## 1交互过程

MySQL客户端与服务器的交互主要分为两个阶段：握手认证阶段和命令执行阶段。交互过程使用小端字节序进行网络传输。

### 1.1 握手认证阶段

握手认证阶段为客户端与服务器建立连接后进行，交互过程如下：

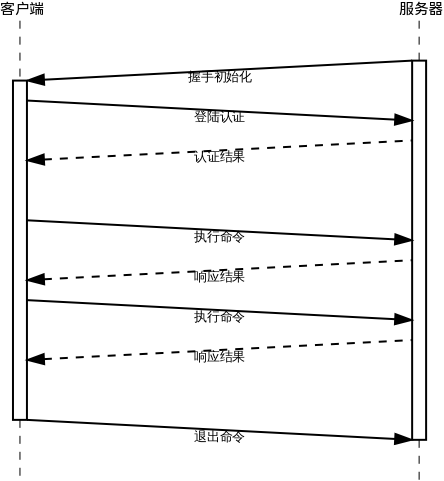
* 服务器 -> 客户端：握手初始化消息
* 客户端 -> 服务器：登陆认证消息
* 服务器 -> 客户端：认证结果消息

1.2 命令执行阶段

客户端认证成功后，会进入命令执行阶段，交互过程如下：

* 客户端 -> 服务器：执行命令消息
* 服务器 -> 客户端：命令执行结果

**MySQL客户端与服务器的完整交互过程如下**：



## 2 基本类型

### 2.1 整型值

分为固定长度整型和可变长度整型。

#### 2.1.1 固定长度整型（Fixed-length integers）

MySQL报文中整型值分别有1、2、3、4、6、8字节长度，分别表示为int<1>、int<2>、int<3>、int<4>、int<6>、int<8>。

#### 2.1.2可变长度整型（Length-encoded integers）

如果整型值 Value < 251 则使用1字节存储此值。

如果 Value >= 251且Value < 65536，则使用占用3个字节，使用0xfc+ int<2>存储。

如果 Value >= 65536且Value < 16777216，则使用占用4个字节，使用0xfd+ int<3>存储。

如果 Value >= 16777216且Value < 2^64，则使用占用9个字节，使用0xfe+ int<8>存储。

注意：0xfb、0xfe、0xff出现需要根据包的长度进行判断，因不同长度有不同意义。0xfb也可能是代表NULL包，0xfe代表EOF包，0xff代表ERR包。表示为int<lenenc>。

### 2.2长度编码（Length Coded Binary）

数据长度不固定，长度值由数据前的1-9个字节决定，其中长度值所占的字节数不定，字节数由第1个字节决定，如下表：

| 第一个字节值 | 后续字节数 | 长度值说明 |
| --- | --- | --- |
| 0-250 | 0 | 第一个字节值即为数据的真实长度 |
| 251 | 0 | 空数据，数据的真实长度为零 |
| 252 | 2 | 后续额外2个字节标识了数据的真实长度 |
| 253 | 3 | 后续额外3个字节标识了数据的真实长度 |
| 254 | 8 | 后续额外8个字节标识了数据的真实长度 |

### 2.3 字符串

分为固定长度字符串、以NULL（0x00）结尾字符串、包结束字符串、可变长度字符串、变量型字符串。

固定长度字符串即字符串长度固定，表示为string<fix>。

NULL结尾字符串即字符串以NULL（0x00）结束，表示为string<NULL>。

包结束字符串即字符串从起始开始直到此数据包结束，表示为string <EOF>。

可变长度字符串即根据上图<长度编码>所示根据标示长度字段的字节确定字符串长度，表示为string<lenenc>。

变量型字符串即根据协议包前面定义的某个长度字段，确定字符串长度。表示为string<var>。

## 3数据包定义

Mysql数据包由包头和包体两部分组成，包头占用固定的4个字节，包体体长度由包头中的长度字段决定。包头结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 层级 | 字段含义 | 数据类型 | 选项说明 |
| {packet\_len} | 0 | 包体长度 | int<3> | Mysql包体长度，最大发送16M |
| {*sequence\_id*} | 0 | 序列号 | int<1> | 从0开始，一次请求应答定义结束，需从0重新开始。 |

### 3.1包体结构

认证阶段服务端发送握手请求，客户端响应握手应答，支持使用[Secure Password Authentication](http://dev.mysql.com/doc/internals/en/secure-password-authentication.html)的方式进行认证，服务端以OK包建立连接或以ERR包断开连接。

指令阶段客户端发起请求，服务进行应答的方式。分为[Text Protocol](http://dev.mysql.com/doc/internals/en/text-protocol.html)(文本协议) 和[Prepared Statements](http://dev.mysql.com/doc/internals/en/prepared-statements.html)(二进制协议)，其根据不同指令进行请求。

#### 3.1.1 握手请求Handshake

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| { protocol \_version} | 协议版本 | int<1> | 0x0a支持V10协议，0x09目前暂不支持V9 |
| { server \_version} | 服务端描述信息 | string<NULL> |  |
| { connection\_id} | 连接ID | int<4> |  |
| { auth\_data\_1} | 鉴权随机数part1 | String<8> |  |
| {filler} | 填充值 | int<1> | 固定填写 0x00 |
| { capability\_flags\_lower} | 服务端能力标志低位 | int<2> | 存放能力标志低2位字节，定义见附录4.1 |
| 如果数据包还有后续字节 | | | |
| [character\_set] | 服务端字符编码 | int<1> |  |
| [status\_flags] | 状态标志 | int<2> | 定义见附录4.2 |
| [capability\_flags\_higher] | 服务端能力标志高位 | int<2> | 存放能力标志高2位字节 |
| [auth\_plugin\_len] | Part2鉴权随机性长度 | int<1> | 若capability\_flags& CLIENT\_PLUGIN\_AUTH为true，则为后续鉴权随机数长度，否则固定填写0x00。 |
| [reserved] | 保留字段 | String<10> | 固定全部填写0x00 |
| [auth\_data\_1] | 鉴权随机数part2 | String<var> | 若capability\_flags& CLIENT\_SECURE\_CONNECTION为true，则长度为var=max(13, [auth\_plugin\_len]-8)，否则无此字段 |
| [auth\_plugin\_name] | 鉴权插件名称 | String[NULL] | 若capability\_flags& CLIENT\_PLUGIN\_AUTH为true，则填写名称，否则无此字段 |

其中：{}必选，[]为可选，\*为多个

如果客户端连接上后服务端发送的是ERR\_Packet，则说明服务端有问题，客户端断开连接。

握手包的解析查看代码sqlClient.cpp:569

#### 3.1.2 握手应答HandshakeResponse

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| { capability\_flags} | 客户端能力标志 | int<4> | 必须capability\_flags| CLIENT\_PROTOCOL\_41，其他标志根据实际需要填写,定义见附件4.1 |
| { max\_packet\_size} | 客户端支持最大包长 | Int<4> |  |
| { character\_set } | 字符编码 | int<1> |  |
| { reserved} | 保留字段 | String<23> | 全部固定填写 0x00 |
| {username} | 用户名 | String<NULL> | 数据库登陆用户名 |
| [auth\_len] | 鉴权数据长度 | int<1>或int<lence> | 若capability\_flags& CLIENT\_PLUGIN\_AUTH\_LENENC\_CLIENT\_DATA为true，则长度类型为int<lence>，否则若capability\_flags& CLIENT\_SECURE\_CONNECTION为true  则长度类型为int<1>，其他则此字段不填写 |
| {auth\_data} | 鉴权数据 | String<var>或string<NULL> | 若capability\_flags & CLIENT\_PLUGIN\_AUTH\_LENENC\_CLIENT\_DATA或capability\_flags& CLIENT\_SECURE\_CONNECTION为true，则数据类型为String<var>，var=[auth\_len]，否则数据类型为string<NULL> |
| [database] | 数据库名 | string<NULL> | 如果capability\_flags& CLIENT\_CONNECT\_WITH\_DB为true，则填写此字段 |
| [auth\_plugin\_name] | 鉴权插件名称 | String<NULL> | 若capability\_flags& CLIENT\_PLUGIN\_AUTH为true，则填写此字段 |
| [key\_value\_len] | 所有KeyValue的长度 | int<lence> | 若capability\_flags& CLIENT\_CONNECT\_ATTRS为true，则须填写此字段与[key\_value]字段，否则不填；当长度为0时，key\_value不填 |
| [key\_value]\* | Key与value参数结构 | KeyValue |  |

KeyValue结构定义如下

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| { key} | 键 | String< lence > |  |
| { value} | 值 | String< lence > |  |

其中：{}必选，[]为可选，\*为多个

握手应答后，如果服务端鉴权成功则返回OK\_Packet建立连接，鉴权失败则返回ERR\_Packet断开连接。

auth\_data字段填写使用[Secure Password Authentication](http://dev.mysql.com/doc/internals/en/secure-password-authentication.html)的加密方式（SHA1进行多次加密）

握手应答包例子代码查看sqlClient.cpp:1323

鉴权数据代码查看sqlPassword.cpp:300

#### 3.1.3 成功应答包OK\_Packet

默认已经具有CLIENT\_PROTOCOL\_41的权能。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {header} | 消息头 | int<1> | 固定填写0x00 |
| {affected\_rows } | 操作影响的行数 | int< lence > | 当执行INSERT/UPDATE/DELETE语句时所影响的数据行数，无影响则填0x00 |
| { last\_insert\_id} | 索引ID | int< lence > | 该值为AUTO\_INCREMENT索引字段生成，如果没有索引字段，则为0x00。注意：当INSERT插入语句为多行数据时，该索引ID值为第一个插入的数据行索引值，而非最后一个。 |
| { status\_flags } | 状态标志 | int<2 > | 客户端可以通过该值检查命令是否在事务处理中。  定义见附件4.2 |
| { warnings } | 告警计数 | int<2> |  |
| [info] | 描述信息 | string<EOF>或string< lence > | 若capability\_flags& CLIENT\_SESSION\_TRACK为true，则长度类型为string< lence >，否则长度类型为string<EOF> |
| [session\_state\_changes] | 事务状态信息 | string< lence > | 若status\_flags & SERVER\_SESSION\_STATE\_CHANGED且capability\_flags& CLIENT\_SESSION\_TRACK为true，则填写此字段，其他则不填写此字段 |

其中{ status\_flags }与{ warnings }需要客户端具有CLIENT\_PROTOCOL\_41的权能。

对于session\_state\_changes的定义如下:

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {type} | 类型 | Int<1> | 0x00：系统变量改变 0x01:概要发生变化 0x02:track发生变化 |
| { data} | 数据信息 | String< lence > | 根据type定义不同 |

如果类型为0x00（**SESSION\_TRACK\_SYSTEM\_VARIABLES）**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {name} | 名称 | String< lence > |  |
| { value} | 值 | String< lence > |  |

如果类型为0x01（**SESSION\_TRACK\_SCHEMA**）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {name} | 名称 | String< lence > |  |

如果类型为0x02（**SESSION\_TRACK\_STATE\_CHANGE**）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {is\_tracked} | Session是否发生变化 | String< lence > | Bool类型，以ascii码0或1表示 |

#### 3.1.4 错误应答包ERR\_Packet

默认已经具有CLIENT\_PROTOCOL\_41的权能。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {header} | 消息头 | int<1> | 固定填写0xFF |
| {error\_code} | 错误码 | int<2> | 错误编码 |
| { sql\_state\_marker} | 服务端状态标志 | String<1> | 固定填写’#’字符 |
| { sql\_state } | 服务端状态 | String<5> | 错误编码转换后的值为5个字节ascii字符组成 |
| {error\_message} | 错误信息 | string<EOF> |  |

其中{ sql\_state\_marker}与{ sql\_state }需要客户端具有CLIENT\_PROTOCOL\_41的权能。

#### 3.1.5 EOF应答包EOF\_Packet

默认已经具有CLIENT\_PROTOCOL\_41的权能。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {header} | 消息头 | int<1> | 固定填写0xFE |
| { warnings } | 告警计数 | int<2> | 错误编码 |
| { status\_flags } | 状态标志 | int<2> | 定义见附件4.2 |

其中{ status\_flags }与{ warnings }需要客户端具有CLIENT\_PROTOCOL\_41的权能。

#### 3.1.6 文本指令协议Text Protocol

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {command} | 协议类型 | int<1> | 填写具体协议如下定义<协议别名><(协议值)> |
| [data] | 参数 | String[EOF] | 具体参数定义与协议类型相关 |

服务端内部使用指令与一些不常用指令不做说明包括COM\_SLEEP(0x00)、COM\_REFRESH(0x07)、COM\_SHUTDOWN(0x08)、 COM\_STATISTICS(0x09)、COM\_PROCESS\_INFO(0x0a)、COM\_CONNECT(0x0b)、COM\_PROCESS\_KILL(0x0c)、COM\_DEBUG(0x0d)、COM\_PING(0x0e)、COM\_TIME(0x0f)、COM\_DELAYED\_INSERT(0x10)、COM\_RESET\_CONNECTION(0x1f)、COM\_CHANGE\_USER (0x11)、COM\_DAEMON(0x1d)。

##### 3.1.6.1 COM\_QUIT(0x01)

功能：客户端关闭连接

请求参数：无

应答：返回OK\_Packet或无应答直接关闭连接。

##### 3.1.6.2 COM\_INIT\_DB(0x02)

功能：切换数据库

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {schema\_name} | 数据库名 | String<EOF> |  |

应答：返回OK\_Packet或ERR\_Packet。

##### 3.1.6.3 COM\_QUERY (0x03)

功能：SQL语句执行

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {query} | Sql语句 | String<EOF> |  |

应答：返回OK\_Packet或ERR\_Packet或[ProtocolText::Resultset](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-ProtocolText::Resultset)(结果集中一次返回多个包)。或[Protocol::LOCAL\_INFILE\_Request](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-Protocol::LOCAL_INFILE_Request)包。

对于首包含义：

如果列数量为0则返回OK\_Packet；如果列数量无效，且列数量为0xFF则为ERR\_Packet；如果列数量无效，且列数量为0xFB则为LOCAL\_INFILE\_Request包；如果列数量包有效大于0，则为[Resultset](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-ProtocolText::Resultset)结果集。

对于LOCAL\_INFILE本地文件请求传输的方式，暂不支持，不做过多说明。

[Resultset](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-ProtocolText::Resultset)结果集定义如下：

| 结构 | 说明 |
| --- | --- |
| [Result Set Header] | 列数量包（1个） |
| [ColumnDefinition] | 列信息包（多个，每个列的信息为一个包，包个数为列数量） |
| [EOF] | 列结束包（1个） |
| [Text Resultset Row] | 行数据包（多个，每一行数据封装一个包） |
| [EOF] | 数据结束（1个） |

Result Set Header结构定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {column\_count} | 列数量 | Int<lence> | 大于0，列的个数与sql语句关联列数一致（如查询10列数的值，则列数为10）。 |

ColumnDefinition包结构定义如下：

默认已经具有CLIENT\_PROTOCOL\_41的权能。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {catalog} | 目录名 | String<lence> |  |
| {schema} | 数据库名 | String<lence> |  |
| { table} | 表名 | String<lence> |  |
| {org\_table } | 原始表名 | String<lence> |  |
| {name} | 列名 | String<lence> |  |
| {org\_name} | 原始列名 | String<lence> |  |
| {fix\_filed\_len} | 列结构定长字段总长 | Int<lence> | 固定填写0x0c，占用1个字节 |
| {character\_set} | 列字符编码 | Int<2> |  |
| {column\_len} | 列长度 | Int<4> |  |
| {column\_type} | 列类型 | Int<1> | 定义见附录4.3 |
| {flags} | 列标记 | Int<2> | 定义见附录4.4 |
| {decimals} | 列精度 | Int<1> | 标识数值的精度 |
| {filler} | 填充值 | string<2> | 固定填写0x0000 |
| [default\_values\_len] | 列默认值长度 | Int<lence> | 对于COM\_FIELD\_LIST指令此字段必须进行填写，其他指令都不需要填写此字段 |
| [default values] | 列默认值 | String<lence> | 对于COM\_FIELD\_LIST指令此字段必须进行填写，其他指令都不需要填写此字段 |

Text Resultset Row包结构定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {row\_value}\* | 字段对应的值 | String<lence> | 多个列字段对用的值，对于NULL的字段值使用0xFB表示，其他均遵循String<lence>类型进行填写 |

如果无记录则直接返回EOF包。

注意：如果返回的最后一个EOF包的status\_flags&[SERVER\_MORE\_RESULTS\_EXISTS](http://dev.mysql.com/doc/internals/en/status-flags.html" \l "flag-SERVER_MORE_RESULTS_EXISTS)（0x0008）为true时，表明还有后续[multi-resultset](http://dev.mysql.com/doc/internals/en/multi-resultset.html" \o "14.8.1 Multi-Resultset)包，处理按照多结果集协议方式进行处理。

测试代码参考：sqlClient:1009

##### 3.1.6.4 COM\_FIELD\_LIST (0x04)

功能：查询表的列定义

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {table\_name} | 表名 | String<NULL> | 要查询的表名 |
| [field\_wildcard] | 列名或通配符（可选） | String<EOF> | 通配符为’%’或‘\_’，无此字段则返回所有字段定义。 |

应答：返回ERR\_Packet或如下结构

|  |  |
| --- | --- |
| 结构 | 说明 |
| [ColumnDefinition] | 列信息包（多个，每个列的信息为一个包，包个数为列数量） |
| [EOF] | 列结束包（1个） |

注意：返回的ColumnDefinition结构中必须包含[default\_values\_len]和[default values]字段定义。

##### 3.1.6.5 COM\_CREATE\_DB (0x05)

功能：创建数据库

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {schema\_name } | 数据库名 | String<EOF> |  |

应答：返回OK\_Packet或ERR\_Packet。

##### 3.1.6.6 COM\_DROP\_DB (0x06)

功能：删除数据库

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {schema\_name } | 数据库名 | String<EOF> |  |

应答：返回OK\_Packet或ERR\_Packet。

#### 3.1.7 二进制指令协议Prepared Statements

二进制指令协议与文本指令协议组成部分相同，由{command}和{data}两部分组成，具体command的定义和data定义如下面介绍。

##### 3.1.7.1 COM\_STMT\_PREPARE(0x16)

功能：预处理SQL语句，使用带有’?’占位符的SQL语句时发送该消息。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {query } | 预处理语句 | String<EOF> |  |

应答：返回[COM\_STMT\_PREPARE\_OK](http://dev.mysql.com/doc/internals/en/com-stmt-prepare-response.html" \l "packet-COM_STMT_PREPARE_OK)或ERR\_Packet。

[COM\_STMT\_PREPARE\_OK](http://dev.mysql.com/doc/internals/en/com-stmt-prepare-response.html" \l "packet-COM_STMT_PREPARE_OK)返回成功结果集定义如下：

| 结构 | 说明 |
| --- | --- |
| [PREPARE\_OK] | 语句信息 |
| [paramDefinition] | 参数信息（多个，每个参数信息为一个包，包数目为参数个数） |
| [EOF] | 参数结束包（1个） |
| [ColumnDefinition] | 列信息包（多个，每个列的信息为一个包，包数目为列个数） |
| [EOF] | 列结束包（1个） |

如果语句信息中的参数个数大于0,则有后续参数定义信息（参数定义与列定义相同）与参数结束EOF包，无参数信息（即参数个数为0）则不返回参数定义的包与参数结束包。列的定义与参数定义的返回同理。

PREPARE\_OK包定义如下

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {status} | 状态 | Int<1> | 固定填写0x00 |
| {statement\_id} | 预处理语句ID | Int<4> | 需要记录此语句ID，后续的操作均需根据此关键值 |
| { num\_columns} | 列个数 | Int<2> | 列个数为语句的关联的列数 |
| { num\_params } | 参数个数 | Int<2> | 参数个数为语句关联的参数个数 |
| {reserved\_1} | 保留值 | Int<1> | 固定填写0x00 |
| { warning\_count} | 告警数 | Int<2> |  |

测试代码参考：sqlClient:436

##### 3.1.7.2 COM\_STMT\_SEND\_LONG\_DATA (0x18)

功能：发送预处理语句参数数据(参数类型为text或blob时使用)

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {statement\_id} | 预处理语句ID | Int<4> | 与prepare的语句ID保持一致 |
| {param\_id} | 参数序号 | Int<2> | 与prepare参数的顺序一致 |
| {data} | 数据 | String<EOF> |  |

应答：无应答返回。

注意必须要在COM\_STMT\_EXECUTE命令执行之前执行此语句。

##### 3.1.7.3 COM\_STMT\_EXECUTE (0x17)

功能：执行预处理语句。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {statement\_id} | 预处理语句ID | Int<4> | 与prepare的语句ID保持一致 |
| {flags} | 标志位 | Int<1> | 0x00: CURSOR\_TYPE\_NO\_CURSOR  0x01: CURSOR\_TYPE\_READ\_ONLY  0x02: CURSOR\_TYPE\_FOR\_UPDATE  0x04: CURSOR\_TYPE\_SCROLLABLE |
| {iteration-count} | 保留字段 | Int<4> | 值固定为0x01 |
| 如果num\_params（参数个数）大于0 | | | |
| {null-bitmap} | 参数空图位 | String<var> | Var= (num\_params+7+offset)/8, offset= 0 |
| { param\_bound\_flag} | 新参数绑定标记 | Int<1> | 0x00：无参数绑定 0x01：绑定参数 |
| 如果param\_bound\_flag等于1 | | | |
| {type\_params}\* | 每个参数的类型 | string<var> | Var= num\_params\*2;低位为类型，高位如果为unsigned则高位填0x80，否则为0x00参数类型见附录4.3。需要根据参数类型进行设置参数设置。 |
| {value\_params}\* | 每个参数的值 | String<EOF> | 根据不同参数类型每个参数值有不同长度，具体参数类型对应的参数值定义见附录4.3。 |

应答：返回OK\_Packet或ERR\_Packet或二进制Resultset结果集(结果集中一次返回多个包)。

应答说明：Binary Protocol Resultset结果集与Text Protocol Resultset结果集类似，有两点需要注意：1、设置flags字段为0时则execute后所有rows返回直到EOF包，如果存在多结果集则按多结果集协议方式进行处理。2、设置flags字段为1，则判断接收的第一个EOF包status\_flags&[SERVER\_STATUS\_CURSOR\_EXISTS](http://dev.mysql.com/doc/internals/en/status-flags.html" \l "flag-SERVER_STATUS_CURSOR_EXISTS)（0x0040）是否为true，若为true则需要使用[COM\_STMT\_FETCH](http://dev.mysql.com/doc/internals/en/com-stmt-fetch.html" \l "packet-COM_STMT_FETCH)获取后续的行数据，否则直接接收行数据。3、Binary Protocol ResultsetRow与Text Protocol ResultRow定义有差异。

Binary Protocol ResultsetRow包体定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {header} | 消息头 | Int<1> | 固定填写为0x00 |
| {null\_bitmap} | 数据空图位 | String<var> | Var = (num\_columns+7+offset)/8, offset =2 |
| {values} | 每列对应字段的值 | String<EOF> |  |

null\_bitmap的计算方式例子代码如下：

//offset = 0 计算参数空图位

//offset = 2 计算行数据空图位

//pos为null\_bitmap的buffer空图位

int storeNull(unsigned int ColumnNo,unsigned char\* pos,int offset /\* = 0 \*/)

{

unsigned char field\_pos = 0;//列序号（从开始）

unsigned char \*to;//指向buff的指针

unsigned char byte;//buff下标

unsigned char bit;//1向左移动指定位数后的值

field\_pos = ColumnNo;//假设第列值为NULL

byte= (field\_pos+ offset)/8;//计算buff的下标为

bit= (1 << ((field\_pos+ offset) & 7));//计算移动的位数为,1向左移动位bit的值为x80

to= (unsigned char\*) pos+byte;//to指针指向buff

\*to= (char) ((unsigned char) \*to | (unsigned char) bit);//此处需要加上原始此位的值

printf("to is %02x\n",\*to);//输出为buff[0]的值为x80

return 0;

}

NULL字段判断代码如下：

// pos为null\_bitmap的buffer

// param\_no为字段的序号（从0开始）

// offset 为ResultsetRow的 offset 值为2，COM\_STMT\_EXECUTE的offset 值为0

inline bool is\_param\_null(const unsigned char \*pos, unsigned long param\_no, unsigned char offset)

{

return pos[(param\_no+offset)/8] & (1 << ((param\_no + offset)& 7));

}

测试代码参考：sqlClient:652

##### 3.1.7.4 COM\_STMT\_FETCH(0x1C)

功能：从COM\_STMT\_EXECUTE的执行后的结果集中获取行数据信息。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {statement\_id} | 预处理语句ID | Int<4> | 与prepare的语句ID保持一致 |
| {num\_rows} | 行数 | Int<4> | 一次提取的行数 |

应答：ERR\_Packet或Multi-RowsResult。

Binary-RowResult定义如下：

|  |  |
| --- | --- |
| [BinaryProtocolResultsetRow]\* | 二进制行数据包（多个，最大返回与num\_rows相同） |
| [EOF] | 行结束包（1个） |

说明：

Multi-RowsResult由多个Binary-RowResult结果集组成，每个Binary-RowResult由多个二进制行数据包和一个EOF包组成，当数据库中满足条件的行数大于等于设置的num\_rows字段值则返回与之数目相同的行数据包，小于则将满足条件的行数据全部返回。当返回的EOF包的status\_flags&[SERVER\_STATUS\_CURSOR\_EXISTS](http://dev.mysql.com/doc/internals/en/status-flags.html" \l "flag-SERVER_STATUS_CURSOR_EXISTS)（0x0040）为true说明还有后续Binary-RowResult，则需要继续使用com\_stmt\_fetch命令获取后续行数据直到status\_flags& status\_flags&[SERVER\_STATUS\_CURSOR\_EXISTS](http://dev.mysql.com/doc/internals/en/status-flags.html" \l "flag-SERVER_STATUS_CURSOR_EXISTS)为true结束。

测试代码参考：sqlClient:652

##### 3.1.7.5 COM\_STMT\_CLOSE(0x19)

功能：关闭预处理语句

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {statement\_id} | 预处理语句ID | Int<4> | 与prepare的语句ID保持一致 |

应答：无。

测试代码参考：sqlClient:652

##### 3.1.7.6 COM\_STMT\_RESET(0x1a)

功能：清空预处理语句参数缓存，关闭游标

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {statement\_id} | 预处理语句ID | Int<4> | 与prepare的语句ID保持一致 |

应答：返回OK\_Packet或ERR\_Packet。

#### 3.1.8 多结果集协议

Mysql允许com\_query一次执行多条sql语句，每个sql语句之间以“；”分隔。此功能需要客户端和服务端具有[CLIENT\_MULTI\_STATEMENTS](http://dev.mysql.com/doc/internals/en/capability-flags.html" \l "flag-CLIENT_MULTI_STATEMENTS)(0x00010000)的权能，可通的握手建立连接时设置或通过com\_set\_option(0x1b)命令设置。

多结果集协议的支持需要客户端和服务端分别拥有CLIENT\_MULTI\_RESULTS（0x00020000）和CLIENT\_PS\_MULTI\_RESULTS（0x00040000）的权能。从mysql5.5.3开始mysql支持CLIENT\_PS\_MULTI\_RESULTS，当使用二进制协议执行存储过程时，如果绑定输出参数，则必需要具有CLIENT\_PS\_MULTI\_RESULTS的权能标志。[multi-resultset](http://dev.mysql.com/doc/internals/en/multi-resultset.html" \o "14.8.1 Multi-Resultset)(多结果集)必须要具有CLIENT\_MULTI\_RESULTS的权能标志。

存储过程或多sql命令均会按照多结果集协议进行返回，[multi-resultset](http://dev.mysql.com/doc/internals/en/multi-resultset.html" \o "14.8.1 Multi-Resultset)(多结果集)定义如下：

|  |  |
| --- | --- |
| [[Resultset](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-ProtocolText::Resultset)]\* | 多个结果集合，二进制协议则为二进制格式结果集，文本协议则为文本格式结果集 |
| {OK} | OK\_Packet包（1个） |

说明：

1、[multi-resultset](http://dev.mysql.com/doc/internals/en/multi-resultset.html" \o "14.8.1 Multi-Resultset)(多结果集)由多个ResultSet结果集组成和一个标识结束的OK\_Packet包组成，[Resultset](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "packet-ProtocolText::Resultset)对于select或绑定输出参数会返回，其他不返回。

2、对resultset的EOF都需要判断status\_flags& SERVER\_MORE\_RESULTS\_EXISTS是否为true，如果为true则有后续ResultSet结果集包，否则无ResultSet结果集包。注意如果为多结果集中的包则每个EOF的status\_flags& SERVER\_MORE\_RESULTS\_EXISTS是均为true。

3、对于二进制协议com\_stmt\_execute执行的结果，需要对Result的第一EOF包的status\_flags&SERVER\_PS\_OUT\_PARAMS是否为true进行判断，如果为true则说明此结果集为绑定输出参数的结果集，否则为普通select结果集。注意：抓包结果为：SERVER\_PS\_OUT\_PARAMS标志只会出现在第一个EOF包，后面的EOF则不会出现，而SERVER\_MORE\_RESULTS\_EXISTS标志则每个EOF包都会出现；官方文档描述为：SERVER\_PS\_OUT\_PARAMS和SERVER\_MORE\_RESULTS\_EXISTS都只会在输出参数结果集的第一个EOF包出现，后面的EOF包则不会出现。

4、对于二进制协议执行存储过程同时存在绑定输出参数的定义结果集和select结果集，结果集返回顺序为先返回select结果集，然后返回输出参数结果集，最后返回OK结束包。

5、对于多语句或存储过程OK包所返回影响的行数，为执行最后一条sql语句所影响的行数，select影响的行数为0。

##### 3.1.8.1 COM\_SET\_OPTION(0x1b)

同文本指令协议组成部分相同，由{command}和{data}两部分组成，具体data定义如下：

功能：设置com\_query是否支持多sql语句

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段含义 | 数据类型 | 选项说明 |
| {option\_flag} | 预处理语句ID | Int<1> | 0: MYSQL\_OPTION\_MULTI\_STATEMENTS\_ON  1: MYSQL\_OPTION\_MULTI\_STATEMENTS\_OFF |

应答：成功则返回OK\_Packet,失败则返回ERR\_Packet。

## 4附录

### 4.1能力标志定义

| 标志位名称 | 标志位 | 说明 |
| --- | --- | --- |
| CLIENT\_LONG\_PASSWORD | 0x0001 | 使用完善后的[Old Password Authentication](http://dev.mysql.com/doc/internals/en/old-password-authentication.html" \l "packet-Authentication::Old)鉴权方式 |
| CLIENT\_FOUND\_ROWS | 0x0002 | 在OK包中使用发现行数替换影响的行数 |
| CLIENT\_LONG\_FLAG | 0x0004 | ColumnDefinition320中使用（低版本使用） |
| CLIENT\_CONNECT\_WITH\_DB | 0x0008 | [Handshake\_Response\_Packet](http://dev.mysql.com/doc/internals/en/connection-phase-packets.html" \l "packet-Protocol::HandshakeResponse" \o "Protocol::HandshakeResponse:)中支持数据库名信息 |
| CLIENT\_NO\_SCHEMA | 0x0010 | 不支持database.table.column方式 |
| CLIENT\_COMPRESS | 0x0020 | 可以使用 compression protocol |
| CLIENT\_ODBC | 0x0040 | 标识Odbc 客户端 |
| CLIENT\_LOCAL\_FILES | 0x0080 | 支持[LOCAL\_INFILE\_request](http://dev.mysql.com/doc/internals/en/com-query-response.html" \l "local-infile-request" \o "14.6.4.1.2 LOCAL INFILE Request)请求 |
| CLIENT\_IGNORE\_SPACE | 0x0100 | 忽略 '('之前的空格 |
| CLIENT\_PROTOCOL\_41 | 0x0200 | 支持新的 4.1 协议 |
| CLIENT\_INTERACTIVE | 0x0400 | This is an interactive client |
| CLIENT\_SSL | 0x0800 | 支持handshake后切换SSL登陆 |
| CLIENT\_IGNORE\_SIGPIPE | 0x1000 | 忽略 sigpipes信号 |
| CLIENT\_TRANSACTIONS | 0x2000 | 在EOF包中发送状态标志 |
| CLIENT\_RESERVED | 0x4000 | 未使用 |
| CLIENT\_SECURE\_CONNECTION | 0x8000 | 支持 4.1 的[Authentication::Native41](http://dev.mysql.com/doc/internals/en/secure-password-authentication.html" \l "packet-Authentication::Native41) |
| CLIENT\_MULTI\_STATEMENTS | 0x0001 0000 | 支持多sql语句执行 |
| CLIENT\_MULTI\_RESULTS | 0x0002 0000 | 支持多结果集 |
| CLIENT\_PS\_MULTI\_RESULTS | 0x0004 0000 | 支持com\_stmt\_execute绑定输出参数（5.5.3以后） |
| CLIENT\_PLUGIN\_AUTH | 0x0008 0000 | 支持插件数据鉴权 |
| CLIENT\_CONNECT\_ATTRS | 0x0010 0000 | 支持设置连接参数（[HandshakeResponse41](http://dev.mysql.com/doc/internals/en/connection-phase-packets.html" \l "packet-Protocol::HandshakeResponse41" \o "Protocol::HandshakeResponse41:)） |
| CLIENT\_PLUGIN\_AUTH\_LENENC\_CLIENT\_DATA | 0x0020 0000 | 支持[HandshakeResponse41](http://dev.mysql.com/doc/internals/en/connection-phase-packets.html" \l "packet-Protocol::HandshakeResponse41" \o "Protocol::HandshakeResponse41:)可变长度鉴权数据 |
| CLIENT\_CAN\_HANDLE\_EXPIRED\_PASSWORDS | 0x0040 0000 | 支持密码过期处理 |
| CLIENT\_SESSION\_TRACK | 0x0080 0000 | 支持OK包中设置status\_flag为SERVER\_SESSION\_STATE\_CHANGE和发送session\_state\_changes数据 |
| CLIENT\_DEPRECATE\_EOF | 0x0100 0000 | 在Text Resultset后可发送OK包 |

### 4.2状态标志定义

| 状态名称 | 状态值 |
| --- | --- |
| SERVER\_STATUS\_IN\_TRANS | 0x0001 |
| SERVER\_STATUS\_AUTOCOMMIT | 0x0002 |
| SERVER\_MORE\_RESULTS\_EXISTS | 0x0008 |
| SERVER\_STATUS\_NO\_GOOD\_INDEX\_USED | 0x0010 |
| SERVER\_STATUS\_NO\_INDEX\_USED | 0x0020 |
| SERVER\_STATUS\_CURSOR\_EXISTS | 0x0040 |
| SERVER\_STATUS\_LAST\_ROW\_SENT | 0x0080 |
| SERVER\_STATUS\_DB\_DROPPED | 0x0100 |
| SERVER\_STATUS\_NO\_BACKSLASH\_ESCAPES | 0x0200 |
| SERVER\_STATUS\_METADATA\_CHANGED | 0x0400 |
| SERVER\_QUERY\_WAS\_SLOW | 0x0800 |
| SERVER\_PS\_OUT\_PARAMS | 0x1000 |
| SERVER\_STATUS\_IN\_TRANS\_READONLY | 0x2000 |
| SERVER\_SESSION\_STATE\_CHANGED | 0x4000 |
|  |  |

### 4.3列字段类型定义

| **类型值** | **名称 值长度** |
| --- | --- |
| 0x00 | MYSQL\_TYPE\_DECIMAL string<lence> |
| 0x01 | MYSQL\_TYPE\_TINY int<1> |
| 0x02 | MYSQL\_TYPE\_SHORT int<2> |
| 0x03 | MYSQL\_TYPE\_LONG int<4> |
| 0x04 | MYSQL\_TYPE\_FLOAT string<4> |
| 0x05 | MYSQL\_TYPE\_DOUBLE string<8> |
| 0x06 | MYSQL\_TYPE\_NULL 存于NULL\_BitMap |
| 0x07 | MYSQL\_TYPE\_TIMESTAMP date<lence> |
| 0x08 | MYSQL\_TYPE\_LONGLONG int<8> |
| 0x09 | MYSQL\_TYPE\_INT24 int<4> |
| 0x0A | MYSQL\_TYPE\_DATE date<lence> |
| 0x0B | MYSQL\_TYPE\_TIME time<lence> |
| 0x0C | MYSQL\_TYPE\_DATETIME date<lence> |
| 0x0D | MYSQL\_TYPE\_YEAR int<2> |
| 0x0F | MYSQL\_TYPE\_VARCHAR (new in MySQL 5.0) string<lence> |
| 0x10 | MYSQL\_TYPE\_BIT (new in MySQL 5.0) string<lence> |
| 0xF6 | MYSQL\_TYPE\_NEWDECIMAL (new in MYSQL 5.0) string<lence> |
| 0xF7 | MYSQL\_TYPE\_ENUM string<lence> |
| 0xF8 | MYSQL\_TYPE\_SET string<lence> |
| 0xF9 | MYSQL\_TYPE\_TINY\_BLOB string<lence> |
| 0xFA | MYSQL\_TYPE\_MEDIUM\_BLOB string<lence> |
| 0xFB | MYSQL\_TYPE\_LONG\_BLOB string<lence> |
| 0xFC | MYSQL\_TYPE\_BLOB string<lence> |
| 0xFD | MYSQL\_TYPE\_VAR\_STRING string<lence> |
| 0xFE | MYSQL\_TYPE\_STRING string<lence> |
| 0xFF | MYSQL\_TYPE\_GEOMETRY string<lence> |

date<lence>定义：

采用length和日期值的组合形式如下：

Length int<1>(有效值为0、4、7、11) 、**year** int<2>、month int<1>、day int<1>、hour int<1>、minute int<1>、second int<1>、 micro\_second int<4>

如果年、月、日、时、分、秒、微秒都为0，则length 为0

如果时、分、秒、微秒为0，则length 为4

如果微秒为0，则length 为7

如果年、月、日、时、分、秒、微秒都不为0，则length为11

time<lence>定义：

采用length和时间值的组合形式如下：

Length int<1>(有效值为0、8、12) 、is\_negative int<1>(1表示-号、0表示+号)、days int<4>、hours int<1>、minutes int<1>、seconds int<1> 、micro\_seconds int<4>

如果天数、小时数、分数、秒数、微秒数都为0，则length 为0

如果微秒数为0，则length 为8

如果天数、小时数、分数、秒数、微秒数都不为0，则length为12

### 4.4列字段标志定义

| **标志位** | **名称** |
| --- | --- |
| 0x0001 | NOT\_NULL\_FLAG |
| 0x0002 | PRI\_KEY\_FLAG |
| 0x0004 | UNIQUE\_KEY\_FLAG |
| 0x0008 | MULTIPLE\_KEY\_FLAG |
| 0x0010 | BLOB\_FLAG |
| 0x0020 | UNSIGNED\_FLAG |
| 0x0040 | ZEROFILL\_FLAG |
| 0x0080 | BINARY\_FLAG |
| 0x0100 | ENUM\_FLAG |
| 0x0200 | AUTO\_INCREMENT\_FLAG |
| 0x0400 | TIMESTAMP\_FLAG |
| 0x0800 | SET\_FLAG |

### 4.5常见字符编码定义

|  |  |
| --- | --- |
| 0x08 | latin1\_swedish\_ci |
| 0x21 | utf8\_general\_ci |
| 0x3f | binary |

具体使用语句SELECT id, collation\_name FROM information\_schema.collations ORDER BY id;进行查询。

### 4.6协议调测总结

对于事务的处理：

client调用com\_query(0x03)命令执行sql语句的方式发送mysqld进行处理。

具体sql语句如 SET AUTOCOMMIT=0、SET AUTOCOMMIT=1、BEGIN、

ROLLBACK、COMMIT。

对于二进制协议：

Mysql客户端API中的mysql\_stmt\_attr\_set根据类型和值可以设置com\_stmt\_execute中的{flags}字段值和com\_stmt\_fetch命令中的{num\_rows}字段值，设置的类型分别为STMT\_ATTR\_CURSOR\_TYPE和STMT\_ATTR\_PREFETCH\_ROWS（对于STMT\_ATTR\_UPDATE\_MAX\_LENGTH类型为客户端自身使用）。需要注意的是select语句执行后每次使用com\_stmt\_fetch命令提取行数生效，需要同时设置flags字段为0x01和num\_rows字段有效。

mysql\_stmt\_bind\_param则会根据API绑定的类型和值分别对com\_stmt\_execute的{type\_params}\*和{value\_params}\*字段进行赋值设置。

对于存储过程com\_stmt\_execute中对flags字段的设置无效。