

第四讲：MySQL 服务启动源码分析

知春路遇上八里桥

<2024-05-15 Wed>



- 1 启动的前奏曲
- 2 服务器系统启动
- 3 核心代码阅读
- 4 初始函数汇总



1

启动的前奏曲



启动选项

❶ 通过命令行参数传入

```
mysqld --default-storage-engine=MyISAM  
mysqld --default_storage_engine=MyISAM  
mysql -h127.0.0.1 -uroot -p
```

❷ 通过配置文件写入, /etc/my.cnf, \$MYSQL_HOME/my.cnf

- ▶ 可以通过 --defaults-file 选项制定

```
[mysqld]  
default-storage-engine = MyISAM
```

❸ 使用命令行自带的帮助输出, 查看可用的启动选项

```
mysqld --help --verbose  
mysql --help
```



系统变量

❶ 影响 `mysqld` 服务器程序运行过程中行为的变量，下面列举几个系统变量¹

- ▶ `default_storage_engine` 表示默认的存储引擎
- ▶ `max_connections` 允许同时连入的客户端数量
- ▶ 更多我们后续用到再介绍

❷ 可以使用 `show variables like ...` 查看

❸ `mysqld` 运行过程中也可不停机设置系统变量，变量有对应的作用范围²

- ▶ GLOBAL：全局变量，影响服务器的整体操作
- ▶ SESSION：会话变量，影响某个客户端连接的操作

```
set global default_storage_engine = MyISAM;
set session default_storage_engine = MyISAM;
show session variables like 'default_storage_engine';
show global variables like 'default_storage_engine';
```

¹<https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html>

²并不是所有系统变量都具有 GLOBAL 和 SESSION 的作用范围



状态变量

① 记录 mysqld 运行实时状态的变量，方便我们更好地了解其运行状态³

- ▶ Threads_connected 当前有多少客户端与服务器建立了连接
- ▶ Uptime 表示服务器开机时长

② 状态变量时只读的，可以使用 `show status like ...` 查看

```
mysql> show status like 'thread%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_cached | 0     |
| Threads_connected | 1     |
| Threads_created | 1     |
| Threads_running | 2     |
+-----+-----+
4 rows in set (0.01 sec)
```

③ 状态变量也有 GLOBAL 和 SESSION 两个作用范围的

³<https://dev.mysql.com/doc/refman/8.0/en/server-status-variables.html>



2

服务器系统启动



从 main() 函数开始

- 主函数 main() 调用 mysqld_main() 函数启动服务器

- ▶ ★ sql/main.cc

```
24  extern int mysqld_main(int argc, char **argv);
```

```
25
```

```
26  int main(int argc, char **argv) { return mysqld_main(argc, argv); }
```

- 初始化函数，开始真正地启动工作

- ▶ ★ sql/mysqld.cc

```
7270  #ifdef _WIN32
```

```
7271  int win_main(int argc, char **argv)
```

```
7272  #else
```

```
7273  int mysqld_main(int argc, char **argv)
```

```
7274  #endif
```

```
7275  {
```

```
// 此处省略共计 1065 行代码
```

```
8340 }
```



使用 gdb/lldb 调试 mysqld 的两种方式

① 方案一：直接加载启动方式⁴

- ▶ 选项 `-o` 可以添加启动的命令
- ▶ 选项 `--` 后可以添加加载参数⁵

```
lldb -o "b main" \  
      -o "r" \  
      -- mysqld --gdb
```

② 方案二：启动进程后，再 attach 到进程上

- ▶ 启动 mysqld 进程
`mysqld --gdb`
- ▶ 附着到 mysqld 运行的进程，`$MYSQLD_PID` 是进程 pid

```
gdb -p $MYSQLD_PID  
lldb -p $MYSQLD_PID
```

③ 来上手试试 ...

⁴更多 lldb/gdb 调试命令 <https://lldb.llvm.org/use/map.html>

⁵查看当前参数 (lldb) `settings show target.run-args`

最早的初始化代码

- 从 `mysqld_main()` 最开始的代码开始看

- ▶ `sql/mysql.cc`

```
7273 int mysqld_main(int argc, char **argv)
7274 #endif
7275 {
7276     initialize_stack_direction();
7277
7278     // Substitute the full path to the executable in argv[0]
7279     substitute_propath(argv);
7280     sysd::notify_connect();
7281     sysd::notify("STATUS=Server startup in progress\n");
```

- `substitute_propath(argv)` 替换全路径

- ▶ 程序名替换成全路径 "mysqld" \Rightarrow "/opt/mysql/bin/mysqld"
- ▶ 替换 ~ 符号成 \$HOME 对应的路径
- ▶ 环境变量 \$PATH 解释

- `sysd::notify_connect()` 尝试连接 socket

- ▶ 查找 NOTIFY_SOCKET 环境变量中包含的 socket 文件名
- ▶ 如果存在, 调用 `connect()` 连接它



最早的初始化代码（续一）

- 继续看代码 ★ sql/mysqld.cc

```
7290 #ifndef _WIN32
7291 #ifdef WITH_PERFSCHEMA_STORAGE_ENGINE
7292     pre_initialize_performance_schema();
7293 #endif /*WITH_PERFSCHEMA_STORAGE_ENGINE */
7294     // For windows, my_init() is called from the win specific mysqld_main
7295     if (my_init()) // init my_sys library & pthreads
7296     {
7297         LogErr(ERROR_LEVEL, ER_MYINIT_FAILED);
7298         flush_error_log_messages();
7299         return 1;
7300     }
7301 #endif /* _WIN32 */
```

- my_init() 初始化 my_sys 库和 pthreads 线程库

- ▶ my_thread_global_init() 初始化线程的环境，主要初始化锁
- ▶ my_thread_init() 申请 mysys 线程的内存，主要用于调试



最早的初始化代码（续二）

- 继续看代码 ★ sql/mysqld.cc

```
7303 orig_argc = argc;
7304 orig_argv = argv;
7305 my_getopt_use_args_separator = true;
7306 my_defaults_read_login_file = false;
7307 if (load_defaults(MYSQL_CONFIG_NAME, load_default_groups, &argc, &argv,
7308                  &argv_alloc)) {
7309     flush_error_log_messages();
7310     return 1;
7311 }
```

- load_defaults() 从配置文件中读取配置项，调用 my_load_defaults()

- ▶ init_default_directories() 获取配置的目录
- ▶ my_search_option_files() 处理在缺省目录下的配置文件
- ▶ my_default_get_login_file() 从集群中读取配置，
 - ① 继续调用 my_search_option_files() 处理
- ▶ set_args_separator() 设置选项分隔符



最早的初始化代码（续三）

- 继续看代码 ★ sql/mysqld.cc

```
7345 bool arg_separator_added = false;
7346 if (persisted_variables_cache.init(&argc, &argv) ||
7347     persisted_variables_cache.load_persist_file() ||
7348     persisted_variables_cache.append_parse_early_variables(
7349         &argc, &argv, arg_separator_added)) {
7350     flush_error_log_messages();
7351     return 1;
7352 }
```

- persisted_variables_cache 持久化的参数配置缓存，采用 JSON 进行存储，
 - ▶ .init() 初始化，调用 my_handle_options() 主要处理参数的逻辑
 - ▶ .load_persist_file() 加载持久化文件的配置，转成 JSON，分析出 KV 结构
 - ▶ .append_parse_early_variables() 将上步分解出的配置参数追加到命令行参数中

• ...



3

核心代码阅读



核心初始化函数

... \Rightarrow sys_var_init() \Rightarrow ... \Rightarrow init_common_variables() \Rightarrow ... \Rightarrow init_server_components()

❶ sys_var_init() 初始化系统变量 \star sql/mysql.cc

```
7328 sys_var_init();
```

❷ init_common_variables() 初始化公共变量 \star sql/mysql.cc

```
7682 if (init_common_variables()) {  
7683     setup_error_log();  
7684     unireg_abort(MYSQLD_ABORT_EXIT); // Will do exit  
7685 }
```

❸ init_server_components() 初始化服务器组件 \star sql/mysql.cc

```
7882 if (init_server_components()) unireg_abort(MYSQLD_ABORT_EXIT);
```



核心初始化函数 - sys_var_init()

- 初始化 ★ sql/sys_vars.cc 中定义静态变量

```
146 static_system_variable_hash = new collation_unordered_map<string, sys_var *>(
147     system_charset_info, PSI_INSTRUMENT_ME);
148
149 dynamic_system_variable_hash = new collation_unordered_map<string, sys_var *>(
150     system_charset_info, PSI_INSTRUMENT_ME);
151
152 never_persistable_vars = new collation_unordered_set<string>(
153     {PERSIST_ONLY_ADMIN_X509_SUBJECT, PERSISTED_GLOBALS_LOAD},
154     system_charset_info, PSI_INSTRUMENT_ME);
155
156 if (add_static_system_variable_chain(all_sys_vars.first)) goto error;
```

- 调用 mysql_add_sys_var_chain() 添加系统变量



核心初始化函数 - `init_common_variables()`

- `init_thread_environment()` 初始化与线程相关的锁
- `mysql_init_variables()` 初始化全局变量为缺省值
- `mysql_bin_log.set_psi_keys()` 完成 binlog 的初始化, 让 PSI 的对 binlog 可见
- `mysql_bin_log.init_pthread_objects()` 初始化 bin log 与线程相关的锁
- `get_options()` 剩余的可选项, binlog, 压缩算法, 页的大小, 线程缓存大小
 - ① `Connection_handler_manager::init()` 初始化连接池
 - ② `Per_thread_connection_handler::init()` 初始化 `Per_thread_connection_handler`, 初始化锁
- `mysql_client_plugin_init()` 初始化 mysql client 的 plugin
- `lex_init()` 词法分析器初始化
- 选字符集/collation
- 日志配置
- `initialize_create_data_directory(mysql_real_data_home)` 创建数据目录
- 表名的大小写的配置
- ...



核心初始化函数 - init_server_components()

- mdl_init() 初始化 metadata locking 子系统
- partitioning_init() 分区初始化
- table_def_init() 表定义缓存
- hostname_cache_init() 初始化 (client) 主机名缓存
- my_timer_initialize() 初始化计时器
- xa::Transaction_cache::initialize() 初始化事务缓存
- MDL_context_backup_manager::init() 维护 XA 事务中元数据锁
- delegates_init() 初始化外部 observer 的委托
- process_key_caches() 用 ha_init_key_cache() 函数初始化所有的 key 缓存
- ha_init_errors() 初始化存储引擎的错误信息
- gtid_server_init() GTID server 初始化
- udf_init_globals() 初始化 udf 结构体
- init_ssl() 配置 SSL 包
- init_sql_command_flags() 初始化 SQL 命令 flag



核心初始化函数 - `init_server_components()` (续)

- `dd::init()` 数据字典初始化
- `plugin_register_dynamic_and_init_all()` 注册动态的 plugin 和初始化所有的 plugin
- `dd::performance_schema::init_pfs_tables()` 处理好 pfs 相关的表
- `upgrade_system_schemas()` 升级系统库表
- `Resource_group_mgr::post_init()` 从磁盘读取资源组数据
- `handle_options()` 处理剩余的参数可选项
- `ha_init()` 初始化存储引擎
- `initialize_storage_engine()` 初始化缺省（或临时）的存储引擎
- `Recovered_xa_transactions::init()` 恢复 XA 事务初始化
- `init_server_auto_options()` 初始化服务自动项，
- `MYSQL_BIN_LOG::open_binlog()` 打开 binlog 文件，分析主备 binlog
- `init_optimizer_cost_module()` 初始化优化器
- `ft_init_stopwords()` 停用词初始化
- `init_max_user_conn()` 初始化 client 连接



进入事件循环

❶ 服务器初始化最后运行 connection_event_loop()

- ▶ 静态全局变量，用于监听处理客户端的连接请求

- ▶ ★ sql/mysqld.cc

```
1822 static Connection_acceptor<Mysqld_socket_listener> *mysqld_socket_acceptor =  
1823     nullptr;
```

- ▶ 进入连接器的事件循环中

- ▶ ★ sql/mysqld.cc

```
8286 mysqld_socket_acceptor->connection_event_loop();
```

❷ 调用连接器中实现的事件循环函数

- ▶ ★ sql/conn_handler/connection_acceptor.h

```
61 void connection_event_loop() {  
62     Connection_handler_manager *mgr =  
63         Connection_handler_manager::get_instance();  
64     while (!connection_events_loop_aborted()) {  
65         Channel_info *channel_info = m_listener->listen_for_connection_event();  
66         if (channel_info != nullptr) mgr->process_new_connection(channel_info);  
67     }  
68 }
```



4

初始函数汇总



初始化流程（一）

- `substitute_progpath()` 将命令行的程序名替换成全路径的程序名
- `sysd::notify_connect()` 尝试连接该 Socket
- `my_init()` 初始化 `my_sys` 函数，资源和变量
- `load_defaults->my_load_defaults()` 从配置文件中读取配置项
- `persisted_variables_cache` 持久化的参数配置缓存，采用 JSON 进行存储，
- `init_variable_default_paths()` 初始化配置文件的作用域，参考 `enum_variable_source`
- `init_pfs_instrument_array()` PSI (performance schema interface) 性能仪表相关
- `handle_early_options()` 处理早期的选项，包括 `performance_schema` 和启动参数等
- `init_sql_statement_names()` 初始化与 sql 语句相关的名字
- `sys_var_init()` 初始化系统变量
- `init_error_log()` 初始化错误日志
- `adjust_related_options()` 调整相关的参数



初始化流程（二）

- `initialize_performance_schema()` 初始化 `performance_schema`
- `LO_init()` 初始化 Lock Order, 主要用 Lock Order Graph 描述锁依赖关系
- 启动 PSI 相关的服务

<code>thread_service</code>	<code> </code>	<code>mutex_service</code>	<code> </code>	<code>rwlock_service</code>	<code> </code>	<code>cond_service</code>
<code>file_service</code>	<code> </code>	<code>socket_service</code>	<code> </code>	<code>table_service</code>	<code> </code>	<code>mdl_service</code>
<code>idle_service</code>	<code> </code>	<code>stage_service</code>	<code> </code>	<code>statement_service</code>	<code> </code>	<code>transaction_service</code>
<code>memory_service</code>	<code> </code>	<code>error_service</code>	<code> </code>	<code>data_lock_service</code>	<code> </code>	<code>system_service</code>

- `init_server_psi_keys()` 注册 psi key 的函数
- `my_thread_global_reinit()` 重新初始化一些比较开始与锁相关的, 因为需要考虑一些仪表
- `component_infrastructure_init()` 初始化基础组件
- `register_pfs_notification_service()`
- `register_pfs_resource_group_service()`



初始化流程（三）

- `res_grp_mgr->init()` 初始化资源组管理器
- 调用 `psi_thread_service` 函数，参考 `key_thread_main`, `set_thread_os_id`, `set_thread`
- `mysql_audit_initialize()` 初始化与 `audit` 相关的变量
- `Srv_session::module_init()` 初始化 `srv_session` 模块
- `query_logger.init()` 查询日志初始化
- `init_common_variables()` 初始化通用变量
- `my_init_signals()` 初始化信号处理
- `Migrate_keyring::init()` 初始化 `Migrate_keyring`
- `Migrate_keyring::execute()` 执行 `Migrate_keyring`
- `set_ports()` 确定 `tcp` 端口
- `init_server_components()` 初始化服务端组件
- `Gtid_state::init()` 将 `server_uuid` 放至 `sid_map` 中
- `MYSQL_BIN_LOG::init_gtid_sets()` 从 `gtid_executed` 表和 `binlog` 文件中初始化参数



初始化流程（四）

- `init_ssl_communication()` 初始化 SSL
- `network_init()` 初始化服务的 listener
- `create_pid_file()` 创建 pid 文件
- `reload_optimizer_cost_constants()` 重新加载优化器的计算成本常量
- `mysql_component_infrastructure_init()` 通过初始化动态加载器来初始化 mysql 服务组件
- `mysql_rm_tmp_tables()` 删除临时文件
- `acl_init()` 初始化 ACL 访问控制
- `init_acl_memory()` 初始化 ACL 的内存
- `my_tz_init()` 初始化时区
- `grant_init()` 授权初始化
- `dynamic_privilege_init()` 动态权限初始化
- `servers_init()` 从 mysql 库中初始化结构数据，例如 `mysql.servers` 表的数据
- `udf_read_functions_table()` 从 `mysql.func` 中加载数据

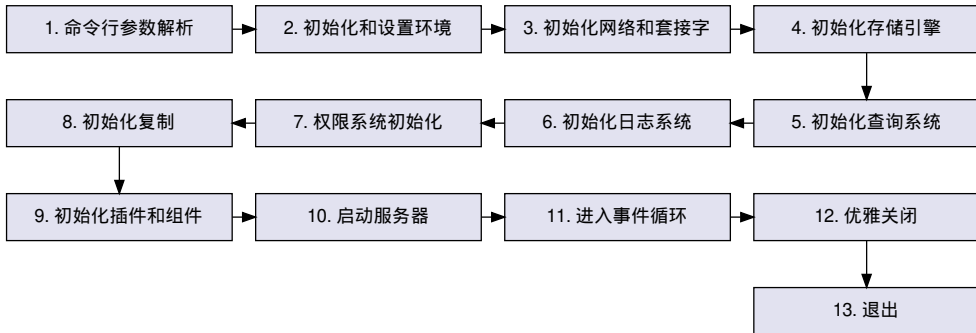


初始化流程（五）

- `init_status_vars()` 初始化状态变量
- `init_slave()` 初始化 slave 线程
- `initialize_performance_schema_acl()` 初始化 performance_schema 的 ACL
- `Events::init()` 初始化 event 结构
- `start_signal_handler()` 启动单独的线程 `signal_hand` 来处理信号
- `process_bootstrap()` 启动进程
- `mysql_audit_notify()` 审计相关
- `start_handle_manager()` 开启 handler 管理线程
- `Connection_acceptor::connection_event_loop()` 开启监听服务



流程总结



结束

