mysql api for C 详细介绍(1)
2009-11-22 16:33
C API 代码是随 MySQL 分发的，它被包含在 mysqlclient 库且允许 C 程序存取一个数据库。在 MySQL 源代码分发中的很多客户是用 C 编写的。如果你正在寻找演示怎样使用 C API 的例子，看一下这些客户程序。
大多数其他客户 API(除了 Java 的所有)都使用 mysqlclient 库与 MySQL 服务器通信。这意味着，例如，你能利用很多被其他客户程序使用的同一环境变量，因为他们从库中引用。对这些变量的一张表，见 12.1 不同的 MySQL 程序的概述。
客户有一个最大通讯缓冲区大小。初始分配的缓冲区大小(16K 字节) 自动地增加到最大尺寸(缺省的最大值是 24M)。因为缓冲区大小只是按保证需求而被增加，简单地增加缺省的最大限制并不造成更多被消耗。该尺寸检查主要是一个对错误的查询和通讯包的检查。
通讯缓冲区必须足够大以便一个单独的 SQL 语句(对客户-服务器传输)和一行返回的数据 (对服务器-客户传输)。每个线程的通讯缓冲区被动态扩大到最大限制来处理任何查询或行。例如，如果你包含大到 16M 数据的 BLOB 值，你必须有一个至少 16M 通讯缓冲区限制(在服务器和客户两端)。客户的缺省最大值是 24M，但是在服务器端的缺省最大值是 1M。你可以在服务器启动时通过改变 max_allowed_packet 参数的值来改变它。见 10.2.3 调节服务器参数。MySQL 服务器在每个查询后缩小每个通讯缓冲区到 net_buffer_length 个字节。对客户，与一个连接相关的缓冲区的大小没被减少，直到连接被关闭，在此时客户内存被回收。
如果你用线程的编程，你应该用--with-thread-safe-client 编译 MySQL C API，这将使 C API 线程对每个连接更安全。你可以让 2 个线程共享相同的连接，只要如果你做下列事情：
两个线程不能同时在同一个连接上发送查询到 MySQL。特别是你必须保证在一个 mysql_query()和 mysql_store_result()之间没有其他线程正在使用同一个连接。
许多线程能存取用 mysql_store_result()检索出来的不同结果集合。
如果你使用 mysql_use_result，你必须保证没有其他线程在同一个连接上正在询问任何东西，直到结果集合被关闭。
20.2 C API 数据类型
MYSQL
这个结构表示对一个数据库连接的句柄，它被用于几乎所有的 MySQL 函数。
MYSQL_RES
这个结构代表返回行的一个查询的(SELECT, SHOW, DESCRIBE, EXPLAIN)的结果。从查询返回的信息在本章下文称为结果集合。
MYSQL_ROW
这是一个行数据的类型安全(type-safe)的表示。当前它实现为一个计数字节的字符串数组。（如果字段值可能包含二进制数据，你不能将这些视为空终 止串，因为这样的值可以在内部包含空字节) 行通过调用 mysql_fetch_row()获得。
MYSQL_FIELD
这个结构包含字段信息，例如字段名、类型和大小。其成员在下面更详细地描述。你可以通过重复调用 mysql_fetch_field()对每一列获得 MYSQL_FIELD 结构。字段值不是这个结构的部分；他们被包含在一个 MYSQL_ROW 结构中。
MYSQL_FIELD_OFFSET
这是一个相对一个 MySQL 字段表的偏移量的类型安全的表示。(由 mysql_field_seek()使用。)偏移量是在一行以内的字段编号，从 0 开始。
my_ulonglong

该类型用于行编号和 mysql_affected_rows()、mysql_num_rows()和 mysql_insert_id()。这种类型提供 0 到 1.84e19 的一个范围。在一些系统上，试图打印类型 my_ulonglong 的值将不工作。为了打印出这样的值，将它变换到 unsigned long 并且使用一个%lu 打印格式。例如：

```
printf (Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
```

MYSQL_FIELD 结构包含列在下面的成员：

char * name

字段名，是一个空结尾的字符串。

char * table

包含该字段的表的名字，如果它不是可计算的字段。对可计算的字段，table 值是一个空字符串。

char * def

这字段的缺省值，是一个空结尾的字符串。只要你使用，只有你使用 mysql_list_fields()才可设置它。

enum enum_field_types type

字段类型。type 值可以是下列之一：类型值 类型含义

FIELD_TYPE_TINY TINYINT 字段

FIELD_TYPE_SHORT SMALLINT 字段

FIELD_TYPE_LONG INTEGER 字段

FIELD_TYPE_INT24 MEDIUMINT 字段

FIELD_TYPE_LONGLONG BIGINT 字段

FIELD_TYPE_DECIMAL DECIMAL 或 NUMERIC 字段

FIELD_TYPE_FLOAT FLOAT 字段

FIELD_TYPE_DOUBLE DOUBLE 或 REAL 字段

FIELD_TYPE_TIMESTAMP TIMESTAMP 字段

FIELD_TYPE_DATE DATE 字段

FIELD_TYPE_TIME TIME 字段

FIELD_TYPE_DATETIME DATETIME 字段

FIELD_TYPE_YEAR YEAR 字段

FIELD_TYPE_STRING 字符串(CHAR 或 VARCHAR)字段

FIELD_TYPE_BLOB BLOB 或 TEXT 字段(使用 max_length 决定最大长度）

FIELD_TYPE_SET SET 字段

FIELD_TYPE_ENUM ENUM 字段

FIELD_TYPE_NULL NULL- 类型字段

FIELD_TYPE_CHAR 不推荐；使用 FIELD_TYPE_TINY 代替

你可以使用 IS_NUM()宏来测试字段是否有一种数字类型。将 type 值传给 IS_NUM()并且如果字段是数字的，它将计算为 TRUE：

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

unsigned int length

字段宽度，在表定义中指定。

unsigned int max_length

对结果集合的字段的最大宽度(对实际在结果集合中的行的最长字段值的长度)。如果你使用 mysql_store_result() 或 mysql_list_fields()，这包含字段最大长度。如果你使用 mysql_use_result()，这个变量的值是零。

unsigned int flags
字段的不同位标志。flags 值可以是零个或多个下列位设置：标志值 标志含义
NOT_NULL_FLAG 字段不能是 NULL
PRI_KEY_FLAG 字段是一个主键的一部分
UNIQUE_KEY_FLAG 字段是一个唯一键的一部分
MULTIPLE_KEY_FLAG 字段是一个非唯一键的一部分。
UNSIGNED_FLAG 字段有 UNSIGNED 属性
ZEROFILL_FLAG 字段有 ZEROFILL 属性
BINARY_FLAG 字段有 BINARY 属性
AUTO_INCREMENT_FLAG 字段有 AUTO_INCREMENT 属性
ENUM_FLAG 字段是一个 ENUM（不推荐）
BLOB_FLAG 字段是一个 BLOB 或 TEXT（不推荐）
TIMESTAMP_FLAG 字段是一个 TIMESTAMP（不推荐）
BLOB_FLAG、ENUM_FLAG 和 TIMESTAMP_FLAG 标志的使用是不推 荐的，因为他们指出字段的类型而非它的类型属性。对 FIELD_TYPE_BLOB 、 FIELD_TYPE_ENUM 或 FIELD_TYPE_TIMESTAMP，最好是测试 field->type。下面例子演示了一个典型的 flags 值用法：

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

你可以使用下列方便的宏决来确定 flags 值的布尔状态：
IS_NOT_NULL(flags) 真，如果该字段被定义为 NOT NULL
IS_PRI_KEY(flags) 真，如果该字段是一个主键
IS_BLOB(flags) 真，如果该字段是一个 BLOB 或 TEXT（不推荐；相反测试 field->type）
unsigned int decimals
对数字字段的小数位数。
20.3 C API 函数概述
在 C API 中可用的函数列在下面，并且在下一节更详细地描述。见 20.4 C API 函数描述。
mysql_affected_rows() 返回被最新的 UPDATE, DELETE 或 INSERT 查询影响的行数。
mysql_close() 关闭一个服务器连接。
mysql_connect() 连接一个 MySQL 服务器。该函数不推荐；使用 mysql_real_connect()代替。
mysql_change_user() 改变在一个打开的连接上的用户和数据库。
mysql_create_db() 创建一个数据库。该函数不推荐；而使用 SQL 命令 CREATE DATABASE。
mysql_data_seek() 在一个查询结果集合中搜寻一任意行。
mysql_debug() 用给定字符串做一个 DBUG_PUSH。
mysql_drop_db() 抛弃一个数据库。该函数不推荐；而使用 SQL 命令 DROP DATABASE。
mysql_dump_debug_info() 让服务器将调试信息写入日志文件。
mysql_eof() 确定是否已经读到一个结果集合的最后一行。这功能被反对；mysql_errno()或 mysql_error()可以相反被使用。
mysql_errno() 返回最近被调用的 MySQL 函数的出错编号。
mysql_error() 返回最近被调用的 MySQL 函数的出错消息。
mysql_escape_string() 用在 SQL 语句中的字符串的转义特殊字符。
mysql_fetch_field() 返回下一个表字段的类型。
mysql_fetch_field_direct () 返回一个表字段的类型，给出一个字段编号。
mysql_fetch_fields() 返回一个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() 从结果集合中取得下一行。
mysql_field_seek() 把列光标放在一个指定的列上。
mysql_field_count() 返回最近查询的结果列的数量。
mysql_field_tell() 返回用于最后一个 mysql_fetch_field()的字段光标的位置。
mysql_free_result() 释放一个结果集合使用的内存。
mysql_get_client_info() 返回客户版本信息。
mysql_get_host_info() 返回一个描述连接的字符串。
mysql_get_proto_info() 返回连接使用的协议版本。
mysql_get_server_info() 返回服务器版本号。
mysql_info() 返回关于最近执行得查询的信息。
mysql_init() 获得或初始化一个 MYSQL 结构。
mysql_insert_id() 返回有前一个查询为一个 AUTO_INCREMENT 列生成的 ID。
mysql_kill() 杀死一个给定的线程。
mysql_list_dbs() 返回匹配一个简单的正则表达式的数据库名。
mysql_list_fields() 返回匹配一个简单的正则表达式的列名。
mysql_list_processes() 返回当前服务器线程的一张表。
mysql_list_tables() 返回匹配一个简单的正则表达式的表名。
mysql_num_fields() 返回一个结果集合重的列的数量。
mysql_num_rows() 返回一个结果集合中的行的数量。
mysql_options() 设置对 mysql_connect()的连接选项。
mysql_ping() 检查对服务器的连接是否正在工作，必要时重新连接。
mysql_query() 执行指定为一个空结尾的字符串的 SQL 查询。
mysql_real_connect() 连接一个 MySQL 服务器。
mysql_real_query() 执行指定为带计数的字符串的 SQL 查询。
mysql_reload() 告诉服务器重装授权表。
mysql_row_seek() 搜索在结果集合中的行，使用从 mysql_row_tell()返回的值。
mysql_row_tell() 返回行光标位置。
mysql_select_db() 连接一个数据库。
mysql_shutdown() 关掉数据库服务器。
mysql_stat() 返回作为字符串的服务器状态。
mysql_store_result() 检索一个完整的结果集合给客户。
mysql_thread_id() 返回当前线程的 ID。
mysql_use_result() 初始化一个一行一行地结果集合的检索。
为了连接服务器，调用 mysql_init()以初始化一个连接处理器，然后用该处理器调用 mysql_real_connect()(还有其他信息例如主机名、用户名和口令)。当你用该连接完成工作后，调用 mysql_close()终止它。
当一个连接活跃时，客户可以用 mysql_query()或 mysql_real_query()将 SQL 查询发送到服务器。两者的差别是 mysql_query()期望查询作为一个空结尾的字符串来指定而 mysql_real_query()期望一个计数的字符串。如果字符串包含二进制数据(它可以包括空字节)，你必须使用 mysql_real_query()。
对与每个非--SELECT 查询(例如，INSERT、UPDATE、DELETE 等)，你可以调用 mysql_affected_rows()知道有多少行受到影响(改变)。

对于 SELECT 查询，你作为一个结果集合来检索选择的行。（注意一些语句是类 SELECT 的，他们返回行。这些包括 SHOW、DESCRIBE 和 EXPLAIN。他们应该象 SELECT 语句相同的方式来对待。）

对客户，有两种方法处理结果集合。一种方法是通过调用 mysql_store_result()立刻检索全部结果。该函数从服务器获得查询返回的所有行，并将他们存储在客户端。第二种方法是对客户通过调用 mysql_use_result()初始化一个一行一行地结果集合的检索。该函数初始化检索，但是实际上不从服务器获得任何行。

在两种情况中，你通过 mysql_fetch_row()存取行。用 mysql_store_result()、mysql_fetch_row()储存取已经从服务器被取出的行。用 mysql_use_result()、mysql_fetch_row()实际上从服务器检索行。调用 mysql_fetch_lengths()可获 得关于每行中数据值尺寸的信息。

在你用完一个结果集合以后，调用 mysql_free_result()释放由它使用的内存。

两种检索机制是互补的。客户程序应该选择最适合他们的要求的途径。在实践中，客户通常更愿意使用 mysql_store_result()。

mysql_store_result()的一个优点是既然行均被客户取到，你不仅能顺序 存取行，你也能 mysql_data_seek()或 mysql_row_seek()在结果集合中前后移动以改变在结果集合中的当前行位置。你也能通过 调用 mysql_num_rows()知道有多少行。另一方面，mysql_store_result()的内存需求对较大结果集合可能很高，并且你最可 能遇到 out-of-memory 情况。

mysql_use_result()的一个优点是客户为结果集合需要较少的内存，因为它 一次只是维持一行(并且因为有较少的分配开销，mysql_use_result()能更快些)。缺点是你必须尽快处理每一行以避免困住服务器，你不必再 结果集合中随意存取行(你只能顺序存取行)，而且你不知道在结果集合中有多少行，直到你检索全部结果。还有，你必须检索出所有行，即使你在检索中途确定你 已找到了想寻找的信息。

API 使得客户正确应答查询成为可能(仅检索必要的行)，不用知道查询是否是一个 SELECT。你可以通过在 mysql_query()（或 mysql_real_query())之后调用 mysql_store_result()做 到。如果结果集合调用成功并且查询是一个 SELECT，你能读取行。如果结果集合调用，调用 mysql_field_count()确定结果是否是实际期 望的。如果 mysql_field_count()返回 0，查询没有返回数据(表明它是一个 INSERT、UPDATE、DELETE 等)，所以不期望返 回行。如果 mysql_field_count()是非零，查询应该有返回行，但是没有。这表明查询是一个失败的 SELECT。见 mysql_field_count()如何能做到的例子的描述。

mysql_store_result()和 mysql_use_result()都允许 你获得有关组成结果集合的字段的信息(字段数量、他们的名字和类型等等)。你可以通过重复调用 mysql_fetch_field()在行中顺序存取字段 信息，或调用 mysql_fetch_field_direct()存取行中的字段编号。当前字段光标位置可以通过调用 mysql_field_seek()改变，设置字段光标影响到后续 mysql_fetch_field()调用。你也能通过调用 mysql_fetch_fields()马上获得字段信息。

对于检测和报告错误，MySQL 借助于 mysql_errno()和 mysql_error()函数提供错误信息的存取。他们返回最近调用的可能成功或失败的函数的错误代码或错误消息，允许你确定何时发生一个错误和什么错误。

20.4 C API 函数描述

在下面的描述中，一个 NULL 参数或返回值含义是在 C 编程语言环境的 NULL，不是一个 MySQL NULL 值。

返回一个值的函数一般返回一个指针或一个整数。除非另外指定，返回一个指针的函数返回一个 非 NULL 值表明成功，或一个 NULL 值表明一个错误，而返回一个整数的函数返回零表示成功，或非零表示一个错误。注意，“非零”只有这个含义。除非函数描 述另外说明，不要测试一个零以外的其他值：

```
if (result)                    /* 正确 */
    ... error ...
if (result < 0)                /* 不正确 */
    ... error ...
if (result == -1)              /* 不正确 */
    ... error ...
```

当函数返回一个错误时，函数描述的错误小节列出错误可能的类型。你可以调用 mysql_errno()找出发生了这些重的哪一个。错误的字符串表示可以调用 mysql_error()获得。

20.4.1 mysql_affected_rows()

my_ulonglong mysql_affected_rows(MYSQL *mysql)

20.4.1.1 说明

返回受到最后一个 UPDATE、DELETE 或 INSERT 查询影响(变化)的行数。可以在针对 UPDATE、DELETE 或 INSERT 语句的 mysql_query()之后立即调用。对于 SELECT 语句，mysql_affected_rows()的功能于 mysql_num_rows() 相同。

mysql_affected_rows()目前以一个宏(macro)来实现。

20.4.1.2 返回值

大于零的一个整数表示受到影响或检索出来的行数。零表示没有匹配查序中 WHERE 子句的记录或目前还没有查询被执行。-1 表示查询返回一个错误，或对于一 个 SELECT 查询，mysql_affected_rows()在调用 mysql_store_result()之前被调用。

20.4.1.3 错误

没有。

20.4.1.4 范例

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%d products updated",mysql_affected_rows(&mysql));
```

20.4.2 mysql_close()

void mysql_close(MYSQL *mysql)

20.4.2.1 说明

关闭一个以前打开了的连接。如果句柄由 mysql_init()或 mysql_connect()自动分配，mysql_close()也释放被 mysql 指向的连接句柄。

20.4.2.2 返回值

没有。

20.4.2.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务者的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.3 mysql_connect()

MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)

20.4.3.1 说明

该函数不推荐使用，而更好使用 mysql_real_connect()。
mysql_connect() 试图建立一个对运行在 host 的一个 MySQL 数据库引擎的连接。mysql_connect()必须在你能执行其他 API 函数之前成功地完成，除了 mysql_get_client_info()。参数的含义与 mysql_connect()相应的参数相同，不同的是连接参数可以是 NULL。在这种情况下，C API 自动为连接结构分配内存，并且当你调用 mysql_close()，释放它。这种方法的缺点是如果连接失败，你不能检索出一条错误消息。（为了从 mysql_errno()或 mysql_error()得到错误信息，你必须提供一个有效的 MYSQL 指针。）
20.4.3.2 返回值
同 mysql_real_connect()。
20.4.3.3 错误
同 mysql_real_connect()。

20.4.4mysql_change_user()
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)
20.4.4.1 说明
改变用户并且使得由 db 指定数据库成为由 mysql 指定的连接上的缺省(当前)数据库。在随后的查询中，这个数据库是不包括一个明确的数据库指定符的表引用的缺省值。
这个函数功能在 MySQL 3.23.3 中引入。
除非连接的用户能被认证或如果他没有权限使用数据库，mysql_change_user()失败。在这种情况下，用户和数据库都没被改变。
如果你不想有一个缺省数据库，db 参数可以被设置为 NULL。
20.4.4.2 返回值
成功，零。如果发生一个错误发生，非零。
20.4.4.3 错误
与你能从 mysql_real_connect()得到的相同。
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务者关闭了。
CR_SERVER_LOST
对服务者的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
ER_UNKNOWN_COM_ERROR
MySQL 服务器未实现这个命令(可能是一个老的服务器）
ER_ACCESS_DENIED_ERROR
用户或口令错误。
ER_BAD_DB_ERROR
数据库不存在。
ER_DBACCESS_DENIED_ERROR
用户没有数据库的存取权利。
ER_WRONG_DB_NAME
数据库名字太长。

20.4.4.4 范例

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
fprintf(stderr, "Failed to change user. Error: %s\n",
              mysql_error(&mysql));
}
```

20.4.5 mysql_create_db()

int mysql_create_db(MYSQL *mysql, const char *db)

20.4.5.1 说明

创建由 db 参数命名的数据库。

这个函数不推荐，而最好使用 mysql_query()发出一条 SQL CREATE DATABASE 语句。

20.4.5.2 返回值

如果数据库成功地被创造，零。如果发生一个错误，非零。

20.4.5.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务者的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.5.4 范例

```
if(mysql_create_db(&mysql, "my_database"))
{
fprintf(stderr, "Failed to create new database. Error: %s\n",
              mysql_error(&mysql));
}
```

20.4.6 mysql_data_seek()

void mysql_data_seek(MYSQL_RES *result, unsigned long long offset)

20.4.6.1 说明

在一个查询结果集合中定位任意行。这要求结果集合结构包含查询的全部结果，这样 mysql_data_seek()可以仅需与 mysql_store_result()一起使用，不是与 mysql_use_result()。

偏移量应该是从 0 到 mysql_num_rows(result)-1 范围的一个值。

20.4.6.2 返回值

无。

20.4.6.3 错误

无。

20.4.7 mysql_debug()

void mysql_debug(char *debug)

20.4.7.1 说明

用一个给定字符串做一个 DBUG_PUSH。mysql_debug()使用 Fred Fish 调试库。为了使用这个函数，你必须编译客户库以支持调试。见 G.1 调试一个 MySQL 服务器和节 G.2 调试一个 MySQL 客户。

20.4.7.2 返回值

无。

20.4.7.3 错误

无。

20.4.7.4 范例

下面所示的调用使得客户库在客户机器上的“/tmp/client.trace”中产生一个跟踪文件：

```
mysql_debug("d:t:O,/tmp/client.trace");
```

20.4.8 mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

20.4.8.1 说明

抛弃由 db 参数命名的数据库。

这个函数不推荐，而最好使用 mysql_query()发出一条 SQL DROP DATABASE 语句。

20.4.8.2 返回值

如果数据库成功地被破抛弃，零。如果发生一个错误，非零。

20.4.8.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务者的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.8.4 范例

```
if(mysql_drop_db(&mysql, "my_database"))
fprintf(stderr, "Failed to drop the database: Error: %s\n",
        mysql_error(&mysql));
```

20.4.9 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

20.4.9.1 说明

指示服务者将一些调试信息写入日志文件。连接的用户对此必须有 precess 权限才能工作。

20.4.9.2 返回值

如果命令成功，零。如果发生一个错误，非零。

20.4.9.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务者关闭了。

CR_SERVER_LOST

对服务器的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.10 mysql_eof()

my_bool mysql_eof(MYSQL_RES *result)

20.4.10.1 说明

这个函数不推荐，而使用 mysql_errno()或 mysql_error()。

mysql_eof()确定是否已经读到了一个结果集合的最后一行。

如果你从成功的 mysql_store_result()调用获得一个结果集合，客户程序用一个操作收到全部集合。在这种情况下，从 mysql_fetch_row()返回一个 NULL 总是意味着已经到达了结果集合的尾部，没必要调用 mysql_eof()。

在另一方面，如果你使用 mysql_use_result()初始化一个结果集合的检索， 该集合的行随着你重复调用 mysql_fetch_row()一个一个地从服务器获得。因为在这个过程中在连接上可能发生一个错误，从 mysql_fetch_row()返回一个 NULL 值并不意味着集合正常到达了尾部。在这种情况下，你能使用 mysql_eof()确定发生了什么。如 果到达结果集合的尾部，mysql_eof()返回非零值，并且如果发生一个错误，返回零。

在时间上，mysql_eof()先于标准 MySQL 错误函数 mysql_errno() 和 mysql_error()。因为这些错误函数提供相同的信息，他们的使用更好 mysql_eof()，它现在不建议使用。（事实上，他们提供更多的信 息，因为 mysql_eof()值返回一个布尔值，而错误函数指出当发生错误时的出错原因。）

20.4.10.2 返回值

如果发生一个错误，零。如果到达结果集合的结束，非零。

20.4.10.3 错误

无。

20.4.10.4 范例

下列例子显示你必须如何使用 mysql_eof()：

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

然而，你可以用标准 MySQL 错误函数完成同样的效果：

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql)) // mysql_fetch_row() failed due to an error
{
```

```
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

20.4.11 mysql_errno()

unsigned int mysql_errno(MYSQL *mysql)

20.4.11.1 说明

对于由 mysql 指定的连接，mysql_errno()返回最近调用的可能成功或失败的 API 函数的错误代码。返回值零意味着没有错误发生。客户错误消 息编号列出在 MySQL“errmsg.h”头文件中。服务器错误消息编号列出在“mysqld_error.h”中。

20.4.11.2 返回值：

一个错误代码值。如果没有错误发生，零。

20.4.11.3 错误

无。

20.4.12 mysql_error()

char *mysql_error(MYSQL *mysql)

20.4.12.1 说明

对于由 mysql 指定的连接，mysql_errno()返回最近调用的可能成功或失败的 API 函数的错误代码。如果没有错误发生，返回空字符串("")。这意味着下列两个测试是等价的：

```
if(mysql_errno(&mysql))
{
    // an error occurred
}
if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

客户错误消息的语言可通过重新编译 MySQL 客户库来改变。目前，你能在几种不同的语言间选取错误消息。见 9.1 MySQL 支持什么语言？。

20.4.12.2 返回值

一个描述错误的字符串。如果没有错误发生，空字符串。

20.4.12.3 错误

无。

20.4.13 mysql_escape_string()

unsigned int mysql_escape_string(char *to, const char *from, unsigned int length)

20.4.13.1 说明

把在 from 中的字符串编码为在一条 SQL 语句中可以发给服务器的转义的 SQL 字符串，将结果放在 to 中， 并且加上一个终止的空字节。编码的字符是 NUL（ASCII 0)、‘\n’、‘\r’、‘\’、‘’’、‘"’和 Control-Z(见 7.1 文字：如何写字符串和数字)。

由 from 指向的字符串必须是 length 个字节长。你必须分配 to 的缓冲区至少 length*2+1 个字节长。（在更坏的情况，每个字符可能需要使用 2 个字节被编码，并且你需要为终止空字节的空间) 当 mysql_escape_string()返回时，to 的内容将是空字符终止的字符串。返回值是编码后的字符串的长度，不包括终止空字符。

20.4.13.2 范例

```
char query[1000],*end;
end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_escape_string(end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_escape_string(end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';
if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
fprintf(stderr, "Failed to insert row, Error: %s\n",
                mysql_error(&mysql));
}
```

例子中所用的 strmov()函数被包括在 mysqlclient 库中且功能类似于 strcpy()，但是返回一个指向空终止的第一个参数的指针。

20.4.13.3 返回值

放进 to 的值的长度，不包括终止空字符。

20.4.13.4 错误

无。

20.4.14 mysql_fetch_field()

MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)

20.4.14.1 说明

返回作为一个 MYSQL_FIELD 结构的一个结果集合的一个列的定义。重复调用这个函数在结果集合中检索所有关于列的信息。当没有剩下更多的字段时，mysql_fetch_field()返回 NULL。在每次你执行一个新的 SELECT 查询，mysql_fetch_field()被重置（reset)以返回有关第一列的信息。由 mysql_fetch_field()返回的字段也受调用 mysql_field_seek()的影响。

如果你调用 mysql_query()在一张表上执行一个 SELECT，但是没调用 mysql_store_result()，如果你调用 mysql_fetch_field()询问一个 BLOB 字段的长度，MySQL 返回缺省 BLOB 长度(8K 字节)。（选择 8K 的长度是因为 MySQL 不知道 BLOB 的最大长度。这应该在某个时候是它可配置) 一旦你已经检索了结果集合，field->max_length 包含了在特定查询中对于该列最大值的长度。

20.4.14.2 返回值

当前列的 MYSQL_FIELD 结构。如果没有列剩下，NULL。

20.4.14.3 错误

无。

20.4.14.4 范例

```
MYSQL_FIELD *field;
while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

20.4.15 mysql_fetch_fields()

MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)

20.4.15.1 说明

返回一个结果集合的所有 MYSQL_FIELD 结构的数组。每个结构提供结果结合中一列的字段定义。

20.4.15.2 返回值

一个结果集合的所有 MYSQL_FIELD 结构的一个数组。

20.4.15.3 错误

无。

20.4.15.4 范例

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;
num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
printf("Field %u is %s\n", i, fields[i].name);
}
```

20.4.16 mysql_fetch_field_direct()

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)

20.4.16.1 说明

给定在一个结果集合中的一个列的字段编号 fieldnr，返回作为 MYSQL_FIELD 结构的列的字段定义。你可以使用这个函数检索任意列的义。fieldnr 的值应该在从 0 到 mysql_num_fields(result)-1 范围内。

20.4.16.2 返回值

指定列的 MYSQL_FIELD 结构。

20.4.16.3 错误

无。

20.4.16.4 范例

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;
num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

20.4.17 mysql_fetch_lengths()

unsigned long *mysql_fetch_lengths(MYSQL_RES *result)

20.4.17.1 说明

返回在结果集合内的当前行的列长度。如果你计划拷贝字段值，这个长度信息对优化也是有用的，因为你可以避免调用 strlen()。另外，如果结果集合中包 含二进制数据，你必须使用这个函数确定数据的大小，因为 strlen()对包含空字符的任何字段返回不正确的结果。
空列和包含 NULL 的列的长度值是零。为了看清如何区分这两种情况，见 mysql_fetch_row()的说明。

20.4.17.2 返回值

表示每列大小的无符号长整数的一个数组(不包括任何终止空字符)。如果出现一个错误，NULL。

20.4.17.3 错误

mysql_fetch_lengths()只对结果集合的当前行有效。如果你在调用 mysql_fetch_row()之前或在检索出在结果中的所有以后，它返回 NULL。

20.4.17.4 范例

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;
row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
}
```

20.4.18 mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

20.4.18.1 说明

检索一个结果集合的下一行。当在 mysql_store_result()之后使用时，如果没有更多的行可见所时，mysql_fetch_row()返 回 NULL。当在 mysql_use_result()之后使用时，当没有更多的行可检索时或如果出现一个错误，mysql_fetch_row()返回 NULL。
在行中值的数量由 mysql_num_fields(result)给出。如果 row 保存了从一个对用 mysql_fetch_row() 调用返回的值，指向该值的指针作为 row[0] 到 row[mysql_num_fields(result)-1]来 存取。在行中的 NULL 值由 NULL 指针指出。
在行中字段值的长度可以通过调用 mysql_fetch_lengths()获得。空字段和包含 NULL 的字段长度都是 0；你可以通过检查该值的指针区分他们。如果指针是 NULL，字段是 NULL；否则字段是空的。

20.4.18.2 返回值

下一行的一个 MYSQL_ROW 结构。如果没有更多的行可检索或如果出现一个错误，NULL。

20.4.18.3 错误

CR_SERVER_LOST
对服务器的连接在查询期间失去。

CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.18.4 范例
```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;
num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
unsigned long *lengths;
lengths = mysql_fetch_lengths(result);
for(i = 0; i < num_fields; i++)
{
        printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
}
printf("\n");
}
```
20.4.19 mysql_field_count()
unsigned int mysql_field_count(MYSQL *mysql)
如果你正在使用一个比 3.22.24 早 MySQL 版本，你应该使用 unsigned int mysql_num_fields(MYSQL *mysql)。
20.4.19.1 说明
返回在连接上的最近查询的列的数量。
这个函数一般用在 mysql_store_result()返回 NULL 时（这样你没有结果设置指针)。在这种情况 中，你能调用 mysql_field_count()确定 mysql_store_result()是否应该产生了一个非空的结果。这允许一个客户程序执 行正确的操作，而不必知道查询是否是一条 SELECT（或类 SELECT)语句。下面显示的例子说明这怎样可以做到。
见 20.4.51 为什么在 mysql_query()返回成功后，mysql_store_result()有时返回 NULL?。
20.4.19.2 返回值
在结果集合中表示字段数量字的一个无符号整数。
20.4.19.3 错误
无。
20.4.19.4 范例
```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;
if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
```

```
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

另一个选择是用 mysql_errno(&mysql)代替 mysql_field_count(&mysql)调用。在这种情况中，你直接检查来自 mysql_store_result()的一个错误而 非从 mysql_field_count()值来推断语句是否是一个 SELECT。

20.4.20 mysql_field_seek()

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)

20.4.20.1 说明

将字段光标设置到给定的偏移量。下一次调用 mysql_fetch_field()将检索与该偏移量关联的列的字段定义。

为了定位于行的起始，传递一个值为 0 的 offset 值。

20.4.20.2 返回值

字段光标的先前的值。

20.4.20.3 错误

无。

20.4.21mysql_field_tell()

MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)

20.4.21.1 说明

返回用于最后一个 mysql_fetch_field()的字段光标的位置。这个值可用作 mysql_field_seek()的一个参数。

20.4.21.2 返回值

字段光标的当前偏移量。

20.4.21.3 错误

无。

20.4.22 mysql_free_result()

void mysql_free_result(MYSQL_RES *result)

20.4.22.1 说明

释放由 mysql_store_result()、mysql_use_result()、mysql_list_dbs()等为一个结果集合分配的内存。当你用完了一个结果集合时，你必须调用 mysql_free_result()来释放它使用的内存。

20.4.22.2 返回值

无。

20.4.22.3 错误

无。

20.4.23 mysql_get_client_info()

char *mysql_get_client_info(void)

20.4.23.1 说明

返回代表客户库的版本的字符串。

20.4.23.2 返回值

代表 MySQL 客户库版本的一个字符串。

20.4.23.3 错误

无。

20.4.24 mysql_get_host_info()

char *mysql_get_host_info(MYSQL *mysql)

20.4.24.1 说明

返回描述正在使用的连接类型的字符串，包括服务其主机名。

20.4.24.2 返回值

表示服务器主机名者和连接类型的字符串。

20.4.24.3 错误

无。

20.4.25 mysql_get_proto_info()

unsigned int mysql_get_proto_info(MYSQL *mysql)

20.4.25.1 说明

返回当前连接使用的协议版本。

20.4.25.2 返回值

表示被当前连接使用的协议版本的一个无符号整数。

20.4.25.3 错误

无。

20.4.26 mysql_get_server_info()

char *mysql_get_server_info(MYSQL *mysql)

20.4.26.1 说明

返回表示服务器版本号的字符串。

20.4.26.2 返回值

表示服务器版本号的一个字符串。

20.4.26.3 错误

无。

20.4.27 mysql_info()

char *mysql_info(MYSQL *mysql)

20.4.27.1 说明

检索一个字符串，它提供有关最近执行的查询的信息，但是对下面列出的语句。对其他语句，mysql_info()返回 NULL。字符串的格式随查询类型而变化，如下所述。数字仅仅是说明性的；字符串将包含对查询适当的值。

INSERT INTO ... SELECT ...

字符串格式： Records: 100 Duplicates: 0 Warnings: 0

INSERT INTO ... valueS (...),(...),(...)...

字符串格式： Records: 3 Duplicates: 0 Warnings: 0

LOAD DATA INFILE ...

字符串格式： Records: 1 Deleted: 0 Skipped: 0 Warnings: 0

ALTER TABLE

字符串绳格式： Records: 3 Duplicates: 0 Warnings: 0

UPDATE

字符串格式： Rows matched: 40 Changed: 40 Warnings: 0

注意，只有多个值在语句中指定，mysql_info()对 INSERT ... valueS 语句才返回非 NULL 值。

20.4.27.2 返回值

表示最近执行的查询的附加信息的一个字符串。如果得不到查询的任何信息，NULL。

20.4.27.3 错误

无。

20.4.28 mysql_init()

MYSQL *mysql_init(MYSQL *mysql)

20.4.28.1 说明

分配或初始化适合 mysql_real_connect()的一个 MYSQL 对象。如果 mysql 是一个 NULL 指针，函数分配、初始化并且返回一个新对 象。否则对象被初始化并且返回对象的地址。如果 mysql_init()分配一个新对象，它将在调用 mysql_close()关闭连接时被释放。

20.4.28.2 返回值

一个被初始化的 MYSQL*句柄。如果没有足够的内存来分配一个新对象，NULL。

20.4.28.3 错误

在内存不够的情况下，返回 NULL。

20.4.29 mysql_insert_id()

my_ulonglong mysql_insert_id(MYSQL *mysql)

20.4.29.1 说明

返回由先前的查询为一个 AUTO_INCREMENT 列生成的 ID。在你执行一个 INSERT 查询向一个包含 AUTO_INCREMENT 字段的表中插入后，使用这个函数。

注意，如果先前的查询不产生一个 AUTO_INCREMENT 值，mysql_insert_id()返回 0。如果你需要在以后保存该值，必须在查询生成了该值后马上调用 mysql_insert_id()。

也要注意，SQL 的 LAST_INSERT_ID()函数总是包含最近生成的 AUTO_INCREMENT 值，并且在查询之间不被重置，因为该函数的值在服务器端维护。

20.4.29.2 返回值

有先前的查询更新的 AUTO_INCREMENT 字段的值。如果在连接上没有先前的询问或如果查询没更新 AUTO_INCREMENT 值，返回零。

20.4.29.3 错误

无。

20.4.30 mysql_kill()

int mysql_kill(MYSQL *mysql, unsigned long pid)

20.4.30.1 说明

要求服务器杀死由 pid 指定的线程。

20.4.30.2 返回值

成功，零。如果出现一个错误，非零。

20.4.30.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务器的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.31 mysql_list_dbs()

MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)

20.4.31.1 说明

返回一个结果集合，它用在服务器上的匹配 wild 参数指定的简单正则表达式的数据库名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有 的数据库的一个 NULL 指针。

调用 mysql_list_dbs()类似于执行查询 SHOW databases [LIKE wild]。

你必须用 mysql_free_result()释放结果集合。

20.4.31.2 返回值

成功，一个 MYSQL_RES 结果集合。如果出现一个错误，NULL。

20.4.31.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_OUT_OF_MEMORY

内存溢出。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务器的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.32 mysql_list_fields()

MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)

20.4.32.1 说明
返回一个结果集合，它用在给定表中的匹配 wild 参数指定的简单正则表达式的列名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有列的 一个 NULL 指针。调用 mysql_list_fields()类似于执行查询 SHOW COLUMNS FROM tbl_name [LIKE wild]。
注意，建议你使用 SHOW COLUMNS FROM tbl_name 而不是 mysql_list_fields()。
你必须用 mysql_free_result()释放结果集合。
20.4.32.2 返回值
成功，一个 MYSQL_RES 的结果集合。如果出线一个错误，NULL。
20.4.32.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务者关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.33 mysql_list_processes()
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
20.4.33.1 说明
返回一个描述当前服务器线程的结果集合。这是与 mysqladmin processlist 或 SHOW PROCESSLIST 查询报告的相同信息。
你必须用 mysql_free_result()释放结果集合。
20.4.33.2 返回值
成功，一个 MYSQL_RES 结果集合。如果发生一个错误，NULL。
20.4.33.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务者关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.34 mysql_list_tables()
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
20.4.34.1 说明
返回一个结果集合，它用在当前数据库中的匹配 wild 参数指定的简单正则表达式的表名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有 表的一个 NULL 指针。调用 mysql_list_tables()类似于执行询问 SHOW tables [LIKE wild]。
你必须用 mysql_free_result()释放结果集合。
20.4.34.2 返回值
成功，一个 MYSQL_RES 结果集合。如果出现一个错误，NULL。
20.4.34.3 错误

CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.35 mysql_num_fields()
unsigned int mysql_num_fields(MYSQL_RES *result)
或
unsigned int mysql_num_fields(MYSQL *mysql)
第二中形式在 MySQL 3.22.24 或更新版本上不能工作。为了传递一个 MYSQL* 参数，你必须使用 unsigned int mysql_field_count(MYSQL *mysql)。
20.4.35.1 说明
在结果集合中返回列的数量。
注意，你也可以通过一个指向一个结果集合或一个连接句柄的指针获得列的数量。如果 mysql_store_result()或 mysql_user_result()返回 NULL，你将使用连接句柄（而这样你没有结果集合指针)。在这 种情况下，你可以调用 mysql_field_count()确定 mysql_store_result()是否应该产生非空的结果。这允许客户程序采取 成正确的行动，不必知道查询是否是一个 SELECT（或类 SELECT)语句。下面被显示出的例子说明这怎么可以被做。
见 20.4.51 为什么在 mysql_query()返回成功后，mysql_store_result()有时返回 NULL?。
20.4.35.2 返回值
表示一个结果集合中字段数量的一个无符号整数。
20.4.35.3 错误
无。
20.4.35.4 范例

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;
if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
```

```
        if (mysql_errno(&mysql))
{
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

另一个选择(如果你知道你查询应该返回了一个结果结合)是用 mysql_field_count(&mysql) = 0 的一个检查来代替 mysql_errno(&mysql)。这只会发生在出错了的情形。

20.4.36 mysql_num_rows()

my_ulonglong mysql_num_rows(MYSQL_RES *result)

20.4.36.1 说明

在结果集合中返回行的数量。

mysql_num_rows()的使用取决于你是否使用 mysql_store_result()或 mysql_use_result()返回一个结果集合。如果你使用 mysql_store_result()，mysql_num_rows()可以马 上被调用。如果你使用 mysql_use_result()，mysql_num_rows()将不会返回正确的值，直到在结果集合中的所有行均被检索 了。

20.4.36.2 返回值

在结果集合中行的数量。

20.4.36.3 错误

无。

20.4.37 mysql_options()

int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)

20.4.37.1 说明

能用于设置额外连接选项并且影响一个连接的行为。这个函数可以被多次调用来设置多个选项。

mysql_options()应该在 mysql_init()之后和 mysql_connect()或 mysql_real_connect()之前调用。option 参数是你想要设置的选项；arg 参数是选项的值。如果选项是一个整数，那么 arg 应该指向整数值。

可能的选项值：

选项 参数类型 功能

MYSQL_OPT_CONNECT_TIMEOUT unsigned int * 以秒计的连接超时。

MYSQL_OPT_COMPRESS 不使用 使用压缩的客户机/服务器协议。

MYSQL_OPT_NAMED_PIPE 不使用 使用命名管道连接一个在 NT 上的 MySQL 服务器。

MYSQL_INIT_COMMAND char * 当连接 MySQL 服务器时执行的命令。当重新连接时，将自动重新执行。

MYSQL_READ_DEFAULT_FILE char * 从命名的选项文件而不是从“my.cnf”读取选项。

MYSQL_READ_DEFAULT_GROUP char * 从“my.cnf”或用 MYSQL_READ_DEFAULT_FILE 指定的文件中的命名组中读取选项。
注意，如果你使用 MYSQL_READ_DEFAULT_FILE 或 MYSQL_READ_DEFAULT_GROUP，总是读取 client。
在选项文件中指定的组可能包含下列选项：
compress 使用压缩的客户机/服务器协议。
database 如果在连接命令中没有指定数据库，使用这个数据库。
debug 调试选项
host 缺省主机名
init-command 在连接 MySQL 服务器时，执行的命令。当重新连接时，将自动重新执行。
password 缺省口令
pipe 使用命名管道连接一个在 NT 上的 MySQL 服务器。
port 缺省端口号
return-found-rows 告诉 mysql_info()返回找到的行，而不是在使用 UPDATE 时，返回更新的行。
socket 缺省套接字号
timeout 以秒计的连接超时。
user 缺省用户
nt mysql_kill(MYSQL *mysql, unsigned long pid)
20.4.30.1 说明
要求服务器杀死由 pid 指定的线程。
20.4.30.2 返回值
成功，零。如果出现一个错误，非零。
20.4.30.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.31 mysql_list_dbs()
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
20.4.31.1 说明
返回一个结果集合，它用在服务器上的匹配 wild 参数指定的简单正则表达式的数据库名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有的数据库的一个 NULL 指针。
调用 mysql_list_dbs()类似于执行查询 SHOW databases [LIKE wild]。
你必须用 mysql_free_result()释放结果集合。
20.4.31.2 返回值
成功，一个 MYSQL_RES 结果集合。如果出现一个错误，NULL。
20.4.31.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_OUT_OF_MEMORY

内存溢出。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.32 mysql_list_fields()
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
20.4.32.1 说明
返回一个结果集合，它用在给定表中的匹配 wild 参数指定的简单正则表达式的列名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有列的一个 NULL 指针。调用 mysql_list_fields()类似于执行查询 SHOW COLUMNS FROM tbl_name [LIKE wild]。
注意，建议你使用 SHOW COLUMNS FROM tbl_name 而不是 mysql_list_fields()。
你必须用 mysql_free_result()释放结果集合。
20.4.32.2 返回值
成功，一个 MYSQL_RES 的结果集合。如果出线一个错误，NULL。
20.4.32.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务者关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.33 mysql_list_processes()
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
20.4.33.1 说明
返回一个描述当前服务器线程的结果集合。这是与 mysqladmin processlist 或 SHOW PROCESSLIST 查询报告的相同信息。
你必须用 mysql_free_result()释放结果集合。
20.4.33.2 返回值
成功，一个 MYSQL_RES 结果集合。如果发生一个错误，NULL。
20.4.33.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不正确的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务者关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.34 mysql_list_tables()

MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)

20.4.34.1 说明

返回一个结果集合，它用在当前数据库中的匹配 wild 参数指定的简单正则表达式的表名组成。wild 可以包含通配符字符“%”或“_”，或可以是匹配所有表的一个 NULL 指针。调用 mysql_list_tables()类似于执行询问 SHOW tables [LIKE wild]。

你必须用 mysql_free_result()释放结果集合。

20.4.34.2 返回值

成功，一个 MYSQL_RES 结果集合。如果出现一个错误，NULL。

20.4.34.3 错误

CR_COMMANDS_OUT_OF_SYNC

命令以一个不正确的次序被执行。

CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。

CR_SERVER_LOST

对服务器的连接在查询期间失去。

CR_UNKNOWN_ERROR

发生一个未知的错误。

20.4.35 mysql_num_fields()

unsigned int mysql_num_fields(MYSQL_RES *result)

或

unsigned int mysql_num_fields(MYSQL *mysql)

第二中形式在 MySQL 3.22.24 或更新版本上不能工作。为了传递一个 MYSQL* 参数，你必须使用 unsigned int mysql_field_count(MYSQL *mysql)。

20.4.35.1 说明

在结果集合中返回列的数量。

注意，你也可以通过一个指向一个结果集合或一个连接句柄的指针获得列的数量。如果 mysql_store_result()或 mysql_user_result()返回 NULL，你将使用连接句柄（而这样你没有结果集合指针)。在这种情况下，你可以调用 mysql_field_count()确定 mysql_store_result()是否应该产生非空的结果。这允许客户程序采取成正确的行动，不必知道查询是否是一个 SELECT（或类 SELECT)语句。下面被显示出的例子说明这怎么可以被做。

见 20.4.51 为什么在 mysql_query()返回成功后，mysql_store_result()有时返回 NULL?。

20.4.35.2 返回值

表示一个结果集合中字段数量的一个无符号整数。

20.4.35.3 错误

无。

20.4.35.4 范例

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
```

```
else // query succeeded, process any data returned by it
{
      result = mysql_store_result(&mysql);
      if (result)    // there are rows
      {
            num_fields = mysql_num_fields(result);
            // retrieve rows, then call mysql_free_result(result)
      }
      else    // mysql_store_result() returned nothing; should it have?
      {
            if (mysql_errno(&mysql))
 {
                fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
 }
            else if (mysql_field_count(&mysql) == 0)
            {
                  // query does not return data
                  // (it was not a SELECT)
                  num_rows = mysql_affected_rows(&mysql);
            }
      }
}
```

另一个选择(如果你知道你查询应该返回了一个结果结合)是用 mysql_field_count(&mysql) = 0 的一个检查来代替 mysql_errno(&mysql)。这只会发生在出错了的情形。

20.4.36 mysql_num_rows()

my_ulonglong mysql_num_rows(MYSQL_RES *result)

20.4.36.1 说明

在结果集合中返回行的数量。

mysql_num_rows()的使用取决于你是否使用 mysql_store_result()或 mysql_use_result()返回一个结果集合。如果你使用 mysql_store_result()，mysql_num_rows()可以马上被调用。如果你使用 mysql_use_result()，mysql_num_rows()将不会返回正确的值，直到在结果集合中的所有行均被检索了。

20.4.36.2 返回值

在结果集合中行的数量。

20.4.36.3 错误

无。

20.4.37 mysql_options()

int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)

20.4.37.1 说明

能用于设置额外连接选项并且影响一个连接的行为。这个函数可以被多次调用来设置多个选项。

mysql_options()应该在 mysql_init()之后和 mysql_connect()或 mysql_real_connect()之前调用。option 参数是你想要设置的选项；arg 参数是选项的值。如果选项是一个整数，那么 arg 应该指向整数值。

可能的选项值：
选项　　参数类型　　功能
MYSQL_OPT_CONNECT_TIMEOUT unsigned int * 以秒计的连接超时。
MYSQL_OPT_COMPRESS　　不使用　使用压缩的客户机/服务器协议。
MYSQL_OPT_NAMED_PIPE　不使用　使用命名管道连接一个在 NT 上的 MySQL 服务器。
MYSQL_INIT_COMMAND　　char *　　当连接 MySQL 服务器时执行的命令。当重新连接时，将自动重新执行。
MYSQL_READ_DEFAULT_FILE char *　　从命名的选项文件而不是从“my.cnf”读取选项。
MYSQL_READ_DEFAULT_GROUP　char *　　从“my.cnf”或用 MYSQL_READ_DEFAULT_FILE 指定的文件中的命名组中读取选项。
注意，如果你使用 MYSQL_READ_DEFAULT_FILE 或 MYSQL_READ_DEFAULT_GROUP，总是读取 client。
在选项文件中指定的组可能包含下列选项：
compress 使用压缩的客户机/服务器协议。
database 如果在连接命令中没有指定数据库，使用这个数据库。
debug　　调试选项
host 缺省主机名
init-command 在连接 MySQL 服务器时，执行的命令。当重新连接时，将自动重新执行。
password 缺省口令
pipe 使用命名管道连接一个在 NT 上的 MySQL 服务器。
port 缺省端口号
return-found-rows 告诉 mysql_info()返回找到的行，而不是在使用 UPDATE 时，返回更新的行。
socket　　缺省套接字号
timeout　以秒计的连接超时。
user 缺省用户
对于选项文件的更多信息，见 4.15.4 选项文件。
20.4.37.2 返回值
成功，零。如果你使用了未知的选项，非零。
20.4.37.3 范例

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
      fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```

上例请求客户使用压缩的客户机/服务器协议并且从 my.cnf 文件的 odbc 小节读取额外的选项。
20.4.38 mysql_ping()
int mysql_ping(MYSQL *mysql)
20.4.38.1 说明

检查到服务器的连接是否正在工作。如果它关闭了，自动尝试一个再连接。
这个函数可被已经空闲很长时间的客户使用，来检查服务器是否关闭了连接并且如有必要重新连接。
20.4.38.2 返回值
如果服务器活着，零。如果出现一个错误，非零。
20.4.38.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_UNKNOWN_ERROR
发生一个未知的错误。

20.4.39 mysql_query()
int mysql_query(MYSQL *mysql, const char *query)
20.4.39.1 说明
执行指向空终止的字符串 query 的 SQL 查询，查询必须由一个单个的 SQL 语句组成。你不应该在语句后加上一个终止的分号(“;”)或\g。
mysql_query()不能被用于包含二进制数据的查询；相反你应该使用 mysql_real_query()。（二进制数据可能包含“\0”字符，而 mysql_query()将解释为查询字符串的结束。）
20.4.39.2 返回值
如果查询成功，零。如果出现一个错误，非零。
20.4.39.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
到服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。

20.4.40 mysql_real_connect()
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 int client_flag)
20.4.40.1 说明
mysql_real_connect()试图建立到运行 host 的一个 MySQL 数据库引擎的一个连接。mysql_real_connect()在你可以执行任何其他 API 函数之前必须成功地完成，除了 mysql_get_client_info()。
参数指定如下：
- 第一个参数应该是一个现存 MYSQL 结构的地址。在调用 mysql_real_connect()之前，你必须调用 mysql_init()初始化 MYSQL 结构。见下面的例子。

- host 值可以是一个主机名或一个 IP 地址。如果 host 是 NULL 或字符串"localhost"，假定是到本地主机的一个连接。如果 OS 支持套接字(Unix)或命名管道(Win32)，使用他们而不是 TCP/IP 与服务器连接。
- user 参数包含用户的 MySQL 登录 ID。如果 user 是 NULL，假定是当前用户。在 Unix 下，它是当前登录名。在 Windows ODBC 下，必须明确地指定当前用户名字。见 16.4 怎样填写 ODBC 管理程序中各种域。
- passwd 参数为 user 包含口令。如果 passwd 是 NULL，只有在 user 表中对于有一个空白口令字段的用户的条目将被检查一个匹配。这允许数据库主管设置 MySQL 权限，使用户获得不同的口令，取决于他们是否已经指定一个口令。注意：不要试图在调用 mysql_real_connect()前加密口令；口令加密自动被客户 API 处理。
- db 是数据库名。如果 db 不是 NULL，连接将缺省数据库设置为这个值。
- 如果 port 不是 0，值对于 TCP/IP 连接将用作端口号。注意 host 参数决定连接的类型。
- 如果 unix_socket 不是 NULL，字符串指定套接字或应该被使用的命名管道。注意 host 参数决定连接的类型。
- client_flag 值通常是 0，但是在很特殊的情况下可以被设置为下列标志的组合：

标志名字　　意味着的标志

CLIENT_FOUND_ROWS 返回找到的(匹配的)行数，不是受到影响的行数。

CLIENT_NO_SCHEMA　不允许 db_name.tbl_name.col_name 语法。这是为了 ODBC；如果你使用该语法，导致语法分析器产生一个错误，它是为在一些 ODBC 程序捕捉错误是有用的。

CLIENT_COMPRESS 使用压缩协议。

CLIENT_ODBC 客户是一个 ODBC 客户。这使 mysqld 变得对 ODBC 更友好。

20.4.40.2 返回值

如果连接成功，一个 MYSQL*连接句柄。如果连接失败，NULL。对一个成功的连接，返回值与第一个参数值相同，除非你传递 NULL 给该参数。

20.4.40.3 错误

CR_CONN_HOST_ERROR

不能连接 MySQL 服务器。

CR_CONNECTION_ERROR

不能连接本地 MySQL 服务器。

CR_IPSOCK_ERROR

不能创建一个 IP 套接字。

CR_OUT_OF_MEMORY

内存溢出。

CR_SOCKET_CREATE_ERROR

不能创建一个 Unix 套接字。

CR_UNKNOWN_HOST

不能找到主机名的 IP 地址。

CR_VERSION_ERROR

由于试图使用一个不同协议版本的一个客户库与一个服务器连接导致的一个协议失配。如果你使用一个非常老的客户库连接一个没有使用--old-protocol 选项启动的新服务器，这就能发生。

CR_NAMEDPIPEOPEN_ERROR;

不能在 Win32 上创建一个命名管道。

CR_NAMEDPIPEWAIT_ERROR;

不能在 Win32 上等待一个命名管道。
CR_NAMEDPIPESETSTATE_ERROR;
不能在 Win32 上得到一个管道处理器。
20.4.40.4 范例
MYSQL mysql;

```
mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
```

20.4.41 mysql_real_query()
int mysql_real_query(MYSQL *mysql, const char *query, unsigned int length)
20.4.41.1 说明
执行由 query 指向的 SQL 查询，它应该是一个 length 个字节的字符串。查询必须由一个单个的 SQL 语句组成。你不应该在语句后增加一个终止的分号(“;”)或\g。
对于包含二进制数据的查询，你必须使用 mysql_real_query()而不是 mysql_query()，因为二进制代码数据可能包含“\0”字符，而且，mysql_real_query()比 mysql_query()更快，因为它对查询字符串调用 strlen()。
20.4.41.2 返回值
如果查询成功，零。如果发生一个错误，非零。
20.4.41.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.42 mysql_reload()
int mysql_reload(MYSQL *mysql)
20.4.42.1 说明
要求 MySQL 服务器再次装载授权表。连接的用户必须拥有 reload 权限。
不推荐这个函数。最好使用 mysql_query()发出一条 SQL FLUSH PRIVILEGES 语句。
20.4.42.2 返回值
成功，零。如果发生一个错误，非零。
20.4.42.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST

对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.43 mysql_row_seek()
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
20.4.43.1 说明
设置行光标为在结果集合中的任意行。这要求结果集合结构包含查询的全部结果，这样 mysql_row_seek()只能与 mysql_store_result()一起使用，而不与 mysql_use_result()。
偏移量应该是调用 mysql_row_tell()或 mysql_row_seek()返回的值。这个值不是简单地一个行号；如果你想要在结果集合内用行号来寻找行，使用 mysql_data_seek()。
20.4.43.2 返回值
行光标先前的值。该值可以被传递给随后的 mysql_row_seek()调用。
20.4.44 mysql_row_tell()
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
20.4.44.1 说明
返回为了 mysql_fetch_row()的行光标的当前位置。这个值可以作为一个参数用于 mysql_row_seek()。
你应该仅在 mysql_store_result()后使用 mysql_row_tell()，而不是在 mysql_use_result()后。
20.4.44.2 返回值
行光标当前的偏移量。
20.4.45 mysql_select_db()
int mysql_select_db(MYSQL *mysql, const char *db)
20.4.45.1 说明
使得由 db 指定的数据库成为 在由 mysql 指定的连接上的缺省(当前)数据库。在随后的查询中，这个数据库对于不包括一个显式的数据库指定符的表的引用是缺省数据库。
除非连接的用户能被认证允许使用数据库，否则 mysql_select_db()失败。
20.4.45.2 返回值
成功，零。如果发生一个错误，非零。
20.4.45.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.46 mysql_shutdown()
int mysql_shutdown(MYSQL *mysql)
20.4.46.1 说明
让数据库服务器关闭。连接的用户必须有 shutdown 权限。
20.4.46.2 返回值
成功，零。如果出现一个错误，非零
20.4.46.3 错误

CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。

20.4.47 mysql_stat()
char *mysql_stat(MYSQL *mysql)
20.4.47.1 说明
返回包含类似于由 mysqladmin status 命令提供的信息的一个字符串。它包括正常运行的秒数和正在运行线程、问题、再次装载和打开的表的数目。
20.4.47.2 返回值
描述服务器状态的一个字符串。如果出现一个错误，NULL。
20.4.47.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.48 mysql_store_result()
MYSQL_RES *mysql_store_result(MYSQL *mysql)
20.4.48.1 说明
对于成功地检索数据的每个询问(SELECT、SHOW、DESCRIBE、EXPLAIN)，你必须调用 mysql_store_result()或 mysql_use_result()。
mysql_store_result()读取一个到客户的查询的全部结果，分配一个 MYSQL_RES 结构，并且把结果放进这个结构中。
如果没有行返回，返回一个空集合集合。(空结果集合不同于一个 NULL 返回值。)
一旦你调用了 mysql_store_result()，你可以调用 mysql_num_rows()找出结果集合中有多少行。
你能调用 mysql_fetch_row()从结果集合中取出行，或 mysql_row_seek()和 mysql_row_tell()结果集合中获得或设置当前的行位置。
一旦你用完结果集合，你必须调用 mysql_free_result()。
见 20.4.51 为什么 mysql_query()返回成功后，mysql_store_result()有时返回 NULL?。
20.4.48.2 返回值
一个保存结果的 MYSQL_RES 结构。如果出现一个错误，NULL。
20.4.48.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_OUT_OF_MEMORY

内存溢出。
CR_SERVER_GONE_ERROR
MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.49 mysql_thread_id()
unsigned long mysql_thread_id(MYSQL *mysql)
20.4.49.1 说明
返回当前连接的线程 ID。这个值可用作 mysql_kill()的一个参数以杀死线程。
如果失去连接并且你用 mysql_ping()重新连接，线程 ID 将改变。这意味着你不应该为以后使用获得线程 ID 并且存储它，当你需要它时，你应该获得它。
20.4.49.2 返回值
当前连接的线程 ID 。
20.4.50 mysql_use_result()
MYSQL_RES *mysql_use_result(MYSQL *mysql)
20.4.50.1 说明
对于成功地检索数据的每个查询(SELECT、SHOW、DESCRIBE、EXPLAIN)，你必须调用 mysql_store_result()或 mysql_use_result()。
mysql_use_result()初始化一个结果集合的检索，但不真正将结果集合读入客户，就象 mysql_store_result()那样。相反，必须通过调用 mysql_fetch_row()单独检索出每一行，这直接从服务器读出结果而不在一个临时表或本地缓冲区中存储它，它比 mysql_store_result()更快一点并且使用较少的内存。客户将只为当前行和一个可能最大 max_allowed_packet 字节的通信缓冲区分配内存。
在另一方面，如果你在客户端对每一行正在做很多的处理，或如果输出被送到屏幕，用户可以打一个^S（停止滚动)，你不应该使用 mysql_use_result()。这将阻塞服务器并且阻止另外的线程从数据被取出的任何表中更新数据。
当使用 mysql_use_result()时，你必须执行 mysql_fetch_row()直到返回一个 NULL 值，否则未取出的行将作为下一个查询的结果集合一部分被返回。如果你忘记做这个，C API 将给出错误 Commands out of sync; You can't run this command now !
你不能在一个从 mysql_use_result()返回的结果集合上使用 mysql_data_seek()、mysql_row_seek()、mysql_row_tell()、mysql_num_rows()或 mysql_affected_rows()，你也不能发出另外的查询直到 mysql_use_result()完成。（然而，在你取出所有的行以后，mysql_num_rows()将精确地返回取出的行数。）
一旦你用完结果集合，你必须调用 mysql_free_result()。
20.4.50.2 返回值
一个 MYSQL_RES 结果结构。 如果发生一个错误发生，NULL。
20.4.50.3 错误
CR_COMMANDS_OUT_OF_SYNC
命令以一个不适当的次序被执行。
CR_OUT_OF_MEMORY
内存溢出。
CR_SERVER_GONE_ERROR

MySQL 服务器关闭了。
CR_SERVER_LOST
对服务器的连接在查询期间失去。
CR_UNKNOWN_ERROR
发生一个未知的错误。
20.4.51 为什么在 mysql_query()返回成功后，mysql_store_result()有时返回 NULL?
有可能在一个对 mysql_query()成功的调用后，mysql_store_result()返回 NULL。当这发生时，它意味着出现了下列条件之一：
- 有一个 malloc()失败(例如，如果结果集合太大)。
- 数据不能被读取(发生在连接上的一个错误)。
- 查询没有返回数据(例如，它是一个 INSERT、UPDATE 或 DELETE)。

你总是可以通过调用 mysql_field_count()检查语句是否应该产生非空的结果。如果 mysql_field_count()返回零，结果是空的并且最后一个查询是不回值的一条语句(例如，一条 INSERT 或 DELETE)。如果 mysql_field_count()返回非零值，语句应该产生非空的结果。见对 mysql_field_count()描述的一个例子。
你可以调用 mysql_error()或 mysql_errno()测试一个错误。
20.4.52 我能从查询中得到什么结果？
除了由查询返回的结果集合外，你也能得到下列信息：
- 当执行一条 INSERT、UPDATE 或 DELETE 时，mysql_affected_rows()返回受到最后一个查询影响的行数。一个例外是如果使用一条没有 WHERE 子句的 DELETE，表被截断，它更快！在这种情况下，mysql_affected_rows()对于影响的记录数量返回零。
- mysql_num_rows()返回结果集合中的行数。用 mysql_store_result()，一旦 mysql_store_result()返回，就可以调用 mysql_num_rows()。用 mysql_use_result()，只有在你已经用 mysql_fetch_row()取出了所有行后，才能调用 mysql_num_rows()。
- mysql_insert_id()返回由将一行插入一个具有 AUTO_INCREMENT 索引的表中的最后查询生成的 ID。见 20.4.29 mysql_insert_id()。
- 某些查询(LOAD DATA INFILE ...、INSERT INTO ... SELECT ...、UPDATE)返回附加的信息。结果由返 mysql_info()返回。对其返回字符串的格式，见 mysql_info()的描述。如果没有附加的信息，mysql_info()返回一个 NULL 指针。

20.4.53 我怎样能得到最后插入的行的唯一 ID？
如果你往包含一个具有 AUTO_INCREMENT 属性的列的一张表中插入一个记录，你能通过 mysql_insert_id()函数获得最近生成的 ID。
你也可以通过在你传递给 mysql_query()的一个查询字符串中使用 LAST_INSERT_ID()函数检索出 ID。
你可以执行下列代码检查是否使用一个 AUTO_INCREMENT 索引。这也检查查询是否是有一个 AUTO_INCREMENT 索引的一条 INSERT：

```
if (mysql_error(&mysql)[0] == 0 &&
    mysql_num_fields(result) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

最近产生的 ID 是在一个按连接的基础上在服务器上进行维护，它将不被其他客户改变。如果你更新另外一个有非奇特(non-magic)值(即一个既不是 NULL 也不是 0 的值)的 AUTO_INCREMENT 列，它甚至将不被改变。
如果你想要使用为一张表生成的 ID 并且把它插入到第 2 张表，你可以使用象这样的 SQL 语句：

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');                    # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');    # use ID in second table
```

20.4.54 链接 C API 的问题
当与 C API 链接时，下列错误可能发生一些系统上：

```
gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocket -lnsl

Undefined               first referenced
 symbol                 in file
floor                   /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

如果它发生在你的系统上，你必须通过在编译/链接命令行的最后增加-lm 以包括数学库。
20.4.55 怎样制作一个线程安全的客户
客户“几乎”是线程安全的。最大的问题是在从套接字读取的“net.c”中的子程序不是中断安全的(interruot-safe)。这样做是这样考虑的，即你可能想有你自己的报警来中断一个长时间的读取服务器。
标准客户库没有用线程选项来编译。
为了获得一个线程安全的客户，使用-lmysys, -lstring 和-ldbug 库和服务器使用的 net_serv.o。当使用一个线程化的客户时，你可以充分利用在“thr_alarm.c”文件中的函数。如果你正在使用来自 mysys 库的函数，你唯一必须记住的是首先调用 my_init()！
所有函数除了 mysql_real_connect()目前是线程安全的。下列注意事项描述怎样编译一个线程安全的客户库并且以一种线程安全的方式使用它。（下面对 mysql_real_connect()的注意事项实际上也适用于 mysql_connect()，但是因为 mysql_connect()不提倡使用，无论如何你应该使用 mysql_real_connect()。）
为了使 mysql_real_connect()是线程安全的，你必须用这个命令重新编译客户库：

```
shell> CPPFLAGS=-DTHREAD_SAFE_CLIENT ./configure ...
```

当链接标准客户时，你可能得到的某些因为未定义符号的错误，因为 pthread 库没有被缺省地包括。
最终的“libmysqlclient.a”库现在是线程安全的。它的含义是只要 2 个线程不同时查询 mysql_real_connect()返回的同一个连接句柄，客户代码是线程安全的；客户机/服务器协议在一个给定的连接上一次只允许一个请求。如果你想在同一个的连接上使用多个线程，你必须在 mysql_query()和 mysql_store_result()调用组合附近有一个 mutex 锁定。一旦 mysql_store_result()就绪，锁可以被释放并且其他线程可以查询同一个连接。(换句话说，不同的线程能使用不同被 mysql_store_result()创建的 MYSQL_RES 指针，只要他们使用适当的锁定协议) 如果你用 POSIX 线程编程，你能使用 pthread_mutex_lock()和 pthread_mutex_unlock()建立并且释放一个 mutex 锁定。
如果你使用 mysql_use_result()而不是 mysql_store_result()，锁定将需要包围 mysql_use_result()和 mysql_fetch_row()的调用，然而，它确实对不使用 mysql_use_result()线程客户是最好的。