目录

[一、摘要 2](#_Toc531100064)

[二、源码路径说明 2](#_Toc531100065)

[三、编译 2](#_Toc531100066)

[3.1、windows 2](#_Toc531100067)

[3.2、linux 2](#_Toc531100068)

[四、配置说明 3](#_Toc531100069)

[五、demo 7](#_Toc531100070)

[5.1、ddl文件测试 8](#_Toc531100071)

[5.2、SQLColumns的测试 8](#_Toc531100072)

# 一、摘要

本文档主要说明如何使用测试代码。本代码适用于linux和windows环境上编译、运行，在linux上通过执行make编译，在windows上需要安装Visual Studio，VS2013、VS2015或者其他的版本即可，新建项目然后添加源码文件到VS工程。

本测试程序主要通过配置文件设置测试选项，因此使用过程中请注意配置文件设置的参数值。

本测试代码的设计原理：

测试项以及测试使用的表、表定义等通过配置文件方式。测试的表可以的单表，也可以是多表，通过字段[table]、[table\_1]、[table\_2]。。。[table\_n]方式定义多表，每一个[table]字段定义一张表，测试时顺序使用表1到表N的定义测试。

所有的测试结果都保存到日志文件，测试完成后，测试结果分析请从日志文件读取。

本文档不能全面的覆盖到代码的实现，实现细节还需要参考代码的实现，测试文档会在适用中不断的完善。

# 二、源码路径说明

**bin:**主要存放bin文件和测试时使用的配置文件等，在linux环境下运行时，进入该目录，执行sh build.sh即可完成编译.编译可以在上一层目录执行make；在该目录下运行测试，主要是方便测试，管理测试日志等；

**case:**测试case的主要实现；

**config:**主要存放配置文件；自定义测试配置时，可参考该目录下的配置文件；

**doc:**主要存放相关文档，如《使用文档.docx》等；

**driver:**对ODBC API实现的第二次封装，方便测试case的设计和调用；

**main:**测试程序的入口实现；

**util:**公共接口的实现

**vs:**vs2013工程路径

# 三、编译

## 3.1、windows

打开VS，新建工程后，添加源码路径下的所有.cpp、.c和.h文件，点击编译即可。测试时请注意配置和可执行文件在同一路径下。

## 3.2、linux

Linux环境下，支持的驱动管理器包括unixODBC和trafodion driver manager，使用的驱动库有区别，因此编译时需要指定使用的驱动管理器；测试时请注意配置和可执行文件在同一路径下。

**使用unixODBC时:**

默认情况是使用unixODBC，vi打开makefile文件，查看**ODBC\_VERSION =UNIXODBC**是否打开，如果被注释掉，请去掉”#”。

**使用trafodion driver manger时：**

Vi打开makefile文件，注释掉**ODBC\_VERSION =UNIXODBC，在行首添加”#”即可，如：#ODBC\_VERSION =UNIXODBC**

修改驱动使用的动态库路径：

如默认安装路径是/usr/lib64，修改如下：

LIBPATH+=-L/usr/lib64

# 四、配置说明

配置文件config.ini主要包含两个字段：[config]字段主要是全局使用，包括配置需要测试的case、数据源信息等；[table]、[table\_1]、[table\_n]字段主要是测试使用表的配置信息。

[config]字段：

CATALOG：catalog名

DSN：测试连接的DSN名

User：连接数据库的用户名

Password:连接数据库使用的密码

Schema：测试使用的SCHEMA

isMultithreading：

TRUE：多线程并发执行测试case

FALSE：串行执行测试case

connect\_string：连接数据时如需要额外添加URL，在这个字段添加，如DataLang=4;TenantName=esyndb;不需要指定时，注释掉此字段或者“connect\_string=“等于空即可；

test\_case：配置本次需要执行的测试case，目前支持的测试case有：

SQLColumns：执行SQLColumns；

SQLTables：执行SQLTables API；

testLoadDirect：通过SQLPrepare 、 SQLBindParameter、 SQLExecute方式插数据测试；

testLoadAtExec：通过SQLPrepare 、 SQLBindParameter、 SQLExecute、SQLPutData方式插数据测试；SQLPutData有两种插入方式：SQL\_DATA\_AT\_EXEC和SQL\_LEN\_DATA\_AT\_EXEC，因此测试时注意配置load\_at\_exec\_type的设置；

testRowset：rowset测试；

testStmt：自定义的ddl测试，ddl语句定义在stmt=字段，该测试适用于列小数据的ddl测试使用，如果测试lob或者varchar很大时请使用testSelect方式测试；

testSqlFile：执行自定义的ddl文件；

testSelect：执行select测试，select语句定义在select\_stmt=字段，该测试可用于测试小量的数据，也可以使用在lob表的select测试；

cleanupTable：清除表的数据；

**备注：**

测试前都会做测试表的准备工作，表定义在xxx.sql文件中，然后测试case配置为“test\_case=testSqlFile”，“sql\_file=xxx.sql”，创建表好后再做其他的测试，否则表未创建好就并发做其他的测试，产生的错误信息多，测试意义不大。

[table]字段：

每一个[table]字段定义一个表的测试信息，如需要测试多表，请安[table\_1]、[table\_2]、[table\_3]。。。[table\_n]方式定义表。

**测试表基本信息配置：**

sql\_file：执行ddl文件名，ddl文件中可定义多张表的ddl，每个ddl直接用分号区分，注释的行用“#”开头，如:

vi xxx.sql

drop table tb;

create table tb(c1 int);

drop table tb1;

create table tb1(c1 int, c2 char(100));

cqd\_sqlfile:执行ddl前需要执行的cqd，如果不需要设置cqd，注释该字段或者该字段等于空即可；样例请参考config路径下的配置文件；

table\_name：测试表名

sql\_type：测试表列的sql类型，从第一列到最后一列依次填写，用逗号隔开；样例请参考config路径下的配置文件；

sql\_c\_type：测试时列对应的C语言类型，从第一列到最后一列依次填写，用逗号隔开；样例请参考config路径下的配置文件；

column\_name：测试表对应列的名称，从第一列到最后一列依次填写，用逗号隔开；样例请参考config路径下的配置文件；该字段主要用于SQLColumns或者SQLTables测试时，对比数据库返回类型比较；

column\_size：测试表列宽度定义，从第一列到最后一列依次填写，用逗号隔开；样例请参考config路径下的配置文件；CHAR、VARCHAR和lob等，配置为实际的宽度，浮点类型时配置为精度值，其他类型的配置为0即可；

decimal\_digits：测试表列类型是浮点类型时，配置为小数点宽度；

data\_size：测试加载数据时，CHAR、VARCHAR和LOB等最大的有效数据长度，长度必须小于等于列定义的最大宽度，浮点类型时配置为测试时有效的精度，该值必须小于等于列定义的精度值，其他类型配置为0即可；如果插入数据是空值时，赋值-1即可；

data\_digits：测试表的列是浮点型时，加载数据有效的小数点后的数据宽度；

lob\_table：测试表是否是包含lob列的表；

TRUE：是；

FALSE:否，默认为否；

lob\_columns：lob列的编号，从0开始，如第一列和第三列是lob类型，则配置如下：

lob\_columns=0,2

**DDL测试基本配置：**

stmt：测试的ddl，如：get tables

stmt\_ignore\_fail:

FALSE：返回失败时做一次失败计数统计；

TRUE：返回失败时不做失败统计处理；

stmt\_checkout：

FALSE：不做结果集处理；

TRUE：读结果集；

stmt\_result\_null：

FASLE：结果集是非空

TRUE：结果集是空

stmt\_times：

ddl执行次数

stmt\_interval：

循环多次执行ddl时，每次间隔时间，单位：秒

stmt\_cqd：

执行ddl前，设置的cqd

**rowset测试基本配置：**

rowset\_batch\_size：

rowset测试时的batch大小；

rowset\_batch\_num：

总的batch数；

rowset\_begin\_value：

rowset加载数据初始值，对应整型的列，加载的数据赋值从设置的begin value开始，CHAR、VARCHAR等字符串类型的，则是随机生成的数据；

rowset\_type：

加载方式：包括INSERT、UPSERT、UPSERT WITH NO CONFIG和UPSERT USING LOAD，可以同时配置多种加载方式，每种方式用逗号隔开，测试时顺序执行配置的加载方式；

rowset\_threads：

rowset测试并发的线程数，没有该字段，默认单线程；

rowset\_cqd：

做rowset前，需要设置的cqd；

rowset\_file：

如加载测试用的表存在lob列，并指定rowset\_file文件，则指定的文件用于lob列加载使用。

**select测试基本配置：**

select\_stmt：

执行select的语句，select语句可以是多条，每条sql语句用分号隔开；如：

select count(\*) from tb; select \* from tb;

select\_times:

重复执行select的次数；

select\_interval：

重复执行时，每次间隔时间；

select\_ignore\_fail：

TRUE：忽略产生的错误，不做测试失败计数处理；

FALSE：执行失败，做一次失败计数处理；

select\_checkout：

TRUE：读取结果集检查

FALSE：不做结果集读取

select\_result\_null：

TRUE:返回的结果集期望值是空值；

FALSE:返回结果集是非空值；

select\_save\_result：

TRUE：查询的结果集保存到文件，如做lob测试时，lob内容是图片，保存成图片后，用于检查图片是否可以正常打开；

FALSE:查询结果集不做保存处理；

select\_save\_file：

保存的文件名；

select\_save\_column\_id：

需要保存的列id，从0开始，如保存第一列和第二列数据，则select\_save\_column\_id=0,1；

多列时，id间隔用逗号隔开；

select\_cqd：

执行select操作前设置的cqd；

select\_calc\_crc：

对应CHAR、VARCHAR或者lob列加载时，随机产生的数据有做CRC校验，因此，做select查询是通过CRC校验判断插入的数据是否完整。当该字段配置为TRUE，则做CRC校验处理，否则不做CRC校验判断；

select\_col\_attr：

查询表数据时，是否调用SQLColAttribute返回列的属性。

select\_c\_type:

查询某列数据时，调用SQLGetData时，C语言的数据类型，如SQL\_C\_TCHAR、SQL\_C\_BINARY等等。

**testLoadDirect方式加载数据配置：**

load\_direct\_rows：

加载的行数；大于0的整数值即可；

load\_direct\_begin\_value：

加载数据初始值，对应整型的列，加载的数据赋值从设置的begin value开始，CHAR、VARCHAR等字符串类型的，则是随机生成的数据；

load\_direct\_insert：

加载方式：包括INSERT、UPSERT、UPSERT WITH NO CONFIG和UPSERT USING LOAD，可以同时配置多种加载方式，每种方式用逗号隔开，测试时顺序执行配置的加载方式；

load\_direct\_threads：

加载时并发的线程数；

load\_direct\_cqd：

加载数据前需要设置的CQD；

load\_direct\_file：

如加载测试用的表存在lob列，并指定文件名，则指定的文件用于lob列加载使用。

**testLoadAtExec方式加载数据配置：**

load\_at\_exec\_file：

如加载测试用的表存在lob列，并指定文件名，则指定的文件用于lob列加载使用。

load\_at\_exec\_rows：

加载的行数，大于0的整数值即可；

load\_at\_exec\_begin\_value：

加载数据初始值，对应整型的列，加载的数据赋值从设置的begin value开始，CHAR、VARCHAR等字符串类型的，则是随机生成的数据；

load\_at\_exec\_threads：

加载数据时的并发线程数；

load\_at\_exec\_insert：

加载方式：包括INSERT、UPSERT、UPSERT WITH NO CONFIG和UPSERT USING LOAD，可以同时配置多种加载方式，每种方式用逗号隔开，测试时顺序执行配置的加载方式；

load\_at\_exec\_cqd：

加载数据前需要设置的CQD；

load\_at\_exec\_type：

SQLPutData插数据有两种方式：SQL\_DATA\_AT\_EXEC和SQL\_LEN\_DATA\_AT\_EXEC，因此，测试时需要配置SQLPutData\_type；**当SQLPutData\_type= SQL\_LEN\_DATA\_AT\_EXEC时，字段data\_size对应的列有效数据长应该配置为你需要插入数据有效长度，特别是在插入文件时，data\_size应配置为文件的实际大小；**

load\_at\_exec\_put\_batch：

执行SQLPutData可以指定每次put数据大小；

**testLobUpdate方式插入数据：**

lobupdate\_rows：

插入的总行数；

lobupdate\_begin\_value：

加载数据初始值，对应整型的列，加载的数据赋值从设置的begin value开始，CHAR、VARCHAR等字符串类型的，则是随机生成的数据；

lobupdate\_insert：

加载方式：包括INSERT、UPSERT、UPSERT WITH NO CONFIG和UPSERT USING LOAD，可以同时配置多种加载方式，每种方式用逗号隔开，测试时顺序执行配置的加载方式；

lobupdate\_cqd：

加载数据前需要设置的CQD；

lobupdate\_threads：

加载数据时的并发线程数；

# 五、demo

以下是列举的两个简单例子的配置，更多配置方法请参考config路径下配置信息；

## 5.1、ddl文件测试

配置文件config.ini配置如下:

[config]

CATALOG =TRAFODION

DSN =ODBC

user =trafodion

password =traf123

schema =QA\_ODBC

isMultiTask =FALSE

test\_case = testSqlFile

[table]

sql\_file =blob.sql

sqlfile\_cqd =cqd traf\_blob\_as\_varchar 'off'

table\_name =TB\_BLOB

blob.sql文件内容如下：

DROP TABLE TB\_CLIP\_VARCHAR;

CREATE TABLE TB\_CLIP\_VARCHAR

(

B\_LOB BLOB

,ID INT NOT NULL

,DATABASENAME VARCHAR(255)

,INSTANCENAME VARCHAR(255)

,USERNAME VARCHAR(255)

,CREATETIME TIMESTAMP

,FDBVERSIONMAJOR INT

,FDBVERSIONMINOR INT

,FDBVERSIONBUGFIX INT

,FDEPROVIDERNAME VARCHAR(255)

,FDEDESCRIPTION VARCHAR(104)

,UID VARCHAR(64)

);

## 5.2、SQLColumns的测试

[config]

CATALOG =TRAFODION

DSN =ODBC

user =trafodion

password =traf123

schema =QA\_ODBC

isMultiTask =FALSE

test\_case = SQLColumns

[table]

table\_name =TB\_TEST\_1

sql\_type = SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_NUMERIC,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_TYPE\_TIMESTAMP,SQL\_TYPE\_TIMESTAMP,SQL\_TYPE\_TIMESTAMP,SQL\_VARCHAR,SQL\_NUMERIC,SQL\_VARCHAR,

SQL\_NUMERIC,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_NUMERIC,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_NUMERIC,SQL\_NUMERIC,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_VARCHAR,SQL\_TYPE\_TIMESTAMP

sql\_c\_type =SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_NUMERIC,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_TIMESTAMP,SQL\_C\_TIMESTAMP,SQL\_C\_TIMESTAMP,SQL\_C\_CHAR,SQL\_C\_NUMERIC,SQL\_C\_CHAR,SQL\_C\_NUMERIC,SQ

L\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_NUMERIC,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_NUMERIC,SQL\_C\_NUMERIC,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_CHAR,SQL\_C\_TIMESTAMP

column\_size =40,40,40,40,40,4,64,50,64,64,0,0,0,40,4,20,4,100,100,4,64,64,4,3,40,40,40,0

decimal\_digits =0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

data\_size =40,40,40,40,40,4,64,50,64,64,0,0,0,40,4,20,4,100,100,4,64,64,4,3,40,40,40,0

data\_digits =0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

column\_name =TRANS\_TRACK\_ID,NODE\_INS\_ID,ACTIVE\_INS\_ID,NODE\_TRACK\_ID,WORK\_ID,TRANS\_FLAG,ACTOR\_ID,ACTOR\_NAME,AUTHORIZER\_ID,AUTHORIZER\_NAME,CREATE\_TIME,HANDLE\_TIME,OVER\_TIME,MSG,MSG\_TYPE,RESULT\_FLAG,BACK\_FLAG,CONDITION,CONDITION\_VA

LUE,CONDITION\_RESULT,RESSIGNM\_PERSON\_ID,RESSIGNM\_PERSON\_NAME,COMPLETE\_TYPE,REVOKE\_FLAG,PRE\_TRANS\_TRACK\_ID,PROCESS\_ID,MAIN\_PROCESS\_INS\_ID,MODIFY\_DATE

其中表TB\_TEST\_1的DDL如下：

CREATE TABLE TB\_TEST\_1

(

TRANS\_TRACK\_ID VARCHAR(40 )

, NODE\_INS\_ID VARCHAR(40 )

, ACTIVE\_INS\_ID VARCHAR(40 )

, NODE\_TRACK\_ID VARCHAR(40)

, WORK\_ID VARCHAR(40 )

, TRANS\_FLAG NUMERIC(4, 0)

, ACTOR\_ID VARCHAR(64 )

, ACTOR\_NAME VARCHAR(50 )

, AUTHORIZER\_ID VARCHAR(64 )

, AUTHORIZER\_NAME VARCHAR(64 )

, CREATE\_TIME TIMESTAMP

, HANDLE\_TIME TIMESTAMP

, OVER\_TIME TIMESTAMP

, MSG VARCHAR(40 )

, MSG\_TYPE NUMERIC(4, 0)

, RESULT\_FLAG VARCHAR(20 )

, BACK\_FLAG NUMERIC(4, 0)

, CONDITION VARCHAR(100 )

, CONDITION\_VALUE VARCHAR(100 )

, CONDITION\_RESULT NUMERIC(4, 0)

, RESSIGNM\_PERSON\_ID VARCHAR(64 )

, RESSIGNM\_PERSON\_NAME VARCHAR(64 )

, COMPLETE\_TYPE NUMERIC(4, 0)

, REVOKE\_FLAG NUMERIC(3, 0)

, PRE\_TRANS\_TRACK\_ID VARCHAR(40 )

, PROCESS\_ID VARCHAR(40 )

, MAIN\_PROCESS\_INS\_ID VARCHAR(40 )

, MODIFY\_DATE TIMESTAMP

);