**MySQL 的 C API预处理语句**

2008-07-16 16:59:43|  分类： [默认分类](http://gisqimeng.blog.163.com/blog/#m=0&t=1&c=fks_095067087080084074093095074068085084085065087095094)|举报|字号 订阅





[下载LOFTER](http://www.lofter.com/app?act=qbbkrzydb_20150408_01)[我的照片书  |](http://yxp.163.com/)

**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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-date-handling)。

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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-prepared-statement-functions)。

|  |  |
| --- | --- |
| **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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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**

[25.2.7.1. mysql\_stmt\_affected\_rows()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-affected-rows)

[25.2.7.2. mysql\_stmt\_attr\_get()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-attr-get)

[25.2.7.3. mysql\_stmt\_attr\_set()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-attr-set)

[25.2.7.4. mysql\_stmt\_bind\_param()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-bind-param)

[25.2.7.5. mysql\_stmt\_bind\_result()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-bind-result)

[25.2.7.6. mysql\_stmt\_close()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-close)

[25.2.7.7. mysql\_stmt\_data\_seek()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-data-seek)

[25.2.7.8. mysql\_stmt\_errno()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-errno)

[25.2.7.9. mysql\_stmt\_error()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-error)

[25.2.7.10. mysql\_stmt\_execute()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-execute)

[25.2.7.11. mysql\_stmt\_fetch()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-fetch)

[25.2.7.12. mysql\_stmt\_fetch\_column()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-fetch-column)

[25.2.7.13. mysql\_stmt\_field\_count()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-field-count)

[25.2.7.14. mysql\_stmt\_free\_result()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-free-result)

[25.2.7.15. mysql\_stmt\_init()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-init)

[25.2.7.16. mysql\_stmt\_insert\_id()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-insert-id)

[25.2.7.17. mysql\_stmt\_num\_rows()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-num-rows)

[25.2.7.18. mysql\_stmt\_param\_count()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-param-count)

[25.2.7.19. mysql\_stmt\_param\_metadata()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-param-metadata)

[25.2.7.20. mysql\_stmt\_prepare()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-prepare)

[25.2.7.21. mysql\_stmt\_reset()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-reset)

[25.2.7.22. mysql\_stmt\_result\_metadata()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-result-metadata)

[25.2.7.23. mysql\_stmt\_row\_seek()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-row-seek)

[25.2.7.24. mysql\_stmt\_row\_tell()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-row-tell)

[25.2.7.25. mysql\_stmt\_send\_long\_data()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-send-long-data)

[25.2.7.26. mysql\_stmt\_sqlstate()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-sqlstate)

[25.2.7.27. mysql\_stmt\_store\_result()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-stmt-store-result)

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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-prepared-statement-datatypes)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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-prepared-statement-datatypes)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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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*](http://dev.mysql.com/doc/refman/5.1/zh/error-handling.html)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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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](http://dev.mysql.com/doc/refman/5.1/zh/error-handling.html)。

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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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()”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#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”](http://dev.mysql.com/doc/refman/5.1/zh/database-administration.html#query-cache-how)。

**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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-prepared-statement-datatypes)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**

[25.2.11.1. my\_init()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#my-init)

[25.2.11.2. mysql\_thread\_init()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-thread-init)

[25.2.11.3. mysql\_thread\_end()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-thread-end)

[25.2.11.4. mysql\_thread\_safe()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-thread-safe)

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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#threaded-clients)。

**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**

[25.2.12.1. mysql\_server\_init()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-server-init)

[25.2.12.2. mysql\_server\_end()](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#mysql-server-end)

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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#libmysqld)。

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”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-api-function-overview)。

**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”](http://dev.mysql.com/doc/refman/5.1/zh/using-mysql-programs.html#option-files)。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。