Leader.us

高端私塾培训

高级网络编程篇

Expert Java
Leader.us

MySQL报文编码与解析

必须掌握的位操作

Expert Java Leader.us

- **位移操作**：<<”, “>>”, “>>>”在Java中是左移、有符号右移和无符号右移运算符

  t>>>n的含义就是把整数t右移n位，高位补上零。所以如果t是个负数，最高位是1，那么经过无符号右移之后，就成了一个正数。比如 -1>>>31=1。

- **位运算**：与（&）、非（~）、或（|）、异或（^）

位移操作与位运算符只对int值进行操作

经典场景： C的无符号数与Java有符号数的转换

```
byte b1 = (byte)255
short  n = b1 & 0xFF
```

0x000000FF（int类型） → 0xFFL（Long类型）

位操作的典型网络编程例子

Expert Java
Leader.us

报文类型，一个字节

对于C里的无符号的byte，比如240，即0xf0=11110000，如何在Java里正确获取？

```
byte a=(byte) 0xf0;
System.out.println(a);
int b=0xff&a;
System.out.println(b);
```

第3到4位为消息紧急程度标示

用位运算的&即可解决问题

```
int flg = 0B01101000;
byte b = (byte) ((flg & 0B00110000) >> 4);
System.out.println(b);
```

无符号数的转换

Expert Java Leader.us

```
public int getUnsignedByte (byte data){      //将data字节型数据转换为0~255 (0xFF 即BYTE)。
    return data&0xFF;
}
public int getUnsignedByte (short data){      //将data字节型数据转换为0~65535 (0xFFFF 即 WORD)。
    return data&0xFFFF;
}
public long getUnsignedIntt (int data){     //将int数据转换为0~4294967295 (0xFFFFFFFF即DWORD)。
    return data&0xFFFFFFFF;
}
```

Big Endian VS Little Endian

Little-endian： A computer architecture that stores multiple-byte numerical values with the least significant byte (LSB) values first.But Big endian byte ordering has been chosen as the "neutral" or standard for network data exchange and thus Big Endian byte ordering is also known as the "**Network Byte Order**". Thus Little Endian systems will convert their internal Little Endian representation of data to Big Endian byte ordering when writing to the network via a socket.

Little-endian:最低字节在最低位，最高字节在最高位

更符合人的理解

MySQL使用了哪种次序？

Expert Java Leader.us

Protocol::Packet

https://dev.mysql.com/doc/internals/en/mysql-packet.html

Data between client and server is exchanged in packets of max 16MByte size.

Payload

A fixed-length integer stores its value in a series of bytes with the least significant byte first.

Type	Name	Description
int<3>	*payload_length*	Length of the payload. The number of bytes in the packet beyond the initial 4 bytes that make up the packet header.
int<1>	*sequence_id*	Sequence ID
string<var>	*payload*	[len=payload_length] payload of the packet

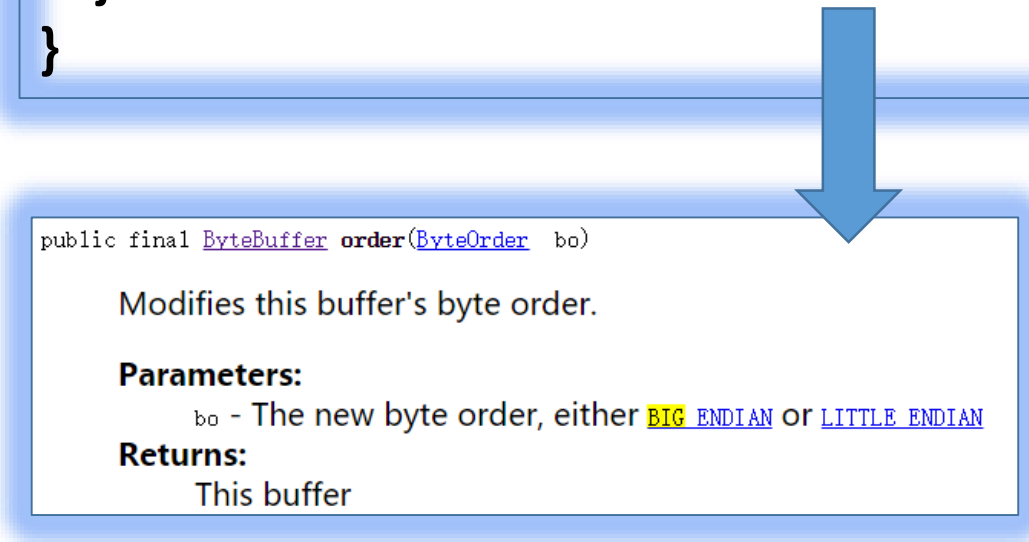
MySQL报文中消息的长度字段

```
public static final int getPacketLength(ConDataBuffer buffer, int offset) throws IOException {
                    int length = buffer.getByte(offset) & 0xff;
                    length |= (buffer.getByte(++offset) & 0xff) << 8;
                    length |= (buffer.getByte(++offset) & 0xff) << 16;
                    return length + msyql_packetHeaderSize;
          }
```

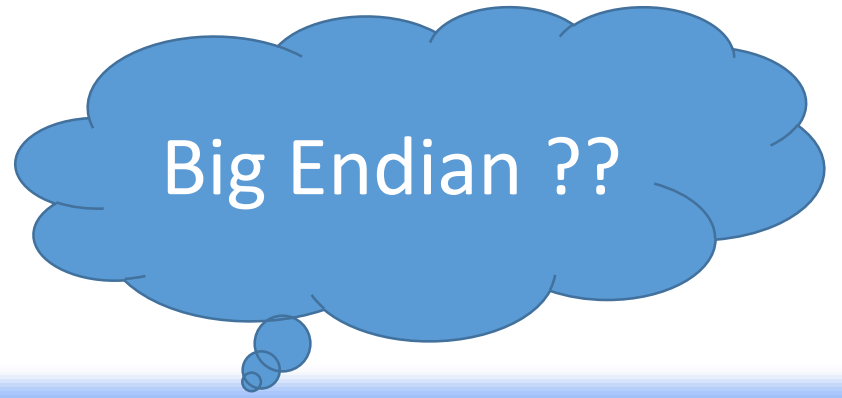
int到byte[]的转换

Expert Java Leader.us

```
public class Main{
    public static void main(String[] args) {
        ByteBuffer b = ByteBuffer.allocate(4);
        b.putInt(2151);
        byte[] result = b.array();
        for (int i = 0; i < 4; i++)
            System.out.printf("%x\n", result[i]);
    }
}
```



```
public final ByteBuffer order(ByteOrder  bo)

    Modifies this buffer's byte order.

    Parameters:
        bo - The new byte order, either BIG_ENDIAN or LITTLE_ENDIAN
    Returns:
        This buffer
```




```
public static int
byteArrayToInt(byte[] b) {
    return   b[3] & 0xFF |
            (b[2] & 0xFF) << 8 |
            (b[1] & 0xFF) << 16 |
            (b[0] & 0xFF) << 24;
}
```

二进制编码常用技巧

Expert Java
Leader.us

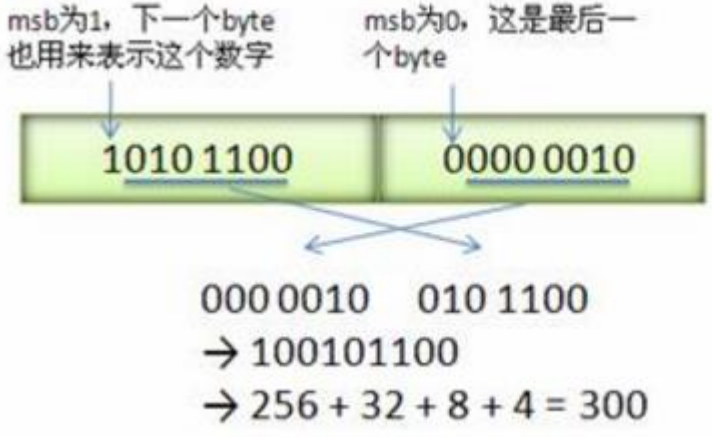
- 判断奇偶，二进制最后一位是0还是1，可以这么写：(a & 1) == 0
- 取一个整数的符号位，int i=a>>31，如果i=0则表示正数，-1则表示负数
- 一个负数的绝对值是取反加一，即：(~a + 1)
- 判断符号数是否相同(a ^ b) > 0

数据编码之Protocal Buffer

Expert Java Leader.us

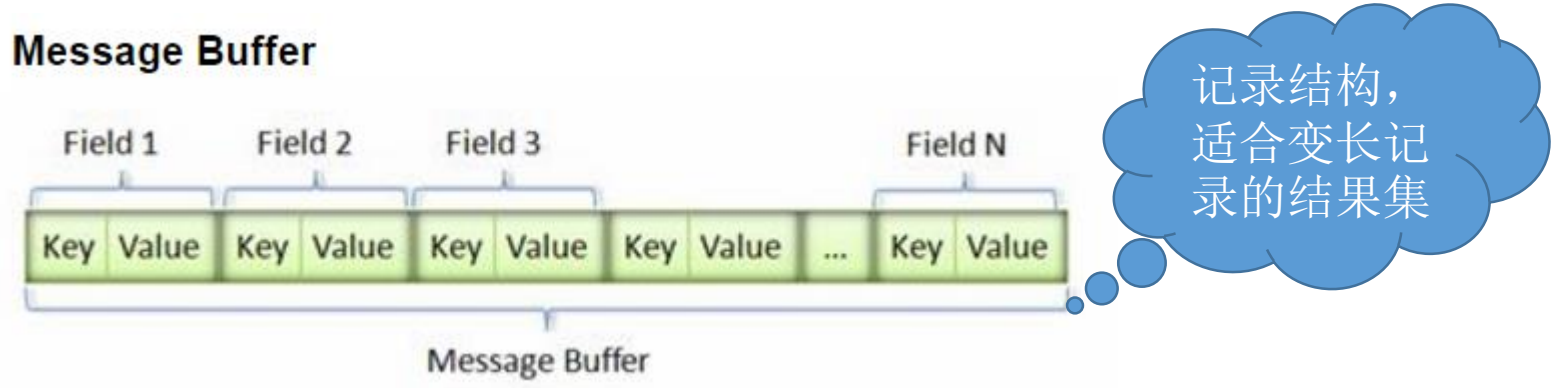
数字编码格式——Varint

用一个或多个字节来表示一个数字，值越小的数字使用越少的字节数。对于 int32 类型的数字，一般需要 4 个 byte 来表示。但是采用 Varint，对于很小的 int32 类型的数字，则可以用 1 个 byte 来表示。当然凡事都有好的也有不好的一面，采用 Varint 表示法，大的数字则需要 5 个 byte 来表示。从统计的角度来说，一般不会所有的消息中的数字都是大数，因此大多数情况下，采用 Varint 后，可以用更少的字节数来表示数字信息。



Protocol Buffer 字节序采用 little-endian 的方式

字符串等则采用类似数据库中的 varchar 的表示方法，即用一个 varint 表示长度，然后将其余部分紧跟在这个长度部分之后即可。



Key 用来标识具体的 field，在解包的时候，Protocol Buffer 根据 Key 就可以知道相应的 Value 应该对应于消息中的哪一个 field。Key的定义如下：

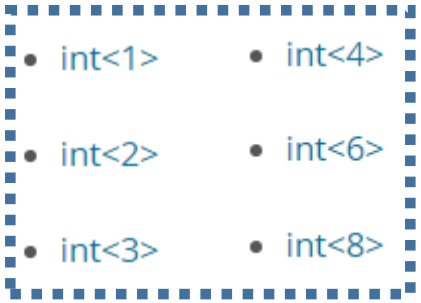
(field_number << 3) | wire_type

MySQL如何编码数据

Expert Java Leader.us

The MySQL Protocol has a set of possible encodings for integers:

- Fixed-length integers
- Length-encoded integers



Protocol::LengthEncodedInteger

An integer that consumes 1, 3, 4, or 9 bytes, depending on its numeric value

To convert a number value into a length-encoded integer:

- If the value is < 251, it is stored as a 1-byte integer.
- If the value is ≥ 251 and < (2^{16}), it is stored as `fc` + 2-byte integer.
- If the value is ≥ (2^{16}) and < (2^{24}), it is stored as `fd` + 3-byte integer.
- If the value is ≥ (2^{24}) and < (2^{64}) it is stored as `fe` + 8-byte integer.

14.1.1.2 String Types

Strings are sequences of bytes and appear in a few forms in the protocol.

Protocol::FixedLengthString

Fixed-length strings have a known, hardcoded length.

An example is the `sql-state` of the `ERR_Packet` which is always 5 bytes long.

Implemented By

string<fix>

Protocol::VariableLengthString:

The length of the string is determined by another field or is calculated at runtime

Protocol::LengthEncodedString

A length encoded string is a string that is prefixed with length encoded integer describing the length of the string.

It is a special case of `Protocol::VariableLengthString`

- If it is < 0xfb, treat it as a 1-byte integer.
- If it is 0xfc, it is followed by a 2-byte integer.
- If it is 0xfd, it is followed by a 3-byte integer.
- If it is 0xfe, it is followed by a 8-byte integer.

如何解析MySQL数据？

Expert Java Leader.us

io.mycat.mysql.packet. MySQLMessage

Fixed-Length Integers

```
public int readUB2() {
    int i = read() & 0xff;
    i |= (read() & 0xff) << 8;
    return i;
}

public int readUB3() {
    int i = read() & 0xff;
    i |= (read() & 0xff) << 8;
    i |= (read() & 0xff) << 16;
    return i;
}

public long readUB4() {
    long l = (long) (read() & 0xff);
    l |= (long) (read() & 0xff) << 8;
    l |= (long) (read() & 0xff) << 16;
    l |= (long) (read() & 0xff) << 24;
    return l;
}
```

Length Encoded Integers

```
public long readLength() {
    int length = read() & 0xff;
    switch (length) {
    case 251:
        return NULL_LENGTH;
    case 252:
        return readUB2();
    case 253:
        return readUB3();
    case 254:
        return readLong();
    default:
        return length;
    }
}
```

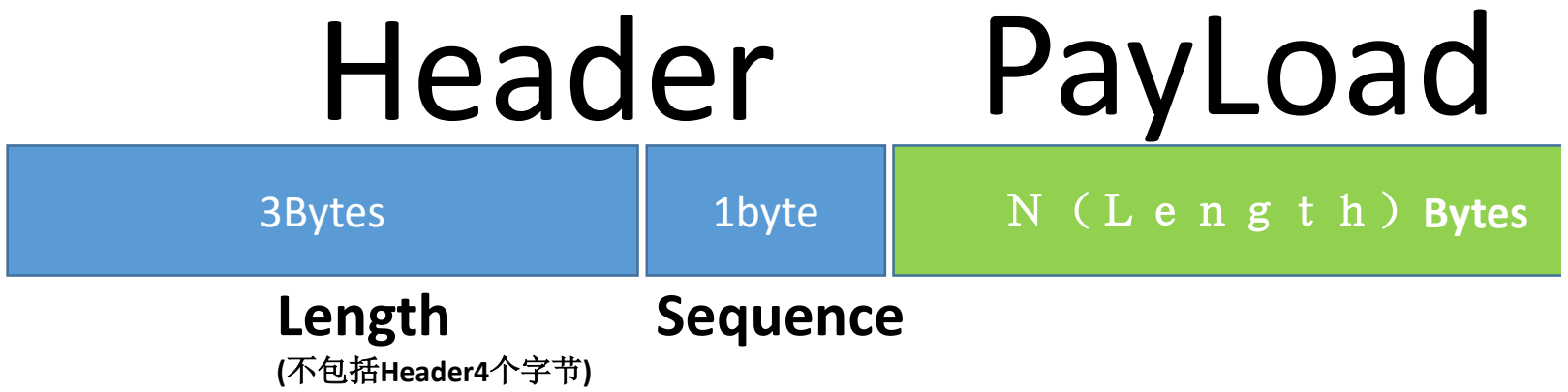
Length Encoded String

```
public String readStringWithLength(String charset) throws UnsupportedEncodingException {
    int length = (int) readLength();
    if (length <= 0) {
        return null;
    }
    byte[] ab = new byte[length - this.position()];
    this.byteBuffer.get(ab);
    String s = new String(ab,charset);
    return s;
}
```

MySQL的报文格式

Expert Java Leader.us

https://dev.mysql.com/doc/internals/en/mysql-packet.html MySQL Packet



A COM_QUIT looks like this:

01 00 00 00 01	* length: 1 * sequence_id: x00 * payload: 0x01

- 报文最大长度16M
- 当传输的长度超过１６M时，Length为2^24-1
  直到最后一个报文的Length<2^24-1结束
- Sequence从０开始，循环使用

MySQL的报文解析

Expert Java Leader.us

io.mycat.front.MySQLFrontConnectionHandler

```
public void handleReadEvent(final MySQLFrontConnection cnxn) throws IOException{
    LOGGER.debug("handleReadEvent(): {}", cnxn);
    final ConDataBuffer buffer = cnxn.getReadDataBuffer();
    int offset = buffer.readPos(), limit = buffer.writingPos();
    // 读取到了包头和长度
    // 是否讀完一個報文
    for(; ; ) {
        if(!MySQLConnection.validateHeader(offset, limit)) {
            LOGGER.debug("C#{}B#{} validate protocol packet header: too short, ready to handle next the read event",
                cnxn.getId(), buffer.hashCode());
            return;
        }
        int length = MySQLConnection.getPacketLength(buffer, offset);
        if((length + offset) > limit) {
             LOGGER.debug("C#{}B#{} nNot a whole packet: required length = {} bytes, cur total length = {} bytes, "
                + "ready to handle the next read event", cnxn.getId(), buffer.hashCode(), length, limit);
            return;
        }
        if(length == 4){
            // @todo handle empty packet
        }
        // 解析报文类型
        final byte packetType = buffer.getByte(offset + MySQLConnection.msyql_packetHeaderSize);
        final int pkgStartPos = offset;
```

- 定位报文
- 处理粘包问题
- 判断报文类型

MySQL报文之OK Packet

Expert Java Leader.us

https://dev.mysql.com/doc/internals/en/packet-OK_Packet.html

Type	Name	Description
int<1>	*header*	[00] or [fe] the OK packe
int<lenenc>	*affected_rows*	affected rows
int<lenenc>	*last_insert_id*	last insert-id
if capabilities & CLIENT_PROTOCOL_41 {		
int<2>	*status_flags*	Status Flags
int<2>	*warnings*	number of warnings
} elseif capabilities & CLIENT_TRANSACTIONS {		
int<2>	*status_flags*	Status Flags
}		
if capabilities & CLIENT_SESSION_TRACK {		
string<lenenc>	*info*	human readable status information
if status_flags & SERVER_SESSION_STATE_CHANGED {		
string<lenenc>	*session_state_changes*	session state info
}		
} else {		
string<EOF>	*info*	human readable status informati
}		

```
public void read(byte[] data) {
    MySQLMessage mm = new MySQLMessage(data);
    packetLength = mm.readUB3();
    packetId = mm.read();
    header = mm.read();
    affectedRows = mm.readLength();
    insertId = mm.readLength();
    serverStatus = mm.readUB2();
    warningCount = mm.readUB2();
    if (mm.hasRemaining()) {
        this.message = mm.readBytesWithLength();
    }
}
```

Name	Value
SESSION_TRACK_SYSTEM_VARIABLES	0x00
SESSION_TRACK_SCHEMA	0x01
SESSION_TRACK_STATE_CHANGE	0x02
SESSION_TRACK_GTIDS	0x03

Flag	Value
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

OK Packet替换EOF报文

Expert Java Leader.us

As of MySQL 5.7.5, OK packes are also used to indicate EOF, and EOF packets are deprecated.

To ensure backward compatibility between old (prior to 5.7.5) and new (5.7.5 and up) versions of MySQL, new clients advertise the `CLIENT_DEPRECATE_EOF` flag:

- Old clients do not know about this flag and do not advertise it. Consequently, the server does not send OK packets that represent EOF. (Old servers never do this, anyway. New servers recognize the absence of the flag to mean they should not.)
- New clients advertise this flag. Old servers do not know this flag and do not send OK packets that represent EOF. New servers recognize the flag and can send OK packets that represent EOF.

如何区分OK报文与EOF报文

OK: header = 0 and length of packet > 7

EOF: header = 0xfe and length of packet < 9

MySQL报文之EOF报文

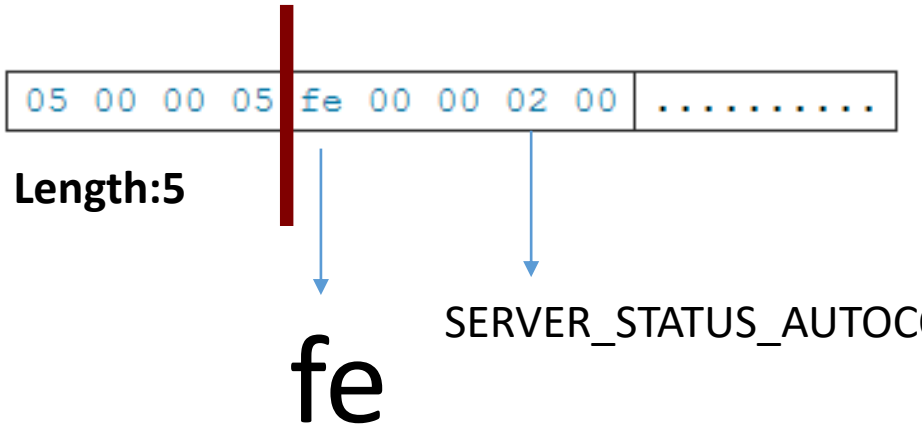
Expert Java Leader.us

In the MySQL client/server protocol, EOF and OK packets serve the same purpose, to mark the end of a query execution result.The EOF packet may appear in places where a Protocol::LengthEncodedInteger may appear. You must check **whether the packet length is less than 9** to make sure that it is a EOF packet.

Payload

Type	Name	Description
int<1>	*header*	[fe] EOF header
if capabilities & CLIENT_PROTOCOL_41 {		
int<2>	*warnings*	number of warnings
int<2>	*status_flags*	Status Flags
}		

A MySQL 4.1 EOF packet with: 0 warnings, AUTOCOMMIT enabled.



MySQL报文之ERR报文

Expert Java
Leader.us

This packet signals that an error occurred. It contains a SQL state value if CLIENT_PROTOCOL_41 is enabled.

Payload

Type	Name	Description
int<1>	*header*	[ff] header of the ERR packet
int<2>	*error_code*	error-code
if capabilities & CLIENT_PROTOCOL_41 {		
string[1]	*sql_state_marker*	# marker of the SQL State
string[5]	*sql_state*	SQL State
}		
string<EOF>	*error_message*	human readable error message

Example

```
17 00 00 01 ff 48 04 23    48 59 30 30 30 4e 6f 20  .....H.#HY000No
74 61 62 6c 65 73 20 75    73 65 64                 tables used
```

String<EOF> If a string is the last component of a packet, its length can be calculated from the overall packet length minus the current position.

MySQL Proxy的两种Connection

前后端连接处理逻辑完全不一样

传说中的雌雄同体

Client

FrontConnection

Mycat作为MySQL Server端，来处理Client请求。

MyCat

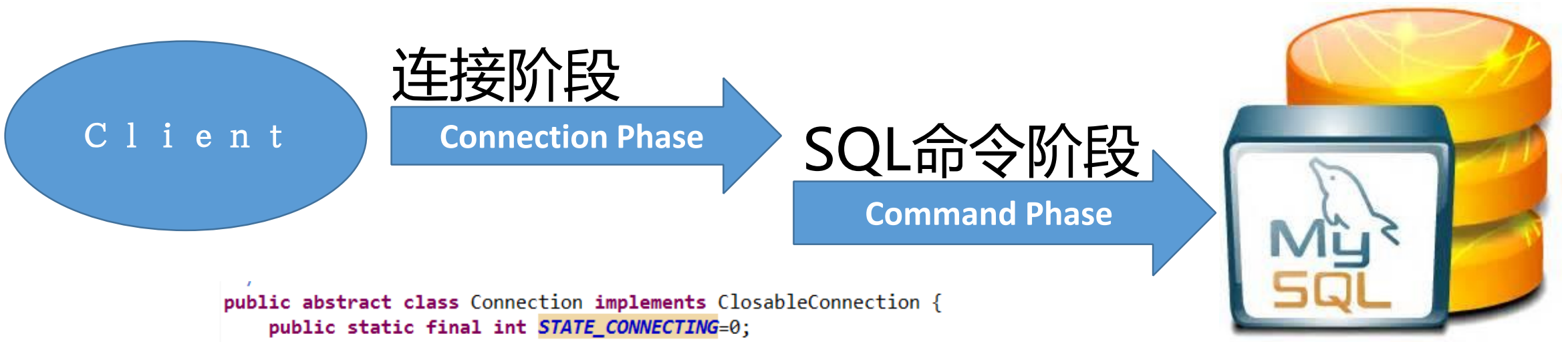
BackendConnection

Mycat作为MySQL Client端，发送请求到MySQL Server，并处理应答

Expert Java Leader.us

MySQL Connection的状态

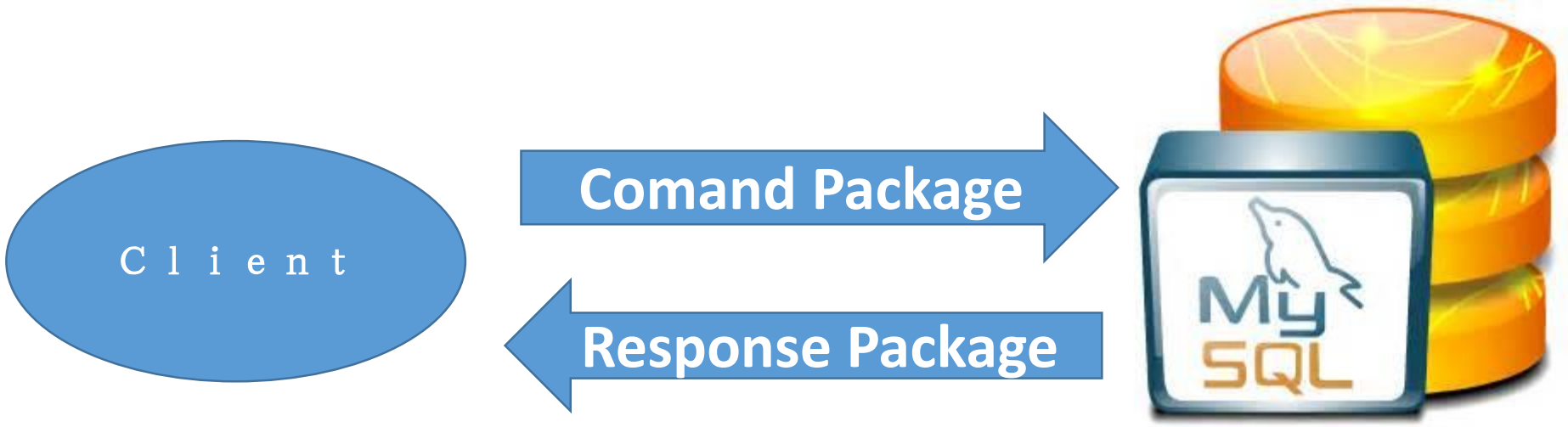
Expert Java Leader.us




```
public abstract class Connection implements ClosableConnection {
    public static final int STATE_CONNECTING=0;
    public static final int STATE_IDLE=1;
    public static final int STATE_CLOSING=-1;
    public static final int STATE_CLOSED=-2;

    public static Logger LOGGER = LoggerFactory.getLogger(Connection.class);
    protected String host;
    protected int port;
    protected int localPort;
    protected long id;
    private String  reactor;
    private Object attachement;
    private int state = STATE_CONNECTING;
```

MySQL的请求/应答模型

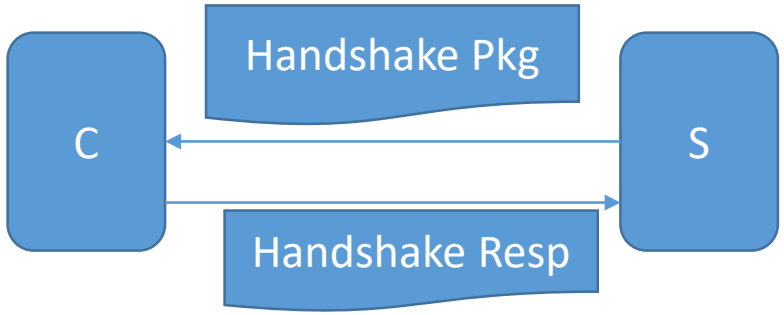
Expert Java
Leader.us



- 一次请求对应一个应答
- 一次请求可能是多个报文（load data）
- 一次请求返还的报文可能是多个（Result set)
- 一次请求结束之前，不能有第二个请求
- 请求的应答时间可能很长

MySQL的连接建立过程

Expert Java Leader.us



1. server sending `Initial Handshake Packet`

2. client replying with `Handshake Response Packet`

Capability Negotiation

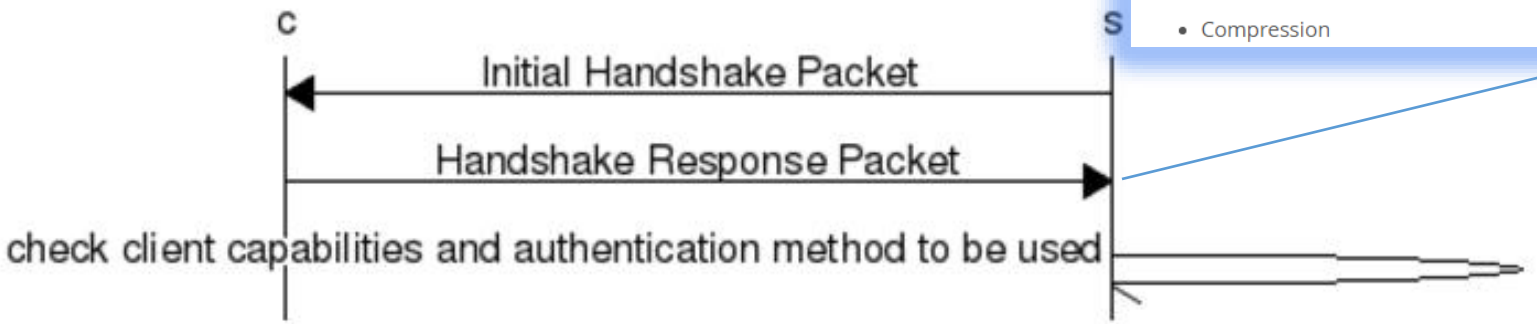
To permit an old client to connect to newer servers, the initial handshake contains

- the MySQL Server version
- the server's `capabilities`

The client should only announce the capabilities in the Handshake Response Packet that it has in common with the server.

They can agree on:

- use of status flags
- use of SQL states for error-codes
- authentication methods
- SSL support
- Compression



Secure Password Authentication

- client-side expects a 20-byte random challenge
- client-side returns a 20-byte response based on the algorithm described later

The password is calculated by:

```
SHA1( password ) XOR SHA1( "20-bytes random data from server" <concat> SHA1( SHA1( password ) ) )
```

MySQL服务端握手代码

Expert Java Leader.us

io.mycat.front.MySQLFrontConnectionHandler

```java
@Override
public void onConnected(MySQLFrontConnection con) throws IOException {
    LOGGER.debug("onConnected(): {}", con);
    con.getSession().changeCmdHandler(loginCmdHandler);
    con.sendAuthPackge();
}
```

```java
public void sendAuthPackge() throws IOException {
    // 生成认证数据
    byte[] rand1 = RandomUtil.randomBytes(8);
    byte[] rand2 = RandomUtil.randomBytes(12);

    // 保存认证数据
    byte[] seed = new byte[rand1.length + rand2.length];
    System.arraycopy(rand1, 0, seed, 0, rand1.length);
    System.arraycopy(rand2, 0, seed, rand1.length, rand2.length);
    this.seed = seed;

    // 发送握手数据包
    HandshakePacket hs = new HandshakePacket();
    hs.packetId = 0;
    hs.protocolVersion = Versions.PROTOCOL_VERSION;
    hs.serverVersion = Versions.SERVER_VERSION;
    hs.threadId = id;
    hs.seed = rand1;
    hs.serverCapabilities = getServerCapabilities();
    // hs.serverCharsetIndex = (byte) (charsetIndex & 0xff);
    hs.serverStatus = 2;
    hs.restOfScrambleBuff = rand2;
    this.writeMsqlPackage(hs);
    // asynread response
    // this.asynRead();
}
```

```
1               [0a] protocol version
string[NUL]     server version
4               connection id
string[8]       auth-plugin-data-part-1
1               [00] filler
2               capability flags (lower 2 bytes)
  if more data in the packet:
1               character set
2               status flags
2               capability flags (upper 2 bytes)
  if capabilities & CLIENT_PLUGIN_AUTH {
1               length of auth-plugin-data
  } else {
1               [00]
  }
string[10]      reserved (all [00])
  if capabilities & CLIENT_SECURE_CONNECTION {
string[$len]    auth-plugin-data-part-2 ($len=MAX(13, length of auth-plugin-data - 8))
  if capabilities & CLIENT_PLUGIN_AUTH {
string[NUL]     auth-plugin name
  }
```

MySQL服务端握手代码 2

Expert Java Leader.us

io.mycat.front. CheckUserLoginResponseCallback

处理客户端发送过来的Auth　Packet

io.mycat.util.SecurityUtil

```
public void processCmd(MySQLFrontConnection con, ConDataBuffer dataBuffer, byte packageType, int pkgStartPos,
        int pkgLen) throws IOException {
    // check quit packet
    if (packageType == MySQLPacket.QUIT_PACKET) {
        con.close("quit packet");
        return;
    }
    ByteBuffer byteBuff = dataBuffer.getBytes(pkgStartPos, pkgLen);
    AuthPacket auth = new AuthPacket();
    auth.read(byteBuff);

    // Fake check user
    LOGGER.debug("Check user name. " + auth.user);
    if (!auth.user.equals("root")) {
        LOGGER.debug("User name error. " + auth.user);
        con.failure(ErrorCode.ER_ACCESS_DENIED_ERROR, "Access denied for user '" + auth.user
        return;
    }

    // Fake check password
    LOGGER.debug("Check user password. " + new String(auth.password));

    // check schema
    LOGGER.debug("Check database. " + auth.database);

    success(con, auth);

}
```

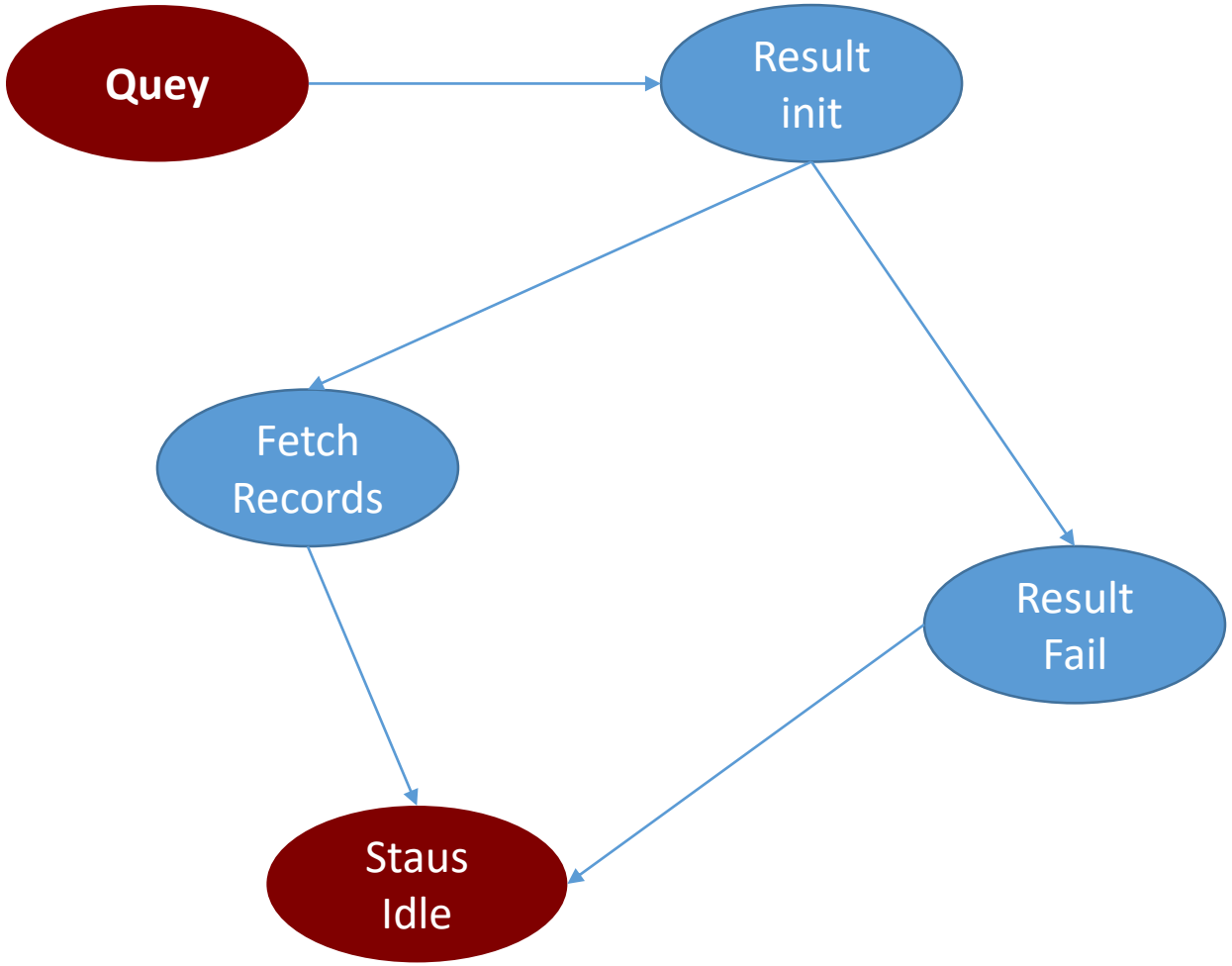
```
private void success(MySQLFrontConnection con, AuthPacket auth) throws IOException {
    LOGGER.debug("Login success");
    // 设置字符集编码
    int charsetIndex = (auth.charsetIndex & 0xff);
    final String charset = CharsetUtil.getCharset(charsetIndex);
    if (charset == null) {
        final String errmsg = "Unknown charsetIndex:" + charsetIndex;
        LOGGER.warn(errmsg);
        con.writeErrMessage(ErrorCode.ER_UNKNOWN_CHARACTER_SET, errmsg);
        return;
    }
    LOGGER.debug("charset = {}, charsetIndex = {}", charset, charsetIndex);
    con.setCharset(charsetIndex, charset);

    //认证成功后，修改changeCmdHandler，由CheckUserLoginResponseCallback改用
    // AbstractSchemaSQLCommandHandler处理
    if (!con.setFrontSchema(auth.database)) {
        final String errmsg = "No Mycat Schema defined: " + auth.database;
        LOGGER.debug(errmsg);
        con.writeErrMessage(ErrorCode.ER_NO_DB_ERROR, errmsg);
    } else {
        con.write(AUTH_OK);
    }
    con.setState(Connection.STATE_IDLE);

}
```

MySQL Comand Phase

Expert Java Leader.us

```
case STATE_IDLE:
    if (packetType == MySQLPacket.COM_QUERY) {
        this.setState(CMD_QUERY_STATUS);
    } else if (packetType == MySQLPacket.COM_QUIT) {
        this.setState(STATE_CLOSING);
    }
    break;
case CMD_QUERY_STATUS:
    if (packetType == MySQLPacket.OK_PACKET) {
        this.setState(RESULT_INIT_STATUS);
    } else if (packetType == MySQLPacket.ERROR_PACKET) {
        this.setState(STATE_IDLE);
    }
    break;
case RESULT_INIT_STATUS:
    if (packetType == MySQLPacket.OK_PACKET) {
        this.setState(RESULT_FETCH_STATUS);
    } else if (packetType == MySQLPacket.ERROR_PACKET) {
        this.setState(RESULT_FAIL_STATUS);
    }
    break;
case RESULT_FETCH_STATUS:
    if (packetType == MySQLPacket.EOF_PACKET) {
        this.setState(STATE_IDLE);
    } else if (packetType == MySQLPacket.ERROR_PACKET) {
        this.setState(RESULT_FAIL_STATUS);
    }
    break;
case RESULT_FAIL_STATUS:
    if (packetType == MySQLPacket.EOF_PACKET) {
        this.setState(STATE_IDLE);
    }
    break;
```

MySQLFrontConnection



Command Of Text Protocol

In the command phase, the client sends a command packet with the sequence-id [00]:

- COM_SLEEP
- COM_QUIT
- COM_INIT_DB
- COM_QUERY
- COM_FIELD_LIST
- COM_CREATE_DB
- COM_DROP_DB
- COM_REFRESH
- COM_SHUTDOWN
- COM_STATISTICS
- COM_PROCESS_INFO
- COM_CONNECT
- COM_PROCESS_KILL
- COM_DEBUG
- COM_PING
- COM_TIME
- COM_DELAYED_INSERT
- COM_CHANGE_USER
- COM_RESET_CONNECTION
- COM_DAEMON

A COM_QUERY is used to send the server a text-based query that is executed immediately.

Payload

```
1              [03] COM_QUERY
string[EOF]    the query the server shall execute
```

Fields

- **command_id** (*1*) -- 0x03 COM_QUERY
- **query** (*string.EOF*) -- query_text

COM_QUERY_Response:

- a ERR_Packet
- a OK_Packet
- a Protocol::LOCAL_INFILE_Request
- a ProtocolText::Resultset

14.6.4.1.1 Text Resultset

14.6.4.1.1.1 Column Type
14.6.4.1.1.2 Column Definition
14.6.4.1.1.3 Text Resultset Row

MySQL ResultSet

Expert Java Leader.us

col1	col2	col3	col4

包含字段个数信息的报文 + N个描述字段Metadata信息的报文 → EOF

ResultsetRow报文（N个字段）

ResultsetRow报文（N个字段）

ResultsetRow报文（N个字段）

OK

ProtocolText::Resultset:

- A packet containing a `Protocol::LengthEncodedInteger` `column_count`
- `column_count` * `Protocol::ColumnDefinition` packets
- If the `CLIENT_DEPRECATE_EOF` client capability flag is not set, `EOF_Packet`
- One or more `ProtocolText::ResultsetRow` packets, each containing `column_count` values
- `ERR_Packet` in case of error. Otherwise: If the `CLIENT_DEPRECATE_EOF` client capability flag is set, `OK_Packet`; else `EOF_Packet`.

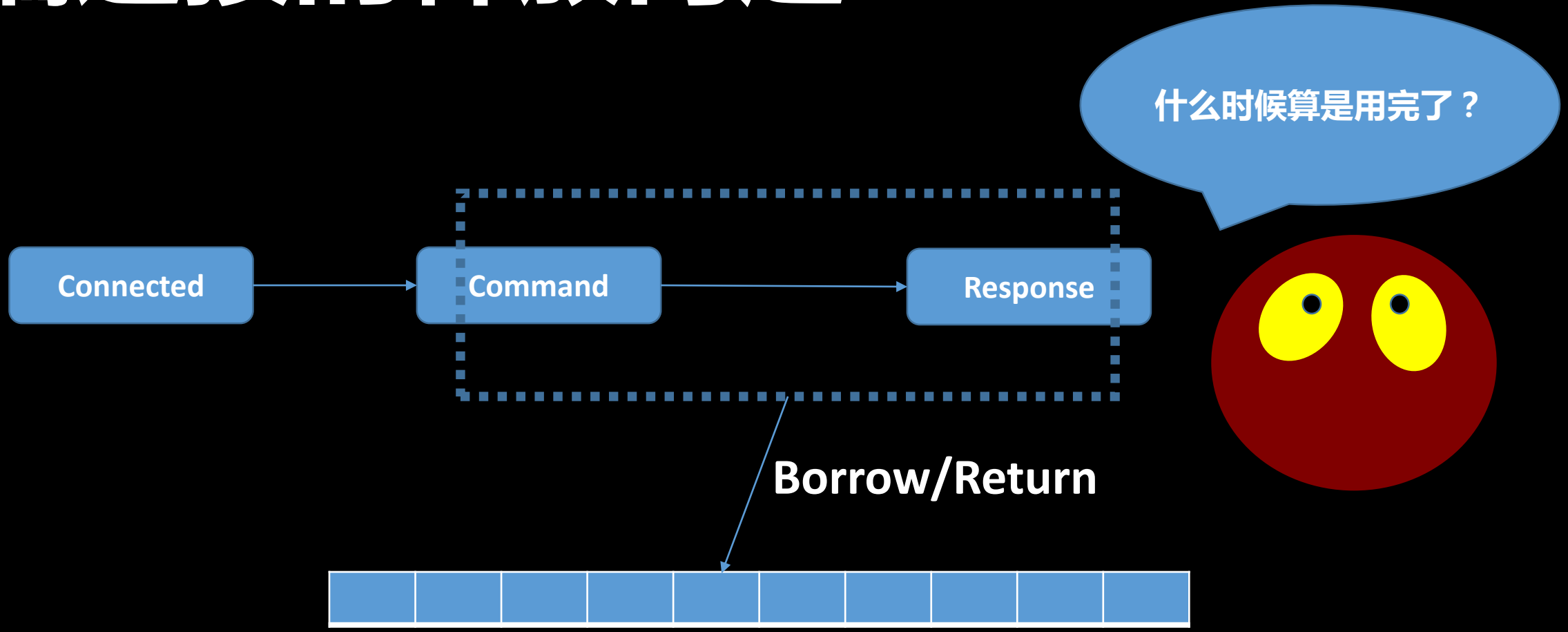
If the `SERVER_MORE_RESULTS_EXISTS` flag is set in the last `EOF_Packet` or (if the `CLIENT_DEPRECATE_EOF` capability flag is set) `OK_Packet`, another `ProtocolText::Resultset` will follow (see *Multi-resultset*).

Expert Java
Leader.us

Mycat的第一个难题.....

Expert Java
Leader.us

后端连接的释放问题



Expert Java
Leader.us

To Be Continued