第25章：API和库

目录

本章介绍了MySQL可使用的API，从哪里获得它们，以及如何使用它们。 详细介绍C API，这是因为它是由MySQL团队开发的，而且它也是大多数其他API的基础。 本章还介绍了libmysqld库（嵌入式服务器），以及对应用程序开发人员有用的一些程序。

25.1. libmysqld，嵌入式MySQL服务器库

25.1.1. 嵌入式MySQL服务器库概述

使用嵌入式MySQL服务器库，能够在客户端应用程序中使用具备全部特性的MySQL服务器。 主要优点在于，增加了速度，并使得嵌入式应用程序的管理更简单。

嵌入式服务器库是以MySQL的客户端／服务器版本为基础的，采用C/C++语言编写。 其结果是嵌入式服务器也是用C/C++语言编写的。 在其他语言中，嵌入式服务器不可用。

API与嵌入式MySQL版本和客户端／服务器版本等效。 要想更改旧的线程式应用程序以使用嵌入式库，正常情况下，仅需添加对下述函数的调用即可。

函数	何时调用
mysql_server_init()	应在调用任何其他MySQL函数之前调用，最好是在main()函数中调用。
mysql_server_end()	应在程序退出前调用。
mysql_thread_init()	应在你所创建的、用于访问MySQL的每个线程中调用。
mysql_thread_end()	应在调用pthread_exit()之前调用。

随后，必须将你的代码与libmysqld.a链接起来，而不是libmysqlclient.a。

在libmysqlclient.a中还包含mysql_server_*xxx*()函数，使用这类函数，通过将应用程序链接到恰当的库，即可在嵌入式版本和客户端／服务器版本之间切换。 请参见25.2.12.1节，“mysql_server_init()”。

嵌入式服务器和独立服务器之间的一项差别在于，对于嵌入式服务器，默认情况下，连接鉴定是禁止的。 对于嵌入式服务器，要想使用鉴定功能，可在激活“**configure**”以配置MySQL分发版时使用“--with-embedded-privilege-control”选项。

25.1.2. 使用libmysqld编译程序

要想获得libmysqld库，应使用“--with-embedded-server”选项配置MySQL。 请参见2.8.2节，“典型配置选项”。

将你的程序与libmysqld链接时，还必须包含系统的pthread库以及MySQL服务器使用的一些库。 执行“**mysql_config --libmysqld-libs**”，可获得库的完整列表。

对于线程程序的编译和链接，必须使用正确的标志，即使你未在代码中直接调用任何线程函数也同样。

要想编译C程序以包含必要文件，并将MySQL服务器库嵌入到程序的编译版本中，可使用GNU C编译器（gcc）。 编译器需要知道各种文件的位置，并需了解如何编译程序的指令。 在下面的示例中，介绍了如何从命令行编译程序的方法：

```
gcc mysql_test.c -o mysql_test -lz \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

在gcc命令后紧跟着未编译C程序文件的名称。 接下来，给定的“-o”选项指明，它后面的文件名是编译器将输出文件的名称，即编译后的程序。 在下一行的代码中，通知编译器获取包含文件和库的位置，以及在其上进行编译的系统的其他设置。 由于“**mysql_config**”存在的问题，在此添加了“-lz”选项（压缩）。 “**mysql_config**”部分包含在backticks中，而不是单引号内。

25.1.3. 使用嵌入式MySQL服务器时的限制

嵌入式服务器存在下述限制：

- 不支持ISAM表。 （这样做的主要目的是为了使库更小）。
- 没有自定义函数（UDF）。
- 没有对核心转储的堆栈跟踪。
- 没有内部RAID支持。 （由于大多数当前操作系统均支持大文件，通常情况下不需要它）。
- 不能将其设置为“主”或“从”（无复制）。
- 在内存较低的系统上，可能无法使用很大的结果集。

· 不能使用套接字或TCP/IP从外部进程连接到嵌入式服务器。但是，你可以连接到中间应用程序，随后，该中间应用程序可代表远程客户端或外部进程连接到嵌入式服务器。

通过编辑“mysql_embed.h”包含文件并重新编译MySQL，可更改某些限制。

25.1.4. 与嵌入式服务器一起使用的选项

对于任何能够与**mysqld**服务器端口监督程序一起给定的选项，也可以与嵌入式服务器库一起使用。在数组中，可将服务器选项作为参量指定给用于初始化服务器的mysql_server_init()。也能在诸如my.cnf的选项文件中给定它们。要想为C程序指定选项文件，请使用“--defaults-file”选项作为函数mysql_server_init()的第2个参量的元素之一。关于mysql_server_init()函数的更多信息，请参见25.2.12.1节，“mysql_server_init()”。

使用选项文件，能够简化客户端／服务器应用程序和嵌入了MySQL的应用程序之间的切换。将常用选项置于[server]组。它们可被两种MySQL版本读取。客户端／服务器选项应被放在[mysqld]部分。将嵌入式MySQL服务器库的选项放在[embedded]部分。将与应用程序相关的选项放在标记为[ApplicationName_SERVER]的部分。请参见4.3.2节，“使用选项文件”。

25.1.5. 嵌入式服务器中尚需完成的事项**(TODO)**

· 我们将提供一些选项以省去MySQL的某些部分，从而使库变得更小。

· 仍有很多速度优化工作需要完成。

· 错误将被写入stderr。我们将增加1个选项为它们指定文件名。

· 使用嵌入式版本时，需要更改InnoDB，使之不再冗长。如果你的数据库不含InnoDB表，要想抑制相关消息，可为组[libmysqd_server]下的选项文件增加“--skip-innodb”选项，或在用**mysql_server_init()**初始化服务器时添加该选项。

25.1.6. 嵌入式服务器示例

在Linux或FreeBSD系统上，无需更改就能使用下面这两个示例程序。对于其他操作系统，需要进行小的修改，主要是文件路径。设计这两个示例的目的在于，为你提供足够的细节信息，以便理解问题，它们是实际应用程序的必要组成部份。第1个示例十分直观。第2个示例采用了一些错误检查功能，略为复杂。在第1个示例的后面，给出了用于编译程序的命令行条目。在第2个示例的后面，给出了GNUmake文件，该文件可用于编译。

示例：**1**

test1_libmysqld.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = { "mysql_test", "--defaults-file=my.cnf" };
int num_elements = sizeof(server_options)/ sizeof(char *);

static char *server_groups[] = { "libmysqld_server", "libmysqld_client" };

int main(void)
{
   mysql_server_init(num_elements, server_options, server_groups);
```

```
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);

   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }

   mysql_free_result(results);
   mysql_close(mysql);
   mysql_server_end();

   return 0;
}
```

下面给出了编译上述程序的命令行命令:

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

示例: **2**

要想检验该示例，创建一个与MySQL源目录同级的test2_libmysqld目录。将test2_libmysqld.c源文件和GNUmakefile保存到该目录，并在test2_libmysqld目录下运行GNU make。

test2_libmysqld.c

```
/*
 * A simple example client, using the embedded MySQL server library
*/

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test2_libmysqld_SERVER", "embedded", "server", NULL
};
```

```
int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_server_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_server_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
   */
  mysql_server_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_server_end();

  exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
```

```
{
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;
```

```
    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

GNUmakefile

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-5.1/include
#lib      := $(HOME)/mysql-5.1/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif
```

```
# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

25.1.7. 嵌入式服务器的许可

我们鼓励所有人在GPL或兼容许可的旗帜下通过发布代码来推广免费软件。对于有能力完成该类事项的人员，也可以选择从MySQL AB购买MySQL的商用许可。详情请参见http://www.mysql.com/company/legal/licensing/。

25.2. MySQL C API

C API代码是与MySQL一起提供的。它包含在mysqlclient库中，并允许C程序访问数据库。

MySQL源码分发版的很多客户端是用C语言编写的。如果你正在寻找能演示如何使用C API的示例，可参看这些客户端程序。你可以在MySQL源码分发版的客户端目录下找到它们。

大多数其他客户端API（除了Connector/J和Connector/NET）采用mysqlclient库来与MySQL服务器进行通信。这意味着（例如），你可以利用很多相同环境变量（与其他客户端程序使用的环境变量相同）带来的好处，这是因为它们是从库中引用的。关于这些变量的详细清单，请参见第8章：客户端和实用工具程序。

客户端具有最大的通信缓冲区大小。初始分配的缓冲区大小（16KB）将自动增加到最大（最大为16MB）。由于缓冲区大小将按需增加，简单地增加默认的最大限制，从其本身来说不会增加资源使用。该大小检查主要是检查错误查询和通信信息包。

通信缓冲区必须足够大，足以包含1条SQL语句（用于客户端-服务器通信）以及1行返回的数据（用于服务器-客户端通信）。每个线程的通信缓冲区将动态增加，以处理直至最大限制的任何查询或行。例如，如果BLOB值包含高达16MB的数据，那么通信缓冲区的大小限制至少为16MB（在服务器和客户端）。客户端的默认最大值为16MB，但服务器的默认最大值为1MB。也可以在启动服务器时，通过更改max_allowed_packet参数的值增加它。请参见7.5.2节，“调节服务器参数”。

每次查询后，MySQL服务器会将通信缓冲区的大小降至net_buffer_length字节。对于客户端，不会降低与连接相关缓冲区大小，直至连接关闭为止，此时，客户端内存将被收回。

关于使用线程的编程方法，请参见25.2.15节，“如何生成线程式客户端”。关于在相同程序创建包含“服务器”和“客户端”的独立应用程序的更多信息（不与外部MySQL服务器通信），请参见25.1节，“libmysqld，嵌入式MySQL服务器库”。

25.2.1. C API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。不应尝试拷贝MYSQL结构。不保证这类拷贝结果会有用。

- MYSQL_RES

  该结构代表返回行的查询结果（SELECT, SHOW, DESCRIBE, EXPLAIN）。在本节的剩余部分，将查询返回的信息称为“结果集”。

- MYSQL_ROW

  这是1行数据的“类型安全”表示。它目前是按照计数字节字符串的数组实施的。（如果字段值可能包含二进制数据，不能将其当作由Null终结的字符串对待，这是因为这类值可能会包含Null字节）。行是通过调用mysql_fetch_row()获得的。

- MYSQL_FIELD

  该结构包含关于字段的信息，如字段名、类型和大小。这里详细介绍了其成员。通过重复调用mysql_fetch_field()，可为每个字段获得MYSQL_FIELD结构。字段值不是该结构的组成部份，它们包含在MYSQL_ROW结构中。

- MYSQL_FIELD_OFFSET

  这是MySQL字段列表偏移量的“类型安全”表示（由mysql_field_seek()使用）。偏移量是行内的字段编号，从0开始。

- my_ulonglong

  用于行数以及mysql_affected_rows()、mysql_num_rows()和mysql_insert_id()的类型。该类型提供的范围为0～1.84e19。

  在某些系统上，不能打印类型my_ulonglong的值。要想打印这类值，请将其转换为无符号长整数类型并使用%lu打印格式，例如：

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

下面列出了MYSQL_FIELD结构包含的成员：

· char * name

字段名称，由Null终结的字符串。如果用AS子句为该字段指定了别名，名称的值也是别名。

· char * org_name

段名称，由Null终结的字符串。忽略别名。

· char * table

包含该字段的表的名称，如果该字段不是计算出的字段的话。对于计算出的字段，表值为空的字符串。如果用AS子句为该表指定了别名，表的值也是别名。

· char * org_table

表的名称，由Null终结的字符串。忽略别名。

· char * db

字段源自的数据的名称，由Null终结的字符串。如果该字段是计算出的字段，db为空的字符串。

· char * catalog

catalog名称。该值总是"def"。

· char * def

该字段的默认值，由Null终结的字符串。仅当使用mysql_list_fields()时才设置它。

· unsigned long length

字段的宽度，如表定义中所指定的那样。

· unsigned long max_length

用于结果集的字段的最大宽度（对于实际位于结果集中的行，最长字段值的长度）。如果使用mysql_store_result()或mysql_list_fields()，它将包含字段的最大长度。如果使用mysql_use_result()，该变量的值为0。

· unsigned int name_length

名称的长度。

· unsigned int org_name_length

org_name的长度。

· unsigned int table_length

表的长度。

· unsigned int org_table_length

org_table的长度。

· unsigned int db_length

db的长度。

· unsigned int catalog_length

catalog的长度。

· unsigned int def_length

def的长度。

· unsigned int flags

用于字段的不同“位标志”。标志的值可以有0个或多个下述位集合：

标志值	标志描述
NOT_NULL_FLAG	字段不能为NULL
PRI_KEY_FLAG	字段是主键的组成部分
UNIQUE_KEY_FLAG	字段是唯一键的组成部分
MULTIPLE_KEY_FLAG	字段是非唯一键的组成部分
UNSIGNED_FLAG	字段具有UNSIGNED属性
ZEROFILL_FLAG	字段具有ZEROFILL属性
BINARY_FLAG	字段具有BINARY属性
AUTO_INCREMENT_FLAG	字段具有AUTO_INCREMENT属性
ENUM_FLAG	字段是ENUM（不再重视）
SET_FLAG	字段是 SET（不再重视）
BLOB_FLAG	字段是BLOB或TEXT（不再重视）
TIMESTAMP_FLAG	字段是TIMESTAMP（不再重视）

不再重视BLOB_FLAG、ENUM_FLAG、SET_FLAG和TIMESTAMP_FLAG标志，原因在于，它们指出了字段的类型，而不是类型的属性。更可取的方式是使用MYSQL_TYPE_BLOB、MYSQL_TYPE_ENUM、MYSQL_TYPE_SET或MYSQL_TYPE_TIMESTAMP测试field->type。

在下面的示例中，介绍了标志值的典型用法：

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

可以使用下述方面的宏来定义标志值的布尔状态：

标志状态	描述
IS_NOT_NULL(flags)	如果该字段定义为NOT NULL，为“真”。
IS_PRI_KEY(flags)	如果该字段是主键，为“真”。
IS_BLOB(flags)	如果该字段是BLOB或TEXT，为“真”（不再重视，用测试field->type取而代之）。

· unsigned int decimals

用于数值字段的十进制数数目。

· unsigned int charset_nr

用于字段的字符集编号。

· enum enum_field_types type

字段的类型。类型值可以是下标所列的MYSQL_TYPE_符号之一：

类型值	类型描述
MYSQL_TYPE_TINY	TINYINT字段
MYSQL_TYPE_SHORT	SMALLINT字段
MYSQL_TYPE_LONG	INTEGER字段
MYSQL_TYPE_INT24	MEDIUMINT字段
MYSQL_TYPE_LONGLONG	BIGINT字段
MYSQL_TYPE_DECIMAL	DECIMAL或NUMERIC字段
MYSQL_TYPE_NEWDECIMAL	精度数学DECIMAL或NUMERIC
MYSQL_TYPE_FLOAT	FLOAT字段
MYSQL_TYPE_DOUBLE	DOUBLE或REAL字段
MYSQL_TYPE_BIT	BIT字段
MYSQL_TYPE_TIMESTAMP	TIMESTAMP字段
MYSQL_TYPE_DATE	DATE字段
MYSQL_TYPE_TIME	TIME字段
MYSQL_TYPE_DATETIME	DATETIME字段
MYSQL_TYPE_YEAR	YEAR字段
MYSQL_TYPE_STRING	CHAR字段
MYSQL_TYPE_VAR_STRING	VARCHAR字段
MYSQL_TYPE_BLOB	BLOB或TEXT字段（使用max_length来确定最大长度）
MYSQL_TYPE_SET	SET字段
MYSQL_TYPE_ENUM	ENUM字段
MYSQL_TYPE_GEOMETRY	Spatial字段
MYSQL_TYPE_NULL	NULL-type字段
MYSQL_TYPE_CHAR	不再重视，用MYSQL_TYPE_TINY取代

可以使用IS_NUM()宏来测试字段是否具有数值类型。将类型值传递给IS_NUM()，如果字段为数值类型，会将其评估为“真”：

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

25.2.2. C API函数概述

这里归纳了C API可使用的函数，并在下一节详细介绍了它们。请参见25.2.3节，“C API函数描述”。

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()	MySQL

	连接到　　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。

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。

25.2.3. C API函数描述

在本节所作的介绍中，按照C编程语言，为NULL的参数或返回值表示NULL，而不是MySQL Null值。

返回值的函数通常会返回指针或整数。除非作了其他规定，返回指针的函数将返回非Null值，以指明成功，或返回NULL值以指明出错。返回整数的函数将返回0以指明成功，或返回非0值以指明出错。注意，非0值仅表明这点。除非在函数描述中作了其他说明，不要对非0值进行测试：

```
if (result)                  /* correct */
    ... error ...


if (result < 0)              /* incorrect */
    ... error ...
```

```
if (result == -1)            /* incorrect */
    ... error ...
```

当函数返回错误时，在函数描述的“错误”部分将列出可能的错误类型。通过调用mysql_errno()可发现出现的错误是什么。通过调用mysql_error()，可获得错误的字符串表示。

25.2.3.1. mysql_affected_rows()

my_ulonglong mysql_affected_rows(MYSQL *mysql)

描述

返回上次UPDATE更改的行数，上次DELETE删除的行数，或上次INSERT语句插入的行数。对于UPDATE、DELETE或INSERT语句，可在mysql_query()后立刻调用。对于SELECT语句，mysql_affected_rows()的工作方式与mysql_num_rows()类似。

返回值

大于0的整数表明受影响或检索的行数。“0”表示UPDATE语句未更新记录，在查询中没有与WHERE匹配的行，或未执行查询。“-1”表示查询返回错误，或者，对于SELECT查询，在调用mysql_store_result()之前调用了mysql_affected_rows()。由于mysql_affected_rows()返回无符号值，通过比较返回值和“(my_ulonglong)-1”或等效的“(my_ulonglong)~0”，检查是否为“-1”。

错误

无。

示例:

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

如果在连接至**mysqld**时指定了标志CLIENT_FOUND_ROWS，对于UPDATE语句，mysql_affected_rows()将返回WHERE语句匹配的行数。

注意，使用REPLACE命令时，如果新行替代了旧行，mysql_affected_rows()返回2。这是因为，在该情况下，删除了重复行后插入了1行。

如果使用“INSERT ... ON DUPLICATE KEY UPDATE”来插入行，如果行是作为新行插入的，mysql_affected_rows()返回1，如果是更新了已有的行，返回2。

25.2.3.2. mysql_autocommit()

my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)

描述

如果模式为“1”，启用autocommit模式；如果模式为“0”，禁止autocommit模式。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.3. mysql_change_user()

my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)

描述

更改用户，并使由db指定的数据库成为由mysql指定的连接上的默认数据库（当前数据库）。在后续查询中，对于不包含显式数据库区分符的表引用，该数据库是默认数据库。

如果不能确定已连接的用户或用户不具有使用数据库的权限，mysql_change_user()将失败。在这种情况下，不会改变用户和数据库。

如果不打算拥有默认数据库，可将db参数设置为NULL。

该命令总是会执行活动事务的ROLLBACK操作，关闭所有的临时表，解锁所有的锁定表，并复位状态，就像进行了新连接那样。即使未更改用户，也会出现该情况。

返回值

0表示成功，非0值表示出现错误。

错误

与从mysql_real_connect()获得的相同。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中丢失了与服务器的连接。

· CR_UNKNOWN_ERROR

出现未知错误。

· ER_UNKNOWN_COM_ERROR

MySQL服务器未实施该命令（或许是较低版本的服务器）。

· ER_ACCESS_DENIED_ERROR

用户或密码错误。

· ER_BAD_DB_ERROR

数据库不存在。

· ER_DBACCESS_DENIED_ERROR

用户没有访问数据库的权限。

· ER_WRONG_DB_NAME

数据库名称过长。

示例:

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

25.2.3.4. mysql_character_set_name()

const char *mysql_character_set_name(MYSQL *mysql)

描述

为当前连接返回默认的字符集。

返回值

默认字符集。

错误

无。

25.2.3.5. mysql_close()

void mysql_close(MYSQL *mysql)

描述

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。

返回值

无。

错误

无。

25.2.3.6. mysql_commit()

my_bool mysql_commit(MYSQL *mysql)

描述

提交当前事务。

该函数的动作受completion_type系统变量的值控制。尤其是，如果completion_type的值为2，终结事务并关闭客户端连接后，服务器将执行释放操作。客户端程序应调用mysql_close()，从客户端一侧关闭连接。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.7. mysql_connect()

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

描述

该函数已过时。最好使用mysql_real_connect()取而代之。

mysql_connect()试图建立与运行在主机上的MySQL数据库引擎的连接。在能够执行任何其他API函数之前，mysql_connect()必须成功完成，但mysql_get_client_info()例外。

这些参数的意义与mysql_real_connect()的对应参数的意义相同，差别在于连接参数可以为NULL。在这种情况下，C API将自动为连接结构分配内存，并当调用mysql_close()时释放分配的内存。该方法的缺点是，如果连接失败，你无法检索错误消息。要想从mysql_errno()或mysql_error()获得错误消息，必须提供有效的MYSQL指针。

返回值

与mysql_real_connect()的相同。

错误

与mysql_real_connect()的相同。

25.2.3.8. mysql_create_db()

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

描述

创建由db参数命名的数据库。

该函数已过时。最好使用mysql_query()来发出SQL CREATE DATABASE语句。

返回值

如果数据库已成功创建，返回0，如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

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

25.2.3.9. mysql_data_seek()

void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)

描述

在查询结果集中寻找任意行。偏移值为行号，范围从0到mysql_num_rows(result)-1。

该函数要求结果集结构包含查询的所有结果，因此，so mysql_data_seek()仅应与mysql_store_result()联合使用，而不是与mysql_use_result()。

返回值

无。

错误

无。

25.2.3.10. mysql_debug()

void mysql_debug(const char *debug)

描述

用给定的字符串执行DBUG_PUSH。mysql_debug()采用Fred Fish调试库。要想使用该函数，必须编译客户端库，使之支持调试功能。请参见E.1节，“调试MySQL服务器”。请参见E.2节，“调试MySQL客户端”。

返回值

无。

错误

无。

示例:

这里给出的调用将使客户端库在客户端机器的/tmp/client.trace中生成1个跟踪文件。

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

25.2.3.11. mysql_drop_db()

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

描述

撤销由db参数命名数据库。

该函数已过时。最好使用mysql_query()来发出SQL DROP DATABASE语句

返回值

如果成功撤销了数据库，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

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

25.2.3.12. mysql_dump_debug_info()

int mysql_dump_debug_info(MYSQL *mysql)

描述

指示服务器将一些调试信息写入日志。要想使之工作，已连接的用户必须具有SUPER权限。

返回值

如果命令成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.13. mysql_eof()

my_bool mysql_eof(MYSQL_RES *result)

描述

该函数已过时。应使用mysql_errno()或mysql_error()取而代之。

mysql_eof()确定是否已读取了结果集的最后1行。

如果通过成功调用mysql_store_result()获得了结果集，客户端将在1次操作中收到整个结果集。在该情况下，从mysql_fetch_row()返回的NULL总表示已到达结果集末尾，而且没必要调用mysql_eof()。与mysql_store_result()一起使用时，mysql_eof()总返回“真”。

另一方面，如果你使用mysql_use_result()来初始化结果集检索，当重复调用mysql_fetch_row()时，将逐个地从服务器获取结果集的行。由于在该过程中，可能出现连接上的错误，从mysql_fetch_row()返回的NULL值不一定表示已正常地抵达结果集末尾。在该情况下，可以使用mysql_eof()来判定出现了什么情况。如果抵达结果集末尾，mysql_eof()返回非0值，如果出现错误，返回0。

从历史的角度上看，mysql_eof()在日期上早于标准的MySQL错误函数mysql_errno()和mysql_error()。由于这类错误函数提供了相同的信息，它们优先于已过时的mysql_eof()。事实上，它们提供了更多信息，这是因为，mysql_eof()仅返回布尔值，错误函数能够在出现错误时指明错误的原因。

返回值

如果未出现错误，返回0。如果抵达结果集的末尾，返回非0值。

错误

无。

示例:

在下面的示例中，介绍了使用mysql_eof()的方法:

```
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));
}
```

但是，你也能使用标准的MySQL错误函数实现相同的结果:

```
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));
}
```

25.2.3.14. mysql_errno()

unsigned int mysql_errno(MYSQL *mysql)

描述

对于由mysql指定的连接，mysql_errno()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。在MySQL errmsg.h头文件中，列出了客户端错误消息编号。在附录B：错误代码和消息中，也列出了这些错误。

注意，如果成功，某些函数，如mysql_fetch_row()等，不会设置mysql_errno()。

经验规则是，如果成功，所有向服务器请求信息的函数均会复位mysql_errno()。

返回值

如果失败，返回上次mysql_*xxx*()调用的错误代码。“0”表示未出现错误。

错误

无。

25.2.3.15. mysql_error()

const char *mysql_error(MYSQL *mysql)

描述

对于由mysql指定的连接，对于失败的最近调用的API函数，mysql_error()返回包含错误消息的、由Null终结的字符串。如果该函数未失败，mysql_error()的返回值可能是以前的错误，或指明无错误的空字符串。

经验规则是，如果成功，所有向服务器请求信息的函数均会复位mysql_error()。

对于复位mysql_errno()的函数，下述两个测试是等效的：

```
if(mysql_errno(&mysql))
{
    // an error occurred
}


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

通过重新编译MySQL客户端库，可以更改客户端错误消息的语言。目前，能够选择数种语言显示错误消息，请参见5.10.2节，“设置错误消息语言”。

返回值

返回描述错误的、由Null终结的字符串。如果未出现错误，返回空字符串。

错误

无。

25.2.3.16. mysql_escape_string()

应使用mysql_real_escape_string()取而代之！

该函数与mysql_real_escape_string()等同，但mysql_real_escape_string()会将连接处理程序作为其第1个参量，并按照当前字符集对字符串进行转义处理。mysql_escape_string()不采用连接参量，而且不考虑当前字符集设置。

25.2.3.17. mysql_fetch_field()

MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)

描述

返回采用MYSQL_FIELD结构的结果集的列。重复调用该函数，以检索关于结果集中所有列的信息。未剩余字段时，mysql_fetch_field()返回NULL。

每次执行新的SELECT查询时，将复位mysql_fetch_field()，以返回关于第1个字段的信息。调用mysql_field_seek()也会影响mysql_fetch_field()返回的字段。

如果调用了mysql_query()以在表上执行SELECT，但未调用mysql_store_result()，如果调用了mysql_fetch_field()以请求BLOB字段的长度，MySQL将返回默认的Blob长度（8KB）。之所以选择8KB是因为MySQL不知道BLOB的最大长度。应在日后使其成为可配置的。一旦检索了结果集，field->max_length将包含特定查询中该列的最大值的长度。

返回值

当前列的MYSQL_FIELD结构。如果未剩余任何列，返回NULL。

错误

无。

示例:

```
MYSQL_FIELD *field;

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

25.2.3.18. mysql_fetch_field_direct()

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

描述

给定结果集内某1列的字段编号fieldnr，以MYSQL_FIELD结构形式返回列的字段定义。可以使用该函数检索任意列的定义。Fieldnr的值应在从0到mysql_num_fields(result)-1的范围内。

返回值

对于指定列，返回MYSQL_FIELD结构。

错误

无。

示例:

```
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);
}
```

25.2.3.19. mysql_fetch_fields()

MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)

描述

对于结果集，返回所有MYSQL_FIELD结构的数组。每个结构提供了结果集中1列的字段定义。

返回值

关于结果集所有列的MYSQL_FIELD结构的数组。

错误

无。

示例:

```
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);
}
```

25.2.3.20. mysql_fetch_lengths()

unsigned long *mysql_fetch_lengths(MYSQL_RES *result)

描述

返回结果集内当前行的列的长度。如果打算复制字段值，该长度信息有助于优化，这是因为，你能避免调用strlen()。此外，如果结果集包含二进制数据，必须使用该函数来确定数据的大小，原因在于，对于包含Null字符的任何字段，strlen()将返回错误的结果。

对于空列以及包含NULL值的列，其长度为0。要想了解区分这两类情况的方法，请参见关于mysql_fetch_row()的介绍。

返回值

无符号长整数的数组表示各列的大小（不包括任何终结NULL字符）。如果出现错误，返回NULL。

错误

mysql_fetch_lengths()仅对结果集的当前行有效。如果在调用mysql_fetch_row()之前或检索了结果集中的所有行后调用了它，将返回NULL。

示例:

```
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]);
    }
}
```

25.2.3.21. mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

描述

检索结果集的下一行。在mysql_store_result()之后使用时，如果没有要检索的行，mysql_fetch_row()返回NULL。在mysql_use_result()之后使用时，如果没有要检索的行或出现了错误，mysql_fetch_row()返回NULL。

行内值的数目由mysql_num_fields(result)给出。如果行中保存了调用mysql_fetch_row()返回的值，将按照row[0]到row[mysql_num_fields(result)-1]，访问这些值的指针。行中的NULL值由NULL指针指明。

可以通过调用mysql_fetch_lengths()来获得行中字段值的长度。对于空字段以及包含NULL的字段，长度为0。通过检查字段值的指针，能够区分它们。如果指针为NULL，字段为NULL，否则字段为空。

返回值

下一行的MYSQL_ROW结构。如果没有更多要检索的行或出现了错误，返回NULL。

错误

注意，在对mysql_fetch_row()的两次调用之间，不会复位错误。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

```
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");
}
```

25.2.3.22. mysql_field_count()

unsigned int mysql_field_count(MYSQL *mysql)

描述

返回作用在连接上的最近查询的列数。

该函数的正常使用是在mysql_store_result()返回NULL（因而没有结果集指针）时。在这种情况下，可调用mysql_field_count()来判定mysql_store_result()是否应生成非空结果。这样，客户端就能采取恰当的动作，而无需知道查询是否是SELECT（或类似SELECT的）语句。在这里给出的示例中，演示了完成它的方法。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”.

返回值

表示结果集中列数的无符号整数。

错误

无。

示例:

```
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));
        }
    }
}
```

另一种可选的方法是，用mysql_errno(&mysql)替换mysql_field_count(&mysql)调用。在该情况下，无论语句是否是SELECT，你将直接从mysql_store_result()查找错误，而不是从mysql_field_count()的值进行推断。

25.2.3.23. mysql_field_seek()

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)

描述

将字段光标设置到给定的偏移处。对mysql_fetch_field()的下一次调用将检索与该偏移相关的列定义。

要想查找行的开始，请传递值为0的偏移量。

返回值

字段光标的前一个值。

错误

无。

25.2.3.24. mysql_field_tell()

MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)

描述

返回上一个mysql_fetch_field()所使用的字段光标的定义。该值可用作mysql_field_seek()的参量。

返回值

字段光标的当前偏移量。

错误

无。

25.2.3.25. mysql_free_result()

void mysql_free_result(MYSQL_RES *result)

描述

释放由mysql_store_result()、mysql_use_result()、mysql_list_dbs()等为结果集分配的内存。完成对结果集的操作后，必须调用mysql_free_result()释放结果集使用的内存。

释放完成后，不要尝试访问结果集。

返回值

无。

错误

无。

25.2.3.26. mysql_get_character_set_info()

void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)

描述

该函数提供了关于默认客户端字符集的信息。可以使用mysql_set_character_set()函数更改默认的字符集。

该函数是在MySQL 5.0.10中增加的。

示例:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set name: %s\n", cs.name);
```

```
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

25.2.3.27. mysql_get_client_info()

char *mysql_get_client_info(void)

描述

返回表示客户端库版本的字符串。

返回值

表示MySQL客户端库版本的字符串。

错误

无。

25.2.3.28. mysql_get_client_version()

unsigned long mysql_get_client_version(void)

描述

返回表示客户端库版本的整数。该值的格式是XYYZZ，其中X是主版本号，YY是发布级别，ZZ是发布级别内的版本号。例如，值40102表示客户端库的版本是4.1.2。

返回值

表示MySQL客户端库版本的整数。

错误

无。

25.2.3.29. mysql_get_host_info()

char *mysql_get_host_info(MYSQL *mysql)

描述

返回描述了所使用连接类型的字符串，包括服务器主机名。

返回值

代表服务器主机名和连接类型的字符串。

错误

无。

25.2.3.30. mysql_get_proto_info()

unsigned int mysql_get_proto_info(MYSQL *mysql)

描述

返回当前连接所使用的协议版本。

返回值

代表当前连接所使用协议版本的无符号整数。

错误

无。

25.2.3.31. mysql_get_server_info()

char *mysql_get_server_info(MYSQL *mysql)

描述

返回代表服务器版本号的字符串。

返回值

代表服务器版本号的字符串。

错误

无。

25.2.3.32. mysql_get_server_version()

unsigned long mysql_get_server_version(MYSQL *mysql)

描述

以整数形式返回服务器的版本号。

返回值

表示MySQL服务器版本的数值，格式如下:

```
major_version*10000 + minor_version *100 + sub_version
```

例如，对于5.0.12，返回500012。

在客户端程序中，为了快速确定某些与版本相关的服务器功能是否存在，该函数很有用。

错误

无。

25.2.3.33. mysql_hex_string()

unsigned long mysql_hex_string(char *to, const char *from, unsigned long length)

描述

该函数用于创建可用在SQL语句中的合法SQL字符串。请参见9.1.1节，“字符串”。

该字符串从形式上编码为十六进制格式，每个字符编码为2个十六进制数。结果被置入其中，并添加1个终结Null字节。

“from”所指向的字符串必须是长度字节“long”。必须为“to”分配缓冲区，缓冲区至少为length*2+1字节长。

当mysql_hex_string()返回时，“to”的内容为由Null终结的字符串。返回值是编码字符串的长度，不包括终结用Null字符。

可采用0x*value*或X'*value*'格式将返回值置于SQL语句中。但是，返回值不包括0x或X'...'。调用者必须提供所希望的格式是何种。

示例：

```
char query[1000],*end;


end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What's this",11);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';


if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

示例中所使用的strmov()函数包含在mysqlclient库中，它的工作方式类似于strcpy()，但返回指向第1个参数终结Null的指针。

返回值

置于“to”中的值的长度，不包括终结用Null字符。

错误

无。

25.2.3.34. mysql_info()

char *mysql_info(MYSQL *mysql)

描述

检索字符串，该字符串提供了关于最近执行查询的信息，但仅对这里列出的语句有效。对于其他语句，mysql_info()返回NULL。字符串的格式取决于查询的类型，如本节所述。数值仅是说明性的，字符串包含与查询相适应的值。

· INSERT INTO ... SELECT ...

字符串格式：记录，100；副本，0；警告，0

· INSERT INTO ... VALUES (...),(...),(...)...

字符串格式：记录，3；副本，0；警告，0

· LOAD DATA INFILE ...

字符串格式：记录，1；删除，0；跳过，0；警告，0

· ALTER TABLE

字符串格式：记录，3；副本，0；警告，0

· UPDATE

字符串格式：匹配行，40；更改，40；警告，0

注意，mysql_info()为INSERT ... VALUES返回非NULL值，INSERT ... VALUES仅用于多行形式的语句（也就是说，仅当指定了多个值列表时）。

返回值

字符串，它表示最近所执行查询的额外信息。如果该查询无可用信息，返回NULL。

错误

无。

25.2.3.35. mysql_init()

MYSQL *mysql_init(MYSQL *mysql)

描述

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。

返回值

初始化的MYSQL*句柄。如果无足够内存以分配新的对象，返回NULL。

错误

在内存不足的情况下，返回NULL。

25.2.3.36. mysql_insert_id()

my_ulonglong mysql_insert_id(MYSQL *mysql)

描述

返回由以前的INSERT或UPDATE语句为AUTO_INCREMENT列生成的值。在包含AUTO_INCREMENT字段的表中执行了INSERT语句后，应使用该函数。

更准确地讲，将在下述条件下更新mysql_insert_id()：

· 将值保存到AUTO_INCREMENT列中的INSERT语句。无论值是通过在列中存储特殊值NULL或0自动生成的，还是确切的非特殊值，都成立。

· 在有多行INSERT语句的情况下，mysql_insert_id()返回第1个自动生成的AUTO_INCREMENT值，如果未生成这类值，将返回插入在AUTO_INCREMENT列中的最后1个确切值。

· 通过将LAST_INSERT_ID(*expr*)插入到任意列中以生成AUTO_INCREMENT值的INSERT语句。

· 通过更新任意列至LAST_INSERT_ID(*expr*)以生成AUTO_INCREMENT值的INSERT语句。

· mysql_insert_id()的值不受诸如SELECT等返回结果集的语句的影响。

· 如果前面的语句返回了错误，mysql_insert_id()的值将是不确定的。

注意，如果前面的语句未使用AUTO_INCREMENT，mysql_insert_id()返回0。如果需要保存值，在生成值的语句后，务必立刻调用mysql_insert_id()。

mysql_insert_id()的值仅受在当前客户端连接内发出的语句的影响。不受由其他客户端发出的语句的影响。

请参见12.9.3节，“信息函数”。

此外还应注意，SQL LAST_INSERT_ID()函数的值总包含最近生成的AUTO_INCREMENT值，而且在语句之间不会被复

位，原因在于该函数的值是在服务器中维护的。另一个区别是，如果设置了AUTO_INCREMENT列来指定非特殊值，不会更新LAST_INSERT_ID()。

LAST_INSERT_ID()不同于mysql_insert_id()的原因在于，LAST_INSERT_ID()在脚本中很容易使用，而mysql_insert_id()则试图提供关于在AUTO_INCREMENT列中出现情况的更准确信息。

返回值

在前面的讨论中予以了介绍。

错误

无。

25.2.3.37. mysql_kill()

int mysql_kill(MYSQL *mysql, unsigned long pid)

描述

请求服务器杀死由pid指定的线程。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.38. mysql_library_end()

void mysql_library_end(void)

描述

它是mysql_server_end()函数的同义词。

关于具体的用法，请参见25.2.2节，“C API函数概述”。

25.2.3.39. mysql_library_init()

int mysql_library_init(int argc, char **argv, char **groups)

描述

这是mysql_server_init()函数的同义词。

关于具体的用法，请参见25.2.2节，“C API函数概述”。

25.2.3.40. mysql_list_dbs()

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

描述

返回由服务器上的数据库名称组成的结果集，该服务器与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符“%”或“_”，也可以是NULL指针，以便与所有的数据库匹配。调用mysql_list_dbs()的方法类似于执行查询SHOW database [LIKE wild]。

必须用mysql_free_result()释放结果集。

返回值

成功后返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.41. mysql_list_fields()

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

描述

返回由给定表中的字段名称组成的结果集，给定表与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符“%”或“_”，也可以是NULL指针，以便与所有的字段匹配。调用mysql_list_fields()的方法类似于执行查询SHOW COLUMNS FROM *tbl_name* [LIKE *wild*]。

注意，建议使用SHOW COLUMNS FROM *tbl_name*，而不是mysql_list_fields()。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.42. mysql_list_processes()

MYSQL_RES *mysql_list_processes(MYSQL *mysql)

描述

返回描述当前服务器线程的结果集。该类信息与**mysqladmin processlist**或SHOW PROCESSLIST查询给出的信息相同。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.43. mysql_list_tables()

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

描述

返回由当前数据库内的表名组成的结果集，当前数据库与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符“%”或“_”，也可以是NULL指针，以便与所有的表匹配。调用mysql_list_tables()的方法类似于执行查询HOW tables [LIKE *wild*]。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。 如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.44. mysql_more_results()

my_bool mysql_more_results(MYSQL *mysql)

描述

如果当前执行的查询存在多个结果，返回“真”，而且应用程序必须调用mysql_next_result()来获取结果。

返回值

如果存在多个结果，返回“真”（1），如果不存在多个结果，返回“假”（0）。

在大多数情况下，可调用mysql_next_result()来测试是否存在多个结果，如果存在多个结果，对检索进行初始化操作。

请参见25.2.9节，“多查询执行的C API处理”。请参见25.2.3.45节，“mysql_next_result()”。

错误

无。

25.2.3.45. mysql_next_result()

int mysql_next_result(MYSQL *mysql)

描述

如果存在多个查询结果，mysql_next_result()将读取下一个查询结果，并将状态返回给应用程序。

如果前面的查询返回了结果集，必须为其调用mysql_free_result()。

调用了mysql_next_result()后，连接状态就像你已为下一查询调用了mysql_real_query()或mysql_query()时的一样。这意味着你能调用mysql_store_result()、mysql_warning_count()、mysql_affected_rows()等等。

如果mysql_next_result()返回错误，将不执行任何其他语句，也不会获取任何更多的结果，

请参见25.2.9节，“多查询执行的C API处理”。

返回值

返回值	描述
0	成功并有多个结果。
-1	成功但没有多个结果。
>0	出错

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。例如，没有为前面的结果集调用mysql_use_result()。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.46. mysql_num_fields()

unsigned int mysql_num_fields(MYSQL_RES *result)

要想传递MYSQL*参量取而代之，请使用无符号整数mysql_field_count(MYSQL *mysql)。

描述

返回结果集中的行数。

注意，你可以从指向结果集的指针或指向连接句柄的指针获得行数。如果mysql_store_result()或mysql_use_result()返回NULL，应使用连接句柄（因而没有结果集指针）。在该情况下，可调用mysql_field_count()来判断mysql_store_result()是否生成了非空结果。这样，客户端程序就能采取恰当的行动，而不需要知道查询是否是SELECT语句（或类似SELECT的语句）。在下面的示例中，介绍了执行该操作的方式。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”。

返回值

表示结果集中行数的无符号整数。

错误

无。

示例:

```
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);
        }
    }
}
```

另一种可选方式是（如果你知道你的查询应返回结果集），使用检查“mysql_field_count(&mysql) is = 0”来替换mysql_errno(&mysql)调用。仅当出错时才应使用它。

25.2.3.47. mysql_num_rows()

my_ulonglong mysql_num_rows(MYSQL_RES *result)

描述

返回结果集中的行数。

mysql_num_rows()的使用取决于是否采用了mysql_store_result()或mysql_use_result()来返回结果集。如果使用了mysql_store_result()，可以立刻调用mysql_num_rows()。如果使用了mysql_use_result()，mysql_num_rows()不返回正确的值，直至检索了结果集中的所有行为止。

返回值

结果集中的行数。

错误

无。

25.2.3.48. mysql_options()

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

描述

可用于设置额外的连接选项，并影响连接的行为。可多次调用该函数来设置数个选项。

应在mysql_init()之后、以及mysql_connect()或mysql_real_connect()之前调用mysql_options()。

选项参量指的是你打算设置的选项。Arg参量是选项的值。如果选项是整数，那么arg应指向整数的值。

可能的选项值：

选项	参量类型	功能
MYSQL_INIT_COMMAND	char *	连接到MySQL服务器时将执行的命令。再次连接时将自动地再次执行。
MYSQL_OPT_COMPRESS	未使用	使用压缩客户端／服务器协议
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	以秒为单位的连接超时。
		对于与libmysqld链接的应用程序，允许库“猜测”是否使用嵌入式服务器或远程服务器。“猜测”表示，如果设置了主机名但不是本地主机，将使用远程服

MYSQL_OPT_GUESS_CONNECTION	未使用	务器。该行为是默认行为。 可使用MYSQL_OPT_USE_EMBEDDED_CONNECTION 和MYSQL_OPT_USE_REMOTE_CONNECTION覆盖它。对于与libmysqlclient链接的应用程序，该选项将被忽略。
MYSQL_OPT_LOCAL_INFILE	指向单元的可选指针	如果未给定指针，或指针指向“unsigned int != 0”，将允许命令LOAD LOCAL INFILE。
MYSQL_OPT_NAMED_PIPE	未使用	使用命名管道连接到NT平台上的MySQL服务器。
MYSQL_OPT_PROTOCOL	unsigned int *	要使用的协议类型。应是mysql.h中定义的mysql_protocol_type的枚举值之一。
MYSQL_OPT_READ_TIMEOUT	unsigned int *	从服务器读取信息的超时（目前仅在Windows平台的TCP/IP连接上有效）。
MYSQL_OPT_RECONNECT	my_bool *	如果发现连接丢失，启动或禁止与服务器的自动再连接。从MySQL 5.0.3开始，默认情况下禁止再连接，这是5.0.13中的新选项，提供了一种以显式方式设置再连接行为的方法。
MYSQL_OPT_SET_CLIENT_IP	char *	对于与libmysqld链接的应用程序（具备鉴定支持特性的已编译libmysqld），它意味着，出于鉴定目的，用户将被视为从指定的IP地址（指定为字符串）进行连接。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_EMBEDDED_CONNECTION	未使用	对于与libmysqld链接的应用程序，对于连接来说，它将强制使用嵌入式服务器。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_REMOTE_CONNECTION	未使用	对于与libmysqld链接的应用程序，对于连接来说，它将强制使用远程服务器。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_RESULT	未使用	不使用该选项。
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	写入服务器的超时（目前仅在Windows平台的TCP/IP连接上有效）。
MYSQL_READ_DEFAULT_FILE	char *	从命名选项文件而不是从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读取选项。
MYSQL_REPORT_DATA_TRUNCATION	my_bool *	通过MYSQL_BIND.error，对于预处理语句，允许或禁止通报数据截断错误（默认为禁止）。
MYSQL_SECURE_AUTH	my_bool*	是否连接到不支持密码混编功能的服务器，在MySQL 4.1.1和更高版本中，使用了密码混编功能。
MYSQL_SET_CHARSET_DIR	char*	指向包含字符集定义文件的目录的路径名。
MYSQL_SET_CHARSET_NAME	char*	用作默认字符集的字符集的名称。
MYSQL_SHARED_MEMORY_BASE_NAME	char*	命名为与服务器进行通信的共享内存对象。应与你打算连接的mysqld服务器使用的选项“-shared-memory-base-name”相同。

注意，如果使用了MYSQL_READ_DEFAULT_FILE或MYSQL_READ_DEFAULT_GROUP，总会读取客户端组。

选项文件中指定的组可能包含下述选项：

选项	描述
connect-timeout	以秒为单位的连接超时。在Linux平台上，该超时也用作等待服务器首次回应的时间。
compress	使用压缩客户端／服务器协议。
database	如果在连接命令中未指定数据库，连接到该数据库。
debug	调试选项。
disable-local-infile	禁止使用LOAD DATA LOCAL。
host	默认主机名。

init-command	连接到MySQL服务器时将执行的命令。再次连接时将自动地再次执行。
interactive-timeout	等同于将CLIENT_INTERACTIVE指定为mysql_real_connect()。请参见25.2.3.51节，“mysql_real_connect()”。
local-infile[=(0|1)]	如果无参量或参量!= 0，那么将允许使用LOAD DATA LOCAL。
max_allowed_packet	客户端能够从服务器读取的最大信息包。
multi-results	允许多语句执行或存储程序的多个结果集。
multi-statements	允许客户端在1个字符串内发送多条语句。（由“;”隔开）。
password	默认密码。
pipe	使用命名管道连接到NT平台上的MySQL服务器。
protocol={TCP | SOCKET | PIPE | MEMORY}	连接到服务器时将使用的协议。
port	默认端口号。
return-found-rows	通知mysql_info()返回发现的行，而不是使用UPDATE时更新的行。
shared-memory-base-name=*name*	共享内存名称，用于连接到服务器（默认为"MYSQL"）。
socket	默认的套接字文件。
user	默认用户。

注意，“timeout”（超时）已被“connect-timeout”（连接超时）取代，但为了保持向后兼容，在MySQL 5.1.2-alpha中仍支持“timeout”（超时）。

关于选项文件的更多信息，请参见4.3.2节，“使用选项文件”。

返回值

成功时返回0。如果使用了未知选项，返回非0值。

示例:

```
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));
}
```

该代码请求客户端使用压缩客户端／服务器协议，并从my.cnf文件的obdc部分读取额外选项。

25.2.3.49. mysql_ping()

int mysql_ping(MYSQL *mysql)

描述

检查与服务器的连接是否工作。如果连接丢失，将自动尝试再连接。

该函数可被闲置了较长时间的客户端使用，用以检查服务器是否已关闭了连接，并在必要时再次连接。

返回值

如果与服务器的连接有效返回0。如果出现错误，返回非0值。返回的非0值不表示MySQL服务器本身是否已关闭，连接可能因其他原因终端，如网络问题等。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.50. mysql_query()

int mysql_query(MYSQL *mysql, const char *query)

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”。如果允许多语句执行，字符串可包含多条由分号隔开的语句。请参见25.2.9节，“多查询执行的C API处理”。

mysql_query()不能用于包含二进制数据的查询，应使用mysql_real_query()取而代之（二进制数据可能包含字符‘\0’，mysql_query()会将该字符解释为查询字符串结束）。

如果希望了解查询是否应返回结果集，可使用mysql_field_count()进行检查。请参见25.2.3.22节，“mysql_field_count()”。

返回值

如果查询成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.51. 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 long client_flag)

描述

mysql_real_connect()尝试与运行在主机上的MySQL数据库引擎建立连接。在你能够执行需要有效MySQL连接句柄结构的任何其他API函数之前，mysql_real_connect()必须成功完成。

参数的指定方式如下：

· 第1个参数应是已有MYSQL结构的地址。调用mysql_real_connect()之前，必须调用mysql_init()来初始化MYSQL结构。通过mysql_options()调用，可更改多种连接选项。请参见25.2.3.48节，“mysql_options()”。

· “host”的值必须是主机名或IP地址。如果“host”是NULL或字符串"localhost"，连接将被视为与本地主机的连接。如果操作系统支持套接字（Unix）或命名管道（Windows），将使用它们而不是TCP/IP连接到服务器。

· “user”参数包含用户的MySQL登录ID。如果“user”是NULL或空字符串""，用户将被视为当前用户。在UNIX环境下，它是当前的登录名。在Windows ODBC下，必须明确指定当前用户名。请参见26.1.9.2节，“在Windows上配置MyODBC DSN”。

· “passwd”参数包含用户的密码。如果“passwd”是NULL，仅会对该用户的（拥有1个空密码字段的）用户表中的条目进行匹配检查。这样，数据库管理员就能按特定的方式设置MySQL权限系统，根据用户是否拥有指定的密码，用户将获得不同的权限。

注释：调用mysql_real_connect()之前，不要尝试加密密码，密码加密将由客户端API自动处理。

· “db”是数据库名称。如果db为NULL，连接会将默认的数据库设为该值。

· 如果“port”不是0，其值将用作TCP/IP连接的端口号。注意，“host”参数决定了连接的类型。

· 如果unix_socket不是NULL，该字符串描述了应使用的套接字或命名管道。注意，“host”参数决定了连接的类型。

· client_flag的值通常为0，但是，也能将其设置为下述标志的组合，以允许特定功能：

标志名称	标志描述
CLIENT_COMPRESS	使用压缩协议。
CLIENT_FOUND_ROWS	返回发现的行数（匹配的），而不是受影响的行数。
CLIENT_IGNORE_SPACE	允许在函数名后使用空格。使所有的函数名成为保留字。
CLIENT_INTERACTIVE	关闭连接之前，允许interactive_timeout（取代了wait_timeout）秒的不活动时间。客户端的会话wait_timeout变量被设为会话interactive_timeout变量的值。
CLIENT_LOCAL_FILES	允许LOAD DATA LOCAL处理功能。
CLIENT_MULTI_STATEMENTS	通知服务器，客户端可能在单个字符串内发送多条语句（由‘;’隔开）。如果未设置该标志，将禁止多语句执行。
CLIENT_MULTI_RESULTS	通知服务器，客户端能够处理来自多语句执行或存储程序的多个结果集。如果设置了CLIENT_MULTI_STATEMENTS，将自动设置它。
CLIENT_NO_SCHEMA	禁止*db_name.tbl_name.col_name*语法。它用于ODBC。如果使用了该语法，它会使分析程序生成错误，在捕获某些ODBC程序中的缺陷时，它很有用。
CLIENT_ODBC	客户端是ODBC客户端。它将**mysqld**变得更为ODBC友好。
CLIENT_SSL	使用SSL（加密协议）。该选项不应由应用程序设置，它是在客户端库内部设置的。

对于某些参数，能够从选项文件获得取值，而不是取得mysql_real_connect()调用中的确切值。为此，在调用mysql_real_connect()之前，应与MYSQL_READ_DEFAULT_FILE或MYSQL_READ_DEFAULT_GROUP选项一起调用mysql_options()。随后，在mysql_real_connect()调用中，为准备从选项文件读取值的每个参数指定“无值”值：

· 对于host，指定NULL值或空字符串("")。

· 对于user，指定NULL值或空字符串。

· 对于passwd，指定NULL值。（对于密码，mysql_real_connect()调用中的空字符串的值不能被选项文件中的字符串覆盖，这是因为，空字符串明确指明MySQL账户必须有空密码）。

· 对于db，指定NULL值或空字符串

· 对于port，指定“0”值。

· 对于unix_socket，指定NULL值。

对于某一参数，如果在选项文件中未发现值，将使用它的默认值，如本节前面介绍的那样。

返回值

如果连接成功，返回MYSQL*连接句柄。如果连接失败，返回NULL。对于成功的连接，返回值与第1个参数的值相同。

错误

· CR_CONN_HOST_ERROR

无法连接到MySQL服务器。

· CR_CONNECTION_ERROR

无法连接到本地MySQL服务器。

· CR_IPSOCK_ERROR

无法创建IP套接字。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SOCKET_CREATE_ERROR

无法创建Unix套接字。

· CR_UNKNOWN_HOST

无法找到主机名的IP地址。

· CR_VERSION_ERROR

协议不匹配，起因于：试图连接到具有特定客户端库（该客户端库使用了不同的协议版本）的服务器。如果使用很早的客户端库来建立与较新的服务器（未使用“--old-protocol”选项开始的）的连接，就会出现该情况。

· CR_NAMEDPIPEOPEN_ERROR

无法在Windows平台下创建命名管道。

· CR_NAMEDPIPEWAIT_ERROR

在Windows平台下等待命名管道失败。

· CR_NAMEDPIPESETSTATE_ERROR

在Windows平台下获取管道处理程序失败。

· CR_SERVER_LOST

如果connect_timeout > 0，而且在连接服务器时所用时间长于connect_timeout秒，或在执行init-command时服务器消失。

示例:

```
MYSQL mysql;


mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
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));
}
```

通过使用mysql_options()，MySQL库将读取my.cnf文件的[client]和[your_prog_name]部分，以确保程序工作，即使某人以某种非标准的方式设置MySQL也同样。

注意，一旦建立了连接，mysql_real_connect()将设置再连接标志（MYSQL结构的组成部份）的值，在低于5.0.3版的API中，将其设为“1”，在较新的版本中，将其设为“0”。对于该标志，值“1”表示，如果因连接丢失而无法执行语句，放弃前，将尝试再次连接到服务器。从MySQL 5.0.13开始，可以对mysql_options()使用MYSQL_OPT_RECONNECT选项，对再连接行为进行控制。

25.2.3.52. mysql_real_escape_string()

unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)

注意，mysql必须是有效的开放式连接。之所以需要它是因为，转义功能取决于服务器使用的字符集。

描述

该函数用于创建可在SQL语句中使用的合法SQL字符串。请参见9.1.1节，“字符串”。

按照连接的当前字符集，将“from”中的字符串编码为转义SQL字符串。将结果置于“to”中，并添加1个终结用NULL字节。编码的字符为NUL (ASCII 0)、‘\n’、‘\r’、‘\’、‘’’、‘"’、以及Control-Z（请参见9.1节，“文字值”）。（严格地讲，MySQL仅需要反斜杠和引号字符，用于引用转义查询中的字符串。该函数能引用其他字符，从而使得它们在日志文件中具有更好的可读性）。

“from”指向的字符串必须是长度字节“long”。必须为“to”缓冲区分配至少length*2+1字节。在最坏的情况下，每个字符或许需要使用2个字节进行编码，而且还需要终结Null字节。当mysql_real_escape_string()返回时，“to”的内容是由Null终结的字符串。返回值是编码字符串的长度，不包括终结用Null字符。

如果需要更改连接的字符集，应使用mysql_set_character_set()函数，而不是执行SET NAMES (或SET CHARACTER SET)语句。mysql_set_character_set()的工作方式类似于SET NAMES，但它还能影响mysql_real_escape_string()所使用的字符集，而SET NAMES则不能。

示例:

```
char query[1000],*end;


end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, 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));
}
```

该示例中使用的strmov()函数包含在mysqlclient库中，工作方式与strcpy()类似，但会返回指向第1个参数终结用Null的指针。

返回值

置于“to”中的值的长度，不包括终结用Null字符。

错误

无。

25.2.3.53. mysql_real_query()

int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)

描述

执行由“query”指向的SQL查询，它应是字符串长度字节“long”。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”。如果允许多语句执行，字符串可包含由分号隔开的多条语句。请参见25.2.9节，“多查询执行的C API处理”。

对于包含二进制数据的查询，必须使用mysql_real_query()而不是mysql_query()，这是因为，二进制数据可能会包含‘\0’字符。此外，mysql_real_query()比mysql_query()快，这是因为它不会在查询字符串上调用strlen()。

如果希望知道查询是否应返回结果集，可使用mysql_field_count()进行检查25.2.3.22节，“mysql_field_count()”。

返回值

如果查询成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.54. mysql_refresh()

int mysql_refresh(MYSQL *mysql, unsigned int options)

描述

该函数用于刷新表或高速缓冲，或复位复制服务器信息。连接的用户必须具有RELOAD权限。

“options”参量是一种位掩码，由下述值的任意组合构成。能够以“Or”（或）方式将多个值组合在一起，用一次调用执行多项操作。

· REFRESH_GRANT

刷新授权表，与FLUSH PRIVILEGES类似。

· REFRESH_LOG

刷新日志，与FLUSH LOGS类似。

· REFRESH_TABLES

刷新表高速缓冲，与FLUSH TABLES类似。

· REFRESH_HOSTS

刷新主机高速缓冲，与FLUSH HOSTS类似。

· REFRESH_STATUS

复位状态变量，与FLUSH STATUS类似。

· REFRESH_THREADS

刷新线程高速缓冲。

· REFRESH_SLAVE

在从复制服务器上，复位主服务器信息，并重新启动从服务器，与RESET SLAVE类似。

· REFRESH_MASTER

在主复制服务器上，删除二进制日志索引中列出的二进制日志文件，并截短索引文件，与RESET MASTER类似。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.55. mysql_reload()

int mysql_reload(MYSQL *mysql)

描述

请求MySQL服务器重新加载授权表。连接的用户必须具有RELOAD权限。

该函数已过时。最好使用mysql_query()来发出SQL FLUSH PRIVILEGES语句。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.56. mysql_rollback()

my_bool mysql_rollback(MYSQL *mysql)

描述

回滚当前事务。

该函数的动作取决于completion_type系统变量的值。尤其是，如果completion_type的值为“2”，终结事务后，服务器将执行释放操作，并关闭客户端连接。客户端程序应调用mysql_close()，从客户端一侧关闭连接。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.57. mysql_row_seek()

MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)

描述

将行光标置于查询结果集中的任意行。“offset”值是行偏移量，它应是从mysql_row_tell()或mysql_row_seek()返回的值。该值不是行编号，如果你打算按编号查找结果集中的行，请使用mysql_data_seek()。

该函数要求在结果集的结构中包含查询的全部结果，因此，mysql_row_seek()仅应与mysql_store_result()一起使用，而不是与mysql_use_result()。

返回值

行光标的前一个值。该值可传递给对mysql_row_seek()的后续调用。

错误

无。

25.2.3.58. mysql_row_tell()

MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)

描述

对于上一个mysql_fetch_row()，返回行光标的当前位置。该值可用作mysql_row_seek()的参量。

仅应在mysql_store_result()之后，而不是mysql_use_result()之后使用mysql_row_tell()。

返回值

行光标的当前偏移量。

错误

无。

25.2.3.59. mysql_select_db()

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

描述

使由db指定的数据库成为由mysql指定的连接上的默认数据库（当前数据库）。在后续查询中，该数据库将是未包含明确数据库区分符的表引用的默认数据库。

除非已连接的用户具有使用数据库的权限，否则mysql_select_db()将失败。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.60. mysql_set_character_set()

int mysql_set_character_set(MYSQL *mysql, char *csname)

描述

该函数用于为当前连接设置默认的字符集。字符串csname指定了1个有效的字符集名称。连接校对成为字符集的默认校对。该函数的工作方式与SET NAMES语句类似，但它还能设置mysql->charset的值，从而影响了由mysql_real_escape_string()设置的字符集。

该函数是在MySQL 5.0.7中增加的。

返回值

0表示成功，非0值表示出现错误。

示例:

```
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));
}


if (!mysql_set_charset_name(&mysql, "utf8"))
{
    printf("New client character set: %s\n", mysql_character_set_name(&mysql));
}
```

25.2.3.61. mysql_set_server_option()

int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)

描述

允许或禁止连接的选项。选项可以取下述值之一:

MYSQL_OPTION_MULTI_STATEMENTS_ON	允许多语句支持。
MYSQL_OPTION_MULTI_STATEMENTS_OFF	禁止多语句支持。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· ER_UNKNOWN_COM_ERROR

服务器不支持mysql_set_server_option()（当服务器版本低于4.1.1时），或服务器不支持试图设置的选项。

25.2.3.62. mysql_shutdown()

int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)

描述

请求数据库服务器关闭。已连接的用户必须具有SHUTDOWN权限。MySQL 5.1服务器仅支持1种关闭类型，shutdown_level必须等效于SHUTDOWN_DEFAULT。设计规划了额外的关闭级别，以便能够选择所需的级别。对于用旧版本libmysqlclient头文件编译并调用mysql_shutdown()的动态链接可执行程序，需要与旧版的libmysqlclient动态库一起使用。

在5.5节，“MySQL服务器关机进程”中，介绍了关机进程。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.63. mysql_sqlstate()

const char *mysql_sqlstate(MYSQL *mysql)

描述

返回由Null终结的字符串，该字符串包含关于上次错误的SQLSTATE错误代码。错误代码包含5个字符。'00000'表示无错误。其值由ANSI SQL和ODBC指定。关于可能取值的列表，请参见附录B：错误代码和消息。

注意，并非所有的MySQL错误均会被映射到SQLSTATE错误代码。值'HY000'（一般错误）用于未映射的错误。

返回值

包含SQLSTATE错误码的、由Null终结的字符串。

另请参见：

请参见25.2.3.14节，“mysql_errno()”。请参见25.2.3.15节，“mysql_error()”。请参见25.2.7.26节，“mysql_stmt_sqlstate()”。

25.2.3.64. mysql_ssl_set()

int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char *ca, const char *capath, const char *cipher)

描述

使用mysql_ssl_set()，可采用SSL建立安全连接。必须在mysql_real_connect()之前调用它。

除非在客户端库中允许了OpenSSL支持，否则mysql_ssl_set()不作任何事。

Mysql是从mysql_init()返回的连接处理程序。其他参数的指定如下：

· key是key文件的路径名。

· cert是证书文件的路径名。

· ca是证书授权文件的路径名。

· capath是指向目录的路径名，该目录中包含以pem格式给出的受信任SSL CA证书。

· cipher是允许密码的列表，用于SSL加密。

对于任何未使用的SSL参数，可为其给定NULL。

返回值

该函数总返回0。如果SSL设置不正确，当你尝试连接时，mysql_real_connect()将返回错误。

25.2.3.65. mysql_stat()

char *mysql_stat(MYSQL *mysql)

描述

返回包含特定信息的字符串，该信息与**mysqladmin status**命令提供的信息类似。包括以秒为单位的正常运行时间，以及运行线程的数目，问题数，再加载次数，以及打开的表数目。

返回值

描述服务器状态的字符集。如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.66. mysql_store_result()

MYSQL_RES *mysql_store_result(MYSQL *mysql)

描述

对于成功检索了数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN、CHECK TABLE等），必须调用mysql_store_result()或mysql_use_result() 。

对于其他查询，不需要调用mysql_store_result()或mysql_use_result()，但是如果在任何情况下均调用了mysql_store_result()，它也不会导致任何伤害或性能降低。通过检查mysql_store_result()是否返回0，可检测查询是否没有结果集（以后会更多）。

如果希望了解查询是否应返回结果集，可使用mysql_field_count()进行检查。请参见25.2.3.22节，“mysql_field_count()”。

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。

如果查询未返回结果集，mysql_store_result()将返回Null指针（例如，如果查询是INSERT语句）。

如果读取结果集失败，mysql_store_result()还会返回Null指针。通过检查mysql_error()是否返回非空字符串，mysql_errno()是否返回非0值，或mysql_field_count()是否返回0，可以检查是否出现了错误。

如果未返回行，将返回空的结果集。（空结果集设置不同于作为返回值的空指针）。

一旦调用了mysql_store_result()并获得了不是Null指针的结果，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()。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”.

返回值

具有多个结果的MYSQL_RES结果集合。如果出现错误，返回NULL。

错误

如果成功，mysql_store_result()将复位mysql_error()和mysql_errno()。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.67. mysql_thread_id()

unsigned long mysql_thread_id(MYSQL *mysql)

描述

返回当前连接的线程ID。该值可用作mysql_kill()的参量以杀死线程。

如果连接丢失，并使用mysql_ping()进行了再连接，线程ID将改变。这意味着你不应获取线程ID并保存它供以后使用。应在需要时获取它。

返回值

当前连接的线程ID。

错误

无。

25.2.3.68. mysql_use_result()

MYSQL_RES *mysql_use_result(MYSQL *mysql)

描述

对于成功检索数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN），必须调用mysql_store_result()或mysql_use_result()。

mysql_use_result()将初始化结果集检索，但并不像mysql_store_result()那样将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。这将直接从服务器读取结果，而不会将其保存在临时表或本地缓冲区内，与mysql_store_result()相比，速度更快而且使用的内存也更少。客户端仅为当前行和通信缓冲区分配内存，分配的内存可增加到max_allowed_packet字节。

另一方面，如果你正在客户端一侧为各行进行大量的处理操作，或者将输出发送到了用户可能会键入“^S”（停止滚动）的屏幕，就不应使用mysql_use_result()。这会绑定服务器，并阻止其他线程更新任何表（数据从这类表获得）。

使用mysql_use_result()时，必须执行mysql_fetch_row()，直至返回NULL值，否则，未获取的行将作为下一个检索的一部分返回。C API给出命令不同步错误，如果忘记了执行该操作，将不能运行该命令。

不应与从mysql_use_result()返回的结果一起使用mysql_data_seek()、mysql_row_seek()、mysql_row_tell()、mysql_num_rows()或mysql_affected_rows()，也不应发出其他查询，直至mysql_use_result()完成为止。（但是，提取了所有行后，mysql_num_rows()将准确返回提取的行数）。

一旦完成了对结果集的操作，必须调用mysql_free_result()。

使用libmysqld嵌入式服务器时，由于在调用mysql_free_result()之前，内存使用将随着每个检索的行增加，内存效益将基

本丧失。

返回值

MYSQL_RES结果结构。如果出现错误，返回NULL。

错误

如果成功，mysql_use_result()将复位mysql_error()和mysql_errno()。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.69. mysql_warning_count()

unsigned int mysql_warning_count(MYSQL *mysql)

错误

返回执行前一个SQL语句期间生成的告警数目。

返回值

告警计数。

错误

无。

25.2.4. C API预处理语句

MySQL客户端／服务器协议提供了预处理语句。该功能采用了由mysql_stmt_init()初始化函数返回的MYSQL_STMT语句处理程序数据结构。对于多次执行的语句，预处理执行是一种有效的方式。首先对语句进行解析，为执行作好准备。接下来，在以后使用初始化函数返回的语句句柄执行一次或多次。

对于多次执行的语句，预处理执行比直接执行快，主要原因在于，仅对查询执行一次解析操作。在直接执行的情况下，每次执行语句时，均将进行查询。此外，由于每次执行预处理语句时仅需发送参数的数据，从而减少了网络通信量。

预处理语句的另一个优点是，它采用了二进制协议，从而使得客户端和服务器之间的数据传输更有效率。

下述语句可用作预处理语句：CREATE TABLE、DELETE、DO、INSERT、REPLACE、SELECT、SET、UPDATE、以及大多数SHOW语句。在MySQL 5.1中，不支持其他语句。

25.2.5. C API预处理语句的数据类型

预处理语句主要使用MYSQL_STMT和MYSQL_BIND数据结构。第3种结构MYSQL_TIME用于传输暂时性数据。

· MYSQL_STMT

该结构表示预处理语句。通过调用mysql_stmt_init()创建语句，返回语句句柄，即指向MYSQL_STMT的指针。该句柄用户所有后续的与语句有关的函数，直至使用mysql_stmt_close()关闭了它为止。

MYSQL_STMT结构没有供应用程序使用的参数。此外，不应尝试复制MYSQL_STMT结构。不保证这类复制物会有用。

多个语句句柄能够与单个连接关联起来。对句柄数目的限制取决于系统资源。

· MYSQL_BIND

该结构用于语句输入（发送给服务器的数据值）和输出（从服务器返回的结果值）。对于输入，它与mysql_stmt_bind_param()一起使用，用于将参数数据值绑定到缓冲区上，以供mysql_stmt_execute()使用。对于输出，它与mysql_stmt_bind_result()一起使用，用于绑定结果缓冲区，以便用于with mysql_stmt_fetch()以获取行。

MYSQL_BIND结构包含下述供应用程序使用的成员。每个成员用于输入和输出，但在某些时候，也能用于不同的目的，具体情况取决于数据传输的方向。

o enum enum_field_types buffer_type

缓冲的类型。在本节后面列出了允许的buffer_type值。对于输入，buffer_type指明了与语句参数捆绑的值类型。对于输出，它指明了你希望从结果缓冲收到的值类型。

o void *buffer

对于输入，这是指向存储语句参数数据值的缓冲的指针。对于输出，它是指向返回结果集列值的缓冲的指针。对于数值列类型，缓冲应指向恰当的C类型变量（如果将该变量与具有UNSIGNED属性的列关联起来，变量unsigned C类型。通过使用is_unsigned成员，指明变量是signed或unsigned类型，详情请参见本节后面的介绍）。对于日期和时间列类型，缓冲应指向MYSQL_TIME结构。对于字符和二进制字符串列类型，缓冲应指向字符缓冲区。

o unsigned long buffer_length

*buffer的实际大小，单位为字节。它指明了可保存在缓冲区内的最大数据。对于字符和二进制C数据，buffer_length值指定了与mysql_stmt_bind_param()一起使用时的*buffer长度，或与mysql_stmt_bind_result()一起使用时能够提取到缓冲区内的最大数据。

o unsigned long *length

指向unsigned long变量的指针，该变量指明了存储在*buffer中数据的实际字节数。“length”用于字符或二进制C数据。对于输入参数数据绑定，“length”指向unsigned long变量，该变量指明了存储在*buffer中参数值的长度，供mysql_stmt_execute()使用。对于输出值绑定，mysql_stmt_fetch()会将返回的列值保存到“length”指向的变量中。

对于数值和临时数据类型，“length”将被忽略，原因在于，数据值的长度是由buffer_type值决定的。

o my_bool *is_null

该成员指向my_bool变量，如果值为NULL，该变量为“真”，如果值为非Null，该变量为“假”。对于输入，将*is_null设置为“真”，指明以语句参数的形式传递NULL值。对于输出，如果从语句返回的结果集列值为NULL，当获取了行后，该值将被设为“真”。

“is_null”是指向布尔类型的指针，而不是布尔标量，以便能以下述方式使用它：

§ 如果数据值总是NULL，使用MYSQL_TYPE_NULL绑定列。

§ 如果数据值总是NOT NULL，设置is_null = (my_bool*) 0。

§ 在所有其他情况下，应将is_null设置为my_bool变量的地址，并在各次执行之间恰当地更改变量的值，以指明数据值是NULL或NOT NULL。

o my_bool is_unsigned

该成员用于整数类型。（对应于MYSQL_TYPE_TINY、MYSQL_TYPE_SHORT、MYSQL_TYPE_LONG、以及MYSQL_TYPE_LONGLONG类型的代码）。对于无符号类型，应将“is_unsigned”设置为“真”，对于带符号类型，应将其设置为“假”。

o my_bool error

对于输出，该成员用于通报数据截短错误。必须通过调用带有MYSQL_REPORT_DATA_TRUNCATION选项的mysql_options()，启用截短通报功能。允许该功能后，mysql_stmt_fetch()返回MYSQL_DATA_TRUNCATED，而且对于出现截短情况的参数，在MYSQL_BIND结构中，错误标志为“真”。截短指明丢失了符号或有效位数，或字符串过长以至于无法容纳在1列中。

要想使用MYSQL_BIND结构，应将其内容置为0以便初始化它，然后对其进行设置，恰当地描述它。例如，要想声明并初始化三个MYSQL_BIND结构的数组，可使用下述代码：

```
MYSQL_BIND    bind[3];
memset(bind, 0, sizeof(bind));
```

· MYSQL_TIME

该结构用于将DATE、TIME、DATETIME和TIMESTAMP数据直接发送到服务器，或从服务器直接接收这类数据。将MYSQL_BIND结构的buffer_type成员设置为临时值之一，并将buffer成员设置为指向MYSQL_TIME结构，即可实现该点。

MYSQL_TIME结构包含下述成员：

o unsigned int year

年份

o unsigned int month

月份

o unsigned int day

天

o unsigned int hour

小时

o unsigned int minute

分钟

o unsigned int second

秒

o my_bool neg

布尔标志，用于指明时间是否为负数。

o unsigned long second_part

秒的分数部分。该成员目前不使用。

仅使用施加在给定临时类型值上的MYSQL_TIME结构的部分：用于DATE、DATETIME和TIMESTAMP的年、月、日部分。用于TIME、DATETIME和TIMESTAMP值的小时、分钟、秒部分。请参见25.2.10节，“日期和时间值的C API处理”。

在下面的表格中，给出了可在MYSQL_BIND结构的buffer_type成员中指定的允许值。在该表中，还给出了与每个buffer_type值最接近的对应SQL类型，对于数值和临时类型，给出了对应的C类型。

buffer_type值	**SQL**类型	**C**类型
MYSQL_TYPE_TINY	TINYINT	char
MYSQL_TYPE_SHORT	SMALLINT	short int
MYSQL_TYPE_LONG	INT	int
MYSQL_TYPE_LONGLONG	BIGINT	long long int
MYSQL_TYPE_FLOAT	FLOAT	float
MYSQL_TYPE_DOUBLE	DOUBLE	double

MYSQL_TYPE_TIME	TIME	MYSQL_TIME
MYSQL_TYPE_DATE	DATE	MYSQL_TIME
MYSQL_TYPE_DATETIME	DATETIME	MYSQL_TIME
MYSQL_TYPE_TIMESTAMP	TIMESTAMP	MYSQL_TIME
MYSQL_TYPE_STRING	CHAR	
MYSQL_TYPE_VAR_STRING	VARCHAR	
MYSQL_TYPE_TINY_BLOB	TINYBLOB/TINYTEXT	
MYSQL_TYPE_BLOB	BLOB/TEXT	
MYSQL_TYPE_MEDIUM_BLOB	MEDIUMBLOB/MEDIUMTEXT	
MYSQL_TYPE_LONG_BLOB	LONGBLOB/LONGTEXT	

隐式类型转换可沿两个方向执行。

25.2.6. C API预处理语句函数概述

在此归纳了预处理语句处理功能可使用的函数，并在后面的章节中详细介绍了它。请参见25.2.7节，“C API预处理语句函数描述”。

函数	描述
mysql_stmt_affected_rows()	返回由预处理语句UPDATE、DELETE或INSERT变更、删除或插入的行数目。
mysql_stmt_attr_get()	获取预处理语句属性的值。
mysql_stmt_attr_set()	设置预处理语句的属性。
mysql_stmt_bind_param()	将应用程序数据缓冲与预处理SQL语句中的参数标记符关联起来。
mysql_stmt_bind_result()	将应用程序数据缓冲与结果集中的列关联起来。
mysql_stmt_close()	释放预处理语句使用的内存。
mysql_stmt_data_seek()	寻找语句结果集中的任意行编号。
mysql_stmt_errno()	返回上次语句执行的错误编号。
mysql_stmt_error()	返回上次语句执行的错误消息。
mysql_stmt_execute()	执行预处理语句。
mysql_stmt_fetch()	从结果集获取数据的下一行，并返回所有绑定列的数据。
mysql_stmt_fetch_column()	获取结果集当前行中某列的数据。
mysql_stmt_field_count()	对于最近的语句，返回结果行的数目。
mysql_stmt_free_result()	释放分配给语句句柄的资源。
mysql_stmt_init()	为MYSQL_STMT结构分配内存并初始化它。
mysql_stmt_insert_id()	对于预处理语句的AUTO_INCREMENT列，返回生成的ID。
mysql_stmt_num_rows()	从语句缓冲结果集返回总行数。
mysql_stmt_param_count()	返回预处理SQL语句中的参数数目。
mysql_stmt_param_metadata()	返回结果集的参数元数据。
mysql_stmt_prepare()	为执行操作准备SQL字符串。
mysql_stmt_reset()	复位服务器中的语句缓冲区。
mysql_stmt_result_metadata()	以结果集形式返回预处理语句元数据。
mysql_stmt_row_seek()	使用从mysql_stmt_row_tell()返回的值，查找语句结果集中的行偏移。
mysql_stmt_row_tell()	返回语句行光标位置。
mysql_stmt_send_long_data()	将程序块中的长数据发送到服务器。
mysql_stmt_sqlstate()	返回关于上次语句执行的SQLSTATE错误代码。
mysql_stmt_store_result()	将完整的结果集检索到客户端。

调用mysql_stmt_init()以创建语句句柄，然后调用mysql_stmt_prepare准备语句，调用mysql_stmt_bind_param()提供参数数据，并调用mysql_stmt_execute()执行语句。通过更改mysql_stmt_bind_param()提供的相应缓冲区中的参数值，可重复执行mysql_stmt_execute()。

如果语句是SELECT或任何其他能生成结果集的语句，mysql_stmt_prepare()也会通过mysql_stmt_result_metadata()以MYSQL_RES结果集的形式返回结果集元数据信息。

你可以使用mysql_stmt_bind_result()提供结果缓冲，以便mysql_stmt_fetch()能自动将数据返回给这些缓冲。这是一种按行获取方式。

此外，你也能使用mysql_stmt_send_long_data()将程序块中的文本或二进制数据发送到服务器。请参见25.2.7.25节，“mysql_stmt_send_long_data()”。

完成语句执行后，必须使用mysql_stmt_close()关闭语句句柄，以便与之相关的所有资源均能被释放。

如果通过调用mysql_stmt_result_metadata()获得了SELECT语句的结果集元数据，也应使用mysql_free_result()释放元数据。

执行步骤

要想准备和执行语句，应用程序必须采取下述步骤：

1. 用msyql_stmt_init()创建预处理语句句柄。要想在服务器上准备预处理语句，可调用mysql_stmt_prepare()，并为其传递包含SQL语句的字符串。

2. 如果语句生成了结果集，调用mysql_stmt_result_metadata()以获得结果集元数据。虽然与包含查询返回列的结果集不同，该元数据本身也采用了结果集的形式。元数据结果集指明了结果中包含多少列，并包含每一列的信息。

3. 使用mysql_stmt_bind_param()设置任何参数的值。必须设置所有参数。否则，语句执行将返回错误，或生成无法预料的结果。

4. 调用mysql_stmt_execute()执行语句。

5. 如果语句生成了结果集，捆绑数据缓冲，通过调用mysql_stmt_bind_result()，检索行值。

6. 通过重复调用mysql_stmt_fetch()，按行将数据提取到缓冲区，直至未发现更多行为止。

7. 通过更改参数值并再次执行语句，重复步骤3到步骤6。

调用mysql_stmt_prepare()时，MySQL客户端／服务器协议将执行下述动作：

· 服务器解析语句，并通过赋值语句ID将OK状态发回客户端。此外，如果它是面向结果集的语句，还将发送总的参数数目，列计数和元数据。在此调用过程中，服务器将检查语句的所有语法和语义。

· 客户端采用该语句ID用于进一步操作，以便服务器能从其语句池中识别语句。

调用mysql_stmt_execute()时，MySQL客户端／服务器协议将执行下述动作：

· 客户端使用语句句柄，并将参数数据发送到服务器。

· 服务器使用由客户端提供的ID来识别语句，用新提供的数据替换参数标记符，并执行语句。如果语句生成了结果集，服务器将数据发回客户端。否则，服务器会将发送OK状态，以及总的变更、删除和插入行数。

调用mysql_stmt_fetch()时，MySQL客户端／服务器协议将执行下述动作：

· 客户端按行从信息包读取数据，并通过执行必要的转换操作将其放入应用程序数据缓冲中。如果应用程序的缓冲类型与服务器返回的字段类型相同，转换十分简明。

如果出现了错误，可分别使用mysql_stmt_errno()、mysql_stmt_error()和mysql_stmt_sqlstate()获取语句错误代码、错误消息和SQLSTATE值。

预处理语句日志功能

对于与mysql_stmt_prepare()和mysql_stmt_execute() C API函数一起执行的预处理语句，服务器会将“准备”和“执行”行写入一般查询日志，以便你能了解语句是在何时准备和执行的。

假定按下述方式准备和执行了语句：

1. 调用mysql_stmt_prepare()以准备语句字符串"SELECT ?"。

2. 调用mysql_stmt_bind_param()将值“3”绑定到预处理语句中的参数。

3. 调用mysql_stmt_execute()，执行预处理语句。

上述调用的结果是，服务器将下述行写入一般查询日志：

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

日志中的每个“准备”和“执行”行均具有[*n*]语句ID标识，这样，你就能跟踪已记录的预处理语句。*N*是正整数。对于客户端，如果同时有多个活动的预处理语句，*n*可能会大于1。替换了“?”参数的数据值后，每个“执行”行将显示一条预处理语句。

版本说明：在MySQL 4.1.10之前，显示的“准备”行无[*n*]标识。在MySQL 4.1.10之前，不显示“执行”行。

25.2.7. C API预处理语句函数描述

为了准备和执行查询，请使用下述部分详细介绍的函数。

注意，与MYSQL_STMT结构一起使用的所有函数均以前缀mysql_stmt_开始。

要想创建MYSQL_STMT句柄，请使用mysql_stmt_init()函数。

25.2.7.1. mysql_stmt_affected_rows()

my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)

描述

返回上次执行语句更改、删除或插入的总行数。对于UPDATE、DELETE或INSERT语句，可在mysql_stmt_execute()之后立刻调用它们。对于SELECT语句，mysql_stmt_affected_rows()的工作方式类似于mysql_num_rows()。

返回值

大于0的整数指明了受影响或检索的行数。对于UPDATE语句，“0”表明未更新任何记录，在查询中没有与WHERE子句匹配的行，或尚未执行任何查询。“-1”表明返回了错误，或对SELECT查询，在调用mysql_stmt_store_result()之前调用了mysql_stmt_affected_rows()。由于mysql_stmt_affected_rows()返回无符号值，可通过比较返回值和“(my_ulonglong)-1”（或等效的“(my_ulonglong)~0”），检查“-1”。

关于返回值的额外信息，请参见25.2.3.1节，“mysql_affected_rows()”。

错误

无。

示例：

关于mysql_stmt_affected_rows()的用法，请参阅25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.2. mysql_stmt_attr_get()

int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, void *arg)

描述

可用于获得语句属性的当前值。

“option”参量是希望获取的选项，“arg”应指向包含选项值的变量。如果“option”是整数，那么“arg”应指向整数的值。

关于选项和选项类型的清单，请参见25.2.7.3节，“mysql_stmt_attr_set()”。

返回值

如果OK，返回0。如果选项未知，返回非0值。

错误

无。

25.2.7.3. mysql_stmt_attr_set()

int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, const void *arg)

描述

可用于影响预处理语句的行为。可多次调用该函数来设置多个选项。

“option”参量是希望设置的选项，“arg”参量是选项的值。如果“option”是整数，那么“arg”应指向整数的值。

可能的选项值：

选项	参量类型	功能
STMT_ATTR_UPDATE_MAX_LENGTH	my_bool *	如果设为1：更新mysql_stmt_store_result()中的元数据MYSQL_FIELD->max_length。
STMT_ATTR_CURSOR_TYPE	unsigned long *	调用mysql_stmt_execute()时，语句将打开的光标类型。*arg可以是CURSOR_TYPE_NO_CURSOR（默认值）或CURSOR_TYPE_READ_ONLY。
STMT_ATTR_PREFETCH_ROWS	unsigned long *	使用光标时，一次从服务器获取的行数。*arg的范围从1到unsigned long的最大值。默认值为1。

如果与CURSOR_TYPE_READ_ONLY一起使用了STMT_ATTR_CURSOR_TYPE选项，当调用了mysql_stmt_execute()时，将为语句打开光标。如果存在由前一个mysql_stmt_execute()调用打开的光标，在打开新的光标前，将关闭该光标。此外，为再执行而准备语句之前，mysql_stmt_reset()还将关闭任何打开的光标。mysql_stmt_free_result()将关闭任何打开的光标。

如果为预处理语句打开了光标，没必要调用mysql_stmt_store_result()，这是因为，该函数会导致在客户端一侧对结果集进行缓冲处理。

在MySQL 5.0.2中增加了STMT_ATTR_CURSOR_TYPE选项。在MySQL 5.0.6中，增加了STMT_ATTR_PREFETCH_ROWS选项。

返回值

如果OK，返回0。如果选项未知，返回非0值。

错误

无。

示例：

在下述示例中，为预处理语句打开了1个光标，并将每次获取的行数设为5：

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                                        (void*) &prefetch_rows);
/* ... check return value ... */
```

25.2.7.4. mysql_stmt_bind_param()

my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)

描述

mysql_stmt_bind_param()用于为SQL语句中的参数标记符绑定数据，以传递给mysql_stmt_prepare()。它使用MYSQL_BIND结构来提供数据。“bind”是MYSQL_BIND结构的某一数组的地址。按照客户端库的预期，对于查询中出现的每个“?”参数标记符，数组中均包含1个元素。

假定你准备了下述语句：

```
INSERT INTO mytbl VALUES(?,?,?)
```

绑定参数时，MYSQL_BIND结构的数组包含3个元素，并能声明如下：

```
MYSQL_BIND bind[3];
```

在25.2.5节，“C API预处理语句的数据类型”中，介绍了应设置的每个MYSQL_BIND元素的成员。

返回值

如果绑定成功，返回0。如果出现错误，返回非0值。

错误

· CR_INVALID_BUFFER_USE

指明“bind”（绑定）是否将提供程序块中的长数据，以及缓冲类型是否为非字符串或二进制类型。

· CR_UNSUPPORTED_PARAM_TYPE

不支持该转换。或许buffer_type值是非法的，或不是所支持的类型之一。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

关于mysql_stmt_bind_param()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”给出的示例。

25.2.7.5. mysql_stmt_bind_result()

my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)

描述

mysql_stmt_bind_result()用于将结果集中的列与数据缓冲和长度缓冲关联（绑定）起来。当调用mysql_stmt_fetch()以获取数据时，MySQL客户端／服务器协议会将绑定列的数据置于指定的缓冲区内。

调用mysql_stmt_fetch()之前，必须将所有列绑定到缓冲。“bind”是MYSQL_BIND结构某一数组的地址。按照客户端库的预期，对于结果集中的每一列，数组应包含相应的元素。如果未将列绑定到MYSQL_BIND结构，mysql_stmt_fetch()将简单地忽略数据获取操作。缓冲区应足够大，足以容纳数据值，这是因为协议不返回成块的数据值。

可以在任何时候绑定或再绑定列，即使已部分检索了结果集后也同样。新的绑定将在下一次调用mysql_stmt_fetch()时起作用。假定某一应用程序绑定了结果集中的列，并调用了mysql_stmt_fetch()。客户端／服务器协议将返回绑定缓冲区中的数据。接下来，假定应用程序将多个列绑定到不同的缓冲。该协议不会将数据置于新绑定的缓冲区，直至下次调用mysql_stmt_fetch()为止。

要想绑定列，应用程序将调用mysql_stmt_bind_result()，并传递类型、地址、以及长度缓冲的地址。在25.2.5节，“C API预处理语句的数据类型”中，介绍了应设置的各MYSQL_BIND元素的成员。

返回值

如果绑定成功，返回0。如果出现错误，返回非0值。

错误

· CR_UNSUPPORTED_PARAM_TYPE

不支持该转换。或许buffer_type值是非法的，或不是所支持的类型之一。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

关于mysql_stmt_bind_result()的用法，请参见25.2.7.11节，“mysql_stmt_fetch()”中给出的示例。

25.2.7.6. mysql_stmt_close()

my_bool mysql_stmt_close(MYSQL_STMT *)

描述

关闭预处理语句。此外，mysql_stmt_close()还会取消由“stmt”指向的语句句柄分配。

如果当前语句已挂起或未读取结果，该函数将取消它们，以便能执行下一个查询，

返回值

如果成功释放了语句，返回0。如果出现错误，返回非0值。

错误

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_UNKNOWN_ERROR

出现未知错误。

示例：

关于mysql_stmt_close()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.7. mysql_stmt_data_seek()

void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)

描述

查找语句结果集中的任意行。偏移量为行编号，应位于从0到mysql_stmt_num_rows(stmt)-1的范围内。

该函数要求语句结果集结构包含上次执行查询的全部结果，这样，mysql_stmt_data_seek()就能与mysql_stmt_store_result()一起使用。

返回值

无。

错误

无。

25.2.7.8. mysql_stmt_errno()

unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_errno()将返回最近调用的语句API函数的错误代码，该函数或成功或失败。“0”返回值表示未出现错误。在MySQL errmsg.h头文件中列出了客户端错误消息编号。在mysqld_error.h中，列出了服务器错误消息。此外，在附录B：错误代码和消息中，也列出了错误消息。

返回值

错误代码值。如果未出现错误，返回0。

错误

无。

25.2.7.9. mysql_stmt_error()

const char *mysql_stmt_error(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_error()返回由Null终结的字符串，该字符串包含最近调用的语句API函数的错误消息，该函数或成功或失败。如果未出现错误，返回空字符串("")。这意味着下述两个测试是等效的：

```
if (mysql_stmt_errno(stmt))
{
  // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

通过重新编译MySQL客户端库，可更改客户端错误消息的语言。目前，能够选择数种语言之一显示错误消息。

返回值

描述了错误的字符串。如果未出现错误，返回空字符串。

错误

无。

25.2.7.10. mysql_stmt_execute()

int mysql_stmt_execute(MYSQL_STMT *stmt)

描述

mysql_stmt_execute()执行与语句句柄相关的预处理查询。在该调用期间，将当前绑定的参数标记符的值发送到服务器，服务器用新提供的数据替换标记符。

如果语句是UPDATE、DELETE或INSERT，通过调用mysql_stmt_affected_rows()，可发现更改、删除或插入的总行数。如果这是诸如SELECT等能生成结果集的语句，调用任何其他能导致查询处理的函数之前，必须调用mysql_stmt_fetch()来获取数据。关于如何获取结果的更多信息，请参见25.2.7.11节，“mysql_stmt_fetch()”。

对于生成结果集的语句，执行语句之前，可通过调用mysql_stmt_attr_set()，请求mysql_stmt_execute()为语句打开光标。如果多次执行某一语句，在打开新的光标前，mysql_stmt_execute()将关闭任何已打开的光标。

返回值

如果执行成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

在下面的示例中，介绍了使用mysql_stmt_init()、mysql_stmt_prepare()、mysql_stmt_param_count()、mysql_stmt_bind_param()、mysql_stmt_execute()、以及mysql_stmt_affected_rows()创建和填充表的方法。假定mysql变量具有有效的连接句柄。

```
#define STRING_SIZE 50


#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                     col2 VARCHAR(40),\
                                                     col3 SMALLINT,\
                                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO test_table(col1,col2,col3) VALUES(?,?,?)"


MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
```

```
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
```

```
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row, then re-execute the statement */
int_data= 1000;
strncpy(str_data, "The most popular Open Source database", STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

```
/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

注释：关于使用预处理语句函数的完整示例，请参见文件tests/mysql_client_test.c。该文件可从MySQL源码分发版获得，或从BitKeeper源码仓库获得。

25.2.7.11. mysql_stmt_fetch()

int mysql_stmt_fetch(MYSQL_STMT *stmt)

描述

mysql_stmt_fetch()返回结果集中的下一行。仅能当结果集存在时调用它，也就是说，调用了能创建结果集的mysql_stmt_execute()之后，或当mysql_stmt_execute()对整个结果集即行缓冲处理后调用了mysql_stmt_store_result()。

使用mysql_stmt_bind_result()绑定的缓冲，mysql_stmt_fetch()返回行数据。对于当前列集合中的所有列，它将返回缓冲内的数据，并将长度返回到长度指针。

调用mysql_stmt_fetch()之前，应用程序必须绑定所有列。

如果获取的数据值是NULL值，对应MYSQL_BIND结构的*is_null值将包含TRUE (1)。否则，将根据应用程序指定的缓冲类型，在*buffer和*length内返回数据及其长度。每个数值类型和临时类型都有固定的长度，请参见下面的表格。字符串类型的长度取决于由data_length指明的实际数据值的长度。

类型	长度
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

返回值

返回值	描述
0	成功，数据被提取到应用程序数据缓冲区。
1	出现错误。通过调用mysql_stmt_errno()和mysql_stmt_error()，可获取错误代码和错误消息。
MYSQL_NO_DATA	不存在行／数据。
MYSQL_DATA_TRUNCATED	出现数据截短。

不返回MYSQL_DATA_TRUNCATED，除非用mysql_options()启用了截短通报功能。返回该值时，为了确定截短的参数是哪个，可检查MYSQL_BIND参数结构的错误成员。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

· CR_UNSUPPORTED_PARAM_TYPE

缓冲类型为MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME、或MYSQL_TYPE_TIMESTAMP，但数据类型不是DATE、TIME、DATETIME、或TIMESTAMP。

· 从mysql_stmt_bind_result()返回所有其他不支持的转换错误。

示例:

在下面的示例中，介绍了使用mysql_stmt_result_metadata()、mysql_stmt_bind_result()和mysql_stmt_fetch()从表中获取数据的方法。（在本示例中，将检索在25.2.7.10节，“mysql_stmt_execute()”一节的示例中插入的两行内容）。假定mysql变量具有有效的连接句柄。

```
#define STRING_SIZE 50


#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 FROM test_table"


MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
```

```
my_bool       is_null[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout, " total columns in SELECT statement: %d\n", column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}
```

```
/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

```
/* Now buffer all results to client */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                    ts.year, ts.month, ts.day,
                    ts.hour, ts.minute, ts.second,
                    length[3]);
```

```
  fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);



/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

25.2.7.12. mysql_stmt_fetch_column()

int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int column, unsigned long offset)

描述

从当前结果集行获取1列。“bind”提供了应将数据置于其中的缓冲。其设置方法应与设置mysql_stmt_bind_result()的相同。“column”指明了将获取哪个列。第1列编号为0。“offset”是数据值内的偏移量，将从该处开始检索数据。可将其用于获取碎片形式的数据值。值开始部分的偏移量为0。

返回值

如果成功获取了值，返回0。如果出现错误，返回非0值。

错误

· CR_INVALID_PARAMETER_NO

Invalid column number.

· CR_NO_DATA

已抵达结果集的末尾。

25.2.7.13. mysql_stmt_field_count()

unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)

描述

为语句处理程序返回关于最近语句的行数。对于诸如INSERT或DELETE等不生成结果集的语句，该值为0。

通过调用mysql_stmt_prepare()准备好了语句后，可调用mysql_stmt_field_count()。

返回值

表示结果集中行数的无符号整数。

错误

无。

25.2.7.14. mysql_stmt_free_result()

my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)

描述

释放与执行预处理语句生成的结果集有关的内存。对于该语句，如果存在打开的光标，mysql_stmt_free_result()将关闭它。

返回值

如果成功释放了结果集，返回0。如果出现错误，返回非0值。

错误

25.2.7.15. mysql_stmt_init()

MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)

描述

创建MYSQL_STMT句柄。对于该句柄，应使用mysql_stmt_close(MYSQL_STMT *)释放。

返回值

成功时，返回指向MYSQL_STMT结构的指针。如果内存溢出，返回NULL。

错误

· CR_OUT_OF_MEMORY

内存溢出。

25.2.7.16. mysql_stmt_insert_id()

my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)

描述

返回预处理INSERT或UPDATE语句为AUTO_INCREMENT列生成的值。在包含AUTO_INCREMENT字段的表上执行了预处理INSERT语句后，使用该函数。

更多信息，请参见25.2.3.36节，“mysql_insert_id()”。

返回值

为在执行预处理语句期间自动生成或明确设置的AUTO_INCREMENT列返回值，或由LAST_INSERT_ID(*expr*)函数生成的值。如果语句未设置AUTO_INCREMENT值，返回值不确定。

错误

无。

25.2.7.17. mysql_stmt_num_rows()

my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)

描述

返回结果集中的行数。

mysql_stmt_num_rows()的用法取决于是否使用了mysql_stmt_store_result()来对语句句柄中的全部结果集进行了缓冲处理。

如果使用了mysql_stmt_store_result()，可立刻调用mysql_stmt_num_rows()。

返回值

结果集中的行数。

错误

无。

25.2.7.18. mysql_stmt_param_count()

unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)

描述

返回预处理语句中参数标记符的数目。

返回值

表示语句中参数数目的无符号长整数。

错误

无。

示例:

关于mysql_stmt_param_count()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.19. mysql_stmt_param_metadata()

MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)

该函数目前不做任何事。

描述

返回值

错误

25.2.7.20. mysql_stmt_prepare()

int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)

描述

给定mysql_stmt_init()返回的语句句柄，准备字符串查询指向的SQL语句，并返回状态值。字符串长度应由“length”参量给出。字符串必须包含1条SQL语句。不应为语句添加终结用分号(‘;’)或\g。

通过将问号字符“?”嵌入到SQL字符串的恰当位置，应用程序可包含SQL语句中的一个或多个参数标记符。

标记符仅在SQL语句中的特定位置时才是合法的。例如，它可以在INSERT语句的VALUES()列表中（为行指定列值），或与WHERE子句中某列的比较部分（用以指定比较值）。但是，对于ID（例如表名或列名），不允许使用它们，不允许指定二进制操作符（如等于号“=”）的操作数。后一个限制是有必要的，原因在于，无法确定参数类型。一般而言，参数仅在DML（数据操作语言）语句中才是合法的，在DDL（数据定义语言）语句中不合法。

执行语句之前，必须使用mysql_stmt_bind_param()，将参数标记符与应用程序变量绑定在一起。

返回值

如果成功处理了语句，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

如果准备操作失败（即mysql_stmt_prepare()返回非0值），可通过调用mysql_stmt_error()获取错误消息。

示例:

关于mysql_stmt_prepare()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.21. mysql_stmt_reset()

my_bool mysql_stmt_reset(MYSQL_STMT *stmt)

描述

在客户端和服务器上，将预处理语句复位为完成准备后的状态。主要用于复位用mysql_stmt_send_long_data()发出的数据。对于语句，任何已打开的光标将被关闭。

要想重新准备用于另一查询的语句，可使用mysql_stmt_prepare()。

返回值

如果语句成功复位，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.7.22. mysql_stmt_result_metadata()

MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)

描述

如果传递给mysql_stmt_prepare()的语句能够成生结果集，mysql_stmt_result_metadata()将以指针的形式返回结果集元数据，该指针指向MYSQL_RES结构，可用于处理元信息，如总的字段数以及单独的字段信息。该结果集指针可作为参量传递给任何基于字段且用于处理结果集元数据的API函数，如：

· mysql_num_fields()

· mysql_fetch_field()

· mysql_fetch_field_direct()

· mysql_fetch_fields()

· mysql_field_count()

· mysql_field_seek()

· mysql_field_tell()

· mysql_free_result()

完成操作后，应释放结果集结构，可通过将其传递给mysql_free_result()完成。它与释放通过mysql_store_result()调用获得的结果集的方法类似。

mysql_stmt_result_metadata()返回的结果集仅包含元数据。不含任何行结果。与mysql_stmt_fetch()一起使用语句句柄，可获取行。

返回值

MYSQL_RES结果结构。如果不存在关于预处理查询的任何元信息，返回NULL。

错误

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例：

关于mysql_stmt_result_metadata()的用法，请参见25.2.7.11节，“mysql_stmt_fetch()”中给出的示例。

25.2.7.23. mysql_stmt_row_seek()

MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)

描述

将行光标设置到语句结果集中的任意行。“offset”值是行偏移的值，行偏移应是从mysql_stmt_row_tell()或mysql_stmt_row_seek()返回的值。该值不是行编号，如果打算按编号查找结果集中的行，可使用mysql_stmt_data_seek()取而代之。

该函数要求结果集结构包含查询的全部结果，以便mysql_stmt_row_seek()能够仅与mysql_stmt_store_result()一起使用。

返回值

行光标的前一个值。可以将该值换递给后续的mysql_stmt_row_seek()调用。

错误

无。

25.2.7.24. mysql_stmt_row_tell()

MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)

描述

返回针对前一个mysql_stmt_fetch()的行光标的当前位置。该值可用作mysql_stmt_row_seek()的参量。

仅应在mysql_stmt_store_result()之后使用mysql_stmt_row_tell()。

返回值

行光标的当前偏移量。

错误

无。

25.2.7.25. mysql_stmt_send_long_data()

my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number, const char *data, unsigned long length)

描述

允许应用程序分段地（分块）将参数数据发送到服务器。可以多次调用该函数，以便发送关于某一列的字符或二进制数据的不同部分，列必须是TEXT或BLOB数据类型之一。

“parameter_number”指明了与数据关联的参数。参数从0开始编号。“data”是指向包含将要发送的数据的缓冲区的指针，“length”指明了缓冲区内的字节数。

注释：自上一个mysql_stmt_execute()或mysql_stmt_reset()后，对于与mysql_stmt_send_long_data()一起使用的所有参数，下一个mysql_stmt_execute()调用将忽略绑定缓冲。

如果希望复位／忽略已发送的数据，可使用mysql_stmt_reset()。请参见25.2.7.21节，“mysql_stmt_reset()”。

返回值

如果成功地将数据发送到服务器，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

在下面的示例中，介绍了以信息块形式为TEXT列发送数据的方法。它会将数据值“MySQL，最流行的开放源码数据库”插入到text_column列中。假定mysql变量具有有效的连接句柄。

```
#define INSERT_QUERY "INSERT INTO test_long_data(text_column) VALUES(?)"


MYSQL_BIND bind[1];
long       length;


smtt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}


 /* Supply data in chunks to server */
 if (!mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

```
 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0," - The most popular Open Source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

25.2.7.26. mysql_stmt_sqlstate()

const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_sqlstate()返回由Null终结的字符串，该字符串包含针对最近调用预处理语句API函数的SQLSTATE错误代码，该函数或成功或失败。错误代码由5个字符构成。"00000"表示“无错误”。这些值由ANSI SQL和ODBC指定。关于可能值的列表，请参见附录B：错误代码和消息。

注意，并非所有的MySQL错误均会被映射到SQLSTATE代码。值"HY000"（一般错误）用于未映射的错误。

返回值

包含SQLSTATE错误代码、由Null终结的字符串。

25.2.7.27. mysql_stmt_store_result()

int mysql_stmt_store_result(MYSQL_STMT *stmt)

描述

对于成功生成结果集的所有语句（SELECT、SHOW、DESCRIBE、EXPLAIN），而且仅当你打算对客户端的全部结果集进行缓冲处理时，必须调用mysql_stmt_store_result()，以便后续的mysql_stmt_fetch()调用能返回缓冲数据。

对于其他语句，没有必要调用mysql_stmt_store_result()，但如果调用了它，也不会造成任何伤害或导致任何性能问题。通过检查mysql_stmt_result_metadata()是否返回NULL，可检测语句是否生成了结果集。更多信息，请参见25.2.7.22节，“mysql_stmt_result_metadata()”。

注释：默认情况下，对于mysql_stmt_store_result()中的所有列，MySQL不计算MYSQL_FIELD->max_length，这是因为，计算它会显著降低mysql_stmt_store_result()的性能，而且大多数应用程序不需要max_length。如果打算更新max_length，可通过调用mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)启用它。请参见25.2.7.3节，“mysql_stmt_attr_set()”。

返回值

如果成功完成了对结果的缓冲处理，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.8. C API预处理语句方面的问题

下面列出了一些目前已知的与预处理语句有关的问题：

· TIME、TIMESTAMP和DATETIME不支持秒部分，例如来自DATE_FORMAT()的秒部分。

· 将整数转换为字符串时，在某些情况下，当MySQL不打印前导0时，可与预处理语句一起使用ZEROFILL。例如，与MIN(number-with-zerofill)一起。

· 将浮点数转换为客户端中的字符串时，被转换值最右侧的位可能会与原始值的有所不同。

· 预处理语句不使用查询高速缓冲，即使当查询不含任何占位符时也同样。。请参见5.13.1节，“查询高速缓冲如何工作”。

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。

25.2.10. 日期和时间值的C API处理

二进制协议允许你使用MYSQL_TIME结构发送和接受日期和时间值（DATE、TIME、DATETIME和TIMESTAMP）。在25.2.5节，“C API预处理语句的数据类型”中，介绍了该结构的成员。

要想发送临时数据值，可使用mysql_stmt_prepare()创建预处理语句。然后，在调用mysql_stmt_execute()执行语句之前，可采用下述步骤设置每个临时参数：

1. 在与数据值相关的MYSQL_BIND结构中，将buffer_type成员设置为相应的类型，该类型指明了发送的临时值类型。对于DATE、TIME、DATETIME或TIMESTAMP值，将buffer_type分别设置为MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME或MYSQL_TYPE_TIMESTAMP。

2. 将MYSQL_BIND结构的缓冲成员设置为用于传递临时值的MYSQL_TIME结构的地址。

3. 填充MYSQL_TIME结构的成员，使之与打算传递的临时支的类型相符。

使用mysql_stmt_bind_param()将参数数据绑定到语句。然后可调用mysql_stmt_execute()。

要想检索临时值，可采用类似的步骤，但应将buffer_type成员设置为打算接受的值的类型，并将缓冲成员设为应将返回值置于其中的MYSQL_TIME结构的地址。调用mysql_stmt_execute()之后，并在获取结果之前，使用mysql_bind_results()将缓冲绑定到语句上。

下面给出了一个插入DATE、TIME和TIMESTAMP数据的简单示例。假定mysql变量具有有效的连接句柄。

```
  MYSQL_TIME  ts;
  MYSQL_BIND  bind[3];
  MYSQL_STMT  *stmt;

  strmov(query, "INSERT INTO test_table(date_field, time_field,
                                        timestamp_field) VALUES(?,?,?");

  stmt = mysql_stmt_init(mysql);
  if (!stmt)
  {
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
  }
  if (mysql_stmt_prepare(mysql, query, strlen(query)))
  {
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
  }

  /* set up input buffers for all 3 parameters */
  bind[0].buffer_type= MYSQL_TYPE_DATE;
  bind[0].buffer= (char *)&ts;
  bind[0].is_null= 0;
  bind[0].length= 0;
  ...
  bind[1]= bind[2]= bind[0];
  ...

  mysql_stmt_bind_param(stmt, bind);

  /* supply the data to be sent in the ts structure */
  ts.year= 2002;
  ts.month= 02;
  ts.day= 03;

  ts.hour= 10;
```

```
  ts.minute= 45;
  ts.second= 20;

  mysql_stmt_execute(stmt);
  ..
```

25.2.11. C API线程函数介绍

当你打算创建线程客户端时，需要使用下述函数。请参见25.2.15节，“如何生成线程式客户端”。

25.2.11.1. my_init()

void my_init(void)

描述

调用任何MySQL函数之前，需要在程序中调用该函数。它将初始化MySQL所需的某些全局变量。如果你正在使用线程安全客户端库，它还能为该线程调用mysql_thread_init()。

通过mysql_init()、mysql_library_init()、mysql_server_init()和mysql_connect()，可自动调用该函数。

返回值

无。

25.2.11.2. mysql_thread_init()

my_bool mysql_thread_init(void)

描述

对于每个创建的线程，需要调用该函数来初始化与线程相关的变量。

它可由my_init()和mysql_connect()自动调用。

返回值

如果成功，返回0，如果出现错误，返回非0值。

25.2.11.3. mysql_thread_end()

void mysql_thread_end(void)

描述

调用pthread_exit()来释放mysql_thread_init()分配的内存之前，需要调用该函数。

注意，该函数不会被客户端库自动调用。必须明确调用它以避免内存泄漏。

返回值

无。

25.2.11.4. mysql_thread_safe()

unsigned int mysql_thread_safe(void)

描述

该函数指明了客户端是否编译为线程安全的。

返回值

如果客户端是线程安全的，返回1，否则返回0。

25.2.12. C API嵌入式服务器函数介绍

如果希望允许应用程序链接到嵌入式MySQL服务器库，必须使用mysql_server_init()和mysql_server_end()函数。请参见25.1节，“libmysqld，嵌入式MySQL服务器库”。

但是，要想提供改进的内存管理，即使是对与“-lmysqlclient”而不是与“-lmysqld”链接的程序，也应包含启用和结束库使用的调用。mysql_library_init()和mysql_library_end()函数可用于该目的。它们实际上是使其等效于mysql_server_init()和mysql_server_end()的#define符号，但它们的名称更清楚地指明，无论应用程序使用的是libmysqlclient或libmysqld，开始使用或结束MySQL C API库的使用时，应调用它们。关于更多信息，请参见25.2.2节，“C API函数概述”。

25.2.12.1. mysql_server_init()

int mysql_server_init(int argc, char **argv, char **groups)

描述

调用任何其他MySQL函数之前，必须在使用嵌入式服务器的程序中调用该函数。它将启动服务器，并初始化服务器使用的任何子系统（mysys、InnoDB等）。如果未调用该函数，对mysql_init()的下一次调用将执行mysql_server_init()。如果你正在使用与MySQL一起提供的DBUG软件包，应在调用了my_init()之后调用它。

对于main()的参量，argc和argv是类似的参量。argv的第1个元素将被忽略（典型情况下，它包含程序名）。为了方便起见，如果没有针对服务器的命令行参量，argc可以是0。mysql_server_init()将复制参量，以便能够在调用之后安全地摧毁argv或groups。

如果打算连接到外部服务器而不启动嵌入式服务器，应为argc指定负值。

“groups”中以Null终结的字符串列表选择了选项文件中的活动“groups”。请参见4.3.2节，“使用选项文件”。为了方便起见，groups可以是NULL，在该情况下，[server]和[embedded]组是活动的。

示例：

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


static char *server_args[] = {
  "this_program",       /* this string is not used */
  "--datadir=.",
  "--key_buffer_size=32M"
};
static char *server_groups[] = {
  "embedded",
  "server",
  "this_program_SERVER",
  (char *)NULL
};
```

```
int main(void) {
  if (mysql_server_init(sizeof(server_args) / sizeof(char *),
                        server_args, server_groups))
    exit(1);

  /* Use any MySQL API functions here */

  mysql_server_end();

  return EXIT_SUCCESS;
}
```

返回值

如果OK，返回0。如果出现错误，返回1。

25.2.12.2. mysql_server_end()

void mysql_server_end(void)

描述

在所有其他MySQL函数后，在程序中必须调用该函数一次。它将关闭嵌入式服务器。

返回值

无。

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

成功调用mysql_query()后，mysql_store_result()能够返回NULL。出现该情况时，表明出现了下述条件之一：

- 出现了malloc()故障（例如，如果结果集过大）。
- 无法读取数据（在连接上出现了错误）。
- 查询未返回数据（例如，它是INSERT、UPDATE或DELETE）。

通过调用mysql_field_count()，始终能检查语句是否应生成非空结果。如果mysql_field_count()返回0，结果为空，而且上一个查询是未返回值的语句（例如INSERT或DELETE）。如果mysql_field_count()返回非0值，语句应生成非空结果。关于这方面的示例，请参见mysql_field_count()函数介绍。

通过调用mysql_error()或mysql_errno()，可测试是否出现了错误。

25.2.13.2. What Results You Can Get from a Query

除了查询返回的结果集外，还能获取下述信息：

- 执行INSERT、UPDATE或DELETE时，mysql_affected_rows()返回上次查询影响的行数。

对于快速在创建，请使用TRUNCATE TABLE。

· mysql_num_rows()返回结果集中的行数。使用mysql_store_result()，一旦mysql_store_result()返回，就能调用mysql_num_rows()。使用mysql_use_result()，仅当用mysql_fetch_row()获取了所有行后，才能调用mysql_num_rows()。

· mysql_insert_id()返回上次查询生成的ID，该查询使用AUTO_INCREMENT索引将行插入到表内。请参见25.2.3.36节，“mysql_insert_id()”。

· 某些查询（LOAD DATA INFILE ...、INSERT INTO ... SELECT ...、UPDATE）将返回额外信息。结果由mysql_info()返回。关于它返回的字符串格式，请参见关于mysql_info()的介绍。如果没有额外信息，mysql_info()将返回NULL指针。

25.2.13.3. 如何获得上次插入行的唯一ID

如果将记录插入包含AUTO_INCREMENT列的表中，通过调用mysql_insert_id()函数，可获取保存在该列中的值。

通过执行下述代码，可从C应用程序检查某一值是否保存在AUTO_INCREMENT列中（假定该语句已成功执行）。它能确定查询是否是具有AUTO_INCREMENT索引的INSERT：

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

关于更多信息，请参见25.2.3.36节，“mysql_insert_id()”。

生成新的AUTO_INCREMENT值时，也能与mysql_query()一起通过执行SELECT LAST_INSERT_ID()语句获得它，并从该语句返回的结果集检索该值。

对于LAST_INSERT_ID()，最近生成的ID是在服务器上按连接维护的。它不会被另一个客户端改变。即使用non-magic值（即非Null非0值）更新了另一个AUTO_INCREMENT列，也不会更改它。

如果打算使用从某一表生成的ID，并将其插入到第2个表中，可使用如下所示的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
```

注意，mysql_insert_id()返回保存在AUTO_INCREMENT列中的值，无论该值是因存储NULL或0而自动生成的，或是明确指定的，均如此。LAST_INSERT_ID()仅返回自动生成的AUTO_INCREMENT值。如果你保存了除NULL或0之外的确切值，不会影响LAST_INSERT_ID()返回的值。

25.2.13.4. 与C API有关的问题

与C API链接时，在某些系统上可能出现下述错误：

```
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
```

如果在你的系统上出现了该情况，必须在编译/链接行的末尾增加“-lm”，通过该方式包含数学库。

25.2.14. 创建客户端程序

如果你编译了自己编写的MySQL客户端，或编译了从第三方获取的MySQL客户端，必须在链接命令中使用“-lmysqlclient -lz”选项链接它们。你或许还应指定“-L”选项，通知链接程序到哪里找到库。例如，如果将库安装到了/usr/local/mysql/lib，可在链接命令中使用sr/local/mysql/lib -lmysqlclient –lz。

对于使用MySQL头文件的客户端，编译它们时还须指定“-I”选项（例如，-I/usr/local/mysql/include），以便编译器能找到头文件。

为了使在Unix平台上编译MySQL程序变得简单，提供了**mysql_config**脚本。请参见25.9.2节，“mysql_config：获取编译客户端的编译选项”。

你也可以使用它来编译MySQL客户端，如下所述：

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

需要使用“sh –c”，使得shell不将**mysql_config**的输出当作1个词对待。

25.2.15. 如何生成线程式客户端

客户端库总是线程安全的。最大的问题在于从套接字读取的net.c中的子程序并不是中断安全的。或许你可能希望用自己的告警中断对服务器的长时间读取，以此来解决问题。如果为SIGPIPE中断安装了中断处理程序，套接字处理功能应是线程安全的。

为了避免连接中断时放弃程序，MySQL将在首次调用mysql_server_init()、mysql_init()或mysql_connect()时屏蔽SIGPIPE。如果你打算使用自己的SIGPIPE处理程序，首先应调用mysql_server_init()，然后安装你的处理程序，

在较旧的发布在我方网站上（http://www.mysql.com/）的二进制版本中，未用线程安全的选项对客户端库进行正常编译（默认情况下，编译的Windows二进制版本是线程安全的）。较新的二进制分发版应是正常的和线程安全的客户端库。

为了获得能从其他线程中断客户端的线程式客户端，并在与MySQL服务器通信时设置超时，应使用“-lmysys”、“-lmystrings”和“-ldbug”库，以及服务器使用的net_serv.o代码。

如果你不需要使用中断或超时，可编译线程安全客户端库（mysqlclient_r）并使用它。。请参见25.2节，“MySQL C API”。在该情况下，不必担心net_serv.o对象文件或其他MySQL库。

使用线程式客户端并打算使用超时或中断时，可更好地利用thr_alarm.c文件中的子程序。如果你正在使用来自mysys库的子程序，唯一需要记住的事是首先调用my_init()！请参见25.2.11节，“C API线程函数介绍”。

对于除mysql_real_connect()外的所有函数，在默认情况下它们均是线程安全的。在下面的说明中，介绍了编译线程安全客户端库的方法，以及以线程安全方式使用它的方法。（下面关于mysql_real_connect()的说明实际上也适用于mysql_connect()，但由于mysql_connect()已不再被重视，总应尽量使用mysql_real_connect()）。

要想使mysql_real_connect()成为线程安全的，必须用下述命令再次编译客户端库：

```
shell> ./configure --enable-thread-safe-client
```

它创建了线程安全客户端库libmysqlclient_r。（假定你的操作系统有线程安全的gethostbyname_r()函数）。按照连接，该库是线程安全的。可遵循下述警告，使两个线程共享相同的连接：

· 在相同的连接上，两个线程不能同时将查询发送到MySQL服务器。尤其是，必须确保在mysql_query()和mysql_store_result()之间，没有使用相同连接的其他线程。

· 很多线程均能访问由mysql_store_result()检索的不同结果集。

· 如果使用了mysql_use_result，务必确保无其他线程正在使用相同的连接，直至关闭了结果集为止。然而，对于线程式客户端，最好是共享相同的连接以使用mysql_store_result()。

· 如果打算在相同的连接上使用多个线程，必须在mysql_query()和mysql_store_result()调用组合上拥有互斥锁。一旦mysql_store_result()准备就绪，可释放锁定，其他线程可在相同的连接上执行查询。

· 如果使用POSIX线程进行编程，可使用pthread_mutex_lock()和pthread_mutex_unlock()来建立并释放互斥锁。

如果你有1个调用MySQL函数的线程，而该函数未创建与MySQL数据库的连接，就需了解下述事宜：

调用mysql_init()或mysql_connect()时，MySQL会为调试库使用的线程创建与线程相关的变量（尤其）。

在线程调用mysql_init()或mysql_connect()之前，如果调用了MySQL函数，该线程将没有所需的线程类变量，而且你很可能或早或晚结束于内核转储。

要想使这些操作平稳工作，需要采取下述措施：

1. 如果程序在调用mysql_real_connect()之前需要调用任何其他MySQL函数，请在启动程序时调用my_init()。

2. 调用任何MySQL函数之前，在线程处理程序中调用mysql_thread_init()。

3. 在线程中，调用pthread_exit()之前请调用mysql_thread_end()。这样，就能释放MySQL线程类变量使用的内存。

将客户端链接到libmysqlclient_r时，如果存在未定义的符号，可能会出错。在大多数情况下，其原因在于，未将线程库包含在link/compile行上。

25.3. MySQL PHP API

PHP是一种服务器端、HTML嵌入式脚本处理语言，可使用该语言创建动态网页。它可用于大多数操作系统和Web服务器，也能访问大多数常见数据库，包括MySQL。PHP可以作为单独程序运行，也能编译为模块，用于Apache Web服务器。

PHP实际上提供了两种不同的MySQL API扩展：

· mysql：适用于PHP版本4和5，该扩展用于MySQL 4.1之前的MySQL版本。该扩展不支持MySQL 5.1中采用的、改进的鉴定协议，也不支持与预处理语句或多语句。如果打算与MySQL 5.1一起使用该扩展，应配置MySQL服务器，以使用“**--old-passwords**”选项（请参见A.2.3节，“客户端不支持鉴定协议”）。在PHP网站的文档中记录了该该扩展http://php.net/mysql。

· mysqli是“MySQL, Improved”的缩写，该扩展仅适用于PHP 5。它能用于MySQL 4.1.1和更高版本。该扩展完全支持MySQL 5.1中采用的鉴定协议，也支持预处理语句和多语句API。此外，该扩展还提供了先进的、面向对象的编程接口。在http://php.net/mysqli上，可找到关于mysqli扩展的文档。在http://www.zend.com/php5/articles/php5-mysqli.php处，给出了一篇有用的文章。

PHP分发版和文档均能从PHP网站获得。

25.3.1. 使用MySQL和PHP的常见问题

· 错误：超出了最大执行时间，这是一种PHP限制，如果需要，进入文件php.ini，并设置最大执行时间（开始为30秒）。此外，还可以将每脚本允许使用的RAM增加一倍，从8MB变为16MB，这也是个不错的主意。

· 致命错误：在…中调用了不支持或未定义的mysql_connect()函数，这意味着，你的PHP版本不支持MySQL。你可以编译动态MySQL模块并将其加载到PHP，或使用内置的MySQL支持重新编译PHP。在PHP手册中，详细介绍了该进程。

· 错误：对'uncompress'的未定义引用，这意味着所编译的客户端库支持压缩客户端／服务器协议。更正方法是，用“-lmysqlclient”进行链接时，在最后添加“-lz”。

· 错误：客户端不支持鉴定协议，与MySQL 4.1.1和更高版本一起使用较旧的mysql扩展时常会遇到该问题。可能的解决方案是：降级到MySQL 4.0，转向PHP 5和较新的mysqli扩展，或用“--old-passwords”配置MySQL服务器（更多信息，请参见A.2.3节，“客户端不支持鉴定协议”）。

25.4. MySQL Perl API

Perl DBI模块为数据库访问提供了一个通用接口。能够编写无需更改就能与不同的数据库引擎一起工作的DBI脚本。要想使用DBI，必须安装DBI模块，并为打算访问的每种服务器安装数据库驱动程序（DBD）模块。对于MySQL，该驱动程序是DBD::mysql模块。

Perl DBI是推荐的Perl接口。它取代了旧的名为mysqlperl的接口，mysqlperl已过时。

关于Perl DBI支持的安装说明，请参见2.13节，“Perl安装注意事项”。

DBI信息能够在命令行上提供，也能以在线方式提供，或采用印刷形式：

· 一旦安装了DBI和DBD::mysql模块，可使用perldoc 命令在命令行上获取关于它们的信息：

· shell> **perldoc DBI**

· shell> **perldoc DBI::FAQ**

· shell> **perldoc DBD::mysql**

也可以使用pod2man、pod2html等将这类信息转换为其他格式。

· 关于Perl DBI的在线信息，请访问DBI网站，http://dbi.perl.org/。该站点还提供了1个一般性DBI邮件列表。MySQL AB提供了1个专门针对DBD::mysql 的邮件列表，请参见1.7.1.1 “MySQL邮件列表”。

· 至于印刷版信息，官方的DBI书籍是编程*Perl DBI*（Alligator Descartes和Tim Bunce，O'Reilly & Associates, 2000）。关于该书的信息，请访问DBI网站http://dbi.perl.org/。

关于与MySQL一起使用DBI的专门信息，请参见针对*Web*的*MySQL*和*Perl*（Paul DuBois, New Riders, 2001）。该书的网站是http://www.kitebird.com/mysql-perl/。

25.5. MySQL C++ API

MySQL++是用于C++的MySQL API。Warren Young负责该项目。要想了解更多信息，请访问http://www.mysql.com/products/mysql++/。

25.5.1. Borland C++

可以使用Borland C++ 5.02编译MySQL Windows源码（Windows源码仅包括用于Microsoft VC++的项目，对于Borland C++，你将不得不自己编制项目文件）。

使用Borland C++时的1个已知问题是，它采用了不同于VC++的结构对齐方式。这意味着，如果你打算与Borland C++一起使用默认的libmysql.dll库（它是使用VC++编译的），将会遇到问题。为了避免该问题，仅应调用将Null作为参量的mysql_init()，而不是预先分配MYSQL结构。

25.6. MySQL Python API

MySQLdb为Python提供了MySQL支持，它符合Python DB API版本2.0的要求，可在http://sourceforge.net/projects/mysql-python/上找到它。

25.7. MySQL Tcl API

MySQLtcl是一种简单的API，用于从Tcl编程语言访问MySQL数据库服务器。可在http://www.xdobry.de/mysqltcl/上找到它。

25.8. MySQL Eiffel Wrapper

Eiffel MySQL是一种与MySQL数据库服务器的接口，它采用的是Eiffel编程语言，由Michael Ravits编写。可在http://efsa.sourceforge.net/archive/ravits/mysql.htm上找到它。

25.9. MySQL程序开发实用工具

在本节中，介绍了开发MySQL程序时可能会有用的一些实用工具。

· msql2mysql

1种shell脚本，用于将mSQL程序转换为MySQL程序。它不能处理所有情况，但能为转换提供良好的开端。

· mysql_config

1种shell脚本，能生成编译MySQL程序时所需的选项值。

25.9.1. msql2mysql：转换mSQL程序以用于MySQL

最初，开发的MySQL C API很类似为mSQL数据库系统开发的API。正因为如此，通过更改C API函数的名称，通常能相对容易地转换mSQL程序，使之用于MySQL。

msql2mysql实用工具用于将mSQL C API函数调用转换为其MySQL对等物。**msql2mysql**能够转换位于恰当位置输入文件，在执行具体转换之前复制原件。例如，可采用下述方式使用**msql2mysql**：

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

然后，检查client-prog.c，并执行可能需要的后期转换修订。

msql2mysql使用**replace**实用工具来替换函数名。请参见8.14节，“replace：字符串替换实用工具”。

25.9.2. mysql_config：获取编译客户端的编译选项

mysql_config提供了关于编译MySQL客户端以及将其连接到MySQL的有用信息。

mysql_config支持下述选项：

· --cflags

编译器标志，用于查找包含文件，以及编译libmysqlclient库时所要使用的关键编译器标志和定义。

· --include

编译器选项，用于查找MySQL包含文件（注意，正常情况下应使用“—cflags”而不是该选项）。

· --libmysqld-libs, ---embedded

与MySQL嵌入式服务器进行链接所需的库和选项。

· --libs

与MySQL客户端库进行链接所需的库和选项。

· --libs_r

与线程安全MySQL客户端库进行链接所需的库和选项。

· --port

默认的TCP/IP端口号，配置MySQL时定义。

· --socket

默认的Unix套接字文件，配置MySQL时定义。

· --version

版本号以及MySQL分发版的版本。

如果未使用任何选项调用了**mysql_config**，将显示它所支持的所有选项的列表，以及它们的值：

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
```

```
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
                    -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lz -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [4.0.16]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                    -lcrypt -lnsl -lm -lpthread -lrt]
```

可以在命令行上使用**mysql_config**，并包含针对特定选项的值。例如，要想编译MySQL客户端程序，可使用**mysql_config**，如下例所示：

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

以这种方式使用**mysql_config**时，务必在字符(‘`’)内调用它。这样，就能通知shell执行它，并将其输出代入到环境命令中。
