openGauss SQL兼容性需求check-in评审报告

评审纪要

时间：
特性名称：openGauss兼容mysql时间函数（第二阶段）
特性责任人：黄振业
评审人：

设计checklist

模块	类别	检查项	检查项详细说明	是否涉及	是否完成检查
GUC参数	功能	新增GUC参数适配项	①guc.cpp里需要新增guc参数的相应代码，包括参数的校验/赋值等功能函数； ②如果要加入postgres.conf，需要修改postgresql_single.conf.sample 路径：src/common/backend/utils/misc/ 一般以注释方式加入，如果非常确认使用某个默认值，也可以以非注释方式加入 ③如果要允许gs_guc工具设置，请修改cluster_guc.conf 路径：src/bin/gs_guc/cluster_guc.conf	否	是
升级	兼容性	版本号变更	src/common/backend/utils/init/globals.cpp：GRAND_VERSION_NUM	否	是
升级	兼容性	前向兼容性	1）涉及系统表修改的特性，必须提供系统表升级脚本和回滚脚本，修改版本号文件 2）对于涉及修改持久化数据（如日志）的特性，必须考虑新老版本共存时的兼容性场景，必要情况下，需要增加版本号以进行识别 3）对于涉及修改执行态数据格式（如syscache结构）的特性，必须考虑新老版本共存时的兼容性场景，必要情况下，需要增加版本号以进行识别	否	是
升级	兼容性	验证升级正常	涉及到升级时需要验证如下场景： 1. 集群环境安装：该步骤会验证集群环境安装是否正常 2. 集群环境升级：从老的版本（1.1.0版本）升级到最新版本是否正常	否	是

模块	类别	检查项	检查项详细说明	是否涉及	是否完成检查
公共数据结构	功能	hashtable	hashtable使用注意事项 ①在使用hash表时，entry必须封装成一个结构体，第一个成员必须是hash key 否则会造成查找失败！！ ②封装的结构体必须只有两个成员：key+value，即value需要封装起来 否则hash表在search的时候会core！！ ③hash表在创建的时候，entry 的size一定要正确 否则在enter的时候会造成写越界，不会报错但是其它变量内存会被踩，不易发现！！	否	是
公共数据结构	功能	禁止使用commandTag判断Query的类型	需要对SELECT\INSERT\UPDATE\DELETE\MERGE INTO等query类型进行识别判断时，应该使用CmdType进行判断或新增标识。 禁止使用commandTag作为判断标识，因为commandTag在带returning*语句执行中变化。	否	是
系统函数	兼容性	系统表升级是否正确	通过对比initdb和升级后的db中pg_proc每一列的值，避免出现升级和非升级不一致的情况	否	是
系统函数	兼容性	新增数据类型适配FENCED函数	新增数据类型如果是FENCED函数使用，需要修改UDFArgumentHandler函数，把数据类型添加到对应收发分支。	否	是
系统函数	功能	NULL值处理	对于非strict函数允许接收NULL入参，在函数逻辑中应对NULL值进行细致的处理	否	是
前向兼容性	兼容性	Node相关数据结构删减	禁止对Node相关数据结构（如Query、RangeTblEntry）数据成员删减	否	是
兼容性	兼容性	兼容性设计原则	目前支持兼容性PG,A,B,C;对于每一个兼容点要求：1、如果该兼容点为非冲突兼容点，应该是各个兼容模式下均支持，不需要开关控制 2、如果该兼容性点为冲突兼容点，应该由兼容开关控制。	否	是
查询解析	兼容性	添加关键字场景	为了尽可能的防止移进规约冲突，最大化的用关键字作为对象名，关键字分有4类：非保留关键字、列名关键字、类型函数关键字、保留关键字。他们的约束逐渐加强，比如保留关键字不可以用做表名。添加关键字需要考虑如下： 1. 添加新的关键字最好先作为非保留关键字添加 2. 如果要加其他关键字类型那么需要考虑是否有前向兼容问题。比如排查客户场景是否有用该关键字作为对象名。 3. 关键字还需要加入kwlist.h文件中。	是	是

模块	类别	检查项	检查项详细说明	是否涉及	是否完成检查
查询解析	兼容性	Gram.y中处理语法冲突	1、合理使用a_expr/b_expr/c_expr使用场景 为了尽可能防止冲突，划分了a_expr/b_expr/c_expr，比如BEWEEN a_expr AND b_expr中 AND属于a_expr范畴，b_expr中不含AND。否则就无法区分BETWEEN之后是Bool判断还是范围。c_expr是a_expr和b_expr的交集。所以 尽量用a_expr这样支持的范围最广。如果用b_expr/c_expr 那需要考虑潜在的用户问题场景（参考BETWEEN...AND...例子）并在手册中写清楚约束。	是	是
其他	功能	新增已有的数据结构成员适配序列化与反序列化	注意适配： src/common/backend/nodes/copyfuncs.cpp src/common/backend/nodes/equalfuncs.cpp src/common/backend/nodes/outfuncs.cpp src/common/backend/nodes/readfuncs.cpp	否	是
公共子系统	内存定位定界	局部变量内存释放，非程序结尾位置，释放进行置空；全局内存变量释放，无论什么位置均需要置空	防止内存使用出现use-after-free模式，内存越界	是	是
公共子系统	内存定位定界	对于使用数组、下标访问内存操作场景，是否提前判断下标符合内存空间要求	防止出现heap overflow场景	是	是
公共子系统	内存定位定界	内存分配报错日志中，是否需要新增信息，便于分析报错原因	内存报错场景下，快速定位内存报错原因	否	是
资源负载管理	兼容性	只能新增系统表参数，不可以删除已有的参数	前向兼容	否	是

模块	类别	检查项	检查项详细说明	是否涉及	是否完成检查
资源负载管理	功能	新增系统表参数的默认初始化数值要合理，不影响已有的功能	前向兼容	否	是
HA用例	功能	ha check	1.HA相关设计实现需要严格系统的测试。 2.需要保证最简单的功能验证，包括搭建主备环境，确认主备可以正常同步，以及跑ha check用例，确认所有用例可以跑过。 3.需要完成各种故障场景的验证，设计详细的故障库，确保覆盖各个场景。	否	是
安全	未公开接口	是否新增或修改函数、视图、系统表	新增或修改函数、视图、系统表需刷新产品文档，考虑权限控制	是	是
安全	未公开接口	是否新增SQL语法	新增SQL语法需刷新产品文档，支持记录审计日志	否	是

特性LLT测试方案及验证结论

基本测试

TO_DAYS()测试

```
-- 正常测试测试
insert into func_test3(functionName,result) values('TO_DAYS(''2022-1-
1'')',TO_DAYS('2022-1-1'));
insert into func_test3(functionName,result)
values('TO_DAYS(''44440101'')',TO_DAYS('44440101'));
insert into func_test3(functionName,result)
values('TO_DAYS(20000229)',TO_DAYS(20000229));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-1-1
1:1:1'')',TO_DAYS('2022-1-1 1:1:1'));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-2-2
2:2:2.0000015'')',TO_DAYS('2022-2-2 2:2:2.0000015'));
insert into func_test3(functionName,result)
values('TO_DAYS(''20220101010101'')',TO_DAYS('20220101010101'));
insert into func_test3(functionName,result)
values('TO_DAYS(20220101010101)',TO_DAYS(20220101010101));
```

```
insert into func_test3(functionName,result)
values('TO_DAYS(''20220101010101.000001'')',TO_DAYS('20220101010101.000001'));
insert into func_test3(functionName,result)
values('TO_DAYS(20220101010101.000002)',TO_DAYS(20220101010101.000002));
-- 非法的date/datetime格式
insert into func_test3(functionName,result) values('TO_DAYS(''2022-1-
32'')',TO_DAYS('2022-1-32'));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-13-
1'')',TO_DAYS('2022-13-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-2-2
2:2:60'')',TO_DAYS('2022-2-2 2:2:60'));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-2-2
2:60:2'')',TO_DAYS('2022-2-2 2:60:2'));
insert into func_test3(functionName,result) values('TO_DAYS(''2022-2-2
24:2:2'')',TO_DAYS('2022-2-2 24:2:2'));
-- 超长参数
insert into func_test3(functionName,result) values('TO_DAYS(''99999999999-1-
1'')',TO_DAYS('99999999999-1-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''00000000000-1-
1'')',TO_DAYS('00000000000-1-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''00000000000-
00000000001-1'')',TO_DAYS('00000000000-00000000001-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''00000000000-
00000000001-0000000001'')',TO_DAYS('00000000000-00000000001-0000000001'));
-- 特殊类型参数
insert into func_test3(functionName,result)
values('TO_DAYS(true)',TO_DAYS(true));
insert into func_test3(functionName,result)
values('TO_DAYS(false)',TO_DAYS(false));
insert into func_test3(functionName,result)
values('TO_DAYS(null)',TO_DAYS(null));
insert into func_test3(functionName,result) values('TO_DAYS(date''2000-1-
1'')',TO_DAYS(date'2000-1-1'));
insert into func_test3(functionName,result) values('TO_DAYS(cast(''2022-2-2
2:2:2'' as datetime))',TO_DAYS(cast('2022-2-2 2:2:2' as datetime)));
insert into func_test3(functionName,result)
values('TO_DAYS(time''1:1:1'')',TO_DAYS(time'1:1:1'));
insert into func_test3(functionName,result)
values('TO_DAYS(time''25:0:0'')',TO_DAYS(time'25:0:0'));
-- 数值
insert into func_test3(functionName,result) values('TO_DAYS(1)',TO_DAYS(1));
insert into func_test3(functionName,result) values('TO_DAYS(1)',TO_DAYS(1));
insert into func_test3(functionName,result) values('TO_DAYS(001)',TO_DAYS(001));
                          -- 单位数年月日
insert into func_test3(functionName,result) values('TO_DAYS(101)',TO_DAYS(101));
                          -- 双位日+单位月
insert into func_test3(functionName,result)
values('TO_DAYS(0101)',TO_DAYS(0101));                          -- 双位日月
insert into func_test3(functionName,result)
values('TO_DAYS(00101)',TO_DAYS(00101));                        -- 双位日月+单位年
insert into func_test3(functionName,result)
values('TO_DAYS(000101)',TO_DAYS(000101));                      -- 双位年月日
insert into func_test3(functionName,result)
values('TO_DAYS(00000101)',TO_DAYS(00000101));                  -- -四位年+双位月日
```

```
insert into func_test3(functionName,result)
values('TO_DAYS(00000101001)',TO_DAYS(00000101001));            -- 四位年+双位月日+单位时分秒
insert into func_test3(functionName,result)
values('TO_DAYS(00000101000001)',TO_DAYS(00000101000001));      -- 四位年+双位月日时分秒
insert into func_test3(functionName,result)
values('TO_DAYS(01000001)',TO_DAYS(01000001));                  -- 双位日+双位时分秒
insert into func_test3(functionName,result)
values('TO_DAYS(0101000001)',TO_DAYS(0101000001));              -- 双位日月+双位时分秒
insert into func_test3(functionName,result)
values('TO_DAYS(00101000001)',TO_DAYS(00101000001));            -- 单位年+双位日月+双位时分秒
insert into func_test3(functionName,result)
values('TO_DAYS(0000101000001)',TO_DAYS(0000101000001));        -- 三位年+双位日月+双位时分秒
-- 边界测试
-- 最小值
insert into func_test3(functionName,result) values('TO_DAYS(''0000-1-1'')',TO_DAYS('0000-1-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''0000-1-1 00:00:00'')',TO_DAYS('0000-1-1 00:00:00'));
insert into func_test3(functionName,result) values('TO_DAYS(''0000-0-0'')',TO_DAYS('0000-0-0'));
-- 最大值
insert into func_test3(functionName,result) values('TO_DAYS(''9999-12-31'')',TO_DAYS('9999-12-31'));
insert into func_test3(functionName,result) values('TO_DAYS(''9999-12-31 23:59:59.999999'')',TO_DAYS('9999-12-31 23:59:59.999999'));
insert into func_test3(functionName,result) values('TO_DAYS(''10000-1-1'')',TO_DAYS('10000-1-1'));
insert into func_test3(functionName,result) values('TO_DAYS(''10000-1-1 00:00:00'')',TO_DAYS('10000-1-1 00:00:00'));
```

TO_SECONDS()测试

```
-- 正常测试
-- date格式
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27'')', TO_SECONDS('2022-07-27'));
insert into func_test3(functionName, result) values('TO_SECONDS(''20220727'')', TO_SECONDS('20220727'));
-- datetime格式
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 15:25:30'')', TO_SECONDS('2022-07-27 15:25:30'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 15:25:30.8888855'')', TO_SECONDS('2022-07-27 15:25:30.8888855'));
insert into func_test3(functionName, result)
values('TO_SECONDS(''20220727152530'')', TO_SECONDS('20220727152530'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 15:25:30.8888854'')', TO_SECONDS('2022-07-27 15:25:30.8888854'));
-- 特异参数测试
-- 不存在的日期或时间
```

```
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-32'')', TO_SECONDS('2022-07-32'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-13-27'')', TO_SECONDS('2022-13-27'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 12:00:61'')', TO_SECONDS('2022-07-27 12:00:61'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 12:61:00'')', TO_SECONDS('2022-07-27 12:61:00'));
insert into func_test3(functionName, result) values('TO_SECONDS(''2022-07-27 25:00:00'')', TO_SECONDS('2022-07-27 25:00:00'));
-- 超大参数
insert into func_test3(functionName, result)
values('TO_SECONDS(''99999999999999999999-07-27'')',
TO_SECONDS('99999999999999999999-07-27'));
-- 特殊参数类型
insert into func_test3(functionName, result) values('TO_SECONDS(null)',
TO_SECONDS(null));
insert into func_test3(functionName, result) values('TO_SECONDS(true)',
TO_SECONDS(true));
insert into func_test3(functionName, result) values('TO_SECONDS(date''2022-04-05'')', TO_SECONDS(date'2022-04-05'));
insert into func_test3(functionName, result) values('TO_SECONDS(cast(''2022-04-05 14:35:00'' as datetime))', TO_SECONDS(cast('2022-04-05 14:35:00' as
datetime)));
insert into func_test3(functionName, result) values('TO_SECONDS(cast(''2022-04-05 14:35:00.888'' as datetime))', TO_SECONDS(cast('2022-04-05 14:35:00.888' as
datetime)));
insert into func_test3(functionName, result) values('TO_SECONDS(time''1:1:1'')',
TO_SECONDS(time'1:1:1'));
insert into func_test3(functionName, result)
values('TO_SECONDS(time''25:00:00'')', TO_SECONDS(time'25:00:00'));
-- 数值类型参数
insert into func_test3(functionName, result) values('TO_SECONDS(050505)',
TO_SECONDS(050505));
insert into func_test3(functionName, result) values('TO_SECONDS(20220801)',
TO_SECONDS(20220801));
insert into func_test3(functionName, result)
values('TO_SECONDS(20220801182030)', TO_SECONDS(20220801182030));
insert into func_test3(functionName, result)
values('TO_SECONDS(20220801182030.8888855)',
TO_SECONDS(20220801182030.8888855));
-- 任意分隔符参数
insert into func_test3(functionName, result)
values('TO_SECONDS(''0,1,1,0,0,0'')', TO_SECONDS('0,1,1,0,0,0'));
-- 边界测试
-- 最大值
insert into func_test3(functionName, result) values('TO_SECONDS(''9999-12-31'')', TO_SECONDS('9999-12-31'));
insert into func_test3(functionName, result) values('TO_SECONDS(''9999-12-31 23:59:59'')', TO_SECONDS('9999-12-31 23:59:59'));
insert into func_test3(functionName, result) values('TO_SECONDS(''9999-12-31 23:59:59.999999'')', TO_SECONDS('9999-12-31 23:59:59.999999'));
insert into func_test3(functionName, result) values('TO_SECONDS(''10000-01-01'')', TO_SECONDS('10000-01-01'));
-- 最小值
```

```
insert into func_test3(functionName, result) values('TO_SECONDS(''0000-01-
01'')', TO_SECONDS('0000-01-01'));
insert into func_test3(functionName, result) values('TO_SECONDS(''0000-01-01
00:00:00'')', TO_SECONDS('0000-01-01 00:00:00'));
insert into func_test3(functionName, result) values('TO_SECONDS(''0000-00-
00'')', TO_SECONDS('0000-00-00'));
insert into func_test3(functionName, result) values('TO_SECONDS(''0000-00-00
00:00:00'')', TO_SECONDS('0000-00-00 00:00:00'));
```

UNIX_TIMESTAMP()测试

```
-- UNIX_TIMESTAMP()
-- 正常测试
-- date格式
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-
27'')', UNIX_TIMESTAMP('2022-07-27'));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(''20220727'')', UNIX_TIMESTAMP('20220727'));
-- datetime格式
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-27
15:25:30'')', UNIX_TIMESTAMP('2022-07-27 15:25:30'));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(''20220727152530'')', UNIX_TIMESTAMP('20220727152530'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-27
15:25:30.8888855'')', UNIX_TIMESTAMP('2022-07-27 15:25:30.8888855'));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(''20220727152530.8888854'')',
UNIX_TIMESTAMP('20220727152530.8888854'));
-- 特异参数测试
-- 不存在的日期或时间
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-
32'')', UNIX_TIMESTAMP('2022-07-32'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-13-
27'')', UNIX_TIMESTAMP('2022-13-27'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-27
12:00:61'')', UNIX_TIMESTAMP('2022-07-27 12:00:61'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-27
12:61:00'')', UNIX_TIMESTAMP('2022-07-27 12:61:00'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2022-07-27
25:00:00'')', UNIX_TIMESTAMP('2022-07-27 25:00:00'));
-- 超大参数
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(''99999999999999999999-07-27'')',
UNIX_TIMESTAMP('99999999999999999999-07-27'));
-- 特殊类型参数
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(null)',
UNIX_TIMESTAMP(null));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(true)',
UNIX_TIMESTAMP(true));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(date''2022-
04-05'')', UNIX_TIMESTAMP(date'2022-04-05'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(cast(''2022-
04-05 14:35:00'' as datetime))', UNIX_TIMESTAMP(cast('2022-04-05 14:35:00' as
datetime)));
```

```
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(cast(''2022-04-05 14:35:00.888'' as datetime))', UNIX_TIMESTAMP(cast('2022-04-05 14:35:00.888' as datetime)));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(time''1:1:1'')', UNIX_TIMESTAMP(time'1:1:1'));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(time''25:00:00'')', UNIX_TIMESTAMP(time'25:00:00'));
-- 数值类型参数
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(050505)', UNIX_TIMESTAMP(050505));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(20220801)', UNIX_TIMESTAMP(20220801));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(20220801182030)', UNIX_TIMESTAMP(20220801182030));
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(20220801182030.8888855)', UNIX_TIMESTAMP(20220801182030.8888855));
-- 任意分隔符参数
insert into func_test3(functionName, result)
values('UNIX_TIMESTAMP(''0,1,1,0,0,0'')', UNIX_TIMESTAMP('0,1,1,0,0,0'));
-- 边界测试
-- 最大值
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2038-01-19'')', UNIX_TIMESTAMP('2038-01-19'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2038-01-19 11:14:07'')', UNIX_TIMESTAMP('2038-01-19 11:14:07'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2038-01-19 11:14:07.9999'')', UNIX_TIMESTAMP('2038-01-19 11:14:07.9999'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2038-01-19 11:14:07.999999999'')', UNIX_TIMESTAMP('2038-01-19 11:14:07.999999999'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''2038-01-19 11:14:08'')', UNIX_TIMESTAMP('2038-01-19 11:14:08'));
-- 最小值
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1970-01-01'')', UNIX_TIMESTAMP('1970-01-01'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1970-01-01 08:00:00'')', UNIX_TIMESTAMP('1970-01-01 08:00:00'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1970-01-01 08:00:01'')', UNIX_TIMESTAMP('1970-01-01 08:00:01'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1970-01-01 08:00:00.999999'')', UNIX_TIMESTAMP('1970-01-01 08:00:00.999999'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1970-01-01 07:59:59'')', UNIX_TIMESTAMP('1970-01-01 07:59:59'));
insert into func_test3(functionName, result) values('UNIX_TIMESTAMP(''1969-12-31'')', UNIX_TIMESTAMP('1969-12-31'));
```

utc_date()测试

```
insert into func_test3(functionName, result) values('UTC_DATE', UTC_DATE);
insert into func_test3(functionName, result) values('UTC_DATE()', UTC_DATE());
```

utc_time()测试

```
insert into func_test3(functionName, result) values('UTC_TIME', UTC_TIME);
insert into func_test3(functionName, result) values('UTC_TIME()', UTC_TIME());
insert into func_test3(functionName, result) values('UTC_TIME(0)', UTC_TIME(0));
insert into func_test3(functionName, result) values('UTC_TIME(1)', UTC_TIME(1));
insert into func_test3(functionName, result) values('UTC_TIME(2)', UTC_TIME(2));
insert into func_test3(functionName, result) values('UTC_TIME(3)', UTC_TIME(3));
insert into func_test3(functionName, result) values('UTC_TIME(4)', UTC_TIME(4));
insert into func_test3(functionName, result) values('UTC_TIME(5)', UTC_TIME(5));
insert into func_test3(functionName, result) values('UTC_TIME(6)', UTC_TIME(6));
insert into func_test3(functionName, result) values('UTC_TIME(-1)',
UTC_TIME(-1));
```

utc_timestamp()测试

```
insert into func_test3(functionName, result) values('UTC_TIMESTAMP',
UTC_TIMESTAMP);
insert into func_test3(functionName, result) values('UTC_TIMESTAMP()',
UTC_TIMESTAMP());
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(0)',
UTC_TIMESTAMP(0));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(1)',
UTC_TIMESTAMP(1));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(2)',
UTC_TIMESTAMP(2));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(3)',
UTC_TIMESTAMP(3));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(4)',
UTC_TIMESTAMP(4));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(5)',
UTC_TIMESTAMP(5));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(6)',
UTC_TIMESTAMP(6));
insert into func_test3(functionName, result) values('UTC_TIMESTAMP(-1)',
UTC_TIMESTAMP(-1));
```

回归测试验证

测试集	结论
fastcheck	通过
memcheck	通过
hacheck	无

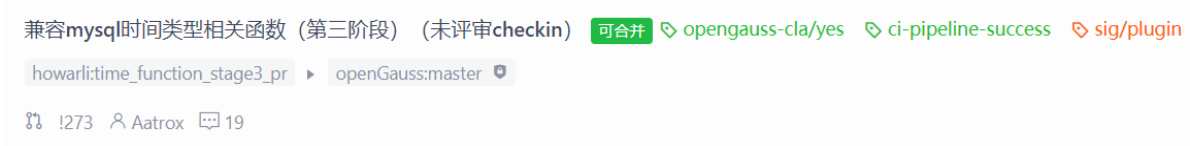
Fastcheck

所有用例的fastcheck通过:

```
============== shutting down postmaster              ==============
stop postmaster now!

========================
 All 155 tests passed.
 Total Time: 339.024329s
========================
```

线上门禁通过:



Memcheck

```
[b8402@dytang2 dolphin]$ cat ~/memchk/asan/runlog.34775
-----------------------------------------------------
Suppressions used:
  count      bytes template
    150       9636 regression_main
  34960    3496000 get_node_info_name
-----------------------------------------------------
```

hacheck

无

代码检视结论

编码规范检查

编程规范

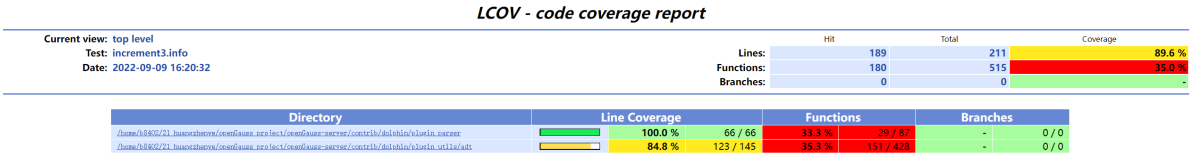
> 按照pr中的修改意见进行修改

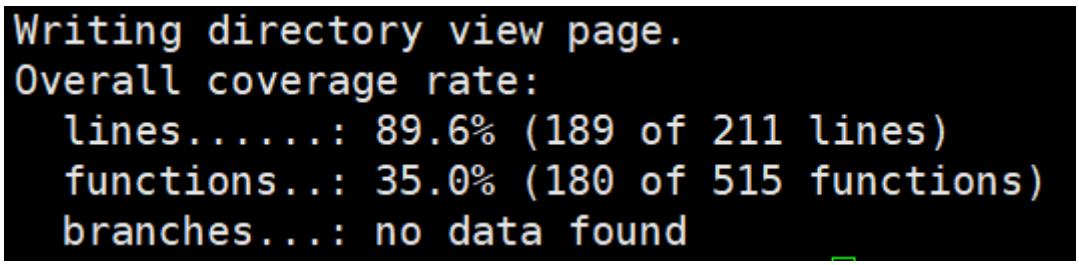
PclintPlus告警

内存使用排查

覆盖率

增量代码测试用例覆盖率80%+





鲲鹏平台乱序排查

编号	排查依据	排查结果
1	**是否为多线程并发场景?**	
2	**是否涉及线程间共享数据?**	
3	**是否未对共享数据加锁保护?**	
4	线程间共享数据是否存在无锁同步模式?	
5	识别是否为典型使用模式?	
6	是否**未** 在正确的位置插入合适的内存屏障	

代码检视意见

提出人	意见	答复详情
pengjiong	函数的第一个左大括号不换行	统一大括号的编码风格
chenxiaobin	一行代码的列数不超过120，请排查	已解决
仲夏十三	整改calc_daynr函数的魔鬼数字	该函数已删除

遗留问题

测试建议