MySQL · 源码分析 · Innodb 引擎Redo日志存储格式简介

MySQL有多种日志。不同种类、不同目的的日志会记录在不同的日志文件中，它们可以帮助你找出mysqld内部发生的事情。比如错误日志：用来记录启动、运行或停止mysqld进程时出现的问题；查询日志：记录建立的客户端连接和执行的语句；二进制日志：记录所有更改数据的语句，主要用于逻辑复制；慢日志：记录所有执行时间超过long\_query\_time秒的所有查询或不使用索引的查询。而对MySQL中最常用的事务引擎innodb，redo日志是保证事务一致性非常重要的。本文结合MySQL版本5.6为分析源码介绍MySQL innodb引擎的重做（Redo）日志存储格式。

## **Redo日志**

任何对Innodb表的变动, redo log都要记录对数据的修改，redo日志就是记录要修改后的数据。redo 日志是保证事务一致性非常重要的手段，同时也可以使在bufferpool修改的数据不需要在事务提交时立刻写到磁盘上减少数据的IO从而提高整个系统的性能。这样的技术推迟了bufferpool页面的刷新，从而提升了数据库的吞吐，有效的降低了访问时延。带来的问题是额外的写redo log操作的开销。而为了保证数据的一致性，都要求WAL（Write Ahead Logging）。而redo 日志也不是直接写入文件，而是先写入redo log buffer，而是批量写入日志。当需要将日志刷新到磁盘时（如事务提交）,将许多日志一起写入磁盘。关于redo的产生及其生命周期详细过程，详见：https://yq.aliyun.com/articles/219。

## **Redo日志文件格式**

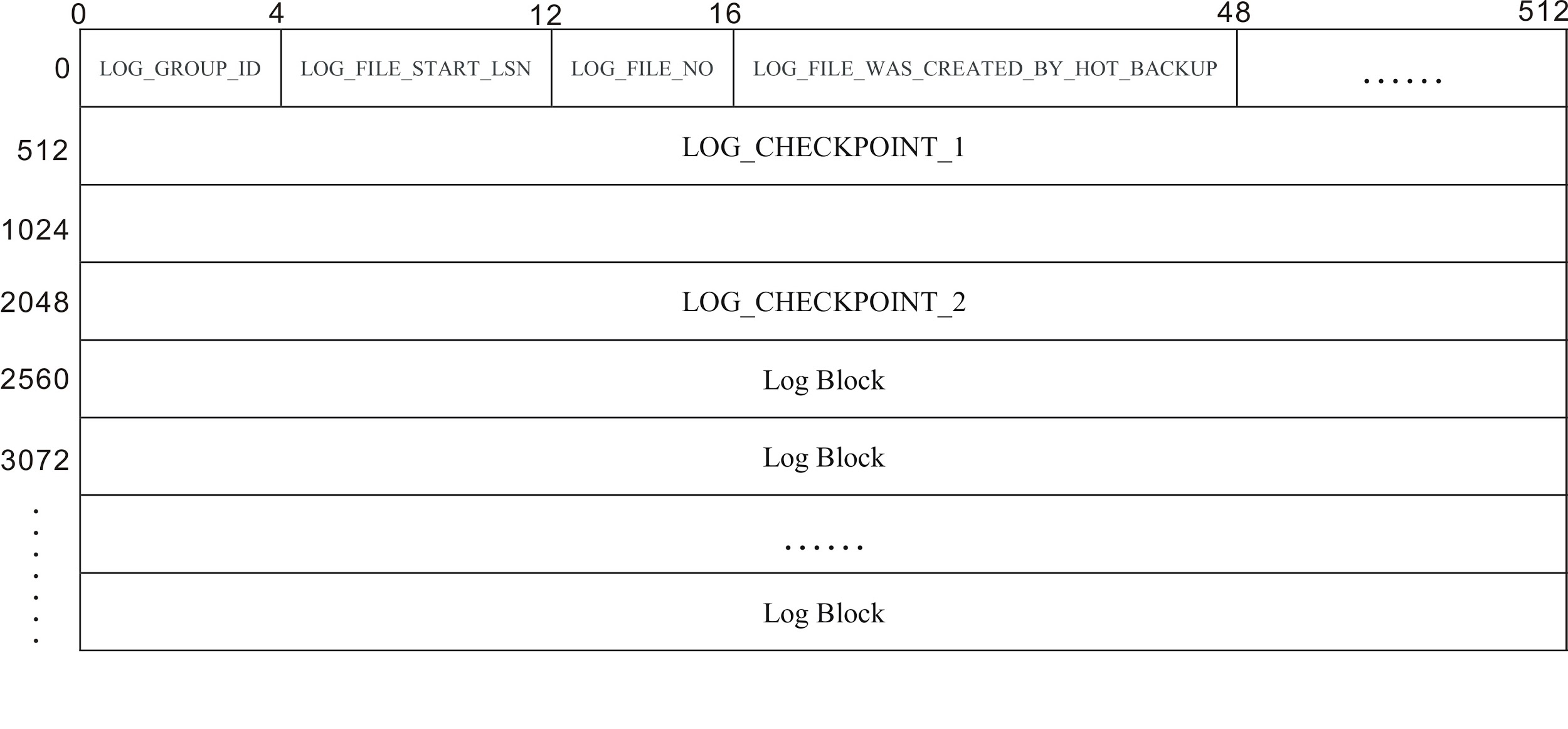
MySQL redo日志是一组日志文件，它们会被循环使用。Redo log文件的大小和数目可以通过特定的参数设置，详见innodb\_log\_file\_size 和 innodb\_log\_files\_in\_group 。

### **日志组结构**

在实现上日志组是由定义在log0log.h中的log\_group\_t结构体来表示的。在日志组结构体定义中含有以下重要信息： 日志文件的大小（file\_size）：记录日志组内每个日志文件的大小，通过参数innodb\_log\_file\_size配置。 日志文件的个数（n\_files）: 记录这个日志组中的文件个数，，通过参数innodb\_log\_files\_in\_group配置。 Checkpoint相关的信息：只有做完checkpoint后，其之前的日志才可以不再保留，否则系统崩溃时则无法恢复。在系统崩溃后的恢复，需要从checkpoint点开始。但我们需要把checkpoint的相关信息持久化的保存下来，才能在系统崩溃时不会丢失这些检查点相关的信息。Checkpoint相关的信息只存放在ib \_logfile0中。

### **日志文件结构**

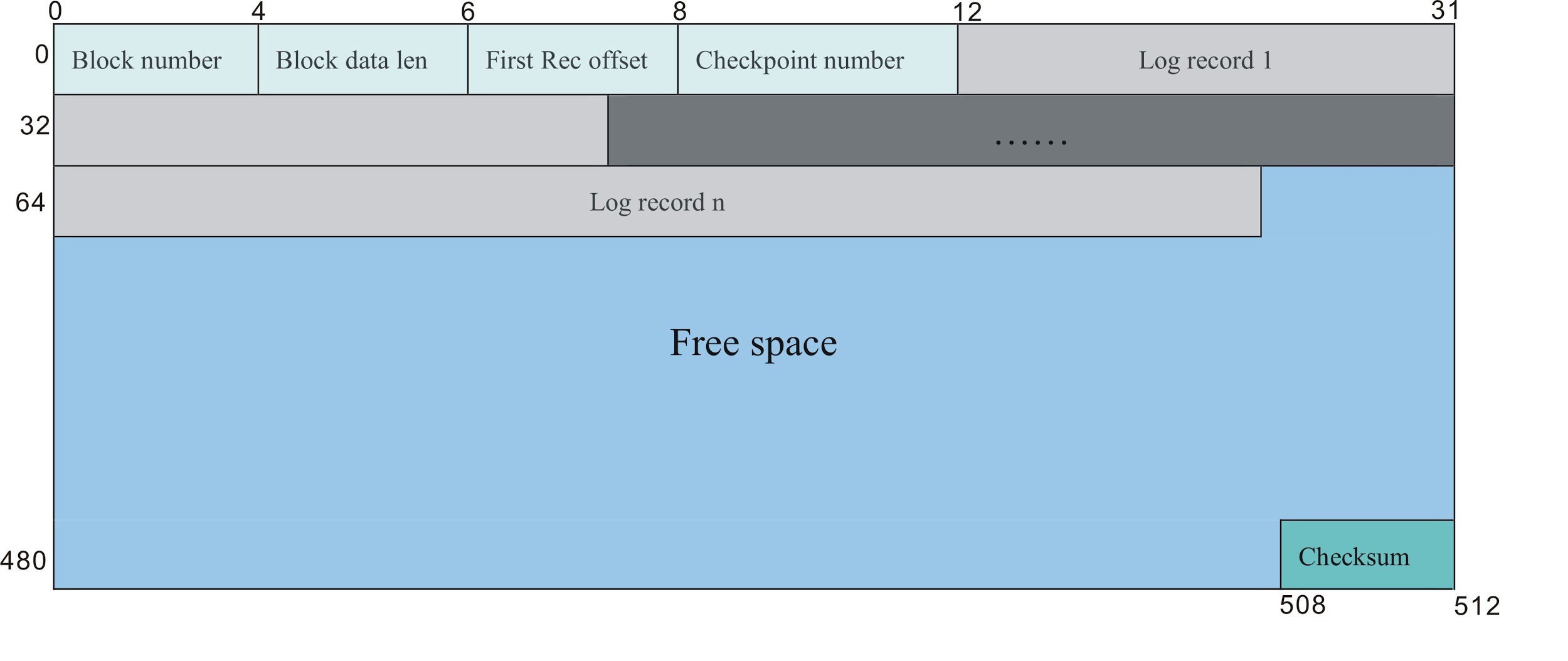
每个日志文件的前2048字节是存放的文件头信息。头结构定义在”storage/innobase/include/log0log.h” 中。其在重做日志文件内的布局如下图所示：



其中几个重要的字段在这里加以说明： 日志文件头共占用4个OS\_FILE\_LOG\_BLOCK\_SIZE的大小，这里对部分字段做简要介绍： 1) LOG\_GROUP\_ID 这个log文件所属的日志组，占用4个字节，当前都是0； 2) LOG\_FILE\_START\_LSN 这个log文件记录的开始日志的lsn，占用8个字节； 3) LOG\_FILE\_WAS\_CRATED\_BY\_HOT\_BACKUP 备份程序所占用的字节数，共占用32字节； 4) LOG\_CHECKPOINT\_1/LOG\_CHECKPOINT\_2 两个记录InnoDB checkpoint信息的字段，分别从文件头的第二个和第四个block开始记录，只使用日志文件组的第一个日志文件。 从地址2KB偏移量开始，其后就是顺序写入的各个日志块（log block）。

### **日志块结构**

所有的redo日志记录是以日志块为单位组织在一起的，日志块的大小为OS\_FILE\_LOG\_BLOCK\_SIZE（默认值为512字节），所有的日志记录以日志块为单位顺序写入日志文件。每一条记录都有自己的LSN（log sequence number， 表示从日志记录创建开始到特定的日志记录已经写入的字节数）。每个日志块包含一个日志头段（12字节）、一个尾段（4字节），以及一组日志记录（512 – 12 – 4 = 496字节） 。



首先看下日志块头结构。 1） log block number字段：占用日志块最开始的4个字节表示这是第几个block块。 其是通过LSN计算得来，计算的函数是log\_block\_convert\_lsn\_to\_no()； 2） block data len 字段:两个字节表示该block中已经有多少个字节被使用； 若是整个块都写满了日志的话它的长度就应该是（OS\_FILE\_LOG\_BLOCK\_SIZE） 512 字节。 3） First Record offset 字段：占用两个字节，表示该block中作为第一个新的mtr开始log record的偏移量。log\_block\_get\_first\_rec\_group()就是用保存在这个字段的值，获取到此块中第一个新的mtr开始的日志位置。 4） 中间496字节存放真正的Redo日志。 5） Checksum字段：是块的尾，占用四个字节，表示此log block计算出的校验值，用于正确性校验。

## **LSN和文件偏移量(offset)之间映射**

在MySQL Innodb引擎中LSN是一个非常重要的概念，表示从日志记录创建开始到特定的日志记录已经写入的字节数，LSN的计算是包含每个BLOCK的头和尾字段的。那如何由一个给定LSN的日志，在日志文件中找到它存储的位置的偏移量并能正确的读出来呢。所有的日志文件要属于日志组，而在log\_group\_t里的lsn和lsn\_offset字段已经记录了某个日志lsn和其存放在文件内的偏移量之间的对应关系。我们可以利用存储在group内的lsn和给定lsn之间的相对位置，来计算出给定lsn在文件中的存储位置。可以参考函数log\_group\_calc\_lsn\_offset()的实现。其核心代码实现如下：

gr\_lsn = group->lsn;

gr\_lsn\_size\_offset = log\_group\_calc\_size\_offset(group->lsn\_offset, group);

group\_size = log\_group\_get\_capacity(group);

**if** (lsn >= gr\_lsn) {

difference = lsn - gr\_lsn;

} **else** {

difference = gr\_lsn - lsn;

difference = difference % group\_size;

difference = group\_size - difference;

}

offset = (gr\_lsn\_size\_offset + difference) % group\_size;

/\* fprintf(stderr,

"Offset is " LSN\_PF " gr\_lsn\_offset is " LSN\_PF

" difference is " LSN\_PF "\n",

offset, gr\_lsn\_size\_offset, difference);

\*/

return(log\_group\_calc\_real\_offset(offset, group));

文章引用来源：

MySQL · 源码分析 · Innodb 引擎Redo日志存储格式简介

http://mysql.taobao.org/monthly/2017/09/07/