第九讲：MySQL 词法分析器的设计与实现

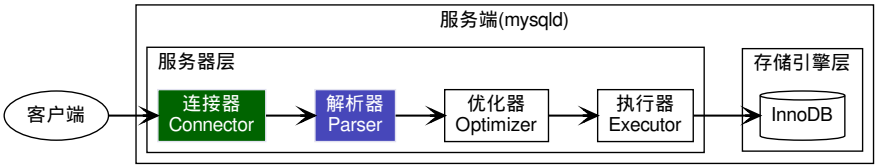
知春路遇上八里桥

<2024-06-04 Tue>

1

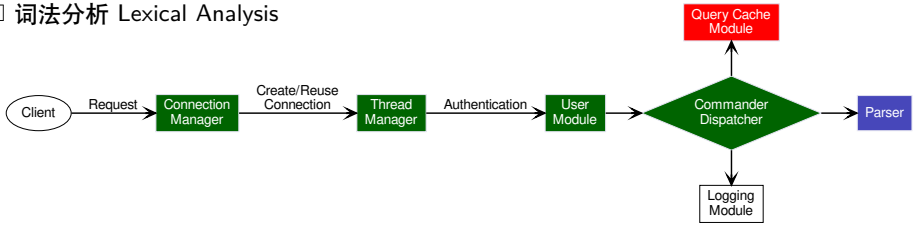
前情提要

执行流程


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

本节内容

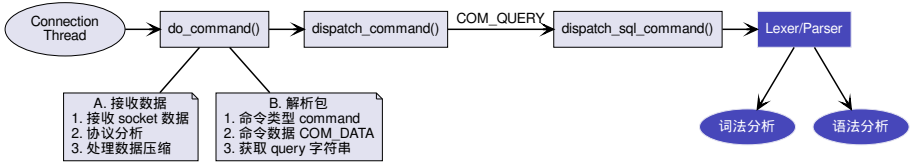
- 连接器
  - ☑ 连接管理器 Connection Manager
  - ☑ 线程管理器 Thread Manager
  - ☑ 用户模块 User Module
- 解析器
  - ☑ 网络模块 Net Module
  - ☑ 派发模块 Commander Dispatcher
  - ☐ 词法分析 Lexical Analysis



2

启动流程

从 do_command 进入解析器



- 通过 `dispatch_sql_command()` 函数进行 Query 字符串的解析
  - 可以通过 `thd->query()` 方法访问解析后的 Query 字符串
- MySQL 的解析代码实现的基本认知 [1]
  - 词法分析是通过编写代码实现
  - 语法分析借助 bison 工具实现

[1] flex/yacc 是传统词法分析和语法分析的工具

bison 语法格式

- bison 语法分析输入文件 ⋆ sql/sql_yacc.yy ， 其格式如下：

```
32    %{
 ⋮        第一部分：C 语言声明部分
513   %}
 ⋮        第二部分：BISON 声明部分
2203  %%
 ⋮        第三部分：语法规则声明
```

- bison 语法：
  - 声明格式 `%token`, `%union`, `%type`, `%left`, `%right`, `%nonassoc`
  - 语法规制定义格式

```
result: components ...
    ;
```

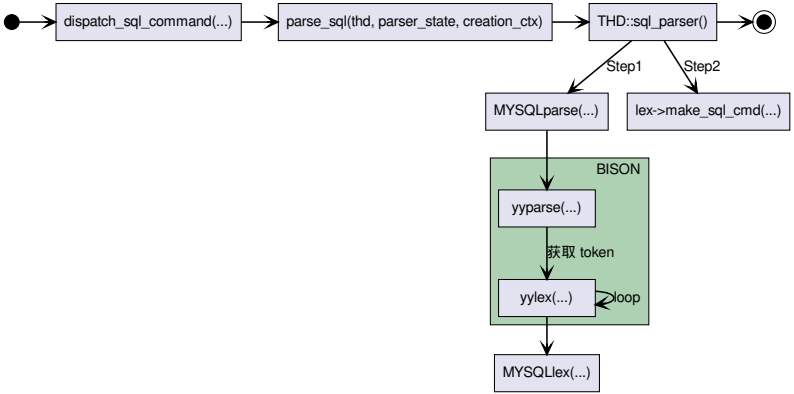
  - 重要的函数：`yylval`, `YYSTYPE`, `yyerror()`, `yyparse()`

MySQL 使用 bison 的方式

- 代码生成 ⋆ sql/sql_yacc.yy ⇒ ⋆ build/sql/sql_yacc.cc
  - 调用词法分析的入口函数：`yylex(...)` 函数
  - 提供语法分析的入口函数：`yyparse(...)` 函数
  - 将生成的代码和其他 `.o` 文件链接到一起生成 mysqld
- 在文件 ⋆ build/sql/sql_yacc.cc 中进行关键函数替换

```
/* Substitute the variable and function names.  */
#define yyparse         MYSQLparse
#define yylex           MYSQLlex
#define yyerror         MYSQLerror
#define yydebug         MYSQLdebug
#define yynerrs         MYSQLnerrs
```

词法分析和语法分析调用关系



- 在文件 ⋆ sql/sql_parse.cc 中通过 `dispatch_sql_command()` 调用

```
5267  err = parse_sql(thd, parser_state, nullptr);
```

3

词法分析

词法分析的实现

1. MySQL 的词法分析的主要实现见文件 ⋆ sql/sql_lex.cc
2. 词法分析入口函数 `MYSQLlex()`
3. `lex_one_token()` 获取一个 token
4. `find_keyword()` 查找关键字
5. `consume_optimizer_hints()` 消费 Optimizer Hits
6. `HINT_PARSER_parse()` 解析 hint

词法分析的状态机

1. 词法分析状态，通过枚举类定义词法分析状态机的所有状态
   - 见文件 ⋆ include/sql_chars.h

```
38  enum MY_ATTRIBUTE((__packed__)) my_lex_states {
39    MY_LEX_START,
40    MY_LEX_CHAR,
41    MY_LEX_IDENT,
```

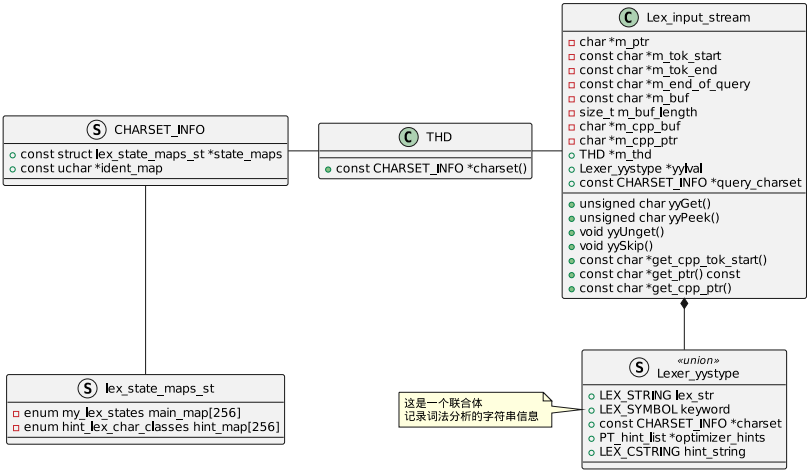
2. state_map[] 表示字符到词法分析的状态映射
   - 构造 256 个字符到词法状态的迁移 Map，key 是字符，value 是词法状态
   - 见文件 ⋆ mysys/sql_chars.cc

```
68  bool init_state_maps(CHARSET_INFO *cs) {
69    uint i;
70    uchar *ident_map;
71    enum my_lex_states *state_map = nullptr;
```

3. lex_one_token() 实现状态机，获取 token ，完成词法分析的解析工作
   - 见文件 ⋆ sql/sql_lex.cc

```
1370  static int lex_one_token(Lexer_yystype *yylval, THD *thd) {
1371    uchar c = 0;
1372    bool comment_closed;
```

输入流及其关联的类


Lex_input_stream
char *m_ptr
const char *m_tok_start
const char *m_tok_end
const char *m_end_of_query
const char *m_buf
size_t m_buf_length
char *m_cpp_buf
char *m_cpp_ptr
THD *m_thd
Lexer_yystype *yylval
const CHARSET_INFO *query_charset
unsigned char yyGet()
unsigned char yyPeek()
void yyUnget()
void ySkip()
const char *get_cpp_tok_start()
const char *get_ptr() const
const char *get_cpp_ptr()
CHARSET_INFO
const struct lex_state_maps_st *state_maps
const uchar *ident_map
THD
const CHARSET_INFO *charset()
lex_state_maps_st
enum my_lex_states main_map[256]
enum hint_lex_char_classes hint_map[256]
«union»
Lexer_yystype
LEX_STRING lex_str
LEX_SYMBOL keyword
const CHARSET_INFO *charset
PT_hint_list *optimizer_hints
LEX_CSTRING hint_string
这是一个联合体
记录词法分析的字符串信息

lex_one_token 函数代码分析

1. Lex_input_stream 词法分析的输入流，实现了对输入字符流的基本操作
2. 重要变量如下：
   - const CHARSET_INFO *cs 实现多种字符集的支持
   - const my_lex_states *state_map 实现状态印射
   - yylval 字符联合体，记录实际解析的字符串
   - tokval 是一个 token 整数，作为函数的返回值
   - ⋆ sql/sql_lex.cc

```
static int lex_one_token(Lexer_yystype *yylval, THD *thd) {
  // 省略
  state = lip->next_state;
  lip->next_state = MY_LEX_START;
  for (;;) {
    switch (state) {
      case MY_LEX_START:  // Start of token
        // Skip starting whitespace
        // 省略
        break;
        // 省略
```

token 数值的定义

- token < 256 直接使用字符串的 ASCII 字符数值
  - 这部分见 init_state_maps() 函数
- token >= 256 在 bison 文件中定义后，经过编译会生成 C 语言代码
  - ⋆ sql/sql_yacc.yy

```
%token<lexer.keyword> FORMAT_SYM 450
%token<lexer.keyword> FOUND_SYM 451              /* SQL-2003-R */
%token  FROM 452
...
%token<lexer.keyword> IDENTIFIED_SYM 483
%token  IDENT_QUOTED 484
%token  IF 485
%token  IGNORE_SYM 486
...
%token<lexer.keyword> SECURITY_SYM 747           /* SQL-2003-N */
%token  SELECT_SYM 748                     /* SQL-2003-R */
%token  SENSITIVE_SYM 749                  /* FUTURE-USE */
%token  SEPARATOR_SYM 750
```

自动生成 token 数组

- 所有的 token 数值通过代码自动收集生成
  - ⋆ sql/gen_lex_token.cc

```
As of now (8.0.0), the mapping looks like this:
- PART 1: [0 .. 255] tokens of single-character lexemes
- PART 2: [256 .. ...] tokens < YYUNDEF from sql_yacc.yy
- PART 3: [... .. 999] reserved for sql_yacc.yy new tokens < YYUNDEF
- PART 4: [1000 .. ...] tokens from sql_hints.yy
- PART 5: [... .. 1099] reserved for sql_hints.yy new tokens
- PART 6: [1100 .. ...] digest special fake tokens
- PART 7: [... .. 1149] reserved for new digest special fake tokens
- PART 8: [1150 .. ...] tokens > YYUNDEF from sql_yacc.yy
```

- 经过编译会生成代码，其中 lex_token_array[] 记录了所有 token 对应关系
  - ⋆ build/sql/lex_token.h

```
lex_token_string lex_token_array[]=
{
/* PART 1: character tokens. */
/* 000 */  { "\x00", 1, true, false},
/* 001 */  { "\x01", 1, true, false},
```

4

追溯日志

token 流过程追溯

- 测试的 SQL

```
mysql> select first_name, last_name from employees where emp_no < 100;
Empty set (0.00 sec)
```

- 输出的 token 流，可以看出输入字符串被转换成 <token, lexeme> 二元组

```
parse_sql: rtc: scan token=748, lexeme=[select ]
parse_sql: rtc: scan token=484, lexeme=[first_name,]
parse_sql: rtc: scan token=44, lexeme=[,]
parse_sql: rtc: scan token=484, lexeme=[last_name ]
parse_sql: rtc: scan token=452, lexeme=[from ]
parse_sql: rtc: scan token=484, lexeme=[employees ]
parse_sql: rtc: scan token=890, lexeme=[where ]
parse_sql: rtc: scan token=484, lexeme=[emp_no ]
parse_sql: rtc: scan token=549, lexeme=[<]
parse_sql: rtc: scan token=628, lexeme=[100]
parse_sql: rtc: scan token=411, lexeme=[]
parse_sql: rtc: scan token=0, lexeme=[]
```

开启 bison 追溯日志

- 开启调试选项

  `debug=d,parser_debug`

- 通过阅读 `sql_yacc.yy` 中可知查看 bison 日志的方式

```
The syntax to run with bison traces is as follows :
- Starting a server manually :
  mysqld --debug="d,parser_debug" ...
- Running a test :
  mysql-test-run.pl --mysqld="--debug=d,parser_debug" ...

The result will be in the process stderr (var/log/master.err)
```

结束