**20 MySQL客户工具和API**

**20.1 MySQL C API**

C API代码是随**MySQL**分发的，它被包含在mysqlclient库且允许C程序存取一个数据库。

在 MySQL源代码分发中的很多客户是用C编写的。如果你正在寻找演示怎样使用C API的例子，看一下这些客户程序。

大多数其他客户 API(除了Java的所有)都使用mysqlclient库与**MySQL**服务器通信。这意味着，例如，你能利用很多被其他客户程序使用的同一环境变量，因为他们从库中引用。对这些变量的一张表，见[12.1 不同的MySQL程序的概述](http://www.yesky.com/imagesnew/software/mysql/manual_Tools.html#Programs)。

客户有一个最大通讯缓冲区大小。初始分配的缓冲区大小(16K字节) 自动地增加到最大尺寸(缺省的最大值是24M)。因为缓冲区大小只是按保证需求而被增加，简单地增加缺省的最大限制并不造成更多被消耗。该尺寸检查主要是一个对错误的查询和通讯包的检查。

通讯缓冲区必须足够大以便一个单独的SQL语句(对客户-服务器传输)和一行返回的数据(对服务器-客户传输)。每个线程的通讯缓冲区被动态扩大到最大限制来处理任何查询或行。例如，如果你包含大到16M数据的BLOB值，你必须有一个至少16M通讯缓冲区限制(在服务器和客户两端)。客户的缺省最大值是24M，但是在服务器端的缺省最大值是1M。你可以在服务器启动时通过改变max\_allowed\_packet参数的值来改变它。见[10.2.3 调节服务器参数](http://www.yesky.com/imagesnew/software/mysql/manual_Performance.html#Server_parameters)。

**MySQL**服务器在每个查询后缩小每个通讯缓冲区到net\_buffer\_length个字节。对客户，与一个连接相关的缓冲区的大小没被减少，直到连接被关闭，在此时客户内存被回收。

如果你用线程的编程，你应该用--with-thread-safe-client编译**MySQL** C API，这将使C API线程对每个连接更安全。你可以让2个线程共享相同的连接，只要如果你做下列事情：

两个线程不能同时在同一个连接上发送查询到**MySQL**。特别是你必须保证在一个mysql\_query()和mysql\_store\_result()之间没有其他线程正在使用同一个连接。

许多线程能存取用mysql\_store\_result()检索出来的不同结果集合。

如果你使用mysql\_use\_result，你必须保证没有其他线程在同一个连接上正在询问任何东西，直到结果集合被关闭。

**20.2 C API数据类型**

MYSQL

这个结构表示对一个数据库连接的句柄，它被用于几乎所有的**MySQL**函数。

MYSQL\_RES

这个结构代表返回行的一个查询的(SELECT, SHOW, DESCRIBE, EXPLAIN)的结果。从查询返回的信息在本章下文称为*结果集合*。

MYSQL\_ROW

这是一个行数据的类型安全(type-safe)的表示。当前它实现为一个计数字节的字符串数组。（如果字段值可能包含二进制数据，你不能将这些视为空终止串，因为这样的值可以在内部包含空字节) 行通过调用mysql\_fetch\_row()获得。

MYSQL\_FIELD

这个结构包含字段信息，例如字段名、类型和大小。其成员在下面更详细地描述。你可以通过重复调用mysql\_fetch\_field()对每一列获得MYSQL\_FIELD结构。字段值不是这个结构的部分；他们被包含在一个MYSQL\_ROW结构中。

MYSQL\_FIELD\_OFFSET

这是一个相对一个**MySQL**字段表的偏移量的类型安全的表示。（由mysql\_field\_seek()使用。) 偏移量是在一行以内的字段编号，从0开始。

my\_ulonglong

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

printf (Number of rows: %lu\n", (unsigned long) mysql\_num\_rows(result));

MYSQL\_FIELD结构包含列在下面的成员：

char \* name

字段名，是一个空结尾的字符串。

char \* table

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

char \* def

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

enum enum\_field\_types type

字段类型。type值可以是下列之一：

|  |  |
| --- | --- |
| **类型值** | **类型含义** |
| FIELD\_TYPE\_TINY | TINYINT字段 |
| FIELD\_TYPE\_SHORT | SMALLINT字段 |
| FIELD\_TYPE\_LONG | INTEGER字段 |
| FIELD\_TYPE\_INT24 | MEDIUMINT字段 |
| FIELD\_TYPE\_LONGLONG | BIGINT字段 |
| FIELD\_TYPE\_DECIMAL | DECIMAL或NUMERIC字段 |
| FIELD\_TYPE\_FLOAT | FLOAT字段 |
| FIELD\_TYPE\_DOUBLE | DOUBLE或REAL字段 |
| FIELD\_TYPE\_TIMESTAMP | TIMESTAMP字段 |
| FIELD\_TYPE\_DATE | DATE字段 |
| FIELD\_TYPE\_TIME | TIME字段 |
| FIELD\_TYPE\_DATETIME | DATETIME字段 |
| FIELD\_TYPE\_YEAR | YEAR字段 |
| FIELD\_TYPE\_STRING | 字符串(CHAR或VARCHAR)字段 |
| FIELD\_TYPE\_BLOB | BLOB或TEXT字段(使用max\_length决定最大长度） |
| FIELD\_TYPE\_SET | SET字段 |
| FIELD\_TYPE\_ENUM | ENUM字段 |
| FIELD\_TYPE\_NULL | NULL- 类型字段 |
| FIELD\_TYPE\_CHAR | 不推荐；使用FIELD\_TYPE\_TINY代替 |

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

if (IS\_NUM(field->type))

printf("Field is numeric\n");

unsigned int length

字段宽度，在表定义中指定。

unsigned int max\_length

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

unsigned int flags

字段的不同位标志。flags值可以是零个或多个下列位设置：

|  |  |
| --- | --- |
| **标志值** | **标志含义** |
| NOT\_NULL\_FLAG | 字段不能是NULL |
| PRI\_KEY\_FLAG | 字段是一个主键的一部分 |
| UNIQUE\_KEY\_FLAG | 字段是一个唯一键的一部分 |
| MULTIPLE\_KEY\_FLAG | 字段是一个非唯一键的一部分。 |
| UNSIGNED\_FLAG | 字段有UNSIGNED属性 |
| ZEROFILL\_FLAG | 字段有ZEROFILL属性 |
| BINARY\_FLAG | 字段有BINARY属性 |
| AUTO\_INCREMENT\_FLAG | 字段有AUTO\_INCREMENT属性 |
| ENUM\_FLAG | 字段是一个ENUM（不推荐） |
| BLOB\_FLAG | 字段是一个BLOB或TEXT（不推荐） |
| TIMESTAMP\_FLAG | 字段是一个TIMESTAMP（不推荐） |

BLOB\_FLAG、ENUM\_FLAG和TIMESTAMP\_FLAG标志的使用是不推荐的，因为他们指出字段的类型而非它的类型属性。对FIELD\_TYPE\_BLOB、FIELD\_TYPE\_ENUM或FIELD\_TYPE\_TIMESTAMP，最好是测试field->type。下面例子演示了一个典型的flags值用法：

if (field->flags & NOT\_NULL\_FLAG)

printf("Field can't be null\n");

你可以使用下列方便的宏决来确定flags值的布尔状态：

|  |  |
| --- | --- |
| IS\_NOT\_NULL(flags) | 真，如果该字段被定义为NOT NULL |
| IS\_PRI\_KEY(flags) | 真，如果该字段是一个主键 |
| IS\_BLOB(flags) | 真，如果该字段是一个BLOB或TEXT（不推荐；相反测试field->type） |

unsigned int decimals

对数字字段的小数位数。

**20.3 C API函数概述**

在 C API 中可用的函数列在下面，并且在下一节更详细地描述。见[20.4 C API函数描述](http://www.yesky.com/imagesnew/software/mysql/manual_Clients.html#C_API_functions)。

|  |  |
| --- | --- |
| **mysql\_affected\_rows()** | 返回被最新的UPDATE, DELETE或INSERT查询影响的行数。 |
| **mysql\_close()** | 关闭一个服务器连接。 |
| **mysql\_connect()** | 连接一个**MySQL**服务器。该函数不推荐；使用mysql\_real\_connect()代替。 |
| **mysql\_change\_user()** | 改变在一个打开的连接上的用户和数据库。 |
| **mysql\_create\_db()** | 创建一个数据库。该函数不推荐；而使用SQL命令CREATE DATABASE。 |
| **mysql\_data\_seek()** | 在一个查询结果集合中搜寻一任意行。 |
| **mysql\_debug()** | 用给定字符串做一个DBUG\_PUSH。 |
| **mysql\_drop\_db()** | 抛弃一个数据库。该函数不推荐；而使用SQL命令DROP DATABASE。 |
| **mysql\_dump\_debug\_info()** | 让服务器将调试信息写入日志文件。 |
| **mysql\_eof()** | 确定是否已经读到一个结果集合的最后一行。这功能被反对; mysql\_errno()或mysql\_error()可以相反被使用。 |
| **mysql\_errno()** | 返回最近被调用的**MySQL**函数的出错编号。 |
| **mysql\_error()** | 返回最近被调用的**MySQL**函数的出错消息。 |
| **mysql\_escape\_string()** | 用在SQL语句中的字符串的转义特殊字符。 |
| **mysql\_fetch\_field()** | 返回下一个表字段的类型。 |
| **mysql\_fetch\_field\_direct ()** | 返回一个表字段的类型，给出一个字段编号。 |
| **mysql\_fetch\_fields()** | 返回一个所有字段结构的数组。 |
| **mysql\_fetch\_lengths()** | 返回当前行中所有列的长度。 |
| **mysql\_fetch\_row()** | 从结果集合中取得下一行。 |
| **mysql\_field\_seek()** | 把列光标放在一个指定的列上。 |
| **mysql\_field\_count()** | 返回最近查询的结果列的数量。 |
| **mysql\_field\_tell()** | 返回用于最后一个mysql\_fetch\_field()的字段光标的位置。 |
| **mysql\_free\_result()** | 释放一个结果集合使用的内存。 |
| **mysql\_get\_client\_info()** | 返回客户版本信息。 |
| **mysql\_get\_host\_info()** | 返回一个描述连接的字符串。 |
| **mysql\_get\_proto\_info()** | 返回连接使用的协议版本。 |
| **mysql\_get\_server\_info()** | 返回服务器版本号。 |
| **mysql\_info()** | 返回关于最近执行得查询的信息。 |
| **mysql\_init()** | 获得或初始化一个MYSQL结构。 |
| **mysql\_insert\_id()** | 返回有前一个查询为一个AUTO\_INCREMENT列生成的ID。 |
| **mysql\_kill()** | 杀死一个给定的线程。 |
| **mysql\_list\_dbs()** | 返回匹配一个简单的正则表达式的数据库名。 |
| **mysql\_list\_fields()** | 返回匹配一个简单的正则表达式的列名。 |
| **mysql\_list\_processes()** | 返回当前服务器线程的一张表。 |
| **mysql\_list\_tables()** | 返回匹配一个简单的正则表达式的表名。 |
| **mysql\_num\_fields()** | 返回一个结果集合重的列的数量。 |
| **mysql\_num\_rows()** | 返回一个结果集合中的行的数量。 |
| **mysql\_options()** | 设置对mysql\_connect()的连接选项。 |
| **mysql\_ping()** | 检查对服务器的连接是否正在工作，必要时重新连接。 |
| **mysql\_query()** | 执行指定为一个空结尾的字符串的SQL查询。 |
| **mysql\_real\_connect()** | 连接一个**MySQL**服务器。 |
| **mysql\_real\_query()** | 执行指定为带计数的字符串的SQL查询。 |
| **mysql\_reload()** | 告诉服务器重装授权表。 |
| **mysql\_row\_seek()** | 搜索在结果集合中的行，使用从mysql\_row\_tell()返回的值。 |
| **mysql\_row\_tell()** | 返回行光标位置。 |
| **mysql\_select\_db()** | 连接一个数据库。 |
| **mysql\_shutdown()** | 关掉数据库服务器。 |
| **mysql\_stat()** | 返回作为字符串的服务器状态。 |
| **mysql\_store\_result()** | 检索一个完整的结果集合给客户。 |
| **mysql\_thread\_id()** | 返回当前线程的ID。 |
| **mysql\_use\_result()** | 初始化一个一行一行地结果集合的检索。 |

为了连接服务器，调用mysql\_init()以初始化一个连接处理器，然后用该处理器调用mysql\_real\_connect()(还有其他信息例如主机名、用户名和口令)。当你用该连接完成工作后，调用mysql\_close()终止它。

当一个连接活跃时，客户可以用mysql\_query()或mysql\_real\_query()将SQL查询发送到服务器。两者的差别是mysql\_query()期望查询作为一个空结尾的字符串来指定而mysql\_real\_query()期望一个计数的字符串。如果字符串包含二进制数据(它可以包括空字节)，你必须使用mysql\_real\_query()。

对与每个非--SELECT查询(例如，INSERT、UPDATE、DELETE等)，你可以调用mysql\_affected\_rows()知道有多少行受到影响(改变)。

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

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

在两种情况中，你通过mysql\_fetch\_row()存取行。用mysql\_store\_result()、mysql\_fetch\_row()储存取已经从服务器被取出的行。用mysql\_use\_result()、mysql\_fetch\_row()实际上从服务器检索行。调用mysql\_fetch\_lengths()可获得关于每行中数据值尺寸的信息。

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

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

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

mysql\_use\_result()的一个优点是客户为结果集合需要较少的内存，因为它一次只是维持一行(并且因为有较少的分配开销，mysql\_use\_result()能更快些)。缺点是你必须尽快处理每一行以避免困住服务器，你不必再结果集合中随意存取行(你只能顺序存取行)，而且你不知道在结果集合中有多少行，直到你检索全部结果。还有，你*必须*检索出所有行，即使你在检索中途确定你已找到了想寻找的信息。

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

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

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

**20.4 C API函数描述**

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

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

if (result) /\* 正确 \*/

... error ...

if (result < 0) /\* 不正确 \*/

... error ...

if (result == -1) /\* 不正确 \*/

... error ...

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

**20.4.1 mysql\_affected\_rows()**

my\_ulonglong mysql\_affected\_rows(MYSQL \*mysql)

**20.4.1.1 说明**

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

mysql\_affected\_rows()目前以一个宏(macro)来实现。

**20.4.1.2 返回值**

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

**20.4.1.3 错误**

没有。

**20.4.1.4 范例**

mysql\_query(&mysql,"UPDATE products SET cost=cost\*1.25 WHERE group=10");

printf("%d products updated",mysql\_affected\_rows(&mysql));

**20.4.2 mysql\_close()**

void mysql\_close(MYSQL \*mysql)

**20.4.2.1 说明**

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

**20.4.2.2 返回值**

没有。

**20.4.2.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务者的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.3 mysql\_connect()**

MYSQL \*mysql\_connect(MYSQL \*mysql, const char \*host, const char \*user, const char \*passwd)

**20.4.3.1 说明**

该函数不推荐使用，而更好使用mysql\_real\_connect()。

mysql\_connect()试图建立一个对运行在host的一个**MySQL**数据库引擎的连接。mysql\_connect()必须在你能执行其他API函数之前成功地完成，除了mysql\_get\_client\_info()。

参数的含义与mysql\_connect()相应的参数相同，不同的是连接参数可以是NULL。在这种情况下，C API 自动为连接结构分配内存，并且当你调用mysql\_close()，释放它。这种方法的缺点是如果连接失败，你不能检索出一条错误消息。（为了从mysql\_errno()或mysql\_error()得到错误信息，你必须提供一个有效的MYSQL指针。）

**20.4.3.2 返回值**

同mysql\_real\_connect()。

**20.4.3.3 错误**

同mysql\_real\_connect()。

**20.4.4mysql\_change\_user()**

my\_bool mysql\_change\_user(MYSQL \*mysql, const char \*user, const char \*password, const char \*db)

**20.4.4.1 说明**

改变用户并且使得由db指定数据库成为由mysql指定的连接上的缺省(当前)数据库。在随后的查询中，这个数据库是不包括一个明确的数据库指定符的表引用的缺省值。

这个函数功能在**MySQL** 3.23.3中引入。

除非连接的用户能被认证或如果他没有权限使用数据库，mysql\_change\_user()失败。在这种情况下，用户和数据库都没被改变。

如果你不想有一个缺省数据库，db参数可以被设置为NULL。

**20.4.4.2 返回值**

成功，零。如果发生一个错误发生，非零。

**20.4.4.3 错误**

与你能从mysql\_real\_connect()得到的相同。

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务者关闭了。

CR\_SERVER\_LOST

对服务者的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

ER\_UNKNOWN\_COM\_ERROR

**MySQL**服务器未实现这个命令(可能是一个老的服务器）

ER\_ACCESS\_DENIED\_ERROR

用户或口令错误。

ER\_BAD\_DB\_ERROR

数据库不存在。

ER\_DBACCESS\_DENIED\_ERROR

用户没有数据库的存取权利。

ER\_WRONG\_DB\_NAME

数据库名字太长。

**20.4.4.4 范例**

if (mysql\_change\_user(&mysql, "user", "password", "new\_database"))

{

fprintf(stderr, "Failed to change user. Error: %s\n",

mysql\_error(&mysql));

}

**20.4.5 mysql\_create\_db()**

int mysql\_create\_db(MYSQL \*mysql, const char \*db)

**20.4.5.1 说明**

创建由db参数命名的数据库。

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

**20.4.5.2 返回值**

如果数据库成功地被创造，零。如果发生一个错误，非零。

**20.4.5.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务者的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.5.4 范例**

if(mysql\_create\_db(&mysql, "my\_database"))

{

fprintf(stderr, "Failed to create new database. Error: %s\n",

mysql\_error(&mysql));

}

**20.4.6 mysql\_data\_seek()**

void mysql\_data\_seek(MYSQL\_RES \*result, unsigned long long offset)

**20.4.6.1 说明**

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

偏移量应该是从0到mysql\_num\_rows(result)-1范围的一个值。

**20.4.6.2 返回值**

无。

**20.4.6.3 错误**

无。

**20.4.7 mysql\_debug()**

void mysql\_debug(char \*debug)

**20.4.7.1 说明**

用一个给定字符串做一个DBUG\_PUSH。mysql\_debug()使用Fred Fish 调试库。为了使用这个函数，你必须编译客户库以支持调试。见[G.1 调试一个MySQL服务器](http://www.yesky.com/imagesnew/software/mysql/manual_Porting.html#Debugging_server)和节[G.2 调试一个MySQL客户](http://www.yesky.com/imagesnew/software/mysql/manual_Porting.html#Debugging_client)。

**20.4.7.2 返回值**

无。

**20.4.7.3 错误**

无。

**20.4.7.4 范例**

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

mysql\_debug("d:t:O,/tmp/client.trace");

**20.4.8 mysql\_drop\_db()**

int mysql\_drop\_db(MYSQL \*mysql, const char \*db)

**20.4.8.1 说明**

抛弃由db参数命名的数据库。

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

**20.4.8.2 返回值**

如果数据库成功地被破抛弃，零。如果发生一个错误，非零。

**20.4.8.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务者的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.8.4 范例**

if(mysql\_drop\_db(&mysql, "my\_database"))

fprintf(stderr, "Failed to drop the database: Error: %s\n",

mysql\_error(&mysql));

**20.4.9 mysql\_dump\_debug\_info()**

int mysql\_dump\_debug\_info(MYSQL \*mysql)

**20.4.9.1 说明**

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

**20.4.9.2 返回值**

如果命令成功，零。如果发生一个错误，非零。

**20.4.9.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务者关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.10 mysql\_eof()**

my\_bool mysql\_eof(MYSQL\_RES \*result)

**20.4.10.1 说明**

这个函数不推荐，而使用mysql\_errno()或mysql\_error()。

mysql\_eof()确定是否已经读到了一个结果集合的最后一行。

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

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

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

**20.4.10.2 返回值**

如果发生一个错误，零。如果到达结果集合的结束，非零。

**20.4.10.3 错误**

无。

**20.4.10.4 范例**

下列例子显示你必须如何使用mysql\_eof()：

mysql\_query(&mysql,"SELECT \* FROM some\_table");

result = mysql\_use\_result(&mysql);

while((row = mysql\_fetch\_row(result)))

{

// do something with data

}

if(!mysql\_eof(result)) // mysql\_fetch\_row() failed due to an error

{

fprintf(stderr, "Error: %s\n", mysql\_error(&mysql));

}

然而，你可以用标准**MySQL**错误函数完成同样的效果：

mysql\_query(&mysql,"SELECT \* FROM some\_table");

result = mysql\_use\_result(&mysql);

while((row = mysql\_fetch\_row(result)))

{

// do something with data

}

if(mysql\_errno(&mysql)) // mysql\_fetch\_row() failed due to an error

{

fprintf(stderr, "Error: %s\n", mysql\_error(&mysql));

}

**20.4.11 mysql\_errno()**

unsigned int mysql\_errno(MYSQL \*mysql)

**20.4.11.1 说明**

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

**20.4.11.2 返回值：**

一个错误代码值。如果没有错误发生，零。

**20.4.11.3 错误**

无。

**20.4.12 mysql\_error()**

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

**20.4.12.1 说明**

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

if(mysql\_errno(&mysql))

{

// an error occurred

}

if(mysql\_error(&mysql)[0] != '\0')

{

// an error occurred

}

客户错误消息的语言可通过重新编译**MySQL**客户库来改变。目前，你能在几种不同的语言间选取错误消息。见[9.1 MySQL支持什么语言？](http://www.yesky.com/imagesnew/software/mysql/manual_Server.html#Languages)。

**20.4.12.2 返回值**

一个描述错误的字符串。如果没有错误发生，空字符串。

**20.4.12.3 错误**

无。

**20.4.13 mysql\_escape\_string()**

unsigned int mysql\_escape\_string(char \*to, const char \*from, unsigned int length)

**20.4.13.1 说明**

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

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

**20.4.13.2 范例**

char query[1000],\*end;

end = strmov(query,"INSERT INTO test\_table values(");

\*end++ = '\'';

end += mysql\_escape\_string(end,"What's this",11);

\*end++ = '\'';

\*end++ = ',';

\*end++ = '\'';

end += mysql\_escape\_string(end,"binary data: \0\r\n",16);

\*end++ = '\'';

\*end++ = ')';

if (mysql\_real\_query(&mysql,query,(unsigned int) (end - query)))

{

fprintf(stderr, "Failed to insert row, Error: %s\n",

mysql\_error(&mysql));

}

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

**20.4.13.3 返回值**

放进to的值的长度，不包括终止空字符。

**20.4.13.4 错误**

无。

**20.4.14 mysql\_fetch\_field()**

MYSQL\_FIELD \*mysql\_fetch\_field(MYSQL\_RES \*result)

**20.4.14.1 说明**

返回作为一个MYSQL\_FIELD结构的一个结果集合的一个列的定义。重复调用这个函数在结果集合中检索所有关于列的信息。当没有剩下更多的字段时，mysql\_fetch\_field()返回NULL。

在每次你执行一个新的SELECT查询，mysql\_fetch\_field()被重置（reset)以返回有关第一列的信息。由mysql\_fetch\_field()返回的字段也受调用mysql\_field\_seek()的影响。

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

**20.4.14.2 返回值**

当前列的MYSQL\_FIELD结构。如果没有列剩下，NULL。

**20.4.14.3 错误**

无。

**20.4.14.4 范例**

MYSQL\_FIELD \*field;

while((field = mysql\_fetch\_field(result)))

{

printf("field name %s\n", field->name);

}

**20.4.15 mysql\_fetch\_fields()**

MYSQL\_FIELD \*mysql\_fetch\_fields(MYSQL\_RES \*result)

**20.4.15.1 说明**

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

**20.4.15.2 返回值**

一个结果集合的所有MYSQL\_FIELD结构的一个数组。

**20.4.15.3 错误**

无。

**20.4.15.4 范例**

unsigned int num\_fields;

unsigned int i;

MYSQL\_FIELD \*fields;

num\_fields = mysql\_num\_fields(result);

fields = mysql\_fetch\_fields(result);

for(i = 0; i < num\_fields; i++)

{

printf("Field %u is %s\n", i, fields[i].name);

}

**20.4.16 mysql\_fetch\_field\_direct()**

MYSQL\_FIELD \*mysql\_fetch\_field\_direct(MYSQL\_RES \*result, unsigned int fieldnr)

**20.4.16.1 说明**

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

**20.4.16.2 返回值**

指定列的MYSQL\_FIELD结构。

**20.4.16.3 错误**

无。

**20.4.16.4 范例**

unsigned int num\_fields;

unsigned int i;

MYSQL\_FIELD \*field;

num\_fields = mysql\_num\_fields(result);

for(i = 0; i < num\_fields; i++)

{

field = mysql\_fetch\_field\_direct(result, i);

printf("Field %u is %s\n", i, field->name);

}

**20.4.17 mysql\_fetch\_lengths()**

unsigned long \*mysql\_fetch\_lengths(MYSQL\_RES \*result)

**20.4.17.1 说明**

返回在结果集合内的当前行的列长度。如果你计划拷贝字段值，这个长度信息对优化也是有用的，因为你可以避免调用strlen()。另外，如果结果集合中包含二进制数据，你*必须*使用这个函数确定数据的大小，因为strlen()对包含空字符的任何字段返回不正确的结果。

空列和包含NULL的列的长度值是零。为了看清如何区分这两种情况，见mysql\_fetch\_row()的说明。

**20.4.17.2 返回值**

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

**20.4.17.3 错误**

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

**20.4.17.4 范例**

MYSQL\_ROW row;

unsigned long \*lengths;

unsigned int num\_fields;

unsigned int i;

row = mysql\_fetch\_row(result);

if (row)

{

num\_fields = mysql\_num\_fields(result);

lengths = mysql\_fetch\_lengths(result);

for(i = 0; i < num\_fields; i++)

{

printf("Column %u is %lu bytes in length.\n", i, lengths[i]);

}

}

**20.4.18 mysql\_fetch\_row()**

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

**20.4.18.1 说明**

检索一个结果集合的下一行。当在mysql\_store\_result()之后使用时，如果没有更多的行可见所时，mysql\_fetch\_row()返回NULL。当在mysql\_use\_result()之后使用时，当没有更多的行可检索时或如果出现一个错误，mysql\_fetch\_row()返回NULL。

在行中值的数量由mysql\_num\_fields(result)给出。如果row保存了从一个对用mysql\_fetch\_row()调用返回的值，指向该值的指针作为row[0]到row[mysql\_num\_fields(result)-1]来存取。在行中的NULL值由NULL指针指出。

在行中字段值的长度可以通过调用mysql\_fetch\_lengths()获得。空字段和包含NULL的字段长度都是 0；你可以通过检查该值的指针区分他们。如果指针是NULL，字段是NULL；否则字段是空的。

**20.4.18.2 返回值**

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

**20.4.18.3 错误**

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.18.4 范例**

MYSQL\_ROW row;

unsigned int num\_fields;

unsigned int i;

num\_fields = mysql\_num\_fields(result);

while ((row = mysql\_fetch\_row(result)))

{

unsigned long \*lengths;

lengths = mysql\_fetch\_lengths(result);

for(i = 0; i < num\_fields; i++)

{

printf("[%.\*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");

}

printf("\n");

}

**20.4.19 mysql\_field\_count()**

unsigned int mysql\_field\_count(MYSQL \*mysql)

如果你正在使用一个比3.22.24早**MySQL**版本，你应该使用unsigned int mysql\_num\_fields(MYSQL \*mysql)。

**20.4.19.1 说明**

返回在连接上的最近查询的列的数量。

这个函数一般用在mysql\_store\_result()返回NULL时（这样你没有结果设置指针)。在这种情况中，你能调用mysql\_field\_count()确定mysql\_store\_result()是否应该产生了一个非空的结果。这允许一个客户程序执行正确的操作，而不必知道查询是否是一条SELECT（或类SELECT)语句。下面显示的例子说明这怎样可以做到。

见[20.4.51 为什么在mysql\_query()返回成功后，mysql\_store\_result()有时返回NULL?](http://www.yesky.com/imagesnew/software/mysql/manual_Clients.html#NULL_mysql_store_result)。

**20.4.19.2 返回值**

在结果集合中表示字段数量字的一个无符号整数。

**20.4.19.3 错误**

无。

**20.4.19.4 范例**

MYSQL\_RES \*result;

unsigned int num\_fields;

unsigned int num\_rows;

if (mysql\_query(&mysql,query\_string))

{

// error

}

else // query succeeded, process any data returned by it

{

result = mysql\_store\_result(&mysql);

if (result) // there are rows

{

num\_fields = mysql\_num\_fields(result);

// retrieve rows, then call mysql\_free\_result(result)

}

else // mysql\_store\_result() returned nothing; should it have?

{

if(mysql\_field\_count(&mysql) == 0)

{

// query does not return data

// (it was not a SELECT)

num\_rows = mysql\_affected\_rows(&mysql);

}

else // mysql\_store\_result() should have returned data

{

fprintf(stderr, "Error: %s\n", mysql\_error(&mysql));

}

}

}

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

**20.4.20 mysql\_field\_seek()**

MYSQL\_FIELD\_OFFSET mysql\_field\_seek(MYSQL\_RES \*result, MYSQL\_FIELD\_OFFSET offset)

**20.4.20.1 说明**

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

为了定位于行的起始，传递一个值为0的offset值。

**20.4.20.2 返回值**

字段光标的先前的值。

**20.4.20.3 错误**

无。

**20.4.21mysql\_field\_tell()**

MYSQL\_FIELD\_OFFSET mysql\_field\_tell(MYSQL\_RES \*result)

**20.4.21.1 说明**

返回用于最后一个mysql\_fetch\_field()的字段光标的位置。这个值可用作mysql\_field\_seek()的一个参数。

**20.4.21.2 返回值**

字段光标的当前偏移量。

**20.4.21.3 错误**

无。

**20.4.22 mysql\_free\_result()**

void mysql\_free\_result(MYSQL\_RES \*result)

**20.4.22.1 说明**

释放由mysql\_store\_result()、mysql\_use\_result()、mysql\_list\_dbs()等为一个结果集合分配的内存。当你用完了一个结果集合时，你必须调用mysql\_free\_result()来释放它使用的内存。

**20.4.22.2 返回值**

无。

**20.4.22.3 错误**

无。

**20.4.23 mysql\_get\_client\_info()**

char \*mysql\_get\_client\_info(void)

**20.4.23.1 说明**

返回代表客户库的版本的字符串。

**20.4.23.2 返回值**

代表**MySQL**客户库版本的一个字符串。

**20.4.23.3 错误**

无。

**20.4.24 mysql\_get\_host\_info()**

char \*mysql\_get\_host\_info(MYSQL \*mysql)

**20.4.24.1 说明**

返回描述正在使用的连接类型的字符串，包括服务其主机名。

**20.4.24.2 返回值**

表示服务器主机名者和连接类型的字符串。

**20.4.24.3 错误**

无。

**20.4.25 mysql\_get\_proto\_info()**

unsigned int mysql\_get\_proto\_info(MYSQL \*mysql)

**20.4.25.1 说明**

返回当前连接使用的协议版本。

**20.4.25.2 返回值**

表示被当前连接使用的协议版本的一个无符号整数。

**20.4.25.3 错误**

无。

**20.4.26 mysql\_get\_server\_info()**

char \*mysql\_get\_server\_info(MYSQL \*mysql)

**20.4.26.1 说明**

返回表示服务器版本号的字符串。

**20.4.26.2 返回值**

表示服务器版本号的一个字符串。

**20.4.26.3 错误**

无。

**20.4.27 mysql\_info()**

char \*mysql\_info(MYSQL \*mysql)

**20.4.27.1 说明**

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

INSERT INTO ... SELECT ...

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

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

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

LOAD DATA INFILE ...

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

ALTER TABLE

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

UPDATE

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

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

**20.4.27.2 返回值**

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

**20.4.27.3 错误**

无。

**20.4.28 mysql\_init()**

MYSQL \*mysql\_init(MYSQL \*mysql)

**20.4.28.1 说明**

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

**20.4.28.2 返回值**

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

**20.4.28.3 错误**

在内存不够的情况下，返回NULL。

**20.4.29 mysql\_insert\_id()**

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

**20.4.29.1 说明**

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

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

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

**20.4.29.2 返回值**

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

**20.4.29.3 错误**

无。

**20.4.30 mysql\_kill()**

int mysql\_kill(MYSQL \*mysql, unsigned long pid)

**20.4.30.1 说明**

要求服务器杀死由pid指定的线程。

**20.4.30.2 返回值**

成功，零。如果出现一个错误，非零。

**20.4.30.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.31 mysql\_list\_dbs()**

MYSQL\_RES \*mysql\_list\_dbs(MYSQL \*mysql, const char \*wild)

**20.4.31.1 说明**

返回一个结果集合，它用在服务器上的匹配wild参数指定的简单正则表达式的数据库名组成。wild可以包含通配符字符“%”或“\_”，或可以是匹配所有的数据库的一个NULL指针。调用mysql\_list\_dbs()类似于执行查询SHOW databases [LIKE wild]。

你必须用mysql\_free\_result()释放结果集合。

**20.4.31.2 返回值**

成功，一个MYSQL\_RES结果集合。如果出现一个错误，NULL。

**20.4.31.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_OUT\_OF\_MEMORY

内存溢出。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.32 mysql\_list\_fields()**

MYSQL\_RES \*mysql\_list\_fields(MYSQL \*mysql, const char \*table, const char \*wild)

**20.4.32.1 说明**

返回一个结果集合，它用在给定表中的匹配wild参数指定的简单正则表达式的列名组成。wild可以包含通配符字符“%”或“\_”，或可以是匹配所有列的一个NULL指针。调用mysql\_list\_fields()类似于执行查询SHOW COLUMNS FROM tbl\_name [LIKE wild]。

注意，建议你使用SHOW COLUMNS FROM tbl\_name而不是mysql\_list\_fields()。

你必须用mysql\_free\_result()释放结果集合。

**20.4.32.2 返回值**

成功，一个MYSQL\_RES的结果集合。如果出线一个错误，NULL。

**20.4.32.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务者关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.33 mysql\_list\_processes()**

MYSQL\_RES \*mysql\_list\_processes(MYSQL \*mysql)

**20.4.33.1 说明**

返回一个描述当前服务器线程的结果集合。这是与mysqladmin processlist或SHOW PROCESSLIST查询报告的相同信息。

你必须用mysql\_free\_result()释放结果集合。

**20.4.33.2 返回值**

成功，一个MYSQL\_RES结果集合。如果发生一个错误，NULL。

**20.4.33.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务者关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.34 mysql\_list\_tables()**

MYSQL\_RES \*mysql\_list\_tables(MYSQL \*mysql, const char \*wild)

**20.4.34.1 说明**

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

你必须用mysql\_free\_result()释放结果集合。

**20.4.34.2 返回值**

成功，一个MYSQL\_RES结果集合。如果出现一个错误，NULL。

**20.4.34.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不正确的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.35 mysql\_num\_fields()**

unsigned int mysql\_num\_fields(MYSQL\_RES \*result)

或

unsigned int mysql\_num\_fields(MYSQL \*mysql)

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

**20.4.35.1 说明**

在结果集合中返回列的数量。

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

见[20.4.51 为什么在mysql\_query()返回成功后，mysql\_store\_result()有时返回NULL?](http://www.yesky.com/imagesnew/software/mysql/manual_Clients.html#NULL_mysql_store_result)。

**20.4.35.2 返回值**

表示一个结果集合中字段数量的一个无符号整数。

**20.4.35.3 错误**

无。

**20.4.35.4 范例**

MYSQL\_RES \*result;

unsigned int num\_fields;

unsigned int num\_rows;

if (mysql\_query(&mysql,query\_string))

{

// error

}

else // query succeeded, process any data returned by it

{

result = mysql\_store\_result(&mysql);

if (result) // there are rows

{

num\_fields = mysql\_num\_fields(result);

// retrieve rows, then call mysql\_free\_result(result)

}

else // mysql\_store\_result() returned nothing; should it have?

{

if (mysql\_errno(&mysql))

{

fprintf(stderr, "Error: %s\n", mysql\_error(&mysql));

}

else if (mysql\_field\_count(&mysql) == 0)

{

// query does not return data

// (it was not a SELECT)

num\_rows = mysql\_affected\_rows(&mysql);

}

}

}

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

**20.4.36 mysql\_num\_rows()**

my\_ulonglong mysql\_num\_rows(MYSQL\_RES \*result)

**20.4.36.1 说明**

在结果集合中返回行的数量。

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

**20.4.36.2 返回值**

在结果集合中行的数量。

**20.4.36.3 错误**

无。

**20.4.37 mysql\_options()**

int mysql\_options(MYSQL \*mysql, enum mysql\_option option, const char \*arg)

**20.4.37.1 说明**

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

mysql\_options()应该在mysql\_init()之后和mysql\_connect()或mysql\_real\_connect()之前调用。

option参数是你想要设置的选项；arg参数是选项的值。如果选项是一个整数，那么arg应该指向整数值。

可能的选项值：

|  |  |  |
| --- | --- | --- |
| **选项** | **参数类型** | **功能** |
| MYSQL\_OPT\_CONNECT\_TIMEOUT | unsigned int \* | 以秒计的连接超时。 |
| MYSQL\_OPT\_COMPRESS | 不使用 | 使用压缩的客户机/服务器协议。 |
| MYSQL\_OPT\_NAMED\_PIPE | 不使用 | 使用命名管道连接一个在NT上的**MySQL**服务器。 |
| MYSQL\_INIT\_COMMAND | char \* | 当连接**MySQL**服务器时执行的命令。当重新连接时，将自动重新执行。 |
| MYSQL\_READ\_DEFAULT\_FILE | char \* | 从命名的选项文件而不是从“my.cnf”读取选项。 |
| MYSQL\_READ\_DEFAULT\_GROUP | char \* | 从“my.cnf”或用MYSQL\_READ\_DEFAULT\_FILE指定的文件中的命名组中读取选项。 |

注意，如果你使用MYSQL\_READ\_DEFAULT\_FILE或MYSQL\_READ\_DEFAULT\_GROUP，总是读取client。

在选项文件中指定的组可能包含下列选项：

|  |  |
| --- | --- |
| compress | 使用压缩的客户机/服务器协议。 |
| database | 如果在连接命令中没有指定数据库，使用这个数据库。 |
| debug | 调试选项 |
| host | 缺省主机名 |
| init-command | 在连接**MySQL**服务器时，执行的命令。当重新连接时，将自动重新执行。 |
| password | 缺省口令 |
| pipe | 使用命名管道连接一个在NT上的**MySQL**服务器。 |
| port | 缺省端口号 |
| return-found-rows | 告诉mysql\_info()返回找到的行，而不是在使用UPDATE时，返回更新的行。 |
| socket | 缺省套接字号 |
| timeout | 以秒计的连接超时。 |
| user | 缺省用户 |

对于选项文件的更多信息，见[4.15.4 选项文件](http://www.yesky.com/imagesnew/software/mysql/manual_Installing.html#Option_files)。

**20.4.37.2 返回值**

成功，零。如果你使用了未知的选项，非零。

**20.4.37.3 范例**

MYSQL mysql;

mysql\_init(&mysql);

mysql\_options(&mysql,MYSQL\_OPT\_COMPRESS,0);

mysql\_options(&mysql,MYSQL\_READ\_DEFAULT\_GROUP,"odbc");

if (!mysql\_real\_connect(&mysql,"host","user","passwd","database",0,NULL,0))

{

fprintf(stderr, "Failed to connect to database: Error: %s\n",

mysql\_error(&mysql));

}

上例请求客户使用压缩的客户机/服务器协议并且从my.cnf文件的odbc小节读取额外的选项。

**20.4.38 mysql\_ping()**

int mysql\_ping(MYSQL \*mysql)

**20.4.38.1 说明**

检查到服务器的连接是否正在工作。如果它关闭了，自动尝试一个再连接。

这个函数可被已经空闲很长时间的客户使用，来检查服务器是否关闭了连接并且如有必要重新连接。

**20.4.38.2 返回值**

如果服务器活着，零。如果出现一个错误，非零。

**20.4.38.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.39 mysql\_query()**

int mysql\_query(MYSQL \*mysql, const char \*query)

**20.4.39.1 说明**

执行指向空终止的字符串query的SQL查询，查询必须由一个单个的SQL语句组成。你不应该在语句后加上一个终止的分号(“；”)或\g。

mysql\_query()不能被用于包含二进制数据的查询；相反你应该使用mysql\_real\_query()。（二进制数据可能包含“\0”字符，而mysql\_query()将解释为查询字符串的结束。）

**20.4.39.2 返回值**

如果查询成功，零。如果出现一个错误，非零。

**20.4.39.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

到服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.40 mysql\_real\_connect()**

MYSQL \*mysql\_real\_connect(MYSQL \*mysql, const char \*host, const char \*user, const char \*passwd, const char \*db, unsigned int port, const char \*unix\_socket, unsigned int client\_flag)

**20.4.40.1 说明**

mysql\_real\_connect()试图建立到运行host的一个**MySQL**数据库引擎的一个连接。 mysql\_real\_connect()在你可以执行任何其他API函数之前必须成功地完成，除了mysql\_get\_client\_info()。

参数指定如下：

* 第一个参数应该是一个现存MYSQL结构的地址。在调用mysql\_real\_connect()之前，你必须调用mysql\_init()初始化MYSQL结构。见下面的例子。
* host值可以是一个主机名或一个IP地址。如果host是NULL或字符串"localhost"，假定是到本地主机的一个连接。如果OS支持套接字(Unix)或命名管道(Win32)，使用他们而不是TCP/IP与服务器连接。
* user参数包含用户的**MySQL**登录ID。如果user是NULL，假定是当前用户。在Unix下，它是当前登录名。在Windows ODBC下，必须明确地指定当前用户名字。见[16.4 怎样填写ODBC管理程序中各种域](http://www.yesky.com/imagesnew/software/mysql/manual_ODBC.html#ODBC_administrator)。
* passwd参数为user包含口令。如果passwd是NULL，只有在user表中对于有一个空白口令字段的用户的条目将被检查一个匹配。这允许数据库主管设置**MySQL**权限，使用户获得不同的口令，取决于他们是否已经指定一个口令。注意：不要试图在调用mysql\_real\_connect()前加密口令；口令加密自动被客户API处理。
* db是数据库名。如果db不是NULL，连接将缺省数据库设置为这个值。
* 如果port不是0，值对于TCP/IP连接将用作端口号。注意host参数决定连接的类型。
* 如果unix\_socket不是NULL，字符串指定套接字或应该被使用的命名管道。注意host参数决定连接的类型。
* client\_flag值通常是0，但是在很特殊的情况下可以被设置为下列标志的组合：

|  |  |
| --- | --- |
| **标志名字** | **意味着的标志** |
| CLIENT\_FOUND\_ROWS | 返回找到的(匹配的)行数，不是受到影响的行数。 |
| CLIENT\_NO\_SCHEMA | 不允许db\_name.tbl\_name.col\_name语法。这是为了ODBC；如果你使用该语法，导致语法分析器产生一个错误，它是为在一些ODBC程序捕捉错误是有用的。 |
| CLIENT\_COMPRESS | 使用压缩协议。 |
| CLIENT\_ODBC | 客户是一个ODBC客户。这使mysqld变得对ODBC更友好。 |

**20.4.40.2 返回值**

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

**20.4.40.3 错误**

CR\_CONN\_HOST\_ERROR

不能连接**MySQL**服务器。

CR\_CONNECTION\_ERROR

不能连接本地**MySQL**服务器。

CR\_IPSOCK\_ERROR

不能创建一个IP套接字。

CR\_OUT\_OF\_MEMORY

内存溢出。

CR\_SOCKET\_CREATE\_ERROR

不能创建一个Unix套接字。

CR\_UNKNOWN\_HOST

不能找到主机名的IP地址。

CR\_VERSION\_ERROR

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

CR\_NAMEDPIPEOPEN\_ERROR;

不能在 Win32 上创建一个命名管道。

CR\_NAMEDPIPEWAIT\_ERROR;

不能在 Win32 上等待一个命名管道。

CR\_NAMEDPIPESETSTATE\_ERROR;

不能在 Win32 上得到一个管道处理器。

**20.4.40.4 范例**

MYSQL mysql;

mysql\_init(&mysql);

if (!mysql\_real\_connect(&mysql,"host","user","passwd","database",0,NULL,0))

{

fprintf(stderr, "Failed to connect to database: Error: %s\n",

mysql\_error(&mysql));

}

**20.4.41 mysql\_real\_query()**

int mysql\_real\_query(MYSQL \*mysql, const char \*query, unsigned int length)

**20.4.41.1 说明**

执行由query指向的SQL查询，它应该是一个length个字节的字符串。查询必须由一个单个的SQL语句组成。你不应该在语句后增加一个终止的分号(“;”)或\g。

对于包含二进制数据的查询，你*必须*使用mysql\_real\_query()而不是mysql\_query()，因为二进制代码数据可能包含“\0”字符，而且，mysql\_real\_query()比mysql\_query()更快，因为它对查询字符串调用strlen()。

**20.4.41.2 返回值**

如果查询成功，零。如果发生一个错误，非零。

**20.4.41.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.42 mysql\_reload()**

int mysql\_reload(MYSQL \*mysql)

**20.4.42.1 说明**

要求**MySQL**服务器再次装载授权表。连接的用户必须拥有**reload**权限。

不推荐这个函数。最好使用mysql\_query()发出一条SQL FLUSH PRIVILEGES语句。

**20.4.42.2 返回值**

成功，零。如果发生一个错误，非零。

**20.4.42.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.43 mysql\_row\_seek()**

MYSQL\_ROW\_OFFSET mysql\_row\_seek(MYSQL\_RES \*result, MYSQL\_ROW\_OFFSET offset)

**20.4.43.1 说明**

设置行光标为在结果集合中的任意行。这要求结果集合结构包含查询的全部结果，这样mysql\_row\_seek()只能与mysql\_store\_result()一起使用，而不与mysql\_use\_result()。

偏移量应该是调用mysql\_row\_tell()或mysql\_row\_seek()返回的值。这个值不是简单地一个行号；如果你想要在结果集合内用行号来寻找行，使用mysql\_data\_seek()。

**20.4.43.2 返回值**

行光标先前的值。该值可以被传递给随后的mysql\_row\_seek()调用。

**20.4.43.3 错误**

无。

**20.4.44 mysql\_row\_tell()**

MYSQL\_ROW\_OFFSET mysql\_row\_tell(MYSQL\_RES \*result)

**20.4.44.1 说明**

返回为了mysql\_fetch\_row()的行光标的当前位置。这个值可以作为一个参数用于mysql\_row\_seek()。

你应该仅在mysql\_store\_result()后使用mysql\_row\_tell()，而不是在mysql\_use\_result()后。

**20.4.44.2 返回值**

行光标当前的偏移量。

**20.4.44.3 错误**

无。

**20.4.45 mysql\_select\_db()**

int mysql\_select\_db(MYSQL \*mysql, const char \*db)

**20.4.45.1 说明**

使得由db指定的数据库成为 在由mysql指定的连接上的缺省(当前)数据库。在随后的查询中，这个数据库对于不包括一个显式的数据库指定符的表的引用是缺省数据库。

除非连接的用户能被认证允许使用数据库，否则mysql\_select\_db()失败。

**20.4.45.2 返回值**

成功，零。如果发生一个错误，非零。

**20.4.45.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.46 mysql\_shutdown()**

int mysql\_shutdown(MYSQL \*mysql)

**20.4.46.1 说明**

让数据库服务器关闭。连接的用户必须有**shutdown**权限。

**20.4.46.2 返回值**

成功，零。如果出现一个错误，非零

**20.4.46.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.47 mysql\_stat()**

char \*mysql\_stat(MYSQL \*mysql)

**20.4.47.1 说明**

返回包含类似于由mysqladmin status命令提供的信息的一个字符串。它包括正常运行的秒数和正在运行线程、问题、再次装载和打开的表的数目。

**20.4.47.2 返回值**

描述服务器状态的一个字符串。如果出现一个错误，NULL。

**20.4.47.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.48 mysql\_store\_result()**

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

**20.4.48.1 说明**

对于成功地检索数据的每个询问(SELECT、SHOW、DESCRIBE、EXPLAIN)，你必须调用mysql\_store\_result()或mysql\_use\_result()。

mysql\_store\_result()读取一个到客户的查询的全部结果，分配一个MYSQL\_RES结构，并且把结果放进这个结构中。

如果没有行返回，返回一个空集合集合。（空结果集合不同于一个NULL返回值。）

一旦你调用了mysql\_store\_result()，你可以调用mysql\_num\_rows()找出结果集合中有多少行。

你能调用mysql\_fetch\_row()从结果集合中取出行，或mysql\_row\_seek()和mysql\_row\_tell()结果集合中获得或设置当前的行位置。

一旦你用完结果集合，你必须调用mysql\_free\_result()。

见[20.4.51 为什么mysql\_query()返回成功后，mysql\_store\_result()有时返回NULL?](http://www.yesky.com/imagesnew/software/mysql/manual_Clients.html#NULL_mysql_store_result)。

**20.4.48.2 返回值**

一个保存结果的MYSQL\_RES结构。如果出现一个错误，NULL。

**20.4.48.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_OUT\_OF\_MEMORY

内存溢出。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.49 mysql\_thread\_id()**

unsigned long mysql\_thread\_id(MYSQL \*mysql)

**20.4.49.1 说明**

返回当前连接的线程ID。这个值可用作mysql\_kill()的一个参数以杀死线程。

如果失去连接并且你用mysql\_ping()重新连接，线程ID将改变。这意味着你不应该为以后使用获得线程ID并且存储它，当你需要它时，你应该获得它。

**20.4.49.2 返回值**

当前连接的线程 ID 。

**20.4.49.3 错误**

无。

**20.4.50 mysql\_use\_result()**

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

**20.4.50.1 说明**

对于成功地检索数据的每个查询(SELECT、SHOW、DESCRIBE、EXPLAIN)，你必须调用mysql\_store\_result()或mysql\_use\_result()。

mysql\_use\_result()初始化一个结果集合的检索，但不真正将结果集合读入客户，就象mysql\_store\_result()那样。相反，必须通过调用mysql\_fetch\_row()单独检索出每一行，这直接从服务器读出结果而不在一个临时表或本地缓冲区中存储它，它比mysql\_store\_result()更快一点并且使用较少的内存。客户将只为当前行和一个可能最大max\_allowed\_packet字节的通信缓冲区分配内存。

在另一方面，如果你在客户端对每一行正在做很多的处理，或如果输出被送到屏幕，用户可以打一个^S（停止滚动)，你不应该使用mysql\_use\_result()。这将阻塞服务器并且阻止另外的线程从数据被取出的任何表中更新数据。

当使用mysql\_use\_result()时，你必须执行mysql\_fetch\_row()直到返回一个NULL值，否则未取出的行将作为下一个查询的结果集合一部分被返回。如果你忘记做这个，C API将给出错误Commands out of sync; You can't run this command now！

你不能在一个从mysql\_use\_result()返回的结果集合上使用mysql\_data\_seek()、mysql\_row\_seek()、mysql\_row\_tell()、mysql\_num\_rows()或mysql\_affected\_rows()，你也不能发出另外的查询直到mysql\_use\_result()完成。（然而，在你取出所有的行以后，mysql\_num\_rows()将精确地返回取出的行数。）

一旦你用完结果集合，你必须调用mysql\_free\_result()。

**20.4.50.2 返回值**

一个MYSQL\_RES结果结构。 如果发生一个错误发生，NULL。

**20.4.50.3 错误**

CR\_COMMANDS\_OUT\_OF\_SYNC

命令以一个不适当的次序被执行。

CR\_OUT\_OF\_MEMORY

内存溢出。

CR\_SERVER\_GONE\_ERROR

**MySQL**服务器关闭了。

CR\_SERVER\_LOST

对服务器的连接在查询期间失去。

CR\_UNKNOWN\_ERROR

发生一个未知的错误。

**20.4.51 为什么在mysql\_query()返回成功后，mysql\_store\_result()有时返回NULL?**

有可能在一个对mysql\_query()成功的调用后，mysql\_store\_result()返回NULL。当这发生时，它意味着出现了下列条件之一：

* 有一个malloc()失败(例如，如果结果集合太大)。
* 数据不能被读取(发生在连接上的一个错误)。
* 查询没有返回数据(例如，它是一个INSERT、UPDATE或DELETE)。

你总是可以通过调用mysql\_field\_count()检查语句是否应该产生非空的结果。如果mysql\_field\_count()返回零，结果是空的并且最后一个查询是不回值的一条语句(例如，一条INSERT或DELETE)。如果mysql\_field\_count()返回非零值，语句应该产生非空的结果。见对mysql\_field\_count()描述的一个例子。

你可以调用mysql\_error()或mysql\_errno()测试一个错误。

**20.4.52 我能从查询中得到什么结果？**

除了由查询返回的结果集合外，你也能得到下列信息：

* 当执行一条INSERT、UPDATE或DELETE时，mysql\_affected\_rows()返回受到最后一个查询影响的行数。一个例外是如果使用一条没有WHERE子句的DELETE，表被截断，它更快！在这种情况下，mysql\_affected\_rows()对于影响的记录数量返回零。
* mysql\_num\_rows()返回结果集合中的行数。用mysql\_store\_result()，一旦mysql\_store\_result()返回，就可以调用mysql\_num\_rows()。用mysql\_use\_result()，只有在你已经用mysql\_fetch\_row()取出了所有行后，才能调用mysql\_num\_rows()。
* mysql\_insert\_id()返回由将一行插入一个具有AUTO\_INCREMENT索引的表中的最后查询生成的ID。见[20.4.29 mysql\_insert\_id()](http://www.yesky.com/imagesnew/software/mysql/manual_Clients.html#mysql_insert_id)。
* 某些查询(LOAD DATA INFILE ...、INSERT INTO ... SELECT ...、UPDATE)返回附加的信息。结果由返mysql\_info()返回。对其返回字符串的格式，见mysql\_info()的描述。如果没有附加的信息，mysql\_info()返回一个NULL指针。

**20.4.53 我怎样能得到最后插入的行的唯一ID？**

如果你往包含一个具有AUTO\_INCREMENT属性的列的一张表中插入一个记录，你能通过mysql\_insert\_id()函数获得最近生成的ID。

你也可以通过在你传递给mysql\_query()的一个查询字符串中使用LAST\_INSERT\_ID()函数检索出ID。

你可以执行下列代码检查是否使用一个AUTO\_INCREMENT索引。这也检查查询是否是有一个AUTO\_INCREMENT索引的一条INSERT：

if (mysql\_error(&mysql)[0] == 0 &&

mysql\_num\_fields(result) == 0 &&

mysql\_insert\_id(&mysql) != 0)

{

used\_id = mysql\_insert\_id(&mysql);

}

最近产生的ID是在一个按连接的基础上在服务器上进行维护，它将不被其他客户改变。如果你更新另外一个有非奇特(non-magic)值(即一个既不是NULL也不是0的值)的AUTO\_INCREMENT列，它甚至将不被改变。

如果你想要使用为一张表生成的ID并且把它插入到第2张表，你可以使用象这样的SQL语句：

INSERT INTO foo (auto,text)

VALUES(NULL,'text'); # generate ID by inserting NULL

INSERT INTO foo2 (id,text)

VALUES(LAST\_INSERT\_ID(),'text'); # use ID in second table

**20.4.54 链接C API的问题**

当与C API链接时，下列错误可能发生一些系统上：

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

Undefined first referenced

symbol in file

floor /usr/local/lib/mysql/libmysqlclient.a(password.o)

ld: fatal: Symbol referencing errors. No output written to client

如果它发生在你的系统上，你必须通过在编译/链接命令行的最后增加-lm以包括数学库。

**20.4.55 怎样制作一个线程安全的客户**

客户“几乎”是线程安全的。最大的问题是在从套接字读取的“net.c”中的子程序不是中断安全的(interruot-safe)。这样做是这样考虑的，即你可能想有你自己的报警来中断一个长时间的读取服务器。

标准客户库没有用线程选项来编译。

为了获得一个线程安全的客户，使用-lmysys, -lstring和-ldbug库和服务器使用的net\_serv.o。

当使用一个线程化的客户时，你可以充分利用在“thr\_alarm.c”文件中的函数。如果你正在使用来自mysys库的函数，你唯一必须记住的是首先调用my\_init()！

所有函数除了mysql\_real\_connect()目前是线程安全的。下列注意事项描述怎样编译一个线程安全的客户库并且以一种线程安全的方式使用它。（下面对mysql\_real\_connect()的注意事项实际上也适用于mysql\_connect()，但是因为mysql\_connect()不提倡使用，无论如何你应该使用mysql\_real\_connect()。）

为了使mysql\_real\_connect()是线程安全的，你必须用这个命令重新编译客户库：

shell> CPPFLAGS=-DTHREAD\_SAFE\_CLIENT ./configure ...

当链接标准客户时，你可能得到的某些因为未定义符号的错误，因为pthread库没有被缺省地包括。

最终的“libmysqlclient.a”库现在是线程安全的。它的含义是只要2个线程不同时查询mysql\_real\_connect()返回的同一个连接句柄，客户代码是线程安全的；客户机/服务器协议在一个给定的连接上一次只允许一个请求。如果你想在同一个的连接上使用多个线程，你必须在mysql\_query()和mysql\_store\_result()调用组合附近有一个mutex锁定。一旦mysql\_store\_result()就绪，锁可以被释放并且其他线程可以查询同一个连接。（换句话说，不同的线程能使用不同被mysql\_store\_result()创建的MYSQL\_RES指针，只要他们使用适当的锁定协议) 如果你用POSIX线程编程，你能使用pthread\_mutex\_lock()和pthread\_mutex\_unlock()建立并且释放一个mutex锁定。

如果你使用mysql\_use\_result()而不是mysql\_store\_result()，锁定将需要包围mysql\_use\_result()和mysql\_fetch\_row()的调用，然而，它确实对不使用mysql\_use\_result()线程客户是最好的。

**20.5 MySQL Perl API**

本节记载了Perl DBI接口。以前的接口被称为mysqlperl。因为DBI/DBD现在是推荐的Perl接口，mysqlperl是过时的并且不在这里记载。

**20.5.1 DBI与DBD::mysql**

DBI是对于很多数据库的一个通用接口。这意味着你能编写一个脚本，不用改变就能工作于很多数据库引擎。你需要为每种数据库类型定义了的一个数据库驱动程序(DBD)。对于**MySQL**，该驱动程序称为DBD::mysql。

有关Perl5 DBI的更多信息，请访问DBI网页并且阅读文档：

<http://www.symbolstone.org/technology/perl/DBI/index.html>

关于在Perl5定义的面向对象的编程(OOP)的更多信息，参见Perl OOP页：

<http://language.perl.com/info/documentation.html>

对于**MySQL** Perl支持的安装指令在[4.10 Perl安装说明](http://www.yesky.com/imagesnew/software/mysql/manual_Installing.html#Perl_support)中给出。

**20.5.2 DBI接口**

**可移植的DBI方法**

|  |  |
| --- | --- |
| connect | 建立到一个数据库服务器的连接 |
| disconnect | 断开数据库服务器的连接 |
| prepare | 准备执行一个SQL语句 |
| execute | 执行准备好的语句 |
| do | 准备并执行一个SQL语句 |
| quote | 加引号于要插入的字符串或BLOB值 |
| fetchrow\_array | 作为一个字段数组取出下一行 |
| fetchrow\_arrayref | 作为一个字段的引用数组取出下一行 |
| fetchrow\_hashref | 作为一个哈希表的引用取出下一行 |
| fetchall\_arrayref | 作为一个字段数组取出所有数据 |
| finish | 完成一条语句并且让系统释放资源 |
| rows | 返回受影响的行数 |
| data\_sources | 返回可在localhost上得到的数据库的数组 |
| ChopBlanks | 控制fetchrow\_\*方法是否剥去空格 |
| NUM\_OF\_PARAMS | 在准备的语句中的占位（placeholder-参数）的数目 |
| NULLABLE | 其列可以是NULL |
| trace | 执行调试跟踪 |

**MySQL特定的方法**

|  |  |
| --- | --- |
| insertid | 最后一个AUTO\_INCREMENT值 |
| is\_blob | 该列是BLOB值 |
| is\_key | 该列是键 |
| is\_num | 该列是数字的 |
| is\_pri\_key | 该列是主键 |
| is\_not\_null | 该列不能是NULL，见NULLABLE。 |
| length | 最大可能的列大小 |
| max\_length | 结果中实际上存在的最大列大小 |
| NAME | 列名字 |
| NUM\_OF\_FIELDS | 返回的字段数目 |
| table | 在返回的集合中的表名 |
| type | 所有的列类型 |

Perl方法在下一节更详细地描述。用于方法返回值的变量有这些含义：

$dbh

数据库句柄

$sth

语句句柄

$rc

返回代码 (经常是一个状态）

$rv

返回值 (经常是一个行数）

**可移植DBI方法**

connect($data\_source, $username, $password)

使用connect方法使得一个数据库连接到数据源。$data\_source值应该以DBI:driver\_name:开始。以DBD::mysql驱动程序使用connect的例子：

$dbh = DBI->connect("DBI:mysql:$database", $user, $password);

$dbh = DBI->connect("DBI:mysql:$database:$hostname",

$user, $password);

$dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",

$user, $password);

如果用户名或口令未定义，DBI分别使用DBI\_USER和DBI\_PASS环境变量的值。如果你不指定主机名，它缺省为'localhost'。如果你不指定一个端口号，它缺省为**MySQL**的缺省端口(3306)。对Msql-Mysql-modules版本1.2009，$data\_source值允许某些修饰词：

mysql\_read\_default\_file=file\_name

读取作为一个选项文件的“filename”。有关选项文件的信息，见[4.15.4 选项文件](http://www.yesky.com/imagesnew/software/mysql/manual_Installing.html#Option_files)。

mysql\_read\_default\_group=group\_name

当读取选项文件时的缺省组通常是[client]组。通过指定mysql\_read\_default\_group选项，缺省组变成[group\_name]组。

mysql\_compression=1

在客户和服务器之间使用压缩通信(**MySQL** 3.22.3或以后)。

mysql\_socket=/path/to/socket

指定用于与服务器连接的Unix套接字的路径名(**MySQL** 3.21.15或以后)。

可以给出多个修饰词；每一个必须前置一个分号。例如，如果你想要避免在一个DBI脚本中硬编码用户名和口令，你可以从用户的“~/.my.cnf”选项文件中取出它们，而不是这样编写你的connect调用：

$dbh = DBI->connect("DBI:mysql:$database"

. ";mysql\_read\_default\_file=$ENV{HOME}/.my.cnf",

$user, $password);

这个调用将读取在选项文件中为[client]组而定义的选项。如果你想做同样的事情，但是也使用未[perl]组指定的选项，你可以使用：

$dbh = DBI->connect("DBI:mysql:$database"

. ";mysql\_read\_default\_file=$ENV{HOME}/.my.cnf"

. ";mysql\_read\_default\_group=perl",

$user, $password);

disconnect

disconnect方法从数据库断开数据库句柄。它一般就在你从程序退出之前被调用。范例：

$rc = $dbh->disconnect;

prepare($statement)

准备一条由数据库引擎执行的SQL语句并且返回语句句柄($sth)，你可以使用它调用execute方法。一般地你借助于prepare和execute来处理SELECT语句(和类SELECT语句，例如SHOW、DESCRIBE和EXPLAIN)。范例：

$sth = $dbh->prepare($statement)

or die "Can't prepare $statement: $dbh->errstr\n";

execute

execute方法执行一个准备好的语句。对非SELECT语句，execute返回受影响的行数。如果没有行受影响，execute返回"0E0"，Perl将它视作零而不是真。对于SELECT语句，execute只是在数据库中启动SQL查询；你需要使用在下面描述的fetch\_\*方法之一检索数据。范例：

$rv = $sth->execute

or die "can't execute the query: $sth->errstr;

do($statement)

do方法准备并且执行一条SQL语句并且返回受影响的行数。如果没有行受到影响，do返回"0E0"，Perl将它视为零而不是真。这个方法通常用于事先无法准备好(由于驱动程序的限制)或不需要执行多次(插入、删除等等)的非SELECT语句。范例：

$rv = $dbh->do($statement)

or die "Can't execute $statement: $dbh- >errstr\n";

quote($string)

quote方法被用来“转义”包含在string中的任何特殊字符并增加所需的外部的引号。范例：

$sql = $dbh->quote($string)

fetchrow\_array

这个方法取下一行数据并且作为一个字段值数组返回它。范例：

while(@row = $sth->fetchrow\_array) {

print qw($row[0]\t$row[1]\t$row[2]\n);

}

fetchrow\_arrayref

这个方法取下一行数据并且作为一个对一个字段值数组的引用返回它。范例：

while($row\_ref = $sth->fetchrow\_arrayref) {

print qw($row\_ref->[0]\t$row\_ref->[1]\t$row\_ref->[2]\n);

}

fetchrow\_hashref

这个方法取一行数据并且返回包含字段名/值对的一个哈希表的一个引用。这个方法不如使用上述数组引用那样有效。范例：

while($hash\_ref = $sth->fetchrow\_hashref) {

print qw($hash\_ref->{firstname}\t$hash\_ref->{lastname}\t\

$hash\_ref- > title}\n);

}

fetchall\_arrayref

这个方法被用来获得从SQL语句被返回的所有数据(行)。它返回一个数组的引用，该数组包含对每行的数组的引用。你用一个嵌套循环来存取或打印数据。范例：

my $table = $sth->fetchall\_arrayref

or die "$sth->errstr\n";

my($i, $j);

for $i ( 0 .. $#{$table} ) {

for $j ( 0 .. $#{$table->[$i]} ) {

print "$table->[$i][$j]\t";

}

print "\n";

}

finish

便名没有更多的数据将从这个语句句柄取出。你调用这个方法释放语句句柄和任何与它相关的系统资源。范例：

$rc = $sth->finish;

rows

返回由最后一条命令改变(更新、删除等)的行数。这通常用在非SELECT的execute语句之后。范例：

$rv = $sth->rows;

NULLABLE

返回一个对一个布尔值数组的引用；对数组的每个成员，一个TRUE值表示该列可以包含NULL值。范例：

$null\_possible = $sth->{NULLABLE};

NUM\_OF\_FIELDS

这个属性表明由一条SELECT或SHOW FIELDS语句返回的字段数目。你可以用它检查一条语句是否返回了结果：一个零值表明一个象INSERT、DELETE或UPDATE的非SELECT语句。范例：

$nr\_of\_fields = $sth->{NUM\_OF\_FIELDS};

data\_sources($driver\_name)

这个方法返回一个数组，它包含在主机'localhost'上的**MySQL**服务器可得到的数据库名。范例：

@dbs = DBI->data\_sources("mysql");

ChopBlanks

这个属性确定fetchrow\_\*方法是否将去掉返回值的头和尾的空白。范例：

$sth->{'ChopBlanks'} =1;

trace($trace\_level)

trace($trace\_level, $trace\_filename)

trace方法开启或关闭跟踪。当作为一个DBI类方法调用时，它影响对所有句柄的跟踪。当作为一个数据库或语句句柄方法调用时，它影响对给定句柄的跟踪(和句柄的未来子孙)。设置$trace\_level为2以提供详细的踪迹信息，设置$trace\_level为0以关闭跟踪。踪迹输出缺省地输出到标准错误输出。如果指定$trace\_filename，文件以添加模式打开并且*所有*跟踪的句柄的手被写入该文件。范例：

DBI->trace(2); # trace everything

DBI->trace(2,"/tmp/dbi.out"); # trace everything to /tmp/dbi.out

$dth->trace(2); # trace this database handle

$sth->trace(2); # trace this statement handle

你也可以通过设置DBI\_TRACE环境变量开启DBI跟踪。将它设置为等价于调用DBI->(value)的数字值，将它设置为等价于调用DBI->(2,value)的路径名。

**MySQL特定的方法**

下面显示的方法是**MySQL**特定的并且不是DBI标准的部分。他们中有几个现在不建议使用：is\_blob、is\_key、is\_num、is\_pri\_key、is\_not\_null、length、max\_length和table。这些已有DBI标准的另一种方法存在，他们在下面说明。

insertid

如果你使用**MySQL**的AUTO\_INCREMENT功能，新的自动加1的值将存储在这里。范例：

$new\_id = $sth->{insertid};

作为另一种选择，你可以使用$dbh->{'mysql\_insertid'}。

is\_blob

返回一个对一个布尔值数组的引用；对数组的每个单元，一个TRUE值表明相应的列是一个BLOB。范例：

$keys = $sth->{is\_blob};

is\_key

返回一个对一个布尔值数组的引用；对数组的每个单元，一个TRUE值表明相应的列是键。范例：

$keys = $sth->{is\_key};

is\_num

返回一个对一个布尔值数组的引用；对数组的每个单元，一个TRUE值表明相应的列包含数字值。范例：

$keys = $sth->{is\_key};

is\_pri\_key

返回一个对一个布尔值数组的引用；对数组的每个单元，一个TRUE值表明相应的列是主键。范例：

pri\_keys = $sth->{is\_pri\_key};

is\_not\_null

返回一个对一个布尔值数组的引用；对数组的每个单元，一个FALSEE值表明该列可以包含NULL值。范例：

$not\_nulls = $sth->{is\_not\_null};

is\_not\_null不建议使用；最好使用NULLABLE属性(在上面描述了)，因为那是一个 DBI 标准。

length

max\_length

这些方法的每个返回一个对列大小数组的引用。length数组指出每列可以有的最大可能的尺寸 (如在表描述中声明的)。max\_length数组指出在结果表中实际存在的最大尺寸。范例：

$lengths = $sth->{length};

$max\_lengths = $sth->{max\_length};

NAME

返回一个对一个列名数组的引用。范例：

$names = $sth->{NAME};

table

返回一个对一个表名数组的引用。范例：

$tables = $sth->{table};

type

返回一个对一个列类型数组的引用。范例：

$types = $sth->{type};

**20.5.3 更多的DBI/DBD信息**

你可以使用perldoc命令得到更多的关于DBI的信息。

perldoc DBI

perldoc DBI::FAQ

perldoc DBD::mysql

你也可以使用pod2man、pod2html等工具转换到其他格式。

而且你当然可以在DBI网页中找到最新的DBI信息：

<http://www.symbolstone.org/technology/perl/DBI/index.html>

**20.6 MySQL Eiffel包装**

**MySQL** [Contrib目录](http://www.mysql.com/Contrib/)包含Michael Ravits编写的一个Eiffel包装程序。

你也能在这里找到：http://www.netpedia.net/hosting/newplayer/

**20.7 MySQL Java连接(JDBC)**

有2个为MySQL支持的JDBC驱动程序(twz和mm驱动程序)。你可以在<http://www.mysql.com/Contrib>找到这些的一个拷贝。对于文档，请教任何JDBC文档和驱动程序本身拥有的针对**MySQL**特定功能的文档。

**20.8 MySQL PHP API**

PHP是一个服务器端、HTML嵌入式脚本语言，可以用来创建动态网页。它包含对存取若干数据库的支持，包括**MySQL**。PHP可以作为一个单独的程序运行，或编译为与Apache服务器一起使用的一个模块。

分发和文档可在[PHP网站](http://www.php.net/)得到。

**20.9 MySQL C++ APIs**

在**MySQL** [Contrib 目录](http://www.mysql.com/Contrib/)中可得到两个API。

**20.10 MySQL Python API**

**MySQL** [Contrib 目录](http://www.mysql.com/Contrib/)包含Joseph Skinner编写的一个Python接口。

你也可以使用对iODBC的Python接口来存取一个**MySQL**服务器。[mxODBC](http://starship.skyport.net/~lemburg/)

**20.11 MySQL TCL API**

[TCL at binevolve](http://www.binevolve.com/~tdarugar/tcl-sql/)。[Contrib 目录](http://www.mysql.com/Contrib)包含一个基于msqltcl 1.50的一个TCL接口。