领略Debug MySQL的魅力

洪 斌

TECHNOLOGY ACTION 爱可生

Why → What → How

- 重启
- 搜索
- SR
- 源码
- 调试

```
Version: '5.6.35-log'  socket: '/data1/mysql/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check
07:35:53 UTC - mysqld got signal 6 ;
This could be because you hit a bug. It is also possible that this binary
or one of the libraries it was linked against is corrupt, improperly built,
or misconfigured. This error can also be caused by malfunctioning hardware.
We will try our best to scrape up some info that will hopefully help
diagnose the problem, but since we have already crashed,
something is definitely wrong and this may fail.

key_buffer_size=536870912
read_buffer_size=65536
max_used_connections=263
max_threads=5000
thread_count=210
connection_count=210
It is possible that mysqld could use up to
key_buffer_size + (read_buffer_size + sort_buffer_size)*max_threads = 1228272 K  bytes of memory
Hope that's ok; if not, decrease some variables in the equation.

Thread pointer: 0x0
Attempting backtrace. You can use the following information to find out
where mysqld died. If you see no messages after this, something went
terribly wrong...
stack_bottom = 0 thread_stack 0x1000000
./bin/mysqld(my_print_stacktrace+0x35)[0x9122f5]
./bin/mysqld(handle_fatal_signal+0x3d8)[0x675f18]
/lib64/libpthread.so.0[0x313920f710]
/lib64/libc.so.6(gsignal+0x35)[0x3138e32925]
/lib64/libc.so.6(abort+0x175)[0x3138e34105]
/usr/lib64/libstdc++.so.6(_ZN9__gnu_cxx27__verbose_terminate_handlerEv+0x12d)[0x31466bea5d]
/usr/lib64/libstdc++.so.6[0x31466bcbe6]
/usr/lib64/libstdc++.so.6[0x31466bcc13]
/usr/lib64/libstdc++.so.6[0x31466bcd0e]
/usr/lib64/libstdc++.so.6(_ZSt20__throw_out_of_rangePKc+0x67)[0x3146661db7]
./bin/mysqld[0xa6b62a]
./bin/mysqld[0xa6f0c7]
./bin/mysqld[0xa6f900]
./bin/mysqld[0xa71240]
/lib64/libpthread.so.0[0x31392079d1]
/lib64/libc.so.6(clone+0x6d)[0x3138ee8b6d]
The manual page at http://dev.mysql.com/doc/mysql/en/crashing.html contains
information that should help you find out what is causing the crash.
```

- 帮助学习原理
- 帮助诊断故障
- 帮助提升竞争力

What

故障诊断就像是看病。越是顽疾越难通过外部诊断出原因，必须通过手术获取样本，做病理分析。

debug就是给程序做手术。

平时用来练“解剖”，故障用来做“手术”

- **Inline Debug Statement & Error Handler**

```
mysql> show variables like 'debug';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| debug         |       |
+---------------+-------+
1 row in set (0.00 sec)

mysql> set global debug='d:F:L:t:i:o,/tmp/mysqld.trace';
Query OK, 0 rows affected (0.00 sec)
```

```
T@4: transaction.cc:   438: | | | | >trans_commit_stmt
T@4: transaction.cc:   265: | | | | | debug: add_unsafe_rollback_flags: 0
T@4:      binlog.cc:  8049: | | | | | >MYSQL_BIN_LOG::commit
T@4:      binlog.cc:  8051: | | | | | | info: query='set global debug='d:F:L:t:i:o,/tmp/mysqld.trace''
T@4:      binlog.cc:  8060: | | | | | | enter: thd: 0x7fe0a0904600, all: no, xid: 0, cache_mngr: 0x0
T@4:     handler.cc:  1879: | | | | | | >ha_commit_low
T@4:     handler.cc:  1939: | | | | | | <ha_commit_low 1939
T@4:      binlog.cc:  8070: | | | | | <MYSQL_BIN_LOG::commit 8070
T@4: rpl_context.cc:    59: | | | | | >Rpl_consistency_ctx::notify_after_transaction_commit
T@4: rpl_context.cc:    64: | | | | | <Rpl_consistency_ctx::notify_after_transaction_commit 64
T@4: transaction.cc:   270: | | | | | debug: reset_unsafe_rollback_flags
T@4: transaction.cc:   474: | | | | <trans_commit_stmt 474
T@4:   sql_union.cc:   895: | | | | >st_select_lex_unit::cleanup
T@4:   sql_union.cc:  1053: | | | | | >st_select_lex::cleanup()
T@4:   sql_union.cc:  1075: | | | | | <st_select_lex::cleanup() 1075
T@4:   sql_union.cc:   927: | | | | <st_select_lex_unit::cleanup 927
```

- **External Debugger**
  - **gdb**
  - **lldb**

Concept

- **Symbol**
  - 函数名、参数、变量名
  - 没有Symbol将无法看到对应函数名
- **Call Stack**
  - 存放某个程序的正在运行的函数的信息的栈
  - 由 Stack Frames 组成，每个 Stack Frame 对应于一个未完成运行的函数

How

Basic Process

1.识别故障

2.场景重现

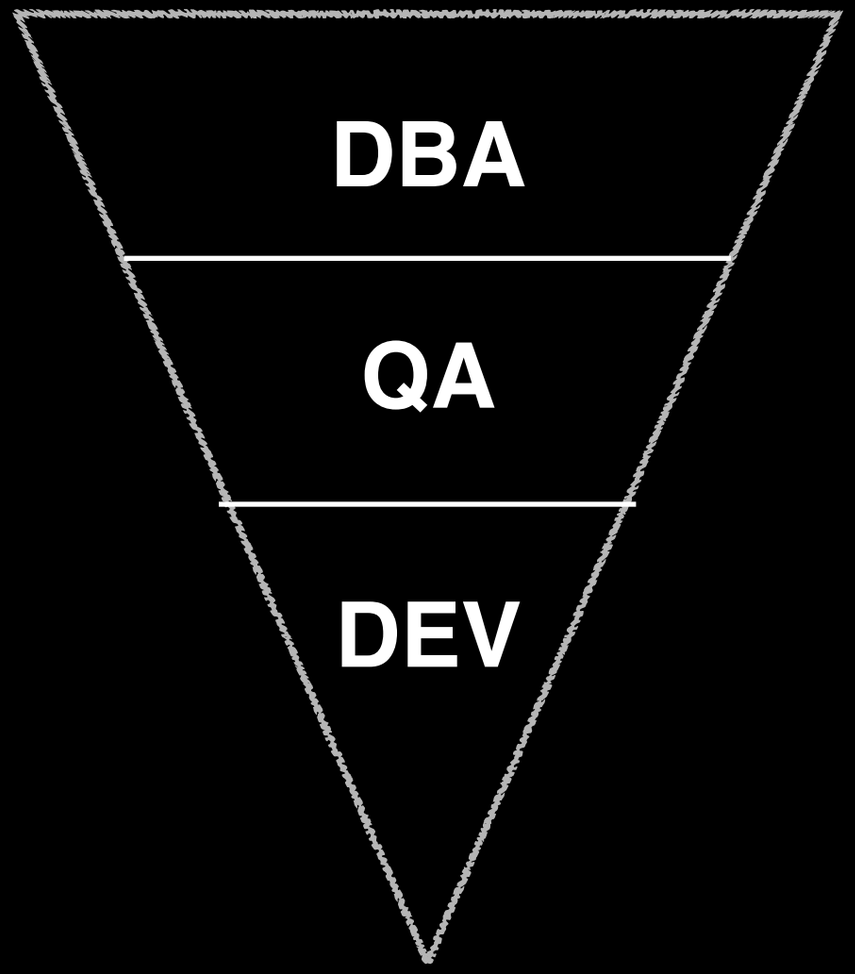
3.创建测试用例确认缺陷

4.定位缺陷根源

5.创建补丁修复缺陷

6.运行测试确认缺陷是否被修复

7.运行回归测试确认补丁没有产生新缺陷



Build

$ git clone https://github.com/mysql/mysql-server.git

$ cd mysql-server

$ git checkout mysql-5.7.17

$ cd BUILD; cmake .. -DWITH_DEBUG=1 -DWITH_BOOST=/path/

$ make

$ make install DESTDIR=“/path/mysql"

更多配置选项https://dev.mysql.com/doc/refman/5.7/en/source-configuration-options.html

Launch

```
hongbin@MBP ~> lldb -f ~/mysql5717/usr/local/mysql/bin/mysqld
(lldb) target create "/Users/hongbin/mysql5717/usr/local/mysql/bin/mysqld"
Current executable set to '/Users/hongbin/mysql5717/usr/local/mysql/bin/mysqld' (x86_64).
(lldb) r --defaults-file=~/.my.cnf
Process 3781 launched: '/Users/hongbin/mysql5717/usr/local/mysql/bin/mysqld' (x86_64)
```

Breakpoint

```
(lldb) b THD::query
Breakpoint 1: where = mysqld`THD::query() const + 16 at sql_class.h:4248, address = 0x0000000100b41ba0
(lldb) br l
Current breakpoints:
1: name = 'THD::query', locations = 1, resolved = 1, hit count = 0
  1.1: where = mysqld`THD::query() const + 16 at sql_class.h:4248, address = 0x0000000100b41ba0, resolved, hit count = 0
```

Continue

```
(lldb) c
Process 3795 resuming
Process 3795 stopped
* thread #28, stop reason = breakpoint 1.1
    frame #0: 0x0000000100b41ba0 mysqld`THD::query(this=0x000000010513e800) const at sql_class.h:4248
   4245   const LEX_CSTRING &query() const
   4246   {
   4247 #ifndef DBUG_OFF
-> 4248     if (current_thd != this)
   4249       mysql_mutex_assert_owner(&LOCK_thd_query);
   4250 #endif
   4251     return m_query_string;
```

Backtrace

```
(lldb) bt
* thread #28, stop reason = breakpoint 1.1
  * frame #0: 0x0000000100b41ba0 mysqld`THD::query(this=0x000000010513e800) const at sql_class.h:4248
    frame #1: 0x0000000100be609a mysqld`dispatch_command(thd=0x000000010513e800, com_data=0x000070000 4aecdb8, command=COM_QUERY) at sql_parse.cc:1437
    frame #2: 0x0000000100be88b6 mysqld`do_command(thd=0x000000010513e800) at sql_parse.cc:999
    frame #3: 0x0000000100d82c80 mysqld`::handle_connection(arg=0x0000000117779380) at connection_handler_per_thread.cc:300
    frame #4: 0x000000010155f81c mysqld`::pfs_spawn_thread(arg=0x000000011777acd0) at pfs.cc:2188
    frame #5: 0x00007fff9c84993b libsystem_pthread.dylib`_pthread_body + 180
    frame #6: 0x00007fff9c849887 libsystem_pthread.dylib`_pthread_start + 286
    frame #7: 0x00007fff9c84908d libsystem_pthread.dylib`thread_start + 13
```

Expression

```
(lldb) p m_query_string
(LEX_CSTRING) $2 = (str = "select 1", length = 8)
(lldb) p m_query_string.str="select 2"
(const char *) $3 = 0x0000000103cf03d0 "select 2"
(lldb) p m_query_string
(LEX_CSTRING) $4 = (str = "select 2", length = 8)
```

```
mysql> select 1;
+---+
| 2 |
+---+
| 2 |
+---+
1 row in set (30.10 sec)
```

Frame select

```
(lldb) fr se 1
frame #1: 0x0000000100be609a mysqld`dispatch_command(thd=0x0000000010513e800, com_data=0x0000700004aecdb8, command=COM_QUERY) at sql_parse.cc:1437
   1434     if (alloc_query(thd, com_data->com_query.query,
   1435                     com_data->com_query.length))
   1436       break;                                    // fatal error is set
-> 1437     MYSQL_QUERY_START(const_cast<char*>(thd->query().str), thd->thread_id(),
   1438                       (char *) (thd->db().str ? thd->db().str : ""),
   1439                       (char *) thd->security_context()->priv_user().str,
   1440                       (char *) thd->security_context()->host_or_ip().str);
```

Frame variable

```
(lldb) frame variable
(THD *) thd = 0x000000010513e800
(const COM_DATA *) com_data = 0x0000700004aecdb8
(enum_server_command) command = COM_QUERY
(bool) error = false
(Global_THD_manager *) thd_manager = 0x000000010407fe00
(_db_stack_frame_) _db_stack_frame_ = {
  func = 0x0000000101bb980f "do_command"
  file = 0x0000000101bb970b "/Users/hongbin/workbench/mysql-server/sql/sql_parse.cc"
  level = 2147483650
  prev = 0x0000700004aecd98
}
(const char *) packet_end = 0x0000000100f19735 "H\x83\U00000090]\x1fD"
(Parser_state) parser_state = {
  m_input = (m_compute_digest = false)
  m_lip = {
    m_thd = 0x0000000100f19735
    yylineno = 0
    yytoklen = 0
    yylval = 0x0000000003669c00
    lookahead_token = 78561920
    lookahead_yylval = 0x0000000101f3f598
    skip_digest = true
    query_charset = 0x0000000101f3f5b8
    m_ptr = 0x0000000000000001 ""
    m_tok_start = 0x0000700004aed000 "DRHT"
    m_tok_end = 0x000000010155f690 "UH\x89H\x8bPH\x89}H\x8bH\x89}H"
    m_end_of_query = 0x0000000000002b07 ""
    m_buf = 0x000000001c0008ff ""
    m_buf_length = 4688686288
```

Help

```
(lldb) apropos thread
The following commands may relate to 'thread':
  _regexp-bt              -- Show the current thread's call stack.  Any numeric argument displays at most that many frames.  The argument 'all' displ
  disassemble             -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and s
  expression              -- Evaluate an expression on the current thread.  Displays any returned value with LLDB's default formatting.
  frame                   -- Commands for selecting and examing the current thread's stack frames.
  frame info              -- List information about the current stack frame in the current thread.
  frame select            -- Select the current stack frame by index from within the current thread (see 'thread backtrace'.)
  process continue        -- Continue execution of all threads in the current process.
  register                -- Commands to access registers for the current thread and stack frame.
  thread                  -- Commands for operating on one or more threads in the current process.
  thread backtrace        -- Show thread call stacks.  Defaults to the current thread, thread indexes can be specified as arguments.  Use the thread-
  thread continue         -- Continue execution of the current target process.  One or more threads may be specified, by default all threads continue
  thread info             -- Show an extended summary of one or more threads.  Defaults to the current thread.
  thread list             -- Show a summary of each thread in the current target process.
  plan                    -- Commands for managing thread plans that control execution.
  thread plan discard     -- Discards thread plans up to and including the specified index (see 'thread plan list'.)  Only user visible plans can be
  thread plan list        -- Show thread plans for one or more threads.  If no threads are specified, show the current thread.  Use the thread-index
  thread select           -- Change the currently selected thread.
  thread step-in          -- Source level single step, stepping into calls.  Defaults to current thread unless specified.
  thread step-inst        -- Instruction level single step, stepping into calls.  Defaults to current thread unless specified.
  thread step-inst-over   -- Instruction level single step, stepping over calls.  Defaults to current thread unless specified.
  thread step-out         -- Finish executing the current stack frame and stop after returning.  Defaults to current thread unless specified.
  thread step-over        -- Source level single step, stepping over calls.  Defaults to current thread unless specified.
  thread until            -- Continue until a line number or address is reached by the current or specified thread.  Stops when returning from the cu
  bt                      -- Show the current thread's call stack.  Any numeric argument displays at most that many frames.  The argument 'all' displ
  c                       -- Continue execution of all threads in the current process.
  call                    -- Evaluate an expression on the current thread.  Displays any returned value with LLDB's default formatting.
  continue                -- Continue execution of all threads in the current process.
  di                      -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and s
  dis                     -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and s
  f                       -- Select the current stack frame by index from within the current thread (see 'thread backtrace'.)
  finish                  -- Finish executing the current stack frame and stop after returning.  Defaults to current thread unless specified.
```

https://lldb.llvm.org/tutorial.html

Core-dump

ulimit -c unlimited
echo "1" > /proc/sys/kernel/core_uses_pid
echo "/tmp/core" > /proc/sys/kernel/core_pattern
echo 2 > /proc/sys/fs/suid_dumpable 对于非root用户启动的程序，需要设置为2

[mysqld]
core-file

Demo

系统表空间被误删，如何恢复数据

1.新实例创建相同表，并拷贝原ibd文件

2.对接数据字典和ibd文件表空间id

3.对接数据字典和ibd索引id