

第五讲：MySQL 的线程模型

知春路遇上八里桥

<2024-05-20 Mon>



① 多进程和多线程

② 线程生命周期

③ 常见线程功能

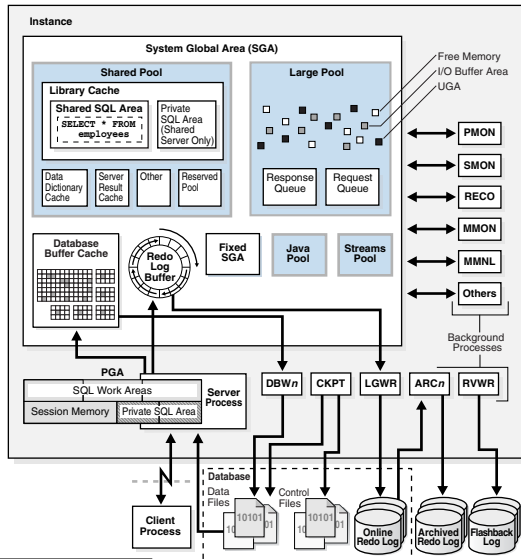


1

多进程和多线程



多进程的数据库架构¹



¹https://docs.oracle.com/cd/E11882_01/server.112/e40540/process.htm

查看 ORACLE 进程列表

top -u oracle -c

```
top - 13:20:19 up 18 min, 1 user, load average: 0.08, 0.08, 0.05
Tasks: 341 total, 1 running, 340 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.0 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 32941528 total, 21490972 free, 1133328 used, 10317228 buff/cache
KiB Swap: 16515068 total, 16515068 free, 0 used. 29999120 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1819	oracle	20	0	9909.0m	43412	36652	S	0.8	0.1	0:05.11	ora_dia0_orallg
2531	oracle	20	0	5291460	297020	13788	S	0.8	0.9	0:21.93	/u01/app/oracle/product/11.2.0/dbhome_1/jc
1827	oracle	20	0	9911.1m	59580	51276	S	0.4	0.2	0:00.27	ora_dbw2_orallg
1709	oracle	20	0	243084	14516	10584	S	0.0	0.0	0:00.23	/u01/app/oracle/product/11.2.0/dbhome_1/b
1804	oracle	20	0	9906.8m	31232	28848	S	0.0	0.1	0:00.31	ora_pmon_orallg
1806	oracle	20	0	9904.5m	14412	12252	S	0.0	0.0	0:00.31	ora_psp0_orallg
1809	oracle	20	0	9904.5m	14028	11868	S	0.0	0.0	0:00.45	ora_vktm_orallg
1813	oracle	20	0	9904.5m	14176	12016	S	0.0	0.0	0:00.05	ora_gen0_orallg
1815	oracle	20	0	9904.5m	13932	11772	S	0.0	0.0	0:00.10	ora_diag_orallg
1817	oracle	20	0	9905.0m	35164	32620	S	0.0	0.1	0:00.15	ora_dbrm_orallg
1821	oracle	20	0	9904.5m	699512	697352	S	0.0	2.1	0:01.19	ora_mman_orallg
1823	oracle	20	0	9912.1m	67376	58444	S	0.0	0.2	0:00.39	ora_dbw0_orallg
1825	oracle	20	0	9911.1m	60144	51852	S	0.0	0.2	0:00.32	ora_dbw1_orallg
1829	oracle	20	0	9911.1m	60984	52688	S	0.0	0.2	0:00.27	ora_dbw3_orallg
1831	oracle	20	0	9919.1m	35652	33140	S	0.0	0.1	0:00.52	ora_lgwr_orallg
1833	oracle	20	0	9905.0m	35308	32948	S	0.0	0.1	0:00.70	ora_ckpt_orallg
1835	oracle	20	0	9911.0m	134876	129624	S	0.0	0.4	0:00.33	ora_smon_orallg
1837	oracle	20	0	9905.0m	21452	18612	S	0.0	0.1	0:00.03	ora_reco_orallg
1839	oracle	20	0	9910.4m	103720	98116	S	0.0	0.3	0:00.76	ora_mmon_orallg
1841	oracle	20	0	9905.7m	49636	46272	S	0.0	0.2	0:01.88	ora_mnnl_orallg
1843	oracle	20	0	9927.5m	14196	11224	S	0.0	0.0	0:00.05	ora_d000_orallg



查看 MySQL 的线程列表

- ① 自 8.0.27 出现操作系统的线程名称，见发布日志²
- ② 通过 top 查看线程列表和详细信息

```
top -H -p <MYSQLD_PID>
```

```
top - 05:34:37 up 4 days, 22:51, 2 users, load average: 0.00, 0.00, 0.00
Threads: 42 total, 0 running, 42 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.1 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7936.5 total, 221.6 free, 916.0 used, 6798.9 buff/cache
MiB Swap: 4096.0 total, 4052.4 free, 43.6 used. 6712.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
73208	mes	20	0	3567416	663412	72952	S	1.3	8.2	64:32.19	ib_log_files_g
73190	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.80	mysqld
73193	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.00	ib_io_ibuf
73194	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.02	ib_io_rd-1
73195	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.02	ib_io_rd-2
73196	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.04	ib_io_rd-3
73197	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:00.01	ib_io_rd-4
73198	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:01.70	ib_io_wr-1
73199	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:01.19	ib_io_wr-2
73200	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:01.44	ib_io_wr-3
73201	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:01.37	ib_io_wr-4
73202	mes	20	0	3567416	663412	72952	S	0.0	8.2	1:12.42	ib_pg_flush_co
73203	mes	20	0	3567416	663412	72952	S	0.0	8.2	0:35.95	ib_log_checkpoint
73204	mes	20	0	3567416	663412	72952	S	0.0	8.2	5:21.59	ib_log_fl_notif

²<https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-27.html#mysqld-8-0-27-performance-schema>

性能视图 performance_schema

```
select
  thread_id,      -- 线程 ID
  thread_os_id,   -- 操作系统线程 ID
  name            -- 线程名称
from
  performance_schema.threads;
```

查询 threads 表示例³

```
mysql> select thread_id, thread_os_id, name from threads;
```

+-----+-----+-----+-----+			
thread_id	thread_os_id	name	
+-----+-----+-----+-----+			
1	73190	thread/sql/main	
3	73193	thread/innodb/io_ibuf_thread	
4	73194	thread/innodb/io_read_thread	
5	73195	thread/innodb/io_read_thread	

³<https://dev.mysql.com/doc/refman/8.0/en/performance-schema-threads-table.html>



2

线程生命周期



gdb 调试多线程程序

- 控制线程启动/退出是打印的事件

```
(gdb) set print thread-events on/off  
[New Thread 0x41e02940 (LWP 25582)]
```

- 线程调试命令

- 1 info threads 查看线程列表和详细信息
- 2 thread <threadno> 切换到线程 <threadno>



线程注册表

- 注册表结构 `static PSI_thread_info ...`
- ★ `storage/innobase/handler/ha_innodb.cc`

```
820 static PSI_thread_info all_innodb_threads[] = {
821     PSI_THREAD_KEY(log_archiver_thread, "ib_log_arch", PSI_FLAG_SINGLETON, 0,
822                     PSI_DOCUMENT_ME),
823     PSI_THREAD_KEY(page_archiver_thread, "ib_page_arch", PSI_FLAG_SINGLETON, 0,
824                     PSI_DOCUMENT_ME),
825     PSI_THREAD_KEY(buf_dump_thread, "ib_buf_dump", PSI_FLAG_SINGLETON, 0,
826                     PSI_DOCUMENT_ME),
827     PSI_THREAD_KEY(clone_ddl_thread, "ib_clone_ddl", PSI_FLAG_SINGLETON, 0,
828                     PSI_DOCUMENT_ME),
```

- PSI_thread_info \Rightarrow PSI_thread_info_v5 结构体

- ▶ `PSI_thread_key *m_key` 注册线程的 key
- ▶ `const char *m_name` 注册的名称
- ▶ `const char *m_os_name` 注册线程在操作系统下看的名称
- ▶ ...



线程创建

- MySQL 原始的创建线程
- `inline_mysql_thread_create()` 创建线程函数，调用 `my_thread_create()`

▶ `★ include/mysql/psi/mysql_thread.h`

```
134 static inline int inline_mysql_thread_create(  
135     PSI_thread_key key [[maybe_unused]],  
136     unsigned int sequence_number [[maybe_unused]], my_thread_handle *thread,  
137     const my_thread_attr_t *attr, my_start_routine start_routine, void *arg) {  
138     int result;  
139     #ifdef HAVE_PSI_THREAD_INTERFACE  
140     result = PSI_THREAD_CALL(spawn_thread)(key, sequence_number, thread, attr,  
141                                           start_routine, arg);
```

- `my_thread_create()` 在 Linux 下调用 `pthread_create()`

▶ `★ include/mysql/psi/mysql_thread.h`

```
78 int my_thread_create(my_thread_handle *thread, const my_thread_attr_t *attr,  
79                     my_start_routine func, void *arg) {  
80     #ifndef _WIN32  
81     return pthread_create(&thread->thread, attr, func, arg);  
82     #else
```



线程创建（续）

- 存储引擎创建 PFS 线程方法
- pfs_spawn_thread() 中调用创建线程函数 create_thread()

▶ ★ storage/perfschema/pfs.cc

```
3002 extern "C" {  
3003 static void *pfs_spawn_thread(void *arg) {  
3004     auto *typed_arg = (PFS_spawn_thread_arg *)arg;  
3005     void *user_arg;  
3006     void *(*user_start_routine)(void *);
```

- create_thread() 函数中分配 PFS_thread 对象内存，并进行基础对象成员的初始化

▶ ★ storage/perfschema/pfs_instr.cc

```
609 PFS_thread *create_thread(PFS_thread_class *klass, PSI_thread_seqnum seqnum,  
610                           const void *identity [[maybe_unused]],  
611                           ulonglong processlist_id) {  
612     PFS_thread *pfs;  
613     pfs_dirty_state dirty_state;
```



线程启动

- pfs_spawn_thread(void *arg) 启动用户代码

- ▶ ★ storage/perfschema/pfs.cc

```
3032     /*
3033         Secondly, free the memory allocated in spawn_thread_v1().
3034         It is preferable to do this before invoking the user
3035         routine, to avoid memory leaks at shutdown, in case
3036         the server exits without waiting for this thread.
3037     */
3038     user_start_routine = typed_arg->m_user_start_routine;
3039     user_arg = typed_arg->m_user_arg;
3040     my_free(typed_arg);
3041
3042     /* Then, execute the user code for this thread. */
3043     (*user_start_routine)(user_arg);
3044
3045     return nullptr;
3046 }
```



查看线程入口函数

① 以 log_archiver_thread 为例

② ★ storage/innobase/handler/ha_innodb.cc

```
820 static PSI_thread_info all_innodb_threads[] = {  
821     PSI_THREAD_KEY(log_archiver_thread, "ib_log_arch", PSI_FLAG_SINGLETON, 0,  
822     PSI_DOCUMENT_ME),
```

③ 搜索代码 log_archiver_thread(

④ 得到线程入口函数 ★ storage/innobase/arch/arch0arch.cc

```
615 /** Archiver background thread */  
616 void log_archiver_thread() {  
617     Arch_File_Ctx log_file_ctx;  
618     lsn_t log_arch_lsn = LSN_MAX;
```

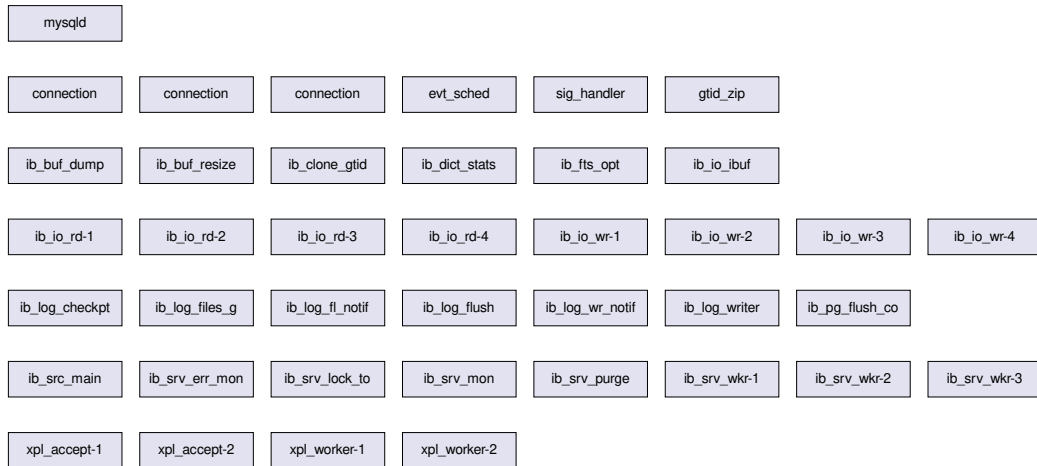


3

常见线程功能



后台线程列表⁴



⁴<https://github.com/Jeanhwea/mysql-source-course/blob/master/assets/thd-name-ref.org>



常见线程功能

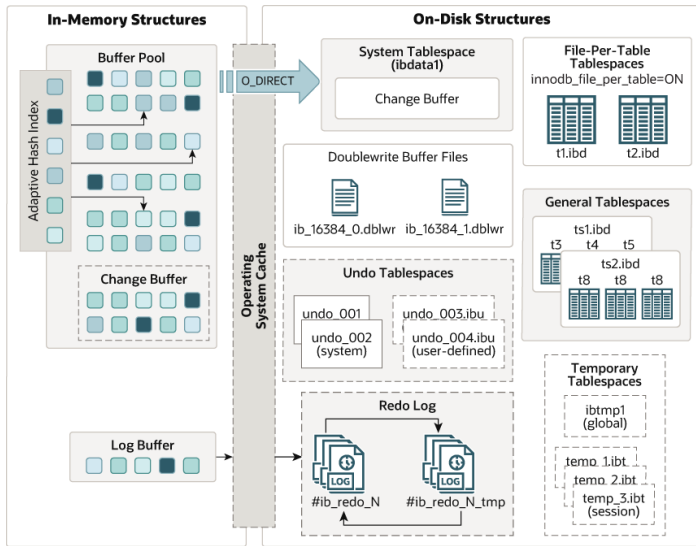
- `mysqld` 连接监听主线程，用于监听处理客户端的连接请求
- `connection` 连接出来线程，客户端建立连接后会分配
- `evt_sched` 事件调度线程，用来调度执行每个表上定义事件，例如

```
CREATE EVENT myevent
ON SCHEDULE EVERY 6 HOUR
COMMENT 'A sample comment.'
DO UPDATE myschema.mytable SET mycol = mycol + 1;
```

- `sig_handler` 信号处理线程，用来处理信号
- `gtid_zip` 开启一个线程用来压缩 `GTID_Table`
- `xpl_accept/xpl_worker` `mysqlx` 实现了 X Protocol 来支持 X Plugin



InnoDB 引擎



innodb 线程

- `ib_src_main` 引擎主线程，优先级最高
 - ① 其内部包含主循环、后台循环、刷新循环、暂停循环
 - ② 会根据其内部运行的相关状态在前述各循环间中进行切换
- `ib_io_rd-n` 读线程，负责从磁盘上读入数据页
 - ① 其由 `innodb_read_io_threads` 控制
 - ② 默认值为 4
- `ib_io_wr-n` 写线程，负责将数据页写入磁盘，
 - ① 其由 `innodb_write_io_threads` 控制
 - ② 默认值为 4
- `ib_log_writer` 负责把日志缓冲中的内容刷新到 redo log 文件中
- `ib_srv_purge` 负责删除无用的 undo 页
- `ib_log_checkpoint` 负责在 redo log 发生切换时，执行 checkpoint 操作
- `ib_srv_err_mon` 负责 mysql 报错的监控
- `ib_srv_lock_to` 负责 mysql 锁的监控



结束

