Day2 of OceanBase 源码解析： 函数与表达式

一、Overview

> 这篇文章目的是通过 issues_11055
>
> 了解 函数表达式是怎么执行的 ，
>
> 先看SELECT MONTHNAME("2017-06-15"); 怎么实现的，然后仿照写。

DAYNAME() function in mysql mode not supported #1055 C | 复制代码

```
https://github.com/oceanbase/oceanbase/issues/1055
obclient [test]> SELECT DAYNAME("2017-06-15");
ERROR 1305 (42000): FUNCTION DAYNAME does not exist
》
```

本文主要对 下面这一句话的理解

在前面《 OceanBase 源码解读（三）SQL 的一生》中讲过，SQL 语句在生成抽象语法树后，经过 resolver 转换解析为 stmt 逻辑执行计划，其中的一个关键部分是表达式的解析。

表达式解析类是位于sql/resolver/expr 的ObRawExprResolver，它输入ObParseNode 树，输出表达式树 ObRawExpr

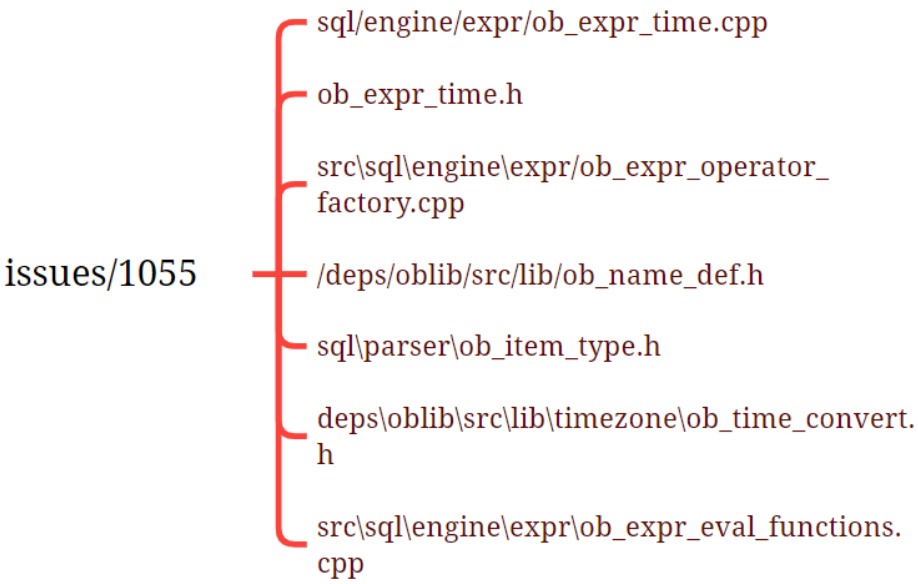
思考 60秒 写下的疑问

1. 这个函数有什么用?
   - Return the weekday name for a date:
2. 表达式 类型如如何注册的，处理类是如何管理的?
3. parer模块解析的 怎么和具体处理函数关联?
4. 表达式内置函数是什么意思?
5.

表达式内置函数 返回计算类型calc_result_type1 参数 type ， type1 这2个有什么区别没看懂?

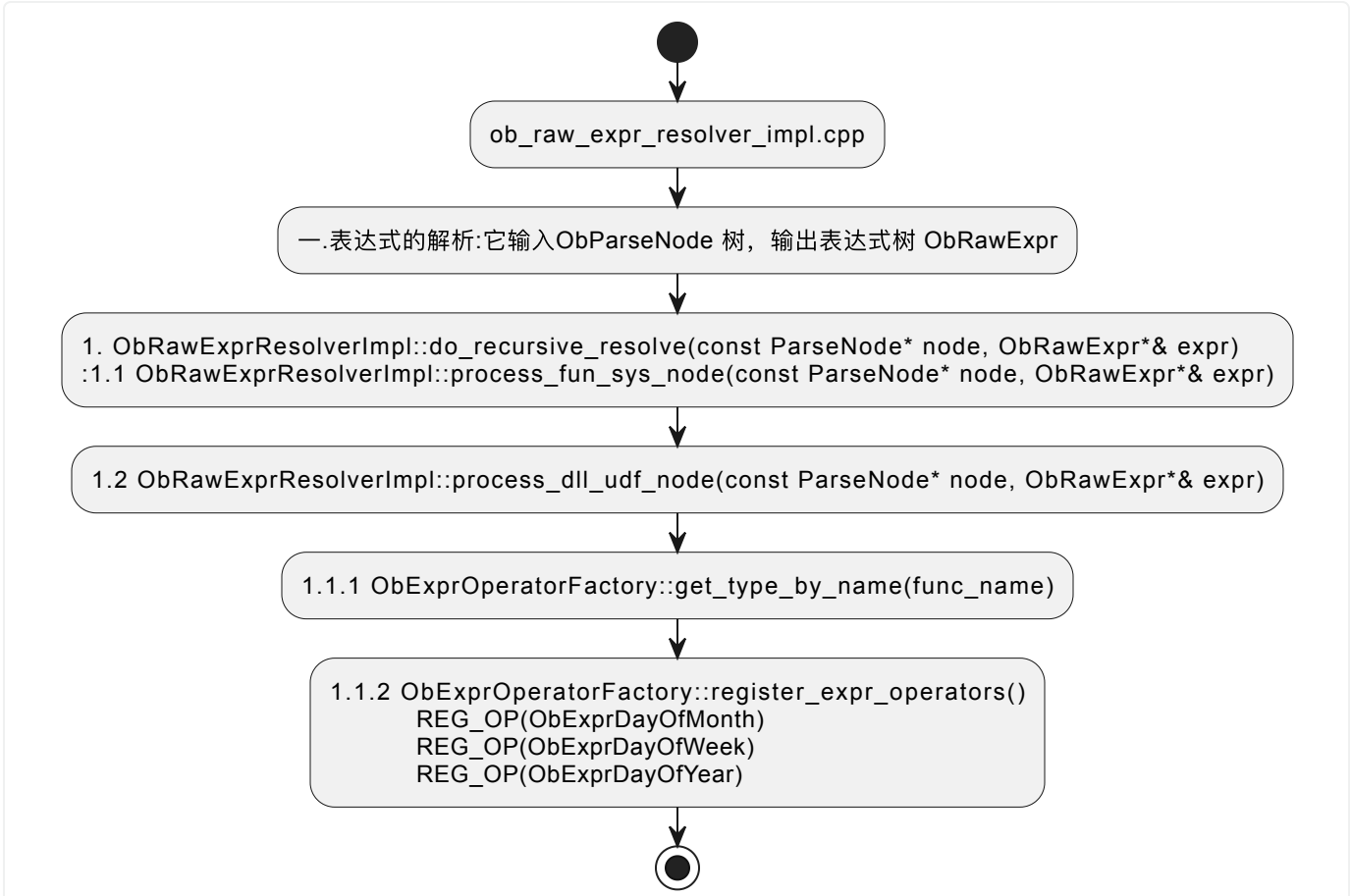
代码位置：ObExprMonthName::calc_result_type1(ObExprResType& type, ObExprResType& type1,)

尝试回答

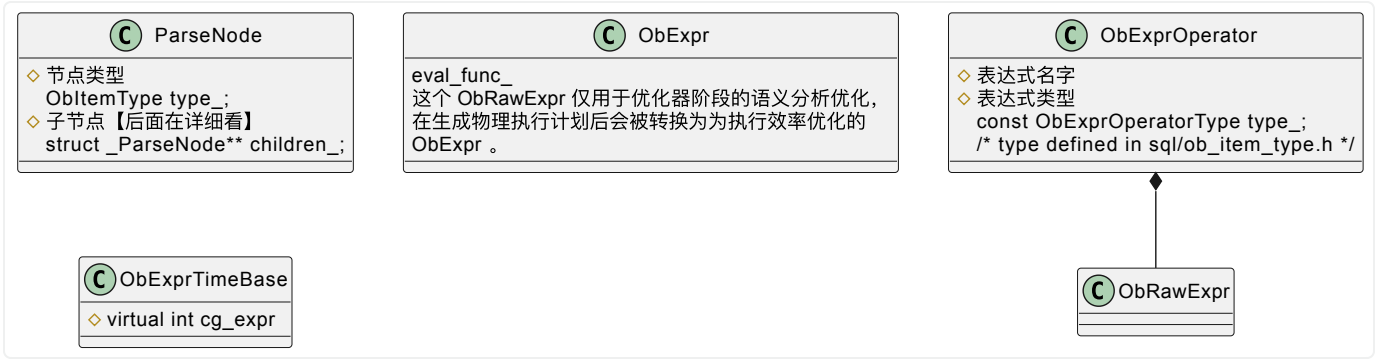

issues/1055
sql/engine/expr/ob_expr_time.cpp
ob_expr_time.h
src\sql\engine\expr/ob_expr_operator_
factory.cpp
/deps/oblib/src/lib/ob_name_def.h
sql\parser\ob_item_type.h
deps\oblib\src\lib\timezone\ob_time_convert.
h
src\sql\engine\expr\ob_expr_eval_functions.
cpp

二、MVP

2.1 流程图



– 类图



2.2 整体流程

- parser阶段： func(arg1, arg2) 类型的表达式函数 统一parser 成T_FUN_SYS 类型，函数名存在子结点里

- 【resolver 模块】: 根据节点 T_FUN_SYS 类型 执行 process_fun_sys_node函数

- 【resolver 模块】: 根据函数名字查找类型: ObExprOperatorFactory::get_type_by_name(func_name)

- 【ob_expr_operator_factory.cpp】表达式名字 和类型 通过自定义实现类 提前注册REG_OP

REG_OP Plain Text 复制代码

```
#define REG_OP(OpClass)                                                 \
  do {                                                                  \
    OpClass op(alloc);                                                  \
    if (OB_UNLIKELY(i >= EXPR_OP_NUM)) {                                \
      LOG_ERROR("out of the max expr");                                 \
    } else {                                                            \
      NAME_TYPES[i].name_ = op.get_name();                              \
      NAME_TYPES[i].type_ = op.get_type();                              \
      OP_ALLOC[op.get_type()] = ObExprOperatorFactory::alloc<OpClass>; \
      i++;                                                              \
    }                                                                   \
  } while (0)
```

- 【ob_expr_time.cpp】自定义表达式类：在定义构造时候 初始化了 表达式的名字 和类型【写死了】

Plain Text 复制代码

```
ObExprDayName::ObExprDayName(ObIAllocator& alloc)
    : ObExprTimeBase(alloc, DT_DAY_NAME, T_FUN_SYS_DAY_NAME, N_DAY_NAME){};
#define N_DAY_NAME "dayname"
```

- [表达式内置函数] virtual cg_expr 注册自己实现函数到 eval_func_
- 【表达式内置函数】提供函数注册是的内存分配，类型定义

calc_result_type1 返回值的类型

calc_result_type2 返回值的类型

看到这里可用了 下面个人测试 和阅读记录

2.3 详细描述

ob_raw_expr_resolver_impl.cpp

ObRawExprResolverImpl::do_recursive_resolve C 复制代码

```
#781
 case T_FUN_SYS: {
          if (OB_FAIL(process_fun_sys_node(node, expr))) {
            if (ret != OB_ERR_FUNCTION_UNKNOWN) {
              LOG_WARN("fail to process system function node", K(ret), K(node));
            } else if (OB_FAIL(process_dll_udf_node(node, expr))) {
              if (ret != OB_ERR_FUNCTION_UNKNOWN) {
                LOG_WARN("fail to process dll user function node", K(ret), K(node));
              } else {
                ParseNode* udf_node = NULL;
                ObString empty_db_name;
                ObString empty_pkg_name;
                bool record_udf_info = true;
                if (OB_FAIL(ObResolverUtils::transform_func_sys_to_udf(
                        &ctx_.expr_factory_.get_allocator(), node, empty_db_name, empty_pkg_name, udf_node))) {
                  LOG_WARN("transform func sys to udf node failed", K(ret));
                } else if (OB_ISNULL(expr)) {
                  ret = OB_ERR_FUNCTION_UNKNOWN;
                  LOG_WARN("function does not exist", K(node->children_[0]->str_value_));
                  LOG_USER_ERROR(
                      OB_ERR_FUNCTION_UNKNOWN, (int32_t)node->children_[0]->str_len_, node->children_[0]->str_value_);
                }
              }
            }
          }
          break;
        }

else if (OB_ISNULL(expr)) {
                  ret = OB_ERR_FUNCTION_UNKNOWN;
                  LOG_WARN("function does not exist", K(node->children_[0]->str_value_));
                  LOG_USER_ERROR(
                      OB_ERR_FUNCTION_UNKNOWN, (int32_t)node->children_[0]->str_len_, node->children_[0]->str_value_);
                }
```

```
  REG_OP(ObExprDayOfWeek);

class ObExprDayOfWeek : public ObExprTimeBase {
public:
  ObExprDayOfWeek();
  explicit ObExprDayOfWeek(common::ObIAllocator& alloc);
  virtual ~ObExprDayOfWeek();
  static int calc_dayofweek(const ObExpr& expr, ObEvalCtx& ctx, ObDatum& expr_datum);

private:
  DISALLOW_COPY_AND_ASSIGN(ObExprDayOfWeek);
};

int ObExprDayOfWeek::calc_dayofweek(const ObExpr& expr, ObEvalCtx& ctx, ObDatum& expr_datum)
{
  int ret = OB_SUCCESS;
  if (ObExprTimeBase::calc(expr, ctx, expr_datum, DT_WDAY, true)) {
    LOG_WARN("calc day of week failed", K(ret));
  } else if (!expr_datum.is_null()) {
    expr_datum.set_int32(expr_datum.get_int32() % 7 + 1);
  }
  return ret;
}
```

src\sql\engine\expr\ob_expr_time.cpp

```
/root/src/oceanbase/deps/oblib/src/lib/timezone

deps\oblib\src\lib\timezone\ob_time_convert.h

int ObExprTimeBase::cg_expr

 case DT_MON_NAME:
        rt_expr.eval_func_ = ObExprMonthName::calc_month_name;
        break;
/root/src/oceanbase/src/sql/engine/expr

in ObExprTimeBase::calc_result1(ObObj& result, const ObObj& obj, ObExprCtx
& expr_ctx) const


    static ObExpr::EvalFunc g_expr_eval_functions[]
```

src\sql\engine\expr\ob_expr_time.cpp C 复制代码

```
int ObExprTimeBase::calc_result1(ObObj& result, const ObObj& obj, ObExprCtx& expr_ctx) const
{
  int ret = OB_SUCCESS;
  if (OB_UNLIKELY(obj.is_null())) {
    result.set_null();
  } else {
    ObTime ot;
    bool with_date = false;
    switch (dt_type_) {
      case DT_HOUR:
      case DT_MIN:
      case DT_SEC:
      case DT_USEC:
        with_date = false;
        break;
      case DT_YDAY:
      case DT_WDAY:
      case DT_MDAY:
      case DT_YEAR:
      case DT_MON:
        with_date = true;
        break;
      case DT_MON_NAME:
      case DT_DAY_NAME:
        with_date = true;
        break;

      default:
        LOG_WARN("ObExprTimeBase calc result1 switch default", K(dt_type_));
    }
    bool ignoreZero = (DT_MDAY == dt_type_ || DT_MON == dt_type_ || DT_MON_NAME == dt_type_);
    if (OB_FAIL(ob_expr_convert_to_dt_or_time(obj, expr_ctx, ot, with_date, ignoreZero))) {
      LOG_WARN("cast to ob time failed", K(ret), K(obj), K(expr_ctx.cast_mode_));
      if (CM_IS_WARN_ON_FAIL(expr_ctx.cast_mode_)) {
        LOG_WARN("cast to ob time failed", K(ret), K(expr_ctx.cast_mode_));
        LOG_USER_WARN(OB_ERR_CAST_VARCHAR_TO_TIME);
        ret = OB_SUCCESS;
        result.set_null();
      }
```

```
    } else {
      if (with_date && DT_MDAY != dt_type_ && ot.parts_[DT_DATE] + DAYS_FROM_ZERO_TO_BASE < 0) {
        result.set_null();
      } else if (DT_WDAY == dt_type_) {
        int32_t res = ot.parts_[DT_WDAY];
        res = res % 7 + 1;
        result.set_int32(res);
      } else if (DT_MON_NAME == dt_type_) {
        int32_t mon = ot.parts_[DT_MON];
        if (mon < 1 || mon > 12) {
          LOG_WARN("invalid month value", K(ret), K(mon));
          result.set_null();
        } else {
          const char* month_name = ObExprMonthName::get_month_name(ot.parts_[DT_MON]);
          result.set_string(common::ObVarcharType, month_name, strlen(month_name));
          result.set_collation_type(result_type_.get_collation_type());
          result.set_collation_level(result_type_.get_collation_level());
        }
      } else {
        result.set_int32(ot.parts_[dt_type_]);
      }
    }
  }
  return ret;
}

ObExprMonthName::ObExprMonthName(ObIAllocator& alloc)
    : ObExprTimeBase(alloc, DT_MON_NAME, T_FUN_SYS_MONTH_NAME, N_MONTH_NAME){};
#define N_MONTH_NAME "monthname"

int ObExprDayOfWeek::calc_dayofweek(const ObExpr& expr, ObEvalCtx& ctx, ObDatum& expr_datum)
{
  int ret = OB_SUCCESS;
  if (ObExprTimeBase::calc(expr, ctx, expr_datum, DT_WDAY, true)) {
    LOG_WARN("calc day of week failed", K(ret));
  } else if (!expr_datum.is_null()) {
    expr_datum.set_int32(expr_datum.get_int32() % 7 + 1);
  }
  return ret;
}
```

src\sql\parser\ob_item_type.h

```
// system function for mysql only
T_FUN_SYS_MONTH_NAME = 708, //sql 命令 怎么知道那个函数执行
#define N_MONTH_NAME "monthname"


ObExprMonthName::ObExprMonthName(ObIAllocator& alloc)
    : ObExprTimeBase(alloc, DT_MON_NAME, T_FUN_SYS_MONTH_NAME, N_MONTH_NAME){};

ObExprIsIpv4::ObExprIsIpv4(ObIAllocator& alloc)
    : ObFuncExprOperator(alloc, T_FUN_SYS_IS_IPV4, N_IS_IPV4, 1, NOT_ROW_DIMENSION)
{}
```

2.3 测试过程

- 根据日志看函数调用关系。

```
 stmt_query (ob_sql.cpp:171) [112642][466][YB427F000001-0005E8344D6D28DD] [lt=3] [dc=0] fail to handle text query(stmt=SELECT DAYNAME("2017-06-15"), ret=-5055)


else if (OB_FAIL(handle_text_query(stmt, context, result))) {
    if (OB_EAGAIN != ret && OB_ERR_PROXY_REROUTE != ret) {
      LOG_WARN("fail to handle text query", K(stmt), K(ret));
    }
  }

int ObSql::handle_text_query(

    [root@h12-storage03 log]# tail -f observer.log |grep "not exist"
[2022-09-09 11:17:54.141943] INFO  [SHARE.SCHEMA] ob_schema_getter_guard.cpp:6335 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=9] [dc=0] udf not exist(tenant_id=1, name=dayname)
[2022-09-09 11:17:54.141953] WARN  [SQL.RESV] do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:798) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=7] [dc=0] function does not exist(node->children_[0]->str_value_="DAYNAME")
[2022-09-09 11:17:54.141966] WARN  do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:800) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=11] [dc=0] FUNCTION DAYNAME does not exist
[2022-09-09 11:17:54.142091] INFO  [SERVER] obmp_base.cpp:1237 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] send error package.(user_error_code=1305, err=-5055, sql_state="42000", message=FUNCTION DAYNAME does not exist
)


[5]+  Stopped                 tail -f observer.log | grep --color=auto "not exist"
[root@h12-storage03 log]# grep "YB427F000001-0005E8344D6D28DF" observer.log
[2022-09-09 11:17:54.141719] INFO  [SQL] ob_sql.cpp:163 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=8] [dc=0] Begin to handle text statement(trunc_stmt=SELECT DAYNAME("2017-06-15"), sess_id=3221487631, execution_id=194262)
[2022-09-09 11:17:54.141943] INFO  [SHARE.SCHEMA] ob_schema_getter_guard.cpp:6335 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=9] [dc=0] udf not exist(tenant_id=1, name=dayname)
[2022-09-09 11:17:54.141953] WARN  [SQL.RESV] do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:798) [112642][466][YB427F000001-0005E8344D6D28DF]
```

```
[lt=7] [dc=0] function does not exist(node->children_[0]->str_value_="DAYNAME")
[2022-09-09 11:17:54.141966] WARN  do_recursive_resolve (:800) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=11] [dc=0] FUNCTION DAYNAME does not exist
[2022-09-09 11:17:54.141975] WARN  [SQL.RESV] resolve_sql_expr (ob_dml_resolver.cpp:208) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] fail to exec expr_resolver.resolve( &node, expr, *output_columns, sys_vars, sub_query_info, aggr_exprs, win_exprs, op_exprs, user_var_exprs)(ret=-5055, &node=0x7f07f2e6aa18, expr=NULL, *output_columns=[], sys_vars=[], sub_query_info=[], aggr_exprs=[], win_exprs=[], op_exprs=[], user_var_exprs=[])
[2022-09-09 11:17:54.141985] WARN  [SQL.RESV] resolve_field_list :1399) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=5] [dc=0] resolve sql expr failed(ret=-5055)
[2022-09-09 11:17:54.141989] WARN  [SQL.RESV] resolve_normal_query (ob_select_resolver.cpp:801) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] fail to exec resolve_field_list(*(parse_tree.children_[PARSE_SELECT_SELECT]))(ret=-5055)
[2022-09-09 11:17:54.141992] WARN  [SQL.RESV] resolve (ob_select_resolver.cpp:934) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] resolve normal query failed(ret=-5055)
[2022-09-09 11:17:54.141996] WARN  [SQL.RESV] select_stmt_resolver_func (ob_resolver.cpp:142) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] execute stmt_resolver failed(ret=-5055, parse_tree.type_=3035)
[2022-09-09 11:17:54.142005] WARN  [SQL] generate_stmt (ob_sql.cpp:1443) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] failed to resolve(ret=-5055)
[2022-09-09 11:17:54.142010] WARN  [SQL] generate_physical_plan (ob_sql.cpp:1531) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=3] [dc=0] Failed to generate stmt(ret=-5055, result.get_exec_context().need_disconnect()=false)
[2022-09-09 11:17:54.142014] WARN  [SQL] handle_physical_plan (ob_sql.cpp:3231) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] Failed to generate plan(ret=-5055, result.get_exec_context().need_disconnect()=false)
[2022-09-09 11:17:54.142018] WARN  [SQL] handle_text_query (ob_sql.cpp:1212) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] fail to handle physical plan(ret=-5055)
[2022-09-09 11:17:54.142023] WARN  [SQL] stmt_query (ob_sql.cpp:171) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=2] [dc=0] fail to handle text query(stmt=SELECT DAYNAME("2017-06-15"), ret=-5055)
[2022-09-09 11:17:54.142035] WARN  [SERVER] test_and_save_retry_state (ob_query_retry_ctrl.cpp:446) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=3] [dc=0] do not need retry(client_ret=-5055, err=-5055, expected_stmt=true, THIS_WORKER.get_timeout_ts()=1662693484141630, retry_type_=0, result.get_stmt_type()=1, result.get_exec_context().need_change_timeout_ret()=true, session->get_retry_info().get_last_query_retry_err()=0)
```

```
[2022-09-09 11:17:54.142042] INFO  [SERVER] ob_query_retry_ctrl.cpp:460 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] check if need retry(client_ret=-5055, err=-5055, retry_type_=0, retry_times=1, multi_stmt_item={is_part_of_multi_stmt:true, seq_num:0, sql:"SELECT DAYNAME("2017-06-15")"})
[2022-09-09 11:17:54.142049] WARN  [SERVER] do_process (obmp_query.cpp:638) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] run stmt_query failed, check if need retry(ret=-5055, cli_ret=-5055, retry_ctrl_.need_retry()=0, sql=SELECT DAYNAME("2017-06-15"))
[2022-09-09 11:17:54.142063] WARN  [SERVER] do_process (obmp_query.cpp:745) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] query failed(ret=-5055, retry_ctrl_.need_retry()=0)
[2022-09-09 11:17:54.142083] INFO  [SERVER] obmp_base.cpp:1163 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] sending error packet(err=-5055, bt="0xb553645 0xa899d40 0xa92d3cd 0xa8c70ce 0xa8c151b 0xb6d881d 0xa740c40 0xa71d3f9 0xa73e9d6 0xa71b2ff 0xa71b7cc 0x1e7613c 0x1e75fcd 0x1e7a00e 0xb4fbd75 0xb4fa515 0xb20b1cf", extra_err_info=0x7f07f2e578c8)
[2022-09-09 11:17:54.142091] INFO  [SERVER] obmp_base.cpp:1237 [112642][466][YB427F000001-0005E8344D6D28DF] [lt=4] [dc=0] send error package.(user_error_code=1305, err=-5055, sql_state="42000", message=FUNCTION DAYNAME does not exist)
[2022-09-09 11:17:54.142161] WARN  [SERVER] process (obmp_query.cpp:291) [112642][466][YB427F000001-0005E8344D6D28DF] [lt=3] [dc=0] fail execute sql(sql_id="", sql=SELECT DAYNAME("2017-06-15"), sessid=3221487631, ret=-5055, ret="OB_ERR_FUNCTION_UNKNOWN", need_disconnect=false)
[root@h12-storage03 log]#


SELECT DAYNAME("2022-09-20");
SELECT DAYOFWEEK('2022-09-20');
SELECT MONTHNAME('2022-09-20');



cp /root/src/oceanbase/build_debug/src/observer/observer /root/src/observer/bin/observer
obd cluster stop test
```

第二次测试

```
[root@h12-storage03 log]# tail -f observer.log |grep DAYNAME
[2022-09-20 21:04:42.773166] INFO  [SQL] ob_sql.cpp:163 [437909][466][YB427F000001-0005E91B467985A3] [lt=9] [dc=0] Begin to handle text statement(trunc_stmt=SELECT DAYNAME("2022-09-20"), sess_id=3221487618, execution_id=62341)
[2022-09-20 21:04:42.773450] WARN  [SQL.RESV] do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:798) [437909][466][YB427F000001-0005E91B467985A3] [lt=7] [dc=0] function does not exist(node->children_[0]->str_value_="DAYNAME")
[2022-09-20 21:04:42.773456] WARN  do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:800) [437909][466][YB427F000001-0005E91B467985A3] [lt=4] [dc=0] FUNCTION DAYNAME does not exist
[2022-09-20 21:04:42.773517] WARN  [SQL] stmt_query (ob_sql.cpp:171) [437909][466][YB427F000001-0005E91B467985A3] [lt=2] [dc=0] fail to handle text query(stmt=SELECT DAYNAME("2022-09-20"), ret=-5055)
[2022-09-20 21:04:42.773537] INFO  [SERVER] ob_query_retry_ctrl.cpp:460 [437909][466][YB427F000001-0005E91B467985A3] [lt=5] [dc=0] check if need retry(client_ret=-5055, err=-5055, retry_type_=0, retry_times=1, multi_stmt_item={is_part_of_multi_stmt:true, seq_num:0, sql:"SELECT DAYNAME("2022-09-20")"})
[2022-09-20 21:04:42.773542] WARN  [SERVER] do_process (obmp_query.cpp:638) [437909][466][YB427F000001-0005E91B467985A3] [lt=4] [dc=0] run stmt_query failed, check if need retry(ret=-5055, cli_ret=-5055, retry_ctrl_.need_retry()=0, sql=SELECT DAYNAME("2022-09-20"))
[2022-09-20 21:04:42.773995] INFO  [SERVER] obmp_base.cpp:1237 [437909][466][YB427F000001-0005E91B467985A3] [lt=4] [dc=0] send error package.(user_error_code=1305, err=-5055, sql_state="42000", message=FUNCTION DAYNAME does not exist)
[2022-09-20 21:04:42.774084] WARN  [SERVER] process (obmp_query.cpp:291) [437909][466][YB427F000001-0005E91B467985A3] [lt=7] [dc=0] fail execute sql(sql_id="", sql=SELECT DAYNAME("2022-09-20"), sessid=3221487618, ret=-5055, ret="OB_ERR_FUNCTION_UNKNOWN", need_disconnect=false)
[2022-09-20 21:04:45.975760] INFO  [SQL] ob_sql.cpp:163 [437909][466][YB427F000001-0005E91B467985A4] [lt=8] [dc=0] Begin to handle text statement(trunc_stmt=SELECT DAYNAME("2022-09-20"), sess_id=3221487618, execution_id=62397)
[2022-09-20 21:04:45.976064] WARN  [SQL.RESV] do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:798) [437909][466][YB427F000001-0005E91B467985A4] [lt=8] [dc=0] function does not exist(node->children_[0]->str_value_="DAYNAME")
[2022-09-20 21:04:45.976072] WARN  do_recursive_resolve (ob_raw_expr_resolver_impl.cpp:800) [437909][466][YB427F000001-0005E91B467985A4] [lt=5] [dc=0] FUNCTION DAYNAME does not exist
[2022-09-20 21:04:45.976166] WARN  [SQL] stmt_query (ob_sql.cpp:171) [437909][466][YB427F000001-0005E91B467985A4] [lt=4] [dc=0] fail to handle text
```

```
query(stmt=SELECT DAYNAME("2022-09-20"), ret=-5055)
[2022-09-20 21:04:45.976194] INFO  [SERVER] ob_query_retry_ctrl.cpp:460 [437909][466][YB427F000001-0005E91B467985A4] [lt=6] [dc=0] check if need retry(client_ret=-5055, err=-5055, retry_type_=0, retry_times=1, multi_stmt_item={is_part_of_multi_stmt:true, seq_num:0, sql:"SELECT DAYNAME("2022-09-20")"})
[2022-09-20 21:04:45.976203] WARN  [SERVER] do_process (obmp_query.cpp:638) [437909][466][YB427F000001-0005E91B467985A4] [lt=6] [dc=0] run stmt_query failed, check if need retry(ret=-5055, cli_ret=-5055, retry_ctrl_.need_retry()=0, sql=SELECT DAYNAME("2022-09-20"))
[2022-09-20 21:04:45.976258] INFO  [SERVER] obmp_base.cpp:1237 [437909][466][YB427F000001-0005E91B467985A4] [lt=4] [dc=0] send error package.(user_error_code=1305, err=-5055, sql_state="42000", message=FUNCTION DAYNAME does not exist)
[2022-09-20 21:04:45.976355] WARN  [SERVER] process (obmp_query.cpp:291) [4

cp /root/src/oceanbase/build_debug/src/observer/observer /root/src/observer/bin/observer


obclient [oceanbase]> SELECT DAYOFWEEK('2022-09-20');
+--------------------------+
| DAYOFWEEK('2022-09-20') |
+--------------------------+
|                        3 |
+--------------------------+
Returns the weekday index for date (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index values correspond to the ODBC standard. Returns NULL if date is NUL
SELECT MONTHNAME('2022-09-20');
SELECT DAYNAME("2022-09-20");
select dayname("1962-03-01");

cp /root/oceanbase/build_debug/src/observer/observer /root/observer/bin/observer

```

三、 Conclusion

- https://github.com/pingcap/tidb/pull/10732

- https://open.oceanbase.com/blog/10900229?currentPage=1
- https://open.oceanbase.com/blog/8600156?currentPage=undefined
- https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_dayname
- https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html