

# Mysql Binlog 日志分析

C++实现日志解析

## 目录

一、Redo log 和 Binlog 区别.....	2
二、Bin log 基本介绍.....	3
三、Bin log 格式分析.....	4
1. format_desc Event （对应枚举变量 15）.....	5
2. QUERY_EVENT （对应枚举变量 2）.....	6
(1) Fixed part 部分（13 字节）:.....	6
(2) Variable part(变长部分):.....	6
3. TABLE_MAP_EVENT （对应 枚举变量 19）.....	7
(1) Fixed part:.....	7
(2) Variable part:.....	7
4. WRITE_ROWS_EVENT/DELETE_ROWS_EVENT/UPDATE_ROWS_EVENT （分别为 30 31 32）.....	9
(1) Fixed part:.....	9
(2) Variable part:.....	9
5. XID_EVENT （对应 16）.....	10
四、代码分析.....	11
1. 如何编译.....	11
2. 如何使用与存在的问题.....	11
3. 源码分析.....	11
五、参考资料.....	21

## 一、Redolog 和 Binlog 区别

redo log 不是二进制日志 (Bin log)。虽然二进制日志中也记录了 innodb 表的很多操作，**也能实现重做的功能**，但是它们之间有很大区别。

二进制日志是在**存储引擎的上层**产生的，不管是什么存储引擎，对数据库进行了修改都会产生二进制日志。而 redo log 是 innodb 层产生的，只记录该存储引擎中表的修改。改**并且二进制日志 (Bin log) 先于 redo log 被记录**。

下面来看二者的区别：

1. 二进制日志记录操作的方法是逻辑性的语句。即便它是基于行格式的记录方式，其本质也还是逻辑的 SQL 设置，如该行记录的每列的值是多少。而 redo log 是在物理格式上的日志，它记录的是数据库中每个页的修改。
2. 二进制日志只在每次事务提交的时候一次性写入缓存中的日志"文件"。而 redo log 在数据准备修改前写入缓存中的 redo log 中，然后才对缓存中的数据执行修改操作；而且保证在发出事务提交指令时，先向缓存中的 redo log 写入日志，写入完成后才执行提交动作。
3. 因为二进制日志只在提交的时候一次性写入，所以二进制日志中的记录方式和提交顺序有关，且一次提交对应一次记录。而 redo log 中是记录的物理页的修改，redo log 文件中同一个事务可能多次记录，最后一个提交的事务记录会覆盖所有未提交的事务记录。例如事务 T1，可能在 redo log 中记录了 T1-1, T1-2, T1-3, T1\* 共 4 个操作，其中 T1\* 表示最后提交时的日志记录，所以对应的数据页最终状态是 T1\* 对应的操作结果。而且 redo log 是并发写入的，不同事务之间的不同版本的记录会穿插写入到 redo log 文件中，例如可能 redo log 的记录方式如下：T1-1, T1-2, T2-1, T2-2, T2\*, T1-3, T1\*。
4. 事务日志记录的是物理页的情况，它具有幂等性，因此记录日志的方式及其简练。幂等性的意思是多次操作前后状态是一样的，例如新插入一行后又删除该行，前后状态没有变化。而二进制日志记录的是所有影响数据的操作，记录的内容较多。例如插入一行记录一次，删除该行又记录一次。

本节介绍了 Redo log 和 Bin log 的主要区别，Mysql 的二进制日志适用于所有 Mysql 引擎，并且记录了所有的 DDL 和 DML (除数据查询语句)，分析 Bin log 的意义大于 Redo log。

## 二、Bin log 基本介绍

### (一) 什么是 Binlog

MySQL 的二进制日志可以说是 MySQL 最重要的日志了，它记录了所有的 DDL 和 DML (除了数据查询语句) 语句，以事件形式记录，还包含语句所执行的消耗的时间，MySQL 的二进制日志是事务安全型的。

一般来说开启二进制日志大概会有 1% 的性能损耗 (参见 MySQL 官方中文手册 5.1.24 版)。二进制日志有两个最重要的使用场景：

其一：MySQL Replication 在 Master 端开启 binlog，Master 把它的二进制日志传递给 slaves 来达到 master-slave 数据一致的目的。

其二：自然就是数据恢复了，通过使用 mysqlbinlog 工具来使恢复数据。

二进制日志包括两类文件：二进制日志索引文件（文件名后缀为 .index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为 .000000\*）记录数据库所有的 DDL 和 DML (除了数据查询语句) 语句事件。

mysql-bin.000001	2018/7/9 13:42	000001 文件
mysql-bin.index	2018/7/9 11:48	INDEX 文件

图 1 windows 系统里的二进制日志文件

### (二) 开启 Binlog

通常情况 MySQL 是默认关闭 Binlog 的，所以你得配置一下以启用它。

启用的过程就是修改配置文件 my.cnf 了，至于 my.cnf 位置请自行寻找。

```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
# 设置mysql客户端默认字符集
default-character-set=utf8
[mysqld]
# 设置3306端口
port = 3306
# 设置mysql的安装目录
basedir=F:\软件\mysql-5.6.24-win32.1432006610\mysql-5.6.24-win32
# 设置mysql数据库的数据的存放目录
datadir=F:\软件\mysql-5.6.24-win32.1432006610\mysql-5.6.24-win32\data
# 二进制日志
log-bin=mysql-bin
binlog-format=Row
# 允许最大连接数
max_connections=200
# 服务端使用的字符集默认为8比特编码的latin1字符集
character-set-server=utf8
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
```

图 2 windows 下修改 Mysql 配置文件

配置 log-bin 和 log-bin-index 的值，如果没有则自行加上。这里的 log-bin 是指以后生成各 Binlog 文件的前缀，比如上述使用 master-bin，那么文件就将会是 master-bin.000001、master-bin.000002 等。而这里的 log-bin-index 则指 binlog index 文件的名称，这里我们设置为 master-bin.index。

配置完成后重启 Mysql 服务。之后可以进入 Mysql 验证是否真的启用了 Binlog。

```
mysql> show variables like '%log_bin%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_bin       | ON    |
| log_bin_basename | F:\   |
| log_bin_index  | F:\   |
| log_bin_trust_function_creators | OFF   |
| log_bin_use_v1_row_events | OFF   |
| sql_log_bin    | ON    |
+-----+-----+
6 rows in set, 2 warnings (0.00 sec)
```

图 3 windows 下 Mysql 启用 Binlog 的情况

### 三、Bin log 格式分析

mysql 5.0 之后 binlog 都采用的 v4 版本，结构如下分为 header 和 data 两部分，header 部分所有 event 都一样占用 19bytes：

```
+-----+
| event | timestamp 0 : 4 | #当前 event 写入时的时间
| header +-----+
|       | type_code 4 : 1 | #当前 event 的类型 ID
|       +-----+
|       | server_id 5 : 4 |
|       +-----+
|       | event_length 9 : 4 | #当前 event 总字节数
|       +-----+
|       | next_position 13 : 4 | #下一个 event 开始的 position
|       +-----+
|       | flags 17 : 2 | #标签
|       +-----+
|       | extra_headers 19 : x-19 |
+-----+
```

event	fixed part	x : y
data	+-----+	
	variable part	
+=====+		

Binlog 文件是二进制文件，由一个一个的 event 组成，每个对数据变动的操作以及 DDL 语句都会产生一系列的 event。详细了解参考 Mysql 官方源码：<https://dev.mysql.com/doc/internals/en/event-classes-and-types.html>

这里分析几个和解析 Binlog 文件有关的几个重要的 Event：

### (1) format\_desc Event （对应枚举变量 15）

可以看出 format\_desc 开始的 pos 位置为 4，这是因为每个 binlog 文件开头都会占用一个固定的 4bytes，编码为\xfe\x62\x69\x6e，现在来开始对他进行解析。

```
mysql> show binlog events;
```

Log_name	Pos	Event_type	Server_id	End_log_pos	Info
mysql-bin.000001	4	Format_desc	1	120	Server ver: 5.6.24-log, Binlog ver: 4

1	fe62	696e	08db	425b	0f01	0000	0074	0000
2	0078	0000	0001	0004	0035	2e36	2e32	342d
3	6c6f	6700	0000	0000	0000	0000	0000	0000
4	0000	0000	0000	0000	0000	0000	0000	0000
5	0000	0000	0000	0000	0000	0008	db42	5b13
6	380d	0008	0012	0004	0404	0412	0000	5c00
7	041a	0800	0000	0808	0802	0000	000a	0a0a
8	1919	0001	d45a	7287	fadb	425b	0201	0000
9	0098	0100	0010	0200	0000	0004	0000	0000

Format\_desc event data 部分的 fixed part 格式分别为

2bytes 记录 binlog version

50bytes 记录 MySQL server version

4bytes 记录 binlog 文件创建时间

1bytes 值为 19, 是所有 event 的 header 长度

下面是最开始分析时, 用 python3 做的解析, 可以看到上面每个字节对应的信息。

```
UnicodeDecodeError: 'gbk' codec can't decode byte 0xfe
>>> datafile = open('G:\mysql-bin.000001', 'rb')
>>> a = datafile.read(4)
>>> a = datafile.read(19)
>>> print(struct.unpack('=IBIIH', a))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'struct' is not defined
>>> import struct
>>> print(struct.unpack('=IBIIH', a))
(1531108104, 15, 1, 116, 120, 1)
>>> a = datafile.read(2)
>>> print(struct.unpack('H', a))
(4,)
>>> a = datafile.read(50)
>>> print(a.decode('utf8'))
5.6.24-log
>>> a = datafile.read(4)
>>> print(struct.unpack('I', a))
(1531108104,)
>>> a = datafile.read(1)
>>> print(struct.unpack('B', a))
(19,)
>>>
```

这里的时间戳解析出来的 1531108104, 代表了这个时间距离 1970 年的秒数, 可以在 linux 里用 `date -d '@1531108104'` 得到我们能理解的年月日的时间。

## (2) QUERY\_EVENT (对应枚举变量 2)

### ① Fixed part 部分 (13 字节):

Thread\_id: 4bytes 产生数据的线程 ID, 可以可以用于 DBA 审计

Execute\_time: 4bytes 该语句执行时间, 单位秒

Databas\_length: 1bytes 库名占字节长度

Error\_code: 2bytes 错误代码, 一般该值都为 0, 比如在 master 上执行 `inser...select...` 语句时 myisam 表出现主键冲突或者 innodb 表执行途中 `ctrl+c` 退出就会记录该值, 在 slave 执行时会检查报错退出

Variable\_block\_length: 2bytes 记录 data part 部分 variable status 的长度 (这部分很重要)

### ② Variable part (变长部分):



Variable\_status : Variable\_block\_length, 每个 variable 对应一个值, 值是紧跟 variable 后面, 这里记录 charset、sqlmode、auto\_increment 等的情况, 自增只有偏移大于 1 的时候才记录 (基本没用)

Database\_name: Databas\_length

Sql\_statement: 整个 event 剩余部分 (还有 4 字节的校验和)

### (3) TABLE\_MAP\_EVENT (对应 枚举变量 19)

#### ① Fixed part:

Table\_id : 6bytes (关键 利用这个得到表的 ID 关联后面解析出的数据库名+表名)

Reserved: 2bytes 预留位置

#### ② Variable part:

Database\_name\_length: 1bytes

Database\_name: database\_name\_length + 1 个空字节

Table\_name\_length: 1bytes

Table\_name: table\_name\_length + 1 个空字节

#后面的内容 记录每个字段的类型

Columns: 1bytes 记录表字段数, 一般情况字段数是不会超过 255 所以占 1bytes

Columns\_type\_code: 记录每个字段类型 id, 每个字段占用 1 个子节点 (columns\*1), 该位记录的值主要对读取后面源数据提供标准, 顺序与表结构字段顺序一致

Metadata\_length: 1bytes 字段元数据占用字节长度, 同样一般不超过 1bytes

Metadata: 可变长度为 metadata\_length, 需根据 Columns\_type\_code 判断每个字段占用长度, 数字类型 (int, tinyint...) 都不会占用空间, 只有可变长度的才会占用, 比如 varchar、char、enum、binary 都占用 2bytes, text、blob、longtext 只占用 1bytes, 这里的元数据对后面读取 row 记录提供格式规范

Variable\_size: 该部分记录字段是否允许为空，一位代表一个字段，占用  $\lceil (N+7)/8 \rceil$  bytes, N 为字段数

Crc: 4bytes 最后 4 字节校验码

/\*Mysql 源码里的 所有 Type\*/

```
enum enum_field_types {  
    MYSQL_TYPE_DECIMAL, MYSQL_TYPE_TINY,  
    MYSQL_TYPE_SHORT, MYSQL_TYPE_LONG,  
    MYSQL_TYPE_FLOAT, MYSQL_TYPE_DOUBLE,  
    MYSQL_TYPE_NULL, MYSQL_TYPE_TIMESTAMP,  
    MYSQL_TYPE_LONGLONG, MYSQL_TYPE_INT24,  
    MYSQL_TYPE_DATE, MYSQL_TYPE_TIME,  
    MYSQL_TYPE_DATETIME, MYSQL_TYPE_YEAR,  
    MYSQL_TYPE_NEWDATE, MYSQL_TYPE_VARCHAR,  
    MYSQL_TYPE_BIT,  
    MYSQL_TYPE_TIMESTAMP2,  
    MYSQL_TYPE_DATETIME2,  
    MYSQL_TYPE_TIME2,  
    MYSQL_TYPE_JSON=245,  
    MYSQL_TYPE_NEWDECIMAL=246,  
    MYSQL_TYPE_ENUM=247,  
    MYSQL_TYPE_SET=248,  
    MYSQL_TYPE_TINY_BLOB=249,  
    MYSQL_TYPE_MEDIUM_BLOB=250,  
    MYSQL_TYPE_LONG_BLOB=251,  
    MYSQL_TYPE_BLOB=252,  
    MYSQL_TYPE_VAR_STRING=253,  
    MYSQL_TYPE_STRING=254,  
    MYSQL_TYPE_GEOMETRY=255  
};
```



Mysql 官网给出的源码里的字段类型对应的 C 代码里数据类型的转换关系：

Input Variable C Type	buffer_type Value	SQL Type of Destination Value
signed char	MYSQL_TYPE_TINY	<u>TINYINT</u>
short int	MYSQL_TYPE_SHORT	<u>SMALLINT</u>
int	MYSQL_TYPE_LONG	<u>INT</u>
long long int	MYSQL_TYPE_LONGLONG	<u>BIGINT</u>
float	MYSQL_TYPE_FLOAT	<u>FLOAT</u>
double	MYSQL_TYPE_DOUBLE	<u>DOUBLE</u>
MYSQL_TIME	MYSQL_TYPE_TIME	<u>TIME</u>
MYSQL_TIME	MYSQL_TYPE_DATE	<u>DATE</u>
MYSQL_TIME	MYSQL_TYPE_DATETIME	<u>DATETIME</u>
MYSQL_TIME	MYSQL_TYPE_TIMESTAMP	<u>TIMESTAMP</u>
char[]	MYSQL_TYPE_STRING	<u>TEXT, CHAR, VARCHAR</u>
char[]	MYSQL_TYPE_BLOB	<u>BLOB, BINARY, VARBINARY</u>
	MYSQL_TYPE_NULL	NULL

**备注：**时间戳格式的为 99 A0 65 50 41 B5 对应的数据应该是 2018-07-18 21:01:01，不知道怎么解析这样的数据，在源码里这样的时间戳对应的数据类型是 MYSQL\_TYPE\_TIMESTAMP2

#### (4) WRITE\_ROWS\_EVENT/DELETE\_ROWS\_EVENT/UPDATE\_ROWS\_EVENT （分别为 30 31 32）

##### ① Fixed part:

Table\_id : 6bytes

Reserved: 2bytes 预留位置

Extra: 2bytes 具体干嘛的不清楚，官方文档也没介绍有这 2bytes 内容

##### ② Variable part:

Columns: 1bytes 字段数

Variable\_size: 可变长度  $\text{int}((n+7)/8)$ , n 是字段数, bit 标识对应字段是否有值, 1 代表有, 0 代表没有, row 模式都是有值的

Variable\_size: 跟上面一个一样, 但是只有 update\_rows\_event 才有

Variable\_size: 跟上面两个一样计算长度, 该值的 bit 位标识后面所跟的行数据每个字段是否为 NULL, 为 NULL 时 bit 位为 0, 1 代表有值, 这个 bit 位和 table\_map\_event 的 columns\_type\_code 顺序对应

Value: 数据内容

Crc: 4bytes 校验码

## (5) XID\_EVENT (对应 16)

Fixed part 为空, 只有 variable part 占有 8bytes 的 xid 及结尾的 4bytes 校验码

要对 binlog 进行解析还需要了解数据存储详细占用情况, 同样的在官方文档有详细的介绍, 参考 <https://dev.mysql.com/doc/refman/5.7/en/storage-requirements.html>, 这里拿常用的几个字段类型来做介绍:

1. Varchar: 占用字节数 0-255bytes 使用 1bytes 记录长度, 超过 255bytes 时使用 2bytes 记录长度 + 数据
2. Int、tinyint、bigint: 分别占用 4bytes、1bytes、8bytes
3. Text: 使用 2 个字节记录长度 + 数据
4. Timestamp(M): 4bytes 记录日期时间 + 精确到的毫秒部分, 占用长度取决于 M
5. Datetime(M): 5bytes 记录日期时间 + 精确到的毫秒部分, 占用长度取决于 M
6. 毫秒部分占用情况, FSP 就是上面所说的 M 值:

FSP	Storage
-----	---------

0,0	0 bytes
-----	---------

1,2	1 bytes
-----	---------

3,4	2bytes
-----	--------

4,5	3bytes
-----	--------

7. Datetime 5bytes 记录分布 (这个没看懂 99 A0 65 50 41 B5 对应的数据应该是 2018-07-18 21:01:0 不知道怎么解析的):

1 bit sign (1= non-negative, 0= negative)

17 bits year\*13+month (year 0-9999, month 0-12)

5 bits day (0-31)

5 bits hour (0-23)

6 bits minute (0-59)

6 bits second (0-59)

-----

40 bits = 5 bytes

## 四、代码分析

### 1. 如何编译

Windows 下 新建工程 使用 visual studio 2005 即可编译

Linux 下 g++ parser.c -o a 即可编译 , 生成的为 a 的文件

### 2. 如何使用与存在的问题

因为是一个简易版本, 主要目的是为了实现在 c++ 语言的 解析 binlog 文件, 现在可以简单的解析出 binlog 中的不同 Events 对应的主要内容。

存在几个可以修改的地方 TO DO: (源码中 TO DO 的位置 ps. 解析的逻辑没问题, 但是耦合度太高, 建议最好重构)

- ① 输入的二进制文件是采用写死的方式
- ② 全局变量存储数据表的信息
- ③ 对于某些字段 (例如 TIMESTAMP 类型的) 分析存在问题
- ④ 解析出的 DELETE 语句不是标准 SQL
- ⑤ 解析出的其他 CRUD 语句, 都是标准格式且格式单一

### 3. 源码分析

#### ① 主要函数

```
/*format_desc Event*/
```

```
void process_format();
```

```
/*打印数据头部的内容*/
```

```

void print_Header();

/*xid*/
void process_xid();

/*QUERY_EVENT*/
void process_query();

/*TABLE_MAP_EVENT*/
void process_table_map();

/*WRITE_ROWS_EVENT*/
void process_write();

/*UPDATE_ROWS_EVENT*/
void process_update();

/*DELETE_ROWS_EVENT*/
void process_delete();

/*从建表的 sql 中提取表内的字段名*/
void fuck(string database_name,string sql);

```

注：主要功能为实现上文提到的不同 Event 的解析

## ② 通用头结构体(event header)

```

#pragma pack(1)
struct BinlogEventHeader
{
    int timestamp;

```

```
    unsigned char type_code;
    int  server_id;
    int  event_length;
    int  next_position;
    short flags;
};
```

### ③ 枚举类型

```
/**
    Enumeration type for the different types of log events.
    这个是 Mysql 源码里各个 Event 的枚举类型
*/
enum Log_event_type
{
    /**
        Every time you update this enum (when you add a type), you
        have to
        fix Format_description_event::Format_description_event().
    */
    UNKNOWN_EVENT= 0,
    START_EVENT_V3= 1,
    QUERY_EVENT= 2,
    STOP_EVENT= 3,
    ROTATE_EVENT= 4,
    INTVAR_EVENT= 5,
    LOAD_EVENT= 6,
    SLAVE_EVENT= 7,
    CREATE_FILE_EVENT= 8,
    APPEND_BLOCK_EVENT= 9,
```

```

EXEC_LOAD_EVENT= 10,
DELETE_FILE_EVENT= 11,
/**
    NEW_LOAD_EVENT is like LOAD_EVENT except that it has a
longer
    sql_ex, allowing multibyte TERMINATED BY etc; both types share
the
    same class (Load_event)
*/
NEW_LOAD_EVENT= 12,
RAND_EVENT= 13,
USER_VAR_EVENT= 14,
FORMAT_DESCRIPTION_EVENT= 15, //?aí •
XID_EVENT= 16,
BEGIN_LOAD_QUERY_EVENT= 17,
EXECUTE_LOAD_QUERY_EVENT= 18,

TABLE_MAP_EVENT = 19,

/**
    The PRE_GA event numbers were used for 5.1.0 to 5.1.15 and are
therefore obsolete.
*/
PRE_GA_WRITE_ROWS_EVENT = 20,
PRE_GA_UPDATE_ROWS_EVENT = 21,
PRE_GA_DELETE_ROWS_EVENT = 22,

/**
    The V1 event numbers are used from 5.1.16 until mysql-trunk-xx

```

```

*/
WRITE_ROWS_EVENT_V1 = 23,
UPDATE_ROWS_EVENT_V1 = 24,
DELETE_ROWS_EVENT_V1 = 25,

/**
    Something out of the ordinary happened on the master
*/
INCIDENT_EVENT= 26,

/**
    Heartbeat event to be send by master at its idle time
    to ensure master's online status to slave
*/
HEARTBEAT_LOG_EVENT= 27,

/**
    In some situations, it is necessary to send over ignorable
    data to the slave: data that a slave can handle in case there
    is code for handling it, but which can be ignored if it is not
    recognized.
*/
IGNORABLE_LOG_EVENT= 28,
ROWS_QUERY_LOG_EVENT= 29,

/** Version 2 of the Row events */
WRITE_ROWS_EVENT = 30,
UPDATE_ROWS_EVENT = 31,
DELETE_ROWS_EVENT = 32,

```



```

GTID_LOG_EVENT= 33,
ANONYMOUS_GTID_LOG_EVENT= 34,

PREVIOUS_GTIDS_LOG_EVENT= 35,

TRANSACTION_CONTEXT_EVENT= 36,

VIEW_CHANGE_EVENT= 37,

/* Prepared XA transaction terminal event similar to Xid */
XA_PREPARE_LOG_EVENT= 38,
/**
    Add new events here - right above this comment!
    Existing events (except ENUM_END_EVENT) should never change
their numbers
    */
    ENUM_END_EVENT /* end marker */
};

/**
    这个是 Mysql 源码里各个类型的枚举类型
    */
enum enum_field_types {
    MYSQL_TYPE_DECIMAL, MYSQL_TYPE_TINY,
    MYSQL_TYPE_SHORT, MYSQL_TYPE_LONG,
    MYSQL_TYPE_FLOAT, MYSQL_TYPE_DOUBLE,
    MYSQL_TYPE_NULL, MYSQL_TYPE_TIMESTAMP,
    MYSQL_TYPE_LONGLONG, MYSQL_TYPE_INT24,

```

```

MYSQL_TYPE_DATE, MYSQL_TYPE_TIME,
MYSQL_TYPE_DATETIME, MYSQL_TYPE_YEAR,
MYSQL_TYPE_NEWDATE, MYSQL_TYPE_VARCHAR,
MYSQL_TYPE_BIT,
MYSQL_TYPE_TIMESTAMP2,
MYSQL_TYPE_DATETIME2,
MYSQL_TYPE_TIME2,
MYSQL_TYPE_JSON=245,
MYSQL_TYPE_NEWDECIMAL=246,
MYSQL_TYPE_ENUM=247,
MYSQL_TYPE_SET=248,
MYSQL_TYPE_TINY_BLOB=249,
MYSQL_TYPE_MEDIUM_BLOB=250,
MYSQL_TYPE_LONG_BLOB=251,
MYSQL_TYPE_BLOB=252,
MYSQL_TYPE_VAR_STRING=253,
MYSQL_TYPE_STRING=254,
MYSQL_TYPE_GEOMETRY=255
} ;

/**
    枚举字段在 C 代码中的类型(需要验证)
*/
enum input_types{
    C_CHAR=0,C_SHORT=1,C_INT=2,
    C_LONG_LONG=3,C_FLOAT=4,
    C_DOUBLE=5,C_TIME=6,C_C
};

```

#### ④ Main 函数

```
int main()
{
    //这里写死了读入的文件（待修改）
    fp = fopen("mysql-bin.000001", "rb");
    if(fp == NULL)
    {
        printf("File not exist!\n");
        return -1;
    }
    int magic_number;
    //判断 magic num 是否正常 即判断文件是否损坏
    fread(&magic_number, 4, 1, fp);

    printf("%d - %s\n", magic_number, (char*)&magic_number);
    if(magic_number != MAGIC_NUMBER)
    {
        printf("File is not BinLog!\n");
        fclose(fp);
        return -1;
    }

    fseek(fp, 0L, SEEK_END);    //定位文件指针到尾部
    long file_len = ftell(fp);  //获得文件长度
    fseek(fp, 4L, SEEK_SET);    //已经读取了 4 字节的头

    //读日志文件 并 根据不同 Event 类型进行分析
    while(!feof(fp))
    {
```

```

        fread(&format_description_event_header, HEADER_LEN, 1,
fp);

        // 打印解析的头信息
        //print_Header();

        // 这个 Event 的结束位置
        int next_position =
format_description_event_header.next_position;

        /* 解析内容 根据 EVENT 的 TYPE*/
        switch (format_description_event_header.type_code)
        {
        case FORMAT_DESCRIPTION_EVENT:
            //printf("Type : FORMAT_DESCRIPTION_EVENT\n");
            process_format();
            break;
        case QUERY_EVENT:
            //printf("Type : QUERY_EVENT\n");
            process_query();
            break;
        case TABLE_MAP_EVENT:
            //printf("Type : TABLE_MAP_EVENT\n");
            process_table_map();
            break;
        case WRITE_ROWS_EVENT:
            //printf("Type : WRITE_ROWS_EVENT\n");
            process_write();
            break;
        case UPDATE_ROWS_EVENT:

```

```

        //printf("Type : UPDATE_ROWS_EVENT\n");
        process_update();
        break;
    case DELETE_ROWS_EVENT:
        //printf("Type : DELETE_ROWS_EVENT\n");
        process_delete();
        break;
    //代表 commit
    case XID_EVENT:
        //printf("Type : XID_EVENT\n");
        process_xid();
        break;
    }
    //移动指针到下一个位置
    printf("!!!!!! %d\n",next_position);
    if(next_position == file_len) break;
    fseek(fp,next_position,SEEK_SET);
}
return 0;
}

```

核心代码逻辑主要按照上文的对 Binlog 的分析进行解析，主要是以下几点注意事项：

- 1) 根据对文件头的解析得到每个 Event 的起始点和结束点，使用 fseek 定位文件指针；
- 2) 采用 fread 函数读取相应的字节；

## 五、参考资料

[1]

[https://www.cnblogs.com/f-ck-need-u/archive/2018/05/08/9010872.html#auto\\_id\\_0](https://www.cnblogs.com/f-ck-need-u/archive/2018/05/08/9010872.html#auto_id_0) 详细分析 MySQL 事务日志 (redo log 和 undo log)

[2] <http://blog.itpub.net/7728585/viewspace-2133188/> 解析 MYSQL BINLOG 二进制格式

[3] <https://xcoder.in/2015/08/10/mysql-binlog-try/> 初探 MySQL 的 Binlog