

# 第七讲：服务层组件线程管理和用户模块

知春路遇上八里桥

<2024-05-28 Tue>



① 前情提要

② 线程描述符

③ 代码分析

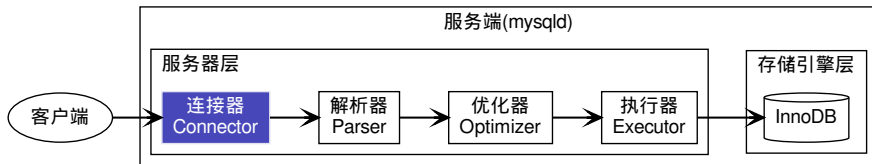


1

## 前情提要



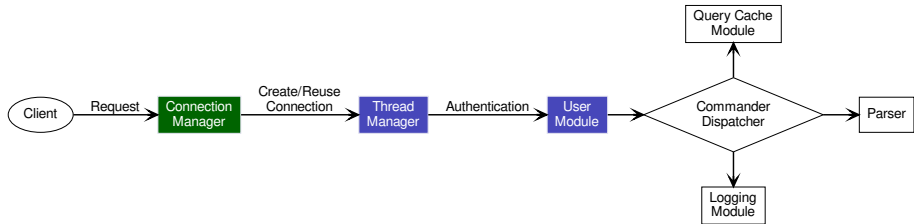
# 执行流程



# 本节内容

## • 连接器

- ▶ ☒ 连接管理器 Connection Manager
- ▶ ☐ 线程管理器 Thread Manager
- ▶ ☐ 用户模块 User Module



2

## 线程描述符



# 初识 THD 类

- THD<sup>1</sup> 是 MySQL 的线程描述符 (Thread Descriptor), 是代码中最重要的数据结构

- ▶ 类定义具体见文件 `* sql/sql_class.h` 的 928 至 4698 行, 共计 3770 行类定义

```
928  class THD : public MDL_context_owner,  
      ⋮  
4698  }; // End of class THD
```

- 它的继承关系如下

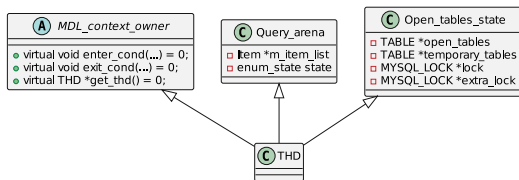
- ▶ MDL\_context\_owner 抽象类, 维护 Metadata Lock 拥有者的相关接口

- ① 元数据的边界控制 (进入/退出)

- ② 元数据的信息通知

- ▶ Query\_arena 维护语句中语法解析后的语法树节点

- ▶ Open\_tables\_state 维护线程打开/锁定的表信息



<sup>1</sup><https://dev.mysql.com/doc/dev/mysql-server/8.0.37/classTHD.html>

# THD 的使用场景

```
src/mysql-server » grep -R 'thd->' sql | wc -l
12787
src/mysql-server » grep -R 'thd->' sql | head -n 20
sql/mysqld_thd_manager.cc: if (thd->get_command() == COM_DAEMON && !m_daemon_allowed) return false;
sql/mysqld_thd_manager.cc: return (thd->thread_id() == m_thread_id);
sql/mysqld_thd_manager.cc: assert(thd->thread_id() != reserved_thread_id);
sql/mysqld_thd_manager.cc: const int partition = thd_partition(thd->thread_id());
sql/mysqld_thd_manager.cc: const int partition = thd_partition(thd->thread_id());
sql/mysqld_thd_manager.cc: assert(unit_test || thd->release_resources_done());
sql/window.cc: ORDER *o = new (thd->mem_root) PT_order_expr(nullptr, ORDER_ASC);
sql/window.cc: if (thd->lex->is_explain() && !m_frame->m_originally_absent &&
sql/window.cc:     thd->mem_root, o->value.size());
sql/window.cc: Item **left_args = new (thd->mem_root) Item *(nr);
sql/window.cc: Item **right_args = new (thd->mem_root) Item *(cmp_arg);
sql/window.cc: assert(thd->lex->current_query_block()->first_execution);
sql/window.cc: Query_block *curr = thd->lex->current_query_block();
sql/window.cc: const char *sav_where = thd->where;
sql/window.cc: thd->where = partition_order ? "window partition by" : "window order by";
sql/window.cc: my_error(ER_BAD_FIELD_ERROR, MYF(0), oi->item_name.ptr(), thd->where);
sql/window.cc: if (thd->is_error()) return true;
sql/window.cc: thd->where = sav_where;
sql/window.cc: char const *save_where = thd->where;
sql/window.cc: thd->where = "window frame bound";
src/mysql-server »
```





# THD 介绍 - 第一部分

C THD	
系统变量	
o	struct System_variables variables
o	struct System_status_var status_var
o	struct System_status_var *initial_status_var
内部属性	
o	ulonglong start_utime
o	query_id_t query_id
□	my_thread_id m_thread_id
o	MDL_context mdl_context
o	THR_LOCK_INFO lock_info
指令类型	
□	enum enum_server_command m_command
o	void set_new_thread_id()
o	my_thread_id thread_id()
o	void set_command(enum enum_server_command command)
o	inline enum enum_server_command get_command()

- Session 级别的变量信息
  - ▶ variables 系统变量
  - ▶ status\_var 状态变量
- 内部属性
  - ▶ start\_utime 线程开始的时间戳
  - ▶ query\_id 当前 Query 的唯一 ID



# 用户变量

- THD 中还存储着一些业务相关的变量，例如用户自定义的变量

```
SET @id:=123;  
SELECT * FROM employee WHERE emp_no = @id;
```

- 在 `★ sql/sql_class.h` 中定义了 HASH `user_vars` 进行存储

```
1098  /**  
1099      Hash for user variables.  
1100      User variables are per session,  
1101      but can also be monitored outside of the session,  
1102      so a lock is needed to prevent race conditions.  
1103      Protected by @c LOCK_thd_data.  
1104  */  
1105  collation_unordered_map<std::string, unique_ptr_with_deleter<user_var_entry>>  
1106      user_vars{system_charset_info, key_memory_user_var_entry};
```



## THD 介绍 - 第二部分

Ⓒ THD
Query
<ul style="list-style-type: none"><li>LEX_CSTRING m_query_string</li><li>LEX_CSTRING m_db</li><li>LEX_CSTRING m_catalog</li><li>String m_rewritten_query</li></ul>
词法/语法分析
<ul style="list-style-type: none"><li>LEX *lex</li><li>Item_change_list change_list</li><li>Query_arena *stmt_arena</li><li>Parser_state *m_parser_state</li><li>Locked_tables_list locked_tables_list</li></ul>
<ul style="list-style-type: none"><li>const LEX_CSTRING &amp;query()</li><li>void set_query(LEX_CSTRING query_arg)</li><li>LEX_CSTRING &amp;db()</li><li>bool set_db(const LEX_CSTRING &amp;new_db)</li></ul>

- 当前 THD 的原始接收的字符串数据
  - m\_query\_string 记录客户端发送的 Query 字符串
  - m\_db 当前的数据库
  - m\_rewritten\_query 记录 rewritten 功能修改后的 Query 字符串
- 词法分析和语法分析的记录变量

# THD 介绍 - 第三部分

Ⓒ THD
网络
<ul style="list-style-type: none"><li>NET net</li><li>String packet</li><li>bool m_is_admin_conn</li></ul>
协议
<ul style="list-style-type: none"><li>Protocol *m_protocol</li><li>std::unique_ptr&lt;Protocol_text&gt; protocol_text</li><li>std::unique_ptr&lt;Protocol_binary&gt; protocol_binary</li></ul>
连接信息
<ul style="list-style-type: none"><li>LEX_CSTRING m_invoker_user</li><li>LEX_CSTRING m_invoker_host</li></ul>
<ul style="list-style-type: none"><li>NET *get_net()</li><li>void set_admin_connection(bool admin)</li><li>bool is_admin_connection()</li><li>const Protocol *get_protocol()</li></ul>

- 网络传输相关变量，数据包
- 当前连接是否为管理员连接方式
- 传输的协议 Protocol
  - ▶ 传统协议 Protocol\_classic
  - ▶ 文本协议 Protocol\_text
  - ▶ 二进制协议 Protocol\_binary



# THD 介绍 - 第四部分

C THD
性能追踪
<ul style="list-style-type: none"><li>o class Query_plan query_plan</li><li>o Opt_trace_context opt_trace</li></ul>
优化器
<ul style="list-style-type: none"><li>o double m_current_query_cost</li><li>o Cost_model_server m_cost_model</li></ul>
执行器
<ul style="list-style-type: none"><li>o ulonglong previous_found_rows</li><li>o ulonglong current_found_rows</li></ul>
<ul style="list-style-type: none"><li>o void lock_query_plan()</li><li>o void unlock_query_plan()</li><li>o void init_cost_model()</li><li>o const Cost_model_server *cost_model()</li><li>o void clear_current_query_costs()</li><li>o void save_current_query_costs()</li></ul>

- 性能追踪工具
  - ▶ query\_plan 执行计划
  - ▶ opt\_trace Optimize Trace 日志
- 优化器
  - ▶ m\_cost\_model cost 代价模型
  - ▶ m\_current\_query\_cost 当前 query 的 cost 值



# Explain/Optimize Trace 举例

- 使用 explain 查看执行计划

```
explain format=json SELECT emp_no FROM employees WHERE emp_no < 11111\G
```

- 使用 Optimize Trace 查看执行计划的计算细节

```
-- 开启 trace
```

```
set optimizer_trace="enabled=on";
```

```
-- 执行 Query
```

```
select emp_no from employees where emp_no < 11111;
```

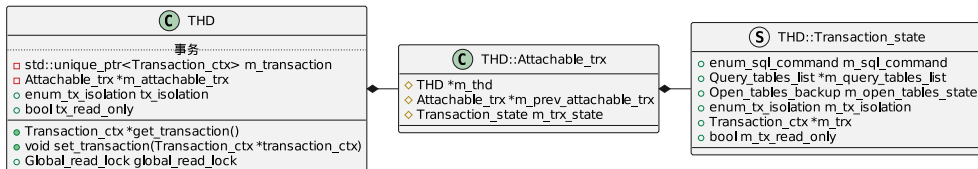
```
-- 查看结果
```

```
select * from information_schema.optimizer_trace\G
```

```
select trace from information_schema.optimizer_trace\G
```



# THD 介绍 - 第五部分



- **m\_transaction 当前事务指针**
  - ▶ tx\_isolation 事务隔离级别
  - ▶ tx\_read\_only 事务是否只读
- **m\_attachable\_trx 可附加事务**
- **事务状态**
  - ▶ m\_query\_tables\_list Query 表列表
  - ▶ m\_tx\_isolation 事务隔离级别
  - ▶ m\_tx\_read\_only 事务是否只读

# THD 介绍 - 更多

## ① 各种锁，在并发访问中保护 THD 中对象

- ▶ `mysql_mutex_t LOCK_thd_data`
- ▶ `mysql_mutex_t LOCK_thd_query`
- ▶ `mysql_mutex_t LOCK_thd_sysvar`
- ▶ `mysql_mutex_t LOCK_thd_protocol`
- ▶ ...

## ② 日志/主从复制相关信息

- ▶ `binlog/Gtid`

## ③ 处理过程中上下文

- ▶ 资源组 (resource group)
- ▶ 授权对象 (security context)

## ④ 与客户端交互的 IO 相关 `vio`

## ⑤ 执行过程中的数据

- ▶ 数据库对象的字典, `schema`, `table`, `tablespace`, `store program`
- ▶ 分区信息, `partition`

## ⑥ 存储引擎的一些数据

## ⑦ ...





# THD 的管理器 - Global\_THD\_manager

## Global\_THD\_manager

- static const my\_thread\_id reserved\_thread\_id
  - static Global\_THD\_manager \*thd\_manager
  - THD\_array thd\_list[NUM\_PARTITIONS]
  - Thread\_id\_array thread\_ids
  - mysql\_mutex\_t LOCK\_thread\_ids
- 
- static Global\_THD\_manager \*get\_instance()
  - void add\_thd(THD \*thd)
  - void remove\_thd(THD \*thd)

- 单例类，通过 get\_instance() 全局唯一的实例
- 维护 THD 列表，提供基本的管理操作
- 列表的操作通过 LOCK\_thread\_ids 锁保护



3

## 代码分析



# 进入 handle\_connection 函数

- mysql\_thread\_create() 创建线程

- ▶ ★ sql/conn\_handler/connection\_handler\_per\_thread.cc

```
415 mysql_thread_create(key_thread_one_connection, &id, &connection_attrib,  
416                     handle_connection, (void *)channel_info);
```

- ▶ 设置启动函数 handle\_connection()
- ▶ 设置启动函数入参 channel\_info

- handle\_connection() 函数实现逻辑

- ▶ ★ sql/conn\_handler/connection\_handler\_per\_thread.cc

```
246 static void *handle_connection(void *arg) {  
247     Global_THD_manager *thd_manager = Global_THD_manager::get_instance();  
248     Connection_handler_manager *handler_manager =  
249         Connection_handler_manager::get_instance();  
250     Channel_info *channel_info = static_cast<Channel_info *>(arg);  
251     bool pthread_reused [[maybe_unused]] = false;
```

- ▶ 通过强转获取入参 channel\_info
- ▶ 后续逻辑 ...

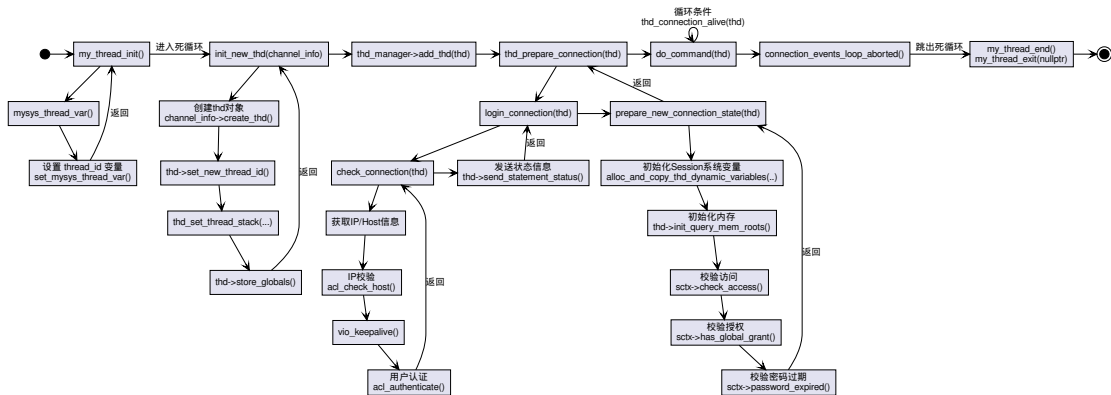


# handle\_connection 入口函数功能

- ❶ 初始化线程
  - ▶ 初始化内存
  - ▶ 设置 thread\_id 变量
- ❷ 初始化当前线程的 THD 对象
  - ▶ 新建并初始化 thd 实例的属性
- ❸ 用户认证
- ❹ 执行当前用户发送的所有的 Query
  - ▶ do\_command()
- ❺ 关闭连接
  - ▶ end\_connection(thd)
  - ▶ close\_connection(thd, 0, false, false)
- ❻ 结束线程, 清理下次重连是需要的缓存
  - ▶ my\_thread\_end()
  - ▶ my\_thread\_exit(nullptr)



# handle\_connection 代码流程图



# 用户认证模块

❶ 位于 `sql/auth` 目录下

❷ `acl_check_host()` 检查 `host` 下是否匹配用户，如果没有返回 `true`

▶ `sql/auth/sql_authentication.cc`

```
2023 bool acl_check_host(THD *thd, const char *host, const char *ip) {  
2024     Acl_cache_lock_guard acl_cache_lock(thd, Acl_cache_lock_mode::READ_MODE);  
2025     if (!acl_cache_lock.lock(false)) return true;
```

❸ `acl_authenticate()` 执行握手，认证客户端，更新 `thd` 的 `sctx` 变量

▶ `sql/auth/sql_authentication.cc`

```
3826 int acl_authenticate(THD *thd, enum_server_command command) {  
3827     int res = CR_OK;  
3828     int ret = 1;  
3829     MPVIO_EXT mpvio;  
3900     :  
4256 end:  
4257     if (mpvio.restrictions) mpvio.restrictions->~Restrictions();  
4258     /* Ready to handle queries */  
4259     return ret;  
4260 }
```



# Security\_context 类

- Security\_context 类定义位于，通常写成 sctx->xxx，见 ★ sql/auth/sql\_security\_ctx.h

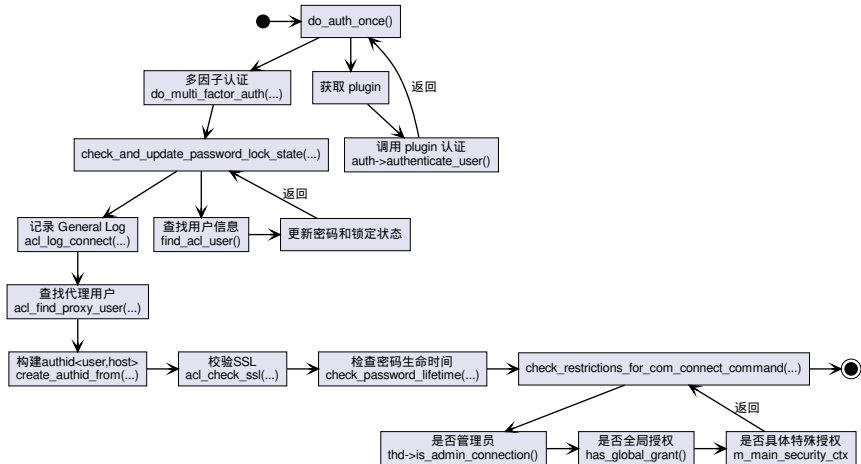
```
54 class Security_context {  
55     public:  
56         Security_context(THD *thd = nullptr);  
57         ~Security_context();
```

- 描述当前认证的用户信息，具体类见下图

Security_context
<ul style="list-style-type: none"><li>String m_user</li><li>String m_host</li><li>String m_ip</li><li>String m_host_or_ip</li><li>String m_external_user</li><li>char m_proxy_user[USERNAME_LENGTH + HOSTNAME_LENGTH + 6]</li><li>bool m_is_locked</li></ul>
<ul style="list-style-type: none"><li>void skip_grants(...)</li><li>LEX_CSTRING user()</li><li>LEX_CSTRING host()</li><li>LEX_CSTRING priv_user()</li><li>bool check_access()</li><li>bool password_expired()</li><li>void assign_user(...)</li><li>void set_master_access(...)</li><li>int activate_role(...)</li><li>void cache_current_db_access(ulong db_access)</li></ul>



# 用户认证代码分析





# 结束

