYSQL_READ_DEFAULT_FILE 和MYSQL_READ_DEFAULT_GROUP 是在 MySQL 3.22.10 中引入的。

- int

```
mysql_ping (MYSQL *conn);
```

检查由 conn 指定的连接是否仍在。如果不在， mysql_ping() 使用与建立初始连接所使用的相同的参数重新连接。因而，在没有成功地调用 mysql_real_connect() 之前不应该调用 mysql_ping()。

mysql_ping() 是在 MySQL3.22.1 中引入的。

■ MYSQL *

```
mysql_real_connect (MYSQL *conn, char *host_name,
                    char *user_name, char
                    *password, char *db_name,
                    unsigned int port_num, char
                    *socket_name,
                    unsigned int flags);
```

连接到服务器并返回一个指向该连接处理程序的指针。 conn 应该是指向一个已有连接处理程序的指针，该处理程序由 mysql_init() 初始化。该处理程序的地址是连接成功时的返回值。如果出现错误则返回 NULL。

如果该连接失败，您可以传递 conn 处理程序的值到 mysql_errno() 和 mysql_error() 中，以获得错误信息。但是，不应该传递 conn 值到可能影响成功连接的任何其他的客户机库例程中。

host_name 是要连接的服务器。表 F-4 示出该客户机试图如何对 UNIX 和 Windows 客户机的不同的 host_name 值进行连接（如果使用 UNIX 套接字或指定的管道进行连接，则 socket_name 参数指定该套接字或管道名）。

表F-4 基于服务器主机名类型的客户机连接类型

主机名值	UNIX 连接类型	Windows 连接类型
主机名	对指定主机的TCP/IP 连接	对指定主机的 TCP/IP 连接
IP 号	对指定主机的TCP/IP 连接	对指定主机的TCP/IP 连接
localhost	对本地主机的 UNIX 套接字连接不使用	对本地主机的 TCP/IP 连接对本地主机的指定的管道连接
NULL	对本地主机的 UNIX 套接字连接	对本地主机（除了 Windows NT）的TCP/IP 连接，在退到TCP/IP 前首先尝试指定的管道连接

user_name 是 MySQL 的用户名。如果它为 NULL，则客户机库发送一个缺省名。在 UNIX 中，该缺省值为您的注册名。在 Winodws 中，该缺省值是像在 USER 环境变量中所指定的名字（如果该变量被设置的话），否则为“ ODBC ”。

password 是口令。如果它为 NULL，则只能在 user 授权表包含与用户名和正在连接的主机相配的项并且该项有空白口令时进行连接。

db_name 是要使用的数据库。如果它为 NULL，则没有初始数据库可选择。

port_num 是TCP/IP 连接使用的端口号。如果它为 0，则使用缺省的端口号。

socket_name 是对本地主机进行连接所使用的套接字名。如果它为 NULL，则使用缺省的套接字名。

端口号和套接字名根据 host_name 的值进行使用。如果您正在连接到本地主机，则 mysql_real_connect() 试图使用 UNIX 域套接字（在 UNIX 下）或指定的管道（在 Windows

下）尝试一个连接。否则，它用 TCP/IP 连接。

flags 可以为 0，表示无选项，或为一个或更多的值，这些值在表 F-5 中示出。这些选项影响服务器的操作。

表F-5 mysql_real_connect() 的 flags 值

flag 值	服务器操作的效果
CLIENT_FOUND_ROWS	对于 UPDATE 查询，返回相配的行数而非变动的行数
CLIENT_NO_SCHEMA	不允许 db_name.tbl_name.col_name 的语法
CLIENT_COMPRESS	使用压缩通信协议，如果服务器支持该协议的话
CLIENT_ODBC	将客户机看作是 ODBC 客户机

如果指定 CLIENT_NO_SCHEMA，服务器在查询中只允许引用格式 tbl_name.col_name、tbl_name 或 col_name。

flag 值是比特位值，因此可以用‘ | ’‘ + ’组合它们，例如， CLIENT_COMPRESS|CLIENT_ODBC 或 CLIENT_COMPRESS+CLIENT_ODBC。

mysql_real_connect() 是在 MySQL 3.21.10 中引入的。db_name 参数是在 3.22.0 版本中引入的。而初始化 MYSQL 参数的 mysql_init() 是在 3.22.1 版本中引入的。

F.3.2 状态和错误报告例程

本节中的函数允许您确定和报告错误的原因。

■ unsigned int

```
mysql_errno (MYSQL *conn);
```

返回最近调用的客户机库例程的错误代码（返回一个状态）。如果没有出现错误，则错误代码为 0，否则为非零值。MySQL 头文件 errmsg.h 和mysqld_error.h 列出了可能的错误代码。

```
if (mysql_errno (conn) == 0)
    printf ("Everything is okay\n");
else
    printf ("Something is wrong!\n");
```

mysql_errno() 是在 MySQL 3.21.7 中引入的。

■ char

```
mysql_error (MYSQL *conn);
```

对于返回一个状态的最近调用的客户机库例程，返回包含错误消息的一个空终结串。如果没有错误，则该返回值是空串（这是长度为零的串 ””，而不是 NULL 指针）。尽管通常是在已经知道出现错误后调用 mysql_error()，但该返回值本身可以用来检测错误的发生：

```
if (mysql_error (conn)[0] == '\0')  /* empty string? */
    printf ("Everything is okay\n");
else
    printf ("Something is wrong!\n");
```

mysql_error() 是在 MySQL 3.21.7 中引进的。

F.3.3 查询结构和执行例程

本节中的函数允许您将查询发送到服务器中。 mysql_escape_string() 帮助您通过需要特殊

处理的转义字符来构造查询。

■ unsigned int

```
mysql_escape_string (char *to_str, char *from_str, unsigned int
from_len);
```

将包含特殊字符的串进行编码使得它可用于 SQL 语句中。表F-6 列出了被认为是特殊的字符和它们的编码。

表F-6 mysql_escape_string() 字符编码

特殊字符	编 码
NUL（ASCII 0）	\0
反斜杠	\\
单引号	\’
双引号	\”
换行	\n
回车	\r
Ctrl-Z	\Z

要编码的缓冲区指定为记数串。 from_str 指向该缓冲区， from_len 指定缓冲区中的字节数。mysql_escape_string() 将结果写入通过 to_str 指向的该缓冲区中，并增加一个空字节。to_str 必须指向一个已有的缓冲区，该缓冲区至少为（ from_len*2）+1 长（最坏的情况下，from_str 的所有字符都以一个二字符的序列进行编码，并且还需要终结空的空间）。

mysql_escape_string() 返回编码串的长度，但不对终结空字节记数。

编码串不包含内部的空值而是空终结的，因此您可以利用函数（如 strlen() 或 strcat()）使用它。

当在程序中书写字母串时，要小心不要将 C 程序设计语言的词汇转义约定与由 mysql_escape_string() 完成的编码相混淆。考虑下面的例子以及所产生的输出结果：

```
to_len = mysql_escape_string (to_str, "\0\\'\"\n\r\032", 7);
printf ("to_len = %d, to_str = %s\n", to_len, to_str);
```

输出结果为：

```
to_len = 14, to_str = \0\\'\"\n\r\Z
```

to_str 的打印值非常像做为 mysql_escape_string() 调用的第二个参数所指定的串，但其实是极为不同的。

■ int

```
mysql_query (MYSQL *conn, char *query_string);
```

如果给定一个指定为空终结串的查询， mysql_query() 将该查询发送到服务器中运行。该串不包含二进制数据；尤其是不包含空字节， mysql_query() 将空字节解释为查询的结束。如果查询包含二进制数据，则用 mysql_real_query 来代替 mysql_query()。

该查询必须由单个的 SQL 语句组成，并不应以分号（‘ ； ’）或‘ \g ’终结（‘ ； ’和‘ \g ’是 mysql 客户机程序的约定，而不是客户机库的约定。）

mysql_query() 运行成功时返回零，失败时返回非零值。一个成功的查询是服务器合法接

受的并且运行无错的查询。成功并不蕴涵着有关受影响或被返回的行数的任何内容。

■ int

```
mysql_real_query (MYSQL *conn, char *query_string, unsigned int
length);
```

如果给定一个指定为空终结串的查询，mysql_real_query() 将该查询发送到服务器中运行。该串可以包含二进制数据（包括空字节）。该查询的文本由 query_string 给出，长度由 length 指定。

该查询必须由单个的 SQL 语句组成，并不应以分号（‘ ； ’）或‘ \g ’终结（‘ ； ’和‘ \g ’是 mysql 客户机程序的约定，而不是客户机库的约定）。

mysql_real_query() 运行成功时返回零，失败时返回非零值。一个成功的查询是服务器合法接受的并且运行无错的查询。成功并不蕴涵着有关受影响或被返回的行数的任何内容。

■ int

```
mysql_select_db (MYSQL *conn, char *db_name);
```

选择由 dn_name 指定的数据库为当前数据库，它成为没有包含明确的数据库说明符的表引用的缺省数据库。您必须具有访问该数据库的许可权，否则， mysql_select_db() 将失败。mysql_select_db() 运行成功时返回零，不成功时返回非零值。

F.3.4 处理例程的结果集

当一个查询产生结果集时，本节的函数允许您检索该集合访问它的内容。 mysql_store_result() 和 mysql_use_result() 函数创建该结果集，并且在使用本节的任何其他函数之前必须调用这两个函数的任一个。表 F-7 对这两个函数进行了比较。

表F-7 mysql_store_result() 和 mysql_use_result() 的比较

mysql_store_result()	mysql_use_result()
结果集中的所有行都由mysql_store_result() 本身提取	mysql_use_result() 初始化该结果集，但服从mysql_fetch_row() 的行检索
使用更多内存；所有行都存储存客户机中	使用很少的内存；一次存储一行
速度慢，因为包括分配内存给整个结果集的开销	速度快，因为对当前行分配所需的内存
mysql_fetch_row() 中的 NULL 值表明结果集的结尾，决不是表明一个错误	mysql_fetch_row() 的 NULL 值表明结果集的结尾或一个错误，因为通信的失败可以破坏对当前记录的检索
mysql_num_rows()可以在调用mysql_store_result()之后的任何时候进行调用	mysql_num_rows() 仅当所有行已经提取后返回正确的行记录
mysql_affected_rows()是mysql_num_ rows()的同义词	不能使用 mysql_affected_rows()
结果集行的随机访问可能利用mysql_data_ seek()、mysql_row_seek()和 mysql_ row_tell()	没有对结果集的随机访问；行必须按照由服务器返回的顺序进行处理； mysql_data_seek()、mysql_row_seek()和mysql_row_tell() 不使用
根本不必调用 mysql_fetch_row(),因为所有的行已经被检索	必须调用 mysql_fetch_row() 提取所有的行，否则“剩余”行将漏入下一个查询的结果中，导致“不一致”错误。
表进行读锁定，因为不再需要提取数据行	表可以暂时读锁定，如果客户机在检索中的话，将试图修改表的客户机锁在外面
结果集MYSQL_FIELD结构的max_length成员实际在结果集中为列设置最长的值	max_length 不设置为任何有意义的值，因为它直到所有行都检索完后才知道

■ my_ulonglong

```
mysql_affected_rows (MYSQL *conn);
```

返回由最近的 DELETE、INSERT、REPLACE 或 UPDATE 查询修改的行数。对于这样的查询，mysql_affected_row() 在对 mysql_query() 成功调用之后立即被调用。还可以在发布一个有返回行的语句后调用该函数。这时，该函数的作用与 mysql_num_rows() 的一样，并遵从相同的约束条件（有意义的值），以及遵从附加的约束条件（在使用 mysql_use_result() 时 mysql_affected_rows() 无意义）。如果没有发布任何查询，或如果该查询返回行但没有任何行被选中，mysql_affected_rows() 则返回零。返回值大于零表明修改的（对于 DELETE、INSERT、REPLACE、UPDATE）或返回的（对于有返回行的查询）行数，返回值为 -1 表明有错，或者是在发布有返回值的查询之后、但在实际检索结果集之前您错误地调用了 mysql_affected_ rows()。因为 mysql_affected_rows() 返回一个无符号值，所以应该通过定义其结果为符号值来完成比较：

```
if ((long) mysql_affected_rows (conn) == -1)
    fprintf (stderr, "Error!\n");
```

如果您已经指明客户机应该返回与 UPDATE（）查询相配的行数，则 mysql_affected_rows() 将返回该值，而非实际已修改的行数（如果要修改的列与新值相同， MySQL 将不对行进行更新）。通过指定选项文件中的 return-found-rows，或者通过将 flags 参数中的 CLIENT_FOUND_ROWS 值传送到 mysql_real_connect()，可以选择该性能。

请参见“标量数据类型”有关打印 my_ulonglong类型值的说明。

■ void

```
mysql_data_seek (MYSQL_RES *res_set, unsigned int offset);
```

搜索结果集的特定行。offset 值的范围从 0 到 mysql_num_rows(res_set)-1。如果 offset 超出范围，则结果是不可预知的。

mysql_data_seek() 要求整个结果集已经检索，因此只有在该结果集通过 mysql_store_result() （而不是通过 mysql_use_result()）创建后才能使用。

■ MYSQL_FIELD*

```
mysql_fetch_field (MYSQL_RES *res_set);
```

返回包含关于结果集中某列的信息（元数据）的结构。在成功地执行有返回行的查询后，对 mysql_fetch_field() 的第一个调用返回有关第一列的信息。后续的调用将返回有关第一列的后续列信息，如果没有那么多的列，返回值为 NULL。

还可以调用 mysql_field_tell() 确定当前列的位置，或调用 mysql_field_seek() 选择在下一个 mysql_fetch_field() 调用中要返回的指定列。

下例检索第一个 MYSQL_FIELD，然后提取连续的列信息结构：

```
MYSQL_FIELD     *field;
unsigned int    i;

mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field (res_set);
    printf ("column %u: name = %s max_length = %lu\n",
            i, field->name, field->max_length);
}
```

■ MYSQL_FIELD *

```
mysql_fetch_fields (MYSQL_RES res_set);
```

返回结果集的所有列信息结构的一个数组。您可以按如下形式对它们进行访问：

```
MYSQL_FIELD     *field;
unsigned int    i;

field = mysql_fetch_fields (res_set);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    printf ("column %u: name = %s max_length = %lu\n",
            i, field[i].name, field[i].max_length);
}
```

将它与 mysql_fetch_field() 的例子进行比较。请注意，尽管两个函数都返回同一类型的值，但访问每个函数的这些值的语法稍有不同。

■ MYSQL_FIELD *

```
mysql_fetch_field_direct (MYSQL_RES *res_set, unsigned int field_num);
```

如果给定一个列的索引，则返回该列的信息结构。 field_num 值的范围从 0 到 mysql_num_fields() - 1。如果 field_num 超出范围，则该结果是不可预知的。

以下例子直接访问 MYSQL_FIELD 的结构：

```
MYSQL_FIELD     *field;
unsigned int    i;

for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field_direct (res_set, i);
    printf ("column %u: name = %s max_length = %lu\n",
            i, field->name, field->max_length);
}
```

mysql_fetch_field_direct() 在 MySQL 3.23 版本之前还不能完全地工作。

■ unsigned long *

```
mysql_fetch_lengths (MYSQL_RES *res_set);
```

返回指向一个 unsigned long 值的数组的指针，这些值表示结果集的当前行中列的长度。每当调用 mysql_fetch_row() 时，或长度与数据值不一致时，必须调用 mysql_fetch_lengths()。

NULL 值的长度为 0，但长度为零并不表示其本身是 NULL 值。因为空串的长度也为零，您必须检查该数据值是否是空指针来区别这两种情况。

下例示出当前行的长度和值，如果该值为 NULL 则打印“ NULL ”：

```
unsigned long *length;

length = mysql_fetch_lengths (res_set);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    printf ("length is %lu, value is %s\n",
            length[i], (row[i] != NULL ? row[i] : "NULL"));
}
```

mysql_fetch_lengths() 是在 MySQL 3.20.5 中引入的。在 MySQL 3.22.7 之前，

mysql_fetch_lengths() 的返回类型为 unsigned int。

■ MYSQL_ROW

```
mysql_fetch_row (MYSQL_RES *res_set);
```

返回指向结果集下一行的指针，该行以一个串数组表示（除 NULL 列值表示为 NULL 指针外）。该行的 i-th 值是该值数组的 i-th 成员。

所有数据类型的值（甚至是数值类型的）都以串形式返回。如果要想利用某个值完成一个数值数值计算，必须将其转换——例如用 atoi() 或 atof()。

当数据集中不再有行时，mysql_fetch_row() 返回 NULL（如果使用 mysql_use_result() 开始逐行的结果集检索，并且出现通信错误的话，mysql_fetch_row() 也返回 NULL。）

数据值是空终结串，但不应将含有二进制数据的值看作时空终结串。应将它们看成时计数串（要想确定列值的长度，调用 mysql_fetch_lengths()）。

以下代码显示出如何循环通过一行数据值，以及如何确定每个值是否为 NULL ：

```
MYSQL_ROW       row;
unsigned int    i;

while ((row = mysql_fetch_row (res_set)) != NULL)
{
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        printf ("column %u: value is %s\n",
                    i, (row[i] == NULL ? "NULL" : "not NULL"));
    }
}
```

为了确定列值的类型，使用存储在 MYSQL_FIELD 列信息结构中的该列的元数据，可以通过调用 mysql_fetch_field()、mysql_fields() 或 mysql_fetch_field_direct() 得到。

■ unsigned int

```
mysql_field_count (MYSQL *conn);
```

返回指定连接中的最近查询的列的数目。该函数通常是在 mysql_store_result() 或 mysql_use_result() 返回 NULL 时使用。mysql_field_count() 指示您结果集是否已经返回。返回值为 0 表明无结果集并无错误。非零值表明有所希望的列，或者是一个错误（因为没有返回任何内容）。

下例说明怎样为检测错误使用 mysql_field_count()：

```
res_set = mysql_store_result (conn);
if (res_set == NULL)    /* no result set was returned */
{
    /*
     * does the lack of a result set mean that an error
     * occurred or that no result set should be expected?
     */
    if (mysql_field_count (conn) > 0)
    {
        /*
         * a result set was expected, but mysql_store_result()
         * did not return one; this means an error occurred
         */
```

```
            printf ("Problem processing result set\n");
        }
        else
        {
            /*
             * a result set was not expected; query returned no data
             * (it was not a SELECT, SHOW, DESCRIBE, or EXPLAIN),
             * so just report number of rows affected by query
             */
            printf ("%lu rows affected\n",
                        (unsigned long) mysql_affected_rows (conn));
        }
    }
    else    /* a result set was returned */
    {
        /* ... process rows here ... */
        mysql_free_result (res_set);
    }
```

mysql_field_count() 是在 MySQL 3.22.24 中引入的。在该版本以前， mysql_numfields() 的作用与之相同。要想编写可使用任何 MySQL 版本的代码，应在使用 mysql_field_count() 的任何文件中包括以下代码段：

```
#if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
#define mysql_field_count mysql_num_fields
#endif
```

当源程序在旧的 MySQL 版本下编译时，此段代码将 mysql_field_count() 影射到 mysql_num_fields() 中。

■ MYSQL_FIELD_OFFSET

```
mysql_field_seek (MYSQL_RES *res_set, MYSQL_FIELD_OFFSET offset);
```

搜索由 offset 指定的列信息结构。对 mysql_fetch_field() 的下一次调用将返回该指定列的信息结构。offset不是列的索引，它是 MYSQL_FIELD_OFFSET 值，是从对 mysql_field_tell() 或 mysql_field_seek() 的调用中得到的。

要想复位到第一列，传递一个为 0 的 offset 值。

■ MYSQL_FIELD_OFFSET

```
mysql_field_tell (MYSQL_RES *res_set);
```

返回当前的列信息结构的偏移量。该值可传递到 mysql_field_seek() 中。

■ void

```
mysql_free_result (MYSQL_RES *res_set);
```

释放结果集使用的内存。必须为您使用的每个结果集调用 mysql_free_result()。通常，通过调用 mysql_store_result() 或 mysql_use_result() 生成结果集。但是，有些客户机函数间接地生成结果集，因此您也有责任释放那些集合。这些函数是 mysql_list_dbs()、mysql_list_fields()、mysql_list_processes() 和 mysql_list_tables()。

■ char*

```
mysql_info (MYSQL *conn);
```

返回一个串，该串包含有关最近执行的下列类型的查询所实现的效果的信息。该串的格式紧跟在每个查询后给出：

```
ALTER TABLE ...
    Records: 0 Duplicates: 0 Warnings: 0
INSERT INTO ... SELECT ...
    Records: 0 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),...
    Records: 0 Duplicates: 0 Warnings: 0
LOAD DATA ...
    Records: 0 Deleted: 0 Skipped: 0 Warnings: 0
UPDATE ...
    Rows matched: 0 Changed: 0 Warnings: 0
```

当然，这些数字将根据您执行的特定的查询而变化。

对于未在上述列表中出现的语句， mysql_info() 返回 NULL。仅当 INSERT INTO ... VALUES 中包含多于一个值的列表时， mysql_info() 对该语句返回非 NULL 值。

由 mysql_info() 返回的串是用服务器所使用的语言的，因此不要指望可以通过查找某些词对它进行语法分析。

■ my_ulonglong

mysql_insert_id (MYSQL conn);

返回由最近执行的给定连接上的查询所产生的 AUTO_INCREMENT 值。如果没有执行查询或者前一查询不产生 AUTO_INCREMENT 值，则返回零。这意味着您应该在一个期待产生新值的查询后立即调用 mysql_insert_id()。如果任何其他的查询在该查询和您要想使用该值的那一刻之间介入， mysql_insert_id() 的值将由介入的查询复位。

零返回值不同于任何合法的 AUTO_INCREMENT 值，因为这些值从 1 开始（例外：如果您创建 AUTO_INCREMENT 列然后插入一个直接量的负数，则序列将从该数字开始，并且最终达到作为该序列的合法成员的 0 。在这种情况下假定您知道正在做什么，因为 mysql_insert_id() 返回一个无符号数并且您要想玩弄这个返回值）。

请注意， mysql_insert_id() 的特点与 SQL 函数 LAST_INSERT_ID() 的不同。mysql_insert_id() 在客户机中维护并对每个查询进行设置。 LAST_INSERT_ID() 的值在服务器中维护并将在所有查询中保持。

由 mysql_insert_id() 返回的值是连接专用的，它不受其他连接中的 AUOT_INVREMENT 活动的影响。

请参阅F.2.1节“ 标量数据类型 ”中关于打印 my_ulonglong 类型值的说明。

■ unsigned int

mysql_num_fields (MYSQL_RES *res_set);

返回结果集的列数目。 mysql_num_rows() 常常用来循环通过该集合的当前行的列，如下例所示：

```
MYSQL_ROW       row;
unsigned int    i;

while ((row = mysql_fetch_row (res_set)) != NULL)
{
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        /* do something with row[i] here ... */
    }
}
```

在 MySQL 3.22.24 以前，mysql_num_fields() 还用来完成现在由 mysql_field_count() 所函数完成的功能，即，测试来自 mysql_store_result() 或 mysql_use_result() 中的 NULL 返回值是否表示有错。这就是为什么在旧的源程序代码中，有时将看到 mysql_num_fields() 正在用指向一个连接处理程序的指针进行调用，而不是用指向结果集的指针。 mysql_num_fileds() 通常可用于这两种方式。利用连接处理程序的 mysql_num_fields() 的使用方式现在不受赞成。取而代之，应该用 mysql_field_count() 来编写程序。在该函数的描述中说明了如何使用它，甚至是对于较旧的 MySQL 版本的。

■ my_ulonglong

```
mysql_num_rows (MYSQL_RES *res_set);
```

返回结果集中的行数。如果是用 mysql_store_result() 产生结果集，则可以在那以后任何时候调用 mysql_num_rows()：

```
if ((res_set = mysql_store_result (conn)) == NULL)
{
    /* mysql_num_rows() can be called now */
}
```

如果是用 mysql_use_result() 产生结果集，则 mysql_num_rows() 在您提取所有行之前不返回正确的值：

```
if ((res_set = mysql_use_result (conn)) == NULL)
{
    /* mysql_num_rows() cannot be called yet */
    while ((row = mysql_fetch_row (res_set)) != NULL)
    {
        /* mysql_num_rows() still cannot be called */
    }
    /* mysql_num_rows() can be called now */
```

请参阅“ 标量数据类型 ”一节关于打印 my_ulonglong 类型值的说明。

■ MYSQL_ROW_OFFSET

```
mysql_row_seek (MYSQL_RES *res_set, MYSQL_ROW_OFFSET offset);
```

搜索结果集中的一个特定行。 mysql_row_seek() 类似于 mysql_data_seek()，但 offset 值不是行号。offset 必须是一个从对 mysql_row_tell() 或 mysql_row_seek() 的调用中得到的值。

mysql_row_seek() 返回前一行的偏移量。

mysql_row_seek() 要求整个结果集已经被检索过，因此仅当该结果集是由 mysql_store_result() 而不是由 mysql_use_result() 创建时才能使用。

■ MYSQL_ROW_OFFSET

```
mysql_row_tell (MYSQL_RES *res_set);
```

返回表示结果集中当前行位置的偏移量。它不是行号，该值只能被传递到 mysql_row_seek() 中，而不传递到 mysql_data_seek() 中。

mysql_row_tell() 要求整个结果集已经检索，因此，仅当该结果集是由 mysql_store_result() 而不是由 mysql_use_result() 创建时才能使用。

■ MYSQL_RES *

```
mysql_store_result (MYSQL *conn);
```

它跟在一个成功查询的后面，返回结果集并将其存储在客户机中。如果该查询不返回任何数据

或者没有出现错误，则返回NULL。当mysql_store_result()返回NULL时，调用mysql_field_coun()或错误报告函数中的一个来确定结果集是否不是所期望的，或者是否出现了错误。

当处理完结果集后，将其传递到 mysql_free_result() 中进行释放。

请参阅表F-7 中的 mysql_store_result() 和 mysql_use_result() 的比较。

■ MYSQL_RES*

```
mysql_use_result (MYSQL *conn);
```

紧跟在一个运行成功的查询后面开始结果集的检索，但不检索任何数据行本身。必须调用 mysql_fetch_row() 来逐个地提取这些行。如果该查询不返回数据或出现错误，则返回NULL。当 mysql_use_result() 返回 NULL 时，调用 mysql_field_count() 或错误报告函数中的一个来确定结果集是否不是所期望的，或者是否出现了错误。

当处理完结果集后，将其传递到 mysql_free_result() 中进行释放。

请参阅表F-7 中的 mysql_store_result() 和 mysql_use_result() 的比较。

mysql_store_result() 和 mysql_use_result() 都是用来检索结果集的，但它们影响您使用其他结果集处理函数。

F.3.5 信息例程

这些函数提供关于客户机、服务器、协议版本和当前连接的信息。

■ char *

```
mysql_get_client_info (void);
```

返回一个空终结串，它描述客户机的版本，如“ 3.22.25 ”。

■ char *

```
mysql_get_host_info (MYSQL *conn);
```

返回一个空终结串，它描述当前的连接，如“ Loaclhost via UNIX socket ”或“ your.host.com via TCP/IP ”。

■ unsigned int

```
mysql_get_proto_info (MYSQL *conn);
```

返回一个数字，它指明用于当前连接的客户机 /服务器协议的版本。

■ char *

```
mysql_get_server_info (MYSQL *conn);
```

返回一个空终结串，它描述服务器的版本，如“ 3.22.25-log ”。跟在服务器版本号后面的后缀是 -log（日志是开启的）、-debug（服务器正在以调试方式运行），或 -demo（服务器正以演示方式运行）。

■ char *

```
mysql_stat (MYSQL *conn);
```

返回一个空终结串，它包含服务器状态信息或 NULL（如果出现错误的话）。该串的格式根据变化而改变。通常它如下所示：

```
Uptime: 864034  Threads: 1  Questions: 32736  Slow queries: 50  Opens: 1428
Flush tables: 1  Open tables: 61
```

这些值的解释如下：

- Uptime 是服务器已经打开的秒数。
- Threads 是当前运行在服务器上的线程的数目。
- questions 是服务器已经执行的查询数目。
- Slow queries 是所花费时间比服务器的 long_query_time 参数长的查询数目。
- Opens 是服务器已经打开的表的数目。
- Flush tables 是已经执行 FLUSH、REFRESH 和 RELOAD 命令的次数。
- Open tables 是服务器当前已经打开的表的数目。

由 mysql_stat() 返回的信息与 mysqladmin status 命令报告的信息相同。（您认为 mysqladmin 应该在哪里得到这些信息？）

- unsigned long

```
mysql_thread_id (MRSQL*Conn);
```

返回服务器与当前连接相关的线程数。可以使用该数目作为 mysql_kill() 的标识符。

在您需要该值之前执行 mysql_thread_id() 不是好主意。如果您得到该值并存储它，打算在后面使用，则该值可能会不正确。这种情况可能发生在连接断开后又重新建立时（如用 mysql_ping()），因为服务器将分配一个新的线程标识符。

F.3.6 管理例程

本节的函数允许您控制服务器运作方式。

- int

```
mysql_kill (MYSQL *conn, unsigned long thread_id);
```

取消由 thread_id 标识的服务器线程。

如果有 PROCESS 权限，则可以取消任何线程。否则，只能取消属于自己的线程。

mysql_kill() 在运行成功时返回零，失败时返回非零值。

- int

```
mysql_refresh (MYSQL *conn, unsigned int options);
```

该函数在效果上类似于 SQL FLUSH 语句，除了您可以指示服务器立即刷新几种类型的事物以外。oprions 值应该是表F-8 所示的一个或多个值。

表F-8 mysql_refresh() 选项

选 项 值	服务器采取的行动
REFRESH_GRANT	重新加载授权表的内容
REFRESH_LOG	开始新的常规和更新日志（它们通常是打开的）
REFRESH_TABLES	关闭所有打开的表
REFRESH_HOSTS	刷新主机的高速缓存
REFRESH_STATUS	将状态变量复位到零

表F-8 中的选项是比特位值，因此可以用‘ | ’或‘ + ’将它们组合，例如， REFRESH_LOG| REFRESH_TABLES 或 REFRESH_LOG+REFRESH_TABLES。

有关刷新操作的详细信息，请参阅附录 D中对 FLUSH 语句的描述。

mysql_refresh() 运行成功时返回零，不成功时返回非零值。

- int

```
mysql_shutdown (MYSQL *conn);
```

指示服务器关闭。您必须具有 SHUTDOWN 权限才能进行。

mysql_shutdown() 运行成功时返回零，不成功返回非零值。

F.3.7 调试例程

这些函数允许您产生该连接的客户机端或服务器端的调试信息。这需要编译 MySQL 以支持调试（在您配置 MySQL 分发包时使用 --with-debug 选项）。

■ void

```
mysql_debug (char *debug_string);
```

用串 debug_string 执行 DBUG_PUSH 选项。该串的格式在 MySQL 参考手册中介绍。

要使用 mysql_debug()，客户机库必须用调试支持环境进行编译。

■ int

```
mysql_dump_debug_info (MYSQL *conn);
```

指示服务器将调试信息写到日志中。您必须具有 PROCESS 权限才能进行。

mysql_dump_debug_info() 运行成功时返回零，不成功返回非零值。

F.3.8 不受赞成的例程

由于有更好的方法可用来进行相同的工作，因此，MySQL 客户机库中包括大量的目前不受赞成的函数。大多数这些函数都通过将等价的查询传递到 mysql_query() 中被取代。例如，mysql_create_db(“*db_name*”) 可以用下列调用取代：

mysql_query (conn,“CREATE DATABASE *db_name*”)

有几个函数，如 mysql_connect() 和 mysql_eof()，都是不赞成的，因为它们已经由可以做更多工作的或提供更多信息的函数取代。

随着时间的推移，MySQL 对 SQL 语句理解得越多，不受赞成的函数就越多。例如，当增加 SQL FLUSH PRIVILEGES 语句时，mysql_reload() 不受赞成。下列描述指明了每个函数不受赞成的 MySQL 版本以及执行每个函数的现在更好的方法。如果客户机库比所列出的 MySQL 版本还旧，当然仍必须使用不赞成的函数。

如果想要对未来打算，应该避免使用本节所列出的所有函数。这些中的一部分或全部将在 MySQL 4.0 中消失。

■ MYSQL *

```
mysql_connect (MYSQL *conn, char *host_name,
          char *user_name, char *password);
```

这是 mysql_real_connect() 的前身。事实上，现在它作为对 mysql_real_connect() 的调用被实施。

该函数自 MySQL 3.22.0 以来不受赞成。

■ int

```
mysql_create_db (MYSQL *conn, char *db_name);
```

用 db_name 指定的名字创建数据库。这个功能现在可以通过利用 mysql_query() 发布

CREATE DATABASE 命令来完成。

mysql_create_db() 运行成功时返回零，不成功返回非零值。

该功能自 MySQL 3.21.15 以来一直不受赞成。

■ int

```
mysql_drop_db (MYSQL *conn, char *db_name);
```

删除 db_name 指定的数据库。这个功能现在可通过用 mysql_query() 发布 DROP DATABASE 命令来进行。

mysql_drop_db() 运行成功时返回零，失败返回非零值。

该函数自 MySQL 3.21.15 以来不受赞成。

■ my_bool

```
mysql_eof (MYSQL_RES *res_set);
```

如果已经到达结果集的末尾返回非零值，如果出错返回零。在您使用开始结果集检索的musql_use_result() 和同时提取数据行的 mysql_fetch_row() 的组合时使用 mysql_eof()。对于mysql_use_result()，mysql_fetch_row() 的 NULL 返回值意味着或者到达集合的末尾或者是出错。mysql_eof() 在两个结果之间进行辨别。

mysql_errno() 和 mysql_error() 现在可用来达到相同的效果，尽管它们实际返回更多的信息（它们指出已经出现的任何错误的原因，而不是只简单地指出是否有错）。

该函数自 MySQL 3.21.7 以来一直不受赞成。

■ MYSQL_RES *

```
mysql_list_dbs (MYSQL *conn, char *wild);
```

返回列出服务器中数据库名的结果集或 NULL（如果有错）。该列表中包含与 wild 指明的 SQL 常规表达式（可能包含通配符‘ % ’和‘ _ ’）相配的所有数据库，或者 wild 为 NULL 时的所有数据库。您有责任调用 mysql_free() 释放结果集。

mysql_list_dbs() 产生的列表可通过利用 mysql_query() 执行 SHOW DATABASE 命令然后处理该结果集来获得。

该函数自 MySQL 3.22.0 以来一直不受赞成。

■ MYSQL_RES *

```
mysql_list_fields (MYSQL *conn, char *tbl_name, char *wild);
```

返回在给定表中显示出列名的结果集或 NULL（如果有错的话）。该列表包含与 wild 指明的 SQL 常规表达式（可能包含通配符‘ % ’和‘ _ ’）相配的所有列名，或者 wild 为 NULL 时的所有列。您有责任调用 mysql_free() 释放结果集。

mysql_list_field() 产生的列表可通过利用 mysql_query() 执行 SHOW COLUMNS 命令然后处理该结果集来获得。

该函数自 MySQL 3.22.0 以来一直不受赞成。

■ MYSQL_RES *

```
mysql_list_processes (MYSQL *conn);
```

返回包含在服务器中运行的进程列表的结果集或 NULL（如果有错的话）。如果具有 PROCESS 特权，该列表将包含所有服务器进程。如果没有，该列表只包含属于自己的进程。您有责任调用 mysql_free() 释放结果集。

mysql_list_processes() 产生的列表可通过利用 mysql_query() 执行 SHOW PROCESSLIST 查询然后处理该结果集来获得。

该函数自 MySQL 3.22.0 以来一直不受赞成。

■ MYSQL_RES *

```
mysql_list_tables (MYSQL *conn, char *wild);
```

返回列出当前数据库的表的结果集或 NULL（如果有错的话）。该列表包含与 wild 指明的 SQL 常规表达式（可能包含通配符‘ % ’和‘ _ ’）相匹配的所有表，或者 wild 为 NULL 时的所有表。您有责任调用 mysql_free() 释放结果集。

mysql_list_table() 产生的列表可通过利用 mysql_query() 执行 SHOW TABLES 命令然后处理该结果集来获得。

该函数自 MySQL 3.22.0 以来一直不受赞成。

■ int

```
mysql_reload (MYSQL *conn);
```

指示服务器重新加载授权表。这个功能现在可以通过利用 mysql_query() 发布 FLUSH PRIVILEGES 查询来进行。必须具有 RELOAD 权限才能使用 mysql_reload().

mysql_reload() 运行成功时返回零，失败返回非零值。

该函数自 MySQL 3.21.9 以来一直不受赞成。