第七讲：服务层组件线程管理和用户模块

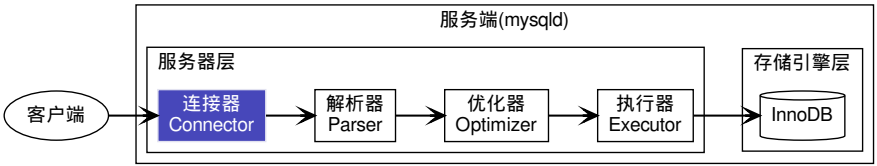
知春路遇上八里桥

<2024-05-28 Tue>

1

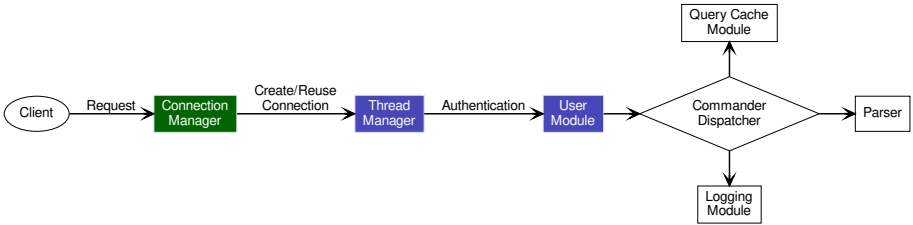
前情提要

执行流程


服务端(mysqld)
服务器层
存储引擎层
客户端
连接器
Connector
解析器
Parser
优化器
Optimizer
执行器
Executor
InnoDB

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

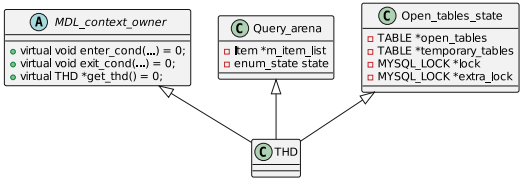
线程描述符

初识 THD 类

- THD [1] 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

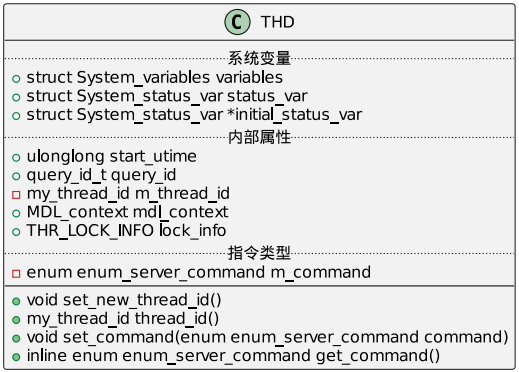


[1] https://dev.mysql.com/doc/dev/mysql-server/8.0.37/classTHD.html

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



- 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

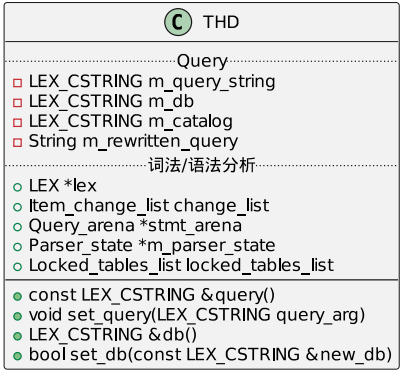
- THD 中还存储着一些业务相关的变量，例如用户自定义的变量

```
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

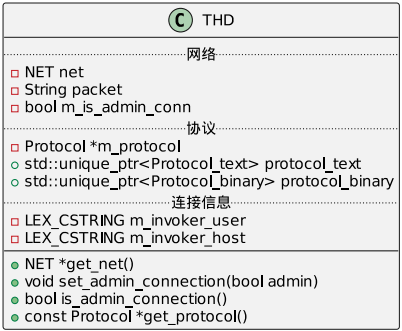
```
1098 /**
1099   Hash for user variables.
1100   User variables are per session,
1101   but can also be monitored outside of the session,
1102   so a lock is needed to prevent race conditions.
1103   Protected by @c LOCK_thd_data.
1104 */
1105 collation_unordered_map<std::string, unique_ptr_with_deleter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

THD 介绍 - 第二部分



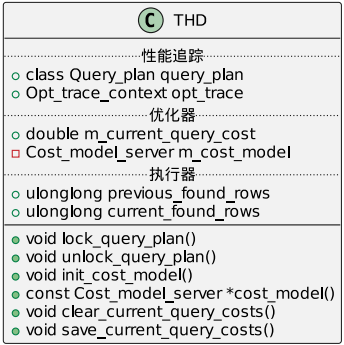
- 当前 THD 的原始接收的字符串数据
  - `m_query_string` 记录客户端发送的 Query 字符串
  - `m_db` 当前的数据库
  - `m_rewritten_query` 记录 rewritten 功能修改后的 Query 字符串
- 词法分析和语法分析的记录变量

THD 介绍 - 第三部分



- 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 Protocal
  - 传统协议 `Protocal_classic`
  - 文本协议 `Protocal_text`
  - 二进制协议 `Protocal_binary`

THD 介绍 - 第四部分



- 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

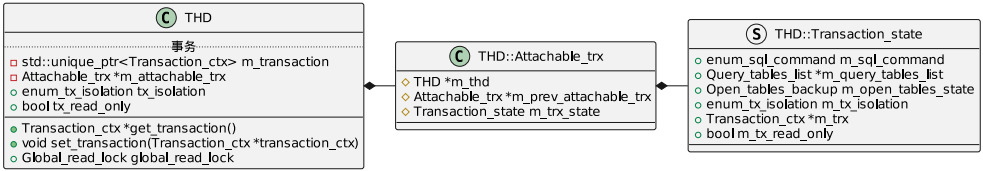
- 使用 explain 查看执行计划

```
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

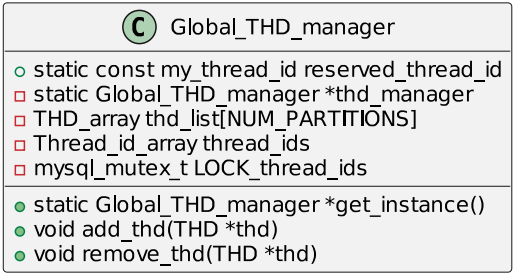


- `m_transaction` 当前事务指针
  - `tx_isolation` 事务隔离级别
  - `tx_read_only` 事务是否只读
- `m_attachable_trx` 可附加事务
- 事务状态
  - `m_query_tables_list` Query 表列表
  - `m_tx_isolation` 事务隔离级别
  - `m_tx_read_only` 事务是否只读

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



- 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

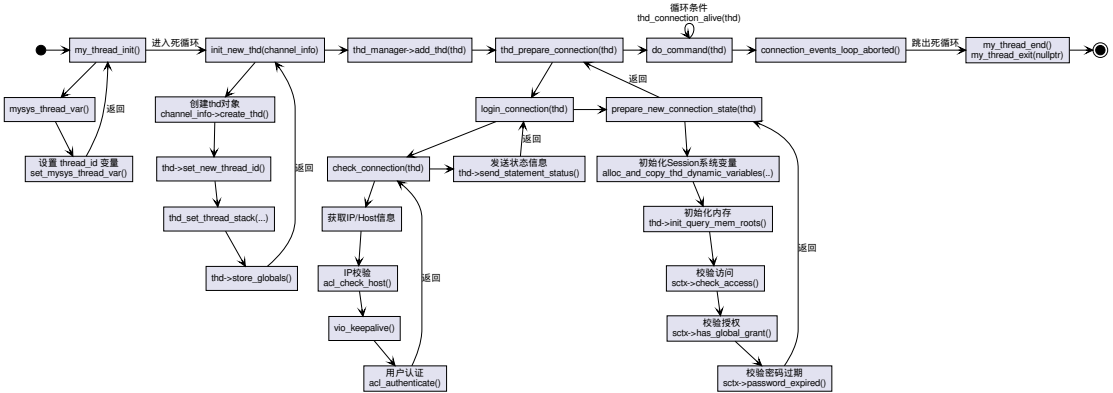
```
246  static void *handle_connection(void *arg) {
247    Global_THD_manager *thd_manager = Global_THD_manager::get_instance();
248    Connection_handler_manager *handler_manager =
249        Connection_handler_manager::get_instance();
250    Channel_info *channel_info = static_cast<Channel_info *>(arg);
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

1. 初始化线程
   - 初始化内存
   - 设置 thread_id 变量
2. 初始化当前线程的 THD 对象
   - 新建并初始化 thd 实例的属性
3. 用户认证
4. 执行当前用户发送的所有的 Query
   - do_command()
5. 关闭连接
   - end_connection(thd)
   - close_connection(thd, 0, false, false)
6. 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

1. 位于 ⋆ sql/auth 目录下
2. 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;
```

3. acl_authenticate() 执行握手，认证客户端，更新 thd 的 sctx 变量
   - ⋆ sql/auth/sql_authentication.cc

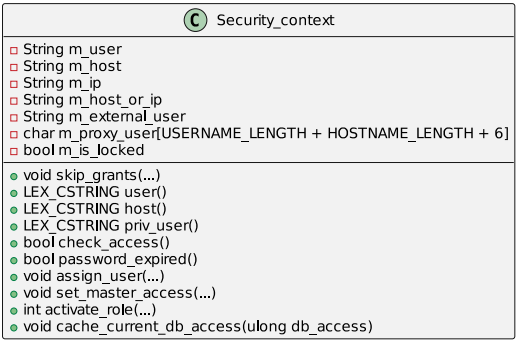
```
3826 int acl_authenticate(THD *thd, enum_server_command command) {
3827   int res = CR_OK;
3828   int ret = 1;
3829   MPVIO_EXT mpvio;
  ⋮
4256 end:
4257   if (mpvio.restrictions) mpvio.restrictions->~Restrictions();
4258   /* Ready to handle queries */
4259   return ret;
4260 }
```

Security_context 类

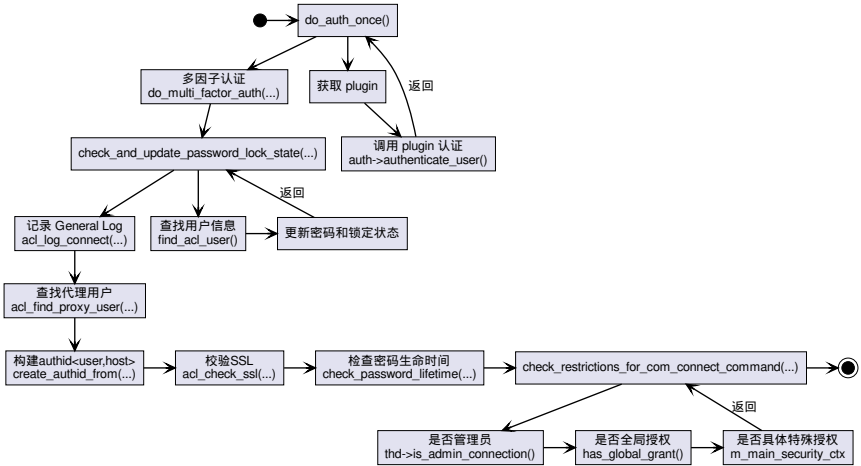
- Security_context 类定义位于，通常写成 `sctx->xxx` ，见 ⋆ sql/auth/sql_security_ctx.h

```
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



用户认证代码分析


do_auth_once()
多因子认证
do_multi_factor_auth(...)
获取 plugin
返回
check_and_update_password_lock_state(...)
调用 plugin 认证
auth->authenticate_user()
返回
记录 General Log
acl_log_connect(...)
查找用户信息
find_acl_user()
更新密码和锁定状态
查找代理用户
acl_find_proxy_user(...)
构建authid<user,host>
create_authid_from(...)
校验SSL
acl_check_ssl(...)
检查密码生命时间
check_password_lifetime(...)
check_restrictions_for_com_connect_command(...)
返回
是否管理员
thd->is_admin_connection()
是否全局授权
has_global_grant()
是否具体特殊授权
m_main_security_ctx

结束